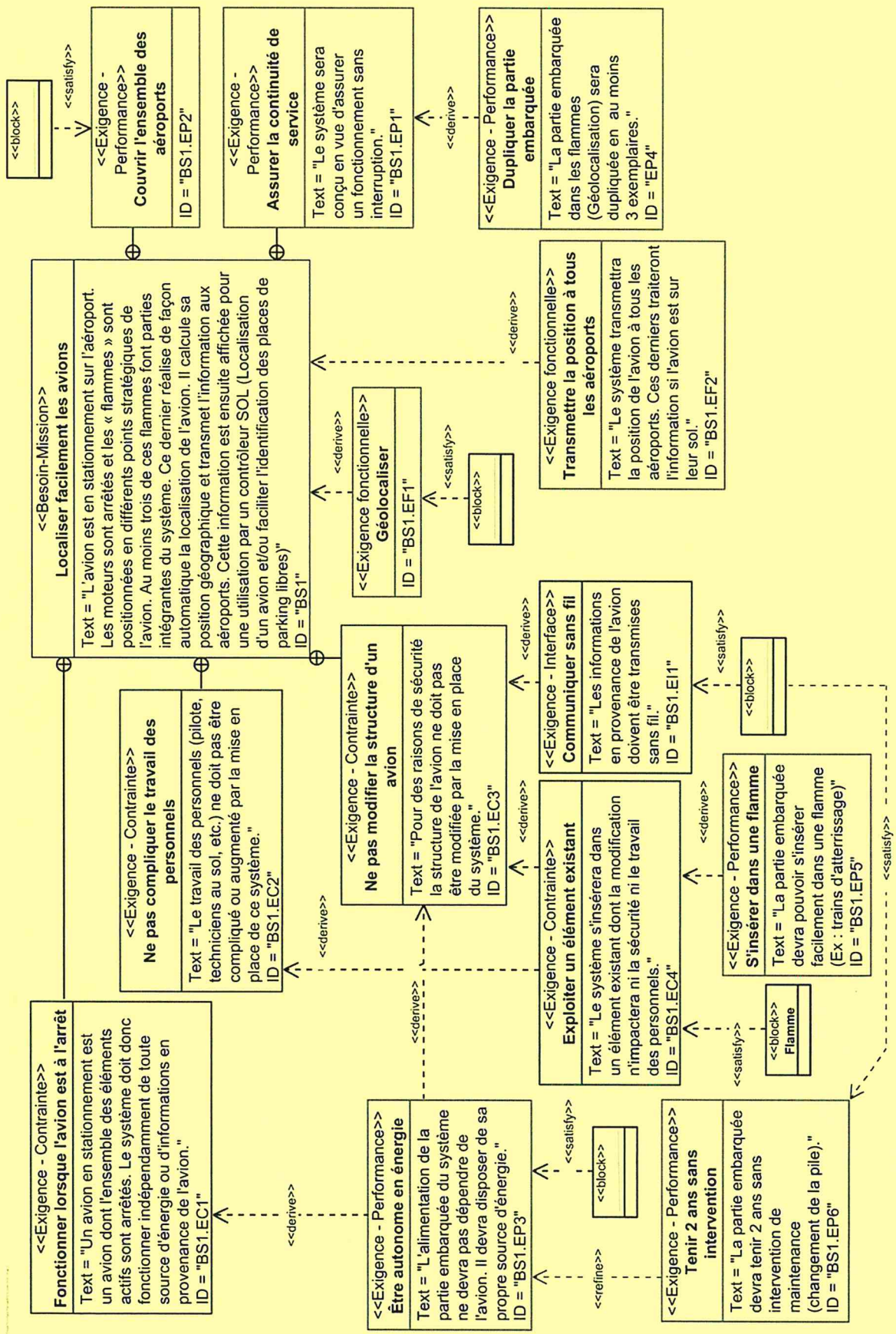
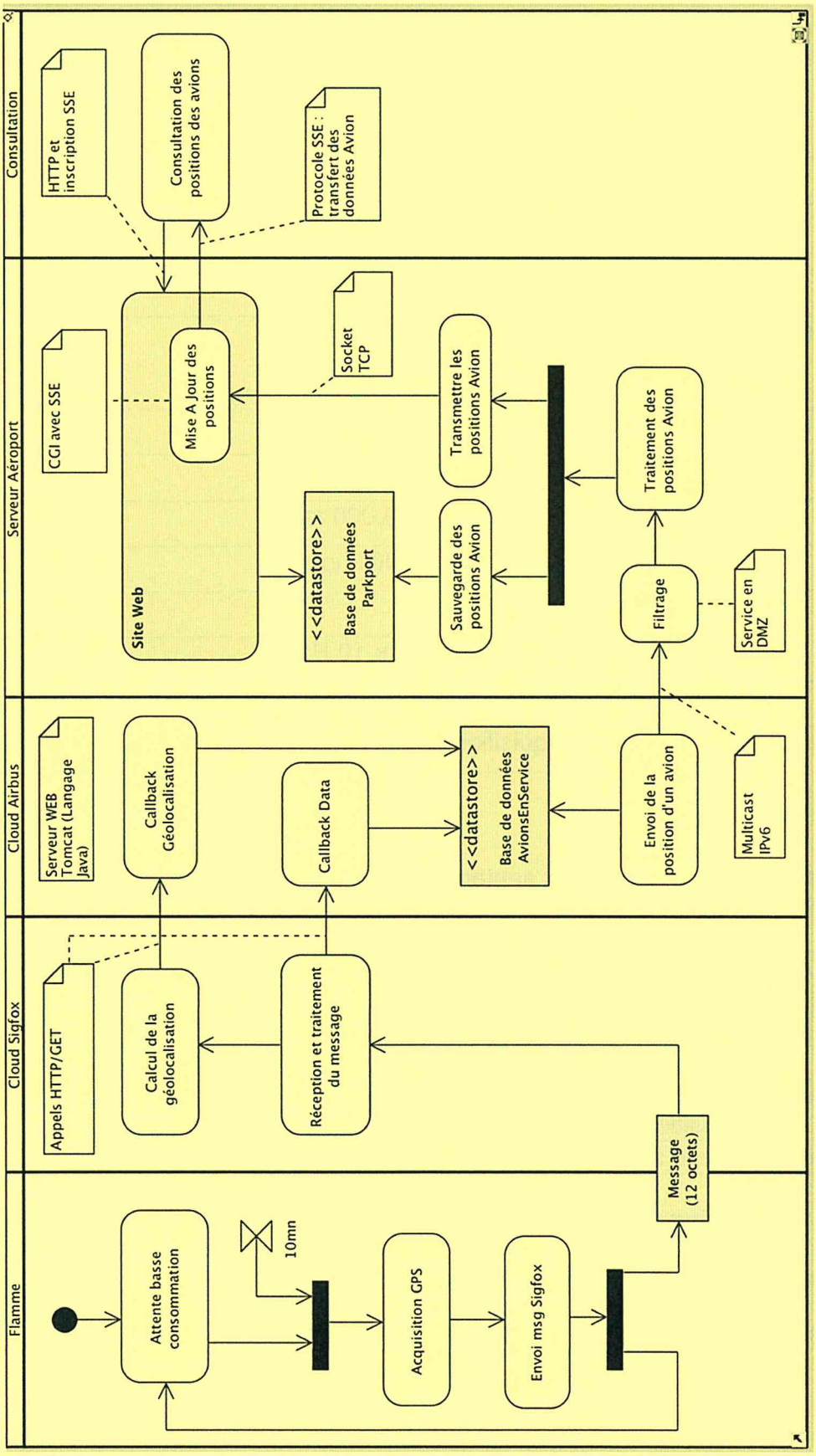
DOCUMENTATION


<<block>>
<<satisfy>>
<<Exigence - Performance>>
Couvrir l'ensemble des aéroports
ID = "BS1.EP2"
<<Exigence - Performance>>
Assurer la continuité de service
Text = "Le système sera conçu en vue d'assurer un fonctionnement sans interruption."
ID = "BS1.EP1"
<<derive>>
<<Exigence - Performance>>
Dupliquer la partie embarquée
Text = "La partie embarquée dans les flammes (Géolocalisation) sera dupliquée en au moins 3 exemplaires."
ID = "EP4"
<<Besoin-Mission>>
Localiser facilement les avions
Text = "L'avion est en stationnement sur l'aéroport. Les moteurs sont arrêtés et les « flammes » sont positionnées en différents points stratégiques de l'avion. Au moins trois de ces flammes font parties intégrantes du système. Ce dernier réalise de façon automatique la localisation de l'avion. Il calcule sa position géographique et transmet l'information aux aéroports. Cette information est ensuite affichée pour une utilisation par un contrôleur SOL (Localisation d'un avion et/ou faciliter l'identification des places de parking libres)"
ID = "BS1"
<<Exigence fonctionnelle>>
Géolocaliser
ID = "BS1.EF1"
<<Exigence fonctionnelle>>
Transmettre la position à tous les aéroports
Text = "Le système transmettra la position de l'avion à tous les aéroports. Ces derniers traiteront l'information si l'avion est sur leur sol."
ID = "BS1.EF2"
<<Exigence - Contrainte>>
Ne pas compliquer le travail des personnels
Text = "Le travail des personnels (pilote, techniciens au sol, etc.) ne doit pas être compliqué ou augmenté par la mise en place de ce système."
ID = "BS1.EC2"
<<Exigence - Contrainte>>
Ne pas modifier la structure d'un avion
Text = "Pour des raisons de sécurité la structure de l'avion ne doit pas être modifiée par la mise en place du système."
ID = "BS1.EC3"
<<Exigence - Interface>>
Communiquer sans fil
Text = "Les informations en provenance de l'avion doivent être transmises sans fil."
ID = "BS1.EI1"
<<Exigence - Contrainte>>
Fonctionner lorsque l'avion est à l'arrêt
Text = "Un avion en stationnement est un avion dont l'ensemble des éléments actifs sont arrêtés. Le système doit donc fonctionner indépendamment de toute source d'énergie ou d'informations en provenance de l'avion."
ID = "BS1.EC1"
<<Exigence - Contrainte>>
Exploiter un élément existant
Text = "Le système s'insérera dans un élément existant dont la modification n'impactera ni la sécurité ni le travail des personnels."
ID = "BS1.EC4"
<<Exigence - Performance>>
Être autonome en énergie
Text = "L'alimentation de la partie embarquée du système ne devra pas dépendre de l'avion. Il devra disposer de sa propre source d'énergie."
ID = "BS1.EP3"
<<Exigence - Performance>>
S'insérer dans une flamme
Text = "La partie embarquée devra pouvoir s'insérer facilement dans une flamme (Ex : trains d'atterrissage)"
ID = "BS1.EP5"
<<Exigence - Performance>>
Tenir 2 ans sans intervention
Text = "La partie embarquée devra tenir 2 ans sans intervention de maintenance (changement de la pile)."
ID = "BS1.EP6"
<<block>>
Flamme
<<derive>>
<<satisfy>>
<<refine>>


Consultation
Serveur Aéroport
Cloud Airbus
Cloud Sigfox
Flamme
HTTP et inscription SSE
Consultation des positions des avions
Protocole SSE : transfert des données Avion
CGI avec SSE
Site Web
Mise A Jour des positions
Socket TCP
Transmettre les positions Avion
Traitement des positions Avion
<<datastore>>
Base de données Parkport
Sauvegarde des positions Avion
Filtrage
Service en DMZ
Serveur WEB Tomcat (Langage Java)
Callback Géolocalisation
Callback Data
<<datastore>>
Base de données AvionsEnService
Envoi de la position d'un avion
Multicast IPv6
Appels HTTP/GET
Calcul de la géolocalisation
Réception et traitement du message
Message (12 octets)
Attente basse consommation
10mn
Acquisition GPS
Envoi msg Sigfox

DOCUMENTATION PP 3 : module GPS FGMMOPA6H (Extraits)

Specification List

	Description
GPS Solution	MTK MT3339
Frequency	L1, 1575.42 MHz
Sensitivity	Acquisition: -148 dBm, cold start Reacquisition: -163 dBm, Hot start Tracking: -165d Bm
Channel	66 channels
TTFF = Time-To-First FIX	Hot start: 1 second typical Warm start: 33 seconds typical Cold start: 35 seconds typical
Position Accuracy	Without aid :3.0 m (50 % CEP)
Velocity Accuracy	Without aid : 0.1 m/s
Altitude	Maximum 18 000 m (60,000 feet)
Velocity	Maximum 515 m/s (1000 knots)
Acceleration	Maximum 4 G
Update Rate	1 Hz (default), maximum 10 Hz
Baud Rate	9600 bps (default)
Current Consumption	25 mA acquisition, 20 mA tracking

NMEA Output Sentences

Table-1 lists each of the NMEA output sentences specifically developed and defined by MTK for use within MTK products

Table-1 : NMEA Output Sentence	
Option	Description
GGA	Time, position and fix type data.
GSA	GPS receiver operating mode, active satellites used in the position solution and
GSV	The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and
RMC	Time, date, position, course and speed data. Recommended Minimum Navigation
VTG	Course and speed information relative to the ground.

GGA—Global Positioning System Fixed Data. Time, Position and fix related data

Table-2 contains the values for the following example :

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

Table-2 : GGA Data Format			
Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	064951.000		hhmmss.sss
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See Table-3
Satellites Used	8		Range 0 to 14
HDOP	0.95		Horizontal Dilution of Precision
MSL Altitude	39.9	m	Antenna Altitude above/below mean-sea-level
Units	M	m	Units of antenna altitude
Geoidal Separation	17.8	m	
Units	M	m	Units of geoids separation
Age of Diff. Corr.		s	Null fields when DGPS is not used
Checksum + <CR> + <LF>	*65		Error control + End of message termination

Table-3 : Position Fix Indicator			
Value	0	1	2
Description	Fix not available	GPS fix	Differential GPS fix

...

DOCUMENTATION PP 4 : MTK Packet Format (Extraits)

PMTK command is MediaTek proprietary data transfer protocol for GNSS. It enables configuring the parameters of the GNSS chipset, aiding assistance position information and receive notifications from the GNSS chip.
The PMTK commands are all send in a specific packet format which is shown below.

Preamble	Talker ID	Pkt Type	Date Field	*	CHK1	CHK2	CR	LF

Field	Length	Type	Description
Preamble	1 byte	Character	"$"
Talker ID	4 byte	Character string	"PMTK"
Pkt Type	3 byte	Character string	From "000" to "999", an identifier used to tell the decoder how to decode the packet
Data Field	variable	-	A "," must be inserted before each data field to help decoder process the Data Field
*	1 byte	Character	The star symbol is used make the end of Data Field
CHK1, CHK2	2 byte	Character string	Checksum of the data between preamble "$" and "*"
CR, LF	2 byte	Binary data	Used to identify the end of a packet

Packet Type : 314 PMTK_API_SET_NMEA_OUTPUT

Packet Meaning : This command sets NMEA sentence output frequencies
Data Field : This packet contains 19 data fields used to set the output frequencies for the 19 supported NMEA sentences individually.
Supported NMEA Sentences :

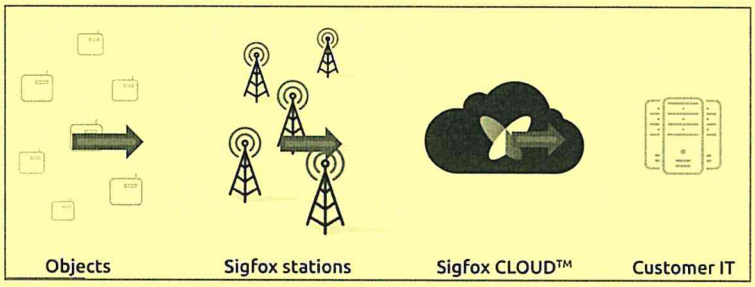
data field position	Sentence	Supported Frequency Setting
0	NMEA_SEN_GLL, // GLL interval - Geographic Position - Latitude longitude	0 = Disabled or not supported 1 = Output once every one position fix 2 = Output once every two position fixes 3 = Output once every three position fixes 4 = Output once every four position fixes 5 = Output once every five position fixes
1	NMEA_SEN_RMC, // RMC interval - Recommended Minimum Specific GNSS Sentence	
2	NMEA_SEN_VTG, // VTG interval -	
3	NMEA_SEN_GGA, // GGA interval - GPS Fix Data	
4	NMEA_SEN_GSA, // GSA interval - GNSS DOPS and Active Satellites	
5	NMEA_SEN_GSV, // GSV interval - GNSS	
6 à 16	//Reserved	
17	NMEA_SEN_ZDA, // ZDA interval – Time & Date	
18	NMEA_SEN_MCHN, // PMTKCHN interval – GPS channel status	

Example : $PMTK314,0,1,1,1,1,5,0,0,0,0,0,0,0,0,0,0,0,0,0*2C<CR><LF>

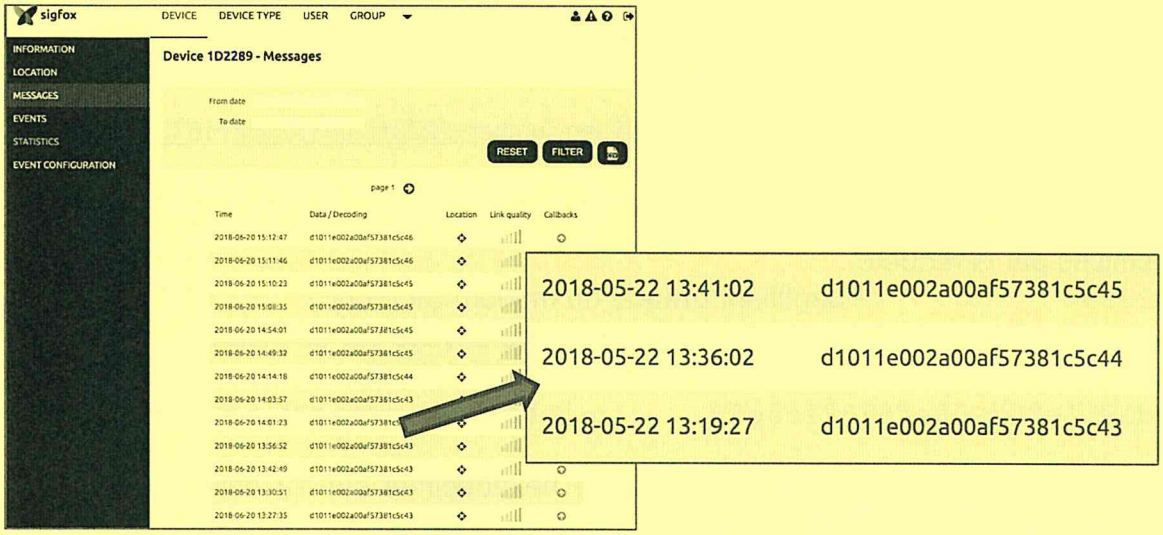
Note : This command set NMEA sentences' outputs as the following :
At 1 update rate (default 1s) : RMC, VTG, GGA, GSA ;
At 5 update rate (default 1s) : GSV.

DOCUMENTATION PP 5 : SIGFOX

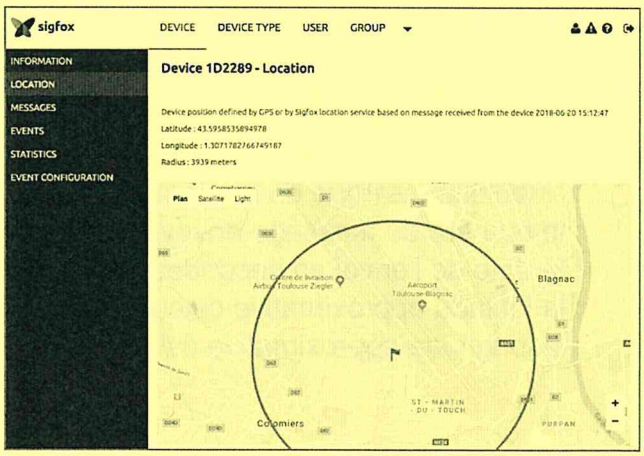
Sigfox, société Toulousaine, propose des services IoT (internet des objets) autour d'un réseau longue portée, très faible consommation, bas débit et faible coût, qui permet la communication de données de taille réduite entre des objets connectés et des serveurs, sans passer par la téléphonie mobile.
Le service coûte moins de 1€ par mois et par objet avec des prix dégressifs en quantité.



Le réseau SIGFOX permet la transmission de messages de 12 octets à raison de 140 messages par jour. Les messages sont disponibles en lecture **et en hexadécimal** sur le serveur Web du Cloud SIGFOX.



SIGFOX propose en option (sans utilisation de GPS) un service de géolocalisation (nommé Spot'it) de l'objet ayant émis le message. Cette géolocalisation se fait à 5 km près.



Pour le développement d'une application intégrée chez le client, le Serveur SIGFOX peut être paramétré pour lui demander de transmettre ces informations à un serveur applicatif (Customer IT), par requête http de type GET à des 'callback'.

Le Cloud SIGFOX appellera à chaque réception d'un message :

1. Une Callback pour transmettre le message reçu et des informations sur le message : date de l'envoi, ID du modem, numéro du message, etc ;
2. Avec l'option de géolocalisation, une autre Callback pour transmettre la position calculée à partir des antennes SIGFOX et des informations système : date de l'envoi, ID du modem, numéro du message, etc.

Configuration d'un message DATA sur le serveur SIGFOX :

http://www.aes.airbus.com/callback1.jsp?id={device}&numsg={seqNumber}&time={time}&data={data}

Le dernier champ contient la trame des données (data).

Configuration d'un message GEOLOC sur le serveur SIGFOX :

http://www.aes.airbus.com/callback2.jsp?id={device}&numsg={seqNumber}&time={time}&lat={lat}&lng={lng}

Appels du serveur SIGFOX vers le serveur applicatif :

1. Du message DATA :

http://www.aes.airport.airbus.com/callback1.jsp?id=1D2289&seqNumber=222&time=1529587167&data=**d1011e002a00af57381c5c43**

Données transmises par la requête :

`id=1D2289` : l'identifiant **unique** du modem SIGFOX ;
`seqNumber=222` : `le numéro` du message envoyé par ce modem ;
`time=1529587167` : la date de l'envoi en secondes depuis le 1 Janvier 1970 ;
`data=d1011e002a00af57381c5c43` : la trame envoyée par le modem.

On remarque que la data du message, exprimée en hexadécimal sur 12 octets, n'est pas visuellement déchiffrable.

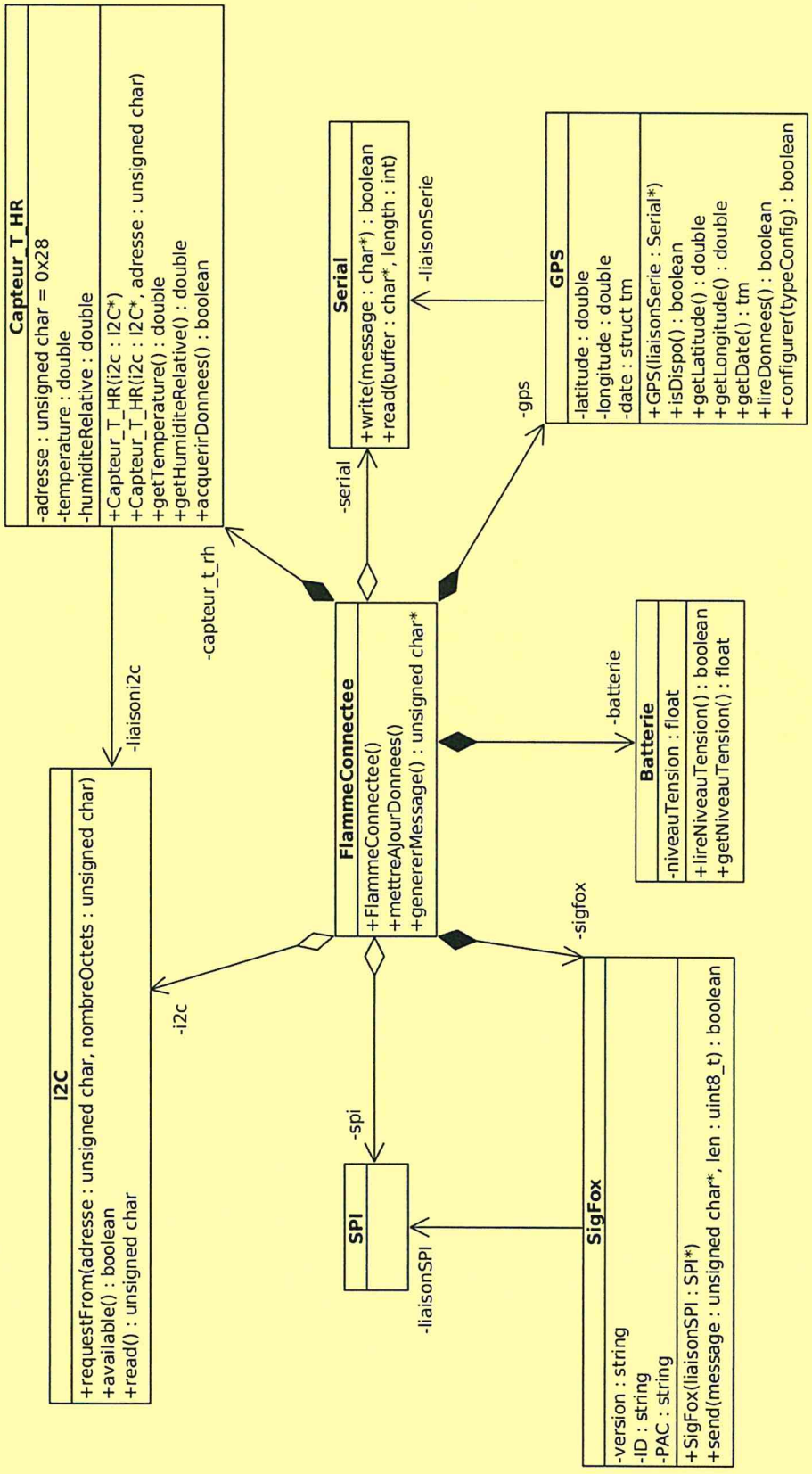
1. Du message GEOLOC :

http://www.aes.airport.airbus.com/callback2.jsp?id=1D2289&seqNumber=222&time=1529587167&lat=43.623676&lng=1.359716

Données transmises par la requête :

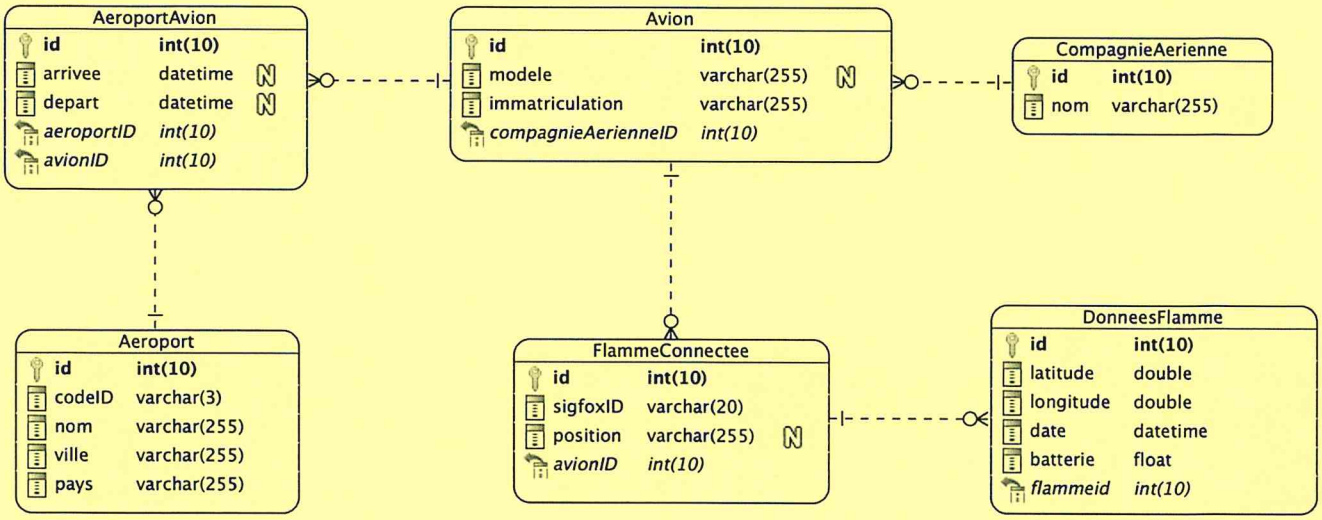
`id=1D2289` : l'identifiant **unique** du modem SIGFOX ;
`seqNumber=222` : le numéro du message envoyé par ce modem ;
`time=1529587167` : la date de l'envoi en secondes depuis le 1 Janvier 1970 ;
`lat=43.623676` : la latitude approximative calculée par SIGFOX ;
`lng=1.359716` : la longitude approximative calculée par SIGFOX.

DOCUMENTATION PP 6 : Diagramme de classes de la flamme connectée

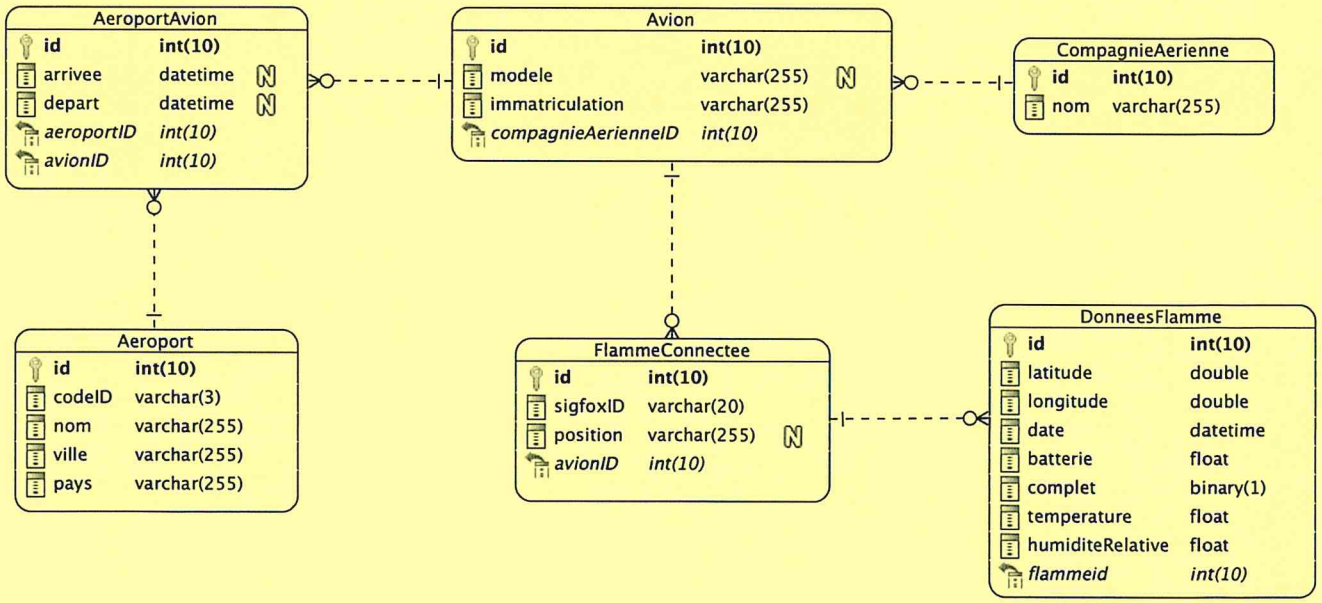


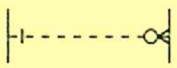
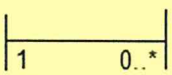
DOCUMENTATION PP 7 : Diagrammes entités/relations

- Base de données `avionsEnService` avant l'amélioration du sytème :



- Base de données `avionsEnService` suite à l'amélioration du système :



Notation « crow's foot »	Equivalence UML
	

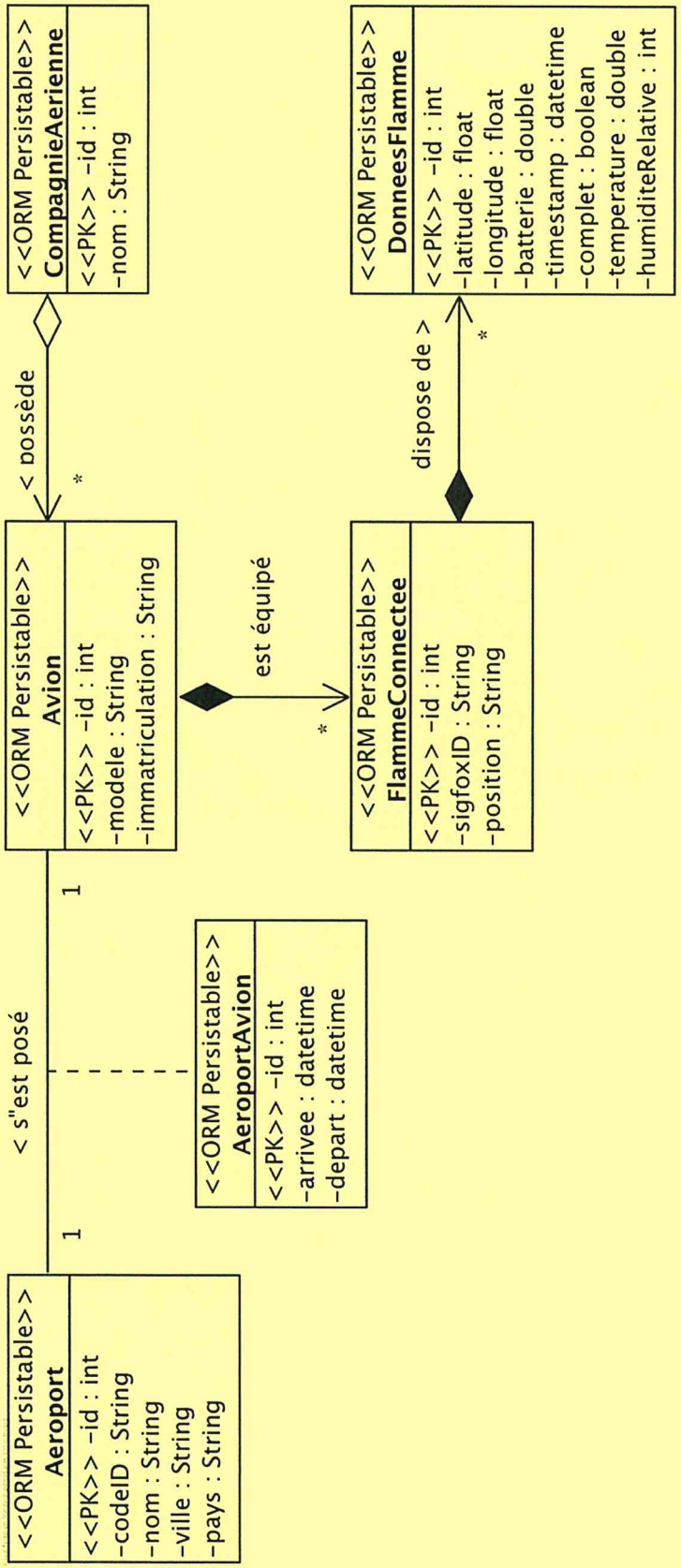
DOCUMENTATION PP 8 : Principales requêtes SQL

Utiliser (rendre active) une base de données existante :	**use** nom_de_la_base;
Créer une base de données :	**create database** nom_de_la_base;
Supprimer une base de données	**drop database** nom_de_la_base;
Créer une table dans la base de données active	**create table** nomTable (id **int auto_increment**, champ1 **double**, champ2 **float**, champ3 **varchar**, champ4 **timestamp not null**, champ5 **boolean default false**, ..., **primary key**(id));
Lister la structure d'une table	**describe** nomTable;
Sélectionner toutes les informations de la table	**select** * **from** nomTable;
Sélectionner seulement les informations d'un champ	**select** nomChamp **from** nomTable;
Sélectionner tous les champs de la table nomTable correspondant à deux critères.	**select** * **from** nomTable **where** nomChamp1 = 'poste' **and** nomChamp3 < 12;
Sélectionner sur plusieurs tables (jointure) nomTable1.nomChamp1 est clé primaire. nomTable2.nomChamp4 est une clé étrangère vers nomTable1.	**select** * **from** nomTable1, nomTable2 **where** nom_table1.nomChamp1 = nom_table2.nomChamp4;
Écrire une nouvelle entrée dans une table	**insert into** nomTable(champ1,champ2) **values**(32.327432, 'un texte');
Modifier les informations d'un enregistrement dont le champ date = '2018/07/21 0:28:12';	**update** nomTable **set** nomChamp1 = 10, valeur2 = 32 **where** date = '2018/07/21 0:28:12';
Ajouter des nouveaux champs (colonnes) dans une table	**alter table** nomTable **add** champ1 **double**, **add** champ2 **boolean default false**;

Remarque : Dans la colonne de droite les mots en gras sont des mots réservés par le langage SQL.

DOCUMENTATION PP 9 : Diagramme des classes persistantes

Après amélioration du système :



Format des adresses IPv6

Les adresses IPv6 contiennent 16 octets soit 8 mots (1 mot = 1groupe de 2 octets) présentés en hexadécimale. Les 8 groupes de 2 octets (soit 16 bits par groupe) sont séparés par un signe deux-points :

2001:0db8:0000:fea3:0000:0000:ac1f:5b01

Il est permis d'omettre de 1 à 3 chiffres zéros non significatifs dans chaque groupe de 4 chiffres hexadécimaux. Ainsi, l'adresse IPv6 ci-dessus est équivalente à :

2001:db8:0:fea3:0:0:ac1f:5b01

De plus, une unique suite de un ou plusieurs groupes consécutifs de 16 bits tous nuls peut être omise, en conservant toutefois les signe deux-points de chaque côté de la suite de chiffres omise, c'est-à-dire une (seule) paire de deux-points (::) . Ainsi, l'adresse IPv6 ci-dessus peut être abrégée en :

2001:db8:0:fea3::ac1f:5b01

En revanche l'écriture suivante n'est pas valide car elle peut avoir plusieurs solutions possibles :

~~**2001:db8::fea3::ac1f:5b01**~~

Adresses IPv6 Multicast

Tableau 11–1 Format d'adresse IPv6 multicast (128 bits)

8 bits	4 bits	4 bits	8 bits	8 bits	64 bits	32 bits
11111111	*FLGS*	*SCOP*	*Réservé 0x00*	*P-len*	*Préfixe réseau*	*ID de groupe de diffusion*

Les adresses multicast IPv6 sont dérivées du préfixe **FF00::/8**.

Le champ drapeaux ***FLGS*** de 4 bits (0,0,P,T) est défini de la manière suivante : le bit T (*Temporaire*) a valeur 0 indique une adresse multicast bien connue gérée par une autorité. La valeur 1 indique une valeur temporaire. P =0 signifie que l'adresse multicast n'est pas assignée en fonction d'un préfixe réseau. Les autres bits de ce champ ne sont pas considérés et mis à 0.

Le champ ***SCOP*** de l'adresse multicast IPv6 permet d'en limiter la portée (*scope* en anglais). En IPv4, la portée d'un paquet est limitée par le champ TTL (*Time To Live*). En IPv6, les préfixes peuvent être définis pour identifier des adresses à portée réduite. Les valeurs suivantes du champ ***SCOP*** sont définies :

- 1 - node-local
- 2 - link-local
- 3 - subnet-local
- 4 - admin-local
- 5 - site-local
- 8 - organisation-local
- E - global

Les portées 0 et F sont réservées.

P-len indique le nombre de bits du champ ***Préfixe réseau*** à prendre en considération pour une trame multicast à portée limitée.

Préfixe réseau identifie le réseau de la portée du multicast.

ID de groupe de diffusion identifie le groupe de diffusion. Sa valeur est en général une transcription d'une adresse IPv4 multicast

DOCUMENTATION PP 11 : Éléments du manuel ip6tables

NOM

ip6tables - outil d'administration pour le filtrage de paquets IPv6

DESCRIPTION

ip6tables est utilisé pour mettre en place, maintenir et inspecter les tables des règles de filtrage des paquets IPv6 du noyau Linux. Différentes tables peuvent être définies. La table par défaut est la table de filtrage 'filter' présentée succinctement ci-dessous :

filter :

C'est la table par défaut (si l'option -t est omise). Elle contient les chaînes prédéfinies INPUT (pour les paquets entrants dans la machine), FORWARD (pour les paquets routés à travers la machine) et OUTPUT (pour les paquets générés localement).

les principales COMMANDES

Ces options précisent une action particulière à accomplir. Une seule option peut être indiquée sur la ligne de commande, sauf indication contraire. Pour tous les noms en version longue des commandes et des options, vous avez le droit d'utiliser un nombre restreint de lettres du moment qu' ip6tables peut identifier chaque commande sans ambiguïté.

-A, --append *chaîne règle*

Ajoute une ou plusieurs règles à la fin de la chaîne sélectionnée. Lorsque les noms source et/ou destination désignent plus d'une adresse, une règle sera ajoutée pour chaque combinaison d'adresses possible.

-D, --delete *chaîne règle*

Efface une ou plusieurs règles de la chaîne sélectionnée. Il y a deux versions de cette commande : on peut désigner la règle par sa position dans la chaîne avec un numéro (commençant à 1 pour la première règle) ou bien par une règle de correspondance avec sa syntaxe exacte.

les CHAINES standards :

FORWARD : chaine désignant les paquets désirant traverser le pare-feu
INPUT : chaine désignant les paquets s'adressant au pare-feu lui même
OUPUT : chaine désignant les paquets expédiés par le pare-feu lui-même
PREROUTING : chaine des paquets attendant t'être routés
POSTROUTING : chaine des paquets venant t'être routés

les principaux PARAMÈTRES

Les paramètres suivants composent une spécification de règle (quand ils sont utilisés dans les commandes **add, delete, insert, replace** et **append**).

-p, --protocol [!] *protocole*

Protocole de la règle ou du paquet à vérifier. Le protocole spécifié est l'un des suivants : *tcp*, *udp*, *ipv6-icmp|icmpv6*, ou *all*, ou bien sous forme d'une valeur numérique, représentant un de ces protocoles ou un protocole différent. Un nom de protocole issu du fichier /etc/protocols est aussi autorisé. Un «!» avant le protocole inverse le test. La valeur zéro est équivalente à *all*. Le protocole *all* correspond à tous les protocoles ; c'est aussi la valeur par défaut lorsque cette option est omise.

-s, --source [!] *adresse*[/*masque*]

Spécification de la source. L'*adresse* peut être un nom d'hôte (attention : spécifier un nom à résoudre avec une requête distante de type DNS est vraiment une mauvaise idée), une adresse de réseau IPv6 (avec /masque) ou une simple adresse IPv6 (un nom de réseau n'est pas encore pris en charge). Le *masque* peut être un masque de réseau ou un nombre entier spécifiant le nombre de bits égaux à 1 dans la partie gauche du masque de réseau (bits de poids fort). Par conséquent, un masque de *64* est équivalent à **ffff:ffff:ffff:ffff:0000:0000:0000:0000**. Un «!» avant la spécification d'adresse inverse la sélection d'adresse. L'option --src est un synonyme de --source.

-d, --destination [!] *adresse*[/*masque*]

Spécification de la destination. Voir la description du paramètre -s (source) pour une description détaillée de la syntaxe. L'option --dst est un synonyme de --destination.

-i, --in-interface [!] [*nom*]

Nom de l'interface qui reçoit les paquets (seulement pour les paquets passant par les chaînes INPUT, FORWARD et PREROUTING). Lorsqu'un «!» est utilisé avant le nom d'interface, la sélection est inversée. Si le nom de l'interface se termine par un «+», il désigne toutes les interfaces commençant par ce nom. Si cette option est omise, toutes les interfaces réseau sont désignées.

-o, --out-interface [!] [*nom*]

Nom de l'interface qui envoie les paquets (seulement pour les paquets passant par les chaînes FORWARD et OUTPUT). Lorsqu'un «!» est utilisé avant le nom d'interface, la sélection est inversée. Si le nom de l'interface se termine par un «+», il désigne toutes les interfaces commençant par ce nom. Si cette option est omise, toutes les interface réseau sont désignées.

-j, --jump *cible*

Ceci détermine la cible de la règle ; c'est-à-dire ce qu'il faut faire si le paquet correspond à la règle.

CIBLES

Une règle de pare-feu spécifie des critères de correspondance pour un paquet, et une cible. Si le paquet correspond, la règle suivante est déterminée par la valeur de la cible, qui peut être une des valeurs spéciales suivantes : *ACCEPT*, *DROP*,...

ACCEPT signifie que le paquet est autorisé à passer.

DROP signifie que le paquet est détruit.

...

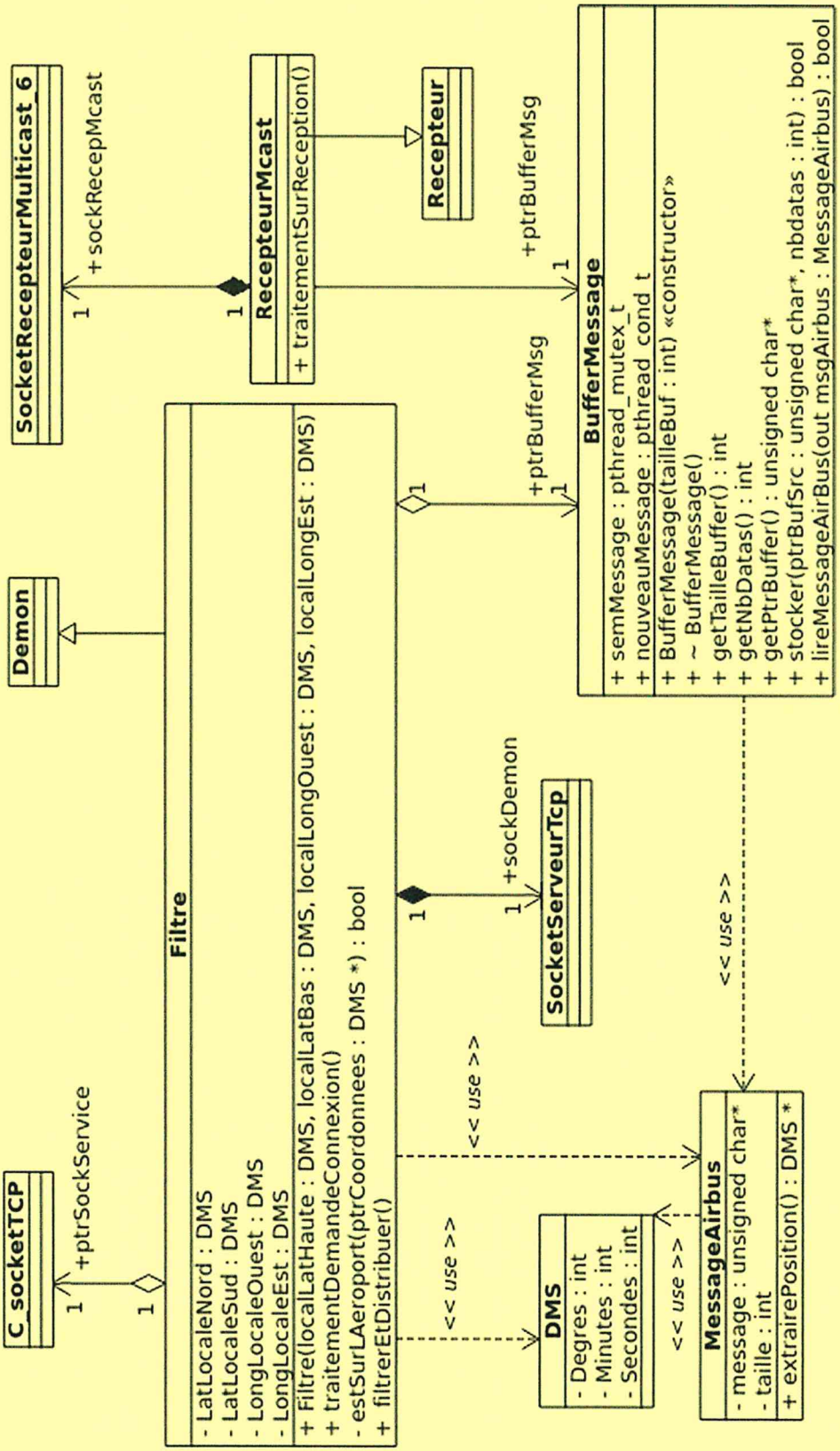
Exemples :

ip6tables -A FORWARD -i eth0 -o eth1 -p icmp -j ACCEPT

ajoute une règle qui autorise les paquets icmp à traverser la machine s'il rentre par l'interface réseau eth0 et sort par l'interface réseau eth1.

ip6tables -A INPUT -s 20a2::/16 -i eth0 -j DROP

ajoute une règle qui jette tous les paquets IPv6 dont les 16 bits de poids forts de l'adresse de l'expéditeur sont identiques à **0x20a2**, entrant par l'interface **eth0** et destinés à cette machine


SocketRecepteurMulticast_6
+ sockRecepMcast
RecepteurMcast
+ traitementSurReception()
Recepteur
+ptrBufferMsg
BufferMessage
+ semMessage : pthread_mutex_t
+ nouveauMessage : pthread_cond_t
+ BufferMessage(tailleBuf : int) «constructor»
+ ~ BufferMessage()
+ getTailleBuffer() : int
+ getNbDatas() : int
+ getPtrBuffer() : unsigned char*
+ stocker(ptrBufSrc : unsigned char*, nbdatas : int) : bool
+ lireMessageAirBus(out msgAirbus : MessageAirbus) : bool
Demon
Filtre
- LatLocaleNord : DMS
- LatLocaleSud : DMS
- LongLocaleOuest : DMS
- LongLocaleEst : DMS
+ Filtre(localLatHaute : DMS, localLatBas : DMS, localLongOuest : DMS, localLongEst : DMS)
+ traitementDemandeConnexion()
- estSurLAeroport(ptrCoordonnees : DMS *) : bool
+ filtrerEtDistribuer()
+sockDemon
SocketServeurTcp
C_socketTCP
+ptrSockService
<< use >>
<< use >>
<< use >>
<< use >>
DMS
- Degres : int
- Minutes : int
- Secondes : int
MessageAirbus
- message : unsigned char*
- taille : int
+ extrairePosition() : DMS *
1

```
10 BufferMessage::BufferMessage()
11 {
12   nbDatas =0;
13   int tailleBuffer = MAX_MSG;
14   // MAX_MSG est la taille maximale d'un message de
15   // positionnement d'avion
16   buffer = new unsigned char[tailleBuffer];
17   pthread_mutex_init(&semMessage, NULL); //initialisation du mutex
18 }
19
20
21 bool BufferMessage::stocker (unsigned char* msg, int taille)
22 {
23   if( nbDatas >0) return false;
24   pthread_mutex_lock(&semMessage);
25   nbDatas = taille;
26   memcpy(buffer ,msg , nbDatas);
27   pthread_mutex_unlock(&semMessage);
28   pthread_cond_signal(&nouveauMessage);
29   return true;
30 }
31
32
33 bool BufferMessage::lireMessageAirbus(MessageAirbus &msgABus)
34 {
35   if( nbDatas >0) return false;
36   pthread_mutex_lock(&semMessage);
37   msgABus.taille = nbDatas;
38   memcpy( msgABus.message ,buffer ,msgABus.taille);
39   nbDatas = 0; // pour signifier que le message a été lu
40   pthread_mutex_unlock(&semMessage);
41   return true;
42 }
```

DOCUMENTATION PP 14 : Outils de synchronisation entre threads

I) Les mutex

pthread_mutex_init, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock, pthread_mutex_destroy

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

pthread_mutex_lock() verrouille le mutex. Si le mutex est déverrouillé, il devient verrouillé et est possédé par le thread appelant et **pthread_mutex_lock**() rend la main immédiatement. Si le mutex est déjà verrouillé par un autre thread, **pthread_mutex_lock** suspend le thread appelant jusqu'à ce que le mutex soit déverrouillé.

pthread_mutex_lock() rend la main immédiatement avec un code de retour indiquant le succès, enregistrant le nombre de fois où le thread appelant a verrouillé le mutex. Un nombre égal d'appels à **pthread_mutex_unlock**() doit être réalisé avant que le mutex retourne à l'état déverrouillé.

pthread_mutex_unlock() déverrouille le mutex. Celui-ci est supposé verrouillé, et ce par le thread courant en entrant dans **pthread_mutex_unlock**(). Si le mutex est de type « rapide », **pthread_mutex_unlock**() le réinitialise toujours à l'état déverrouillé. S'il est de type « récursif », son compteur de verrouillage est décrémenté (nombre d'opérations **pthread_mutex_lock**() réalisées sur le mutex par le thread appelant), et déverrouillé seulement quand ce compteur atteint 0.

Sur les mutex « vérification d'erreur », **pthread_mutex_unlock**() vérifie lors de l'exécution que le mutex est verrouillé en entrant, et qu'il est verrouillé par le même thread que celui appelant **pthread_mutex_unlock**() Si ces conditions ne sont pas réunies, un code d'erreur est renvoyé et le mutex n'est pas modifié. Les mutex « rapide » et « récursif » ne réalisent pas de tels tests, permettant à un mutex verrouillé d'être déverrouillé par un thread autre que celui l'ayant verrouillé. Ce comportement n'est pas portable et l'on ne doit pas compter dessus.

EXEMPLE

Une variable globale partagée *x* peut être protégée par un mutex comme suit :

```
int x;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
```

Tous les accès et modifications de *x* doivent être entourés de paires d'appels à **pthread_mutex_lock**() et **pthread_mutex_unlock**() comme suit :

```
pthread_mutex_lock(&mut);
/* agir sur x */
pthread_mutex_unlock(&mut);
```

II) Les événements internes

pthread_cond_init, pthread_cond_destroy, pthread_cond_signal, pthread_cond_broadcast, pthread_cond_wait, pthread_cond_timedwait

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
```

DESCRIPTION

Une condition (abréviation pour « variable condition ») est un mécanisme de synchronisation permettant à un thread de suspendre son exécution jusqu'à ce qu'une certaine condition (un prédicat) sur des données partagées soit vérifiée. Les opérations fondamentales sur les conditions sont : signaler la condition (quand le prédicat devient vrai), et attendre la condition en suspendant l'exécution du thread jusqu'à ce qu'un autre thread signale la condition.

Une variable condition doit toujours être associée à un mutex, pour éviter les accès concurrents où un thread se prépare à attendre une condition et un autre signale la condition juste avant que le premier n'attende réellement.

pthread_cond_signal() relance l'un des threads attendant la variable condition *cond*. S'il n'existe aucun thread répondant à ce critière, rien ne se produit. Si plusieurs threads attendent sur *cond*, seul l'un d'entre eux sera relancé, mais il est impossible de savoir lequel.

pthread_cond_wait() déverrouille atomiquement le *mutex* (comme **pthread_mutex_unlock**) et attend que la variable condition *cond* soit signalée. L'exécution du thread est suspendue et ne consomme pas de temps CPU jusqu'à ce que la variable condition soit signalée. Le *mutex* doit être verrouillé par le thread appelant à l'entrée de **pthread_cond_wait**(). Avant de rendre la main au thread appelant, **pthread_cond_wait**() reverrouille *mutex* (comme **pthread_mutex_lock**. Le déverrouillage du mutex et la suspension de l'exécution sur la variable condition sont liés atomiquement. Donc, si tous les threads verrouillent le mutex avant de signaler la condition, il est garanti que la condition ne peut être signalée (et donc ignorée) entre le moment où un thread verrouille le mutex et le moment où il attend sur la variable condition.

VALEUR RENVOYÉE

Toutes ces fonctions renvoient 0 en cas de succès et un code d'erreur non nul en cas de problème.

EXEMPLE

Considérons deux variables globales partagées *x* et *y*, protégées par le mutex *mut*, et une variable condition *cond* pour signaler que *x* devient plus grand que *y*.

```
int x,y;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Attendre que *x* devienne plus grand que *y* se réalise de la manière suivante :

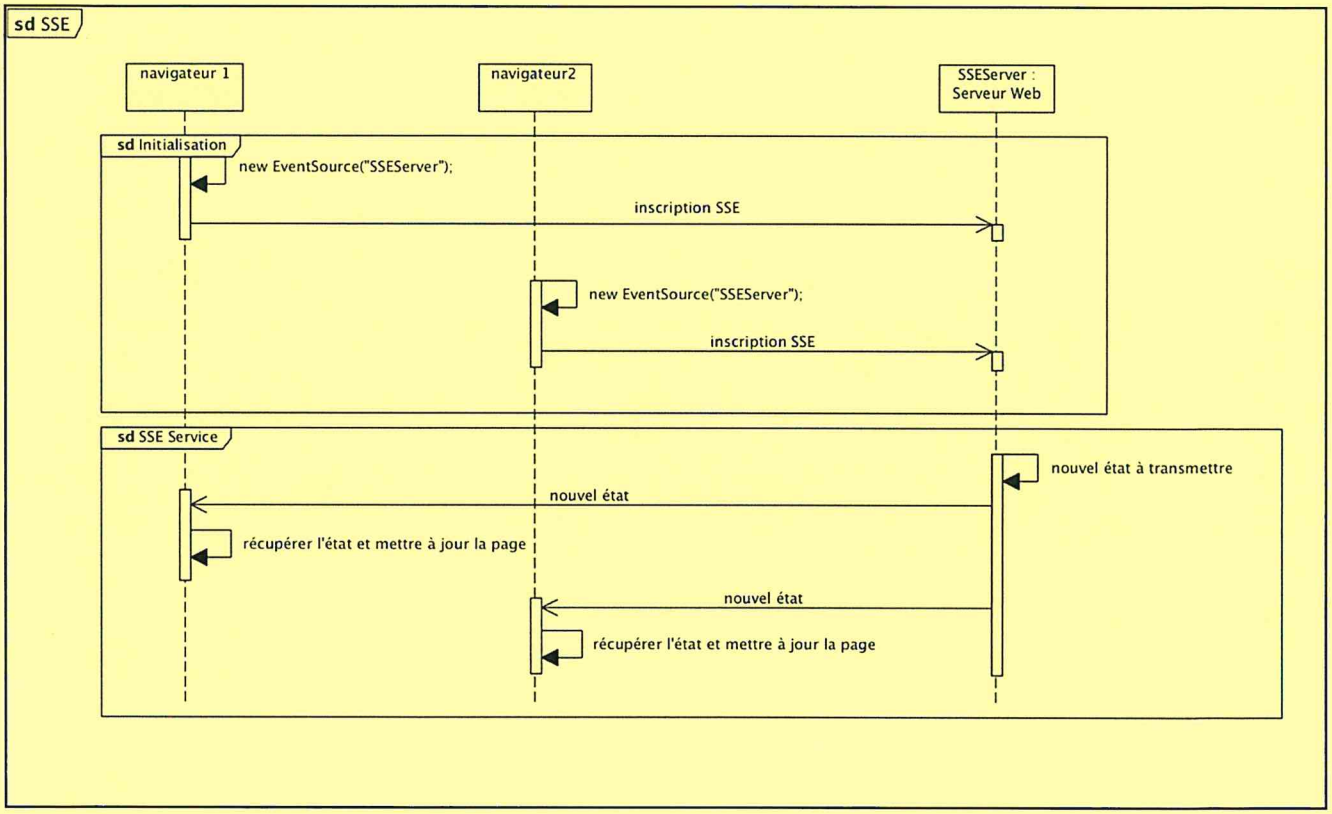
```
pthread_mutex_lock(&mut);
while (x <= y) {
    pthread_cond_wait(&cond, &mut);
}
/* agir sur x et y */
pthread_mutex_unlock(&mut);
```

Les modifications de *x* et *y* qui peuvent rendre *x* plus grand que *y* doivent signaler la condition si nécessaire :

```
pthread_mutex_lock(&mut);
/* modifer x et y */
if (x > y) pthread_cond_broadcast(&cond);
pthread_mutex_unlock(&mut)
```

Description

Server-sent events est une norme décrivant la façon dont les serveurs HTTP peuvent initier la transmission de données vers les clients une fois que la connexion initiale du client a été mise en place. Cette technologie est couramment utilisée pour envoyer des mises à jour de messages ou de flux de données en continu à un navigateur client. Elle a été conçue pour améliorer nativement le support du streaming de données multi-navigateurs à l'aide d'une API Javascript **EventSource**, par laquelle un client demande une URL particulière afin de recevoir un flux d'événements (source Wikipedia).



Les navigateurs s'inscrivent auprès du serveur pour être tenu informés par le serveur à chaque fois qu'un nouvel événement (nouvel état) se produira.

À chaque fois qu'un état change sur le serveur, l'événement correspondant est envoyé aux navigateurs inscrits sur cet événement sans que le navigateur n'ait besoin de faire la moindre demande.