Utiliser des graphiques SVG avec ASP.NET

Sommaire

I. La norme SVG

SVG signifie *Scalable Vector Graphics*, soit **graphique vectoriel redimensionnable**. Il s'agit d'une norme W3C dont les spécifications peuvent être trouvées sur leur site **W3 - SVG**. Le W3C définit SVG comme des "graphiques XML pour le Web". Pour être exact, SVG permet de décrire de façon textuelle des dessins 2D vectoriels. Le texte suit le formalisme XML. Les premiers drafts de SVG ont été proposés en 1999, la norme actuelle étant de niveau 1.1.
Du point de vue du contenu visuel, SVG permet de définir :

- des formes vectorielles (courbes, traits, cercles...),
- des images,
- du texte.

Les objets graphiques SVG peuvent être *groupés*, *transformés* par des matrices (un peu comme sous GDI+), *filtrés* et *composés* dans d'autres objets. Les objets peuvent aussi se voir accrocher des *styles* **CSS**. Autre avantage de SVG, les graphiques créés peuvent être dynamiques et interactifs sachant que les animations peuvent être soit intégrées dans le fichier SVG soit pilotées par un langage de script externe. Une interface DOM complète SVG afin de pouvoir accéder à tous les éléments et tous les attributs. Les graphiques SVG sont redimensionnables sans perte de qualité puisqu'ils sont définis de façon vectorielle, les parties d'un graphique sont réutilisables et partageables pour créer d'autres graphiques, éventuellement par d'autres utilisateurs. L'adoption du formalisme XML permet d'utiliser tous les outils existants : parser, transformation, etc. Les fichiers SVG peuvent en outre être compressés pour diminuer encore plus leur taille. Bref, vous l'avez compris, SVG est une nouvelle norme ouverte de dessin pour le Web, très souple, dynamique, interactive, tout ce qui en fait une concurrente redoutable pour contrecarrer l'hégémonie de Flash sur ce créneau.

II. Les outils SVG

Dessiner correctement n'est déjà pas donné à tout le monde, alors s'il faut décrire une image par des ordres textuels cela peut être très fastidieux... Bien heureusement de nombreux outils permettent de créer des graphiques SVG de façon traditionnelle. C'est le cas par exemple de l'incontournable Adobe *Illustrator CS* dont le domaine de prédilection est à la base le dessin vectoriel. Ce logiciel est plutôt réservé aux infographistes qui l'ont depuis des années érigé en standard professionnel, aussi bien sur PC que des années avant sur Apple Macintosh.

Tous les infographistes possèdent Illustrator et si vous avez besoin d'un dessin correctement réalisé c'est de toute façon vers l'un de ces professionnels que vous vous tournerez. Toutefois il existe d'autres logiciels manipulant SVG, comme le projet gratuit *Sodipodi* que l'on trouve sur SourceForge ou *WebDwarf* tout aussi gratuit, sans oublier le visualiseur gratuit de Adobe qui doit être installé sur la machine cliente pour afficher les images SVG dans les pages Web.
Vous trouverez d'autres informations en cherchant "SVG" sous Google et en vous promenant sur le site français de SVG (encore un peu maigre) **SVG en français**.
Enfin n'oublions pas que le format de définition des graphiques SVG se plie parfaitement à la **génération d'images**, et de façon très simple. C'est donc un outil parfaitement taillé pour les développeurs qui, généralement, ne savent pas dessiner mais qui savent écrire du code capable de générer du texte XML. SVG devient alors un outil d'une grande souplesse et d'une grande simplicité de mise en œuvre pour, par exemple, générer des business graphics de type histogramme ou camembert sans avoir à calculer le fichier image sur le serveur et sans avoir à transmettre une image bitmap (lourde) ou un jpeg (de médiocre qualité à cause de la compression).

Comme vous le constater, bien que très jeune, la norme SVG dispose déjà de nombreux atouts et des mauvaises langues susurrent dans les couloirs que chez Macromedia (éditeur de Flash), des sueurs froides commencent à saisir certains cadres... Ajoutons à leurs inquiétudes l'apparition de convertisseurs Flash

vers SVG, ainsi que Mobile SVG, appelé **SVG-T** (T pour *Tiny*, petit) pour les unités mobiles comme les **Pocket PC** et les**smartphones**.

II. Et ASP.NET dans tout cela ?

Bien entendu SVG est un format graphique qui n'a rien à voir de spécial avec ASP.NET pas plus qu'avec d'autres technologies. Il reste donc tout à fait possible de créer et visualiser des images SVG sur un PC ou un Mac sans que cela ne serve au Web. Il reste tout aussi possible d'intégrer une image SVG dans une bête page HTML comme on affiche un GIF ou un PNG.
Certes, mais SVG se positionne comme une norme qui va *très largement* au-delà de ce que peut être PNG par exemple. SVG peut être utilisé de *façon similaire à Flash*, un outil largement répandu dans le monde du Web.

Dès lors on voit nettement (!) que la connexion entre SVG et ASP.NET semble naturelle, une alliance gagnante entre d'une part un outil graphique pour concevoir de belles interfaces et de l'autre une infrastructure de développement robuste pour le Web.
L'existence de SVG-T pour les unités mobiles et celle du Compact Framework côté .NET renforce encore ce qui semble bien être un mariage d'amour et de raison entre deux *technologies complémentaires* visant les mêmes types d'utilisateurs.

Nul doute donc que SVG sera de plus en plus utilisé par les développeurs et plus particulièrement par ceux créant des projets ASP.NET.
Alors comment se servir de SVG sous ASP.NET ? C'est ce que nous allons voir dans la suite de cet article.

IV. A quoi ressemble un dessin SVG ?

La figure 1 montre un dessin très simple réalisé sous Illustrator CS. Trois cercles de couleurs non opaques se recouvrant.

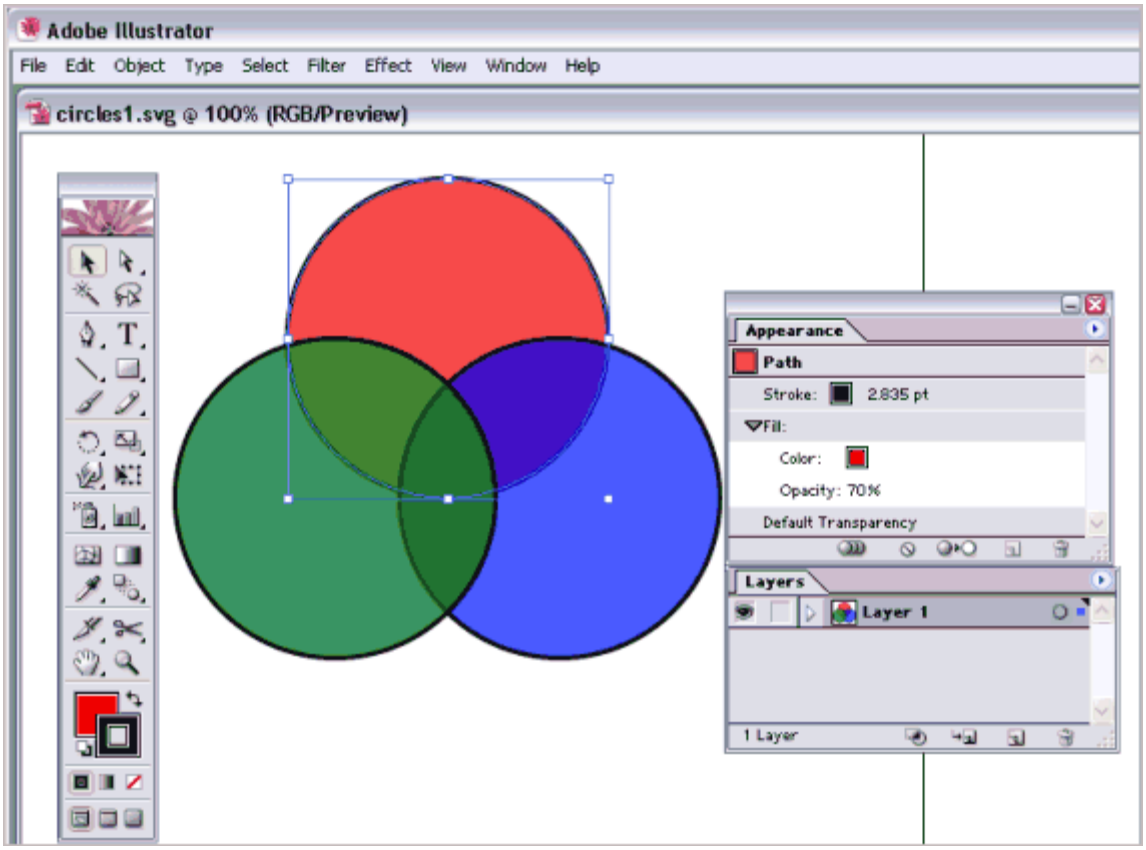

Figure 1 - Un dessin très simple sous Illustrator CS

Le fichier SVG résultant est le suivant :

Code 1 - Le contenu du fichier Circles1.svg

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/REC-SVG-
20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
<style type="text/css">
    circle:hover {fill-opacity:0.9;}
 </style>
 <g style="fill-opacity:0.7;">
 <circle cx="6.5cm" cy="2cm" r="100"
style="fill:red; stroke:black;
```

```
     stroke-width:0.1cm"
transform="translate(0,50)" />
 <circle cx="6.5cm" cy="2cm" r="100"
style="fill:blue; stroke:black;
     stroke-width:0.1cm"
transform="translate(70,150)" />
 <circle cx="6.5cm" cy="2cm" r="100"
style="fill:green; stroke:black;
     stroke-width:0.1cm"
transform="translate(-70,150)"/>
 </g>
</svg>
```

Même sans connaître les détails du format SVG, on peut facilement lire le code et comprendre que trois cercles de 6,5 cm sont dessinés avec un certain nombre d'attributs. Voici donc à quoi ressemble un fichier SVG, rien de bien compliqué, et c'est justement ce qui en fait une partie de l'intérêt.

V. Intégrer un SVG dans une page HTML

Commençons par une chose simple. Puisque SVG n'est pour l'instant pas un format image reconnu par les browsers, on ne peut placer une simple balise IMG dans le code HTML d'une page Web. On utilisera alors une balise **EMBED** qui permettra, en outre, de préciser le lien pour télécharger le plugin dans le cas où le client ne le possède pas.

Code 2 - Une page HTML affichant un SVG

```
<html>
 <body>
       <h1>Démo Graphique SVG</h1>
       <embed name="SVGDemo"

 pluginspage="http://www.adobe.com/svg/viewer/
install/"
       src="quadbezier1.svg" width="1198"
height="598"
```

```
        type="image/svg-xml" />
 </body>
</html>
```

Il est aussi possible d'utiliser la balise **OBJECT** à la place de EMBED et de préciser dans ce cas une image alternative, par exemple au format PNG, qui sera affichée dans le cas où le client ne puisse pas afficher le SVG (voir code 3).

Code 3 - Utilisation de la balise Object

```
<object data="quadbezier1.svg"
codetype="image/svg-xml"
style="width:1198;height:598;">
<img src="test2.png" width="1198" height="598"
/>
</object>
```

VI. Le plugin SVG de Adobe

Ce plugin est gratuit et existe pour la majorité des OS. D'autres plugins existent et existeront au fur et à mesure de la popularité croissante de SVG.
Pour télécharger le plugin (pour IE notamment) utiliser la page suivante **installation du viewer SVG Adobe**.
Pour tester la bonne installation du plugin il suffit d'afficher cette page **test d'installation du plugin Adobe**.
Il n'existe pas de setup pour FireFox pour l'instant et l'installation manuelle du plugin dans ce dernier semble causer des dysfonctionnements. Normalement la prochaine version de FireFox intègrera l'interprétation des SVG, cette version n'était pas encore disponible au moment de l'écriture de ces lignes.

VI. Insérer un graphique SVG dans une page ASP.NET

Il existe plusieurs façons d'obtenir le même résultat, comme souvent en informatique. Chacune ayant ses avantages et ses inconvénients.
Nous avons vu plus haut comment placer un SVG dans un page Html standard, cette technique reste parfaitement utilisable pour une page ASP.NET, il suffit d'insérer le code dans la page aspx.

Lorsqu'on travaille en code-behind les choses peuvent sembler moins simples. En effet, il n'existe pas de balise SVG particulières pas plus que la balise standard IMG ne sait afficher des SVG. De fait il faut pouvoir injecter le code de la balise **EMBED** à l'endroit précis où l'on souhaite afficher le graphique. Si cela peut être effectué en agissant directement sur la page aspx, il est généralement plus pratique de poser un composant sur la page en conception.

L'astuce consiste alors à utiliser un composant ASP.NET **Literal** dont on fixe par programmation la propriété **Text**. La figure 2 montre une page ASP.NET en conception avec un composant**Literal** dont l'**ID** est **ZoneImage**.

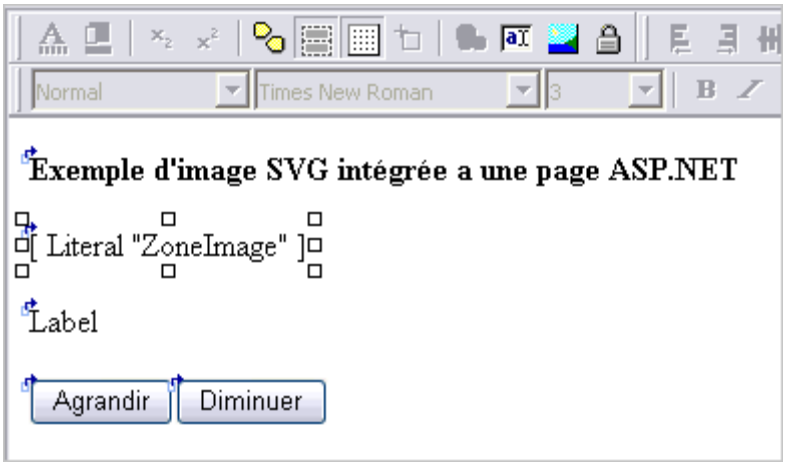

Figure 2 - Le composant Literal qui contiendra le SVG

Pour que le SVG puisse s'afficher il reste alors dans le code à initialiser la propriété **Text** du **Literal** de la façon suivante (syntaxe Delphi.net) :

Code 4 - Insertion du SVG dans la page ASP.NET

```
ZoneImage.Text := '<embed name="demo"
type="image/svg-xml" src="arcs1.svg" ' +
                  'width="' +
SVGWidth.Tostring + '" height="' +
SVGHeight.Tostring + '">';
```

Généralement on placera ce code dans le **Page_Load** après un test à un **IsPostBack** pour éviter de refaire l'opération à chaque *round-trip*, l'option **EnableViewState** du **Literal** étant par défaut à *true* la valeur sera conservée automatiquement.

Dans notre exemple (figure 2 et code 4), la zone image peut être retaillée (crop) par l'utilisateur grâce à deux boutons (*Agrandir* et *Diminuer*). Pour éviter un cycle de postback le code des boutons pourraient être du JavaScript exécuté purement en local sur le client. Ici nous utilisons les mécanismes classiques de ASP.NET et le code est exécuté par le serveur. La figure 3 montre la page ASP.NET en action après un clic sur le bouton *Agrandir*.

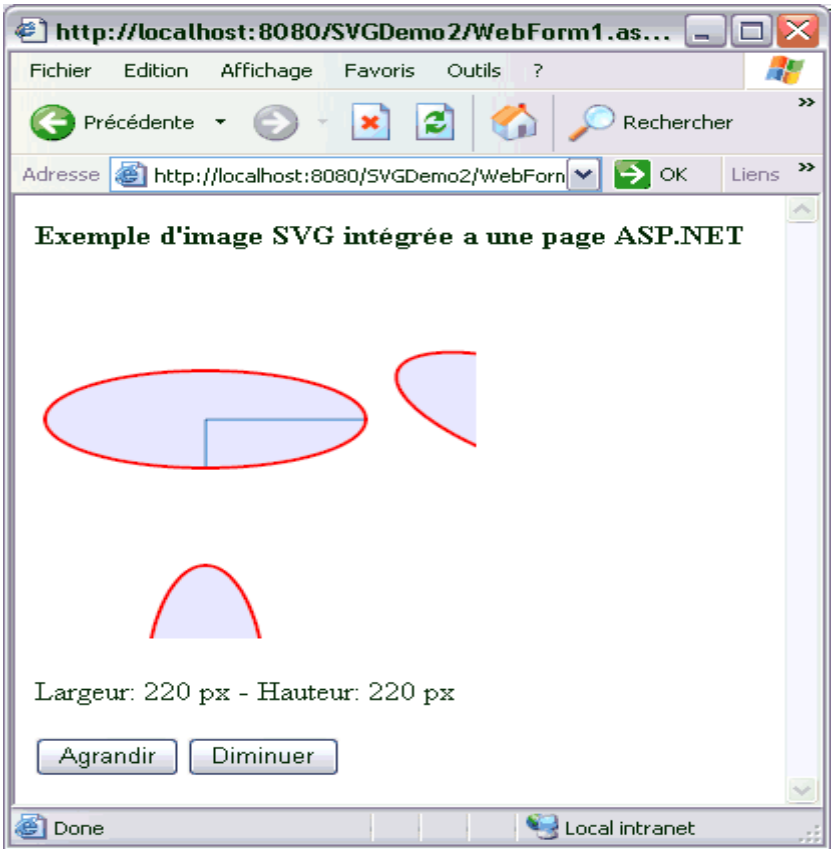

Figure 3 - Le SVG intégré à la page Web

Par défaut nous avons fixé la taille de la zone image à 200x200 px. Les boutons de changement de taille agissent par pas de 20 px sur les deux dimensions.

La gestion de la taille de l'image est rudimentaire et utilise deux variables **SVGWidth** et **SVGHeight** de type entier. Lors du premier appel de la page ces variables sont initialisées à 200 et ces valeurs sont placées dans le **ViewState** pour les conserver. Lors des appels ultérieurs à la page (appui sur les boutons) les variables sont récupérées du ViewState, manipulées, puis stockées à nouveau dans ce dernier.

Le code 5 montre le **Page_Load** de la page réalisant une partie de ce travail (syntaxe Delphi.net facilement traduisible en C#).

Code 5 - le Page_Load de la page ASP.NET de l'exemple

```
procedure TWebForm1.Page_Load(sender:
System.Object; e: System.EventArgs);
begin
if not IsPostBack then // premier chargement
de la page
 begin
  SVGWidth := 200;
  SVGHeight := 200;
  ViewState['SVGW']:=TObject(SVGWidth); //
boxing explicite, inutile sous C#
  ViewState['SVGH']:=TObject(SVGHeight); //
idem
 end else
 begin // chargements ultérieurs de la page
  SVGWidth :=
Convert.ToInt32(ViewState['SVGW']);
  SVGHeight :=
Convert.ToInt32(ViewState['SVGH']);
 end;
  CalcImage; // modification du Literal
end;
```

Le code 6 donne montre le corps de la méthode **CalcImage**. Celle-ci récupère la valeur des dimensions de l'image depuis le **ViewState** puis génère la chaîne qui remplacera le **Literal** dans la page.

Code 6 - La méthode CalcImage

```
procedure TWebForm1.CalcImage;
begin
  ViewState['SVGW']:=TObject(SVGWidth);
  ViewState['SVGH']:=TObject(SVGHeight);
  ZoneImage.Text := '<embed name="demo"
type="image/svg-xml" src="arcs1.svg" ' +
                    'width="' +
SVGWidth.Tostring + '" height="' +
SVGHeight.Tostring + '">';
  label2.Text := 'Largeur: ' +
SVGWidth.ToString + ' px - Hauteur: ' +
SVGHeight.ToString + ' px';
end;
```

Le code 7 montre ce que fait l'appui sur le bouton *Diminuer* : diminution de 20 px de chaque dimension et appel à **CalcImage** qui reformatera le texte du **Literal**.

Code 7 - Le bouton Diminuer

```
procedure TWebForm1.Button2_Click(sender:
System.Object; e: System.EventArgs);
begin
  SVGWidth := SVGWidth - 20;
  SVGHeight := SVGHeight - 20;
  CalcImage;
end;
```

La méthode que nous utilisons ici n'est pas la plus efficace ni la plus subtile mais elle a l'avantage de montrer comme traiter un SVG uniquement par code côté serveur. Comme nous le disions il serait ici plus judicieux de coder l'effet des boutons en script côté client, de même que puisque notre démonstration gère une zone carrée nous n'aurions dans l'absolu besoin que d'une seule variable et non de deux.

VIII. La production de graphiques à la volée

Pour l'instant nous nous sommes contentés d'intégrer des graphiques existants dans des pages Html ou ASP.NET. Cela est déjà très intéressant puisque SVG est un format image beaucoup plus léger que tous les autres en raison de sa nature vectorielle descriptive. Aucun bitmap, même compressé JPG ou PNG n'a besoin d'être transmis et l'affichage sera propre quelque soit le niveau de zoom.

Mais ce n'est exploiter qu'une infime partie des possibilités qu'offrent SVG.
Nous le disions, SVG est plus qu'un format graphique, c'est en réalité un système programmable permettant de faire des animations et de les piloter depuis l'extérieur, créant de fait une alternative à Flash. Les techniques afférentes à ce genre de programmation s'écarte de trop du sujet du présent article et nous ne pouvons que renvoyer le lecteur intéressé vers les sites, de plus en plus nombreux, qui abordent la création de graphiques SVG interactifs.

Nous allons ici nous intéresser à un autre aspect de SVG lié à sa nature descriptive et vectoriel : la génération à la volée de graphiques.
Cela est en réalité fort simple puisqu'il suffit de créer un flux XML contenant le texte décrivant le dessin à produire. En suivant la syntaxe et la grammaire de SVG il est donc possible de fabriquer un tel flux qui reproduirait la Joconde… à condition de savoir la dessiner !
Les informaticiens ont rarement besoin d'avoir le talent de Da Vinci pour concevoir des applications professionnelles. Les graphiques les plus utilisés sont souvent des cartes géographiques ou des business graphics de type histogramme ou camembert. Des choses beaucoup plus simple à dessiner, et donc à décrire, que le doux visage de Mona Lisa.

Pour simplifier les choses nous allons mettre en œuvre une classe écrite par *Dennis Forbes* et que vous trouverez sur cette page : **YaFla**. Il s'agit d'une classe C# permettant de générer

des**histogrammes** ou des **camemberts** au format SVG. Toute la logique de fabrication du code SVG XML est incluse dans la classe et son étude vous apprendra comment réexploiter vous-mêmes cette possibilité dans votre propre code.

Notre application exemple utilisera donc YaFla pour générer un histogramme. La classe utilise une source de données **DataTable** qu'il suffira donc de créer et initialiser avec les valeurs du graphique. La page ASP.NET elle-même se comportera *comme un générateur d'image* c'est-à-dire qu'elle renverra directement un SVG au lieu d'une page ASP.NET. La technique peut être utilisée pour créer un compteur de page Web par exemple (les compteurs du site **e-naxos** fonctionnent comme cela d'ailleurs, en générant un JPEG depuis une page ASP.NET se comportant au final comme une image).

Pour exploiter Yafla il vous suffit de télécharger le code source depuis la page indiquée plus haut et de créer un assemblage .NET pour en faire une DLL. Cela ne pose aucun problème (un exemple est fourni dans le zip accompagnant l'article).

Les graphiques SGV peuvent être compressés par la méthode zip, donnant alors naissance à des fichiers dont l'extension est **.svgz**. Pour créer et utiliser de tels fichiers il faut une routine zip… Pour ce faire nous utiliserons la librairie **SharpZipLib**, sous licence GPL, que vous trouverez à cette endroit : **SharpZipLib**. Cette excellente librairie peut bien entendu être utilisée dans vos applications partout où vous avez besoin de gérer des **données compressées**. Sur le site indiqué vous trouverez d'ailleurs une application de type *WinZip* utilisant SharpZipLib. Bien entendu ici encore il s'agit de code C#. Le présent article a pris le parti d'utiliser Delphi.NET pour illustrer les exemples afin de rendre justice à cet excellent langage et en raison de ses similarités avec C# rendant le portage du code vers ce dernier très facile. Cela étant dit, votre humble serviteur ne saurait que trop conseiller aux utilisateurs de Delphi.NET de se former aussi à C#, cela est plus facile qu'il n'y paraît et c'est un savoir que vous ne regretterez pas d'avoir acquis…

Armés de YaFla et de SharpZipLib nous pouvons aisément générer des graphiques SVG compressés… Et vous allez voir à quel point

cela est facile … quand d'autres ont écrit le code utile ! (ceci est une remarque de M. Lapalisse qui passait par là…).

Yafla utilise une technique très simple pour le passage des données, il suffit de lui donner un **DataTable** correctement rempli. Cela peut bien évidemment être fait dans le cadre d'une connexion à des données externes, cas le plus probable pour un histogramme, mais cela peut aussi être réalisé en créant et en initialisant par programmation une instance de DataTable.
La seule contrainte imposée par la librairie est que la table doit contenir au moins deux colonnes, l'une nommée **Caption** contenant le texte à afficher et l'autre **Value** contenant la valeur.
Le code 8 montre l'initialisation effectuée dans Page_Load (syntaxe Delphi.NET).

Code 8 - Initialisation du graphique

```
procedure TWebForm1.Page_Load(sender:
System.Object; e: System.EventArgs);
var
 BarChart : yaflaSVG2DBarChart;
 tmpData : DataTable;
 NewDataColumn : DataColumn;
 NewDataRow : DataRow;
begin
 BarChart := yaflaSVG2DBarChart.Create;
 BarChart.SVGTitle := 'Prix des
Microprocesseurs';
 BarChart.SvgSubTitle := 'Juillet 2005';
 tmpData := DataTable.Create('Demo');

 NewDataColumn :=
DataColumn.Create('Caption');
 NewDataColumn.DataType :=
System.Type.GetType('System.String');
 tmpData.Columns.Add(NewDataColumn);
 NewDataColumn := DataColumn.create('Value');
 NewDataColumn.DataType :=
System.Type.GetType('System.Double');
 tmpData.Columns.Add(NewDataColumn);
```

La variable **BarChart** est l'instance du graphique qui sera manipulée. On fixe un titre et un sous-titre puis on créé une table

temporaire de type **DataTable**. Cette table se voit ensuite agrémentée de deux colonnes comme indiqué plus haut, l'une de type **String** et l'autre de type **Double**.

Chaque nouvelle barre de l'histogramme sera rendue sur la base d'une valeur contenue dans une ligne de la table. On ajoute une nouvelle ligne de la façon suivante (code 9):

Code 9 - L'ajout d'une ligne à la table

```
NewDataRow := tmpData.NewRow();
NewDataRow['Caption'] := 'Athlon 64 2800';
NewDataRow['Value'] := Tobject(107.0);
tmpData.Rows.Add(NewDataRow);
```

Une fois toutes les lignes ajoutées, il ne reste plus qu'à fixer les paramètres de l'histogramme, ce qui est réalisé par le code suivant (code 10).

Code 10 - Paramétrage de l'histogramme

```
BarChart.SVGDataSource := tmpData;       //
source de données
BarChart.SVGDynamic := true;             //
contient des éléments dynamiques
BarChart.SVGLabels := true;              //
affichage des labels
BarChart.SVGHeightRatio := 1.0;          //
ratio hauteur / largeur
BarChart.SVGLegendSizePercentage := 15; //
taille des légendes en %
BarChart.BarWidthPercentage := 80.0;     //
taille en  % des barres
BarChart.GridInterval := 50;             //
intervalle pour la grille de fond
BarChart.BackgroundColor :=
System.Drawing.Color.Azure; // couleur fond
BarChart.ValueFormat := 'c';             //
format d'affichage du champ valeur
BarChart.SVGLegend := false;             //
affichage des légendes (pavé à côté du dessin)
BarChart.Maximum := 500;                 //
valeur maxi du dessin
```

```
BarChart.SvgOrientation :=
GraphOrientation.Vertical; // ou bien
Horizontal;
```

Enfin, il convient maintenant de retourner le graphique et de faire passer la page ASP.NET pour un simple fichier statique SVG comme le montre le code 11.

Code 11 - Retourner l'image au browser

```
      Response.Clear;
 Response.BufferOutput := true;
 BarChart.SVGXML.Save(Response.OutputStream);
      Response.ContentType := 'image/svg-xml';
      Response.AppendHeader('Content-
Disposition','inline;filename=graphic.svgz');
 Response.Flush(); // ou Response.&End ();
```

Ce code commence par effacer l'objet réponse. La propriété **BufferOuput** est placée à *true* pour indiquer que la page doit être envoyée qu'une fois la bufferisation terminée. On appelle ensuite la méthode **Save** de la propriété **SVGXML** du graphique pour remplir le stream de l'objet réponse par la représentation XML de l'histogramme.
Pour terminer, le type de contenu **"image/svg-xml"** est spécifié, ce qui permettra au browser de comprendre que ce qu'il reçoit est une image SVG. L'ajout de l'entête **Content-Disposition**fixe un nom de fichier pour l'image. Il semble que ni IE ni le plugin Adobe ne s'en servent pour l'instant puisque lorsqu'on clique sur l'image pour la sauvegarder le nom proposé reste WebForm1. Mais cela changera peut être, alors autant placer le bon entête !
La réponse est finalement envoyée au client par l'ordre **Flush**. On peut toutefois préférer l'utilisation de la méthode **End()** qui met fin totalement et proprement à la transmission.

Le résultat obtenu est montré figure 4.

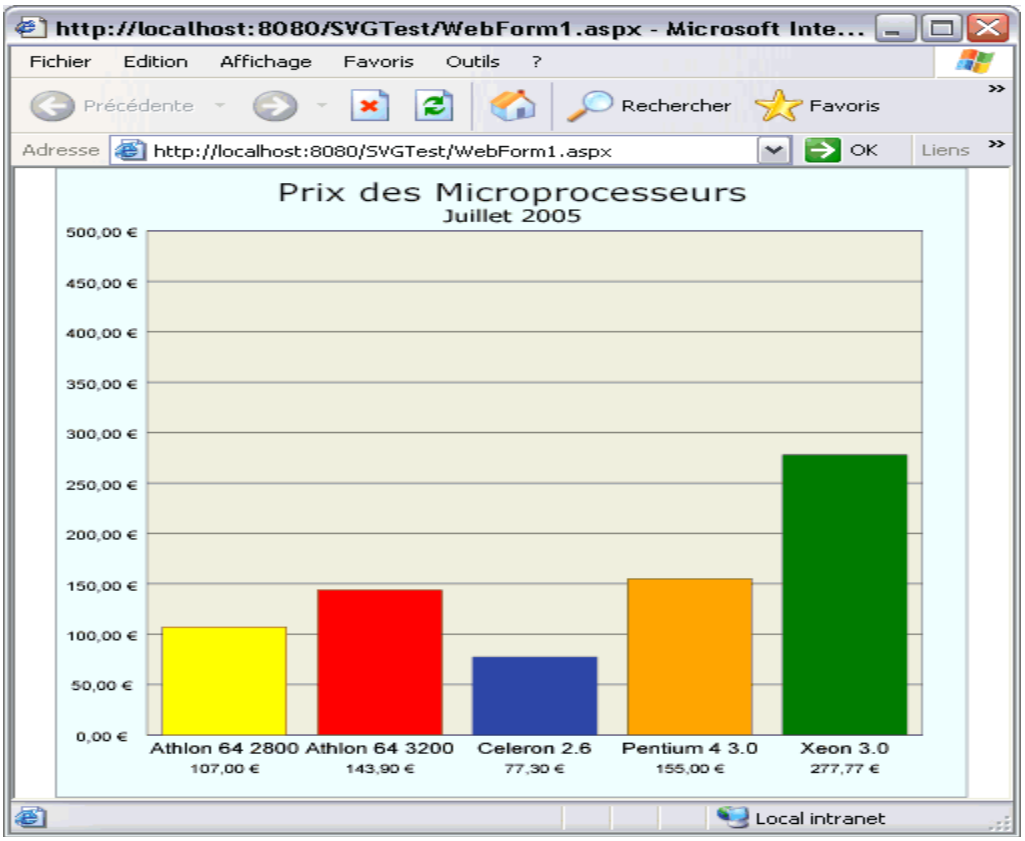

Figure 4 - L'histogramme généré par l'application

IX. Conclusion

Cet article devait au départ être un simple point sur les techniques tournant autour de SVG. Au fil de son écriture les choses sont apparues bien plus touffues que prévues et arrivé à cette conclusion l'auteur se rend bien compte qu'il n'a fait qu'à peine effleurer le sujet...
Mais finalement il n'était pas question de faire un livre sur SVG ni un cours de développement, le but était de vous faire découvrir SVG que vous ne connaissez peut être pas encore, et de faire voir comment on pouvait réaliser simplement des petites choses utiles.
Si cet article a atteint ce but l'auteur en sera déjà satisfait. Puissent ces quelques pages vous être utiles. Bon développement