Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS
LOG792 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

Réalisation d'une interface interactive pour l'analyse d'impact environnemental

PHILIPPE PIGEON
PIGP08058908

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Alain April

MONTRÉAL, 13 DÉCEMBRE 2016
AUTOMNE 2016

 Philippe PIGEON, 2016

Cette licence Creative Commons signifie qu'il est permis de diffuser, d'imprimer ou de sauvegarder sur un autre support une partie ou la totalité de cette œuvre à condition de mentionner l'auteur, que ces utilisations soient faites à des fins non commerciales et que le contenu de l'œuvre n'ait pas été modifié.

REMERCIEMENTS

Merci à Mathieu Dupuis, pour le temps qu'il a pris pour m'introduire et m'aviser lors de mon projet de fins d'études.

RÉALISATION D'UNE INTERFACE INTERACTIVE POUR L'ANALYSE D'IMPACT ENVIRONNEMENTAL

PHILIPPE PIGEON
PIGP08058908

RÉSUMÉ

Ce projet a pour objectif de concevoir une interface utilisateur qui présente et permet de modifier les informations des processus apparaissant dans l'arbre de processus impliqué dans le modèle d'analyse de cycle de vie d'un bâtiment.

Une méthode agile est utilisée, comportant des itérations d'une semaine, quia débuté en octobre 2016 et s'est terminé en décembre 2016. À la fin de chaque itération, une rencontre a été effectuée avec le client. Le travail réalisé durant l'itération lui est présenté et le sujet de la prochaine itération est discuté.

Le résultat de ce projet a permis d'accomplir ne grande partie des objectifs initiaux, soit la présentation de l'information et la modification de celles-ci. Les solutions trouvées pour répondre aux problèmes rencontrés, lors de la conception de l'interface, satisfont le client.

Pour la suite, une attention particulière devrait être portée sur l'apparence visuelle de l'interface. Ensuite des détails tels que l'affichage des types de processus dans l'arbre et certaines règles d'affaires devra être implémentés. Finalement, il serait intéressant de commencer l'affichage des résultats obtenus.

Mots-clés : développement durable, analyse de cycle de vie, interface utilisateur, JavaScript.

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

JSON – JavaScript object notation. Un format de donnée pouvant contenir des structures de données diverses accessibles via une propriété, ainsi que du code sous forme de méthode.

REST – Representation state transfer. Un service Web permettant l'interaction et le transfert de données sous forme de texte en accédant a diverses URL avec les méthodes correspondantes.

ACV/LCA – Analyse de cycle de vie / Life-cycle analysis.

INTRODUCTION

Il y a peu de logiciels qui permettent de faire l'analyse de cycle de vie (ACV), et ceux qui existent sont complexes lents. Cependant, les considérations écologiques sont grandissantes dans le domaine de la construction.

Ce projet a pour but de concevoir une interface utilisateur pour un projet de recherche « UBUBI » visant à remplir la demande pour les produits d'ACV. Dans le cadre de ce projet, il sera question de produire une interface qui affiche les informations du modèle ACV et qui permet aussi la modification de ceux-ci.

L'interface devra pouvoir gérer l'affichage et la modification de la grande quantité d'information associée au domaine ACV, ainsi qu'être extensible afin de pouvoir continuer à rajouter des fonctionnalités lorsque le mandat en question sera terminé.

Dans un premier temps, le contexte, l'objectif et la méthodologie du projet seront détaillés. Suivant ceci, le processus d'analyse sera décrit, suivi des étapes de conception requises pour atteindre les objectifs établis.

CHAPITRE 1

CONTEXTE ET OBJECTIFS DU PROJET

Le domaine de la construction est présentement en plein changement et plusieurs nouveaux aspects influencent la conception d'un bâtiment. Un des aspects de plus en plus mis de l'avant est l'empreinte écologique d'un bâtiment. Cependant, les logiciels de calculs actuels permettant de faire de l'analyse de cycle de vie afin de connaitre l'empreinte écologique des produits qui le compose sont très complexes, peu performants et peu adaptés aux travaux des architectes, ce qui nécessite de nouveaux outils de calculs modernes.

Ce projet consiste à analyser et concevoir une interface utilisateur Web pour un nouveau logiciel libre d'analyse de cycle de vie d'un bâtiment. Il est basé sur deux projets de recherche effectuée en parallèle, à l'ÉTS, qui fourniront les données nécessaires à l'interface.

Le défi principal se situe dans la présentation d'un arbre de processus requis à l'analyse environnementale. Cet arbre contient plus de 8000 processus (c.-à-d. des nœuds) et chaque processus contient plusieurs propriétés qui affectent l'impact environnemental du processus et de ses sous-processus. De plus, l'utilisateur doit pouvoir manipuler l'information, par exemple changer des propriétés et même la structure de l'arbre, afin d'optimiser l'empreinte environnementale du bâtiment.

L'objectif principal de ce projet est de concevoir un logiciel front-end, basé sur des services REST, simples, intuitifs et adaptés à des analystes d'impact environnemental de bâtiment. Le logiciel devra gérer une quantité massive de données sur l'impact écologique d'un bâtiment et devra simplifier la manipulation et la consultation, par des architectes, des données affichées.

CHAPITRE 2

MÉTHODOLOGIE

2.1 Gestion du projet

Le projet est divisé en sous-tâches exécutées d'une manière incrémentale : l'affichage en arbre des processus, l'affichage des informations d'un processus individuel, et finalement la modification de celles-ci.

Une méthode agile « SCRUM » est utilisée pour le développement de ce projet. Les itérations s'échelonnent sur 1 à 2 semaines, selon les fonctionnalités à implémenter. Toutes les itérations ont les mêmes étapes, soit :

1. Analyse : choix des nouvelles fonctionnalités à implémenter durant l'itération. Analyse fonctionnelle des nouvelles fonctionnalités;
2. Développement : implémentation des nouvelles fonctionnalités;
3. Tests et démonstration : effectuer des tests ad hoc pour vérifier le prototype, faire la validation du prototype auprès du client et faire un compte-rendu, et ajustement selon les commentaires du client.

Une rencontre « SCRUM » est effectuée chaque vendredi. L'objectif de cette réunion est de confirmer que ce qui a été fait correspond aux attentes du client, identifier les éléments bloquants et permet de planifier la prochaine semaine.

Pour guider la discussion, un « Compte-rendu » de la semaine est envoyé aux participants. Celui-ci contient une liste de tâches effectuées, l'effort requis, et une démonstration visuelle ou codifiée du progrès (c.-à-d. une capture d'écran ou du code source). À ceci s'ajoutent une liste d'éléments bloquants et des questions à aborder lors de la rencontre. Finalement il y a une section « Plan d'action » qui est remplie lors de la rencontre, elle détaille les tâches à effectuer lors de la prochaine itération (qui s'échelonne sur une semaine).

Ce document sert aussi de journal de bord nécessaire pour les objectifs académiques du projet de fin d'études.

2.2 Outils utilisés

Afin d'effectuer le développement logiciel, un ensemble d'outils est nécessaire. Les tâches de développement nécessitant un outil afin de mieux les supporter sont :

- Le contrôle de version;
- Les tests et validations;
- La modification du code source.

Pour la gestion de la configuration, Git et SourceTree sont utilisés pour faire le suivi des modifications. Pour la validation, le navigateur Google Chrome est utilisé pour tester et valider l'interface. Pour la modification du code source, l'IDE est IntelliJ. Celui-ci permet de compiler et d'exécuter le serveur UBUBI en mode test.

CHAPITRE 3

ANALYSE ET CONCEPTION

3.1 Analyse des technologies

La partie analyse de ce projet a été brève puisque nous nous basons sur un logiciel existant, OpenLCA, des requis que le client avait élaborés (réf. [1] et [2]) et un prototype partiel non fonctionnel de l'application. De plus, puisque ce projet imbriqué dans l'application UBUBI qui utilise le cadriciel PlayFramework, les choix technologiques majeurs (choix de la plateforme et du langage) est déjà fait. Ceci permet de commencer le développement de l'application plus rapidement.

Cependant, il y a plusieurs choix à faire concernant les librairies et cadriciels à utiliser pour ce projet.

Dans un premier temps, une recherche est faite pour trouver un cadriciel qui faciliterait la manipulation et l'affichage de l'information. Les deux candidats retenus sont React et AngularJS version 2. Pour faire le choix entre l'un d'eux, le temps d'apprentissage, la popularité, la quantité de documentation disponible, et la performance, à un certain degré, sont les critères déterminants. Faisant des recherches avec ces critères en tête, AngularJS s'est distingué comme étant le cadriciel le plus populaire, et donc ayant plus de support. De plus, AngularJS est décrit comme étant un cadriciel « prêt a porté », tandis que React est plus comme une boite a outil que le développeur construit a fur et a mesure (House, C, réf. [3]). Ceci est donc un avantage pour AngularJS en termes de temps d'apprentissage. Et finalement, AngularJS version 2 a amélioré la performance comparée à la version 1, le mettant sur le même niveau de performance que React.
Donc, le cadriciel AngularJS 2 a été choisi.

Ensuite, il faut une librairie pour afficher un arbre. Une option possible est de faire un arbre soit même, sans utiliser de librairies externes, mais le temps et le risque requis pour faire ceci sont trop considérables. Deux librairies pour l'affichage se sont présentées après quelque temps de recherche. La première, boostrap-ui-treeview, est une librairie qui utilise AngularJS pour afficher une représentation hiérarchique d'un item. N'étant pas familier avec AngularJS, il est difficile de vérifier si cette librairie répond aux besoins du projet. La deuxième librairie, jsTree, utilise jQuery, un cadriciel familier et très populaire, qui est déjà inclus dans le projet UBUBI. L'utilisation de jQuery et l'abondance d'exemples et de documentation sur la page de jsTree, ainsi que la possibilité de remplir l'arbre avec une fonction ce qui permettrait le chargement paresseux, fait de jsTree un bon candidat pour ce projet.
Donc, la librairie jsTree a été choisie pour afficher l'arbre.

En ce qui concerne l'affichage graphique, le cadriciel Bootstrap a été choisi pour les éléments HTML de base, simplement pour sa popularité, sa familiarité, et par la recommandation du client. La librairie d3js a été aussi sélectionnée comme librairie pour afficher des représentations graphiques complexes (p.e une matrice ou un diagramme de Sankey) par sa familiarité et par la recommandation du client.

3.2 Intégration au projet UBUBI existant

Au début de ce projet, il était question d'intégrer le code d'interface utilisateur dans le projet UBUBI existant, pour faire en sorte qu'il n'y ait pas deux processus à exécuter, un pour le serveur REST et un pour le serveur Web, et pour que tout le code soit dans le même projet, ce qui faciliterait la maintenance.

Donc, par ce fait, avant de pouvoir commencer à concevoir l'interface, il y a eu une période d'apprentissage concernant le code existant. Ceci avait pour but de :

- Prendre connaissance des services REST présents via le code
- Apprendre comment inclus des dépendances au projet, par exemple jsTree ou AngularJS
- Apprendre comment PlayFramework et Scala interagissent pour servir une page Web
- Apprendre le structure du projet pour pouvoir rapidement réagir aux modifications côté serveur, par exemple l'ajout ou la modification d'un service REST

Ayant complété cette phase d'apprentissage, la création de la page Web qui servira à présenter l'interface utilisateur peut commencer.

3.3 Accès au service REST

Pour accéder aux informations du projet UBUBI, il faut utiliser le service REST offert.
Au début du projet, cette opération était faite en utilisant jQuery pour créer une requête AJAX vers une URL correspondant à une ressource REST, et ensuite utilisant cet objet AJAX ainsi que les promesses de jQuery pour obtenir une réponse en format JSON.

```
getModelAjax = function(modelId) {
   return $.ajax({
      type: "GET",
      url: `/api/model/BIMModels/${modelId}`,
      contentType: 'application/json; charset=utf-8',
      timeout: 120000
   });
};

getModelAjax(1).done(function(modelJson) {
   var model = JSON.parse(modelJson);
   //utilisation du modèle.
});
```

Rapidement, avoir des URL statiques vers les ressources REST n'était pas pratique. Puisque le coté serveur est aussi en développement, ces URL peuvent changer a n'importe quel moment.

Donc une méthode générique pour générer un objet AJAX ainsi qu'un router JavaScript ont été implémentés.

Ce routeur JavaScript[1] est un objet JavaScript créé par PlayFramework qui sert ici à obtenir un objet contenant l'URL et la méthode d'appel pour une ressource REST correspondant au type et a l'identifiant désiré.

Par exemple l'appel au routeur pour supprimer un échange dont l'identifiant est « 282888 »:

jsModel.controllers.API.model.Exchange.delete(282888)

Ceci retourne un objet JSON avec plusieurs propriétés, notamment l'URL de la ressource REST, */api/model/exchanges/282888*, et le type de requête requise pour effectuer cet appel, ici étant *DELETE*.

Ce routeur permet de minimiser les références statiques vers les URL des ressources REST.

Donc, équipée du routeur et d'une méthode générique, il est possible de faire la même opération illustrée plus haut comme ceci :

[1] Plus d'information disponible au sujet des routeurs JavaScript dans PlayFramework est disponible ici : https://www.playframework.com/documentation/2.6.x/ScalaJavascriptRouting

```
getGenericAjax = function(url) {
   return $.ajax({
      type: "GET",
      url: url,
      contentType: 'application/json; charset=utf-8',
      timeout: 120000
   });
};
var modelUrl = jsModel.controllers.API.model.BIMModel.getInfo(1).URL;
getGenericAjax(modelUrl).done(function(modelJson) {
   var model = JSON.parse(modelJson);
   //utilisation du modèle.
});
```

Pour diminuer encore plus des références vers les URL des ressources REST, les objets retournés par ces services contiennent les URL vers les ressources associées, au lieu d'un ID correspondant à une clé étrangère.
Donc au lieu de devoir prendre un ID contenu dans l'objet et le passé au routeur, il est possible de simplement utiliser l'URL contenue dans l'objet pour obtenir la ressource associée.
Par exemple *exchange.producer* ne contient pas l'ID du processus (*12345*), mais l'URL vers ce processus (*/api/model/processes/12345*).

3.4 Conception du modèle de données

La première étape de ce projet consiste à utiliser le service REST offert par le serveur UBUBI pour construire un modèle de données. Ayant rencontré le client et pris connaissance de la structure des donnés et de ce qui devrait être affiché a l'utilisateur, il faut maintenant

parcourir la structure des données en utilisant les ressources REST pour pouvoir construire un modèle de données contenant les informations a affiché.

En premier lieu il a été établi que l'utilisateur ne devrait voir que les processus dans l'arbre, et non les échanges ou les flux. Donc, en partant du processus racine (*rootProcess*), il faut parcourir les liens entre les processus et les échanges, et ne stocker que les informations des processus, tout en conservant la hiérarchie de ceux-ci.

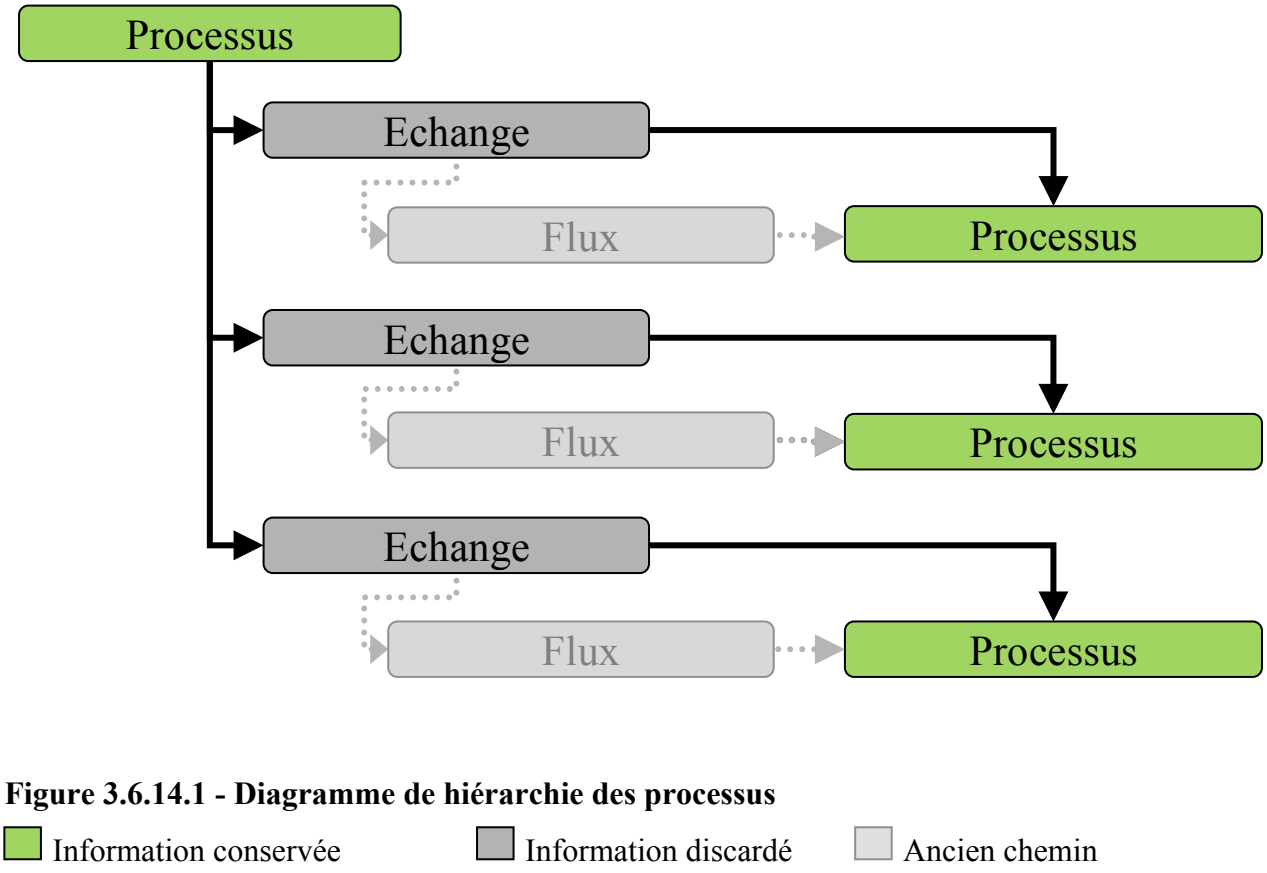

Figure 3.6.14.1 - Diagramme de hiérarchie des processus

En premier temps, le parcours de la hiérarchie des processus impliquai aussi les flux, mais une modification à la ressource REST utilisée pour les échanges a permis d'obtenir le processus producteur directement à partir d'un échange, ce qui a limité le nombre d'appels REST et la complexité du code de parcours des données.

Pour parcourir ces informations de nature hiérarchique, une méthode récursive à deux « modes », inspirée du patron visiteur, est utilisée. Pour un processus, la méthode récursive est appelée en mode « processus » avec ce processus en paramètre, et elle s'occupe d'entreposer les informations de ce processus dans le modèle de données. Ensuite, pour chaque échange appartenant au processus, la méthode s'appelle soit même en mode « échange » avec un échange en paramètre. Ce mode est responsable d'obtenir le processus associé à cet échange et de s'appeler en mode « processus » avec ce nouveau processus en paramètre.

Puisque la hiérarchie est cyclique, avec certains processus se contenant soit même, il faut limiter la profondeur parcourue pour cette première version du modèle de données. Ceci est accompli en implémentant un compteur de profondeur dans le mode processus qui arrête le fil d'exécution lorsque la profondeur désirée est atteinte.

Ayant implémenté ceci, il était évident que le modèle de données devrait être traité différemment, car ce chargement est très lent, même avec une profondeur de 4 processus. Ceci est accompli en utilisant une méthode de chargement paresseux dans jsTree.

Cette méthode est responsable de charger sur demande les processus enfants d'un processus en particulier et de les acheminer à jsTree dans un format préétabli via une méthode de rappel (*callback*). jsTree est responsable de déclencher l'événement pour récupérer les enfants d'un processus lors de l'interaction avec l'arbre. La procédure pour obtenir ces informations est très similaire à la procédure utilisée dans la méthode récursive, sauf qu'il n'y a pas de récursivité. Donc le code était majoritairement réutilisable.

Cependant, pour faciliter les manipulations d'échanges à venir, le code a été modifié pour que l'arbre contienne aussi l'échange associé au processus affiché. Les raisons pour cette modification seront décrites avec plus de détails dans la section 3.5.4 « Synchronisation de l'arbre ».

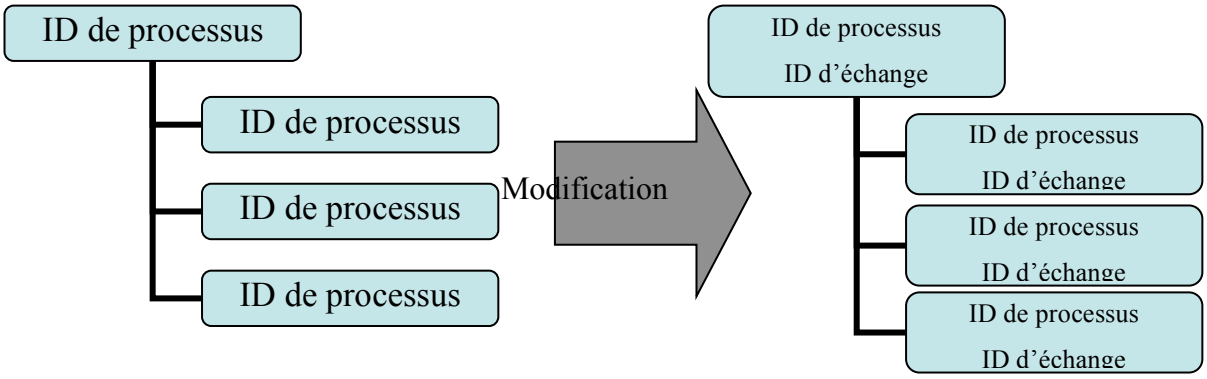

Figure 3.6.1.2 - Modification du type de données dans le modèle

Pour les fins du modèle de données, ces deux structures affichent toujours le même résultat final; un processus. Mais en conservant l'information de l'échange dans l'arbre, il est possible de retrouver quel est l'échange associé à ce processus, tout en étant aussi facilement capable de trouver le processus à afficher.

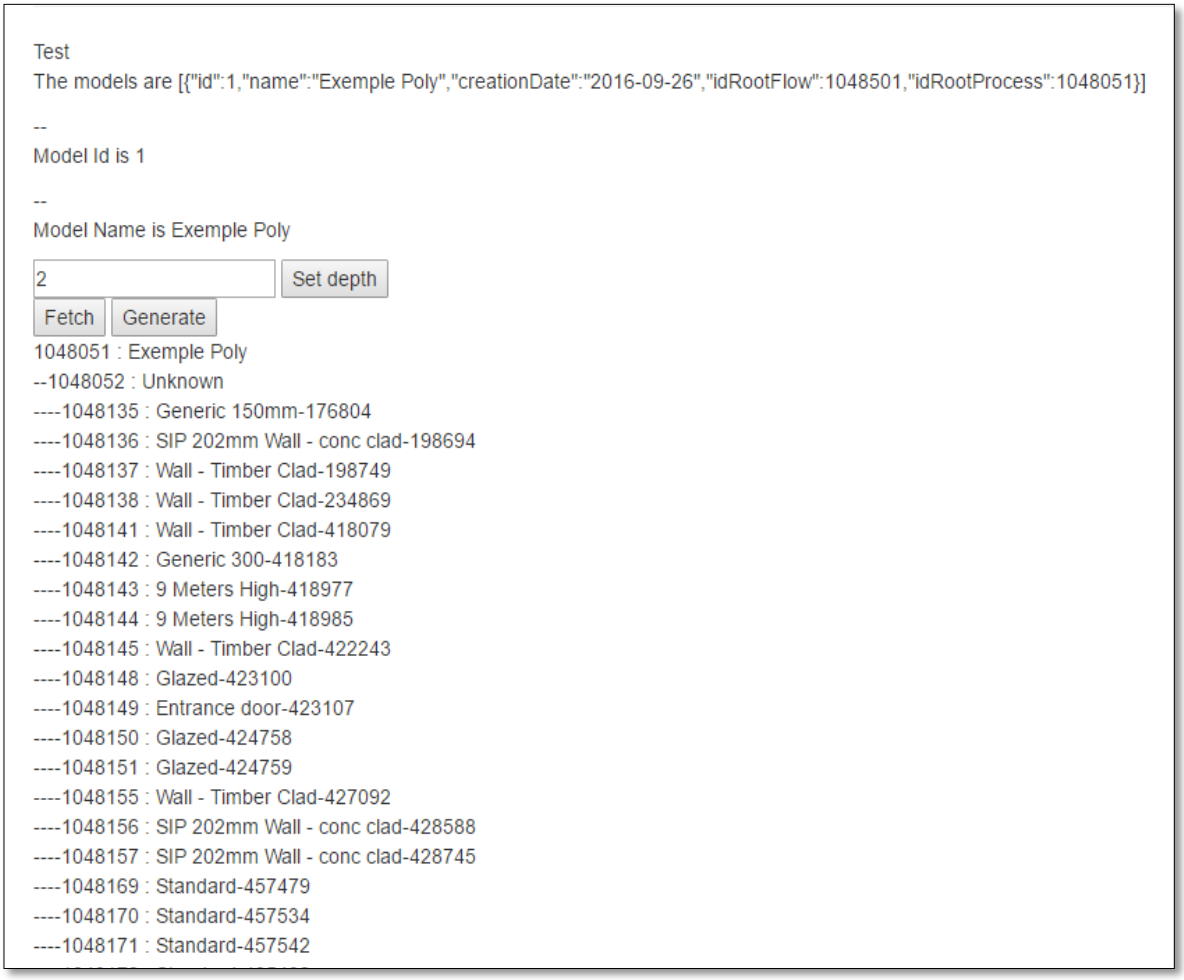

Figure 3.6.1.3 - Le modèle de données affiché en format brut (utilisant les ID de processus)

3.5 Conception de l'interface

Ayant conçu le modèle de données, ou plutôt ayant devisé une méthode pour construire le modèle de données sur demande via le chargement paresseux, il est temps de s'attaquer à la présentation de l'information.
En rencontrant le client, il a été conclu que l'interface serait divisée en deux parties. Sur le côté gauche, il aura l'arbre de processus, et dans un grand panneau à droite, il y aura les informations du processus que l'utilisateur aurait sélectionné dans l'arbre. C'est-à-dire :

- Les propriétés du processus (nom, description, etc.)
- La documentation du processus (date de création, technologie, géographie, etc.)
- Les échanges entrants et sortants du processus, dans deux tableaux
- Les paramètres associés au processus, dans un tableau

Considérant la quantité d'informations à afficher, une méthode pour diviser les groupes est requise. Deux méthodes sont possibles, un affichage en onglets, ou un affichage avec des sections déroulantes. En discutant avec le client, il a été décidé d'utiliser des onglets, par préférence, et aussi après la constatation que lorsque toutes les sections déroulantes seront déroulées, un scénario inévitable, il y aura beaucoup de contenu vertical à traverser, ce qui réduira la convivialité.

Pour l'affichage de l'information elle-même, des gabarits HTML sont créés pour être utilisés dans la création du contenu des onglets. Au lieu de créer des contrôles statiques dans le fichier HTML et de les remplir via des sélecteurs JavaScript, ces gabarits contiennent les contrôles pour chaque onglet, ainsi que des mot-clés utilisés pour insérer de l'information au bon endroit.

Un gabarit peut ressembler à ceci :

```
var MON_GABARIT = `
<div id="mongab_div">
   <input id="process_id" type="text" value="%processIdVal%"/>
   <input id="process_name" type="text" value="%processNameVal%"/>
   <input id="btnMonGabOk" type="button" value="OK"/>
   <input id="btnMonGabAnnuler" type="button" value="Annuler"/>
</div>`;
```

Le processus pour utiliser ces gabarits est simple :

- Définir une variable contenant une copie du gabarit :

  let newhtml = MON_GABARIT;

- Remplacer les mot-clés par l'information souhaitée :

  newhtml = newhtml

  .replace("%processIdVal%", process.id)

  .replace("%processNameVal%", process.name);

  Insérer le gabarit rempli dans la page HTML :

  $("body").html(newhtml);

Cette méthode a quelques avantages. Le principal avantage est que le code JavaScript est moins affecté lorsque le gabarit est modifié, il faut seulement prendre en compte les nouveaux mot-clés, il y a moins de couplage au niveau des sélecteurs. Deuxièmement, ces gabarits sont réutilisables, par exemple il y a un gabarit pour un tableau d'échanges et un gabarit pour une ligne dans un tableau d'échange, ces deux gabarits sont utilisés autant pour crée le tableau d'échanges entrants que le tableau d'échanges sortants. Aussi, il existe des gabarits utilitaires, par exemple pour créer une ligne d'option dans un contrôle de sélection, ou pour afficher une alerte.

Par contre, le gabarit pour l'affichage des informations d'un processus existe plutôt pour simplifier la modification du code HTML lors du développement, idéalement il serait remplacé par du code HTML statique.

Maintenant que l'arrangement de l'interface est décidé et que le code HTML est prêt a être rempli avec les donnés, l'affichage des informations d'un processus débute avec la sélection d'un nœud représentant un processus dans l'arbre.

Lorsque la librairie jsTree détecte un « *click* » sur un nœud, elle envoie un événement a l'interface avec quel nœud a été sélectionnée. Ce nœud contient les informations de l'échange associé. Avec cette information, il est possible d'obtenir le processus producteur, c'est-à-dire le processus sélectionné.

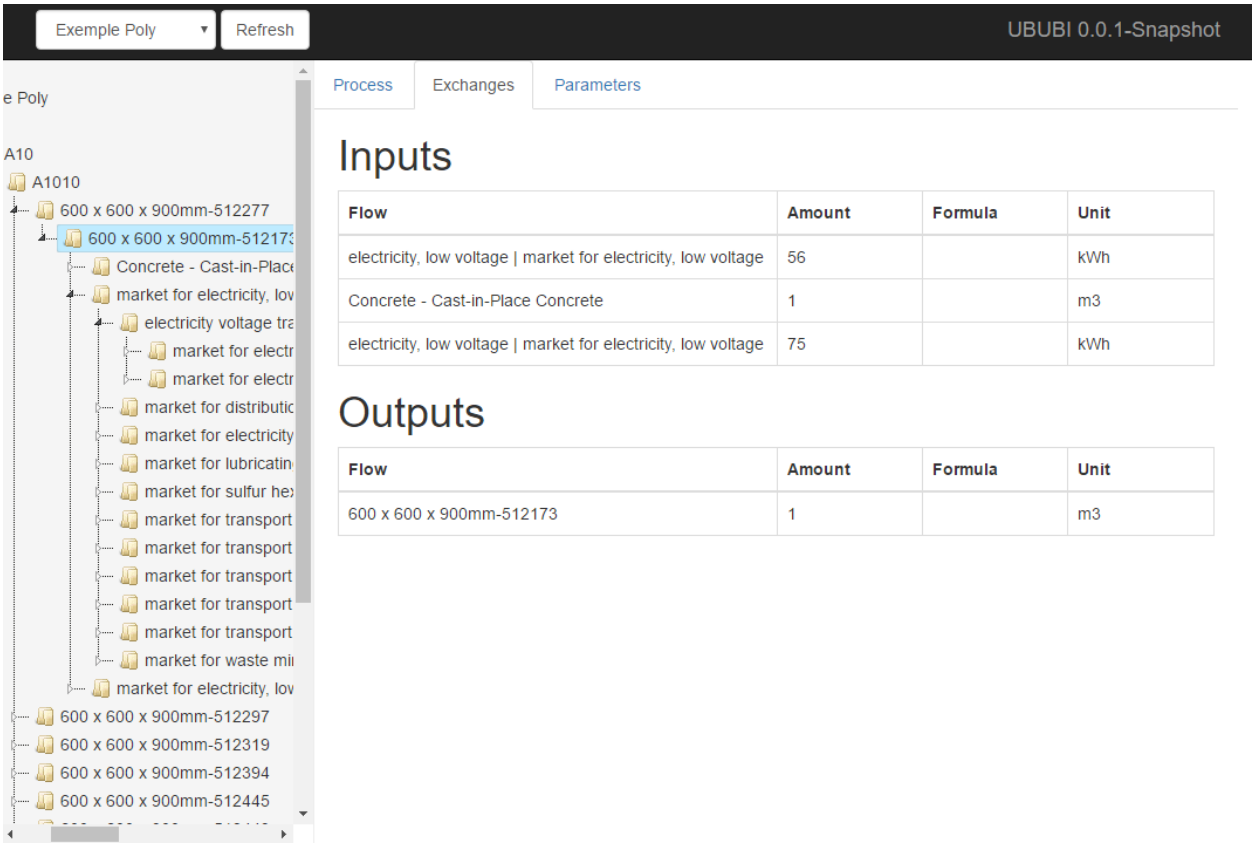

Figure 3.6.1.1 - L'affichage des échanges dans un onglet

3.6 Gestion de l'ajout, modification et de la suppression d'un échange

L'objectif principal de ce projet, autre que l'affichage des processus, qui est plus considéré comme un prérequis, est la manipulation des échanges. Ayant affiché l'arbre de processus et les informations d'un processus sélectionné, ce prérequis est atteint, et maintenant il est temps de passer à la seconde partie.

Notez que dans les 3 sections concernant l'ajout, modification et suppression, il est question de « mise à jour de l'arbre ». Ce sujet est abordé en détail dans la dernière section, 3.5.4, « Synchronisation de l'arbre ».

3.6.1 Ajout d'un échange

Avec un processus de sélectionné, l'utilisateur a la possibilité d'ajouter un échange a celui-ci. Pour faire ceci, l'interface doit demander à l'utilisateur quels sont les paramètres de ce nouvel échange, et à quel processus est-il lié.
Ces informations sont :

- Le processus lié a cet échange
- Est-ce que cet échange est sortant ou entrant
- Quelle unité utiliser pour la quantité utilisée/produite
- Quelle quantité utilisée/produite
- Et optionnellement : Le producteur par défaut, la formule, et est-ce que c'est un produit a évité. (Ces options sont mentionnées pour la complétude des informations possibles pour d'ajout d'échange, elles n'ont pas été traitées durant la conception de l'interface à cause du manque de temps)

Le défi principal pour cette étape est de déterminer comment de permettre a l'utilisateur de choisir quel processus lier a cet échange. Considérant la quantité de processus disponibles, un service REST de recherche de processus par mot-clé a été exposé. Ayant ce service, une

boite de recherche est jumelée avec un contrôle de sélection. L'utilisateur peut donc entrer un mot-clé d'au moins 4 caractères et obtenir une liste de processus correspondant au mot-clé.

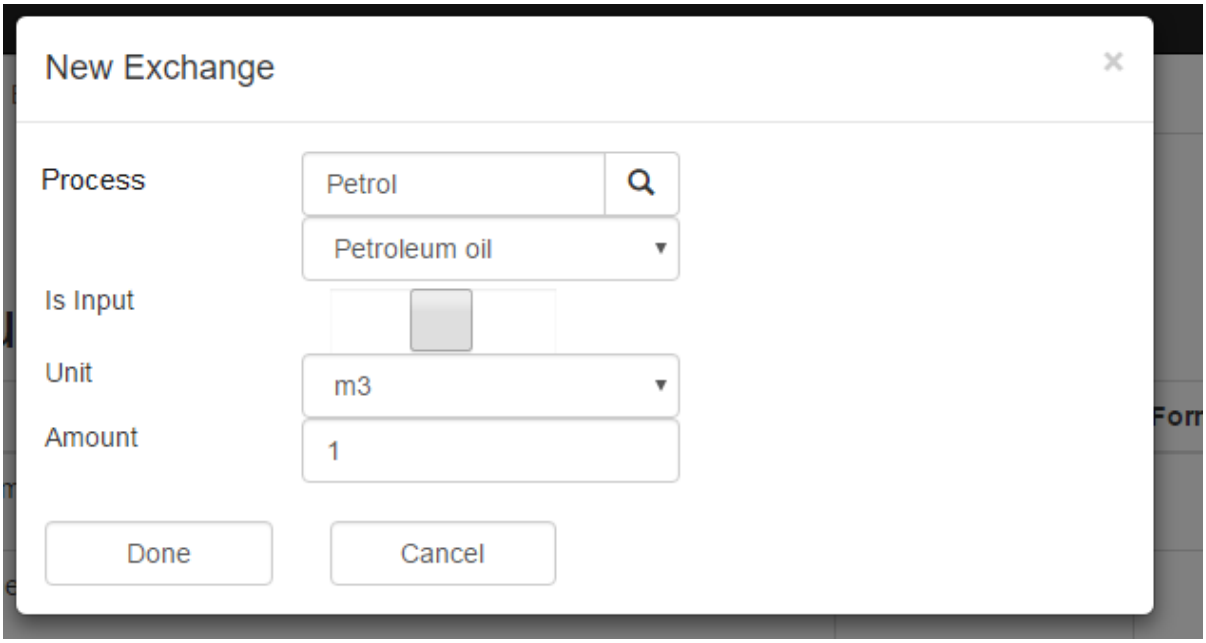

Figure 3.6.1.1 - Le formulaire pour créer un échange

La présentation du contrôle de sélection de processus n'est pas idéale. Idéalement ces deux contrôles auraient été combinés dans un défilé possédant une fonctionnalité de recherche, comme illustrer dans la figure 3.5.2 « Défilé avec recherche ». Mais puisque la source des données est un service REST, et que les données renvoyées par ce service ne sont pas simplement une liste de chaines de caractères (ce sont des objets flux), ce qui sont des requis pour la plupart des librairies offrant ce type de défilé, l'implémentation n'a pas pus été possible dans le temps alloué.

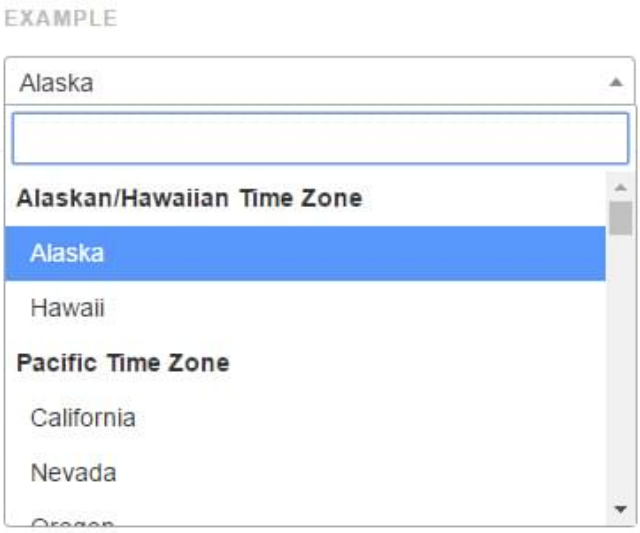

Figure 3.6.1.2 – Défilé avec recherche[2]

Après la collecte des informations nécessaires a l'ajout d'un échange, l'interface envoi une requête de type « PUT » à l'URL associée a la création d'un échange avec les informations collectées dans le formulaire, ainsi que le processus parent de ce nouvel échange, c'est-a-dire le processus sélectionné. Le serveur REST renvoi le code *200 OK* si l'ajout s'est accompli, ou renvoi un code d'erreur 404 *Not Authorized* s'il y a eu un problème.

Lorsque l'interface confirme que l'ajout s'est fait sans problème, elle rafraichit les onglets du processus pour afficher les échanges modifiés et met à jour l'arbre pour refléter les changements.

[2] Source: Cette capture d'écran a été tirée de l'article de Simon Codrington, *13 jQuery SelectBox/Drop-down Plugin*. Dans cet article, il compare plusieurs modules de défilés offrant des fonctionnalités de recherche.
Lien : https://www.sitepoint.com/13-jquery-selectboxdrop-down-plugins/

3.6.2 Modification du producteur par défaut d'un échange

Pour introduire cette fonctionnalité, il est important de décrire comment les échanges et les processus sont liés.
Un échange peut posséder un ou plusieurs processus qui le produisent, des producteurs (le processus producteur qui produit un échange est ce qui est affiché dans l'arbre). Lequel est utilisé dépend de la propriété *defaultProviderId* dans l'objet échange. Ceci correspond à l'ID du processus à utiliser. Cet ID peut aussi être « 0 », dans quel cas c'est le processus par défaut qui sera utilisé. Celui-ci est déterminé par le serveur UBUBI en utilisant des règles d'affaires.
Concernant le processus producteur, l'interface doit offrir le choix à l'utilisateur de modifier lequel est utilisé.
Pour accomplir ceci, un contrôle de sélection est affiché à côté de chaque échange dans l'onglet d'échanges pour un processus sélectionné (voir Figure 3.5.3). Ce contrôle contient l'option « <Default> », qui représente la valeur « 0 » pour *defaultProviderId*. Il contient ensuite tous les processus apparaissant dans la liste de producteurs pour cet échange. Lors du chargement, la valeur sélectionnée correspond a ce que *defaultProviderId* contient; soit « <Default> » ou le processus ayant l'ID correspondant a l'ID contenu dans *defaultProviderId*.

Figure 3.6.2.1- Structure des échanges et des producteurs

Avec ce contrôle de sélection, l'utilisateur peut ainsi changer le producteur par défaut d'un échange.

Dans le logiciel, cette modification se fait simplement en envoyant une requête « PUT » avec un objet JSON contenant la propriété *defaultProviderId* avec la valeur choisie a l'adresse obtenue via le routeur JavaScript pour l'échange en question. Le serveur retourne *200 OK* si la modification est accomplie, ou *404 Not Authorized* s'il y a eu un problème.

Lorsque la modification a été acceptée par le serveur, l'interface met à jour l'arbre en modifiant toutes les occurrences de cet échange pour que le processus affiché soit le nouveau processus producteur de cet échange.

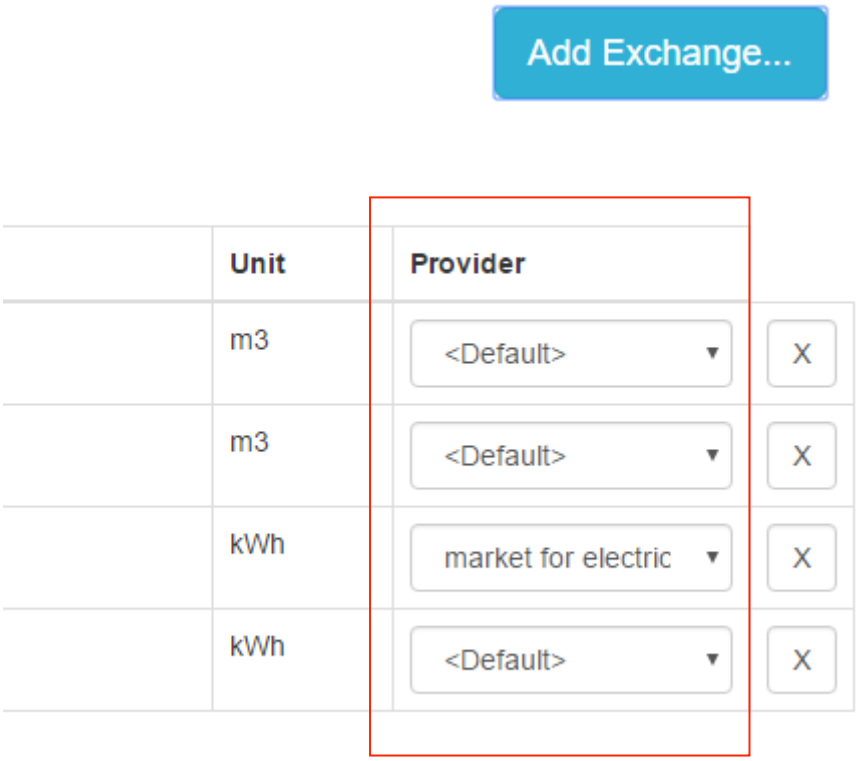

Figure 3.6.2.2 - Modification du producteur par défaut

3.6.3 Suppression d'un échange

Très similaire à l'ajout d'un échange, la suppression d'un échange se fait dans l'onglet des échanges d'un processus, en appuyant sur le bouton « X » à droite de l'échange désiré.
Lorsque l'utilisateur confirme son intention, l'interface envoi *DELETE* a l'URL obtenue via le routeur JavaScript pour une requête de suppression d'un échange. Le serveur répond avec 200 OK ou 404 Not Authorized selon le résultat. Lorsque l'opération est complétée, l'interface met à jour l'arbre en supprimant toutes les occurrences de l'échange supprimé.

3.6.4 Synchronisation de l'arbre

Tout au long de ces 3 dernières sections, il a été question de la mise à jour de l'arbre.
Ce sujet a aussi été brièvement abordé dans la section 103.4 Conception du modèle de données.
Lorsqu'une modification dans la hiérarchie de processus et d'échanges, il faut la refléter dans l'arbre. Cependant, puisque cet arbre est profond, il n'est pas souhaitable de tout recharger la hiérarchie, car ceci aurait comme effet de fermer l'arbre jusqu'à la racine. Il y a deux options.
La première est de conserver l'état de l'arbre actuel et, après que l'arbre se soit refermé, restaurer incrémentalement l'état de l'arbre. Ceci est une option complexe à implémenter, car l'utilisateur peut avoir plusieurs branches ouvertes dans l'arbre, pas nécessairement une seule hiérarchie. Aussi, cette option est lente à exécuter. Il faudrait effectuer de requêtes REST pour chaque nœud, jusqu'à que l'état soit restauré.
La deuxième est de répliquer les modifications dans l'arbre de façon conservative, juste modifier ce qu'il faut. Cette option permet de ne pas perdre l'état de l'arbre, mais tous les nœuds affectés par le changement de l'échange doivent être modifiés. La deuxième option est choisie, et c'est ici que la modification décrite à la fin de la section 3.4 Conception du modèle de données rentre en jeu. En rappel, la modification fait en sorte que chaque nœud de l'arbre contient l'identificateur de l'échange en plus de l'identificateur du processus.

Pour l'ajout, l'opération est simple; trouver toutes les instances du processus contenant le nouvel échange et ajouter un nouveau nœud enfant à celui-ci correspondant au nouveau processus dans un état fermé. Puisque l'arbre contient l'identificateur de processus, il n'y a pas de requêtes REST à faire, seulement une comparaison avec le processus sélectionné et tous les processus dépliés.

Pour la suppression l'opération est élémentaire; supprimer toutes les occurrences du nœud qui proviennent de l'échange supprimé.

Pour la modification du producteur par défaut, la manipulation est un peu plus compliquée. Il faut retrouver tous les nœuds ayant l'échange modifié, fermer ce nœud s'il est déplié, et modifier l'identifiant et le nom de processus associé a ce nœud. Faisant ceci, un nœud est changé pour refléter la modification du producteur et est prêt à être déplié pour afficher les nouveaux processus appartenant au producteur différent.

Une méthode est aussi mise en place pour limiter le nombre d'occurrences d'un même processus; si l'utilisateur déplie un processus, et que ce processus contient un processus déjà déplié dans la hiérarchie actuelle, ce processus qui est déjà déplié ne sera pas dépliable ici.

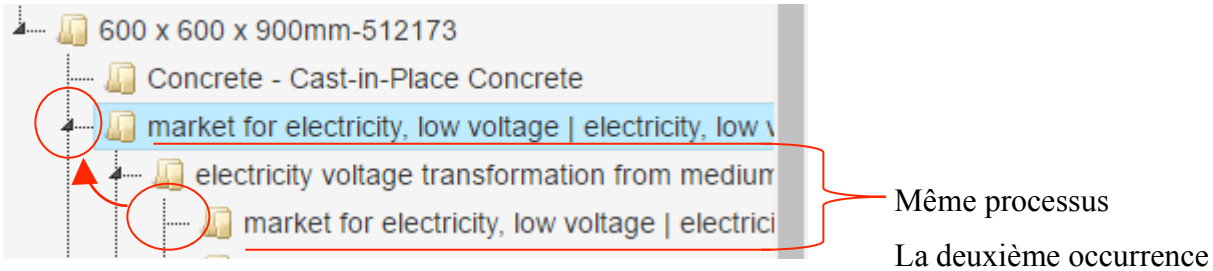

Figure 3.6.4.1 - Limitation du dépliement redondant

CONCLUSION

Le projet UBUBI avance, ce projet d'interface utilisateur est un autre pas vers l'avant, parmi plusieurs à venir.
La méthodologie agile utilisée lors du projet a permis de garder un rythme tout au long de la session, et les comptes-rendus on permis de corriger rapidement les déraillements et résoudre les problèmes a fur et à mesure qu'ils ont apparus.
En regardant les objectifs établis au début du projet, ceux-ci ont comporté des défis attendus, mais d'autres ont comporté des défis inattendus. La structure des donnés étaient relativement complexe a apprendre, les liens entre les processus, les flux, les échanges, et toutes les autres propriétés que possèdent ceux-ci on mener a confusion a plusieurs reprises, mais surprenamment, la quantité d'information a géré n'était pas un défi. Somme toute, l'affichage des informations et la modification des échanges sont accomplis. L'utilisation du chargement paresseux est une solution qui permet d'offrir une interface fonctionnelle et complète sans temps d'attente ou de limitations. Et la solution trouvée pour synchroniser l'arbre lors des changements semble être fonctionnelle, mais le manque de temps ne laisse pas de place pour complètement tester ces fonctionnalités.

Au sujet des développements futurs, l'interface profiterait d'une période pour améliorer l'apparence visuelle, car ceci est un aspect important pour toute interface utilisateur. Il reste aussi a implémenter les icônes de types de processus dans l'arbre (voir annexe II, point 5.9), un tache qui été mise en arrière plan pour concentrer les efforts sur l'ajout, modification et suppression des échanges.
Mais tout ceci est mineur comparé à la prochaine grande étape pour ce projet qui est l'affichage visuel avec d3js des résultats de l'empreinte écologique des processus. Ceci sera un grand pas vers un produit final.

LISTE DE RÉFÉRENCES

[1] Mathieu Dupuis, UBUBI wp1 - Software requirement software (SRS) , v0.1 2016

[2] Mathieu Dupuis, UBUBI wp1 – Document de vision, v1.0.1, 2015

[3] House, C. (3 janvier 2016). *Angular2 versus React : There Will Be Blood*. Source: https://medium.freecodecamp.com/angular-2-versus-react-there-will-be-blood-66595faafd51

BIBLIOGRAPHIE

AngularJS	https://angularjs.org/
React	https://facebook.github.io/react/
jsTree	https://www.jstree.com/
d3js	https://d3js.org/

ANNEXE I

COMPILATION DES COMPTES-RENDUS SCRUM

Cette annexe se trouve dans le fichier joint nommé
LOG792_CompilationDesComptesRendus_PIGEON_Philippe.docx

Ce document permet de voir la progression du projet avec des captures d'écrans et permet aussi de voir les accomplissements et les défis rencontrés chaque semaine.

ANNEXE II

RÉCAPITULATION DU PLAN DE TRAVAIL

ıce	Termine	Effort estimé*	Effort actuel**	Tâches/Jalon	Livrable(s)/Artéfacts	Re
	16-09	-		Rencontre – professeur superviseur		
	16-09	1	1	Remise de la fiche de renseignements	Fiche de renseignements	
	26-09	4	4	Remise de la proposition de projet	Proposition de projet	
	20-09	3	3	Analyse – Intégration		
	08-10	3	10	Analyse – Outils technologiques		
	15-10	8	8	Conception – Configuration de l'environnement		
	14-10	8	8	Conception – Familiarisation		
	21-10	16	8	Conception – Prototype avec REST	Prototype 1	
	28-10	4	4	Conception – Implémentation avec PlayFramework	Rapport d'étape	
	28-10	12	12	Conception – Arbre de processus	Prototype 2	
	04-11	8	12	Conception – Détails d'un processus	Prototype 3	
	11-11	4	8	Conception – Raffinement de l'interface		
	26-11	4	8	Remise du rapport d'étape	Rapport d'étape	
	-	4	-	Conception – Affichage des types dans l'arbre		
	16-12	24	30	Conception – Modification des processus	Prototype 6	
	01-12	-	-	Démonstration par Mathieu Dupuis		
	02-12	16	3	Modifications suite aux commentaires	Prototype final	
		16	10	Présentation	Présentation	
		8	8	Élaboration du rapport final		
		-	-	Remise du travail	Rapport	