L’interopérabilité entre applications

Sylvie Malalatiana

Architecte Logiciel

2016 - 2018

Capgemini

ENEDIS
L'ELECTRICITE EN RESEAU

nextadvance
Formez-vous à des métiers d'avenir

Next group

Remerciements

Je remercie le groupe Capgemini de m'avoir accueilli au sein de l'entreprise afin de mener à bien mon alternance.

Je remercie également Enedis de m'avoir accueilli dans leur projet.

Un grand merci à toutes les équipes du projet Infrastructure d'Echange et surtout aux membres de l'équipe Suicide Squad : Nicolas, Salwa, Roody, Kevin, Ameline et Laetitia de m'avoir permis d'évoluer dans leur projet et de m'avoir aidé et surtout supporté durant ces deux longues et belles années.

Merci à notre Responsable de site à NextAdvance, Madame Anabelle Martin, de nous avoir accompagné et conseillé pendant notre formation et d'avoir répondu à chacune de nos questions.

Je remercie également Madame Amandine Meurer de nous avoir partagé sa bonne humeur et sa gentillesse tout au long de notre formation.

Merci à ma formatrice Laure Bourgois de m'avoir accompagnée lors de la rédaction de ce document.

Je souhaite également remercier mon formateur Mbengue Mohamadou de nous avoir accompagnés tout au long de la formation et de nous avoir aidés dans nos projets.

Mon plus grand remerciement s'adresse à mon Product Owner Nicolas Guillerme qui a répondu à toutes mes questions (un peu abusées des fois) et sans qui, je n'aurais pas pu avancer dans la rédaction de ce document, mais aussi de nous avoir fait profiter de très bons moments et de fous rires sur le projet.

Je remercie mon groupe de projet : Arnaud et Manassé pour le bon déroulement du projet. Mais je remercie particulièrement Arnaud d'avoir supporté mes mauvaises humeurs.

Merci à tous ceux qui ont contribué de près ou de loin à la préparation de mon diplôme et à la rédaction de ce mémoire.

Objectif du document

Face aux problèmes d’interopérabilité auxquels font face les entreprises actuellement, il existe plusieurs solutions qui résolvent les problèmes d’intégrations et de communications entre applications.

Ce document a pour objectif d’expliquer ces problèmes d’interopérabilité et les solutions proposées par le projet Infrastructure d’Echange dans lequel je travaille actuellement, pour résoudre ces problèmes.

Ce document est composé de trois grandes parties dans lesquelles je parlerai de l’entreprise d’accueil ainsi que l’entreprise cliente, ensuite je présenterai le projet au niveau fonctionnel et au niveau technique, et enfin je parlerai de la solution technique utilisée par le projet et une solution alternative qui aurait pu remplacer la solution en place. En dernière partie, une étude comparative que j’ai effectuée sur les deux solutions sera proposée afin de donner une idée sur les avantages et les inconvénients de chaque outil.

SOMMAIRE

SOMMAIRE DES IMAGES

INTRODUCTION

Les systèmes d'informations en entreprise deviennent de plus en plus complexes et moins en moins homogènes. Dans le but de les faire communiquer et de les faire coopérer, les intégrer devient indispensable. Il est nécessaire de définir une certaine interopérabilité entre les applications concernées.

Le but de cette interopérabilité est de pouvoir garantir l'échange d'information avec le moins de dépendance possible au niveau des logiciels utilisés. Elle permet également d'éviter les restrictions d'accès ou de mise en œuvre, comme l'impossibilité de lire certains formats de fichier par exemple. Une des solutions que l'on pourrait proposer face à ce genre de situation serait de faire en sorte que chaque application soit conçue de façon à pouvoir communiquer avec le reste lors de la conception du système d'information.

Le grand problème s'impose alors lorsque les applications au sein du système d'information sont amenées à évoluer, ou que d'autres applications fraîchement conçues font leur apparition. Dans ce cas l'impact sera énorme car il faudra retoucher chaque application existante pour pouvoir intégrer chaque nouvelle. Ces modifications impliquent un investissement en temps et en argent qui n'est pas des moindres. Sans compter le fait qu'il est possible que les applications soient très compliquées à intégrer.

Il existe actuellement de multitudes de possibilités pour pouvoir mener à bien l'interopérabilité entre applications et l'intégration d'entreprise avec le moins de configuration possible et d'interventions au cœur même de l'application en question.

Le projet dans lequel j'interviens aujourd'hui, répond à cette problématique. En effet, il offre des solutions afin que plusieurs applications puissent communiquer entre elles de manière automatisée et sécurisée. Pour ce faire, le projet propose l'utilisation d'un pattern qui est l'« EIP » (Enterprise Integration Patterns ou Modèle d'Intégration d'Entreprise). Pour implémenter ce pattern, nous utilisons actuellement le framework Apache Camel.

Durant ce document, nous allons aborder plus en profondeur les enjeux du projet, son fonctionnement et les fonctionnalités qu'il propose. Nous allons aussi définir plus en détails ce qu'est un EIP. Comme Camel Apache est l'implémentation proposée et utilisée dans ce contexte, nous allons donner une description approfondie comprenant son architecture et son fonctionnement interne. Dans ce document, sera aussi proposée une solution alternative que l'on aurait pu aisément utiliser à la place de Camel Apache.

PRESENTATION DE CHAQUE PARTIE

Partie I : Présentation de l'Entreprise

Actuellement en poste chez Capgemini qui est une ESN[1] (Entreprise de Services Numériques) proposant des prestations aux entreprises clientes, je suis amenée à travailler en régie chez Enedis en tant que prestataire. Dans cette partie, nous allons donc présenter Capgemini, ensuite nous allons voir en plus détail ce le périmètre et la structure de la société Enedis, dans quel secteur elle intervient et comment elle s'est construit.

Partie II : Présentation du projet

Contexte et Problématique

Actuellement, il est important de pouvoir intégrer et de faire communiquer plusieurs applications entre elles. Et ce, de manière totalement automatique, c'est-à-dire ne pas avoir besoin de modifier a posteriori ni l'architecture ni les technologies des applications concernées. Le projet dans lequel je suis actuellement impliquée, a pour mission de développer des outils qui permettent cet échange, d'où son nom Infrastructure d'Echange (IE). Ainsi, en passant par notre plateforme, ces applications peuvent s'envoyer des données ou faire appel à des Web Services. Dans cette partie, nous allons voir en détails les problèmes auxquels répondent le projet et sa raison d'être.

Présentation fonctionnelle

Pour mener à bien son rôle, le projet offre des fonctionnalités aux clients qui souhaitent faire des demandes de flux[2] sur le projet. Ces fonctionnalités sont classées par niveau de complexité allant du niveau 1 qui est le plus simple, jusqu'au plus complexe qui est le niveau 4. Selon les fonctionnalités demandées, le développement de l'outil peut être fait en automatique ou en manuel. Cette partie explique en détails ce que chaque fonctionnalité offre et comment elle fonctionne. C'est une présentation du projet au niveau purement fonctionnel.

Présentation Technique

L'outil réalisé par les équipes de développement pour permettre les échanges d'application est appelé chorégraphie sur le projet. Chaque chorégraphie développée est propre à chaque client et à chaque demande qu'il fait sur l'outil de demande que l'on

[1] ESN : Entreprise offrant des services informatiques et de la transformation digitale
[2] Flux : Transfert d'informations entre deux ou plusieurs applications

appelle actuellement « ASTRE[3] Demande ». Une chorégraphie permet de faire une ou plusieurs échanges de manière synchrone ou asynchrone entre plusieurs applications. Pour permettre le développement et le déploiement de ces outils, une architecture technique a été mise en place et plusieurs technologies collaborent entre elles. Cette partie consiste à décrire le fonctionnement du projet au niveau technique, à voir plus précisément ce qu'est l'Infrastructure d'Echange et comment elle fonctionne et aussi les différents types d'échanges que l'on peut effectuer. Les différentes technologies utilisées au sein du projet sont aussi présentées dans cette partie.

Partie III : Solution actuellement proposée : les « Enterprise Integration Patterns »

Pour l'intégration et la communication entre applications, il existe actuellement plusieurs technologies et plusieurs manières de procéder. La solution proposée par le projet Infrastructure d' Echange est un pattern d'intégration que l'on appelle Enterprise Integration Patterns. C'est un pattern qui utilise le principe de messagerie pour transférer des paquets de données immédiatement, de manière fiable. Dans cette partie, le mode de fonctionnement de ce pattern est expliqué en détails. Nous allons présenter également l'architecture globale d'un EIP et son utilisation.

Solution d'implémentation proposée actuellement : Camel Apache

Comme expliqué dans la section précédente, un EIP est un concept d'architecture basée sur la messagerie. Il existe sur le marché plusieurs technologies qui implémentent ce pattern. Pour répondre aux besoins des clients, le projet s'appuie sur Camel Apache pour le développement de ses chorégraphies. Camel Apache est un framework qui implémente le pattern EIP et utilise le système de messagerie pour l'intégration et l'envoi des données entre plusieurs applications. Cette partie parle de Camel Apache, de sa structure et de son fonctionnement. Nous allons voir son architecture interne ainsi que le trajet et les transformations éventuelles subies par un message depuis son départ d'un point A jusqu'à son arrivée à un point B. Ses avantages et ses limites seront présentés dans cette partie.

Solution d'implémentation alternative : Spring Integration

Comme je l'ai déjà introduit précédemment, les technologies d'intégrations sont nombreuses sur le marché informatique. Ne sortant pas de ce cadre d'EIP, une autre technologie que l'on pourrait utilisée pour remplacer Camel Apache serait Spring Integration. C'est une extension de Spring framework qui offre les mêmes fonctionnalités que Camel Apache. Son architecture interne diffère un peu de celle de Camel mais se rapproche beaucoup plus de celle d'un EIP. Ce sont ces détails que l'on abordera dans cette section. Nous allons présenter Spring Integration, expliquer son architecture et son mode de fonctionnement. Nous allons également parler des avantages et de ses limites.

[3] ASTRE : Application de supervision de transport et de reprise des échanges

Etude comparative entre Camel Apache et Spring Integration

Camel Apache et Spring Integration sont deux logiciels similaires et permettent d’effectuer le même travail. Ce sont deux implémentations différentes d’un EIP et qui possèdent chacun leurs inconvénients et leurs avantages. Ce sont deux frameworks basés sur Java et sont assez similaires dans leur mode de fonctionnement. Malgré les points communs entre ces deux frameworks, ils présentent de nombreuses différences et chacun ses inconvénients et ses avantages sur certains points. L’objectif de cette partie et de voir quelques points de comparaisons entre ces deux logiciels.

I. PRESENTATION DE L'ENTREPRISE

A. Le Groupe Capgemini

Capgemini est la première entreprise de services du numérique (ESN) en France ainsi que le numéro six mondial du secteur en 2016. Elle a été créée par Serge Kampf le 1er octobre 1967 à Grenoble sous le nom de Sogeti (Société pour la gestion de l'entreprise et traitement de l'information). Fort de 50 ans d'expérience et d'une grande expertise des différents secteurs d'activité, il accompagne les entreprises et organisations dans la réalisation de leurs ambitions, de la définition de leur stratégie à la mise en œuvre de leurs opérations. Pour Capgemini, ce sont les hommes et les femmes qui donnent toute sa valeur à la technologie. Résolument multiculturel, le Groupe compte 200 000 collaborateurs présents dans plus de 40 pays. Il a réalisé un chiffre d'affaires de 12,8 milliards d'euros en 2017. Capgemini est une ESN intervenant dans multiples secteurs d'activités et offrant ses services à de nombreuses entreprises et dans différents secteurs d'activités. L'entreprise cliente d'accueil que l'on présentera dans le cadre de ce mémoire est située dans le domaine de l'énergie.

B. L'Entreprise Cliente : ENEDIS

Suite à l'ouverture du marché de l'électricité à la concurrence, les activités de production, de transport et de commercialisation d'EDF ont été séparées de ses activités de distribution. Cette dernière activité a alors été confiée à Enedis.

a. Historique[4]

La fin du XIXe siècle et le début du XXe sont marqués par l'essor des usages de l'électricité, avec notamment l'électrification des tramways, des métros et des chemins de fer. Les compagnies d'électricité s'implantent alors dans les grandes villes de France. Elles y construisent des centrales électriques et de petits réseaux locaux. À cette époque, chacune utilise des fréquences et des niveaux de tension variables, sans se préoccuper de l'interconnexion des réseaux. Rapidement, les compagnies d'électricité se rendent compte qu'elles ont besoin de se porter assistance pour continuer à alimenter leurs clients en cas de panne. Elles décident d'adopter les mêmes standards techniques. Le maillage des réseaux peut alors commencer. Lorsque l'électrification des campagnes est envisagée, le développement du réseau de distribution est confronté à un problème de coût. Pour le résoudre, le système des concessions de service public est créé. Grâce à des contrats de concession, les communes confient à des sociétés tierces la construction des infrastructures électriques. En contrepartie, ces sociétés en assurent l'exploitation et perçoivent des redevances sur l'usage du réseau. Grâce à ce système, le réseau s'étend rapidement localement sous l'impulsion de compagnies électriques privées. On passe ainsi de 7 000 communes électrifiées en 1919 à plus de 36 500 en 1938. Ces compagnies sont amenées à

[4] Source : Wikipedia

collaborer pour interconnecter leurs réseaux de transport d'électricité, créant à cette occasion des sociétés communes spécialisées. Le début des années 1930 marque le début des interconnexions de grande capacité (220 000 volts) entre les régions où sont produites l'électricité notamment les Alpes et la capitale. En quelques années, la France se dote ainsi d'un grand réseau d'interconnexion à 220 000 volts. L'Etat intervient alors au nom de l'intérêt général. En 1938, il lance un plan d'interconnexion national pour porter l'électricité dans toutes les régions du territoire. Ce programme prévoit des investissements de 3 milliards de francs sur 5 ans : 1,5 milliard pour augmenter de moitié la production hydraulique et 1,5 milliard pour construire 4 000 kilomètres de lignes de transport d'électricité. À la Libération, le réseau de transport est le plus dense du monde, avec 22,5 km de lignes de plus de 100 000 volts pour 1 000 km2 (contre 5 km pour les États Unis, 15 pour la Grande-Bretagne et 18 pour l'Allemagne).

Le 8 avril 1946, la loi qui nationalise les entreprises d'électricité est votée. Un nouvel établissement public, Électricité de France (EDF), est créé. Il intègre les sociétés de production, de distribution et de transport d'électricité. EDF devient un instrument essentiel de la reconstruction de la France. Quant aux entreprises d'électricité fonctionnant en régie, elles ne sont pas nationalisées, car elles sont des émanations d'organismes publics. Elles forment aujourd'hui les entreprises locales de distribution (ELD). La 1ère mission d'EDF consiste à gérer la pénurie. La distribution d'électricité est fonction des priorités nationales et régionales. Jusqu'en 1950, il faudra, faute d'énergie suffisante, organiser des coupures d'électricité. Autre tâche : l'harmonisation des standards hérités des sociétés nationalisées. La tension à 225 000 volts se substitue progressivement à 150 000 volts. La fréquence de 50 Hz se généralise à l'ensemble du territoire (elle était de 25 Hz sur une grande partie du littoral méditerranéen). De 1950 aux années 2000, le réseau va se développer au rythme de la consommation croissante d'électricité. La tension à 400 000 volts, mise au point dès 1946, est adoptée comme norme européenne. Elle se déploie à partir des années 1960 et pendant les 2 décennies suivantes, accompagnant la montée en puissance de la production d'énergie d'origine nucléaire. Les interconnexions avec les pays voisins se développent pour soutenir un solde de plus en plus exportateur. La loi de février 2000 introduit l'ouverture progressive du marché de la fourniture de l'électricité conformément aux engagements de la directive européenne de 1996 qu'elle transpose**.** Cette loi définit les missions de service public en matière d'électricité et leur financement. Elle prévoit la création d'un gestionnaire du réseau de transport d'électricité indépendant de même que la séparation comptable des activités de réseau. Elle crée également la Commission de Régulation de l'Énergie (CRE), qui concourt au bon fonctionnement des marchés de l'électricité et du gaz naturel. Elle prévoit pour tous les utilisateurs un accès non discriminatoire au réseau électrique. En juillet 2000**,** EDF crée une direction autonome dédiée à la gestion du réseau de transport de l'électricité. Ainsi naît RTE (Réseau de Transport Électrique). En juin 2003, une 2ème directive européenne impose la séparation juridique du gestionnaire du réseau de transport et du gestionnaire du réseau de distribution.

L'année suivante, conformément à la loi du 9 août 2004 qui transpose cette 2ème directive, EDF sépare fonctionnellement ses activités, la production et la fourniture entrent dans le secteur concurrentiel, le transport et la distribution restent des activités régulées.

Ainsi, s'agissant de la distribution, EDF crée un service gestionnaire de réseau de distribution indépendant fonctionnellement, EDF Réseau Distribution (ERD), un opérateur commun avec Gaz de France, EDF Gaz de France Distribution (EGD).

La loi prévoit également la séparation juridique du gestionnaire de réseau de transport de l'électricité (RTE). La loi relative au secteur de l'énergie de décembre 2006 parfait le dispositif en complétant les missions des gestionnaires de distribution et en prévoyant la séparation juridique des gestionnaires de réseau de distribution (ayant plus de 100 000 clients) en opérant le transfert des biens, droits et obligations liés à l'activité de distribution auxdits gestionnaires. Elle adapte les mesures garantissant l'indépendance des gestionnaires et impose la création d'un service commun aux filiales gaz et électricité.

Etant une société anonyme à conseil de surveillance et directoire, filiale à 100% d'EDF chargée de la gestion et de l'aménagement de 95% du réseau de transport d'électricité en France, Enedis, anciennement appelé ERDF a été créée le 1er janvier 2008. Elle ne doit pas être confondue avec RTE qui est le gestionnaire du réseau de transport d'électricité en haute tension (supérieure à 50kV). Aujourd'hui, l'entreprise survole de plus d'un siècle de distribution de l'électricité en France.

b. Rôles

Au niveau global, Enedis gère les câbles, les lignes, les transformateurs et les postes électriques répartis sur le territoire français. À l'échelle cliente, cela comprend aussi l'entretien, la réparation, l'installation de votre compteur électrique ou encore l'ouverture et la fermeture de l'électricité, sans oublier la modification de puissance de votre courant et les différentes pannes, et ce, qui que soit votre fournisseur d'électricité. Enedis reste un acteur essentiel de la chaîne de valeur de l'électricité. Il assure l'approvisionnement en électricité de près de 35 millions de clients.

i. Fonctionnement technique

L'électricité produite par les centrales est d'abord acheminée sur de longues distances dans des lignes à très haute et à haute tension (dites « THT » et « HT », entre 400 000 et 63 000 volts) gérées par RTE[5]. L'électricité est ensuite « transformée » (modification des niveaux de tension et d'intensité) dans des postes de transformation placés à l'interconnexion des réseaux de transport et de distribution. Elle est enfin distribuée dans des lignes à moyenne et à basse tension (dites « MT » et « BT », entre 20 000 et 230 volts) gérées par le réseau de distribution Enedis. Une fois sur le réseau de distribution, l'électricité moyenne tension (MT) alimente directement les clients industriels. Pour les autres clients (particuliers, commerçants, artisans, etc.), elle est convertie en basse tension (BT) par des postes de transformation avant de leur être livrée. Enedis assure divers types de prestations pour ses clients (particuliers, entreprises ou localités). Citons notamment le raccordement physique d'une installation au réseau de distribution, le dépannage pour assurer la continuité de la fourniture ou bien encore le relevé des consommations ainsi que le contrôle, l'entretien et le renouvellement du matériel de comptage.

Le processus d'acheminement de l'électricité depuis sa production jusqu'à l'arrivée au client final est illustré par l'image ci-après :

[5] Sigle du réseau de transport d'électricité. C'est une entreprise qui gère le réseau public de transport d'électricité haute tension en France (Source : Wikipedia)

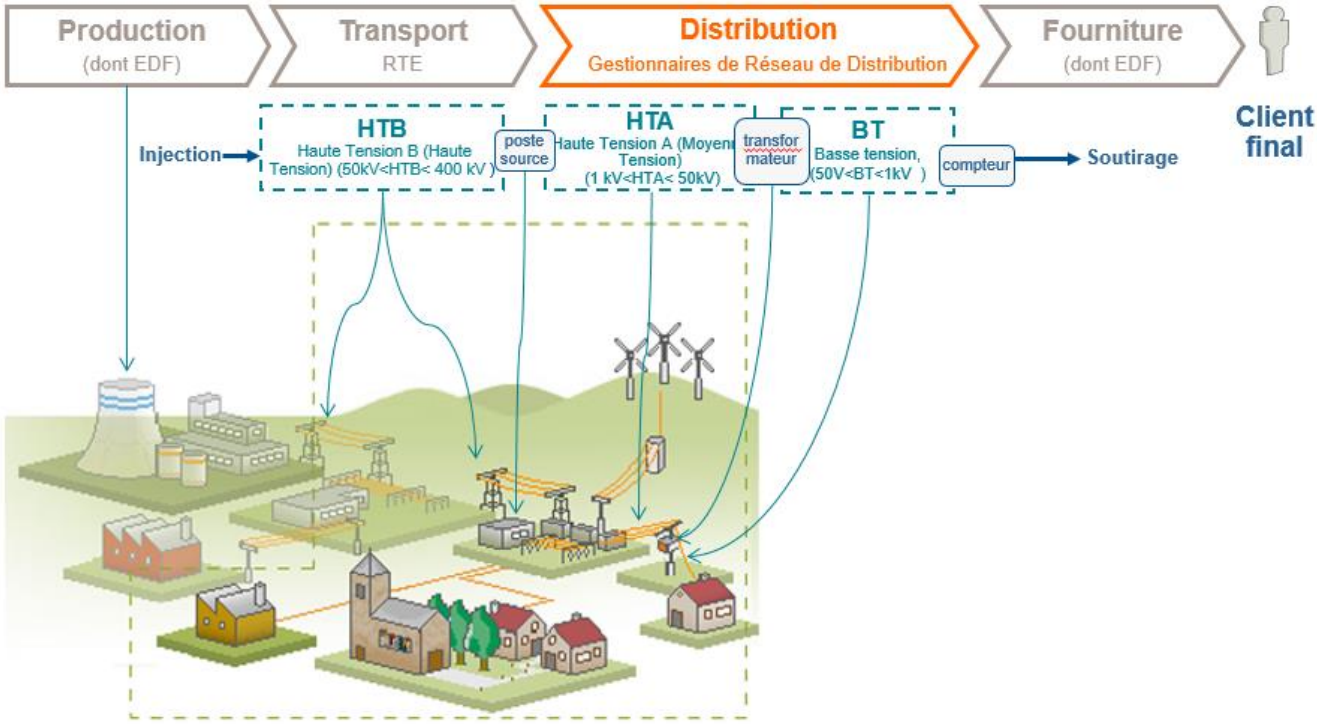


Figure 1 : Architecture technique d'ENEDIS (Document fourni par l'entreprise)

Zone de couverture

Enedis est donc gestionnaire du réseau de distribution en France métropolitaine et continentale. Il a l'obligation de garantir un accès au réseau électrique aux clients des fournisseurs d'électricité quels qu'ils soient. Le lieu d'intervention et d'exploitation d'Enedis se trouve dans toute la France et il :

- Exploite, entretient et développe 1 332 942 km de lignes à moyenne tension dites « MT» (aussi appelées lignes « HTA » pour Haute Tension A) et de lignes à basse tension dites « BT » à fin 2014.
- Exploite près de 2 250 postes sources permettant de relier le réseau de transport de RTE à son réseau de distribution.
- Exploite près de 770 000 postes de transformation permettant d'abaisser progressivement la tension de l'électricité distribuée.

Le schéma suivant montre les zones de couvertures de l'entreprise :

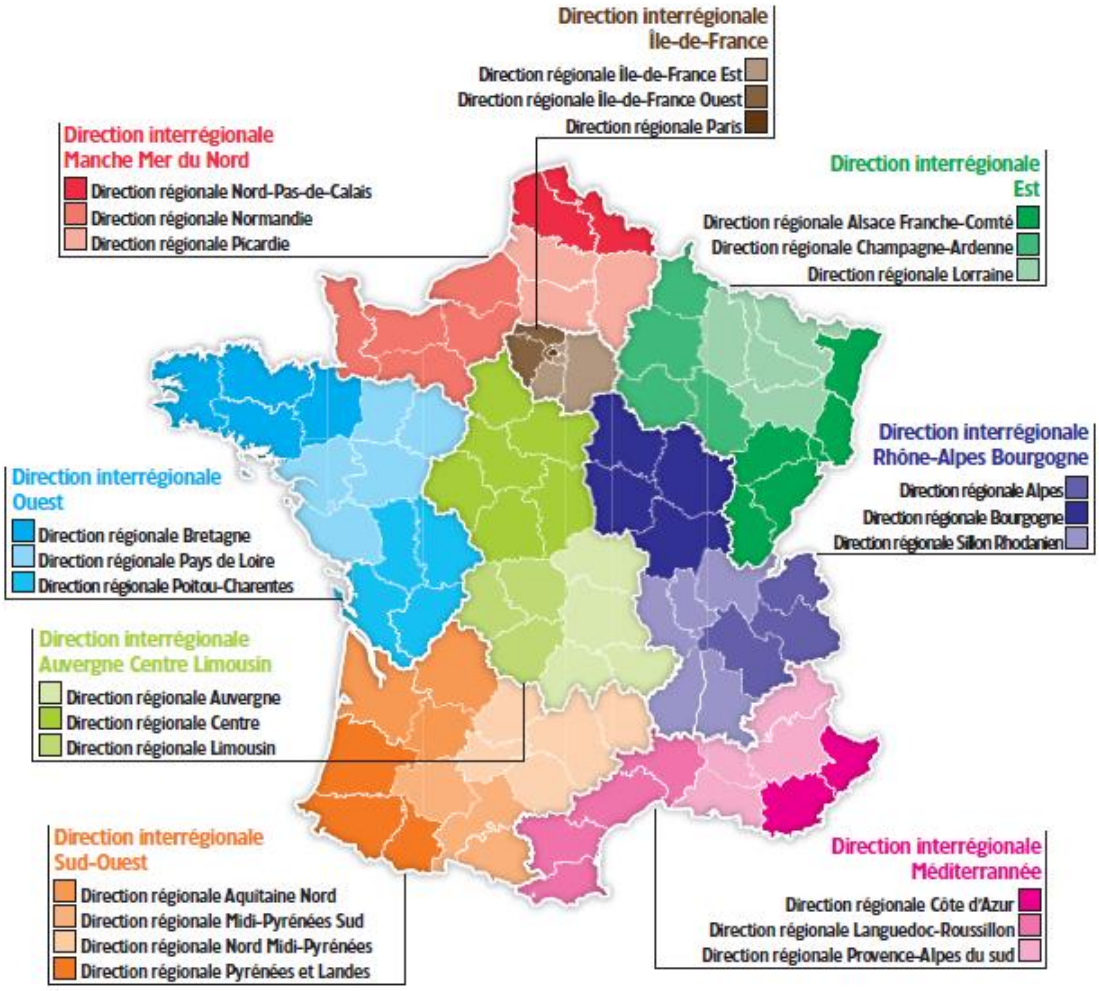


Figure 2 : Localisations des sites ENEDIS en France(Document fourni par l'entreprise)

II. PRESENTATION DU PROJET

A. Contexte et Problématique

De manière générale, il est rare que deux projets, au sein d'une même entreprise, utilisent les mêmes applications. Il est assez courant que ces projets puissent ou doivent, selon les cas, communiquer entre eux avec des envois de données ou de documents.

La communication en mode manuel (par l'utilisation d'email par exemple ou autre outil de partage) peut s'avérer coûteuse en temps et en organisation car il nécessite une intervention humaine. Une des solutions qui pourraient être proposée est de passer par les différentes applications pour effectuer les échanges.

Le problème est que ces applications ne sont pas toujours compatibles entre elles. Chacune d'elle possède sa propre architecture et son propre langage.
Par exemple, le format de données utilisé par deux Système d'information peut différer selon les besoins de son projet et des données qu'il utilise. L'une peut requérir du XML avec

enveloppe SOAP (Service Oriented Application Protocol) tandis que l'autre du JSON, et l'interopérabilité poserait un problème.

Pour résoudre ce problème, une plate-forme permettant les différents échanges pourrait être mise en place. Elle s'occupera de la transformation des données dans le format de l'application réceptrice et de l'envoi des données vers cette dernière. La plate-forme peut également s'occuper de la sécurisation des données si besoin. Par exemple le cryptage d'un fichier.

Dans le cadre d'Enedis, le projet s'occupant de cette tâche se nomme actuellement Infrastructure d'échange.
Le projet Infrastructure d'échange au sein d'Enedis permet aux différentes applications des différents projets, et même aux applications externes en interactions avec l'entreprise, de pouvoir échanger plus facilement des données entre elles. Chaque donnée envoyée par l'application A est envoyée sur la plateforme d'échange où elle sera traitée et c'est la plateforme d'échange qui va l'envoyer vers l'application B. Les échanges peuvent se faire dans les deux sens.

L'image suivante illustre cet échange entre deux applications :

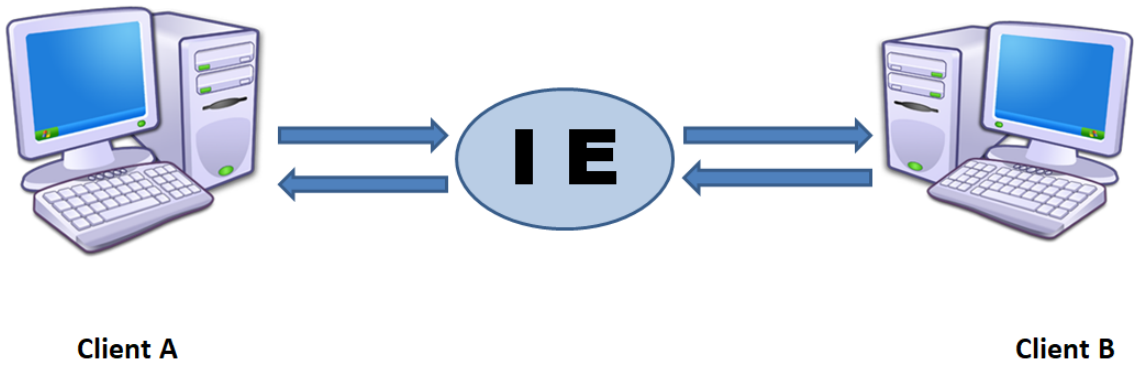


Figure 3 : Représentation d'un échange passant par IE[6]

B. Présentation Fonctionnelle

Le projet Infrastructure d'échange propose une Offre de Services Échanges que l'on appellera simplement OSE[7] qui permet à tous ses utilisateurs de mettre en place des échanges inter-applicatifs.

Lors de cette description fonctionnelle, certains termes utilisés sont spécifiques au projet et ils sont définis dans le tableau ci-dessus :

[6] Infrastructure d'Echange (voir figure 4)
[7] Offre de Service d'Echange (voir figure 4)

Terme	Définition et compléments
Chorégraphie	Ensemble de fonctionnalités parmi celles offertes par IE, incluant le transport de données et déclenché par un évènement unique (simple ou complexe). Chaque chorégraphie est affublée d'un identifiant par le demandeur.
Echange	Exécution d'une chorégraphie au sein d'IE. Chaque échange est affublé d'un identifiant par IE.
Rejeu	Nouvelle tentative d'envoi d'un message/fichier.
Reprise sur erreur	Rejeu sur un échange en erreur
Réémission	Rejeu d'un envoi depuis IE exécuté précédemment avec succès.
Rejeu	Nouvelle tentative d'envoi d'un message/fichier dont la dernière tentative d'envoi est en échec.
IE	Infrastructure d'Echanges
OSE	Offre de Services Echanges
OSMOSE	IHM de gestion des demandes de chorégraphie
ASTRE	Application de Supervision du Transport et de Reprise des Echanges

Figure 4 : Glossaire des termes utilisés au sein de l'entreprise[8]

Les échanges effectués au sein du projet sont classés par protocole selon le type de données à échanger. Les protocoles actuellement supportés sont les suivants :

Protocole de transport	Limites de taille des messages/fichiers	Contenu des messages/fichiers
HTTP (HTTPS)	2 Mo / 40 Mo avec pièces jointes	JSON SOAP XML Binaire Texte
FTP (FTPS)	100 Mo	
JMS	2 Mo / 40 Mo avec pièces jointes	
SFTP	100 Mo	

Figure 5 : Liste des protocoles supportés[9]

Les fonctionnalités proposées sont découpées par niveau de service représentées dans le tableau ci-après :

Niveau	Nom	Contraintes	Gouvernance
1	Initialisé	Quasi nul	Échanges
2	Industrialisé	Très faible	
3	Organisé	Faible	
4	Rationnalisé	Moyen	Données

Figure 6 : Les différents niveaux de flux proposés[10]

a. Niveau 1 : Initialisé

Le niveau 1 assure les fonctions de bases de support des échanges (transport, sécurité, traçabilité, reprise sur erreur, inventaire).

Pour les échanges utilisant les protocoles JMS (Java Message Service) ou FTP (File Transfert Protocol), chaque message (supporté par JMS) / fichier (supporté par FTP) est récupéré à intervalle régulier et traité par IE. En cas d'indisponibilité soit de l'application émettrice ou soit de l'application IE, la prise en charge du message/fichier est assurée

[8] Document fourni par l'entreprise
[9] Document fourni par l'entreprise
[10] Document fourni par l'entreprise

automatiquement. Les protocoles en entrée et en sortie et les typologies d'appel (synchrone ou asynchrone) doivent être identiques à l'exception de l'application de la sécurité.

Voici des exemples concrets pour illustrer :

- Une chorégraphie (voir définition dans la figure 4) de niveau 1 peut utiliser le protocole FTP en entrée et le protocole FTPS en sortie.
- Une chorégraphie de niveau 1 ne peut pas avoir un protocole FTP en entrée et un protocole de sortie en HTTP (ce que l'on appelle rupture protocole du niveau 2). Cette fonctionnalité ne sera donc disponible que si le client choisi un échange de niveau 2.

Pour chaque chorégraphie, les messages/fichiers récupérés par IE sont envoyés à un et un seul destinataire, sans aucune modification de contenu. Les échanges sollicitant un partenaire situé à l'extérieur du réseau Enedis sont pris en compte pour les fichiers, pas pour les messages, donc uniquement pour le protocole FTP.

Les échanges entre IE et ses partenaires peuvent être sécurisés avec les mécanismes standards de sécurité du transport (SSL[11] uni ou bidirectionnel) à l'exception des protocoles implémentant JMS.

Pour pouvoir superviser et suivre les chorégraphies qui ont été déployées en productions, tous les échanges exécutés sur l'Infrastructure d'Echanges sont tracés dans ASTRE. Pour les échanges en échec, un message d'erreur est affiché.

Pour le cas des échanges en échec, un message d'erreur est affiché. Ces échanges en erreur peuvent être repris, unitairement ou en masse, depuis ASTRE[12].

L'exemple suivant est une illustration du propos précédemment énoncé :

Une application A souhaite envoyer des documents à une application B, mais B étant indisponible au moment de l'envoie, l'échange n'est pas passé. Il est possible de rechercher cet échange sur la page supervision d'ASTRE et de relancer le processus d'Echange.

Les échanges non aboutis des chorégraphies JMS et HTTP avec pièces jointes ne peuvent être repris.

Donc seuls les échanges en FTP(S) ou SFTP (envoie de fichiers) ont une possibilité de retransmission si l'échange n'a pas abouti et seulement à la demande du client (la retransmission automatisée en cas d'échec n'est pas implémentée ici).

b. Niveau 2 : Industrialisé

En plus des fonctionnalités offertes dans le niveau 1, ce niveau propose d'autres fonctionnalités.

[11] SSL (Secure Socket Layer) : protocoles de sécurisation des échanges sur Internet. (Wikipedia)
[12] Astre : Application de Supervision du Transport et de Reprise d'Echange

Ce niveau permet d'accéder au routage multi-destinataires statique ou tout simplement l'envoie de données à plusieurs destinataires qui, par contre doivent figurer dans une liste de destinataires immuable sauf évolution de la chorégraphie.

Ci-dessous un exemple pour illustrer ces propos :

Pour une chorégraphie ayant « A » comme émetteur et « B » et « C » comme destinataires, si un nouveau destinataire « D » doit être ajouté, une évolution de la chorégraphie est nécessaire.

Le niveau 2 offre également une possibilité de rupture de protocole.

Exemple :

Une application A veut envoyer des fichiers à une application B. Le protocole proposé de base pour les fichiers est le FTP. Mais supposons que B ne possède pas de serveur FTP mais plutôt un serveur JMS donc elle ne pourra recevoir que des messages, l'infrastructure d'échange permet de faire ce changement de protocole.

Ce niveau 2 permet également l'agrégation en un seul fichier ou le découpage de fichiers ou de messages selon des critères simples.

Les échanges d'une chorégraphie peuvent être synchronisés par l'IE c'est-à-dire l'acquittement est retourné à l'appelant sans attendre la réponse de l'appelé.

Pour chaque chorégraphie, chaque destinataire peut choisir de ne pas recevoir les messages/fichiers au fil de leur traitement par IE, mais de manière groupée, une fois par jour.

IE peut réaliser la validation des messages/fichiers XML par comparaison avec une XSD (Xml Schema Description).

Les messages conservés peuvent être consultés dans ASTRE, hors pièce jointe (contenu binaire) et les fichiers conservés peuvent être téléchargés.

c. Niveau 3 : Urbanisé

Ce niveau permet d'accéder aux échanges avec les partenaires hors du réseau Enedis, ainsi qu'à une forme élémentaire de routage dynamique mono-destinataire. Il se distingue essentiellement par sa composante organisationnelle forte : l'équipe IE s'implique dans la mise en place des échanges, entre les partenaires impliqués, mais aussi vis-à-vis d'entités tierces.

La mise en place d'un échange sortant du réseau interne Enedis nécessite le passage par des infrastructures de sécurité et des restrictions de certains protocoles et formats de message.

L'équipe IE gère les aspects administratifs et la gouvernance associée à la mise en place de tels échanges.

Pour chaque chorégraphie, les messages/fichiers reçus peuvent être envoyés à un et un seul destinataire parmi une liste propre à la chorégraphie considérée, en fonction du code du

destinataire qui peut être contenu dans le nom (standardisé) du fichier entrant dans IE ou dans l'entête (standardisée) du message entrant dans IE

Contrairement au type de routage proposé dans le niveau 2, ce type de routage est dynamique.

IE prend en charge la sécurité des contenus échangés (tokens WS-Security et OAuth, chiffrement et déchiffrement par clés symétriques ou asymétriques, encodage de messages en base 64) à partir de ce niveau.

IE peut limiter le nombre d'appels concurrents à l'un de ses services exposés ou IE permet de suspendre les sollicitations à une application partenaire lorsque celle-ci est déclarée indisponible. Cela permet de protéger les applications appelées par IE.

Les contenus qui ont été correctement acheminés vers le(s) destinataire(s) peuvent être réémis, unitairement ou en masse, depuis ASTRE. On appellera ce mécanisme la réémission.

d. Niveau 4 : Rationalisé

Ce niveau vise à découpler les applications et à garantir le respect de la stratégie Référentiels et services d'Enedis. IE peut assurer la transformation des messages ou des fichiers et l'orchestration d'appels de services pour remplir ces objectifs. L'équipe IE doit pour ce faire accompagner les projets partenaires dans le cadrage des besoins métiers et concevoir les chorégraphies.

IE peut effectuer le routage des appels vers un ensemble de partenaires ou de services en fonction des données reçues dans le message ou fichier entrant, à partir de données obtenues auprès de partenaires ou de ses propres données. IE gère dynamiquement l'indisponibilité de chaque partenaire en se basant sur les échanges en erreurs. IE peut transformer le contenu des messages ou des fichiers comme le renommage, mappings ou transcodifications. IE permet l'orchestration stateless d'appels de services applicatifs pour les protocoles HTTP.

Pour rappel, tous les niveaux embarquent toutes les fonctionnalités du niveau en dessous.

Ci-dessous est représenté un tableau récapitulatif des différents niveaux d'offres proposées par l'Infrastructure d'échange.

Niveau 1	Niveau 2	Niveau 3	Niveau 4
Mono-protocole	Multi-protocoles	+ Transport hors RIN/ZHB	
Mono-destinataire	Routage statique	Routage dyn. simple	Routage dynamique sur données Métier
Sécurité du transport		+ Sécurité des contenus échangés	
Supervision Exécut°	+ Visualisation du contenu		+ Supervision Métier
Reprise sur erreur simple		+ Réémission	+ Aide à la décision pour rejeu
	Rétention standard du contenu		Rétention étendue du contenu
	Rupture temporelle		
	Groupage		
	Dégroupage		
	Validation du format XML (par XSD)		Contrôles avancés de données
		Régulation des sollicitations	
		Indispo programmée	Gestion dynamique de l'indisponibilité
			Transformation des contenus
			Orchestration stateless

13

Figure 7 : Tableau récapitulatif des différents niveaux d'offres proposées (Document fourni par l'entreprise)

C. Présentation Technique

Chaque échange effectué au sein du projet est caractérisé par une chorégraphie et est affublé d'un identifiant. Un échange peut être synchrone ou asynchrone selon le type de protocole choisi par les partenaires ayant demandé l'échange. Peu importe le niveau de chorégraphie déterminé, chacune est basée sur du niveau 1 à laquelle est ajoutée les fonctionnalités du niveau supérieur auquel il est destiné.

a. Responsabilité d'une brique d'échanges

Un échange entre deux applications est caractérisé par deux demi-échanges ayant chacun leur propres caractéristiques qui sont résumés sur l'image ci-dessous :

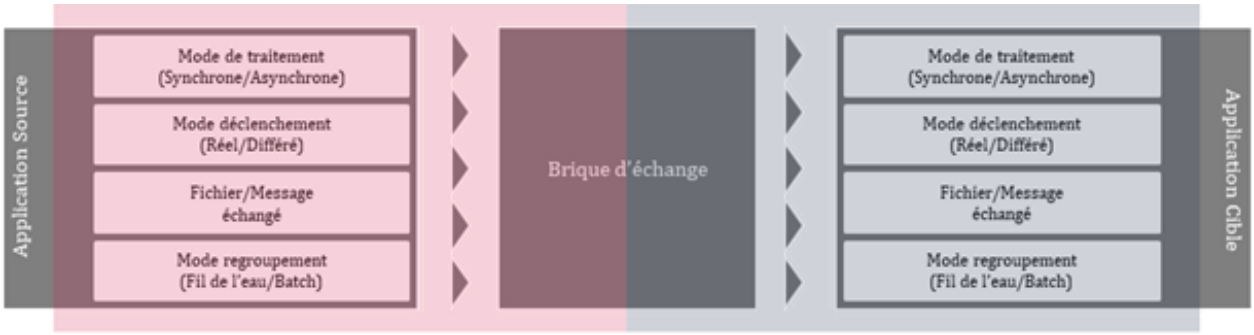


Figure 8 : Architecture d'une brique d'échange

[13] RIN : Réseau d'Interconnexion National
ZHB : Zone d'Hébergement de Base

Cette division de l'échange fait peser de lourdes responsabilités et contraintes sur la brique d'échanges. Elle s'articule autour des axes techniques, fonctionnel, architectural et organisationnel.

b. Instance et définition

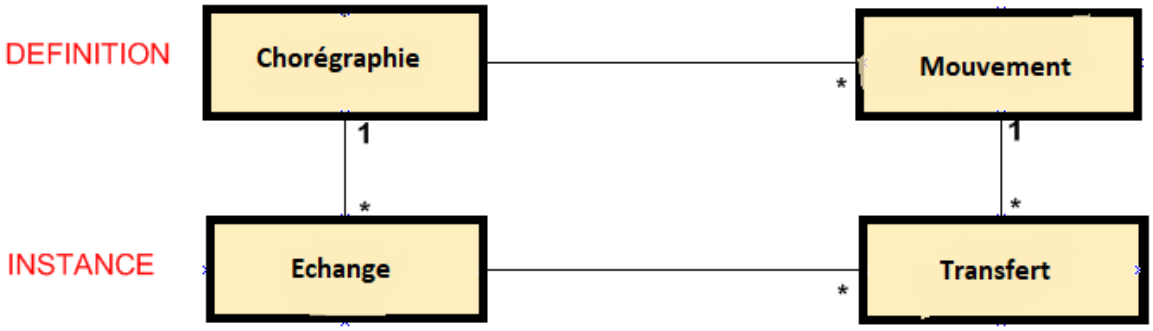


Figure 9 : Architecture de définition et d'instance

Si nous détaillons l'image précédente, une chorégraphie est constituée de plusieurs mouvements dont un entrant et un ou plusieurs sortants. Le mouvement entrant représente l'application entrante et le ou les mouvements sortants représentent la ou les applications sortantes. Donc à chaque application est attribué un mouvement.

Chaque mouvement est spécifié par une route Camel que l'on verra dans la description complète de Camel Apache un peu plus tard.

Un échange est une instance de chorégraphie ou son exécution et est constitué de plusieurs transferts. Un échange a lieu lors d'un appel à une chorégraphie.

c. Structure d'une chorégraphie

Dans le projet actuel, l'architecture d'une chorégraphie a été définie comme suit :

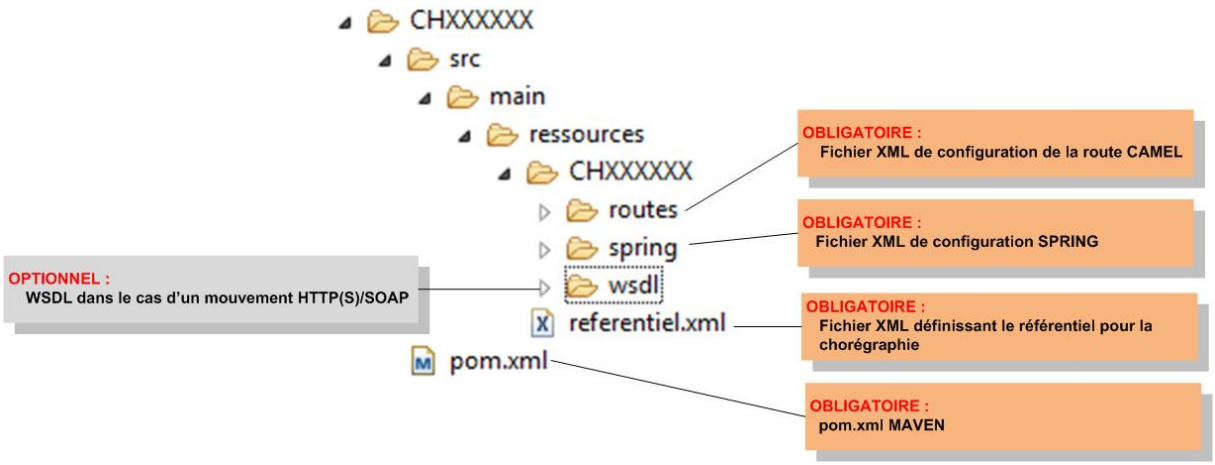


Figure 10 : Structure d'une chorégraphie ENEDIS document interne

L'archive CHXXXXXX.jar constitue le package global de la chorégraphie. Lorsqu'on extrait cette archive, on obtient l'arborescence présentée dans le schéma précédent.

Le code de la chorégraphie est représenté par le nom du sous-répertoire CHXXXXXX. Ce sous-répertoire contient :

- Le répertoire « routes » où il y a les routes Camel de la chorégraphie.
- Le répertoire « spring » avec les configurations spring de la chorégraphie
- Selon les cas : les wsdl, les XSD en cas de chorégraphie web-service SOAP, les XLS en cas de transformations du format de requête pour l'appel du web Service.
- Le fichier XML referentiel.xml contient toutes les informations du référentiel relatives à la chorégraphie.

  Exemple : Les partenaires entrants et sortants de la chorégraphie (créés si n'existent pas déjà), la durée de conservation des échanges (créée si n'existe pas), les mouvements de la chorégraphie avec le nom de leurs paramètres de connectivité.

- Le fichier pom.xml pour la configuration maven[14]

Ce package ne doit contenir aucun code java.

d. Architecture de fonctionnement des Chorégraphies

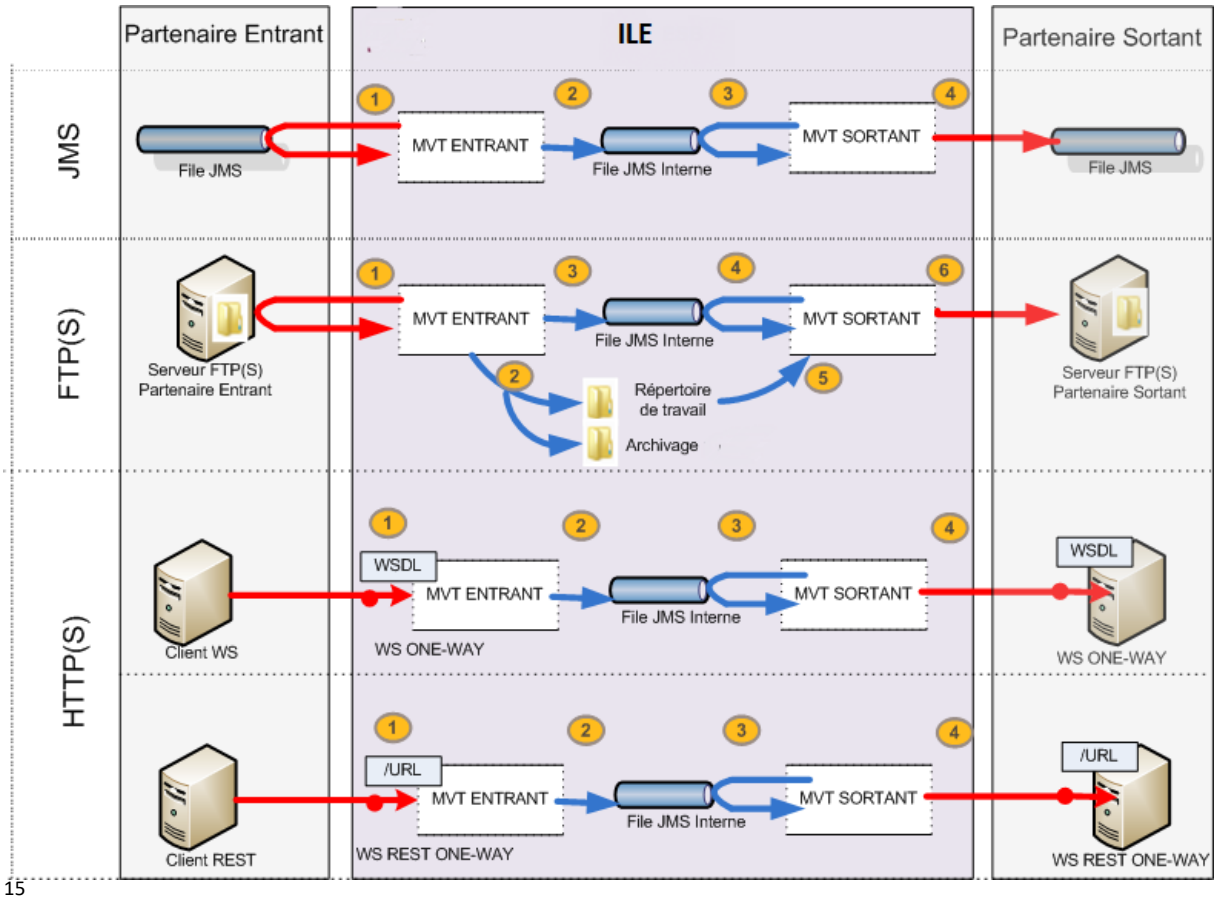
i. Chorégraphie « one-way » ou asynchrone

Ce type de chorégraphie est dit asynchrone car l'appel fait par l'application entrante ne nécessite pas forcément une réponse immédiate.

Par exemple, l'envoi de fichier(s) à l'aide du protocole FTP ne demande pas forcément une réponse de l'application appelée. Le fichier peut être transmis à l'application destinatrice sans que celle-ci n'émette une réponse.

Ce type d'appel peut être satisfait par tous protocoles comme illustré dans le diagramme représenté dans la figure 11. Les nombres affichés représentent la chronologie des échanges.

[14] Couramment appelé Maven, Apache Maven est un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier.



15

Figure 11 : Diagramme représentant les appels asynchrones

- Protocole JMS (Java Message Services) :

  La route entrante récupère le message en entrée dans une file JMS attribuée à l'application entrante. Elle va le stocker dans la file JMS interne de la plateforme d'échange d'où la route sortante viendra le récupérer pour le déposer dans la file JMS appartenant au récepteur. Le destinataire d'un message JMS peut être une « Queue » ou un « Topic ». Il s'agit d'une « Queue » quand un seul message sera reçu par exactement un consommateur. S'il n'y a pas de consommateurs disponibles au moment où le message est envoyé, il sera conservé jusqu'à ce qu'un consommateur disponible puisse traiter le message. Si un consommateur reçoit un message et ne le reconnaît pas avant la fermeture, le message sera remis à un autre consommateur. Une file d'attente peut avoir de nombreux consommateurs avec des messages équilibrés entre les consommateurs disponibles. Tandis que dans le cas d'un « Topic », un message publié est transmis à tous les abonnés qui sont intéressés. Ainsi, zéro à plusieurs abonnés recevront une copie du message. Seuls les abonnés ayant un abonnement actif au moment où le courtier reçoit le message recevront une copie du message.

- Protocole FTP(S) :

  La route entrante contenue dans le mouvement entrant contient l'emplacement du fichier de l'application entrante. Elle va ensuite déposer le fichier dans le

[15] ILE : Infrastructure légère d'échange

répertoire de travail et le répertoire d'archivage. Ce dernier répertoire est utilisé pour la réémission[16] de fichiers ou le rejeu[17]. La file JMS interne est utile pour stocker le nom des fichiers qui ont été déposés dans le répertoire de travail. Elle sert également à notifier à la route de sortie qu'un fichier et en attente pour qu'elle puisse le traiter. C'est lors de ce passage que le fichier sera traité selon les fonctionnalités choisies par l'utilisateur. S'il y a un renommage sur le fichier, le nom dans la file JMS interne sera également modifié. La route du mouvement sortant va ensuite chercher le nom du fichier dans la file JMS et récupère le fichier correspondant dans le répertoire de travail et dépose ce fichier dans le répertoire de l'application sortante dont le chemin est spécifié dans la route du mouvement sortant.

- Protocole HTTP(S) :

  Les appels traités par le protocole HTTP(S) sont généralement des appels à des Web Services REST ou SOAP. Lorsqu'un appel est fait en entrée, une notification est mise dans file JMS pour signifier qu'un appel a été effectué et que le service sortant doit y répondre. Comme l'appel est asynchrone, la réponse n'est pas obligatoirement immédiate. Dans le cas d'un Web Service SOAP, le client entrant appelle la plateforme d'échange grâce à un WSDL (Web Services Description Language ou aisément appelé WSDL fourni par l'application sortante. Et c'est la plateforme d'échange qui appelle l'application sortante avec le même WSDL. Pour les Web service REST, l'appel se fait sur une url vers la plateforme d'échange et la plateforme appelle l'url de sortie. Tous les traitements de données se font au sein de la plateforme d'échanges lors du transfert

ii. Chorégraphie « two-way » ou synchrone

Dans ce type d'appel, quand l'application entrante appelle, l'application de sortie se doit de répondre immédiatement. Ce cas concerne principalement les appels aux Web Services par le biais les protocoles HTTP REST ou SOAP. Pour le traitement des chorégraphies synchrones, la file JMS interne n'est pas nécessaire car l'appel se fait directement et la réponse est immédiate. Comme dans le cas d'une chorégraphie asynchrone, le partenaire entrant n'appelle jamais le partenaire sortant directement. Il doit toujours passer par l'infrastructure d'échange qui s'occupera de transférer les appels et d'effectuer les opérations correspondantes si besoin.

Ce type d'appel est illustré dans l'image 12.

La chronologie d'appel est représentée par les chiffres indiqués.

[16] Retransmission des données (à la demande du client)

[17] Retransmission des données en cas d'échec (à la demande du client)

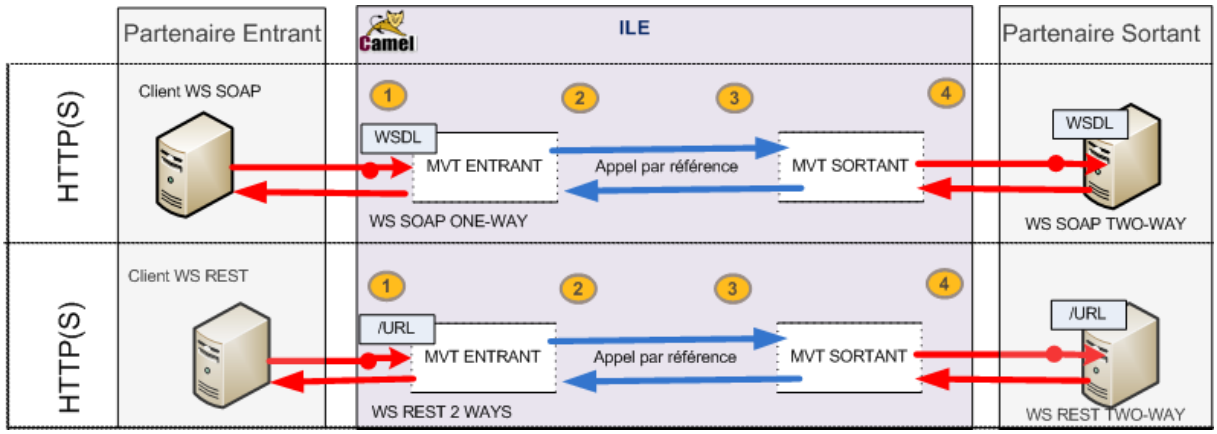


Figure 12 : Diagramme représentant les appels synchrones

e. Architecture technique de l'infrastructure d'échange

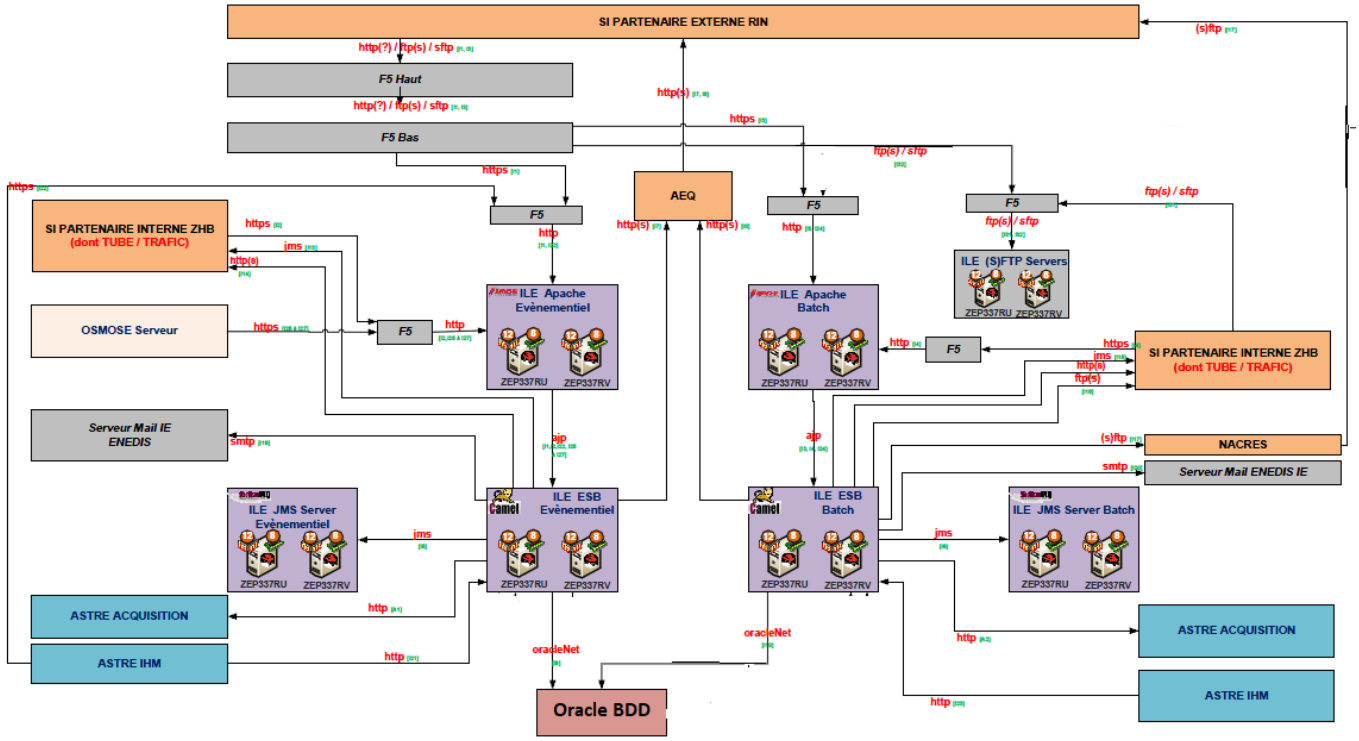


Figure 13 : Architecture technique de l'infrastructure d'échange

Ci-dessus est représentée l'architecture générale du projet. Cette image spécifie toutes les briques appelées lors d'un déploiement et d'un appel à une chorégraphie.

Pour les applications clientes qui font appel à l'infrastructure d'échange, il existe les partenaires internes qui appartiennent au réseau Enedis, et les partenaires externes qui ne font pas partie de ce réseau mais qui communiquent avec les différents projets au sein d'Enedis.

Ces applications externes doivent passer par des murs de protections pour pouvoir appeler la plateforme d'échanges. En entrée, les appels fichiers passent par un proxy que l'on appelle Nacres, et les appels web services passent par un proxy nommé AEQ.

En sortie, le proxy est le même pour tout protocole que l'on appelle F5. Les applications internes ne passent par aucun proxy et font appel directement à l'appel des chorégraphies.

Pour des raisons de sécurité, toutes les url d'appel sur un web Service ILE[18] passent par un serveur Apache qui s'occupe de transformer les url externes au format utilisé par la plateforme d'échange.

Pour le déploiement des chorégraphies, chaque environnement de déploiement possède deux types de serveurs :

- Les chorégraphies de type évènementiel c'est-à-dire les appels aux web Services ou encore l'envoi d'e-mails ou de message JMS, sont déployées dans un serveur Tomcat que l'on appelle serveur EVT pour Evènementiel et possède un serveur JMS du même type pour les files internes.
- Tandis que les chorégraphies de type fichiers comme pour le protocole FTP(S) sont déployées dans un serveur Tomcat nommé BTC pour Batch et possède aussi leurs propres serveur JMS.

Chacun de ces serveurs BTC ou EVT possède deux instances qui font une répartition de charge (« load balancing »)[19] entre chaque action effectuée sur ce serveur.

Après le développement, chaque chorégraphie est déployée dans le serveur correspondant où les routes Camel seront démarrées. A partir de là, les chorégraphies sont prêtes à être utilisée.

f. Bases de données

Le stockage des données est obligatoire dans ce genre de projet. Pour le déploiement d'une chorégraphie, trois schémas principaux sont définis:

- REFERENTIEL : Comme son nom l'indique, ce schéma contient les éléments nécessaires au référencement de la chorégraphie. Il contient les informations générales sur les chorégraphies. Exemple : La liste des niveaux, les types des connectivités, les mouvements, la liste des partenaires, etc...
- RUNTIME : Ce schéma comprend les informations utiles sur les stations d'exécutions et de déploiement d'une chorégraphie. Exemple : Une chorégraphie déployée a comme statut 1, quand les routes sont démarrées, le statut passe à 2 et si la

[18] Infrastructure légère d'échange : Terme interne que l'on utilise au sein du projet pour nommer la plateforme d'échange

[19] Le *load balancing* est un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe. Ces techniques permettent à la fois de répondre à une charge trop importante d'un service en la répartissant sur plusieurs serveurs, et de réduire l'indisponibilité potentielle de ce service que pourrait provoquer la panne logicielle ou matérielle d'un unique serveur. Wikipedia

chorégraphie est supprimée, le statut est à 3. On peut démarrer une route manuellement en changeant directement son statut dans la base de données.

- CONNECTIVITE : Ce schéma contient les informations sur les connectivités d'une chorégraphie c'est-à-dire les informations communiquées par les partenaires entrants et sortants sur les applications qui vont appeler et être appelé par la route.

  Exemple : les url d'appels entrants pour les web services, le Host, le login et mot de passe, et le répertoire sur le serveur sur lequel le fichier va être récupéré en entrée et déposé en sortie.

g. Ose IHM

Pour les partenaires souhaitant faire appel à la plateforme d'échange, une IHM (Interface Homme Machine) a été mise en place. Ose IHM est actuellement séparée en deux briques distinctes dont la première brique concerne la création des demandes, et la seconde concerne la supervision des chorégraphies mises en production. La première étant nommée ASTRE DEMANDE, et la seconde ASTRE SUPERVISION.

Le mode de fonctionnement de l'IHM vis-à-vis des chorégraphies est démontré sur l'image ci-dessous :

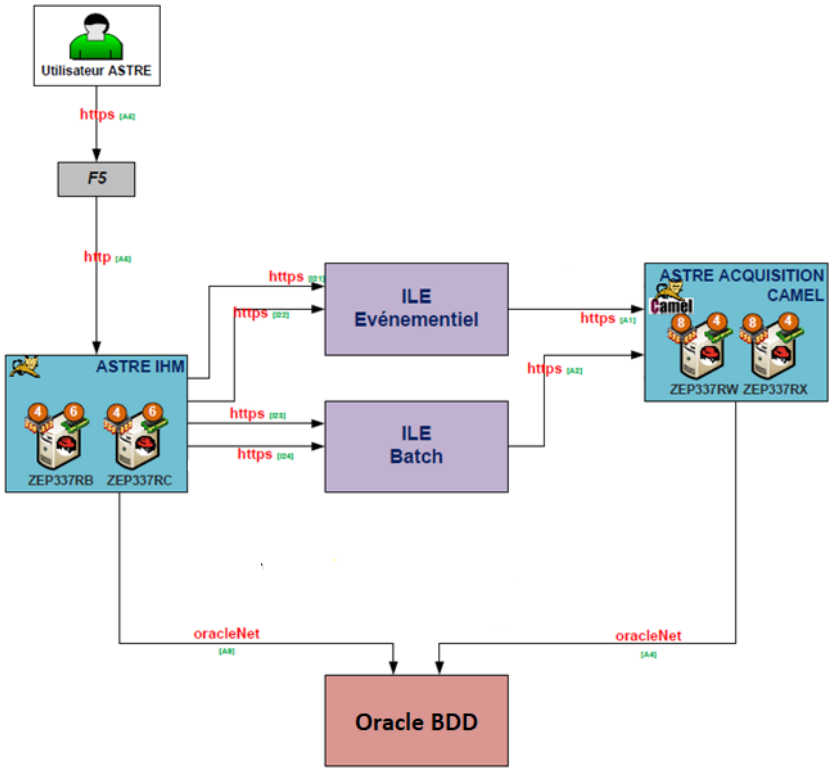


Figure 14 : Représentation du fonctionnement d'OSE IHM

Lorsque les chorégraphies sont créées sur l'IHM, cette dernière les déploie dans les serveurs Tomcat Evènementiels et Batch à partir desquels interviendra la supervision des transferts.

i. Astre Demande

C'est sur cette partie de l'interface que les partenaires font leur demande et renseigne toutes les informations relatives à la chorégraphie. Une chorégraphie est spécifique à chaque partenaire. Les chorégraphies de niveau 1, niveau 2 et niveau 3 sont actuellement disponibles en développement automatique.

Comment fonctionne le développement automatique ? Le client crée sa chorégraphie sur Astre Demande en renseignant les descriptions nécessaires (protocoles, applications entrante et sortante et toutes les différentes options). Ces renseignements seront stockés dans une base de données attribuée à la partie IHM. Un programme codé en java s'occupera ensuite de récupérer ces données sur la base pour constituer les fichiers nécessaires à la chorégraphie. Quand tous les fichiers nécessaires sont générés, ils sont empaquetés (voir le package de l'image 10), et déposés sur l'Artifactory[20] et SVN(Subversion) par un job Jenkins.

Le processus de développement automatique est présenté sur le schéma suivant :

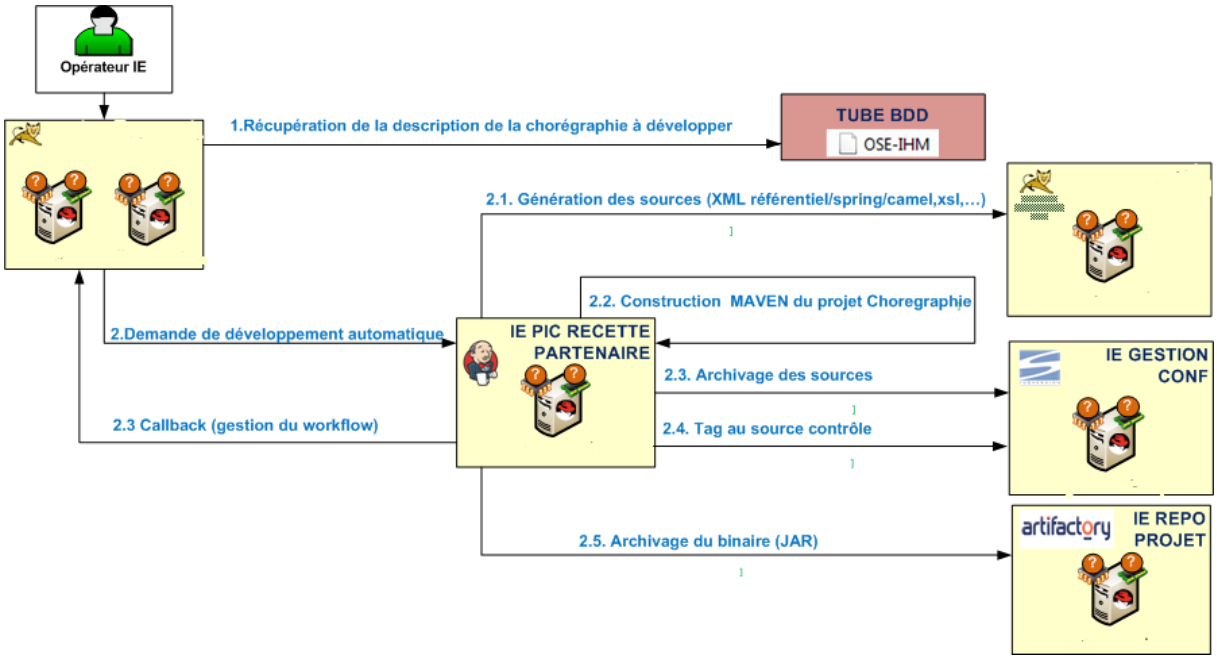


Figure 15 : Processus du développement automatique

Ce « .jar » est ensuite récupéré de l'Artifactory par l'environnement de recette et est déposé dans le Nexus[21] pour la mise en production après la validation de la phase de recette. Les paramètres de connectivités sont renseignés et la chorégraphie est déployée en production. Il suffit de redémarrer les environnements de production et les routes sont démarrées et la chorégraphie peut être exploitée.

Le déploiement d'une chorégraphie en production se fait comme suit :

[20] Artifactory : Répertoire de dépôts sur lequel les chorégraphies sont déposées pour le développement

[21] Nexus : Gestionnaire de dépôt sur lequel les chorégraphies sont déposées avant la mise en production

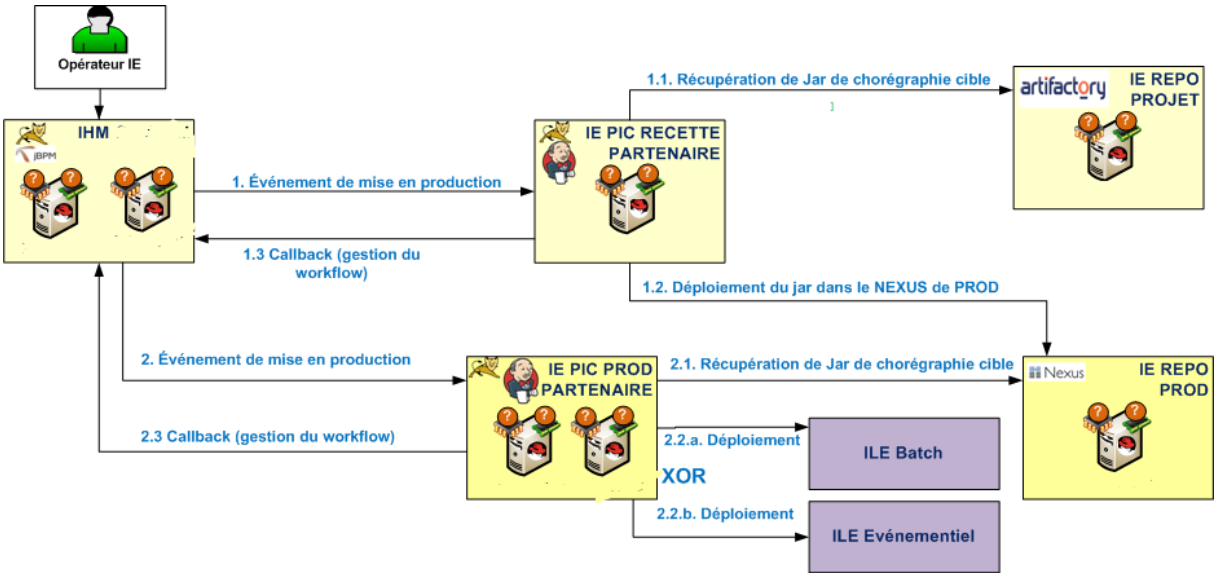


Figure 16 : Processus de déploiement en production

Les chorégraphies de niveau 4 peuvent être générées sur l'IHM mais les fonctionnalités doivent être développées manuellement. Pour ce faire, le client créé sa demande de flux en renseignant toutes les informations nécessaires. Les fonctionnalités qu'il aura choisies détermineront si la chorégraphie passe en développement manuel ou automatique. Lorsque le paquet sera généré par l'IHM, il sera déposé sur SVN afin que le développeur puisse le récupérer pour y ajouter les fonctionnalités demandées pour cette chorégraphie. Quand le développement manuel est terminé, le développeur dépose le « .jar » sur l' « Artifactory » où il pourra être récupéré par un job jenkins.

La création de demande sur l'IHM est dirigée par un « workflow »[22] bien précis depuis la page de création de la chorégraphie jusqu'à la page de déploiement en production.

Si le développement est manuel, le « workflow » s'arrête aux saisies des informations complémentaires (voir figure 17). Et lorsque le développeur aura déposé son « .jar » sur l'Artifactory, le « workflow » reprend exactement là où il s'est arrêté.

La schématisation du « workflow » est représentée ci-dessous :

[22] Un workflow, anglicisme pour flux de travaux, est la représentation d'une suite de tâches ou opérations effectuées par une personne, un groupe de personnes, un organisme, etc. Le terme flow (flux) renvoie au passage du produit, du document, de l'information, etc., d'une étape à l'autre. Wikipedia

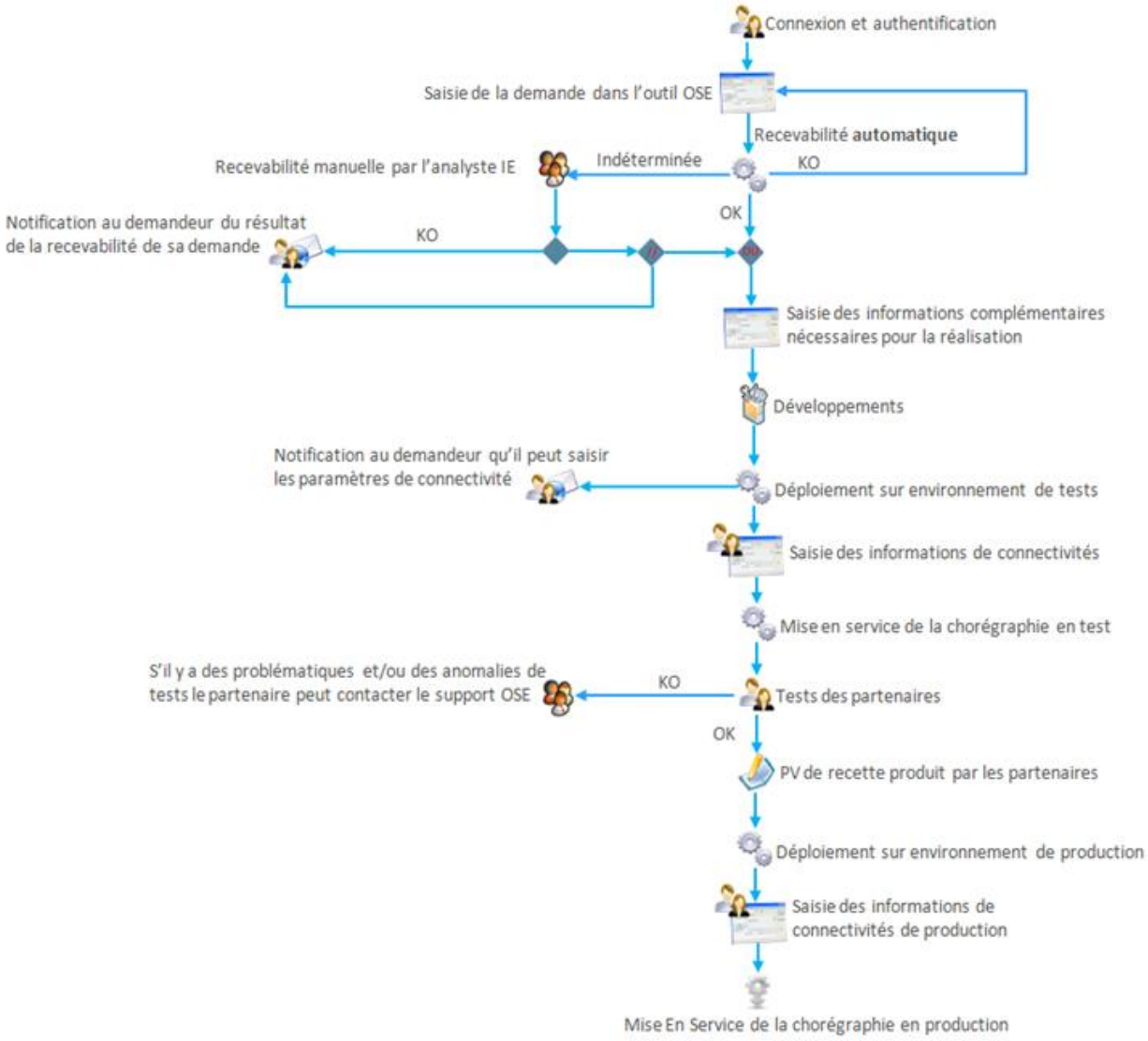


Figure 17 : Schématisation du moteur d'avancement de l'IHM

ii. Astre Supervision

L'IHM possède également une brique de supervision pour les chorégraphies en production. Dès lors qu'une chorégraphie est mise en production, sa supervision est systématique. Les échanges sont surveillés, tous les transferts sont listés qu'il soit OK ou KO. Si le transfert est en échec, il est possible de repasser le transfert à la demande du client. Ce dernier peut être fait de manière individuelle ou en masse. Pour les statuts OK, il est aussi possible de faire les re-transférer de manière individuelle ou en masse. Le principe de rejeu n'est disponible que pour les transferts de fichier donc pour le protocole FTP(S) ou SFTP.

La supervision est une partie très importante du projet car c'est l'outil qui permet aux clients de suivre si leurs transferts se sont bien passés ou non. A chaque fois que le partenaire appelle la chorégraphie, un identifiant est attribué à chaque transfert et cet identifiant est inséré en base de données avec le statut du transfert. Si c'est un appel fichier, le fichier transféré peut être téléchargé sur la page de supervision. Si c'est un appel web Service, le corps de la requête ainsi que le corps de la réponse sont également disponible sur cette page.

iii. Base de données d'Ose IHM

La partie IHM possède deux schémas de base de données. Le premier est utilisé pour la persistance des données au niveau métier c'est-à-dire tout ce qui est saisi au niveau de l'IHM.

Exemple : nom de la chorégraphie, protocole, les applications entrantes et sortantes…

Tandis que l'autre est utilisé pour la persistance du moteur d'avancement ou encore le « workflow ».

h. Les technologies utilisées

La technologie principale du projet est Camel Apache. C'est celle qui construit les routes pour les chorégraphies. Un programme Java permet de générer les sources nécessaires au développement automatique. Ces sources sont ensuite mises en paquets par un job Jenkins et qui le déposera ensuite sur l'Artifactory. Un autre job Jenkins récupère ce .jar sur l'Artifactory pour le déposer sur répertoire de dépôt de la production. Lorsqu'il se trouve sur l'environnement de production, ils sont déployés dans les serveurs Tomcat soit Evènementiel soit Batch selon le type de protocole de la chorégraphie. Les files JMS internes sont gérées par Apache ActiveMQ.

Ose IHM est entièrement développée JSF/Primefaces au niveau de la vue, Java, Spring pour le coté service et Hibernate pour persistence de données. Et le moteur qui gère le « workflow » est JBPM.

La base de données utilisée au sein du projet est Oracle.

Toutes les installations sur les machines et les gestions des configurations sont gérées par Ansible. Et la gestion de version s'effectue sur SVN.

La synchronisation des différentes technologies au sein du projet est représentée comme suit :

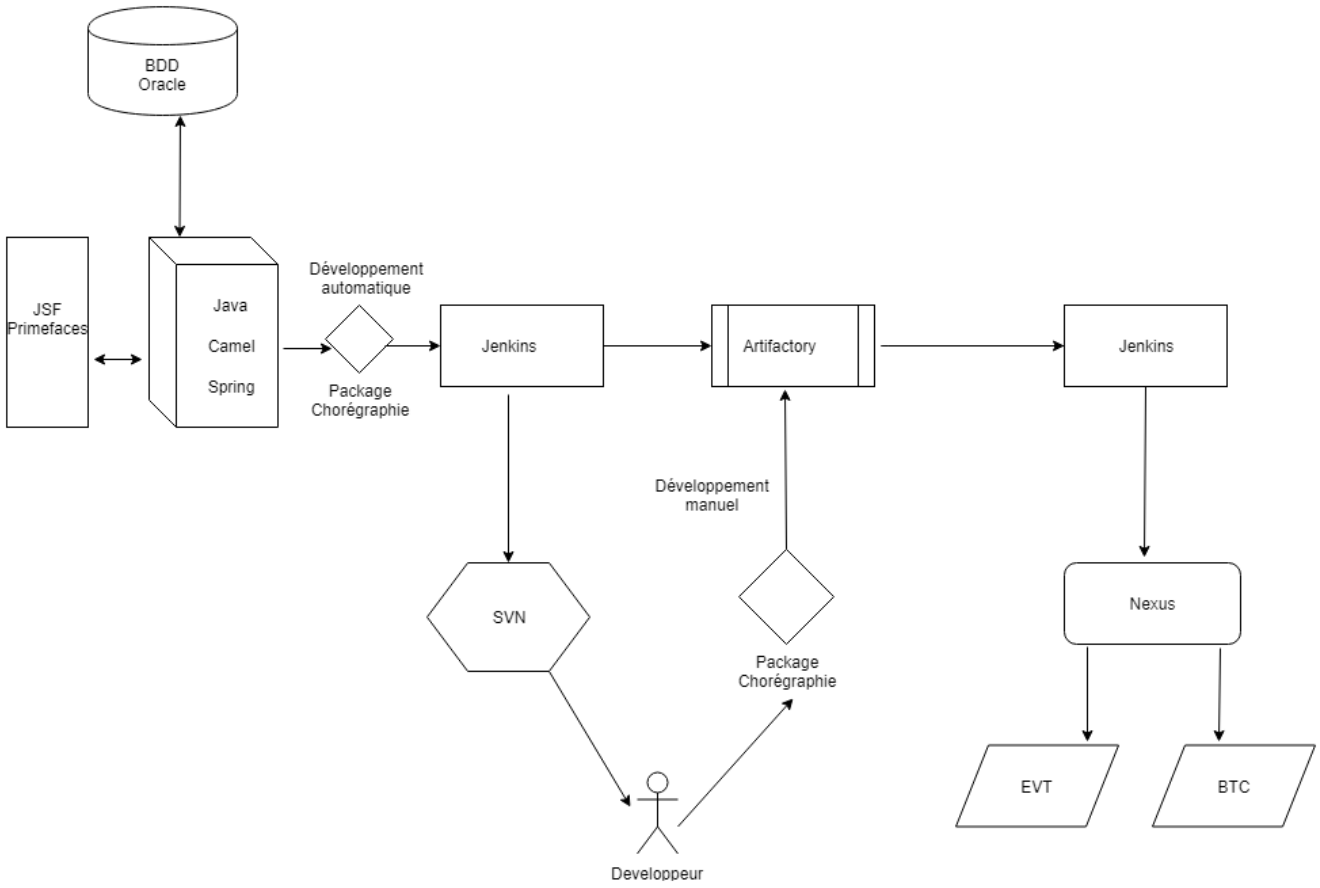


Figure 18 : Architecture des technologies utilisées

III. SOLUTIONS UTILISEES ACTUELLEMENT: LES « ENTERPRISE INTEGRATION PATTERNS » ou EIP

Les différentes applications existantes actuellement sont utilisées rarement de façon isolée. Peu importe l'application que nous pouvons avoir sous la main, elle fonctionne toujours mieux en s'intégrant à d'autres applications. Cependant, l'intégration de ces applications ne sont pas toujours évidentes et font souvent face à certains problèmes :

- Les réseaux ne sont pas toujours fiables : Les solutions d'intégrations doivent transporter des données d'une machine à une autre sur les réseaux. Par rapport à un processus exécuté sur un seul ordinateur, l'informatique distribuée doit être préparée pour faire face à un ensemble beaucoup plus large de problèmes possibles. Deux systèmes à intégrer peuvent être séparés par de nombreux kilomètres entre eux et les données doivent transiter par des lignes téléphoniques, des segments de réseau local, des routeurs, des commutateurs, des réseaux publics et des liaisons par satellite. Chacune de ces étapes peut entraîner des retards ou des interruptions.

- Les réseaux sont lents : L'envoi de données sur un réseau est plus lent que la création d'un appel de méthode locale. Concevoir une solution largement distribuée comme

nous le ferions pour une seule application pourrait avoir des conséquences désastreuses sur les performances.

- Les deux applications sont différentes : Les solutions d'intégration doivent transmettre des informations entre des systèmes utilisant différents langages de programmation, plates-formes d'exploitation et formats de données. Une solution d'intégration doit pouvoir s'interfacer avec toutes ces différentes technologies.

- Le changement est inévitable : Les applications changent avec le temps. Une solution d'intégration doit suivre l'évolution des applications connectées. Les solutions d'intégration peuvent facilement être prises dans un effet d'avalanche de modification, si un système change, les systèmes dépendants seront affectés. Une solution d'intégration doit minimiser les dépendances d'un système à l'autre en utilisant un couplage souple entre les applications.

Au fil du temps, les développeurs ont surmonté ces défis avec quatre approches principales :

- Le transfert de fichier : Deux applications peuvent s'envoyer des fichiers entre elles. Elles doivent définir à l'avance le nom de chaque fichier, l'emplacement ou le format utilisé. Elles se conviennent également du moment d'envoi du fichier, de la lecture et de laquelle de ces applications supprimera le fichier.

- Une base de données partagée : Plusieurs applications partagent le même schéma de base de données, situé dans une seule base de données physique. De ce fait, aucune donnée ne sera transférée d'une application à l'autre.

- L'invocation de procédure à distance : Une application expose certaines de ses fonctionnalités afin de pouvoir y accéder à distance par d'autres applications en tant que procédure distante. La communication s'effectue en temps réel et de manière asynchrone.

- La messagerie : Une application publie un message sur un canal de messagerie commun. D'autres applications peuvent y accéder et les lire. Les applications se mettent d'accord sur un canal ainsi que sur le format du message. La communication est asynchrone.

Dans cette section, ce sont les services de messagerie que nous allons aborder. Une des solutions qui permet de répondre à cette approche est les Enterprise Integration Patterns.

A. Définitions

Les EIP sont des patterns qui permettent de normaliser les échanges de messages dans un système asynchrone. Comme beaucoup de patterns, ces notions sont généralement connues et les EIP ne font que récapituler, normaliser et nommer de nombreux concepts comme ce que doit contenir les messages ou ce qu'il faut faire pour router un message. Les EIP trouvent parfaitement leurs places dans une infrastructure SOA (Service Oriented Architecture) que ce soit de manière transparente ou apparente.

B. Architecture d'un EIP

Les EIP sont des solutions testées pour des problèmes spécifiques rencontrés pendant de nombreuses années dans le développement de systèmes informatiques. Ce qui est d'autant plus important, c'est qu'elles sont indépendantes de la technologie, ce qui signifie que le langage de programmation ou le système d'exploitation utilisé importe peu. L'architecture d'un système de messagerie dans un EIP se fait comme suit :

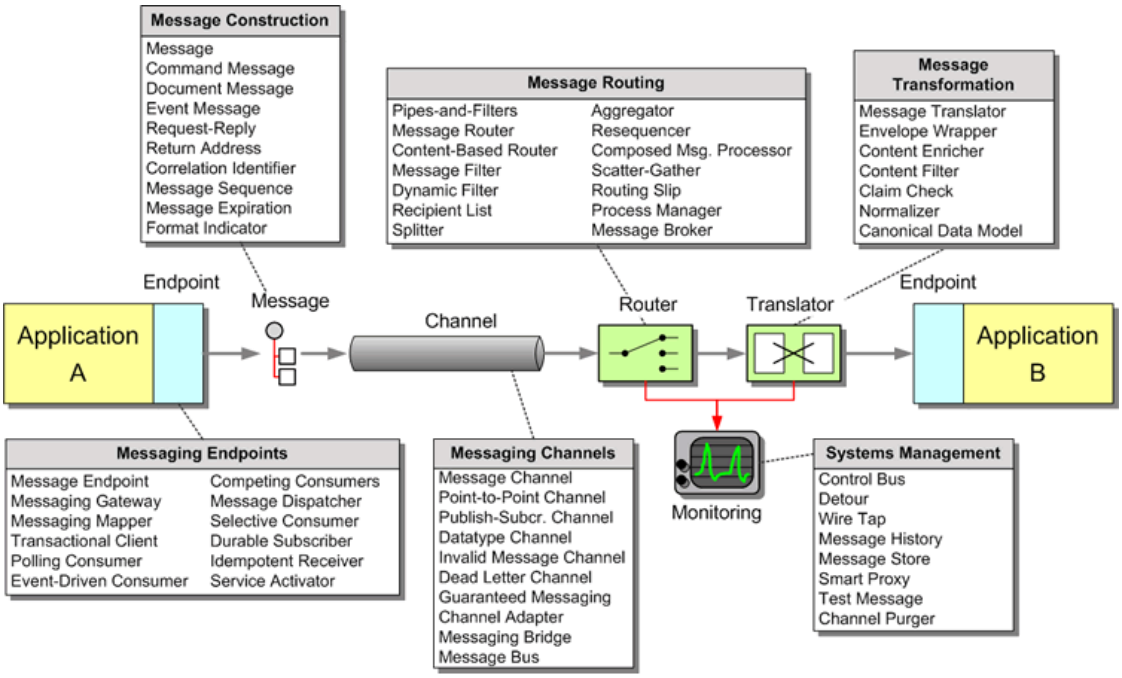


Figure 19 Architecture d'un EIP

a. Les « Endpoints»

Un « Endpoint » est une interface entre une application et un système de messagerie. Il existe l' « Endpoint » émetteur qui est parfois appelé proxy ou consommateur de service, responsable de l'envoi de messages depuis l'application entrante. De même, il existe un « Endpoint » destinataire ou parfois appelé service, chargé de recevoir les messages qui arrivent vers l'application sortante.

Le code des « Endpoints » de message est personnalisé pour les deux applications clientes du système. Les formats de messages transférés ne sont pas connus du reste de l'application, ni des canaux de messagerie ni des autres parties de la communication par messagerie. L'application sait seulement qu'il doit envoyer un message vers une autre application ou recevoir des messages d'une autre application. C'est le code des « Endpoints» de message qui s'occupe de récupérer les commandes ou les données, et de la transformation de ces données en messages avant de les envoyer sur un canal de messagerie particulier. L' « Endpoint » récepteur en extrait le contenu et le transmet à l'application réceptrice. C'est cette section qui gère les problématiques d'interconnexion avec l'extérieur. Les « Endpoint » sont placés comme suit dans une structure d'EIP :

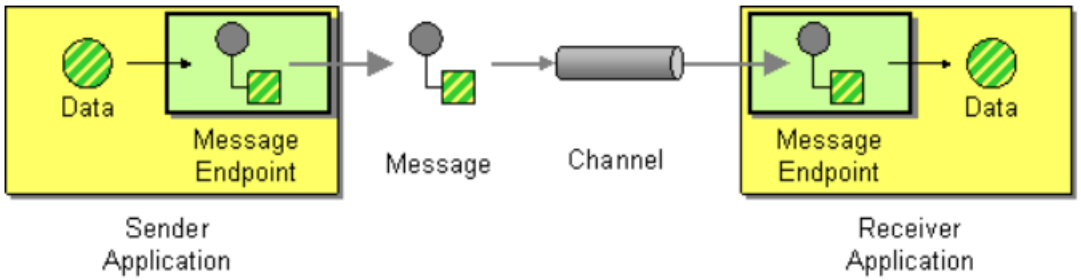


Figure 20 : Représentation des Endpoints

b. Les Messages

Un message est défini comme la plus petite unité de transmissions de données dans un système de messagerie. Le message lui-même peut avoir une structure interne composée des éléments suivants :

- Un entête qui contient les métadonnées relatives aux messages.
- Un corps contenant le message entier dans sa forme brute.
- Une pièce jointe qui contient les données supplémentaires pouvant être ajoutées aux messages.

Il faut savoir que cette division en entête, corps et pièce jointe est un modèle abstrait du message. C'est l'implémentation du composant sous-jacent (ici Camel Apache, que nous verrons un peu plus tard) qui va décider de ce qui sera réellement placer dans les entêtes et le corps des messages.

Un message est structuré comme suit, dont chaque figure géométrique colorée (gris, vert, orange) représente un composant du message :

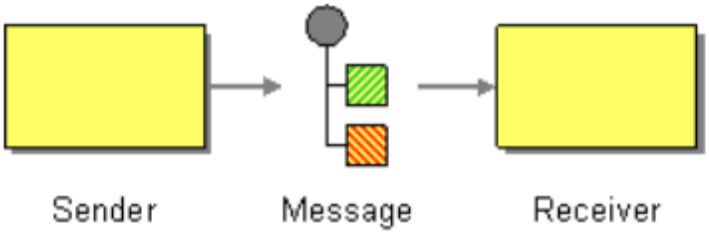


Figure 21: Représentation d'un message XX

c. Les « Channels »

Un Channel est un canal logique où transitent tous les messages envoyés depuis une application vers une autre. Lorsqu'une application a des informations à envoyer, elle ne les diffuse pas au hasard dans le système de messagerie. Elle les envoie dans un canal particulier. Et l'application réceptrice récupèrera les messages dans ce canal. Ce sont les « Endpoints » qui s'occupent de dispatcher les messages dans tel ou tel canal. Un canal est représenté comme suit :

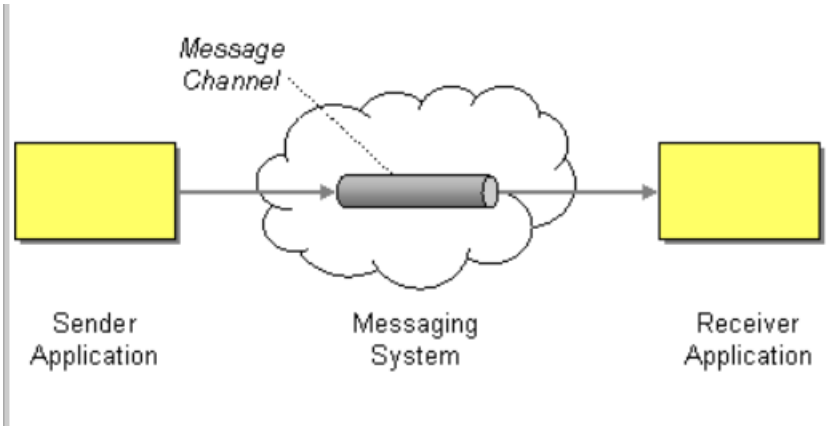


Figure 22 : Représentation d'un canal

d. Les « Routers »

Un « Router » est un type de filtre qui récupère des messages provenant d'un « Endpoint » particulier dans le canal et les redirige vers le « Endpoint » cible approprié. Un « Router » ne modifie pas le contenu des messages.

Les « Routers » permettent également de gérer les problèmes liés aux routages en offrant des composants de routage ainsi que des composants de filtrage, de découpage, d'agrégation ou de séquencement. Un « Router » fonctionne comme décrit dans l'image suivante :

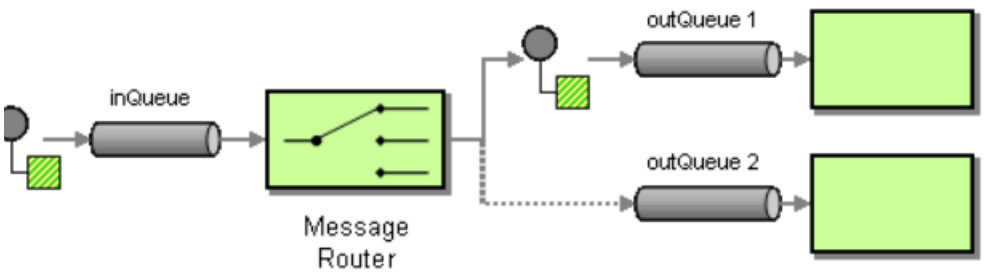


Figure 23 : Représentation d'un routeur

e. Les « Translators »

Les « Translators », comme leurs noms l'indiquent, décrivent un composant qui modifie le contenu d'un message et le traduit dans un autre format. Ils gèrent donc tout problème relié à la transformation et à l'encapsulation. Les formats des messages transformés peuvent être utilisés dans un contexte différent. L'image suivante illustre bien cette transformation :

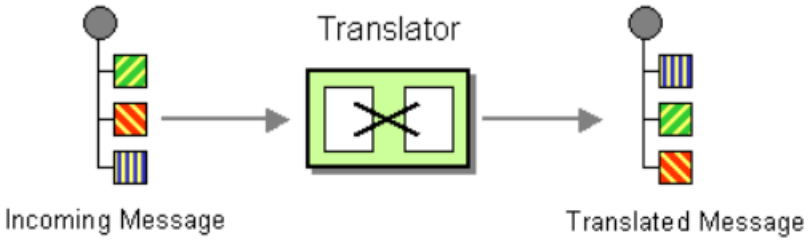


Figure 24: Représentation d'un transformateur

Les différents composants cités ci-dessus sont ici décrits de manière générale, mais ce sont les différentes implémentations existantes qui vont définir plus précisément sa composition et son fonctionnement.

C. Solution d'implémentation actuelle : Camel Apache

Pour la mise en place des échanges, le projet a choisi l'utilisation du Framework APACHE CAMEL qui prend en charge la plupart des EIP.

I. Présentation Fonctionnelle

1. Définitions

Apache Camel est un logiciel qui met en œuvre des échanges de messages entre différentes applications informatiques. Il prend en compte un grand nombre de protocoles et permet de définir ses propres règles de routage, de décider à partir de quelles sources les messages doivent être acceptés et déterminer comment traiter et envoyer ces messages aux autres destinataires. Et tout cela grâce à son moteur de routage constructeur. Camel utilise un langage d'intégration qui permet de définir des règles de routages complexes semblables aux processus métier. L'un des principes fondamentaux de Camel est qu'il ne fait aucune hypothèse sur le type de données à traiter. Cela donne au développeur une opportunité d'intégrer n'importe quel type de système sans avoir à convertir les données.
Camel propose également des abstractions de plus haut niveau permettant d'interagir avec différents systèmes en utilisant la même API quel que soit le protocole ou le type de données utilisé par les systèmes.
Son architecture modulaire et extensible permet de mettre en œuvre et brancher en toute transparence le support des protocoles. Ces choix architecturaux éliminent le besoin de conversions inutiles et rendent Camel non seulement plus rapide mais aussi très maigre.

Il est important de préciser que Camel n'est pas un ESB (Enterprise Service Bus) car il n'offre pas de conteneur ou de système de bus de messagerie. Néanmoins, il est souvent appelé ESB léger en raison de sa prise en charge du routage, transformation, surveillance ou encore orchestration. Camel est plutôt un « Framework » d'Intégration.

2. Utilisations

Camel introduit quelques idées novatrices dans l'espace d'intégration d'où sa création à la place de l'utilisation des « Framework » déjà existants. Il offre plusieurs fonctionnalités notamment :

- Moteur de routage et de médiation :

  C'est la principale caractéristique de Camel. Ce moteur déplacera de manière sélective un message en fonction de la configuration du routage. Dans Camel, les routes sont configurées avec une combinaison de modèles d'intégration d'entreprise et un langage spécifique au domaine.

- Modèles d'intégration de l'entreprise :

  Camel est fortement basé sur les EIP ou Enterprise Integration Patterns. Bien que ces derniers décrivent un problème d'intégration et solutions et aussi fournissent un vocabulaire commun, ce vocabulaire n'est pas formalisé. Camel tente résoudre ce problème en fournissant un langage décrivant les solutions d'intégration.

- Langage spécifique au domaine (DSL):

  Camel DSL offre une contribution majeure à l'espace d'intégration. Contrairement à Camel, les DSL compris actuellement dans quelques Framework d'intégration sont basées sur des langages personnalisés. Ce qui fait la particularité de Camel c'est sa capacité à offrir plusieurs DSL dans la programmation régulière des langages tels que Java, Scala, Groovy et permet même la spécification des routes en XML. Le but du DSL est de permettre au développeur sur le problème d'intégration plutôt que sur le langage de programmation. Camel est principalement écrit en java mais il supporte un mélange de plusieurs langages. Cela permet de construire une solution avec le moins de contraintes possible. Exemple de DSL utilisant différentes langues et restant fonctionnellement équivalent :

```
Java DSL :
                from("file:data/inbox").to("jms:queue:order");

Specification XML :
                <route>
                    <from uri="file:data/inbox"/>
                    <to uri="jms:queue:order"/>
                </route>

Scala DSL :
                from "file:data/inbox" -> "jms:queue:order"
```

Dans le projet Infrastructure d'échange actuel, le format utilisé est la spécification XML

- Bibliothèques de composants étendus :

Camel fournit une vaste bibliothèque de plus de 80 composants. Ces composants permettent à Camel de se connecter grâce des transports, d'utiliser des API et de comprendre les formats de données.

- Routeur de « «Payload-agnostic »

Camel peut acheminer n'importe quel type de « payload », et n'est pas limité au XML. Cette liberté signifie que le « payload » n'a pas besoin de transformation en format canonique.

- Modèle POJO :

Beans ou POJO (Plain Old Java Object) sont considérés comme des composants de premières classes dans Camel et il s'efforce d'autoriser l'utilisation des beans n'importe où et n'importe quand dans les projets d'intégration. Les fonctionnalités intégrées peuvent être étendues.

- Configuration facile :

Le paradigme de la convention sur la configuration est suivi autant que possible, ce qui minimise les exigences de configurations. Afin de configurer les points de terminaison directement dans les routes, Camel utilise une configuration URI simple et intuitive.

- Convertisseur de type automatique :

Camel dispose d'un mécanisme de conversion de type intégré qui est livré avec plus de 150 convertisseurs. Par exemple, les configurations des règles de convertisseur de type pour passer des tableaux d'octets aux chaines de caractères ne sont plus nécessaires. Et s'il y a besoin de convertir en types que Camel ne supporte pas, il est possible de créer son propre convertisseur.

- Noyau léger :

Le noyau de Camel peut être considéré comme assez léger, avec une bibliothèque totale d'environ 1,6 Mo et ayant seulement une dépendance sur Apache Commons Logging et Fuse-Source Commons Management. Cela rend Camel facile à intégrer ou à déployer n'importe où, comme dans une application Spring, Java, conteneur JBI ou l'application Google moteur. Camel n'a pas été conçu pour être un ESB mais plutôt à être intégré dans n'importe quelle plateforme choisie.

- Kit de test :

Camel fournit un kit de test qui permet de tester plus facilement les propres applications Camel. Le même kit de test est largement utilisé pour tester Camel lui-même, et il comprend plus de 6000 tests unitaires. Le kit contient des composants spécifiques au test qui, par exemple, peut aider à tester de réels « endpoints ».

II. Présentation Technique

La fonction primaire de Camel est le routage de « Message » d'une application vers une autre. Pour modéliser ces messages, il existe deux abstractions à considérer dans Apache Camel : les Messages et les Echanges.

A. Définition et architecture d'un Message

Un message est l'entité fondamentale contenant les données qui sont routées et transférées. Ce sont également les entités véhiculées par le système pour faire communiquer ces systèmes entre eux.

Dans l'implémentation de Camel Apache, les messages sont constitués de :

- Un body (payload) : Le corps est de type java.lang.Object. Cela signifie qu'un message peut stocker n'importe quel type de contenu. Cela signifie également que c'est au concepteur de l'application de s'assurer que le destinataire peut comprendre le contenu du message. Lorsque l'expéditeur et le destinataire utilisent différents formats de corps, Camel fournit un certain nombre de mécanismes pour transformer les données dans un format acceptable, et dans de nombreux cas, la conversion se produit automatiquement avec des convertisseurs de type.

- Des entêtes (headers) et les attachements : Ce sont des valeurs associées au message, telles que l'expéditeur identifiants, astuces sur le codage de contenu, informations d'authentification. Les en-têtes sont des paires « nom-valeur »; le nom est une chaine unique, insensible à la casse, et la valeur est de type java.lang.Object. Camel n'impose aucune contrainte sur le type d'en-têtes.

L'image suivante illustre cette description :

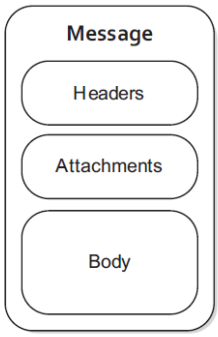


Figure 25 : Architecture d'un message Camel

Les messages sont identifiés par un identifiant unique de type String. Ils circulent dans une direction d'un expéditeur à un récepteur comme illustré ci-dessous :

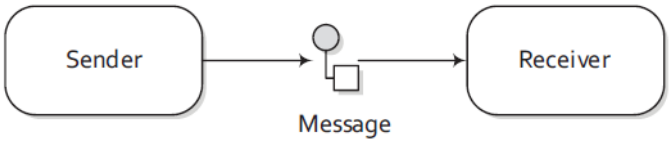


Figure 26 : Architecture d'un envoi de message

La notion d'unicité de l'identifiant et assuré par l'appelant, il dépend du protocole et il n'a pas de format garanti. Pour les protocoles qui n'ont pas d'identification unique, Camel utilise son générateur d'UID[23]

B. Définitions et architecture d'un Echange

Un échange est un conteneur de messages pendant le routage. Un échange supporte un nombre varié d'interactions entre les systèmes (MEP – Message Exchange Pattern). Les MEP sont utilisés pour différencier les messages one-way (asynchrones) des messages two-way (requêtes-réponses ou synchrones).
Un échange est structuré comme suit :

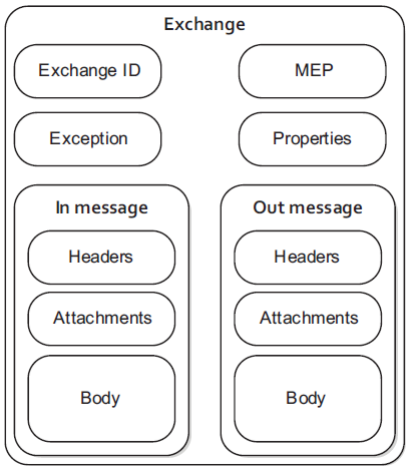


Figure 27 : Architecture d'un échange

- Exchange ID : C'est l'identifiant unique qui est souvent géré directement par Camel.

- MEP ou Message Exchange Patterns : C'est un modèle qui indique si le message est InOnly (one-way) ou InOut (two-way). Quand le message est InOnly, l'échange ne contient qu'un seul message entrant. Mais dans le cas d'un InOut, il y existe un message sortant conçu pour le message de réponse à l'appelant.

- Exceptions : Si une erreur se produit à tout moment pendant le routage, une exception sera définit dans ce champ.

- Properties : Elles sont similaires aux en-têtes de message, mais durent pendant la durée de l'échange. Elles sont utilisées pour contenir des informations de niveau global, tandis que les en-têtes de message sont spécifiques à un message particulier.

[23] Identifiant Unique

Le développeur peut stocker et récupérer des propriétés à tout moment pendant la durée de vie d'un échange.

- In message : Il s'agit du message d'entrée obligatoire. Il contient le message de requête.

- Out Message : Il s'agit d'un message facultatif qui n'existe que si le MEP est InOut. Le message out contient le message de réponse.

C. Architecture Globale de Camel Apache

Comme expliqué dans la section précédente, Camel utilise un langage DSL (Routing Domain Specific Language) basé sur Java ou une configuration XML pour configurer les règles de routage et de médiation qui sont ajoutées à CamelContext pour implémenter différents modèles d'intégration d'entreprise. CamelContext est une interface utilisée pour représenter le contexte utilisé pour configurer les routes et les stratégies à utiliser lors des échanges de messages entre les systèmes d'extremité.
Sur un plus haut niveau, Camel se compose d'un CamelContext qui contient une collection d'instances et de composants. Un composant est essentiellement une fabrique d'instances de « Endpoint ». Les instances de composant peuvent être configurées en code Java ou en conteneur loC comme Spring ou Guice.
Un « Endpoint » agit plutôt comme un URI (Uniform Resource Identifier) ou une URL (Uniform Resource Locator) dans une application web ou une destination dans un système JMS. La communication peut se faire par un « Endpoint », soit en lui envoyant des messages, soit en consommant des messages. Il est ensuite possible de créer un producteur ou un consommateur sur un « Endpoint » pour échanger avec lui. L'envoi ou la réception de message par « Endpoint » se fait comme suit :

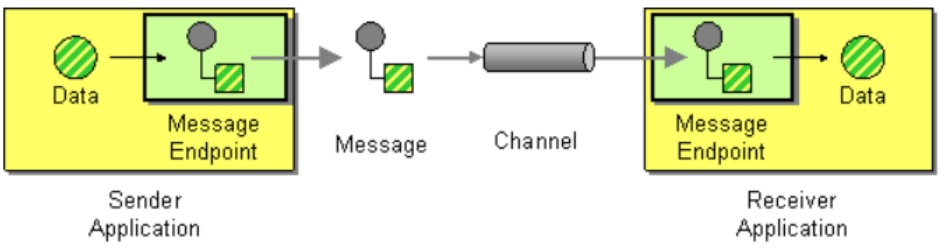


Figure 28 : Endpoint des messages Camel

L'architecture globale de Camel se définit comme suit :

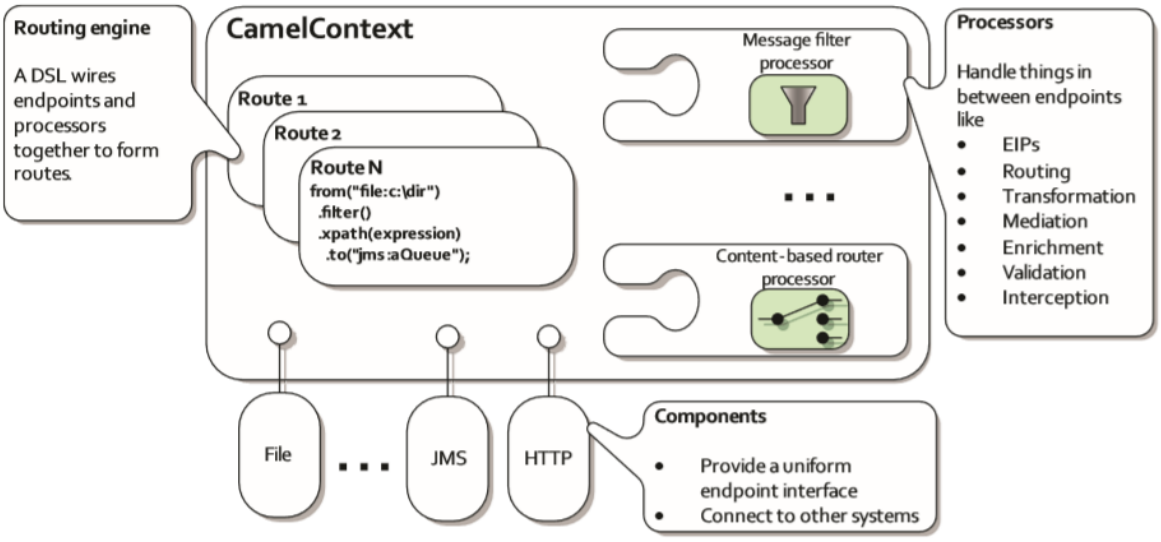


Figure 29: Architecture globale de Camel Apache

La figure précédente révèle de nombreux nouveaux concepts que nous allons examiner en détail pour mieux comprendre.

a. « CamelContext »

Comme affiché dans la figure 29, Camel est un conteneur de plusieurs outils. « CamelContext » peut donc être vu comme un système d'exécution de Camel qui conserve tous les morceaux ensemble. De nombreux services sont gardés ensemble dans « CamelContext » comme le montre l'image suivante :

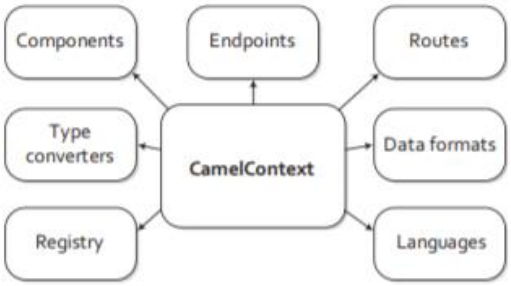


Figure 30 : CamelContext

- « Components » : Ils contiennent les composants utilisés. Camel possède la capacité de pouvoir charger à la volée soit par découverte automatique sur le « classpath », soit lorsqu'un nouvel ensemble est activé dans un conteneur.

- « Endpoints » contiennent les nouveaux « Endpoints » qui ont été nouvellement créés.

- « Routes » : Les nouvelles routes ajoutées sont mises dans ce conteneur.

- « Type converters » : Ils contiennent les convertisseurs de type chargés dans Camel. Ce dernier a un mécanisme qui permet manuellement ou automatiquement de convertir un type dans un autre.

- « Data formats » : Les formats de données chargés sont contenus dans cet ensemble.

- « Registry » : Ils contiennent les registres permettant de chercher des beans. Par défaut, c'est un registre JNDI (Java Naming and Directory Interface). Dans le cas de l'utilisateur de Camel avec Spring, il s'agira ici du Spring ApplicationContext.

- « Languages » : Ils contiennent les langues qui ont été chargées.

b. « Routing engine »

Le moteur de routage de Camel est ce qui fait transiter les messages. Ce moteur n'est pas visible pour le développeur, mais il faut savoir qu'il est présent et c'est lui qui fait le plus gros travail en s'assurant du bon acheminement des messages.

c. « Routes »

Les routes constituent une abstraction essentielle pour Camel. Le moyen le plus simple de le définir est de le considérer comme une chaîne de processeur. Il existe plusieurs raison à l'utilisation des routeurs dans un système de messagerie. En dissociant les clients des serveurs, ainsi que les producteurs des consommateurs, les routes peuvent :

- Décider dynamiquement quel serveur un client évoquera.
- Fournir un moyen flexible d'ajouter du traitement supplémentaire.
- Permettre aux clients et aux serveurs d'être développés indépendamment.
- Encourager les meilleures pratiques de conceptions en connectant des systèmes disparates qui en font une bonne chose.
- Améliorer les fonctionnalités de certains systèmes.

Chaque route de Camel possède un identifiant unique. Les routes ont exactement une source d'entrée pour les messages et sont donc liés à un « Endpoint » d'entrée. Un DSL (Domain Specific Language) est utilisé pour définir les routes.

d. « Domain Specific Language »

Pour connecter les processeurs, que nous verrons un peu plus bas, et les « Endpoints » afin de créer des routes, Camel utilise des DSL. Ce terme est utilisé assez vaguement car dans Camel, il représente une API Java fluide qui contient des méthodes. Les DSL fournissent une abstraction intéressante aux utilisateurs de Camel pour la création d'applications.

Mais au fond, une route est en réalité composée d'un graphe de processeurs.

e. « Processor »

C'est un concept de base de Camel qui représente un nœud capable d'utiliser, de créer ou de modifier un échange entrant. Pendant le routage, les échanges vont d'un processeur à un autre. De ce fait, on peut considérer une route comme un graphe dont les nœuds sont les processeurs, et les lignes qui connectent la sortie d'un processeur à l'entrée d'un autre. La plupart des processeurs sont des implémentations des EIP (Enterprise Integration Patterns) mais il est possible d'implémenter son propre processeur et de l'insérer dans une route.

f. « Component »

Les composants sont le point d'extension principal de Camel. A ce jour, il existe plus de 80 composants dans l'ecosystème Camel qui vont en fonction des transports de données aux DSL, formats de données, etc. Il est même possible de créer ses propres composants pour Camel.
Du point de vue de la programmation, ils sont assez simples : ils sont associés avec un nom qui est utilisé dans un URI (Uniform Source Identifier), et ils agissent comme une usine de « Endpoints ».
Les « Endpoints » constituent un concept encore plus fondamental dans Camel.

g. « Endpoints »

Un « Endpoint » est l'abstraction de Camel qui modélise la fin d'un canal par lequel un système peut envoyer ou recevoir des messages.

III. Avantages et limites de Camel Apache

1. Avantages de Camel Apache

- **Forte documentation :**
  En plus du guide utilisateur présentant une vue générale du produit, on trouve un cookbook, des rubriques spécifiques à chaque pattern EIP implémenté par le framework, le détail des composants avec pour chacun d'entre eux, quelques cas d'utilisation courants, la présentation de l'architecture du produit, des exemples commentés, le détail des formats de données utilisés, la présentation des différents langages d'expression disponibles
- **Les patterns :**
  EIP dans le domaine de l'intégration, les patterns d'entreprises représentent aujourd'hui les bases du langage communément employé. Il est donc indispensable de savoir comment les Frameworks intègrent cette terminologie. Camel propose pas moins de 40 EIP prêts à l'emploi.
- **La connectivité :**
  Camel Apache embarque actuellement plus de 90 composants de connectivités.

- **La possibilité de rejouer le message** :
  Le framework permet de définir un certain nombre de réémission d'un message suite à la remontée d'une exception, quel que soit le composant à l'origine de l'erreur. Ce nombre d'essai peut être spécifié directement dans les paramètres des composants ou indirectement via l'implémentation d'une stratégie de rejeu.
- **Testabilité :**
  Une fois les flux définis, il convient de pouvoir les tester, par morceaux ou de bout en bout, de manière simple. Camel propose des mécanismes de templates ou de mocks appropriés aux problématiques d'échanges.

2. Limites de Camel Apache

- Camel Apache est un outil léger d'intégration. De ce fait, il ne faut pas s'attendre aux grandes performances qui sont fournies par les ESB[24] ou les WebMethods. Par exemple, une demande de transfert de 7000 fichiers à la minute pourrait présenter une difficulté pour le logiciel et pourrait même le bloquer.

- Même si Camel est assez facile à configurer et à mettre en place, sa prise en main n'est pas toujours évidente. Le code est écrit en Java et une bonne connaissance en java est donc requise. Le code peut être également écrit en XML mais il est plus lisible en java.

- Camel ne possède pas de support commercial de la part des auteurs originaux, mais plutôt des sociétés tierces. Il ne dispose pas de framework appropriés basé sur l'interface utilisateur, ce qui le rend difficile à utiliser pour des personnes n'ayant pas fait de la programmation.

- Bien que fortement documenté, la documentation de Camel Apache est loin d'être aisée à lire et les informations retrouvées dans le wiki ne sont pas facile à comprendre du premier coup.

Pour la plateforme d'échange, il existe d'autres solutions alternatives plus récentes que l'on pourrait utiliser à la place de Camel Apache telle que Spring Integration.

D. Solution d'implémentation alternative : Spring Integration

I. Présentation fonctionnelle

1. Définition

Spring Integration est un logiciel open source qui étend le modèle de programmation Spring pour prendre en charge les EIP bien connus. Il offre un système de messagerie légère dans les applications basées sur Spring et intègre les systèmes externes par le biais des adaptateurs déclaratifs. Ces adaptateurs fournissent un niveau d'abstraction supérieur au

[24] Enterprise Service Bus

support Spring pour la communication à distance, la messagerie et la planification. Spring Integration s'appuie sur les mêmes objectifs et principes que Spring Framework utilisé actuellement dans de nombreuses applications. L'objectif premier de Spring Integration est donc est de pouvoir traiter les responsabilités pour le compte des composants gérés dans son contexte. De ce fait, les composants eux-mêmes sont simplifiés car ils sont déchargés de ces responsabilités.

Pour y voir plus clair, prenons l'exemple de l'injection de dépendances de Spring Framework. Le fait que la structure de Spring gère elle-même la création de dépendances soulage les composants de la responsabilité de localiser ou de créer leurs propres dépendances.

Un autre exemple que l'on peut présenter est la programmation orientée aspect. Elle soulage les composants métiers des tâches transversales génériques en les réintégrant dans des aspects réutilisables. Le résultat est donc un système plus facile à tester, à comprendre, à maintenir et à étendre.

En tant qu'implémentation d'un EIP, Spring Integration étend le framework Spring au domaine de la messagerie. Il prend en charge les architectures pilotées par message où l'inversion de contrôle s'applique aux problèmes d'exécution. Spring Integration prend également en charge le routage et la transformation des messages afin que différents transports et différents formats de données puissent être intégrés.

En tant qu'extension de Spring framework, Spring Integration fournit une grande variété d'options de configuration, y compris des annotations, XML avec prise en charge d'espace de noms, XML avec des éléments « bean » génériques et bien sûr l'utilisation directe de l'API sous-jacente.

2. Objectifs et principes

Spring Integration a été conçu pour simplifier les solutions d'intégration d'entreprise et à faciliter la prise en main par les utilisateurs de Spring framework des principes EDA (Event Driven Architecture). Pour ce faire, Spring Integration a mis en place quelques objectifs qui sont :

- Pouvoir fournir un modèle simple pour la mise en œuvre des solutions d'intégrations d'entreprises complexes.
- Faciliter le comportement asynchrone et basé sur les messages au sein d'une application Spring.
- Promouvoir la prise en main intuitive et incrémentielle pour les utilisateurs Spring déjà existants.

D'autres parts, Spring Integration repose sur certains principes qui sont :

- Les composants doivent être faiblement dépendants entre eux pour assurer la modularité et la testabilité, c'est ce que l'on appelle le couplage lâche.
- La séparation des préoccupations (separation of concerns) entre la logique métier et la logique d'intégration doit être respectée.
- Abstraction et définition claire des points d'extension afin de promouvoir la réutilisabilité et portabilité.

II. Présentation technique

1. Architecture globale de Spring Integration

Dans une vue globale, Spring Integration se compose de deux parties. A la base, c'est un framework de messagerie qui prend en charge des interactions légères et évènementielles au sein d'une application. En plus de ce noyau, il fournit une plateforme basée sur un adaptateur qui prend en charge une intégration flexible d'applications dans l'entreprise.

Les deux parties décrites ci-dessus sont représentées dans figure qui suit :

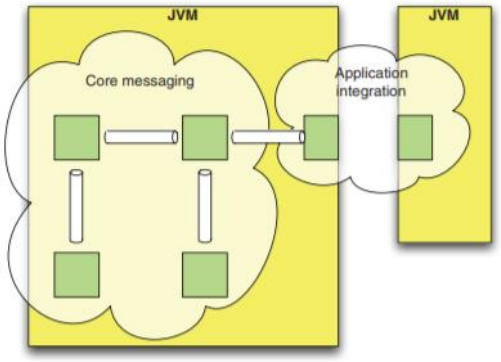


Figure 31: Architecture globale de Spring Integration

Tout ce qui est écrit dans la zone de messagerie principale de la figure existe au sein de la portée d'un seul contexte d'une application Spring. Ces composants font les échanges de messages de manière légère car ils fonctionnent dans la même JVM (Java Virtual Machine). Il n'est pas nécessaire de s'inquiéter de la sérialisation, et sauf si nécessaire pour un composant particulier, le contenu du message n'a pas besoin d'être représenté en XML. Au lieu de cela, la plupart des messages contiendront des POJO (Plain Old Java Object) instanciés comme leurs « payloads ».

La zone d'intégration est différente. Les adaptateurs sont utilisés pour planifier le contenu des messages sortants dans le format attendu par certains systèmes externes pour recevoir et mapper le contenu de ces systèmes externes en messages. La manière dont la planification est implémentée dépend de l'adaptateur particulier, mais Spring Integration fournit un modèle cohérent facile à étendre.

Spring Integration actuellement prend en charge les protocoles suivants :

- File Transfer Protocol (FTP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- HyperText Transfert Protocol sur le protocole REST (HTTP)
- Web Services (SOAP)
- Envoi de mail (POP3, IMAP, SMTP)

- Java Message Service (JMS)
- Java Database Connectivity (JDBC)
- Remote Method Invocation (RMI)
- Java Management Extensions (JMX)

La plupart des protocoles proposés ci-dessus et les transports répertoriés peuvent être utilisés aussi bien pour la partie entrante que pour la partie sortante. Dans Spring Integration, le pattern « Channel Adapter » s'applique à tout protocole entrant ou sortant unidirectionnel. En d'autres termes, un « Channel Adapter » entrant n'accepte qu'un message uniquement entrant et un « Channel Adapter » sortant n'accepte qu'un message uniquement sortant. Tout echange bi-directionnelle ou demande-réponse est assuré par ce que l'on appelle « Gateway » dans Spring Integration.

2. Composants de Spring Integration

Le modèle EIP décrit les modèles dans la messagerie. En tant qu'implémentation du modèle EIP, Spring Integration repose sur les mêmes bases. Il est donc constitué de trois principaux modèles de bases : les messages, les canaux ainsi que les « Endpoints » comme décrit dans le schéma ci-après :

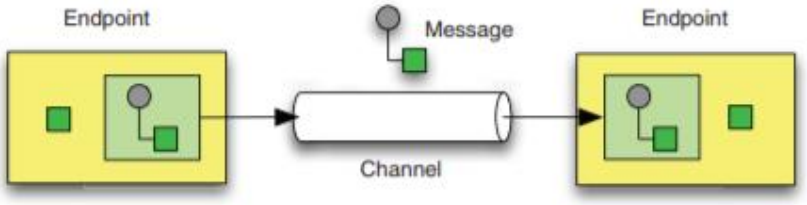


Figure 32: Composants principaux de Spring Integration

a. Les messages

Un message est une unité d'information que l'on peut transmettre entre les différents composants que l'on a déjà eu l'occasion de voir qui sont les « Endpoints ».

Les messages sont généralement envoyés après qu'un « Endpoint » ait fait une part du travail, et ils déclenchent un autre « Endpoint » pour effectuer l'autre part du travail. N'importe quel format de message est accepté à partir du moment où ce format convient au « Endpoint » d'envoi et à celui de réception.

Par exemple, le « payload » du message peut être en XML, une simple chaîne de caractère ou une clé primaire qui référencie une donnée dans une base de données.

Chaque message est composé d'un entête et d'un « payload » comme illustré ci-dessus :

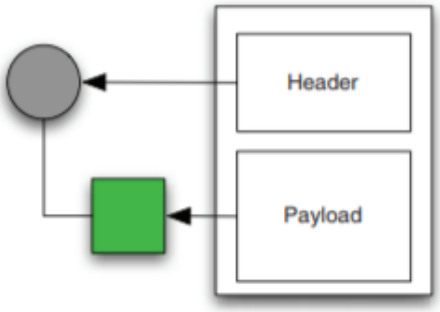


Figure 33: Composants d'un message dans Spring Integration

Comme dans de nombreuses implémentations d'EIP, l'entête contient des données pertinentes pour le système de messagerie tel que l'adresse de retour ou l'identifiant de corrélation.

Le « payload » contient la donnée principale à consulter ou à traiter par le destinataire. Les messages peuvent avoir différentes fonctions comme un message de commande qui indique au destinataire de faire quelque chose ou un d'évènement qui avertit le destinataire que quelque chose est en train de se passer , ou encore un message de document qui transfère simplement les données de l'expéditeur vers le récepteur.

Un message peut être présenté comme un contrat entre l'expéditeur et le destinataire.

b. Les « Channels »

Les « Channels » ou canaux servent de connexions entre deux « Endpoints ». Un canal gère les détails de comment et où les messages vont être envoyés. Ils n'interfèrent en aucun cas avec le contenu du message. Tout canal dissocie logiquement les producteurs et les consommateurs. Il existe un certain nombre d'options d'implémentations pratiques.

Par exemple, un canal particulier peut envoyer des messages directement aux consommateurs passifs dans le même fil de contrôle. D'autre part, une implémentation de canal différente peut faire en sorte de mettre en place un mémoire tampon qui vont stockés les messages dans une file d'attente dont la référence est partagée par l'expéditeur et un consommateur actif telles que les opérations d'envoi et réception se produisent chacune dans différents fils de contrôle. De plus, les canaux peuvent être classés suivant le fait que les messages soient livrés directement à un « Endpoint » (point-to-point) ou si il est écouté par un « Endpoint » de sortie (publish-suscribe). Seuls deux « Endpoint » connectés par un canal peuvent s'envoyer des messages. Les « Channels Messages » sont les facteurs du couplage lâche qui est un des objectifs principaux de Spring Integration.

Dans la messagerie « point-to-point », chaque message envoyé par un producteur est reçu exactement par un consommateur. Si aucun consommateur ne reçoit le message, alors il est considéré comme une erreur. C'est l'équivalent du principe de « Queue » dans Camel Apache.

L'image suivante illustre ce type de canal :

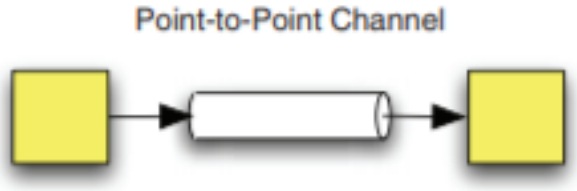


Figure 34: Figure 35: Canal point-to-point

Contrairement à la messagerie « point-to-point », un canal « publish-suscribe » offre le même message à zéro ou plusieurs abonnés. Il offre un gain de flexibilité car les consommateurs peuvent syntoniser[25] au canal à l'exécution. Cependant, dans ce type de messagerie, l'expéditeur n'est pas informé de la livraison du message ni dans le cas de l'échec de transmission. Ce type d'acheminement nécessite souvent un schéma de gestion des pannes. Ce type de canal équivaut au principe de « Topics » dans Apache Camel.

Le canal « publish-suscribe » est représenté comme suit :

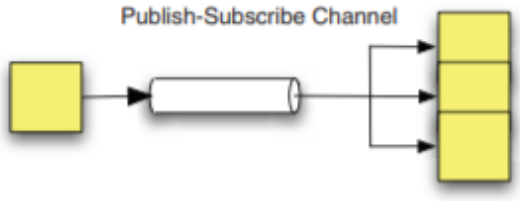


Figure 36: Canal Publish-Suscribe

c. Les « Messages Endpoints »

Dans le cas de Spring Integration, les « Endpoints » sont les composants qui traitent réellement les messages. Qu'il s'agisse d'un simple routage vers un autre canal, ou d'un fractionnement de message en plusieurs parties ou encore un regroupement des parties, ce sont les « Endpoints » qui interviennent. Ces derniers fournissent essentiellement les connexions entre les services fonctionnels et le framework de messagerie. Du point de vue du framework, les « Endpoints » se trouvent à l'extrémité des canaux. Cela veut tout simplement dire qu'un message peut quitter le canal avec succès seulement s'il a été consommé par un « Endpoint ». Et un message ne peut entrer dans un canal que s'il a été produit par un « Endpoint ».

Il existe plusieurs types d' « Endpoints » dans Spring Integration :

- « Channel Adapter » : Il connecte une application au système de messagerie. Dans Spring Integration, un flux de message unidirectionnel commence et se termine par un adaptateur de canal comme illustré ci-dessous :

[25] Syntoniser : Ajuster deux circuits à la même fréquence. Wikipedia

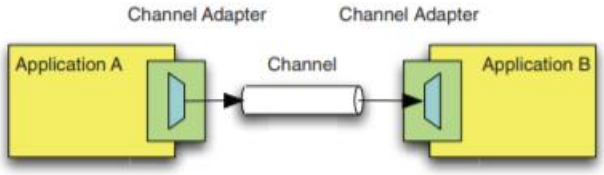


Figure 37: Channel adapter

- « Messaging Gateway » : C'est une connexion spécifique à la messagerie bidirectionnelle. Si une demande entrante doit être traitée par plusieurs files, mais que l'envoyeur ne doit pas s'en rendre compte, la solution est de passer par une passerelle. Du côté de la partie sortante, un message qui arrive peut être utilisé dans un appel synchrone et le résultat est envoyé sur un canal de réponse. Par exemple lors des appels de web services, ou pour la synchronisation des questions-réponses dans le cadre d'un JMS.

  L'image suivante illustre l'architecture d'une passerelle :

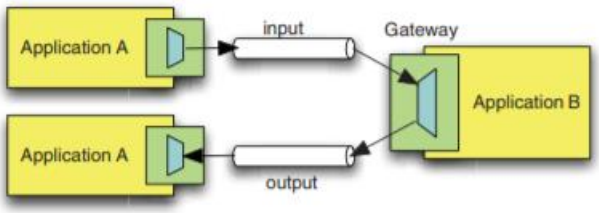


Figure 38 : Messaging Gateway

Une passerelle peut également être utilisée en cours de route dans un flux de message unidirectionnel comme le montre le schéma suivant :

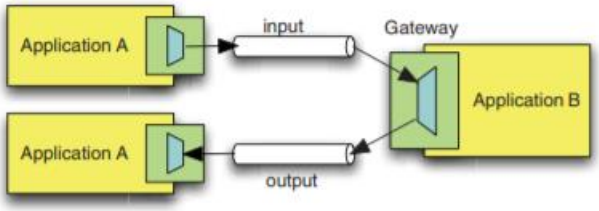


Figure 39 : Messaging Gateway et Channel Adapter

- « Service Activator » : Ce composant appelle un service basé sur un message entrant et envoie un message sortant qui se base sur la valeur de retour du précédent appel. Dans Spring Integration, cet appel est limité à un appel de méthode locale de façon à ce que l'utilisateur considère l'activateur de service comme à une passerelle d'appel de méthode sortante. La méthode

appelée est définie sur un objet référencé dans le même contexte de Spring Application.
L'activateur de service est représenté comme suit :

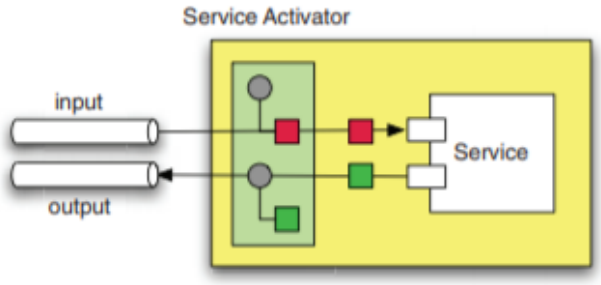


Figure 40 : Service Activator

- « Router » : C'est le routeur qui détermine le prochain canal où sera envoyé le message basé sur le message entrant. Il peut être utile pour envoyer des messages avec différents « payloads » vers différents consommateurs spéciaux. Ce type de routeur est un routeur basé sur le contenu. Le routeur ne touche pas au contenu des messages et connaît les canaux par lesquels vont passer les messages. Par conséquent, c'est l' « Endpoint » le plus proche de l'infrastructure et le plus éloigné des préoccupations de l'entreprise.
  Ci-dessous est schématisé un routeur dans Spring Integration :

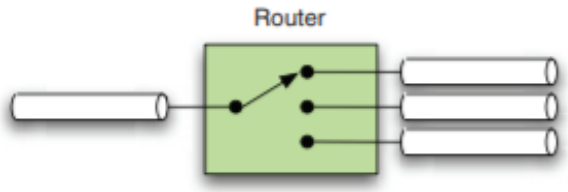


Figure 41 : Router

- « Splitter » : Un séparateur reçoit un message et le divise en plusieurs parties avant de l'envoyer au canal de sortie. Ce type d'action est utile chaque fois que le contenu du message traité peut être divisé en plusieurs étapes et exécuté par différents consommateurs en même temps.
  Le fonctionnement d'un séparateur est illustré dans l'image suivante :

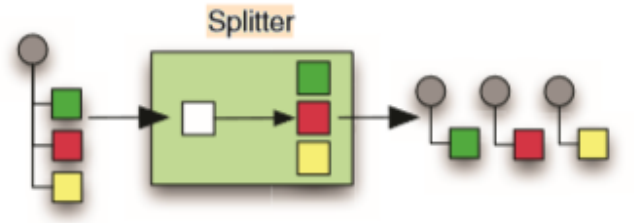


Figure 42 : Splitter

- « Aggregator » : Contrairement à un séparateur, un agrégateur attend un groupe de messages corrélés et les fusionne quand le groupe est complet. La corrélation est généralement basé sur un identifiant de corrélation et l'achèvement est lié à la taille du groupe. Un agrégateur et un séparateur sont souvent utilisés dans une configuration symétrique, ou une partie du travail d'un agrégateur est effectué en parallèle après un séparateur et le résultat agrégé est envoyé au prochain canal.
  L'image suivante montre que l'agrégateur effectue le travail inverse du séparateur :

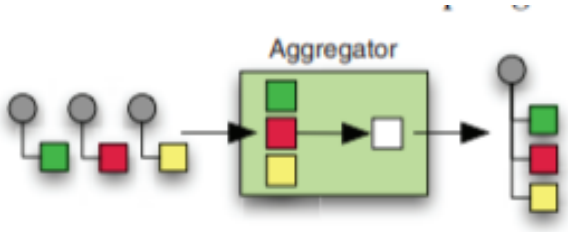


Figure 43 : Aggregator

III. Avantages et Limites de Spring Integration

Spring Integration a été conçu pour étendre sur Spring Framework. Il reprend les mêmes principes que ce dernier et aussi les mêmes objectifs, ce qui le rend plus facile à prendre en main pour les développeurs déjà initiés à Spring Framework. Cependant cette dépendance possède une certaine contrepartie. Le logiciel aura les mêmes avantages d'utilisation et les mêmes inconvénients que Spring Framework au niveau technique.

1. Avantages de Spring Integration

- La testabilité est l'un des critères les plus importants dans le choix d'un outil de développement. Or, l'injection de dépendance offerte par Spring

augmente la testabilité du logiciel car il simplifie l'injection de données avec l'utilisation de JavaBean POJO.

- Le développement avec Spring est basé sur les POJO. L'utilisation de ce dernier entraîne l'inutilité d'utiliser un serveur d'applications. On privilégiera les conteneurs de servlet tel que Tomcat.

- La configuration en est flexible. Spring prend en charge l'utilisation de la configuration XML ainsi que des annotations basées sur Java pour la configuration des Beans. Par conséquent, il offre la possibilité d'utiliser n'importe laquelle de ces configurations pour le développement.

- Le module Spring AOP (Aspect Oriented Programming) que Spring propose représente plusieurs avantages pour le développeur. Il permet d'avoir une unité de compilation différente ou un chargeur de classes séparé.

- L'un des principaux avantages de Spring est qu'il est inutile de réinventer la roue. Il utilise des technologies connues telles que les timers JDK (Java Development Kit), ou les frameworks ORM, Java EE, etc.

- Spring offre une modularité aux développeurs. Cela aide à choisir quels paquets ou quelles classes peuvent être utilisés ou ignorés. Même avec un grand de paquets ou de classes, les développeurs ont la possibilité d'identifier et de choisir ceux qu'il veut utiliser sans problème.

- Spring peut facilement augmenter ou réduire les transactions locales ou globales à l'aide de JTA (Java Transaction Annotation). Cela est dû à une interface de gestion des transactions cohérentes.

2. Limites de Spring Integration

Spring est un très bon outil de développement. Mais comme toutes technologies, il possède aussi ses faiblesses.

- L'inconvénient majeur de Spring et sa grande complexité au niveau de sa structure et de son manque de précision. En effet, il contient plus de 2400 classes et 49 autres outils qui compliquent sa prise en main par les développeurs.

- Spring travaille beaucoup avec du XML. Une connaissance dans ce domaine est donc pré-requise pour apprendre le framework.

- La documentation de Spring est aujourd'hui encore un peu floue. Elle ne contient pas de directives claires sur plusieurs sujets pour les développeurs.

- Bien prendre en main Spring Integration nécessite une bonne compréhension de Java. Il pourrait présenter une difficulté pour ceux qui travaillent avec d'autres langages de programmation.

E. Etude comparative entre Camel Apache et Spring Integration

Certaines personnes préfèreront utiliser Camel Apache alors que d'autres se plongeront plus dans Spring Integration. Bien que ces deux frameworks possèdent des similitudes et permettent de fournir le même travail, ils ont aussi leurs différences qui feront pencher une partie des développeurs vers l'un plutôt que vers l'autre.

1. Configuration

Apache Camel	Spring Integration
- Fichier de configuration Spring (XML) - DSL interne (pour Java DSL) - DSL externe (Pour Scala DSL)	- Fichier de configuration Spring (XML) - Par annotations

2. Concept

Apache Camel	Spring Integration
Il s'appuie sur les notions de routes qui relient deux « Endpoints ». Les messages transitent sur la route à travers des échanges.	Spring Integration reprend parfaitement le concept des EIPs avec l'utilisation des « Messages », des « Messages Channels », et des « Endpoints » qui regroupent le routage, la transformation et le point d'accès des messages.

3. Documentation

Apache Camel	Spring Integration
Bien que fortement documenté, la documentation offerte par Apache Camel n'est pas facile à appréhender.	Spring Integration possède une très bonne documentation et ses forums sont très actifs. Ce qui facilite la prise en main et l'apprentissage par les utilisateurs.

4. Prise en main

Apache Camel	Spring Integration
Camel Apache peut être ardu à prendre en main pour la première fois. Cependant, il est possible de faire ce dont on a besoin avec une simple description XML et des POJOS. Il est possible de faire les mêmes choses de différentes manières avec Camel Apache.	Spring Integration est très facile à prendre en main à partir du moment où on possède des bases en java et en Spring.

5. Approche

Apache Camel	Spring Integration
Camel est orienté échange ou « top-down »	Spring propose une approche orientée outil ou « bottom-up »

6. La connectivité

Apache Camel	Spring Integration
Apache Camel présente un énorme avantage de ce côté car il embarque plus de 90 composants de connectivité. Et en plus, il est possible de développer ses propres adaptateurs.	Spring s'interconnecte avec les systèmes par le biais des adaptateurs. Il en existe un peu plus de 5 connecteurs par défaut actuellement proposés par Spring. En dehors de ces adaptateurs embarqués directement dans le framework, d'autres sont fournis dans Spring Extension.

7. Les conversions de données

Apache Camel	Spring Integration
Il a la capacité de convertir implicitement les données contenues dans les messages échangés lors du passage d'un composant à l'autre. On retrouve aujourd'hui plus de 100 conversions disponibles par défaut. Camel propose également des opérateurs de transformations permettant de travailler avec de nombreux formats de données. Il est possible de définir explicitement les conversions ou les transformations si besoin.	Spring n'offre pas de conversion implicite de données. Toutes conversions ou transformations doivent se faire explicitement en fournissant une implémentation spécifiquement.

8. La gestion d'erreurs

Apache Camel	Spring Integration
Camel permet de définir facilement des blocs « try-catch » sur une partie de l'échange ou d'utiliser le filtrage d'exceptions pour gérer finement la prise en charge des erreurs.	Le framework propose une implémentation du pattern « dead letter channel » pour gérer ce point.

9. Le rejeu

Apache Camel	Spring Integration
Il propose un certain nombre de réémission de message suite à la remontée d'une exception, quel que soit le composant à l'origine de l'erreur. Le nombre d'essai en cas d'échec peut être spécifié directement dans les paramètres des composants ou indirectement par l'implémentation d'un service de rejeu.	Spring Integration ne répond malheureusement pas à ce besoin.

10.Tests

Apache Camel	Spring Integration
Camel offre des mécanismes de templates appropriés aux problématiques d'échanges.	Spring n'offre pas de solution spécifique sur les tests de bout en bout. Cependant, il est toujours possible de passer par les mécanismes classiques.

Conclusion

De nos jours, l'interopérabilité entre applications ne doit plus constituer un problème majeur dans les entreprises car plusieurs solutions pour y remédier peuvent être proposées et implémentées. Avec toutes les technologies existantes sur le marché, chaque entreprise peut choisir celle qui lui convient et celle adaptée à ces besoins et à son architecture interne.

Dans ce document, j'ai abordé le sujet sur les EIP et le principe de messagerie car c'est la solution que le projet infrastructure d'Echange a choisi pour permettre la communication entre les différentes applications. Pour ne pas sortir du cadre des EIP, j'ai proposé une solution d'implémentation similaire à Camel Apache, qui pourrait répondre aux mêmes attentes qui est basé aussi sur le principe de messagerie. Le but de ce document a été de voir en détails l'architecture technique de Camel Apache et de Spring Integration afin de pouvoir se faire une idée sur leurs similitudes, sur leurs différences mais aussi sur leur fonctionnement. Les technologies choisies auront toujours leurs avantages et leurs inconvénients selon le cas d'utilisation. Cependant, le choix de l'une de ces technologies doit reposer sur les objectifs propres à chaque entreprise.

Le choix de Spring Integration comme solution alternative dans ce cas précis a été guidé par mon expérience sur Spring Framework et aussi un choix personnel en adéquation sur les technologies du projet. En effet, Spring Integration est une extension de Spring Framework qui un framework que l'on utilise déjà sur le projet et qui aurait pu facilement être intégré.

Personnellement, étant passionnée du langage de programmation Java et utilisatrice quotidienne de Spring framework, j'ai une petite préférence pour Spring Integration dont le code est plus propre et les configurations sont proches de celles de Spring framework. Mon expérience sur le projet a également renforcé cette conviction. Comme décrit dans le document, Camel propose également une spécification en XML pour la configuration des routes. Dans ce cas précis, l'utilisation du XML dans le code peut rendre ce dernier peu lisible et non intuitif, et peut rendre la maintenabilité difficile. Nous utilisons actuellement la description XML sur le projet. Cependant, d'un point de vue utilisateur et non développeur, Camel est un très bon outil, assez performant et facile à intégrer grâce à sa structure légère.

De nos jours, d'autres architectures beaucoup plus récentes sont de plus en plus répandues sur le marché pour résoudre les problèmes d'intégration d'entreprise. L'architecture micro-service qui consiste à diviser une application en plusieurs micro-applications qui communiqueront entre elles par le biais des API (Application Programming Interface) est l'un de ces nouveaux concepts. C'est une façon de procéder complètement différente des EIP, mais au fond toutes ces architectures existent pour répondre au même problème qui est de permettre l'interopérabilité entre les différentes applications.

Avant d'arriver sur le projet, je ne connaissais pas les problèmes d'interopérabilités retrouvées en Entreprise. Travailler sur le projet infrastructure d'échange a été pour moi une expérience très enrichissante car j'ai appris de nouvelles technologies que je ne connaissais pas auparavant tel Camel Apache, ou Ansible. J'ai également pu approfondir mes connaissances dans les technologies comme Spring, hibernate ou jenkins. Mais le plus

important pour moi a été d'avoir pris connaissance des problèmes de communications entre les applications et l'importance de pouvoir les faire communiquer entre elles.

En tant que développeuse, j'étais en mesure d'utiliser Camel Apache, mais la rédaction de ce mémoire m'a permis de voir son architecture et son fonctionnement interne et de mieux comprendre ce qui se passe derrière chaque ligne que je configure dans le code. Ce document m'a également permis de voir de près Spring Integration que je trouve très intéressant au niveau de sa structure. Mon manque d'expérience au niveau du développement sur Spring Integration ne me permet pas aujourd'hui de donner un avis personnel sur son utilisation réelle.

Glossaire

API : En informatique, une interface de programmation applicative (souvent désignée par le terme API pour *application programming interface*) est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes *consommateurs* peuvent se servir des fonctionnalités du programme *fournisseur*. Wikipedia

https://fr.wikipedia.org/wiki/Interface_de_programmation

REST : Les services web de type *representational state transfer* (REST) exposent entièrement ces fonctionnalités comme un ensemble de ressources (URI) identifiables et accessibles par la syntaxe et la sémantique du protocole HTTP. Les Services Web de type REST sont donc basés sur l'architecture du web et ses standards de base : HTTP et URI. Wikipedia

https://fr.wikipedia.org/wiki/Representational_state_transfer

Microservice : En informatique, les microservices sont un style d'architecture logicielle à partir duquel un ensemble complexe d'applications est décomposé en plusieurs processus indépendants et faiblement couplés, souvent spécialisés dans une seule tâche. Les processus indépendants communiquent les uns avec les autres en utilisant des API langage-agnostiques. Des API REST sont souvent employées pour relier chaque microservice aux autres. Un avantage avancé est que lors d'un besoin critique en une ressource, seul le microservice lié sera augmenté, contrairement à la totalité de l'application dans une architecture classique, par exemple une architecture trois tiers. Cependant, le coût de mise en place, en raison des compétences requises, est parfois plus élevé. Wikipedia

https://fr.wikipedia.org/wiki/Microservices

POJO : POJO est un acronyme qui signifie plain old Java object que l'on peut traduire en français par *bon vieil objet Java*. Cet acronyme est principalement utilisé pour faire référence à la simplicité d'utilisation d'un objet Java en comparaison avec la lourdeur d'utilisation d'un composant EJB. Ainsi, un POJO n'implémente pas d'interface spécifique à un *framework* comme c'est le cas par exemple pour un composant EJB. Wikipedia

https://fr.wikipedia.org/wiki/Plain_old_Java_object

SOA : L'architecture orientée services (calque de l'anglais service oriented architecture, SOA) est une forme d'architecture de médiation qui est un modèle d'interaction applicative qui met en œuvre des services (composants logiciels) :

- avec une forte cohérence interne (par l'utilisation d'un format d'échange pivot, le plus souvent XML ou JSON) ;
- des couplages externes « lâches » (par l'utilisation d'une couche d'interface interopérable, le plus souvent un service web WS-*). Wikipedia

https://fr.wikipedia.org/wiki/Architecture_orient%C3%A9e_services

Web Service : Un service web (ou service de la toile) est un protocole d'interface informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone. Le protocole de communication est défini dans le cadre de la norme SOAP dans la signature du service exposé (WSDL). Actuellement, le protocole de transport est essentiellement HTTP(S). Wikipedia

https://fr.wikipedia.org/wiki/Service_web

SOAP (ancien acronyme de *Simple Object Access Protocol*) : est un protocole d'échange d'information structurée dans l'implémentation de services web bâti sur XML.

https://fr.wikipedia.org/wiki/SOAP

JMS : L'interface de programmation Java Message Service (JMS) permet d'envoyer et de recevoir des messages de manière asynchrone entre applications ou composants Java. JMS permet d'implémenter une architecture de type MOM (message oriented middleware). Un client peut également recevoir des messages de façon synchrone dans le mode de communication point à point. Wikipedia

https://fr.wikipedia.org/wiki/Java_Message_Service

Framework : En programmation informatique, un *framework* (appelé aussi infrastructure logicielle, cadre applicatif, cadre d'applications, cadriciel, socle d'applications ou encore infrastructure de développement) désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). Wikipedia

https://fr.wikipedia.org/wiki/Framework

OAuth : « **OAuth** est un protocole libre, créé par Blaine Cook et Chris Messina. Il permet d'autoriser un site web, un logiciel ou une application (dite « consommateur ») à utiliser l'API sécurisée d'un autre site web (dit « fournisseur ») pour le compte d'un utilisateur. OAuth *n'est pas* un protocole d'authentification, mais de « délégation d'autorisation ». Wikipédia

https://fr.wikipedia.org/wiki/OAuth