Description des échanges entre les divers acteurs du système
Air Conditioner Solar Management System

Table des matières

1. Description SysML du système : focus communication

1.1. Composition du système

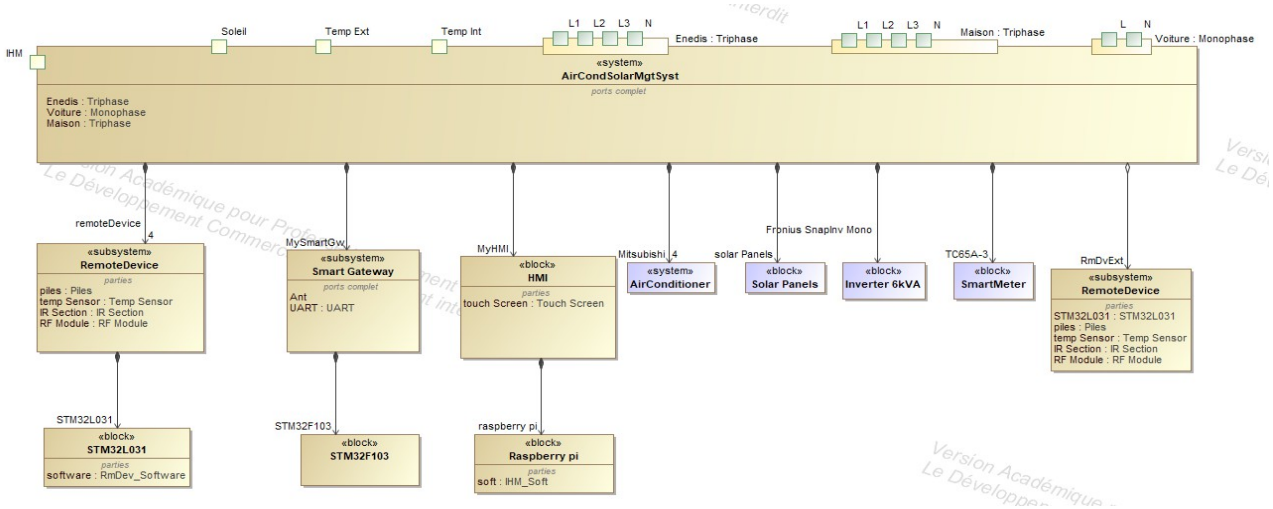


Fig1 *: BDD système*

1.2. Description des éléments du système

Air Conditioner : Les climatiseurs *Mitsubishi* air-air monosplit. Il y en 4 (salon, salle à manger, entrée, couloir).

Remote Device : Ce sont les télécommandes des climatiseurs joignables en 433MHz FSK (module RT606) par une pile « propriétaire » ***FSKStack*** (*Remote Device* et *SmartGateway*). Le protocole est documenté, il fonctionne sur STM32F103 (passerelle intelligente) et STM32L041 (*Remote Device*). La pile fonctionne un peu à la façon de Xbee (My , destination adress, broadcast). Chaque *remote device* commande son climatiseur par infra-rouge (comme un télécommande classique)

Smart Gateway : passerelle entre le réseau des *Remote Device* et l'IHM. La liaison se fait avec ce dernier via une liaison UART classique. Le protocole de communication qui gère cette liaison est ***UARTStack***. Les informations reçues par la passerelle depuis les divers *remote Devices* sont centralisées puis traitée localement, puis sont transmises vers l'*IHM*. Le traitement fait par la *Smart Gateway* consiste en la stratégie de commande des climatiseurs. C'est ce qui justifie le « *Smart* » dans la dénomination... toute l'intelligence du système est dans ce module à base de STM32F103.

IHM : Unité basée sur une *Raspberry pi 2* et un écran tactile. Elle communique :

- vers la passerelle intelligente en UART pour recevoir l'état des climatiseurs et transmettre des nouvelles informations (puissance disponibles, informations tempo, paramètres utilisateur....)
- vers l'onduleur en HTTP via des requêtes locale (*HMI*-box-onduleur),
- vers le web en HTTP pour recevoir les informations EDF tempo, pour recevoir les infos météo ?
- Avec le user via le touch screen pour paramétrer le système, pour visualiser en temps réels les flux d'énergie de la maison, de l'onduleur, de la voiture ?

L'ensemble PV + onduleur + Smart Meter : c'est évidemment le système de production d'énergie PV. Les informations utiles se trouvent au niveau du *SmartMeter* et sont utilisables via l'onduleur et son lien Ethernet, en utilisant l'API fournie par Fronius (voir document dans le répertoire RS585)

1.3. Interactions entre les éléments du système

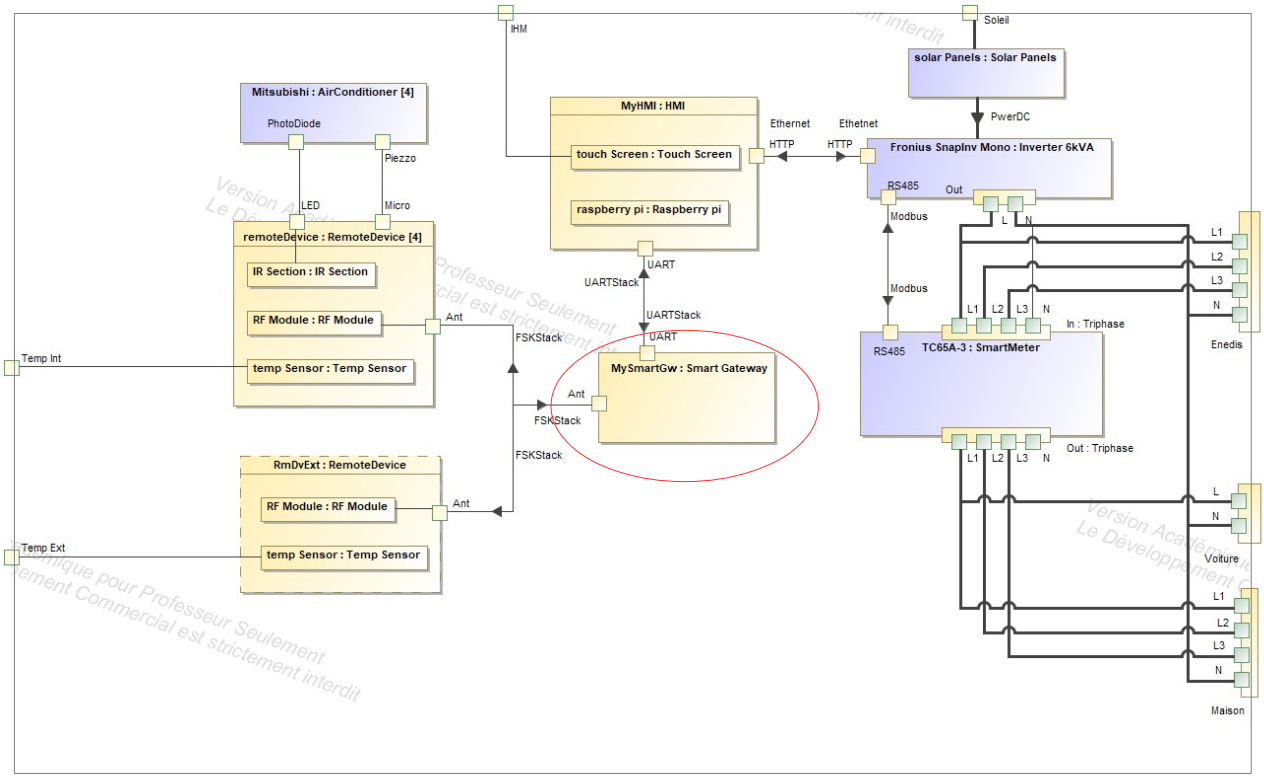


Fig 2 *: IBD système*

1.4. Grands principes de fonctionnement

L'intelligence du système est confiée à la passerelle, la *SmartGateway*. Elle échange à la fois avec l'*HMI* et les *RemoteDevices*.

Ces derniers étant alimentés sur piles, ne peuvent pas être en écoute permanente. Ils sont endormis sur de très longue période : typiquement 30mn de repos pour quelques secondes d'activités. Cela permet d'économiser les piles.

→ les *RemoteDevices* sont donc obligatoirement à l'initiative des requêtes lors des transactions et c'est la *SmartGateway* qui est en écoute permanente (alimentée sur secteur)

L'*HMI* est faite sur une Raspberry pi. La communication série en python se fait en scrutatif (en tout cas j'ai pas trouvé le mode événementiel!). Par ailleurs le *STM32* gère parfaitement cela.

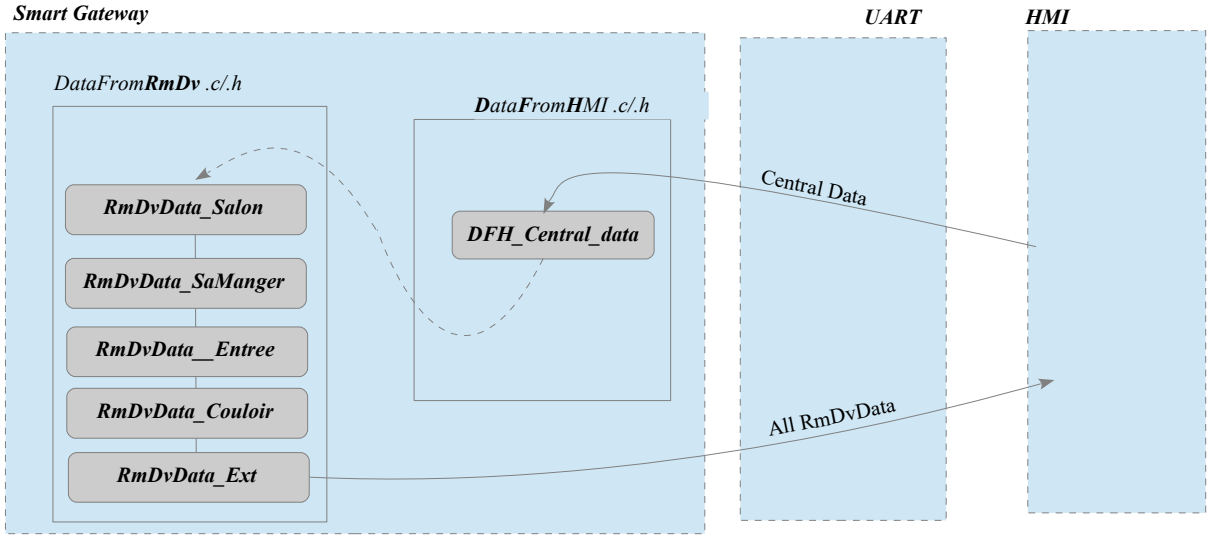
→ la *HMI* elle aussi sera à l'initiative des requêtes pour fournir les **paramètres** permettant d'élaborer la stratégie de commande des climatiseurs (puissance solaire disponible, couleur jour tempo, mode vacances, auto...), en en retour récupérer les températures à afficher pour information à l'utilisateur (voir 1.4).

La *SmartGateway* doit pouvoir répondre instantanément à chaque *RemoteDevice*. Pour cela la nouvelle consigne de température et le prochain rendez-vous (date du prochain réveil) doivent être produit très rapidement.

→ la *SmartGateway* doit disposer en local de toutes les informations nécessaires, les dernières qui auront été communiquées par l'*HMI*. En effet, la *SmartGateway* ne peut pas interroger l'*HMI* à la volée (elle n'est pas à l'initiative des requêtes et ça prend trop de temps).

1.5. Architecture montrant le flux de donnée IHM-SGw

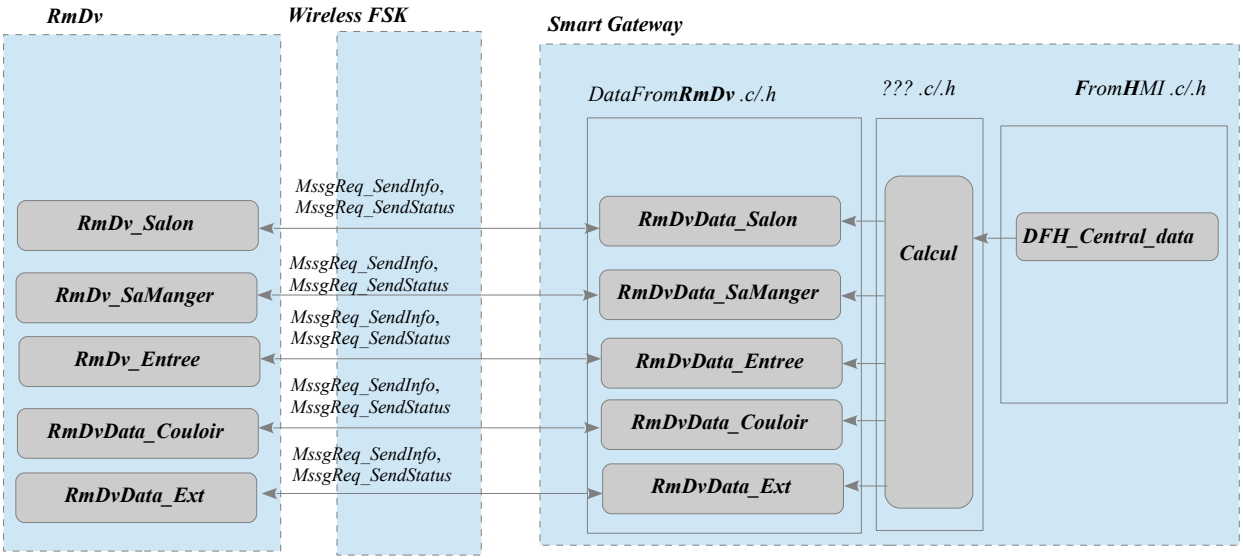
La *HMI* produit la requête *HMI2SGw_DataReq*. Elle Envoie les paramètres, et reçoit les données des *RmDv*.



NB : les pointillés indique un séquencement, non un flux. Les flux sont en traits peins.

1.6. Architecture montrant le flux de données RmDv - SGw

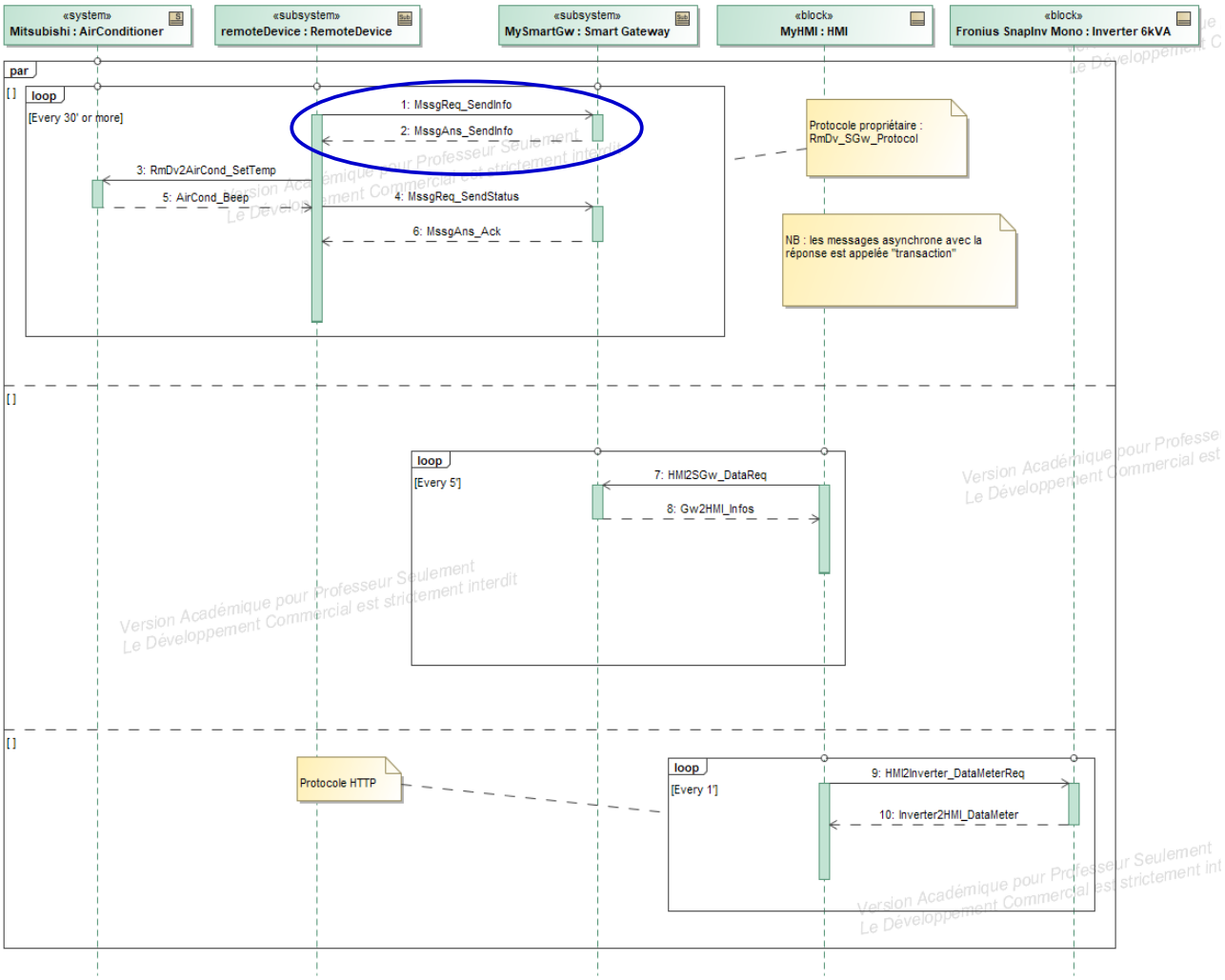
Chacune des *RmDv* émet deux requêtes à intervalles réguliers: *MssgReq_SendInfo*, *MssgReq_SendStatus*. Ces requêtes alimentent les champs de données correspondant.



NB : Les 5 variables *RmDvData_xx* sont accessibles de manière asynchrone par un *RmDv* ou bien par l'*IHM*. Il ne peut pas y avoir de conflit, car, comme nous allons le voir, les accès en écriture de ces variables sont exclusives.

Lorsqu'une transaction *RmDv* est initié les calculs (rapides) nécessaires sont faits (prochaine valeur de consigne et prochain délai) via le module ???.c/.h

1.7. Séquencement des communications



On observe 3 processus de communication qui se déroulent en même temps, à des périodicités différentes.

Transaction : Correspond à un message asynchrone associé à sa réponse (voir ci-dessus, bulle bleue). Il existe donc 3 types de transaction :

- Entre un *RmDv* et la *SmartGateway* : utilisation du protocole ***RmDv_SGw_Protocol*** qui s'appuie sur la pile ***FSKStack*** ,
- Entre la *SmartGateway* et l' *HMI* : utilisation de la pile, ***UARTStack*** . Le message étant unique, pas besoin de couche de protocole,
- Entre le *HMI* et l'onduleur : utilisation de requête ***HTTP***.

Elles sont toutes basées sur le même principe : une requête, une réponse.

On note que deux acteurs sont maîtres (à l'initiative des requêtes des transactions) :

- les *RmDv* (4 associés aux climatiseurs et 1 pour la température extérieure)
- le *HMI* (Raspberry pi, programme Phyton)

1.8. Algorithme de gestion des la communication dans la SGw

Le point de gestion délicat est donc la *SmartGateway* qui reçoit jusqu'à 6 messages asynchrones. Le choix qui est fait pour cette gestion est celle d'une boucle sans fin jamais bloquante qui tour à tour srute des indicateurs d'arrivée de message (5 pour la pile ***FSKStack***, et 1 pour l'*IHM,* ***UARTStack***).

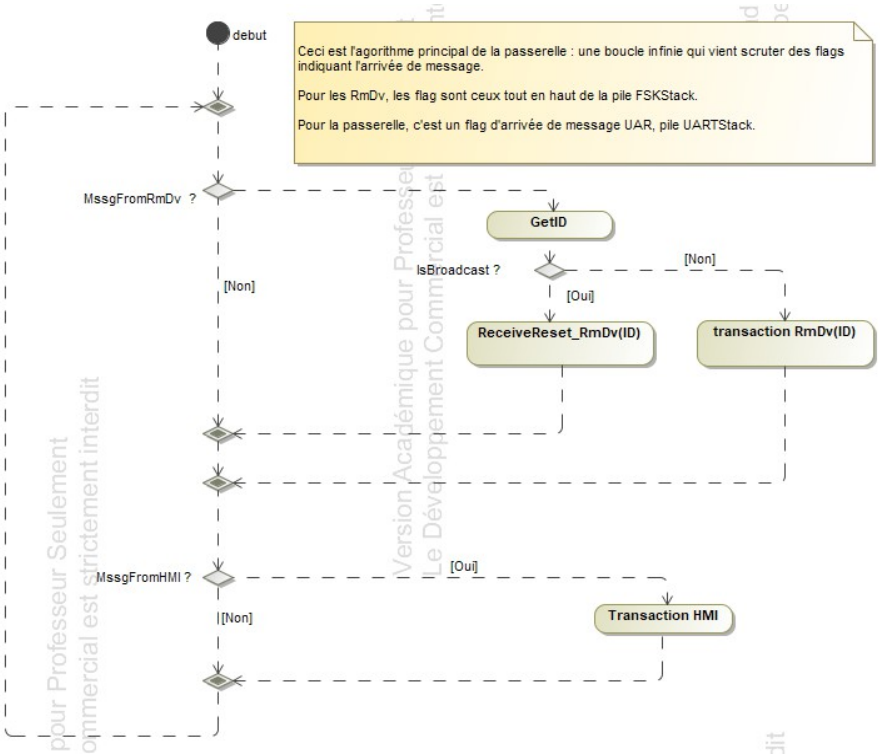


Fig 4*: algorithme principal gérant la communication globale du système à bord de la SGw*

De cette manière, on ne se pose pas la question d'accès multiple à une variable partagée par un *RmDv* et la *HMI* puisque dès qu'une transaction démarre, elle se finit (au pire un timeout permet d'interrompre une transaction infructueuse). Du coup les autres requêtes susceptibles d'arriver sont simplement mises en attente d'être traitée.

2. Transaction IHM - SGw

2.1. Présentation des modes de fonctionnement

Mode :	***Jour HP / Nuit HC (6h00 / 22h00)***	***Tempo Rouge*** *paramètres...*	***Tempo Blanc*** *paramètres...*	***Tempo Bleu*** *paramètres...*
Auto Gère la puissance de chauffe en fonction de la puissance disponible, avec un minimum de température mesurée	HP	θ_{min} = 16°C Montée en température si excédent de puissance est supérieur à 1000W. Retour à la consigne si excédent de puissance est inférieur à 500W. Prio Clim : Entrée, Salon, Salle à manger, Couloir	θ_{min} = 17°C Montée en température si excédent de puissance est supérieur à 500W. Retour à la consigne si excédent de puissance est inférieur à 0W. Prio Clim : Salle à manger, Entrée, Salon Couloir	θ_{min} = 19°C Montée en température si excédent de puissance est supérieur à 500W. Retour à la consigne si excédent de puissance est inférieur à 0W. Prio Clim : Salle à manger, Entrée, Salon Couloir
	HC	Si $\theta_{ext} > \theta_{ext\ min}$ =10 °C Climatiseur Off Sinon θ_{min} = 18°C	Si $\theta_{ext} > \theta_{ext\ min}$ =10 °C Climatiseur Off Sinon θ_{min} = 18°C	Si $\theta_{ext} > \theta_{ext\ min}$ =10 °C Climatiseur Off Sinon θ_{min} = 18°C
Programmé Gère les climatiseurs en fonction de l'heure et des jours.		6h00 θ = 18°C 15h00 θ = 20°C 8h00 θ = 20°C 18h00 θ = 19°C 10h00 θ = 20°C 22h00 θ = 19°C pour chacune des 4 climatisations	6h00 θ = 18°C 15h00 θ = 20°C 8h00 θ = 20°C 18h00 θ = 19°C 10h00 θ = 20°C 22h00 θ = 19°C pour chacune des 4 climatisations	6h00 θ = 18°C 15h00 θ = 20°C 8h00 θ = 20°C 18h00 θ = 19°C 10h00 θ = 20°C 22h00 θ = 19°C pour chacune des 4 climatisations
Vacances Date d'arrivée : Température Arrivée Reprise en mode automatique ou programmé 1 jours avant		θ_{HG} = 12°C	θ_{HG} = 12°C	θ_{HG} = 12°C

Tab 1 *: Modes de fonctionnement*

2.2. La variable DFH_CentralData

Ci-dessous on trouve la variable structurée *CentralData* qui est définie à la fois au niveau de la *SGw* et de *l'IHM*.

CentralData.Stamp : 6 short int (12 bytes)

Nom Variable	**LastHMITimeStamp**	
	type	Valeurs
	TimeStampTypedef	HH:mn:ss:D:Month:Year

***Tab 2** : les champs de la variable structurée CentralData*

Le champ donne l'horodatage correspondant à la dernière mise à jour de la structure par l'IHM.

CentralData.Mode: 1 int (4 bytes)

Nom Variable	**Mode**	
	type	Valeurs
	Enum	HMI_Mode_Off / HMI_Mode_Auto / HMI_Mode_Program / HMI_Mode_Hollidays /

Le champ qui indique quel est le mode demandé par la *HMI,* mode à exécuter en temps réel par la *SGw*.

CentralData.Auto (20 bytes)

Nom Variable	**Auto** structure	
	Champs / type	Valeurs
	TempMinExt / float	11
	PowExcessStart / float	500
	PowExcessStop / float	250
	char ClimPrio[4] / char	1 / 2 / 3 / 4
	SetTempMinHC / Short int	18
	SetTempMinHP / Short int	19

Ce champ structuré contient les paramètres utiles pour le mode automatique, paramètres remplis par l'IHM.

CentralData.Program (24 bytes)

Nom Variable	**Program** structure	
	Champs / type	Valeurs
	Temperature_6h[4] / char	Températures à 6 h pour chaque clim
	Temperature_8h[4] / char	
	Temperature_10h[4] / char	
	Temperature_15h[4] / char	
	Temperature_18h[4] / char	
	Temperature_23h[4] / char	

Ce champ structuré contient les paramètres utiles pour le mode programmation, paramètres remplis par l'IHM. Ce sont les températures attendus pour chaque des heures spécifiées et pour chaque climatiseur.

CentralData.Hollidays (16 bytes)

Nom Variable	**Hollidays** structure	
	Champs / type	Valeurs
	Mode / char	(char)HMI_Mode_Auto=0xA1
	TempHG_bleu / char	17
	TempHG_blanc / char	17
	TempHG_rouge / char	17
	ArrivalDate / TimeStampTypedef	HH:mn:ss:D:Month:Year

Cette structure correspond aux paramètres du mode vacances. Le premier paramètre est le mode de reprise 1 jour avant la date d'arrivée.

CentralData.OptPowData (16 bytes)

Nom Variable	**Option** structure	
	Champs / type	Valeurs
	PowExcess / float	Positif si excès de puissance
	PowL1_Home / float	Forcément positif
	PowInverter / float	Forcément positif. Si négatif → inverter inactif.
	ReapeatBeep / char	0 / 1 : répétition consigne clim si identique à la précédente
	PrioVE / char	0 / 1 : indique qu'un VE est connecté, avec priorité de charge
	CoupureNuit / char	0 / 1 : force la coupure des climatisation la nuit
	CouleurTempo / char	0 indéfini / 1 bleu / 2 blanc / 3 rouge

Paramètres, toujours remplis par l'IHM, qui correspondent au données de puissance qui vont permettre à la SGw d'opérer sa stratégie.

Total octets pour toute la variable CentralData : 92 octets

Calcul de la durée de transfert :

En comptant large, 100 bytes , T = NbOctets x Nbbits x 1/Débit = 1000 / Débit

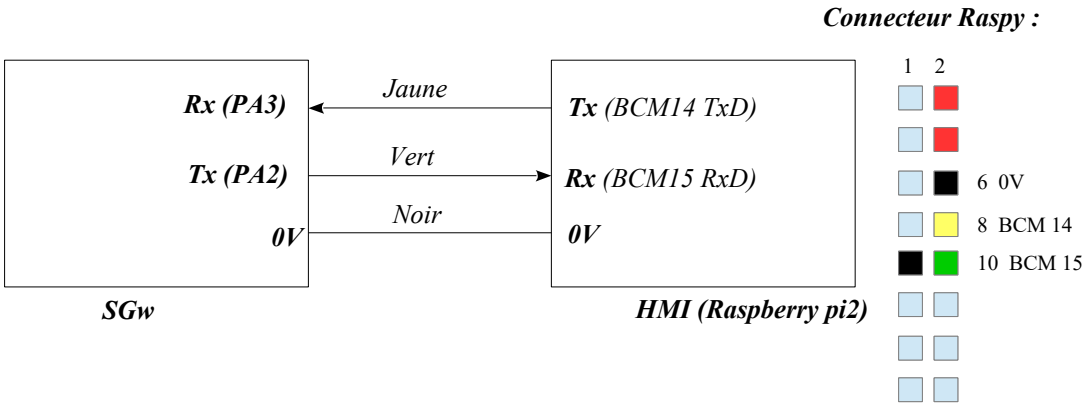
- à 9600 bd , T=104 ms.
- à 38400 bd , T=26ms.
- à 115200 bd , T = 8.6ms

→ prio maxi sur cette communication. Tester la rapidité de l'algo. A 115200 Bds, on a 8.6µs pour traiter le logiciel associé.

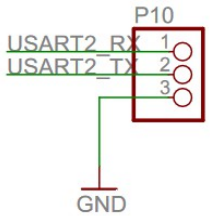
→ L'ensemble de la donnée *CentralData* est mise à jour d'un coup par l'IHM

2.3. Connectique IHM (Raspberry pi 2) avec SGw (STM32F103)

La liaison est une liaison série asynchrone classique. Pas de bit de parité, débit 9600 bds (pour l'instant, accélerer …)



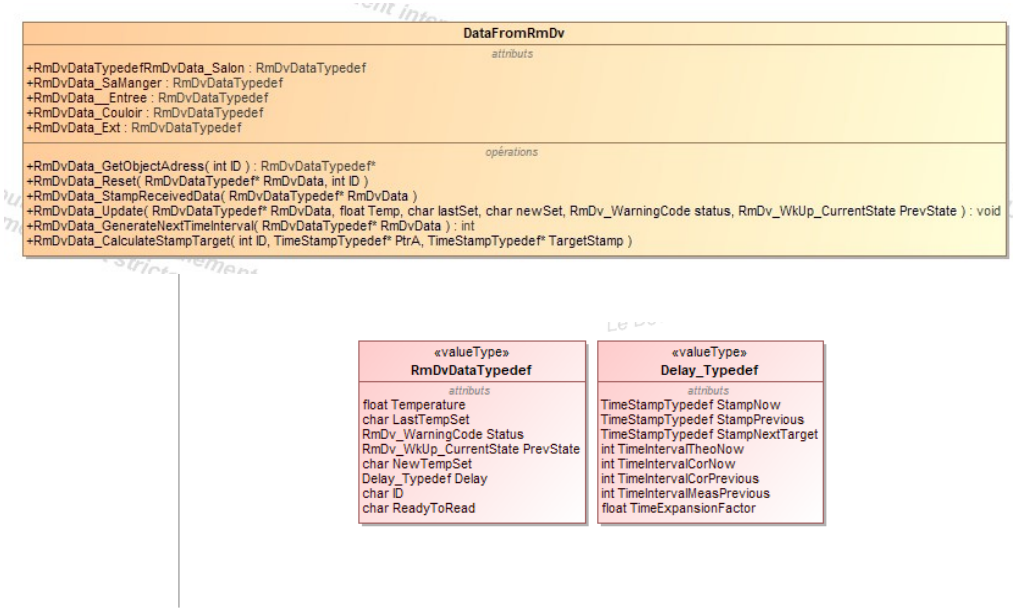
Connecteur carte réseau INSA

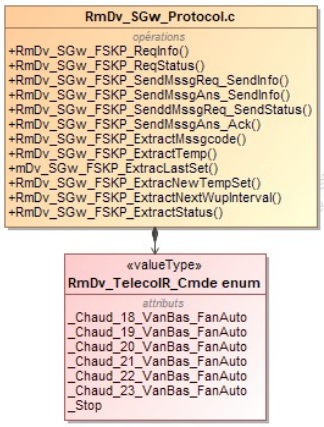


3. Transaction RmDv_i – SGw

On renvoie le lecteur au document *Protocoles.pdf* pour en savoir plus sur le détail de transaction. L'objectif ici est simplement de détailler la variable *RmDv_Data_xx*.

Ces 5 variables (salon, salle à manger, entrée, couloir, extérieur) sont créées dans le module *DataFromRmDv.c/.h* :





Le module *RmDv_SGw_Protocol.c/.h* est utilisé par le module DataFromRmDv.c/.h pour pouvoir accéder à l'énumération *RmDv_TelecoIR_Cmde*. Cette dernière liste les commandes possibles communiquées à un *RmDv*.

3.1. La variable RmDv_Data_xx

	RmDv_Data	
	Champs / type	Valeurs
Réception *RmDv → SGw* *SGw → IHM*	Température / float	Température exprimée en °C
	LastTempSet / RmDv_TelecoIR_Cmde	Dernière commande que le *RmDv* a traité
	RmDv_WarningCode/ Status	Message d'erreur/warning du *RmDv*
	RmDv_WkUp_CurrentState / PrevState	Indique l'état du *RmDv* lors de la précédente activité. Si tout c'est bien passé le contenu vaut `WarningMssg.` Dans le cas contraire, la variable contient l'état de la FSM du *RmDv* au moment du plantage.
RmDv ← SGw	RmDv_TelecoIR_Cmde/ NewTempSet	Commande à envoyer au *RmDv* concerné. Elle est déterminée par la *SGw* en fonction des derniers paramètres transmis par l'*IHM*.
	Delay_Typedef / Delay	Structure de donnée gérant le temps (voir ci après). Principalement, intervalle de temps corrigé pour le prochain réveil (en seconde, int 32 bits) à envoyer au *RmDv*. Il est calculé par le *SGw* dès réception.
Variable d'état	ID / char	Identifiant du *RmDv*. Unique. 0xD1 à 0xD5.
	ReadyToRead / char	Inutilisé pour linstant...

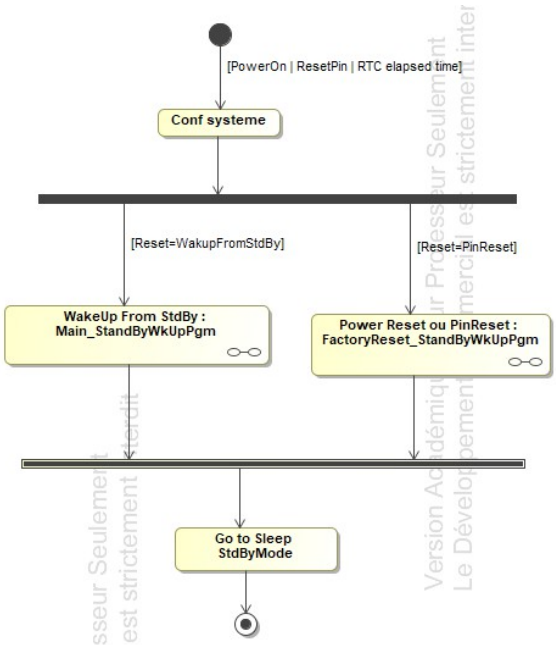
Le champ *Delay* est particulier. C'est une structure regroupant plusieurs champs qui vont permettre de déterminer précisément l'intervalle de temps pour la prochaine transaction.
Tout le principe, notamment la correction des erreurs de timing des *RmDv* est explicité dans le document *GestionTemps.pdf*.

La structure *Delay_Typedef* est définie ci après :

	Delay_Typedef	
	Champs / type	Valeurs
	StampNow / TimeStampTypedef	C'est l'horodatage lors de la réception de la première requête du *RmDv* considéré.
	StampPrevious / TimeStampTypedef	L'horodatage lors de la précédente réception. Permettra de déterminer la durée réelle du précédent intervalle.
	StampNextTarget / TimeStampTypedef	C'est l'horodatage cible, celui de la prochaine transaction.
	TimeIntervalTheoNow / int	C'est l'intervalle de temps théorique qui sépare le *StampNow* et le *StampNextTarget* calculé maintenant.
	TimeIntervalCorNow / int	Intervalle de temps corrigé calculé maintenant.
	TimeIntervalCorPrevious / int	Intervalle de temps corrigé calculé précédemment, celui transmis au *RmDv* lors de la transaction précédente. Permettra de comparé le temps voulu et le temps réellement écoulé.
	TimeIntervalMeasPrevious / int	Intervalle de temps précédent mesuré.
	TimeExpansionFactor / float	Facteur de correction basé sur la comparaison (rapport) des deux champs précédents.

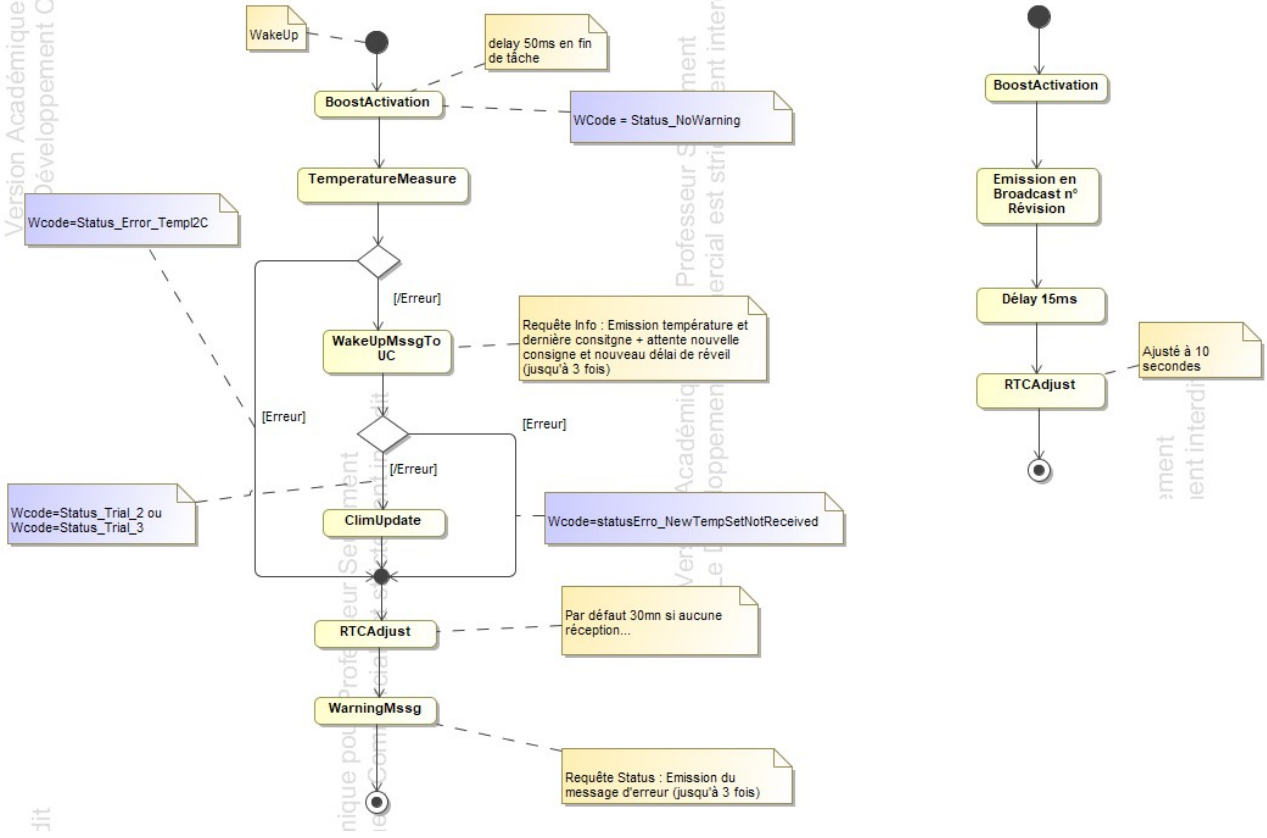
3.2. Algorithme du RmDv

***Main** program*



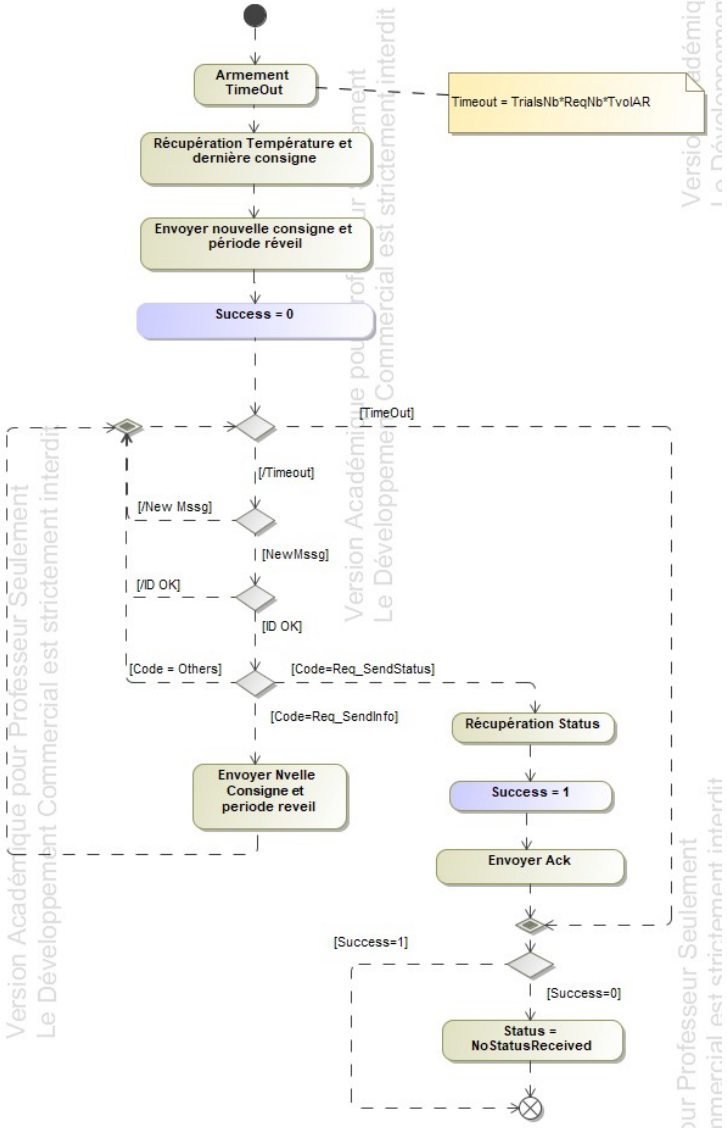
***Main_StandByWkUpPgm** function*

***FactoryReset_StandByWkUpPgm** function*



3.3. Algorithme transaction RmDv i au niveau SGw

On se reportera à *1.8 Algorithme de gestion des la communication dans la SGw*. On décrit ci-après uniquement la partie transaction *RmDv i* avec *SGw*.



3.4. Détail du timing de la transaction

Chacune des transactions doit avoir un maximum de chance d'aboutir. Côté *RmDv*, on cherchera à minimiser les redondances pour économiser les piles.

Durée d'une transaction FSK

Voici une trame FSK typique qui est transmise :

```
|0xFF|0xFF|0xFF|0xFF|'#'|'#'|'#'|'#'|'#'|Len|My|Dest@|Code|byte0|byte1|byte2|byte3|CheckSum|
```

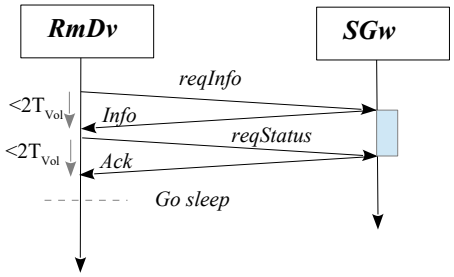
Dans cet exemple, la payload (en bleu) compte 5 octets. En tout il en faut 18. Nous prenons une marge de 40 octets maximum à transmettre, soit encore 400 bits (en incluant *start* et *stop*). La durée totale de transmission est donc $T_{Vol} = \frac{400}{R(baud)}$

Le calcul se fait automatiquement dans un fichier de configuration .h (*GLOBAL_RmDv.h*) pour le *remote device* en fonction du débit de l'UART associé au RT606 (module FSK).

Stratégie :

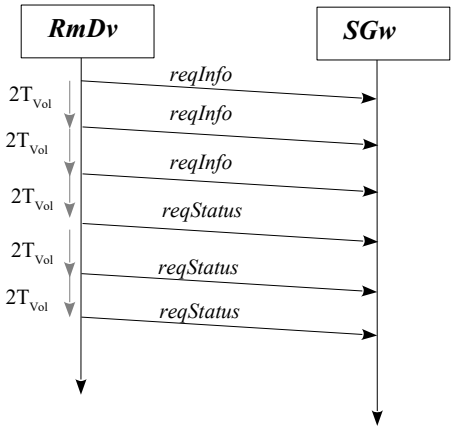
Le *RmDv* envoie une requête et attend la réponse à concurrence du *timeout*. On se donne par défaut 3 essais (*TrialNb*).

Idéal



Requête info n'arrive jamais au Sgw

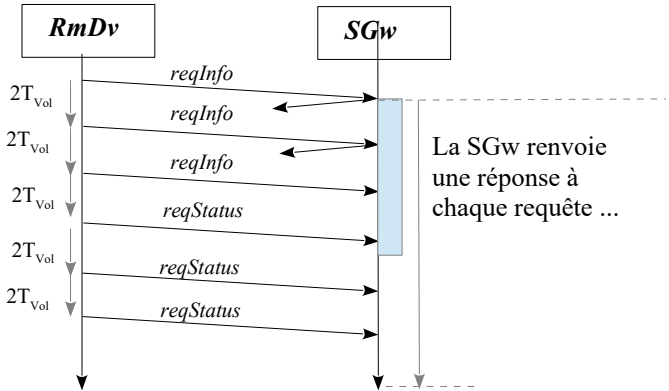
3 tentatives ...



Aucune activité côté SGw puisque aucun signal reçu

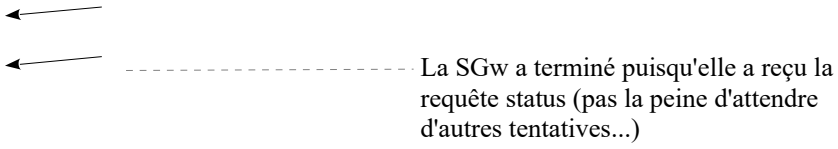
Requête info arrive au Sgw mais pas de retours

3 tentatives ...



La SGw renvoie une réponse à chaque requête ...

La SGw a détecté la requête et part dans son algorithme avec timeout



$2T_{Vol}$ x TrialNb x Req Nb
$= 2T_{Vol}$ x 6