



République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
ÉCOLE NATIONALE D'INGÉNIEURS DE TUNIS
Département Technologies de l'information et de la Communication



RAPPORT DE PROJET DE FIN D'ÉTUDES

Présenté en vue de l'obtention du
Diplôme National d'Ingénieur en Informatique

Par Mohamed FARHAOUI

Migration de l'application Rmess vers une architecture adaptée aux cloud

Soutenu le **12-07-2021**

Devant le Jury :

Président : **Pr. Wady Naanaa**

Rapporteur : **Dr. Mohamed Koubaa**

Encadrant professionnel : **M. Taha BAKLI**

Encadrant académique : **Dr. Mohamed Ramzi HADDAD**

Réalisé au sein de **IGA Groupe**



J'autorise l'étudiant Mohamed FARHAOUI à faire le dépôt de son rapport de stage en vue d'une soutenance.

Encadrant professionnel, **M. Taha BAKLI**

Signature et cachet

J'autorise l'étudiant Mohamed FARHAOUI à soumettre son rapport de stage en vue d'une soutenance.

Encadrant académique, **Dr. Mohamed Ramzi HADDAD**

Signature

Dédicaces

Je dédie ce travail :

À mes chers parents, qui m'ont toujours poussé et motivé dans mes études. Pour tous les sacrifices qu'ils ont faits et pour tout le soutien qu'ils ont offert tout au long de mes études. Sans eux, je n'aurais certainement pas fait d'études longues. Ce projet de fin d'études représente donc l'aboutissement du soutien et des encouragements qu'ils en soient remerciés par cette trop modeste dédicace.

J'espère qu'ils puissent trouver dans ce modeste travail un témoignage d'amour et d'affection envers eux.

À mon frère et mes sœurs pour leurs encouragements incessants.

À mes amies et mes collègues, pour leur encouragement et pour tous les bons moments qu'on a vécus ensemble. J'espère que notre amitié durera éternellement.

À celui qui m'a conseillé de choisir le domaine d'informatique.

Pour toute personne m'a donné un conseil dans ma vie.

Mohamed FARHAOUI

Remerciements

En guise de reconnaissance, je tiens à témoigner mes sincères remerciements à toutes les personnes qui ont participé de près ou de loin au bon déroulement de mon projet de stage et à l'élaboration de ce modeste travail.

Je tiens particulièrement à témoigner toute ma gratitude et ma haute considération à mon encadrant académique monsieur Mohamed Ramzi HADDAD pour l'aide qu'il m'a apportée durant toutes les étapes de ce projet. Les discussions que nous avons tenues ont permis d'orienter ce travail d'une manière sûre et pertinente. Je le remercie vivement pour sa disponibilité et ses efforts.

J'adresse aussi mes profonds remerciements à monsieur Taha BAKLI, mon encadrant au sein d'IGA Groupe qui m'a accompagné et formé durant cette expérience professionnelle avec beaucoup de pédagogie et de patience aussi pour sa rigueur scientifique, sa disponibilité et son sens d'écoute et d'échange.

Je voudrais témoigner par la suite ma reconnaissance à monsieur Fatah MAIRECHE, chef de projet chez IGA, qui malgré ses responsabilités et ses occupations, a toujours eu le temps pour m'écouter et me faire bénéficier de ses conseils.

Je tiens à remercier mes collègues d'IGA Groupe pour leur aide, leur soutien, leurs conseils et leur amitié. Le temps que j'ai passé dans l'entreprise a été très agréable et fructueux et c'est grâce à eux.

J'exprime également toute ma reconnaissance à mes enseignants pour la qualité de leurs enseignements, leurs conseils et l'intérêt incontestable qu'ils portent à tous les étudiants.

Mon dernier mot s'adresse à tous les membres du jury pour l'honneur qu'ils me font par la participation à l'évaluation de ce travail.

Résumé

Le présent rapport synthétise le travail effectué dans le cadre du projet de fin d'études pour l'obtention du diplôme national d'ingénieur en informatique au sein de l'entreprise IGA Groupe . Le but du travail est de faire la migration d'une application d'intégration qui est basée sur l'architecture ESB vers une architecture moderne adaptée au cloud.

Mots clés : Apache camel, SOAP, file JMS, routes, JPA.

Abstract

This report summarizes the work, carried out as part of my end of studies project, in order to obtain my national degree, in Software Engineering within the IGA Groupe company. The aim of the work is to migrate an integration application that is based on the ESB architecture to a modern architecture adapted to the cloud.

Keywords : Apache camel, SOAP, Queue JMS, routes, JPA.

Table des matières

Introduction générale	11
1 Cadre générale du projet	13
Introduction	13
1.1 Présentation de l'organisme d'accueil	13
1.2 Contexte du projet	14
Conclusion	14
2 Analyse et spécification des besoins	16
Introduction	16
2.1 Etude de l'existant	16
2.2 Solution proposée	19
2.3 Cahier des charges	19
2.3.1 Spécification des besoins fonctionnels	19
2.3.2 Spécification des besoins non fonctionnels	26
Conclusion	26
3 Conception	28
Introduction	28
3.1 Conception générale	28
3.1.1 Architecture générale	28
3.1.2 Conception des données	30
3.2 Conception détaillée	33
3.2.1 Module de réception	33

3.2.2	Module de onramp	34
3.2.3	Module de l'identification	35
3.2.4	Module de conversion	36
3.2.5	Module de distribution	37
3.2.6	Module de replay	40
3.2.7	Module d'archivage	40
3.2.8	Module logger	41
	Conclusion	41
4	Réalisation	42
	Introduction	42
4.1	Environnement de travail	42
4.1.1	Environnement matériel	42
4.1.2	Environnement logiciel	42
4.2	Apache Camel : framework d'intégration	44
4.2.1	Présentation	44
4.2.2	Architecture	44
4.2.3	Modèle de messagerie	44
4.2.4	Les concepts de Camel	47
4.2.5	Les avantages de Apache Camel	48
4.3	Scénarios d'exécution	49
4.3.1	Service de connection	50
4.3.2	Service onramp	51
4.3.3	Service d'identification	53
4.3.4	Service de conversion	55
4.3.5	Service de distribution	56
4.3.6	Service d'archivage	59
4.3.7	Service logger	60
4.3.8	Service replay	61
4.4	Tests et performance	61

4.4.1	Tests de validation	62
4.4.2	Observabilité et tests de performances	64
Conclusion	68
Conclusion générale et perspectives		69
Bibliographie		72

Liste des figures

1.1	Contexte du projet.	15
2.1	Architecture ESB.	17
2.2	Diagramme de cas d'utilisation globale.	22
2.3	Diagramme de cas d'utilisation "Recevoir un message".	22
2.4	Diagramme de cas d'utilisation "Transformer un message".	23
2.5	Diagramme de cas d'utilisation "Identifier un message".	23
2.6	Diagramme de cas d'utilisation "Convertir un message".	24
2.7	Diagramme de cas d'utilisation "Distribuer un message".	24
2.8	Diagramme de cas d'utilisation "Rejouer un message".	25
2.9	Diagramme de cas d'utilisation "Archiver un message".	25
2.10	Diagramme de cas d'utilisation "Stocker les exception".	26
3.1	Architecture globale de l'application.	29
3.2	Diagramme de paquetage.	30
3.3	Example de fichier de type ota (format xml).	31
3.4	Exemple de fichier de type tms (format tms).	31
3.5	Diagramme de classe de l'application RMESS.	32
3.6	Diagramme de séquence service de connection.	33
3.7	Diagramme de séquence relatif à la réception et onramp.	34
3.8	Diagramme de séquence relatif à l'identification.	35
3.9	Diagramme de séquence relatif à l'appel du web service.	35
3.10	Diagramme de séquence relatif au service de conversion.	36

3.11 Diagramme de séquence relatif au sous processus de service conversion.	37
3.12 Diagramme de séquence relatif à l'appel du méthode http post.	37
3.13 Diagramme de séquence relatif à l'appel du web service de conversion.	38
3.14 Diagramme de séquence relatif à l'appel du web service de distribution avec MCTO.	38
3.15 Diagramme de séquence relatif au service de distribution.	39
3.16 Diagramme de séquence relatif à au service replay.	40
3.17 Diagramme de séquence relatif à au service d'archivage.	41
3.18 Diagramme de séquence relatif à au service d'archivage.	41
 4.1 Architecture de Apache camel.	45
4.2 (a) Contenu message Camel (b) Communication expéditeur destinataire	46
4.3 Exchange Camel	46
4.4 Camel context	47
4.5 Exemple d'un message "AIR".	50
4.6 Service de connection	51
4.7 Service onramp	52
4.8 Résultat service onramp	52
4.9 Service d'identification.	54
4.10 Identification spécifique.	54
4.11 Identification standard.	55
4.12 Service de conversion.	55
4.13 Résultat de service de conversion.	56
4.14 Service de distribution.	57
4.15 Résultat final.	58
4.16 Concaténation des messages AIR.	60
4.17 Archivage des anciens messages.	60
4.18 Service logger.	61
4.19 Service replay.	61
4.20 Une vue sur la table Archive.	62
4.21 Une vue sur la table Journal.	62

4.22 Une vue sur la table ArchiveComplementaire.	62
4.23 Test de validation table Journal.	63
4.24 Test de validation table Archive.	63
4.25 Test de validation table ArchiveComplementaire.	64
4.26 Temps de démarrage de l'ancienne application.	64
4.27 Temps de démarrage de nouvelle application.	65
4.28 Exposition des métriques avec hawtio.	67

Liste des tableaux

2.1	Les besoins associés à chaque acteurs.	21
3.1	Entités et description.	33
4.1	Environnement matériel.	42
4.2	Liste des sources de données.	49
4.3	Liste des services de distribution.	58
4.4	Temps de traitement des messages.	65

Liste des abréviations

+

API	Application Programming Interface
CSV	comma-Separated values
DAO	DataAccess Object
DSL	Domain Specific Language
EIP	Enterprise Integration Pattern
ESB	Enterprise Service Bus
FTP	File Transport Protocol
GDS	GlobalDistributed System
HTTP	HypertextTransfer Protocol
JMS	JavaMessage Service
JMX	Java Management Extensions
JSON	JavaScript Object Notation
MEP	MessageExchange Pattern
ORM	Object Relational Mapping
REST	Representational State Transfer
SFTP	Ssh File Transport Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	StructuredQuery Language
UML	UnifiedModeling Language
WS	WebService

Introduction générale

Dans le monde du développement informatique, un logiciel ne peut pas exister dans un vide informationnel. C'est du moins l'hypothèse que nous, les ingénieurs logiciels, pouvons faire pour la plupart des applications que nous développons.

À n'importe quelle échelle, chaque logiciel communique avec un autre logiciel pour diverses raisons, notamment pour obtenir des données de référence de quelque part, pour envoyer des signaux de surveillance, pour être en contact avec d'autres services tout en faisant partie d'un système distribué, et plus encore.

La conception de l'architecture pour l'intégration des systèmes est l'un des plus grands défis pour les ingénieurs logiciels.

C'est dans ce contexte que s'inscrit la mission que m'a confiée l'entreprise IGA Groupe et plus spécifiquement l'équipe IGA Voyage et qui consiste à migrer une application basée sur l'architecture ESB vers une architecture adaptée au Cloud.

Ce rapport prend en charge la description des phases de la conception et de la réalisation de ce projet. Il s'étale sur quatre chapitres.

Le premier chapitre comporte une brève présentation de l'organisme d'accueil, le cadre général de ce projet et le sujet à traiter. Nous y intégrons l'étude de l'existant, la problématique et la solution proposée.

Le deuxième chapitre aborde les concepts théoriques clés, liés à l'architecture d'intégration des applications.

Le troisième chapitre exposera notre vision conceptuelle du projet via une présentation globale de l'architecture de notre solution et l'architecture détaillée de chaque service de l'application munie des diagrammes de classes et de séquences.

Le quatrième chapitre traite les détails de la réalisation : l'environnement de travail, les différents patrons d'intégration d'entreprise utilisés pour chaque service de l'application ainsi que des captures d'écran illustrant quelques fonctionnalités de notre application.

Le dernier chapitre illustre une étude sur les tests de validation, la performance et la mesure des métriques de la nouvelle application.

Chapitre 1

Cadre générale du projet

Introduction

Ce premier chapitre expose le cadre général et le contexte de travail. Nous présenterons dans un premier temps l'entreprise Française IGA Groupe et son prestataire de service 2IGA Tunis de droit Tunisien. Par la suite, nous expliciterons le contexte du projet.

1.1 Présentation de l'organisme d'accueil

Le groupe IGA est un acteur majeur sur le marché de l'édition et l'intégration de logiciels pour les secteurs de l'assurance et du voyage depuis 1985. IGA est devenu un partenaire de référence pour ses clients en étant en mesure de répondre à leurs défis et exigences d'affaires et cela détermine bien les missions de groupe IGA qui s'appuient sur :

- La compréhension des enjeux du client et la spécification des priorités.
- L'aide dans les choix technologiques et la prise de décision des solutions qui peuvent améliorer la performance de l'organisation de client.
- Développer des applications informatiques dans un domaine spécifique comme le voyage puis personnaliser ces applications pour répondre aux spécificités de l'entreprise.
- La bénédice de tous les évolutions technologiques auprès de plus de 2000 points de vente équipés des solutions IGA.

IGA Groupe est une entreprise d'édition et d'intégration de logiciels avec :

- 13000 Utilisateurs.
- 2000 clients IGA.
- 110 salariés.

Le Groupe IGA a une expérience et une expertise sectorielle sur leur marchés de référence : le voyage et l'assurance [10].

IGA Voyage a su se démarquer et devenir le partenaire privilégié des agences de voyages (plus de 2000 agences équipées en France) grâce à une expertise d'affaires reconnue et des investissements constants pour

innover et répondre aux besoins de ce marché.

Pour répondre aux exigences et besoins de ses clients IGA Groupe développe un logiciel IGA T-9 mais en fait ce n'est pas un simple logiciel d'agence de voyages. C'est une suite logicielle complète et modulable créée à partir des dernières innovations technologiques et fonctionnelles. Cette solution répond à tous les besoins des agences et permet d'optimiser le pilotage de leurs activités pour les recentrer sur la valorisation de l'expérience client [11].

On a effectué le stage au sein de l'entreprise 2IGA Tunis, qui est le prestataire de service pour le compte de IGA société de droit français. Cette entreprise a été créée en 2018 avec :

- Effectifs : 15 personnes.

- Prestations :

*Support fonctionnel voyage.

*Support fonctionnel assurance.

*Service développement informatique.

1.2 Contexte du projet

Les clients de IGA Voyage sont les agences de voyage, pour cela l'équipe a développé un logiciel T9 qui permet à l'agence d'établir l'ensemble de la facturation sur toutes les ventes qui ont été réalisées. Cet outil permet aussi de suivre les ventes, les achats, la trésorerie et même assurer la tenue comptable de l'agence et d'établir des états Reporting.

En contre partie les clients d'une agence de voyage sont des clients sociétés, des clients tourisme et des clients de passage.

Un client fait une réservation dans une agence, cette réservation va être envoyé sous forme d'un message vers un système distribué globale qui est associé à plusieurs agences.

Chaque GDS stocke les fichiers reçus et les envoie en temps réel vers un dossier spécifique et par la suite il y a l'intervention de l'application RMESS.

Cette application a comme objectif de prendre toutes les messages de n'importe quel type et faire un processus de traitement pour générer un format universel et exploitable qui est le format pivot pour n'importe quel fichier et envoie le résultat vers le back office T9.

La figure 1.1 illustre bien le processus complet de réservation et la position de l'application RMESS dans ce processus.

Le présent travail s'inscrit dans le cadre de mon projet de fin d'études en vue de l'obtention du diplôme national d'ingénieur en informatique. Ce projet consiste à migrer une application de traitement des fichiers vers une architecture moderne adaptée au cloud et avec des nouvelles technologies.

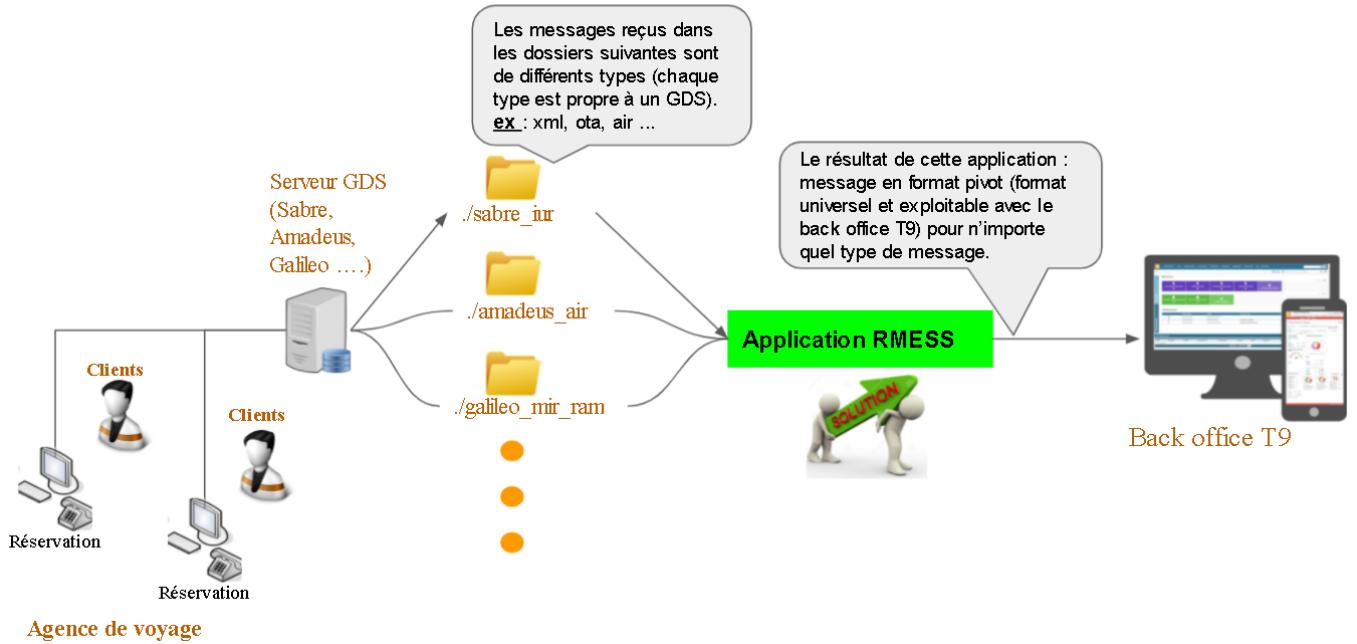


FIGURE 1.1 – Contexte du projet.

[3]

Conclusion

Ce premier chapitre constitue une étape primordiale pour fixer les repères de notre projet. Après avoir présenté l'organisme d'accueil, nous avons déterminé le contexte du projet. Dans le chapitre à suivre, nous détaillerons la solution proposée et leur cahier des charges ainsi que l'analyse et les spécifications des besoins.

Chapitre 2

Analyse et spécification des besoins

Introduction

Dans ce chapitre nous allons étudier en premier lieu l'existant. Ensuite, nous passerons à présenter le cahier des charges de solution proposé en discutant l'analyse et la spécification des besoins auxquelles notre application doit répondre.

2.1 Etude de l'existant

Le groupe IGA voyage est en train de migrer leur logiciel vers des architectures adaptées au cloud, parmi les applications qui constituent leur logiciel est l'application RMESS qui a pour rôle de prendre des messages des clients issus de différentes plateformes des agences de voyages et faire le traitement de ces fichiers tout au long d'un processus complet pour y arriver vers un format bien déterminé pour qu'ils soient pris par le logiciel T9. Cette application est développée par l'architecture entreprise service bus (ESB).

Cette architecture est basé sur un bus qui implémente un système de communication entre des applications logicielles interagissant mutuellement dans une architecture orientée services (SOA). Il représente une architecture logicielle pour l'informatique distribuée et est une variante spéciale du modèle client-serveur plus général, dans lequel toute application peut se comporter en serveur ou en client. Son utilisation principale est l'intégration d'applications d'entreprise (EAI) de paysages de services hétérogènes et complexes.

Un ESB contrôle une SOA en diminuant la quantité, la taille et les problèmes des interfaces entre les applications et les services. Pour cela l'ESB offre plusieurs fonctionnalités comme par exemple :

La transformation des messages, routage des données, conversion de protocole de transport, gestion des transactions....

La figure 2.1 illustre les fonctionnalités offertes par un bus ESB.

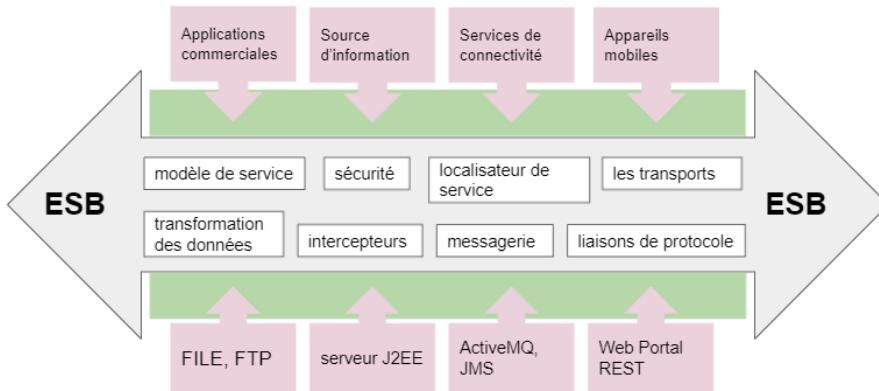


FIGURE 2.1 – Architecture ESB.

[15]

Parmi les désavantages de cette architecture qui nous ramène à chercher une autre solution avec une autre technologie sont :

- Vitesse de communication plus lente, en particulier pour les services déjà compatibles.
- Point de défaillance unique, peut couper toutes les communications dans la totalité de l'application.
- Complexité élevée de configuration et de maintenance.

Sans oublier de mentionner que cette architecture nécessite d'avoir un serveur d'application et cela rend le lancement lourd.

-Aussi bien si on parle de migration vers le cloud on doit prendre en considération la notion de scalabilité.

•Scalabilité verticale : cela revient à ajouter des ressources à un unique serveur en lui ajoutant de la RAM, en changeant son CPU pour un plus rapide ..., mais cette solution a des limites est qu'on peut multiplier par 2 les capacités de notre serveur, peut-être même par 5, mais pas par 100.

Cela reste dans la majorité des cas la solution la plus pragmatique, et en particulier pour des applicatifs en production depuis de nombreuses années.

•Scalabilité horizontale : consiste à ajouter de nouveaux serveurs réalisant le même type de tâche et donc cela va résoudre plusieurs problèmes comme par exemple le stockage des données qui va être sur un autre service ou bien sur plusieurs services, aussi le problème de

défaillance ; en cas de défaillance matérielle sur l'une des instances, les autres instances pourront absorber la charge avec une interruption de service minimale pour l'utilisateur ainsi que la mise à jour de notre application sans interruption de service.

Cette application qui est basé sur l'architecture ESB, n'est pas scalable qu'à la verticale.

Bien entendu, ceci a engendré plusieurs défis :

- La nouvelle application doit avoir les mêmes fonctionnalités que l'ancienne.

• Les applications utilisent différents protocoles et formats de données. Cela signifie qu'un système d'intégration est un rideau pour les transformations de données et les adaptateurs vers d'autres participants et utilise une variété de technologies. Ceux-ci peuvent inclure des appels d'API REST simples, mais peuvent également accéder à un courtier de file d'attente, envoyer des commandes CSV via FTP ou extraire des données par lots vers une table de base de données. C'est une longue liste et elle ne sera jamais plus courte.

• La haute disponibilité, la vitesse de traitement des messages et la fiabilité sont des priorités pour les fonctionnalités de base de la nouvelle architecture.

• Le lancement rapide de l'application et la consommation limitée des ressources car si on parle de cloud on parle de l'un des principes et celui du coût "vous payez pour ce que vous obtenez".

• Pour assurer la facilité de maintenance de l'application, une couche middleware doit fournir une image claire des dépendances avec un routage et une transformation des données polyvalents.

Ces idées doivent être gardées à l'esprit lors de la conception de l'intégration et du choix de la solution middleware la plus appropriée.

L'une des manières possibles de le gérer consiste à utiliser l'architecture bus de service d'entreprise (ESB). Mais les ESB fournies par les principaux fournisseurs sont généralement trop lourdes et posent souvent plus de problèmes :

Il est presque impossible de démarrer rapidement avec une ESB, sa courbe d'apprentissage est assez abrupte et sa flexibilité est sacrifiée à une longue liste de fonctionnalités et d'outils intégrés. Pour cela les solutions d'intégration open source légères sont de loin supérieures : elles sont plus élastiques, faciles à déployer dans le cloud et faciles à faire évoluer.

L'intégration de logiciels n'est pas facile à faire. Aujourd'hui, alors que nous construisons des architectures de micro services et traitons des essaims de petits services, nous avons également des attentes élevées quant à l'efficacité de leur communication.

2.2 Solution proposée

Le développement du routage et de la transformation des données est comme le développement logiciel en général, implique des opérations répétitives. L'expérience dans ce domaine a été résumée et systématisée par des professionnels qui traitent les problèmes d'intégration depuis un certain temps. Dans le résultat, il y a un ensemble de modèles extraits appelés modèles d'intégration d'entreprise (EIP) utilisés pour concevoir des flux de données.

Ces EIP aident à rédiger une conception d'architecture, qui n'exploré pas le niveau du code mais décrit les flux de données de manière suffisamment détaillée. Une telle notation pour décrire les itinéraires d'intégration rend non seulement la conception concise, mais définit également une nomenclature commune et un langage commun, qui sont très importants dans le contexte de la résolution d'une tâche d'intégration avec des membres de l'équipe de divers domaines d'activité.

Pour cela nous avons besoin d'un framework middleware orienté message qui met en œuvre la liste des EIP. Il utilise ces modèles, prend en charge tous les protocoles de transport courants et comprend un vaste ensemble d'adaptateurs utiles.

2.3 Cahier des charges

2.3.1 Spécification des besoins fonctionnels

Nous présentons dans cette partie les différents acteurs de notre système et nous spécifions les besoins fonctionnels ainsi que les différents diagrammes de cas d'utilisation.

2.3.1.1 Identification des acteurs

L'identification des acteurs d'un système permet en première phase de délimiter le système et de comprendre le rôle de chaque acteur et de ses exigences. Dans notre application, nous avons identifié les acteurs principaux suivants qui sont tous des acteurs non humains et sont divisés en deux catégories :

1- Acteurs primaires :

- **Service de connection** : Reçoit un message sauvegardé dans un dossier.
- **Service onramp** : Transforme un message en lui ajoutant leur configuration.
- **Service d'identification** : Identifie un message en appelant un web service.
- **Service de conversion** : Convertit un message en appelant un ou plusieurs web services.

- **Service de distribution :** Distribue un message vers le back office correspond.
- **Service replay :** Traiter de nouveau un message bloqué.
- **Service d'archivage :** Archive les anciens messages.
- **Service logger :** Sauvegarde les exceptions dans la base de données.

1- Acteurs secondaires :

- **Serveur BD :** Serveur avec lequel on communique avec la base de données pour le stockage des données nécessaires.
- **Système de messagerie :** Donne la possibilité de communication entre les services.
- **Serveur HTTP**
- **Serveur FTP**
- **Service Web**

2.3.1.2 Besoins fonctionnels

Notre application traite des messages sous un processus complet et donc cela nécessite d'avoir plusieurs services et plusieurs serveur pour avoir le bon résultat. Dans le tableau 2.1 nous détaillerons les besoins associés à chaque acteur.

Acteur	Besoins associés
Service de connection	<ul style="list-style-type: none"> -Recevoir un message -Ajouter un en-tête correspond au type de ce message. -Envoyer le message vers la file d'attente de service onramp.
Service onramp	<ul style="list-style-type: none"> -Récupérer un message de file d'attente. -Récupérer la configuration de message. -Envoyer le message transformé vers la file d'attente de service identification.
Service d'identification	<ul style="list-style-type: none"> -Recevoir un message de la file d'attente. -Identifier le message : <ul style="list-style-type: none"> • Identifier de manière spécifique. • Identifier de manière standard. -Mettre à jour la base de données. -Envoyer le résultat vers la file d'attente de service de conversion.
Service de conversion	<ul style="list-style-type: none"> -Récupérer un message de la file d'attente. -Convertir le message vers le format pivot. -Mettre à jour la base de données. -Envoyer le message vers la file d'attente de distribution.

Service de distribution	<ul style="list-style-type: none"> -Récupérer le message à partir de la file d'attente. -Distribuer le message vers le back office correspond : <ul style="list-style-type: none"> • Distribuer le message vers un serveur ftp. • Distribuer le message vers un serveur http. • Distribuer le message vers un serveur locale. -Mettre à jour la base de données.
Service replay	<ul style="list-style-type: none"> -Consulter et récupérer les messages bloqués de la base de données. -Envoyer les messages vers la file d'attente d'identification.
Service d'archivage	<ul style="list-style-type: none"> -Récupérer la liste des anciens messages. -Enregistrer les messages récupérés dans une autre table de la base de données.
Service logger	<ul style="list-style-type: none"> -Récupérer la liste des exceptions générées dans la file d'attente. -Sauvegarder les exceptions dans la base de données.
Service web d'identification (acteur secondaire)	-Identifier un message en extractant des informations nécessaires.
Service web de conversion (acteur secondaire)	-Convertir un message en format pivot.
Serveur HTTP (acteur secondaire)	—
Serveur FTP (acteur secondaire)	—
Serveur BD (acteur secondaire)	—
Système de messagerie (acteur secondaire)	—

Tableau 2.1: Les besoins associés à chaque acteurs.

2.3.1.3 Vue globale du système

Le diagramme de cas d'utilisation UML est la principale forme d'exigences système/-logiciel pour un nouveau programme logiciel sous-développé. Les cas d'utilisation spécifient le comportement attendu (quoi), et non la méthode exacte pour le réaliser (comment). Une fois spécifiés, les cas d'utilisation peuvent être représentés de manière textuelle et visuelle (c'est-à-dire par un diagramme de cas d'utilisation). Un concept clé de la modélisation des cas d'utilisation est qu'elle nous aide à concevoir un système du point de vue de l'utilisateur final. C'est une technique efficace pour communiquer le comportement du système dans les termes de l'utilisateur en spécifiant tout comportement du système visible de l'extérieur[14].

La figure 2.2 montre le diagramme de cas d'utilisation global de notre système.

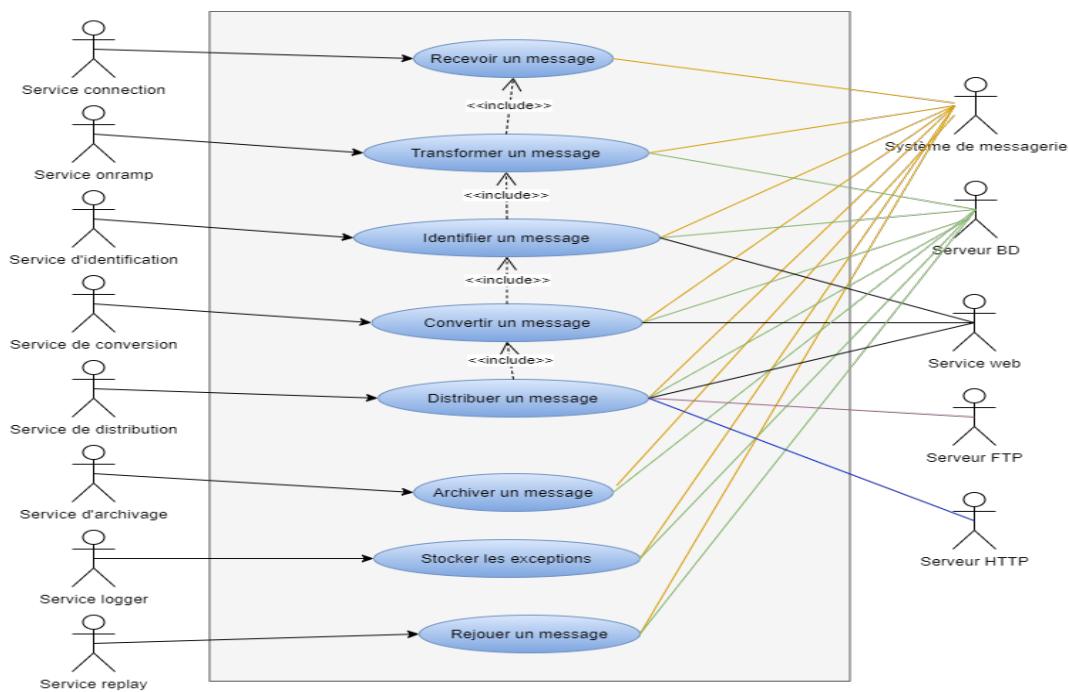


FIGURE 2.2 – Diagramme de cas d'utilisation globale.

2.3.1.4 Raffinement des cas d'utilisation

Dans cette partie nous raffinons les diagrammes de cas d'utilisation par fonctionnalité.

•Diagramme de cas d'utilisation "Recevoir un message"

La figure 2.3 montre le diagramme de cas d'utilisation "Recevoir un message".

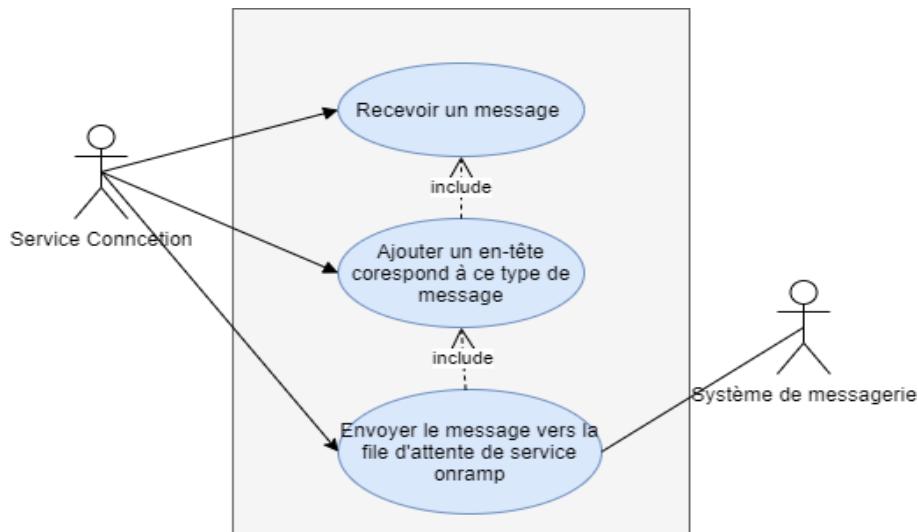


FIGURE 2.3 – Diagramme de cas d'utilisation "Recevoir un message".

•Diagramme de cas d'utilisation "Transformer un message"

La figure 2.4 illustre le diagramme de cas d'utilisation "Transformer un message".

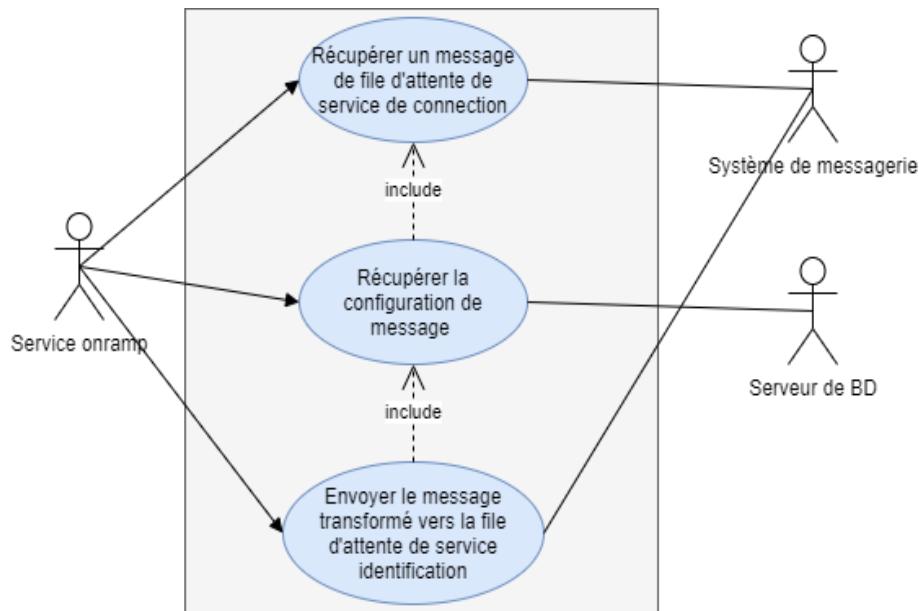


FIGURE 2.4 – Diagramme de cas d'utilisation "Transformer un message".

•Diagramme de cas d'utilisation "Identifier un message"

Le diagramme de cas d'utilisation "Identifier un message" est illustré dans la figure 2.5.

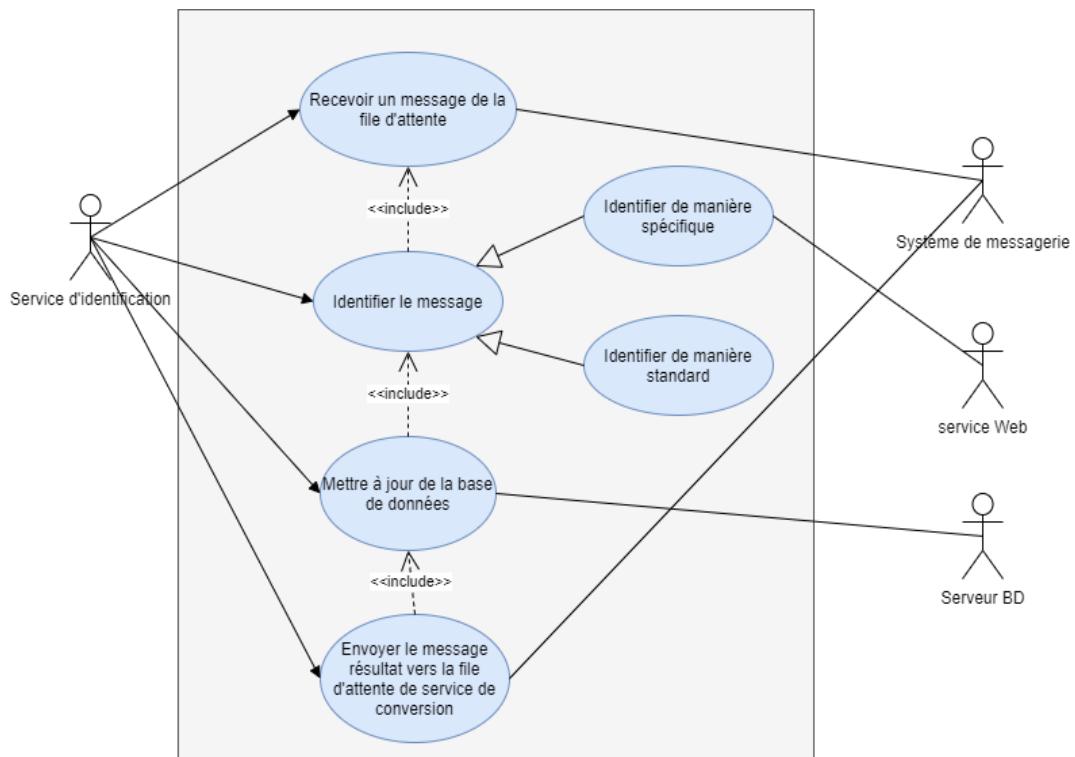


FIGURE 2.5 – Diagramme de cas d'utilisation "Identifier un message".

•Diagramme de cas d'utilisation "Convertir un message"

La figure 2.6 présente le diagramme de cas d'utilisation "convertir un message".

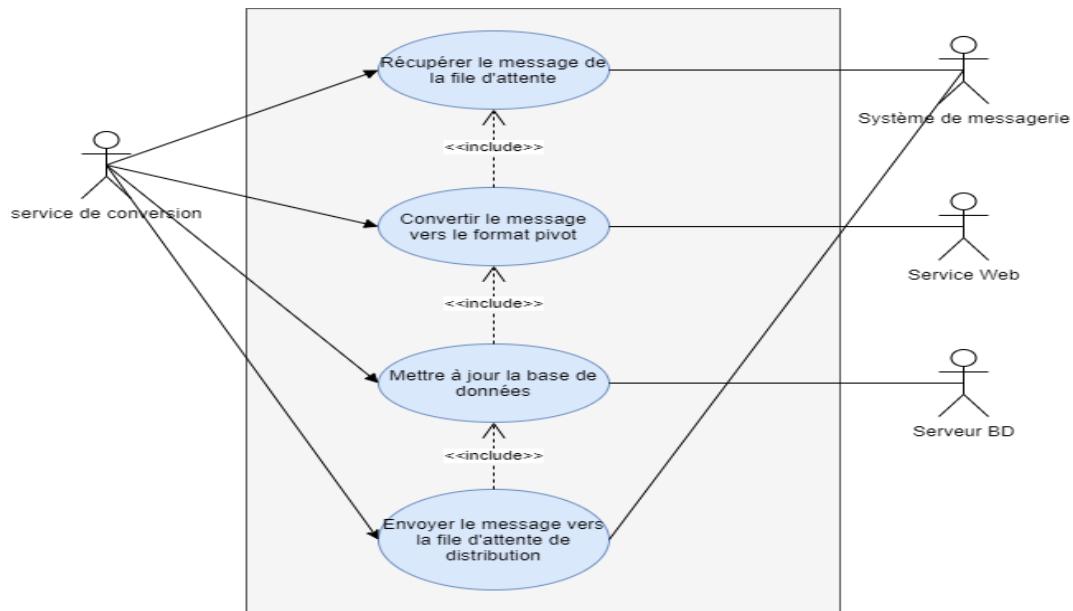


FIGURE 2.6 – Diagramme de cas d'utilisation "Convertir un message".

•Diagramme de cas d'utilisation "Distribuer un message"

Le diagramme de cas d'utilisation "Distribuer un message" est détaillé dans la figure 2.7.

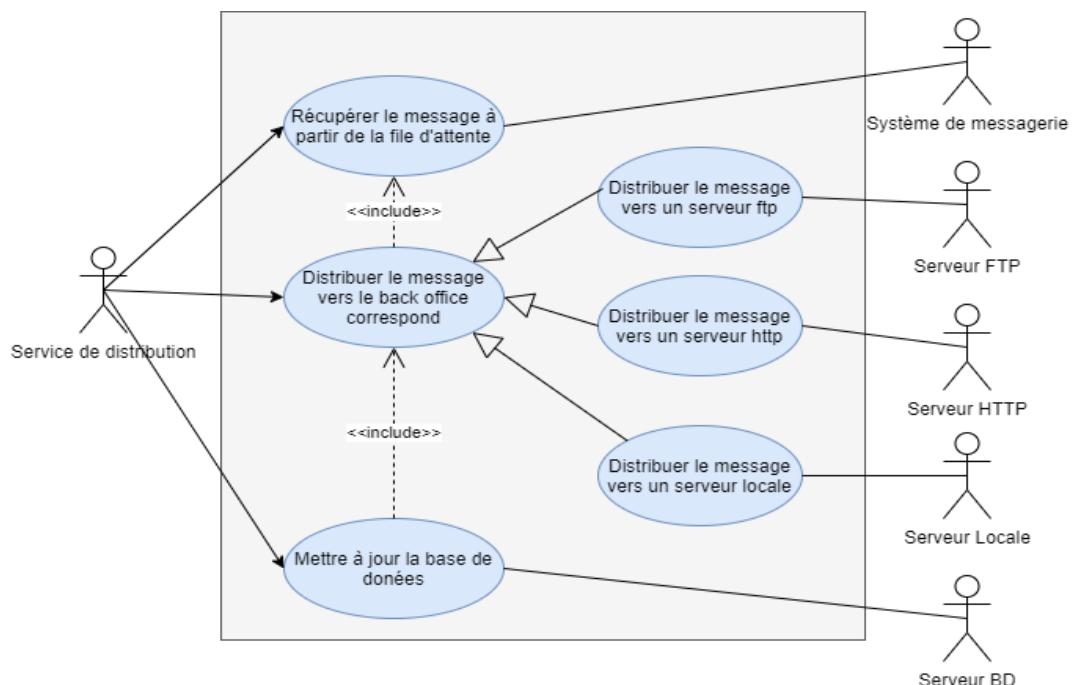


FIGURE 2.7 – Diagramme de cas d'utilisation "Distribuer un message".

•Diagramme de cas d'utilisation "Rejouer un message"

Un message bloqué dans une étape de traitement va être rejouer de nouveau et donc on doit avoir d'un service qui exporte ce message de la base de données et lui envoie vers le service d'identification.

La figure 2.8 montre bien le diagramme de cas d'utilisation "Rejouer un message".

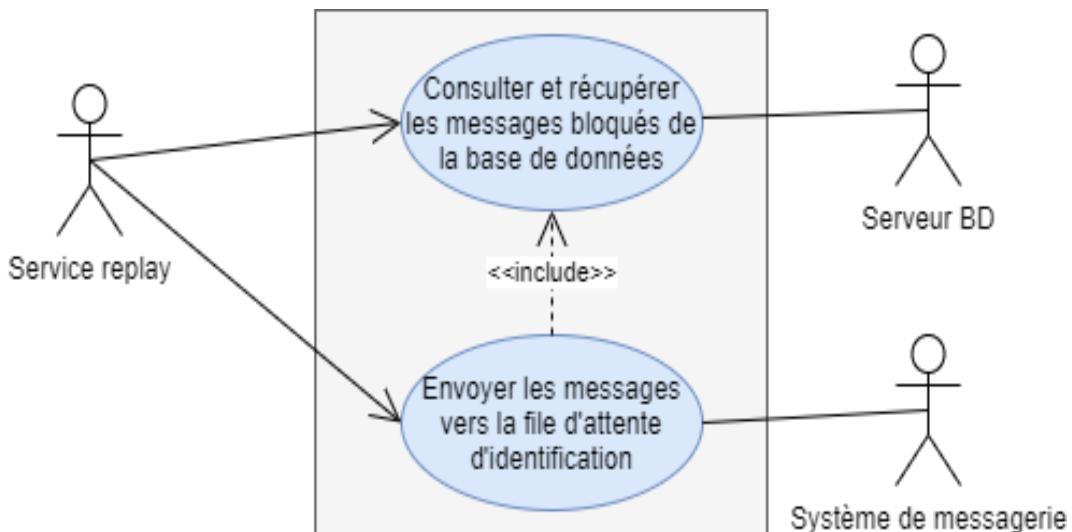


FIGURE 2.8 – Diagramme de cas d'utilisation "Rejouer un message".

•Diagramme de cas d'utilisation "Archiver un message"

"Archiver un message" c'est à dire changer l'emplacement d'un message vers l'archive. Cette fonctionnalité est détaillée dans le diagramme de cas d'utilisation de la figure 2.9.

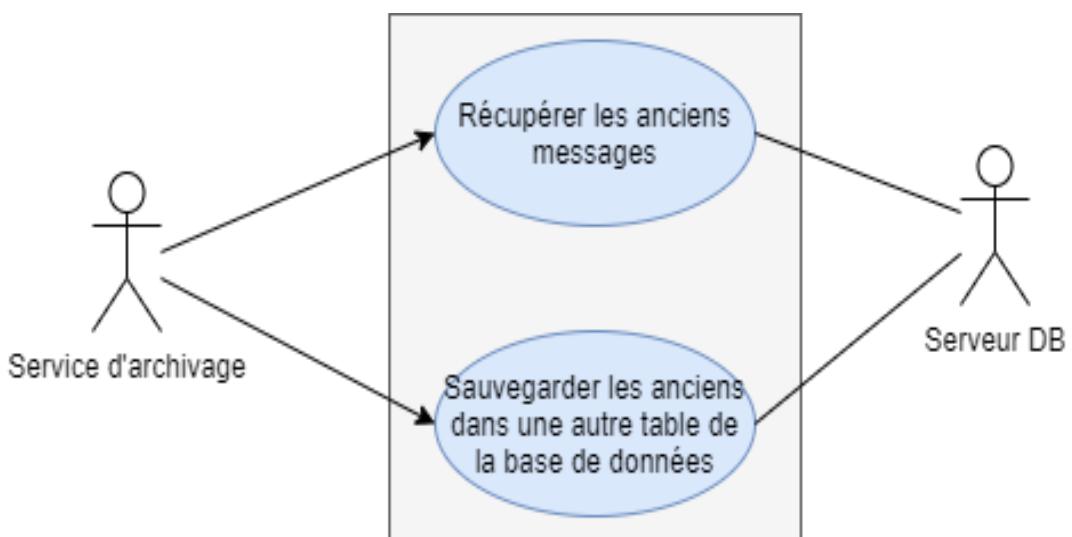


FIGURE 2.9 – Diagramme de cas d'utilisation "Archiver un message".

•Diagramme de cas d'utilisation "Stocker les exception"

Parmi les aspects importants de notre application et le stockage de toutes les exceptions générées lors du traitement d'un message. La figure 2.10 montre le diagramme de cas d'utilisation "stocker les exceptions".

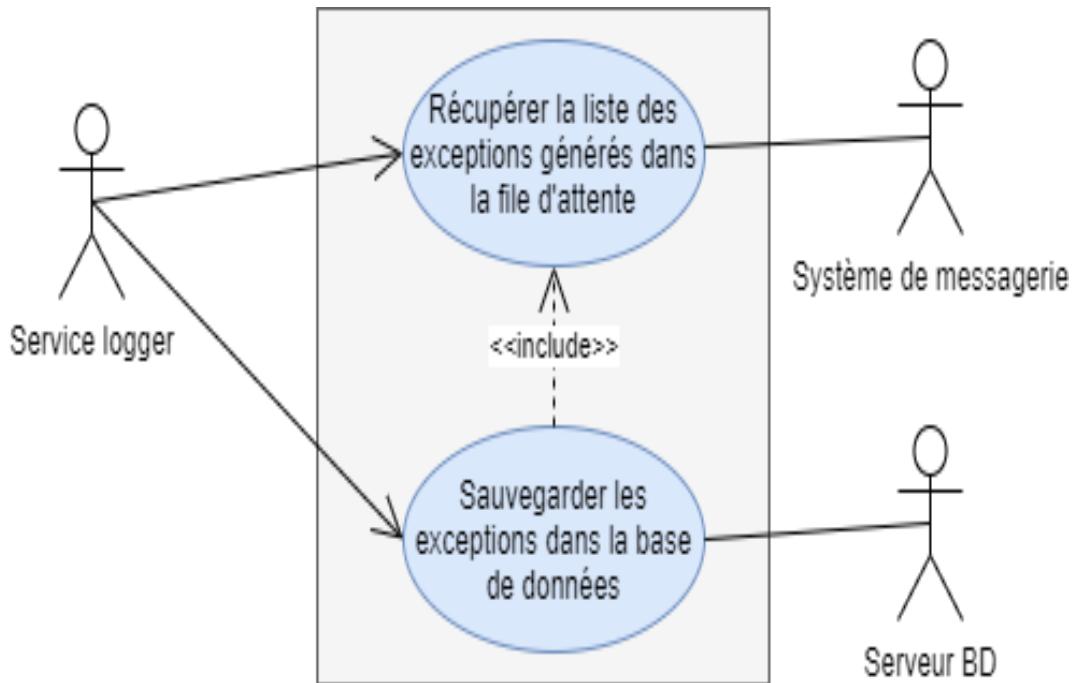


FIGURE 2.10 – Diagramme de cas d'utilisation "Stocker les exception".

2.3.2 Spécification des besoins non fonctionnels

Les besoins non fonctionnels représentent les contraintes auxquelles le système doit répondre afin de garantir un bon déroulement. Parmi ces besoins nous pouvons énumérer :

- **Extensibilité** : L'architecture de l'application doit être claire pour permettre des futures évolutions ou améliorations.
- **Aptitude à la maintenance** : l'application doit être ouverte à la modification et maintenable facilement.
- **Portabilité** : On doit avoir une application facile à installer à n'importe quel système d'exploitation.
- **Performance** : C'est mieux d'avoir une application avec :
 - Temps de réponse rapide – le chargement de l'application.
 - Temps de traitement minimale – traitement des messages, appel au web services, stockage des données.

Conclusion

Dans ce chapitre, nous avons cerné l'étude de l'existant de notre application. Nous avons aussi présenté le cahier des charges pour la solution proposée qui contient la spécification des besoins et les différents diagrammes de cas d'utilisation des différentes phases de l'application. Cette phase va nous permettre d'entamer le chapitre suivant de conception.

Chapitre 3

Conception

Introduction

Dans ce chapitre, nous nous focalisons sur la conception générale et détaillée de la nouvelle solution de l'application ainsi qu'une description logique de la façon dont le processus de traitement des messages fonctionne.

3.1 Conception générale

3.1.1 Architecture générale

L'application RMESS sert à traiter des messages pendant un processus complet. Ce processus se décompose en 5 services principales et 3 services supplémentaires :

- Le service de réception : sert à récupérer un fichier à partir d'un dossier.
- Le service onramp : consiste à enregistrer le contenu de message dans la base de données et récupérer leur configuration.
- Le service d'identification : a comme rôle d'appeler un service web pour faire l'extraction des informations nécessaires à partir d'un message.
- Le service de conversion : il s'agit d'un processus complet de conversion ; il peut être donc un appel à un web service comme il peut être un appel à plusieurs web services dont le but est de faire la conversion d'un message vers le format acceptable.
- Le service de distribution : sert à distribuer un message vers le back office d'un ou de plusieurs distributeurs.
- Service replay : pour refaire de traitement d'un message bloqué.
- Service logger : sauvegarde dans la base de données les exceptions générées pendant l'exécution

de l'application.

-Service d'archivage : a pour objectif d'archiver les anciens messages.

La figure 3.1 montre bien l'architecture de notre processus.

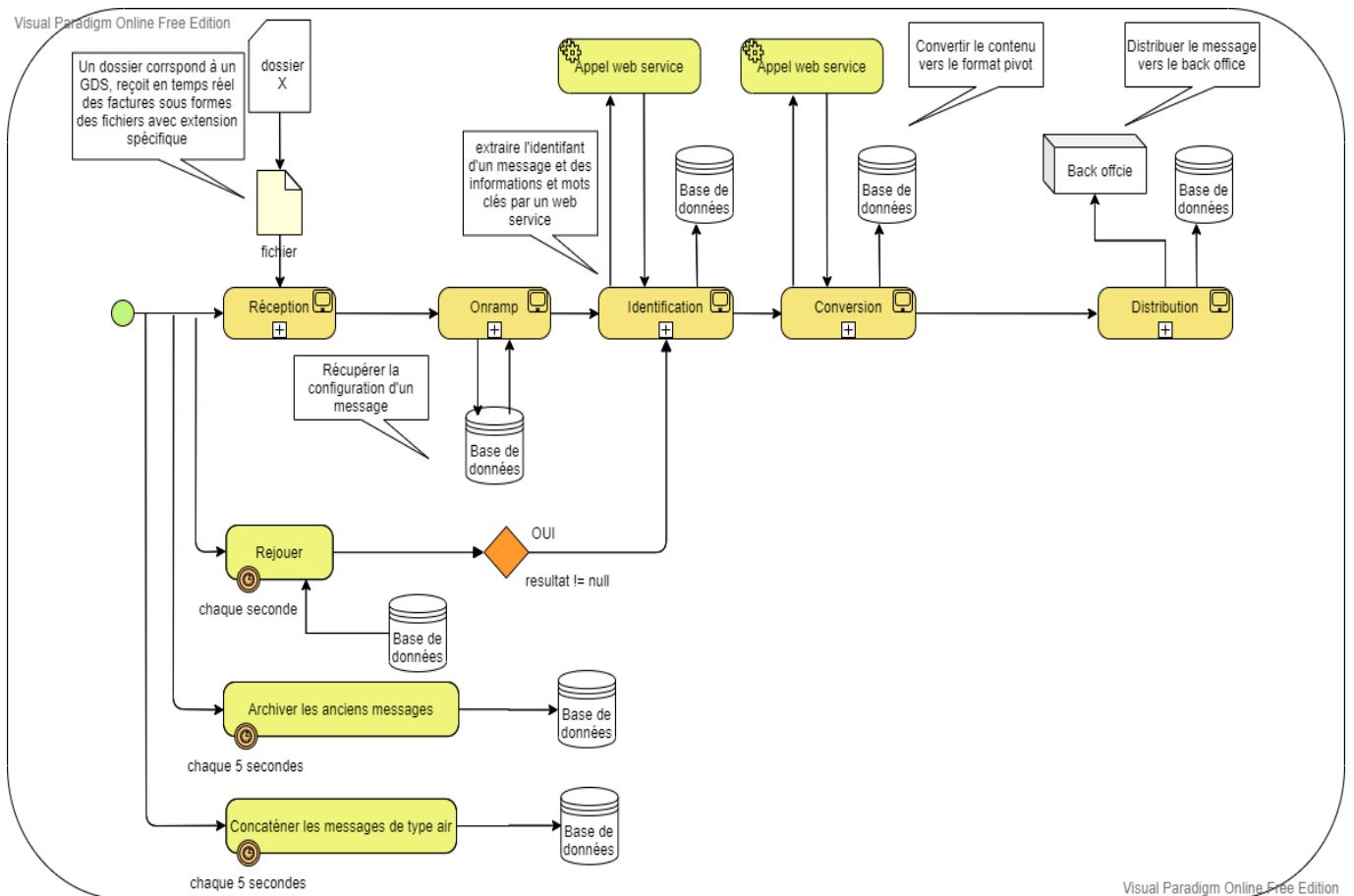


FIGURE 3.1 – Architecture globale de l'application.

Après avoir exposé une vue globale sur l'architecture de notre application, nous présenterons dans cette partie, l'architecture interne de nouvelle solution qui se compose de 6 paquetages représentés par le diagramme de paquetages de la figure 3.2.

Le diagramme de paquetage est une représentation simplifier des dépendances d'importation et d'accès entre les paquetages, les classes, les composants et les autres éléments nommés de notre système. Un paquetage est un ensemble d'éléments UML logiquement liés[13].

- fr.iga.rmess.routes** : Contient des sous paquetages dont chacun contient la route qui représente l'intermédiaire entre deux services. Chaque route manipule un processeur spécifique.

- fr.iga.rmess.process** : Présente des sous paquetages dont chaque paquetage contient un processeur propre à un service, puisqu'il communique à chaque fois avec la base de données

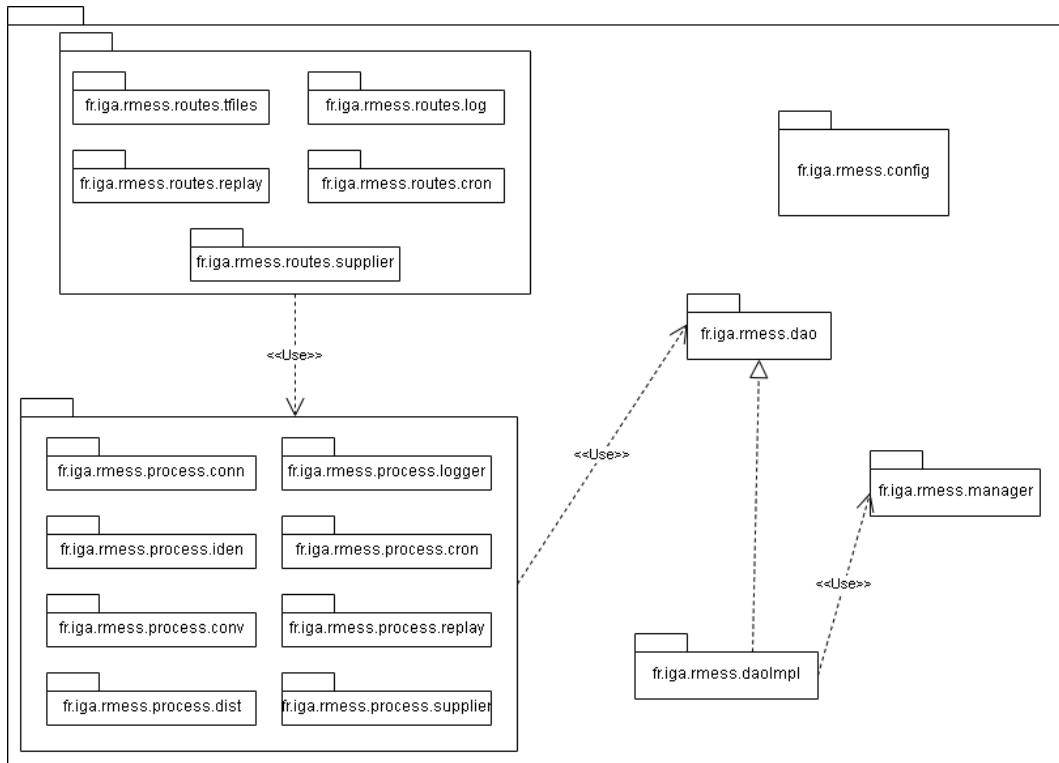


FIGURE 3.2 – Diagramme de paquetage.

il devrait passer par les interfaces dao de l'application.

- **`fr.iga.rmess.dao`** : Encapsule les entêtes des différents méthodes utilisés pour communiquer avec la base de données.
- **`fr.iga.rmess.daoImpl`** : Implémente les méthodes de la chouche dao.
- **`fr.iga.rmess.config`** : Contient des configurations supplémentaires nécessaires.

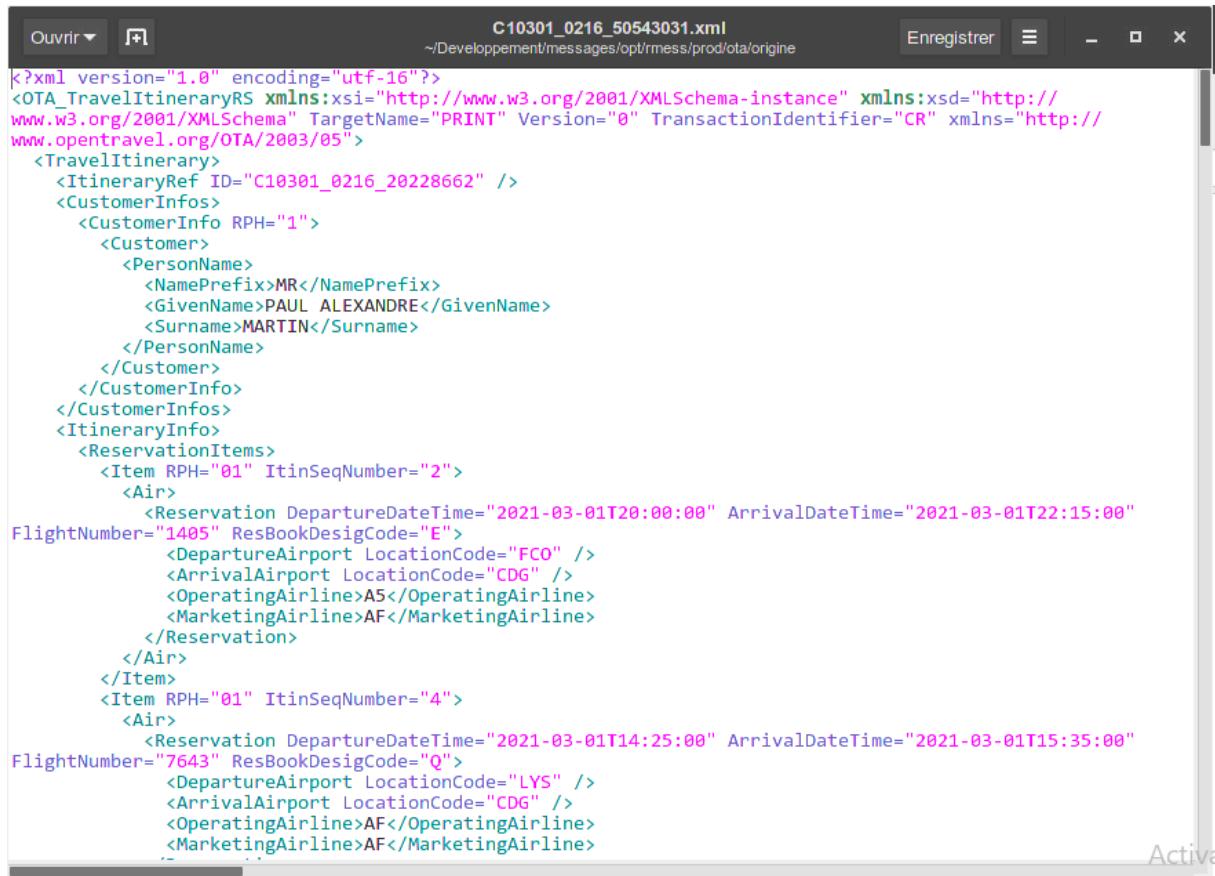
3.1.2 Conception des données

L'application Rmess traite plusieurs format de fichiers, les deux figures 3.3 et 3.4 présente deux exemples de fichiers, la première, qui correspond au GDS ota, de format xml et l'autre, qui correspond au GDS tms, de format tms.

les différents types et formats des messages sont discutés en détail dans la partie réalisation.

Notre application communique avec la base de données à travers des tables qui sont partagés avec d'autres application comme l'application IHM qui intervient dans le remplissage de quelques tables à travers des interfaces web.

La figure 3.5 illustre le diagramme de classe complet de l'application RMESS.

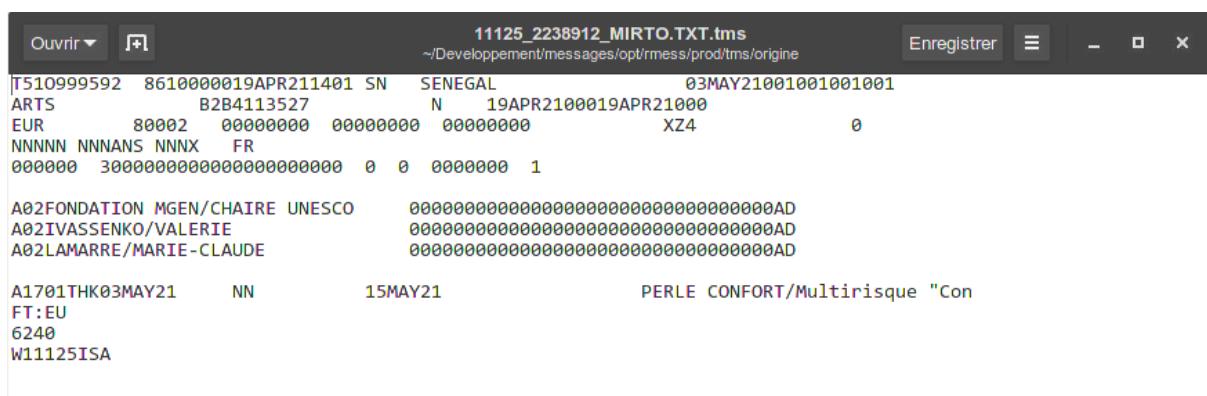


```

<?xml version="1.0" encoding="utf-16"?>
<OTA_TravelItineraryRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" TargetName="PRINT" Version="0" TransactionIdentifier="CR" xmlns="http://www.opentravel.org/OTA/2003/05">
    <TravelItinerary>
        <ItineraryRef ID="C10301_0216_20228662" />
        <CustomerInfos>
            <CustomerInfo RPH="1">
                <Customer>
                    <PersonName>
                        <NamePrefix>MR</NamePrefix>
                        <GivenName>PAUL ALEXANDRE</GivenName>
                        <Surname>MARTIN</Surname>
                    </PersonName>
                </Customer>
            </CustomerInfo>
        </CustomerInfos>
        <ItineraryInfo>
            <ReservationItems>
                <Item RPH="01" ItinSeqNumber="2">
                    <Air>
                        <Reservation DepartureDateTime="2021-03-01T20:00:00" ArrivalDateTime="2021-03-01T22:15:00"
FlightNumber="1405" ResBookDesigCode="E">
                            <DepartureAirport LocationCode="FCO" />
                            <ArrivalAirport LocationCode="CDG" />
                            <OperatingAirline>A5</OperatingAirline>
                            <MarketingAirline>AF</MarketingAirline>
                        </Reservation>
                    </Air>
                </Item>
                <Item RPH="01" ItinSeqNumber="4">
                    <Air>
                        <Reservation DepartureDateTime="2021-03-01T14:25:00" ArrivalDateTime="2021-03-01T15:35:00"
FlightNumber="7643" ResBookDesigCode="Q">
                            <DepartureAirport LocationCode="LYS" />
                            <ArrivalAirport LocationCode="CDG" />
                            <OperatingAirline>AF</OperatingAirline>
                            <MarketingAirline>AF</MarketingAirline>
                        </Reservation>
                    </Air>
                </Item>
            </ReservationItems>
        </ItineraryInfo>
    </TravelItinerary>

```

FIGURE 3.3 – Example de fichier de type ota (format xml).



Field	Value	Field	Value
T510999592	8610000019APR211401 SN	SENEGAL	03MAY21001001001001
ARTS	B2B4113527	N	19APR2100019APR21000
EUR	80002 00000000 00000000 00000000	XZ4	0
NNNN NNNANS NNNX	FR		
000000 30000000000000000000000000000000	0 0 000000 1		
A02FONDATION MGEN/CHAIRE UNESCO	00000000000000000000000000000000AD		
A02IVASSENKO/VALERIE	00000000000000000000000000000000AD		
A02LAMARRE/MARIE-CLAUDE	00000000000000000000000000000000AD		
A1701THK03MAY21 NN 15MAY21	PERLE CONFORT/Multirisque "Con		
FT:EU			
6240			
W11125ISA			

FIGURE 3.4 – Exemple de fichier de type tms (format tms).

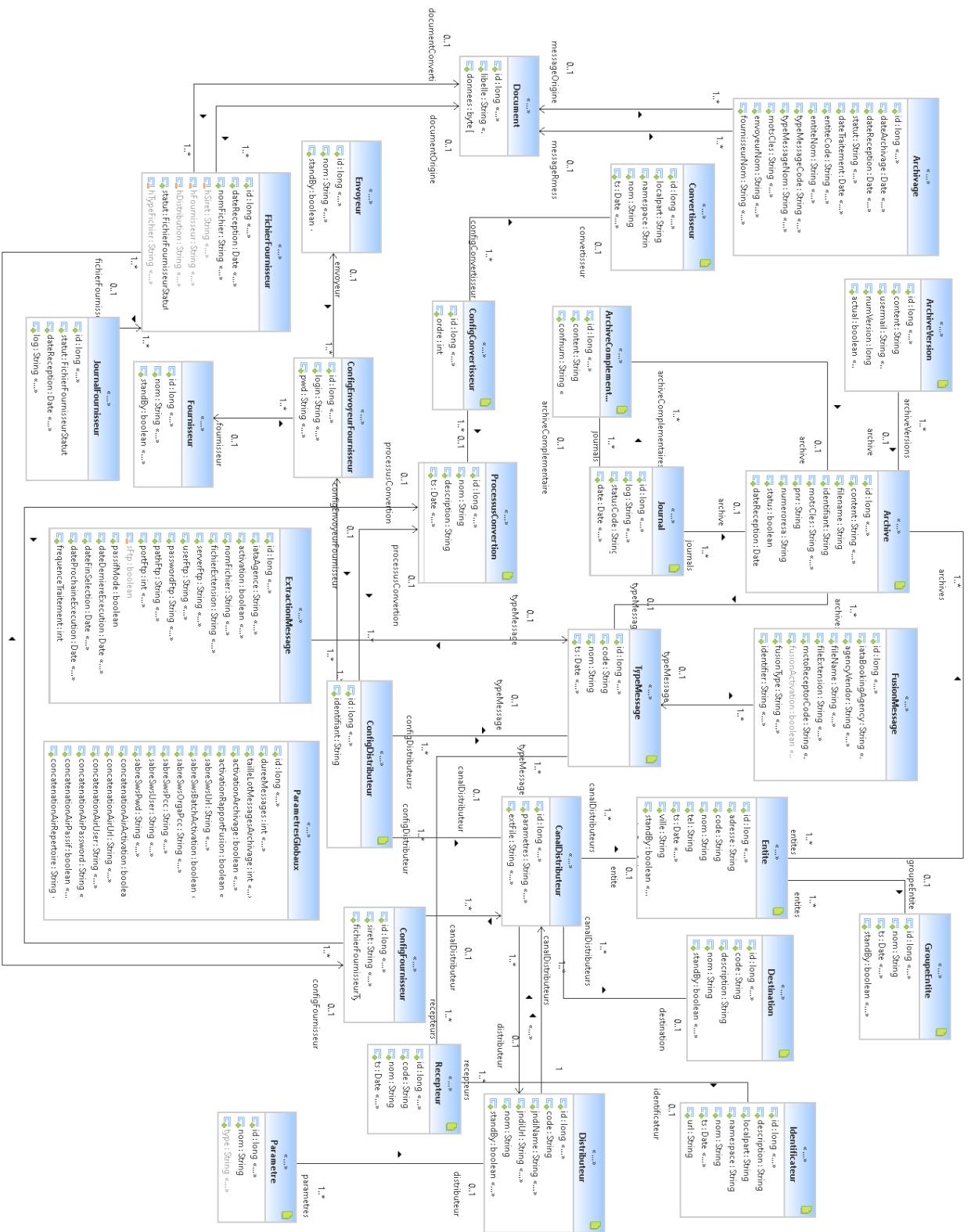


FIGURE 3.5 – Diagramme de classe de l'application
RMESS.

Les classes qui nous intéressent dans notre projet sont illustrés dans le tableau 3.1 :

Classe	Description
Archive	Contient le message original et quelques informations et mots clés ajoutés lors du traitement de ce message.
Archivage	Contient les anciens messages traités avant une certaine date.
Journal	Englobe la traçabilité du traitement des messages.
Document	Contient le contenu des fichiers qui ont le type fournisseur.
JournalFournisseur	Englobe la traçabilité du traitement des messages de type fournisseur.
ConfigFournisseur	Possède la configuration des fichiers de type fournisseur.
Identificateur	Présente les différents web services d'identification (leurs wsdl url, localpart,).
Convertisseur	Présente les différents web services de conversion (leurs wsdl url, localpart,).

Tableau 3.1: Entités et description.

Pour les autres classes on communique avec en mode lecture seulement, comme par exemple la récupération de configuration d'un récepteur ou d'un distributeur.

3.2 Conception détaillée

Dans cette partie, nous allons présenter la conception détaillée de notre application. Pour modéliser les interactions entre les différents services, nous avons utilisé les diagrammes de séquences représentés par les figures qui suivent.

3.2.1 Module de réception

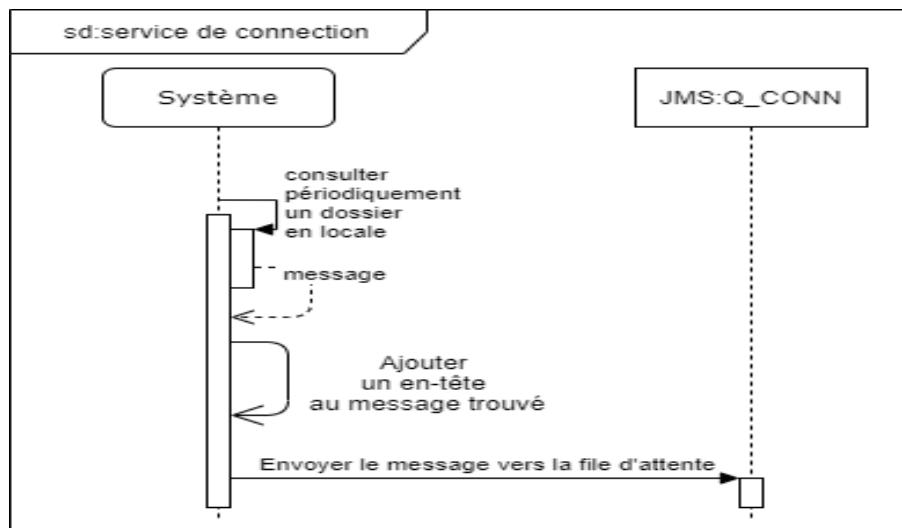


FIGURE 3.6 – Diagramme de séquence service de connection.

La figure 3.6 illustre le premier service de l'application. En fait, la réception consiste à consulter un dossier spécifique à un système globale de distribution et lorsqu'il y a un fichier dans ce dossier, un en-tête qui prend la valeur le code de récepteur qui correspond à ce type de message et après ce dernier va être transférer vers l'autre service à travers une file d'attente via un système de messagerie.

3.2.2 Module de onramp

Le service oramp dès qu'il reçoit un fichier il va récupérer sa configuration à partir de la base de données et une fois la configuration est ajoutée au message, un routeur prend la décision pour l'acheminement de ce message, soit vers l'identification spécifique ou bien l'identification standard selon la valeur de l'identificateur s'il est null ou pas.

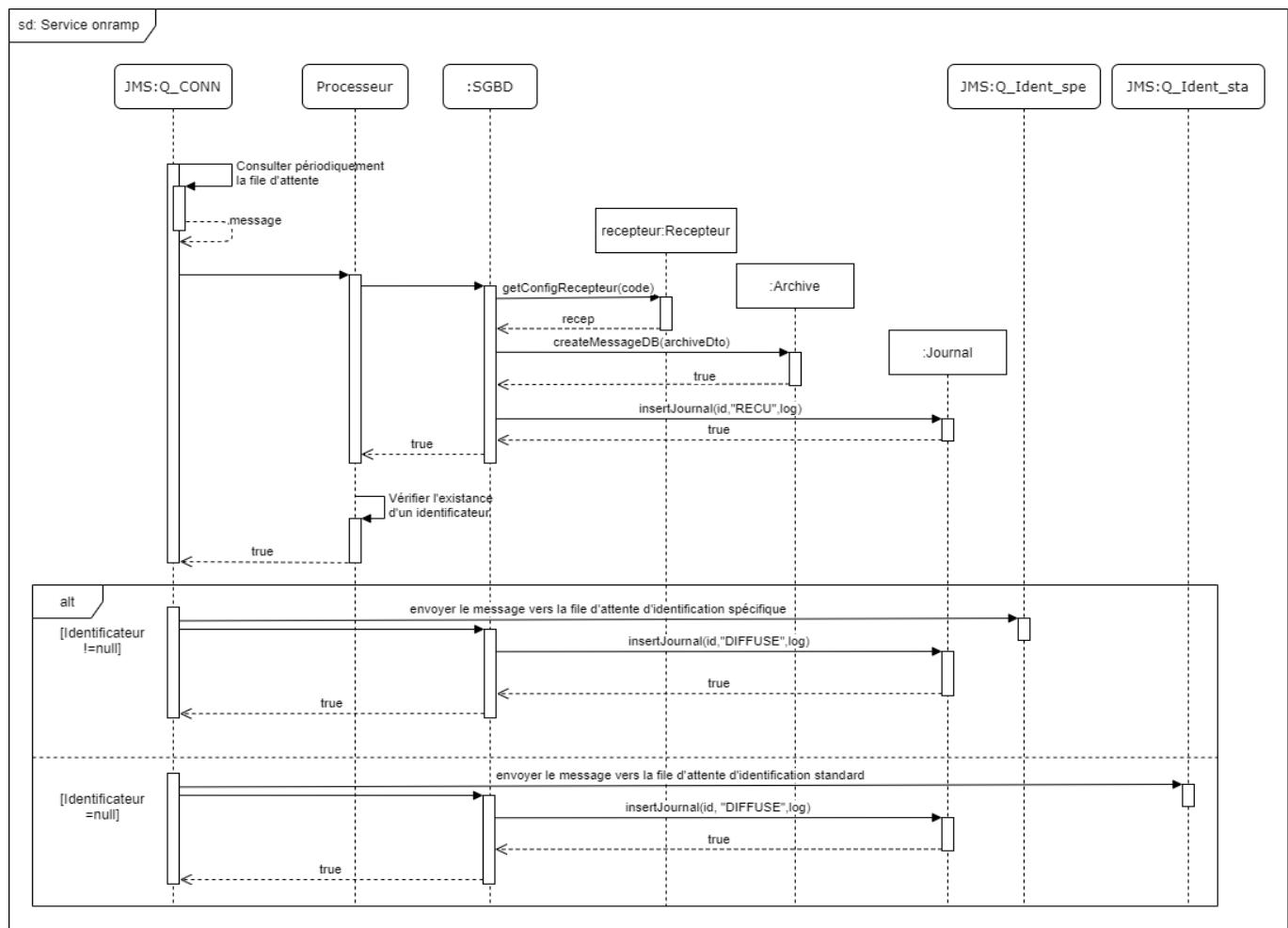


FIGURE 3.7 – Diagramme de séquence relatif à la réception et onramp.

3.2.3 Module de l'identification

La figure 3.8 présente l'identification de message. Chaque message reçu va être identifié par un web service qui correspond à un GDS spécifique. Après la récupération de résultat de WS, une mise à jour de statut de message et une insertion de log se font dans la base de données. La figure 3.9 montre le diagramme de séquence pour l'appel au web service d'identification. Il y a un autre type d'identification lorsque un message n'a pas un identificateur, donc au lieu d'un web service qui fait l'extraction des mots clés et des informations, il y a une méthode qui fait l'identification d'une manière statique.

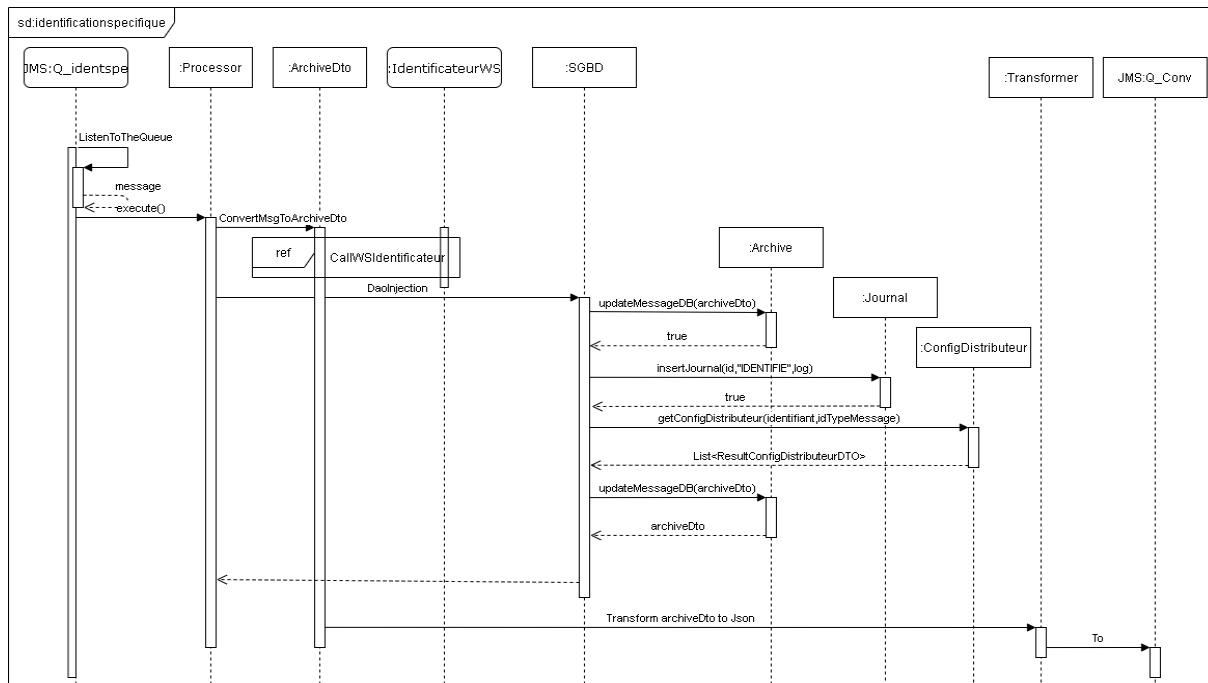


FIGURE 3.8 – Diagramme de séquence relatif à l'identification.

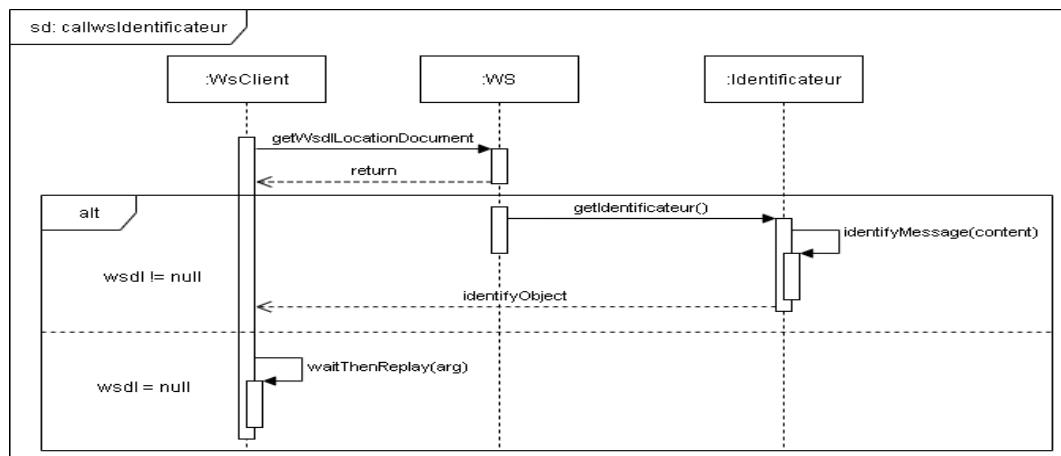


FIGURE 3.9 – Diagramme de séquence relatif à l'appel du web service.

3.2.4 Module de conversion

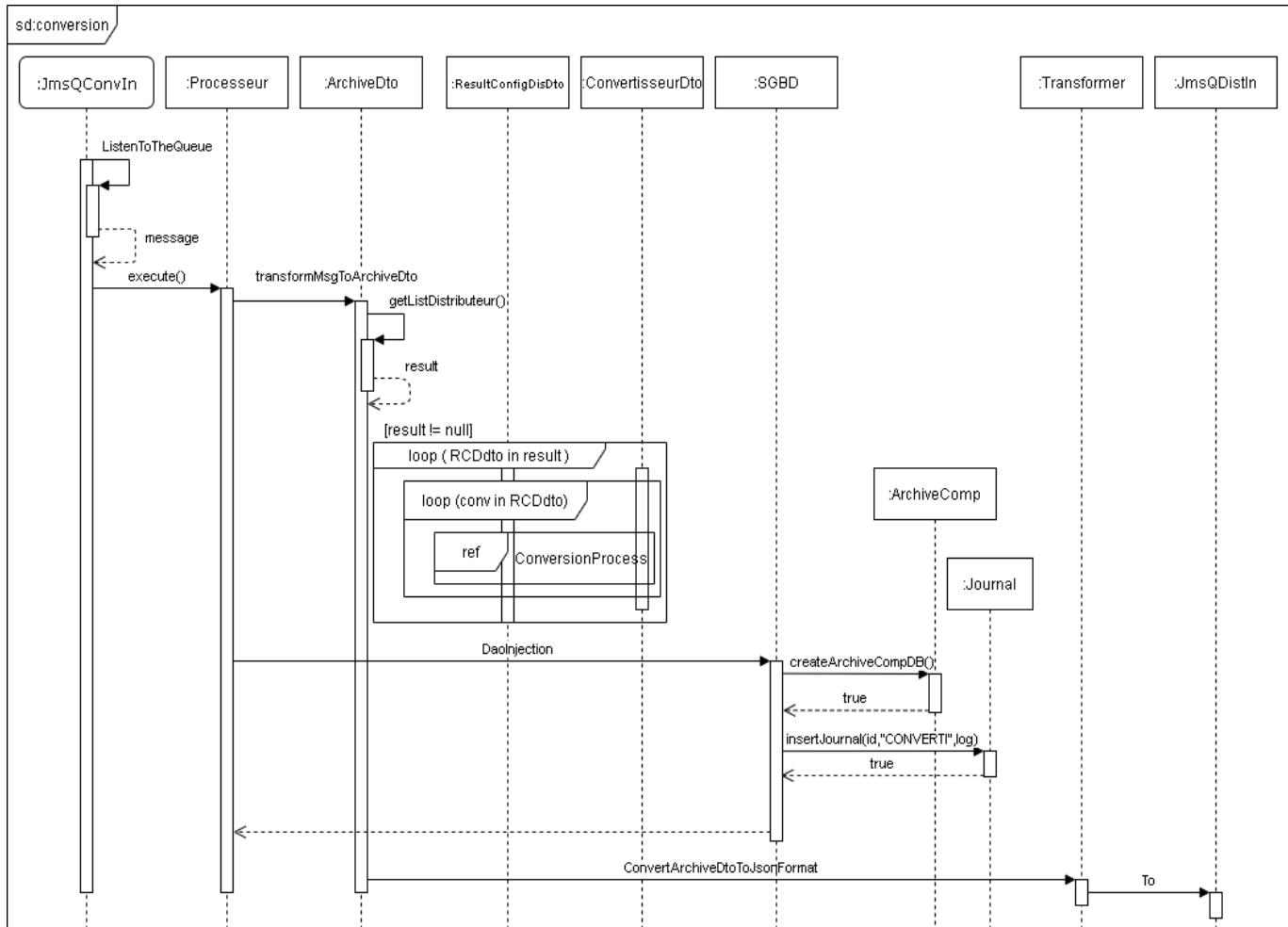


FIGURE 3.10 – Diagramme de séquence relatif au service de conversion.

Le service de conversion est illustré dans la figure 3.10. En fait, la conversion d'un message se fait sous un processus complet. Tous les messages doivent être convertis vers un même format qui est le format pivot, il y a des messages qui ont un ensemble des web services qui interviennent dans ce traitement et d'autres messages ont un seul web service qui fait la conversion, et cela dépend de ce que chaque message possède comme configuration dans la base de données.

Donc le système consulte d'une manière illimitée sa file d'attente et lorsqu'il trouve un message il le prend et exécute un processeur qui récupère tout d'abord la liste des convertisseurs pour ce message et avec un boucle il fait appel à un web service pour chaque partie de conversion. La figure 3.11 présente le sous processus de conversion qui englobe la conversion soit avec un web service qui est détaillée dans la figure 3.13 ou bien l'appel à une méthode http illustré dans la figure 3.12.

Une fois il y a une résultat le système fait une mise à jour de statut et quelques informations dans la base de données.

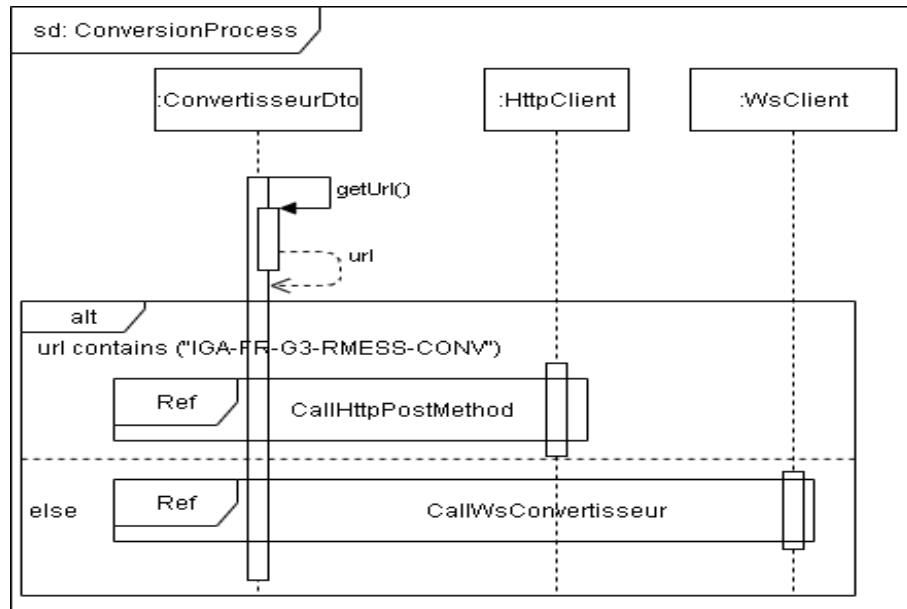


FIGURE 3.11 – Diagramme de séquence relatif au sous processus de service conversion.

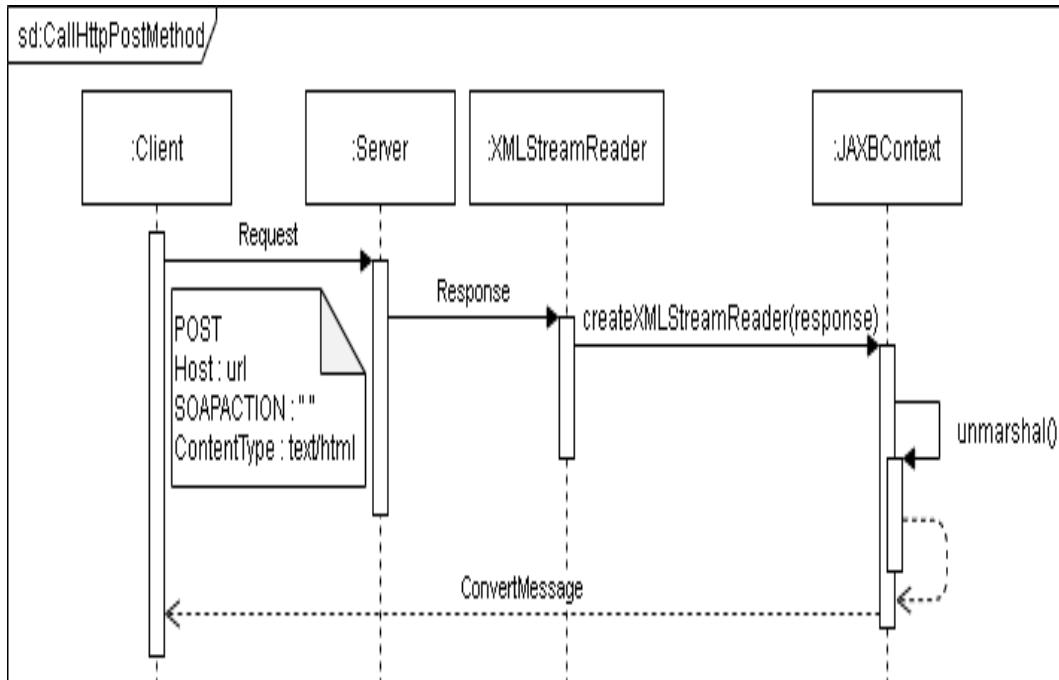


FIGURE 3.12 – Diagramme de séquence relatif à l'appel du méthode http post.

3.2.5 Module de distribution

La phase de distribution d'un message vers le back office est effectué avec plusieurs protocoles, soit l'envoi de message vers un serveur FTP ou un serveur SFTP, soit une distribution

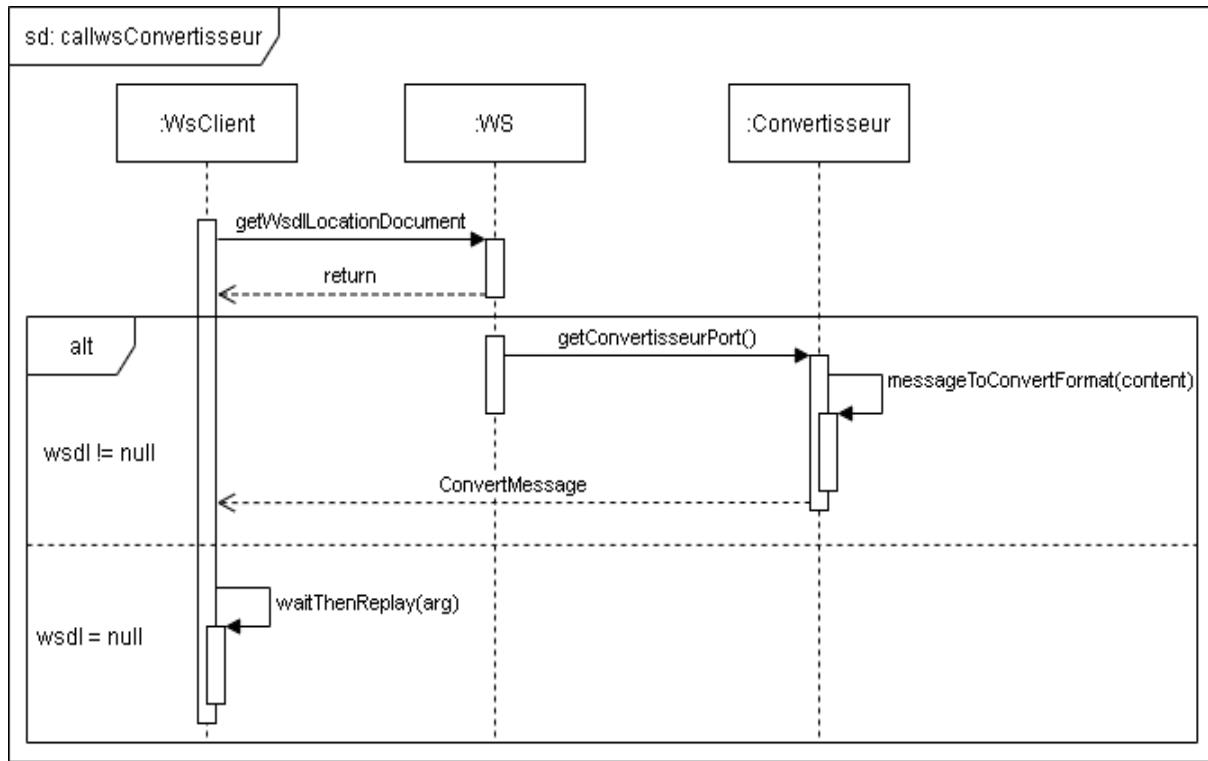


FIGURE 3.13 – Diagramme de séquence relatif à l'appel du web service de conversion.

locale, ou bien avec une méthode POST http vers un serveur http.

Un message peut être distribué avec un seul protocole comme il peut être distribué avec plusieurs et cela tout dépend de son configuration dans la base de données.

Après la réception d'un message vers le back office, une mise à jour de table Journal se fait pour chaque méthode de distribution.

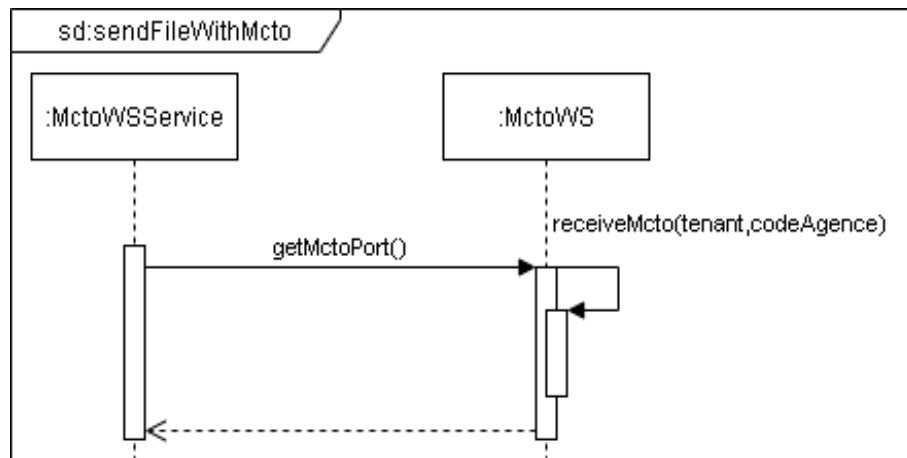


FIGURE 3.14 – Diagramme de séquence relatif à l'appel du web service de distribution avec MCTO.

La figure 3.15 montre le diagramme de séquence détaillée pour le service de distribution et le diagramme de la figure 3.14 concerne l'appel au web service de distributeur Mcto.

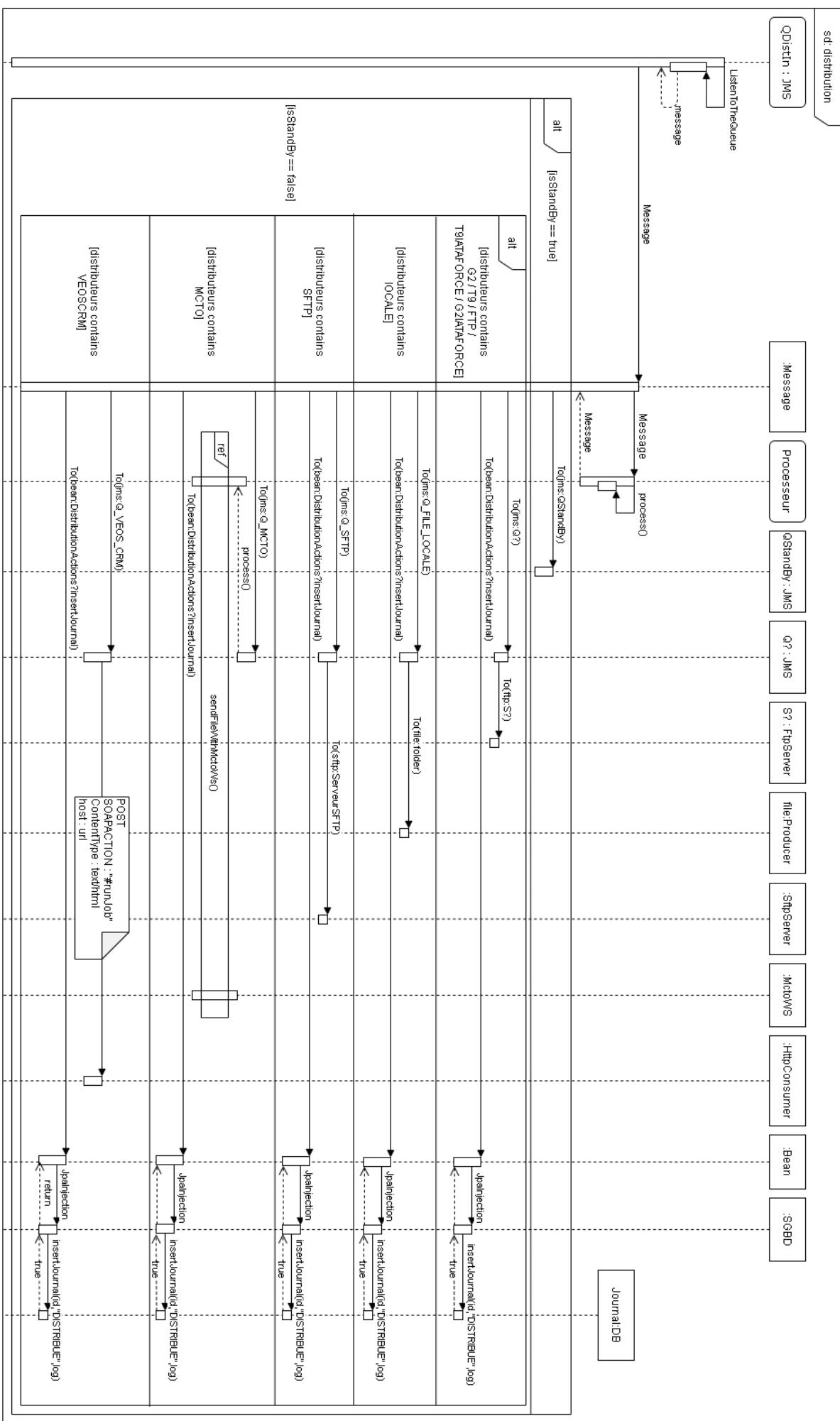


FIGURE 3.15 – Diagramme de séquence relatif au service de distribution.

La phase de distribution commence par la consultation de la file d'attente d'entrée de service de distribution et une fois il y a une détection d'un message, le système le prend et consulte ces en-têtes pour décider vers quel back office ce message doit transférer et une fois cela est terminé le système passe vers la deuxième phase de distribution, soit l'envoie de message par FTP, ou bien vers un serveur HTTP ou bien distribuer le message localement.

3.2.6 Module de replay

Rejouer un message sert à consulter la base données à chaque période de temps et une fois il y a des messages qui porte le statut "true" c'est à dire que ce message nécessite d'être rejouer et traiter de nouveau en lui envoyant vers la file d'attente de service identification. La figure 3.16 illustre le diagramme de séquence détaillé de service replay.

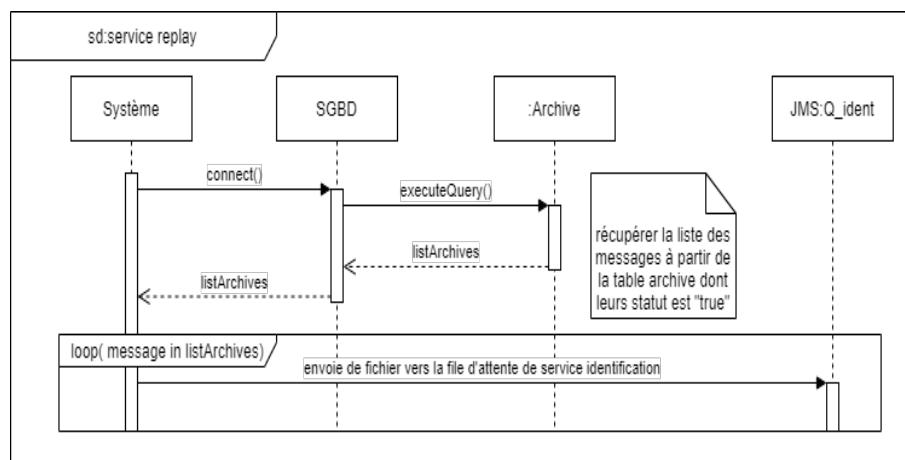


FIGURE 3.16 – Diagramme de séquence relatif à au service replay.

3.2.7 Module d'archivage

L'application propose la possibilité d'archiver les anciens messages, donc cette événement est programmé c'est à dire un programme qui se lance chaque période du temps donc le système consulte la base de données et récupère la liste des messages sauvegardés avant une certaine période et les sauvegarde dans la table Archivage de la base de données.

Le diagramme de séquence de ce service est illustré dans la figure 3.17.

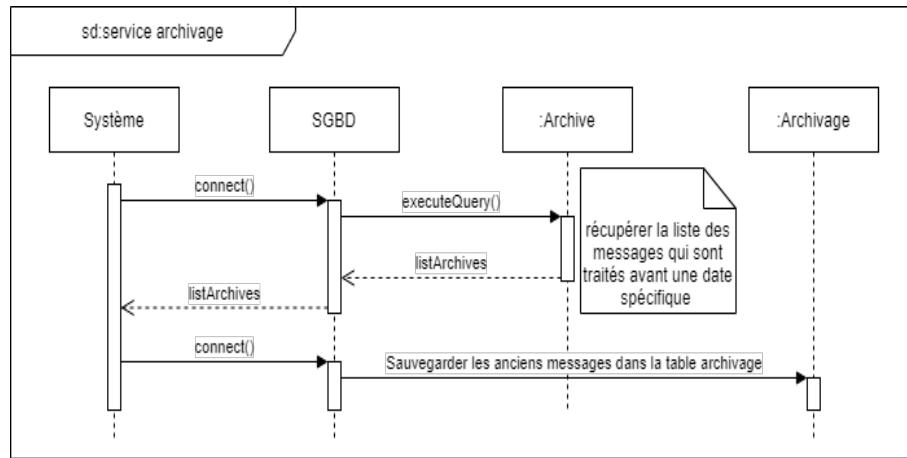


FIGURE 3.17 – Diagramme de séquence relatif à au service d’archivage.

3.2.8 Module logger

Pour assurer le bon fonctionnement ainsi que les cas où on a des erreurs, on doit avoir un service qui pour chaque exception détectée lors de l'exécution de l'application il prend cette exception et la sauvegarde dans la base de données avec l'id de message.

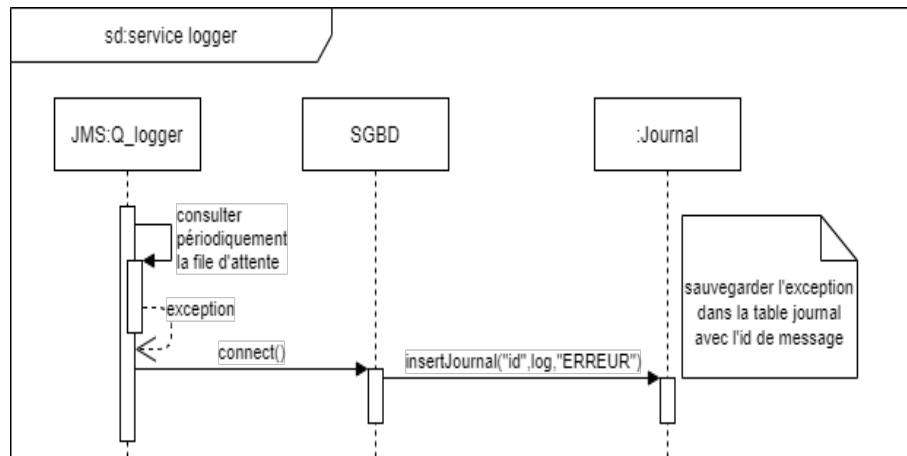


FIGURE 3.18 – Diagramme de séquence relatif à au service d’archivage.

Conclusion

Durant ce chapitre, nous avons décrit la composition dynamique et statique de notre application commençant par l'architecture globale, où nous avons présenté la conception globale, enchaînant par la conception détaillée à travers les diagrammes UML. Une réalisation des idées détaillées dans ce chapitre fera l'objet du prochain chapitre. On y justifiera le choix des outils de développement et des langages programmation, nous présenterons le travail terminé.

Chapitre 4

Réalisation

Introduction

Dans ce chapitre, nous présenterons d'abord, les techniques que nous allons utiliser pour implémenter le projet. Nous enchaînerons par présenter l'environnement de travail et les outils utilisés. Enfin, nous présenterons les scénarios d'exécution à travers quelques captures d'écran.

4.1 Environnement de travail

4.1.1 Environnement matériel

Nous montrons dans le tableau 4.1, l'ensemble des outils matériels que nous avons utilisés pour réaliser notre application.

Ordinateur	serveur dédié
Processeur	AMD Opteron(tm) Processor 6386 SE x 16
Mémoire	95 GO de RAM
Disque Dur	2 Tib
Système d'exploitation	openSUSELeap v 42.3 64bits Noyau Lunix 5.3.18

Tableau 4.1: Environnement matériel.

4.1.2 Environnement logiciel

Nous mettons ici l'accent sur les technologies globales avec lesquelles nous réaliseraisons notre projet.

- **Eclipse** : Eclipse est un environnement de développement intégré qui simplifie la programmation pour le développement de logiciels informatiques. Il est développé par IBM, est gratuit et disponible pour la plupart des systèmes d'exploitation [6].

- **Java** : Java est un ensemble de logiciels et de spécifications informatiques développé par James Gosling chez Sun Microsystems, qui a ensuite été acquis par la société Oracle, qui fournit un système pour développer des logiciels d'application et les déployer dans un environnement informatique multiplateforme [7].

- **Maven** : Maven est un outil puissant de gestion de projet qui est basé sur le POM (modèle d'objet de projet). En bref, nous pouvons dire que maven est un outil qui peut être utilisé pour la construction et la gestion de tout projet basé sur Java. maven facilite le travail quotidien des développeurs Java et aide généralement à la compréhension de tout projet basé sur Java [2].

- **Apache Activemq** : Apache activemq est un système de messagerie open source écrit en Java. ActiveMQ offre la puissance et la flexibilité nécessaire pour prendre en charge n'importe quel cas d'utilisation de messagerie. Il fournit des fonctionnalités pour les entreprises comme de simplifier et encourager la communication des applications par messages. Cette outil a simplifié la communication des différents services de l'application, il offre la possibilité de communication en mode asynchrone et ça ce que nous avons besoin pour envoyer et stocker temporairement des messages avec ces informations supplémentaires. [1].

- **Postgresql** : PostgreSQL est un système de base de données objet relationnel puissant et open source qui utilise le langage SQL combiné à de nombreuses fonctionnalités qui stockent et adaptent en toute sécurité les charges de travail de données les plus complexes[9].

PostgreSQL a acquis une bonne réputation pour son architecture éprouvée, sa fiabilité, l'intégrité des données , son ensemble de fonctionnalités, son extensibilité et le dévouement de la communauté open source derrière le logiciel pour fournir des solutions performantes et innovantes[9].

- **Hibernate** : Hibernate est un framework de mapping objet relationnel (ORM) qui sert à simplifier la programmation de base de données pour les développeurs, réduire le temps de développement et augmenter la productivité[12].

- **Apache camel** : Apache camel est un framework open source d'intégration léger écrit en java, qui supporte la plupart des patterns d'intégration d'entreprise.

Nous allons présenté dans la section qui suivre une étude préalable de l'outil apache camel.

4.2 Apache Camel : framework d'intégration

4.2.1 Présentation

Apache camel est un framework open source d'intégration léger écrit en java, qui supporte la plupart des patterns d'intégration d'entreprise.

Apache camel réalise la transformation, l'agrégation, l'enrichissement et le routage des messages entre les applications. Il propose aussi un large nombre de connecteurs afin de pouvoir s'interfacer avec plusieurs protocoles et technologies. Essentiellement, il nous permet de connecter des systèmes ou des microservices via des routes. Ces routes acceptent les messages qui peuvent être de n'importe quel type de données.

L'infrastructure des routes peut être déployée dans plusieurs applications java autonomes et sur plusieurs serveurs d'applications comme wildfly et Tomcat. Cette infrastructure peut être considérée comme une infrastructure d'intégration[5].

4.2.2 Architecture

Comme le montre la figure , Camel est composé de processeurs, de composants et de routes. Tous ces éléments sont contenus dans le CamelContext.

Le moteur de routage utilise des routes comme spécifications pour l'acheminement des messages. Les routes sont définies à l'aide de l'un des langages spécifiques au domaine (DSL) de Camel. Les processeurs sont utilisés pour transformer et manipuler les messages pendant le routage et également pour implémenter tous les modèles EIP, qui ont des mots-clés correspondants dans les langages DSL et les composants sont les points d'extension dans Camel pour ajouter de la connectivité à d'autres systèmes. Pour exposer ces systèmes au reste de Camel, les composants fournissent une interface de point de terminaison (les Endpoints en anglais)[5].

La figure 4.1 illustre l'architecture complet de Apache Camel.

4.2.3 Modèle de messagerie

Dans Camel, il existe deux abstractions pour la modélisation des messages, qui sont le Message et le Exchange.

Nous commencerons par examiner la première notion “Message” pour comprendre comment les données sont modélisées et transportées dans Camel. Ensuite, nous verrons comment une conversation est modélisée dans Camel par le Exchange[5].

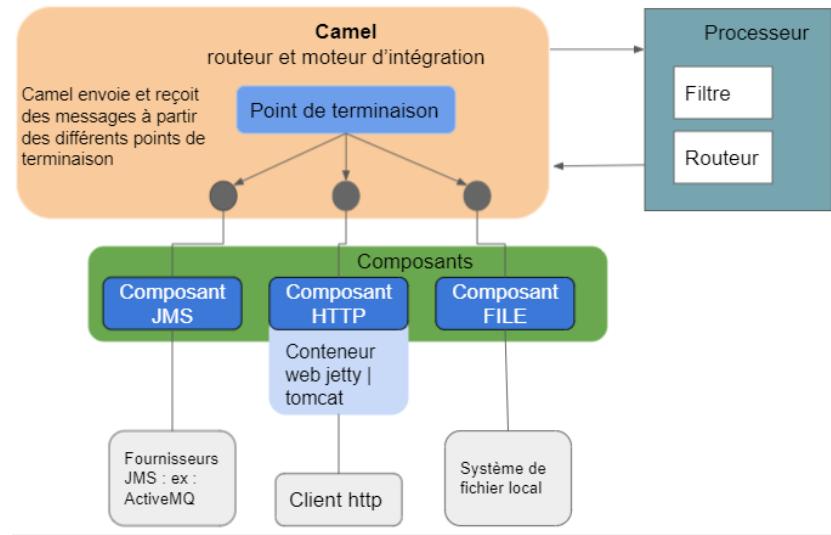


FIGURE 4.1 – Architecture de Apache camel.

[16]

4.2.3.1 Message

Les messages sont les données qui sont transférées à l'intérieur d'une route Camel. Un message peut avoir un corps, qui correspond aux données elles-mêmes, des en-têtes, qui sont des entrées de type dictionnaire qui peuvent être utilisées tout au long du traitement pour faciliter le routage des données car une des avantages de cette notion est que lorsque on utilise les entêtes pour le routage des données est beaucoup mieux qu'on utilise le routage en se basant sur le corps de message et des pièces jointes facultatifs (attachment) qui sont généralement utilisés pour le service Web et les composants de messagerie.

Un aspect important à garder à l'esprit, cependant, est que le long d'une route Camel, nos messages sont modifiés lorsque nous convertissons le corps avec un convertisseur, et lorsque cela se produit, nous perdons toutes nos en-têtes. Ainsi, les en-têtes de message doivent être considérés comme des données éphémères, qui ne seront pas utilisées tout au long de la route. Pour ce type de données, il est préférable d'utiliser les propriétés Exchange.

Le corps du message peut être composé de plusieurs types de données, tels que des binaires, JSON, etc. Les messages sont les entités utilisées par les systèmes pour communiquer entre eux lors de l'utilisation de canaux de messagerie. Les messages circulent dans un seul sens d'un expéditeur à un destinataire :

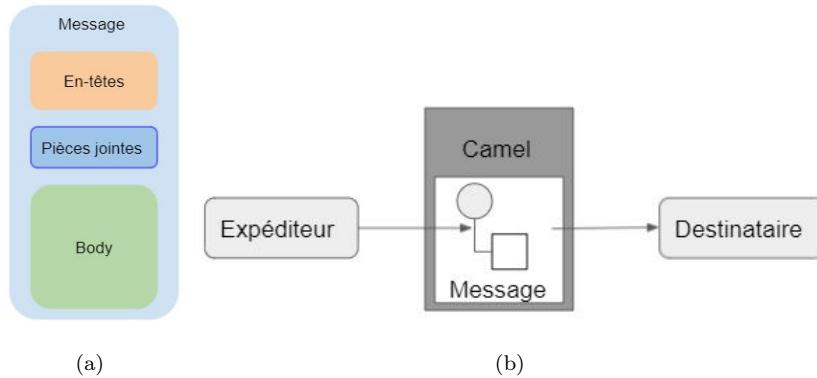


FIGURE 4.2 – (a) Contenu message Camel (b) Communication expéditeur destinataire

[4]

4.2.3.2 Exchange

Un Exchange dans Camel est le conteneur du message lors du routage, prend également en charge les différents types d'interactions entre les systèmes, également connus sous le nom de modèles d'échange de messages (MEP). Les MEP sont utilisés pour différencier les styles de messagerie unidirectionnelle et demande-réponse[5].

Un Exchange de Camel a une caractéristique de modèle qui peut être :

- **InOnly** : un message unidirectionnel (également appelé message d'événement). Par exemple, la messagerie JMS est souvent une messagerie unidirectionnelle.
- **InOut** : Un message de demande-réponse. Par exemple, les transports basés sur HTTP sont souvent une réponse de demande, où un client demande à récupérer une page Web, en attendant la réponse du serveur.

Un "Exchange" peut contenir comme le montre la figure 4.4, deux messages, un représentant l'entrée et un autre représentant la sortie d'une intégration. Le message de sortie de Camel est facultatif car nous pourrions avoir une intégration qui n'a pas de réponse.

L'interface Exchange fournit une abstraction pour un échange de messages, c'est-à-dire un message de demande et son message de réponse ou d'exception correspondant.

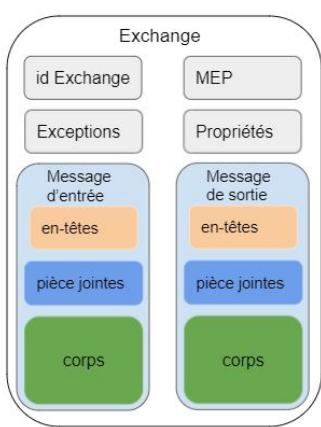
Dans la terminologie Camel, les messages de demande, de réponse et d'exception sont des messages d'appel, de sortie et d'erreur.

Un Exchange contient les éléments suivants :

- **ID d'échange** : Un identifiant unique qui identifie l'échange. Camel générera un identifiant unique par défaut, si on ne le définit pas explicitement un.

FIGURE 4.3 – Exchange Camel

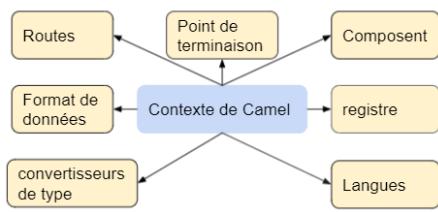
[4]



- **MEP** : un modèle qui indique si on utilise le style de messagerie InOnly ou InOut. Lorsque le modèle est InOnly, l'Exchange contient un message in. Pour InOut, un message de sortie existe également qui contient le message de réponse pour l'appelant.
- **Exception** : Si une erreur se produit à tout moment pendant le routage, une Exception sera définie dans le champ d'exception.
- **Propriétés** : similaires aux en-têtes de message, mais elles durent pendant toute la durée de l'échange. Les propriétés sont utilisées pour contenir des informations de niveau global, tandis que les en-têtes de message sont spécifiques à un message particulier. Camel lui-même ajoutera diverses propriétés à l'échange pendant le routage.
- **Message d'entrée** : C'est le message d'entrée, qui est obligatoire. Le message in contient le message de demande.
- **Message de sortie** : Il s'agit d'un message facultatif qui n'existe que si le MEP est InOut. Le message de sortie contient le message de réponse[5].

4.2.4 Les concepts de Camel

4.2.4.1 Camel context



Camel context est un conteneur qui permet d'accéder à tous les autres services de camel par exemple le endpoint, component, route ... etc[5].

FIGURE 4.4 – Camel context

[4]

4.2.4.2 Route

Les routes dans Camel sont le cœur de traitement des données. Il se compose d'un flux, qui commence sur un point de terminaison, passe par un flux de processeurs/convertisseurs et se termine sur un autre point de terminaison, on peut dire donc qu'une route possède deux points de terminaison, l'une d'entrée et l'autre de sortie. Il est possible d'enchaîner des routes en appelant une autre route qui commence par le point final d'une route précédente[5]. Une route commence d'abord par un consommateur qui remplit l'échange initial. A chaque étape du processeur, le message de sortie de l'étape précédente est le message d'entrée de l'étape suivante.

Chaque route dans Camel possède un identifiant unique qui est utilisé pour la journalisation, le débogage, la surveillance et le démarrage et l'arrêt des routes[5].

À la fin d'une route, le MEP de l'échange détermine si une réponse doit être renvoyée à l'appelant de la route. Si le MEP est InOnly, aucune réponse ne sera renvoyée. Si c'est InOut, Camel prendra le message de sortie de la dernière étape et le renverra[5].

4.2.4.3 Composant et point de terminaison

Pour pouvoir s'intégrer à différentes applications sur différentes plates-formes, Apache Camel prend en charge plusieurs composants, où chacun de ces composants permet à Apache Camel d'envoyer et de recevoir des messages à l'aide d'un certain protocole ou middleware, par exemple un courtier de messages[5].

Parmi les composants les plus communs de Camel :

- * Composant JMS : permet à Apache Camel de se connecter à n'importe quel fournisseur JMS comme le ActiveMQ.
- * Composant FILE : permet à Camel de lire des fichiers à partir d'un système de fichiers ou d'écrire des messages en tant que fichiers dans un système de fichiers.
- * Composant SEDA : qui implémente des files d'attente en mémoire pour pouvoir produire et consommer les messages de manière asynchrone.
- * Composant HTTP : ce composant a pour but de faire appel à des serveurs externes à travers le protocole http.
- * Composant CRON : qui permet de lancer des événements programmés c'est-à-dire des événements qui se répètent à chaque période précise de temps.
- * Composant SQL : permet de communiquer à une base de données via des requêtes sql[5].

4.2.5 Les avantages de Apache Camel

- Meilleure solution d'intégration légère.
- Produire et consommer à partir de n'importe quel système de messagerie.
- Conversion des types de messages.
- Grande bibliothèque de support.
- Camel s'intègre bien avec des frameworks bien connus comme Spring Boot et d'autres produits middleware, ce qui lui permet d'être déployé selon nos besoins.
- Il existe plus de 150 composants pour le framework Camel qui aident à s'intégrer à diverses plates-formes logicielles.

4.3 Scénarios d'exécution

Les entrées de notre application sont des fichiers enregistrés en temps réel dans des dossiers en locale dont chaque dossier correspond à un GDS spécifique. Le tableau suivant montre bien les différents types de message et leurs dossiers associés ainsi qu'une description.

Chemin dossier	Extension fichier	Code récepteur	Description
/opt/rmess/prod/aerticket	*.xml	AERTICKET	–
/opt/rmess/prod/amadeus_air	*.AIR	AIR	Concerne l'aérien, le maritime, la SNCF, les loueurs et les hotels.
/opt/rmess/prod/com1prod	*.com	COM1PROD	–
/opt/rmess/prod/fournisseurs	*.supp	SUPPLIER	–
/opt/rmess/prod/galileo_mir_ram	*.mir	GDS-GALILEO	Concerne l'aérien, la SNCF, les loueurs et les hotels.
/opt/rmess/prod/igf	*.igf	IGF	–
/opt/rmess/prod/mcto_brut	*.xml	MCTO	–
/opt/rmess/prod/ota	*.xml	OTA	–
/opt/rmess/prod/sabre_iur	*.iur	IUR	Concerne l'aérien, le maritime, la SNCF, les loueurs et les hotels.
/opt/rmess/prod/tms	*.tms	TMS	Concerne les agences d'assurance.

Tableau 4.2: Liste des sources de données.

Le but de notre application est de faire un traitement pour un message reçu dans n'importe quel dossier présenté dans le tableau 4.2.

La figure 4.5 montre un exemple d'un message de type amadeus_air.

Le processus de traitement d'un message nécessite donc des services pour y arriver vers la destination finale.

```

Ouvrir ▾  AIMVOYAGESPV100013275.AIR
~/Développement/messages/opt/rmess/prod/amadeus_air/origine
Enregistrer  -  ×
AIR-BLK208;7A;245;0000000000;GW3351902;001001
AMD 1900324986;1/1;
1A1120129;GW3351902;PARTQ219N;AIR
MUC1A JRVVIX003;0101;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;;;;;
JRVVIX
A-AIR FRANCE;AF 0571
B-TTP/ET
C-7906/ 0203WNSU-0203WNSU-I-0-1-F
D-210419;210419;210419
G- ;;PARPAR;FR
H-001;002ORY;PARIS ORLY ;NCE;NICE ;AF 6202 L L 24APR20210715 0840
24APR2021;OK01;HK01;;0;318;;0PC;3 ;0635 ;ET;0125 ;N;420;FR;FR;2
Y-001;002ORY;PARIS ORLY;NCE;NICE;AF;AIR FRANCE;AF;AIR FRANCE;AF;AIR FRANCE;318;N
H-002;003ONCE;NICE ;CDG;PARIS CDG ;AF 7711 E E 27APR20210630 0805
27APR2021;OK01;HK01;;0;318;;0PC;2 ;0600 ;ET;0135 ;N;432;FR;FR;2F
Y-002;003NCE;NICE;CDG;PARIS CDG;AF;AIR FRANCE;AF;AIR FRANCE;AF;AIR FRANCE;318;N
K-FEUR177.00 ;;;;;;;EUR233.77 ;;;
KFTF; EUR9.32 FR SE; EUR19.90 FR TI; EUR2.26 IZ EB; EUR3.00 04 VC; EUR17.12 QW LO; EUR5.17 UI VZ
TAX-EUR9.32 FR ;EUR19.90 FR ;EUR27.55 XT ;
L-
M-LS50MALG ;ES50MALG
N-EUR116.50;60.00
O-24APR24APR;27APR27APR;LD20APR212359
Q-PAR AF NCE116.50AF PAR60.00EUR176.50END XT2.26IZ3.000417.12QW5.17UI;FXB
I-001;01EVENE/ZOE;;APPAR 01.40.55.80.60 - AIM VOYAGES - A//E-Z.EVENE@TNDA.EU;5K002F;
SSR ADTK 1A /TO AF BY 20APR 1500 PAR OTHERWISE WILL BE XLD
OSI YY OIN FR09647
T-K057-3833774547
FM**0.00/0.00A/V0.00A;S2-3;P1
FPCH;P1
FSAXG01TUF;P1
FVAF;S2-3;P1
TKOK19APR/PARTQ219N//ETAF;P1
RMRRC/SNF/PLOIJE/29090104529944605/03JUL2020;P1
RMRLC/SNF/GV/29090109408593357;P1
ENDX

```

FIGURE 4.5 – Exemple d'un message "AIR".

4.3.1 Service de connection

Le service de connection consiste à mettre en place une route qui utilise le composant "camel-file" dont le rôle est de consulter un dossier en local pour un type de message.

Ce composant a des options qui réalisent une lecture spécifique d'un message c'est-à-dire des options qui spécifient l'extension d'un message lu, la suppression ou non d'un message de dossier après sa lecture...etc.

Dans ce service on ajoute à chaque message reçu un entête qui contient un code de récepteur et une fois l'entête est ajouté le message va être transmis vers une file d'attente jms correspond à ce type de message.

Par exemple : le message reçu de dossier amadeus_air a comme extension air, et un code de récepteur "AMADEUS", une fois le message est traité va être transmis vers la file d'attente RMESS-CONN-AMADEUS.

La figure 4.6 montre l'architecture détaillée de service de connexion.

Après la réception d'un message de n'importe quel type, ce dernier va être transféré vers la file d'attente d'entrée RMESS-ROUT-ONRAMP-IN-JMS de service Onramp.

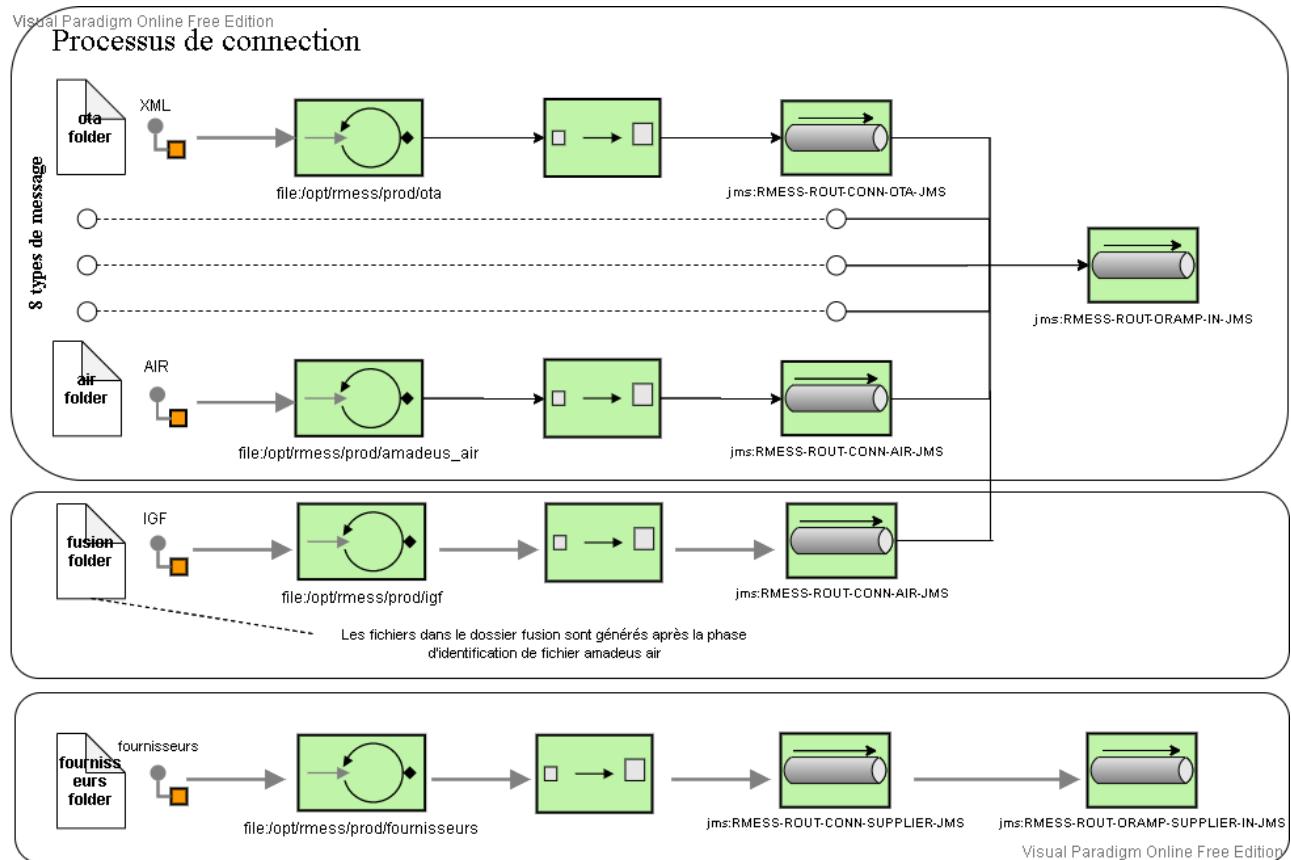


FIGURE 4.6 – Service de connection

4.3.2 Service onramp

Les messages reçus dans la file d'attente RMESS-ROUT-ONRAMP-IN-JMS sont des messages issus des différents connecteurs de notre application et le seul point de distinction pour ces types de message est l'entête ajouté dans le service de connection.

Pour chaque message on lit le code de récepteur correspondant puis on consulte la base de données et on récupère la configuration de ce récepteur.

Cette configuration va être ajoutée au contenu de message pour construire un objet archiveDto et cela nous facilite beaucoup mieux la communication avec la base de données. Après la construction de cet objet un traitement aura lieu dans le processeur de ce service :

- Ajout des données dans la table archive.
- Ajout de log dans la table journal (l'état de message ici est "RECU").

Une fois le traitement est terminé on fait une transformation de message vers un type JSON car les files d'attente jms de activemq ne supportent pas des messages POJO.

Le résultat est transmis vers la file d'attente de sortie de service onramp.

La figure 4.7 illustre le service onramp de notre application.

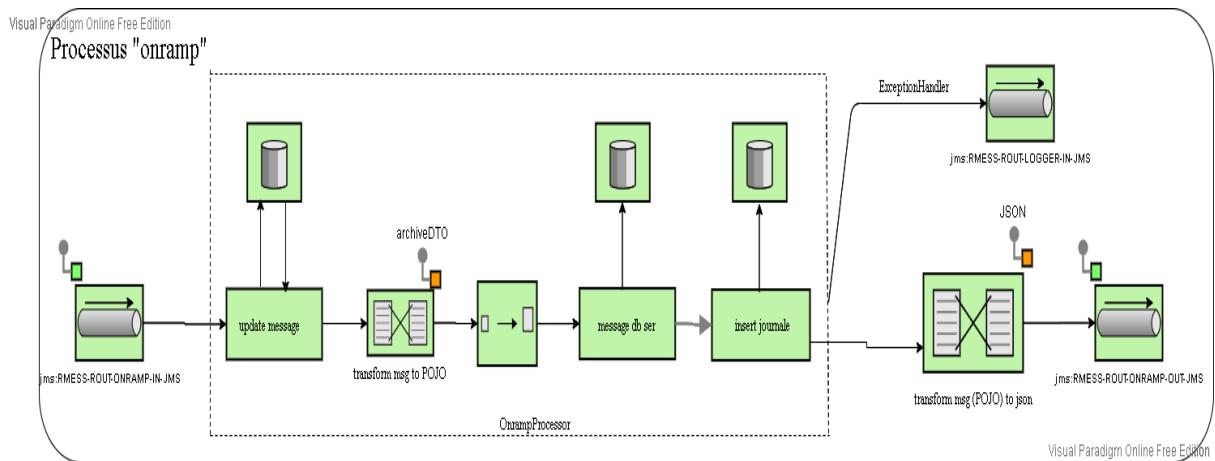


FIGURE 4.7 – Service onramp

AIMVOYAGESPV100013275.AIR
~\Développement\messages\onramp

```
{"archive": "AIR-BLK208;7A;;245;0000000000;GW3351902;001001\nAMD 1900324986;1/1;
\n1A1120129;GW3351902;PARTQ219N;AIR\nMUC1A
JRVVIX003;0101;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;;;;;;
AF JRVVIX
\nA-AIR FRANCE;AF 0571\nB-TPP/ET\nC-7906/ 0203WNSU-0203WNSU-I-0-1-F\nD-210419;210419;210419\nG- ;;PARPAR;FR
\nH-001;0020ORY;PARIS ORLY ;NCE;NICE ;AF 6202 L L 24APR20210715 0840
24APR2021;OK01;HK01;;0;318;;0PC;3 ;0635 ;ET;0125 ;N;420;FR;FR;2 \nY-001;0020ORY;PARIS ORLY;NCE;NICE;AF;AIR FRANCE;AF;AIR
FRANCE;AF;AIR FRANCE;318;N\nH-002;0030NCE;NICE ;CDG;PARIS CDG ;AF 7711 E E 27APR20210630 0805
27APR2021;OK01;HK01;;0;318;;0PC;2 ;0600 ;ET;0135 ;N;432;FR;FR;2F\nY-002;003NCE;NICE;CDG;PARIS CDG;AF;AIR FRANCE;AF;AIR
FRANCE;AF;AIR FRANCE;318;N\nK-FEUR177.00 ;;;;;;;EUR233.77 ;;;;\nKFTF; EUR9.32 FR SE; EUR19.90 FR TI;
EUR2.26 IZ EB; EUR3.00 04 VC; EUR17.12 QW LO; EUR5.17 UI VZ\nTAX-EUR9.32 FR ;EUR19.90
FR ;EUR27.55 XT ;\nL-\nM-LS50MALG ;ES50MALG \nN-EUR116.50;\n0.00\no-24APR24APR;27APR27APR;LD20APR212359\nQ-
PAR AF NCE116.50AF PAR60.00EUR176.50END XT2.26IZ3.000417.12QW5.17UI;FXB\nI-001;01EVENE/ZOE;APPAR 01.40.55.80.60 - AIM
VOYAGES - A/E-Z.EVENE@TNDA.EU;5K002F;\nSSR ADTK 1A /TO AF BY 20APR 1500 PAR OTHERWISE WILL BE XLD\nOIN FR09647
\nT-K057-3833774547\nFM*C*0.00/0.00A/V0.00A/S2-3;P1\nFPCH;P1\nFSAXG01TUF;P1\nFVAF;S2-3;P1\nTKOK19APR/PARTQ219N//ETAF;P1
\nRMRC/SNF/PLOIJE/29090104529944605/03JUL2020;P1\nRMRLC/SNF/GV/29090109408593357;P1\nENDX\n
\f", "idArchive": 421221, "codeTypeMessage": null, "motsCles": null, "identifiant": null, "currentLog": "", "filename": "AIMVOYAGESPV100013275.AIR", "identificateur": {"nom": "AMADEUS AIR IDENTIFIER", "namespace": "http://ident.rmess.g3.iga.fr", "url": "http://localhost:8081/rmess-ident-air/Identificateur?wsdl"}, "localpart": "IdentificateurService", "typeMessage": {"AMADEUS_GDS_AIR": {"id": 4, "code": "AMADEUS_GDS_AIR", "nom": "AMADEUS AIR VIA AMADEUS"}}, "recepCode": "AIR", "idConvertisseur": null, "versionArchive": "ORIGINAL"}
```

Texte brut ▾ Largue des tabulations : 8 ▾ Lig 1, Col 1116 ▾ INS

FIGURE 4.8 – Résultat service onramp

La figure 4.8 montre le résultat obtenu après le service onramp ; le contenu est de format json qui contient le message et les configurations récupérés à partir de la base de données.

4.3.3 Service d'identification

Les connecteurs externes "Identificateur" servent à identifier un message. Ils permettent :

- D'identifier le type de message (exemple : AIR, IUR et etc...).
- D'extraire l'identifiant du destinataire (exemple dans un message comptable le IATA ou autre).
- D'extraire des informations clés pour faciliter la recherche du message par un chargé de clientèle.
- Et enfin logger le traitement de l'analyse.

Une fois le message est reçu dans la file d'attente de sortie de service onramp il va être transmis directement vers la file d'attente d'identification RMESS-ROUT-IDENT-IN-JMS puis il est transformé de format json vers un objet archiveDto pour qu'on puisse appliquer un processeur de camel qui fait appel à des méthodes et des traitements dépendant au contenu des attributs de cet objet.

L'objet archiveDto contient comme attribut un objet identificateur qui va distinguer le type d'identification d'un message.

Si l'identificateur est null il s'agit donc d'une identification standard et un entête type_identification va être ajouté au message et qui va prendre la valeur “standard identification” , si non il s'agit d'une identification spécifique et un entête type_identification va être ajouté au message avec une valeur “specific identification”.

Selon la valeur de l'entête type_identification le message est transmis vers la file correspondante ; RMESS-TOU-IDEN-SPE-IN-JMS si l'identification de ce message est spécifique et RMESS-ROUT-IDEN-STA-IN-JMS si l'identification de ce message est standard.

La figure 4.9 montre bien la route de la première partie de service identification.

4.3.3.1 Identification spécifique

La route d'identification spécifique est toujours en état d'écoute, donc à chaque message reçu dans la file d'attente, un processeur va exécuter le traitement suivant :

-Au début il va consulter le contenu de message et va appeler un web service d'identification dont l'url est l'un des attributs de archiveDto.

-Une fois l'appel au web service est terminé le processeur continu à faire d'autres traitements (la mise à jour de message, l'insertion des données de log dans la table journal de base de données et la récupération de la liste de distributeurs de ce message de la base de données) puis le message va être transformé de nouveau vers le format json pour terminer par l'envoi de

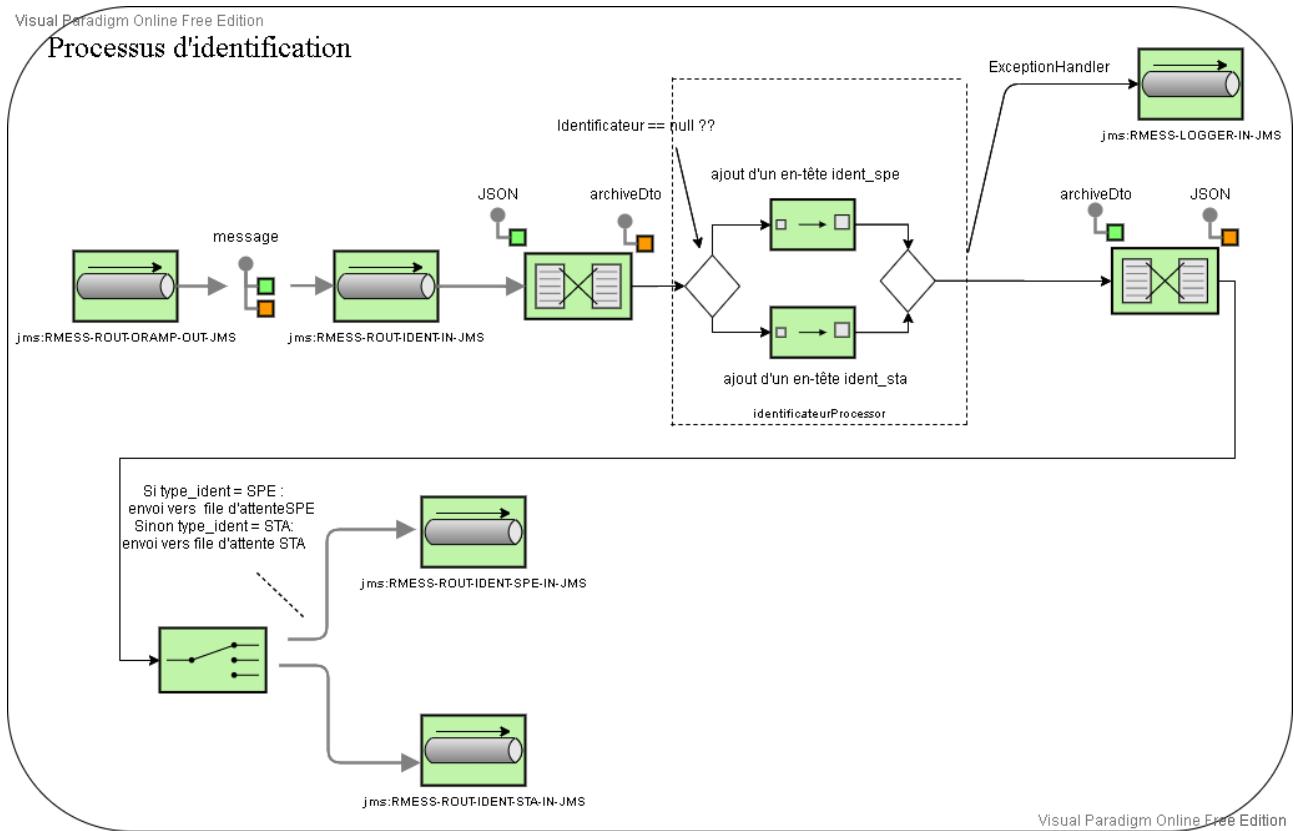


FIGURE 4.9 – Service d'identification.

résultat vers la file d'attente de sortie de service identification spécifique.

La partie de l'identification spécifique est illustrée dans la figure 4.10.

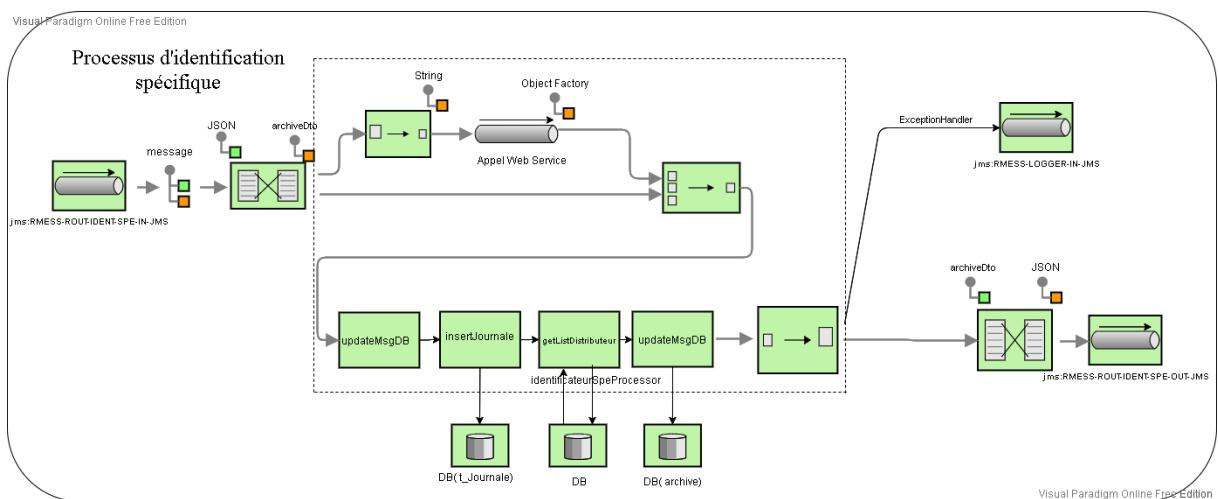


FIGURE 4.10 – Identification spécifique.

4.3.3.2 Identification standard

L'autre type d'identification est une identification standard c'est-à-dire sans appel à un web service. Pour chaque message reçu, on a exécution de traitement suivant :

- Identifier le message avec une simple méthode.
- Mettre à jour la table Archive.
- Insérer les logs dans la table Journal.

La figure suivante montre bien la conception de service d'identification standard :

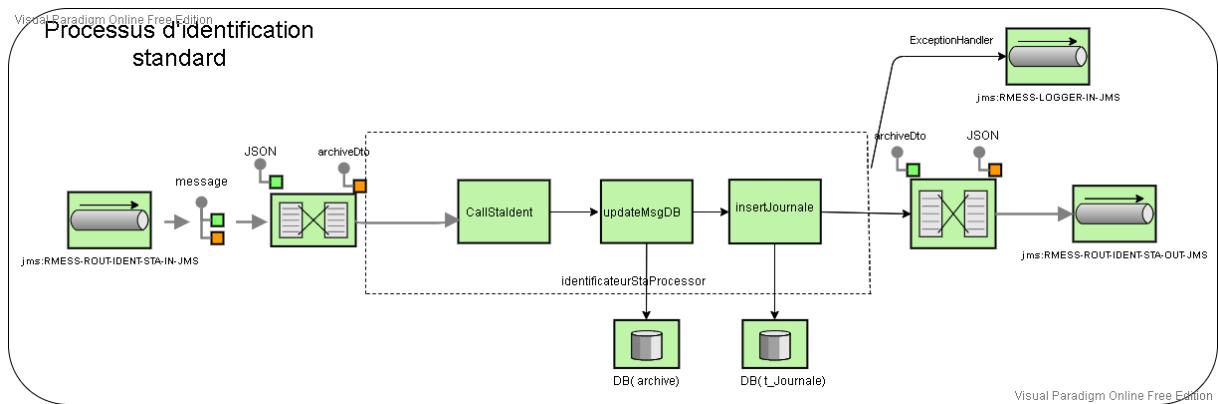


FIGURE 4.11 – Identification standard.

4.3.4 Service de conversion

Il s'agit d'un appel à des web app permettant la transformation d'un message entrant en un format de sorti spécifique (AIR → pivot rmess).

Un message converti va ensuite être envoyé vers la file d'attente d'entrée de service de distribution.

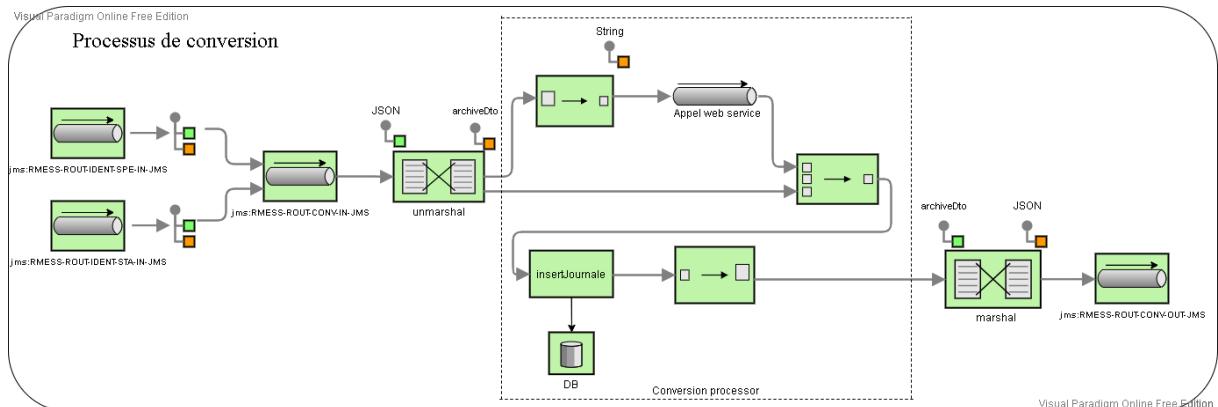


FIGURE 4.12 – Service de conversion.

La figure 4.13 montre le résultat de service de conversion ; le contenu est de type json qui contient le contenu de fichier d'origine, des informations et mots clés et le résultat final obtenu.

The screenshot shows a JSON editor window with the following details:

- Title Bar:** AIMVOYAGESPV100013275.AIR ~Développement/messages/conv
- Buttons:** Ouvrir, Enregistrer, Minimize, Maximize, Close.
- Content Area:**

```
{"archive": "AIR-BLK208;7A;;245;0000000000;GW3351902;001001;nAMD 1900324986;1/1; \n1A1120129;GW3351902;PARTQ219N;AIR\nMUC1A JRVVIX003;0101;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;PARTQ219N;20227141;;;;;;AF JRVVIX\nn-AIR FRANCE;AF 0571\n\nB-TTP/ET\\nc-7906/ 0203WNSU-0203WNSU-I-0-1-F\\nD-210419;210419;210419\\nG- ;;PARPAR;FR\\nH-001;0020RY;PARIS ORLY ;NCE;NICE ;AF 6202 L L 24APR20210715 0840 24APR2021;OK01;HK01;0;318;;0PC;3 ;0635 ;ET;0125 ;N;420;FR;FR;2\nnY-001;0020RY;PARIS ORLY;NCE;NICE;AF;AIR FRANCE;AF;AIR FRANCE;AF;AIR FRANCE;318;N\\nH-002;003ONCE;NICE ;CDG;PARIS CDG ;AF 7711 E E 27APR20210630 0805 27APR2021;OK01;HK01;;0;318;;0PC;2 ;0600 ;ET;0135 ;N;432;FR;FR;2F\\nY-002;003ONCE;NICE;CDG;PARIS CDG;AF;AIR FRANCE;AF;AIR FRANCE;318;N\\nk-FEUR177.00 ;;EUR233.77 ;;\n\\nkFTF; EUR9.32 FR SE; EUR19.90 FR TI; EUR2.26 IZ EB; EUR3.00 04 VC; EUR17.12 QW LO; EUR5.17 UI VZ\\nTAX-EUR9.32 FR ;EUR19.90 FR ;EUR27.55 XT ;\\nL-\\nM- LS50MALG ;E550MALG \\n-EUR116.50;60.00\\n0-24APR24APR;27APR27APR;LD20APR212359\\nQ-PAR AF NCE116.50AF PAR60.00EUR176.50END XT2.26IZ3.000417.12QW5.17UI;FBX\\nI-001;01EVENE/ZOE;;APPAR 01.40.55.80.60 - AIM VOYAGES - A/E-Z.EVENE@TNDA.EU;SK002F;\\nSSR ADTK 1A /TO AF BY 20APR 1500 PAR OTHERWISE WILL BE XLD\\nOSI YY OIN FR09647\\nT-K057-3833774547\\nFM*C*0.00/0.00A/0.00A;S2-3;P1\\nFPCP;P1\\nFSAXG01TUF;P1\\nFVAF;S2-3;P1\\nTKOK19APR\\nPARTQ219N\\nETAF;P1\\nRMRC\\nSNF\\nLOIJE/29090104529944605\\n3JUL2020;P1\\nRMLC\\nSNF\\nGV/29090109408593357;P1\\nENDX\\nf", "idArchive": "421221", "codeTypeMessage": "AMADEUS_GDS_AIR", "motsCles": "PSG:01, IATA:20227141, PNR:JRVVIX, OFFICEID:PARTQ219N", "identifiant": "PARTQ219N", "listDistributeur": [{"id": 754, "adresse": null, "code": "AIM_001", "nom": "AIM VOYAGES"}], "parametres": "PATH=t9-data/rmess/aim/001||URL=10.156.7.6||USER=jboss||PASSWORD=jboss||IATA=20227141||", "extFile": ".AIR", "archiveComplementaire": "01TUF 00WN 017200570038337745470EVENE/ZOE", "CARIAGEReturn": "240421EUR00002337700000000100000000190421CH 0000023377 0000000000 00005677 F", "ZEVENE": "000000000000 NCE L AF NCE CDG E AF", "ZEVENE@TNDA.EU": "JRVVIX [NOMVOYAGEUR$EVENE/ZOE|COURRIER$C|CLASSE$E;;|STOPOVER$0;0;;|TKTT1$ET;ET;;|DESTLIEUS|DESTPAYS$FR|DESTCTY$NCE|TKTTYPE$K|CONTRIB$L;E;;|CNLRESA$T|LIVRAISON$|TOURCODE$|PNRGDS$JRVVIX|IATAEMETTEUR$20227141|DEPAEROPORTCDE$|DEPAEROPORTNOM$PARIS ORLY;NICE;;|ARRAEROPORTNOM$NICE;PARIS CDG;;|ARRAEROPORTCDE$|REFITRE$|EMDSEGMENTS|EMDTYPES|SEGMENTVAL$116.50;60.00;;|CIEEXPLOIT$AF|TXCOM$0.00|NUMVOLS$6202;7711;;|DATEDEP$240421;270421;;|DEPTIME$0715;0630;;|ARRDEP$240421;270421;;|ARRTIME$0840;0805;;|SEGMENTMILE$420;432;;|FAREBASIS$0MALG;0MALG;;|TYPETRF$L5;E55;;|PRIXACHAT$|SFMS|TAXE$56.77|NOSNCFAGC$|NOSNCFNAT$|CLASSTARIF$PUB|CODEGARE$|DUREEVOL$0125 ;0135;;|TYPECOM$|NUMTRNS|UICTRNS|DEPGARENOM$|DEPGARECODE$|ARRGARENOM$|LOWCOST$|NBRSEG$2|DATERESA$190421|OIDRESA$PARTQ219N|DTRETUR$270421|BILLINGNUMBER$|CONFNUM$|PICKUPCODE$|PICKUPADD$|DROPOFFCODE$|DROPOFFADD$|IATAAGENCE$|VOUCHERIND$|CODELOUEUR$|NOMLOUEUR$|FORMPAI$|PRESSCODE$|PRESSTRANS$|NBRPRESS$|FREEPRESS$|NBRDAY$|SOURCE$AIR|MCOSEGMENTS|TYPEMESSAGE$|MISSION|RESAHOTEL$|RESAVOIURE$|TYPESEGMENTS|ACTIF|C02$|NBPASSAGER$01|CODEVENDEUR$0203WNSU|VENDEURRESA$0203WNSU|CLASSETARIFONG$;;|TYPECARTE$|RETRANSMISSIONS|ATFS|CIEEEE$|CIEMARKS$|MCOREFUNDABLE$|EMAIL$|PENALITE$|NUM_SEGMENTS$|CARTEPERSO$|CITYAPT$|CARNUMBER$|SEATNUMBER$|SEATTYPE$|TYPEVEHICULE$|SMOKING$|PNREXTERNE$|JRVVIX|NOMCARTE$|CODETARIF$|TYPEBILLET$|MODEMISSION$|ORIGINE$|SERVICEFEES$|MARKUPSS$|DISCOUNTS$|MARGINPRODUCTTYPE$AIR|NUMEROITREORIGINE$|VIRTUALACCODE$|VIRTUALTICKETNUMBER$|VIRTUALAIRLINECODE$|RSRRC/SNF/PLOIJE/29090104529944605/03JUL2020|RSLRC/SNF/GV/29090109408593357|RSIDFBASIS:LS50MALG;E550MALG|RSRMESSID:421221", "idArchiveComplementaire": "421221", "dateReception": "1623076222611", "lastStatus": "null", "replayNbr": "0", "config": {"identificateur": {"nom": "AMADEUS AIR IDENTIFIER", "namespace": "http://ident.rmess.g3.iga.fr", "url": "http://localhost:8081/rmess-ident-air/Identificateur?wsdl"}, "localpart": "IdentificateurService"}, "typeMessage": {"AMADEUS_GDS_AIR": {"id": 4, "code": "AMADEUS_GDS_AIR", "nom": "AMADEUS AIR VIA AMADEUS"}}, "recepCode": "AIR", "idConvertisseur": null, "versionArchive": "ORIGINAL"}]
```

FIGURE 4.13 – Résultat de service de conversion.

4.3.5 Service de distribution

Dans le service de distribution on a tout d'abord une vérification de message s'il est en mode "standBy" ou pas, si oui ce message va être envoyé vers la file d'attente ROUT-RMESS-STANDBY, si non il va être diffusé vers un ensemble de files d'attentes selon des valeurs de son entête "distributeurs".

La deuxième étape de distribution consiste à distribuer le message vers le back office correspondant.

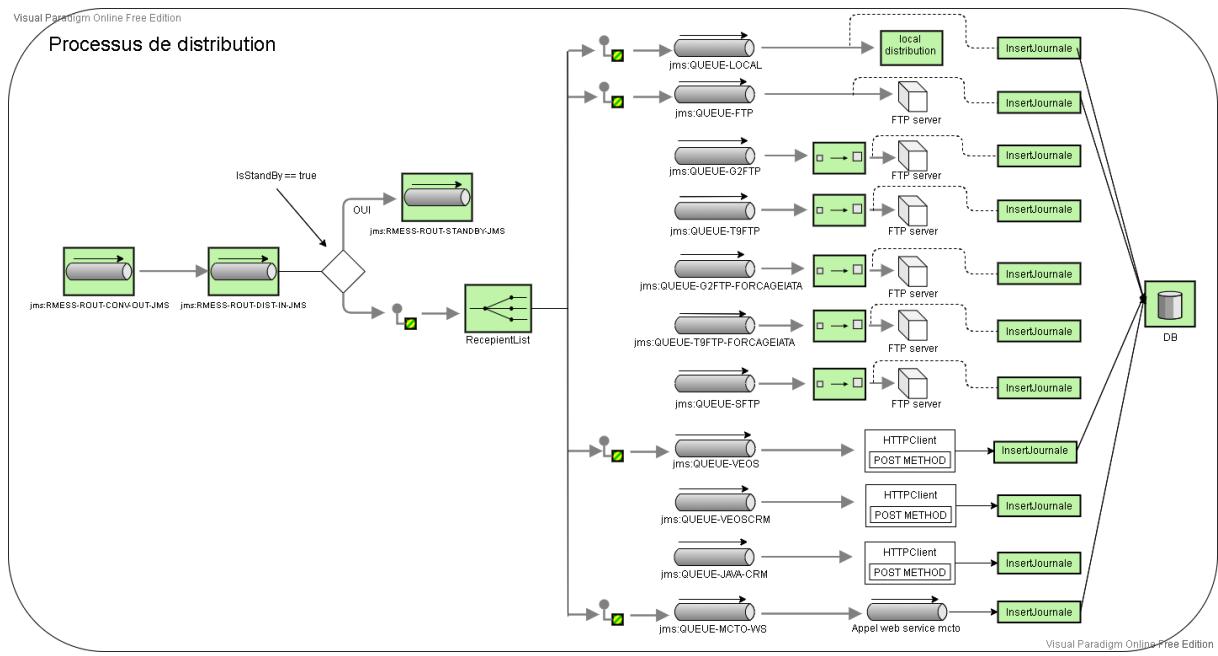


FIGURE 4.14 – Service de distribution.

Nativement la plateforme rMess embarquera la configuration des distributeurs suivants :

Type distributeur	Nom de la file d'attente	Description
LOCALE	RMESS-DELI-FILE-LOCAL-JMS-QUEUE-ESB	Envoi de fichier par le protocole local
FTP	RMESS-DELI-FILE-FTP-JMS-QUEUE-ESB	Envoi de fichier par le protocole ftp.
G2FTP	RMESS-DELI-FILE-G2-JMS-QUEUE-ESB	Envoi le fichier par le protocole ftp avec le paramètre spécial IATA.
T9FTP	RMESS-DELI-FILE-T9-JMS-QUEUE-ESB	Envoi de fichier par le protocole local
SFTP	RMESS-DELI-FILE-SFTP-JMS-QUEUE-ESB	Envoi le fichier par le protocole sftp.
VEOS	RMESS-DELI-FILE-VEOS-JMS-QUEUE-ESB	Envoi le fichier par le protocole ws VEOS VOYAGES.
VEOSCRM	RMESS-DELI-FILE-VEOS-CRM-JMS-QUEUE-ESB	Envoi de fichier par le protocole local

JAVACRM	RMESS-DELI-FILE-JAVACRM-JMS-QUEUE-ESB	Envoi le fichier par le protocole à la servlet CRMJAVA.
G2IATAFORCE	RMESS-DELI-FILE-G2-IATAFORCE-JMS-QUEUE-ESBB	Envoi de fichier par le protocole local
T9IATAFORCE	RMESS-DELI-FILE-T9-IATAFORCE-JMS-QUEUE-ESB	Envoi de fichier par le protocole local
MCTO	RMESS-DELI-FILE-MCTOWS-JMS-QUEUE-ESB	Envoi de fichier par le protocole local

Tableau 4.3: Liste des services de distribution.

On prend l'exemple de distribution locale :

pour le distributeur FTP, on retrouve les paramètres suivants associés table "parametre" :

- PATH • URL • USER • PASSWORD

PARAMETRES =>PATH=path||URL=url||USER=user||PASSWORD=password.

**FIGURE 4.15 – Résultat final.**

La figure 4.15 montre le résultat final de traitement de message distribué dans la destination finale.

4.3.6 Service d'archivage

4.3.6.1 Description

L'archivage des messages rMess sur la plateforme rMess devient nécessaire compte tenu du volume de messages transitant sur la plateforme. Le but de cette évolution est d'alléger les volumes de données présentes dans les tables de travail en déplaçant les anciens messages en archive périodiquement. De cette façon, cela réduira le volume des données présentes en tables et allègera les temps d'utilisation de la plateforme. Les anciens messages sont consultables par la plateforme via un nouveau menu.

Exemple d'un état des lieux de la base au 20/08/2015 :

- Table ARCHIVE : 801 124 lignes.
- Table ARCHIVE_COMPLEMENTAIRE : 739 887 lignes.
- Table ARCHIVE_ENTITE : 692 998 lignes.
- Table JOURNAL : 3 839 696 lignes.

4.3.6.2 Archive Périodique des messages

Une tâche périodique déclenche un archivage des messages. A chaque déclenchement de la tâche, les messages qui ont plus d'un certain nombre de mois sont enregistrés sur une table d'archivage.

Cette table d'archivage contient :

- le message d'origine
- la date de réception
- le dernier message rMess envoyé
- la date d'envoi
- le dernier statut enregistré pour le message
- le code et le nom de l'entité
- le code et le nom du type de message
- les mots-clés
- le nom du fichier
- le nom du fournisseur
- le nom de l'envoyeur

Une fois les informations des messages enregistrées, les tables de travail de la plateforme rMess sont purgées des messages déplacés.

4.3.6.3 Les deux types d'archivage

Concaténation des messages air.

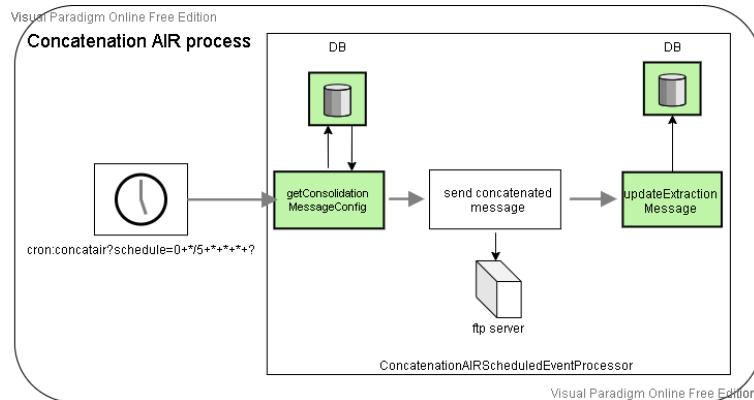


FIGURE 4.16 – Concaténation des messages AIR.

Archivage des anciens messages.

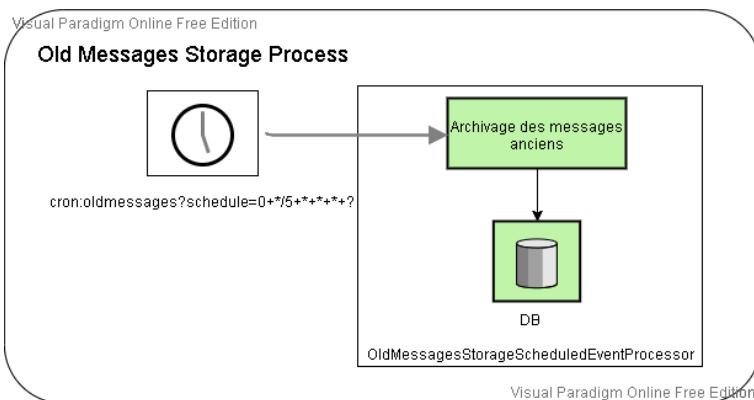


FIGURE 4.17 – Archivage des anciens messages.

4.3.6.4 Gestion des erreurs

Si une erreur a lieu lors de l'archivage d'un message, celui-ci est logué dans un fichier pour analyse. L'archivage des messages continue avec les messages suivants. De plus, un email est envoyé à messages_toiga.fr pour indiquer que des erreurs ont eu lieu lors de l'archivage des données.

4.3.7 Service logger

Toutes les exceptions générées durant l'exécution de cette application sont envoyées vers une file d'attente "RMESS-LOGGER-JMS-QUEUE" puis ces exceptions sont stockées dans la base de données.

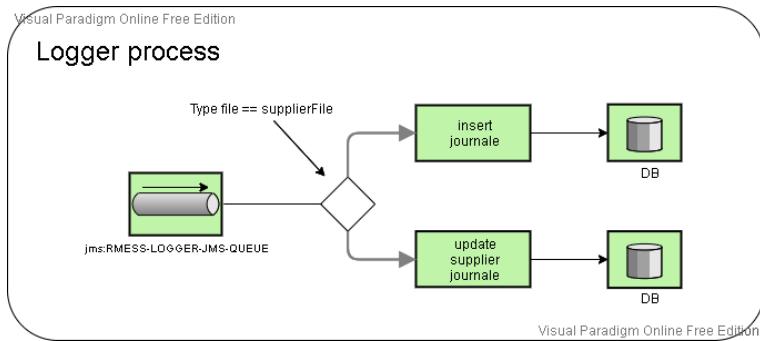


FIGURE 4.18 – Service logger.

La figure 4.18 montre bien la conception de service LOGGER.

4.3.8 Service replay

À certains moments il y a des messages dont leur traitement n'est pas terminé c'est à dire son statut n'est pas "DISTRIBUE".

L'application web dispose donc d'une fonctionnalité qui donne la possibilité de rejouer un message et pour cela mon application contient une route qui est toujours en état d'écoute à la base de données et dès qu'elle détecte un message nécessite un traitement de nouveau il va le consommer et l'envoyer vers la file d'attente d'entrée de service d'identification.

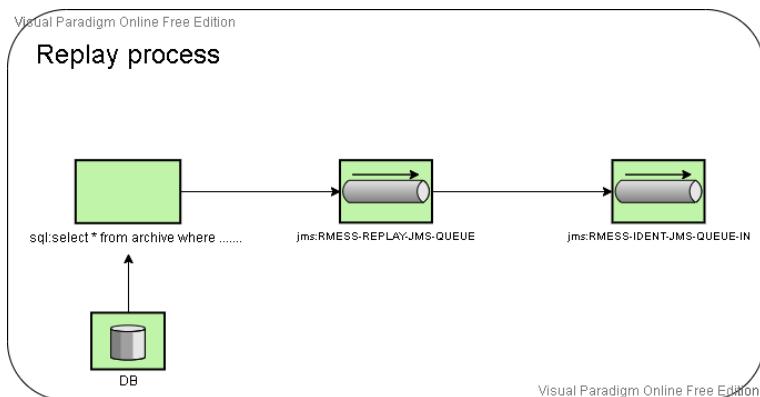


FIGURE 4.19 – Service replay.

4.4 Tests et performance

La migration d'une application d'une architecture vers une autre nécessite d'avoir les mêmes fonctionnalités. Pour cela nous présenterons en premier lieu les tests faits après le développement de la nouvelle application et en deuxième lieu une comparaison entre la performance des deux applications et la mesure des metrics pour la nouvelle.

4.4.1 Tests de validation

Notre application assure la traçabilité de traitement d'un message pour chaque étape et donc nous avons pris cet aspect pour comparer et valider le résultat obtenu avec le résultat attendu de l'ancienne application. Durant le traitement d'un message, le système envoie toujours à chaque étape des informations à la base de données et plus spécifiquement vers trois tables :

- *La table archive : contient le contenu de fichier dès qu'elle a reçu dans l'application et d'autres informations ajoutées pendant le traitement.
- *La table journal : contient la trace d'exécution de chaque étape comme par exemple le statut d'un message s'il est reçu, diffusé, identifié, converti ou distribué.
- *La table archive_complementaire : enregistre le résultat final de traitement.

	id	id_archive	abc_content	ts	id_archive_complementaire	abc_log
1	3 026	423 260	<?xml version="1.0" encoding="UTF-8"?><interf 2021-06-17 11:42:19	2021-06-17 11:42:19	3 026	Recep code :AERTICKET

FIGURE 4.20 – Une vue sur la table Archive.

	id	id_archive	status_code	ts	id_archive_complementaire	abc_log
1	2 166 594	423 260	RECU	2021-06-17 11:42:12	[NULL]	Recep code :AERTICKET
2	2 166 600	423 260	DIFFUSE	2021-06-17 11:42:13	[NULL]	
3	2 166 605	423 260	IDENTIFIE	2021-06-17 11:42:17	[NULL]	*****VERSIONMESSAGE
4	2 166 611	423 260	CONVERTI	2021-06-17 11:42:20	3 026	*****VERSIONMESSAGE
5	2 166 617	423 260	DISTRIBUE	2021-06-17 11:42:23	[NULL]	*****DISTRI:G2FTPURL:

FIGURE 4.21 – Une vue sur la table Journal.

	id	abc_content	ts	id_type_message	abc_mots_cles	abc_identifiant	abc_filename	status
1	423 260	<?xml version="1.0" encoding="UTF-8"?><interf 2021-06-17 11:42:11	2021-06-17 11:42:11	645	PCC:PARAE28BK,OFFICEID:704813,PNR:L6KN 704813,PARAE28BK	interfaceRecord-L6KNUN-2021-04-10_19-00-33-70		checked

FIGURE 4.22 – Une vue sur la table ArchiveComplementaire.

Pour valider le résultat obtenu il suffit de faire la comparaison entre les lignes ajoutées par l'ancienne application et les lignes ajoutées par la nouvelle application pour le même message. Pour cela nous avons fait trois classes de test de validation et nous avons testé ça en utilisant Junit pour tous les types de message et nous avons trouvé le même résultat que le résultat

attendu.

Les figures 4.23, 4.24 et 4.25 montrent les trois tests de validation de résultat.

The screenshot shows the Oracle Database SQL Developer interface. At the top, there are tabs for 'Propriétés', 'Données', and 'ER Diagram'. Below is a search bar with placeholder text 'Entrez une expression SQL pour filtrer les résultats (utilisez Ctrl+Espace)'. The main area displays the 'journal' table with 11 rows of data. The columns are: id, id_archive, asc_status_code, ts, id_archive_complementaire, asc_log. The log column contains various messages such as 'Recep code:AERTICKET', '*****VERSIONMESSAGE : ORIGINAL|IDENT : COCKPIT AERTICKET IDENTIFIER|URL : http://...', and '*****VERSIONMESSAGE : ORIGINAL|ENTITE : INFLUE_001 - INFLUENCE PVT|CONV : COCKP...'. Below the table is a toolbar with icons for Save, Cancel, Script, and various navigation buttons. To the right, there is a 'Properties' panel and a 'Panneaux' sidebar. At the bottom, a status bar indicates '11 ligne(s) ramenées - 107ms'. On the left, a vertical pane shows the JUnit test results:

```
<terminated> UnitTestJournalTable [JUnit] /usr/lib64/jvm/java-ltsmc/bin/java (8 Juin 2021 à 16:53:02 – 16:53:18)
INFO Testing: testRoute() (fr.iga.rmess.test.UnitTestJournalTable)
INFO ****
INFO Routes startup summary (total:0 started:0)
INFO Apache Camel 3.8.0 (camel-1) started in 172ms (build:128ms init:4ms start:3ms)
INFO -----
INFO ----- Unit Test Journal Table -----
INFO ----- ID_ARCHIVE (obtained message) :
421256
INFO -----
INFO ----- Unit Test Journal Table -----
INFO ----- ID_ARCHIVE (expected message) :
21065200
INFO -----
INFO Asserting: mock://expectedResult is satisfied
INFO Asserting: mock://obtainedResult is satisfied
INFO ****
INFO Testing done: testRoute() (fr.iga.rmess.test.UnitTestJournalTable)
INFO Took: 13s869ms (13869 millis)
INFO ****
INFO Apache Camel 3.8.0 (camel-1) shutting down (timeout:10s)
INFO Routes shutdown summary (total:2 stopped:2)
INFO Stopped route2 (direct://start)
INFO Stopped route1 (direct://start)
INFO Apache Camel 3.8.0 (camel-1) shutdown in 47ms (uptime:13s592ms)
```

FIGURE 4.23 – Test de validation table Journal.

The screenshot shows the Oracle Database SQL Developer interface. At the top, there are tabs for 'Propriétés', 'Données', and 'ER Diagram'. Below is a search bar with placeholder text 'Entrez une expression SQL pour filtrer les résultats (utilisez Ctrl+Espace)'. The main area displays the 'archive' table with 2 rows of data. The columns are: id, asc_content, ts, id_type_message, asc_mots_cles, asc_identifiant, asc_filename, status. The status column has a checked checkbox. Below the table is a toolbar with icons for Save, Cancel, Script, and various navigation buttons. To the right, there is a 'Properties' panel and a 'Panneaux' sidebar. At the bottom, a status bar indicates '2 ligne(s) ramenées - 12ms (+1ms)'. On the left, a vertical pane shows the JUnit test results:

```
<terminated> UnitTestArchiveTable [JUnit] /usr/lib64/jvm/java-ltsmc/bin/java (8 Juin 2021 à 17:04:29 – 17:04:44)
INFO ****
INFO Testing: testRoute() (fr.iga.rmess.test.UnitTestArchiveTable)
INFO ****
INFO Routes startup summary (total:0 started:0)
INFO Apache Camel 3.8.0 (camel-1) started in 134ms (build:87ms init:42ms start:5ms)
INFO -----
INFO ----- Unit Test Archive Table -----
INFO ----- ID (obtained message) :
421256
INFO -----
INFO ----- Unit Test Archive Table -----
INFO ----- ID (expected message) :
21065200
INFO -----
INFO Asserting: mock://expectedResult is satisfied
INFO Asserting: mock://obtainedResult is satisfied
INFO ****
INFO Testing done: testRoute() (fr.iga.rmess.test.UnitTestArchiveTable)
INFO Took: 17s428ms (17428 millis)
INFO ****
INFO Apache Camel 3.8.0 (camel-1) shutting down (timeout:10s)
INFO Routes shutdown summary (total:2 stopped:2)
INFO Stopped route2 (direct://start)
INFO Stopped route1 (direct://start)
INFO Apache Camel 3.8.0 (camel-1) shutdown in 34ms (uptime:17s472ms)
```

FIGURE 4.24 – Test de validation table Archive.

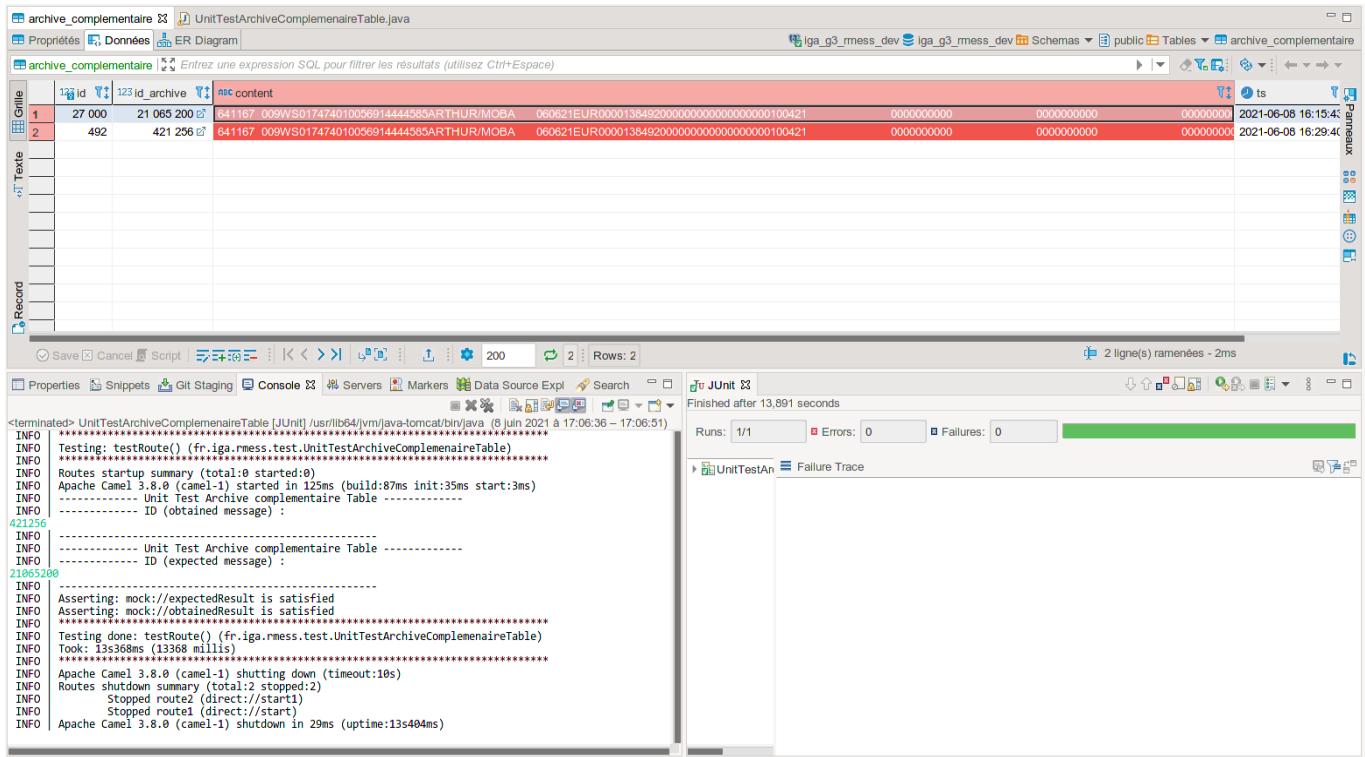


FIGURE 4.25 – Test de validation table ArchiveComplementaire.

4.4.2 Observabilité et tests de performances

Les messages sont abondants et peuvent avoir des pics, donc il est nécessaire d'avoir une solution performante en termes de latence et de temps de traitement.

4.4.2.1 Mesure / critères de performances

Le premier critère que nous avons pensé de le prendre en considération est le temps de démarrage de l'application. L'ancienne est démarré avec le serveur JBOSS qui est un serveur lourd qui prend beaucoup de temps pour leur lancement et comme le montre la figure 4.26 l'application qui est développée avec l'esprit de l'ESB et le serveur JBOSS est lancé dans une minute et quelques secondes par contre pour la nouvelle application le temps de lancement de est seulement 2 secondes comme illustré dans la figure 4.27 et cela grâce à l'une des avantages de apache Camel qui est la possibilité de lancement d'une application java en mode standelone c'est à dire sans besoin d'un serveur d'application.

```

16:29:01,153 INFO [Http11Protocol] Démarrage de Coyote HTTP/1.1 sur http-localhost%2F127.0.0.1-8080
16:29:01,170 INFO [AjpProtocol] Starting Coyote AJP/1.3 on ajp-localhost%2F127.0.0.1-8009
16:29:01,174 INFO [ServerImpl] JBoss (Microcontainer) [5.1.0.GA (build: SVNTag=JBoss_5_1_0_GA date=200905221634)] Started in 1m:38s:707ms

```

FIGURE 4.26 – Temps de démarrage de l'ancienne application.

```
INFO | Started Logger_Supplier (jms://queue:RMESS-SUPPLIER-LOGGER-JMS-QUEUE)
INFO | Apache Camel 3.8.0 (camel-1) started in 2s867ms (build:78ms init:1s277ms start:1s512ms)
```

FIGURE 4.27 – Temps de démarrage de nouvelle application.

Pour comparer la performance de la nouvelle solution il suffit de prendre des lots de messages pour les deux applications et comparer leurs temps de traitement de la réception vers la distribution en se basant sur la table journal de la base de données qui contient la trace d'exécution de chaque service pour chaque message.

Nous avons développé en premier lieu une version de l'application et la chose qu'on a pris comme un critère primordiale est d'avoir les mêmes fonctionnalités pour les deux applications et lorsque nous avons atteint cet objectif nous avons passé vers une deuxième version plus performante dans laquelle nous avons changé un peu l'architecture et la manière d'injection de l'entité manager.

Le tableau 4.4 montre le temps de traitement de chaque lot de message pour l'ancienne application, la version 0 et la version 1 pour la nouvelle solution.

NB : temps de traitement d'un lot de messages = la date de dernier message distribué - la date de premier message reçu.

	Ancienne application	Nouvelle application (version 0)	Nouvelle application (version 1)
5 messages	13 secondes	23 secondes	12 secondes
10 messages	13 secondes	23 secondes	10 secondes
15 messages	12 secondes	34 secondes	14 secondes
20 messages	17 secondes	41 secondes	19 secondes
25 messages	34 secondes	49 secondes	19 secondes
35 messages	24 secondes	69 secondes	23 secondes
45 messages	32 secondes	90 secondes	34 secondes

Tableau 4.4: Temps de traitement des messages.

4.4.2.2 Exposition des métriques

Le monitoring est crucial lorsqu'il s'agit de connaître l'état de la plate-forme. Le suivi de bon fonctionnement du système a tendance à être une priorité pour s'assurer que l'activité habituelle se poursuit normalement. En plus des mesures de santé de base, telles que l'utilisa-

tion du processeur et de la mémoire, il peut également être intéressant de surveiller d'autres indicateurs de niveau supérieur pertinents pour une application[8].

Parmi les caractéristiques de monitoring que camel offre est la possibilité d'exposer des informations statistiques des routes à travers les MBeans qui sont accessibles par le JMX. Il existe également une console Web extensible qui fournit une interface utilisateur pour la gestion de Camel, ActiveMQ et bien d'autres, appelée hawtio.

Nous pouvons voir comme le montre les trois figures suivantes, que tout ce qui concerne le routage est sous contrôle. Nous avons le nombre de messages en cours, le nombre d'erreurs, le nombre des messages dans les files d'attentes et même le temps d'exécution d'un service défini dans une route spécifique.

Ces informations peuvent être acheminées vers un ensemble d'outils de surveillance avec des fonctionnalités riches comme Grafana.

Name	State	Uptime	Completed	Failed	Handled	Total	Inflight	Mean time
Cron_ConcatAir	Started	13m9s	12	0	0	12	0	9058 ms
Cron_OldMessagesStorage	Started	13m9s	12	0	0	12	0	9211 ms
Distribution	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_FTP	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_G2FTP	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_G2FTP_FOR...	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_LOCATE	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_MCTD	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_SFTP	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_SUPPLIER	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_T9FTP	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_T9FTP_FOR...	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_VEOS	Started	13m9s	0	0	0	0	0	-1 ms
Distribution_VEOSCRM	Started	13m9s	0	0	0	0	0	-1 ms
IdenSTA_to_Conversion	Started	13m9s	0	0	0	0	0	-1 ms
Identification_STA	Started	13m9s	0	0	0	0	0	-1 ms
Logger	Started	13m9s	0	0	0	0	0	-1 ms
Logger_Supplier	Started	13m9s	0	0	0	0	0	-1 ms
OnRamp_SUPPLIER	Started	13m9s	0	0	0	0	0	-1 ms
OnRamp_SUPPLIER_TO...	Started	13m9s	0	0	0	0	0	-1 ms
Reception_AERTICKET	Started	13m9s	0	0	0	0	0	-1 ms
Reception_AIR	Started	13m9s	0	0	0	0	0	-1 ms
Reception_COMM1PROD	Started	13m9s	0	0	0	0	0	-1 ms
Reception_GDS_GALILEO	Started	13m9s	0	0	0	0	0	-1 ms
Reception_JGF	Started	13m9s	0	0	0	0	0	-1 ms
Reception_JUR	Started	13m9s	0	0	0	0	0	-1 ms
Reception_MCTD	Started	13m9s	0	0	0	0	0	-1 ms
Reception_OTA	Started	13m9s	0	0	0	0	0	-1 ms
Reception_SUPPLIER	Started	13m9s	0	0	0	0	0	-1 ms
Reception_SUPPLIER_TO...	Started	13m9s	0	0	0	0	0	-1 ms
Reception_TMS	Started	13m9s	0	0	0	0	0	-1 ms
Reception_COMM1PROD	Started	13m9s	0	0	0	0	0	-1 ms

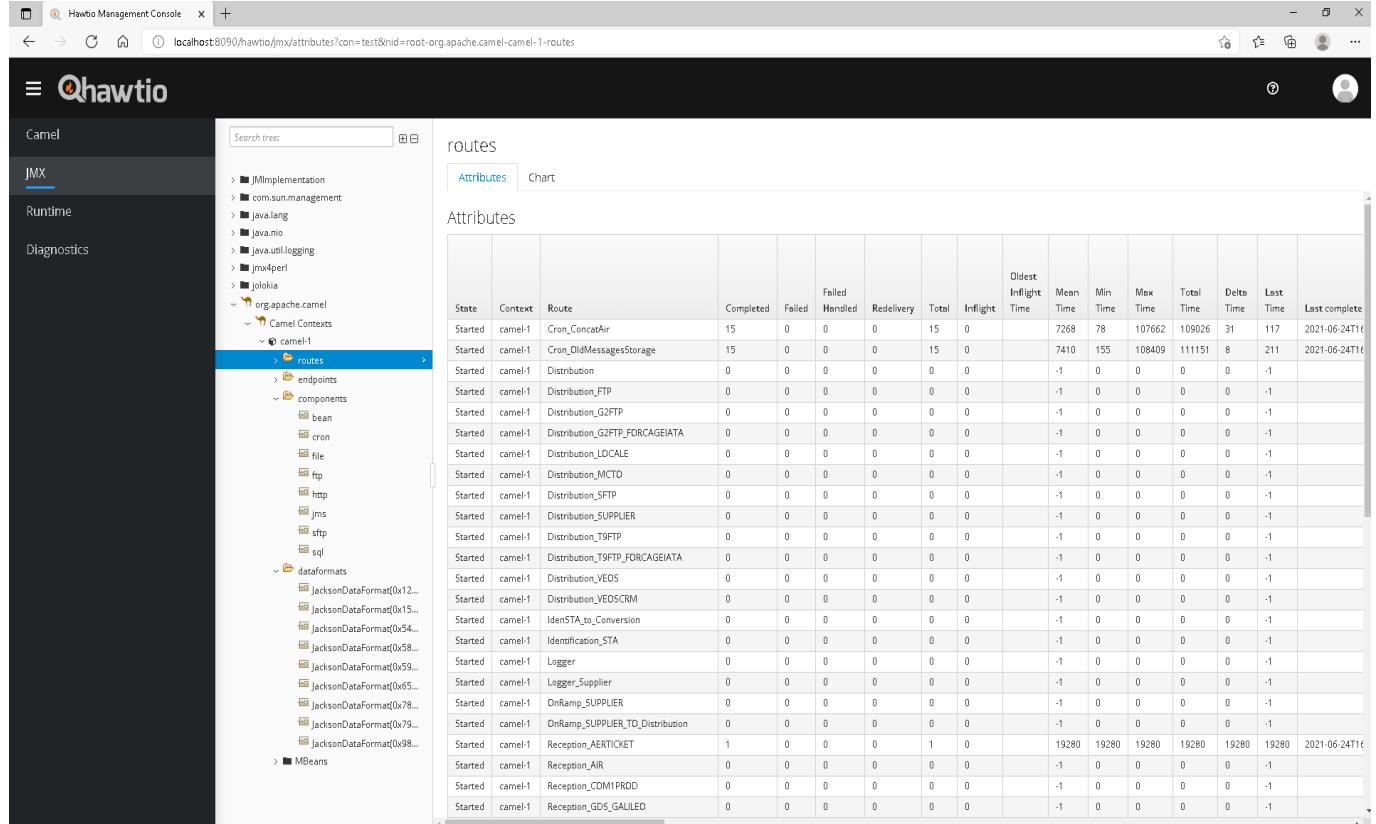
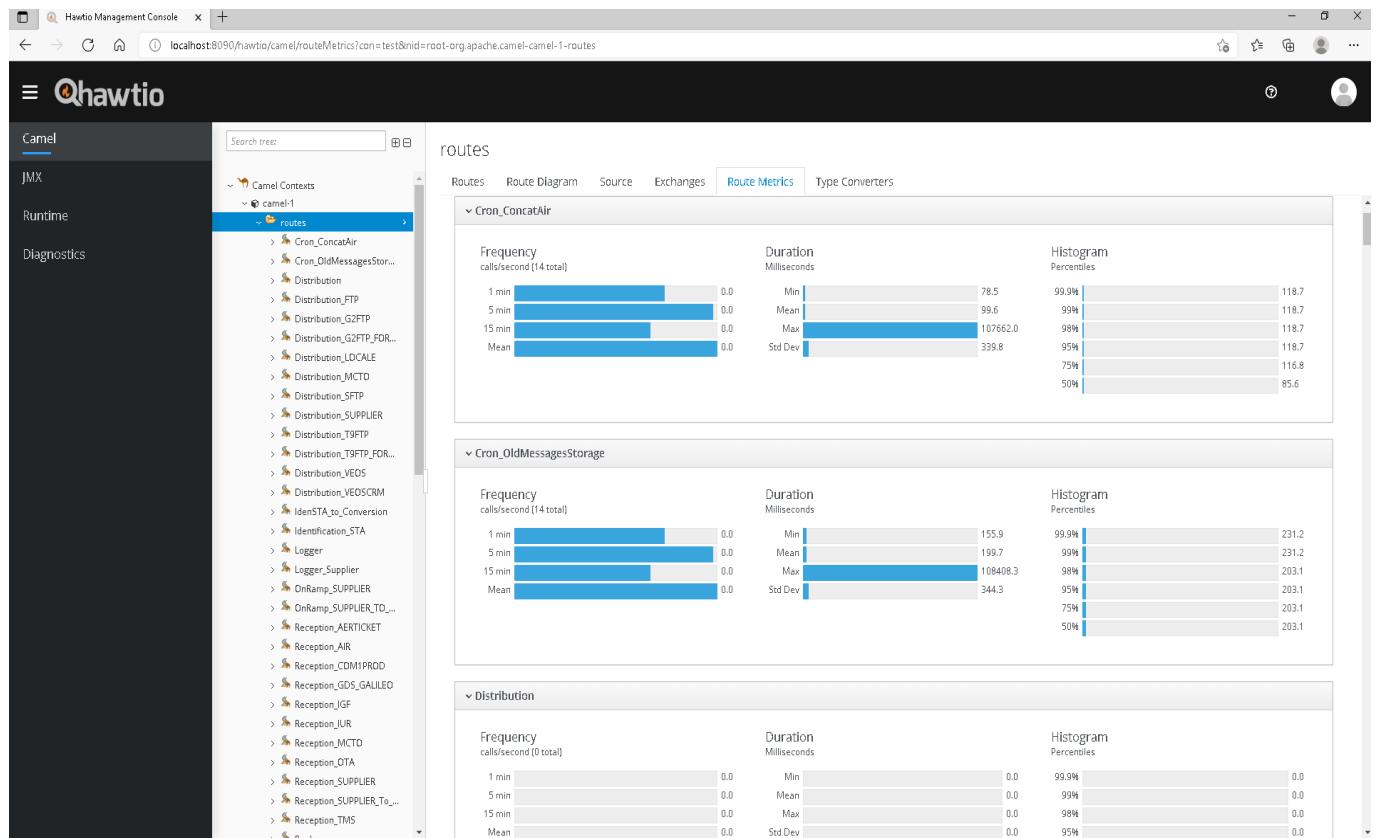


FIGURE 4.28 – Exposition des métriques avec hawtio.

Conclusion

Durant ce chapitre, nous avons présenté les environnements matériels et logiciels de notre application. Ensuite, nous avons faire une étude théorique pour expliquer l'outil apache Camel puis nous avons décrit quelques scénarios d'exécution par des captures d'écran et les diagrammes de apache camel. Et pour clôturer nous avons parler des tests de validation et performance et l'exposition des métriques des routes de notre application.

Conclusion générale et perspectives

L'objectif de notre projet de fin d'études était la migration d'une application d'intégration d'entreprise d'une architecture basée sur l'entreprise service bus ; c'est comme un canal de communication qui assure le routage et médiation des données entre des services qui sont connectés à lui par des connecteurs, vers une technologie moderne et plus avantageuse. Nous avons donc pensé d'utiliser un framework open source de routage et médiation des données qui est le Apache camel.

La première étape était l'étude de l'existant et la proposition d'une solution adaptée à notre besoin. Par la suite nous avons compris l'utilisation de ce framework.

L'étape suivante était la conception détaillée de nouvelle solution et la précision des différents composants nécessaires pour le développement de chaque service. Ensuite, nous sommes passés à la phase de mise en œuvre. Au cours de cette phase, nous avons utilisé différents langages et technologies de programmation pour aboutir au résultat souhaité.

Le dernier volet de notre projet était la mesure de performance de nouvelle application et la comparaison avec l'ancienne solution.

Nous avons atteint, par ce projet, les objectifs que nous nous sommes fixés dès le début de sa réalisation, tel que l'obtention des mêmes fonctionnalités que l'ancienne, l'utilisation d'un framework léger et qui va être une bonne solution si nous pensons au cloud. Néanmoins, plusieurs perspectives sont envisagées telles que l'exposition des métriques et le monitoring de l'application avec des outils modernes comme le Grafana.

Par ailleurs, ce projet de fin d'études nous a été très bénéfique, aussi bien au niveau technique qu'au niveau professionnel. En effet, on a eu l'occasion d'enrichir notre base de connaissances sur les technologies d'intégration des applications.

Nétographie

- [1] *ActiveMQ*. URL : <https://activemq.apache.org/>. [Consulté : Juin 2021].
- [2] *Apache Maven*. URL : maven.apache.org/. [Consulté : Juin 2021].
- [3] *Architecture point à point*. URL : <https://zato.io/docs/intro/esb-soa-fr.html>. [Consulté : Juin 2021].
- [4] *Introduction au Framework Apache Camel*. URL : <https://www.cours-gratuit.com/cours-framework-java/cours-d-introduction-au-framework-apache-camel>. [Consulté : Juin 2021].
- [5] *Introduction to Apache Camel*. URL : <https://docs-external.u4pp.com/integration-kit/Introduction-to-Apache-Camel/index.html>. [Consulté : juin 2021].
- [6] *Introduction à l'utilisation d'eclipse*. URL : <https://dept-info.labri.fr/ENSEIGNEMENT/programmation2/intro-eclipse/>. [Consulté : Juin 2021].
- [7] *Java Is the Language of Possibilities*. URL : https://www.java.com/en/download/help/whatis_java.html. [Consulté : Juin 2021].
- [8] *Monitoring Camel*. URL : <https://www.opensourcerers.org/2017/11/29/monitoring-camel-prometheus-red-hat-openshift/>. [Consulté : Juin 2021].
- [9] *PostgreSQL*. URL : <https://www.postgresql.org/about/>. [Consulté : Juin 2021].
- [10] *Présentation de l'entreprise IGA Groupe*. URL : <http://www.iga.fr/>. [Consulté : Août 2021].
- [11] *Présentation de l'équipe IGA Voyage*. URL : <http://www.iga.fr/logiciel-agence-de-voyage/>. [Consulté : Août 2021].
- [12] *Présentation du framework hibernate*. URL : <https://mail.codejava.net/frameworks/hibernate/java-hibernate-jpa-annotations-tutorial-for-beginners>. [Consulté : Juin 2021].
- [13] *Qu'est-ce qu'un diagramme de paquetages UML ?* URL : <https://www.lucidchart.com/pages/fr/diagramme-package-uml>. [Consulté : Juin 2021].
- [14] Margaret ROUSE. *use case diagram (UML use case diagram)*. URL : <https://whatis.techtarget.com/definition/use-case-diagram>. [Consulté : Juillet 2021].
- [15] *The ESB Architecture*. URL : <https://www.hcltech.com/blogs/everything-you-need-know-about-enterprise-service-bus-esb>. [Consulté : Juin 2021].

- [16] *What is Apache Camel ?* URL : <http://blog.coralic.nl/2011/03/18/10-minutes-hands-on-apache-camel/>. [Consulté : Juin 2021].

Bibliographie

- [1] Omer Aziz : Research Trends in Enterprise Service Bus (ESB). In : (2020). URL : <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8985259>. [consulté : août 2021].
- [2] Anton Goncharov : Streamline software integration : An Apache Camel tutorial. In : (2018). URL : <https://jaxenter.com/streamline-software-integration-an-apache-camel-tutorial-140621.html>. [consulté : août 2021].
- [3] Simon Denel : SCALABILITÉ HORIZONTALE VS SCALABILITÉ VERTICALE. In : (2018). URL : <https://www.smartwavesa.com/blog-articles/scalabilite-horizontale-vs-scalabilite-verticale/>. [consulté : Juillet 2021].