Dynamic conversion between XML-based languages for vector graphics

Angelo Di Iorio¹, Fabio Vitali¹ and Gianluca Zonta¹

¹University of Bologna, Mura Anteo Zamboni 7, 40100 Bologna, Italy

ABSTRACT

Vector graphics is increasingly gaining importance within the Word Wide Web community, because it allows users to create images that are easily manageable, modifiable and understandable. Two formats play a leading role among the languages for vector graphics: SVG and VML. Achieving a complete interoperability between these two languages means providing users a complete support for vector images across implementations, operating systems and media. In this paper we describe VectorConverter, a tool developed at the University of Bologna that allows easy, automatic and reasonably good conversion between two vector graphic formats, SVG and VML, and one raster format, GIF. This tool makes good translations between languages with very different functionalities and expressivity, by applying translation rules, approximation and heuristics. A high-level discussion about implementation details, open issues and future developments of VectorConverter is provided as well.

Keywords: SVG, VML, VectorConverter, image converter, XSLT

1 INTRODUCTION

The dichotomy between raster and vector graphics has been deeply discussed among researchers and professionals. Today all users agree that it is impossible to select the best approach, once and for all: both kinds of graphic formats, in fact, are particularly suited to some contexts while are inappropriate to others.

A raster image is a grid of pixels where the color of each pixel is individually defined. Well-known standards like JPEG, GIF and PNG have shown advantages and limits of this approach. Raster images assure high resolution with accurate colors and photographic quality and are widely supported on the Web. On the other hand, they have some evident drawbacks: first of all, they are not scalable (when increasing or reducing their dimension, the quality is compromised); secondly, their internal code is not human-readable and modifiable, so that little changes force users to re-generate the whole image by scratch. Finally, a high-quality raster image has a large byte size, a key factor when transmitting it on slow internet connections.

On the contrary, vector graphics uses mathematical functions to draw shapes and lines, so that scalability, small dimensions and small footprint are guaranteed. Moreover these images are text-based and easily modifiable whenever vector information is expressed through a human-readable language such as XML. Although vector images seem to be more manageable, sizeable and portable, raster images are today by far the most frequent kind on the Web. This is mostly due to the fact that although standards for raster graphics exist that can be reliably used for publishing on the web, no comparably successful standard exist for vector graphics.

Actually different vector languages have been proposed in the last years but none of them has really been successful as its raster-oriented cousins. PGML (Precision Graphics Markup Language) [3] is an XML language proposed by Adobe and IBM on April 1998, based on the imaging model of postscript and PDF. Subsequently, new functionalities have been added towards a complete integration with other web languages and applications. VML (Vector Markup Language) [18] was proposed by Microsoft and Macromedia in May 1998. Similar to HTML as to internal structure and syntax, it relies on CSS for the definition of presentational aspects. VML documents do not actually exist as stand-alone resources, but they have to be coded within an HTML document, through elements and attributes belonging to a given namespace.

The rivalry between PGML and VML has been tackled by the W3C in the 1998, when an initiative started to define a new format partially based on both proposals: SVG [11]. SVG strengthens some existing functionalities and offers many additional features such as animations, advanced management of text fragments and so on. With SVG, even
accessibility requirements are considered, for the first time in the graphic community: metainformation can be added to the code in order to describe the graphical elements and give alternative descriptions for disabled users. The development of SVG has not stopped yet: SVG 1.1 [12] provides modularization, Mobile SVG Profiles [8] customizes the language for mobile devices and SVG 1.2 [14] promises to add new interesting functionalities.

Among the others proposals, only VML seems to be a strong antagonist of SVG: being substantially a Microsoft product directly usable with Internet Explorer and MS Office documents, VML is still used in some contexts and, in some cases, more successful than SVG. On the contrary, PGML is now (quite) completely dismissed.

Interoperability between SVG and VML is therefore important for allowing complete support for vector graphics across implementations, operating systems and media types. VML images can be created with little effort by using Microsoft tools (images editors are integrated within MS Word or MS Power Point, too) but require proprietary viewers; similarly SVG, even if published as an public standard, requires specific tools to display images: a bidirectional and automatic conversion would allow users to draw and display images regardless of their original format. Moreover average users can use well-known applications, such as MS Word or MS Power Point, to create SVG images without the need to master complex graphic tools or technologies.

A direct conversion further simplifies and speeds up the re-use of existing material. Using legacy VML images in an SVG-based environment, today, is a manual and difficult task, in particular for inexpert users. On the contrary, more quick and correct results can be achieved by an automatic converter, in charge of performing translations without manual user interventions.

Even conversions between raster and vector graphics are worth further investigations. Producing raster from vector images is really useful to make up for the inadequate support of vector images implemented in current applications: VML has the same diffusion problems of any other proprietary language, and even SVG, although standard is not yet widely supported. The inverse conversion, from raster to vector, has evident and undeniable benefits but it is still far from being fully feasible.

In this paper we present VectorConverter, a tool that allows easy and automatic conversion between SVG and VML and a raster format, GIF. This tool makes good translations between the vector languages despite their substantial different functionalities and expressivity, and can also create, when needed, raster images for unsupported features: actually six conversions are performed, back and forth each of these formats (although the conversion from GIF to SVG or VML is really just an inclusion, as we will explain in section 3.2).

The rest of the paper is structured as follows: section 2 describes existing tools to edit, convert and manage images in XML-based vector graphics; section 3 describes the conversions performed by VectorConverter, moving off of the analysis of differences and similarities among the languages; section 4 and conclusions discuss open issues, limitations of the current implementation and future developments.

2 VIEWING, EDITING AND MANAGING VECTOR GRAPHICS

Although the vast majority of web images are raster, vector graphics is increasingly gaining importance for the WWW. While VML is still readable only by Microsoft applications (recent versions of Internet Explorer include a VML renderer), most browsers read SVG documents though specific plug-ins, such as Adobe SVG Viewer [1] for Internet Explorer (and Netscape) or KSVG [6] for Konqueror, or through a native engine as Mozilla and Firefox do [20].

Creating and editing vector images is a bit more complex issue. VML documents can be created only with Microsoft applications such as MS Office components (Word, PowerPoint and MS Excel) or VML Generator [19]. As expected, the support for SVG is wider: Amaya [25], a browser by W3C that allows users to modify pages directly during the navigation, includes an SVG editor; Jasc WebDraw [10] is an open-source authoring tool with a simple and usable interface; Sodipodi [16] is a cross-platform and open-source drawing program running on Linux, Windows and Mac OS. Commercial graphic tools, such as Adobe Illustrator [2], export SVG images with very good results too.

Among SVG tools, a leading role is surely played by Batik [9], a Java application that allows users to view, generate and manipulate images. Composed by different modules, Batik can be integrated in other applications: many viewers use the Batik engine to display, rotate and zoom images, as well as to export JPEG, PNG or TIFF images, as well as to convert TrueType fonts into SVG.
SVG and VML images are very often produced by converting documents and images too. SVGMaker [24] and VMLMaker [23] are MS Windows applications that convert any printable resource into SVG following a common approach: each classified part of the input (text or image) is translated into a vector element while the rest is included as a raster fragment. Actually SVGMaker proves to achieve much better results, above all on MS Office, PDF or CAD files. CR2V [22] is a C++ tool that transforms digital photos in vector drawings, exclusively composed by path elements. [7] converts raster images into SVG files, to be used in storing, processing, and delivering of geographic data. SVG Converters have also been developed to transform business XML data for their presentation to the end user [15], to simplify the representation of technical graphics and data [13] and to derive SVG resources from PostScript and Flash files [21].

3 VECTORCONVERTER

VectorConverter is a server-side application that allows easy and automatic translations between images in three supported graphic formats: SVG, VML and GIF. Currently a simple web interface is provided to the users, but VectorConverter can be integrated in other applications with little effort. The actual conversions are implemented through different technologies: XSLT transformations are performed back and forth the vector formats, while PHP scripts execute the conversions into (and from) GIF format. The following picture summarizes the VectorConverter transformations:

![Figure 1: Conversions performed by VectorConverter](image)

3.1 Conversion between SVG and VML

SVG and VML share a lot of features. After all, SVG was initially developed from VML, and both of them face the same issues. A detailed comparison between these two languages is out of the scope of this paper, but a high-level description is useful to understand how the converter actually works.

SVG and VML have many names of elements syntactically identical (like rect, ellipse, path, and so on), they have similar mechanisms to define position and dimension of the objects (attributes `viewbox` in SVG, and `coordorigin` and `coordsize` for VML) and several common attributes and properties. On the other hand, they have relevant syntactical and structural differences: for instance, SVG uses different attributes for style properties, while VML surrounds them in a single `style` attribute, and SVG requires a `svg` tag for any graphical object while VML uses different elements.

More important, both of them have some features unmanaged by the other. For instance, VML allows users to define the order of figures, to write text in HTML syntax and simplifies arcs and curves definitions; on the other hand, SVG defines particular graphical effects (filter, clipping, and so on), animation, advanced transformations and provides a really complete management of text fragments. On the whole, VML is a much less sophisticated language than SVG, whose documents are less verbose and easier to be written and updated. For these reasons, a full conversion between these languages is practically impossible.
However VectorConverter achieves good result, by adopting a “semantic” approach rather than a “syntactical” one: each construct of the input language is not necessarily converted into a syntactically equivalent construct of the output, but it is mapped into an element that assures the most similar graphical effect. The inner code of the output image can be very different from the input code, but the two images are visually equivalent, except for insoluble or unsupported incompatibilities.

VectorConverter is simply composed by two XSLT that respectively convert an SVG file into a VML, and vice versa. The choice of XSLT as a means to produce SVG files is not a novelty: [17] applies XSLT on structured data to create graphs, maps or schematic diagrams and [4] produces SVG histograms and diagram graphs to make scientific data more appealing. XSLT technologies are powerful, easy to be used, and widely supported by server-side and client-side languages. Being based on XSLT, VectorConverter is completely cross-platform and integrable in publishing packages that handle both images and text, for both web and print rendering.

3.1.1 From SVG to VML:

A full conversion from SVG to VML cannot be performed because of the sophistication of the first language in respect to VML. However, good results have been achieved by VectorConverter, by applying heuristics and approximations. A number of translations are simple and direct and consists of changing names of elements and attributes, while others require a more complex management. Let us briefly explain how VectorConverter works, with a simple example: the SVG code fragment in fig. 2 draws a (red-filled and blue-stroked) rectangle and a multi-colored text saying “You are tall”. Fig. 3 shows the VML conversion of that fragment.

Converting the root of the documents is the first issue to be addressed: SVG requires that each image is included within an `svg` element, which fixes the dimension and the coordinate system of that image (through the `viewBox` attribute); VML does not impose this constraint and allow different elements to be root of the image. VectorConverter converts the SVG root element into a VML `group` element, which plays an equivalent role, through the `width` and `height` properties of the `style` attribute, and the attribute `coordsize`. After setting the canvas of the VML output image, the converter goes on in translating individually each element.

The conversion of the rectangle is a clear example of direct mapping from a SVG to VML: the element `rect` has the same name in both of the languages, the coordinates and dimensions are transformed from SVG attributes to properties of the `style` attribute in VML and the `fill` and `stroke` attributes are transformed into corresponding elements (children of `rect`).
The management of the text fragment, on the contrary, is an example of complex and not straightforward translations. SVG allows users to create text fragments within text elements with a different position and formatting (in the example the substring “tall” is not aligned with “You’re” but moved up), through the `tspan` element. VML does not support such advanced text management and allows users to only add text as HTML fragment or within a path. The only possible solution is converting a `tspan` element in something that gives a similar graphical effect: VectorConverter creates a `path` element to define position and direction of the text and, then, include the actual content into a `textpath` element following that `path`. A similar approach is adopted in converting all the other features of SVG unsupported by VML: each feature has been individually studied and managed by an indirect but enough precise conversion. A full example of conversion from SVG to VML is shown in fig. 4:

![Conversion from SVG to VML](image)

Figure 4: Examples of conversion from SVG to VML

### 3.1.2 From VML to SVG

VectorConverter achieves better results in the inverse transformations, from VML to SVG: VML is less sophisticated and complete than its antagonist and each construct can be, more or less directly, translated into SVG. Fig. 5 shows an example of these high-quality conversions:

![Conversion from VML to SVG](image)

Figure 5: Examples of conversion from VML to SVG
VectorConverter addresses issues related to the root element (which changes in VML, but has to be a **svg** element in SVG) with a basic solution: the output has always a general-purpose **svg** root, which contains the actual converted image. The examples of code in the previous section have shown another important difference between VML and SVG: the management of the style and formatting properties. VML expresses information about positions, dimensions, colors and fonts within the attribute **style**, while SVG has a specific attribute, for each property. For this reason, VectorConverter scans each **style** attribute in the source, extracts properties’ values and produces the corresponding SVG objects. Some names have to be changed but the conversion is substantially immediate.

Other elements require indirect and more complex transformations: the compounded text fragments are a clear example of such issue. VML, in fact, allows users to include text fragment written in HTML syntax, while SVG does not support HTML tags. For this reason, VectorConverter maps each HTML tag into a generic **text** element containing a **tspan** for each component: for instance, a **div** containing paragraphs which, in turn, contain in-lines is transformed into a **text** element containing nested **tspan**, which specify proprieties of each paragraph and in-line. Fig. 6 and 7 show a simple HTML fragment within a VML image and its conversion in SVG.

![Fig 6: A simple VML fragment](image1)

![Fig 7: The SVG conversion of the VML fragment in figure 6.](image2)

### 3.2 Conversions between vector formats and GIF

Although VectorConverter was initially designed to transform SVG images back and forth VML, the application allows users to produce also raster images from vector ones, in particular GIF files. Actually, the conversion is performed by exploiting an external graphic library, GD [5], an open-source set of functions for the dynamic creation of images by programmers. Even if the core of GD was developed in C, several wrappers exist that allow programmers to include GD in PHP, Perl or other languages applications. VectorConverter is written in PHP and takes in input an SVG or VML file to produce an equivalent GIF image, invoking GD methods. Since GD can create also PNG and JPEG images, we reckon extending the application to other raster formats with little effort.

Three phases can be identified during the conversion process: (i) parsing the source and, for each element, (ii) drawing a corresponding shape and (iii) adding formatting properties. During parsing, VectorConverter collects all the required information about the image, in particular positions, dimensions and colors; then, invokes GD methods to draw each shape with its specific graphical properties. Most information can be directly read from a single source element: for instance, an SVG **rect** element has attributes or sub-elements which indicate its position, width, height or stroking/filling colors as well as the elements **path**, **circle**, **text**, **tspan** and so on. Similarly VML elements use the **style** attribute to express the same properties.

However two complex issues needed to be addressed: inheritance and coordinates. In VML or SVG, the properties of an element can be either directly expressed with an attribute or inherited by the element’s ancestors. For this reason, it is...
not enough to read static attributes but values have to be computed dynamically: to achieve this goal, VectorConverter exploits internal data structures, i.e. stacks, and keeps values of attributes and nesting levels so that relevant information are always available. Similarly, SVG and VML allow users to define a new coordinates system, so that elements’ position and dimension are relative to that customized system: VectorConverter computes the effective position and dimension of each component, taking into account all these relative data. Fig. 8 shows a SVG image and its corresponding translation in GIF format:

Fig 8: Examples of conversion from SVG into GIF

The support of VectorConverter for raster images is completed by a transformer of GIF images into VML or SVG files. Actually, raster images are not converted into vector ones but simply included in their source. SVG and VML, in fact, allow users to include references to external (and raster) images, which will be displayed within the original image. Yet, this side of conversion is relatively useful today, but we plan to strengthen it in future developments of the prototype.

4 OPEN ISSUES

Although the results obtained with VectorConverter are overall satisfactory, the tool is not yet complete. This depends on a number of factors: first of all, this release does not consider some features and does not recognize some properties of these two languages, even if all the important constructs are already supported. Secondly, some functionalities are recognized but remain intrinsically impossible to convert because they are available in just one or the other language or because they currently not managed by the PHP graphic library. Finally, some features are converted with partial and approximated results: VectorConverter applies heuristics and solutions that simulate the behavior of the input file but it is sometimes difficult to obtain the same effect in the output. A more detailed, even if incomplete, discussion about open issues on each conversion is necessary to outline these limitations and to make clear the potential and soundness of VectorConverter.

Let us start from the conversion that achieves the most correct and satisfactory results: from VML to SVG. Some attributes are not yet recognized: z-index, which specifies the overlapping rules of positioned objects, as well as shadow, which specifies shadowing effects, and other advanced text properties. Elements formulas, which defines mathematical formulas computed on attributes’ values, are not managed as well as some HTML tags: as mentioned before, VML allows users to include HTML tags into images but, even if the most common tags are converted, such as div, p, a, b, i and so on, some of them are still unsupported by VectorConverter. The management of paths and text following paths is another aspect to surely work on: the commands of VML to draw arcs, lines and part of circles are approximated with SVG lines but the output is not yet completely equivalent to the input, as evident in fig. 5.

Even the conversion from SVG to VML exhibits some limitations. Some features are not converted because they are not available in VML: the attribute PreserveAspectRatio, which indicates whether or not to force uniform scaling, as well as the commands for clipping, i.e. hiding something what normally would be drawn. Other features are still
unsupported, even if they would be partially available (or indirectly feasible) with VML: animations, filters, graphical effects like shadowing or text modifications, such as orientation. Moreover, approximations from SVG to VML are necessary in the management of text and paths but should be polished, as shown by the examples in fig. 4.

Although not yet completed, VectorConverter converts the fundamental set of objects and attributes from and to VML and SVG with very good results. On the contrary, the conversion from vector formats into GIF has still some very important limitations. The problem is not simple to be solved, since the main limitations depend on the included graphic libraries that do not handle some characteristics. On the other hand, the GD library is the most used and reliable solution to draw images with PHP scripts, as far as we know. Waiting for new releases of GD, or other so powerful graphic libraries, the only viable solution to extract acceptable raster images from input vector ones is applying approximations and indirect conversions.

The most relevant unsupported features in VectorConverter are: gradients, color opacity, rotations and text following paths. They are partially translated by VectorConverter but need to be improved: gradients are substituted with a single color, opacity is ignored as well as rotations, and the text fragments following complex paths are converted into horizontal strings starting from the same point of the path. The management of borders is particularly meaningful: PHP functions do not allow programmers to define the border’s width, so that VectorConverter draws a set of lines, with increasing dimension, in order to simulate the visual effect of a border. Similarly, paths and arcs are rendered through poly-lines with a good approximation, but much better results can be achieved by polishing this drawing process.

Another point is worth to be emphasized about the conversion into GIF format: all the elements and attributes unsupported by the conversion between SVG and VML are not converted. Even if it would be technically possible and simple, we preferred a conversion fully consistent with the others in order to have an engine that handles the same subset of elements in any format.

5 CONCLUSIONS

A careful analysis of the languages for raster and vector graphics shows that converting images in different languages is not a simple task: a direct and complete conversion between VML and SVG images is quite impossible because of the intrinsic differences between these languages and, similarly, transformation of raster into vector images (and vice versa) is neither simple nor infallible. However, very good results proved to be achievable in both these contexts, through a mixed approach: on the one hand, some features can be directly mapped from the input into the output language; on the other hand, some workaround, based on approximations and heuristics, can be found to indirectly translate the unsupported features.

In this paper we presented VectorConverter, a server-side application that performs automatic conversion between SVG, VML and GIF by exploiting such approach. We firstly investigated the most relevant similarities and, above all, differences between the languages and then discussed some implementation choices of VectorConverter. The application is composed of a few XSLT files to manage the conversion between vector formats, and some PHP scripts that work on raster images.

Although experimental results are quite satisfactory, some issues are still open. We plan to go on in developing VectorConverter and to follow two research directions: (i) providing workarounds to deal with the characteristics that are currently unimplemented and (ii) polishing the approximated translations. Moreover, we plan to support more raster formats, such as JPEG, PNG, etc. A further step will consist in integrating VectorConverter in external applications, such as content management systems, publishing systems, web portals, documents converters and so on. Currently a simple interface to test the prototype can be found at http://vitali.web.cs.unibo.it/Progetti/VectorConverter.

6 REFERENCES