

SVG, sXBL, Ontologically Animated Fuzzy-Sets, and deBono / HML Diagrams

David Dodds Open-Meta Computing Inc open-meta.com

SVG is a well-known xml namespace which provides interactive, dynamic vector graphics both in the web-environment and on the desktop. Of the many values of SVG one of them is that SVG has a built-in meta-data capability. Since SVG is (xml) text it can be read by other programs.

Next we see an example of SVG (code) which includes (embedded) meta-data. The picture is in colour but the printing (of it) is in black and white.



```
<?xml version="1.0" standalone="yes" ?>
<svg xmlns = 'http://www.w3.org/2000/svg'>
<metadata xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/TR/. ..-schema#"
xmlns:daxsvg="http://www.open-meta.com/daxschema/" >
<rdf:Description about="#text1">
<daxsvg:Below resource="#xbaseline"/>
</rdf:Description>
<rdf:Description about="#text1">
<daxsvg:Near resource="#xbaseline" />
</rdf:Description>
<rdf:Description about="#text2">
<daxsvg:Below resource="#text1"/>
```

```
</rdf:Description>
<rdf:Description about="#text2">
<daxsvg:Near resource="#text1" />
</rdf:Description>
<rdf:Description about="#endlineleft">
<daxsvg:AtRight resource="#line1"/>
</rdf:Description>
<rdf:Description about="#endlineleft">
<daxsvg:Near resource="#line1" />
</rdf:Description>
<rdf:Description about="#endlineright">
<daxsvg:AtLeft resource="#bar13"/>
</rdf:Description>
<rdf:Description about="#endlineright">
<daxsvg:Near resource="#bar13" />
</rdf:Description>
<rdf:Description about="#line1">
<daxsvg:AtRight resource="#line2" />
</rdf:Description>
<rdf:Description about="#line2">
<daxsvg:AtRight resource="#line3" />
</rdf:Description>
<rdf:Description about="#line3">
<daxsvg:AtRight resource="#line4" />
</rdf:Description>
<rdf:Description about="#line4">
<daxsvg:AtRight resource="#line5" />
</rdf:Description>
<rdf:Description about="#line5">
<daxsvg:AtRight resource="#line6" />
</rdf:Description>
<rdf:Description about="#line6">
<daxsvg:AtRight resource="#line7" />
</rdf:Description>
<rdf:Description about="#line7">
<daxsvg:AtRight resource="#line8" />
</rdf:Description>
```

```
<rdf:Description about="#line8">
<daxsvg:AtRight resource="#line9" />
</rdf:Description>
<rdf:Description about="#line9">
<daxsvg:AtRight resource="#line10" />
</rdf:Description>
<rdf:Description about="#line10">
<daxsvg:AtRight resource="#line11" />
</rdf:Description>
<rdf:Description about="#line11">
<daxsvg:AtRight resource="#line12" />
</rdf:Description>
</metadata>
<rect id="lineval18" x="37" y="190" width="280" height="1"
style="stroke:black;
stroke-width:1" />
<text id="text3" x="317" y="194"
style="font-family:Verdana; font-size:12.333; fill:indigo">
18
</text>
<rect id="xbaseline" x="37" y="200" width="329" height="1"
style="stroke:blue;
stroke-width:1" />
<rect id="endlineright" x="333" y="96" width="1" height="104"
style="stroke:black;
stroke-width:1" />
<rect id="endlineleft" x="37" y="96" width="1" height="104"
style="stroke:black;
stroke-width:1" />
<rect id="line1" x="40" y="160" width="20" height="40"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line2" x="60" y="140" width="20" height="60"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<rect id="line3" x="80" y="111" width="20" height="89" style="stroke:red;
fill:red;
```

```
stroke-width:0" />
<rect id="line4" x="100" y="130" width="20" height="70"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<rect id="line5" x="120" y="173" width="20" height="27"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line6" x="140" y="191" width="20" height="09"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line7" x="160" y="140" width="20" height="60"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<rect id="line8" x="180" y="167" width="20" height="33"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line9" x="200" y="175" width="20" height="25"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line10" x="220" y="129" width="20" height="71"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<rect id="line11" x="240" y="150" width="20" height="50"
style="stroke:green;
fill:green; stroke-width:0" />
<rect id="line12" x="260" y="139" width="20" height="61"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<rect id="line13" x="280" y="125" width="20" height="75"
style="stroke:yellow;
fill:yellow; stroke-width:0" />
<text id="text1" x="37" y="210"
style="font-family:Verdana; font-size:12.333; fill:black">
87 88 89 90 91 92 93 94 95 96 97 98 99
</text>
<text id="text2" x="37" y="230"
style="font-family:Verdana; font-size:12.333; fill:brown">
```

Mean High Ratings August 1999

</text>

</svg>

Some examples of XBL coding follow:

<http://www.croczilla.com/svg/samples/>

example XBL_1

```
<?xml version="1.0"?>
<?xml-stylesheet href="xbl1-bindings.css"
type="text/css"?>
<doc>
  <svg xmlns="http://www.w3.org/2000/svg"
xmlns:svg="http://www.w3.org/2000/svg"
id="canvas">
    <triangle transform="translate(100,100)"
style="fill: blue;"/>
    <triangle transform="translate(300,100)"
style="fill: red;"/>
    <triangle transform="translate(100,300)"
style="fill: green;"/>
  </svg>
</doc>
```

xbl1-bindings.css

```
triangle
{
    -moz-binding: url("xbl1-
bindings.xml#triangle");

    stroke-width: 0.5;
    stroke: black;
```

```

        fill-opacity: 0.6;
    }

triangle > *:hover
{
    stroke-width: 2;
}

```

xbl1-bindings.xml

```

<?xml version="1.0"?>
<?xml-stylesheet href="xbl1-bindings.css"
type="text/css"?>

<bindings xmlns="http://www.mozilla.org/xbl"
           xmlns:xbl="http://www.mozilla.org/xbl"
           xmlns:svg="http://www.w3.org/2000/svg">
  <binding id="triangle" extends="svg:generic">
    <content>
      <svg:polygon xbl:inherits="style, transform"
points="0,0 300,50 100,200" class="area"/>
    </content>
    <implementation>
      <constructor>
        <![CDATA[
          dump("constructed triangle!\n");
        ]]>
      </constructor>
      <destructor>
        <![CDATA[
          dump("killed triangle!\n");
        ]]>
      </destructor>
      <property name="dX"> 0 </property>
      <property name="dY"> 0 </property>
      <method name="_mm">
        <parameter name="evt"/>
      <body>

```

```

        <![CDATA[
            document.draggedItem.inner.transform.ba
seVal.getItem(0).setTranslate(
            evt.clientX - document.draggedItem.dX,
            evt.clientY - document.draggedItem.dY);
        ]]>
    </body>
</method>
<method name="_mu">
    <parameter name="evt"/>
    <body>
        <![CDATA[
            document.removeEventListener
("mousemove", document.draggedItem._mm,true);
            document.removeEventListener("mouseup",
document.draggedItem._mu,true);
            document.draggedItem = null;
        ]]>
    </body>
</method>
<property name="inner" readonly="true">
    <getter>
        <![CDATA[
            return document.getAnonymousNodes(this)
[0];
        ]]>
    </getter>
</property>
</implementation>
<handlers>
    <handler event="mousedown">
        <![CDATA[
            // make sure we're at the top of the z-
order:
            if (this != this.parentNode.lastChild) {
                // save our internal state back onto
the bound element, so that
                // it will get restored correctly when
we move this node

```

```

        this.setAttribute("transform",
this.inner.getAttribute("transform"));
        // suspending redraw stops the flicker
during removing & appending the node:
        this.inner.ownerSVGElement.suspendRedra
w(1000);

        // now move us to the top of the z-
order
        // this will remove & reappend us:
this.parentNode.appendChild(this);

        this.inner.ownerSVGElement.unsuspendRed
raw(1);
        // XXX. For some reason the style
doesn't get resolved correctly.
        // The border should be thick but it
isn't until you move the mouse...
    }
    document.draggedItem = this;
    this.dX = event.clientX -
this.inner.transform.baseVal.getItem(0).matrix.e;
    this.dY = event.clientY -
this.inner.transform.baseVal.getItem(0).matrix.f;
    document.addEventListener("mousemove",
this._mm, true);
    document.addEventListener("mouseup",
this._mu, true);
    ]]>
    </handler>
    </handlers>
    </binding>
</bindings>

```

XUL and SVG 1

```

<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin"

```

```

type="text/css"?>
<window
xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul"
    xmlns:svg="http://www.w3.org/2000/svg">
    <svg:svg datasources="xulsvg1.rdf"
ref="urn:root">
    <template>
    <rule>
    <conditions>
    <content uri="?root"/>
    <triple subject="?root"
    predicate="urn:croczilla:xulsvg1:
shapes"
    object="?shapes"/>
    <member container="?shapes"
child="?shape"/>
    <triple subject="?shape"
    predicate="urn:croczilla:xulsvg1:
xform"
    object="?xform"/>
    <triple subject="?shape"
    predicate="urn:croczilla:xulsvg1:
fill"
    object="?fill"/>
    </conditions>
    <action>
    <svg:polygon uri="?shape"
    fill="?fill"
    transform="?xform"
    fill-opacity=".5"
    points="100,100 200,100
150,200"/>
    </action>
    </rule>
    </template>
    </svg:svg>
</window>

```

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
        xmlns:s="urn:croczilla:xulsvg1:">
  <rdf:Description about="urn:root">
    <s:shapes>
      <rdf:Bag>
        <rdf:li>
          <rdf:Description s:xform="translate(100) "
s:fill="red" />
        </rdf:li>
        <rdf:li>
          <rdf:Description s:xform="translate(200) "
s:fill="green" />
        </rdf:li>
        <rdf:li>
          <rdf:Description s:xform="translate(300) "
s:fill="blue" />
        </rdf:li>
        <rdf:li>
          <rdf:Description s:xform="rotate(10) "
s:fill="red" />
        </rdf:li>
        <rdf:li>
          <rdf:Description s:xform="rotate(20) "
s:fill="green" />
        </rdf:li>
        <rdf:li>
          <rdf:Description s:xform="rotate(30) "
s:fill="blue" />
        </rdf:li>
      </rdf:Bag>
    </s:shapes>
  </rdf:Description>
</rdf:RDF>
```

SVG means of capturing metaphorical and analogical knowledge

This is an SVG animation which the author wrote to demonstrate an XML (SVG) means of capturing metaphorical and analogical knowledge and providing visual / diagrams to represent such. The complete code for this SVG animation is at open-meta.com, called `anim01.svg`.

```
<?xml-stylesheet type="text/xsl" href="anim01.xsl"?>
<svg width="16cm" height="14cm" viewBox="0 0 255 201">
<metadata xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/TR/. ..-schema#"
xmlns:daxsvg="http://www.open-meta.com/daxschema/" >
<rdf:Description about="#arrowstreamer">
<daxsvg:Touches resource="#obstructor"/>
</rdf:Description>
<rdf:Description about="#arrowstreamer">
<daxsvg:Static resource="#obstructor" />
</rdf:Description>
<rdf:Description about="#particlestreamer">
<daxsvg:Touches resource="#obstructor"/>
</rdf:Description>
<rdf:Description about="#particlestreamer">
<daxsvg:Static resource="#obstructor" />
</rdf:Description>
</metadata>
<desc>Copyright 2001 - 2005 David Dodds Example anim01a - demonstrate
deBono
diagram SVG animation with Lakoff spatial metaphor</desc>
<rect x="1" y="1" width="253" height="199"
fill="black" stroke="blue" stroke-width="7" />
<text id="uplabel" x="230" y="20"
style="font-family:Verdana; font-size:12.333; fill:blue">
UP
</text>
<text id="downlabel" x="200" y="180"
style="font-family:Verdana; font-size:12.333; fill:blue">
DOWN
```

```

</text>
<text id="goallabel" x="110" y="42"
style="font-family:Verdana; font-size:12.333; fill:blue">
GOAL
</text>
<g id="leftfunnel" side">
<path d="M 99 180 L 99 57"
style="fill:none; stroke:green; stroke-width:10"/>
</g>
<g id="rightfunnel" side">
<path d="M 153 57 L 153 180 "
style="fill:none; stroke:green; stroke-width:10"/>
</g>
<text id="Hide" x="15" y="190"
style="font-family:Verdana; fill:orange; font-size:4; " >
HIDE Scrolling Text
</text>
<text id="Explain" x="15" y="180"
style="font-family:Verdana; fill:orange; font-size:4;"
>
Explanation of animation
</text>
<rect id="arrowstreamer" x="110" width="3" height="20" >
<animate attributeName="y" attributeType="XML"
begin="0s" dur="5s" fill="freeze" from="180" to="55" />
<animate attributeName="height" attributeType="XML"
begin="0s" dur="5s" fill="freeze" from="20" to="143" />
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(0,0,255)" to="rgb(110,0,0)"
begin="0s" dur="5s" fill="freeze" />
</rect>
<rect id="particlestreamer" x="120" width="3" height="3" >
<animate attributeName="y" attributeType="XML"
begin="2s" dur="5s" fill="freeze" from="170" to="55" />
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(0,0,255)" to="rgb(110,0,0)"
begin="2s" dur="5s" fill="freeze" />

```

```

</rect>
<rect id="misdirectedbehaviourstreamer" x="147" y="190" width="5"
height="5"
>
<animate attributeName="y" attributeType="XML"
begin="3s" dur="5s" fill="freeze" from="190" to="130" />
<animate attributeName="x" attributeType="XML"
begin="3s" dur="5s" fill="freeze" from="147" to="230" />
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(0,0,255)" to="rgb(128,0,0)"
begin="3s" dur="5s" fill="freeze" />
</rect>
<rect id="obstructor" class="hitBox" x="40" y="49" width="50" height="5"
>
<animate attributeName="x" attributeType="XML"
begin="3s" dur="5s" fill="freeze" from="40" to="100" />
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(10,0,0)" to="rgb(255,0,0)"
begin="3s" dur="3s" fill="freeze" />
</rect>
<text id="obstructorlabel" x="120" y="53"
style="font-family:Verdana; font-size:4; fill:black">
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(0,0,0)" to="rgb(255,255,255)"
begin="7s" dur="2s" fill="freeze" />
obstructor
</text>
<text id="misdirectedbehaviourlabel" x="200" y="125"
style="font-family:Verdana; font-size:4; fill:black">
<animateColor attributeName="fill" attributeType="CSS"
from="rgb(0,0,0)" to="rgb(255,255,255)"
begin="9s" dur="3s" fill="freeze" />
misdirectedbehaviour
</text>
</svg>

```

SVG can have embedded meta-data, it can also be parsed by an XML parser,

XSLT can treat SVG via its “patterns” and transform it into something else XML. SVG can be used “metaphorically” or analogically to represent the process of cutting, such as cutting wire-link fence. That can in turn be rasterized so that a program can “know what to look for” in a tv image of cut material, such as fence.

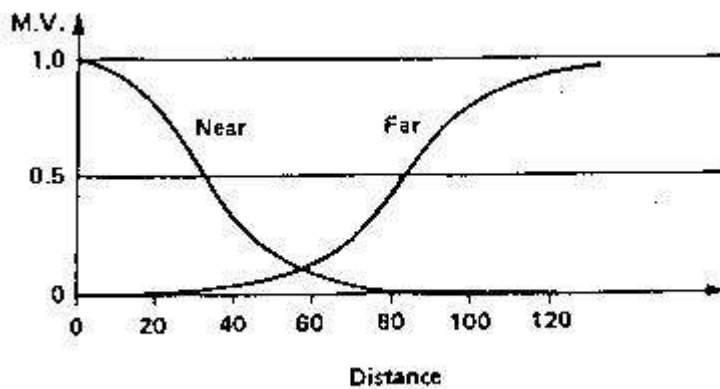


Fig. 1.

$$f(x) = \frac{1}{2} + \frac{1}{\pi} \tan^{-1}\left(\frac{x - k(1)}{k(2)}\right).$$

see Open-Meta .com site for full length versions of this poster paper, more details

Firefox runs SVG natively