SWARMBOTS
Dossier de Conception
Incrément 2 – Version 2.01 – Révision 11

VERSION 2023

Responsable du document : Florentin LEPELTIER

État du document : Validé

AVERTISSEMENT :

Le présent document a été réalisé dans un cadre pédagogique relatif au projet ProSE A2 de l'option Systèmes Embarqués (SE) du cursus ingénieur de l'ESEO Angers. Il a été conçu par l'équipe A2 constituée des 7 membres (Mahery FONG, Louison LEGROS, Joshua MONTREUIL, Fatoumata TRAORÉ, Julien NICOT, Matéo RONDEAU et Florentin LEPELTIER). La réalisation a été accompagnée par Thales Services Numériques, entreprise partenaire de l'ESEO et considérée comme client dans le cadre du projet ProSE.

Le présent document est réalisé sous la direction de Jérome Delatour, il a été modifié et adapté par l'équipe A2 travaillant pour l'entreprise Thales Services Numériques dans le cadre pédagogique du projet ProSE de l'ESEO, année 2023. En dehors des activités pédagogiques de l'ESEO, ce document ne peut être diffusé ou recopié sans l'autorisation écrite de son propriétaire. Ce document est à destination de l'entreprise Thales Services Numériques, à qui sont cédés les droits de lecture et de modification dès lors de la livraison de la suite logicielle SWARMBOTS.

Date	Modifications apportées	Auteur	Version	Révision
12/06/2023	Modifications mineures à la suite de l'AN Code C.	Louison LEGROS & Joshua MONTREUIL	2.01	11
02/06/2023	Modifications AN Conception. Validation du document	Florentin LEPELTIER	2.00	10
31/05/2023	Complétion du dossier avant AN.	Joshua MONTREUIL	1.03	9
28/05/2023	Ajout des descriptions de la conception détaillée de SB_IHM.	Mahery FONG	1.02	8
27/05/2023	Archi détaillée en C.	Joshua MONTREUIL	1.01	7
26/04/2023	Modifications post audit consultatif	Florentin LEPELTIER	1.00	6
19/04/2023	Modifications mineures après une relecture du dossier pour l'audit consultatif avec Jérôme DELATOUR	Mahery FONG	0.04	5
19/04/2023	Ajout des diagrammes de séquence et MAE par l'équipe	Joshua MONTREUIL	0.03	4
16/04/2023	Ajout des descriptions textuelles des diagrammes de séquence (rédigé par Fatoumata).	Florentin LEPELTIER	0.02	3
16/04/2023	Ajout des descriptions textuelles des diagrammes de séquences (Rafraichir Ecran Commande, Vérifier Connection, Sélectionner un robot et Commander un robot). Ajout des descriptions textuelles des classes (ControllerRinger, ControllerCore, Camera, Radar et GUIRinger).	Joshua MONTREUIL	0.01	2
09/04/2023	Ajout des parties des grands principes de fonctionnement en rapport avec les CU, ajout des parties de la description des classes.	Joshua MONTREUIL	0.01	1
29/03/2023	Création du document	Florentin LEPELTIER	0.00	0

1. Introduction 7
1.1. Objet 7
1.2. Portée 7
1.3. Copyright 7
1.4. Définition, acronymes et abréviations 7
1.5. Références 8
1.5. Vue d'ensemble 9
1.6. Versions 9
2. Conception générale 10
2.1. Architecture candidate 10
2.2. Grands principes de fonctionnement 11
2.2.1. Piloter un essaim de robots 12
2.2.2. Initialiser le système 12
2.2.3. Initialiser un nouveau robot connecté 13
2.2.4. Récupérer Mode_Fonctionnement 14
2.2.5. Commander un robot 15
2.2.6. Sélectionner un robot 16
2.2.7. Déplacer un robot 17
2.2.8. Vérifier la présence d'un obstacle 18
2.2.9. Rafraîchir Ecran_Commande 19
2.2.10. Transmettre flux caméra 19
2.2.11. Vérifier connexion 20
2.2.12. Consulter les logs (Robot) 21
2.2.13. Consulter les logs (IHM) 22
2.2.14. Flusher les logs 23
2.2.15. Exporter les logs 24
2.2.16. Journaliser un évènement 24
2.2.17. Quitter SB_IHM 25
2.3. Description des composants 26
2.3.1. Description des types manipulés entre composants 26
2.3.2. Diagramme de classes de SWARMBOTS 27
2.3.3. Description des classes de SWARMBOTS 28
2.3.3.1. [IHM] La classe GUI 28
2.3.3.1.1. Philosophie de conception 28
2.3.3.1.2. Description structurelle 28
2.3.3.1.3. Description comportementale 30
2.3.3.2. [Object] La classe GUISecretary 31
2.3.3.2.1. Philosophie de conception 31
2.3.3.2.2. Description structurelle 31
2.3.3.2.3. Description comportementale 32
2.3.3.3. [Object] La classe Viewer 33
2.3.3.3.1. Philosophie de conception 33
2.3.3.3.2. Description structurelle 33
2.3.3.3.3. Description comportementale 34
2.3.3.4. [Object] La classe LogsManager 34
2.3.3.4.1. Philosophie de conception 34
2.3.3.4.2. Description structurelle 34
2.3.3.4.3. Description comportementale 35
2.3.3.5. [Object] La classe Robot 36
2.3.3.5.1. Philosophie de conception 36
2.3.3.5.2. Description structurelle 36

2.3.3.6. [Object] La classe ConfigReader 37
2.3.3.6.1. Philosophie de conception 37
2.3.3.6.2. Description structurelle 37
2.3.3.7. [Object] La classe ControllerCore 37
2.3.3.7.1. Philosophie de conception 37
2.3.3.7.2. Description structurelle 37
2.3.3.7.3. Description comportementale 38
2.3.3.8. [Object] La classe StateIndicator 39
2.3.3.8.1. Philosophie de conception 39
2.3.3.8.2. Description structurelle 39
2.3.3.8.3. Description comportementale 40
2.3.3.9. [Object] La classe Pilot 41
2.3.3.9.1. Philosophie de conception 41
2.3.3.9.2. Description structurelle 41
2.3.3.9.3. Description comportementale 41
2.3.3.10. [Object] La classe ControllerLogger 42
2.3.3.10.1. Philosophie de conception 42
2.3.3.10.2. Description structurelle 42
2.3.3.10.3. Description comportementale 43
2.3.3.11. [Object] La classe ControllerRinger 43
2.3.3.11.1. Philosophie de conception 43
2.3.3.11.2. Description structurelle 44
2.3.3.11.3. Description comportementale 44
2.3.3.12. [Object] La classe Radar 45
2.3.3.12.1. Philosophie de conception 45
2.3.3.12.2. Description structurelle 45
2.3.3.13. [Object] La classe Camera 45
2.3.3.13.1. Philosophie de conception 45
2.3.3.13.2. Description structurelle 45
2.3.3.13.3. Description comportementale 46
2.3.3.13. [Object] La classe GUIRinger 46
2.3.3.13.1. Philosophie de conception 46
2.3.3.13.2. Description structurelle 46
2.3.3.13.3. Description comportementale 47

3. Conception détaillée 48

3.1. Architecture physique 48
3.1.1. Architecture matérielle de SWARMBOTS 48
3.1.1.1. Schéma de SWARMBOTS 48

3.2. Conception détaillée de SB_C 48
3.2.1. Description des classes de SB_C 48
3.2.1.1. Schéma de SB_C 49
3.2.1.2. [Object] La classe Starter 49
3.2.1.2.1. Philosophie de conception 49
3.2.1.2.2. Description structurelle 49
3.2.1.2.3. Gestion du multitâche 49
3.2.1.2.4. Séquence de démarrage et arrêt de SB_C 50
3.2.1.3. [Com] La classe Postman 51
3.2.1.3.1. Philosophie de conception 51
3.2.1.3.2. Description structurelle 51
3.2.1.3.3. Gestion du multitâche 51
3.2.1.4. [Com] La classe Disptacher 52
3.2.1.4.1. Philosophie de conception 52
3.2.1.4.2. Description structurelle 52
3.2.1.4.3. Gestion du multitâche 52
3.2.1.5. [Com] La classe ProxyGui 52

3.2.1.5.1. Philosophie de conception 52
3.2.1.5.2. Description structurelle 52
3.2.1.6. [Com] La classe ProxyGuiSecretary 53
3.2.1.6.1. Philosophie de conception 53
3.2.1.6.2. Description structurelle 53
3.2.1.7. [Com] La classe ProxyGuiRinger 53
3.2.1.7.1. Philosophie de conception 54
3.2.1.7.2. Description structurelle 54
3.2.1.8. [Com] La classe ProxyLogsManager 54
3.2.1.8.1. Philosophie de conception 54
3.2.1.8.2. Description structurelle 54
3.2.1.9. [Alphabot2] La classe Buzzer <boundary> 55
3.2.1.9.1. Philosophie de conception 55
3.2.1.9.2. Description structurelle 55
3.2.1.10. [Alphabot2] La classe Motor <boundary> 55
3.2.1.10.1. Philosophie de conception 55
3.2.1.10.2. Description structurelle 55
3.2.1.11. [Alphabot2] La classe ServoMotor <boundary> 56
3.2.1.11.1. Philosophie de conception 56
3.2.1.11.2. Description structurelle 56
3.2.1.12. [Alphabot2] La classe Leds <boundary> 57
3.2.1.12.1. Philosophie de conception 57
3.2.1.12.2. Description structurelle 57
3.2.1.12.3. Gestion du multitâche 57
3.2.1.12.4. Description comportementale 57
3.2.2. Identification des accès concurrents côté SB_C 58

3.3. Conception détaillée coté SB_IHM 59
3.3.1. Description des classes de SB_IHM 59
3.3.1.1. Schéma de SB_IHM 59
3.3.1.1.1. HomeScreenActivity <<active>> 60
3.3.1.1.2. CommandScreenActivity <<active>> 60
3.3.1.1.3. LogsScreenActivity <<active>> 60
3.3.1.1.4. HomeScreenWaitingConnectionFragment 60
3.3.1.1.5. HomeScreenRobotConnectedFragment 60
3.3.1.1.6. HomeScreenErrorConnectingRobots 60
3.3.1.1.7. CommandScreenCameraRobot1Fragment 60
3.3.1.1.8. CommandScreenCameraRobot2Fragment 61
3.3.1.1.9. CommandScreenCameraOFFFragment 61
3.3.1.1.10. CommandScreenBlankSwitchFragment 61
3.3.1.1.11. CommandScreenSwitchFragment 61
3.3.1.1.12. CommandScreenRadarOnFragment 61
3.3.1.1.13. CommandScreenRadarOffFragment 61
3.3.1.1.14. CommandScreenRadarDisabledFragment 61
3.3.1.1.15. LogsScreenShowLogsFragment 62
3.3.1.1.16. LogsScreenNotShowLogsFragment 62
3.3.1.1.17. HomeScreenViewModel 62
3.3.1.1.18. CommandScreenViewModel 62
3.3.1.1.19. LogsScreenViewModel 62
3.3.1.2. [Object] La classe Starter 63
3.3.1.2.1. Philosophie de conception 63
3.3.1.2.2. Description structurelle 63
3.3.1.2.3. Gestion du multitâche 63
3.3.1.2.4. Séquence de démarrage et arrêt de SB_IHM 63
3.3.1.3. [Com] La classe Postman 64
3.3.1.3.1. Philosophie de conception 64
3.3.1.3.2. Description structurelle 65
3.3.1.4. [Com] La classe Dispatcher 66

3.3.1.4.1. Philosophie de conception 66
3.3.1.4.2. Description structurelle 66
3.3.1.4.3. Gestion du multitâche 66
3.3.1.5. [Com] La classe Protocol 67
3.3.1.5.1. Philosophie de conception 67
3.3.1.5.2. Description structurelle 67
3.3.1.6. [Com] La classe ProxyControllerCore 67
3.3.1.6.1. Philosophie de conception 67
3.3.1.6.2. Description structurelle 67
3.3.1.7. [Com] La classe ProxyControllerRinger 68
3.3.1.7.1. Philosophie de conception 68
3.3.1.7.2. Description structurelle 68
3.3.1.8. [Com] La classe ProxyControllerLogger 68
3.3.1.8.1. Philosophie de conception 68
3.3.1.8.2. Description structurelle 68
3.3.1.9. [Com] La classe ProxyPilot 69
3.3.1.9.1. Philosophie de conception 69
3.3.1.9.2. Description structurelle 69
3.3.1.10. [Com] La classe ProxyCamera 70
3.3.1.10.1. Philosophie de conception 70
3.3.1.10.2. Description structurelle 70
3.3.2. Identification des accès concurrents côté SB_IHM 70

3.4. Protocole de communication 71
3.4.1. Protocole de communication entre SB_C et SB_IHM 71
3.4.1.1. Formalisation du protocole 71
3.3.1.2. Types de messages 71
3.3.1.3. Détails des messages 72
3.3.1.4. Exemples 72
3.3.1.4.1. ASK_AVAILABILITY 72
3.3.1.4.2. SET_AVAILABILITY 72
3.3.1.4.3. ASK_CMD 73
3.3.1.4.4. SET_STATE 73
3.3.1.4.5. ASK_MODE 73
3.3.1.4.6. SET_MODE 74
3.3.1.4.7. ASK_LOGS 74
3.3.1.4.8. SET_LOGS 74
3.3.1.4.9. ALERT 75
3.3.1.4.10. ASK_TO_DISCONNECT 75
3.3.1.4.11. ACK_DISCONNECTION 75
3.3.1.4.12. SET_RADAR 76
3.3.1.4.13. SET_CURRENT_TIME 76
3.3.1.4.14. SET_IP_PORT 76
3.3.1.4.15. LOGS_RECEIVED 77

4. Dictionnaire de domaine 77

4.1. Définitions 77

1. Introduction

1.1. Objet

Ce dossier de conception a pour objectif de rassembler toute la conception du logiciel SWARMBOTS. Il permettra à l'équipe ProSE A2 de développer le logiciel ainsi que d'élaborer des tests.

Les éléments de conception présentés dans ce document ont été déterminés à la suite de l'étude du dossier de spécification [[ProSE]_A2_SPEC_2024].

Ce dossier de conception suit les recommandations de la norme IEEE 830 [IEEE-830_1998]. Il utilise des schémas et illustrations respectant la norme UML en version 2.5 [UML_2.5_2015]. Il respecte les exigences du Plan d'Assurance Qualité Logicielle (PAQL) défini par l'équipe ProSE A2 [ProSE]_A2_ PAQL_2024.

1.2. Portée

Ce document décrit les éléments de conception du Système à l'Etude (SaE). Il est destiné :

- À l'équipe de développement C et celle de développement Android afin de préciser l'implémentation des objets constituants le SaE.
- Aux testeurs, afin qu'ils puissent élaborer les tests adéquats vérifiant la philosophie de conception adoptée par l'équipe.
- Aux auditeurs de la société FORMATO lors de leurs différents suivis du projet.
- Au client pour que le cadre du projet et la direction prise par l'équipe soient claire et dans la continuité des spécifications.

1.3. Copyright

1.4. Définition, acronymes et abréviations

Les abréviations utilisées dans le présent document sont répertoriées et expliquées dans le tableau présenté ci-dessous. Les termes utiles pour interpréter correctement ce dossier de spécifications sont définis dans le dictionnaire de domaine présent dans ce dossier dans la partie 4.

Acronymes, abréviations	Définition
Client	Société Thales Services Numériques
CSI (Camera Serial Interface)	Connecteur utilisé et implémenté sur les Raspberry Pi pour s'interfacer avec des caméras utilisant la même connectique.
CU	Cas d'Utilisation
ENT (Environnement Numérique de Travail)	Environnement de travail utilisé par l'équipe dans le cadre du projet ProSE afin d'y centraliser les documents du projet.
GPIO (General Purpose Input Output)	Terme désignant les broches d'entrées / sorties de la Raspberry Pi.
ID	Identifiant.
IEEE (Institute of Electrical and Electronics Engineers)	Association professionnelle internationale définissant entre autres des normes dans le domaine informatique et électronique.

IHM (Interface Homme Machine)	Moyens permettant aux utilisateurs de SWARMBOTS d'interagir avec SWARMBOTS.
IP (Internet Protocol)	Famille de protocoles de communication de réseaux informatiques conçus pour être utilisés sur internet.
IR (Infra Rouge)	Infrarouge, rayonnement électromagnétique.
LED (Light-Emitting Diode)	Diode électroluminescente.
MAC (Media Access Control)	Adresse/identifiant physique stocké dans une carte réseau ou une interface réseau.
N.A	Non Applicable
PWM (Pulse Width Modulation)	Modulation de largeur d'impulsions, génération de signaux pseudo analogique à partir d'un environnement numérique.
RGB (Red Green Blue)	Rouge Vert Bleu, espace de couleur défini par la Commission internationale de l'éclairage, permet de quantifier la couleur.
SàE (Système à l'étude)	Il s'agit de l'ensemble des logiciels SWARMBOTS_CONTROLEUR et ***SB_IHM***.
SAV	Service Après-Vente
SB	SwarmBots
SB_C	SwarmBots_Controleur, contrôleur utilisé pour les robots du SàE.
SB_IHM	SwarmBots_IHM. Interface Homme-Machine utilisée pour le SàE.
UDP (User Datagram Protocol)	Protocole de communication utilisé sur internet pour les transmissions sensibles au temps (vidéo etc). Il fournit une communication rapide car il n'y a pas d'établissement formel de connexion avant le transfert de données.
UML (Unified Modeling Language)	Notation graphique normalisée, définie par l'OMG et utilisée en génie logiciel.
WIFI (Wireless Fidelity)	Technologie de connexion sans fil.

1.5. Références

Voici un tableau récapitulatif des documents utilisés pour le dossier de spécifications ainsi que les liens permettant d'accéder aux fichiers :

Références	Description
[[ProSE]_A2_CR_01_2024]	ROBOMATIC, « compte rendu de réunion avec M. RAJAOARISOA, M. ROCOURT, M.STAMBOULI en date du 09/02/2023. » ENT rubrique gestion de projet – Equipe ProSE A2, 2023.
[[ProSE]_A2_CR_09_2024]	ROBOMATIC, « compte rendu de réunion avec M. RAJAOARISOA, M. ROCOURT, M.STAMBOULI en date du 09/03/2023. » ENT rubrique gestion de projet – Equipe ProSE A2, 2023.
[[ProSE]_fonct_client]	Document de spécification des fonctionnalités développées au cours du projet. Disponible sur l'ENT rubrique gestion de projet - M.LEPELTIER, 2023.
[Alphabot2-user-manual-en_2023]	Documentation technique de l'alphabot 2 disponible sur le site http://waveshare.com (Alphabot2-user-manual-en.pdf) et sur l'ENT rubrique specification/doc - Waveshare, juillet 2019.
[GA12-N20_2023]	Documentation technique des moteurs du robot. Disponible sur l'ENT rubrique specification/doc - Handson Technology, 2023.

[IEE Std 802.11b-1999 _2023]	Standard pour les comunications WIFI. Disponible sur l'ENT rubrique specification/doc - IEE, 2023.
[raspberry-pi-3-b-plus-product-brief_2023]	Brève documentation technique de la raspberry 3B+. Détaille les spécifications techniques générales de la carte. Disponible sur l'ENT rubrique specification/doc, raspberrypi.org, 2023.
[RPi_Camera_(B)_2023]	Caméra adaptée à la raspberry de waveshare. Disponible sur l'ENT rubrique specification/doc - raspberrypi.com, 2023.
[ST188-EN-2_2023]	Documentation technique du capteur IR. Disponible sur l'ENT rubrique specification/doc - npec.com, 2023.
[SG90_2023]	Documentation technique des servo motor. Disponible sur l'ENT rubrique specification/doc - luxoparts, 2023.
[SMA13_2023]	Documentation technique du buzzer. Disponible sur l'ENT rubrique specification/doc - cnmic.com, 2023.
[UML 2.5.1_ 2017].	OMG, "Unified Modeling Language", version 2.5.1, https://www.omg.org/spec/UML/2.5.1/, 2017.
[WS2812B_2023]	Documentation technique des LEDs adressables RGB. Disponible sur l'ENT rubrique specification/doc - worldSemi, 2023.

1.5. Vue d'ensemble

Ce document de conception est structuré en plusieurs parties :

- Une première partie concerne la conception générale du prototype de SWARMBOTS. Cette partie présente l'Architecture Candidate et donne les grands principes de fonctionnement de notre projet. Elle détaille ensuite chaque composant du système, en présentant pour chacun leur description structurelle. Une description comportementale est présente pour les composants actifs qui sont également représentés sous forme de machines à états.
- Une deuxième partie concerne la conception détaillée. Cette partie présente les diagrammes de classe du SàE, ainsi que la description des ses classes de communication. Cette partie présente enfin le protocole de communication et la gestion du multitâche au sein du logiciel.
- En troisième partie est présenté le Dictionnaire de domaine.

Il est bon de noter que pour le développement logiciel, des règles spécifiques sont définies dans le PAQL [ProSE_A2_PAQL_2024] quant au nommage des variables. Dans un souci de simplification, ce dossier ne respecte pas l'ensemble des règles définies, notamment au niveau des noms de variables et objets, qui sont exprimés CamelCase dans l'ensemble de l'architecture candidate.

1.6. Versions

Les logiciels SB_C et SB_IHM de SWARMBOTS seront développés en 2 incréments. Les fonctionnalités implémentées pour l'incrément 1 et l'incrément 2 sont respectivement détaillées dans le document [ProSE_fonct_client] fourni par l'équipe ProSE A2.

2. Conception générale

2.1. Architecture candidate

Dans le diagramme suivant, nous présentons l'Architecture Candidate de notre Système à l'Étude. Les différents objets y sont présents ainsi que les interactions qu'ils entretiennent. On peut également observer les objets actifs ainsi que l'organisation du logiciel.

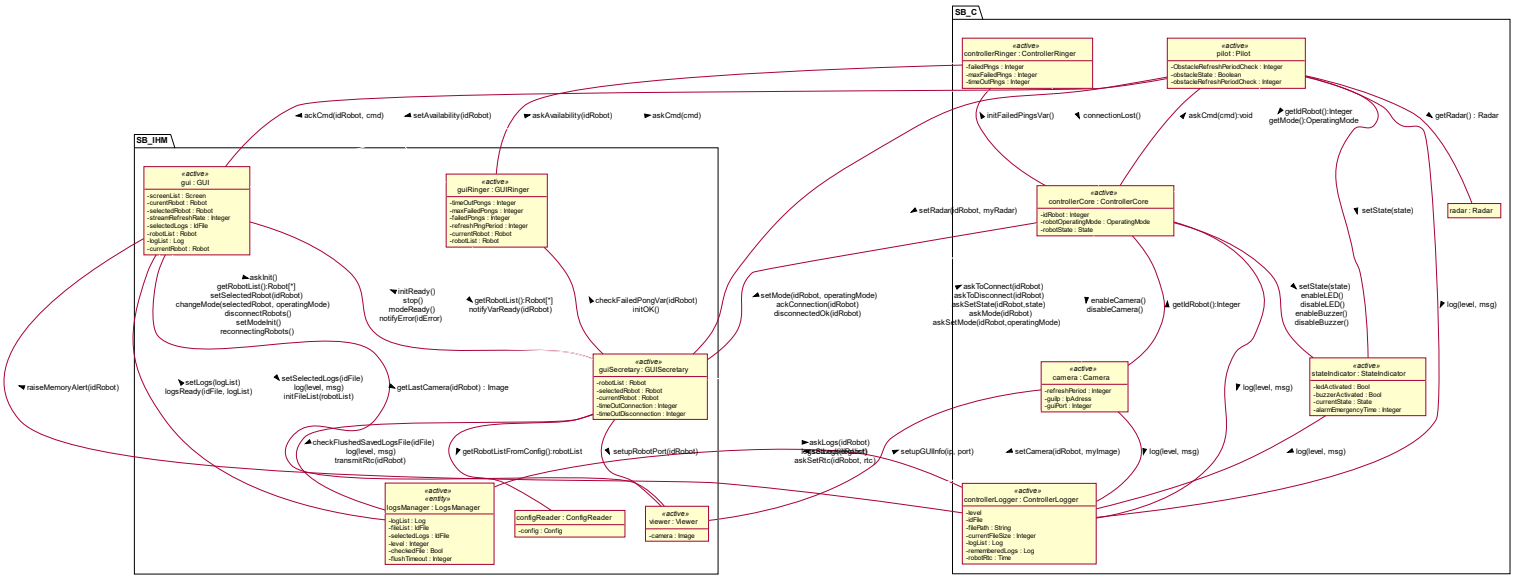


Figure 1 – Architecture Candidate de SWARMBOTS représentée par un diagramme de communication

Les objets suivants forment l'application SB_IHM, destinée à être exécutée sur une tablette Android :

-GUI : Cet objet gère l'affichage des différents écrans.

-GUISecretary : Cet objet gère la logique métier de la manipulation de la liste de robots.

-ConfigReader : Cet objet gère la lecture du fichier de configuration.

-GUIRinger : Cet objet gère l'établissement et la vérification de la connexion entre SB_C et SB_IHM.

-LogsManager : Cet objet gère le stockage des logs de l'IHM ainsi que la récupération des logs des robots connectés.

-Viewer : Cet objet gère la récupération des flux vidéo et radar des robots connectés.

Les objets suivants forment l'application SB_C, destinée à être exécutée sur une une raspberry 3B+ dans le cadre de ce prototype :

-ControllerCore : Cet objet gère la configuration globale du robot ainsi que les requêtes de connexion arrivant de SB_IHM.

-Pilot : Cet objet gère le déplacement du robot et envoie le flux radar à SB_IHM.

-StateIndicator : Cet objet gère l'affichage de l'état du robot (en attente de connexion, connecté, sélectionné, urgence, non sélectionné).

-Radar : Cet objet gère l'état du périphérique radar.

-Camera : Cet objet gère l'envoi du flux vidéo vers SB_IHM.

-**ControllerRinger** : Cet objet gère la vérification de la connexion entre SB_IHM et SB_C.

-ControllerLogger : Cet objet gère la journalisation des événements au sein de SB_C.

2.2. Grands principes de fonctionnement

Cette partie présente le fonctionnement général de SWARMBOTS. Cela est représenté sous forme de diagrammes de séquence. Pour chaque diagramme de séquence, une histoire nominale est racontée, les erreurs ne sont pas gérées dans ce type de diagramme.

Voici la liste des diagrammes de séquences présentés dans la suite de ce document :

- Piloter un essaim de robots.
- Initialiser le système.
- Initialiser un nouveau robot.
- Récupérer Mode_Fonctionnement.
- Commander un robot.
- Sélectionner un robot.
- Déplacer un robot.
- Vérifier la présence d'un obstacle.
- Rafraichir Ecran_Commande.
- Transmettre flux caméra.
- Vérifier connexion.
- Consulter les logs (Robot).
- Consulter les logs (IHM).
- Flusher les logs.
- Exporter les logs.
- Journaliser un évènement.
- Quitter SB_IHM

2.2.1. Piloter un essaim de robots

Ce diagramme représente le scénario nominal du CU stratégique du dossier de spécifications [[ProSE]_A2_SPEC_2024]. L'utilisateur démarre SB_IHM afin de sélectionner un robot et de pouvoir le commander. Il peut également consulter les logs des robots ou de l'IHM et quitter le logiciel s'il le souhaite.

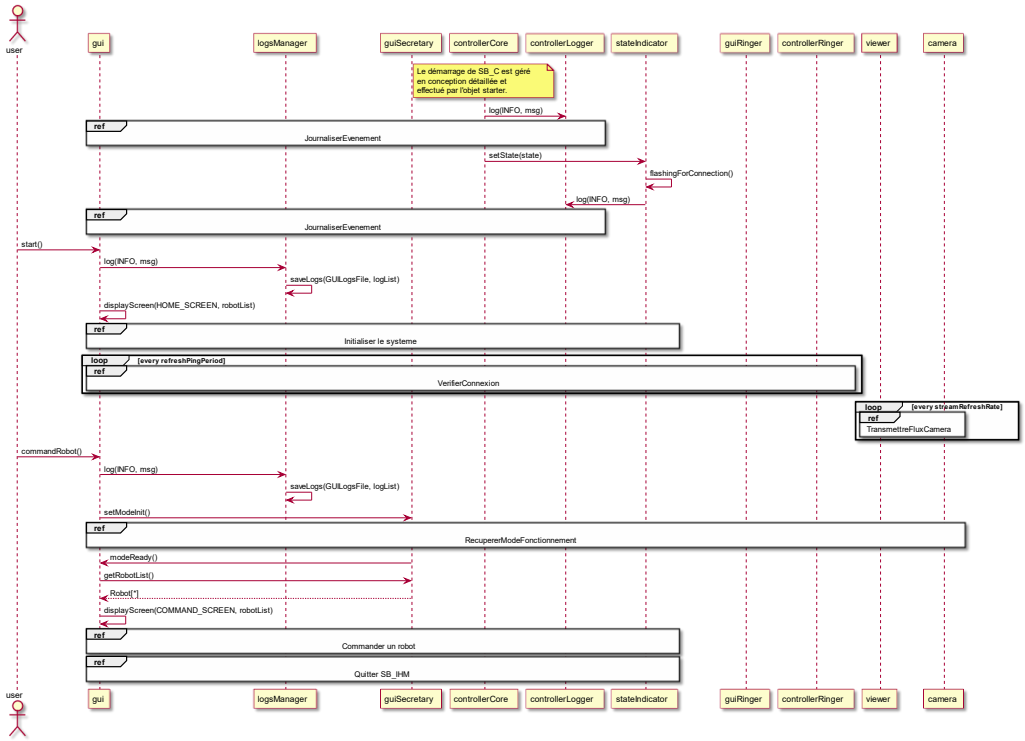


Figure 2 – Diagramme de séquence de : Piloter un essaim de robots

2.2.2. Initialiser le système

Ce diagramme représente le scénario nominal du CU « Initialiser le système » du dossier de spécifications [[ProSE]_A2_SPEC_2024]. SB_IHM s'initialise à l'aide du fichier de configuration qu'il vient lire. Pour chaque robot (SB_C) défini dans le fichier de configuration, il établit une connexion et gère l'initialisation du système. L'écran d'accueil est finalement affiché à l'écran. Si un fichier de configuration non valide est fourni, l'utilisateur est averti via un message à l'écran (variante du scénario nominal non représentée ici).

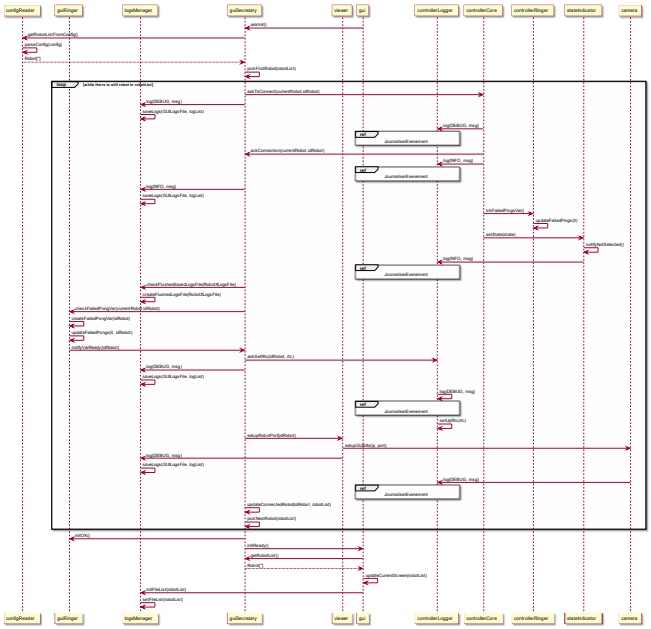


Figure 3 – Diagramme de séquence de : Initialiser le système

2.2.3. Initialiser un nouveau robot connecté

Ce diagramme représente le scénario du CU « Initialiser un nouveau robot connecté » du dossier de spécification [[ProSE]_A2_SPEC_2024].

Un robot (SB_C) se connecte à SB_IHM, cette action est confirmée à l'utilisateur par un allumage des leds du robot en mode « non_sélectionné ».

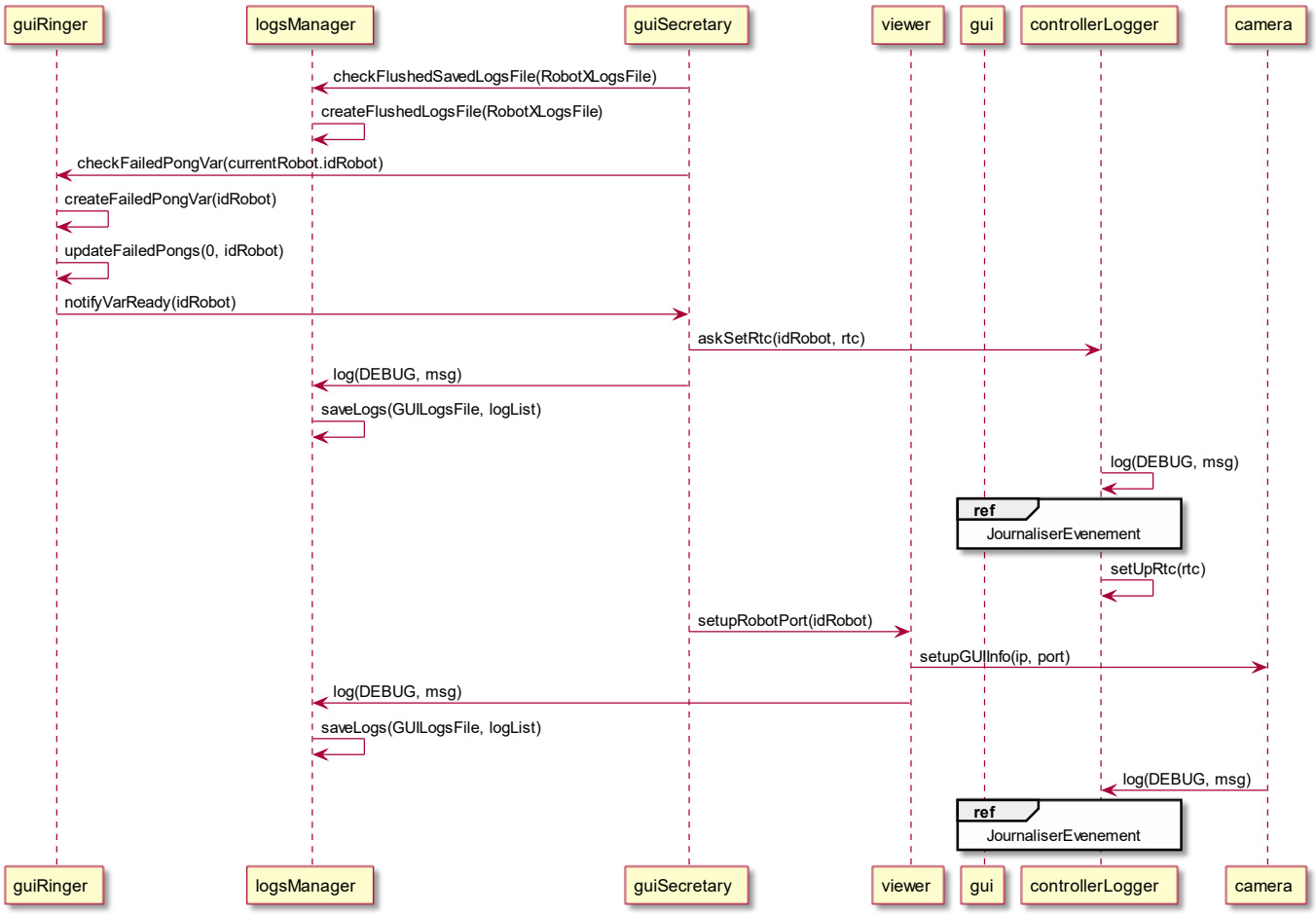


Figure 4 – Diagramme de séquence de : Initialiser un nouveau robot connecté (également présenté dans initialiser)

2.2.4. Récupérer Mode_Fonctionnement

Ce diagramme représente la fonction « récupérer Mode_Fonctionnement » du dossier de spécification [[ProSE]_A2_SPEC_2024]. La classe GUISecretary parcourt la liste de robot afin de récupérer leurs modes de fonctionnement.

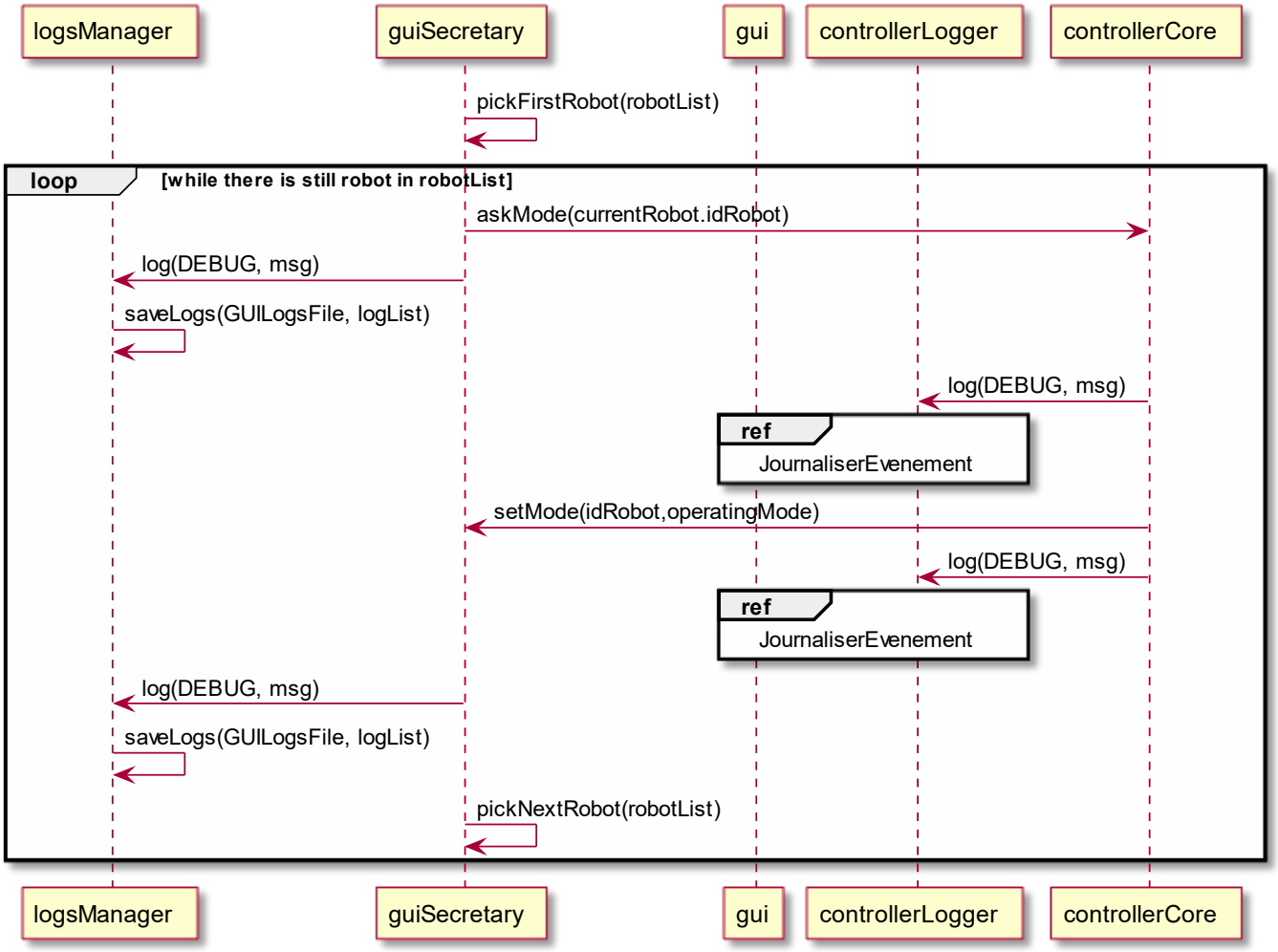


Figure 5 – Diagramme de séquence de vérifier : Mode_Fonctionnement

2.2.5. Commander un robot

Ce diagramme représente le scénario du CU « Commander un Robot » du dossier de spécification [[ProSE]_A2_SPEC_2024]. L'utilisateur sélectionne un robot qu'il souhaite contrôler. Cela déclenche le diagramme de séquence « Sélectionner un robot ». Si l'utilisateur, après sélection d'un robot, décide de déplacer le robot, l'action déclenche le diagramme de séquence « Déplacer un robot ». Sinon, si l'utilisateur décide d'éditer un mode de fonctionnement du robot qu'il a sélectionné, alors ce mode de fonctionnement est premièrement mis à jour du côté de SB_IHM , puis du côté de SB_C.

La répartition de l'objet controllerCore pour transmettre le changement du mode de fonctionnement aux objets concernés se fait par la fonction applyEditMode(operatingMode), avec en paramètre la variable mode de fonctionnement. En fonction du mode de fonctionnement modifié, pour :

- Activation du mode de fonctionnement de la caméra : enableCamera() sur l'objet Camera.
- Désactivation du mode de fonctionnement de la caméra : disableCamera() sur l'objet Camera.
- Activation du mode de fonctionnement de la led : enableLED() sur l'objet StateIndicator.
- Désactivation du mode de fonctionnement de la led : disableLED() sur l'objet StateIndicator.
- Activation du mode de fonctionnement du buzzer : enableBuzzer() sur l'objet StateIndicator.
- Désactivation du mode de fonctionnement du buzzer : disableBuzzer() sur l'objet StateIndicator.

Enfin, si l'utilisateur souhaite changer de robot sélectionné, alors on déclenche à nouveau le diagramme de séquence « Sélectionner un robot ».

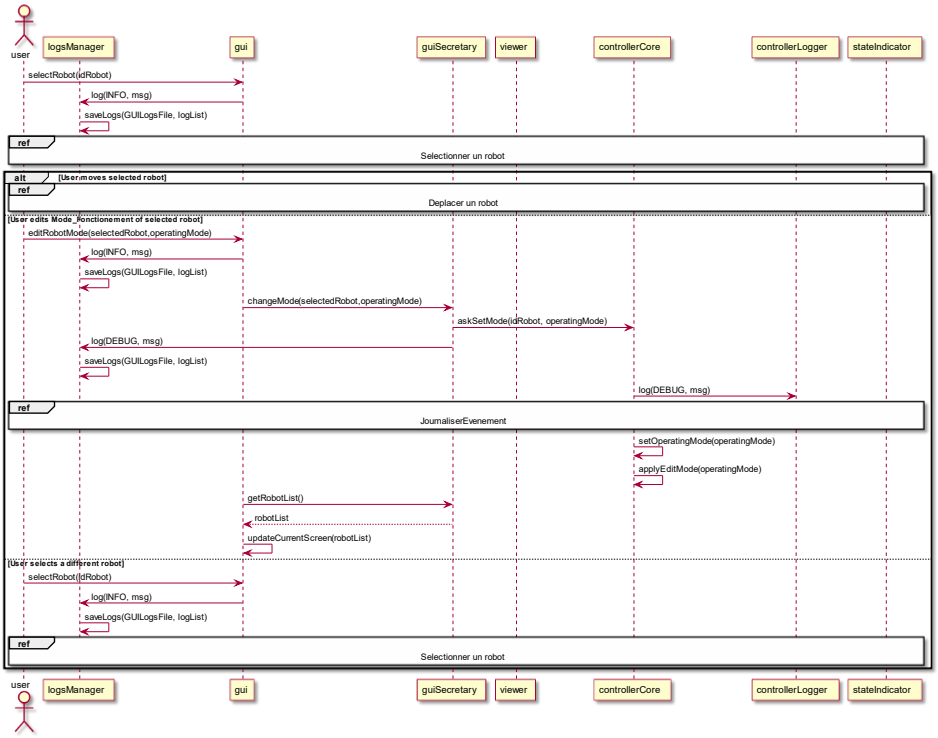


Figure 6 – Diagramme de séquence de : Commander un robot

2.2.6. Sélectionner un robot

Ce diagramme représente le scénario du CU « Sélectionner un Robot » du dossier de spécification [[ProSE]_A2_SPEC_2024]. La sélection d'un robot par l'utilisateur sur le diagramme de séquence parent déclenche ce diagramme de séquence. SB_IHM enregistre le robot qui est sélectionné puis notifie au robot en question qu'il est sélectionné. Par la suite, le robot notifie à l'utilisateur qu'il est sélectionné par un changement de couleur de ses leds qui passent au bleu.

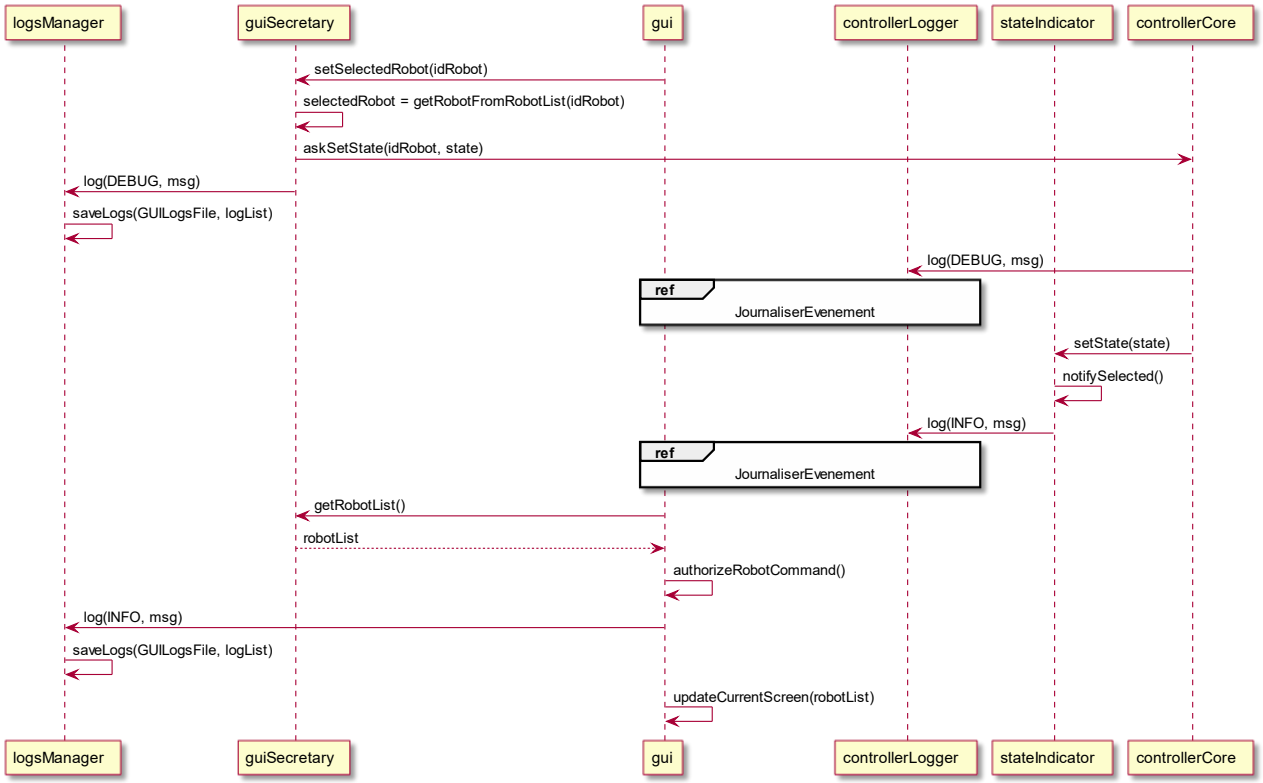


Figure 7 – Diagramme de séquence : Sélectionner un robot

2.2.7. Déplacer un robot

Ce diagramme représente le scénario du CU « Déplacer un robot » du dossier de spécification [[ProSE]_A2_SPEC_2024]. L'utilisateur demande à déplacer le robot sélectionné. SB_IHM envoie la commande à SB_C qui l'acquitte et l'exécute par la suite. On notera qu'ici, l'envoi d'une seule commande est représenté. Dans le CU "Déplacer un robot", on envoie explicitement une commande cmd_arret (ou stop) non représentée ici.

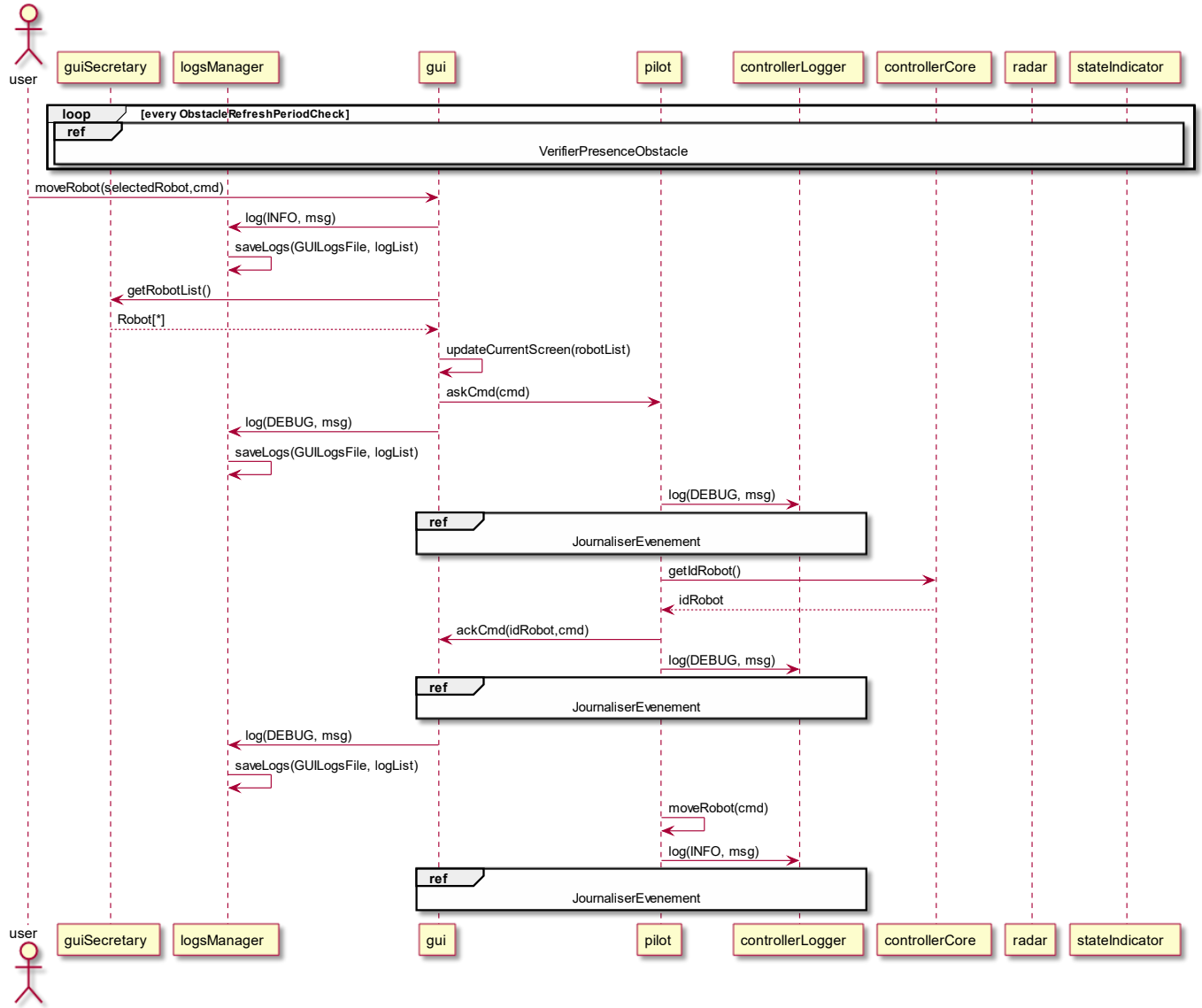


Figure 8 – Diagramme de séquence de : Déplacer un robot

2.2.8. Vérifier la présence d'un obstacle

Ce diagramme représente le scénarion du CU « Vérifier la présence d'un obstacle » du dossier de spécification [[ProSE]_A2_SPEC_2024]. Lors du déplacement du robot vers l'avant, Pilot va aller vérifier périodiquement l'état du radar. En cas d'une détection d'obstacle par le radar, Pilot arrêtera le robot et se mettra dans l'état « emergency » et aura ses leds en rouge.

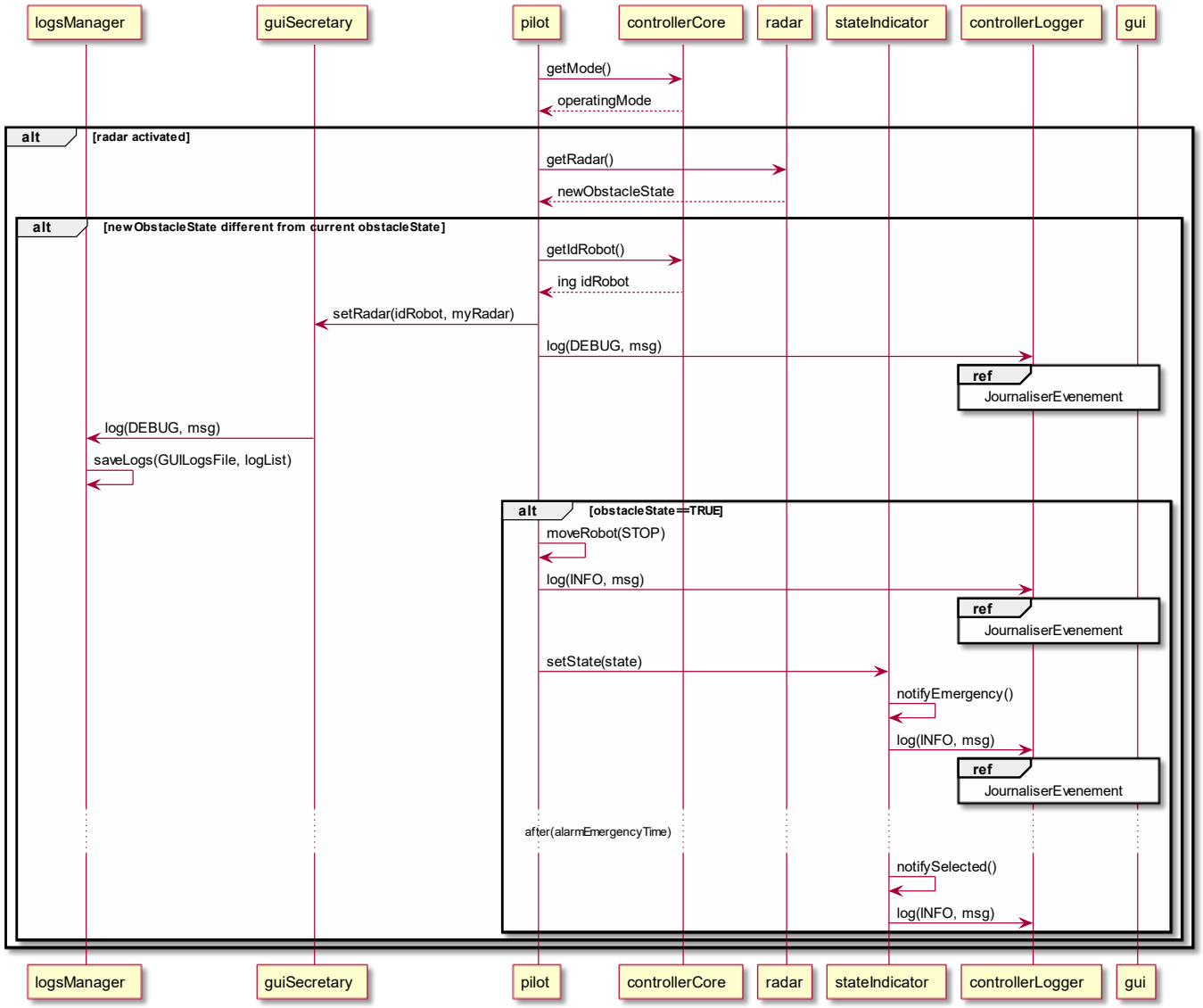


Figure 9 – Diagramme de séquence de : Vérifier la présence d'un obstacle

2.2.9. Rafraîchir Ecran_Commande

Ce diagramme représente le scénario du CU « Rafraichir Ecran_Commande » du dossier de spécification [[ProSE]_A2_SPEC_2024]. Tous les *Temps_Rafraichissement*, ici « streamRefreshRate », GUI vient récupérer les données caméra stockées par Viewer pour rafraichir le flux camera de chaque robot.

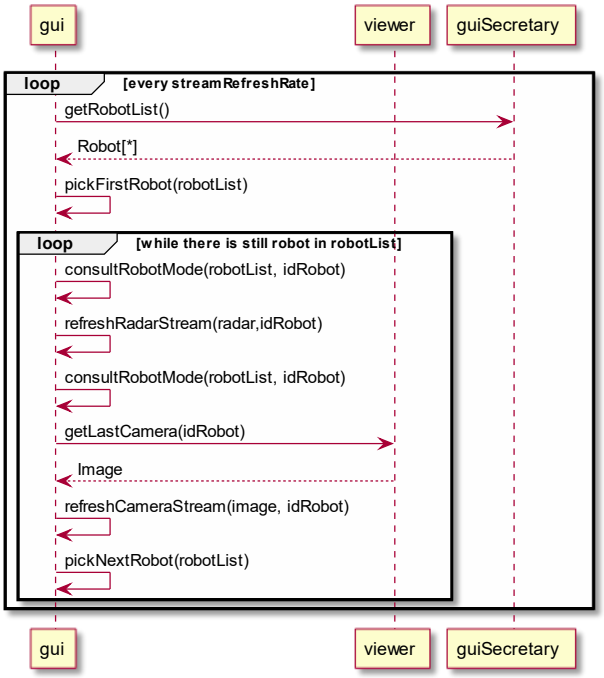


Figure 10 – Diagramme de séquence de : Rafraichir Ecran_Commande

2.2.10. Transmettre flux caméra

Ce diagramme représente le scénario du CU « Transmettre flux camera » du dossier de spécification [[ProSE]_A2_SPEC_2024]. Tous les streamRefreshRate (boucle représentée sur le diagramme de séquence parent), l'objet Camera change la valeur de l'attribut camera de l'objet Viewer.

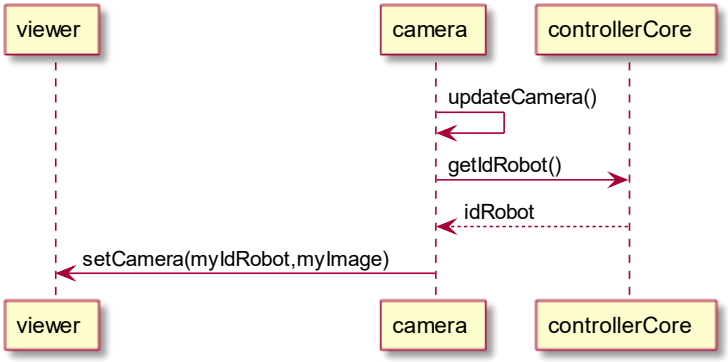


Figure 11 – Diagramme de séquence de : Transmettre flux caméra

2.2.11. Vérifier connexion

Ce diagramme représente le scénario du CU « Vérifier connexion » du dossier de spécification [[ProSE]_A2_SPEC_2024]. À chaque *refreshPingPeriod*, la classe GUIRinger vérifie par un « ping » si les robots connectés le sont toujours. Lorsque la classe GUIRinger reçoit un « pong » du robot ciblé, la classe passe à l'envoi d'un « ping » au robot suivant. La réception d'un « ping » du côté de la classe controllerRinger entraine une réinitialisation de sa var_connexion à 0. De même pour GUIRinger, la réception d'un « pong » réinitialise la var connexion du robot associé.

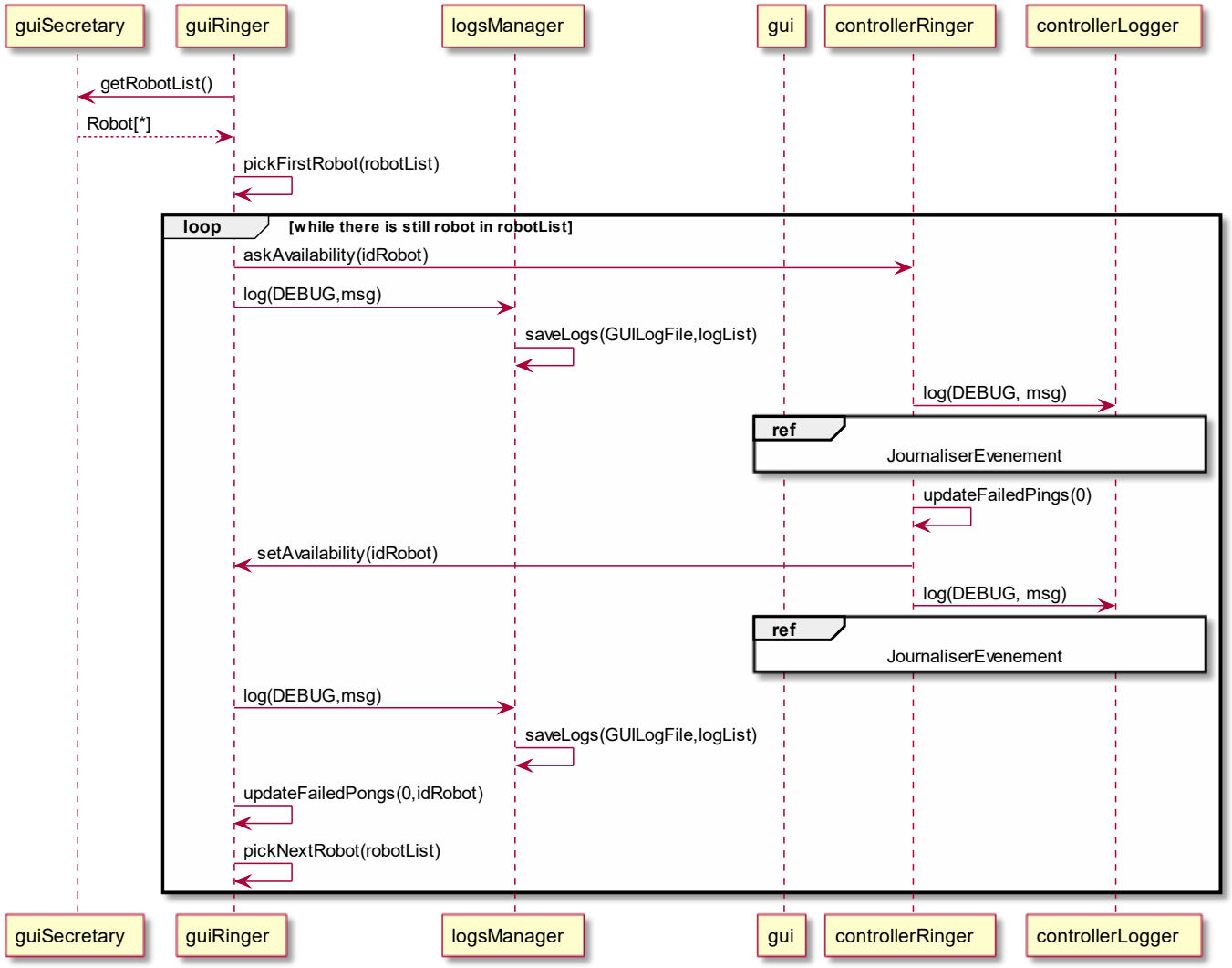


Figure 12 – Diagramme de séquence de : Vérifier connexion

2.2.12. Consulter les logs (Robot)

Ce diagramme représente le scénario du CU « Consulter les logs » du dossier de spécification [[ProSE]_A2_SPEC_2024]. Il représente le cas où un robot est sélectionné depuis l'IHM. Dans cette situation, SB_IHM récupère les logs (flush) du robot concerné par la sélection.

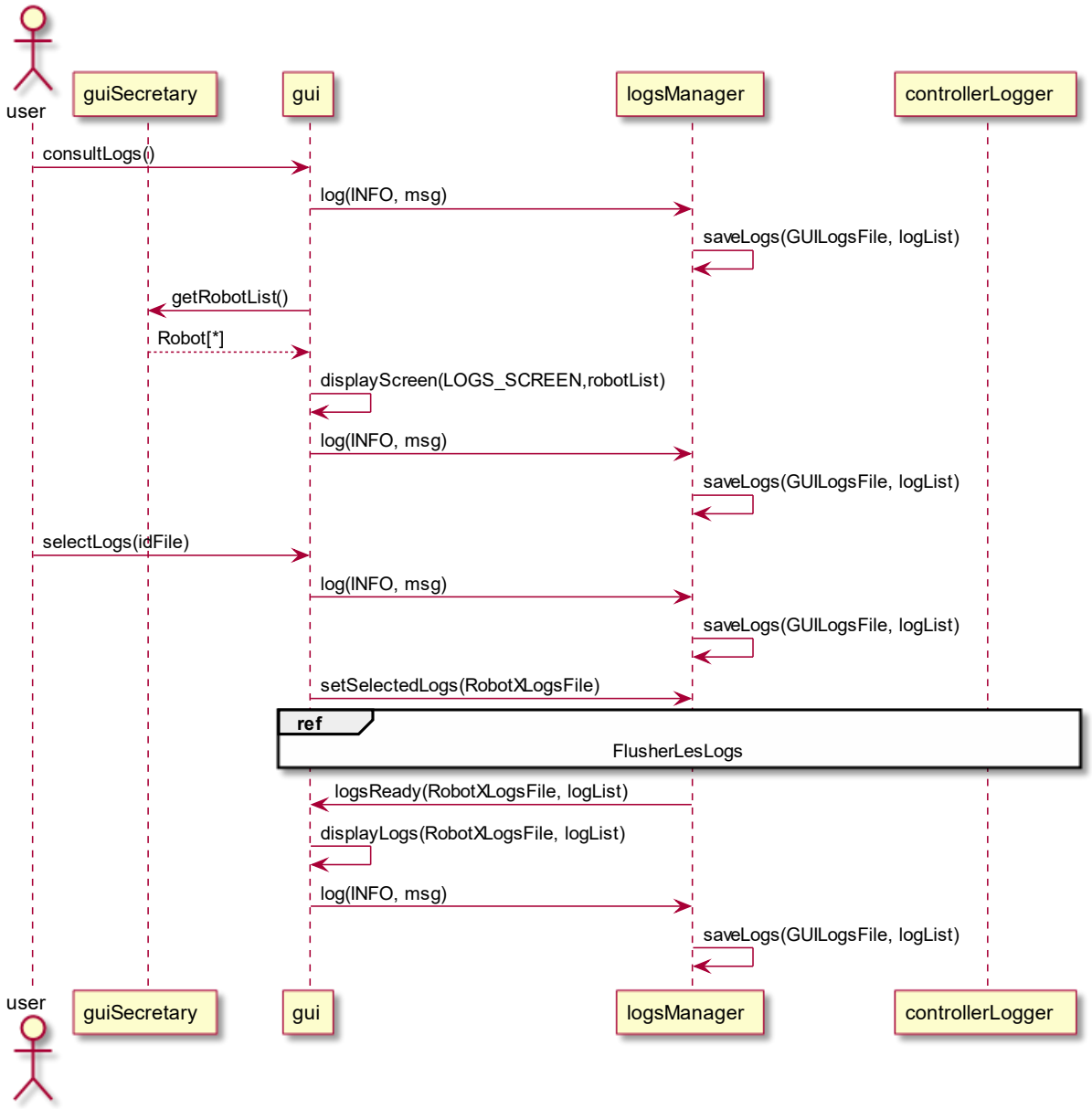


Figure 13 – Diagramme de séquence de : Consulter les logs (Robot)

2.2.13. Consulter les logs (IHM)

Ce diagramme représente le scénario du CU « Consulter les logs » du dossier de spécification [[ProSE]_A2_SPEC_2024]. Il représente le cas où les logs de l'IHM sont sélectionnés par l'utilisateur. Dans cette situation, SB_IHM récupère ses propres logs stockés dans son fichier local.

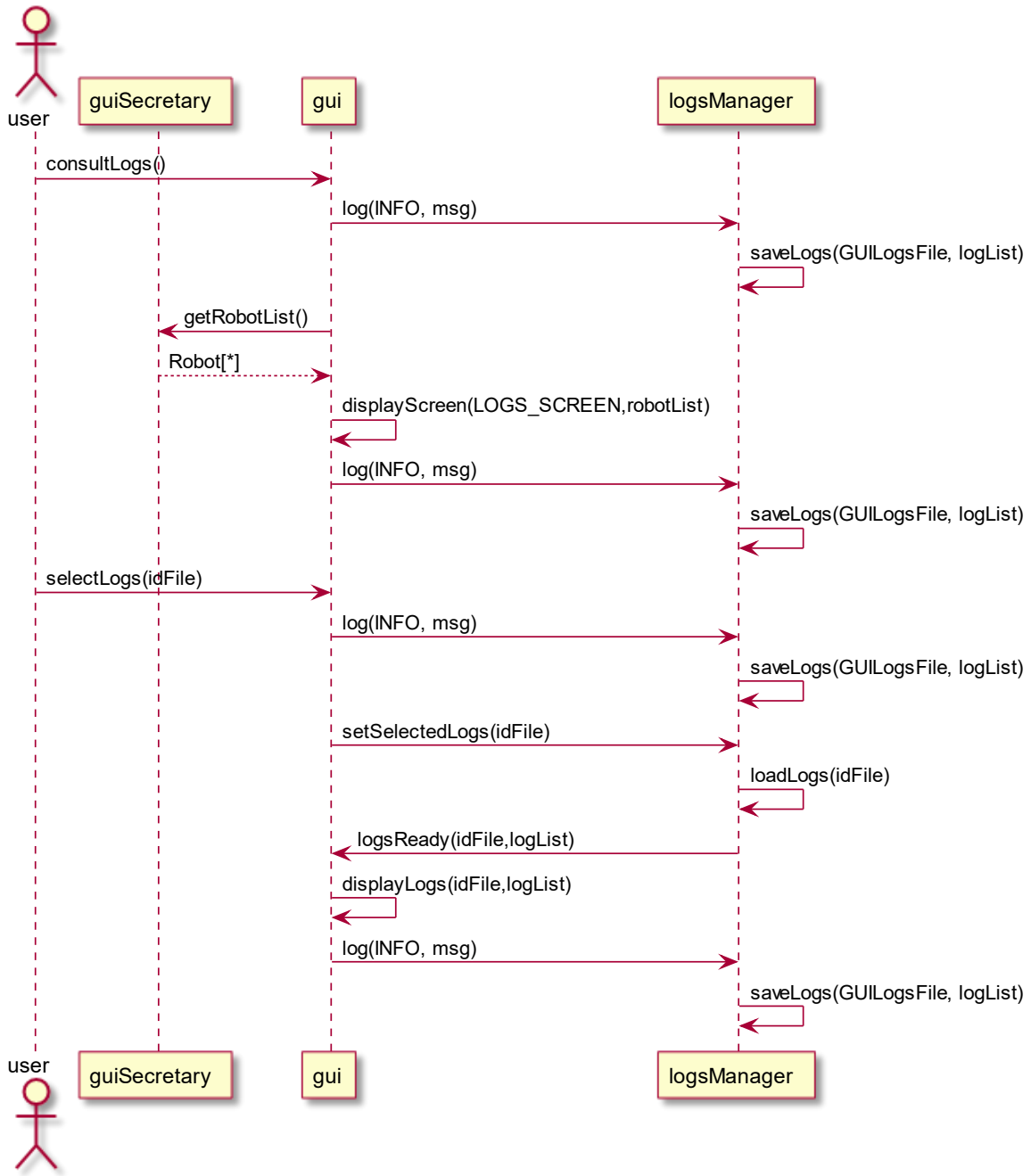


Figure 14 – Diagramme de séquence de : Consulter les logs SB_IHM

2.2.14. Flusher les logs

Ce diagramme représente le scénario du CU « Flusher les logs » du dossier de spécifications [[ProSE]_A2_SPEC_2024]. Lorsque l'utilisateur sélectionne un fichier de logs de robots ou que la mémoire est pleine, l'action de flusher se déclenche. LogsManager demande les logs de ControllerLogger. ControllerLogger suite à cette demande charge les logs présents dans son fichier et les donne à LogsManager. Lors de la réception de la totalité des logs, LogsManager acquitte la bonne réception de ceux-ci à ControllerLogger.

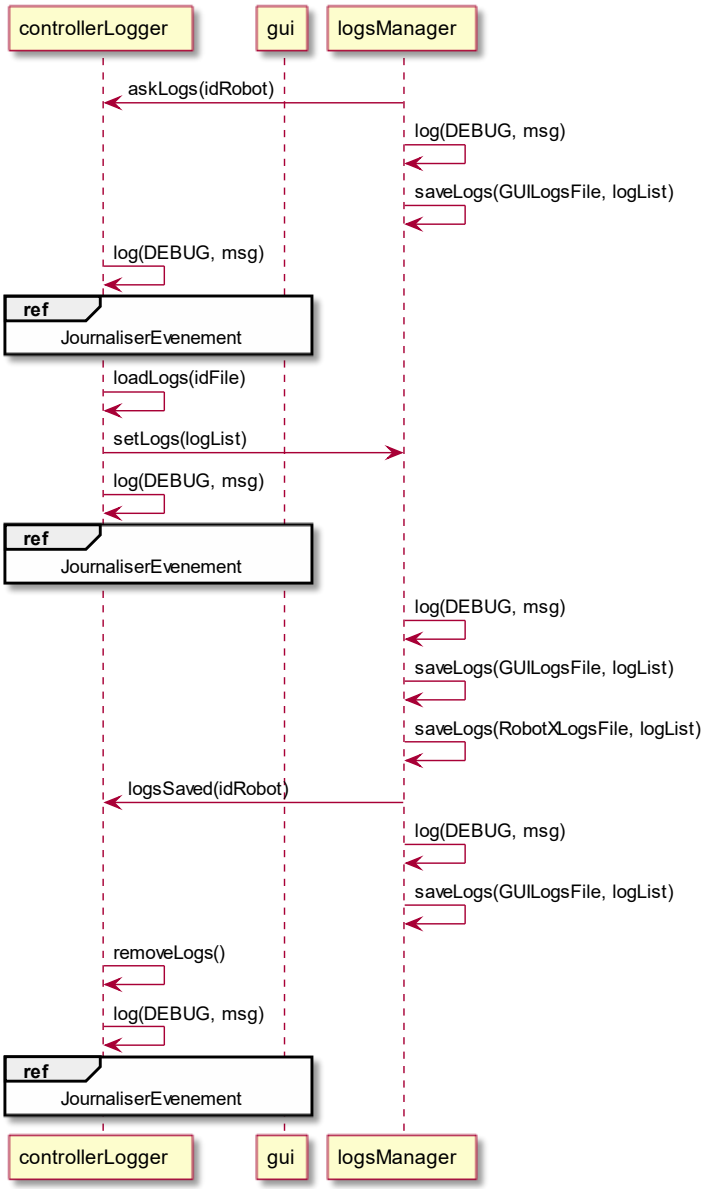
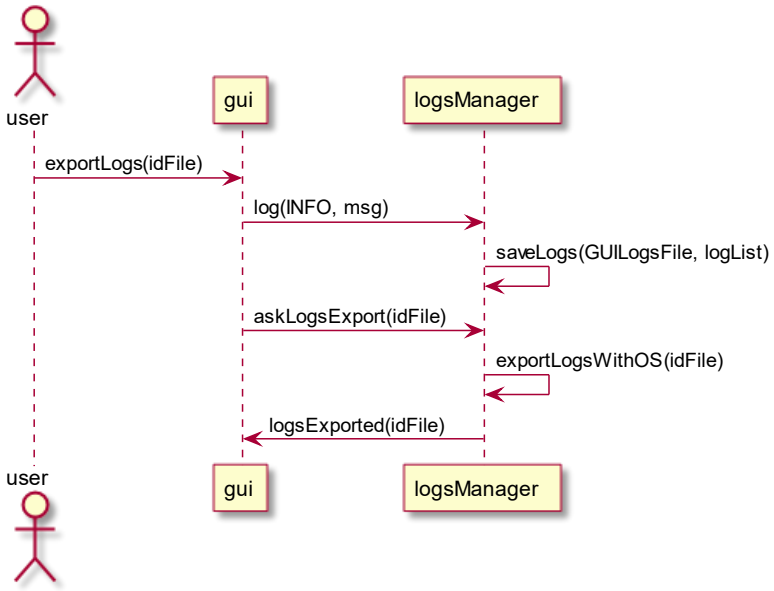


Figure 15 – Diagramme de séquence de : Flusher les logs

2.2.15. Exporter les logs

Ce diagramme représente le scénario du CU « Exporter les logs » du dossier de spécifications



[[ProSE]_A2_SPEC_2024]. Lorsque l'utilisateur sélectionne l'export des logs, LogsManager exporte les logs vers un fichier de la tablette et indique à l'utilisateur que les logs sont exportés.

Figure 16 – Diagramme de séquence de : Exporter les logs

2.2.16. Journaliser un évènement

Ce diagramme représente le scénario du CU « Journaliser un événement » du dossier de spécifications [[ProSE]_A2_SPEC_2024]. Lorsqu'une entrée de log est générée sur SB_C, ControllerLogger va vérifier la taille du fichier de logs avant d'enregistrer l'entrée de log. Si la taille est insuffisante alors ControllerLogger lève une alerte mémoire.

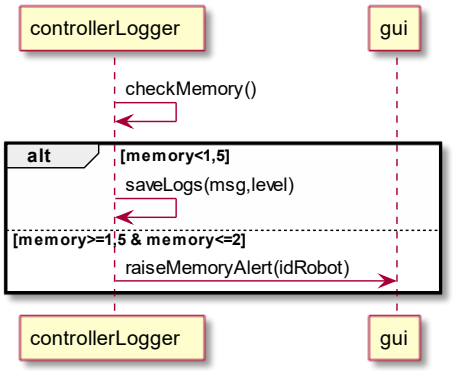


Figure 17 – Diagramme de séquence de : Journaliser un évènement

2.2.17. Quitter SB_IHM

Ce diagramme représente le scénario du CU « Quitter SB_IHM » du dossier de spécifications [[ProSE]_A2_SPEC_2024]. L'utilisateur demande à quitter SB_IHM. GUISecretary envoie une demande de deconnexion à chaque robot connecté. Chaque robot se déconnecte et avertit l'utilisateur via un changement de comportement des leds (ces dernières clignotent en vert).

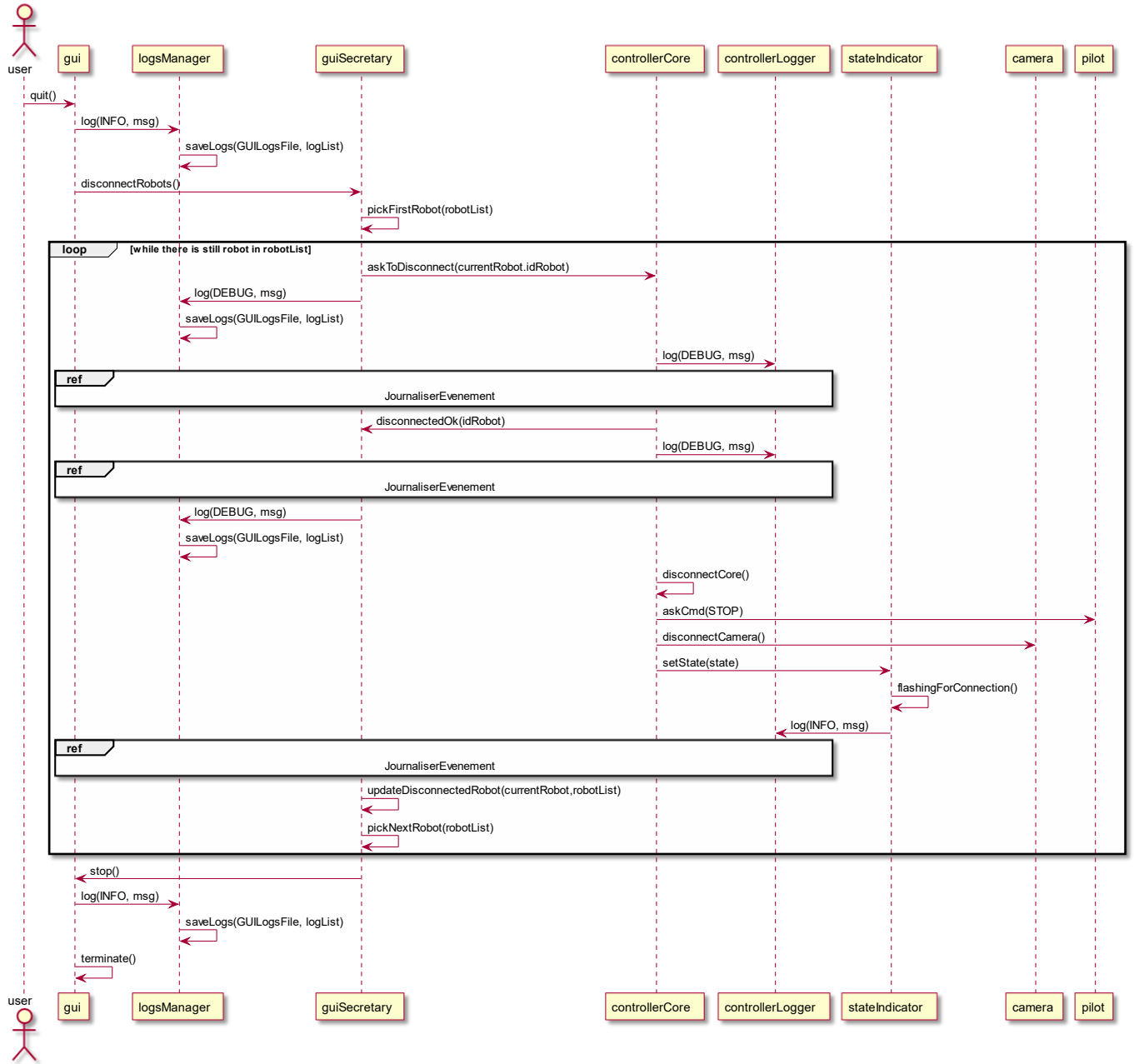


Figure 18 – Diagramme de séquence de : Quitter SB_IHM

2.3. Description des composants

2.3.1. Description des types manipulés entre composants

- **Robot :**

Modélise un robot possédant des paramètres suivants :

id : Entier compris entre 0 et 255 permettant d'identifier le robot
macAddress : Chaine de caractères ascii identifiant le robot physiquement parlant. Taille maximale de 50 caractères.
ipAddress : Chaîne de caractères au format XXX.XXX.XXX.XXX avec XXX des entiers indépendants compris entre 0 et 255. Correspond à l'adresse ip du robot.
connectionState : Valeur énumérée pouvant prendre les valeurs suivantes :
SELECTED : signifiant que le robot est sélectionné.
UNSELECTED : significant que le robot n'est pas sélectionné.
selectionState : Valeur énumérée pouvant prendre les valeurs suivantes :
CONNECTED : signifiant que le robot est connecté.
DISCONNECTED : signifiant que le robot n'est pas connecté.
failedPings : Valeur entière comprise entre 0 et 255.
operatingMode : Défini ci-dessous.

- **IdFile :**

Différents fichiers de logs manipulés par SB_IHM. Valeur énumérée pouvant prendre les valeurs suivantes:
GUILogsFile
Robot*X*LogsFile
Avec *X* l'id_robot du robot concerné.

- **Log :**

Chaîne de caractères ascii ayant pour longueur maximale 100 caractères.

- ***Radar* :**

Modélise la détection d'obstacle du radar et peut prendre les valeurs énumérées suivantes :
OBJECT_DETECTED
OBJECT_NOT_DETECTED

- **Image :**

Image capturée par la caméra d'un robot. Encodée en base64 sous forme de chaîne de caractères.

- ***OperatingMode* :**

Modélise le mode de fonctionnement des périphériques du robot. Cela concerne les modes d'activation des périphériques suivants : ModeCamera, ModeBuzzer, ModeRadar, ModeLed. Chaque mode d'activation de périphérique est associé à une valeur énumérée :
ENABLED : signifiant que le périphérique est en état de fonctionner.
DISABLED : signifiant que le périphérique n'est pas en état de fonctionner.

- ***State* :**

Modélise l'état du robot. Valeur énumérée qui peut prendre les valeurs suivantes :
EMERGENCY (les LED clignotent en rouge et le buzzer sonne, un obstacle a été détecté)
WAITING_FOR_CONNECTION (les LED clignotent en vert, le robot n'est pas connecté)
SELECTED (les LED sont bleues, le robot est sélectionné)
NOT_SELECTED (les LED sont vertes, le robot est connecté)

- ***LogLevel* :**

Modélise le niveau d'un log. Valeur énumérée qui peut prendre les valeurs suivantes :
DEBUG (message de debuggage)
INFO (message d'information)
WARNING (message d'avertissement)
ERROR (message d'error)
NONE (aucun message)

2.3.2. Diagramme de classes de SWARMBOTS

Ci-dessous, le diagramme de classe de SWARMBOTS, chaque classe y est représentée. Les méthodes et attributs associés sont décrits plus bas, dans la description des classes.

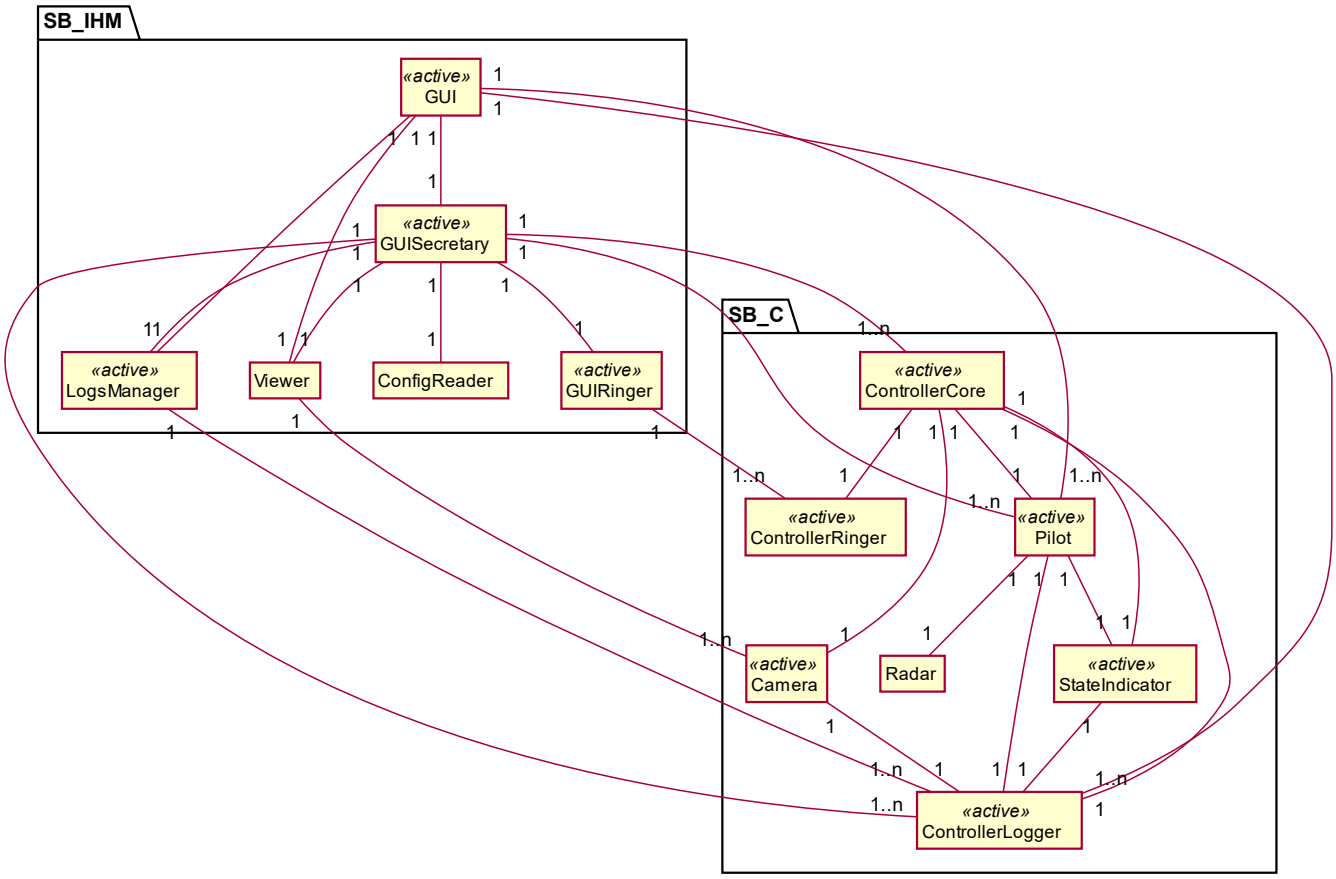


Figure 19 – Diagramme de classe de SWARMBOTS

2.3.3. Description des classes de SWARMBOTS

2.3.3.1. [IHM] La classe GUI

Figure 20 – Diagramme de classe de GUI

2.3.3.1.1. Philosophie de conception

La classe GUI représente la classe principale de SB_IHM. Elle s'occupe de l'interface utilisateur entre Utilisateur et SB_IHM et permet également d'afficher les vues.

2.3.3.1.2. Description structurelle

Attributs

- ***screenList* : Screen[] :** Liste des écrans avec leur identifiant idScreen.
- ***currentRobot* : Robot :** Désigne le robot actuellement pointé lors du parcours de robotList, cet attribut est notamment utilisé par les fonctions pickFirstRobot() et pickNextRobot().
- ***selectedRobot* : Robot :** Désigne le robot actuellement sélectionné par l'utilisateur.
- ***streamRefreshRate* : Integer :** Taux de rafraîchissement des flux radar et vidéo des robots connectés.
- ***selectedLogs* : IdFile :** Désigne l'identifiant du fichier de logs sélectionné par l'utilisateur.

- ***robotList* : Robot[] :** Liste contenant les informations sur les robots de l'application. Cela correspond à la Liste_Robots définie dans les spécifications.
- ***logList* : Log[] :** Désigne les logs affichés sur l'IHM.

Services offerts

- **(+)start() : void :** Méthode publique permettant de démarrer SB_IHM.
- **(+)commandRobot() : void :** Méthode publique permettant de commander le robot.
- **(+)selectRobot(*idRobot*) : void :** Méthode publique permettant de sélectionner le robot à commander, le robot est identifié par le paramètre *idRobot*.
- **(+)moveRobot(*selectedRobot*, *cmd*) : void :** Méthode publique permettant de déplacer le robot sélectionné, avec la commande identifié par le paramètre *cmd.*
- **(+)editRobotMode(*selectedRobot*, *operatingMode*) : void :** Méthode publique permettant à l'utilisateur de demander la modification du Mode_Fonctionnement, exprimé par *operatingMode*, du Robot_Selectionné, exprimé par *selectedRobot*.
- **(+)modeReady() : void :** Méthode publique permettant de notifier la fin de récupération des modes de fonctionnement.
- **(+)returnHome() : void :** Méthode publique permettant de retourner sur Ecran _Accueil.
- **(+)consultLogs() : void :** Méthode publique permettant de consulter les logs.
- **(+)selectLogs(*idFile*) : void :** Méthode publique permettant de sélectionner le log à consulter avec l'identifiant (*idFile*) du fichier dans lequel il sont enregistrés
- **(+)setLogs(*logList*) : void :** Méthode permettant d'enregistrer la liste des logs (logList) consultables par l'utilisateur sur l'IHM.
- **(+)exportLogs(*idFile*) : void :** Méthode publique permettant d'exporter les logs
- **(+)notifyError(*idError*) : void :** Méthode publique permettant d'avertir Utilisateur en cas d'erreur identifiée par le paramètre (*idError*).
- **(+)validate() : void :** Méthode publique permettant de fermer la fenêtre d'erreur.
- **(+)raiseMemoryAlert(*idRobot*) : void :** Méthode publique permettant d'alarmer Utilisateur lorsque la mémoire stockant les logs sur un des SB_C, identifié par son identifiant (*idRobot*), est pleine.
- **(+)flush() : void :** Méthode publique permettant de récupérer les logs sur SB_C.
- **(+)initReady() : void :** Méthode publique indiquant la fin d'initialisation des robots.
- **(+)logsReady(*idFile*, *logList*) : void :** Méthode publique permettant d'acquitter la bonne réception des logs par LogsManager.
- **(+)ackCmd(*idRobot*, *cmd*) : void :** Méthode publique permettant à Pilot d'acquitter la commande de déplacement précédemment envoyée.
- **(+)quit() : void :** Méthode publique permettant de quitter SB_IHM.
- **(+)stop() : void :** Méthode publique permettant d'avertir la classe de la déconnexion des robots .
- **(-)refreshCameraStream(*image*,*idRobot*) : void :** Méthode interne permettant d'actualiser un flux vidéo.
- **(-)refreshRadarStream(*radar*,*idRobot*) : void :** Méthode interne permettant d'actualiser un flux du radar.
- **(-)displayScreen(*idScreen*, *robotList*) : void :** Méthode interne permettant d'afficher l'écran dont l'identifiant correspond au paramètre idScreen avec la

liste des robots.

- **(-)displayLogs(*idFile*, *logList*) : void :** Méthode interne permettant d'afficher les logs sur Ecran_logs.
- **(-)updateCurrentScreen(*robotList*) : void :** Méthode interne permettant d'actualiser l'affichage de la liste de robots sur l'écran.
- **(-)displayPopUp(*idPopup*, *idRobot*) : void :** Méthode interne permettant d'afficher un écran PopUp sur SB_IHM.
- **(-)pickFirstRobot(*robotList*) : void :** Méthode interne permettant de choisir le premier robot de la liste des robots en paramètre. Change la valeur de currentRobot.
- **(-)pickNextRobot(*robotList*) : void :** Méthode interne permettant de choisir le prochain robot de la liste des robots en paramètre en fonction de currentRobot. Change la valeur de currentRobot.
- **(-)consultRobotMode(*robotList*, *idRobot*) : void :** Méthode interne permettant de consulter l'état des périphériques du robot entré en paramètre.
- **(-)authorizeRobotCommand() : void :** Méthode interne autorisant les commandes des robots.
- **(-)denyRobotCommand() : void :** Méthode interne refusant les commandes des robots.
- **(-)terminate() : void :** Méthode interne permettant de terminer le processus de l'application.

2.3.3.1.3. Description comportementale

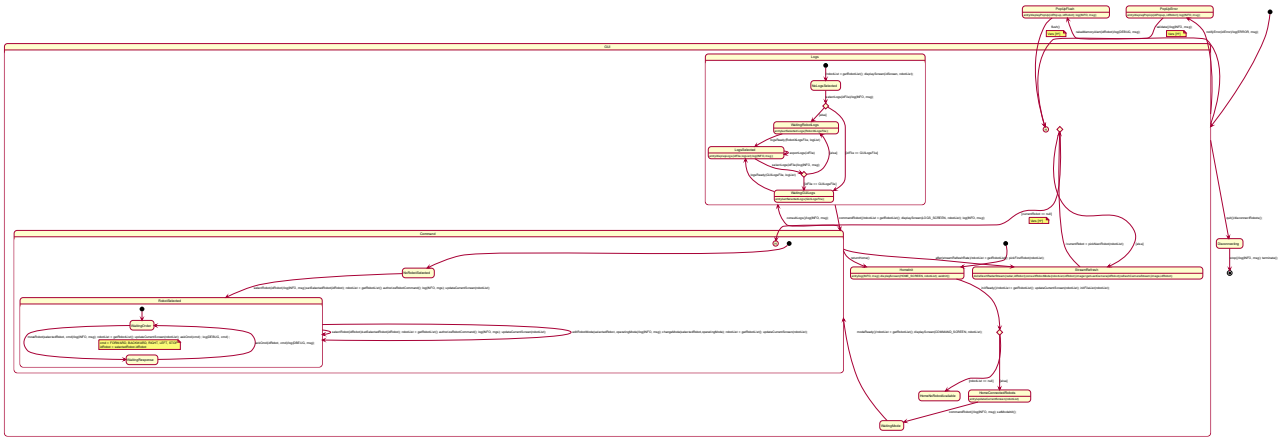


Figure 21 – Machine à états de GUI

2.3.3.2. [Object] La classe GUISecretary

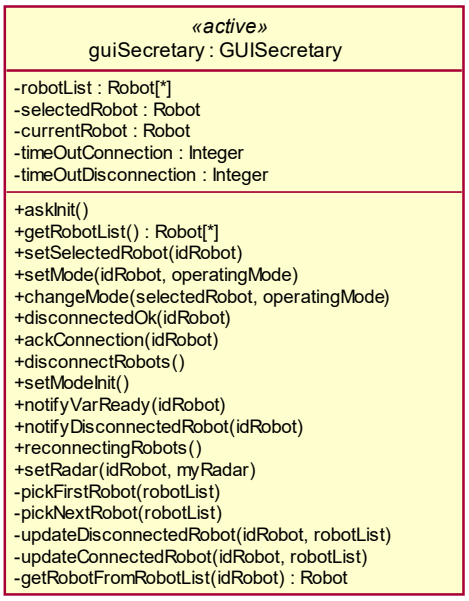


Figure 22 – Diagramme de la classe GUISecretary

2.3.3.2.1. Philosophie de conception

La classe GUISecretary permet d'intéragir avec les différents objets de SB_IHM ainsi que ControllerCore pour SB_C.

2.3.3.2.2. Description structurelle

Attributs

- ***robotList* : Robot[]** : liste des robots connectés.
- ***selectedRobot* : Robot** : Robot sélectionné.
- ***currentRobot* : Robot** : Robot auquel on cherche à se connecter ou se déconnecter.
- ***timeOutConnection* : Integer** : Temps pour lequel GUISecretary attendra la réponse d'un robot lors de sa connexion.
- ***timeOutDisconnection* : Integer** : Temps pour lequel GUISecretary attendra la réponse d'un robot lors de sa déconnexion.

Services offerts

- **(+)askInit() : void** : Méthode publique lançant l'initialisation de l'application pour interroger ConfigReader pour récupérer la liste de robots et les connecter.
- **(+)getRobotList() : Robot[]** : Méthode publique et passive renvoyant la robotList pour GUI.
- **(+)setSelectedRobot(*idRobot*) : void** : Méthode publique qui assigne quel robot est sélectionné par l'utilisateur. Si un robot a été précédemment sélectionné, cette méthode transmet l'ordre de désélection au robot concerné.
- **(+)setMode(*idRobot*, *operatingMode*) : void** : Méthode publique permettant de récupérer le Mode_Fonctionnement du robot.
- **(+)changeMode(*selectedRobot*, *operatingMode*) : void** : Méthode publique demandant à demander la modification d'un mode de fonctionnement identifié par le paramètre *operatingMode* d'un robot.

- **(+)disconnectedOk(*idRobot*) : void** : Méthode publique assurant la bonne déconnexion du robot choisi.
- **(+)ackConnection(*idRobot*) : void** : Méthode publique assurant la bonne connexion du robot choisi.
- **(+)disconnectRobots() : void** : Méthode publique demandant à déconnecter tous les robots.
- **(+)setModeInit() : void** : Méthode publique permettant d'enclencher les demandes de récupération les Mode_Fonctionnement des robots.
- **(+)notifyVarReady(idRobot) : void** : Méthode publique permettant d'indiquer que le failedPongsVar, du robot identifié par l'*idRobot*, a été initialisé.
- **(+)notifyDisconnectedRobot(idRobot) : void :** Méthode publique qui indiquant qu'un des robots est déconnecté pour le déconnecter.
- **(+)reconnectingRobots() : void** : Méthode publique pour reconnecter les robots déconnectés en revenant à l'étape d'attente de connexions.
- **(+)setRadar(idRobot, myRadar) : void** : Méthode publique pour transmettre un changement d'état de la détection d'obstacle du radar du robot identifié par *idRobot*.
- **(-)pickFirstRobot(*robotList*) : void** : Méthode interne récupérant le premier robot de *robotList*.
- **(-)pickNextRobot(*robotList*) : void** : Méthode interne récupérant le prochain robot dans le parcours de *robotList*.
- **(-)updateDisconnectedRobot(*idRobot, robotList*) : void** : Méthode publique mettant à jour robotList pour le currentRobot.
- **(-)updateConnectedRobot(*idRobot, robotList*) : void** : Méthode publique mettant à jour robotList pour le currentRobot.
- **(-)updateRobotList(*robotList*) : void** : Méthode publique mettant à jour robotList.
- **(-)getRobotFromRobotList(*idRobot*) : Robot** : Méthode interne pour renvoyer un robot dans la liste de robots à partir d'un *idRobot*.

2.3.3.2.3. Description comportementale

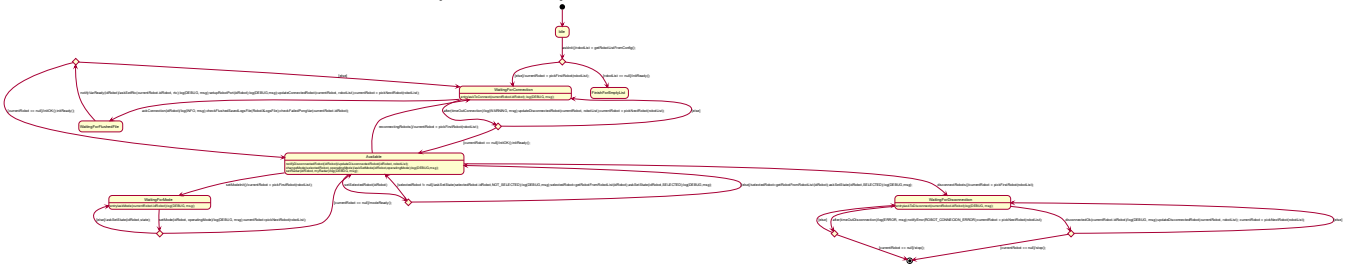


Figure 23 – Machine à états de GUISecretary

2.3.3.3. [Object] La classe Viewer

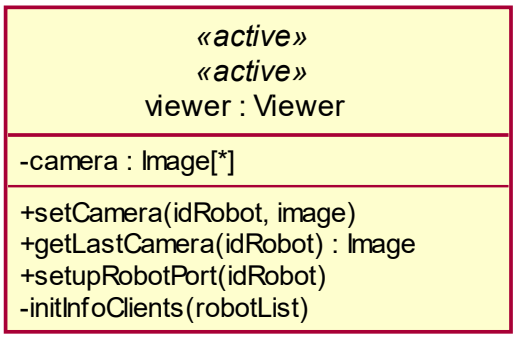


Figure 24 – Diagramme de classe Viewer

2.3.3.3.1. Philosophie de conception

La classe Viewer stocke la liste des données radar et d'images, permettant à GUI de les récupérer afin de les afficher.

2.3.3.3.2. Description structurelle

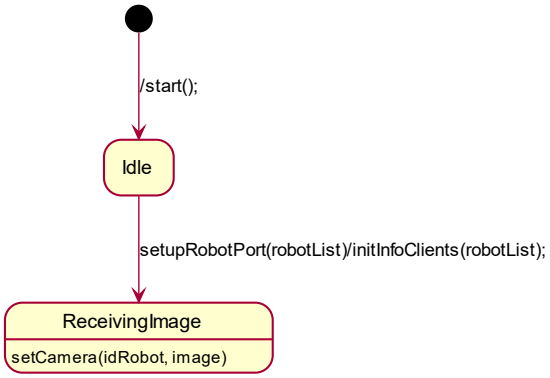
Attributs

- ***camera* : Image[]** : liste des données d'images.

Services offerts

- **(+)setCamera(*idRobot*, *image*) : void** : Méthode publique permettant la liste des données d'images
- **(+)getLastCamera(*idRobot*) : Image** : Méthode publique permettant d'obtenir la dernière image capturée par la caméra.
- **(+)setupRobotPort(*idRobot*) : void :** Méthode publique qui permet d'envoyer les informations de transmission du flux caméra à SB_C du robot identifié par *idRobot*.
- **(-)initInfoClients(robotList) : void :** Méthode publique signalant à Viewer d'envoyer les informations de Tablette à chaque robot. Informations qui sont l'adresse IP et le port dédié au robot.

2.3.3.3.3. Description comportementale



2.3.3.4. [Object] La classe LogsManager

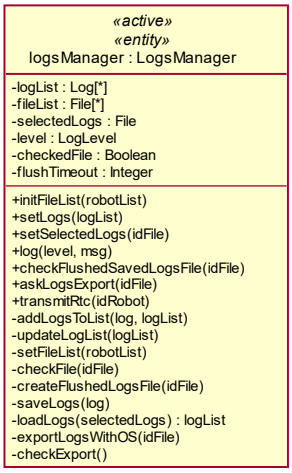


Figure 25 – Diagramme de la classe LogsManager

2.3.3.4.1. Philosophie de conception

La classe LogsManager permet de récupérer et stocker les logs de l'appareil souhaité, SB_IHM ou l'un des SB_C, sur demande de GUI ou de GUISecretary. Elle est également capable de créer des fichiers associés aux appareils (SB_IHM ou SB_C) correspondants pour y stocker les logs.

2.3.3.4.2. Description structurelle

Attributs

- ***logList* : Log[]** : Liste des logs sélectionnés avec toutes leurs informations.
- ***fileList* : IdFile[]** : Liste des fichiers de logs associés individuellement à un appareil (SB_IHM ou SB_C).
- ***selectedLogs* : IdFile** : Identifiant du fichier à lire sur demande de GUI.
- ***level* : LogLevel** : Niveau de criticité des logs minimum à afficher.
- ***checkedFile* : Bolean** : Etat renseignant sur le check effectué ou non sur un fichier.
- ***FLushTimeOut* : Integer** : Temps d'attente maximale pendant l'action de flush les logs.

Services offerts

- **(+)initFileList(*robotList*) : void** : Méthode publique s'assurant de la présence et de la disponibilité des fichiers de logs.
- **(+)setLogs(*logList*) : void** : Méthode publique récupérant les logs de Controller-Logger.
- **(+)setSelectedLogs(*idFile*) : void** : Méthode publique retournant à GUI logList.
- **(+)log(*level*, *msg*) : void** : Méthode publique enregistrant les logs de SB_IHM.
- **(+)checkFlushedSavedLogsFile(*idFile*) : void** : Méthode publique qui vérifie que le fichier de logs est créé de *idFile*. S'il ne l'est pas, il appellera une méthode en interne pour le créer.
- **(+)askLogsExport(*idFile*) : void** : Méthode publique permettant de demander l'export des logs.
- **(+)transmitRtc() : void :** Méthode publique demandant d'envoyer l'heure de SB_IHM vers SB_C par la méthode askSetRtc(idRobot, rtc).
- **(-)addLogsToList(*log*, *logList*) : void** : Méthode publique ajoutant le dernier log à la loglist associée.
- **(-)setFileList(*robotList*) : void** : Méthode interne modifiant la liste des fichiers à avoir.
- **(-)checkFile(*idFile*) : void** : Méthode interne permettant de vérifier si un fichier existe en mémoire grâce à l'identifiant du fichier voulu. Elle renvoie le résultat de cette vérification.
- **(-)createFlushedLogsFile(*idFile*) : void** : Méthode interne créant un fichier de logs du *idFile* donné.
- **(-)saveLogs(*log*) : void** : Méthode interne sauvegardant le fichier des logs de l'appareil sélectionné.
- **(-)loadLogs(*selectedLogs*) : logList** : Méthode interne lisant le fichier des logs de l'appareil sélectionné.
- **(-)exportLogsWithOS(*idFile*) : void** : Méthode interne permettant d'exporter les logs à l'aide de l'OS Android.
- **(-)checkExport() : void** : Méthode interne permettant de vérifier si une erreur a été détectée pendant l'export.

2.3.3.4.3. Description comportementale

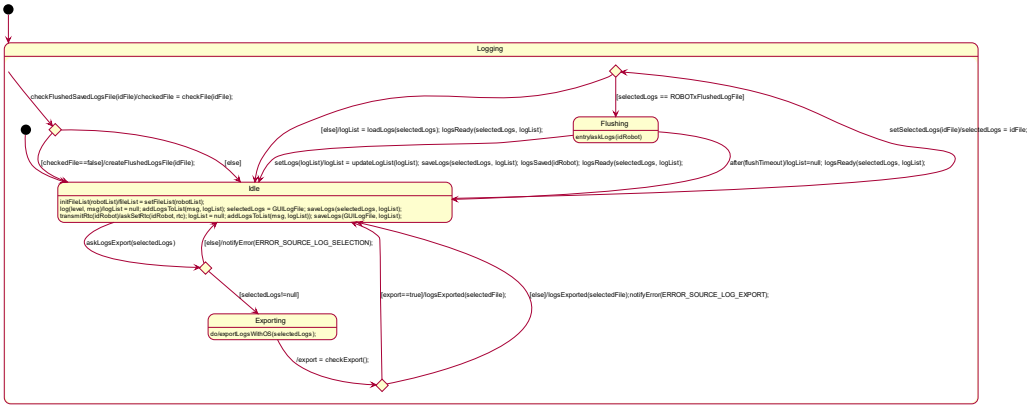


Figure 26 – Machine à états de LogsManager

2.3.3.5. [Object] La classe Robot

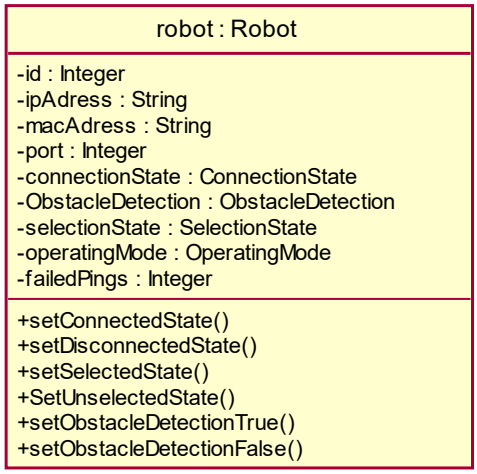


Figure 26 – Diagramme de la classe Robot

2.3.3.5.1. Philosophie de conception

La classe Robot est utilisée par tous les objets pour transmettre les informations relatives aux robots. Elle fait partie des types manipulés entre les composants.

2.3.3.5.2. Description structurelle

Attributs

- ***ip* : Integer** : Identifiant du robot.
- ***ipAdress* : String** : Adresse ip du robot.
- ***macAdress* : String** : Adresse mac du robot.
- ***port* : Integer** : Port du robot.
- ***connectionState* : ConnectionState** : Etat de connexion du robot.
- ***selectionState* : SelectionState** : Etat de sélection du robot.
- ***operatingMode* : OperatingMode** : Mode de fonctionnement du robot.
- ***failedPings* : Integer** : Pings perdus du robot.

Services offerts

- **(+)setConnectedState() : void** : Méthode publique permettant de mettre le robot désigné en état connecté.
- **(+)setDisconnectedState() : void** : Méthode publique permettant de mettre le robot désigné en état déconnecté.
- **(+)setSelectedState() : void** : Méthode publique permettant de mettre le robot désigné en état sélectionné.
- **(+)setUnselectedState() : void** : Méthode publique permettant de mettre le robot désigné en état sélectionné.
- **(+)setObstacleDetectionTrue() : void** : Méthode publique permettant de mettre la détection de l'obstacle du robot désigné en état détecté.
- **(+)setObstacleDetectionFalse() : void** : Méthode publique permettant de mettre la détection de l'obstacle du robot désigné en état non-détecté.

2.3.3.6. [Object] La classe ConfigReader

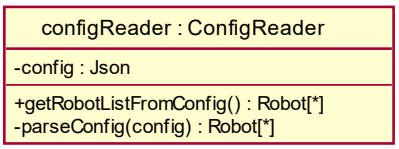


Figure 27 – Diagramme de la classe ConfigReader

2.3.3.6.1. Philosophie de conception

La classe configReader s'occupe de la création de la liste des à partir d'un fichier de configuration des robots.

2.3.3.6.2. Description structurelle

Attributs

- ***config* : Json** : Renvoie la configuration enregistrée.

Services offerts

- **(+)getRobotsListFromConfig() : Robot[]** : Méthode publique permettant de récupérer le fichier de configuration et d'afficher une liste de robots.
- **(-)parseConfig(config) : Robot[]** : Méthode privée permettant de préparer une liste de robots à partir des configurations prises du Fichier_Config.

2.3.3.7. [Object] La classe ControllerCore

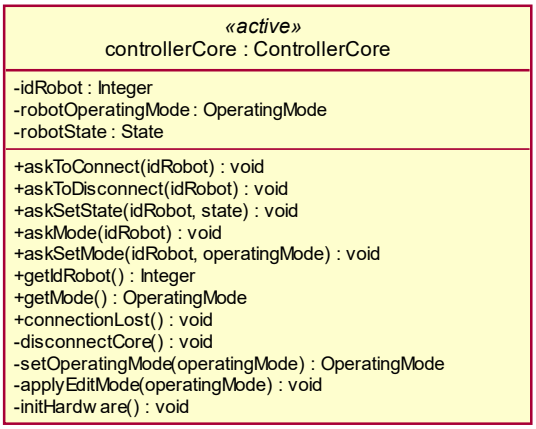


Figure 28 – Diagramme de classe représentant ControllerCore

2.3.3.7.1. Philosophie de conception

La classe ControllerCore à un rôle « central » au sein de SB_C, elle se charge d'activer/désactiver les périphériques par la méthode askSetMode(*state*) et d'indiquer à la classe StateIndicator l'état dans lequel se trouve le robot par la méthode askSetState(*state*), ces fonctions sont appelées par la classe GUISecretary. C'est également cette classe qui permet l'établissement de la connexion entre SB_C et SB_IHM.

2.3.3.7.2. Description structurelle

Attributs

- ***idRobot* : Integer :** Identifiant du robot.

- ***robotOperatingMode* : OperatingMode** : Variable privée permettant de stocker l'état courant de chaque périphérique. (Valeurs Booléennes).
- ***robotState* : State** : Variable privée permettant de stocker l'état du robot.

Services offerts

- **(+)askToConnect(*idRobot*) : void** : Méthode publique utilisée par GUISecretary pour établir une connexion entre SB_IHM et SB_C identifié par ***idRobot***.
- **(+)askToDisconnect(*idRobot*) : void** : Méthode publique utilisée par GUISecretary pour arrêter la connexion entre SB_IHM et SB_C identifié par ***idRobot***.
- **(+)askSetState(*idRobot, state*) : void** : Méthode publique permettant d'indiquer une demande de changement d'état, état représenté par le paramètre *state*.
- **(+)askMode(*idRobot*) : void** : Méthode publique permettant à GUISecretary d'obtenir le mode de fonctionnement (***operatingMode***) du robot identifié par *idRobot*.
- **(+)askSetMode(*idRobot, operatingMode*) : void** : Méthode publique permettant d'indiquer une demande de changement de mode de fonctionnement, ce nouveau mode de fonctionnement est indiqué par le paramètre ***operatingMode***.
- **(+)getIdRobot() : Integer** : Méthode publique passive permettant de renvoyer l'identifiant du robot.
- **(+)getMode() : OperatingMode** : Méthode publique passive renvoyant les modes de fonctionnements courant des périphériques enregistrés en mémoire.
- **(+)connectionLost() : void** : Méthode publique permettant de demander à se déconnecter à cause d'une perte de connexion à SB_IHM.
- **(-)disconnectCore() : void** : Méthode interne permettant de déconnecter SB_C en cas de perte de connexion avec SB_IHM.
- **(-)setOperatingMode(*operatingMode*) : OperatingMode** : Méthode interne permettant de modifier l'attribut ***operatingMode***.
- **(-)applyEditMode(*operatingMode*) : void** : Méthode interne permettant de changer les modes de fonctionnement des périphériques.
- **(-)initHardware() : void :** Méthode interne permettant d'initialiser des périphériques externes. (servoMotor en conception détaillée).

2.3.3.7.3. Description comportementale

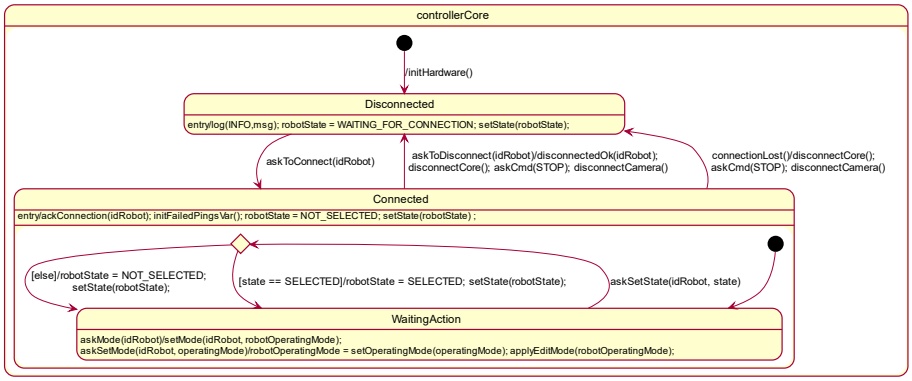


Figure 29 – Machine à états de ControllerCore

2.3.3.8. [Object] La classe StateIndicator

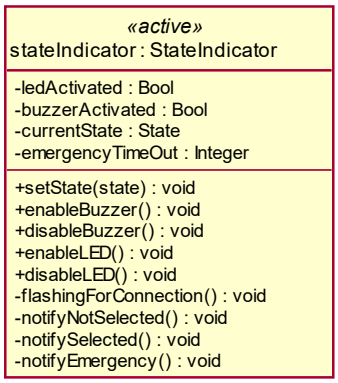


Figure 30 – Diagramme de la classe StateIndicator

2.3.3.8.1. Philosophie de conception

La classe StateIndicator permet de connaître l'état des périphériques de SB_C et d'en avertir Utilisateur.

2.3.3.8.2. Description structurelle

Attributs

- ***ledActivated* : Boolean** : Renseigne sur l'état de la led.
- ***buzzerActivated* : Boolean** : Renseigne sur l'état du buzzer.
- ***currentState* : State** : Etat demandé par ControllerCore, ce qui entraînera un changement de couleur des LEDs.
- **alarmEmergencyTime : Integer** : Temps d'activement du buzzer à la détection d'un obstacle.

Services offerts

- **(+)setState(*state*) : void** : Méthode publique pour recevoir l'ordre de changement d'état du robot, état représenté par le paramètre state. Cette fonction transmet l'ordre concerné en appelant une des fonctions internes.
- **(+)enableBuzzer() : void** : Méthode publique activant le mode de fonctionnement buzzer.
- **(+)disableBuzzer() : void** : Méthode publique désactivant le mode de fonctionnement buzzer.
- **(+)enableLED() : void** : Méthode publique activant le mode de fonctionnement de la led.
- **(+)disableLED() : void** : Méthode publique désactivant le mode de fonctionnement de la led.
- **(-)flashingForConnection() : void** : Méthode interne renseignant sur state avant la connexion à SB_IHM.
- **(-)notifyNotSelected() : void** : Méthode interne mettant state en mode non sélectionné.
- **(-)notifySelected() : void** : Méthode interne mettant state en mode sélectionné.

- **(-)notifyEmergency() : void** : Méthode interne mettant la led en couleur d'urgence si state le permet.

2.3.3.8.3. Description comportementale

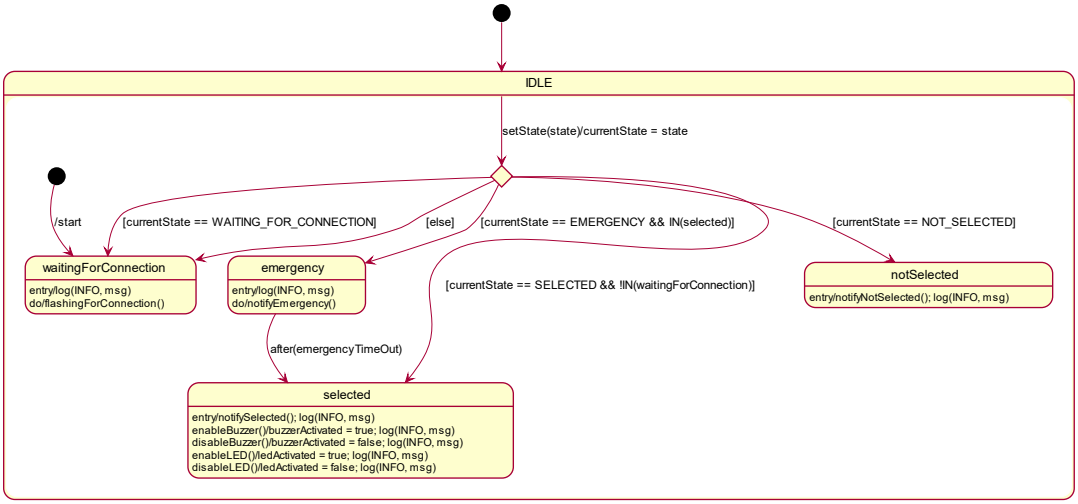


Figure 31 – Machine à états de StateIndicator

2.3.3.9. [Object] La classe Pilot

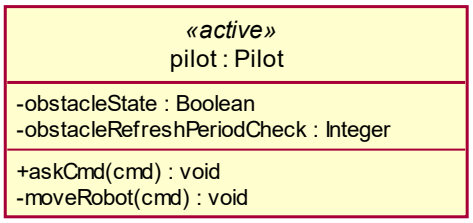


Figure 33 – Diagramme de la classe Pilot

2.3.3.9.1. Philosophie de conception

La classe Pilot permet de piloter le robot à la demande de GUI et de notifier des obstacles détectés par Radar.

2.3.3.9.2. Description structurelle

Attributs

- ***obstacleRefreshPeriodCheck* : Integer** Renseigne sur la présence ou non d'un obstacle devant SB_C.
- ***obstacleState* : Boolean** : Etat actuel de détection du radar.

Services offerts

- **(+)askCmd(*cmd*) : void** : Méthode publique récupérant l'instruction à exécutée.
- **(-)moveRobot(*cmd*) : void** : Méthode interne exécutant l'instruction demandée.

2.3.3.9.3. Description comportementale

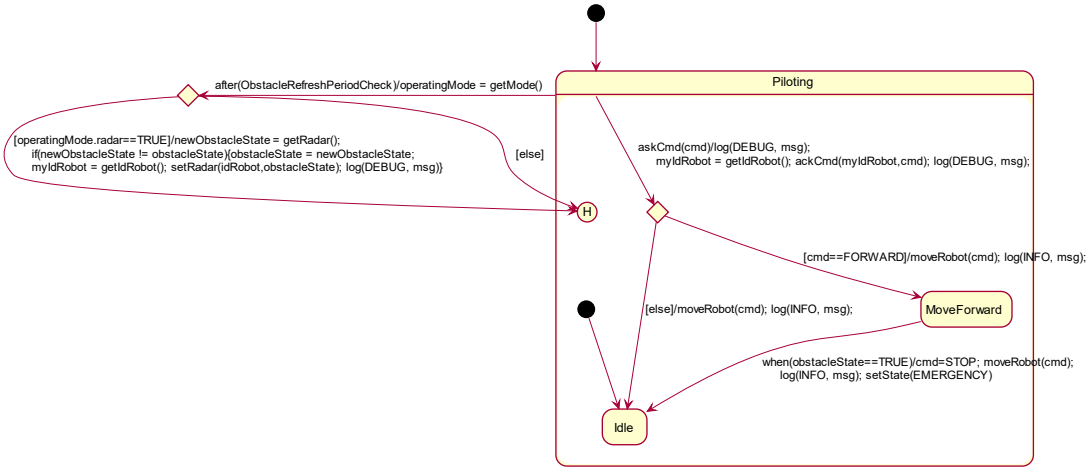


Figure 34 – Machine à états de Pilot

2.3.3.10. [Object] La classe ControllerLogger

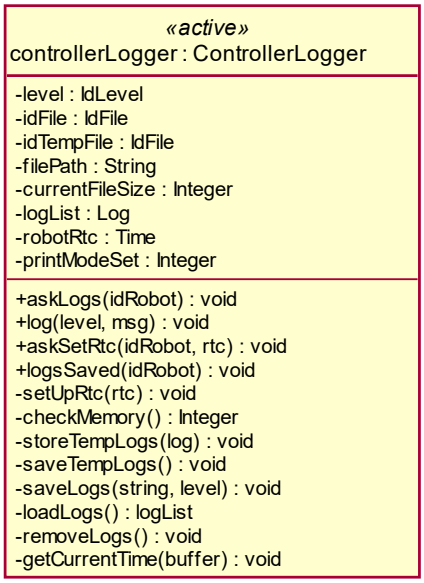


Figure 35 – Diagramme de la classe ControllerLogger

2.3.3.10.1. Philosophie de conception

La classe ControllerLogger est utilisée pour enregistrer les logs venant des différents modules de SB_C. SB_IHM à la possibilité de demander les logs de ControllerLogger afin de les récupérer.

2.3.3.10.2. Description structurelle

Attributs

- ***level* : LogLevel** :Niveau de criticité des logs minimum à afficher.
- ***idFile* : IdFile** : Identifiant du fichier où écrire les logs.
- ***filePath* : String** : Chemin d'accès du fichier de logs.
- ***currentFileSize* : Integer** : Taille actuelle du fichier de logs.
- ***logList* : Log[]** : Liste des logs avec toutes leurs informations.
- ***robotRtc :* Time *:*** Temps actuel synchronisé avec SB_IHM et permet l'horodatage des logs.
- ***printModeSet* : Integer** : Mode d'affichage du logger, en console ou dans un fichier ou les deux.

Services offerts

- **(+)askLogs(*idRobot*) : void** : Méthode publique permettant à LogsManager de récupérer les logs du SB_C identifié par le paramètre *idRobot*.
- **(+)log(*level*, *msg*) : void :** Méthode publique enregistrant les logs de SB_C avec un horodatage du log, un niveau de criticité du log avec le paramètre *level* et un message associé avec le paramètre *msg*.
- **(+)askSetRtc(*idRobot, rtc*) : void :** Méthode publique permettant de synchroniser les heures de SB_IHM et SB_C en l'enregistrant dans la variable robotRtc.
- **(+)logsSaved(*idRobot*) : void** : Méthode publique permettant à LogsManager de valider qu'il a reçu les logs envoyés précédemment afin de pouvoir les supprimer.
- **(-)setUpRtc(*rtc*) : void** : Méthode interne permettant de synchroniser l'heure de SB_C avec celle de SB_IHM. Heure exprimée par le paramètre *rtc*.

- **(-)checkMemory() : Integer :** Méthode interne observant l'espace mémoire restant sur l'espace de stockage de SB_C.
- **(-)storeTempLogs(*log*) : void** : Méthode interne permettant de stocker dans un buffer les logs en attendant que la rtc soit envoyé par SB_IHM.
- **(-)saveTempLogs(*log*) : void** : Méthode interne enregistrant dans le fichier les logs temporairement stockés en attendant la rtc envoyé par SB_IHM.
- **(-)saveLogs(*string, log*) : void** : Méthode interne concaténant avec un indicateur de temps le paramètre ***string*** et ***log*** pour ensuite enregistrer le log dans un fichier ou l'afficher dans le terminal ou les deux.
- **(-)loadLogs() : LogList :** Méthode interne lisant le fichier de logs de SB_C.
- **(-)removeLogs() : void** : Méthode permettant de supprimer les logs dans le fichier de logs de SB_C.
- **(-)getCurrentTime(*buffer*) : void** : Récupère le temps courant du robot.

2.3.3.10.3. Description comportementale

Figure 36 – Machine à état de ControllerLogger

2.3.3.11. [Object] La classe ControllerRinger

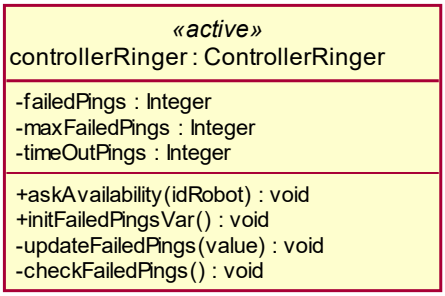


Figure 37 – Diagramme de classe représentant ControllerRinger

2.3.3.11.1. Philosophie de conception

La classe ControllerRinger permet la gestion du « ping-pong » entre SB_IHM et SB_C en répondant au « ping » de la part de GUIRinger. Ainsi qu'en vérifiant si le nombre de « ping » non-reçus atteint une certaine limite afin de demander une déconnexion à ControllerCore. Une limite est atteinte lorsque SB_IHM n'est plus opérationnelle et n'envoie plus de « ping ».

2.3.3.11.2. Description structurelle

Attributs

- ***failedPings* : Integer :** Total de ping non reçus. (Correspond à la var_connexion)
- ***maxFailedPings* : Integer :** Limite de ping non reçus.
- ***timeOutPings* : Integer :** Temps d'attente de réception d'un « ping » avant d'estimer que celui-ci ne sera pas reçu.

Services offerts

- **(+)askAvailability(*idRobot*) : void :** Méthode publique utilisée par GUISecretary pour faire le « ping » vers SB_C identifié par ***idRobot***.
- **(+)initFailedPingsVar() : void :** Méthode publique utilisée par ControllerCore lors de l'initialisation afin de s'assurer que l'attribut ***failedPings*** est bien initialisé à 0.
- **(-)updateFailedPings(*value*) : void** : Méthode privée permettant d'incrémenter la valeur de l'attribut ***failedPings***.
- **(-)checkFailedPings() : void :** Méthode interne permettant de comparer ***failedPings*** avec ***MaxFailedPings***.

2.3.3.11.3. Description comportementale

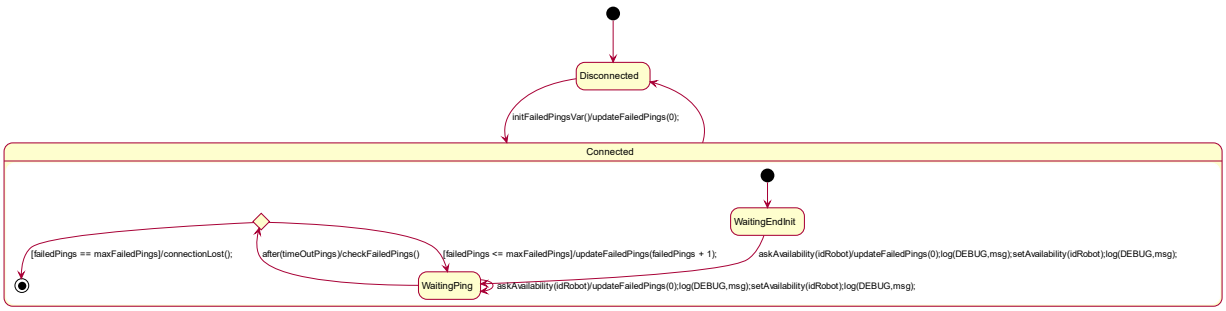


Figure 38 – Machine à états de la classe ControllerRinger

2.3.3.12. [Object] La classe Radar

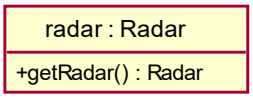


Figure 39 – Diagramme de classe représentant Radar

2.3.3.12.1. Philosophie de conception

La classe Radar se charge d'activer et désactiver le radar à la demande de ControllerCore. Radar viendra envoyer ses données à Viewer à intervalles réguliers. La classe viendra également vérifier les données du radar pour déterminer ou non la présence d'un obstacle à la demande de Pilot.

2.3.3.12.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)getRadar() : Radar** : Récupération de l'état du périphérique radar.

2.3.3.13. [Object] La classe Camera

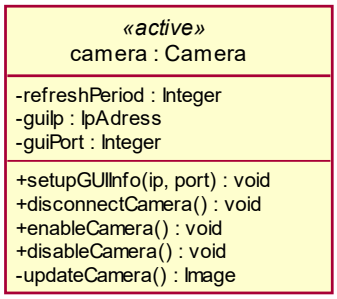


Figure 40 – Diagramme de classe représentant Camera

2.3.3.13.1. Philosophie de conception

La classe Camera se charge d'activer et désactiver le périphérique Camera à la demande de ControllerCore. Ainsi que d'envoyer ses données à Viewer à intervalles réguliers

2.3.3.13.2. Description structurelle

Attributs

- ***refreshPeriod* : Integer** : Période permettant d'aller chercher les images.
- ***guiIp* : ipAddress** : Ip de la tablette pour diffuser le flux UDP.
- ***guiPort* : Integer** : Port de la tablette pour diffuser le flux UDP.

Services offerts

- **(+)setupGUIInfo(*ip*, *port*) : void** : Méthode publique permettant d'envoyer l'ip et le port de la tablette, informations utiles pour commencer la diffusion en UDP.
- **(+)disconnectCamera() : void** : Méthode publique utilisée par ControllerCore pour déconnecter la caméra.
- **(+)enableCamera() : void** : Méthode publique utilisée par ControllerCore pour activer le périphérique camera.
- **(+)disableCamera() : void** : Méthode publique utilisée par ControllerCore pour désactiver le périphérique camera.
- **(-)updateCamera() : Image** : Méthode privée permettant à la classe de récupérer les données du périphérique camera.

2.3.3.13.3. Description comportementale

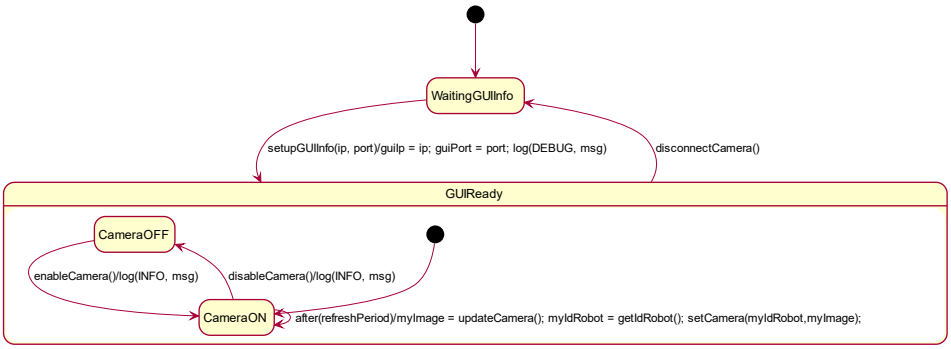


Figure 41 – Machine à états de Camera

2.3.3.13. [Object] La classe GUIRinger

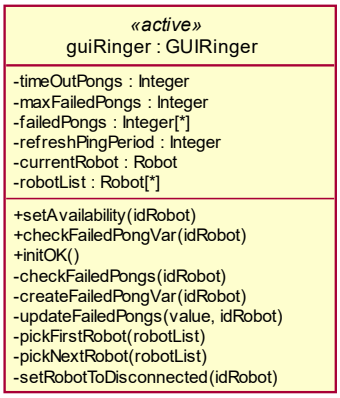


Figure 42 – Diagramme de classe représentant GUIRinger

2.3.3.13.1. Philosophie de conception

La classe GUIRinger permet la gestion du « ping-pong » entre SB_IHM et SB_C en envoyant un ping à chaque SB_C. Elle permet aussi de créer une var_connexion à chaque robot connectés de robotList. La classe vérifie si le nombre de « pong » non-reçus atteint une certaine limite afin de détecter une déconnexion d'un des robots. Une limite est atteinte lorsqu'un SB_C n'est plus connecté et ne répond plus de pong.

2.3.3.13.2. Description structurelle

Attributs

- ***timeOutPongs*** : Temps d'attente de réception d'un « pong » avant d'estimer que celui-ci ne sera pas reçu.
- ***maxFailedPongs*** : Constante permettant de définir la limite de « pong » « non-reçus » avant de considérer une déconnexion.
- ***refreshPingPeriod*** : Temps d'attente entre l'envoie des « ping ».
- ***currentRobot*** : **Robot** : Désigne le robot, dont sa connexion est en train d'être vérifiée. Variable contenant les infos présentes dans robotList mais uniquement du robot en question.
- ***robotList*** **: Robot[]** : Liste globale des robots avec toutes les informations de ceux-ci.

Services offerts

- **(+)setAvailibility(*idRobot*) : void** : Méthode publique permettant à la classe ControllerRinger de faire son « pong » en indiquant son identifiant par le paramètre *idRobot*.
- **(+)checkFailedPongVar(*idRobot*) : void** : Méthode publique permettant à la classe GUISecretary de venir créer une var_connexion pour le robot identifié par *idRobot*.
- **(+)initOK() : void** : Méthode publique permettant à GUISecretary d'indiquer à GUIRinger qu'il a fini son initialisation et qu'il est possible de commencer à faire le « ping » « pong ».
- **(-)checkFailedPongs(*idRobot*) : void** : Méthode interne permettant de vérifier si la var_connexion du robot indiqué par le paramètre *idRobot* a atteint la limite de « pong » non-reçus.
- **(-)createFailedPongVar(*idRobot*) : void** : Méthode interne permettant d'ajouter une var_connexion au robot présent passé en paramètre *idRobot*.
- **(-)updateFailedPongs(*value, idRobot*) : void** : Méthode interne permettant de changer la valeur (*value*) de la var_connexion d'un robot.
- **(-)pickFirstRobot(*robotList*) : void** : Méthode interne permettant d'attribuer à l'attribut *currentRobot* le premier robot de la *robotList* passée en paramètre.
- **(-)pickNextRobot(*robotList*) : void** : Méthode interne sélectionnant le prochain *currentRobot* à partir de la *robotList* et en fonction du currentRobot courant.
- **(-)setRobotToDisconnected(*idRobot*) : void** : Méthode interne permettant de changer l'état du robot ayant pour identifiant *idRobot* passé en paramètre à « déconnecté ».

2.3.3.13.3. Description comportementale

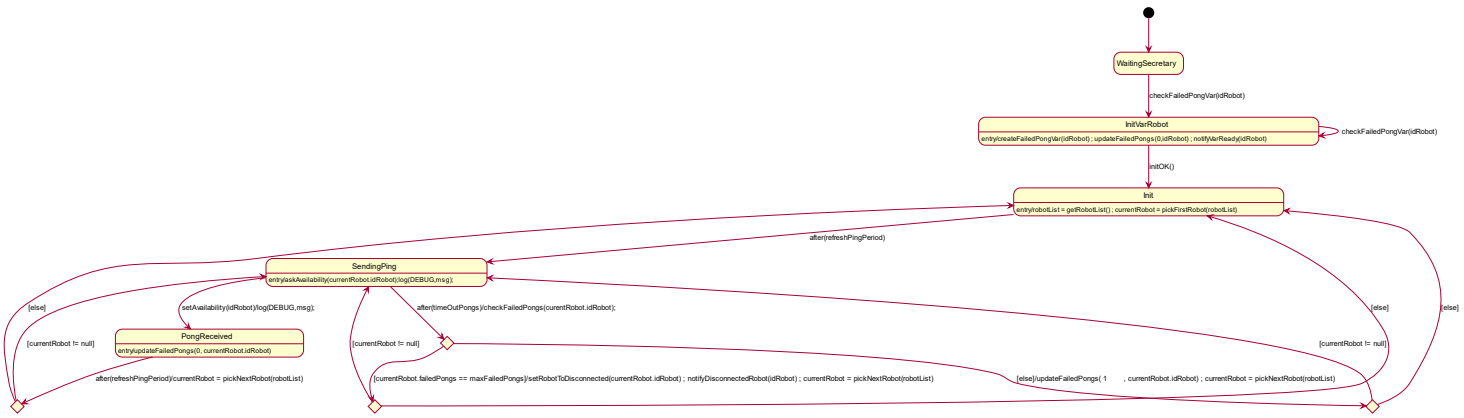


Figure 43 – Machine à états de GUIRinger

3. Conception détaillée

3.1. Architecture physique

3.1.1. Architecture matérielle de SWARMBOTS

3.1.1.1. Schéma de SWARMBOTS

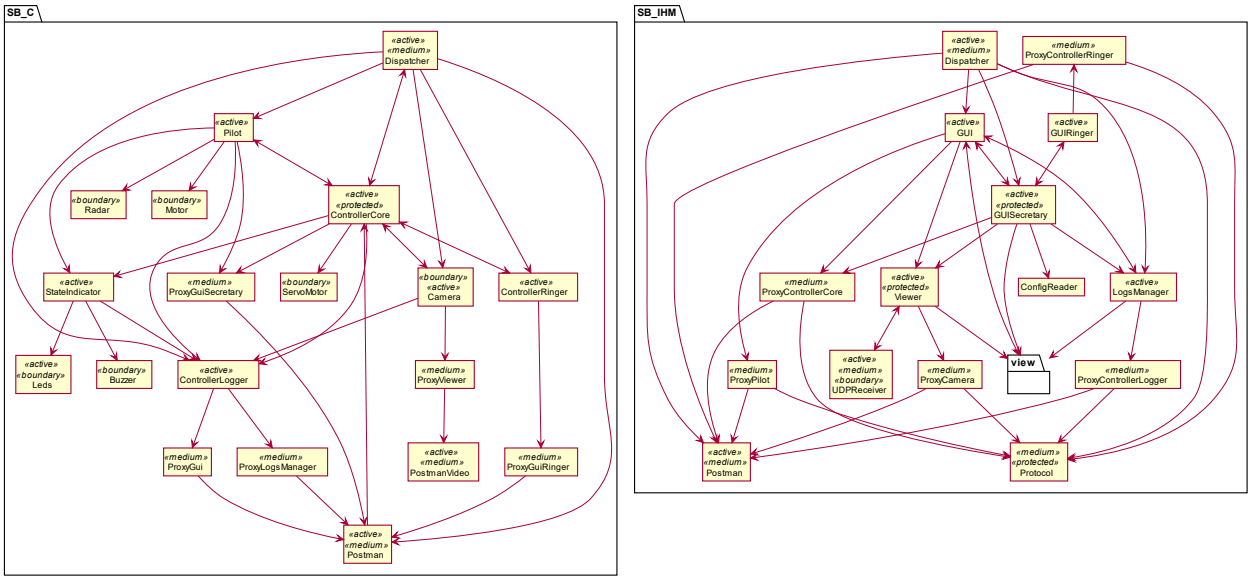


Figure 44 – Architecture de SwarmBots

3.2. Conception détaillée de SB_C

3.2.1. Description des classes de SB_C

La conception détaillée apporte un plus haut niveau de détails pour l'architecture de SB_C avec l'arrivée d'un objet Starter, instanciant les différents objets de l'application.

L'apparition du package de communication [Com] avec un postman, un dispatcher et des proxys. La communication par TCP/IP de SWARMBOTS se faisant par socket, certaines méthodes de la conception générale n'ont plus d'utilité car elles sont déjà implicitement implémentées par la technologie des sockets. Par exemple les méthodes « ackCmd », « ackConnection », « askToConnect », ces méthodes sont utilisées différemment, comme déclencheurs d'événements, par exemple, plutôt qu'envoyées par socket.

Ce package arrive avec le package alphabot 2 [Alphabot2], celui-ci regroupe tous les objets frontières faisant le lien entre « l'applicatif » SB_C et les périphériques « matériels » de l'alphabot 2 avec le radar, la camera, les leds, le buzzer, les servo-moteurs et les moteurs.

Le package de contrôle [Controller] permet de rassembler les fonctions qui font le « cœur » de l'application. Enfin le package [Logs], permet de rassembler les classes permettant de logger les actions de SB_IHM. Sur le schéma ci-dessous, ControllerLogger n'est relié à aucune classe par simplification de lecture du schéma, effectivement, toutes les classes utilisent ControllerLogger.

Dans le cadre de ce développement logiciel, les objets proxyViewer et PostmanVideo seront directement intégrés à l'objet Camera pour des raisons de simplification. En effet, l'émission du flux caméra en UDP se fera via une librairie existante appelée gstreamer. Ce dossier ne détaillera donc pas ces deux classes.

Ainsi, la classe camera n'utilise pas la méthode privé updateCamera pour récupérer l'image de la camera, elle est remplacée par 2 méthodes privées startStream() (méthode démarrant le transfert du flux vers la classe Viewer) et stopStream() (méthode arrêtant le transfert du flux vers la classe Viewer). De plus, la méthode publique setCamera() de Viewer est appelée par la bibliothèque gstreamer.

3.2.1.1. Schéma de SB_C

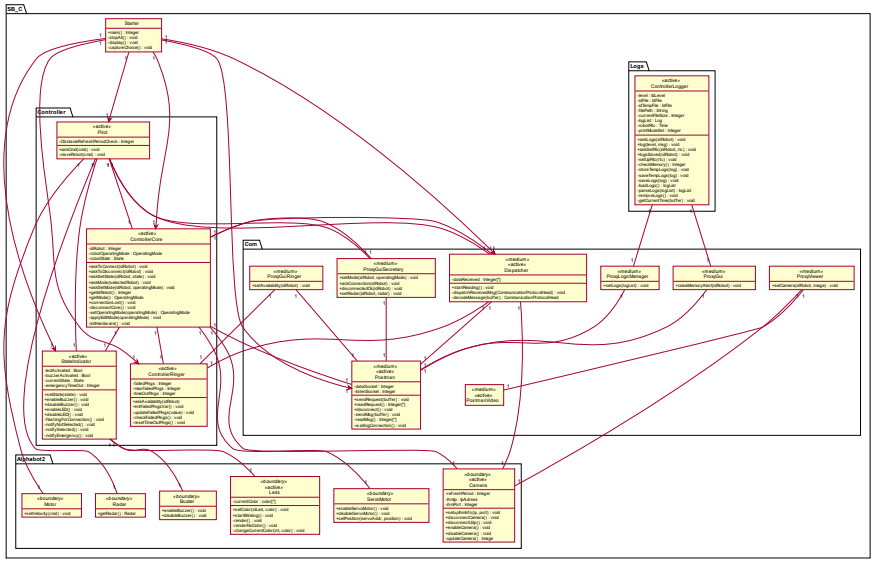


Figure 45 – Schéma de SB_C

3.2.1.2. [Object] La classe Starter

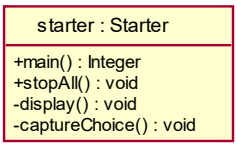


Figure 46 – Diagramme de classe représentant Starter

3.2.1.2.1. Philosophie de conception

La classe Starter a pour rôle de lancer l'application, c'est la classe qui va créer ou lancer la création de tous les objets utilisés par SB_C ainsi que de démarrer les différentes machines à état.

Cette classe permet aussi d'arrêter les différents objets actifs et de détruire les objets instanciés à l'arrêt de SB_C.

3.2.1.2.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)main() : Integer :** Première méthode appelée au lancement de SB_C. Instancie et démarre les classes du programme.
- **(+)stopAll() : void :** Arrêt de SB_C.
- **(-)display() : void :** Affichage d'une IHM en ligne de commandes.
- **(-)captureChoice() : void :** Capture des commandes de l'utilisateur.

3.2.1.2.3. Gestion du multitâche

Après l'instanciation des classes et le démarrage des différents objets <<active>>, le thread de Starter est bloqué dans l'attente d'un appel de stopAll() pour son extinction.

3.2.1.2.4. Séquence de démarrage et arrêt de SB_C

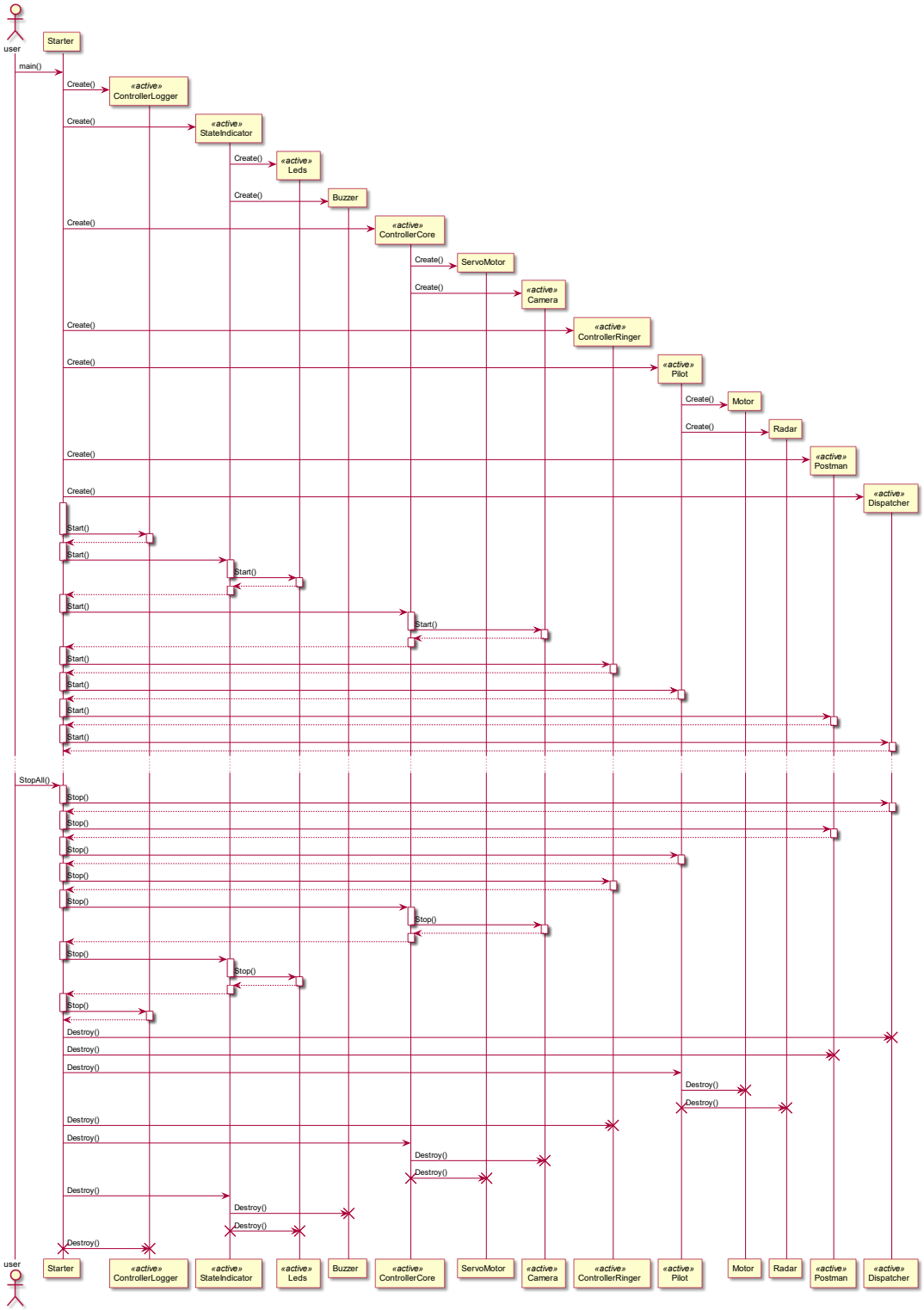


Figure 47 – Diagramme de séquence de l'initialisation et de l'arrêt de SB_C

3.2.1.3. [Com] La classe Postman

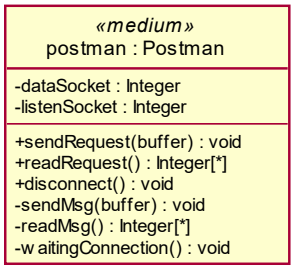


Figure 48 – Diagramme de classe représentant Postman

3.2.1.3.1. Philosophie de conception

La classe Postman se charge de gérer la socket de communication avec SB_IHM tout en prenant le rôle de serveur dans cette communication.

La classe Postman, pour rester générique, n'a pas la connaissance du protocole de communication, elle se charge d'envoyer des trames brutes reçues par les proxys ou de recevoir des trames brutes pour le dispatcher.

Cette classe est un objet actif car elle reste en attente d'arrivée de message ou d'évènements sur sa boîte au lettre pour, par la suite, envoyer d'éventuels messages vers le client (SB_IHM)

3.2.1.3.2. Description structurelle

Attributs

- ***dataSocket* : Integer** : Attribut permettant de stocker l'identifiant de la socket de donnée.
- ***listenSocket* : Integer** : Attribut permettant de stocker l'identifiant de la socket d'écoute.

Services offerts

- **(+)sendRequest(*buffer*) : void** : Permet aux proxy de mettre dans la file de message de la classe une demande d'envoi de message à travers la socket.
- **(+)readRequest() : Integer[]** : Permet au Dispatcher de lire les messages entrants sur la socket, le retour de cette fonction est la trame brute sous forme d'un buffer.
- **(+)disconnect() : void** : Permet d'indiquer à la classe qu'une déconnexion a été détectée.
- **(-)sendMsg(*buffer)* : void** : Envoi de la trame au format du protocole de communication à travers la socket sous forme d'un buffer.
- **(-)readMsg() : Integer[]** : Lecture des trames sur la socket.
- **(-)waitingConnection() : void** : Attends la connexion du client (SB_IHM) sur la socket.

3.2.1.3.3. Gestion du multitâche

La MàE de Postman s'exécute dans un thread qui est créé au lancement de SB_C après avoir connecté le client (SB_IHM). Les demandes d'envoi et consignes sont transmises à la classe à travers une boite aux lettres.

3.2.1.4. [Com] La classe Disptacher

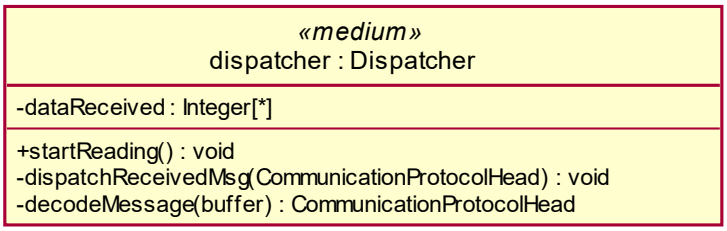


Figure 49 – Diagramme de classe représentant Dispatcher

3.2.1.4.1. Philosophie de conception

La classe Dispatcher est représentée comme un objet actif car c'est elle qui va rester en écoute d'un message entrant sur la socket. A l'arrivé d'un message, la classe décodera celui-ci pour ensuite appeler les classes ciblées par le message entrant.

3.2.1.4.2. Description structurelle

Attributs

- ***dataReceived* : Integer[]** : Trame brute du message reçu sur la socket sous la forme d'un buffer.

Services offerts

- **(-)dispatchReceivedMsg(*CommunicationProtocolHead*) : void** : Redirige les informations du message reçu vers les méthodes des classes cibles.
- **(-)decodeMessage(*buffer*) : CommunicationProtocolHead** : Traduit la trame brute reçue avec le protocole de communication établie.

3.2.1.4.3. Gestion du multitâche

La légère MàE de Dispatcher s'exécute dans un thread créé à la connexion d'un client, ce thread va venir se mettre en attente sur la méthode de lecture de la socket de Postman à l'arrivée d'un message. L'arrivée d'un message lancera le décodage et la redirection de celui-ci vers la méthode cible.

3.2.1.5. [Com] La classe ProxyGui

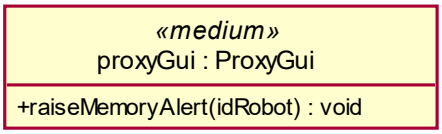


Figure 50 – Diagramme de classe représentant ProxyGui

3.2.1.5.1. Philosophie de conception

La classe ProxyGui qui se trouve dans SB_C va venir simuler le comportement de la classe GUI de SB_IHM. Toutes les classes de SB_C communiquant avec GUI, ici ControllerLogger, passent par ce proxy. Pour résumer, cette classe a pour but de « faire croire » aux classes l'utilisant qu'elles communiquent avec GUI. Ainsi elle traduit les données qui lui sont envoyés pour ensuite les passer à Postman.

3.2.1.5.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)raiseMemoryAlert(*idRobot*) : void** : Permet à LogsManager d'indiquer que la taille de l'emplacement alloué pour enregistrer les logs sur SB_C arrive à satturation.

3.2.1.6. [Com] La classe ProxyGuiSecretary

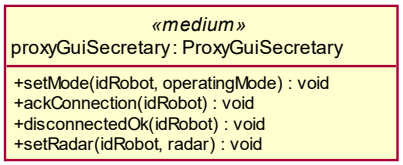


Figure 51 – Diagramme de classe représentant ProxyGuiSecretary

3.2.1.6.1. Philosophie de conception

La classe ProxyGuiSecretary qui se trouve dans SB_C va venir simuler le comportement de la classe GUISecretary de SB_IHM. Toutes les classes de SB_C communicant avec GUISecretary, ici ControllerCore, passent par ce proxy. Pour résumer, cette classe a pour but de « faire croire » aux classes l'utilisant qu'elles communiquent avec GUISecretary. Ainsi elle traduit les données qui lui sont envoyés pour ensuite les passer à Postman.

3.2.1.6.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)setMode(*idRobot, operatingMode*) : void** : Permet à ControllerCore d'envoyer son mode de fonctionnement.
- **(+)ackConnection(*idRobot*) : void** : Permet à ControllerCore de « valider » la connexion entre SB_C et SB_IHM. *(Cette fonction ne déclenchera qu'une entrée de log et pas d'écriture sur la socket car cette information de validation de connexion est déjà implicitement implémentée par la technologie des socket).*
- **(+)disconnectedOk(*idRobot*) : void** : Permet d'indiquer à SB_IHM que SB_C s'est bien déconnecté.
- **(+)setRadar(*idRobot, radar*) : void** : Permet à la classe Radar d'envoyer les données du radar de SB_C.

3.2.1.7. [Com] La classe ProxyGuiRinger

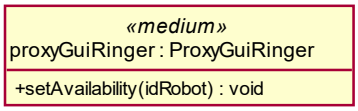


Figure 52 – Diagramme de classe représentant ProxyGuiRinger

3.2.1.7.1. Philosophie de conception

La classe ProxyGuiRinger qui se trouve dans SB_C va venir simuler le comportement de la classe GUIRinger de SB_IHM. Toutes les classes de SB_C communicant avec GUIRinger, ici ControllerRinger, passent par ce proxy. Pour résumer, cette classe a pour but de « faire croire » aux classes l'utilisant qu'elles communiquent avec GUIRinger. Ainsi elle traduit les données qui lui sont envoyés pour ensuite les passer à Postman.

3.2.1.7.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)setAvailability(*idRobot*) : void** : Permet à ControllerRinger de répondre par un « pong » au « ping » envoyé par GUIRinger.

3.2.1.8. [Com] La classe ProxyLogsManager

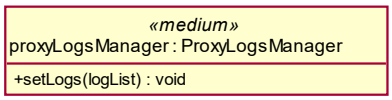


Figure 53 – Diagramme de classe représentant ProxyLogsManager

3.2.1.8.1. Philosophie de conception

La classe ProxyLogsManager qui se trouve dans SB_C va venir simuler le comportement de la classe LogsManager de SB_IHM. Toutes les classes de SB_C communicant avec LogsManager, ici ControllerLogger, passent par ce proxy. Pour résumer, cette classe a pour but de « faire croire » aux classes l'utilisant qu'elles communiquent avec LogsManager. Ainsi elle traduit les données qui lui sont envoyés pour ensuite les passer à Postman.

3.2.1.8.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)setLogs(*logList*) : void** : Permet à LogsManager d'envoyer les logs de SB_C à la demande de SB_IHM.

3.2.1.9. [Alphabot2] La classe Buzzer <boundary>

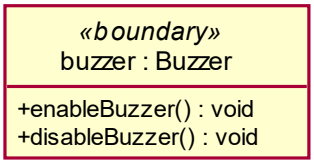


Figure 54 – Diagramme de classe représentant l'objet frontière Buzzer

3.2.1.9.1. Philosophie de conception

La classe Buzzer représente un objet frontière de SB_C. Cette classe fait le lien entre l'application (SB_C) et le périphérique matériel buzzer du robot. Les broches d'utilisation de ce périphérique sont initialisées à l'instanciation du module.

3.2.1.9.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)enableBuzzer() : void** : Mise à l'état haut de la pin du buzzer -> allumage du périphérique.
- **(+)disableBuzzer() : void** : Mise à l'état bas de la pin du buzzer -> désactivation du périphérique.

3.2.1.10. [Alphabot2] La classe Motor <boundary>

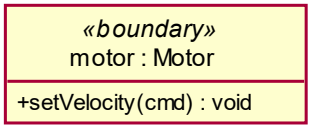


Figure 55 – Diagramme de classe représentant l'objet frontière Motor

3.2.1.10.1. Philosophie de conception

La classe Motor représente un objet frontière de SB_C. Cette classe fait le lien entre l'application (SB_C) et les périphériques matériels moteurs du robot. Les broches d'utilisation de ce périphérique sont initialisées à l'instanciation du module.

3.2.1.10.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)setVelocity(*cmd*) : void** : Change la direction du robot en fonction de ***cmd***.

3.2.1.11. [Alphabot2] La classe ServoMotor <boundary>

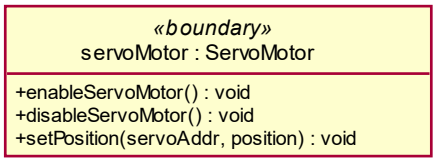


Figure 56 – Diagramme de classe de l'objet frontière ServoMotor

3.2.1.11.1. Philosophie de conception

La classe ServoMotor représente un objet frontière de SB_C. Cette classe fait le lien entre l'application (SB_C) et les périphériques matériels servo-moteurs du robot. Ce module est utilisé au démarrage de l'application pour remettre la camera dans le bon axe du robot.

3.2.1.11.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)enableServoMotor() : void** : Active le périphérique servo-moteur.
- **(+)disableServoMotor() : void** : Désactive le périphérique servo-moteur.
- **(+)setPosition(*servoAddr, position*) : void** : Change la position d'un servo-moteur en fonction de l'angle passé en paramètres, position comprise entre 0 et 180**.**

3.2.1.12. [Alphabot2] La classe Leds <boundary>

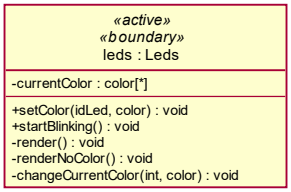


Figure 57 – Diagramme de classe de l'objet frontière Leds

3.2.1.12.1. Philosophie de conception

La classe Leds représente un objet frontière de SB_C. Cette classe fait le lien entre l'application (SB_C) et les périphériques matériels leds du robot. Ce module permet l'allumage des leds dans différentes couleurs et le clignotement de celles-ci.

3.2.1.12.2. Description structurelle

Attributs

- ***currentColor*** : **color[]** : Couleur actuellement affiché des leds.

Services offerts

- **(+)setColor(*idLeds, color*) : void** : Méthode publique permettant de changer la couleur des leds.
- **(+)startBlinking() : void** : Méthode publique permettant de faire clignoter les leds.
- **(-)render() : void** : Méthode interne mettant les leds à l'état de *currentColor.*
- **(-)renderNoColor() : void** : Méthode interne permettant d'éteindre les leds.
- **(-)changeCurrentColor(*int*, *color*) : void** : Méthode interne permettant de changer *currentColor*.

3.2.1.12.3. Gestion du multitâche

Le thread de l'objet frontière Leds est créé et démarré au lancement de SB_C, il sert à faire passer les leds de l'état haut à l'état bas et ainsi générer un clignotement.

3.2.1.12.4. Description comportementale

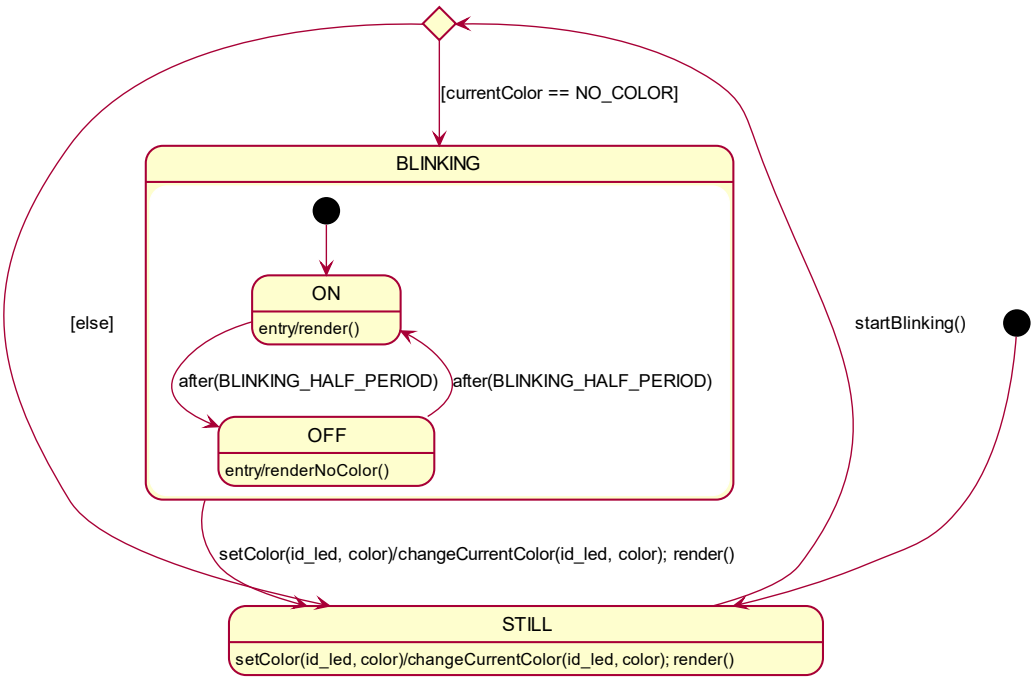


Figure 58 – Machine à état de Leds

3.2.2. Identification des accès concurrents côté SB_C

Le tableau suivant décrit quelles tâches (ici une tâche correspond à un thread) accèdent à quels objets. Ainsi, cela permet d'identifier les accès concurrents possibles, c'est-à-dire lorsque deux tâches peuvent accéder à la même donnée d'un objet. Les proxys sont exclus du tableau car ils ne possèdent pas d'attribut. De plus, les objets actifs communiquent de manière asynchrone entre eux, à l'aide de message queue, ce qui empêche les accès concurrents.

Tâches	Buzzer	ServoMotor	Radar	Motor	ControllerCore(passif)
Pilot			**x**	**x**	**x**
ControllerCore(actif)		**x**			**x**
StateIndicator	**x**				
ControllerRinger					
Leds					
Camera					**x**
Postman					
ControllerLogger					
Dispatcher					

L'objet actif ControllerCore comprend deux attributs (idRobot et robotOperatingMode) accessibles en lecture par les méthodes passives (synchrone) getMode() et getIdRobot(). Ces attributs étant accédés en écriture par le thread de ControllerCore, il est nécessaire de les protéger à l'aide de deux mutex pour éviter les accès concurrents.

3.3. Conception détaillée coté SB_IHM

3.3.1. Description des classes de SB_IHM

Le package SB_IHM est l'expression du logiciel de SB stocké sur la Tablette. Il sera remplacé par le package project/src/. Le package view, contient tous les modules liés aux activités, fragments et objets ViewModel de l'application. Son contenu est détaillé plus bas.

3.3.1.1. Schéma de SB_IHM

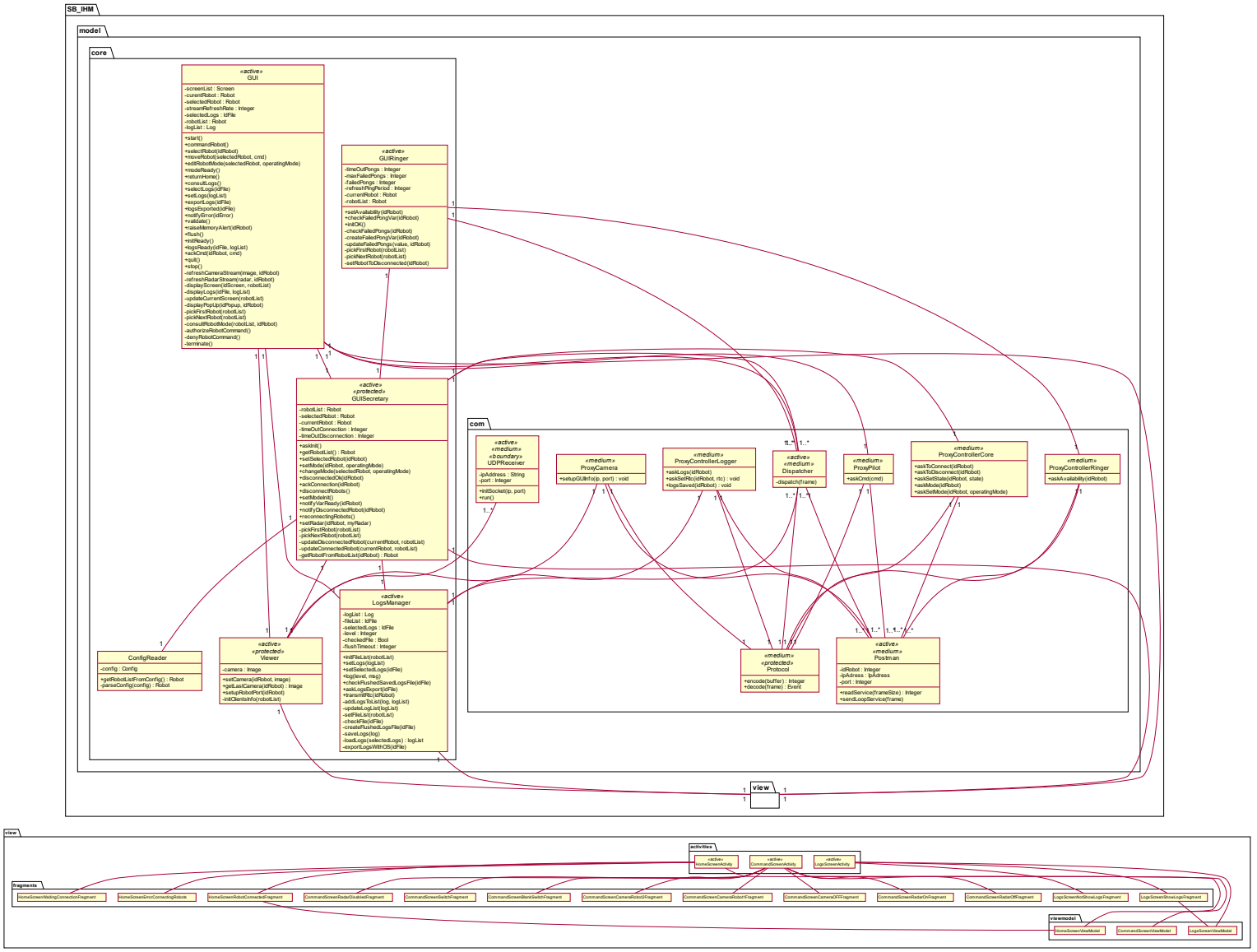


Figure 59 – Digramme de classe de SB_IHM & son package view

3.3.1.1.1. HomeScreenActivity <<active>>

3.3.1.1.1.1. Philosophie de conception

La classe HomeScreenActivity correspond au contrôleur de la page d'accueil de l'application. C'est le contrôleur qui va pouvoir charger tous les éléments nécessaires à l'utilisation de l'application.

3.3.1.1.2. CommandScreenActivity <<active>>

3.3.1.1.2.1. Philosophie de conception

La classe CommandScreenActivity correspond à la page de contrôle des différents robots ainsi que des affichages de leurs flux caméras.

3.3.1.1.3. LogsScreenActivity <<active>>

3.3.1.1.3.1. Philosophie de conception

La classe LogsScreenActivity correspond à l'activité permettant d'afficher les logs des différents robots (SB_C) ou de la tablette (SB_IHM).

3.3.1.1.4. HomeScreenWaitingConnectionFragment

3.3.1.1.4.1. Philosophie de conception

La classe HomeScreenWaitingConnectionFragment correspond au fragment permettant l'affichage d'un message en cours de connexion et d'initialiser la liste de robots.

3.3.1.1.5. HomeScreenRobotConnectedFragment

3.3.1.1.5.1. Philosophie de conception

La classe HomeScreenRobotConnectedFragment correspond au fragment permettant l'affichage de la liste de robots ainsi que leur état de connexion par rapport à l'application. Il affiche aussi un message indiquant l'appui sur l'écran pour continuer vers l'activité CommandScreenActivity.

3.3.1.1.6. HomeScreenErrorConnectingRobots

3.3.1.1.6.1. Philosophie de conception

La classe HomeScreenErrorConnectingRobots correspond au fragment permettant l'affichage d'un message d'erreur de connexion aux robots de la liste donnée.

3.3.1.1.7. CommandScreenCameraRobot1Fragment

3.3.1.1.7.1. Philosophie de conception

La classe CommandScreenCameraRobot1Fragment correspond au fragment permettant l'affichage du flux caméra reçu du premier robot.

3.3.1.1.8. CommandScreenCameraRobot2Fragment

3.3.1.1.8.1. Philosophie de conception

La classe CommandScreenCameraRobot2Fragment correspond au fragment permettant l'affichage du flux caméra reçu du second robot.

3.3.1.1.9. CommandScreenCameraOFFFragment

3.3.1.1.9.1. Philosophie de conception

La classe CommandScreenCameraOFFFragment correspond au fragment lorsque le flux camera est désactivé ou lorsqu'un robot est déconnecté et affiche le logo du projet swarmbots.

3.3.1.1.10. CommandScreenBlankSwitchFragment

3.3.1.1.10.1. Philosophie de conception

La classe CommandScreenBlankSwitchFragment correspond au fragment n'affichant pas les switchs si aucun robot est sélectionné.

3.3.1.1.11. CommandScreenSwitchFragment

3.3.1.1.11.1. Philosophie de conception

La classe CommandScreenSwitchFragment correspond au fragment affichant les switchs permettant de gérer les périphériques du robot.

3.3.1.1.12. CommandScreenRadarOnFragment

3.3.1.1.12.1. Philosophie de conception

La classe CommandScreenRadarOnFragment correspond au fragment permettant d'afficher à l'écran l'indicateur radar quand aucun obstacle n'est détecté par le radar du robot.

3.3.1.1.13. CommandScreenRadarOffFragment

3.3.1.1.13.1. Philosophie de conception

La classe CommandScreenRadarOffFragment correspond au fragment permettant d'afficher à l'écran l'indicateur radar quand un obstacle est détecté par le radar du robot.

3.3.1.1.14. CommandScreenRadarDisabledFragment

3.3.1.1.14.1. Philosophie de conception

La classe CommandScreenRadarDisabledFragment correspond au fragment permettant de cacher l'indicateur radar quand le périphérique radar est désactivé ou quand le robot est déconnecté.

3.3.1.1.15. LogsScreenShowLogsFragment

3.3.1.1.15.1. Philosophie de conception

La classe LogsScreenShowLogsFragment correspond au fragment permettant d'afficher les logs demandés par l'utilisateur.

3.3.1.1.16. LogsScreenNotShowLogsFragment

3.3.1.1.16.1. Philosophie de conception

La classe LogsScreenNotShowLogsFragment correspond au fragment permettant d'afficher à l'écran le logo de l'application à la place des logs.

3.3.1.1.17. HomeScreenViewModel

3.3.1.1.17.1. Philosophie de conception

Cette classe a pour but de faire évoluer les données sur les Views. Elle permet de faire la communication entre GuiSecretary et UI.

3.3.1.1.18. CommandScreenViewModel

3.3.1.1.18.1. Philosophie de conception

Cette classe a pour but de faire évoluer les données sur les Views. Elle permet de faire la communication entre GuiSecretary et UI, ainsi que Viewer et UI.

3.3.1.1.19. LogsScreenViewModel

3.3.1.1.19.1. Philosophie de conception

Cette classe a pour but de faire évoluer les données sur les Views. Elle permet de faire la communication entre GuiSecretary et UI, ainsi que GuiLogger et UI.

3.3.1.2. [Object] La classe Starter

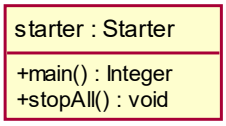


Figure 60 – Diagramme de classe de Starter

3.3.1.2.1. Philosophie de conception

La classe Starter a pour rôle d'arrêter les différents objets actifs et de détruire les objets instanciés à l'arrêt de SB_IHM. La méthode main() a un rôle fantôme pour représenter l'action de démarrage du système par Utilisateur.

Les objets actifs GUI et GUISecretary, étant des singletons basiques, sont créés et lancés au démarrage de l'application SB_IHM. Ces derniers ont pour rôle de créer et lancer les autres objets actifs de SB_IHM.

3.3.1.2.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)main() : Integer** : Méthode qui représente le démarrage de SB_IHM.
- **(+)stopAll() : void** : Arrêt de SB_IHM.

3.3.1.2.3. Gestion du multitâche

Après l'instanciation des classes et le démarrage des différents objets <>, le thread de Starter est en bloqué dans l'attente d'un appel de stopAll() pour son extinction.

3.3.1.2.4. Séquence de démarrage et arrêt de SB_IHM

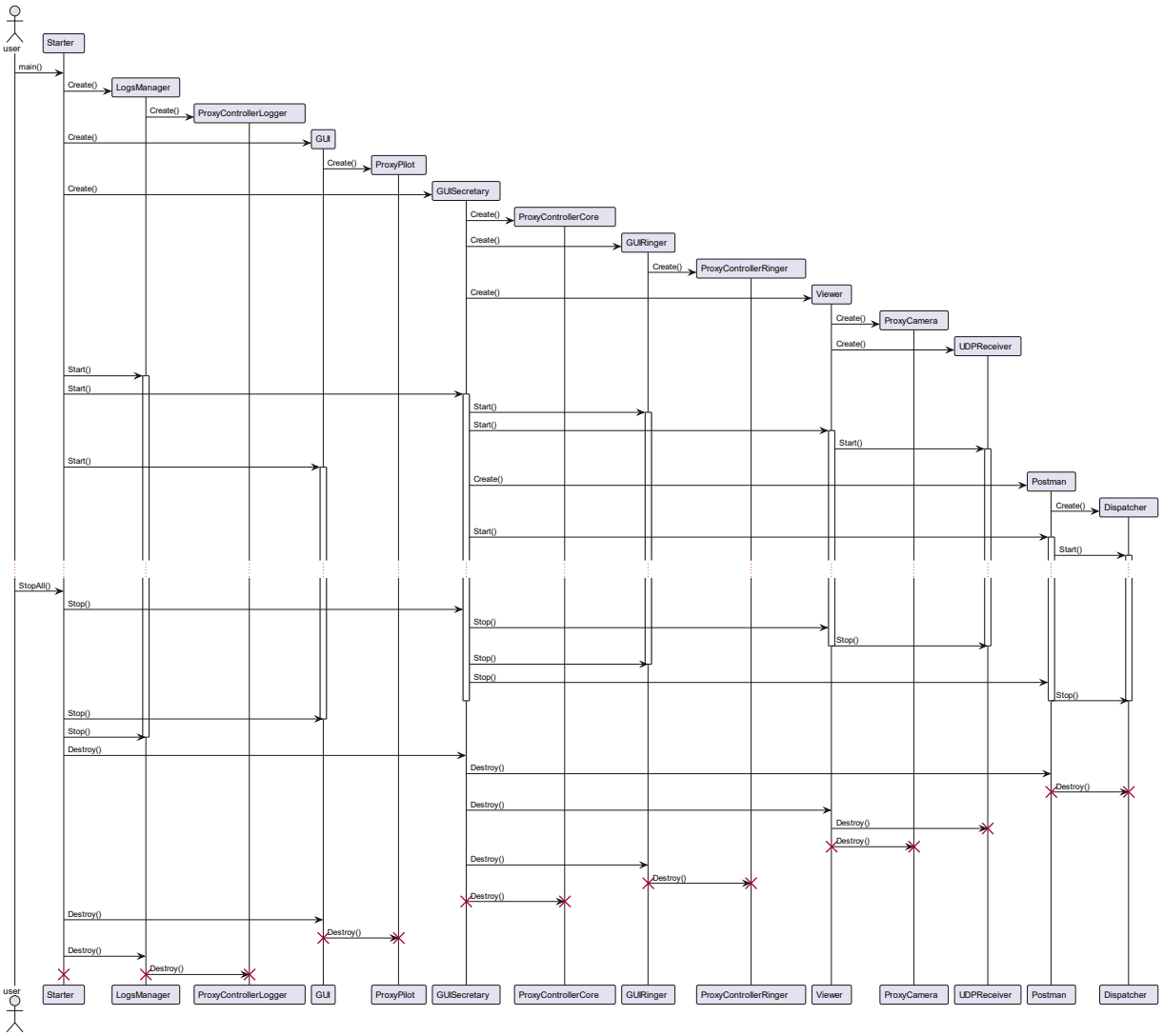


Figure 61 – Séquence de démarrage et d'arrêt de SB_IHM

3.3.1.3. [Com] La classe Postman

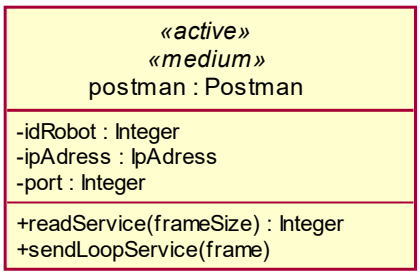


Figure 63 – Diagramme de classe de Postman

3.3.1.3.1. Philosophie de conception

La classe Postman se charge de gérer le socket de communication avec SB_C tout en prenant le rôle de client dans cette communication.

La classe Postman, pour rester générique, n'a pas la connaissance du protocole de communication, elle se charge d'envoyer des trames brutes reçues par les proxys ou de recevoir des trames brutes pour le dispatcher.

Cette classe est un objet actif car elle reste en attente d'arrivée de message ou d'événements sur sa boîte

aux lettres pour, par la suite, envoyer d'éventuels messages vers le serveur (SB_C).

Un objet de cette classe est créé pour chaque robot renseigné dans Fichier_Config pour lequel SB_IHM se connectera au démarrage de l'application. Enfin, elle a la responsabilité de créer et lancer un objet dispatcher qui lui est associé pour transmettre les messages reçus aux autres objets de SB_IHM. L'application assigne donc un objet de classe Postman et un objet de classe Dispatcher pour chaque robot.

3.3.1.3.2. Description structurelle

Attributs

- ***ipAdress :* IpAddress** : Adresse IP de SB_C du robot pour s'y connecter.
- ***port :* Port** : Port de SB_C du robot pour s'y connecter.
- ***idRobot :* Integer** : Identifiant du robot pour lequel l'objet est assigné.

Services offerts

- **(+)readService(*frameSize*) : Integer** : Réceptionne les messages envoyés de SB_C.
- **(+)sendLoopService(*frame*) : void** : Envoi à SB_C les messages qui sont stockés dans sa BAL.

3.3.1.4. [Com] La classe Dispatcher

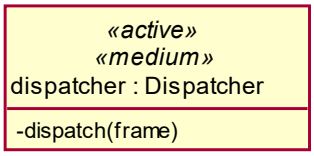


Figure 64 – Diagramme de classe de Dispatcher

3.3.1.4.1. Philosophie de conception

La classe Dispatcher est représentée comme un objet actif car c'est elle qui va rester en écoute d'un message entrant sur le socket, et va transmettre le message aux classes ciblées après l'avoir décodé avec la classe Protocole.
Les messages sont transmis sous la forme d'objet de la classe Event.

3.3.1.4.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)dispatch(*frame*) : void** : Transmet les messages sous forme d'événements aux autres objets.

3.3.1.4.3. Gestion du multitâche

La légère MàE de Dispatcher s'exécute dans un thread qui va venir se mettre en attente sur la méthode de lecture du socket de Postman pour réceptionner les messages.

3.3.1.5. [Com] La classe Protocol

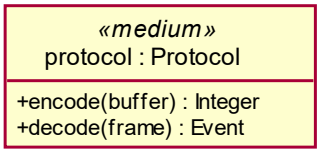


Figure 65 - Diagramme de classe de Protocol

3.3.1.5.1. Philosophie de conception

Les autres objets proxy font appel à la classe statique Protocole pour encoder les messages avant de les envoyer, et les objets dispatcher pour décoder les messages en objets de la classe Event.

3.3.1.5.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)encode(*buffer*) : Integer** : Encode les messages pour qu'ils soient envoyés à SB_C.
- **(+)decode(*frame*) : Event :** Décode les trames reçus en objet de la classe Event.

3.3.1.6. [Com] La classe ProxyControllerCore

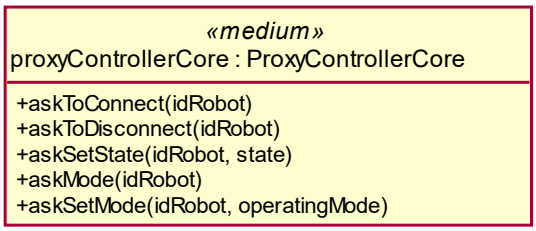


Figure 66 – Diagramme de classe de ProxyControllerCore

3.3.1.6.1. Philosophie de conception

La classe ProxyControllerCore qui se trouve dans SB_IHM va venir simuler le comportement de la classe ControllerCore de SB_C. Toutes les classes de SB_IHM voulant communiquer avec ControllerCore passent par ce proxy. Autrement dit, c'est l'intermédiaire de communication qui expose les fonctions publiques de ControllerCore avec lesquelles les objets de SB_IHM communiquent.

3.3.1.6.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)askToConnect(*idRobot*) : void** : Demande de connexion au SB_C du

robot identifié par idRobot.

- **(+)askToDisconnect(*idRobot*) : void :** Demande de déconnexion au SB_C du robot identifié par idRobot.
- **(+)askSetState(*idRobot*, *state*) : void :** Demande de changement de l'état de sélection au SB_C du robot identifié par idRobot.
- **(+)askMode(*idRobot*) : void :** Demande de récupération du Mode_Fonctionnement au SB_C du robot identifié par idRobot.
- **(+)askSetMode(*idRobot*, *operatingMode*) : void :** Demande de changement du Mode_Fonctionnement du SB_C du robot identifié par idRobot.

3.3.1.7. [Com] La classe ProxyControllerRinger

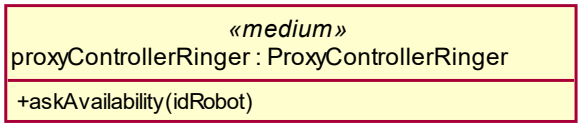


Figure 67 – Diagramme de classe de ProxyControllerRinger

3.3.1.7.1. Philosophie de conception

La classe ProxyControllerRinger qui se trouve dans SB_IHM va venir simuler le comportement de la classe ControllerRinger de SB_C. Elle sert d'intermédiaire et expose la fonction publique de ControllerRinger avec laquelle GUIRinger puisse communiquer.

3.3.1.7.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)askAvailability(*idRobot*) : void** : S'assure de la disponibilité du robot identifié par *idRobot*.

3.3.1.8. [Com] La classe ProxyControllerLogger

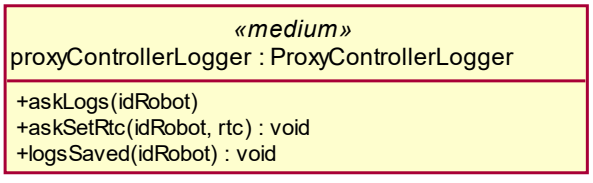


Figure 68 – Diagramme de classe de ProxyControllerLogger

3.3.1.8.1. Philosophie de conception

La classe ProxyControllerLogger qui se trouve dans SB_IHM va venir simuler le comportement de la classe ControllerLogger de SB_C. Elle sert d'intermédiaire et expose la fonction publique de ControllerLogger avec laquelle LogsManager puisse communiquer.

3.3.1.8.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)askLogs(*idRobot*) : void** : Demande d'envoyer les logs de SB_C du robot identifié par *idRobot*.
- **(+)askSetRtc(*idRobot, rtc*) : void** : Envoie à SB_C son heure courante pour que SB_C puisse définir la sienne.
- **(+)logsSaved(*idRobot*) : void** : Permet d'indiquer à SB_C que SB_IHM a fini de recevoir les logs.

3.3.1.9. [Com] La classe ProxyPilot

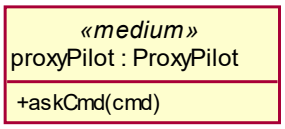


Figure 69 – Digramme de classe de ProxyPilot

3.3.1.9.1. Philosophie de conception

La classe ProxyPilot correspond au proxy permettant de faire une demande pour pouvoir piloter le robot (avance, reculer, droite, tache, stop).

3.3.1.9.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)askCmd(cmd) : void** : Envoie une demande au robot pour piloter le robot avec la commande choisie.
-

3.3.1.10. [Com] La classe ProxyCamera

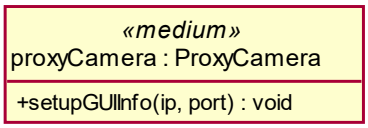


Figure 70 – Diagramme de classe de ProxyCamera

3.3.1.10.1. Philosophie de conception

La classe ProxyCamera correspond au proxy permettant d'envoyer l'adresse ip et le port au robot pour configurer son système UDP.

3.3.1.10.2. Description structurelle

Attributs

N.A.

Services offerts

- **(-)setupGUIInfo(*ip*, *port*)** : Envoie l'IP et le port au robot pour configurer l'UDP de la camera.

3.3.2. Identification des accès concurrents côté SB_IHM

Le tableau suivant décrit quelles tâches (ici une tâche correspond à un thread) accèdent à quels objets. Ainsi, cela permet d'identifier les accès concurrents possibles, c'est-à-dire lorsque deux tâches peuvent accéder à la même donnée d'un objet. Les proxys sont exclus du tableau car ils ne possèdent pas d'attribut. De plus, les objets actifs communiquent de manière asynchrone entre eux, à l'aide de message queue, ce qui empêche les accès concurrents, sauf exceptions listées après le tableau des accès concurrents.

TÂCHES	ConfigReader	Protocol	ProxyControllerRinger	ProxyPilot	ProxyControllerLogger	ProxyControllerCore	ProxyCamera	Viewer(passif)	GUISecretary(passif)
View (package)									
GUI		x		x		x		x	x
GUISecretary(actif)	x	x				x		x	
GUIRinger		x	x						x
Viewer(actif)		x					x		
LogsManager		x			x				
UDPReceiver								x	
Postman									
Dispatcher		x							
Concurrence	*OK*	*<<protected>>*	*OK*	*OK*	*OK*	*OK*	*OK*	*<<protected>>*	*<<protected>>*

L'objet actif Viewer comprend un attribut camera accessible en lecture par la méthode passive (synchrone) getLastCamera(). Cet attribut étant également accédé en écriture par le thread de Viewer, il est nécessaire de le protéger à l'aide de mutex pour éviter les accès concurrents. L'appel de Viewer par GuiSecretary via la méthode setupRobotPort(idRobot) lui permet de démarrer les connexions d'écoute des flux vidéo, ainsi, il faut également protéger l'objet de façon a ce qu'il puisse instancier les threads correctement.

De même, l'objet actif GuiSecretary comprend un attribut robotList accessible en lecture par les méthode passives (synchrone) getRobotList() et getRobotFromList(id_robot). Cet attribut étant également accédé en écriture par le thread de GuiSecretary, il est nécessaire de le protéger à l'aide de mutex pour éviter les accès concurrents.

L'objet passif Protocol est également sujet aux problèmes de concurrence car il est sollicité par 4 objets actifs, ainsi, il sera également protégé à l'aide de mutex.

Le package view manipulant des ViewModel (Android), nous considérerons qu'il n'y a pas de concurrence dans le cadre de cette étude.

3.4. Protocole de communication

3.4.1. Protocole de communication entre SB_C et SB_IHM

3.4.1.1. Formalisation du protocole

Les communications entre les deux composants logiciels suivront le même protocole de communication défini de la forme suivante :

Size	Msg	Data

- **Size :** Taille totale du message, comprenant la taille des champs Msg et Data. Size sera envoyée sur 2 octets ce qui laisse la possibilité d'avoir une taille maximale de 65535 octets de données à envoyer.
- **Msg :** Type du message transmis, il sera envoyé sous la forme de deux octets ce qui laisse 65535 possibilités de types différents.
- **Data :** Il s'agit de la donnée à envoyer. Sa taille est variable (de 0 à 65533 octets) c'est pourquoi le champ Size est nécessaire.

Size et Msg constituent l'entête de la trame envoyée, cet header de 4 octets reste constant en taille.
Les commandes sont formatées en hexadécimal. Les logs sont formatés en chaîne de caractères ascii.

3.3.1.2. Types de messages

Le tableau ci-dessous détail les types de messages définis dans le protocole de communication.

Identifiant en hexadécimal	Msg	Commentaires
0x0100 (256)	ASK_AVAILABILITY	SB_IHM envoie un « ping » à SB_C.
0x0200 (512)	SET_AVAILABILITY	SB_C répond à SB_IHM par un « pong ».
0x0300 (768)	ASK_CMD	SB_IHM envoie une commande de direction à SB_C.
0x0400 (1024)	SET_STATE	SB_IHM indique à SB_C son nouvel état.
0x0500 (1280)	ASK_MODE	SB_IHM demande à SB_C son mode de fonctionnement.
0x0600 (1536)	SET_MODE	Message utilisé par les deux applications : - Par SB_IHM pour changer les modes de fonctionnement de SB_C. - Par SB_C pour donner ses modes de fonctionnement à la demande de SB_IHM.
0x0700 (1792)	ASK_LOGS	SB_IHM demande les logs de SB_C.
0x0800 (2048)	SET_LOGS	SB_C donne ses logs à SB_IHM.
0x0900 (2304)	ALERT	Un message d'alerte est envoyé par SB_C à SB_IHM.
0x1000 (4096)	ASK_TO_DISCONNECT	SB_IHM indique à SB_C qu'il se déconnecte.
0x1100 (4352)	ACK_DISCONNECTION	SB_C acquitte la déconnexion à SB_IHM.
0x1200 (4608)	SET_RADAR	SB_C donne l'état du radar.
0x1300 (4864)	SET_CURRENT_TIME	SB_IHM indique son heure courante à SB_C.
0x1400 (5120)	SET_IP_PORT	SB_IHM donne son adresse ip et son port pour diffuser le flux caméra en UDP.
0x1500 (5376)	LOGS_RECEIVED	SB_IHM indique à SB_C qu'il a bien reçu l'entièreté des logs.

3.3.1.3. Détails des messages

Sur les 12 types de messages, 6 messages n'ont pas de données à embarqués ce ne sont que de simples commandes, ces messages sont : ASK_AVAILABILITY, SET_AVAILABILITY, ACK_DISCONNECTION, ASK_TO_DISCONNECT, ASK_LOGS, ASK_MODE.

Pour les 6 autres, le tableau ci-dessous détail les données envoyés par type de message sauf pour SET_LOGS et SET_IP_PORT :

Identifiant	**Msg**	**Data (1 octet = 1 case)**						
0x0300 (768)	ASK_CMD	***cmd_type***						
0x0400 (1024)	SET_STATE	***state***						
0x0600 (1536)	SET_MODE	***mode*** camera	***mode*** radar	***mode*** buzzer	***mode*** leds			
0x0900 (2304)	ALERT	***id_alert***						
0x1200 (4608)	SET_RADAR	***Radar***						
0x1300 (4864)	SET_CURRENT_TIME	**Year**	**Year**	**Month**	**Day**	**Hours**	**Minutes**	**Seconds**
0x1400 (5120)	SET_IP_PORT	**ip**	**Ip**	**Ip**	**ip**	**Port** *(1 à 2 octets)*		

(cf : dictionnaire de domaine page 82 et description des types manipulés entre composants page 20).

Pour le type de message SET_LOGS [0x0800 (2048)], la taille de la **Data** est différente en fonction du nombre de logs à envoyer, pour une trame, il est possible d'envoyer au maximum 65531 octets de logs. Néanmoins, le nombre de logs peuvent dépasser cette limite, c'est pourquoi un indicateur de page de 2 octets sera ajouté au début de la trame. Forme d'entête d'une trame de logs :

Identifiant	**Msg**	**Data (1 octet = 1 case)**	
0x0800 (2048)	SET_LOGS	*Page courante*	*Pages totales*

Ici, *Pages totales* indique le nombre total de trames envoyés contenant des logs et *Page courante* indique à quelle trame de logs nous sommes rendus. De cette manière, il est plus simple d'envoyer plusieurs trames de logs à la suite et de les récupérer dans l'ordre pour SB_IHM. (cf partie Exemples pour des trames d'exemple).

3.3.1.4. Exemples

3.3.1.4.1. ASK_AVAILABILITY

SB_IHM envoie un « ping » à SB_C à travers la classe GUIRinger, la trame suivante est transmise :

Size	**Msg**	**Data**
0x00 0x02	0x01 0x00	

Ici, ASK_AVAILABILITY est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x0100 ce qui correspond à la requête ASK_AVAILABILITY.

3.3.1.4.2. SET_AVAILABILITY

SB_C répond au « ping » à SB_IHM par un « pong » à travers la classe ControllerRinger, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x02 0x00	

Ici, SET_AVAILABILITY est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x0200 ce qui correspond à la requête SET_AVAILABILITY.

3.3.1.4.3. ASK_CMD

SB_IHM transmet à SB_C une commande de déplacement à travers la classe GUI, la trame suivante est transmise :

Size	Msg	Data
0x00 0x03	0x03 0x00	0x00
0x00 0x03	0x03 0x00	0x01
0x00 0x03	0x03 0x00	0x02
0x00 0x03	0x03 0x00	0x03
0x00 0x03	0x03 0x00	0x04

Ici, ASK_CMD est une trame de 5 octets :

- **Size** vaut 0x03 (3) ce qui correspond aux deux octets de **Msg** plus l'octet de **Data.**
- **Msg** vaut 0x0300 ce qui correspond à la requête ASK_CMD.
- **Data** vaut :
  - Première trame : 0x00 (0) -> vers l'avant. (FORWARD)
  - Deuxième trame : 0x01 (1) -> vers la droite. (RIGHT)
  - Troisième trame : 0x02 (2) -> vers la gauche. (LEFT)
  - Quatrième trame : 0x03 (3) -> vers l'arrière. (BACKWARD)
  - Cinquième trame : 0x04 (4) -> arrêt. (STOP)

3.3.1.4.4. SET_STATE

SB_IHM transmet à SB_C le nouvel état dans lequel il se trouve, la trame suivante est transmise :

Size	Msg	Data
0x00 0x03	0x04 0x00	0x00
0x00 0x03	0x04 0x00	0x01
0x00 0x03	0x04 0x00	0x02
0x00 0x03	0x04 0x00	0x03

Ici, ASK_CMD est une trame de 5 octets :

- **Size** vaut 0x03 (3) ce qui correspond aux deux octets de **Msg** plus l'octet de **Data.**
- **Msg** vaut 0x0400 ce qui correspond à la requête SET_MODE.
- **Data** vaut :
  - Première trame : 0x00 (0) -> en attente de connexion (waiting for connection).
  - Deuxième trame : 0x01 (1) -> état d'alerte (emergency).
  - Troisième trame : 0x02 (2) -> sélectionné (selected).
  - Quatrième trame : 0x03 (3) -> non-sélectionné (not selected).

3.3.1.4.5. ASK_MODE

SB_IHM demande à SB_C son mode de fonctionnement, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x05 0x00	

Ici, ASK_MODE est une trame de 4 octets :
- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x0500 ce qui correspond à la requête ASK_MODE.

3.3.1.4.6. SET_MODE

SB_IHM ou SB_C transmettent leur mode de fonctionnement, les trames suivantes correspondent à des modes de fonctionnement différents :

Size	Msg	Data
0x00 0x06	0x06 0x00	0x00 0x01 0x00 0x00
0x00 0x06	0x06 0x00	0x01 0x01 0x01 0x01

Ici, SET_MODE est une trame de 8 octets :
- **Size** vaut 0x06 (6) ce qui correspond aux deux octets de **Msg** plus les 6 octets de **Data.**
- **Msg** vaut 0x0600 ce qui correspond à la requête SET_MODE.
- **Data** vaut :
  - Première trame : 0x00010000 : 1er octet (0x00) la caméra est désactivée. 2ème octet (0x01) le radar est activé. 3ème octet (0x00) le buzzer est désactivé. 4ème octet (0x00) les leds sont désactivées.
  - Deuxième trame : 0x01010101 : tous les périphériques sont activés.

Pour ce type de message, le décodage de **Data** se fait octet par octet et la valeur correspond à l'état du périphérique correspondant.

3.3.1.4.7. ASK_LOGS

SB_IHM demande les logs de SB_C, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x07 0x00	

Ici, ASK_LOGS est une trame de 4 octets :
- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x0700 ce qui correspond à la requête ASK_LOGS.

3.3.1.4.8. SET_LOGS

SB_C transmet ses logs à SB_IHM, l'exemple suivant montre l'envoie d'un message de test (« ceci est un test ») :

Size	Msg	Data
0x00 0x14	0x08 0x00	0x01 0x01 0x63 0x65 0x63 0x69 0x20 0x65 0x73 0x74 0x20 0x75 0x6E 0x20 0x74 0x65 0x73 0x74

Ici, SET_LOGS est une trame de 22 octets :
- **Size** vaut 0x14 (20) ce qui correspond aux deux octets de **Msg** plus les 18 octets de **Data**.
- **Msg** vaut 0x0800 ce qui correspond à la requête SET_LOGS.
- **Data** vaut 0x0101636563692065737420756E2074657374.

- En prenant les deux premiers octets qui correspondent à l'indicateur de page. 0x0101 : le 1er octet 0x01 indique que nous sommes à la page 1. Le 2ème octet 0x01 indique que nous avons un total d'une page.
- Le reste du message correspond à des chaines de caractères contenant les logs, ici 0x6365636920657374207 56E2074657374 correspond à « ceci est un test » en ascii.

Pour l'exemple qui suit, nous allons modéliser 3 pages de logs à transmettre uniquement avec les indicateurs de page dans **Data**, effectivement il est compliqué de modéliser des trames de 65535 octets :

Size	Msg	Data
0xFF 0xFF	0x08 0x00	0x01 0x03
0xFF 0xFF	0x08 0x00	0x02 0x03
0x00 0xFF	0x08 0x00	0x03 0x03

Ici, les première et deuxième trames ont une taille (**Size**) de 65535 octets, 2 octets pour **Msg** et 65533 octets pour **Data**. La dernière trame, elle, à une taille (**Size**) de 255 octets, 2 octets pour **Msg** et 253 octets pour **Data**.

Si l'on se penche sur le champ **Data**, sans oublier que pour simplifier il n'y a que les indicateurs de pages, l'indicateur de page de la première trame indique que nous sommes rendus à la page 0x01 (1) sur 0x03 (3) (première trame de logs), pour la deuxième, l'indicateur indique que nous sommes à la page 0x02 (2) sur 0x03 (3) (deuxième trame de logs) et pour la dernière trame, l'indicateur indique que nous sommes à la page 0x03 (3) sur 0x03 (3) (dernière trame de logs).

3.3.1.4.9. ALERT

SB_C indique un état d'alerte à SB_IHM, la trame suivante est transmise :

Size	Msg	Data
0x00 0x03	0x09 0x00	0x00

Ici, ALERT est une trame de 5 octets :

- **Size** vaut 0x03 (3) ce qui correspond aux deux octets de **Msg** et à l'octet de **Data**.
- **Msg** vaut 0x0900 ce qui correspond à la requête ALERT.
- **Data** vaut 0x00 (0) ce qui correspond à une alerter mémoire.

3.3.1.4.10. ASK_TO_DISCONNECT

SB_IHM indique à SB_C qu'il se déconnecte, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x10 0x00	

Ici, ASK_TO_DISCONNECT est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x1000 ce qui correspond à la requête ASK_TO_DISCONNECT.

3.3.1.4.11. ACK_DISCONNECTION

SB_C acquitte à SB_IHM sa déconnexion, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x11 0x00	

Ici, ACK_DISCONNECTION est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**.
- **Msg** vaut 0x1100 ce qui correspond à la requête ACK_DISCONNECTION.

3.3.1.4.12. SET_RADAR

SB_C transmet la valeur de détection de son radar à SB_IHM, la trame suivante est transmise :

Size	Msg	Data
0x00 0x03	0x12 0x00	0x00
0x00 0x03	0x12 0x00	0x01

Ici, SET_RADAR est une trame de 5 octets :

- **Size** vaut 0x03 (3) ce qui correspond aux deux octets de **Msg** plus l'octet de **Data**.
- **Msg** vaut 0x1200 ce qui correspond à la requête SET_RADAR.
- **Data** vaut :
  - Première trame : 0x00 : le radar n'a pas détecté d'obstacle (OBJECT_NOT_DETECTED).
  - Deuxième trame : 0x01 : le radar a détecté un obstacle (OBJECT _DETECTED).

3.3.1.4.13. SET_CURRENT_TIME

SB_IHM transmet à SB_C sa date et son temps courant afin que SB_C puisse avoir les mêmes repères de temps que SB_IHM, la trame suivante est transmise :

Size	Msg	Data
0x00 0x09	0x13 0x00	0x07 0xE7 0x05 0x18 0x0B 0x2C 0x22

Ici, SET_CURRENT_TIME est une trame de 11 octets :

- **Size** vaut 0x09 (9) ce qui correspond aux deux octets de **Msg** plus les 7 octets de **Data**.
- **Msg** vaut 0x1300 ce qui correspond à la requête SET_CURRENT_TIME.
- **Data** vaut
  - Pour les deux octets d'années : 0x07E7 (2023).
  - Pour l'octet du mois : 0x05 (05).
  - Pour l'octet du jour : 0x18 (24).
  - Pour l'octet de l'heure : 0x0B (11).
  - Pour l'octet des minutes : 0x2C (44).
  - Pour l'octet des secondes : 0x22 (34).

3.3.1.4.14. SET_IP_PORT

SB_IHM transmet son adresse IP et son port pour que le périphérique caméra puisse diffuser un flux vidéo à travers l'UDP, la trame suivante est transmise :

Size	Msg	Data
0x00 0x07	0x14 0x00	0x7F 0x00 0x00 0x01 0x16

Ici, SET_IP_PORT est une trame de 15 octets :

- **Size** vaut 0x07 (7) ce qui correspond aux deux octets de **Msg** plus les 5 octets de **Data**.
- **Msg** vaut 0x1400 ce qui correspond à la requête SET_IP_PORT.
- **Data** vaut :

- Les 4 premiers octets correspondent à l'adresse Ip qui est : 0x7F (127) puis 0x00 (0) puis 0x00 (0) puis 0x01 (1).
- Le dernier octet, 0x16, qui correspond au port est 22.

3.3.1.4.15. LOGS_RECEIVED

SB_IHM indique à SB_C qu'il a reçu tous les logs envoyés précédemment, la trame suivante est transmise :

Size	Msg	Data
0x00 0x02	0x15 0x00	

Ici, LOGS_RECEIVED est une trame de 4 octets :
- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de **Msg**..
- **Msg** vaut 0x1500 ce qui correspond à la requête LOGS_RECEIVED.

4. Dictionnaire de domaine

4.1. Définitions

Terme	Définition
IPAdress	Adresse IP d'un serveur. C'est une chaine de caractère constituée d'une suite de 4 nombres compris entre 0 et 255 et séparés par un point. Exemple : « 192.168.1.42 ». L'adresse IP de ***SB_IHM*** est transmise à ***SB_C*** pour la communication UDP selon le protocole de communication : en conservant les zéros transparents, sans les points de séparation. En reprenant l'exemple précédent : « 192168001042 ».
Ecran_Accueil	Terme désignant l'écran de ***SB_IHM*** utilisé pour l'accueil de l'application.
Ecran_Commande	Terme désignant l'écran de ***SB_IHM*** permettant de commander le robot.
Ecran_Logs	Terme désignant l'écran de ***SB_IHM*** permettant de consulter les logs.
cmd_type	Type de commande de déplacement envoyée par SB_IHM à SB_C : - FORWARD : 0x00 (0). - RIGHT : 0x01 (1). - LEFT : 0x02 (2). - BACKWARD : 0x03 (3). - STOP : 0x04 (4).
currentRobot	Terme désignant le robot couramment sélectionné.
id_alert	[Valeur énumérée] Cette énumération indique un identifiant d'alerte envoyé de SB_C à SB_IHM par le message de type ALERT, les valeurs sont : - MEMORY : 0x00 (0). -
idRobot	Terme correspondant à l'identifiant d'un robot.
Lancement d'un objet actif	Exécution du comportement attendu de l'objet. Cela représente la mise en fonctionnement de sa machine à états et des interactions qu'il peut avoir avec les autres objets.
mode	[Valeur énumérée] Cette énumération indique l'état du mode de fonctionnement et est utilisée pour le type de message SET_MODE dans le protocole de communication pour décrire le mode de fonctionnement des périphériques, ces valeurs sont : - DISABLED : 0x00 (0).

	- ENABLED : 0x01 (1). **(cf : page 20)**
operatingMode	Correspond au ***mode*** de fonctionnement du robot ou ***Mode_Fonctionnement***. Regroupe les modes d'activation des périphériques présents sur *Carte_Elec* : • Etat_Camera • Etat_Radar • Etat_Buzzer • Etat_LEDs **(cf : page 20)**
Radar	[Valeur énumérée] Cette énumération indique l'état de détection du radar (présence ou non d'un obstacle), cette valeur est utilisée pour le type de message SET_RADAR du protocole de communication et prends les valeurs : - OBJECT_NOT_DETECTED : 0x00 (0). - OBJECT_DETECTED : 0x01 (1). **(cf : page 20)**
Raspberry Pi	Carte électronique représentée par Carte_Elec et utilisée comme support d'exécution de SB_C.
refreshPeriod	[Constante] Cette constante correspond au temps de temporisation de la mise à jour de l'image capturée par la caméra et le radar.
refreshPingPeriod	[Constante] Cette constante correspond au temps d'attente du déclenchement de l'envoi du ping pour vérifier le maintien de la connexion entre SB_IHM et SB_C.
streamRefreshRate	[Constante] Cette constante correspond au temps de rafraîchissement de l'écran commande pour afficher la dernière image capturée par la caméra et le dernier état du radar.
SB_C	Logiciel exécuté sur Robot et, communicant avec SB_IHM.
SB_IHM	Logiciel exécuté sur Tablette, communicant avec SB_C.
state	[Valeur énumérée] Cette énumération est utilisée pour indiquer l'état courant de SB_C, cette valeur est utilisée en paramètres de fonctions mais également en donnée sur une trame du protocole de communication correspondant au type de message SET_STATE, les valeurs sont : - WAITING_FOR_CONNECTION : 0x00 (0). - EMERGENCY : 0x01 (1). - SELECTED : 0x02 (2). - NOT_SELECTED : 0x03 (3). **(cf : page 20)**
timeOutPongs	[Constante] Cette constante correspond au temps d'attente pour lequel GUIRinger considère que ControllerRinger ne répond pas.
var_connexion	Variable présente sur chaque ***SB_C*** et sur ***SB_IHM*** pour chaque robot dans ***Liste_Robots***. Elle permet de stocker les erreurs de connexion entre ***SB_IHM*** et ***SB_C*** afin de détecter de potentielles déconnexion. Cette variable est incrémentée lorsque les erreurs de connexion surviennent.