Université du Québec
École de technologie supérieure
Département de génie logiciel

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG795-GTI795 PROJET DE FIN D'ÉTUDES EN LOG-TI

CONCEPTION D'UNE APPLICATION WEB POUR LE CALCUL ENVIRONNEMENTAL D'UN BÂTIMENT
ÉQUIPE 008 - UBUBI CALCUL

Cydrick Trudel (TRUC03029301)
Daniel Enachescu (ENAD25079107)
Jean-Pierre Bertrand-Dorion (BERJ03119209)
Aravinthan Sivaneswaran (SIVA19029209)

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Alain April

MONTRÉAL, 19 AOÛT 2017
ÉTÉ 2017

REMERCIEMENTS

Cydrick Trudel, Daniel Enachescu, Jean-Pierre Bertrand-Dorion et Aravinthan Sivaneswaran tiennent à remercier le professeur Alain April pour ses conseils tout au long du PFE ainsi que ses connaissances de la réalité des projets étudiants en industrie. En effet, le professeur April connaît bien ce que les étudiants vivent au quotidien ainsi que ce qu'ils les attendent sur le marché du travail.

Cydrick Trudel, Daniel Enachescu, Jean-Pierre Bertrand-Dorion et Aravinthan Sivaneswaran tiennent aussi à remercier monsieur Mathieu Dupuis. Ses talents en gestion de projet nous ont grandement aidés durant ce PFE. Son entregent fût apprécié par l'ensemble de l'équipe.

CONCEPTION D'UNE APPLICATION WEB POUR LE CALCUL ENVIRONNEMENTAL D'UN BÂTIMENT
ÉQUIPE 008 - UBUBI CALCUL

RÉSUMÉ

Ce rapport est le dernier livrable de l'équipe UBUBI calcul assemblé dans le cadre d'un projet de fin d'études (PFE) de l'ÉTS durant la session d'été de l'année 2017. Il présente, entre autres, l'avancement effectué par chacun des membres, semaine par semaine, et de leur contribution individuelle dans ce projet. Le projet UBUBI est une application permettant de faire une analyse de cycle de vie des nouvelles constructions. Développé par les étudiants de l'ÉTS et en partenariat avec des chercheurs du domaine de la construction à l'École Polytechnique de Montréal. Encadrés par le professeur Alain April et l'étudiant au doctorat Mathieu Dupuis, plusieurs étudiants ont contribués à la programmation d'un prototype avant d'entamer les travaux présentés dans ce rapport. De plus, plusieurs autres étudiants y contribueront dans les sessions à venir.

L'équipe actuelle de PFE actuelle est constituée de quatre ingénieurs logiciel, aux profils différents, cherchant à faire avancer le projet. Le projet est majoritairement un projet de conception et de programmation et il est intimement relié aux principes de développement durable. Ce PFE est d'une durée d'environ quatre mois où les étudiants doivent réaliser plusieurs tâches telles que des ajouts de fonctionnalités, de manière itérative, ainsi que des mises à jour au code existant du prototype logiciel.

Ce rapport présente, entre autres, une introduction au projet, la contribution détaillée par chaque membre de l'équipe à chaque semaine de travail ainsi que plusieurs résultats de travaux qui ont été effectués durant la session. Il se termine par une conclusion qui contient les recommandations de l'équipe.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BIM	Building Information Modeling
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
HTML	Hypertext Markup Language
IP	Internet Protocol (réfère plus souvent aux adresses plutôt qu'au protocole)
JS	JavaScript
JSON	JavaScript Object Notation
LCI	Life Cycle Inventory
LCIA	Life Cycle Inventory Assessment
ORM	Object/Relational Mapping
PFE	Projet de Fin d'Étude
REST	Representational State Transfer
SLCIA	Sensitivity LCIA
SQL	Structured Query Language
UBUBI	Uncertain But Useful Building Information
ULCIA	Uncertainty LCIA
URL	Uniform Resource Locator
VPN	Virtual Private Network
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations

LISTE DES DÉFINITIONS

Action	Méthode publique d'une API REST qui peut être appelé par un navigateur web. Une action reçoit des paramètres et retourne des valeurs. Permet la communication entre un navigateur web et un serveur web.
Gabarit	Contenue de fichier nécessaire pour qu'un logiciel puisse interpréter le fichier comme étant valide, mais ne contenant aucune donnée.
Mock	Objet simulant un autre vrai objet et permettant le contrôle complet sur les entrées et les sorties.

INTRODUCTION

L'analyse de cycle de vie consiste à calculer les impacts sur l'environnement d'un produit, de sa création jusqu'à sa fin de vie utile. Par exemple, le choix des matériaux de base, la distance qu'ils ont à voyager pour arriver à leur emplacement final, le moyen de transport utilisé, la source d'énergie utilisée pour générer l'électricité nécessaire à la fabrication du produit et même la méthode de fin de vie du produit est considéré dans une analyse de cycle de vie. Il est possible de changer certains paramètres de l'analyse de cycle de vie (par exemple, choisir un matériel moins polluant) et comparer les impacts sur l'air, le sol et la terre entre les deux options. Cependant, la quantité de données à fournir pour produire une analyse de cycle de vie est très grande, et les résultats sont difficiles à interpréter.

UBUBI (Uncertain But Useful Building Information) est un logiciel en développement qui a pour but d'effectuer une analyse de cycle de vie sur les bâtiments. L'avantage de ce logiciel par rapport aux autres présentement publiées est qu'il a pour but de permettre cette analyse pendant la phase de conception de la maquette 3D du bâtiment. Il permet de téléverser des données provenant d'un modèle de type BIM (Building Information Modeling) et d'obtenir rapidement une analyse environnementale. Les multiples résultats d'une analyse de cycle de vie représentent beaucoup de données. Les données sont donc présentées dans un format qui permet une interprétation facile par du personnel oeuvrant dans le domaine de la construction. Ce projet est déjà en cours et est supervisé principalement par le promoteur de projet, Mathieu Dupuis. Le projet a débuté il y a quelques mois, et est encore en phase de développement. Il est planifié de rendre le code source du projet ouvert (Open Souce).

L'équipe de PFE, pour le projet UBUBI calcul, est composée de: Cydrick Trudel, Daniel Enachescu, Jean-Pierre Bertrand-Dorion et Aravinthan Sivaneswaran. Le mandat de cette équipe fut de travailler sur la sauvegarde des résultats d'analyse environnementale et leurs présentations sous divers formats. Plus précisément, l'équipe a travaillé sur l'enregistrement des résultats retournés par un service de calcul distant dans une base de données locale. De plus, l'équipe à travaillé sur l'affichage des données sous forme de graphiques et la génération d'un fichier Excel contenant les données. La figure 0.1.1 présente un diagramme décrivant l'interaction entre les différentes entités du projet. Le rectangle rouge montre la portée du présent projet, ce qui exclut la génération et le téléversement du modèle de type BIM ainsi que le calcul des résultats d'une analyse de cycle de vie à partir du modèle de type BIM (ce travail étant fait par le service de calcul).

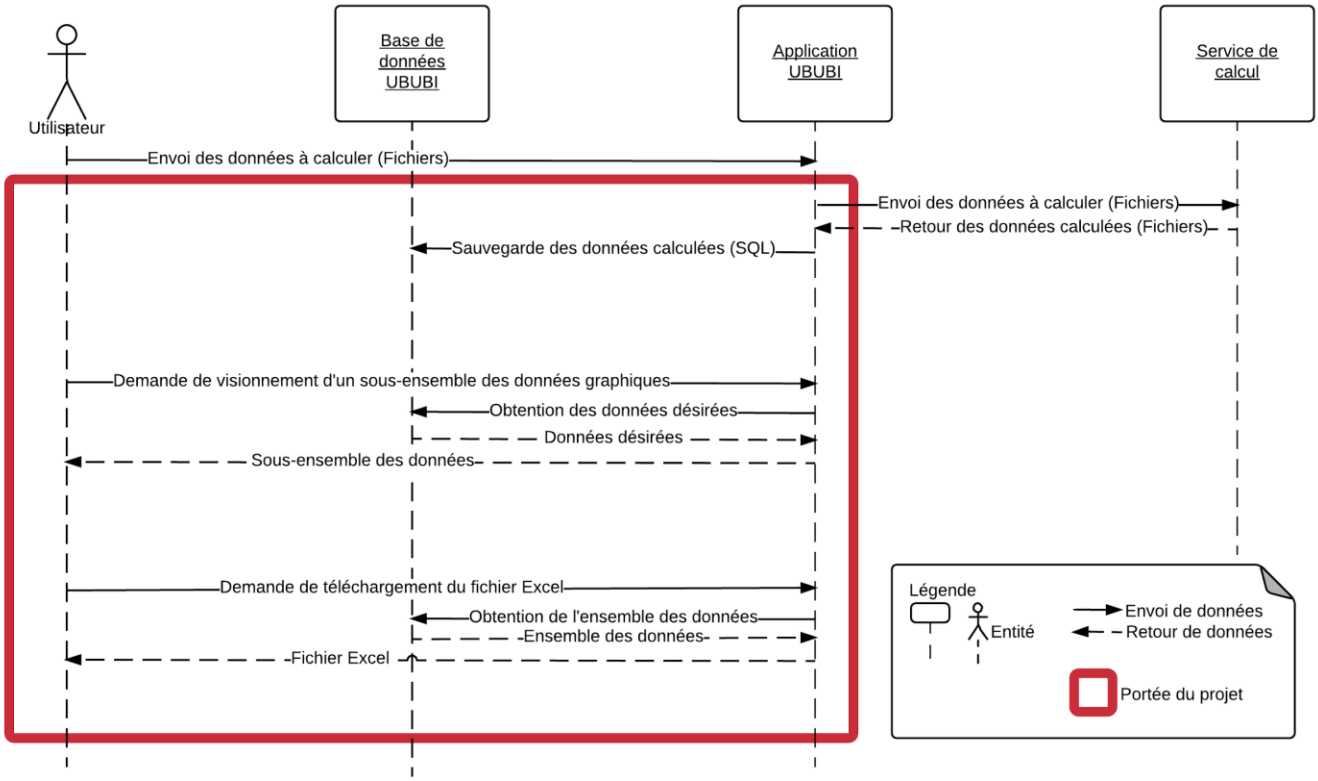


Figure 0.0.1: Diagramme de la communication entre entités et de la portée du projet

L'équipe a aussi participé à l'ajout de certaines fonctionnalités, dont l'édition des formules de variables globales et locales ainsi qu'à la maintenance de certaines parties du code source du prototype existant. Ce rapport a pour objectif de présenter les contributions des membres de l'équipe UBUBI calcul effectuées à chaque semaine du PFE, le fonctionnement des parties conçues et développées ainsi que les décisions qui ont dû être prises pour accomplir les tâches demandées. Ce rapport est structuré sous forme de chapitres représentant chacun une semaine de travail.

CHAPITRE 1: 9 mai au 16 mai

Dans cette première semaine, l'équipe a eu sa première rencontre avec le promoteur du projet. Le promoteur a expliqué le projet et le domaine d'affaire. Il a détaillé les fonctionnalités qu'il voulait que l'application UBUBI possède. Il a utilisé plusieurs exemples pour expliquer qu'est-ce qu'une analyse de cycle de vie. Il a expliqué comment les experts divisaient les intrants et extrants d'une manufacture de production de produit analysé.

À la suite de l'explication du projet et du fonctionnement du prototype existant. L'équipe s'est connecté à plusieurs technologies, telles que Trello, GitLab et d'autres afin de se familiariser aux ressources et techniques nécessaires pour commencer à travailler dans le projet de façon structurée.

Le promoteur nous a aidés aussi à configurer le projet dans nos ordinateurs portables puisque les configurations étaient faites de façon spécifique. Il a également distribué des machines virtuelles qui contenaient des données dans une base de données.

Une fois que tous les membres de l'équipe étaient configurés avec le projet, le promoteur a donné notre première tâche à effectuer. Celle-ci consistait à créer un graphique à l'aide de la librairie AmCharts, dont les données à afficher étaient obtenues à l'aide d'un appel à l'application UBUBI (bien que ces données d'essais étaient complètement aléatoires pour le moment). Cette première tâche était principalement définie pour se familiariser avec le code source du prototype actuel et de s'assurer que l'ensemble des outils de développement étaient bien configurés.

Daniel

Pendant cette première semaine, Daniel a pris le temps de comprendre l'architecture du projet afin de se familiariser avec le fonctionnement des classes. Une fois qu'il a eu une bonne idée du projet, il a effectué des essais de fonctionnement du logiciel AmCharts. Il a suivi un petit tutoriel[1] en ligne pour comprendre la façon que fonctionnent les graphiques.

À la fin du tutoriel, il a commencé par implémenter le code d'un service web pour retourner des données fictives pour le graphique. En implémentant tout le nécessaire pour faire fonctionner ce code, il l'a testé en suivant la structure des URL du projet. En s'assurant que le retour des données fonctionne correctement, il a implémenté le code dans l'interface pour faire l'appel au contrôleur. Un simple appel utilisant les liens dans le fichier « *routes* » lui a permis d'accéder aux données fictives. Concernant l'interface, Daniel a eu des difficultés à faire afficher le graphique qui s'est résolu lorsqu'il a spécifié une grandeur pour le graphique. La figure 1.1.1 présente le résultat final de l'expérimentation de Daniel.

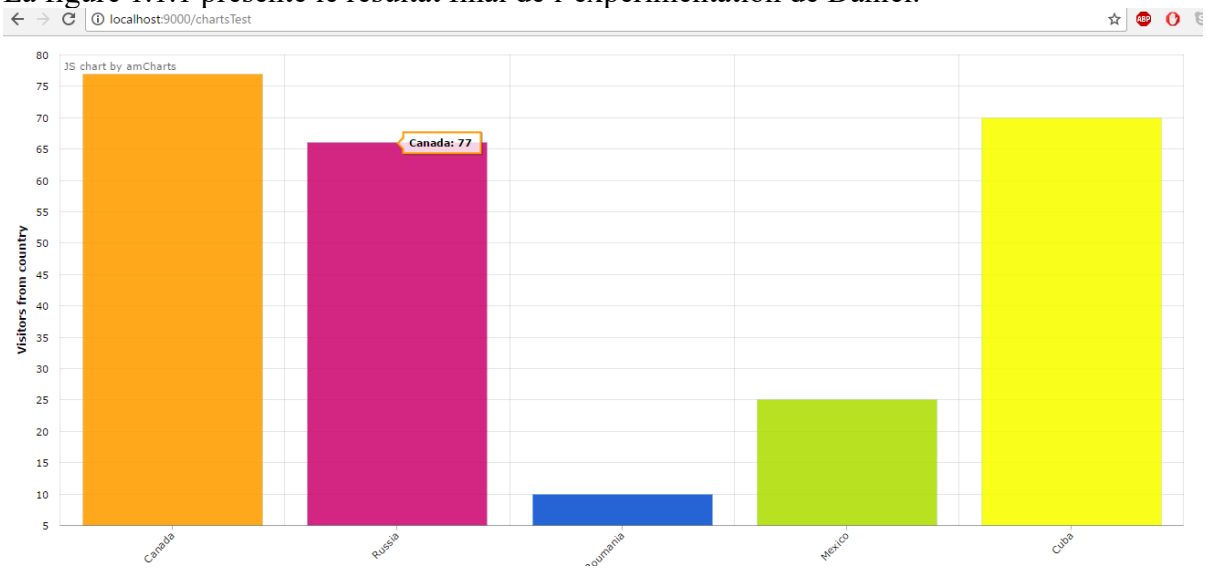


Figure 1.1.1: Test de diagramme AmCharts de Daniel

[1] AmCharts. https://www.amcharts.com/demos/simple-column-chart/

Cydrick

Pour Cydrick, il lui fut nécessaire de bien comprendre le fonctionnement de Play Framework. Le travail d'implémentation pour l'ajout d'une méthode pour retourner les données n'étant pas triviale, il lui fut nécessaire de bien comprendre comment l'information était retournée. La fonctionnalité déjà établi d'obtention des objets de type « BIMModels » auprès de l'application a permis de mieux comprendre comment l'information était acheminée et retournée vers l'interface utilisateur.

Après avoir ajouté le code nécessaire dans les classes « *JSONFormatter.java* » , « *Spark.java* », « *Router.java* » et « *routes* », il fut nécessaire pour Cydrick de consulter un tutoriel sur le site web officiel de AmCharts[2] concernant les graphiques à bandes simples pour comprendre le fonctionnement de AmCharts. L'implémentation de ce graphique contenant les données retournées par l'application fut relativement simple à développer. La figure 1.2.1 représente les résultats obtenus par Cydrick:

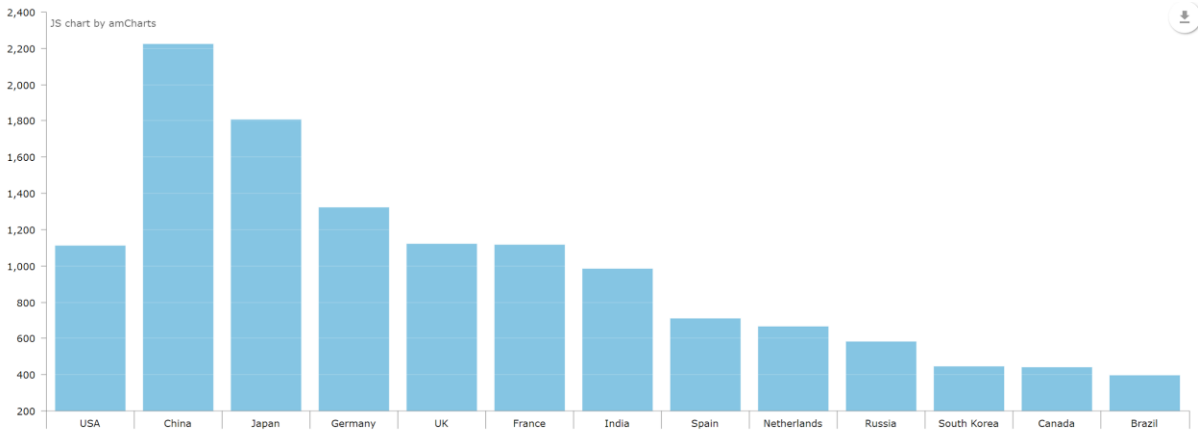


Figure 1.2.1: Test de diagramme à bandes de Cydrick

Cette même semaine, Cydrick a aussi fait quelques recherches concernant le meilleur moyen d'afficher des données bidimensionnelles avec profondeur en plus de la valeur, ce qui n'est pas possible dans un diagramme à bandes régulier. En effet, un des graphiques utiles à présenter aux utilisateurs est la décomposition des effets de certaines parties du bâtiment sur l'environnement. Il serait aussi possible de cliquer sur un sous-ensemble d'un bâtiment pour voir leurs impacts sur l'environnement. La figure 1.2.2 montre un diagramme qui présente de telles informations:

[2] AmCharts. https://www.amcharts.com/demos/simple-column-chart/

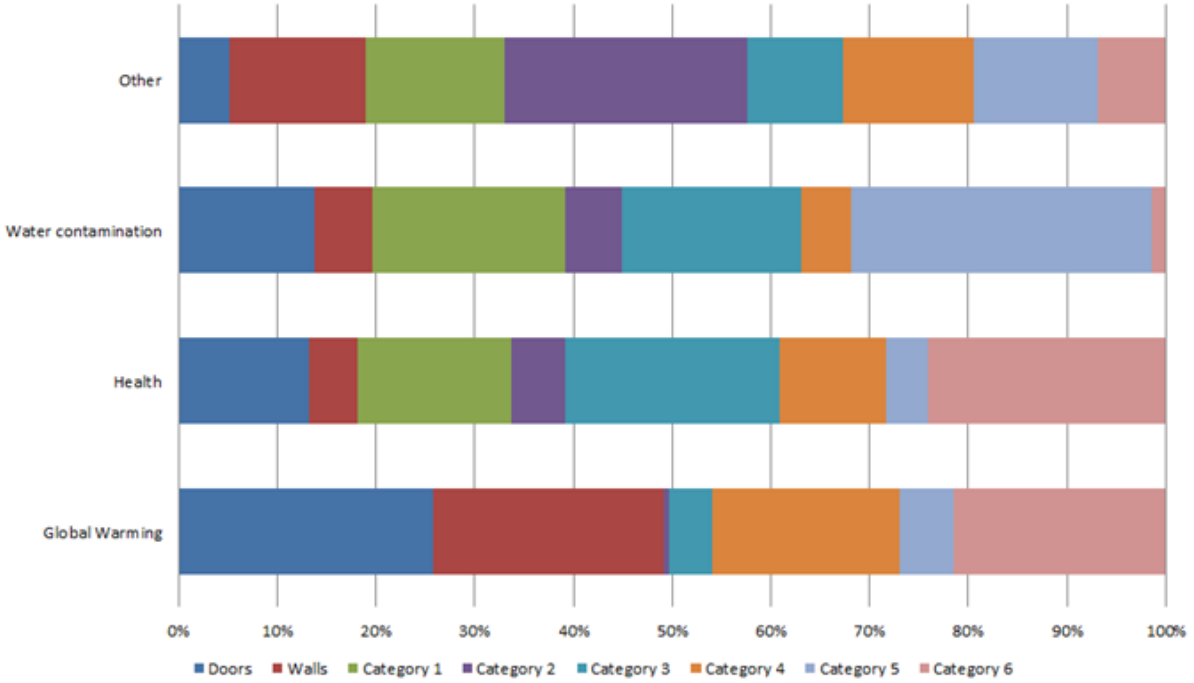


Figure 1.2.2: Diagramme à bandes empilées de référence

Le diagramme de la figure 1.2.2 permet la comparaison rapide des valeurs de chacune des parties d'un bâtiment entre elles. Il est toutefois difficile d'effectuer une comparaison entre les parties d'un bâtiment puisqu'elles n'ont pas le même point d'origine sur le diagramme. Il est toutefois important de noter que même s'il est impossible d'obtenir la valeur en pourcentage de chacune des catégories, ceci ne représente pas un problème puisqu'il serait facile de mettre cette valeur sur la catégorie directement à même le graphique.

Cydrick a ainsi observé qu'il serait possible d'utiliser un diagramme de type « Sankey »[3] pour représenter ces valeurs. Ce diagramme peut facilement montrer deux dimensions ainsi que de montrer la valeur en plaçant la souris sur la bande. La figure 1.2.3 démontre de quoi pourrait avoir l'aire d'un diagramme de type Sankey adapté au problème:

[3] Wikipedia. https://fr.wikipedia.org/wiki/Diagramme_de_Sankey

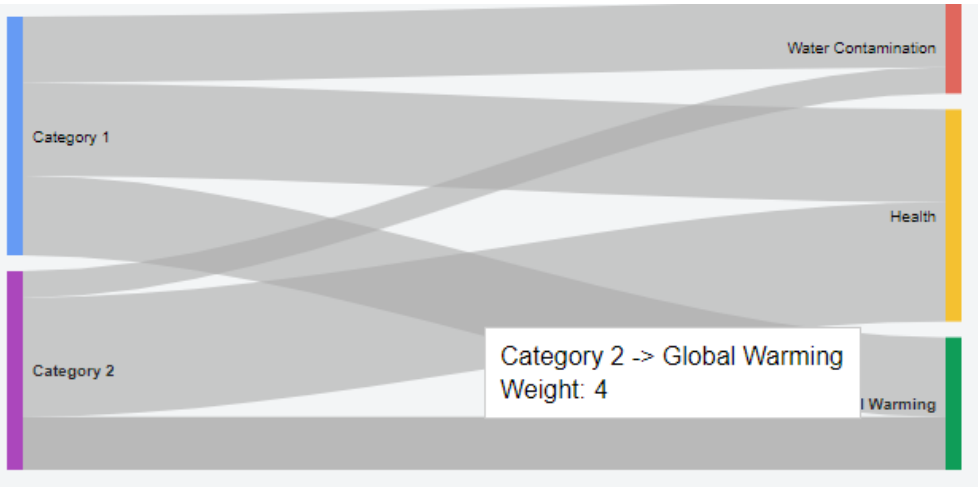


Figure 1.2.3: Exemple de diagramme de type Sankey

Toutefois, sans demande précise de la part du promoteur de projet concernant l'affichage sous forme de diagramme Sankey, le résultat n'a pas été utilisé durant le PFE.

Jean-Pierre

Pour Jean-Pierre, le projet démarre difficilement. Plusieurs erreurs surviennent à l'importation du projet dans Intellij-Idea : L'IDE ne détecte pas la structure du projet, le JRE n'est pas défini, le Play Framework et le *plugin* Scala refusent de s'installer et le projet refuse de démarrer, car l'application n'arrive pas à communiquer avec la base de données.

En premier lieu, l'IDE Intellij-Idea n'arrive pas à afficher le projet après l'importation pour des raisons inconnues. L'IDE est alors réinstallé, puis le projet est importé encore, mais sans succès. Une recherche sur internet aura permis de trouver la solution au problème; il s'agissait de recréer un *module* dans Intellij-Idea du même nom pour que le projet soit réinitialisé dans l'IDE. Cette étape a permis à Intellij-Idea de détecter la structure du projet.

L'erreur du JDK est commune, surtout sur Linux. Sous Windows, le JDK est installé en même temps que l'IDE, car les dépendances ne sont pas partagées de facto comme sur Linux. Pour remédier à cette erreur, il faut installer le JDK via le gestionnaire de paquet de la distribution installée. Par la suite dans Intellij-Idea il ne suffit que d'aller dans "File > Other Settings > Default Project Structure…" et d'aller dans l'onglet "*Project*" pour y sélectionner le SDK utilisé pour le projet.

Par la suite, l'installation de plugin ne fonctionne pas. Lors de l'installation, l'IDE tombe en erreur pour des raisons inconnues. Une recherche approfondie sur internet dévoile que l'erreur est due à la présence d'un dossier sur l'ordinateur d'une ancienne installation d'Intellij-IDEA. Pour fixer ce problème, il ne suffit que d'enlever le dossier et l'installation de plugins fonctionnera tel que prévu.

La dernière erreur est située dans la connexion entre l'application et la base de données. L'application ne démarre pas, indiquant qu'elle n'est pas capable de se connecter au serveur MySQL. Cette erreur est dûe à l'utilisation de « virt-manager » plutôt que « VirtualBox ». La machine fonctionne, mais pour une certaine raison inconnue, le serveur DHCP sur sa machine n'attribue pas l'IP attendu par l'application contrairement à VirtualBox (la machine virtuelle s'attribue l'IP 191.168.1.6 plutôt que 192.168.1.3, tel que codé dans l'application). C'est probablement dû à la présence d'autres machines virtuelles sur l'ordinateur. Pour remédier à la situation, il ne suffit que de changer la configuration IP du serveur MySQL en configuration statique plutôt que DHCP. De cette façon, l'erreur ne peut plus se reproduire si l'IP codé dans l'application est le même que l'IP statique.

À la fin de la semaine, Jean-Pierre à allouer du temps pour documenter tous ces problèmes afin que les futurs étudiants travaillant sur ce projet n'est pas à faire tous ces recherches.

Aravinthan

Aravinthan a profité pour faire l'installation des composants pour pouvoir exécuter l'application tel que l'IDE et la machine virtuelle qui contient la base de données. Par la suite, il a analysé l'architecture du logiciel. Puisqu'il n'a jamais travaillé avec le Play framework, il a profité pour apprendre un peu plus en lisant des articles et les documentations en ligne. Une fois qu'il à l'aise avec le cadriciel, il a commencé à programmer une page de test. Il a créé l'URL dans le fichier routes et par la suite la page qui permettait de visualiser la page.

Une fois qu'il a fait apparaître la page de test, Aravinthan a profité pour lire un peu plus sur le fonctionnement de AmChart. Il s'est rendu compte rapidement qu'AmChart offrait beaucoup plus de fonctionnalité et permettait d'avoir des diagrammes beaucoup plus complexes que d'autre cadriciel tel que ChartJS.

CHAPITRE 2: 16 mai au 23 mai

Cydrick & Daniel

Lors de cette itération, Cydrick et Daniel ont travaillés ensemble afin de faire l'analyse à l'appel au service de calcul dans le but de pouvoir accomplir la tâche de sauvegarde données du service de calcul distant. Cette tâche leurs a demandés de faire des recherches et une rétro-ingénierie de l'application afin de trouver l'endroit dans le code qui fait un appel au serveur externe. En faisant plusieurs recherches dans le code, ils ont trouvé la source de l'appel dans la classe du contrôleur « *Spark* ».

Le code qui fait l'appel au service utilise le modèle « *BIMModels* ». Celui-ci possède l'identifiant pour indiquer quel modèle va se faire calculer. Une suite de configuration est nécessaire afin de compléter la communication avec le serveur externe. La capture d'écran suivante présente ce code.

```
//Mandatory post parameters
lstParam.add(new DataPart("ProcessId", model.getId() + ""));
lstParam.add(new DataPart("quantity", "1"));
lstParam.add(new DataPart("ProjectId", id + ""));
lstParam.add(new DataPart("LCIA", "true"));
lstParam.add(new DataPart("MonteCarlo", "true"));
lstParam.add(new DataPart("Montecarlo_Iterations", "20"));
lstParam.add(new DataPart("LCIAMethod", idMethod.toString()));
lstParam.add(new DataPart("prefix", util.Spark.getPrefix()));
lstParam.add(new DataPart("version", lastVersion));
```

Une fois ce code identifié, ils ont testé l'appel au service via lien www.locahost:9000/api/Spark/Calculate/1/95147. Ce lien fait appel au code arrière qui a été identifié ci-haut. Le chiffre *1* dans le lien indique l'identifiant du modèle et le chiffre *95147* indique le type de méthode de calcul.

En faisant l'appel à ce lien, plusieurs problèmes sont survenus, tels que: le serveur externe ne répondait pas et bien d'autres problèmes. Ces problèmes vont être détaillés à la semaine prochaine lorsque le promoteur a aidé à les régler, puisque nous n'arrivons pas à les régler durant la semaine.

Aravinthan & Jean-Pierre

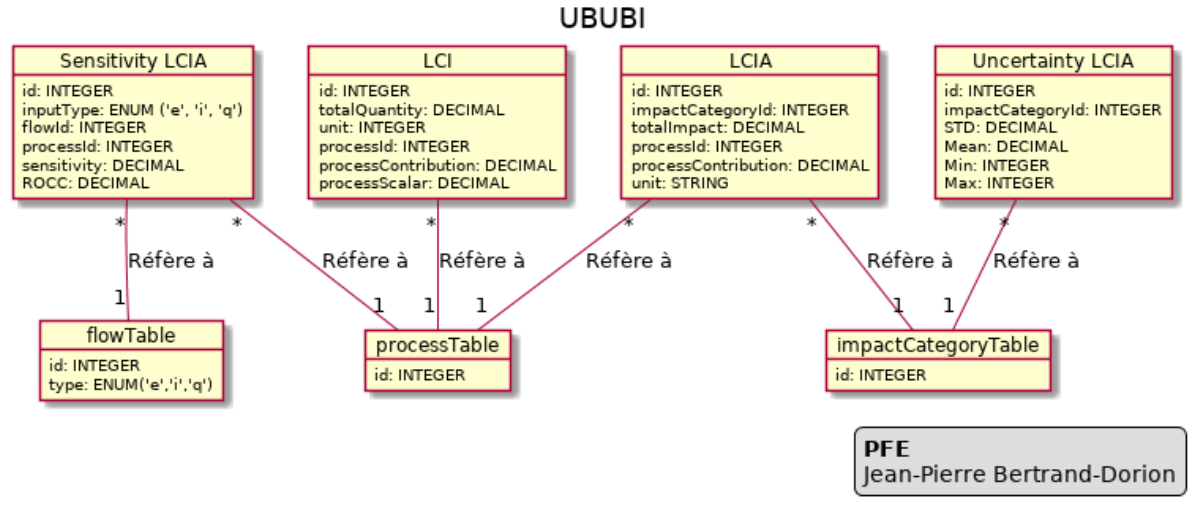
Pour cette itération, Aravinthan et Jean-Pierre ont eu comme tâche de revoir l'architecture de la base de données. Il faut qu'on puisse lier les tables LCI avec les tables process, flow et impactCategory qui leur est associé. Il fallait également trouver une architecture de la base donnée pour intégrer la table des resultSets. Il faut qu'on puisse avoir une relation entre cette table et les tables history. Pour cette semaine, le duo s'est concentré sur la première problème. Ils ont analysé les requis pour générer une solution qui répondra mieux aux besoins du projet. Une première esquisse de la solution a vu jour.

La première esquisse utilise un tableau de pivot pour pouvoir faire le lien entre les différentes tables LCI et les trois autres tables qui leur est associé. En utilisant une table de pivot, toutes les relations se retrouvent à un endroit et on évite de créer plusieurs tables avec des foreign keys. Cependant, cette solution rajoute de la complexité et n'est pas nécessairement intuitive

La seconde version présentait une architecture beaucoup plus simple, mais tout aussi efficace. Cette architecture lie directement la table des processus aux différentes tables LCI, dans une relation « OneToMany », ce qui allège la complexité tout en étant tout de même capable de répondre au besoin.

Par la suite, la table « flowTable » a été liée à la table « SLCIA » pour être utilisée à titre de référence dans les résultats « SLCIA ». Finalement, la table « impactCategoryTable » a été liée aux différentes tables LCI à titre de référence. Elle permettra de classer les résultats par catégories d'impacts plus tard dans le projet.

L'itération se termine avec une première proposition d'architecture de base de données qu'est la suivante puisqu'elle est plus simple et plus intuitive.



CHAPITRE 3: 23 mai au 30 mai

Daniel

Lors de cette semaine, Daniel a procédé à la sérialisation des données des fichiers CSV que génère le service de calcul. Cette étape a demandé une grande analyse afin d'obtenir la meilleure façon pour prendre les données du fichier et les séparer dans les modèles. La meilleure idée retenue fut l'utilisation des modèles Hibernate afin d'insérer les données. Cependant, les modèles n'étaient pas encore créés, car l'analyse pour la structure des tables de la base de données était encore en cours.

Il fallait donc implémenter un algorithme qui permet de prendre un fichier CSV, le sérialiser et créer des objets à partir de ces données. Puisque les modèles n'existaient pas encore, des tables et un modèle temporaires ont été créés seulement pour tester l'algorithme.

Pour les modèles temporaires, l'utilisation du polymorphisme lors de la sérialisation des données, car ceci faciliterait celle-ci. Une classe de type « *factory* » pourrait être à l'origine de la création des classes nécessaires et seuls les champs différents vont être pris en compte. Ceux qui sont identiques, ils vont être attribués de la même façon par la classe parente.

Cette idée n'a pas été poursuivie, car il y avait trop de champs différents, ce qui amenait des complications dans le code pour faire fonctionner la sérialisation. Donc il a été décidé de créer les objets *LCI, LCIA, SCLIA* et *ULCIA* qui représentent les quatre fichiers CSV retournée par le service de calcul. Les attributs sont en fonction des données documentées dans la documentation fournis par le promoteur.

En commençant cette tâche, l'accès aux fichiers contenant les vraies données n'était pas disponible. Donc des fichiers ont été créé avec des données fictives qui respectent le format de la réponse des appels de services.

La première version de l'algorithme prenait en entrée un lien vers le fichier voulant être sérialisé. Par la suite, un *BufferReader* permet de lire chaque ligne du fichier. Pour chacune ligne, les données sont séparées grâce à la fonction *split* et ensuite, elles étaient attribuées aux attributs de la classe correspondante.

Dépendant du type de fichier, une association des données au modèle cible est différente. C'est pour cela que l'utilisation d'un « *enum* » a été choisi pour les différencier. Cette façon de procéder permet de garder le code propre et d'en faire une utilisation efficace.

Cydrick

Durant la semaine 2, Cydrick a principalement travaillé sur le bon fonctionnement de UBUBI avec le service de calcul.

Au courant de la semaine précédente, il fut découvert que la méthode de calcul ne retournait rien et attendait un temps infini si un mauvais numéro de modèle était fourni ou s'il n'était pas présent. Ce problème fut relativement simple à régler une fois que la source du problème fut comprise. En effet, Play Framework fonctionne de manière asynchrone. Dans chaque méthode, il est possible de donner au Play Framework une promesse de réponse sous forme d'un objet « *CompletionStage<Result>* ». Puis, une fois le code exécuté et la promesse retournée, Play Framework exécute le code présent dans la promesse et cette promesse a la responsabilité de retourner un résultat (dans ce cas, un objet de type « *Result* »). Il est possible de chaîner ces promesses avec la méthode « *thenApply* » de telle manière que lorsque le résultat de la première promesse est retourné, son résultat est donné à une deuxième promesse.

Un mauvais emploi de la méthode « *thenApply* » causait cette attente infinie. En effet, le code devait retourner une erreur dans le cas d'un de numéro de modèle invalide était appelé dans une deuxième promesse (avec « *thenApply* »), sans mentionner la première. Par conséquent, cette deuxième promesse attendait un résultat d'une première promesse qui n'existe pas. La solution fut de marquer la promesse « complété » dès le début avec « *completedFuture* ». Voici le changement:

```
        BIMModel model = JPA.em().find(BIMModel.class,id);

        if(model==null)
-           return new CompletableFuture<Result>().thenApply(res -> jsonResult(notFound(EMPTY_REST_RESULT)));
+           return new CompletableFuture<Result>().completedFuture(jsonResult(notFound(EMPTY_REST_RESULT)));
```

Figure 2.2.1: Code de promesse corrigé

Ensuite, lorsque Cydrick a tenté d'appeler l'action de calcul dans l'application UBUBI, une erreur fût retournée par celle-ci:

{"exception":"@743hg2jgn: Execution exception in /opt/workspace/app/controllers/HomeController.scala:250"}

L'absence du fichier « *HomeController.scala* » dans UBUBI ainsi que le fait que le répertoire où se trouve le fichier a démontré que cette erreur était générée dans le service de calcul. Ce service de calcul, développé par un étudiant ne faisant pas partie de ce PFE, comportait une erreur. Après l'avoir signalé au promoteur du projet, celui-ci nous a confirmé que l'erreur était générée par le service de calcul et nous a donné le correctif à apporter, ce qui était de tout simplement d'envoyer un paramètre supplémentaire au service de calcul.

Ensuite, il a eu un problème intermittent concernant l'appel au service de calcul. Le service de calcul, étant déployé sur un serveur dans le réseau de l'ÉTS, devrait être en théorie fonctionnelle en tout temps. Toutefois, il n'était pas possible de contacter le service de calcul

à partir de l'extérieure de l'ÉTS, même lorsque le logiciel de VPN de l'ÉTS était exécuté sur l'ordinateur à l'extérieure de l'ÉTS. Le promoteur de projet a donc compris que le problème provenait des règles de pare-feu de l'ÉTS et a contacté le service d'assistance technique de l'ÉTS pour régler le problème.

Malgré tout, une autre erreur est survenue lorsque la méthode de calcul fut appelée:

{"exception":"@745846lil: Execution exception in /opt/workspace/app/controllers/FilesUtils.scala:51"}

Une fois de plus, cette erreur semblait provenir du service de calcul. Après l'avoir signalé au promoteur du projet, celui-ci a décidé de développer un Mock qui permet la simulation du service de calcul. Ce « Mock » permet de simplement téléverser les modèles BIM et retourne l'URL des fichiers contenant les données d'impacts environnementaux.

Pendant ce temps, Cydrick a amélioré le moyen de retourner des erreurs provenant de UBUBI. Auparavant, les erreurs générées dans UBUBI étaient soit attrapées et ignorées de manière silencieuse, soit l'erreur était affichée dans la console du serveur. Le retour envoyé au navigateur pouvait donc être vide ou rien (c.-à-d. un temps d'attente infini). Idéalement, il serait nécessaire d'envoyer l'erreur au navigateur pour que le code JavaScript puisse prendre une décision (par exemple, montrer l'erreur à l'utilisateur ou au moins lui signaler que quelque chose s'est mal passé).

Par conséquent, l'ensemble des clauses « *try-catch* » furent retirées de la classe « *Spark.java* » ainsi que de l'ensemble des méthodes qui étaient appelées par les actions. Ceci a permis de faire remonter les exceptions jusqu'au niveau de l'action qui s'occupe de retourner le résultat au navigateur. Puis, un « *try-catch* » englobant l'ensemble des opérations de l'action permet d'attraper toute erreur et de la retourner dans un format approprié. Par conséquent, le serveur va maintenant retourner une erreur de type 500 (erreur de serveur interne) avec le contenue suivant:

{"exception":"trace de la pile"}

Ce format permet la détection d'erreur (puisque le mot « exception » est présent) ainsi que son interprétation du côté du client JavaScript.

Jean-Pierre

Durant cette semaine, Jean-Pierre, en collaboration avec Aravinthan, ont mis à jour l'architecture de la base de données selon les commentaires du promoteur. Un nouveau diagramme du modèle logique a été généré pour démontrer ces changements.

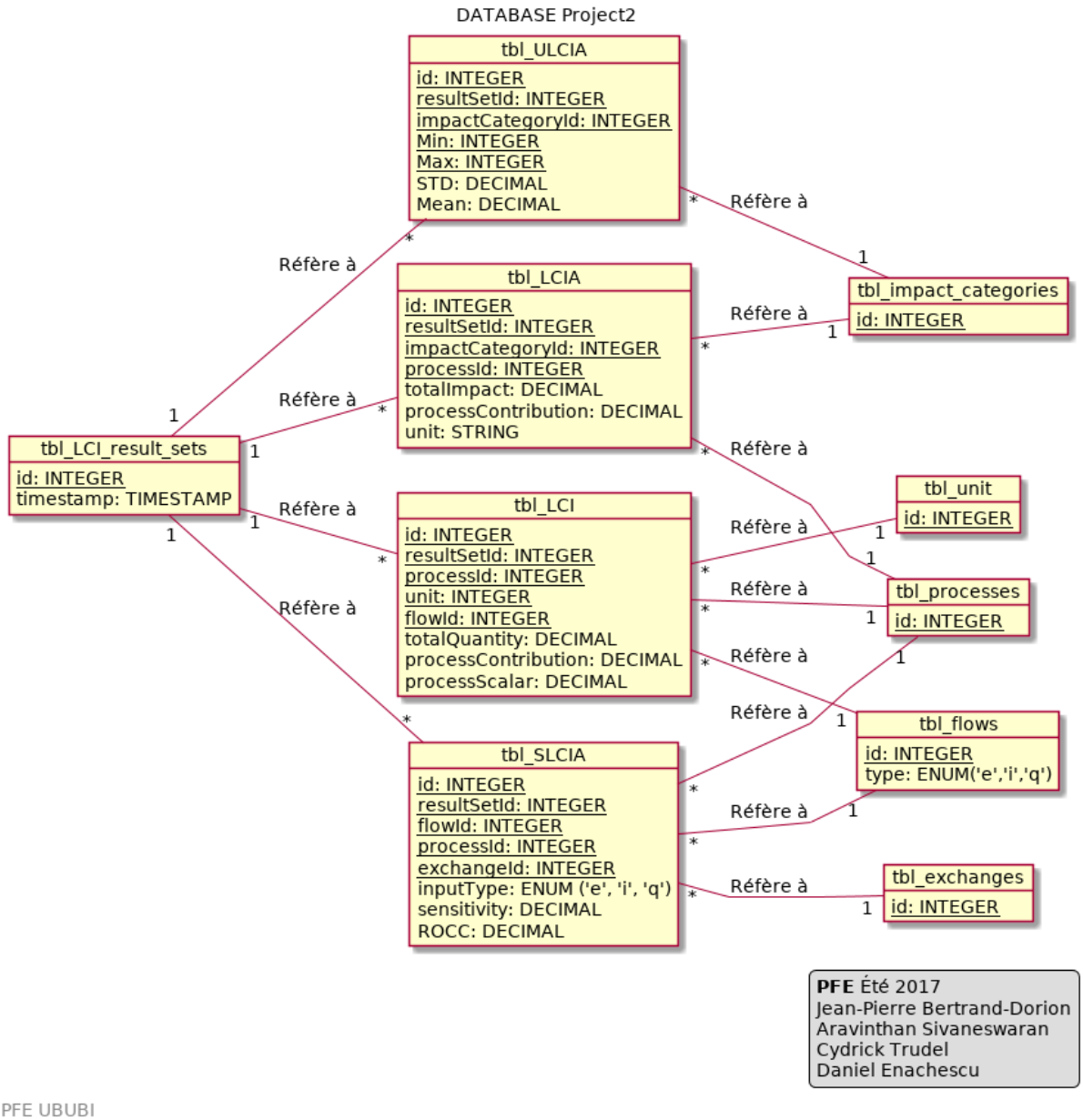


Figure 3.3.1 Diagramme final du modèle logique
de la base de données

Le modèle final est beaucoup plus complexe que l'esquisse réalisée lors de la semaine précédente. La table *LCIResultSet* fut ajouté pour lier ensemble les résultats obtenus lors d'une simulation, pour permettre de les retrouver facilement par la suite considérant que tous les résultats sont liés à un *LCIResultSet* commun. Il faut garder en tête qu'une requête peut contenir plusieurs centaines de milliers d'entrées, donc il est important de se faciliter la tâche (et celle des autres) avec ce genre de regroupement et ainsi éviter d'avoir à faire des recherches minutieuses ou qui pourraient potentiellement manquer de précision.

Deux autres liens ont été rajoutées: la table « *exchange* » à la table « *SLCIA* » ainsi que la table « *unit » à la table « LCI »*.

La table « *exchange* » est liée seulement à la table SLCIA car il s'agit d'un attribut pris en compte lors du calcul du SLCIA. Plusieurs SLCIA peuvent être liés à un seul « *exchange »* donc la relation est « *OneToMany »*.

La table « *unit* » est liée seulement à la table LCI. Elle permettra entre autres de faciliter les opérations mathématiques en normalisant les unités utilisées dans les LCI. En effet, il sera plus facile de travailler avec les données si elles ont une unité précise; les formules mathématiques seront adaptées en fonction des conversions et opérations possibles avec ces unités. De plus, ces unités sont utiles dans le travail d'architecture pour donner un sens aux résultats dans le monde réel.

Bref, cette itération conclut le travail associé à la mise à jour du modèle de la base de données. La prochaine itération traitera d'ajouter ces modifications à l'application.

Aravinthan

Durant cette semaine, Aravinthan avait comme mandat d'analyser les différentes façons de lier la table « ResultSet » aux différentes tables « history ». L'objectif est de pouvoir naviguer dans le temps et voir les « process », « flows » et « exchanges » tel qu'ils étaient pendant un calcul. Plusieurs idées ont été discutés, mais ont été rejetés rapidement puisqu'il ne cadrait pas les demandes spécifiques. Une des idées était par exemple de créer plusieurs tableaux qui auraient les données, mais pas l'action. Ainsi, une action peut être associée à plusieurs données. Cependant, cette idée rajoute beaucoup trop de « query » et rajoute de la complexité. En effet, il était hors de question d'utilise une méthode qui impliquait faire plus qu'un « query » pour la sélection À la fin, il y avait deux possibilités qui semblaient être les meilleurs pour la situation.

Il y avait la possibilité de créer un tableau qui servirait de pivot entre la table « ResultSet » et les tableaux « history ». En effet, à chaque fois qu'un calcul est inséré dans le tableau « ResultSet » , on va insérer une donnée dans cette table. On va insérer le « ResultSet id » et également le dernier id des différents tableaux « history ». Ainsi, quand on va chercher un « ResultSet » en particulier, on aurait également les « history » qui lui sont assignées. Cependant, il faut considérer qu'il peut y avoir des milliers et milliers de données. Alors, l'insertion et par la suite un « select » peuvent devenir très lourds pour la performance.

Une autre idée est d'utiliser les « timestamps » disponibles dans les tables « history » et « ResultSet » pour déterminer les changements qui ont été fait sur le modèle. Il suffit d'aller dans la première donnée disponible avant cette date et on aurait le « history » à jour. L'avantage de cette méthode est qu'on évite de faire des insertions inutiles. Cependant, la sélection peut prendre plus de temps puisque les table « history » peuvent comporter une grande quantité de données.

Aravinthan suggère d'utiliser la deuxième solution, mais veut en parler avec les autres membres du groupe et Mathieu pour avoir leur avis.

CHAPITRE 4: 30 mai au 6 juin

Daniel

Durant cette semaine, Daniel a poursuivi dans l'importation des fichiers. Il a également ajouté un bouton dans l'interface qui va servir plus tard pour lancer le calcul des résultats. Il a aussi fait des « Mock-up » des tableaux qui vont présenter les données afin de recueillir les commentaires du promoteur.

Il y a eu un changement dans le type de fichier cette semaine. L'extension des fichiers des données n'est plus de type CSV. Ils ont changé pour des fichiers texte. Cependant, le contenu des fichiers n'a pas changé. Ils contiennent quand même les données qui sont séparées comme dans un fichier CSV. L'algorithme fait lors de la dernière semaine est toujours valable.

La semaine dernière, le promoteur a créé un « Mock » afin qu'il nous retourne les liens vers le serveur pour obtenir les fichiers contenant les données. Il fallait d'abord décomposer la réponse « JSon » du « Mock » pour garder les liens. Ensuite, il fallait les enlever des apostrophes qui se situaient au début et à la fin de chaque lien.

Une fois que les liens avaient le bon format, un « *Stream* » a été utilisé pour importer les fichiers localement. Il a fallu créer des fichiers locaux et insérer les données bit par bit. Le nom des fichiers locaux était représenté par des constantes afin de garder le code propre. L'utilisation d'une « *Map* » pour associer les noms des nouveaux fichiers avec son type a été utilisée afin de connaître le type de chaque fichier. Les nouveaux fichiers étaient ajoutés localement dans le projet temporairement pour la lecture des données.

Une deuxième tâche accomplie a été d'ajouter un bouton qui va permettre de faire le calcul d'un modèle. Ce bouton a été ajouté dans l'interface à côté de la sélection du modèle. Il a été connecté au backend par un appel en JavaScript utilisant les routes. Une méthode qui comprend seulement l'appel a été créée afin de faire la connexion. Présentement, le backend retourne un « *Ok* ». Dans les prochaines semaines, une tâche va être de compléter la logique du bouton.

La dernière tâche accomplie fut la création de « Mock-up » pour présenter des tableaux contenant plusieurs données. Ces « Mock-up » ont été faites sur papier. Cette décision a été prise puisque cela nous a permis pendant la prochaine réunion de dessiner directement dessus suite au commentaire du promoteur. Ceci approche permet d'avoir une meilleure interaction de l'équipe et avec le promoteur, car tout le monde a accès finalement à l'image sur le papier devant lui. La figure suivante (Figure 4.1.1) suivante présente ces Mock-up.

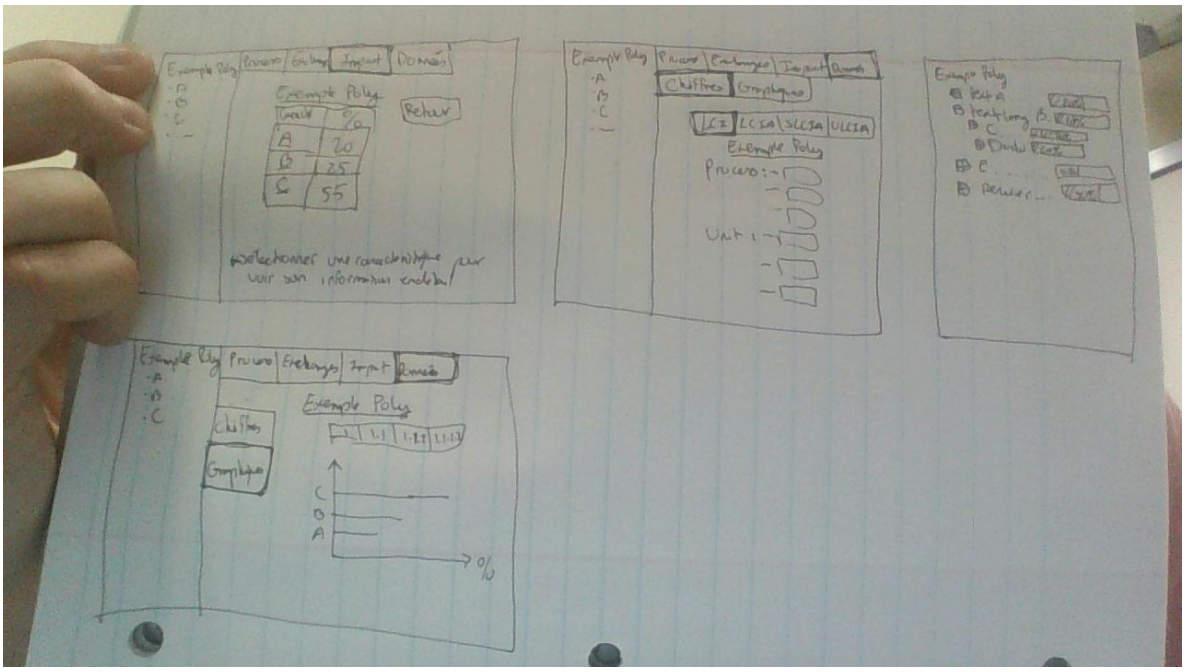


Figure 4.1.1: Mock-up présentant les données dans l'interface

Dans la partie en haut à droite, on retrouve la section des impacts des processus. C'est une partie qui permet à l'utilisateur d'accéder à plus de détails sur l'impact du processus sélectionné. Ces impacts sont présentés par des pourcentages dépendant des autres types de processus.

Dans la partie en haut au milieu, on retrouve la section des données. Celles-ci présentent toutes les données reliant au processus. On y retrouve les intrants comme les extrants. Elles y sont présentées de façon numérique.

Dans la partie en bas à gauche, on retrouve ces mêmes données, mais présentées de façon graphique. On peut voir qu'il a un sous-onglet qui permet de naviguer entre les données de façons numériques ou graphiques.

Cydrick

Durant cette itération, Cydrick a débuté la tâche d'exportation des données reçues par le service de calcul vers un classeur Microsoft Excel. Pour débuter, une phase d'analyse des besoins concernant l'exportation de ces données était nécessaire.

Tout d'abord, il était suggéré d'utiliser le format Excel XML puisque le taux de compression natif de ce format est beaucoup plus élevé que le format propriétaire de Microsoft Excel (le format .xslx). De plus, le format Excel XML est public. Il y a donc moins de chance que Microsoft y apporte des changements drastiques ou délaisse la compatibilité de ce format.

Après quelques observations et tests d'exportation vers ce format, Cydrick a pu constater que ce format comporte une haute verbosité. Le nombre de noeuds XML nécessaires pour générer un fichier fonctionnel est relativement élevé, même pour un ensemble de données minimal. Voir l'annexe III pour un exemple d'un fichier Excel XML.

Il fut aussi décidé d'obtenir les données dans la base de données et non du service de calcul. On suppose que l'utilisateur désire visualiser les données en ligne avant de télécharger le fichier Excel contenant ces mêmes données. Ceci a pour avantage de permettre la génération d'un fichier Excel avec d'anciennes données sans devoir refaire un appel au service de calcul, ce qui est plus rapide. De toute manière, si l'utilisateur désire obtenir un fichier Excel sans visualiser les données, il sera possible d'appeler l'action de calcul (qui va insérer les données dans la base de données), puis d'appeler l'action qui va générer le fichier Excel (qui va lire les données de la base de données)

Après une conversation avec le promoteur de projet, celui-ci a demandé de simplement exporter dans le fichier Excel l'ensemble des données d'un « *ResultSet* » (voir Annexe I pour les détails). Par contre, il a expliqué que plusieurs modifications du fichier seraient nécessaires durant le PFE, notamment l'ajout et retrait de données, le changement de la présentation des données et l'agrégation de résultats. Par conséquent, il fut nécessaire de considérer une haute maintenabilité sur le gabarit du fichier Excel, c'est-à-dire sur l'ensemble des éléments nécessaires pour qu'Excel puisse lire le fichier, mais excluant les données.

De plus, la quantité de données retournées par le service de calcul est considérable. Pour un seul « *ResultSet* », 4 fichiers sont retournés. Voici un exemple de « *ResultSet* »:

- LCI: 242320 lignes, 6 colonnes;
- LCIA: 34309 lignes, 5 colonnes;
- SLCIA: 1800 lignes, 5 colonnes;
- ULCIA: 18 lignes, 12 colonnes.

Il est donc nécessaire de considérer une grande quantité de données à inclure dans le fichier Excel.

Cydrick a ensuite considéré l'ensemble des options pour générer un tel fichier à partir du

code, pour pouvoir y insérer les données provenant du service de calcul. Il a identifié trois grandes approches. Voici un résumé des trois approches, leurs avantages et inconvénients:

Approche 1: Génération par *Element* XML
Java permet la génération de structures XML à l'aide d'objets de type *Element*. Il est possible d'imbriquer les *Elements* les uns dans les autres pour recréer les structures XML qui possèdent une profondeur. Cette approche consisterait à générer entièrement la structure du fichier incluant les valeurs à l'aide d'*Element* imbriquée pour satisfaire le format de fichier Excel XML. Puis, les *Element* XML seraient convertis en un fichier XML.

Avantages:
Cette approche est facile à utiliser. Il est facile de configurer et de comprendre comment les *Element* fonctionnent. Cette approche est aussi relativement bien maintenable. Il serait facile de déplacer une sous-structure d'*Element* pour la rattacher ailleurs dans la structure.

Inconvénients:
La génération d'*Element* est très lente. Quelques milliers d'éléments sont suffisants pour constituer un temps considérable de génération. De plus, le développement initial d'une structure d'*Elements* aussi volumineuse que celle nécessaire pour le gabarit d'Excel XML peut prendre un certain temps puisqu'il y a beaucoup d'éléments et d'attributs à considérer.

Approche 2: Génération par texte
Il serait possible de générer le gabarit Excel nécessaire, de le séparer en sections et de l'inclure dans le code Java à l'intérieur de variables de type *String*. Ceci impliquerait de devoir séparer le gabarit en plusieurs sections. Par exemple, une section pour l'entête de fichier, une section pour l'entête de feuille de calcul Excel, une section pour l'entête de ligne (se situant dans une boucle itérative), etc. Puis, les différentes variables *String* seraient concaténées pour générer le fichier Excel XML.

Avantages:
Le temps de génération est très rapide. Des tests ont été effectués et selon une estimation, il pourrait être possible de générer le fichier avec cette approche en environ 13sec. Le développement de cette approche est aussi relativement rapide puisqu'il s'agit simplement de copier-coller des portions du gabarit à plusieurs endroits

Inconvénients:
Cette approche offre une très mauvaise maintenabilité. Par exemple, si un des styles de cellules doit être modifié, il serait nécessaire de le modifier dans plusieurs sections disjointes dans le code, incluant dans des boucles qui créent des lignes. Cette modification à plusieurs endroits différents et disjoints peut mener à certains problèmes de maintenabilité.

Approche 3: Génération par combinaison de gabarit et de valeurs
Il serait possible d'envisager un système où le gabarit du fichier Excel se trouve dans un fichier et possède des valeurs spéciales (ou étiquettes) montrant où insérer les données. Puis, les données sont générées et insérées dans une structure spéciale qui permet d'attacher une

étiquette à chaque valeur. Finalement, un transformateur remplace les étiquettes du gabarit par les bonnes données. Ce transformateur doit supporter les structures itératives, puisque chaque feuille de calcul Excel doit contenir plusieurs lignes.

Avantages:
Cette approche offre une maintenabilité raisonnable. En effet, le gabarit se trouve dans un seul fichier. Il est relativement facile de comprendre la structure de ce fichier à l'aide de l'indentation présente. De plus, les performances sont aussi raisonnables. Dans le cas d'un transformateur XML, il serait nécessaire de fournir les données à l'aide d'une structure d'*Element* ce qui est relativement lent, mais moins que si le gabarit serait aussi généré en à l'aide d'une structure d'*Element*.

Inconvénients:
Cette approche requiert une étape de configuration relativement difficile. Des aspects de cache et d'ordonnancement d'éléments sont à considérer pour obtenir des performances optimales.

Cydrick a décidé de choisir la troisième approche, c'est-à-dire la génération par combinaison de gabarit et de valeurs. Cette approche permet un compromis entre performance et maintenabilité.

Cydrick a ensuite fait un prototype de cette approche utilisant la librairie Apache Xalan. Cette librairie permet la transformation d'un gabarit sous un format XSLT vers un fichier XML. Le gabarit utilise notamment les balises *<xsl:foreach select="...">* pour dénoter une boucle sur une structure d'*Elements* et *<xsl:value-of select="..." />* pour y insérer une valeur. Ce prototype ne retournait toutefois aucune donnée réelle, seulement des nombres aléatoires.

Il est important de comprendre que cette librairie permet seulement de transformer un gabarit ayant le format XSLT vers un fichier XML.

Jean-Pierre

Dans cette itération, l'objectif pour Jean-Pierre était d'ajouter les modèles dans l'application et la base de données. Au lancement, l'application communique avec la base de données et s'occupe de l'ajout des tables manquantes si des nouveaux *modèles* sont apparus.

Cinq nouveaux *modèles* ont été ajoutés à la base de données:

1. LCI.java
2. LCIA.java
3. LCIResultSet.java
4. SLCIA.java
5. ULCIA.java

Pour coder un *modèle*, il faut créer une classe et le déclarer comme un objet Java avec des attributs. Ces attributs ont une visibilité et un type de donnée associé comme dans une classe Java ordinaire. En ce qui concerne le côté SQL, les spécificités reliées aux données sont déclarées explicitement dans la classe du *modèle* avec une notation semblable au Javadoc. On peut ajouter des propriétés tel déclarer que l'attribut est un « id » (donc une clé primaire), ou bien s'il s'agit d'un « *ENUM* » et même déclarer le type de relation qu'un attribut a avec d'autres tables (une clé étrangère, par exemple « *ManyToOne* » ou « *ManyToMany* »).

Au démarrage de l'application, les *modèles* manquant dans la base de données sont créés (chaque *modèle* équivaut à une nouvelle table. Cette façon astucieuse de procéder assure à l'application qu'il n'y aura pas de problème lorsqu'elle aura à faire des transactions avec la base de données.

On peut aussi définir des fonctions pour générer des valeurs pour les attributs lors de leur création, telle que dans le *modèle* « LCIResultSet.Java » :

```
@Data
@Entity
@Table(name="tbl_LCIResultSet")
public class LCIResultSet {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    private Timestamp calc_time = Timestamp.from(Instant.now());
}
```

Ici, on souhaite ajouter un *Timestamp* à l'objet, donc on utilise une fonction tout simplement.

Une classe de *modèle* ressemble à ceci :

```java
@Data
@Entity
@Table(name="tbl_LCI")
public class LCI {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(nullable = false)
    private LCIResultSet resultSet;

    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(nullable = false)
    private Process process;

    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(nullable = false)
    private Unit unit;

    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(nullable = false)
    private Flow flow;

    private double totalQuantity;

    private double processContribution;

    private double processScalar;
}
```

Les modèles sont très semblables; ils contiennent chacun une référence vers un « *Process* » et un « LCIResultSet ». Il est important de mentionner que la table dans la base de données portera le nom déclaré via la déclaration "*@Table*" préalablement à la classe.

Bref, la fin de cette itération marque la fin des travaux associés aux bases de données pour l'équipe. Les modèles sont ajoutés tel que convenu auparavant et nous pouvons maintenant continuer avec d'autres tâches.

Aravinthan

Après avoir discuté des deux possibilités pour l'architecture de la base de données, Aravinthan s'est donné la tâche de faire des tests pour évaluer les avantages/inconvénients des deux possibilités. Il devait également analyser plus en profondeur pour établir les avantages et les désavantages des différents libraires « Charts » qui étaient disponibles en logiciel libre.

Il avait commencé la semaine en regardant les différentes librairies de « Charts ». Le premier était le ChartJs. C'était très rapide pour lui parce qu'il l'avait déjà utilisé alors il connaissait déjà le fonctionnement. Après avoir fouillé un peu plus en profondeur, il s'est rendu compte très rapidement que faire des diagrammes complexes était très pénible avec ChartJs. En effet, après avoir lu la documentation, regarder sur « Stackoverflow[4] » et faire des tests, c'était difficile de faire les diagrammes requis par l'application UBUBI. Puisque Jean-Pierre s'occupait d'investiguer les autres possibilités, c'est-à-dire AmCharts et D3 Charts, Aravinthan s'est penché sur les tests de la base de données.

Pour commencer, il s'est concentré principalement sur la lecture de la documentation pour voir quelle méthode[5], entre l'utilisation d'une table pivot et les timestamps, serait le plus efficace. Malheureusement, il n'a pas trouvé d'informations qui pointaient clairement sur une des deux solutions. Par la suite, il a lu plusieurs discussions sur StackOverflow[5] et d'autres forums pour voir ce que d'autres programmeurs pensaient. Cependant, il y avait qu'un consensus pour dire que cela dépendait de la situation. Ensuite, il s'est entretenu avec des collègues de travail et amis dans le domaine. Il en est arrivé à la conclusion que les deux solutions étaient valables et il fallait avoir un peu plus d'informations pour trouver ce qui serait le mieux pour cette application.

[4] Soham Dasgupta. https://stackoverflow.com/questions/33675468/chartjs-drill-down-issue-with-multiple-y-axis-bar-line-graph

[5] Guillermo Guitérrez. https://stackoverflow.com/questions/7448453/sql-server-pivot-vs-multiple-join

CHAPITRE 5: 6 juin au 13 juin

Daniel

Dans cette semaine, Daniel a changé le type d'importation. Également, il a connecté la sérialisation faite la semaine dernière avec l'importation des données. À la suite de cette connexion, il a persisté dans la base de données les modèles comportant les données.

Lors de la dernière réunion, le promoteur a montré une façon qu'il utilisait déjà pour appeler une page web en utilisant le cadriciel « WSClient » de *Play framework*. Une réingénierie du code d'importation fut nécessaire pour utiliser le cadriciel. En supprimant l'importation des fichiers du serveur dans le projet local, une lecture à distance du fichier est faite. Cette lecture est plus efficace que d'importer le fichier au complet et de le lire ligne par ligne. Malgré ces modifications, l'algorithme pour sérialité est resté intact.

À la suite de ce changement de type d'importation de données, la sérialisation a été connecté avec l'importation. Le seul ajustement qui a été nécessaire de faire est d'enlever la façon de boucler à travers un fichier pour le remplacer par une boucle dans une réponse « JSon ». Celle-ci a été renvoyée par le serveur contenant les données.

Une fois en possession des données, il a fallu créer les instances des modèles en question pour ensuite attribuer les bonnes valeurs aux attributs respectifs. Une petite correction a été nécessaire aux attributs des modèles puisqu'ils y en avaient qui était déclaré du mauvais type. Il a également fallu modifier un attribut pour être un « *Enum* » à la place d'un caractère. Les modèles qui ont été utilisé cette semaine étaient ceux développé par un coéquipier. Une réingénierie du code a été nécessaire afin de les utiliser.

Après avoir attribué les bonnes valeurs aux modèles, il fallait les persister. Le choix dans cette situation a été d'utiliser la fonction « *JPA* » d'Hibernate. Cette décision a été prise, parce que les modèles étaient faits en fonction des tables de la base de données. Ce fut une utilisation facile, puisque celui-ci ne demande pas beaucoup de code pour enregistrer les modèles.

Cependant, pour chaque modèle qui possédait des entités d'autres classes, il fallait s'assurer d'aller les chercher dans la base de données avant de pouvoir insérer les nouveaux modèles contenant les données du serveur externe. C'était une étape supplémentaire ce qui faisait partie entre autres de la lenteur de l'insertion des données. La capture d'écran suivante présente un exemple d'une insertion.

```
+        ULCIA ulcia = new ULCIA();
+        ulcia.setResultSetId(new LCIResultSet());
+
+        ImpactCategory impactCategory = (ImpactCategory)entityManager.createQuery("select ic from ImpactCategory ic where ic.id =
:icId")
+                .setParameter("icId", Long.parseLong(lineWithDatas[0]))
+                .getSingleResult();
+        ulcia.setImpactCategory(impactCategory);
+
+        ulcia.setStd(Double.parseDouble(lineWithDatas[1]));
+        ulcia.setMean(Double.parseDouble(lineWithDatas[2]));
+        ulcia.setMinimumImpactCategory(Double.parseDouble(lineWithDatas[3]));
+        ulcia.setMaximumImpactCategory(Double.parseDouble(lineWithDatas[4]));
+
+        save(ulcia);
```

En faisant un test final pour s'assurer que le code fonctionnait, un problème est survenu. Le programme prenait trop de temps pour insérer chaque modèle. Par exemple, seulement pour compléter l'insertion des données des objets *LCI*, le temps était d'une heure et demie. De plus, j'utilisais un lien pour accéder directement à l'appel du service. Ce temps ne prenait pas en compte le temps pour appuyer un bouton dans l'interface et faire l'appel jusqu'au backend.

En implémentant ces algorithmes, je pensais que cela allait suffire pour satisfaire la demande du promoteur. Cependant, nous allons voir dans les prochaines semaines que j'ai eu besoin de faire plusieurs modifications.

Cydrick

Durant cette semaine, Cydrick c'est concentré sur la première génération d'un fichier XML Excel contenant les données provenant du « Mock » du service de calcul. Il fut nécessaire d'attendre en partie le travail de Daniel, puisque celui-ci travaillait en même temps sur l'importation des données vers la base de données.

Pendant ce temps, Cydrick a développé le premier gabarit de fichier Excel XML. Ce gabarit permettait l'affichage de l'ensemble des données retournées par le service de calcul sans aucune modification ou agrégation. Consultez l'annexe IV pour voir le détail de ce gabarit.

La structure d'objets créée pour supporter la transformation de gabarits était à ce moment séparée en 3 parties. Voici le détail de chacune des classes:

- *CalculationApiXmlData.java*: Une classe qui s'occupe d'obtenir les informations à partir de la base de données à l'aide d'Hibernate et retourne une structure d'*Element* XML structuré dans un format interprétable par le gabarit.
- *template.xsl*: Le gabarit Excel XML sous le format XSLT. Accepte les données retournées par *CalculationApiXmlData.java*.
- *ExcelXmlFileCreator.java*: Combine *template.xsl* et les données retournées par *CalculationApiXmlData.java*, ajoute des métaéléments (la date de génération du fichier) et écrit le fichier sur le disque.

L'avantage avec cette structure est qu'il serait possible d'utiliser « *CalculationApiXmlData.java* » pour générer d'autres choses que le fichier Excel XML, tant que cette nouvelle approche prend les données dans le format d'*Elements* XML. Il serait aussi possible d'utiliser un autre sous-ensemble de données, tout en gardant le fichier *template.xsl* puisque les données et le gabarit sont séparés.

L'action qui permet la génération et le téléchargement de ce fichier était « */api/Spark/GetFile/ResultSetId* », où « *ResultSetId* » est l'« id » du « *ResultSet* » (par exemple, *1*). Consultez l'annexe I pour plus de détails sur le « *ResultSet* ».

Il est important de comprendre qu'à ce stade, les données sont obtenues à l'aide de la librairie Hibernate. Cette librairie gère de manière transparente la sérialisation et la déserialisation des données d'objets vers la base de données et vice-versa. Il n'est donc pas nécessaire d'écrire des requêtes SQL pour lire les données dans la base de données et les charger dans un objet et vice-versa puisque Hibernate fait ce traitement. Une autre particularité importante de cette approche est que le fichier est sauvegardé sur le disque, puis est retourné à l'utilisateur qui l'a demandé.

Lorsque la tâche de Daniel fut accomplie, une première génération de fichier a été faite. Avec cette approche, voici les temps obtenus pour les données de tests (ces mêmes données seront utilisées tout au long du PFE):

- Obtention des données avec Hibernate: 1h 1min 32sec;
- Transformation du gabarit sur les données avec Apache Xalan: 2min 55sec;

- Téléchargement du fichier localement: 5sec;
- Total: 1h 4min 32sec.

Jean-Pierre

Lors de cette itération, Jean-Pierre est responsable de créer des entêtes de fichier pour indiquer la licence Open-Source du projet.

Cette nouvelle tâche se divise en trois parties :

1. Déterminer ce que l'entête contiendra comme information.
2. Appliquer l'entête sur les fichiers existants.
3. Trouver une méthode pour partager ces entêtes aisément

En premier lieu, il faut considérer ce qui doit être dans un entête. L'entête doit contenir au minimum le nom du détenteur des droits (c.-à-d. dans ce cas, le promoteur) avec la mention « *copyright* » ainsi que l'entête de la licence que l'on souhaite associer au fichier. Il est important de mettre la licence dans une seule langue (habituellement en anglais seulement) pour éviter d'invalider son poids légal. La licence choisie par le projet UBUBI lors de sa création est la licence Apache 2.0, qui permet à un contributeur de conserver ses droits tout en permettant plusieurs collaborateurs de se joindre au projet, et ce sans problème légal. Ce choix de licence permettra au projet de devenir un logiciel libre lors de sa publication officielle.

Durant ses recherches, Jean-Pierre a trouvé un outil s'appelant « *header-generator* » sur Github, un projet libre qui permet d'automatiquement appliquer des entêtes aux fichiers en fonction de leur extension. Cet outil sera utile dans la prochaine itération. Ce projet fournit des exemples d'entêtes modifiables par la suite par l'usager. L'entête de base est la suivante :

```
/* <FILENAME>
 * File: <FILE>
 * Auth: <USERNAME> <EMAIL>
 * Date: <DATE>
 */
```

L'entête sera modifié pour contenir la licence et le nom du promoteur pour tous les modules existants et permettra d'être amendé par les membres du projet là où ils ont contribués. Le « *filename* » et le champ « *file* » sont répétitifs et inutiles donc seront enlevés. Nous ajouterons le champ « *Project* » pour spécifier que le fichier fait partie du projet UBUBI, et la date ne sera qu'un descripteur pour la session en cours.

Après modification, l'entête ressemble à ceci :

```
/**--------------------------
* PROJECT: UBUBI
* Auth:
*   Mathieu Dupuis
* Mail: mathieu.dupuis@ububi.org
* Date: ETE 2017
*
* Copyright [2017] [Mathieu Dupuis]
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*    http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*
*-------------------------*/
```

Cette entête convient à nos requis et projette une apparence professionnel. Elle sera appliquée aux fichiers existants lors de la prochaine itération. Pour terminer, Jean-Pierre a trouvé qu'il est possible de facilement partager des entêtes directement via l'IDE Intellij-Idea, en ajoutant les entêtes dans le dossier « .idea » du répertoire du projet. Ce sujet sera traité plus en détail dans l'itération de la semaine prochaine.

Aravinthan

En ayant obtenu des informations additionnelles, Aravinthan a débuté des tests pour voir la performance des deux solutions[6]. La première solution est l'utilisation d'un table pivot et la deuxième est l'utilisation des timestamps pour faire la liaison. En utilisant la base de données sur la machine virtuelle, Aravinthan a inséré des données de test et ensuite a exécuté les query SQL. Les données étaient très semblables. Mais il a été convenu qu'il fallait avoir plus de données pour pouvoir avoir une meilleure idée de la performance de la solution proposée. Il s'est donc penché sur un langage qu'il connaît bien, le PHP. Il a utilisé le cadriciel Laravel[6] , qui permettait de faire des « mocks » d'objets et les utilisera dans les tests. Il a donc pu insérer des milliers de données et testé les deux alternatives de solution. Les deux avaient un temps similaire, mais il y avait un léger avantage pour la solution qui évitait de faire un « query » de plus. C'est pour cette raison que la solution des « timestamps » a été choisie.

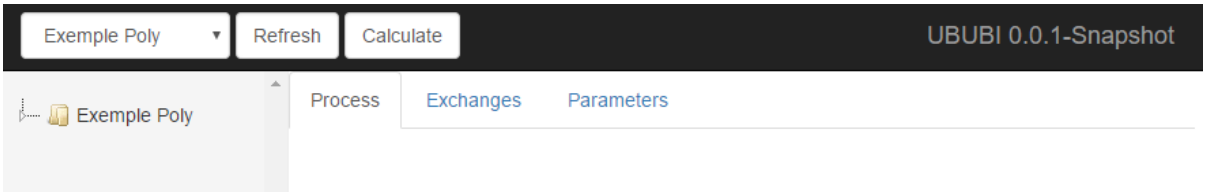
[6] Laravel. https://laravel.com/

CHAPITRE 6: 13 juin au 27 juin

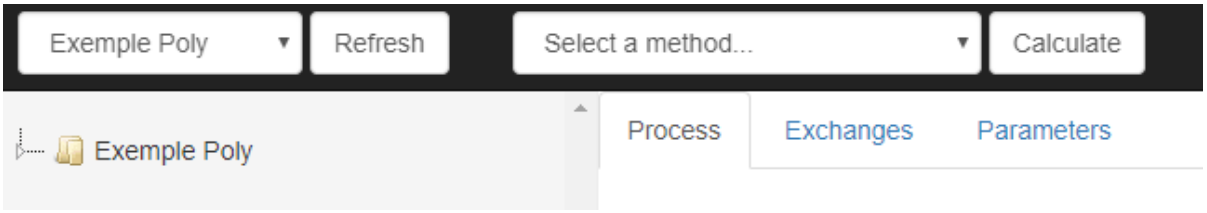
Daniel

Dans cette semaine, Daniel a complété le flux du calcul en ajoutant un bouton dans l'interface avec une liste déroulante comprenant des méthodes de calcul. Une modification de l'algorithme pour insérer les données a été faite également après la demande du promoteur. Les détails de cette modification se trouvent dans le texte plus bas.

Afin de connecter l'importation des données, il a fallu ajouter un bouton dans l'interface ainsi qu'une liste déroulante pour permettre à l'utilisateur de choisir sa méthode de calcul. Avant d'ajouter cette liste déroulante, le bouton a été ajouté. Je pensais que c'était suffisant pour lancer l'importation. L'image suivante présente la façon dont ce bouton était placé dans l'interface.



À la suite des commentaires du promoteur, j'ai ajouté la liste déroulante. Elle est très importante, car dans l'appel de services du serveur externe, il faut préciser la méthode d'analyse d'impacts. La capture d'écran suivante présente le nouvel emplacement de cette liste avec le bouton.



Dans cette liste déroulante, on y retrouve le nom des méthodes provenant de la base de données. Une méthode dans le backend a été ajoutée afin d'aller chercher toutes les méthodes. Le code suivant présente la façon qui a été utilisée pour précéder à cette démarche.

```
@Transactional(readOnly = true)
public Result getAllImpactMethod() {

    Query query = JPA.em().createQuery( qlString: "Select m from ImpactMethod m");

    List<ImpactMethod> impactMethods = query.getResultList();
    if (impactMethods == null || impactMethods.size() == 0)
        return jsonResult(notFound());

    return jsonResult(ok( content: "{"+JSONUtil.parseArray( key: "lst",impactMethods,JSONFormater.ImpactMethodFormater)+"}"));
}
```

Il est important de mettre dans le « *Transactional* » que c'est une action qui demande seulement une lecture. Dans le code, l'utilisation de *JPA* pour créer une requête fut la

meilleure option, car c'est la même méthode qui est utilisée dans le fichier et c'est une façon qui rend le code propre. Ce choix implique également de garder une cohésion de méthodes dans la classe. Dans le cas où il n'y a pas de méthode, une réponse indiquant qu'il n'y a pas de résultat est envoyée à l'interface. Ceci va permettre à l'utilisateur de voir qu'il y a un problème et de signaler le promoteur. En toute circonstance, il doit toujours y avoir des méthodes.

Afin de faire la connexion de l'interface avec le backend, il a fallu insérer un appel en JavaScript. Dans celui-ci, on y retrouve un appel via les routes du service et lorsqu'on reçoit la réponse, on populle la liste déroulante avec le nom des méthodes. L'identifiant de chaque méthode est attribué comme valeur afin de l'envoyer lorsque le calcul est activé. L'image suivante présente cet algorithme.

```
getImpactMethodsAjax().done(function(modelsJson) {
    let impactMethods = modelsJson.lst;

    for(let impactMethod of impactMethods) {
        methodSelect.append(`<option value="${impactMethod.id}">${impactMethod.name}</option>`);
    }
});
```

Lorsque le bouton de calcule a été ajouté dans l'interface, la méthode dans le backend prenait seulement en compte l'identifiant du modèle sélectionné. Il a fallu que j'ajoute dans les paramètres l'identifiant de la méthode sélectionnée.

Une autre tâche que Daniel a accomplie est la modification de l'algorithme d'insertion des données importées. En effet, la nouvelle stratégie est d'insérer directement les données dans la base de données sans faire de vérification. On accorde une certaine confiance aux programmeurs qui s'occupent de nous envoyer les données et que les clés secondaires correspondent avec les clés primaires des autres tables.

Cette décision a été prise puisque l'insertion avec Hibernate était trop longue. Également, on voulait avoir plus de rapidité afin que l'utilisateur n'attende pas trop longtemps après avoir appuyé sur le bouton « *Calculate* ».

Le code qui a été produit lors de la dernière semaine pour sérialiser les données ne sert plus pour cette nouvelle stratégie. Il faut alors tout supprimer afin de garder l'espace de travail propre. Il est important de retirer le code mort dans un logiciel.

La façon d'aller chercher les données avec la fonction « WSClient », est restée pareil. Cependant, la nouvelle façon d'insérer les données nécessite l'utilisation des « *PrepareStatement* ». Cette méthode a permis d'attribuer correctement les données dans les bonnes colonnes des tables.

Cette façon de procéder a beaucoup accéléré le processus d'insertion. J'ai réussi à réduire le temps d'une heure et demie à sept minutes. Il y a eu amélioration du temps, car les « *PrepareStatement* » ne prennent pas en compte des modèles comme ceux de Hibernate. Il fait

simplement insérer les données qui leur sont fournies.

Cydrick

Cette semaine, Cydrick s'est concentré sur l'amélioration du temps d'obtention des données ainsi que certaines améliorations au gabarit Excel demandées par le promoteur de projet.

Tout d'abord, le promoteur du projet a signalé que le temps de génération du fichier était trop long. En effet, plus d'une heure pour générer le fichier est trop long, considérant que l'ensemble des fichiers retournés par le service de calcul prennent que quelques secondes à calculer et à générer. La première amélioration considérée fut de réduire le temps d'obtention des données dans la base de données puisque cette étape semble être la plus longue, soit de 1h 1min 32sec.

Cydrick a tout d'abord tenté de créer une classe Java pour mettre en cache les objets fréquemment utilisés, soit les Impact Category, les Flow, les Unit et les Process. Ces objets sont souvent utilisés durant l'obtention de données comme valeur de jointure entre deux autres objets. De manière plus technique, l'approche consistait à faire ceci:

1. Vérifier si l'objet désiré se trouvait dans une *Map<ID, CachedObject>* (où *ID* représente la clé primaire de l'objet dans la base de données et *CachedObject* l'objet désiré);
2. Si l'objet s'y trouve, retourner l'objet;
3. Si l'objet ne s'y trouve pas:
    3.1. Aller chercher l'objet dans la base de données à l'aide d'Hibernate;
    3.2. Le sauvegarder dans le *Map<ID, CachedObject>*;
    3.3. Le retourner.

Cette logique a été implémentée dans la classe *DatabaseCache*. Toutefois, cette approche semble être déjà implémentée dans Hibernate puisque les gains de temps étaient de l'ordre de quelques secondes seulement.

Par conséquent, Cydrick a pris la décision de contourner Hibernate et d'utiliser des requêtes SQL natives codées manuellement pour accéder aux données. Les changements prirent un certain temps à implémenter puisqu'il fut nécessaire de remplacer la logique d'obtention des données à l'aide d'accesseurs chaînés (ex: *object.getX().getY().getZ()*) par une approche où la requête SQL sélectionner exactement les bonnes données.

Il fut aussi particulièrement complexe de représenter les données de type LCIA avec cette nouvelle approche puisque ces données sont représentées dans un tableau bidimensionnel où le nombre de processus (les lignes) et le nombre de catégories d'impact (les colonnes) sont variables, et où il n'y a pas nécessairement une valeur présente pour chaque catégorie d'impact dans un processus. Il est donc nécessaire de représenter ces données sous forme d'une matrice creuse (*Sparse Matrix*) dans Excel. L'approche préconisée dans cette situation fut d'obtenir les catégories d'impacts ordonnées par ID de catégorie d'impact, puis obtenir les processus ordonnés en fonction du ID de processus et du ID de catégorie d'impact. De cette manière, il était possible de connaître le nombre total de colonnes d'avance, puis de déterminer s'il manque une catégorie d'impact pour un processus et d'y insérer la valeur 0.

Le temps d'obtention des données fut grandement amélioré. Voici les nouveaux temps de génération d'un fichier:

- Obtention des données avec SQL: 2sec;
- Transformation du gabarit sur les données avec Apache Xalan: 2min 55sec;
- Téléchargement du fichier localement: 5sec;
- Total: 3min 2sec.

L'approche comportant des requêtes SQL au lieu d'Hibernate a donc été utilisée pour le reste du PFE pour le téléchargement de fichier Excel XML.

Il fut aussi nécessaire de réduire la taille des nombres retournés. En effet, Microsoft Excel est capable de supporter des nombres avec au plus 19 chiffres après la virgule. Certains des nombres retournés par le service de calcul sont toutefois plus longs. Par conséquent, un arrondissement au 19e chiffre après la virgule est fait. Une amélioration possible pour le futur serait de représenter ces nombres à l'aide de la notation scientifique que Microsoft Excel supporte (par exemple, 1.42E-22 pour représenter 1.42×10^{-22}).

Finalement, certaines modifications triviales ont été effectuées sur le gabarit. Notamment:

- L'ajout de style de nombre scientifique;
- L'ajout de style d'entête de feuille de calcul;
- Le réglage de la largeur de colonnes;
- L'ajout de tri automatique sur les colonnes.

Jean-Pierre

Lors de cette itération, la tâche à effectuer était d'ajouter les entêtes aux fichiers texte existants. Les fichiers sont des fichiers *.java, .js* et *.html*. Ces trois types de fichiers doivent avoir des entêtes encodés différemment, car les différents langages n'ont pas la même façon de déclarer des commentaires dans le code.

L'entête final sera une version modifiée de l'exemple fourni avec l'outil *header-generator* trouvé lors de l'itération précédente. L'entête contient entre autres le nom du projet, le nom de l'auteur, son courriel, la date de création, le détenteur des droits ainsi que la déclaration de la licence Apache 2.0.

Pour appliquer l'entête, l'outil *header-generator* donne des résultats inconsistants et retourne parfois des messages d'erreurs même lorsqu'utilisé convenablement. Après plusieurs tests, Jean-Pierre à déterminé que finalement l'outil n'est pas tout à fait au point et donc il faudra trouver une autre solution. Finalement, la tâche peut être effectuée simplement avec une commande itérative en *bash* pour concaténer le fichier d'entête correspondant avec l'extension de chaque fichier.

La commande suivante permet d'accomplir la tâche à effectuer :

```
for i in $(find . -name '*.html'); do cat htmlheader $i > tmp; mv tmp $i; done
```

La commande est une boucle *for* qui sera effectuée pour chaque fichier terminant par l'extension ".html" trouvée récursivement dans le répertoire où cette commande est lancée. La commande utilise la commande "cat" (pour conCATenate, concaténer en anglais) pour joindre l'entête au fichier trouvé dans un fichier temporaire, qui est ensuite renommé pour écraser l'ancien fichier sans entête.

Cette commande sera aussi lancée par la suite pour les fichiers *JavaScript* (.js) et Java (.java). Il ne suffit que de changer le « **.html* » pour une autre extension et de changer le fichier d'entête. La tâche d'insérer les entêtes sur les fichiers existants est alors terminée dans cette itération. Il ne reste qu'à trouver une façon pour que l'IDE ajoute l'entête au fichier à sa création pour les classes qui seront créées plus tard.

Aravinthan

Durant cette semaine, Aravinthan avait comme mandat de faire des mock-ups des pages. Il y avait trois pages qu'il devait travailler sur. La première est la page avec la vue 3D du bâtiment. La deuxième page est celle avec les tableaux des détails. Finalement, la troisième vue est la page avec les graphiques pour voir l'impact des catégories sur bâtiment. Pour commencer, il a commencé à regarder les différents outils qui permettent de créer des mock-ups. Finalement, l'application web mockflow.com a été choisie pour la création mockup. L'application offre plusieurs fonctionnalités, comme utiliser le style Bootstrap et également de pouvoir drag and drop les éléments directement sur la page. Il était très simple et intuitif à utiliser. Il offre également la chance de pouvoir exporter ces pages en image, PDF ou même en format HTML. Aravinthan a fait le « mock-up » de trois pages. La première est la page avec la vue 3d du bâtiment, il y a également la page détail qui comporte les données en forme de tableau et finalement l'impact par catégorie.

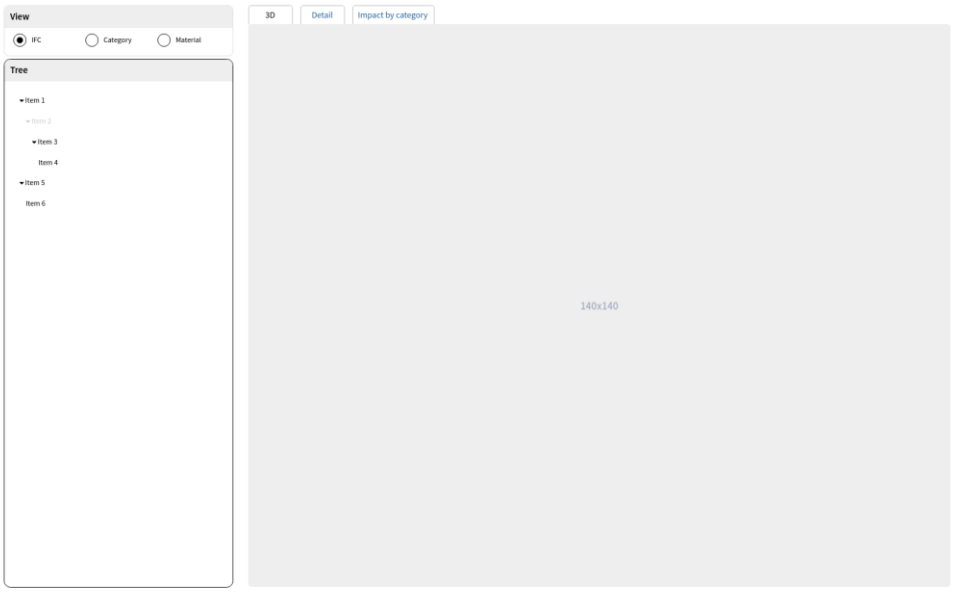


Figure 6.4.1 Page avec vue du bâtiment en 3D

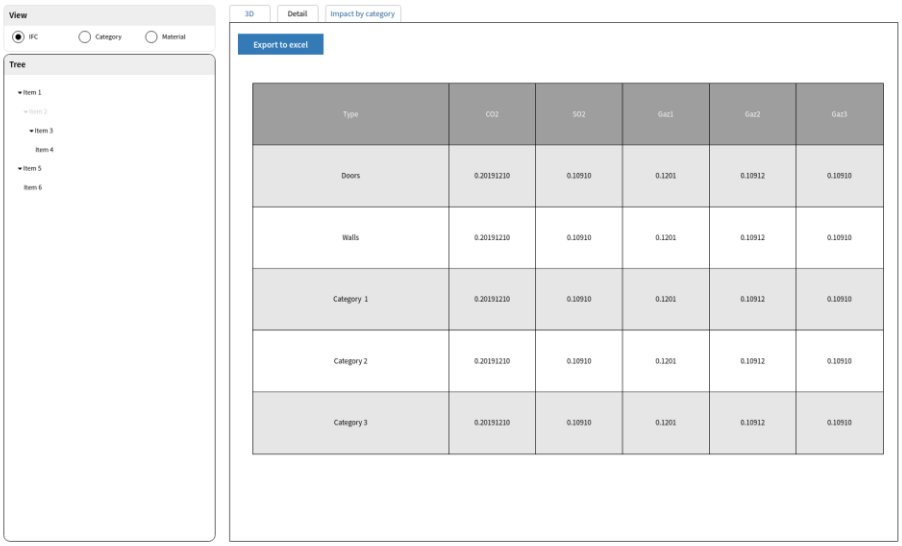


Figure 6.4.2 Page avec les détails en format tableau

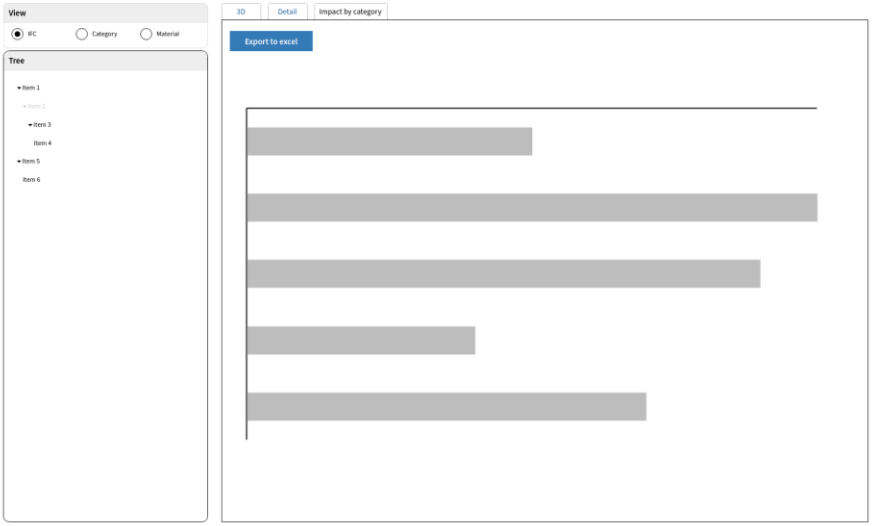


Figure 6.4.3 Graphique de l'impact des catégories

Aravinthan a cependant exporté ces pages en format image pour la simple raison de pouvoir l'utiliser dans une autre application qui permet de collaborer sur ces mock-ups. Il a utilisé l'application web « InvisionApp ». L'application permet d'ajouter un peu de dynamique au site web et simuler des liens. Ainsi, il est possible maintenant de naviguer à travers les différentes pages des mock-ups. De plus, l'application permet également d'inviter des personnes à collaborer sur ces « mock-ups ». À l'aide d'un lien, les autres membres de

l’équipe peuvent aller sur les différents pages des « mock-ups » et écrire leurs commentaires sur un élément en particulier ou bien toute la page. Ainsi, il est plus facile d’itérer à l’aide des commentaires.

CHAPITRE 7: 27 juin au 4 juillet

Daniel

Cette semaine, Daniel a modifié le code de l'insertion des données afin d'obtenir un meilleur temps. De plus, il a connecté le bouton de l'interface afin de faire le calcul. Il en a également profité pour faire un peu de réusinage dans le code et ajouter des attributs dans les modèles.

Pendant deux semaines, j'ai attendu la confirmation du promoteur pour qu'il me dise vers quelle clé primaire la colonne « *OutputId* » se réfère. Cette colonne était un attribut du modèle de *SLCIA* et une colonne dans la table « *tbl_SLCIA* ». Afin qu'il n'y ait pas d'erreur, il fallait que j'attende la confirmation de l'association de cette colonne.

Dans ce mêmes modèle et table, le promoteur m'a demandé d'ajouter des attributs qui représentent des données de type quantiles. De la façon que le code est fait, il m'a seulement fallu ajouter les attributs dans le modèle *SLCIA* et de supprimer la table « *tbl_SLCIA* ». . En exécutant l'application, Hibernate s'occupe de créer la table de nouveau avec les attributs qui représentent les champs.

En parcourant les nouveaux modèles qui ont été créés il y a quelques semaines, j'ai modifié le nom de certains attributs. Le problème provenait qu'une fois dans la base de donnée, puisque le nom de l'attribut était « processId », la colonne créée par Hibernate s'appelait *processId_id*. On était rendu avec des noms de colonnes qui possédaient plusieurs fois le mot *id*.

La modification des noms des variables fut d'enlever le mot *id*. Par exemple, *processId* est rendu « *process* ». En même temps, puisque ces modèles utilisent Hibernate, cela avait plus de sens d'enlever ce mot, car la variable contient l'objet *Process* et non juste son identifiant.

L'insertion devenait un problème de performance. Il a fallu que je trouve une nouvelle solution pour accélérer le processus d'insertion des données. En naviguant dans internet, j'ai découvert un problème avec les *PrepareStatement*. De la façon que je les utilisais, cela causait un problème de ralentissement. En effet, à chaque insertion je crée un nouveau *PrepareStatement* pour définir les nouvelles données. Le problème ici est qu'une nouvelle instance est créée à chaque fois et ça devenait lourd à la longue.

Un réusinage du code m'a permis d'obtenir un meilleur résultat. Ce réusinage constitue de ne pas créer à chaque fois un nouveau « *PrepareStatement* », mais d'utiliser le même à chaque fois. La capture d'écran suivante présente ce processus.

```
preparedStatementLCI.setDouble(1, Double.parseDouble(lineWithDatas[4]));
preparedStatementLCI.setDouble(2, Double.parseDouble(lineWithDatas[5]));
preparedStatementLCI.setDouble(3, Double.parseDouble(lineWithDatas[1]));
preparedStatementLCI.setInt(4, Integer.parseInt(lineWithDatas[0]));
preparedStatementLCI.setInt(5, Integer.parseInt(lineWithDatas[3]));
preparedStatementLCI.setLong(6, lciResultSetId);
preparedStatementLCI.setInt(7, Integer.parseInt(lineWithDatas[2]));

preparedStatementLCI.executeUpdate();
```

Dans cet exemple, une instance de « *prepareStatementLCI* » est créée et initialisée dans le constructeur. Ceci fait en sorte d'utiliser tout le temps la même instance pour un type de modèle. Je me suis permis de faire ceci, car l'initialisation de ces variables était toujours pareille en fonction du modèle. L'image suivante présente l'initialisation de la variable pour le modèle *LCI*.

```
private void initialisePrepareStatementLCI() {

    try {
        preparedStatementLCI = connection.prepareStatement("INSERT INTO tbl_LCI"
                + "(processContribution, processScalar, totalQuantity, flow_id, process_id, resultSet_id, unit_id) VALUES"
                + "(?,?,?,?,?,?,?)");
    } catch(SQLException ex) {
        System.out.println("Error in initialising prepare statement for LCI: " + ex.getMessage());
        ex.getStackTrace();
    }
}
```

En prenant la connexion établie avec la base de données, j'ai créé une requête d'avance en fonction du modèle. Ce sont toutes des requêtes d'insertions qui permettent de donner des valeurs différentes aux champs des colonnes pour chaque insertion.

Ce changement m'a permis de passer du temps de sept minutes à trois minutes. Ce fut une grande amélioration. Ceci est à cause que « *PrepareStatement* » possède une cache. À chaque itération d'insertion, il utilise sa cache pour garder sa connexion ainsi que la déclaration de requête à l'intérieur. Le temps a principalement été gagné dans la création et l'initialisation des variables.

Pendant l'insertion des données, il fallait que je spécifie l'identifiant du « *ResultSet* » afin de garder une traçabilité du temps de chaque processus. Il a donc fallu que je crée avant toutes les insertions de données un champ dans la table « *LCIResultSet* ». En procédant à une requête traditionnelle, je n'avais pas de moyen de recevoir l'identifiant du nouveau champ créé, puisque c'est une colonne qui s'incrémente toute seule. Il a fallu que j'ajoute l'option « *RETURN_GENERATED_KEYS* » dans la requête de création. L'image suivante présente cette requête.

```
PreparedStatement preparedStatement = database.getConnection().prepareStatement( sql: "INSERT" +
        " INTO tbl_LCIResultSet (idBimModel, calc_time, description) VALUES (?, ?, ?)", Statement.RETURN_GENERATED_KEYS);
preparedStatement.setLong( parameterIndex: 1, this.bimId);
preparedStatement.setTimestamp( parameterIndex: 2, Timestamp.from(Instant.now()));
preparedStatement.setString( parameterIndex: 3, this.description);

return preparedStatement;
```

Cette option m'a permis ensuite d'aller chercher la ligne qui a été modifiée. Dans mon cas, c'était l'insertion de la nouvelle donnée. Grâce à cela, j'ai facilement pu accéder au nouvel identifiant créé.

Il fallait une dernière connexion entre la vue et le code arrière pour faire fonctionner tout le flux. Dans la classe qui sert de contrôleur entre la vue et le code arrière, j'ai ajouté un appel à la méthode qui s'occupe de faire l'insertion de données. L'image suivante présente cet appel.

```
public CompletionStage<Result> calculateResult(Long idModel, Long idImpactMethod, String description) {

    return spark.Calculate(idModel, idImpactMethod, description);
}
```

C'est dans la méthode « *Calculate* » de la classe *Spark* qu'on retrouve toute la logique derrière cette implémentation. L'objet *Spark* dans cette image est injecté lors du lancement du projet.

Cydrick

Durant cette semaine, plusieurs améliorations au contenu du fichier ont été apportées par Cydrick. La plus grosse consiste à afficher les catégories complètes dans le fichier Excel. Une catégorie est par exemple « *0116: Growing of fibre crops* ». Ces catégories sont toutefois récursives. Par exemple, voici une catégorie complète, incluant la catégorie racine: « *A: Agriculture, forestry and fishing → 01:Crop and animal production, hunting and related service activities → 011:Growing of non-perennial crops → 0116:Growing of fibre crops* ». Les catégories peuvent se trouver dans des flux ou dans des processus. De plus, les catégories complètes sont présentées de la catégorie la plus générale vers la plus spécifique, avec des barres obliques les séparant.

Un des besoins était aussi que dans certains cas, il ne faudrait pas inclure la catégorie racine. Par exemple, dans la catégorie récursive précédemment illustrée, la catégorie racine serait « *A: Agriculture, forestry and fishing* ». Un autre besoin était de ne pas avoir de barre oblique avant la première catégorie dans le fichier Excel puisque cette barre oblique complexifie la tâche de sélection de flux dans une colonne.

Il existe déjà un contrôleur nommé « *Category.java* » qui permet d'obtenir cette information de manière récursive sous forme textuelle. L'action en question propose de fournir un identifiant de catégorie en format *Long* et retourne un « *Result* » (objet qui permet de retourner un résultat à un client) contenant la catégorie complète en texte. Cette méthode inclut toutefois une barre oblique avant la catégorie racine ainsi que la catégorie racine elle-même.

Par conséquent, Cydrick a décidé de faire de la réingénierie dans le contrôleur *Category.java*. Une méthode « *getFullCategory* » a été créée et permet d'obtenir la catégorie complète en *String* à partir d'un ID de catégorie ou d'une catégorie. Cette méthode permet de fournir en paramètre l'option d'afficher ou non la première barre oblique et d'afficher ou non la catégorie racine:

```
@Transactional
public String getFullCategory(Long id, Boolean omitRootCategory,  Boolean omitFirstSlash){
    org.openlca.core.model.Category category = JPA.em().find(org.openlca.core.model.Category.class, id);
    if (category == null)
        return null;
    return getFullCategory(category, omitRootCategory, omitFirstSlash);
}

@Transactional
public String getFullCategory(org.openlca.core.model.Category category, Boolean omitRootCategory, Boolean omitFirstSlash){
    return getFullCategoryAsString(category,  currentStringRepresentation: "", omitRootCategory, omitFirstSlash);
}
```

Figure 7.2.1: Code des deux nouvelles méthodes de catégorie complète

L'ancienne méthode d'obtention de la catégorie complète qui retournait un « *Result* » a été changée pour utiliser la méthode « *getFullCategory* », en spécifiant qu'il était nécessaire d'ajouter la première barre oblique ainsi que la catégorie racine.

Ce contrôleur est ensuite injecté dans « *Spark.java* », puis retransmis à « *CalculationApiXmlData* » pour que cette classe soit en mesure de générer la catégorie complète, et ainsi générer les « *Element* » XML nécessaires.

Une autre modification qui a nécessité un peu de recherche fut la gestion des zéros dans la feuille de calcul de « LCI by category » (précédemment nommée LCIA). En effet, le fait que cette feuille représente une matrice creuse (ou « *Sparse Matrix* ») génère beaucoup de valeurs 0 qui ne sont pas pertinentes à afficher et ne font que remplir inutilement l'espace. Microsoft Excel possède un format d'affichage de valeurs assez avancé qui permet de configurer comment afficher une valeur positive, négative, nulle et vide. Il fut donc possible de configurer un format pour cette feuille de calcul qui n'affiche tout simplement pas les zéros, mais qui affiche les valeurs positives et négatives dans le format scientifique. Le format utilisé fut :

<NumberFormat ss:Format="0.00E+00;\-0.00E+00;;@"/>

De plus, voici une brève liste de modifications supplémentaires qui ont été appliquées sur le gabarit:

- Renommer des colonnes et des feuilles de calcul pour utiliser l'anglais;
- Renommer le type d'entrée affichée de SLCIA pour utiliser l'anglais;
- Ajouter une instruction qui permet à Microsoft Excel d'ouvrir le fichier .xml en double-cliquant dessus si le sélecteur d'application Microsoft Office est bien paramétré.

Jean-Pierre

Pour cette itération, Jean-Pierre a implémenté les fichiers d'entête dans l'IDE Intellij-idea. Il a écrit la procédure pour ajouter des *gabarits* au IDE pour appliquer l'entête à la création de fichiers. Ces entêtes déclarent la licence Apache 2.0 qui est lié aux fichiers, ce qui signifie que le promoteur gardera les droits sur le code, en plus que le code sera considéré entièrement comme logiciel libre. La procédure pour ajouter les *gabarits* au IDE est la suivante :

1. Fermer l'IDE intellij s'il est ouvert.
2. Acquérir les gabarits, soit via la branche sur gitlab ou via le fichier templates.zip
3. Déposer les gabarits dans le dossier .idea/fileTemplates/internal/ du projet.
4. Démarrer intellij, et ensuite faire CTRL+ALT+S dans le projet.
5. Naviguer à Editor > File And Code Templates
6. En haut à droite, changer "Schema" de "Default" à "Project". Si tout s'est bien passé, certains gabarits dans la liste seront en bleu.
7. Les gabarits sont ajoutés avec succès. Maintenant, lorsque vous créerez des nouvelles classes, sélectionnez le gabarit dans la liste et ils seront automatiquement ajoutés.

Il est important de comprendre le fonctionnement des *gabarits* dans Intellij-idea. Les *gabarits* "Default" sont les *gabarits* utilisés pour chaque nouvelle classe créée dans l'IDE, peu importe à quels projets elles appartiennent. Alors que les *gabarits* "Project" sont propres au projet dans laquelle ils sont joints. Les futurs usagers voudront ajouter les *gabarits* au "Project" pour éviter que l'entête de projet soit appliqué à tout leur code. Il est important de mentionner que si un usager décide de supprimer son dossier .idea, il faudra refaire la procédure.

Le fichier .zip contenant les *gabarits* a été déposé sur *gitlab* et a été remis au promoteur pour les prochaines équipes les utilisent.

De plus, certaines modifications ont été apportées aux entêtes depuis la semaine dernière. Les entêtes des fichiers lisibles sur l'ordinateur client (les fichiers *JavaScript* et *HTML*) ne contiennent plus l'adresse courriel du promoteur par mesure de sécurité.

Pour finir, le travail de l'itération précédente contenait une erreur qui fut corrigée lors de cette itération: l'automatisation de l'addition des entêtes au début des fichiers existant à causer un problème aux fichiers terminant par l'extension ".scala.html".

```
<!--
/**--------------------------
 * PROJECT: UBUBI
 * Auth:
 *   Mathieu Dupuis
 * Date: ETE 2017
 *
 * Copyright [2017] [Mathieu Dupuis]
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 *--------------------------*/
-->
@main("Calculation") {
```

Figure 7.3.1 Entête mal placée causant l'erreur

Ces fichiers doivent commencer par une déclaration de fonction et le contenu HTML doit être placé dans une des fonctions du fichier. L'opération précédente ne faisait que concaténer des textes ensemble, un après l'autre, donc l'entête s'est retrouvé mal placé dans ces fichiers.

Bref, une fois cette erreur corrigée, les entêtes sont une tâche terminée pour le reste du projet.

Aravinthan

Cette semaine, Aravinthan a commencé à coder l'arbre de navigation qui permet de voir l'état des « process », « flows » et « exchanges » avant de faire des calculs. Pour commencer, il devait comprendre comment le code Angular fonctionnait. En fouillant dans le code, il s'est rendu compte que l'arbre est rafraîchi à plusieurs places et que ça va être beaucoup plus long que prévu. En effet, il faut qu'Aravinthan regarde tous les appels API qui sont faits et s'assurer que ces appels sont également répétés pour aller chercher les tables history. Il a donc commencé par identifier tous les endroits à modifier. Une fois cette étape complété, il a

ensuite ajouté un nouveau endpoint qui sera utilisé pour le tableau process_history.

Par la suite, il a commencé à explorer les différentes façons d'aller chercher les données qui sont dans le tableau. Il a donc commencé par créer un modèle ProcessHistory. En faisant un modèle, il pourra utiliser les méthodes Hibernate pour pouvoir chercher une donnée facilement.

```java
package models.LCA;

import ...

@Entity
@Table(name = "tbl_processes_history")
public class ProcessHistory extends CategorizedEntity {

    @Column(name = "default_allocation_method")
    @Enumerated(EnumType.STRING)
    private AllocationMethod defaultAllocationMethod;

    @OneToMany(cascade = { CascadeType.ALL }, orphanRemoval = true)
    @JoinColumn(name = "f_process")
    private final List<AllocationFactor> allocationFactors = new ArrayList<>();

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "f_owner")
    private List<Exchange> exchanges = new ArrayList<>();

    @OneToOne
    @JoinColumn(name = "f_location")
    private Location location;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "f_owner")
    private final List<Parameter> parameters = new ArrayList<>();

    @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "f_process_doc")
    private ProcessDocumentation documentation;

    @Column(name = "process_type")
    @Enumerated(EnumType.STRING)
    private ProcessType processType = ProcessType.UNIT_PROCESS;

    @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "f_quantitative_reference")
    private Exchange quantitativeReference;

    @Column(name = "infrastructure_process")
    private boolean infrastructureProcess;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
```

Finalement, il a commencé à écrire la méthode qui permet de trouver la bonne donnée. Cependant, il n'avait pas eu beaucoup de temps pour le terminer. Il s'est donc laissé les étapes à prendre en commentaires pour pouvoir le finir dans les prochaines semaines.

CHAPITRE 8: 4 juillet au 11 juillet

Daniel

Pour cette semaine, Daniel a finalisé l'importation des données en modifiant pour une dernière fois l'algorithme. Il a aussi inséré un cercle de chargement dans la vue pour indiquer que l'insertion est en cours. Il en a également profité pour faire un peu de réusinage dans le code.

La semaine dernière j'attendais la confirmation du promoteur pour qu'il me dise l'association du champ *OutputId*. Cette semaine, je l'ai reçu. Il représente le modèle *ImpactMethod*. En ayant cette information, j'ai pu procéder à l'insertion.

Malgré l'amélioration du temps de la dernière semaine, ce n'était pas suffisamment rapide pour le promoteur. J'ai donc cherché d'autres façons de faire l'insertion des données. J'ai réussi à trouver une méthode efficace pour l'insertion, mais qui comporte des risques. Puisque dans notre cas on veut prioriser la rapidité, cette nouvelle méthode remplit parfaitement les critères.

Celle-ci est « *LOAD DATA LOCAL INFILE* ». Cette méthode m'a permis de prendre un fichier texte pour l'insérer directement dans la base de données. Puisque les fichiers se trouvaient dans un serveur externe, il a fallu que je crée un algorithme pour les importer localement. L'image suivante présente la façon que j'ai utilisée.

```
private void importFilesFromUrlLocation() throws IOException {

    for (Map.Entry<ModelLCAType, String> entry : modelLCATypeWithURLLocation.entrySet()) {

        URL website = new URL(entry.getValue());
        ReadableByteChannel rbc = Channels.newChannel(website.openStream());
        FileOutputStream fos = new FileOutputStream(modelLCATypeWithNewFileName.get(entry.getKey()));
        fos.getChannel().transferFrom(rbc,  position: 0, Long.MAX_VALUE);
        fos.close();
    }
}
```

En allant chercher le fichier externe, je crée un tunnel de bite pour transférer le fichier bite par bite à l'intérieur. Une fois que le transfert est terminé, il est important de fermer ce tunnel, sinon il est impossible de le supprimer par la suite. Également, si le tunnel n'est pas fermé, il peut y avoir des problèmes de performance par la suite dans l'exécution, car la ressource ne se libère pas.

Une fois les fichiers sont importés localement, je les insère dans la base de données via la méthode « *LOAD DATA LOCAL INFILE* ». Un problème qui peut survenir lors de l'insertion est dans le cas où une colonne qui prend une clé secondaire reçoit une donnée qui ne réfère pas à une colonne de clé primaire. L'application va continuer l'insertion, mais va prendre ce problème comme un avertissement seulement. Il faut donc régler le problème manuellement à la fin de l'insertion.

Dans l'initialisation de la requête, il est important d'identifier la façon dont les données sont séparées dans une ligne et comment les sauts de lignes sont effectués. L'image suivante présente la façon que j'ai procédé pour faire l'insertion via cette méthode.

```
loadQuery = "LOAD DATA LOCAL INFILE '" + modelLCATypeWithNewFileName.get(ModelLCAType.LCI) +
        "' INTO TABLE tbl_LCI FIELDS TERMINATED BY ','" + " LINES TERMINATED BY '\\n' " +
        "(@flow_id, @totalQuantity, @unit_id, @process_id, @processContribution, " +
        "@processScalar, @resultSet_id) set " +
        "flow_id=@flow_id, totalQuantity=@totalQuantity, unit_id=@unit_id, process_id=@process_id, " +
        "processContribution=@processContribution, processScalar=@processScalar, resultSet_id="+resultSetId+";";
statement.execute(loadQuery);
```

Cet exemple montre seulement pour la table *tbl_LCI*. Les trois autres modèles respectent la même procédure d'insertion. L'avantage également de ce code, c'est qu'il utilise toujours la même connexion. Ceci fait en sorte qu'une nouvelle connexion n'a pas besoin d'être effectuée pour chaque insertion.

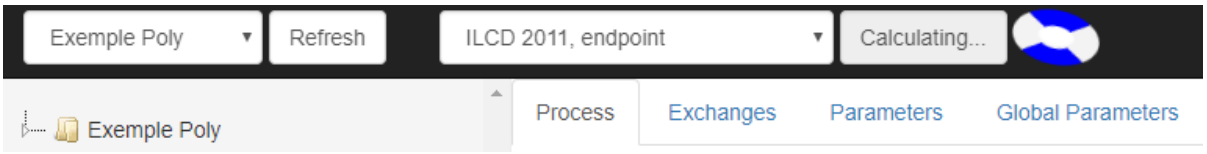
Une fois que toutes les données sont insérées dans la base de données, je procède à la suppression des fichiers importés localement. C'est une étape importante, car on ne veut pas garder ces fichiers dans le projet. Dans le cas où plusieurs insertions vont être effectuées, il va y avoir trop de fichiers. Un simple algorithme m'a permis de faire la suppression de ces fichiers. La capture d'écran suivante le démontre.

```
public void removeImportedFilesFromUrlLocation() {

    for (Map.Entry<ModelLCAType, String> entry : modelLCATypeWithNewFileName.entrySet()) {
        File fileToDelete = new File(entry.getValue());
        fileToDelete.delete();
    }
}
```

Dans la classe principale pour faire le calcul, il y a un *try/catch*. Dans le *try*, c'est là que je fais l'appel à mon algorithme d'insertion. Dans le cas où il y a une erreur, je supprime dans le *catch* les fichiers qui ont été importés localement. Ceci fait en sorte de ne pas se retrouver avec des fichiers inutilement.

L'utilisation du « *LOAD DATA LOCAL INFILE* » m'a permis de passer de trois minutes de la semaine dernière à quinze secondes. Cette amélioration est due au fait que cette nouvelle méthode faire seulement une vérification MySQL avant de faire l'insertion. Dans le cas des *PrepareStatement*, MySQL faisant une vérification à chaque insertion pour s'assurer que les requêtes respectaient la syntaxe SQL.

Pendant l'insertion des données, il fallait que j'ajoute un cercle de chargement animé dans la vue pour indiquer à l'utilisateur que les données se font insérer. Lorsque celui-ci procède au calcul des données pour le modèle choisi, le nom du bouton change et le cercle apparaît. La capture d'écran suivante présente cet évènement.



Il est à noter que le nom du bouton à changer pour « *Calculating… »* et il n'est plus accessible lorsqu'il y a un calcul en cours. Ceci fait en sorte que l'utilisateur comprenne que les données se font insérer. Lorsque les données sont toutes entrées dans la base de données, le bouton reprend sa forme originale avec texte et le cercle de chargement disparaît.

C'est dans la partie du JavaScript que j'ai pu implémenter cette fonctionnalité. En appuyant sur le bouton « *Calculate* », j'utilise du JQuery pour faire les changements dans l'interface. Une fois que je reçois la réponse du service web que l'insertion est terminée, je remets l'interface comme elle était avant. L'image suivante présente le code qui m'a permis d'agir ainsi.

```
var calculateResult = function() {
    showLoadingCircle();
    let modelId = Number.parseInt(modelSelect.val());
    let impactMethodId = Number.parseInt(methodSelect.val());
    let description = $('#methodDescription').val();

    getAjaxFromRouter(
        jsModel.controllers.API.model.BIMModel.calculateResult(modelId, impactMethodId, description),""
    ).done(function(response){
        hideLoadingCircle();
        console.log(response);
    }).fail(function(jqXHR, textStatus, errorThrown) {
        hideLoadingCircle();
        console.log("failed!");
        console.log(errorThrown);
    });
};
$("#btnCalculateResult").click(calculateResult);
```

Cydrick

La prochaine modification effectuée par Cydrick fut de s'attarder aux performances du téléchargement et des améliorations au gabarit du fichier Excel XML.

Dans l'approche qui est utilisée jusqu'à ce moment, les données et le gabarit sont fusionnés ensemble par Apache Xalan pour générer un fichier. Ce fichier est sauvegardé temporairement sur le disque, puis le Play Framework retourne ce fichier temporaire à l'utilisateur qui en a fait la demande.

Ceci cause des problèmes. Tout d'abord, cette étape serait plus rapide si le contenu du fichier pouvait rester dans la mémoire de l'application Java, puis être envoyé directement à l'utilisateur sans être sauvegardé sur le disque. De plus, la suppression des anciens fichiers temporaires déjà téléchargés peut s'avérer complexe. En effet, si aucun fichier temporaire n'est supprimé, ceci peut causer des problèmes d'espace disque sur le serveur. Si le fichier est supprimé trop rapidement (par exemple, durant son téléchargement), ceci peut causer des erreurs de téléchargement.

La solution évidente est de garder le contenu à envoyer au client dans une variable de type « *String* ». Cette variable contiendrait l'ensemble du fichier à envoyer au client. Lorsque la transformation serait terminée, cette variable serait retournée au client. Il est possible de changer les entêtes HTML retournées à l'utilisateur pour que le navigateur interprète le texte comme un fichier, et même lui mentionner le nom de fichier. Ceci aurait pour avantage d'éviter de faire cette gestion de fichiers, tout en accélérant légèrement le temps de traitement puisque le fichier n'aurait pas à être écrit sur le disque par Apache Xalan, puis lu par le Play Framework. Toutefois, le fichier est trop volumineux et il n'est pas possible de le garder entièrement dans un « *String* ». En effet, Apache Xalan lance un « *BufferOverflowException* » lorsqu'on paramètre la librairie pour sauvegarder le contenue du fichier dans une variable de type « *String* ».

Après quelques heures de recherche, Cydrick a remarqué qu'il était possible avec Play Framework d'envoyer du contenu à l'utilisateur sous forme de flux (Stream) au navigateur à l'aide d'un flux de lecture, ou « *InputStream* » Java. Il a aussi remarqué qu'il était possible que le résultat de la transformation de Apache Xalan soit retourné dans un flux d'écriture, ou « *OutputStream* » Java. Puisque ces deux types de flux sont à la base incompatible, il fut nécessaire de faire de la recherche supplémentaire.

La solution fut d'employer une paire de « *PipedInputStream* » et de « *PipedOutputStream* ». La particularité de cette paire de flux est que le « *PipedOutputStream* » écrit dans un tampon (un tableau d'octet dans ce cas), puis le « *PipedInputStream* » lit le même tampon. Donc, une instance d'un « *PipedOutputStream* » est fournie à Apache Xalan. Une instance de « *PipedInputStream* » est aussi créée, et paramétrée pour qu'elle pointe sur le même tampon que le « *PipedOutputStream* ». Cette instance de « *PipedInputStream* » est ensuite fournie au Play Framework. Le résultat est que Apache Xalan est capable de convertir des données et en

même temps, puis le Play Framework les lit et les envoie au client. Il est toutefois nécessaire de mettre le « *PipedOutputStream* » et le « *PipedInputStream* » sur deux fils d'exécution (« *Threads* ») différents puisque sinon, ils vont se bloquer mutuellement pour les opérations de lecture et d'écriture sur le tampon.

Voici la structure des flux et le passage d'informations de manière graphique:

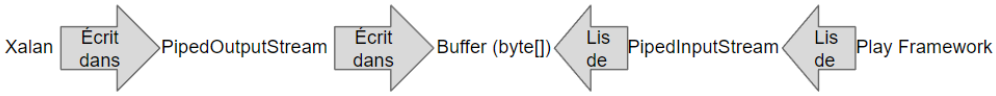


Figure 8.2.1: Structure de flux avec Apache Xalan

Cette solution élimine la gestion de fichiers et le temps de traitement est plus rapide (puisqu'il n'est pas nécessaire d'écrire le fichier sur disque, puis de le relire). Cette approche aussi a pour avantage de démarrer le téléchargement beaucoup plus rapidement puisqu'il n'est plus nécessaire d'attendre la fin de la transformation d'Apache Xalan pour débuter le téléchargement.

Par contre, Cydrick a fait face à différents problèmes avec la paire de « *PipedOutputStream* » et de « *PipedInputStream* ». Il est obligatoire de vider le « *PipedOutputStream* » ainsi que le « *PipedInputStream* » à l'aide de la méthode « *.flush()* ». Si ceci n'est pas fait, le navigateur lève une erreur puisque les derniers octets du fichier ne lui sont pas envoyés. Il est aussi important de ne pas fermer le « *PipedInputStream* » et de laisser le Play Framework le fermer car Play Framework ne sera pas en mesure de lire les derniers octets du fichier, et le navigateur va lever la même erreur qu'avant.

Avec cette approche, voici les temps nécessaires pour terminer l'opération:

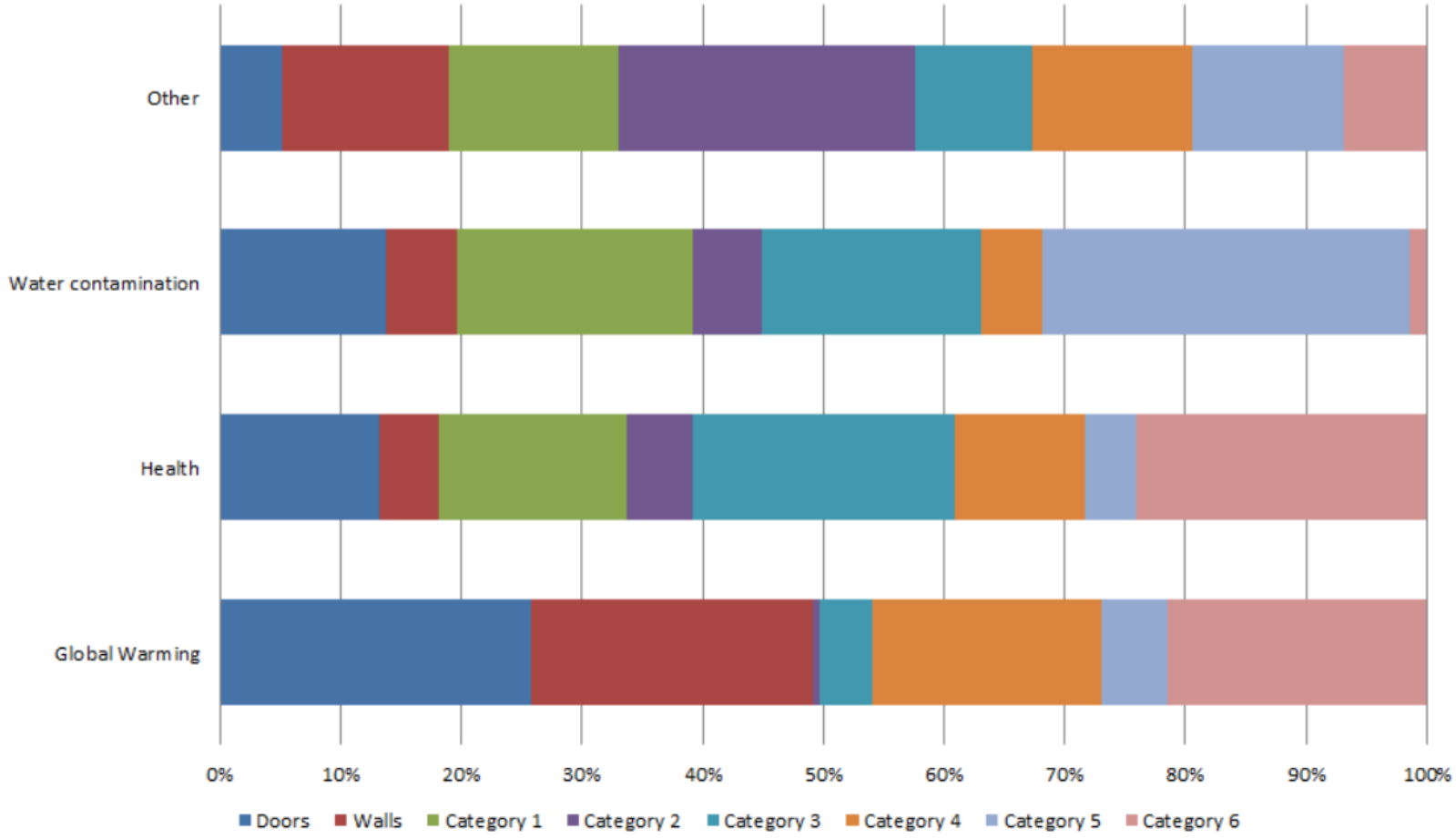
- Obtention des données avec SQL: 2sec
- Transformation du gabarit sur les données avec Apache Xalan: 2min 55sec
    - Téléchargement du fichier localement: Inclus dans le temps de conversion
- Total: 2min 57sec

Des modifications ont aussi été apportées au gabarit, soit:

- Ajouter une colonne de catégorie complète
- Créer une nouvelle feuille de calcul nommée « Characterized Inventory » qui contient l'inventaire groupée par flux.

Jean-Pierre

Pour cette itération, Jean-Pierre s'est concentré sur sa nouvelle tâche assignée, soit de déterminer quelle technologie utilisée pour la génération de graphique en *front-end*. Le client a déjà pré-approuvé un prototype provenant d'une version antérieure de l'application, donc la technologie choisie devra être en mesure de recréer ce genre de graphique, mais via



technologies web (HTML, Javascript et CSS) plutôt qu'une application.

Figure 8.3.1 Prototype préapprouvé par le client

Les contraintes du client sont les suivantes :

1. Le graphique doit garder cette apparence; des barres empilées à l'horizontale.
2. L'utilisateur doit être capable de cliquer sur les barres pour explorer les données.
3. Les données doivent être affichées en pourcentage.
4. La technologie doit être capable de fonctionner avec des appels *AJAX* asynchrones.
5. La technologie doit être capable de manipuler des grands ensembles de données.

Jean-Pierre a comparé trois technologies. Les technologies comparées sont D3.js, Charts.js et

AmCharts.

Contraintes	D3.js	Charts.js	AmCharts
Graphique en bar empilé à l'horizontal, en pourcentage.	Oui	Pas en pourcentage	Oui
Prend en charge l'exploration (drill-down)	Oui	Non	Oui
Affichage en pourcentage	Oui	Oui	Oui
Prend en charge les appels asynchrone *AJAX*	Oui	Oui	Oui
Capable de manipuler des grands ensemble de données	Incertain	Incertain	Oui, Selon Alain

Tableau 8.3.1 Comparaison des différentes librairies
de dessin graphique disponible selon le besoin.

D'après ses recherches, Jean-Pierre a déterminé que *D3.JS* ainsi que *AmCharts* pouvaient accomplir la tâche. *D3.JS* contient tout de même une certaine incertitude quant à la réactivité lorsque les ensembles de données sont grands, alors que *AmCharts* ne posait pas ce genre d'incertitude car il est utilisé dans des projets de recherche sur le Big Data. Au final, Jean-Pierre a décidé d'utiliser *AmCharts*, car le code est beaucoup plus simple et lisible avec *AmCharts* qu'avec *D3.js* ce qui en facilite sa maintenabilité. L'équipe a déjà travaillé avec *AmCharts* durant la première itération, le résultat final avec *AmCharts* est plus dynamique et esthétique que celui avec *D3.JS* et la librairie *AmCharts* est un peu plus légère que *D3.JS*.

Aravinthan

Aravinthan était en France du 7 juillet jusqu’au 13 pour un tournoi sportif international. Ce temps perdu fut repris lors de son retour.

CHAPITRE 9: 11 juillet au 18 juillet

Daniel

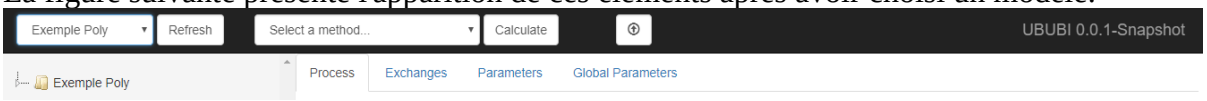
Lors de cette semaine, Daniel a ajouté des fonctionnalités ciblant la liste déroulante de méthodes et le bouton à calculer. Également, plusieurs modifications ont étées ajoutées afin de restreindre l'accessibilité à plusieurs onglets. Le tout a été terminé avec un réusinage de l'ancien code et de celui produit lors de semaine.

Lors de la dernière rencontre avec le promoteur et l'équipe, nous avons discuté de la façon dont la liste déroulante de méthodes et le bouton de calcul sont implémentés dans l'interface. Voici la façon qui a été implémentée avant la rencontre:

Nous avons donc décidé de cacher la liste déroulante et le bouton « Calculate » dans le cas où il n'y a pas de modèle de sélectionné. Cette décision a été prise puisqu'il est impossible de faire un calcul lorsqu'on n'a pas choisi de modèle. Il y avait aussi une question de compréhension de son utilisation. Ce n'était pas évident de comprendre quand on peut calculer, car l'option est toujours affichée.

La liste déroulante ainsi que le bouton « Calculate » sont cachés en utilisant du CSS. L'utilisation du CSS à permis que ces éléments prennent aucune place dans l'interface lorsqu'ils sont cachés. Voici le résultat final:

La figure suivante présente l'apparition de ces éléments après avoir choisi un modèle:



Une autre tâche qui a été accomplie par Daniel est la suppression d'un « alert » en JavaScript lorsqu'un utilisateur ne choisit l'option d'aucun modèle dans la liste déroulante. Le message indiquait que le système n'est pas capable d'aller chercher le modèle à l'index -1. Un ancien développeur l'avait laissé dans le code. J'ai eu besoin de supprimer la ligne de code contenant cet « alert »

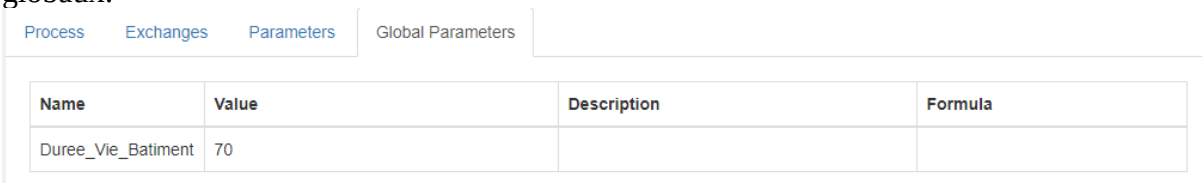
L'avant-dernière tâche accomplie par Daniel cette semaine était d'enlever l'accessibilité à certains onglets. Le problème survient lorsqu'il y a une sélection d'un modèle en particulier et l'utilisateur change la sélection pour aucun modèle. L'accès aux onglets de l'ancien modèle est encore disponible à l'utilisateur. Il faut donc enlever cet accès à la suite d'une sélection d'aucun modèle.

De façon plus technique, le JavaScript génère du HTML, ce qui permet d'obtenir le contenu de l'interface. Il m'a fallu modifier le contenu de la génération du HTML afin de vider les

champs affichés lorsqu'il n'y a aucun modèle de sélectionné. Une fois qu'un modèle est sélectionné, le contenu du HTML est donné au JavaScript afin qu'il l'affiche. La capture d'écrans suivante présente la logique pour l'accessibilité des onglets et des composantes pour le calcul.

```
if(modelId > 0) {
    showCalculateButtonAndComboBox();
    retrieveGlobalParams();
} else {
    hideCalculateButtonAndComboBox();
    hideProcessDetails();
}
```

La dernière tâche effectuée est d'afficher les paramètres globaux dans les processus. Présentement, il y a seulement un paramètre global dans le système. Cet ajout a été fait à la demande du promoteur. L'image suivante présente l'interface présentent ces paramètres globaux.



Pour cette tâche, un service web fut créé afin d'aller chercher les paramètres globaux dans la base de données et les renvoyer à la vue. Voici le code qui a permis d'aller chercher ces paramètres:

```
private List<Parameter> getGlobalParametersFromDB() {

    Query query = JPA.em().createQuery( qlString: "Select p from Parameter p where p.scope = :global");
    query.setParameter("global", ParameterScope.GLOBAL);

    return (List<Parameter>) query.getResultList();
}
```

L'utilisation de Hibernate afin de créer une requête est très efficace, rapide et simple. Le code utilisé se restreint dans seulement quatre lignes. Cette méthode a été ajoutée dans le fichier de routes afin d'y avoir accès dans la partie du JavaScript.

Il est à noter que les paramètres globaux ne prennent pas présentement en compte le modèle, car ce n'était pas prévu dans l'avancement du projet. Dans le cas où est-ce qu'il va y avoir plusieurs modèles, les paramètres globaux vont devoir s'afficher en conséquence de chacun d'entre eux.

Dans la partie frontend, il a fallu ajouter un onglet dans les processus pour afficher les paramètres globaux. L'ajout de cet onglet a été fait en ajoutant un gabarit similaire à celui des paramètres. Seul le contenu du tableau va différer.

Le service web est légèrement différent, car il n'est pas nécessaire de rafraîchir constamment

les données dans le tableau, puisque ce sont des valeurs globales. Elle ne dépendant pas d'aucune sélection de modèle ou processus comme les paramètres. L'image suivante présente la façon d'appeler la méthode présentée ci-haut (getGlobalParametersFromDB).

```
retrieveGlobalParams = function() {
    getAjaxFromRouter(jsModel.controllers.API.model.Process.getGlobalParameters(),"").done(function(paramJson) {
        let params = paramJson.lst;
        displayGlobalParams(params);
    });
};
```

L'appel Ajax au service web peut être fait, car elle a été ajoutée dans le fichier des routes. La méthode « *displayGlobalParam(params)* » boucle à travers les paramètres globaux et affiche dans le tableau HTML les valeurs.

À la fin de cette dernière tâche, Daniel a décidé de faire un peu de réusinage dans son ancien code afin d'obtenir une meilleure lisibilité à travers la classe. La figure suivante présente le nouveau code:

```
@Transactional(readOnly = true)
public Result getAllImpactMethod() {

    Query query = JPA.em().createQuery( qlString: "Select m from ImpactMethod m");

    List<ImpactMethod> impactMethods = query.getResultList();
    if (impactMethods == null || impactMethods.size() == 0)
        return jsonResult(notFound());

    return jsonResult(ok( content: "{"+JSONUtil.parseArray( key: "lst",impactMethods,JSONFormater.ImpactMethodFormater)+"}"));
}
```

L'ancien code ne prenait pas en compte du retour de la requête. Dans le cas où la liste était vide, il retournait une réponse vide et aucun message ne serait affiché. Également, avant d'utiliser JPA, j'utilisais des prepareStatement ce qui causait un manque de constance dans la classe. Il était difficile de savoir quelle méthode était utilisée dans le logiciel. En gardant cette constance, cela permet aux prochains programmeurs de comprendre plus rapidement le code et d'amener une bonne qualité au logiciel.

Cydrick

Pour la semaine 9, Cydrick a fait que des modifications au gabarit. Voici une liste des modifications effectuées sur le gabarit suite aux commentaires du promoteur:

- Changer la taille de certaines colonnes;
- Ajouter des colonnes de catégorie de processus;
- Renommer le nom de plusieurs colonnes;
- Améliorer le formatage de plusieurs cellules;
- Changer l'ordonnancement de lignes dans la feuille de calcul « Characterized Inventory »;
- Remettre la catégorie racine pour les catégories associées à un processus;
- Ajouter une feuille de calcul « LCI by Category ».

Jean-Pierre

Pour cette itération, Jean-Pierre a pris du temps pour se familiariser avec AmCharts. En regroupant plusieurs exemples sur internet fourni par AmCharts, le premier prototype d'essai démontre bien que la technologie choisie est capable de prendre en charge les demandes du client: un graphique en barres empilées à l'horizontale, capable d'approfondir dans les données. Ce graphique n'a pas encore de titre et aucune fonction *AJAX* n'y est actuellement présente pour permettre l'acquisition de données à partir d'un service web. Les données utilisées lors de cette itération sont inscrites directement dans le code JavaScript. Jean-Pierre planifie réusiner ce travail pour sauver du temps (de programmation) et prioriser la finalisation du graphique. Cependant, ce travail devra être réadapté pour répondre aux besoins du client.

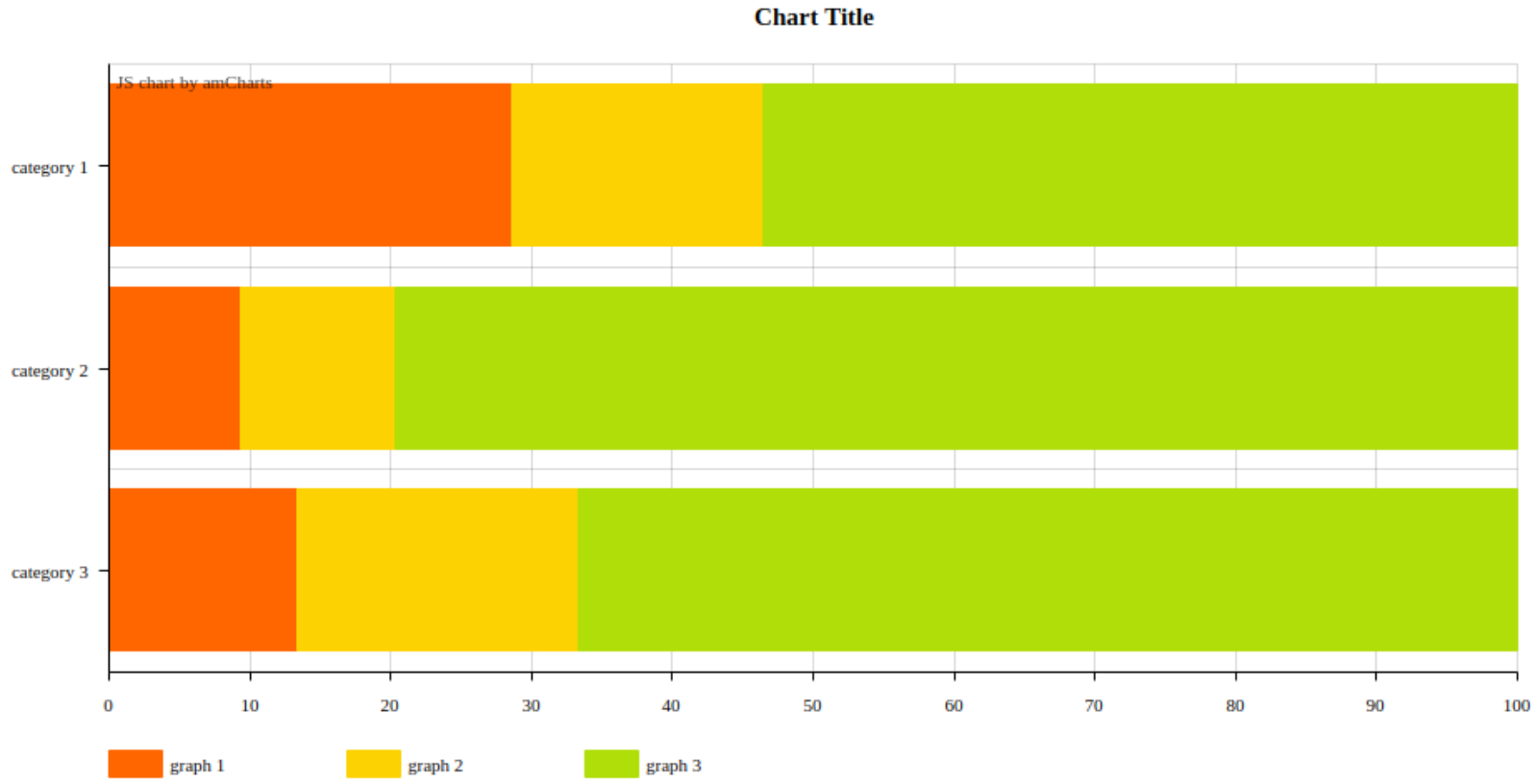


Figure 9.3.1 Premier prototype d'essai

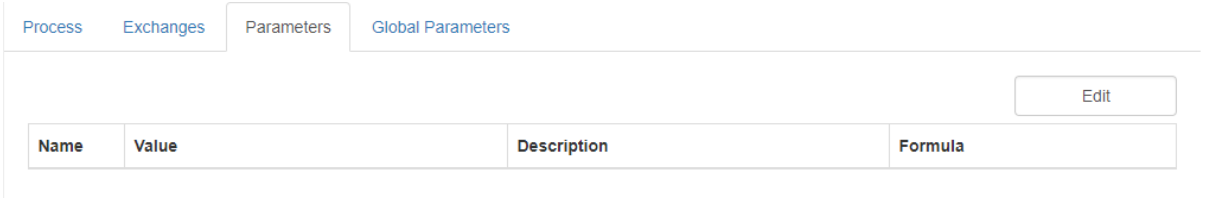
Aravinthan

Durant la deuxième semaine de son séjour en France, Aravinthan n’a pas eu la chance de pouvoir travailler sur le projet.

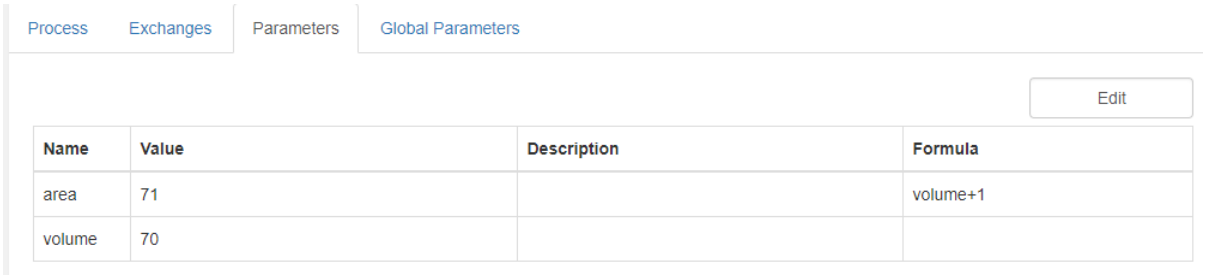
CHAPITRE 10: 18 juillet au 25 juillet

Daniel

Lors de cette semaine, Daniel a accompli la tâche d'éditer les paramètres locaux. Celle-ci comportait plusieurs fonctionnalités. La principale est de pouvoir modifier les valeurs des champs « *value* » et « *formula* ». Un des défis fut de placer dans l'interface le bouton de sauvegarde. Le choix a été fait pour ressembler à l'onglet « Process ». Il y a d'abord un bouton pour modifier les champs. Celui-ci se nomme « *Edit* ». L'image suivante présente l'interface du bouton.



Dans le cas qu'un processus possède des paramètres, ils sont affichés de la manière suivante.



Une autre fonctionnalité est d'interprété des fonction mathématique qui sont entrée dans le champ « formula ». Comme la figure précédente l'illustre, la section dans le tableau « *formula* » contient le nom des autres paramètres pour obtenir la valeur. Par exemple, le paramètre « *area* » est égal au « *volume+1* », dont celui-ci à une valeur égale à 71.

Dans le code du service web, il a fallu ajouter une méthode pour recevoir les nouvelles valeurs, les modifier, les appliquer aux paramètres cibles et les persister. Voici la méthode qui fait ceci:

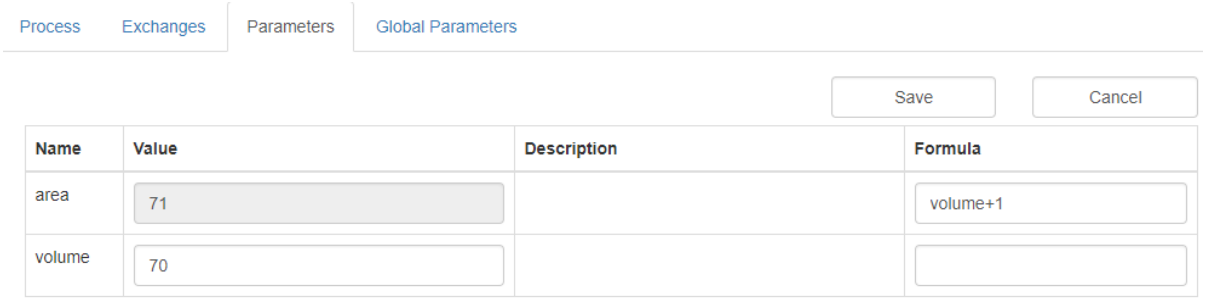
```
@Transactional
public Result setParameterFormula(Long idProcess, String newFormula, String parameterName, Double value) {

    org.openlca.core.model.Process retrievedProcess = JPA.em().find(org.openlca.core.model.Process.class,idProcess);

    String newFormulaWithValue = replaceNewFormulaWithRealValue(retrievedProcess, newFormula);
    double calculatedValue = evaluateValueWithFormula(newFormulaWithValue);

    for(Parameter parameter : retrievedProcess.getParameters()) {
        if(parameter.getName().equalsIgnoreCase(parameterName)) {
            setValue(parameter, newFormula, value, calculatedValue);
            setFormula(newFormula, parameter);
            JPA.em().persist(parameter);
            break;
        }
    }

    return jsonResult(ok( content: "{"+JSONUtil.parseArray( key: "lst",retrievedProcess.getParameters(),JSONFormater.ParameterFormater)+"}"));
}
```

Plusieurs étapes sont importantes à illustrer dans cette méthode. La première étant la gestion des formules. Il faut remplacer dans la formule les noms des paramètres par leurs valeurs. Un algorithme qui boucle à travers les paramètres pour aller chercher chaque valeur et les remplacer dans le cas qu'ils sont présents dans la formule.

La prochaine étape est l'évaluation de la formule. Une fois qu'elle ne contient que des chiffres numériques ou des fonctions mathématiques, elle peut être envoyée à l'interprète de formule. À la demande du promoteur, l'ajout des classes de *OpenLCA* pour évaluer les formules a été ajouté directement dans le logiciel. Une tentative de créer un JAR et l'ajouter dans le projet fut plus complexe que prévu, c'est pour cela que les classes ont été misent directement dans le projet.

La troisième étape est la modification de la valeur et de la formule dans le cas nécessaire pour le paramètre indiqué par l'utilisateur. L'algorithme est une boucle qui passe à travers les paramètres du processus pour trouver lequel a été sélectionné. L'utilisation de *JPA* pour sauvegarder l'entité est une façon facile et rend le code lisible. Un retour du processus est renvoyé à la vue pour pouvoir afficher les modifications.

Dans la partie frontend, une autre variable a été ajouté dans les gabarits de JavaScript contenant les champs de text pour permettre la modification des valeurs. Les colonnes du tableau « *value* » et « *formula* » possèdent des textfield afin de permettre à l'utilisateur d'entrer une nouvelle valeur. L'image suivante montre l'interface après avoir appuyé sur le bouton « *Edit* ».



Il est à noter un comportement dans la figure précédente. Lorsqu'il y a une formule, les valeurs ne peuvent pas se faire modifier par l'utilisateur. Dans le cas où il n'y a pas de formule, la valeur du paramètre en question peut être modifiée.

Ce comportement a été implémenté grâce à l'utilisation de JQuery lors de la population du tableau. Une vérification est faite pour déterminer s'il y a une formule, si c'est le cas, le champ de la valeur est désactivé. Puisque tous les champs possèdent le même attribut id dans HTML, il a fallu désactiver les champs en fonction des identifiants de chaque paramètre.

Au moment d'appuyer sur le bouton « *Edit* », deux autres options remplacent ce bouton. Le bouton « *Save* » permet de sauvegarder les modifications faites dans les champs et le bouton « *Cancel* » permet d'annuler toutes les modifications qui ont voulu être faites, mais qui n'ont

pas été enregistrées.

Le JavaScript possède une méthode qui appel le service web pour sauvegarder les données et rafraîchir ensuite le tableau. L'image suivante présente le code qui fait ceci.

```
$('#table_local_parameter').each(function(){
    $(this).find('tr').not(':first').each(function(){
        let name = $(this).find('td').eq(0).text();
        let value = $(this).find('input').eq(0).val();
        let formula = $(this).find('input').eq(1).val();

        if(!formula)
            formula = " ";
        if(!value)
            value = 0;

        getAjaxFromRouter(
            jsModel.controllers.API.model.Process.setParameterFormula(processId, formula, name, value),""
        ).fail(function(jqXHR, textStatus, errorThrown) {
            console.log("failed!");
            console.log(errorThrown);
        });
    });
});

if(iterateOnce != 0){
    iterateOnce--;
    setParameterFormula(processId, iterateOnce);
}

refreshProcessParams(PARAM_TABLE_TEMPLATE,PARAM_ROW_TEMPLATE);
```

Lorsqu'il n'y a pas de formule ou de valeur, je me suis assuré d'envoyer soit un *String* vide ou une valeur de 0 afin de permettre à l'interpréteur de formule de fonctionner. Le service web s'occupe de remettre les valeurs voulues par l'utilisateur.

Après l'appel au service web, il y a un appel récursif permettant de modifier le tableau au complet à la suite d'une modification. Puisque l'algorithme passe à travers le tableau et prend les valeurs dans les colonnes, il faut passer une deuxième fois après que l'interpréteur de formule calcule les nouvelles valeurs. Un rafraîchissement du tableau avec les nouvelles données est appelé.

Cydrick

Durant cette itération, Cydrick a travaillé pour réduire le temps que prend la conversion des résultats en fichier Excel. Cydrick a aussi amélioré le gabarit selon les demandes du promoteur du projet. Notamment, la feuille de calcul nommée « LCI by category » comportait les mauvaises données sur les lignes et les colonnes. Le correctif a été apporté.

À ce stade, Apache Xalan est le goulot d'étranglement pour l'exportation Excel. Cette librairie de transformations de documents nécessite 2min 55sec pour générer un fichier XML. En comparaison, des estimations faites au début du projet prévoient qu'environ 13sec seraient nécessaire pour effectuer l'exportation à l'aide de concaténation de texte. Ce fait démontre qu'Apache Xalan ralentit le temps d'exportation.

Une première tentative d'amélioration de temps d'exportation a été effectuée en changeant Apache Xalan par Saxon. Saxon est une librairie de transformations XML qui respecte la même syntaxe de balisage qu'Apache Xalan, c'est-à-dire des balises *<xsl:... />*. Saxon respecte aussi les mêmes interfaces de code Java qu'Apache Xalan. Par conséquent, le changement a été relativement rapide puisqu'il ne fut pas nécessaire de modifier la logique d'obtention des données, le gabarit ni la logique d'instanciation du transformateur. Toutefois, les performances se sont avérées décevantes. La transformation s'est effectuée en 2min 15sec, ce qui ne représente qu'une amélioration de 40sec par rapport à Apache Xalan.

Cydrick a ensuite tenté d'écrire une librairie qui permet la transformation de fichiers contenant des balises génériques. Cette librairie aurait permis la conversion à l'aide de flux de données, ce qui aurait permis de garder les avantages d'Apache Xalan. Toutefois, la tâche s'est avérée relativement complexe. Coder une librairie de transformations de gabarits qui supporte des boucles itératives ainsi que des boucles dans des boucles peut prendre un certain temps à développer. Si la notion de flux avait été insérée dans cette librairie, le temps de développement et de tests aurait grandement augmenté. Avec les deux semaines restantes, la tâche ne semblait pas réalisable. Donc, cette approche a été évitée.

Une troisième approche a été tentée avec Apache Velocity. Apache Velocity est une librairie de transformation de gabarit, mais qui n'est pas limitée à la transformation de fichiers XML. Apache Velocity permet les mêmes fonctionnalités qu'Apache Xalan, c'est-à-dire des boucles, des conditions, des sélections de données, etc. L'avantage majeur d'Apache Velocity est que les données sont envoyées au transformateur sous la forme de « *Map<Key, Value>* », où « *Value* » peut être un nombre, du texte ou même une autre « *Map<Key, Value>* », ce qui permet une profondeur similaire aux « *Element* » XML. En utilisant Apache Velocity, la syntaxe des balises est légèrement différente. Par exemple, une balise de boucle sur une « *Map* » s'écrit comme « *#foreach($submap in $Map.Key)* », et une sélection de variables dans cette « *Map* » s'écrit comme « *${submap.SubMapKey}* ». Voir l'annexe V pour une comparaison de la syntaxe des balises d'Apache Xalan et d'Apache Velocity.

Après la conversion du gabarit pour la syntaxe d'Apache Velocity, la classe « *ExcelVelocityFileCreator.java* » a été créée. Cette classe a pour but de replacer « *ExcelXmlFileCreator.java* » et de combiner le gabarit de fichier Excel XML codé selon la

syntaxe d'Apache Velocity ainsi que les données. La classe « *CalculationApiMappedData.java* » a aussi été créé et son but de remplacer « *CalculationApiXmlData* » en écrivant les données dans des « *Map* » au lieu d'« *Element* » XML. Toutefois, la classe « *CalculationApiXmlData* » n'a pas été complétée cette semaine puisque les deux autres tentatives ont pris plus de temps que prévue à réaliser.

Il fut nécessaire d'apporter des modifications à la structure des flux. En effet, Apache Velocity ne permet pas la conversion dans un flux d'écriture (ou « *OutputStream* »), mais bien d'une classe d'écriture d'octet (ou « *Writer* »). La solution fut d'utiliser un « *OutputStreamWriter* » qui implémente « *Writer* » et qui écrit le résultat dans un « *OutputStream* » fourni en paramètres. Voici donc la chaîne de flux utilisée à ce moment du développement:

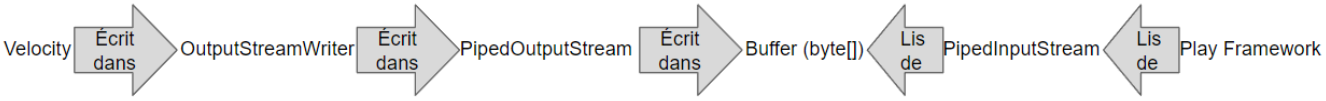


Figure 10.2.1: Structure de flux avec Apache Velocity

Jean-Pierre

Dans cette itération, Jean-Pierre a continué d'élaborer le graphique final. Le graphique est capable de présenter des données provenant d'un service web. Pour l'instant, les données sont simulées de façon aléatoire; elles ne proviennent pas de la base de données. Il est à noter que les données ne sont pas seulement un "Math.random()", car les valeurs retournées par cette fonction sont situées entre 0 et 1. Donc, on multiplie le résultat par 100 pour avoir des données plus facilement lisibles et on ajoute 0.2 avant la multiplication, pour s'assurer que le nombre soit suffisamment grand pour être visible sur le graphique (car des valeurs bases telles que 0.001 multipliées par 100 sont quand même trop petites pour être convenablement affichée sur le graphique).

```
private double rand(){
    //we add 0.2 to avoid low data (and invisible lines in the chart)
    return (double) Math.ceil((Math.random() + 0.2) * 100);
}
```

Pour obtenir les données du moteur arrière de l'application, il faut faire un appel *AJAX* dans le fichier *javascript* vers la classe "ChartsTest.*java*" comme ci:

```
//get base data
getAjaxFromRouter(
        jsModel.controllers.API.model.ChartsTest.getData("base"),""
).done(function(response){
    setGraphData(response);
    setDataProvider(0);
    refreshChart();
```

La classe répondra selon l'indicateur demandé. Dans cet exemple, on demande "base" qui sont les données de base du graphique. En arrière-plan, les données sont retournées à l'aide d'une opération "Switch/Case", ce qui pourrait par plus tard être changé entièrement avec des requêtes vers la base de données. Dans le fichier javascript, il ne suffit que de changer l'attribut "DataProvider*"* du graphique *AmCharts* puis de réutiliser la fonction *refreshChart*() pour le mettre à jour.

Le graphique présentement fonctionne bien, mais ne répond pas aux attentes du promoteur. L'exploration des données fonctionne par catégorie et le promoteur voudrait une exploration par type de données. Il faudra trouver comment, via le code, détecter quel type de données est cliqué par l'utilisateur pour permettre le fonctionnement voulu. Aussi, le graphique n'a pas encore de titre, ni aucune façon de savoir où nous sommes rendus dans les données. La fonction "Go Back" retourne toujours au début, mais le promoteur souhaiterait pouvoir revenir d'un niveau plutôt que la remise à zéro à chaque fois.

De plus, le promoteur souhaiterait qu'il ne soit pas possible de cliquer sur la légende des données pour cacher les données correspondantes sur le graphique. Il est d'avis qu'un utilisateur pourrait les cacher par inadvertance et fausser les données.

Bref, il reste quelques ajustements à faire sur le graphique pour qu'il soit terminé.

Figure 10.3.1 Premier prototype fonctionnel

Aravinthan

Enfin revenu de son voyage, Aravinthan a continué à améliorer son algorithme pour naviguer dans des résultats tel qu'il était au moment du calcul. Il s'est rendu bien vite qu'il y avait des parties qui ne fonctionnaient pas comme qu'il le pensait. En effet, Aravinthan pensait qu'un resultSet contiendrait des foreign key pour les différents table history mais ce n'est pas le cas. Il voulait donc parler avec le promoteur à la prochaine réunion pour pouvoir régler cela. Par la suite, Aravinthan a travaillé sur la partie graphique de l'arbre de navigation. Il a créé une boite déroulante pour pouvoir choisir les différents résultats présents dans la base de données. Il a donc créé un nouveau endpoint dans les routes. Il a également créé une méthode, code dans l'image ci-dessous, dans le contrôleur pour avoir la liste des résultats.

```
 */
public class ResultSets extends UBUBIController
{
    @Inject
    private Spark spark;

    @Inject
    public ResultSets(JPAApi api) { super(api); }

    @Transactional(readOnly = true)
    public Result all()
    {
        List<models.LCA.LCIResultSet> lstVal = null;
        try
        {
            Query query = JPA.em().createQuery( qlString: "Select m from LCIResultSet m");
            lstVal = query.getResultList();
        }
        catch (NoResultException e) {}

        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append(JSONUtil.parseArray( key: "lst",lstVal, JSONFormater.LCIResultSetFormater));
        sb.append("}");

        return jsonResult(ok(sb.toString()));
    }
}
```

CHAPITRE 11: 25 juillet au 1er août

Daniel

Dans cette dernière semaine, Daniel a apporté des ajustements pour la modification des paramètres locaux. Ces ajustements ont été demandés à la suite de la rencontre hebdomadaire. Cette tâche fut la seule effectué cette semaine, en plus de l'accès au paramètre global dans les paramètres locaux.

Un problème d'affichage intermittent lorsqu'un paramètre est mise à jour, certain paramètre dépendant de celui-ci ne se mettais pas à jour. Le problème provenait lors du rafraîchissement du tableau, car il se faisait rafraîchir sans posséder les nouvelles données calculées. Cependant, l'appel au service web ne se synchronise pas avec le reste du code JavaScript. J'ai donc dû ajouter l'appel à la méthode « *refreshProcessParams* » dans le « *done* » à la suite de l'appel au service web. Ceci a permis de ne plus avoir de problème d'affichage des valeurs après avoir fait des modifications.

```
getAjaxFromRouter(
    jsModel.controllers.API.model.Process.setParameterFormula(processId, formula, name, value),""
).done(function(response){
    refreshProcessParams(PARAM_TABLE_TEMPLATE, PARAM_ROW_TEMPLATE);
}).fail(function(jqXHR, textStatus, errorThrown) {
    console.log("failed!");
    console.log(errorThrown);
});
```

Une autre tâche accomplie par Daniel est l'accès des paramètres globaux dans les formules des paramètres locaux. Un ajout dans le service web fut nécessaire pour la gestion de la modification d'une formule si elle contient un paramètre global. De la même façon que les paramètres locaux, je suis allé chercher la liste de ces paramètres et je passais à travers la liste pour modifier la formule contenant le nom du paramètre avec la nouvelle valeur. L'image suivante présente cet algorithme.

```
private String replaceNewFormulaWithRealValue(List<Parameter> parameters, String newFormula) {

    for(Parameter parameter : parameters) {
        if(newFormula.toLowerCase().contains(parameter.getName().toLowerCase())) {
            newFormula = newFormula.replaceAll("(?i)"+parameter.getName(), String.valueOf(parameter.getValue()));
        }
    }

    return newFormula;
}
```

Il est maintenant possible de mettre dans la formule des majuscules et des minuscules pour calculer la valeur. La partie dans l'image ci-haut permet de gérer les minuscules et majuscules. La figure suivante montre l'utilisation de variable globale dans la formule des paramètres locaux. La variable globale est « *duree_vie_batiment* » qui a une valeur de 70.

Process | Exchanges | Parameters | Global Parameters

Edit

Name	Value	Description	Formula
area	71		volume+1
volume	70		duree_vie_batiment

Cydrick

Pour cette dernière semaine, Cydrick a continué l'amélioration des performances de l'exportation Excel et a finalisé le gabarit du fichier Excel XML.

Tout d'abord, il fut nécessaire de finaliser la transition du format des données d'« *Element* » vers « *Map* » dans la classe qui s'occupe d'obtenir les données. Une fois cette transition effectuée, il fut possible de faire une première exportation en utilisant Apache Velocity. Voici le temps nécessaire pour terminer l'opération:

- Obtention des données avec SQL: 2sec
- Transformation du gabarit sur les données avec Apache Velocity: 5sec
    - Téléchargement du fichier localement: Inclus dans le temps de conversion
- Total: 7sec

Le temps de conversion a donc passé de 2min 55sec à 5 secs. Selon une hypothèse, ceci est dû au fait qu'Apache Xalan doit itérer au travers des « *Element* » XML (donc en O(n)) pour insérer les bonnes données aux bons endroits dans le gabarit. L'accès de données dans une « *Map* » est plus rapide puisqu'une « *Map* » utilise des clés pour obtenir exactement la bonne donnée (donc en O(1)). Une autre hypothèse est qu'Apache Xalan doit interpréter les « *Element* » en « *String* » avant de pouvoir les utiliser pour transformer le gabarit, ce que Velocity n'a pas à faire puisque les données possèdent déjà le bon format dans le « *Map* ».

Le promoteur du projet a aussi demandé de compresser le contenue du fichier dans le format Zip avant de le retourner au navigateur. Ceci est une bonne idée puisque le fichier contient beaucoup de répétitions. En effet, chaque ligne d'une feuille de calcul se ressemble beaucoup, seulement les valeurs changent. L'algorithme de compression Zip serait donc en mesure de donner un haut taux de compression.

L'implémentation de la compression Zip fut relativement facile. En effet, puisque Java offre une classe de compression Zip qui supporte les flux, il suffit d'ajouter un flux dans la série de flux déjà présent. Voici la nouvelle chaîne de flux, incluant le nouveau « *ZipOutputStream* »:

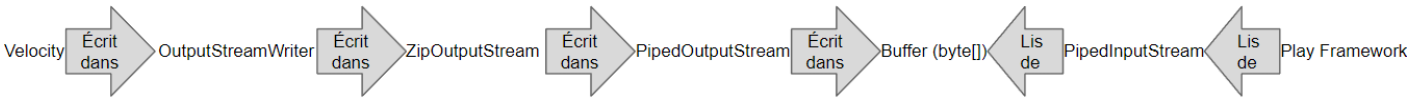


Figure 11.2.1: Structure de flux avec Apache Velocity & compression Zip

La compression du fichier Excel XML permet un ratio de compression de 98%. Alors, bien que le fichier décompressé possède une taille de 266.026mb, la taille du fichier compressé est de 5.613mb. De plus, le temps de téléchargement est aussi accéléré puisque le fichier est considérablement plus petit. Voici les temps finaux avec la compression Zip:

- Obtention des données avec SQL: 2sec

- Transformation du gabarit sur les données avec Apache Velocity: 3sec
    - Téléchargement du fichier localement: Inclus dans le temps de conversion
- Total: 5sec

Ce temps est final, les modifications subséquentes pour la dernière semaine n'affectent pas ces temps. Ce temps est aussi calculé sur une configuration matérielle d'une qualité moyenne. Les temps peuvent varier sur des ordinateurs dont la puissance est plus ou moins haute. Malgré tout, le temps de l'opération a passé de 1h 4min 32sec à 5sec, ce qui représente une amélioration de 1h 4min 27sec, ou de 99.87%.

Par la suite, des modifications finales au gabarit ont été faites, notamment:

- Afficher les valeurs dans la feuille « Contribution Analysis » en pourcentage;
- Cacher les pourcentages à 0 dans la feuille « Contribution Analysis »;
- Fixer le nombre de colonnes filtrées dans les feuilles « Contribution Analysis » et « LCI by Category » pour avoir exactement le bon nombre de colonnes retournées (et non 100 par défaut).

La version finale du gabarit est disponible dans l'annexe VI.

Ensuite, une réingénierie a été faite sur le module de conversion pour pouvoir permettre de transformer d'autres gabarits que celui de Microsoft Excel XML, bien que ce ne fut pas une exigence du projet. Un autre but de cette réingénierie fut de permettre plus d'un fichier dans le fichier compressé Zip. Le résultat a consisté en 5 classes Java:

- Paquet *velocityFileCreation*
    - *IVelocityContextCreator*: Interface qui permet de retourner une liste de *VelocityMappedFile*.
    - *VelocityMappedFile*: Contient la correspondance entre l'emplacement d'un gabarit, les données insérer dans le gabarit (contenue dans la variable *velocityContext*) ainsi que le nom de fichier que le gabarit transformé doit posséder dans le fichier Zip.
    - *VelocityZipFileCreator*: S'occupe de créer le fichier compressé Zip et de donner son contenu dans un *InputStream*. Prends une classe de type *IVelocityContextCreator*, obtiens la liste des *VelocityMappedFile*, applique la transformation sur chacun d'eux et insère le résultat dans le fichier compressé.
- Paquet *excel*
    - *CalculationApiMappedData*: S'occupe d'obtenir les données dans la base de données et les insère dans plusieurs *Map* avec une profondeur appropriée pour bien pouvoir représenter les données.
    - *ExcelVelocityMappedData*: Hérite de *CalculationApiMappedData* et implémente *IVelocityContextCreator*. Prends l'ensemble des données retournées par *CalculationApiMappedData* et les insère dans une *List<VelocityMappedFile>* que *VelocitySipFileCreator* peut ensuite utiliser pour inclure dans le fichier compressé.

Pour voir le diagramme de classe de ces deux modules, consulter l'annexe VII.

Ceci offre trois avantages. Tout d'abord, le transformateur peut supporter différents gabarits puisque ces informations sont entreposées dans « *VelocityMappedFile* ». Ensuite, il est possible de mettre plusieurs fichiers dans le fichier compressé puisque « *IVelocityContextCreator* » permet de retourner une « *List<VelocityMappedFile>* ». Finalement, les données obtenues par « *CalculationApiMappedData* » pourraient être utilisé pour être affichées dans un autre fichier que le fichier Excel XML.

Finalement, pour la 12e semaine, le promoteur du projet a demandé à Cydrick d'implémenter un bouton dans le « *devZone* » pour permettre le téléchargement du fichier compressé. L'URL finale choisie pour le téléchargement du fichier est « */api/Spark/GetExcelFile/<ResultSetId>* », où « *<ResultSetId>* » est l'« id » du « *ResultSet* » (par exemple, *1*).

Jean-Pierre

Pour la dernière itération, Jean-Pierre a modifié le graphique pour mieux répondre aux demandes du promoteur. Le graphique permet d'approfondir les données par type, en conservant les catégories d’impacts (santé, réchauffement climatique, contamination de l’eau et autre) plutôt que d'approfondir par les catégories d’impacts.

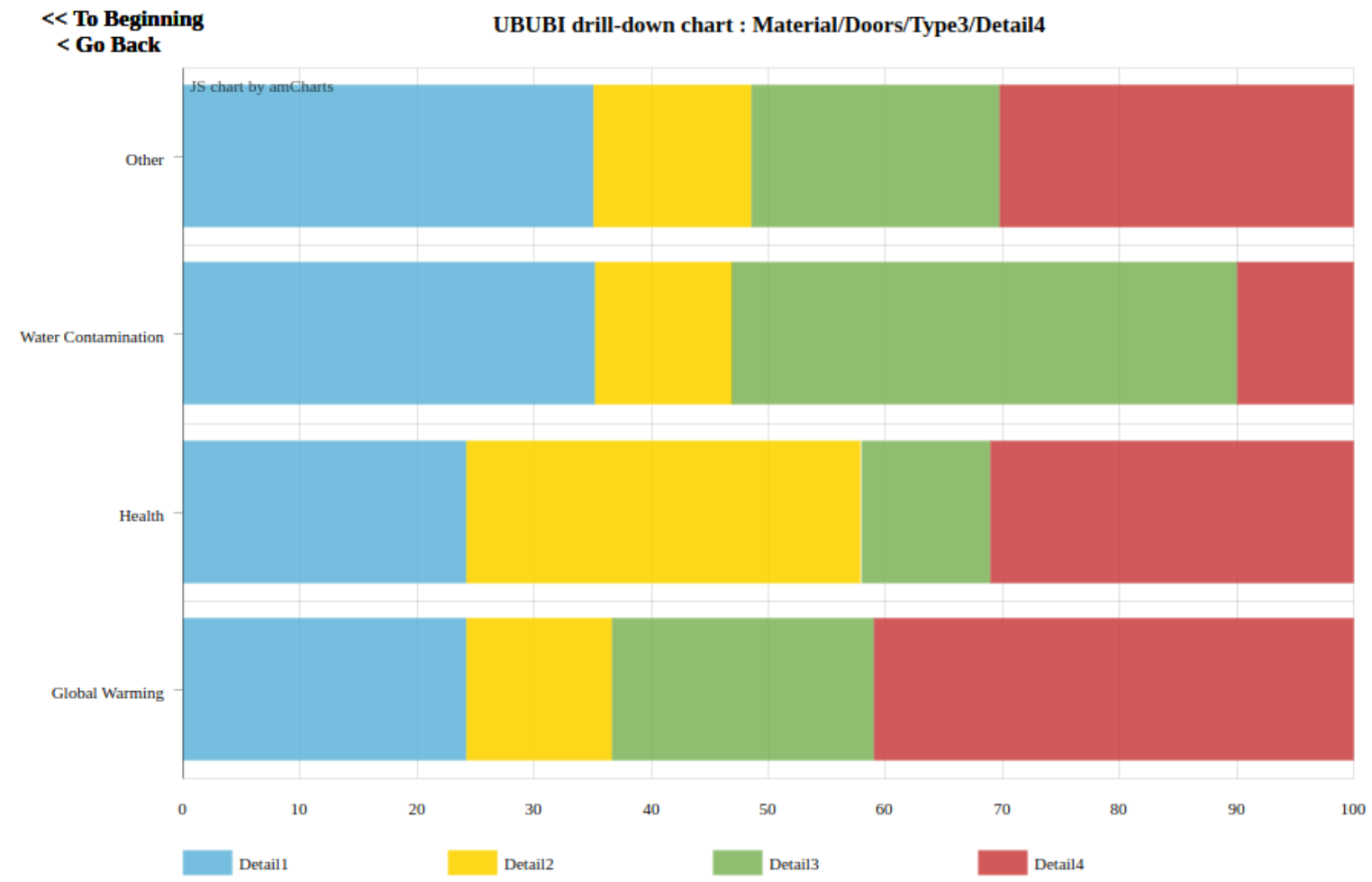


Figure 11.3.1 Graphique final

Aussi, la fonction « Go Back » permettait précédemment de revenir à l’état initial du graphique. Cette fonction existe encore, mais maintenant s’appelle « To Beginning ». La fonction « Go Back » maintenant permet de remonter d’un niveau dans l’approfondissement du graphique.

De plus, le code contient maintenant un tableau de résultats appelé « graphData », qui agit comme un patron « Memento ». Ceci permet de conserver les données lorsque l'on souhaite remonter de niveaux le graphique, ainsi que les données originales si on souhaite revenir au tout début. Le fonctionnement est simple; un index appelé « graphIndex » est gardé en mémoire pour savoir où nous sommes situés dans le tableau de résultat, puis cet index est manipulé lorsque nécessaire pour ajouter des résultats ou revenir en arrière (ou même revenir à zéro). Le seul pépin important à mentionner qui est apparu suite à cette implémentation est l'affichage du titre, où Jean-Pierre a dû utiliser la fonction « split » et une boucle « for » pour manipuler le texte du titre lorsque l'on cherche à retourner en arrière. Ce n'est pas optimal, mais c'est le prix à payer pour avoir traité ce problème après avoir codé le nécessaire pour permettre l'approfondissement dans le graphique.

```
// function which resets the chart back to base data
// taken from https://www.amcharts.com/kbase/drill-column-chart/
function resetChart() {
    setGraphIndex(0);
    setDataProvider(0);
    chart.titles[0].text = "UBUBI drill-down chart : Material";

    // remove the labels
    chart.allLabels = [];

    //reset values
    updateGraphsLabels(0);
    refreshChart();
}

function goBack() {
    if (getGraphIndex() == 1) {
        resetChart();
    } else {
        setGraphIndex(getGraphIndex() - 1);
        updateGraphsLabels(getGraphIndex());
        setDataProvider(getGraphIndex());
        var titleSplit = chart.titles[0].text.split("/");
        var newtitle = "";
            for(x=0; x <= getGraphIndex(); x++) {
                newtitle = newtitle + titleSplit[x];
                if (x != getGraphIndex()){
                    newtitle = newtitle + "/";
                }
            }
        chart.titles[0].text = newtitle;
        refreshChart();
    }
}
```

Aussi, la valeur en pourcent est maintenant affichée à côté des types de données lorsque la souris survole une rangée. Cette addition permet d'afficher les données sous un format simple à lire et compréhensible

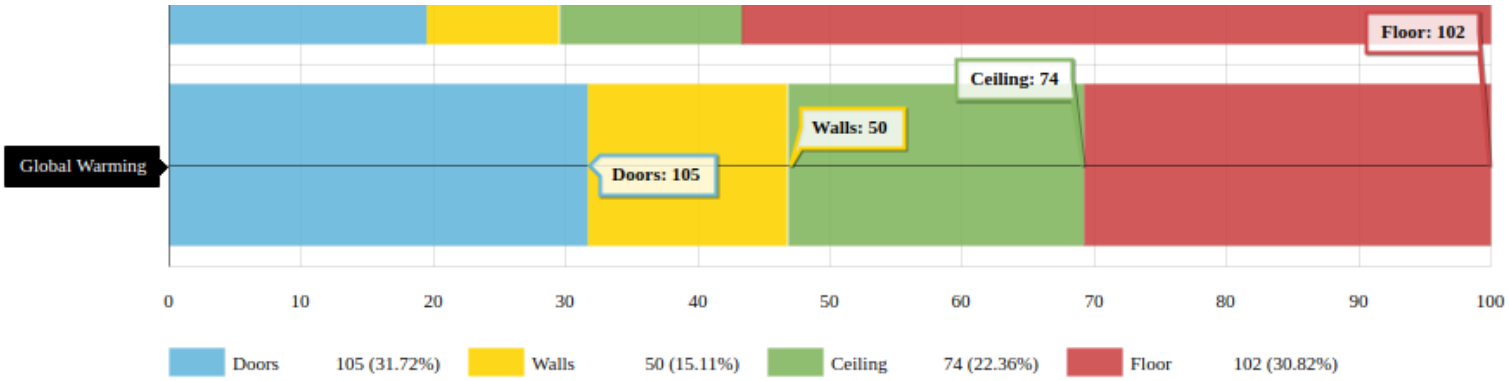


Figure 11.3.2 Pourcentages des données dans la légende

Pour terminer, Jean-Pierre a aussi trouvé comment désactiver la possibilité de cacher des données du graphique. Avant, il était possible de cliquer sur la légende du graphique pour désactiver ce type de données (disons qu'on clique sur le rouge, alors le rouge disparaît du graphique dessiné). Le promoteur considérait que c'était inutile et ne pouvait seulement que fausser les données au cas où un utilisateur clique sur la légende par inadvertance. Bref, il ne suffisait que d'indiquer que la légende ne soit pas « switchable » dans le code pour désactiver ce comportement.

```
"legend": {
    "enabled": true,
    "useGraphSettings": true,
    "valueText": "[[value]] ([[percents]]%)",
    "valueWidth": 80,
    "switchable": false
}
```

Aravinthan

Durant la réunion, le promoteur s'est rendu compte qu'il fallait trouver une façon d'identifier les différents « resultSet » et non juste avec le « timestamp ». Aravinthan a donc proposé d'utiliser un champ description. Chaque fois qu'un utilisateur va utiliser la fonction de « result », il va falloir ajouter une description en même temps.

Aravinthan a donc rajouté un « textfield » sur la page calcul. Il a également utilisé le code jQuery pour que ce champ apparaisse en même temps que les différents éléments de calcul.

```
hideCalculateButtonAndComboBox = function() {
    $('#methodSelect').css('visibility', 'hidden');
    $('#methodSelect').hide();

    $('#methodDescription').css('visibility', 'hidden');
    $('#methodDescription').hide();

    $('#btnCalculateResult').css('visibility', 'hidden');
    $('#btnCalculateResult').hide();
};

showCalculateButtonAndComboBox = function() {
    $('#methodSelect').css('visibility', 'visible');
    $('#methodSelect').show();

    $('#methodDescription').css('visibility', 'visible');
    $('#methodDescription').show();

    $('#btnCalculateResult').css('visibility', 'visible');
    $('#btnCalculateResult').show();
};
```

Par la suite, il a dû rajouter ce champ dans le modèle et trouver comment ajouter ce champ dans le SQL. Une fois cette étape terminée, ajouter la description était assez simple. Grâce à ce petit ajout, il est maintenant beaucoup plus facile de comprendre ce qu'on cherche.

```
    JsonNode json = ws.url(calculateServiceUrl).post(Source.from(lstParam)).toCompletableFuture().get().asJson();

    insertModelLCADatabase = new InsertModelLCADatabase(json, database, id, description);
    insertModelLCADatabase.startInsertingModelInDatabase();
```

Aravinthan a réussi à terminer le code qui permet d'aller chercher le « history » qui est juste avant celui d'un calcul. Par la suite, à moins d'une insertion, on va utiliser le tableau « history ». Cependant, si c'est une insertion, les données du tableau « history » sera vide. Alors, il faut donc utiliser les données du modèle courant.

```
public Result getInfoWithHistory(Long id, Long idResultSet) throws SQLException
{
    org.openlca.core.model.Process returnVal = JPA.em().find(org.openlca.core.model.Process.class,id);
    if (returnVal == null)
        return jsonResult(notFound());

    PreparedStatement preparedStatement = this.database.getConnection().prepareStatement( sql: "SELECT * FROM `tbl_processes_history` WHERE id=?", Statement.RETURN_GENERATED_KEYS);
    preparedStatement.setLong( parameterIndex: 1, id);

    ResultSet rs = preparedStatement.executeQuery();
    if (!rs.next() ) {
        return jsonResult(notFound());
    }

    LCIResultSet resultSet = JPA.em().find(LCIResultSet.class, idResultSet);
    Timestamp timestamp = resultSet.getCalc_time();

    while (rs.next()) {
        if(rs.getTimestamp( columnLabel: "dt_datetime").before(timestamp)){
            if(rs.getString( columnLabel: "action").compareTo( anotherString: "insert") == 0){
                return jsonResult(ok(JSONFormater.ProcessFormater.parseObject(returnVal)));
            }
            else {

                long historyId = rs.getLong( columnLabel: "id");

                ProcessHistory data = JPA.em().find(ProcessHistory.class, historyId);

                String strData = "{" +
                        JSONUtil.parseCategorizedEntity(data) + "," +
                        JSONUtil.parseEscapedString("processType", data.getProcessType()==null?null:data.getProcessType().toString()) + "," +
                        JSONUtil.parseEscapedString("defaultLocationMethod", data.getDefaultAllocationMethod()==null?null:data.getDefaultAllocationMethod().toString()) + "," +
                        JSONUtil.parse("infrastructure", data.isInfrastructureProcess()) + "," +
                        JSONUtil.parse("processType", data.getProcessType().toString()) + "," +
                        JSONUtil.parseEscapedString("documentation", data.getDocumentation()==null?null:routes.ProcessDocumentation.getInfo(data.getDocumentation().getId()).url()) + ","
                        JSONUtil.parse("exchanges", routes.Process.getExchanges(data.getId()).url()) + "," +
                        JSONUtil.parse("parameters", routes.Process.getParameters(data.getId()).url()) + "," +
                        JSONUtil.parseEscapedString("location", data.getLocation()==null?null:routes.Location.getInfo(data.getLocation().getId()).url()) + "," +
                        JSONUtil.parseEscapedString("kmz", data.getKmz()==null || data.getKmz().length==0?null:routes.Process.getKmz(data.getId()).url()) + "," +
                        JSONUtil.parseEscapedString("quantitativeReferenceId",data.getQuantitativeReference()==null?null:routes.Exchange.getInfo(data.getQuantitativeReference().getId())
                        "}";
                return jsonResult(ok(strData));
            }
        }
```

Par la suite, il a utilisé la même approche pour faire les méthodes pour le FlowHistory et ExchangesHistory.

```
@Transactional(readOnly = true)
public Result getInfoWithHistory(Long id, Long idResultSet) throws SQLException {
    org.openlca.core.model.Flow returnVal = JPA.em().find(org.openlca.core.model.Flow.class, id);
    if (returnVal == null)
        return jsonResult(notFound());

    PreparedStatement preparedStatement = this.database.getConnection().prepareStatement( sql: "SELECT * FROM `tbl_flows_history` WHERE id=?", Statement.RETURN_GENERATED_KEYS);
    preparedStatement.setLong( parameterIndex: 1, id);

    ResultSet rs = preparedStatement.executeQuery();
    if (!rs.next()) {
        return jsonResult(notFound());
    }

    LCIResultSet resultSet = JPA.em().find(LCIResultSet.class, idResultSet);
    Timestamp timestamp = resultSet.getCalc_time();

    while (rs.next()) {
        if (rs.getTimestamp( columnLabel: "dt_datetime").before(timestamp)) {
            if (rs.getString( columnLabel: "action").compareTo( anotherString: "insert") == 0) {
                return jsonResult(ok(JSONFormater.FlowFormater.parseObject(returnVal)));
            } else {
                long historyId = rs.getLong( columnLabel: "id");

                FlowHistory data = JPA.em().find(FlowHistory.class, historyId);

                String strData = "{" +
                        JSONUtil.parseCategorizedEntity(data) + "," +
                        JSONUtil.parse("producers",routes.Flow.getFlowProducers(data.getId()))+","+
                        JSONUtil.parse("consumers",routes.Flow.getFlowConsumers(data.getId()))+","+
                        JSONUtil.parseEscapedString("flowType", data.getFlowType().toString())+","+
                        JSONUtil.parse("units",routes.Flow.getUnit(data.getId()))+","+
                        JSONUtil.parse("fullCategory", routes.Flow.getFullCategory(data.getId())) +
                        "}";
                return jsonResult(ok(strData));
            }
        }
    }

    return jsonResult(notFound());
}
```

CONCLUSION

Chaque semaine, nous avons su satisfaire les nouvelles demandes du promoteur en implémentant de nouvelles fonctionnalités dans le projet ou faire de l'analyse. La contribution des membres de l'équipe, les problèmes qu'ils ont rencontrés et les décisions qu'ils ont prises sont présents dans chaque chapitre. Chacun s'est comporté de façon professionnelle en respectant les demandes du promoteur et en livrant du code de qualité.

Ce projet nous a permis de mettre en pratique plusieurs aspects que nous avons vus à travers notre cheminement scolaire. Que ce soit de l'analyse, la conception, de l'implémentation, des techniques d'algorithme ou de la gestion de projet, nous avons mis en pratique tous ces aspects dépendamment des tâches. Certaines d'entre-elles demandaient d'utiliser plusieurs notions apprises dans des cours différents afin de bien les compléter.

En prenant du recul face au projet, nous constatons que nous avons accompli plusieurs tâches afin d'agrandir le projet. Tous les membres sont satisfaits de leurs accomplissements et de leur engagement. Certaines semaines furent plus longues et plus difficiles que d'autres, mais nous sommes arrivés à accomplir toutes les tâches demandées par le promoteur.

BIBLIOGRAPHIE

AmCharts. 2017. «Simple Column Chart». In *AmCharts*. En ligne. <https://www.amcharts.com/demos/simple-column-chart/>. Consulté le 13 août 2017.

AmCharts. 2017. «Online Chart Maker». In *AmCharts*. En ligne. <https://live.amcharts.com/>. Consulté le 15 août 2017.

AmCharts. 2017. «Stacked Column Chart». In *AmCharts*. En ligne. <https://www.amcharts.com/demos/stacked-column-chart/>. Consulté le 15 août 2017.

AmCharts. 2017. «Drill-down column chart». In *AmCharts*. En ligne. <https://www.amcharts.com/kbase/drill-column-chart/>. Consulté le 15 août 2017.

AmCharts. 2017. «Multidimensional drill-down». In *AmCharts*. En ligne. <https://www.amcharts.com/kbase/multidimensional-drilldown-back-button/>. Consulté le 15 août 2017.

Apache Software Foundation. 2006. «Xalan - Basic usage patterns». In *The Apache XML Project*. En ligne. <https://xml.apache.org/xalan-j/usagepatterns.html>. Consulté le 17 août 2017.

Free Software Foundation. 2017. «Unofficial Translations». In *GNU*. En ligne. <https://www.gnu.org/licenses/translations.en.html>. Consulté le 15 août 2017.

Geir Magnusson Jr. 2001. «How-To Start up the Velocity Template Engine». In *JavaWorld*. En ligne. <http://www.javaworld.com/article/2075966/core-java/start-up-the-velocity-template-engine.html> Consulté le 17 août 2017.

Google. 2017. «Sankey Diagram». In *Google Charts*. En ligne. <https://developers.google.com/chart/interactive/docs/gallery/sankey>. Consulté le 13 août 2017.

Java2S. 2017. «Test PipedInputStream and PipedOutputStream with Thread: PipedInputStream << File Input Output << Java». In *Java2S*. En ligne. <http://www.java2s.com/Code/Java/File-Input-Output/TestPipedInputStreamandPipedOutputStreamwithThread.htm>. Consulté le 16 août 2017.

Microsoft. 2016. «Overview of XML in Excel». In *Office Support*. En ligne. <https://support.office.com/en-us/article/Overview-of-XML-in-Excel-f11faa7e-63ae-4166-b3ac-c9e9752a7d80>. Consulté le 14 août 2017.

Oracle. 2016. «Class PipedInputStream». In *Java Docs*. En ligne. <https://docs.oracle.com/javase/7/docs/api/java/io/PipedInputStream.html>. Consulté le 16 août 2017.

RAHMAN, Syed Fazle. 2017. «16 JavaScript Libraries for Creating Beautiful Charts». In *sitepoint*. En ligne. <https://www.sitepoint.com/15-best-javascript-charting-libraries/>. Consulté le 15 août 2017.

The Apache Software Foundation. 2017. «Apache License Version 2.0». In *Apache*. En ligne. <https://www.apache.org/licenses/LICENSE-2.0>. Consulté le 15 août 2017.

Utilisateur nommé "user479534". 2011. «How do I add text to the beginning of a file in Bash?». In *SuperUser*. En ligne. <https://superuser.com/questions/246837/how-do-i-add-text-to-the-beginning-of-a-file-in-bash>. Consulté le 15 août 2017.

Utilisateur nommé "Roland". 2011. «How to loop through a directory recursively to delete files with certain extensions». In *StackOverflow*. En ligne. <https://stackoverflow.com/questions/4638874/how-to-loop-through-a-directory-recursively-to-delete-files-with-certain-extensi>. Consulté le 15 août 2017.

Utilisateur nommé "vikingsteve". 2015. «Can't see project folders in IntelliJ IDEA». In *StackOverflow*. En ligne. <https://stackoverflow.com/questions/28359018/cant-see-project-folders-in-intellij-idea>. Consulté le 16 août 2017.

Utilisateur nommé "GamerJosh". 2013. «How do i change the IntellJ IDEA default JDK». In *StackOverflow*. En ligne. <https://stackoverflow.com/questions/18987228/how-do-i-change-the-intellij-idea-default-jdk>. Consulté le 16 août 2017.

Utilisateur nommé "monolithed". 2012. «Markdown plugin crashes the IDE #82». In *Github*. En Ligne. <https://github.com/nicoulaj/idea-markdown/issues/82>. Consulté le 16 août 2017.

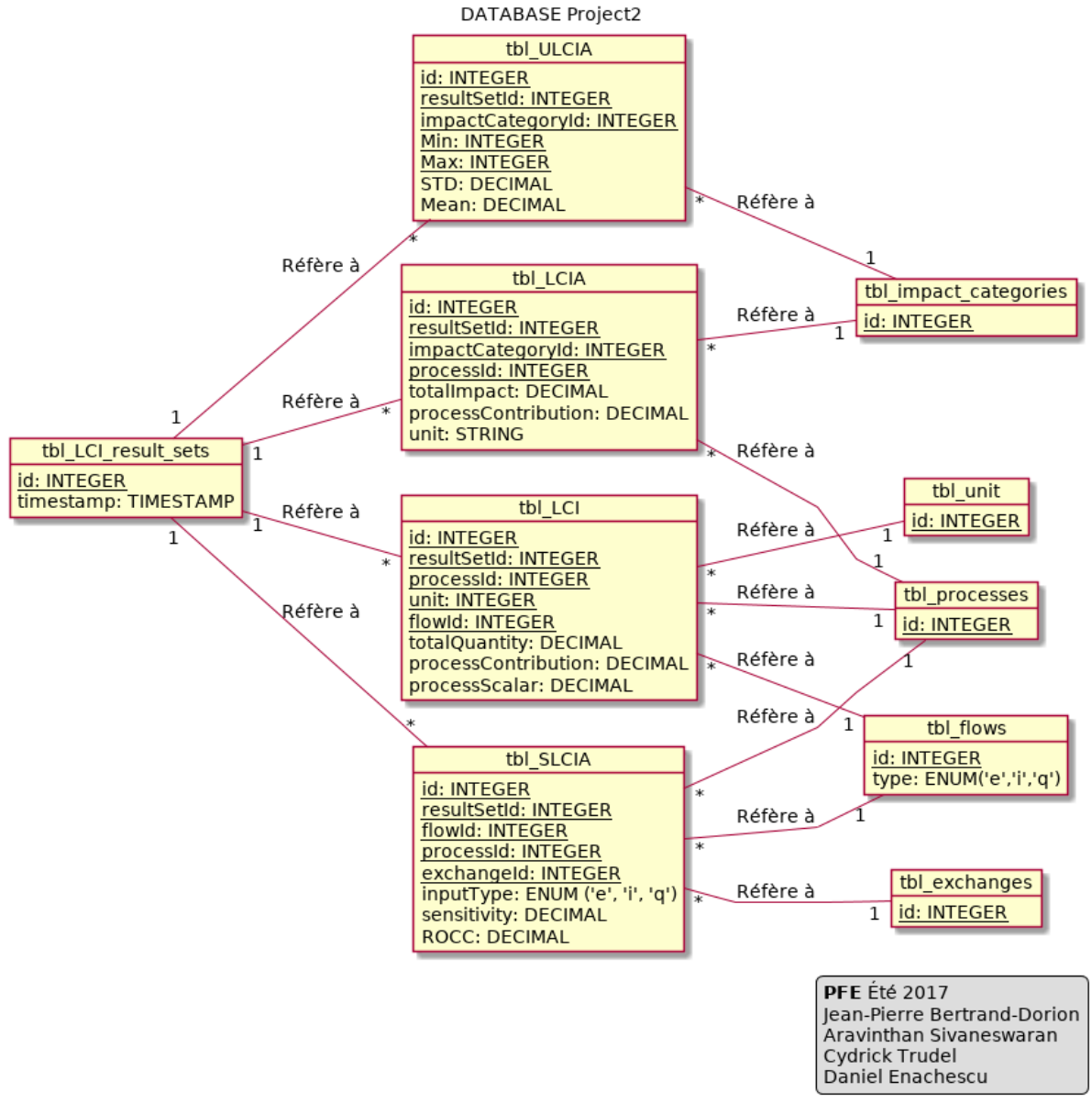
Wayne Burkett. 2012. «How to set Saxon as the Xslt processor in Java». In *StackOverflow*.

En ligne. <https://stackoverflow.com/questions/11314604/how-to-set-saxon-as-the-xslt-processor-in-java>. Consulté le 17 août 2017.

Wikipedia. 2017. «Diagramme de Sankey». In *Wikipedia*. En ligne. <https://fr.wikipedia.org/wiki/Diagramme_de_Sankey>. Consulté le 17 août 2017.

Wikipedia. 2017. «Comparison of JavaScript charting frameworks». In *Wikipedia*. En ligne. <https://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_frameworks>. Consulté le 15 août 2017.

ANNEXE I: Diagramme du modèle logique de la base de données



ANNEXE II: Fichier texte pour générer le modèle logique (avec l'outil PlantUML

```
@startuml DatabaseUML.png
left to right direction
skinparam linetype polyline
skinparam defaultFontSize 14

title DATABASE Project2

object "tbl_LCI" as lci
       lci : <u>id: INTEGER /'PK'/
       lci : <u>resultSetId: INTEGER /' fk avec resultSet '/
       lci : <u>processId: INTEGER /' fk avec processTable '/
       lci : <u>unit: INTEGER /' fk avec processTable '/
       lci : <u>flowId: INTEGER /' fk avec flowTable '/
       lci : totalQuantity: DECIMAL
       lci : processContribution: DECIMAL
       lci : processScalar: DECIMAL

object "tbl_LCIA" as lcia
       lcia : <u>id: INTEGER /'PK'/
       lcia : <u>resultSetId: INTEGER /' fk avec resultSet '/
       lcia : <u>impactCategoryId: INTEGER /' fk avec impactCategoryTable '/
       lcia : <u>processId: INTEGER /' fk avec processTable '/
       lcia : totalImpact: DECIMAL
       lcia : processContribution: DECIMAL
       lcia : unit: STRING

object "tbl_ULCIA" as ulcia
       ulcia : <u>id: INTEGER /'PK'/
       ulcia : <u>resultSetId: INTEGER /' fk avec resultSet '/
       ulcia : <u>impactCategoryId: INTEGER /' fk avec impactCategoryTable '/
       ulcia : <u>Min: INTEGER /' fk avec impactCategoryTable '/
       ulcia : <u>Max: INTEGER /' fk avec impactCategoryTable '/
       ulcia : STD: DECIMAL
       ulcia : Mean: DECIMAL

object "tbl_SLCIA" as slcia
       slcia : <u>id: INTEGER /'PK'/
       slcia : <u>resultSetId: INTEGER /' fk avec resultSet '/
       slcia : <u>flowId: INTEGER /' fk avec flowTable '/
       slcia : <u>processId: INTEGER /' fk avec processTable '/
       slcia : <u>exchangeId: INTEGER /' fk avec exchange '/
       slcia : inputType: ENUM ('e', 'i', 'q')
       slcia : sensitivity: DECIMAL /' un calcul dépendant de ROCC '/
       slcia : ROCC: DECIMAL
```

```
object "tbl_LCI_result_sets" as rs
       rs : <u>id: INTEGER /'PK'/
       rs : timestamp: TIMESTAMP

object "tbl_processes" as process
       process : <u>id: INTEGER

object "tbl_impact_categories" as ict
       ict : <u>id: INTEGER

object "tbl_flows" as ft
       ft : <u>id: INTEGER
       ft : type: ENUM('e','i','q')

object "tbl_unit" as u
       u : <u>id: INTEGER

object "tbl_exchanges" as e
       e : <u>id: INTEGER

lci "*" -- "1" process : Réfère à
lci "*" -- "1" u : Réfère à
lci "*" -- "1" ft : Réfère à
lcia "*" -- "1" process : Réfère à
lcia "*" -- "1" ict : Réfère à
ulcia "*" -- "1" ict : Réfère à
slcia "*" -- "1" ft : Réfère à
slcia "*" -- "1" process : Réfère à
slcia "*" -- "1" e : Réfère à
rs "1" -- "*" lci : Réfère à
rs "1" -- "*" lcia : Réfère à
rs "1" -- "*" ulcia : Réfère à
rs "1" -- "*" slcia : Réfère à

legend right
**PFE** Été 2017
Jean-Pierre Bertrand-Dorion
Aravinthan Sivaneswaran
Cydrick Trudel
Daniel Enachescu
end legend

Left footer   PFE UBUBI
@enduml
```

ANNEXE III: Exemple de fichier Excel XML simple

```
<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:o="urn:schemas-microsoft-com:office:office"
 xmlns:x="urn:schemas-microsoft-com:office:excel"
 xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:html="http://www.w3.org/TR/REC-html40">
 <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
  <Author>Cydrick Trudel</Author>
  <LastAuthor>Cydrick Trudel</LastAuthor>
  <Created>2017-08-14T20:41:01Z</Created>
  <Version>16.00</Version>
 </DocumentProperties>
 <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
  <AllowPNG/>
 </OfficeDocumentSettings>
 <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
  <WindowHeight>13785</WindowHeight>
  <WindowWidth>28800</WindowWidth>
  <WindowTopX>0</WindowTopX>
  <WindowTopY>0</WindowTopY>
  <ProtectStructure>False</ProtectStructure>
  <ProtectWindows>False</ProtectWindows>
 </ExcelWorkbook>
 <Styles>
  <Style ss:ID="Default" ss:Name="Normal">
   <Alignment ss:Vertical="Bottom"/>
   <Borders/>
   <Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="11"
ss:Color="#000000"/>
   <Interior/>
   <NumberFormat/>
   <Protection/>
  </Style>
  <Style ss:ID="s62">
   <NumberFormat ss:Format="Fixed"/>
  </Style>
 </Styles>
 <Worksheet ss:Name="Feuille1">
  <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="4"
x:FullColumns="1"
   x:FullRows="1" ss:DefaultRowHeight="15">
   <Row>
    <Cell><Data ss:Type="String">Colonne1</Data></Cell>
    <Cell><Data ss:Type="String">Colonne2</Data></Cell>
    <Cell><Data ss:Type="String">Colonne3</Data></Cell>
   </Row>
   <Row>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">1</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">2</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">3</Data></Cell>
```

```
   </Row>
   <Row>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">4</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">5</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">6</Data></Cell>
   </Row>
   <Row>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">7</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">8</Data></Cell>
    <Cell ss:StyleID="s62"><Data ss:Type="Number">9</Data></Cell>
   </Row>
  </Table>
  <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
   <PageSetup>
    <Header x:Margin="0.3"/>
    <Footer x:Margin="0.3"/>
    <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7"
x:Top="0.75"/>
   </PageSetup>
   <Selected/>
   <Panes>
    <Pane>
     <Number>3</Number>
     <ActiveRow>15</ActiveRow>
     <ActiveCol>7</ActiveCol>
    </Pane>
   </Panes>
   <ProtectObjects>False</ProtectObjects>
   <ProtectScenarios>False</ProtectScenarios>
  </WorksheetOptions>
 </Worksheet>
</Workbook>
```

ANNEXE IV: Premier gabarit de fichier Excel XML pour Apache Xalan

```
<?xml version="1.0"?>
<!--
Here is the structure expected by this transformation sheet:

report
      metadata
            creation-date (Date in GMT in the following format: 2017-06-
15T17:30:02Z)
      data
            lci
                  item
                        substance
                        category
                        unit
                        scalar
            lcia
                  categories
                        item
                              name
                              total-impact
                              unit
                  elements
                        item
                              name
                              value (all values, as many as there is
categories)
            uncertainty-lcia
                  item
                        name
                        unit
                        value
                        std
                        mean
                        min
                        max
            sensitivity-lcia
                  item
                        input-type
                        flow
                        process
                        sensitivity
-->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
      <?mso-application progid="Excel.Sheet"?>
      <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
       xmlns:o="urn:schemas-microsoft-com:office:office"
       xmlns:x="urn:schemas-microsoft-com:office:excel"
       xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
       xmlns:html="http://www.w3.org/TR/REC-html40">
```

```
       <DocumentProperties xmlns="urn:sc  hemas-microsoft-
com:office:office">
        <Author>UBUBI</Author>
        <LastAuthor>UBUBI</LastAuthor>
        <Created><xsl:value-of select="report/metadata/creation-
date"/></Created>
        <LastSaved><xsl:value-of select="report/metadata/creation-
date"/></LastSaved>
        <Version>16.00</Version>
       </DocumentProperties>
       <Styles>
        <Style ss:ID="Default" ss:Name="Normal">
         <Alignment ss:Vertical="Bottom"/>
         <Borders/>
         <Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="11"
ss:Color="#000000"/>
         <Interior/>
         <NumberFormat/>
         <Protection/>
        </Style>
        <Style ss:ID="s62">
         <NumberFormat ss:Format="Scientific"/>
        </Style>
        <Style ss:ID="s63">
         <NumberFormat ss:Format="Percent"/>
        </Style>
       </Styles>
       <Worksheet ss:Name="Inventaires">
        <Names>
         <NamedRange ss:Name="_FilterDatabase"
ss:RefersTo="=Inventaires!R1C1:R4C3"
            ss:Hidden="1"/>
        </Names>
        <Table>
         <Column ss:AutoFitWidth="0" ss:Width="266.25"/>
         <Column ss:AutoFitWidth="0" ss:Width="108.75" ss:Span="2"/>
         <Row ss:AutoFitHeight="0">
            <Cell><Data ss:Type="String">Substance</Data><NamedCell
              ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Catégorie
complète</Data><NamedCell
              ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Unité</Data><NamedCell
ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Valeur</Data></Cell>
         </Row>
         <xsl:for-each select="report/data/lci/item">
                  <Row ss:AutoFitHeight="0">
                        <Cell><Data ss:Type="String"><xsl:value-of
select="substance" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
                        <Cell><Data ss:Type="String"><xsl:value-of
select="category" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
                        <Cell><Data ss:Type="String"><xsl:value-of
select="unit" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
```

```
                        <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="scalar" /></Data></Cell>
                    </Row>
          </xsl:for-each>
         </Table>
         <AutoFilter x:Range="R1C1:R4C3" xmlns="urn:schemas-microsoft-
com:office:excel">
         </AutoFilter>
        </Worksheet>
             <Worksheet ss:Name="Analyse de contribution">
         <Names>
          <NamedRange ss:Name="_FilterDatabase"
             ss:RefersTo="='Analyse de contribution'!R1C1:R7C1"
ss:Hidden="1"/>
         </Names>
         <Table>
          <Column ss:AutoFitWidth="0" ss:Width="135"/>
          <Column ss:AutoFitWidth="0" ss:Width="82.5" ss:Span="4"/>
          <Row ss:AutoFitHeight="0">
               <Cell />
                  <xsl:for-each
select="report/data/lcia/categories/item">
                        <Cell><Data ss:Type="String"><xsl:value-of
select="total-impact" /> <xsl:value-of select="unit" /></Data></Cell>
                  </xsl:for-each>
          </Row>
          <Row ss:AutoFitHeight="0">
               <Cell />
                  <xsl:for-each
select="report/data/lcia/categories/item">
                        <Cell><Data ss:Type="String"><xsl:value-of
select="name" /></Data></Cell>
                  </xsl:for-each>
          </Row>
             <xsl:for-each select="report/data/lcia/elements/item">
                  <Row ss:AutoFitHeight="0">
                        <Cell><Data ss:Type="String"><xsl:value-of
select="name" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
                        <xsl:for-each select="value">
                              <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="." /></Data></Cell>
                        </xsl:for-each>
                  </Row>
             </xsl:for-each>
         </Table>
         <AutoFilter x:Range="R1C1:R7C1" xmlns="urn:schemas-microsoft-
com:office:excel">
         </AutoFilter>
        </Worksheet>
        <Worksheet ss:Name="LCIA non comparatifs-tabulaire">
         <Names>
          <NamedRange ss:Name="_FilterDatabase"
             ss:RefersTo="='LCIA non comparatifs-tabulaire'!R1C1:R5C2"
ss:Hidden="1"/>
```

```
        </Names>
        <Table>
         <Column ss:AutoFitWidth="0" ss:Width="135" ss:Span="1"/>
         <Column ss:Index="3" ss:AutoFitWidth="0" ss:Width="82.5"
ss:Span="7"/>
         <Column ss:Index="14" ss:AutoFitWidth="0" ss:Width="82.5"/>
         <Row ss:AutoFitHeight="0">
            <Cell><Data ss:Type="String">Catégorie
d'impact</Data><NamedCell
              ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Unité</Data><NamedCell
ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Valeur</Data></Cell>
            <Cell><Data ss:Type="String">SD</Data></Cell>
            <Cell><Data ss:Type="String">Moyenne</Data></Cell>
            <Cell><Data ss:Type="String">Min</Data></Cell>
            <Cell><Data ss:Type="String">Max</Data></Cell>
         </Row>
         <xsl:for-each select="report/data/uncertainty-lcia/item">
               <Row ss:AutoFitHeight="0">
                  <Cell><Data ss:Type="String"><xsl:value-of
select="name" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
                  <Cell><Data ss:Type="String"><xsl:value-of
select="unit" /></Data><NamedCell ss:Name="_FilterDatabase"/></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="value" /></Data></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="std" /></Data></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="mean" /></Data></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="min" /></Data></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="max" /></Data></Cell>
               </Row>
         </xsl:for-each>
        </Table>
        <AutoFilter x:Range="R1C1:R5C2" xmlns="urn:schemas-microsoft-
com:office:excel">
        </AutoFilter>
       </Worksheet>
       <Worksheet ss:Name="Sensibilite LCIA">
        <Names>
         <NamedRange ss:Name="_FilterDatabase"
            ss:RefersTo="='Sensibilite LCIA'!R1C1:R1C4" ss:Hidden="1"/>
        </Names>
        <Table>
         <Column ss:AutoFitWidth="0" ss:Width="135" ss:Span="4"/>
         <Row ss:AutoFitHeight="0">
            <Cell><Data ss:Type="String">Type d'entrée</Data><NamedCell
              ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Flow</Data><NamedCell
ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Processus</Data><NamedCell
```

```
              ss:Name="_FilterDatabase"/></Cell>
            <Cell><Data ss:Type="String">Sensibilité</Data></Cell>
          </Row>
          <xsl:for-each select="report/data/sensitivity-lcia/item">
               <Row ss:AutoFitHeight="0">
                  <Cell><Data ss:Type="String"><xsl:value-of
select="input-type" /></Data></Cell>
                  <Cell><Data ss:Type="String"><xsl:value-of
select="flow" /></Data></Cell>
                  <Cell><Data ss:Type="String"><xsl:value-of
select="process" /></Data></Cell>
                  <Cell ss:StyleID="s62"><Data
ss:Type="Number"><xsl:value-of select="sensitivity" /></Data></Cell>
                </Row>
          </xsl:for-each>
         </Table>
         <AutoFilter x:Range="R1C1:R1C4" xmlns="urn:schemas-microsoft-
com:office:excel">
         </AutoFilter>
        </Worksheet>
       </Workbook>
  </xsl:template>
</xsl:stylesheet>
```

ANNEXE V: Comparaison des syntaxes d'Apache Xalan et d'Apache Velocity

Apache Xalan

```
<xsl:for-each select="report/data/contribution-analysis/processes/item">
	<Row>
		<Cell ss:StyleID="s66">
			<Data ss:Type="String">
				<xsl:value-of select="name" />
			</Data>
			<NamedCell ss:Name="_FilterDatabase" />
		</Cell>
		<Cell ss:StyleID="s67">
			<Data ss:Type="String">
				<xsl:value-of select="category" />
			</Data>
			<NamedCell ss:Name="_FilterDatabase" />
		</Cell>
		<xsl:for-each select="value">
			<Cell ss:StyleID="s16">
				<Data ss:Type="Number">
					<xsl:value-of select="." />
				</Data>
				<NamedCell ss:Name="_FilterDatabase" />
			</Cell>
		</xsl:for-each>
	</Row>
</xsl:for-each>
```

Apache Velocity (réorganisé pour plus de clarté)

```
#foreach($item in $ContributionAnalysis.Processes)
	<Row>
		<Cell ss:StyleID="s66">
			<Data ss:Type="String">
				${item.Name}
			</Data>
			<NamedCell ss:Name="_FilterDatabase" />
		</Cell>
		<Cell ss:StyleID="s67">
			<Data ss:Type="String">
				${item.Category}
			</Data>
			<NamedCell ss:Name="_FilterDatabase" />
		</Cell>
		#foreach($itemValue in $item.OrderedValues)
			<Cell ss:StyleID="s73">
				<Data ss:Type="Number">
					${itemValue}
				</Data>
				<NamedCell ss:Name="_FilterDatabase" />
			</Cell>
		#end
	</Row>
#end
```

ANNEXE VI: Gabarit final adaptée pour Apache Velocity

```
#*
Here is the structure expected by this template:

Metadata (map)
      CreationDate (Date in GMT in the following format: 2017-06-
15T17:30:02Z)
Inventory (list of map)
      Substance
      FlowCategory
      ProcessName
      ProcessCategory
      Unit
      Scalar
CharacterizedInventory (list of map)
      Substance
      FlowCategory
      Unit
      GroupedScalar
LciByCategory (map of map)
      FlowCategories (list of map)
            Name
      Flows (list of map)
            Substance
            Unit
            OrderedValues (list of all values, as many as there is
FlowCategories)
ContributionAnalysis (map of map)
      ImpactCategories (list of map)
            Name
            Total-impact
            Unit
      Processes (list of map)
            Name
            Category
            OrderedValues (list of all values, as many as there is
ImpactCategories)
LciaNonComparative (list of map)
      Name
      Unit
      Std
      Mean
      Min
      Max
SensitivityLcia (list of map)
      Input-Type
      Flow
```

```
      FlowCategory
      Process
      Process-category
      Sensitivity

*#
<?xml version="1.0" encoding="UTF-8"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:html="http://www.w3.org/TR/REC-html40" xmlns:o="urn:schemas-
microsoft-com:office:office" xmlns:ss="urn:schemas-microsoft-
com:office:spreadsheet" xmlns:x="urn:schemas-microsoft-com:office:excel">
      <DocumentProperties xmlns="urn:schemas-microsoft-
com:office:office">
            <Author>UBUBI</Author>
            <LastAuthor>UBUBI</LastAuthor>
        <Created>${Metadata.CreationDate}</Created>
            <Version>16.00</Version>
      </DocumentProperties>
      <OfficeDocumentSettings xmlns="urn:schemas-microsoft-
com:office:office">
            <AllowPNG />
      </OfficeDocumentSettings>
      <Styles>
            <Style ss:ID="Default" ss:Name="Normal">
                  <Alignment ss:Vertical="Bottom" />
                  <Borders />
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" />
                  <Interior />
                  <NumberFormat />
                  <Protection />
            </Style>
            <Style ss:ID="s16">
                  <NumberFormat ss:Format="0.00E+00;\-0.00E+00;;@" />
            </Style>
        <Style ss:ID="s20" ss:Name="Percent">
            <NumberFormat ss:Format="0%"/>
        </Style>
            <Style ss:ID="s65">
                  <Borders>
                        <Border ss:Position="Bottom"
ss:LineStyle="Continuous" ss:Weight="2" />
                  </Borders>
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" ss:Bold="1" />
            </Style>
            <Style ss:ID="s66">
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" ss:Bold="1" />
            </Style>
            <Style ss:ID="s67">
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" />
```

```
            </Style>
        <Style ss:ID="s73" ss:Parent="s20">
            <Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="11"
ss:Color="#000000"/>
            <NumberFormat ss:Format="0.0000000%;\-0.0000000%;;@"/>
        </Style>
            <Style ss:ID="s77">
                  <Borders>
                        <Border ss:Position="Top"
ss:LineStyle="Continuous" ss:Weight="1" />
                  </Borders>
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" ss:Bold="1" />
            </Style>
            <Style ss:ID="s78">
                  <Borders>
                        <Border ss:Position="Top"
ss:LineStyle="Continuous" ss:Weight="1" />
                  </Borders>
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" />
            </Style>
            <Style ss:ID="s79">
                  <Borders>
                        <Border ss:Position="Top"
ss:LineStyle="Continuous" ss:Weight="1" />
                  </Borders>
                  <Font ss:FontName="Calibri" x:Family="Swiss"
ss:Size="11" ss:Color="#000000" ss:Bold="1" />
                  <NumberFormat ss:Format="0.00E+00;\-0.00E+00;;@" />
            </Style>
      </Styles>
      <Worksheet ss:Name="Inventory">
            <Names>
                  <NamedRange ss:Name="_FilterDatabase"
ss:RefersTo="=Inventory!R1C1:R1C4" ss:Hidden="1" />
            </Names>
            <Table>
                  <Column ss:AutoFitWidth="0" ss:Width="266.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="213.75" />
                  <Column ss:AutoFitWidth="0" ss:Width="345" />
                  <Column ss:AutoFitWidth="0" ss:Width="213.75" />
                  <Column ss:AutoFitWidth="0" ss:Width="56.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="108.75" />
                  <Row>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Substance</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Flow Category</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
```

```
                              <Data ss:Type="String">Process</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Process
Category</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Unit</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Value</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                  </Row>
                  #foreach($item in $Inventory)
                  <Row>
                        <Cell>
                              <Data
ss:Type="String">${item.Substance}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.FlowCategory}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.ProcessName}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.ProcessCategory}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data ss:Type="String">${item.Unit}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data
ss:Type="Number">${item.Scalar}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                  </Row>
                  #end
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-
com:office:excel">
                  <FreezePanes />
```

```
                  <FrozenNoSplit />
                  <SplitHorizontal>1</SplitHorizontal>
                  <TopRowBottomPane>1</TopRowBottomPane>
            </WorksheetOptions>
            <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R2C6" />
      </Worksheet>
      <Worksheet ss:Name="LCI by Category">
            #set( $LciByCategoryNumberOfColumns =
$LciByCategory.ImpactCategories.size() + 2 )
            <Names>
                  <NamedRange ss:Name="_FilterDatabase"
ss:RefersTo="='LCI by
Category'!R1C1:R1C#evaluate($LciByCategoryNumberOfColumns)" ss:Hidden="1"
/>
            </Names>
            <Table>
                  <Column ss:AutoFitWidth="0" ss:Width="161.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="56.25" />
                  #foreach($item in $LciByCategory.FlowCategories)
                  <Column ss:AutoFitWidth="0" ss:Width="135" />
                  #end
                  <Row>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Flow</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Unit</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #foreach($item in $LciByCategory.FlowCategories)
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">${item.Name}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
                  </Row>
                  #foreach($item in $LciByCategory.Flows)
                  <Row>
                        <Cell ss:StyleID="s67">
                              <Data ss:Type="String">${item.Name}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s67">
                              <Data ss:Type="String">${item.Unit}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #foreach($itemValue in $item.OrderedValues)
                        <Cell ss:StyleID="s16">
                              <Data ss:Type="Number">${itemValue}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
```

```
                  </Row>
                  #end
                  <Row>
                        <Cell ss:StyleID="s77">
                              <Data ss:Type="String">Total</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s77" />
                        #foreach($item in $LciByCategory.FlowCategories)
                        <Cell ss:StyleID="s79" ss:Formula="=SUBTOTAL(9,
R2C:R[-1]C)">
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
                  </Row>
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-
com:office:excel">
                  <FreezePanes />
                  <FrozenNoSplit />
                  <SplitHorizontal>1</SplitHorizontal>
                  <TopRowBottomPane>1</TopRowBottomPane>
            </WorksheetOptions>
            <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R1C#evaluate($LciByCategoryNumberOfColumns)" />
      </Worksheet>
      <Worksheet ss:Name="Characterized Inventory">
            <Names>
                  <NamedRange ss:Name="_FilterDatabase"
ss:RefersTo="='Characterized Inventory'!R1C1:R1C4" ss:Hidden="1" />
            </Names>
            <Table>
                  <Column ss:AutoFitWidth="0" ss:Width="266.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="213.75" />
                  <Column ss:AutoFitWidth="0" ss:Width="56.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="108.75" />
                  <Row>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Substance</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Flow Category</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Unit</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Value</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                  </Row>
```

```
                  #foreach($item in $CharacterizedInventory)
                  <Row>
                        <Cell>
                              <Data
ss:Type="String">${item.Substance}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.FlowCategory}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell>
                              <Data ss:Type="String">${item.Unit}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data
ss:Type="Number">${item.GroupedScalar}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                  </Row>
                  #end
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-
com:office:excel">
                  <FreezePanes />
                  <FrozenNoSplit />
                  <SplitHorizontal>1</SplitHorizontal>
                  <TopRowBottomPane>1</TopRowBottomPane>
            </WorksheetOptions>
            <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R2C4" />
      </Worksheet>
      <Worksheet ss:Name="Contribution Analysis">
            #set( $ContributionAnalysisNumberOfColumns =
$ContributionAnalysis.ImpactCategories.size() + 2 )
            <Names>
                  <NamedRange ss:Name="_FilterDatabase"
ss:RefersTo="='Contribution
Analysis'!R1C1:R1C#evaluate($ContributionAnalysisNumberOfColumns)"
ss:Hidden="1" />
            </Names>
            <Table>
                  <Column ss:AutoFitWidth="0" ss:Width="161.25" />
                  <Column ss:AutoFitWidth="0" ss:Width="240" />
                  #foreach($item in
$ContributionAnalysis.ImpactCategories)
                  <Column ss:AutoFitWidth="0" ss:Width="135" />
                  #end
                  <Row>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Process</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
```

```
                        </Cell>
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">Process
Category</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #foreach($item in
$ContributionAnalysis.ImpactCategories)
                        <Cell ss:StyleID="s65">
                              <Data ss:Type="String">
                                    ${item.Name}(${item.Unit})
                              </Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
                  </Row>
                  #foreach($item in $ContributionAnalysis.Processes)
                  <Row>
                        <Cell ss:StyleID="s66">
                              <Data ss:Type="String">${item.Name}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s67">
                              <Data
ss:Type="String">${item.Category}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #foreach($itemValue in $item.OrderedValues)
                        <Cell ss:StyleID="s73">
                              <Data ss:Type="Number">${itemValue}</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
                  </Row>
                  #end
                  <Row>
                        <Cell ss:StyleID="s77">
                              <Data ss:Type="String">Total</Data>
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        <Cell ss:StyleID="s77" />
                        #foreach($item in
$ContributionAnalysis.ImpactCategories)
                        <Cell ss:StyleID="s79" ss:Formula="=SUBTOTAL(9,
R2C:R[-1]C)">
                              <NamedCell ss:Name="_FilterDatabase" />
                        </Cell>
                        #end
                  </Row>
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-
com:office:excel">
                  <FreezePanes />
                  <FrozenNoSplit />
```

```
                <SplitHorizontal>1</SplitHorizontal>
                <TopRowBottomPane>1</TopRowBottomPane>
            </WorksheetOptions>
            <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R1C#evaluate($ContributionAnalysisNumberOfColumns)" />
      </Worksheet>
    <Worksheet ss:Name="LCIA Non-Comparative Tabulary">
        <Names>
            <NamedRange ss:Name="_FilterDatabase" ss:RefersTo="='LCIA
Non-Comparative Tabulary'!R1C1:R1C7" ss:Hidden="1" />
        </Names>
        <Table>
            <Column ss:AutoFitWidth="0" ss:Width="213.75" ss:Span="1" />
            <Column ss:Index="3" ss:AutoFitWidth="0" ss:Width="82.5"
ss:Span="4" />
            <Row>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Impact Category</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Unit</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Value</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">SD</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Mean</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Min</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Max</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
            </Row>
                #foreach($item in $LciaNonComparative)
                <Row>
                      <Cell>
                            <Data ss:Type="String">${item.Name}</Data>
                      </Cell>
                      <Cell>
                            <Data ss:Type="String">${item.Unit}</Data>
                      </Cell>
                      <Cell ss:StyleID="s16">
```

```
                              <Data ss:Type="Number">${item.Mean}</Data>
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data ss:Type="Number">${item.Std}</Data>
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data ss:Type="Number">${item.Mean}</Data>
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data ss:Type="Number">${item.Min}</Data>
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data ss:Type="Number">${item.Max}</Data>
                        </Cell>
                  </Row>
            #end
        </Table>
        <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
            <FreezePanes />
            <FrozenNoSplit />
            <SplitHorizontal>1</SplitHorizontal>
            <TopRowBottomPane>1</TopRowBottomPane>
        </WorksheetOptions>
        <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R1C7" />
    </Worksheet>
    <Worksheet ss:Name="LCIA Sensitivity">
        <Names>
            <NamedRange ss:Name="_FilterDatabase" ss:RefersTo="='LCIA
Sensitivity'!R1C1:R1C6" ss:Hidden="1" />
        </Names>
        <Table>
            <Column ss:AutoFitWidth="0" ss:Width="135" />
            <Column ss:AutoFitWidth="0" ss:Width="345" />
            <Column ss:AutoFitWidth="0" ss:Width="213.75" />
            <Column ss:AutoFitWidth="0" ss:Width="345" />
            <Column ss:AutoFitWidth="0" ss:Width="213.75" />
            <Column ss:AutoFitWidth="0" ss:Width="82.5" />
            <Row>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Input Type</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Flow</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Flow Category</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Process</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
```

```
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Process Category</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
                <Cell ss:StyleID="s65">
                    <Data ss:Type="String">Sensitivity</Data>
                    <NamedCell ss:Name="_FilterDatabase" />
                </Cell>
            </Row>
                  #foreach($item in $SensitivityLcia)
                  <Row>
                        <Cell>
                              <Data
ss:Type="String">${item.InputType}</Data>
                        </Cell>
                        <Cell>
                              <Data ss:Type="String">${item.Flow}</Data>
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.FlowCategory}</Data>
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.Process}</Data>
                        </Cell>
                        <Cell>
                              <Data
ss:Type="String">${item.ProcessCategory}</Data>
                        </Cell>
                        <Cell ss:StyleID="s16">
                              <Data
ss:Type="Number">${item.Sensitivity}</Data>
                        </Cell>
                  </Row>
            #end
        </Table>
        <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
            <FreezePanes />
            <FrozenNoSplit />
            <SplitHorizontal>1</SplitHorizontal>
            <TopRowBottomPane>1</TopRowBottomPane>
        </WorksheetOptions>
        <AutoFilter xmlns="urn:schemas-microsoft-com:office:excel"
x:Range="R1C1:R1C6" />
    </Worksheet>
</Workbook>
```

ANNEXE VII: Diagramme de classe du module de transformation Velocity