BUILDING WORKFLOW DIAGRAMS WITH SVG, HTML5, CSS3 AND JAVASCRIPT

Dhori TERPO[1], Endri XHINA[2], Roland VASILI[3]

[1]University "E. Çabej", Faculty of Natural Sciences, Department of Mathematics & Informatics, Gjirokastra, Albania, dhterpo@uogj.edu.al
[2] University of Tirana, Faculty of Natural Sciences, Department of Informatics, Tirana, Albania, endri.xhina@fshn.edu.al
[3]University "E. Çabej", Faculty of Natural Sciences, Department of Mathematics & Informatics, Gjirokastra, Albania, rvasili@uogj.edu.al

Abstract

According to Workflow Management Coalition (WfMC) , workflow is a term used to describe the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action (activities), based to a set of procedural rules. To a software developer, the word workflow, typically conjures up images of a highly graphical environment where complex business rules are declared visually rather than entirely in code. Individual tasks are organized into an appropriate sequence, and branching and looping decision are declared to control the flow of execution between tasks. In this paper we will present the usage of SVG, HTML5, CSS3 and JavaScript for the development of a GUI web interface that allow us to easily design and model the tasks of a workflow process. A comparison of the features that the above technologies provide for symbolizing workflow statuses and connector steps will be presented in an example driven manner.

Keywords: *Workflow, Html5, Css3, JavaScript, SVG*

1. The need for web graphic interface for the building of a workflow.

The Workflow Management Coalition (WfMC, 1999) defines workflow as the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Business processes in the today's business world are constantly changing, unpredictable, volatile, and seems to become more complex every day. Building workflow from internet gives us the possibility to quickly adapt to new requirements by changing or improving the steps and activities of business process, irrespective of geographic location, away from office, in time, without cost of installation of any program and just by simply having access to internet.

2. Available Tools.

SVG and HTML5 Canvas are both web technologies that allow you to create rich graphics inside the browser, but they are fundamentally different.

SVG is used to describe Scalable Vector Graphics, a retained mode graphics model that persist in an in-memory model. According to W3C (2011), SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and dynamic. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting.

HTML5 Canvas is an immediate mode bitmapped area of the screen that can be manipulated with JavaScript (Fulton & Fulton, 2013). Immediate mode refers to the way the canvas renders pixels on the screen. HTML5 Canvas completely redraws the bitmapped screen on every frame by using Canvas API calls from JavaScript. As a programmer, your job is to set up the screen display before each frame is rendered so that the correct pixels will be shown.
The basic HTML5 Canvas API includes a 2D context that allows a programmer to draw various shapes, render text, and display images directly onto a defined area of the browser window. You can apply colours, rotations, gradient fills, alpha transparencies, pixel manipulations, and various types of lines, curves, boxes, and fills to augment the shapes, text, and images you place onto the canvas.

3. SVG Web Graphic.

3.1 Development history.

SVG was developed by the W3C SVG Working Group starting in 1998, after Macromedia and Microsoft introduced Vector Markup Language (VML) whereas Adobe Systems and Sun Microsystems submitted a competing format known as PGML. The working group was chaired by Chris Lilley of the W3C. SVG 1.0 became a W3C Recommendation on 2001-09-04. SVG 1.1 and SVG mobile Profiles (SVG Basic and SVG Tiny) were approved as W3C recommendations in January 2003. SVG 1.2 is the specification currently being developed and is available in draft form. It aims at introducing new features which include the addition of native state-of the-art text wrapping and flowing, SMIL integration enhancements (including audio and video), and miscellaneous DOM enhancements etc. Today, the SVG 1.1 specification forms the core of the current SVG developments and many implementations are already available.

3.2 Why SVG?

There are a number of reasons why SVG is ideally suited to the web platform:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable (and the image can be zoomed without degradation)
- SVG is an open standard
- SVG files are pure XML

3.3 SVG Language.

3.3.1 SVG Structure.

In every SVG file, the content is enclosed with the svg element’s tags: <svg>…</svg>. All SVG content must appear between these two tags. The only data that should appear outside of these two tags will be XML data defining the document type and nature.
Thus, one of the simplest SVG documents that you can create is shown in listing 3.1.

Listing 3.1 SVG structure

```
<?xml version="1.0"?>
 <svg xmlns="http://www.w3.org/2000/svg">
        ......
</svg>
```

In order to use an SVG Graphic within HTML5, you can:

- copy/paste SVG code within HTML code (inlining)
- use the HTML *img* tag
- use HTML *object* tag
- use the HTML *iframe* tag
- use CSS (background images)
- including SVG within SVG using the *image* tag.

3.3.2 Basic SVG shapes elements.

The basic SVG shapes elements (Watt & Lilley, 2002) that describe commonly used graphic shapes are:

Line

Draws a line from the starting point (start-x/start-y) to the end point (end-x/end-y).

<line x1="start-x" y1="start-y" x2="end-x" y2="end-y" />

The listing 3.2 shows the code for drawing a black line.

Listing 3.2 Drawing a line

```
<svg width="300" height="300" >
  <line x1="100" y1="100" x2="200" y2="100" style="stroke: black;" />
</svg>
```

Rectangle

Draws a rectangle/square with an upper left corner at (left-x/top-y) and the given width and height.

<rect x="left-x" y="top-y" width="width" height="height" />

The listing 3.3 shows the code for drawing a black rectangle.

Listing 3.3 Drawing a rectangle

```
<svg width="300" height="300" >
  <rect x="50" y="50" width="50" height="50" />
</svg>
```

Circle

Draws a perfect circle with the given radius centered at (center-x/center-y)

<circle cx="center-x" cy="center-y" r="radius" />

The listing 3.4 shows the code for drawing a circle.

Listing 3.4 Drawing a circle

```
<svg width="300" height="300" >
  <circle cx="100" cy="50" r="50" style="stroke: black; fill: none;" />
</svg>
```

Polyline

Draws a series of connected lines described by x/y point pairs in points.

<polyline points="points" />

The listing 3.5 shows the code for drawing a resistor symbol.

Listing 3.5 Drawing a polyline

```
<svg width="300" height="300" >
  <polyline points="5 20, 20 20, 25 10, 35 30, 45 10, 55 30, 65 10, 75 30, 80 20, 95 20"
style="stroke: black; stroke-width: 2; fill: none;" />
</svg>
```

Polygon

Draws an arbitrary closed polygon outlined by x/y point pairs in points.

<polygon points="points" />

The listing 3.6 shows the code for drawing a parallelogram.

Listing 3.6 Drawing a polygon

```
<polygon points="30,70 60,70 45,100 15,100"
style="fill: red; stroke: black;"/>
```

Path

A path is defined by including a '*path*' element which contains a *d="(path data)"* attribute, where the 'd' attribute contains the *moveto*, *line*, *curve* (both cubic and quadratic Béziers) , *arc* and *closepath* instructions.

The listing 3.7 shows the code for drawing a black triangle.

Listing 3.7 Drawing a path

```
<svg width="300" height="300" >
  <path d="M150 0 L75 100 L225 100 Z" />
</svg>
```

Text

The <*text*> element allows positioning and graphic styling of the text.

Listing 3.8 shows the code for drawing a simple text.

Listing 3.8 Drawing a simple text

```
<svg width="300" height="300" >
  <text x="50" y="50">Simplest Text</text>
</svg>
```

3.3.3 SVG Interactivity.

SVG drawings can be interactive and dynamic. The SVG has its own SVG DOM. The SVG DOM is a language-neutral Application Programming Interface (API), which gives any programming language access to the parts of the SVG document. Every SVG element and its attributes can be accessed and manipulated using scripts. The most commonly used scripting language is JavaScript. Using JavaScript we can create elements on the fly, change elements, remove elements, and re-order the DOM –graph. The most useful methods for modifying an SVG document are:

- *getElementById*
- *getStyl*
- *setProperty*
- *setAttribute*
- *getAttribute*
- *cloneNode*

We can write a script in JavaScript to interact with an SVG graphic. Interaction occurs when graphic objects respond to events. The basics of using a scripting language in SVG require using the <script> element to enclose the script and setting the type attribute of the <script> element.

Script consists of functions that are called by events that are triggered by elements in SVG. Objects can respond to mouse events associated with clicking the mouse button: *click*, *mousedown*, and *mouseup*; events associated with moving the mouse: *mouseover, mouseout*, and *mousemove*; events associated with an object's status: *load* (the object has been fully parsed and is ready to render); and the non-standardized events associated with pressing keys*: keydown* and *keyup*.

To allow an object to respond to an event, we add an *oneventName* attribute to the element in question. The value of the attribute will be a JavaScript statement, usually a function call. This function usually takes the reserved word *evt* as one of its parameters. The *evt* has properties and methods that describe the event that has occurred.
The listing 3.9 demonstrates the use of the SVG script tag. In this code, we use JavaScript to change the radius of the SVG <circle> element.

Listing 3.9 Changing the radius of a circle

```
<svg width="100" height="100" viewBox="0 0 100 100"
   xmlns="http://www.w3.org/2000/svg">
 <script type="text/javascript">
  // <![CDATA[
  function change(evt) {
    var target = evt.target;
    var radius = target.getAttribute("r");
    if (radius == 15) {
      radius = 45;
    } else {
      radius = 15;
    }
    target.setAttribute("r",radius);
  }
 // ]]>
 </script>
 <circle cx="50" cy="50" r="45" fill="green"
       onclick="change(evt)" />
</svg>
```

4. Using SVG to draw workflows diagrams.

Drawing workflows diagrams requires certain features that are well implemented on SVG such as:

- Dynamically generated SVG diagrams based on server side data such as databases or other files.
- Drawing complex paths.
- Animating graphic objects within the diagram.
- Changing the diagram as a response to browser events.

4.1 Dynamically generated diagrams with SVG.

The listing 4.1 represents a php script that generates an SVG graph which includes some of the basic elements for drawing a diagram: a circle, a polyline, a polygon and text (see figure 4.1).

Listing 4.1 Dynamically generated diagrams with SVG

```
<?php
header("Content-type: image/svg+xml");
echo '<?xml version="1.0" standalone="no"?>';
echo '<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">';
echo
'<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
9.width="300px" height="300px">';
10.echo "
11.<title>Small SVG example</title>
12.<circle id='rrethi' cx='120' cy='150' r='60' fill='#F0F0F0'>
13.<animate attributeName='r' from='2' to='80' begin='0'
14.dur='3' repeatCount='indefinite' /></circle>
15.<rect x='19' y='37' fill='#52B848' stroke='#000000' stroke-miterlimit='10' width='80'
height='59' id='greenBlock' />
16.<polyline points='120 30, 25 150, 290 150'  id='vije'
17.stroke-width='4' stroke='brown' fill='none' />
18.<polygon points='210 100, 210 200, 270 150'   id='pol'
19.fill='green' />
20.<text id='mesazh' x='60' y='250' fill='blue'>Hello, World!</text>
21.</svg> ";
22.?>
```

In line 2 an information is sent to the browser identifying the content-type of the data as image/svg using the php function *header ("Content-type: image/svg+xml");.*
Afterwards the output is sent to the browser through *echo* commands which allows to send to the browser data which could be selected by a database.
The *animate* command is used at the circle svg element to change its radius size from 2 to 80 pixels with a delay between iterations of 3 milliseconds (*dur=3*). The animation starts immediately on page load *(begin='0')* and repeats continuously *(repeatCount='indefinite').*

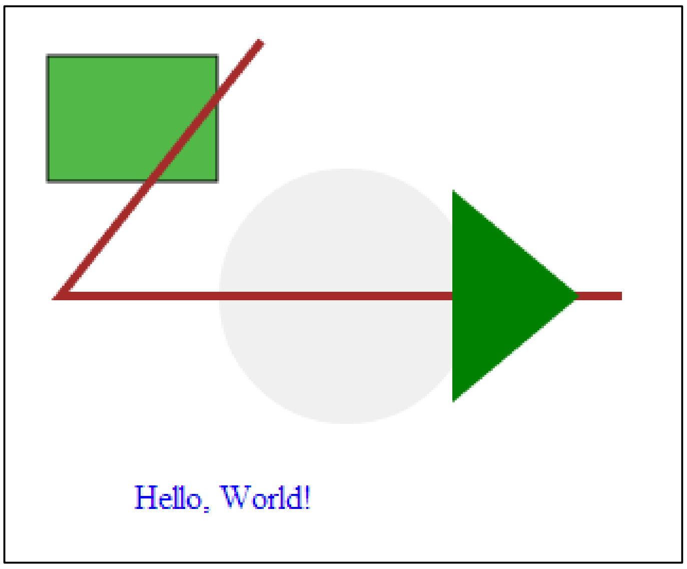


Figure 4.1 Dynamically generated diagrams with SVG

4.2 Drawing complex paths in SVG.

SVG is reach with elements to draw complex lines. This functionality is offered by the following elements:

- Polyline
- Path
- Beziers Curve

4.3 Animating objects within the diagram.

Graphic Animation is the process of changing the attribute(s) of a graphic object. In SVG it is very easy to implement animation. We have given a first example of animating the radius of a circle. More complex animations, especially concurrently animating several attributes or animating the form of lines, which are the connecting backbone of the workflow diagram are also possible in SVG.

From the listing 4.1 changing the code in line 15 with the code below:

```
<rect x='19' y='37' fill='#52B848' stroke='#000000' stroke-miterlimit='10' width='80'
height='59' id='greenBlock' >
<animateTransform
          attributeName="transform"
          begin="0s"
          dur="20s"
          type="rotate"
          from="0 60 60"
          to="360 60 60"
          repeatCount="indefinite"
      />
</rect>
```

and the code in lines 12,13,14 with the code below:

```
<circle id='rrethi' cx='120' cy='150' r='60' fill='#F0F0F0'>
  <animate attributeName='cx' from='120' to='300' begin='0'
 dur='10' repeatCount='indefinite' />
  <animate attributeName='cy' from='150' to='-20' begin='0'
 dur='10' repeatCount='indefinite' />
 </circle>
```

will animate simultaneously the position of the circle and will rotate the rectangle (see figure 4.2).

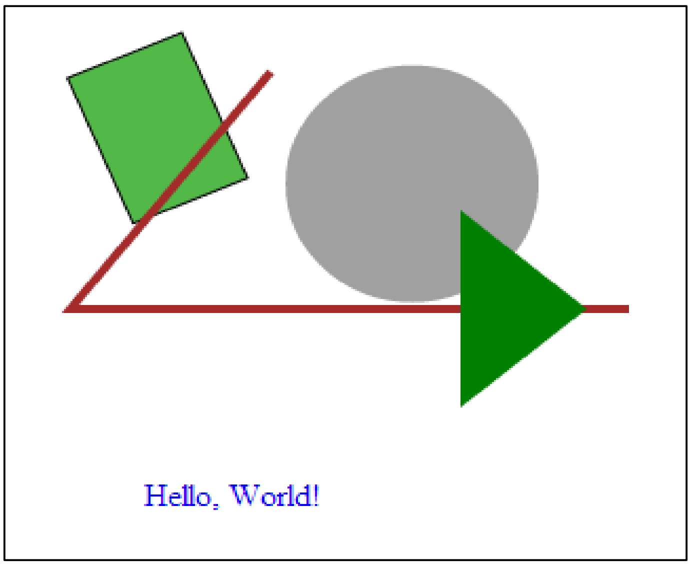


Figure 4.2 SVG simultaneously animation

4.4 Changing the SVG diagram on browser events.

The example below (see figure 4.3) will allow the user to choose through an *onChange* browser event of an input of type color to assign the color of an element in the diagram.
The JavaScript function in listing 4.2 uses getSVGDocument() method to create an SVG document object. This SVG document object can be used in a similar manner like the HTML DOM to access the objects within SVG document or add or remove objects in the SVG document. The *getElementById("pol");* is used to get a pointer on the *poligon* object and then uses the *setAttribute* method of the SVG object to set the *fill* atribute to the color which is taken as an argument of the function.

Listing 4.2 Changing the color of svg diagram elements

```
function poligon(choosen_color)
{
var svgEmbed = document.querySelector("#svgembed");
var svg = svgEmbed.getSVGDocument();
var p = svg.getElementById("pol");
p.setAttribute("fill", choosen_color);
}
```

The listing 4.3 shows an input of type color which is new to HTML5 to allow the user to choose the color for the polygon.

Listing 4.3 Selecting the color of svg diagram elements

```
Color the  poligon:<input type="color" name="poligon_color"
onChange="poligon(this.value);">
```

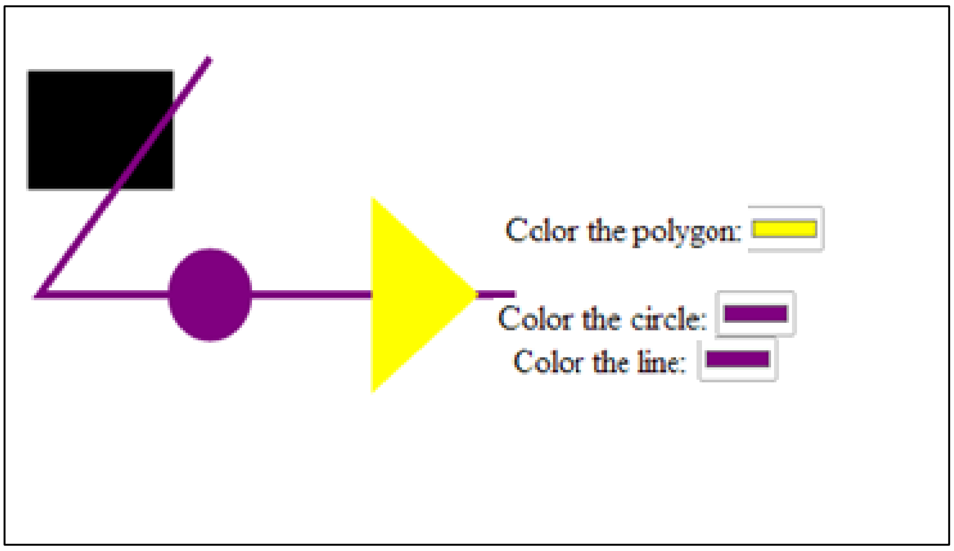
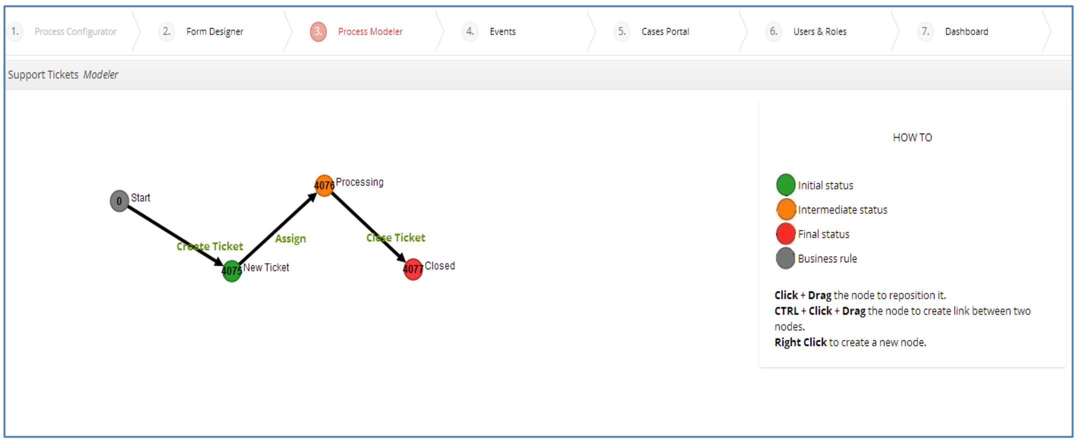


Figure 4.3 Changing the color of svg diagram elements

The above tools and techniques are used to develop a Process Modeler application module, for a commercial business process automation tool, which is called Smart Processes.[1] The figure 4.4 provides a picture of this application.

[1] A Demo version of Smart Processes which included Process Modeler can be found at http://www.ictsofts.com



Click + **Drag** the node to reposition it.
CTRL + **Click** + **Drag** the node to create link between two nodes.
Right Click to create a new node.

Figure 4.4 Process Modeler

5. Conclusions.

Tools for building standard web based workflow diagrams are available without the need of any additional browser plugins. With HTML5 there are two elements for graphic designing Canvas and SVG. SVG is a standard language recommended by W3C for 2D graphics on the web. Native SVG support from modern browsers made it possible to develop dynamic and interactive graphics on the web with a seamless integration of server side data dynamically. SVG contains the features that are needed to develop a web based application to design workflow diagrams. In an example driven approach this article presented the basic graphical elements for drawing diagrams. Dynamic content, animation and user interaction is also explained in this article through examples. The above explained tools are used to develop a Process Modeler application module for commercial business process automation.

6. References.

Fulton, S., & Fulton, J. (2013). *HTML5 Canvas*.(2nd ed.). O'Reilly Media.

W3C.(2011, August 16). Scalable Vector Graphics (SVG) 1.1 (Second Edition). Retrieved from http://www.w3.org/TR/SVG/.

Watt, A., & Lilley, Ch. (2002). *Svg Unleashed*. Sams.

WfMC.(1999).Terminology and Glossary, Document No WFMC-TC-1011. Retrieved from http://www.wfmc.org/docs/TC-1011_term_glossary_v3.pdf.