Rapport de projet DDS

**—**

Rayane Makri (robot, diagrammes d’activités, rédaction du compte-rendu)

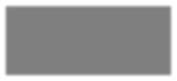
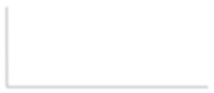
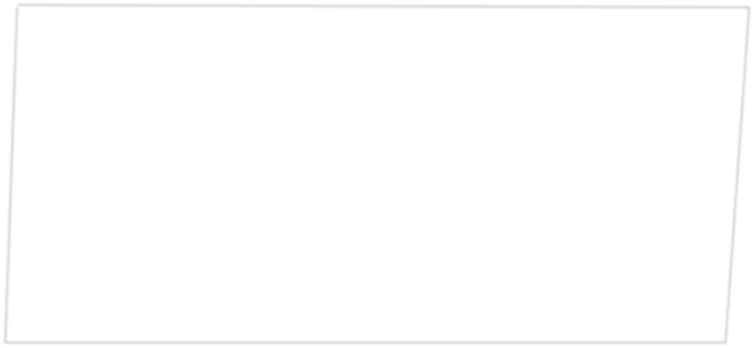
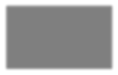
1. **Conception**

## 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)](#_bookmark0) en indiquant pour chaque groupe de threads les données et ports partagés. La figure [1.1](#_bookmark0) 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**).**

|  |  |  |
| --- | --- | --- |
| monitor | | outputstream |
|  |
|  |  | |

Fig. 1: Diagramme fonctionnel du système



inputStream

connectServer

th\_group\_vision

th\_group\_gestion\_moniteur

connecterServeur outputstream inputStream openComRobot

messageToMon

startRobot

th\_group\_gestion\_robot

openComRobot startRobot

messageToMon

ord

ordre

reponse

reponse

robot

re

Move

## 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](#_bookmark1) réalisée à partir du dossier de conception),
    - les diagrammes d’activité de chaque thread de ce groupe.

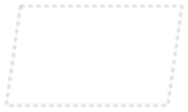
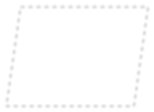
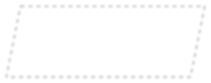
Décrivez tous les éléments (paramètres, variables, etc.) qui vous semblent pertinents pour comprendre les diagrammes.

### Diagramme fonctionnel du groupe gestion du moniteur

Exemple de diagramme fonctionnel pour le groupe de thread de gestion du moniteur.

Mettez à jour ce diagramme avec votre conception.

messageToMon



th\_sendToMon

A

th\_server

A

serverOK

connectServer

messageToMon

serverOK

outputstream

th\_receiveFromMon

A

OpenComRobot

serverOK

startRobot inputStream

connectServer

outputstream

openComRobot

inputStream

startRobot

Move

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

### Diagrammes d’activité du groupe gestion du moniteur

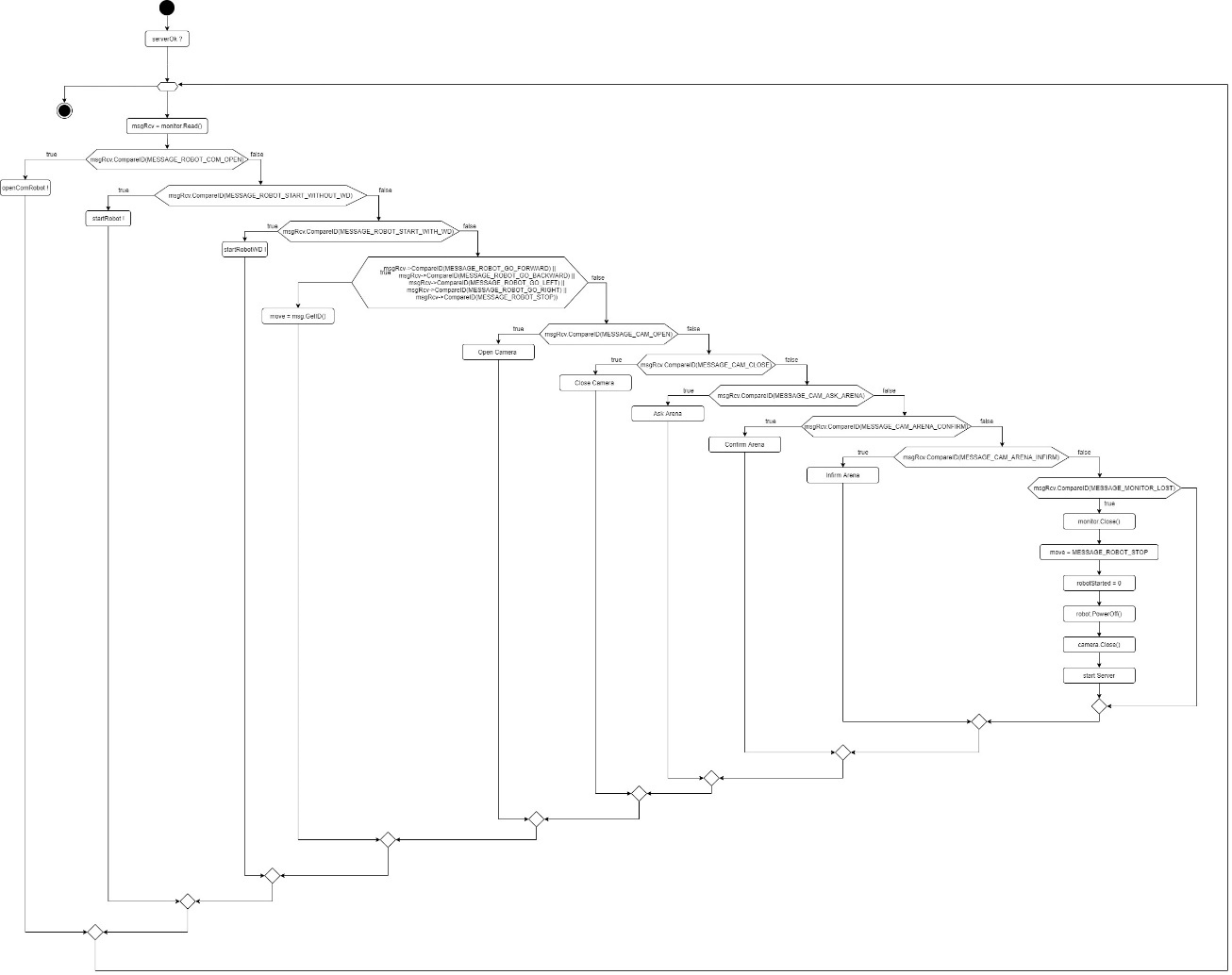


Fig. 3: Diagramme d’activité du thread th\_receiveFromMon

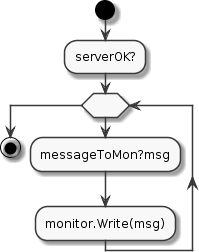


Fig. 4: Diagramme d’activité du thread th\_sendToMon

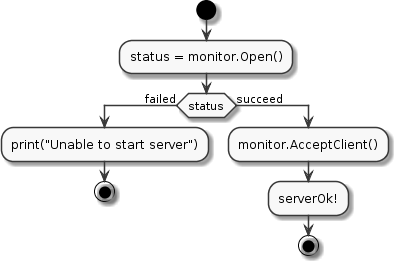


Fig. 5: Diagramme d’activité du thread th\_server

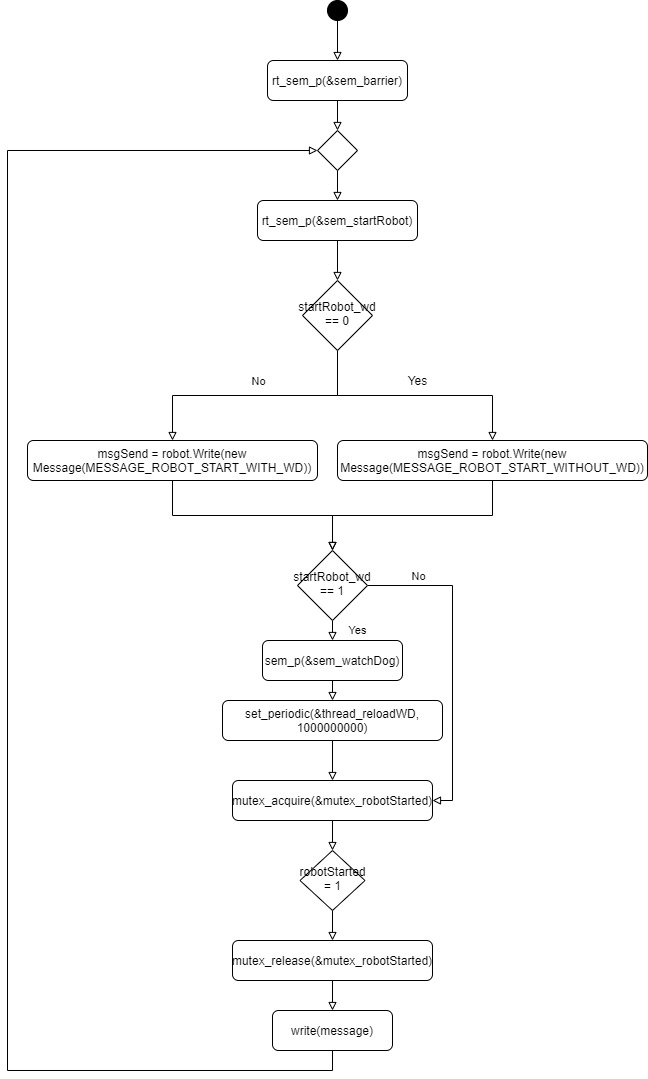
Pour le détail des messages concernant la caméra, voir les diagrammes du groupe vision

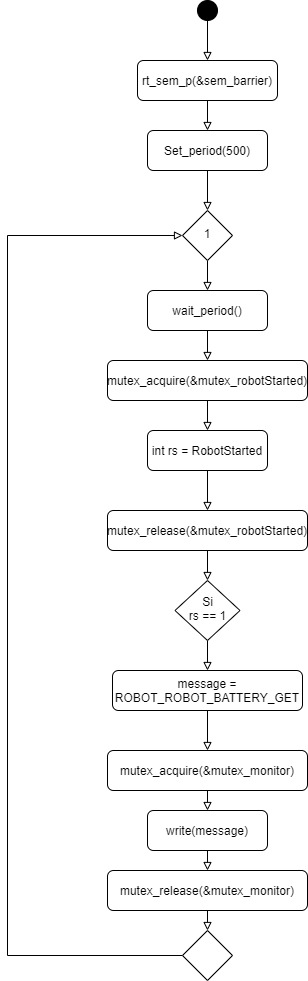
## Groupe de threads gestion du robot

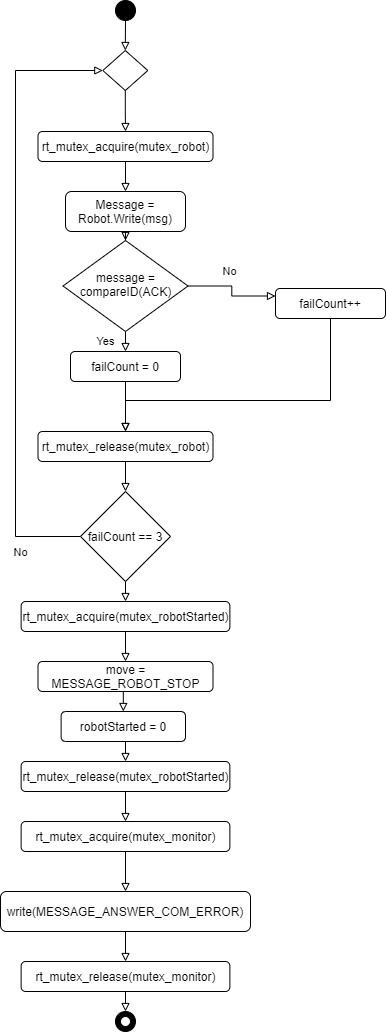
### Diagramme fonctionnel du groupe gestion robot

Ajoutez le diagramme fonctionnel du groupe de threads de gestion du robot.

### Diagrammes d’activité du groupe robot





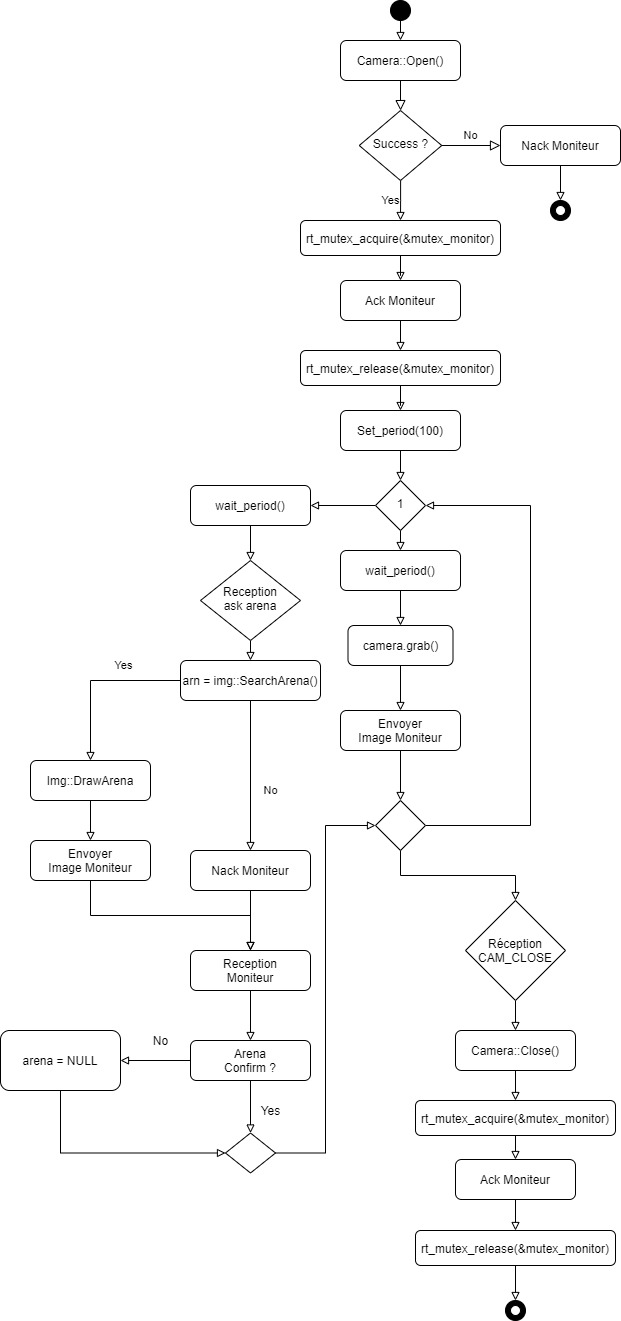


## Groupe de threads vision

### Diagramme fonctionnel du groupe vision

Ajoutez le diagramme fonctionnel du groupe de threads de vision.

### Diagrammes d’activité du groupe vision



## Transformation AADL vers Xenomai

## Thread

### Instanciation et démarrage

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)

### Code à exécuter

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

On fait le lien avec rt\_task start qui donne le nom de la méthode à exécuter lier au thread

### Niveau de priorités

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

On le donne au moment de rt\_task create en paramètres avec des variables définies au début du programme

### Activation périodique

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

On utilise rt\_task\_set\_periodic auquel on donne la tâche que l’on souhaite modifier (ou NULL si c’est le thread courant), la date avant laquelle exécuter le changmement de période et la périodicité en nanosecondes.

## Donnée partagée

### Instanciation

Quelle structure instancie une donnée partagée ?

Les données partagées sont le moniteur, le robot, la camera et l’arène qui ont chacun leur propre structure ainsi que les variables numéraires failCount, robotStarted et move.

### Accès en lecture et écriture

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

Grâce au mutex que l’on définie au début de l’exécution du programme. Avant d’accéder à une ressource, on vérifie que cette dernière est disponible en utilisant un mutex\_acquire. On libère la ressource quand on a fini avec un mutex\_release.

## Port d’événement

### Instanciation

Comment avez-vous instancié un port d’événement ?

Un thread est bloqué de base jusqu’à réception de l’évènement correspondant qui le débloque.

### Envoi d’un événement

Quels services ont été employés pour signaler un événement ?

On peut le faire à l’aide d’un sémaphore.

### Réception d’un événement

Comment se fait l’attente d’un événement ?

On le fait avec les méthodes rt\_sem\_p pour signaler l’attente du sémaphore et rt\_sem\_v qui signale l’évènement associé à ce sémaphore (pour une tâche, celle ayant la plus grande priorité) ou en rt\_sem\_broadcast pour le signaler à tout le monde.

## Ports d’événement-données

### Instanciation

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

A l’aide des données de type structures partagées (robot, moniteur ….) ainsi que les variables partagées (startRobot, failCount …)

### Envoi d’une donnée

Quels services avez-vous employés pour envoyer des données ?

Avec les fonctions associées au structure (par exemple robot.write) qui envoie un message à l’équipement associé. On peut également modifier directement les variables dont on parlait auparavant cependant on doit utiliser un mutex pour protéger la ressource. On peut enfin utiliser les queues pour communiquer des messages avec la fonction writeInQueue.

### Réception d’une donnée

Quels services avez-vous employés pour recevoir des données ?

On utilise un thread dédié pour la réception des messages provenant du moniteur. Ces messages sont alors traités dans ce thread. Pour la réception des messages provenant du robot, ces derniers sont toujours des réponses à des messages précédemment envoyés. On peut récupérer cette réponse immédiatement après l’envoi à l’aide du retour de la fonction d’envoi.