

DOSSIER TECHNIQUE DU PROJET ESB - BRUTUSIP

Dans le cadre de la formation POEI Conception et développement d'applications

Organisme formateur : AFPA

Entreprise à l'origine du recrutement : Atos

Joan Séverin



Projet ESB – Joan Séverin



AVERTISSEMENT

Avis de droits de propriété intellectuelle

Ce document est protégé par des droits de propriété intellectuelle et est la propriété exclusive d'Atos. Le procédé exposé dans le présent rapport est protégé par la loi sur la propriété intellectuelle et reste la propriété d'Atos ou de parties tierces quant à l'utilisation de progiciels.

Vous pouvez utiliser ce rapport à des fins académiques et non commerciales conformément aux principes régissant le droit de la propriété intellectuelle. Toute autre utilisation ou modification du rapport sans l'autorisation écrite préalable de la Société est interdite.

Sauf mention contraire expresse, tous les droits de propriété intellectuelle contenus dans ce rapport et propre à la solution technique exposée sont la propriété d'Atos.



Projet ESB – Joan Séverin



REMERCIEMENTS

Je tiens à remercier l'AFPA d'être à l'initiative de ce projet qui m'a permis de me former à un nouveau métier, et d'être recruté en CDI.

Plus particulièrement Patricia Gallais, Pascal Dangu, Bruno Collin, Philippe Viguier, Djamila Causse.

Je remercie Atos de m'avoir recruté et fait confiance.

Plus particulièrement Nathalie Grès, Lise Romans, Jérôme Jansou, Eloïse Issert et mon équipe projet : Inès Benelhadj Ali, Patricia Segonds, Mohamed Ali Bach Tobji, Mouslih Ezzaki, Michael Denais.

Je remercie la DTI et tous les fonctionnaires rattachés au projet pour leur accueil et leur disponibilité, plus particulièrement Laëtitia Mabilleau, Brunilde Girardet.

Merci au jury de consacrer son temps à la lecture de ce rapport.



Projet ESB – Joan Séverin



SOMMAIRE

Avertissement	2
Remerciements	3
Sommaire	4
Glossaire	6
Introduction	9
Présentation de l'entreprise	10
Historique	10
Présentation du groupe	10
Présentation de l'agence	12
Différents services – Organigramme	12
Présentation du client	13
Présentation du projet	17
Contexte du projet	17
Objectifs du projet	18
Descriptif du projet – Cahier des charges	21
Planification du projet	23
Productions attendues	23
Environnement technique	24
Analyse des besoins	25
Conception	32
Implémentation	48
Tests et validation	64



Projet ESB – Joan Séverin



Bilan de projet	69
Difficultés rencontrées	69
Etat d'avancement du projet	70
Bilan et perspectives	71
Bilan personnel	71
Perspectives	71



Projet ESB – Joan Séverin



GLOSSAIRE

Programmation

Métier

AMAN	Arrival Manager
API	Application Programming Interface
ATC	Air Traffic Control
ATM	Air Traffic Management
CAUTRA	Coordinateur Automatique du Trafic Aérien
CRNA	Centre en-Route de la Navigation Aérienne
DGAC	Direction Générale de l'Aviation Civile
DSL	Domain-Specific Language
DSNA	Direction des Services de la Navigation Aérienne
DTI	Direction Technique et de l'Innovation



Projet ESB – Joan Séverin



EAI	Enterprise Application Integration
ENAC	Ecole Nationale de l'Aviation Civile
EOS	Exigences Opérationnelles des Systèmes
ESA	Exigences Systèmes et Architecture
ESB	Enterprise Service Bus
HA	High Availability
IHM	Interface Homme-Machine
MOM	Message-Oriented Middleware
MRT	Manufacturing, Retail & Transport
OLDI	On-Line Data Interchange
OMS	Objet de Métier Spécifique
PSSI	Politique de Sécurité du Système Informatique
REST	Representational State Transfer



Projet ESB – Joan Séverin



SOA	Service Oriented Architecture
SES	Single European Sky
SESAR	Single European Sky ATM Research
STPV	Système de Traitement Plan de Vol
SWIM	System-Wide Information Management



Projet ESB – Joan Séverin



INTRODUCTION

Dans le cadre d'une Préparation Opérationnelle à l'Emploi Individuelle, j'ai été recruté par Atos pour être formé au métier de concepteur développeur d'application. Suite à 6 mois de formation encadrée par l'AFPA, j'ai intégré Atos pour une période de 9 mois.

J'ai été rattaché à l'agence MRT, qui gère des clients relevant des secteurs du transport et de l'industrie, et on m'a fait intégré un projet pour le compte de la DTI, partie de la DSNA, elle-même antenne de la DGAC.

Le projet concerne le développement d'un ESB pour répondre à différents enjeux.

Durant mon affectation à la DTI, j'ai travaillé sur le projet BrutusIP, qui est la migration d'une solution de transfert de données sur protocole IP.

Ma participation au projet a donné lieu au développement de deux demi-flux entrants, la rédaction de cahiers de tests pour l'ensemble du flux. Dernièrement j'ai commencé le développement d'une appli multicouche qui permet l'administration d'abonnements au travers d'une IHM basée sur une API REST.



Projet ESB – Joan Séverin



PRESENTATION DE L'ENTREPRISE

HISTORIQUE

Né de la fusion en 1997 d'Axime et Sligos, deux entreprises de service informatique, Atos devient en 2000 ATos Origin suite à sa fusion avec Origin. De nombreux rachats d'entreprise et fusions plus tard, Atos Origin reprend en 2011 le simple nom d'Atos. La différence étant que le logo, avec son S majuscule, signifie désormais Atos Origin to SIS suite au rachat de Siemens IT Solutions.

D'autres entreprises continuent de se faire acquérir par Atos, et fait en 2017 son entrée au CAC 40.

Plus récemment, Atos a beaucoup investi dans l'innovation de l'intelligence artificielle avec Google Cloud, et la réduction de l'empreinte carbone avec EcoAct, cabinet de conseil spécialisé dans la décarbonation.

PRESENTATION DU GROUPE

En quelques chiffres, Atos représente 110 000 employés répartis sur 73 pays, 12 milliards de chiffre d'affaire en 2019, 4500 brevets actifs et 235 millions investis dans la recherche et le développement.

Atos est également le partenaire officiel informatique des Jeux Olympiques et Paralympiques.

Ses activités sont répartis sur trois divisions : Business & platform solutions, Infrastructure & data management, Big Data & cybersecurity. Ces trois divisions répondent à un marché segmenté en six catégories : Manufacturing ; Financial Services & Insurance ; Public Sector & Defense ; Telecom, Media & Technology ; Resources & Services, Healthcare & Life Sciences.



Projet ESB – Joan Séverin



Le président non exécutif est Bertrand Meunier, et le directeur général, Elie Girard.



Projet ESB – Joan Séverin



PRESENTATION DE L'AGENCE

Atos France représente 3.5% du revenu du groupe.

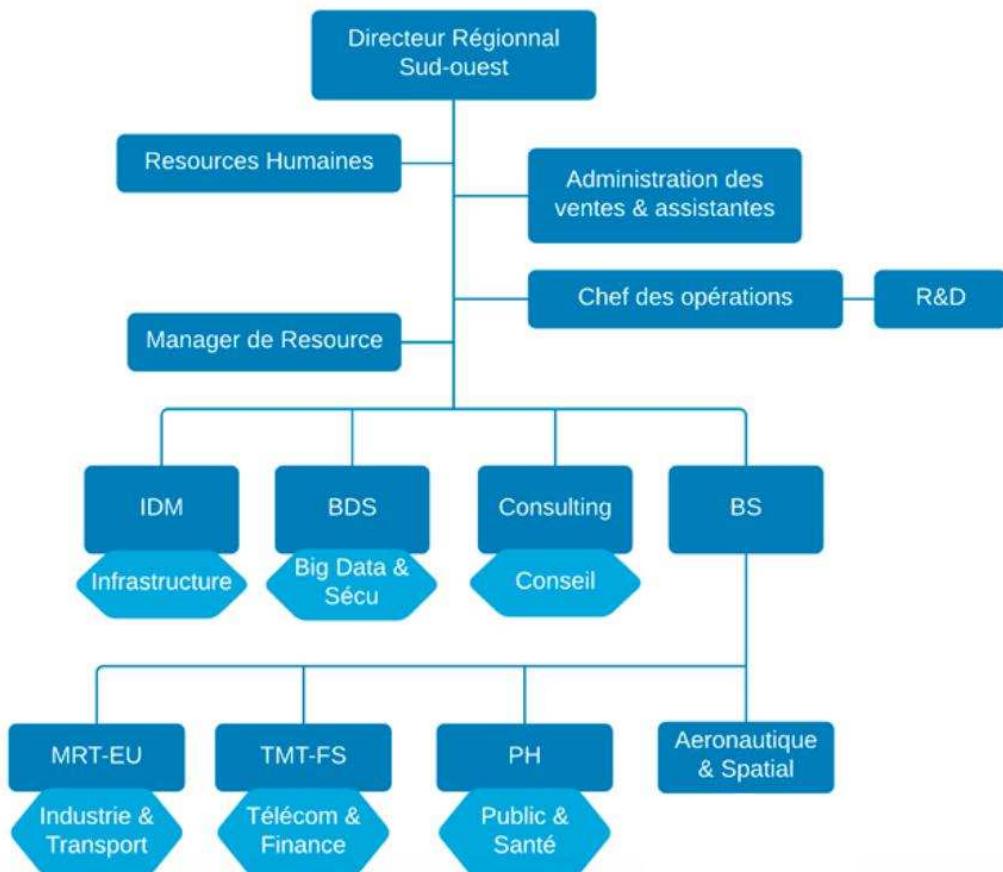
L'agence d'Atos basée à Toulouse se répartit en quatre agences, représentant des marchés et secteurs d'activités :

- MRT (Industrie et transport)
- TMT (Télécom et finance)
- PH (Public et santé)
- Aéronautique et spatial

DIFFERENTS SERVICES – ORGANIGRAMME



Projet ESB – Joan Séverin



PRESENTATION DU CLIENT

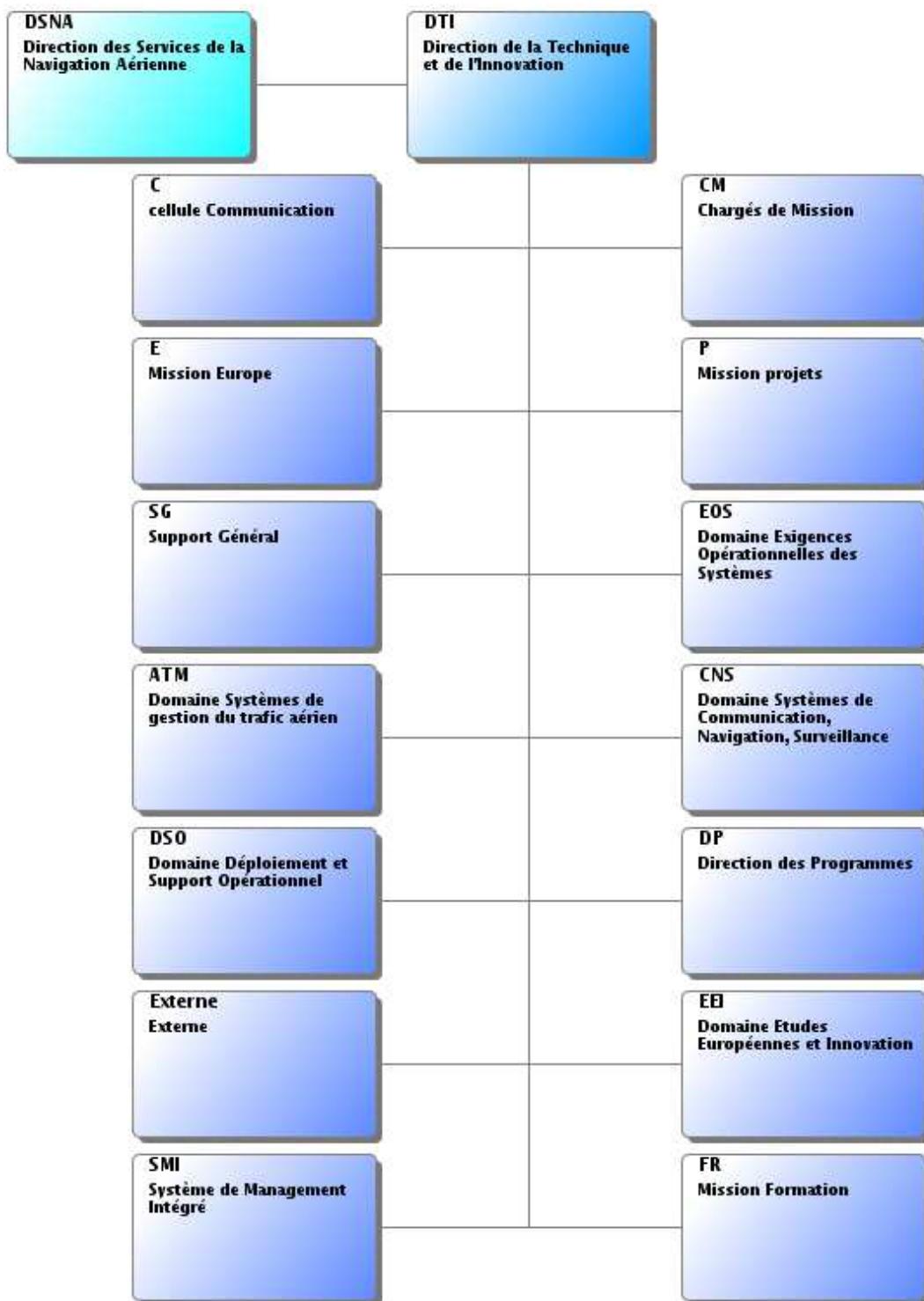
Le client rattaché au projet est la DTI (Direction de la Technique et de l'Innovation), un service de la DSNA (Direction des Services de la Navigation Aérienne) et plus largement à la DGAC (Direction Générale de l'Aviation Civile). La DTI est chargée de définir, de réaliser et de déployer des systèmes ATC (Air Traffic Control) et d'assister techniquement le contrôle aérien pour la DSNA qui assume le rôle d'opérateur de contrôle du trafic aérien pour la France.



Projet ESB – Joan Séverin



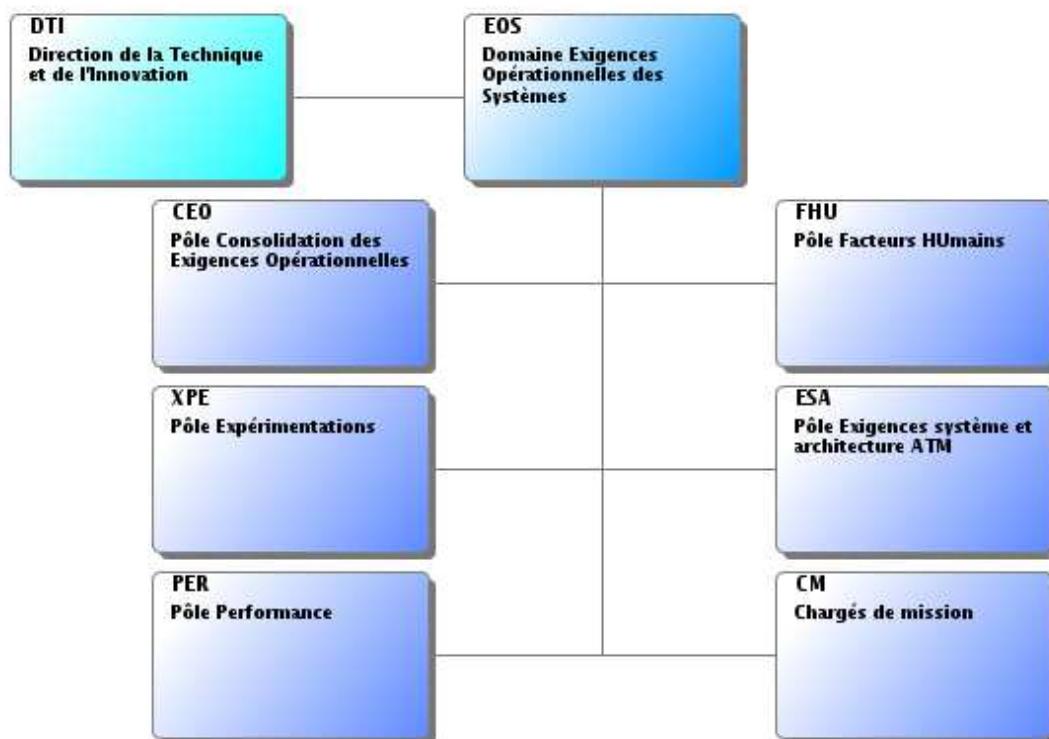
La DTI est l'architecte des équipements de la navigation aérienne concernant les cinq CRNA (Centres en-Route de la Navigation Aérienne), la dizaine de grands aéroports métropolitains avec leur espace associé et les aérodromes d'outre-mer d'intérêt national ainsi que l'ENAC (Ecole Nationale de l'Aviation Civile). Proche du monde industriel, elle fait régulièrement appel à des SSII qui l'assistent dans la prise en charge des activités liées à la réalisation des projets les plus complexes.



Le domaine EOS (Exigences Opérationnelles des Systèmes) de la DTI est chargé de contribuer au retour d'expérience, à la consolidation des besoins opérationnels et des évolutions correspondantes des méthodes de travail et de formation.

Il définit les exigences techniques relatives aux systèmes d'assistance automatisée, à la gestion du trafic aérien et aux systèmes d'information aéronautique. Il conduit également des études relatives aux facteurs humains et à la performance.

Enfin le pôle ESA (Exigences Systèmes et Architecture ATM), au sein duquel j'ai travaillé, est chargé de définir les exigences techniques relatives aux systèmes de gestion du trafic aérien et d'information aéronautique, ainsi qu'aux simulateurs de formation. Plus concrètement, le pôle a pour mission le recueil du besoin, le design d'architectures, l'innovation, toujours en étroite coordination avec les utilisateurs de la navigation aérienne.





Projet ESB – Joan Séverin



PRESENTATION DU PROJET

CONTEXTE DU PROJET

Le principe de l'ATM (Air Traffic Management) relève de l'agrégation de l'ensemble des services basés au sol et dans l'air, et requis pour garantir un mouvement sécurisé et fluide des aéronefs à chaque stade des opérations.

Les contrôleurs aériens ont ainsi besoin d'un échange permanent d'informations relatives aux arrivées, aux départs, aux projections de trajectoire des avions. Le CAUTRA (Coordinateur Automatique du Trafic Aérien) a été inventé dans les années 60 dans le but de répondre à ce besoin, et consiste en la première automatisation du traitement des données de vol civil.

En 1999, l'initiative Ciel Unique Européen, ou SES (Single European Sky) proposée par la Commission Européenne, ambitionne de moderniser et uniformiser les systèmes de gestion du trafic aérien. Le SESAR (Single European Sky ATM Research) est le versant technique de ce programme. Après quatre ans de définition de ces nouvelles générations de systèmes, près de 350 projets sont développés en Europe.

En France, la DGAC, au travers de la DSNA, dresse un constat négatif sur le caractère vétuste et hétéroclite de ses outils informatiques dans le cadre de l'ATM.

Des formations sont organisées, axées notamment sur l'architecture orientée services (SOA). C'est en s'inspirant de ce qui a été réalisé en Suisse, où un ESB a été mis il y a cinq ans au centre de l'architecture de ses systèmes d'ATM, que le projet prend forme.

L'équipe ESA développe ainsi son ESB. Après un premier essai en 2017, le projet se dote d'un nom iSODA, d'un architecte et d'une équipe plus étoffée, malgré un fort turn-over depuis lors.



Projet ESB – Joan Séverin



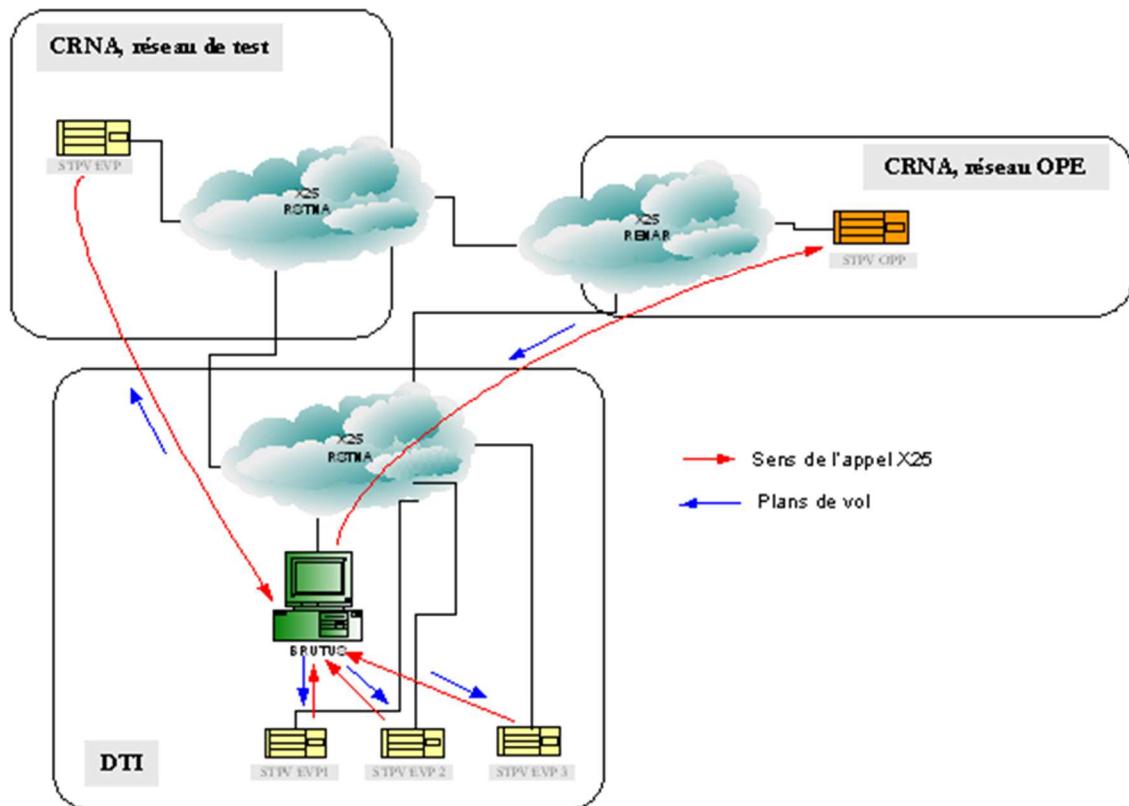
OBJECTIFS DU PROJET

Pour mener les tests de systèmes ATM, les plateformes de tests de la DTI ont besoin d'être alimentées en données réelles, dont les données « plans de vol », qu'elles soient issues d'enregistrements de trafic ou du trafic en temps réel ("live traffic").

Cette alimentation en données plans de vol est aujourd'hui assurée via une solution technique, appelée BRUTUS, qui permet de récupérer à la DTI les données distribuées par les STPV opérationnels (OPP) des 5 CRNA.

Pour ce faire, le STPV OPP dispose d'une sortie spécifique appelée EVAL2 qui est la recopie du flux de trafic opérationnel. Cette recopie est destinée à l'origine au STPV d'évaluation (EVP) local à chaque CRNA, sur le domaine de test.

Dans cette architecture, BRUTUS a été intercalé entre les STPV OPP et EVP de test. Il joue ainsi le rôle de serveur proxy et collecte les flux en provenance des STPV OPP pour les renvoyer, en fonction des demandes, à un ou plusieurs clients « EVAL2 » qui sont, soit les STPV EVP du réseau de test de chaque CRNA, soit des STPV de tests de la DTI. Le schéma suivant représente l'architecture à remplacer :



Cet utilitaire est aujourd'hui indispensable pour tous les tests de la DSNA nécessitant des données plans de vol. Il permet également d'alimenter d'autres entités hors DSNA, telle que la Défense, qui utilisent ces flux pour mettre au point leurs propres systèmes.

L'outil Brutus fonctionne actuellement à la DTI sur des PC LINUX munis d'une carte X25. L'acquisition des données est réalisée uniquement via le protocole X25, conformément à l'infrastructure technique « Télécom & Réseaux » qui prédominait à la DSNA jusqu'à il y a peu. Cette technologie étant obsolète, la DSNA a prévu la migration de toutes ses liaisons réseau vers la pile de protocoles TCP/IP.

L'objectif est donc de migrer les flux en IP, de revoir l'architecture pour répondre aux problématiques actuelles (conformité avec la PSSI, intégrité des données et indépendance des CRNA pour l'alimentation de leurs STPV de test) et d'avoir un outil évolutif et facilement administrable.

La solution retenue implique le déploiement d'un relais Brutus-IP local à chaque CRNA. Il permettra d'alimenter de façon autonome le STPV EVP de test du CRNA, sans passer par un



Projet ESB – Joan Séverin



rebond à la DTI comme c'est actuellement le cas. Ce composant est nommé "Brutus-IP@Site". Un serveur Brutus-IP à la DTI, appelé "Brutus-IP@DTI", collecte et redistribue les données vers les applications TPV clientes (STPV, SHIVA).

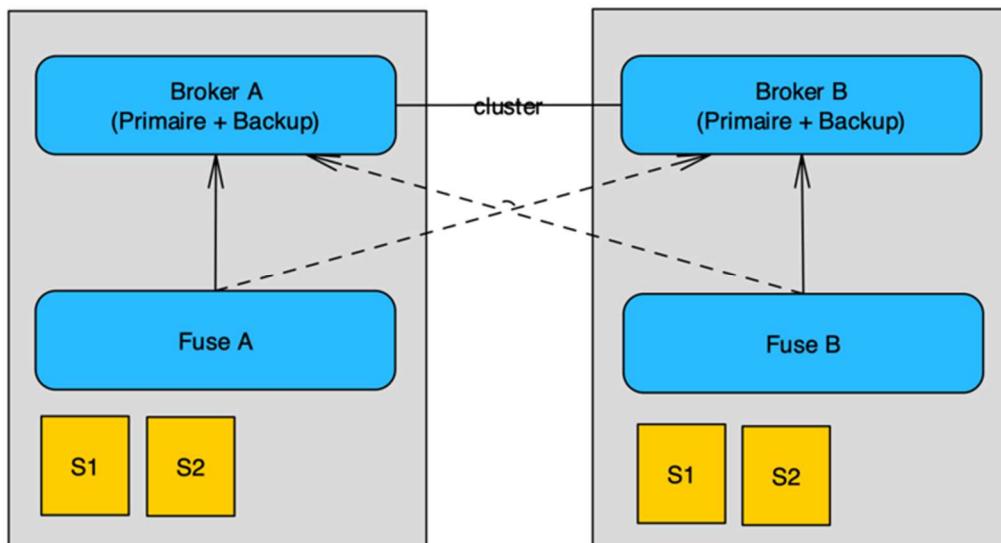
DESCRIPTIF DU PROJET – CAHIER DES CHARGES

Le cahier des charges du projet ESB est résumé ainsi :

- **Architecture haute-disponibilité**

Redondance physique, redondance logique.

2 brokers actifs/ 2 backups, 2 serveurs applicatifs Fuse, 2 instances par service.



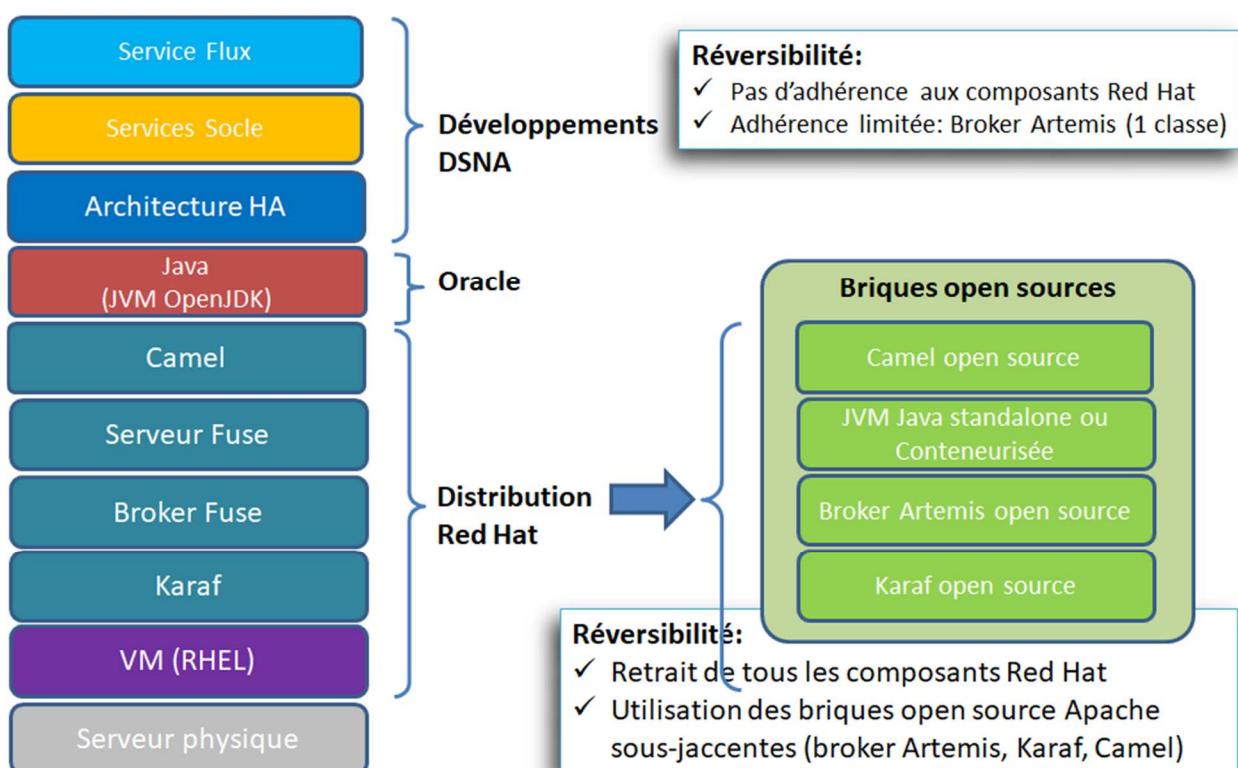
Continuité de service assurée en cas de :

- Perte d'un serveur physique.
- Perte de 3 brokers sur 4.
- Perte d'un serveur applicatif Fuse.
- Perte d'une instance de service.
- Purge automatisée en cas de saturation du stockage.
- Dépendance technique aux fournisseurs

Pile logicielle full open-source.

- **Maturité et processus de développement**

Process et livrables disponibles :



- conception logicielle (et volumes de code existants)
- spécification
- revues de code
- tests unitaires
- tests fonctionnels
- tests d'intégration
- tests de non régression
- gestion de configuration logicielle



Projet ESB – Joan Séverin



- traçabilité entre les référentiels

Compatibilité SWAL 4 et SWAL 3.

Le cahier des charges spécifique au projet BrutusIP n'a pas été réalisé.

PLANIFICATION DU PROJET

Initialement prévu pour une livraison à l'été 2020, le projet BrutusIP doit être livré respectivement fin avril 2021 pour BrutusIP@CRNA, et fin avril 2021 pour BrutusIP@DTI.

Le projet est réparti sur 5 itérations ; chaque itération comprend 4 sprints de 3 semaine, plus une semaine d'innovation.

PRODUCTIONS ATTENDUES

Le projet ESB iSODA est attendu sur la production de cinq services de flux, en se basant sur un certain nombre (13 à l'heure où ce rapport est rédigé) de services socles, qui sont développés à chaque nouveau besoin.

- Flux DataStore EEI - all (Diffusion de données brutes sur AMQP)
- Flux Proxy NM
- Flux Proxy Swim MET
- Flux Swim Aman - arrival sequence
- Flux STPV - inter cautra

J'ai pour ma part travaillé sur le flux STPV - inter cautra



Projet ESB – Joan Séverin



ENVIRONNEMENT TECHNIQUE

Environnement technique

Le produit pour l'ESB utilisé par la DTI est Red Hat FUSE 6.3 (avec un broker Artemis).

Les services sont écrits en JAVA (8).

Les routes sont décrites via un DSL camel.

IDE utilisé dans le développement back-end des services : Red Hat CodeReady Studio 11.2

IDE utilisé dans le développement front-end des services : Visual Studio Code 1.52.1

Artemis, utilisé dans la gestion des flux et demi-flux : apache-artemis-2.11.0-bin.zip

Serveur d'application, utilisé pour lancer localement le back-end : jboss-fuse-karaf-6.3.0.redhat-187.zip

Un environnement de test est accessible sur VM via connexion SSH.

Les projets java développés sont des projets Camel qui ont la structure de projets Maven.



Projet ESB – Joan Séverin



ANALYSE DES BESOINS

Les besoins du client concernant le projet BrutusIP sont définies sous forme d'une liste d'exigences qui servira comme guide lors de la phase d'implémentation, ainsi qu'on le verra plus bas. La liste est la suivante, et les besoins sont hiérarchisés ainsi : **prioritaires P**, **secondaires S**, optionnels O

Généralités :

- **P - Compatibilité avec les correspondants actuels**
La solution doit pouvoir s'interfacer avec les entités fournisseurs et consommateurs.
- **S - Gestion des liaisons dédiées**
- **P - Gestion des protocoles RFC1006 et CAUTRA**
Brutus-IP implémente les protocoles de communication suivants : TCP/IP, CAUTRA, RFC1006.
- **S - Gestion du protocole FMTCP**
Pour l'alimentation de certains clients, BRUTUS-IP implémente le protocole de communication FMTCP.
- **S - Gestion du protocole MTP**
Pour l'alimentation de certains clients, BRUTUS-IP implémente le protocole de communication MTP.
- **O - Désactivation du protocole CAUTRA**
Il est possible de désactiver la gestion du protocole CAUTRA.
- **P - Relais pour les données STPV**
Brutus-IP joue le rôle de proxy au travers d'une interface "fournisseur" qui permet de récupérer les données plans de vol, ainsi que d'une interface "utilisateur" qui permet



de redistribuer ces données. Le STPV OPP du domaine ATM du CRNA se connecte au port "fournisseur" de Brutus-IP@Site. Brutus-IP@Site se connecte au STPV EVP du domaine de test local et au Brutus-IP@DTI. Les clients présents à la DTI (SHIVA et STPV) se connectent au Brutus-IP@DTI sur le port "utilisateurs". Brutus-IP@DTI peut également initier les connexions vers les abonnés (mode client ou serveur).

- **O - Limitation des connexions fournisseurs**

Brutus-IP permet de limiter le nombre de connexions entrantes simultanées sur son port "fournisseur". S'il y a une tentative de connexion alors que le nombre de liaisons établies autorisé est déjà atteint, celle-ci doit être rejetée et une erreur doit être générée. Cette valeur est configurable. Par défaut, il sera paramétré pour n'accepter qu'une seule connexion entrante mais ce paramètre sera modifiable notamment pour l'utilisation à la DTI où il sera possible, au besoin, d'autoriser jusqu'à 5 connexions entrantes sur une même instance de l'outil. Ce mécanisme permet de détecter une mauvaise configuration des clients, notamment un mauvais numéro de port utilisé.

- **P - Nombre de clients à desservir**

Brutus-IP doit pouvoir gérer à minima 75 connexions simultanées entrantes sur le port "utilisateurs" (clients à alimenter en données plans de vol). De même, l'outil est en capacité de se connecter à au moins 75 clients (connexions sortantes)

- **P - Sens des flux**

L'échange des données applicatives est unidirectionnel, du fournisseur vers l'utilisateur. Seuls les messages utiles à la gestion des connexions sont autorisés dans les deux sens.

- **P - Contraintes sécurité et sûreté**

Brutus@IP doit être conforme aux exigences de la PSSI et s'assurer de son innocuité vis-à-vis du STPV opérationnel.

- **S - Relance automatique des connexions**

Les liaisons préalablement paramétrées doivent être relancées automatiquement au



Projet ESB – Joan Séverin



démarrage de l'application. De plus, en cas d'échec de connexion ou de déconnexion inopinée, Brutus-IP doit tenter automatiquement de se reconnecter (le nombre de tentatives est configurable ainsi que la durée entre chaque tentative).

- O - Généricité de la solution

Le même produit est déployé pour les différents composants : Brutus-IP@Site et Brutus-IP@DTI. Seule la configuration spécifique à chaque site et composant variera.

Gestion des données :

- P - Gestion des abonnés / clients

Brutus-IP gère les abonnements des clients à un flux de données d'un CRNA spécifique (mapping des flux). Tous les messages applicatifs en provenance d'un CRNA reçus sur le port "fournisseur" sont transmis vers les clients abonnés à ce flux et connectés au port "utilisateur".

- O - Intégrité des données

Brutus-IP fournit un mécanisme de vérification de l'intégrité des données.

- S - Temps de routage des messages

Les données sont transmises vers les utilisateurs en temps réel : le temps de transit des messages ne doit pas excéder 1 seconde.

- S - Statistiques

Des données sont fournies sur :

- Le nombre de liaisons actives.
- Le nombre total de clients desservis par centre.
- Le nombre de connexions/déconnexions sur une liaison.

- P - Filtrage des données STPV

Brutus-IP permet de filtrer certains messages de types actions internes et externes lors de la diffusion des données vers un utilisateur donné.



- O - Comptage par type de messages
Brutus-IP permet de compter les différents messages applicatifs reçus, par type, grâce aux titres présents dans les messages CAUTRA.
- O - Découverte automatique des clients
Brutus-IP peut détecter automatiquement les tentatives de connexion d'un nouveau client et proposer l'ajout de nouvelle liaison.
- O - Changement de version de format
Brutus-IP peut faire de la conversion entre différentes versions formats de messages STPV (exemple du format AIN15 vers AIN17).
- O - Initialisation des clients
Brutus-IP d'effectuer une initialisation rapide des clients grâce à l'utilisation des liaisons dédiées.

Paramétrage :

- P - Configuration des liaisons
BRUTUS-IP permet à l'utilisateur de configurer les paramètres des connexions :
 - Nom de la liaison.
 - Adresse IP ou nom d'hôte du correspondant.
 - Type du correspondant :
 - Fournisseur : BRUTUS-IP@Site ou STPV OPP.
 - Utilisateur : BRUTUS-IP@DTI, STPV EVP, SHIVA ou autre ESB.
 - Site concerné : AIX, BORDEAUX, BREST, ATHIS, REIMS.
 - Numéro de port TCP local associé au service.
 - Mode : client ou serveur (d'un point de vue TCP/IP, initiateur ou accepteur de la connexion).
 - Numéro de port TCP distant : obligatoire si le mode client est utilisé.
 - Timers émission/réception pour le protocole CAUTRA (par défaut : 3 et 10 secondes).



Projet ESB – Joan Séverin



- Nombre de tentatives de reconnexion automatique en cas de déconnexion (par défaut : 3).
- **S - Choix des protocoles à utiliser**
Il est possible de choisir le protocole parmi ceux supportés.
- **P - Synchronisation horaire**
Le système hôte de Brutus-IP offre un service de synchronisation horaire.

Traces :

- **P - Niveaux de traces**
Brutus-IP doit permettre à l'utilisateur de choisir le niveau de traces relatives aux états de connexion. Les différents niveaux de traces sont les suivants :
 - ERROR : uniquement les messages d'erreur (niveau par défaut).
 - INFO : les messages d'erreur et les messages d'information.
 - DEBUG : les messages d'erreur, les messages d'information et les messages de débogage utiles pour les développeurs.
- **P - Messages d'erreur**
Les messages d'erreurs relatifs aux échecs ou aux ruptures de connexion doivent être explicites afin de permettre de déterminer précisément le problème.
- **P - Journal de bord**
Un journal de bord trace l'ensemble des événements importants liés à l'applicatif BRUTUS-IP.
- **P - Rétention des traces**
La durée de conservation des messages de log est de minimum :
 - 6 mois pour le journal de bord et les autres logs applicatifs.
 - 1 mois pour les traces du système hôte.

Administration :



Projet ESB – Joan Séverin



- **P - Console**
Brutus-IP doit être administrable en ligne de commandes.
- **P - Interface graphique**
Brutus-IP doit être administrable via une IHM graphique.
- **P - Connexion distante**
Brutus-IP est administrable à distance via connexion SSH.
- **P - Actions d'administration en console**
Les commandes de base à prévoir sont :
 - Lancer/arrêter les différents processus/services.
 - Lancer/arrêter l'IHM.
 - Ajouter/modifier/supprimer une liaison.
 - Ouvrir/fermer une liaison.
 - Paramétrier le niveau de traces.
 - Afficher le journal de bord.
 - Afficher les autres logs.
 - Visualiser l'état du système et des connexions.
 - Afficher la version logicielle.
- **P - Actions d'administration par l'IHM**
Avoir une vue synthétique de l'état du système
 - Consulter les traces/le journal de bord.
 - Gérer les liaisons (configuration et monitoring).
 - Administrer l'outil.
- **P - Profils utilisateurs**
Gestion des rôles pour les différents utilisateurs du système avec des droits associés à leur fonction. Les deux rôles à définir à minima sont : superviseur et administrateur.

Supervision :



Projet ESB – Joan Séverin



- **S - Supervision de l'outil**

Brutus-IP doit inclure un outil qui permet d'avoir une vue synthétique de l'état du système au niveau :

- matériel (ex : disques, interfaces réseau).
- logiciel (ex : processus, espace de stockage, synchronisation NTP).
- applicatif (ex : états des connexions, des services).

- **P - Supervision des liaisons**

L'utilisateur doit avoir une vue de l'ensemble des connexions et accéder aux informations courantes ou actions associées à une liaison :

- Sa configuration.
- Son état.
- Commande 'ouvrir' ou 'fermer' en fonction de l'état de la liaison.

- **S - Génération d'alertes**

Brutus-IP doit permettre de configurer et générer des alertes en cas de défaillance d'une liaison.



Projet ESB – Joan Séverin



CONCEPTION

L'ESB en général

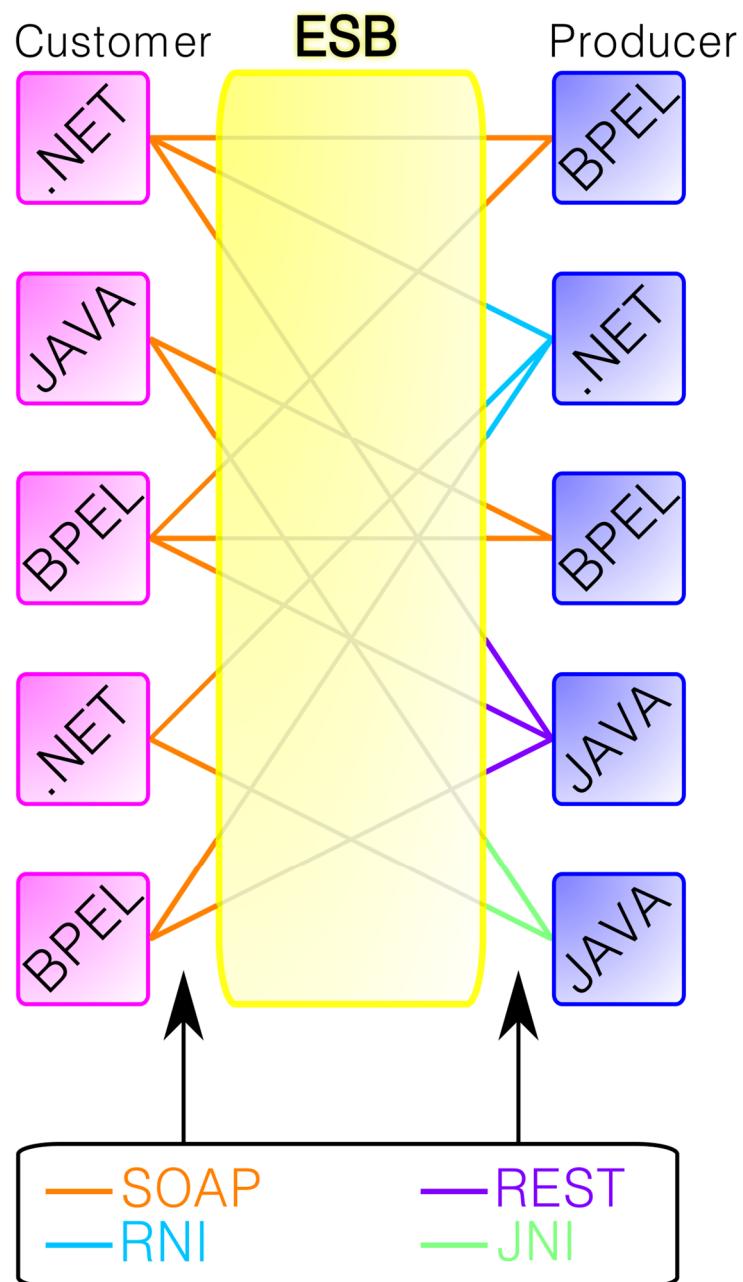


Schéma générique de la logique d'un ESB



Projet ESB – Joan Séverin



L'ESB, Enterprise Service Bus, est considéré comme une nouvelle génération d'EAI (Enterprise Application Integration), ou tout du moins une évolution. C'est-à-dire un middleware qui permet la mise en communication d'applications clientes qui ne sont pas destinées, par les caractéristiques de leurs OMS (Objets de Métier Spécifiques), à échanger des données. Il permet à des applications diverses de se connecter à un bus de données via de nombreuses possibilités d'adaptateurs (WS, AMQP, BDD, SOAP, etc) et de s'abonner à des topics de messages en fonction de règles structurelles dictées par le métier.

Il s'agit donc d'une plateforme de médiation entre clients et fournisseurs de services, un MOM (message-oriented middleware) indépendant des applications qu'il met en relation, et pourtant au cœur de son architecture SOA.

Il se détaille en une file de messages (broker) et en composants apparentés à des micro-services qui vont router automatiquement les messages selon un itinéraire défini et ainsi mettre en relation un fournisseur et un consommateur. Chacun de ces services qui peuvent relever de l'orchestration, de logging, de mapping, ou encore de manipulation de base de données, peuvent être déployés indépendamment les uns des autres.

Concernant l'adaptabilité de l'ESB, son architecture distribuée permet une évolution de ses composants pour répondre efficacement à des potentielles montées en charge.



Projet ESB – Joan Séverin



L'ESB à la DTI

L'ESB iSODA, aussi appellé en interne "SwimInfra" ou "FUSE".

Il route des messages entre producteurs et consommateurs (end-points), on parle d'une « route » pour un échange donné sur l'ESB.

Les services qui réalisent ce travail sont des micro-services indépendants (OSGI) qui s'échangent des messages via un broker interne à l'ESB. Ces services sont stateless et leur conduite est dictée par un circuit de commande transverse (gestion des abonnements, par exemple).

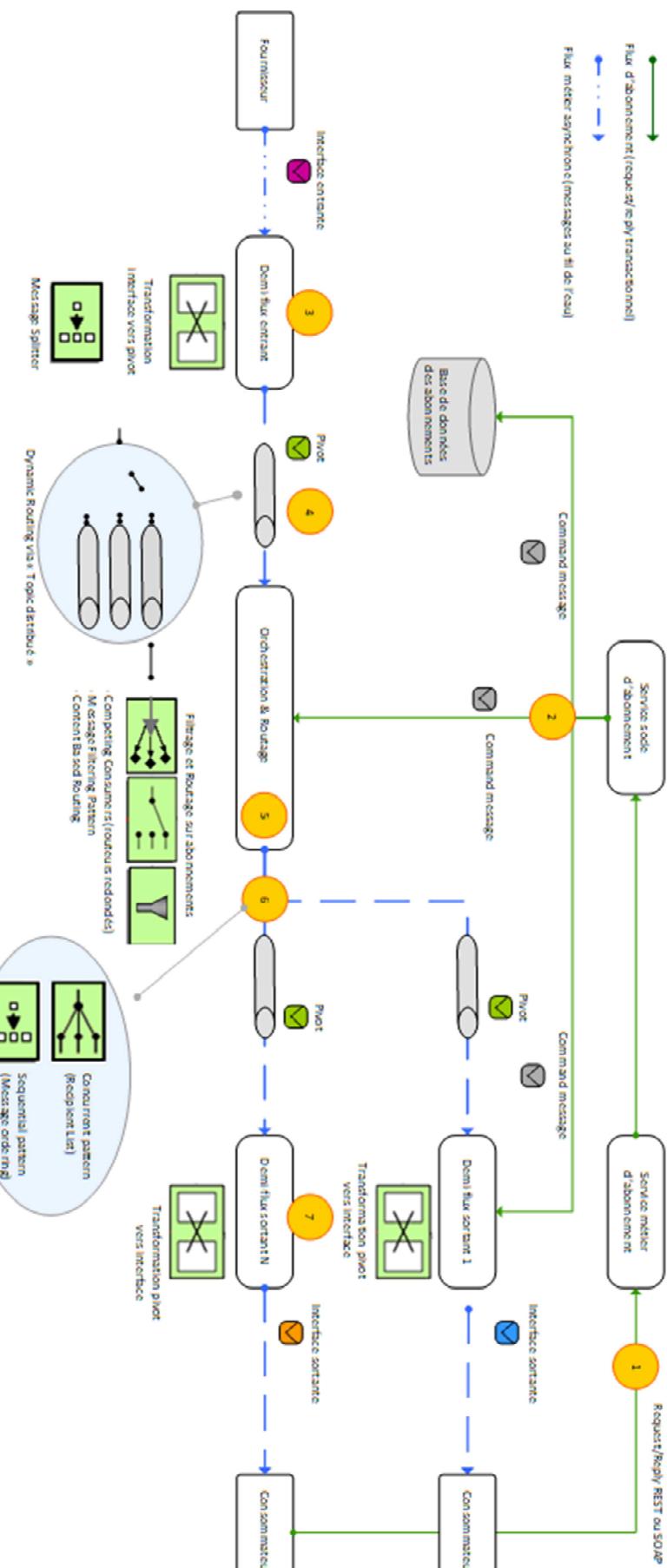
La solution implémentée garantit la redondance, la scalabilité et une supervision des flux.

Tout « service de flux » nécessite à minima 3 briques (ou services) :

- Un demi flux entrant (qui extrait une donnée, lui associe un format pivot, et la poste sur une file de message d'un broker)
- Un service de médiation ou orchestration qui consomme cette donnée, et sur la base du format pivot, duplique, filtre et route les messages vers d'autres files
- Un demi-flux sortant (qui dépile la donnée et offre le service final)

Autour de ces trois briques, sont associées des services techniques qui composent un socle de fonctionnalités, telles que la manipulation d'une base de données pour enregistrer et éditer les abonnements des consommateurs, ou encore l'exposition des dits abonnements sur un endpoint http REST.

Ci-contre un schéma logique du fo
moment de la rédaction de ce rapp
donc pas exhaustif.



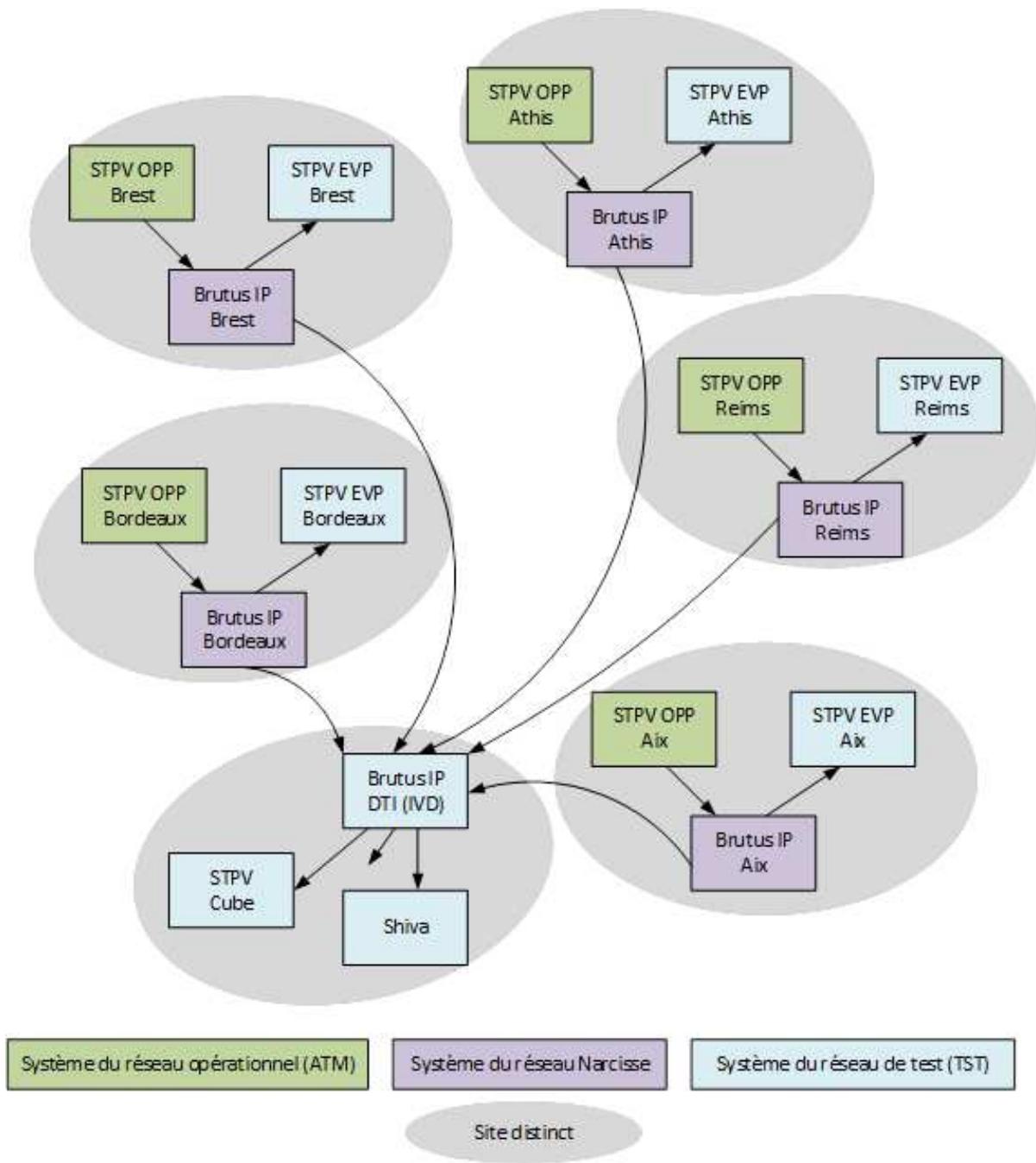


Projet ESB – Joan Séverin



Comme on peut le voir ci-dessus, on retrouve aux extrémités gauche et droite un fournisseur et un consommateur. Entre ces derniers, divers composants que nous allons détailler :

- Les trois briques mentionnées plus haut :
 - demi-flux entrant qui fait l'interface avec le fournisseur et récupère la donnée métier. Sans modifier le message lui-même, il va le mapper sur un format pivot qui va être sur une file de messages configurée sur le broker.
 - flux d'orchestration qui consomme le message. En fonction de la configuration du format pivot et des objectifs du service de flux, il va le filtrer et le router.
 - demi-flux sortant qui transmet le pivot sur l'interface cible.
- Service d'abonnement, qui a pour but de gérer une base de données. Celle-ci est alimentée en abonnements, qui correspondent à l'association des données d'un topic (ou queue de messages) et celles d'un client qui s'y est abonné. Ces abonnements vont être utilisés pour pouvoir router les messages pivots.
- Service d'administration des abonnements, qui expose une api REST ainsi qu'une IHM pour manipuler les abonnements.
- Service de données à la demande, qui fournit les données d'un flux selon deux types de requête : pattern last-value, délivrant la dernière donnée de la queue de messages, et pattern flow-redelivery, délivrant un ensemble de messages sur une période et un débit donné.



Le service de flux STPV - InterCautra

Schéma du rôle du service de flux BrutusIP@DTI



Projet ESB – Joan Séverin



Le schéma ci-dessus décrit le rôle du service de flux BrutusIP@DTI dans le système global de l'ATM de la DSNA. Les "sites distincts" en gris représentent les cinq CRNA en France, ainsi que le site dont tous les cinq dépendent : la DSNA à Toulouse. Nous avons donc :

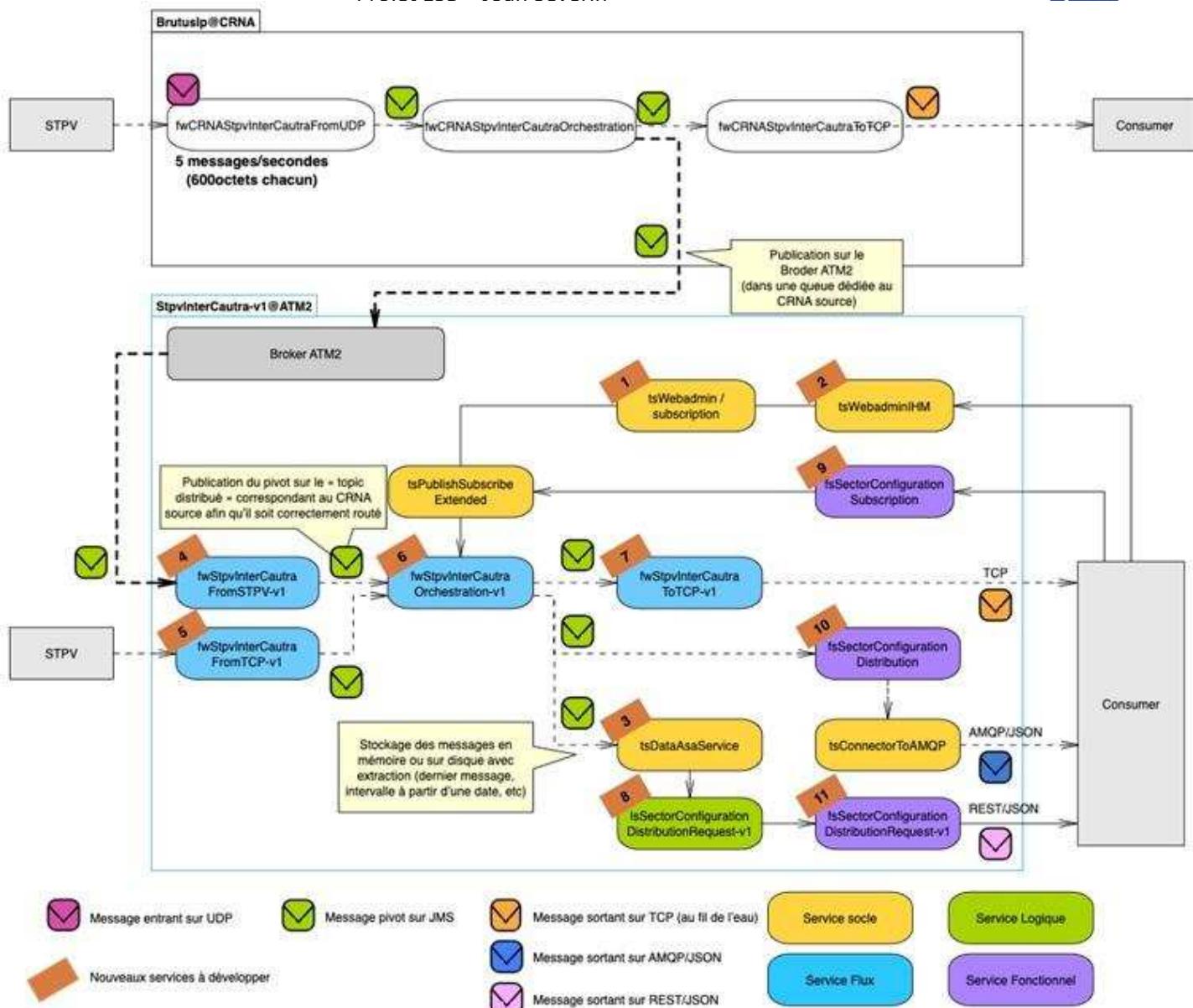
- CRNA Nord, Athis-Mons (situé près de l'aéroport d'Orly en région parisienne) ;
- CRNA Est, Reims ;
- CRNA Ouest, Loperhet (situé près de Brest) ;
- CRNA Sud-Est, Aix-en-Provence ;
- CRNA Sud-Ouest, Mérignac (situé près de l'aéroport de Bordeaux).

Ce sont ces cinq sites qui produisent les données STPV (Système de Traitement Plan de Vol), ici notre OMS. À leur origine, les plateformes STPV OPP, qui transmettent l'OMS à un premier service de flux ESB : le flux BrutusIP@CRNA qui, comme son nom l'indique, est propre à chaque région. C'est depuis ce flux que parvient l'OMS au flux BrutusIP@DTI qui, en plus de récupérer les données propres à sa propre région, agrège celles de chacun des cinq autres sites.

Ces messages sont par la suite mis en forme et mappés par le demi-flux entrant, routés par le service d'orchestration, et selon les use cases, transférés au consommateur via le demi-flux sortant sur protocole TCP.

Le service de flux BrutusIP@DTI se décompose ainsi comme présenté sur le schéma ci-après:

Projet ESB – Joan Séverin



Architecture logique du service de flux BrutusIP@DTI

Briques développées

J'ai pour ma part développé trois briques de cette architecture :

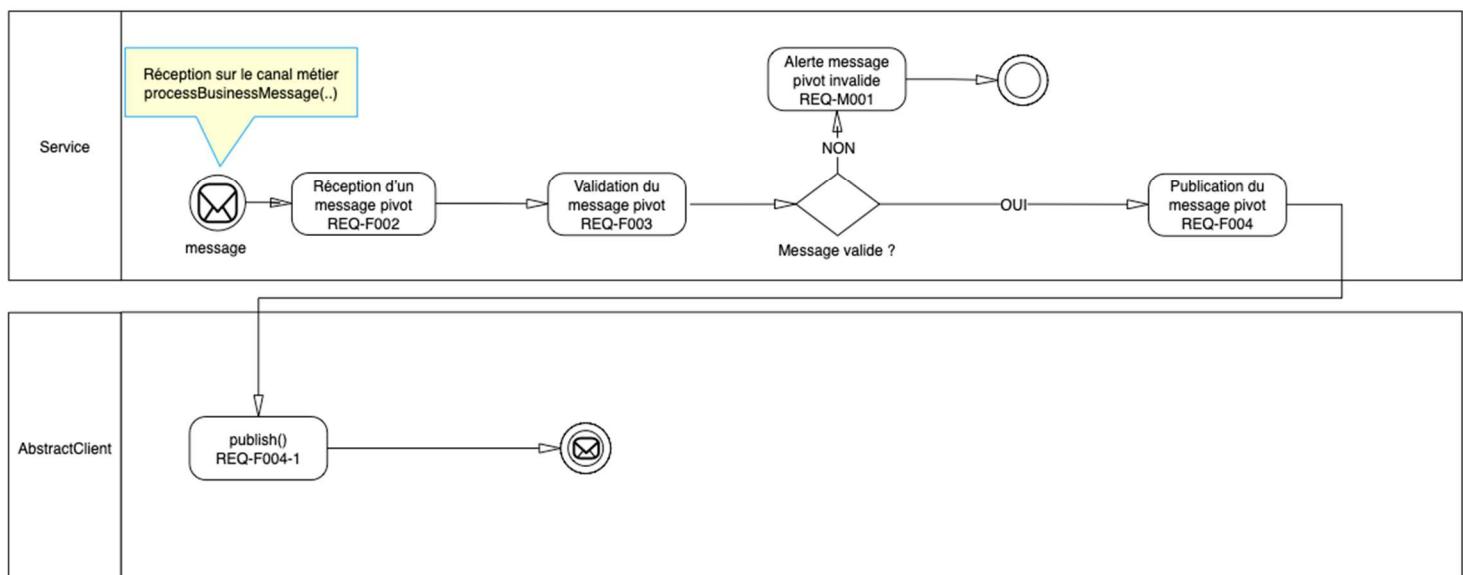
- Le demi-flux entrant fromSTPV
- Le demi-flux entrant fromTCP
- Le service technique tsWebAdmin (en cours)

Demi flux entrant fromSTPV

Ce service de demi-flux entrant reçoit les messages STPV pivots émis par les CRNA (flux BrutusIP@CRNA).

Ses fonctions sont les suivantes :

- Réception des messages pivots



- Validation des messages pivots
- Publication des pivots valides sur l'orchestration dédiée au CRNA émetteur

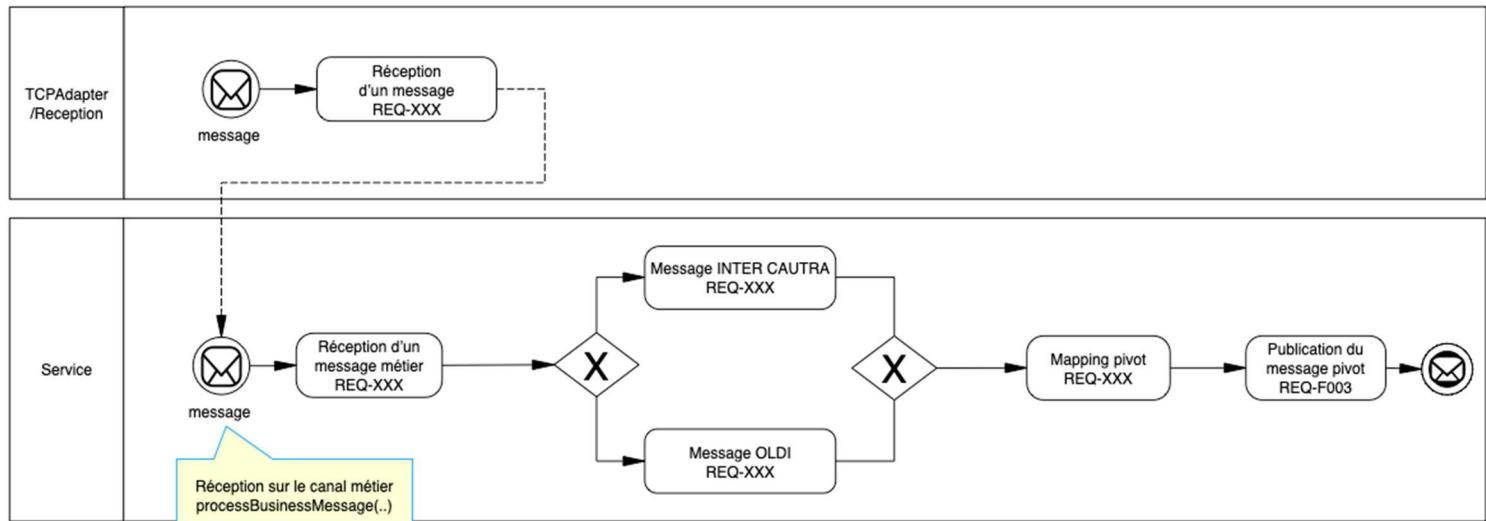


Demi flux entrant fromTCP

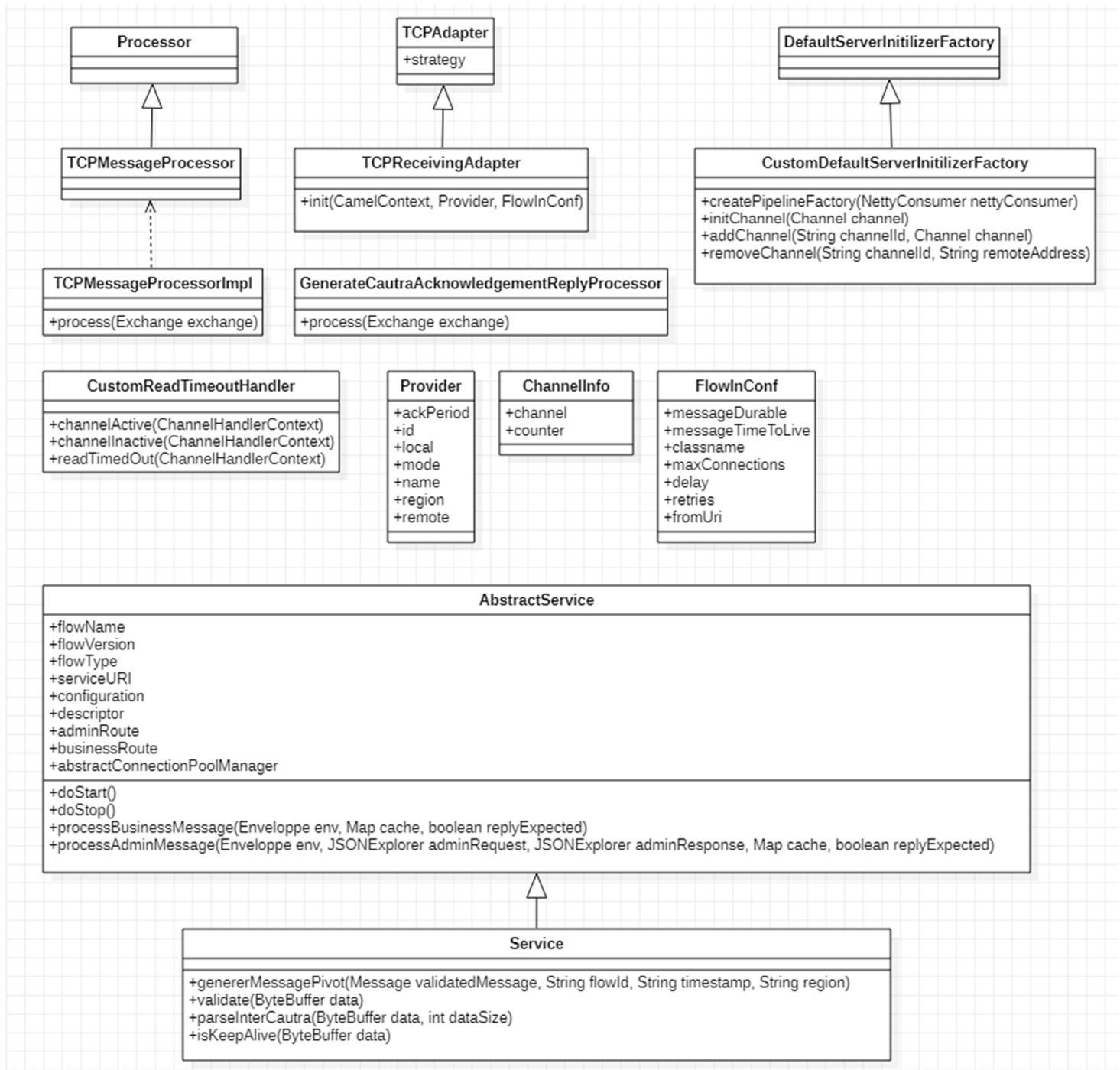
Le service de demi-flux entrant fromTCP reçoit les messages STPV Inter CAUTRA et OLDI émis depuis ATM2 sur TCP.

Ses fonctions sont les suivantes :

- Démarrer l'adapter TCP pour chaque fournisseur configuré. La connexion est établie en mode client (à l'initiative du demi-flux) ou en mode serveur (à l'initiative du fournisseur)
- Sur réception d'un message par l'adapter TCP, celui-ci est transmis au traitement du demi-flux par la classe d'implémentation de l'adapter.
- Le traitement standard du demi-flux entrant s'exécute :



- Mapping du message vers le pivot
- Publication du pivot sur l'orchestration dédiée au fournisseur STPV





Projet ESB – Joan Séverin



Service technique tsWebAdmin

Ce service technique est une application back et front.

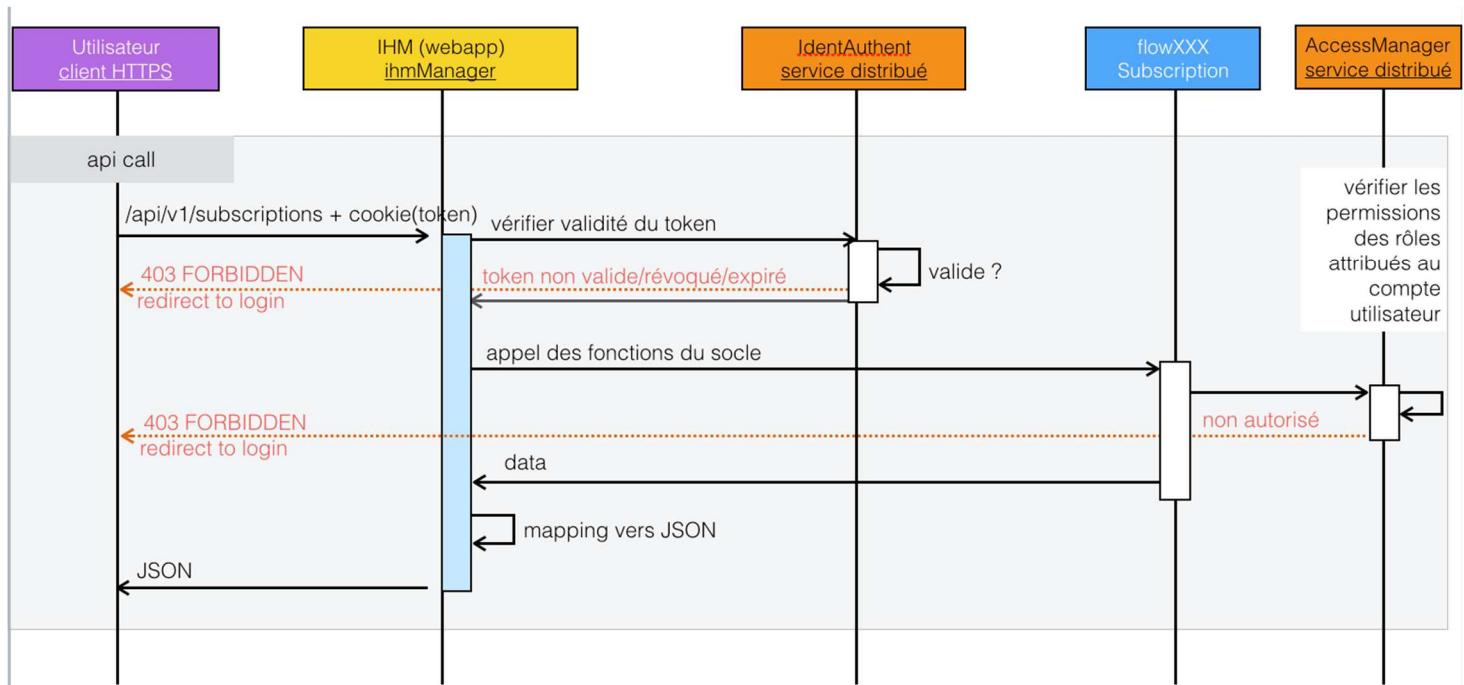
Partie Back-End

La partie back a deux fonctions :

- Exposer une API REST qui offre des fonctionnalités génériques de gestion d'abonnements :

Authentifier l'utilisateur et vérifier ses droits
Lister les abonnements
Créer des abonnements (sans les démarrer)
Modifier des abonnements
Démarrer des abonnements
Arrêter des abonnements (sans les supprimer)
Supprimer des abonnements

- Exposer sur un serveur web une IHM de gestion des abonnements qui utilise l'API.



Partie Front-End

L'IHM est partiellement développée sous plusieurs versions, uniquement . Une version sous jQuery qui utilise le framework Backbone, qui simule une architecture MVC sur du front-end, ainsi qu'une version sous Angular. La version jQuery est la première à émerger, la version Angular est proposée, mais lorsque le projet d'IHM est de nouveau à l'ordre du jour, il est décidé que c'est la version jQuery qui sera aboutie.



Projet ESB – Joan Séverin



En voici la maquette initiale.

DSNA Cloud Services
Abonnements au bus de services
User icon
Power icon

☰
Tous les CRNA
+

Statut	Abonnement	Topic	Fournis...	Connect...	Consumer	Action ▾
☰ CRNA-N						
☰						
✓	A14 abonnement CRN...	STPV_INTER_CAUTR...	provider...	TCP/Int...	consumer-crna-n	<input type="checkbox"/>
✓	B22 abonnement CRN...	STPV_INTER_CAUTR...	provider...	TCP/Int...	consumer-crna-n	<input type="checkbox"/>
!	C85 abonnement CRN...	STPV_INTER_CAUTR...	provider...	TCP/Int...	consumer-crna-n	<input type="checkbox"/>
▶	D34 abonnement CRN...	STPV_INTER_CAUTR...	provider...	TCP/Int...	consumer-crna-n	<input type="checkbox"/>
✓	E51 abonnement CRN...	STPV_INTER_CAUTR...	provider...	TCP/Int...	consumer-crna-n	<input type="checkbox"/>

Et la proposition que j'en fais suite à la réévaluation du besoin client :



Projet ESB – Joan Séverin



logo

Titre

CRNA connectés

- CRNA-Est**
- CRNA-Nord
- CRNA-Ouest
- CRNA-Sud-Est
- CRNA-Sud-Ouest

🔍 Rechercher...

CRNA-Est

Diagnostic

Abonnements

Statut	Abonnement	Topic	Fournisseur	Connecteur	Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
!	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
✓	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
▶	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer
▶	nom_Abonnement	nom_Topic	nom_Fournisseur	nom_Connecteur	nom_Consumer

logo

Titre

CRNA connectés

- CRNA-Est**
- CRNA-Nord
- CRNA-Ouest
- CRNA-Sud-Est
- CRNA-Sud-Ouest

🔍 Rechercher...

CRNA-Est

Diagnostic

Création d'un abonnement

Général
Filtres
Extensions

Abonnement	nom_abonnement
Topic	nom_topic
Région	nom_region
Fournisseur	nom_fournisseur
Flux	nom_connecteur
Type de flux	nom_consumer
Version	nom_fournisseur
Consumer	nom_connecteur
Message	nom_consumer

Annuler
Création



IMPLEMENTATION

Flux STPV InterCautra ; version BrutusIP@DTI

- Génération des logs demi-flux entrant

Les logs sont pris en charge via un framework fmkLogCollect développé en interne avant mon arrivée. Il s'utilise en instanciant un objet LogManager. Les logs sont ainsi envoyés vers le broker. Pour en garder une trace écrite dans un fichier de log sur le disque dur, il faut lancer un service technique dédié à cette tâche, le tsLogCollect.

Le LogManager est un bean configuré dans le Camel-Context.xml :

```
<bean class="org.dsna.esb.framework.logCollect.LogManager" id="logManager">
    <constructor-arg index="0"
        value="file:///${isoda.home}/flows/stpv/interCautra/v1/fwStpvInterCautraFromSTPV-
        logger.json"/>
</bean>
```

On l'instancie ensuite dès le lancement du flux (classe Service, méthode doStart())

```
// extraction du LogManager depuis le Camel Context
logManager = this.getCamelContext().getRegistry().lookupByNameAndType("logManager",
LogManager.class);
long defaultTimeToLive = logManager.getConfiguration().getDefaultTimeToLive();

// Injection de la strategie d'envoi des logs
logManager.setLogSenderStrategy(new LogSenderStrategy() {

    // methode appelee à chaque emission de log vers le broker
    public void execute(String uri, LogMessage logMessage) throws Exception {
        MessageSender sender = null;

        // recuperation d'un sender depuis le pool
        sender = logConnectionPool.getProducer();
```



```

// Envoi du message de log
sender.distribute(uri, new Enveloppe(logMessage), true, defaultTimeToLive);

// libération du sender dans le pool
if (sender != null)
    sender.release();
}

});

// Obtention d'un logger du framework LogCollect
this.log = LogManager getLogger(Service.class);

```

Cette méthode de log est commune à l'ensemble des briques du flux.

Demi-flux entrant fromSTPV

Le demi-flux entrant fromSTPV reçoit les messages des différents CRNA, et donc des messages qui sont envoyés et traités par le flux BrutusIP@CRNA. Par conséquent son implémentation est relativement simple.

- Réception d'un message pivot

Les messages envoyés vers fromSTPV sont reçus sur sa queue \$.from avec un format d'enveloppe technique (classe Enveloppe du framework d'orchestration), qui permet de stocker une liste d'entêtes et un payload.

- Validation du message pivot

Lors de la réception du message pivot, et avant de le publier sur la route du flux d'orchestration, il faut en analyser les entêtes et en vérifier la validité.

- Publication du message pivot

Avant publication, le message doit être enrichi. Cela correspond au rajout d'entête pour la région du CRNA émetteur, ainsi qu'à la récupération de données de configuration spécifique au demi-flux pour les fournir en paramètres, tels que la durée de vie du message, son caractère persistant, le type de flux, ou encore sa version.



Projet ESB – Joan Séverin



La publication se fait en utilisant la classe AbstractClient, qui gère les appels request/reply ou publish vers les services distribués en automatisant l'implémentation du pattern haute dispo en fonction du mode d'exécution du service.

Demi-flux entrant fromTCP

Plus compliqué, ce demi-flux entrant reçoit ses messages de fournisseurs en protocole TCP. Il a donc fallu développer un adaptateur TCP.

- Multi-fournisseurs

Le service de demi-flux entrant doit supporter plusieurs fournisseurs connectés simultanément en mode serveur et/ou en mode client.

- En mode serveur l'adaptateur expose le serveur TCP, qui attribue une socket asynchrone à chaque consommateur qui se connecte.
- En mode client l'adaptateur établie une connexion vers chaque partenaire distant.

Cela est rendu possible par la classe Service, point d'entrée du demi-flux, qui instancie un TCPAdapter pour chaque fournisseur dans sa configuration spécifique. Chacune de ces instances vérifie, également dans la configuration spécifique, son mode, serveur ou client.

- Connexion d'un fournisseur

Les « connexions » sont démarrées au lancement du service selon la logique suivante :

- Pour tout fournisseur
 - Si mode server, alors démarrage d'un serveur TCP à l'adresse local (configuration spécifique).
 - Si mode client, alors démarrage d'une connexion TCP à l'adresse remote (configuration spécifique).

La connexion est réalisée grâce à la librairie Netty de Camel, qui permet la mise en place de sockets client et serveur. La connexion se fait via du DSL dans la classe TCPAdapter.



```

public void start(CamelContext context) throws Exception {
    NettyConfiguration nettyTcpConf = createNettyTcpConf(this.configurationProvider, delay, retries, true);
    NettyComponent nettyComponent = context.getComponent("netty4", NettyComponent.class);
    NettyEndpoint nettyEndp = new NettyEndpoint(null, nettyComponent, nettyTcpConf);
    RouteBuilder builder = new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from(nettyEndp)
                .routeId(String.format("route-%s", TCPAdapterIN.this.configurationProvider.getId()))
                .process(businessMessageProcessor)
                .process(messagePostProcessing);
        }
    };
    context.addRoutes(builder);
}

```

- Détection de la perte de connexion d'un fournisseur

La classe CustomServerInitializerFactory, instanciée par la classe TCPAdapter pour la connexion du fournisseur, va avoir la responsabilité de cette connexion. La bibliothèque Netty donne accès à un grand nombre de handlers, qui sont des sous-interfaces de la connexion. Un handler “ReadTimeOutHandler”, en lui passant en paramètre un temps T, va automatiquement vérifier l'état de la connexion tous les T. Nous allons également nous en servir pour l'envoi de messages protocolaires.

```

// ajout du handler de detection de perte de connexion
// [REQ-F102-1] Détection de la perte de connexion d'un fournisseur
// [REQ-F106-2] Émission d'un message de vie protocolaire
ChannelHandler timedHandler = new CustomReadTimeoutHandler(logManager, this.getProvider().getAckPeriod(),
    TimeUnit.SECONDS, this);

channel.pipeline().addLast("timedHandler", timedHandler);

super.initChannel(channel);
}

```



Projet ESB – Joan Séverin



- Reconnexion automatique d'un fournisseur

Pour les connexions en mode serveur, la reconnexion est à l'initiative du fournisseur. L'adaptateur TCP doit accepter les tentatives de reconexions d'un fournisseur si celui-ci n'est pas déjà connecté, dans la limite du nombre maximum de connexions simultanées établies.

Pour les connexions en mode client, la reconnexion est à l'initiative de l'adaptateur TCP. Toutes les connexions établies en mode client doivent être reconnectées automatiquement :

- Sur interruption de la connexion.
- Au démarrage de l'adaptateur TCP (lui-même démarré par la méthode doStart() du demi-flux).

Pour chaque connexion, les modalités de reconnexion automatique sont les suivantes :

- Le nombre de tentatives limitées à \$.connections.retries sur l'intervalle [0, 10].
 - 0 indique qu'aucune tentative de reconnexion n'est exécutée.
 - 3 tentatives sont exécutées par défaut.

Le délai d'attente entre chaque tentative de reconnexion est défini dans la configuration spécifique du flux.



Lors de linstanciation par la classe TCPReceivingAdapter dune configuration Netty, composant nécessaire pour létablissement de la connexion avec le fournisseur :

```
// [REQ-F102-2] Reconnexion automatique d'un fournisseur
if (conf.getRetries() == 0) {

    nettyConf.setReconnect(false);

} else {

    nettyConf.setReconnect(true);
    nettyConf.setReconnectInterval(conf.getDelay());
}

this.customServerInitializerFactory = new CustomServerInitializerFactory(nettyConf, provider, conf,
    this.logManager);
nettyConf.setServerInitializerFactory(customServerInitializerFactory);

return nettyConf;
```

- Limitation du nombre de fournisseurs

La limite de fournisseurs pour chacun desquels la classe Service va instancier un adaptateur TCP est défini dans la configuration spécifique du flux.

- Identification dun fournisseur

Chaque fournisseur est identifié et conservé en couple avec sa connexion. Au niveau de TCPReceivingAdapter, cette map est un attribut qui est alimentée à chaque connexion.

```
// [REQ-F104] Identification d'un fournisseur
private Map<String, NettyEndpoint> providersEndpoints;
```



Projet ESB – Joan Séverin



```
public void init(CamelContext camelContext, Provider provider, FlowInConf conf) throws Exception {
    NettyConfiguration nettyTcpConf = createNettyTcpConf(provider, conf);
    NettyComponent nettyComponent = camelContext.getComponent("netty4", NettyComponent.class);
    NettyEndpoint nettyEndp = new NettyEndpoint(null, nettyComponent, nettyTcpConf);

    providersEndpoints.put(provider.getId(), nettyEndp);

    createTCPProcessingRoute(camelContext, nettyEndp, this.getStrategy());

    TCPMessageProcessorImpl strategy = (TCPMessageProcessorImpl) getStrategy();
}
}
```

- Protocole entrant

Le protocole est défini au moment de la création de la configuration Netty pour l'établissement de connexion.

- Emission d'un message de vie protocolaire

Le message de vie produit a le format suivant : 5 octets composé d'un numéro de message valorisé à 0, et une taille du message valorisée à 5 (octets).

Le mécanisme d'émission du message de vie réutilise le ChannelHandler utilisé pour la détection de perte de connexion d'un fournisseur.



Projet ESB – Joan Séverin



```

@Override
protected void readTimedOut(ChannelHandlerContext ctx) throws Exception {
    log.debug("READ_TIMED_OUT", "Read from consumer timed out",
        new KeyValue<>(KevLog.SERVICE, fwStpvInterCautraFromTCP),
        new KeyValue<>(KevLog.FLOW, "stpv"),
        new KeyValue<>(KevLog.FLOW_TYPE, "interCautra"),
        new KeyValue<>("PROVIDER_HOST", ctx.channel().remoteAddress()),
        new KeyValue<>("PROVIDER_NAME", this.serviceInitializerFactory.getProvider().getName()));

    byte[] keepalivemessage = new byte[5];
    keepalivemessage[0] = 5;
    if (ctx.pipeline().channel().isActive()) {
        try {
            ctx.pipeline().writeAndFlush(Unpooled.copiedBuffer(keepalivemessage)).get();
            log.info("LIFE_MESSAGE_SENT", "life message sent to the provider id = " + this.serviceInitializerFactory.getProvider().getId(),
                new KeyValue<>(KevLog.SERVICE, fwStpvInterCautraFromTCP),
                new KeyValue<>(KevLog.FLOW, "stpv"),
                new KeyValue<>(KevLog.FLOW_TYPE, "interCautra"),
                new KeyValue<>("PROVIDER_HOST", ctx.channel().remoteAddress()),
                new KeyValue<>("PROVIDER_NAME", this.serviceInitializerFactory.getProvider().getName()));
        } catch (Exception e) {
            logger.error("I/O operation failed for channel " + ctx.pipeline().channel().remoteAddress(), e);
        }
    }
}
}

```

- Sélection des fournisseurs à notifier

La sélection du fournisseur est automatisée par le ChannelHandler.

- Sélection du canal fournisseur

La sélection du canal fournisseur est automatisée par le ChannelHandler.

- Réception d'un message métier

La réception se fait depuis l'endpoint Netty, on définit dans la classe d'entrée Service une stratégie de traitement des messages. Il s'agit d'une implémentation spécifique à notre flux.

La stratégie consiste à mapper le message reçu dans une enveloppe technique.



Projet ESB – Joan Séverin



```

@Override
public void process(Exchange exchange) throws Exception {

    synchronized (lock) {

        ByteBuf tcpPacketByteBuf = exchange.getIn().getBody(ByteBuf.class);
        tcpPacketByteBuf.readableBytes();
        ByteBuffer payload = ByteBuffer.allocate(tcpPacketByteBuf.readableBytes());
        payload.put(tcpPacketByteBuf.nioBuffer());
        payload.position(0);

        // Réinitialisation de l'index du buffer pour sauter l'entete (RFC-1006)

        if (payload.remaining() > 5) {
            payload.position(5);
        } else {
            payload.position(0);
        }

        // le timestamp est à générer, pas à récupérer
        // Long timestamp = payload.getLong() / 1000;
        Long timestamp = System.currentTimeMillis();

        exchange.getIn().getHeader("CamelNettyRemoteAddress", String.class);
        this.provider.getId();

        ByteBuffer data = ByteBuffer.allocate(payload.remaining());
        payload.get(data.array());

        HashMap headersLst = new HashMap<String, Object>();

        headersLst.put("application.ISODA_PROVIDER_ID", this.provider.getId());
        headersLst.put("application.ISODA_REGION", this.provider.getRegion());
        headersLst.put("application.ISODA_MSG_TYPE", "BUSINESS_MESSAGE");
        headersLst.put("application.ISODA_FLOW_TIMESTAMP", new Date(timestamp).toString());

        Enveloppe env = new Enveloppe();
        env.addData(data.array());
        env.addEntetes(headersLst);

        MessageSender sender = null;
    }
}

```

CONFIDENTIEL INDUSTRIE



Projet ESB – Joan Séverin





```

try {
    sender = businessConnectionPool.getProducer();
    sender.distribute(this.conf.getFromUri(), env, this.conf.getMessageDurable(),
                      this.conf.getMessageTimeToLive());
    log.info("BUSINESS MESSAGE DISTRIBUTED",
             String.format("message correctly distributed with the headers PROVIDER_ID : %s, REGION : %s, MSG_TYPE : %s, FLOW_TIMESTAMP : %s",
                           env.getEntete("application_ISODA_PROVIDER_ID"),
                           env.getEntete("application_ISODA_REGION"), env.getEntete("application_ISODA_MSG_TYPE"),
                           env.getEntete("application_ISODA_FLOW_TIMESTAMP")),
             new KeyValue<>(KeyLog.SERVICE, Service.SERVICE_NAME), new KeyValue<>(KeyLog.FLOW, Service.FLOW),
             new KeyValue<>(KeyLog.FLOW_TYPE, Service.FLOW_TYPE),
             new KeyValue<>("PROVIDER_NAME", provider.getId()));
} finally {
    if (sender != null) {
        sender.release();
    }
}

```

- Message métier entrant

Le message envoyé sur la route from du flux, celui-ci est récupéré par la classe Service (méthode processBusinessMessage()). Il est ensuite analysé pour en vérifier la validité et en déterminer la nature : OLDI ou INTER CAUTRA.

- Identification d'un message INTER CAUTRA ou OLDI

Les messages sont des données brutes encodées en binaire. Le protocole RFC1006 impose entre autre des entêtes qui sont codées jusqu'au 5ème octet. L'analyse du reste permet de déterminer la nature du message : OLDI, INTER CAUTRA, ou message de vie.



Projet ESB – Joan Séverin



```

private Message validate(ByteBuffer data) {

    Message message;

    if (data.remaining() < 5) {
        message = new MessageNonIdentifie(data.array());
    } else if (data.remaining() == 5) {
        if (isKeepAlive(data)) {

            message = new MessageCautra(data.array());
        } else {
            message = new MessageNonIdentifie(data.array());
        }
    } else if (data.remaining() >= 6) {
        // data.position(5);

        int dataSize = data.getShort(4);

        if (dataSize != data.remaining()) {}

        int dleStx = data.getShort();

        if (dleStx == 0x1002) {

            message = new MessageNonIdentifie(data.array());
        } else {

            message = new MessageOldi(data.array());
        }
    } else {

        message = parseInterCautra(data, dataSize);
    }
} else {

    message = new MessageNonIdentifie(data.array());
}

return message;
}

```

CONFIDENTIEL INDUSTRIE



Projet ESB – Joan Séverin





Projet ESB – Joan Séverin



- Extraction du titre INTER CAUTRA

Le titre CAUTRA est un entier encodé sur 1 octet en position DE_7.

- Extraction du sous-titre INTER CAUTRA

Le sous-titre CAUTRA est un entier encodé sur 1 octet en position DE_8.

- Mapping vers le pivot

Chaque message métier entrant est publié sur le broker sous forme de message pivot pour y être routé.



```

/**
 * [REQ-F109]
 *
 * Cette méthode fait le mapping d'un pivot valide selon sa nature Oldi ou INTER
 * CAUTRA
 *
 * @param flowId           l'id unique du message à envoyer
 * @param validatedMessage le message métier validé
 * @return le type du message identifié
 */
private Enveloppe genererMessagePivot(Message validatedMessage, String flowId, String timestamp, String region) {
    Enveloppe enveloppe = new Enveloppe(validatedMessage.getData());
    enveloppe.addEntete("application.ISODA_FLOW_ID", flowId);
    enveloppe.addEntete("application.ISODA_CALLER_ID", "/flows/stpv/interCautra/v1/fwStpvInterCautraOrchestration");
    enveloppe.addEntete("application.ISODA_FLOW_TIMESTAMP", timestamp);
    enveloppe.addEntete("application.ISODA_FLOW", getFlowName());
    enveloppe.addEntete("application.ISODA_FLOW_TYPE", getFlowType());
    enveloppe.addEntete("application.ISODA_REGION", region);
    enveloppe.addEntete("application.ISODA_FLOW_VERSION", getFlowVersion());

    if (validatedMessage instanceof MessageOldi) {
        enveloppe.addEntete("application.ISODA_FLOW_MESSAGE", "oldi");
    } else {
        enveloppe.addEntete("application.ISODA_FLOW_MESSAGE", "interCautra");
        Integer titre = ((MessageCautra) validatedMessage).getTitre();
        if (titre != null) {
            enveloppe.addEntete("routing.interCautra.TITRE", titre.toString());
        }

        Integer soustitre = ((MessageCautra) validatedMessage).getSoustitre();
        if (soustitre != null) {
            enveloppe.addEntete("routing.interCautra.SOUS-TITRE", soustitre.toString());
        }
    }

    log.info("BUSINESS MESSAGE MAPPED");
    String.format("message correctly mapped with the headers ISODA_FLOW_ID : %s, "
        + "ISODA_CALLER_ID : %s, ISODA_FLOW_TIMESTAMP : %s, ISODA_FLOW : %s, "
        + "ISODA_FLOW_TYPE : %s, ISODA_FLOW_VERSION : %s, ISODA_REGION : %s, "
        + "| routing.interCautra.TITRE : %s, routing.interCautra.SOUS-TITRE : %s",
        enveloppe.getEntete("application.ISODA_FLOW_ID"),
        enveloppe.getEntete("application.ISODA_CALLER_ID"),
        enveloppe.getEntete("application.ISODA_FLOW_TIMESTAMP"),
        enveloppe.getEntete("application.ISODA_FLOW"),
        enveloppe.getEntete("application.ISODA_FLOW_TYPE"),
        enveloppe.getEntete("application.ISODA_FLOW_VERSION"),
        enveloppe.getEntete("application.ISODA_REGION"),
        enveloppe.getEntete("application.routing.interCautra.TITRE"),
        enveloppe.getEntete("application.routing.interCautra.SOUS-TITRE")),
        new KeyValue<>(keyLog.SERVICE, Service.SERVICE_NAME), new KeyValue<>(keyLog.FLOW, Service.FLOW),
        new KeyValue<>(keyLog.FLOW_TYPE, Service.FLOW_TYPE));
}

return enveloppe;
}

```



Projet ESB – Joan Séverin



- Publication du message pivot

Point d'entrée, mais également de sortie, la classe Service, dans son traitement du message métier entrant (méthode processBusinessMessage()), utilise un AbstractClient pour publier le pivot sur la queue du service d'orchestration.

```
abstractClient.publish(uriOrchestration, envToSend, timeToLive, persistent, lstParams);
```

Service technique tsWebAdmin (en cours)

L'implémentation de ce service et de son IHM associée ne sont pas à un stade suffisamment avancé pour être développée dans ce rapport.



Projet ESB – Joan Séverin



TESTS ET VALIDATION

Flux STPV ; version BrutusIP@DTI

Je suis chargé de la rédaction des cahiers de tests du service de flux BrutusIP@DTI. Ces cahiers recouvrent des tests de qualification ainsi que d'intégration.

- Demi-flux entrant fromSTPV

Afin de valider l'ensemble des exigences d'implémentation, j'ai rédigé des tests nominaux avec cas d'erreur. Ils permettent de retracer le demi-flux entrant de bout en bout, activant au fur et à mesure les logs validant que l'exigence est traitée.

Ces tests sont réalisés sous JUnit.

Le test nominal consiste à créer un fournisseur qui va alimenter en message la queue from du demi-flux entrant, et un consommateur qui va réclamer le message après traitement, jouant le rôle du flux d'orchestration.

Détaillons le tests ci-après :



Projet ESB – Joan Séverin



```

@Test
@Order(1)
public void testProducerConsumer() throws Exception {

    List<Object> dataTest = new ArrayList<Object>();
    byte[] cautraData = createData();
    dataTest.add(cautraData);

    AbstractConnectionPoolManager poolManager =
        new AbstractConnectionPoolManager(ServiceTest.MIN_CONNECTIONS,
                                         ServiceTest.MAX_CONNECTIONS, true, ServiceTest.TIMEOUT, |
                                         ServiceTest.UNIT, ServiceTest.BROKER_USERNAME,
                                         ServiceTest.BROKER_PASSWORD, ServiceTest.BROKER_URL) {
    };

    MessageSender producer = poolManager.getProducer();

    try {

        for (int i = 0; i < NB_MESSAGES; i++) {

            Enveloppe pivot = new Enveloppe();
            pivot.addData(dataTest);
            pivot.addEntetes(headersLst);
            logger.info("message distribue num " + i);
            logger.info("pivot data = " + pivot.getData());
            producer.distribute(ROUTE_FROM, pivot, true, 15000);
        }

    } catch (Exception e) {
        log.debug("MESSAGE-ERROR", "Le message n'est pas parvenu au flux",
                 new KeyValue<>(KeyLog.SERVICE, Service.SERVICE_NAME),
                 new KeyValue<>(KeyLog.FLOW, Service.FLOW),
                 new KeyValue<>(KeyLog.FLOW_TYPE, Service.FLOW_TYPE));
    } finally {
        producer.release();
    }
}

```

Cette partie décrit la création d'un fournisseur de données. La donnée en question, dataTest, est créée à partir d'un fichier métier parsé en tableau de bytes. Les paramètres du test prévoient de pouvoir envoyer un certain nombre de messages, utile pour tester la robustesse du mécanisme.



```

// pause le process 20 sec le temps de voir la console, avant que le consumer ne
// soit
// détruit
Thread.sleep(2000);

MessageConsumer consumer = poolManager.getConsumer();
try {

    List<Enveloppe> response = consumer.receive(ROUTE_TO, NB_MESSAGES, 10000, Enveloppe.class);

    assert (!response.isEmpty());

    for (int i = 0; i < NB_MESSAGES; i++) {

        Enveloppe env = response.get(i);

        logger.info("message recuper num " + i);
        logger.info("message data = " + env.getData());

        assert (env.getEntete("application.ISODA_CALLER_ID")
                .equals("/flows/stpv/interCautra/v1/fwStpvInterCautraFromSTPV"));
        assert (env.getEntete("application.ISODA_FLOW").equals("stpv"));
        assert (env.getEntete("application.ISODA_FLOW_TYPE").equals("interCautra"));
        assert (env.getEntete("application.ISODA_ROUTE_TO").equals(ROUTE_TO));

        // On vérifie si le body du message est pas modifié entre producer et consumer
        logger.info("===== LOG LOCAL =====");
        logger.info("variable data = " + dataTest);
        logger.info("response data = " + env.getData());
        //
        assert (env.getData().equals(dataTest));
    }
} catch (Exception e) {
    log.debug("MESSAGE-ERROR", "Aucun message à recuperer " + e.getMessage(),
        new KeyValue<>(KeyLog.SERVICE, Service.SERVICE_NAME),
        new KeyValue<>(KeyLog.FLOW, Service.FLOW),
        new KeyValue<>(KeyLog.FLOW_TYPE, Service.FLOW_TYPE));
} finally {
    consumer.release();
    poolManager.releasePool();
}
}

```

Comme détaillé en commentaires, on met le process en pause afin d'avoir le loisir d'analyser la console IHM mise à disposition par Artemis. Sur celle-ci on peut notamment avoir une vue d'ensemble des routes mises en service, et sur celles-ci, le nombre de consommateurs et de messages correctement ajoutés.



On vérifie enfin que le mapping du message pivot a été correctement effectué, que la donnée en elle-même n'a pas été modifiée.

On effectue ensuite le même test en transmettant un message dont il manque un header différent pour chaque message. On vérifie ensuite, à condition que le service de collecte des logs soit correctement lancé, qu'une alerte concernant la validité du message est bien remontée.

- Demi-flux entrant fromTCP

La logique de test du demi-flux entrant fromTCP est la même que le demi-flux entrant fromSTPV, à la différence du point d'entrée. Si fromSTPV recevait ses messages directement sur queue du broker, fromTCP doit d'abord établir une connexion distante sur protocole TCP.

Le test est donc quasi identique dans sa structure (envoi de données, consommation de celles-ci, et vérification que leur traitement a été effectué comme escompté).

La grande différence va donc être la façon d'envoyer les données, puisque le demi-flux doit pouvoir établir une connexion TCP en mode serveur (à l'initiative du fournisseur), ou en mode client (à l'initiative du demi-flux) :

Demi-flux en mode client :

```
// Creation de connexion avec le serveur TCP et envoi d'un message cautra
byte[] sendTcpMessage() throws UnknownHostException, IOException, InterruptedException {
    Socket socket1 = new Socket("127.0.0.1", 1027);
    Socket socket2 = new Socket("127.0.0.1", 1028);
    Socket socket3 = new Socket("127.0.0.1", 1027);
    socket3.close();
    Socket socket4 = new Socket("127.0.0.1", 1027);

    byte[] dataToSend = createData();

    socket1.getOutputStream().write(dataToSend);
    socket2.getOutputStream().write(dataToSend);

    return dataToSend;
}
```

Demi-flux en mode serveur :



Projet ESB – Joan Séverin



```
// Creation de connexion avec le client TCP et envoi d'un message cautra
ServerSocket server1 = new ServerSocket(1028);
Socket socket1 = server1.accept();
ServerSocket server2 = new ServerSocket(1028);
Socket socket2 = server2.accept();

byte[] data = createData();
socket1.getOutputStream().write(data);
socket2.getOutputStream().write(data);
|
socket1.close();
server2.close();
}
```

Afin que le test puisse être par la suite plus facilement automatisable, j'ai développé un script jar exécutable en ligne de commande qui simule un fournisseur de données. Le programme accepte comme arguments : le mode (serveur ou client), le port de connexion, et le nombre de messages à envoyer.

- Service technique tsWebAdmin

L'implémentation de ce service et de son IHM associée ne sont pas à un stade suffisamment avancé pour être développée dans ce rapport.



Projet ESB – Joan Séverin



BILAN DE PROJET

DIFFICULTES RENCONTREES

Contexte sanitaire et télétravail

Comme tout un chacun durant cette année 2020, j'ai dû m'adapter au contexte de crise sanitaire. La principale difficulté fut que j'ai intégré l'entreprise, le client, le projet et l'équipe quasi exclusivement en télé-travail. Je n'ai rencontré mes collègues que plusieurs semaines après avoir démarré, lorsqu'on a été autorisés à revenir sur place à raison de deux jours par semaine. Mes jours en présentiel impliquant cependant un bureau où j'étais seul toute la journée, cela ne représentait pas une franche amélioration.

La volonté à partir d'octobre 2020 de travailler davantage en mode agile a permis de dynamiser le quotidien en multipliant les échanges avec les collègues.

Contexte métier

Le contexte métier a été particulièrement difficile à assimiler. Le manque d'accompagnement (j'ai eu la première présentation sur le sujet mardi 19/01/21) à mon arrivée ne m'a pas permis de saisir rapidement les enjeux du projet, ni sa place et son usage dans le corps métier de la DSNA. Les sources d'informations à ma disposition se sont révélées être des notes de service, d'architecture, à destination d'un public de fonctionnaires déjà initiés. Saisir le sens des *très nombreux* sigles ne fut pas tâche aisée.

Complexité du projet

pour un profil junior (passation de compétences sur deux trois jours, turnover)

D'un point de vue technique, le projet est complexe pour un profil tel que le mien. Le travail sur un ESB a demandé une adaptation et un apprentissage rapide de tout un environnement technologique que je n'avais jamais rencontré.



Projet ESB – Joan Séverin



Lors de mon arrivée, le développeur le plus expérimenté de l'équipe sur l'ESB était sur le départ. La passation de compétences a duré pour moi deux jours, durant lesquels j'ai reçu des informations sur le fonctionnement local du broker et du serveur d'application. Mais le manque de recul par rapport à l'ensemble, théorique et pratique, du concept d'ESB m'a posé des difficultés lorsque j'ai commencé à développer.

Contexte politique interne

Le projet a souffert de nombreux départs, côté sous-traitants et fonctionnaires. Ma propre tutrice qui devait m'encadrer durant mes neuf mois en entreprise a souhaité partir vers un autre projet en septembre 2020, après être arrivée peu avant moi. Elle était en outre responsable technique sur le projet et n'a pas été remplacée sur ce poste. Ma nouvelle tutrice est arrivée en octobre sur un poste de chef de projet, poste occupé précédemment par le product-owner, parti également en septembre.

Besoin client pas saisi pour l'IHM

Début décembre 2021, on m'a donné la tâche de développer l'IHM de gestion des abonnements. Le projet en était à un stade où une version prototype avait déjà été proposée, puis mise en pause. J'ai donc travaillé avec l'architecte ESB qui avait proposé le prototype, pour récupérer et adapter le code. Néanmoins lors de la démonstration du 08 janvier 2021 faite devant le client à l'origine de la demande d'IHM, celui-ci a insisté sur le fait que le prototype ne correspondait pas à sa demande.

Après débriefing, il semble que le besoin client n'a pas été correctement sauvegardé, ce qui implique maintenant de nouvelles phases de brief client, une refonte de la maquette, et un retour au développement.

ETAT D'AVANCEMENT DU PROJET

- Le projet BrutusIP subit un retard de 8 mois.
- Le développement de la partie BrutusIP@CRNA est terminé.
- Le développement du service de flux de la partie BrutusIP@DTI est terminé.



Projet ESB – Joan Séverin



- Il reste les tâches :
 - d'intégration
 - de déploiement sur les environnements cibles
 - de connexion des deux flux
 - de déploiement de l'IHM de gestion des abonnements

BILAN ET PERSPECTIVES

BILAN PERSONNEL

Assurément l'expérience de ces neuf mois en entreprise a été formatrice. Le contexte a été particulièrement difficile d'un point de vue personnel (télétravail avec un enfant en bas âge), métier (aridité du milieu du contrôle du trafic aérien) et technique (technologie difficile d'accès pour mon niveau).

Néanmoins la proposition d'embauche en CDI et l'évaluation de ma tutrice prouve que j'ai pu faire mes preuves sur le projet, monter en compétences et gagner en autonomie.

Bien que j'en attende encore beaucoup des formations technique et métier prévues prochainement, j'ai une vision plus large et claire sur le projet ESB à la DTI.

Je me suis amélioré sur l'organisation de mon travail, et ma capacité à trouver les réponses à mes interrogations. Le contexte du projet BrutusIP@DTI n'a pas laissé beaucoup de place à la conception, et c'est dommage. Développer à partir d'une spécification fonctionnelle, même précisément détaillée, ne donne pas de vision d'ensemble sur le composant à développer.

PERSPECTIVES



Projet ESB – Joan Séverin



Le projet ESB a en principes de nombreux chantiers à venir sur les trois prochaines années. Des choses intéressantes sur le moyen terme, comme la migration de l'ESB sur Cloud.

D'un point de vue métier également, nous sommes à un moment charnière où les systèmes informatiques du contrôle aérien sont en pleine migration. Ces évolutions sont déjà anticipées et prises en compte, mais un changement majeur tel que le retrait du CAUTRA au profit de co-flight dans le cadre de 4-flight n'a pas encore d'impact sur les applications que nous développons.

Je me réjouis également de savoir qu'au delà de mes compétences à traiter tel ou tel projet, on prendra en compte mon appétence pour le sujet, tel que ce fut le cas lors de mon attribution de l'IHM.