

XIII Konferencja PLOUG
Kościelisko
Październik 2007

SVG – wizualizacje w XML

Tomasz Traczyk
Politechnika Warszawska

ttraczyk@ia.pw.edu.pl

Abstrakt. Bardzo często do efektywnej prezentacji danych wykorzystujemy grafikę. Gdy wystarczą po prostu wykresy, użyć można gotowych narzędzi czy bibliotek. Gdy jednak chcielibyśmy nasze dane przedstawić w mniej typowy sposób, przydać może się SVG. *Scalable Vector Graphics* jest standardem opisu dwuwymiarowej grafiki wektorowej w języku XML. Pozwala tworzyć skalowalne rysunki i włączać je do dokumentów, np. do stron HTML. Gdy do SVG dodać język ECMAScript, można tworzyć grafikę interaktywną. Zaś gdy połączymy to z XSLT lub XQuery oraz z możliwościami pozyskiwania informacji z baz danych w postaci XML, otrzymamy potężne narzędzie, pozwalające na podstawie danych z bazy tworzyć złożone wizualizacje. Referat prezentuje podstawy SVG, pokazuje jak stworzyć grafikę interaktywną, a także w jaki sposób za pomocą ogólnie dostępnych narzędzi tworzyć wizualizacje danych.

Informacja o autorze. Autor jest adiunktem na Politechnice Warszawskiej. Specjalizuje się w zastosowaniach baz danych i języka XML.

1. Po co SVG?

1.1. Grafika wektorowa

Dominująca w Internecie grafika rastrowa nadaje się dobrze do reprezentowania zdjęć, ale użycie jej do rysunków, napisów czy wykresów jest mało efektywne. Rozmiar plików graficznych jest duży, ale przede wszystkim przesyłany obraz nie daje się skalować bez wyraźnej straty jakości. W sposób oczywisty właściwym rozwiązaniem do reprezentowania obrazów o charakterze rysunków jest grafika wektorowa. Pozwala ona na bezstratne skalowanie obrazu oraz na zachowanie podziału rysunku na części składowe, co umożliwi jego dalsze przetwarzanie, animacje itd.

Zastosowanie języka XML do reprezentowania grafiki wektorowej daje dalsze ważne zalety. Jest to format tekstowy, zatem łatwy do tworzenia (w najprostszych przypadkach wystarczy edytor tekstu), interpretacji i do dalszego przetwarzania. Uzyskuje się także możliwości:

- zapisu metadanych opisujących obraz i jego poszczególne części;
- wyodrębniania poszczególnych części obrazu i manipulowania nimi;
- tworzenia oszczędnych w zapisie animacji – animować można poszczególne części rysunku;
- tworzenia grafiki interaktywnej, reagującej na działania oglądającego;
- generowania i przetwarzania obrazu za pomocą ogólnie uznanych narzędzi XML-owych, np. DOM, XSLT, XQuery.

W dodatku na ogół pliki z grafiką wektorową są znacznie mniejsze od reprezentacji rastrowej, co ma istotne znaczenie w przypadku elementów stron internetowych. Wadą jest większe zużycie mocy obliczeniowej procesora, niezbędne do wyświetlenia obrazu, ale przy możliwościach współczesnych procesorów nie ma to większego znaczenia.

1.2. Język SVG

Niestety, do niedawna przeszkodę w upowszechnieniu grafiki wektorowej stanowił brak odpowiedniego powszechnie akceptowanego formatu zapisu. Od dawna wprowadzono różne formaty grafiki wektorowej (np. CDR, EPS), ale były one trudne do przetwarzania, w dodatku z reguły stanowiły też własność jakiejś firmy. Jednak gdy pojawił się język XML, zauważono, że może on być doskonałym fundamentem do zbudowania języka opisu grafiki wektorowej. Pierwsze próby zbudowania takiego języka nastąpiły prędko. Najbardziej znaną jest zapewne VML (*Vector Markup Language*), propozycja m.in. firm Microsoft i Macromedia, która wprowadzono nie stała się ogólnie uznanym standardem, ale wsparcie dla niej wbudowano do powszechnie używanych programów, w szczególności do MSIE od wersji 5. Konkurencyjna propozycja pod nazwą PGML (*Precision Graphics Markup Language*) powstała z udziałem m.in. firm Adobe i Sun. *World Wide Web Consortium* (W3C) na podstawie tych propozycji opracowało nowy otwarty standard SVG.

SVG (*Scalable Vector Graphics*) jest zatem bazującym na XML językiem opisu dwuwymiarowej grafiki wektorowej, stanowiącym standard zarządzany przez W3C. Pierwsza wersja standardu SVG pojawiła się w roku 2001 [W3C01], a w roku 2003 ukazała się obecnie obowiązująca wersja 1.1 [W3C03a]. Trwają prace nad wersją 1.2 [W3C05a, W3C07a].

Oprócz pełnej wersji języka, przeznaczonej do przetwarzania na „pełnowymiarowych” komputerach, zaproponowano także uproszczone wersje dla urządzeń przenośnych. Język Mobile

SVG [W3C03b] tworzony jest w dwóch odmianach: SVG Tiny [W3C06] dla telefonów komórkowych i SVG Basic dla urządzeń typu palmtop.

Język SVG powstawał głównie z myślą o grafice wyświetlanej na ekranie, np. na stronach internetowych. Zauważono jednak, że wydruk grafiki, a zwłaszcza włączanie jej do profesjonalnych publikacji drukowanych, wymaga dodatkowych środków, zapewniających możliwość użycia drukarskich przestrzeni bar (np. CMYK), dokładnego skalowania rysunków (podane wymiary w różnych jednostkach długości muszą być dokładnie odwzorowane) itp. Trzeba też w jakiś sposób odwzorować statycznie elementy animowane. Wreszcie przy wydrukach rozwiązać trzeba problem ewentualnego podziału wydruku na strony. Tych problemów dotyczy tworzona obecnie specyfikacja SVG Print [W3C07b].

1.3. Zastosowania SVG

Grafika wektorowa w zapisie SVG znajduje liczne ważne zastosowania, wśród których warto wymienić:

- tworzenie graficznych elementów stron WWW, zwłaszcza elementów animowanych i interaktywnych oraz ozdobnych napisów (także napisów animowanych);
- tworzenie map, planów, schematów itp. – zarówno na stronach WWW, jak i w aplikacjach;
- typowe profesjonalne zastosowania grafiki wektorowej, jak CAD czy GIS;
- zastosowanie do wyświetlania grafiki (także animowanej) na urządzeniach mobilnych (w tym wypadku niewielka objętość plików ma zasadnicze znaczenie), jak telefony komórkowe, palmtopy, urządzenia nawigacji satelitarnej itp.

W przyszłości na bazie SVG powstanie zapewne otwarty język opisu stron, o zastosowaniach podobnych do Postscriptu czy PDF.

2. Podstawy SVG

SVG jest dialektem XML, dokument w SVG musi więc być poprawnym dokumentem XML. Znaczniki języka SVG znajdują się w przestrzeni nazw <http://www.w3.org/2000/svg>; niektóre instrukcje języka odwołują się także do przestrzeni <http://www.w3.org/1999/xhtml>. Dokument może się także odwoływać do publicznego DTD: `-//W3C//DTD SVG 1.1//EN`.

Element główny dokumentu tworzy znacznik `<svg>`. Oto przykładowy prościutki dokument SVG¹:

```
<?xml version="1.0"?>
<svg:svg version="1.1" width="50" height="50"
  xmlns:svg="http://www.w3.org/2000/svg">
  <svg:rect x="10" y="10" width="30" height="20"/>
  <svg:text x="25" y="40" text-
anchor="middle">SVG</svg:text>
</svg:svg>
```



Ponieważ znaczna część języka SVG dotyczy wyglądu, współdzielili on wiele rozwiązań z językami opisu stylu CSS i CSS2.

Grafika w SVG może być zapisywana w osobnych plikach, o zwyczajowym rozszerzeniu `.svg`; popularne programy umieją także odczytywać pliki SVG skompresowane za pomocą pro-

¹ Użyto tu prefiksu przestrzeni nazw SVG, by uwypuklić odwoływanie się do niej. W kolejnych przykładach przestrzeń nazw SVG będzie zdefiniowana jako domyślna, zatem prefiksu nie będzie.

gramu `gzip`; używa się wtedy rozszerzenia `.svgz`, a przy większych obrazach kompresja jest zazwyczaj bardzo efektywna. Typ MIME dokumentów SVG to `image/svg+xml`.

Grafikę w SVG można wykorzystywać w dokumentach w (X)HTML. By wstawić rysunek z osobnego pliku, użyć należy znacznika `<object>`. W dokumencie XHTML można teoretycznie po prostu zanurzać znacznik `<svg>` z zawartością, byleby zachować poprawność dokumentu w sensie *well-form*. Niestety, niektóre przeglądarki nie rozpoznają tak umieszczonego kodu SVG i trzeba użyć dodatkowych sztuczek, by spowodować jego wyświetlenie (patrz 4.1.).

Obrazy w SVG można także włączać do publikacji tworzonych za pomocą standardu XSL-FO, procesory tego języka (np. FOP) potrafią odpowiednio interpretować i wstawiać taką grafikę.

2.1. Tworzenie obrazów w SVG

2.1.1. Współrzędne i jednostki

Każdy element `<svg>` tworzy własny układ współrzędnych, może też podawać wymiary rysunku. Współrzędne rosną od lewego górnego rogu rysunku ku prawej i ku dołowi. Elementy `<svg>` można zagnieżdżać, tworząc lokalne układy współrzędnych.

Dostępne są liczne jednostki długości, znane z CSS, bezwzględne: `px` (domyślne), `pt`, `cm`, `in`, `ex`, `em`, i względne: procenty dostępnego obszaru.

Wielkość okna, w którym rysowany jest rysunek, określają atrybuty `width` i `height` znacznika `<svg>`. Dodatkowy atrybut `viewBox` pozwala zdefiniować wirtualny „otwór”, przez który oglądany jest rysunek. Odpowiednio manipulując rozmiarami i położeniem tego „otworu” można wyświetlić jedynie wybrany fragment rysunku (wyskalowany do wielkości okna znacznika `<svg>`), lub przeskalować rysunek tak, by zmieścił się w oknie (z zachowaniem proporcji lub bez – steruje tym atrybut `preserveAspectRatio`).

2.1.2. Kolejność rysowania

Poszczególne elementy graficzne są rysowane w takiej kolejności, w jakiej znajdują się w dokumencie SVG. Jeśli późniejszy element pokrywa się z wcześniejszym, to – o ile nie jest przezroczysty – przesłania go. Każda grupa jest najpierw rysowana w całości i jest w całości umieszczana na wynikowym rysunku.

2.2. Strukturalizacja

2.2.1. Grupowanie i definicje

Elementy graficzne można grupować, używając znacznika `<g>`. Grupy takie można zagnieżdżać. Cechy wizualne określone dla grupy są dziedziczone przez elementy zawarte w grupie.

Raz zdefiniowaną grupę można poddawać transformacjom, można też wywoływać ją wielokrotnie za pomocą znacznika `<use>`. Elementy i grupy przeznaczone do wielokrotnego wywołania można definiować wewnątrz znaczników `<defs>` – nie powoduje to wyświetlania, zdefiniowane elementy graficzne zostaną pokazane dopiero gdy zostaną wywołane. Do grupowania można także użyć znacznika `<symbol>` – nie powoduje on wyświetlenia zdefiniowanego fragmentu (trzeba go wywołać) i ma lokalny system współrzędnych.

2.2.2. Identyfikatory i referencje

Każdemu elementowi, w tym grupie, można nadać identyfikator za pomocą atrybutu `id`. Do identyfikatorów można odwoływać się używając atrybutu `xlink:href` z przestrzeni nazw języka

XLink. Każdy element może także mieć metadane: opisy w znacznikach `<title>` i `<desc>` oraz dowolne metadane, np. opis w RDF, w znaczniku `<metadata>`.

2.2.3. Włączanie obrazów

Do rysunku w SVG mogą być włączane obrazy zawarte w zewnętrznych plikach, zarówno zapisane w SVG jak i w formatach rastrowych. Służy to tego element `<image>`. Za pomocą tego elementu można także włączyć obraz rastrowy, zakodowany znakowo (np. w kodzie base64), wprost do dokumentu SVG.

2.3. Elementy graficzne

SVG definiuje instrukcje pozwalające rysować różne kształty – od bardzo prostych do niezwykle złożonych. Rysowanym elementom graficznym można nadawać cechy wizualne, określające kolory, obwiednie, grubość i inne cechy linii, wzór wypełnienia itd.

2.3.1. Proste figury

Podstawowe figury można rysować za pomocą dedykowanych instrukcji, podanych w tabeli 1.

2.3.2. Ścieżki

Bardziej złożone kształty rysuje się za pomocą kombinacji linii łamanych i krzywych – nazwa- no je łącznie ścieżkami. Do rysowania tych linii służy instrukcja `<path>`; jej atrybut `d` zawiera opis poszczególnych kroków rysowania linii (przykłady zamieszczono w tabeli 1). Opis ten składa się z liter oznaczających rodzaj kroku i następujących po nich parametrów liczbowych. `M` oznacza przesunięcie bez rysowania, `L` – rysowanie linii prostej, `H` i `V` – odpowiednio rysowanie linii prostej pionowo lub poziomo, `A` – rysowanie łuku eliptycznego, `Z` – domknięcie linii.

Bardziej złożone kształty rysuje się za pomocą krzywych Béziera stopnia 2-go (litera `Q`) lub 3-go (litera `C`). Można żądać, by kolejne fragmenty takiej krzywej były łączone w sposób gładki (litery odpowiednio `T` i `S`).

Poszczególne kroki mogą być wyznaczone przez współrzędne bezwzględne – oznacza się to przez użycie wielkich liter w opisie linii, lub względne – używa się wtedy liter małych.

Krzywe mogą mieć wypełnienie; zasady tworzenia wypełnień są dość złożone, ale użytkownik ma dobrą kontrolę nad tą czynnością.

Tabela 1. Podstawowe kształty w SVG² – przykłady

	Kod SVG
	<code><rect x="25" y="0" width="50" height="20"/></code>
	<code><circle cx="50" cy="20" r="15"/></code>
	<code><ellipse cx="50" cy="20" rx="30" ry="10"/></code>
	<code><line x1="30" y1="0" x2="60" y2="20"/></code>
	<code><polygon points="50,0 70,30 30,30"/></code>
	<code><polyline points="30,0 70,0 30,20 70,20"/></code>
	Ścieżka: linie proste; M – punkt startu, L – kolejne punkty „ruchu” <code><path d="M 30,0 L 70,10 L 30,20"/></code>
	Ścieżka: linie poziome, pionowe i zamknięcie krzywej <code><path d="M 30,0 H 70 V 20 Z"/></code>
	Krzywa Béziera 3 stopnia. <code><path d="M 10,20 C 10,0 50,0 50,20"/></code>
	Krzywa Béziera 3 stopnia; małe litery – położenie względne <code><path d="M 10,20 c 0,-20 40,-20 40,0"/></code>
	Połączenie dwóch krzywych Béziera 3-go stopnia <code><path d="M 10,20 C 10,0 50,0 50,20 C 50,20 70,40 90,20"/></code>
	Gładkie połączenie dwóch krzywych Béziera 3-go stopnia <code><path d="M 10,20 C 10,0 50,0 50,20 S 70,40 90,20"/></code>

2.4. Teksty

Rysunek oczywiście może zawierać teksty. Umieszcza się je za pomocą znacznika `<text>`. „Pudełko” z tekstem może być pozycjonowane względem początku, końca lub środka – steruje tym atrybut `text-anchor`. Tekst może być wypisywany wzdłuż krzywej, do tego służy znacznik `<text-path>`, zawierający referencję do elementu rysującego tę krzywą. Można też indywidualnie rozmieścić i obracać poszczególne litery tekstu.

Cechy wizualne tekstu (czcionkę, styl czcionki, kolor, wielkość, wytłuszczenie itp.) określa się podobnie, jak w języku CSS2. Możliwe jest wyróżnienie w tekście fragmentów (znacznik `<tspan>`) i indywidualne sterowanie położeniem i cechami wizualnymi poszczególnych fragmentów. Dla liter można też określać cechy wizualne jak dla figur, np. wypełnienie (domyślnie jednolicie czarne), przezroczystość czy obwiednię (domyślnie brak).

Teksty mogą oczywiście zawierać znaki narodowe (obsługiwane są także alfabety niełacińskie); kodowanie jest zgodne ze standardem XML, ale niektóre przeglądarki mogą nie wspierać mniej typowych stron kodowych (w tym wschodnioeuropejskich); zawsze jednak działa domyślne kodowanie UTF-8.

2.4.1. Czcionki

Przeglądarki SVG akceptują różne formaty czcionek wektorowych, m.in. TrueType, OpenType, Adobe Type1 oraz czcionki SVG. Używać można czcionek dostępnych w środowisku przeglądarki, w sieci Web (WebFonts z CSS2), oraz czcionek zdefiniowanych w SVG i osadzonych w dokumencie.

² Dla każdej figury zastosowano własną skalę osi y; użyto arkusza stylu – patrz 2.5.1.

Czcionki wektorowe to jedno z bardziej powszechnych zastosowań grafiki wektorowej, choć normalny użytkownik zwykle nie zdaje sobie z tego sprawy. Od typowych rysunków czcionki różnią się jednak sposobem skalowania: nie skalują się w pełni liniowo, ale dla skrajnych wielkości liter (zwykle bardzo małych) korzystają z dodatkowych informacji (tzw. *hints*), które nieco zmieniają kształt litery, by zachować czytelność. Litery muszą też być przypisane do swych kodów, a całe czcionki – opisane odpowiednimi atrybutami określającymi przynależność do odpowiedniej rodziny, stronę kodową itp.

W SVG istnieje element ``, umożliwiający definiowanie w pełni funkcjonalnych czcionek. Są też narzędzia umożliwiające konwersję czcionek wektorowych w typowych formatach na czcionki SVG.

2.5. Cechy wizualne

Cechy wizualne elementów graficznych ustala się określając ich odpowiednie właściwości. Cechy określone dla elementów nadrzędnych (np. grup) są, podobnie jak w CSS, dziedziczone przez elementy podrzędne (zagnieżdżone), ale mogą być lokalnie przykryte.

Znaczna część właściwości określających cechy wizualne pokrywa się z właściwościami znanymi z języków CSS i CSS2.

2.5.1. Style i atrybuty

Cechy wizualne elementu graficznego można określić używając wprost jego odpowiednich atrybutów. Można także użyć stylu i odpowiednie właściwości zapisać w języku CSS. Podobnie jak w przypadku HTML, można używać wewnętrznych arkuszy stylu (zawartych w znaczniku `<style>`), zewnętrznych arkuszy CSS (dołączanych instrukcją `<?xml-stylesheet?>`) oraz lokalnych atrybutów `style`. Sposób użycia języka CSS jest taki, jak zwykle dla XML-a: można używać nazw znaczników SVG, klas (definiowanych w SVG atrybutem `class`), odwołań do identyfikatorów, odwołań wg pozycji znacznika w dokumencie. Na przykład figury pokazane w tabeli 1 uzyskano, nadając im klasę `.figury` i stosując arkusz stylu o następującej zawartości:

```
.figury { stroke: black; stroke-width: 2; fill: gray }
```

Decyzja, czy do zdefiniowania danej cechy wizualnej użyć bezpośrednio atrybutu, czy stylu, nie powinna być całkiem arbitralna. Podobnie jak w dokumentach tekstowych, tak i w przypadku rysunków postuluje się bowiem oddzielenie treści od prezentacji; tyle że w tym przypadku mamy do czynienia z treścią graficzną, nie zawsze jest więc jasne, które cechy wizualne stanowią istotną treść przekazu, a które są tylko mogącą ulegać modyfikacjom dekoracją. Dobrze ilustruje to przykład obrazu ukazującego w czasie rzeczywistym sytuację w węźle kolejowym. Kolory przyporządkowane poszczególnym rodzajom torów są umowne i należą do sfery prezentacji, mogą więc być potraktowane jako styl. Natomiast konkretne kolory świateł na semaforach oczywiście odzwierciedlają istotną treść.

2.5.2. Linie i obwiednie

Wyglądem rysowanych linii sterują ich właściwości: `stroke` określa kolor, `stroke-width` – grubość; inne ważniejsze właściwości linii pokazano w tabeli 2.

Każda figura może być obwiedziona linią. Domyślnie grubość obwiedni jest zerowa dla wszystkich figur oprócz linii.

2.5.3. Przezroczystość i wypełnienia






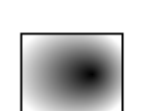
Dla każdego elementu graficznego można za pomocą właściwości `opacity` określić stopień przezroczystości. Elementy częściowo przezroczyste są widoczne, ale nie przesłaniają całkowicie obiektów pod sobą (patrz tabela 2).

Figura może być wypełniona jednolitym kolorem (domyślnie czarnym), gradientem liniowym lub radialnym (przykłady pokazano w tabeli 2) albo powtórzeniami wzoru wektorowego lub rastrowego („kafelkami”).

2.5.4. Kolory

Dla linii i wypełnień określa się kolory. Można używać kolorów z przestrzeni sRGB lub korzystać z nazw kolorów znanych z HTML, możliwe jest także użycie innych przestrzeni barwnych, definiowanych za pomocą profili ICC.

Tabela 2. Cechy wizualne – przykłady

	Zakończenie linii: <code>stroke-linecap="butt" / "square" / "round"</code>
	Połączenie linii: <code>stroke-linejoin="miter" / "round" / "bevel"</code>
	Wzór przerywania: <code>stroke-dasharray="9 4" / "6 3 2 3"</code>
	Przezroczystość: <code>stroke="gray" fill="lightgray" opacity="0.8"</code>
	Wypełnienie gradientem liniowym: <pre><linearGradient id="lgrad" x1="0%" y1="0%" x2="100%" y2="100%"> <stop offset="0%" stop-color="white" /> <stop offset="40%" stop-color="lightgray"/> <stop offset="100%" stop-color="black" /> </linearGradient></pre>
	Wypełnienie gradientem radialnym: <pre><radialGradient id="rgrad" cx="50%" cy="50%" r="60%" fx="70%" fy="50%"> <stop offset="0%" stop-color="black"/> <stop offset="100%" stop-color="white"/> </radialGradient></pre>

2.6. Przekształcenia

Zdefiniowane fragmenty rysunku (np. grupy) można poddawać różnorodnym przekształceniom.

2.6.1. Proste transformacje

Fragmenty rysunku można przesuwac, skalowac, obracac i skrecac. Transformacjami steruje atrybut `transform`, w którym opisuje się poszczególne operacje – przykłady podano w tabeli 3. Transformację można także opisać podając macierz przekształcenia.

2.6.2. Przycięcia i maskowanie

Rysunek można dowolnie przycięc, definiujac figurę (np. złożoną ścieżkę), która zostanie wykorzystana jako obwódka przycięcia. Obwódki taką definiuje się w znaczniku `<clip-path>`, a potem wywołuje atrybutem `clip-path` przycinanego elementu lub grupy.

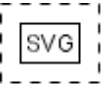


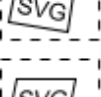
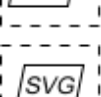

Przycięcie można porównac do nałożenia całkowicie nieprzezroczystej maski z wyciętym otworem. Maskowanie jest przysłanianiem delikatniejszym: jasność maski określa jej nieprzezroczystość: im ciemniejszy obszar na masce, tym mniej przezroczysta maska. Ciekawy przykład

stanowi nałożenie kolorowego zdjęcia jako maski na czarny prostokąt o tych samych wymiarach. Wynik tej operacji to po prostu czarno-biały negatyw!

2.6.3. Filtry

Filtry w SVG umożliwiają osiągnięcie różnych efektów „artystycznych”. Konstruuje się je z prymitywów, umożliwiających m.in. manipulowanie kolorami, ich nasyceniem, jasnością, kontrastem itp., łączenie obrazów (także rastrowych), rozmywanie i wyostrażanie, generowanie tekstur, symulowanie oświetlenia itp.

Tabela 3. Proste transformacje – przykłady

	Bez transformacji
	Przesunięcie: <code>transform="translate(5,5)"</code>
	Skalowanie: <code>transform="scale(1.3,0.7)"</code>
	Obrót: <code>transform="rotate(10)"</code>
	Pochylenie wg osi X: <code>transform="skewX(-10)"</code>
	Złożenie transformacji: <code>transform="skewX(-10) translate(4,0)"</code>

3. Animacje i grafika interaktywna w SVG

3.1. Animacje

W SVG można definiować obrazy animowane, są na to dwie metody:

- deklaratorywna – używający wewnątrz języka SVG części języka SMIL [W3C05b];
- proceduralna – gdzie animacje wywołuje się manipulując elementami graficznymi za pomocą języka ECMAScript (patrz 3.2.)

Użycie metody deklaratorywnej jest stosunkowo łatwe, animowane pliki mają też niemal niezmienioną wielkość w stosunku do odpowiedników bez animacji. Metoda proceduralna większe możliwości (np. można generować zdarzenia pseudolosowe) i może pozwalać na szybsze animacje, ale jest trudniejsza i bardziej podatna na błędy programowania. W obu metodach animacja jest wykonywana przez klienta – przeglądarkę.

W metodzie deklaratorywnej animacja polega na zmianie wartości atrybutów w czasie. W ten sposób można zmieniać kształt, położenie, wielkość i cechy wizualne (np. kolor, widoczność, przezroczystość czy cechy gradientu) obiektów. Znacznik `<animate>` pozwala określić zakres, prędkość i okresowość zmiany wartości; zmiany te mogą następować liniowo albo według podanej listy wartości: skokowo albo z interpolacją liniową lub kwadratową. Poszczególne elementy rysunku mogą być animowane jednocześnie lub kolejno: zakończenie jednej animacji może wyzwać start następnej. Animacja może być rozpoczynana lub zatrzymywana na skutek zdarzeń związanych z myszką. Animować można dowolne części rysunku, także czcionki SVG!

3.2. Grafika interaktywna

Prostą interaktywność uzyskać można wykorzystując zdarzenia myszki w animacji deklaratywnej, jak to opisano poprzednio. Bardziej złożone efekty wymagają programowania. W SVG można zanurzyć kod w języku ECMAScript³; robi się to podobnie jak w HTML, za pomocą znacznika `<script>` oraz atrybutów `script`. Skrypty mogą manipulować drzewem DOM dokumentu, zmieniając atrybuty poszczególnych elementów lub nawet strukturę drzewa.

Skrypty mogą być wywoływane przez zdarzenia. Lista zdarzeń jest nieco rozszerzona w stosunku do znanej z HTML, np. dołączono zdarzenia związane z przebiegiem animacji deklaratywnej. Metody umożliwiają m.in. odczytywanie bieżących współrzędnych kursora, znajdowanie korzenia drzewa DOM aktualnego obrazu, manipulacje przestrzeniami nazw, czy programowe sterowanie parametrami animacji deklaratywnej.

Niestety, różne przeglądarki bardzo różnie implementują zarówno język ECMAScript (JavaScript, JScript), jak i jego współdziałanie z grafiką SVG, pisanie kodu przenośnego może więc być dość trudne.

3.2.1. DOM dla dokumentów SVG

Drzewo DOM dokumentu SVG jest oczywiście normalnym drzewem XML-owym. W stosunku do ogólnego standardu DOM język SVG wprowadza jednak pewne rozszerzenia. Jeśli kod SVG jest zanurzony w dokumencie (X)HTML czy XML lub jeśli w dokumencie SVG zanurzony jest fragment z XML, drzewo DOM daje oczywiście dostęp do wszystkich elementów tego dokumentu, skrypty mogą zatem czytać dane z niegraficznych części dokumentu (np. z formularza HTML albo z „wyspy danych” w XML) i sterować grafiką na podstawie tych danych.

3.2.2. Łączniki

W dokumentach SVG można stosować łączniki. Służy do tego element `<a>`, różniący się jednak od tego z HTML, gdyż do adresowania używa się prostych łączników XLink. W SVG w ogóle przyjęto zasadę, że odwołania do dokumentów (lub fragmentów) są wyrażane w standardzie XLink, podobnie zapisuje się np. odwołania w elemencie `<use>` (patrz 2.2.2. i 5.1.3.).

4. Narzędzia

4.1. Przeglądarki

Grafikę zapisaną w SVG wyświetla się za pomocą przeglądarek WWW lub specjalizowanych przeglądarek SVG. Wsparcie dla standardu SVG w przeglądarkach jest niestety ciągle niewystarczające, ale notuje się znaczny postęp. W szczególności obsługę SVG wbudowano w przeglądarki z grupy Mozilla (w tym w program Firefox) i w program Opera. Nie ma jej w MSIE, ale powszechnie używany jest darmowy *plug-in* Adobe SVG Viewer. Jego popularna wersja 3.0 ma jednak spore braki, znacznie większą część standardu obejmuje mniej znana wersja 6 beta. Niestety, firma Adobe zapowiedziała ostatnio zaprzestanie dalszych prac nad tym produktem.

Wszystkie przeglądarki implementują tylko podzbiory standardu, w dodatku implementacje różnią się ważnymi szczegółami. Stworzenie dokumentu, który będzie prawidłowo wyświetlany w wielu przeglądarkach jest ciągle zadaniem trudnym.

MSIE z dodatkiem Adobe SVG Viewer nie rozpoznaje prawidłowo znaczników `<svg>` wstawionych do dokumentów XHTML; by spowodować wyświetlenie tak umieszczonej grafiki należy do dokumentu wstawić następujący tajemniczy fragment:

³ ECMAScript jest to ustandaryzowana odmiana języka JavaScript.

```
<object id="AdobeSVG" classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"/>
<?import namespace="svg" urn="http://www.w3.org/2000/svg"
  implementation="#AdobeSVG"?>
```

4.2. Narzędzia do tworzenia grafiki SVG

SVG jest formatem tekstowym, do wyprodukowania prostej grafiki wystarczy zatem edytor tekstu i przeglądarka. Bardziej złożone rysunki trudno byłoby jednak tworzyć w taki sposób. Na szczęście istnieją już dość liczne programy graficzne umożliwiające tworzenie grafiki w SVG. Najbardziej znanym z produktów darmowych jest program Inkscape – wektorowy edytor graficzny o znacznych możliwościach, który SVG używa jako podstawowego formatu zapisu. Znane komercyjne edytory grafiki wektorowej na ogół są wyposażone w możliwość eksportu, a niektóre także importu grafiki w standardzie SVG.

4.3. Narzędzia dla programistów

Programiści także potrzebują narzędzi ułatwiających produkowanie grafiki w SVG oraz umożliwiających wykorzystywanie takiej grafiki w aplikacjach. Najbardziej znanym pakietem oprogramowania tego typu jest Apache Batik – zbiór gotowych narzędzi i bibliotek wspomagających użycie SVG w programach pisanych w języku Java. Biblioteki pakietu Batik zawierają m.in. parser XML uwzględniający rozszerzenia modelu DOM właściwe dla SVG, interpreter języka ECMAScript, generator tworzący zapis w SVG na podstawie grafiki tworzonej przez aplikacje w Javie (z użyciem standardowej klasy `Graphics2D`), komponent Swing umożliwiający wyświetlanie grafiki w SVG, narzędzia programistyczne do konwersji grafiki SVG na formaty rastrowe. Dołączono też przykładowe, ale użyteczne narzędzia, wykonane z pomocą wyż. wym. bibliotek: przeglądarkę do plików SVG (z obsługą skryptów ECMAScript), program do konwersji SVG na formaty rastrowe, konwerter zamieniający czcionki TTF w czcionki SVG i narzędzie do drukowania grafiki w SVG.

5. Wizualizacje danych w SVG

SVG znacznie lepiej od formatów rastrowych nadaje się do wizualizacji danych. Wiele typowych wizualizacji danych jest bowiem rysunkami doskonale dającymi się zrealizować za pomocą grafiki wektorowej. Do przykładów należą:

- wykresy (tak użyto SVG np. w Oracle Application Express);
- synoptyki wyświetlające aktualny stan instalacji przemysłowych, sieci przesyłowych itp.;
- zastosowania typu GIS – mapy i plany z różnorodną dynamiczną treścią;
- zastosowania typu CAD – schematy, rysunki montażowe itp.

Przewaga SVG wynika nie tylko z dobrego dostosowania możliwości grafiki wektorowej do typowych potrzeb i ze skalowalności wynikowych rysunków, ale też z łatwości tworzenia złożonych interaktywnych wizualizacji. Wizualizacje danych można uzyskać na wiele sposobów, m.in.

- budując grafikę SVG programowo, za pomocą odpowiedniego API, np. używając pakietu Batik lub bibliotek dostarczających funkcje graficzne wyższego poziomu;
- generując tekst SVG bezpośrednio lub przez programowe tworzenie drzewa DOM;
- generując strony WWW w SVG (lub zawierające osadzony tekst SVG) za pomocą technologii takich jak JSP, PHP czy XSQL;

- przekształcając XML-owe źródła danych (styczne lub dynamiczne) za pomocą XSLT lub XQuery – skrypty mogą np. umieszczać odczytane dane w przygotowanych szablonach grafiki SVG.

Wszystkie te technologie pozwalają oczywiście na dynamiczne tworzenie grafiki SVG na podstawie informacji pobieranych z baz danych. Wygenerowane strony mogą zawierać skrypty w języku ECMAScript, zapewniające interaktywność grafiki. Możliwe jest też użycie w nich technologii Ajax.

5.1. Przykład

Pokażemy jak w SVG stworzyć prostą mapę pogody, ukazującą temperatury w wybranych rejonach kraju.

5.1.1. Szablon dokumentu SVG

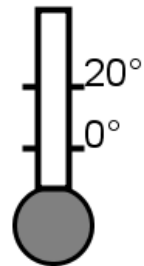
Najpierw przygotowujemy szablon dokumentu SVG. Zawierać on będzie element główny, definicje przestrzeni nazw, tło oraz definicję powtarzalnego symbolu.

Tłem wizualizacji będzie obraz konturów województw⁴, który ładujemy z pliku:

```
<image x="0" y="0" width="500" height="500" xlink:href="pl_map.png"/>
```

Następnie definiujemy wektorowy symbol termometru:

```
<symbol id="thermo">
  <circle class="thermo1" cx="50" cy="150" r="25"/>
  <line class="thermo1" x1="30" y1="100" x2="70" y2="100"/>
  <text x="70" y="100" font-size="19pt">0&#176;</text>
  <line class="thermo1" x1="30" y1="60" x2="70" y2="60"/>
  <text x="70" y="60" font-size="19pt">20&#176;</text>
  <rect class="thermo2" x="40" y="10" width="20" height="116"/>
</symbol>
```



5.1.2. Dane wejściowe

Przygotowujemy statyczny dokument XML, zawierający spis interesujących nas województw – ich nazwy i współrzędne na mapie konturowej:

```
<ex:województwa xmlns:ex="http://www.ploug.org.pl/examples">
  <ex:woj id="W" xy="330,180">mazowieckie</ex:woj>
  <ex:woj id="L" xy="410,270">lubelskie</ex:woj>
  ...
</ex:województwa>
```

Z bazy danych uzyskujemy (np. za pomocą technologii XSQL) aktualne dane na temat temperatur w postaci XML.

5.1.3. Scalanie informacji

Za pomocą skryptu XSLT łączymy szablon dokumentu SVG z danymi w taki sposób, że do wartości znacznika `<svg>` dołączamy wygenerowane na podstawie danych każdego województwa fragmenty o postaci:

```
<g id="W" transform="translate(330,180) scale(0.3333)"
  name="mazowieckie" temp="23">
  <use xlink:href="#thermo"/>
  <rect x="42" y="54" height="96" width="16" class="thermo3"
    onclick="interact(evt)"/>
  <text x="100" y="150" font-size="30pt" visibility="hidden"></text>
</g>
```

⁴ Obraz ten został przygotowany jako grafika wektorowa w SVG, ale skonwertowano go (za pomocą pakietu Batik) do postaci rastrowej, gdyż nie wszystkie przeglądarki potrafią wyświetlić grafikę wektorową ładowaną instrukcją `<image>`.

Każdy fragment tworzy grupę złożoną z wywołania symbolu termometru, prostokąta obrazującego słupek rtęci (skrypt XSLT oczywiście musi wyliczyć jego rozmiar i położenie na podstawie aktualnych danych) oraz napisu-zaślepki (będzie ona wykorzystana do interakcji). Cała grupa jest zmniejszana trzykrotnie i przesuwana na odpowiednie miejsce (skrypt XSLT wstawia współrzędne przesunięcia na podstawie danych województw).

Atrybuty `name` i `temp` nie są elementami języka SVG, zostały dodane jako rodzaj metadanych, przydatnych np. do tworzenia interakcji (patrz 5.1.4.).

Cały dokument łączymy z arkuszem stylu o zawartości:

```
.thermo1 {fill: gray; stroke: black; stroke-width: 4}
.thermo2 {fill: white; stroke: black; stroke-width: 4}
.thermo3 {fill: gray; stroke-width: 0}
```

za pomocą umieszczonej w prologu instrukcji przetwarzania:

```
<?xml-stylesheet type="text/css" href="mapa.css"?>
```

W wyniku tego scalania powstaje mapa z wyświetlonymi symbolami termometru, pokazującymi lokalne temperatury – patrz rys. 1.

Jeśli naszą wizualizację umieścimy na serwerze, w środowisku pozwalającym na wykonywanie skryptów XSLT, a dokument za temperaturami uzyskiwać będziemy z bazy danych na bieżąco, to nasza wizualizacja stanie się dynamiczna – będzie odzwierciedlać najnowsze dane.



Rys. 1. Wizualizacja temperatur: interaktywna grafika wektorowa na tle grafiki rastrowej

5.1.4. Elementy interakcji

Do wizualizacji dodamy prostą interakcję. Odpowiednie elementy graficzne „uczulamy” na kliknięcie myszką, nadając im atrybut `onclick`, wskazujący sposób obsługi zdarzenia (patrz przykład w 5.1.3.). W początkowej części dokumentu SVG dodajemy skrypt w języku ECMAScript, zawierający funkcję obsługującą zdarzenie:

```
<script type="text/ecmascript">
  function interact(evt) {
    svgobj = evt.target.parentNode();
    t = svgobj.getAttribute('temp');
    n = svgobj.getAttribute('name');
    e = svgobj.getElementsByTagName('text').item(0);
    e.firstChild.nodeValue = n + ' ' +t+'&#176;C';
    e.setAttribute('visibility', 'visible');
  }
</script>
```

Funkcja ta otrzymuje jako parametr obiekt definiujący zdarzenie. Odczytuje atrybuty `temp` i `name` ze znacznika `<g>`, nadrzędnego w stosunku do obiektu na którym kliknięto, wstawia odpowiedni tekst do napisu-zaśleпки i włącza jego wyświetlanie.

Kliknięcie w odpowiedni termometr spowoduje zatem wyświetlenie napisu zawierającego nazwę województwa i lokalną temperaturę, co pokazano na rys. 1.

6. Podsumowanie

SVG is the HTML for Graphics – takie było zamierzenie twórców języka SVG. Choć jeszcze sporo brakuje do jego realizacji, sam język w pełni zasługuje na uznanie. Implementacje SVG wciąż nie są doskonałe i znajdziemy je nie wszędzie gdzie być powinny, ale widać wyraźny postęp.

Sam standard SVG oferuje za darmo możliwości obrazowania, o których do niedawna tylko marzyć mogli nawet użytkownicy poważnych komercyjnych programów graficznych. Tekstowa postać zapisu obrazu pozwala go łatwo generować i przekształcać, można też w niej zapisać łatwe do odczytania metadane. Zgodność ze standardem XML umożliwia wykorzystanie licznych narzędzi XML-owych do operowania na danych graficznych.

Rostrastające się w błyskawicznym tempie światowe zasoby informacji w coraz większym stopniu są zasobami graficznymi. Duża część tej informacji graficznej jest składowana i przekazywana w postaci rastrowej, mimo że z natury jest rysunkiem: wykresy, napisy, mapy, schematy itp. Rastrowy zapis tych zasobów prowadzi do utraty istotnych informacji (np. nie da się już łatwo rozdzielić rysunku na poszczególne figury czy podzbiory) i nie daje szans na łatwe przekształcenia (choćby skalowanie). Coraz więcej jest też w Sieci stron animowanych czy interaktywnych. Zwykle są one wykonane w technologiach firmowych, jak Flash, co ogranicza możliwości ich wykorzystania. Do reprezentowania takich zasobów SVG nadaje się idealnie; jego rozpowszechnienie może istotnie wpłynąć na sposób tworzenia i przechowywania stron WWW i innych zasobów informacji.

Z punktu widzenia informatyka – projektanta systemów informacyjnych, SVG daje wielkie możliwości stosunkowo łatwej, efektywnej i elastycznej graficznej prezentacji różnorodnych danych, można się więc spodziewać, że coraz więcej aplikacji będzie te możliwości wykorzystywać. SVG zatem może się już niedługo stać jednym z podstawowych standardów prezentacji i wymiany informacji.

Bibliografia

- Herman04 Herman I.: SVG Tutorial. <http://www.w3.org/Consortium/Offices/Presentations/SVG/>
[W3C01] Scalable Vector Graphics (SVG) 1.0 Specification. W3C Recommendation, September 2001
- [W3C03a] Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation, January 2003.
<http://www.w3.org/TR/2003/REC-SVG11-20030114/>

- [W3C03b] Mobile SVG Profiles: SVG Tiny and SVG Basic. W3C Recommendation, January 2003. <http://www.w3.org/TR/2003/REC-SVGMobile-20030114/>
- [W3C05a] Scalable Vector Graphics (SVG) Full 1.2 Specification. W3C Working Draft, April 2005. <http://www.w3.org/TR/2005/WD-SVG12-20050413/>
- [W3C05b] Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation, January 2005. <http://www.w3.org/TR/2005/REC-SMIL2-20050107/>
- [W3C06] Scalable Vector Graphics (SVG) Tiny 1.2 Specification. W3C Candidate Recommendation, August 2006. <http://www.w3.org/TR/2006/CR-SVGMobile12-20060810/>
- [W3C07a] SVG Filters 1.2. W3C Working Draft, May 2007. <http://www.w3.org/TR/2007/WD-SVGFilterPrimer12-20070501/>
- [W3C07b] SVG Print 1.2. W3C Working Draft, May 2007. <http://www.w3.org/TR/2007/WD-SVGPrintPrimer12-20070501/>