Réalisation d'une plateforme inter-applicative de gestion de flux de données

Patrice Lapierre

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE LYON

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : INFORMATIQUE

OPTION : SYSTEMES D'INFORMATION

par

Patrice LAPIERRE

Réalisation d'une plateforme inter-applicative de gestion de flux de données

Soutenu le 18 janvier 2013

JURY

PRESIDENT : **Monsieur Bertrand DAVID (CNAM Lyon)**

MEMBRES : **Monsieur Claude GENIER (CNAM Lyon)**
Monsieur Christophe PICOULEAU (CNAM Paris)
Madame Laurence ROINET (OSIATIS)
Monsieur Arnaud BESOIN (SNCF)

Remerciements

Je remercie tout d'abord Laurent CHEVALLIER, chef de la section DSI-T/EH/A de la SNCF, et Arnaud BESOIN, chef de projet dans cette même section, qui m'ont intégré dans leur équipe en tant que prestataire et qui, dès les premiers balbutiements du projet, m'ont confié des responsabilités concernant notamment l'étude du besoin.

Je remercie également tous les membres de l'équipe du projet pour leur professionnalisme et leur bonne humeur. Je tiens par ailleurs à remercier Laurence ROINET, manager, qui a fait son possible afin de me placer sur une mission me permettant de réaliser mon mémoire d'ingénieur pour le CNAM.

Je remercie aussi le CNAM de Lyon, pour la qualité des enseignements que j'ai pu y suivre.

Enfin, je tiens à remercier les membres du jury pour leur participation à ma soutenance, et plus précisément M Bertrand DAVID qui m'a accompagné et conseillé pour la réalisation de ce mémoire.

Liste des abréviations

CFT (Cross File Transfer) : Logiciel de transfert de fichiers sécurisés.

CSV (Comma-Separated Values) : Format de fichier dont les champs sont séparés par un caractère défini. La caractère de séparation des champs est généralement la virgule « , ». Toutefois dans sa variante française, le caractère de séparation est, le plus souvent,le point-virgule « ; ».

DSIT : Direction des Services Informatiques et des Télécommunications.

MSMQ (Microsoft Message Queuing) : Composant Windows sous forme de service permettant la gestion de files de messages.

ORM (Object-Relationnal Mapping) : Outil permettant de créer l'illusion d'utilisation d'une base de données orientée objet à partir d'une base de données relationnelle. L'outil permet de pouvoir définir des correspondances entre les tables d'une base de données et les objets implémentés par un programme.

SNCF : la Société Nationale des Chemins de fer Français est une entreprise publique opérant dans le transport ferroviaire.

UML (Unified Modeling Language) : Langage de modélisation graphique permettant de spécifier, de visualiser, de construire et de documenter les artéfacts constituant un système informatique.

WCF (Windows Communication Foundation) : Couche de communication ajoutée au Framework .NET à partir de la version 3.0. Cette couche permet au développeur de se focaliser avant tout sur les caractéristiques du service sans se soucier de son implémentation.

XML (Extensible Markup Language) : Langage informatique de balisage générique permettant de stocker de l'information de manière structurée.

Glossaire

Brique Technique : Module de traitement des données et de mise en forme utilisé dans le Frontal RH.

Central M : Progiciel de gestion des ressources humaines et de gestion de la paie utilisé uniquement par le service médical de la SNCF.

Central P : Progiciel de gestion des ressources humaines et de gestion de la paie utilisé actuellement à la SNCF (à l'exception du service médical).

COLDIF : Module de collecte et de diffusion des flux de données entre les applications et la Brique Technique. Le nom COLDIF désigne Collecte et Diffusion.

Filtrage horizontal des données : Filtrage des données permettant, à partir d'une table initiale contenant un certain nombre de colonnes, de ne sélectionner que les colonnes désirées.

Filtrage vertical des données : Filtrage des données permettant d'identifier, au sein d'un volume initial, un sous ensemble correspondant à l'information désirée.

Frontal RH : Plateforme d'échange de flux de données inter-applicative utilisée actuellement par la SNCF. Ce logiciel se divise en deux parties distinctes dénommées Brique Technique et COLDIF.

HR Access : Progiciel de gestion des ressources humaines et de la paie qui sera utilisé par l'ensemble des services SNCF à partir de 2014.

MQSeries : Service de messagerie inter-applicative de type FIFO (First In, First Out[1]) développé par la société IBM. Il permet l'échange d'informations entre plusieurs applications via l'envoi de messages placés dans des files d'attente.

[1] En français : Premier entré, premier sorti

Table des matières

Introduction

La SNCF (Société Nationale des Chemins de fer Français) possède actuellement, au sein de son service informatique dédié aux ressources humaines, une multitude d'applications nécessitant l'utilisation d'informations issues d'autres applications du système informatique. A titre d'exemple, la quasi-totalité des applications nécessite la liste des personnes travaillant au sein de la SNCF afin de pouvoir gérer les autorisations et les rôles que chaque personne peut avoir dans un programme. Dans cette organisation, on considère que le système permettant de saisir et d'administrer une donnée en est le propriétaire. Il est ainsi primordial de pouvoir faire échanger d'importantes quantités d'informations entre les diverses applications qui composent le Système d'Information de la SNCF. Suite à une importante modification du Système d'Information (SI) des ressources humaines, les besoins au niveau des échanges de données ont considérablement changés.

Le projet présenté dans ce mémoire consiste à mettre en place une plateforme de gestion de flux de données inter-applicative capable de réceptionner un flux et de pouvoir le personnaliser pour chaque application consommatrice[2]. Les consommateurs n'ayant pas tous les mêmes besoins en matière de flux d'informations, la personnalisation de ces derniers se révèle essentielle dans ce projet. La plateforme de flux doit donc être capable de récupérer un flux, de le personnaliser, puis de l'expédier au bon destinataire dans le format attendu par celui-ci.

J'ai participé à ce projet au sein de la DSIT, et plus précisément, dans une cellule nommée DSIT/EH/A, qui est un service transverse dans l'organisation de la direction des services informatiques et des télécommunications (DSIT) dédié aux ressources humaines de la SNCF. Sur ce projet, mon rôle fut, dans un premier temps, de réaliser les spécifications fonctionnelles de l'application jusqu'à leur validation par la maitrise d'ouvrage.

[2] Afin de faciliter la lecture, dans la suite de ce mémoire, les applications consommatrices seront désignées par les termes « consommateurs » ou « clients » et les applications envoyant des données à la plateforme seront nommées « fournisseurs ».

J'ai donc eu la responsabilité de faire coïncider les besoins exprimés par la MOA[3] avec les spécifications et surtout de m'assurer que l'intégralité des exigences formulées soient couvertes par les spécifications Après cette étape, j'ai rejoint l'équipe de développement qui avait préalablement eu le temps de mettre en place l'architecture technique de l'application avec l'aide de l'architecte. Durant cette étape, j'ai principalement eu la tâche de travailler à la mise en place des configurations des flux et, plus particulièrement, concernant les problématiques de désérialisation[4] et de covariance[5] des objets. J'ai également participé au développement de diverses fonctionnalités de l'application. Enfin, pendant la phase de recette, j'ai eu la responsabilité d'effectuer de nombreux tests dans le but de valider le comportement de l'application vis-à-vis de celui qui était attendu. Cette étape a eu pour finalité de s'assurer que l'outil développé respectait les exigences du cahier des charges. Lors de cette phase, j'ai également assisté la maitrise d'ouvrage en lui fournissant les configurations de flux qu'elle souhaitait voir mises en place, ainsi qu'en répondant à toutes les interrogations qu'elle pouvait avoir concernant le fonctionnement de l'application.

Afin de permettre au lecteur de se projeter dans la mise en œuvre de ce projet, nous présentons, dans un premier temps, le cadre dans lequel celui-ci s'est déroulé, ainsi que les besoins qui nous avaient été exprimés. Dans un second temps, nous abordons les solutions qui s'offraient à nous pour répondre à la demande puis détaillons les raisons qui nous ont permis de faire un choix. Nous traitons ensuite de la manière dont nous avons procédé lors de la conception puis du développement de l'application. Enfin, nous terminons par l'analyse de la phase de recette et la présentation de l'approche que nous avons choisi d'appliquer pour valider le travail que nous avions réalisé dans le cadre de ce projet.

---

[3] Maîtrise d'ouvrage

[4] La désérialisation permet de charger un objet (au sens codage informatique) à partir d'une structure de données établie qui, le plus souvent, est sous format XML.

[5] La covariance permet de pouvoir utiliser un sous-type en lieu et place d'un type attendu (cf paragraphe 5.3.3).

Chapitre 1

Cadrage du projet

Il existe, actuellement, au sein du système informatique interne de la SNCF, un ensemble applicatif nommé communément *Frontal RH*. Cet ensemble est responsable des échanges de flux de données entre la majorité des applications qui composent le système d'information de la SNCF. Aujourd'hui, la refonte de plusieurs outils majeurs du système d'information de la SNCF nécessite d'avoir la capacité de prendre en charge un volume croissant de données au niveau des flux échangés. Le *Frontal RH* arrivant à bout de souffle en termes de capacités de traitement et étant difficile à maintenir en raison d'une composition hétérogène comprenant un nombre important de langages et de scripts différents, la décision fut prise de créer un nouvel outil plus facile à maintenir et, surtout, pouvant traiter une plus grande quantité de flux. Dans ce contexte, il nous a naturellement été demandé que la Nouvelle Plateforme de Flux (NPF) puisse reprendre l'intégralité des fonctionnalités de l'ancienne. Elle devait également pouvoir s'adapter à de nouveaux besoins mais également pouvoir répondre à une contrainte forte de haute disponibilité quel que soit le volume de données à traiter.

1.1 – Présentation de la SNCF

La SNCF a été créée le 31 aout 1937 par convention entre l'Etat français et les compagnies ferroviaires privées de l'époque. A sa création, la SNCF était une société anonyme d'économie mixte détenue par l'Etat à 51% ; les 49% restants étant détenus par les anciennes compagnies ferroviaires regroupées. Le 1er janvier 1983, la convention de 1937 arrivant à son terme, la SNCF est revenue en totalité à l'Etat qui l'a dotée d'un nouveau statut : celui d'établissement public à caractère industriel et économique (EPIC).

En 1997, pour se conformer à la directive européenne visant à permettre une ouverture à la concurrence, la gestion de l'infrastructure et celle de l'exploitation ont été séparées. Le RFF (Réseau Ferré de France) naquit ainsi de cette séparation et devint propriétaire du réseau ferré français qui comprend les voies ferrées, les installations de télécommunications et les bâtiments liés aux infrastructures. La SNCF conserva alors l'exploitation du réseau ferré en contrepartie d'une redevance pour l'utilisation des voies. Bien que l'entretien et la gestion de la circulation sur les voies ferrées soient du ressort du RFF, c'est aujourd'hui la SNCF qui exécute ces tâches pour le compte du RFF et reçoit en contrepartie une rémunération pour ces actions.

Le groupe SNCF a plusieurs activités réparties en cinq secteurs :

- **SNCF Infrastructures :** Entretien et exploitation du réseau ferré et ingénierie ferroviaire,
- **SNCF Proximités :** Transport public urbain, périurbain et régional pour les voyageurs,
- **SNCF Voyage :** Transport ferroviaire de voyageurs longue distance et à grande vitesse,
- **SNCF Geodis :** Transport et logistique de marchandises,
- **Gare & Connexions :** Gestion et développement des gares.

La SNCF emploie actuellement près de 160 000 personnes uniquement pour l'exploitation et l'entretien des lignes ferroviaires qui comptent 30 870 kilomètres et font circuler 13 400 trains par jour. Actuellement, la SNCF transporte plus de 1 milliard de voyageurs par an. Pour toutes ces activités et afin de gérer le volume important de voyageurs, la SNCF compte sur un important dispositif informatique. Ainsi, le Service DSI-T[6] assure et garantit le fonctionnement et la mise en service des systèmes d'information et des télécommunications de la SNCF.

[6] DSI-T : Direction des Services Informatiques et des Télécommunication.

Au sein de la DSI-T, il existe quatre grandes familles de métiers :

- **Les métiers de l'exploitation :** ils garantissent la mise en œuvre, le bon fonctionnement et la sécurité des infrastructures et des applicatifs dont ils ont la charge.
- **Les métiers de conseil, support et assistance :** ils permettent l'évolution, l'optimisation et la veille du domaine dont ils sont spécialistes, et assurent un soutien aux différents acteurs du Système d'Information (SI).
- **Les métiers de commercialisation :** ils se chargent de la relation avec les clients en pilotant la conception d'une offre et/ou en proposant les différents services du SI.
- **Les métiers des études et développements :** ils assurent la réalisation de composants logiciels applicatifs depuis le pilotage du projet jusqu'à la maintenance applicative.

Les métiers des études et développements sont regroupés au sein du service nommé DSI-T/E[7], qui est lui-même subdivisé en plusieurs services, dont la DSI-T/EH/A[8] qui est le service dans lequel j'ai effectué mon stage.

1.2 - Présentation du service DSI-T/EH/A

La DSI-T/EH/A a en charge les offres des prestations d'architecture, d'intégration applicative et de suivi du système, et enfin de développement d'applications pour le SIRH (Système d'Information des Ressources Humaine).

[7] DSI-T/E : Département Etudes et développements de la DSI-T. Ce département est chargé de concevoir, réaliser, qualifier et maintenir les applications informatiques.

[8] DSI-T/EH/A : Service dédié au SIRH ayant en charge les prestations d'architecture, d'intégration applicative et de suivi système.

Plus précisément, on retrouve, au sein de ce service, trois métiers bien différents :

- **L'intégration :** Une équipe est chargée de l'intégration des applications gravitant autour du SIRH
- **Le système :** Une équipe a la charge du paramétrage des serveurs de développement et d'intégration utilisés par les applications du SIRH. Elle veille également au bon fonctionnement des serveurs.
- **L'assistance technique :** Une équipe est chargée d'accompagner techniquement les autres équipes travaillant pour le SIRH. Cette équipe réalise, contrôle et valide également les architectures pour d'autres services. Enfin, elle a en charge la création, la maintenance et le suivi des applications transverses au SIRH (exemples : application d'envoi d'emails, application gérant l'identification et les rôles des utilisateurs, plateforme d'échange de flux de données inter-applicative...).

Au sein du service, l'équipe en charge de la réalisation de la Nouvelle Plateforme de Flux est composée des personnes suivantes :

Sylvain MEILHAC : Chef de projet

Stéphane CAREME : Architecte .NET

Laurent CATHERIN : Concepteur développeur

Jean-Vincent LAMY : Concepteur développeur

Patrice LAPIERRE : Concepteur développeur

Chapitre 2

Présentation du projet

Le service DSI-T/EH/A étant un service transverse vis-à-vis des autres services qui composent la DSI-T, il a notamment la charge de la gestion des flux de données qui transitent entre les diverses applications qui composent le SIRH. Dans cet objectif, il a été décidé, il y a maintenant un peu plus de cinq années, de centraliser les flux transitant entre les applications afin de pouvoir contrôler le bon fonctionnement des échanges, mais également de pouvoir facilement identifier les problèmes éventuels. Au fur et à mesure des nouveaux besoins qui firent leurs apparitions, cette plateforme d'échange des flux d'information centralisée a été modifiée et améliorée.

2.1 – Origines du projet

Actuellement, la SNCF utilise, pour la gestion des ressources humaines et de la paie de l'ensemble du personnel qui la compose, un outil nommé *Central P*. Cet outil va prochainement être changé au profit de l'outil *HR Access*. Il est planifié que, lors de ce changement d'outil, une marche en double (les deux outils continueront de fonctionner conjointement) sera observée pendant deux années après le transfert total des données et des traitements vers le nouveau système. Cette marche en double a pour objectif de s'assurer du bon transfert des traitements, sans régression, par comparaison des résultats qui seront obtenus sur les deux logiciels. Cette période transitoire va, de ce fait, générer une très forte hausse du volume des flux. Cette hausse ne pourra être supportée par le processus de gestion des flux actuellement en place.

Il est donc prévu qu'avec ce changement d'outil, les flux de données qui s'effectueraient encore entre les applications clientes et l'outil de gestion des ressources humaines passent exclusivement par la nouvelle plateforme de flux afin de centraliser et faciliter leur supervision.

Le projet qui fait l'objet de ce mémoire consiste donc en une profonde refonte de l'actuelle plateforme de flux, qui se trouve limitée face à une prochaine montée du volume de données à traiter. Le projet consiste donc à concevoir une plateforme d'échange de flux capable de pouvoir prendre en compte un volume de flux et de données croissant, mais également de mettre en place une structure capable de pouvoir prendre en compte, facilement, de nouveaux besoins. Le nom retenu pour ce projet est NPF pour *Nouvelle Plateforme de Flux.*

Avant la mise en place de la *Nouvelle Plateforme de Flux*, les flux de données étaient traités par le *Frontal RH* et, plus précisément, par une application nommée *Brique technique*. A cette période, les flux provenaient exclusivement de l'application *Central P* puis étaient personnalisés par la *Brique technique* afin d'être distribués à divers consommateurs. La figure 1 représente le fonctionnement du *Frontal RH* avant la *NPF*.

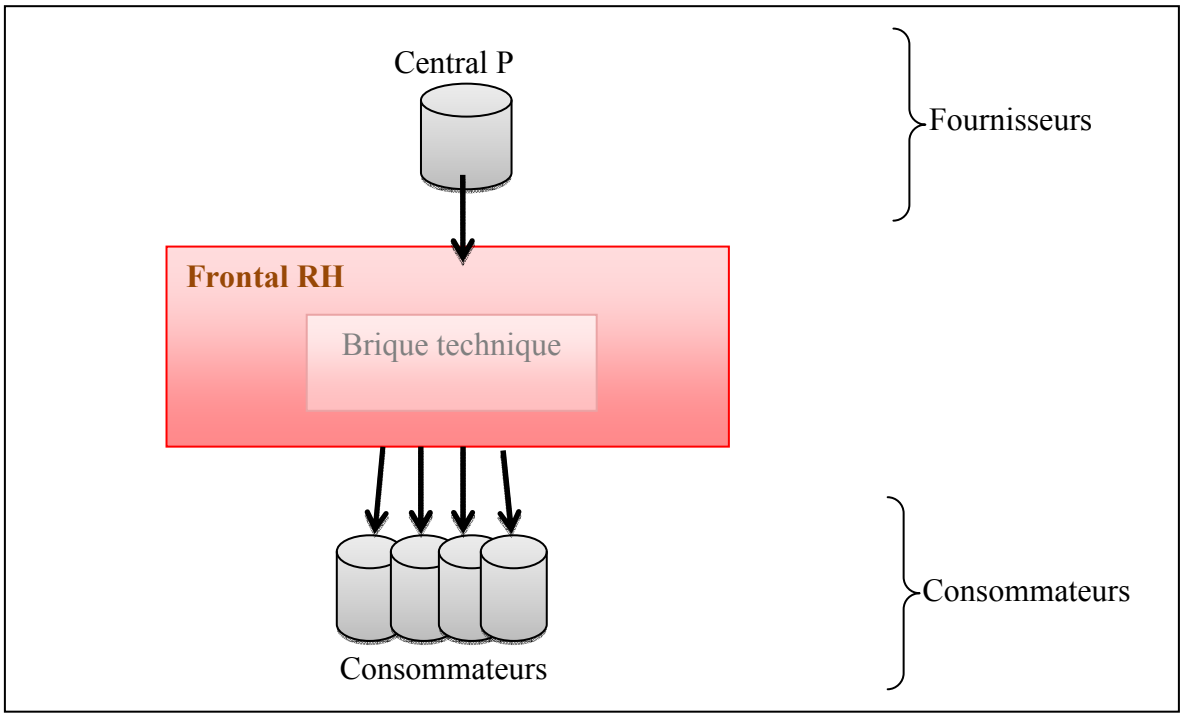

Figure 1 : Echanges de flux avant la mise en place de la *Nouvelle Plateforme de Flux*

Avec la mise en place de la *Nouvelle Plateforme de Flux*, la *Brique technique* continue de recevoir les flux venant de *Central P* et d'alimenter les anciens consommateurs. De plus, un système de rebond est mis en place sur la *Brique technique* : lorsque le flux arrive sur la *Brique technique*, il est automatiquement renvoyé vers la nouvelle plateforme, afin que les flux reçus de *Central P* parviennent également à la *NPF* sans subir de modification. La *NPF* reçoit également les flux envoyés par *HR Access* ainsi que par de nouveaux services fournisseurs d'informations. Les logiciels *Central P* et *HR Access* deviennent à leurs tours des services consommateurs d'informations notamment dans le but d'échanger des informations entre eux pour que les deux systèmes se mettent systématiquement à jour pendant la phase transitoire de transfert des applications d'un système vers l'autre. Enfin, les nouveaux services consommateurs sont directement alimentés par la *NPF*. La figure 2, représente le fonctionnement pendant la phase de transition entre l'ancienne et la nouvelle plateforme de gestion des flux.

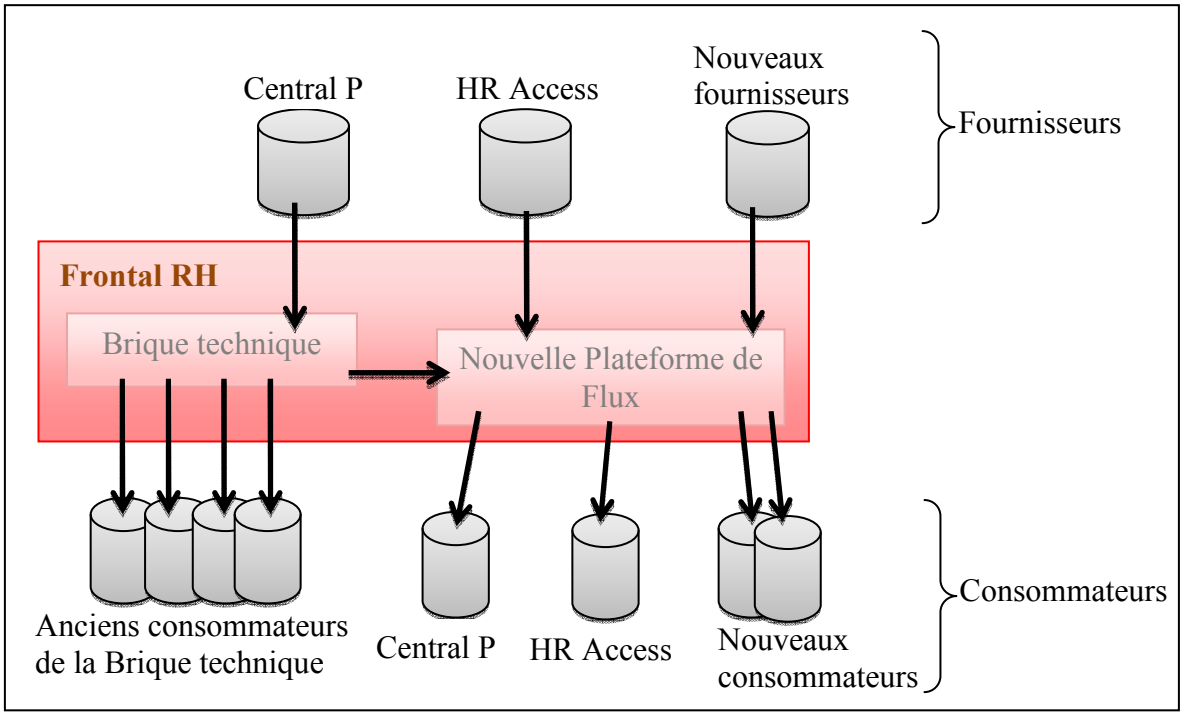

Figure 2 : Echanges de flux après la mise en place de la *Nouvelle Plateforme de Flux*

A plus long terme (plus de quatre ans), la *NPF* viendra remplacer complètement la *Brique technique*. La disparition de *Central P* entrainera une diminution du volume de flux à traiter. Enfin, tous les consommateurs seront directement alimentés par la *Nouvelle Plateforme de Flux* qui devra gérer l'intégralité des flux de données devant être échangés. La figure 3 représente le fonctionnement à long terme de la nouvelle plateforme de flux.

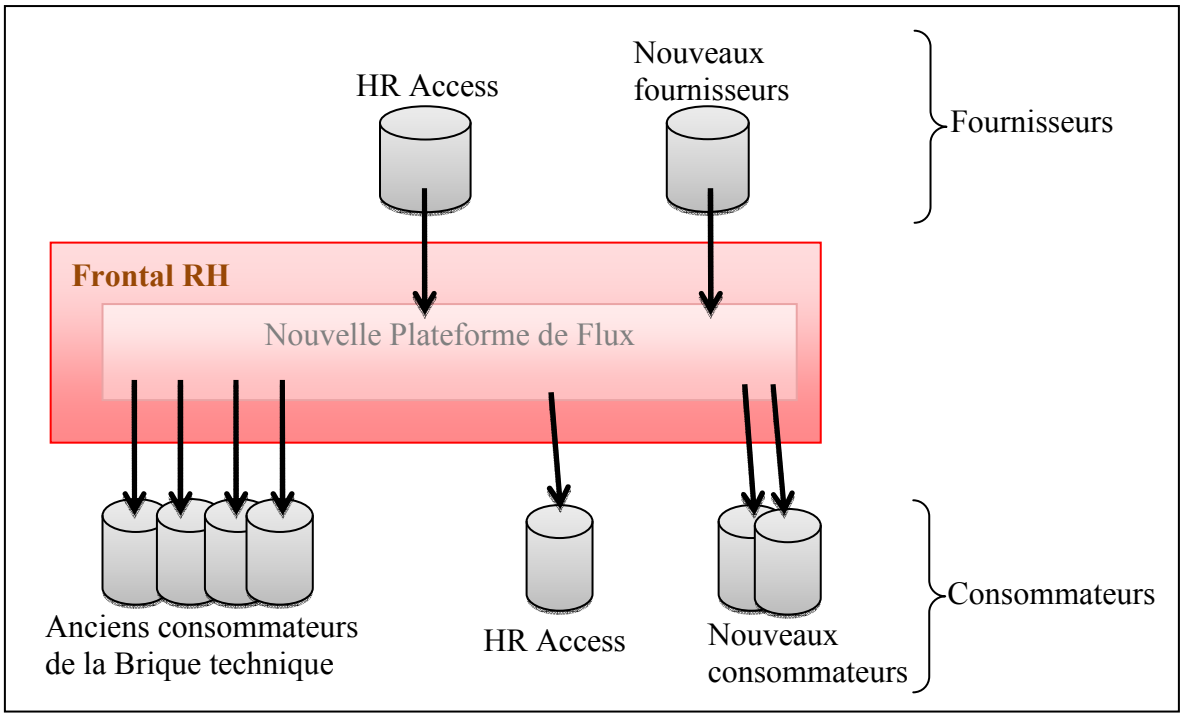

Figure 3 : Echanges de flux à long terme

Le projet global étant d'un volume important en termes d'heures de réalisation, il fut découpé en plusieurs lots. Ce mémoire traite plus spécifiquement le premier lot qui est présenté de manière détaillée au chapitre 3.5.1.

2.2 – Reprise de l'existant

Parmi les actions que *Frontal RH* est capable de réaliser, nous avons déterminé trois catégories afin de mieux les analyser. Nous distinguons ainsi les actions liées à l'acquisition d'un flux, les actions liées à la génération du flux sortant et enfin les actions liées à leur diffusion.

2.2.1 - L'acquisition du flux

Le composant *Frontal RH* n'est capable de récupérer que des flux transmis via CFT[9] grâce au déclanchement de scripts lors de la réception du fichier. Seul le format de fichier CSV[10] avec largeur de champ fixe peut être récupéré et analysé par le *Frontal RH*.

Par ailleurs, un module complémentaire du *Frontal RH,* nommé *COLDIF* (Collecte Diffusion), est capable de recevoir des flux morcelés provenant de plusieurs serveurs différents et de les unifier pour ne former qu'un seul flux contenant l'ensemble des informations afin de pouvoir les traiter comme les autres flux (voir figure 4). Il gère également les contraintes de réception des flux morcelés. En effet, pour pouvoir recomposer un flux, considéré comme cohérent dans son ensemble il est défini des serveurs pour lesquels on doit obligatoirement recevoir des données, et d'autres où il n'y a pas d'obligation de réception.

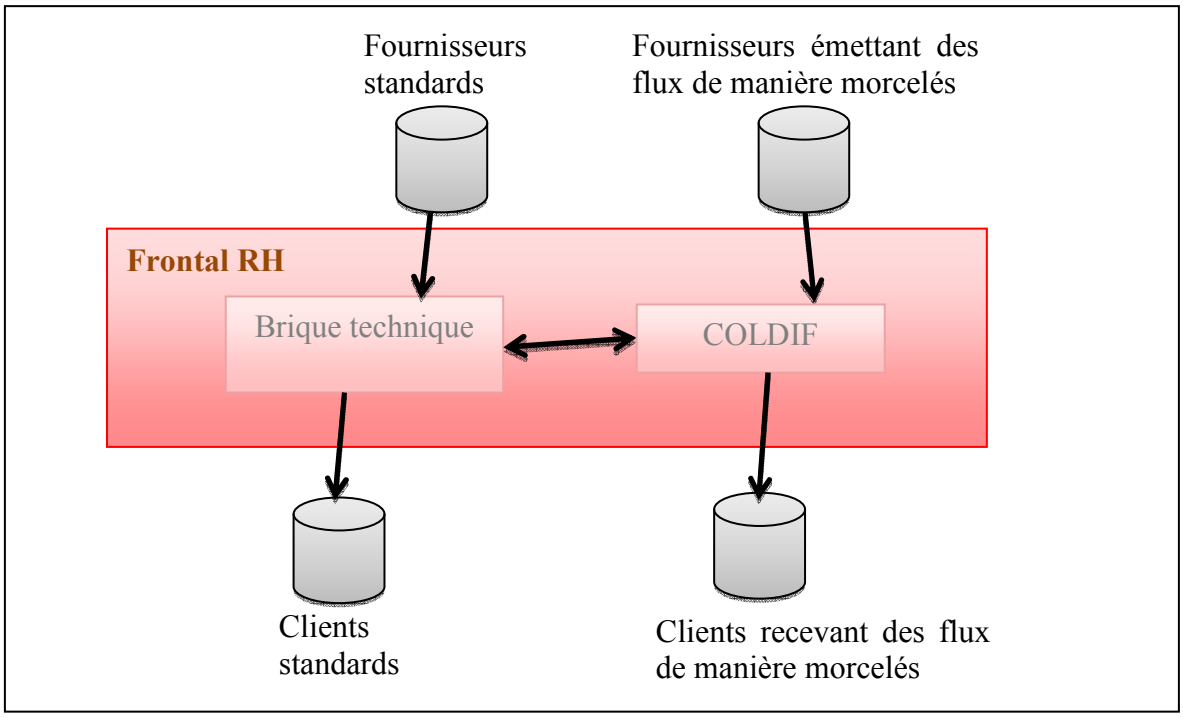

Figure 4 : Schéma représentant l'ensemble des composants du Frontal RH

[9] CFT : (Cross File Transfert) est un logiciel permettant l'échange de fichiers de manière sécurisée. Il utilise notamment un mécanisme d'acquittement lors des échanges afin de valider la bonne réception des fichiers. Le sigle CFT est souvent utilisé en tant que protocole dans l'expression ou la modélisation.

[10] CSV : (Comma-Separated Values) : Format de fichier dont les champs sont séparés par un caractère séparateur étant généralement « , » ou « ; ».

2.2.2 - La génération du flux

Au niveau de la génération du flux, l'application actuelle est capable d'effectuer deux types de filtres sur les données : des filtres horizontaux et des filtres verticaux. Les filtres horizontaux permettent de limiter le nombre de colonnes envoyées à celles uniquement désirées par l'application cliente. Les filtres verticaux permettent de filtrer les lignes de données à renvoyer. Ce type de filtre permet d'effectuer des tests d'égalité, de différence ou d'égalité partielle. L'égalité partielle permet d'identifier une donnée qui, dans son écriture, aurait une partie de sa valeur correspondant à un modèle préalablement déterminé. Il est également possible d'appliquer plusieurs filtres verticaux qui seront concaténés (utilisation uniquement des filtres verticaux en mode cumulatif appelé également mode « ET »).

La génération des flux sortants, peut se faire sous deux formes : le mode complet et le mode différentiel. Le mode complet prend l'intégralité du dernier flux entrant reçu, puis applique les différents filtres paramétrés pour créer le flux désiré par le service client. Le mode différentiel compare le dernier flux entrant reçu avec la version du dernier flux entrant ayant été envoyé au client afin de pouvoir établir une liste des lignes de données ayant évoluées entre les deux envois. La comparaison se fait uniquement sur une liste de colonne de données qui fait partie de la configuration du flux.

Enfin, il convient de souligner que la plateforme de gestion des flux actuelle n'est capable de générer que des flux sous un format de fichier unique, qui comme pour le format en entrée est de type CSV à largeur de champ fixe. Ce format utilise des largeurs fixes pour les champs, mais il ajoute également le caractère « ; » comme séparateur de champ ainsi qu'après le dernier champ.

2.2.3 - La diffusion du flux

La diffusion du flux ne peut se faire que via CFT. Cependant, le module *COLDIF*, permet de décomposer un flux sortant afin de le répartir sur plusieurs serveurs avec leurs spécificités. Ainsi, chaque serveur reçoit uniquement les informations dont il a besoin.

2.3 – Nouveaux besoins

En plus de la reprise de l'existant, il est apparu que certains clients qui ne collaboraient pas avec l'ancienne plateforme de flux, mais également des clients actuels, avaient des besoins plus spécifiques pour lesquels aucune solution n'avait pu être mise en place jusqu'alors. Tout comme pour l'étude de l'existant, la description des nouveaux besoins a été découpée en plusieurs catégories : la collecte des flux, la génération des flux, la diffusion des flux, le routage et enfin les besoins liés à l'architecture de la nouvelle application.

2.3.1 - L'acquisition du flux

Pour l'acquisition du flux, des clients ont exprimé le besoin d'accroître les protocoles de réception des flux. Ainsi, certains souhaitaient pouvoir simplement copier leurs flux dans un répertoire donné afin que la plateforme puisse venir le récupérer. D'autres envisageaient l'utilisation du protocole FTP (File Transfer Protocol) afin de pouvoir envoyer les flux à la plateforme de gestion.

L'étude de besoin, a également permis de mettre en évidence de nouveaux formats de réception de flux. Des clients souhaitent en effet pouvoir utiliser, pour la transmission de leurs flux, des fichiers à largeur fixe, ou encore des fichiers XML[11]. Un client a également demandé qu'il soit possible que l'application vienne directement chercher les données dans sa base de données sous MS ACCESS.

2.3.2 - La génération du flux

La génération doit pouvoir permettre la consolidation des données entre plusieurs tables de données d'un même flux ou entre plusieurs tables de données de flux différents. Ainsi, il est nécessaire de pouvoir réaliser des jointures entre plusieurs tables de données. Pour les jointures d'un même flux, l'étude n'a révélé aucune contrainte.

[11] XML (Extensible Markup Language) est un langage informatique de balisage permettant le stockage d'information de manière structurée.

En revanche, dans le cas de jointures entre deux flux différents, les deux flux peuvent avoir des fréquences de réception différentes. Il a donc fallu ajouter un nombre de jours de décalage personnalisable (au niveau des dates de valeur du flux) entre le flux principal et le flux joint, afin de pouvoir réaliser ces jointures dans toutes les hypothèses envisageables. Cette notion de date de décalage permet ainsi de réaliser une jointure entre un flux daté du jour J et un autre flux daté de J-1.

Un bon exemple d'utilisation des jointures est la génération d'un flux contenant l'adresse principale de chaque agent SNCF. La figure 5 représente ce traitement. Pour cela, nous avons, en entrée, un flux agent composé de deux tables de données distinctes. La première contient la liste des agents avec leur nom, prénom et civilité. La seconde contient la liste des adresses des agents (un agent pouvant avoir plusieurs adresses, dont une principale). Dans la liste des adresses, la commune est représentée par son code INSEE[12]., Ainsi, il faut à partir de ce code, retrouver le nom correspondant. Pour cela, nous recevons également un flux entrant composé de données référentielles dont notamment la liste des codes INSEE des communes. Pour réaliser le flux de sortie, le générateur doit donc, pour chaque agent SNCF, rechercher son adresse principale. Cette jointure des deux tables de données est faite sur le même flux. Puis, pour chaque adresse, il faut rechercher le nom de la commune correspondant au code INSEE. Nous avons ici une jointure entre deux flux de données différents.

[12] Code INSEE : Code numérique élaborée par l'Institut National de la Statistique et des Etudes Economiques

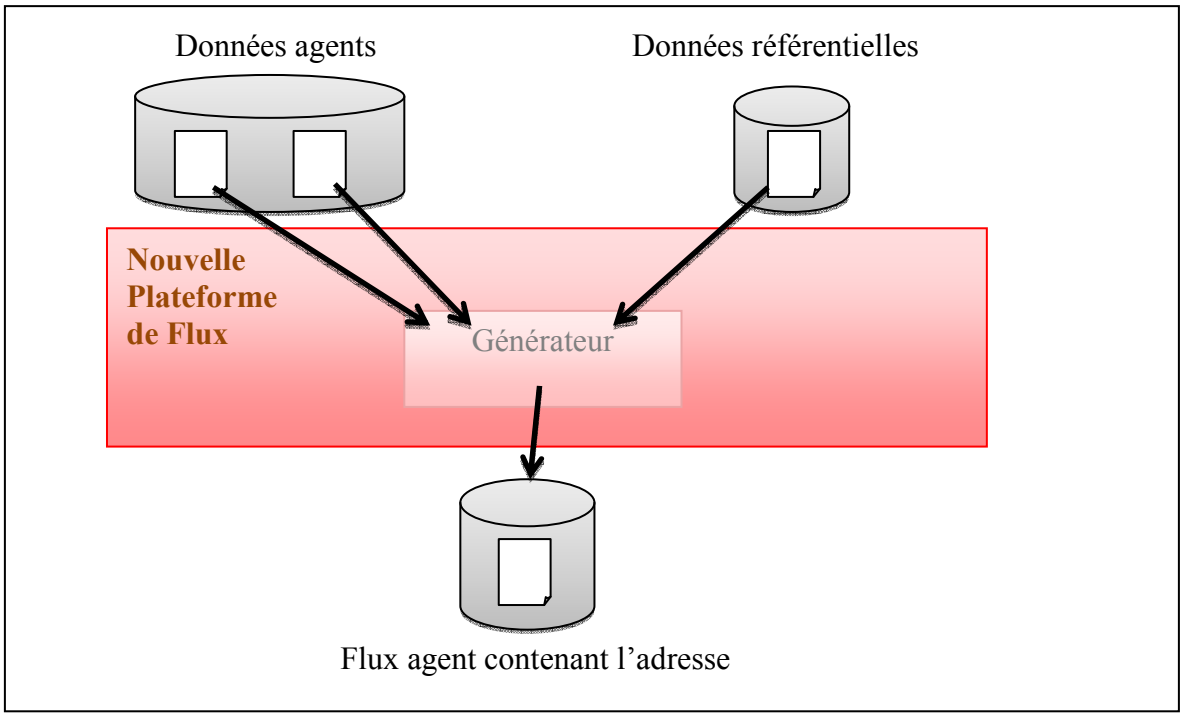

Figure 5 : Schéma représentant la génération de flux avec des jointures entre différentes tables de données

2.3.3 - La diffusion du flux

La diffusion des flux doit non seulement reprendre l'existant mais également pouvoir envoyer un flux généré via FTP. Elle doit aussi pouvoir copier un flux de données directement dans un répertoire, après authentification.

2.3.4 - Le routage de flux

Le routage de flux consiste à pouvoir retransmettre un flux acquis vers des consommateurs sans effectuer de modification sur celui-ci. Cependant, avant de router un flux, il est possible de le contrôler afin de vérifier que sa structure et les données qu'il contient correspondent bien à ce qui est attendu.

Cette fonctionnalité permet de pouvoir utiliser la plateforme comme garant de la bonne distribution des données et de faire en sorte que la plateforme soit vue par toutes les applications comme un BUS[13] central d'intercommunication. Ainsi les applications voulant échanger leurs données avec d'autres systèmes devront les confier à la *NPF* et cette dernière aura en charge de les délivrer au bon destinataire.

2.3.5 - Les demandes d'informations personnalisées via Web-Service[14]

L'étude des besoins a également mis l'accent sur la nécessité, pour plusieurs clients, de pouvoir venir chercher, sur la *NPF* (Nouvelle Plateforme de Flux), certaines informations précises, et ce de manière ponctuelle et synchrone[15]. Ce besoin de mise à disposition de données ne concerne que de petits volumes. Ainsi, la recherche de ces informations doit également être conditionnée et restreinte à l'aide de paramètres transmis par le client lors de sa demande.

Pour pouvoir répondre à ce besoin, nous avons donc choisi de mettre en place un Web-Service. Celui-ci est capable, à partir d'une liste de paramètres fournie par le demandeur, d'effectuer une recherche sur les données les plus récentes que l'on possède sur la NPF. Cette recherche permet de constituer le flux de sortie correspondant au périmètre attendu par le client.

---

[13] BUS : en informatique, ce terme désigne un système de communication entre deux composants. Dans le contexte de ce mémoire, la NPF peut donc être vue par toutes les applications comme un moyen d'émettre et de recevoir des flux de données, sans avoir à se préoccuper de problématiques liées à la capacité d'émission ou de réception de données des systèmes fournisseurs ou consommateurs.

[14] Web-Service : technologie permettant à plusieurs applications de dialoguer et d'échanger à distance, en utilisant des protocoles et des standards connus.

[15] Synchrone : le flux de données attendu par le système demandeur doit être émis en réponse immédiate à sa demande.

2.3.6 – L'acquittement de réception des flux

Pour pouvoir suivre, le plus finement possible, les flux de données après leurs émissions, nous devons être capables de recevoir un message d'un consommateur pour que ce dernier puisse nous confirmer avoir effectivement reçu le flux que nous lui avons envoyé. Afin de pouvoir gérer ces accusés de réception, nous mettons en place un nouveau Web-Service, où chaque consommateur peut nous confirmer la réception des données demandées. L'utilisation de l'acquittement des flux émis est optionnelle pour les clients car nous ne pouvons pas venir impacter le mode d'utilisation des consommateurs déjà existants sur la *Brique technique*.

2.3.7 - L'interface de supervision et d'administration

La nouvelle plateforme de flux doit également posséder une interface permettant l'administration et la supervision des flux.

Concernant l'administration, l'interface doit permettre de pouvoir saisir et modifier toutes les informations liées aux flux à traiter, de la réception de flux de données jusqu'à la diffusion de flux sortants. De manière plus précise, l'interface doit offrir la possibilité d'enregistrer de nouveaux flux entrants comme de modifier les flux entrants existants ainsi que les opérations nécessaires au traitement et au contrôle. Elle doit également gérer la saisie et la modification des flux sortants ainsi que leur diffusion.

La supervision doit permettre d'identifier les flux entrants bien arrivés comme ceux n'ayant pu être reçus. Elle doit aussi offrir la possibilité de voir les flux sortants diffusés correctement de ceux n'ayant pu aboutir. Ainsi, à partir de ces données, il est possible d'établir des statistiques de réussite des flux en fonctions des contraintes des clients. Enfin, pour l'équipe en charge du suivi du traitement des flux, l'interface de supervision doit également permettre de détecter finement tous les problèmes rencontrés lors du traitement des flux afin de pouvoir intervenir rapidement sur la résolution de toute nouvelle problématique pouvant survenir.

Chapitre 3

Choix de solutions et lotissement

Afin de pouvoir répondre aux besoins d'échanges de flux de données entre les diverses applications, plusieurs pistes ont été explorées. Dans un premier temps, l'étude de l'existant fut menée afin de déterminer la faisabilité ainsi que les coûts que pourraient engendrer les évolutions futures des besoins des utilisateurs. Dans un second temps, une étude fut réalisée afin de savoir si des outils, pouvant répondre aux attentes, existaient déjà sur le marché du logiciel. En dernier ressort, une analyse financière a permis de déterminer le budget nécessaire à l'écriture d'une application répondant en totalité, et de manière très personnalisée, aux besoins exprimés. L'étude des besoins a mis en évidence, une contrainte forte concernant la possibilité de pouvoir rajouter, à la demande, un nouveau nœud de traitement s'intégrant à l'ensemble de la solution existante afin de prendre en charge un volume toujours croissant de flux de données. On appelle cela la scalabilité.[16]. Une autre contrainte majeure dans cette recherche résidait dans le fait que certains clients avaient déjà émis des besoins pour de nouveaux flux ne pouvant être couverts par la *Brique technique* et devant être mis en place dès Novembre 2012. Cela a donc impliqué une gestion rigoureuse du planning du projet afin de pouvoir envisager une mise en production pour cette date. Le lotissement du projet a ici permis de respecter les délais imposés.

[16] Scalabilité : de l'anglais « scalability », désigne un système informatique capable d'être réparti sur plusieurs machines afin de s'adapter au volume de traitement à effectuer.

3.1 – Etude des possibilités d'évolution de l'existant

Rappelons qu'actuellement, au sein de la division DSI-T/EH, il existe déjà une plateforme d'échange de flux inter-applicative nommée *Frontal-RH*, celle-ci se compose des deux outils distincts qui sont la *Brique Technique* permettant le traitement des données et d'un autre outil nommé *COLDIF* permettant la collecte et la diffusion de flux morcelés.

Initialement, cette plateforme de flux ne permettait que l'échange de données plus ou moins standardisées. Puis, des besoins plus spécifiques sont apparus et des scripts utilisant plusieurs langages différents ont été ajoutés afin de pouvoir répondre aux différents besoins exprimés. Cet empilement d'évolutions diverses et non coordonnées a ainsi conduit à avoir aujourd'hui un outil qui permet certes de répondre aux besoins des clients actuels, mais rend surtout difficile toute maintenance et complexifie toutes les évolutions que l'on pourrait lui apporter.

Compte tenu de la nécessité de haute disponibilité des flux, le *Frontal-RH* est déjà à bout de souffle concernant la volumétrie des données traitées. La forte hausse du volume de flux de données prévue prochainement suite au changement de progiciel de gestion de la paie, et plus précisément pour la marche en double qui est prévue sur deux années, est problématique. En effet, en aucun cas le *Frontal-RH* ne peut prendre en compte, dans son état actuel, le volume de flux attendu.

Enfin, à chaque modification de configuration de flux, mais également à chaque ajout de partenaire en réception de flux ou en diffusion, cette plateforme nécessite de modifier des scripts, puis de livrer l'application en production. Ce processus induit des livraisons régulières nécessitant, à chaque fois, de réaliser des tests. Cela conduit finalement à pénaliser la réactivité vis-à-vis de la demande.

Aujourd'hui, on constate donc que ce *Frontal-RH* est devenu, au fil du temps, un ensemble technologique hétérogène dont la maintenance devient de plus en plus complexe mais dont la capacité à prendre en compte de nouveaux besoins devient également limitée. A cela, s'ajoute le fait que cette plateforme atteint ses limites dans la gestion d'un volume des flux de données à traiter sans cesse grandissant. Face à ces constats, il paraît donc incohérent de choisir une solution visant à faire évoluer une nouvelle fois l'existant. C'est pourquoi, cette approche n'a finalement pas été retenue.

3.2 – Etude des solutions existantes

Une étude a été réalisée afin de déterminer les coûts que nécessiterait la mise en place d'un nouveau progiciel pour répondre aux besoins identifiés. Cette étude s'est appuyée sur un produit Open Source[17] nommé *Talend Integration Suite* ainsi que sur un produit de l'éditeur *IBM* nommé *DataStage*.

Pour les deux produits, l'étude a abouti à un délai de mise en œuvre important dans le cas d'utilisation de progiciels existants. En effet, l'estimation est de onze à douze mois de mise en place pour une plateforme industrielle de gestion des flux de données mais également de un à deux mois de paramétrage des flux attendus. Il en résulte que la non-connaissance de ces outils en interne fait apparaître un risque de dérapage en termes de délais pouvant alors également engendrer un impact en termes de coûts. Enfin, au niveau du budget, l'achat des licences représente un coût très important. En outre, en sus des licences, il faut prendre en compte la formation des équipes sur le produit et enfin le temps de paramétrage et de mise en place des flux.

Que ce soit vis-à-vis du coût élevé de mise en place, comme de la contrainte de délais qui ne peut être respectée, ces deux solutions ne peuvent être retenues.

[17] Open Source : On parle de logiciel Open Source lorsque la licence de celui-ci respecte les critères établis par l'Open Source Initiative, à savoir le libre accès au code source et la libre redistribution.

3.3 – Création d'une nouvelle application

Enfin, une dernière étude fut menée afin de déterminer ce que coûterait la réalisation, en interne, d'une application qui pourrait prendre en charge l'intégralité des besoins et pourrait s'inscrire parfaitement dans le Système d'Information de la SNCF.

L'avantage de l'écriture d'une nouvelle application, réside dans le fait qu'elle sera à l'image même des besoins exprimés par la maitrise d'ouvrage. En procédant au lotissement[18] de l'application, nous pouvons estimer que la contrainte au niveau du délai de mise en place peut être respectée si on considère que seules les fonctionnalités essentielles au lancement seront présentes dans un premier temps. Enfin, en termes de coûts, cette solution permet de pouvoir totalement les maitriser, car il n'y a ni nécessité de formation des équipes, ni même de licences d'exploitation à acquérir concernant l'application en elle-même. De plus, un macro-chiffrage de la réalisation a fait apparaitre un coût moindre à celui nécessité par l'utilisation de solutions existantes.

[18] Le lotissement, en informatique consiste, à découper une application en plusieurs lots.

3.4 – Bilan de l'étude

Critères	Solution spécifique	Solution Open Source type Talend	Solution éditeur type IBM DataStage	Evolution de l'existant
Couverture fonctionnelle	Oui	Oui	Oui	Oui
Délai de mise en place de la première version	Octobre 2012 à condition de lotir l'application.	Au mieux décembre 2012. La mise au point initiale est lourde. Risque fort de glissement du fait du manque de compétences internes sur le produit. - Difficulté pour apprécier les obstacles que l'on rencontrera, surtout en termes d'architecture - Difficulté pour apprécier la vitesse de montée en compétence des équipes	Au mieux décembre 2012. La mise au point initiale est lourde. Risque fort de glissement du fait du manque de compétences internes sur le produit. - Difficulté pour apprécier les obstacles que l'on rencontrera, surtout en termes d'architecture. - Difficulté pour apprécier la vitesse de montée en compétence des équipes	Difficile à déterminer. Compte tenu des nombreux langages utilisés et de la forte imbrication des différents traitements. Le travail de scalabilité semble titanesque.
Robustesse et scalabilité	La scalabilité de l'application fait partie des besoins initiaux.	Risque fort sur la possibilité de scalabilité de la solution	Conforme à la demande	Difficile de mettre en place une scalabilité compte tenu de la forte dépendance des traitements
Coûts (hors mise en place des flux)	Coût global de cette solution : moyen. Les coûts restent toutefois maitrisables, car il est possible de prioriser les besoins et donc de répartir les coûts sur le temps total du développement de l'application.	Coût important. Celui-ci inclut la formation des équipes et la mise en place de la solution.	Coût très important. Celui-ci inclut les licences, la formation des équipes et la mise en place de la solution.	Coût très important. La structure de l'application est entièrement à revoir et à retravailler.

Figure 6 : Comparaison et récapitulatif des différentes solutions envisagées

Malgré le fait qu'une part importante du traitement des besoins soit déjà existante au sein du *Frontal-RH*, la non-standardisation des développements passés et le fait que tous les traitements se fassent sur les fichiers sans passer par une base de données font que le coût de mise en conformité vis-à-vis du besoin est prohibitif. De plus, l'architecture ne correspondant pas aux attentes de la nouvelle plateforme, la mise en conformité nécessiterait une réécriture de la quasi-totalité de l'ensemble de l'existant.

Les progiciels du marché permettent, certes de démarrer le projet avec une base solide, mais la nécessité de formation des équipes, ainsi que le coût important des licences, couplé avec le temps significatif de préparation en termes de paramétrage des flux, en font une solution entrainant une grande incertitude quant à la réussite du projet. Enfin, le service responsable de la mise en place de l'outil n'ayant jamais travaillé avec les progiciels étudiés, il en résulte une réticence due au manque de connaissance des produits.

En définitif, le choix s'est porté sur l'écriture d'une nouvelle solution. Celle-ci doit être capable de reprendre l'ensemble des fonctionnalités existantes, mais également d'intégrer l'ensemble des nouveaux besoins. Ce choix impose donc d'envisager une solution pouvant évoluer très facilement. Ainsi, l'ajout de nouveaux protocoles ou de nouveaux formats de flux ne doit pas nécessiter d'importantes modifications du cœur de l'application.

3.5 – Planning de réalisation du projet

Afin de pouvoir tenir les délais fixés, les besoins exprimés par la maitrise d'ouvrage ont été répartis sur plusieurs lots. Ainsi, les besoins ont été ventilés dans trois lots en tenant compte de leurs importances et des impacts qu'ils avaient sur les applications tierces devant communiquer avec la *Nouvelle Plateforme de Flux*.

3.5.1 - Lot 1

Le lot 1 prend en compte la mise en place de l'architecture globale de l'application. Cela consiste à rendre l'application scalable et à faire en sorte que les divers modules de traitement puissent communiquer entre eux. Ce premier lot permet de répondre avant tout aux nouveaux flux pour lesquels la *Brique technique* n'est pas en mesure, à ce jour, de fournir une réponse et dont le délai de mise en place est compris entre octobre 2012 et avril 2013.

Cette phase intègre l'acquisition des flux qui ne prend en charge que les flux aux formats CSV, les fichiers à largeur fixe, les fichiers mixtes (fichier à largeur fixe avec séparateur) et les fichiers XML. La génération des flux, quant à elle, prend en compte les modes de génération complets et différentiels avec des filtrages horizontaux et verticaux. Puis, les flux sortants sont à consolider (prise en charge des jointures entre plusieurs flux). Seuls les formats de sortie des fichiers à largeur fixe, CSV, mixte et XML sont disponibles. La diffusion des flux générés ne peut utiliser que le protocole CFT. Enfin, le Web Service d'interrogation personnalisé est disponible.

Durant ce premier lot, l'ancienne application de gestion des flux est conservée, une petite partie des flux existants est transférée vers la nouvelle plateforme de flux et les nouveaux clients devront, dans la mesure du possible, être directement pris en charge par la nouvelle plateforme.

Le planning de cette première version (voir figure 7) a vu sa phase d'étude et de spécifications commencer en décembre 2011 et s'étaler jusqu'à fin mars 2012. Puis, la phase de développement a débuté en mars 2012 pour se terminer début juillet de la même année. Enfin, une phase de recette s'étalant de juillet à septembre 2012 termine ce premier lot pour une mise en production début octobre 2012.

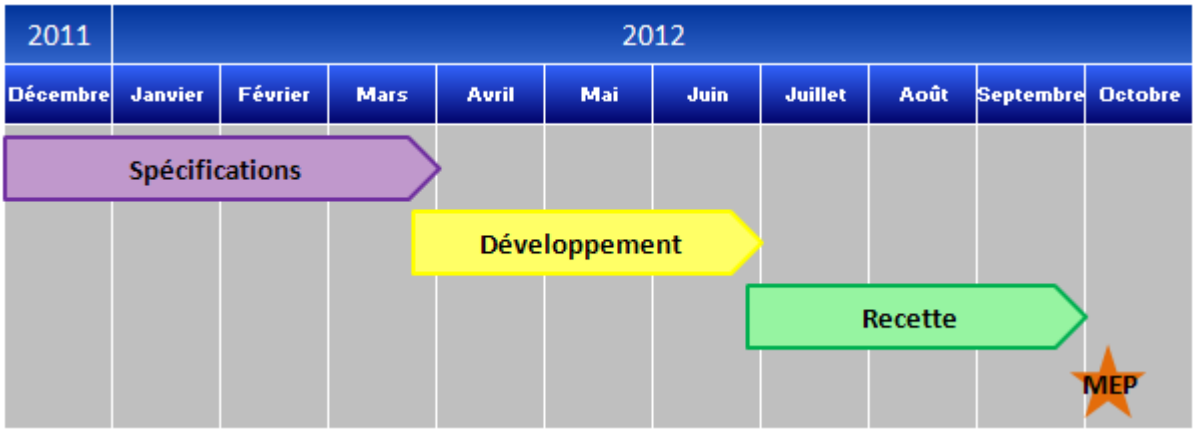

Figure 7: Planning du lot 1

3.5.2 - Lot 2

Le lot 2 vise la mise en place de la consolidation des flux lors de l'acquisition ainsi que la diffusion vers des instances multiples (fonctionnalités reprises de *COLDIF*). Le routage des flux est ajouté. Enfin, les interfaces d'administration et de supervision sont créées.

Ce second lot permet une reprise totale des fonctionnalités de l'ancienne plateforme. Après sa mise en production, les flux de l'ancienne plateforme seront, au fur et à mesure, transférés sur la nouvelle application. Toutefois, une marche en double sera conservée pendant quelques temps afin de s'assurer de la bonne qualité des flux émis. Enfin, pour administrer et superviser l'application, un site web est mis en place pour permettre d'exécuter les tâches de maintenance sans avoir à passer directement par la base de données gérant la *Nouvelle Plateforme de Flux*.

Dans l'immédiat, le planning du lot 2 (voir figure 8) est prévu pour s'étaler de début septembre 2012 jusqu'à la fin du mois de mai 2013. Même si, au niveau de son contenu, ce lot semble déjà bien défini, il peut encore varier sur la durée ou son contenu pour des raisons principalement budgétaires.

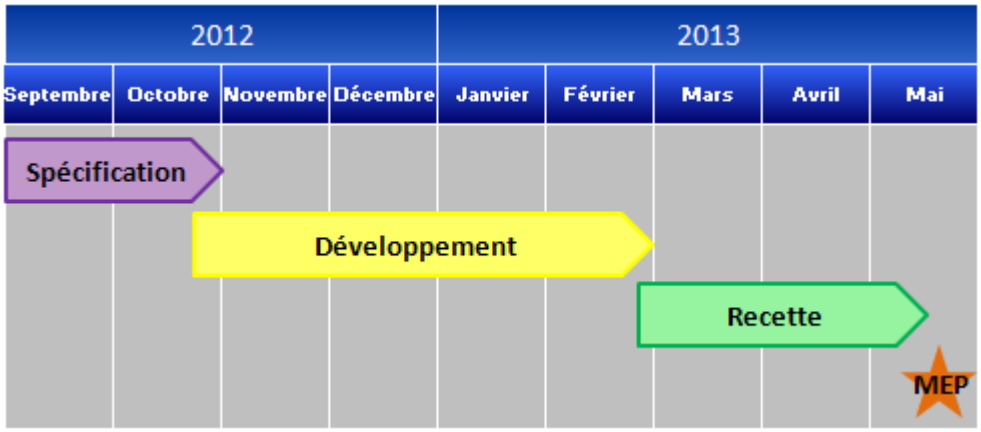

Figure 8: Planning du lot 2

3.5.3 - Lot 3

Le lot 3 vise la mise en place de nouveaux protocoles d'acquisition comme de diffusion de flux dont, notamment, la prise en charge du protocole FTP. La génération des flux sortants est également enrichie par de nouveaux formats ; les formats XLS et PDF ayant déjà été demandés par des clients.

Ce lot a essentiellement pour but de répondre aux besoins de nouveaux clients. Son périmètre n'est pour le moment pas encore arrêté. Néanmoins, un planning indicatif a été établi afin que les futurs clients de la plateforme puissent se préparer et étudier les besoins qu'ils pourraient avoir.

Le lot 3 est donc prévu pour aller du début avril 2013 à fin novembre 2013 (voir figure 9).

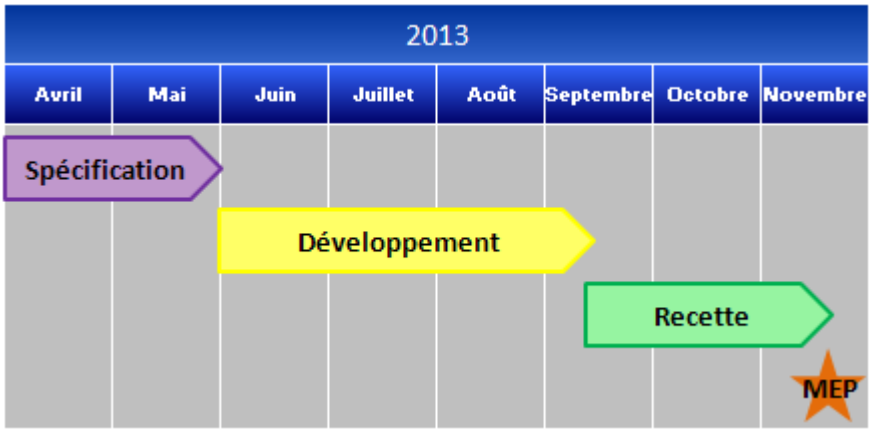

Figure 9 : Planning du lot 3

Chapitre 4

Spécification et conception

Lors de la conception de la nouvelle application ainsi que dans les choix techniques pour son implémentation, il a fallu tenir compte des besoins et nécessités que l'étude préliminaire avait pu mettre en exergue.

4.1 – Spécification de la solution retenue

Pour concevoir la solution à mettre en place, nous nous sommes appuyés sur une liste d'exigences fonctionnelles qui nous avait été remise par la maitrise d'ouvrage (MOA) (voir annexe 1 : « Liste d'exigences fonctionnelles fournies par la MOA pour concevoir la Nouvelle Plateforme de Flux »). Une étude préalable réalisée par des consultants indépendants avait permis de déterminer les exigences fonctionnelles nécessaires à la réalisation de la nouvelle plateforme de gestion des flux. Cette liste avait été remise à la maitrise d'ouvrage qui l'a étudiée, enrichie, puis validée et enfin, nous l'a transmise en tant que cahier des charges pour la réalisation de la nouvelle application.

Afin de pouvoir réaliser l'étude de l'application, nous avons utilisé le langage UML pour spécifier la solution. Pour commencer, nous avons réalisé plusieurs diagrammes de cas d'utilisation. Ce modèle permet d'identifier facilement les différentes fonctionnalités de l'application ainsi que les interactions que celles-ci ont entre elles. Nous avons volontairement regroupés les cas d'utilisation par domaines d'activités afin de voir apparaître toutes les fonctionnalités de chaque module qui compose l'application.

Dans ce diagramme (voir figure 10) on constate plusieurs acteurs. On y retrouve des acteurs cités de manière explicite (Client, Fournisseur, Responsable et Administrateur), mais aussi le système lui-même qui, au travers des échanges entre les différents modules de l'application, devient acteur à son tour.

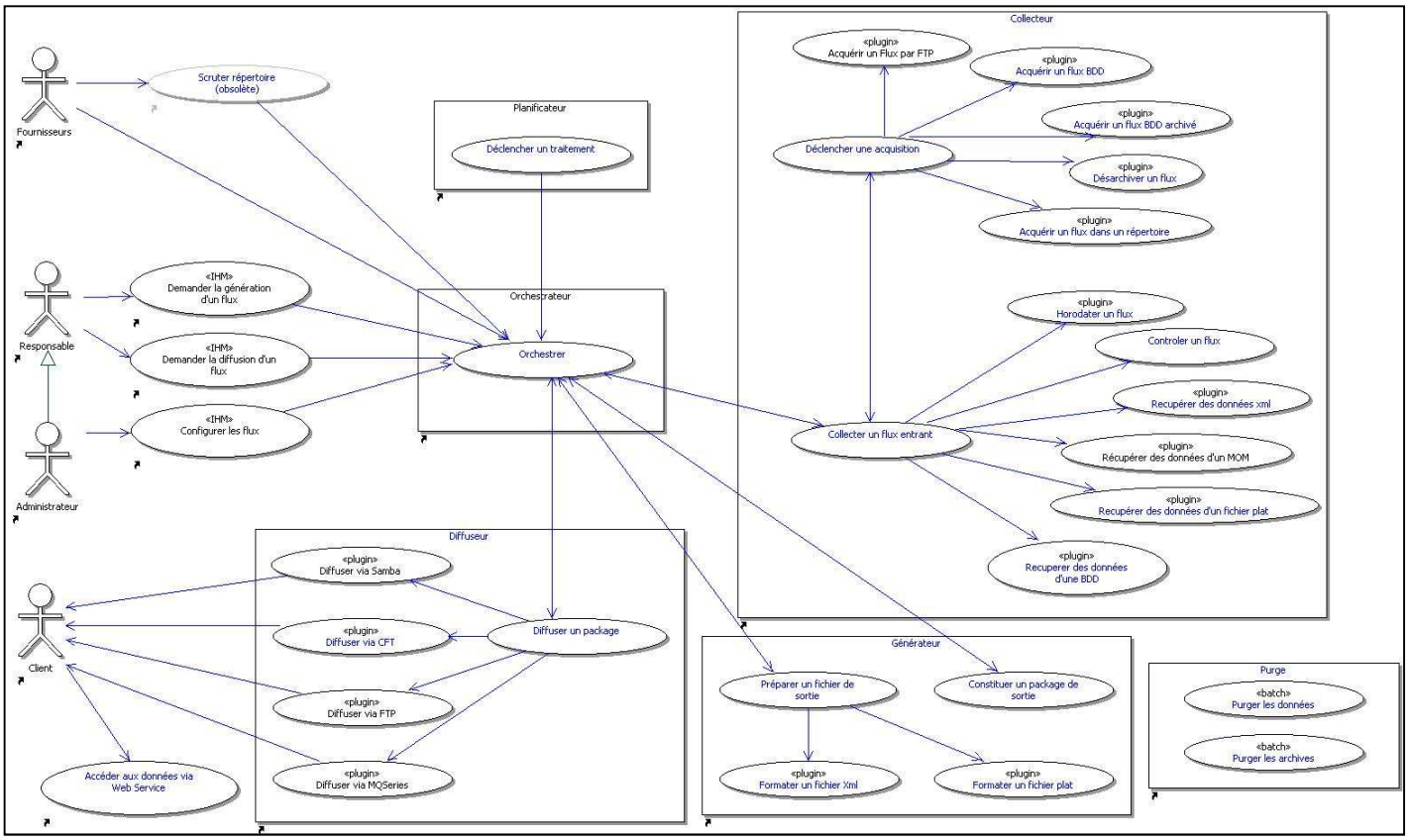

Figure 10 : Diagramme des cas d'utilisation de la nouvelle plateforme de flux

Puis, chaque cas d'utilisation identifié est décrit dans un document dont la structure a été déterminée par le service qualité de la SNCF (voir annexe 2 : Document décrivant le cas d'utilisation « Acquérir un flux dans un répertoire »). Ce document permet de décrire facilement les différents scénarios qui peuvent être observés pour un même cas d'utilisation. Une fois cette description faite, nous avons repris les descriptions des cas d'utilisation pour en réaliser des diagrammes d'activité représentant les différents scénarios (voir un exemple, ci-dessous figure 11, représentant l'acquisition d'un flux dans un répertoire).

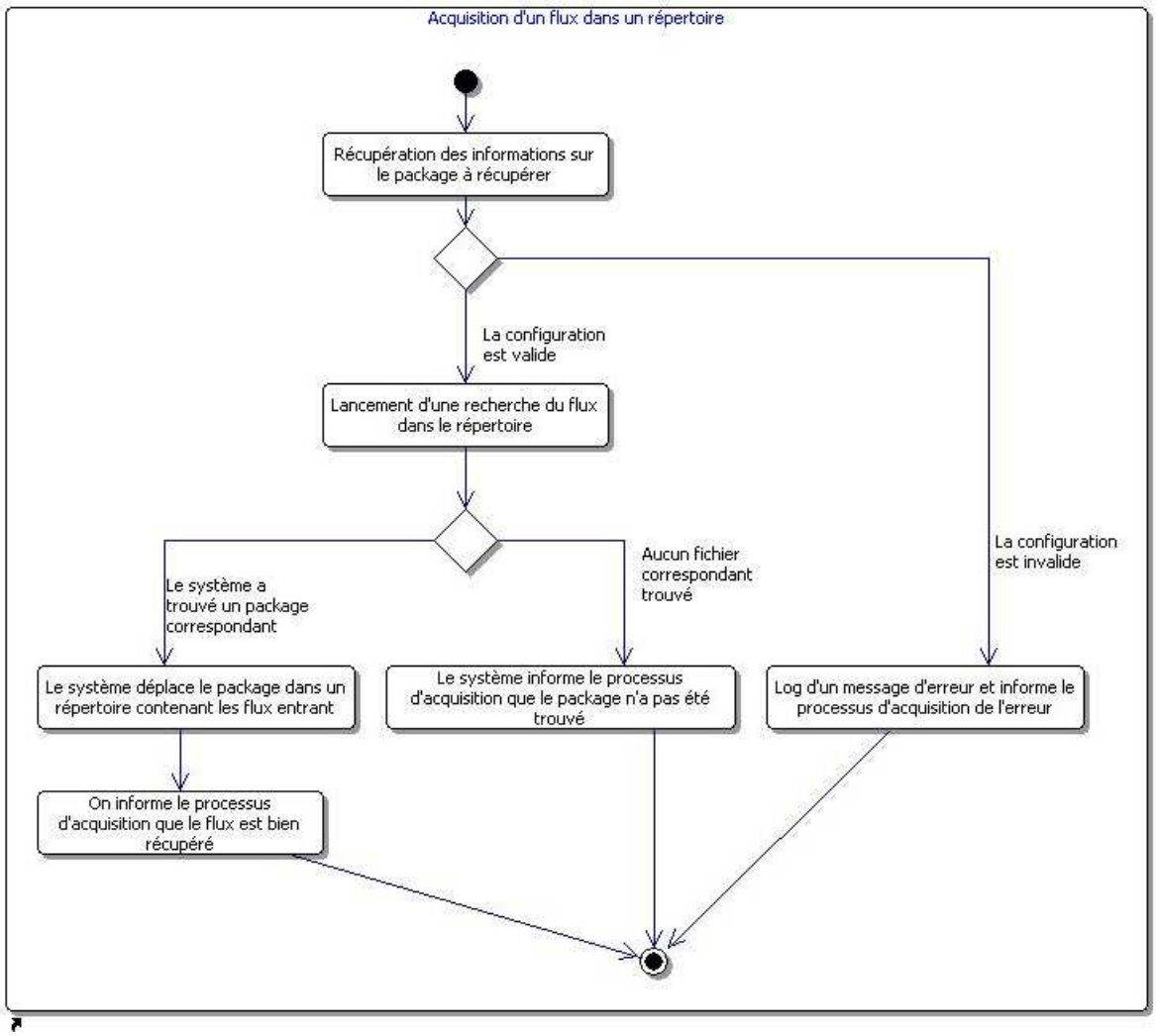

Figure 11 : Diagramme d'activité représentant le cas d'utilisation « acquisition d'un flux dans un répertoire

Enfin, nous avons identifié les interactions entre les acteurs et les différents modules de l'application. Pour cela, nous avons représenté tous les processus de l'application afin de pouvoir faire ressortir clairement les messages qui transitent entre les divers acteurs qui interviennent avec et dans le système. La figure 12 représente les interactions entre les différents acteurs lors de l'acquisition d'un flux.

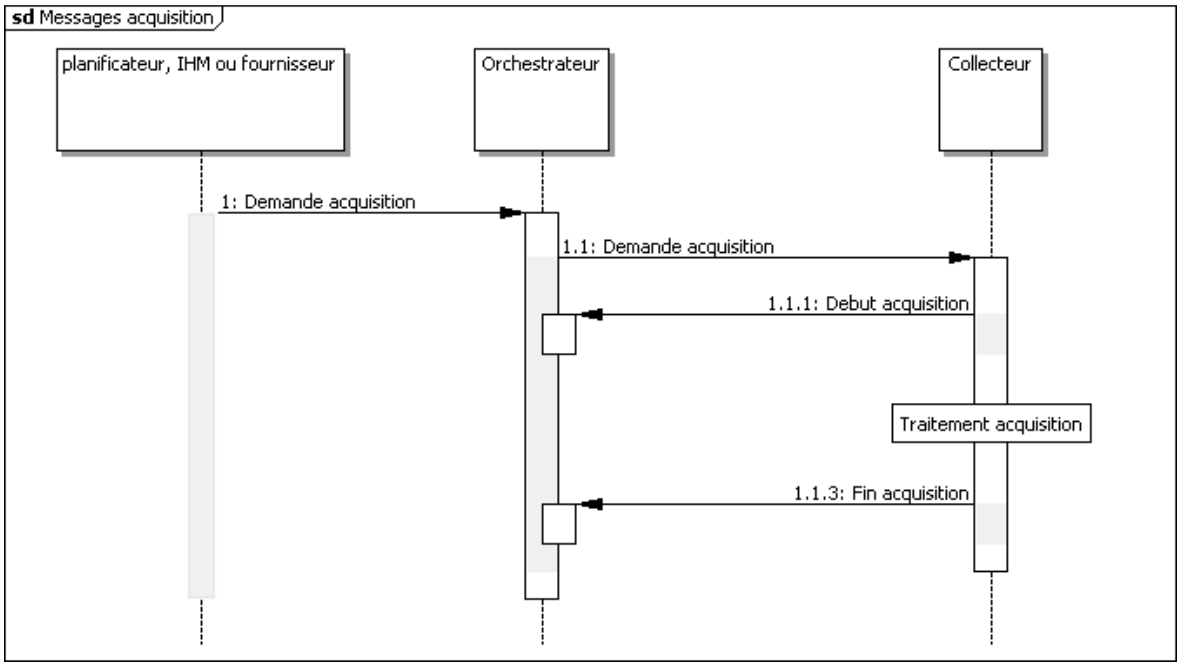

Figure 12 : Diagramme de séquence représentant les interactions lors du processus d'acquisition

Une fois toutes les interactions entre les processus et les processus eux-mêmes décrits, nous avons pu réaliser un diagramme de classes décrivant les différents éléments métiers qui composent l'application. La figure 13 représente le diagramme de classes pour le lot 1.

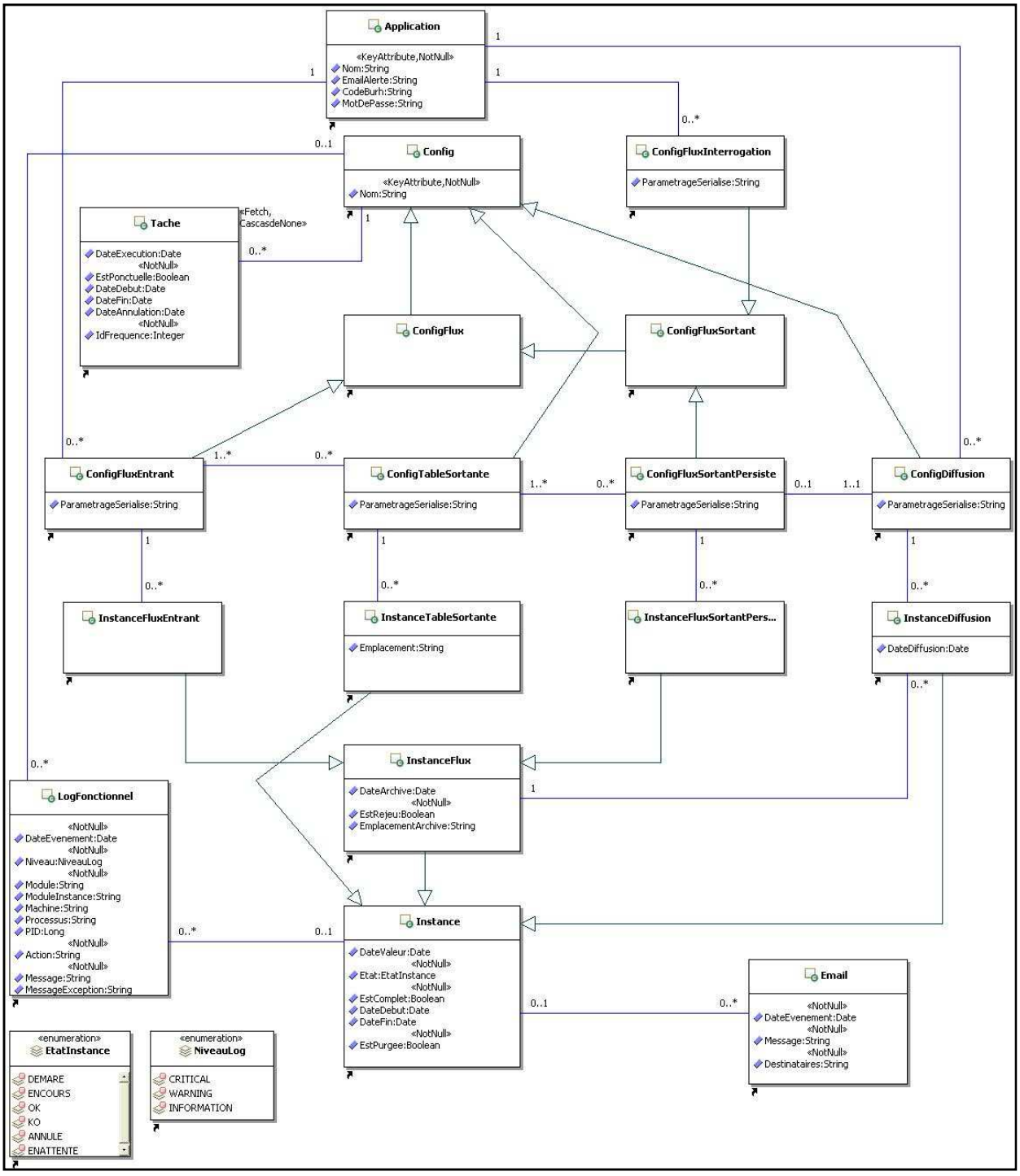

Figure 13 : Diagramme de classes de la nouvelle plateforme de flux (lot 1)

4.2 - Architecture technique et logiciel

Afin de pouvoir assurer une grande flexibilité au niveau des traitements et de garantir une grande personnalisation des flux sortants, sans pour autant nuire aux contraintes de performance, nous avons choisi d'utiliser une base de données, dans l'objectif de stocker les flux entrants une fois les contrôles de cohérence établis. Le stockage des flux permet d'intégrer dans une base, l'ensemble des données qui composent un même flux, tout en conservant la structure qui est définie dans la configuration du flux entrant. L'utilisation d'une base de données pour le stockage des flux entrants, nous permet, dans un premier temps, de ne parcourir le flux entrant qu'une seule fois lors de son intégration, et non à chaque fois que l'on aura besoin de générer un flux de sortie. Enfin, la base de données permet également de pouvoir utiliser le moteur de la base de données afin d'identifier et de filtrer les informations pour la génération des flux sortants. La génération des flux peut ainsi se comparer à un générateur de code SQL qui, à partir d'une configuration, fabriquera une requête SQL plus ou moins complexe pour pouvoir immédiatement identifier les données à envoyer au client.

Pour le choix de l'architecture logicielle, l'application a été découpée en plusieurs modules autonomes pouvant être répartis sur plusieurs machines différentes. Chaque type de module permet de réaliser une partie bien précise du métier du traitement et de la gestion des flux. L'avantage de cette structure modulaire réside avant tout dans la facilité d'adaptation des capacités de traitement de l'application aux besoins réels en déployant simplement une nouvelle instance d'un module à chaque fois que l'on observe un ralentissement des traitements.

Nous avons ainsi décidé de séparer l'application en sept modules différents pour le premier lot :

- **L'orchestrateur :** Il a en charge l'intelligence des traitements. Ce module gère, à partir de demandes ou de réponses d'autres modules, le bon déroulement du traitement des flux.
- **Le planificateur :** Ce module envoie des demandes d'exécution de tâches à l'orchestrateur en fonction d'une planification préétablie. Il a uniquement pour rôle le fait d'avertir l'orchestrateur qu'il est l'heure de réaliser certains travaux.

- **Le collecteur :** Il a pour rôle de récupérer les flux entrants en fonction des protocoles d'acquisition paramétrés, de contrôler ces flux, puis de les insérer en base de données.
- **Le générateur :** Il génère les fichiers correspondants aux flux de sortie. Il crée aussi les packages de sortie (archive contenant les fichiers de sortie) avant leur diffusion.
- **Le diffuseur :** Son rôle est la distribution des archives préalablement créées.
- **Le Web Service d'interrogation :** Ce module, à partir d'une demande personnalisée d'un client, effectue des recherches de données voulues dans les flux entrants, puis renvoie les données au client de manière synchrone.
- **La purge :** L'utilisation d'une base de données permettant le stockage des flux entrants, nécessite également l'ajout d'un processus de purge permettant de pouvoir délester les données jugées trop anciennes. De même, les archives des flux reçus et émis sont supprimées au bout d'un certain temps.

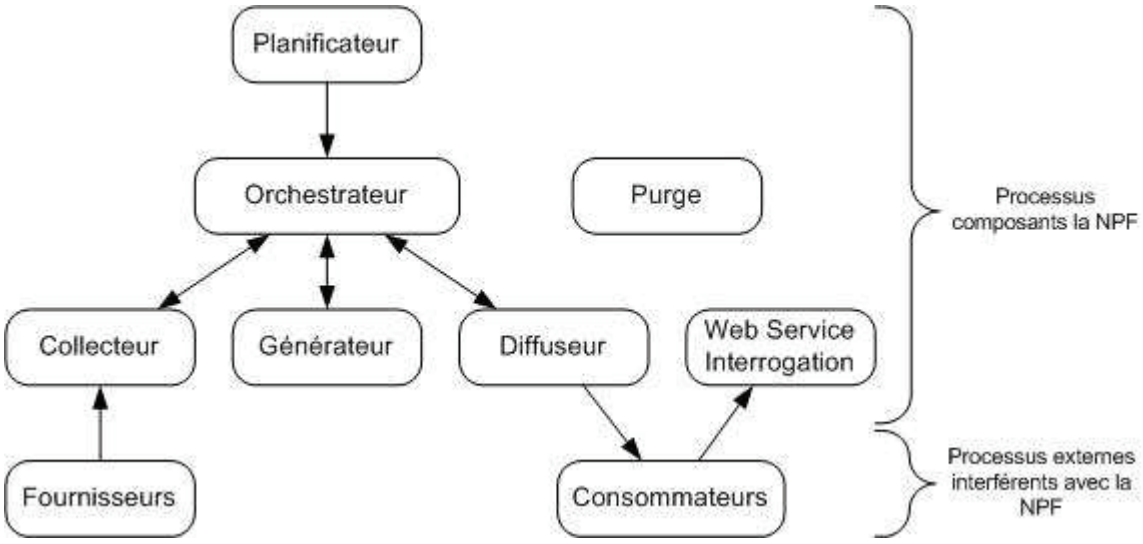

Figure 14 : Architecture logicielle de la Nouvelle Plateforme de Flux

Tous ces modules communiquent entre eux à l'aide d'un outil de gestion de messages de type MOM (Message Oriented Middleware). Les logiciels de type MOM permettent d'échanger des messages entre plusieurs processus en utilisant des files de messages de type FIFO (First In First Out). Ces files de messages nécessitent d'être authentifiées et peuvent également être paramétrées pour être transactionnelles. Le fait que ces files soit transactionnelles, oblige le processus venant chercher un message, à valider la transaction à la fin de son traitement. Dans le cas contraire, un autre processus peut venir prendre le message pour effectuer la tâche que le premier n'aura pu mener à son terme.

Chapitre 5

Réalisation

5.1 - Outil utilisé pour la conception

La SNCF dispose de plusieurs outils pouvant être utilisés pour la conception d'une application. Chaque outil a fait l'objet au préalable d'une étude dans le but de valider ou non son utilisation dans le processus de conception de l'application. Dans le cas de la nouvelle plateforme de flux, nous avons choisi d'utiliser l'outil *Together* développé par l'éditeur Borland. Cet outil permet de réaliser l'ensemble des modèles UML dans la version 2.0. Il a été développé sur la base de l'environnement de développement *Eclipse* connu essentiellement pour le développement *Java*. Grace à cette base sous *Eclipse*, le service expertise de la SNCF a pu développer un générateur de code qui, à partir d'un diagramme de classes, permet de générer l'ensemble des éléments modélisés pour l'application. Cette partie sera développée dans le paragraphe 5.2.4 relatif au StarterKit SNCF 2011.

5.2 – Choix techniques

Les choix techniques furent fortement impactés par les connaissances de l'équipe ayant en charge la réalisation du projet, mais également par les habitudes du service DSI-T/EH/A en termes de technologies comme de méthode. Cependant, des contraintes externes telles que la nécessité d'une haute disponibilité, exprimées sous forme d'exigences, ont également orientés voir impactés de manière significative, les choix que nous avons pu faire.

5.2.1 - Choix du langage de programmation

Le service ayant en charge la réalisation de la *Nouvelle Plateforme de Flux*, à savoir la DSI-T/EH/A, étant composée, dans sa quasi-totalité, de personnes ayant l'habitude de produire et maintenir des applications sous le langage C# du Framework .NET, ce langage fut donc logiquement choisi comme étant celui que nous devions utiliser. Concernant la version du Framework, nous avons choisi d'utiliser la version 4 du Framework .NET en raison des nouvelles fonctionnalités et améliorations qui lui ont été apportées. En effet, les améliorations au niveau des performances du Framework 4 permettent de paralléliser automatiquement certaines actions grâce au « Parallel Computing[19] » ; notamment sur les boucle de type « For ». D'autre part, le Framework .NET, dans sa version 4, prend en charge la covariance, élément fondamental pour la réalisation de notre solution. Nous verrons, plus en détail, dans le paragraphe 5.3.3, l'importance de la covariance pour le paramétrage de l'application.

Afin de pouvoir développer l'application avec le Framework .NET 4, nous avons utilisé l'environnement de développement Visual Studio 2010 créé par Microsoft. Cet outil permet de développer des applications Web, des applications bureautiques, des applications mobiles et enfin des librairies.

5.2.2 - Choix de la base de données

Dans l'architecture technique de l'application, nous avons fait le choix d'utiliser une base de données dans laquelle seront stockés les flux entrants dans le but d'accélérer la génération des flux sortants qui devront être personnalisés en fonction des destinataires. La *Nouvelle Plateforme de Flux* doit avoir une haute disponibilité. Sa base de données doit donc avoir de la résistance aux pannes.

[19] Parallel Computing : permet d'exécuter du code simultanément, sans que le développeur n'ait à le faire expressément. A l'exécution, le Framework .NET estime lui-même le nombre d'éléments qu'il peut exécuter en parallèle.

Elle doit également être capable de pouvoir gérer un grand volume de données. En effet, la base de données doit être capable de stocker trois mois d'informations. Cette durée de rétention des données est la plus petite que nous puissions utiliser. En effet, pour déterminer cette durée, nous nous sommes basés sur le fait que la réception des flux ayant la plus petite fréquence de réception concerne des flux mensuels. De plus, nous devons avoir en mémoire au minima les deux dernières versions des flux pour pouvoir déterminer les différences qui composent le flux afin de pouvoir réaliser des émissions de flux en mode différentiel. Nous devons également conserver une marge de sécurité en cas d'anomalie dans les flux réceptionnés comme pour ceux émis.

Enfin, avant la mise en place d'*HR Access*, nous estimons les flux en réception à environ 150 Mo de données compressées au quotidien, et à environ 300 Mo de données compressées supplémentaires le weekend. Rapidement, avec l'arrivée de l'application *HR Acess* parmi les fournisseurs, ce volume devrait à minima tripler.

Pour toutes ces raisons, nous avons décidé d'utiliser la base de données nommée Oracle RAC (Real Application Clusters). La version RAC d'Oracle permet la prise en compte de cluster[20] de base de données ; ce qui permet, avant tout, de pouvoir disposer de plusieurs machines qui vont pouvoir accéder en même temps à une même base de données. Cela permettra alors, en cas de panne d'un nœud, de pouvoir automatiquement équilibrer la charge sur les autres nœuds du cluster sans que le processus utilisant la base de données en ait connaissance.

5.2.3 - Choix de l'outil pour la communication des différents processus de l'application

Pour que les différents modules qui composent la *Nouvelle Plateforme de Flux* puissent communiquent entre eux, nous avons précédemment évoqué l'utilisation d'un outil de gestion de message de type MOM (Message Oriented Middleware).

[20] Un cluster est une grappe de machines (au minimum 2) permettant la répartition de charge et/ou la continuité de service.

Microsoft a développé un outil de type MOM nommé MSMQ (Microsoft Message Queuing) parfaitement intégré avec la technologie .NET et notamment le langage de programmation C#. Il permet d'échanger des messages entre les différents modules pouvant se trouver sur des machines différentes grâce à une gestion de files d'attente de type FIFO (First In First Out). Pour gérer l'envoi de messages à destination de MSMQ, ainsi que leurs réceptions, le Framework .NET, depuis sa version 3.0, contient ce qui se nomme WCF (Windows Communication Foundation). WCF est une couche d'abstraction qui simplifie la mécanique d'intégration, notamment avec MSMQ. Ainsi, le développeur n'aura pas à se soucier des aspects de réception et d'émission des messages, mais uniquement de l'aspect orienté métier de l'application à développer.

5.2.4 - Utilisation du StarterKit SNCF 2011

Le service DSI-T/E7 de la SNCF (service dédié à l'expertise méthodologique et l'expertise logicielle) met à disposition un StarterKit en langage .NET permettant de pouvoir rapidement débuter la réalisation d'une application.

Ce StarterKit se compose de :

- **Un projet de base** qui offre les services de base pour débuter une application, des exemples de codes, des exemples d'implémentations de classes objet, des exemples de tests unitaires automatisés.
- **Un générateur de code** permettant, à partir d'un diagramme de classes modélisé avec l'outil Together, de générer le code représentant les classes modélisées ainsi que les services minimums d'accès à la base de données correspondant aux classes, tels que les services CRUD (Create Read Update Delete[21]).

[21] CRUD : Ces méthodes sont les méthodes de base permettant de créer des informations en base de données, pour les lire, les mettre à jour et enfin les supprimer.

- **Un hébergement sous une plateforme d'intégration continue :** cela permet de suivre facilement la qualité du logiciel en cours de réalisation. La plateforme d'intégration continue récupère chaque nuit la dernière version du code source de l'application, puis compile les sources pour contrôler qu'il n'y ait pas d'erreur. Ensuite, elle joue l'ensemble des tests unitaires automatisés pour s'assurer qu'il n'y a pas de régression dans le code. Elle utilise également d'autres outils permettant d'évaluer la qualité du code. Enfin, elle compose un rapport disponible en intranet, qui reprend tous ces éléments et permet ainsi de suivre facilement l'évolution des sources, mais surtout la qualité du code développé.

Pour le mode d'interrogation d'une base de données, ce StarterKit laisse le choix entre l'utilisation de l'ORM (Object-Relational Mapping[22]) NHibernate et ADO.NET (ActiveX Data Objects[23]). L'utilisation de NHibernate permet de définir la correspondance entre la base de données et les objets qui composent le domaine de l'application. L'ADO.NET, quant à lui, utilise le langage de requête SQL afin d'interroger la base de données. Il faudra alors ensuite charger l'objet correspondant aux données récupérées.

Dans notre cas, nous avons choisi d'utiliser les deux méthodes simultanément. Le Framework NHibernate est utilisé pour l'accessibilité aux données qui composent le domaine métier de l'application permettant de définir les flux et leur structure. L'ADO.NET est utilisé pour les données issues des flux entrants. L'utilisation des deux méthodes nous permet, avant tout, de gagner du temps pour les développements touchant le domaine de l'application (les classes métiers modélisées), mais également de pouvoir concevoir des requêtes SQL complexes. Ceci permet de délester le travail de sélection, de tri et de filtrage des données qui doivent composer les flux de données sortants, vers la base de données.

---

[22] ORM : Outil permettant de créer l'illusion d'utilisation d'une base de données orientée objet à partir d'une base de données relationnelle. L'outil permet de pouvoir définir des correspondances entre les tables d'une base de données et les objets implémentés par un programme.

[23] ADO.NET : est une bibliothèque logicielle fournit par Microsoft fournissant une interface d'accès aux données.

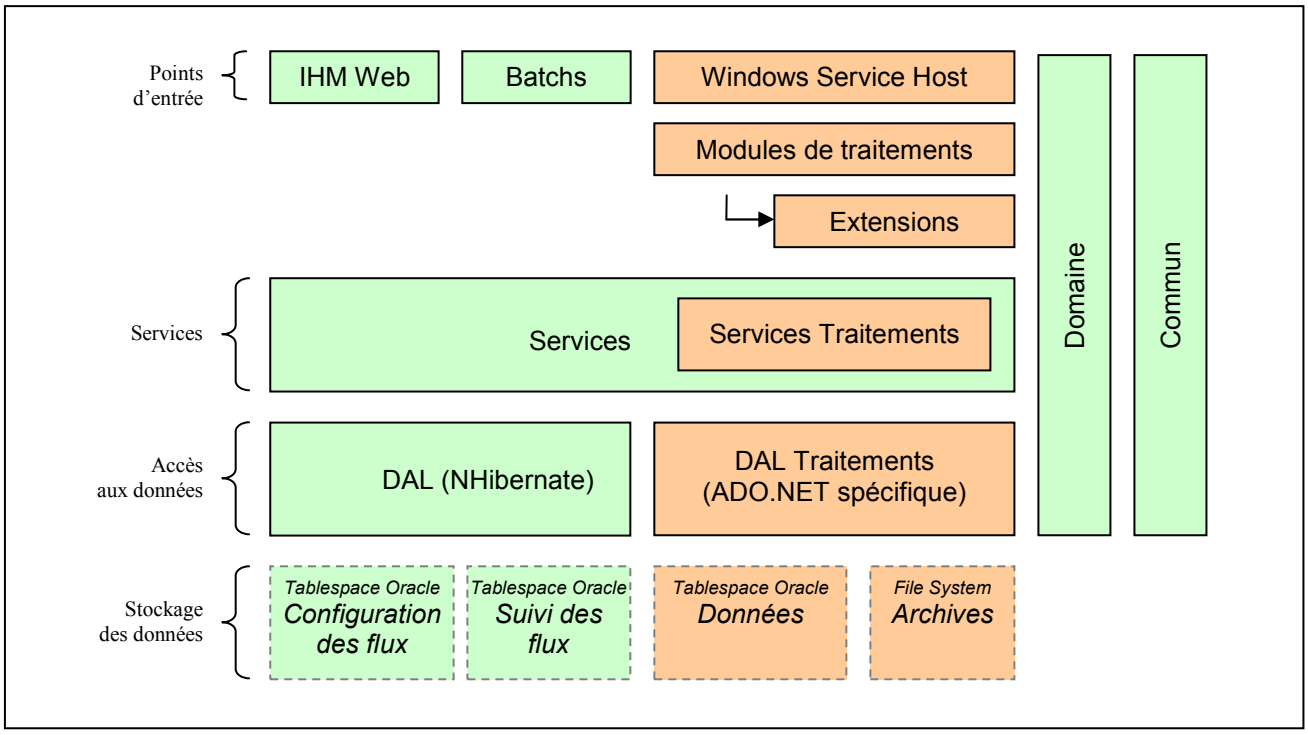

Figure 15 : Architecture applicative retenue de l'application

5.2.5 - Utilisation des Tests Unitaires Automatisés

Dès le lancement du projet, nous avons voulu mettre un point d'honneur à veiller à l'aspect qualitatif du projet. Pour cela, en plus d'une relecture croisée, du code source développé, entre les développeurs, nous avons volontairement voulu que des tests unitaires automatisés puissent couvrir la plus grande partie possible de l'application. Les tests unitaires automatisés permettent d'effectuer des tests sur les différentes fonctions qui composent une application. Pour cela, les fonctions sont isolées du reste du code afin de pouvoir valider leur bon fonctionnement. Enfin, l'ensemble de ces tests sont exécutés à intervalles réguliers (généralement de manière quotidienne) afin de s'assurer qu'avec l'avancée du développement, on ne génère pas de régression dans le fonctionnement de l'application.

Nous avons tenu compte du surcoût de développement lié à ce choix de tests, dès le chiffrage de l'application. Ainsi, et dans la mesure du possible, chaque développeur doit en même temps qu'il développe une fonctionnalité ou une méthode, créer les tests unitaires automatisés permettant de valider son travail.

Pour la mise en place des tests unitaires automatisés, le StaterKit 2011 utilise les Frameworks suivants :

- **Rhino Mock :** Il s'agit d'un Framework de simulation permettant de bouchonner les appels d'une méthode vers d'autres méthodes du code source. Le bouchonnage permet d'isoler le code source à tester afin d'être certain qu'aucune erreur potentielle, détectable lors du test, ne puisse provenir d'un autre bloc de code, externe à celui testé (voir un exemple en figure 16).
- **NDBUnit :** Ce Framework permet de pouvoir définir le contenu de la base de données pour les tests dans des fichiers de paramétrage. A chaque exécution des tests, la base est chargée avec les données contenues dans les fichiers de paramétrage, puis le test est exécuté et enfin la base est vidée. Ce Framework permet de pouvoir tester les blocs de code effectuant des traitements sur la base de données.

L'utilisation des tests unitaires automatisés permet, dans un premier temps, de détecter toutes les anomalies pouvant se situer dans le code source à tester, mais également, de pouvoir détecter les éventuelles régressions pendant le cycle de développement lui-même. Ainsi, quand un développeur effectue une modification sur un bloc de code existant, il doit également rejouer les anciens tests existants afin de pouvoir valider qu'il n'y a pas d'impact sur le fonctionnement attendu du code source.

```
/// <summary>
/// Cas nominal une config de flux valide correctement une application
/// </summary>
[TestMethod]
public void ControleDroitApplicationSurFluxInterrogation()
{
    service.ConfigSrv = mocks.CreateMock<ConfigSrv>();
    string nomFlux = "essaiFlux";
    string nomApplication = "applicationNom";

    using (mocks.Record())
    {
        Expect.Call(service.ConfigSrv.ChargerParCleFonctionnelle(nomFlux)).Return(
new ConfigFluxInterrogation(1)
        {
            Nom = nomFlux,
            ParametrageSerialise = "essai",
            ObjApplication = new Application(2)
            {
                Nom = nomApplication,
                MotDePasse = "toto"
            }
        });
    }
    using (mocks.Playback())
    {
        bool result = service.ControleDroitApplicationSurFluxInterrogation(
nomApplication, nomFlux);
        Assert.IsTrue(result, "Le service doit valider les droits de l'application
connectée");
    }
}
```

Figure 16 : Exemple d'un test unitaire automatisé utilisant le bouchonnage

5.2.6 - Configuration évolutive des flux

Un des besoins les plus importants et ayant un fort impact sur les choix de l'architecture comme des spécifications, est la grande facilité de l'application à pouvoir évoluer. Pour cela, lors de la phase de conception, nous avons décidé de stocker en base de données les configurations des flux sous un format sérialisé[24]. Ainsi, le diagramme des classes UML du projet s'en trouve grandement simplifié (voir figure 13) et, seuls les objets rentrant dans la composition de la configuration des flux, doivent implémenter leur désérialisation. Le processus de désérialisation sera détaillé dans le chapitre 5.3.1.

[24] Un format sérialisé est la représentation d'un objet (au sens codage informatique) sous la forme d'une structure de données établie, qui le plus souvent utilise le format XML.

5.3 – Difficultés rencontrées

Tous au long du cycle de développement, nous avons été confrontés à des difficultés techniques. Certaines difficultés nous étaient imposées par les exigences émises par la maitrise d'ouvrage et d'autres étaient des contraintes techniques que nous avons découvertes lors du développement de l'application. Pour toutes difficultés rencontrées, nous avons dû trouver, ou parfois aussi imaginer, des solutions à apporter. Ces difficultés ont essentiellement été dues au fait de vouloir concevoir une application extrêmement souple pour l'ajout de nouvelles fonctionnalités ; ce qui nous a poussés à utiliser des configurations sérialisées en base de données, à pouvoir manipuler des objets non spécialisés dans les zones communes de traitement, comme à utiliser des modules spécialisés indépendants. Enfin une difficulté concernant les transactions, fut également générée par l'architecture même de l'application.

5.3.1 - La désérialisation des configurations

Afin de faciliter les futures évolutions de l'application, les configurations des flux précisant les traitements à effectuer sur les données et la structure de sauvegarde de la configuration des flux en base de données doivent être les plus génériques possibles. Ainsi, afin de pouvoir rendre les configurations génériques, nous avons opté pour la sérialisation des configurations. Chaque configuration prend alors la forme d'un fichier XML qui sera stocké sous la forme d'une chaine de caractères en base de données. Le diagramme des classes concernant la partie stockant les configurations (figure 17), s'en trouve ainsi simplifié.

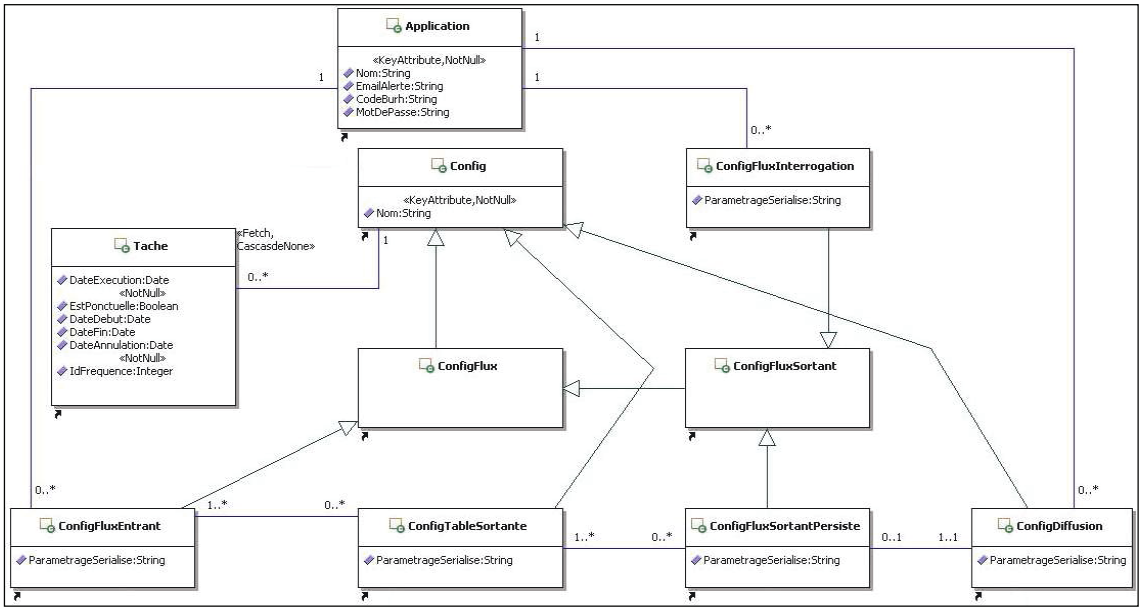

Figure 17 : Diagramme de classes représentant uniquement la partie configuration de l'application

Après étude de la solution à mettre en place, nous avons identifié cinq types de configurations différentes :

- **La configuration des flux entrants :** Cette configuration contient le protocole de réception du flux, la fréquence de réception du flux, le format de réception des données et enfin la structure du flux reçu (tables de données qui composent le flux entrant ainsi que les colonnes de données qui composent les tables).
- **La configuration des tables sortantes :** Cette configuration contient les colonnes de données qui la composent, le format de sortie et enfin, les traitements que l'on doit réaliser pour effectuer la sélection des données.
- **La configuration des flux sortants persistés :** Cette configuration permet de générer le flux à envoyer. Elle contient la liste des tables sortantes qui composent le flux, le format du flux à envoyer (ZIP, TAR, RAR), le nom du fichier qui doit être envoyé, et enfin la fréquence de génération du flux ainsi que son mode de génération (génération en mode complet ou en mode différentiel).

- **La configuration de la diffusion :** Cette configuration contient le protocole de diffusion à utiliser pour envoyer le flux au destinataire.
- **La configuration du flux interrogation :** Cette configuration contient le descriptif des traitements à réaliser pour générer les données de sorties mais également une description des paramètres attendus lors de la demande via Web Service.

Ces cinq configurations ont été séparées au niveau de leur stockage en base de données car elles correspondent à cinq activités différentes pour l'application. Cependant, toutes ces configurations sont stockées en base de manière sérialisée et le procédé de dé-sérialisation est identique pour chaque configuration. La figure 18 représente un exemple de configuration de flux entrant sérialisé, tel qu'il est stocké en base de données.

```
<?xml version="1.0" encoding="utf-8"?>
<ParametrageFluxEntrant xmlns="http://Npf/Parametrage.xsd"
Type="Sncf.Dsit.Npf.Traitements.ExtensionsFichier.ProtocoleFichier,
Sncf.Dsit.Npf.Traitements.ExtensionsFichier, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null">
  <Acquisition>
    <Protocole
Type="Sncf.Dsit.Npf.Traitements.ExtensionsFichier.ProtocoleRepertoire,
Sncf.Dsit.Npf.Traitements.ExtensionsFichier, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null">
      <ProtocoleRepertoire Nom="perssone.zip" EstCompresse="true" Format="Zip"
MotDePasseArchive="" Chemin="C:\" />
    </Protocole>
    <Frequence DateDebutPlanification="2012-02-12" FrequenceNb="2"
HeureDebutReception="11:12:00" DureeReception="PT2H35M" >
      <FrequenceTypeVariable>
        <ListeJoursVariables>
          <JourVariable Mode="PremierJourMois" Jour="Lu" Nombre="2" />
        </ListeJoursVariables>
      </FrequenceTypeVariable>
    </Frequence>
    <Horodatage
Type="Sncf.Dsit.Npf.Traitements.ExtensionsHorodatage.HorodatageParDate,
Sncf.Dsit.Npf.Traitements.ExtensionsHorodatage, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null">
      <HorodatageParDate NbJourDecalage="1" />
    </Horodatage>
  </Acquisition>
  <TablesEntrantes>
    <TableEntrante
Type="Sncf.Dsit.Npf.Traitements.ExtensionsFichier.TableEntranteFichierLargeurFixe,
Sncf.Dsit.Npf.Traitements.ExtensionsFichier, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null">
      <TableEntranteFichierLargeurFixe Nom="Personne" Encodage="UTF-8"
Separateur=";" EstPersiste="true" EstControle="true" NbLigneMin="0" NbLigneMax="0"
NomFichier="personne.csv" PossedeEntete="false" RetourChariot="CrLf" >
        <ColonnesEntrantes>
          <ColonneEntrante
Type="Sncf.Dsit.Npf.Traitements.ExtensionsFichier.ColonneEntranteFichierLargeurFixe,
Sncf.Dsit.Npf.Traitements.ExtensionsFichier, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null"     >
            <ColonneEntranteFichierLargeurFixe Nom="colonne1" TypeDonnee="String"
Longueur="50" FormatChampBase="nom" FormatDonneeEntrante="" EstCle="false"
EstNullable="false" EstIndexe="false" ValidationExpression=""
SuppressionEspaceFinChaine="false" SuppressionEspaceDebutChaine="true"
VideEstNull="true" LargeurColonne="12" />
          </ColonneEntrante>
        </ColonnesEntrantes>
      </TableEntranteFichierLargeurFixe>
    </TableEntrante>
  </TablesEntrantes>
</ParametrageFluxEntrant>
```

Figure 18 : Exemple d'une configuration sérialisée

Chaque classe pouvant être sérialisée, implémente elle-même sont comportement de désérialisation. Pour l'implémentation de la désérialisation on note deux méthodes distinctes. La première méthode consiste à préciser le comportement par annotations. On définit la classe pouvant être désérialisée par l'annotation [Serializable]. Les propriétés peuvent aussi être annotées afin de définir très précisément leur traduction une fois sérialisées. Cette méthode est utilisée dans les cas les plus simples. La figure 19 représente un exemple de désérialisation par annotation d'une classe « Frequence » ayant une propriété « DateDebutPlanification » de type « DateTime ».

```
/// <summary>
/// Définition de la fréquence
/// </summary>
[Serializable]
public class Frequence
{
    /// <summary>
    /// Date de début de la planification => permet de calculer la première
exécution de la planification
    /// </summary>
    [System.Xml.Serialization.XmlAttribute(AttributeName =
"DateDebutPlanification", DataType = "date")]
    public DateTime DateDebutPlanification { get; set; }
…
```

Figure 19 : Code C# permettant la désérialisation par annotation

La seconde méthode consiste à décrire le comportement de sérialisation comme celui de désérialisation directement dans le code. Pour ce faire, la classe sérialisable doit implémenter l'interface « IXmlSerialisable ». Cela permet d'ajouter deux méthodes qui sont « ReadXml », utilisée pour la désérialisation, et « WriteXml », utilisée pour la sérialisation. La figure 20 montre un exemple d'une classe utilisant la désérialisation par l'implémentation de l'interface « IXmlSerialisable ».

```
/// <summary>
/// Classe permettant d'accéder à toutes les informations concernant le flux
entrant
/// </summary>
/// <typeparam name="TProtocole"></typeparam>
[XmlRoot(ElementName = "ParametrageFluxEntrant"]
public class ParametrageFluxEntrant<TProtocole> : Parametrage,
IParametrageFluxEntrant<TProtocole>, IXmlSerializable
    where TProtocole : ProtocoleAcquisition
{
    /// <summary>
    /// Permet la désérialisation d'un paramétrage de flux entrant
    /// </summary>
    /// <param name="reader"></param>
    public void ReadXml(System.Xml.XmlReader reader)
    {
        ...
    }

    /// <summary>
    /// Permet la sérialisation d'un paramétrage de flux entrant
    /// </summary>
    /// <param name="writer"></param>
    public void WriteXml(System.Xml.XmlWriter writer)
    {
        ...
    }
}
```

Figure 20 : Code C# permettant la désérialisation par implémentation d'une interface spécialisée

La méthode utilisant la désérialisation par le code a notamment été utilisée, dans le projet, pour les cas complexes. Dans l'exemple de configuration présenté en figure 18, certains éléments XML possèdent un attribut nommé « Type ». Cet attribut permet de déterminer, de manière dynamique, *l'assembly*[25] et la classe concernés par le contenu XML. Ceci permet de pouvoir facilement apporter une évolution sans pour autant avoir à repenser l'ensemble du projet. Il suffit alors uniquement de créer une nouvelle *assembly* puis, de la publier sur le serveur hébergeant l'application. Ainsi, pour la désérialisation de l'objet dont la classe est spécifiée par l'attribut XML « Type », il nous suffit, au niveau du code, de récupérer le contenu de cet attribut, puis de lancer sa désérialisation. La figure 21 représente la récupération dynamique du type d'objet à désérialiser.

[25] Une assembly, en langage de programmation .Net, représente une librairie de code compilé. On distingue deux types d'assembly, les process assemblies (les fichiers exécutables .EXE) et les library assemblies (les fichiers .DLL). Dans notre cas les assemblies visées sont des librairies.

```
string typeTableEntranteAttrib = readerTablesEntrantes.GetAttribute("Type");

// Contrôle que l'on a bien récupéré un type
if (typeTableEntranteAttrib == null)
{
    throw new ArgumentNullException("L'attribut 'Type' du noeud ' TableEntrante'
est manquant ou vide.");
}

Type typeTableEntrante = Type.GetType(typeTableEntranteAttrib);

// Contrôle que le type est une sous-classe de TableEntrante
if (!typeTableEntrante.IsSubclassOf(typeof(TableEntrante)))
{
    throw new InvalidCastException(string.Format("Le type '{1}' spécifié dans le
nœud '{0}' n'est pas compatible.", "TableEntrante", typeTableEntrante.Name));
}

readerTablesEntrantes.ReadStartElement("TableEntrante");

// On désérialise la table entrant et on l'ajoute à la collection
var tableEntrante = new XmlSerializer(typeTableEntrante,
Namespace).Deserialize(readerTablesEntrantes);
```

Figure 21 : Code C# permettant la récupération d'un type de classe de façon dynamique

Dans l'exemple ci-dessus, il convient tout de même de vérifier que le type de classe identifié dérive bien d'un type parent attendu, afin de s'assurer que ce dernier soit bel et bien un objet correspondant au métier concerné.

5.3.2 - Utilisation de modules spécialisés indépendants pour les traitements spécifiques

Afin de pouvoir garantir une grande flexibilité de l'application pour ses futures évolutions, il a été décidé d'utiliser des modules spécialisés, à appeler de manière dynamique, pour effectuer les traitements désirés. A titre d'exemple, prenons deux fichiers représentant deux flux entrants : le premier utilise le format CSV et le second est sous format de largeur fixe. Même si les deux flux sont des fichiers, il n'en reste pas moins que, compte tenu de leurs différences de structures, ils doivent utiliser tous deux des méthodes différentes pour lire et récupérer les données. De ce fait, nous avons eu besoin de spécialiser certains traitements en fonction des formats de flux, des protocoles de réception et d'émissions utilisés, ou encore en fonction des méthodes de détermination des dates de valeur des flux entrants.

Pour mettre en place ce mécanisme, nous nous sommes appuyés sur les mécanismes des annotations ainsi que sur celui des extensions que propose le langage .NET. Dans un premier temps, les annotations permettent de rajouter des métadonnées au code-source. Dans un second temps, les extensions permettent de déterminer quel objet est capable d'effectuer le traitement spécifique requis sur les données. La figure 22 représente un exemple d'ajout de métadonnées par annotation en utilisant les « Extension ».

```
/// <summary>
/// paramétrage spécifique au protocole répertoire
/// </summary>
[Extension(typeof(RepertoireCollecteurExtension))]
[Serializable, System.Xml.Serialization.XmlRoot("ProtocoleRepertoire")]
public class ProtocoleRepertoire : ProtocoleFichier
{
    /// <summary>
    /// Chemin du répertoire d'arrivée des fichiers
    /// </summary>
    [System.Xml.Serialization.XmlAttribute(AttributeName = "Chemin", DataType =
"string")]
    public string Chemin { get; set; }
}
```

Figure 22 : Code C# permettant l'ajout de métadonnées par annotation

Dans l'exemple ci-dessus, l'annotation d'extension nous permet de définir que, pour le traitement de l'objet « ProtocoleRepertoire », nous devons utiliser l'objet « RepertoireCollecteurExtension » afin de récupérer le flux de données. L'objet « ProtocoleRepertoire » correspond, dans notre projet, au fait que nous devons récupérer un flux se présentant sous le format d'un fichier dans un répertoire prédéfini. Puis, lors du traitement (la figure 23 est un exemple de code C# mettant en œuvre le mécanisme des annotations), nous utilisons du code générique afin de récupérer l'extension désirée implémentant une interface traduisant le métier que l'on cherche à mettre en œuvre. Dans cet exemple, l'activité est l'acquisition du flux.

```
// Recherche de l'extension de collecte
var extension =
ExtensionAttributeHelper.GetExtension<IExtensionCollecteurProtocole>(parametrage.Acq
uisition.Protocole);

object donneesRecuperees = null;
// CU "Contrôler un flux entrant"
bool res = false;
if (!instance.EstRejeu)
{
    res = extension.RecupererEtArchiverFlux(instance.Id,
instance.ObjConfigFluxEntrant, out donneesRecuperees);
}
else
{
    // Si rejeu
    // On ne peut rejouer qu'un flux à l'état KO
    res = extension.DesarchiverFlux(instance.Id, instance.ObjConfigFluxEntrant, out
donneesRecuperees, emplacementFlux);
}
```

Figure 23 : Code C# montrant la mise en œuvre des extensions

Par ce mécanisme, il est possible de définir plusieurs extensions sur un même objet, à condition que chaque extension implémente une interface différente afin de pouvoir identifier précisément l'extension recherchée pour la suite du traitement. Ce mécanisme est également la base de la répartition du code en fonction des spécificités identifiées.

Nous avons ainsi choisi de décomposer les traitements spécifiques en quatre domaines distincts sachant que, lors de la compilation du projet, chaque extension est traduite en une *assembly*. La figure 24 montre les différents domaines permettant de générer les différentes assemblies représentant les extensions.

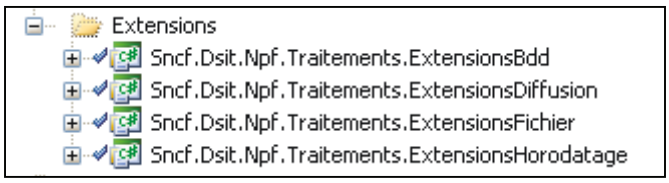

Figure 24 : Liste des *assemblies* regroupant les extensions

Les domaines identifiés pour les extensions sont les suivants :

- **ExtensionBdd :** Il contient la définition des flux entrants provenant d'une base de données ainsi que les traitements spécifiques à réaliser pour la récupération des données en base.

- **ExtensionDiffusion :** Il contient la définition et les traitements spécifiques permettant l'envoi d'un flux à destination d'un client.
- **ExtensionFichier :** Il contient la définition des flux entrants sous format fichier ainsi que les traitements spécifiques à réaliser pour la récupération des données en base.
- **ExtensionHorodatage :** il contient la définition et les traitements spécifiques afin de pouvoir déterminer la date de valeur d'un flux entrant.

Ce mécanisme, couplé à celui permettant la désérialisation des configurations, nous permet de dissocier facilement les traitements communs à tous les flux de données, de ceux, plus spécifiques, qui peuvent être liés, par exemple, au format même du flux de données que l'on reçoit comme de celui que l'on émet. Il permet également de faciliter les tests car, une fois la partie commune des traitements validée, il ne restera plus qu'à tester l'ensemble des comportements spécifiques. De même, lors de futurs ajouts de traitements spécifiques dans l'application, nous n'aurons alors pas à effectuer de tests de non régressions sur l'ensemble de l'application, mais uniquement à les effectuer sur les extensions ajoutées.

5.3.3 - La covariance

Dans notre cas, la covariance est principalement utilisée pour préciser le comportement des classes liées à la configuration des flux. Plus généralement, la covariance permet, avant tout chose, de pouvoir utiliser un sous-type en lieu et place d'un type attendu. Plus précisément, si le type A hérite du type B et que, dans une interface générique, on attend un type B, on pourra alors utiliser également le type A, puisque celui-ci, héritant du type B, est donc nativement un sous-type du type B.

De manière plus concrète, dans notre cas, un flux entrant est composé de plusieurs tables de données entrantes qui, elles-mêmes, sont composées de plusieurs colonnes de données entrantes. Ainsi, la récupération d'un flux entrant nécessite un protocole d'acquisition de flux. Ce principe nous évite donc d'avoir, dans la configuration de nos flux, des types de tables de données avec des protocoles d'acquisition qui ne correspondraient pas. Par exemple, une table entrante de type fichier, ne peut pas avoir un protocole de réception lié à une base de données.

En langage de programmation C#, la covariance au sein des interfaces génériques[26] est déterminé par le mot clé « out ». Ainsi dans l'exemple en figure 25, l'interface de base « ITableEntrante » utilise le générique « out TProtocole », où « TProtocole » doit nécessairement être de type « ProtocoleAcquisition ». Ainsi, toutes les classes dérivant de « ProtocoleAcquisition » pourront être utilisées pour le paramètre générique.

```
/// <summary>
/// Interface de ttableEntrante permetant de lier une table à un protocole
/// </summary>
/// <typeparam name="TProtocole"></typeparam>
public interface ITableEntrante<out TProtocole> : ITableEntrante
    where TProtocole : ProtocoleAcquisition
{
}


/// <summary>
/// Classe abstraite permettant de définir le comportement élémentaire d'une table
entrante
/// </summary>
/// <typeparam name="TProtocole"></typeparam>
public abstract class TableEntrante<TProtocole> : TableEntrante,
ITableEntrante<TProtocole>, IXmlSerializable
    where TProtocole : ProtocoleAcquisition
{
    /// <summary>
    /// Liste des colonnes entrantes de la table de données
    /// </summary>
    public override IList<ColonneEntrante> ColonnesEntrantes { get; set; }
}

/// <summary>
/// Classe abstraite de colonne entrante permettant de la lier à un protocole et à
une table entrante
/// </summary>
/// <typeparam name="TTableEntrante"></typeparam>
/// <typeparam name="TProtocole"></typeparam>
public abstract class ColonneEntrante<TTableEntrante, TProtocole> :
IXmlSerializable
    where TProtocole : ProtocoleAcquisition
    where TTableEntrante : TableEntrante<TProtocole>
{
}
```

Figure 25 : Classes de base montrant l'implémentation du principe de covariance en langage C#

[26] Les interfaces génériques sont issues du principe de la généricité, qui permet à une méthode et/ou une classe d'être indépendante du type de données utilisé. On l'utilise, par exemple, dans la description de listes d'objets de même type. On crée alors une classe utilisant un paramètre générique représentant une liste d'objets pour laquelle on précisera le type d'éléments qu'elle contiendra lors de sa définition. On peut donc, à partir d'un même code de définition d'une liste, définir des listes de nombres entiers ou des listes de chaînes de caractère, et plus précisément des listes d'objet de même type.

La covariance nous permet de partir de ce principe générique puis de spécialiser le comportement. Dans notre application, nous avons, par exemple, des tables de données des flux entrants de type fichier. Ces tables ne peuvent être composées que de colonnes entrantes de type fichier également. Enfin, le protocole de réception du flux est également spécialisé puisque l'on précise qu'il ne peut s'agir que d'un protocole de type fichier (la figure 26 illustre ce principe).

```
/// <summary>
/// Classe abstraite définissant le comportement d'une table de type fichier
/// </summary>
/// <typeparam name="TProtocoleFichier"></typeparam>
public abstract class TableEntranteFichier<TProtocoleFichier> :
TableEntrante<ProtocoleFichier>, IXmlSerializable
    where TProtocoleFichier : ProtocoleFichier
{
}

/// <summary>
/// Colonne entrante générique pour les fichiers
/// </summary>
public abstract class ColonneEntranteFichier<TTableEntranteFichier,
TProtocoleFichier> : ColonneEntrante<TTableEntranteFichier, TProtocoleFichier>,
IXmlSerializable
    where TProtocoleFichier : ProtocoleFichier
    where TTableEntranteFichier : TableEntrante<TProtocoleFichier>
{
}
```

Figure 26 : Premier niveau d'héritage montrant le principe de covariance en langage C#

A partir de cette étape, on spécialise une fois de plus le comportement de la classe pour arriver à la classe définitive qui contient le comportement final attendu. Dans l'exemple ci-dessous (figure 27), cela nous permet de pouvoir définir la structure d'un fichier à largeur de champ fixe, sachant que chaque colonne de données dispose d'une longueur de stockage dans le fichier.

```
/// <summary>
/// Définit une table entrante de type fichier à largeur fixe
/// </summary>
public class TableEntranteFichierLargeurFixe :
TableEntranteFichier<ProtocoleFichier>, IXmlSerializable
{
}

/// <summary>
/// Classe représentant une colonne d'une table de type fichier à largeur fixe
/// </summary>
public class ColonneEntranteFichierLargeurFixe :
ColonneEntranteFichier<TableEntranteFichierLargeurFixe, ProtocoleFichier>,
IXmlSerializable
{
}
```

Figure 27 : Dernier niveau d'héritage montrant le principe de covariance en langage C#

Dans l'exemple présenté ci-dessus, le principe de covariance, couplé à la généricité, nous permet également de définir un comportement non modélisable en UML. Nous sommes partis du principe de base selon lequel, une table de données entrante possède des colonnes entrantes, et que, à la fin du processus de spécialisation, une table entrante de type fichier à largeur fixe ne peut posséder que des colonnes entrantes de type fichier à largeur fixe.

La covariance nous permet donc de manipuler, dans les zones de traitement global de l'application, des objets de type « TableEntrante<ProtocoleAcquisition> » sans pour autant savoir qu'en définitif, le type réel que l'on manipule est « TableEntranteFichierLargeurFixe » ; le type définitif n'étant utilisé que dans les parties de traitements spécifiques qui le concerne.

5.3.4 - Les Transactions

Dans l'application, nous utilisons simultanément des transactions sur deux types d'éléments distincts. Tout d'abord, nous utilisons des transactions pour les messages que l'on reçoit via MSMQ. Le paramétrage que nous avons choisi d'utiliser dans la configuration des messages, précise que le message ne peut être rejoué que deux fois, avant qu'il soit considéré par le système comme invalide. Dans ce cas, il sera mis à l'écart pour ne pas déranger le déroulement des autres messages. Cela permet, en cas d'erreur lors d'un traitement, que le message reste dans la file et puisse être traité par un autre processus. Enfin, et de manière plus classique, nous avons également besoin d'utiliser des transactions quand nous effectuons des tâches sur la base de données. Ceci nous permet d'effectuer l'ensemble des traitements nécessaires pour la bonne exécution d'une tâche, avant de valider les modifications que cela engendre au niveau des données en base.

Lors de la phase de développement, nous nous sommes rapidement rendu compte que la validation d'une transaction sur la base de données validait également la transaction sur le message MSMQ. Ce phénomène est plus que critique, car il remet en question l'utilisation des transactions au niveau MSMQ, mais également la sécurité des traitements. Le fait de pouvoir perdre des messages n'ayant pas été traités est inacceptable. On peut ici distinguer deux sources potentielles pouvant conduire au fait qu'un traitement ne puisse aller à son terme. La première concerne le cas où le service est arrêté volontairement alors qu'il était en cours de traitement d'un message.

Dans ce cas, le message ne doit pas être perdu, il doit être traité par un autre processus. La seconde source apparaît lorsque le processus traitant le message s'arrête suite à une anomalie qui se révèle après la réception du message. Dans ce cas, le fait d'avoir précisé au préalable qu'un message ne pouvait être rejoué que deux fois, va nous permettre d'éliminer naturellement le message indésirable après sa deuxième tentative. Ce second cas étant parfaitement maitrisé par notre configuration, il ne peut donc pas être acceptable que l'on puisse perdre un message en raison du fait que pour son traitement nous nécessitons la base de données ce qui par la même occasion valide la transaction du message.

Pour pouvoir pallier à ce comportement, et après quelques recherches, nous avons dû utiliser le composant Windows MSDTC (Microsoft Distributed Transaction Coordinator) pour la gestion des transactions des messages MSMQ. L'avantage majeur de cette solution est qu'elle n'impacte pas le code source de la solution. Pour sa mise en place, il a simplement suffit d'installer le composant MSDTC de Windows sur toutes les machines par lesquelles transitent les messages MSMQ puis de paramétrer, sur chaque machine, le composant en question. Cela concerne donc les machines hébergeant les services qui composent l'application tout comme la machine qui héberge le composant MSMQ. Ainsi, le composant Windows prend en charge la gestion des transactions entre les diverses machines sur lesquelles est répartie la plateforme de gestion des flux de données. Ce composant permet donc de décharger la responsabilité de la transaction MSMS du Framework .NET, pour qu'elle puisse être gérée par MSDTC directement. Ce transfert de gestion de la transaction à un module extérieur permet que le Framework .NET ne mélange pas les transactions.

En termes de paramétrage, il convient de préciser, dans la zone « Transaction Manager Communication », que l'on autorise les transactions entrantes et sortantes et que les systèmes doivent obligatoirement s'authentifier de manière mutuelle. Ainsi, cela permet aux services de pouvoir créer une transaction sortante avec la machine hébergeant le service MSMQ. L'authentification mutuelle est en outre une sécurité nécessaire pour que seuls les services reconnus puissent utiliser le composant de gestion des transactions (voir figure 28 pour le paramétrage du composant MSDTC).

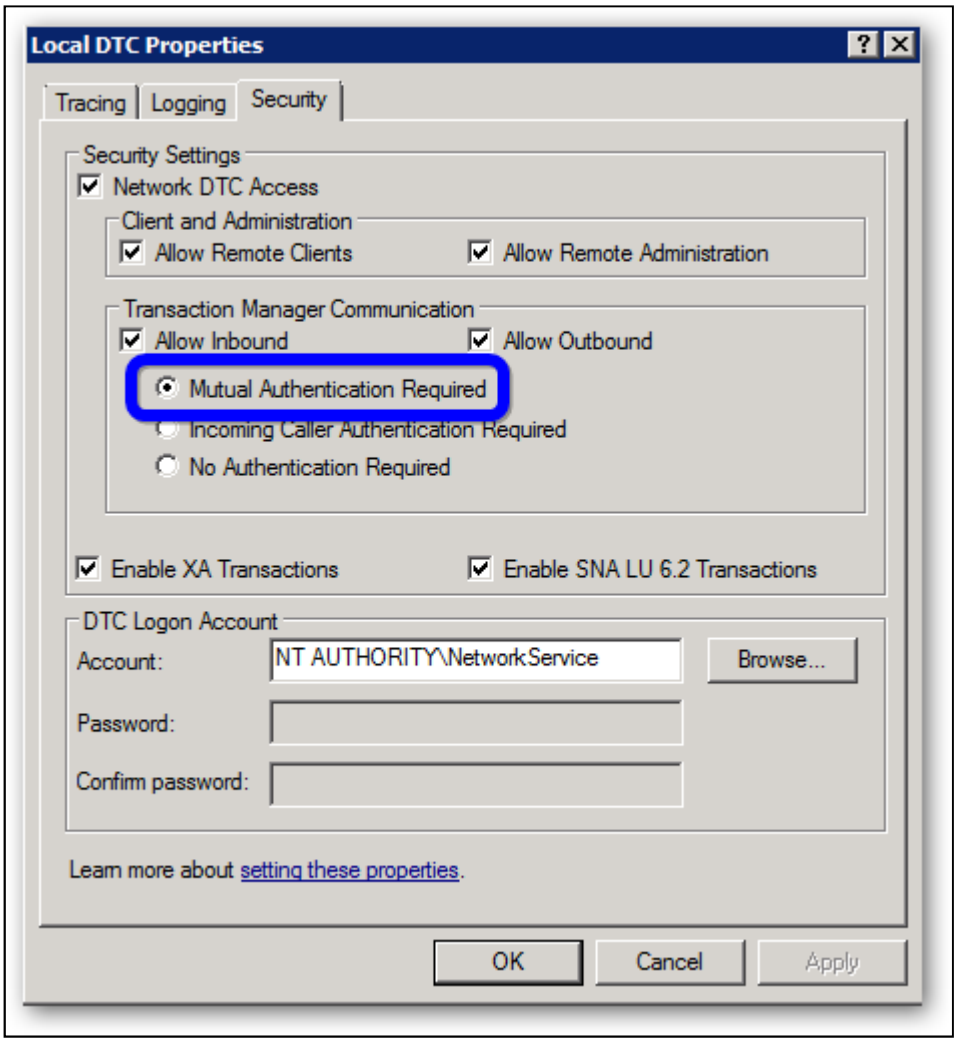

Figure 28 : Paramétrage du composant MSDTC sur une machine utilisant le système d'exploitation Windows 2008 Server

Chapitre 6

Phase de recette et bilan du projet

Après avoir réalisé l'application, nous sommes passés à la phase de recette afin de tester l'application et valider son bon fonctionnement. Cette phase nous a permis de pouvoir véritablement voir comment se comportait l'application dans un environnement proche de celui qui sera déployé par la suite en production. Suite à cette phase de recette, nous avons pu tirer un bilan de ce projet, notamment sur la façon dont nous avons choisi de l'aborder, comme sur l'architecture qui a pu être mise en place.

6.1 – Phase de recette

La phase de recette s'est globalement déroulée en deux temps. Dans un premier temps, nous nous sommes concentrés sur les fonctionnalités de l'application en elles-mêmes. Pour cela, nous avons testé, module par module, chaque cas possible que l'application est capable de prendre en compte. Puis, nous avons comparé les résultats obtenus avec ceux attendus. Dans un second temps, nous avons reproduit, le plus fidèlement possible l'environnement de production. Puis, nous y avons placées des configurations de flux proches de celles que nous aurons en production afin d'observer les résultats à long terme.

Afin de pouvoir réaliser nos tests, nous disposons de trois serveurs distincts. Un premier serveur sous système d'exploitation UNIX héberge la base de données sous Oracle. Cette machine est dédiée à la base de données car elle a vocation à être sollicitée par tous les modules de l'application et nous avions besoin d'une grande réactivité de celle-ci. Les deux autres machines sont des serveurs Windows sur lesquels nous avons pu répartir nos différents modules.

Un cluster est également installé entre ces deux machines Windows (voir figure 29). Le cluster permet de posséder une machine virtuelle reposant sur plusieurs machines physiques. L'avantage de cette solution réside dans le fait que, si une machine venait à être arrêtée, l'ensemble des processus tournants sur le cluster serait immédiatement et automatiquement exécuté par une autre machine qui compose le cluster. Ce cluster permet également, avant tout, de pouvoir disposer d'une seule et unique instance du service MSMQ mais aussi d'un partage unique de réseaux pour la gestion des différents fichiers manipulés par les différents modules et notamment pour la gestion des archives. Enfin, ce cluster permet de n'avoir qu'une seule et unique instance des modules d'orchestration et de planification ; ce qui est une condition pour que notre application ne travaille pas, au même moment, deux fois sur un même flux de données.

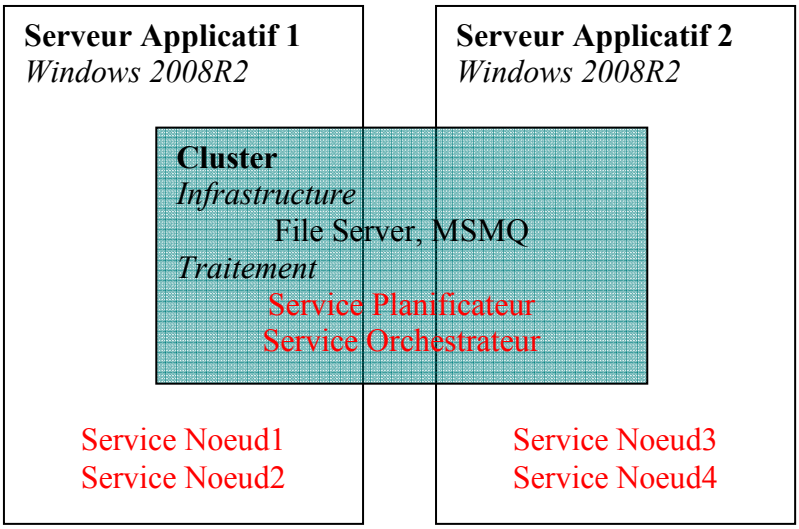

Figure 29 : Schéma de l'architecture technique de la solution

6.1.1 - Test de toutes les fonctionnalités de l'application

Dans la première phase de la recette, nous nous sommes efforcés de tester l'intégralité des fonctionnalités de l'application. Pour cela nous avons créé un nombre important de configurations simples, nous permettant de tester chaque cas. Pour chaque configuration, nous avons également préparé des jeux de tests avec les résultats que l'on attendait.

Pour ces tests, nous avons utilisé une configuration légère en termes de service (voir figure 30), afin de ne pas saturer les machines pendant leurs traitements. Le module de collecte des flux entrant étant le service qui consomme le plus de ressource machine (mémoire et processeur), nous avons choisi de ne faire fonctionner qu'un seul collecteur par serveur.

Plus globalement, nous n'avons disposé qu'un seul type de service par serveur (service autre que ceux devant fonctionner de manière unique) afin de pouvoir tout de même paralléliser nos tests et ainsi gagner du temps.

	Cluster	Serveur 1	Serveur 2
Orchestrateur	1 service		
Planificateur	1 service		
Collecteur		1 service	1 service
Générateur		1 service	1 service
Diffuseur		1 service	1 service
Web Service Interrogation		1 service	1 service

Figure 30 : Architecture logicielle de la première phase de recette

Cette première phase de test nous a permis de vérifier, d'analyser et de contrôler la communication entre les différents modules qui composent l'ensemble de la nouvelle plateforme de flux. Nous avons également pu constater que l'ensemble des fonctionnalités implémentées dans l'application étaient conformes aux exigences émises par la maitrise d'ouvrage. Toutefois, cette phase nous a révélé des faiblesses au niveau de la communication entre les modules et la base de données. En effet, quand la base de données était arrêtée puis redémarrée (cas que nous avons pu constater suite à une sauvegarde qui s'était mal déroulée), l'application ne parvenait plus à détecter de nouveau la base car elle essayait sans cesse de réutiliser des connexions perdues. Pour corriger ce problème, nous avons, dans un premier temps, fait installer par l'équipe en charge du suivi et du contrôle des serveurs, un correctif sur le serveur de base de données Oracle. Après des recherches plus poussées, nous avons également identifié que, grâce à une légère modification dans la chaine de connexion à la base de données, nous pouvions faire valider les connexions avec la base de données avant même que le programme ne l'utilise. Nous avons ainsi pu éviter que le serveur de base de données ne s'arrête de façon intempestive et, dans le même temps, permettre que notre application puisse pallier à un tel problème.

6.1.2 - Reproduction de l'architecture de production

Dans la seconde phase de la recette, nous avons voulu reproduire sur nos serveurs les mêmes configurations de traitement que ce qui serait installé par la suite en production (voir figure 31). Cette configuration nous a permis de pouvoir tester l'application dans sa version définitive.

	Cluster	Serveur 1	Serveur 2
Orchestrateur	1 service		
Planificateur	1 service		
Collecteur		2 services	2 services
Générateur		5 services	5 services
Diffuseur		2 services	2 services
Web Service Interrogation		1 service	1 service

Figure 31 : Architecture logicielle de la seconde phase de recette

En plus de reproduire l'architecture logicielle destinée à la production, nous y avons aussi déposées des configurations de flux destinées elles-aussi à la production. Cette étape nous a permis de valider les imports de données que nous aurons à mettre en place en production. Nous avons également pu fournir, à nos futurs clients, des flux sortants pour que ceux-ci puissent les valider, mais également afin qu'ils puissent les utiliser pour effectuer des tests.

Enfin, et toujours dans un souci de validation du comportement de l'application, nous avons reproduit, sur notre plateforme, certains flux à l'identique de ceux qui s'effectuent encore avec la version actuelle de la plateforme. Ainsi, par comparaison journalière des résultats des deux systèmes, nous avons pu déterminer et identifier les écarts éventuels. Lors de cette phase nous avons déterminé deux éléments différenciant les flux sortants de l'ancien système et du nouveau. La première différence concernait le format de sortie des fichiers CSV. Dans l'ancienne version de la plateforme de flux, les fichiers en CSV possédaient un séparateur « ; » à la fin de chaque ligne ; élément qui ne fait pas partie du standard CSV et qui ne nous avait pas été précisé par la maitrise d'ouvrage. Nous avons donc rajouté un paramètre dans notre configuration permettant d'ajouter, à la demande, un séparateur en fin de ligne.

La seconde différence concernait les sorties de flux en mode différentiel et, plus précisément, les lignes devant être supprimées. En effet, l'ancienne plateforme proposait plus de lignes en suppression que la nouvelle. Après de nombreuses recherches et après avoir retracé le fonctionnement des deux applications, nous sommes arrivés à la conclusion que l'anomalie provenait de l'ancienne plateforme de flux qui émettait des suppressions sur certaines lignes n'ayant jamais été créées au préalable.

6.2 – Bilan du projet

Le projet, dans son lot 1 objet de ce mémoire, a été terminé dans les délais prévus à son lancement. Toutefois, et comme beaucoup de projets informatiques, nous avons été confrontés, au fil de son avancement, à certaines modifications des besoins exprimés par la maitrise d'ouvrage. Certaines modifications devaient être mises en place immédiatement. Etant mineures, elles n'ont pas entrainées de décalage de planning. Les autres ont pu être déplacées vers de prochains lots de développement de l'application.

Le fait d'avoir voulu intégrer un haut degré de qualité dans l'application dès son lancement et d'avoir inclus, dans les chiffrages, le surcoût que cela engendrait en raison notamment de la forte couverture du code développé par des tests unitaires automatisés, a sans aucun doute contribué à avoir aussi peu de retours sur les fonctionnalités livrées. Cette volonté de vouloir mettre l'accent sur la qualité de la *Nouvelle Plateforme de Flux* a été globalement bien acceptée par le client ; surtout quand celui-ci a pu tester l'application livrée et qu'il a pu constater la conformité totale de l'application avec les besoins qu'il avait exprimés.

Conclusion

La *Nouvelle Plateforme de Flux* permet de gérer des flux sous différentes formes, mais, surtout, elle est capable de s'adapter au volume de données à traiter, sans modification de structure. Elle est donc tout à fait en mesure de répondre aux contraintes de forte disponibilité à l'unique condition cependant qu'on lui mette à disposition suffisamment de ressources matérielles pour effectuer ses tâches.

La grande originalité de ce projet se situe essentiellement dans le mécanisme mis en place afin de faciliter les évolutions futures. En effet, le fait d'utiliser des configurations sérialisées en base de données simplifie certes grandement le modèle de classes ainsi que la structure de la base de données, mais nécessite également la mise en place de processus complexes permettant de venir lire ces configurations.

Lors du déroulement de ce projet, j'ai pu bénéficier d'une grande autonomie pour la réalisation des spécifications de l'application. J'ai notamment pu travailler seul à l'élaboration des désérialisations des configurations. Il y avait, sur cette tâche, de fortes contraintes de temps, afin de ne pas engendrer de retard dans le travail des autres développeurs, qui nécessitaient ces éléments pour pouvoir avancer dans les processus métier. Grâce à ce projet, j'ai également, pu découvrir l'importance de la gestion des flux de données au sein d'un grand groupe, mais surtout le fort besoin de supervision pour suivre ces échanges.

Les prochaines versions de l'application, vont mettre l'accent sur la supervision des flux. Cela passera notamment par la mise en place d'un site intranet qui sera capable d'afficher, à tout moment, l'état exact dans lequel se trouve chaque flux à traiter. Cette interface permettra également à tout personnel autorisé de pouvoir modifier, voir saisir, de nouvelles configurations de flux. Cette étape donnera la possibilité de pouvoir déléguer le suivi et l'administration des flux directement à la maitrise d'ouvrage, qui deviendra alors responsable du bon déroulement des échanges de données entre les différentes applications qui composent le système d'information des ressources humaines au sein de la SNCF.

Bibliographie

Talend Data Integration – Talend – Disponible sur : http://fr.talend.com/products/data-integration

IBM InfoSphere DataStage – IBM – Disponible sur : http://www-142.ibm.com/software/products/fr/fr/ibminfodata/

Creating Variant Generic Interfaces (C# and Visual Basic) – Microsoft – Disponible sur : http://msdn.microsoft.com/en-us/library/dd997386%28v=vs.100%29.aspx

Covariance and Contravariance FAQ – Microsoft – Disponible sur : http://blogs.msdn.com/b/csharpfaq/archive/2010/02/16/covariance-and-contravariance-faq.aspx

Oracle Real Application Clusters – Oracle – Disponible sur : http://www.oracle.com/fr/products/database/options/real-application-clusters/overview/index.html

What is MSDTC and why do I need to care about it? – Microsoft – Disponible sur : http://blogs.msdn.com/b/florinlazar/archive/2004/03/04/what-is-msdtc-and-why-do-i-need-to-care-about-it.aspx

Table des annexes

Annexe 1
Liste d'exigences fonctionnelles fournies par la MOA pour concevoir la Nouvelle Plateforme de Flux

Exigences liées au socle de l'application

Id	Nom	Module	Description	Questions
Socle_01	Socle technique	Socle	Module d'orchestration de la plateforme, contient l'enchainement des traitements, la définition des flux et la connaissance des partenaires de la plateforme. Ce module ne peut pas être multi-instances. Ce module contient également la logique de communication entre les différents modules, ainsi que la configuration permettant la prise en compte des différents modules (description de leur emplacement, et de leur rôle).	
Socle_02	Orchestrer les flux entrants et les traitements	Socle	L'orchestrateur est en mesure d'indiquer aux modules concernés les traitements à effectuer lors de la réception d'un flux : - Vérification du flux - Extraction des données - Normalisation des données Chaque traitement est paramétré avec la configuration du flux (constitution du flux, constitution des ensembles de données, récupération de l'horodatage, méthodes de normalisation, destination des données).	

Exigences liées à la collecte des flux

Id	Nom	Module	Description	Questions
Acquisition_01	Alimenter la brique en mode push	Collecteur	Capacité à déclencher un traitement après réception d'un flux ou d'un ordre	
Acquisition_02	push/CFT puis commande	Collecteur	CFT peut exécuter un script après un transfert réussi. Ce script renseigne la plateforme sur le flux reçu pour déterminer les traitements à lancer.	Identification du flux en paramètre de l'appel ?
Acquisition_03	push/transfert + scrutation	Collecteur	Après dépôt d'un flux de type fichier (CSV, Xml ...) un utilitaire de surveillance d'un répertoire renseigne la plateforme sur le flux reçu pour déterminer les traitements à lancer	
Acquisition_04	CFT	Collecteur	Configuration de la réception d'un flux en Cft	
Acquisition_05	FTP	Collecteur	Configuration de la réception d'un flux en Ftp	
Acquisition_06	SMB	Collecteur	Configuration de la réception d'un flux en Smb	
Acquisition_07	Alimenter la brique en mode pull ; Le planificateur doit pouvoir être initiateur	Collecteur	La plateforme peut initialiser la création d'un flux de différentes manières.	
Acquisition_08	Requête SQL	Collecteur	La plateforme interroge une base SQL pour récupérer des données	
Acquisition_09	Appel WS synchrone par le système	Collecteur	La plateforme interroge un Web Service pour récupérer des données	
Acquisition_10	Appel script à distance	Collecteur	La plateforme lance un script. Ce script doit déclencher chez un fournisseur la création d'un flux	
Acquisition_11	Appel WS pour donner ordre -> push	Collecteur	La plateforme appelle un Web Service qui doit déclencher la création d'un flux. Le Web Service ne renvoie pas les données, l'appel est asynchrone.	
Acquisition_12	Alimenter la brique par BaseRéf	Collecteur	La plateforme peut interroger la BaseRef pour récupérer des données	- A l'initiative de la plateforme ? - Basé sur le collecteur SQL ?
Acquisition_13	Alimenter la brique par Central P	Collecteur	La plateforme peut intégrer les fichiers en provenance de Central-P	
Acquisition_14	Alimenter la brique par HRA	Collecteur	La plateforme peut intégrer les fichiers en provenance de HRA	
Acquisition_15	Alimenter la brique par CEGEDIM	Collecteur	La plateforme peut intégrer les fichiers en provenance de CEGEDIM	
Acquisition_16	Alimenter la brique par Base Vieillesse	Collecteur	La plateforme peut intégrer les fichiers en provenance de Base Vieillesse	
Acquisition_17	Accepter les formats d'alimentation de la brique en SQL	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues de BDD	- SGBD supportés ? - mode de requêtage (requête, PS) ?
Acquisition_18	Accepter les formats d'alimentation de la brique en .txt Colonne fixe	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues de fichier à colonne fixe.	
Acquisition_19	Accepter les formats d'alimentation de la brique en .csv Vrai séparateur	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues de fichier avec un caractère de séparation.	

Acquisition_20	Mixte (colonne fixe + séparateur)	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues de fichiers mixte (colonne fixe + caractère de séparation)	- Ce format ne peut-il pas être traité comme un format à colonne fixe ? - Doit-on supprimer les espaces en fin de chaine ? (paramétrable?)
Acquisition_21	Accepter les formats d'alimentation de la brique en .xml	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues de fichier Xml.	
Acquisition_22	Accepter les formats d'alimentation de la brique en MQSeries ? MSMQ ? ActiveMQ ?...	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des données issues d'un MOM	- Quels seraient les outils à supporter ?
Acquisition_23	Accepter les formats d'alimentation de la brique en LDAP Réception LDIF ? Requête LDAP directement ?	Collecteur	La plateforme possède un collecteur capable de récupérer et de traiter des issues d'un LDAP	- Mode de fonctionnement ?
Acquisition_24	Collecter N ensemble de données	Collecteur	Le collecteur peut récupérer des parties d'ensembles de données depuis différentes sources et possède un jeu complexe de validation du flux calqué sur l'existant	- Quelles sont aujourd'hui les règles de validation des flux ldap ?
Acquisition_25	Décodage des messages MQ	Collecteur	Le collecteur peut décoder les messages MQ en provenance d'ldap pour en extraire les données contenues.	- Quel est le format d'encodage des messages ldap ?
Acquisition_26	Fusion et traitement des données	Collecteur	Le collecteur peut aggrêger et contrôler la cohérence des sous-ensembles de données	

Exigences liées au stockage des flux

Id	Nom	Module	Description	Questions
Stockage_01	Stockage physique normalisé	Stockage	Création d'un espace de stockage des données des flux	
Stockage_02	Vue GU	Stockage	Création d'une vue GU	Définition de la vue
Stockage_03	Vue GA	Stockage	Création d'une vue GA	Définition de la vue
Stockage_04	Vue médicale	Stockage	Création d'une vue médicale	Définition de la vue
Stockage_05	Vue organisation	Stockage	Création d'une vue organisation	Définition de la vue
Stockage_06	Possibilité de modifier par configuration la description du stockage physique normalisé et des vues	Stockage	L'IHM de la plateforme permet : - La création du stockage d'un flux - La création de vues	- Lier à la description d'un flux ? - Définir via l'IHM la normalisation des données ?
Stockage_07	Générer une requête	Stockage	Accès au stockage / à une copie du stockage via un outil de requêtes SQL (ex : Toad)	

Exigences liées à la génération des flux sortants

Id	Nom	Module	Description	Questions
Génération_01	Constituer un package de diffusion	Générateur	La plateforme assemble plusieurs ensemble de données pour constituer un flux	
Génération_02	Constituer un fichier avec des données filtrées (filtre colonne)	Générateur	La plateforme constitue un ensemble de données à partir d'une vue fonctionnelle. L'ensemble de données contient tout ou partie des colonnes de la vue.	
Génération_03	Constituer un fichier avec sélection des occurrences (filtre ligne)	Générateur	L'ensemble de données contient tout ou partie des lignes de la vue.	- Limitation des filtres ? Quelle complexité admettre avant que le filtrage ne soit considéré comme métier ?
Génération_04	Constituer un fichier en mode différentiel quotidien/hebdomadaire/mensuel : désynchroniser la fréquence d'acquisition et de diffusion ?	Générateur	La plateforme constitue un flux qui ne contient que les données modifiées depuis l'émission précédente.	- Doit-on proposer des fréquences d'émission de flux différentes des fréquences d'alimentation (ex : alimentation complet hebdomadaire et diffusion différentiel mensuel) ?
Génération_05	Constituer un fichiers en mode complet quotidien/hebdo/mensuel	Générateur	La plateforme constitue un flux qui contient toutes les lignes de données	
Génération_06	Préparer la diffusion manuelle d'un package : - natif : génération sans diffusion (récupérable via IHM)	Générateur	La plateforme peut recevoir des ordres de génération ou de diffusion d'un flux à la demande. Le flux doit être complètement configuré au préalable pour répondre à l'ordre demandé	
Génération_07	- à prévoir : envoi d'une commande de génération	Générateur	L'IHM permet de lancer l'ordre de génération d'un flux. Ce flux est ensuite récupérable dans l'IHM. Il n'est envoyé à aucun consommateur.	
Génération_08	- à prévoir : diffusion configurée pour un flux, mais non planifiée - envoi d'une commande de diffusion	Générateur	l'IHM permet de lancer l'ordre de diffusion d'un flux généré automatiquement. La diffusion est paramétrée (type de diffusion, cible) mais le planificateur ne l'exécute pas.	
Génération_09	Fournir les formats standards de fichier en sortie : .txt Colonne fixe	Générateur	La plateforme possède un générateur capable de constituer des flux avec des fichiers en format colonne fixe	
Génération_10	Fournir les formats standards de fichier en sortie : .csv Vrai séparateur	Générateur	La plateforme possède un générateur capable de constituer des flux avec des fichiers en format avec caractère de séparation	
Génération_11	Mixte (colonne fixe + séparateur)	Générateur	La plateforme possède un génération capable de constituer des flux avec des fichiers en format mixte.	
Génération_12	Fournir les formats standards de fichier en sortie : xml	Générateur	La plateforme possède un générateur capable de constituer des flux avec des fichiers Xml.	
Génération_13	Validation sur un schéma	Générateur	Le génération de flux Xml peut valider le contenu des fichiers avec un schéma Xsd.	
Génération_14	Paramétrer les descripteurs de fichiers standardisés	Générateur	Les vues peuvent être créés dans l'espace de stockage	

Exigences liées à la diffusion des flux préalablement générés

Id	Nom	Module	Description	Questions
Diffusion_01	Diffuser un package de diffusion en mode push	Diffuseur	La plateforme diffuse à échéances fixes un flux. Ces échéances sont définies dans le planificateur	- Doit-on définir par flux des marges d'acceptation de décalage en cas de retard du fournisseur ?
Diffusion_02	CFT (avec ou sans script à réception)	Diffuseur	Diffusion d'un flux via CFT	
Diffusion_03	SMB	Diffuseur	Diffusion d'un flux via SMB	
Diffusion_04	FTP	Diffuseur	Diffusion d'un flux via FTP	
Diffusion_05	Diffuser un package de diffusion en mode pull : WS pour récupérer un package déjà généré mais non diffusé	Diffuseur	La plateforme peut générer et diffuser un flux à la demande d'un consommateur. Cette demande peut prendre plusieurs formes.	
Diffusion_06	Découpage des données suivant un référentiel Agent -> Etablissement	Générateur	Le diffuseur peut découper les ensembles de données suivant les règles définissant les périmètres ldap.	- Quelles sont les règles de découpage des données ldap ?
Diffusion_07	Encodage message MSMQ	Diffuseur		
Diffusion_08	Paramétrer les services de diffusion (abonnements) : identifier le(s) fichier(s) souhaités dans le package de diffusion	Diffuseur	Un fichier Xml permet de définir l'intégralité de la configuration d'un flux de diffusion	- Définition d'un flux ?

Exigences liées à la planification des tâches

Id	Nom	Module	Description	Questions
Planificateur_01	Assurer une fréquence d'alimentation quotidienne, hebdomadaire et mensuelle en mode pull	Planificateur	Le planification propose des programmations de flux quotidienne, hebdomadaire et mensuelles.	Autres ?
Planificateur_02	Assurer toutes les fréquences d'alimentation en mode push	Planificateur	La plateforme peut recevoir des flux en fonction des fréquences imposées par le fournisseur	- Limiter les possibilités de fréquences ? - Vérification de la planification à réception d'un flux ?
Planificateur_03	Paramétrer le calendrier d'acquisition	Planificateur	La planificateur est configurable via une base de données (ou via un fichier de configuration)	

Exigences liées à l'interface d'administration des flux

Id	Nom	Module	Description	Questions
IHM_Admin_01	Paramétrer le calendrier d'acquisition	IHM Administration	la configuration peut être renseignée dans l'IHM	
IHM_Admin_02	Paramétrer les services de diffusion (abonnements) : identifier le(s) fichier(s) souhaités dans le package de diffusion	IHM Administration	l'IHM permet de générer le fichier de configuration	
IHM_Admin_03	Paramétrer les descripteurs de fichiers bruts	IHM Administration	l'IHM permet de générer le fichier de configuration	
IHM_Admin_04	Paramétrer les descripteurs de fichiers standardisés	IHM Administration	L'IHM permet de générer les vues dans l'espace de stockage	
IHM_Admin_05	Gérer les habilitations - IHM de supervision	IHM Administration	IHM Burh	
IHM_Admin_06	Rejouer les flux de diffusion en cas d'échec	IHM Administration	L'IHM permet de relancer la génération et la diffusion d'un flux qui a précédemment échoué	

Exigences liées à l'interface de supervision des flux

Id	Nom	Module	Description	Questions
IHM_SuperV_01	Consulter/télécharger les fichiers bruts	IHM Supervision	On peut consulter les archives des flux réceptionnés via l'IHM.	
IHM_SuperV_02	Consulter/télécharger les packages diffusés	IHM Supervision	On peut consulter les archives des flux diffusés via l'IHM.	
IHM_SuperV_03	Superviser l'ensemble des traitements de la brique	IHM Supervision		
IHM_SuperV_04	Administrateur technique : - Vision de l'état (et étapes) de tous les flux (entrant/sortant) du jour choisi (aujourd'hui par défaut)	IHM Supervision	L'IHM permet à un administrateur de visualiser l'état de tous les flux entrants et sortants pour une date données : - En attente - Réalisé - En erreur ... La page s'ouvre par défaut sur la date du jour.	
IHM_SuperV_05	Client habilité : - vision de l'état macro du flux choisi pour le jour choisi	IHM Supervision	Le système d'habilitation permet à un utilisateur de visualiser l'état des flux de son périmètre	
IHM_SuperV_06	Consulter les tableaux de bord mensuels	IHM Supervision	l'IHM présente des vues statistiques mensuelles des flux : - Nombre de flux réussis - Nombres de flux échoués ...	- Quelles sont les vues attendues dans le tableau de bord ?

Exigences liées au routage des flux

Id	Nom	Module	Description	Questions
Routeur_01	Routage statique	Routeur	?	
Routeur_02	Diffusion vers des destinations multiples	Routeur	Le diffuseur peut transmettre des sous-ensembles de données vers différences destinations	

Exigences liées au Web Service à destination des clients

Id	Nom	Module	Description	Questions
WS_IC_01	WS pour envoyer une commande de génération	WS Interface clients	Un Web Service permet de lancer l'ordre de génération d'un flux.	
WS_IC_02	WS	WS Interface clients	Web Service commandant la diffusion d'un flux	
WS_IC_03	Réceptionner un accusé de réception du client : WS	WS Interface clients	Un Web Service permet au consommateur d'accuser réception du flux.	- Doit-on définir un comportement en cas de non réception ? (Sinon quel est l'intérêt de cette exigence ?)

Exigences liées au Web Service de récupération d'information en temps réel

Id	Nom	Module	Description	Questions
WS_TR_01	Web service d'acquisition de données agent	WS Temps réel	La plateforme propose un Web Service qui permet de récupérer des données liés à un agent (identification par code Immat).	- Définition des données Agent ? - Multiple fournisseurs de données ? - Domaine de données évolutif ? - Protection de ce service ?
WS_TR_02	WS d'acquisition paginé	WS Temps réel	Le Web Service des données agent évolue pour proposer la récupération de données de plages d'agents.	- Définition de taille de plage autorisée ? - Limitation de l'utilisation de ce service ?

Exigences liées à l'architecture logicielle

Id	Nom	Description	Questions
Plateforme_01	Mettre à disposition les résultats de la requête	Outil de requêtes SQL	
Plateforme_02	Consulter les tableaux de bord journaliers bien différencier la supervision du besoin de statistiques Est-ce vraiment nécessaire d'avoir des stat journalières pour un client donné ?	Exigence abandonnée	
Plateforme_03	Avoir 3 environnements distincts : - Production - Pré-production - Recette		
Plateforme_04	Prévoir une haute disponibilité de l'application		- Quels sont les disponibilités attendues ?
Plateforme_05	Prévoir un outil modulable et scalable	L'architecture de la plateforme est composée de différents modules indépendants. A l'exception du planificateur et de l'orchestrateur les modules peuvent être dupliqués sur différents serveurs.	

Annexe 2
Document décrivant le cas d'utilisation « Acquérir un flux dans un répertoire »

Titre :	Acquérir un flux dans un répertoire		
But :	Permet de récupérer un package dans un répertoire Les scénarios : • Scénario commun,		
Résumé :	Récupère un flux dans un répertoire		
Acteurs :			
Pré-conditions	Réception d'une demande d'acquisition d'un flux dans un répertoire.		
Post-conditions			
Exigences	1. Acquisition_06		
Règles de gestion	1.		
Relations avec d'autres cas d'utilisation	Appelant	Appelé	
	Déclencher une acquisition		
Date de création :	06/02/2012	Date de mise à jour :	
Version :	1.0	Créateur :	LAPIERRE Patrice

Historique des modifications :

DATE	AUTEUR	VERSION	MOTIF

Scénario commun aux enchaînements décrits par la suite :

Action de l'acteur	Réponse du système
A- Le cas d'utilisation commence lorsque le système reçoit une demande d'acquisition de flux devant se trouver dans un répertoire.	Le système récupère des informations sur le nom du package à identifier, ainsi que sur le répertoire dans lequel il faut le chercher.
B- Le système possède les informations nécessaires au traitement.	Le système lance une recherche du flux dans le répertoire.
C- Le système identifie un package correspondant.	Le système déplace le package dans un répertoire de travail. Il informe le cas d'utilisation appelant que le flux a été trouvé.
D – Décompression	Si le fichier composant le flux est compressé, le système décompresse le contenu du fichier dans le répertoire de travail. Un mot de passe peut être défini pour l'archive.

Enchaînements alternatifs au scénario commun

Point du scénario	Action de l'acteur	Réponse du système
C	Le système ne détecte pas le flux dans le répertoire	Le système informe le cas d'utilisation appelant que le flux n'a pas été récupéré.

Exceptions au scénario commun

Point du scénario	Type	Description	Message
B	Erreur de configuration	Des informations de récupération du flux sont erronées ou incomplètes	Le flux [nom_flux] n'a pu être récupérer en raison d'un problème dans sa configuration.

Contraintes non-fonctionnelles

Règle de gestion de la création du répertoire de travail

Le système crée dans un répertoire commun à la plateforme un répertoire de travail nommé par l'identifiant du flux. Si le répertoire du flux existe déjà il est préalablement supprimé avec ton son contenu.

Configuration nécessaire à l'exécution du cas d'utilisation :

- Entité `FluxEntrant`
  - Chemin d'accès au répertoire où trouver le package
  - Format ou nom du flux à identifier
  - `Protocole` : protocole (voir entité `ProtocoleFichier`) utilisé par le flux
- Entité `ProtocoleFichier`
  - `EstArchive` : indique si le fichier est une archive
  - `Format` : indique le format de l'archive (zip, rar, tar)
  - `MotDePasseArchive` : mot de passe de l'archive

Liste des figures

Réalisation d'une plateforme inter-applicative de gestion de flux de données.

Mémoire d'Ingénieur C.N.A.M., Lyon 2012

RESUME

Suite à des modifications au sein du système d'information des ressources humaines de la SNCF, l'outil gérant les flux de données, initialement en place s'est vu dans l'incapacité de pouvoir traiter le volume attendu. Après une étude de plusieurs solutions, la décision fut prise de procéder au développement d'une Nouvelle Plateforme de Flux.

Une étude de besoin fit apparaitre de nouveaux éléments à prendre en compte, dans le développement de la nouvelle plateforme, en plus de ce qu'est capable de faire l'ancienne application. La réalisation de cette application a nécessité un lotissement des fonctionnalités afin de respecter des contraintes liées au planning. Enfin, une fois l'architecture établie, il a fallu relever plusieurs difficultés d'ordre technique.

Mots clés : flux de données, configuration, désérialisation

SUMMARY

Due to some modifications of the information system of the human ressources of the SNCF, the original application to manage datas flow was not able to process the expected quantity. After studying several alternatives, the decision was made to proceed of the development of a New data Flow Plateform.

A study of needed did to appear new elements to take account, in the development of the new platform, in addition of what the old application can do. The realization of this application has required a subdivision of functionalities, to respect schedule constraints. Finally, once the established architecture, we have had to overcome several technical difficulties.

Key words : datas flow, configuration, deserialization.