User Guide Protocole utilisant stack FSK

RmDv_SGw_Protocol.c/.h

Solar Management System for air conditioners

Table des matières

1.Présentation....2
2.Les fonctions du module protocole RmDv_SGw_Protocol....4
2.1.Les fonctions simples d'émission....4
2.2.Les fonctions d'extraction....4
2.3.Les fonctions requête....4
2.4.Les codes de message, et les codes de warning....5

1. Présentation

Le protocole propriétaire *RmDv_SGw* constitue le formatage des trames échangées entre les *Remote Device* et la *Gateway* du système *Air Conditioner Solar Management*. Il s'appuie entièrement sur la pile propriétaire *FSK_Stack.h*. Le protocole s'appuie aussi sur le service qui gère les timeout, *TimeManagement_RmDv.h* (si on se situe dans le *Remote Device*) ou bien *TimeManagement.h* (si on se situe dans la *Smart GateWay*).

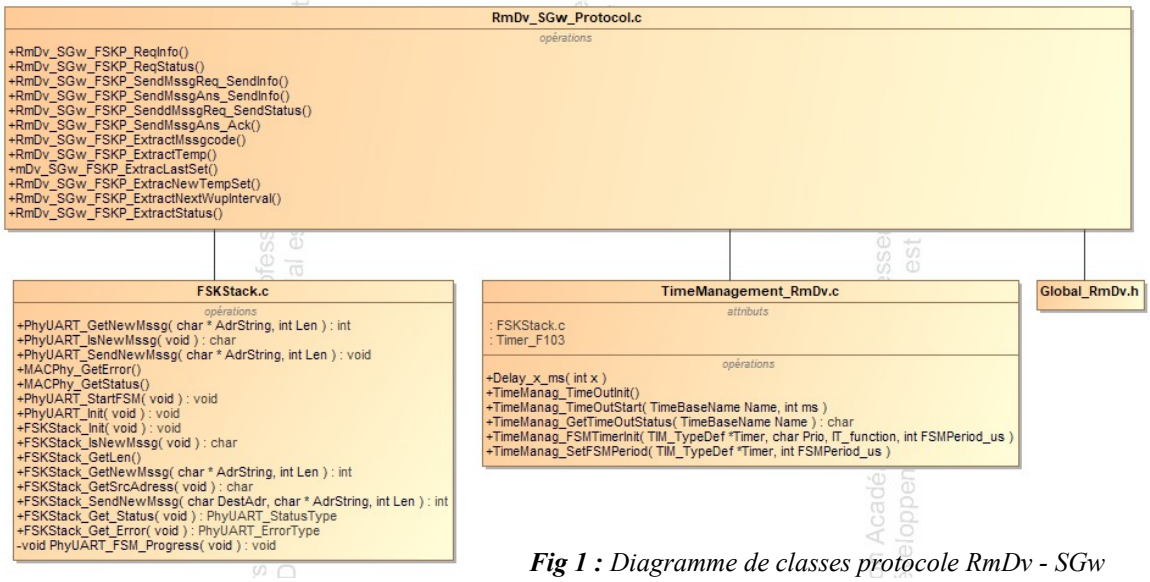

Fig 1 : *Diagramme de classes protocole RmDv - SGw*

Le principe est basé sur la communication suivante, décrite sous forme de diagramme de séquence :

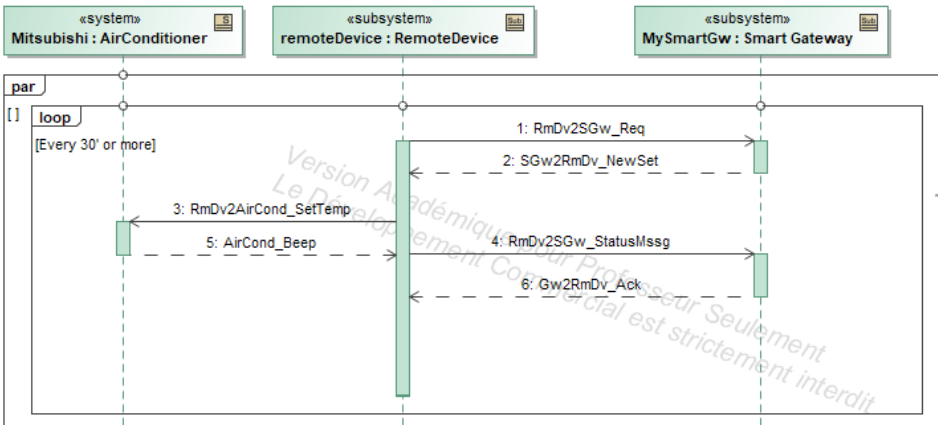

Fig 2 : *Diagramme de séquences entre Remote Device et Smart Gateway*

(se reporter si besoin au document *System_CommunucationPart.odt*.).

Les variables échangées dans ces requêtes sont décrites ci-dessous :

	Champs / type	Valeurs
Requête *Info* *code Req_SendInfo*	**RmDv → SGw**	
	Temperature / float	
	LastTempSet / char	Valeur entière de température , 0 = stop
Réponse *Info* *code Ans_SendInfo*	**SGw → RmDv**	
	NewTempSet / char	Valeur entière de température , 0 = stop
	NextTimeInterval_sec / unsigned short int	Max = 65535 sec = 18h
Requête *Status code Req_SendStatus*	**RmDv → SGw**	
	Status / *RmDv_WarningCode*	Parmi la liste enum *RmDv_WarningCode*
Réponse *Status code Ans_Ack*	**SGw → RmDv**	
	AckToRmDv	0xAB code Acknoledge

Tab 1 : *échanges et variables mises en jeu dans l'échange*

Le principe général est le suivant :

- Basé sur la notion de requête côté *RmDv* et de réponse côté *SGw*,
- une requête consiste en l'émission de données avec un code spécifique (comme on peut le voir dans la première colonne du tableau) et en l'attente d'une réponse,
- le code correspond au premier octet de donnée transmise à la couche *FSK_Stack*.
- l'attente de réponse est conditionnée par un timout,
- à l'issue d'un timeout, une nouvelle tentative peut être lancée (jusqu'à 10 essais, 3 est le réglage par défaut),
- la requête et la réponse se font via une adresse source et destination (propre à la couche *FSK_Stack*).

Format typique d'un message protocole :
`|Code|byte0|byte1|byte2|byte3|` ici la longueur à spécifier pour la couche FSK est 5 (nombre d'octets + code). On fournira l'adresse de destination également.

Format typique d'une trame FSK_Stack :
`|0xFF|0xFF|0xFF|0xFF|'#'|'#'|'#'|'#'|'#'|Len|My|Dest@|Code|byte0|byte1|byte2|byte3|CheckSum|`
Rappel :

- la suite de 0xFF est le préambule de synchro utile pour le module RT606,
- la suite de # est « l'identifiant réseau » qui permet de filtrer nos trame vis à vis des agressions extérieures à 433MHz

Le header de la couche protocole, donc `RmDv_SGw_Protocol.h`, contient les définitions suivantes (parmi d'autres) :

```
#define RMDV_ChronoName Chrono_Protocole
#define RMDV_TimeOutReq   TimeOutReq
#define RMDV_StatusReqTrialNb  StatusReqTrialNb
#define RMDV_InfoReqTrialNb StatusReqTrialNb
```

Les équivalences en seconde partie d'une définition, sont définies dans le fichier *global_RmDv.h*. qui contient tout les généralités propre au *remote device*.

- Le premier #define : spécifie quel chronomètre on utilise pour gérer le timeout des requêtes,
- le second indique la valeur du timeout (en ms),
- les deux derniers indiquent le nombre d'essais pour chacune des deux requêtes possibles.

2. Les fonctions du module protocole *RmDv_SGw_Protocol*

2.1. Les fonctions simples d'émission

Il s'agit des fonctions des 4 fonctions qui commencent par **`RmDv_SGw_FSKP_Send`**:

```
void RmDv_SGw_FSKP_SendMssgReq_SendInfo(char DestAdr, float Temp, char LastSet);
void RmDv_SGw_FSKP_SendMssgAns_SendInfo(char DestAdr, char NewSet, unsigned short int
NextWupInterval);
void RmDv_SGw_FSKP_SenddMssgReq_SendStatus(char DestAdr,  char Status);
void RmDv_SGw_FSKP_SendMssgAns_Ack(char DestAdr);
```

Ces fonctions sont basiques, elles se chargent juste d'émettre une trame.

2.2. Les fonctions d'extraction

Lorsqu'une trame est arrivée, c'est à dire que son champ destination est égal au *my* (on le sait via une fonction de la couche *FSK_Stack*), on récupère la trame protocole, puis on extrait les champs de données utiles.
Il existe 6 fonctions d'extraction qui commencent toutes par : **`RmDv_SGw_FSKP_Extract :`**

```
MssgCode RmDv_SGw_FSKP_ExtractMssgcode(char * MssgTempStr);
float RmDv_SGw_FSKP_ExtractTemp(char * MssgTempStr);
char RmDv_SGw_FSKP_ExtracLastSet(char * MssgTempStr);
char  RmDv_SGw_FSKP_ExtracNewTempSet(char * MssgTempStr);
unsigned short int  RmDv_SGw_FSKP_ExtractNextWupInterval(char * MssgTempStr);
RmDv_WarningCode   RmDv_SGw_FSKP_ExtractStatus(char * MssgTempStr);
```

2.3. Les fonctions requête

Il existe deux requêtes, il y a donc deux fonctions. Chacune d'elles gère l'émission et la réception. Elles se chargent donc de remonter les données nécessaires. Elles sont bloquantes avec un timeout :

chacune des fonctions s'accompagne d'un paramètre structuré qui contient toutes les données de statut, d'émission et de réception correspondant à la requête.

2.3.1. *La requête info*

```
typedef struct
{
	char DestAdr;		/* Adresse de destinataire*/
	float Temp;  /* Température : out */
	char LastSet;		/* dernière consigne : out */
	int TimeOut_ms;		/* durée maximale pour un essai */
	char TrialMaxNb;	/* nbre maximum de tentatives autorisées */
	char TrialActualNb;/* nbre effectif de tentatives */
	char success;		/* 1 : la requête est OK, 0 sinon*/
	char NewSet; /* nouvelle consigne de température : in*/
	int NextInterval;	/* nombre de secondes à attendre pour
				prochain réveil : in*/

}RmDv_SGw_FSKP_ReqInfoTypedef;

void RmDv_SGw_FSKP_ReqInfo(RmDv_SGw_FSKP_ReqInfoTypedef* Req);
```

2.3.2. *La requête status*

```
typedef struct
{
	char DestAdr;			/* Adresse de destinataire*/
	RmDv_WarningCode Status;	/* le status de l'échange côté RmDv : out*/
	int TimeOut_ms;			/* durée maximale pour un essai */
	char TrialMaxNb;		/* nbre maximum de tentatives autorisées */
	char TrialActualNb;		/* nbre effectif de tentatives */
	char success;			/* 1 : la requête est OK, 0 sinon*/
}RmDv_SGw_FSKP_ReqStatusTypedef;

void RmDv_SGw_FSKP_ReqStatus(RmDv_SGw_FSKP_ReqStatusTypedef* Req);
```

2.4. Les codes de message, et les codes de warning

2.4.1. *Les codes de messages*

Suivant le document décrivant les échanges (*System_CommunucationPart.odt*), les messages de la table 1 ont été établis. Les codes associés sont établis dans le fichier header du protocole:

```
	MssgReq_SendInfo = 0x50,
	MssgAns_SendInfo = 0x51,
	MssgReq_SendStatus = 0x52,
	MssgAns_Ack = 0x53,
```

D'autres codes apparaissent mais sont obsolètes et répondaient à d'autres exigences protocolaires.

2.4.2. Les codes warning

Lorsque le *remote device* enchaîne ses actions il passe forcément par un état qui donne des informations sur ce qui s'est passé. Dans cette phase, le *remote device* envoie un code warning, qui est en fait un status. Cela permet à la passerelle de savoir s'il y a un problème au niveau du composant à distance.

La machine à états qui régit le fonctionnement du RmDv est le suivant :

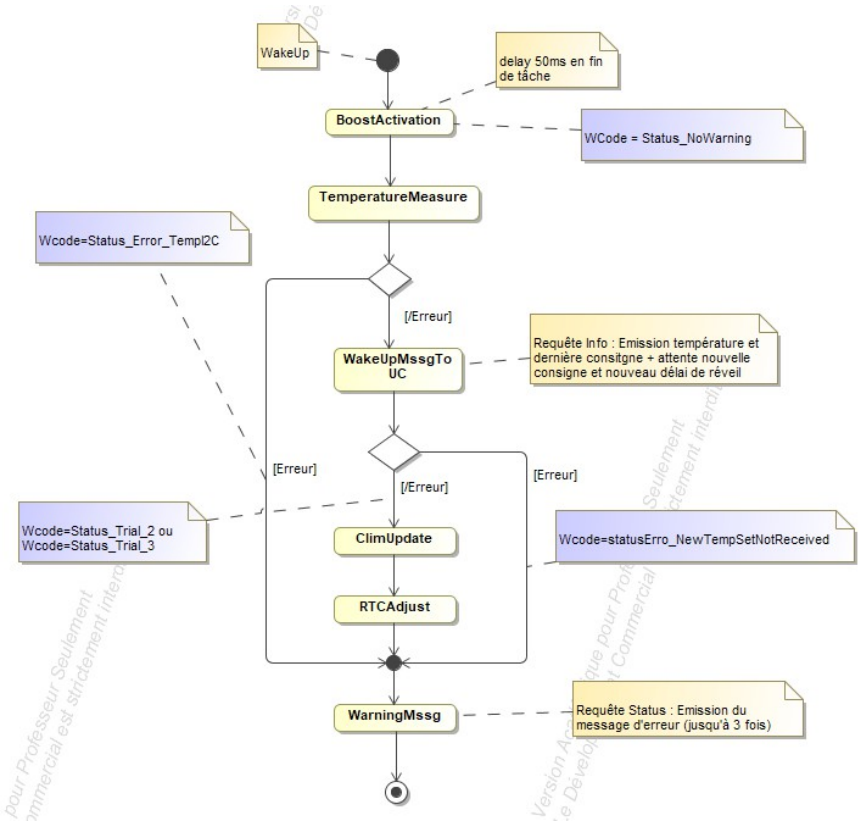

Fig 3 : *State machine du Remote Device*

La figure 3 montre au passage les divers statuts possibles (Warning Code, en violet). Ils sont résumés dans l'API du protocole (ci-dessous). Le message idéal , *NoWarning*, sera émis si tout s'est bien passé, notamment lorsque la requête info a été traitée du premier coup (premier essais).

```
/* Liste des warnings, fonctionnement normal*/
typedef enum {
        Status_NoWarning=1,
        Status_Trial_2=2,
        Status_Trial_3=3,
        Status_Trial_4=4,
        Status_Trial_5=5,
        Status_Trial_6=6,
        Status_Trial_7=7,
        Status_Trial_8=8,
        Status_Trial_9=9,
        Status_Trial_10=10,

        Status_Error_TempI2C=20,
        Status_Error_NewTempSetNotReceived=21,
        Status_NoStatusReceived=22,
}RmDv_WarningCode;
```