

**NOM DU DOSSIER**

**Sous titre**

Date

Service

**RAPPORT DE PROJET**

**Antoine ALAVERDOV, Clémence LEMEILLEUR, Clément VIGAND**

**Promo 56, Année 2021/2022 – 4IR-SI-B1**

*« Projet Robot, Temps Réel »*

Février 2022

Encadrant : J.Tury

**RAPPORT DE PROJET**

Antoine ALAVERDOV, Clémence LEMEILLEUR, Clément VIGAND

Promo 56, Année 2021/2022 – 4IR-SI-B1

*“Projet Robot, Temps réel*”

Février 2022

Encadrant: J.Tury

SOMMAIRE :

1. **Conception du système1**
2. Diagramme fonctionnel général1
3. Groupe de threads gestion du moniteur1
4. Diagramme fonctionnel du groupe de gestion du moniteur5
5. Diagrammes d’activité du groupe gestion du moniteur5
6. Groupe de threads gestion du robot1
7. Diagramme fonctionnel du groupe de gestion du robot5
8. Diagramme fonctionnel du groupe d’activité du robot5
9. Groupe de threads vision1
10. Diagramme fonctionnel du groupe vision5
11. Diagramme fonctionnel du groupe de gestion du robot5
12. **Transformation AADL vers Xenomai7**
13. Thread7
14. Instanciation et démarrage11
15. Code à exécuter11
16. Niveau de priorités11
17. Activation périodique11
18. Donnée partagée8
19. Instanciation11
20. Accès en lecture et écriture11
21. Port d’évènement 8
22. Instanciation11
23. Envoi d’un évènement11
24. Réception d’un évènement11
25. Ports d’évènements-données8
26. Instanciation11
27. Envoi d’une donnée11
28. Réception d’une donnée11
29. **Analyse et validation de la conception9**

**Table des illustrations13**

**Table des annexes14**

**I-** **Conception du système**

1. *Diagramme fonctionnel général*

Voici un diagramme fonctionnel qui présente les principaux blocs de notre conception.

On y retrouve pour chaque groupe de thread les données et ports partagés. Les 3 groupes de threads son visibles : th\_group\_gestion\_moniteur, th\_group\_vision, th\_group\_gestion\_robot.

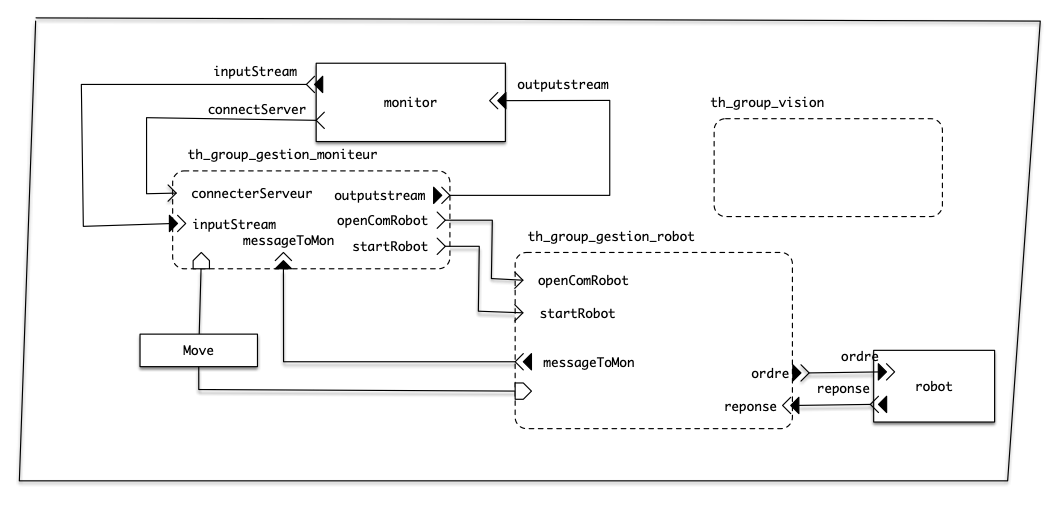


Fig. 1: Diagramme fonctionnel du système

1. *Groupe de threads gestion du moniteur*

Dans cette partie, nous retrouvons le diagramme fonctionnel en AADL décrivant le groupe de threads de gestion du moniteur et les diagrammes d’activité de chaque thread de ce groupe.

1. Diagramme fonctionnel du groupe de gestion du moniteur

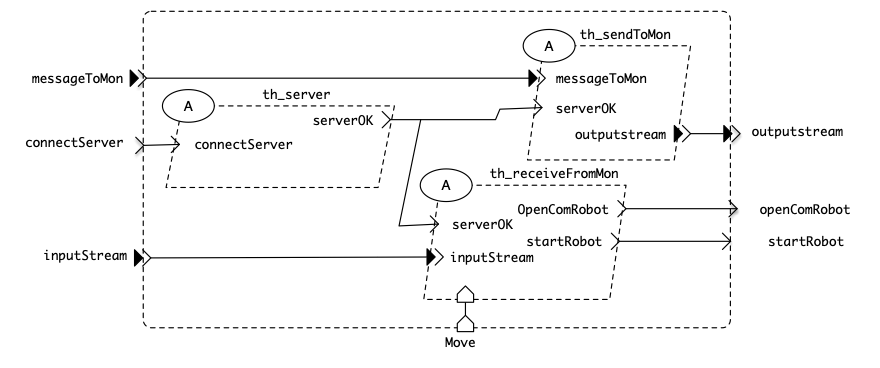


Fig. 2: Diagramme fonctionnel du groupe de threads gestion du moniteur

1. Diagramme d’activité du groupe gestion du moniteur

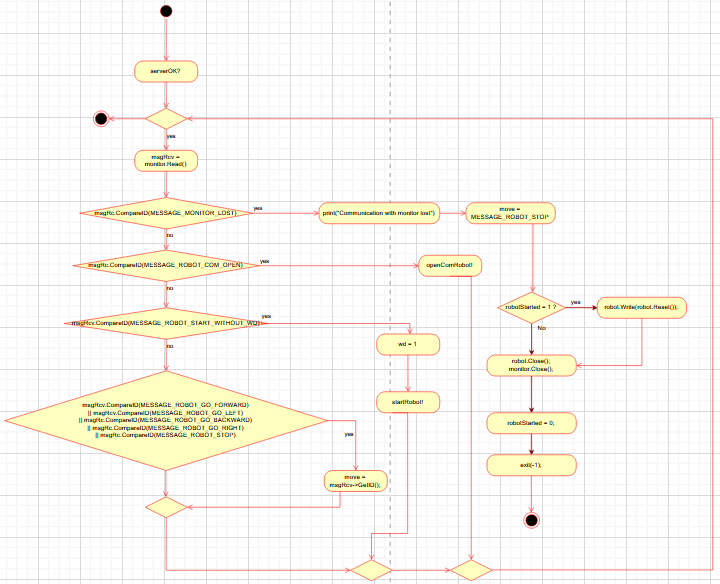


Fig. 3: Diagramme d’activité du thread th\_receiveFromMon

Le moniteur envoie des messages au superviseur qui les agit en conséquence en débloquant les sémaphores correspondants et en mettant à jour les variables partagées. Lorsque la communication entre les deux est coupée, le message correspond à "MESSAGE\_MONITOR\_LOST", alors on réinitialise le système complet.

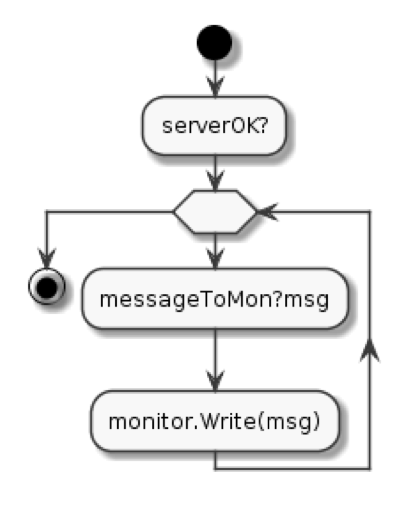


Fig. 4: Diagramme d’activité du thread th\_sendToMon

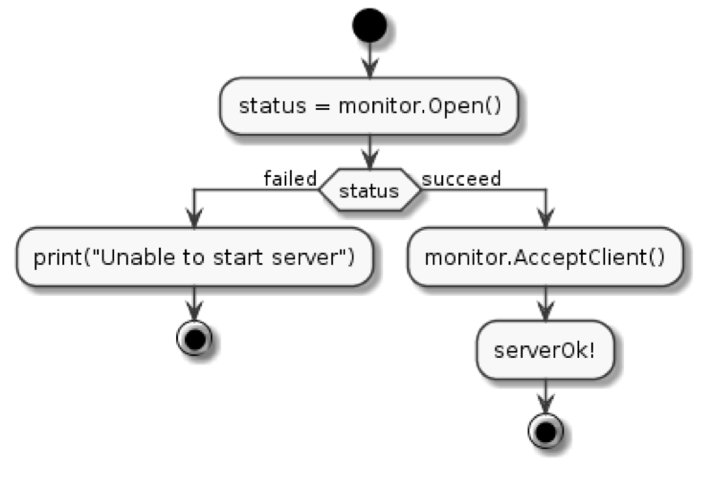


Fig. 5: Diagramme d’activité du thread th\_server

1. *Groupe de threads gestion du robot*
2. Diagramme fonctionnel du groupe de gestion du robot

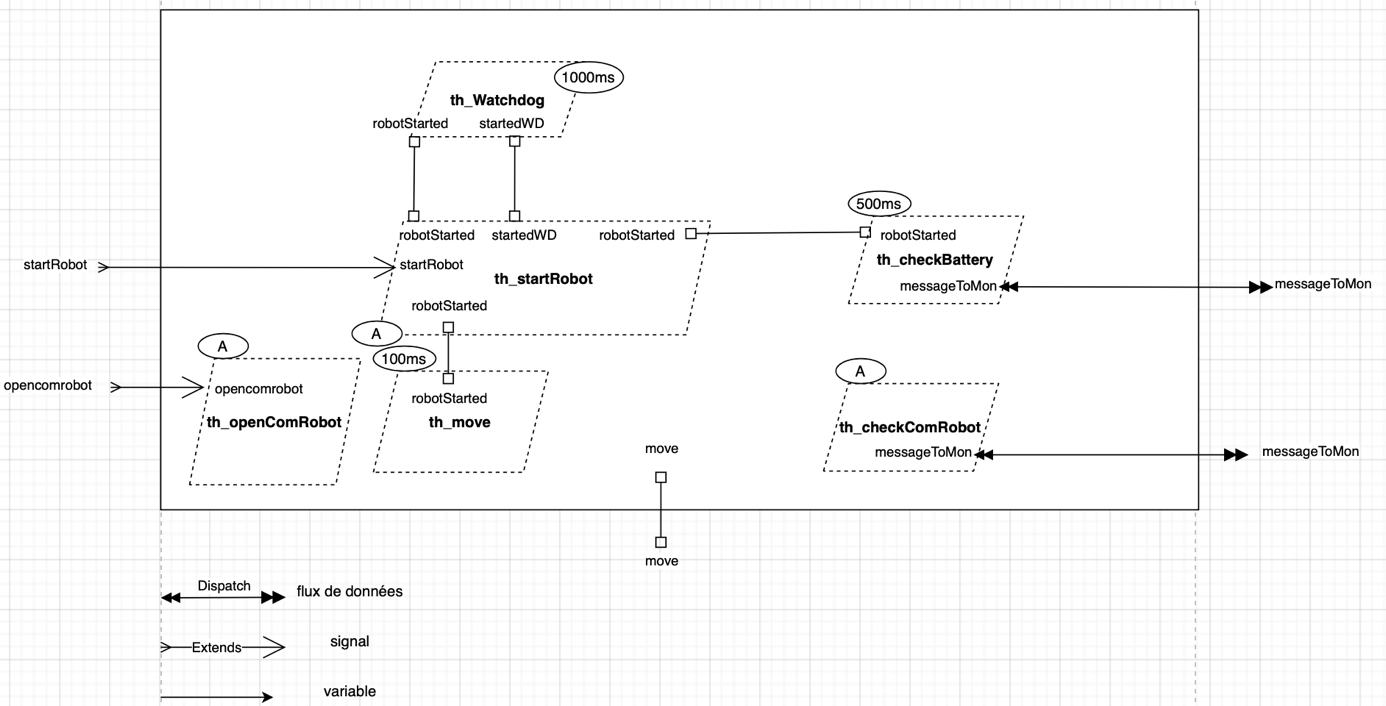


Fig. 6: Diagramme fonctionnel du groupe de gestion du robot

1. Diagramme d’activité du groupe d’activité du robot

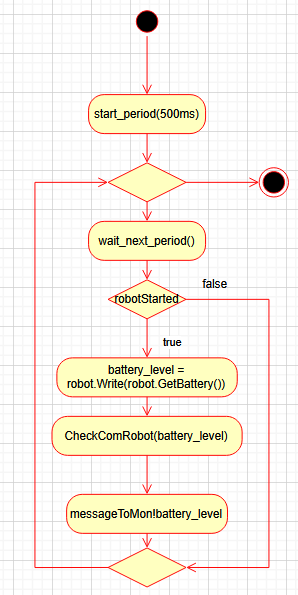


Fig. 7: Diagramme d’activité du thread th\_checkBattery

Toutes les 500ms, le superviseur demande le niveau de batterie du robot puis le transmet au moniteur.

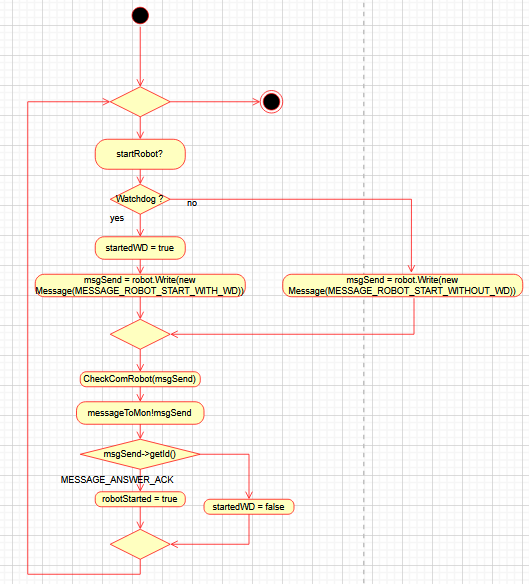


Fig. 8: Diagramme d’activité du thread th\_startRobot

La variable watchdog correspond à la demande de démarrage en mode watchdog tandis que la variable startedWD indique que le robot est bien démarré en mode watchdog.

Ce thread démarre le robot dans le mode demandé par le moniteur.

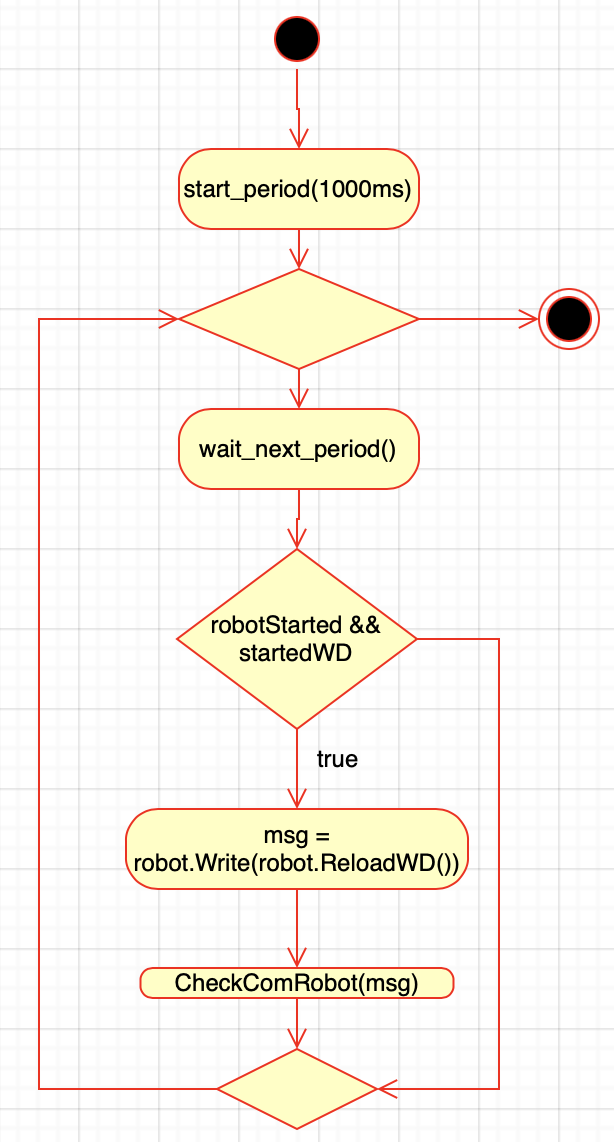


Fig. 9: Diagramme d’activité du thread th\_Watchdog

Lorsque le robot est démarré en mode watchdog, toutes les secondes le superviseur envoie l’ordre au robot de réinitialiser le compteur du watchdog.

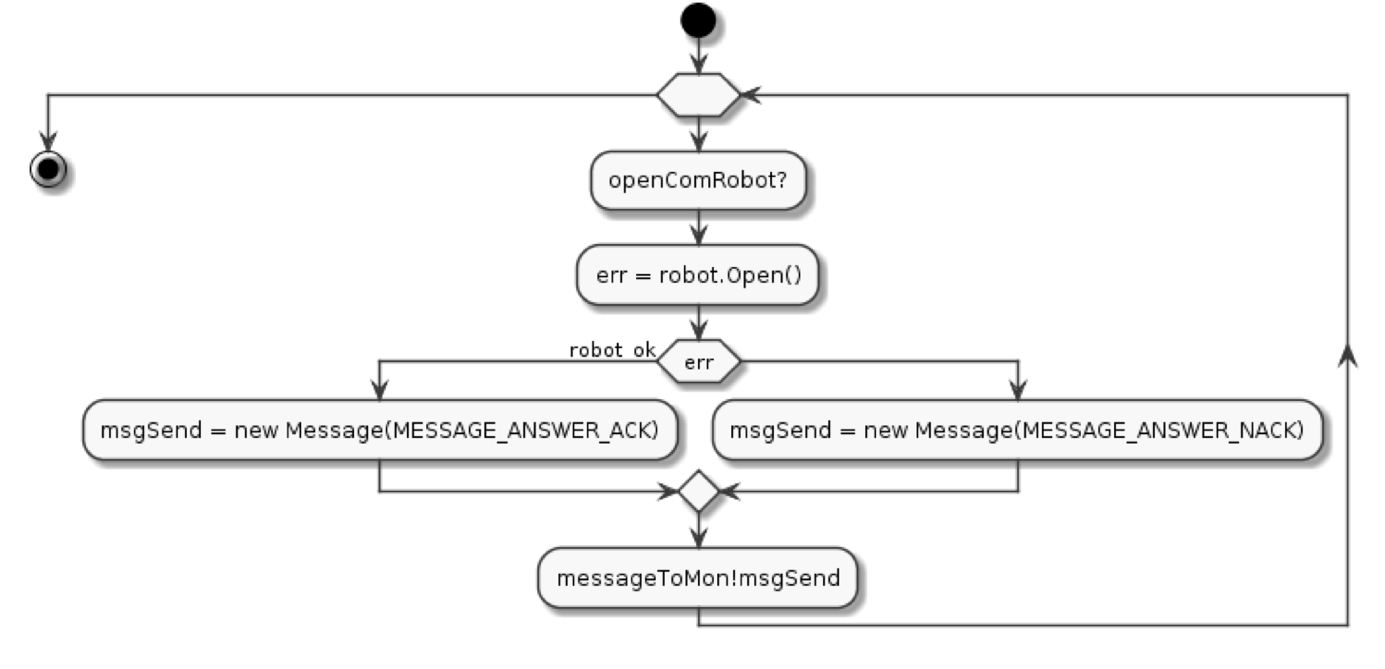


Fig. 10: Diagramme d’activité du thread th\_openComRobot

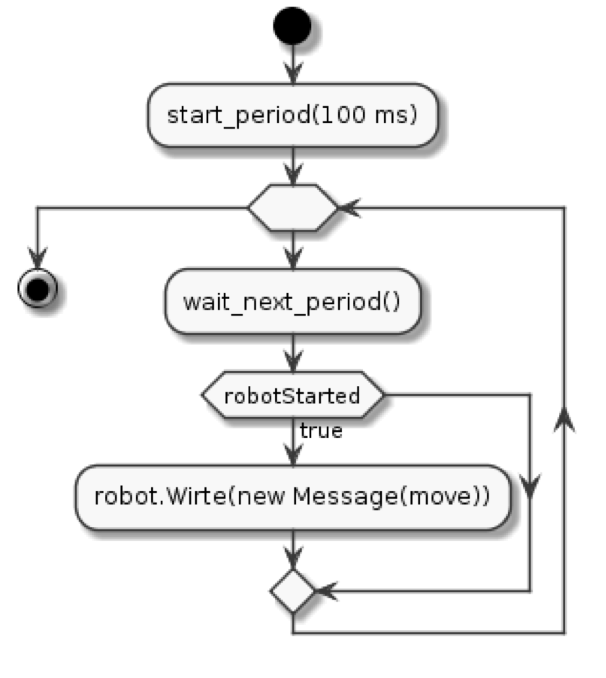


Fig. 11: Diagramme d’activité du thread th\_move

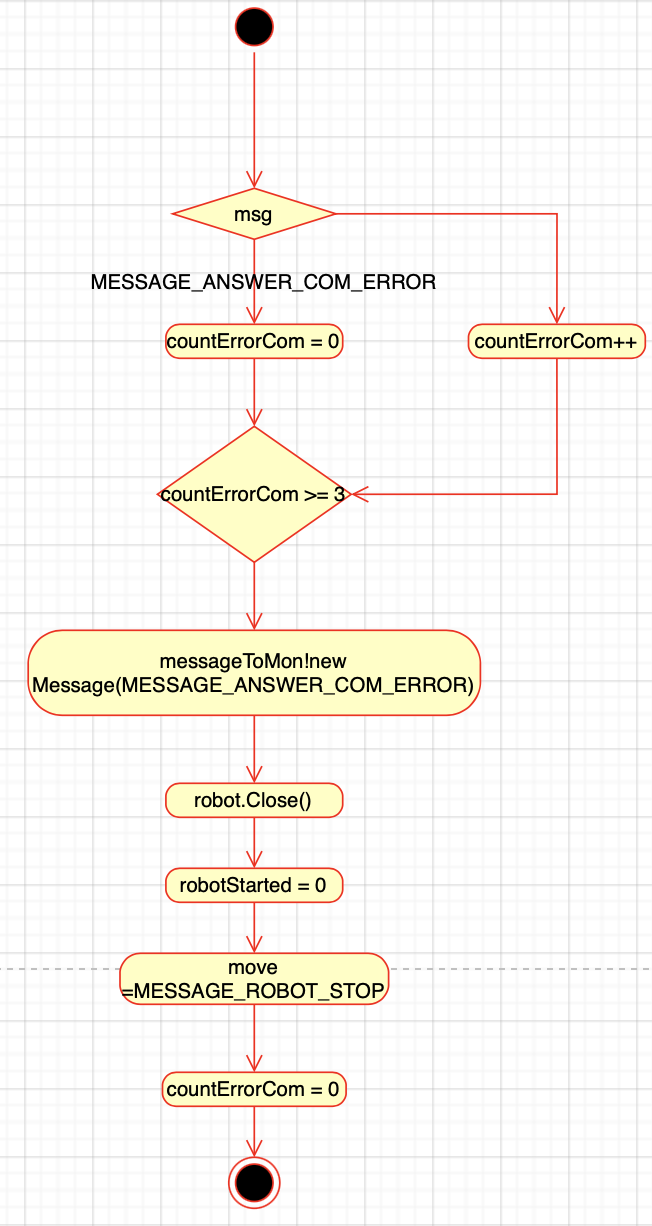


Fig. 12: Diagramme d’activité du thread th\_checkComRobot

C'est une fonction qui est exécutée à chaque fois qu'un message est émis vers le robot. Si le message produit une erreur on incrémente un compteur qui indique que la communication a été perdue si ce compteur dépasse 3.

1. *Groupe de threads vision*
2. Diagramme fonctionnel du groupe vision

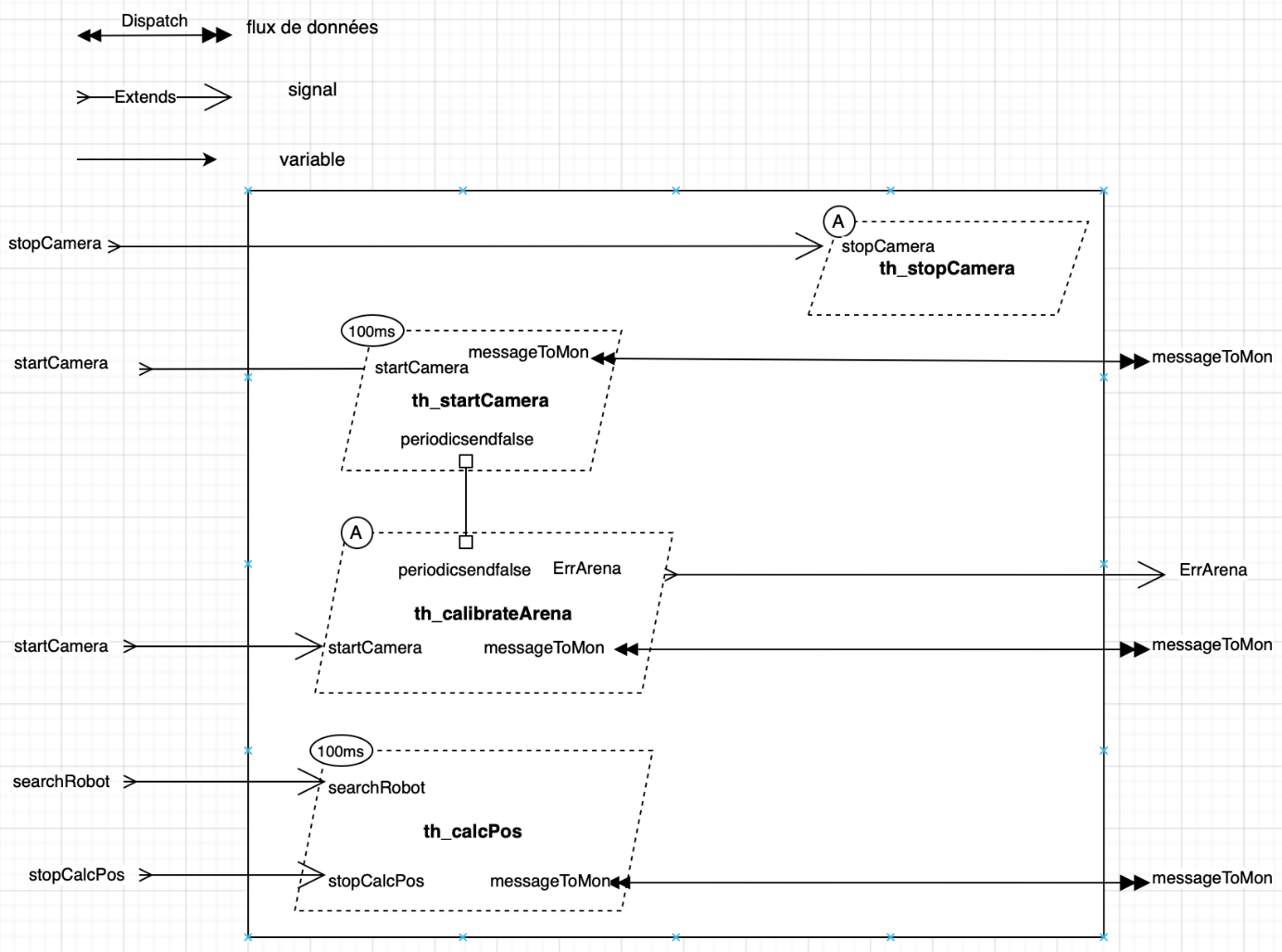


Fig. 13: Diagramme fonctionnel du groupe vision

1. Diagramme d’activité du groupe de gestion du robot

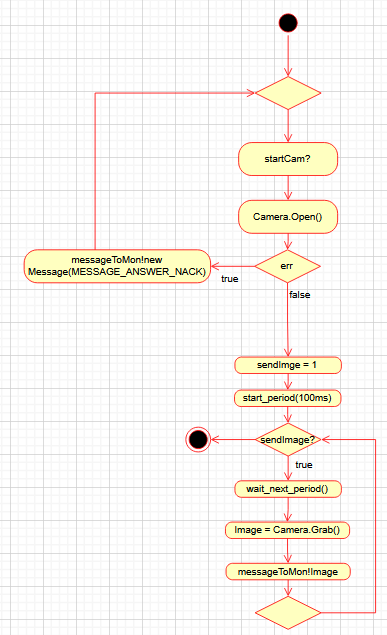


Fig. 14: Diagramme d’activité du thread th\_startCamera

Le thread démarre la caméra, si cela réussie alors une image est envoyée au moniteur toutes les 100ms.

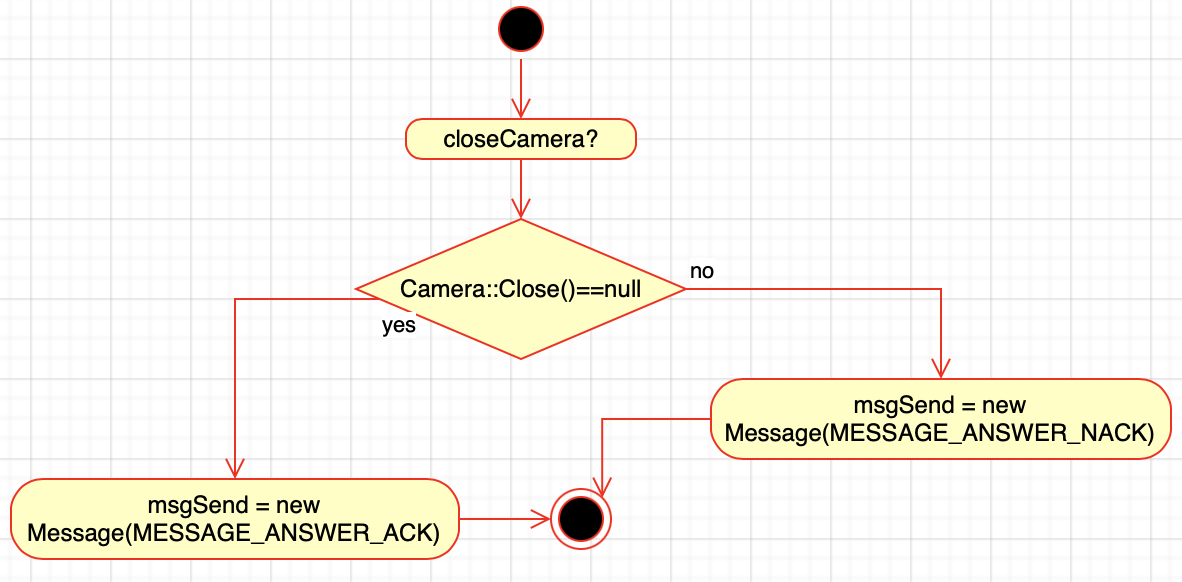


Fig. 15: Diagramme d’activité du thread th\_stopCamera

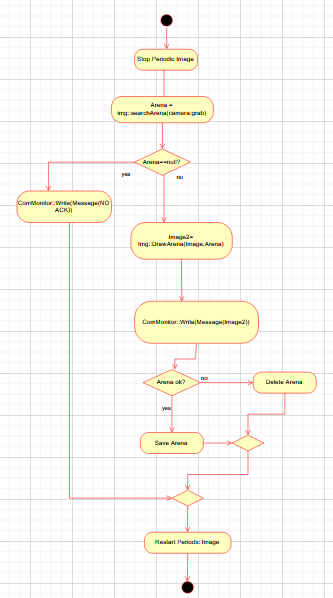


Fig. 16: Diagramme d’activité du thread th\_calibrateArena

L'utilisateur fait une demande de recherche d'arène qui envoie et affiche celle-ci. S’il n'y a pas d'erreur à la réception de l'image de l’arène alors l'utilisateur doit valider ou non la calibration. On sauvegarde l’arène si elle est ok, on la supprime sinon.

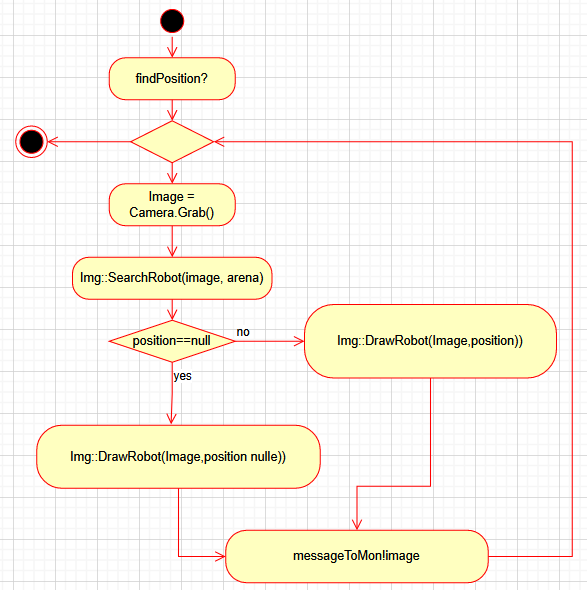


Fig. 17: Diagramme d’activité du thread th\_calcPos

**II-** **Transformation AADL vers Xenomai**

Dans cette partie on abordera la méthode pour passer d’un modèle AADL à un code sous Xenomai.

1. *Thread*
2. Instanciation et démarrage

Chaque thread a été implémenté par un RT\_TASK déclarés dans le fichier tasks.h. La création de la tâche se fait à l’aide du service rt\_task\_create et son démarrage à l’aide de rt\_task\_start.

Toutes les tâches sont crées dans la méthode init de tasks.cpp et démarrées dans la méthode run. Par exemple, pour la tâche th\_checkBattery, sa déclaration est faite ligne 83 dans le fichier tasks.h : RT\_TASK th\_checkBattery;

Sa création ligne 136 de tasks.cpp lors de l’appel de rt\_task\_create(&th\_checkBattery, "th\_checkBattery", 0, PRIORITY\_TBATTERY, 0) et son démarrage ligne 188 avec rt\_task\_start(&th\_checkBattery, (void(\*)(void\*)) & Tasks::CheckBattery, this).

1. Code à éxécuter

Le lien sous Xenomai entre le thread et le traitement à exécuter se fait via l’appel à rt\_task\_start qui permet de lancer le thread et permet au système opérant d’exécuter les instructions présentes dans la tâche.

1. Niveau de priorités

Pour définir le niveau de priorité d’un thread sous Xenomai, nous avons défini des macros représentant leur priorité dans le fichier tasks.cpp : PRIORITY\_TSERVER 30. Ensuite cette valeur de priorité est passé en paramètre de rt\_task\_create lors de la création de la tâche : rt\_task\_create(&th\_server, "th\_server", 0, **PRIORITY\_TSERVER**, 0). Puis le système opérant s’occupera tout seul de l’ordonnancement des tâches selon leur niveau de priorité.

1. Activation périodique

Un thread est rendu périodique par l’utilisation de la primitive Xenomai rt\_task\_set\_periodic au début de la tâche, par exemple rt\_task\_set\_periodic(NULL, TM\_NOW, 500000000); pour rendre la tâche th\_checkBattery périodique de 500ms. Il est également nécessaire d’utiliser rt\_task\_wait\_period au début du while(1) pour bloquer l’execution du thread jusqu’au prochain cycle.

1. *Donnée partagée*
2. Instanciation

Les données partagées sont déclarées et initialisées si besoin dans le fichier tasks.h.

1. Accès en lecture et écriture

L’utilisation d’un mutex est nécessaire pour l’accès à une donnée partagée, pour accéder à la donnée il faut d’abord prendre le mutex, une fois la lecture ou l’écriture finie, il faut le libérer pour laisser d’autres tâches y accéder, par exemple :

rt\_mutex\_acquire(&mutex\_robotStarted, TM\_INFINITE);

rs = robotStarted;

rt\_mutex\_release(&mutex\_robotStarted);

Ici nous lisons la valeur de la donée partagée robotStarted.

Les mutex sont tous définis dans le fichier tasks.h : RT\_MUTEX mutex\_robotStarted; Ils sont ensuite créés dans la fonction init du fichier tasks.cpp : rt\_mutex\_create(&mutex\_robotStarted, NULL).

1. *Port d’évènement*
2. Instanciation

Nous avons utilisé des sémaphores pour modéliser les ports d’événement. Ceux-ci sont déclarés dans le fichier tasks.h puis créés dans la fonction init du fichier tasks.cpp via rt\_sem\_create dans la fonction init.

Par exemple pour le semaphore sem\_startRobot, sa déclaration est faite ligne 103 du fichier tasks.h RT\_SEM sem\_startRobot, sa création ligne 103 du fichier tasks.cpp rt\_sem\_create(&sem\_startRobot, NULL, 0, S\_FIFO).

1. Envoi d’un évènement

La libération d’un sémaphore représente l’envoi d’un événement, pour demander le démarrage du robot : rt\_sem\_v(&sem\_startRobot);

1. Réception d’un évènement

La prise de sémaphore représente la réception d’un événement, exemple : rt\_sem\_p(&sem\_startRobot, TM\_INFINITE); SI le sémaphore est disponible, il le prend et continue son exécution, sinon il reste bloqué jusqu’à ce que le sémaphore soit libéré.

1. *Port d’évènement-données*
2. Instanciation

Les données transmises sont de type Message. Pour communiquer entre le moniteur et le superviseur, nous avons utilisé une file d’attente de type RT\_QUEUE, celle-ci est déclarée ligne 110 du fichier tasks.h puis créée ligne 149 du fichier tasks.cpp dans la fonction init : rt\_queue\_create(&q\_messageToMon, "q\_messageToMon", sizeof (Message\*)\*50, Q\_UNLIMITED, Q\_FIFO).

1. Envoi d’une donnée

Pour envoyer une donnée au moniteur, nous avons encapsulé cette donnée dans un message qui est ensuite envoyé au moniteur via le thread WriteInQueue qui écrit le message dans la file, ensuite le thread SendToMonTask lit le dernier message de la queue puis l’envoie au moniteur via monitor.Write().

1. Réception d’une donnée

Le thread ReceiveFromMonTask s’occupe de la réception des données depuis le moniteur, il lit via monitor.Read() les messages reçus par le superviseur depuis le moniteur puis selon le contenu du message envoi ou non l’événement correspondant.

**III-** **Analyse et validation de la conception**

Pour cette partie nous avons réalisé une vidéo dans laquelle nous montrons que chacune des fonctionnalités 1 à 13 est opérationnelles. Nous avons particulièrement axé notre démonstration sur les fonctionnalités 5,6, 8, 9, 11 et 13 étant donné que les autres étaient déjà codé et que nous ne les avons pas modifiés.

Dans notre vidéo il est possible de voir que (détailler ce que font les fonctionnalités qu’on a codé)

Cette vidéo est disponible sur notre github également.

Pour réaliser la vidéo, enregistrer votre écran en commentant. Une vidéo d’exemple (rapidement produite et avec le simulateur) pour les fonctionnalités (1,2,3, 4, 7, 10 et 12) est postée sur https://drive.google.com/open?id=1Dk5C9FLq-bFZCgSRN2y9dTCk0oxM7fp4

METTRE A JOUR LES PAGES   
METTRE A JOUR LE SOMMAIRES ET LE NUM DES FIGURES AVEC LA TABLE DES ILLUSTRATIONS

METTRE QUI A FAIT QUOI EN DEBUT DE RAPPORT

Table des illustrations

**Figure 1.** – Dossier de conception [16/02/2022]. *Diagramme fonctionnel du système*.

Table des annexes

1. Annexe 1 : Lien GitHubA

**Annexe 1 :** Lien GitHub : https://github.com/Clement-INS/Projet\_robot

