Introduction à SVG dynamique

Code: svg-dyn

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/svg-intro/svg-dyn.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/svg-dyn.pdf**

Auteurs et version

- Daniel K. Schneider - Stéphane Lattion

- Version: 0.8 (modifié le 28/11/06)

Prérequis

Module technique précédent: **svg-intro**

Autres modules

Module technique suppl.: xml-xslt

Module technique suppl.: visu-gen (ainsi que les divers prérequis Php !)

Abstract

Introduction à SVG dynamique

Note: SVG dynamique server-side (PHP/XML/DOM/XSLT) fait l'objet du module visu-gen.

A faire

- beaucoup ! (DOM plus avancé et interaction HTML/SVG surtout).
- Voir aussi notre petite page SVG: **http://tecfa.unige.ch/guides/svg/pointers.html**

Objectifs

- Animations de type SMIL
- Interactivité de type SMIL
- Animations et interactivité avec JavaScript / DOM

Remerciements

- J.J.Solari (le traducteur de la recommandation SVG1 du W3C en version française).
  J'ai "pompé" pas mal de ce texte.

1. Table des matières détaillée

2. Animation SVG

2.1 Principe et méthodes d'animation

- Animer veut dire changer un attribut d'un élément SVG (et donc aussi d'un groupe)

2 méthodes:

1. Animation de type "XML/SMIL" avec des balises spéciales
   - Il faut installer un plugin comme Adobe SVG Viewer
   - on peut animer pratiquement chaque attribut avec les éléments d'animation SVG/SMIL (un peu comme en VRML)
   - L'animation SVG/SMIL étend celle de SMIL avec quelques extensions.
   - Il existe des balises SVG spéciales pour construire des animations.
2. Animation via le DOM de SVG avec un script
   - marche avec Firefox 1.5 et mieux
   - chaque attribut et "style sheet setting" est accessible selon DOM1 & 2. En plus, il existe un jeu d'interfaces DOM additionnelles.
   - En gros, on écrit un petit programme ECMAScript (JavaScript) qui modifie les attributs d'éléments ou qui ajoute/enlève des éléments/attributs du DOM.

Principe du "time-based"

- L'animation est "time-based" (vs. le "frame-based" de Flash): on indique le départ et la durée d'une animation
- on peut animer des attributs de façon indépendante (animations en parallèle)
- une animation peut se déclencher suite à un geste de l'utilisateur (event) ou encore au temps voulu après le chargement du SVG.

2.2 Balises pour l'animation "XML/SMIL"

- Il existe 5 balises (éléments XML) pour l'animation
- Il faut un plugin SVG pour le moment (celui de Adobe par exemple) !

L'élément 'animate':

s'utilise pour animer un seul attribut, ou une seule propriété, au cours du temps.

L'élément 'set':

offre un moyen simple pour le paramétrage de une seule valeur d'attribut pour une durée spécifiée.

L'élément 'animateMotion':

entraîne le déplacement d'un élément le long d'un tracé de mouvement.

L'élément 'animateColor':

spécifie une transformation de couleur au cours du temps.

L'élément 'animateTransform':

anime un attribut de transformation sur un élément cible, permettant de ce fait aux animations de contrôler translation, changement d'échelle, rotation et/ou inclinaison.

Exemple 2-1: Exemple de sensibilisation avec animate

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/translation.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/ (répertoire)**

```
<svg xmlns="http://www.w3.org/2000/svg">

 <rect x="50" y="50" width="200" height="100"
  style="fill:#CCCCFF;stroke:#000099">
  <animate attributeName="x" attributeType="XML"
           begin="0s" dur="5s" from="50" to="300" fill="freeze"/>
 </rect>
 <text x="55" y="90" style="stroke:#000099;fill:#000099;fontsize:24;">
  Hello. Let's show a crawling rectangle ! </text>
</svg>
```

- L'élément <animate /> est simplement placé à l'intérieur de l'élément
- On anime la position horizontale ("x" du rect)
  - `attributeName` = "x" indique qu'on anime l'élément "x" du parent
  - `begin = "0s"` indique que l'animation commence après 0 secondes
  - `dur="5s"` indique que la durée est de 5 secondes
  - `from="50"` point de départ de X
  - `to="300"` point d'arrivé de X
  - `fill="freeze"` : l'animation gèle (c.a.d. le rectangle reste où il est)

2.3 Les attributs communs aux balises d'animation

- Voici un petit résumé sans rentrer dans les détails
- Utiliser à titre indicatif et ensuite regarder un manuel si nécessaire

xlink:href = "uri"

- SVG utilise xlink pour designer le "animation target".
- Il s'agit d'une référence d'URI vers l'élément qui est la cible de cette animation et dont un attribut sera donc modifié au moment voulu.
- Dit plus simplement: On fait un lien vers l'attribut id de l'élément à animer
- Note: on a pas toujours besoin de se référer à un id pour animer. Si on place des balises d'animation à l'intérieur d'un élément, on se réfère aux attributs de cet élément.
- il faut déclarer le namespace "xlink" dans un élément parent

```
<svg width="8cm" height="3cm"  xmlns="http://www.w3.org/2000/svg">
```

attributeName: "nom"

- indique le nom de l'attribut qu'il faut animer

attributType: "type"

- indique le type d'attribut qu'il faut animer, exemple:

```
Syntaxe: attributeType = "CSS | XML | auto"
```

Ci-dessous, on anime l'attribut "x" (la position x) d'un rectangle et c'est du type XML (SMIL):

```
<rect x="300">
 <animate attributeName="x" attributeType="XML"
   begin="0s" dur="9s" fill="freeze" from="300" to="100" />
</rect>
```

begin : begin-value-list

- définit quand l'élément devrait commencer (i.e. devenir actif).
- "begin-value-list à une définition très compliquée, on peut définir des multiples débuts et fins selon les besoins de synchronisation (cf. la spécification)
- Dans les cas simples on indique juste un début. L'exemple suivant dit "après 10 secondes" (voir ci-dessous pour d'autres "clock-value" simples:

```
<animate attributeName="x" attributeType="XML"
   begin="10s" dur="9s" fill="freeze" from="300" to="100" />
```

- Plus tard, on verra qu'on peut aussi utiliser un événement (ex. cliquer dessus) qui déclenche !

dur : Clock-value | "media" | "indefinite"

- définit la durée de l'animation
- `Clock-value` spécifie la longueur d'une durée de présentation. La valeur doit être supérieure à 0.
- `"indefinite"` spécifie la durée simple comme indéfinie (c'est le défaut si vous ne précisez ni durée ni "média")
- Typiquement on indique des seconds, par ex. 10s = 10 secondes

```
<animate attributeName="x" attributeType="XML"
   begin="0s" dur="9s" fill="freeze" from="300" to="100" />
```

La définition de la valeur "clock-value" est compliquée:

Voici qqs. exemples qui marchent:

- Valeurs d'horloge complètes :

```
02:30:03    = 2 heures, 30 minutes et 3 secondes
50:00:10.25 = 50 heures, 10 secondes et 250 millisecondes
```

- Valeur d'horloge partielles :

```
02:33   = 2 minutes et 33 secondes
00:10.5 = 10.5 secondes = 10 secondes et 500 millisecondes
```

- Les valeurs Timecount :

```
3.2h    = 3.2 heures = 3 heures et 12 minutes
45min   = 45 minutes
30s     = 30 secondes
5ms     = 5 millisecondes
12.467  = 12 secondes et 467 millisecondes
```

end : end-value-list

min : Clock-value | "media"

max : Clock-value | "media"

- Ces trois attributs moins souvent utilisés peuvent contraindre la durée active
- on peut indiquer une fin, un minimum ou encore un maximum de durée.
- voir la spécification :)

restart : "always" | "whenNotActive" | "never"

- `always`: L'animation peut être relancée à tout instant. C'est la valeur par défaut.
- `whenNotActive`: L'animation ne peut être relancée que si elle n'est pas active

repeatCount : numeric value | "indefinite"

- Spécifie le nombre d'itérations de la fonction d'animation. La valeur d'attribut peut être l'une des suivantes :
- `numeric value` = valeur numérique « décimale » qui spécifie le nombre d'itérations.
- `"indefinite"` = L'animation est définie comme se répétant indéfiniment (i.e. jusqu'à la fin du document).

repeatDur : Clock-value | "indefinite"

- Spécifie la durée totale pour la répétition.
- `Clock-value` = spécifie la durée
- `"indefinite"` = l'animation est définie comme se répétant indéfiniment (i.e. jusqu'à la fin du document).

fill : "freeze" | "remove"

- `freeze` = l'effet d'animation F(t) est défini pour geler la valeur d'effet à la dernière valeur de la durée active. L'effet d'animation est « gelé » pour le restant de la durée du document (ou jusqu'à ce que l'animation soit relancée)
- `remove` = l'effet d'animation est stoppé (ne s'applique plus) quand la durée active de l'animation est terminée. (à moins que celle-ci ne soit relancée)

2.4 Les attributs pour les valeurs d'animation au cours du temps

- Voici un petit résumé sans rentrer dans les détails, à utiliser à titre indicatif
- Le principe de base de ce type de langage (comme dans VRML) est de laisser la possibilité à l'utilisateur d'indiquer simplement au moins 2 valeurs (début/fin) et de laisser la machine gérer le passage entre ces valeurs (interpolation)
- Définition de la notion d'interpolation
  - En animation, calcul des images intermédiaires entre deux formes polynomiales ou encore entre 2 positions sur l'écran.
  - Pour la couleur, calcul des couleurs intermédiaires entre deux couleurs
  - Il existe ensuite plusieurs modes (cf. ci-dessus)

calcMode = "discrete | linear | paced | spline"

- Spécifie le mode d'interpolation pour l'animation.
- discrete = la fonction d'animation passera d'une valeur à l'autre sans aucune interpolation.
- linear = une interpolation linéaire simple entre les valeurs est utilisée pour le calcul de la fonction d'animation. Sauf pour l'élément 'animateMotion', c'est la valeur par défaut pour calcMode.
- paced = interpolation qui produit une vitesse de transition égalisée au cours de l'animation.
- spline = [selon la traduction française de la spécification SVG:] interpole à partir d'une valeur de la liste de l'attribut values sur la suivante, selon une fonction temporelle définie par une courbe (spline) de Bézier cubique. Les points de la spline sont définis dans l'attribut keyTimes et les points de contrôle pour chaque intervalle sont définis dans l'attribut keySplines.

values = "<liste>"

- une liste d'une ou plusieurs valeurs, séparées par des points-virgules.

keyTimes = "<liste>"

- Une liste de valeurs de temps, séparées par des points-virgules, utilisée pour le contrôle de la vitesse de défilement de l'animation. Chaque temps de la liste correspond à une valeur dans la liste de l'attribut values et définit quand la valeur est utilisée dans la fonction d'animation.
- Chaque valeur de temps, dans la liste de l'attribut keyTimes, est spécifié en valeur décimale comprise entre 0 et 1 (inclus), représentant un décalage proportionnel dans la durée simple de l'élément d'animation.

keySplines = "<liste>"

- Un jeu de points de contrôle de Bézier, associé avec la liste de l'attribut keyTimes, définissant une fonction de Bézier cubique qui contrôle l'allure du défilement des intervalles.

2.5 Les attributs qui contrôlent la succession des animations

additive = replace | sum

- Cette balise est utilisée lorsqu'on effectue plus qu'une seule transformation en même temps.
- sum = l'animation va s'ajouter à la valeur sous-jacente de l'attribut et des autres animations de faible priorité.
- replace = l'animation va surclasser la valeur sous-jacente de l'attribut et des autres animations de faible priorité. C'est la valeur par défaut
- Si une simple animation « agrandir » peut accroître la largeur d'un objet de 10 pixels ....

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
           additive="sum"/>
</rect>
```

- ..., il est souvent pratique, pour des animations répétées, de se construire sur des résultats précédents, qui s'accumulent avec chaque itération. L'exemple suivant fait que le rectangle continue à grandir au fur et à mesure de la répétition de l'animation :

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
           additive="sum" accumulate="sum" repeatCount="5"/>
</rect>
```

- À la fin de la première répétition, le rectangle a une largeur de 30 pixels, à la fin de la deuxième une largeur de 40 pixels et à la fin de la cinquième une largeur de 70 pixels.

accumulate = "none | sum"

- C.f. la spécification

Exemple 2-2: Deltas d'une animation

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/inc-growth.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/**

- À la fin de la première répétition, le rectangle a une largeur de 40 pixels, à la fin de la deuxième une largeur de 60 pixels et à la fin de la cinquième une largeur de 120 pixels.
- L'animation totale dure 5 * 5 s = 25 secondes
- On construit sur des résultats précédents, qui s'accumulent avec chaque itération:

```
<rect x="50" y="50" width="20px" height="20px"
  style="fill:yellow;stroke:black">
  <animate attributeName="width" dur="5s" repeatCount="5"
           fill="freeze"
           from="0px" to="20px"
           additive="sum" accumulate="sum"/>
 </rect>

 <text x="55" y="90" style="stroke:#000099;fill:#000099;fontsize:24;">
  Hello. Let's show a growing rectangle ! </text>
```

2.6 L'élément set

- De la spécification: L'élément 'set' offre un moyen simple pour le paramétrage d'une seule valeur d'attribut pour une durée spécifiée. Il gère tous les types d'attributs, y compris ceux qui ne peuvent pas être raisonnablement interpolés, comme les chaînes et les valeurs booléennes.
- L'élément 'set' n'est pas additif. Donc 2.5 "Les attributs qui contrôlent la succession des animations" [14] ne marchera pas.

Exemple 2-3: Simple animation avec set et animation

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/simple-set.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/ (répertoire)**

```
<rect x="50" y="50" width="200" height="100"
    style="fill:#CCCCFF;stroke:#000099"
    visibility ="hidden" >
    <set attributeName="visibility" attributeType="XML"
      begin="4s" dur="5s" to="visible"/>
  </rect>

<rect style="fill:yellow;stroke:#000099" x="250" y="50" width="200" height="100"
opacity="0" >
    <animate attributeName="opacity" attributeType="XML"
      begin="1s" dur="5s" from="0" to="1" fill="freeze"/>
  </rect>
```

2.7 AnimateColor

Exemple 2-4: Animation d'une couleur

url: **http://tecfa.unige.ch/guides/svg/ex/anim-colors/anim-color1.svg**

```
<svg height="900" width="900">

<!-- un gros rectangle qui remplit l'écran -->
<rect style="fill:#000000;" height="900" width="900" y="0" x="0">
  <animate fill="freeze" dur="5s" begin="0.1s"
        to="#FFFFFF" from="#000000" calMode="linear" attributeName="fill"/>
</rect>

<!-- représentation d'une tige (rectangle haut et fin) -->
<rect style="fill:#CC9933;stroke:#CC9933" width="20" height="500"
      ry="5" rx="5" y="100" x="400">
   <animate dur="5s" begin="1s" to="#000000"
            from="#CC9933" calMode="linear" attributeName="fill"/>
   <animate dur="5s" begin="6s" from="#000000"
            to="#CC9933" calMode="linear" attributeName="fill"/>
</rect>
```

- Le principe est assez simple: on passe d'une couleur à une autre
- Ici on a deux animations:
  - l'arrière-plan va de blanc à noir (from="#000000") à (to="#FFFFFF")
  - la tige va d'abord de from="#CC9933" à to="#000000" (après 1s et pendan 4sec) et ensuite de #000000" à "#CC9933".

2.8 AnimateTransform

Exemple 2-5: Simple rotation

url: **http://tecfa.unige.ch/guides/svg/ex/anim-trans/rotation.svg**

```
<title>Simple rotation example</title>
<desc> Rotation with a grouping node </desc>
<g>
  <rect x="50" y="50" rx="5" ry="5" width="200" height="100"
     style="fill:#CCCCFF;stroke:#000099"/>
 <text x="55" y="90"
                style="stroke:#000099;fill:#000099;fontsize:24;">
   Hello. Let's rotate</text>
  <animateTransform attributeName="transform" type="rotate"
             values="0 150 100; 360 150 100"
             begin="0s" dur="5s" />
</g>
```

- Cette exemple moche fait une rotation
  - on anime l'attribut "rotate" du <g> (qui est au départ est à son défaut, c.à.d. sans rotation)
  - on doit indiquer juste le degré de rotation (0 au départ et 360 à la fin).

2.9 AnimateMotion

Exemple 2-6: Animation d'un mouvement le long d'un tracé

url: **http://www.yoyodesign.org/doc/w3c/svg1/animate.html (original)**

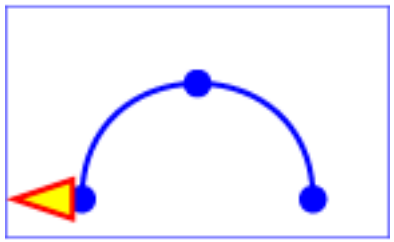
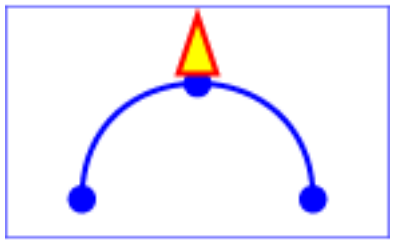
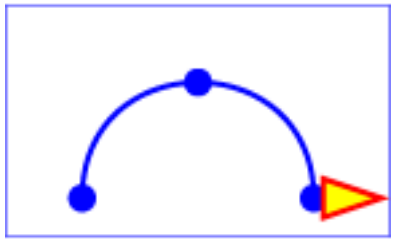
url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/animMotion01.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/**

- L'élément 'animateMotion' entraîne le déplacement d'un élément appelé le long d'un tracé de mouvement.
  - A ne pas confondre avec des simples translations, rotations etc. qu'on fait avec animate ou animateTransform.
- Voici la définition d'un triangle et de son animation le long d'un chemin (voir les slides svg-intro pour la définition d'un path)

```
<path d="M-25,-12.5 L25,-12.5 L 0,-87.5 z"
        fill="yellow" stroke="red" stroke-width="7.06"  >
 <!-- Definit l'animation sur le trace de mouvement -->
  <animateMotion dur="6s" repeatCount="indefinite"
                  path="M100,250 C 100,50 400,50 400,250" rotate="auto" />
</path>
```

- Note: La définition du dessin du tracé suit exactement le même path

2.10 Animations combinées

- Il est assez difficile de synchroniser un script compliqué, par contre faire des animations en parallèle est simple (il suffit de regarder les secondes ...)

Exemple 2-7: Exemple d'une animation combinée

url: **http://www.yoyodesign.org/doc/w3c/svg1/animate.html**
(home de la traduction française de la spécification SVG 1.0)

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/anim01.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/**

- Plusieurs animations:
  - largeur/hauteur du rectangle jaune
  - déplacement de l'origine du rectangle jaune (vers le haut à gauche)
  - animation de couleur pour le texte
  - déplacement le long d'un chemin du texte
  - rotation du texte
  - agrandissement du texte

3. Interactivité avec SVG

3.1 Principe et gestion des événements

- L'interactivité et donc la gestion des événements ressemble au principe des GUI modernes (par exemple JavaScript) et elle est conforme à la spécification DOM2

Voici les types d'interactivité

- Des actions initiées par l'utilisateur, telle que presser le bouton d'une souris, peuvent causer l'exécution d'animations (voir 2. "Animation SVG" [5]) ou de scripts ;
- L'utilisateur peut initier des hyperliens vers de nouvelles pages Web par des actions (par ex. cliquer sur un élément)
- L'utilisateur peut zoomer dans un contenu SVG ou effectuer un panoramique autour de celui-ci (dépend du client).
- Les déplacements par l'utilisateur du dispositif de pointage peuvent modifier le curseur qui indique sa position

L'essentiel: SVG comprend

- des ***attributs d'événements*** (ex. `onclick="hello()"`) pour lancer des scripts
- des ***noms d' événements*** qu'on peut insérer dans certains attributs (ex: `begin="click"`) et qui définissent l'élément d'animation à déclencher quand un événement donné survient.

3.2 Evénements de base SVG/SMIL

- La spécification SVG1 définit plus de 20 événements
- Les plus importants peuvent être capturés par des "attributs d'événements" qu'on insère dans les balises qu'on veut rendre interactives.
- Il existe certains événements sans attributs qui concernent la modification de l'arbre DOM
- En plus, à l'aide du Modèle Objet de Document (DOM) de SVG, un script peut enregistrer des guetteurs d'événements (event handlers) DOM2 de manière à ce qu'un script puisse être invoqué quand un événement donné survient .

Noms d'événement et attributs d'événement: petit résumé

- le nom de l'événement sera utilisé pour identifier un event dans des balises d'animation (comme "begin").

```
<animateColor fill="freeze" dur="0.1s" to="blue" from="yellow"
                    attributeName="fill" begin="mouseover"/>
```

- l'attribut d'événement permet de placer des "guetteurs d'événement" (event handlers) et qui pourront déclencher des scripts (voir 4. "Interactivité avec le DOM et EcmaScript" [27]

```
<circle onmouseover="circle_click(evt)" cx="300" cy="225" r="100" fill="red"/>
```

- Donc ***attention***: Ne confondez pas nom d'événement à utiliser dans les attributs des éléments d'animation et attributs d'événements pour lancer des scripts !!

  "mouseover" pas égal à "onmouseover" (.... ou vous perdrez qq. heures ...)

La table suivante réunit la définition de quelques éléments et attributs associés

Nom de l'événement	Description: un événement survient quand ...	Attribut d'événement
focusin	un élément reçoit l'attention (le focus), par ex. l'élément 'text' devient sélectionné.	onfocusin
focusout	un élément perd l'attention, (désélection).	onfocusout
activate	un élément est activé, par exemple, au travers d'un clic de souris ou l'appui d'une touche. • Un argument numérique est fourni pour indiquer le type d'activation : 1 pour une activation simple (par ex. simple clic ou la touche Entrée), 2 pour une hyperactivation (par ex. un double-clic ou la combinaison de touches).	onactivate
click	le bouton du dispositif de pointage (souris, touchpad, stylo etc.) est cliqué au-dessus d'un élément. • Un clic est l'appui et le relâchement du bouton de la souris au-dessus d'une même position sur l'écran. (mousedown/mouseup sont avant/ après).	onclick
mousedown	le bouton de pointage est pressé au-dessus d'un élément.	onmousedown
mouseup	le bouton de pointage est relâché au-dessus d'un élément.	onmouseup
mouseover	le dispositif de pointage est déplacé sur un élément.	onmouseover
mousemove	le dispositif de pointage est déplacé alors qu'il est au-dessus d'un élément..	onmousemove
mouseout	le dispositif de pointage (souris) est écarté de l'élément.	onmouseout
SVGLoad	le client a complètement interprété l'élément et ses descendants • il est prêt à agir de manière appropriée sur cet élément (affichage par exemple). Les ressources externes appelées requises doivent être chargées, interprétées et prêtes aussi.	onload

Nom de l'événement	Description: un événement survient quand ...	Attribut d'événement
SVGUnload	le client enlève le document SVG (élément svg le plus externe)	onunload
SVGAbort	le chargement de la page est interrompu avant qu'un élément ait pu être complètement chargé.	onabort
SVGError	un élément ne se charge pas correctement ou quand une erreur survient lors de l'exécution d'un script.	onerror
SVGResize	une vue de document (svg le plus externe) est redimensionnée.	onresize
SVGScroll	une vue du document est glissée sur X, sur Y ou les deux.	onscroll
SVGZoom	le document change de niveau de zoom lors d'une interaction avec l'utilisateur. (éléments 'svg' les plus externes).	onzoom
beginEvent	un élément d'animation commence. • voir la description de l'interface TimeEvent dans la spéci.SMIL	onbegin
endEvent	un élément d'animation s'achève (voir TimeEvent)	onend
repeatEvent	un élément d'animation se répète. • Il est déclenché toutes les fois où l'élément se répète, après la première itération. (voir l'interface TimeEvent aussi)	onrepeat

Exemple 3-1: Simple animation avec un événement mouseover

url: **http://tefa.unige.ch/guides/svg/ex/mouse-over/mouse-over1.svg**

- Un lien vers un autre URL se met simplement autour de l'élément cliquable

```
<a xlink:href="http://tecfa.unige.ch">
 <rect style="fill:#00FF00;stroke:#00FF00" width="200"
       height="26" ry="5" rx="5" y="100" x="100"/>
</a>
```

- L'ellipse jaune qui contient un texte va activer une animation (changement de couleur) lorsqu'on glisse la souris dessus ou lorsqu'on la sort de nouveau (l'attribut "begin" va s'activer lorsqu'il y a un mouseover ou un mouseout).

```
<g transform="translate(100,100)">
    <ellipse stroke-width="2" stroke="black" fill="yellow" ry="1cm" rx="2cm"
             id="hint_button">
      <animateColor fill="freeze" dur="0.1s" to="blue" from="yellow"
                    attributeName="fill" begin="mouseover"/>
      <animateColor fill="freeze" dur="0.1s" to="yellow" from="blue"
                    attributeName="fill" begin="mouseout"/>
    </ellipse>
    <text style="font-size:12;" alignment-baseline="middle" x="-1cm">Touch me !</text>
</g>
```

- Ensuite, on a un contenu caché et qui s'affiche lorsque l'élément qui a l'id "**hint_button**" a un mouseover. Autrement dit: l'attribut "begin" d'une animation ne peut pas seulement être un temps, un événement de l'élément parent comme ci-dessus, mais également un événement qui a lieu dans un autre élément (l'ellipse dans notre cas anime la propriété style du cercle caché)!!

```
<g transform="translate(100,200)" style="display:none">

<circle style="fill:yellow;" r="2cm" cy="1cm" cx="1cm"/>
    <text style="font-size:12;" alignment-baseline="middle" x="-0.8cm" y="1cm">
    Find the scecret URL !</text>

    <animate fill="freeze" dur="0.1s" begin="hint_button.mouseover"
      from="none" to="block" attributeName="display"/>
    <animate fill="freeze" dur="0.1s" begin="hint_button.mouseout"
      from="block" to="none" attributeName="display"/>
</g>
```

Résumé

- Des animations peuvent se déclencher suite à un geste d'utilisateur.
- Le geste de l'utilisateur crée un événement géré (ici) par les balises "begin".
- Autrement dit, le "begin" fait guetteur d'événement pour l'événement nommé.
- Une animation d'un élément peut se déclencher suite à un événement qui a lieu dans un autre élément (on bouge la souris sur l'ellipse et c'est le cercle qui apparaît)

4. Interactivité avec le DOM et EcmaScript

- plus difficile, mais on peut faire ce qu'on veut ...
- marche avec Firefox 1.5.x et Adobe SVG

4.1 Introduction et exemple de sensibilisation

- Certains types d'interactions peuvent se programmer sans savoir grand chose de la programmation DOM (c'est pareil pour le traitement de formulaires HTML avec Javascript ...)
- Dans l'élément qui doit déclencher un script on place un attribut d'événement
  - (re)voir 3.2 "Evénements de base SVG/SMIL" [22]
  - exemple d'attribut: **`onclick`**
- Cet attribut aura 2 valeurs:
  - le nom d'une fonction ECMAScript, par ex: **`circle_click`**
  - un nom d'argument avec lequel sera transmis un objet événement, par ex. **`evt`**
- l'objet événement indiquera au script l'objet (élément) où l'événement a été déclenché etc. Ensuite on peut manipuler cet objet avec des méthodes DOM

```
  function circle_click(evt) {
    var circle = evt.target;
    var currentRadius = circle.getAttribute("r");

....
<circle onclick="circle_click(evt)" cx="300" cy="225" r="100" fill="red"/>
```

Exemple 4-1: Simple switch avec DOM

url: **http://www.yoyodesign.org/doc/w3c/svg1/animate.html**

(home de la traduction française de la spécification SVG 1.0)

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/script01.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/**

```
<script type="text/ecmascript">
<![CDATA[
  function circle_click(evt) {
    var circle = evt.target;
    var currentRadius = circle.getAttribute("r");
    if (currentRadius == 100)
       circle.setAttribute("r", currentRadius*2);
    else
       circle.setAttribute("r", currentRadius*0.5);
    }
  ]]> </script>

<rect x="1" y="1" width="598" height="498" fill="none" stroke="blue"/>

<!-- S'execute a chaque clic -->
<circle onclick="circle_click(evt)" cx="300" cy="225" r="100" fill="red"/>

<text x="300" y="480"
      font-family="Verdana" font-size="32" text-anchor="middle">
    Cliquer sur le cercle change sa taille</text>
```

Explications:

- L'exemple implémente un switch (assimilable à un interrupteur de lumière)
- On définit un event-handler comme attribut du cercle. Autrement dit: en cliquant sur le cercle, la fonction circle_click va être activée.

```
<circle onclick="circle_click(evt)" cx="300" cy="225" r="100" fill="red"/>
```

- La fonction **`circle_click`** qu'on a défini va recevoir comme argument un objet événement (le paramètre **`evt`**)
- L'objet evt peut nous donner l'objet qui a été touché par l'événement. Ici on obtiendra l'objet DOM du cercle et on l'associe avec la variable **`circle`**.

```
function circle_click(evt) {
   var circle = evt.target;
   .....
```

- Ensuite on demande au cercle son attribut "r" (qui définit le radius)

```
var currentRadius = circle.getAttribute("r");
```

- Le code suivant fait simplement une substitution du "r":
  - si le radius == 100, alors on va l'agrandir
  - sinon, on va le diminuer

```
if (currentRadius == 100)
    circle.setAttribute("r", currentRadius*2);
else circle.setAttribute("r", currentRadius*0.5);
```

facile, non ? (ce genre de choses se fait aussi en X3D, en HTML et en VRML)

4.2 Introduction au scripting avec DOM

A. Placement et initialisation du script

- Le script doit se trouver dans une section CDATA (comme en XHTML) !

```
<script type="text/ecmascript">
  <![CDATA[
  function circle_click(evt) {
    .....    }
  ]]>
</script>
```

- Un script travaille souvent avec des variables globales et il faut faire de sorte à ce que l'initialisation soit faite lorsque l'utilisateur interagit avec la scène.
- Il faut explicitement charger une fonction d'initialisation. Voici ma façon de faire:

```
<svg
  <script type="text/ecmascript"> <![CDATA[
  // variables globales qu'on utilise dans toutes les fonctions
  var svgdoc, eye_right, ..... ;

function init (big_bang) {
    svgdoc =    big_bang.target.ownerDocument;          // instance of the document
     .....  }
<svg onload="init(evt)">
    <desc> Ici on insère la véritable scène SVG </desc>
   </svg>
</svg>
```

- L'élément svg imbriqué déclenche le script init() lorsqu'il est chargé

B. Le déboggage

- Comme actuellement les clients n'ont pas de console sophistiquée, il faut trouver soi-même la source des problèmes ....
- Une solution est de semer partout dans votre code des "alert" qui produisent des fenêtres pop-up lorsque la ligne est exécutée.
- Au lieu de les copier/coller tout le temps on propose la solution suivante:
- Une variable globale debug qu'on peut mettre à 0, 1 ou 2 selon vos besoin.
- Si par exemple un problème est résolu, on met le "debug level" à 2. Si debug==1 on ne voit plus l'alert.

```
// set to 0 in production, 1 for things to debug, 2 for things fixed
var debug = 1;
```

- Voici une du code pour produire les "alert"

```
if (debug==2) alert ("The document did load nicely, global variables are defined");
   }
....
// affiche un string plus une représentation d'un objet
if (debug==1) alert ("button =" + currentButton);
```

... sans ce genre de stratégie vous allez souffrir, car il est facile de mal interpréter les spécifications du DOM ....

C. Classes et méthodes SVG DOM et XML DOM 2

- Une liste complète serait monstrueuse, il existe une vingtaine de "chapitres" avec nombreuses classes et méthodes.
- Tout SVG est répliqué dans DOM (donc y compris animations et interactivité)
- A titre d'exemple, il y a 13 classes DOM spécifiques à la structure du document SVG: SVGDocument, SVGSVGElement, SVGGElement, SVGDefsElement, SVGDescElement, SVGTitleElement, SVGSymbolElement, SVGUseElement, ...
- Sinon, SVG implémente DOM2 et nous allons surtout utiliser cela par la suite

L'interface SVGDocument

- un objet SVGDocument va exister quand l'élément racine de la hiérarchie du document XML est un élément 'svg' (cela correspond à HTMLDocument)
- On peut l'utiliser pour chercher des éléments par leur id par exemple.

```
svgdoc = big_bang.target.ownerDocument;// instance of the document
eye_right = svgdoc.getElementById("eyeRightId"); // getElementById()
```

- Attention: Si votre SVG est imbriqué dans un autre document XML (XHTML par exemple) cet objet n'existera pas !

L'interface SVGSVGElement

- SVGSVGElement est l'interface pour l'élément 'svg'. Cette interface contient des méthodes utilitaires variées diverses couramment employées, telles que des opérations matricielles et la capacité de contrôler le moment du ré-affichage sur les appareils de rendu visuel....

D. Eléments utiles du XML DOM2

L'interface Document

- L'interface Document représente le document XML entier. Donc soit le document SVG standalone ou encore le document XML ou HTML dans lequel SVG est imbriqué
- Voir aussi: C. “Classes et méthodes SVG DOM et XML DOM 2” [32]
- Voir aussi: *http://tecfa.unige.ch/guides/tie/html/php-xml/php-xml.html*

L'interface Node

- L'interface Node est le type de données principal pour tous les modèles DOM. Il représente un seul nœud dans l'arbre du document
- Le plupart des classes implémentent la plupart des méthodes (cela veut dire que pour manipuler un élément SVG, vous pouvez utiliser des méthodes "Node").
- Ainsi, les attributs `nodeName`, `nodeValue` et `attributes` forment un mécanisme pour obtenir une information sur un nœud, sans devoir faire appel à des interfaces dérivées spécifiques. (voir exemple suivant)
- Sinon, il existe plein de méthodes pour connaître et manipuler les enfants.

L'interface Element

- L'interface Element représente un élément dans un document XML. Les éléments peuvent avoir des attributs associés.

getAttribute ("string")

- Ramène une valeur d'attribut par son nom

```
element = ... // objet qui représente un élément
element.getAttribute("r")
```

setAttribute ("name", "value")

- Ajoute un nouvel attribut. Si un attribut avec ce nom est déjà présent dans l'élément, sa valeur est changée pour celle du paramètre value

```
element.setAttribute("fill", "yellow");
```

getElementById ("id")

- Retourne l'élément dont l'ID est donné par l'attribut elementId. Si un tel élément n'existe pas, cela retourne null.
- Méthode très importante pour les animations, car elle vous permet d'identifier facilement n'importe quel élément enfant qui a un identificateur (attribut "id").
- Voir Exemple 4-3: "Simple interaction avec DOM" [36]

```
eye_right = svgdoc.getElementById("eyeRightId");
```

L'interface Event

- permet de fournir des informations sur un événement, par ex. l'attribut target

```
function infos(evt) {
   var element = evt.target; .... }
<circle onclick="infos(evt)" .... />
```

Exemple 4-2: DOM alert

- montre quelques méthodes DOM simples

  url: **http://tecfa.unige.ch/guides/svg/ex/svg-dom/dom-alert.svg (animation)**

  url: **http://tecfa.unige.ch/guides/svg/ex/svg-dom/**

```
<script type="text/ecmascript"> <![CDATA[
  function infos(evt) {
    var element = evt.target; XML DOM
    el_name = element.nodeName; // XML DOM2
    el_attributes = element.attributes; // XML DOM2
    circle_size = element.getAttribute("r") // XML DOM2
    display_text = "Element name = " + el_name;
    display_text += " \nAttributes = " + el_attributes;
    display_text += " \nCircle size = " + circle_size;
    alert(display_text);
    element.setAttribute("fill", "yellow"); //XML DOM2
  }
]]> </script>

<!-- S'exécute a chaque clic -->
<circle onclick="infos(evt)" cx="300" cy="225" r="100" fill="red"/>
```

4.3 Exemple DOM simple

Exemple 4-3: Simple interaction avec DOM

url: **http://tecfa.unige.ch/guides/svg/ex/svg-dom/omme-dom1.svg (animation)**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-dom/omme.svg (dessin de départ)**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-dom/**

Note sur la fabrication du dessin

- Le dessin a été fait avec le logiciel InkScape (qui est un très bon logiciel gratuit qui produit du SVG statique), mais il ne fait pas (encore) du dynamique
- Il fallait faire 2-3 retouches:
  - remplacer des éléments CSS par des attributs XML (je ne sais pas comment accéder facilement à un élément CSS dans une longe chaîne via le DOM)
  - changer quelques valeurs (opacité, couleurs)
  - corriger qq. bugs mineurs
  - changer qq. id pour des raisons pédagogiques
- Il faut identifier les id de certains éléments
  - Trop souvent ce type de logiciel produit des path (au lieu d'une cercle par ex.), donc on ne s'y retrouve pas dans le code.
  - Toutefois, on cliquant sur l'objet dans l'éditeur on peut le voir dans l'arbre DOM dans l'éditeur XML. Il suffit de noter l'id sur une feuille de papier ou encore de le changer en une valeur intelligible.

Cet exemple:

- implémente des boutons (à droite) qui changent l'émotion du visage.
- les yeux changent de couleur (on aurait pu faire cela sans DOM)
- différentes "bouches" sont cachées/montrées et pour cela il faut un script à mon avis.

Voici quelques éléments SVG:

- Voici l'oeil gauche produit par Inkscape, vert au départ.

```
<path
    id="eyeLeftId" fill = "#00ff00"
    style="fill-opacity:1;stroke:#000000;stroke-width:2.0000000;stroke-
miterlimit:4.0000000;stroke-opacity:1.0000000"
    transform="translate(-284.0000,-122.0000)"
    d="M 492.00000,311.00000 C 492.00000,330.32997 475.43454,346.00000
455.00000,346.00000 C 434.56546,346.00000 418.00000,330.32997 418.00000,311.00000 C
418.00000,291.67003 434.56546,276.00000 455.00000,276.00000 C 475.43454,276.00000
492.00000,291.67003 492.00000,311.00000 L 492.00000,311.00000 z " />
```

- Voici la bouche "angry", une simple ligne (cachée au départ)

```
<path
 id="angry" visibility="hidden"
 style="fill:none;fill-opacity:0.75000000;fill-rule:evenodd;stroke:#000000;stroke-
width:14.173228;stroke-linecap:butt;stroke-linejoin:miter;stroke-
miterlimit:4.0000000;stroke-opacity:1.0000000"
 d="M 139.22652,280.11050 L 339.77901,278.45304" />
```

- Voici le bouton (rouge) qui déclenche le visage "angry"

```
<rect
    onclick="button_click(evt)"
    id="rect1333"
    style="fill:#ff0000;fill-opacity:1;stroke-width:14.173228"
    y="197.23756"
    x="474.03317"
    height="79.558014"
    width="77.900551" />
```

Les event handlers

- lorsque l'utilisateur clique sur un des rectangles à droite, un script se déclenche

```
onclick="button_click(evt)"
```

Les variables globales du script

```
// variables globales qu'on utilise dans les 2 fonctions
 var svgdoc, eye_right, eye_left, mouth_angry, mouth_happy, mouth_sad ;
```

La fonction d'initialisation sert à trouver les objets qu'on veut animer

```
function init (big_bang) {
    svgdoc =    big_bang.target.ownerDocument;          // instance of the document
    eye_right = svgdoc.getElementById("eyeRightId");    // getElementById()
    eye_left =  svgdoc.getElementById("eyeLeftId");
    mouth_angry = svgdoc.getElementById("angry");
    mouth_happy = svgdoc.getElementById("path1363");
    mouth_sad   = svgdoc.getElementById("path2168");
    }
```

La fonction button_click

- Fonction qui gère les clics des l'utilisateur sur les rectangles

```
function button_click(evt) {
  var button = evt.target;
  var currentButton = button.getAttribute("id");

if (currentButton == "blue_button")  {
      face_change ("blue", mouth_happy );
      }
  else if (currentButton == "rect1333") {
     face_change ("red", mouth_angry);
     }
  else {
     face_change ("black", mouth_sad);
     }   }
```

- La fonction button_click reçoit un objet informatique (le paramètre "evt") qui contient toutes les informations sur le click (et surtout l'objet sur lequel on a cliqué) et qu'on va sauver dans une variable "button" ici.

```
var button = evt.target;
```

  - la propriété "target" de l'objet "evt" référence l'objet sur lequel on a cliqué

```
var currentButton = button.getAttribute("id");
```

  - ensuite on identifie le nom du bouton sur lequel on a cliqué (on lui demande son "id")
- En fonction de l'id du bouton (donc 1 des 3 rectangles) on appelle une fonction face_change qui va transformer le visage.

```
face_change ("black", mouth_sad);
```

- Le premier argument donné est une couleur de cheveux (blue, red ou black).
- Le 2ème est une variable associée à l'objet qu'il faut utiliser pour dessiner la bouche

La fonction face_change

```
function face_change(eye_color, emotion) {
  eye_left.setAttribute("fill", eye_color);
  eye_right.setAttribute("fill", eye_color);
  if (debug==2) alert ("mouth emotion =" + emotion);
  switch (emotion) {
    case mouth_angry:
     mouth_angry.setAttribute ("visibility","visible");
     mouth_happy.setAttribute ("visibility","hidden");
     mouth_sad.setAttribute ("visibility","hidden");
     break;
    case mouth_happy:
      mouth_angry.setAttribute ("visibility","hidden");
      mouth_happy.setAttribute ("visibility","visible");
      mouth_sad.setAttribute ("visibility","hidden");
      break;
    case mouth_sad:
      mouth_angry.setAttribute ("visibility","hidden");
      mouth_happy.setAttribute ("visibility","hidden");
      mouth_sad.setAttribute ("visibility","visible");
      break;
     }
    }
```

- L'instruction switch nous permet en fonction de l'objet "emotion/bouche" transmise de rendre cet objet visible et les autres invisibles
- Note: au lieu de rendre visible/invisible un objet on aurait aussi pu l'enlever/ajouter de la scène
  - Avec une balise "<g>" on peut regrouper des enfants (dessin de la "bouche")
  - Ensuite, avec le DOM, on peut en ajouter ou enlever

5. Animation avec le DOM

Exemple 5-1: Exemple dom01 de la spécification française

url: **http://www.yoyodesign.org/doc/w3c/svg1/animate.html**
(home de la traduction française de la spécification SVG 1.0)

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/dom01.svg**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/**

A notre avis il faut surtout utiliser cette fonctionnalité (au lieu des balises animation de SVG):

- lorsque certaines types d'animation et d'interpolation n'existent pas, par exemple un rectangle qui se promène sur l'écran au hasard (on ne connaît pas son chemin)
- lorsqu'on utilise un navigateur de type Firefox qui n'implémente pas encore les animations.

(explications à faire)