

Titre: Développement d'une plateforme de communication sol-air pour les systèmes avioniques du futur
Title:

Auteur: Hadrien Farineau
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Farineau, H. (2023). Développement d'une plateforme de communication sol-air pour les systèmes avioniques du futur [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. <https://publications.polymtl.ca/54138/>
Citation:

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/54138/>
PolyPublie URL:

Directeurs de recherche: Guchuan Zhu, & Philippe Doyon-Poulin
Advisors:

Programme: Génie électrique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Développement d'une plateforme de communication sol-air pour les systèmes
avioniques du futur**

HADRIEN FARINEAU

Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Juin 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Développement d'une plateforme de communication sol-air pour les systèmes
avioniques du futur**

présenté par **Hadrien FARINEAU**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Jérôme LE NY, président

Guchuan ZHU, membre et directeur de recherche

Philippe DOYON-POULIN, membre et codirecteur de recherche

Giovanni BELTRAME, membre

DÉDICACE

À ma famille pour leur soutien inconditionnel...

REMERCIEMENTS

Tout d’abord, je tiens à remercier mes directeurs de recherche Philippe Doyon-Poulin et Guchuan Zhu pour leurs nombreux conseils ainsi que pour la chance qu’ils m’ont offerte en participant au projet ADAIR. Leur savoir et leur partage de connaissances m’ont été essentiels afin de mener à bien ce projet.

Je tiens également à remercier les membres de mon jury, Jérôme Le Ny et Giovanni Beltrame, pour avoir accepté de lire ce mémoire.

Un remerciement spécial à nos partenaires industriels d’Adacel, de Bombardier, de CMC Electronics, de Marinvent, de Nav Canada, de Presagis et de Thales avec qui j’ai pu collaborer et qui ont pris le temps de répondre à toutes mes questions, et de partager leur grande expertise en aviation.

Merci à ma famille qui m’a soutenu pendant toute la durée de mes études et m’a toujours poussé à atteindre mes objectifs. Merci à mes colocataires pour cette motivation au quotidien.

Finalement, je tiens à remercier mes camarades du laboratoire de recherche pour tous les bons moments partagés, merci à Gabriel, Enzo, Joliann, Clara, Thomas et Ilrick !

RÉSUMÉ

Face à l'augmentation de la densité du trafic aérien dans les années à venir, la gestion de l'information à l'échelle du système (SWIM) apparaît comme une technologie émergente qui sera largement exploitée dans l'industrie aéronautique à l'avenir. Ce système permettra de répondre à des événements imprévus pendant les vols. L'infrastructure de communication entre le sol et l'avion pour assurer l'échange des données n'est cependant pas encore déployée. Parallèlement, la nécessité d'améliorer l'intégration de ces nouvelles données dans les cockpits a été soulevée. Face à la grande quantité de données rendues disponibles, leur présentation sur les affichages de bord est cruciale.

Ce travail de recherche a permis de développer SWIMAIR, une plateforme d'échange et de filtrage de messages SWIM, pouvant être liée avec des logiciels de création d'interfaces. Ceci est permis par l'utilisation du portail SWIFT mis à disposition par l'Administration fédérale de l'aviation (FAA) et offrant l'accès aux données issues du SWIM. Les principales fonctions de la plateforme permettent l'acquisition des messages auprès du portail, leur analyse basée sur la comparaison des noeuds XML avec les filtres appliqués, et leur envoi au travers de sockets TCP/IP. Une communication éditeur-abonné est ainsi instaurée afin de garantir à l'avion la réception des messages le concernant.

La caractérisation de la plateforme développée a été réalisée par le biais d'un test de la quantité de messages analysés et reçus par l'avion dans des aéroports de différentes tailles. Elle nous permet ainsi de traiter plus de 30 000 messages par minute et d'en extraire des messages d'intérêt en fonction des filtres appliqués. Elle permettra par exemple d'alimenter les interfaces avec plus de 1 000 messages par minute lors de son utilisation dans un contexte optimal, s'intéressant à des vols entre des aéroports majeurs lors des heures de pointe. L'étude des performances montre le faible impact temporel de la plateforme sur l'acheminement des messages. Ce travail constitue ainsi le point de départ d'une série d'études en facteurs humains sur la création et l'intégration de nouvelles interfaces se basant sur les données issues du SWIM.

ABSTRACT

As the density of air traffic increases in the coming years, System Wide Information Management (SWIM) is recognized to be an emerging technology that will be widely exploited in aviation industry in a near future. This system will enable responses to unexpected events during flights. However, the ground-to-aircraft communications infrastructure for data exchange has yet to be deployed. Meanwhile, the need for improvements in terms of integration of these new data in the cockpits has been raised. Faced with the large amount of data made available, their presentation on the on-board displays is crucial.

In this research work we developed a platform, called SWIMAIR, for SWIM message exchange and filtering, which can be linked with interface creation software. This is made possible by the use of the SWIFT portal provided by the Federal Aviation Administration (FAA) and offering access to SWIM data. The main functions of the platform allow the acquisition of messages from the portal, their analysis based on the comparison of the XML nodes with the applied filters, and their sending via a data link based on TCP/IP sockets. A publisher-subscriber air-ground network connectivity is thus established in order to guarantee the reception of messages concerning the aircraft.

Characterization of the developed platform has been carried out via the test of the quantity of messages analyzed and received by the aircraft in airports of different sizes. More than 30,000 messages per minute are processed by the platform and messages of interest are extracted according to the filters applied. For example, it will allow us to feed the interfaces with more than 1,000 messages per minute when used in an optimal context, focusing on flights between major airports during peak hours. The performance study shows that the platform has little impact on message routing over time. This work is the starting point for a series of human factors studies on the creation and integration of new interfaces based on SWIM data.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vi
TABLE DES MATIÈRES	vii
LISTE DES TABLEAUX	ix
LISTE DES FIGURES	x
LISTE DES SIGLES ET ABRÉVIATIONS	xii
LISTE DES ANNEXES	xv
CHAPITRE 1 INTRODUCTION	1
CHAPITRE 2 REVUE DE LITTÉRATURE	4
2.1 Gestion de l'information à grande échelle (SWIM)	4
2.1.1 Principe	4
2.1.2 Organisation de l'échange d'information au sein du SWIM	4
2.1.3 Mise en forme des données	6
2.1.4 Accès de l'aéronef au SWIM	8
2.1.5 Portail SWIFT	11
2.2 État de l'art des solutions existantes	11
2.2.1 Patrons de communication	11
2.2.2 Utilisation d'un middleware	13
CHAPITRE 3 DÉFINITION DES OBJECTIFS DE RECHERCHE	18
3.1 Objectifs	18
3.2 Méthodologie	19
CHAPITRE 4 DÉVELOPPEMENT DE LA PLATEFORME SWIMAIR	21
4.1 Architecture de la plateforme	21

4.1.1	Outils utilisés	22
4.1.2	Bloc Éditeur	25
4.1.3	Bloc Abonné	37
4.2	Utilisation des données au travers de cas d'usage	39
4.2.1	Évolutions météorologiques	39
4.2.2	Surveillance du trafic autour de l'appareil	42
CHAPITRE 5 CARACTÉRISATION ET DIMENSIONNEMENT DE LA PLATE-		
FORME SWIMAIR		45
5.1	Description des tests	45
5.1.1	Intérêts de la réalisation des tests	45
5.1.2	Variables indépendantes	45
5.1.3	Variables dépendantes	46
5.1.4	Méthode d'acquisition des données	47
5.2	Résultats	48
5.2.1	Influence du moment de la journée sur la quantité de messages fournis par le portail SWIFT	48
5.2.2	Influence de la taille de l'aéroport sur la quantité de messages reçus par l'abonné	49
5.2.3	Répartition des éditeurs SWIM	51
5.2.4	Poids moyen des messages par minute	54
5.2.5	Étude de l'influence de la plateforme sur le retard	55
5.2.6	Étude de l'empilement des files	57
5.2.7	Mise en conditions avec le test d'un vol	59
5.3	Discussion	59
CHAPITRE 6 CONCLUSION		64
6.1	Synthèse des travaux	64
6.2	Limitations de la solution proposée	64
6.3	Futurs travaux de recherche	66
RÉFÉRENCES		68
ANNEXES		73

LISTE DES TABLEAUX

Tableau 2.1	Exemples de qualités de service assurées par DDS	15
Tableau 4.1	Fonctions créées lors du développement de la plateforme	22
Tableau 4.2	Association des premiers noeuds aux éditeurs SWIFT correspondants	30
Tableau 5.1	Répartition des messages reçus par l'abonné au cours de l'acquisition de données, en fonction de l'aéroport étudié et du moment de la journée	54
Tableau 5.2	Tableau récapitulatif des résultats portant sur la mesure du temps d'exécution des différentes fonctions de la plateforme	58
Tableau 5.3	Tableau récapitulatif des résultats portant sur la mesure du poids de messages en lien avec le temps d'exécution des fonctions de la plateforme	58
Tableau 5.4	Tableau récapitulatif des résultats portant sur la mesure du temps total d'exécution	58
Tableau 5.5	Tableau récapitulatif des résultats portant sur la mesure du poids de messages en lien avec le temps total d'exécution	58
Tableau A.1	Types de messages du système de publication des données de vol en route (ERDFP)	79
Tableau A.2	Types de messages du système de publication de données sur l'espace aérien en route (ERADP)	80
Tableau A.3	Types de messages du système de publication de données opération- nelles en route (ERODP)	80
Tableau A.4	Types de messages du système de publication de messages généraux en route (ERGMP)	80

LISTE DES FIGURES

Figure 2.1	Passage d'une architecture à liens directs (gauche) une architecture orientée services (droite)	5
Figure 2.2	Organisation des différentes couches dans le système SWIM	5
Figure 2.3	Exemple de message XML	8
Figure 2.4	Éditeurs de données présents sur le portail SWIFT	11
Figure 4.1	Architecture de la solution développée	21
Figure 4.2	Organisation de la partie éditeur de la plateforme	26
Figure 4.3	Diagramme de flux de l'acquisition d'un message provenant du portail SWIFT	27
Figure 4.4	Exemple de lignes "type" à ignorer lors de l'acquisition de messages .	28
Figure 4.5	Exemple de message STDDS concernant le TDES	32
Figure 4.6	Extension du bloc éditeur aux cinq éditeurs SWIFT	37
Figure 5.1	Classement des aéroports choisis en fonction du nombre d'opérations sur l'année 2021	46
Figure 5.2	Marqueurs temporels ajoutés pour la réalisation des mesures de temps de traitement	48
Figure 5.3	Nombre moyen de messages analysés par minute par SWIMAIR selon la période de la journée, et écarts-types associés	50
Figure 5.4	Nombre moyen de messages reçus par minute par l'abonné après l'application d'un filtre "aéroport" par rapport à la taille de l'aéroport sélectionné, et écarts-types associés	50
Figure 5.5	Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux grands aéroports	52
Figure 5.6	Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux aéroports de taille moyenne	52
Figure 5.7	Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux petits aéroports	53
Figure 5.8	Répartition des messages reçus par l'abonné au cours de l'acquisition de données le matin dans les grands aéroports	53
Figure 5.9	Moyenne des poids des messages reçus par l'abonné en fonction de l'éditeur SWIM d'origine dans les aéroports de grande taille, et écarts-types associés	56

Figure 5.10	Mesure du débit de données reçues par l’abonné dans les grands aéroports étudiés en fonction du moment de la journée	56
Figure 5.11	Nombre d’occurrences de chaque taille lors de l’observation de l’évolution de la taille de “m_queue”	60
Figure 5.12	Mesure de l’évolution de la taille de la file de messages à envoyer au client “s_queue”	60
Figure 5.13	Répartition de l’origine des messages lors de l’acquisition des données du vol DAL482 entre JFK et LAX	61
Figure A.1	Produits fournis par ITWS	74
Figure A.2	Exemple de message de précipitations fourni par ITWS	76
Figure A.3	Exemple de configuration pour le TFMS pour une sortie StdoutOutput	81

LISTE DES SIGLES ET ABRÉVIATIONS

ACARS	Aircraft Communication Addressing and Reporting System
ADAIR	Advanced Airspace Usability (ADAIR)
ADS-B	Automatic Dependent Surveillance Broadcast
AAtS	Aircraft Access to SWIM
AeroMACS	Aeronautical Mobile Airport Communication System
AFP	Airspace Flow Program
AIM FNS	Aeronautical Information Management Federal NOTAM System
AIRMET	Airman's Meteorological Information
AIXM	Aeronautical Information Exchange Model
ANSP	Air Navigation Service Providers
APDS	Airport Data service
ASBU	Aviation System Block Upgrades
ASCII	American Standard Code for Information Interchange
ASDAR	Aircraft Data Acquisition and Relay System
ASDE	Airport Surface Detection Equipment
ASDI	Aircraft Situation Display to Industry
ASP	ATM Service Providers
ASR	Airport Surveillance Radar
ATC	Air Traffic Control
ATM	Air Traffic Management
CPDLC	Controller-Pilot Datalink Communication
CPU	Central Processing Unit
DDS	Data Distribution Service
EFB	Electronic Flight Bag
ERAM	En Route Automatisation Modernization
ERFDP	En Route Flight Data Publication
ERGMP	En Route General Message Publication
ERODP	En Route Operational Data Publication
FAA	Federal Aviation Administration
FACE	Futur Airborn Capability Environment
FCA	Flow Constrained Area
FF-ICE	Flight & Flow Information for a Collaborative Environment
FIFO	First In First Out

FIXM	Flight Information Exchange Model
FMS	Flight Management System
FTM	Flight Table Manager
GDP	Ground Delay Programs
IATA	International Air Transport Association
IFR	Instrument Flight Rule
IP	Internet Protocol
ISMC	Infrastructure System Monitoring and Control
ITWS	Integrated Terminal Weather Service
JMSDD	Java Messaging Service Description Document
LIFO	Last In First Out
METAR	Meteorological Aerodrome Report
MQTT	Message Queuing Telemetry Transport
NAVAID	Navigational Aids
NextGen	Next Generation Air Transportation System
NEXRAD	Next Generation Weather Radar
NDN	Name Data Network
NOTAM	Notice to Airmen
OACI	Organisation de l'Aviation Civile Internationale
OMG	Object Management Group
OMM	Organisation Météorologique Mondiale
OSI	Open Systems Interconnection
PBN	Performance Based Navigation
PIREP	Pilot Report
QoS	Quality of Service
RTA	Required Time of Arrival
RVR	Runway Visual Range
SATCOM	Satellite Communication
SESAR	Single European Sky ATM Research
SIGMET	Significant Meteorological Information
SMES	Surface Movement Event Service
SOA	Service Oriented Architecture
SFDPS	SWIM Flight Data Publication Service
STDDS	SWIM Terminal Data Distribution System
STARS	Standard Terminal Arrival
SWIFT	SWIM Industry-FAA Team

SWIM	System Wide Information Management
TAF	Terminal Area Forecast
TAIS	Terminal Automation Information Service
TBFM	Time Base Flow Management
TBO	Trajectory Based Operation
TCAS	Traffic Collision Avoidance System
TCP	Transmission Control Protocol
TDES	Tower Departure Event Service
TDWR	Terminal Doppler Weather Radar
TFMDG	Traffic Flow Management Data to Government
TFMS	Traffic Flow Management System
TIS-B	Traffic Information Service - Broadcast
TMI	Traffic Management Initiatives
TRACON	Terminal Radar Approach Control Facilities
UDP	User Datagram Protocol
UML	Unified Model Language
URI	Uniform Ressource Identifier
UTF	Unicode Transformation Format
VHF	Very High Frequency
WXXM	Weather Data Exchange Model
XML	Extensible Markup Language
Zenoh	Zero Overhead Network Protocol

LISTE DES ANNEXES

Annexe A	Descriptif des éditeurs disponibles au travers du portail SWIFT . . .	73
----------	---	----

CHAPITRE 1 INTRODUCTION

D’après les prédictions de l’Association Internationale du Transport Aérien (IATA) [1], le trafic aérien global doublera d’ici 2036 par rapport aux chiffres de 2017, posant de nouveaux enjeux en termes de gestion des flux aériens, de sécurité et de respect de l’environnement. Des travaux sont ainsi menés au sein des grandes institutions internationales telles que l’Organisation de l’Aviation Civile Internationale (OACI) afin d’anticiper cette augmentation et répondre à ces problèmes émergents. Ces travaux ont abouti en 2012 à la “Modernisation des blocs de systèmes d’aviation” (*Aviation System Block Upgrades* en anglais ou ASBU) [2] décrivant un ensemble de solutions et de pistes de recherches visant à moderniser les systèmes de gestion du trafic aérien. Parmi les principaux points de questionnements soulevés, on retrouve l’amélioration des opérations aéroportuaires (améliorations dans la prise de décision, séquençage des opérations de taxi, optimisation des procédures de départ et d’approche), la mise en avant de l’interopérabilité des systèmes et des données (échanges de données air-sol et sol-sol, systèmes de gestion des données météorologiques) [3,4], l’optimisation de la capacité de la gestion des vols (séparation des avions en vol, surveillance des trajectoires, prévision des collisions en vol) et de l’efficacité du plan de vol (amélioration des profils de descente et de montée, intégration de la notion de temps). Les opérations basées sur la trajectoire (*Trajectory Based Operation* - TBO) [2,5,6] apparaissent ainsi comme un concept majeur de ce travail de modernisation. Il vise à offrir une meilleure planification et un meilleur suivi des vols en y incorporant une dimension temporelle avec une trajectoire décrite en 4D (3 composantes spatiales et une composante temporelle). En effet, les trajectoires suivies par les avions seront alors prédéfinies bien avant le départ du vol et pourront évoluer en cours d’opération en fonction des conditions météorologiques, des zones aériennes restreintes, ou encore de l’optimisation de la consommation de carburant [5]. Cette technologie permettra de mieux gérer les flux aériens en permettant un séquençage plus précis des avions et une actualisation des plans de vol en cours d’opération. Cependant pour que le concept de TBO prenne tout son sens, il nécessitera le développement de moyens de communication sol-sol, sol-air et air-air plus efficaces permettant de relier entre eux tous les acteurs du secteur aérien. C’est dans cette optique, qu’un concept tel que le FF-ICE (pour Informations sur les vols et les flux pour un environnement collaboratif) [7] aidera à une meilleure gestion dynamique des trajectoires, à la réduction des délais et à une meilleure connaissance de l’espace aérien grâce au partage de l’information et des contraintes. Grâce au FF-ICE, la gestion du trafic aérien passera à un environnement plus collaboratif, où les échanges de données seront standardisés permettant un meilleur partage des trajectoires pour les optimiser pendant toute la durée du

vol. Un tel concept nécessite d'augmenter le partage des données entre les différents acteurs (gestion du trafic, avions, compagnies aériennes, aéroports...) [8] Le concept de SWIM (pour *System Wide Information Management*) apparaît alors comme un outil fondamental à la réussite de ce nouveau paradigme de gestion aérienne. Cet "intranet de l'aviation" basé sur une architecture orientée services (*Service Oriented Architecture* - SOA), mettant en avant la réutilisabilité des différents services fournis, permettra à tous les acteurs en jeu lors d'un vol d'avoir accès à l'information dont ils auraient besoin au moment auquel ils en auraient besoin et ce grâce à un langage commun basé sur le XML et des modèles de données établis. Malgré la volonté initiale d'aboutir à une infrastructure au sol du SWIM fonctionnelle dès 2019 et de permettre l'intégration des avions comme nœuds à part entière dans le système SWIM dès 2023, les derniers rapports de suivi de l'implémentation des ASBU [9] publié par EUROCONTROL en 2022 évoquent un retard sur ces plans. Les travaux actuels porteraient ainsi toujours sur la mise en place de l'infrastructure bien que certains travaux aient déjà débuté sur la connexion à l'avion. Dans un objectif de travail sur l'utilisation des données provenant du SWIM, l'Administration Fédérale de l'Aviation (FAA) a mis en place le portail SWIFT (pour SWIM Industry-FAA Team) qui permet aux industriels d'accéder directement aux données SWIM au travers de 6 éditeurs de données différents : ITWS pour les données météorologiques ; TFMS pour le partage des plans de vol ; AIM FNS pour les données aéronautiques ; STDDS pour les mesures par les équipements aéroportuaires ; TBFM pour les contraintes temporelles et SFDPS pour les données en vol.

Ainsi, face à l'état actuel du développement du SWIM ne permettant pas d'avoir accès à une infrastructure fonctionnelle pour l'échange de données relatives au SWIM entre les avions et les stations au sol, nous nous tournons vers le portail SWIFT afin d'accéder à ces données. En effet, de par le contexte industriel dans lequel s'inscrit notre projet, nous faisons face au besoin de travailler sur l'intégration des données SWIM sur de nouveaux affichages pour les cockpits. Nous nous fixons donc comme objectif de permettre l'alimentation d'interfaces du cockpit en données SWIM en proposant une solution permettant de réaliser le pont entre le portail SWIFT et les outils de création d'interfaces. Ce pont est vu comme un moyen de mettre en forme les messages et de les sélectionner sur la base d'un filtrage conditionnel afin que ces messages correspondent aux nécessités du vol suivi. Au travers du développement de notre plateforme, nous nous imposons également la nécessité d'assurer l'acheminement de la totalité des messages émanant du portail. Une attention particulière est portée sur l'influence de la plateforme sur le temps de traitement et la possibilité de création d'un retard temporel sur les messages. Il est indiqué dans le standard RTCA DO-257A¹ portant sur les

1. RTCA DO-257A Minimum Operational Performance Standards for the Depiction of Navigational Information on Electronic Maps

“Normes minimales de performance opérationnelle pour la représentation des informations de navigation sur les cartes électroniques” que le temps maximal permis afin de mettre à jour les informations sur les affichages électroniques de cartes est de 1 seconde. Nous utilisons alors cette norme comme objectif en assurant qu’un message SWIFT ne prenne pas plus de 1 seconde à atteindre les interfaces.

Le reste de ce mémoire est structuré comme suit. Le chapitre 2 présente une revue de littérature abordant le traitement des données au sein du SWIM et les travaux menés sur leur acheminement à l’avion. Nous y évoquerons également les différents outils pouvant être utilisées afin d’aboutir à une solution au problème d’acheminement de données en rapport avec le portail SWIFT. Le chapitre 3 présente les objectifs de la recherche ainsi que la méthode adoptée. Le chapitre 4 présente les détails du développement de notre solution en y expliquant les différentes fonctions développées. Le chapitre 5 présente les principaux résultats portant sur le dimensionnement et la caractérisation de la plateforme nous permettent de conclure sur ses performances et d’anticiper les conditions dans lesquelles il est préférable d’utiliser la plateforme pour en extraire le plus de données. Finalement, le chapitre 6 conclue ce mémoire en présentant une synthèse des travaux, les limitations existantes et les avenues à explorer pour de futures recherches. Un descriptif des différents types de messages et leurs contenus, proposés par le portail SWIFT est présenté en annexe.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Gestion de l'information à grande échelle (SWIM)

2.1.1 Principe

La mise en place de nouveaux concepts tels que le TBO inclut le développement de systèmes permettant le respect des contraintes temporelles fixées (systèmes RTA) tout au long du plan de vol ainsi que des contraintes de précision spatiales afin de rester sur la trajectoire prévue en tout temps. Cependant, afin de pouvoir arriver à un niveau de précision tant spatial que temporel, de nouveaux moyens de partage de l'information sont nécessaires afin de connecter le sol aux aéronefs et les aéronefs entre eux. C'est dans ce but que le concept de Gestion de l'information à grande échelle (*System Wide Information Management* - SWIM) [10,11] a été développé. La figure 2.1 illustre le passage d'une architecture à liens directs entre les aéronefs et les services, à une architecture orientée services dans laquelle les informations partagées le sont à tous et sont utilisables par les différents acteurs sans compromettre leur utilisation par d'autres. L'objectif du SWIM est de permettre à tous les acteurs en jeu lors d'un vol d'avoir accès à l'information dont ils auraient besoin au moment auquel ils en auraient besoin. Cette disponibilité de l'information va jouer un rôle important dans la prise de décision tant du côté de l'ATC avec tous les paramètres connus afin de garantir la trajectoire la plus efficace dans les meilleures conditions, que du côté de l'aéronef en permettant au personnel de bord d'anticiper des situations contraignantes (exemple : fermeture de piste à l'aéroport d'arrivée ou turbulence sur la trajectoire).

2.1.2 Organisation de l'échange d'information au sein du SWIM

Afin de décrire la façon dont le SWIM organise le partage de l'information, cinq couches sont mises en place [10]. La figure 2.2 présente ces différentes couches et les relations entre elles. Dans la couche de plus haut niveau, on retrouve les applications qui correspondent aux fournisseurs et aux utilisateurs de l'information transitant via le SWIM. La couche suivante correspond aux services d'échange de l'information permettant de regrouper l'information par type (données aéronautiques, données de vol, données météorologiques...) et de les fournir aux utilisateurs intéressés. La troisième couche correspond quant à elle à la couche des modèles d'échanges. On y retrouve principalement les trois modèles d'échanges de données que sont FIXM (*Flight Information Exchange Model*), AIXM (*Aeronautical Information Exchange Model*) et WXXM (*Weather Information Exchange Model*). Ces modèles

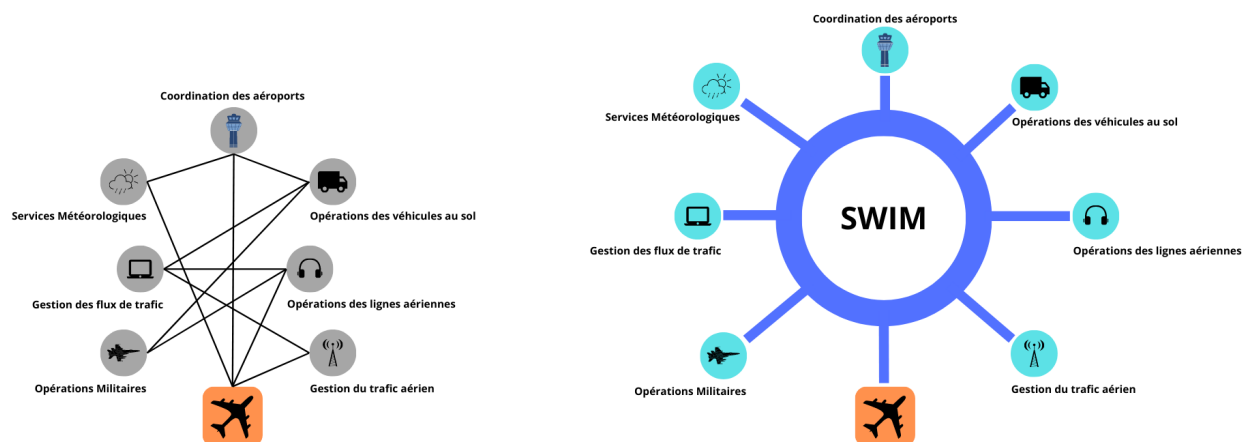


FIGURE 2.1 Passage d'une architecture à liens directs (gauche) une architecture orientée services (droite)

permettent l'organisation des données, ainsi que leur encodage (voir 2.1.3). La couche dite d'infrastructure est la suivante et est celle permettant le partage de l'information par le biais d'abonnements avec un fonctionnement en éditeur-abonné, en pousser-tirer ou encore par le principe de demande-réponse. Finalement, la dernière couche correspond à l'aspect réseau permettant la connexion des différents systèmes sur la base d'un réseau IP.

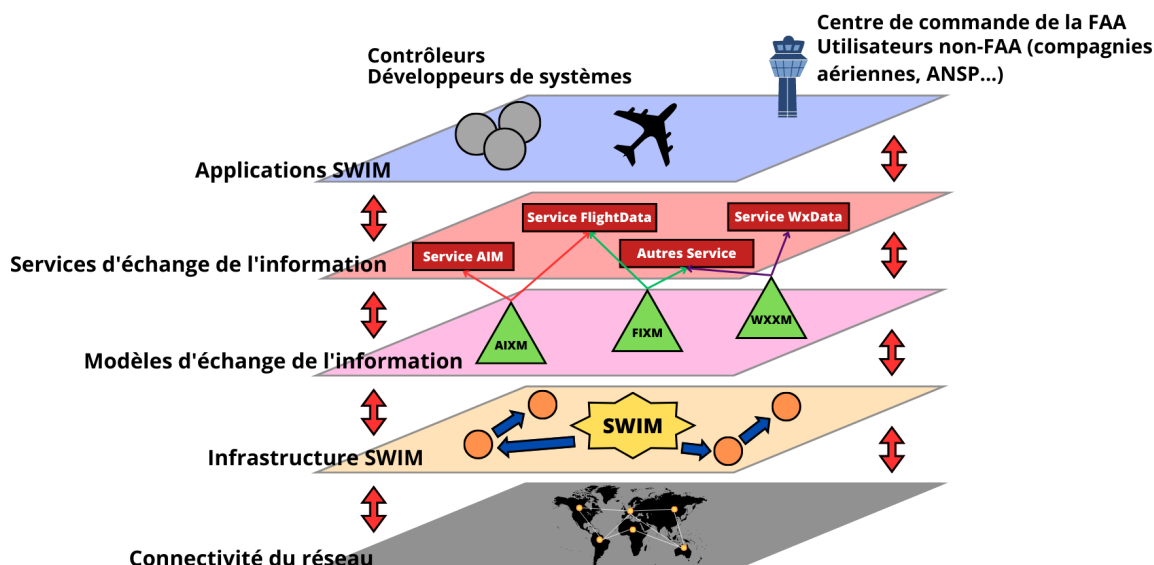


FIGURE 2.2 Organisation des différentes couches dans le système SWIM, inspiré de [10]

2.1.3 Mise en forme des données

L'un des principaux problèmes de l'ancien système d'échange de données avec une architecture à liens directs était la cohérence des données. En effet, lorsque deux systèmes ne parlent pas le même langage, des objets ayant la même signification peuvent se retrouver interprétés différemment. Il est alors nécessaire de fournir un vocabulaire commun afin que toutes les données soient présentées sous le même format et que les systèmes puissent les interpréter de la bonne façon. Comme évoqué dans la sous-section 2.1.2, la troisième couche du SWIM correspond à la couche des modèles d'échanges et est composée principalement par trois modèles : FIXM, AIXM et WXXM. Ces modèles sont donc ce langage commun évoqué précédemment et ont pour objectifs de définir des liens entre les différents éléments définissant une information.

Le modèle FIXM [12], acronyme de Modèle d'échange d'informations de vol (en anglais *Flight Information Exchange Model*), se concentre principalement sur l'échange d'informations de vol tels que les plans de vol, les trajectoires, le type de vol (militaire, aviation civile...), les performances des avions ou encore les informations sur le départ et l'arrivée. Les messages ainsi composés seront utilisés par les services de contrôle du trafic aérien, les compagnies aériennes et d'autres services d'aviation.

Le modèle AIXM [13, 14], acronyme de Modèle d'échange d'informations aéronautiques (en anglais *Aeronautical Information Exchange Model*), permet l'échange d'informations aéronautiques entre différents systèmes. On retrouve parmi celles-ci des données portant sur les aéroports, l'accessibilité aux pistes, aux espaces aériens, sur les aides à la navigation, ainsi que sur les tours de contrôle. Les messages construits sur la base des règles imposées par l'AIXM (par exemple dans les services STDDS et SFDPS décrits dans A et A) seront utilisés par plusieurs acteurs tels que les aéroports, les services de la navigation aérienne ou encore les compagnies aériennes.

Finalement, le modèle WXXM [15], acronyme de Modèle d'échange d'informations météorologiques (en anglais *Weather Information Exchange Model*), permet de standardiser l'échange de messages portant sur la météorologie. En effet, il permet aux différents systèmes d'échanger des prévisions météorologiques, des observations, et d'autres informations relatives à la météo. Le modèle est basé sur la norme d'échange d'informations de l'Organisation météorologique mondiale (OMM) et utilise, comme AIXM et FIXM les normes de la série ISO 191xx portant sur les informations géographiques. Les règles définies par le WXXM permettront la construction des messages portant sur des informations météorologiques (par exemple au travers du service ITWS dans A) et échangés entre les différents acteurs que sont les contrôleurs aériens, les services météorologiques, et les compagnies aériennes. Ces trois modèles (FIXM,

AIXM, WXXM) sont formés de trois composantes : les schémas d’application, de base (*core*) et d’extension. Les schémas d’application définissent les éléments et champs nécessaires à la représentation d’une information spécifique à un domaine (exemple : les aéroports pour AIXM ou les prévisions météorologiques pour WXXM). Les schémas de base sont quant à eux plus généraux et permettent la représentation des données qui sont partagées au travers de plusieurs de ces domaines, telles que des informations spatiales ou temporelles. Finalement, les schémas d’extension fournissent des informations supplémentaires sur des cas spécifiques de schémas d’application qui ne sont pas couverts par les schémas de base. On retrouvera par exemple des informations régionales qui ne s’appliqueraient pas partout dans un pays.

Afin de définir un message envoyé rendant compte de l’état d’un vol, plusieurs catégories seront ainsi imbriquées. L’utilisation de ces modèles basés sur le langage XML (*Extensible Markup Language*) permet ainsi de standardiser les échanges de données aéronautiques, de vol et de météo. Ils permettent également d’établir une hiérarchisation des données permettant de regrouper les informations entre elles et de créer un cheminement logique vers une information recherchée. Afin de représenter cette hiérarchisation, des diagrammes UML (*Unified Model Language*) sont utilisés. Le XML fait partie des langages de balisage. Ce sont des langages utilisant une structure à base de balises afin d’organiser et de contenir des séquences de caractères. Plusieurs éléments composent un document XML : l’en-tête qui informe sur la version XML utilisée ainsi que son encodage (l’UTF-8, l’UTF-16 ou encore l’ASCII) ; l’élément racine qui est l’élément parent de tous les autres et qui donne le contexte des données ; les attributs qui sont ajoutés dans les balises pour préciser le contexte d’un élément ; les valeurs se trouvant entre les balises de début et de fin et qui contiennent les données en question. La figure 2.3 permet d’illustrer les différents éléments décrits ci-dessus par l’entremise d’un exemple de message XML rendant compte des informations fournies sur un vol. En effet, on peut ainsi voir sur la première ligne la déclaration de l’en-tête précisant la version 1.0, suivie de la définition de l’élément racine <flight> qui va alors contenir toute l’information décrite dans la suite. On compte ensuite sept éléments enfants à <flight>, ce sont <from>, <to>, <airline>, <departure>, <arrival>, <duration> et <price>. Certains de ces éléments possèdent à leurs tours des éléments enfants tels que <departure> possède <date> et <time> tandis que d’autres tels que <price> définissent des attributs (ici “currency” afin de définir la devise utilisée). Finalement, on retrouve entre les bornes de ces éléments des valeurs afin de caractériser ces champs. On comprend ainsi que le vol concerné sera entre Paris CDG et New York JFK, opéré par Air France le 01/03/2023 à 10h30. L’utilisation du XML permet ainsi d’avoir accès à un format dont la lisibilité est possible tant par l’homme que par la machine et qui est extensible, dans le sens où des éléments peuvent être ajoutés à volonté afin de donner le plus de contexte à l’information transmise.

```

<?xml version="1.0" encoding="UTF-8"?>
<flight>
  <from>Paris CDG</from>
  <to>New York JFK</to>
  <airline>Air France</airline>
  <departure>
    <date>2023-03-01</date>
    <time>10:30</time>
  </departure>
  <arrival>
    <date>2023-03-01</date>
    <time>16:45</time>
  </arrival>
  <duration>7:15</duration>
  <price currency="EUR">500</price>
</flight>

```

FIGURE 2.3 Exemple de message XML

2.1.4 Accès de l'aéronef au SWIM

Différentes études ont porté sur la mise en place de l'infrastructure nécessaire à la connexion de l'avion au SWIM. Bien que les technologies employées varient afin d'établir la connexion avec l'avion, toutes sont basées sur la communication par internet. On retrouve ainsi des expériences utilisant des systèmes satellitaires tels qu'Inmarsat [16], tandis que d'autres privilégient des systèmes au sol tels que l'AeroMACS (pour système de communication mobile aéronautique pour les aéroports) [17,18] ou encore Gogo Inflight Internet (anciennement Air-cell) [16,19], un fournisseur de services internet en vol basé aux États-Unis fonctionnant grâce aux quelques centaines d'antennes réparties au travers du pays. Cependant, d'après [20], bien que l'essentiel des réseaux cellulaires 3G/4G soit utilisable pour ces échanges de données, ils restent cantonnés à des utilisations à des altitudes inférieures à 10 000 pieds et dans des zones au-dessus du continent américain. Pour ce qui est des systèmes satellitaires, bien qu'ils peuvent couvrir la quasi-totalité du globe, leur utilisation est restreinte aux régions polaires et aux espaces océaniques. Une liaison par satellite permettrait un débit variant entre 32 et 650 Kbits/s tandis que les réseaux cellulaires permettraient d'atteindre près de 9.8 Mbits/s [20].

Comme évoqué précédemment dans la section 2.1.2, plusieurs patrons de communication sont utilisables pour la mise en place d'une architecture SWIM. Celui que l'on retrouve le plus dans les travaux reste tout de même le patron éditeur-abonné (ou *publish/subscribe*) [16,17,20] permettant aux pilotes de s'abonner à un produit afin d'être assurés de recevoir les mises à jour de celui-ci [20]. Cela a été mis en pratique dans [17] lorsque le vol en question s'est

abonné à tous les messages entrants le concernant, et qu’il commençait à les recevoir une fois la connexion établie. Cependant, d’autres patrons tels que la demande-réponse (ou *request-reply*) [16, 17, 19] ou le pousser-tirer (ou *push-pull*) [16] sont testés. Le patron demande-réponse permet aux pilotes de demander aux services au sol des informations supplémentaires par exemple concernant des événements météorologiques (rapports METAR ou TAF). Il est généralement utilisé pour des informations statiques et ne nécessitant pas forcément du temps réel [17]. Un patron pousser-tirer est quant à lui utilisé dans des cas d’urgences où les services au sol cherchent à transmettre une alerte à un certain vol [17].

Pour ce qui est des données partagées, ces différentes mises en application de l’AAAtS montrent plusieurs utilisations faites de cette communication. Alors que [17] et [18] se concentrent principalement sur l’utilisation du SWIM lors de la navigation de surface en échangeant des informations de surveillance, de météorologie ou sur l’évolution du statut de l’aéroport avant le décollage, les autres tests évoquent directement des applications en vol. En effet, [20] prouve la disponibilité des données aéronautiques (tels que les NOTAM) et météorologiques (rapports METAR, TAF, AIRMET, SIGMET, PIREP ou NEXRAD) directement dans le poste de pilotage grâce aux services du SWIM. L’aspect bidirectionnel de la communication est également mis en avant avec la publication d’informations de vol et de rapports météorologiques directement dans les services SWIM les rendant disponibles aux services au sol, mais également aux autres avions dans le secteur. Des informations portant sur la gestion de la trajectoire et sur les préférences des pilotes lors de l’initiation d’une correction de la trajectoire 4D sont également partagées par ce biais [19]. Les retours des pilotes sur ces expériences ont permis de mieux appréhender la question du développement de ce type de produits. En effet, ces derniers ont trouvé utile la possibilité d’avoir accès à des informations globales dans un but de planification et d’anticipation sur le long terme [16]. Les données météorologiques et opérationnelles actuelles et prévisionnelles permettent alors des phases de descente et d’approche plus prévisibles. L’étude de plusieurs scénarios comparatifs avec et sans AAAtS permet de comprendre l’évolution apportée du point de vue des pilotes [21]. On prend ici l’exemple du vol numéro 2 de l’étude, lors duquel le vol fait face à tempête de neige provoquant la fermeture de la piste à l’arrivée. On remarque que grâce à la préparation pré-vol et la connexion aux services de messagerie SWIM, les pilotes sont tenus au courant tout au long du vol de l’évolution des conditions à l’arrivée et des possibles turbulences sur leur chemin. Les différents rapports reçus indiquent des zones de turbulences et les pilotes peuvent ainsi les éviter ou préparer la cabine à la perturbation, ce qui n’était pas forcément le cas sans l’AAAtS. De plus, tout au long de la fermeture de la piste et de l’attente, les pilotes sont tenus au courant des retards prévus et préparent en parallèle une possible déviation. Ce travail permet une nouvelle fois une anticipation de possibles conséquences de ce retard, bien

que dans ce scénario la situation se rétablisse à la normale et que l'avion atterrisse à l'endroit souhaité. Ces différents scénarios présentés dans [21] mettent en avant l'amélioration de la conscience de la situation des pilotes et en leur permettant de mieux anticiper les différents scénarios auxquels ils pourraient avoir à faire face, permettant donc de mieux assurer la sécurité du vol.

Finalement, plusieurs supports ont été évoqués afin de permettre aux pilotes de visualiser l'information. Les sacs de vol électroniques (*electronic flight bag* - EFB) apparaissent dans la totalité des travaux étudiés. En effet, ces équipements ne nécessitent pas de certification et peuvent être installés dans le cockpit, généralement dans l'axe de l'épaule des pilotes [22,23]. L'utilisation de ces EFB a été le sujet de plusieurs études afin de comprendre l'impact qu'ils auraient sur la présentation de l'information aux pilotes et notamment sur les conflits d'informations engendrés en comparaison avec l'information affichée sur les systèmes habituels utilisés [23]. L'installation de ces EFB a ainsi été largement approuvée par les pilotes, car plus accessibles que les affichages principaux, et permettant la comparaison des données fournies par plusieurs systèmes, augmentant donc leur conscience de la situation [22]. Les types d'EFB varient selon les situations mais les pilotes privilégient généralement des appareils tels que les IPAD [16] ou bien d'autres appareils développés spécialement pour cette utilisation (exemple : navAero dans [16]). Dans les applications de l'AAtS, ces EFB étaient notamment utilisés pour afficher les informations reçues du SWIM et pour entrer directement les requêtes de données additionnelles [19]. Les retours des pilotes sur ces différentes expériences d'utilisation des EFB ont également permis d'affirmer la nécessité de proposer des interfaces plus claires pour présenter l'information [16,19]. En effet, le design des interfaces est qualifié de "crucial" car il affecte la compréhension de l'information de la part des pilotes et donc de l'utilisation qu'ils pourraient en faire. Une mauvaise compréhension de l'information, par exemple une information météorologique périmée, peut contribuer à une utilisation erronée et donc à la création d'un potentiel danger.

Ainsi, depuis quelques années déjà, l'intégration du SWIM dans les cockpits est un axe de recherche important. Les différents travaux étudiés ici permettent de comprendre la faisabilité du concept de communication air-sol au travers du SWIM. En augmentant la quantité d'information rendue disponible aux pilotes, le SWIM permettrait d'améliorer la conscience de la situation des pilotes et une meilleure interaction avec les équipes au sol sur divers sujets. Cependant un enjeu majeur reste tout de même la présentation de ces données aux pilotes. Les interfaces réalisées sur des appareils non certifiés tels que les EFB se doivent de répondre à de nombreux critères d'un point de vue des facteurs humains [22,23] et ne pas engendrer une charge de travail supplémentaire aux pilotes.

2.1.5 Portail SWIFT

Dans son objectif de recherche et de développement des applications du SWIM, la FAA a créé le portail SWIFT¹, acronyme pour *SWIM Industry-FAA TEAM*. Ce portail est accessible sur la base d'une inscription à tous les utilisateurs et donne accès aux messages publiés par les systèmes SWIM de la FAA. Afin d'accéder à ces données, des demandes d'abonnement aux différents éditeurs de données sont nécessaires. Parmi ces éditeurs, on retrouve alors : STDDS (*SWIM Terminal Data Distribution System*), ITWS (*Integrated Terminal Weather Service*), TFMS (*Traffic Flow Management System*), TBFM (*Time Base Flow Management*), SFDPS (*SWIM Flight Data Publication Service*) et AIM FNS (*Aeronautical Information Management Federal NOTAM System*). Chacun de ces éditeurs fournira des informations relatives à la météo, à la surveillance du trafic aérien, à l'environnement aéronautique ou encore aux vols et flux aériens (*flight and flow*). La figure 2.4 présente l'ensemble des éditeurs proposés au travers du portail SWIFT.



FIGURE 2.4 Éditeurs de données présents sur le portail SWIFT

2.2 État de l'art des solutions existantes

2.2.1 Patrons de communication

Comme observé au travers des différents travaux portant sur le SWIM, plusieurs patrons de communications ont été évoqués afin de permettre les échanges de messages. D'après [17], les

1. <https://portal.swim.faa.gov/home>

échanges sol/sol et air/sol seront permis majoritairement grâce aux patrons éditeur-abonné, demande-réponse et pousser-tirer.

Pousser-Tirer

Le patron de communication Pousser-Tirer (ou *Push-Pull*) [24,25] définit un fonctionnement en pipeline dans laquelle des données sont poussées aux consommateurs qui les traitent et les envoient ensuite à un serveur en bout de réseau. Il s'agit d'une communication à sens unique dans laquelle les données sont toujours transmises en aval, ne bloquant alors pas les autres opérations car ne nécessitant pas de réponse directe. Il est également possible de partager les données vers plusieurs consommateurs. Les données sont alors partagées de manière égale entre tous les consommateurs avant que les données issues de leur traitement ne soient ensuite regroupées dans un collecteur en fin de parcours. Ce type de patron devrait être employé dans le cadre du SWIM avec la publication d'informations d'urgences [17].

Demande-Réponse

Le patron de communication demande-réponse (ou *Request-Reply*) [26,27] apparaît comme le patron de communication standard consistant en l'émission d'une requête par l'un des sockets qui entre ensuite en attente d'une réponse qui sera fournie par un socket visé. Ce patron se veut synchrone et bloquant dans le sens où dès qu'une requête a été soumise, aucune tâche n'est effectuée tant qu'une réponse n'a été retournée [26]. Les demandes-réponses soutiennent un style de communication axé sur la demande qui rappelle les architectures client/serveur. Pour ce qui est de leur fonctionnement, une enveloppe est transmise avec la requête et contient l'adresse du socket afin de retourner la réponse. Les utilisations de ce patron de communication sont cependant assez limitées étant donné cet aspect bloquant empêchant l'exécution d'autres tâches en parallèle le temps de l'attente d'une réponse. Dans le cas d'une utilisation conjointe au SWIM, le patron de communication demande-réponse peut être associé au partage d'informations dites statiques [17] qui n'évoluent pas en temps réel. On peut notamment évoquer des évolutions météorologiques qui se présenteraient sous la forme de rapports et qui seraient demandés en vol par les pilotes.

Éditeur-Abonné

Finalement, nous abordons le patron de communication éditeur-abonné revenu à plusieurs reprises dans la littérature par son utilisation dans les premiers essais de connexion de l'avion au SWIM. Ce dernier permet l'acheminement de données d'un éditeur vers un ou plusieurs abon-

nés en fonction de l'intérêt de ces derniers pour un type spécifique de données. La principale caractéristique de ce patron de communication est sa capacité à dissocier les participants en connectant anonymement les producteurs de données aux consommateurs, c'est le découplage spatial. Il existe également un découplage temporel [28, 29], ne nécessitant pas la présence des deux entités qui sont le producteur de données et son consommateur en même temps pour fonctionner. Le modèle éditeur-abonné répond à un style de communication basé sur des événements en permettant la diffusion de l'information. Nous pouvons identifier plusieurs éléments qui vont participer à cet échange [29], nous les listons ci-dessous : L'éditeur fournissant les données ; le *DataWriter* en interface avec l'éditeur, permettant à ce dernier de savoir qu'un changement d'une valeur a eu lieu ; l'abonné qui reçoit les données ; le *DataReader* permettant à l'abonné d'accéder aux données en fonction du type de données ; et le sujet (ou *Topic*) associant un nom, un type de données et une qualité de service (*QoS*). Plusieurs mécanismes sont utilisés pour acheminer les messages entre l'éditeur et un abonné [30]. On retrouve ainsi par exemple l'utilisation de la multi-diffusion (un à plusieurs) vers les appareils d'un réseau qui traiteront le message s'il correspond à l'information recherchée, mais aussi l'emploi d'une liaison point à point (*peer-to-peer*) avec les consommateurs par le biais de protocoles un-à-un tels que le TCP/IP. Finalement des intermédiaires (*brokers*) peuvent être utilisés et permettent de recevoir directement les demandes d'abonnements. Il est cependant important de noter que la première solution n'est applicable que localement. Pour ce qui est de la liaison point à point, elle est très commune mais ne convient qu'à un nombre faible d'abonné. Finalement, les intermédiaires sont utilisés généralement dans les environnements distribués [30, 31]. Dans le cadre du SWIM, ce type de patron de communication peut être employé afin de partager des informations dynamiques et en temps-réel [17].

2.2.2 Utilisation d'un middleware

Intégration des middlewares à l'avionique

Avec le passage d'une architecture fédérée, dans laquelle toutes les fonctions avioniques étaient réparties sur des ressources informatiques distinctes, à une architecture modulaire [32], dans laquelle ces ressources se retrouvent partagées entre des fonctions de criticités différentes, de nouvelles approches de développement sont apparues. C'est dans l'objectif de réduction des coûts engendrés par le poids croissant des équipements et leur consommation d'énergie que cette transition a été initiée. Elle s'accompagne cependant d'une complexification de l'architecture nécessitant alors l'introduction de nouveaux outils afin d'optimiser le développement de ces nouveaux systèmes avioniques. Le concept de middleware est ainsi intégré dans les systèmes avioniques afin d'accélérer le développement des composants. C'est au travers

du standard FACE (pour *Future Airborn Capability Environment*) [33] que le concept de middleware apparaît dans le contexte avionique. En effet, ils permettent d’assurer “l’interopérabilité dans le segment des services de transport”. Ils amélioreraient ainsi la productivité en facilitant à la fois la programmation, la maintenance ainsi que le débogage des applications développées. Bien qu’utilisés depuis des dizaines d’années principalement sur des applications dont la criticité était peu élevée, des middlewares ont déjà été employés dans des tâches plus complexe tel que ce fut le cas avec CORBA sur des systèmes de combat ou avec DDS sur la gestion du trafic aérien [33]. Des questionnements sont également nécessaires vis-à-vis de leur certificabilité pour une intégration dans les systèmes avioniques. En effet, bien qu’ils soient employés dans de nombreux domaines, ils doivent répondre aux exigences en termes de fiabilité et sécurité notamment soulevés par les standards DO-178C² (pour l’aspect logiciel), DO-254³ (pour l’aspect matériel) et DO-297⁴ (pour l’avionique modulaire intégrée). Nous évoquerons dans les sous-parties suivantes, quelques solutions basées sur un patron de communication éditeur-abonné afin de comparer leurs utilisations et performances.

Zenoh

Zenoh (pour *Zero Overhead Network Protocol*) est un middleware basé sur le patron éditeur-abonné présenté comme une solution unifiée et transparente pouvant être utilisée sur des réseaux hétérogènes [34, 35]. Il permet de fournir une abstraction aux développeurs leur permettant d’interagir directement avec les données sans avoir à prendre en compte les différents types de réseaux et protocoles. Zenoh appartient à ce que l’on nomme le Réseau de données sur les noms (ou *Names Data Network* - NDN) qui est un concept proposant un adressage directement sur les données plutôt que sur les machines hôtes. En effet, les données sont alors identifiées par une paire correspondant à leur nom ainsi qu’une valeur, ouvrant alors de nouvelles possibilités au niveau des requêtes en utilisant ces noms. Ce passage par un réseau NDN permet la réduction du trafic en retirant le besoin d’effectuer des recherches multiples aboutissant aux mêmes données [34, 35]. D’un point de vue des avantages proposés, Zenoh se veut flexible et extensible pouvant s’adapter rapidement à un nombre élevé d’applications, et à un flux croissant de données. Cette flexibilité s’illustre par la tendance des éditeurs-abonnés à utiliser un protocole TCP pour assurer leur fiabilité [34], alors que Zenoh peut quant à lui opérer sur plusieurs autres protocoles de transport afin de s’adapter. Il est également intéressant de noter que Zenoh peut également fonctionner au-dessus d’autres protocoles tels que MQTT ou DDS.

2. DO-178C, Software Considerations in Airborne Systems and Equipment Certification

3. RTCA DO-254 / EUROCAE ED-80, Design Assurance Guidance for Airborne Electronic Hardware

4. DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations

DDS

DDS (pour *Data Distribution Service*) est un standard développé par le consortium *Object Management Group* (OMG), concernant la communication dans les systèmes distribués sur la base d'un patron de communication éditeur-abonné [28,30]. C'est un système centré sur les données. DDS utilise des objets-données pour représenter les informations qui sont échangées entre les éditeurs et les abonnés. Les mises à jour de ces objets-données sont distribuées sous la forme de messages. Afin de gérer l'identification et l'accès aux différents types de données, DDS utilise l'association d'un sujet et d'une clé [28]. Alors que le sujet représente un sous-ensemble des données transmises de l'éditeur à l'abonné, la clé permet quant à elle d'identifier de façon unique une instance de ces données. Cette association permet ainsi d'accéder efficacement à des données spécifiques en fonction d'un besoin.

Chaque sujet est annoté de ses qualités de services qu'il doit respecter. On retrouvera des exemples de qualités de service de DDS au travers du tableau 2.1. Celles-ci permettent aux applications de contrôler la manière de distribuer les données au travers du DDS [36].

TABLEAU 2.1 Exemples de qualités de service assurées par DDS

Qualité de service	Caractéristiques
Échéance	Intervalle de temps maximal entre l'envoi et la réception de données
Histoire	Quantité de données gardées en mémoire avant d'être effacées par les suivantes
Budget de latence	Délai maximal afin de livrer les données
Durée de vie	Durée pendant laquelle les données sont considérées comme valables
Propriété	Permettant à un lecteur de recevoir les données d'un seul rédacteur
Fiabilité	Choix du renvoi en cas de perte des données
Priorités de transport	Autorise à donner une priorité de transport différente selon les rédacteurs
Sélection de transport	Autorise à choisir différents transports (TCP, UDP,...) pour chaque lecteur et rédacteur

MQTT

MQTT (pour *Message Queuing Telemetry Transport*) est un protocole de messagerie déployé initialement pour le domaine de l'internet des objets. Sa principale caractéristique par rapport à d'autres est sa légèreté lui permettant de fonctionner dans des environnements restreints et à faibles ressources [30,37]. De par cette légèreté, il s'intègre bien dans des réseaux à faible bande passante. Il utilise également le patron éditeur-abonné et est basé sur l'utilisation de sujets pour le filtrage. Sa communication est assurée majoritairement par le biais d'une connectivité TCP/IP. Trois niveaux de qualité de service sont proposés par MQTT afin de

répondre aux besoins de l'utilisateur : le premier correspond à un envoi sans garantie de réception, le second à une livraison assurée au moins une fois, tandis que le dernier implique une livraison exactement une fois [30]. Ainsi, grâce à sa légèreté et son efficacité dans des environnements restreint, et par le fait de son utilisation popularisée, MQTT apparaît comme un protocole de communication robuste et facilement déployable.

Kafka

Basé sur un système d'éditeur-abonné similaire à MQTT, Apache Kafka est une plateforme de messages en continu, conçue afin de collecter, traiter et stocker des flux de données en continu [37, 38]. Kafka utilise un fonctionnement à base de intermédiaires afin de réaliser le stockage, la gestion et la réplication des données. Chaque hôte est accompagné d'un broker qui lui est propre. Ils permettent ainsi une meilleure extensibilité et de gérer les pannes car les données sont répliquées et stockées, et peuvent ainsi être réutilisées [38]. Grâce à son architecture Kafka offre un flux de données à haut débit et à faible latence. Pour ce qui est de la qualité de service, Kafka présente tout comme MQTT les trois niveaux de service : sans garantie de livraison, au moins une fois et exactement une fois. Ainsi, Kafka propose une solution à base du patron éditeur-abonné capable de distribuer les messages en continu grâce à un traitement de ces derniers, tout en assurant leur stockage en cas de besoin à long terme.

Comparaison des solution présentées

Maintenant que plusieurs solutions permettant un traitement et une communication en continu de message d'un (ou plusieurs) éditeur vers des abonnés, nous nous concentrons sur la comparaison de leurs performances. Plusieurs travaux ont été menés afin de réaliser une étude comparative des performances de ces solutions et on s'appuie ici sur celle réalisée dans [37]. Il est important de noter que dans cette étude et en raison de la configuration de MQTT et Kafka ne permettant qu'une utilisation avec des intermédiaires (alors que Zenoh et DDS autorisent la liaison point à point), c'est cette configuration qui sera utilisée pour tous. Deux scenarios ont été mis en place pour réaliser ces mesures. On se place dans un premier temps dans le cas de l'utilisation d'une seule machine donnant alors accès à une bande passante quasi-illimitée, puis dans le cas de l'utilisation de machines multiples connectées à réseau physique. Dans tous les tests réalisés portant sur le débit, Zenoh est apparue comme la meilleure solution des quatre avec un débit maximal de 67 Gbps sur une seule machine et 51 Gbps sur plusieurs machines. À titre de comparaison sur plusieurs machines, Kafka atteint 5Gbps avant de saturer en raison d'une charge trop importante, DDS atteint 26 Gbps

et MQTT 9Gbps avec des performances se dégradant avec l'augmentation de la charge en données. Pour ce qui est de la latence Zenoh se retrouve cette fois-ci en deuxième position derrière DDS mais avec une latence plus faible que Kafka et MQTT. Des mesures ont tout de même été réalisées avec Zenoh en connexion point à point dans cette étude afin d'avoir une comparaison et les performances en sont d'autant meilleures. Dans ce cas, la latence se retrouve divisée par un peu plus de deux. Une seconde étude a été menée dans [39] sur DDS, MQTT et une troisième solution non évoquée dans notre revue, ZeroMQ. Les résultats ont montré que DDS présentait dans la majorité des scénarios étudiés une meilleure latence et un meilleur débit. La comparaison de ces quatre solutions nécessite également de nous attarder sur les qualités de services proposées. DDS, MQTT et Kafka proposent une fiabilité élevée en garantissant la livraison des messages, alors que du côté de la durabilité Kafka est la seule solution proposant un stockage sur le long terme des données permettant leur redistribution en cas de besoin. Finalement, on rappelle la capacité de Zenoh à pouvoir fonctionner au-dessus d'autres protocoles et offrant donc une interopérabilité intéressante.

CHAPITRE 3 DÉFINITION DES OBJECTIFS DE RECHERCHE

3.1 Objectifs

Bien que les travaux sur le SWIM n'aient à ce jour pas encore aboutis à une infrastructure fonctionnelle et que l'alimentation des avions en données SWIM ne soit pas encore possible, le portail SWIFT offre la possibilité de contourner cette contrainte et d'avoir accès à ces données. Dans le cadre de notre projet de recherche initié par un besoin des industriels du secteur de l'aéronautique, nous cherchons à étudier l'impact de l'introduction des données SWIM dans le cockpit sur les pilotes. C'est ainsi pour cette raison que nous avons choisi d'utiliser le portail SWIFT comme sources des données et de bâtir un pont entre ce portail et les interfaces de cockpits afin de les alimenter avec des données réelles.

N'ayant pris connaissance des références mentionnées à la section 2.2 qu'après le développement de notre solution, nous avons initialement décidé de développer une plateforme réalisant ce pont évoqué entre le portail et les logiciels de création d'interface. Nous reconnaissons cependant que le choix de passage par un middleware existant aurait permis d'accélérer l'implémentation et de nous concentrer plus sur la performance de cette plateforme plutôt que sur le traitement et la compréhension des données SWIFT. Ainsi, au travers de ce travail de développement d'une plateforme nous nous sommes imposés des conditions à respecter afin d'assurer la livraison en continu de l'ensemble des messages sortants du portail SWIFT. La première condition concerne l'accès à l'entièreté des messages provenant du portail SWIFT. En effet, le portail SWIFT peut être vu comme une boîte noire dont nous ne pouvons qu'observer la sortie. Nous devons donc assurer que l'entièreté des messages en sortant soient pris en compte par notre plateforme. De plus, nous nous imposons la nécessité d'assurer la fiabilité au travers de l'acheminement de la totalité des messages sélectionnés vers le cockpit. Finalement, nous avons voulu fixer une condition directement sur les performances de la plateforme en veillant à ce que cette dernière ne perturbe pas temporellement l'acheminement des messages vers le cockpit. Nous cherchons ainsi à rendre la plateforme temporellement transparente. Pour ce faire, nous partons du constat réalisé dans le standard RTCA DO-290 portant sur les "normes minimales de performance opérationnelle pour la représentation des informations de navigation sur les cartes électroniques" qui donne le temps maximal permis afin de mettre à jour les informations sur les affichages électroniques de cartes à 1 seconde. Nous utilisons alors cette norme comme objectif visé par notre travail en cherchant à assurer qu'un message SWIFT ne prenne pas plus de 1 seconde à atteindre les interfaces.

En résumé nos objectifs vis-à-vis du développement de la plateforme sont :

1. Accéder à l'entièreté des messages sortant du portail SWIFT
2. Assurer la livraison au cockpit de la totalité des messages sélectionnés
3. Acheminer un message SWIFT vers l'abonné en moins d'une seconde

3.2 Méthodologie

En choisissant alors le portail SWIFT comme source des données que nous souhaitons utiliser, nous constatons la nécessité d'un traitement des celles-ci avant leur utilisation. En effet, alors que le portail permet l'accès à l'ensemble des messages partagés au travers du système SWIM, chaque avion n'aura pas besoin d'avoir accès à toutes ces données, mais plutôt à un sous-ensemble de celles-ci. Ce sous-ensemble sera constitué d'informations pouvant directement l'impacter (par exemple : avions aux alentours, événements aux aéroports de départ et d'arrivée, événements météorologiques sur son trajet) et nécessitera donc l'utilisation d'un filtrage. Le rôle de la plateforme que nous développons est alors d'acquérir les messages, de les mettre en forme, de les analyser et d'envoyer ceux sélectionnés par un filtrage. En effet, en raison du contexte d'application de cette plateforme, c'est-à-dire lié à la distribution de données provenant du sol vers les avions concernées par celles-ci, nous avons dans un premier temps fait le choix d'un patron de communication de type éditeur-abonné. Ce choix s'explique notamment par la nécessité de répondre à des événements qui se manifestent par la livraison de nouveaux messages. En effet, chaque message correspond à la mise à jour d'une donnée nécessaire à l'avion qui doit lui être transmise au moment voulu. Par l'utilisation d'un tel patron de communication, nous gagnons également l'accès à la possibilité de filtrer les messages afin de transmettre uniquement les messages d'intérêt vers l'avion concerné. Nous faisons également le choix d'une liaison point à point afin de connecter directement l'éditeur (contenant tous les éditeurs SWIFT) aux abonnés.

Ce choix d'un patron de communication éditeur-abonné point-à-point nous a également guidé dans le choix du protocole de communication utilisé pour transmettre les messages entre l'éditeur et l'abonné. En effet, comme évoqué dans la revue de littérature, le TCP/IP est généralement privilégié. Ce choix est également motivé par une contrainte au niveau des objectifs du projet car les partenaires industriels nous ont mis à disposition le logiciel VAPS XT afin de réaliser la tâche de création des interfaces. En effet, ce logiciel privilégie notamment le protocole TCP/IP pour ses communications. Grâce à ce choix, nous nous assurons de répondre à l'un des objectifs que nous nous sommes fixés. En effet, la fiabilité est assurée par le protocole TCP qui permet de renvoyer un message qui n'aurait pas été reçu par l'abonné.

À la suite du développement de la plateforme, nous nous concentrerons sur la caractérisation de celle-ci au travers de l'exécution de plusieurs scénarios de tests. Ces tests permettront de connaître les conditions optimales dans lesquelles utiliser la plateforme pour collecter un nombre maximal de messages, mais également d'en étudier les performances, notamment celles en liens directs avec nos objectifs.

CHAPITRE 4 DÉVELOPPEMENT DE LA PLATEFORME SWIMAIR

Dans ce chapitre, nous aborderons le développement de la solution proposée afin d’obtenir les messages XML à partir du portail SWIFT et de les filtrer en fonction du besoin d’un abonné. Nous aborderons ensuite des cas d’usage permettant de mettre en avant la façon dont la disponibilité de ces nouvelles données pourrait permettre d’augmenter la conscience de la situation des pilotes. Le développement de cette plateforme a été réalisé en C++ sur l’environnement Microsoft Visual Studio Community 2022 (64-bit) Version 17.3.6. Nous utilisons les bibliothèques standard auxquelles nous ajoutons les bibliothèques pugixml 1.13 [40], Windows Sockets 2 (Winsock) [41] et une pour l’édition de fichiers csv [42].

4.1 Architecture de la plateforme

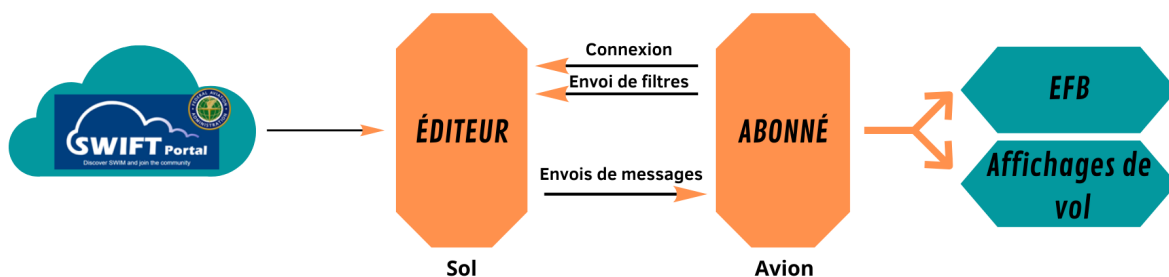


FIGURE 4.1 Architecture de la solution développée

La figure 4.1 présente l’architecture retenue pour la plateforme SWIMAIR développée. Celle-ci réalise l’interface entre les données SWIM issues du portail SWIFT, et les affichages de vol ou sacs de vol électroniques (EFB). La plateforme est ainsi divisée en deux parties. La première est “l’éditeur”, il permet l’acquisition des données auprès de SWIFT et le filtrage des messages avant de les envoyer à la seconde partie, “l’abonné”. Il est possible de voir cette architecture bipartite comme l’interaction entre une station de communication au sol, représentée par l’éditeur, et un avion qui prend alors le rôle de l’abonné. Dans ce fonctionnement, l’abonné va d’établir des filtres en fonction de l’information qu’il souhaite obtenir, et d’initier la communication avec l’éditeur afin de les recevoir. Une fois reçus, ces messages sont par la suite redirigés vers les applications adéquates sur les dispositifs d’affichages concernés. Le travail effectué dans le cadre de ce mémoire se concentre sur la mise en place de l’architecture éditeur-abonné indiquée en orange sur la figure 4.1 mais n’abordera pas la question de

la transmission et de l'utilisation des données sur des systèmes avioniques (partie verte de la figure). Cette tâche sera réalisée dans de futurs travaux sur la plateforme.

Comme évoqué dans la section 2.1.5, cinq éditeurs de données sont mis à notre disposition au travers du portail SWIFT : STDDS, ITWS, TFMS, TBFM et SFDPS. Cependant, afin d'assurer du bon fonctionnement de la plateforme et de ses performances, nous avons opté pour un développement en deux temps. La première phase consistait à obtenir un bon fonctionnement individuel de chacune des tâches afin de voir que le message traverse bien toutes les étapes, et cela en utilisant un seul éditeur de données. Dans un second temps, la plateforme est étendue à l'utilisation des cinq éditeurs afin de vérifier les performances dans son cas d'utilisation nominal.

L'un de nos objectifs lors du développement était d'obtenir une plateforme facilement extensible. Grâce à cela, il serait possible d'ajouter plus d'éditeurs de données dans le cas où le portail SWIFT en proposerait de nouveaux. De plus, il serait également possible d'adapter les capacités de traitement des messages dans le cas d'une augmentation du nombre de messages en entrée de la plateforme.

Le tableau 4.1 liste toutes les fonctions qui sont créées afin de permettre le fonctionnement de la plateforme.

TABLEAU 4.1 Fonctions créées lors du développement de la plateforme

Nom de la fonction	Bloc d'appartenance	Description
MessageGetter()	Éditeur	Permet l'acquisition des messages auprès du portail SWIFT
MessageAnalyzer()	Éditeur	Permet l'analyse des messages XML sur la base des filtres appliqués
MessageSender()	Éditeur	Permet l'établissement de la connexion avec l'abonné, la réception des filtres et l'envoi des messages sélectionnés
ConnectionToPub()	Abonné	Permet l'établissement de la connexion avec l'éditeur, l'envoi de filtres et la réception des messages sélectionnés
Signal_handler()	Abonné et Éditeur	Permet la libération des ressources lors de l'interruption de l'exécution de la plateforme

4.1.1 Outils utilisés

Bibliothèque pugiXML

Dans le développement de notre solution, nous allons utiliser la bibliothèque pugiXML [40]. Cette bibliothèque d'analyse syntaxique prend en charge le traitement de fichier XML grâce à un large éventail de fonctions. Nous avons choisi de nous tourner vers cette bibliothèque pour sa simplicité d'utilisation avec des fonctions claires donnant accès aux différentes parties des fichiers XML. De plus, sa rapidité d'exécution des différentes tâches ainsi que sa facilité d'installation, étant donnée sa taille faible, sont des atouts non négligeables. Le guide d'uti-

lisation [40] permet d’avoir accès au référencement de toutes les fonctions accessibles ainsi qu’à la façon de les utiliser avec des exemples. On pourra ainsi, en reprenant pour exemple la figure 2.3, importer le fichier XML afin d’obtenir une variable de type `pugi::xml_document` à partir de laquelle on va pouvoir accéder aux différents noeuds (`pugi::xml_node`) et attributs (`pugi::xml_attribute`) du fichier. D’autres fonctions permettent également de naviguer au travers des fichiers XML lorsque sa structure n’est pas connue en accédant au premier noeud du fichier puis en allant au suivant d’un même étage ou encore d’atteindre celui de niveau inférieur. Ainsi, cette bibliothèque répond à tous les besoins auxquels nous ferons face dans le développement de notre plateforme.

Tube

En informatique, les tubes (ou *pipes* en anglais) sont des fonctionnalités permettant d’établir une connexion entre plusieurs processus. Cet outil permet ainsi à deux programmes de communiquer entre eux et de se partager des données de façon bidirectionnelle. Cela permet à chaque processus de traiter les données au fur et à mesure qu’elles sont rendues disponibles par l’un des programmes sans devoir attendre que tout le transfert ait été effectué. On utilise souvent les pipes afin de lier des programmes en chaîne, pour que la sortie d’un des programmes corresponde à l’entrée du suivant. Nous avons choisi de nous tourner vers les tubes afin de lier notre programme à la sortie de données du portail SWIFT. En effet comme évoqué dans A, plusieurs types de sorties de données sont mises à disposition de l’utilisateur du portail. Afin de ne pas stocker les messages directement sur l’ordinateur utilisé, nous faisons le choix de diriger le flux standard de sortie (stdout) du portail vers le flux standard d’entrée du bloc éditeur de notre programme (voir 4.1). Cela est effectué grâce à l’utilisation de cinq tubes couvrant chacun des éditeurs SWIM.

Utilisation de fils d’instructions multiples

L’utilisation de fils d’instructions multiples (ou *multithreading* en anglais) est la possibilité pour le processeur (CPU) d’un ordinateur de permettre l’utilisation de plusieurs fils d’instructions (ou *threads* en anglais) pour exécuter plusieurs tâches en parallèle. Ainsi, le programme va créer plusieurs fils d’instructions s’occupant chacun d’une tâche précise en agissant principalement sur ses propres variables locales, mais également sur des variables qui peuvent être partagées entre différents fils d’instructions. Il sera cependant nécessaire d’accorder une importance considérable à la gestion des accès à ces variables partagées afin d’assurer une cohérence et éviter les problèmes de désynchronisation. Les fils d’instructions multiples sont très souvent utilisés dans le cas d’applications nécessitant l’exécution de tâches simultanées

et de façon indépendante telles que les applications de simulation utilisant un grand nombre de ressources ou encore pour le traitement de vidéos pour des questions de performances. De même pour du traitement en temps réel, les fils d'instructions multiples offrent ainsi la possibilité de gérer plusieurs flux de données en même temps et d'augmenter les capacités de calculs.

Utilisation de files

Les files (ou *queues* en anglais) sont des types d'objets en informatique permettant un stockage temporaire, en attendant d'effectuer un traitement. Des éléments peuvent ainsi être insérés un à un dans la file et sont rendus accessibles à d'autres structures selon un principe de FIFO (First In First Out - Premier entré premier sorti). Les files sont en opposition avec les piles (ou *stacks* en anglais) qui utilisent le principe de LIFO (Last In First Out - Dernier entré premier sorti). La bibliothèque "*queue*" permet d'accéder à ce type d'objet et propose des fonctions afin d'interagir avec. On a ainsi la possibilité de vérifier si des éléments sont contenus dans cette file avec la fonction `empty()`, connaître le nombre d'éléments contenus avec la fonction `size()` ou encore d'accéder aux valeurs du premier et du dernier élément de la file respectivement grâce aux fonctions `front()` et `back()`. Finalement, on ajoute des éléments en bout de file avec la fonction `push()` alors qu'on retire le premier élément avec `pop()`.

Dans le cadre du développement de l'application, ce choix de la file nous permet tout d'abord de garantir la conservation de l'ordre chronologique lors de l'accès aux messages. En effet, nous interagissons ainsi uniquement avec le premier élément de la file, tandis que dans d'autres structures on ne pourrait avoir accès qu'au dernier élément (avec les piles) ou bien à tout élément en connaissant son indice dans la structure (avec des listes) ce qui n'est pas souhaitable. De plus les files sont intéressantes dans ce contexte étant donné que les tâches permettant l'insertion d'un objet dans celles-ci et les tâches les extrayant par la suite possèdent des temps de traitement différents. Nous pouvons donc stocker les objets pendant ce laps de temps sans influencer sur leur ordre d'arrivée et donc toujours traiter en priorité les messages arrivés en premier.

En manipulant des files il est important de comprendre que des éléments peuvent à la fois être ajoutés et enlevés de la file. Cela ne pose en général pas de problème majeur, car on ajoute en général une vérification de la taille de la file avant d'en retirer des éléments afin de ne pas engendrer d'erreurs. Cependant, la section 4.1.1 portant sur l'utilisation de fils d'instructions multiples ajoute la possibilité d'avoir plusieurs fils d'instructions s'exécutant en parallèle, on peut alors imaginer la possibilité que chacun de ces fils d'instructions ajoute un élément

à une file. Ce cas présente alors un problème sur l'accès simultané des files par plusieurs entités et l'ordre dans lequel les éléments seront insérés dans la file. On peut également avoir le cas dans lequel plusieurs fils d'instructions sont chargés d'analyser la sortie d'une file et on ne peut alors pas laisser la possibilité à deux de ces fils d'instructions d'acquérir le même élément en bout de file, car cela provoquerait un dédoublement. On propose alors l'utilisation d'une classe d'objets sur la base des files, ce seront des *SafeQueues* pour rendre compte de l'aspect sécuritaire de l'utilisation de ces files vis-à-vis de leur accessibilité. Cette classe contient trois attributs : *q*, une file de type *queue*, *m* un mutex permettant d'assurer la priorité d'utilisation à un fil d'instructions unique, et *c* une variable de condition permettant de mettre en attente les autres fils d'instructions. Trois méthodes calquées sur celles des files sont mises à disposition. Elles intègrent les mutex afin de s'assurer que les files soient correctement utilisées par les bons fils d'instructions. On retrouvera ainsi `dequeue()` qui va attendre qu'un objet soit disponible dans la file avant de se saisir du mutex et d'extraire le message. La méthode `enqueue()` s'occupe quant à elle d'ajouter un élément dans la file après s'être accaparé du mutex, il est par la suite directement relâché afin de permettre aux autres de l'utiliser. Finalement, nous ajoutons une méthode `Size()` utilisant la méthode `size()` des files afin de retourner la taille des *SafeQueues* et d'étudier leur évolution au cours de la réalisation des tâches (exemple : se rendre compte d'une surcharge des files lorsque plus d'éléments entrent que n'en sortent).

4.1.2 Bloc Éditeur

Cette section s'intéresse au bloc "Éditeur" identifié à la figure 4.1. Son rôle est de mettre en forme les messages reçus depuis le portail SWIFT avant de pouvoir les analyser et d'appliquer les filtres, avant de finalement les envoyer à l'abonné. La figure 4.2 présente le fonctionnement par bloc des fonctions de l'éditeur.

La description du fonctionnement de l'éditeur suit la progression présentée par cette figure, en commençant à partir du portail SWIFT et de ses messages bruts, pour atteindre en fin de parcours l'envoi du message vers l'abonné.

Le programme utilise plusieurs fils d'instructions (voir 4.1.1) afin de paralléliser les tâches. En effet, étant donné que la réception des messages se fait en continu et que les tâches d'analyses ne pourraient s'effectuer en même temps, il est nécessaire de placer ces différentes tâches dans des fils d'instructions séparés. Nous utilisons alors trois fils d'instructions différents afin de pouvoir simultanément recevoir, analyser et envoyer les messages. Un système de files est mis en place afin de réaliser l'intermédiaire entre ces différents fils d'instructions.

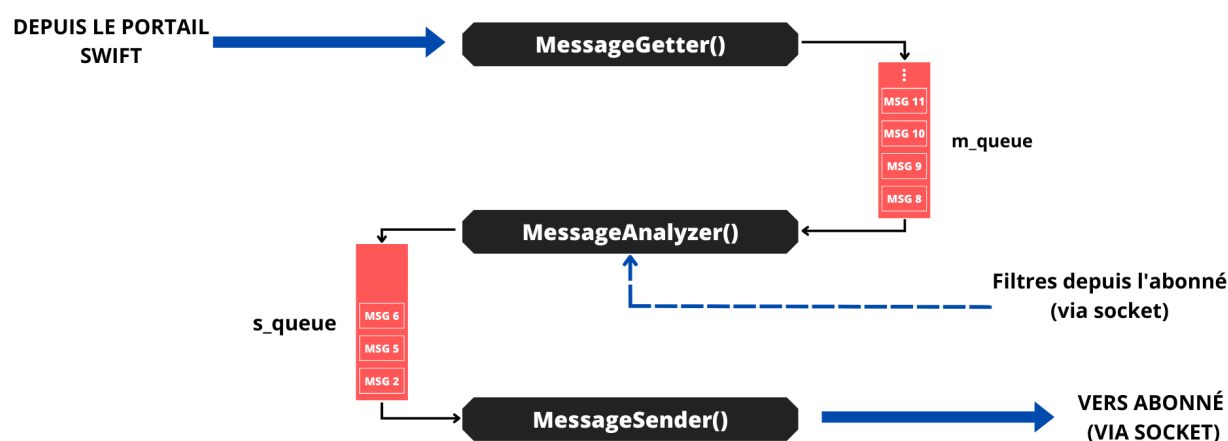


FIGURE 4.2 Organisation de la partie éditeur de la plateforme

Sortie du Portail SWIFT

Comme évoqué dans la section A, plusieurs options de sorties de données sont offertes par le portail SWIFT. Étant donnée la nécessité d'obtenir les données en temps réel dès qu'elles sont disponibles et de ne pas encombrer l'espace de stockage de la machine utilisée, nous avons choisi l'option d'une sortie « StdoutOutput ». Cette sortie permet l'impression des messages directement dans la console au fur et à mesure qu'ils sont publiés. Nous utilisons donc un tube (tel que décrite à la section 4.1.1) afin de diriger la sortie standard du SWIFT vers l'entrée de notre programme. Dès qu'un message sera posté au travers du portail SWIFT, ils sera rendu accessible au bloc éditeur.

Acquisition des messages

Une fois le lien établi avec le portail SWIFT, il était nécessaire de pouvoir récupérer les messages provenant du tube afin de pouvoir interagir avec eux. Pour ce faire, nous avons utilisé la fonction `MessageGetter()`. La figure 4.3 permet d'illustrer sous la forme d'un diagramme de flux le fonctionnement de la fonction `MessageGetter()` lors de l'apparition d'une nouvelle ligne dans le tube.

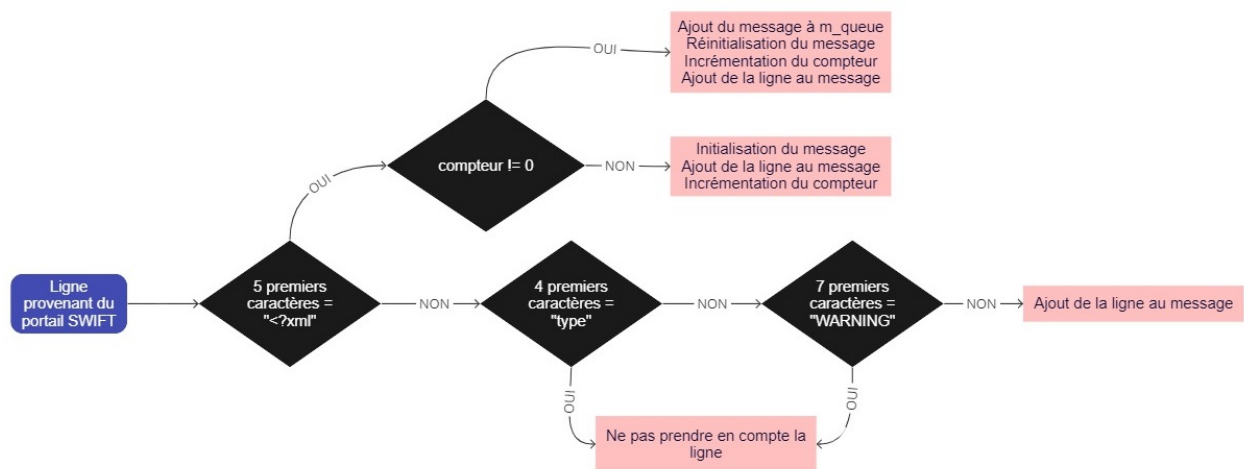


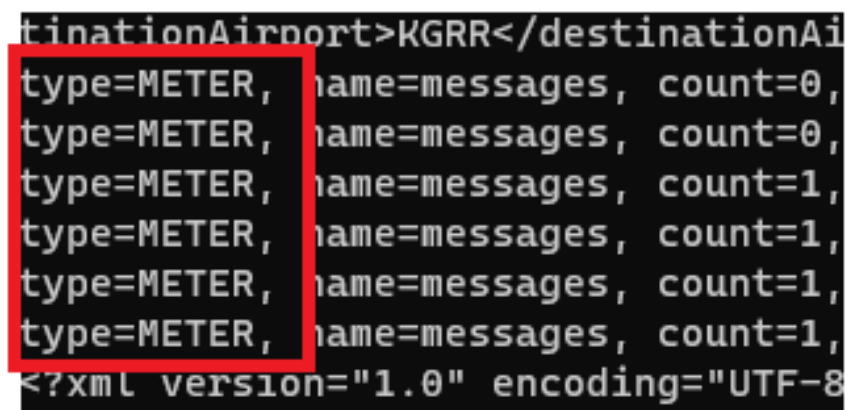
FIGURE 4.3 Diagramme de flux de l'acquisition d'un message provenant du portail SWIFT

L'idée principale de cette fonction est d'observer la sortie du tube et de récupérer ligne par ligne le contenu en découplant afin de former les messages complets sous la forme d'une chaîne de caractères (string). Ce choix de passage par des chaînes de caractères s'explique par la nécessité de recomposer les messages en collant une à une les différentes lignes provenant du SWIFT. De plus, la taille variable des messages reçus nous empêche d'utiliser directement les mémoires tampons (en anglais *buffers*) pour lesquelles il aurait fallu fixer une taille précise avant leur remplissage. La fonction prend alors en entrée le tube provenant du fournisseur SWIM concerné ainsi qu'un objet file de type *SafeQueue* (voir 4.1.1) et nommé "*m_queue*" pour queue de message.

La fonction initialise une variable **message** qui contiendra par la suite les lignes des messages issus du fournisseur SWIM. Un compteur est également initialisé afin de tenir compte du nombre de messages reçus. Finalement, un pointeur sur une chaîne de caractères (que l'on désignera par **buffer** dans la suite) est initialisé et permet de récupérer les messages à partir du tube.

L'acquisition des messages se fait alors en lisant le contenu en sortie du tube. Chaque ligne du message apparaît individuellement dans ce tube et nous permet d'identifier facilement le début d'un message. Nous avons choisi par mesure de précaution d'instaurer une limite de lecture de 2000 caractères dans l'espace de lecture du tube. Ce choix des 2000 caractères a été effectué à la suite d'une analyse des messages provenant de SWIFT afin d'inclure la totalité des caractères d'une ligne de message, quelque soit l'éditeur. La ligne va ainsi être lue jusqu'à rencontrer un caractère de fin de ligne « \n » ou bien lorsque les 2000 caractères sont

remplis (ce qui n’arrivera que dans le cas où les messages ne sont pas formatés en utilisant les caractères de fin de ligne). La ligne de message est finalement stockée dans la mémoire tampon. On transforme alors ce qui est y est stocké en chaîne de caractères avant d’effectuer un traitement à base de boucles conditionnelles afin de bien pouvoir distinguer le début et la fin d’un message. En effet, la première boucle permet de reconnaître le début d’un message sur la base de la typographie “< ?xml” présente au début de chacun des messages étudiés. Deux autres conditions OU sont également ajoutées et concernent les typographies “*type*” et “*WARNING*” qui sont présentes entre certains messages et qui ne doivent alors pas être considérées lors de l’acquisition des messages. La figure 4.4 montre un exemple de l’affichage de lignes ne devant pas être copiées dans les messages.



```

tinationAirport>KGRR</destinationAi
type=METER, name=messages, count=0,
type=METER, name=messages, count=0,
type=METER, name=messages, count=1,
type=METER, name=messages, count=1,
type=METER, name=messages, count=1,
type=METER, name=messages, count=1,
<?xml version="1.0" encoding="UTF-8

```

FIGURE 4.4 Exemple de lignes “*type*” à ignorer lors de l’acquisition de messages

Lorsque l’une des conditions “< ?xml”, “*WARNING*” ou “*type*” est vérifiée, deux nouvelles conditions vont orienter la suite des actions. Si la variable compteur indique 0 (signifiant qu’aucun message n’a encore été traité) et que **buffer** contient “< ?xml” (c’est alors le premier message) sur ces 5 premiers caractères alors le message est initialisé sans caractère et on lui ajoute la ligne contenue dans la mémoire tampon avant d’incrémenter le compteur. La seconde condition porte sur un cas similaire à la lecture d’un “< ?xml” mais cette fois-ci avec un compteur non nul. On se retrouve alors au début d’un nouveau message. Étant donné que l’on sait grâce au compteur qu’un message est déjà contenu dans la variable message, on procède dans un premier temps en ajoutant le message précédant à la file de messages **m_queue**, afin de le sauvegarder et le transmettre aux tâches d’analyses suivantes (voir 4.1.2 - Analyses des messages XML). Le compteur est alors incrémenté, et le message réinitialisé. On lui ajoute ensuite la ligne de **buffer** correspondant au début du nouveau message. Dans

les cas évoqués où **buffer** débute par “type” ou “WARNING”, la situation est ignorée et aucune action n’est effectuée afin de ne pas ajouter ces lignes dans les messages finaux.

Finalement, dans le cas où aucune des premières conditions citées n’est remplie, on se retrouve alors avec une ligne correspondant au contenu d’un message. On l’ajoute simplement au message en cours. Il est intéressant de remarquer que l’indentation des messages XML est conservée pour la majorité des fournisseurs SWIM mais ne l’est pas pour tous (elle ne le sera pas avec l’éditeur STDDS par exemple).

Analyse des messages XML

Une fois les messages acquis par la plateforme, il est question de les traiter afin d’en connaître le contenu et de savoir s’ils sont d’intérêt pour l’abonné. En effet, sur la base des filtres imposés par l’abonné, un nouveau fil d’instructions va agir en parallèle de l’acquisition des messages afin de réaliser cette sélection.

L’idée de la fonction **MessageAnalyzer()**, exécutée dans ce nouveau fil d’instructions, est d’avoir accès en continu à la file de messages obtenus par la fonction **MessageGetter()**. Les filtres sont alors ensuite comparés avec les différentes valeurs des nœuds correspondants dans les fichiers XML afin de sélectionner ou non ces messages.

La fonction prend en entrée les références à deux files de messages de type *SafeQueue*, ainsi qu’une chaîne de caractères. On ajoute également un mutex et la référence à un entier dans un but d’extensibilité de l’architecture du code. La première file de messages correspond aux messages entrants à analyser, *m_queue* évoquée précédemment, tandis que la seconde correspond aux messages sélectionnés après application des filtres et qui seront envoyés, elle est nommée *s_queue* pour *sending queue*. Finalement, la chaîne de caractères correspond aux filtres à appliquer aux messages XML. Le mutex est utilisé avec l’entier afin de garantir la cohérence d’un compteur de messages dans le cas de l’utilisation de plusieurs analyseurs de messages.

Dans un premier temps, les filtres reçus de la part de l’abonné sont extraits. L’envoi des filtres et l’établissement de la connexion entre l’éditeur et l’abonné seront décrits dans la section 4.1.3 et dans la sous-partie sur l’envoi des messages 4.1.2. Les trois différents filtres que sont l’aéroport de départ, l’aéroport d’arrivée ainsi que le numéro de vol sont contenus dans une chaîne de caractères unique sous la forme XXXYYYZZZZZZZ avec XXX l’aéroport de départ, YYY l’aéroport d’arrivée et ZZZZZZZ le numéro de vol (qui peut contenir entre 6 et 7 caractères). On les sépare donc dans de nouvelles chaînes de caractères correspondant chacune à une des trois informations afin de pouvoir les utiliser par la suite.

Une boucle infinie est utilisée afin d'analyser tous les messages ayant été stockés dans *m_queue*. Cette analyse débute par l'extraction du premier message en attente dans la file de messages en utilisant la méthode `dequeue()` de *SafeQueue*. Cette méthode le retire de la file tout en étant sûr qu'aucune autre fonction n'y a à ce moment accès par l'utilisation d'un mutex. Les messages sont analysés grâce à l'utilisation de la librairie pugiXML (présentée dans la section 4.1.1). On crée alors un objet de type `pugi::xml_document` nommé *docXML*, dans lequel le message en question est chargé. L'objet *docXML* correspond alors à un fichier XML auquel nous appliquons les fonctions mises à dispositions par pugiXML afin d'accéder à son contenu.

Il est ensuite nécessaire de déterminer à quel éditeur de données correspond le message à analyser. Malgré l'utilisation, dans un premier temps, d'un seul éditeur de données, nous mettons tout de même en place le filtrage pour tout type de messages afin de pouvoir tester la plateforme individuellement sur chacun des éditeurs. Pour ce faire, nous utilisons les fonctions `first_child()` et `name()` afin d'extraire le nom du premier noeud du fichier. En effet, il est important de remarquer que chacun des éditeurs présente un ou plusieurs premiers noeuds spécifiques, et c'est grâce ces noeuds spécifiques que nous effectuerons le filtrage. Nous listons les différents noeuds associés aux éditeurs de données dans le tableau 4.2.

TABLEAU 4.2 Association des premiers noeuds aux éditeurs SWIFT correspondants

Éditeur SWIFT	Type de message (si nécessaire)	Nom du premier nœud XML
TFMS	Flight Data	<ds:tfmDataService>
TFMS	Flow Information	<ns5:tfmDataService>
ITWS	--	<itws_msg>
SFDPs	--	<ns5:MessageCollection> <FDPSMsg>
TBFM	--	<env>
STDDS	Surface Movement Event	<ns2:asdexMsg> <ns2:SurfaceMovementEventMessage> <ns2:SafetyLogicHoldBar>
STDDS	Terminal Automation Information	<ns2:TATrackAndFlightPlan> <ns2:TASStatus>
STDDS	Tower Departure Event	<ns3:TowerDepartureEvent> <ns2:DATISData> <ns3:DATISData> <ns4:DATISData> <ns5:DATISData>
STDDS	Airport Data Service	<ns2:RVRDDataUpdateMessage> <ns3:AssetMonitorMessage>
STDDS	Infrastructure System Monitor and Control	<ns3:TowerDepartureEventMessage>

L'appel à `FirstNode(docXml.first_child().name())` permet alors de renvoyer le nom de ce noeud sous forme de chaîne de caractères, et c'est ensuite au travers de boucles conditionnelles que nous vérifions de quel éditeur le message provient. Une fois l'éditeur déterminé, il est ensuite question d'extraire du message les trois valeurs pouvant être comparées à nos filtres : les aéroports de départ et d'arrivée et le numéro de vol. En fonction du type de message, il se peut que le numéro de vol n'apparaisse pas dans les informations fournies ou qu'un seul aéroport soit présent. C'est notamment le cas des messages provenant de l'ITWS qui rendent compte des conditions météorologiques autour d'un aéroport, et ce pour tous les avions concernés. Le code ci-dessous présente la façon dont la fonction d'analyse de message XML détermine l'éditeur de données (ici comme exemple le STDDS avec des messages concernant le TDES) et sélectionne ou non le message en fonction des différents noeuds et valeurs indiquées dans ce dernier en comparant avec les filtres reçus depuis l'abonné.

```
//Research in nodes
std::string FirstNode(docXml.first_child().name());

// For STDDS (Tower Departure Event Message type) messages
if (FirstNode == "ns3: TowerDepartureEventMessage")
{
    pugi::xml_node dep_airport = docXml.child("ns3:
        TowerDepartureEventMessage").child("departureAirport");
    pugi::xml_node arr_airport = docXml.child("ns3:
        TowerDepartureEventMessage").child("enhancedData")
        .child("destinationAirport");
    pugi::xml_node aircraft_ID = docXml.child("ns3:
        TowerDepartureEventMessage").child("aircraftID");
    std::string val_dep_airport = dep_airport.child_value();
    std::string val_arr_airport = arr_airport.child_value();
    std::string val_aircraft_ID = aircraft_ID.child_value();
    if (val_arr_airport.find(filter_arrival_airport) ||
        val_dep_airport.find(filter_departure_airport) ||
        val_aircraft_ID == filter_flight_id)
    {
        s_queue.enqueue(messageFromQueue);
    }
}
```

La figure 4.5 présente un exemple de message provenant de l'éditeur STDDS du SWIFT et permet de mieux comprendre l'application du code présenté. Ainsi, la variable `val_dep_airport`

prendra la valeur “KLAX” tandis que `val_arr_airport` sera “KSEA” et `val_aircraft_ID` sera “ASA1075”. Il est également intéressant de remarquer que les filtres ont été définis selon la nomenclature de l’IATA utilisant 3 lettres alors que dans certains messages SWIM la nomenclature utilisée est celle de l’OACI utilisant 4 lettres. Cette nomenclature ajoute généralement une lettre devant et qui représente le pays (K pour les États Unis). Ainsi, nous utiliserons la fonction `find()` afin de chercher le code à trois lettres du filtre dans les valeurs qui nous seront fournies par les messages SWIM, qu’elles soient à trois ou quatre lettres.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns3:TowerDepartureEventMessage xmlns:ns3="urn:us:gov:dot:faa:atm:terminal:entities:v4-0:tdes:towerdepartureevent"
  xmlns:ns6="urn:us:gov:dot:faa:atm:terminal:entities:v4-0:tdes:reconstitutionsubscriptiontodeparturedatamessage" >
  <eventTime>2023-02-13T14:57:19.035Z</eventTime>
  <flightID>KL50801200</flightID>
  <aircraftID>ASA1075</aircraftID>
  <computerID>861</computerID>
  <departureAirport>KLAX</departureAirport>
  <clearanceDeliveryTime>2023-02-13T14:37:19.000Z</clearanceDeliveryTime>
  <parkingGate>65B</parkingGate>
  <taxiStartTime>2023-02-13T14:57:19.035Z</taxiStartTime>
  <enhancedData>
    <eramGufi>KL50801200</eramGufi>
    <sfdpsGufi>us.fdp.2023-02-13T14:06:41Z.000/11/200</sfdpsGufi>
    <destinationAirport>KSEA</destinationAirport>
    <aircraftType>B739</aircraftType>
    <beaconCode>6763</beaconCode>
  </enhancedData>
</ns3:TowerDepartureEventMessage>
```

FIGURE 4.5 Exemple de message STDDS concernant le TDES

Si l’une des valeurs d’un des noeuds correspond à la valeur d’un des filtres, le message est sélectionné et ajouté à la file “`s_queue`” afin d’être passé au fil d’instructions réalisant l’envoi des messages.

Envoi des messages

Au fur et à mesure de l’analyse des messages, la file d’envois est remplie lorsque des messages sont sélectionnés. Il est alors nécessaire de mettre en place un système afin d’envoyer ces messages sélectionnés à l’abonné. Pour ce faire, nous avons utilisé la liaison préalablement établie entre l’abonné et l’éditeur pour l’envoi des filtres, mais dans le sens inverse.

Nous utilisons une communication basée sur les sockets afin d’échanger des données dans les deux sens entre l’abonné et l’éditeur. Pour ce faire, nous utilisons la bibliothèque winsock2, version la plus récente des socket de Windows. Nous utilisons ainsi winsock afin d’obtenir une communication asynchrone en permettant l’envoi des messages lorsque ces derniers sont rendus disponibles dans la file d’envois. Ce choix s’explique notamment par sa simplicité étant donné que la plateforme a été développée sur des ordinateurs opérant sous Windows et que winsock2 est ainsi optimisé pour ce système d’exploitation. Cette bibliothèque propose des

échanges de données rapides et efficaces en utilisant un jeu de fonctions complet permettant de remplir toutes les tâches nécessaires à l'établissement et au bon fonctionnement de cette communication. Sa robustesse et sa gestion des erreurs permettent à l'utilisateur d'obtenir une application fiable et dont les performances sont bonnes.

Le fil d'instruction d'envoi des messages exécute la fonction `MessageSending()` prenant comme arguments la référence à la file de chaîne de caractères des messages à envoyer, `s_queue`, ainsi que la référence à une chaîne de caractères correspondant aux filtres au format `XXXYYYZZZZZZ`. L'idée de cette fonction est alors d'établir une connexion TCP/IP avec l'abonné, de recevoir les filtres, et d'envoyer les messages qui ont été sélectionnés. Nous allons donc discuter de ces trois tâches séparément.

Dans un premier temps, nous allons évoquer l'établissement de la connexion entre l'abonné et l'éditeur du point de vue de l'éditeur. Plusieurs étapes sont nécessaires : création du socket, liaison, écoute des connexions entrantes, acceptation de la connexion et démarrage de communication.

Nous débutons en initialisant des paramètres nécessaires à cette connexion : la structure `WSADATA` permettant de stocker l'état de la bibliothèque winsock, le socket `ListenSocket`, et un entier `iResult` qui permettra de savoir au travers de tests si les différentes étapes citées précédemment ont bien été réalisées. Nous initialisons la bibliothèque avec la fonction `WSAStartup()` donnant une valeur non nulle à `iResult` si celle-ci a échoué. Il est ensuite question de paramétrer et d'initier le socket de l'éditeur. Pour ce faire, nous utilisons la fonction `socket()` et nous vérifions si l'exécution a eu l'effet attendu (dans le cas contraire, `GetLastError()` affiche la source de l'erreur). Plusieurs paramètres sont donnés à ce socket : `AF_INET` permet d'indiquer l'utilisation des adresses IP, `SOCK_STREAM` indique le type de socket en précisant l'utilisation d'un protocole de communication orienté vers le flux de données, tandis que `IPPROTO_TCP` indique que le protocole de communication utilisé est le TCP. L'exécution de cette fonction s'accompagne de celle de `InetPton(AF_INET, L"127.0.0.1", &publisher.sin_addr.s_addr)`. Une structure de type `sockaddr_in` appelée `publisher` est définie préalablement afin de structurer la socket. Elle contient quatre champs : `sin_port` pour le port matériel, `sin_addr` pour l'adresse IP de le socket, `sin_family` pour la famille d'adresse et `sin_zero` pour avoir un structure de bonne taille. Ainsi, la fonction évoquée précédemment permet de convertir l'adresse IP indiquée en binaire afin de la stocker dans le champ `sin_addr` de la structure. Les lignes suivantes permettent quant à elles de fixer le port, en utilisant la fonction `htons()` pour assurer la lecture dans le bon ordre des octets du port pour le réseau, et la famille de socket en imposant l'utilisation de l'IPv4. Pour réaliser nos tests, nous utiliserons l'adresse de l'hôte local "127.0.0.1" ainsi que le port "8888".

Une fois cette structure paramétrée, elle est ensuite utilisée afin de lier le socket au travers de la fonction `bind()` associée à une vérification d'erreur dans le cas où l'exécution n'aurait pas eu l'effet escompté. Un message est ensuite imprimé dans la console afin d'indiquer que le bind a été effectué avec succès et qu'un abonné est maintenant attendu afin de continuer. L'éditeur entre donc, grâce à la fonction `listen(ListenSocket, SOMAXCONN)` dans la phase d'écoute des connexions¹. De même que précédemment, si cette étape venait à échouer, `WSAGetLastError()` renverrait l'erreur en question dans la console tout en s'assurant de fermer le socket et de libérer toutes les ressources utilisées grâce à `closesocket()` et `WSACleanup()`. L'éditeur peut ensuite accepter ladite connexion entrante au travers de la fonction `accept()` et le nouveau socket créé `SubSocket`. Une vérification de la conformité du socket après cette étape est réalisée pour éviter les erreurs.

Deuxièmement, une fois les deux sockets prêts et liés, on entre ensuite dans la phase de réception et d'envoi de données. Il sera dans un premier temps question de la réception des filtres. Le socket côté éditeur va jouer un rôle de récepteur en restant à l'écoute des données envoyées par l'abonné. L'objectif est ainsi de recevoir les filtres de l'abonné et de les rendre accessibles aux différents fils d'instructions de la plateforme afin d'initier l'acquisition des messages et le filtrage. En effet, nous remarquerons plus tard dans le `main()` dans la section 4.1.2 que le fil d'instructions d'envoi des messages est exécuté en premier et qu'il est le seul en exécution jusqu'à ce que des filtres aient été obtenus. L'envoi des filtres par l'abonné sera détaillé dans la section 4.1.3. Afin de recevoir les filtres, et par la suite, l'envoi de messages, nous utilisons une boucle do-while afin de recevoir/envoyer des messages tant qu'il y'en a à disposition. En effet, l'utilisation de ce type de boucle garantit l'exécution de son contenu au moins une fois avant que la condition de sortie ne soit vérifiée. En effet, notre condition est `iResult > 0`, alors que ce `iResult` est calculé avec la fonction `recv(SubSocket, recvbuf, sizeof(recvbuf) - 1, 0)` qui stocke ainsi ce qui est reçu depuis le socket `SubSocket` dans une mémoire tampon nommée `recvbuf` que nous avons initialisé avec une taille de 14 octets afin de correspondre à la taille des filtres. On note cependant la nécessité de garder un caractère pour le caractère nul en fin de chaîne. On donne alors à la variable `filtre` passée en référence dans l'appel de la fonction `MessageSending()` la valeur du `recvbuf`. Cela permet alors, comme évoqué précédemment, d'avoir accès à la valeur des filtres ainsi fixée dans les autres fils d'instructions, et d'initier l'acquisition et l'analyse des messages.

Troisièmement, il est question d'envoyer vers l'abonné les messages filtrés par le fil d'instructions d'analyse des messages. On utilise cette fois-ci l'autre sens du socket avec la fonction `send()`. On va ainsi utiliser la méthode `dequeue()` de la classe `SafeQueue` sur la fille d'en-

1. SOMAXCONN indique le nombre de connexions que l'éditeur peut accepter, bien que nous n'en utiliserons qu'une ici

voici `s_queue` afin de vérifier s'il y a un message en attente dans cette file et de l'extraire si c'est le cas. Étant donné que les sockets permettent l'échange de mémoires tampons, on transforme la chaîne de caractères stockée dans la file en `char*`, que l'on nomme `sendbuf`, grâce à la fonction `c_str()`. La fonction d'envoi va ainsi prendre en argument `sendbuf` et sa taille incrémentée de un (pour le caractère nul) pour envoyer le message en question. Une boucle `while(true)` est utilisée afin d'initier l'envoi dès qu'un nouveau message est rendu disponible. Une vérification est une nouvelle fois effectuée sur la base d'une comparaison entre `iResultat`, résultat de la fonction `send`, et de `SOCKET_ERROR` afin de savoir si l'un des envois a échoué. Dans le cas où une erreur a lieu, les sockets sont fermés et les ressources libérées.

Exécution des fils d'instructions

Comme évoqué tout au long du fonctionnement de l'éditeur, les différentes tâches sont exécutées au travers de plusieurs fils d'instructions. Le `main()` de notre fichier `.cpp` (dont le code est présenté ci-dessous) permet ainsi de déclarer les variables utilisées et partagées entre les différents fils d'instructions et de les initier à partir des fonctions décrites précédemment. Ainsi, trois fils d'instructions seront utilisés dans le cas de l'acquisition des données auprès d'un éditeur (ici ce sera le STTDS) : `MessageSending()` exécute la fonction `MessageSender()` en premier afin de recevoir les filtres et d'initier l'envoi des messages ; on s'assure ensuite de bien avoir reçu des filtres avant d'initier l'acquisition des messages avec le fil d'instructions `MessageGetterSTDDS` exécutant `MessageGetter()` avec en paramètre le tube correspondant au STDDS ; finalement, le fil d'instructions d'analyse `MessageAnalyzing` exécute la fonction `MessageAnalyzer()` afin d'analyser les messages XML au fur et à mesure de leur acquisition.

```
void main()
{
    // variables

    std::mutex mtx;
    int counter = 0;
    std::string filters = "";
    SafeQueue<std::string> msg_queue;
    SafeQueue<std::string> sending_queue;

    std::thread MessageSending(MessageSender, std::ref(
        sending_queue), std::ref(filters)); // Connection with
        client (Receiving filters + sending selected messages)
```

```

while (filters == "") {}; // wait to receive filter and then
    initiates the message receiving

std::thread MessageGetterSTDDS(MessageGetter, pipe_STDDS, std
    ::ref(msg_queue)); // thread to get the message coming and
    stack the in the queue for STDDS

std::thread MessageAnalyzing(MessageAnalyzer, std::ref(
    msg_queue), std::ref(sending_queue), filters, std::ref(mtx),
    std::ref(counter)); // thread to analyze the xml messages
    extracted from the queue

MessageGetterSTDDS.join();

MessageAnalyzing.join();
MessageSending.join();

signal(SIGINT, signal_handler);
}

```

On remarque finalement sur la dernière ligne de ce code que l'on implémente un appel système afin de répondre aux signaux système de type SIGINT correspondants aux interruptions d'un processus par l'utilisateur (par exemple lorsque l'on arrête l'exécution du code en cours d'acquisition de messages). Ainsi, lors de l'apparition d'un signal SIGINT, la fonction `signal_handler()` est exécutée et va fermer le tube STDDS afin de la rendre disponible lorsque la plateforme sera relancée par la suite.

Extension aux cinq éditeurs de données

L'éditeur étant désormais en place, il est maintenant question d'augmenter son champ d'action afin de réaliser l'acquisition des messages provenant des cinq éditeurs du portail SWIFT. Comme évoqué précédemment, notre volonté a été de proposer une plateforme extensible afin de pouvoir ajouter autant d'éditeurs de données que rendus disponibles par le SWIFT, et de pouvoir également augmenter la puissance d'analyse en cas de besoin. La figure 4.6 est l'adaptation de la figure 4.2 dans le cas de cinq éditeurs de données. On remarque simplement que cette fois-ci ce sont cinq tubes et cinq fils d'instructions reprenant la fonction `MessageGetter` pour chacun des éditeurs qui sont utilisés afin de réaliser l'acquisition des messages. Ces messages sont par la suite stockés dans une unique queue de message afin de

les rendre accessibles à la fonction d'analyse qui les interprète, quelque soit leur origine sur la base des noeuds décrits dans le tableau 4.2. En fonction des performances observées et d'une possible augmentation du flux de données entrantes il sera également possible d'ajouter des fils d'instructions d'analyse en parallèle en utilisant plusieurs `MessageAnalyzer()` récupérant (de façon cohérente) un message dans la queue de messages afin de l'analyser et de décharger cette dernière et éviter qu'elle ne se remplisse plus rapidement qu'elle ne se vide.

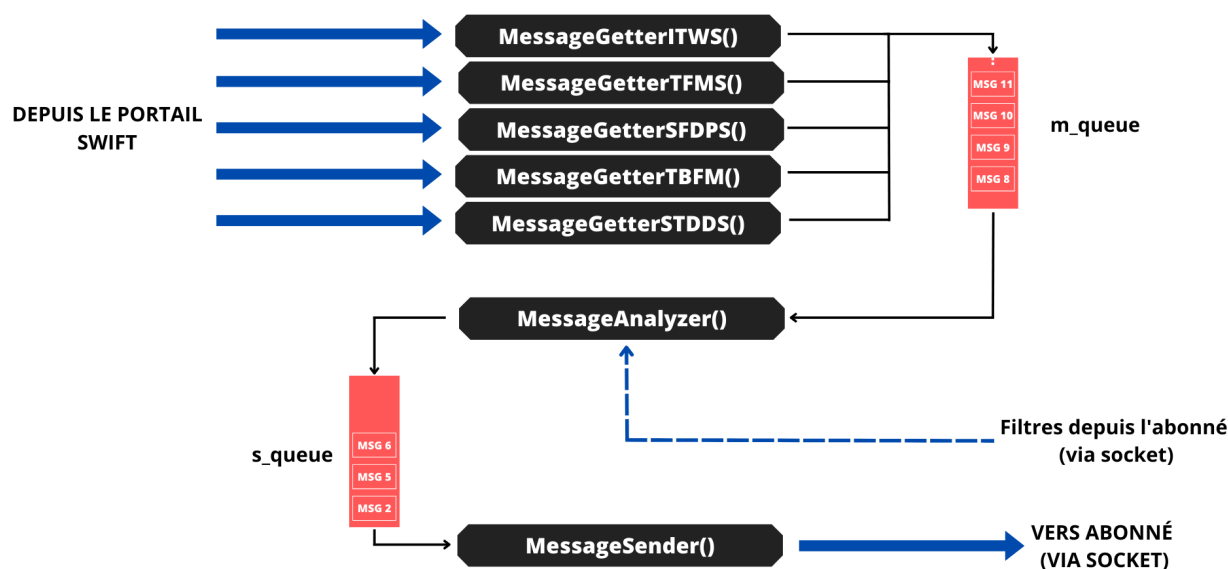


FIGURE 4.6 Extension du bloc éditeur aux cinq éditeurs SWIFT

4.1.3 Bloc Abonné

Nous allons désormais évoquer le second bloc de la figure 4.1, le bloc “Abonné”. Ce dernier représenterait dans un cas concret un avion souhaitant se connecter au SWIM afin de recevoir des messages le concernant, sur la base de filtres sélectionnés. Le rôle du bloc abonné est alors de recevoir les messages et de les stocker en vue de les utiliser par la suite dans de nouvelles interfaces dans les cockpits. Il est important de rappeler que dans le cadre de cette maîtrise, nous nous limitons à rendre disponibles les données du côté de l’abonné (exemple : à des applications EFB ou avioniques). L’utilisation de ces données par l’abonné et le développement des nouvelles interfaces fera l’objet de futurs travaux.

Réception des messages

Comme vu dans la section 4.1.2 sur l’envoi des données par l’éditeur, une connexion à base de sockets est utilisée. D’un point de vue de l’abonné, il est ainsi nécessaire de créer ce socket, de le connecter à l’éditeur grâce à son adresse IP et au port sélectionné, de permettre la sélection et l’envoi des filtres, et la réception des messages filtrés.

Afin de connecter l’abonné à l’éditeur, nous utilisons la fonction `ConnectiontoPublisher()` qui prend en entrée une queue de réception de type `SafeQueue` nommée `r_queue`. On procède de façon similaire à l’éditeur en initiant dans un premier temps la bibliothèque `winsock2` avec la fonction `WSAStartup()` avant de créer le socket `ConnectSocket` avec la fonction `socket(AF_INET, SOCK_STREAM, 0)` en précisant que les adresses seront de la famille IPv4 (avec `AF_INET`) et que le protocole de communication est de type TCP (avec `SOCK_STREAM`). Des tests sont effectués sur ces deux étapes afin d’interrompre l’exécution du code en cas de possibles erreurs. On utilise une nouvelle fois la structure `publisher` afin d’indiquer l’adresse IP de l’éditeur ainsi que le port auquel l’abonné doit se connecter. On procède ensuite à la connexion avec l’éditeur grâce à la fonction `connect()`, avant de vérifier si la connexion est bien établie en comparant `ConnectSocket` avec `INVALID_SOCKET`. Il est important de remarquer qu’il est nécessaire d’avoir dans un premier temps exécuté l’éditeur et que ce dernier soit en écoute avant de pouvoir espérer établir la connexion, sans quoi une erreur sera retournée. Une fois cette étape complétée, l’abonné est connecté à l’éditeur et l’on peut ensuite procéder aux échanges de filtres et de messages.

La première étape de l’échange d’information consiste à l’envoi des filtres. En effet, l’utilisateur est invité à entrer dans un premier temps l’aéroport de départ de son vol, puis l’aéroport d’arrivée, et finalement le numéro de vol. Ces informations sont stockées dans trois variables de type `string` qui sont ensuite mises bout à bout afin de former la variable `filters` (de format `XXXYYYZZZZZZZZ`) qui sera envoyée à l’éditeur. On alloue ensuite une mémoire tampon `sendbuf` de la taille de `filters` incrémentée de un pour le caractère de fin de chaîne, et on copie les caractères de `filters` dans cette même mémoire tampon avec la fonction `memcpy()` avant de placer un zéro sur le dernier caractère. La mémoire tampon des filtres étant prête à être envoyée, on utilise la fonction `send()` pour le faire. Étant donné que l’on ne va définir qu’une seule fois des filtres par utilisation de la plateforme (il sera donc nécessaire de la redémarrer afin d’en entrer de nouveaux), nous procédons à la fermeture de la communication dans le sens de l’envoi pour le socket `ConnectSocket` grâce à la fonction `shutdown(ConnectSocket, SD_SEND)`. Cela va ainsi permettre de signaler à l’éditeur que l’envoi des données relatives aux filtres est terminé et qu’il peut procéder à son tour à l’envoi de ses données.

Une fois les filtres envoyés et le début de l’envoi des messages filtrés par le éditeur, il est désormais question de réceptionner les messages du côté abonné. On utilise une nouvelle fois une boucle `do-while` avec une condition sur `iResult > 0` permettant de savoir si on reçoit toujours des messages. La taille des messages entrants étant inconnue, nous devons prévoir des mémoires tampons de tailles suffisamment grandes afin de pouvoir recueillir l’entièreté d’un même message. Nous avons donc réalisé plusieurs tests afin de connaître la taille maximale des messages et il s’est avéré que la taille maximale rencontrée ne dépassait pas les 370 000 caractères pour un message de SFDPS. Nous fixons donc une mémoire tampon de réception `recvbuf` à une taille de 400 000 caractères afin de contenir tout type de message. La fonction `recv()` est ensuite utilisée afin de lire ce qui transite sur le socket et de le recopier dans la mémoire tampon. On fixe le dernier caractère à zéro et on passe la mémoire tampon sur un string que l’on place ensuite dans la queue de réception `r_queue` grâce à laquelle le message sera rendu disponible pour de futures utilisations.

4.2 Utilisation des données au travers de cas d’usage

Rappelons que l’objectif principal de cette plateforme est de rendre accessible les données provenant du SWIM (au travers du portail SWIFT) aux systèmes avioniques et d’affichage dans les cockpits. Ainsi, grâce à une analyse des différents éditeurs fournis au travers du portail SWIFT et présentée en annexe A, nous avons vu que les données proposés étaient riches et variés et concernaient tant la météo que la gestion des flux aériens. Nous cherchons donc dans cette section à identifier de possibles utilisations de ces données au travers de cas d’utilisations. Après analyse des différents messages transmis par le SWIM, mis en parallèle avec les actions effectuées dans le cockpit, nous avons retenu deux situations dans lesquelles les données apportées permettront une augmentation de la conscience de la situation des pilotes. Pour chacune de ces deux situations, nous procéderons en évoquant la situation actuelle vis-à-vis du problème, puis nous détaillerons la solution apportée par le SWIM en l’associant aux éditeurs du SWIFT concernés, et enfin nous nous projetterons dans de possibles améliorations du SWIM avec des éléments à ajouter pour atteindre une situation idéale. Nos deux cas utilisateurs étudieront ainsi la gestion des événements météorologiques au cours du vol et notamment dans la zone terminale, et la connaissance de l’environnement autour de l’appareil par les pilotes.

4.2.1 Évolutions météorologiques

Cette première étude concerne l’accès aux données météorologiques et l’influence qu’aurait une meilleure connaissance de celles-ci sur les pilotes dans les différentes phases de vol.

En l'état actuel des choses, plusieurs moyens sont mis à la disposition des pilotes afin d'obtenir des informations relatives à la météo [43]. En effet, l'ATC est l'intermédiaire principal avec les pilotes lorsqu'il est question d'informations météorologiques et permet au travers de messages vocaux ou écrits d'informer les pilotes sur les conditions sur sa route. Des fréquences spécifiques à chaque zone sont ainsi attribuées et si un évènement météorologique est détecté dans ces zones, l'ATC en informe les pilotes afin de minimiser les risques ou de prévoir un ajustement de la trajectoire afin de contourner la contrainte. Des messages écrits peuvent également être envoyés à l'avion par le biais de systèmes de communication très hautes fréquences (VHF) avec le système d'adressage et de compte rendu des communications d'aéronefs (ACARS) ou encore via des communications satellitaires avec le système d'acquisition et de relais de données de l'avion au satellite (ASDAR). Ces messages se présentent ainsi sous plusieurs formats tels que le METAR (METeorological Aerodrome Report), rendant compte des conditions actuelles autour d'un aéroport cible et sont émis toutes les heures ou toutes les demi-heures en cas d'évolution rapide, le SIGMET (SIGnificant METeorological information), pour les évènements de forte intensité tels que les orages ou les vents forts, ou encore le PIREP (Pilot REPort) qui sont directement réalisés par des pilotes pour signaler des turbulences, des zones nuageuses denses ou encore des vents forts et avertir les avions qui atteindront ces zones. En plus de ces informations provenant de l'ATC, des équipements à bord de l'appareil permettent aux pilotes d'avoir accès à des informations supplémentaires qui sont ainsi affichées sur les affichages du cockpit. En effet, les avions sont notamment équipés d'un radar Doppler installé dans le nez de chaque appareil et qui permet de détecter les zones orageuses devant lui, mais aussi de différents capteurs de pression permettant de mesurer les changements d'altitude et de pression dans l'atmosphère afin de détecter les zones de turbulences. Ces systèmes sont des systèmes traditionnels en place depuis des dizaines d'années et se sont avérés fiables en respectant les contraintes imposées par les différentes agences de régulation, dont l'OACI. De plus le besoin de certifier tout nouveau système installé à bord des appareils reste une contrainte à l'évolution du secteur, surtout lorsque les systèmes tels que ceux-ci ne sont pas remis en cause. Cependant d'un point de vue des informations partagées, on remarque tout de même un manque d'information sur des évènements ponctuels et soudains qui sont pour le moment uniquement évoqués au travers de messages audio, pouvant rendre leur compréhension compliquée et ne permettant pas leur affichage sur des cartes. De même les pilotes sont à ce jour les seules sources en temps réel pour ce qui est des turbulences ou des zones de givres au travers de leurs rapports (exemple : PIREP).

Le SWIM apparaît ici avec des solutions à certains des problèmes évoqués ci-dessus. En effet, comme évoqué dans la sous-section A, le système météorologique intégré (ITWS) offre

l'accès à des messages concernant un éventail d'évènements météorologiques impactant les zones terminales des aéroports et les TRACON. Ces données mises à jour à des fréquences courtes permettent ainsi d'avoir accès aux conditions en temps réel sur des évènements localisés et soudains tels que les fronts de rafales ou les micro-rafales. La variété des évènements météorologiques recensés et la possibilité d'avoir accès à la fois à des données graphiques et textuelles permettent de proposer plusieurs solutions pour présenter les informations aux pilotes, soit au travers de nouveaux affichages (sur EFB par exemple) ou en superposition à des affichages existants. Prenons ici l'exemple des micro-rafales qui sont des rafales de vent provenant de cellules orageuses et impactant le sol provoquant des vents divergeants au niveau du point d'impact. Ces dernières sont très soudaines, pouvant survenir en quelques minutes, et apparaissent comme un gros risque pour les aéronefs effectuant leurs manoeuvres de départ ou d'arrivée car provoquant de forts vents de face, augmentant donc fortement la portance, suivis de forts vents arrières après le passage du point d'impact, diminuant subitement la portance. Des changements brusques d'altitudes peuvent ainsi survenir à cause de ces vents et impactent les trajectoires verticales des appareils dans des zones où une forte précision est demandée. Avoir une connaissance à l'avance de l'apparition de tels phénomènes permettrait aux pilotes d'anticiper ces possibles changements d'altitudes ou bien dans les cas extrêmes d'annuler la procédure d'approche. En rendant accessibles ce type de données, SWIM permettra aux pilotes d'avoir plus d'information à leur disposition avant d'entamer les étapes les plus critiques d'un vol, et donc de leur permettre d'anticiper de possibles difficultés. La rapidité de l'accessibilité des données permet également d'améliorer la prise de décision en mettant le pilote au fait de toutes ses alternatives le plus tôt possible. Cependant, l'environnement proposé par le SWIM ne peut tout de même pas encore être considéré comme complet et de nouvelles fonctionnalités et types de données pourraient permettre de répondre à d'autres problèmes persistants lorsqu'il est question de gestion de la météo. En effet, les données de l'ITWS sont restreintes aux conditions dans la zone terminale autour d'un aéroport et ne permettent pas d'avoir accès aux évènements en vol de façon automatique (turbulence, givrage...). De plus, en raison de la soudaineté des évènements décrits par les messages SWIM et de la rapidité de l'évolution des ces évènements météorologiques, il est nécessaire de faire particulièrement attention à la fréquence des rapports et à l'expiration des données afin d'avertir le pilote dans le cas où un évènement est en cours, et de l'informer lorsque l'information est périmée ou bien qu'elle a évolué (exemple : direction ou force du vent) afin que celui-ci puisse évaluer son niveau de confiance dans l'information présentée.

Finalement, plusieurs améliorations au SWIM peuvent être imaginées afin d'augmenter son champ d'action. On peut ainsi penser à l'ajout d'un système collaboratif avec la même fonctionnement des PIREP permettant aux pilotes de signaler directement les évènements locali-

sés sur des affichages graphiques afin que les pilotes traversant des épisodes météorologiques les signalent aux suivants. Comme évoqué précédemment, l'ajout d'un service permettant l'acquisition de données relatives à la météo en-route serait aussi intéressant afin d'avoir plus d'informations sur les zones à éviter en vol. Celui-ci pourrait être également le résultat d'un regroupement d'information d'autres systèmes dont le système collaboratif évoqué ci-dessus. Finalement, ces nouveaux ajouts amenés par le SWIM aident le pilote à parfaire sa prise de décision à l'approche de conditions météorologiques défavorables, il pourrait ainsi être judicieux de développer un outil d'aide à la décision. Celui-ci se baserait sur les données recueillies à partir de tous les systèmes disponibles et proposerait au pilote la marche à suivre, que ce soit de continuer ou bien d'annuler la procédure s'il l'estime nécessaire.

4.2.2 Surveillance du trafic autour de l'appareil

Cette seconde étude de cas se concentre sur la connaissance de l'environnement autour de l'avion par le pilote. En effet, les avions évoluent dans des zones spatiales très grandes, mais se retrouvent lors de certaines phases de vol dans des zones contraintes, dans lesquelles ils se doivent de respecter des distances entre les appareils. Avec l'avènement du TBO, ces séparations seront amenées à être réduites afin de permettre l'insertion de plus en plus d'appareil dans le trafic, tout en gardant la sécurité de tous comme un enjeu majeur.

Plusieurs systèmes de suivi du trafic sont mis à disposition des pilotes afin de naviguer au travers des zones à densité de trafic aérien élevée en assurant la sécurité de l'appareil. Parmi ceux-ci, on retrouve le système de surveillance dépendante automatique en mode diffusion (*Automatic dependent surveillance-broadcast* ou *ADS-B*) qui permet la mise en relation des différents appareils en vol, mais également avec le sol au travers d'émetteurs-récepteurs qui partagent, entre autres, la position de l'avion à des intervalles courts. Pour les avions non équipés de l'ADS-B mais possédant un transpondeur, des systèmes tels que le (*Traffic information service – broadcast* ou *TIS-B*) permettent aux autres appareils évoluant dans les environs d'obtenir des informations sur la position, l'altitude et la vitesse. De plus, pour tous les avions équipés d'un transpondeur, un autre système appelé le système d'alerte de trafic et d'évitement des collisions (*Traffic Collision Avoidance System* ou *TCAS*) permet de détecter la présence d'un autre appareil dans une enveloppe autour de l'avion et de fournir aux deux appareils les instructions de vol nécessaires afin de procéder à l'évitement. C'est ainsi au travers des affichages de vol tels que le PFD que les informations transmises par les systèmes décrits ci-dessus sont présentées aux pilotes. Les informations sur la position d'autres appareils peuvent notamment être affichés sur les cartes en utilisant une symbolologie indiquant leur orientation et position relative par rapport à l'appareil, tandis que le TCAS pourra

présenter ses instructions sur l'indicateur d'attitude ou encore sur l'indicateur de vitesse verticale. Une nouvelle fois, malgré la présence de ces systèmes fiables, il peut être intéressant de se questionner sur l'impact qu'aurait l'augmentation du trafic aérien sur l'efficacité de ces derniers, notamment dans les zones terminales où le nombre d'appareil exploserait. De plus, en complément de ces systèmes, la connaissance du trafic pour un pilote passe aussi par les informations fournies oralement par la radio laissant donc une nouvelle fois place à la possibilité de malentendus. On se retrouve finalement dans une situation très centrée sur son appareil avec peu de connaissance sur autres appareils (leur plan de vol, leurs intentions, les instructions qui leur sont fournis et qui pourraient nous impacter..).

Le rôle du SWIM dans l'évolution de cette compréhension du trafic aérien par les pilotes est primordial. En effet, parmi les éditeurs de données décrits dans la section 2.1.5, nous avons pu en identifier quatre que l'on estime qui pourraient permettre d'apporter des informations pertinentes aux pilotes dans cette situation. En effet, les messages TFMS (ref. A) portent sur la surveillance du trafic aérien en permettant le partage de données relatives aux plans de vols des différents avions en vol ainsi que l'état d'achèvement de celui-ci au travers de temps d'arrivées prévus aux points de passages, croisements de routes aériennes ou pour la sortie d'un secteur. Le TBFM (ref A) peut également intervenir en offrant l'accès aux données de vol que sont l'orientation, la vitesse au sol et la vitesse verticale, mais aussi des informations relatives au statut de changement d'altitude ou à l'arrivée à l'altitude de croisière d'un vol. Des informations relatives à la configuration des aéroports et aux taux d'acceptation des avions peuvent aussi être obtenues. De la même façon, le SFDPS (ref A) contribue en proposant des données telles que les temps d'arrivées estimés de chacun des appareils aux aéroports, mais aussi la position, la vitesse et l'altitude attendues par l'ATC. Finalement, le STDDS (ref A) permet de se concentrer sur les deux phases que sont la navigation de surface en donnant les positions et vitesses au sol des autres appareils, et l'accès à la zone terminale en offrant l'accès aux données proposées par le système STARS. La participation de ces quatre éditeurs permet l'accès à une large variété de source de données qui vont pouvoir se compléter les unes les autres, nous permettant d'obtenir une image plus complète du trafic autour d'un appareil. De même, toutes les phases de vol, du taxi jusqu'à l'atterrissage sont couvertes grâce aux quatre éditeurs. On peut également remarquer que des données issues de l'ordinateur de vol (FMS), telles que les RTA, sont partagées à d'autres avions ce qui n'était pas forcément le cas et permet ainsi une augmentation des possibilités en termes de compréhension du trafic environnant. Cependant, afin de pouvoir être pleinement utilisé en pratique, il est nécessaire d'avoir accès à ces données fréquemment. En effet, un suivi de position au sol par exemple nécessiterait des mises à jour au bout de quelques secondes sans quoi il deviendrait inefficace et ne rendrait pas compte de la réalité, devenant même dangereux. De même, des

questionnements sur la latence entre le moment où un avion transmet sa position au sol, le moment où le message est constitué, et le moment où ce dernier est reçu par l'avion cible, sont nécessaires afin de garantir que l'information fournie n'est pas expirée.

Finalement, cette étude nous permis de remarquer que certains points étaient encore à éclaircir avec le SWIM et que des pistes d'améliorations/ajouts sont ouvertes. En effet, on peut penser par exemple à l'augmentation des informations fournies par l'ATC dans les cockpits des autres appareils afin de connaître l'évolution prévue des trajectoires des différents avions environnants. En effet, l'accès à plus de données prévisionnelles, par rapport à des données instantanées nous permettrait de mieux anticiper les potentiels dangers et de prendre les décisions au bon moment. De la même façon que ce que l'on avait proposé dans le premier cas d'utilisateur, les données obtenues à partir des différents éditeurs impliqués pourraient permettre de créer un outil de déconflictualisation automatique, ou même d'améliorer ceux existants (ex : TCAS).

CHAPITRE 5 CARACTÉRISATION ET DIMENSIONNEMENT DE LA PLATEFORME SWIMAIR

Ce chapitre s'intéresse à la caractérisation et au dimensionnement de la plateforme développée afin d'estimer ses capacités en vue des applications futures souhaitées. Une première partie présente les différents tests réalisés et leurs objectifs, puis nous aborderons les résultats obtenus, et finalement nous discuterons des implications de ces résultats.

5.1 Description des tests

5.1.1 Intérêts de la réalisation des tests

L'objectif de ces tests est de comprendre comment la plateforme SWIMAIR peut être utilisée afin de collecter un maximum de données pour permettre la création d'interfaces graphiques. Nous cherchons donc à connaître les conditions optimales d'utilisation afin d'avoir accès au plus grand volume de données possible.

5.1.2 Variables indépendantes

Nous avons deux variables indépendantes : l'heure de la journée et la taille de l'aéroport.

Heure de la journée

Pour ce qui est des conditions de tests, nous partons du constat observé dans [44] sur la répartition des passagers au sein des aéroports au cours d'une journée. En effet, les auteurs de cet article ont étudié l'évolution du flux de passagers au niveau de la sécurité de l'aéroport de Zurich (ZRH) en Suisse, et leurs résultats nous permettent de sélectionner trois périodes de la journée lors desquelles le nombre de passagers était élevé : le matin (entre 6 et 9h), le midi (entre 11 et 13h) et la fin de journée (entre 16 et 20h).

Taille de l'aéroport

Nous avons choisis la taille de l'aéroport comme seconde variable indépendante afin de comprendre son influence sur le nombre de messages reçus. Afin de catégoriser les aéroports, nous utilisons le nombre d'opérations, c'est-à-dire le nombre de départs et d'arrivées d'appareils, comme facteur de classification. Ainsi nous utilisons le terme "grands aéroports" pour désigner les aéroports opérant plus de 500 000 opérations par an, le terme de "moyens aéroports"

pour ceux opérant entre 30 000 et 500 000 opérations par an, et “petits aéroports” pour ceux avec moins de 30 000 opérations par an.

Pour réaliser les tests, nous avons choisi trois aéroports dans chacune de ces catégories en nous basant sur les données fournies par l’outil ATADS¹ de la FAA [45], visibles à la figure 5.1. Cet outil recense les statistiques en termes de nombre d’opérations aériennes dans les différents aéroports américains, et ce sur les trente dernières. Pour les grands aéroports, nous choisissons donc : l’aéroport international Hartsfield–Jackson d’Atlanta (ATL), l’aéroport international de Denver (DEN) et l’aéroport international de Los Angeles (LAX). Pour les aéroports de taille moyenne, nous choisissons : l’aéroport John-Wayne de Santa Ana (SNA), l’aéroport international Bozeman Yellowstone (BZN) et l’aéroport Will Rogers World Airport d’Oklahoma City (OKC). Finalement, pour les aéroports de petite taille, nous choisissons : l’aéroport régional Owensboro–Davies County (OWB), l’aéroport régional de Pocatello (PIH) et l’aéroport Albert J. Ellis Airport (OAJ).

ATADS : Airport Operations : Ranking Report

From 01/2021 To 12/2021 | Facility=ATL, BZN, DEN, LAX, OAJ, OKC, OWB, PIH, SNA
Ranked by : Total Operations

Itinerant						
#	Facility	Air Carrier	Air Taxi	General Aviation	Military	Total
grand	1 ATL	661,551	40,530	5,287	293	707,661
	2 DEN	468,638	117,132	2,955	130	588,855
	3 LAX	464,803	27,623	13,948	395	506,769
moyen	4 SNA	75,408	27,103	111,056	372	213,939
	5 BZN	22,673	11,726	45,148	433	79,980
	6 OKC	42,150	8,956	15,088	28,521	94,715
petit	7 OWB	191	2,807	11,859	1,275	16,132
	8 PIH	236	5,425	13,078	222	18,961
	9 OAJ	5,017	2,373	3,551	1,491	12,432
Total:		1,740,667	243,675	221,970	33,132	2,239,444

FIGURE 5.1 Classement des aéroports choisis en fonction du nombre d’opérations sur l’année 2021 d’après [45]

5.1.3 Variables dépendantes

Nous avons décidé de suivre six grandeurs mesurables pour évaluer la performance de la plateforme :

1. le nombre de messages par minute provenant du portail SWIFT et analysés par la plateforme (en msg/min) ;
2. le nombre de messages reçus du côté abonné après l’application d’un filtre (en msg/-min) ;

1. disponible à : <https://aspm.faa.gov/opsnet/sys/airport.asp>

3. le taux de transfert de données (en kbytes/sec) ;
4. le poids moyen des messages par minute (en kbytes) ;
5. l’empilement des files d’attente (en nombre de messages) ;
6. et le retard induit par la plateforme (en seconde).

5.1.4 Méthode d’acquisition des données

Afin de collecter ces données, nous modifions le code de la plateforme afin de collecter ces différentes grandeurs dans des fichiers .csv et de traiter les données obtenues directement sous Microsoft Excel. Les mesures du nombre de messages reçus, du taux de transfert, et du poids moyen des messages sont réalisées du côté du bloc “Abonné” tandis que celles du nombre de messages analysés et de l’empilement des files sont réalisées au niveau du bloc “Éditeur”.

Pour ce qui est de l’étude des performances de la plateforme en termes de retard engendré, nous mesurons le temps mis pour l’exécution de chacune des tâches implémentées. Pour cela, nous utilisons des marqueurs temporels de la bibliothèque standard `<chrono>` nous permettant de mesurer la timestamp correspondant à l’horloge locale de l’ordinateur. Nous mesurons donc le *Wall-clock time* correspondant au temps réel écoulé entre le début et la fin d’une tâche. La figure 5.2 rend ainsi compte de la position des différents marqueurs temporels utilisés. Ainsi, pour `MessageGetter()`, nous plaçons notre premier marqueur (T1 sur la figure 5.2) lors de l’acquisition de la première ligne d’un message et le second (T2) lorsque le message est envoyé dans la file `m_queue`. Pour `MessageAnalyzer()`, nous plaçons notre premier marqueur temporel (T3) lorsque le message est extrait de la file `m_queue` et le second marqueur (T4) à la fin de la boucle d’analyse XML. Pour la fonction `MessageSender()`, nous plaçons le premier marqueur (T5) lorsque le message est extrait de la file `s_queue` et le second marqueur (T6) après l’exécution de la fonction d’envoi du message à l’abonné. Grâce à ces marqueurs, on peut également mesure les temps passés dans les files `m_queue` (entre T2 et T3) et `s_queue` (entre T4 et T5). Finalement, on mesure le temps de transmission en intégrant la mesure du temps à T6 sous format de string dans le message à envoyer à l’abonné, que l’on extrait alors du côté de l’abonné et que l’on compare à T7 correspondant au marqueur temporel de l’arrivée du message à l’abonné.

Pour la réalisation des tests, nous prenons dans un premier temps chaque aéroport individuellement en appliquant des filtres avec les aéroports de départ et d’arrivée identiques, et sans numéro de vol. Pour chaque aéroport, nous effectuons l’acquisition de données sur les trois plages horaires prévues. Ces tests nous permettent de mesurer le nombre de messages analysés, le nombre de messages reçus, le taux de transfert de données et le poids des messages par minute. Un second test est ensuite réalisé sur les aéroports ayant obtenu les

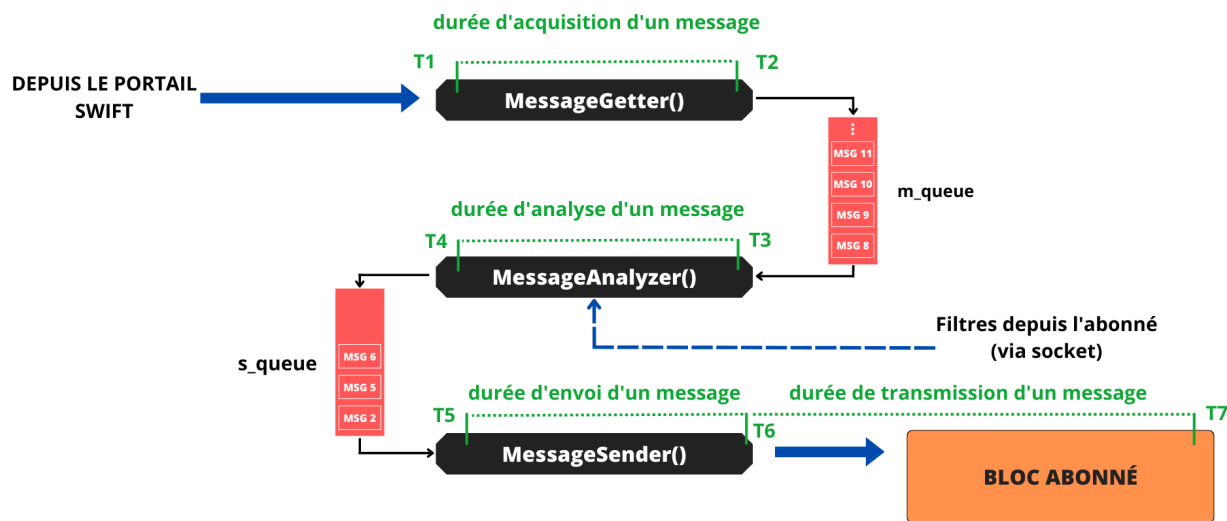


FIGURE 5.2 Marqueurs temporels ajoutés pour la réalisation des mesures de temps de traitement

meilleurs résultats afin de mesurer les retards associés aux différentes fonctions ainsi que les empilements des files d'attente. Finalement, nous réalisons un dernier test afin de nous intéresser à un vol en particulier afin d'estimer le nombre de messages reçus lorsque l'on a comme seul filtre le numéro de vol. Nous avons également réalisé une étude du retard global sur chacun des message transmis à l'abonné en intégrant le marqueur temporel T1 de début d'acquisition d'un message directement en en-tête du message afin de l'envoyer à l'abonné et de le comparer directement à T7 une fois le message reçu.

Tous les tests ont été réalisés sur un ordinateur HP Pavilion - Intel Core i7-10750H CPU @ 2.60GHz - 16GB de RAM, en local.

5.2 Résultats

5.2.1 Influence du moment de la journée sur la quantité de messages fournis par le portail SWIFT

Nous nous sommes d'abord intéressés à l'influence qu'a le moment de la journée sur la quantité de messages envoyés depuis le portail SWIFT à la plateforme SWIMAIR. Nous mesurons ici le nombre de messages analysés, divisé par la durée de l'acquisition de données. La figure 5.3

rend ainsi compte de la moyenne du nombre de messages analysés par la plateforme par minute lors de la réalisation de nos neuf tests sur chacune des tranches horaires.

On remarque ainsi que sur les neuf tests effectués le matin, la moyenne de messages analysés s'élève à 34386 messages par minute, alors qu'elle est légèrement inférieure en fin de journée avec une moyenne de 33841 messages par minute, et nettement en dessous lors du midi avec une moyenne de 22574 messages par minute. Ces résultats peuvent être mis en parallèle avec ceux obtenus dans [44]. En effet, on aurait pu s'attendre à obtenir des résultats similaires montrant des pics d'affluences en matinée et le midi, avec une légère baisse en fin de journée, indiquant alors que les opérations seraient plus concentrées sur ces deux premiers horaires, mais ce n'est pas ce qui est observé avec nos résultats. En effet, les deux plages horaires que sont le matin et la fin de journée se distinguent comme étant celles pendant lesquelles le plus de messages sont partagés par le portail SWIFT. On se retrouve ainsi le midi avec une baisse du nombre de messages par minute avoisinant les 35% par rapport à ce que l'on a mesuré en matinée et de 33% par rapport à ce que l'on avait mesuré en fin de journée. À la vue de ces résultats, nous avons choisi de privilégier les plages horaires du matin et de la fin de journée lors de la réalisation de nos autres tests afin d'avoir accès au plus grand nombre possible de messages.

5.2.2 Influence de la taille de l'aéroport sur la quantité de messages reçus par l'abonné

Le prochain test évalue l'influence de l'application d'un filtre "aéroport" sur la quantité de messages reçus par l'abonné. En effet, nous jouons cette fois-ci sur la taille de l'aéroport en utilisant les neuf aéroports sélectionnés précédemment. Nous appliquons dans le filtre des aéroports de départ et d'arrivée des valeurs similaires (exemple : ATL et ATL) et laissons le champ du numéro de vol avec une valeur ne correspondant à aucun vol : "AAAAAA". La figure 5.4 présente la moyenne, dans chacune des catégories de taille des aéroports, du nombre de messages reçus par l'abonné par minute après l'application du filtre correspondant à l'aéroport en question.

On remarque ainsi que lors de l'application de filtres correspondants aux grands aéroports, on obtient en moyenne 1058 messages par minute, ce qui correspond à plus de 10 la valeur obtenue lors de l'application d'un filtre pour un aéroport de taille moyenne qui est de 103 messages par minute. De plus, on voit que l'on a en moyenne bien moins d'un messages par minute lorsque l'on s'intéresse aux petits aéroports. L'écart-type élevé observé dans le cas de l'application d'un filtre « Grand » aéroport s'explique par la considération des mesures

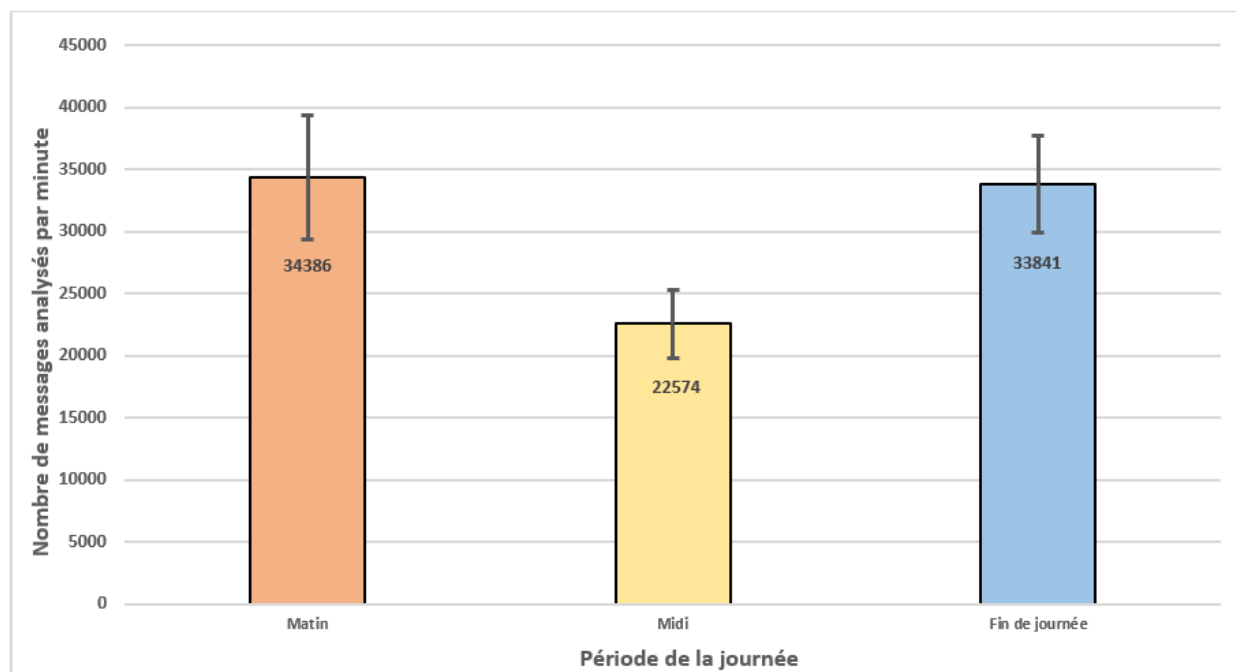


FIGURE 5.3 Nombre moyen de messages analysés par minute par SWIMAIR selon la période de la journée, et écarts-types associés

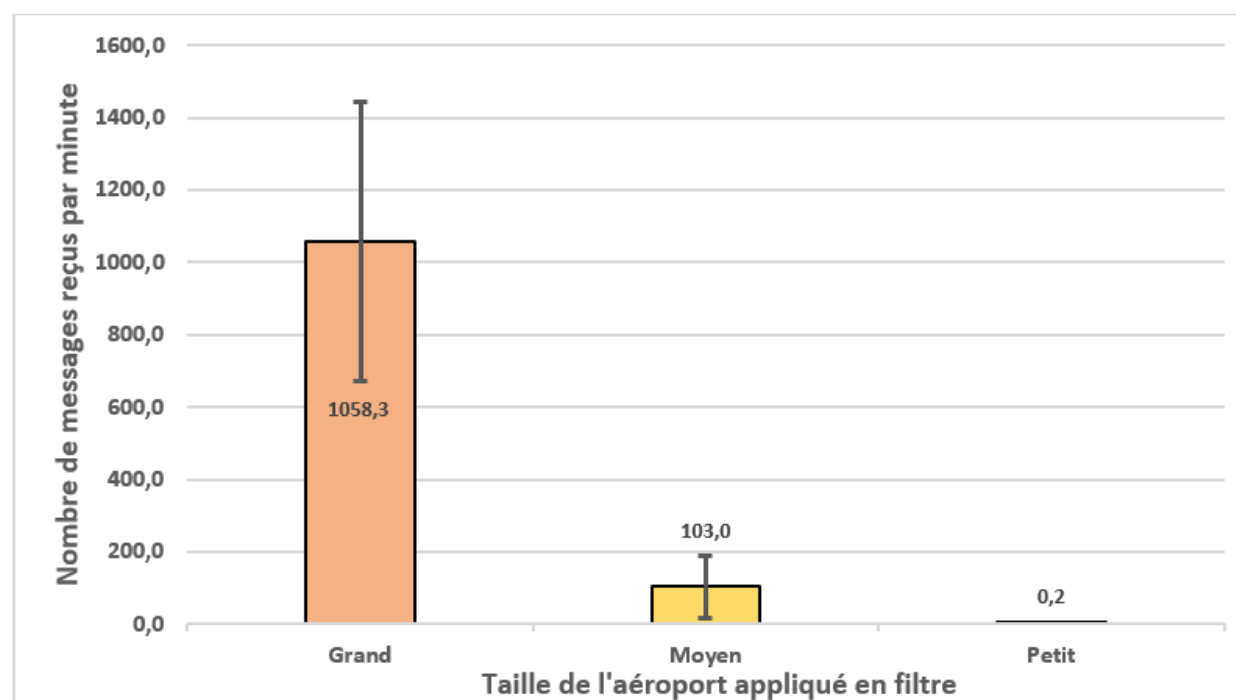


FIGURE 5.4 Nombre moyen de messages reçus par minute par l'abonné après l'application d'un filtre "aéroport" par rapport à la taille de l'aéroport sélectionné, et écarts-types associés

réalisées lors du midi qui, comme observé dans la partie précédente, voient le portail SWIFT fournir moins de messages que le matin ou en fin de journée. On se retrouve ainsi avec moins de messages au niveau de l’abonné sur cette plage horaire (comme observé à la figure 5.5). Les figures 5.5, 5.6 et 5.7 montrent en détail les résultats obtenus pour chacun des aéroports dans chacune des catégories de taille, en ajoutant également la différence pour un même aéroport. Le constat réalisé précédemment sur les aéroports de petite taille est directement visible sur la troisième figure citée, car on remarque moins d’un message reçu par minute, et même une valeur nulle dans trois des tests. Nous choisissons donc d’exclure les petits aéroports de nos prochains tests car les résultats obtenus seraient basés sur un échantillon trop faible de messages.

Lorsque l’on se concentre uniquement sur les aéroports de grande taille, l’observation réalisée dans la sous-partie 5.2.1 sur l’influence des plages horaires sur le nombre de messages rendus disponibles par le portail SWIFT se retrouve dans le nombre de messages reçus par l’abonné après l’application des filtres. En effet, lorsque l’on regarde individuellement par aéroport, les plages horaires du matin et de la fin de journée permettent d’avoir accès à plus de messages que la plage horaire du midi. De plus, sur la figure correspondant aux aéroports de taille moyenne, on remarque que ce constat n’est plus vraiment vérifié et que dans le cas de l’aéroport SNA par exemple, le nombre de messages reçus par minute est supérieur le midi que durant les autres plages horaires. On ne peut cependant pas en tirer de conclusions trop fortes en raison du nombre peu élevé de messages reçus par minute en comparaison avec les aéroports de grande taille.

5.2.3 Répartition des éditeurs SWIM

Les deux sous-parties précédentes nous ont permis de comprendre l’influence qu’avaient le moment de la journée et la taille de l’aéroport sur le nombre de messages rendus disponibles par le portail SWIFT et sur le nombre de messages reçus par l’abonné. Afin d’obtenir un maximum de messages du côté de l’abonné après l’application d’un filtre “aéroport”, nous devons réaliser les tests sur les plages horaires du matin ou de la fin de journée et en utilisant des aéroports de grande taille. Dans la suite, nous utiliserons ces conditions afin d’avoir les résultats les plus représentatifs.

Une fois les messages obtenus du côté de l’abonné, nous nous intéressons à identifier l’éditeur SWIM à son origine. En effet, rappelons que la plateforme SWIMAIR permet la réception et le filtrage de messages provenant de cinq éditeurs de données SWIM que sont : ITWS, TBFM, SFDPS, STDDS et TFMS. Pour réaliser cette mesure, nous observons dans chaque message XML reçu le nom de son premier noeud afin de le relier à son éditeur d’origine.

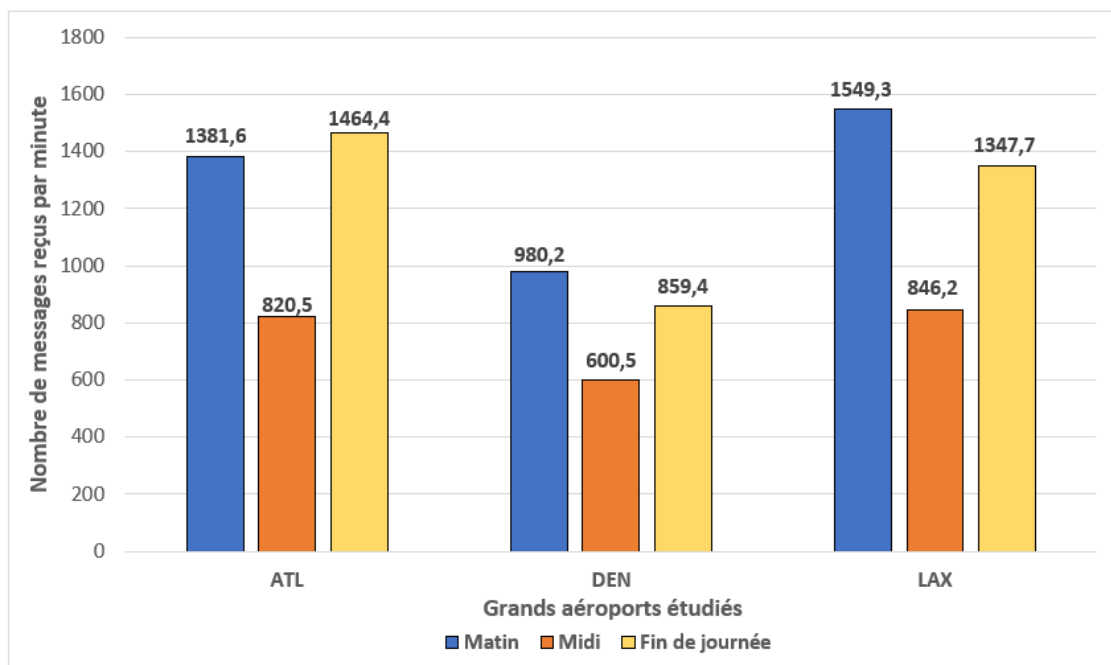


FIGURE 5.5 Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux grands aéroports

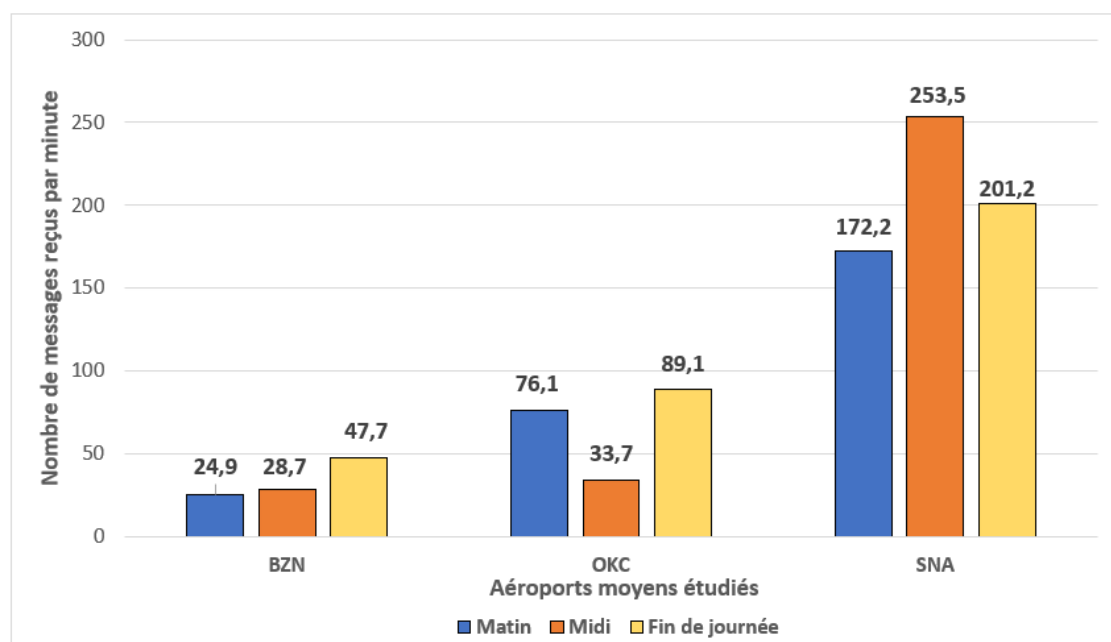


FIGURE 5.6 Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux aéroports de taille moyenne

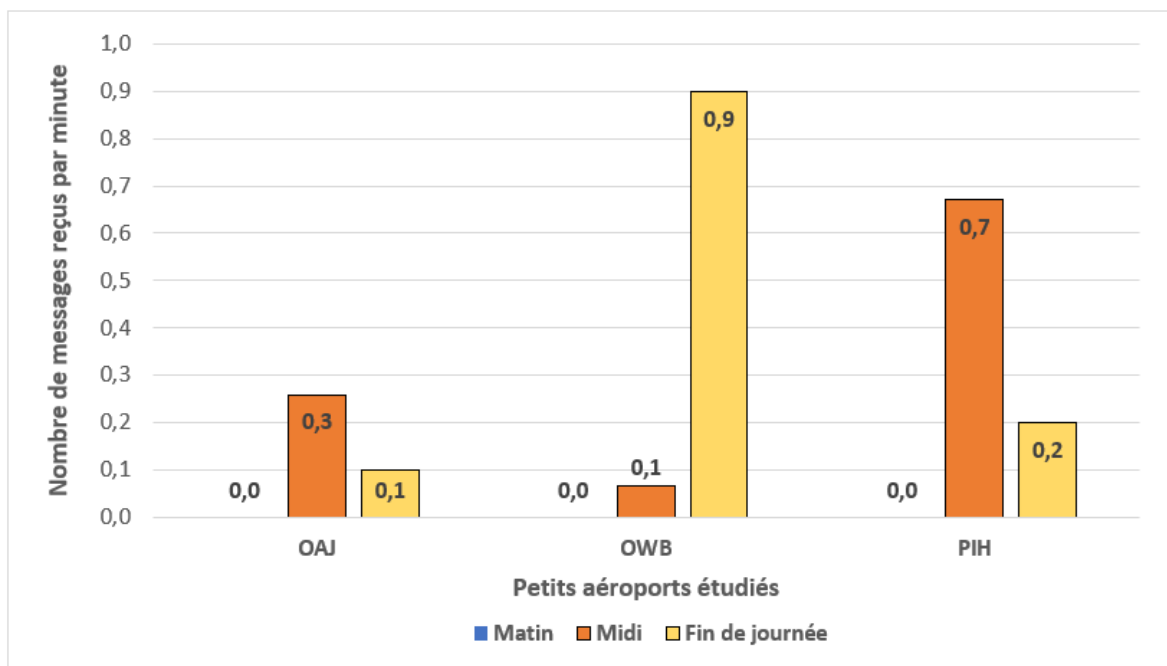


FIGURE 5.7 Mesure du nombre de messages reçus par minute par l'abonné après l'application d'un filtre correspondant aux petits aéroports

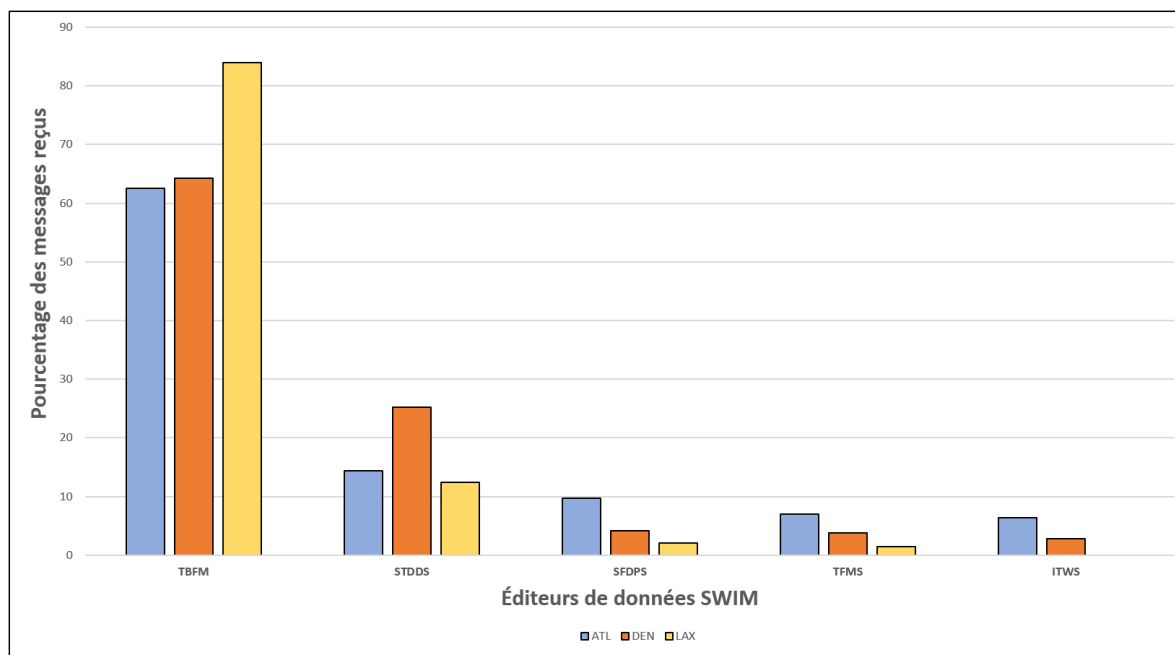


FIGURE 5.8 Répartition des messages reçus par l'abonné au cours de l'acquisition de données le matin dans les grands aéroports

La figure 5.8 rend compte de cette répartition en pourcentage de messages pour les trois grands aéroports sur la plage horaire matinale. On remarque ainsi pour les trois aéroports la même répartition avec, en ordre décroissant, le TBFM, le STDDS, le SFDPS, le TFMS et finalement l'ITWS. Les messages TBFM représentent ainsi la majorité des messages reçus par l'abonné (plus de 60%) contre environ 15% pour le STDDS et moins de 10% pour les autres éditeurs.

Nous avons également réalisé ces tests sur les autres horaires et sur les aéroports de taille moyenne et les résultats sont compilés dans tableau 5.1. On remarque que pour les aéroports de grande taille, les chiffres sont similaires quel que soit le moment de la journée et l'ordre des éditeurs SWIM en termes de pourcentages des messages sont le même. Cependant, bien que les aéroports de taille moyenne nous donnent des résultats quelque peu similaires, on remarque que dans certains cas, les pourcentages entre deux éditeurs sont proches par exemple avec TBFM et STDDS (pour SNA le midi), ou bien encore que l'ordre trouvé pour les grands aéroports est altéré.

TABLEAU 5.1 Répartition des messages reçus par l'abonné au cours de l'acquisition de données, en fonction de l'aéroport étudié et du moment de la journée

	MATIN						MIDI						SOIR				
	TBFM	STDDS	SFDPS	TFMS	ITWS		TBFM	STDDS	SFDPS	TFMS	ITWS		TBFM	STDDS	SFDPS	TFMS	ITWS
ATL	4548 (62,5%)	1048 (14,4%)	705 (9,69%)	511 (7,02%)	465 (6,39%)	ATL	2028 (79,08%)	2028 (11,84%)	722 (4,21%)	588 (3,43%)	246 (1,44%)	ATL	3235 (59,21%)	1666 (30,49%)	299 (5,47%)	162 (2,96%)	102 (1,87%)
DEN	8541 (64,23%)	3346 (25,16%)	552 (4,15%)	495 (3,72%)	364 (2,74%)	DEN	2733 (72,79%)	2733 (17,67%)	559 (3,61%)	806 (5,21%)	111 (0,72%)	DEN	7111 (54,86%)	4786 (36,92%)	416 (3,21%)	540 (4,17%)	110 (0,85%)
LAX	23132 (84,02%)	3420 (12,42%)	577 (2,10%)	401 (1,46%)	0	LAX	2029 (74,54%)	2029 (17,92%)	379 (3,35%)	475 (4,19%)	0	LAX	4032 (69,90%)	1301 (22,56%)	138 (2,39%)	297 (5,15%)	0
BZN	312 (85,48%)	0	26 (7,12%)	27 (7,40%)	0	BZN	268 (78,82%)	1 (0,29%)	46 (13,53%)	25 (7,35%)	0	BZN	280 (70,35%)	45 (11,31%)	73 (18,34%)	0	0
OKC	336 (28,45%)	704 (59,61%)	12 (1,02%)	56 (4,74%)	73 (6,18%)	OKC	151 (20,71%)	495 (67,90%)	27 (3,70%)	19 (2,61%)	37 (5,08%)	OKC	329 (29,88%)	476 (43,23%)	71 (6,45%)	24 (2,18%)	201 (18,26%)
SNA	306 (10,74%)	2488 (87,30%)	34 (1,19%)	22 (0,77%)	0	SNA	1078 (43,40%)	1180 (47,5%)	177 (7,13%)	49 (1,97%)	0	SNA	1360 (41,88%)	1715 (52,82%)	124 (3,82%)	48 (1,48%)	0

5.2.4 Poids moyen des messages par minute

Nous nous intéressons ensuite aux poids des messages ainsi qu'au taux de données transmises entre l'éditeur et l'abonné. La figure 5.9 rend compte de la moyenne des poids des messages reçus par l'abonné lors de l'application d'un filtre d'aéroport de grande taille en fonction de l'éditeur SWIM d'origine. Nous avons utilisé les données des trois aéroports, recueillies sur les trois plages horaires afin de réaliser cette figure. On remarque cette fois-ci que le SFDPS et le TFMS sont les éditeurs dont le poids moyen des messages est le plus élevé avec respectivement une moyenne de 30.2 et 28.5 kilobytes par message. On retrouve ensuite l'ITWS et le STDDS avec respectivement 4.1 et 3.8 kilobytes par message, et finalement en dernière position le TBFM avec seulement 0.5 kilobytes par message. Il est intéressant

de mettre en comparaison les figures 5.9 et 5.8 car nous avons fait le choix de conserver le même ordre d’affichage des éditeurs. En effet, cela nous permet de montrer que bien que certains éditeurs tels que le TBFM et le STDDS représentent à eux deux entre 70 et 85% des messages transmis à l’abonné, le poids individuel de chacun de ces messages est plus de 95% fois inférieur à ceux provenant du SFDPS et du TFMS.

Il en vient ensuite, à la vue de ces résultats, d’étudier le débit de données. La figure 5.10 présente le débit en kilobytes par seconde dans les trois grands aéroports pendant les deux plages horaires que sont le matin et la fin de journée. On note que pour ces trois aéroports, le débit de données, que ce soit le matin ou la fin de journée, avoisine 1 kilobytes par seconde et les dépasse même dans certains cas tel qu’avec ATL le matin qui atteint la valeur la plus élevée avec 1.69 kilobytes par seconde.

5.2.5 Étude de l’influence de la plateforme sur le retard

Afin d’avoir accès à des données en temps réel dans les cockpits d’avions, il est nécessaire que les données soient transmises du sol à l’avion le plus rapidement possible. En effet, certains messages expirant après quelques secondes, il est important qu’ils soient transmis et pris en considération par les pilotes avant cette limite. C’est dans cette optique que nous nous intéressons aux temps de traitement et d’exécution des fonctions de notre plateforme.

Nous avons effectué une première acquisition pour mesurer le temps d’exécution de chaque fonction du côté de l’éditeur. Le tableau 5.2 rend ainsi compte des résultats obtenus. On remarque alors que la fonction `MessageGetter()` met en moyenne 9.4 millisecondes à s’exécuter, la fonction `MessageAnalyser()` en moyenne 0.2 millisecondes et la fonction `MessageSender()` en moyenne moins de 0.1 millisecondes. On peut également noter que ces trois fonctions présentent des maximums respectifs à 1 386 ms, 14 ms et 3 ms. De plus, les temps passés par les messages dans les files `m_queue` et `s_queue` sont faibles, tout comme le temps nécessaire afin de transmettre le message à l’abonné.

Lorsque l’on s’intéresse un peu plus aux résultats obtenus pour la fonction `MessageGetter()`, on remarque que, bien que le maximum soit obtenu à 1 386 ms, cette valeur n’est obtenue qu’une seule fois sur plus de 700 000 messages analysés et que 99.9% des valeurs sont inférieures à 337 ms. Cette valeur nous permet de comprendre que ces cas sont rares bien que toutes fois présents.

Pour expliquer ces pics temporels, nous avons cherché un lien entre le poids du message à acquérir et le temps nécessaire pour l’acquisition. Nous avons donc, dans ce même tableau, associé le maximum de temps de traitement au poids du message concerné. Nous finissons par

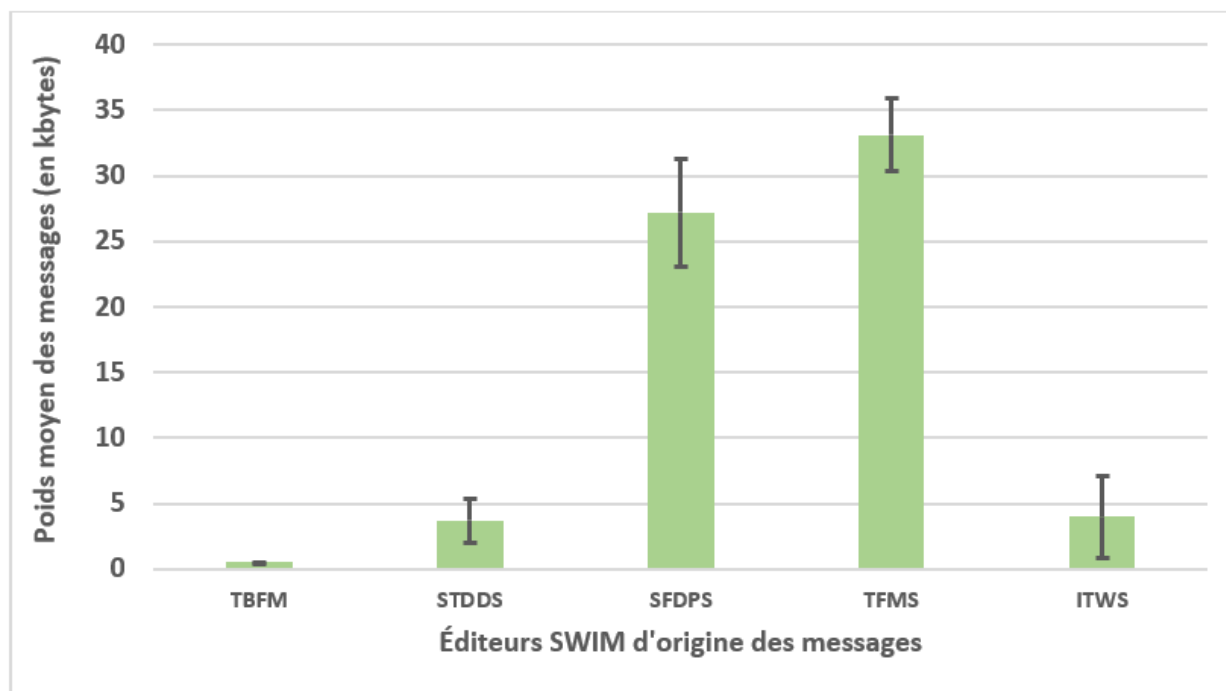


FIGURE 5.9 Moyenne des poids des messages reçus par l'abonné en fonction de l'éditeur SWIM d'origine dans les aéroports de grande taille, et écarts-types associés

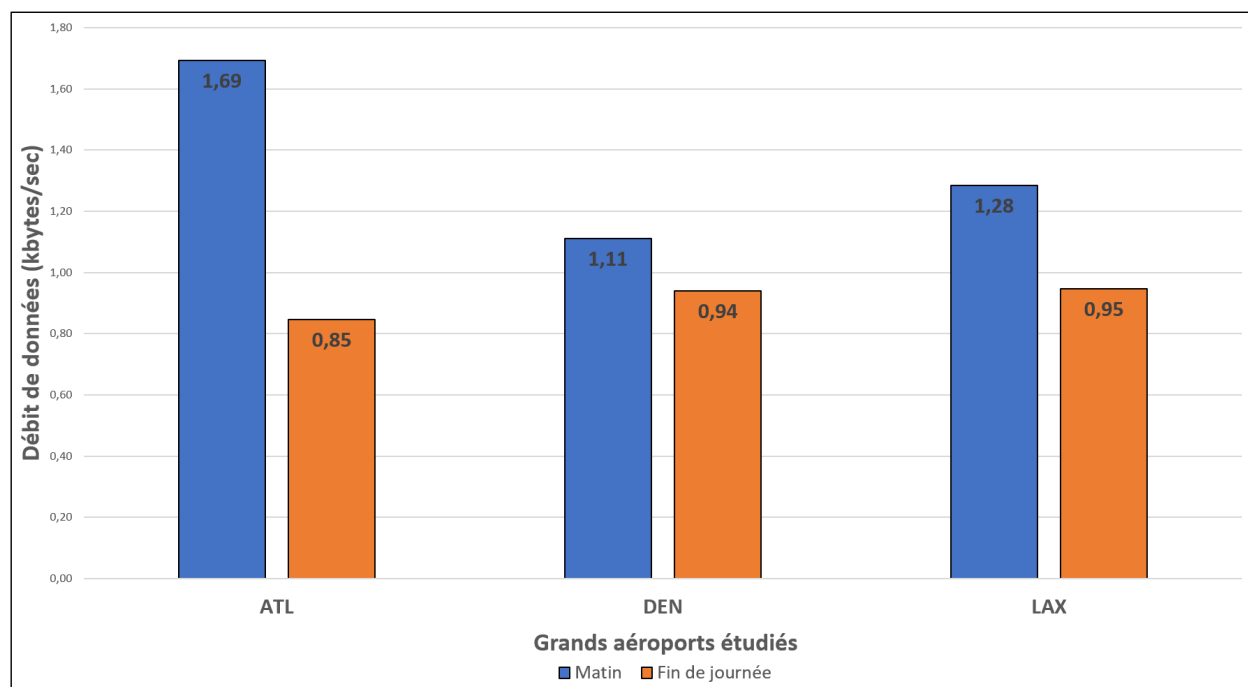


FIGURE 5.10 Mesure du débit de données reçues par l'abonné dans les grands aéroports étudiés en fonction du moment de la journée

remarquer que ce ne sont pas forcément les messages les plus lourds qui présentent le plus long temps de traitement nécessaire. Par exemple, le message correspondant au maximum observé de 1 386 ms pour `MessageGetter()` n'était que de 39 bytes tandis que `MessageAnalyzer()` a mis 14 ms pour un message de 493 bytes.

Nous nous sommes ensuite intéressés au retard global de la plateforme. Nous avons réalisé une seconde mesure d'une durée de 30min05s pour un ensemble de 852 304 messages analysés et 40 626 messages sélectionnés après l'application de filtres (aéroport de départ : ATL et aéroport d'arrivée : LAX). Ainsi, nous obtenons un retard moyen pour le cheminement d'un message du portail SWIFT vers l'abonné de 11.3 millisecondes avec un maximum atteint à 1 060 millisecondes. Les résultats sont présentés au tableau 5.4.

Finalement nous présentons aux tableaux 5.3 et 5.5 les répartitions des poids des messages dans nos deux études. On y remarque une variation importante entre les messages les plus petits et les plus grands atteignant près de 360 000 bytes. L'étude des quartiles et de la valeur à 95% montrent que ces valeurs sont en faibles proportions. Ainsi, l'origine de ces temps de traitement élevés n'est pas déterminée mais nous avons pu éliminer l'influence du poids des messages. Cependant, nous notons que nous utilisons un OS Windows qui n'est pas déterministe et que d'autres tâches peuvent ainsi utiliser d'autres ressources en même temps. Afin d'obtenir des temps plus faibles dans ces cas, nous pourrions chercher à optimiser les fonctions développées ou utiliser un système temps réel déterministe.

5.2.6 Étude de l'empilement des files

Notre prochaine étude nous mène à l'évolution du remplissage des files. En effet, comme nous l'avons évoqué, l'utilisation de files, notamment entre les fonctions d'acquisition et d'analyse, nécessite de porter une attention particulière à l'évolution de leur taille. Dans une logique de transition entre deux fonctions, il est nécessaire d'être capable d'extraire les éléments aussi rapidement qu'ils ont été ajoutés à la file afin d'éviter que la taille de celle-ci explose et provoque de nombreux retards sur l'acheminement des messages. Nous nous intéressons donc ici à suivre l'évolution de la taille de nos deux files que sont *m_queue* et *s_queue*. La figure 5.11 rend ainsi compte de l'évolution de la taille de la file *m_queue* lors de l'acquisition des 198 000 premiers messages. En raison du très grand nombre de messages, la figure ne permet que de remarquer l'existence de pics indiquant que la taille de la file a atteint à plusieurs reprises les 36 messages mais sans jamais aller plus haut.

La figure 5.11 illustre le nombre d'occurrences de chacune des tailles de *m_queue* au cours de l'expérience. En effet, on observe comme attendu que la file de taille nulle est la plus courante, et que dans 65% des cas, la taille de la file est inférieure à 10 éléments. Ces observations

TABLEAU 5.2 Tableau récapitulatif des résultats portant sur la mesure du temps d'exécution des différentes fonctions de la plateforme

Tâche	Moyenne	Écart-Type	Maximum	Valeur à 99.9%	Poids du maximum
MessageGetter()	9.4 ms	35 ms	1 386 ms	337 ms	39 bytes
Passage dans m_queue	0.3 ms	0.8 ms	28 ms	2 ms	480 bytes
MessageAnalyzer()	0.2 ms	0.4 ms	14 ms	4 ms	493 bytes
Passage dans s_queue	< 0.1 ms	0.2 ms	1 ms	1 ms	580 bytes
MessageSender()	< 0.1 ms	0.3 ms	3 ms	1 ms	2680 bytes
Transmission	0.2 ms	1.3 ms	58 ms	1 ms	740 bytes

TABLEAU 5.3 Tableau récapitulatif des résultats portant sur la mesure du poids de messages en lien avec le temps d'exécution des fonctions de la plateforme

Moyenne	Écart-type	Maximum	1 ^{er} Quartile	2 nd Quartile	3 ^{ème} Quartile	95%
2785 bytes	16 415 bytes	354 311 bytes	450 bytes	534 bytes	685 bytes	8282 bytes

TABLEAU 5.4 Tableau récapitulatif des résultats portant sur la mesure du temps total d'exécution

Moyenne	Écart-Type	Maximum	Valeur à 99.9%	Poids du maximum
11.3 ms	31 ms	1 060 ms	237 ms	9331 bytes

TABLEAU 5.5 Tableau récapitulatif des résultats portant sur la mesure du poids de messages en lien avec le temps total d'exécution

Moyenne	Écart-type	Maximum	1 ^{er} Quartile	2 nd Quartile	3 ^{ème} Quartile	95%
4755 bytes	25 844 bytes	366 637 bytes	463 bytes	547 bytes	966 bytes	9856 bytes

nous permettent d'affirmer que la taille de la file *m_queue* demeure sous contrôle lors de l'expérience, et donc que la tâche d'analyse des messages est au moins tout aussi rapide que la tâche d'acquisition.

Au travers de la figure 5.12, nous nous intéressons cette fois à l'évolution de la taille de la file d'envois *s_queue*. Bien que celle-ci soit moins sujette à une explosion de sa taille, nous cherchons tout de même à voir si les messages ne s'entassent pas en attendant d'être transmis à l'abonné. Sur la figure nous pouvons donc voir qu'en grande majorité la taille reste entre 0 et 1, mais atteint ponctuellement des valeurs supérieures avec notamment un pic à une taille de 4. Une nouvelle fois, la file ne semble pas se remplir à l'infini et on a alors le fonctionnement attendu.

5.2.7 Mise en conditions avec le test d'un vol

Finalement, notre dernier test porte sur le suivi d'un vol uniquement grâce à son numéro de vol. Pour ce faire, nous choisissons un vol entre deux aéroports majeurs des États-Unis afin de le suivre, et ce dans la matinée afin d'espérer avoir le plus grand nombre de messages possible. Pour ce test, nous avons suivi le vol 482 opéré par Delta Airlines (identifié DAL482) entre l'aéroport international John F. Kennedy de New York (JFK) et l'aéroport international de Los Angeles (LAX). Nous avons suivi ce vol le 20 mars 2023 à 9h50 (UTC-5) pour une durée de 20min 25sec et avons collecté 563 491 messages du portail SWIFT. Parmi ceux-ci, 152 messages (7.6 messages par minute) ont été sélectionnés après l'application du filtre "numéro de vol" et ont donc été envoyés à l'abonné.

La figure 5.13 rend compte de la répartition des éditeurs d'origine des messages pour ce vol. On remarque le même constat que lors de l'étude de la figure 5.8, avec une prédominance de l'éditeur TBFM. Cependant, on remarque également que certains éditeurs de données n'ont pas transmis de messages. En effet, les messages ITWS et STDDS ne présentent pas de noeuds relatifs au numéro de vol et donc lorsque l'on applique uniquement un filtre "numéro de vol", aucun message ne peut être sélectionné. Cela montre l'importance pour une collecte de données complète de toujours préciser les aéroports dans les filtres afin de bien considérer tous les éditeurs de données SWIFT.

5.3 Discussion

Les différents tests décrits dans la partie précédente ont apporté plusieurs réponses quant au fonctionnement de la plateforme et aux conditions optimales de son utilisation.

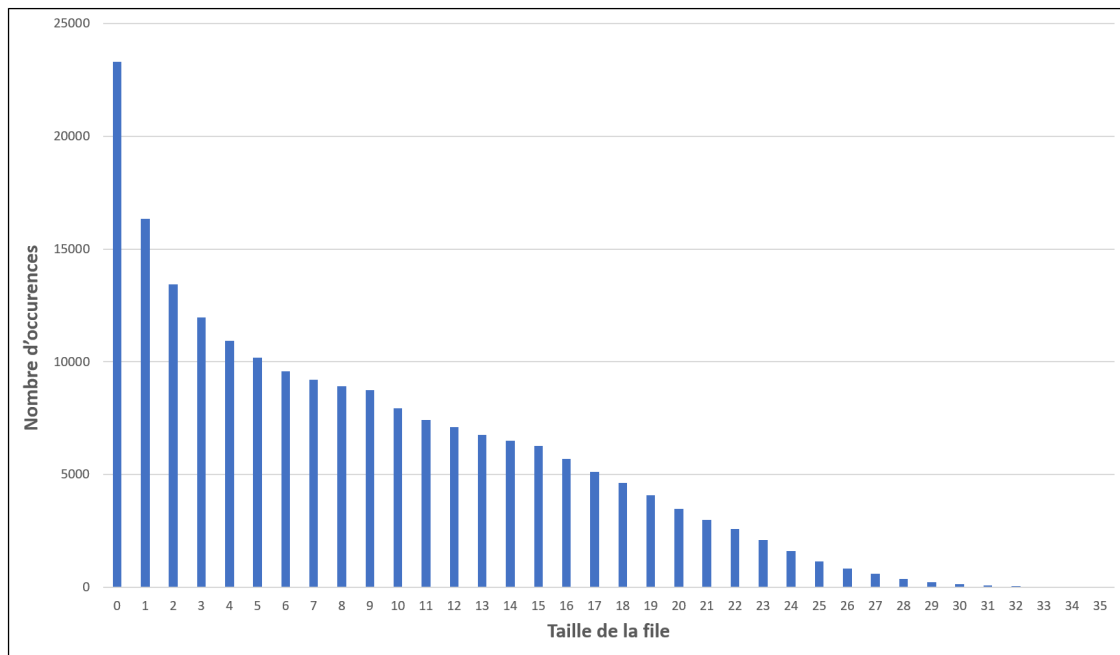


FIGURE 5.11 Nombre d'occurrences de chaque taille lors de l'observation de l'évolution de la taille de "m_queue"

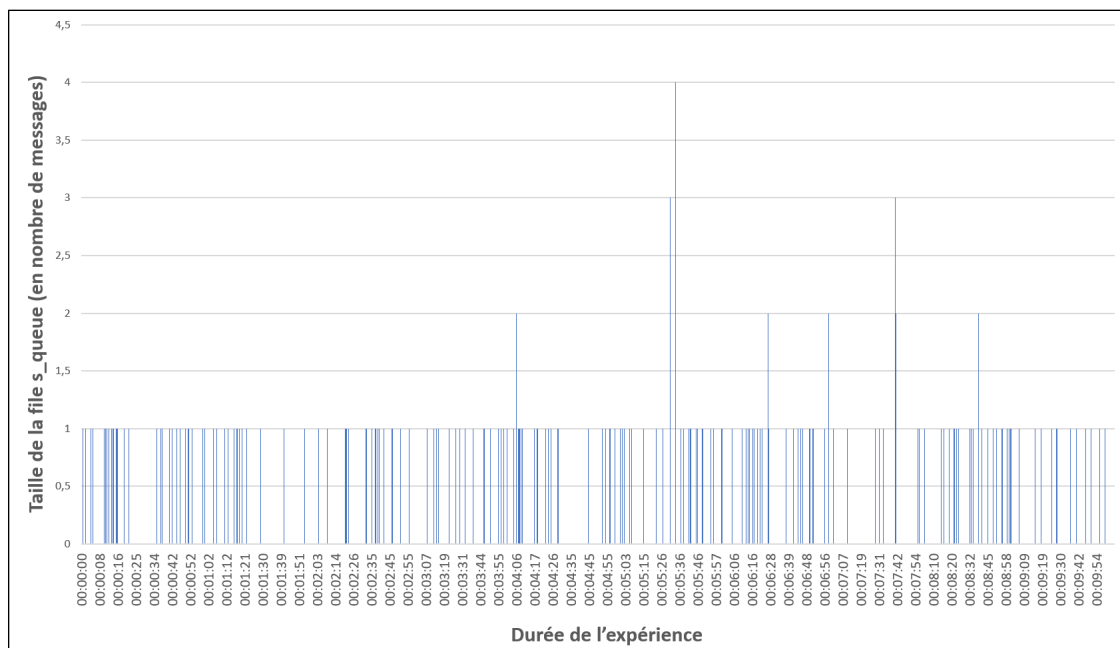


FIGURE 5.12 Mesure de l'évolution de la taille de la file de messages à envoyer au client "s_queue"

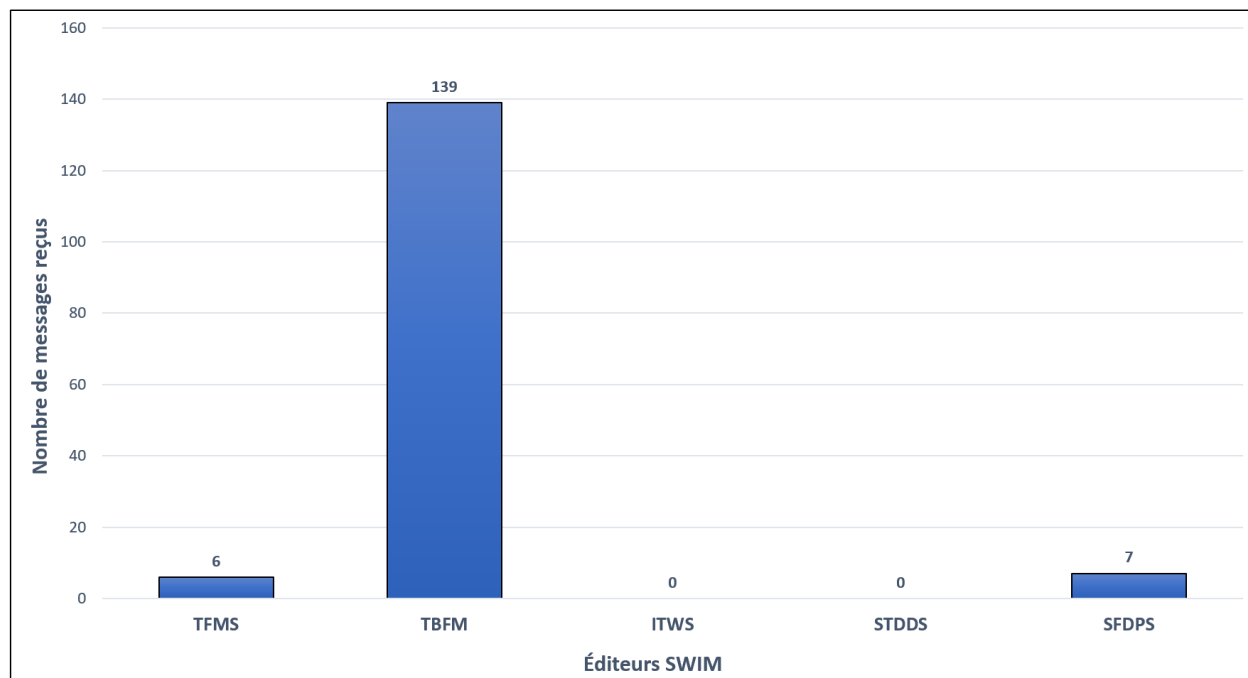


FIGURE 5.13 Répartition de l'origine des messages lors de l'acquisition des données du vol DAL482 entre JFK et LAX

Ainsi, les premiers tests portant sur le nombre de messages transmis à la plateforme par le portail SWIFT nous ont permis de comprendre l'importance du moment de la journée auquel les acquisitions de données sont effectuées. En effet, après avoir défini trois plages horaires pendant lesquelles nous avons fait les tests, il s'est avéré que le matin et la fin de journée sont à privilégier afin d'avoir accès aux plus de messages possibles. Ensuite, nous avons étudié l'influence du choix des filtres sur la quantité de messages obtenus pour l'abonné. Lors de l'utilisation des filtres "aéroports", il a été clairement observé que seul le choix d'un aéroport que nous avons jugé appartenir à la catégorie "grands aéroports" (dans lesquels plus de 500 000 opérations sont effectuées chaque année) permettait d'obtenir un nombre suffisant de messages par minute. Nous noterons que certaines exceptions peuvent être faites pour des aéroports de taille moyenne (comme SNA) en fonction de la quantité de messages nécessaires au bon fonctionnement des interfaces. Finalement, le test portant sur le suivi d'un vol en utilisant uniquement le numéro de vol comme filtre a montré que ce choix de filtrage n'était pas suffisant, car il ne donne pas accès à suffisamment de messages. Ce test nous permet cependant d'estimer combien de messages concernent individuellement chaque vol. Cela supporte aussi l'utilisation des données des aéroports de départ et d'arrivée du vol d'intérêt afin d'améliorer la conscience de la situation des pilotes au travers de la réalisation des interfaces.

Nous nous sommes ensuite intéressés directement aux messages et à la répartition de ces derniers en fonction de leur éditeur de données SWIM d'origine. Il a été montré que près de 80% des messages appartenaient aux deux éditeurs que sont TBFM et STDDS, partageant des données aéronautiques (plans de vol, temps d'arrivées estimées et prévues aux points de passages) et collectées par les tours des aéroports et TRACON (navigation des surface, automatisations terminales, évènements de départ). Bien qu'ils représentent la grande majorité des messages reçus, les poids moyens de ces deux types de messages restent parmi les plus faibles et ne contribuent donc pas à une utilisation notable de la bande passante. Grâce à cette étude menée sur le taux de données transmises par seconde, nous pouvons ainsi essayer d'estimer sur quel type de technologies de télécommunications le SWIM pourrait être implémenté. En effet, plusieurs technologies sont utilisées en aviation et parmi celles-ci on retrouve l'ACARS (évoqué dans 4.2.1, l'ADS-B ou encore les systèmes de communication par satellite SATCOM. Dans le cadre de communications ADS-B, la taille des messages [46] est comprise entre 56 et 112 bits alors que les messages sont envoyés à chaque avion une fois toutes les secondes. À la vue des résultats que nous avons trouvés, les messages SWIM étant plus gros et en plus grande quantité, l'ADS-B ne répond pas à ces contraintes et ne pourra donc pas être utilisé en support du SWIM. Pour ce qui est de l'ACARS, le taux de transmission théorique maximal [47] est de 2.4 kbytes/s et pourrait donc couvrir les besoins du SWIM tels qu'on a pu les identifier. Cependant, il faut garder à l'esprit que ces systèmes possèdent déjà des applications existantes et que l'on viendrait ajouter le SWIM à celles-ci ce qui risquerait de dépasser le taux de transmission indiqué précédemment. Finalement, les systèmes SATCOM sont des systèmes en constante évolution pour s'adapter à de nouvelles utilisations. Comme évoqué dans la revue de littérature, ces derniers permettraient un débit de données compris entre 32 et 650 kbits/s (donc 4 à 81.25 kbytes/s) [20]. Une nouvelle fois, les applications utilisant les systèmes SATCOM sont nombreuses, mais la faible proportion que représenterait le SWIM selon nos estimations (un peu plus de 1 kbytes/s) par rapport au débit disponible évoqué ferait de SATCOM un candidat idéal au support du SWIM.

Finalement, en nous concentrant sur l'étude des performances de la plateforme, nous avons dans un premier temps pu étudier le retard induit par la plateforme sur l'acheminement des messages. Nous avons ensuite pu, grâce à une étude plus globale, remarquer que les messages mettaient en moyenne 11.3 ms à atteindre l'abonné depuis la sortie du portail SWIFT. Il est cependant important de remarquer que, bien que ce soit le cas d'une très faible minorité, certains messages peuvent présenter un retard supérieur. En effet, le maximum obtenu s'est élevé à 1 060 ms lors de l'analyse globale, tandis que l'analyse par fonction a montré que les temps maximums de traitements par les fonction `MessageGetter()`, `MessageAnalyzer()` et `MessageSender()` étaient respectivement de 1 386 ms, 14 ms et 3 ms.

En revenant aux objectifs fixés en début de projet, l'emploi d'une communication utilisant le protocole TCP nous assure la fiabilité de la plateforme en permettant de détecter les messages manquants et grâce à un mécanisme de retransmission automatique de les récupérer du côté de l'abonné. En effet, ce mécanisme a pu être observé lorsque nous avons voulu comparer l'ordre des messages des deux côtés de la plateforme. Les messages au niveau de l'abonné correspondaient bien aux messages de l'éditeur mais pouvaient se trouver à des compteurs différents. Pour ce qui est du traitement de tous les messages provenant du portail SWIFT, nous avons pu observer grâce aux différentes mesures réalisées que la plateforme n'analysait pas un nombre constant de messages à chaque exécution mais qu'elle semblait s'adapter au flux sortant du portail SWIFT. Des analyses plus poussées menées sur les statistiques présentées sur le site du portail SWIFT nous ont permis d'estimer le flux de messages sortant du portail SWIFT lors de certaines de nos mesures. Nous avons ainsi observé l'évolution du maximum de messages émanant de chacun des éditeurs sur les quinze dernières minutes, et ce sur plusieurs plages horaires, et nous avons estimé un flux de données compris entre 25 000 et 43 000 messages par minute. Ainsi, les acquisitions réalisées le matin dans les aéroports d'Atlanta et d'Oklahoma City, ont mesuré l'acquisition de respectivement 45 148 et 29 460 messages par minute. Au-delà de cet aspect non constant du portail SWIFT, ces chiffres montrent également que la plateforme a réussi à s'adapter à un flux sortant allant jusqu'à un peu plus de 45 000 messages par minute qui est légèrement supérieur à l'observation de 43 000 messages par minute que nous avons faite directement sur le site du portail. Nous pouvons donc dire grâce à ces résultats et aux observations réalisées sur l'empilement des files que tous les messages sortant du portail SWIFT sont pris en compte par la plateforme et la traversent bien sans saturation. Finalement, les études menées sur le retard engendré par la plateforme nous ont permis d'estimer le temps de traitement moyen que subit un message lors de son passage par la plateforme. Avec une moyenne globale de 11.3 ms, la plateforme respecte en général la condition que nous nous étions fixés à partir de la norme DO-257A de 1 seconde. Il est cependant nécessaire de remarquer que les études menées par fonctions ont montré que dans de rares cas certains messages nécessitaient un traitement plus long par certaines fonctions comme ce fut notamment le cas avec le maximum obtenu pour la fonction `MessageGetter()` à 1 386 ms. On peut supposer, à la vue de l'écart-type et des autres mesures, que la rareté de ces cas ne devrait pas avoir de grande influence sur la présentation des données étant donné la fréquence d'arrivée des messages. Ainsi, les objectifs fixés en début d'étude sont tous les trois respectés par la plateforme développée. Un travail pourra toujours être possible afin d'améliorer l'efficacité individuelle des fonctions afin de réduire encore plus l'impact de la plateforme sur les messages.

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Ce travail a abouti au développement de la plateforme SWIMAIR qui permet le transfert de messages SWIM du portail SWIFT vers des applications d'interfaçage de données. De plus, deux scénarios d'utilisation des données SWIM ainsi partagées ont été décrits afin d'anticiper et d'orienter la création des interfaces.

Les résultats des tests de caractérisation menés nous permettent d'affirmer qu'afin d'obtenir un nombre maximal de messages du côté de l'avion et donc d'obtenir le plus de données afin d'alimenter les interfaces, les expérimentations doivent être effectuées principalement le matin (entre 6 et 9h) et en fin de journée (entre 16 et 20h). De plus, le choix des filtres est crucial et l'utilisation de grands aéroports (avec plus de 500 000 opérations aériennes par an) est à privilégier. Ces tests nous ont également permis de mieux comprendre le flux de données rendu disponible par le portail SWIFT en identifiant les deux éditeurs de données publiant le plus de messages. Cela va certainement orienter les choix d'interfaces qui seront réalisées avec ces données car elles seront rendues disponibles plus fréquemment. Finalement, une étude des performances de la plateforme au travers de l'étude de l'empilement des files ainsi que du retard temporel provoqué par la plateforme nous a permis d'estimer le temps moyen nécessaire à un message pour traverser la plateforme. Le temps moyen s'élève à 11.3 ms bien que nous ayons remarqué que cette valeur dépasse la seconde dans de rares cas. Ces différents tests nous ont ainsi permis de valider les trois objectifs quantitatifs que nous nous étions fixés.

La plateforme SWIMAIR, rend désormais accessibles les données SWIFT pour les intégrer à des logiciels de création d'interfaces pour concevoir les affichages des cockpits du futur.

6.2 Limitations de la solution proposée

Lors du développement de la plateforme, plusieurs choix ont été réalisés afin de proposer une solution au problème soulevé. Cette partie a pour objectif de discuter des limitations engendrées par ces choix.

Premièrement, le fonctionnement de la plateforme se base sur l'utilisation d'un seul éditeur partageant ses données uniquement à un seul abonné, et ce, sur la base de filtres uniques (une seule combinaison aéroport de départ, aéroport d'arrivée et numéro de vol). Cependant, si

l'on prend le cadre de l'utilisation du SWIM dans des conditions réelles, ce ne sera pas ce fonctionnement qui sera utilisé, mais bien l'interaction entre plusieurs éditeurs (plusieurs stations au sol partageant les informations tout au long du trajet) et plusieurs abonnés (plusieurs avions), utilisant chacun des filtres spécifiques à leur vol. Bien que l'application permette d'ajouter facilement des abonnés en changeant uniquement l'adresse IP et le port visé, il ne sera pas possible dans ces conditions d'imposer des filtres différents pour chacun des abonnés sans apporter des modifications à la fonction d'analyse des messages. Cependant, dans le cadre de l'utilisation souhaitée de la plateforme, c'est-à-dire, dans un cadre de recherche et de travaux sur des interfaces de vol, le fonctionnement actuel de la plateforme est approprié.

Deuxièmement, l'utilisation du portail SWIFT peut également être remise en question. En effet, bien que ce portail soit mis à disposition par la FAA, les données partagées ne représentent pas la totalité des messages qui pourraient être partagés dans le cadre d'une utilisation du SWIM. En effet, il est bien indiqué sur le portail que toutes les données présentées sont pré-approuvées et que certaines données supplémentaires ont été délaissées. De plus, l'indication *"Data obtained via SWIM Cloud Distribution Service (SCDS) is NOT for OPERATIONAL USE"* précisant qu'il ne faut pas utiliser ces données dans un cadre opérationnel laissent penser que celles-ci ne sont pas entièrement fiables. Ces remarques ne remettent pas en question la légitimité de l'utilisation de la plateforme dans un but de recherche, mais permettent d'ores et déjà d'anticiper l'absence de certains services ou messages qui seraient utilisés en pratique dans un cadre opérationnel. Nous pouvons également évoquer la dépendance aux données transmises par le portail SWIFT. En effet, nous n'avons que très peu de contrôle (hormis le choix des éditeurs) sur les données fournies par le portail et notamment sur la quantité de données à disposition à un moment donné. Bien que nous ayons réussi à identifier deux périodes de la journée pendant lesquelles le flux de données sortant du portail était maximal, nous avons également pu observer au sein même de ces horaires que la reproduction de conditions similaires de jour en jour (exemple : même heure et même aéroport) ne donnait pas forcément les mêmes résultats, et ce avec parfois des écarts considérables.

Troisièmement, notre choix de patron de communication s'est porté sur le Éditeur-Abonné à la suite de la lecture des différentes études abordées dans la revue de littérature. Nous avons cependant également vu que d'autres patrons auraient pu être utilisés, comme la requête-réponse ou le pousser-tirer. Nous avons ainsi privilégié une communication ascendante plutôt qu'une communication descendante qui est aussi à considérer dans le cadre du SWIM comme on a pu le constater dans la littérature. Cependant, face au flux à sens unique mis à disposition par le portail SWIFT, il ne nous a pas semblé intéressant d'étudier la communication descendante, mais plutôt de nous focaliser sur l'intégration dans le cockpit des données provenant du SWIM. De futurs travaux sur le sujet pourront éventuellement intégrer ces fonctionnalités.

De plus, le matériel utilisé tant pour le développement que pour la réalisation des tests peut constituer une limitation supplémentaire. En effet, toutes ces tâches ont été effectuées sur un ordinateur personnel (HP Pavilion - Intel Core i7-10750H CPU @ 2.60GHz - 16GB de RAM) et dépendent donc, en termes de vitesse de traitement des tâches, des capacités de ce dernier. Il est donc possible que les performances de l'appareil utilisé influent par exemple sur la vitesse d'acquisition des données auprès du portail SWIFT, augmentant ou diminuant donc les chances d'obtenir des messages vérifiant les filtres et donc de les envoyer à l'abonné. De plus, nous avons utilisé l'adresse IP et un port appartenant à l'hôte local, faisant donc fonctionner la plateforme entièrement en local. Ce choix nous permet de soustraire de la dépendance à la vitesse de la connexion internet utilisée, mais ne réplique pas le contexte d'utilisation futur dans lequel l'abonné sera hébergé par les logiciels d'interfaces graphiques utilisés dans les cockpits.

Finalement, nous pouvons également citer parmi les limitations de notre travail le choix de nous orienter vers le développement d'une solution partant de zéro, en comparaison avec l'utilisation d'un middleware existant. Il est clair qu'en cas de volonté d'une intégration de notre plateforme dans un avion, notre plateforme ferait face à certains problèmes en termes de tests et validation étant donnée la faible robustesse du code développé. Cependant cette preuve de concept nous a permis de montrer la faisabilité d'une telle implémentation et d'étudier la chaîne de données du SWIM, faisant ainsi progresser le projet global. Dans le futur, un travail utilisant un middleware établi permettrait de se focaliser plus sur l'aspect des performances et pourrait être intégré plus aisément dans les avions aux termes d'un processus de certification. D'un point de vue de l'aspect développement logiciel, l'utilisation d'un middleware permet de diminuer le temps de développement de par la réutilisabilité du code et des solutions existantes, l'interopérabilité facilitant l'intégration à de nombreuses applications mais permet également une meilleure gestion des ressources (allocation et la relocalisation de la mémoire, les données, les processus, la réplication).

6.3 Futurs travaux de recherche

Dans le cadre du projet ADAIR, d'autres étudiants seront amenés à travailler sur l'intégration des données transmises par le SWIM aux avions. Le travail réalisé sur cette plateforme sera donc continué afin d'amener les données ainsi sélectionnées à l'avion au travers d'interfaces. Les données rendues disponibles du côté de l'abonné ne sont pas ordonnées et nécessiteront un nouveau traitement (utilisant par exemple pugixml) afin de diriger les messages vers leurs applications respectives. Des études en facteurs humains seront menées afin de comprendre l'impact de l'apport de nouvelles données sur les pilotes et sur leur charge mentale.

Pour ce qui est du travail du côté de la plateforme, bien que cette preuve de concept ait permis de mettre en avant la faisabilité de la transmission des données SWIM, il sera nécessaire dans le futur de considérer une approche employant l'utilisation de middleware établis afin d'en améliorer les performance par rapport à ce qui a été proposé dans ce travail.

RÉFÉRENCES

- [1] Association Internationale du Transport Aérien, “2036 Forecast Reveals Air Passengers Will Nearly Double to 7.8 Billion,” [https://aerolatinnews.com/destacado/2036-forecast-reveals-air-passengers-will-nearly-double-to-7-8-billion/#:~:text=2036%20Forecast%20Reveals%20Air%20Passengers%20Will%20Nearly%20Double%20to%207.8%20Billion,-por%20Soporte%20SocialWeb&text=The%20International%20Air%20Transport%20Association,expected%20to%20fly%20this%20year,2017,accédé le : 2022-11-08](https://aerolatinnews.com/destacado/2036-forecast-reveals-air-passengers-will-nearly-double-to-7-8-billion/#:~:text=2036%20Forecast%20Reveals%20Air%20Passengers%20Will%20Nearly%20Double%20to%207.8%20Billion,-por%20Soporte%20SocialWeb&text=The%20International%20Air%20Transport%20Association,expected%20to%20fly%20this%20year,2017,accédé%20le%202022-11-08).
- [2] Organisation de l’aviation civile internationale, “The Aviation System Block Upgrades - Framework for Global Harmonization,” https://www.icao.int/airnavigation/Documents/ASBU_2016-FINAL.pdf, 2016.
- [3] R. Jamrok, “Aircraft datalink communications for the future,” dans *20th DASC. 20th Digital Avionics Systems Conference*, vol. 2, 2001.
- [4] R. Bone et K. Long, “Pilot use of voice and datalink communications for interval management,” dans *2015 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, 2015.
- [5] Organisation de l’aviation civile internationale, “Global TBO Concept (Version 0.11), Operational Scenarios,” <https://www.icao.int/airnavigation/tbo/Pages/Why-Global-TBO-Concept.aspx>, 2017.
- [6] T. L. Kim Cardosi, “Flight Deck Human Factors Issues for Trajectory Based Operations,” <https://rosap.ntl.bts.gov/view/dot/50930>, aout 2020, *DOT-VNTSC-FAA-20-07*.
- [7] Administration Fédérale de l’Aviation, “Factsheet FF-ICE : Flight and Flow Information for a Collaborative Environment,” <https://www.faa.gov/sites/faa.gov/files/FactSheet-FF-ICE.pdf>, 2022.
- [8] Organisation de l’aviation civile internationale, “Introduction to FF-ICE : a Concept to Support the ATM System of the Future,” <https://www.icao.int/SAM/Documents/2018-AIDC/FFICE.pdf>, avr 2017.
- [9] EUROCONTROL, “ASBU Implementation Monitoring Report,” https://www.icao.int/EURNAT/Documents/ASBU/ICAO-ASBU-Monitoring-Report_2021_EASPG_Released.pdf, jan 2021.
- [10] Organisation de l’aviation civile internationale, “Manual On System Wide Information Management (SWIM) Concept,” <https://www.icao.int/airnavigation/IMP/Documents/>

- SWIM%20Concept%20V2%20Draft%20with%20DISCLAIMER.pdf, 2015, *Doc 10039 AN/511*.
- [11] M. Zolanvari, R. Jain et T. Salman, “Potential data link candidates for civilian unmanned aircraft systems : A survey,” *IEEE Communications Surveys Tutorials*, vol. 22, n^o. 1, p. 292–319, 2020.
 - [12] “FIXM,” <https://fixm.aero/>, accédé le : 2023-04-09.
 - [13] “AIXM,” <https://www.aixm.aero/>, accédé le : 2023-04-09.
 - [14] B. Ren et Y. Jiang, “The Implementation of Aeronautical Information Exchange Model in SWIM,” dans *Advances in Artificial Intelligence and Security*, 2021, p. 188–199.
 - [15] “WXXM,” [https://dbpedia.org/page/WXXM_\(data_model\)](https://dbpedia.org/page/WXXM_(data_model)), accédé le : 2023-04-09.
 - [16] R. Klein *et al.*, “Aircraft access to swim — a collaborative atm pathfinder to support a tbo environment,” dans *2015 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, 2015.
 - [17] X. Lu *et al.*, “Air/ground swim integration to achieve information collaborative environment,” dans *Air Traffic Management and Systems IV*, vol. 731 LNEE, 2021, p. 301 – 314.
 - [18] NASA, “FAA Demonstrate Wireless Communication with Aircraft,” <https://www.nasa.gov/press-release/nasa-faa-demonstrate-wireless-communication-with-aircraft>, 2016, accédé le : 2023-04-10.
 - [19] B. Abraham, “Air/ground system wide information management : Concept demonstration and lessons learned,” *2017 Integrated Communications, Navigation and Surveillance Conference (ICNS)*, p. 2C3–1–2C3–7, 2017.
 - [20] M. Moallemi *et al.*, “Aircraft access to system-wide information management infrastructure,” dans *59th Air Traffic Control Association Conference*, 2014, p. 95 – 100.
 - [21] Administration Fédérale de l’Aviation, “Aircraft Access to SWIM Implementation Guidance Document,” https://portal.ogc.org/files/?artifact_id=55402, jan 2013, *Version 2.0*.
 - [22] M. Carroll *et al.*, “Integrating uncertified information from the electronic flight bag into the aircraft panel : Impacts on pilot response,” *International Journal of Human-Computer Interaction*, vol. 37, n^o. 7, p. 630–641, 2021.
 - [23] M. Yeh, J. M. Jaworski et S. G. Chase, “Pilot perceptions on the integration of electronic flight bag information in new flight deck designs,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 63, p. 91 – 95, 2019.

- [24] A. Polgár, “0MQ Messaging Patterns, Push/Pull,” <https://learning-0mq-with-pyzeromq.readthedocs.io/en/latest/pyzeromq/patterns/pushpull.html>, consulté le 29/06/2023.
- [25] “Simple, Non-Blocking, Push/Pull Example with ZeroMQ,” <https://hightower.space/posts/zeromq-push-pull/>, consulté le 29/06/2023.
- [26] P. Hintjens, “0MQ zguide, Advanced Request-Reply Patterns,” <https://zguide.zeromq.org/docs/chapter3/>, consulté le 29/06/2023.
- [27] D. Noakes, “Request/Response,” <https://netmq.readthedocs.io/en/latest/request-response/>, consulté le 29/06/2023.
- [28] G. Pardo-Castellote, “Omg data-distribution service : architectural overview,” dans *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, p. 200–206.
- [29] X. Wang, M. Yang et P. Han, “Distributed system based on publish-subscribe middleware automated testing method,” *Journal of Physics : Conference Series*, vol. 1453, n°. 1, p. 012105, jan 2020. [En ligne]. Disponible : <https://dx.doi.org/10.1088/1742-6596/1453/1/012105>
- [30] S. Tarkoma, *Publish/Subscribe Systems Design and Principles*. Helsinki : John Wiley Sons, 2012.
- [31] A. S. Mohammad Ashiqur Rahaman, Yves Roudier, “A Publish/Subscribe Model for Secure Content Driven XML Dissemination,” *Journal of Information Assurance and Security (JIAS)*, vol. Volume 6, 2011.
- [32] C. B. Watkins et R. Walter, “Transitioning from federated avionics architectures to integrated modular avionics,” dans *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, 2007, p. 2.A.1–1–2.A.1–10.
- [33] M. García-Valls, J. Domínguez-Poblete et I. E. TOUAHRIA, “Using dds middleware in distributed partitioned systems,” *ACM SIGBED Review*, vol. 14, p. 14–20, 01 2018.
- [34] A. Santos, J. Quevedo et D. Corujo, “Realizing zenoh with programmable dataplanes,” dans *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. Association for Computing Machinery, 2022, p. 125–128.
- [35] G. Baldoni *et al.*, “Zenoh-based dataflow framework for autonomous vehicles,” dans *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2021, p. 555–560.
- [36] K. Landry, J.-F. Boland et G. Bois, “Integration and performances analysis of a data distribution service middleware in avionics,” 09 2015.

- [37] W.-Y. Liang, Y. Yuan et H.-J. Lin, “A performance study on the throughput and latency of zenoh, mqtt, kafka, and dds,” 2023.
- [38] Cloudera, “Apache Kafka Guide,” <https://docs.cloudera.com/documentation/enterprise/6/6.1/PDF/cloudera-kafka.pdf>, consulté le 30/06/2023.
- [39] Z. Kang *et al.*, “A study of publish/subscribe middleware under different iot traffic conditions,” dans *Proceedings of the International Workshop on Middleware and Applications for the Internet of Things*. Association for Computing Machinery, 2021, p. 7–12.
- [40] pugiXML, “pugixml 1.13 quick start guide,” dernière mise à jour en 2022, disponible à : <https://pugixml.org/docs/quickstart.html>, accédé le : 2023-04-03.
- [41] “Winsock Socket 2,” <https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2>, 2021, accédé le : 2022-10-08.
- [42] “Vince’s CSV Parser,” <https://github.com/vincentlaucsb/csv-parser>, 2021, accédé le : 2023-02-17.
- [43] Administration Fédérale de l’Aviation, *Pilot’s Handbook of Aeronautical Knowledge*, 2016, ch. Aviation Weather Services.
- [44] M. Waltert *et al.*, “Ratio-based design hour determination for airport passenger terminal facilities,” *Journal of Air Transport Management*, vol. 96, p. 102125, 2021.
- [45] Administration Fédérale de l’Aviation, “Air Traffic Activity System (ATADS), Airport Operations,” <https://aspm.faa.gov/opsnet/sys/Airport.asp>, accédé le : 2023-03-27.
- [46] A. Abdulaziz *et al.*, “Optimum receiver for decoding automatic dependent surveillance broadcast (ads-b) signals,” *American Journal of Signal Processing*, vol. 5, p. 23–31, 01 2015.
- [47] “VDL Systems : Current Status in the USA and Spectrum Considerations for Potential Future VDL Systems,” <https://www.icao.int/safety/FSMP/MeetingDocs/Forms/AllItems.aspx?RootFolder=%2fsafety%2fFSMP%2fMeetingDocs%2fFSMP%20WG%2dF33%2fIP&FolderCTID=0x012000556AC038F4589F4281B27ABB7E901CAE>, aout 2015, *FSMP-WG F-33/IP-05*.
- [48] Administration Fédérale de l’Aviation, “System-Wide Information Management (SWIM) Terminal Data Distribution System (STDDS) Concept of Operations,” <https://nsrr.faa.gov/node/2085/documents>, oct 2018, *Version 1.1*.
- [49] —, “JAVA Messaging Service Description Document SWIM Terminal Data Distribution System (STDDS) Surface Movement Event Service (SMES),” <https://nsrr.faa.gov/node/2062/documents>, sept 2021, *NAS-JMSDD-4307-003 Rev C*.

- [50] —, “JAVA Messaging Service Description Document SWIM Terminal Data Distribution System (STDDS) Airport Data Service (APDS),” <https://nsrr.faa.gov/node/1386/documents>, sept 2021, *NAS-JMSDD-4307-001 Rev C*.
- [51] —, “JAVA Messaging Service Description Document SWIM Terminal Data Distribution System (STDDS) Tower Departure Event Service (TDES),” <https://nsrr.faa.gov/node/2053/documents>, sept 2021, *NAS-JMSDD-4307-005 Rev C*.
- [52] —, “JAVA Messaging Service Description Document SWIM Terminal Data Distribution System (STDDS) Terminal Automation Information Service (TAIS),” <https://nsrr.faa.gov/node/2085/documents>, sept 2021, *NAS-JMSDD-4307-004 Rev C*.
- [53] John A. Volpe National Transportation Systems Center, “ITWS-SWIM Segment 2 Comprehensive User Guide,” <https://nsrr.faa.gov/node/2055/documents>, aout 2015, *Version 1.5*.
- [54] Administration Fédérale de l’Aviation, “Concept of Operations, Traffic Flow Management System (TFMS) for SWIM Segment 1,” <https://nsrr.faa.gov/node/2051/documents>, nov 2014, *Version 0.1*.
- [55] —, “System Wide Information Management (SWIM) Traffic Flow Management (TFM) Data Flight Data Information Operational Context Document,” 2020.
- [56] —, “System Wide Information Management (SWIM) Traffic Flow Management System (TFMS) - Flow Information Operational Context Document,” 2018.
- [57] —, “System Wide Information Management (SWIM) Traffic Flow Management System (TFMS) - Status Operational Context Document,” 2019.
- [58] —, “JAVA MESSAGING SERVICE DESCRIPTION DOCUMENT Traffic Flow Management Data Service (TFMData),” <https://nsrr.faa.gov/node/9206/documents>, sept 2015, *Version 3.2b*.
- [59] —, “Time Based Flow Management (TBFM) Metering Information Service JAVA Messaging Service Description Document (JMSDD),” <https://nsrr.faa.gov/services/tbfm-mis/documents>, jun 2021, *FAA-TBFM-2021-XXXX*.
- [60] —, “Java Message Service Description Document (JMSDD) System Wide Information Management (SWIM) Flight Data Publication Service (SFDPS),” <https://nsrr.faa.gov/node/2665/documents>, dec 2015, *NAS-JMSDD-4309-001*.
- [61] —, “Concept of Operations For Aeronautical Information Management Modernization,” <https://nsrr.faa.gov/services/fns-nds/documents?page=0%2C1>, sept 2015.

ANNEXE A DESCRIPTIF DES ÉDITEURS DISPONIBLES AU TRAVERS DU PORTAIL SWIFT

STDDS

Le Système de distribution des données du terminal du SWIM, ou STDDS [48], permet l’automatisation de la collecte des données issues des différents équipements aéroportuaires afin de les rendre disponibles au NAS au travers du SWIM. Le terme “terminal” permet de regrouper à la fois les systèmes de contrôle du trafic aérien (ATC) et les systèmes radars de contrôle des approches (TRACON) qui sont à l’origine de ces informations.

Plusieurs types de messages sont alors distribués par le STDDS et sont chacun associés à des systèmes à partir desquels ils puisent leurs informations. Les événements de mouvements à la surface (SMES) [49] se basent à la fois sur les données issues des différents équipements de détection de surface des aéroports (ASDE) ainsi que sur les informations collectées par un autre éditeur, le SFDPS (voir A), pour fournir la localisation de tous les véhicules (avions ou véhicules de services) circulant au sein de l’aéroport, ainsi que des signaux d’alerte en cas de besoin. On retrouve ensuite les messages concernant le service des données aéroportuaires (APDS) [50] regroupant et diffusant les données fournies par les équipements mesurant la distance visuelle de la piste (RVR) dans un but de surveillance et de prise de décision. Les événements de départ de la tour (TDES) [51] concernent quant à eux les données entrées par les contrôleurs aériens en termes de procédures de départ (départ de la porte, navigation de surface, décollage...) et des autorisations émises par la tour de contrôle. Pour ce qui est du service d’information sur l’automatisation des terminaux (TAIS) [52], des informations de vol en direct, des données météorologiques ainsi que des données de surveillance des performances émises par le système standard de remplacement de l’automatisation des terminaux (STARS) sont regroupés afin d’être partagés aux utilisateurs. Finalement, le service de surveillance et de contrôle du système d’infrastructure (ISMC) sert quant à lui à publier des messages d’information sur le statut des systèmes fournissant les autres services du STDDS.

ITWS

Le système météorologique terminal intégré (ITWS) [53] est un service de partage d’informations météorologiques dans les zones terminales. Au travers de données graphiques comme textuelles, il permet de rendre compte des événements météorologiques actuels et des prévisions sur les 60 prochaines minutes. La figure A.1 liste les différents types de produits

proposés par le service ITWS. On y retrouve alors cinq évènements que sont les fronts de rafales, les micro-rafales, les précipitations, les tornades et les vents, en plus d'autres types de données telles que des informations sur la configuration de l'aéroport et les contraintes appliquées aux pistes ou encore des prédictions. Il est important de remarquer que tous ces produits présentent de courtes durées de vie (les plus courtes sont d'environ 30 secondes), nécessitant donc des mises à jour fréquentes afin d'être en phase avec les conditions réelles.

Microburst/Windshear:
9832 "Microburst TRACON Map Product"
9893 "Microburst ATIS Product"
9894 "Wind Shear ATIS Product"

Gust Front:
9833 "Gust Front TRACON Map Product"
9834 "Gust Front ETI Product"

Text Alerts:
9840 "Configured Alerts Product"
9857 "Hazard Text 5nm Product"
9858 "Hazard Text TRACON Product"
9904 "Hazard Text Long Range Product"

Terminal Weather Graphic and Text:
9843 "Terminal Weather Graphics Product"
9844 "Terminal Weather Text Normal Product"
9895 "Terminal Weather Text Special Product"

Precipitation:
9849 "Precipitation 5nm Product"
9850 "Precipitation TRACON Product"
9905 "Precipitation Long Range Product"
9848 "AP Indicated Precipitation Product"

Tornado:
9838 "Tornado Detections Product"
9839 "Tornado Alert Product"

Forecasts:
9901 "Forecast Image Product"
9902 "Forecast Accuracy Product"
9903 "Forecast Contour Product"
9906 "Storm Motion SEP Long Range"
9911 "Storm Motion SEP 5nm Product"
9912 "Storm Motion SEP TRACON Product"

Runway Configuration
9861 "Runway Configuration Product"

Wind Profile
9837 "Wind Profile Product"

Miscellaneous
9847 "AP Status"

FIGURE A.1 Produits fournis par ITWS

Le besoin en information pour le pilote sur ce type d'évènements est primordial, car ces évènements sont soudains et peuvent évoluer très rapidement. Des rapports fréquents et rapides sont donc nécessaires. Des micro-rafales peuvent par exemple apparaître en moins de deux minutes et sont dangereuses pour les appareils entrant dans les zones terminales étant donnée la précision des manœuvres effectuées (décollage, atterrissage).

Les informations fournies sur les micro-rafales permettent la détection et la localisation en

coordonnées de celles-ci, ainsi que leur prévision en se basant sur les radars météorologiques Doppler terminaux (TDWR).

Les fronts de rafales sont surveillés à l'aide de prévisions sur leur évolution sur les 10 à 20 prochaines minutes. Ces données sont préparées pour être utilisées sur des outils graphiques. Le produit d'estimation du temps d'impact permet quant à lui d'avoir une prédiction toutes les 60 secondes du moment à partir duquel le front atteindra l'aéroport.

Les alertes textuelles sont utilisées afin de rendre compte de la configuration de l'aéroport. En effet, des messages sont fréquemment postés pour chaque piste et informent sur la configuration des pistes, ainsi que sur l'orientation et la force des vents les impactant, mais aussi sur l'éventuelle présence d'un évènement météorologique. D'autres éléments textuels viennent s'ajouter aux produits graphiques fournis en informant sur la grêle ou la circulation d'orages violents.

Les produits graphiques résument les informations délivrées par le service ITWS pour un aéroport sous un format graphique localisant les événements et sont rafraîchis toutes les 60 secondes. Les produits textuels viennent eux en trois versions : la version normale rendant compte textuellement des informations délivrées par le service ITWS, mise à jour toutes les 10 minutes ; la version « send » générant des messages d'alertes lors d'évènements dangereux ; et la version « cancel » indiquant la fin de ces mêmes événements.

Quatre produits de précipitation sont ensuite offerts par l'ITWS. Le premier rend compte des précipitations dans une zone de 5 miles nautiques autour du TDWR, avec une fréquence de 1 ou 6 minutes en fonction du mode sélectionné, alors que le second prend pour source de ses données les radars de surveillances (ASR) de l'aéroport avec une fréquence de mise à jour de 30 secondes. Le troisième produit concerne des zones plus éloignées de l'aéroport en question et regroupe les informations de plusieurs radars, il est mis à jour toutes les 150 secondes. Finalement, le dernier produit rend compte des erreurs de propagations détectées par les systèmes et indiquées dans les grilles de précipitations. La figure A.2 présente un exemple de message fourni au travers du SWIFT pour un message ITWS de précipitations.

Les détections de tornades sont également un enjeu dans les zones terminales. ITWS permet de les localiser et d'estimer leur direction et force, mais également de prédire leur formation dans les zones terminales.

Pour ce qui est des produits dits de « prévisions », il en existe 6 variantes. Trois de celles-ci concernent les prévisions tandis que les trois autres suivent les mouvements des tempêtes. On retrouve ainsi dans un premier temps les images des prévisions, présentant jusqu'à sept grilles de données contenant les données de précipitations ainsi que les prévisions basées sur

```

<?xml version="1.0" encoding="UTF-8"?>
<itws_msg>
  <packet_header>
    <packet_header_msgno>14462076</packet_header_msgno>
    <packet_header_product>156</packet_header_product>
    <packet_header_packetsize>32</packet_header_packetsize>
  </packet_header>
  <product_header>
    <product_header_msg_id>
      <product_msg_id>9848</product_msg_id>
      <product_msg_name>AP Indicated Precipitation Product</product_msg_name>
      <product_header_displayable>0</product_header_displayable>
    </product_header>
  </product_header>
  <precip>
    <prcp_TRP_latitude unit="degrees" precision="0.000001">36132842</prcp_TRP_latitude>
    <prcp_xoffset unit="meters">-120000</prcp_xoffset>
    <prcp_dx unit="meters">232</prcp_dx>
    <prcp_dz unit="meters">1</prcp_dz>
    <prcp_nrows>240</prcp_nrows>
    <prcp_nlevels>1</prcp_nlevels>
    <prcp_rotation unit="degrees" precision="0.001">0</prcp_rotation>
    <prcp_no_coverage>15</prcp_no_coverage>
    <prcp_volume_scan_num>7</prcp_volume_scan_num>
    <prcp_nws_levels>
      <prcp_grid_max_x>240</prcp_grid_max_x>
      <prcp_grid_max_y>240</prcp_grid_max_y>
      <prcp_grid_compression_encoding_scheme>Simple Run-Length Encoding</prcp_grid_compression_encoding_scheme>
      <prcp_grid_compressed>15,2268 0,24 15,209 0,38 15,197 0,27 1,10 0,11 15,188 0,31 1,10 0,15 15,180 0,35 1,
15,128 0,114 15,125 0,116 15,122 0,120 15,118 0,124 15,115 0,126 15,113 0,128 15,110 0,132 15,107 0,134 15,
15,77 0,164 15,76 0,164 15,75 0,166 15,73 0,168 15,71 0,170 15,69 0,172 15,67 0,174 15,66 0,174 15,65 0,
0,192 15,47 0,194 15,46 0,194 15,45 0,196 15,44 0,196 15,43 0,198 15,42 0,198 15,41 0,200 15,40 0,200 15,
15,29 0,212 15,28 0,212 15,28 0,212 15,27 0,214 15,26 0,214 15,26 0,214 15,26 0,214 15,25 0,216 15,24 0,
0,220 15,20 0,220 15,20 0,220 15,19 0,222 15,18 0,222 15,18 0,222 15,18 0,222 15,18 0,222 15,18 0,222 15,
15,18 0,222 15,18 0,222 15,18 0,222 15,18 0,222 15,18 0,222 15,19 0,220 15,20 0,220 15,20 0,
0,216 15,24 0,216 15,25 0,214 15,26 0,214 15,26 0,214 15,26 0,214 15,27 0,212 15,28 0,212 15,28 0,212 15,
15,39 0,200 15,40 0,200 15,41 0,198 15,42 0,198 15,43 0,196 15,44 0,196 15,45 0,194 15,46 0,194 15,47 0,
0,176 15,65 0,174 15,66 0,174 15,67 0,172 15,69 0,170 15,71 0,168 15,73 0,166 15,75 0,164 15,76 0,164 15,
15,105 0,134 15,107 0,132 15,110 0,128 15,113 0,126 15,115 0,124 15,118 0,120 15,122 0,116 15,125 0,
0,56 15,188 0,48 15,197 0,38 15,209 0,24 15,2268</prcp_grid_compressed>
      <prcp_grid_max_precip_level>1</prcp_grid_max_precip_level>
    </prcp_nws_levels>
  </precip>
</itws_msg>

```

FIGURE A.2 Exemple de message de précipitations fourni par ITWS

celles-ci. Ces messages sont publiés toutes les 5 minutes et sont accompagnés de messages sur la précision et les contours des prévisions. Les messages de précision évaluent la qualité des prévisions alors que les contours permettent de délimiter les zones de ces prévisions. Pour ce qui est des mouvements de tempêtes, ces trois types de messages concernent, comme pour les précipitations, la zone des 5 miles nautiques, les données recueillies par le TRACON et la zone longue distance de l'aéroport. Ces informations permettent de connaître la position et la direction des cellules orageuses afin d'en estimer leurs positions futures sur la base de vecteurs de déplacements.

Le produit de configuration des pistes est mis à jour et envoyé dès qu'un changement est effectué à la configuration de l'aéroport. Ce message indique l'heure à laquelle cette mise à jour a été effectuée ainsi que l'identifiant de la nouvelle configuration.

Les profils de vents sont envoyés toutes les 5 minutes et indiquent l'orientation du vent ainsi que sa vitesse en fonction de l'altitude.

Finalement, le dernier produit nommé « divers » permet d'indiquer la qualité de l'information contenue dans les messages de précipitations et est envoyé aux mêmes moments que ces derniers.

TFMS

Le système de gestion des flux de trafic (TFMS) [54] est un éditeur permettant le partage d'information sur la gestion et la surveillance du trafic aérien dans l'espace aérien national (NAS). Il s'intéresse notamment aux plans de vol et amendements qui leur sont apportés ainsi qu'aux messages de départ et d'arrivées. L'utilisation de ce service est restreinte aux vols aux instruments (IFR). Plusieurs types de messages sont fournis par cet éditeur : les messages de "Flight Data", de "Flow Information", de "Terminal Flight Data" et de "TFMS Status".

Les données de vols [55] (messages Flight Data - "fltOutput") sont regroupées à partir de plusieurs sources du NAS qui fournissaient les systèmes précédemment en place (Aircraft Situation Display to Industry (ASDI), Traffic Flow Management Data to Government (TFMDG), Time Based Flow Management (TBFM), et Flight Table Manager (FTM)) et en ajoutent sur la gestion par le TFMS. Celles-ci contiennent la programmation du plan de vol ainsi que les modifications qui lui ont été faites, les temps de départ et d'arrivée et des informations sur la position de l'appareil, notamment lors de changements de zones (« boundary crossing »). Les séquences de points de passages ou de repères prévus par le plan de vol ainsi que le temps auquel ils vont être traversés sont indiquées dans ces messages.

Les informations sur les flux [56] (messages Flow Information - "fiOutput") rendent compte des créations, des mises à jour et des annulations des initiatives de gestion du trafic (TMI) telles que les programmes de retardements au sol (GDP) visant à contrôler les flux d'arrivées en jouant sur les heures de départ, les programmes de débit de l'espace aérien (AFP) avec le même objectif mais en jouant sur les zones de flux restreints (FCA), ou encore les reroutages pour réorienter les flux dans des zones moins occupées. Ces TMI permettent par exemple de mettre à jour la configuration d'un aéroport, ou la disponibilité des routes de départ.

Les données terminales de vol (messages Terminal Flight Data - "tfdOutput") proviennent des compagnies aériennes partenaires et d'autres systèmes de la FAA. Elles sont utilisées à des fins de planifications dans les zones terminales. Elles peuvent être également utilisées dans des outils de gestion des flux au sol.

Finalement, les messages de statut TFMS [57] (messages TFMS Status - "tfmsStatusOutput") permettent de donner le statuts des messages entrants et sortants et rendent compte de l'état des services TFMS.

La formation des messages XML et la signification de chacun des champs est décrit dans le document de description du service de messages Java (JMSDD) : [58]

TBFM

La gestion des flux basée sur le temps (TBFM) [59] est un service offert par le SWIM permettant l'optimisation des flux aériens en fournissant aux différents acteurs les estimations des temps d'arrivées à chaque point de passage et à l'arrivée des aéronefs, ainsi que l'heure à laquelle cela était prévu par le plan de vol. Cet outil permettra le séquençage des aéronefs dans les zones terminales des aéroports. Ce service est également sous-divisé en plusieurs types de messages. On retrouve ainsi les messages <air> correspondants à la catégorie d'informations sur les aéronefs, les messages <con> sur la catégorie d'informations sur la configuration, les messages <oth> sur les catégories d'informations sur les autres configurations et les messages <adp> sur les catégories d'informations d'adaptation.

Les messages <air> rendent compte d'informations utiles sur les vols dont notamment sur les plans de vol, les heures d'arrivées estimées et prévues et les différents éléments de référence définis (point de passages, repères, seuils de la piste...).

Les messages <con> informant sur les configurations des différents systèmes impactant le vol, dont notamment les aéroports de départ et d'arrivée. On retrouve également des informations à propos des taux d'acceptation dans les aéroports.

Les messages <oth> et <adp> concernent respectivement les statuts des systèmes de mesures et d'interface, et les noms des différents systèmes tels que les TRACON et points de repère utilisés.

SFDPS

Le Service de publication des données de vol SWIM (SFDPS) est un service permettant la distribution des données en temps réel provenant du système de modernisation de l'automatisation en route (ERAM) qui se trouve au coeur du projet NextGen. Cet éditeur permet la publication de quatre types de données [60] : Publication des données de vol en route (ERFDP), Publication de données sur l'espace aérien en route (ERADP), Publication de données opérationnelles en route (ERODP), Publication de messages généraux en route (ERGMP). Cependant chacun de ces quatre types de données est lui-même divisé en plusieurs types de messages en fonction de l'information transmise par chacun. Ainsi, les tableaux A.1, A.2, A.3 et A.4 référencent les différents types de messages que chacun d'entre eux peut produire. Chacun des types cités dans ces tableaux présente des formats de messages différents afin de transmettre le type d'information souhaité. Leur encodage est décrit dans le document de description du service de messages Java (JMSDD) : [60].

TABLEAU A.1 Types de messages du système de publication des données de vol en route (ERDFP)

Nom du message	Code du message
Information sur le plan de vol	FH
Information sur les modifications de vol	AH
Information sur la route convertie	HX
Information sur l'annulation	CL
Information sur le départ	DH
Information sur la modification de l'identification de l'aéronef	IH
Information sur la mise en attente	HH
Information sur le rapport d'avancement	PH
Information sur l'arrivée du vol	HV
Information sur la mise à jour du plan de vol	HU
Information sur l'heure de départ prévue	ET
Information sur la mise à jour de la position	HP
Information sur le plan de vol provisoire	NP
Modification provisoire de l'identification de l'aéronef	NI
Suppression du plan de vol provisoire	NL
Informations relatives à la modification du plan de vol provisoire	NU
Informations sur la piste	TH
Informations sur la trajectoire de largage	RH
Information sur l'altitude provisoire	LH
Information sur la piste de contrôle de flux	HZ
Réaffectation du code de la balise	BA
Code de balise restreint	RE
Information sur la quatrième ligne du bloc de données complet	HF
Information sur le point de sortie	HT
Information sur les points de sortie entrants	PT
État du transfert	OH
Message de reconstitution du plan de vol	DBRTFPI

AIM FNS

Le service de gestion de l'information aéronautique et système fédéral des NOTAM [61] permet le partage des données aéronautiques entre les différents acteurs du trafic aérien. Parmi ces données, on retrouve celles des aéroports, d'aide à la navigation (NAVAID), de procédures de vol, et des avis aux aviateurs (NOTAM).

Ce service n'est pour l'instant pas utilisable au travers de la plateforme SWIFT étant donné qu'il ne publie pas de messages.

TABLEAU A.2 Types de messages du système de publication de données sur l'espace aérien en route (ERADP)

Nom du message	Code du message
Statut d'assignation de secteur	SH
Statut de la route	HR
Calage altimétrique	HA
Reconstitution du statut de la route adaptée	DBRTRI
Reconstitution de l'état de l'altimètre	DBRTAI
Reconstitution de l'assignation de secteur	DBRTSI

TABLEAU A.3 Types de messages du système de publication de données opérationnelles en route (ERODP)

Nom du message	Code du message
Ajustement du comptage du trafic	AK
Ajustement du comptage de l'approche aux instruments	AC
Signalisation d'entrée et de sortie	SY
Utilisation des codes de balise	UB
Utilisation du code balise géographique	UG

TABLEAU A.4 Types de messages du système de publication de messages généraux en route (ERGMP)

Nom du message	Code du message
Information générale	GH
Information sur l'état de l'altitude provisoire	HE
Information d'état d'attente	HO
Information d'état ERAM	HS
Information sur l'échec de la transmission	UI

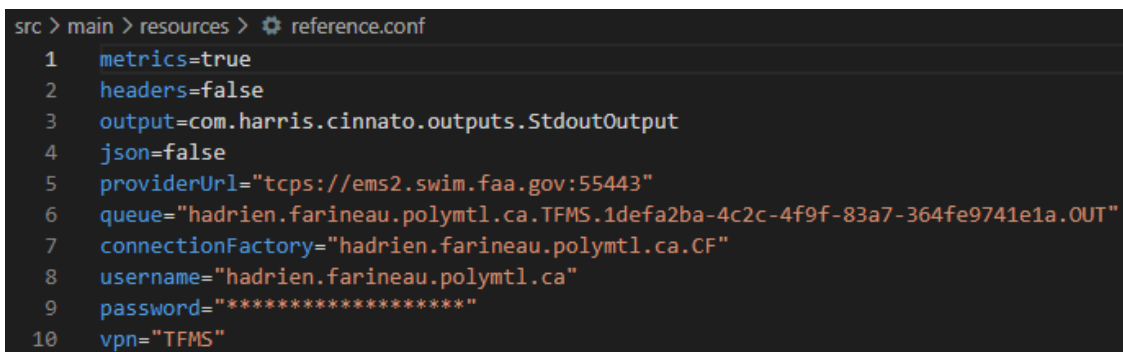
Obtention des données

L'accès aux services proposés par la plateforme SWIFT s'effectue par le biais d'inscriptions à chacun de ces services. En effet, après avoir obtenu les différentes autorisations de la part de la FAA, l'accès aux services sera garanti et un kit de démarrage sera fourni afin d'initier la réception de données.

Ces kits donnent ainsi accès à une application Java permettant de tester les services. Il est

possible de personnaliser l'application au travers de son code afin de choisir le type de sortie de données souhaité. Ceci est réalisé au travers du fichier « reference.conf », dans lequel on entre les paramètres de connexion à chacun des services. En effet, quatre différents types de sorties sont disponibles : `FileOutput`, `MessageFileOutput`, `NoopOutput` et `StdoutOutput`.

`NoopOutput` est l'option de sortie par défaut et ne sort tout simplement pas de message. `MessageFileOutput` permet de récupérer individuellement chacun des messages émis par le service en question et le stocke dans le fichier `/log/` accompagnant l'application. La nomenclature des messages est effectuée en fonction du nombre de millisecondes au moment du message (exemple : `./log/1675373283909`). `FileOutput` fonctionne de façon similaire mais va enregistrer tous les messages entrants au fur et à mesure dans un fichier texte « message » contenu dans `/log/`. Finalement, `StdoutOutput` permet d'écrire les messages émis par le service directement dans la sortie standard (et donc visible sur la console).



```
src > main > resources > ⚙ reference.conf
1  metrics=true
2  headers=false
3  output=com.harris.cinnato.outputs.StdoutOutput
4  json=false
5  providerUrl="tcps://ems2.swim.faa.gov:55443"
6  queue="hadrien.farineau.polymtl.ca.TFMS.1defa2ba-4c2c-4f9f-83a7-364fe9741e1a.OUT"
7  connectionFactory="hadrien.farineau.polymtl.ca.CF"
8  username="hadrien.farineau.polymtl.ca"
9  password="*****"
10 vpn="TFMS"
```

FIGURE A.3 Exemple de configuration pour le TFMS pour une sortie `StdoutOutput`