Dossier de Conception

ProSE B1 2024 - CANvengers
Passerelle Android-CAN vers banc CAN réel ou simulé

Responsable du document	Elisa DECLERCK
État du document	En réalisation
Version	2.0
Révision	0

AVERTISSEMENT :

Le présent document est un document à but pédagogique. Le document de conception ci-joint est strictement confidentiel et réservé à un usage interne. Il a été réalisé sous la direction de Jérôme DELATOUR, en collaboration avec des enseignants et des étudiants de l'option SE du groupe ESEO. Ce document est la propriété de Jérôme DELATOUR, du groupe ESEO. Toute utilisation, diffusion ou reproduction de ce document sans autorisation écrite préalable de Jérôme DELATOUR est interdite. Nous tenons à souligner que toute violation de cette politique pourrait engager la responsabilité civile et pénale de son auteur. Nous vous demandons de prendre toutes les précautions nécessaires pour assurer la sécurité et la confidentialité de ce document.

Date	Actions	Auteur	Version	Révision
05/04/2023	Création du document	Elisa Declerck	0.0	0
03/04/2023	Réalisation du diagramme de séquence de "Démarrer le SàE"	Paul TRÉMOUREUX	0.0	1
07/04/2023	Rédaction de Portée	Gabriel MARQUETTE	0.0	2
07/04/2023	Rédaction de Objet	Gabriel MARQUETTE	0.0	3
08/04/2023	Réalisation de certains diagrammes de séquence	Théo BÉNARD	0.0	4
10/04/2023	Correction des diagrammes de séquence	Elisa DECLERCK	0.0	5
12/04/2023	Rédaction de la machine à états de Sender	Gabriel MARQUETTE	0.0	6
12/04/2023	Rédaction des descriptions des classes Sender, Basket et Network	Gabriel MARQUETTE	0.0	7
12/04/2023	Rédaction de la description générale des classes suivante : GUI, UI, Dealer, Logger, Object, Frame, Sniffer	Paul TRÉMOUREUX	0.0	8
12/04/2023	Rédaction de diagrammes de séquence	Thomas ROCHER	0.0	9
12/04/2023	Rédaction de diagrammes de séquence	Camille LENNE	0.0	10
13/04/2023	Correction du CU "Démarrer le SàE"	Paul TRÉMOUREUX	0.0	11
13/04/2023	Relecture et correction des doublons de descriptions générales de toutes les classes	Paul TRÉMOUREUX	0.1	0
13/04/2023	Rédaction de la description de l'architecture candidate	Elisa DECLERCK	0.1	1
13/04/2023	Ajout des multiplicitées au diagramme de classe	Elisa DECLERCK	0.1	2
15/04/2023	Relecture des parties Références, Types de données et Services Offerts	Camille LENNE	0.1	3

15/04/2023	Relecture et correction de la description des diagrammes de séquence	Camille LENNE	0.1	4
16/04/2023	Correction CU Reconnecter l'application CANdroid	Thomas ROCHER	0.1	5
16/04/2023	Correction de la description de l'architecture candidate	Thomas ROCHER	0.1	6
16/04/2023	Inclusion de la MAE de GUI	Elisa DECLERCK	0.1	7
17/04/2023	Relecture section 2.2 + rédaction de note	Thomas ROCHER	0.2	0
17/04/2023	Relecture partie + rédaction de note	Gabriel MARQUETTE	0.2	1
17/04/2023	Rédaction de la description de la MAE	Elisa DECLERCK	0.2	2
18/04/2023	Correction du schéma et de la description de la MAE de GUI	Elisa DECLERCK	0.2	3
18/04/2023	Correction de la section 2.2	Thomas ROCHER	0.2	4
19/04/2023	Correction du dossier dans sa globalité	Théo BÉNARD	1.0	0
26/04/2023	Correction de certaines descriptions des diagrammes de séquence	Elisa DECLERCK	1.0	1
27/04/2023	Création de la MAE de UI et rédaction de sa description	Elisa DECLERCK	1.0	2
27/04/2023	Correction de la MAE de GUI	Gabriel MARQUETTE	1.0	3
27/04/2023	Correction de l'ensemble du dossier à la suite de l'audit consultatif	Elisa DECLERCK	1.0	3
30/04/2023	Relecture de la partie CANgateway de la conception détaillée	Thomas ROCHER	1.1	0
24/05/2023	Ajout de l'architecture du programme CANgateway	Elisa DECLERCK	1.1	1
26/05/2023	Correction de la MAE de GUI	Elisa DECLERCK	1.1	2

26/05/2023	Rédaction de la description des classes du programme CANgateway	Elisa DECLERCK	1.1	3
28/05/2023	Rédaction de la description des classes proxyGUI, proxyLogger, Dispatcher et Postman du programme CANgateway	Thomas ROCHER	1.1	4
30/05/2023	Rédaction des différents diagrammes de classes	Gabriel MARQUETTE	1.1	5
30/05/2023	Rédaction du protocole de communication	Thomas ROCHER	1.1	6
31/05/2023	Relecture de la conception détaillée	Thomas ROCHER	1.2	0
05/06/2023	Correction dossier de conception	Elisa DECLERCK	1.2	1
05/06/2023	Correction et relecture de la conception détaillée coté Android	Camille LENNE	1.2	2
09/06/2023	Correction du dossier après audit normatif de code	Elisa DECLERCK	1.2	3
13/06/2023	Correction du dossier de conception dans sa globalité	Thomas ROCHER	2.0	0

TABLE 2 – Table des évolutions et validations internes du document

Table des matières

1 Introduction

1.1 Objet

Ce dossier de conception a pour objectif de rassembler toute la conception du logiciel "Passerelle Android-CAN vers banc CAN réel ou simulé". Il permettra à l'équipe CANvengers de développer le logiciel ainsi que d'élaborer les tests.

Les éléments de conception présentés dans ce document ont été déterminés suite à l'étude du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

Ce dossier de conception suit les recommandations de la norme [ISO/IEC/IEEE 29148 : 2018]. Il utilise des schémas et illustrations respectant la norme UML en version 2.5 [UML_2.5]. Il respecte également les exigences du Plan d'Assurance Qualité Logicielle ([PAQL_B1_2024]) défini par l'équipe CANvengers.

1.2 Portée

Ce document a pour but de présenter les éléments de conception du Système à l'Étude (SàE). Il est destiné à plusieurs parties prenantes :

- L'équipe de développement C et celle de développement Android, pour préciser l'implémentation des objets constituants le SàE.
- Les testeurs, pour élaborer les tests appropriés vérifiant la philosophie de conception adoptée par l'équipe.
- Les auditeurs de la société FORMATO lors de leurs différents suivis du projet.
- Le Client, afin de clarifier le cadre du projet et la direction prise par l'équipe, en continuité avec les spécifications.

1.3 Définitions, 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 conception sont définis dans le dictionnaire de domaine présent dans ce dossier, dans la section 4.

Acronymes/Abréviations	Définitions
Client	Société KEREVAL, Numéro SIRET 44278921000030.
CU	Cas d'utilisation.
ID	Identifiant.
IEEE (*Institute of Electrical and Electronics Engineers*)	Association professionnelle internationale définissant des normes dans le domaine informatique et électronique.
IHM (Interface Homme Machine)	Moyens permettant aux utilisateurs de l'application CANdroid d'interagir avec le programme CANgateway.

MAE (*Machine à états*)	Modèle qui décrit le comportement d'un système à travers un ensemble fini d'états et de transitions.
N.A.	Non Applicable.
OMG (*Object Management Group*)	Consortium international à but non lucratif créé en 1989, dont l'objectif est de standardiser et de promouvoir le modèle objet sous toutes ses formes.
SàE (Système à l'Étude)	Ensemble composé de l'application Android, CANdroid, et du programme en C, CANgateway.
SSH (*Secure Shell*)	Protocole de communication sécurisé.
UML (*Unified Modeling Language*)	Notation graphique normalisée, définie par l'OMG et utilisée en génie logiciel.
TCP/IP (*Transmission Control Protocol/Internet Protocol*)	Protocole de communication utilisé pour transmettre des données entre l'application CANdroid et le programme CANgateway.

1.4 Références

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

[CdC_KEREVAL_2023]	Société KEREVAL "Cahier des charges : développement d'une Passerelle Android-CAN vers banc CAN réel ou simulé", 2023.
[ISO/IEC/IEEE 29148 : 2018]	International standard, systems and software engineering life cycle processes requirements engineering, 2018, https ://standards.ieee.org/standard/29148-2018.html.
[UML_2.5]	OMG, Unified Modeling Language, version 2.5, 2015.
[Simulateur ICSim]	Société KEREVAL, Simulateur d'un tableau de bord de voiture, version 3, 2007.
[PAQL_B1_2024]	P. Trémoureux et T. Bénard, Plan d'Assurance Qualité Logicielle, version 0.3.0, 2023, `Git/doc/qualite/PAQL/version/`.
[plan_de_test _TEST_B1_2024]	P. Trémoureux et T. Bénard, Plan de test, version 0.5.1, 2023, `Git/doc/test/plan_test/livrables/`.
[dossier_de_specification_SPEC_B1_2024]	CANvengers, Dossier de specification, version 1.4, 2023, `Git/doc/specification/livrables/`.

1.5 Vue d'ensemble

Ce document de conception est structuré en 4 parties :

- La première partie présente les objectifs et la portée de ce document.
- La deuxième partie concerne la conception générale du logiciel "Passerelle Android-CAN vers banc CAN réel ou simulé". Cette partie présente l'architecture candidate et donne les grands principes de fonctionnement du projet. Elle détaille ensuite chaque composante du système, en présentant pour chacune leur description structurelle. Une description comportementale est présente pour les composantes actives ayant une machine à états.
- La troisième partie présente la conception détaillée. Cette partie présente les composantes du système en précisant cette fois-ci la gestion des entrées et des sorties, le multitâche ainsi que la gestion de la persistance.
- La quatrième partie présente le dictionnaire de domaine. Ce dictionnaire contient la liste des termes techniques utilisés dans le document ainsi que leur définition.

2 Conception générale

2.1 Architecture candidate

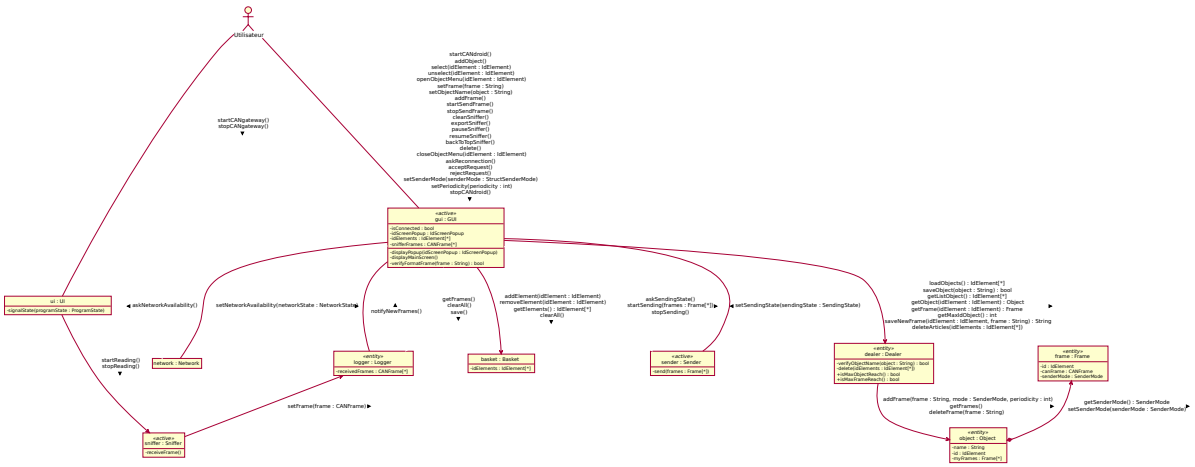

FIGURE 1 – Architecture candidate

Le diagramme de la figure 1 représente l'architecture candidate du système.

Il s'agit de la conception générale ; l'hypothèse d'un système matériel à ressources infinies est pour l'instant posée.

- Dans ce diagramme, on retrouve deux objets représentant des interfaces homme-machine :
  - *ui*, permettant de démarrer ou arrêter le programme CANgateway et d'informer Utilisateur du bon fonctionnement du programme
  - *gui*, permettant de démarrer et arrêter l'application CANdroid. Il permet aussi à Utilisateur de réaliser divers actions et d'afficher les écrans (EcranPrincipal et Popup).
- L'objet *dealer* stocke et fournit à *gui* les informations nécessaires à l'affichage des écrans. Il peut ajouter de nouveaux objets ou trames ou supprimer des éléments sélectionnés par Utilisateur et stockés dans *basket*.
- L'objet *object* permet de stocker et récupérer les informations d'une instance d'un objet, tout comme l'objet *frame* permettant de stocker et récupérer les informations d'une instance d'une trame.
- L'objet *basket* contient l'ensemble des objets et des trames sélectionnés par Utilisateur.
- L'objet *sender* permet d'envoyer les trames sélectionnées par Utilisateur et stockées dans *basket*.
- L'objet *sniffer* permet de récupérer les trames reçues par le bus CAN et de les stocker dans l'objet *logger*.
- L'objet *logger* permet de stocker les trames reçues par le bus CAN et de notifier *gui* qu'une nouvelle trame doit être affichée.

— L'objet *network* permet d'informer *gui* de l'état de connexion entre l'application CANdroid et le programme CANgateway.

2.2 Diagrammes de séquence

2.2.1 *CU Échanger des trames CAN*

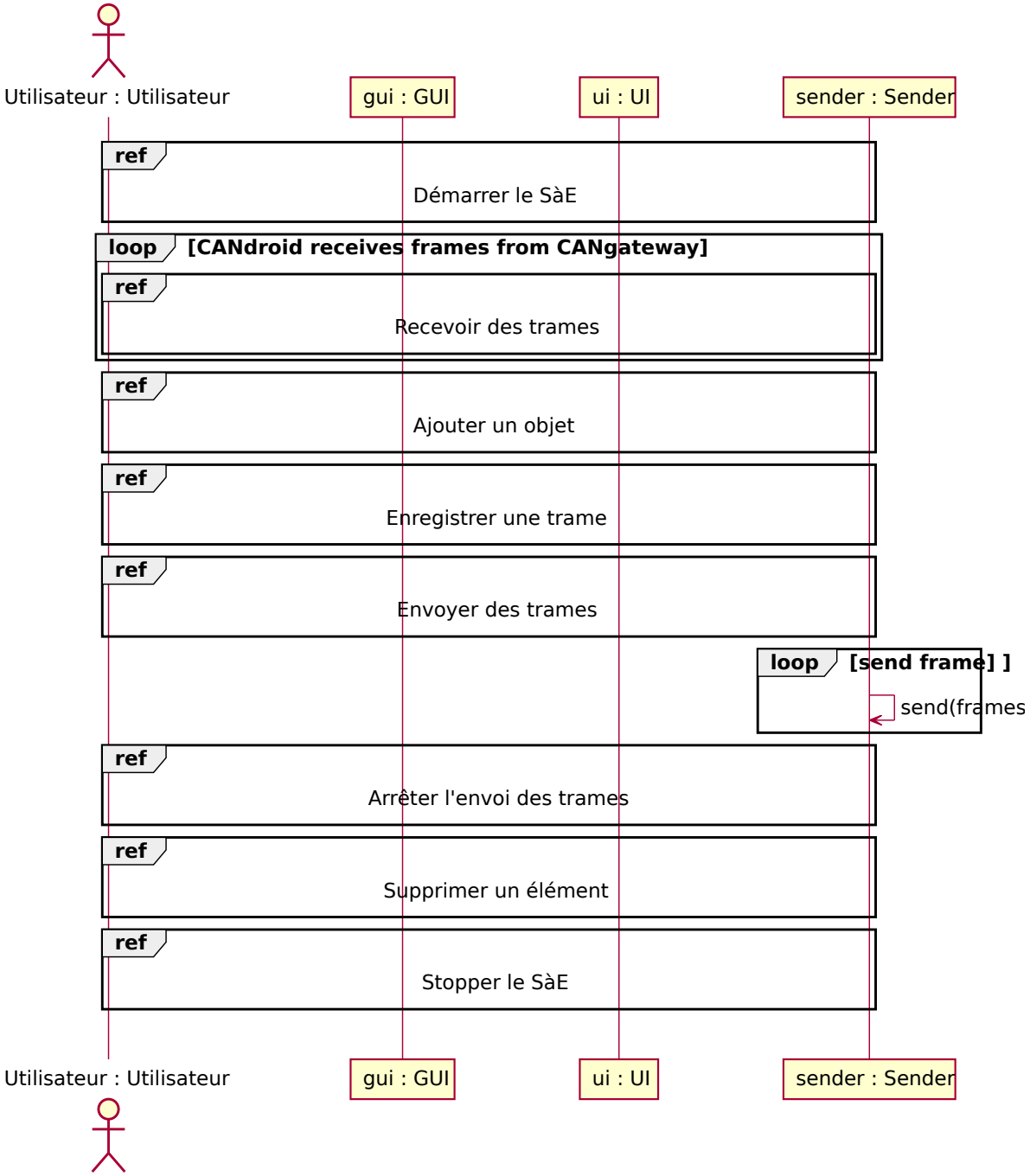

FIGURE 2 – Diagramme de séquence du *CU Échanger des trames CAN*

Le diagramme de la figure 2 représente le diagramme de séquence du *CU Échanger des trames CAN*.
Utilisateur démarre le SàE, et il commence à recevoir continuellement des trames. Utilisateur peut ensuite, à partir de l'application CANdroid, exercer différentes actions : ajouter un objet, enregistrer une trame, envoyer des trames, arrêter l'envoi de trames, supprimer un objet et/ou une trame et stopper le SàE.
Sur le diagramme, Utilisateur effectue toutes ces actions dans un ordre précis mais en réalité, il peut effectuer ces actions dans l'ordre qu'il souhaite, ou bien, n'en effectuer aucune.
En cas de perte de connexion entre l'application CANdroid et le programme CANgateway, Utilisateur peut les reconnecter ensemble. Ceci ne figure cependant pas sur le diagramme de la figure 2, qui ne montre que le scénario nominal.

2.2.2 *CU Démarrer le SàE - Scénario nominal*

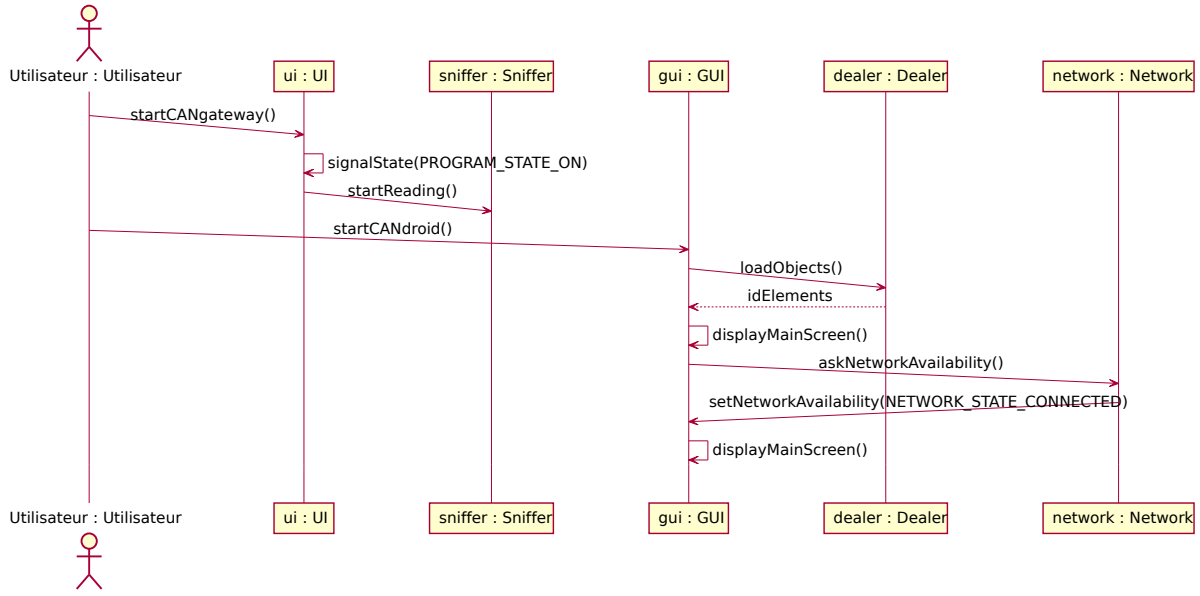

FIGURE 3 – Diagramme de séquence du *CU Démarrer le SàE - Scénario nominal*

Le diagramme de la figure 3 représente le diagramme de séquence du *CU Démarrer le SàE - Scénario nominal*.
Pour rappel, en dehors de la portée du système, Utilisateur met en fonctionnement Tableau de Bord, connecte la Raspberry PI au bus CAN et met la Raspberry PI sous tension.
Utilisateur doit ensuite démarrer manuellement le programme CANgateway via une connexion SSH avec la Raspberry PI. Quand le programme CANgateway est démarré, la LED de la Raspberry PI renvoie l'information à Utilisateur. Le programme CANgateway commence alors à lire les trames du bus CAN.
Ensuite, Utilisateur démarre l'application CANdroid et EcranPrincipal s'affiche. L'application CANdroid charge l'ensemble des éléments créés lors des précédentes utilisations et les affiche

sur EcranPrincipal.
Enfin, l'application CANdroid se connecte au programme CANgateway et EcranPrincipal se met à jour.
Il est possible que la connexion entre l'application CANdroid et le programme CANgateway échoue. Dans ce cas, PopupReconnexion s'affiche et demande à Utilisateur s'il souhaite réessayer. Ce scénario n'est pas représenter sur la figure 3.

2.2.3 *CU Reconnecter application CANdroid - Scénario nominal*

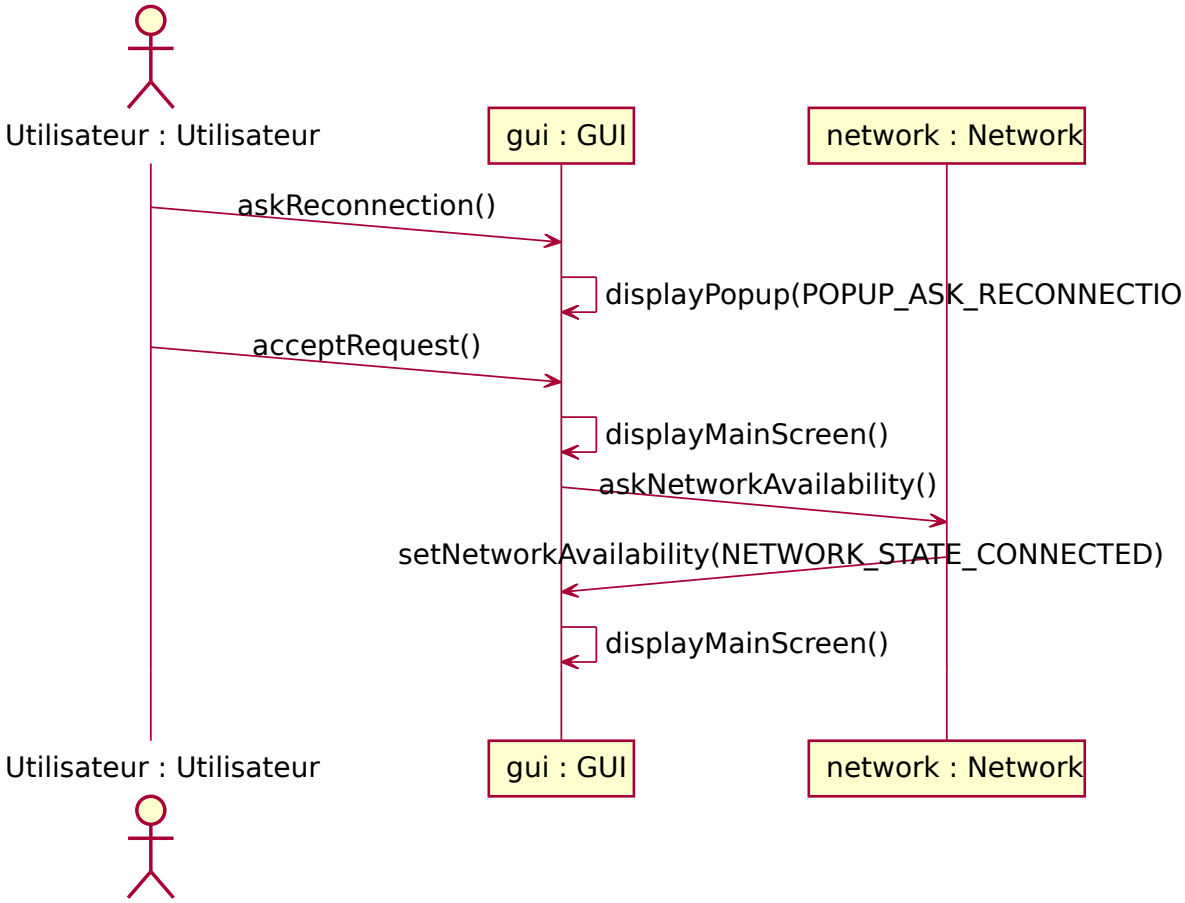

FIGURE 4 – Diagramme de séquence du *CU Reconnecter application CANdroid - Scénario nominal*

Le diagramme de la figure 4 représente le diagramme de séquence du *CU Reconnecter application CANdroid - Scénario nominal*.
Si la connexion entre l'application CANdroid et le programme CANgateway échoue, Utilisateur peut demander de les reconnecter. Dans ce cas, l'application CANdroid affiche PopupDemandeReconnexion. PopupDemandeReconnexion s'affiche également automatiquement en cas de

perte de connexion imprévue.
Si Utilisateur confirme la demande de reconnexion, EcranPrincipal s'affiche et l'application CANdroid tente alors de se reconnecter au programme CANgateway. Si la connexion aboutit, l'application CANdroid met à jour EcranPrincipal.
Dans le cas général, Utilisateur peut également refuser la reconnexion, et l'application CANdroid affiche simplement EcranPrincipal sans tenter de se reconnecter. Utilisateur peut aussi redemander la reconnexion suite à un échec. Ces cas de figure ne sont pas présentés sur ce scénario, mais sont explicitement décrits dans le CU "Reconnecter l'application CANdroid" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.4 *CU Recevoir des trames - Scénario nominal*

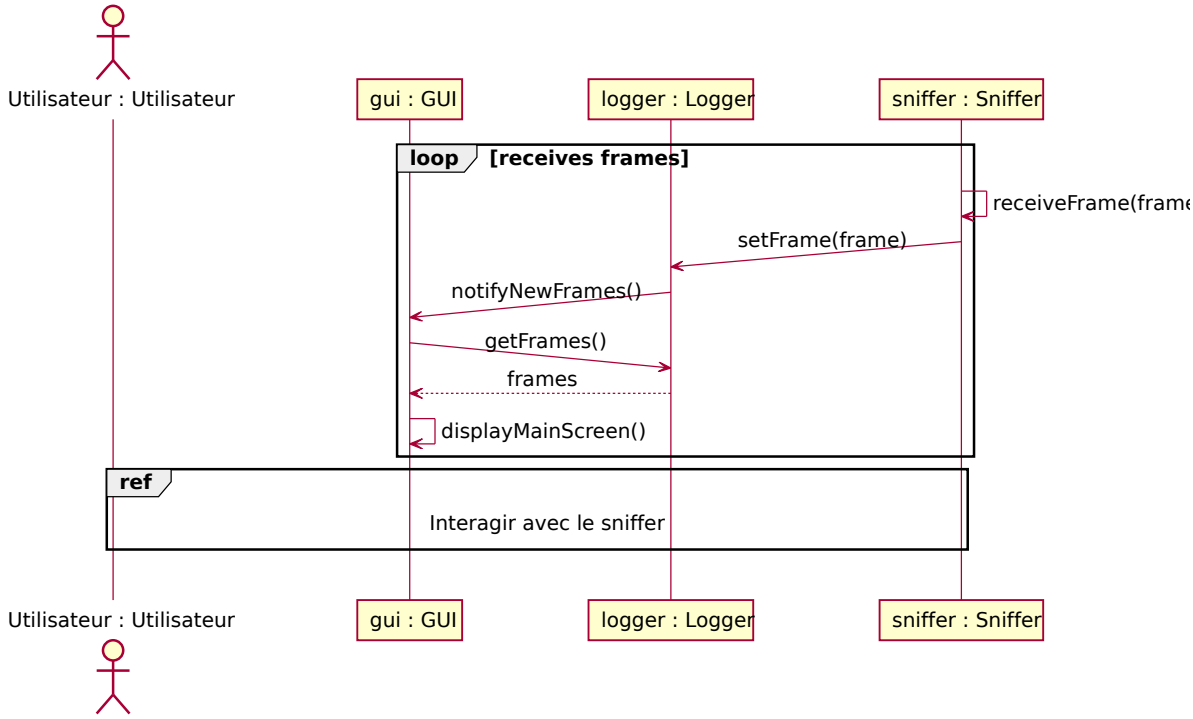

FIGURE 5 – Diagramme de séquence du *CU Recevoir des trames - Scénario nominal*

Le diagramme de la figure 5 représente le diagramme de séquence du *CU Recevoir des trames - Scénario nominal*.
La première étape du scénario nominal du CU "Recevoir des trames" du dossier de spécification [dossier_de_specification_SPEC_B1_2024] n'est pas représentée dans le diagramme de séquence car elle est implicite. On suppose que Tableau de Bord fonctionne et est capable d'émettre des trames sur le bus CAN.
Périodiquement, *sniffer* reçoit une trame de Tableau de Bord. Lorsqu'il reçoit une trame, il transmet celle-ci à *logger* qui informe ensuite *gui* de la réception d'une nouvelle trame. Cela permet ainsi à *gui* de récupérer et afficher cette trame sur EcranPrincipal.

Le diagramme de la figure 5 représente la réception d'une trame, cependant, il est possible que *sniffer* reçoive beaucoup de trames en peu de temps. Dans ce cas, l'opération "notifyNewFrames()" est appelée périodiquement et *gui* récupère toutes les trames reçues depuis le dernier appel de "getFrames()". Ainsi, les opérations "notifyNewFrames()" et "getFrames()" sont au pluriel car il est possible de récupérer un paquet de trames. Pour finir, comme on peut le voir dans la figure 6, Utilisateur a la possibilité de mettre en pause la réception des trames, ce qui suspend temporairement la boucle.

2.2.5 *CU Interagir avec le sniffer - Scénario nominal*

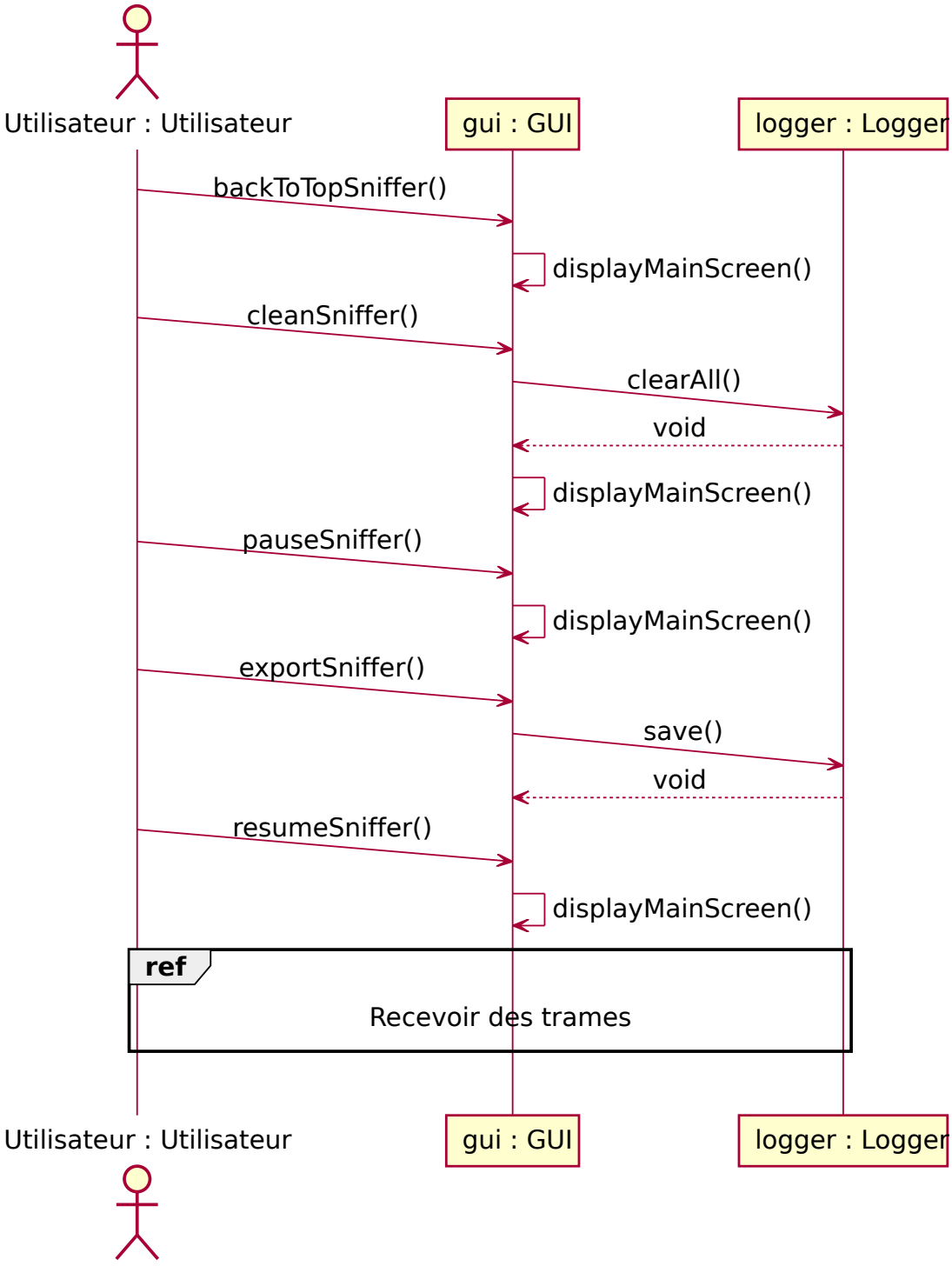

FIGURE 6 – Diagramme de séquence du *CU Interagir avec le sniffer - Scénario nominal*

Le diagramme de la figure 6 représente le diagramme de séquence du *CU Interagir avec le sniffer - Scénario nominal.*
Utilisateur peut interagir avec le sniffer et réaliser plusieurs actions :

- Revenir en haut du sniffer pour voir apparaître les nouvelles trames.
- Nettoyer le sniffer c'est-à-dire supprimer toutes les trames présentes dans le sniffer.
- Lancer ou mettre en pause la réception des trames.
- Exporter les trames dans un fichier log en dehors de l'application CANdroid. Cette action est uniquement possible si Utilisateur a mis en pause la réception des trames.

De plus, la figure 6 indique un ordre spécifique pour les actions à effectuer, à savoir l'ordre du CU "Interagir avec le sniffer" du dossier de spécification [dossier_de_specification_SPEC_B1_2024]. Cependant, dans la version comprenant les variantes, il est indiqué que les actions peuvent être effectuées dans l'ordre souhaité, à l'exception de l'exportation des trames, qui doit être réalisée après la mise en pause du sniffer. En d'autres termes, Utilisateur a la liberté d'effectuer les actions dans l'ordre qui lui convient, excepté pour l'exportation des trames qui nécessite une mise en pause du sniffer au préalable.

2.2.6 *CU Ajouter un objet - Scénario nominal*

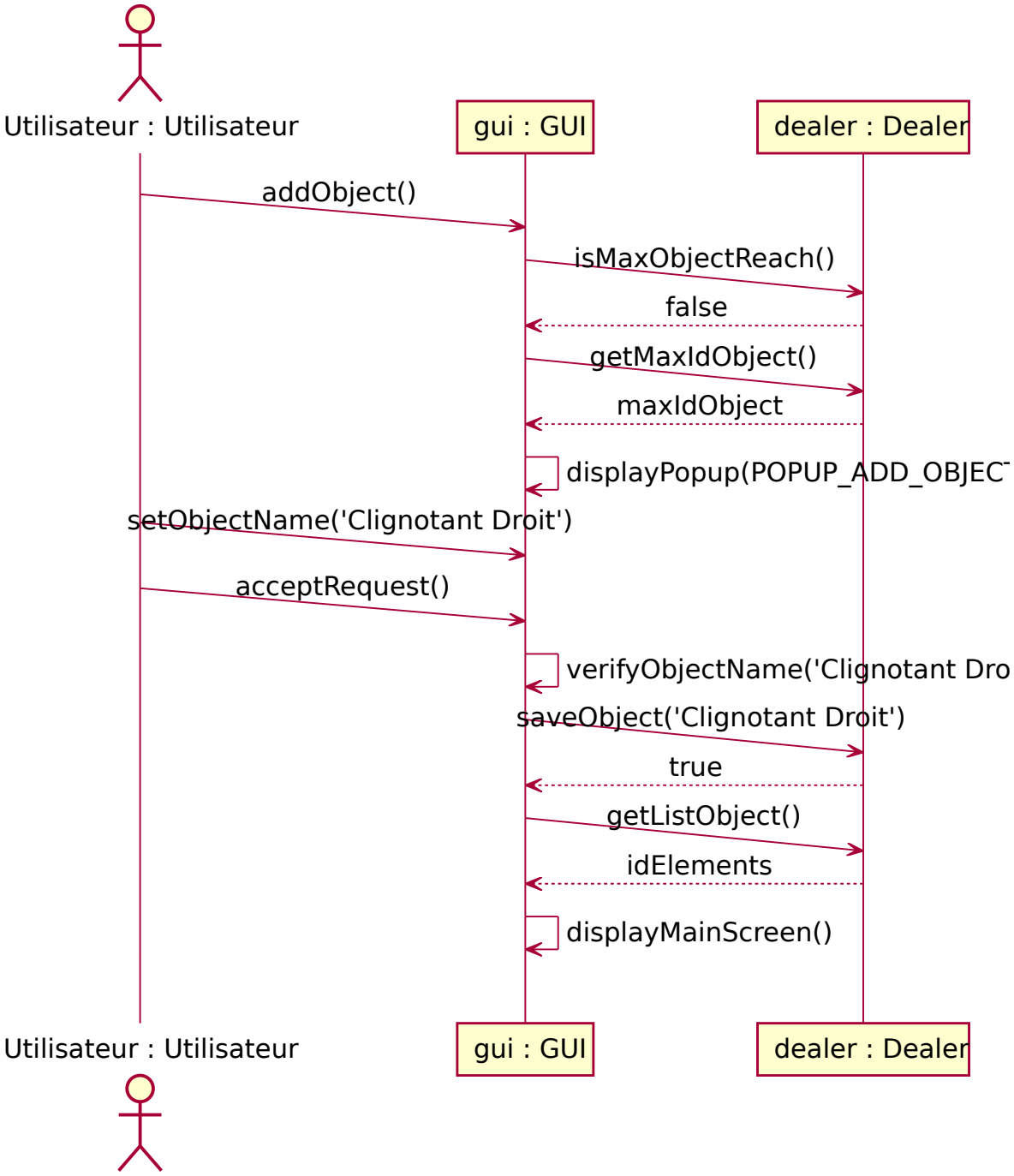

FIGURE 7 – Diagramme de séquence du *CU Ajouter un objet - Scénario nominal*

Le diagramme de la figure 7 représente le diagramme de séquence du *CU Ajouter un objet - Scénario nominal*.
Utilisateur demande l'ajout d'un nouvel objet sur l'application CANdroid. Si le nombre d'objets existants dans l'application CANdroid est inférieur à la limite maximale, PopupAjoutObjet s'affiche. Ce pop-up permet à Utilisateur de donner un nom à l'objet qu'il souhaite ajouter. Une fois le nom saisi, Utilisateur valide la demande d'ajout. Enfin, l'objet est ajouté aux objets déjà présents sur l'application CANdroid et EcranPrincipal est mis à jour en conséquence.
Dans le cas général, Utilisateur peut choisir de laisser le nom d'objet par défaut présent dans <champNomObjet> pré-rempli. Ce nom est obtenu grâce à l'appel de l'opération "getMaxIdObject()" qui permet de donner le numéro du nom de l'objet. Par exemple, si l'opération renvoie le nombre "4" alors le Nom d'objet par défaut est "Objet_4".
De plus, si Utilisateur a entré un nom déjà utilisé par un autre objet, l'application CANdroid affiche PopupErreurAjoutObjet. S'il essaie d'ajouter un objet alors que le nombre d'objet maximum est atteint, l'application CANdroid affiche PopupErreurNombreObjet. Ces cas de figure ne sont pas présentés sur ce scénario, mais sont explicitement décrits dans le CU "Ajouter un objet" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.7 *CU Ajouter une trame - Scénario nominal*

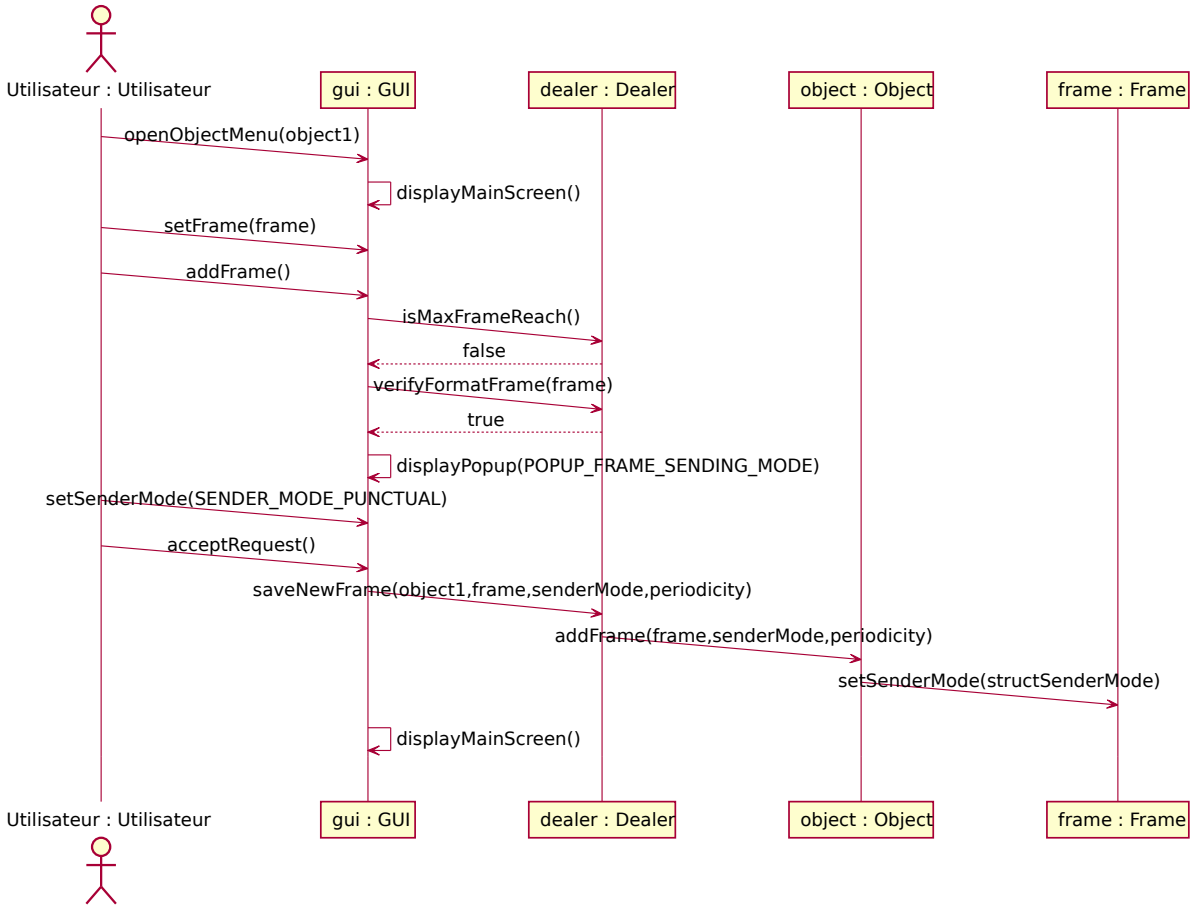

FIGURE 8 – Diagramme de séquence du *CU Ajouter une trame - Scénario nominal*

Le diagramme de la figure 8 représente le diagramme de séquence du *CU Ajouter une trame - Scénario nominal*.
Lorsque Utilisateur ouvre le menu de l'objet, EcranPrincipal se met à jour. Utilisateur entre une nouvelle trame dans le menu de l'objet associé et l'enregistre si le nombre de trames dans cet objet est inférieur à la limite maximale autorisée, pour l'enregistrement des trames. L'application CANdroid affiche PopupModeEnvoiTrame, permettant à Utilisateur de définir le Mode Envoi de la trame. Utilisateur définit et valide son choix, la trame s'enregistre et EcranPrincipal s'affiche.

Dans le cas général, la saisie de la trame peut être incorrecte, l'application CANdroid affiche donc PopupErreurSaisieTrame, qui indique le bon format des trames à enregistrer. Utilisateur peut également décider d'ajouter une trame alors que le nombre de trame maximum est atteint, dans ce cas, l'application CANdroid afficher PopupErreurNombreTrame. De plus, dans la figure 8, le Mode Envoi est défini par défaut en mode cyclique avec une périodicité par dé-

faut de 100 ms et Utilisateur le définit en mode ponctuel. Cependant, Utilisateur peut choisir de ne pas effectuer ce changement. Ces cas de figure ne sont pas présentés sur ce scénario, mais sont explicitement décrits dans le CU "Enregistrer une trame" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.8 *CU Envoyer des trames - Scénario nominal*

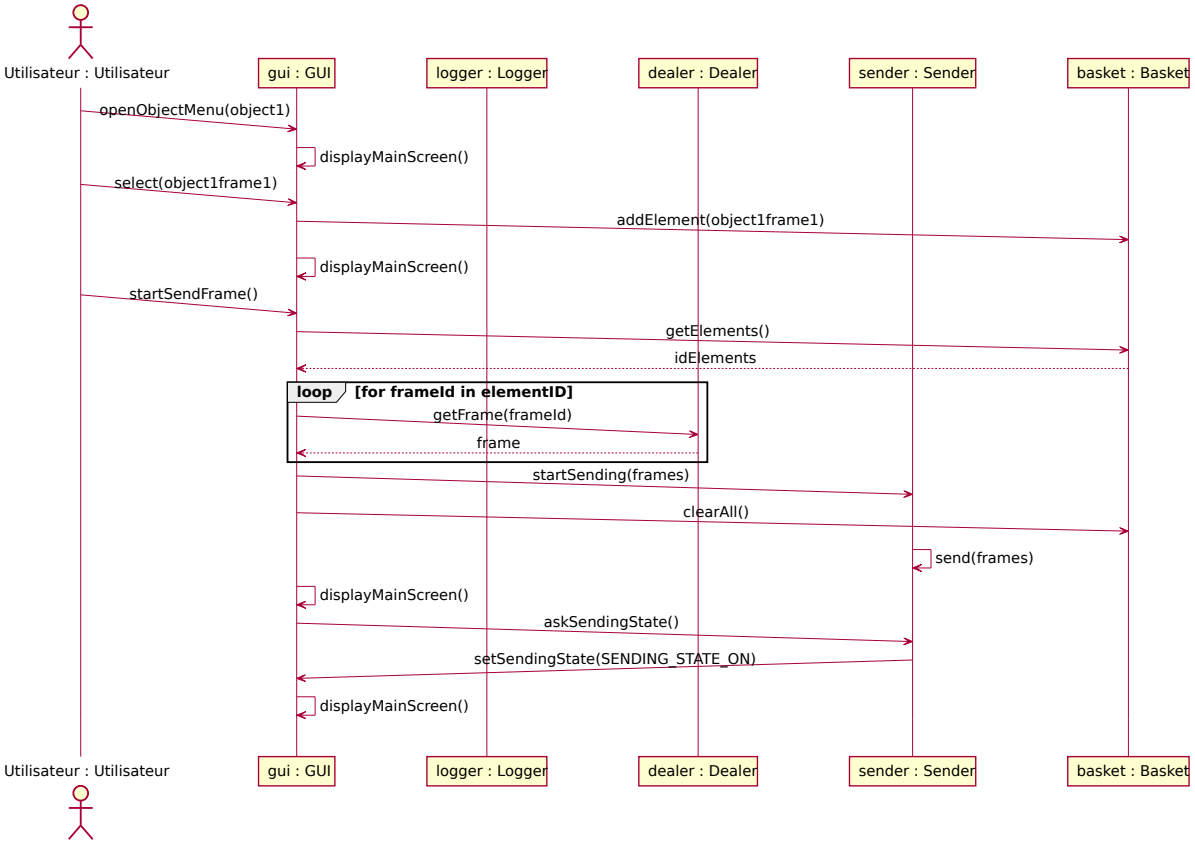

FIGURE 9 – Diagramme de séquence du *CU Envoyer des trames - Scénario nominal*

Le diagramme de la figure 9 représente le diagramme de séquence du *CU Envoyer des trames - Scénario nominal*.
Utilisateur ouvre le menu de l'objet et sélectionne la trame qu'il souhaite envoyer. L'application CANdroid affiche ensuite EcranPrincipal. Utilisateur commence l'envoi de la trame sélectionnée. Pour que l'envoi commence, *sender* a besoin des informations concernant la trame choisie. Ces informations lui sont transmises grâce à *gui* qui les récupère de *dealer*. Dès que l'application passe en Mode Envoi, EcranPrincipal est mis à jour en conséquence. Afin d'afficher la trame émise dans le sniffer de l'application, il est nécessaire de transmettre la trame à *logger*. Le diagramme de la figure 9 ne représente que l'envoi d'une trame mais il est possible d'envoyer plusieurs trames à la fois.
En réalité, les opérations "askSendingState()" et "setSendingState(sendingState : Sending-

State)" sont appelées périodiquements pour informer *gui* de l'état de l'application CANdroid.

Dans le cas général, si Utilisateur souhaite désélectionner un élément, il lui suffit de cliquer à nouveau sur ce dernier. Si un objet a été sélectionné, il n'est pas possible de désélectionner individuellement une trame qui fait partie de ce même objet.
Pour chaque trame sélectionnée, *gui* récupère les informations de la trame grâce à l'opération "getFrames()" de *dealer*.
De plus, si les trames sélectionnées ont des modes d'envoi périodiques, alors l'appel des opérations "send()", "setFrame(frame)", "notifyNewFrames()", "getFrames()" et "displayMainScreen()" se répète jusqu'à la demande d'arrêt.
Ces cas de figure ne sont pas présentés sur ce scénario, mais sont explicitement décrits dans le CU "Envoyer des trames" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.9 *CU Arrêter envoi des trames - Scénario nominal*

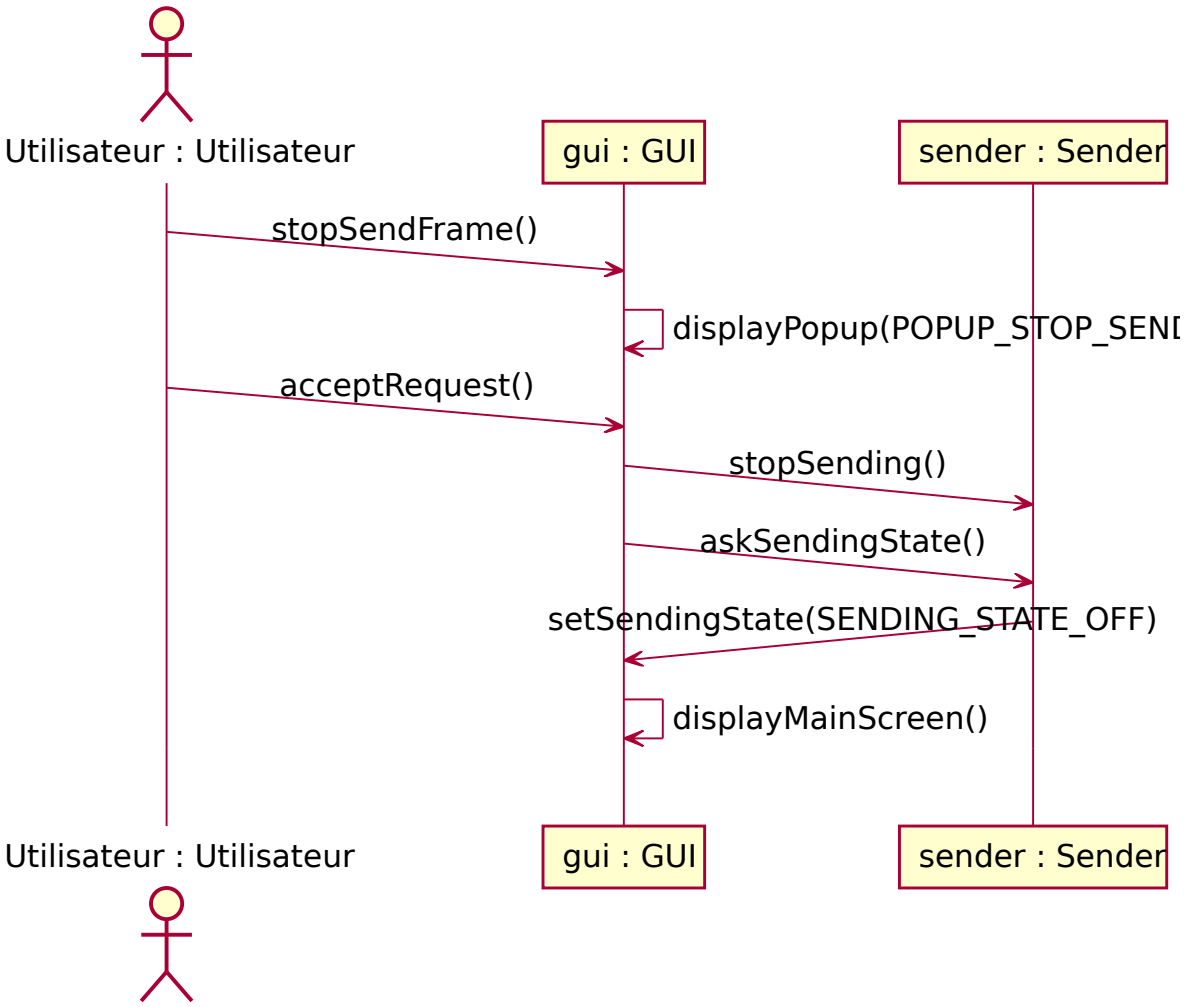

FIGURE 10 – Diagramme de séquence du *CU Arrêter envoi des trames - Scénario nominal*

Le diagramme de la figure 10 représente le diagramme de séquence du *CU Arrêter envoi des trames - Scénario nominal*.
Lorsque Utilisateur demande l'arrêt d'envoi des trames, l'application CANdroid affiche PopupArretEnvoi. Utilisateur valide la demande d'arrêt, l'application CANdroid stoppe l'envoi de trames et met à jour EcranPrincipal en conséquence.
Dans le cas général, si Utilisateur refuse la demande d'arrêt d'envoi des trames, l'application CANdroid ne s'arrête pas et continue à envoyer des trames. Ce cas de figure est explicitement décrit dans le CU "Arrêter l'envoi des trames" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.10 *CU Supprimer un élément - Scénario nominal*

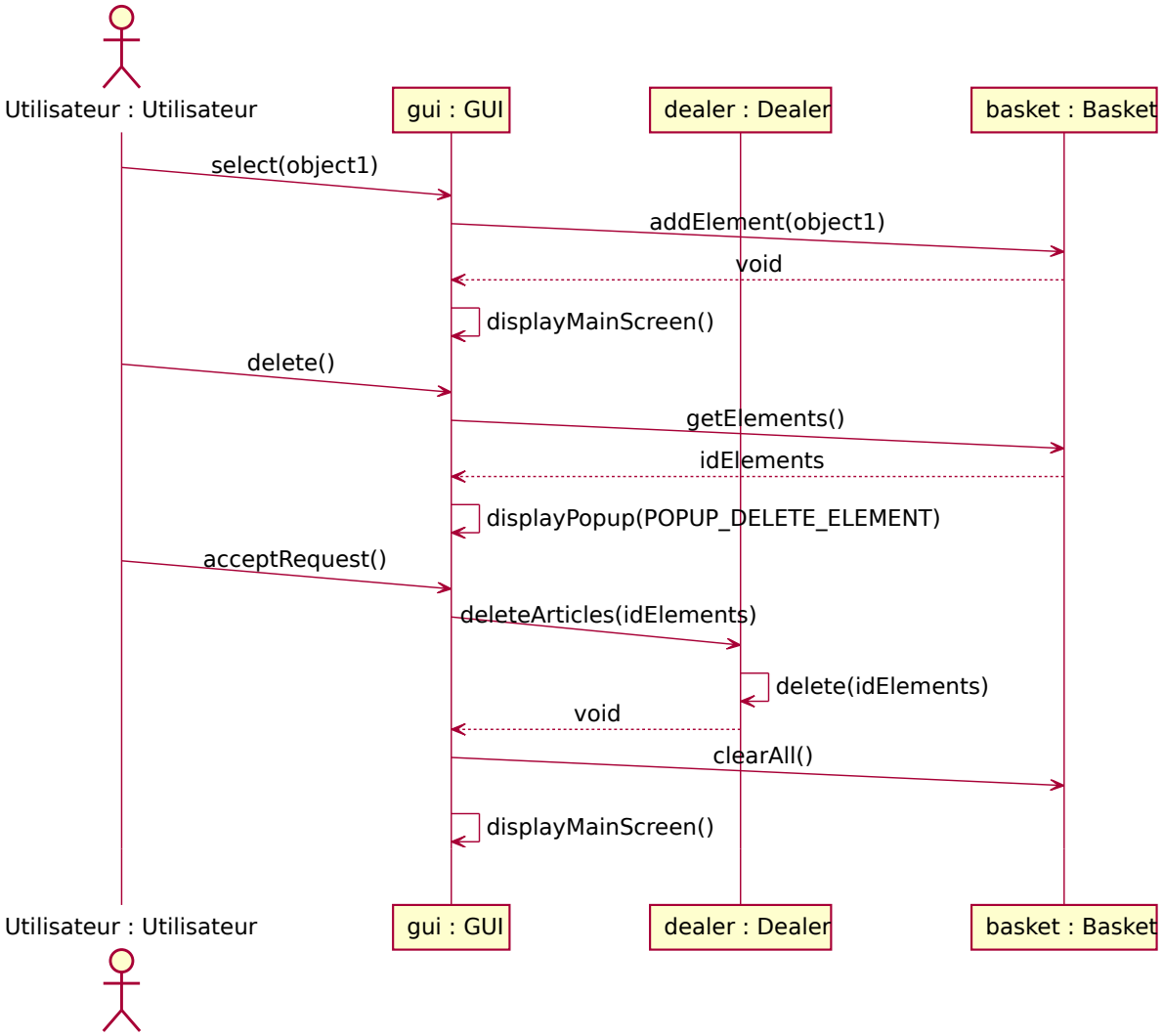

FIGURE 11 – Diagramme de séquence du *CU Supprimer un élément - Scénario nominal*

Le diagramme de la figure 11 représente le diagramme de séquence du *CU Supprimer un élément - Scénario nominal*.
Utilisateur sélectionne un objet et toutes les trames qui lui sont associées sont automatiquement sélectionnées en conséquence. Par la suite, L'application CANdroid se met à jour. Après avoir sélectionné les éléments à supprimer, l'application CANdroid affiche PopupSuppressionElement et Utilisateur valide la suppression. Une fois la suppression confirmée, l'application CANdroid met à jour EcranPrincipal pour refléter les changements.

Dans le cas général, Utilisateur a la possibilité de sélectionner une trame sans nécessairement sélectionner l'objet qui la contient, et de la supprimer. De plus, il est possible de sélection-

ner plusieurs éléments simultanément pour les supprimer, et l'application CANdroid met à jour EcranPrincipal en conséquence. Ces cas de figure ne sont pas présentés sur ce scénario, mais sont explicitement décrits dans le CU "Supprimer un élément" du dossier de spécification [dossier_de_specification_SPEC_B1_2024].

2.2.11 *CU Stopper le SàE - Scénario nominal*

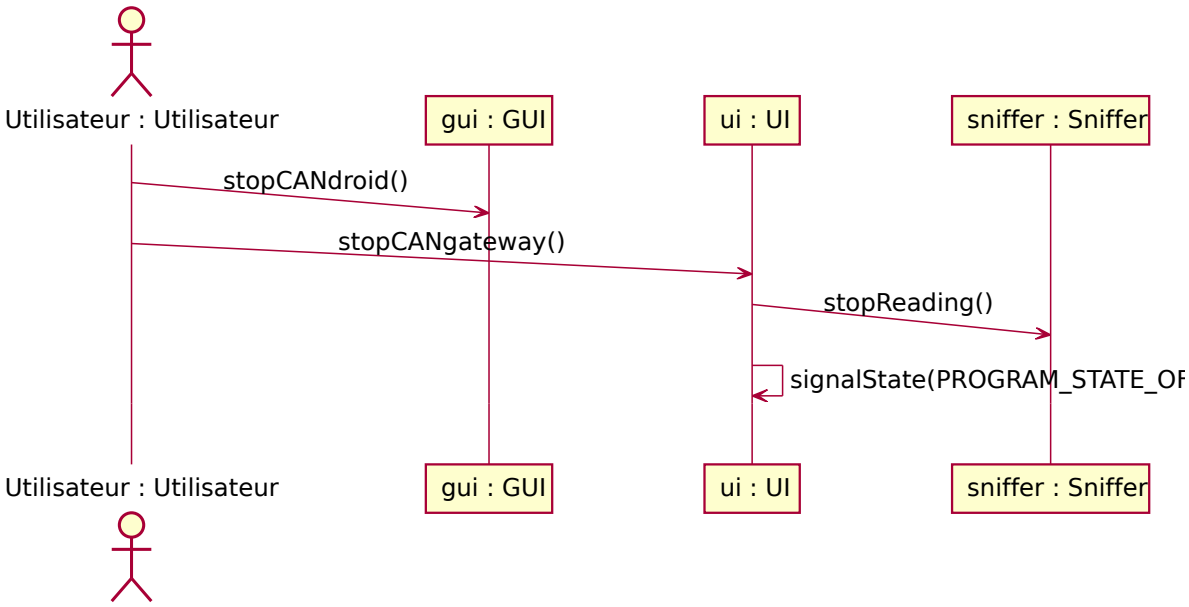

FIGURE 12 – Diagramme de séquence du *CU Stopper le SàE - Scénario nominal*

Le diagramme de la figure 12 représente le diagramme de séquence du *CU Stopper le SàE - Scénario nominal*.
Pour arrêter le SàE, Utilisateur commence par arrêter l'application CANdroid grâce à l'opération "stopCANdroid()". Il arrête ensuite le programme CANgateway avec l'opération "stopCANgateway()". Cette opération se charge d'appeler l'opération "stopReading()" qui arrête la lecture en continue des trames CAN. Pour finir, le programme CANgateway informe Utilisateur que le SàE est arrêté grâce à l'opération "signalState(programmeState : ProgrammeState).

Dans le cas général, Utilisateur peut décider d'utiliser seulement le Smartphone avec l'application CANdroid, il n'a donc pas besoin d'arrêter le programme CANgateway.

2.3 Types de données

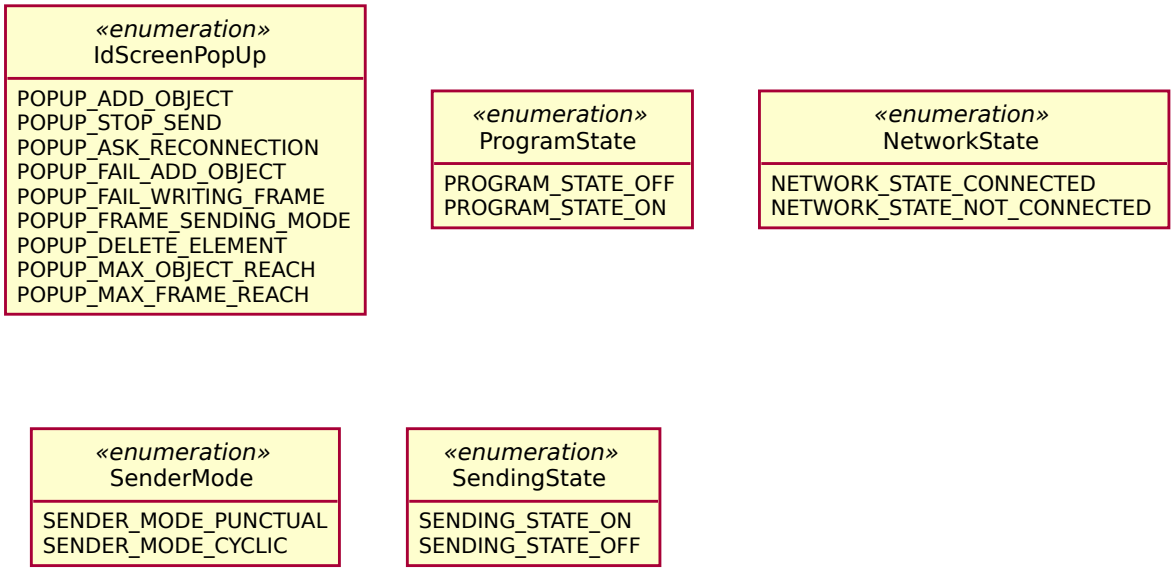

FIGURE 13 – Diagramme des types de données

Le diagramme de la figure 13 représente les types de données utilisés.

2.3.1 Description de l'énumération IdScreenPopUp

Cette énumération représente les différents types de pop-ups utilisés dans le système. Chaque valeur de l'énumération correspond à un type de pop-up spécifique, tel que, par exemple, l'ajout d'objet, l'arrêt d'envoi ou la demande de reconnexion.

En voici le détail des énumérations :

- **POPUP_ADD_OBJECT** — Correspond à PopupAjoutObjet.
- **POPUP_STOP_SEND** — Correspond à PopupArretEnvoi.
- **POPUP_ASK_RECONNECTION** — Correspond à PopupDemandeReconnexion.
- **POPUP_FAIL_ADD_OBJECT** — Correspond à PopupErreurAjoutObjet.
- **POPUP_FAIL_WRITING_FRAME** — Correspond à PopupErreurSaisieTrame.
- **POPUP_FRAME_SENDING_MODE** — Correspond à PopupModeEnvoiTrame.
- **POPUP_DELETE_ELEMENT** — Correspond à PopupSuppressionElement.
- **POPUP_MAX_OBJECT_REACH** — Correspond à PopupErreurNombreObjet.
- **POPUP_MAX_FRAME_REACH** — Correspond à PopupErreurNombreTrame.

2.3.2 Description de l'énumération ProgramState

Cette énumération représente l'état du programme, qui peut être soit "OFF" lorsque le programme est arrêté, soit "ON" lorsque le programme est en cours d'exécution.

En voici le détail des énumérations :

— **PROGRAM_STATE_OFF** — Correspond à l'état éteint du programme CANgateway.

— **PROGRAM_STATE_ON** — Correspond à l'état allumé du programme CANgateway.

2.3.3 Description de l'énumération NetworkState

Cette énumération représente l'état de la connexion réseau, qui peut être soit "CONNECTED" lorsque la connexion est établie, soit "NOT_CONNECTED" lorsque la connexion est perdue.

En voici le détail des énumérations :

— **NETWORK_STATE_CONNECTED** — Correspond à l'état connecté de l'application CANdroid au programme CANgateway.

— **NETWORK_STATE_NOT_CONNECTED** — Correspond à l'état non connecté de l'application CANdroid au programme CANgateway.

2.3.4 Description de l'énumération SenderMode

Cette énumération représente le Mode Envoi des données, qui peut être soit "PUNCTUAL" pour un envoi ponctuel, soit "CYCLIC" pour un envoi cyclique.

En voici le détail des énumérations :

— **SENDER_MODE_PUNCTUAL** — Correspond au Mode Envoi de trames ponctuel.

— **SENDER_MODE_CYCLIC** — Correspond au Mode Envoi de trames cyclique.

2.3.5 Description de l'énumération SendingState

Cette énumération représente l'état d'envoi des données, qui peut être soit "ON" lorsque l'envoi est activé, soit "OFF" lorsque l'envoi est désactivé.

En voici le détail des énumérations :

— **SENDING_STATE_ON** — Correspond à l'état ON du Mode d'Envoi. Cela signifie que les trames sont en cours d'envoi.

— **SENDING_STATE_OFF** — Correspond à l'état OFF du Mode d'Envoi. Cela signifie que les trames ne sont pas en cours d'envoi.

2.3.6 Descriptions des autres types de données

Voici une liste des autres types de données utilisés dans le dossier ainsi que leur description :

- **bool** : booléen, peut prendre la valeur true ou false.
- **CANFrame** : structure de données représentant un trame CAN. Elle contient les champs suivants :
  - **id** : identifiant de la trame. C'est un entier non signé de 32 bits.
  - **size** : taille de la trame. C'est un entier non signé de 8 bits.
  - **data** : tableau de 8 octets non signés contenant les données de la trame.
- **IdElement** : identifiant des objets et des trames. C'est un entier non signé de 16 bits. L'ID d'un objet est de la forme XX000 où XX représente l'ID de l'objet. L'ID d'une trame est de la forme XXYYY où XX représente l'ID de l'objet et YYY représente l'ID de la trame. Par exemple, si l'objet "Clignotant droit" a pour ID 06000, alors la première trame associée à cet objet aura pour ID 06001.
- **String** : chaîne de caractères.
- **StructSenderMode** : structure de données contenant le Mode Envoi (enumération SenderMode) et la périodicité si le Mode Envoi est cyclique.

2.4 Classes

2.4.1 Vue générale

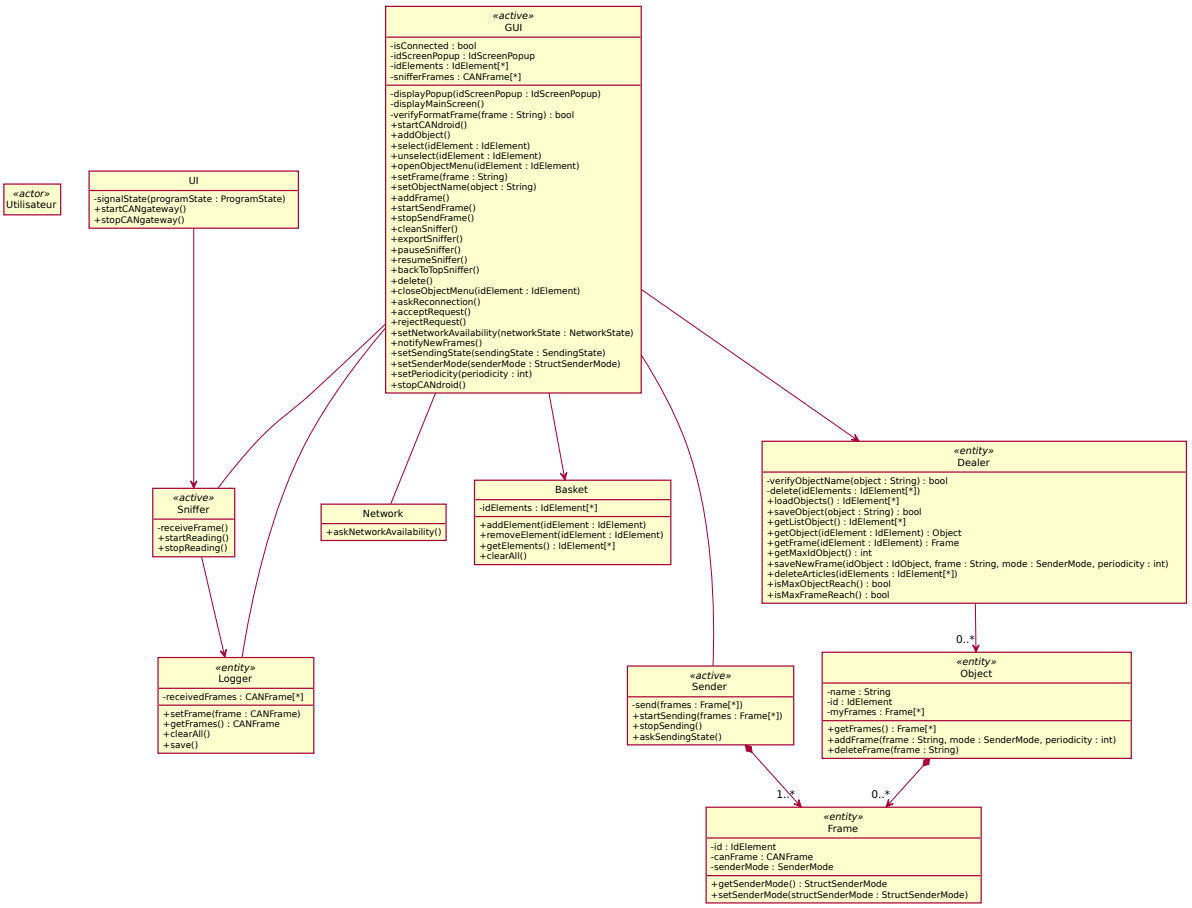

FIGURE 14 – Diagramme de classes

Le diagramme de la figure 14 représente les classes du système.

2.4.2 La classe Utilisateur

Le diagramme de la figure 15 représente la classe Utilisateur.

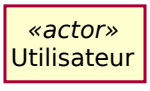

FIGURE 15 – Diagramme de la classe Utilisateur

2.4.2.1 Attributs

N.A.

2.4.2.2 Services offerts

N.A.

2.4.3 La classe GUI

Le diagramme de la figure 16 représente la classe GUI.

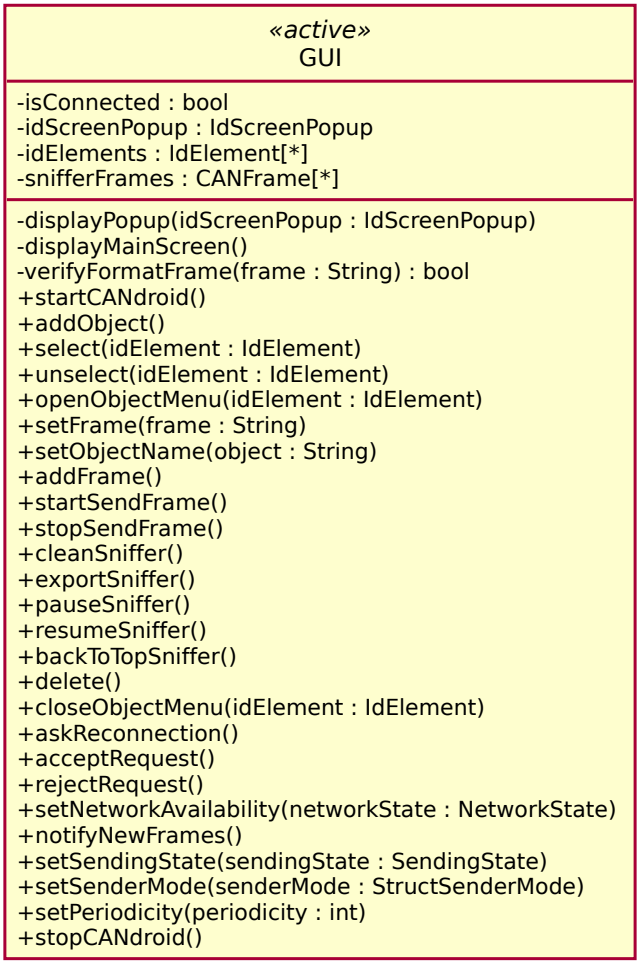

FIGURE 16 – Diagramme de la classe GUI

2.4.3.1 Philosophie de conception

La classe GUI permet de gérer les interfaces utilisateur de l'application CANdroid.

2.4.3.2 Attributs

- **isConnected : bool** — Indique si l'application CANdroid est connectée au programme CANgateway.
- **idScreenPopup : IdScreenPopup** — Identifiant du pop-up actuellement affiché sur E_Smartphone.
- **idElements : IdElement[*]** — Liste des identifiants des objets et des trames actuellement affichés sur EcranPrincipal.
- **snifferFrames : CANFrame[*]** — Liste des trames actuellement affichées dans la partie sniffer de EcranPrincipal.

2.4.3.3 Services offerts

- **displayPopup(idScreenPopup : IdScreenPopup)** — Permet d'afficher le pop-up d'identifiant idScreenPopup sur E_Smartphone.
- **displayMainScreen()** — Permet d'afficher EcranPrincipal sur E_Smartphone.
- **verifyFormatFrame(frame : String) : bool** — Permet de vérifier le format de la trame saisie par Utilisateur. Le paramètre frame correspond à la chaîne de caractère saisie par Utilisateur dans le champ de texte <champTrame> sur EcranPrincipal. Retourne true si le format de la trame est correct, false sinon.
- **startCANdroid()** — Démarre l'application CANdroid.
- **addObject()** — Opération appelée lorsque Utilisateur clique sur le bouton [ajouter]. Permet d'ajouter un objet à l'application CANdroid. Déclenche l'affichage de PopupAjoutObjet.
- **select(idElement : IdElement)** — Opération appelée lorsque Utilisateur clique sur un objet ou une trame non sélectionné. Permet de sélectionner l'objet ou la trame cliqué. Le paramètre idElement correspond à l'identifiant de l'objet ou de la trame sélectionné.
- **unselect(idElement : IdElement)** — Opération appelée lorsque Utilisateur clique sur un objet ou une trame déjà sélectionné. Permet de désélectionner l'objet ou la trame cliqué. Le paramètre idElement correspond à l'identifiant de l'objet ou de la trame sélectionné.
- **openObjectMenu(idElement : IdElement)** — Opération appelée lorsque Utilisateur clique sur le bouton [deplierMenuObjet]. Permet de déplier le menu de l'objet. Le paramètre idElement correspond à l'identifiant de l'objet choisi. Change le bouton [deplierMenuObjet] cliqué en [replierMenuObjet].
- **setFrame(frame : String)** — Permet d'écrire une trame associée à un objet. Le paramètre frame correspond à la chaîne de caractère saisie par Utilisateur dans le champ de texte <champTrame> sur EcranPrincipal.
- **setObjectName(object : String)** — Permet d'écrire le nom de l'objet créé par Utilisateur. Le paramètre object correspond à la chaîne de caractère saisie par Utilisateur dans le champ de texte <champNomObjet> sur PopupAjoutObjet.

— **addFrame()** — Opération appelée lorsque Utilisateur clique sur le bouton [ajouterTrame]. Permet d'ajouter la trame saisie à la liste des trames associées à l'objet. Déclenche l'affichage de PopupModeEnvoiTrame si le format de la trame est correct et PopupErreurSaisieTrame si le format de la trame est incorrect.

— **startSendFrame()** — Opération appelée lorsque Utilisateur clique sur le bouton [lancerEnvoi]. Permet de débuter l'envoi des trames précédemment sélectionnées. Change le bouton [lancerEnvoi] en [arreterEnvoi].

— **stopSendFrame()** — Opération appelée lorsque Utilisateur clique sur le bouton [arreterEnvoi]. Permet d'arrêter l'envoi des trames. Change le bouton [arreterEnvoi] en [lancerEnvoi]. Déclenche l'affichage de PopupArretEnvoi.

— **cleanSniffer()** — Opération appelée lorsque Utilisateur clique sur le bouton [viderSniffer]. Permet de supprimer toutes les trames actuellement affichées dans la partie sniffer de EcranPrincipal.

— **exportSniffer()** — Opération appelée lorsque Utilisateur clique sur le bouton [exporterSniffer]. Permet de sauvegarder l'ensemble des trames affichées dans la partie sniffer de EcranPrincipal, dans un fichier présent sur E_Smartphone.

— **pauseSniffer()** — Opération appelée lorsque Utilisateur clique sur le bouton [pause]. Stoppe l'affichage de nouvelles trames dans le sniffer. Change le bouton [pause] en [play].

— **resumeSniffer()** — Opération appelée lorsque Utilisateur clique sur le bouton [play]. Reprend l'affichage de nouvelles trames dans le sniffer. Change le bouton [play] en [pause].

— **backToTopSniffer()** — Opération appelée lorsque Utilisateur clique sur le bouton [revenirEnHaut]. Permet de revenir en haut de l'affichage de la partie sniffer de EcranPrincipal.

— **delete()** — Opération appelée lorsque Utilisateur clique sur le bouton [suppressionElement]. Permet de supprimer les objets ou les trames précédemment sélectionnés par Utilisateur et stockés dans Basket. Déclenche l'affichage de PopupSuppressionElement.

— **closeObjectMenu(idElement : IdElement)** — Opération appelée lorsque Utilisateur clique sur le bouton [replierMenuObjet]. Permet de replier le menu de l'objet. Le paramètre idElement correspond à l'identifiant de l'objet choisi. Change le bouton [replierMenuObjet] cliqué en [deplierMenuObjet].

— **askReconnection()** — Opération appelée lorsque Utilisateur clique sur le bouton [connexion]. Permet de demander la reconnexion entre le programme CANgateway et l'application CANdroid. Déclenche l'affichage de PopupDemandeReconnexion.

— **acceptRequest()** — Opération appelée lorsque Utilisateur clique sur le bouton [validerXXXXX] (XXXXX étant le nom du pop-up) du pop-up actuellement affiché sur E_Smartphone. L'opération appelée ensuite dépend du pop-up actuellement affiché sur E_Smartphone.

— **rejectRequest()** — Opération appelée lorsque Utilisateur clique sur le bouton [annulerXXXXX] (XXXXX étant le nom du pop-up) du pop-up actuellement affiché sur E_Smartphone. E_Smartphone affiche EcranPrincipal.

— **setNetworkAvailability(networkState : NetworkState)** — Permet de mettre à jour l'état de connexion entre le programme CANgateway et l'application CANdroid. L'état courant de connexion correspond à networkState.

— **notifyNewFrames()** — Informe GUI qu'une nouvelle trame a été reçue par Sniffer et qu'il peut la récupérer pour pouvoir l'afficher dans la partie sniffer de EcranPrincipal.

— **setSendingState(sendingState : SendingState)** — Permet de mettre à jour l'état d'envoi de GUI.

— **setSenderMode(senderMode : StructSenderMode)** — Opération appelée lorsque Utilisateur clique sur les radio-boutons ([radioBoutonCyclique] ou [radioBoutonPonctuel]) dans PopupModeEnvoiTrame.

— **setPeriodicity(periodicity : int)** — Permet d'écrire la périodicité d'envoi de la trame que Utilisateur a saisi dans le champ de texte <champPeriodicite> sur PopupModeEnvoiTrame. Le paramètre periodicity correspond à la périodicité en ms.

— **stopCANdroid()** — Arrête l'application CANdroid.

2.4.3.4 Description comportementale

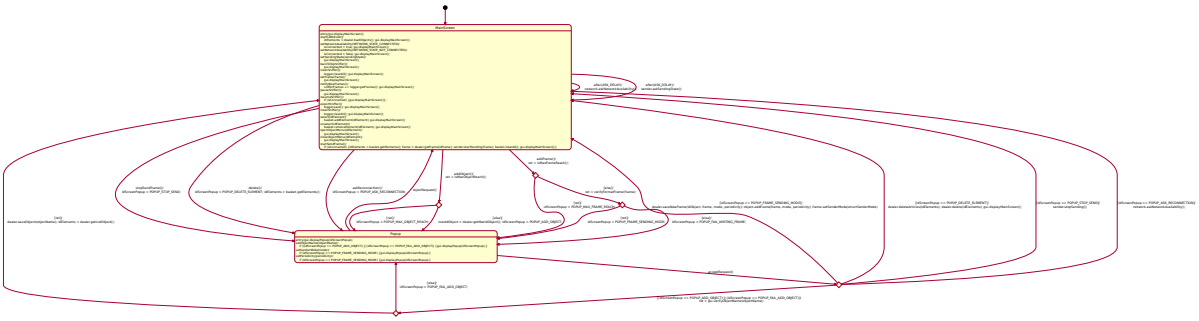

FIGURE 17 – Machine à états de *GUI*

Le diagramme de la figure 17 représente la machine à états de *GUI*.
Lorsque l'application CANdroid démarre, l'état par défaut de la MAE est l'état MainScreen. Entrer dans cet état appelle systématiquement l'opération pour afficher EcranPrincipal. Un certain nombre d'opérations peuvent être appelées dans cet état. Elles sont internes à l'état.

L'application CANdroid entre dans l'état Popup lorsque l'opération associée à l'apparition d'un pop-up est appelée. Entrer dans cet état appelle l'opération pour afficher le pop-up correspondant. L'appel de l'opération "rejectRequest()" permet de ramener l'application CANdroid à l'état MainScreen. L'appel de l'opération "acceptRequest()" permet de ramener l'application CANdroid à l'état MainScreen et de lancer l'opération correspondante à l'action de Utilisateur selon le pop-up qui était précédemment affiché sur Smartphone.

2.4.4 La classe UI

Le diagramme de la figure 18 représente la classe UI.

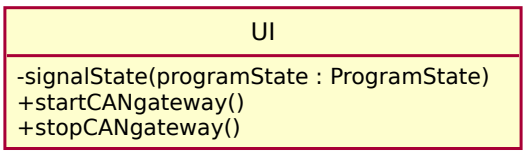

FIGURE 18 – Diagramme de la classe UI

2.4.4.1 Philosophie de conception

La classe UI permet de gérer les interactions de Utilisateur avec le programme CANgateway.

2.4.4.2 Attributs

N.A.

2.4.4.3 Services offerts

- **signalState(programState : ProgramState)** — Informe Utilisateur de l'état de lancement du programme CANgateway.
- **startCANgateway()** — Démarre le programme CANgateway.
- **stopCANgateway()** — Arrête le programme CANgateway.

2.4.5 La classe Dealer

Le diagramme de la figure 19 représente la classe Dealer.

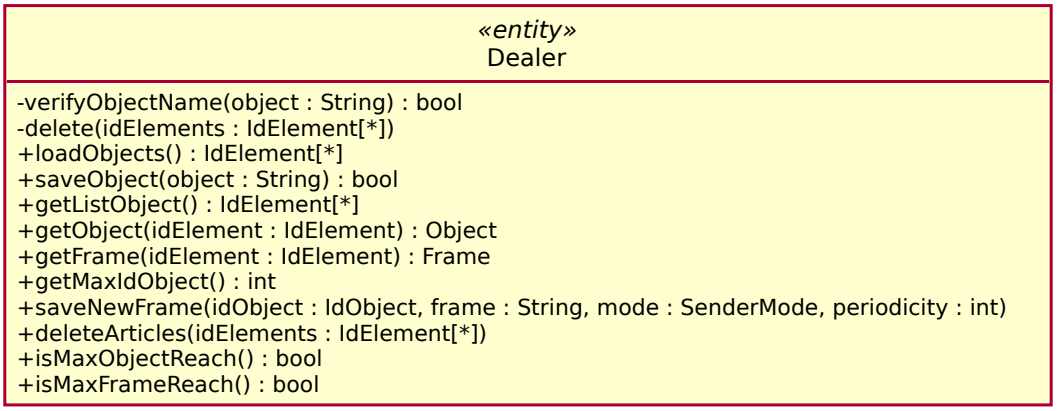

FIGURE 19 – Diagramme de la classe Dealer

2.4.5.1 Philosophie de conception

La classe Dealer permet de gérer les objets et trames enregistrés sur l'application CANdroid.

2.4.5.2 Attributs

N.A.

2.4.5.3 Services offerts

- **verifyObjectName(object : String) : bool** — Vérifie que le nom de l'objet que Utilisateur souhaite ajouter n'existe pas déjà. Le paramètre object correspond à la chaîne de caractère saisie par Utilisateur dans le champ de texte <champNomObjet> sur PopupAjoutObjet. Retourne true si le nom de l'objet est correct (i.e. n'existe pas encore), false sinon.
- **delete(idElements : IdElement[*])** — Permet de supprimer les éléments précédemment sélectionnés.
- **loadObjects() : IdElement**[*] — Permet à GUI de récupérer les éléments (objets et trames) créés lors d'un précédent démarrage de l'application CANdroid. Retourne la liste de ces éléments.
- **saveObject(object : String) : bool** — Permet d'enregistrer un nouvel objet. Le paramètre object correspond au nom de l'objet.
- **getListObject() : IdElement**[*] — Renvoie la liste des identifiants des objets sauvegardés.
- **getObject(idElement : IdElement) : Object** — Renvoie l'objet possédant l'identifiant passé en paramètre.
- **getFrame(idElement : IdElement) : Frame** — Renvoie la trame possédant l'identifiant passé en paramètre.
- **getMaxIdObject() : int** — Renvoie l'identifiant du prochain objet créé (i.e. l'identifiant le plus grand déjà utilisé + 1).
- **saveNewFrame(idObject : IdObject, frame : String, mode : SenderMode, periodicity : int)** — Permet d'enregistrer une nouvelle trame. Le paramètre frame correspond à la trame sous le bon format. Le paramètre idElement correspond à l'identifiant de l'objet auquel la trame sera liée.
- **deleteArticles(idElements : IdElement[*])** — Permet de supprimer les éléments précédemment sélectionnés et stockés dans Basket.
- **isMaxObjectReach() : bool** — Permet de vérifier si le nombre d'objets créés par Utilisateur est inférieur au nombre d'objet max. Retourne true si le nombre d'objets créés est inférieur au nombre d'objet max, false sinon.
- **isMaxFrameReach() : bool** — Permet de vérifier si le nombre de trames créées par Utilisateur est inférieur au nombre de trame max. Retourne true si le nombre de trames créées est inférieur au nombre de trame max, false sinon.

2.4.6 La classe Logger

Le diagramme de la figure 20 représente la classe Logger.

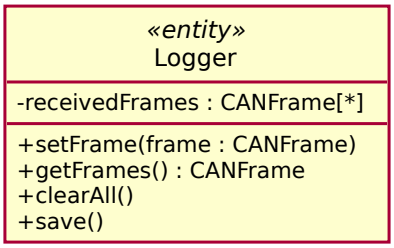

FIGURE 20 – Diagramme de la classe Logger

2.4.6.1 Philosophie de conception

La classe Logger permet de mémoriser les trames reçues par Sniffer et de les enregistrer dans un fichier.

2.4.6.2 Attributs

— **receivedFrames : CANFrame[*]** — Ensemble des trames reçues sur le bus CAN et affichées dans la partie sniffer de l'application CANdroid.

2.4.6.3 Services offerts

— **setFrame(frame : CANFrame)** — Ajoute la trame frame à receivedFrames.

— **getFrames() : CANFrame** — Renvoie la trame la plus ancienne de receivedFrames et la retire de la liste.

— **clearAll()** — Vide le fichier framesFile

— **save()** — Exporte le fichier framesFile.

2.4.7 La classe Object

Le diagramme de la figure 21 représente la classe Object.

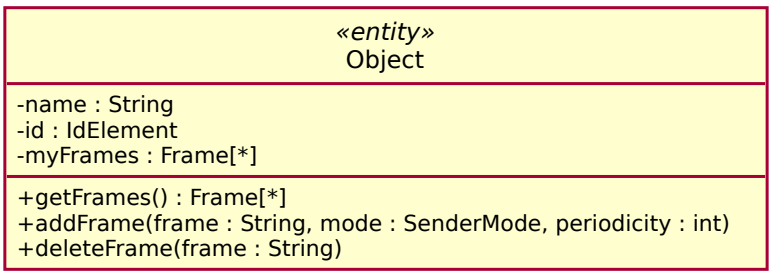

FIGURE 21 – Diagramme de la classe Object

2.4.7.1 Philosophie de conception

La classe Object représente une instance d'un objet *object*. Il est caractérisé par son nom, son ID et les trames qui lui sont associées. Par cette classe, on peut récupérer les trames qui lui sont associées, en ajouter de nouvelles ou en supprimer.

2.4.7.2 Attributs

- **name : String** — Nom de l'objet.
- **id : IdElement** — Id de l'objet.
- **myFrames : Frame[*]** — Trames associées à l'objet.

2.4.7.3 Services offerts

- **getFrames() : Frame[*]** — Renvoie la liste des trames associées à l'objet.
- **addFrame(frame : String, mode : SenderMode, periodicity : int)** — Convertit le paramètre frame en Frame et ajoute la trame à myFrames.
- **deleteFrame(frame : String)** — Convertit le paramètre frame en Frame et retire la trame de myFrames si elle existe. Sinon ne fait rien.

2.4.8 La classe Frame

Le diagramme de la figure 22 représente la classe Frame.

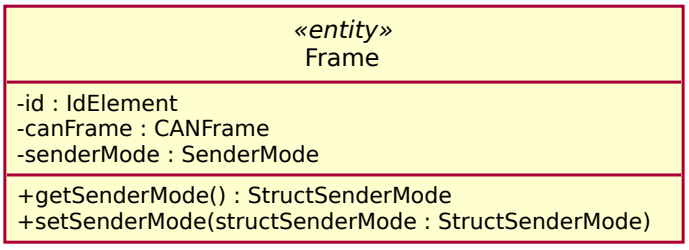

FIGURE 22 – Diagramme de la classe Frame

2.4.8.1 Philosophie de conception

La classe Frame représente une instance de l'objet *frame*. Elle est caractérisée par sa trame (de type CANFrame) et son Mode Envoi. Avec cette classe, on peut récupérer le Mode Envoi de la trame mais aussi le modifier.

2.4.8.2 Attributs

— **id : IdElement** — Id de la trame.

— **canFrame : CANFrame** — Trame CAN (avec identifiant, taille et données).

— **senderMode : SenderMode** — Mode Envoi de la trame.

2.4.8.3 Services offerts

— **getSenderMode() : StructSenderMode** — Renvoie la structure contenant le Mode Envoi de la trame et sa périodicité.

— **setSenderMode(structSenderMode : StructSenderMode)** — Met à jour le Mode Envoi de la trame. Le paramètre correspond la structure contenant le Mode Envoi de la trame et la périodicité choisi par Utilisateur.

2.4.9 La classe Sniffer

Le diagramme de la figure 23 représente la classe Sniffer.

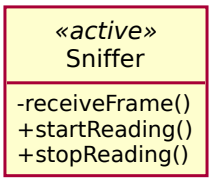

FIGURE 23 – Diagramme de la classe Sniffer

2.4.9.1 Philosophie de conception

La classe Sniffer gère la lecture des trames sur le bus CAN et l'envoi des trames lues vers l'application CANdroid.

2.4.9.2 Attributs

N.A.

2.4.9.3 Services offerts

— **receiveFrame()** — Reçoit la trame sniffée sur le bus CAN et la transmet à Logger.
— **startReading()** — Commence l'écoute des trames sur le bus CAN.
— **stopReading()** — Arrête l'écoute des trames sur le bus CAN.

2.4.9.4 Description comportementale

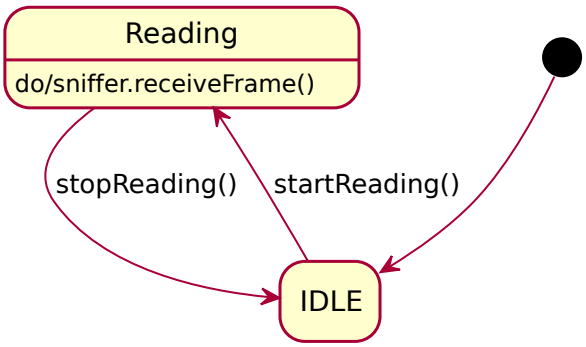

FIGURE 24 – Machine à états de *Sniffer*

Le diagramme de la figure 24 représente la machine à états de *Sniffer*.
Lors du lancement de l'application CANdroid, la machine à états est dans l'état IDLE. Dès que l'ensemble des connexions entre le Smartphone, la Raspberry PI et Tableau de Bord sont faites, la machine à états passe à l'état Listening. Dans cet état, Sniffer reçoit les trames en continu, comme le montre la fonction "receiveFrame()".
Lorsque Utilisateur souhaite mettre en pause l'arrivée des trames, la machine repasse dans l'état IDLE via l'opération "stopListening()". Il peut ainsi retourner en écoute lors de la reprise du sniffer via l'opération "startListening()".

2.4.10 La classe Network

Le diagramme de la figure 25 représente la classe Network.

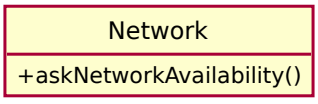

FIGURE 25 – Diagramme de la classe Network

2.4.10.1 Philosophie de conception

La classe Network permet d'interagir avec GUI et de déterminer si la connexion entre le smartphone et la Rapsberry Pi est établie.

2.4.10.2 Attributs

N.A.

2.4.10.3 Services offerts

— **askNetworkAvailability()** — Demande l'état du réseau TCP/IP.

2.4.11 La classe Basket

Le diagramme de la figure 26 représente la classe Basket.

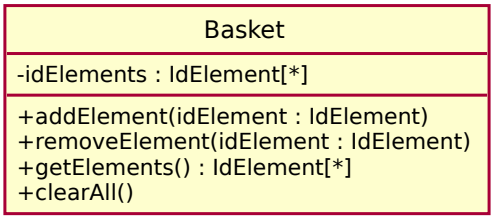

FIGURE 26 – Diagramme de la classe Basket

2.4.11.1 Philosophie de conception

La classe Basket permet d'enregistrer les éléments sélectionnés par Utilisateur.

2.4.11.2 Attributs

— **idElements : IdElement[*]** — Contenu du panier stocké sous forme de liste des identifiants des éléments dans le panier.

2.4.11.3 Services offerts

— **addElement(idElement : IdElement)** — Ajoute l'élement d'identifiant idElement au panier.

— **removeElement(idElement : IdElement)** — Retire l'élement d'identifiant idElement du panier.

— **getElements() : IdElement[*]** — Retourne un liste contenant l'ensemble des identifiants des éléments du panier.

— **clearAll()** — Vide le panier.

2.4.12 La classe Sender

Le diagramme de la figure 27 représente la classe Sender.

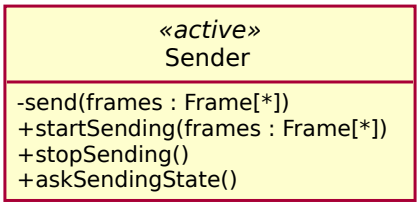

FIGURE 27 – Diagramme de la classe Sender

2.4.12.1 Philosophie de conception

La classe Sender permet d'envoyer les trames sur le bus CAN.

2.4.12.2 Attributs

N.A.

2.4.12.3 Services offerts

- **send(frames : Frame[*])** — Envoie les trames sur le bus CAN. Si les trames à envoyer ont des modes d'envoi ponctuel, les trames sont envoyées une seule fois. Si les trames à envoyer ont des modes d'envoi cyclique, les trames sont envoyées à intervalles réguliers selon leur périodicité.
- **startSending(frames : Frame[*])** — Démarre le cycle d'envoi de l'application CANdroid. Le paramètre frame correspond à la liste des trames à envoyer.
- **stopSending()** — Arrête le cycle d'envoi de l'application CANdroid.
- **askSendingState()** — Demande l'état de l'envoi.

2.4.12.4 Description comportementale

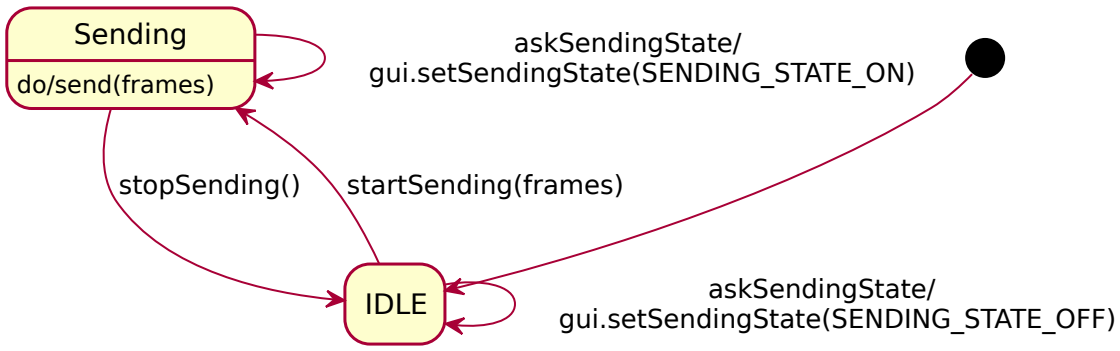

FIGURE 28 – Machine à états de *Sender*

Le diagramme de la figure 28 représente la machine à états de *Sender*.
Lorsque l'application CANdroid est en fonctionnement, la machine à états est dans l'état IDLE. L'opération "setSendingState()" renvoi la valeur de l'énumération correspondant à l'état d'envoi en cours. Ici, les trames ne sont pas envoyées sur Tableau de Bord, donc le retour est SENDING_STATE_OFF.
Dès lors que Utilisateur souhaite envoyer des trames via l'opération "startSending()", la machine à états passe dans l'état Sending. État durant lequel, l'opération "setSendingState()" va retourner SENDING_STATE_ON afin d'informer que les trames sont en cours d'envoi, comme il est possible de le voir avec "do/send()" qui agit constamment. L'état est quitté lorsque l'opération "stopSending()" est appelée, cela arrête l'envoi des trames et remet l'état de la machine à états sur IDLE.

3 Conception détaillée

La conception détaillée définit l'architecture logique du système, c'est-à-dire :

- répartir le système sur l'architecture matérielle.
- gérer les entrées/sorties et les IHM.
- élaborer les protocoles de communication.
- définir les parallélismes et l'initialisation.
- définir le démarrage, l'arrêt, et la destruction du SàE.

3.1 Architecture physique

Le diagramme ci-dessous représente l'architecture physique du programme CANgateway et de l'application CANdroid dans leur intégralité. En plus des classes vues précédemment, ce diagramme présente la gestion de la communication ainsi que les objets frontières nécessaires au bon fonctionnement du système. Dans ce diagramme, dans un souci de visibilité, nous avons décidé de ne pas détailler les opérations ainsi que les attributs des classes.

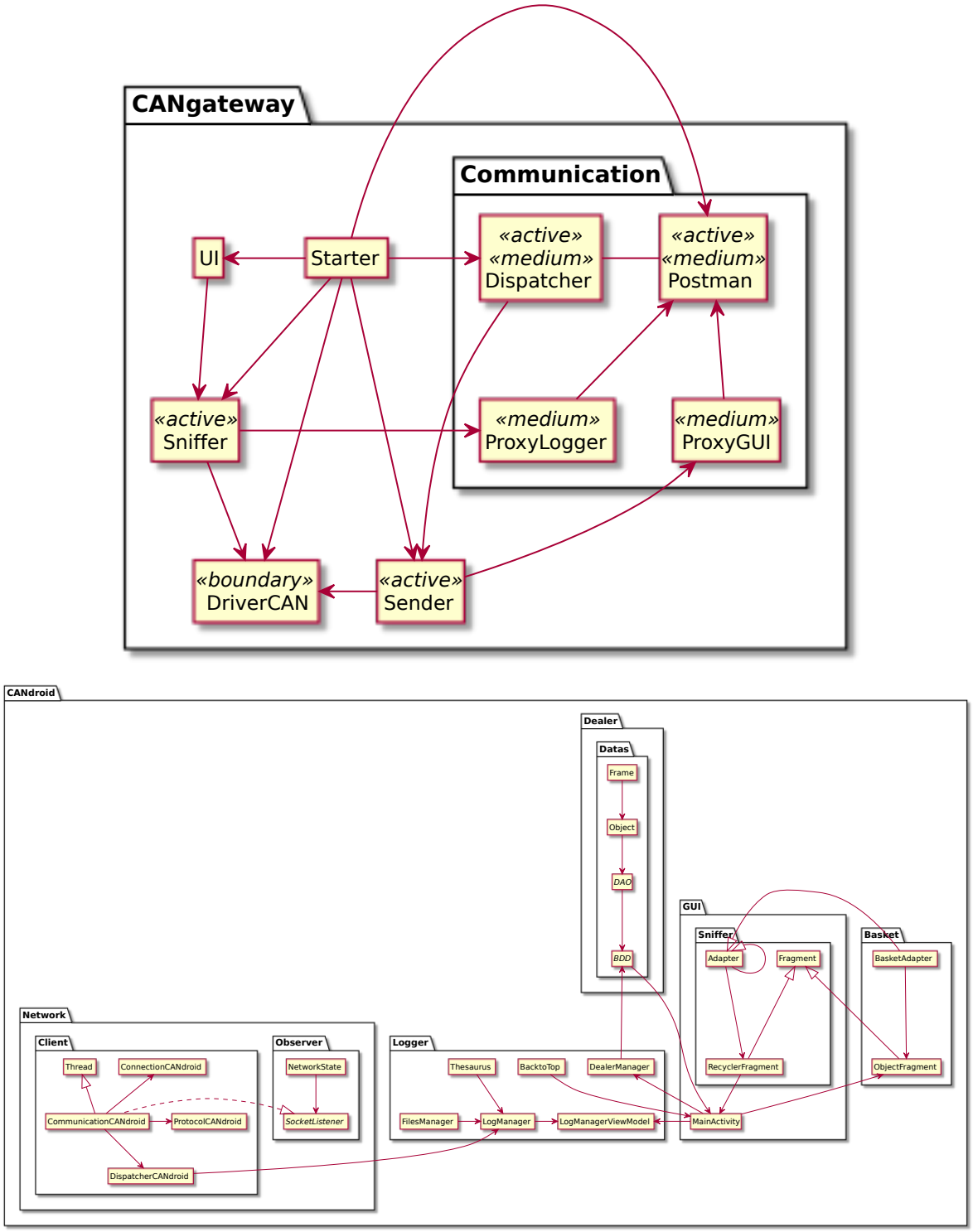

FIGURE 29 – Architecture physique

3.2 Description des classes

3.2.1 Description des classes de CANdroid

Suite à la conception générale, nous avons décidé de diviser certaines classes afin d'améliorer la lisibilité du programme de l'application CANdroid, de faciliter sa réutilisation et son amélioration ultérieure. En effet, certaines classes de la conception générale, telles que Gui, Logger ou encore Basket, sont devenues des packages dans la conception détaillée. Le rôle du package dans la conception détaillée reste identique à celui des classes dans la conception générale. Bien qu'elles aient été divisées, nous fournirons néanmoins une description de chaque classe.

3.2.1.1 Diagramme de classes de CANdroid

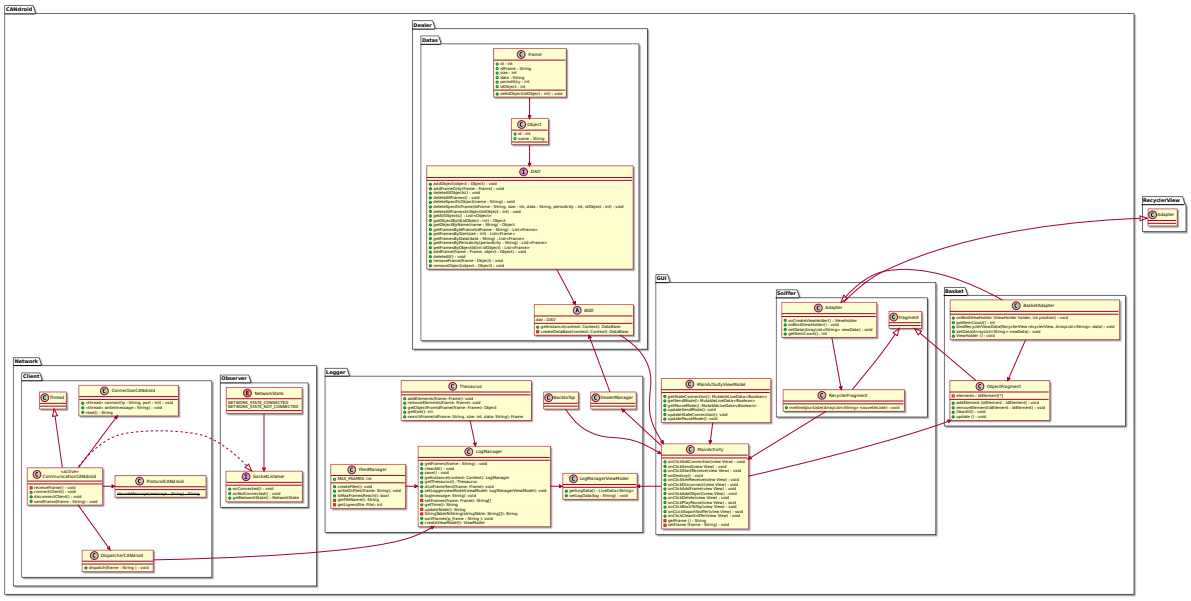

FIGURE 30 – Diagramme de classes de CANdroid

3.2.1.2 La classe Thesaurus

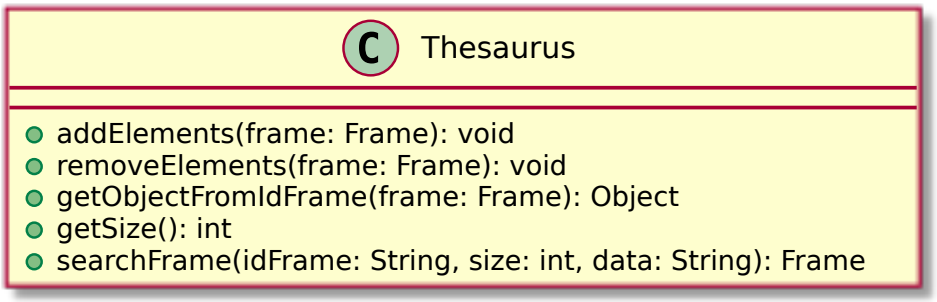

FIGURE 31 – Diagramme de classe de Thesaurus

3.2.1.2.1 Philosophie de conception

La classe Thesaurus a pour rôle de répertorier sous la forme d'un dictionnaire les trames qui sont envoyées par l'application CANdroid.

3.2.1.2.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **getObjectFromIdFrame(idFrame : String) : Object** — Opération qui recherche dans le dictionnaire la trame correspondant à l'identifiant "idFrame". Cet identifiant est lié à un objet qui est retourné afin de savoir si un objet contient cette trame.
- **addElements(frame : Frame) : void** — Opération qui permet d'ajouter un élément au dictionnaire.
- **removeElements(frame : Frame) : void** — Opération qui permet de supprimer un élément du dictionnaire.
- **getObjectFromIdFrame(frame : Frame) : Object** — Opération qui permet de récupérer l'objet à partir d'une trame.
- **getSize() : int** — Opération qui permet de retourner la taille du dictionnaire.
- **searchFrame(idFrame : String, size : int, data : String) : Frame** — Opération qui permet de trouver la trame correspondante dans le dictionnaire sans posséder tous les attributs.

3.2.1.3 La classe FilesManager

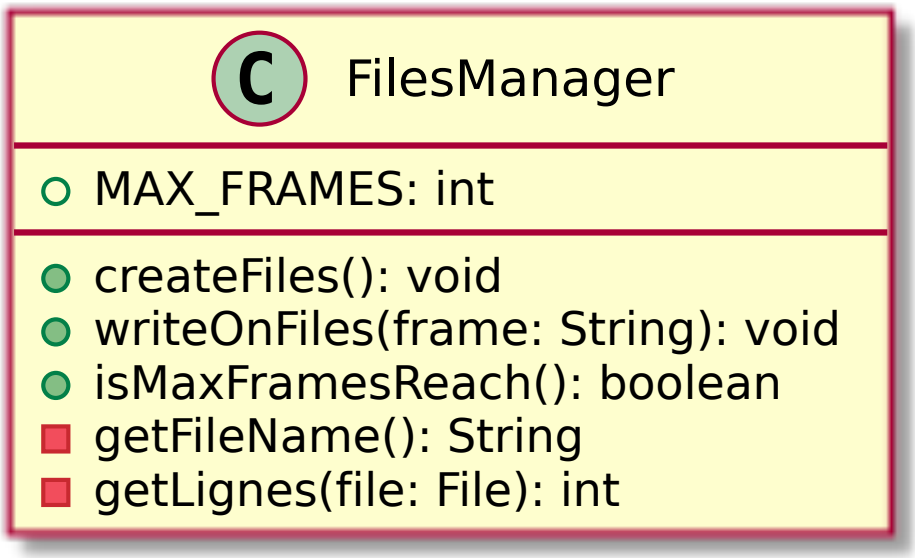

FIGURE 32 – Diagramme de classe de LogManager

3.2.1.3.1 Philosophie de conception

La classe FilesManager a pour rôle de créer et d'écrire sur un fichier sauvegardé sur le téléphone.

3.2.1.3.2 Description structurelle

Attributs :

— **MAX_FRAMES : int** — Nombre maximal de trames par fichier.

Services offerts :

— **createFiles() : void** — Opération qui crée le fichier de sauvegarde des trames.

— **writeOnFiles(frame : String) : void** — Opération qui écrit sur le fichier en cours la trame passée en paramètres.

— **isMaxFramesReach() : bool** — Opération qui vérifie si le nombre de trames est atteint dans le fichier.

— **getFileName() : String** — Opération qui permet de créer le nom du fichier de log.

— **getLignes(file : File) : int** — Opération qui permet de retourner le nombre de lignes présente dans le fichier.

3.2.1.4 La classe LogManager

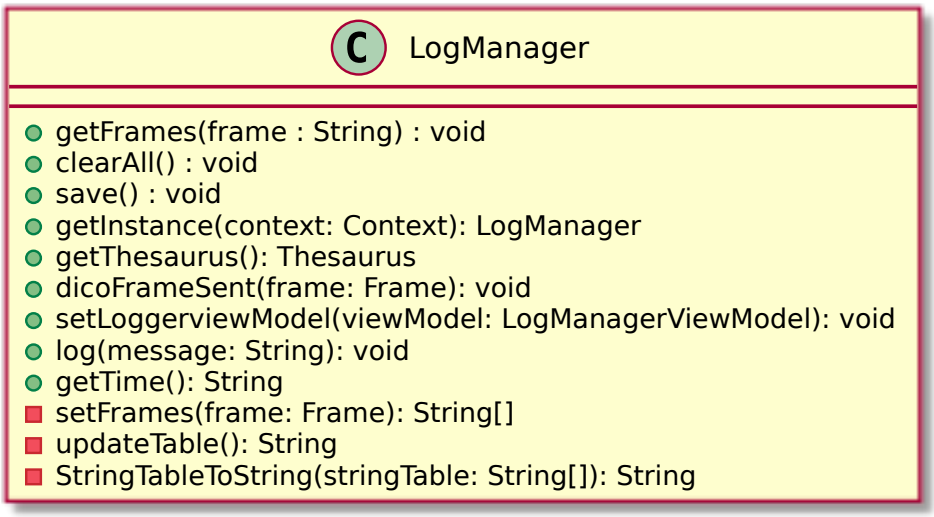

FIGURE 33 – Diagramme de classe de LogManager

3.2.1.4.1 Philosophie de conception

La classe LogManager a pour rôle de traiter les trames reçues depuis la classe DispatcherCANdroid. En effet, lorsque la trame est reçue, elle doit être analysée pour savoir si elle a été envoyée ou non. De plus, elle est mise en forme afin de correspondre à l'affichage.

3.2.1.4.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **dicoFrameSent(frame : Frame) : void** — Opération qui ajoute au dictionnaire la trame envoyée.
- **setFrame(frame : String) : String** — Opération qui met en forme la trame pour l'afficher.
- **getFrames(frame : String) : void** — Opération qui récupère les trames reçues.
- **clearAll() : void** — Opération qui permet de vider le sniffer en faisant appel à logManagerViewModel.
- **save() : void** — Opération qui permet de sauvegarder le fichier en cours.
- **getInstance(context : Context) : LogManager** — Opération qui permet de retourner une instance de la classe.
- **getThesaurus() : Thesaurus** — Opération qui permet de retourner le dictionnaire.
- **dicoFrameSent(frame : Frame) : void** — Opération qui ajoute au dictionnaire la trame envoyée.
- **setLoggerviewModel(viewModel : LogManagerViewModel) : void** — Opération qui permet de définir le viewModel.
- **log(message : String) : void** — Opération qui permet de transmettre le message au viewModel.
- **getTime() : String** — Opération qui permet de retourner l'heure.
- **setFrames(frame : Frame) : String[]** — Opération qui met en forme la trame pour l'afficher.
- **updateTable() : String** — Opération qui permet de mettre à jour le tableau comme souhaité pour l'affichage.
- **StringTableToString(stringTable : String[]) : String** — Opération qui permet de transformer le tableau de String en String.

3.2.1.5 La classe LogManagerViewModel

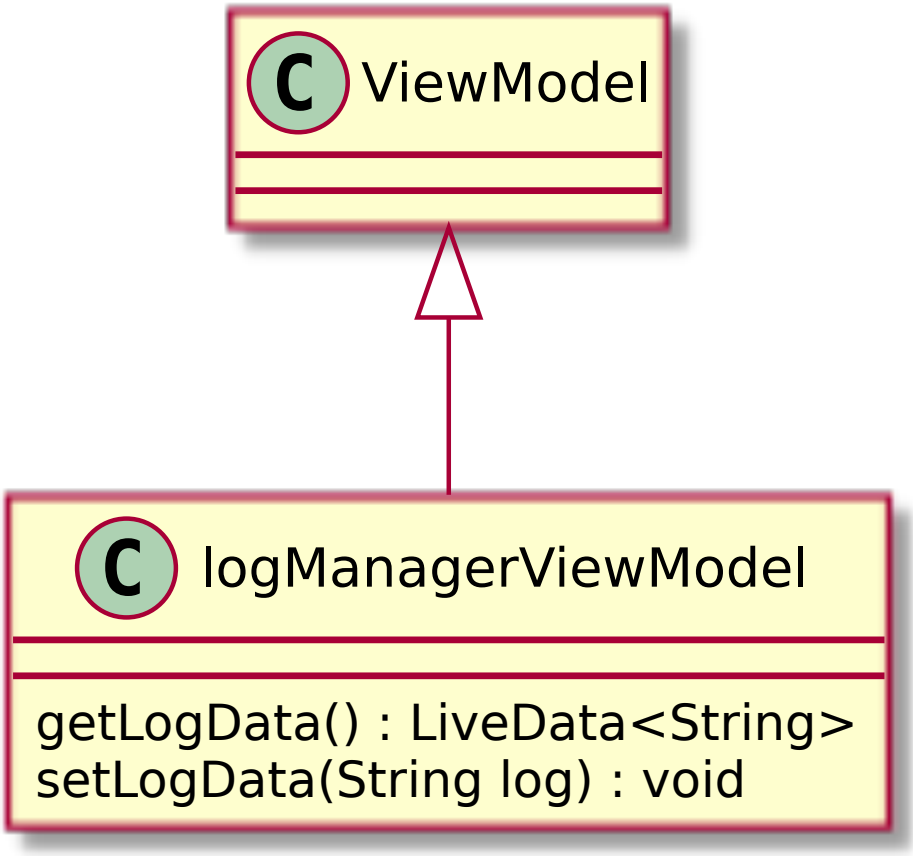

FIGURE 34 – Diagramme de classe de LogManagerViewModel

3.2.1.5.1 Philosophie de conception

La classe LogManagerViewModel permet de faire le lien entre la vue du sniffer et les sources de données adjacentes.

3.2.1.5.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **getLogData() : LiveData<String>** — Opération qui a pour but de stocker des chaînes de caractères représentant les messages de trames.
- **setLogData(log : String) : void** — Opération qui permet de mettre à jour l'interface utilisateur en fonction des changements de logData.

3.2.1.6 L'interface DAO

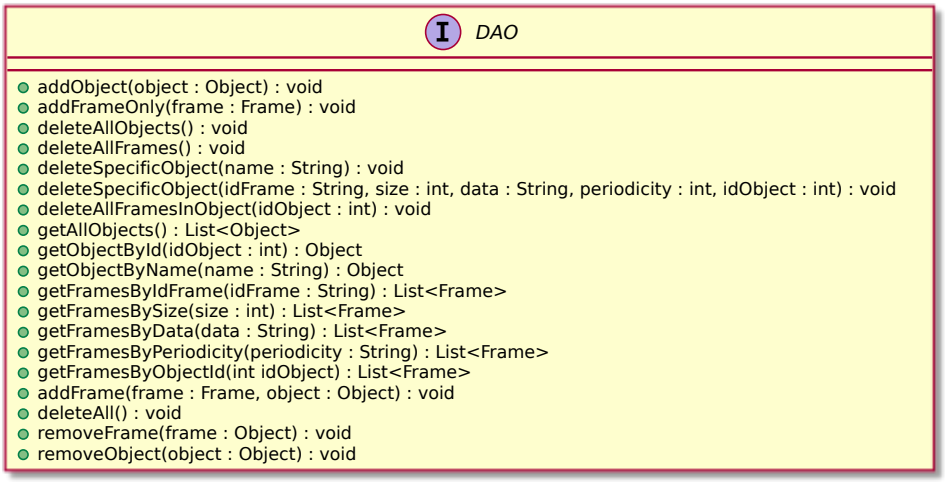

FIGURE 35 – Diagramme de l'interface DAO

3.2.1.6.1 Philosophie de conception

L'interface DAO définit les différents accès possibles à la base de données.

3.2.1.6.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **addObject(object : Object) : void** — Opération qui permet d'ajouter un objet à la base de données.
- **addFrameOnly(frame : Frame) : void** — Opération qui permet d'ajouter une trame à la base de données.
- **deleteAllObjects() : void** — Opération qui permet de supprimer tous les objets de la base de données.
- **deleteAllFrames() : void** — Opération qui permet de supprimer toutes les trames de la base de données.
- **deleteSpecificObject(name : String) : void** — Opération qui permet de supprimer un objet précis dans la base de données.
- **deleteSpecificFrame(idFrame : String, size : int, data : String, periodicity : int, idObject : int) : void** — Opération qui permet de supprimer une trame précise dans la base de données.
- **deleteAllFramesInObject(idObject : int) : void** — Opération qui permet de supprimer toutes les trames dans un objet.

— **getAllObjects() : List<Object>** — Opération qui permet de sélectionner tous les objets créés.

— **getObjectById(idObject : int) : Object** — Opération qui permet de chercher un objet selon son identifiant.

— **getObjectByName(name : String) : Object** — Opération qui permet de chercher un objet selon son nom.

— **getFramesByIdFrame(idFrame : String) : List<Frame>** — Opération qui permet de chercher une ou plusieurs trames selon leur identifiant de trame.

— **getFramesBySize(size : int) : List<Frame>** — Opération qui permet de chercher une ou plusieurs trames selon leur taille.

— **getFramesByData(data : String) : List<Frame>** — Opération qui permet de chercher une ou plusieurs trames selon leur message.

— **getFramesByPeriodicity(periodicity : String) : List<Frame>** — Opération qui permet de chercher une ou plusieurs trames selon leur périodicité.

— **getFramesByObjectId(int idObject) : List<Frame>** — Opération qui permet de chercher une ou plusieurs trames selon leur identifant d'objet.

— **addFrame(frame : Frame, object : Object) : void** — Opération qui permet d'ajouter une trame en fonction d'un objet. Cette opération appelle l'opération qui ajoute la trame à la base de données tout en la liant avec un objet.

— **deleteAll() : void** — Opération qui permet de supprimer tous les éléments de la base de données.

— **removeFrame(frame : Object) : void** — Opération qui permet de supprimer une trame précisement en ne fournissant en paramètre que la trame à supprimer au lieu des différents paramètres.

— **removeObject(object : Object) : void** — Opération qui permet de supprimer un objet précisement en ne fournissant en paramètre que l'objet à supprimer au lieu des différents paramètres.

3.2.1.7 La classe BDD

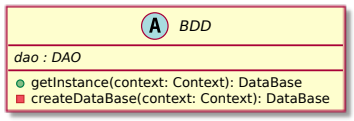

FIGURE 36 – Diagramme de classe de BDD

3.2.1.7.1 Philosophie de conception

La classe BDD a pour rôle d'instancier l'interface DAO.

3.2.1.7.2 Description structurelle

Attributs :

— **dao : DAO** — Opération qui permet d'instancier l'interface DAO.

— **getInstance(context : Context) : DataBase** — Opération qui de créer une instance de la base de données.

— **createDataBase(context : Context) : DataBase** — Opération qui de créer la base de données.

Services offerts :
N.A.

3.2.1.8 La classe Object

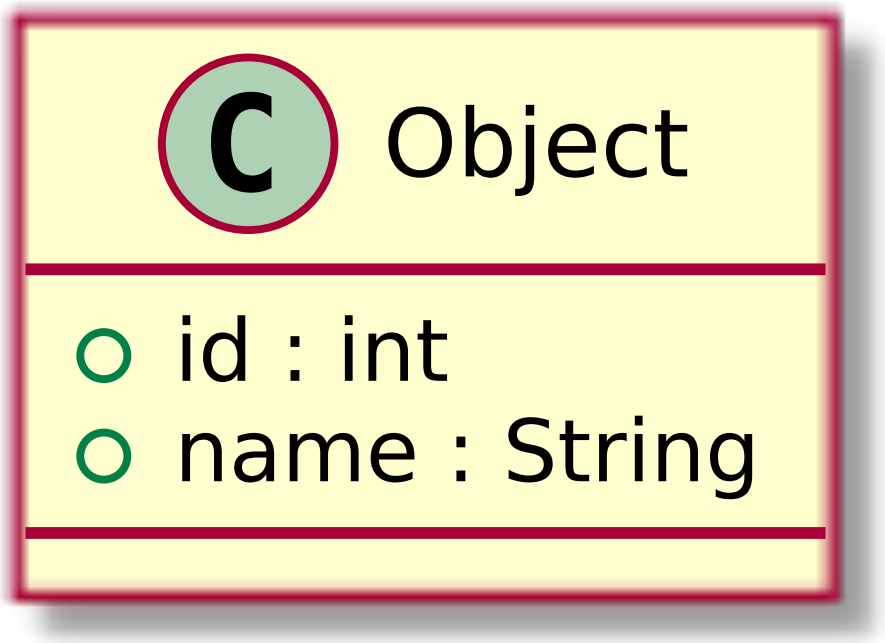

FIGURE 37 – Diagramme de classe de Object

3.2.1.8.1 Philosophie de conception

La classe Object définit les attributs des objets.

3.2.1.8.2 Description structurelle

Attributs :

— **id : int** — Identifiant unique et interne à l'application de l'objet afin de l'identifier.

— **name : String** — Nom de l'objet.

Services offerts :
N.A.

3.2.1.9 La classe Frame

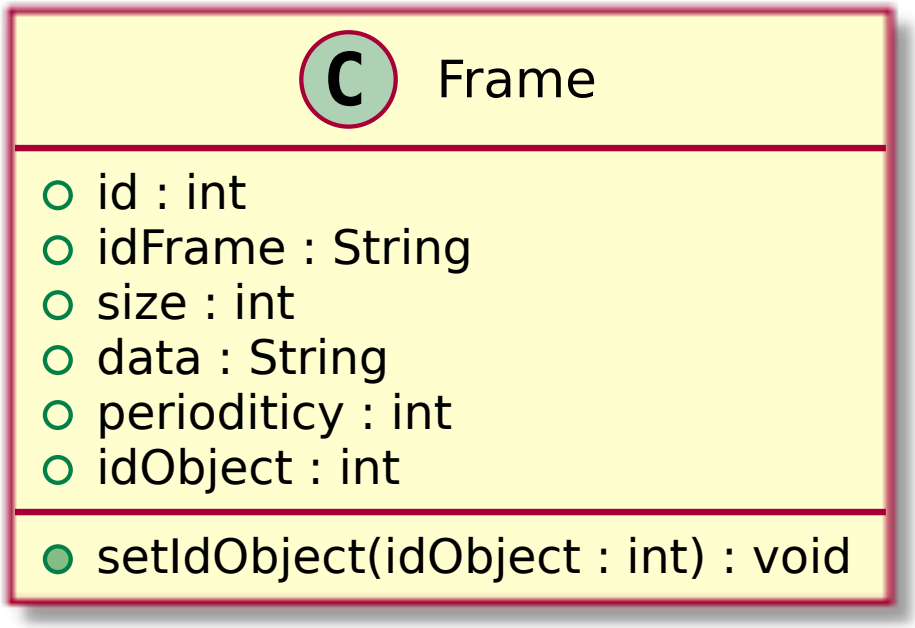

FIGURE 38 – Diagramme de classe de Frame

3.2.1.9.1 Philosophie de conception

La classe Frame définit les attributs des trames.

3.2.1.9.2 Description structurelle

Attributs :

— **id : int** — Identifiant unique et interne à l'application de la trame afin de l'identifier.

— **idFrame : String** — Identifant de la trame.

— **size : int** — Taille de la trame.

— **data : String** — Message de la trame.

— **periodicity : int** — Périodicité de la trame, la valeur est à 0 lorsque la trame est ponctuelle.

— **idObject : int** — Identifiant permettant de lier un objet à la trame.

Services offerts :

— **setIdObject(idObject : int) : void** — Opération qui permet de définir l'id de l'object (idObject) avec l'objet associé. Cela permet de créer une relation entre les trames et les objets.

3.2.1.10 La classe ProtocolCANdroid

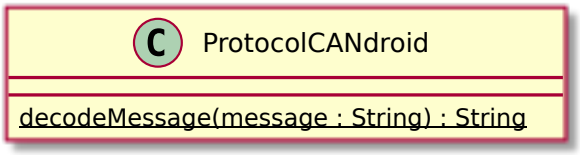

FIGURE 39 – Diagramme de classe de ProtocolCANdroid

3.2.1.10.1 Philosophie de conception

La classe ProtocolCANdroid a pour rôle de décoder le message reçu par le programme CAN-gateway.

3.2.1.10.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **decodeMessage(message : String) : String** — Opération qui permet le décodage du message reçu. L'application CANdroid reçoit ~frame#id$size@message, le décodage permet de retourner #id$size@message au dispatcher.

3.2.1.11 La classe CommunicationCANdroid

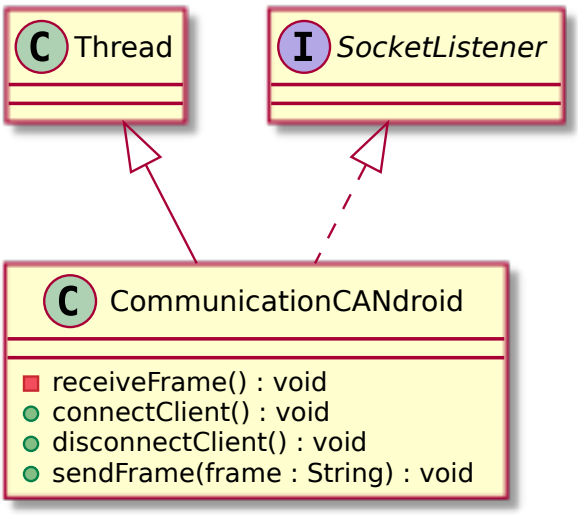

FIGURE 40 – Diagramme de classe de CommunicationCANdroid

3.2.1.11.1 Philosophie de conception

La classe CommunicationCANdroid a pour rôle d'utiliser les opérations de connexion afin de faire la liaison entre le métier et le logiciel.

3.2.1.11.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **receiveFrame() : void** — Opération qui permet de recevoir des trames.
- **connectClient() : void** — Opération qui permet de se connecter au programme CANgateway.
- **disconnectClient() : void** — Opération qui permet de se déconnecter au programme CANgateway.
- **sendFrame(frame : String) : void** — Opération qui permet d'envoyer les trames au programme CANgateway.

3.2.1.12 La classe ConnectionCANdroid

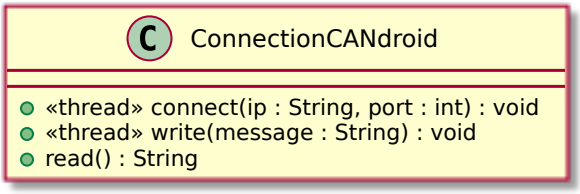

FIGURE 41 – Diagramme de classe de ConnectionCANdroid

3.2.1.12.1 Philosophie de conception

La classe ConnectionCANdroid a pour rôle de se connecter au programme CANgateway, d'envoyer des données et d'en lire.

3.2.1.12.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **connect(ip : String, port : int) : void** — Opération qui permet de se connecter au port et à l'IP donnés en paramètres.
- **write(message : String) : void** — Opération qui permet d'écrire sur le socket.
- **read() : String** — Opération qui permet de lire le socket.

3.2.1.13 L'interface SocketListener

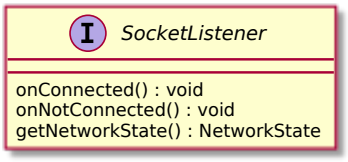

FIGURE 42 – Diagramme de l'interface SocketListener

3.2.1.13.1 Philosophie de conception

L'interface SocketListener a pour rôle de retourner l'état de la connexion.

3.2.1.13.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **onConnected() : void** — Opération qui retourne l'état connecté.
— **onNotConnected() : void** — Opération qui retourne l'état non connecté.
— **getNetworkState() : NetworkState** — Opération qui permet de retourner l'état de la connexion à l'ensemble de l'application.

3.2.1.14 L'énumération NetworkState

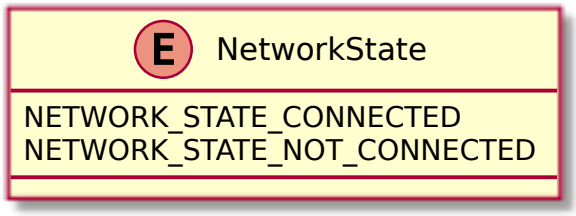

FIGURE 43 – Diagramme de l'énumération NetworkState

3.2.1.14.1 Philosophie de conception

L'énumération NetworkState a été définie précédemment, ici, les énumérations seront uniquement listées.

3.2.1.14.2 Description structurelle

Attributs :
N.A.
Services offerts :
— NETWORK_STATE_CONNECTED
— NETWORK_STATE_NOT_CONNECTED

3.2.1.15 La classe Dispatcher

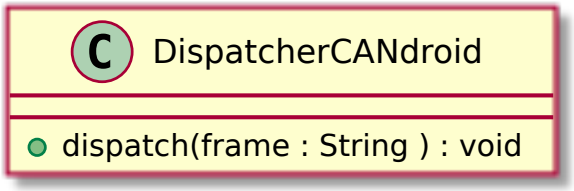

FIGURE 44 – Diagramme de classe de DispatcherCANdroid

3.2.1.15.1 Philosophie de conception

La classe DispatcherCANdroid a pour objectif de faciliter la communication. Elle joue le rôle d'intermédiaire entre la classe CommunicationCANdroid et la classe LogManager. Elle permet une communication centralisée et efficace dans l'application CANdroid.

3.2.1.15.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **dispatch(frame : String) : void** — Opération qui permet l'envoi de la trame vers logManager pour ensuite l'afficher.

3.2.1.16 La classe SendFrames

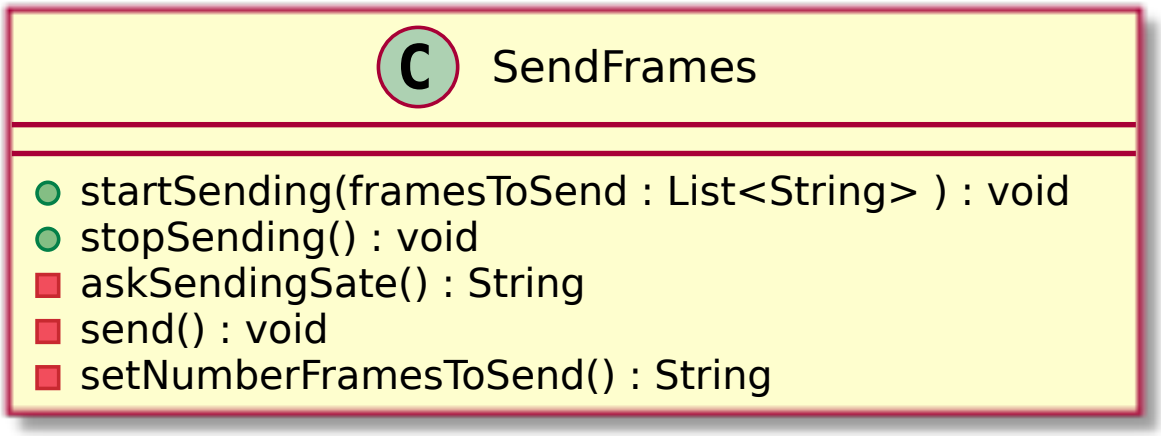

FIGURE 45 – Diagramme de classe de SendFrames

3.2.1.16.1 Philosophie de conception

La classe SendFrames a pour rôle d'envoyer les trames au programme CANgateway.

3.2.1.16.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **startSending(framesToSend : List<String>) : void** — Opération qui permet de commencer à envoyer des trames.

— **stopSending() : void** — Opération qui permet d'arrêter l'envoi des trames.

— **askSendingState() : String** — Opération qui permet l'envoi de la demande d'état afin de savoir si les trames sont en cours d'envoi ou non.

— **send() : void** — Opération qui envoie les différentes informations à envoyer ainsi que les trames.

— **setNumberFramesToSend() : String** — Opération qui permet de connaitre le nombre de trames à envoyer et retourne le nombre sous la forme "nb=X" où X est le nombre de trames à envoyer.

3.2.1.17 La classe ObjectFragment

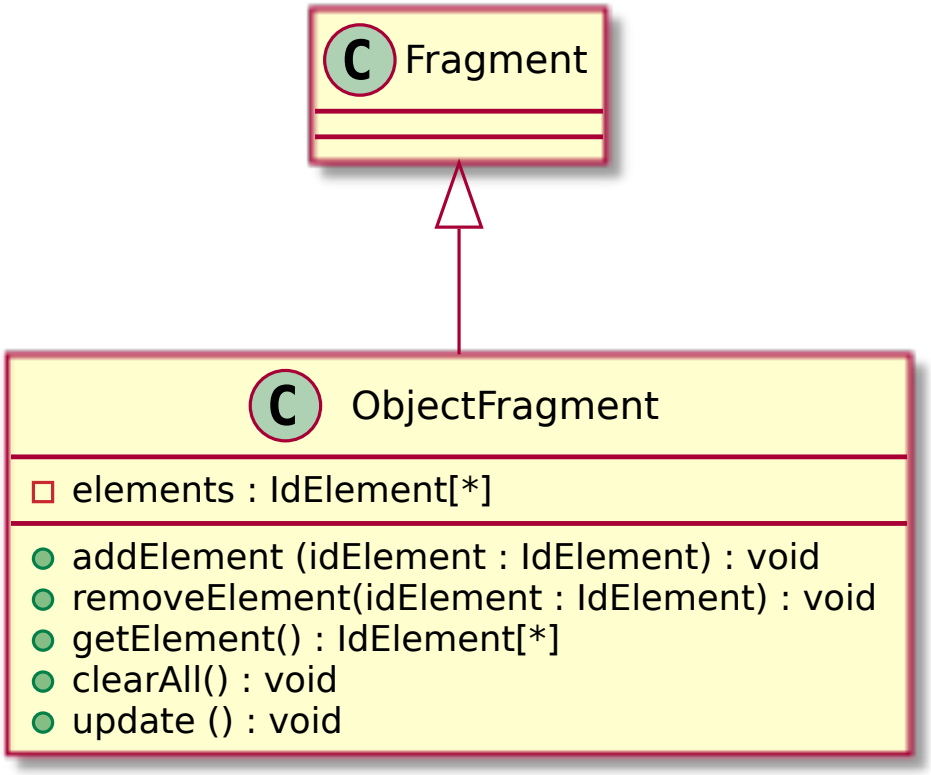

FIGURE 46 – Diagramme de classe de ObjectFragment

3.2.1.17.1 Philosophie de conception

La classe ObjectFragment est la classe gère le fragment des objets. Elle a aussi pour objectif d'effectuer les ajouts et soustractions d'éléments. Les opérations étant déjà détaillées dans la conception générale, elles seront uniquement listées.

3.2.1.17.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **elements : IdElement[*]**

— **addElement (idElement : IdElement)**

— **removeElement(idElement : IdElement)**
— **getElements() : IdElement[*]**
— **clearAll()**
— **update () : void**

3.2.1.18 La classe BasketAdapter

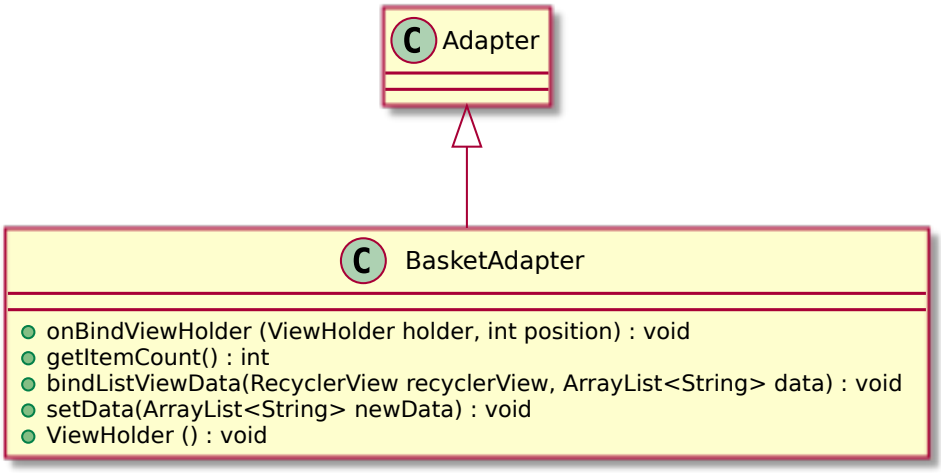

FIGURE 47 – Diagramme de classe de BasketAdapter

3.2.1.18.1 Philosophie de conception

La classe BasketAdapter est la classe d'adapter du listView qui gère les items des objets. Elle permet de mettre en forme chaque éléments individuellement et de mettre en forme l'assemblage.

3.2.1.18.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **onBindViewHolder (ViewHolder holder, int position) : void** — Opération qui permet le binding du ViewHolder.
— **getItemCount() : int** — Opération qui permet de retourner le nombre d'items à répartir dans la liste.
— **bindListViewData(RecyclerView recyclerView, ArrayList<String> data) : void** — Opération qui permet le binding du listView.
— **setData(ArrayList<String> newData) : void** — Opération qui permet la mise à jour de la liste.

— **ViewHolder () : void** — Classe qui permet la création du ViewHolder pour positionner chaque élément individuellement.

3.2.1.19 La classe MainActivity

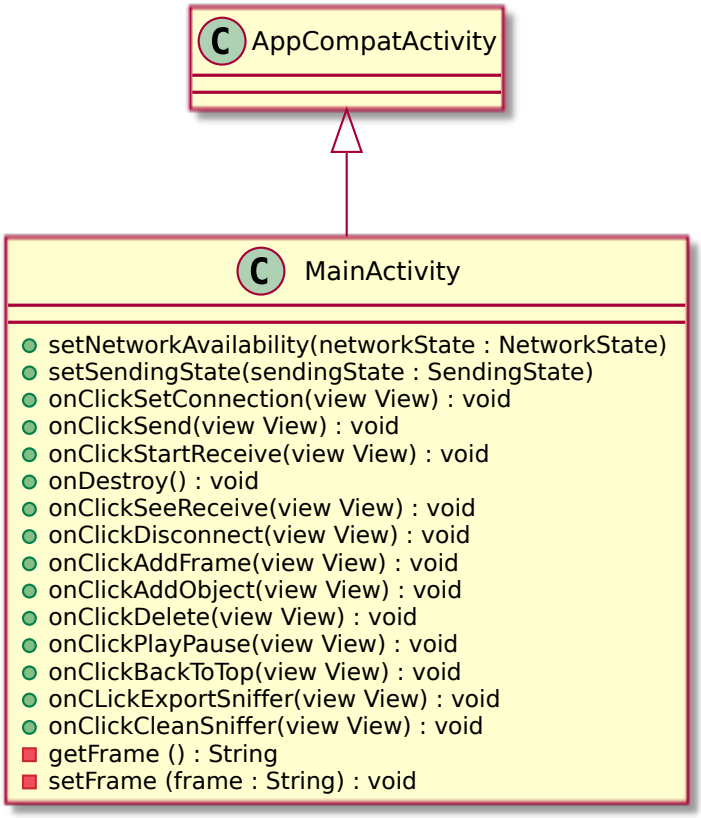

FIGURE 48 – Diagramme de classe de MainActivity

3.2.1.19.1 Philosophie de conception

La classe MainActivity permet de gérer les interfaces utilisateur de l'application CANdroid. Elle est la classe centrale de l'IHM.

3.2.1.19.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **setNetworkAvailability(networkState : NetworkState)** — Opération qui est appelée lorsqu'on clique sur le bouton de connexion.

— **setSendingState(sendingState : SendingState)** — Opération qui permet de mettre à jour l'état d'envoi.

— **onClickSetConnection(view View) : void** — Opération qui est appelée lorsqu'on clique sur le bouton de connexion.

— **onClickSend(view View) : void** — Opération qui est appelée lorsqu'on clique sur le bouton d'envoi.

— **onClickStartReceive(view View) : void** — Opération qui est appelée lorsqu'on clique sur le bouton de réception.

— **onDestroy() : void** — Opération qui est appelée lors de la fermeture de l'activité principale. Elle permet la déconnexion.

— **onClickSeeReceive(view View) : void** — Opération qui est appelée lorsqu'on clique sur le bouton d'affichage de la réception.

— **onClickDisconnect(view View) : void** — Opération qui est appelée lorsqu'on clique sur le bouton de déconnexion.

— **onClickAddFrame(view View) : void** — Opération qui est appelée lors de l'ajout d'une trame dans un objet, par un clique sur le bouton [ajouterTrame].

— **onClickAddObject(view View) : void** — Opération qui est appelée lors de l'ajout d'un objet, par un clique sur le bouton [ajouterObjet].

— **onClickDelete(view View) : void** — Opération qui est appelée lors de la suppression d'un ou plusieurs éléments.

— **onClickPlayPause(view View) : void** — Opération qui est appelée lors de la mise en pause/play du sniffer.

— **onClickBackToTop(view View) : void** — Opération qui est appelée lorsque Utilisateur souhaite revenir en haut du sniffer.

— **onCLickExportSniffer(view View) : void** — Opération qui est appelée lors de l'export du sniffer.

— **onClickCleanSniffer(view View) : void** — Opération qui est appelée lors d'un clique sur le bouton [viderSniffer], c'est à dire, de manière à vider le sniffer des trames actuelles.

— **getFrame () : String** — Opération qui est appelée lorsqu'on souhaite récupérer la dernière trame reçue.

— **setFrame (frame : String) : void** — Opération qui permet de mettre à jour la dernière trame reçue.

3.2.1.20 La classe ObjectAdapter

La classe ObjectAdapter est une classe du package Objects dont nous détaillerons le rôle pour une meilleure compréhension.

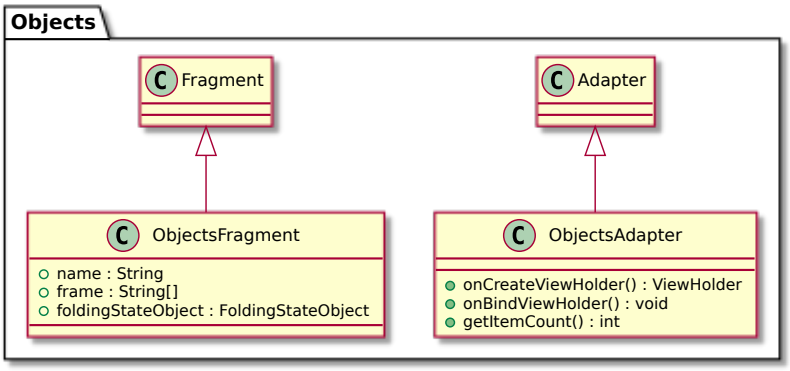

FIGURE 49 – Diagramme du package Objects

3.2.1.20.1 Philosophie de conception

Le package Objects possède la classe ObjectAdapter du RecyclerView qui permet d'afficher la liste des objets. Elle permet de faire le lien entre les fragments et le RecyclerView. Chaque objet est matérialisé dans la liste des objets par une instance de la classe ObjectsFragment.

3.2.1.20.2 Description structurelle

Attributs :
N.A.
Services offerts :
La classe ObjectAdapter contient les opérations suivantes :

- **onCreateViewHolder() : ViewHolder** — Opération qui permet de créer un objet ViewHolder qui représente la vue d'un élément individuel dans le RecyclerView.
- **onBindViewHolder() : void** — Opération qui permet de lier les données d'un objet à la vue correspondante dans le RecyclerView. Elle est appelée lorsque le RecyclerView souhaite afficher ou mettre à jour les données d'un élément spécifique.
- **getItemCount() : int** — Opération qui permet de renvoyer le nombre total d'éléments dans la liste de données de l'adaptateur. Elle est utilisée par le RecyclerView pour déterminer combien d'éléments doivent être affichés.

3.2.1.21 La classe ObjectFragment

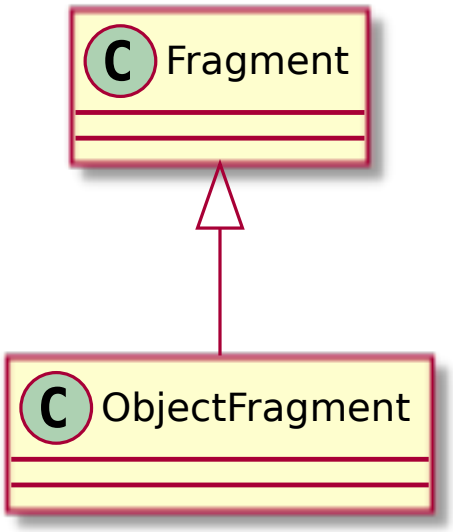

FIGURE 50 – Diagramme de classe de ObjectFragment

3.2.1.21.1 Philosophie de conception

La classe ObjectFragment est la classe associée à chaque fragment d'objet. Cette classe a pour rôle de créer un objet sur l'IHM et de lui associer ses trames. Dans cette classe on détecte aussi si Utilisateur souhaite ajouter de nouvelles trames.

3.2.1.21.2 Description structurelle

Attributs :
Les attribut associés à chaque objets sont :

- String name — nom de l'objet
- String[] frame — trames associées
- FoldingStateObject foldingStateObject — état de l'objet (replié ou déplié)

Services offerts :
N.A.

3.2.1.22 L'énumération FoldingStateObject

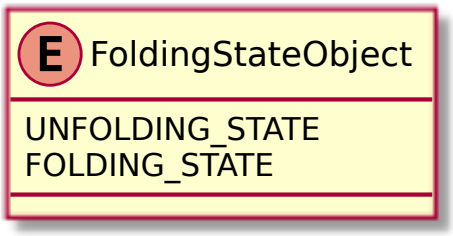

FIGURE 51 – Diagramme de l'énumération FoldingStateObject

3.2.1.22.1 Philosophie de conception

L'énumération FoldingStateObject permet de répertorier les deux états dans lesquels un objet peut être affiché. L'objet est visible sur l'écran, soit sous forme déplié (on peut observer les trames qu'il possède) soit sous forme replié (les potentielles trames qu'il possède ne sont pas visibles). Chaque objet visible sur l'écran possède cet attribut.

3.2.1.22.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **UNFOLDING_STATE** — Correspond à l'état déplié de l'objet (on observe les trames qu'il possède).
- **FOLDING_STATE** — Correspond à l'état replié de l'objet (les trames ne sont pas visibles).

3.2.1.23 La classe MainActivityViewModel

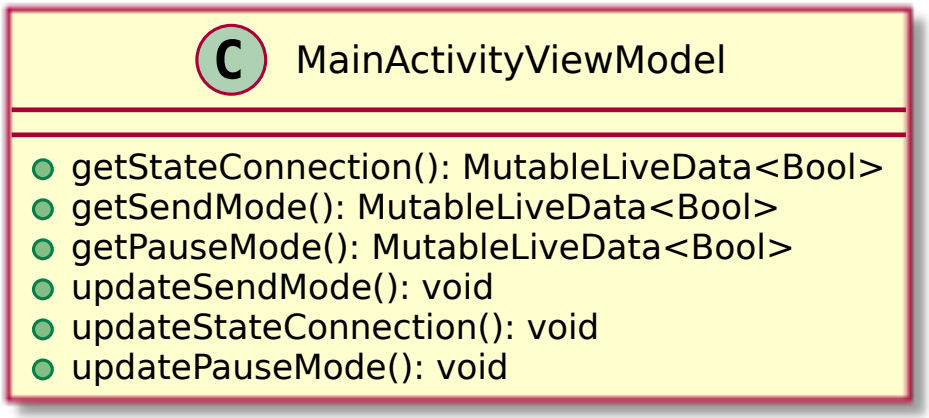

FIGURE 52 – Diagramme de classe de MainActivityViewModel

3.2.1.23.1 Philosophie de conception

La classe MainActivityViewModel a pour rôle de de mettre à jour l'affichage des boutons en fonctions des différents états de l'application.

3.2.1.23.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **getStateConnection() : MutableLiveData<Bool>** — Opération qui retourne l'état de la connexion.
- **getSendMode() : MutableLiveData<Bool>** — Opération qui retourne l'état du mode d'envoi.
- **getPauseMode() : MutableLiveData<Bool>** — Opération qui retourne l'état du mode pause ou play du sniffer.
- **updateSendMode() : void** — Opération qui défini la valeur du mode d'envoi.
- **updateStateConnection() : void** — Opération qui défini la valeur de la connexion.
- **updatePauseMode() : void** — Opération qui défini la valeur du mode pause ou play du sniffer.

3.2.2 Description des classes de CANgateway

3.2.2.1 Diagramme de classes de CANgateway

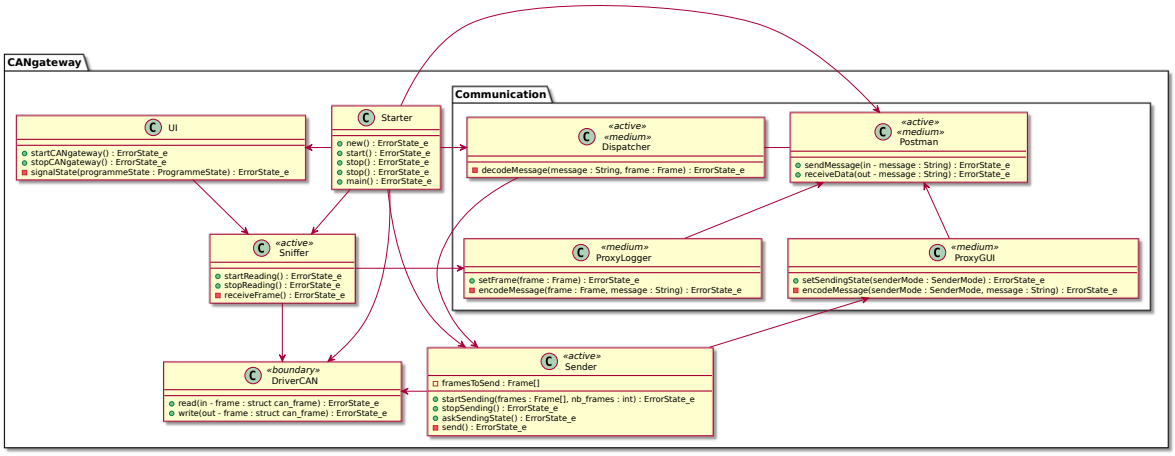

FIGURE 53 – Diagramme de classes de CANgateway

Le diagramme de classes de la figure ci-dessus représente l'architecture du programme CANgateway. Cette architecture ne respecte pas totalement l'architecture de la conception générale due à l'implémentation d'une gestion d'erreur en C qui change donc les types de retours des fonctions. Ces fonctions retournent un état d'erreur de la forme :

— Enumération ErrorState_e
— Éléments :
— — ERROR_STATE_SUCCESS : si tout se passe bien
— ERROR_STATE_FAILURE : s'il y a au moins une erreur
— ERROR_STATE_TIMEOUT : si la cause de l'erreur est un timeout

3.2.2.2 La classe Starter

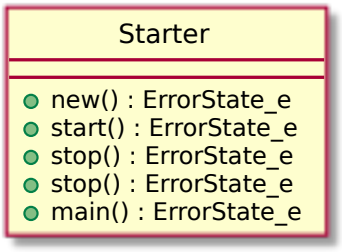

FIGURE 54 – Diagramme de classe de Starter

3.2.2.2.1 Philosophie de conception

La classe Starter a pour rôle de créer et d'initialiser les classes qui seront utilisées par le programme CANgateway.
Les objets créés sont les suivants :

— UI,
— Sender
— Sniffer
— DriverCAN
— Postman
— Dispatcher

3.2.2.2.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **new() : ErrorState_e** — Opération qui crée les objets *ui*, *sender*, *sniffer*, *driverCAN*, *dispatcher* et *postman*.
— **start() : ErrorState_e** — Opération qui lance le programme CANgateway.
— **stop() : ErrorState_e** — Opération qui arrête le programme CANgateway.
— **free() : ErrorState_e** — Opération qui libère la mémoire allouée à l'objet Starter et à tous les objets qu'il a créés.

— **main() : int** — Point d'entrée du programme. Opération qui instancie tous les objets et lance le programme CANgateway.

3.2.2.2.3 Séquence de démarrage et arrêt de CANgateway

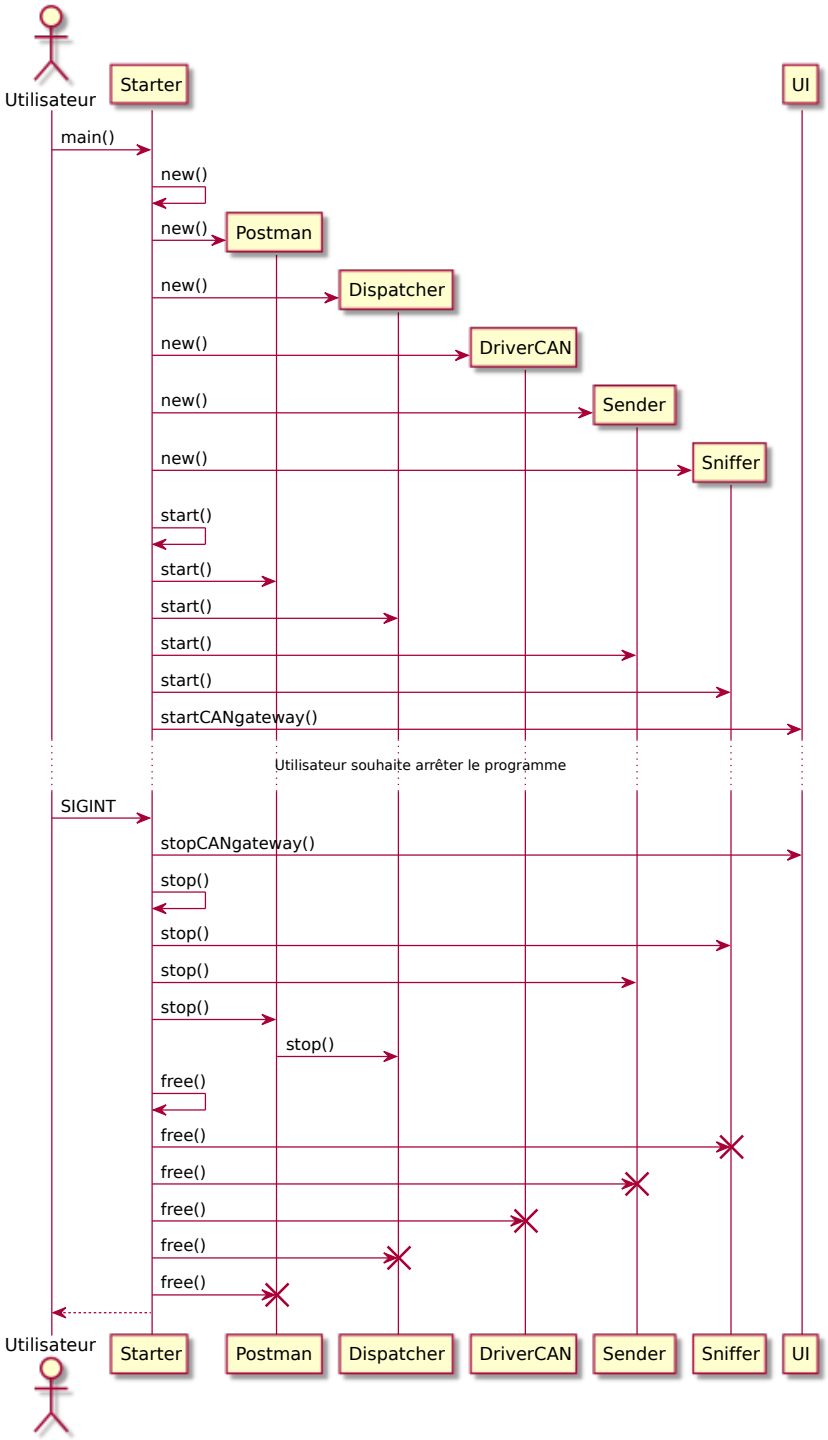

FIGURE 55 – Diagramme de séquence de démarrage de CANgateway

3.2.2.3 La classe DriverCAN

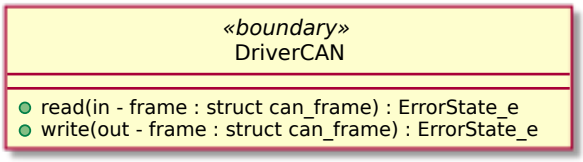

FIGURE 56 – Diagramme de classe de DriverCAN

3.2.2.3.1 Philosophie de conception

La classe DriverCAN a pour rôle de communiquer avec le bus CAN. Elle est utilisée respectivement par les classes Sender et Sniffer pour envoyer et recevoir des trames sur le bus CAN. Cette classe utilise les SocketCAN pour communiquer avec le bus CAN et est donc dépendante de la bibliothèque can.h de Linux.

3.2.2.3.2 Description structurelle

Attributs : N.A.
Services offerts :

- **write(in - frame : struct can_frame) : ErrorState_e** — Opération qui envoie une trame sur le bus CAN. Elle renvoie un code d'erreur si l'envoi de la trame a réussi ou échoué.
- **read(out - frame : struct can_frame) : ErrorState_e** — Opération qui lit une trame sur le bus CAN. Elle renvoie un code d'erreur si la lecture de la trame a réussi ou échoué.

3.2.2.4 La classe Dispatcher

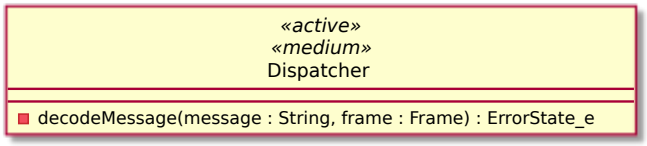

FIGURE 57 – Diagramme de classe de Dispatcher

3.2.2.4.1 Philosophie de conception

La classe Dispatcher a pour rôle de lire et décoder les trames et les demandes d'états du Mode Envoi reçu. Ces informations sont sous forme de chaînes de caractères lors de leur réception par la classe Dispatcher. Il transmet les trames décodées à la classe Sender.

3.2.2.4.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **decodeMessage(message : String, frame : Frame) : ErrorState_e** — Opération qui décode une chaîne de caractères en une structure représentant une trame à envoyer sur le réseau CAN. Permet également le décodage des demandes d'état du Mode Envoi.

3.2.2.5 La classe Postman

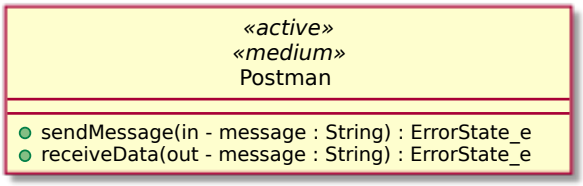

FIGURE 58 – Diagramme de classe de Postman

3.2.2.5.1 Philosophie de conception

Postman est une classe qui permet de communiquer avec l'application CANdroid. Elle échange divers messages avec l'application suivant le protocole de communication défini dans la sous-section 3.3.

3.2.2.5.2 Description structurelle

Attributs :
N.A.
Services offerts :

— **sendMessage(in - message : String) : ErrorState_e** — Opération qui permet d'envoyer un message à l'application CANdroid.

— **receiveData(out - message : String) : ErrorState_e** — Opération qui permet de recevoir un message de l'application CANdroid.

3.2.2.6 La classe ProxyGUI

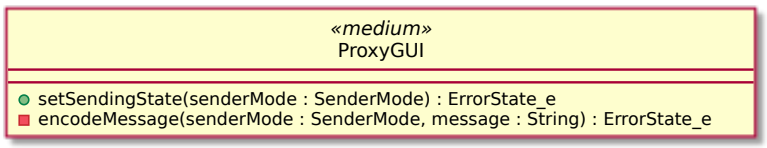

FIGURE 59 – Diagramme de classe de ProxyGUI

3.2.2.6.1 Philosophie de conception

La classe ProxyGUI qui est dans le programme CANgateway a pour rôle de simuler le comportement de la classe GUI présente dans l'application CANdroid. Toutes les requêtes que Sender doit envoyer à GUI passent par cette classe.

3.2.2.6.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **setSendingState(senderMode : SenderMode) : ErrorState_e** — Opération qui permet de définir l'état du Mode Envoi du système.
- **encodeMessage(senderMode : SenderMode, message : String) : ErrorState_e** — Opération qui permet d'encoder un état du Mode Envoi en une chaine de caractère.

3.2.2.7 La classe ProxyLogger

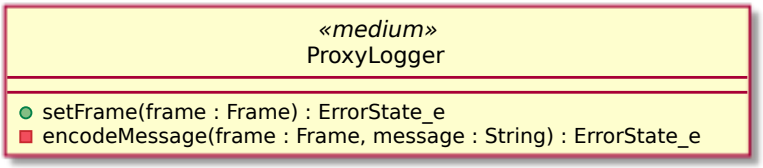

FIGURE 60 – Diagramme de classe de ProxyLogger

3.2.2.7.1 Philosophie de conception

La classe ProxyLogger qui est dans le programme CANgateway a pour rôle de simuler le comportement de la classe Logger présente dans l'application CANdroid.

3.2.2.7.2 Description structurelle

Attributs :
N.A.
Services offerts :

- **setFrame(frame : Frame) : ErrorState_e** — Opération qui permet de fournir les trames sniffées au Logger.
- **encodeMessage(frame : Frame, message : String) : ErrorState_e** — Opération qui permet d'encoder une trame en une chaîne de caractères.

3.3 Protocole de communication

Notre protocole de communication repose sur l'utilisation de sockets TCP/IP. Les données échangées sont des chaînes de caractères, assurant ainsi un protocole textuel.

3.3.1 Protocole de communication de CANdroid vers CANgateway

3.3.1.1 Formalisation du protocole

Lors de la schématisation des formats des messages, nous avons choisi de séparer les parties des messages par des espaces pour des questions de lisibilité. Cependant, ces espaces ne sont pas présents lors de l'exécution du protocole. Par conséquent, nous ne les avons pas représentés dans la partie 3.3.1.2.

L'application CANdroid transmet trois types d'informations au programme CANgateway : une demande de l'état du Mode Envoi, une demande d'arrêt d'envoi des trames et une séquence de trames.

Lorsque Utilisateur envoie des trames, l'application CANdroid transmet un message lui-même composé de plusieurs messages correspondant à des chaînes de caractères. Ce message comprend : un message indiquant le nombre trames envoyées dans la séquence, et un message par trame de la séquence. L'application CANdroid envoie donc une donnée sous le format suivant :

nb= N \n	# ID $ LENGTH @ DATA / PERIODICITY \n	...

- **nb=** : indique que la suite du message sera N ;
- **N** : correspond au nombre de trames dans la séquence à envoyer ;
- **\n** : caractère de fin du message, indiquant que le prochain message sera la première trame de la séquence ;
- **#** : indique que la suite du message est ID ;
- **ID** : ID de la trame CAN lu sur le réseau CAN. ID correspond à un nombre hexadécimal à 3 chiffre. Dans notre cas, nous traitons des trames au format standard, ID peut donc prendre une valeur minimale de 0x000 et une valeur maximale de 0x7FF, ce qui correspond à 2047 en décimal ;
- **$** : séparateur entre ID et LENGTH ;
- **LENGTH** : taille de la trame. Cette valeur est un entier qui correspond à un nombre d'octet. Au format standard, une trame a une longueur maximale de 8 octets, LENGTH va donc de 0 à 8 ;
- **@** : séparateur entre LENGTH et DATA ;
- **DATA** : correspond aux données de la trame. Au format standard, on peut avoir jusqu'à 8 octets de données, où chaque octet peut contenir une valeur hexadécimale allant de 0x00 à 0xFF ;

— / : séparateur entre DATA et PERIODICITY ;
— **PERIODICITY** : correspond à la périodicité de l'envoi de la trame. Cette valeur est un entier qui a pour valeur minimale 0, et n'a pas de valeur maximale. Cette valeur est en milliseconde ;
— **\n** : caractère de fin de message ;
— **...** : représente les N autres trames de la séquence.

Utilisateur peut demander l'arrêt d'envoi des trames. L'application CANdroid envoie alors une demande d'état du Mode Envoi sous forme de chaîne de caractères. Cette donnée est envoyée sous le format suivant :

! \n

— **!** : caractère définissant la nature du message, c'est-à-dire une demande d'arrêt du Mode Envoi ;
— **\n** : caractère de fin de message.

Lorsque Utilisateur souhaite envoyer des trames, ou arrêter l'envoi de trames, le Mode Envoi doit être mis à jour. L'application CANdroid envoie alors une demande d'état du Mode Envoi sous forme de chaîne de caractères. Cette donnée est envoyée sous le format suivant :

? \n

— **?** : caractère définissant la nature du message, c'est-à-dire une demande d'état du Mode Envoi ;
— **\n** : caractère de fin de message.

3.3.1.2 Exemples

Pour rappel, nous appliquons un protocole textuel, les données envoyées sont sous la forme de chaîne de caractères.

3.3.1.2.1 Envoyer des trames

Les échanges présentés dans le diagramme de la figure 9 représente un envoi de donnée de l'application CANdroid vers le programme CANgateway via le protocole de communication décrit dans le paragraphe 3.3.1.1. Cette donnée correspond à une séquence de trame et peut être de la forme suivante :

nb=2\n	#19B$6@00000E000000/1000\n	#19B$6@000007000000/0\n

La première trame est envoyée toutes les secondes, et permet d'ouvrir la portière de la porte avant-gauche sur le Simulateur ICSim. La seconde trame est envoyée une seule fois et permet d'ouvrir la portière arrière droite sur le Simulateur ICSim.
Ensuite, une seconde donnée est envoyée, correspondant à une demande d'état du Mode Envoi. Elle est de la forme suivante :

? \n

3.3.1.2.2 Arrêter d'envoyer des trames

Les échanges présentés dans le diagramme de la figure 10 représente un envoi de données de l'application CANdroid vers le programme CANgateway via le protocole de communication décrit dans le paragraphe 3.3.1.1. Ces données correspondent à une demande d'arrêt d'envoi de trames, puis une demande d'état du mode Envoi. Elles sont de la forme suivante :

!\n

Puis ce message :

?\n

3.3.2 Protocole de communication de CANgateway vers CANdroid

3.3.2.1 Formalisation du protocole

Lors de la schématisation des formats des messages, nous avons choisi de séparer les parties des messages par des espaces pour des questions de lisibilité. Cependant, ces espaces ne sont pas présents lors de l'exécution du protocole. Par conséquent, nous ne les avons pas représentés dans la partie 3.3.2.2.

Le programme CANgateway transmet deux types d'informations à l'application CANdroid : une trame et un état du Mode Envoi. Les trames sont celles sniffées sur le réseau CAN. Après encodage en chaîne de caractères, elles sont envoyées sous le format suivant :

~ frame # ID $ LENGTH @ DATA \n

- ~ : caractère de début de message ;
- **frame** : indique la nature du message envoyé. Dans le cas suivant le message est une trame ;
- # : séparateur entre la nature du message et l'ID de la trame ;
- **ID** : ID de la trame CAN lu sur le réseau CAN. ID correspond à un nombre hexadécimal à 3 chiffre. Dans notre cas, nous traitons des trames au format standard, ID peut donc

prendre une valeur minimale de 0x000 et une valeur maximale de 0x7FF, ce qui correspond à 2047 en décimal ;

— **$** : séparateur entre ID et LENGTH ;

— **LENGTH** : taille de la trame. Cette valeur est un entier qui correspond à un nombre d'octet. Au format standard, une trame a une longueur maximale de 8 octets, LENGTH va donc de 0 à 8 ;

— **@** : séparateur entre LENGTH et DATA ;

— **DATA** : correspond aux données de la trame. Au format standard, on peut avoir jusqu'à 8 octets de données, où chaque octet peut contenir une valeur hexadécimale allant de 0x00 à 0xFF ;

— **\n** : caractère de fin de message.

L'état du Mode Envoi est envoyé en réponse à une demande d'état du Mode Envoi. Après encodage en chaîne de caractères, cette information est sous le format suivant :

~ sendingState # DATA \n

— ~ : caractère de début de message ;

— **sendingState** : indique la nature du message envoyée. Dans le cas suivant le message est un état du Mode Envoi ;

— # : séparateur entre la nature du message et DATA ;

— **DATA** : correspond à l'état du Mode Envoi. DATA peut prendre les deux valeurs logiques : 0 si l'état du Mode Envoi est OFF, et 1 si l'état du Mode Envoi est OFF ;

— **\n** : caractère de fin de message.

3.3.2.2 Exemples

Pour rappel, nous appliquons un protocole textuel, les données envoyées sont sous la forme de chaîne de caractères.

3.3.2.2.1 Recevoir des trames

Les échanges présentés dans le diagramme de la figure 5 représente un envoi de donnée du programme CANgateway vers l'application CANdroid via le protocole de communication décrit dans le paragraphe 3.3.2.1. Cette donnée correspond à une trame et peut être de la forme suivante :

~frame#19B$6@00000E000000\n

Cette trame permet d'ouvrir la porte avant-gauche de la portière sur le Simulateur ICSim.

3.3.2.2.2 Envoyer des trames

Les échanges présentés dans le diagramme de la figure 9 représente un envoi de donnée du programme CANgateway vers l'application CANdroid via le protocole de communication décrit dans le paragraphe 3.3.2.1. Il y a deux données transmises lors d'un envoi de trames :

— L'état du Mode Envoi ;

— La trame envoyée.

L'état du Mode Envoi est de la forme suivante :

~sendingState#1\n

La trame envoyée est de la forme suivante :

~frame#19B$6@00000E000000\n

3.3.2.2.3 Arrêter d'envoyer des trames

Les échanges présentés dans le diagramme de la figure 10 représente un envoi de donnée du programme CANgateway vers l'application CANdroid via le protole de communication décrit dans le paragraphe 3.3.2.1. Cette donnée correspond à un état du Mode Envoi et est de la forme suivante :

~sendingState#0\n

3.4 Gestion du multitâche

3.4.1 Identification des accès concurrents

3.4.1.1 Côté CANdroid

Tâches	ConnectionCANdroid	ProtocolCANdroid	DispatcherCANdroid
GUI			
Interface DAO			
CommunicationCANdroid	X	X	X

Table 5 – Accès concurrents côté CANdroid

Analyse : GUI est considéré ici en tant que package car il correspond à l'interface dans sa globalité. Il n'y a pas de potentiels accès concurrents car uniquement CommunicationCANdroid peut accéder aux opérations liées à la connexion. L'interface ne peut pas poser de potentiels accès concurrents car l'appel des opérations de l'interface se font via un ExecutorService, qui va s'assurer de ne pas bloquer le thread principal. De son coté, GUI va être utilisé par l'UI Thread et donc a son propre thread et ne peut pas poser de problèmes d'accès concurrents.

3.4.1.2 Côté CANgateway

Tâches	DriverCAN	ProxyLogger	ProxyGUI
UI			
Postman			
Dispatcher			
Sniffer	X	X	
Sender	X	X	X

Table 6 – Accès concurrents côté CANgateway

Analyse générale : Il y a de potentiels accès concurrents entre les tâches Sniffer et Sender. En effet, ces deux tâches utilisent la classe DriverCAN pour communiquer avec le bus CAN et la classe ProxyLogger pour informer l'application CANdroid des trames envoyées et reçues sur le bus CAN. Il est donc nécessaire de protéger les accès concurrents à ces classes.

Problèmes avec la classe ProxyLogger : En réalité, il n'y a pas de problèmes d'accès concurrents avec la classe ProxyLogger car elle utilise la classe Postman afin de transmettre les informations à fournir à Logger et que cette classe est déjà protégée contre les accès concurrents.

Problèmes avec la classe DriverCAN : La classe DriverCAN utilise les socketCAN pour communiquer avec le bus CAN. Or les classes Sender et Sniffer utilisent deux sockets différentes. La classe Sender n'utilise sa socketCAN que pour écrire sur le bus. À l'inverse,

la classe Sniffer n'utilise sa socketCAN que pour lire sur le bus. Il n'y a donc pas de réels problèmes d'accès concurrents.

3.5 Gestion de la persistance

Afin de gérer la persistance des données dans l'application CANdroid, une base de données a été utilisée. Cette base de données se décompose en deux classes qui sont les objets Object et les trames Frame. Ces deux classes consitituent le coeur de la base de données. Il y a également une interface appelée DAO, qui permet de faire des recherches dans la base de données. Ces recherches sont des requêtes SQL appliquées à la base de données. Afin d'utiliser la base de données, une instance de la base de données est nécessaire, c'est l'utilité de la classe BDD. Cette instance de la base de données permet d'accéder à l'interface ainsi qu'à toutes ses requêtes.

4 Dictionnaire de domaine

— **ASK_DELAY** : Correspond à un délay de demande d'informations de 500 ms.

— **BackToTop** : Correspond en anglais à "La fin du fil" dans [dossier_de_specification_SPEC_B1_2024] dans le dictionnaire de domaine.

— **Boutons** : Pour rappel, les boutons et leur positionnement sont décrit dans [dossier_de_specification_SPEC_B1_2024] dans la section sur les IHM.

— **CAN** : Controller Area Network, il s'agit d'un protocole de communication série utilisé pour connecter par exemple des capteurs et des actionneurs.

— **Champs de textes** : Tout comme pour les boutons, les champs de textes et leur positionnement sont décrits dans [dossier_de_specification_SPEC_B1_2024] dans la section sur les IHM.

— **Correspondance des opérations de la classe GUI avec le contexte logique dans [dossier_de_specification_SPEC_B1_2024]** :
  — acceptRequest() : correspond à l'opération valider() du contexte logique.
  — addFrame() : correspond à l'opération ajouterTrame() du contexte logique.
  — addObject() : correspond à l'opération ajouterObjet() du contexte logique.
  — askReconnection() : correspond à l'opération reconnecter() du contexte logique.
  — backToTopSniffer() : correspond à l'opération revenirEnHaut() du contexte logique.
  — cleanSniffer() : correspond à l'opération supprimerTramesSniffer() du contexte logique.
  — closeObjectMenu(idElement : IdElement) : correspond à l'opération fermerMenuObjet(idElement : IdElement) du contexte logique.
  — delete() : correspond à l'opération supprimer() du contexte logique.
  — displayMainScreen() : correspond à l'opération afficherEcranPrincipal() du contexte logique.
  — displayPopup(idScreenPopup : IdScreenPopup) : correspond à l'opération afficherPopup(idScreenPopup : IdScreenPopup) du contexte logique.
  — exportSniffer() : correspond à l'opération exporterTramesSniffer() du contexte logique.
  — openObjectMenu(idElement : IdElement) : correspond à l'opération ouvrirMenuObjet(idElement : IdElement) du contexte logique.
  — pauseSniffer() : correspond à l'opération desactiverReceptionTrames() du contexte logique.
  — rejectRequest() : correspond à l'opération refuser() du contexte logique.

  - resumeSniffer() : correspond à l'opération activerReceptionTrames() du contexte logique.
  - select(idElement : IdElement) : correspond à l'opération selectionner(idElement : IdElement) du contexte logique.
  - setFrame(frame : string) : correspond à l'opération ecrireTrame(trame : string) du contexte logique.
  - setObject(object : string) : correspond à l'opération nommerObjet(nom : string) du contexte logique.
  - setSenderMode(senderMode : StructSenderMode) : correspond à l'opération definirModeEnvoiTrame(modeEnvoi : booléen) du contexte logique.
  - setPeriodicity(periodicity : int) : correspond à l'opération saisirPeriodicite(peridodicite : int) du contexte logique.
  - unselect(idElement : IdElement) : correspond à l'opération deselectionner(idElement : IdElement) du contexte logique.

- **Écrans utilisés** : les écrans utilisés sont décrits dans [dossier_de_specification_SPEC_B1_2024] dans la section sur les IHM. Voici un rappel des traductions anglaises des noms des écrans utilisés dans le dossier :
  - **MainScreen** : Correspond à EcranPrincipal.
  - **PopupAddObject** : Correspond à PopupAjoutObjet.
  - **PopupAskReconnection** : Correspond à PopupDemandeReconnexion.
  - **PopupDeleteElement** : Correspond à PopupSuppressionElement.
  - **PopupFailNumberObject** : Correspond à PopupErreurNombreObjet.
  - **PopupFailAddObject** : Correspond à PopupErreurAjoutObjet.
  - **PopupFailWritingFrame** : Correspond à PopupErreurSaisieTrame.
  - **PopupFailNumberFrame** : Correspond à PopupErreurNombreTrame.
  - **PopupFrameSendingMode** : Correspond à PopupModeEnvoiTrame.
  - **PopupStopSend** : Correspond à PopupArretEnvoi.

- **Format de la trame** : Afin d'éviter à Utilisateur de taper tous les zéros non significatifs lors de la saisie d'une trame, les trames doivent être saisies avec des séparateurs de la forme suivante :
  - #id$size@message

- **Git** : Correspond au dépôt utilisé par l'équipe CANvengers dans le cadre du projet Passerelle Android-CAN vers banc CAN réel ou simulé.

- **LED** : *light-emitting diode*, Diode électroluminescente présente sur la Raspberry PI.

— **Mode Envoi** : ce mode définit si des trames sont en cours d'envoi ou non.

— **Nom d'objet par défaut** : Le nom par défaut est le nom donné lorsque Utilisateur ne spécifie pas de nom pour l'ajout d'un objet. Le nom par défaut sera "Objet_" suivi de l'identifiant le plus grand déjà utilisé pour un objet, augmenté de 1. Par exemple, si les identifiants des derniers objets créés sont 10, 11 et 12, le prochain objet créé aura le nom par défaut "Objet_13".

— **Raspberry PI** : Correspond à E_Raspberry dans [dossier_de_specification_SPEC_B1_2024], ordinateur monocarte créé par la Fondation Raspberry PI.

— **Smartphone** : Correspond à E_Smartphone dans [dossier_de_specification_SPEC_B1_2024], téléphone portable sous système Android.

— **Sniffer** : Un sniffer est un programme qui capture tous les paquets circulant dans le réseau. Dans notre cas, le terme sniffer est utilisé pour désigner le terminal d'affichage des trames du réseaux CAN de l'application CANdroid.

— **Tableau de Bord** : Représente l'un des (ou les) deux systèmes ci-dessous :
  — **Simulateur ICSim (correspond à E_ICSim dans [dossier_de_specification_SPEC_B1_2024])** : Simulateur d'un tableau de bord de voiture.
  — **Banc de test (correspond à E_Banc_De_Test dans [dossier_de_specification_SPEC_B1_2024])** : Banc de test d'un tableau de bord de voiture

  Il est connecté au SàE afin d'envoyer et recevoir des trames CAN. Dans tout le dossier, on emploie le terme Tableau de Bord (correspond à E_TableauDeBord dans [dossier_de_specification_SPEC_B1_2024]) au singulier, car le scénario nominal du SàE n'utilise que le SimulateurICSim.

