[Connecteur Coda] – Compte Rendu de l’analyse du code source

**Liste de Diffusion**

| Nom | Fonction | RACI |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  | *nommément si ils sont peu nombreux, sinon leur entité, comme par exemple "CDEV", "TMA IBM à tel endroit, …* |  |
|  | *Si un éditeur est concerné par le PGCL car il doit se conformer à certaines obligations qui y sont décrites, donner son nom et sa fonction "Editeur"* | *I* |
|  | *Pas la peine d'ajouter ici le RGC car c'est le rédacteur* |  |

**Evolutions du document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version  Si le document est géré sous C'You, il n'est pas nécessaire d'indiquer ici les mini ajustements (donnant lieu à une version automatique de C'You) qui sont plutôt à préciser dans la zone commentaire à chaque archivage dans C'You | Date de Validation  Pas celle de parution de la version, mais celle d'acceptation par l'acteur de la colonne Validation | Rédacteur  Par défaut le RGC, mais cela peut être aussi le RGCT, le RPA, le RA si il n'est pas aussi RGC | Nature des modifications | Validation  Qui valide le document  RGCT au moins  RA si ce n'est pas le RGC  RPA possiblement |
| V1.0 | 06/03/2023 | Mohamed HAMMA | Version initiale | Othmane MESKINI |
|  |  |  |  |  |

Table des matières

[I. Introduction 3](#_Toc129959857)

[II. Architecture technique et logicielle 3](#_Toc129959858)

[III. Problème de performances constaté 4](#_Toc129959859)

[IV. Les traitements 6](#_Toc129959860)

[1. Description des traitements du batch 6](#_Toc129959861)

[2. Schéma LaunchConnecteur.run() 7](#_Toc129959862)

[3. Schéma détaillé des méthodes 8](#_Toc129959863)

[4. Schéma détaillé ImputationFichierECN 9](#_Toc129959864)

[V. Description détaillée des traitements 10](#_Toc129959865)

[*1.* *LauncherConecteur.run* 10](#_Toc129959866)

[2. Traitement Particulier 10](#_Toc129959867)

[3. FileUtils.getOldestFile 11](#_Toc129959868)

[4. saxDecoupageEcnParser.deconcatener 11](#_Toc129959869)

[5. saxDecoupageEcnParser.supprimerListeFichiers 11](#_Toc129959870)

[*6.* *LauncherConecteur.moveFile* 11](#_Toc129959871)

[7. imputationFichierEcn.imputation 11](#_Toc129959872)

[8. imputationFichierEcn.imputationFichierPlat 12](#_Toc129959873)

[9. flatStreamManager.processFile 12](#_Toc129959874)

[10. flatStreamManager.addNewTraitement 13](#_Toc129959875)

[11. FlatStreamManager.refreshTraitement 13](#_Toc129959876)

[12. traitementProcessor.refresh 13](#_Toc129959877)

[13. flatStreamManager.addLigneMouvementComptable 14](#_Toc129959878)

[14. traitementProcessor.addLigneMouvementComptable 14](#_Toc129959879)

[15. traitementProcessor.flush 16](#_Toc129959880)

[16. traitementProcessor.flushLinkHead 16](#_Toc129959881)

[VI. Analyse Sonar du projet 17](#_Toc129959882)

[VII. Propositions et axes d’amélioration 17](#_Toc129959883)

[1. Bugs sonar 17](#_Toc129959884)

[2. Mettre à jour les librairies utilisées 18](#_Toc129959885)

[3. Utilisation de LinkedList 18](#_Toc129959886)

[4. Augmenter la taille de la mémoire tampon 18](#_Toc129959887)

[5. Récrire la partie lecture du fichier TXT et ces lignes 18](#_Toc129959888)

[6. Paralléliser les traitements 18](#_Toc129959889)

[7. Migrer le batch en spring batch 19](#_Toc129959890)

[8. Analyse de logs 19](#_Toc129959891)

# Introduction

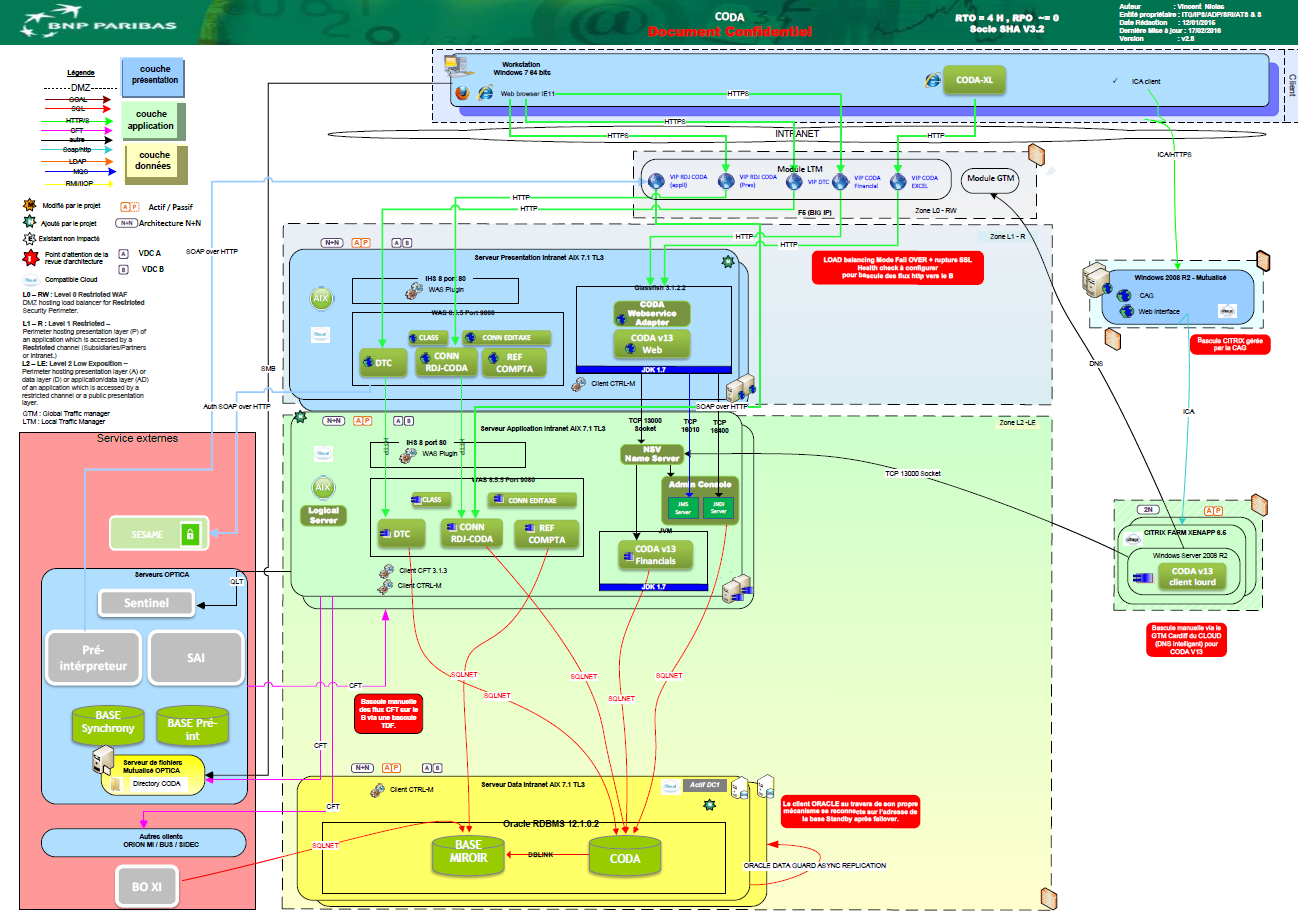
Ce document a pour objectif de décrire l’existant sur le batch connecteur Coda et proposer des améliorations pour les problèmes de performances constaté le 31/01/2013 (lenteurs lors de lancement du batch)

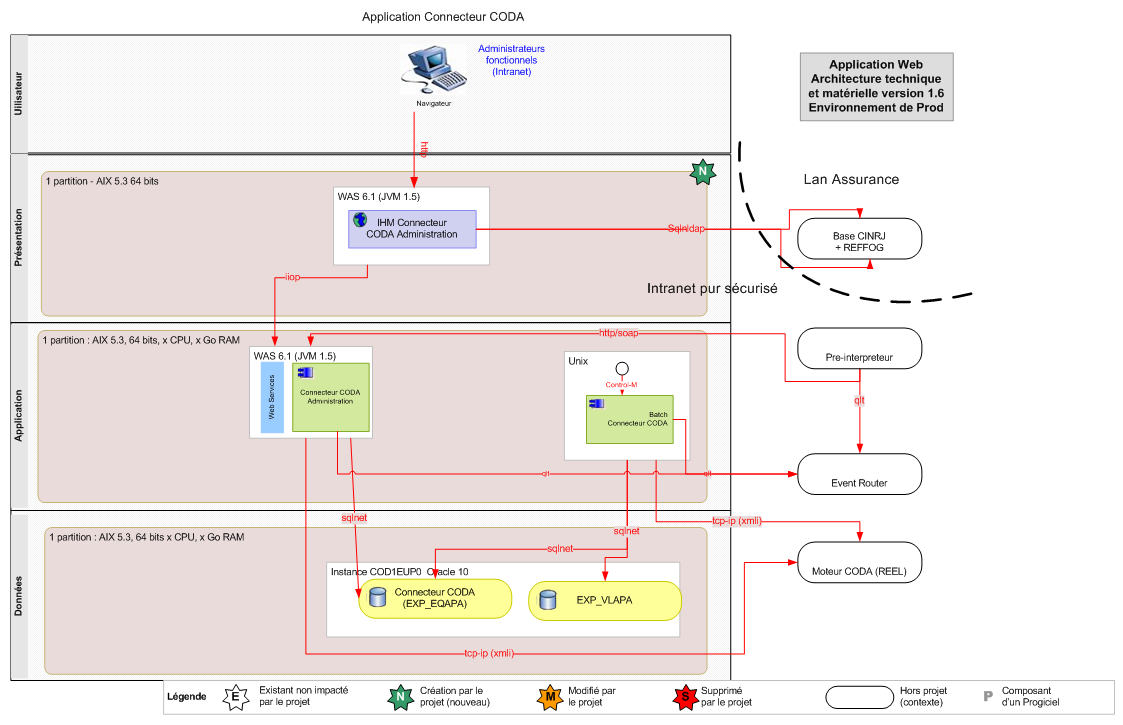
Il permet de restituer l’analyse effectué sur l’application connecteur CODA.

A noter que l’analyse a été effectuée sans exécution du bach en local

# Architecture technique et logicielle

Le connecteur CODA est responsable du chargement dans CODA Cible des écritures qui lui sont livrées par CFT en provenance de l’application PAEJA. ***L’application dispose d’une interface WEB pour son administration.***



**

Ci-dessous, un tableau des versions des principales Frameworks utilisés :

|  |  |  |
| --- | --- | --- |
| Version Framework | Année | version |
| Java | 2014 | 1.8 |
| Compilateur | 2011 | 1.7 |
| CINRJ | 2014 | 5.8 |
| SESAME authentification | 2015 | 2.0 |
| SESAME authorization | 2015 | 1.0 |
| SESAME identity | 2015 | 1.0 |
| SESAME app authentification | 2015 | 1.0 |
| HIBERNATE | 2014 | 4.2.16.Final |
| SPRING | 2014 | 4.1.2.RELEASE |
| HIBERNATE-JPA | 2010 | 1.0.0.Final |
| SPRING Security | 2014 | 3.2.5.RELEASE |
| STRUTS | 2006 | 1.2.9 |
| J2EE EL | 2011 | 2.2 |
| J2EE JSF | 2013 | 2.1.23 |
| LOG4J | 2012 | 1.2.17 |
| SLF4J | 2014 | 1.7.7 |
| HSQLDB | 2014 | 2.3.2 |

A noter que la version 5.8 de CINRJ utilise logj4j 1.2.15. Nous conseillons de migrer vers CINRJ version 5.10.7 ou supérieure, qui utilise des versions plus récentes de log4j.

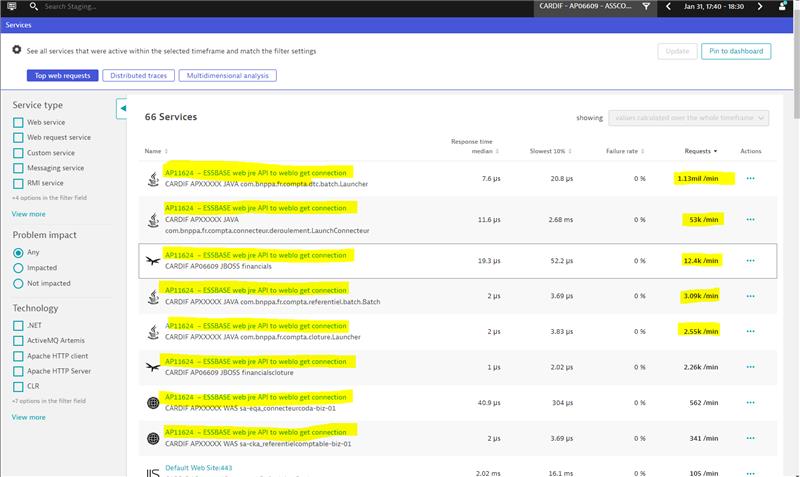
Concernant Hibernante JPA, la dernière version (1.0.1.Final) de l’artéfact utilisé (hibernate-jpa-2.0-api) date de juin 2011 (la version utilisée par l’application est la 1.0.0.Final) :

|  |
| --- |
| <dependency>  <groupId>org.hibernate.javax.persistence</groupId>  <artifactId>hibernate-jpa-2.0-api</artifactId>  <version>1.0.0.Final</version>  </dependency> |

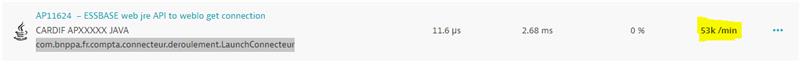
# Problème de performances constaté

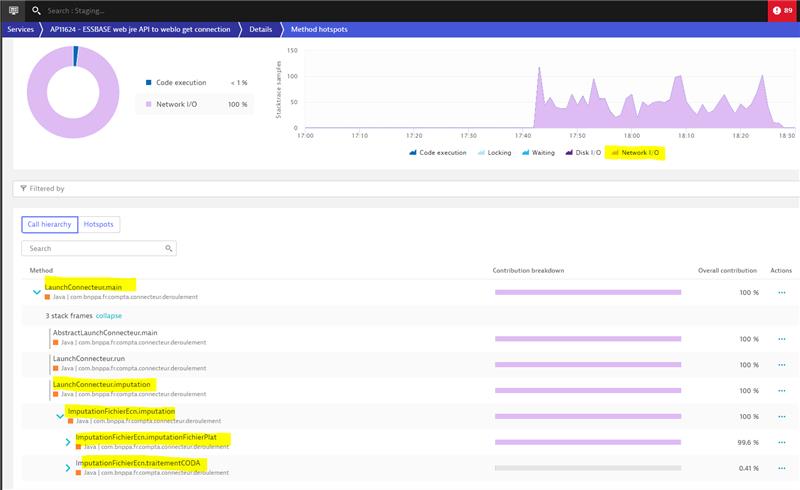
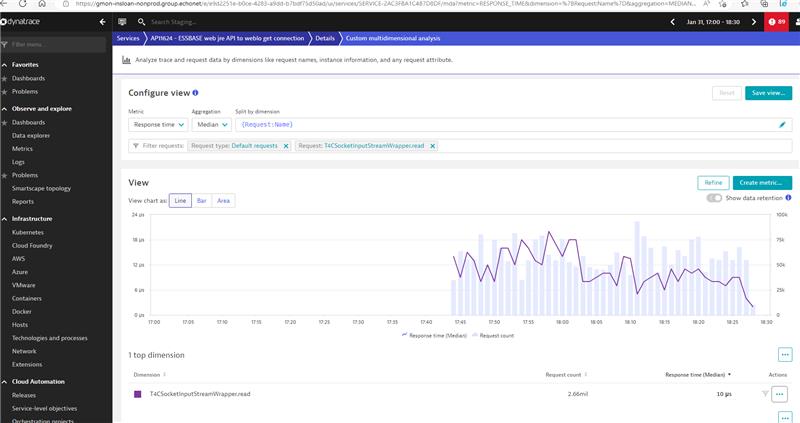
Une analyse Dynatrace a été effectuée pour expliquer le comportement du batch ci-dessous, le compte rendu de cette analyse:

Une forte sollicitation du service « **AP11624 - ESSBASE web jre API to weblo get connection** » a été constaté pour le batch connecteur coda (presque 53000 REQUESTS/MIN durant la durée du traitement du batch)



Et en zoomant sur le service en question :





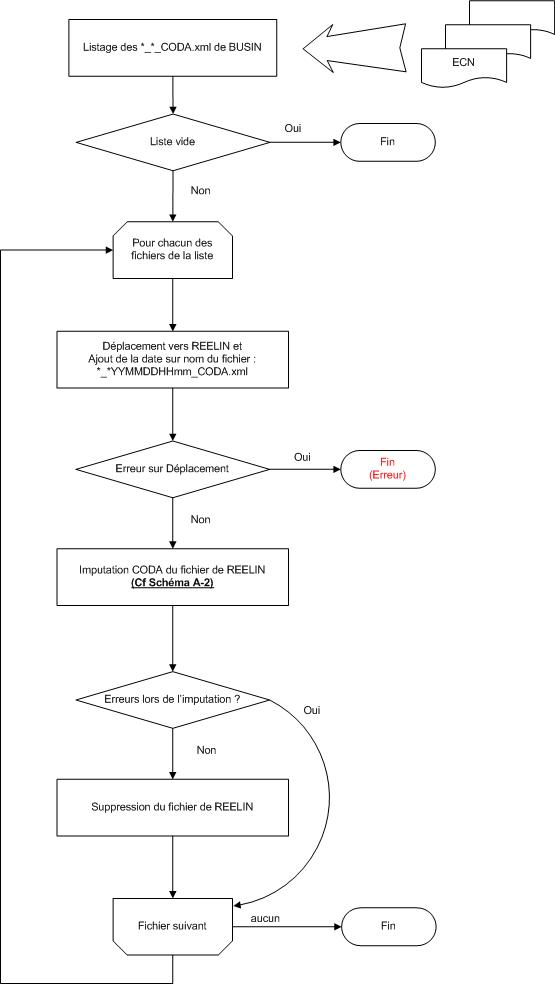
D’après cet analyse DynaTrace, la méthode qui altère les performances en grande partie est :

|  |
| --- |
| *public List imputationFichierPlat(File file) throws BusinessException* …. |

Et nous allons proposer des actions à mettre en place pour améliorer les performances de cette méthode.

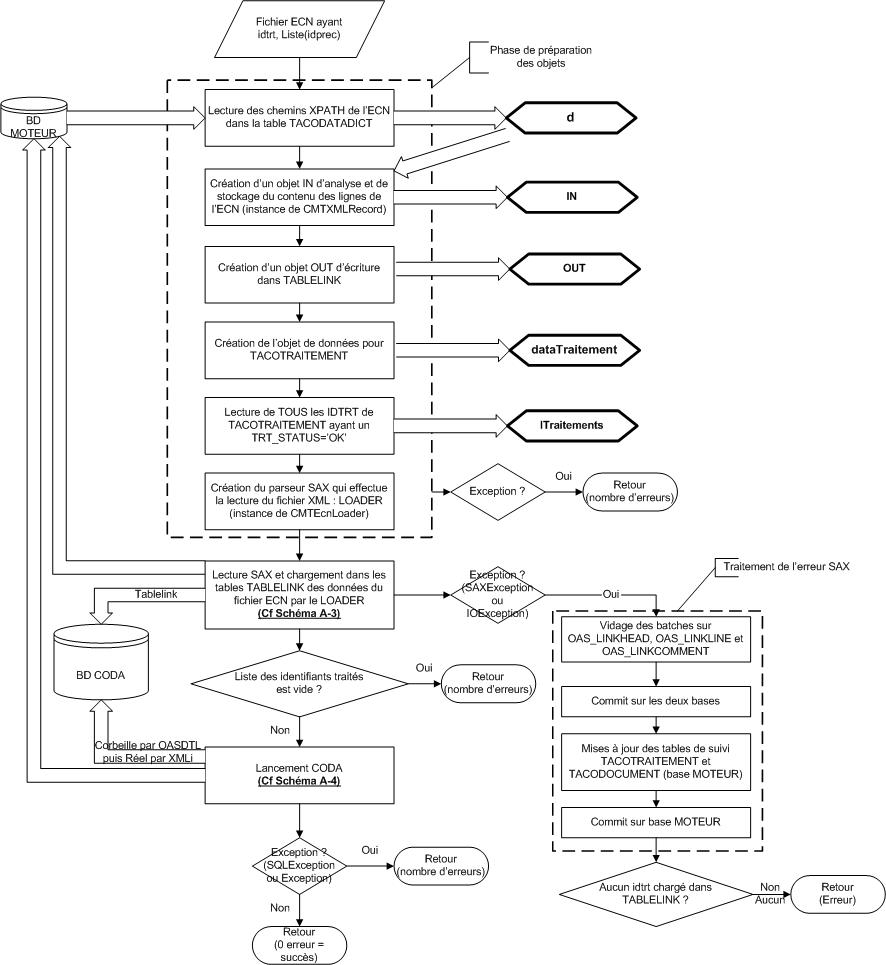
# Les traitements

## Description des traitements du batch



## Schéma LaunchConnecteur.run()

Ce schéma décrit le fonctionnement de la méthode LaunchConnecteur.run() :



Ci-après, des schémas détaillés des objets techniques qui illustrent l’enchainement du traitement du batch (lancement automatique):

## Schéma détaillé des méthodes

FileUtils (getOldestFile)

2

dirBusIn, pattern

LaunchConnecteu (run)

1

File

File

Si fichier se termne par xml

saxDecoupageEcnParser (supprimerListeFichiers)

4

saxDecoupageEcnParser (deconcatener)

3

Traitemnt Particulier

5

1 ou 0

En cas d’erreur dirBusIn dirReelIn

Suite Traitement

(Launchconnecteur)

6

LaunchConnecteur (getDirBusIn)

8

LaunchConnecteur (getDirReelIn)

7

dirBusIn, sAppli, dirReelIn

Liste Moved Files Si aucun fichier

LaunchConnecteur (moveFile)

9

LaunchConnecteur (imputation)

12

Logger l’information

10

Si il ya des fichiers et pour chaque fichier (moved file)

imputationFichierEcn (imputation)

13

LaunchConnecteur (moveFile)

File …

Traitemnt fichier

11

File

## Schéma détaillé ImputationFichierECN

imputationFichierEcn (imputation)

13

imputationFichierEcn (imputationFichierXML)

14

Suire Fichier XML

Liste Fichier TXT

FlatStreamManager

(processFile)

16

Liste traitements file

imputationFichierEcn (imputationFichierPlat)

15

Si pas de traitement Si traitement Coda traitements chaque ligne

Ligne assurance

Logger l’information

29

FlatStreamManager

addNewTraitement

17

imputationFichierEcn (traitementCODA)

27

17

LaunchCoda

(executeMiseEncorbeille)

28

1 ou 0

Traitements

Entête, idME

basculementCorbeille

(executeBasculement)

33

Liste Traitements Liste traitements KO

FlatStreamManager

addLigneMouvementComptable

19

FlatStreamManager

(refreshTraitement)

18

1 ou 0 Ligne assurance

launchCoda (executeBoucleBasculements)

29

Chaque traitement

traitementProcessor

addLigneMouvementComptable

21

LaunchCoda

(execute)

32

impuationFichierEcn (Suite Traitement)

31

Identificatoin, « 01 »

Traitemens,issupprime,null,comment IdCRE, previousIdentification oasDocument

isNewRupture plsrs params

traitementProcessor(refresh)

19

FlatStreamManager

flushLinkHead

25

FlatStreamManager

Flush

22

Traitement traitement

reporting

(createEndOk)

35

reporting

(createEndError)

34

linkHeadDTO

linkLines, linkComments 26

BD CODA

FlatStreamManager

flushLinkLineBlock

23

linkCommentsDTO

24

# Description détaillée des traitements

Nous allons dans ce chapitre décrire l’ensemble des traitements du batch.

## *LauncherConecteur.run*

Cette méthode prend en entrée, le paramètre :

|  |  |
| --- | --- |
| Type | Nom |
| *String* | *sAppli* |

Et renvoie :

|  |  |
| --- | --- |
| Type | Valeur |
| *int* | *Compteur des erreurs* |

Cette méthode effectue le traitement suivant :

* Le type de retour est fixé au départ à 0.
* La méthode récupère le fichier le plus ancien de la liste des fichiers.
* Elle effectue une de-concaténation (cf. §V.4) si ce fichier n’est pas n*ul*l et s’il s’agit d’un fichier XML.
* En cas d’erreur, un traitement particulier est fait (cf. §V.2).
* La suite du traitement est lancée et un déplacement est effectué vers le répertoire *BUSIN* (cf. §V.6).
* Si la liste des fichiers est vide, ceci est tracé.
* Sinon, Le traitement boucle sur le la liste des fichiers.
* Chaque fichier est traité à l’aide de la méthode *imputation* (cf. §V.6).
* En cas d’erreur fonctionnelle, le traitement continue et le type de retour est fixé à 201 (le traitement trace l’exception).
* Le traitement traite aussi des exceptions de type Warning avec un type d’erreur égale a 0.
* Un bloc Finally lance la suite du traitement
* Si le fichier a un flag archive, le contenu du fichier est compressé.
* Suppression du fichier temporaire (qui sert pour la compression)
* Si la suppression se passe mal, un message indiquant l’erreur est tracé et le type de retour est fixé à 201.
* Si le fichier n’a pas de flag suppression, l’information est tracée.
* Le fichier est supprimé du dossier REELIN la suppression est tracée.
* Si la suppression se passe mal, un message indiquant l’erreur est tracé et le type de retour est fixée à 201.
* En cas d’une autre erreur, une erreur est tracée et le type de retour est fixé à 1.
* Affiche le durée d’exécution du batch.
* Le type de retour est renvoyé.

## Traitement Particulier

Ce traitement est lancé lors d’une erreur lors de là de-concaténation d’un fichier :

* Cette erreur est tracée.
* Le fichier est renommé
* Le fichier est déplacé du dossier *BUSIN* vers le dossier *REELIN.*
* Si le déplacement du fichier se passe mal, l’erreur est tracée et une exception est générée.
* Sinon, l’erreur lors de la de-concaténation génère une exception.
* Le traitement s’arrête.

## 

## FileUtils.getOldestFile

Cette méthode effectue le traitement suivant :

* Lister Fichiers.
* Garder que les *XML* et T*XT* et ceux qui commence ou se termine par *TMP*.
* Trier par ordre décroissant (selon date)
* Renvoyer le premier dans la liste.

## saxDecoupageEcnParser.deconcatener

Cette méthode effectue le traitement suivant :

* Ouverture d’un flux de lecture vers le fichier XML.
* Dé-concaténer le fichier en entrée. Les balises &lt;flux&gt; et leurs contenus sont séparés et placés dans un seul fichier par balise.
* En cas d’exception lié au parsing ou d’une exception de type entrée/sortie, une exception est levée.
* Dans un bloc FINALLY , on supprime la liste des fichiers générés lors de cette dé-concaténation (cf. §V.5).

## saxDecoupageEcnParser.supprimerListeFichiers

Cette méthode effectue le traitement suivant :

* On ferme le flux d’écriture vers le dernier fichier crée.
* En cas d'erreur lors du découpage, les fichiers sont supprimés.

## *LauncherConecteur.moveFile*

A compléter…

## imputationFichierEcn.imputation

Cette méthode prend en entrée, le paramètre :

|  |  |
| --- | --- |
| Type | Nom |
| *Java.io.File* | *file* |

Et renvoie rien (void).

Cette méthode effectue le traitement suivant :

* Si le fichier est un fichier XML, fait appel à la méthode *imputationFichierEcn .*imputationFichierXML avec la gestion des exceptions adéquate*.*
* Si le fichier est un fichier TXT, on fait les tests suivant :
* Si le nom de fichier ne commence pas par ME\_RDJ ou ME\_RDJREP, on génère une exception et on trace l’information (niveau ERROR). Sinon, le traitement continue.
* Si le nom de fichier a la bonne taille (34), le traitement continue. Sinon, on génère une exception et on trace l’information (niveau ERROR) et le traitement s’arrête.
* On fait appel à la méthode *imputationFichierEcn.imputationFichierPlat* pour traiter le fichier (cf. §V.8).
* Dans les autres cas, on trace le message ‘"Le fichier : " + file.getAbsolutePath() + " n'est ni un fichier .txt ni un fichier .xml"’ (niveau ERROR) et on généré une exception.

## imputationFichierEcn.imputationFichierPlat

Elle permet de créer la liste des traitements à partir du fichier plat passé en paramètre. Cette méthode effectue le traitement suivant :

* Les traitements stockés dans l’objet *FlatStremManager* sont vidés.
* On affecte le ficher en cours au FlatStremManager.
* On fait appel à la méthode *processFile* (cf. §V.9).
* En cas d’exception de type WarningException qui est générée, cette exception est tracée.
* En cas d’une exception de type BusinessException, l’erreur est tracée en debug mode et une exception est générée.
* Et dans un bloc Finally, le flux flatStremManager est clos.
* Et si pas, d’erreur, les traitements du flatStreamManager sont renvoyés par la méthode.

A noter qu’il y deux implémentations de ce traitement. Selon la version du ME, on prend l’une ou l’autre des implémentions :

* *FlatStreamReaderMouvementElementaireImpl.*
* *FlatStreamReaderMouvementElementaire\_V2\_0\_Impl.*

Dans ce qui suit, nous allons décrire les traitements de la deuxième implémentation.

## flatStreamManager.processFile

Cette méthode permet de lire le fichier et construit la liste des traitements à injecter dans CODA.

Cette méthode prend en entrée, le paramètre :

|  |  |
| --- | --- |
| Type | Nom |
| *File* | *flatECN* |

Et renvoie rien (void).

Elle effectue le traitement suivant :

* Récupère l'identifiant du ME.
* Récupère la version du ME.
* Récupère le flux de lecture correspondant à la version du ME.
* Ce flux de lecture est ouvert.
* L’objet qui se charge de la lecture est récupéré (selon la version du ME).
* On initialise le lecteur pour le fichier en cours de lecture.
* Un traitement particulier est fait pour la première ligne :
* On construit l'entête et on le fait qu'une seule fois.
* On ne construit l'entête qu'une seule fois.
* On ne récupère les infos fichiers qu'une seule fois.
* On récupère la ligne en cours de lecture dans un objet POJO java (ligne assurance).
* En cas d’erreur fonctionnelle ou une erreur d’exécution dans les 4 précédents points, une exception est levée et l’erreur est tracée en DEBUG mode.
* Un nouveau traitement est créé (cf. §V.10).
* On met à jour les données du traitement en fonction de l’entête des lignes du fichier (toujours là même Pour toutes les lignes d'un ME).
* On affecte 2 au type du mouvement comptable.
* On ajoute un mouvement comptable au traitement associé au fichier.
* On parcourt les autres lignes et pour chaque ligne, on fait ceci :
* On ajoute une ligne mouvement comptable à la liste des traitements (cf. §V.13).
* En cas d’erreur, une exception est générée.
* En cas d’erreur, une exception est générée.
* Et dans un bloc finally, le flux de lecture et fermé.
* Les lignes lues sont stockées dans des objets traitements qui sont renvoyées par la méthode.

## flatStreamManager.addNewTraitement

Ce traitement permet d’initialiser un objet POJO entête. Cette méthode effectue le traitement suivant :

* On trace le début du traitement d’une nouvelle ligne.
* On commence le traitement processor.
* On initialise un objet POJO java pour la ligne en cours de traitement.
* On vide les lignes issues du traitement précédent.
* On réinitialise l’ensemble des objets utilisés dans le traitement de la ligne précédente.
* On affecte le fichier au traitement processor en cours.

## 

## FlatStreamManager.refreshTraitement

Cette méthode prend en entrée, le paramètre :

|  |  |
| --- | --- |
| Type | Nom |
| *EnteteMouvementComptable* | *enteteMouvementComptable* |
| *String* | *idME* |

Et renvoie rien (void).

Cette méthode fait appel à *traitement.refresh* (cf. §V.12).

## traitementProcessor.refresh

Cette méthode prend en entrée, le paramètre :

|  |  |
| --- | --- |
| Type | Nom |
| *String* | *idTraitement* |
| *Sring* | *idME* |
| *String* | *DEFAULT\_RELANCE\_ID = "01"* |
| *String* | *idPrecedents* |
| *String* | *ensembleComptable="CODA"* |
| *String* | *issuer* |

Et renvoie rien (void).

Cette méthode fait le traitement suivant :

* Trace le message ‘"Mise a jour du traitement [" + idTraitement + "], relance [" + idRelance + "]"’ dans le fichier de logs.
* Prépare un objet traitement.
* Si l’un des paramètres en entrée est *null,* met le statut du traitement FAILED.
* Sinon, on met le statut du traitement à INITIALIZED.
* On Ignore le traitement si l'ensemble comptable n'est pas CODA.
* Ignore le traitement s'il est déjà OK dans les tables de suivi

Ou si l'un des autres traitements pour le même identifiant de

Traitement moteur est à OK.

* Ensuite, on écrit le traitement dans la table SUIVI sans tenir compte de son statut et uniquement si son *IdTraitementMoteur* n’est pas *null*.

## flatStreamManager.addLigneMouvementComptable

Cette méthode effectue le traitement suivant :

* N’ajoute pas la ligne si le statut du traitement est IGNORED ou fAILED.
* Si tout est ok, la ligne est ajoutée en appelant la méthode *addLigneMouvementComptable* (cf. §V.14).

## traitementProcessor.addLigneMouvementComptable

Cette méthode effectue le traitement suivant :

* Si le traitement n’est pas au statut FAILED, on continue.
* On construit la clé primaire de la ligne qui sera insérée en base.
* On procède à quelques vérifications dont le fait que la ligne n’a pas été déjà insérée dans la liste des lignes qui seront inséreras en BDD.
* Ces vérifications permettent d’initialiser le statut ainsi que d’autres informations concernant la ligne en cours de traitement.
* On récupère l’objet rupture de la ligne traitée.
* En cas d’erreur lors de la récupération de l’objet rupture, une exception est générée et le traitement est mis au statut FAILED\_RUPTURE. Sinon le traitement continue.
* On récupère l’objet *DocNumsParRupture* de la rupture.
* Et selon le type du mouvement comptable, on initialise un une variable *activite* :
* Mouvement comptable ets égale à 2 : activite = "ASSURANCE".
* Autre mouvement comptable : activité = "ASSURANCE".
* Ensuite, les données de la table EQUILIBRE sont récupérées par activité si ce n’est pas déjà le cas.
* On récupère un certain nombre des données (*oasDocument*) du document courant (ligne en cours de traitement) et ceci si le *oasDocument* est *null* ou si son code compagnie est diffèrent de celui de la société de l’objet *rupture* ou son code est différent du code BOOKENTRY de la *rupture*.
* Dans le cas où les données précédentes sont récupérées, cette récupération est tracée en debug mode.
* Et en cas d’erreur fonctionnelle(*BusinessException*) lors de la récupération de données précédentes, le traitement est mis au statut FAILED et le traitement en cours est insère dans la table SUIVI s’ils sont pas *null et on arrete l’insertion des données de la ligne en cours (exception non propagé) et* *notifie l’échec de recuperation du document => le document n'existe pas dans CODA*.
* Si l’objet *docNumsParRupturePrecedent* est *null* ou si le *docNumsParRupture* est différent du précédent *docNumsParRupture* ou si le nombre des lignes des mouvement comptable est supérieure ou égale nombre maximum des lignes par *DocNum* ou si le traitement est au statut ‘Nouvelle document Société’ ‘ (*isNewSocDoc*) :
* Dans le cas ou *docNumsParRupturePrecedent* n’est pas *null* :

° Si on est en debug mode, on trace le message suivant : ‘RUPTURE Detectée par rapport au lignes prédentes’.

° Si le nombre des lignes des mouvement comptable est supérieure ou égale nombre maximum des lignes par *DocNum*, un message d’information est tracée en debug mode et le champ *decoupeEnCours* de l’objet *equilibre* est mis à vrai (true).

° Si on est en debug mode, un message concernant l’écriture précédente est tracé.

° Si le docNumsParRupture est différent du précédent *docNumsParRupture*, on déclenche l’écriture en base des *LinkLine* et *LinkComment* (objet previousOasDocument) puis on vide la liste des ECN en cours pour un *docNum* donné et on trace le message ‘Nouvelle rupture sur le document’. Sinon, on fait la même chose et on trace le message ‘Pas de rupture sur le document’ (cf. V.§15).

* *Si changement de docNum, il faut initialiser l’objet Equilibre, mais d'abord, il faut verifier que l'équilibre, si celui-ci n'est pas bon, il faut tout supprimer de la corbeille.*
* La variable locale *isDocnumFichier* est initialisé à *false* et ensuite, elle est modifiée selon la valeur de *enBKRef* de la ligne mouvement comptable et si currentdocNum est null ou vide.
* Si *isDocumentFichier* est vrai la variable *nextDocNum* est initialisée et égale à *currentDocNum.* Sinon, elle est initialisée selon la valeur de *currentDocNum*.
* Si *docNumsParRupture* est null, on trace le message ‘ « Rupture jamais traitée => creation rupture pour» + nextDocNum‘ en debug mode et on procédé à l’initialisation de la rupture.
* Sinon, on trace le message ‘"Rupture traitée => ajout du docnum à la liste des docnums pour la rupture"+ nextDocNum’ et on ajout *nextDocNum* à la liste *docNumsParRupture.*
* Ensuite, on affecte nextDocNumà *currentDocNum* et on trace les messages ‘ocNum courant devient : " + nextDocNum’ et ‘ [Corbeille] Ecriture LinkHead, docNum [" + docNumsParRupture.getCurrentDocNum() + "]"’ en debug mode.
* Si *TraitementStatus* est *isNewHead (docnumList* ne contient pas la pk de la ligne en cours) , on déclenche l' écriture en base (cf. §V.16) de la *LinkHead* et en cas d’exception dans cette partie, le traitement en cours est mis au statut FAILED .
* Si une extourne est à faire, *currentDocNum* est valorisé en fonction de sa valeur acteulle.
* Ensuite, le messages suivants ‘Le document doit être extourné => Nouveau docNum: " + currentDocNum’,’ "DocNum courant = : " + currentDocNum ‘ et ’ Ecriture LinkHead, docNum Extourne [" + currentDocNum + "]"’ sont tracés en debug mode..
* Si *TraitementStatus* est *isNewHead (docnumList* ne contient pas la pk de la ligne en cours) , on déclenche l' écriture de la LinkHead en base (cf. V.§15) pour l'extourne et en cas d’exception dans cette partie, le traitement en cours est mis au statut FAILED.
* Ensuite, l’objet ligneMvtComptableExtourne, ruptureExtourne et docNumExtourne sont récupères en fonction de la ligne mouvement comptable.
* Si docNumExtourne est null (La rupture n'a jamais été traitée), il est initialisé en fonction de l’objet ruptureExtourne. Sinon on ajoute du *docnum* au document.
* Ensuite, on initialise à nouvel objet OasDocument, par oasDocument, à previousOasDocument et on trace le message ‘"Conserve valeur OasDocument prédent : SOC = [" + previousOasDocument.getCmpCode()"], DOCCODE=[previousOasDocument.getCode() + "], EXTOURNE=["previousOasDocument.getExtRefLabel1() + "], TEN99=[" + previousOasDocument.getTen99()+ "]‘ en debug mode.
* On ajoute l’objet *ligneCouranteMouvementComprable* à la liste des lignesMouvementComptable et on affecte docNumsParRupture à docNumsParRupturePrecedent.

## traitementProcessor.flush

Cette méthode prend en entrée, les paramètres :

|  |  |
| --- | --- |
| Type | Nom |
| *OasDocument* | *aasDocument* |
| *boolean* | *isNewRupture* |

Et renvoie rien (void).

Cette méthode effectue le traitement suivant :

* On initialise les différentes variables locales.
* S’il y a des lignes ECN :
* On boucle sur ces lignes et on fait un traitement particulier pour chaque ligne.
* S’il y a des lignes Mouvement Comptable :
* On boucle sur ces lignes et on fait un traitement particulier pour chaque ligne.
* Si l’équilibre avec découpe en cours
* On fait un traitement particulier.
* Sinon, un autre traitement pour le cas sans découpe en cours.
* Ensuite, on procède au vidage de la liste des lignes du lprPrécédent pour éviter

de charger la mémoire (Toutes les 500 lignes, valeur 500 est en dur dans le code).

* En cas d’erreur technique (TechnicalException) ,l’exception est propagée.

## traitementProcessor.flushLinkHead

Cette méthode prend en entrée, les paramètres :

|  |  |
| --- | --- |
| Type | Nom |
| *String* | *currentDocNum* |
| *LigneMouvementComptable* | *ligneCouranteMouvementComptable* |
| *Boolean* | *extourne* |
| *String* | *previousDocNum* |
| *OasDocument* | *aasDocument* |

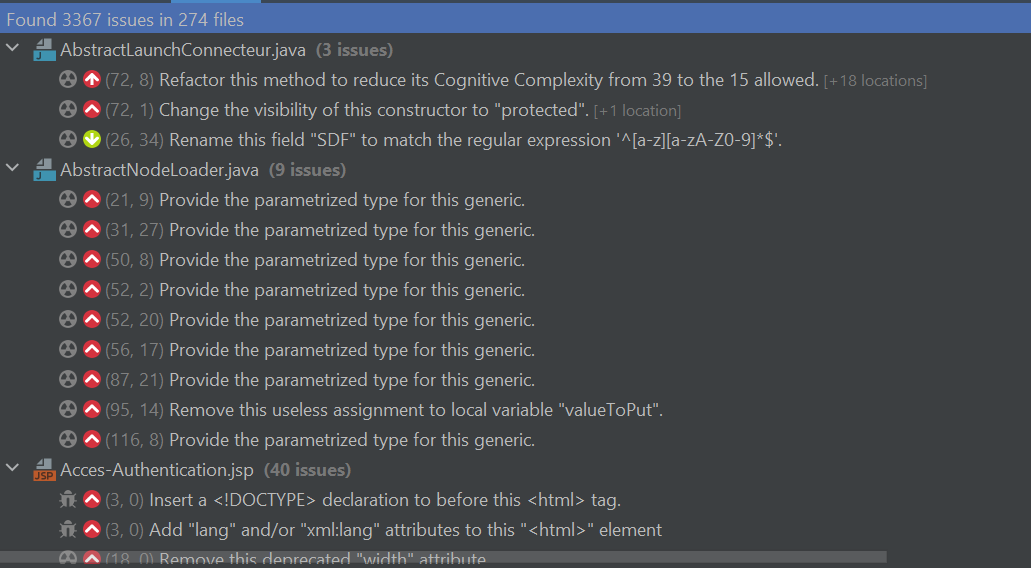
Et renvoie rien (void).

Cette méthode effectue le traitement suivant :

* La ligne linkHead est initialisée en fonction des paramètres en entrée ci-dessus.
* Dans le cas d’une extourne, on fait un traitement particulier et on valorise certains champs de la ligne LinkHead qui sera insérée en base.
* Ensuite et dans un bloc try/catch :
* On trace certains messages en debug mode.
* On crée l’objet linkHeadDTO à partir de la ligne LinkHead.
* On crée la clé primaire pkhead.
* Si la liste headList ne contient pas la clé primaire, on insère pkhead dans la position 1 de la liste headList et on fait appel à LinkHeadDAO pour insérer linkHeadDTO.
* En cas d’erreur technique (TechnicalException) ,l’exception est propagée.

# Analyse Sonar du projet

Nous avons lancé l’analyse Sonar du projet dont le résultat est le suivant :



# Propositions et axes d’amélioration

Nous allons dans ce chapitre parler des axes d’améliorations à explorer à l’issu de cette analyse basée sur :

* L’analyse Dynatrace
* Lecture du code source
* Documents Projets
* Analyse SonarQube

## Bugs sonar

Nous recommandons de corriger au moins toutes les alertes critiques remontées par l’analyse SONAR. De préférence corriger un maximum des alertes majeures (réduire à 20 si possible).

## Mettre à jour les librairies utilisées

* Hibernate JPA

La librairie utilisée actuellement par le connecteur CODA est la suivate :

|  |
| --- |
| *<dependency>*  *<groupId>org.hibernate.javax.persistence</groupId>*  *<artifactId>hibernate-jpa-2.0-api</artifactId>*  *<version>1.0.0.Final</version>*  *</dependency>* |

Passer à une version plus récente de l’api *cette librairie*, au moins utiliser la version 1.0.2 de cette librairie :

|  |
| --- |
| *<dependency>*  *<groupId>org.hibernate.javax.persistence</groupId>*  *<artifactId>hibernate-jpa-2.1-api</artifactId>*  *<version>1.0.2<version>*  *</dependency>* |

* CINRJ : la version recommandée par SOT est 5.10.7 pour rester à jours avec la release note.
* Sinon, mettre à jour les autres librairies avec des versions plus récentes que celles utilisées tout en gardant l’application fonctionnelle avec le moins de modification de code possible.

## Utilisation de LinkedList

On a constaté l’utilisation à plusieurs reprises de l’objet LinkedList qui altère significativement les performances et il est conseillé d’utiliser *ArrayList (ou Set pour garder l’ordre)* à la place.

## Augmenter la taille de la mémoire tampon

Nous pouvons aussi jouer sur la taille de la mémoire tampon (buffer size) et ceci si les fichiers lus dépassent 8192 caractères (valeur actuelle). La nouvelle valeur à définir sera définit en faisant plusieurs tests en phase de développement.

## Récrire la partie lecture du fichier TXT et ces lignes

Cette proposition consiste à utiliser la méthode *Files.Lines* pour lire toutes les lignes d’un coup et le traiter par la suite, cela va éviter un aller-retour vers le disque pour lire les lignes un par un (IO).

Aussi il faut penser à refactoriser le traitement de chaque ligne du fichier.

## Paralléliser les traitements

Il faudrait Paralléliser certains traitements pour améliorer les performances du batch.

Nous avons constaté des blocs de codes sources flaqués synchronisés, par contre on ne voit pas l’utilité ou le besoin pour le faire.

## Migrer le batch en spring batch

Une autre solution serait de migrer le batch vers spring batch. Cette solution serait intéressante dans le cas où les traitements, ou certains d’entre eux, peuvent être parallélisés.

## Analyse de logs

En complément de ces propositions, il faut revoir la gestion des logs pour améliorer les analyses et les investigations en cas de problème.