Volume 21 (2002), number 4 pp. 777–786

SVG Linearization and Accessibility

Ivan Herman[1] and Daniel Dardailler[2]

[1]World Wide Web Consortium, c/o W3C Benelux Office at CWI, 1098 SJ Amsterdam, The Netherlands
[2]World Wide Web Consortium, c/o INRIA, 2004, route de Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex, France
ivan@w3.org, danield@w3.org

Abstract
The usage of SVG (Scaleable Vector Graphics) creates new possibilities as well as new challenges for the accessibility of Web sites. This paper presents a metadata vocabulary to describe the information content of an SVG file geared towards accessibility. When used with a suitable tool, this metadata description can help in generating a textual ("linear") version of the content, which can be used for users with disabilities or with non-visual devices.
Although this paper concentrates on SVG, i.e. on graphics on the Web, the metadata approach and vocabulary presented below can be applied in relation to other technologies, too. Indeed, accessibility issues have a much wider significance, and have an effect on areas like CAD, cartography, or information visualization. Hence, the experiences of the work presented below may also be useful for practitioners in other areas.

Keywords: 2D graphics, Scalable Vector Graphics (SVG), web accessibility, metadata in graphics, semantic web

ACM CSS: I.3.4 Graphics Utilities—*Graphics Packages*, I.3.6 Methodology and Techniques—*Graphics data structures and data types, Standards*, K.4.2 Social Issues—*Assistive technologies for persons with disabilities*

1. Introduction

Images play an important role in conveying information on the Web. However, the information presented in images must be accessible to all users, including those with visual disabilities or simply using non-visual devices. This problems has been recognized for a while already. For example, the Web Content Accessibility Guidelines [19], published by the World Wide Web Consortium (W3C), addresses this issue as part of its checklist. With reference to graphics, for example, authors of Web pages using traditional HTML are supposed to make use of the extra attributes available in the `img` element (`alt`, `longdesc`): using these attributes, the author can give a textual equivalent of the information contained in the image. Based on these attributes a specialized browser is able to convey the right information to the user (in parallel to or instead of the graphics content itself).

In this respect, SVG [4,17] creates new possibilities as well as new challenges. Some elements of SVG, namely the `title` and `desc` elements, make it possible to annotate every element in an SVG file. Thus, extra information can be provided, by-passed by a "traditional" viewer but usable by specialized user agents. This, and other accessibility features of SVG (grouping, styling, etc.) have already been explored by McCathieNevile and Koivunen in a W3C Note [8]. However, their approach is by no means a complete solution to the problem.

SVG is also very different from a traditional PNG or JPEG image. A bitmap image does not reveal the original *structure* of the information content. For example, the fact that one object in the image is "behind" or "on the left" of another cannot be retrieved from the pixels themselves (except maybe through sophisticated image analysis); unless the accompanying description is very verbose, it is quite difficult to add this sort of information to description attribute values. On the other hand, the SVG file itself retains these sorts of relationships and they are potentially available on the client side. This means that the information, which can be conveyed to a user with disabilities (or using non-visual devices) can be potentially much richer and more complex, hence also much more informative. Also, SVG may include animation and interaction; revealing the dynamic aspect of the SVG content may be essential to understand the content of the graphics.

A more systematic usage of metadata is a possible step forward. The term *metadata* (or "data about data") has been used for a while to denote data describing a particular resource (whether on the Web or not). A typical example is the cataloguing system used by libraries. The electronic version of metadata, often included in the resource itself, has come to the fore with the appearance of digital libraries, on-line databases, etc. These applications often rely on the availability of a coherent metadata about their resources, using a well-established, common vocabulary.

The World Wide Web consortium has engaged into an activity called the "Semantic Web" [2] whose goal is to provide a systematic, and standardized mechanism to define and describe metadata for Web resources. As part of this activity, W3C has defined the Resource Description Framework (RDF) [13,14], which is a way to encode simple metadata statements in XML. Furthermore, the `metadata` element in SVG allows the inclusion of any XML vocabulary into the SVG file; this data is ignored by the usual graphics players, but can be used by other processing agents. For example, through the inclusion of RDF into the `metadata` it is possible to add additional, and more structured information on the SVG content. Using RDF has the extra advantage of using a standardized formalism for metadata, thereby being able to re-use tools, visual editors, etc., to generate and process the metadata itself. (If the user is not familiar with RDF and/or its representation in XML, the RDF Primer can be particularly useful [14].)

The idea of the *linealizer tool*, described in this paper, is to extract information from the SVG file using the included metadata and to produce a textual ("linear") description of the information content of the graphics. This description, in our case a simple and text-only HTML file, can be used, for example, by a voice browser. This linealizer tool is not simply a metadata to HTML converter, though; while interpreting the metadata elements, the tool also extracts further information from the SVG structure itself. This approach has been first explored by Lovet and Dardailler in their SVG linearizer tool [6], as part of Lovet's internship at W3C/INRIA. This report continues the work explored by Lovet and Dardailler, by revisiting, improving, and extending the metadata vocabulary used in the original report and by extending the functionality of the linealizer tool.

In short, the tool performs the following steps:

(1) The author describes the SVG content using an RDF vocabulary described below. In general, this RDF information can be either embedded into the SVG file or can be in a separate file.

(2) The author also adds textual descriptions using the `desc` elements to (at least!) all elements which are referred to as primary RDF resources. All these elements should also be identifiable through an `id` attribute. For a specific element the tools considers the first child `desc` element as being the corresponding description.

(3) The linealizer tool reads the RDF information, combines each RDF resource with the textual description, possibly extract further information from the SVG file itself, and produces a simple HTML file on its output.

The rest of this paper concentrates on the RDF vocabulary used. The tool can be downloaded and tested; more about that below.

2. Some examples

The tool has been tested on a number of SVG files. All the files, as well as the generated HTML files, are available online, using the base URL:

```
 http://www.svgopen.org/papers/2002/
herman_dardailler_svg__linearization_
and_accessibility/
```

The first example has also been reproduced in the Appendix. The detailed description of the predicates in this paper makes use of the metadata contained in these files.

3. Predicates

This section gives a definition of the RDF predicates.

By default, the linealizer tool displays a textual version of the predicate, including some simple structure (e.g. in the form of bulleted lists) when possible. In some cases the tool tries to do more, as described with the predicate itself.

3.1. A Few Words about RDF

It is obviously not possible to give a full description of RDF here, just a few words to make the essentials features of what is described later understandable (again, the reader is referred to the RDF Primer of W3C [14] for a more systematic introduction to RDF).

An RDF file is a collection of simple statements of the type "subject predicate object". For example, the English statement "Blackwell is the publisher of CGF" could be described as:

- the *subject* is "Blackwell"
- the *predicate* is "publisher of"
- the *object* is "CGF"

Table 1:

	SVG File (with embedded RDF)	Generated HTML file
Architecture of a WebCGM [20] file	cgmarch.svg	cgmarch.html
A small SVG example with accompanying explanation and code	duckWithCode1.svg	duckWithCode1.html
Examples for various type of SVG paths	pathexamples.svg	pathexamples.html
SVG transformation examples	svgtransformations.svg	svgtransformations.html
The network example used in the accessibility note of McCathieNevile and Koivunen [8]	network.svg	network.html
W3C membership evolution chart	membership.svg	membership.html
Overview of W3C recommendations	recOverview.svg	recOverview.html
Simple deformation of a curve	PathAnimation.svg	PathAnimation.html
Spinning W3C Logo	animw3c.svg	animw3c.html
Dataflow network of filter effects	dataflowWithDucks.svg	dataflowWithDucks.html

The reader can look at the complete RDF description of these files by looking at the SVG source. Also, as a further example, a more complete slide set (prepared for a STAR report at the Eurographics'2001 conference in Manchester, UK [4]) has been prepared in SVG, and the content of each slide has been annotated with the metadata described in this paper. This slide set can be downloaded from the Web, too:
`http://www.w3.org/2001/Talks/IH-0609Manchester/`

In the case of RDF, both the subject and the object can be a general resource, identified for example by a URL; the object can also be a simple literal. A statement like the one above can be translated into XML as follows:

```
<rdf:Description rdf:about="URLforBlackwell">
   <PublisherOf resource="URLforCGF">
</rdf:Description>
```

In other words, predicates are described as XML elements, which are the immediate descendants of `rdf:Description` elements (identifying the subject resource) and which point a the URL for the object. If the object is a literal, then this value can simply appear as a text element enclosed by the predicate element; there will be examples for this later.

3.2. General Predicates

`GraphicsType` The object must be a literal, namely one of **`Chart`**, **`Conceptual`**, **`Clip-art`**, **`Decorative`**, **`Special`**.

The possible diversity of the graphics content on the Web is extremely high and some of the predicates below make sense only if the content's general category is clear to the user. The current taxonomy has been defined by Daniel Dardailler [3] based on an informal survey of a large number of Web pages (see also Section 5).

`IsAnchor` The predicate states whether the object is, or is part of, a hyperlink anchor (i.e. an SVG `a` element). If the object of the predicate is the literal **`true`**, the tool locates the closest `a` ancestor (possibly the object element itself if it is an anchor) and displays the value of the `xlink:href` attribute. Possible `desc` element belonging to the `a` element, as well as the `xlink:title` and `xlink:role` attributes (if present), are also displayed as a characterization of the link. If a resource is explicitly given, this is considered to be the URI for the anchor and is displayed literally.

`ContainsAnchor` The predicate states whether the object *contains* hyperlink anchors (i.e. an SVG `a` elements). If the object of the predicate is the literal **`true`**, the tool locates the descendant `a` elements and displays the values of the `xlink:href` attributes. Possible `desc` elements belonging to the elements, as well as the `xlink:title` and xlink:role attributes, are also displayed as a characterization of the links.

3.3. Structural Predicates

`Contains` Lists other objects (i.e. SVG content) contained by the (graphics) subject. This predicate describes the basic structure of the graphics content. The immediate object of this predicate is often a **`rdf:Bag`**, but this is not necessary.

`IsPartOf` This is the "counterpart" of **`Contains`**.

One might be tempted to extract the content automatically using the group structure of SVG, instead of relying on the metadata. Although this works well in simple cases, it can be misleading. Indeed, the SVG content may include elements which are present for "fanciness" (i.e. background color filters, etc.) but do not convey meaningful information to a textual agent. It is often better to ignore those for the purpose of linearization. Hence the introduction of separate predicates describing the structure of the information.

The following example (extracted from `cgmarch.svg`) states that the slide contains four parts, identified as `MetafileStructure`, `PictureDetails`, `PictureBody`, and `SymbolLibraries`. Note that here and in what follows the axsvg namespace prefix is used to identify the linealizer predicates. This namespace declaration is usually part of the enclosing **`rdf:RDF`** element (see the examples for details).

```
<rdf:Description about"#SlideContent">
  <axsvg:Contains>
    <rdf:Bag>
      <rdf:li rdf:resource="#MetafileStructure"/>
      <rdf:li rdf:resource="#PictureDetails"/>
      <rdf:li rdf:resource="#PictureBody"/>
      <rdf:li rdf:resource="#SymbolLibraries"/>
    </rdf:Bag>
  </axsvg:Contains>
</rdf:Description>
```

The usage of **`rdf:Bag`** is optional:

```
<rdf:Description about="#SlideContent">
  <axsvg:Contains resource="#MetafileStructure"/>
  <axsvg:Contains resource="#PictureDetails"/>
  <axsvg:Contains resource="#PictureBody"/>
  <axsvg:Contains resource="#SymbolLibraries"/>
</rdf:Description>
```

However, by using **`rdf:Bag`** the linealizer tool can produce a more structured output (turning the bag into a bulleted list).

`IsConvergencePoint`
`IsConnected`
`PointsTo`
`ExpandsTo`
`Associated` All these predicates describe connections between the subject and the object resources, often used in presentation diagrams. The objects of these predicates are often RDF bags.

The following example (again extracted from `cgmarch.svg`) states that the resource Picture1 (which is part of the `MetafileStructure`) “expands” to `PictureDetails`, i.e. the latter give the details for the former.

```
<rdf:Description about="#Picture1">
  <axsvg:ExpandsTo rdf:resource="#PictureDetails"/>
</rdf:Description>
```

`Connects` This predicate is used when the subject is the graphical “link” (i.e. line, curve, etc.) itself, and one wants to describe which other objects are connected by the subject. The immediate resource used by the predicate is usually an **`rdf:Seq`**, conveying also the strict order of the connected elements.

The following example (from `network.svg`) states that the resource CableA, referring to a cable, connects the computer to a socket.

```
<rdf:Description about="#CableA">
  <axsvg:Connects>
    <rdf:Seq>
      <rdf:li rdf:resource="ComputerA"/>
      <rdf:li rdf:resource="sock1"/>
    </rdf:Seq>
  </axsvg:Connects>
</rdf:Description>
```

`IsDef` The predicate states that the subject is a graphical template defined either through the `symbol` element or within a `defs`. The object of the predicate is the literal **`true`** or **`false`**.

`InstanceOf` The predicate states that the subject is an instance of a graphical template. The object of the predicate is either an SVG resource (which is also identified as the subject of an `IsDef` predicate), or a reified RDF statement making use of the **`Symbol`** and the **`Transformation`** predicates (see example below).

`Symbol` Identifies a template resource.

`Transformation` Describes the transformation applied on the symbol when instantiating it. The object is either a textual literal, describing the transformation, or a reference to an SVG element. In the latter case the tool extracts the transform attribute of the target and display it in the output.

The following example (extracted from `svgtransformations.svg`) contains three statements:

(1) The resource `duck` is a graphical template
(2) The resource `Original` is an instantiation of `duck`, without specifying any transformation
(3) The resource `simpleTranslate` is also an instantiation of `duck`, but translated to the (200,190) position.

```
<rdf:Description about="#duck">
  <axsvg:IsDef>true</axsvg:IsDef>
</rdf:Description>
<rdf:Description about="#Original">
  <axsvg:InstanceOf rdf:resource="#duck"/>
</rdf:Description>
<rdf:Description about="#simpleTranslate">
  <axsvg:InstanceOf>
    <rdf:Description>
      <rdf:type
       rdf:resource="http://www.w3.org/2001/svgRdf/
               axsvg-resource.rdf#UseDescription"/>
      <axsvg:Symbol rdf:resource="#duck"/>
     <axsvg:Transformation>translate(200,190)
     </axsvg:Transformation>
    </rdf:Description>
  </axsvg:InstanceOf>
</rdf:Description>
```

Note the usage of the **`rdf:type`** predicate. This predicate refers to a more precise specification of the subject, using RDF Schemas [15]. This predicate tells us that the subject of the **`axsvg:InstanceOf`** predicate is an anonymous resource of a special type, describing the details of a `use` SVG element.

`Text` The predicate states that the subject is a textual node, i.e. that the text content can be displayed directly. The object of the predicate is the literal **`true`** or **`false`**. The tool extracts the text from the SVG element and displays it verbatim.

`LabelledBy` The predicate refers to an SVG resource or a bag of resources which label the subject. The tool extracts the text from the SVG element(s) and displays the content(s) verbatim. If the object is a literal value, this is displayed instead.

`Labelled` The predicate has the possible literal value of **`true`** or **`false`**. In the former case the tool extracts the text from the subject SVG element(s) and displays the content verbatim.

The resources identified by these predicates are *not* (necessarily) text nodes in the SVG sense. They can be, for example, texts enclosed in groups or even more complicated graphical objects, where only the textual parts are meaningful for a non-visual environment.

The following example (extracted from `svgtransformations.svg`) identifies three textual elements used as labels for `scaleSkewXTranslate`. The tool displays the textual contents of the `translate4`, `skew4`, and `scale4` elements.

```
<rdf:Description about="#scaleSkewXTranslate">
  <axsvg:LabelledBy>
    <rdf:Bag>
      <rdf:li resource="#translate4"/>
      <rdf:li resource="#skew4"/>
      <rdf:li resource="#scale4"/>
    </rdf:Bag>
  </axsvg:LabelledBy>
</rdf:Description>
```

3.4. Layout and Positional Predicates

These predicates give an abstract description on the placement of the various SVG objects relative to one another or relative to the full canvas.

`InPosition` The predicate gives a rough absolute position of the subject on the full canvas. The object of the predicate is a literal with the possible value of **`N`**, **`S`**, **`W`**, **`E`**, **`NE`**, **`NW`**, **`SE`**, **`SW`**, **`NNE`**, **`NNW`**, **`ENE`**, **`WNW`**, **`SSW`**, **`SSE`**, **`ESE`**, **`WSW`**, or **`M`**, denoting 16 possible directions (north, south, west, east, north-east, etc.) and the term “middle”.

`InDirection` Similar to **`InPosition`**, except that the position of the subject is described *relative* to another object. The object of the predicate is a reified statement using **`RelativeTo`** and **`Direction`**.

`RelativeTo` Identifies an object used for **`InDirection`**.

`Direction` Refers to a direction value, used for **`InDirection`**. The possible objects are identical to **`InPosition`**.

The following example (extracted from `svgtransformations.svg`) states that `Original` is placed in the middle of the full SVG viewport, and that `simpleTranslate` is in a south-east direction of Original.

```
<rdf:Description about="#Original">
  <axsvg:InPosition>M</axsvg:InPosition>
</rdf:Description>
<rdf:Description about="#simpleTranslate">
  <axsvg:InDirection>
    <rdf:Description>
      <rdf:type
       rdf:resource="http://www.w3.org/2001/svgRdf/
         axsvg-resource.rdf#DirectionDescription"/>
      <axsvg:RelativeTo rdf:resource="#Original"/>
      <axsvg:Direction>SE</axsvg:Direction>
    </rdf:Description>
  </axsvg:InDirection>
</rdf:Description>
```

`IsGoingThrough`
`MaskedBy`
`InFrontOf`
`Behind`
`AtLeft`
`AtRight`
`OnTop`
`Under`
`HasOnTop`
`IsLayeredOn` All these predicates can be used to provide additional information of the object’s position relative to the subject. Their role is quite clear.

3.5. Chart Predicates

These predicates are specific to chart-like figures.

`ChartType` The predicate describes the type of the chart. Possible (literal) object values are: **`Bar`**, **`Line`**, **`Pie`**, **`Scatter`**, **`Area`**, **`Doughnut`**, **`Radar`**, **`Bubble`**, **`Stock`**, **`Surface`**. (The terms are those used by Microsoft Excel for the generation of charts.)

`Chart` This predicate describes the important data described by a chart. The object of the predicate is a special resource using **`DataCategory`** and **`DataValues`**. (See the example below.)

`DataCategory` The object of this predicate is either a resource treated as a textual node, or a literal. It gives a user-dependent characterization of the data set.

`DataValues` The object of this predicate gives the real data values. The object is either a literal (which contains the real data) or another SVG object (in which case the tool attempts to extract a `desc` object as a textual description of the values).

The following example (extracted from `membership.svg`) describes the three data lines which is central to the chart; each has a category and a reference to a resource. To make the example clearer, the SVG content for `totalMemberLine` is also shown; the tool displays the content of the `desc` element.

```
<rdf:Description about="#DataLines">
  <axsvg:Chart>
    <rdf:Bag>
      <rdf:li>
        <rdf:Description>
          <rdf:type
           rdf:resource="http://www.w3.org/2001/svgRdf/
                axsvg-resource.rdf#ChartDescription"/>
          <axsvg:DataCategory>
                Total Members
          </axsvg:DataCategory>
          <axsvg:DataValues
                rdf:resource="#totalMemberLine"/>
        </rdf:Description>
      </rdf:li>
      <rdf:li>
        <rdf:Description>
          <rdf:type
           rdf:resource="http://www.w3.org/2001/svgRdf/
                axsvg-resource.rdf#ChartDescription"/>
          <axsvg:DataCategory>
                Full Members
          </axsvg:DataCategory>
          <axsvg:DataValues
                rdf:resource="#fullMemberLine"/>
        </rdf:Description>
      </rdf:li>
      <rdf:li>
        <rdf:Description>
          <rdf:type
           rdf:resource="http://www.w3.org/2001/svgRdf/
               axsvg-resource.rdf#ChartDescription"/>
          <axsvg:DataCategory>
                Affiliate Members</axsvg:DataCategory>
          <axsvg:DataValues
                rdf:resource="#affMemberLine"/>
        </rdf:Description>
      </rdf:li>
    </rdf:Bag>
  </axsvg:Chart>
</rdf:Description>
...
<path id="totalMemberLine" class="totalMembers"
                d="M0 50
                   L50, 108 L100,117 L150,152
                   L200,158 L250,179 L300,231
                   L350,264 L400,293 L450,329
                   L500,363 L550,419 L600,478
                   L650,509">
 <desc>50,108,117,152,158,179,231,264,293,329,
                363,419,478,509</desc>
</path>
```

Unfortunately, this mechanism forces the author to write the data twice: once for the coordinates in path and once in the description. One could envisage an automatic mechanism extracting the coordinate values from the path directly, but the correct interpretation of the “d” attribute of a path is not obvious (only the “Y” values are meaningful in this example). Besides, other shape elements could also be used, not only paths.

`XLegend`
`YLegend`
`ZLegend` Referring to the legend of the various chart axes. If the object of the predicate is a SVG resource, it will be interpreted similarly to a textual node.

3.6. Animation Predicates

These predicates describe the effects of animation objects on other objects in SVG.

Animation objects in SVG have a relatively high “granularity”: to perform a specific animation effect one may have to use several SVG animation objects whose combined effects are perceived by the user as one animation. On a metadata level one should usually hide this granularity. This means that while, in what follows, the term “animation” or “animate” will be used in the sense used in the SVG recommendation, “animation resource” might also mean a group of animation objects, or an RDF resource defined directly in the `metadata` section to describe an animation effect.

`AnimatedBy` Refers to the fact that the subject is animated (moved, deformed, etc.) by an animation object, whose reference is the object of the predicate.

`AnimationSubjects` This predicate should be used with a resource describing an animation; it refers to either an SVG resource or a bag of resources which are affected by the animation.

`AnimationBegins`
`AnimationDuration` Beginning and duration of an animation. The object may refer to a literal or a “real” SVG animation object; in the latter case the tool extracts the `begin` (resp. `dur`) attribute of the SVG element (if it exists). If a literal is used, one can add a general description or refer to the formalism used in SVG for the `begin` (resp. `dur`) attribute.

`AnimationSeq`
`AnimationPar` Although it is possible to describe the full time line of the animations with the begin and duration attributes, in a number of cases this is much too detailed. **`AnimationSeq`** (resp. **`AnimationPar`**) lists a number of animation resources which are executed in a sequence (resp. in parallel). These are simplified versions of the SMIL constructs [9,18].

The following example (extracted from `PathAnimation.svg`) contains the animation on a simple path. Note the usage of:

- **`rdf:ID`** to identify a separate resource describing animation, instead of referring to an SVG element. It is sometimes quite useful to use this construction (i.e. generate description elements which are not directly bound to a specific SVG element).
- `desc` in conjunction with the `#Animation` resource. If the RDF description is part of an SVG file the `desc` *is* the same description element as elsewhere in the SVG file, and can be used for a description (and reused by the tool).
- **`rdf:type`** to make it sure that the resource is really the description of the animation; this presupposes the separate RDF Schema specification.

```
<rdf:Description rdf:about="#AnimatedDuck">
  <axsvg:AnimatedBy rdf:resource="#Animation"/>
</rdf:Description>
<rdf:Description rdf:ID="#Animation">
  <desc>
     Deforms the duck into a potato and back
  </desc>
  <rdf:type
     rdf:resource="http://www.w3.org/2001/svgRdf/
       axsvg-resource.rdf#AnimationDescription"/>
  <axsvg:AnimationSubjects
       rdf:resource="#AnimatedDuck"/>
  <axsvg:AnimationBegins>
    user clicks on duck
  </axsvg:AnimationBegins>
  <axsvg:AnimationDuration>
    2s
  </axsvg:AnimationDuration>
</rdf:Description>
```

The following example (extracted from `animw3c.svg`) is more complex, and describes the animation on the spinning W3C logo and the disappearing domain logos. Note that, instead of describing the detailed timing, **`AnimationSeq`** is used. This is not absolutely precise, because the fading out of the activity logos begin one second *before* the end of the W3C logo spin, but this detail is not really important if the tool is used to display the content of the image textually.

```
<rdf:Description rdf:about="#opaciteimg2">
  <axsvg:AnimationSubjects
      rdf:resource="#domainLogos"/>
  <axsvg:AnimationDuration>
      2s
  </axsvg:AnimationDuration>
</rdf:Description>
<rdf:Description rdf:ID="#logoAnimate">
  <desc>
      Spins around the logo by also reducing
          its size to almost invisible
  </desc>
  <axsvg:AnimationSubjects rdf:resource="#W3CLogo"/>
  <axsvg:AnimationDuration>
      5s
  </axsvg:AnimationDuration>
</rdf:Description>
<rdf:Description rdf:ID="#logoAnimate2">
  <desc>
      Scales the logo up to a reasonable size
  </desc>
  <axsvg:AnimationSubjects rdf:resource="#W3CLogo"/>
  <axsvg:AnimationDuration>
      4s
  </axsvg:AnimationDuration>
</rdf:Description>
<rdf:Description rdf:ID="#animationTiming">
  <rdf:type
      rdf:resource="http://www.w3.org/2001/svgRdf/
          axsvg-resource.rdf#AnimationDescription"/>
  <desc>
    Describes the relative timing of the animations
  </desc>
  <axsvg:AnimationSeq>
    <rdf:Seq>
      <rdf:li rdf:resource="#logoAnimate"/>
      <rdf:li rdf:resource="#logoAnimate2"/>
      <rdf:li rdf:resource="#opaciteimg2"/>
    </rdf:Seq>
  </axsvg:AnimationSeq>
  <axsvg:AnimationBegins>
      at load time
  </axsvg:AnimationBegins>
</rdf:Description>
```

3.7. Miscalleneous Predicates

`VisualEffects` Refers to special filters, color gradient fills, pattern fills, etc., which produce complex visual effects. Most of these effects may simply be ignored in the metadata description if the goal is to produce information for a visually impaired user; if there is a necessity to refer to the existence of these effects, the textual description should suffice. This predicate simply binds an object to a subject controlling the visual effects.

`Shape` Refers to the geometric shape of the subject; the literal value can be `rectangle`, `circle`, etc., i.e. the shapes defined by SVG. Note that the tool could extract, for example, the shapes coordinates from the corresponding SVG element, but it is not clear that this information would really be meaningful.

In most of the cases, the shape itself does not convey any important information on the content of the slide, so the usage of that predicate might be less useful than one might think at first.

4. The Linealizer Tool and its Future Developments

The current linealizer tool is a proof-of-concept implementation rather than finished software. The first version of the tool, by Lovet and Dardailler [6] was written in Java, the current version is based on XSLT for easier maintainability. The tool is restricted in its management of resources: it can manage simple `#id` type of fragment identifiers only, and only for elements in the same file (i.e. the RDF metadata *must* be in the same SVG file). A more elaborate version of the tool should include a full URI management (which could lead to metadata files "outside" the SVG source) as well as a full XPath implementation to identify elements. Also, at present, the RDF parser can only handle the basic serialization

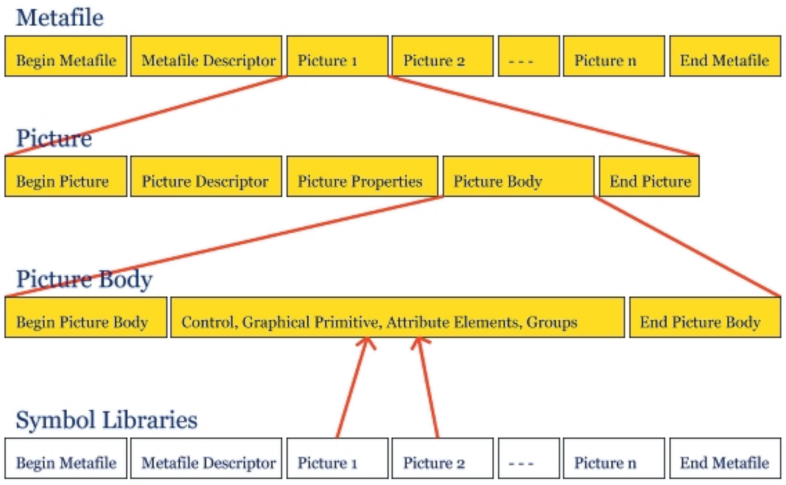


Figure 1: *Representation of a WebGCM file structure.*

syntax of RDF [13]. The tool itself consists of two XSLT files which can be downloaded from the W3C web site: `http://www.w3.org/2001/svgRdf/rdfp.xsl` and `http://www.w3.org/2001/svgRdf/Properties.xsl`. The XSLT files rely on the Saxon XSLT processor [16].

Generating the RDF metadata is done mostly by hand for now. SVG authoring tools should give the authors the possibility to annotate the graphics elements using at least the `desc` and `title` elements, but this is not yet the case. Ideally, the `metadata` section should also be editable from within the authoring tool. One of the advantages of using RDF, though, is that one can reuse tools developed for RDF in general. As an example, the IsaViz [12] tool for the visual generation and editing of RDF metadata has become recently available, and has already proven to be extremely helpful in generating the metadata for linearization, too.

The vocabulary, as described in the paper, has also been formally defined using using RDF Schema [15] (the schema can also be downloaded; the file name is `axsvg-schema.rdf`). Although the current tool does not check the validity of the metadata predicates against this schema yet, this is clearly one of the directions for further work.

5. Further Research

One of the important features of the system is that the RDF vocabulary does not try to cover everything and it also relies on the human description provided with the resources. In our view, it is this interplay between the "formal" description in RDF and the "informal" descriptions which gives an extra power to our approach.

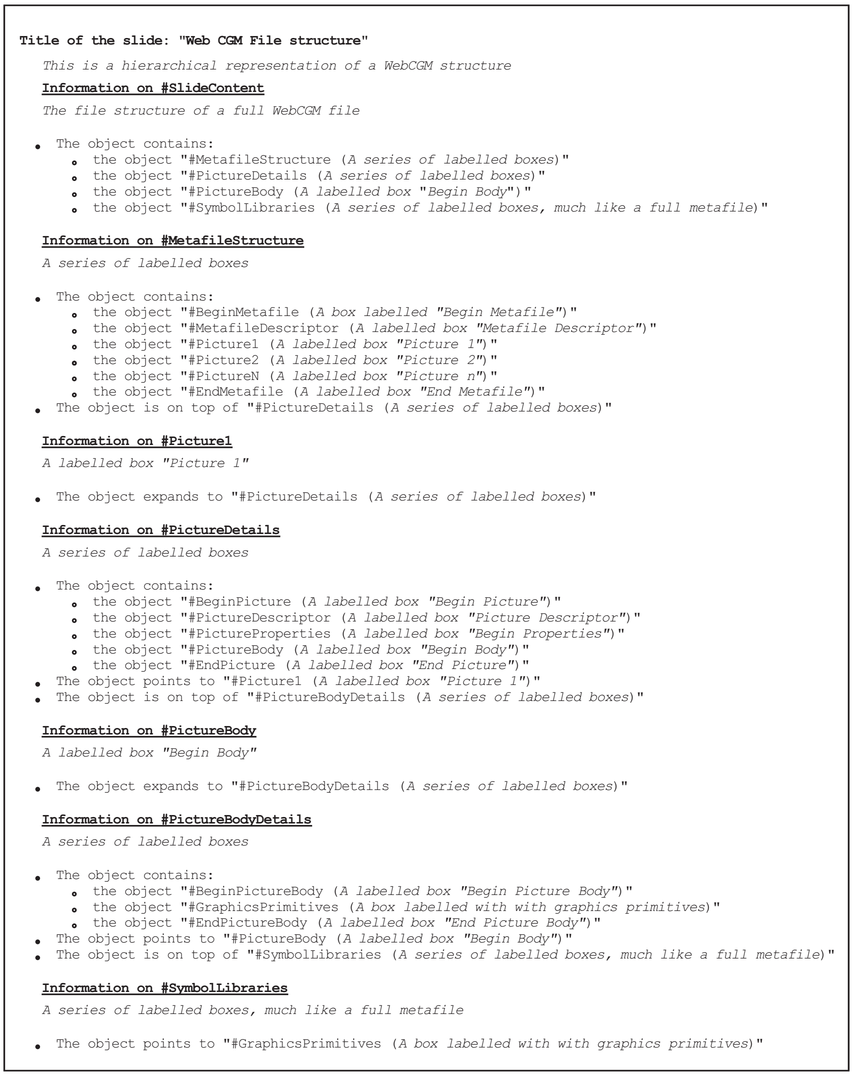
Nevertheless, the specification of the right vocabulary is undeniably the hardest research issue to evolve this approach further. It is probably not possible to define one all-encompassing vocabulary; rather, one should rely on various vocabularies developed in specific areas (e.g. constraint based graphics [1], automatic generation of multimedia content [10], etc.). Using different namespaces (instead of the only namespace used in this document, i.e. `axsvg`) one could envisage a rich metadata vocabulary structure well adapted to the particular usage of a specific SVG file, and where vocabularies belonging to different areas could be used side by side. Possible equivalence of the various taxonomies can be secured through web ontology tools currently under development within the framework of the W3C Semantic Web activities [2].

However, even if one can borrow from the terminology of specific areas, like the ones cited above, work still has to be done in adapting or developing vocabularies. Perhaps surprisingly, certain areas of information visualization research might be useful here. Information visualization is concerned about finding the right abstractions to present data visually. However, it has been recognized that to find those abstractions, one has to gain a thorough understanding of both the low, perceptual level of human understanding, as well as the higher, cognitive level mechanisms [5,7,11]. Understanding the higher level cognitive processes may help in finding the right taxonomy for the description of pictorial data, which is exactly what the metadata vocabulary is all about. But this is clearly the subject of further, extensive and interdisciplinary research.

Appendix

The WebCGM Example

The SVG file in `cgmarch.svg` defines the image shown in Figure 1:



Title of the slide: "Web CGM File structure"

This is a hierarchical representation of a WebCGM structure

Information on #SlideContent

The file structure of a full WebCGM file

- The object contains:
  - the object "#MetafileStructure (*A series of labelled boxes*)"
  - the object "#PictureDetails (*A series of labelled boxes*)"
  - the object "#PictureBody (*A labelled box "Begin Body"*)"
  - the object "#SymbolLibraries (*A series of labelled boxes, much like a full metafile*)"

Information on #MetafileStructure

A series of labelled boxes

- The object contains:
  - the object "#BeginMetafile (*A box labelled "Begin Metafile"*)"
  - the object "#MetafileDescriptor (*A labelled box "Metafile Descriptor"*)"
  - the object "#Picture1 (*A labelled box "Picture 1"*)"
  - the object "#Picture2 (*A labelled box "Picture 2"*)"
  - the object "#PictureN (*A labelled box "Picture n"*)"
  - the object "#EndMetafile (*A labelled box "End Metafile"*)"
- The object is on top of "#PictureDetails (*A series of labelled boxes*)"

Information on #Picture1

A labelled box "Picture 1"

- The object expands to "#PictureDetails (*A series of labelled boxes*)"

Information on #PictureDetails

A series of labelled boxes

- The object contains:
  - the object "#BeginPicture (*A labelled box "Begin Picture"*)"
  - the object "#PictureDescriptor (*A labelled box "Picture Descriptor"*)"
  - the object "#PictureProperties (*A labelled box "Begin Properties"*)"
  - the object "#PictureBody (*A labelled box "Begin Body"*)"
  - the object "#EndPicture (*A labelled box "End Picture"*)"
- The object points to "#Picture1 (*A labelled box "Picture 1"*)"
- The object is on top of "#PictureBodyDetails (*A series of labelled boxes*)"

Information on #PictureBody

A labelled box "Begin Body"

- The object expands to "#PictureBodyDetails (*A series of labelled boxes*)"

Information on #PictureBodyDetails

A series of labelled boxes

- The object contains:
  - the object "#BeginPictureBody (*A labelled box "Begin Picture Body"*)"
  - the object "#GraphicsPrimitives (*A box labelled with with graphics primitives*)"
  - the object "#EndPictureBody (*A labelled box "End Picture Body"*)"
- The object points to "#PictureBody (*A labelled box "Begin Body"*)"
- The object is on top of "#SymbolLibraries (*A series of labelled boxes, much like a full metafile*)"

Information on #SymbolLibraries

A series of labelled boxes, much like a full metafile

- The object points to "#GraphicsPrimitives (*A box labelled with with graphics primitives*)"

Figure 2: *Representation of a WebGCM file structure.*

Using the metadata included in this file, the generated textual output (formatted as a simple HTML file) is shown in Figure 2 (the text in italics is extracted from the user's description elements).

References

1. G.J. Bardos, J.J. Tirtowidjojo, K. Marriott, B Meyer, W. Portnoy and A. Borning. A constraint extension to scalable vector graphics. In *Proceedings of The Tenth International World Wide Web Conference*. ACM Press, 2001.

2. T. Berners-Lee, J. Hendler and O. Lassila. *The Semantic Web*. Scientific American, 2001.

3. D. Dardailler. *Dependency statements for SVG*. W3C Note. Available at `http://www.w3.org/WAI/PF/dep-svg`, February 2000.

4. D.A. Duce, I. Herman and F.R.A. Hopgood. Web 2D graphics file formats. *Computer Graphics Forum*, 21(1):43–64, 2002.

5. D.J. Duke and I. Herman. Minimal Graphics. *IEEE Computer Graphics & Application*, 21(6), 2001. A more detailed version is in: Reports of the Centre for Mathematics and Computer Sciences, INS-9903 (1999), ISSN 1386-3681, available at `http://www.cwi.nl/˜ivan/AboutMe/Publications/INS-R9903.pdf.`

6. G. Lovet and D. Dardailler. *SVG Linearizer tools*. W3C Note. Available at `http://www.w3.org/WAI/ER/ASVG/`, September 2000.

7. A.M. MacEachren. *How Maps Work*. Guilford Press, New York, 1995.

8. C. McCathieNevile and M-R. Koivunen. Accessibility Features of SVG. *W3C Note*. Available at `http://www.w3.org/TR/SVG-access/`, August 2000.

9. L. Rutledge. SMIL 2.0: XML for Web Multimedia. *IEEE Internet Computing*, 5(5), 2001.

10. L. Rutledge, B. Bailey, J. van Ossenbruggen, L. Hardman and J. Geurts. Generating presentation constraints from rhetorical structure. In *Proceedings of the 11th ACM conference on Hypertext and Hypermedia*. ACM Press, New York, 2000.

11. C. Ware. *Information Visualization*. Morgan Kaufman, San Francisco, 2000.

12. IsaViz: A Visual Authoring Tool for RDF, `http://www.w3.org/2001/11/IsaViz/`.

13. Resource description framework (RDF) model and syntax. *W3C Recommendation*. Available at `http://www.w3.org/TR/REC-rdf-syntax/`, February 1999.

14. *RDF Primer*. W3C Working Draft. Available at `http://www.w3.org/TR/rdf-primer/`, April 2002.

15. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Working Draft. Available at `http://www.w3.org/TR/rdf-schema/`, April 2002.

16. *Saxon XSLT Processor*. `http://saxon.sourceforge.net`.

17. *Scalable Vector Graphics (SVG) 1.0 Specification*. W3C Recommendation. Available at `http://www.w3.org/TR/SVG/`, September 2001.

18. *Synchronized Multimedia Integration Language (SMIL 2.0) Specification*. W3C Recommendation. Available at `http://www.w3.org/TR/smil20/`, August 2001.

19. *Web Content Accessibility Guidelines 1.0*. W3C Recommendation. Available at `http://www.w3.org/TR/WAI-WEBCONTENT/`, May 1999.

20. *WebCGM Profile*. W3C Recommendation. Available at `http://www.w3.org/TR/REC-WebCGM/`, January 1999.