AGENCE
DU NUMÉRIQUE
EN SANTÉ

Pilote Hub Santé & connecteurs

Dossier des Spécifications Techniques (DST)

Statut : En cours | *Classification : Restreinte* | *Version : v1.1*

Agence du Numérique en Santé
9, rue Georges Pitard – 75015 Paris
T. 01 58 45 32 50
esante.gouv.fr

Historique du document

Version	Rédigé par		Vérifié par		Validé par	
0.1	Romain Fouilland	Le 20/01/23				
	Motif et nature de la modification : Création du document					
0.1.1	Romain Fouilland	Le 06/02/2023				
	Motif et nature de la modification : Prise en compte des retours lors des 1-to-1 de présentations des spécifications					
0.1.2	Romain Fouilland	Le 24/02/2023				
	Motif et nature de la modification : Corrections (dates, génération de certificats) et ajout (heartbeat)					
1.0.0	Romain Fouilland Benjamin Bonche	Le 05/05/2023				
	Motif et nature de la modification : Stabilisation des spécifications suite au développement et déploiement du Hub					
1.1	Benjamin Bonche Romain Fouilland	Le 20/07/2023				
	Motif et nature de la modification : Corrections, mise à jour des règles de nommage du Hub, ajout de schémas (gestion des erreurs), ajout des règles d'autorisation					

SOMMAIRE

1. Introduction 3
1.1. Objet du document 3
1.2. Contexte 3
1.3. Versions et approche itérative 3
1.4. *Repository* GitHub partagé 3
1.5. Lexique 4
2. Architecture 5
2.1. Fonctionnement général 5
2.1.1. Présentation du maillage de HubEx 5
2.1.2. Articulation avec les DSF 5
2.1.3. Orientations techniques macros 5
2.2. Sécurisation des échanges 6
2.2.1. Protocoles 6
2.2.2. Obtention des certificats 7
2.2.3. Traçabilité 7
2.2.4. Persistance des messages 8
2.2.5. Ouvertures d'accès 8
3. Interfaçage 9
3.1. Connexion 9
3.2. Entrée et sortie des messages 9
3.3. Flot des messages et acquittements 10
3.3.1. Informer du transfert de responsabilité – acquittement technique 10
3.3.2. Informer de la réception – acquittement de réception finale 11
3.3.3. Informer du traitement – message fonctionnel 11
3.3.4. Gestion des erreurs 11
3.3.5. Schémas récapitulatifs 12
3.4. Messages 13
3.4.1. Format des messages : JSON et XML 13
3.4.2. Identifiants et logique de routage 14
3.4.3. Enveloppe EDXL-DE 14
3.4.4. Adressage destinataire 17
3.4.5. Schémas et exemples de messages 17
4. Etapes d'implementation 18
5. Evolutions techniques 18
6. Développements à réaliser 19
7. Annexes 20
7.1. Génération d'un certificat 20

1. INTRODUCTION

1.1. Objet du document

Ce document décrit en détail les principes et les spécifications techniques permettant de connecter et de gérer les échanges entre :

- Le « Hub Santé » de l'ANS
- Les logiciels d'éditeurs (appelés « Clients ») se conformant au DST : LRM (Logiciels de Régulation Médicale), logiciels de gestion des hélicoptères, logiciels de tablettes, …

1.2. Contexte

Dans un cadre d'interopérabilité des services d'urgences aussi bien Santé qu'inter-forces, l'ANS souhaite construire une solution d'échanges de messages appelée « Hub Santé » et la connecter avec les logiciels des éditeurs Santé. Afin de connecter ces Clients avec le Hub Santé, l'ANS partage les présentes spécifications.

1.3. Versions et approche itérative

Les travaux et spécifications pour mettre en œuvre un pilote de Hub Santé et de connecteurs associés sont réalisés selon une approche itérative et agile[1]. La démarche est constituée de petites étapes consécutives permettant de délivrer rapidement des fonctionnalités, valider progressivement les travaux réalisés et de s'adapter aux retours, aux contraintes d'implémentation et à l'expérience acquise lors de la réalisation.

Pour cadrer ces itérations, nous avons introduit un système de versions correspondant à des « sprints »[2] et permettant de suivre les différentes étapes du projet. Les sprints concernent le Hub Santé au sens large et n'impliquent donc pas tous des modifications côté Client. La liste des différentes versions réalisée est la suivante :

- v0.1 = Messages vides et acquittements techniques
- v0.2 = + Accusés de bonne réception
- v0.3 = + Sécurisation par certificats et mTLS
- v0.4 = + Transcodage et spécifications AsyncAPI
- v0.5 = + Hub en Spring Boot et messages CISU
- v0.6 = + Transcodage et validation, authentification par certificat
- v0.7 = + Migration Cloud et gestion JSON/XML
- v0.8 = + Web (landing & JSV), stabilisation et tests

Dans cette approche itérative et composée de *sprints* successifs, les présentes spécifications ne sont pas finales et sont amenées à évoluer en étant à la fois corrigées ou précisées au besoin et également étendues pour couvrir plus de fonctionnalités et de cas d'usage à mesure que les travaux avancent.

1.4. *Repository* GitHub partagé

Afin de partager les travaux et les développements réalisés, un *repository* de code privé a été créé sur le GitHub de l'ANS[3] à l'adresse suivante : https://github.com/ansforge/SAMU-Hub-Sante.

Le GitHub sert de centralisation pour

[1] Nous cherchons notamment à suivre une approche inspirée par l'*Agile Manifesto* https://agilemanifesto.org/iso/fr/principles.html
[2] https://www.atlassian.com/agile/scrum/sprints
[3] https://github.com/ansforge

- Le moyen de lancer localement et rapidement un RabbitMQ minimal
- Le moyen de lancer facilement les tutoriels[4] Java de RabbitMQ
- Les spécifications techniques de connexion (notamment les différentes versions de ce DST) et de format des messages (AsyncAPI, XSD, …)
- L'implémentation Java du Hub Santé et le moyen d'en lancer une instanciation locale

Des modèles de code d'un consommateur et d'un producteur Java implémentant les différentes versions de notre approche (les *releases*[5] indiquant la dernière version implémentée et disponible) et permettant de mutualiser les travaux (voir les points 1 et 2 de la partie

[4] https://www.rabbitmq.com/getstarted.html
[5] https://github.com/ansforge/SAMU-Hub-Sante/releases

- Développements à réaliser pour plus de détails). En fonction des besoins et contributions, l'ajout d'autres langages est envisageable.

Le GitHub est pour le moment privé et les utilisateurs doivent être ajoutés manuellement pour accéder au contenu. Pour cela, il vous suffit de transmettre les identifiants[6] des personnes impliquées chez vous à l'adresse mail hubsante.contact@esante.gouv.fr

1.5. Lexique

Abréviations	Significations
AC	*Autorité de Certification*
AMQP	*Advanced Message Queuing Protocol*
CISU	*Cadre d'Interopérabilité des Services d'Urgence*
CSR	*Certificate Signing Request*
DSF	*Dossier des Spécifications Fonctionnelles : document spécifiant les cas d'usage couverts ainsi que les modèles de données et cinématiques associés*
DST	*Dossier des Spécifications Techniques : document spécifiant le fonctionnement du Hub Santé et les caractéristiques techniques du raccordement au Hub Santé*
EDXL	*Emergency Data Exchange Language*
HubEx	*Plate-forme d'échanges – Hub d'échanges*
MOM	*Middleware Orienté Messages*
PoC	*Proof of concept*
PTP	*Point To Point*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

[6] Si besoin, l'inscription à GitHub est gratuite et accessible au lien suivant https://github.com/signup

2. ARCHITECTURE

2.1. Fonctionnement général

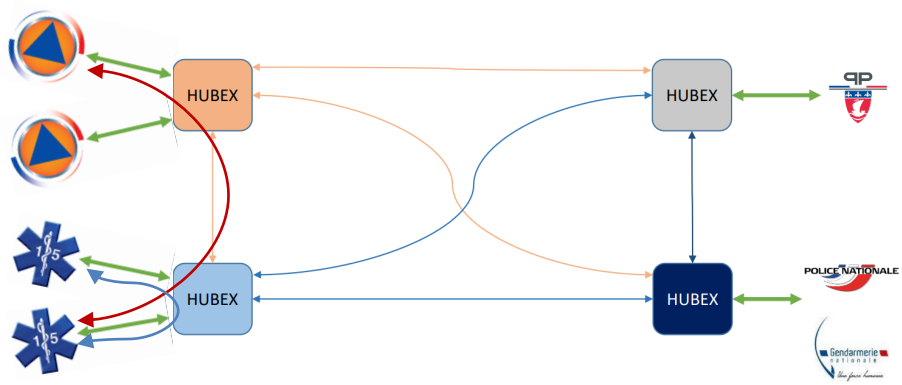
2.1.1. Présentation du maillage de HubEx

Les échanges entre les acteurs de l'urgence seront portés par des Hubs d'échanges (aussi appelés « HubEx ») métiers (un par force) interconnectés en un maillage. Les acteurs seront connectés à leur HubEx métier (et donc également appelés « clients ») et pourront échanger des messages avec n'importe quel autre acteur par l'intermédiaire de leur HubEx métier. Les communautés métiers sont ainsi sous la responsabilité technique de leur HubEx métier.

Les HubEx métiers considérés dans les travaux CISU sont portés par :

- L'Agence du Numérique de Santé (ANS)
- L'Agence du Numérique de la Sécurité Civile (ANSC)
- La Gendarmerie Nationale (GN)
- La Police Nationale (PN)
- La Préfecture de Police de Paris (PP)

Le schéma suivant synthétise ce fonctionnement en maillage de HubEx et la centralisation des acteurs d'une force autour de leur HubEx métier :



2.1.2. Articulation avec les DSF

Les présentes spécifications détaillent le fonctionnement du Hub Santé ainsi que le raccordement au Hub Santé pour des clients Santé. Ce raccordement est le prérequis technique à l'implémentation d'échanges fonctionnels tels que décrits dans des Dossiers de Spécifications Fonctionnelles (DSF) complémentaires et portant notamment sur les échanges 15-NexSIS (*i.e.* entre un LRM Santé et le logiciel national Pompiers NexSIS) comme illustré par la flèche rouge sur le schéma ou les échanges 15-15 (*i.e.* entre deux LRM Santé ou autres Clients Santé) comme illustré par la flèche bleue sur le schéma.

2.1.3. Orientations techniques macros

Messages asynchrones, AMQP et RabbitMQ

Les échanges entre HubEx ou entre un acteur et son HubEx métier (dans le cadre de ces spécifications entre un Client et le Hub Santé) se font via des opérations d'envoi et/ou de réception de messages en mode asynchrone selon une architecture MOM. La solution technique proposée pour le réseau de HubEx à la suite des travaux techniques

du CISU (Cadre d'Interopérabilité des Services d'Urgence) est basée sur le protocole AMQP 0-9-1[7] et son implémentation open-source RabbitMQ[8].

Autorisations : acteurs et messages

L'ajout d'un client sur le Hub Santé se fait par prise de contact auprès de l'ANS[9], production et transmission d'un certificat validé auprès de l'AC utilisée par le Hub Santé[10] et intégration de ces informations dans le Hub Santé. Les acteurs auront le droit d'échanger uniquement avec les acteurs également enregistrés par les différents HubEx[11]. Les échanges autorisés sur le réseau de HubEx seront définis par des DSF précisant notamment les modèles de données attendus et qui seront contrôlés par les HubEx lors des échanges de messages.

Communications : point à point, files fixées et nommage

Le mode de communication est basé sur un mode d'échange point à point (PTP) via des files de messages.

Un client peut donc publier un message à un unique destinataire (c'est à dire un autre client d'un HubEx métier) sur un échangeur d'envoi dédié. Ce même client reçoit ses messages sur des files d'écoute spécifiques. L'ensemble des files sont portées par les HubEx.

Chaque client dispose de 3 files d'écoute selon la typologie des messages reçus :

- « message » pour les échanges fonctionnels
- « ack » pour les acquittements de réception finale
- « info » pour les messages généraux d'informations, alertes et erreurs

La structuration du nom des files est $\{identifiantClient\}.\{typologie\}$ donnant, par exemple, fr.health.samu001.message ou fr.health.samu002.ack[12].

Grace à un mécanisme de routage interne au HubEx, chaque message publié sur l'échangeur d'envoi est transféré sur une file de messages accessible au destinataire. Le message est ensuite récupéré par le client destinataire. Il dispose alors des informations suffisantes pour traiter ce message au sein de son système d'information.

2.2. Sécurisation des échanges

2.2.1. Protocoles

Le HubEx Santé est accessible directement depuis Internet à l'adresse messaging.hub.esante.gouv.fr.

Le protocole AMQPS (AMQP sécurisé par TLS) est utilisé pour sécuriser les services dans les deux sens (Client vers Hub Santé et inversement). Il permet d'assurer la confidentialité et l'authenticité des échanges. Le canal AMQPS est établi avec une authentification mutuelle des services par certificats (aussi appelé mTLS[13]). La connexion doit être établie avec un canal TLS 1.2. RabbitMQ gère directement la sécurisation TLS et mTLS[14] : il autorise et authentifie un client grâce au certificat client présenté.

[7] https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf
[8] https://www.rabbitmq.com/
[9] hubsante.contact@esante.gouv.fr
[10] Le Hub Santé sera initialement sa propre AC afin de faciliter le lancement des travaux côté éditeurs et de permettre de seulement transmettre une CSR. A terme, l'AC utilisée sera l'IGC Santé et les éditeurs devront donc disposer des droits d'administrateurs techniques et générer directement des certificats sur les URL utilisées ce qui pourra ralentir et complexifier le lancement de leurs travaux.
[11] Le raccordement entre HubEx étant progressif et lié au déploiement des HubEx des différentes forces, l'ensemble des acteurs ne sera pas immédiatement accessible aux acteurs Santé raccordés au Hub Santé. Dans un premier temps, seuls les acteurs Santé seront accessibles puis les acteurs Pompiers après le raccordement au Hub Pompiers de NexSIS.
[12] Voir la section Identifiants et logique de routage pour plus de détails sur les identifiants clients.
[13] https://en.wikipedia.org/wiki/Mutual_authentication#mTLS
[14] https://www.rabbitmq.com/ssl.html

A l'établissement de la session TLS, le Hub Santé et le Client doivent présenter un certificat X509 serveur *SERV_SSL* (aussi appelé SSL_SERVEUR) issu de l'IGC Santé. Dans un premier temps et pour faciliter le début des travaux, les éditeurs pourront fournir une CSR à l'équipe du Hub Santé qui leur retournera un certificat signé par l'AC du Hub Santé.

Le Client doit s'assurer que le certificat serveur présenté à l'établissement de la connexion est bien celui du Hub Santé.

Le Hub Santé doit s'assurer que le certificat présenté par le Client fait partie des certificats autorisés pour se connecter au Hub Santé. Le Hub Santé vérifie également que l'identifiant du SAMU envoyé dans les messages et les files de messages concernées (en lecture et écriture) correspondent au certificat identifié pour celui-ci en s'appuyant sur les mécanismes d'authentification et d'autorisation des accès de RabbitMQ[15].

2.2.2. Obtention des certificats

Certificats non IGC Santé

L'annexe Génération d'un certificat détaille la procédure pour générer un certificat auto-signé pour les tests en local et pour générer une CSR à transmettre à l'équipe Hub Santé afin de faciliter les premiers raccordements au Hub Santé en ligne.

Certificats IGC Santé

Le portail des industriels de l'ANS[16] est le point d'entrée pour l'accompagnement technique et la commande de certificats issus de l'IGC Santé grâce aux formulaires « Industriels »[17].

Pour les tests, le formulaire F414 (en démarche simplifiée[18] ou en PDF[19]) permet de commander des cartes (section 4) et des certificats de test (section 5.2). Il est également possible de déclarer des administrateurs techniques (section 5.3) – sur de nouvelles ou d'anciennes cartes – qui pourront ensuite facilement générer des certificats en ligne[20].

Pour la production, le formulaire F413 (en démarche simplifiée[21] ou en PDF[22]) permet de déclarer des administrateurs techniques qui pourront ensuite facilement générer des certificats en ligne[23]. Si vous possédez déjà des droits d'administrateur technique, le formulaire F503 (en démarche simplifiée[24] ou en PDF[25]) permet de mettre à jour les domaines couverts par ces droits.

2.2.3. Traçabilité

La traçabilité des échanges doit être faite à la fois par le Hub Santé et par le Client.

La traçabilité doit être assurée :

- Au niveau de l'échange (réception et émission du message) pour les deux systèmes,
- Au niveau du résultat du traitement du message (pour le système recevant le message uniquement), ce traitement étant fait de façon asynchrone.

[15] https://www.rabbitmq.com/access-control.html
[16] https://industriels.esante.gouv.fr/
[17] https://esante.gouv.fr/index-des-formulaires
[18] https://www.demarches-simplifiees.fr/commencer/f414
[19] https://esante.gouv.fr/sites/default/files/media_entity/documents/F414.pdf
[20] https://pfc.eservices.esante.gouv.fr/pfcng-ihm/authentication.xhtml
[21] https://www.demarches-simplifiees.fr/commencer/f413
[22] https://esante.gouv.fr/sites/default/files/media_entity/documents/F413.pdf
[23] https://pfc.eservices.esante.gouv.fr/pfcng-ihm/authentication.xhtml
[24] https://www.demarches-simplifiees.fr/commencer/f503
[25] https://esante.gouv.fr/sites/default/files/media_entity/documents/F503.pdf

Chaque trace doit être horodatée, et afin de faciliter la corrélation entre les traces du Hub Santé et des différents Clients, le serveur du Client qui enregistre les traces doit être synchronisé sur une source de temps fiable (serveur NTP public par exemple).

Les traces doivent être accessibles en cas de requête spécifique.

La durée de conservation des traces doit être conforme à la réglementation.

2.2.4. Persistance des messages

Chaque message transitant par le Hub Santé est sauvegardé par le cluster RabbitMQ (voir section Entrée et sortie des messages) pendant une durée paramétrable. Au-delà de cette durée ou de la durée d'expiration du message (voir la partie Enveloppe EDXL-DE), les messages seront purgés.

Cette gestion de l'expiration s'appuiera directement sur le Time-To-Live de RabbitMQ[26]. Un Time-To-Live général est appliqué par le Hub. Il est actuellement de 24 heures. Lors de la transmission d'un message de la file de l'émetteur à la file du destinataire, le Dispatcher du Hub Santé renseignera un *per-message TTL*[27] basé sur le *dateTimeExpires*[28] de l'Enveloppe EDXL-DE et permettant au message d'être purgé automatiquement vers RabbitMQ. Cette opération permet à l'émetteur de définir une durée de validité au niveau du message, qui puisse être fonction du cas d'usage. Si le champ dateTimeExpires de l'Enveloppe EDXL-DE est postérieur au Time-To-Live général du Hub, ce dernier s'applique. Que le message expire avant son arrivée au Dispatcher ou dans la file du destinataire (et soit récupéré dans un *Dead-Letter Exchange*[29] au niveau du Hub Santé), le Hub Santé enverra un message à l'expéditeur pour l'informer de l'expiration du message avant sa délivrance.

Pour le moment, aucun stockage supplémentaire des messages (approche d'Event Hub) en dehors des besoins de fonctionnement de RabbitMQ n'est envisagé en production. Les fonctionnalités d'épidémiologie ou de reprise d'activité avec un *backfill* des messages du Hub Santé ne sont de toute façon pas pleinement envisageables tant que seulement une partie réduite des dossiers transite par le Hub Santé. Néanmoins, afin de faciliter les travaux d'implémentation, de correction et d'amélioration continue du Hub Santé, tout ou partie des messages de tests pourra être stocké.

2.2.5. Ouvertures d'accès

En fonction de la situation dans les SAMU et du serveur du Client, il peut être nécessaire de demander l'ouverture des domaines utilisés aux équipes informatiques de l'établissement de santé. Ainsi, il faut *a minima* l'accès à l'URL utilisée par RabbitMQ. Il peut éventuellement être intéressant d'ajouter les URL des sites de présentation et démonstration en accès sur les postes des utilisateurs pour leur faciliter l'accès à l'information et la formation.

[26] https://www.rabbitmq.com/ttl.html
[27] https://www.rabbitmq.com/ttl.html#per-message-ttl-in-publishers
[28] L'utilisation directe d'un TTL par le Client sur ses messages n'est pas recommandé car la logique du Hub s'appuie sur le champs dateTimeExpires pour permettre au Client de se concentrer seulement sur le message et d'avoir des messages autoporteurs.
[29] https://www.rabbitmq.com/dlx.html

3. INTERFAÇAGE

3.1. Connexion

La connexion utilise le protocole AMQP 0-9-1. Chaque client se connecte au Hub Santé via un accès sécurisé par certificat remplaçant ainsi l'usage d'un couple identifiant / mot de passe. Le chiffrement du flux, l'autorisation et l'authentification des clients sont ainsi réalisés via des certificats X509 signés par l'AC (Hub Santé au début, IGC Santé à terme), voir la partie Sécurisation des échanges pour plus de détails.

L'URL d'accès au RabbitMQ du Hub Santé est messaging.hub.esante.gouv.fr. Afin de gérer différentes versions (notamment les itérations successives des spécifications techniques), le Hub Santé pourra utiliser le concept technique de « Virtual Hosts ». Ainsi, et conformément aux spécifications d'AMQP 0-9-1[30], l'**URI complète** (sans identifiant et mot de passe car l'authentification est portée par les certificats) est donc de la forme

$$"amqps://messaging.hub.esante.gouv.fr:5671" ["/" vhost]$$

Afin d'**activer les échanges en TLS**, le Client doit configurer son connecteur. La documentation de RabbitMQ a une page très complète sur le TLS[31] et donne des pistes pour Java[32] et .NET[33]. D'autre part, le GitHub offre deux exemples d'implémentation : en Java dans les Consumer/Producer[34] et en Spring Boot pour le Dispatcher du Hub Santé[35]. Le protocole TLS doit être utilisé dans la version 1.2 et les Clients doivent se connecter en « Peer Verification » (avec un truststore permettant de valider le certificat du serveur) ainsi qu'en « Server Hostname Verification » (en validant la correspondance entre le domaine du serveur et le certificat présenté).

Afin d'**utiliser les certificats pour l'authentification** et non un couple identifiant et mot de passe, le Client doit spécifier le mécanisme d'authentification utilisé dans son connecteur pour y spécifier la valeur « EXTERNAL ». La documentation de RabbitMQ propose quelques pistes pour Java, .NET et Erlang[36]. D'autre part, le GitHub offre deux exemples d'implémentation : en Java dans les Consumer/Producer[37] et en Spring Boot pour le Dispatcher du Hub Santé[38].

Le Client est autorisé, suite à une authentification basé sur le Common Name (CN) de son certificat :

- À lire les messages publiés uniquement sur ses trois files d'écoute ({identifiantClient}.message, {identifiantClient}.ack et {identifiantClient}.info)
- À publier un message sur l'échangeur d'envoi uniquement en fournissant son identifiant client en tant que clé de routage

3.2. Entrée et sortie des messages

Afin de garantir la reprise d'activité en cas de problème sur le Hub Santé, les files de messages sont déclarées comme persistantes. Ainsi, leurs métadonnées sont stockées directement sur le disque et non seulement en mémoire vive ce qui permet de les récupérer lorsque le système est relancé.

[30] https://www.rabbitmq.com/uri-spec.html
[31] https://www.rabbitmq.com/ssl.html
[32] https://www.rabbitmq.com/ssl.html#java-client
[33] https://www.rabbitmq.com/ssl.html#dotnet-client
[34] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.6/client/src/main/java/com/hubsante/Consumer.java#L65 et https://github.com/ansforge/SAMU-Hub-Sante/blob/0.6/client/src/main/java/com/hubsante/TLSConf.java#L9-L45
[35] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.6/hub/dispatcher/src/main/resources/application.properties#L3-L9
[36] https://www.rabbitmq.com/access-control.html#client-mechanism-configuration
[37] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.6/client/src/main/java/com/hubsante/Consumer.java#L60
[38] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.6/hub/dispatcher/src/main/java/com/hubsante/hub/config/AmqpConfiguration.java#L39

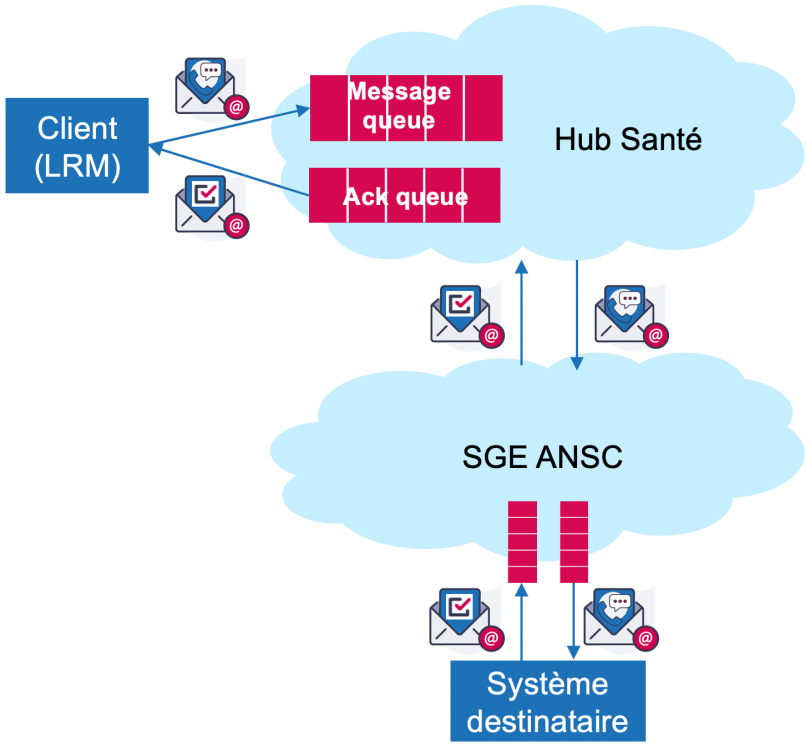
Cependant, pour pouvoir récupérer les messages présents sur les files, tous les messages en entrée des HubEx doivent être également insérés avec un mode persistant faute de quoi ils ne seront pas récupérés lorsque le système est relancé[39].

Le GitHub offre un exemple d'implémentation en Java dans le Producer[40]. En Spring Boot, les messages sont envoyés en mode persistant par défaut[41].

3.3. Flot des messages et acquittements

Le schéma suivant illustre le cheminement d'un message dans l'architecture des HubEx. Dans cet exemple, le Client LRM génère un message contenant les données d'un appel à destination d'un Client du Hub Pompiers (le Système de Gestion des Échanges – SGE – de l'ANSC). Comme expliqué dans la partie Messages, ce message au format CISU est inscrit dans une enveloppe EDXL-DE. Le client publie ensuite son message sur l'échangeur $hubsante$ du Hub Santé en fournissant comme clé de routage son Identifiant.

Le Hub Santé récupère le message et le transmet au SGE de l'ANSC qui l'inscrit sur la queue $\{IdentifiantDestinataire\}.message$ qui est consommée par le client destinataire.



Afin de garantir la fiabilité du système, des acquittements sont nécessaires pour

- Informer chaque acteur lorsque le message a bien été traité par l'intermédiaire suivant
- Informer l'émetteur que le message a bien été reçu par le destinataire

Pour informer l'émetteur du traitement ou des actions prises par le destinataire, les messages fonctionnels doivent être utilisés

3.3.1. Informer du transfert de responsabilité – acquittement technique

[39] https://www.rabbitmq.com/queues.html#durability
[40] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.7/client/src/main/java/com/hubsante/Producer.java#L79
[41] https://github.com/spring-projects/spring-amqp/blob/v3.0.4/spring-amqp/src/main/java/org/springframework/amqp/core/MessageProperties.java#L65

Un point clé de l'approche est que les HubEx vont conserver un message dans leurs files et tenter de le renvoyer à l'intermédiaire suivant jusqu'à réception d'une confirmation de prise en charge par l'intermédiaire suivant. En effet, une fois qu'un consommateur a bien pu traiter un message (*i.e.* pour un destinateur, le récupérer et l'enregistrer dans son système, ou, pour un HubEx, l'inscrire dans une de ses files d'envoi ce qui en garantira la persistance et les tentatives d'envoi multiples en cas de soucis), la responsabilité du message lui incombe désormais et il peut informer l'intermédiaire précédent que le message est bien passé pour que celui-ci le supprime de sa file d'envoi.

Cette fonctionnalité est directement intégrée au protocole AMQ 0-9-1 sous la forme de *Consumer Acknowledgement*[42]. Ces acquittements doivent être intégrés à la fin du code de traitement d'un message. En cas d'oubli, les messages resteraient dans la file d'envoi et RabbitMQ tenterait de les renvoyer au client destinataire ce qui conduirait à différents problèmes (duplication de messages, blocage des files, problèmes de mémoire, …).

Le GitHub offre un exemple d'implémentation en Java dans le Consumer[43]. En Spring Boot, les messages sont acquittés automatiquement à la suite de la bonne exécution du Listener[44].

3.3.2. Informer de la réception – acquittement de réception finale

La validation auprès de l'émetteur, appelée *Publisher Confirms* par RabbitMQ, n'est pas nativement intégrée dans le protocole AMQP 0-9-1. Afin d'offrir cette fonctionnalité, un deuxième type de files $\{IdentifiantClient\}.ack$ permet de remonter les accusés de réception finale.

Une fois le message intégré dans le système du Client destinataire, ce dernier peut en informer le Client émetteur en lui envoyant un acquittement de réception finale sur ces files de messages. Le cheminement est similaire au message envoyé mais pris en sens inverse.

D'un point de vue fonctionnel, tant que l'acquittement de réception finale n'est pas reçu côté émetteur, le Client émetteur doit considérer que le message n'a pas été délivré au Client destinataire et doit en informer ses utilisateurs pour que ceux-ci puissent éventuellement contacter le destinataire.

3.3.3. Informer du traitement – message fonctionnel

L'accusé de réception finale n'informe que de la réception du message par le Client destinataire et ne donne aucune information sur le traitement et les actions réalisés par le destinataire. Pour garder des messages d'acquittements les plus simples possibles et devant la pluralité des réponses possibles, ces informations doivent faire l'objet de nouveaux messages fonctionnels et ne pas passer par les acquittements.

3.3.4. Gestion des erreurs

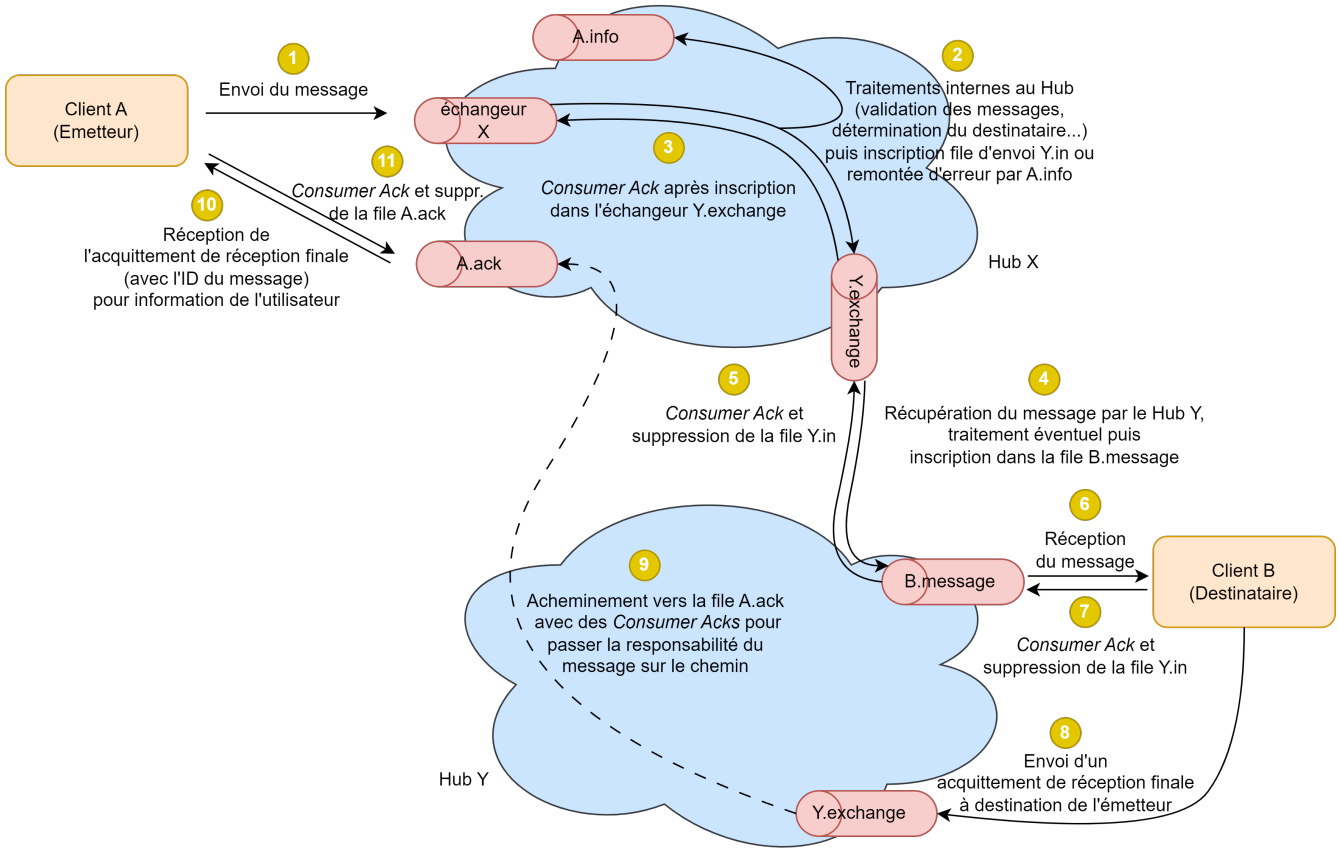
Un troisième type de file, $\{IdentifiantClient\}.info$, est mis en place pour remonter des informations et de potentielles erreurs aux émetteurs et destinataires des messages. Une erreur survenue au cours d'un envoi de message peut aboutir à l'envoi d''un message d'erreur dans les cas suivants :

- Définition non conforme du Content-Type dans les propriétés du message AMQP (voir la section Format des messages : JSON et XML)
- Incohérence entre la clé de routage utilisée à la publication et le champ $SenderID$ dans l'Enveloppe EDXL-DE (voir la section Enveloppe EDXL-DE)
- Non-conformité du message envoyé aux modèles de données spécifiées dans les différents DSF
- Expiration du message avant dépilement par le destinataire
- Rejet du message par le SI destinataire lors de son dépilement

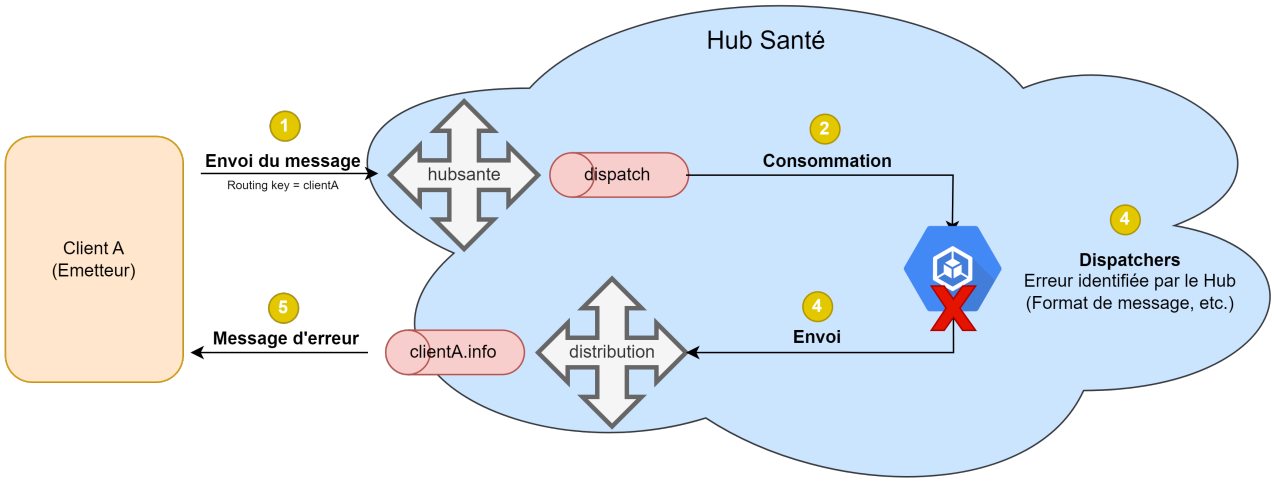
[42] https://www.rabbitmq.com/confirms.html#consumer-acknowledgements
[43] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.7/client/src/main/java/com/hubsante/ConsumerRun.java#L48
[44] https://docs.spring.io/spring-amqp/reference/html/#acknowledgeMode

3.3.5. Schémas récapitulatifs

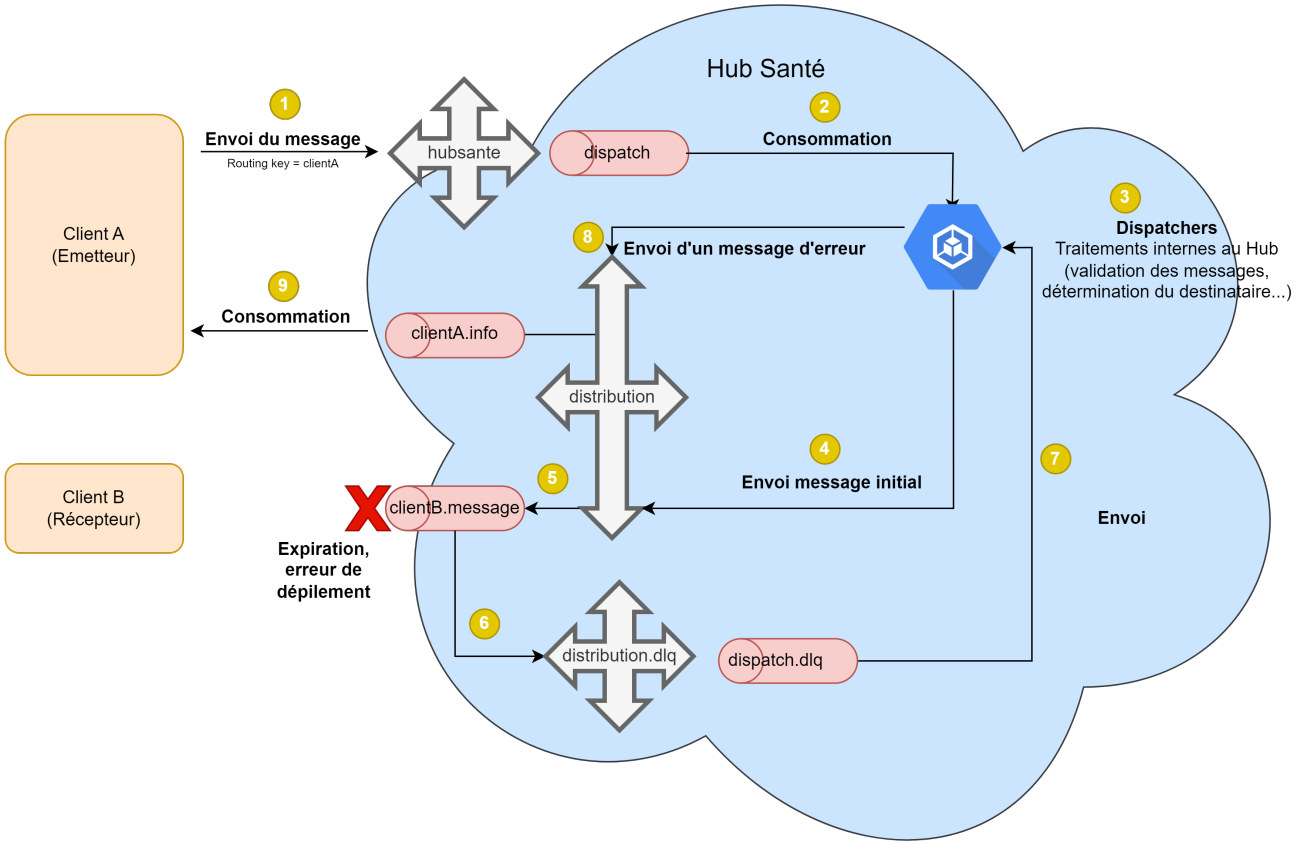
Le schéma suivant illustre le cheminement d'un message entre deux Clients liés à des HubEx différents ainsi que les acquittements techniques (*Consumer Acks*) et l'acquittement de réception finale (message *ack*).



Le schéma suivant illustre le cheminement d'un message rejeté par le Hub Santé.



Le schéma suivant illustre le cheminement d'un message accepté par le Hub mais non dépilé (expiration du délai, erreur interne côté destinataire...)



3.4. Messages

Un message transitant par le réseau de HubEx est composé :

- D'une enveloppe de type EDXL-DE contenant les informations utiles au routage du message. Il s'agit de la seule partie exploitée par les HubEx.
- D'un corps de message contenant les informations fonctionnelles à transmettre au Client destinataire.

Pour l'ensemble des cas d'usages portés par le Hub Santé, un contrat d'interface (prenant la forme d'un Dossier des Spécifications Fonctionnelles) sera partagé et accompagné d'un modèle de données décrivant le format des messages autorisés (AsyncAPI / JSON Schema en JSON, XSD en XML). Bien que, pour faciliter les premières implémentations, le Hub Santé n'a pas activé la validation des messages pour le moment, la cible est bien une validation de la conformité des messages passant par le Hub Santé aux modèles de données validés.

La base commune à tous ces cas d'usage est les règles de nommage des acteurs et les règles de routage associées ainsi que l'enveloppe EDXL-DE permettant de porter ces informations d'adressage dans les messages.

3.4.1. Format des messages : JSON et XML

Par défaut, les messages échangés doivent être en JSON. Pour certains cas d'usage, notamment le lien 15-NexSIS basé sur les travaux CISU (Cadre d'Interopérabilité des Services d'Urgence), les messages pourront également être échangés en XML. La conversion entre les deux formats et la correspondance entre les contrats d'interface et

modèles de données seront assurés par le Hub Santé. Ainsi, les Clients peuvent ne s'occuper que du contrat d'interface et modèle de données dans le format de leur convenance.

Pour permettre au Hub de prendre en charge ces deux formats, il est nécessaire de définir le $Content-Type$ au niveau des propriétés du message AMQP. Cela se fait généralement au niveau du client RabbitMQ. Deux valeurs sont acceptables par le Hub Santé : $application/json$ et $application/xml$. Toute autre valeur, ou si le $Content-Type$ n'est pas explicitement défini, entraînera un échec de transmission et l'envoi d'un message d'erreur correspondant sur la file info de l'émetteur.

3.4.2. Identifiants et logique de routage

La logique de nommage est encore en travaux au niveau de CISU et est amenée à évoluer. Néanmoins, l'approche retenue actuellement est basée sur une logique de domaines imbriqués et permet d'échanger à plusieurs échelles.

Ainsi, l'identifiant d'un SAMU aurait la forme $fr.health.samuXXX$ permettant d'indiquer qu'il s'agit :

- D'un acteur français
- Du domaine de la Santé
- Correspondant au SAMU du département XXX

XXX étant le code CRRA (ex : FR680 pour Mulhouse, FR2A0 pour Ajaccio ; FR641 pour Bayonne).

Ces identifiants permettent aux différents HubEx de pouvoir router les messages en interne ou vers le Hub adapté.

3.4.3. Enveloppe EDXL-DE

Le tableau ci-dessous précise les balises qui doivent être envoyées et qui sont nécessaires au routage des messages. Ces balises sont spécifiées dans le document EDXL-DE (Emergency Data Exchange Language - Distribution Element), version 2.0 du 19 septembre 2013[45].

Bien que le format par défaut du Hub Santé soit le JSON, pour certains cas d'usage (notamment le lien 15-NexSIS), cette enveloppe peut également être envoyée sous une forme XML. Les mêmes règles de format et de cardinalité s'appliquent. Le nommage des champs est identique dans les deux cas, à l'exception de l'élément encapsulant le message fonctionnel en lui-même : pour plus de clarté, dans le cas d'une enveloppe en JSON, il a été décidé de renommer les champs **contentXML** en **JsonContent** et **embeddedXMLContent** en **embeddedJsonContent**.

[45] https://docs.oasis-open.org/emergency/edxl-de/v2.0/edxl-de-v2.0.html

edxlDistribution

Paramètre	Description	Cardinalité	Type
distributionID	Identifiant unique de ce message d'urgence attribué par l'expéditeur et comme étant unique pour cet expéditeur.	[1..1]	**Chaîne de caractères** devant respecter le format suivant : **<senderId>_<internalId>** senderId : voir paramètre senderID internalId : identifiant libre d'attribution par l'expéditeur mais étant garanti unique dans son système Ex : $fr.health.samuXXX\_ID12345$
senderID	Identifiant de l'émetteur parmi une liste donnée.	[1..1]	**Chaîne de caractères** parmi la liste des acteurs raccordés au réseau de HubEx et suivant les règles de nommage. Ex : $fr.health.samuXXX$
dateTimeSent	Date et heure auxquelles le message a été envoyé.	[1..1]	**Horodatage avec décalage horaire par rapport au fuseau GMT**, Ex : 2022-06-21T23:59:00+02:00 (21 juin 2022 à 23h59, heure de Paris)
dateTimeExpires	Date et heure auxquelles le message doit expirer : c'est-à-dire que les données doivent être détruites et non délivrées au-delà de cette date.	[1..1]	**Horodatage avec décalage horaire** (cf supra)
distributionStatus	Statut du message à choisir parmi plusieurs valeurs. *Actuellement sans effet dans le routage du Hub Santé mais utilisé à terme pour faire passer des messages de tests en recette ou investigation.*	[1..1]	Valeurs énumérées parmi la liste ci-dessous : **Actual, Exercise** Les valeurs ci-dessous spécifiées par la spécification EDXL ne sont actuellement pas utilisées : System, Test, Unknown, NoAppropriateDefault
distributionKind	Permet de préciser le type de message. *Le routage s'appuie sur ce champ : le Hub traite les valeurs Report, Update, Cancel pour transférer le message sur la file message du destinataire, la valeur Ack vers la file ack et la valeur Error vers la file info.*	[1..1]	Valeurs énumérées parmi la liste ci-dessous : **Report, Update, Cancel, Ack, Error** Les valeurs ci-dessous spécifiées par la spécification EDXL ne sont actuellement pas utilisées : Request, Response, Dispatch, SensorConfiguration, SensorControl, SensorStatus, SensorDetection, Unknown, NoAppropriateDefault
descriptor	Description des sous-éléments liés au message.	[1..1]	Voir tableau **descriptor** ci-après.
content	Contenu du message.	[1..1]	Voir tableau **content** ci-après.

descriptor

Paramètre	Description	Cardinalité	Type
combinedConfidentiality	Non utilisé	Non utilisé	Non utilisé
language	Langage du message échangé.	[1..1]	**Étiquette** de langue complète de l'IETF[46]. Ex : fr-FR
senderRole	Non utilisé	Non utilisé	Non utilisé
recipientRole	Non utilisé	Non utilisé	Non utilisé
keyword	Non utilisé	Non utilisé	Non utilisé
explicitAddress	Identifiants de l'expéditeur et du destinataire du message.	[1..1]	Voir tableau **explicitAddress** ci-après.
targetAreas	Non utilisé	Non utilisé	Non utilisé
urgency	Non utilisé	Non utilisé	Non utilisé
severity	Non utilisé	Non utilisé	Non utilisé
certainty	Non utilisé	Non utilisé	Non utilisé
incidentID	Non utilisé	Non utilisé	Non utilisé
incidentDescription	Non utilisé	Non utilisé	Non utilisé
link	Non utilisé	Non utilisé	Non utilisé
extension	Non utilisé	Non utilisé	Non utilisé

content

Paramètre	Description	Cardinalité	Type
contentObject	Contenu du message.	[1..*]	Voir tableau **contentObject** ci-après.
link	Non utilisé	Non utilisé	Non utilisé
other	Non utilisé	Non utilisé	Non utilisé

[46] https://fr.wikipedia.org/wiki/%C3%89tiquette_d%27identification_de_langues_IETF

contentObject			
Paramètre	**Description**	**Cardinalité**	**Type**
contentDescriptor	Non utilisé	Non utilisé	Non utilisé
contentXML	Contient les données dans une structure XML donnée, dans un unique élément enfant **embeddedXMLContent**.	[1..1]	Contenu du message au format **XML**.
JsonContent ***(alternative en JSON)***	Contient les données dans un schéma JSON donné, dans un unique élément enfant **embeddedJsonContent**.	[1..1]	Contenu du message en **JSON**.
otherContent	Non utilisé	Non utilisé	Non utilisé

3.4.4. Adressage destinataire

La norme EDXL-DE permet de spécifier plusieurs balises <*explicitAddress*>. Cependant, il a été retenu de ne permettre l'émission que vers un unique destinataire et de n'avoir que des messages point-à-point. Le message complet doit donc être dupliqué s'il est nécessaire de l'envoyer à plusieurs destinataires.

Le tableau ci-dessous précise les balises qui doivent être envoyées pour l'utilisation de l'élément <*explicitAddress*> :

explicitAddress			
Paramètre	**Description**	**Cardinalité**	**Type**
explicitAddressScheme	Identifiant du SI pilotant le Hub	[1..1]	Pour tout message envoyé au Hub Santé, la **valeur est fixe : "hubsante".**
explicitAddressValue	Identifiant du destinataire du message parmi une liste donnée	[1..1]	**Chaîne de caractères** parmi la liste des acteurs raccordés au réseau de HubEx et suivant les règles de nommage. Ex : $fr.health.samuXXX$ *Nota : il s'agit de la même liste que pour la balise senderID.*

3.4.5. Schémas et exemples de messages

Les schémas (AsyncAPI et XSD) ainsi que des exemples de messages (JSON et XML) sont fournis dans le dossier *docs/DST* du GitHub partagé (voir la section *Repository* GitHub partagé pour plus de détails).

4. ETAPES D'IMPLEMENTATION

Afin de faciliter l'implémentation des présentes spécifications, voici une liste indicative des différentes étapes à réaliser :

- Se familiariser avec le client RabbitMQ dans le langage utilisé : https://www.rabbitmq.com/devtools.html
- Demander les accès et se familiariser avec la structure du GitHub (notamment le dossier *docs*) : https://github.com/ansforge/SAMU-Hub-Sante
- Faire des tests d'implémentation en local ou en utilisant le certificat client[47] présent sur GitHub (utilisateur partagé avec tous les autres Clients implémentant les spécifications)
- Afin de faciliter et séquencer les développements, il est possible de suivre de façon incrémentale les différentes versions du Hub Santé[48]
- Pour les Clients utilisant Java, il est possible d'intégrer facilement le code des Consumer et Producer[49] dans votre code en adaptant les ConsumerRun et ProducerRun pour correspondre au besoin de votre code. Pour les Clients utilisant un autre langage que Java, il est possible de co-construire avec l'équipe du Hub Santé les équivalents dans votre langage.
- S'enregistrer auprès du Hub Santé en transmettant une CSR pour obtenir un certificat signé par l'AC du Hub avec votre identifiant en CN ainsi que la création de files dédiées

5. EVOLUTIONS TECHNIQUES

Au-delà du périmètre initial du pilote, plusieurs fonctionnalités techniques sont considérées :

- Monitoring et *heart beat* sur les services opérationnels. RabbitMQ implémente déjà un certain niveau de vérification sur ses connections (voir https://www.rabbitmq.com/heartbeats.html) et nous avons prévu de travailler sur une partie plus avancée du monitoring du Hub Santé. Ajouter une vérification des liens et une notification aux autres acteurs lorsque certains liens ne sont plus fonctionnels sont notamment des fonctionnalités intéressantes qui seront considérées dans ces travaux de monitoring.
- Établissement d'une matrice de distribution selon les versions des référentiels et standards implémentés par chacun des clients
- Transposition des messages entre les versions supportées par les différents destinataires

[47] https://github.com/ansforge/SAMU-Hub-Sante/tree/main/certs
[48] https://github.com/ansforge/SAMU-Hub-Sante/tree/main/docs
[49] https://github.com/ansforge/SAMU-Hub-Sante/tree/main/client/src/main/java/com/hubsante

6. DEVELOPPEMENTS A REALISER

Les développements du Hub Santé (demi-connecteur et logique d'exploitation) sont réalisés côté ANS en Java. Le code est accessible sur le GitHub partagé (voir la partie *Repository* GitHub partagé pour plus de détails).

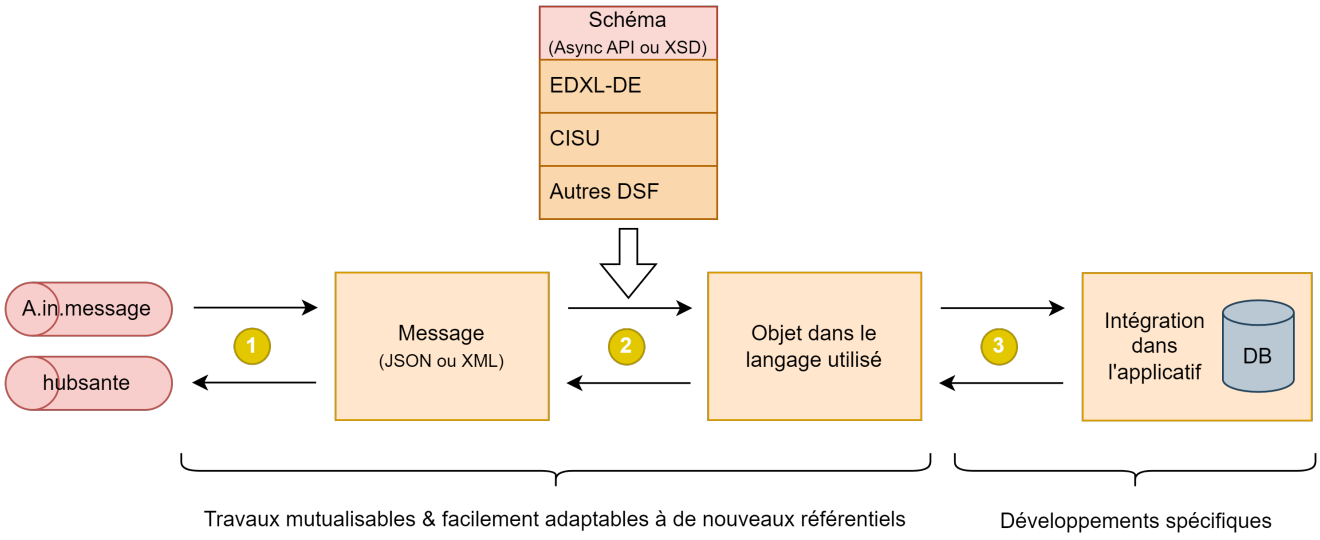
AMQP 0-9-1 est un protocole très répandu et dispose notamment de clients open-source dans un nombre important de langages de programmation. Afin de vous familiariser avec les clients liés à votre code, RabbitMQ propose des tutoriels[50] sur les langages suivants : Python, Java, Ruby, PHP, C#, JavaScript, Go, Elixir, Objective-C, Swift et Spring AMQP. Les tutoriels Java ont été intégrés au GitHub partagé[51] pour pouvoir être lancé avec quelques instructions en ligne de commande.

Les travaux côté éditeur consisteront à

1. S'appuyer sur un client AMQP correspondant au langage utilisé pour lire et écrire dans les files ouvertes par le Hub Santé
2. S'appuyer sur les schémas (AsyncAPI ou XSD) définissant les échanges pour valider les messages et faire le transcodage entre les messages (JSON ou XML) et les objets utilisés dans le code
3. Faire le lien entre les objets (issus de ou convertis en messages) et l'applicatif (base de données, logique, …).

Pour les langages les plus répandus, les développements sur les deux premiers éléments peuvent être co-construits entre les éditeurs concernés et l'ANS. Le dernier point est spécifique à chaque éditeur et devra être implémenté de façon autonome.

Le schéma ci-dessous illustre le cheminement d'un message côté éditeur et les différentes étapes.



[50] https://www.rabbitmq.com/getstarted.html
[51] https://github.com/ansforge/SAMU-Hub-Sante/tree/main/src/main/java/com/tutorials

7. ANNEXES

7.1. Génération d'un certificat

Pour qu'un client d'un HubEx dispose d'un certificat pour s'authentifier, il convient préalablement qu'il génère une requête de certification. Cette requête de certification se compose de deux parties :

- Une partie privée : la clé privée
- Une partie publique : la CSR (Certificate Signing Request)

La partie privée doit être conservée par le client.
La partie publique peut être auto-signée par le client ou peut être transmise à l'équipe du Hub Santé pour signature avec un certificat faisant office de CA pour le pilote.

En réponse à l'envoi de la CSR, l'équipe en charge de l'autorité de certification devra :

- Signer cette CSR pour en faire un certificat,
- Transmettre le certificat généré au client.

Le script *certs/generate.sh*[52] du GitHub permet de générer une clé privée, une CSR à partager ainsi qu'un certificat auto-signé.

[52] https://github.com/ansforge/SAMU-Hub-Sante/blob/0.7/certs/generate.sh