

**REPRÉSENTATION ET  
TRANSFORMATION DES  
GRAPHES CONCEPTUELS**

*Romain Bouleis*

# TABLES DES MATIÈRES

---

Tables des matières.....	2
Introduction.....	3
Présentation du sujet.....	3
DTD.....	4
Représentation en calcul des prédicats.....	6
Représentation en SVG.....	8
Principe.....	8
La feuille de style.....	8
Exemples.....	11
Exemple 1 : Marie envoie un tintin à Jacques.....	11
Exemple 2 : Jacques va à Chambéry en bus.....	12

# INTRODUCTION

---

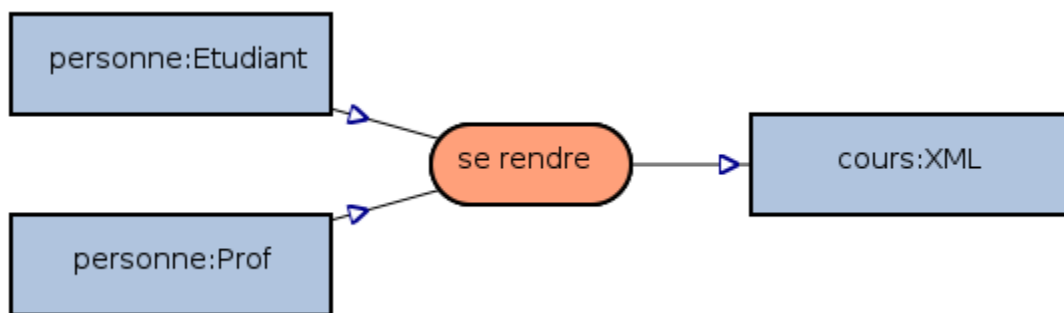
## PRÉSENTATION DU SUJET

La modélisation de phrases peut être faite à l'aide de « Graphes Conceptuels ». Ces graphes peuvent eux-mêmes être représentés par un fichier XML associé à une feuille de style XSL générant une image SVG.

Cette représentation a l'avantage de séparer les informations liées à la phrase et la mise en forme liée au graphe. On pourra ainsi avoir, pour la même phrase, plusieurs représentations possibles.

Le but du travail est de définir une représentation XML des phrases et une mise en forme XSL afin de pouvoir les afficher sous forme de graphes conceptuels et de prédicats.

Un graphe conceptuel est constitué de relations reliés à des concepts. Une relation peut avoir plusieurs concepts en entrées, mais un seul en sortie. Voici un exemple de graphe conceptuel représentant la phrase « Un étudiant et un professeur se rendent au cours de XML » :



*Illustration 1: exemple de graphe conceptuel.*

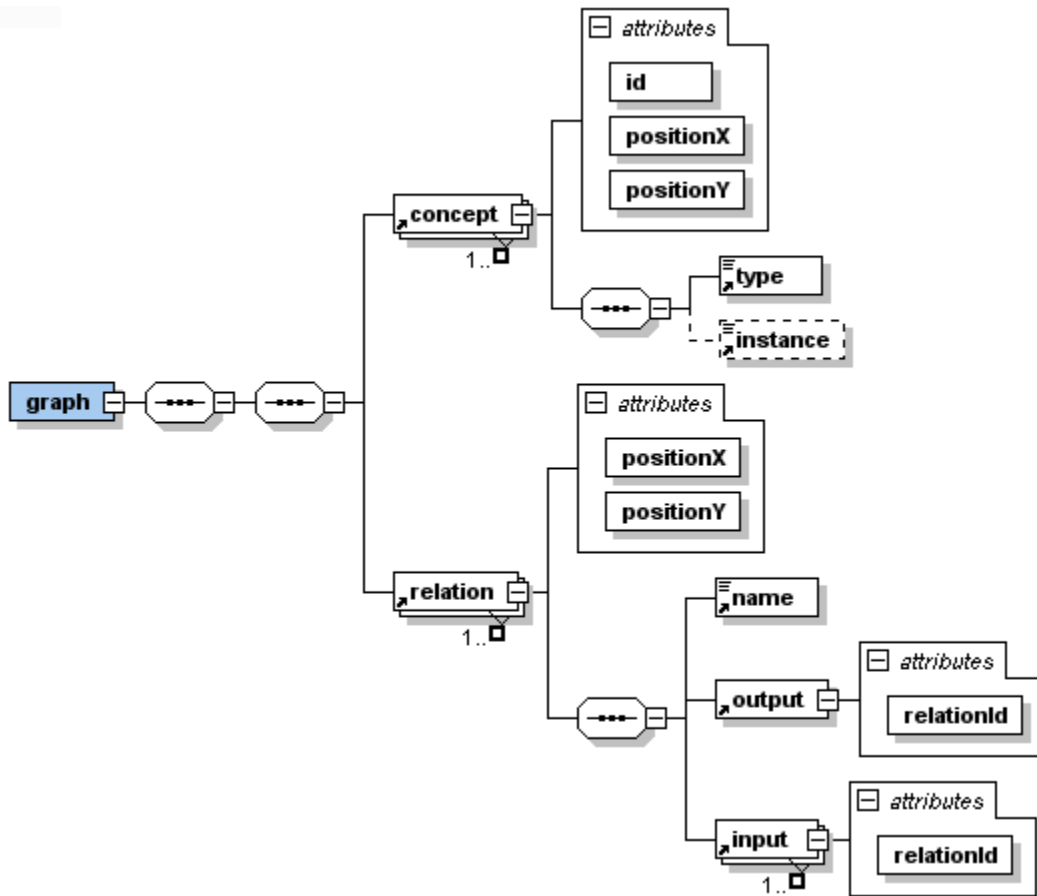
Dans un premier temps, il faut écrire une DTD permettant la représentation de ces graphes.

Puis, une feuille de style XSL afin de transformer un fichier XML sous la forme de calcul des prédicats.

Enfin, une feuille de style XSL permettant de représenter graphiquement le graphe grâce à SVG.

## DTD

Voici l'arbre représentant la hiérarchie du fichier XML désiré :



Remarque :

- Dans l'exemple précédent, un concept est représenté par un rectangle bleu et une relation par un rectangle arrondi saumon.
- Les relations ont un ou plusieurs concepts en entrée et un en sortie.
- Les attributs positionX et positionY servent à la présentation du graphe et non à son contenu. Ils représentent les coordonnées x et y du coin supérieur gauche du rectangle qui contiendra la relation ou le concept.
- Les attributs « relationId » sont les identifiants des concepts en sortie ou en entrée des relations.
- Un concept n'a pas nécessaire d'instance.

Voici maintenant la DTD que l'arbre précédent représente :

```
<!ELEMENT graph (( concept+, relation+ )) >
<!ELEMENT concept ( type, instance?) >
<!ATTLIST concept
    id ID #REQUIRED
    positionX CDATA #REQUIRED
    positionY CDATA #REQUIRED
```

```

>
<!ELEMENT type (#PCDATA) >
<!ELEMENT instance (#PCDATA) >

<!ELEMENT relation ( name, output, input+ ) >
<!ATTLIST relation
    positionX CDATA #REQUIRED
    positionY CDATA #REQUIRED
>

<!ELEMENT output EMPTY>
<!ATTLIST output relationId IDREF #REQUIRED>

<!ELEMENT input EMPTY>
<!ATTLIST input relationId IDREF #REQUIRED>

<!ELEMENT name (#PCDATA) >

```

Pour la suite de ce compte rendu, l'exemple de la première page sera utilisé ; voici sa représentation XML :

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE graph SYSTEM "graphes.dtd">
<graph>
  <concept id="a" positionX="0" positionY="100">
    <type>personne</type>
    <instance>Prof</instance>
  </concept>
  <concept id="b" positionX="0" positionY="0">
    <type>personne</type>
    <instance>Etudiant</instance>
  </concept>
  <concept id="c" positionX="370" positionY="50">
    <type>cours</type>
    <instance>XML</instance>
  </concept>
  <relation positionX="210" positionY="55">
    <name>se rendre</name>
    <output relationId="c" />
    <input relationId="a"/>
    <input relationId="b"/>
  </relation>
</graph>

```

## REPRÉSENTATION EN CALCUL DES PRÉDICATS

Avant d'entrer dans les détails ; voici le résultat que l'on souhaite obtenir (et que l'on obtiendra !)

$$(\exists a:personne)(\exists b:personne)(\exists c:cours) \\ [inst(Prof,a) \wedge inst(Etudiant,b) \wedge inst(XML,c) \wedge se\ rendre(a,b,c)]$$

Tout d'abord, il faut générer la première ligne. Pour ce faire, il suffit de parcourir les concepts et de les afficher sous la forme «  $(\exists id:type)$  ». Ceci est fait grâce à la boucle suivante :

```
<!-- Walk all concepts -->
<xsl:for-each select="/graph/concept">
  <xsl:text>(&#8707;</xsl:text>
  <!-- Display the id -->
  <xsl:value-of select="./@id" />
  <xsl:text>:</xsl:text>
  <!-- Display the type -->
  <xsl:value-of select="./type/text()" />
  <xsl:text></xsl:text>
</xsl:for-each>
```

Il faut ensuite afficher la seconde ligne. Ici, il faudra à nouveau parcourir les concepts, puis les relations. Tout d'abord, les concepts ayant une instance :

```
<xsl:for-each select="/graph/concept">
  <!-- if the concept has an instance -->
  <xsl:if test="./instance">
    <xsl:text>inst(</xsl:text>
    <!-- display the instance -->
    <xsl:value-of select="./instance/text()" />
    <xsl:text>,</xsl:text>
    <!-- display the context id -->
    <xsl:value-of select="./@id" />
    <xsl:text></xsl:text>
    <xsl:text> &#8743; </xsl:text>
  </xsl:if>
</xsl:for-each>
```

Puis les relations :

```
<!-- Walk all relation -->
<xsl:for-each select="/graph/relation">
  <!-- display the relation name -->
  <xsl:value-of select="./name/text()" />
  <xsl:text></xsl:text>
  <!-- Walk all relation's input -->
  <xsl:for-each select="./input" >
    <!-- display the id of the relation -->
    <xsl:value-of select="@relationId" />
    <xsl:if test="not(position()=last())">
      <xsl:text>,</xsl:text>
    </xsl:if>
  </xsl:for-each>
  <xsl:text>,</xsl:text>
  <!--display the id of the output relation in bold -->
```

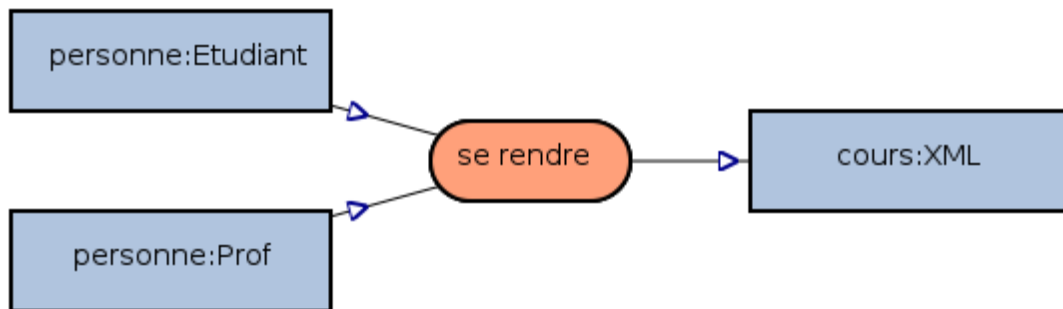
```
<b><xsl:value-of select="./output/@relationId" /></b>
<xsl:text>)</xsl:text>
<xsl:if test="not(position()=last())">
  <xsl:text> ^ </xsl:text>
</xsl:if>
</xsl:for-each>
<xsl:text>]</xsl:text>
```

On peut remarquer qu'il y a deux boucles imbriquées ; la première parcourt les relations, la seconde parcourt les entrées de la relation courante.

Remarque : le nom de la feuille de style est « graphes.xsl ».

# REPRÉSENTATION EN SVG

De la même façon, voici le résultat que l'on souhaite obtenir :



## PRINCIPE

Le fichier XML nous aide un petit peu pour faire ce graphe, il nous donne les coordonnées du coin supérieur gauche des relations et des concepts.

Pour obtenir ce graphe, il y a trois grandes étapes :

- Le dessin des relations
- Le dessin des concepts
- Le dessin des flèches.

Et aussi bizarre que cela puisse paraître, c'est dans le sens opposé que ces éléments seront dessinés. En effet, les flèches doivent être à l'arrière plan, ceci afin que leurs extrémités soient cachées par les concepts et les relations qui viendront par dessus. Une flèche est dite soit « d'entrée » dans le cas où elle pointe vers une relation, soit « de sortie » si elle pointe vers un concept. Quoiqu'il en soit, trois coordonnées sont nécessaires pour les dessiner :

- Le point de départ : c'est le centre d'un concept dans le cas d'une flèche d'entrée, ou le centre d'une relation dans le cas d'une flèche de sortie.
- Le point d'arrivée : c'est le centre d'une relation dans le cas d'une flèche de sortie, ou le centre d'un concept dans le cas d'une flèche d'entrée.
- Le point central : c'est sur ce point que sera dessiné le pointeur. En effet, les flèches étant recouvertes par les concepts et relations dessinés par dessus, le pointeur ne pouvait se trouver à une extrémité.

## LA FEUILLE DE STYLE

J'ai tout d'abord, j'ai défini les tailles des rectangles :

```
<xsl:variable name="conceptsWidth">160</xsl:variable>
<xsl:variable name="conceptsHeight">50</xsl:variable>
<xsl:variable name="relationWidth">100</xsl:variable>
<xsl:variable name="relationHeight">40</xsl:variable>
```

Après quoi, j'ai défini un « marker ». Il correspond en fait au pointeur de la flèche :

```

<!-- fleche -->
<defs><marker
  id="arrow"
  viewBox="0 0 10 10" refX="1" refY="5"
  markerUnits="strokeWidth"
  markerWidth="10" markerHeight="10"
  stroke="darkblue"
  stroke-width="2"
  fill="white"
  orient="auto">
  <path d="M 0 0 L 10 5 L 0 10 z" />
</marker></defs>

```

On peut désormais parcourir les relations afin de dessiner les flèches. Pour chaque relation, il faudra calculer les coordonnées des flèches, par exemple, pour le « x » de départ, le calcul est le suivant :

```

<xsl:variable name="x1">
  <xsl:if test="name() = 'input'">
    <xsl:value-of select="$xcornerConcept + $conceptsWidth div
2"/>
  </xsl:if>
  <xsl:if test="name() = 'output'">
    <xsl:value-of select="../@positionX + $relationWidth div 2"/>
  </xsl:if>
</xsl:variable>

```

Où « \$xcornerConcept » est la valeur X du coin supérieur gauche du concept « vers où va » ou « d'où vient » la flèche.

Pour pouvoir dessiner une flèche, il faut définir un « path ». Un path est une succession de point reliés entre eux. Le pointeur devant être placé sur un point, une simple « line » ne suffit pas puisque seul les points des deux extrémités sont définis.

Le path est défini comme suit :

```

<xsl:variable name="dPath">
  <xsl:text> M </xsl:text>
  <xsl:value-of select="$x1"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="$y1"/>
  <xsl:text> L </xsl:text>
  <xsl:value-of select="((( $x1 ) - ( $x2 ) ) div 2)+$x2"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="((( $y1 ) - ( $y2 ) ) div 2)+$y2"/>
  <xsl:text> L </xsl:text>
  <xsl:value-of select="$x2"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="$y2"/>
</xsl:variable>

```

Les flèches sont enfin prêtes à être dessinées ! Voici ce qui permet d'afficher de les afficher :

```

<path
  transform="translate(10,10)"

```

```
fill="none"
stroke="black"
stroke-width="1"
marker-mid="url(#arrow)"
d="{ $dPath }"
/>
```

Les flèches dessinées, le reste est simple, voici pour les relations :

```
<rect
  transform="translate(10,10)"
  x="{ ../@positionX }"
  y="{ ../@positionY }"
  width="{ $relationWidth }"
  height="{ $relationHeight }"
  style="fill:lightsalmon"
  stroke="black"
  stroke-width="2"
  rx="25"
>
</rect>
<text
  x="{ ../@positionX }"
  y="{ ../@positionY }"
  transform="translate({ $textPositionX }, 32)"
  font-size="14">
  <xsl:value-of select=" ../name/text() "/>
</text>
```

Enfin, de la même manière, les concepts seront dessinés.

# EXEMPLES

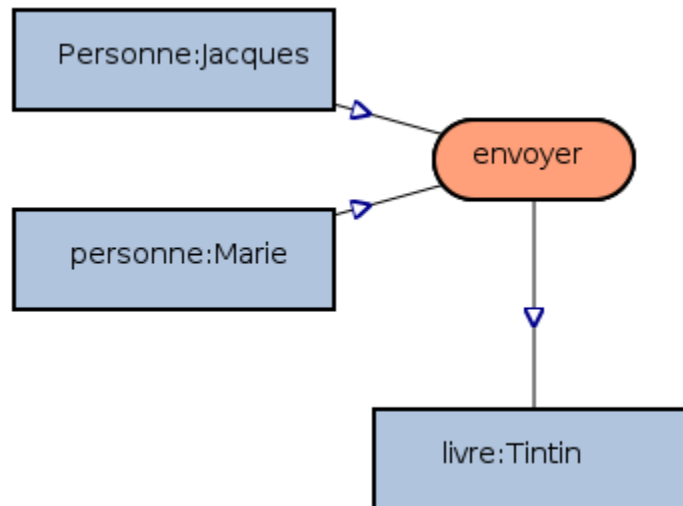
---

## EXEMPLE 1 : MARIE ENVOIE UN TINTIN À JACQUES

graphes\_tintin.xml

$(\exists a:\text{personne})(\exists b:\text{Personne})(\exists c:\text{livre})$

$[\text{inst}(\text{Marie},a) \wedge \text{inst}(\text{Jacques},b) \wedge \text{inst}(\text{Tintin},c) \wedge \text{envoyer}(a,b,c)]$



## EXEMPLE 2 : JACQUES VA À CHAMBERY EN BUS

graphes\_bus.xml

$(\exists a:ville)(\exists b:Personne)(\exists c:bus)(\exists d:aller)$

$[inst(Chambery,a) \wedge inst(Jacques,b) \wedge agent(d,b) \wedge destination(d,a) \wedge instrument(d,c)]$

