REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSIEGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIQUE

UNIVERSITE LARBI BEN M'HIDI – OUM EL BOUAGHI

FACULTE DES SCIENCES EXACTES ET DES SCIENCES DE LA NATURE ET DE LA VIE

Département de Mathématiques et Informatique

N° D'ordre :……

Série :…….

**Mémoire pour obtenir le diplôme de magister en Informatique**

**Option : Intelligence Artificielle et Imagerie**

# *Titre : Simulation basée agents des protocoles de communication des systèmes embarqués distribués temps réel pour l'automobile*

***Présenté par : ZAIDI SOFIANE***

***Soutenu le : …./…./2014***

*Devant le jury composé de :*

| | | | |
|---|---|---|---|
| **NINI BRAHIM** | **MCA** | **Université d'Oum El Bouaghi** | **Président** |
| **MOKHATI FARID** | **MCA** | **Université d'Oum El Bouaghi** | **Examinateur** |
| **AMIRAT ABDALKARIM** | **MCA** | **Université de Souk Ahras** | **Examinateur** |
| **BOUTEKKOUK FATEH** | **MCA** | **Université d'Oum El Bouaghi** | **Encadreur** |

*2013/2014*

# *REMERCIMENTS*

*Je tiens à remercier vivement Monsieur F. BOUTEKKOUK ; maitre de Conférences classe'A' à l'université d'oum el bouaghi pour avoir accepté de diriger ce travail ainsi que pour ses encouragements et son aide.*

*Je remercie aussi Monsieur B.NINI; maitre de conférences classe 'A' à l'université d'Oum El Bouaghi pour avoir accepté de présider le jury de magister.*

*Mes remerciements vont aussi à Monsieur F.MOKHATI maitre de conférences classe'A' à l'université d'Oum El Bouaghi ainsi qu'à Monsieur A.AMIRAT maitre de conférences classe'A' à l'université de Souk Ahras pour avoir accepté d'examiner ce travail.*

# *Dédicace*

*A la mémoire du mon oncle ZAIDI FATEH, qui a déployé des efforts dans le domaine de la recherche scientifique jusqu'au dernier jour de sa vie.*

*A mes deux parents, eux qui m'ont offert l'un des plus beaux cadeaux De la vie : le savoir. Je leur dis merci pour tout ce qu'ils ont fait et Continuent à faire pour moi.*

*A mes deux sœurs assia et houda et ses enfants : mehdi, souhaib, amine et salsabil*

*A mes collègues d'étude en magistère : salim, ayoub, fatima, hajer, skender, mouhamed, karima, soumia, ahlem.*

*A mes collègues du travail : azzedine, hellal, abed el rezak, et tous les travailleurs de mon organisme*

*A tous mes amis qui me sont chers*

## Résumé

Dans ce mémoire nous présentons notre approche pour la modélisation et la simulation des services de base des protocoles TTP et FlexRay pour les systèmes embarqués distribués temps réel de l'automobile, tels que l'initialisation du cluster, l'appartenance et l'acquittement, la synchronisation d'horloges, l'envoi et la réception des messages, la détection et le traitement d'erreurs pour le protocole TTP, et le processus de réveil des nœuds du cluster, l'initialisation du cluster, la synchronisation d'horloges, l'envoi et la réception des messages dans les segment statique et dynamique pour le protocole FlexRay. Chaque architecture du protocole TTP ou FlexRay proposée est modélisée à l'aide de la méthodologie O-MaSE sous forme d'un système multi-agent et implémentée sous la plateforme Jade. Nous développons une ontologie Jade et un environnement qui offrent une interprétation automatique des différents champs de la trame (TTP et FlexRay) et de simuler les services de base de ces deux protocoles respectivement.



**Mots Clés :** systèmes embarqués distribués temps réel de l'automobile, protocole TTP, protocole FlexRay, Système Multi-Agents (SMA), Méthodologie O-MaSE, Ontologie, Simulation, Jade.



## Abstract

In this work, we present our agent based approach for design and simulation of the basic services of the TTP and FlexRay Protocols for automotive real time distributed embedded systems, such as the startup, the membership and acknowledgement, the messages sending and reception, synchronization, errors detection and handling for TTP protocol and wakeup, startup, synchronization, the messages sending and reception in the static and dynamic segments for FlexRay protocol. The TTP and FlexRay proposed architectures are modeled as multi-agent systems and implemented in the Jade platform following mainly the so-called O-MaSE methodology. Using Jade, we developed an ontology and an environment that provide an automatic interpretation of frame fields and simulate the services of the two protocols respectively.



**Key Words :** Automotive real time distributed embedded systems, TTP protocol, FlexRay protocol, Multi-Agents System (MAS), O-MaSE methodology, Ontology, Simulation, Jade.



## ملخص

من خلال هاته المذكرة قمنا بعرض اقتراحنا باستخدام الانظمة المتعددة الخدمات لتصميم و محاكات الوظائف الاساسية للبروتوكولين تي تي بي و فلاكس راي المعمول بهما علي مستوي شبكات المعلوماتية المدمجة في السيارة. علي سبيل المثال لهاته الوظائف بداء اشغال الشبكة, توقيت ساعات الوحدات, ارسال واستقبال المعلومات بين وحدات الشبكة... قمنا ببرجمة تصميمنا المقترح بالاستعانة بالجاد التي تسمح لنا بإنشاء انطولوجيا تسمح بالترجمة الالية لمحتوي المعلومات. هاته المذكرة تقدم حالات دراسة لنتائج محاكات وظائف هاذان البروتكولان.



**الكلمات الدالة** : انظمة المعلوماتية المدمجة للسيارة, بورتوكول تي تي بي, بورتوكول فلاكس راي, الانظمة متعددة الخدمات, محاكات, الجاد.

# Table des matières

# Liste des Figures

# Liste des Tableaux

# Introduction Générale

# Introduction Générale

## 1. Contexte du mémoire

L'industrie de l'automobile est aujourd'hui la sixième plus grande économie dans le monde, produisant environ 70 millions de véhicule chaque année. Les technologies actuelles dans le domaine de l'automobile visent à améliorer les fonctionnalités, le confort, la sécurité, et le coût de l'automobile à travers l'intégration des systèmes électroniques et informatiques au sein de l'automobile. Le système embarqué de l'automobile est une architecture distribuée d'unités de contrôle électroniques qui contrôlent les composants mécaniques et hydrauliques de l'automobile (moteur, système de freinage, boite de vitesses…).La recherche scientifique dans le domaine des systèmes embarqués de l'automobile se concentre sur plusieurs axes comme : le système d'exploitation temps réel du système embarqué de l'automobile, l'analyse et l'ordonnancement temps réel, la simulation et la vérification formelle des protocoles de communication.

Les unités de contrôle électroniques du système embarqué de l'automobile sont interconnectées via des réseaux de communication, et utilisent des protocoles de communication standards comme CAN, TTP, FlexRay, TTCAN…

Nous présentons dans notre mémoire une approche basée agents permettant la modélisation et la simulation des différents services et aspects des protocoles de communication TTP et FlexRay au sein du système embarqué de l'automobile. Nous utilisons la méthodologie multi-agent O-MaSE et le langage AUML pour la modélisation de ces protocoles de communication, et la plateforme multi-agent jade pour l'implémentation du modèle conceptuel des protocoles TTP et FlexRay. Nous avons choisi ces deux protocoles d'une part parce que ces deux protocoles sont les plus connus dans le domaine de l'automobile et d'une autre part, leurs services présentés dans les spécifications officielles de deux protocoles sont de haut niveau, donc une simulation multi-agent de ces protocoles est appropriée.

Le protocole de communication TTP est dirigée par le temps, leur services garantissent une communication déterministe et tolérante aux faute comme : startup, synchronisation d'horloges, l'émission et réception d'une trame, appartenance et acquittement…

Le protocole de communication FlexRay est un protocole mixte dirigé par les événements et le temps, ses services permettent de garantir une communication déterministe, flexible et rapide comme : le wakeup, le startup, la synchronisation d'horloges, l'émission d'une trame statique, l'émission d'une trame dynamique …

La simulation de ces deux protocoles permet de bien comprendre la réalité de leurs fonctionnements et ouvre la porte vers d'autres axes de recherche comme la vérification

formelle et les nouvelles technologies offertes par le génie logiciel comme la programmation orientée aspects, les transformations des modèles...

## 2. Problématique et motivation

Les limites des différents travaux de recherche sur la modélisation et la simulation des protocoles de communication TTP et FlexRay nous poussent de proposer notre approche basée agents pour modéliser et de simuler les services de base offerts par les deux protocoles TTP et FlexRay. Nous pouvons résumer ces limites dans les points suivants:

- Chaque travail de recherche se concentre généralement sur la simulation de quelques aspects et/ou services de protocole TTP/FlexRay. Donc il n'y a pas une simulation **complète** qui couvre tous les services de ces protocoles.
- Les approches classiques (fonctionnels, orienté objet…) ne peuvent pas simuler efficacement Les protocole TTP et FlexRay à cause de la complexité de ces protocoles : les services ne sont pas séparés et nécessitent des interactions entre eux pour qu'ils puissent terminer avec succès comme le cas de service de startup et la synchronisation dans les deux protocoles. Donc le recours à une **approche de développement des protocoles complexes** en vu de leurs simulations s'avère très intéressant.
- Les différentes techniques de simulation proposées utilisent des modèles et des outils de simulation, mais ils ne suivent pas généralement une **méthodologie** pour la conception et l'implémentation du simulateur de protocole, ce qui influence sur le facteur de la qualité et de sureté de simulateur implémenté. Donc le recours à l'application d'une méthodologie de modélisation de ces protocoles est motivé.
- Malgré que les services des protocoles TTP et FlexRay sont de haut niveau, les différentes simulations ignorent généralement la conception abstraite des deux protocoles et passent directement à une implémentation de bas niveau. Donc le recours à une modélisation **abstraite** de ces protocoles est motivé.
- Généralement les différents travaux se concentrent sur la simulation d'un seul protocole malgré que le système embarqué de l'automobile puisse supporter une variété des protocoles de communication. Donc le recours à modélisation **générique** de ces deux protocoles est motivé.

Donc nous avons besoin d'une modélisation abstraite, expressive et assez générique que possible qui s'appuie sur un paradigme pouvant capturer tous les aspects et les spécificités des systèmes embarqués distribués pour l'automobile tels que l'autonomie, l'intelligence, l'adaptabilité, la complexité etc. en suivant une méthodologie de développement des systèmes complexes qui supporte bien des outils d'implémentation appropriés au paradigme choisi. Pour répondre à ces besoins, nous proposons d'utiliser le paradigme agent, la méthodologie O-MaSE, le langage AUML, et la plateforme

d'implémentation Jade pour simuler les différents services de deux protocoles TTP et FlexRay.

## 3. Organisation du mémoire

Notre mémoire se compose de six chapitres organisés de la façon suivante :

- Le premier chapitre présente le système embarqué de l'automobile et les différents protocoles de communications au sein de ce système.
- Le deuxième chapitre met l'accent sur les deux protocoles de communication TTP et FlexRay et ses services.
- Le troisième chapitre est consacré à présenter un état de l'art sur les différentes approches de modélisation et de simulation des protocoles de communication TTP et FlexRay.
- Le quatrième chapitre présente la méthodologie de la conception des systèmes multi-agents O-MaSE, le langage AUML et la plateforme Jade.
- Le cinquième chapitre détaille notre modélisation proposée des services des protocoles TTP et FlexRay à base de la méthodologie O-MaSE et le langage AUML.
- Le dernier chapitre présente une implémentation de modèle conceptuel du protocole FlexRay avec la plateforme multi-agents Jade et discute quelques résultats de la simulation.

Le présent mémoire se termine par une conclusion générale et les perspectives envisageables.

# Chapitre I

# Systèmes embarqués pour l'automobile

## 1. Introduction

La tendance actuelle dans le domaine de l'automobile est de remplacer les parties mécaniques de l'automobile par des parties électroniques embarquées ; "les applications drive-by-wire", qui sont moins couteuses et plus sécurisées. Les systèmes embarqués de l'automobile sont implémentés pour garantir les facteurs de confort et de sécurité.

Le réseau embarqué de l'automobile est constitué d'un ensemble d'unités de contrôle électroniques (ECU), chacune d'entre elles est responsable de la gestion d'un équipement de l'automobile, comme ABS (Anti-lock braking system) qui contrôle le système de freinage classique de l'automobile, ECM (Engine Control Module) qui contrôle le moteur de l'automobile, ESP (Electronic Stability Program) qui contrôle la trajectoire de l'automobile…

Les unités de contrôle électroniques de l'automobile communiquent entre eux grâce à des réseaux de communication. Chaque réseau possède un protocole de communication qui spécifie les services et les règles de communication. Les protocoles de communication sont classés en trois groupes selon le paradigme d'accès au support de communication : les protocoles orientés événements (Event Triggered Protocols) comme CAN (Controller Area Network), les protocoles orientés temps (Time Triggered Protocols) comme TTP (Time Triggered Protocol), et les protocoles mixtes comme TT-CAN (Time-Triggered CAN) et FlexRay.

Les systèmes embarqués de l'automobile peuvent être divisés en quatre domaines principaux : « contrôle moteur », « châssis », « habitacle », « télématique et multimédia ». Ces domaines n'imposent pas les mêmes propriétés de performances et de sûreté [1]. Par exemple pour les applications critiques qui nécessitent une sûreté de fonctionnement et un comportement déterministe qui dépend du temps, les protocoles les plus appropriés pour ce type d'applications sont ceux orientés temps, en revanche, pour les applications nécessitant un comportement flexible qui dépend de l'occurrence des évènements, les protocoles les plus appropriés pour ce type d'applications sont ceux orientés évènement.

En 1993, la SAE (Society of Automotive Engineers) a classifié les réseaux de communication embarqués de l'automobile selon le débit de transmission offert en quatre classes : classe A, classe B, classe C et classe D [2].

- **Classe A :** Les réseaux de cette classe offrent un débit inférieur à 10 kb/s ; ces réseaux sont utilisés au sein de l'automobile par exemple pour les fonctions de contrôle (Contrôle d'allumage des feux, ouverture de coffre…). Le support de communication de ces réseaux généralement est un fil.
- **Classe B :** Les réseaux de cette classe offrent un débit entre 10kb/s et 125kb/s ; ces réseaux sont utilisés au sein de l'automobile par exemple pour le transfert général d'information (Tableau de bord). Le support de communication de ces réseaux généralement est une paire torsadée.
- **Classe C :** Les réseaux de cette classe offrent un débit entre 125kb/s et 1Mb/s ; ces réseaux sont utilisés au sein de l'automobile par exemple pour le contrôle en temps réel

(Contrôle Moteur, Airbag …). Le support de communication de ces réseaux généralement est un câble coaxial.

- **Classe D :** les réseaux de cette classe offrent un débit supérieur à 1Mb/s ; ces réseaux sont utilisés au sein de l'automobile par exemple pour les applications multimédia (Internet, vidéo application,…). Ces réseaux utilisent par exemple la fibre optique comme un support de communication.

## 2. Définitions

### 2.1. Système

Joël de Rosnay (1975) définit un Système comme un ensemble d'éléments en interaction, organisés en fonction d'un but déterminé.

### 2.2. Système temps réel

Un système temps réel est un système informatique dont son fonctionnement est conditionné par l'évolution dynamique de l'état de l'environnement qui lui est connecté et dont il doit suivre ou piloter cet environnement imposé en respectant les contraintes de temps [3].

### 2.3. Système distribué

Selon Tanenbaum [4], un système distribué est un ensemble d'ordinateurs indépendants qui apparait pour ses utilisateurs comme un seul système cohérent. [5] définit un système distribué comme un réseau d'entités calculantes ayants le même but commun : celui de la réalisation d'une tâche globale à laquelle chaque entité contribue par ses calculs locaux et les communications qu'elle entreprend, sans même qu'elle ait connaissance du dessein global du système.

### 2.4. Système embarqué

Un système embarqué peut être défini comme une combinaison autonome de matériel et logiciel (électronique et informatique) qui ne possède pas des E/S standards comme un clavier ou un écran d'ordinateur et qui est dédié à réaliser une tâche bien précise en interaction avec son environnement et en respectant des contraintes telles que : la consommation d'énergie, le poids, la fiabilité, le temps réel, et le coût [6].

## 3. Le Système embarqué de base de l'automobile

Le système Electronique de base de l'automobile fonctionne selon le principe ETS (Entrées, Traitement, Sortie). Les entrées sont des capteurs appelés aussi des générateurs des signaux, les sondes ou transducteurs de mesure. Le traitement est réalisé à l'aide de l'unité de contrôle électronique. Les sorties sont les actionneurs.

### 3.1. Capteur

Est un dispositif qui mesure une grandeur physique observée (Température, Pression, Accélération …) en la convertissant en un signal électrique. Exemples des Capteurs de l'automobile : MAP (Manifold Absolute Pressure) : Capteur de pression d'admission du

moteur, CLT (Coolant Temperature) : Capteur température du liquide de refroidissement du moteur …

### 3.2. Actionneur

Est un dispositif qui transforme un signal électrique fourni par l'unité de contrôle électronique en un phénomène physique, par exemple des actionneurs connectées au moteur de l'automobile : les injecteurs, bobines, papillon de gaz, moto ventilateur …

### 3.3. Unité de Contrôle Electronique (ECU)

Appelée aussi ordinateur de contrôle, est un dispositif électronique qui exécute un ensemble des tâches de contrôle spécifiques au niveau applicatif, par exemple le contrôle de l'injection de carburant, échantillonnage de la vitesse de rotation d'une roue…, . Dans une automobile il existe plusieurs ECUs, chacune a des tâches dédiées. Ces ECUs traitent des données qui proviennent des différents capteurs de l'automobile et contrôlent les différents actionneurs [7]. Une ECU est divisée en parties principales suivantes :

- Alimentation (Power supply) (elle consomme 12V auprès de la batterie de l'automobile)
- Mémoires RAM/ROM
- Des Interfaces de communication au réseau
- Processeurs (CPU)

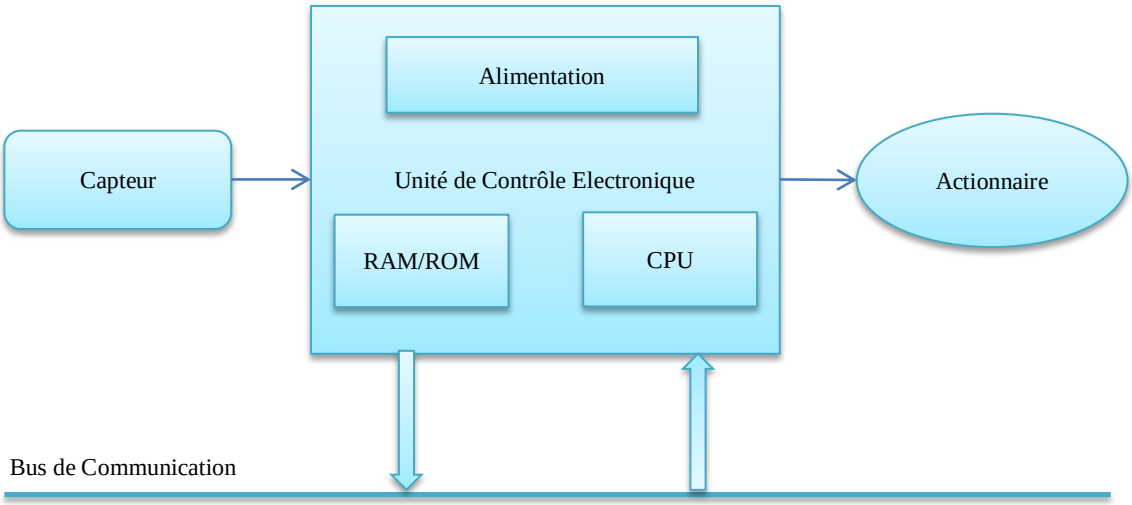


Fig. I-1 : Le système embarqué de base de l'automobile

Il existe deux types d'ECU : monoprocesseur et multiprocesseurs. L'ECU monoprocesseur est utilisé par exemple dans le domaine de confort puisqu'il ne nécessite pas un niveau élevé de la sûreté et fiabilité, par contre les domaines de power train et châssis utilisent des ECUs multiprocesseurs puisqu'ils ont besoin d'un niveau élevé de sûreté et de fiabilité.

## 4. L'architecture embarquée de l'automobile

Le système embarqué de l'automobile comporte plusieurs calculateurs (ECU), distribués dans tous le volume de véhicule et communicants entre eux via plusieurs réseaux de communications [8]. Il existe deux parties de l'architecture électronique de l'automobile :

### 4.1. Architecture matérielle

Inclut les unités de contrôle électroniques ECUs, les processeurs, les capteurs, les actionnaires, les bus de communications, et la topologie d'interconnexion des ECUs.

### 4.2. Architecture logicielle

Inclut les spécifications logicielles standards des composants, l'architecture en couches des ECUs, les interfaces entre les couches, les protocoles de communication, et les fonctionnalités logicielles des composants. L'implémentation de l'automobile est réalisée d'une maniéré distribuée : distribution des ECUs, des capteurs, des actionnaires, et des bus de communication.

## 5. Les types d'architectures embarqués de l'automobile

### 5.1. Architecture centralisée

Elle est composée d'une seule ECU qui est responsable de toutes les fonctionnalités, et qui peut être monoprocesseur ou multiprocesseurs à mémoire partagée, et un ensemble de capteurs et actionneurs reliés à cet ECU [9].

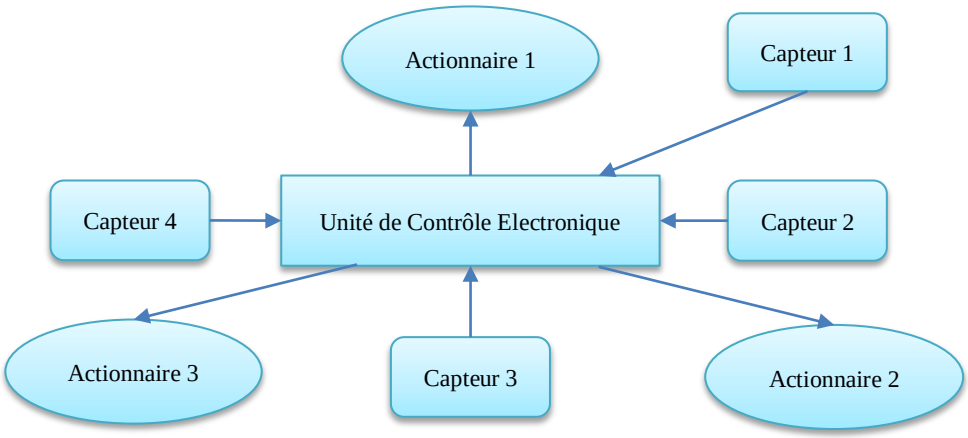


Fig. I-2 : L'architecture centralisée du système embarqué de l'automobile

### 5.2. Architecture multi-calculateurs

Elle est composée d'un ensemble des ECUs non reliées. Chaque ECU est responsable d'un ensemble de fonctionnalités. Cette architecture permet de réduire le câblage et augmenter la fiabilité du système : si une ECU tombe en panne, les autres ECUs continuent leur fonctionnement normalement. Le problème de cette architecture est le risque de l'incohérence de comportement global de l'ensemble des ECUs à cause de l'absence de la communication entre elles.

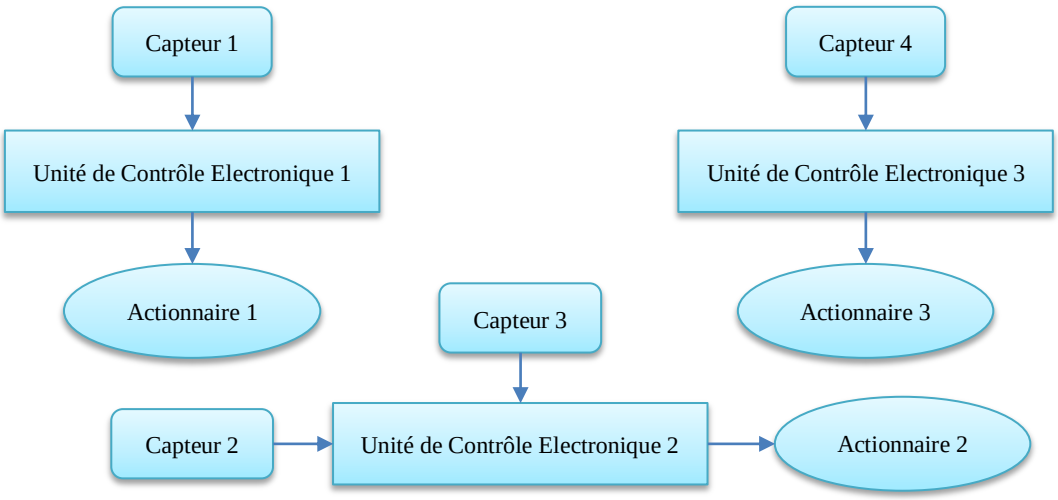


Fig. I-3 : L'architecture multi-calculateurs du système embarqué de l'automobile

### 5.3. Architecture faiblement distribuée

Elle est composée d'un ensemble de ECUs reliées entre elles via un réseau de communication et qui sont développées par des constructeurs d'automobiles tels que : CAN par Bosh, VAN (Vehicle Area Network) par PSA et Renaut, A-Bus par Volkswagen, K-Bus par BMW et bien d'autres encore. Les avantages de cette architecture est la possibilité de partager des données et des capteurs ; la répartition de la même fonction entre plusieurs ECUs et le contrôle du comportement global du système.

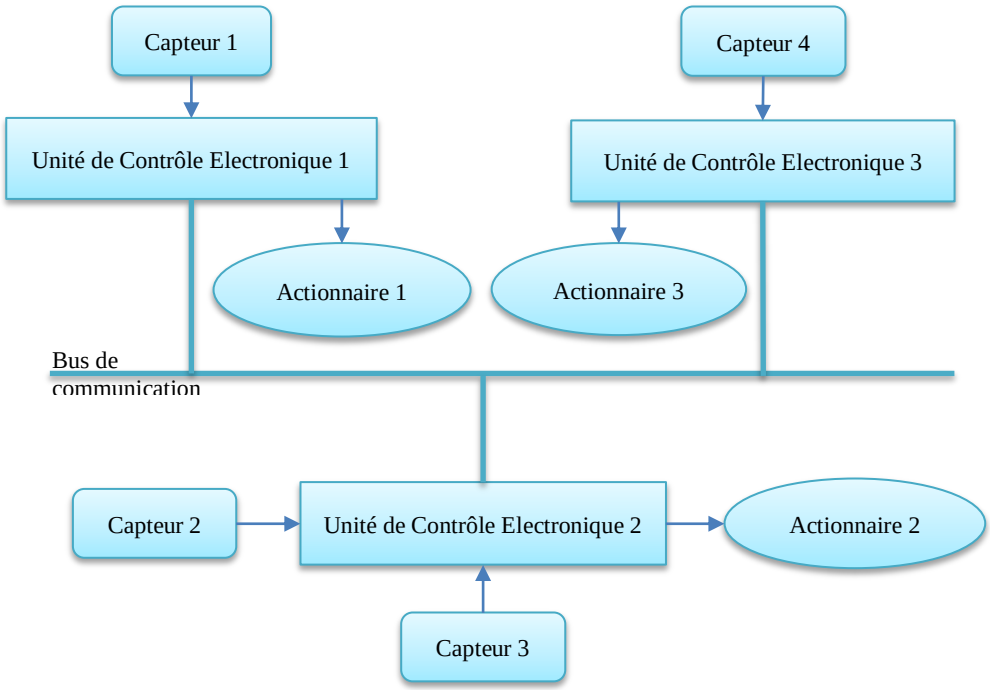


Fig. I-4 : L'architecture faiblement distribuée du système embarqué de l'automobile

### 5.4. Architecture fortement distribuée

Les capteurs et les actionneurs dans cette architecture peuvent directement être connectés sur le bus de communication. Cette architecture permet de réduire le câblage.

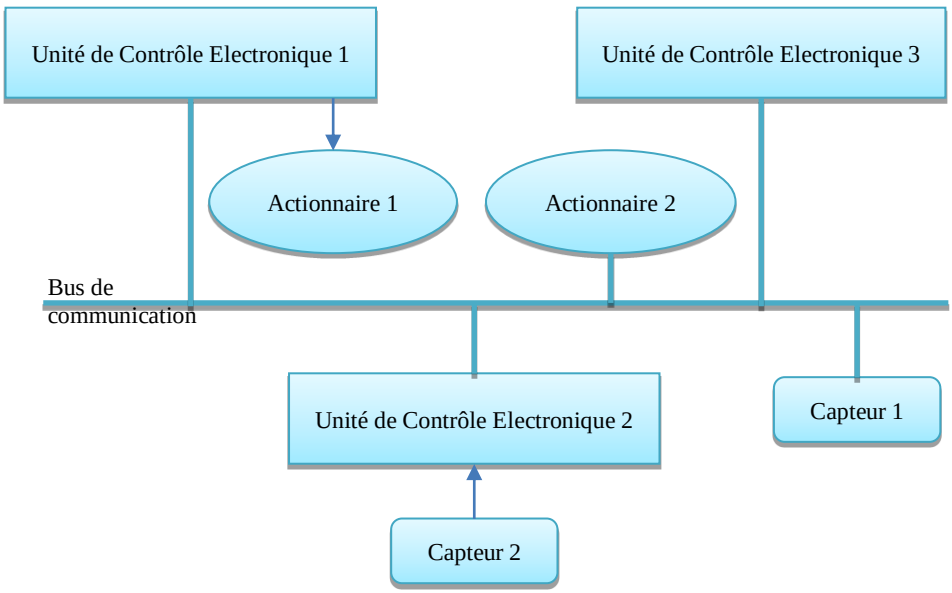


Fig. I-5 : L'architecture fortement distribuée du système embarqué de l'automobile

## 6. Les domaines des systèmes embarqués de l'automobile

Les systèmes embarqués de l'automobile sont divisés en quatre domaines principaux en suivant un ensemble des critères tels que l'architecture, services, et contraintes [1] :

### 6.1. Moto-propulseur (Power Train)

Il englobe deux fonctionnalités : le contrôle du moteur et la transmission de l'état de véhicule en fonction des demandes de conducteur.

### 6.2. Châssis (Chassis)

C'est le domaine de la sécurité de véhicule, il s'occupe de contrôle de la suspension, du guidage et de freinage. Il est responsable de contrôler des équipements comme ASC (Automatic Stability Control), ABS, ESP et 4WD (Four Wheel Drive)…

Les domaines de Moto-Propulseur et Châssis regroupent des applications critiques qui sont soumises à des contraintes temporelles strictes.

### 6.3. Habitacle (Body)

Il regroupe les applications de l'affichage de bord, le contrôle des essuie-glaces, les phares, les portières, les vitres, la climatisation…

### 6.4. Télématique et multimédia

Il regroupe des applications telles que : la radio, la lecture CD, aide à la navigation, les jeux, le télé-diagnostic de l'automobile …

Fig. I-6 montre une partie de l'architecture générale du système embarqué de l'automobile Mercedes E-class BR211 [7], où la topologie de l'automobile comporte plus de 50 ECUs divisées en 4 domaines de l'automobile : Mon-Propulseur (Power train), Châssis (Chassis), Habitacle(Body), Télématique (Telematics). Chaque domaine utilise un réseau de communication approprié.

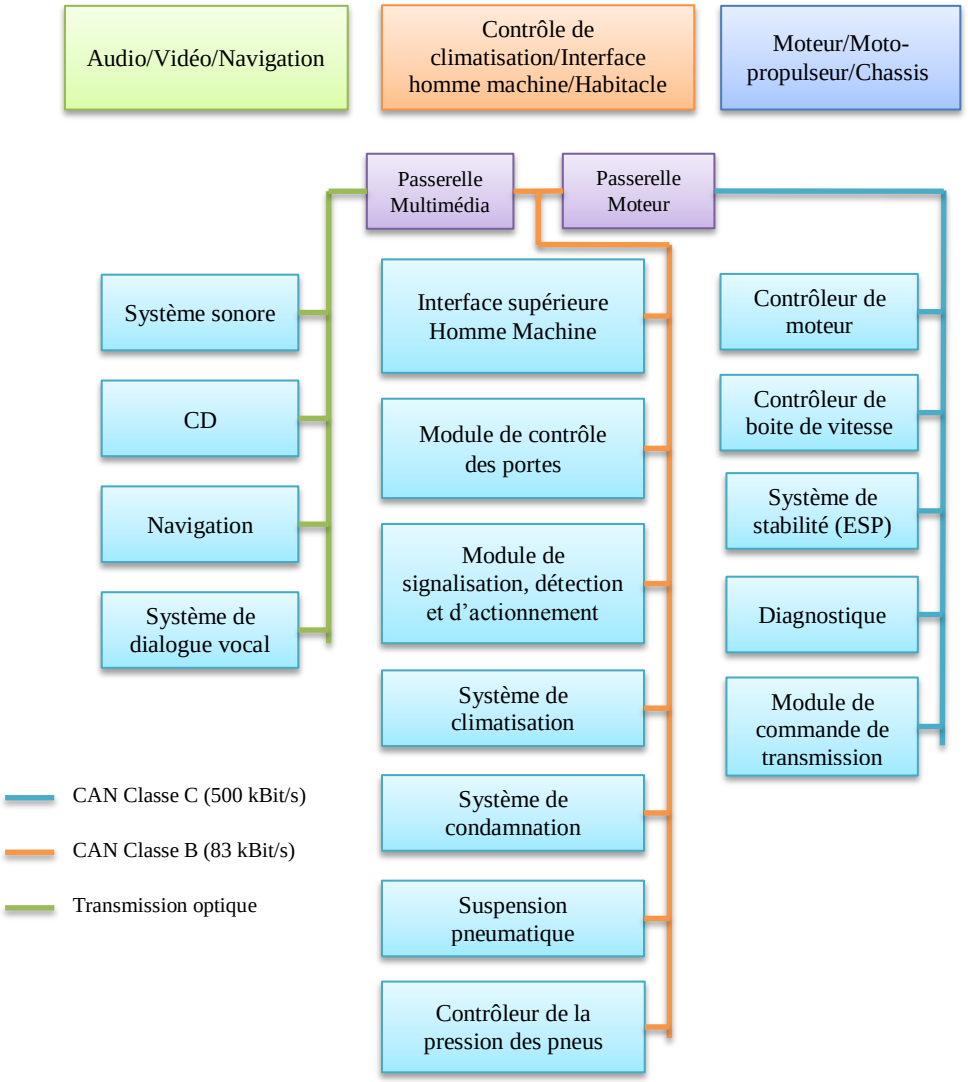


Fig. I-6 : L'architecture générale de l'automobile Mercedes E-class BR211

## 7. Les réseaux de communication au sein du système embarqué de l'automobile

Les réseaux embarqués de l'automobile relèvent de trois paradigmes de communication principaux : leur comportement est dirigé par les évènements ou par le temps ou bien mixtes (par événement et temps) [1].

### 7.1. Réseaux à accès guidé par les événements (Event Triggered)

Un comportement est guidé par les événements signifie que les messages sont transmis au contrôleur de communication à l'occurrence d'un événement reconnu comme par exemple, la fermeture d'une portière, la demande d'appel de phare… etc. les réseaux basés sur ce paradigme offrent une communication flexible, une bonne utilisation de la bande passante et ils sont peu chères en terme du coût. Le réseau CAN s'avère particulièrement représentatif des réseaux à comportement guidé par les événements.

**CAN**

CAN est un réseau de communication série qui a été conçu par la société allemande Robert BOSCH en 1983 [10] et standardisé par ISO (International Organization for Standards) dans les normes : 11898 pour les applications à haut débit (jusqu'à 1Mb/s) et

11519 pour les applications à bas débit (jusqu'à 125kb/s). Le réseau CAN est utilisé pour l'interconnexion de plusieurs nœuds, chaque nœud est responsable d'une tâche par exemple : le contrôle du moteur, la suspension, l'anti blocage de frein, contrôler la transmission, commande feux, climatisation. Le protocole CAN est orienté évènements. Si à un instant donné plusieurs évènements arrivent en même temps (C'est-à-dire plusieurs nœuds veulent transmettre des trames sur le bus partagé), seul le nœud le plus prioritaire peut utiliser le bus pour transmettre sa trame. Le nœud le plus prioritaire est le nœud qui possède la trame ayant la valeur de l'identificateur la plus minimale par rapport à toutes les trames des autres nœuds en émission. La technique de sélection du nœud le plus prioritaire est appelée le routage bit à bit (non destructif).

### 7.2. Réseaux à accès guidé par le temps (Time Triggered)

Un comportement est guidé par le temps signifie que les messages sont transmis au contrôleur de communication au sein des trames sur le réseau et toute trame est émise dans un intervalle de temps prédéfini appelé slot. Les réseaux basés sur ce paradigme offrent une communication fiable, déterministe et une possibilité de prévisibilité et validation temporelle. Le réseau TTP s'avère particulièrement représentatif des réseaux à comportement guidé par le temps.

**TTP**

TTP est le protocole de communication de l'architecture TTA qui a été développé par l'université de Vienna pour rependre aux besoins de la sûreté, le déterminisme, la fiabilité et la tolérance aux fautes de la communication. TTP est strictement guidé par le temps, et inclut des services qui garantissent la tolérance aux fautes tels que : l'initialisation du réseau, l'appartenance (membership) et l'acquittement, la synchronisation d'horloges et la redondance.

### 7.3. Réseaux à accès guidé par les événements et le temps

Ces réseaux combinent les deux paradigmes de communication par évènement et par temps ; ces réseaux offrent les avantages de ces deux paradigmes en termes de flexibilité et de déterminisme. Les protocoles mixtes font une séparation temporelle entre l'accès orienté par le temps (TT) et celui orienté par les événements (ET) généralement en deux segments d'une manière périodique (cyclique): segment statique et segment dynamique [11]. Les réseaux TTCAN et FlexRay sont des exemples des réseaux à comportement guidé par le temps et les événements.

**TTCAN**

TTCAN est une extension de protocole CAN qui offre le déterminisme. TTCAN est défini par ISO 11898-4 [12] qui rajoute la couche session par rapport au CAN. TTCAN est un protocole synchrone où la transmission des trames est réalisée en fonction de temps par rapport au CAN qui est asynchrone.

# Première Partie du Chapitre II

# Le protocole de communication TTP

## 1. L'architecture TTA

Le réseau TTA est composé d'un ou plusieurs clusters ; chaque cluster est composé d'un ou plusieurs nœuds ; chaque nœud du cluster est indépendant et interconnecté aux autres nœuds à travers un réseau de communication dupliqué [15].

Un nœud est appelé SRU (Smallest Replaceable Unit). Chaque nœud TTA est une entité autonome qui exécute une partie de l'application distribuée et il échange des messages avec les autres nœuds. La communication entre les clusters est réalisée à travers une passerelle (Gateway) (voir Fig. II-1-1).

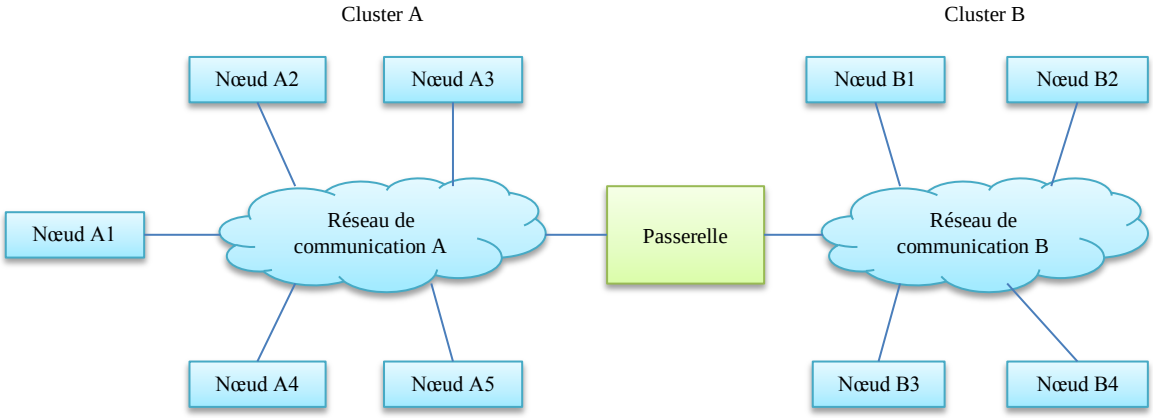


Fig. II-1-1 : Exemple d'une architecture TTA

### 1.1. Topologies d'un réseau TTA

Il existe deux versions de TTA selon la topologie [16] :

- **TTA-bus:** le réseau de communication est composé de deux bus répliqués.
- **TTA-star:** le réseau de communication est composé de deux topologies en étoile, qui contiennent deux concentrateurs répliqués.

### 1.2. La différence entre TTA-bus et TTA-star

- **Le nombre de Bus Guardian**

  En TTA-bus, chaque nœud a besoin de deux bus gardians ; chacun est responsable de la protection d'accès à l'un des deux bus répliqués, par contre en TTA-star le nombre de bus gardians sur le réseau est égal à deux (au niveau des deux concentrateurs) ; chacun est responsable de la protection d'accès au réseau de tous les nœuds , ce qui implique que le coût des bus Guardian en TTA-bus est plus élevé que celui de bus-star.

- **Le mode de diffusion**

  En TTA-bus le mode de diffusion est multidiffusions ; c'est-à-dire le message émis sur les bus doit être reçus par tous les nœuds du réseau, par contre en TTA-star le mode est point à point ; c'est à dire que le message émis est reçus par les concentrateurs ensuite il est dirigé vers seulement les nœuds récepteurs intéressés par le message.

- **Résistance aux pannes de support de communication**

  Si une panne touche une zone contenant les deux bus de communication, la topologie en bus ne peut plus assurer la transmission de trames (paralysie du système), par

contre dans la topologie en étoile la panne touche seulement une liaison point à point (entre les concentrateurs et un nœud) sans paralyser complètement le système.
Les bus gardiens de TTA-star peuvent offrir des services supplémentaires qui permettent essentiellement d'assurer la tolérance aux fautes.

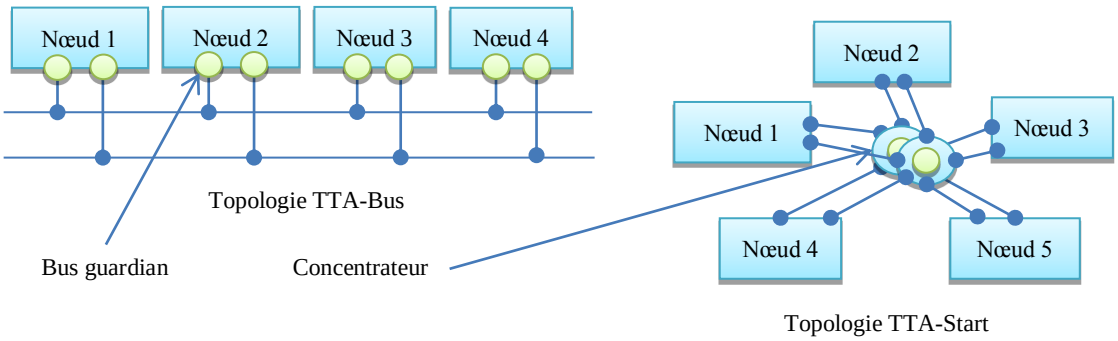


Fig. II-1-2 : Topologies de l'architecture TTA

### 1.3. La technologie orientée temps (Time Triggered)

L'architecture TTA est basée sur la technologie orientée temps au niveau de :

- **La couche système d'exploitation temps réel**
  Le déclenchement de l'exécution des tâches applicatives aux nœuds est réalisé en fonction de l'évolution du temps.
- **La couche transmission**
  Le déclenchement de la transmission des messages des nœuds est réalisé en fonction de l'évolution de temps.

## 2. L'architecture d'un nœud TTA

Un nœud TTA est un module électronique qui est composé de deux parties : Hôte et Communication.

### 2.1. La partie hôte

Elle se compose de deux Couches :

#### 2.1.1. Couche application

Le rôle de cette couche est d'exécuter les tâches temps réel distribuées.

#### 2.1.2. Couche système d'exploitation temps réel

Cette couche est basée sur la norme OSEK Time, elle joue deux rôles :

- L'ordonnancement des tâches à la base d'une table globale statique d'ordonnancement prédéfini : TADL (TAsk Descriptor List). Les tâches ne doivent pas être bloquantes, selon la norme OSEKTime, l'ordonnancement des tâches peut être préemptif à condition que le temps de préemption d'une tâche ne doit pas dépasser sa valeur prévue dans la table TADL. Le système d'exploitation gère aussi les interruptions en cas d'urgence.
- La gestion des fonctions de la couche FT.

### 2.2. La partie communication

Elle se compose d'une (ou deux) couches :

#### 2.2.1. Couche FT (Fault Tolerant layer)

C'est une couche optionnelle spécifique ; son rôle est la gestion de la redondance des applications par duplication selon un taux qui dépend de la nature de l'application et cela pour permettre la tolérance aux fautes.

#### 2.2.2. Couche de transmission

Assure la transmission des messages sur le bus ; le protocole de la transmission est appelé TTP. Il existe deux types de ce protocole : TTP/C et TTP/A. Cette couche contient les parties suivantes : Noyau de protocole (protocol processor), MEDL (Message Descriptor List) et un équipement indépendant qui s'appelle le Bus Guardian.

La Couche FT est séparée de la partie Hôte par l'interface FTCNI (Fault Tolerant Controller Network Interface) qui représente une mémoire partagée. La Couche de Transmission est séparée de la couche FT par l'interface CNI (Controller Network Interface).

Le transfert des messages entre l'interface FTCNI et CNI est assuré par la couche FT ; le passage d'un message de l'interface FTCNI vers l'interface CNI s'effectue sans transformation, par contre le passage inverse d'un message nécessite une gestion de la redondance au niveau de la couche FT puisqu'elle reçoit plusieurs copies d'un même message à partir de l'interface CNI et dépose une seule copie de ce message dans l'interface FTCNI.

L'interface entre la partie communication et le support de communication est l'interface LLI (logical line interface) [17].

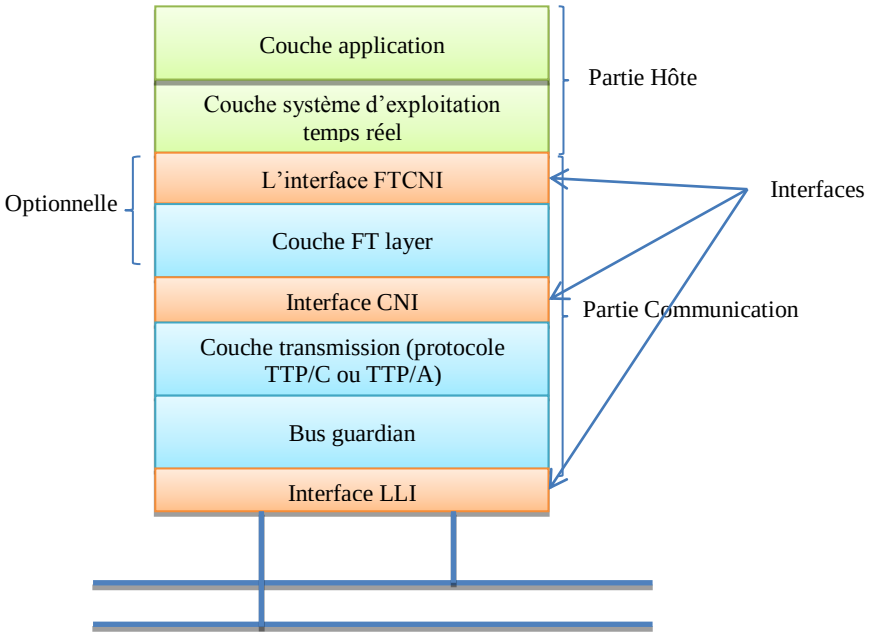


Fig. II-1-3: L'architecture d'un nœud TTA

Le bus Guardian est un composant physique autonome (possède une horloge et une alimentation séparée) qui contrôle et autorise l'accès au bus suivant le plan statique

temporel prédéfini. Le coût de bus Guardian dépend de son taux d'indépendance et de sa complexité.

### 3. Les Types de trames du protocole TTP/C

Selon le Protocole TTP/C de réseau TTA, il existe trois types de trames [18]:

- **Trame d'initialisation (coldstart):** cette trame est utilisée pour l'initialisation du système, elle contient les informations globales du système : le temps global, le slot courant …
- **Trame à CState implicite (trame normale):** cette trame est constituée de données de la couche application et des informations de contrôle liées au protocole : le type de la trame, CRC … Le CState de nœud émetteur n'est pas inclus dans la trame, mais il est inclus dans le calcul de CRC de cette trame.
- **Trame à CState explicite :** elle est identique à la trame à CState implicite, sauf que cette trame contienne en plus le CState de nœud émetteur ; ce CState est présent dans le calcul de CRC de cette trame.

### 4. Les services de protocole TTP/C

#### 4.1. Contrôle d'accès au support de communication

La transmission des messages par les nœuds est réalisée à l'aide de la technologie orientée temps selon une stratégie appelée *TDMA (Time Division Multiple Access).* Le principe de cette technologie est de partager le temps global prévu en un ensemble des slots. *Un slot* est une période de temps fixe, à chaque nœud du réseau il est attribué un slot pendant lequel il pourra émettre des messages. *Un round* est un regroupement des slots, où chaque nœud de réseau ayant le droit d'émettre pendant un seul slot au maximum ; le round se répète de manière *cyclique.*

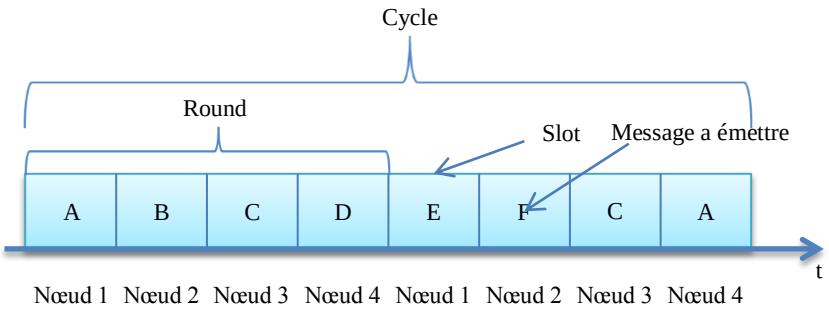


Fig. II-1-4 : Exemple d'une communication TDMA

L'ordonnancement statique d'accès des nœuds au réseau de communication est réalisé selon une table globale statique prédéfinie qui s'appelle *MEDL (Message Descriptor List).* Cette table sauvegarde toutes les informations globales de transmission des massages ; par exemple elle sauvegarde pour chaque message : le nœud émetteur de ce message, l'instant de l'émission, la période d'émission …, chaque nœud possède une copie identique de cette table. Lors de la réception, les nœuds récepteurs savent bien le type de message et son émetteur grâce à la table MEDL alors ce n'est pas la peine d'identifier le message. Le protocole TTP/C est basé

également sur une autre structure de données qui permet de donner une vue globale commune du système : c'est le CState. Les informations stockées dans CState sont :

- Le mode courant de fonctionnement du système.
- L'existence d'une requête de changement de mode en attente.
- Le slot courant.
- La valeur de temps global.
- L'ensemble des nœuds actifs du système (Appartenance).

Chaque nœud possède son propre CState qui doit être la même pour tous les nœuds du système.

- **Le slot :**

  Le slot est découpé en quatre phases :

  - **La phase de transmission TP (Transmission Phase) :** c'est la période de la transmission effective de la trame.
  - **La phase de poste-réception (Post-Receive Phase) :** c'est la période de la réception de la trame (il existe des algorithmes liés à la réception de la trame qui nécessitent une période de temps).
  - **La phase de repos (Idle) :** c'est une période durant laquelle le contrôleur doit attendre.
  - **La phase de pré-émission (Pre-Send Phase):** c'est la période de vérification et de préparation de l'émission d'une trame pendant le slot prochain.

### 4.2. Le service d'initialisation (Startup)

Le processus d'initialisation du système se compose de trois phases : intégration, coldstart, et synchronisation (Fig. II-1-5 présente le service startup) [19] :

- **Intégration**

  Après le démarrage du système, chaque nœud initialisé entre dans la phase d'intégration. Chaque nœud possède une copie de la table TDMA, où il est attribué d'un slot dans chaque round pendant lequel il peut transmettre ces données. Le nœud écoute pendant une durée le réseau de communication ; s'il reçoit une séquence de trames (des trames normales non pas d'initialisation) suffisantes qui indiquent qu'il existe un nombre suffisant des nœuds synchronisés, alors le nœud passe à la phase synchronisation, sinon il passe à la phase coldstart.

- **ColdStart**

  Dans cette phase le nœud attend pendant une durée de temps, la réception d'un nombre suffisants des trames de type coldstart ; s'il reçoit ces messages il passe à la phase de synchronisation, sinon il envoie une trame coldstart sur le réseau de communication. Cette phase se termine avec succès lorsqu'un ensemble suffisant des nœuds sont synchronisés. Dans ce cas le nœud passe à la phase de synchronisation, sinon elle termine avec échec lorsque le nombre des nœuds synchronisés est insuffisant et repasse à la phase d'intégration.

- **Synchronisation**

  Durant cette phase le nœud ajuste son état local (CState) avec le CState de la trame reçue. Un nœud dans cette phase peut repasser à la phase d'intégration lorsqu'il perd sa synchronisation.

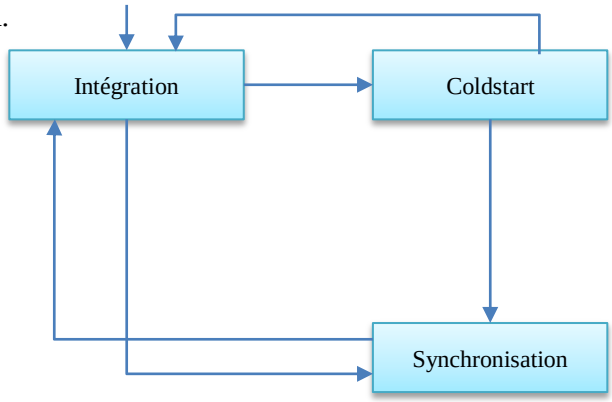


Fig. II-1-5 : Service Startup de protocole TTP

### 4.3. Le service d'intégration d'un nœud au réseau

Le mécanisme de réinitialisation permet l'intégration d'un nœud externe dans un cluster qui est déjà en fonctionnement. Cette intégration s'effectuera sur une trame à CState explicite émise par l'un des nœuds du cluster ; cette trame permet à ce nouveau nœud intégré de récupérer l'état global du système et de se synchroniser avec les autres nœuds (même horloge) dans le même mode de fonctionnement. Le mécanisme de réintégration permet à un nœud exclu (par exemple à cause d'une défaillance au niveau du nœud) du cluster de réintégrer à ce cluster. Cette réintégration s'effectuera lorsque le nœud exclu (dans l'état passif) reçoit un nombre de trames correctes supérieur à une valeur MIC (Minimum Integration Count).

### 4.4. Le service de synchronisation d'horloge

Permet l'établissement du temps global. Lorsqu'un nœud reçoit une trame normale il met à jour son CState local ce qui permet de partager le même point de vue sur le temps global avec le nœud émetteur.

### 4.5. Les services de la tolérance aux fautes

La tolérance aux fautes est un service de protocole TTP/C qui représente l'assurance de fonctionnement correcte du système même s'il existe des nœuds fautifs.

#### 4.5.1. Hypothèses des fautes

Les hypothèses des fautes de protocole TTP/C sont des informations sur les fautes tolérées par le protocole TTP/C comme : le type, la fréquence, le nombre de fautes et les composants touchés par les fautes. La gestion des fautes non inclues dans les hypothèses est gérée par la couche application.

#### 4.5.2. Types des fautes tolérées

- **En fonction de leurs comportements**

  Il existe trois types des fautes tolérées par le protocole TTP/C, en fonction de leurs effets sur le système:

- **Fautes manifestes :** sont les fautes les plus simples à détecter, par exemple un SRU qui n'émet pas.
- **Fautes symétriques :** sont les fautes où leurs effets sont les mêmes surtouts les éléments récepteurs du système.
- **Fautes arbitraires :** en particulier les fautes asymétriques ou byzantines sont des fautes où leurs effets ne sont pas les mêmes surtouts les éléments récepteurs du système.

- **En fonction de domaine affecté**

  Il existe trois types de fautes tolérées par le protocole TTP/C, en fonction de domaine affecté par ces fautes :

  - **Faute des valeurs :** elle affecte les données d'une trame.
  - **Faute de temps :** elle représente une transmission, une réception ou un calcul d'une donnée au mauvais temps.
  - **Faute de l'espace :** elle touche une zone physique du système.

  Parmi les hypothèses de fautes, on trouve l'identification des FCUs (Fault Containment Unit). Les FCUs sont des ensembles des éléments devant être affectés par des fautes d'une manière indépendante sans affecter le reste du système.

### 4.5.3. Solutions de la tolérance aux fautes

Il existe deux solutions pour la tolérance aux fautes :

#### 4.5.3.1. Solution matérielle

Le bus gardien permet de résoudre le problème de babbling idiot. Ce problème se rencontre quand un nœud fautif accède au bus durant une période qui n'est pas lui concernée ce qui implique un conflit d'accès au bus. La résolution de ce problème propose que le bus gardian de ce nœud fautif n'autorise pas l'accès au bus dans une période qui n'est lui pas concernée (l'horloge de bus gardian est séparée de l'horloge de leur nœuds).

#### 4.5.3.2. Solution logicielle

Cette solution permet la détection d'erreur de transmission (CRC) et de détecter les pannes des nœuds (membership et clique avoidance).

##### 4.5.3.2.1. Calcul de CRC

L'hors de la réception d'une trame, le récepteur calcule le CRC (c'est un code détecteur d'erreurs) de la trame reçue pour vérifier que la trame soit correcte ou erronée. Il existe deux types de calcul de CRC :

- **Si la trame est à CState explicite :** le CRC est calculé au niveau de la trame reçue totale.
- **Si la trame est à CState implicite :** le CRC s'est calculé au niveau de la trame reçue en plus le CState de récepteur puisque le CRC qui est inclut dans la trame s'est calculé au niveau de la trame en plus le CState de l'émetteur.

Dans les deux cas de calcul de CRC, l'identifiant de MEDL en cours est inclut dans le calcul de CRC afin de vérifier que l'émetteur et le récepteur se trouvent en même mode de

fonctionnement (puisque chaque mode de fonctionnement ayant son propre ordonnancement alors son propre MEDL).

Après le calcul de CRC au niveau de récepteur, une comparaison de ce CRC avec le CRC qui a été inclut dans la trame calculée par l'émetteur est effectuée pour vérifier si la trame est correcte ou non.

Un test négatif de CRC indique soit que la trame est erronée soit que l'émetteur et le récepteur n'ayant pas la même vue globale du système (CStates différents) ; dans les deux cas la trame sera rejetée.

#### 4.5.3.2.2. Le service d'appartenance(Membership) et d'acquittement

L'algorithme d'acquittement du protocole TTP s'effectue en deux phases à la base d'une série de comparaison de CRC de la trame reçue (qui contient le vecteur d'appartenance de l'émetteur) avec celui de la même trame contenant le vecteur d'appartenance de récepteur (l'objectif est de comparer le vecteur d'appartenance de l'émetteur de la trame avec celui de récepteur de la trame). Le vecteur d'appartenance du nœud récepteur peut se modifier en fonction de résultat de l'algorithme d'acquittement. (Fig. II-1-6 présente l'algorithme complet de service d'appartenance) [16].

- **La première phase d'acquittement**
  Après l'émission d'une trame, le nœud émetteur *A* tente d'acquitter sa trame émise ; il attend la réception d'une trame depuis son premier successeur. Lorsqu'il reçoit cette trame il effectue deux tests :
  - **Check Ia :** le nœud *A* met son bit et le bit de son premier successeur dans son vecteur d'appartenance à 1 et il compare son vecteur d'appartenance avec celui de la trame reçue, si le test est valide, le nœud *A* constate que sa dernière trame a été transmise sans erreur (acquittement positif).
  - **Check Ib :** si le test ChekIa n'est pas valide, le nœud *A* met son bit à 0 et le bit de son premier successeur à 1 et il compare son vecteur d'appartenance avec celui de la trame reçue, si le test est invalide, le nœud *A* constate que le problème ne concerne pas sa dernière trame émise. Le premier successeur est considéré alors comme fautif et son bit d'appartenance est mis à 0. Le nœud *A* va prendre un autre premier successeur et reste dans la première phase d'acquittement en attente d'une trame reçue depuis un nouveau premier successeur.
- **La deuxième phase d'acquittement**
  Le nœud *A* passe à cette phase si le test Check Ib de la première phase est valide. Durant cette phase le nœud *A* attend la réception d'une deuxième trame depuis son deuxième successeur pour confirmer que la trame qui a été émise par *A* est vraiment erronée en raison que *A* soit fautif ou bien que le premier successeur de *A* soit fautif. Lorsque *A* reçoit cette trame il effectue deux tests :
  - **Check II-1a :** le nœud *A* met le bit de son premier successeur dans son vecteur d'appartenance à 0 et il compare son vecteur d'appartenance avec celui de la

trame reçue ; si le test est valide le nœud *A* constate que le problème réside dans son premier successeur ; il met le bit de ce dernier à 0 et il acquitte sa trame (acquittement positif).

- **Check II-1b :** si le test Chek II-1a n'est pas valide, le nœud *A* met son bit à 0 et le bit de son premier successeur à 1, et il compare son vecteur d'appartenance avec celui de la trame reçue ; si le test est valide, le nœud *A* constate que le problème concerne sa dernière trame qui a été mal émise (acquittement négatif). Le nœud *A* est alors considéré comme fautif et son bit d'appartenance est mis à 0. Cette faute s'appelle la perte d'appartenance et le nœud *A* devra se réintégrer. Si le test II-1b n'est pas valide, le deuxième successeur est considéré comme fautif et son bit d'appartenance est mis à 0 et le nœud *A* reste dans la deuxième phase d'acquittement en attente d'une trame reçue depuis un nouveau deuxième successeur.

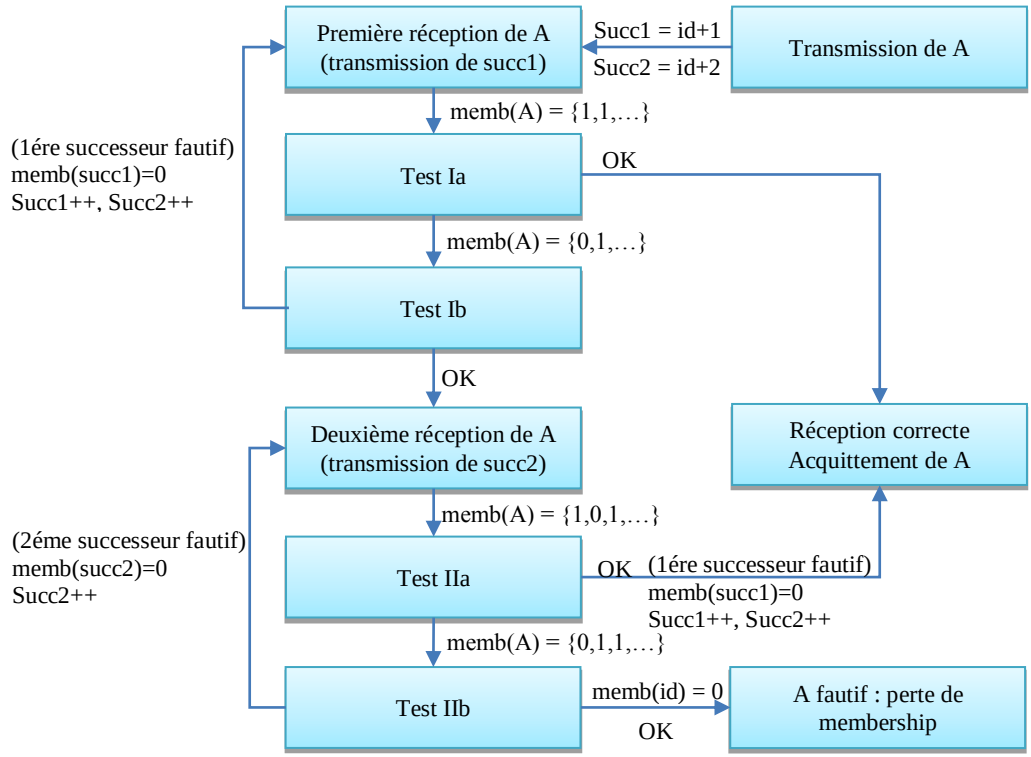


Fig. II-1-6 : l'algorithme d'acquittement implicite du protocole TTP

#### 4.5.3.2.3. Evitement de clique (Clique Avoidance)

Ce mécanisme est utilisé pour éviter l'apparition des sous-groupes de nœuds possédant des visions différentes de la vision globale du système. La détection des sous-groupes se fait en calculant le nombre de trames rejetés par rapport au nombre de trames acceptés pour chaque nœud du système ; si ce nombre est grand alors le nœud concerné s'exclu et se réinitialise avec le CState du système.

#### 4.5.3.2.4. La Redondance d'applications

La Redondance d'applications est une autre technique de la tolérance aux fautes, notamment ceux qui affectent les valeurs des données. La redondance d'applications

permet de masquer les effets lorsqu'un nœud fautif émit des trames erronées sur le réseau. Elle consiste à faire plusieurs copies de même nœud qui exécutent la même application et produisent les mêmes données (le nombre de ces copies dépend de taux de la fiabilité désirée pour l'application concernée). Ces copies de même nœud sont inclues dans la même FTU (Fault Containment Unit). L'objectif de cette redondance est de masquer la faute de l'un des nœuds de même FTU.

Il existe trois types de redondances :

- **La redondance active**
  Consiste à faire plusieurs copies d'un même nœud ; ces copies sont des nœuds actifs, c'est-à-dire que chacun d'eux possède un slot pour l'émission des trames, et il reçoit des trames, alors que son comportement soit le même que le nœud original.
- **La redondance passive**
  Consiste à faire une copie (nœud ombre) d'un même nœud ; ce nœud ombre est passif, c'est-à-dire il ne possède pas un slot pour l'émission mais il reçoit toutes les trames émises sur le bus et effectue des opérations comme : la synchronisation et l'appartenance. Le nœud ombre passe à l'état actif et remplace le nœud original si ce dernier tombe en panne.
- **Le Multiplexage**
  Consiste à faire plusieurs copies d'un même nœud ; ces copies sont des nœuds actifs, et aussi faire une copie (nœud ombre) de ce nœud original. Le nœud ombre passe à l'état actif et remplace l'un des nœuds actifs si ce dernier tombe en panne.

## 5. Les points forts de protocole TTP

- **La fiabilité**
  Ses mécanismes de la tolérance aux fautes permettent de supporter les applications critiques qui nécessitent un taux élevé de fiabilité.
- **Simplicité de la prévisibilité et validation**
  On peut facilement estimer exactement le temps de réponse et de démonter formellement le bon fonctionnement du réseau tout en respectant les contraintes temporelles.
- **Résolution de problème babbling idiot**
  Grâce à la séparation du bus Guardian, un nœud ne peut pas émettre des trames dans un slot de temps qui n'est lui pas affecté.
- **Débit de transmission élevé.**

## 6. Les points faibles de protocole TTP

- **Limite de flexibilité**
  TTP est très peu flexible, par exemple l'ajout d'un nouveau nœud nécessite de revoir la conception de tout le réseau (la solution de cette limite c'est l'architecture FlexRay).

- Les mécanismes d'accusé de réception sont gérés de manière implicite alors qu'ils deviennent plus longs en cas d'erreurs.

## 7. Comparaison entre CAN et TTP

Le tableau suivant présente une comparaison entre le protocole CAN et TTP [20].

| Critère /Protocole | CAN | TTP |
|---|---|---|
| Déclenchement de l'exécution des tâches applicatives | Par évènements | Par temps |
| Applications | Non Critiques | Critiques |
| Contrainte de Temps | Garantie par la priorité des évènements | Garantie par l'ordonnancement prédéfini des tâches et transmissions |
| Débit | Jusqu'à 1Mbits/s | Jusqu'à 2Mbits/s |
| Taille de la Trame | 44 bits à 108 bits (44 bits de contrôle + 0 à 64 bits de données) | 21 bits à 149 bits (21 bits de contrôle + 0 à 128 bits de données) |
| Types de Trame | 04 Types de Trame :<br>Trame de données<br>Trame de requête<br>Trame d'erreur<br>Trame de surcharge | 03 Types de Trame :<br>Trame d'Initialisation<br>Trame à CState Explicite<br>Trame à CState Implicite |
| L'identification de la trame | Identifiée | Non Identifiée |
| Codage des bits | Codage NRZ (No Return To Zero) | Modulation par Modification Fréquentielle (MFM) |
| Eviter Le Problème de Babbling idiot | n'est pas garantie | Garantie par le bus gardian |
| Stratégie d'accès au réseau | L'arbitrage bit à bit CSMA/CD | TDMA (Time Division Multiple Access) |
| Fiabilité | Moyenne | Haute |
| Flexibilité | Haute | Faible |
| Prévisibilité, Validation | Faible | Haute |
| Classe | B, C | D |
| Topologie | bus | Bus, étoile |
| Détection des erreurs de transmission | CRC | CRC<br>Codage MFM |
| Acquittement | Transmission explicite du Bit ACK | Transmission implicite à l'aide d'appartenance |

Tab. II-1-7 : Comparaison entre CAN et TTP

# Deuxième Partie du Chapitre II

# Le protocole de communication FlexRay

## 1. Introduction

Le protocole FlexRay est un protocole de communication pour les systèmes distribués. Il a été développé spécialement pour les applications embarquées de l'automobile. FlexRay est un protocole propriétaire proposé par un consortium dont les membres fondateurs sont BMW, Bosch, Daimler Chrysler, General Motors, Motorola, Philips et Volkswagen et qui inclut actuellement les acteurs majeurs de l'industrie automobile, dont l'objectif est de développer un protocole de communication à la fois flexible, tolérant aux fautes et rapide [1, 13].

L'objectif de protocole FlexRay est de repondre aux besoins de nouvelles applications dans le domaine châssis, propulsions et applications X-by-Wire. FlexRay est un protocole dit «sécurisé» capable de gérer les fonctions sensibles du véhicule qui concernent principalement le freinage et la direction. Les débits escomptés pour les composants actuels allant de 5 à 25 Mbit/s [14].

FlexRay offre une communication hybride qui permet la transmission des messages orientée temps et évènement sur le même support de communication.

## 2. Topologies de réseau FlexRay

Un cluster FlexRay consiste en un ensemble de nœuds interconnectés à un ou deux canaux de communication. Le protocole FlexRay supporte une variété des topologies qui fournissent une flexibilité de configuration [21] comme le bus linéaire passif, l'étoile passive, l'étoile active, la topologie hybride …, (Fig. II-2-1 présente les différentes topologies d'un cluster FlexRay).

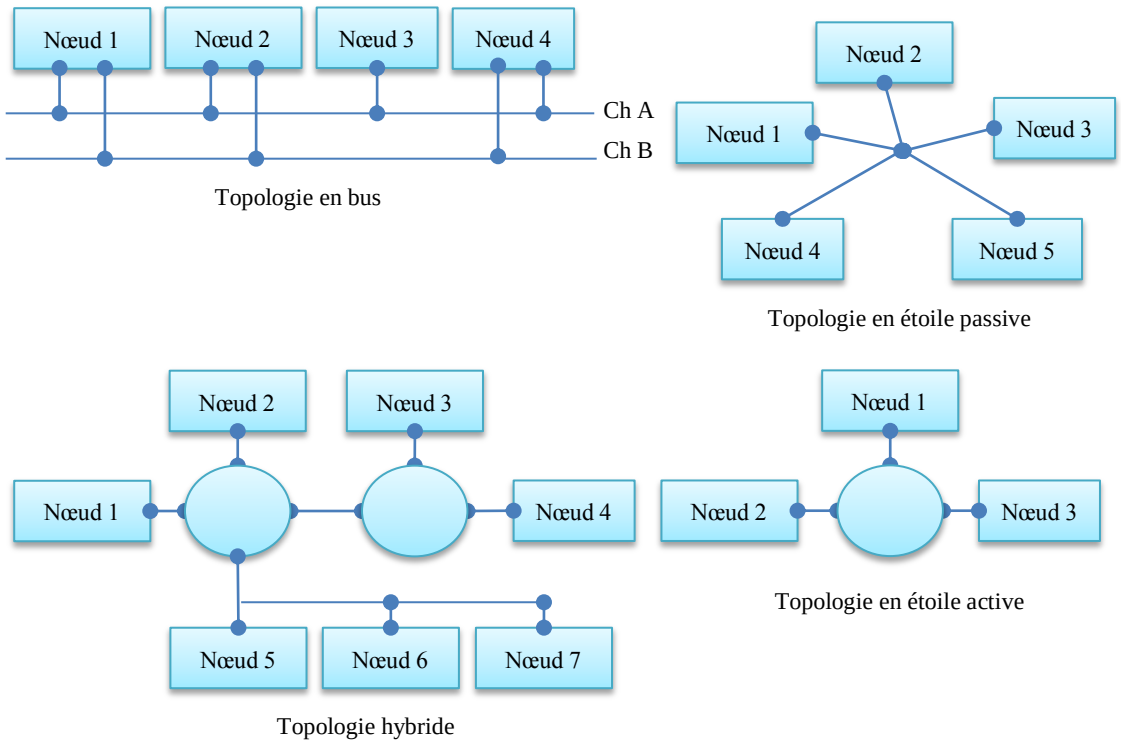


Fig. II-2-1 : Les différentes topologies d'un cluster FlexRay

## 3. L'architecture d'un nœud FlexRay

Un nœud FlexRay est responsable de contrôler une application de l'automobile. Il est constitué des parties suivantes (voir Fig. II-2-2):

- **Host**
  Est la partie qui est responsable de l'exécution des applications. Elle permet de contrôler et configurer le contrôleur de communication ; elle fournit aussi les données transmises pendant la communication.
- **Contrôleur de communication (Communication controller)**
  Est un composant électronique qui garantit l'exécution des services de protocole FlexRay. Il offre au host le statut de communication et les données des trames reçues.
- **Bus Driver**
  Est un composant électronique consiste en un transmetteur et un récepteur ; le contrôleur de communication est connecté à chaque canal de communication à travers un bus driver. Ce dernier peut être utilisé par exemple pour le codage et le décodage de données.
- **Bus Guardian**
  Est un composant électronique optionnel qui permet de protéger le canal de communication contre les accès fautifs du nœud (le cas où le nœud envoie une trame dans un temps qui n'est pas lui réservé).
  L'interface entre le host et le contrôleur de communication s'appelle le CHI (Controller Host Interface). Elle se constitue de trois parties : une partie pour les données, une partie pour les informations de configuration et une partie pour les signaux de contrôle et de statuts.

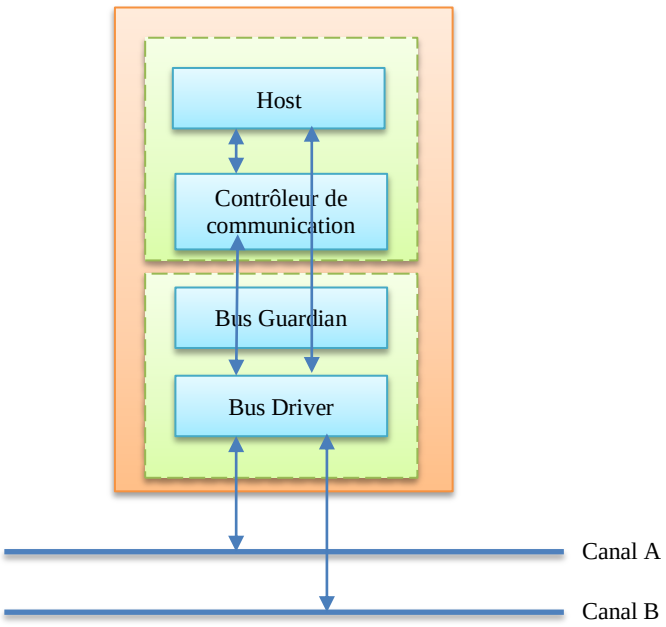


Fig. II-2-2 : Structure d'un nœud FlexRay

## 4. Structure de la trame du protocole FlexRay

La trame du protocole FlexRay est composée de trois sections: Tête de la trame (Header frame), Payload frame, et Trailer frame [22] (Fig. II-2-3 présente la structure de la trame FlexRay).

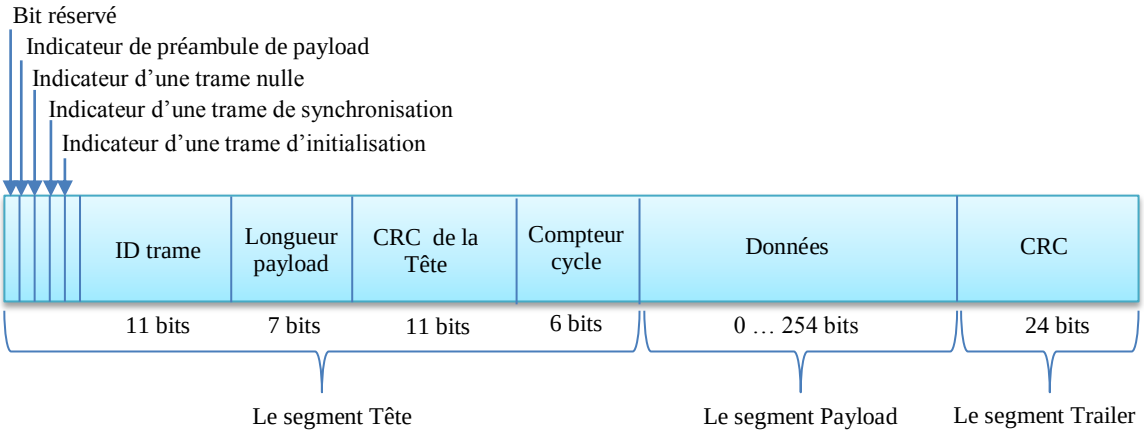


Fig. II-2-3: Structure de la trame FlexRay

### 4.1. Section Tête de la trame (Header frame)

Elle est composée de :

- 5 bits : un bit réservé, indicateur de préambule de payload, indicateur de la trame nulle, indicateur de la trame de synchronisation, indicateur de la trame d'initialisation.
- Identificateur de la trame (11 bits).
- Longueur de la section payload (7 bits).
- CRC de la section tête (11 bits).
- Comptage de cycle (6 bits).

#### 4.1.1. Bit réservé

Ce bit est réservé pour des utilisations futures du protocole ; le nœud émetteur met toujours ce bit à 0 et le nœud récepteur doit ignorer ce bit.

#### 4.1.2. Indicateur de préambule de payload (Payload preamble indicator)

Si le segment de transmission est le segment statique, ce champ indique la présence ou non d'un vecteur de gestion de réseau (network management vector) dans la section payload de la trame transmise.

Si le segment de transmission est le segment dynamique, ce champ indique la présence ou non d'un identificateur de la trame dans la section payload de cette trame.

Le bit indicateur de préambule payload est mis à 1 en cas de présence d'un vecteur de gestion de réseau ou identificateur de trame dans la section payload de la trame transmise, sinon il est mis à 0.

#### 4.1.3. Indicateur de la trame nulle (Null frame indicator)

Il indique que la trame transmise est nulle ou non, c'est-à-dire que la section de payload contient des données valides ou non.

Si la section de payload contient des données valides alors le bit indicateur de la trame nulle est mis à 1, sinon il est mis à 0.

#### 4.1.4. Indicateur de la trame de synchronisation (Sync frame indicator)

Il indique que la trame est une trame de synchronisation ou non, c'est-à-dire elle est utilisée pour la synchronisation de la communication ou non.

Si aucun nœud récepteur n'utilise la trame pour la synchronisation alors le bit indicateur de la trame de synchronisation est fixé à 0.

Si la trame est utilisée par tous les nœuds récepteurs pour la synchronisation alors le bit indicateur de la trame de synchronisation est fixé à 1.

#### 4.1.5. Indicateur de la trame d'initialisation (Startup frame indicator)

Il indique que la trame est une trame d'initialisation (Coldstart frame) ou non, c'est-à-dire elle est utilisée pour le mécanisme d'initialisation de la communication ou non.

Si la trame est une trame d'initialisation alors le bit indicateur de la trame d'initialisation est fixé à 1, Sinon il est fixé à 0.

#### 4.1.6. ID de la trame (ID frame)

Ce champ indique le numéro de slot durant lequel la trame doit être transmise. ID peut varier de 1 à 2047.

#### 4.1.7. Longueur de la section payload (Payload length)

Il indique la taille des données de la section payload, cette taille égale au nombre d'octets des données de la section payload divisé par 2.

#### 4.1.8. CRC de la section tête (Header CRC)

Ce champ contient le code CRC de la trame qui est calculé sur les champs : indicateur de la trame de synchronisation, indicateur de la trame d'initialisation, ID de la trame et le champ longueur de la section payload. Ce champ est calculé pour la détection des erreurs de transmission.

#### 4.1.9. Comptage de cycle (Cycle count)

Il indique la valeur de compteur de cycles du nœud émetteur lors de la transmission de la trame.

### 4.2. Section Payload

Elle contient de 0 à 254 octets des données de la trame.

Si la trame est transmise dans le segment statique, les premiers octets de 0 à 12 sont optionnellement utilisés pour indiquer le vecteur de la gestion de réseau.

Si la trame est transmise dans le segment dynamique, les deux premiers octets de la section payload sont optionnellement utilisés pour indiquer l'identificateur de la trame. Ce dernier représente un nombre qui identifie le contenu des données de cette section.

### 4.3. Section Trailer

Elle contient 24 bits de CRC de la trame. Ce CRC est calculé sur les champs des deux sections : tête et payload. Ce champ est calculé pour la détection des erreurs de transmission.

## 5. Les Etats principaux d'un contrôleur de communication FlexRay

Le contrôleur de communication peut être en états suivants [23] : (voir Fig. II-2-4)

- **Default config**

  Dans cet état la communication est impossible. Donc aucune information concernant le réseau FlexRay est disponible. Lorsque le nœud est mis en marche (powered on), il sorte de cet état vers l'état config.

- **Config**
  Dans cet état le host du nœud s'occupe de configurer la communication de telle sorte que le canal de communication soit disponible pour l'opération de Wakeup.
- **Ready**
  Le contrôleur de communication entre dans cet état lorsqu'il est configuré avec succès. Dans cet état le contrôleur de communication est prêt pour démarrer le processus de début de communication. Il passe à l'état Wakeup ou à l'état Startup.
- **Wakeup**
  Le contrôleur passe à cet état s'il est un Wakeup node ; il est responsable de réveiller les autres nœuds du même cluster. Lorsque le contrôleur termine le processus de Wakeup il rentre dans l'état ready pour passer à l'état startup.
- **Startup**
  Dans cet état, le contrôleur de communication tente de synchroniser son horloge avec les autres nœuds du même cluster. Lorsque le contrôleur termine le processus de Startup il entre dans l'état Normal Active.
- **Normal Active**
  Dans cet état, la communication avec le bus est possible. Le nœud peut envoyer et réceptionner des trames et synchroniser son horloge [24].
- **Normale Passive**
  Le nœud passe à cet état en cas où des erreurs sont apparues. Dans cet état le nœud est seulement autorisé à écouter le bus sans aucune transmission de données.
- **Halt**
  Dans cet état, le contrôleur de communication du nœud est complètement arrêté. Le nœud passe à cet état en cas où des erreurs internes sont apparues. Lorsqu'il sorte de cet état il doit entrer à l'état Default Config.

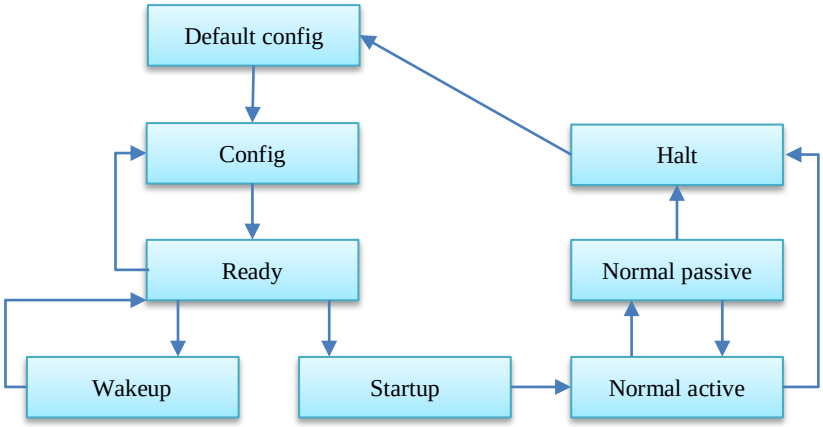


Fig. II-2-4 : Les états d'un contrôleur de communication FlexRay

## 6. Les services de protocole FlexRay

### 6.1. Service de contrôle d'accès au support de communication (Media Access Control)

Chaque cycle de communication de protocole FlexRay est divisé en quatre segments : statique, dynamique, fenêtre symbol et NIT (voir Fig. II-2-5).

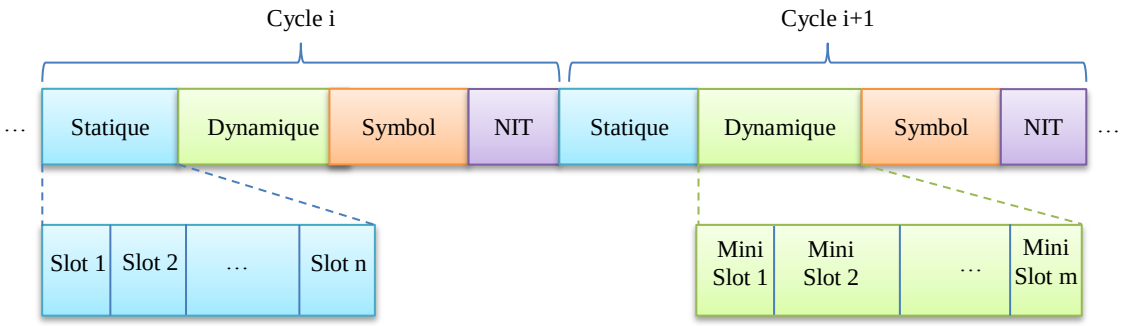


Fig. II-2-5 : La communication en cycle du protocole FlexRay [25]

#### 6.1.1. Le segment statique

Ce segment est approprié pour la transmission des messages orientés temps. La stratégie d'accès au support de communication appliquée dans ce segment est TDMA (Time Division Multiple Access) qui permet à tous les nœuds du réseau d'émettre et de recevoir les trames dans des temps prédéfinis. Le segment statique est subdivisé en des périodes de temps de duré fixe s'appelles slots statiques. Il affecté à chaque nœud un ou plusieurs slots statiques mais seulement le nœud propriétaire de slot est autorisé d'émettre sa trame pendant ce slot.

Si le nœud est connecté à deux canaux de communication, il peut envoyer la même trame pendant un slot statique sur les deux canaux ou deux trames différentes sur les deux canaux ou bien utilise un seul canal pour la transmission d'une trame et l'autre canal est utilisé par d'autre nœud. Fig. II-2-6 présente un exemple de communication dans le segment statique pendant un cycle [26]. Les Nœuds A et C utilisent les deux canaux de communication pendant les slots 1 et 3 pour l'émission redondante de la même trame ; le nœud B est connecté au canal b, il transmit une trame pendant les slots 2 et 4 ; les nœuds C et E partagent le slot 7 pour la transmission de leurs trames. Le nœud A transmit deux trames différentes sur les deux canaux de communication pendant le slot 5; le slot 6 n'est pas utilisé.

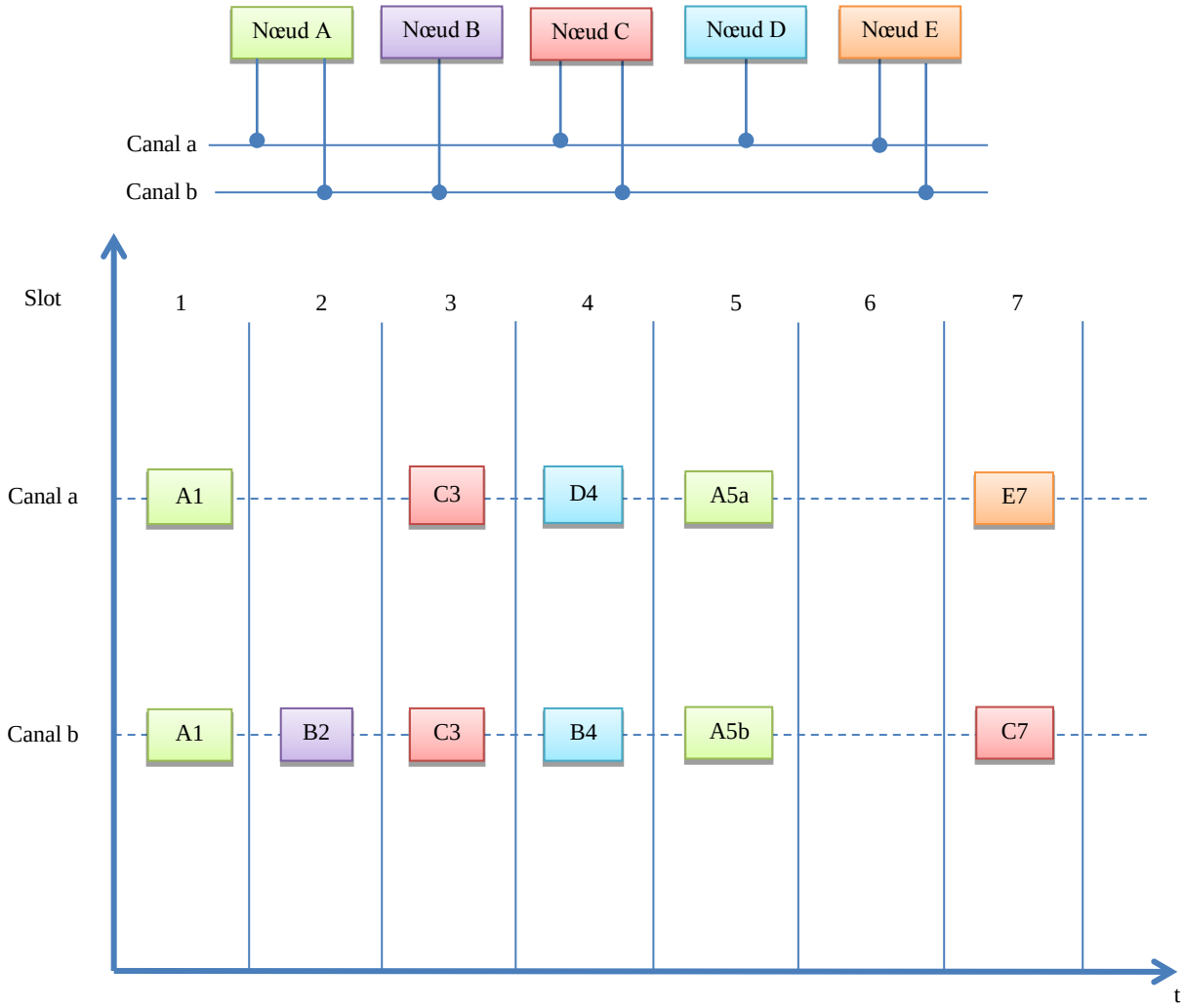


Fig. II-2-6 : Exemple d'une communication TDMA dans le segment statique

### 6.1.2. Le segment dynamique (Optionnel)

Ce segment est approprié pour la transmission des messages orientés événement. La stratégie d'accès au support de communication dans ce segment est FTDMA (Flexible Time Division Multiple Access), cette stratégie est basée sur la technique d'arbitrage pour choisir une seule trame la plus prioritaire en cas d'accès simultané de plusieurs nœuds au bus.

Selon le principe de l'arbitrage bit à bit, si deux nœuds ou plus veulent émettre des trames en même temps, le nœud le plus prioritaire (l'identificateur de sa trame est le minimal) émit sa trame en premier, et les autres nœuds restent en attente jusqu'à la fin de transmission de la trame, et le processus d'arbitrage sera recommencé. Le segment dynamique est divisé en un ensemble de petites durées s'appelles minislots. Comme les slot statiques, les minislots peuvent être affectés aux trames. Si le contrôleur de communication veut transmettre sa trame, le minislot s'est étendu à une taille adéquate. Fig. II-2-7 présente un exemple de communication dans le segment dynamique.

Dans le segment statique, une trame statique est transmise pendant un slot statique, mais en segment dynamique une trame dynamique peut être transmise pendant plusieurs minislots.

En cas où un slot statique n'est pas utilisé ; cela implique un gaspillage de temps contrairement au segment dynamique où le minislot est relativement petit.

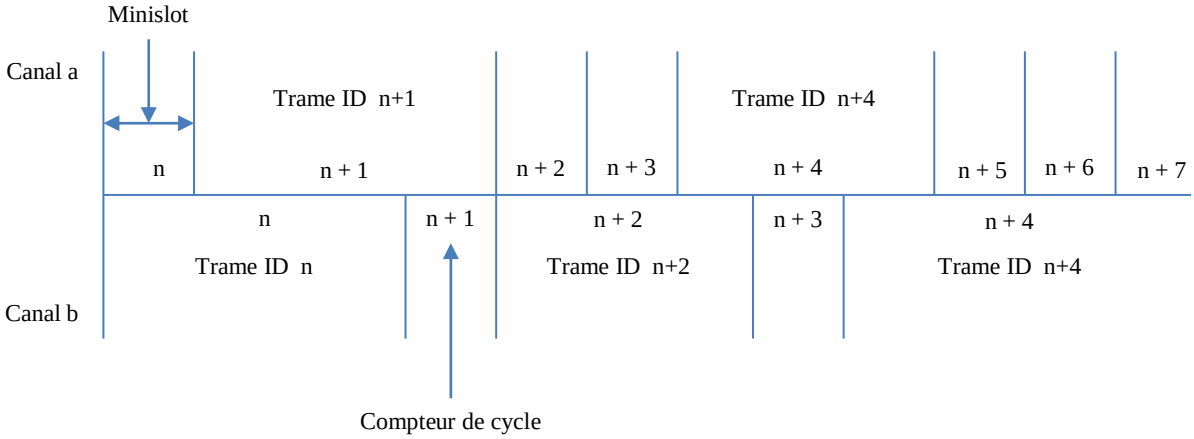


Fig. II-2-7 : Exemple d'une communication pendant le segment dynamique

### 6.1.3. Fenêtre de symbole (Symbol window)

Dans cette période les nœuds transmissent des informations de contrôle sur le réseau, par exemple le Wakeup pattern pour réveiller les autres nœuds du cluster, le signal CAS (Collision Advances Symbol) pour l'initialisation du cluster …

### 6.1.4. Temps Idle (Network Idle Time)

Cette période est réservée pour le calcul et l'application des corrections d'horloges des nœuds.

## 6.2. Service Wakeup

Avant que le cluster commence la communication, il doit être initialisé. Dans le protocole FlexRay, l'initialisation d'un cluster consiste en deux phases : wakeup et startup.

La procédure de wakeup est l'une des opérations de POC (Protocol Operation Control) [22]. Elle est dédiée pour réveiller les nœuds d'un cluster qui sont en mode "sleep" avant de commencer l'opération de startup. La procédure de wakeup est complètement contrôlée par le host, tandis que le contrôleur de communication est responsable de la transmission de wakeup pattern. Le wakeup pattern est une séquence d'un nombre de symbole wakeup ; un symbole wakeup est composé de n bits de bas niveau suivi par m bits de haut niveau (bas et haut sont des niveaux du signal) [27].

Dans un cluster FlexRay, au moins un nœud de cluster aura besoin d'une source externe de wakeup ; ce nœud (wakeup node) est responsable de réveiller les autres nœuds de même cluster lorsqu'il reçoit un évènement externe.

L'opération de wakeup est effectivement commencée lorsque le nœud wakeup node passe à l'état ready après que son host configure le canal pour la transmission de wakeup pattern dans l'état config, il envoie un wakeup pattren sur le bus de communication. Pour

que l'opération de wakeup doive être complétée, le contrôleur de communication retourne à l'état ready et signale au host le résultat de la tentative de wakeup. Ce dernier envoie des commandes au contrôleur de communication en fonction de ce résultat.

Lorsque le bus driver d'un nœud qui en mode sleep reçoit un wakeup pattern sur son canal de communication, il transmit ce signal aux autres composants du son nœud pour les réveiller.

Le contrôleur de communication de nœud wakeup node émetteur d'un wakeup pattern ne peut pas vérifier si tous les nœuds récepteurs sont connectés au canal configuré ; s'ils sont réveillés ou non après la transmission de cet wakeup pattern sauf jusqu'à la phase de startup.

Wakeup est tolérante avec la collision, c'est à dire même si il existe plusieurs nœuds wakeup node qui transmissent le wakeup pattern en même temps sur le même canal, le signal résultant de cette collision permet de réveiller les autres nœuds.

### 6.2.1. Les états de l'opération Wakeup

Un nœud wakeup node passe d'un état à un autre dans la phase de wakeup pour réveiller les autres nœuds du même cluster (Fig. II-2-8 présente les états d'un nœud dans la phase de wakeup). Après avoir terminé l'opération de wakeup, le contrôleur de communication doit envoyer un signal représentant le résultat de wakeup au host (une initialisation de la variable result value variable). Si l'opération de wakeup ne s'effectue pas, la variable de résultat prend la valeur "UNDEFINED".

- **Etat écoute (Listen state)**

  Le nœud écoute le bus pendant un temps prédéfini, ce temps doit permettre au nœud de confirmer si le bus est libre ou non. Cet état est pour ignorer la transmission de wakeup pattern si une communication est détectée. Si une communication ou un wakeup pattern est détecté sur le bus, la tentative de wakeup est annulée. Si le contrôleur de communication reçoit une tête valide de trame dans cet état, le statut de la variable sera mis en valeur "RECEIVED_HEADER". Si le contrôleur de communication reçoit un pattern wakeup valide dans cet état, le statut de la variable sera mis en valeur "RECEIVED_WUP".

- **Etat envoi (Send state)**

  Le contrôleur envoie un wakeup pattern à travers le canal configuré et teste l'existence de collision. Si le contrôleur ne détecte aucune collision après la transmission de wakeup pattern, le statut de la variable est mis en valeur "TRANSMITTED". Si le contrôleur détecte un mouvement (une collision) sur le canal il sorte de la phase send et il entre dans la phase detect state.

- **Etat détecte (Detect state)**

  Dans cet état le contrôleur tente de découvrir la raison de la collision qui a été détectée dans l'état send. Le nœud écoute une autre fois le canal pendant un temps prédéfini. Une détection de wakeup pattern indique qu'il y ait une tentative de wakeup par un autre nœud, dans ce cas le statut de la variable est mis en valeur COLLISION_WUP. La réception d'un header d'une trame indique l'existence d'une

communication en cours, dans ce cas le statut de la variable est mis en valeur COLLISION_HEADER. Si le contrôleur détecte une collision mais il ne peut pas connaitre la cause de cette collision (wakeup pattern ou header frame), il met le statut de la variable à la valeur COLLISION_UNKNOWN.

Après la terminaison de l'opération de wakeup, le host signale une erreur seulement si la valeur de la variable est COLLISION_UNKNOWN. Cette erreur est envoyée au contrôleur de communication pour entrer à l'état listen.

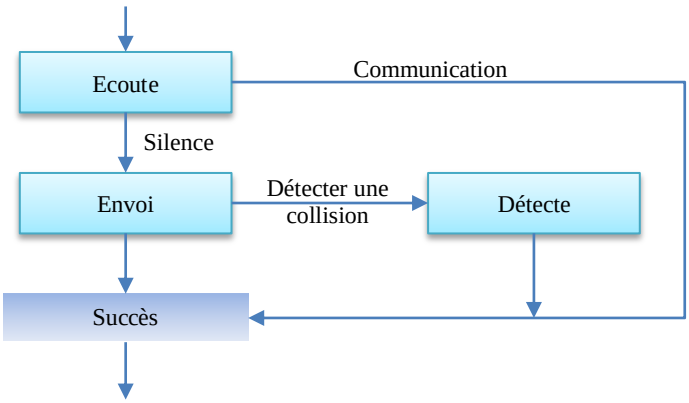


Fig. II-2-8 : Les états de l'opération Wakeup

### 6.3. Le service d'initialisation (Startup)

Le service startup (s'appelle aussi <<coldstart>>) est un service de POC, qui permet la synchronisation initiale et le partage de la même vue de cycles TDMA entre tous les nœuds d'un cluster. Pour que l'opération startup doive être effectuée il fallait que tous les nœuds doivent être réveillés (awake). Selon le mécanisme de startup il existe deux types de nœuds : les nœuds coldstart et les nœuds non-coldstart (la Fig. II-2-9 présente une vue simple d'un cluster flexray selon le mécanisme de startup) [28], pour que l'opération de startup doive être effectuée, il fallait qu'il existe dans le cluster au moins deux nœuds coldstart qui sont seuls capables d'initialiser le cluster. Les nœuds non-coldstart nécessitent au moins deux trames d'initialisation pour qu'ils puissent s'intégrer dans le cluster. La trame d'initialisation est en même temps une trame de synchronisation.

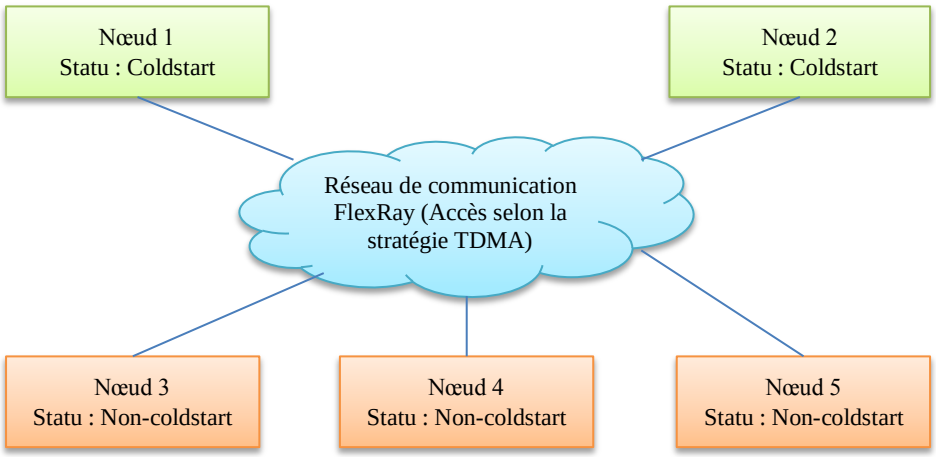


Fig. II-2-9: Un cluster FlexRay Selon le mécanisme de Startup

Le service startup consiste en deux phases : l'initialisation des nœuds coldstart et l'initialisation des nœuds non-coldstart.

### 6.3.1. L'initialisation des nœuds coldstart (Startup of coldstart nodes)

La Fig. II-2-10 présente le diagramme d'état transition d'un nœud coldstart au cours de l'opération de startup [27]. Dans cette étape il existe deux types de nœuds coldstart : maître (leader node) et l'esclave (following node). Le leader est choisi parmi tous les nœuds coldstart du cluster lors de l'état collision resolution. Ce nœud est le premier qui émit les trames d'initialisation ; les autres nœuds coldstart sont les suivants qui transmissent des trames d'initialisation.

- **Ecoute (Listen state)**

  Dans cet état, chaque nœud coldstart écoute le réseau pendant un temps prédéfini (supérieur à un cycle). L'objectif de cet état est de détecter les transmissions des trames ou des tentatives de startup sur le réseau. En cas où le nœud détecte une trame coldstart sur le réseau il passe à l'état initialize schedule, sinon après l'expiration de temps d'écoute le nœud passe à l'état collision resolution.

- **Résolution de la collision (Collision resolution)**

  Dans cet état, le nœud lance sa tentative d'initialiser le cluster. Dans cette phase le nœud transmit sur le réseau un signal CAS suivi par quatre (4) trames coldstart. Ces trames sont transmises pour initialiser le temps d'ordonnancement des autres nœuds (followings et non-coldstart nodes). Le nœud écoute le réseau au cours de sa transmission pour détecter s'il existe une collision, seul le nœud leader peut passer à l'état suivant (consistency check). Si le nœud n'est pas choisi comme un leader (following node) il retourne à l'état Listen.

- **Initialiser l'ordonnancement (Initialize schedule**)

  Seuls les following nodes entrent dans cet état. Cet état s'est activé lors de la réception d'une trame coldstart valide ; le nœud récepteur (following node) attend une deuxième trame startup pour initialiser le temps d'ordonnancement. En cas où le

nœud reçoit une deuxième trame coldstart valide il passe à l'état integration check, sinon il retour à l'état Listen.

- **Intégration (Integration check)**
  Dans cet état, le nœud attend deux autres trames pour tester le mécanisme de correction d'horloge. Si la correction de l'horloge est réalisée avec succès avec ces nouvelles trames, le nœud passe à l'état intégré, sinon il retourne à l'état Listen. L'objectif de cet état est d'éliminer dès le début les nœuds fautifs (faute d'horloge …).
- **Joindre (Join)**
  Dans cet état, le nœud following commence à émettre des trames coldstart, s'il reçoit des réponses (trames) après ses transmissions depuis le nœud leader il passe à l'état Normal Active , sinon il retourne à l'état Listen.
- **Test de la cohérence (Consistency check)**
  Dans cet état, le leader attend des réponses sur ces trames coldstart qui étaient transmises. S'il ne reçoit aucune réponse pendant un temps bien défini, il constate que les autres nœuds ne sont pas encore jointés au cluster, alors il retourne à l'état collision resolution. Si le nœud reçoit des réponses invalides qui ne correspondent pas à son temps d'ordonnancement, le leader retourne à l'état Listen ; s'il reçoit des réponses valides il passe à l'état Normal Active.
- **Active Normale (Normal Active)**
  Dans cet état, la communication au bus est possible. Le nœud peut envoyer et réceptionner des trames et synchroniser son horloge.

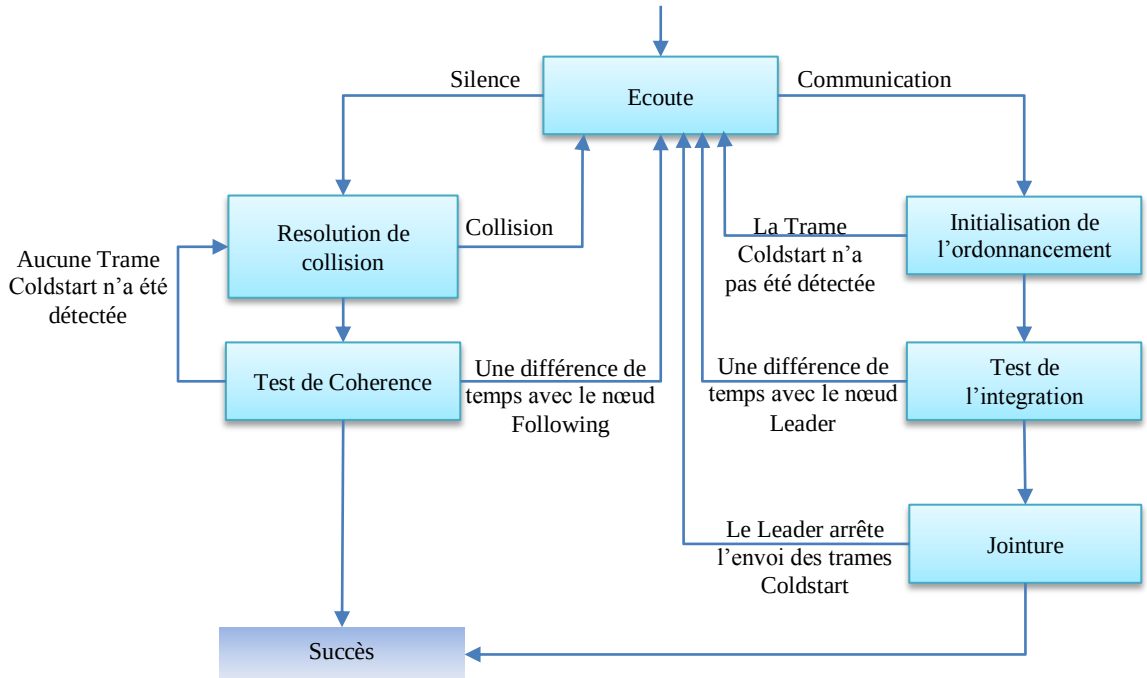


Fig. II-2-10: Diagramme d'état transition de service startup des nœuds coldstart

### 6.3.2. L'initialisation des nœuds non-coldstart (Startup of non-coldstart nodes)

Les nœuds non-coldstart exécutent presque le même chemin d'initialisation que les nœuds coldstart de type following, à la différence que les nœuds non-coldstart ne

transmissent pas des trames d'initialisation durant la phase de startup (Fig. II-2-11 présente le diagramme d'état transition d'initialisation d'un nœud non-coldstart).

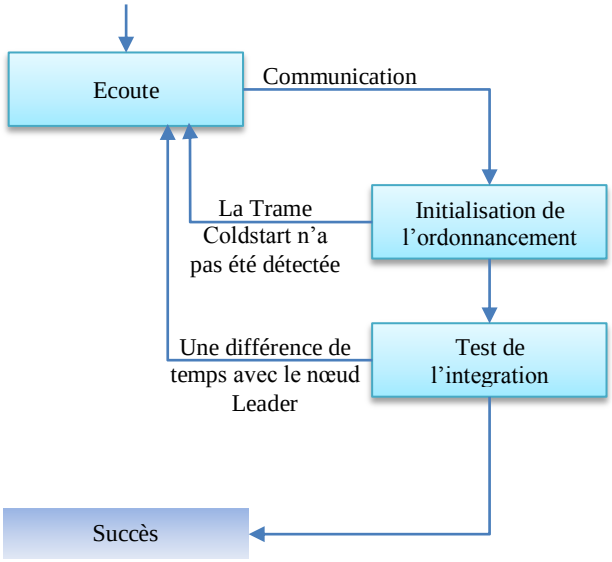


Fig. II-2-11: Diagramme d'état transition de service startup des nœuds non-coldstart

## 6.4. Synchronisation d'horloge

### 6.4.1. Représentation de temps

Dans le protocole FlexRay le temps est représenté sous forme hiérarchique de trois niveaux : cycle, macrotick et microtick (Fig. II-2-12 présente la hiérarchie du temps dans le protocole FlexRay).

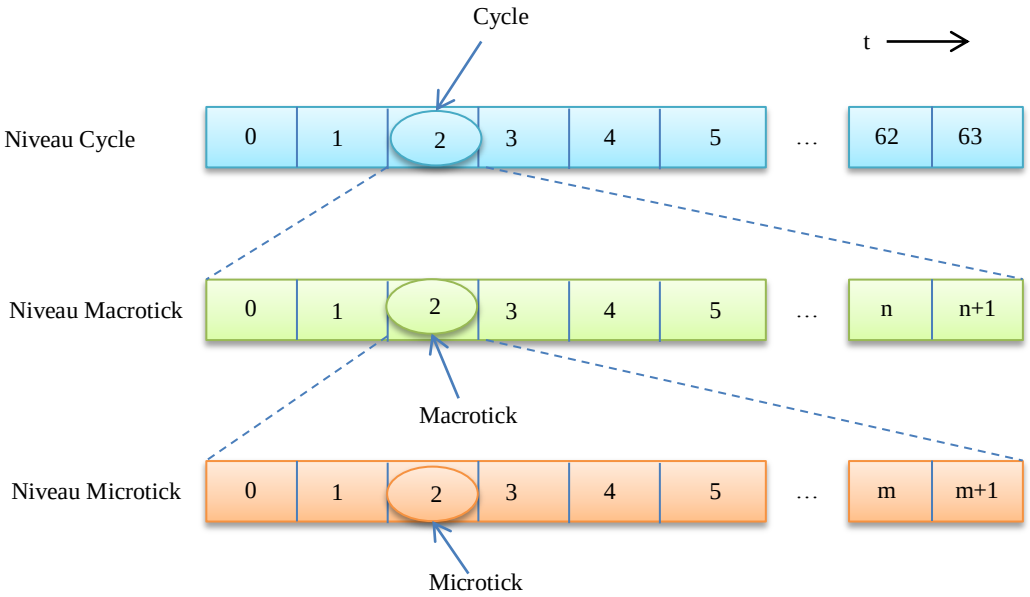


Fig. II-2-12 : La hiérarchie du temps

- **Microtick**
  C'est l'unité de base générée par l'horloge locale d'un nœud. Il présente une granularité de temps local d'un nœud. Le microtick peut se varier d'un nœud à l'autre.
- **Macrotick**
  C'est un nombre entier des microticks, il présente le temps local d'un nœud. La durée d'un macrotick est la même dans tous les nœuds d'un cluster. Le nombre de microticks par macrotick peut diffère d'un nœud à l'autre.
- **Cycle**
  C'est un nombre entier des macroticks ; le nombre de macroticks par cycle doit être le même dans tous les nœuds d'un cluster.
- **Temps Global**
  C'est la vue de temps commune entre tous les nœuds d'un cluster. Dans le protocole FlexRay, il n'existe pas une référence absolue de temps global ; chaque nœud possède son point de vue sur le temps global.
- **Temps Local**
  C'est le temps de l'horloge d'un nœud, il est représenté en 3 niveaux : cycle, macrotick et microtick.

L'objectif de l'algorithme de synchronisation est d'adapter le temps local d'un nœud pour qu'il doive correspondre au temps global (La Fig. II-2-13 montre l'objectif de processus de synchronisation du protocole FlexRay).

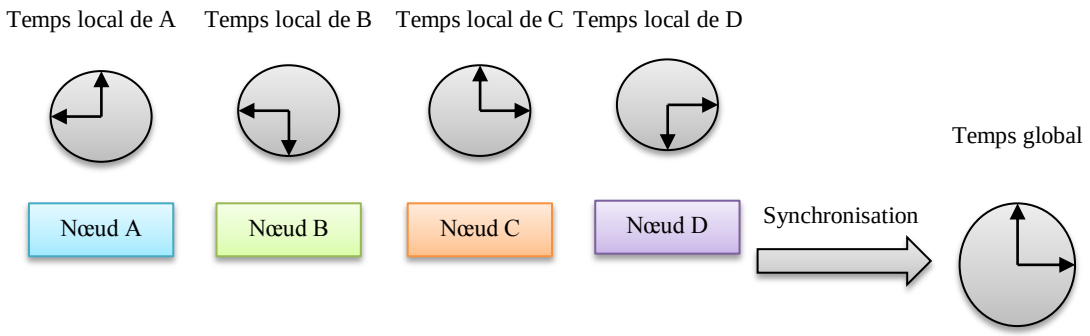


Fig. II-2-13 : L'objectif de service de synchronisation

### 6.4.2. L'algorithme de synchronisation d'horloges

La synchronisation dans le protocole FlexRay permet à chaque nœud de calculer les différences de fréquence (rate) et de phase (offset) et d'appliquer les corrections d'offset et de rate pour rendre le temps local identique au temps global. La différence de phase entre deux horloges est un décalage de temps entre les deux, tandis que La différence de fréquence est une différence de rythme d'horloge (Fig. II-2-14 montre les différences entre les fréquences et les phases des deux horloges).

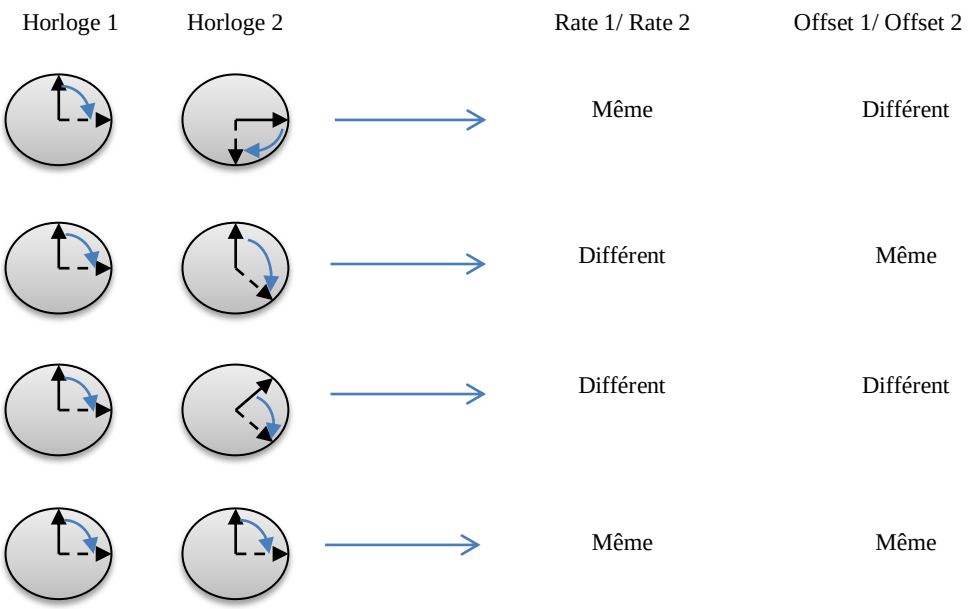


Fig. II-2-14 : Les différences de fréquence et de phase entre deux horloges

La synchronisation consiste en deux processus concurrents. Le processus de génération de macroticks (MTG) qui contrôle les compteurs de cycle et de macroticks et applique la correction de rate et d'offset et le processus de synchronisation d'horloge (CSP) qui mesure et enregistre les valeurs de déviation et calcule les valeurs de correction de rate et d'offset (La Fig. II-2-15 présente les étapes des deux processus de synchronisation pour le calcul et l'application de la correction des valeurs de rate et d'offset).

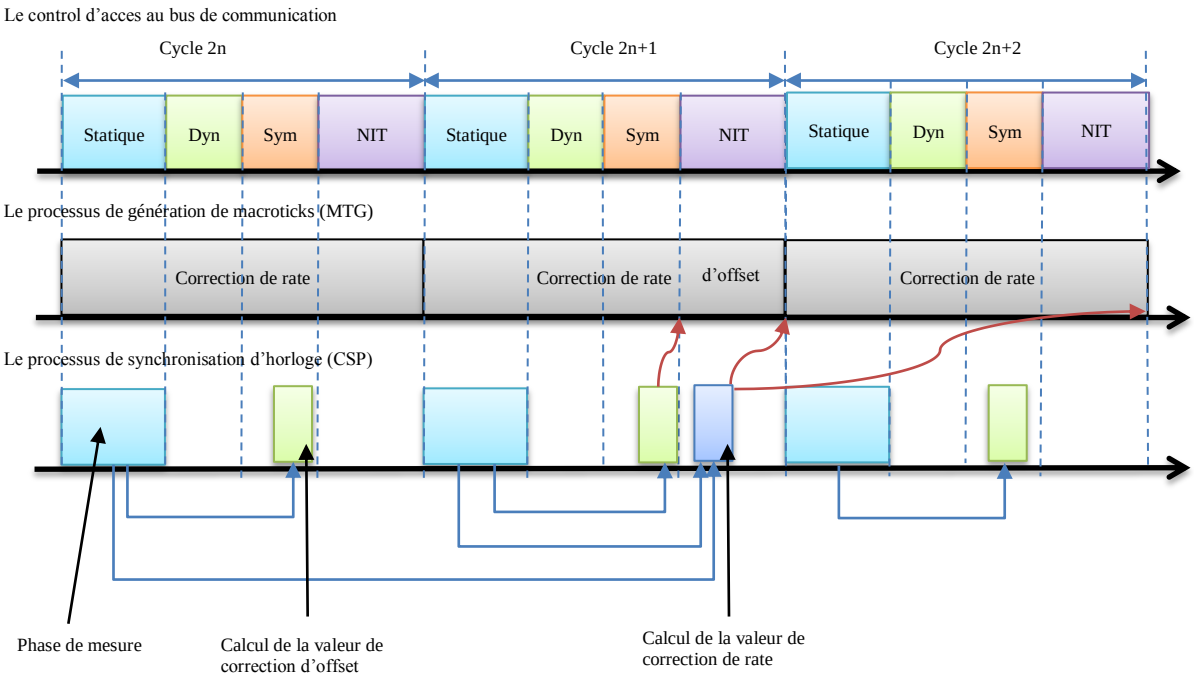


Fig. II-2-15 : La relation en fonction de temps entre la synchronisation d'horloge et l'accès au support de communication

#### 6.4.2.1. Le processus de synchronisation d'horloge (CSP)

Lors de la réception d'un message, ce processus permet de mesurer à priori la différence entre le temps prévu d'arrivée et le temps actuel d'arrivée de ce message. Les valeurs de correction d'offset et de rate sont calculées à base des différences dans les arrivées des messages mesurées dans le segment statique.

La valeur de correction de l'offset est calculée à chaque cycle après le segment statique et avant le segment NIT.

La valeur de correction de rate est calculée à chaque deux cycles, juste après la fin de segment statique de chaque cycle impair.

- **L'algorithme FTM (Fault-Tolerant Midpoint)**

  Le protocole FlexRay utilise l'algorithme FTM pour calculer la valeur de correction d'offset et de rate. Toutes les mesures des différences entre les arrivées prévues et réelles des messages sont calculées avant d'appliquer cet algorithme.

Les étapes de l'algorithme FTM sont les suivantes :

- Déterminer le nombre de la valeur de paramètre K à base de nombre des différences des valeurs du tableau Tab. II-2-16.
- Trier les valeurs selon un ordre croissance, et supprimer les k plus grandes valeurs et les k plus petites valeurs.
- Choisir parmi les valeurs restantes, la valeur la plus grande et la valeur la plus petite et calculer la moyenne de ces deux valeurs ; si cette valeur n'est pas entière, elle s'est arrondie en entier le plus proche. Cette valeur représente le terme de correction.

| Nombre de valeurs | K |
|---|---|
| 1-2 | 0 |
| 3-7 | 1 |
| > 7 | 2 |

Tab. II-2-16 : la valeur de paramètre K en fonction de nombre de valeurs

#### 6.4.2.2. Le processus de génération de macroticks (MTG)

Ce processus permet d'appliquer les corrections d'offset et de rate pour produire des macroticks correctes à la base des valeurs d'offset et de phase calculées.

La valeur de correction de l'offset est appliquée à chaque deux cycle. Elle consiste à agrandir ou à diminuer la phase de NIT.

La valeur de correction de rate est appliquée durant tous les segments de chaque cycle de communication. Elle consiste à corriger la fréquence de l'horloge pour augmenter ou diminuer le nombre des microticks par macroticks.

## 7. Les points forts de protocole FlexRay

- ✓ Le segment dynamique dans le cycle de communication garantie une communication flexible.
- ✓ Le segment statique dans le cycle de communication garantie une communication déterministe.
- ✓ Le segment statique assure une prévisibilité de temps de réponse de la transmission de messages.
- ✓ Une consommation basse de bande passante puisque la transmission sur événement est possible dans la fenêtre dynamique.

## 8. Les points faibles de protocole FlexRay

- ✓ La tolérance aux fautes est faible par rapport au protocole TTP.
- ✓ FlexRay ne spécifie pas au niveau protocolaire d'algorithme d'appartenance ou de mécanismes de gestion des modes de marche. Si ces services doivent être assurés, il faut alors les implémenter au-dessus des couches protocolaires de FlexRay ce qui peut s'avérer plus complexe et moins performant.
- ✓ FlexRay souffre encore de sa jeunesse et contrairement à TTP/C, peu de travaux ont tenté de prouver formellement le protocole.

# Chapitre III

# Etat de l'art sur la simulation des protocoles TTP et FlexRay

## 1. Définition de la simulation

La littérature propose plusieurs définitions de la simulation. Selon [29], la simulation est la discipline de la modélisation d'un système physique réel ou théorique, ensuite l'exécution de ce modèle sur un ordinateur, enfin l'analyse de l'exécution et des résultats obtenus. Selon [30], la simulation est le processus de la conception du modèle d'un système réel et d'expérimentation sur ce modèle dans le but de comprendre le comportement du système et/ou d'évaluer des différentes stratégies pour le fonctionnement du système. On trouve une définition de la simulation dans l'Encyclopédie Universalis comme une procédure de recherche scientifique qui consiste à réaliser une reproduction artificielle (modèle) du phénomène que l'on désire étudier, à observer le comportement de cette reproduction lorsque l'on fait varier expérimentalement les actions que l'on peut exercer sur celle-ci, et à en induire ce qui se passerait dans la réalité sous l'influence d'actions analogues.

Donc la simulation est le processus de la conception ensuite l'exécution d'un modèle du système dans le but d'analyser et d'évaluer le comportement de ce système selon certains paramètres expérimentaux.

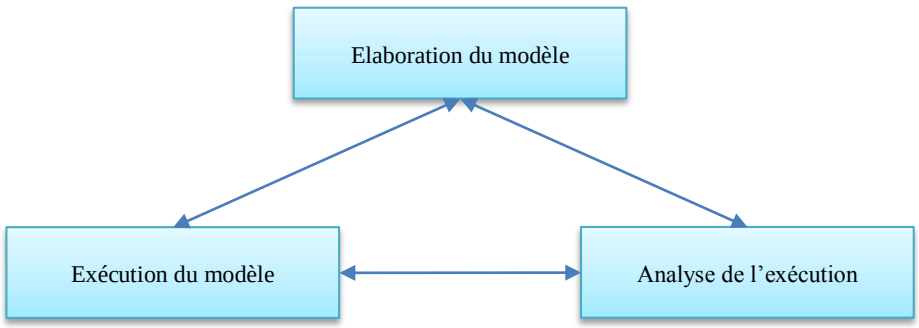


Fig. III-1: La définition de la simulation selon [29]

## 2. Les Types de La Simulation

On peut distinguer quatre catégories des simulations :

### 2.1. La simulation discrète

Le déroulement de la simulation se fait de façon événementielle ; dans ce cas l'horloge est modifiée par l'occurrence d'un évènement. Ce type de simulation est facile à implémenter.

### 2.2. La simulation continue

Le déroulement de La simulation se fait par intervalle de temps régulier ; dans ce cas l'horloge est modifiée par certain laps de temps constant. Ce type de simulation est très pratique mais pour certain domaine peut être coûteuse en terme de temps d'exécution.

### 2.3. La simulation orientée objet

Ce type de simulation permet de modéliser un système sous forme de classes d'objets. Cette simulation est basée sur les notions d'orientée objet tel que l'héritage et l'encapsulation. Le langage de conception appropriée pour la simulation orientée objet est

l'UML (Unified Modelling Language) et les langages de programmation orientés objet tels que Delphi et Java [31].

### 2.4. La simulation orientée agent

Le principe de ce type de simulation est d'utiliser les SMA pour la simulation. Elle offre des solutions à des problèmes complexes où les méthodes classiques montrent leurs limites. Elle est dédiée particulièrement à la représentation des comportements d'un système. La simulation orientée agent permet de modéliser un système sous forme d'un ensemble d'agents, les relations entre ces agents, et leur environnement. Les agents sont considérés comme étant des entités possédant un esprit (dans lequel se tiennent les processus décisionnels de l'agent) et un "corps" (qui est sa représentation dans l'environnement) [32]. Le lien entre un agent et son environnement se fait via l'utilisation de mécanismes d'influence et de perception qui garantissent la cohérence de système par exemple si deux agents veulent tous les deux se saisir d'un même objet de l'environnement (voir Fig. III-2).

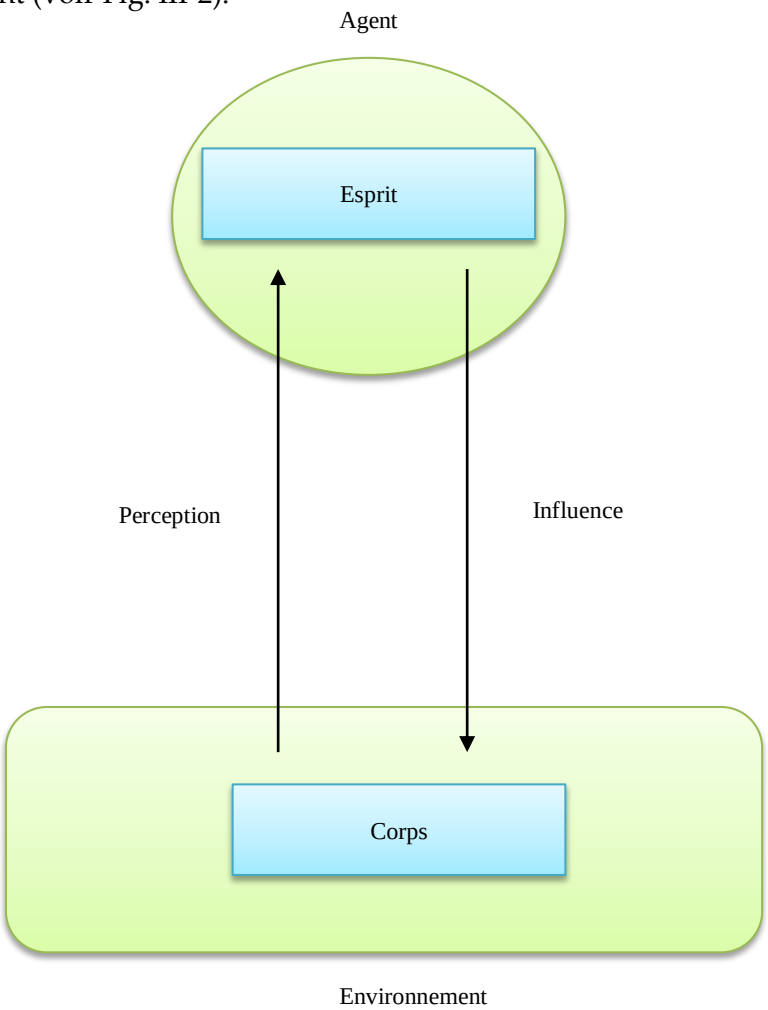


Fig. III-2 : Le principe de la simulation orientée agent

## 3. Les domaines d'application de la simulation orientée agent

Les travaux sur la simulation multi-agent s'appliquent à de nombreux domaines comme :

- La modélisation de phénomènes sociaux.
- La modélisation de phénomènes biologiques.
- La modélisation de phénomènes économiques.
- L'étude du comportement de matériaux granulaires dans le domaine de la physique.
- Traitement d'images

## 4. Etat de l'art sur la simulation des protocoles TTP et FlexRay

- Les auteurs dans [17] ont appliqué le paradigme orienté objet pour la simulation et l'implémentation de la spécification officielle du protocole TTP (par exemple la fonction d'appartenance, processus d'initialisation …) à l'aide UML et le processus de développement introduit par Graig Larma. Ce projet a utilisé quelques outils qui supportent le développement et le test de cette simulation tels que : Rational Rose UML pour la conception et JBuilder Java pour l'implémentation et JUnit pour le test. L'objectif de cette simulation est de s'assurer que la spécification officielle de TTP soit correcte, complète, et non ambiguë, et aussi de donner un aperçu sur les différentes modifications et améliorations sur cette spécification. Malheureusement, cette simulation est incomplète à cause de la complexité de la spécification de protocole TTP.
- Les auteurs dans [16] ont proposé une approche de modélisation et de vérification de protocole TTP à l'aide des automates. Selon ce travail, la partie communication d'un nœud TTP est modélisée sous forme de quatre automates. L'automate *Fonctionnement* pour la gestion du fonctionnement ; il permet par exemple de démarrer tous les automates simultanément après initialisation du système et aussi la gestion du temps. L'automate *MEDL* pour gérer l'ordonnancement des messages et l'accès au support de communication. L'automate *Bus Guardian* pour la gestion de fonctionnement de bus Guardian et de protéger l'accès au support de communication. L'automate *Contrôleur* pour gérer les services de réintégration, la gestion d'acquittement et d'appartenance de la partie communication d'un nœud TTA. Ce Travail permet de modéliser les deux parties d'un nœud TTA (Hôte et communication), mais il ne propose pas une technique de modélisation et de vérification des autres services de la partie communication comme le startup et la synchronisation.
- Les auteurs dans [33] ont utilisé le modèle SIDERA (SImulation model for DEpendable Realtime Architectures) qui permet la simulation de quelques services de protocole TTP comme le mécanisme d'initialisation (startup), la communication, la synchronisation des horloges, le service d'appartenance et la détection des erreurs. Ce travail présente le service de synchronisation comme une étude de cas avec un seul cluster ou bien un système multi-clusters ; il ne prend pas le facteur de non stabilité de rythme d'horloge.
- Les auteurs dans [34] ont proposé une simulation du protocole FlexRay à l'aide de langage VHDL et une implémentation FPGA. Ce travail se concentre sur la

spécification VHDL des états d'un nœud, la structure de la trame et les opérations de protocole pour les transitions des états comme la transition de l'état défault config à l'état wakeup et de l'état startup à l'état normal active..., le code VHDL généré est translaté à une implémentation FPGA à l'aide des outils XILINX et Leonardo Spectrum. Ce travail est limité à la simulation de quelques opérations élémentaires du contrôleur de protocole FlexRay. Ce travail ne prend pas en compte par exemple la simulation de bus Guardian, en plus cette simulation est une simulation de bas niveau.

- Les auteurs dans [35] ont proposé une technique de simulation et de vérification au niveau système de contrôleur de communication FlexRay à base du SystemC. Le papier propose de générer un code SystemC à partir d'une description SDL qui représente une analyse de la spécification de FlexRay. Ce travail a modélisé le contrôleur de communication sous forme d'un ensemble de modules, où chaque module est connecté aux autres modules grâce à des ports qui sont responsables de la transmission des signaux et des commandes entres les modules. Un module est implémenté sous forme d'un ensemble de macros, où chaque macro exécute une fonction spécifique ; chaque macro est translatée en une fonction dans le langage SystemC. Le travail se limite à présenter le développement et la vérification de mécanisme de startup de protocole FlexRay. Le résultat de la simulation est identique au comportement de startup dans la spécification de protocole.
- Les auteurs dans [36] ont proposé une technique de simulation et d'analyse de performance et de stabilité de l'algorithme de synchronisation des horloges de protocole FlexRay à l'aide de modèle SIDERA. Ils ont prouvé la robustesse de l'algorithme pour la correction des erreurs de phase et de fréquences des horloges des nœuds en cas de stabilité ou non de rythme de ces horloges. En revanche, ce travail ne tient pas en compte de simulation de l'algorithme de synchronisation en cas d'un système multi-clusters.
- Les auteurs dans [37] ont proposé une modélisation et une simulation de segment dynamique de protocole FlexRay au niveau système à base de langage systemC. Le simulateur proposé est basé sur trois modules : un module générateur des entrées (Input Generator Module) qui est responsable de générer les messages qui doivent être transmises dans le segment dynamique. Ce module est responsable d'attribuer à chaque message les valeurs de ces propriétés : la longueur (nombre de minislot occupé pour la transmission de message), la priorité (FrameID de message), pLatestTx(la valeur maximale de compteur minislot qui doit être affectée au message) et la période (le temps de génération de message en millisecondes).Un module de messages (Message Module) est responsable d'adapter le buffer de nœud pour qui 'il supporte les messages générés. Le module Dynamique (Dynamic Module) qui est responsable de fonctionnement de segment dynamique de protocole FlexRay. Il utilise la technique de minislot pour l'accès au support de communication. Cette simulation limite les propriétés des messages tels que la longueur de message doit être entre 2 et 15 minislots.

- Les auteurs dans [38] ont proposé une simulation de protocole FlexRay à base d'une analyse temporelle au niveau système. Ils offrent un modèle de simulation de la communication dans les segments statique et dynamique, et de calcul des valeurs de correction d'horloge (la phase et la fréquence) et l'application de cette correction dans la partie Idle de chaque cycle. Cette simulation offre une bonne analyse de temps d'accès au bus de communication dans les segments statique et dynamique avec une application de processus de synchronisation. Malgré que cette simulation soit intéressante, le processus de synchronisation et le service de startup qui garantit une synchronisation initiale n'ont pas été simulés ainsi que la partie window symbol n'a pas été implémentée dans cette simulation.
- Les auteurs dans [39] ont proposé une simulation avec Matlab pour évaluer l'utilisation de réseau FlexRay dans les deux segments statique et dynamique. Cette simulation a prouvé que l'utilisation de réseau FlexRay est influencé par quelques paramètres tels que : le nombre de trames, la longueur de trames, le nombre de slots statique et de minislots…
- Les auteurs dans [40] ont proposé une modélisation et une vérification de mécanisme de Startup de protocole FlexRay sous forme des automates temporels à l'aide de l'outil UPPAAL. Le mécanisme de startup de chaque nœud FlexRay est modélisé sous forme d'un automate de temps. Ce travail propose de créer deux types d'automates, l'un est pour les nœuds coldstart et l'autre pour les nœuds non coldstart. L'envoi et la réception des trames d'initialisation et le signal CAS sont modélisés sous forme des liens entre les automates temporels. Ce travail modélise tous les cas possibles des nœuds dans la phase de startup et il vérifie le service en fonction de temps. Malheureusement, le travail présenté ne modélise pas le service de synchronisation pour la correction d'offset et de rate dans la phase de startup.

# Chapitre IV

# La Méthodologie O-MaSE, le langage AUML et la plateforme Jade

# 1. Définitions

## 1.1. Agent

Il existe dans la littérature scientifique plusieurs définitions de l'agent.

[41] définit un agent comme un système informatique situé dans un environnement, capable de réaliser des taches d'une manière autonome sur cet environnement pour atteindre a ses objectifs.

[42] définit un agent comme une entité physique ou virtuelle

- qui est capable d'agir dans un environnement,
- qui peut communiquer directement avec d'autres agents,
- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- qui possède des ressources propres,
- qui est capable de percevoir (mais de manière limitée) son environnement,
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

## 1.2. Système multi-agent

[42] définit un SMA comme un système composé de :

- Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.
- Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- Un ensemble A d'agents, qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système.
- Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
- Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.
- Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.

# 2. Les méthodologies de développement des SMA

Une méthodologie est une démarche progressive qui commence par la définition des besoins préalables de l'utilisateur pour arriver à une implémentation d'un système capable de satisfaire les besoins initiaux [43].

On distingue généralement trois groupes des méthodologies de développement des SMA, celles qui sont basées sur l'orientée objet, celles qui sont basées sur l'ingénierie de connaissances et celles qui sont fondées sur l'orienté agent. On peut trouver des

méthodologies mixtes s'appuyant sur l'orienté objet et l'orienté agent comme la méthodologie GAlA, des méthodologies se focalisant sur l'orienté objet et l'ingénierie de connaissance [44] comme la méthodologie MAS-CommonKADS (voir Fig. IV-1).

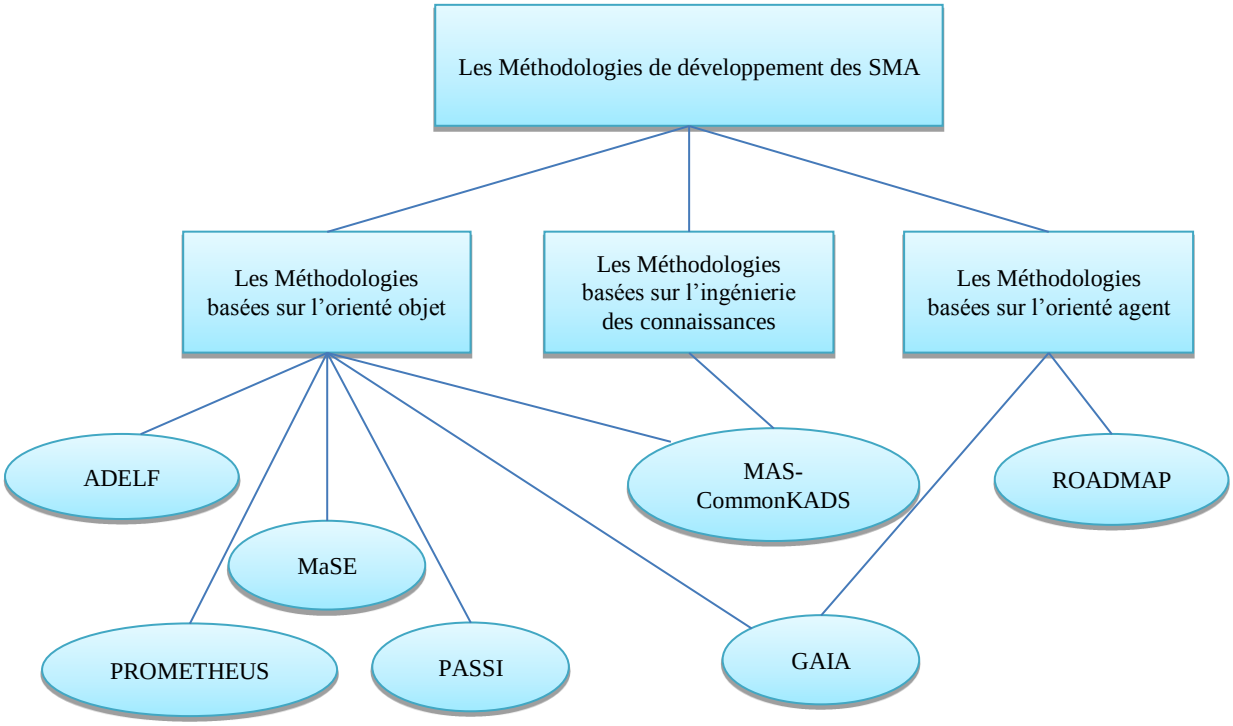


Fig. IV-1: Quelques Méthodologies de développement des SMA

Ce chapitre présente les différentes méthodologies de conception des SMA : GAIA, MAS-CommonKADS, MaSE, O-MaSE et enfin le langage AUML.

## 2.1. La méthodologie GAIA

GAIA est l'une des premières méthodologies d'analyse et de conception des SMA. GAIA considère un SMA comme une organisation ; elle se compose de deux phases principales (voir Fig. IV-2) [45] à base de modèle en cascade [43] :

### 2.1.1. Phase d'analyse

Cette phase produite à partir des besoins de SMA deux modèles abstraits : le modèle de rôle qui énumère les rôles et leurs attributs (responsabilités, les permissions, les activités et les protocoles) et le modèle d'interactions qui définit les dépendances entre les rôles du système.

### 2.1.2. Phase de conception

Transforme les modèles de la phase d'analyse en trois modèles plus concrètent : modèle d'agent qui définit les types d'agents du système et les instances correspondantes, modèle de service qui définit les différentes fonctions de chaque rôle du système, et le modèle d'accointances qui définit les liens de communications entre les types d'agents.

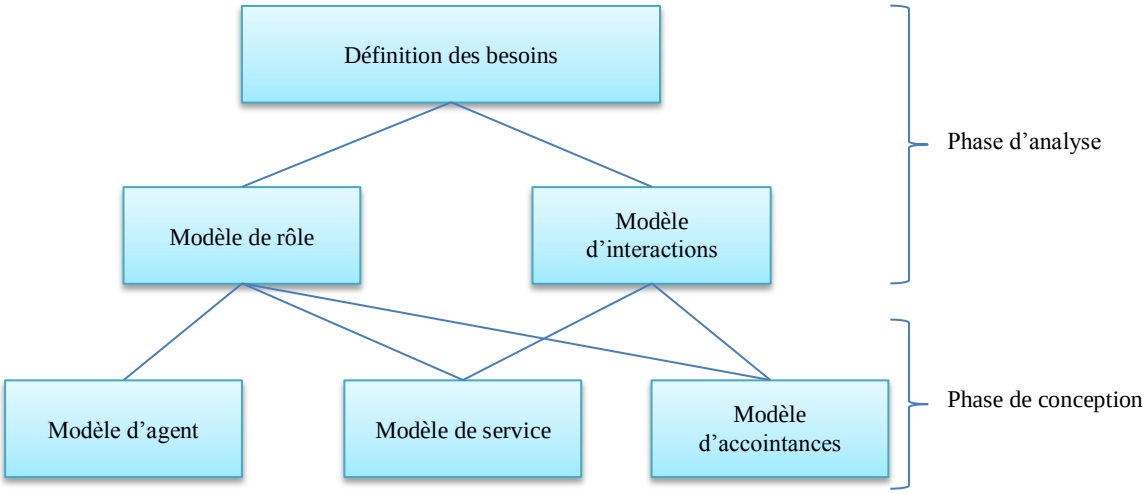


Fig. IV-2 : Les phases de la Méthodologie GAIA

### 2.1.3. Les points forts de la méthodologie GAIA

- GAIA rend facile la manipulation d'un grand nombre de concepts multi-agents, grâce à ses modèles.
- GAIA se concentre sur une spécification assez abstraite du SMA, elle laisse la conception de bas niveau et l'implémentation ouvertes au concepteur pour adopter l'architecture et le langage de programmation qu'il désire.

### 2.1.4. Les points faibles de la méthodologie GAIA

- Cette méthodologie est basée sur des SMA avec un petit nombre d'agents.
- GAIA impose aux agents d'avoir des liens statiques, de ce fait elle ne respecte pas la dynamique de système.
- GAIA ne propose pas une démarche pour aller d'un modèle conceptuel à un modèle d'implémentation du système.
- Malgré que les notations de GAIA soient simples, elles sont insuffisantes pour les problèmes complexes.
- GAIA manque la façon de construire l'ontologie du système.

## 2.2. La méthodologie MAS-CommonKADS (Mutiagent System-Knowledge Analysis and Development System)

MAS-CommonKADS est une méthodologie de génie de connaissance en ajoutant les techniques de l'orienté objet et l'ingénierie des protocoles**.** MAS-CommonKADS se compose de trois phases :

### 2.2.1. Phase de conceptualisation

Cette phase aide les développeurs d'obtenir une description préliminaire de problème à résoudre ; elle produite le modèle de cas d'utilisation.

### 2.2.2. Phase d'analyse

Le résultat de cette phase est une spécification des besoins de SMA, elle produit les modèles suivants :

- **Modèle d'agent :** pour la création des agents et ses instances, avec une spécification des caractéristiques de l'agent : capacités, comportement, services, groupes et hiérarchie d'agents.
- **Modèle de tâche :** définit les tâches exécutées par l'agent.
- **Modèle d'expertise :** définit les connaissances nécessaires à l'agent pour la réalisation d'un but.
- **Modèle d'organisation :** définit l'organisation sociale des agents, et les relations statiques entre les agents
- **Modèle de coordination :** définit les protocoles d'interaction entre les agents.

### 2.2.3. Phase de conception

Le modèle conceptuel est modélisé dans cette phase à partir des modèles produits par la phase d'analyse, ce modèle conceptuel est spécifié à travers les trois tâches suivantes : définir le réseau d'agents, définir l'architecture de chaque agent, définir la plateforme pour le développement de l'architecture de chaque agent.

### 2.2.4. Les points forts de MAS-CommonKADS

- Permet de modéliser les connaissances utilisées par les experts pour résoudre les problèmes complexes.
- Le modèle de coordination peut représenter les relations dynamiques entre les agents, donc cette méthodologie peut être utilisée pour le développement des systèmes multi-agents dynamiques.

### 2.2.5. Les points faibles de MAS-CommonKADS

- La méthodologie MAS-CommonKADS ne fournit pas des notations pour le modèle de conception.
- MAS-CommonKADS n'offre pas des techniques pour la transformation des modèles en d'autres modèles.

### 2.3. La méthodologie MaSE

Dans les années 1999 et 2000, DeLoach et Wood proposent la méthodologie MaSE (Multi-agents System Engineering) qui hérite les concepts des méthodologies orientées objet OMT (Object Modeling Technique) proposée par James Rumbaugh en 1991 et UML adopté en 1997 par l'OMG (Object Management Group), dans le but de supporter la spécification des agents. MaSE est un processus de développement d'un système SMA pour l'objectif de guider le concepteur depuis la spécification initiale du SMA jusqu'à l'implémentation de ses agents [46].

## 3. La méthodologie O-MaSE

O-MaSE (Organization based Multi-agents System Engineering) [47] est une extension de la méthode MaSE qui complète la modélisation de l'axe organisationnel d'un système multi-agent. Plusieurs diagrammes utilisés dans la méthodologie O-MaSE sont des variantes des diagrammes de l'UML comme le diagramme de classes, le diagramme de séquence, le diagramme d'état transition…

### 3.1. Méta-modèle de la méthodologie O-MaSE

Le méta modèle de la méthodologie O-MaSE défini à base des règles (grammaire) les concepts clés et les relations entre eux nécessaires pour la spécification d'un système multi agents [48], il est basé sur l'approche organisationnelle. L'organisation est composée d'un ensemble d'entités : Goal, Role, Agent, Domain Model…

### 3.2. Le processus de modélisation de la méthodologie O-MaSE

Le cycle de vie d'un SMA développé à base de la méthodologie O-MaSE est divisé en deux phases : phase d'analyse et phase de conception (voir Fig. IV-3).

- **La phase d'analyse**

  Cette phase prend en entrée les besoins fonctionnels et les exigences collectées du SMA et donne en sortie quatre diagrammes : un diagramme de but structuré sous forme hiérarchique, un diagramme d'organisation qui définit l'interface du SMA avec les acteurs externes, un diagramme de domaine qui définit l'ontologie de système et un diagramme de rôle nécessaire pour réaliser les buts. Pour faciliter le passage de diagramme de but au diagramme de rôle, MaSE propose un diagramme de cas d'utilisation qui détaille le diagramme de but et qui permet la création initiale des rôles.

- **La phase de conception**

  Cette phase prend en entrée les diagrammes produits par la phase d'analyse et donne en sortie des diagrammes qui représentent un modèle concret du SMA. Elle consiste en digrammes suivants : un diagramme de classes d'agent, qui relie les rôles aux types d'agents spécifiques, un diagramme de protocole qui décrit le détail des conversations entre les classes d'agents et un diagramme de plan qui décrit le détail de comportement d'une classe d'agent.

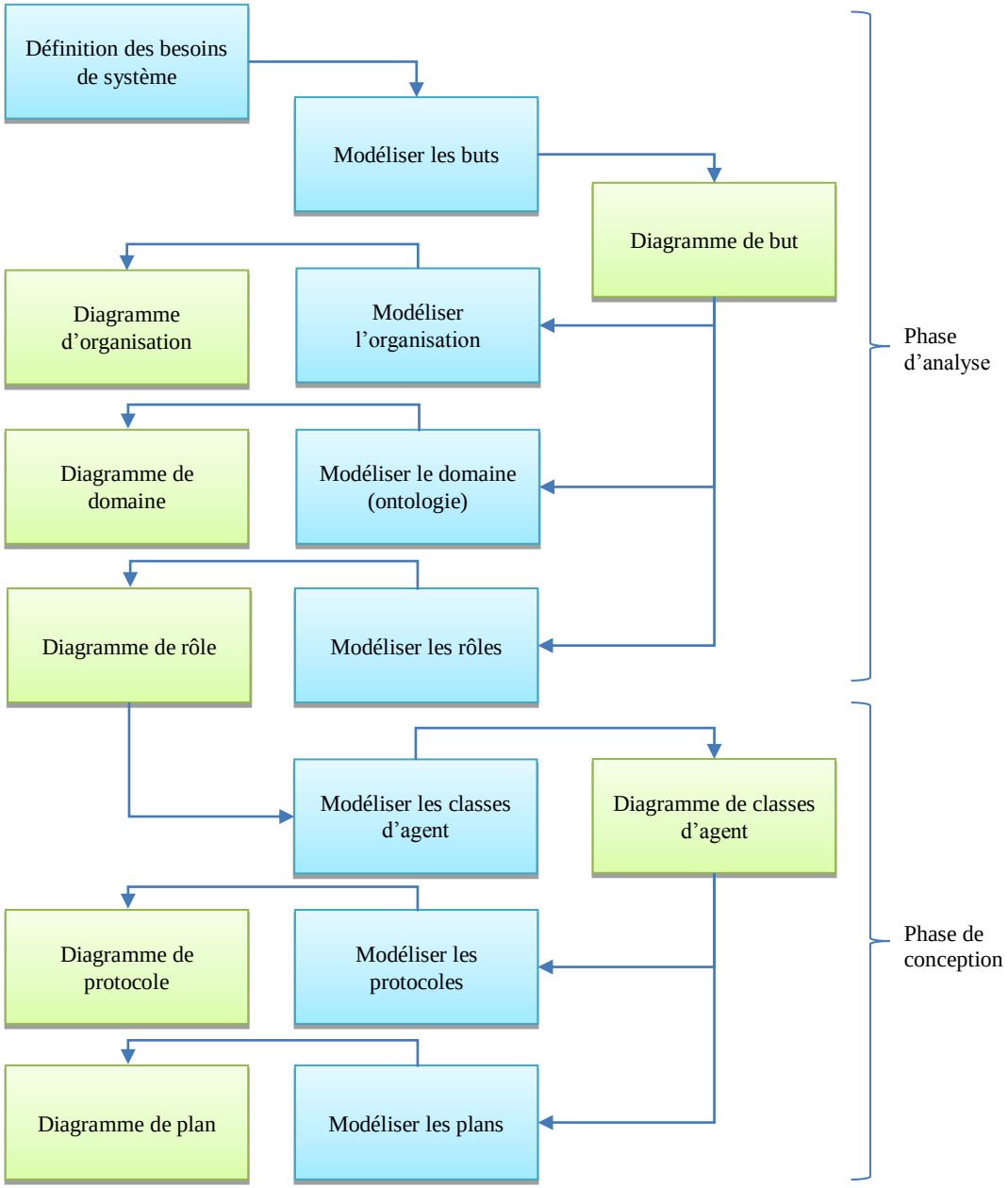


Fig. IV-3 : Processus de Modélisation O-MaSE

### 3.2.1. Diagramme de but (Goal Model)

L'objectif de cette étape est de capturer et identifier les besoins essentiels du système multi-agent et les transformer en un diagramme de buts structuré sous forme hiérarchique, où les buts sont organisés selon leur importance. On définit le but global (But0), et en raffine ce but en sous buts en utilisant la méthode de décomposition <<ET/OU>>, on continue la décomposition des sous buts d'une manière itérative jusqu'à ce que les buts ne pouvant pas être décomposés. Chaque but est noté sur le diagramme par le mot clé <<Goal>>.

### 3.2.2. Diagramme d'organisation (Organization Model)

L'objectif de ce diagramme est l'identification de l'interface du système (organisation) avec les acteurs externes. Alors Le système est représenté sous forme d'une organisation qui est notée sur le diagramme par le mot clé <<Organization>>.

### 3.2.3. Diagramme de domaine (Domain Model)

Appelé aussi le diagramme d'ontologie, il permet de capturer les types d'objets (dans notre cas l'objet c'est la trame) de l'environnement, les relations entre ces objets, et les propriétés générales de ces objets [49].L'objectif de modèle de domaine est d'aider les agents d'interpréter le contenu de ces objets. Le modèle de domaine d'O-MaSE est similaire au diagramme de classe d'UML.

### 3.2.4. Diagramme de rôle (Role Model)

Le diagramme de rôle est définit à partir des diagrammes de but et d'organisation. Un rôle est une description abstraite d'une fonction prévue d'une entité [50].Chaque but dans le diagramme de but doit être affecté au moins à un rôle qui le réalise. Chaque rôle doit réaliser au moins un ou plusieurs buts. Un rôle doit disposer d'une ou plusieurs capacités [51], qui représentent les comportements qui caractérisent ce rôle. Dans le diagramme, le rôle est noté par le mot clé <<Role>>, la relation entre le rôle et ses buts est sur titrée par le mot clé <<acheives>>. La capacité d'un rôle est notée par le mot clé <<Capacity>> et la relation entre le rôle et sa capacité est sur titrée par le mot clé <<requiers>>.

### 3.2.5. Diagramme de classes d'agent (Agent Class Model)

L'objectif de cette étape est de translater le diagramme de rôle qui capture les fonctionnalités de base du SMA à une forme plus concrète facile à implémentée, ce diagramme est similaire au diagramme de classe d'UML [52], où chaque composant du diagramme est une classe d'agent, attribuée des rôles, capacités et services. Les relations entre les classes d'agent représentent des conversations. La classe d'agent est notée par le mot clé <<Agent>> et chacun de ses rôles est noté dans le corps de la classe par le mot clé <<plays>>.

### 3.2.6. Diagramme de protocole (Protocol Model)

Ce diagramme est pour définir les détails des interactions entre les agents, il représente les messages échangés entre les agents et les acteurs externes selon l'ordre chronologique. Ce diagramme est similaire au diagramme de séquence de l'UML, où chaque composant du diagramme est une ligne de vie, sa tête est notée par le mot clé <<Agent>> suivi par le nom de la classe d'agent.

### 3.2.7. Diagramme de plan d'agent (Agent Plan Model)

Un diagramme de plan d'un agent définit comment l'agent se comporte (algorithme) dans une organisation lorsque il joue un rôle spécifique qui requiert une capacité spécifique pour réaliser un but spécifique. Donc pour chaque capacité d'un rôle d'agents, on doit créer un diagramme de plan. Ce diagramme est similaire au diagramme d'état transition de l'UML, où un état est sur titré par le mot clé <<State>>.

## 4. Une comparaison entre les Méthodologies

Ce tableau représente une comparaison entre quelques méthodologies de développement des SMA [53], en fonction de la couverture c'est à dire les modèles générés par chaque méthodologie.

| Modèles | MAS-CommonKADS | MaSE | GAIA | O-MaSE |
|---|---|---|---|---|
| Modèle de buts | Non | Oui | Non | Oui |
| Modèle de cas d'utilisation | (option) | Oui | Non | (option) |
| Modèle de rôles/de tâches | tâche | rôle | rôle | rôle |
| Modèle d'organisation | Oui | Non | Oui | Oui |
| Modèle d'ontologie/connaissance | connaissance | ontologie | Non | Domaine |
| Modèle d'agent | Oui | Oui | Oui | Oui |
| Modèle de protocole/coordination | coordination | protocole | interaction | protocole |
| Modèle d'état d'agent | Oui | Oui | Oui | Oui |
| Modèle de plateforme | Oui | Non | Non | Non |

Tab. IV-4 : Une comparaison entre quelques Méthodologies de développement des SMA

On remarque que GAIA ne modélise pas l'ontologie de SMA par rapport au MaSE, en plus de ça une étude comparative entre GAIA et MaSE prouve que MaSE est plus détaillée que GAIA [54].

On remarque que MaSE ne modélise pas l'organisation d'un agent, alors elle n'est pas adaptive à un SMA dynamique, par rapport au O-MaSE.

La méthodologie MAS-CommonKADS utilise le modèle de connaissance pour modéliser l'ontologie de système, où il y a une quantité considérable de type de connaissances.

## 5. Le Choix de la Méthodologie

On a choisi la Méthodologie O-MaSE pour la modélisation des protocoles de communication au sein d'un système embarqué distribué pour l'automobile, à cause de :

- La simplicité de la Méthodologie O-MaSE et sa large couverture du processus de développement.
- O-MaSE est indépendante de [55] :
  - L'architecture particulière du système multi-agents.
  - L'architecture d'un agent.
  - Langage de programmation.
  - La nature des messages échangés entre les agents.

- O-MaSE offre la possibilité de modéliser l'ontologie qui facilite au niveau de nos protocoles simulés une interprétation de contenu des champs de la trame.
- Dans les protocoles simulés, un cluster contient plusieurs nœuds, et un nœud contient plusieurs agents, alors O-MaSE offre la possibilité de modéliser cette hiérarchie organisationnelle de SMA.
- Une translation des diagrammes d'O-MaSE à la plateforme d'implémentation JADE est facile puisque les deux ont les mêmes concepts : agent, comportement, ontologie…
- Parmi les protocoles simulés, le protocole FlexRay ; il est possible que des agents apparaissent ou disparaissent au cours de segment dynamique de protocole, alors O-MaSE offre la possibilité de modéliser cette dynamique.
- Notre SMA ne contient pas beaucoup de type de connaissances (le seul type de connaissance modélisé c'est la trame), c'est pour cela la méthode O-MaSE est suffisante pour modéliser l'ontologie de notre SMA. Donc on pas besoin d'utiliser la méthodologie MAS-CommonKADS.
- Le problème de la méthodologie O-MaSE réside dans le diagramme de protocole ou la tête de la ligne de vie représente un agent et non pas un rôle, cette modélisation ne marche plus lorsqu'un agent peut jouer plusieurs rôles, alors on utilise le diagramme de protocole de langage AUML qui offre la modélisation des conversations entre les rôles d'agents au niveau de notre SMA.

## 6. Le Langage AUML (Agent Unified Modeling Language)

Est une extension de l'UML adopté par Bauer et Odell [56] pour la modélisation des agents au lieu des objets. Contrairement à l'objet, un agent est actif et autonome puisque il peut contrôler à lui-seul son état interne sans intervention d'une entité externe, aussi un agent n'est pas isolé mais en coopérativité avec les autres agents. Afin de modéliser les interactions entre les agents au niveau de notre SMA proposé, nous avons choisi le diagramme de protocole de l'AUML.

### 6.1. Diagramme de Protocole de l'AUML

C'est une extension de diagramme de séquence de l'UML, il est adopté par FIPA (Foundation for intelligent Physique Agents) pour spécifier les protocoles d'interaction entre les agents AIP (Agent Interaction Protocol), ce diagramme permet de représenter les échanges de messages entres les agents qui jouent des rôles (à l'intérieur de la tête de la ligne de vie on affiche le nom de rôle souligné ou le nom d'agent suivi par le nom de rôles). Ce diagramme hérite tous les spécificités de diagramme de séquence de l'UML qui permet de décrire : la transmission des messages (synchrones et asynchrones), les contraintes (bloquantes, non bloquantes et temporelles), et les cadres d'interaction (Alternative, Parallèle, Négative, Ignore, Loop). Il permet en plus de décrire les types d'envoi de messages à travers les connecteurs (AND, OR, XOR) (voir Fig. IV-5).

- **AND :** les messages liés à ce connecteur sont envoyés simultanément.
- **OR :** il permet l'envoi de plusieurs (zéro ou plus) messages liés à ce connecteur simultanément.
- **XOR :** il permet l'envoi d'un et un seul message lié à ce connecteur.

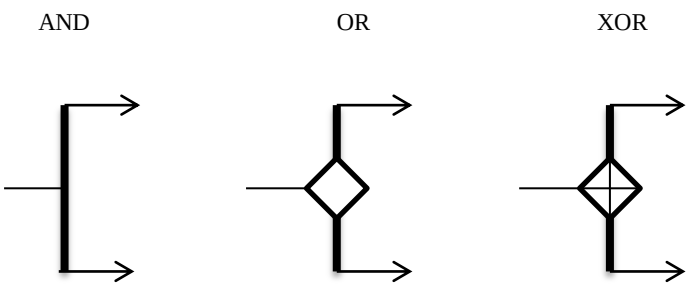


Fig. IV-5: Les Connecteurs d'envoi de message de diagramme de protocole de l'AUML

## 7. La Plateforme Jade

Nous présentons ci-dessous une description brève de la plateforme Jade. Puisque notre objectif ne vise pas à exploiter toutes les capacités de cette plateforme comme les protocoles de FIPA sous Jade où les outils de débogage qui sont disponibles, mais nous utilisons seulement le code Jade pour implémenter notre protocoles de communication.

Jade (Java Agent DEvelopement framework) [57] est une plateforme d'implémentation et d'exécution des SMA, crée par le laboratoire TILAB. Les SMA implémentés par Jade sont conformes à la norme de FIPA [58]. Jade est implémentée sous Java, il possède trois modules principaux :

- DF (DirectorFacilitor) fournit un service de « pages jaunes» à la plate-forme.
- ACC (Agent Communication Channel) gère la communication entre les agents.
- AMS (Agent Management System) supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

### 7.1. Les propriétés des agents implémentées sous Jade

Un agent implémenté sous Jade possède six propriétés [59] :

- **Autonomie**

  Chaque agent ayant un thread propre qui lui permet de contrôler leurs actions et de prendre leurs propres décisions afin de réaliser leurs buts.
- **Réactivité**

  Chaque agent peut percevoir les évènements de son environnement et régir en fonction de ces évènements.
- **Aspects sociaux**

  Chaque agent communique avec les autres agents de son environnement à travers le passage de messages asynchrones. Les messages ont une sémantique et une structure définie par le standard FIPA.
- **Dynamicité**

  Chaque agent peut découvrir dynamiquement d'autres agents et de communiquer avec eux.

- **Offre de service**
  Chaque agent offre un ensemble de services, il peut enregistrer ses services et les modifier, et il peut aussi chercher d'autres agents qui offrent les services dont il a besoin.
- **Mobilité**
  Chaque agent est implémenté dans un conteneur et il peut se déplacer.

## 7.2. Le langage de communication de la plateforme Jade

Les agents sous la plateforme jade communiquent entre eux à travers des messages sous la norme FIPA-ACL (Agent Communication Language). La classe ACLMessage représente les messages pouvant être échangés entre les agents [60]. Un agent utilise la méthode Send() pour envoyer un message à un autre agent, lorsqu'il veut recevoir un message il utilise la méthode Receive(). Les propriétés d'un message ACLMessage les plus importantes qui sont utilisés dans notre simulation sont les suivantes :

| Propriété | Signification |
|---|---|
| Sender | expéditeur du message |
| Receiver | destinataire du message |
| content | contenu du message |
| Language | description du langage du contenu de message |
| Ontology | description du l'ontologie du contenu de message |

Tab. IV-6 : Quelques propriétés d'un message de la classe ACLMessage du Jade

## 7.3. Comportement d'un agent sous la plateforme jade

Un agent Jade est capable de gérer plusieurs comportements simultanément. Un comportement est un objet de la classe Behaviour. La méthode setup() d'une classe d'agent peut contenir plusieurs Behaviours. Chaque Behaviour doit implémenter au moins les deux méthodes action() qui désigne les opérations à exécuter par le Behaviour, onEnd() : qui spécifie si le Behaviour a terminé son exécution ou pas. Un Behaviour peut être simple ou composé de sous Behaviours. Nous présentons ci-dessous les formats généraux de l'implémentation des quelque concepts clés de la plateforme Jade :

- La Méthode Send () est de la forme
  Send (message), où message est un objet de la classe ACLMessage.
- La Méthode Receive () est de la forme
  message = receive(), où message est un objet de la calsse ACLMessage.
- Une classe d'agent sous jade est implémentée de la forme

```
public class Nom_classeAgent extends Agent{
protected void setup() {
}
}
```

- Une ontologie sous Jade est implémentée de la forme

```
public class Nom_Ontology extends Ontology{
}
```

- Un Behaviour simple est implémenté de la forme

```
ClassNom_Behaviour extends OneShotBehaviour{
int valeurRetour = 0;
@Override
public void action() {
}
Public int onEnd(){
Return valeurRetour;
}
}
```

## 8. Choix de la plate-forme multi-agent Jade

Nous avons choisi la plateforme Jade pour implémenter les protocoles de communication au sein d'un système embarqué de l'automobile, à cause de :

- Notre objectif est de simuler d'une façon abstraite le comportement des protocoles de communication des systèmes embarqués de l'automobile et Jade utilise l'abstraction de comportement pour modéliser les tâches qu'un agent peut exécuter et les agents instancient leurs comportements selon leurs besoins et leurs capacités. Donc Jade garantit le critère d'abstraction.
- Le système embarqué de l'automobile est un système distribué, les services des protocoles de la partie communication sont aussi des services distribués telle que la synchronisation d'horloges. Jade permet de développer et d'exécuter des applications distribuées basées sur les agents.
- Les protocoles de communication sont modélisés sous forme d'agents réactifs et non pas cognitifs, Jade est appropriée pour l'implémentation des systèmes multi-agents à base des agents réactifs.
- Dans notre model conceptuel, nous avons supposé que chaque nœud de cluster du système embarqué simulé se compose de plusieurs agents, alors le nombre total d'agents dans le cluster est important, Jade utilise un modèle de programmation concurrente "un thread-par-agent" au lieu du modèle "un thread-par-comportement" pour éviter une augmentation du nombre de threads d'exécution exigés sur la plate-forme d'agents. Alors l'utilisation de Jade est très appropriée à notre modélisation.
- Les concepts de d'agent, centenaire et plateforme de jade correspondent bien aux concepts d'agent, nœud et cluster de notre simulation.
- Jade offre une transmission efficace, flexible et transparente des messages ACL entre les agents.
- Les concepts de Jade sont identiques aux concepts de la méthodologie O-MaSE choisie tels que : l'agent, l'ontologie, le comportement …

# Chapitre V

# Approche proposée

## 1. Introduction

L'objectif de notre travail est, principalement la modélisation et la simulation basées agents des services de bases des protocoles de communication orientée temps et mixte (orientée temps et événement) des systèmes embarqués distribués temps réel pour l'automobile. Tout au long de ce mémoire, nous nous sommes focalisés sur les deux protocoles de communication TTP et FlexRay. L'approche proposée permet dans un premier lieu de modéliser les aspects fonctionnels, structurels et comportementaux liés aux protocoles TTP et FlexRay à l'aide des diagrammes proposés par la méthodologie de conception des SMA O-MaSE et le diagramme de protocole du langage AUML, puis de translater ces diagrammes en une description formelle en code Jade.

## 2. Notre démarche

Fig. V-1 donne une vue globale sur la démarche de l'approche proposée.

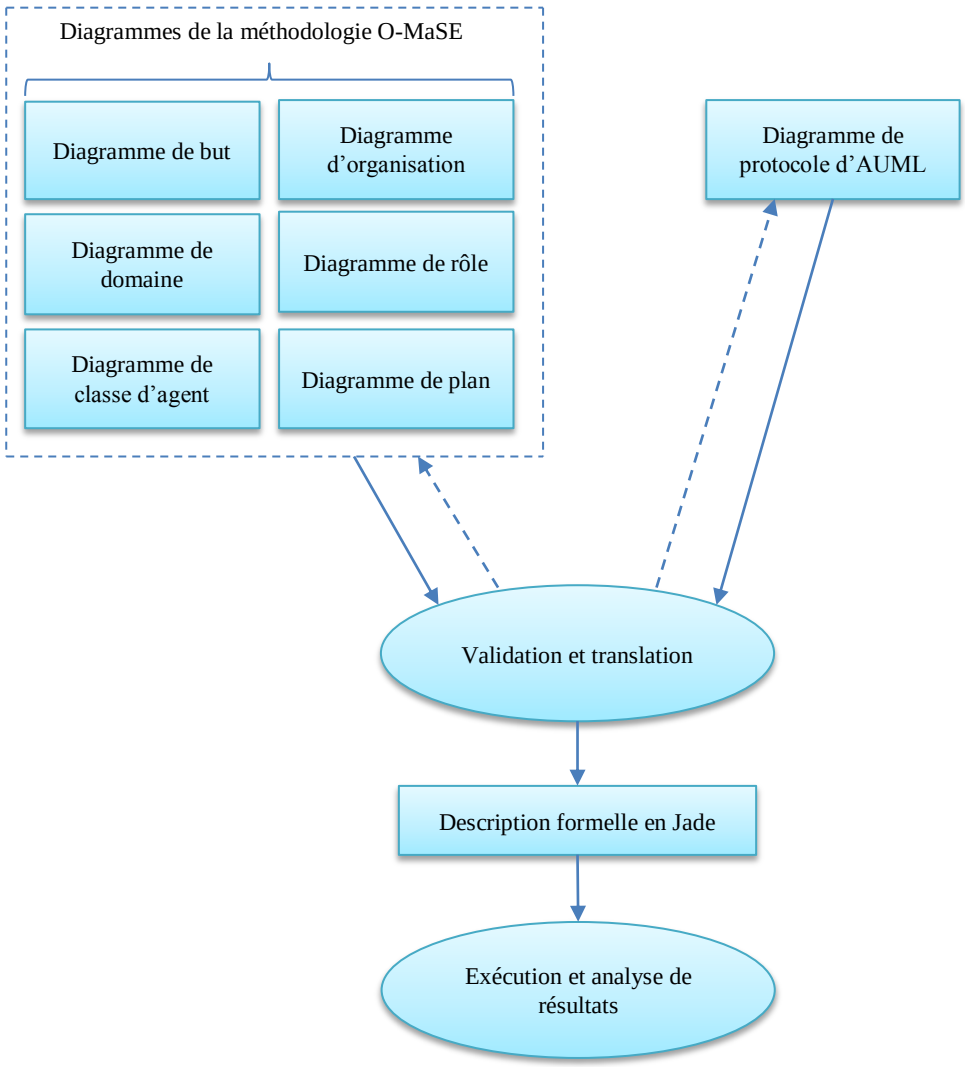


Fig. V-1 : Démarche générale de l'approche proposée

Cette démarche est basée sur les étapes suivantes :

## 2.1. Modélisation des servies des protocoles de communication à l'aide des diagrammes proposés par la méthodologie O-MaSE et le langage AUML

Cette étape consiste à établir les diagrammes suivants :

- **Diagramme de But**
  Le diagramme de but modélise les objectifs visés par les protocoles de communication des systèmes embarqués au sein de l'automobile. Le but du protocole de communication orienté évènement est de garantir une communication flexible et rapide entre les nœuds pour les applications distribuées de l'automobile. Le but du protocole de communication orienté temps est de garantir une communication tolérante aux fautes et déterministe entre les nœuds pour les applications distribuées critiques de l'automobile. Les protocoles mixtes permettent à la fois la flexibilité, la rapidité, le déterminisme et la tolérance aux fautes.
- **Diagramme d'organisation**
  Le diagramme d'organisation modélise l'interface du cluster avec les entités externe du cluster au niveau macro, et l'interface d'un nœud avec les entités externes du nœud au niveau micro.
- **Diagramme de domaine**
  Le diagramme de domaine modélise la trame du protocole de communication sous forme d'une classe avec ses différents champs sous forme des sous classes, le type de chaque champ, les classes de base à héritées, et les relations entre les différentes classes.
- **Diagramme de rôle**
  Le diagramme de rôle modélise les différents rôles jouant par les services du protocole de communication, comme l'émission d'une trame, la réception d'une trame, L'arbitrage, le changement d'état et la détection d'erreurs pour les protocoles orientés événements. La synchronisation d'horloges, l'initialisation du cluster, l'acquittement pour les protocoles orientés temps …
- **Diagramme de classe d'agent**
  Le diagramme de classe d'agent modélise les différents agents au niveau micro (nœud) et au niveau macro (cluster). A titre d'exemple, le bus de communication, l'arbitraire, le générateur d'événements, pour les protocoles orientés évènements, le noyau de protocole, le synchronisateur, le bus Guardian pour les protocoles orientés temps…
- **Diagramme de plan d'agent**
  Le diagramme de plan décrit le détail de chaque capacité de chaque rôle. A titre d'exemple la capacité qui modélise le mécanisme d'arbitrage pour les protocoles orientés évènements et la capacité qui modélise l'algorithme de startup pour les protocoles orientés temps.

- **Diagramme de protocole d'AUML**
  Le diagramme de protocole d'AUML décrit les scénarios d'échange de messages entre les agents dans le cas d'émission ou de réception des trames dynamiques pour les protocoles orientés événement, les trames statiques pour les protocoles orientés temps et pour les deux types des trames pour les protocoles mixtes.

## 2.2. Processus de translation

Le processus de translation en code jade est décrit à l'aide des règles suivantes :

| O-MaSE et AUML | Jade |
|---|---|
| Un groupe d'agents (un nœud) | container Jade |
| Un cluster (Groupe des nœuds) | plateforme Jade |
| Un rôle | un comportement parallèle composé de sous comportements simples, chacun d'eux correspond à la translation d'une capacité de ce rôle |
| Une classe d'agent d'O-MaSE | classe d'agent Jade |
| Un diagramme de plan d'une capacité | un comportement |
| Le diagramme de domaine | ontologie Jade |
| Les classes d'objets (la classe trame). | des classes de concepts en jade |
| Les diagrammes de protocole d'AUML | passage de message de type ACL Message entre les agents en Jade |

Tab. V-2 : Les règles de translation en code Jade

## 2.3. Exécution et analyse de résultat

Cette étape consiste à exécuter et analyser les résultats des différents services des protocoles de communication simulés par exemple pour vérifier que tous les nœuds réalisent la stratégie d'arbitrage d'une façon successive pour les protocoles orientés événements, et vérifier que les nœuds se synchronisent parfaitement pendant la phase de synchronisation pour les protocoles orientés temps, vérifier le contrôle d'accès orienté événement et temps pour les protocoles mixtes …

# Première Partie du Chapitre V

# Etude de cas : Modélisation et translation du protocole TTP

## 1. Le Modèle Conceptuel du protocole TTP à base de la Méthodologie O-MaSE et le langage AUML

Selon la conception réelle d'un nœud TTP, nous avons supposé qu'un nœud TTP se compose de six agents suivants :

- Un agent «CNI interface» : responsable de gérer la communication entre la partie hôte et la partie communication.
- Un agent «Membership» : responsable de gérer l'opération d'appartenance et d'acquittement.
- Un agent «Protocol processor» : responsable de l'initialisation, l'émission et la réception d'une trame.
- Un agent «Synchronizer and TDMA manager» : responsable de la gestion et la synchronisation de l'horloge du nœud.
- Un agent «LLI interface» : gère la communication entre le nœud et le bus de communication.
- Un agent «Bus gurdian» : pour la protection de bus contre des accès fautifs.

Nous avons proposé deux autres agents externes aux nœuds qui sont :

- Un agent «Communication bus» : qui transmet les trames entre les nœuds du cluster.
- Un agent «Scheduler» : responsable de la création et la mise à jour de la table TDMA lorsque un nouveau nœud s'intègre au cluster.

Cette conception permet d'exécuter les différents services du protocole TTP de façon parallèle et autonome. L'architecture proposée de notre SMA est présentée en Fig. V-1-1.

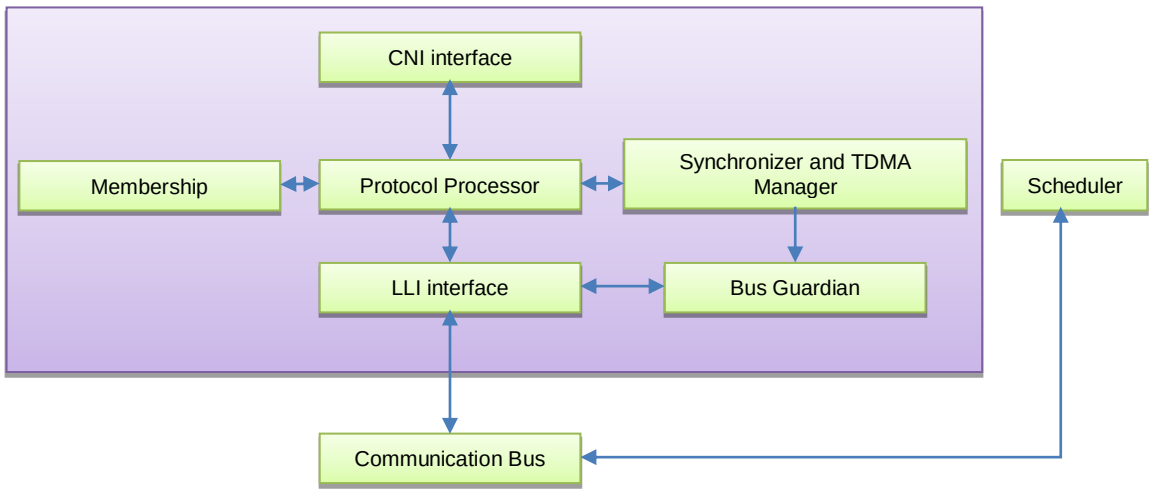


Fig. V-1-1 : Architecture Générale de SMA proposée

Le reste de cette partie présente la modélisation du protocole TTP à base de la méthodologie O-MaSE et le langage AUML puis la translation de notre modèle conceptuel en code Jade.

## 1.1. Diagramme de Buts

L'objectif global du protocole TTP est de garantir une communication tolérante aux fautes pour les applications distribuées temps réels critiques de l'automobile. Cet objectif est noté (But0). But0 peut être divisé en trois sous buts : La communication entre les nœuds (But1), une communication tolérante aux fautes (But2) et une communication déterministe (But3). But2 peut être divisé à son tour en deux sous buts : une tolérance aux fautes des nœuds (But2.1) et une tolérance aux fautes de bus de communication (But2.2) (voir Fig.V-1-2).

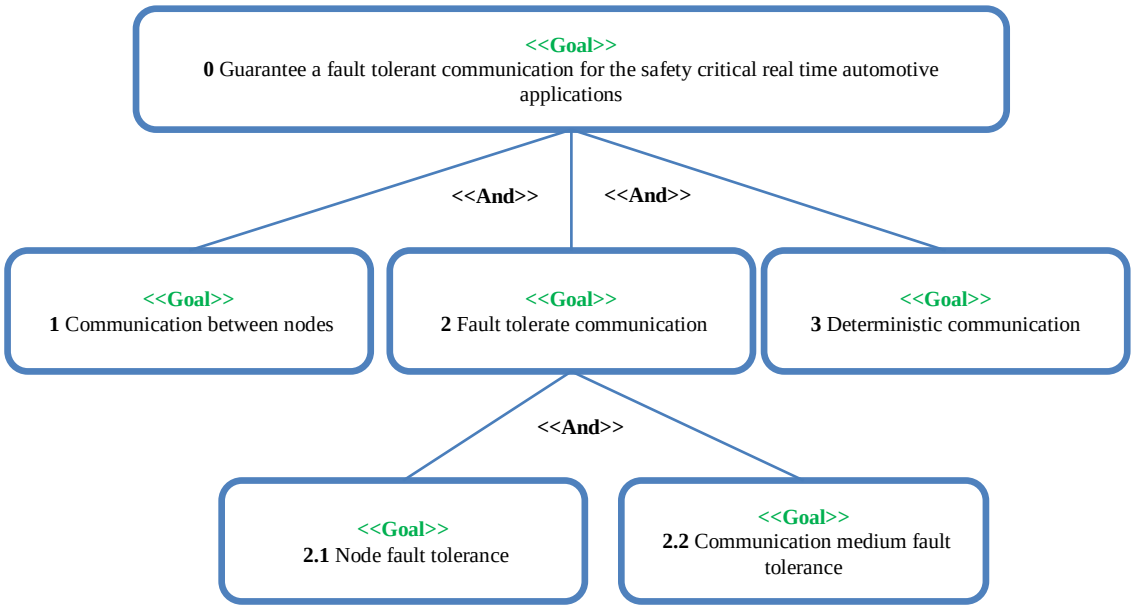


Fig.V-1-2 : Diagramme de buts du protocole TTP

## 1.2. Diagramme d'organisation

Dans notre approche, nous avons identifié deux niveaux d'abstraction de l'organisation de l'architecture TTA. Macro organisation qui représente le cluster ; l'acteur externe dans ce niveau est l'agent «Simulateur» et Micro organisation qui représente le nœud ; les acteurs externes dans ce niveau sont les agents Bus de communication, l'ordonnanceur et le Simulateur. Fig.V-1-3 présente le diagramme d'organisation au niveau macro du cluster TTA et Fig.V-1-4 présente le diagramme d'organisation au niveau micro du nœud TTA.

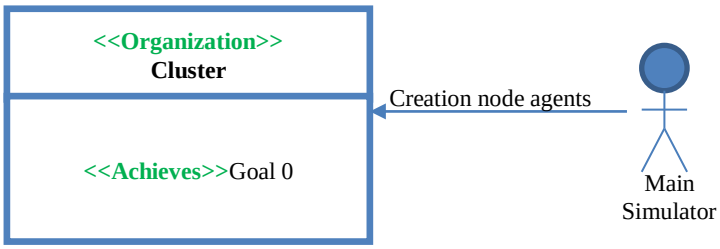


Fig. V-1-3 : Diagramme d'organisation du cluster TTA au niveau Macro

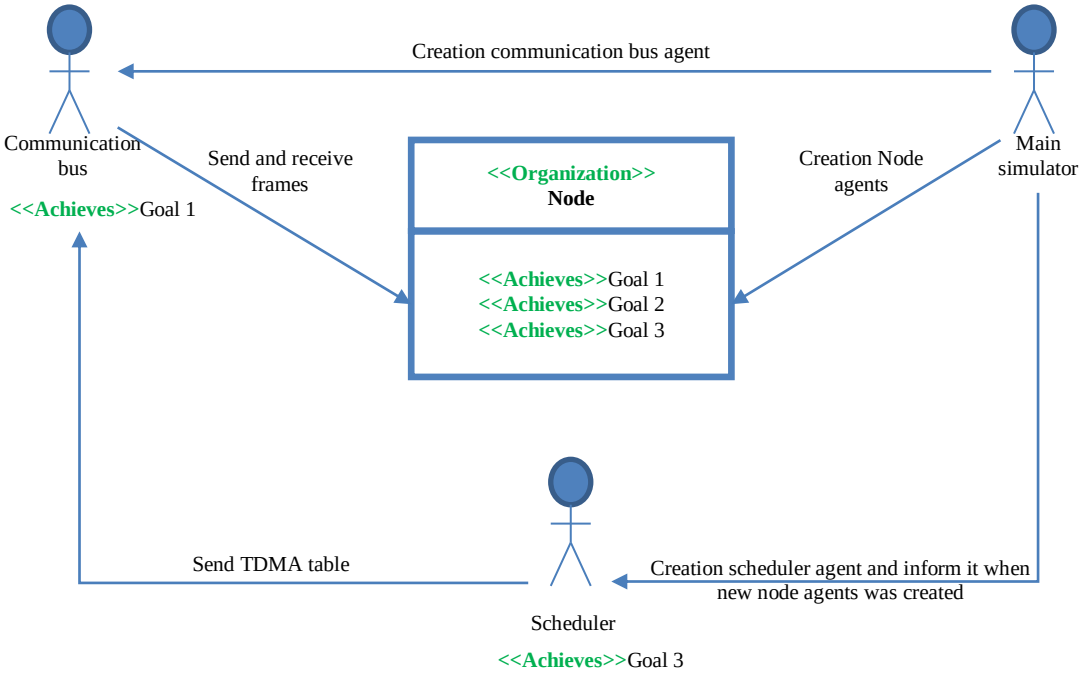


Fig. V-1-4 : Diagramme d'organisation du noeud TTA au niveau Micro

## 1.3. Diagramme de Domaine

La Trame TTP est composée d'une séquence de champs, elle est représentée sous forme d'une classe, et chacun de ses composants est aussi une sous classe où la relation entre la trame et un champ est une relation de composition. Le contenu des champs peut étre un entier, string …, donc ces champs héritent des classes de base : Integer, String, Boolean … (voir Fig. V-1-5).

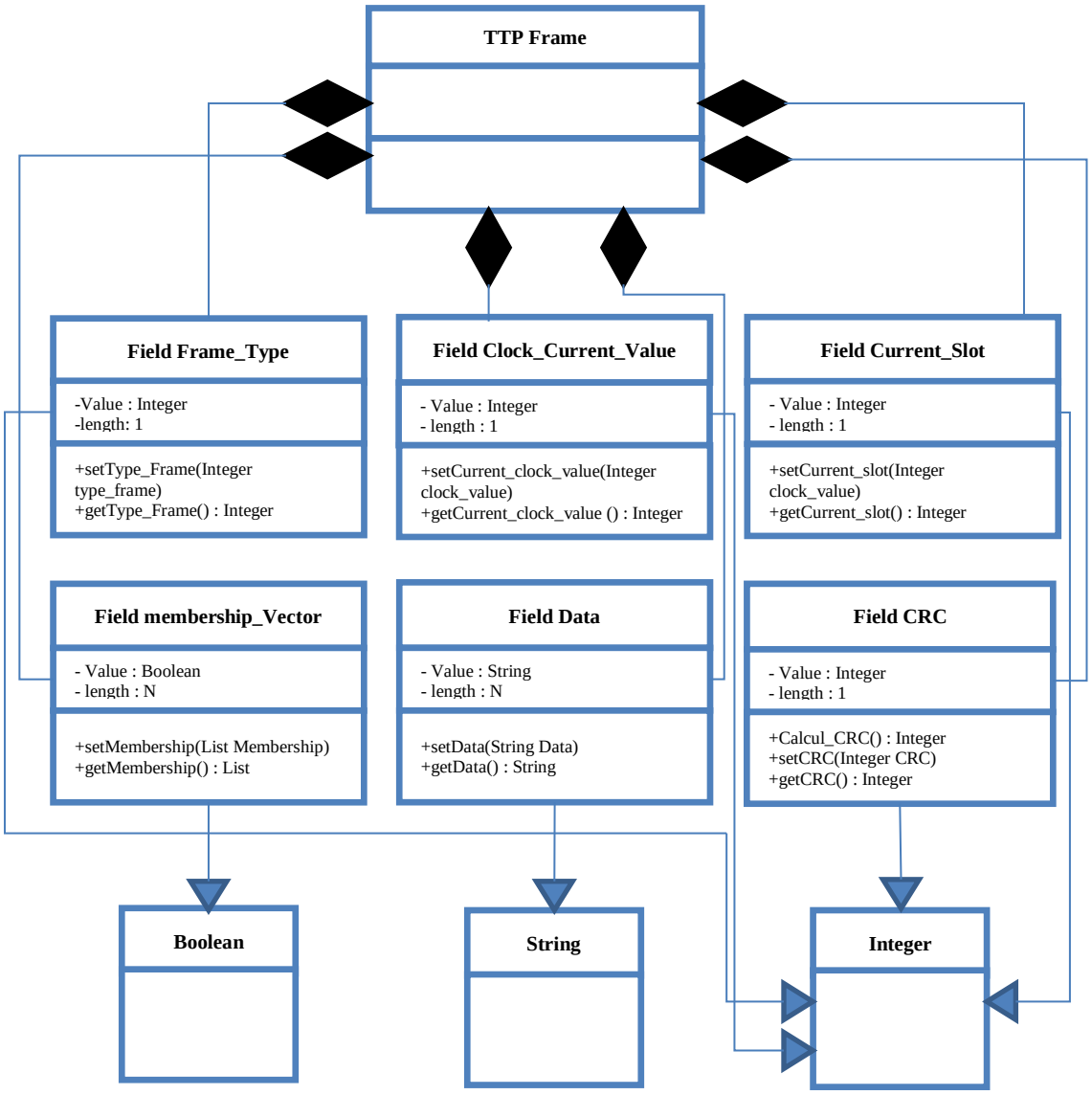


Fig. V-1-5 : Diagramme de domaine de la trame TTP

## 1.4. Diagramme de Rôles

Notre objectif est de créer le diagramme de rôles qui décrit les différents rôles pouvant être joués par les agents du système. Pour réaliser les buts décrits précédemment dans le diagramme de buts, Nous avons identifié sept rôles comme suit :

- Pour le But1 « La communication entre les nœuds» nous créons les rôles suivants : «Service de transmission», « L'émission d'une trame», « La Réception d'une trame».
- Le rôle «Service de transmission» doit pouvoir transférer les différentes trames entre les nœuds d'un même cluster, et les différentes données entre les couches d'un même nœud alors la capacité requise pour ce nœud est le plan «transmission d'informations».

- Le rôle «L'émission d'une trame» permet de remplir les différents champs de la trame à émettre par les différentes informations demandés auprès des autres rôles et l'envoi de cette trame. Ce rôle s'occupe des capacités suivantes : «Réception des informations de synchronisation» qui permet de remplir les champs de la trame suivantes : la valeur courante de l'horloge, la position courante de la table TDMA reçue et demander le vecteur membership. «Réception du vecteur membership» permet de remplir le champ vecteur membership de la trame et demander les données de la trame. «Réception des données» permet de remplir le champ de données de la trame.

- Le rôle «La Réception d'une trame» permet d'effectuer toutes les opérations nécessaires lors de la réception d'une trame. Ce rôle dépend de la capacité «Vérification de CRC et Diffusion de contenu de la trame reçue» qui permet de vérifier le CRC de la trame reçue et d'envoyer à chaque rôle les champs d'intérêt de la trame reçue.

- Pour le But2.1 «La tolérance aux fautes des nœuds» nous créons les rôles : «Service Membership», «Startup», «Synchronisation d'horloges et Gestion de la table TDMA», «La protection de bus contre des accès fautifs», «L'émission d'une trame», «La Réception d'une trame».

- Le rôle «Service Membership» offre à un nœud les services de l'acquittement implicite de ces trames qui sont déjà envoyés et la gestion de vecteur membership en fonction de la trame reçue. La première capacité requise pour ce rôle est le plan «l'Acquittement Implicite» qui permet d'exécuter l'algorithme d'acquittement du protocole TTP décrit dans le deuxième chapitre (Partie I). De plus, pour pouvoir mettre à jour le vecteur membership en cas d'intégration de nouveaux nœuds, une deuxième capacité est donc requise, prise en charge par le plan «Mise à jour Vecteur Membership». Pour pouvoir mettre à jour le bit de l'émetteur de la trame dans le vecteur membership en cas de la réception d'une trame erronée, une troisième capacité est requise, prise en charge par le plan «Mise à jour vecteur membership en cas de la réception d'une trame erronée». Enfin, la quatrième capacité requise pour ce rôle est «Envoi de vecteur membership», qui permet d'offrir le vecteur membership en cas d'une demande.

- Le rôle «Startup» requiert la capacité «Initialisation de cluster» qui permet d'exécuter l'algorithme d'initialisation de cluster du protocole TTP décrit dans le deuxième chapitre (partie I) précédent.

- Le rôle «Synchronisation d'horloges et Gestion de la table TDMA» doit pouvoir garantir d'une part une vue commune de l'horloge et la position de la table TDMA entre tous les nœuds du cluster et d'autre part appliquer la stratégie TDMA d'accès au bus de communication décrite dans le deuxième chapitre (partie I). Ce rôle nécessite les capacités suivantes : «Réception d'une valeur d'horloge et une position de la table TDMA» pour remplacer la valeur courante de l'horloge et la position courante de la table TDMA par celle de CState de la trame reçue en cas de la réception d'une trame

normale ou d'initialisation. «Réception de la table TDMA» pour pouvoir remplacer l'ancienne table TDMA par la nouvelle table TDMA en cas de la réception d'une telle table. Enfin «Envoi de la valeur courante d'horloge et la position courante de la table TDMA» en cas où le slot courant est réservé pour l'émission d'une trame.

- Le rôle «mécanisme de redondance» permet de réaliser le But2.1 «La tolérance aux fautes des nœuds» et le But 2.2 «La tolérance aux fautes de bus de communication». Ce rôle permet d'appliquer le principe de redondance expliqué dans le deuxième chapitre (partie I).

- Le But3 «communication déterministe» est réalisé par le rôle «ordonnancement déterministe» et le rôle «Synchronisation d'horloges et Gestion de la table TDMA». Le rôle «ordonnancement déterministe» permet la création de la table TDMA et l'ajout des nouveaux nœuds dans la table tout en prisant en compte les contraintes temporelles de chaque nœud. Ce rôle nécessite la capacité «Réception information d'entrer d'un nouveau nœud» qui permet d'ajouter un nouveau nœud dans la table TDMA crée par ce rôle. De plus, ce rôle requiert d'autres capacités : «Réception d'une trame» pour garantir la synchronisation de l'horloge et la position de la table TDMA avec les autres nœuds du cluster et la capacité «Envoi de la table TDMA» qui permet d'envoyer la nouvelle table TDMA en cas de rajout de nouveaux nœuds aux autres nœuds d'un même cluster (Fig. V-1-6 présente le diagramme de rôles du protocole TTP).

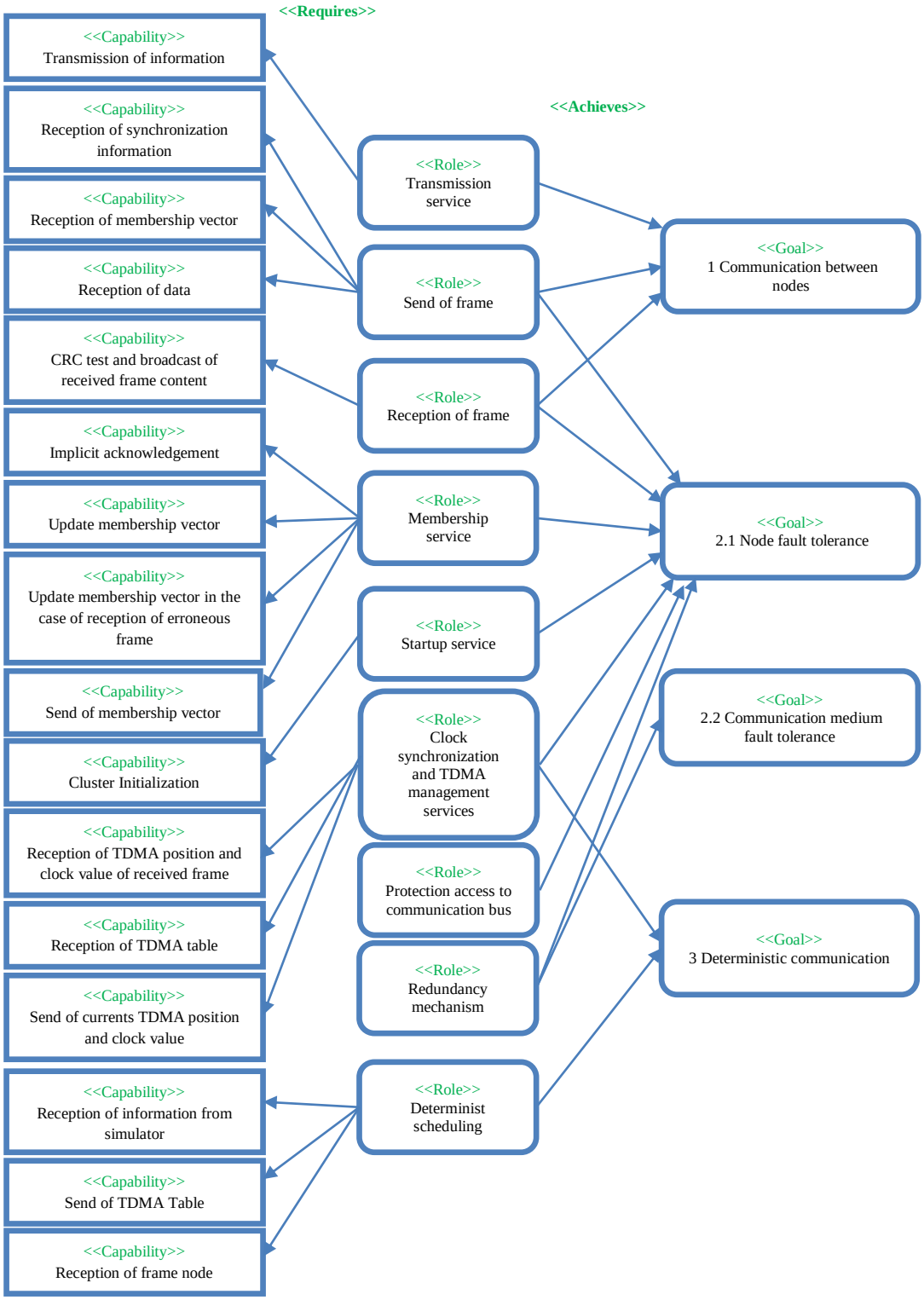


Fig. V-1-6 : Diagramme de rôles de protocole TTP

## 1.5. Diagramme de classes d'agent

Nous avons identifié huit classes d'agent qui fonctionnent simultanément. Un agent nommé «Noyau Protocole» jouant les rôles : «Startup», «L'émission d'une trame», «Réception d'une trame». L'agent «Membership» jouant le rôle «Service Membership». Un agent nommé «Synchronisateur et Gestionnaire de la table TDMA» jouant le rôle : «Synchronisation d'horloges et Gestion de la table TDMA». Un agent nommé «Bus gardian» jouant le rôle «La protection de bus contre des accès fautifs». Les Agents : «Bus de communication», «LLI interface» et «CNI interface» jouent le rôle : «Service de Transmission». Enfin l'agent «Ordonnanceur» jouant le rôle «Ordonnancement déterministe». Le rôle «Mécanisme de redondance» est joué par l'instanciation de deux instances de la classe d'agent «bus de communication» et des instances de chaque classe d'agent définie (Fig. V-1-7 présente le diagramme d'agents de protocole TTP).

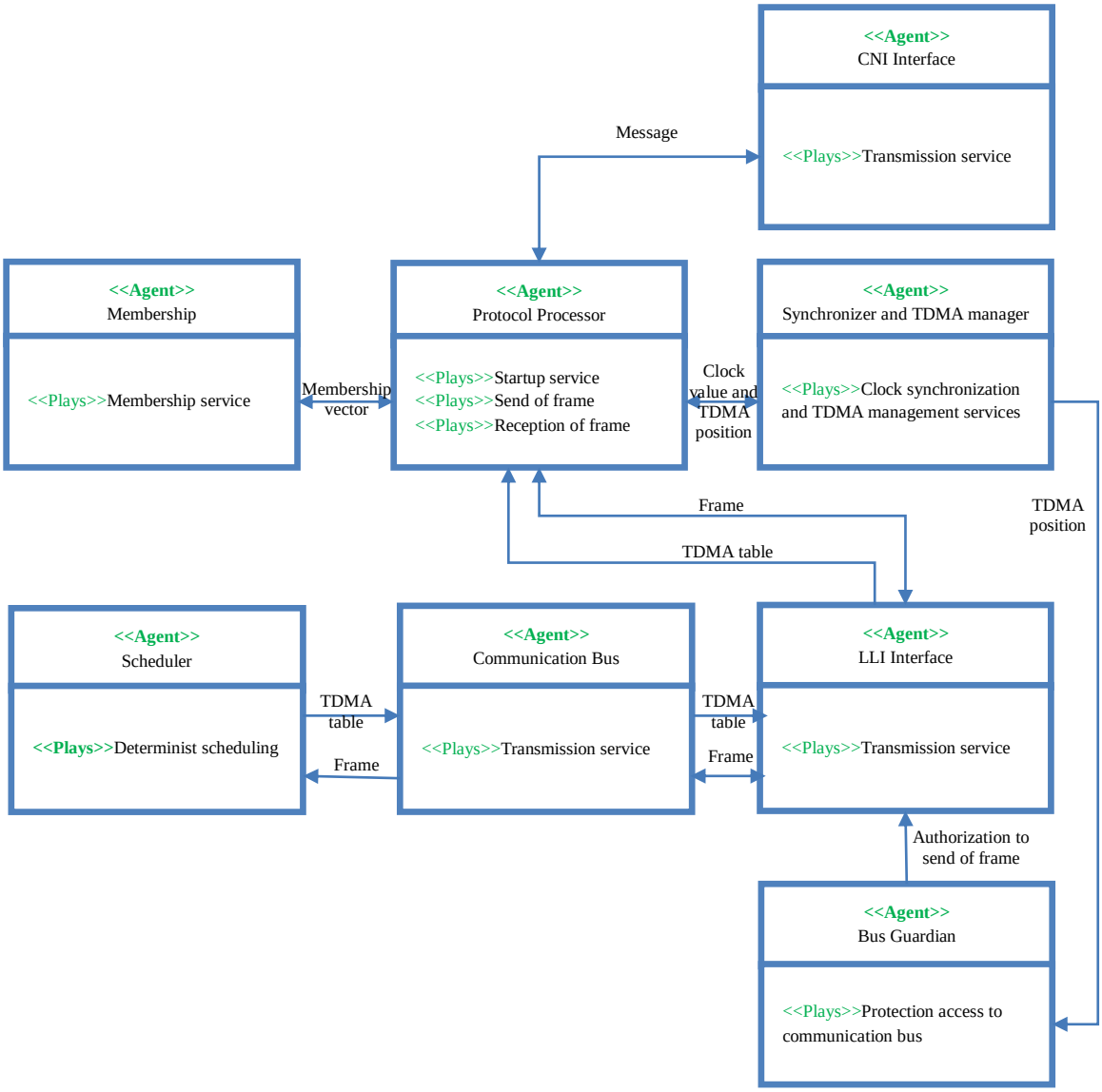


Fig. V-1-7 Diagramme de classes d'agent du protocole TTP

## 1.6. Diagrammes de plan

### 1.6.1. Plan «transmission d'informations» du rôle «Service de transmission»

Suite à la réception d'une information qui peut être une trame (normale ou d'initialisation), une table TDMA, un message (en cas d'une transmission au niveau de l'interface CNI), le rôle «Service de transmission» envoie cette information aux autres rôles (voir Fig. V-1-8).

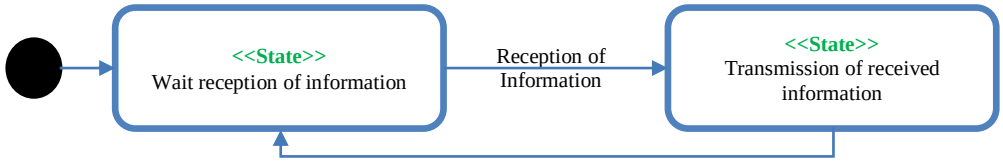


Fig. V-1-8 : Diagramme de plan «transmission d'informations»

### 1.6.2. Diagrammes de plan pour les capacités du rôle «L'émission d'une trame»

#### 1.6.2.1. Plan «Réception des informations de synchronisation»

Suite à la réception des informations de synchronisation auprès l'agent qui joue le rôle «Synchronisation d'horloges et Gestion de la table TDMA» qui représentent la valeur courante de l'horloge et la position courante de la table TDMA, l'agent qui joue le rôle «L'émission d'une trame» constate que le slot courant correspond au moment d'envoi de sa trame. En effet, il remplit les champs de synchronisation de la trame à émettre par ceux reçus et il envoie une demande d'un vecteur membership à l'agent qui joue le rôle «Service membership» (voir Fig. V-1-9).

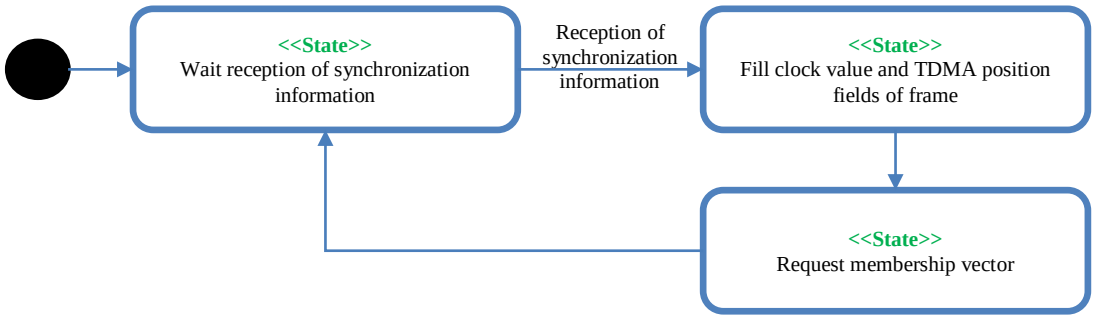


Fig. V-1-9: Diagramme de plan «Réception des informations de synchronisation»

#### 1.6.2.2. Plan «Réception du vecteur membership»

Suite à la réception de vecteur membership auprès l'agent qui joue le rôle «Service Membership», l'agent qui joue le rôle «L'émission d'une trame» remplit le champ de vecteur membership de la trame à émettre par celui reçu, si cette trame qui va être transmise est une trame d'initialisation alors il envoie la trame directement à l'agent «Interface LLI», si la trame est une trame normale alors il envoie une demande des données à l'agent «Interface CNI» (voir Fig. V-1-10).

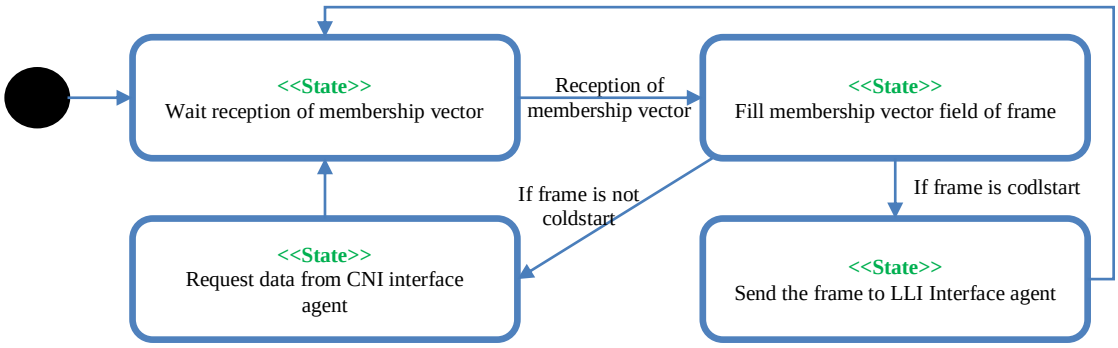


Fig. V-1-10 : Diagramme de plan «Réception du vecteur membership»

### 1.6.2.3. Plan «Réception des données»

Suite à la réception d'un message qui contient les données auprès l'agent «Interface CNI», l'agent qui joue le rôle «L'émission d'une trame» remplit le champ de données de la trame à émettre par celui reçue et il envoie cette trame à l'agent «LLI Interface» (après de l'insertion des champs de type de trame et CRC) (voir Fig. V-1-11).

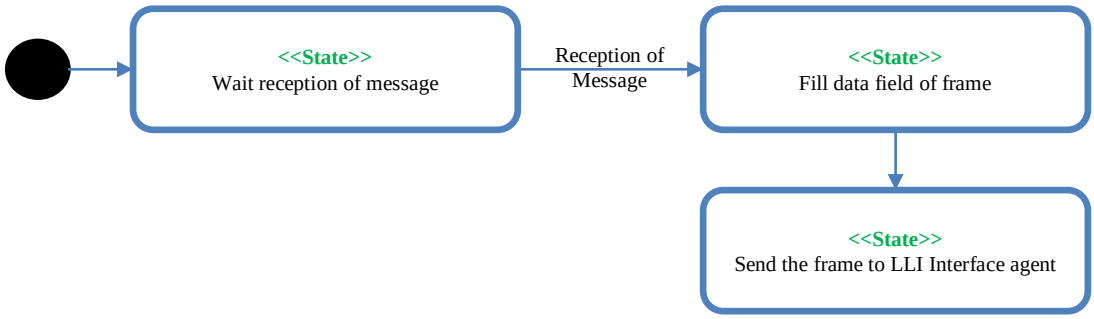


Fig. V-1-11 : Diagramme de plan «Réception des données»

Dans les deux derniers plans, les champs de la trame à émettre, les champs type de la trame et CRC sont remplis juste avant l'envoi de la trame. Ces tâches sont inclues dans l'état «Envoie de trame à l'agent Interface LLI».

### 1.6.3. Plan «Vérification CRC et diffusion de contenu de la trame reçue» du rôle «La Réception d'une trame»

Suite à la réception d'une trame, l'agent qui joue le rôle «Réception d'une trame» vérifie le CRC de la trame reçue, si elle est erronée il envoie une information à l'agent qui joue le rôle «Membership Service» pour lui dire que la trame reçue est erronée. Si la trame est correcte, il vérifié l'identité de la trame, Dans le cas où la trame est une trame normale ou d'initialisation alors il envoie le champ de vecteur membership à l'agent qui joue le rôle «Membership Service», les champs de synchronisation à l'agent qui joue le rôle «Synchronisation d'horloges et Gestion de la table TDMA» et les champs de données à l'agent «Interface CNI». Dans le cas où la trame contient la table TDMA alors il envoie cette nouvelle table à l'agent qui joue le rôle « Synchronisation d'horloges et Gestion de la

table TDMA» et un nouveau vecteur de nœuds à l'agent qui joue le rôle «Membership Service» (voir Fig. V-1-12).

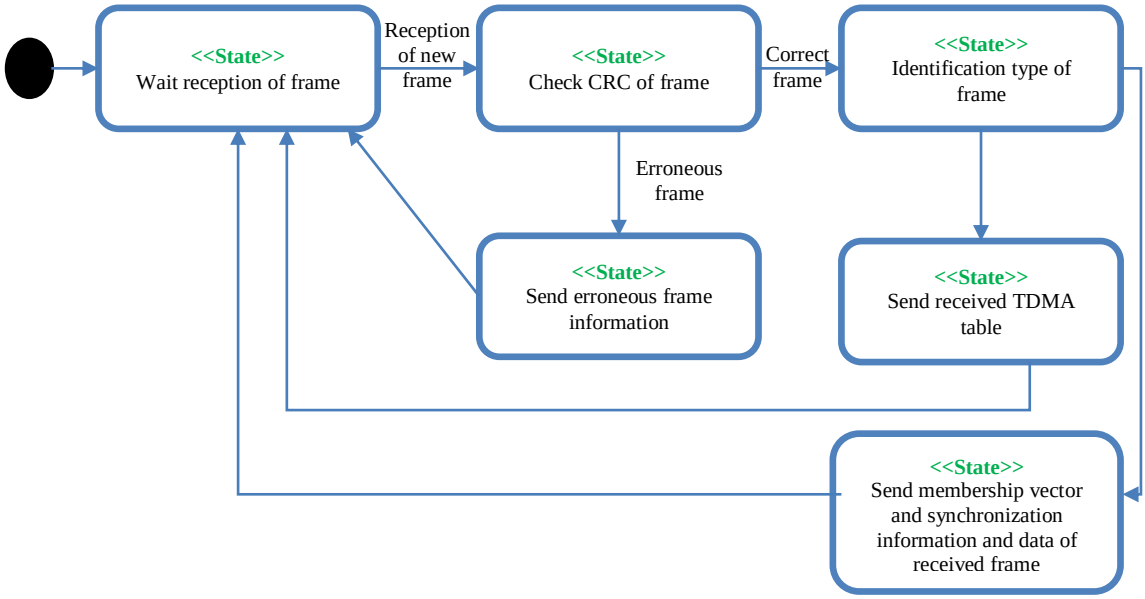


Fig. V-1-12: Diagramme de plan «Vérification CRC et Diffusion de contenu de la trame reçue»

### 1.6.4. Diagrammes de plan pour les capacités du rôle «Service Membership»

#### 1.6.4.1. Plan «l'Acquittement Implicite»

Ce diagramme de plan représente l'algorithme de membership et d'acquittement du protocole TTP expliqué dans le deuxième chapitre (partie I) (voir Fig. V-1-13).

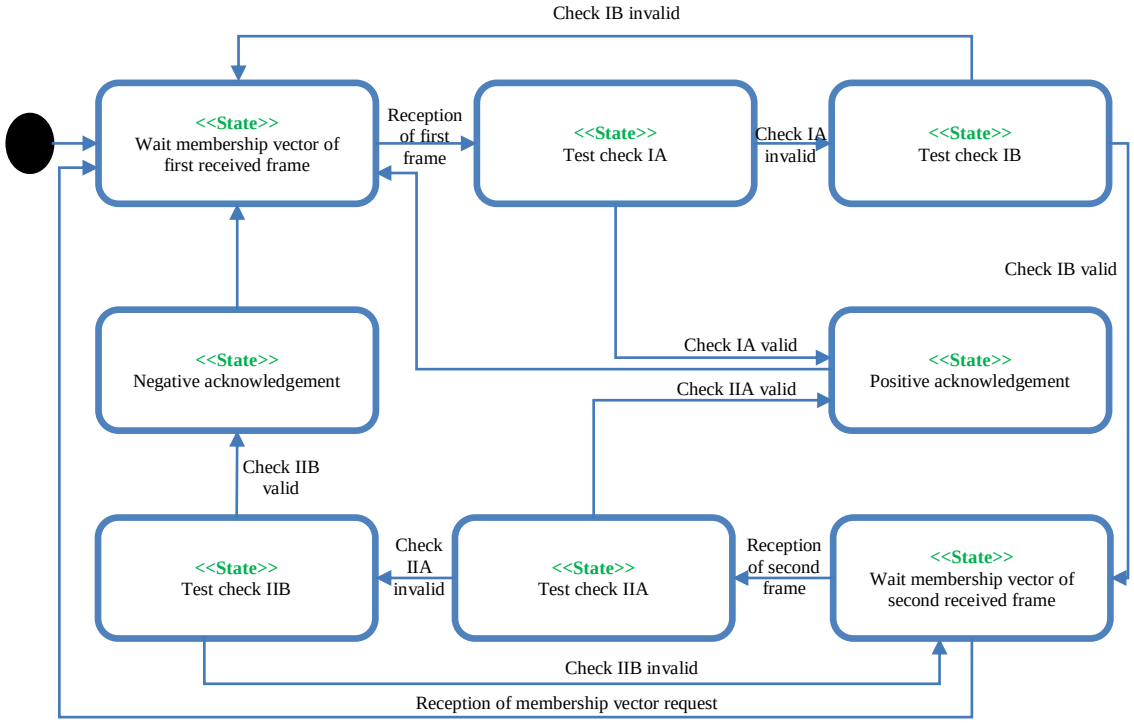


Fig. V-1-13 : Diagramme de plan «l'Acquittement Implicite»

### 1.6.4.2. Plan «Mise à jour vecteur membership»

Suite à la réception d'un nouveau vecteur des nœuds auprès de l'agent qui joue le rôle «Réception d'une trame» en cas d'intégration des nouveaux nœuds au cluster, l'agent qui joue le rôle «Service membership» remplace son vecteur de nœuds par celui reçu et ajoute des nouvelles cases dans le vecteur membership attribuées aux nouveaux nœuds (voir Fig. V-1-14).

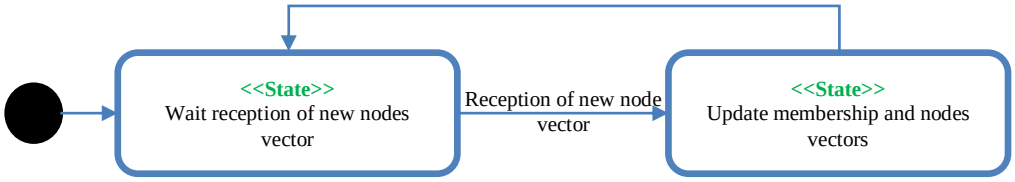


Fig. V-1-14 : Diagramme de plan «Mise à jour vecteur membership»

### 1.6.4.3. Plan «Mise à jour vecteur membership en cas de la réception d'une trame erronée»

Suite à la réception d'une information stipulant que la trame reçue est erronée auprès de l'agent qui joue le rôle «Réception d'une trame», l'agent qui joue le rôle «Service membership» met la case de son vecteur membership qui est attribuée à l'émetteur de la trame reçue à 0 (faut) (voir Fig. V-1-15).

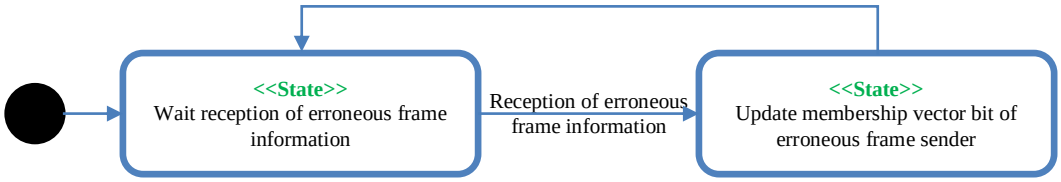


Fig. V-1-15: Diagramme de plan «Mise à jour vecteur membership en cas d'une trame erronée»

### 1.6.4.4. Plan «Envoi de vecteur membership»

Suite à la réception d'une demande d'un vecteur membership auprès de l'agent qui joue le rôle «Emission d'une trame», l'agent qui joue le rôle «Service membership» envoie le vecteur membership (voir Fig. V-1-16).

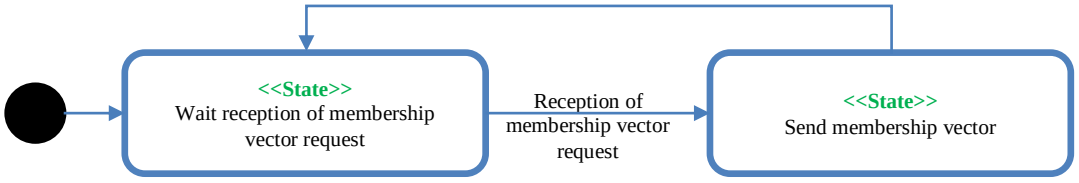


Fig. V-1-16 : Diagramme de plan «Envoi de vecteur membership»

### 1.6.5. Plan «Initialisation de cluster» du rôle «Startup»

Ce diagramme de plan modélise l'algorithme de startup du protocole TTP expliqué dans le deuxième chapitre (partie I) (voir Fig. V-1-17).

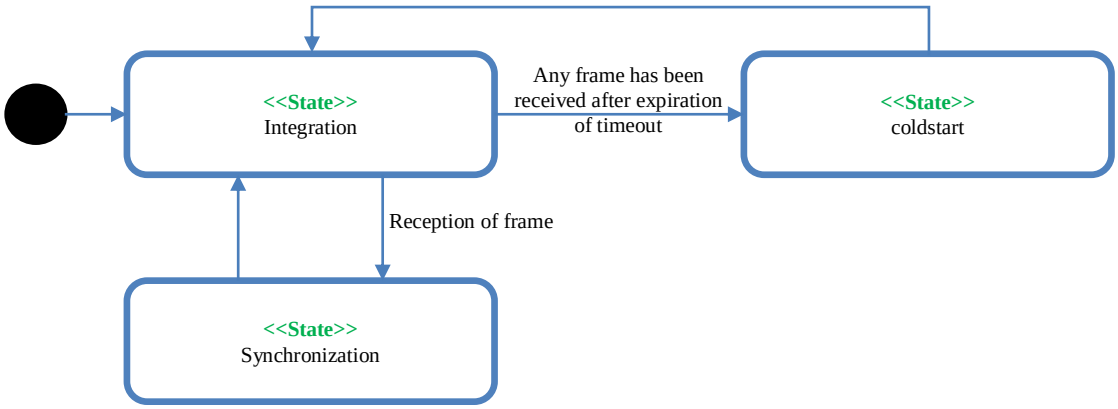


Fig. V-1-17 : Diagramme de plan «Initialisation de cluster»

### 1.6.6. Diagrammes de plan pour les capacités du rôle «Synchronisation d'horloges et Gestion de la table TDMA»

#### 1.6.6.1. Plan «Réception d'une valeur d'horloge et une position de la table TDMA»

Suite à la réception d'une valeur d'horloge et une position de la table TDMA de la trame reçue auprès de l'agent qui joue le rôle «Réception d'une trame», l'agent jouant le rôle «Synchronisation d'horloges et Gestion de la table TDMA» met à jour la valeur courante de son horloge et la position courante de sa table TDMA par celles reçues (voir Fig. V-1-18).

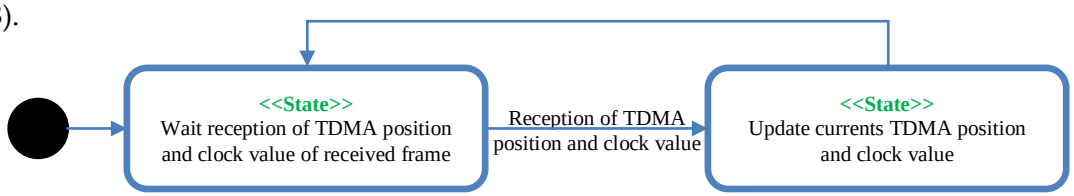


Fig. V-1-18 : Diagramme de plan «Réception d'une valeur d'horloge et une position de la table TDMA»

#### 1.6.6.2. Plan «Réception de la table TDMA»

Suite à la réception d'une nouvelle table TDMA auprès de l'agent qui joue le rôle «Réception d'une trame», l'agent jouant le rôle «Synchronisation d'horloges et Gestion de la table TDMA» remplace sa table TDMA par celle reçue (voir Fig. V-1-19).

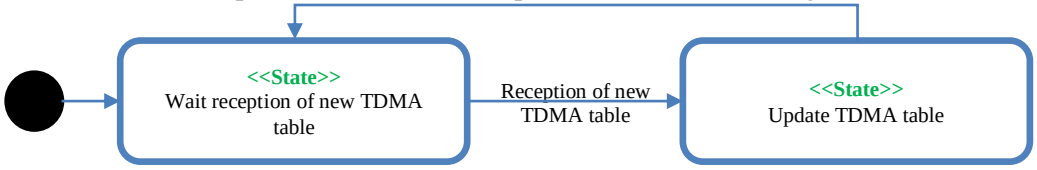


Fig. V-1-19 : Diagramme de plan «Réception de la table TDMA»

#### 1.6.6.3. Plan «Envoi de la valeur courante d'horloge et la position courante de la table TDMA»

L'agent jouant le rôle «Synchronisation d'horloges et Gestion de la table TDMA» fait parcourir à tout moment la table TDMA en fonction de la valeur courante de l'horloge, il cherche si la valeur courante de la position de la table TDMA correspond au slot de l'émission d'une trame, alors il envoie la valeur courante de l'horloge et la position courante de la table TDMA à l'agent qui joue le rôle «Emission d'une trame» (voir Fig. V-1-20).

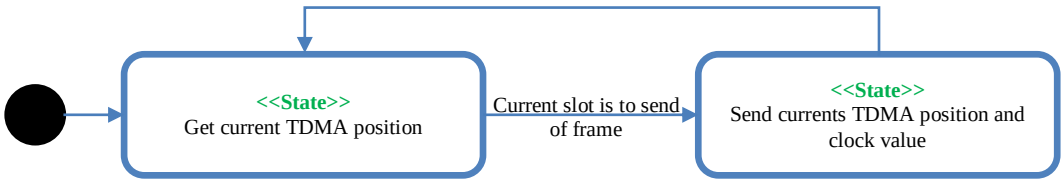


Fig. V-1-20: Diagramme de plan «Envoi de la valeur courante d'horloge et la position courante de la table TDMA»

### 1.6.7. Diagrammes de plan pour les capacités du rôle «ordonnancement déterministe»

#### 1.6.7.1. Plan «Réception information d'entrer d'un nouveau nœud»

Suite à la réception d'une information stipulant qu'un nouveau nœud a été intégré dans le cluster, l'agent jouant le rôle «ordonnancement déterministe» doit ajouter les attributs de ce nouveau nœud (identification, temps début slot, temps fin slot) dans la table TDMA (voir Fig. V-1-21).

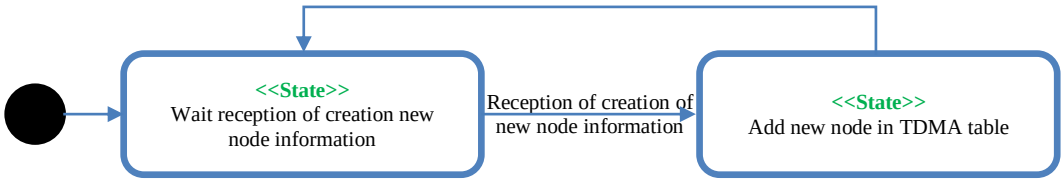


Fig. V-1-21 : Diagramme de plan «Réception information d'entrer d'un nouveau nœud»

#### 1.6.7.2. Plan «Réception d'une trame»

Suite à la réception d'une trame, l'agent jouant le rôle «ordonnancement déterministe» doit mettre à jour la valeur courante de son horloge et la position courante de sa table TDMA avec les mêmes valeurs de la trame reçue (voir Fig. V-1-22).

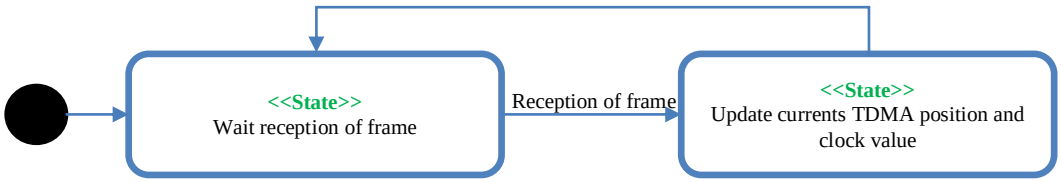


Fig. V-1-22: Diagramme de plan «Réception d'une trame»

#### 1.6.7.3. Plan «Envoi de la table TDMA»

Lorsque la position de la table TDMA de l'agent jouant le rôle «ordonnancement déterministe» se pointe sur le slot de la transmission d'une nouvelle table TDMA par cet agent, ce dernier envoie la table TDMA à tous les nœuds du cluster (voir Fig. V-1-23).

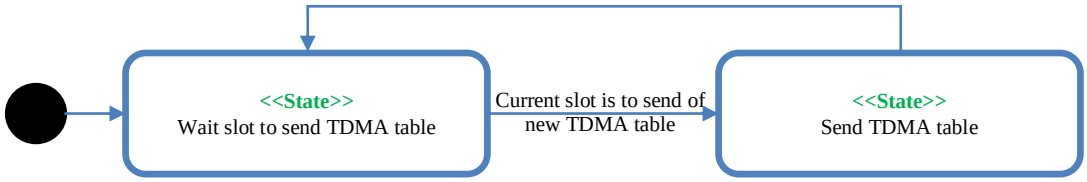


Fig. V-1-23: Diagramme de plan «Envoi de la table TDMA»

### 1.7. Diagramme de Protocole de l'AUML

Nous proposons deux diagrammes de protocole : «Diagramme de protocole d'émission» et «Diagramme de protocole de réception».

#### 1.7.1. Diagramme de protocole d'émission

Ce protocole est lancé lorsque l'agent «Synchronisateur et gestionnaire de la table TDMA» exécute le plan «Envoi de la valeur courante d'horloge et la position courante de la table TDMA» pour informer les agents «Noyau de Protocole» et «Bus Guardian» que le slot courant est attribué à leur nœud pour émettre une trame. Quand l'agent «Noyau de Protocole» qui joue le rôle «L'émission d'une trame» reçoit ce message, il exécute le plan «Réception des informations de synchronisation». Lorsque cet agent reçoit le vecteur membership depuis l'agent «Membership», il exécute le plan «Réception du vecteur membership». Lorsqu'il reçoit un message, il exécute le plan «Réception des données» et il envoie la trame à l'agent «Interface LLI». Lorsque ce dernier reçoit la trame il envoie une demande d'autorisation auprès de l'agent «Bus Guardian» et il attend. Lorsqu'il reçoit cette autorisation il envoie la trame à l'agent «Bus de Communication» (voir Fig. V-1-24).

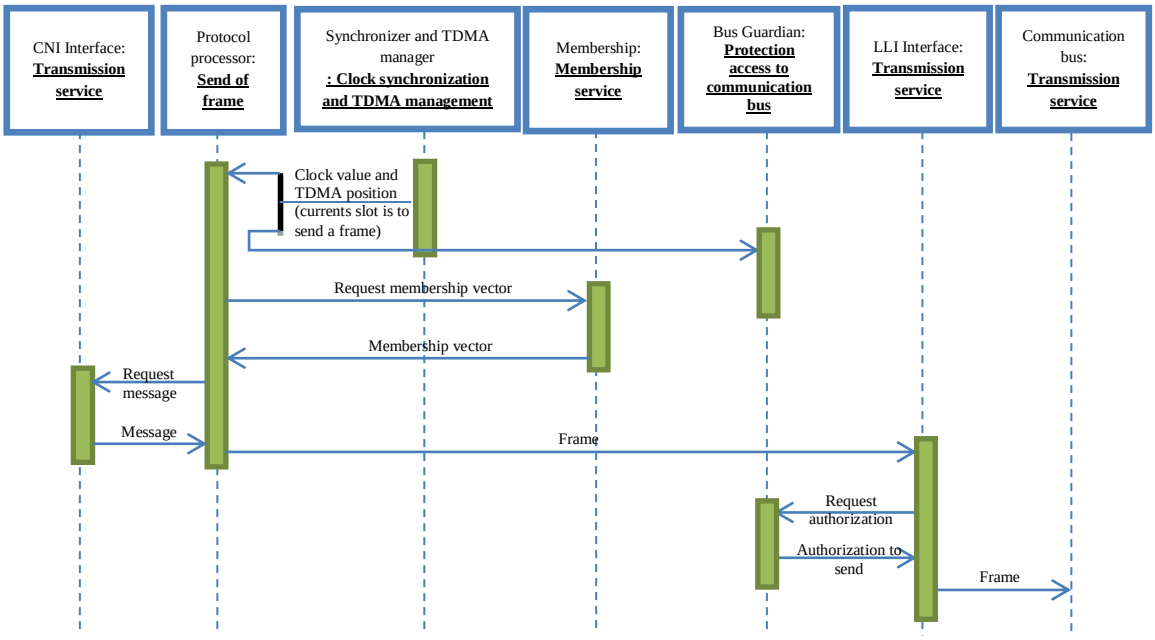


Fig. V-1-24 : Diagramme de protocole d'Emission

### 1.7.2. Diagramme de protocole de réception

Ce protocole est lancé lorsque l'agent «Interface LLI» reçoit une trame auprès de l'agent «Bus de communication». Il exécute le plan capacité «transmission d'informations» pour envoyer la trame reçue à l'agent «Noyau de protocole» qui joue le rôle «La Réception d'une trame», l'agent «Noyau de protocole» exécute de sa part le plan «Vérification de CRC et Diffusion de contenu de la trame reçue». En cas où la trame reçue est erronée, l'agent «Membership» reçoit l'information disant que la trame reçue est erronée alors il exécute le plan «Mise à jour vecteur membership en cas d'une trame erronée». En cas où la trame est une trame normale l'agent «Membership» exécute le plan «l'Acquittement Implicite» à cause de la réception du champ vecteur membership, l'agent «Synchronisateur et Gestionnaire de la table TDMA» exécute le plan «Réception d'une valeur d'horloge et une position de la table TDMA» à cause de la réception des champs des informations de synchronisation et l'agent «Interface CNI» exécute le plan «transmission d'informations» lorsqu'il reçoit un message. En cas où la trame contient une nouvelle table TDMA, l'agent «Membership» exécute le plan «Mise à jour Vecteur Membership» à cause de la réception d'un nouveau vecteur des nœuds, et l'agent «Synchronisateur et Gestionnaire de la table TDMA» exécute le plan «Réception de la table TDMA» à cause de la réception d'une nouvelle table TDMA (voir Fig. V-1-25).

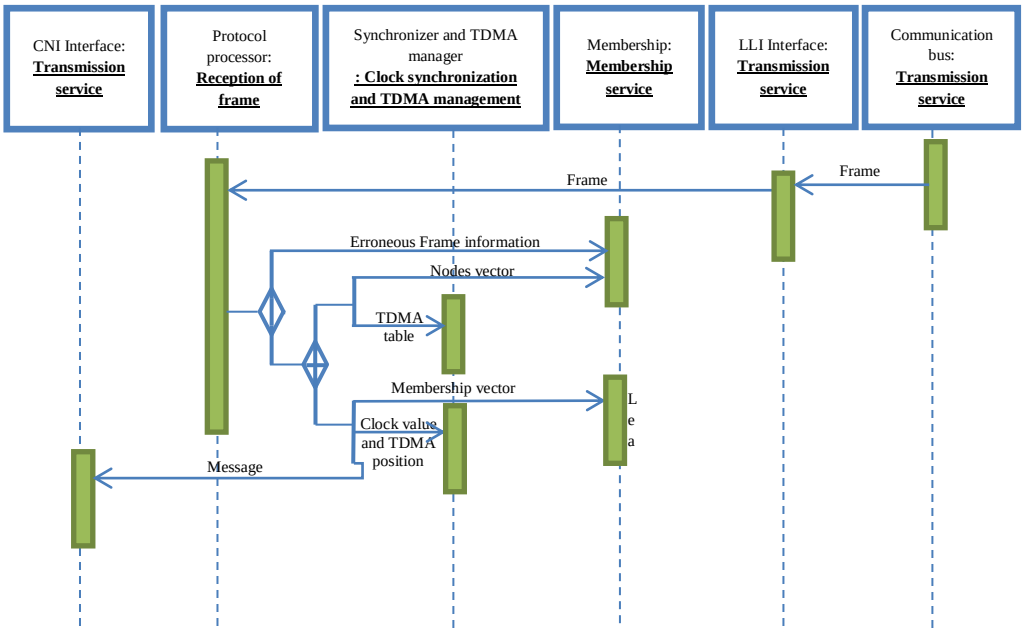


Fig. V-1-25: Diagramme de protocole de Réception

# 2. Translation du modèle conceptuel de protocole TTP en code JADE

## 2.1. Translation de diagramme de Domaine

```
packageTTPAgents;

importjade.content.onto.*;
importjade.content.schema.*;
importjade.content.onto.BasicOntology;
publicclassTTP_Frame_Ontologyextends Ontology{
publicfinalstatic String ONTOLOGY_NAME = "TTP-Frame-Ontologie";
publicfinalstatic String Frame = "Frame";
publicfinalstatic String Frame_Type= "Frame_Type";
publicfinalstatic String Current_Clock_Value= "Current_Clock_Value";
publicfinalstatic String Current_Slot= "Current_Slot ";
publicfinalstatic String Membership_Vector= "Membership_Vector ";
publicfinalstatic String Data = "Data";
publicfinalstatic String CRC= " CRC";

publicstaticfinal String OWNS = "Owns";
publicstaticfinal String OWNS_OWNER = "owner";
publicstaticfinal String OWNS_ITEM = "item";

privatestatic Ontology theInstance = newTTP_Frame_Ontology();
publicstatic Ontology getInstance() {
returntheInstance;
                                        }

privateTTP_Frame_Ontology(){
super(ONTOLOGY_NAME, BasicOntology.getInstance());
try{
add(newConceptSchema(Frame),TTP_Frame.class);
add(newPredicateSchema(OWNS), Owns.class);

ConceptSchemacs = (ConceptSchema) getSchema(Trame);
cs.add(Frame_Type, (PrimitiveSchema) getSchema(BasicOntology.INTEGER),);
cs.add(Current_Clock_Value, (PrimitiveSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);
cs.add(Current_Slot, (PrimitiveSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);
cs.add(Membership_Vector, (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN),0,ObjectSchema.UNLIMITED);
cs.add(Data, (PrimitiveSchema) getSchema(BasicOntology.STRING),ObjectSchema.OPTIONAL);
cs.add(CRC, (PrimitiveSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

PredicateSchemaps = (PredicateSchema) getSchema(OWNS);
ps.add(OWNS_OWNER, (ConceptSchema) getSchema(BasicOntology.AID));
ps.add(OWNS_ITEM, (ConceptSchema) getSchema(Frame));
} catch (OntologyExceptionoe) {oe.printStackTrace();}
}

}
```

## 2.2. Translation de diagramme de plan «transmission d'informations» du rôle «Service de transmission»

```
Transmission_of_Informations = newFSMBehaviour(this) {
};
// States
Transmission_of_Informations.registerFirstState(newWait_Reception_Information(),"Wait_Reception_Information");
Transmission_of_Informations.registerState(new Transmission_Received_Information(),"Transmission_Received_Information");
// Transitions
Transmission_of_Informations.registerTransition("Wait_Reception_Information","Wait_Reception_Information",1);
Transmission_of_Informations.registerTransition("Wait_Reception_Information","Transmission_Received_Information",0);
Transmission_of_Informations.registerDefaultTransition("Transmission_Received_Information","Wait_Reception_Information");
```

## 2.3. Translation des diagrammes de plan du rôle «L'émission d'une trame»

### 2.3.1. Translation de diagramme de plan «Réception des informations de synchronisation»

```
Reception_Synchronization_Information_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Synchronization_Information_Behaviour.registerFirstState(new
Wait_Reception_Synchronization_Information(),"Wait_Reception_Synchronization_Information");
Reception_Synchronization_Information_Behaviour.registerState(new
Fill_Clock_TDMAPosition_Frame_Fields(),"Fill_Clock_TDMAPosition_Frame_Fields");
Reception_Synchronization_Information_Behaviour.registerState(newRequest_Membership_Vector(),"Request_Membership_Vector");
// Transitions
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Synchronization_Information","Wait_Reception_Synchroniz
ation_Information",0);
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Synchronization_Information","Fill_Clock_TDMAPosition_
Frame_Fields",1);
Reception_Synchronization_Information_Behaviour.registerDefaultTransition("Fill_Clock_TDMAPosition_Frame_Fields","Request_Membership_Ve
ctor");
Reception_Synchronization_Information_Behaviour.registerDefaultTransition("Request_Membership_Vector","Wait_Reception_Synchronization_Info
rmation");
```

### 2.3.2. Translation de diagramme de plan «Réception du vecteur membership»

```
Reception_Membership_Vector_Behaviour= newFSMBehaviour(this) {
};
                                        // States
Reception_Membership_Vector_Behaviour.registerFirstState(new
Wait_Reception_Membership_Vector(),"Wait_Reception_Membership_Vector");
Reception_Membership_Vector_Behaviour.registerState(new Fill_Membership_Vector_Frame_Field(),"Fill_Membership_Vector_Frame_Field");
Reception_Membership_Vector_Behaviour.registerState(newRequest_Message_From_CNI(),"Request_Message_From_CNI");
Reception_Membership_Vector_Behaviour.registerState(newSend_Frame_To_LLI(),"Send_Frame_To_LLI");
// Transitions
Reception_Membership_Vector_Behaviour.registerTransition("Wait_Reception_Membership_Vector","Wait_Reception_Membership_Vector",0);
Reception_Membership_Vector_Behaviour.registerTransition("Wait_Reception_Membership_Vector","Fill_Membership_Vector_Frame_Field",1);
Reception_Membership_Vector_Behaviour.registerTransition("Fill_Membership_Vector_Frame_Field","Request_Message_From_CNI",1);
Reception_Membership_Vector_Behaviour.registerDefaultTransition("Request_Message_From_CNI","Wait_Reception_Membership_Vector");

Reception_Membership_Vector_Behaviour.registerTransition("Fill_Membership_Vector_Frame_Field","Send_Frame_To_LLI",0);
Reception_Membership_Vector_Behaviour.registerDefaultTransition("Send_Frame_To_LLI","Wait_Reception_Membership_Vector");
```

### 2.3.3. Translation de diagramme de plan «Réception des données»

```
Reception_Message_From_CNI_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Message_From_CNI_Behaviour.registerFirstState(newWait_Reception_Message(),"Wait_Reception_Message");
Reception_Message_From_CNI_Behaviour.registerState(newFill_Data_Frame_Field(),"Fill_Data_Frame_Field");
Reception_Message_From_CNI_Behaviour.registerState(newSend_Frame_To_LLI(),"Send_Frame_To_LLI");
// Transitions
Reception_Message_From_CNI_Behaviour.registerTransition("Wait_Reception_Message","Wait_Reception_Message",0);
Reception_Message_From_CNI_Behaviour.registerTransition("Wait_Reception_Message","Fill_Data_Frame_Field",1);
Reception_Message_From_CNI_Behaviour.registerDefaultTransition("Fill_Data_Frame_Field","Send_Frame_To_LLI");
Reception_Message_From_CNI_Behaviour.registerDefaultTransition("Send_Frame_To_LLI","Wait_Reception_Message");
```

## 2.4. Translation de diagramme de Plan «Vérification CRC et diffusion de contenu de la trame reçue» du rôle «La Réception d'une trame»

```
Reception_Frame_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Frame_Behaviour.registerFirstState(newWait_Reception_Frame(),"Wait_Reception_Frame");
Reception_Frame_Behaviour.registerState(newVerification_CRC_Frame(),"Verification_CRC_Frame");
Reception_Frame_Behaviour.registerState(newIdentification_Frame_Type(),"Identification_Frame_Type");
Reception_Frame_Behaviour.registerState(new Send_Erroneous_Frame_Information(),"Send_Erroneous_Frame_Information");
Reception_Frame_Behaviour.registerState(new
Send_MembershipVector_And_SynchronizationInf_Received_Frame(),"Send_MembershipVector_And_SynchronizationInf_Received_Frame");
Reception_Frame_Behaviour.registerState(newSend_Received_TDMA_Table(),"Send_Received_TDMA_Table");
// Transitions
Reception_Frame_Behaviour.registerTransition("Wait_Reception_Frame","Wait_Reception_Frame",0);
Reception_Frame_Behaviour.registerTransition("Wait_Reception_Frame","Verification_CRC_Frame",1);
Reception_Frame_Behaviour.registerTransition("Verification_CRC_Frame","Identification_Frame_Type",0);
Reception_Frame_Behaviour.registerTransition("Verification_CRC_Frame","Send_Erroneous_Frame_Information",1);
Reception_Frame_Behaviour.registerDefaultTransition("Send_Erroneous_Frame_Information","Wait_Reception_Frame");
Reception_Frame_Behaviour.registerTransition("Identification_Frame_Type","Send_MembershipVector_And_SynchronizationInf_Received_Frame",1
);
Reception_Frame_Behaviour.registerTransition("Identification_Frame_Type","Send_Received_TDMA_Table",0);
Reception_Frame_Behaviour.registerTransition("Identification_Frame_Type","Wait_Reception_Frame",2);
Reception_Frame_Behaviour.registerDefaultTransition("Send_MembershipVector_And_SynchronizationInf_Received_Frame","Wait_Reception_Fram
e");
Reception_Frame_Behaviour.registerDefaultTransition("Send_Received_TDMA_Table","Wait_Reception_Frame");
```

## 2.5. Translation des diagrammes de plan du rôle «Membership»

### 2.5.1. Translation de diagramme de plan «l'Acquittement implicite»

```
FSMBehaviourAcknowledgement_Behaviour = newFSMBehaviour(this) {
};
//Definition of States
Acknowledgement_Behaviour.registerFirstState(new
Wait_Membership_Vector_of_First_Received_Frame(),"Wait_Membership_Vector_of_First_Received_Frame");
Acknowledgement_Behaviour.registerState(newTest_ChechIA(),"Test_ChechIA");
Acknowledgement_Behaviour.registerState(newTest_ChechIB(),"Test_ChechIB");
Acknowledgement_Behaviour.registerState(new
Wait_Membership_Vector_of_Second_Received_Frame(),"Wait_Membership_Vector_of_Second_Received_Frame");
Acknowledgement_Behaviour.registerState(newTest_ChechIIA(),"Test_ChechIIA");
Acknowledgement_Behaviour.registerState(newPositif_Acknowledgement(),"Positif_Acknowledgement");
Acknowledgement_Behaviour.registerState(newTest_ChechIIB(),"Test_ChechIIB");
Acknowledgement_Behaviour.registerState(newNegatif_Acknowledgement(),"Negatif_Acknowledgement");
//Definition of Transactions
Acknowledgement_Behaviour.registerTransition("Wait_Membership_Vector_of_First_Received_Frame","Wait_Membership_Vector_of_First_Received
_Frame",0);
Acknowledgement_Behaviour.registerTransition("Wait_Membership_Vector_of_First_Received_Frame","Test_ChechIA",1);
Acknowledgement_Behaviour.registerTransition("Test_ChechIA","Test_ChechIB",1);
Acknowledgement_Behaviour.registerTransition("Test_ChechIA","Positif_Acknowledgement",0);
Acknowledgement_Behaviour.registerDefaultTransition("Positif_Acknowledgement","Wait_Membership_Vector_of_First_Received_Frame");
Acknowledgement_Behaviour.registerTransition("Test_ChechIB","Wait_Membership_Vector_of_Second_Received_Frame",0);
Acknowledgement_Behaviour.registerTransition("Test_ChechIB","Wait_Membership_Vector_of_First_Received_Frame",1);
Acknowledgement_Behaviour.registerTransition("Wait_Membership_Vector_of_Second_Received_Frame","Wait_Membership_Vector_of_Second_Re
ceived_Frame",0);
Acknowledgement_Behaviour.registerTransition("Wait_Membership_Vector_of_Second_Received_Frame","Test_ChechIIA",1);
Acknowledgement_Behaviour.registerTransition("Wait_Membership_Vector_of_Second_Received_Frame","Wait_Membership_Vector_of_First_Recei
ved_Frame",2);
Acknowledgement_Behaviour.registerTransition("Test_ChechIIA","Test_ChechIIB",1);
Acknowledgement_Behaviour.registerTransition("Test_ChechIIA","Positif_Acknowledgement",0);
Acknowledgement_Behaviour.registerTransition("Test_ChechIIB","Wait_Membership_Vector_of_Second_Received_Frame",1);
Acknowledgement_Behaviour.registerTransition("Test_ChechIIB","Negatif_Acknowledgement",0);
Acknowledgement_Behaviour.registerDefaultTransition("Negatif_Acknowledgement","Wait_Membership_Vector_of_First_Received_Frame");
```

## 2.5.2. Translation de diagramme de plan «Mise à jour Vecteur Membership»

```
FSMBehaviourUpdate_Membership_Vector_Behaviour = newFSMBehaviour(this) {
};
//Definition of States
Update_Membership_Vector_Behaviour.registerFirstState(new
Wait_Received_Frame_Nodes_Vector(),"Wait_Received_Frame_Nodes_Vector");
Update_Membership_Vector_Behaviour.registerState(new
Update_Membership_Nodes_Vectors(),"Update_Membership_Nodes_Vectors");
//Definition of Transactions
Update_Membership_Vector_Behaviour.registerTransition("Wait_Received_Frame_Nodes_Vector","Wait_Received_Frame_No
des_Vector",0);
Update_Membership_Vector_Behaviour.registerTransition("Wait_Received_Frame_Nodes_Vector","Update_Membership_Node
s_Vectors",1);
Update_Membership_Vector_Behaviour.registerDefaultTransition("Update_Membership_Nodes_Vectors","Wait_Received_Fra
me_Nodes_Vector");
```

## 2.5.3. Translation de diagramme de plan «Mise à jour vecteur membership en cas de la réception d'une trame erronée»

```
FSMBehaviour Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour = newFSMBehaviour(this) {
};
//Definition of States
Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour.registerFirstState(new
Wait_Reception_Erroneous_Frame_Information(),"Wait_Reception_Erroneous_Frame_Information");
Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour.registerState(new
Update_Membership_Vector_Sender_Erroneous_Frame_Bit(),"Update_Membership_Vector_Sender_Erroneous_Frame_Bit");
//Definition of Transactions
Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour.registerTransition("Wait_Reception_Erroneous_Frame_I
nformation","Wait_Reception_Erroneous_Frame_Information",0);
Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour.registerTransition("Wait_Reception_Erroneous_Frame_I
nformation","Update_Membership_Vector_Sender_Erroneous_Frame_Bit",1);
Update_Membership_Vector_In_Case_of_Reception_Erroneous_Frame_Behaviour.registerDefaultTransition("Update_Membership_Vector_
Sender_Erroneous_Frame_Bit","Wait_Reception_Erroneous_Frame_Information");
```

## 2.5.4. Translation de diagramme de plan «Envoi de vecteur membership»

```
FSMBehaviourSend_Membership_Vector_Behaviour = newFSMBehaviour(this) {
};
//Definition of States
Send_Membership_Vector_Behaviour.registerFirstState(new
Wait_Received_Membership_Vector_Request(),"Wait_Received_Membership_Vector_Request");
Send_Membership_Vector_Behaviour.registerState(new
Send_Membership_Vector_To_Protocol_Processor(),"Send_Membership_Vector_To_Protocol_Processor");
//Definition of Transactions
Send_Membership_Vector_Behaviour.registerTransition("Wait_Received_Membership_Vector_Request","Wait_Received_Membership_Vector_
Request",0);
Send_Membership_Vector_Behaviour.registerTransition("Wait_Received_Membership_Vector_Request","Send_Membership_Vector_To_Protoc
ol_Processor",1);
Send_Membership_Vector_Behaviour.registerDefaultTransition("Send_Membership_Vector_To_Protocol_Processor","Wait_Received_Members
hip_Vector_Request");
```

## 2.6. Translation de diagramme de plan «Initialisation de cluster» du rôle «Startup»

```
Startup_Behaviour= newFSMBehaviour(this) {
                                        };
// States
Startup_Behaviour.registerFirstState(new Integration(),"Integration");
Startup_Behaviour.registerState(newClode_Start(),"Clode_Start");
Startup_Behaviour.registerState(new Synchronization(),"Synchronization");
// Transitions
Startup_Behaviour.registerTransition("Integration","Integration",0);
Startup_Behaviour.registerTransition("Integration","Synchronization",1);
Startup_Behaviour.registerDefaultTransition("Synchronization","Integration");
Startup_Behaviour.registerTransition("Integration","Clode_Start",2);
Startup_Behaviour.registerTransition("Clode_Start","Clode_Start",0);
Startup_Behaviour.registerTransition("Clode_Start","Integration",1);
```

## 2.7. Translation des diagrammes de plan du rôle «Synchronisation d'horloges et Gestion de la table TDMA»

### 2.7.1. Translation de diagramme de Plan «Réception d'une valeur d'horloge et une position de la table TDMA»

```
Reception_TDMA_Position_and_Clock_Value_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_TDMA_Position_and_Clock_Value_Behaviour.registerFirstState(new
Wait_Reception_of_TDMA_Position_and_Clock_Value(),"Wait_Reception_of_TDMA_Position_and_Clock_Value");
Reception_TDMA_Position_and_Clock_Value_Behaviour.registerState(new
Update_Current_TDMA_Position_and_Clock_Value(),"Update_Current_TDMA_Position_and_Clock_Value");
// Transitions
Reception_TDMA_Position_and_Clock_Value_Behaviour.registerTransition("Wait_Reception_of_TDMA_Position_and_Clock_Value","
Wait_Reception_of_TDMA_Position_and_Clock_Value",0);
Reception_TDMA_Position_and_Clock_Value_Behaviour.registerTransition("Wait_Reception_of_TDMA_Position_and_Clock_Value","
Update_Current_TDMA_Position_and_Clock_Value",1);
Reception_TDMA_Position_and_Clock_Value_Behaviour.registerDefaultTransition("Update_Current_TDMA_Position_and_Clock_Valu
e","Wait_Reception_of_TDMA_Position_and_Clock_Value");
```

### 2.7.2. Translation de diagramme de Plan «Réception de la table TDMA»

```
Reception_TDMA_Table_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_TDMA_Table_Behaviour.registerFirstState(newWait_Reception_of_TDMA_Table(),"Wait_Reception_of_
TDMA_Table");
Reception_TDMA_Table_Behaviour.registerState(newUpdate_TDMA_Table(),"Update_TDMA_Table");
// Transitions
Reception_TDMA_Table_Behaviour.registerTransition("Wait_Reception_of_TDMA_Table","Wait_Reception_of_TD
MA_Table",0);
Reception_TDMA_Table_Behaviour.registerTransition("Wait_Reception_of_TDMA_Table","Update_TDMA_Table",
1);
Reception_TDMA_Table_Behaviour.registerDefaultTransition("Update_TDMA_Table","Wait_Reception_of_TDMA_
Table");
```

### 2.7.3. Translation de diagramme de Plan «Envoi de la valeur courante d'horloge et la position courante de la table TDMA»

```
Send_TDMA_Position_and_Clock_Value_Behaviour = newFSMBehaviour(this) {
};
Send_TDMA_Position_and_Clock_Value_Behaviour.registerFirstState(new
Get_Current_Position_TDMA_Table(),"Get_Current_Position_TDMA_Table");
Send_TDMA_Position_and_Clock_Value_Behaviour.registerState(new
Send_TDMA_Current_Position_and_Clock_Current_Value(),"Send_TDMA_Current_Position_and_Clock_Current_Value");
// Transitions
Send_TDMA_Position_and_Clock_Value_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Get_Current_Position_T
DMA_Table",0);
Send_TDMA_Position_and_Clock_Value_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Send_TDMA_Current_P
osition_and_Clock_Current_Value",1)
Send_TDMA_Position_and_Clock_Value_Behaviour.registerDefaultTransition("Send_TDMA_Current_Position_and_Clock_Current_Val
ue","Get_Current_Position_TDMA_Table");
```

## 2.8. Translation des diagrammes de plan du rôle «ordonnancement déterministe»

### 2.8.1.Translation de diagramme de Plan «Réception information d'entrer d'un nouveau nœud»

```
Reception_Simulator_Information_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Simulator_Information_Behaviour.registerFirstState(new
Wait_Reception_Simulator_Information(),"Wait_Reception_Simulator_Information");Reception_Simulator_Information_Behaviour.reg
isterState(newAdd_New_Node_In_TDMA_Table(),"Add_New_Node_In_TDMA_Table");
                        // Transitions
Reception_Simulator_Information_Behaviour.registerTransition("Wait_Reception_Simulator_Information","Wait_Reception_Simulator
_Information",0);
Reception_Simulator_Information_Behaviour.registerTransition("Wait_Reception_Simulator_Information","Add_New_Node_In_TDM
A_Table",1);
Reception_Simulator_Information_Behaviour.registerDefaultTransition("Add_New_Node_In_TDMA_Table","Wait_Reception_Simula
tor_Information");
```

### 2.8.2. Translation de diagramme de plan «Réception d'une trame»

```
Reception_Frame_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Frame_Behaviour.registerFirstState(newWait_Reception_Frame(),"Wait_Reception_Frame");
Reception_Frame_Behaviour.registerState(new
Update_Clock_And_Position_TDMA_Table(),"Update_Clock_And_Position_TDMA_Table");
// Transitions
Reception_Frame_Behaviour.registerTransition("Wait_Reception_Frame","Wait_Reception_Frame",0);
Reception_Frame_Behaviour.registerTransition("Wait_Reception_Frame","Update_Clock_And_Position_TDMA_Table",1);
Reception_Frame_Behaviour.registerDefaultTransition("Update_Clock_And_Position_TDMA_Table","Wait_Reception_Frame");
```

## 2.8.3. Translation de diagramme de plan «Envoi de la table TDMA»

```
Send_TDMA_Table_Behaviour = newFSMBehaviour(this) {
};
Send_TDMA_Table_Behaviour.registerFirstState(new
Get_Current_Position_TDMA_Table(),"Get_Current_Position_TDMA_Table");
Send_TDMA_Table_Behaviour.registerState(newSend_TDMA_Table(),"Send_TDMA_Table");
// Transitions
Send_TDMA_Table_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Get_Current_Position_TDMA
_Table",0);
Send_TDMA_Table_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Send_TDMA_Table",1);
Send_TDMA_Table_Behaviour.registerDefaultTransition("Send_TDMA_Table","Get_Current_Position_TDMA_Table");
```

# Deuxième Partie du Chapitre V

# Etude de cas : Modélisation et translation du protocole FlexRay

# 1. Le Modèle Conceptuel du protocole FlexRay à base de la Méthodologie O-MaSE et le langage AUML

Selon la conception réelle d'un nœud FlexRay, nous avons supposé qu'un nœud FlexRay se compose de sept (7) agents suivants :

- Un agent «Controller Host Interface» : responsable de gérer la communication entre la partie hôte et la partie communication.
- Un agent «Protocol Operation Control» : responsable des services de wakeup, l'initialisation et l'émission d'une trame statique et la réception d'une trame…
- Un agent «Synchronizer and Communication Controller» : responsable de la gestion et la synchronisation de l'horloge du nœud et aussi réaliser toutes les tâches de contrôle liées au temps.
- Un agent «Generator» et « Arbitrator»: responsables d'appliquer la stratégie FTDMA d'accès au bus de communication pendant le segment dynamique.
- Un agent «Bus Driver» : gère la communication entre le nœud et le bus de communication
- Un agent «Bus Guardian» : pour la protection de bus contre les accès fautifs.

Nous avons proposé un agent externe aux nœuds qui est:

- Un agent «Communication Bus» : qui transmet les trames entre les nœuds du cluster.

Cette conception permet d'exécuter les différentes tâches du protocole FlexRay de façon parallèle et autonome. L'architecture proposée de notre SMA est présentée en Fig. V-2-1.

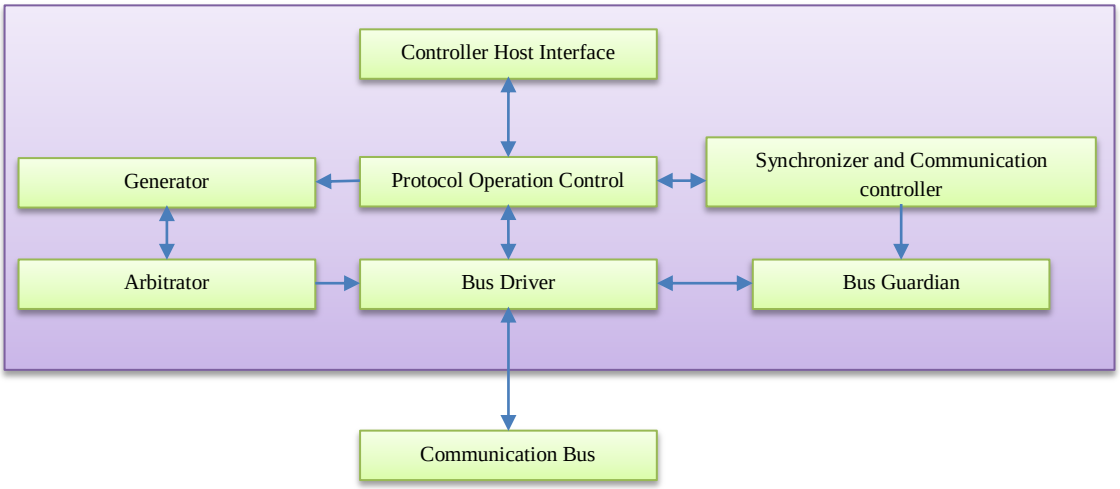


Fig. V-2-1 : Architecture générale de SMA proposée

## 1.1. Diagramme de buts

L'objectif global du protocole FlexRay est de garantir une communication de haute performance pour les applications distribuées temps réels mixtes (orientées Temps et événement) critiques de l'automobile. Cet objectif est noté (But0). But0 peut être divisé en quatre sous buts : La communication entre les nœuds (But1), une communication tolérante aux fautes (But2), une communication déterministe (But3), et une

communication flexible d'un débit élevé (But4). But2 peut être divisé à son tour en deux sous buts : une tolérance aux fautes des nœuds (But2.1) et une tolérance aux fautes de bus de communication (But2.2) (voir Fig. V-2-2).

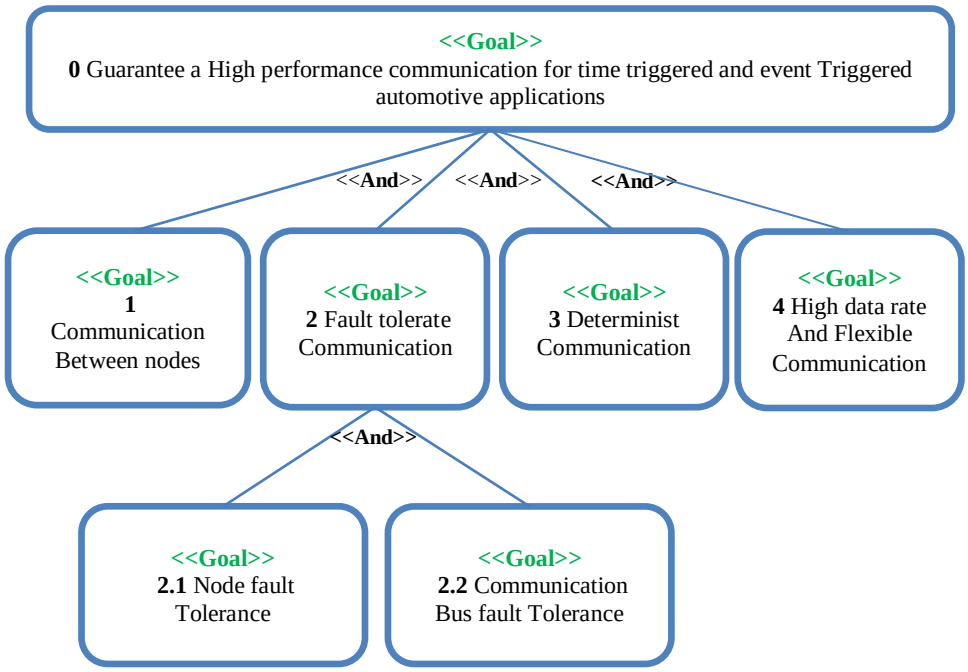


Fig. V-2-2 : Diagramme de buts du protocole FlexRay

## 1.2. Diagramme d'organisation

Dans notre approche proposée, nous avons identifié deux niveaux d'abstraction d'organisation de l'architecture FlexRay. Macro organisation qui représente le cluster ; l'acteur externe dans ce niveau est l'agent «Simulateur» et Micro organisation qui représente le nœud ; les acteurs externes dans ce niveau sont les agents «Bus de communication» et le «Simulateur». Fig. V-2-3 présente le diagramme d'organisation au niveau macro du cluster FlexRay et Fig.V-2-4 présente le diagramme d'organisation au niveau micro du nœud FlexRay.

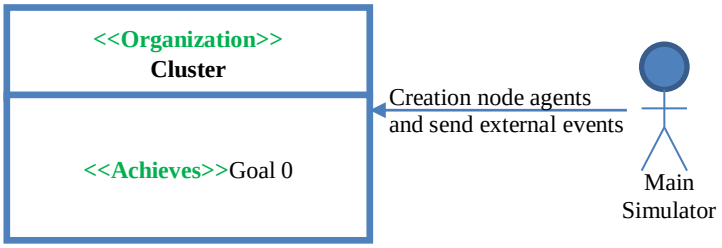


Fig. V-2-3 : Diagramme d'organisation du cluster FlexRay au niveau Macro

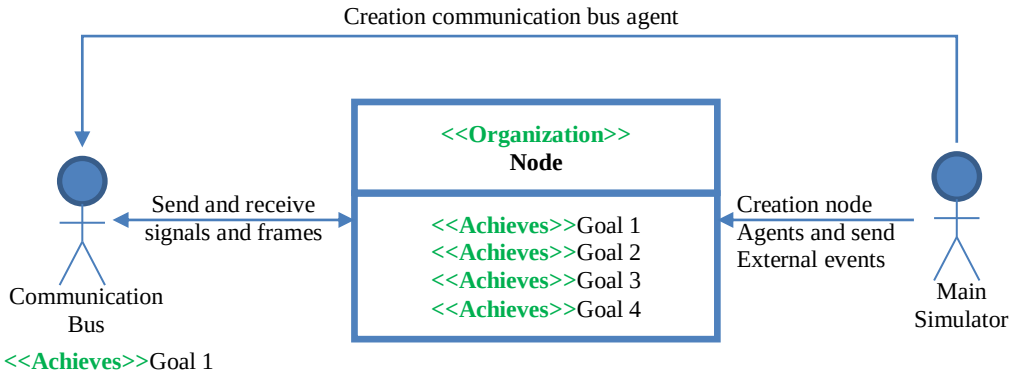


Fig. V-2-4 : Diagramme d'organisation du noeud FlexRay au niveau Micro

### 1.3. Diagramme de domaine

Comme il a été mentionné dans le deuxiéme chapitre (partie II), une Trame FlexRay est composée d'une séquence de segments ; elle est modélisée sous forme d'une classe et chacun de ses segments est aussi une sous classe où la relation entre la trame et un segment est une relation de composition. Le contenu des champs d'un segment peut étre un entier, string .., donc ces segments héritent des classes de base : Integer,String ..(voir Fig. V-2-5).

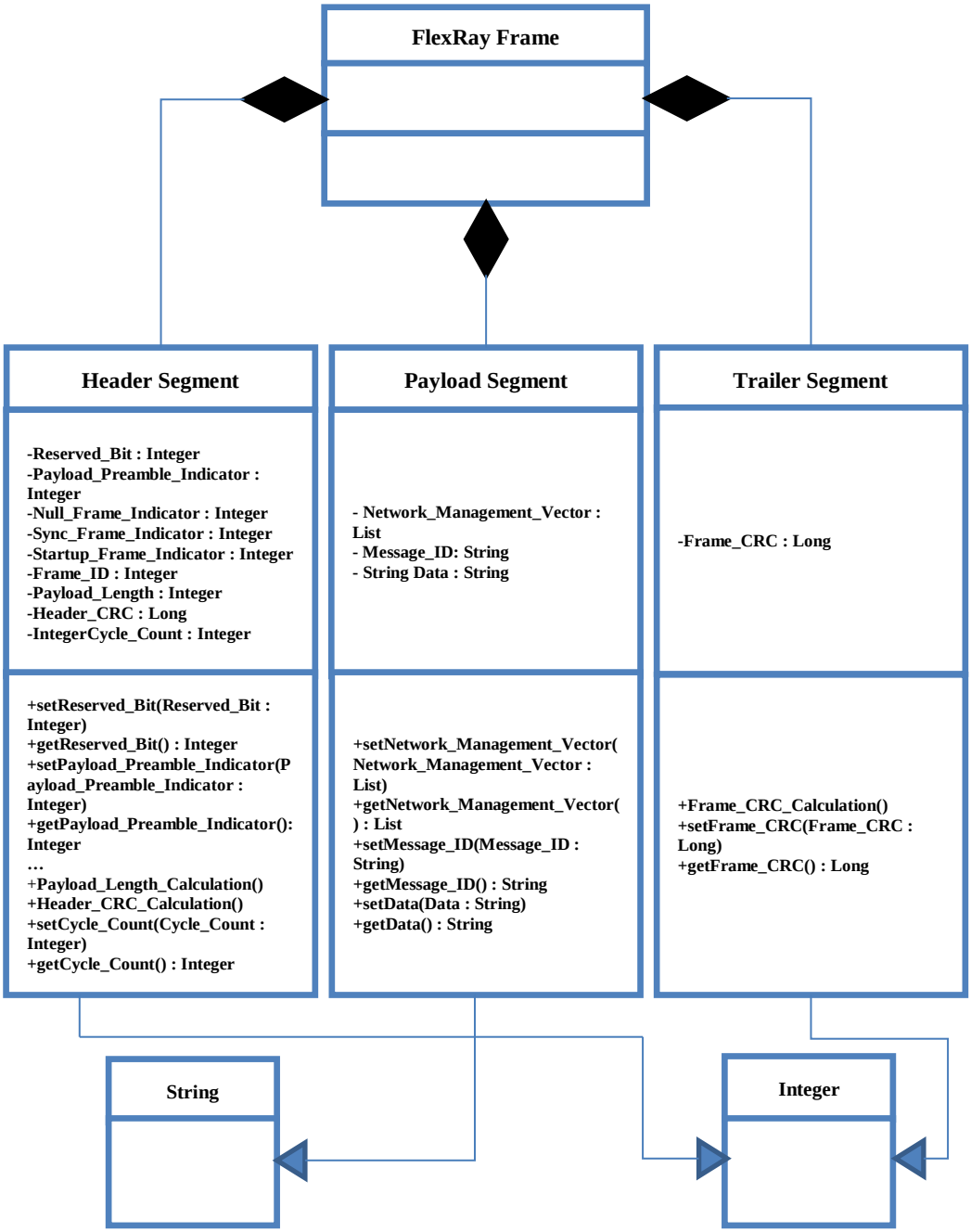


Fig.V-2-5 : Diagramme de domaine de la trame FlexRay

## 1.4. Diagramme de rôle

Notre objectif est d'établir le diagramme de rôle qui modélise les différents rôles pouvant être joués par les agents du système. Pour réaliser les Buts décrits précédemment dans le diagramme de buts, Nous avons identifié treize (13) rôles comme suit:

- Pour le But1 «La communication entre les nœuds» nous créons les rôles suivants : «Service de transmission», «L'émission d'une trame statique», «L'arbitrage et l'émission d'une trame dynamique», «La Réception d'une trame», «Wakeup», «Startup».

- Le rôle «Service de transmission» doit pouvoir transférer les différentes signaux et trames entre les nœuds d'un même cluster et les différentes informations de données et de contrôle entre les couches du même nœud, alors la capacité requise pour ce nœud est le plan «transmission d'information».
- Le rôle «émission d'une trame statique» qui s'occupe de la capacité «Réception des informations de synchronisation» qui permet de remplir les différents champs de la trame statique à émettre par les différentes informations reçues et l'envoi de cette trame pendant le segment statique.
- Le rôle «L'arbitrage et l'émission d'une trame dynamique». Ce rôle s'occupe d'une part de la sélection du message le plus prioritaire dans le cas d'occurrence de plusieurs messages évènements à émettre dans le segment dynamique (deuxième chapitre, Partie II, Section IV.1) et d'autre part de remplir les champs et l'envoi de la trame dynamique de ce message sélectionné sur le bus de communication. Il requiert les capacités : «Réception d'un message évènement» qui permet lors de la réception d'un message évènement d'envoyer l'identificateur de ce message aux autres agents pour réaliser l'opération de l'arbitrage. «Réception de l'identification d'un message» qui permet lors de la réception des identificateurs des messages évènement de choisir à un certain moment le message le plus prioritaire et la construction et l'envoi d'une trame dynamique sur le bus de communication.
- Le rôle «La Réception d'une trame» permet d'effectuer toutes les opérations nécessaires lors de la réception d'une trame. Ce rôle dépend de la capacité «l'envoi des données de la trame reçue» qui permet d'envoyer à chaque rôle les champs de la trame reçue qu'ils lui sont intéressants.
- Le rôle «Wakeup» requiert la capacité «Réveiller les nœuds du cluster» qui permet d'exécuter l'algorithme de wakeup du protocole FlexRay décrit dans le deuxième chapitre (Partie II, Section IV.2).
- Pour le But2.1 «La tolérance aux fautes des nœuds» nous créons les rôles : «Startup», «Synchronisation d'horloges et control du déroulement des segments», «L'arbitrage et l'émission d'une trame dynamique». «Contrôler l'état du contrôleur de communication», «La protection de bus contre des accès fautifs».
- Le rôle «Startup» requiert la capacité «Initialisation de cluster» qui permet d'exécuter l'algorithme d'initialisation de cluster du protocole FlexRay décrit dans le deuxième chapitre (Partie II, Section IV.3).
- Le rôle «Synchronisation d'horloges et control du déroulement des segments» permet d'une part d'exécuter l'algorithme de synchronisation d'horloge du protocole FlexRay décrit dans le deuxième chapitre (Partie II, Section IV.4) pour garantir une vue commune de l'horloge entre tous les nœuds du cluster, et d'autre part pour garantir le parcours d'un segment à un autre pendant le cycle de communication. Ce rôle nécessite les capacités suivantes : «Calcul des valeurs de correction d'offset et de rate» qui permet d'exécuter l'algorithme de calcul des valeurs de correction d'offset et de rate. «Changement de Segment et l'application de la correction de rate et l'offset» qui

permet d'une part après l'expiration de la période d'un segment de passer au segment suivant (Les segments du cycle de communication FlexRay sont présentés dans le deuxième chapitre, Partie II, Section IV.1), et d'autre part d'appliquer la correction de l'offset et de rate. «Validation du slot de la trame coldstart reçue» qui permet de valider les trames coldstart reçues dans les phases de startup. «Envoi de l'information Début de Segment Dynamique» qui permet d'indiquer aux autres rôles le début de segment dynamique pour pouvoir lancer la stratégie FTDMA d'accès au bus de communication. «Envoi de l'information Fin de Segment Dynamique» qui permet d'indiquer aux autres rôles la fin de segment dynamique pour pouvoir arrêter la stratégie FTDMA d'accès au bus de communication.

- Le Rôle «Contrôler l'état du contrôleur de communication» requiert la capacité «Changement des Etats du contrôleur de communication» qui permet au contrôleur de communication de passer d'un état à un autre comme il est décrit dans le deuxième chapitre (Partie II, Section III).
- Le Rôle «Protection de bus contre des accès fautifs» permet d'empêcher l'accès d'un nœud dans le segment statique au bus de communication dans un temps qui n'est lui pas réservé.
- Le Rôle «Gestion de la table TDMA» permet de réaliser But3 « une communication déterministe». Ce rôle permet d'appliquer durant le segment statique la stratégie TDMA d'accès au bus de communication, il requiert la capacité «Envoi des valeurs courantes des compteurs de cycle et de Slot» qui réalise le parcours instantané de la table TDMA et l'envoi de la valeur courante des compteurs de cycle et de slot en cas où le slot courant est réservé pour émettre une trame statique.
- Pour le But4 «une communication flexible et d'un débit élevé» nous créons les rôles «Contrôler la stratégie FTDMA d'accès au bus de communication», «Génération des Messages Evènements» et «L'arbitrage et l'émission d'une trame dynamique».
- Le Rôle «Contrôler la stratégie FTDMA d'accès au bus de communication» permet de lancer et arrêter la stratégie FTDMA d'accès au bus de communication. Ce rôle requiert les capacités «Réception l'information Début de Segment Dynamique» qui permet lors de la réception de l'information de début de segment dynamique et si le nœud est dans l'état Active de lancer la stratégie FTDMA, «Réception l'information Fin de Segment Dynamique» qui permet lors de la réception de l'information de la fin de segment dynamique d'arrêter la stratégie FTDMA.
- Le Rôle «Génération des Messages Evènements» permet de générer un message de type évènement dans le segment dynamique lorsqu'il est autorisé. Il requiert les capacités suivantes : «Réception de l'autorisation de lancer la stratégie FTDMA d'accès au bus de communication» qui permet lors de la réception d'une commande de lancer la stratégie FTDMA pendant le segment dynamique d'accès au bus de communication. «Réception de la commande d'arrêter la stratégie FTDMA d'accès au bus de communication» qui permet lors de la réception d'une commande d'arrêt lorsque le segment dynamique est terminé de stopper la stratégie FTDMA. «Réception de

l'autorisation de génération auprès de rôle arbitrage» qui permet lors d'une réception d'un signal d'autorisation auprès du rôle arbitrage de continuer la génération de message événement suivant.

- Le rôle «mécanisme de redondance» permet de réaliser le But2.1 et le But 2.2 «La tolérance aux fautes de bus de communication». Ce rôle permet d'appliquer le principe de redondance expliqué dans le deuxième chapitre.

Fig. V-2-6 présente le diagramme de rôle du protocole FlexRay.

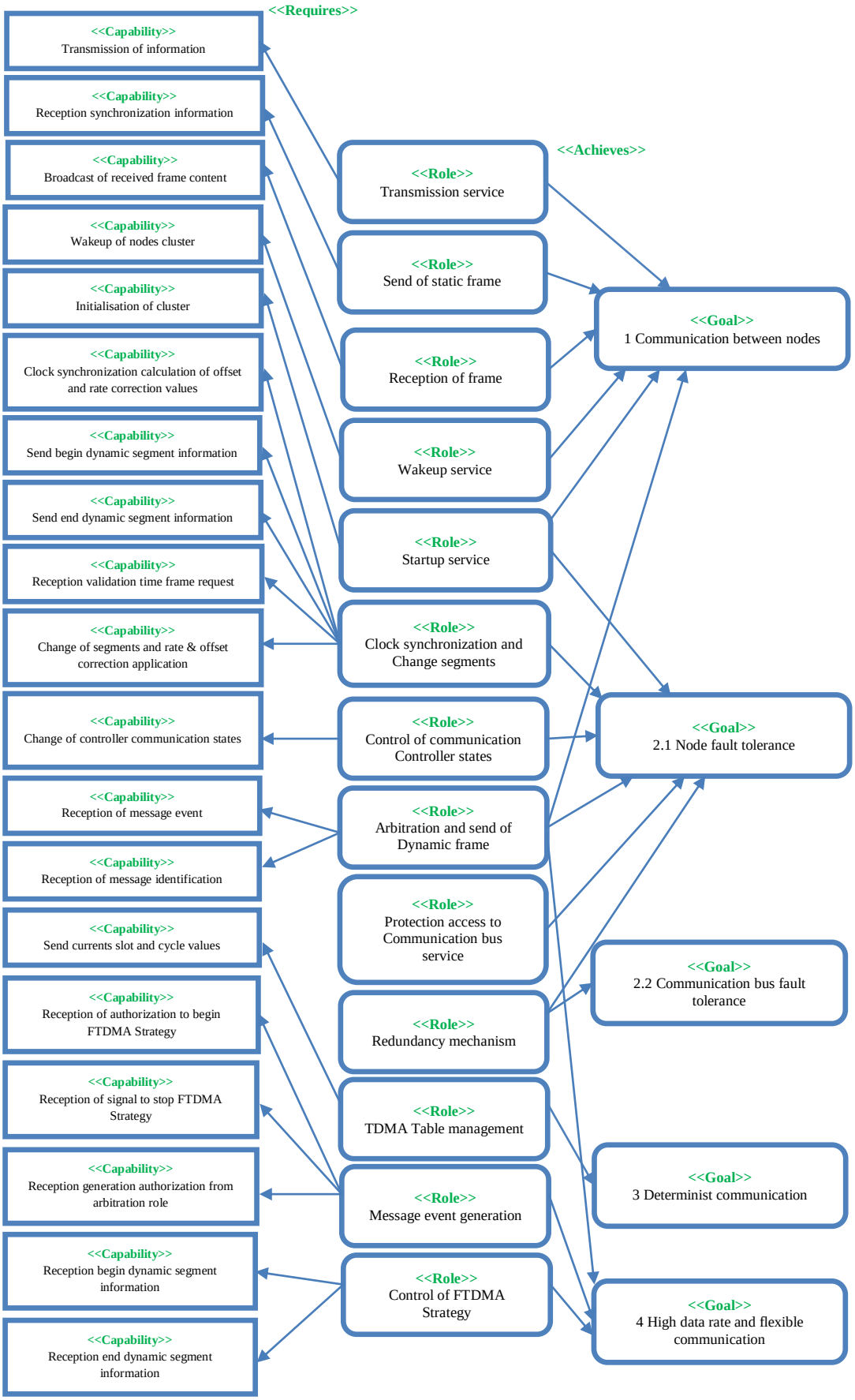


Fig. V-2-6 : Diagramme de rôle du protocole FlexRay

### 1.5. Diagramme de classes d'agent

Nous avons identifié huit (8) classes d'agent qui fonctionnent simultanément. Un agent nommé «Controller Host Interface» jouant le rôle «Service de transmission», un agent nommé « Protocol Operation Control» jouant les rôles : «Wakeup>, «Startup», «L'émission d'une trame statique», «Réception d'une trame», «Contrôler l'état du contrôleur de communication» et «Contrôler la stratégie FTDMA d'accès au bus de communication». Un agent nommé «Synchronisateur et Contrôleur de communication» jouant les rôles : «Synchronisation d'horloges et control du déroulement des segments» et «Gestion de la table TDMA». Un agent nommé «Bus gardian» jouant le rôle «La protection de bus contre des accès fautifs». Un agent nommé «Bus Driver» jouant le rôle «Service de Transmission». Un Agent «Générateur» jouant le rôle «Génération des Messages Evènements». Un Agent nommé «Arbitraire» jouant le rôle «L'arbitrage et l'émission d'une trame dynamique». Enfin l'agent : «Bus de communication» jouant le rôle «Service de Transmission». Le rôle «Mécanisme de redondance» est joué par l'instanciation de deux instances de la classe d'agent «bus de communication» et des instances de chaque classe d'agent définie (Fig.V-2-7 présente le diagramme d'agents de protocole FlexRay).

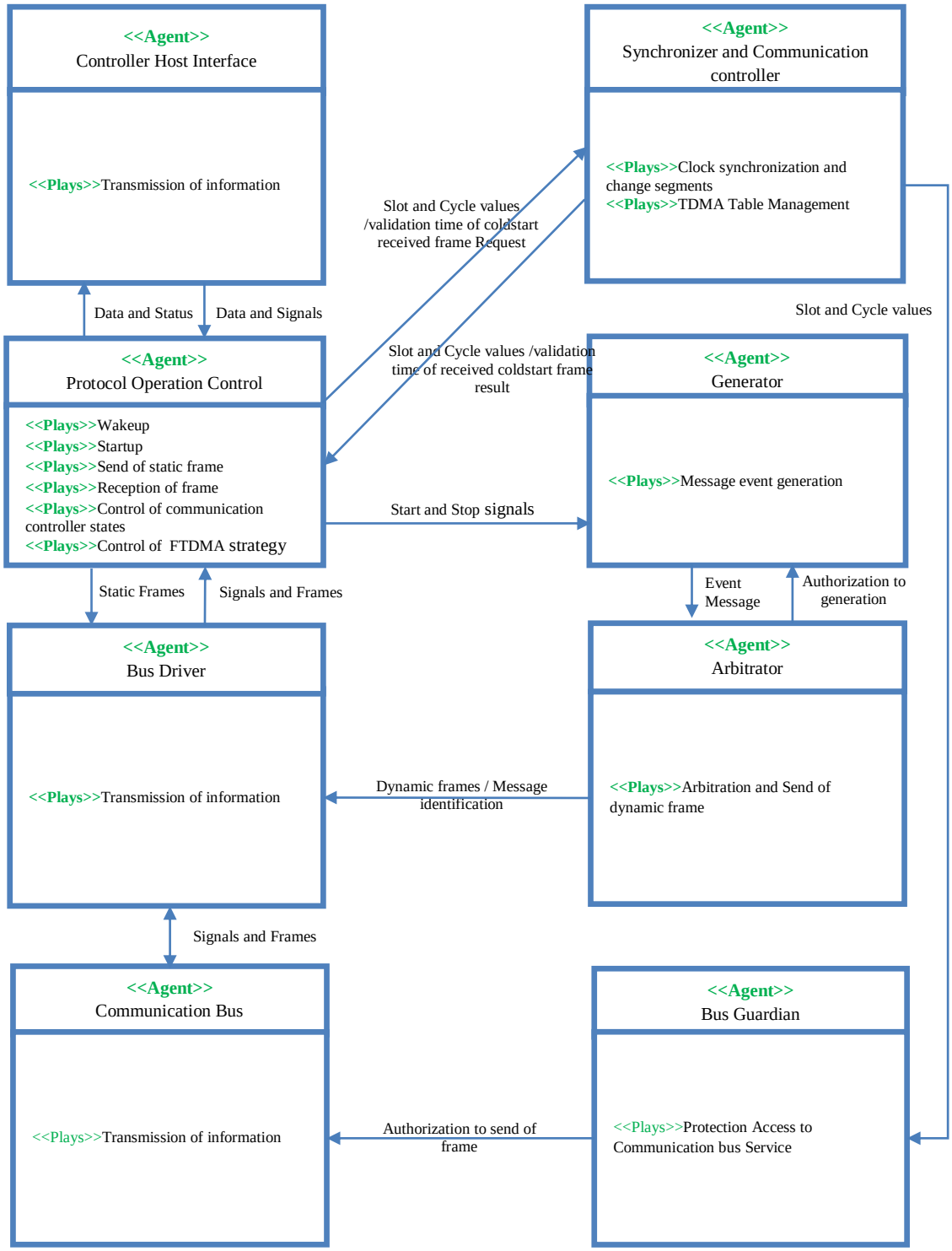


Fig. V-2-7 : Diagramme de classes d'agent du protocole FlexRay

## 1.6. Diagrammes de plan

### 1.6.1. Plan «transmission d'informations» du rôle «Service de transmission»

Suite à la réception d'une information qui peut être une trame statique (normale ou d'initialisation), une trame dynamique, un wakeup pattern, un signal CAS (Collision advances symbol), des informations de contrôle, des informations de configuration ou

bien des données…, le rôle «Service de transmission» envoie cette information reçue aux autres rôles et il retourne à l'état d'attente (voir Fig. V-2-8).

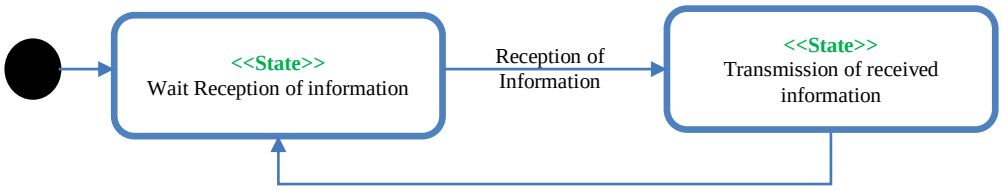


Fig. V-2-8 : Diagramme de plan «transmission d'informations»

### 1.6.2. Diagrammes de plan «Réception des informations de synchronisation» du rôle «L'émission d'une trame statique»

Suite à la réception des informations de synchronisation qui représentent la valeur courante des compteurs de slot et de cycle, l'agent jouant ce rôle constate que le slot courant correspond au moment pour envoyer sa trame statique. En effet il remplit les champs ID et Cycle de la trame à émettre par ceux reçus et il envoie une demande de données, lorsque qu'il reçoit des données, il remplit le champ de données de la trame à émettre par les donnes reçues et il envoie la trame après qu'il complète de remplir les autres champs de la trame et il retourne à l'état d'attente (voir Fig. V-2-9).

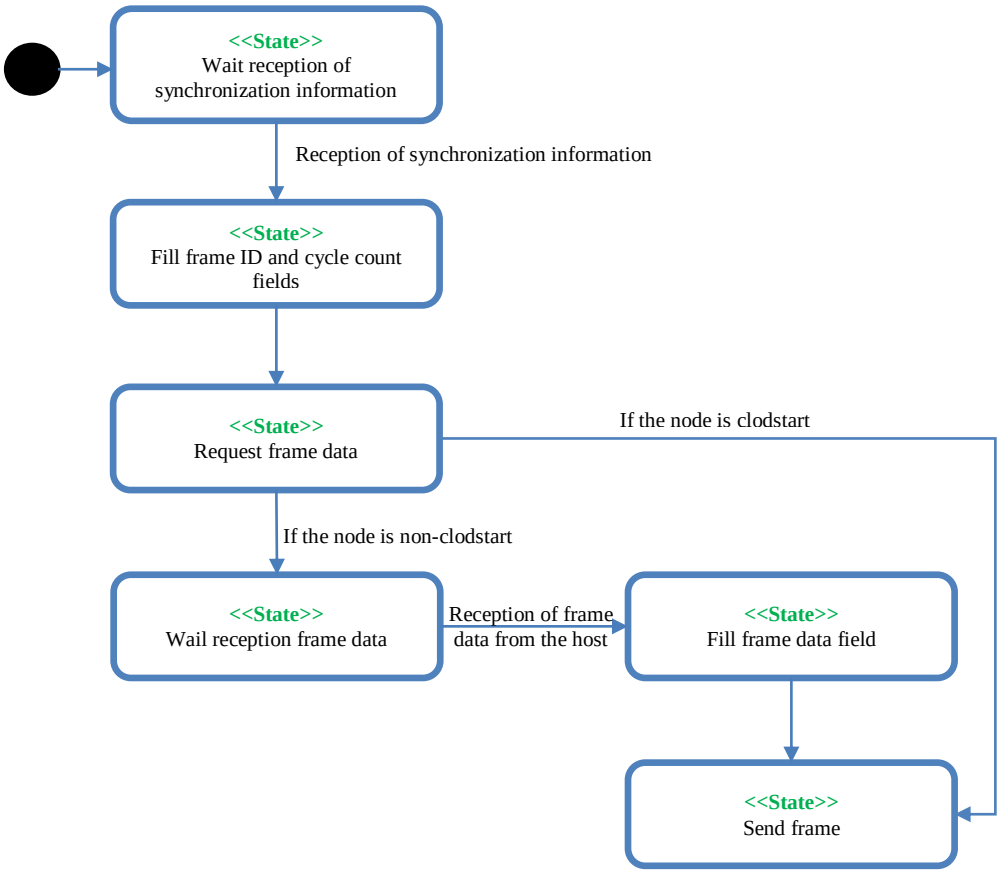


Fig. V-2-9 : Diagramme de plan «Réception des informations de synchronisation»

### 1.6.3. Diagrammes de plan pour les capacités du Rôle «l'arbitration et l'émission d'une trame dynamique»

#### 1.6.3.1. Le Plan «Réception d'un message évènement»

Suite à la réception d'un message évènement auprès de l'agent qui joue le rôle «Génération des Messages Evènements», l'agent qui joue ce rôle envoie l'identification (priorité) de messages à tous les autres nœuds du cluster et retourne à l'état d'attente (voir Fig. V-2-10).

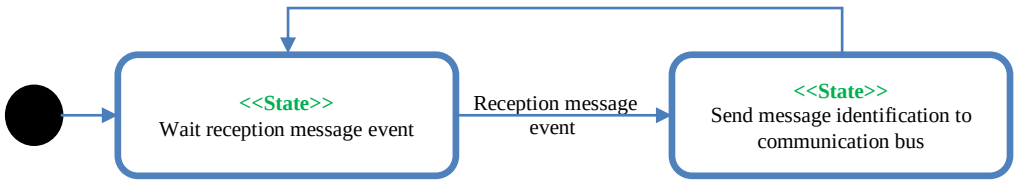


Fig. V-2-10: Diagramme de plan «Réception d'un message évènement»

#### 1.6.3.2. Le Plan «Réception de l'identification d'un message»

Suite à la réception d'une identification d'un message évènement, l'agent qui joue ce rôle doit ajouter cette identification à la table des identifications des messages, si le temps d'écoute est non plus écoulé il retourne à l'état d'attente, sinon il passe à l'état de sélection du message le plus prioritaire, si le message local de cet agent est le plus prioritaire alors il construit une trame dynamique et il envoie cette trame sur le bus de communication et il retourne à l'état d'attente, sinon il retourne à l'état d'attente sans aucune émission (voir Fig. V-2-11).

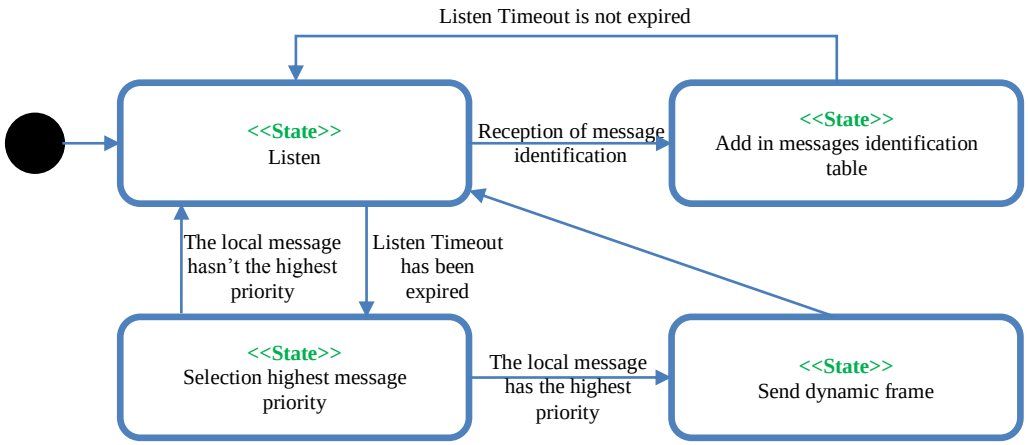


Fig. V-2-11: Diagramme de plan «Réception de l'identification d'un message»

### 1.6.4. Plan «Réveiller les nœuds du cluster» du rôle «Wakeup»

Ce diagramme modélise l'algorithme de Wakeup du protocole FlexRay décrit dans le deuxième chapitre (Partie II, Section IV.2) (voir Fig. V-2-12).

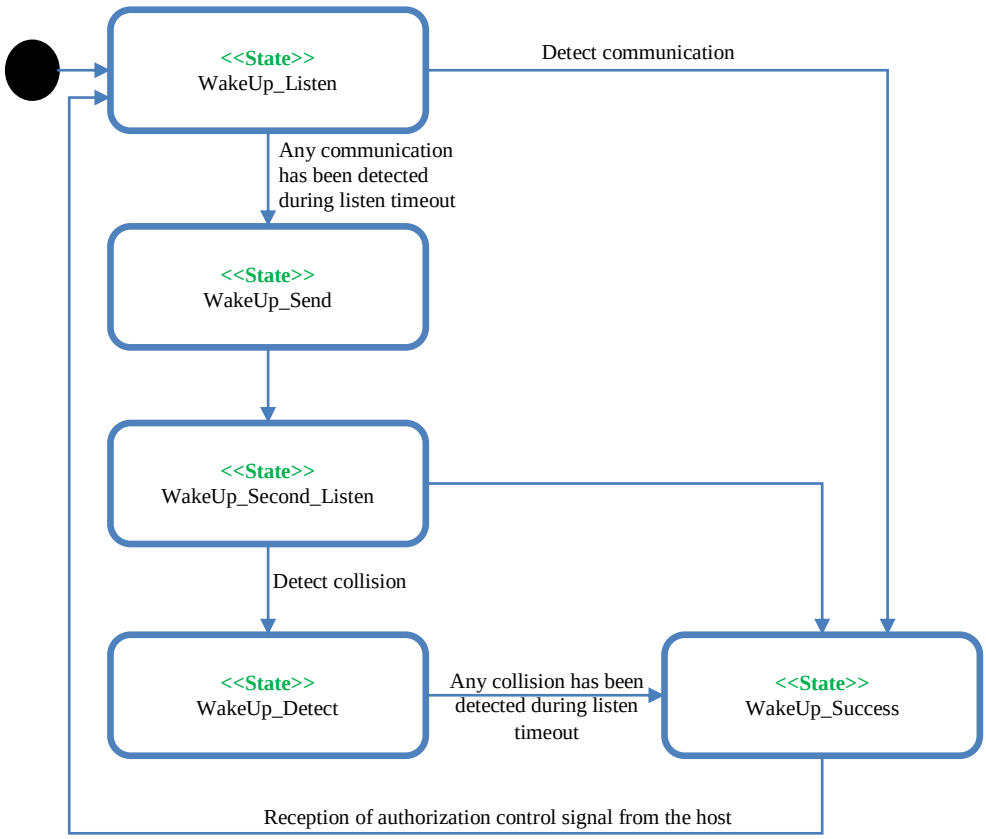


Fig. V-2-12 : Diagramme du plan «Réveiller les nœuds du cluster»

### 1.6.5. Plan «Initialisation du cluster» du rôle «Startup»

Ce diagramme de plan modélise l'algorithme de startup du protocole FlexRay décrit dans le deuxième chapitre (Partie II, Section IV.3) (voir Fig. V-2-13).

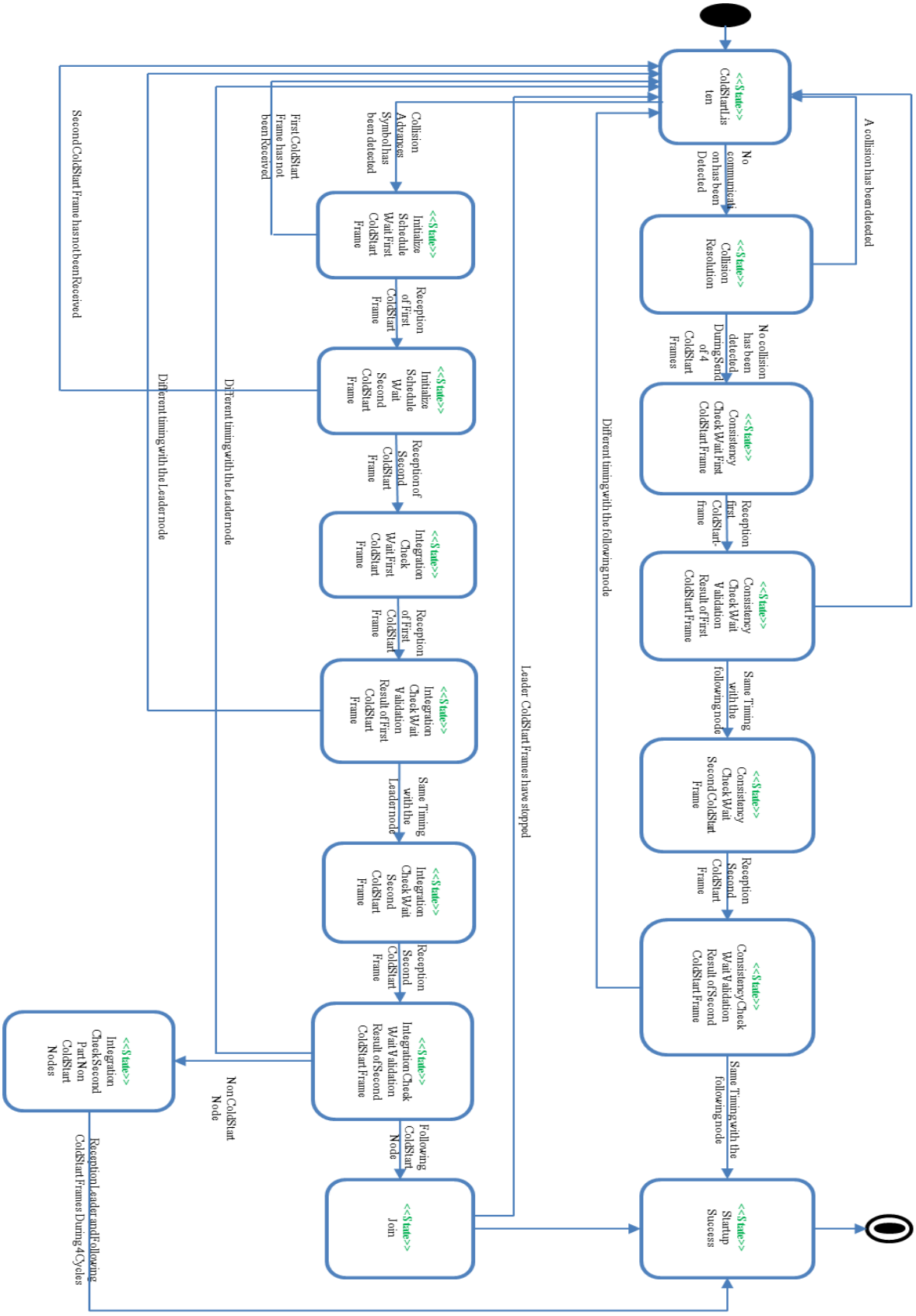


Fig. V-2-13: Diagramme du plan «Initialisation du cluster»

### 1.6.6. Diagrammes de plan pour les capacités du rôle «Synchronisation d'horloges et control du déroulement des segments»

#### 1.6.6.1. Plan «Calcul des valeurs de correction d'offset et de rate»

Suite à la réception d'une trame de synchronisation, si le segment courant est le segment statique, l'agent jouant ce rôle mesure la différence entre le temps d'arrivée prévu de la trame et le temps d'arrivée réel. Si le segment statique est terminé, l'agent jouant ce rôle exécute l'algorithme de FTM pour calculer la valeur de correction de l'offset. En cas où le compteur de cycle prend une valeur paire, l'agent applique le même algorithme pour calculer la valeur de correction de rate. Lorsqu'il termine les calculs il retourne à l'état d'attente d'une trame (voir Fig. V-2-14).

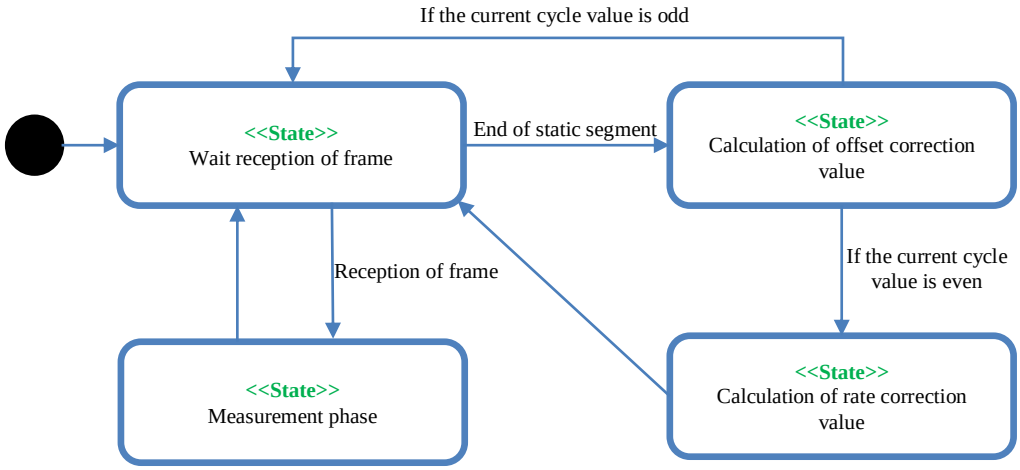


Fig. V-2-14: Diagramme du plan «Calcul des valeurs de correction d'offset et de rate»

#### 1.6.6.2. Plan «Changement de segment et l'application de la correction de rate et l'offset»

Si l'agent qui joue ce rôle est dans l'état statique il attend l'expiration de temps de ce segment pour passer à l'état dynamique et de la même façon il passe aux segments symbol et NIT. L'agent applique la correction d'offset pendant le segment NIT et il applique la correction de rate durant les quatre segments (voir Fig. V-2-15).

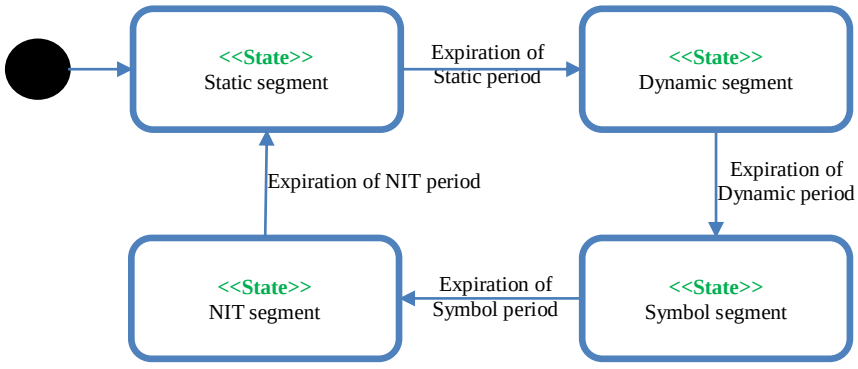


Fig. V-2-15 : Diagramme du plan «Changement de segment et l'application de la correction de rate et l'offset»

#### 1.6.6.3. Plan «Validation du slot de la trame coldstart reçue»

Suite à la réception d'une trame coldstart pour valider son temps auprès de l'agent qui joue le rôle «startup», l'agent joue ce rôle doit vérifier si le slot et le cycle de la trame reçue sont les mêmes que le slot et le cycle courants pour constater que le temps du nœud émetteur de la trame reçue est le même que le temps de ce nœud. Enfin cet agent envoie un résultat de cette vérification à l'agent qui joue le rôle «startup», et il retourne à l'état d'attente (voir Fig. V-2-16).

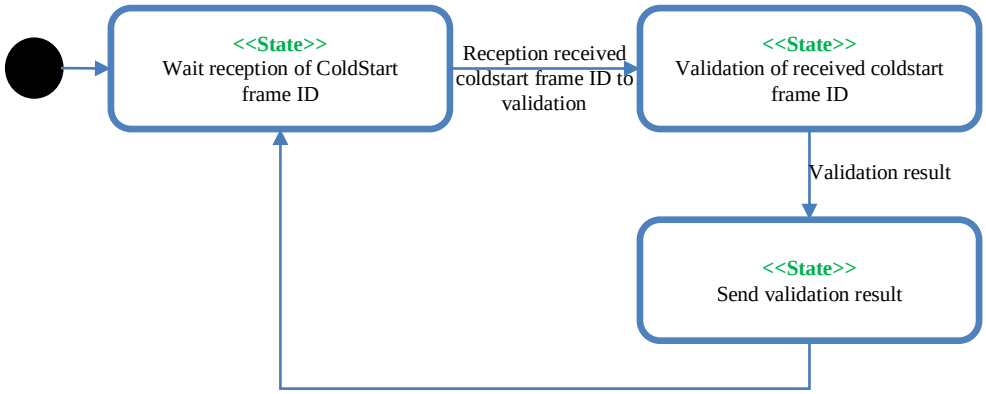


Fig. V-2-16 : Diagramme de plan «Validation du slot de la trame coldstart reçue»

#### 1.6.6.4. Plan «Envoi de l'information début de segment dynamique»

Lorsque l'agent qui joue ce rôle détecte que le segment courant est le segment dynamique, il envoie l'information de début de segment dynamique à l'agent qui joue le rôle «Contrôler la stratégie FTDMA d'accès au bus de communication», et il retourne à l'état d'attente (voir Fig. V-2-17).

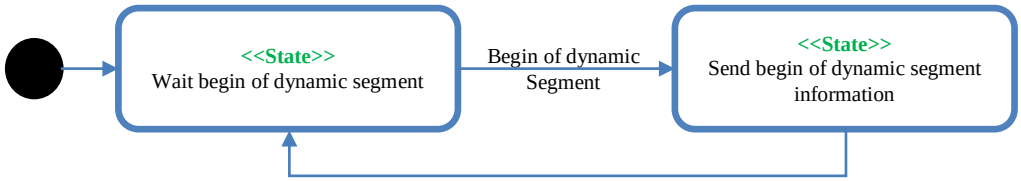


Fig. V-2-17 : Diagramme du plan «Envoi de l'information début de segment dynamique»

#### 1.6.6.5. Plan «Envoi de l'information fin de segment dynamique»

Lorsque l'agent qui joue ce rôle détecte la fin du segment dynamique, il envoie l'information de fin de segment dynamique à l'agent qui joue le rôle «Contrôler la stratégie FTDMA d'accès au bus de communication» et retourne à l'état d'attente (voir Fig. V-2-18).

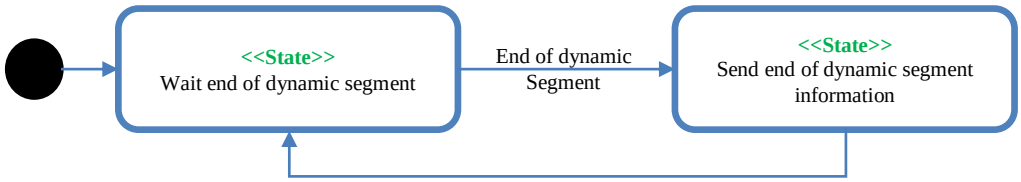


Fig. V-2-18 : Diagramme du plan «Envoi de l'information fin de segment dynamique»

### 1.6.7. Plan «Changement des états du contrôleur de communication» du rôle «Contrôler l'état du contrôleur de communication»

Ce Plan présente le passage du contrôleur de communication d'un état à un autre comme il a été mentionné dans le deuxième chapitre (Partie 2, Section III) (voir Fig. V-2-19).

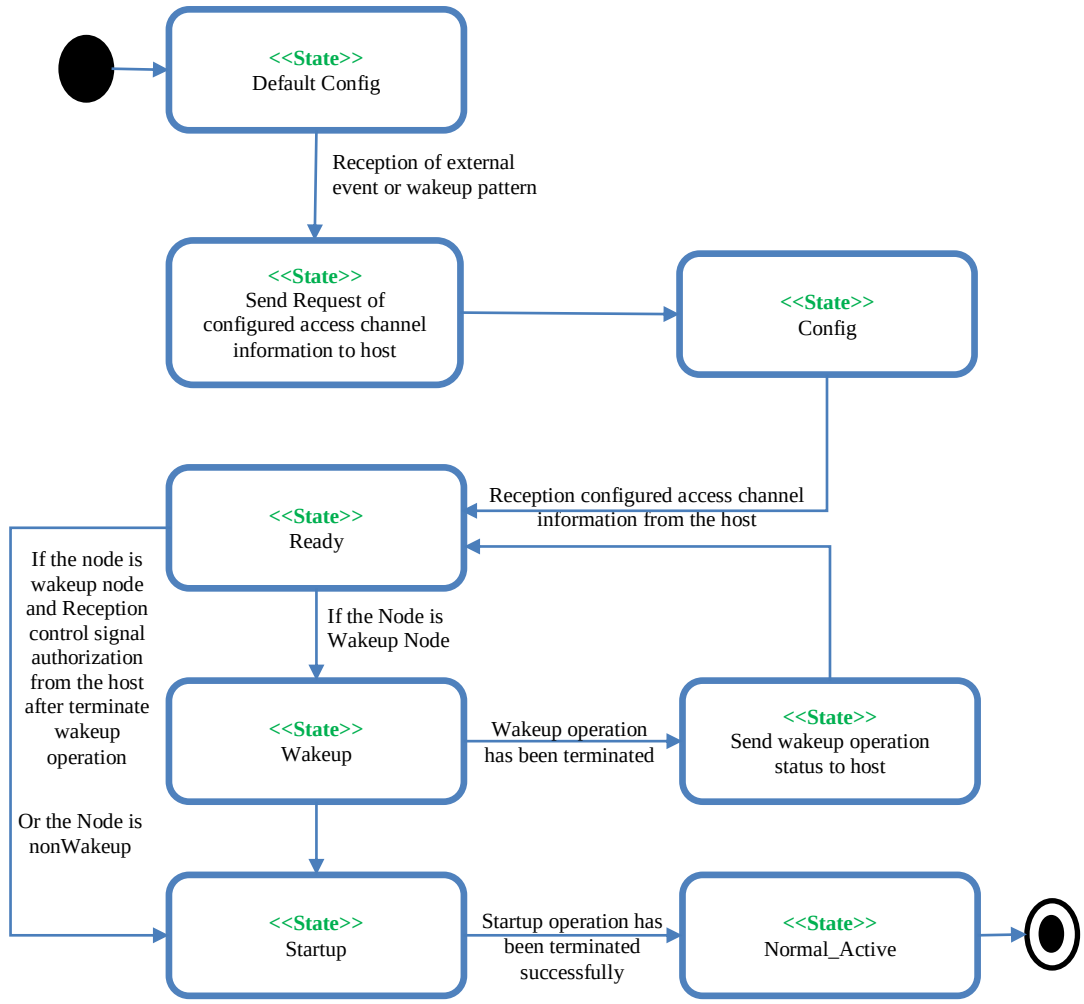


Fig. V-2-19: Diagramme du plan «Changement des états du contrôleur de communication»

### 1.6.8. Pan «Envoi des valeurs courantes des compteurs de cycle et de slot» du rôle «Gestion de la table TDMA»

L'agent jouant ce rôle fait parcourir à tout moment la table TDMA en fonction de la valeur courante de macrotick, il cherche si la valeur courante de la position de la table TDMA est correspondante au slot de l'émission d'une trame, il envoie les valeurs courantes de compteurs de cycle et de slot à l'agent qui joue le rôle «l'émission d'une trame statique» (Fig. V-2-20).

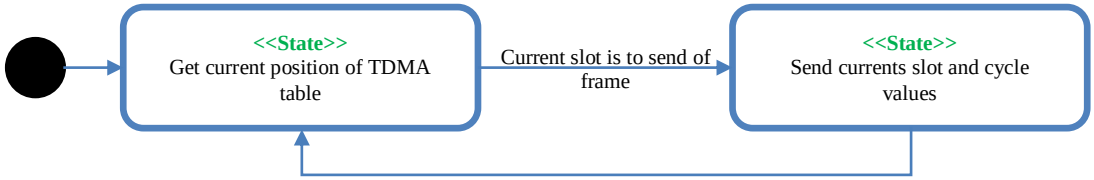


Fig. V-2-20 : Diagramme du plan «Envoi des valeurs courantes des compteurs de cycle et de slot»

### 1.6.9. Diagrammes de plan pour les capacités du rôle «Contrôler la stratégie FTDMA d'accès au bus de communication»

#### 1.6.9.1. Pan «Réception l'information début de segment dynamique»

Suite à la réception de l'information du début de segment dynamique, l'agent qui joue ce rôle doit vérifier son état, si cet état est actif normal, il envoie cette information à l'agent qui joue le rôle «Génération des massages évènement» pour lui autoriser d'appliquer la stratégie FTDMA d'accès au bus de communication et il retourne à l'état d'attente (voir Fig. V-2-21 ).

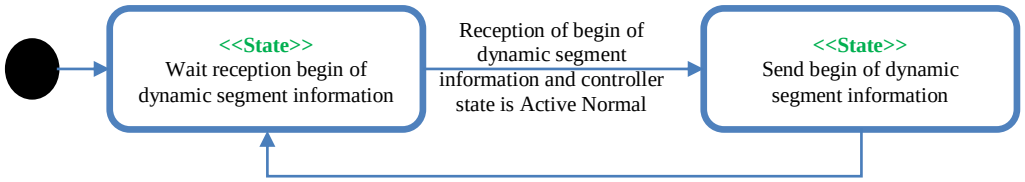


Fig. V-2-21 : Diagramme de plan «Réception l'information début de segment dynamique»

#### 1.6.9.2. Pan «Réception l'information fin de segment dynamique»

Suite à la réception de l'information de fin de segment dynamique, l'agent jouant ce rôle doit vérifier son état, si cet état est actif normal, il envoie cette information à l'agent qui joue le rôle «Génération des massages évènement» pour lui empêcher d'appliquer la stratégie FTDMA d'accès au bus de communication et retourne à l'état d'attente (voir Fig. V-2-22).

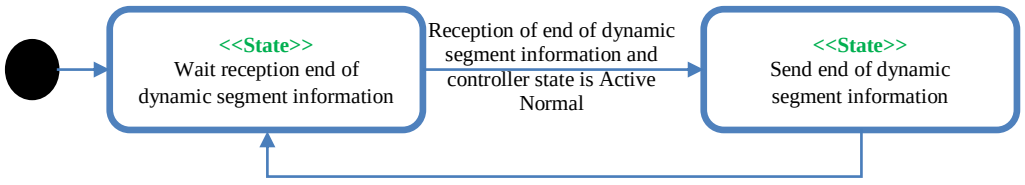


Fig. V-2-22 : Diagramme de plan «Réception l'information fin de segment dynamique»

### 1.6.10. Diagrammes de plan pour les capacités du rôle «Génération des messages événement»

#### 1.6.10.1. Plan «Réception de l'autorisation de lancer la stratégie FTDMA d'accès au bus de communication»

Suite à la réception d'un signal d'autorisation pour appliquer la stratégie FTDMA auprès de l'agent qui joue le rôle «Contrôler la stratégie FTDMA d'accès au bus de communication», l'agent qui joue ce rôle génère d'une manière aléatoire un message avec une priorité (identification), s'il génère un message il envoie ce message au rôle «l'arbitration et l'émission d'une trame dynamique», sinon il retourne à l'état d'attente (voir Fig. V-2-23).

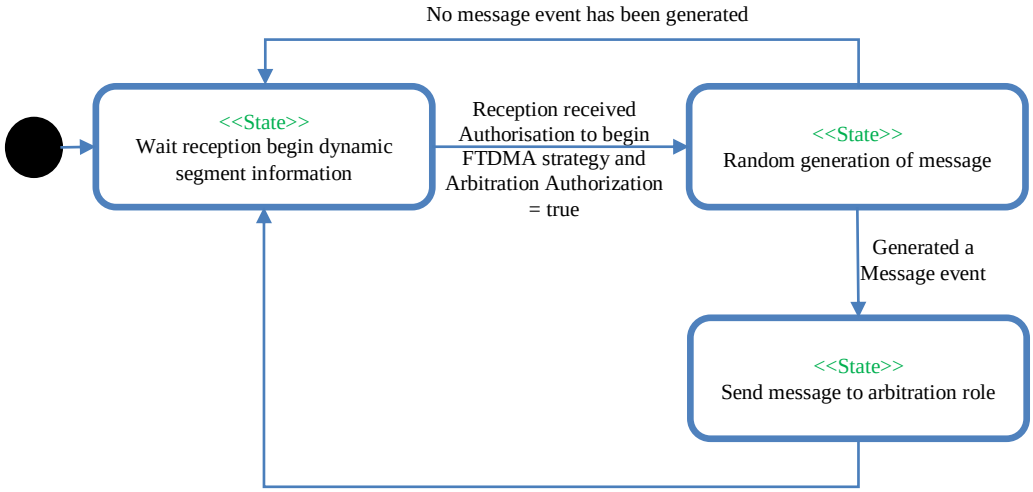


Fig. V-2-23 : Diagramme de plan «Réception de l'autorisation de lancer la stratégie FTDMA d'accès au bus de communication»

#### 1.6.10.2. Plan «Réception de la commande d'arrêter la stratégie FTDMA d'accès au bus de communication»

Suite à la réception d'un signal d'arrêt auprès de l'agent qui joue le rôle «Contrôler la stratégie FTDMA d'accès au bus de communication» qui indique la fin de segment dynamique, l'agent qui joue ce rôle arrête la génération des messages événement et retourne à l'état d'attente (voir Fig. V-2-24).

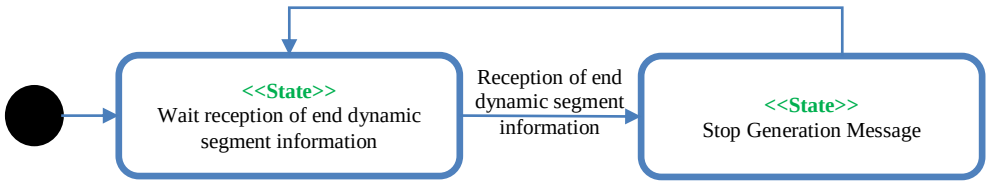


Fig. V-2-24 : Diagramme de plan «Réception de la commande d'arrêter la stratégie FTDMA d'accès au bus de communication»

#### 1.6.10.3. Plan «Réception de l'autorisation de génération auprès de rôle arbitrage»

Suite à la réception d'une autorisation de génération auprès de l'agent qui joue le rôle «l'arbitration et l'émission d'une trame dynamique», ce rôle met à jour la variable d'autorisation de génération à « vraie » et il retourne à l'état d'attente (voir Fig. V-2-25).

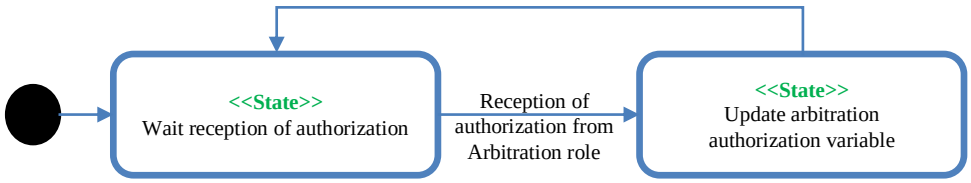


Fig. V-2-25 : Diagramme de plan «Réception de l'autorisation de génération auprès de rôle arbitrage»

### 1.7. Diagrammes de protocoles d'AUML

Nous proposons trois diagrammes de protocole : «Diagramme de protocole d'émission d'une trame statique», «Diagramme de protocole d'émission d'une trame dynamique» et «Diagramme de protocole de réception». Nous présentons dans ce chapitre les deux premiers diagrammes de protocoles.

#### 1.7.1. Diagramme de protocole d'émission d'une trame statique

Ce protocole est lancé lorsque l'agent «Synchronisateur et Contrôleur de communication» qui joue le rôle «Gestion de la table TDMA» exécute le plan «Envoi des valeurs courantes des compteurs de cycle et de Slot» pour informer les agents «Protocol Control Operation» et «Bus Guardian» que le slot courant est attribué à leur nœuds pour émettre une trame statique. Quand l'agent «Protocol Control Operation» qui joue le rôle «L'émission d'une trame statique» reçoit ce message, il exécute le plan «Réception des informations de synchronisation», pendant l'exécution de ce plan cet agent reçoit les données auprès de l'agent «Controller Host Interface». La fin de l'exécution de ce plan consiste à envoyer une trame statique à l'agent «Bus Driver». Lorsque ce dernier reçoit la trame il envoie une demande d'autorisation auprès de l'agent «Bus Guardian» et il attend. Lorsqu'il reçoit cette autorisation il envoie la trame à l'agent «Bus de Communication» (voir Fig. V-2-26).

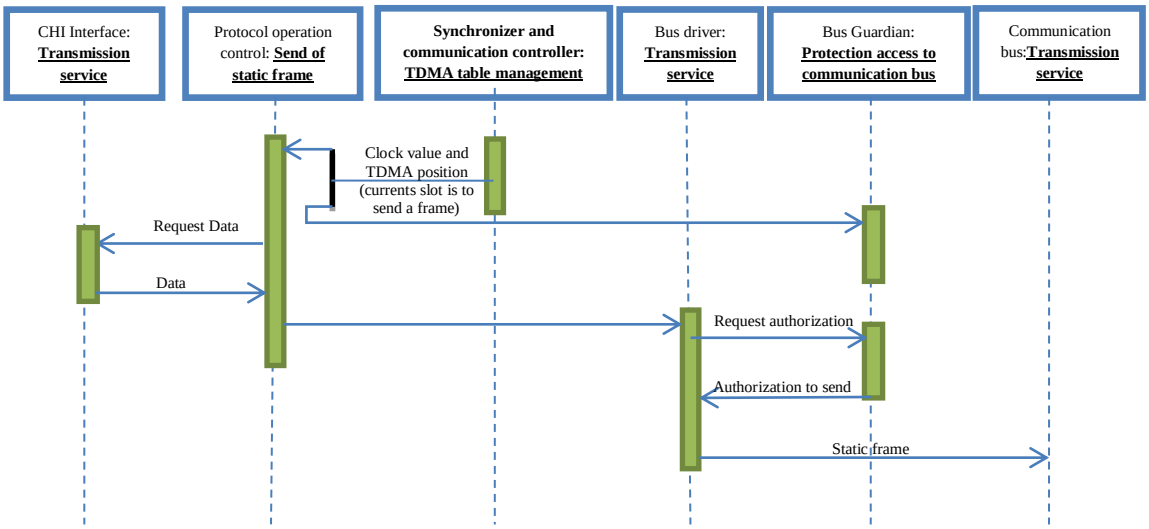


Fig. V-2-26 : Diagramme de protocole d'émission d'une trame statique

### 1.7.2. Diagramme de protocole d'émission d'une trame dynamique

Ce protocole est lancé lorsque l'agent «Synchronisateur et Contrôleur de communication» qui joue le rôle «Synchronisation d'horloges et control du déroulement des segments» exécute le plan «Envoie l'information Début de Segment Dynamique» pour informer l'agent «Protocol Control Operation» que le segment dynamique est lancé. Quand ce dernier qui joue le rôle «Contrôler la stratégie FTDMA d'accès au bus de communication» reçoit ce message, il exécute le plan «Réception l'information Début de Segment Dynamique». Le résultat de cette exécution est l'envoi d'une autorisation de lancement de la stratégie FTDMA à l'agent «Générateur» qui joue le rôle «Génération des Messages Evènements», lorsque ce dernier reçoit ce signal il exécute le plan «Réception de l'autorisation de lancer la stratégie FTDMA d'accès au bus de communication». Le résultat de l'exécution de ce plan est la génération aléatoire d'un message avec son priorité et l'envoi de ce message à l'agent «Arbitraire» qui joue le rôle «Arbitrage et L'émission d'une trame dynamique». Lorsque l'arbitraire reçoit le message il exécute le plan «Réception d'un message évènement». Le résultat de l'exécution de ce plan est l'envoi de la priorité de message à l'agent «Bus Driver». Pendant une période d'écoute, l'agent «Arbitraire» exécute le plan «Réception de l'identification d'un message» lorsqu'il reçoit les priorités des autres messages, le résultat de ce plan est la sélection de message le plus prioritaire, si le message local de cet agent est le plus prioritaire il envoie une trame dynamique à l'agent «Bus Driver», sinon il attend pour un certain temps ensuite il envoie une autorisation de génération à l'agent «Générateur». Lorsque l'agent «Bus Driver» reçoit la priorité d'un message ou une trame il exécute le plan «Transmission d'informations» pour envoyer cette information à l'agent «Bus de Communication» (voir Fig. V-2-27).

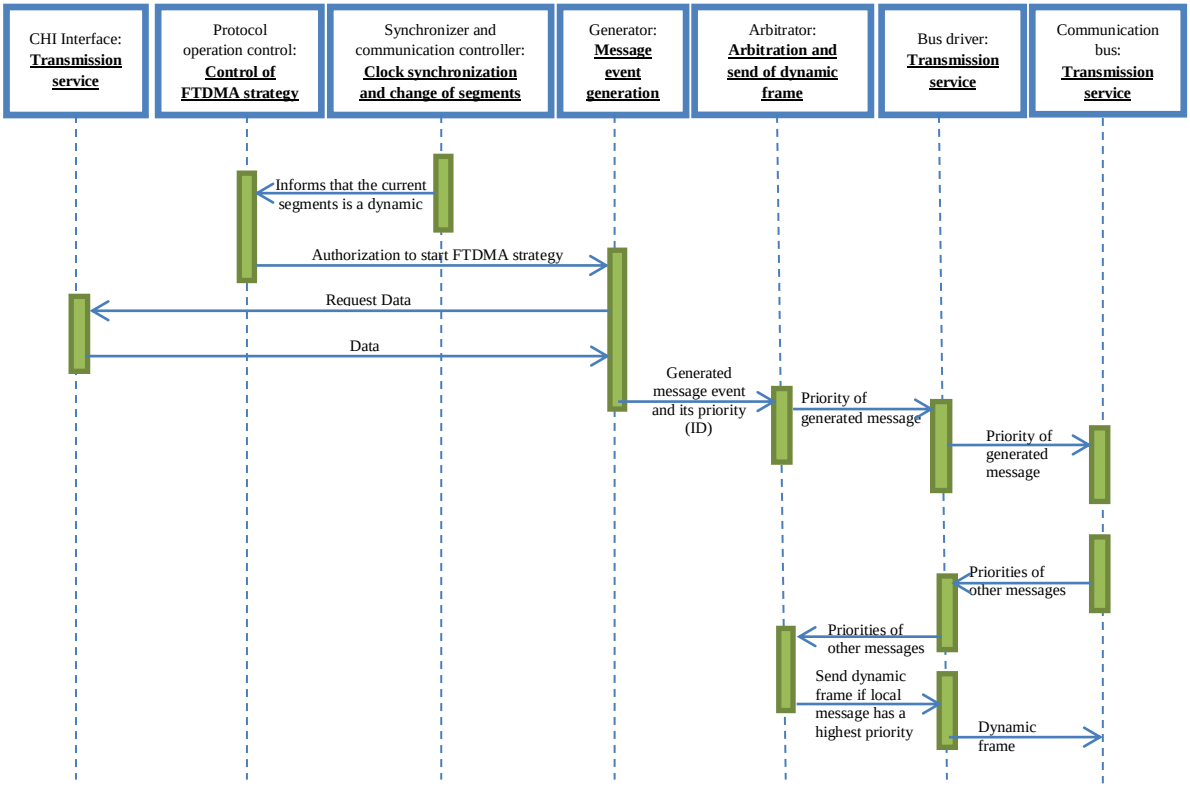


Fig. V-2-27 : Diagramme de protocole de l'émission d'une trame dynamique

## 2. Translation du modèle conceptuel de protocole FlexRay en code JADE

### 2.2. Translation de diagramme de domaine

```
packageFlexrayAgents;
importjade.content.onto.*;
importjade.content.schema.*;
importjade.content.onto.BasicOntology;
importjade.util.leap.List;
publicclassFlexRayFrameOntologyextends Ontology{
publicfinalstatic String ONTOLOGY_NAME = "FlexRayFrameOntology";
publicfinalstatic String FlexRayFrame = "FlexRayFrame";
publicfinalstatic String Reserved_Bit = "Reserved_Bit";
publicfinalstatic String Payload_Preamble_Indicator= "Payload_Preamble_Indicator";
publicfinalstatic String Null_Frame_Indicator= "Null_Frame_Indicator";
publicfinalstatic String Sync_Frame_Indicator= "Sync_Frame_Indicator";
publicfinalstatic String Startup_Frame_Indicator= "Startup_Frame_Indicator";
publicfinalstatic String Frame_ID= "Frame_ID";
publicfinalstatic String Payload_Length= "Payload_Length";
publicfinalstatic String Header_CRC= "Header_CRC";
publicfinalstatic String Cycle_Count= "Cycle_Count";
publicfinalstatic String Network_Management_Vector= "Network_Management_Vector";
publicfinalstatic String Message_ID= "Message_ID";
publicfinalstatic String Data= "Data";
publicfinalstatic String Frame_CRC= "Frame_CRC";

publicstaticfinal String OWNS = "Owns";
publicstaticfinal String OWNS_OWNER = "owner";
publicstaticfinal String OWNS_ITEM = "item";

privatestatic Ontology theInstance = newFlexRayFrameOntology();

publicstatic Ontology getInstance() {
        returntheInstance;
        }

privateFlexRayFrameOntology(){
super(ONTOLOGY_NAME, BasicOntology.getInstance());
try{
add(newConceptSchema(FlexRayFrame),FlexRayFrame.class);
add(newPredicateSchema(OWNS), Owns.class);
ConceptSchemacs = (ConceptSchema) getSchema(FlexRayFrame);
cs.add(Reserved_Bit, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Payload_Preamble_Indicator, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Null_Frame_Indicator, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Sync_Frame_Indicator, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Startup_Frame_Indicator, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Frame_ID, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Payload_Length, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Header_CRC, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
cs.add(Cycle_Count, (PrimitiveSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);
cs.add(Network_Management_Vector, (PrimitiveSchema) getSchema(BasicOntology.STRING),0,ObjectSchema.UNLIMITED);
cs.add(Message_ID, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add(Data, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add(Frame_CRC, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));

PredicateSchemaps = (PredicateSchema) getSchema(OWNS);
ps.add(OWNS_OWNER, (ConceptSchema) getSchema(BasicOntology.AID));
ps.add(OWNS_ITEM, (ConceptSchema) getSchema(FlexRayFrame));
} catch (OntologyExceptionoe) {oe.printStackTrace();}
}
}
```

## 2.3. Translation de diagramme de plan «transmission d'informations» du rôle «Service de transmission»

```
Transmission_of_Informations = newFSMBehaviour(this) {
};
// States
Transmission_of_Informations.registerFirstState(newWait_Reception_Information(),"Wait_Reception_Information");
Transmission_of_Informations.registerState(new Transmission_Received_Information(),"Transmission_Received_Information");
// Transitions
Transmission_of_Informations.registerTransition("Wait_Reception_Information","Wait_Reception_Information",1);
Transmission_of_Informations.registerTransition("Wait_Reception_Information","Transmission_Received_Information",0);
Transmission_of_Informations.registerDefaultTransition("Transmission_Received_Information","Wait_Reception_Information");
```

## 2.4. Translation de diagramme de plan «Réception des informations de synchronisation» du rôle «Emission d'une trame statique»

```
Reception_Synchronization_Information_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Synchronization_Information_Behaviour.registerFirstState(new
Wait_Reception_Synchronization_Information(),"Wait_Reception_Synchronization_Information");
Reception_Synchronization_Information_Behaviour.registerState(new
Fill_Frame_ID_Cycle_Count_Fields(),"Fill_Frame_ID_Cycle_Count_Fields");
Reception_Synchronization_Information_Behaviour.registerState(newRequest_Frame_Data(),"Request_Frame_Data");
Reception_Synchronization_Information_Behaviour.registerState(newWait_Reception_Frame_Data(),"Wait_Reception_Frame_Data");
Reception_Synchronization_Information_Behaviour.registerState(newFill_Frame_Data_Field(),"Fill_Frame_Data_Field");
Reception_Synchronization_Information_Behaviour.registerState(newSend_Frame(),"Send_Frame");
// Transitions
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Synchronization_Information","Wait_Reception
_Synchronization_Information",0);
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Synchronization_Information","Fill_Frame_ID_
Cycle_Count_Fields",1);
Reception_Synchronization_Information_Behaviour.registerTransition("Fill_Frame_ID_Cycle_Count_Fields","Request_Frame_Data",2);
Reception_Synchronization_Information_Behaviour.registerTransition("Fill_Frame_ID_Cycle_Count_Fields","Send_Frame",1);
Reception_Synchronization_Information_Behaviour.registerDefaultTransition("Request_Frame_Data","Wait_Reception_Frame_Data");
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Frame_Data","Wait_Reception_Frame_Data",0);
Reception_Synchronization_Information_Behaviour.registerTransition("Wait_Reception_Frame_Data","Fill_Frame_Data_Field",1);
Reception_Synchronization_Information_Behaviour.registerDefaultTransition("Fill_Frame_Data_Field","Send_Frame");
Reception_Synchronization_Information_Behaviour.registerDefaultTransition("Send_Frame","Wait_Reception_Synchronization_Inform
ation");
```

## 2.5. Translation des diagrammes de plan du rôle «Arbitration et émission d'une trame dynamique»

### 2.5.2. Translation du Plan «Réception d'un message évènement»

```
Reception_Message_Event_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Message_Event_Behaviour.registerFirstState(newWait_Reception_Message_Event(),"Wait_Reception_Message_Event");
Reception_Message_Event_Behaviour.registerState(new Send_Message_Identification_To_Bus(),"Send_Message_Identification_To_Bus");
// Transitions
Reception_Message_Event_Behaviour.registerTransition("Wait_Reception_Message_Event","Wait_Reception_Message_Event",0);
Reception_Message_Event_Behaviour.registerTransition("Wait_Reception_Message_Event","Send_Message_Identification_To_Bus",1);
Reception_Message_Event_Behaviour.registerDefaultTransition("Send_Message_Identification_To_Bus","Wait_Reception_Message_Event");
```

### 2.5.3. Translation du plan «Réception de l'identification d'un message»

```
Reception_Identification_Message_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Identification_Message_Behaviour.registerFirstState(new Listen(),"Listen");
Reception_Identification_Message_Behaviour.registerState(newAdd_In_Messages_Table(),"Add_In_Messages_Table");
Reception_Identification_Message_Behaviour.registerState(newSelection_Highest_Priority(),"Selection_Highest_Priority");
Reception_Identification_Message_Behaviour.registerState(newSend_Dynamic_Frame(),"Send_Dynamic_Frame");
// Transitions
Reception_Identification_Message_Behaviour.registerTransition("Listen","Listen",0);
Reception_Identification_Message_Behaviour.registerTransition("Listen","Add_In_Messages_Table",1);
Reception_Identification_Message_Behaviour.registerTransition("Listen","Selection_Highest_Priority",2);
Reception_Identification_Message_Behaviour.registerDefaultTransition("Add_In_Messages_Table","Listen");
Reception_Identification_Message_Behaviour.registerTransition("Selection_Highest_Priority","Send_Dynamic_Frame",1);
Reception_Identification_Message_Behaviour.registerTransition("Selection_Highest_Priority","Listen",0);
Reception_Identification_Message_Behaviour.registerDefaultTransition("Send_Dynamic_Frame","Listen");
```

## 2.6. Translation du plan «Réveiller les nœuds du cluster» du rôle «Wakeup»

```
Wakeup_Behaviour= newFSMBehaviour(this) {
};
// States
Wakeup_Behaviour.registerFirstState(newWakeUp_Listen(),"WakeUp_Listen");
Wakeup_Behaviour.registerState(newWakeUp_Send(),"WakeUp_Send");
Wakeup_Behaviour.registerState(newWakeUp_Second_Listen(),"WakeUp_Second_Listen");
Wakeup_Behaviour.registerState(newWakeUp_Detect(),"WakeUp_Detect");
Wakeup_Behaviour.registerState(newWakeUp_Success(),"WakeUp_Success");
// Transitions
Wakeup_Behaviour.registerTransition("WakeUp_Listen","WakeUp_Listen",0);
Wakeup_Behaviour.registerTransition("WakeUp_Listen","WakeUp_Send",1);
Wakeup_Behaviour.registerTransition("WakeUp_Listen","WakeUp_Success",2);
Wakeup_Behaviour.registerDefaultTransition("WakeUp_Send","WakeUp_Second_Listen");
Wakeup_Behaviour.registerTransition("WakeUp_Second_Listen","WakeUp_Second_Listen",0);
Wakeup_Behaviour.registerTransition("WakeUp_Second_Listen","WakeUp_Detect",1);
Wakeup_Behaviour.registerTransition("WakeUp_Second_Listen","WakeUp_Success",2);
Wakeup_Behaviour.registerDefaultTransition("WakeUp_Detect","WakeUp_Success");
Wakeup_Behaviour.registerDefaultTransition("WakeUp_Success","WakeUp_Listen");
```

## 2.7. Translation du plan «Initialisation le Cluster» du rôle «Startup»

```
Startup_Behaviour= newFSMBehaviour(this) {
};
// States
Startup_Behaviour.registerFirstState(new Listen(),"Listen");
Startup_Behaviour.registerState(newCollision_Resolution(),"Collision_Resolution");
Startup_Behaviour.registerState(new Consistency_Check_Wait_First_ColdStart_Frame(),"Consistency_Check_Wait_First_ColdStart_Frame");
Startup_Behaviour.registerState(new
Consistency_Check_Wait_Second_ColdStart_Frame(),"Consistency_Check_Wait_Second_ColdStart_Frame");
Startup_Behaviour.registerState(new
Consistency_Check_Wait_Result_First_ColdStart_Frame(),"Consistency_Check_Wait_Result_First_ColdStart_Frame");
Startup_Behaviour.registerState(new
Consistency_Check_Wait_Result_Second_ColdStart_Frame(),"Consistency_Check_Wait_Result_Second_ColdStart_Frame");
Startup_Behaviour.registerState(new Initialize_Schedule_Wait_First_ColdStart_Frame(),"Initialize_Schedule_Wait_First_ColdStart_Frame");
Startup_Behaviour.registerState(new
Initialize_Schedule_Wait_Second_ColdStart_Frame(),"Initialize_Schedule_Wait_Second_ColdStart_Frame");
Startup_Behaviour.registerState(new Integration_Check_Wait_First_ColdStart_Frame(),"Integration_Check_Wait_First_ColdStart_Frame");

Startup_Behaviour.registerState(new
Integration_Check_Wait_Second_ColdStart_Frame(),"Integration_Check_Wait_Second_ColdStart_Frame");
Startup_Behaviour.registerState(new
Integration_Check_Wait_Result_First_ColdStart_Frame(),"Integration_Check_Wait_Result_First_ColdStart_Frame");
Startup_Behaviour.registerState(new
Integration_Check_Wait_Result_Second_ColdStart_Frame(),"Integration_Check_Wait_Result_Second_ColdStart_Frame");
Startup_Behaviour.registerState(new
Integration_Check_Second_Part_Non_ColdStart_Nodes(),"Integration_Check_Second_Part_Non_ColdStart_Nodes");
Startup_Behaviour.registerState(newStartup_Success(),“Startup_Success");
Startup_Behaviour.registerState(new Join(),"Join");
// Transitions
Startup_Behaviour.registerTransition("Listen","Listen",0);
Startup_Behaviour.registerTransition("Listen","Initialize_Schedule_Wait_Second_ColdStart_Frame",3);
Startup_Behaviour.registerTransition("Listen","Initialize_Schedule_Wait_First_ColdStart_Frame",1);
Startup_Behaviour.registerTransition("Initialize_Schedule_Wait_First_ColdStart_Frame","Initialize_Schedule_Wait_First_ColdStart_Frame",0)
;
Startup_Behaviour.registerTransition("Initialize_Schedule_Wait_First_ColdStart_Frame","Initialize_Schedule_Wait_Second_ColdStart_Frame"
,1);
Startup_Behaviour.registerTransition("Initialize_Schedule_Wait_Second_ColdStart_Frame","Initialize_Schedule_Wait_Second_ColdStart_Fra
me",0);
Startup_Behaviour.registerTransition("Initialize_Schedule_Wait_Second_ColdStart_Frame","Integration_Check_Wait_First_ColdStart_Frame",
1);
Startup_Behaviour.registerTransition("Integration_Check_Wait_First_ColdStart_Frame","Integration_Check_Wait_First_ColdStart_Frame",0);
Startup_Behaviour.registerTransition("Integration_Check_Wait_First_ColdStart_Frame","Integration_Check_Wait_Result_First_ColdStart_Fra
me",1);
Startup_Behaviour.registerTransition("Integration_Check_Wait_First_ColdStart_Frame","Listen",2);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_First_ColdStart_Frame","Integration_Check_Wait_Result_First_ColdSt
art_Frame",0);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_First_ColdStart_Frame","Integration_Check_Wait_Second_ColdStart_F
rame",1); Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_First_ColdStart_Frame","Listen",2);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Second_ColdStart_Frame","Integration_Check_Wait_Second_ColdStart_Frame
",0);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Second_ColdStart_Frame","Integration_Check_Wait_Result_Second_ColdStart
_Frame",1);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Second_ColdStart_Frame","Listen",2);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_Second_ColdStart_Frame","Integration_Check_Wait_Result_Second_C
oldStart_Frame",0); Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_Second_ColdStart_Frame","Listen",2);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_Second_ColdStart_Frame","Integration_Check_Second_Part_Non_Cold
Start_Nodes",3);
Startup_Behaviour.registerTransition("Integration_Check_Wait_Result_Second_ColdStart_Frame","Join",1);
Startup_Behaviour.registerTransition("Integration_Check_Second_Part_Non_ColdStart_Nodes","Integration_Check_Second_Part_Non_ColdSt
art_Nodes",0);
Startup_Behaviour.registerTransition("Integration_Check_Second_Part_Non_ColdStart_Nodes","Startup_Success",1);
Startup_Behaviour.registerTransition("Join","Join",0);
Startup_Behaviour.registerTransition("Join","Listen",2);
Startup_Behaviour.registerTransition("Join","Startup_Success",1);
Startup_Behaviour.registerDefaultTransition("Startup_Success","Listen");
```

## 2.8. Translation des diagrammes de plan du rôle «Synchronisation d'horloges et control du déroulement des segments»

### 2.8.1. Translation du plan «Calcul des valeurs de correction d'offset et de rate»

```
Clock_Synchronization_Calculation_Behaviour= newFSMBehaviour(this) {
};
// States
Clock_Synchronization_Calculation_Behaviour.registerFirstState(newWait_Reception_of_Frame(),"Wait_Reception_of_Frame");
Clock_Synchronization_Calculation_Behaviour.registerState(newMeasurement_Phase(),"Measurement_Phase");
Clock_Synchronization_Calculation_Behaviour.registerState(new
Calculation_of_Rate_Correction_Value(),"Calculation_of_Rate_Correction_Value");
Clock_Synchronization_Calculation_Behaviour.registerState(new
Calculation_of_Offset_Correction_Value(),"Calculation_of_Offset_Correction_Value");
// Transitions
Clock_Synchronization_Calculation_Behaviour.registerTransition("Wait_Reception_of_Frame","Wait_Reception_of_Frame",0);
Clock_Synchronization_Calculation_Behaviour.registerTransition("Wait_Reception_of_Frame","Measurement_Phase",1);
Clock_Synchronization_Calculation_Behaviour.registerTransition("Measurement_Phase","Wait_Reception_of_Frame",0);
Clock_Synchronization_Calculation_Behaviour.registerTransition("Wait_Reception_of_Frame","Calculation_of_Offset_Correction_Value",2);
Clock_Synchronization_Calculation_Behaviour.registerTransition("Calculation_of_Offset_Correction_Value","Wait_Reception_of_Frame",0);
Clock_Synchronization_Calculation_Behaviour.registerTransition("Calculation_of_Offset_Correction_Value","Calculation_of_Rate_Correction_Va
lue",1);
Clock_Synchronization_Calculation_Behaviour.registerDefaultTransition("Calculation_of_Rate_Correction_Value","Wait_Reception_of_Frame");
```

### 2.8.2. Translation du plan «Changement de segment et l'application de la correction de rate et l'offset»

```
Change_Of_Segment_Behaviour= newFSMBehaviour(this) {
};
// States
Change_Of_Segment_Behaviour.registerFirstState(newStatic_Segment(),"Static_Segment");
Change_Of_Segment_Behaviour.registerState(newDynamic_Segment(),"Dynamic_Segment");
Change_Of_Segment_Behaviour.registerState(newSymbol_Segment(),"Symbol_Segment");
Change_Of_Segment_Behaviour.registerState(newNIT_Segment(),"NIT_Segment");
// Transitions
Change_Of_Segment_Behaviour.registerTransition("Static_Segment","Static_Segment",0);
Change_Of_Segment_Behaviour.registerTransition("Static_Segment","Dynamic_Segment",1);
Change_Of_Segment_Behaviour.registerTransition("Dynamic_Segment","Dynamic_Segment",0);
Change_Of_Segment_Behaviour.registerTransition("Dynamic_Segment","Symbol_Segment",1);
Change_Of_Segment_Behaviour.registerTransition("Symbol_Segment","Symbol_Segment",0);
Change_Of_Segment_Behaviour.registerTransition("Symbol_Segment","NIT_Segment",1);
Change_Of_Segment_Behaviour.registerTransition("NIT_Segment","NIT_Segment",0);
Change_Of_Segment_Behaviour.registerTransition("NIT_Segment","Static_Segment",1);
```

## 2.8.3. Translation du plan «Validation du slot de la trame coldstart reçue»

```
Reception_Validation_Frame_Request_Behaviour = newFSMBehaviour(this) {
};
// States
Reception_Validation_Frame_Request_Behaviour.registerFirstState(new
Wait_Reception_Of_ID_Cold_Start_Frame(),"Wait_Reception_Of_ID_Cold_Start_Frame");
Reception_Validation_Frame_Request_Behaviour.registerState(new
Validation_Of_ID_Received_Cold_Start_Frame(),"Validation_Of_ID_Received_Cold_Start_Frame");
Reception_Validation_Frame_Request_Behaviour.registerState(newSend_Validation_Result(),"Send_Validation_Result");
// Transitions
Reception_Validation_Frame_Request_Behaviour.registerTransition("Wait_Reception_Of_ID_Cold_Start_Frame","Wait_Reception_Of_ID_Cold_Star
t_Frame",0);
Reception_Validation_Frame_Request_Behaviour.registerTransition("Wait_Reception_Of_ID_Cold_Start_Frame","Validation_Of_ID_Received_Cold
_Start_Frame",1);
Reception_Validation_Frame_Request_Behaviour.registerDefaultTransition("Validation_Of_ID_Received_Cold_Start_Frame","Send_Validation_Resu
lt");
Reception_Validation_Frame_Request_Behaviour.registerDefaultTransition("Send_Validation_Result","Wait_Reception_Of_ID_Cold_Start_Frame");
```

## 2.8.4. Translation du plan «Envoi de l'information début de segment dynamique»

```
Send_Begin_Dynamic_Segment_Information= newFSMBehaviour(this) {
};
// States
Send_Begin_Dynamic_Segment_Information.registerFirstState(new
Wait_Begin_of_Dynamic_Segment(),"Wait_Begin_of_Dynamic_Segment");
Send_Begin_Dynamic_Segment_Information.registerState(new
Send_Begin_of_Dynamic_Segment_Information(),"Send_Begin_of_Dynamic_Segment_Information");  // Transitions
Send_Begin_Dynamic_Segment_Information.registerTransition("Wait_Begin_of_Dynamic_Segment","Wait_Begin_of_Dynamic_Segment",0);
Send_Begin_Dynamic_Segment_Information.registerTransition("Wait_Begin_of_Dynamic_Segment","Send_Begin_of_Dynamic_Segment_Infor
rmation",1);
Send_Begin_Dynamic_Segment_Information.registerDefaultTransition("Send_Begin_of_Dynamic_Segment_Information","Wait_Begin_of_Dy
namic_Segment");
```

## 2.8.5. Translation du plan «Envoi de l'information fin de segment dynamique»

```
Send_End_Dynamic_Segment_Information= newFSMBehaviour(this) {
};
// States
Send_End_Dynamic_Segment_Information.registerFirstState(newWait_End_of_Dynamic_Segment(),"Wait_End_of_Dynamic_Segment");
Send_End_Dynamic_Segment_Information.registerState(new
Send_End_of_Dynamic_Segment_Information(),"Send_End_of_Dynamic_Segment_Information");
// Transitions
Send_End_Dynamic_Segment_Information.registerTransition("Wait_End_of_Dynamic_Segment","Wait_End_of_Dynamic_Segment",0);
Send_End_Dynamic_Segment_Information.registerTransition("Wait_End_of_Dynamic_Segment","Send_End_of_Dynamic_Segment_Informati
on",1);
Send_End_Dynamic_Segment_Information.registerDefaultTransition("Send_End_of_Dynamic_Segment_Information","Wait_End_of_Dynamic
_Segment");
```

## 2.9. Translation du plan «Changement des états du contrôleur de communication» du rôle «Contrôler l'état du contrôleur de communication»

```
Communication_Controller_States_Behaviour= newFSMBehaviour(this) {
};
// States
Communication_Controller_States_Behaviour.registerFirstState(newDefault_Config(),"Default_Config");
Communication_Controller_States_Behaviour.registerState(newConfig(),"Config");
Communication_Controller_States_Behaviour.registerState(new Ready(),"Ready");
Communication_Controller_States_Behaviour.registerState(new Wakeup(),"Wakeup");
Communication_Controller_States_Behaviour.registerState(new Startup(),"Startup");
Communication_Controller_States_Behaviour.registerState(newNormal_Active(),"Normal_Active");
Communication_Controller_States_Behaviour.registerState(newSend_Wakeup_Status_To_Host(),"Send_Wakeup_Status_To_Host");
Communication_Controller_States_Behaviour.registerState(newSend_Request_Configuration_Access_Channel(),"Send_Request_Configuration
_Access_Channel");
// Transitions
Communication_Controller_States_Behaviour.registerTransition("Default_Config","Default_Config",0);
Communication_Controller_States_Behaviour.registerTransition("Default_Config","Send_Request_Configuration_Access_Channel",1);
Communication_Controller_States_Behaviour.registerDefaultTransition("Send_Request_Configuration_Access_Channel","Config");
Communication_Controller_States_Behaviour.registerTransition("Config","Config",0);
Communication_Controller_States_Behaviour.registerTransition("Config","Ready",1);
Communication_Controller_States_Behaviour.registerTransition("Ready","Ready",0);
Communication_Controller_States_Behaviour.registerTransition("Ready","Wakeup",1);
Communication_Controller_States_Behaviour.registerTransition("Ready","Startup",2);
Communication_Controller_States_Behaviour.registerTransition("Wakeup","Wakeup",0);
Communication_Controller_States_Behaviour.registerTransition("Wakeup","Send_Wakeup_Status_To_Host",1);
Communication_Controller_States_Behaviour.registerDefaultTransition("Send_Wakeup_Status_To_Host","Ready");
Communication_Controller_States_Behaviour.registerTransition("Startup","Startup",0);
Communication_Controller_States_Behaviour.registerTransition("Startup","Normal_Active",1);
Communication_Controller_States_Behaviour.registerTransition("Normal_Active","Normal_Active",0);
```

## 2.10. Translation du plan «Envoi des valeurs courantes des compteurs de cycle et de slot» du rôle «Gestion de la table TDMA»

```
Send_Slot_and_Cycle_Values_Behaviour = newFSMBehaviour(this) {
};
Send_Slot_and_Cycle_Values_Behaviour.registerFirstState(new Get_Current_Position_TDMA_Table(),"Get_Current_Position_TDMA_Table");
Send_Slot_and_Cycle_Values_Behaviour.registerState(new Send_Currents_Slot_and_Cycle_Values(),"Send_Currents_Slot_and_Cycle_Values");
// Transitions
Send_Slot_and_Cycle_Values_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Get_Current_Position_TDMA_Table",0);
Send_Slot_and_Cycle_Values_Behaviour.registerTransition("Get_Current_Position_TDMA_Table","Send_Currents_Slot_and_Cycle_Values",1);
Send_Slot_and_Cycle_Values_Behaviour.registerDefaultTransition("Send_Currents_Slot_and_Cycle_Values","Get_Current_Position_TDMA_Table");
```

## 2.11. Translation des diagrammes de plan du rôle «Contrôler la stratégie FTDMA d'accès au bus de communication»

### 2.11.1. Translation du plan «Réception de l'information début de segment dynamique»

```
Reception_Begin_Dynamic_Segment_Information_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Begin_Dynamic_Segment_Information_Behaviour.registerFirstState(new
Wait_Reception_Begin_Dynamic_Segment_Information(),"Wait_Reception_Begin_Dynamic_Segment_Information");
Reception_Begin_Dynamic_Segment_Information_Behaviour.registerState(new
Send_Begin_Dynamic_Segment_Information_If_State_Active_Normal(),"Send_Begin_Dynamic_Segment_Information_If_State_Active_Normal
"); // Transitions
Reception_Begin_Dynamic_Segment_Information_Behaviour.registerTransition("Wait_Reception_Begin_Dynamic_Segment_Information","Wai
t_Reception_Begin_Dynamic_Segment_Information",0);
Reception_Begin_Dynamic_Segment_Information_Behaviour.registerTransition("Wait_Reception_Begin_Dynamic_Segment_Information","Sen
d_Begin_Dynamic_Segment_Information_If_State_Active_Normal",1);
Reception_Begin_Dynamic_Segment_Information_Behaviour.registerDefaultTransition("Send_Begin_Dynamic_Segment_Information_If_State_
Active_Normal","Wait_Reception_Begin_Dynamic_Segment_Information");
```

### 2.11.2. Translation du plan «Réception de l'information fin de segment dynamique»

```
Reception_End_Dynamic_Segment_Information_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_End_Dynamic_Segment_Information_Behaviour.registerFirstState(new
Wait_Reception_End_Dynamic_Segment_Information(),"Wait_Reception_End_Dynamic_Segment_Information");
Reception_End_Dynamic_Segment_Information_Behaviour.registerState(new
Send_End_Dynamic_Segment_Information_If_State_Active_Normal(),"Send_End_Dynamic_Segment_Information_If_State_Active_Normal");
// Transitions
Reception_End_Dynamic_Segment_Information_Behaviour.registerTransition("Wait_Reception_End_Dynamic_Segment_Information","Wait_R
eception_End_Dynamic_Segment_Information",0);
Reception_End_Dynamic_Segment_Information_Behaviour.registerTransition("Wait_Reception_End_Dynamic_Segment_Information","Send_E
nd_Dynamic_Segment_Information_If_State_Active_Normal",1);
Reception_End_Dynamic_Segment_Information_Behaviour.registerDefaultTransition("Send_End_Dynamic_Segment_Information_If_State_Acti
ve_Normal","Wait_Reception_End_Dynamic_Segment_Information");
```

## 2.12. Translation des diagrammes de plan du rôle «Génération des messages événement»

### 2.12.1. Translation du plan «Réception de l'autorisation de lancer la stratégie FTDMA d'accès au bus de communication»

```
Reception_Authorization_To_Begin_FTDMA_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerFirstState(new
Wait_Reception_Begin_Dynamic_Segment_Information(),"Wait_Reception_Begin_Dynamic_Segment_Information");
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerState(newRandom_Generation_Message(),"Random_Generation_Message");

Reception_Authorization_To_Begin_FTDMA_Behaviour.registerState(new
Send_Message_To_Arbitartion_Role(),"Send_Message_To_Arbitartion_Role");
// Transitions
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerTransition("Wait_Reception_Begin_Dynamic_Segment_Information","Wait_Rec
eption_Begin_Dynamic_Segment_Information",0);
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerTransition("Wait_Reception_Begin_Dynamic_Segment_Information","Random_
Generation_Message",1);
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerTransition("Random_Generation_Message","Random_Generation_Message",0);
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerTransition("Random_Generation_Message","Send_Message_To_Arbitartion_Rol
e",1);
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerTransition("Random_Generation_Message","Wait_Reception_Begin_Dynamic_S
egment_Information",2);
Reception_Authorization_To_Begin_FTDMA_Behaviour.registerDefaultTransition("Send_Message_To_Arbitartion_Role","Random_Generation_
Message");
```

### 2.12.2. Translation du plan «Réception de la commande d'arrêter la stratégie FTDMA d'accès au bus de communication»

```
Reception_Stop_of_FTDMA_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Stop_of_FTDMA_Behaviour.registerFirstState(new
Wait_Reception_End_Dynamic_Segment_Information(),"Wait_Reception_End_Dynamic_Segment_Information");
Reception_Stop_of_FTDMA_Behaviour.registerState(newStop_Generation_Message(),"Stop_Generation_Message");
// Transitions
Reception_Stop_of_FTDMA_Behaviour.registerTransition("Wait_Reception_End_Dynamic_Segment_Information","Wait_Reception_End_Dyna
mic_Segment_Information",0);
Reception_Stop_of_FTDMA_Behaviour.registerTransition("Wait_Reception_End_Dynamic_Segment_Information","Stop_Generation_Message",
1);
Reception_Stop_of_FTDMA_Behaviour.registerDefaultTransition("Stop_Generation_Message","Wait_Reception_End_Dynamic_Segment_Infor
mation");
```

### 2.12.3. Translation du plan «Réception de l'autorisation de génération auprès de rôle arbitrage»

```
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour= newFSMBehaviour(this) {
};
// States
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour.registerFirstState(newWait_Reception_Autorization(),"Wait_Reception_Autorization");
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour.registerState(new
Update_Arbitration_Autorization_Variable(),"Update_Arbitration_Autorization_Variable");
// Transitions
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour.registerTransition("Wait_Reception_Autorization","Wait_Reception_Autorization",0);
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour.registerTransition("Wait_Reception_Autorization","Update_Arbitration_Autorization_Variable",1);
Reception_Generation_Autorisation_From_Arbitration_Role_Behaviour.registerDefaultTransition("Update_Arbitration_Autorization_Variable","Wait_Reception_Autorization");
```

# Chapitre VI

# Environnement développé et analyse des résultats de simulation du protocole FlexRay

## 1. Introduction

Nous présentons, dans ce chapitre notre environnement développé pour la simulation du protocole FlexRay. Nous avons utilisé comme langage de programmation, le langage Java eclipse. Ce langage est en fait très puissant et couvre plusieurs domaines d'applications. Nous avons choisi ce langage pour la programmation de notre environnement puisqu'il est facile d'intégrer la plateforme multi-agent Jade avec Java eclipse. Cette intégration de la plateforme Jade dans l'environnement développé est pour pouvoir exploiter les concepts de Jade pour l'implémentation multi-agent du protocole de communication FlexRay. Comme nous avons décrit dans le chapitre 5 (partie II), un cluster FlexRay est translaté en une plateforme Jade, et un nœud est translaté en container Jade, et un agent d'un nœud FlexRay en un agent Jade. Fig. VI-1 présente un cluster sous la plateforme Jade qui est composé de quatre nœuds.

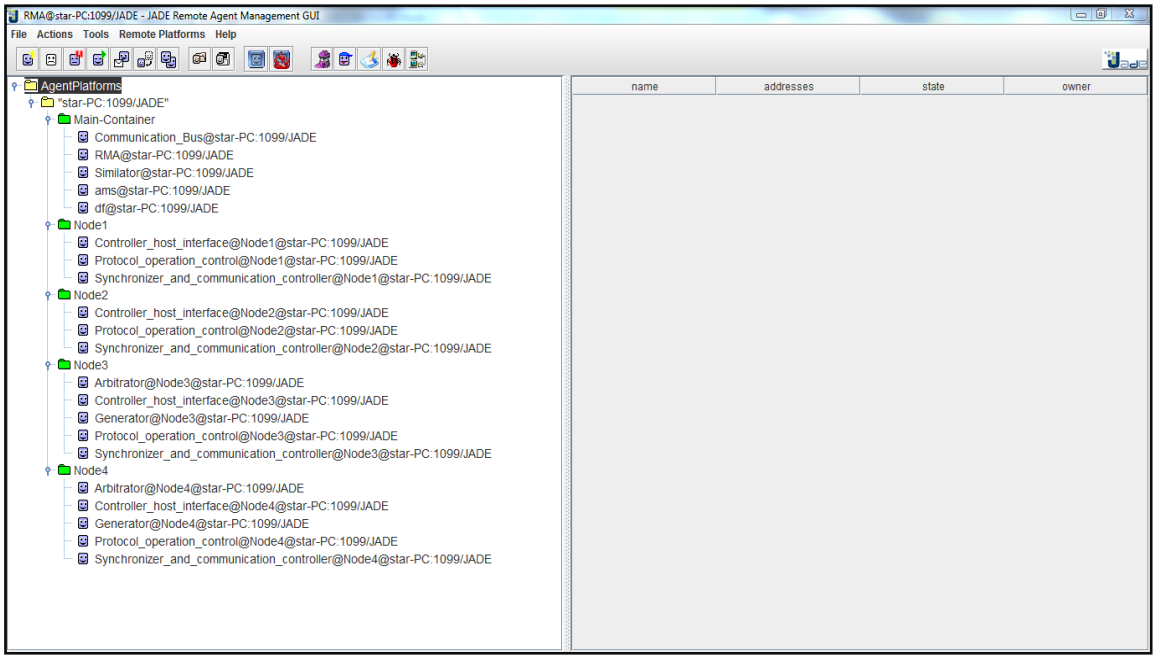


Fig. VI-1 : Exemple d'un cluster composé de quatre nœuds sous la plateforme Jade

## 2. Description de l'environnement développé

L'environnement de simulation basée multi-agent du protocole FlexRay développé présente trois fenêtres ; une pour introduire les paramètres généraux de la simulation ; la deuxième pour les nœuds du cluster, où chaque nœud est représenté sous forme d'un ensemble d'agents qui s'affichent dans le même panneau, et la dernière fenêtre pour afficher les états du bus de communication. Fig. VI-2 montre les agents d'un nœud dans l'environnement développé qui sont :«Protocol operation control», «Synchronizer and communcation controller», «Controller host interface», «Generator», and «Arbitrator».

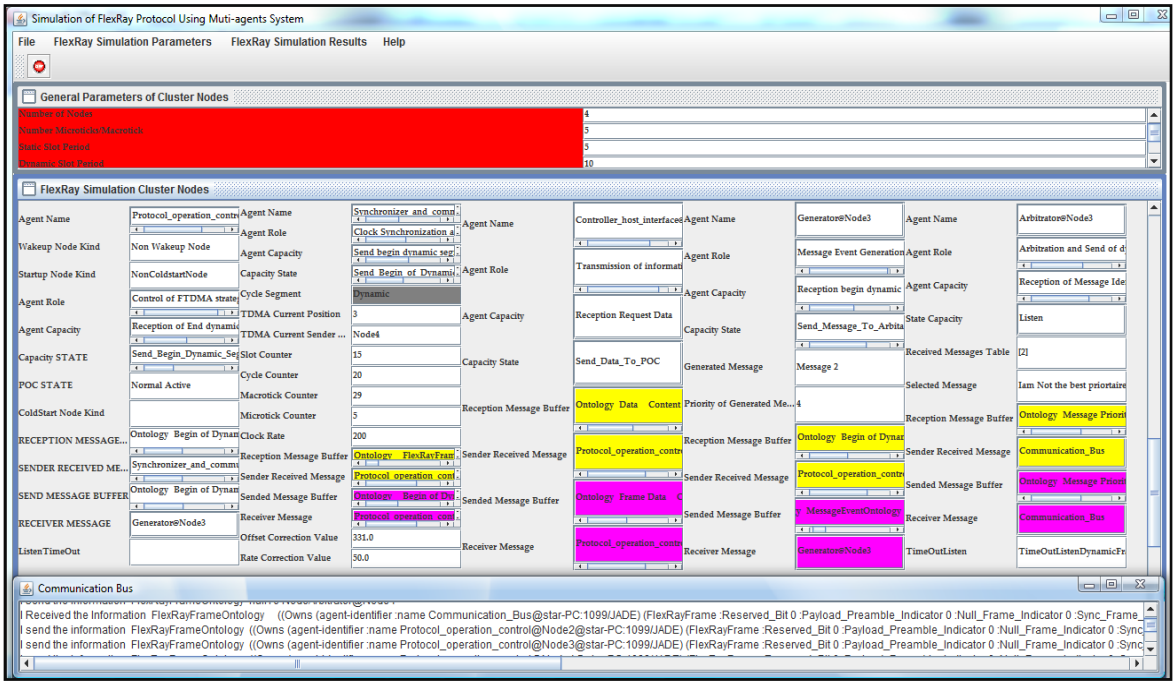


Fig. VI-2 : L'environnement développé

## 3. Etude de cas et analyse des résultats de simulation du protocole de communication FlexRay

### 3.1. Paramètres

Les paramètres de notre simulation sont les suivants :

- Un cluster qui consiste en 4 nœuds, deux nœuds coldstart (Nœud 1 et Nœud 2) et deux nœuds non coldstart (Nœud 3 et Nœud 4).
- Le Nœud 1 est un Nœud Wakeup.
- Les rythmes d'horloges des nœuds sont :

  Nœud 1 : 230, c'est-à-dire que l'horloge génère à chaque 230 ms un microtick.

  Nœud 2 : 200, c'est-à-dire que l'horloge génère à chaque 200 ms un microtick.

  Nœud 3 : 250, c'est-à-dire que l'horloge génère à chaque 250 ms un microtick.

  Nœud 4 : 270, c'est-à-dire que l'horloge génère à chaque 270 ms un microtick.
- Le nombre de microtick par macrotick est 5.
- Le nombre de macrotick par Slot est 5.
- La période de segment statique est 20 macrotick.
- La période de segment dynamique est 10 macrotick.
- La période de la fenêtre Symbol est 3 macrotick.
- La période de NIT statique est 5 macrotick.
- A chaque nœud dans la séquence : Nœud 1, Nœud 2, Nœud 3, Nœud 4 est attribué un seul slot.
- Les messages orientés événement et leurs priorités : (Message 1, 2), (Message 2, 4), (Message 3, 1).

## 3.2. Le Service de synchronisation d'horloge

Cette partie présente le résultat de simulation de l'algorithme de synchronisation d'horloge du protocole FlexRay décrit dans le chapitre II (partie II) sous l'environnement développé. L'algorithme consiste à calculer les valeurs de correction d'offset et de rate.

### 3.2.1. Avant la synchronisation

La Fig. VI-3 montre les agents «Synchronizer and communcation controller» des quatre nœuds à un instant donné. On remarque que les quatre agents n'ont pas les mêmes valeurs des compteurs de cycle, macrotick et microtick, car la synchronisation d'horloges n'aura pas encore lieu.

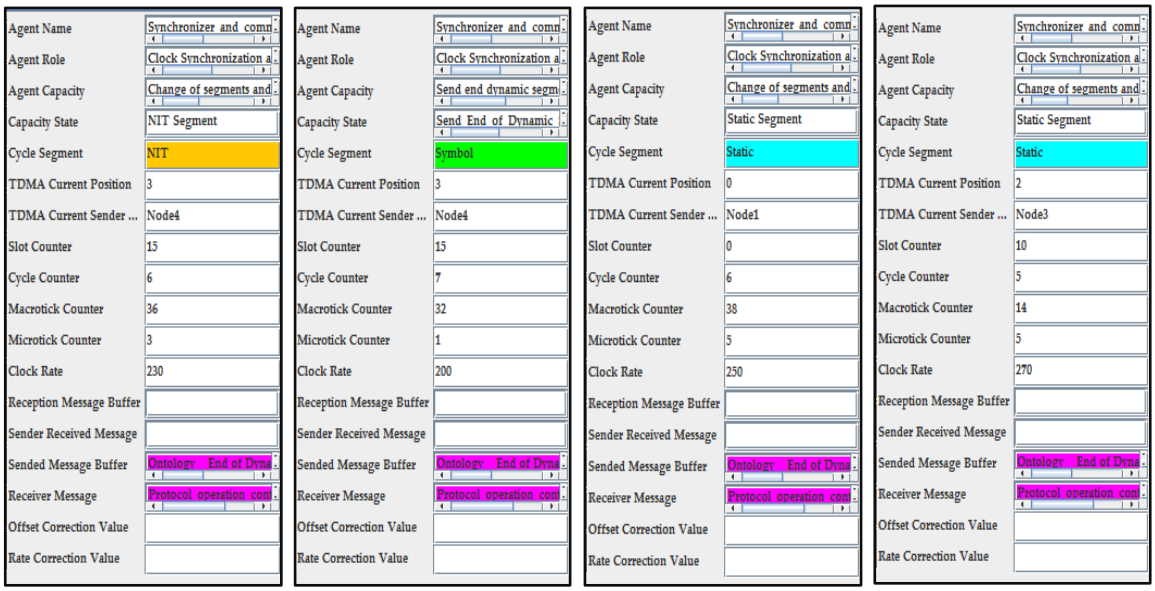


Fig. VI-3 : Les agents «Synchronizer and communcation controller» des quatre nœuds avant la synchronisation d'horloges

### 3.2.2. Après la Synchronisation

La Fig. VI-4 montre les quatre agents après la correction d'offset et de rate. On remarque que les quatre agents ont les mêmes valeurs de cycle, des valeurs proches de macrotick. Ce qui implique que tous les nœuds se trouvent dans le même segment et dans la même position de la table TDMA.


Fig. VI-4 : Les agents «Synchronizer and communcation controller» des quatre nœuds après la synchronisation d'horloges

### 3.3. Le Service Wakeup

Fig. VI-5 montre les différents états du nœud 1 (wakeup), ce nœud est responsable de réveiller les autres nœuds du cluster comme il est décrit dans le chapitre 2 (partie II). Dans le cycle 1, le nœud 1 est dans l'état «Wakeup State» ; lorsque le temps d'écoute est expiré sans aucune détection d'une communication sur le bus il passe à l'état «Wakeup Send» (dans le cycle 3) ; il envoie un wakeup pattern aux autres nœuds du cluster et il passe à l'état «Wakeup Second Listen». Après l'expiration de temps d'écoute sans aucune détection d'une collision sur le bus il passe à l'état «Wakeup Success» (dans le cycle 4). Lorsque les autres nœuds reçoivent le signal wakeup pattern ils entrent dans l'état Config.

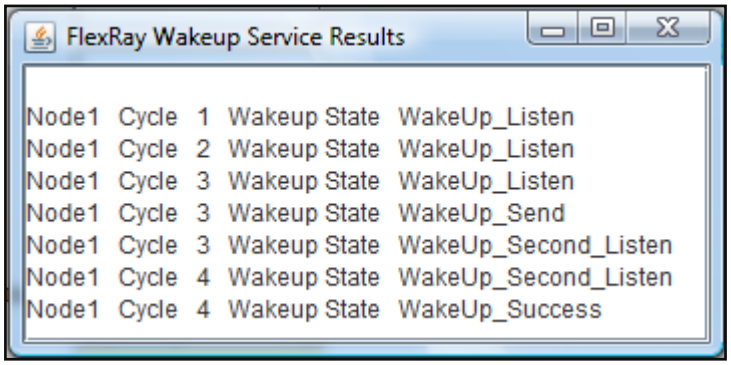


Fig. VI-5 : Résultat de l'exécution de service de Wakeup du protocole FlexRay

Fig. VI-6 montre l'agent «Protocol operation control» du nœud 1 dans l'état «Wakeup Send», l'agent «Bus de communication» lorsqu'il reçoit le wakeup pattern du nœud 1 et l'agent «Protocol operation control» du nœud 3 lors de sa réception de ce wakeup pattern auprès de l'agent «Communication bus».

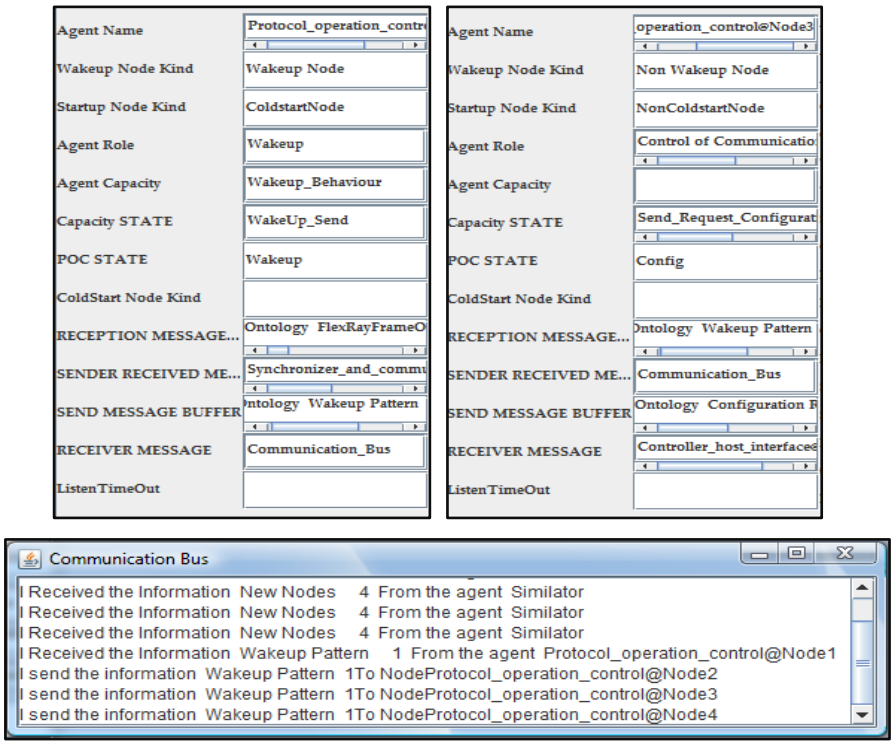


Fig. VI-6 : Les agents «Protocol operation control» du nœud 1, «Protocol operation control» du nœud 3, et «Communication bus» lors de l'émission d'un wakeup pattern

### 3.4. Le Service d'initialisation (Startup)

Fig. VI-7 montre les agents «Protocol operation control» des quatre nœuds à un instant donné. On remarque que les quatre nœuds sont en étape de startup. Les nœuds 1, 3, et 4 sont dans l'état «Initialize Schedule Wait First Coldstart Frame», tandis que le nœud 2 est dans l'état «Collision Resolution». Puisque le nœud 2 ne recevait aucun signal CAS ou trame coldstart après le temps d'écoute il passe à l'état «Collision Resolution». Les autres nœuds recevaient un signal CAS du nœud 2 c'est pour cela ils passent à l'état «Initialize Schedule Wait First Coldstart Frame».

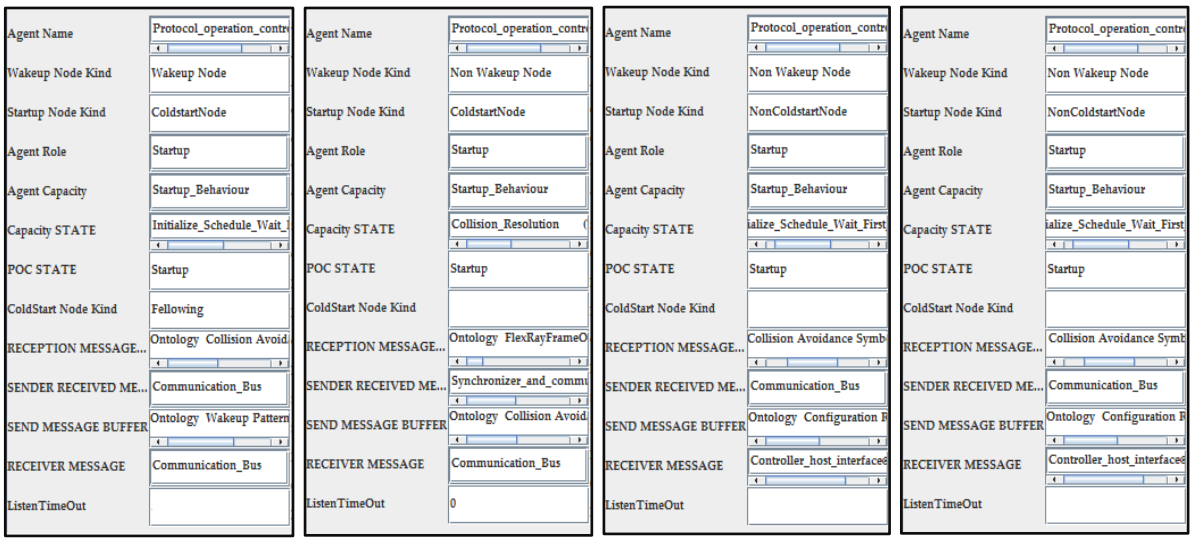


Fig. IV-7 : Les agents «Protocol operation control» des quatre nœuds du cluster

Fig. VI-8 montre l'historique de changements d'états des quatre nœuds durant la phase de startup. Au début, les quatre nœuds se trouvent dans l'état «Listen». Le nœud 2 passe à l'état «Resolution Collision» si son temps d'écoute est expiré sans aucune détection d'un mouvement dans le bus de communication. Dans cet état, le nœud 2 envoie aux autres nœuds un signal CAS suivi par quatre trames coldstart. Lorsque les nœuds 1, 3 et 4 reçoivent le signal CAS (le nœud 1 constate qu'il est le nœud following) ils passent à l'état «Initialize Schedule Wait First Coldstart Frame», et quand ils reçoivent la première trame coldstart du nœud 2 ils exécutent l'algorithme du calcul des valeurs de correction d'offset et passent à l'état «Initialize Schedule Wait Second Coldstart Frame». Lorsqu'ils reçoivent la deuxième trame coldstart du nœud 2 ils exécutent l'algorithme du calcul des valeurs de correction de rate et ils passent à l'état «Integration Check Wait First Coldstart Frame». Quand ils reçoivent la troisième trame du nœud 2 ils passent à l'état «Integration Check Wait Second Coldstart Frame». Si le nœud 1 (coldstart) reçoit la quatrième trame coldstart du nœud 2 il vérifie le temps de réception de la trame ; il trouve que ce temps correspond au temps prévisionnel (alors la correction d'offset et de rate a été effectuée avec succès) et il passe à l'état «Join». Quand les nœuds 3 et 4 reçoivent la quatrième trame du nœud 2 ils vérifient le temps de réception de la trame ; ils trouvent que ce temps correspond au temps prévisionnel (alors la correction d'offset et de rate a été effectuée avec succès) et ils passent à l'état «Integration Check Second Part Non Coldstart Nodes». Le nœud 2 termine l'état «Resolution Collision» sans aucune détection d'une collision dans le bus alors il constate qu'il est le leader et il passe à l'état «Consistency Check Wait First Coldstart Frame». Le nœud 1 qui se trouve dans l'état « Join » envoie une première trame colstart sur le bus. Comme le nœud 2 reçoive cette trame il passe à l'état «Consistency Check Wait Second Coldstart Frame» et il envoie une trame coldstart dans le même cycle. Si le nœud 2 reçoit une deuxième trame du nœud 1 dans le cycle suivant, alors il vérifie le temps de réception de la trame ; il trouve que ce temps est le même que le temps prévisionnel ; en conséquence il constate que son temps d'horloge est le même que celui du nœud 1. Il envoie une trame coldstart dans le cycle suivant et il passe à l'état «Startup Success». Si le nœud 1 reçoit cette trame il passe à l'état «Startup Success». Les nœuds 3 et 4 attentent la réception de huit (8) trames coldstart des nœuds 1 et 2 (pour confirmer que les deux nœuds coldstart sont initialisés) et ils passent à l'état «Startup Success».

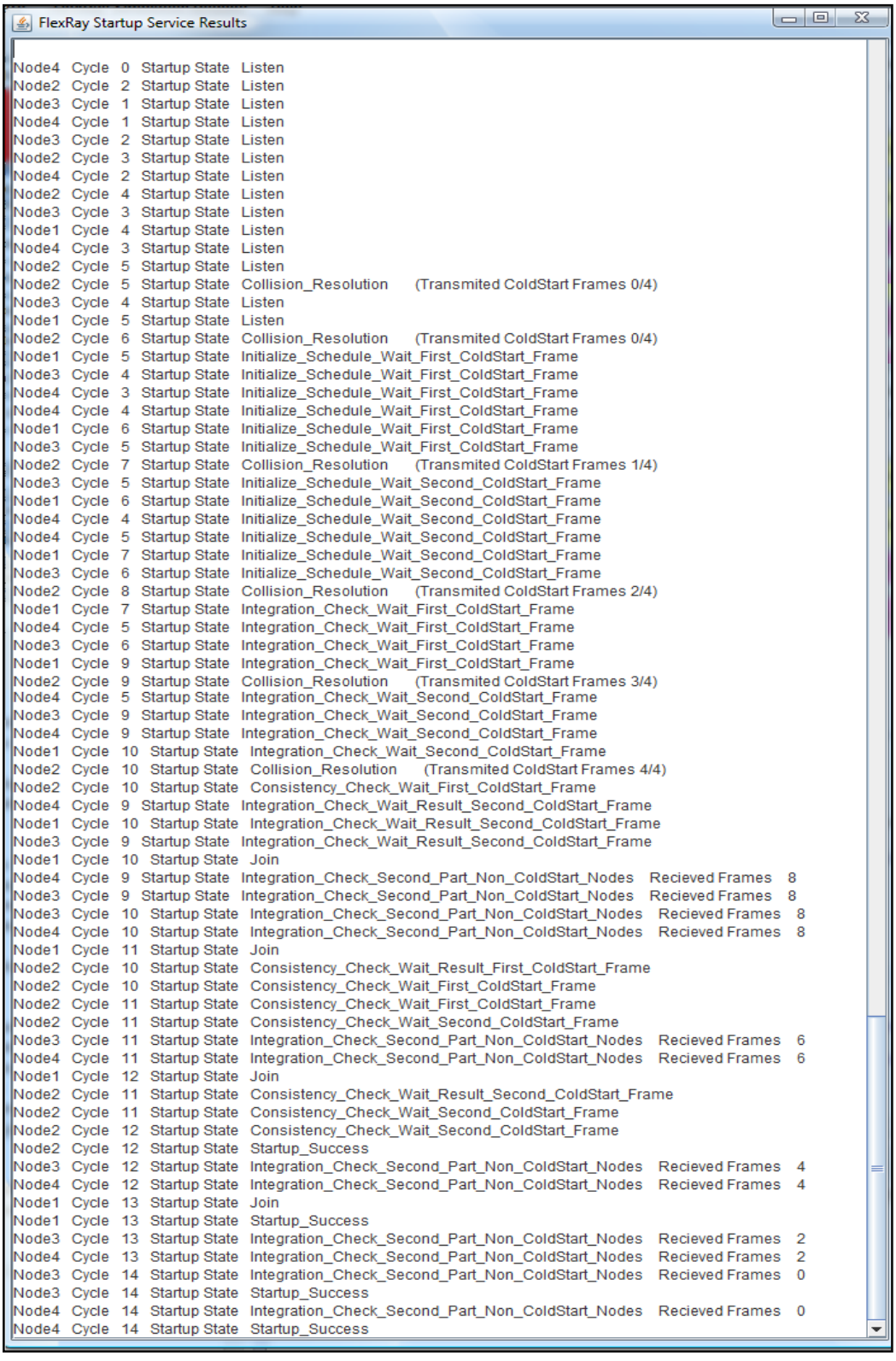


Fig. VI-8 : Résultat de l'exécution du service de startup du protocole FlexRay

### 3.5. Le Service d'accès au support de communication selon la stratégie TDMA

Fig. VI-9 montre l'historique d'accès au bus de communication des quatre nœuds pendant le segment statique selon la stratégie TDMA. Dans chaque cycle, le slot 0 est réservé pour le nœud 1, le slot 1 pour le nœud 2, le slot 2 pour le nœud 3 et le slot 3 pour le nœud 4. Cette stratégie permet d'éviter le problème de la collision.

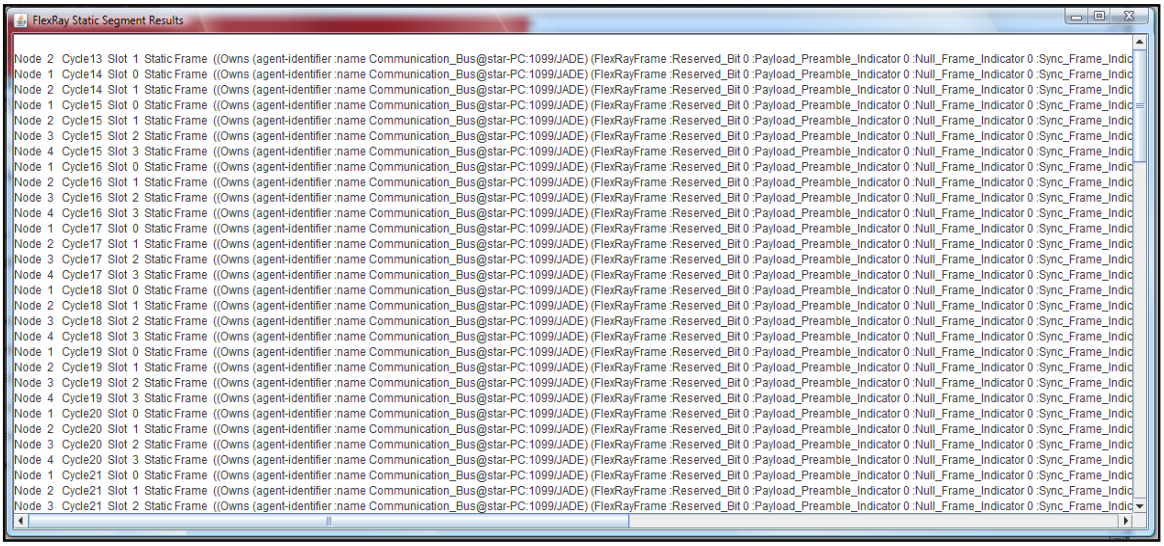


Fig. VI-9 : Résultat du service d'accès au bus de communication selon la stratégie TDMA du protocole FlexRay

Fig. VI-10 montre une capture d'écran présentant l'émission d'une trame statique auprès du nœud 3 durant le segment statique. Elle illustre les trois agents («Protocol operation control», «Synchronizer and communcation controller» et «Controller host interface»). L'agent «Synchronizer and communcation controller» envoie une trame FlexRay à l'agent «Protocol operation control» qui contient des informations de synchronisation (la position de la table TDMA se pointe sur le slot 2 qui a été réservé pour le nœud 3). Lorsque l'agent «Protocol opération control» reçoit ce message il envoie une demande de données à l'agent «Controller host interface». Fig. VI-11 présente le comportement de l'agent «Protocol operation control» : quand il reçoit les données auprès de l'agent «Controller host interface», il envoie sa trame à l'agent «Communication bus». Fig. VI-12 montre le comportement de l'agent «Communication bus» : quand il reçoit la trame du nœud 3 il envoie cette trame aux autres nœuds du cluster.

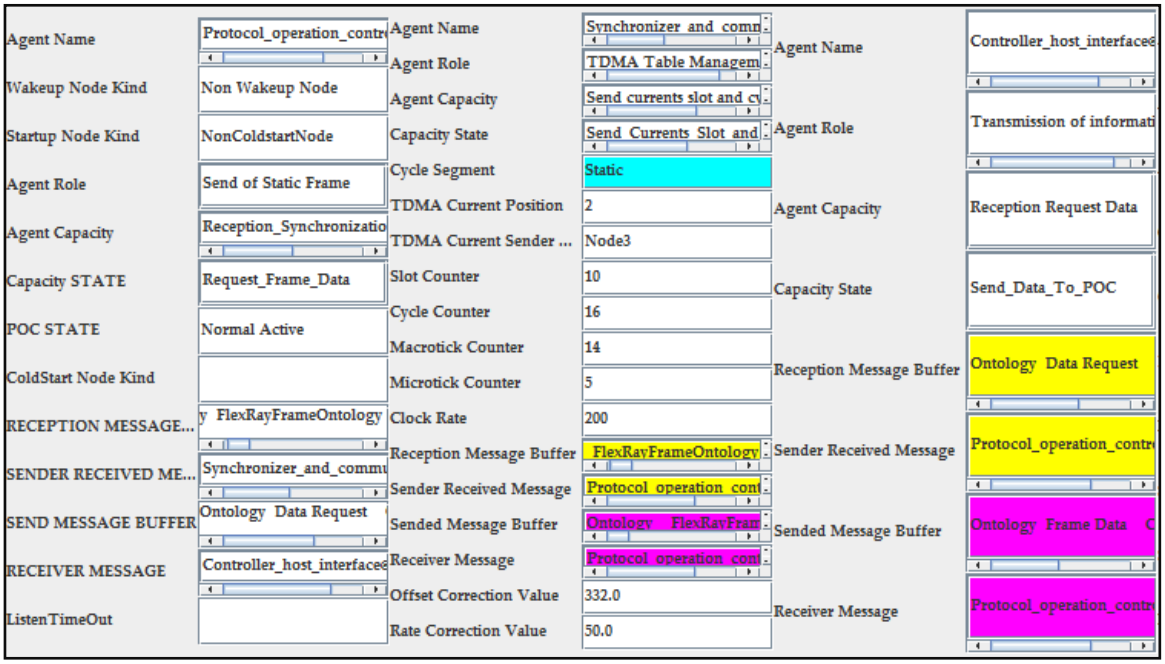


Fig. VI-10 : Les agents «Protocol operation control» «Synchronizer and communcation controller» «Controller host interface» du nœud 3 lors de l'émission d'une trame statique

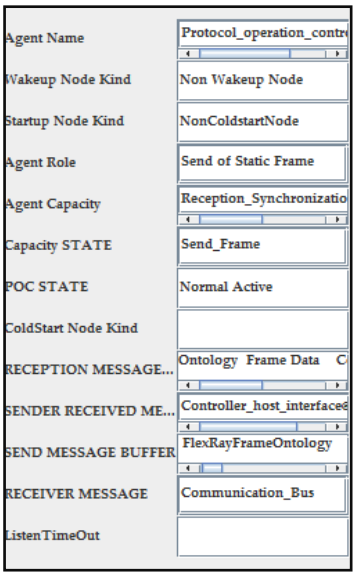


Fig. VI-11 : l'agent «Protocol operation control» du nœud 3 lors de la réception des données de l'agent «Controller host interface»

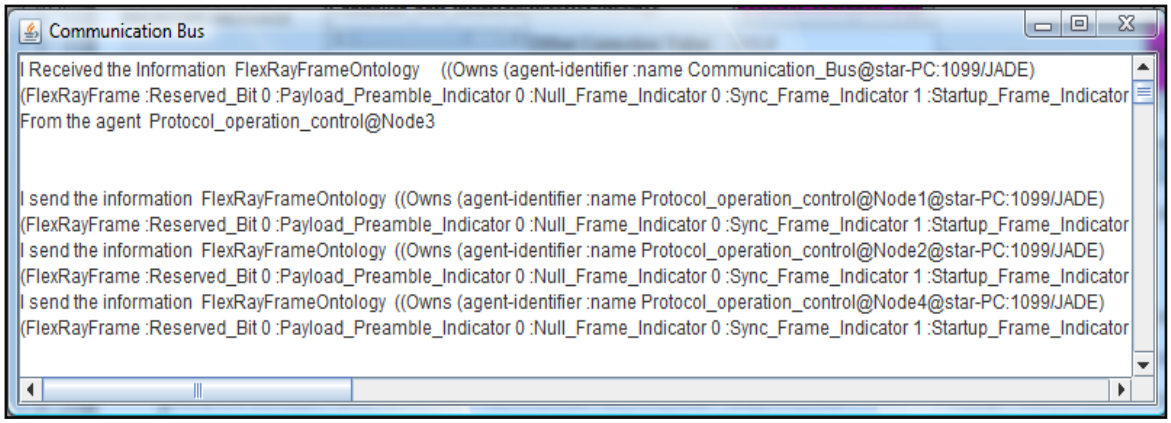


Fig. VI-12 : L'agent « Communication bus» lors de la réception d'une trame du nœud 3

### 3.6. Le Service d'accès au support de communication selon la stratégie FTDMA

Fig. VI-13 montre l'accès au bus de communication à base de la stratégie FTDMA durant le segment dynamique du protocole FlexRay. Dans le cycle 14, le nœud 3 génère un message événement avec une priorité 1 et en même temps le nœud 4 génère un message avec une priorité 2. Le message le plus prioritaire est ce du nœud 3 puisque sa priorité est la moins élevée alors il envoie une trame dynamique à l'agent «Communication bus».

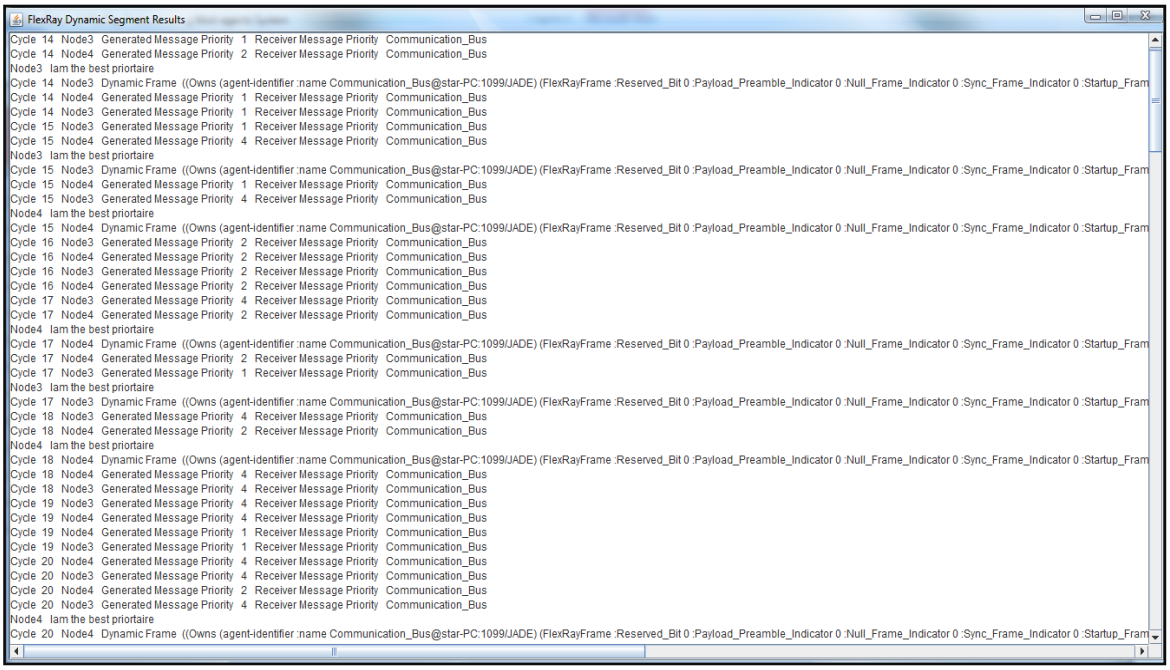


Fig. VI-13 : Résultat du service d'accès au bus de communication selon la stratégie FTDMA du protocole FlexRay

Fig. VI-14 montre l'agent «Generator» du nœud 3 lors de la génération aléatoire d'un message durant le segment dynamique. Il envoie ce message généré à l'agent « Arbitrator».

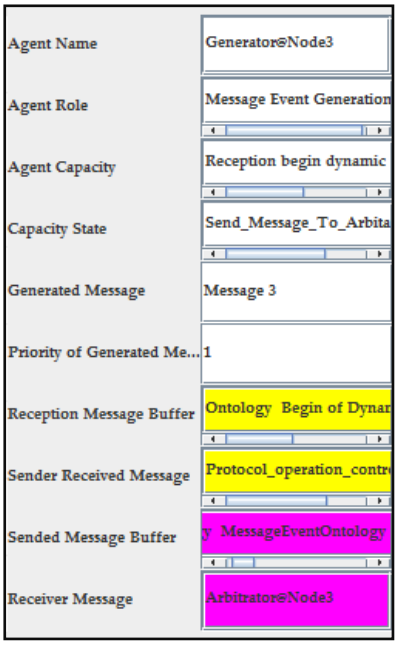


Fig. VI-14 : L'agent «Generator» du nœud 3 lors de la génération aléatoire d'un message

Fig. VI-15 montre les états de l'agent « Arbitrator» du nœud 3. En état 1, il reçoit le message généré auprès de générateur de messages puis il envoie la priorité de ce message à l'agent «Communication bus». Dans l'état 2 il reçoit la priorité de message généré par le nœud 4 ; il constate que son message à transmettre est le plus prioritaire. Dans l'état 3 il envoie sa trame dynamique à l'agent «Communication bus».


Fig. VI-15 : L'agent «Arbitrator» du nœud 3 lors de l'arbitrage

## 4. Discussion de résultats de simulation

Nous avons utilisé la plateforme jade pour simuler les différents services offerts par le protocole de communication FlexRay. L'utilisation de la simulation multi-agent facilite l'implémentation du protocole FlexRay qui est un protocole très complexe à cause de l'exécution simultanée intégrant plusieurs services comme les services d'initialisation et de synchronisation d'horloges. En plus cette simulation à base d'agents offre l'avantage de paralléliser le fonctionnement des agents qui facilite la modélisation du protocole FlexRay, de tel sorte que les agents des nœuds peuvent fonctionner simultanément et aussi les agents du même nœud peuvent exécuter des services en parallèle. La propriété de la distribution du système multi-agent permet de simuler notre cluster qui représente un réseau distribué. Aussi le passage de messages entre les agents permet de simuler le passage de trames entre les nœuds du cluster. Les résultats obtenus sont encourageants : nous pouvons par exemple trouver une correction d'offset et de rate optimale dans la phase d'initialisation du cluster (startup) dans la première tentative car l'utilisation des systèmes multi-agent permet d'exécuter l'algorithme de synchronisation parfaitement parce que chaque agent exécute cet algorithme d'une façon autonome et indépendante. Nous pouvons bien comprendre le comportement des nœuds du cluster FlexRay lors des phases de wakeup, startup, l'émission d'une trame statique durant le segment statique, l'émission d'une trame dynamique durant le segment dynamique, la réception d'une trame…

# Conclusion Générale & Perspectives

# Conclusion générale et perspectives

La simulation des protocoles de communication au sein des systèmes embarqués de l'automobile est un problème complexe à cause de l'intégrité, la distribution et le parallélisme de leurs services.

Nous avons présenté dans ce travail une proposition d'une approche de modélisation et de simulation basées agents des protocoles de communication TTP et FlexRay au sein du système embarqué de l'automobile.

La méthodologie O-MaSE choisie est une méthodologie multi-agent organisationnelle qui sépare les agents par rapport aux rôles qu'ils jouent, ce qui autorise une grande flexibilité de conception. La correspondance de concepts entre la méthodologie O-MaSE et la plateforme Jade facilité la translation de notre modèle conceptuel en une implémentation Jade.

L'implémentation du protocole FlexRay prouve notre choix de la simulation multi-agent de tel sort que les services peuvent s'exécuter en parallèle d'une façon autonome et distribuée et donne des bons résultats en ce qui concerne la synchronisation d'horloges des nœuds et offre un support pour comprendre le fonctionnement de ce protocole de communication comme les mécanismes de wakeup, startup, synchronisation d'horloges communication selon les stratégies TDMA, et FTDMA...

Comme perspectives, nous planifions :

1. Etendre notre approche de modélisation et simulation afin d'élargir les paramètres de simulation comme le nombre de macroticks par cycle, le nombre de microticks par macrotick, les rythmes d'horloges …
2. Effectuer des simulations sur les services du protocole TTP.
3. Appliquer cette approche de la simulation sur la partie application des réseaux embarqués de l'automobile.
4. Vérifier formellement le modèle conceptuel proposé.

# Bibliographie

# Bibliographie


[1] F. Simonot-Lion et N. Navet. Les réseaux temps réel embarqués dans les véhicules. Systèmes temps réel 2-ordonnancement, réseaux et qualité de service, 2006.

[2] K. Dahmen. Conception d'un générateur d'intergiciels temps réel embarqués dans l'automobile. Mémoire d'ingénieur en informatique de l'université d'ISI, Tunisie, 2007.

[3] Y. Khadidja. L'apport des outils de l'intelligence artificielle dans les systèmes temps réel : ordonnancement des taches. Thèse de Doctorat d'état de l'université d'Oran, Mai 2013.

[4] A.S. Tanenbaum, et M. v. Steen. Distributed systems – principles and paradigms. Second edition, prentice hall, 2007.

[5] B. Ducourthial. Introduction aux systèmes répartis. v0.9-POLY, 2007.

[6] M. Bachir. Tolérance aux fautes des systèmes temps-réel embarqués basée sur la redondance. Mémoire de Magistère de l'université de Batna, 2011.

[7] X. Chen. Requirements and concepts for future automotive electronic architectures from the view of integrated safety.Thèse de PhD de l'université de Karlsruhe, 2008.

[8] A. Monot. Vérification des contraintes temporelles de bout-en-bout dans le contexte Autosar. Thèse de Doctorat de l'université de Lorraine, Novembre, 2012.

[9] J.-P. Elloy. L'optimisation des systèmes distribués temps réel embarquées: application au prototypage rapide d'un véhicule électrique autonome. Thèse de Doctorat en sciences de l'université de Rouen U.F.R de Sciences, 2000.

[10] B. Hicham. Spécification formelle de protocole CAN. Mémoire de Master de l'université d'Oum El Bouaghi, 2011.

[11] T. Pop. Scheduling and optimization of heterogeneous Time/Event-Triggered distributed embedded systems. Mémoire de License de l'université de Linköping, N° d'ordre 1022, 2003.

[12] D. Keating, A. McInnes, et M. Hayes. Model checking a TTCAN implementation.In software testing, verification and validation (ICST), 2011 IEEE fourth international conference, p. 387–396, 2011.

[13] P. Böhm. Introduction to FlexRay and TTA.November, 2005.

[14] M. O. Carrion. Communications sur le réseau d'énergie électrique d'un véhicule : modélisation et analyse du canal de propagation. Thèse de Doctorat d'état de L'université des sciences et technologies de Lille, N° d'ordre 3818, Juillet, 2006.

[15] H. Kopetz et G. Grunsteidl. TTP-a protocol for fault-tolerant real-time systems.Computer, vol. 27, n° 1, p. 14–23, 1994.

[16] K.Godary. Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile. Thèse d'état présentée devant l'institut national des sciences appliquées de Lyon, France, Novembre, 2004.

[17] D. Bradbury. Simulation of a time triggered protocol. Basser department of computer science, university of Sydney, Australia, 2000.

[18] N. Navet. Le réseau TTP (Time Triggered Protocol). INRIA Lorraine, projet TRIO, 2004.

[19] W. Steiner et H. Kopetz. The startup problem in fault-tolerant time-triggered communication.In dependable systems and networks, DSN 2006, international conference, p. 35–44, 2006.

[20] H. Kopetz. A comparison of CAN and TTP. Annual reviews in Control, vol. 24, p. 177–188, 2000.

[21] A. Zhao. Reliable in-vehicle FlexRay network scheduler design. Master of science thesis in electrical engineering, p. 17–23, 2011.
[22] Flexray Consortium. FlexRay communications system protocol specification version 2.1. 2005.
[23] P. Göhner. Introduction to the FlexRay bus system. December, 2012.
[24] R.Shaw. Improving the reliability and performance of FlexRay vehicle network applications using simulation techniques.Waterford institute of technology, Mai, 2009.
[25] R. Rieb. FlexRay.Seminar, Février, 2009.
[26] B. Schatz, C. kuhnel, M. Gonschorek. FlexRay Protocol. In automotive embedded systems handbook (Ed. N. Navet and F. Simonot-Lion), CRC Press/Taylor and Francis, December 2008.
[27] S. Kosov. FlexRay communication protocol (Wakeup and Startup). Institute for computer architecture and parallel computing, universitat des saarlandes, 2005.
[28] J. Malinsky. The application of the timed automata for FlexRay start-up testing. 2008.
[29] P. A. Fishwick. Computer simulation: growth through extension. Transactions of the society for computer simulation, vol. 14, n° 1, p. 13–24, 1997.
[30] R. E. Shannon. Introduction to the art and science of simulation.In simulation conference proceedings, vol. 1, p. 7–14, 1998.
[31] B. Lazhar. Modélisation et simulation basées multi-agents du contrôle de processus industriels. Mémoire de Magister de l'université de Skikda, 2009.
[32] D. David. Prospective territoriale par simulation orientée agent. L'université de la Réunion, 2010.
[33] A. Hanzlik. SIDERA-a simulation model for time-triggered distributed real-Time systems.International review on computers and software (IRECOS), vol. 1, n° 3, p. 181–193, 2006.
[34] M. Khanapurkar, J. Y. Hande, et P. Bajaj. Approach for VHDL and FPGA implementation of communication controller of FlexRay controller.Journal of information hiding and multimedia signal processing. Volume 1, number 4,October,2010.
[35] W. S. Kim, H. A. Kim, J.-H. Ahn, et B. Moon. System-level development and verification of the FlexRay communication controller model based on SystemC.In future generation communication and networking.FGCN'08, second international conference, vol. 2, p. 124–127, 2008.
[36] A. Hanzlik. A case study of clock synchronization in flexray. 2006.
[37] V. R. K. R. Podduturi. A SystemC simulator for the dynamic segment of the FlexRay protocol.Linköping, 2012.
[38] S. Buschmann, T. Steinbach, F. Korf, et T. C. Schmidt. Simulation based timing analysis of FlexRay communication at system level. 2013.
[39] F. Ren, Y. R. Zheng, et J. Sarangapani. FlexRay network utilization evaluations based on static and dynamic segments.Proceedings of the 3rd annual ISC research symposium. April, 2009.
[40] J. Malinskỳ et J. Novák. Verification of flexray start-up mechanism by timed automata. 2010.
[41] N. R. Jennings, et M. Wooldridge. Applications of intelligent agents. Springer, 1998.
[42] J. Ferber. Les systèmes multi-agents: vers une intelligence collective (Vol. 16). Paris: InterEditions, 1995.

[43] T. Jarraya. Réutilisation des protocoles d'interaction et démarche orientée modèles pour le développement multi-agents. Thèse de doctorat d'état de l'université de Reims Champagne Ardenne, 2006.
[44] V. M. B. Werneck, R. M. E. M. Costa, et L. M. Cysneiros. Modelling multi-agent system using different methodologies, multi-agent system: modeling, interactions, simulations and case studies. Rijeka: Intech, p. 77-96, 2011.
[45] A. SABAS. Systèmes multi-agents: une analyse comparative des méthodologies de développement. Mémoire de Maîtrise, 2001.
[46] Q. Zhou. A tutorial on agent based software engineering. 2002.
[47] S. A. DeLoach. Engineering organization-based multiagent systems. In software engineering for multi-agent systems IV, Springer, p. 109–125, 2006.
[48] J. C. Garcia-Ojeda, S. A. DeLoach, W. H. Oyenan, et J. Valenzuela. O-MaSE: a customizable approach to developing multiagent development processes. Springer, 2008.
[49] S. A. DeLoachet J. C. Garcia-Ojeda. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. International journal of agent-oriented software engineering, vol. 4, n° 3, p. 244–280, 2010.
[50] S. A. DeLoach. Developing a multiagent conference management system using the O-MaSE process framework. In agent-oriented software engineering VIII, Springer, p. 168–181, 2008.
[51] F. Mguis, K. Zidi, K. Ghedira, et P. Borne. Modélisation d'un système multi-agent pour la résolution d'un problème de tournées de véhicule dans une situation d'urgence. In 9th international conference on modeling, optimization &simulation, 2012.
[52] M. F. Feki. Optimisation distribuée pour la recherche des itinéraires multi-opérateurs dans un réseau de transport co-modal. Thèse de Doctorat de l'école centrale de Lille, 2010.
[53] N. M. Hung. Génie logiciel orienté agent. 2006.
[54] H. Joumaa. Analyse des performances d'un système multi-agents par visualisation. Thèse de Doctorat de l'université Joseph-Fourier-Grenoble I, 2010.
[55] M. F. Wood et S. A. DeLoach. An overview of the multiagent systems engineering methodology.In agent-oriented software engineering, p. 207–221, 2001.
[56] B. Bauer, J. P. Müller, et J. Odell. Agent UML: A formalism for specifying multiagent software systems. International journal of software engineering and knowledge engineering, vol. 11, n° 03, p. 207–230, 2001.
[57] F. Bellifemine, G. Caire, T. Trucco, et G. Rimassa. JADE programmer's guide. JADE 2.44, September, 2001.
[58] N. Taghezout. Conception et développement d'un système multi-agent d'aide à la décision pour la gestion de production dynamique. Thèse de Doctorat de l'université de Toulouse III-Paul Sabatier, 2011.
[59] S. Azaiez. Approche dirigée par les modèles pour le développement de systèmes multi-agents. Thèse de Doctorat en informatique de l'université de Savoie, Décembre, 2007.
[60]Plate-forme JADE. [En ligne]. Disponible sur: http://perso.limsi.fr/jps/enseignement/examsma/2005/1.plateformes_3/index-Ferguen.html. [Consulté le: 08-nov-2013].

# Liste des Acronymes

ABS: Anti-lock Braking System

ACC: Agent Communication Channel

ACL: Agent Communication Language

AMS: Agent Management System

AIP: Agent Interaction Protocol

ASC: Automatic Stability Control

AUML: Agent UnifiedModeling Language

CAN: Controller Area Network

CAS: Collision Advances Symbol

CHI: Controller Host Interface

CLT: Coolant Temperature

CNI: Controller Network Interface

CSP: Clock Synchronization Process

DF: Director Facilitor

ECU: Elctronic Control Unit

ECM: Engine Control Module

ESP: Electronic Stability Program

ET: Event Triggered

FCU: Fault Containment Unit

FIPA: Foundation for intelligent Physique Agents

FPGA: Field Programmable Gate Arrays

FTCNI: Fault Tolerant Controller Network Interface

FTDMA: Flexible Time Division Multiple Access

FTM: Fault-Tolerant Midpoint

4WD: Four Wheel Drive

ISO: International Organization for Standards

JADE: Java Agent DEvelopement framework

LLI: Logical Line Interface

MAP: Manifold Absolute Pressure

MAS-CommonKADS: MultiAgent System-Knowledge Analysis and Development System

MaSE: Multi-agents System Engineering

MEDL: Message Descriptor List

MIC: Minimum Integration Count

MTG: Macrotick Generation Process

O-MaSE: Organizationbased Multi-agents System Engineering

OMG: Object Management Group

OMT: Object Modeling Technique

POC: Protocol Operation Control

SAE: Society of Automotive Engineers

SDL: Specification and DescriptionLanguage

SIDERA: SImulation model for DependableRealtime Architectures

SMA: Système Multi-Agent

SRU: Smallest Replaceable Unite

TADL: TAsk Descriptor List

TDMA: Time Division Multiple Access

TILAB: Telecom Italia Lab

TTA: Time Triggered Architecture

TT-CAN: Time-Triggered CAN

TTP: Time-Triggered Protocol

TT: Time Triggered

UML: Unified Modeling Language

VAN: Vehicle Area Network

VHDL: VHSIC Hardware Description Language