An approach to a CGM DOM

Dieter Weidenbruck, ITEDO Software,
Lofton Henderson, Consultant
Draft: March 26, 2000

Introduction

Increasing use of WebCGM structured graphics has led to new requirements for accessing and manipulating aspects of a graphics file at runtime, for which needs proprietary approaches have already been prototyped and shown. WebCGM clearly describes the content of intelligent CGM graphics, but it doesn't address a standard interface between the viewer and its hosting application needed for navigation, query, and extraction requirements. Similar to Document Object Models (DOM) for XML and HTML, a CGM DOM would provide standardized, direct, programmable access to individual components of CGM graphics, and would take Intelligent Graphics viewing to the next level of integration with other structured content. In this paper, we examine the issues and options for a standard graphics DOM, discuss possible design for a CGM DOM, and review activity towards a standard CGM DOM in standards organizations.

Considerations

The Document Object Model (DOM), a W3C Recommendation (standard), defines a way to access and manipulate the content of a document at runtime. Thus it provides a standardized interface to structured content. This access is not limited to any specific format such as XML. For any format a DOM can be defined based on the DOM specification. As described in DOM section 1.1.2 , the DOM specifies interfaces that can be used at runtime. It is not required that the data format itself stores the content in classes as they are exposed through the API.
Considering this basic design principle it becomes apparent that similar formats with structured content could use the same DOM. At least there could be a common core DOM with extensions for specific formats. This paper discusses a shared DOM for structured graphic formats.

WebCGM and SVG

WebCGM has been a W3C Recommendation since early 1999. It describes a specific version of CGM for usage on web pages. SVG is about to become a recommendation of W3C in 2000. It describes an XML-encoded, stylable vector graphics format for usage in XML or HTML environments. WebCGM is perceived as a high-end format for long-lived technical graphics on the Web, whereas SVG is especially suited for Web graphics with emphasis on high display quality and effects such as animation.
Although both formats seem to be very different at first glance they serve one purpose only: the usage of structured graphics in structured information environments. Both formats establish a tree structure of nodes at runtime. These nodes contain elements with graphical and non-graphical attributes. The fundamental difference is found within the files: A CGM is a metafile, i.e. a sequence of graphical primitives and attributes that establish a picture when rendered one after the other. SVG is an XML grammar, and an SVG file is a well-formed XML file: the graphic and non-graphic content, with all associated attributes, form a proper tree structure. However, at runtime both formats can be read and interpreted such that the elements with all attributes are kept in memory in a similar way.
The SVG Working Draft specifies an SVG DOM that allows for access and manipulation of objects at runtime. At present, there is no known CGM DOM available yet.

Review of existing approaches

At XML Europe 99 da Ponte/Goertz/Henderson [10] presented a paper discussing an architecture for standardized intelligent graphics. This study examined the requirements in aerospace industry and developed a systematic approach how standards should be used regarding intelligent graphics. One of the conclusions of this paper was that there is a need for a CGM DOM, and the authors designed a very clean and CGM-oriented draft of such a DOM. In this context the term “CGM-oriented” means that the functionality and the data model expressed in their document object model would reflect very specific CGM properties and functionality. Here are some examples quoted from this paper:

Excerpt from da Ponte/Goertz/Henderson [10]

metafile

Properties	Collections	Methods	Events
Title	Pictures	...	onload
Version	...		onunload
Description			...
Picture			
Location			
...			

DESCRIPTION

Represents the Computer Graphics Metafile in a given CGM viewer. You can use the metafile object to retrieve information about the metafile, to examine the pictures within the metafile, and to process events. The metafile object is available at all times. You can retrieve the object by applying the **metafile** property to a viewer object.

...

PROPERTIES

title	The BeginMetafile string.
version	The MetafileVersion element.
description	The MetafileDescription string.
picture	The currently displayed picture object.
location	The URI of the metafile.

COLLECTIONS

pictures	A collection of all picture objects contained in the metafile.

...

Comment: This example shows how somebody could extract very detailed information about a CGM file at runtime down to the level of individual CGM elements. This represents an excellent way to describe a CGM file.

Excerpt from Weidenbrück [9]

A second contribution to this topic was made at XML Philadelphia 99 by Weidenbrück [9] regarding the manipulation of CGM objects in viewing environments. This paper focussed on the runtime aspect of a CGM file without looking at CGM details found in the file. Apart from generic structures it did not provide calls to extract specific metafile information like the first approach [10]. Instead, an initial list of generic properties and structures was described that could be inspected and/or manipulated at runtime. Here is one of these generic properties of a line element:

Stroke

The stroke attribute stands for the line weight of the lines of the object(s). It is expressed as a floating number in millimeters or as a percent number. The stroke can be set in an absolute or relative way.

Example:

Stroke = 0.6 mm	absolute setting for all line elements having a stroke
Stroke = 150 %	relative setting, maintains the relationship between line weights

Comment: The motivation for this approach stems from the fact that users want to manipulate the content of a graphic at runtime. They are less interested in the details of how an element was stored in a physical file.

Both papers formed the foundation for the approach described in this paper. Additional input came from the work of the SVG working group [3].

A Standard Graphics DOM

Based on the considerations above and their own experiences with CGM the authors of this paper recognized the opportunity to utilize the separation between interface and implementation to draft a generic DOM for structured vector graphics. Currently only (Web)CGM and SVG are considered, however, as this is a very generic approach it should work with other formats too.
The clear benefit would be that the same API could be used to control identical actions while using different formats, e.g. the highlighting of an object in the graphics viewer.
As the scope of WebCGM is not congruent with SVG this approach may not work for all features. SVG defines animation, whereas CGM does not. Still a common DOM could contain API calls for animation, but they would be empty stubs for CGM.
As a first step the authors examined the elements and properties of CGM and SVG to find analogies. Subsequently the current draft for an SVG DOM was tested for usability in CGM environments.

Comparing CGM, WebCGM, and SVG

The following table shows a comparison of the graphical output capabilities of the three formats. A substantial overlap in functionality between CGM and SVG is apparent from the table.

SVG-CGM Comparative Graphical Output Capabilities			
Descriptive Capability	ISO CGM	SVG 1.0	Web-CGM
Straight lines & segmented straight lines (polylines)	X	X	X
Segmented straight lines (disjoint polylines)	X	-	X
Filled polygons and polygon sets	X	X	X
Rectangles and circles	X	X	X
Markers of predefined styles	X	-	X
Text strings	X	X	X
Circular arcs	X	X	X
Elliptical arcs	X	X	X
Circular and elliptical pie and chord sectors	X	-	X
Pixmap and RLC raster images	X	X	X
Color attributes (RGB and indexed color)	X	[P]	X
Dash style (predefined) and line width for lines and edges	X	-	X
Area fills, pre-defined patterns	X	-	X
Area fills, user-defined patterns	X	X	X
Text font, character set, placement, alignment, extent, orientation, etc.	X	X	X
Segments which can be transformed and instantiated	X	X	-
Closed figures consisting of multiple elements, end-to-end	X	X	X
Precise clipping and shielding, rectangular and general path	X	X	[P]
Definable attribute bundles (SVG: styles)	X	X	-
Save/restore graphical context	X	?	-
Tiled raster images: compression schemes include G4, JPEG, PNG	X	[P]	X
Advanced curve definitions: cubic Bezier	X	X	X
Advanced curve definitions: conic arcs and B-splines	X	-	-
Definable symbols and markers	X	X	-
External symbol libraries	X	X	X
Text along arbitrary paths	X	X	-
Advanced text controls and attributes	X		[P]
Definable cross hatching	X	X	X
Advanced areas fills including geometric, wallpaper, and interpolated	X	X	-
New color models including CMYK, CIELUV, and CIELAB	X	?	-
Precise line attributes: caps, joins, miter limit, definable dash patterns	X	X	X
Application structures	X	X	X
Structure attributes	X	X	[P]
Picture directory	X	-	-

Structure directory	X	-	-

The difference between file and memory

Both SVG and CGM comprise a definition of a line element with certain attributes. In SVG the line may look as follows:

```
<svg width="400" height="400">
  <g style="fill:none; stroke:green">
    <line x1="100" y1="300" x2="300" y2="100"
          style="stroke-width:5" />
  </g>
</svg>
```

Here is the same line expressed in a CGM:

```
BEGMF 'sample.cgm                        ';
...
 BEGPIC 'Picture 1';
VDCEXT 0,400 400,0;
 ...
 BEGPICBODY;
 ...
  LINECOLR 5;
  LINEWIDTH 0.5;
  LINETYPE 1;
  LINE 100,300 300,100;
 ENDPIC;
ENDMF;
```

Of course a line can carry additional attributes like a dash-pattern or an ID. The example shows that CGM and SVG use a different syntax to describe the line element. However, once the line has been read into memory at runtime it will be a line with two end points, a solid green stroke of a defined width. Other elements show similar analogies. The conclusion is that the syntax inside the files is different, however, the result on screen is very similar. This leads to the question whether this line could be accessed in the same way disregarding the fact whether it once had been stored inside a CGM or an SVG file.

Comparing structures

To access individual elements or their attributes it is essential that the formats use similar structures. In general, the required structure is a tree structure with nodes, as described in the DOM specification.
Both SVG and CGM exhibit a true tree structure with only three differences. CGM has the concept of multiple pictures residing inside the same file. These pictures represent independent images that are logically independent. In practice, most CGM files contain only one picture, and in fact some important industry profiles of CGM contain this restriction. It is the opinion of the authors that this should be defined as the default case. There is little to no benefit coming from the fact that multiple pictures reside in one file. On the other hand there are disadvantages, e.g. the entire file needs to be downloaded to view one picture only. If a CGM file is restricted to one picture only the difference to SVG is removed.
The second structural difference is that CGM can contain objects that form an entity but are not in subsequent order in the file, a.k.a. the “continued APS (application structure) principle”. Some elements may be listed that belong to a group, then other elements follow that belong to a different group again followed by elements belonging to the first group. The intent is to enable the grouping of elements that are not subsequent in the Z or display order of the file. There are several ways how this structural difference could be expressed to avoid discrepancies between CGM and SVG. This is considered to be outside the scope of this paper.
The third difference is that some attributes inside a CGM file can occur multiple times on the same graphic object. In XML there can only be one instance of an attribute on an element. There are several ways around this situation, e.g. adding some kind of enumeration, or redefining the attribute in a way that it can contain multiple values by means of a structured data record (SDR).
So in essence both formats provide the following structure:
Top node or root: the picture itself (embedded or standalone)
Children: objects with other objects embedded

DOM at work

The separation of interface and implementation opens a generic way to work with structured graphics at runtime no matter what format they have been stored in. Here are some examples:
Current SVG DOM:

```
string = SVGCtl1.getSVGDocument.getElementById("line1").nodeName
```

Generic DOM used with SVG viewer:

```
string = SVGCtl1.getDocument. getElementById("line1").nodeName
```

Generic DOM used with CGM viewer

```
string = CGMCtl1.getDocument. getElementById("line1").nodeName
```

It is obvious that the name of the viewer control can be hidden in a variable that would work for both formats:

```
string =GrCtl.getDocument. getElementById("line1").nodeName
```

This variable would be set to point to the viewer needed for the actual file. The script itself would be identical.

Conclusion & Further Work

Users want to work with the logical structure of a graphic without having to care too much for the file format being used. Fortunately, there are substantial similarities between today's structured 2D vector graphics formats for Web application, WebCGM, and SVG. In addition to these predominant web formats there are others that fall into the same category, including but not limited to IGES, DXF, VRML, and others.
Standardization means to harmonize existing approaches in order to define a common interface that works for all parties. With regard to document object models, this should not be restricted to a standard DOM for SVG plus a standard DOM for CGM plus standard DOMs for other formats. Even these different models share a substantial portion of analogous functionality and therefore should be harmonized. The result would be one DOM for structured vector graphics with a functionality that could be used for a multitude of graphic formats. As the functionality varies between formats not all of this DOM may be supportable within the scope of each format. The authors are convinced that a generic graphics DOM would greatly simplify the work with intelligent graphics and ensure that these complex object models indeed become available in a lot of viewers. To that end, we are continuing with this investigation, and will present further results in an update of the paper and at the XML 2000 Europe.

References

[1] ISO/IEC 8632-1, -3, -4:1999

[2] W3C WebCGM Recommendation
http://www.w3.org/TR/REC-Where??? - TBD
[3] W3C Scalable Vector Graphics Working Draft
http://www.w3.org/TR/WD-SVG/
[4] W3C CSS Level 2 Recommendation, 12 May 1998
http://www.w3.org/TR/REC-CSS2/
[5] W3C DOM Level 1 Recommendation, 1 October 1998
http://www.w3.org/TR/REC-DOM-Level-1/
[6] W3C DOM Level 2 Working Draft, 4 March 1999
http://www.w3.org/TR/WD-DOM-Level-2/
[7] W3C DOM Level 2 Working Draft, Ch. 4. Document Object Model CSS
http://www.w3.org/TR/WD-DOM-Level-2/css.html
[8] W3C DOM Level 2 Working Draft, Ch. 5. Document Object Model Events
http://www.w3.org/TR/WD-DOM-Level-2/events.html
[9] Dieter Weidenbrück: Manipulation of properties of CGM objects in viewing environments
Conference proceedings of XML 99, Philadelphia
[10] da Ponte/Goertz/Henderson: Implementing a viable architecture for standardized intelligent graphics
Conference proceedings of XML Europe 99, Granada