heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Gestion de tâche en XML

Deuxième partie : affichage des informations et transformations XSL/Javascript

Florian Zysset
Leeroy Brun

Contents

Introduction

Dans le cadre de notre formation à la HEIG-VD, et durant le module "AppliServ", il nous a été demandé de modéliser en XML un utilitaire de gestion de tâches. Ce projet a été détaillé dans le premier cahier des charges fournis.

Ce document décrit la deuxième partie, qui concerne l'affichage de ces infirmations dans un navigateur web ainsi que la transformation de celles-ci via XSL/Javascript.

Nous avons ensuite dû y ajouter des feuilles de style xsl afin d'afficher élégamment les informations de ce fichier XML.

Ce document décrit ce que nous avons choisi d'afficher, pourquoi et comment nous l'avons fait.

Tous les fichiers ont été testés avec Firefox uniquement.

Ce projet a été réalisé par Florian Zysset et Leeroy Brun.

Cahier des charges

Le but est de permettre l'affichage facile des données contenues dans le fichier XML, ce qui inclus :

- Liste de toutes les tâches
- Liste des tâches pour une personne donnée
- Détails d'une tâche donnée
- Diagramme des tâches ainsi que des relations entre elles

Nous allons décrire maintenant chacun de ces aspects.

Liste des tâches

La liste des tâches permet d'avoir une vue d'ensemble du projet en cours.

Il est important d'y afficher les informations de chaque tâche de façon visuelle :

- Resource assignée à la tâche
- Progrès : affichage d'une barre de progression
- Emplacement
- Durée
- Statut : affiché avec des couleurs reconnaissables, permet d'avoir rapidement une vue d'ensemble

Il est important de pouvoir accéder rapidement au détail de chaque tâche en cliquant sur le titre de celle-ci.

Nous devons aussi pouvoir afficher les tâches assignées à une personne en cliquant sur son nom.

Ceci étant le point d'entrée de notre « application », nous devons également avoir des liens vers :

- Le diagramme des tâches
- Le filtre des tâches pour chaque personne

Liste des tâches pour une personne donnée

Cette page doit permettre de sélectionner une personne et d'en afficher les tâches auxquelles elle est associée.

Cela permet d'avoir rapidement une vue d'ensemble du travail en cours pour une personne sur un projet.

La liste des tâches d'une personne donnée sera présentée de la même manière que la liste des tâches principale.

Détails d'une tâche donnée

Cette page permet d'afficher les détails d'une tâche donnée.

Les informations qui seront affichées sont les mêmes que pour la liste des tâches, ainsi que les commentaires associés à la tâche.

Diagramme des tâches ainsi que des relations entre elles

Il s'agit d'une illustration SVG représentant les différentes tâches du projet, ainsi que des relations qu'elles ont entre elles (les events).

Il est important d'afficher sur chaque relation les conditions de l'événement, afin de savoir ce qui lie chacune des tâches.

Il doit être possible de cliquer sur une tâche pour en afficher les détails.

Implémentation

Cette partie va décrire comment nous allons implémenter chacune des fonctionnalités décrites dans le cahier des charges.

Mécanisme de chargement du fichier XML

Afin d'éviter d'avoir le même fichier XML copié plusieurs fois pour utiliser les différents XSL, nous avons mis en place un mécanisme de chargement de celui-ci.

Ainsi, nous avons un seul fichier XML contenant les données. Dès qu'une modification est affectée dans celui-ci, elle est donc directement visible dans les différentes vues (pages).

Voici un exemple de fichier XML pour une page. Il se contente de charger le fichier XSL voulu.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="tasksList.xsl"?>
<xml/>
```

C'est dans le fichier XSL que nous chargeons ensuite le fichier XML contenant les données dans une variable.

```
<xsl:variable name="taskManager" select="document('tasks.xml')/taskManager" />
```

Lorsque nous effectuons ensuite le premier « apply-template », nous l'effectuons sur le contenu de la variable, ce qui a pour conséquence de traiter les éléments qu'elle contient (notre fichier XML de données).

```
<xsl:template match="/">
        <html><head><link rel="stylesheet" href="../tasks.css" /></head><body>
                <div class="wrapper">
                        <xsl:apply-templates select="$taskManager" />
                </div>
        </body></html>
</xsl:template>
```

Cacher les données non voulues

Nous avons remarqué lors de l'implémentation que si nous ne créions pas de template pour les éléments que nous ne voulions pas afficher, ceux-ci s'affichaient quand même au bas de la page (format « brut »).

La seule manière que nous ayons trouvé pour ne pas les afficher a été de créer des templates vides pour ceux-ci.

Vous trouverez donc à la fin de chaque fichier XSL une partie comme celle-ci pour ignorer les parties non utiles dans la vue en cours.

```
<xsl:template match="events"></xsl:template>
<xsl:template match="resources"></xsl:template>
<xsl:template match="locations"></xsl:template>
```

Liste des tâches

La liste de tâche est le point d'entrée de l'application de gestion de tâche. Pour visualiser cette vue, ouvrez le fichier « tasksList.xml »

Tasks

Description	Resource	Progrès	Emplacement	Durée	Status
Create a task manager	Florian Zysset	80 %	HEIG-VD	200 heures	En cours
Create XML Schema	Florian Zysset	80 %	Home	-	Terminée
Create DTD	Leeroy Brun	20 %	HEIG-VD	-	En cours
Create DOC	Leeroy Brun	4 %		-	Pas commencée

Voir le diagramme des tâches

Voir les tâches assignées à une personne

2 fichiers concernent cette vue :

- tasksList.xsl : Template pour l'affichage
- tasksList.xml : « Loader » XSL (voir chapitre Mécanisme de chargement du fichier XML)

tasksList.xsl

Nous commençons par matcher le template « / » afin de créer la structure minimale d'un fichier HTML.

```
<xsl:template match="/">
		<html><head><link rel="stylesheet" href="../tasks.css" /></head><body>
			<div class="wrapper">
				<xsl:apply-templates select="$taskManager" />
			</div>
		</body></html>
	</xsl:template>
```

Le second template XSL utilisé est au niveau « Tasks », qui va mettre en place la structure du tableau pour la liste des tâches, en appliquant les templates suivants. C'est aussi à cet endroit que nous créons les 2 liens vers les autres fonctionnalités décrites à la suite de ce chapitre.

```
<xsl:template match="tasks">
		<h1>Tasks</h1>
		<TABLE class="tasks-list" cellspacing="0" cellpadding="0">
			<TR class="heading">
				<th class="description">Description</th>
				…
			</TR>

			<xsl:apply-templates />

		</TABLE>
		<p><a href="tasksSVG.xml">Voir le diagramme des tâches</a></p>
		<p><a href="personTasks.html">Voir les tâches assignées à une personne</a></p>
	</xsl:template>
```

Le sous-template suivant est celui concernant les tâches à proprement parler. Chacune des tâches va créer une ligne du tableau.

```
<xsl:template match="task">
		<xsl:element name="TR">
			<TD><a>
				<xsl:attribute name="href">taskDetails.html?taskId=<xsl:value-of
select="./@id"/></xsl:attribute>
				<xsl:value-of select="./description"/>
			</a></TD>
			…
			<td>
				<xsl:if test="(./status/@value='IN_PROGRESS')">
					<xsl:attribute name="style">background-
color:#66CC99</xsl:attribute>
					<xsl:text>En cours</xsl:text>
				</xsl:if>
				…
				<xsl:if test="(./status/@value='NOT_STARTED')">
					<xsl:attribute name="style">background-
color:#D35400</xsl:attribute>
					<xsl:text>Pas commencée</xsl:text>
				</xsl:if>
			</td>
		</xsl:element>
	</xsl:template>
```

Afin d'afficher certaines informations de façon plus « user-friendly », nous avons réalisé des changements de styles en fonction des données du XML.

Par exemple, en fonction du statut, la couleur de fond de la cellule change. Pour cela, nous réalisons une condition sur le nœud « Status » et sa « Value ». Selon la valeur retournée, nous allons modifié l'attribut style, et plus précisément le « background-color ».

```
<xsl:if test="(./status/@value='IN_PROGRESS')">
				<xsl:attribute name="style">background-
color:#66CC99</xsl:attribute>
				<xsl:text>En cours</xsl:text>
			</xsl:if>
			<xsl:if test="(./status/@value='COMPLETED')">
				<xsl:attribute name="style">background-
color:#26A65B</xsl:attribute>
				<xsl:text>Terminée</xsl:text>
			</xsl:if>
			<xsl:if test="(./status/@value='NOT_STARTED')">
				<xsl:attribute name="style">background-
color:#D35400</xsl:attribute>
```

```
                                        <xsl:text>Pas commencée</xsl:text>
                                    </xsl:if>
```

Nous avons aussi réalisé une barre de progression, qui se met à jour automatiquement en fonction de l'avancement de la tâche. La valeur « Progess » de la tâche est assigné à l'attribut « Style », et au mot clé « width » de l'élément « div » pour réaliser cet effet. La valeur est aussi affichée à côté de la barre de progression.

```
<div class="progress-bar-wrapper">
                                        <div class="progress-bar">
                                            <xsl:attribute name="style">
                                                width:<xsl:value-of
select="./progress/@value"/>px;
                                            </xsl:attribute>
                                        </div>
                                    </div>
                                    <div class="progress-value"><xsl:value-of
select="./progress/@value"/> %</div>
```

Pour finir, nous avons les templates vides utilisés pour cacher les informations disponibles dans le fichier, qui ne sont pas nécessaires à cette vue.

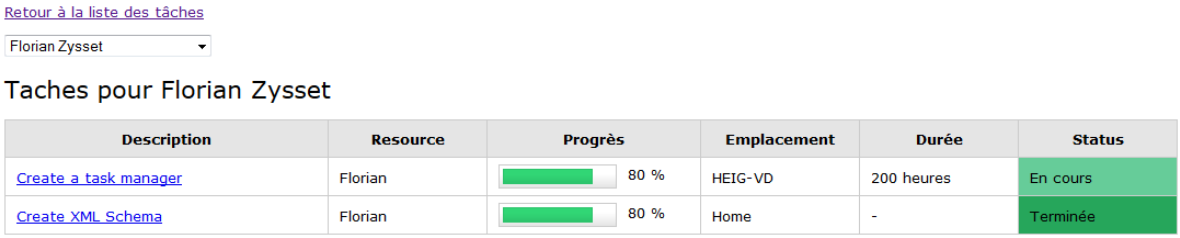
```
<xsl:template match="events"></xsl:template>
        <xsl:template match="resources"></xsl:template>
        <xsl:template match="locations"></xsl:template>
        <xsl:template match="ui"></xsl:template>
```

tasksList.xml

Uniquement utilisé pour charger le fichier XSL afin de gérer l'affichage. (Voir le chapitre correspondant)

Liste des tâches pour une personne donnée

Pour visualiser cette vue, ouvrez le fichier « personTasks.html ».



Les fichiers qui concernent cette vue sont les suivants :

- personsFilter.xsl : génère depuis le fichier XML une liste déroulante <select> avec les personnes
- tasksListPerson.xsl : d'après une variable personId, affiche la liste des tâches de cette personne
- personTasks.html : charge les différents éléments et gère les interactions (onchange)

personsFilter.xsl

Ce fichier génère une liste déroulante de personnes.

```
<xsl:template match="resources">
	<select id="personsFilter">
		<option disabled="disabled" selected="selected">Sélectionnez une personne</option>
		<xsl:apply-templates />
	</select>
</xsl:template>
```

Nous allons commencer par matcher le tag « resources » qui contient toutes les personnes du projet.

Nous créons donc un select avec un attribut ID (pour pouvoir le récupérer depuis le fichier HTML) avec une option par défaut.

```
<xsl:template match="person">
	<option>
		<xsl:attribute name="value"><xsl:value-of select="./@id"/></xsl:attribute>
		<xsl:value-of select="./@firstName"/><xsl:text> </xsl:text><xsl:value-of
select="./@lastName"/>
	</option>
</xsl:template>
```

Nous matchons ensuite tous les tags « person » qui correspondront à chaque personne du projet. Pour chaque personne, nous générons donc une <option> avec en value l'ID de la personne et comme texte affiché son nom et prénom.

```
<xsl:template match="tasks"></xsl:template>
<xsl:template match="events"></xsl:template>
<xsl:template match="locations"></xsl:template>
<xsl:template match="ui"></xsl:template>
```

Pour finir, nous définissons des templates vides pour chacun des éléments que nous ne voulons pas afficher.

tasksListPerson.xsl

Ce fichier va filtrer les tâches en fonction de la valeur d'une variable « personId ».

```
<xsl:variable name="personId">res_1</xsl:variable>
```

On match ensuite l'élément « tasks » pour lequel on va afficher le tableau des tâches avec le nom de la personne en cours.

```
<xsl:template match="tasks">
	<h1>Taches pour <xsl:value-of select="//resources/person[@id =
$personId]/@firstName"/><xsl:text> </xsl:text><xsl:value-of select="//resources/person[@id =
$personId]/@lastName"/></h1>
	<TABLE class="tasks-list" cellspacing="0" cellpadding="0">
		<TR class="heading">
			<th class="description">Description</th>
			<th class="resource">Resource</th>
			<th class="progress">Progrès</th>
			<th class="place">Emplacement</th>
			<th class="duration">Durée</th>
			<th class="status">Status</th>
		</TR>

		<xsl:apply-templates />
	</TABLE>
</xsl:template>
```

Pour chacune des tâches on vérifie ensuite si la ressource qui y est associée correspond à la variable « personId », et si c'est le cas, on l'affiche. Sinon, il ne se passe rien (donc on ne l'affiche pas).

Il les affichera ensuite de la même manière que tasksList.xsl décris plus haut dans le document.

```
<xsl:template match="task">
	<xsl:if test="./assignedTo/@resource = $personId">
	…
</xsl:template>
```

personTasks.html

Ce fichier sert à afficher la liste déroulante générée grâce à personsFilter.xsl, puis lors de la sélection d'un élève dans celle-ci, afficher la liste des tâches auxquelles il est assigné.

Nous avons 3 variables globales dans notre script Javascript : xml, personsFilterXsl, tasksListXsl. Elles permettent de charger les différents fichiers XSL et XML une seule fois au chargement de la page et de les garder en mémoire tout le long du fonctionnement du programme.

```
function loadXML(filename){
	if (window.ActiveXObject){
		xhttp = new ActiveXObject("Msxml2.XMLHTTP");
	} else  {
		xhttp = new XMLHttpRequest();
	}
	xhttp.open("GET", filename, false);
	try {xhttp.responseType = "msxml-document"} catch(err) {} // Helping IE11
	xhttp.send("");
	return xhttp.responseXML;
}
```

La fonction « loadXML » permet de charger un fichier XML/XSL fournis en paramètre et de le retourner.

```
function getParameterByName(name) {
	name = name.replace(/[\[]/, "\\[").replace(/[\]]/, "\\]");
	var regex = new RegExp("[\\?&]" + name + "=([^&#]*)"),
	results = regex.exec(location.search);
	return results === null ? "" : decodeURIComponent(results[1].replace(/\+/g, " "));
}
```

La fonction « getParameterByName » permet de récupérer un paramètre fournis dans l'URL. Ceci nous permet dans ce cas de récupérer un éventuel paramètre « ?personId=xxxxx » dans l'URL pour afficher les tâches d'une personne donnée.

```
// Transforme le XML avec le XSL et le place dans l'élément du dom ayant pour ID "nodeId"
function transformXmlXslIntoNode(xml, xsl, nodeId) {
	var containerNode = document.getElementById(nodeId);
	containerNode.innerHTML = '';

	try {
		// code for Chrome, Firefox, Opera, etc.
		if (window.XSLTProcessor) {
			xsltProcessor = new XSLTProcessor();
			xsltProcessor.importStylesheet(xsl);
			resultDocument = xsltProcessor.transformToFragment(xml, document);
			containerNode.appendChild(resultDocument);
		}
		// code for IE
		else if (window.ActiveXObject) {
			ex = xml.transformNode(xsl);
			containerNode.innerHTML = ex;
		 }
	} catch(e) {
```

```
            return e;
        }
}
```

La function “transformXmlXslIntoNode” prend en parameter un fichier XML, un fichier XSL ainsi que l’id d’un element du DOM dans lequel retourner la transformation du XML par le XSL.

```
// Fonction appelée lorsqu'on veut charger les taches d'une personne
function displayPersonTasks(personId){
        var found = false;

        // Parcours l'arborescence des enfants du noeud principal pour trouver l'emplacement de la variable
        for (var i = 0; i < tasksListXsl.documentElement.childNodes.length && !found; i++) {
                var node = tasksListXsl.documentElement.childNodes[i];

                // Dès qu'on trouve la variable, on la modifie et on arrête la boucle
                if(node.nodeName == 'xsl:variable' && node.getAttribute('name') == 'personId') {
                        node.childNodes[0].nodeValue = personId;
                        found = true;
                }
        }

        // Transforme le XML avec le XSL et le place dans le div #listeTaches
        transformXmlXslIntoNode(xml, tasksListXsl, 'tasksList');
}
```

La fonction displayPersonTasks est appelée lors de l’événement « onchange » sur la liste (défini plus bas dans le document). Il prend en paramètre l’ID d’une personne et transformera le fichier XML d’après le fichier XSL « tasksListPerson.xsl ». Elle permet d’afficher la liste des tâches d’une personne donnée.

Nous devons commencer par rechercher la variable « personId » dans les enfants du nœud racine (son nom est « xsl:variable » et son attribut « name » correspond à « personId »). Une fois qu’on le trouve, on remplace son contenu par l’ID de la personne fournie en paramètre de la fonction et on arrête la recherche.

Il nous suffit ensuite d’appeler transformXmlXslIntoNode pour transformer le XML avec le XSL et placer le résultat dans le div d’id “tasksList”.

```
function displayPersons(){
        xml = loadXML("tasks.xml");
        personsFilterXsl = loadXML("personsFilter.xsl");
        tasksListXsl = loadXML("tasksListPerson.xsl");

        transformXmlXslIntoNode(xml, personsFilterXsl, 'personsFilterWrapper');

        // Est-ce qu'on a un personId en paramètre de l'URL ?
        var personId = getParameterByName('personId') || null;
```

```
        // Si oui, on charge déjà sa liste de taches
        if(personId) {
                displayPersonTasks(personId);

                // Ensuite on sélectionne la personne dans la liste
                selectPersonInList(personId);
        }

        // Lorsqu'on sélectionne une valeur dans la liste de personnes, on charge sa liste de taches
        document.getElementById('personsFilter').onchange = function() {
                displayPersonTasks(document.getElementById('personsFilter').value);
        };
}
```

La function "displayPersons" est appelée au chargement de la page. Elle permet d'afficher la liste déroulante pour sélectionner la personne de qui l'on souhaite afficher les tâches.

Elle commence par charger les fichier XSL/XML dans les variables globales. Cela permet de ne charger d'une seule fois ces fichiers pour optimiser les performances.

Il transforme ensuite le fichier XML avec le fichier XSL pour récupérer la liste déroulante des personnes. Il place ce résultat dans le div ayant pour ID « personsFilterWrapper ».

Il vérifie ensuite si nous avons passé un paramètre dans l'URL pour définir la personne à afficher, si c'est le cas, on appelle directement la fonction « displayPersonTasks » avec l'ID fournis pour afficher les tâches de la personne. On appelle également la fonction « selectPersonInList » qui va permettre de sélectionner la personne passée en paramètre dans la liste déroulante.

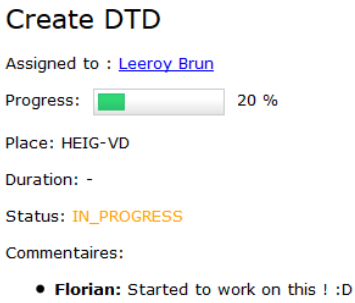
Pour finir, il définit l'événement « onchange » de la liste déroulante. Lorsque celui-ci est appelé, il va appeler « displayPersonTasks » avec la valeur sélectionnée dans la liste (qui correspond à l'id de la personne de qui afficher les tâches).

```
<body onload="displayPersons()">
        <div class="wrapper">
                <p><a href="tasksList.xml">Retour à la liste des tâches</a></p>
                <div id="personsFilterWrapper"></div>
                <div id="tasksList"></div>
        </div>
</body>
```

Ceci est le code du body de la page web avec les différents éléments dans lesquels les informations seront chargées par le Javascript.

Détails d'une tâche donnée

Pour afficher cette vue, ouvrez le fichier « tasksDetails.html »



Afin d'afficher le détail d'une seule tâche, nous utilisons 3 fichiers distincts :

- taskDetails.html : contient la page de base ainsi que le javascript nécessaires
- tasks.xml : contient les données
- taskDetails.xsl : contient la logique pour l'affichage d'une tâche.

taskDetails.html

Lorsque le chargement du « body » est fait, nous avons implémenté plusieurs étapes.

La première étape du javascript est de charger le fichier XML, ainsi que le fichier XSL. Pour cela nous avons créé une fonction simple permettant de charger les 2 fichiers.

```
function loadXMLDoc(filename){
        if (window.ActiveXObject){
                xhttp = new ActiveXObject("Msxml2.XMLHTTP");
        } else  {
                xhttp = new XMLHttpRequest();
        }

        xhttp.open("GET", filename, false);
        try {xhttp.responseType = "msxml-document"} catch(err) {} // Helping IE11
                xhttp.send("");
                return xhttp.responseXML;
        }
}
```

La fonction « DisplayResult » est donc la suivante à cet instant

```
function displayResult(){
        xml = loadXMLDoc("tasks.xml");
        xsl = loadXMLDoc("taskDetails.xsl");
```

Dès lors que les fichiers XML sont chargés, particulièrement le XML, nous récupérons la valeur passée en paramètre, qui représente l'ID d'une tâche. Le paramètre se trouve dans l'url, par exemple :

…HEIGVD_AppliSrv_Projet/UI/taskDetails.html?taskId=task_2

Lorsque nous avons cet ID (ou s'il n'est pas passé, nous utilisons l'ID « task_1 »), nous allons rechercher dans le fichier XSL la variable qui contient cette valeur, afin de limiter l'affichage à cette tâche uniquement (voir le fichier XSL pour plus d'information)

```
var taskId = getParameterByName('taskId') || 'task_1'; // On récupère le paramètre d'URL

var found = false;

// Parcours l'arborescence des enfants du noeud principal pour trouver l'emplacement de la variable
for (var i = 0; i < xsl.documentElement.childNodes.length && !found; i++) {
        var node = xsl.documentElement.childNodes[i];

        // Dès qu'on trouve la variable, on la modifie et on arrête la boucle
        if(node.nodeName == 'xsl:variable' && node.getAttribute('name') == 'taskId') {
                node.childNodes[0].nodeValue = taskId;
                found = true;
        }
}
```

La dernière étape consiste à appeler le moteur de rendu XML/XSL, afin d'afficher le résultat du XSL modifié en mémoire.

```
// code for IE
if (window.ActiveXObject || xhttp.responseType == "msxml-document") {
        ex = xml.transformNode(xsl);
        document.getElementById("example").innerHTML = ex;
 }
// code for Chrome, Firefox, Opera, etc.
else if (document.implementation && document.implementation.createDocument) {
        xsltProcessor = new XSLTProcessor();
        xsltProcessor.importStylesheet(xsl);
        resultDocument = xsltProcessor.transformToFragment(xml, document);
        document.getElementById("example").appendChild(resultDocument);
 }
```

taskDetails.xsl

Le fichier XSL va permettre le rendu d'une tâche XML prédéfinie, et passée en paramètre comme expliqué précédemment.

```
<xsl:variable name="taskId">task_1</xsl:variable>
```

Tout d'abord, nous avons défini une variable nommée « TaskID », qui va être modifiée par le javascript (comme expliqué précédemment), afin de filtrer le XML et ne prendre en compte que les données voulues.

```
<xsl:template match="/">
	<html><head><link rel="stylesheet" href="tasks.css" /></head><body>
		<div class="wrapper">
			<xsl:apply-templates />
		</div>
	</body></html>
</xsl:template>
```

Ensuite nous définissons le template matchant « / », pour mettre la structure de base de HTML, ainsi que la « stylesheet » correspondante.

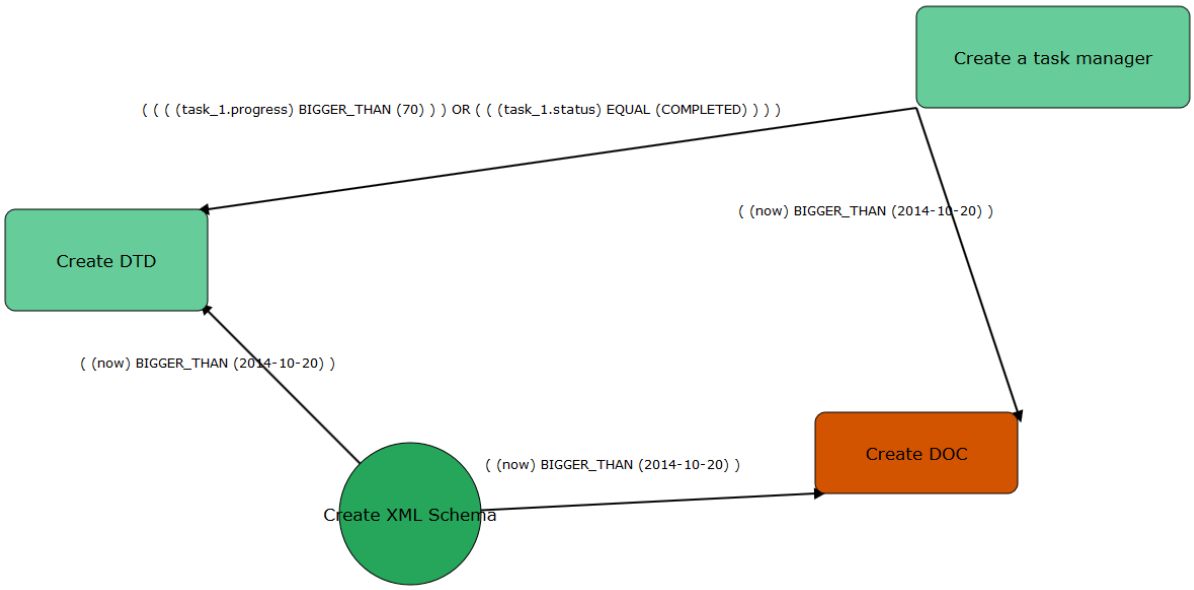
```
<xsl:template match="//tasks/task[@id = $taskId]">

	<h1><xsl:value-of select="./description"/></h1>
	<p>Assigned to :
		<a>
			<xsl:attribute name="href">personTasks.html?personId=<xsl:value-of
select="./assignedTo/@resource"/></xsl:attribute>
			<xsl:value-of select="//resources/person[@id =
current()/assignedTo/@resource]/@firstName"/>
			<xsl:text> </xsl:text>
			<xsl:value-of select="//resources/person[@id =
current()/assignedTo/@resource]/@lastName"/>
		</a>
	</p>
	….
	</p>
	<xsl:if test="count(./comments/comment) > 0">
		<p>Commentaires:</p>
		<ul>
			<xsl:for-each select="./comments/comment">
				<li><b><xsl:value-of select="//resources/person[@id =
current()/@from]/@firstName"/>:</b><xsl:text> </xsl:text><xsl:value-of select="."/></li>
			</xsl:for-each>
		</ul>
	</xsl:if>
</xsl:template>
```

Lorsque la tâche « current » correspond à l'ID de la variable « TaskID », nous affichons les données pour cette tâche, tel que la personne à qui elle est assignée, et les commentaires.

Les commentaires ne seront affichés que lorsque le nombre de commentaire est plus grand que « 0 », afin d'éviter d'afficher une section vide, avec le titre qui va avec.

Diagramme des tâches ainsi que des relations entre elles

Pour afficher cette vue, ouvrez le fichier « tasksSVG.xml » dans Firefox.



Nous avons eu différents « challenges » pour la mise en place de ce diagramme, nous allons vous les détailler ci-dessous.

Création d'un fichier SVG à la volée à l'aide de XSL

Le fichier SVG n'était pas reconnu comme tel par le navigateur, nous avons donc dû ajouter des déclarations au début du fichier XSL pour le faire reconnaître.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
        <xsl:output method="xml" indent="yes" standalone="no" doctype-public="-//W3C//DTD SVG
1.1//EN" doctype-system="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" media-
type="image/svg" />
```

Création des liens entre les tâches grâce aux événements

Les événements permettent de définir une ou plusieurs tâches de départ et une ou plusieurs tâches d'arrivée, nous avons donc dû gérer ces différents cas de figure afin de créer tous les liens entre les tâches.

Pour cela nous avons mis en place des templates récursifs :

- **outputTasksFromLinks**
  C'est le premier template appelé. Il permet de traiter une liste de tâches de départ et d'appeler pour chacune d'elle le deuxième template qui se chargera de créer les liens avec les tâches d'arrivée.
  Il prend en paramètre une liste de tâches de départ « fromList », une liste de tâches d'arrivée « toList » et le nœud de l'événement concerné« eventNode », qui servira à traiter l'expression postfixée.

- **outputTasksToLinks**
  Ce template est appelé pour chaque tâche de départ avec une liste de tâches d'arrivée. Il va se charger de créer le lien entre la tâche de départ et les différentes tâches d'arrivée. Il va également calculer la position où placer le départ et l'arrivée du lien en fonction de l'emplacement des tâches (gauche, droite, haut, bas). Pour finir, il va appeler un autre template recursif qui se chargera de traiter l'expression postfixée de l'événement pour la transformer en expression infixée, qui sera alors affichée par dessus la ligne qui relie les tâches.

Ces templates sont ensuite appelés comme ceci dans le fichier XSL :

```
<xsl:template match="events/event">
        <xsl:call-template name="outputTasksFromLinks">
                <xsl:with-param name="fromList" select="./from/@task"></xsl:with-param>
                <xsl:with-param name="toList" select="./to/@task"></xsl:with-param>
                <xsl:with-param name="eventNode" select="."></xsl:with-param>
        </xsl:call-template>
</xsl:template>
```

Transformation de l'expression postfixée en expression infixée

Chaque événement défini des conditions pour sa réalisation de façon postfixée. Afin de les afficher sur le lien entre les tâches, nous avons dû les parser et les transformer en expression infixée, plus facilement compréhensible par les humains.

Voici un exemple de conditions postfixées :

```
<operande>
    <value>COMPLETED</value>
</operande>
<operande>
    <taskValue task="task_1"
element="status"/>
</operande>
<operator name="EQUAL"/>
<operande>
    <value>70</value>
</operande>
<operande>
    <taskValue task="task_1"
element="progress"/>
</operande>
<operator name="BIGGER_THAN"/>
<operator name="OR"/>
```

Elle sera alors transformée et affichée comme ceci :

```
( ( ( (task_1.progress) BIGGER_THAN (70) ) )
OR ( ( (task_1.status) EQUAL
(COMPLETED) ) ) )
```

Nous avons deux templates qui permettent cette transformation :

- **parsePrefixedExpression**
  Ce template prend en paramètre une liste de nœuds « operande » et « operator ». Il va ensuite tous les traiter de façon recursive, en prenant à chaque fois le dernier nœud de la liste comme nœud courant. Si c'est un operateur, il va traiter le premier paramètre, afficher l'opérateur, puis traiter le deuxième paramètre (en évitant bien sûr les « sous-éléments » du premier paramètre). Si c'es une opérande, nous affichons les différentes valeurs en fonction de si il s'agit de <now>, <value> ou <taskValue>.

- **countPrefixedExpressionDepth**
  Ce template prend en paramètre une liste de nœuds « operande » et « operator ». Il va ensuite compter le nombre d'éléments qui composent cette expression. Il permet à parsePrefixedExpression de savoir combien de nœud il doit ignorer lors du traitement du deuxième paramètre d'un opérateur (car ils appartiennent au premier paramètre).

Affichage des tâches en couleur

Les tâches sont affichées de couleur différente en fonction de leur status. Ceci permet d'avoir rapidement une vue d'ensemble du projet.

Pour cela, nous plaçons la couleur dans une variable en fonction du status. Elle est ensuite utilisée pour renseigner l'attribut « fill » des éléments du diagramme SVG.

Conclusion

En conclusion, nous somme satisfait du résultat de notre projet. Visuellement l'application est agréable, et surtout, fonctionnelle.

Les difficultés rencontrées ont surtout étés au niveau des templates récursifs pour l'affichage des relations entre les tâches en SVG et la transformation des expressions post-fixées en expression infixée. Etant habitués au Javascript/CSS/HTML, cette partie ne nous a pas posé de gros problèmes.

La seule fonctionnalité réellement manquante est l'ajout des tâches. De par le fait que nous n'avons qu'un seul fichier XML contenant les données, cela pourrait facilement être mis en place en évitant de compromettre l'intégrité des données.

Parmi les améliorations possibles de notre projet nous pouvons citer :

- Ajout/suppression des tâches et des ressources liées grâce à une interface
- Système permettant de déplacer les éléments du SVG, afin de modifier les positions des éléments.