1 Fonctionnement

La station de base (que l'on programme) doit être capable, lorsqu'elle reçoit une trame spéciale (beacon), de renvoyer les données qu'elle a collecté. Ces données sont :

1. Événements (par exemple un mouvement sur le joystick)
2. État mesuré (par exemple l'accéléromètre embarqué)

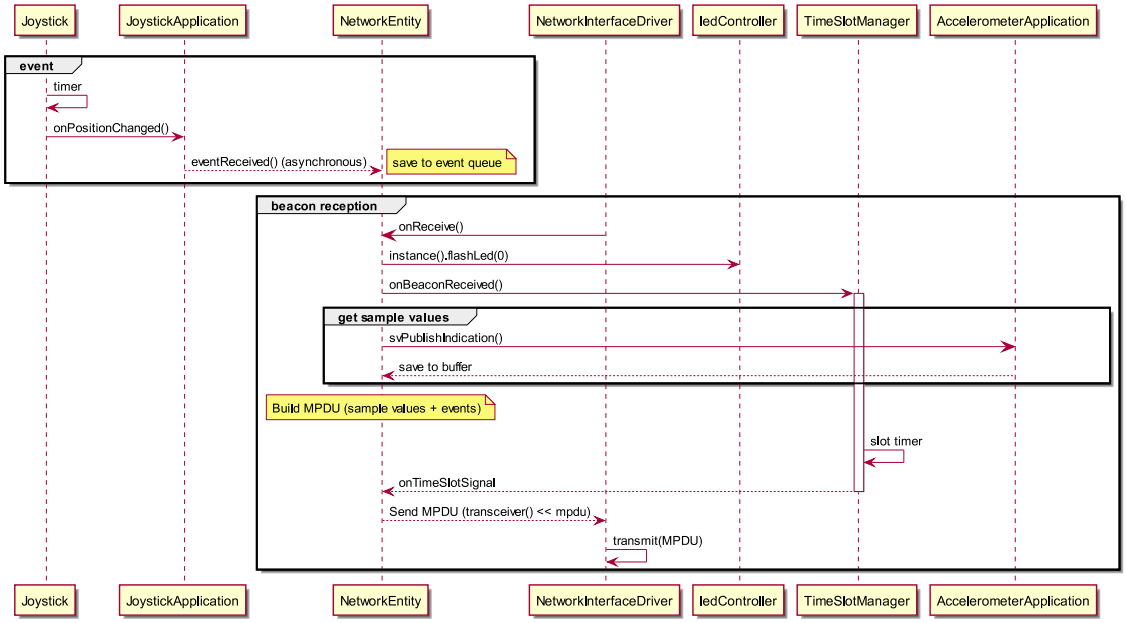


FIGURE 1 – Diagramme de séquence

1.1 Classe `MPDU`

Le classe MPDU permet de construire une frame contenant des données d'événements et/ou des "sampled values" (valeurs mesurées demandées par la station de base).

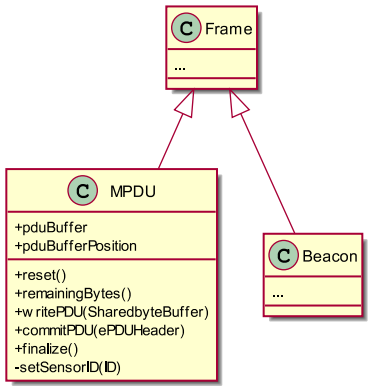
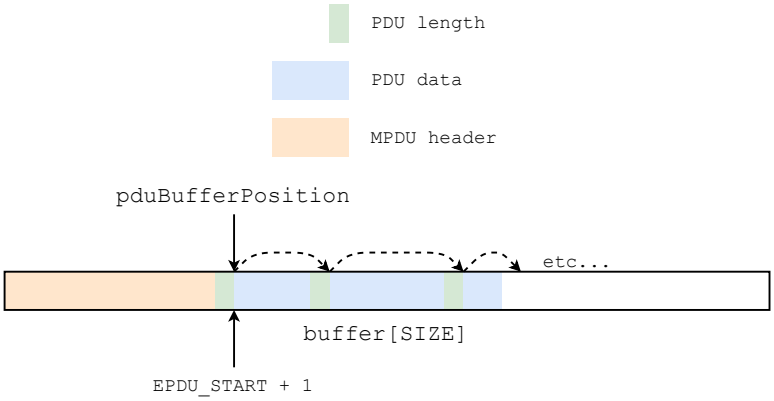


FIGURE 2 – Classe `MPDU`

Les informations des classes `Beacon` et `Frame` n'ont pas été notées car elles n'ont pas été modifiées.

1.1.1 Description des méthodes

1. `pduBuffer` : `SharedByteBuffer` qui permet l'écriture sur le buffer du MPDU. Le début de ce buffer avance continuellement à mesure que de nouvelles données sont écrites dans le MPDU.
2. `pduBufferPosition` : Position actuelle du `pduBuffer` à l'intérieur du "vrai" buffer. Ceci permet à une application d'écrire dans la frame au bon endroit



3. `reset()` permet de réinitialiser le MPDU (position dans le buffer, données, etc...)
4. `remainingBytes()` : donne une information sur la place restante pour écrire les données des applications
5. `writePDU()` : Écrire des données dans le buffer (à la position donnée par `pduBufferPosiiton`
6. `finalize()` : Conditionne le MPDU pour l'envoi final (headers)
7. `setSensorID()` : Enregistre le numéro de slot dans le MPDU (pour l'écrire du header MPDU)

La création d'un MPDU (pour l'envoi à la station de base) se fait de la manière suivante :

1. Reset
2. Ajout des sample values (accéléromètre)
3. Ajout des événements (jusqu'à ce qu'il n'y en ait plus ou que le buffer soit plein). Effectuer un "commit" après chaque événement
4. Finalisation

Une fois qu'un MPDU est prêt, il est envoyé au moment où le temps du slot est atteint (slot timer).

1.1.2 svPDU

Pour stocker les données mesurées, on commence par vérifier si le groupe de mesures (`svGroup`) est demandé dans le beacon.
Si c'est le cas, on appelle la méthode `svPublishIndication` de chaque application en passant un buffer. L'application va écrire les données dans ce buffer puis retourner le nombre de bytes écris. On appelle ensuite la méthode `commit` du MPDU qui va créer le header pour ces données. On recommence cette séquence pour chaque svGroup.

1.1.3 evPDU

Une fois que tous les svPDU sont stockés, on rempli le MPDU avec des evPDU (tant qu'il y en a ou jusqu'à ce que le MPDU soit plein). Pour cela on va lire une liste des événements (`eventsQueue`) et ajouter chaque élément dans l'ordre.
Si le MPDU est plein, on supprime les événements restants (il est possible de perdre des changements d'état sur le joystick).

1.1.4 Envoi

Lorsque le timeslot est atteint (Temps du slot × numéro de slot en millisecondes), on envoie le MPDU à la station de base.
On utilise la fonction `onTimeSlotSignal` en vérifiant que le signal émis est `OWN_SLOT_START`. Si cette vérification n'est pas faite, le MPDU sera envoyé plusieurs fois (lors d'autres signaux qui ne correspondent pas).

1.2 Classe `JoystickApplication`

La classe `JoystickApplication` a été copiée à partir de la classe `AccelerometerApplication`

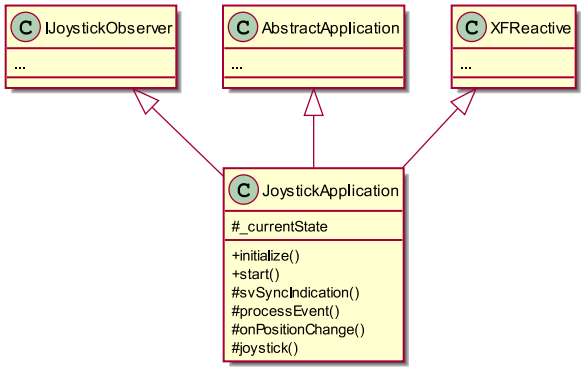


FIGURE 3 – Classe `JoystickApplication`

1. `_currentState` : état actuel de la machine d'état
2. `initialize()` : Initialisation de l'application (inscription vers le `networkentity` avec le bon numéro de groupe)
3. `start()` : démarrage de la machine d'état
4. `svSyncIndication()` : méthode appelée lorsque le beacon est reçu (pour la synchronisation des applications)
5. `processEvent()` : traitement des événements de la machine d'état
6. `onPositionChange()` : méthode appelée lorsqu'un mouvement est détecté sur le joystick (vient de `IJoystick`)
7. `joystick()` méthode permettant d'obtenir le singleton `joystick` (depuis la `Factory`)

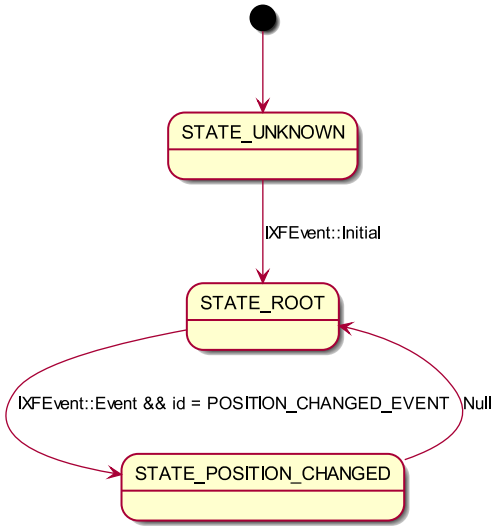


FIGURE 4 – Machine d'état de `JoystickApplication`