

**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**

La conception doit couvrir l’ensemble des fonctionnalités du cahier des charges.

Mettez dans cette partie tous les éléments de votre conception en particulier vos dia- grammes AADL (vue globale du système) et les diagrammes d’activité (détails des threads). Cette partie doit être auto-suffisante pour comprendre votre application.

Pour faciliter la lecture des schémas, vous allez présenter votre conception en trois parties, l’une focalisée sur la communication entre le moniteur et le superviseur, la seconde consacrée au traitement vidéo et la troisième au contrôle du robot.

Si vous le souhaitez, au lieu de dessiner vos diagrammes sous un éditeur, vous pouvez joindre un scan de vos schémas — ils doivent être lisibles et propres.

1. *Diagramme fonctionnel général*

Mettez ici un diagramme fonctionnel qui présente les principaux blocs de votre conception. Pour cela, inspirez vous du diagramme ci-dessous (fig. 1.1) en indiquant pour chaque groupe de threads les données et ports partagés. La figure 1.1 a été réalisée à partir du document de conception. Vous devez absolument conserver le découpage en trois groupes de threads (th\_group\_gestion\_moniteur, th\_group\_vision, th\_group\_gestion\_robot).

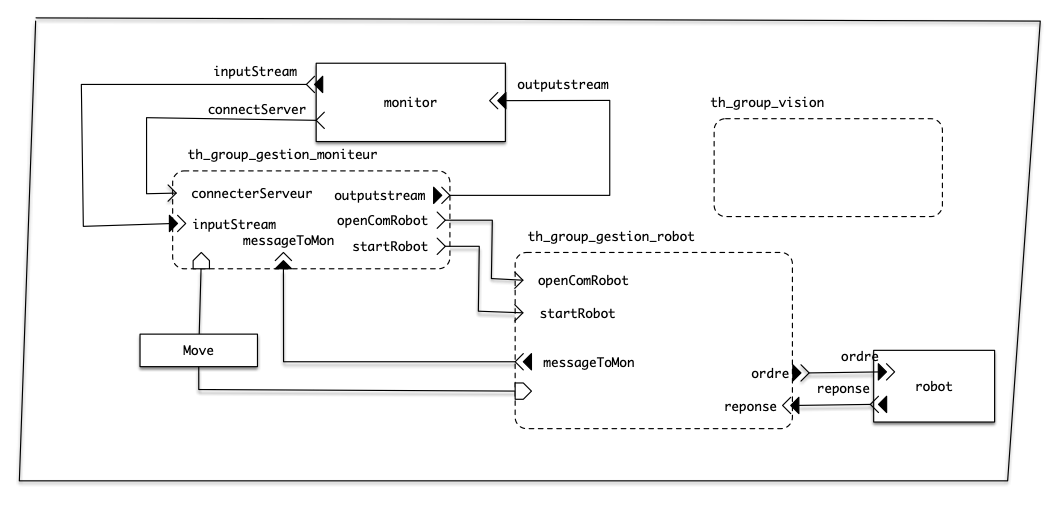


Fig. 1: Diagramme fonctionnel du système

1. *Groupe de threads gestion du moniteur*

Placez ici :  
— le diagramme fonctionnel en AADL décrivant le groupe de threads de gestion du mo-

niteur (voir exemple de la figure 1.2.1 réalisée à partir du dossier de conception), — les diagrammes d’activité de chaque thread de ce groupe.

Décrivez tous les éléments (paramètres, variables, etc.) qui vous semblent pertinents pour comprendre les diagrammes.

1. Diagramme fonctionnel du groupe de gestion du moniteur

Exemple de diagramme fonctionnel pour le groupe de thread de gestion du moniteur. Mettez à jour ce diagramme avec votre conception.

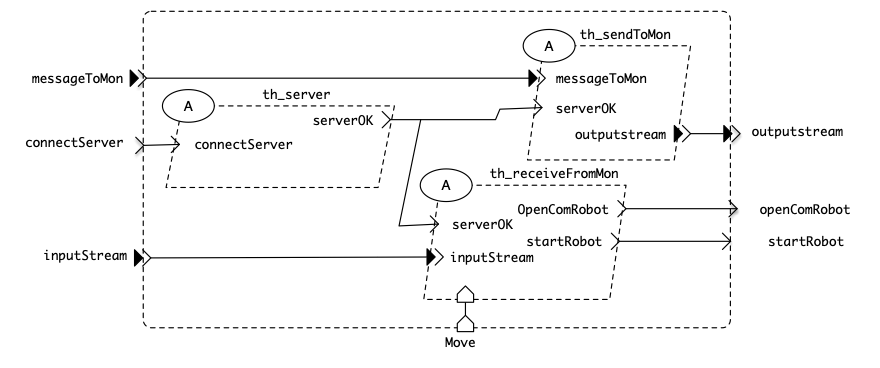


Fig. 2: Diagramme fonctionnel du groupe de threads gestion du moniteur

1. Diagramme d’activité du groupe gestion du moniteur

Décrivez le comportement de chacun de vos threads avec des diagrammes d’activité. Ap- portez les explications qui vous semblent nécessaires pour comprendre votre conception. A titre d’exemple les diagrammes fonctionnels tirés du document de conception sont remis.

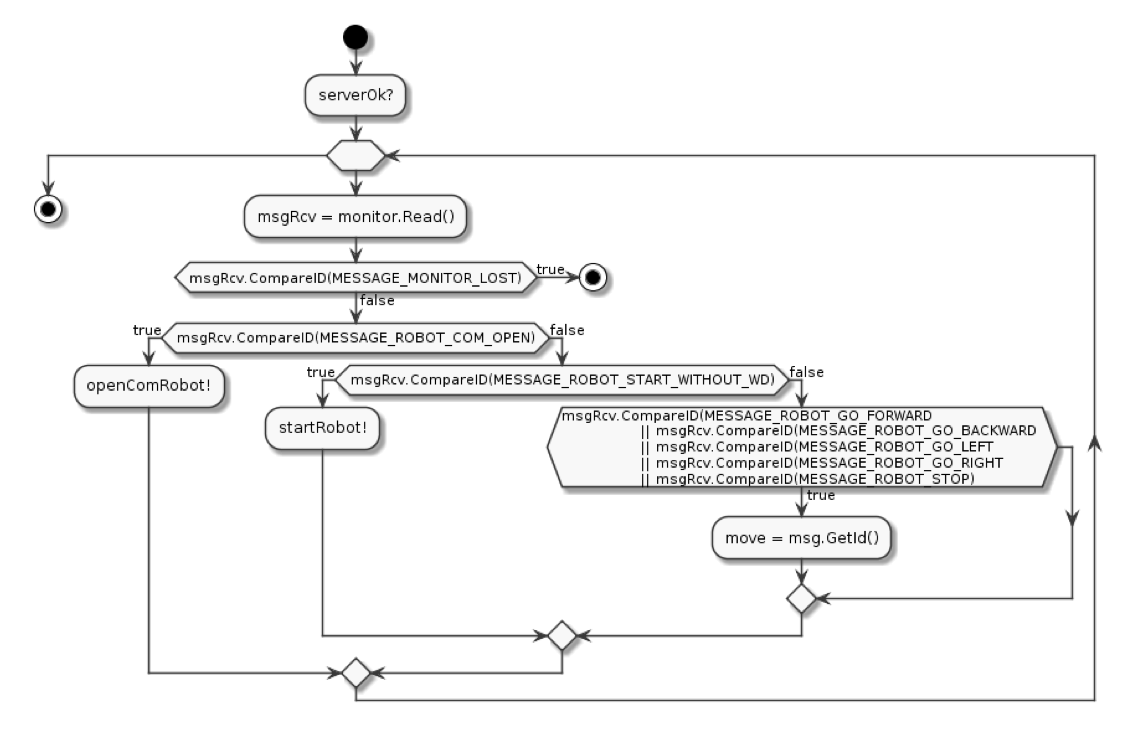


Fig. 3: Diagramme d’activité du thread th\_receiveFromMon

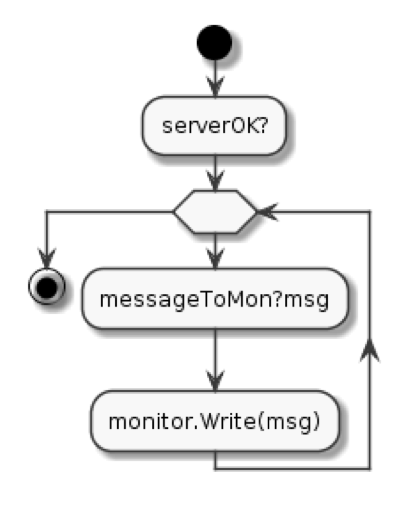


Fig. 4: Diagramme d’activité du thread th\_sendToMon

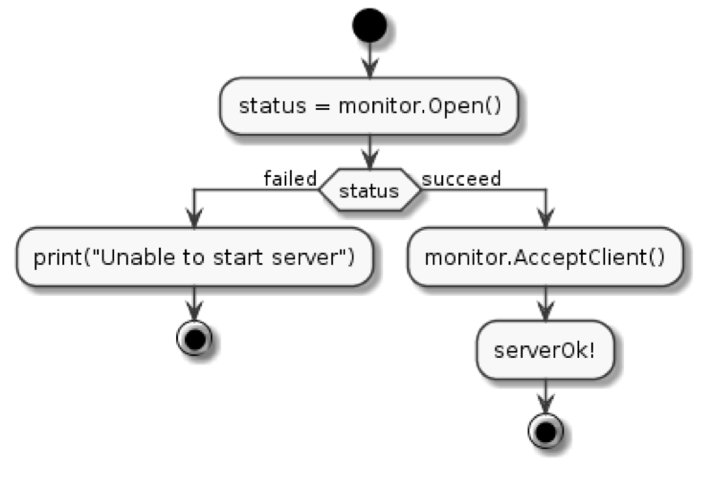


Fig. 5: Diagramme d’activité du thread th\_server

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

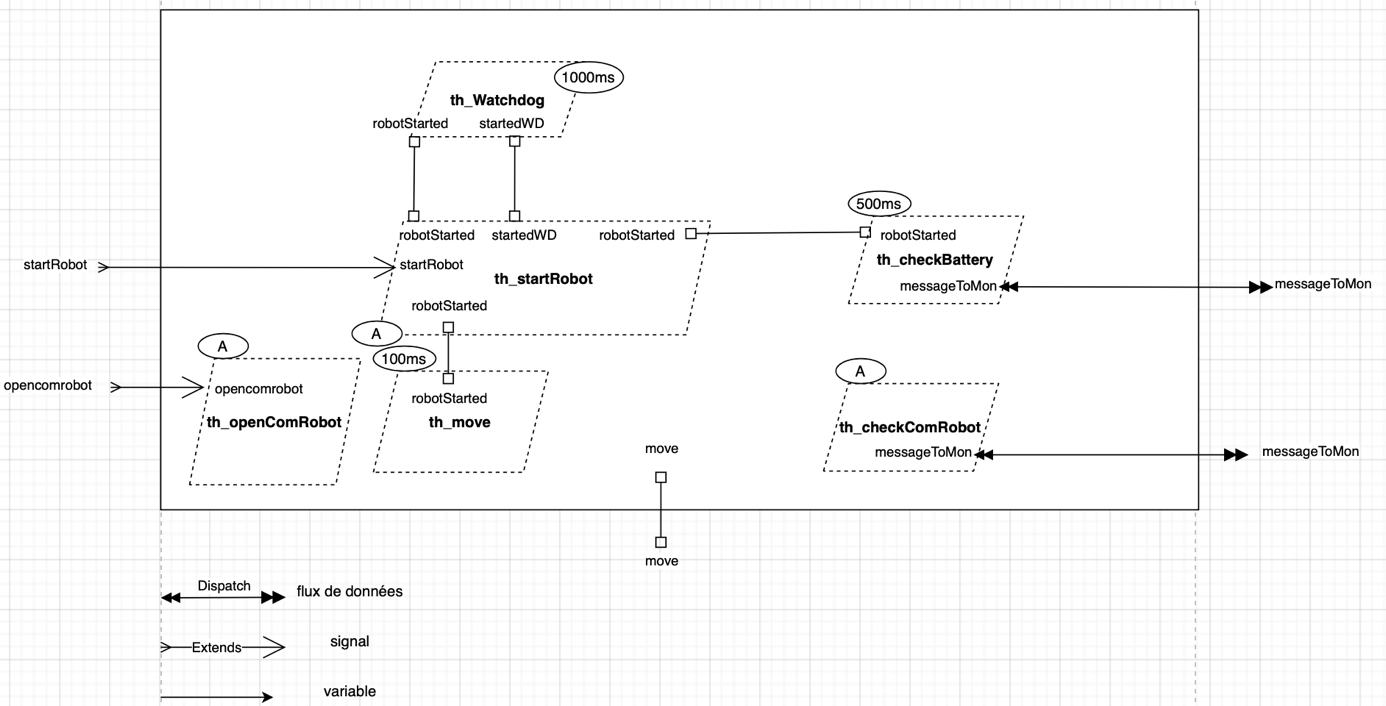


Fig. 6: Diagramme fonctionnel du groupe de gestion du robot

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

Décrivez le comportement de chacun de vos threads avec des diagrammes d’activité. Ap- portez les explications qui vous semblent nécessaires pour comprendre votre conception

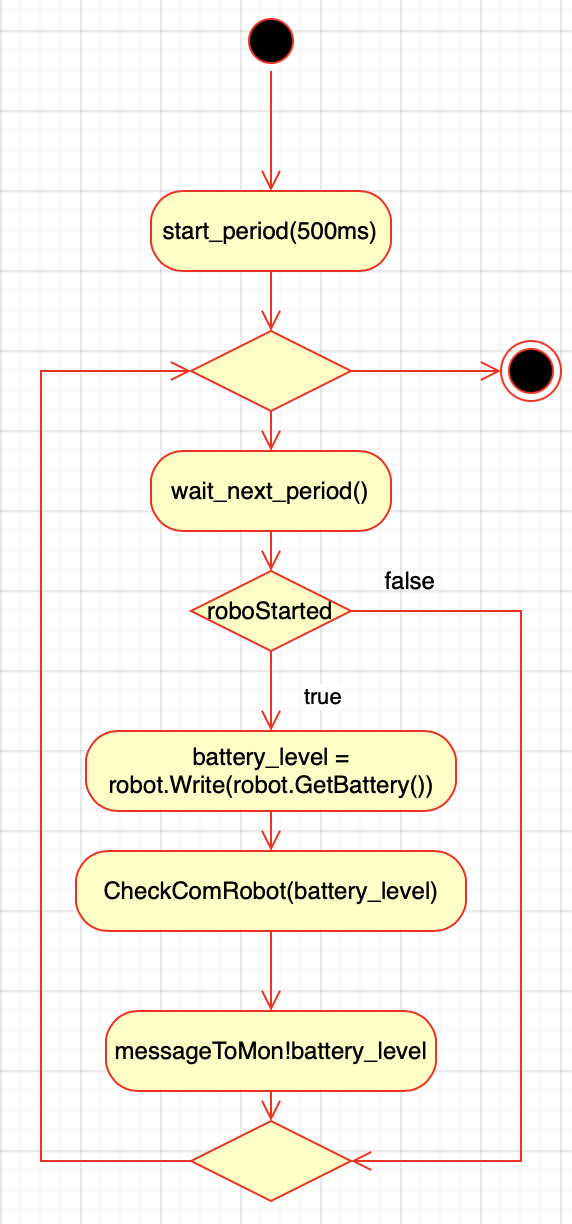


Fig. 5: Diagramme d’activité du thread check battery

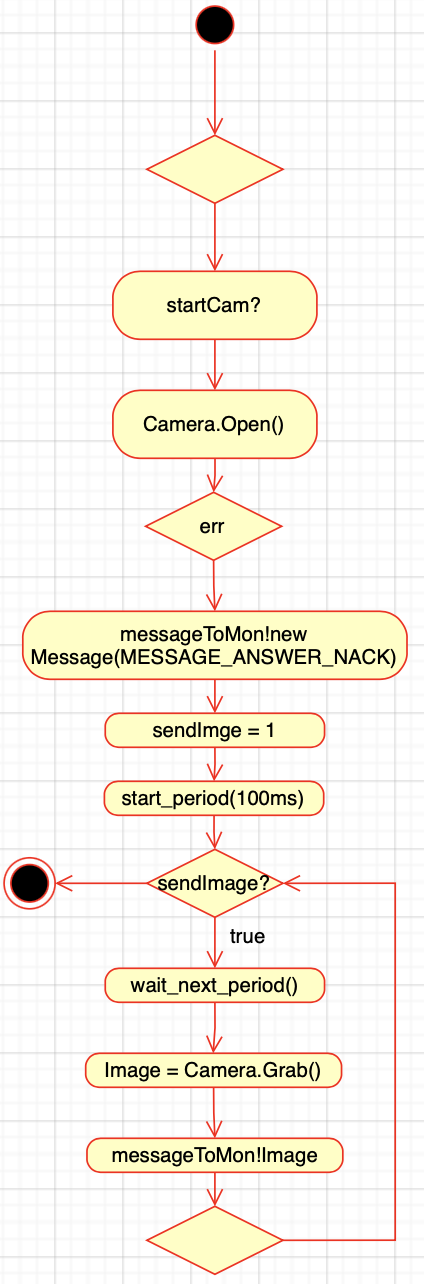


Fig. 5: Diagramme d’activité du thread start robot

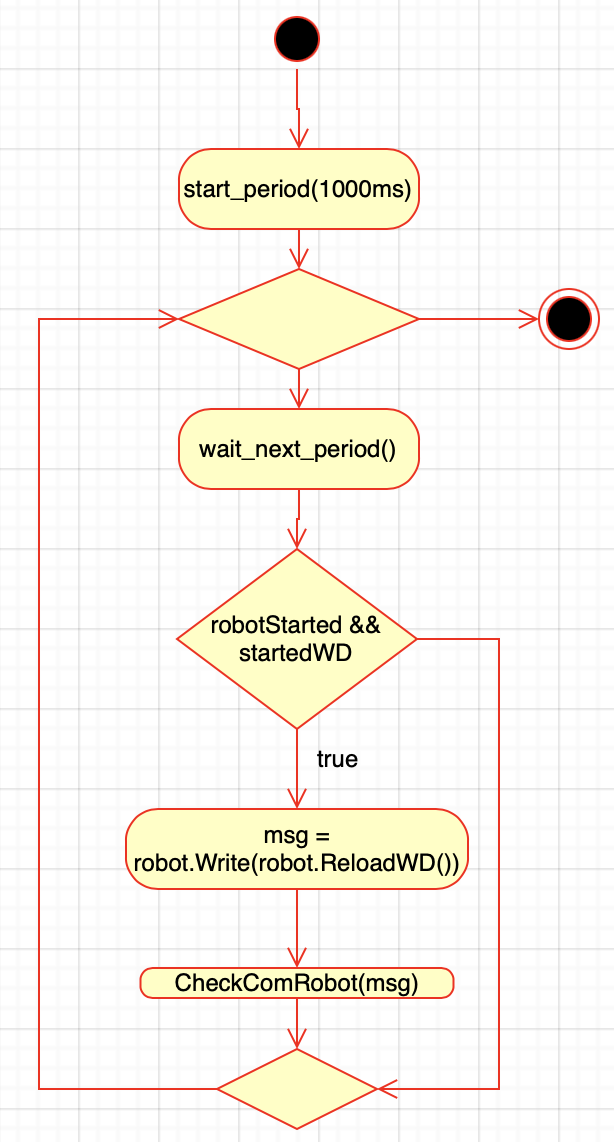


Fig. 5: Diagramme d’activité du thread watchdog

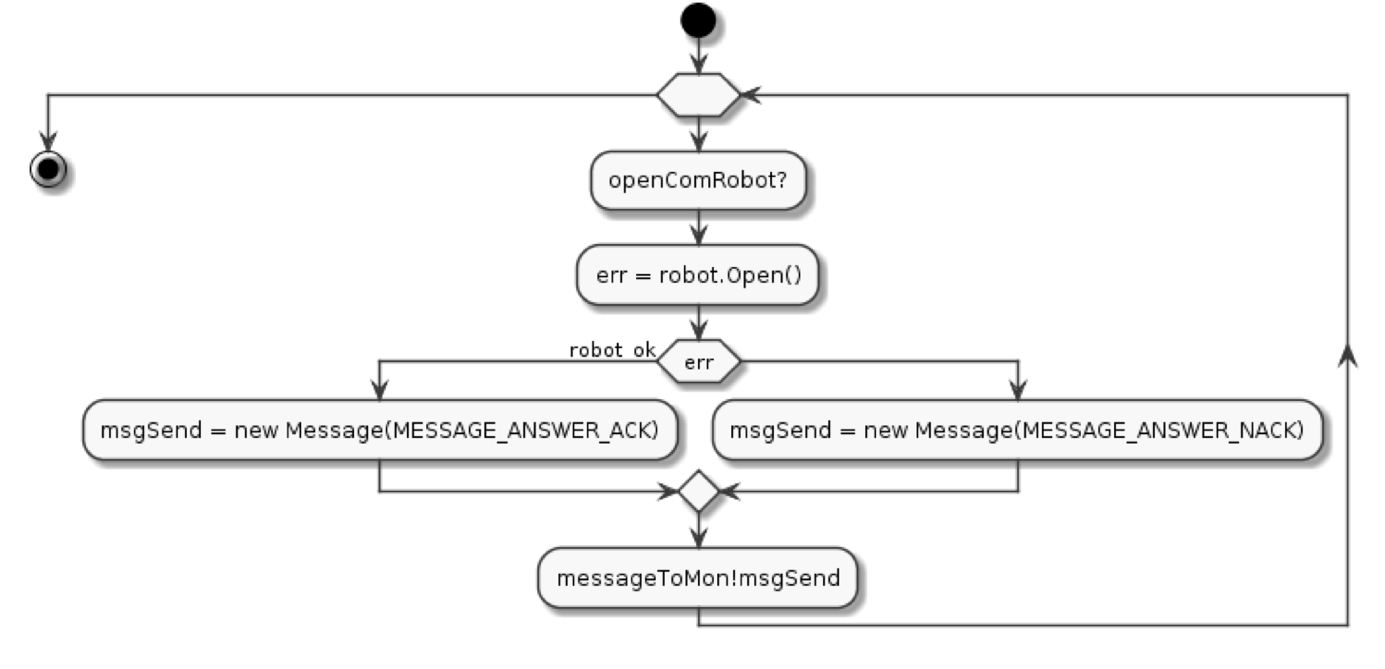


Fig. 5: Diagramme d’activité du thread th\_openComRobot

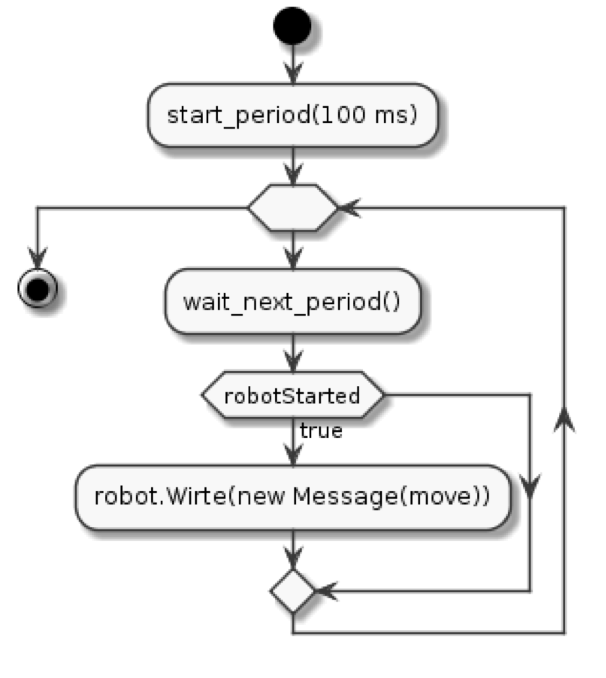


Fig. 5: Diagramme d’activité du thread th\_move

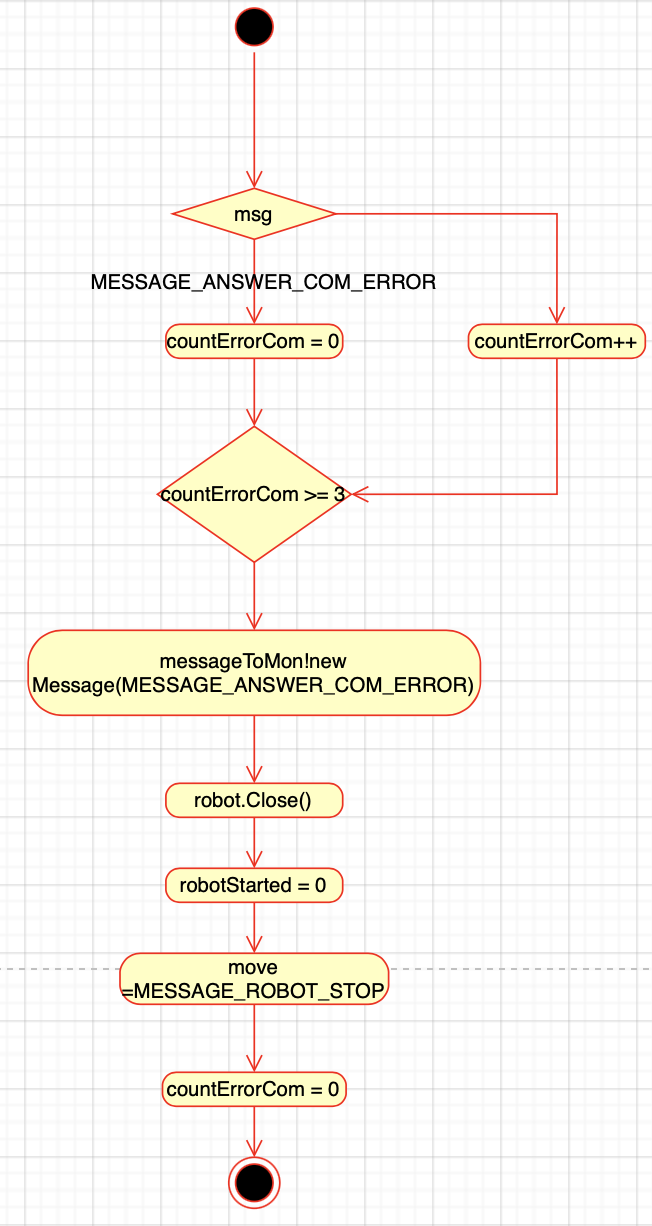


Fig. 5: Diagramme d’activité du thread th\_checkComRobot

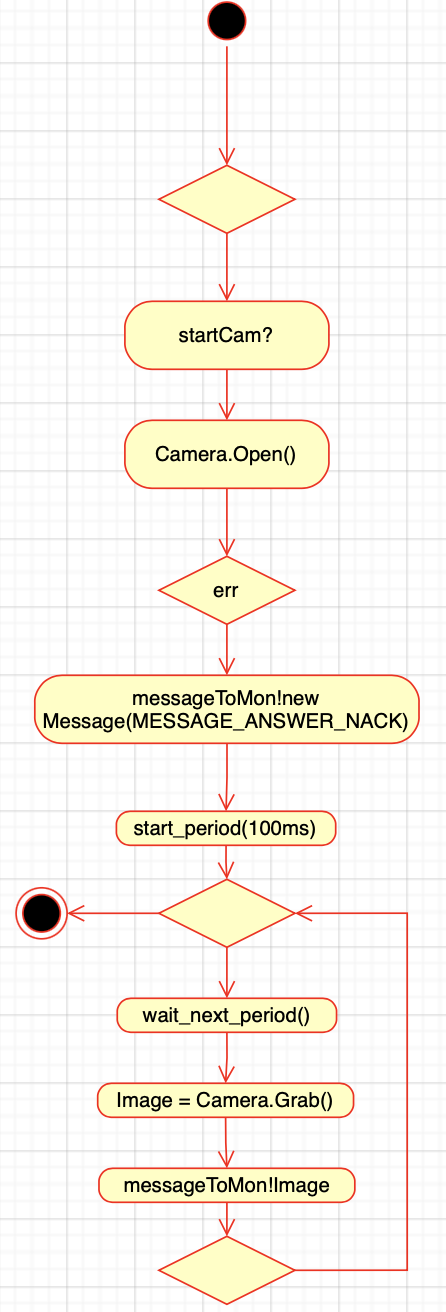


Fig. 5: Diagramme d’activité du thread th\_startCamera

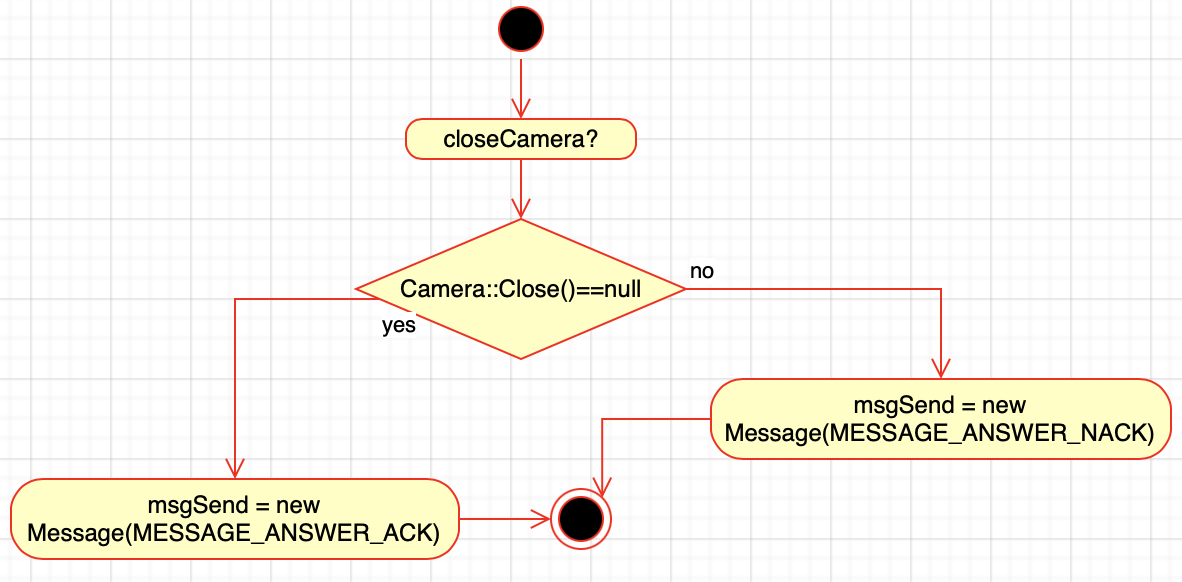


Fig. 5: Diagramme d’activité du thread th\_stopCamera

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

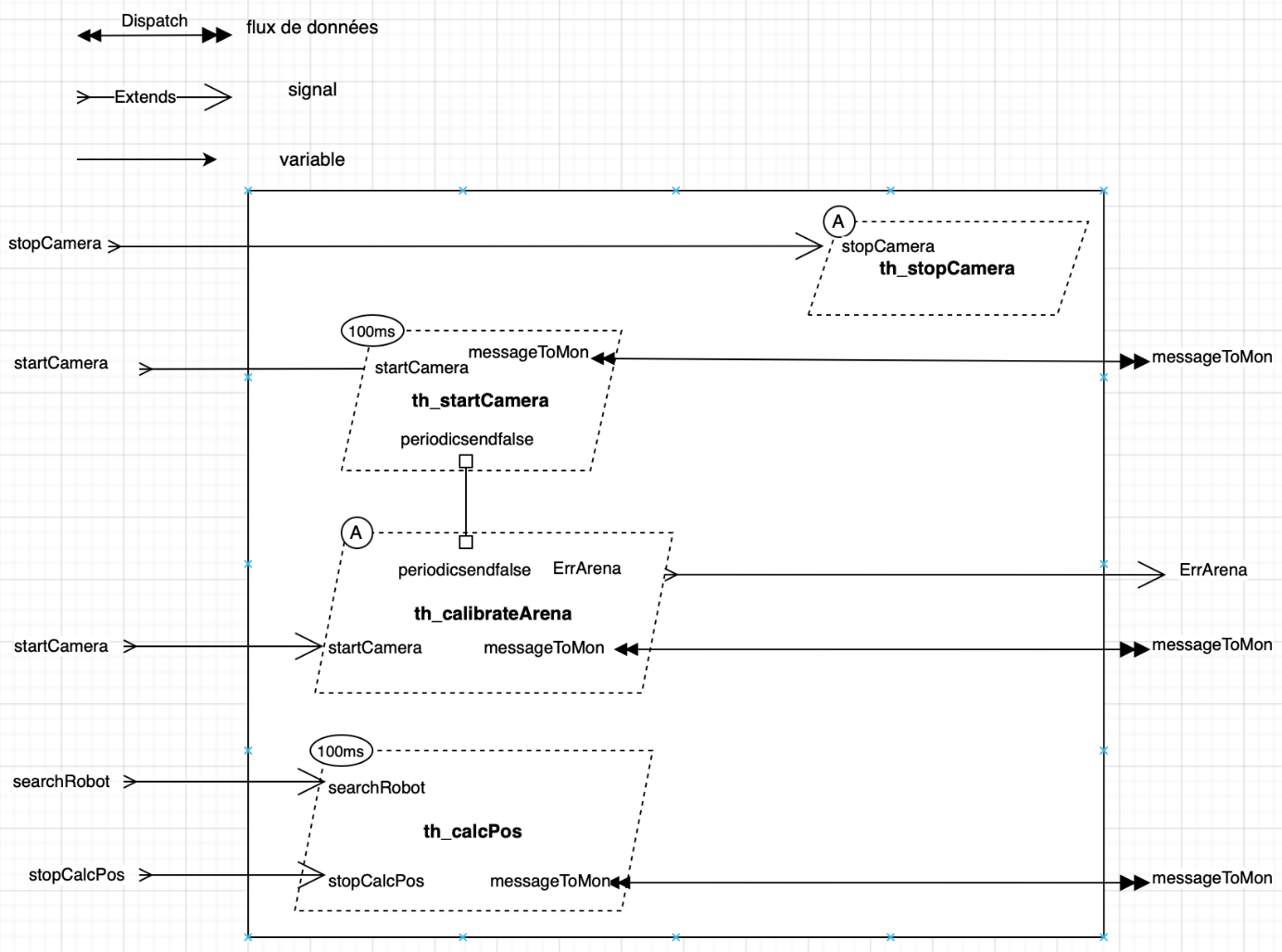


Fig. 5: Diagramme fonctionnel du groupe vision

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

Décrivez le comportement de chacun de vos threads avec des diagrammes d’activité. Ap- portez les explications qui vous semblent nécessaires pour comprendre votre conception.

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

Cette section est consacrée à la méthode pour passer d’un modèle AADL à un code sous Xenomai. Pour chacun des éléments AADL, vous expliquerez comment vous l’avez traduit en code et quels services de Xenomai vous avez utilisés en expliquant ce qu’ils font. Chaque élément devra être illustré avec des extraits de code de votre projet.

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).

Expliquer comment vous implémentez sous Xenomai l’instanciation et le démarrage d’un thread AADL.

Exemple de réponse : 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\_server, sa déclaration est faite ligne 73 dans le fichier tasks.h

RT\_TASK th\_server;  
sa création ligne 102 de tasks.cpp lors de l’appel de

rt\_task\_create(&th\_server, "th\_server", 0, PRIORITY\_TSERVER, 0)

et son démarrage ligne 146 avec

rt\_task\_start(&th\_server, (void(\*)(void\*)) & Tasks::ServerTask, this)

1. Code à éxécuter

Comment se fait le lien sous Xenomai entre le thread et le traitement à exécuter.

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

Expliquer comment vous fixez sous Xenomai le niveau de priorité d’un thread AADL.

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.

Expliquer comment vous rendez périodique l’activation d’un thread AADL sous Xenomai.

1. *Donnée partagée*
2. Instanciation

Quelle structure instancie une donnée partagée ?

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

1. Accès en lecture et écriture

Comment garantissez-vous sous Xenomai l’accès à une donnée partagée ?

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).

Comment avez-vous instancié un port d’événement ?

1. Envoi d’un évènement

Quels services ont été employés pour signaler un événement ?

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

Comment avez-vous instancié un port d’événement ?

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

Donnez la solution retenue pour implémenter un port d’événement-données avec Xenomai.

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

Quels services avez-vous employés pour envoyer des données ?

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

Quels services avez-vous employés pour recevoir des données ?

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**

Cette partie sera à réaliser par la production d’une vidéo dans laquelle vous montrerez que chacune des fonctionnalités de 1 à 13 (communication et robot) sont opérationnelles. Vous n’êtes pas obligé de reprendre les fonctionnalités déjà faites (1,2,3, 4, 7, 10 et 12) sauf si vous les avez modifiées.

Faites une courte séquence vidéo pour chaque fonctionnalité et montrer clairement qu’elle respecte les besoins. 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

Table des illustrations

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

Table des annexes

1. Annexe 1 : Lien GitHubA
2. Annexe 2 : Lien Video sur DriveA

**Annexe 1 :** Lien GitHub :

**Annexe 2 :** Lien Vidéo Drive : https://drive.google.com/open?id=1Dk5C9FLq-bFZCgSRN2y9dTCk0oxM7fp4

