UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR

ECOLE DOCTORALE DE MATHEMATIQUES ET INFORMATIQUE

FACULTE DES SCIENCES ET TECHNIQUES

Année : 2017

N° d'ordre : 108

THESE DE DOCTORAT

Mention : INFORMATIQUE ET TELECOMMUNICATIONS

Spécialité : INFORMATIQUE

Présentée par :

Fatou FAYE

Titre : Modèles et Protocoles de coordination pour la reconfiguration dynamique des architectures logicielles embarquées

Soutenue le 09 Décembre 2017 devant le jury composé de :

Président : M. Hamidou DATHE	Professeur	UCAD, Dakar Sénégal
Rapporteurs :		
M. Samir AKNINE	Professeur	Université Lyon1, France
M. Mohamed Tahar KIMOUR	Professeur	Université d'Annaba, Algérie
Examinateurs :		
M. Mohamed Tayeb LASKRI	Professeur	Université d'Annaba, Algérie
M. Abdourahmane RAIMY	Maître de conférences	UCAD, Dakar Sénégal
M. Oumar DIANKA	Professeur	UCAD, Dakar Sénégal
Directeur de thèse :		
M. Mbaye SENE	Maître de conférences	UCAD, Dakar Sénégal

Dédicaces

A mes parents pour leur soutien incontestable,

A mon mari, mes enfants pour leur amour incommensurable,

A mes Frères et sœur pour leur conseil fraternel,

A mes beaux parents pour leur présence chaleureuse,

A ma meilleure amie et sœur Mame Anta FAYE paix à son âme,

A tous les membres des familles FAYE, DIOUF

DIA, DIONE, SOW, DIALLO

Et à tous ceux qui me sont chers

Remerciements

C’est avec un immense plaisir que j’écris cette section de ma réalisation qui est le fruit d’un travail de longue haleine avec la collaboration et la complicité de mon Directeur de thèse Dr Mbaye SENE.

J’en profite pour remercier Dr Abdou Karim Konaté notre Chef de Section Informatique de la FST de l’UCAD et tout le personnel.

Je remercie également Mr Ahmadou Bamba Mbacké et Mme Keita du Département Génie Informatique de l’ESP, Dr Amadou Lamine Ndiaye du Département Génie Electrique de l’ESP, Dr Komlan Beda Directeur d’ESITEC du Groupe SUPDECO pour leurs orientations, conseils et travaux scientifiques.

Ensuite, je tiens à remercier Dr Ousmane Diallo de l’UASZ, Dr Mahmoud Ould DEYE de la Section Informatique pour la relecture de ce document.

Nous remercions vivement les professeurs Mohamed Tahar KIMOUR et Samir AKNINE pour avoir accepté d’être les rapporteurs de ce travail et les professeurs Abdourahmane RAIMY, Mohamed Tayeb LASKRY et Oumar DIANKA d’être examinateurs de ce travail.

Résumé

Aujourd'hui, le monde ne cesse d'être de plus en plus informatisé à une vitesse exponentielle. Cela exige l'accomplissement de tâches complexes avec des exigences en performance en termes de délai et de fiabilité. En effet, les machines utilisées sont dotées de systèmes embarqués composés de matériels et logiciels. Ces systèmes subissent des modifications architecturales en temps réel de leurs parties logicielles embarquées. Ainsi, nait la reconfiguration dynamique qui consiste à modifier l'architecture des logiciels sans entraver leur fonctionnement. Ces systèmes embarqués (enfouis) sur puce sont utilisés dans des domaines d'applications divers et variés (médecine, automobile, aéronautique, télécommunications, domotique, …). Ils sont particulièrement présents dans les applications à forte fréquence d'utilisation. Ces applications constituées d'ensemble d'algorithmes sont soumises aux exigences grandissantes de ses utilisateurs et aux contraintes de leur environnement d'exécution. En effet, leur environnement d'exécution s'est étendu avec leur connectivité à l'internet d'où le nom *Internet of Things* (IoT). Ces objets sont souvent soumis à des contraintes temporelles qui doivent être respectées. La complexité grandissante et les coûts élevés de conception de ces systèmes font aujourd'hui que les concepteurs se tournent de plus en plus vers la reconfiguration dynamique.

Dans le cadre de cette thèse, nos contributions sont orientées sur la conception de ces architectures logicielles embarquées :

- Un nouveau profil UML appelé M4DRES,
- Une nouvelle plateforme d'exécution de ces objets connectés nommée ESDRES,
- Un nouveau processus de développement qui intègre les deux points précédents et une étude de performance de ces systèmes.

Mots-clés: Modèles, protocoles, reconfiguration dynamique, architectures logicielles embarquées

Abstract

Nowadays, the world keeps on being more and more computerized in an exponential speed. It requires the fulfillment of complex tasks with requirements in performance in terms of deadlines and reliabilities. Indeed, the used machines are endowed with embedded systems consisted of equipments and softwares. The latter suffer from real time architectural modifications of their embedded software parts. So, the dynamic reconfiguration comes into existence, which consists in modifying the architecture of the software without hindering their functioning. These embedded systems (buried) in chip are used in diverse and varied domains of implementations (medicine, automobile, aeronautics, telecommunications, home automation,…). They are particularly present in the implementations with strong frequency of uses. These implementations composed of a set of algorithms which are submitted both to the growing requirements of users and the constraints of their execution environment. In fact, their use environment extended with their internet connectivity, hence the appellation of 'Internet of Things' (IoT). These objects are often submitted to temporal constraints which must be respected. The growing complexity coupled with the expensive costs are the results of the fact that conceptors are increasingly focusing on the dynamic reconfiguration.

In this thesis, ours contributions will be oriented on the conception of embedded software architectures:

- A new UML profile known as M4DRES,
- A new execution platform of the connected objects called ESDRES,
- A new development process linking the two previous parts and a performance study of these systems.

Keywords: Models, Protocols, dynamic reconfiguration, embedded software architectures

Table des matières

Liste des figures

Liste des tableaux

Liste des formules

Liste des abréviations

AADL	Architecture Analysis & Design Language
AI	Artificial Intelligence
ATL	Atlas Transform Language
CADML	Component Assembly and Deployment Modeling Language
CART	Classification and Regression Trees
CCPN	Complex Choice Petri Nets
CCM	Corba Component Model
CHAID	CHi-squared Automatic Interaction Detector
CIAO	Component Integrated ACE ORB
CORBA	Common Object Request Broker Architecture
Dr	Docteur
DMS	Deadline Monotonic Scheduling
EDF	Earliest Deadline First
EPI	Ecran Publicitaire Intelligent
ESDRES	Execution Support For Dynamically Reconfigurable Embedded Systems
ESP	Ecole Supérieure Polytechnique
ETAs	Automates Temporisés Etendus
FST	Faculté des Sciences et Techniques
FLARe	Fault-tolerant Load-aware and Adaptive middlewaRe
GSPN	General Stochastic Petri Nets
HLAM	HighLevel Application Modeling
IA	Intelligence Artificielle
IAMM	Internal Application Meta-model
IDM	Ingénierie Dirigée par les Modèles
ID3	Induction Decision Tree
IMM	Implementation Meta-Model
IoT	Internet of Things
IPMM	Internal Platform Meta-Model
LLF	Least Laxity First
LTAMM	Label Timed Automata Meta-model
LwCCM	Lightweight Corba Component Model
MARTE	Modeling and Analysis of Real Time and Embedded Systems
MDA	Model Driven Architecture

MDE	Model Driven Engineering
MIT	Massachusetts Institute of Technology
ModES	Model-driven design approach
ModSyn	Model driven Co-synthesis for embedded system
MOFM2T	Models to Text Transformation Language
MMM	Mapping Meta-Model
M2M	Model To Model
M2T	Model To Text
M4DRES	Modeling For Dynamically Reconfigurable embedded Systems
OCL	Object Constraints language
OCML	Options Configuration Modeling Language
OMG	Object Management Group
ORB	Object Request Brokers
PC	Personal Computer
PCP	Priority Ceilling Protocol
PIM	Platform Independent Models
PolyORB_HI	PolyORB-High Integrity
PSM	Platform Specific Models
QoS	Quality of Services
QVT	Query/View/transform
UASZ	Université Assane Seck de Ziguinchor
UCAD	Université Cheikh Anta Diop
UML	Unified Modeling Language
UML-RT	UML-Real Time
RdP	Réseau de Pétri
RMS	Rate Monotonic Scheduling
RTPN	Real-Time Petri Nets
RTOS	Real Time Operating System
RT-CORBA	Real Time CORBA
SAE	Society of Automotive Engineers
SGM	State Graph Manipulators
SPN	Réseaux de Pétri Stochastiques
SPT	UML Profile for Schedulability, Performance and Time
SoCP	System On a Chip Design
SRM	Software Resource Modeling

SWN	Réseaux de Pétri Stochastiques Bien Formés
SysML	Systems Modeling Language
TelcoML	Telecommunication Modeling Library
VERTAF	Verifiable Embedded Real-Time Application Framework

Introduction générale

De nos jours, les matériels utilisés qui sont voiture, avion, téléphone, PC, robots, automates, électroménagers) sont de plus en plus intelligents et sont constitués de circuits électroniques miniaturisés (nano-ordinateurs, microcontrôleurs). Ces matériels sont appelés objets connectés (IoT) en raison de leur interconnectivité. Ils comprennent :

- Une partie matérielle qui constitue le support physique, la partie visible et palpable,
- Une partie logicielle qui constitue le cerveau où sont implémentés les algorithmes.

La mise en œuvre de ces matériels passe par des méthodes d'analyse et de conception du système. Ces méthodes doivent respecter les exigences matérielles, les attentes des utilisateurs, leurs délais fixés ; d'où leur complexité.

Les algorithmes constituant la partie logicielle dictent toutes les actions exécutées par le support matériel. Aujourd'hui, ces matériels se trouvent face à la concurrence ardue du marché de vente. Ainsi, ils proposent plus de fonctionnalités répondant aux exigences des utilisateurs et aux variations de leur environnement d'exécution. Dans le monde technologique, ces matériels sont appelés systèmes embarqués.

Ces systèmes embarqués proposent deux types de reconfigurations possibles :

1. La reconfiguration architecturale est la modification matérielle du système (ajout ou suppression des connexions et/ ou des composants),
2. La reconfiguration comportementale est le changement de comportement du système. Ce changement s'effectue par la partie logicielle du système qui propose différentes modes de fonctionnement.

Ainsi, émerge le problème de mise à jour, voire de modification partielle ou complète des logiciels qui sont embarqués ou enfouis dans ces systèmes. De plus, ces systèmes embarqués ne peuvent pas souvent s'arrêter afin d'effectuer des mises à jour. En effet, l'arrêt et le redémarrage du système entrainent une perte d'état et de temps. Cette perte d'état du système peut entrainer des conséquences désastreuses, telles que des pertes humaines, matérielles, économiques. Ces systèmes sont soumis à des délais de réponse qui sont généralement rigoureux. Par conséquent, nous devrons nous tourner vers des solutions de modification d'exécution des tâches sans interruption de service et perte d'état du système. Ceci est appelé la **reconfiguration dynamique**.

Ces systèmes face à des contraintes (humaines, physiques, temporelles) sont toujours capables d'effectuer des tâches complexes. Ainsi, ces systèmes sont dotés d'intelligence. Ces systèmes fonctionnent grâce à leurs architectures logicielles embarquées. En effet, dans ces architectures logicielles embarquées sont implémentées des méthodes de l'intelligence artificielle.

De nos jours, l'intelligence artificielle est la science qui rend nos matériels intelligents. En effet, ces matériels effectuent des tâches complexes qui antérieurement n'étaient confiées qu'aux êtres humains.

Les travaux de cette thèse s'orientent sur une étude approfondie des méthodes de développement des systèmes embarqués à temps réel :

1. En passant d'abord par les méthodes d'analyse et de conception des systèmes embarquées,
2. Ensuite, en s'intéressant aux algorithmes pouvant implémenter la reconfiguration dynamique des architectures logicielles embarquées,
3. En étudiant et proposant un processus de développement de ces architectures logicielles embarquées,
4. Enfin, en faisant une étude de performance de ces systèmes à l'aide du modèle SWN.

Contexte de la thèse

Cette thèse est orientée sur les méthodes de développement des architectures logicielles embarquées afin d'assurer leur reconfiguration dynamique.

Ces architectures logicielles embarquées font partie de notre quotidien et constituent l'IoT. Elles sont présentes dans nos smartphones, voitures, matériels électroménagers sous forme de composants électroniques miniaturisés. Ces composants représentent des **microcontrôleurs**. En effet, un microcontrôleur est un circuit miniaturisé composé d'éléments électroniques (**processeur**, **mémoire morte** qui stocke le programme ou logiciel informatique et **mémoire vive** pour les données, unités périphériques et interfaces d'entrées et sorties) et possède une connectivité à internet. Cette dernière étend leur environnement d'exécution et facilite leur communication et interconnexion entre objets. L'**architecture logicielle embarquée** désigne la structure des algorithmes implémentés dans un équipement n'ayant pas une vocation informatique.

La **reconfiguration statique [Fat16]** est une amélioration d'une fonctionnalité (ou des fonctionnalités) du système qui s'effectue sous les yeux de l'utilisateur car nécessitant une installation de modules et un redémarrage.

Cependant, la **reconfiguration dynamique** est la capacité d'un logiciel à modifier son comportement (son architecture au besoin) selon l'évolution des exigences de son utilisation et la variation de son environnement d'exécution. Ainsi, cette modification du comportement du système se déroule pendant l'exécution à l'insu de son utilisateur.

Problématiques et Objectifs de la thèse

Cette thèse s'intéresse à la reconfiguration dynamique des systèmes embarqués. Il existe un large spectre de systèmes d'exploitation embarqués ainsi que diverses implémentations de reconfiguration dynamique des applications installées.

Un système est dit reconfigurable s'il peut modifier une de ses parties pendant son exécution. Ainsi, le système doit pouvoir répondre aux questions : Qui ? Quoi ? Quand ? Comment ? de sa configuration. Le système est dit reconfigurable s'il est capable de s'adapter aux différentes contraintes. Ces contraintes qui sont parfois contradictoires constituent les changements de son environnement et des exigences des utilisateurs. Ainsi, le système est aussi dit adaptatif et propose différentes modes de fonctionnement selon ces contraintes. Chaque mode de fonctionnement correspond à une configuration du système. Afin de réussir la mise en œuvre de ces systèmes adaptatifs, le concepteur doit prendre en compte :

- Les différentes modes de fonctionnement (Configurations) qui caractérisent le système. Chaque configuration correspond à l'exécution des algorithmes qui permettent son exécution avec succès. Cette dernière est effectuée grâce à l'allocation des parties matérielles compétentes
- Les mécanismes de transitions entre différentes configurations qui sont des algorithmes permettant au système de passer d'un mode de fonctionnement à un autre sans l'intervention humaine. De ce fait, le système est autonome et reste cohérent pendant et après les reconfigurations
- Les moyens qui permettent au système d'effectuer le choix adéquat de configuration selon les contraintes imposées par l'environnement et les utilisateurs

L'élément central d'une reconfiguration dynamique constitue les mécanismes (algorithmes) implémentés. La problématique d'un système reconfigurable ne réside pas seulement dans la définition des mécanismes (algorithmes) implémentés, la construction de système reconfigurable mais essentiellement comment programmer les reconfigurations une fois le système déployé.

La reconfiguration est une modification du comportement du système. Ainsi, cette modification concerne essentiellement l'architecture logicielle déployée. En effet, ces architectures sont composées d'algorithmes basés sur l'intelligence artificielle. Nous nous intéressons à un processus de développement des architectures logicielles embarqués.

Ainsi, nous proposons des objectifs qui sont :

- Une étude des systèmes reconfigurables
- Une étude complète de la reconfiguration des systèmes embarqués, plus précisément la reconfiguration dynamique

- Les algorithmes d'intelligence artificielle pouvant faciliter la reconfiguration dynamique d'architecture logicielle embarquée
- Une analyse complète des processus de développement des systèmes embarqués
- Une étude expérimentale de mise en œuvre d'architecture logicielle embarquée : Phase d'analyse et de conception, Phase d'implémentation et de générations de code
- Une étude de performance du système avec une implémentation de modèle stochastique Bien formé sous Greatspn

Plan de la thèse

Dans le cadre cette thèse, nous nous intéressons essentiellement aux différentes méthodes de conception des systèmes embarqués, aux algorithmes basés sur l'intelligence artificielle afin d'exécuter les reconfigurations dynamiques des systèmes embarqués.

L'intelligence artificielle en elle seule constitue un domaine riche de recherches. Cependant, la reconfiguration dynamique des architectures embarquées est un mode d'exécution des nouveaux systèmes utilisés qui les rendent de plus en plus sûrs, fiables, performants, intelligents.

Ce document est structuré en neuf chapitres principaux. Le premier chapitre présente les différents composants des systèmes embarqués. Nous nous intéressons aux nano-ordinateurs qui constituent des ordinateurs dont les caractéristiques (capacité de stockage, processeur, …) sont réduites et leurs systèmes d'exploitation. Ainsi, Ils peuvent détenir des périphériques d'entrée (Clavier, Souris, bouton poussoir, ..) et des périphériques de sortie (Ecran simple ou tactile, lampe, …). Les causes de reconfiguration dans ces systèmes et les choix technologiques disponibles font aussi l'objet d'étude dans ce premier chapitre.

Le second chapitre étudie les types de changement, les catégories d'adaptation de ces systèmes embarqués. Il s'intéresse aussi à l'avènement de l'intelligence artificielle et ses méthodes. Nous essayons de remonter jusqu'aux paradigmes énoncés dans la cybernétique. Cette dernière est considérée comme le point de départ de plusieurs domaines innovants, en l'occurrence l'intelligence artificielle, les méthodes mathématiques appliquées au traitement de l'information. Les méthodes étudiées basées sur l'intelligence artificielle sont : les arbres de décision, les réseaux de neurones, la logique floue, etc.

Le troisième chapitre est aussi dédié à l'étude des différentes méthodes d'analyse et de conception de systèmes à temps réel et embarquées. Ces méthodes d'analyse et de conception de ces systèmes sont issues de l'ingénierie dirigée par les modèles qui permettent aux concepteurs de mettre à évidence les fonctionnalités en fonction des besoins des utilisateurs, des contraintes physiques et temporelles.

Le quatrième chapitre permet de présenter le processus de développement des systèmes temps réel dynamiquement reconfigurables, sa démarche, ses solutions apportées.

Dans le cinquième chapitre, nous élaborons notre étude approfondie d'intergiciel dédié aux systèmes embarqués à temps réel.

Le sixième chapitre s'intéresse aux différents outils qui permettent la réalisation du processus de développement de ces systèmes embarqués.

Le septième chapitre est consacré à une étude expérimentale d'analyse et de conception d'architecture logicielle embarquée faite grâce à notre processus de développement de systèmes embarqués. Nous élaborons des modèles respectant l'architecture matérielle, logicielle, les contraintes des utilisateurs et de l'environnement de notre système à concevoir.

Dans le huitième chapitre, nous procédons à une étude de performance de notre système sous l'outil d'étude de performances, le ***GreatSPN***.

Le neuvième et dernier chapitre est destiné à la conclusion générale de ce document. Ce chapitre revient sur les travaux effectués, présente quelques améliorations qui pourraient être élaborées, avant de présenter d'éventuelles perspectives de notre recherche.

Première partie: Etat de l'art

Chapitre 1 : Rappels sur les concepts

Ce chapitre est consacré au fonctionnement des systèmes embarqués à temps réel. Ce qui nous permet de définir certains concepts de base : l'architecture logicielle embarquée, la configuration, la reconfiguration, les types de changements et les catégories d'adaptation. Ainsi, le système implémenté sera intelligible d'où l'importance de l'étude des méthodes d'algorithme basée sur l'intelligence artificielle. En effet, l'intelligence artificielle est la science qui permet aux systèmes d'accomplir des tâches complexes intelligemment.

1.1 Introduction

Un système d'exploitation est le programme central installé sur notre ordinateur, nano-ordinateur, voire microcontrôleur afin de faciliter à l'utilisateur ses interactions (utilisation de mémoire (vive ou morte), de périphériques), installation et utilisation d'applications (Ms Office, Antivirus,…) avec le système.

Nous pouvons citer divers systèmes d'exploitation reconfigurables. Ces systèmes peuvent être regroupés suivant certains critères d'architecture, de fonctionnement, …

Dans cette section, nous pouvons étudier brièvement :

- Systèmes à usage général : Ces systèmes d'exploitation sont utilisés afin de satisfaire les besoins d'un utilisateur donné (C'est le cas des systèmes d'exploitation : Linux, Windows). Généralement, ces systèmes possèdent des noyaux monolithiques. En effet, Ces noyaux présentent d'immense code source. Ainsi, ils ne sont reconfigurés (modifiés) qu'au niveau de certaines fonctionnalités qui sont généralement les pilotes de périphériques. Cependant, la reconfiguration des systèmes à usage général est utile dans les serveurs qui font l'objet de multiples requêtes à traiter simultanément,
- Systèmes micronoyaux : Ces noyaux rendent les systèmes plus simples (faciles à modifier), stables, sûres. En effet, seuls les services utiles sont chargés dans le micronoyau (simplification du code source); les autres se retrouvent dans l'espace utilisateur. La seule faiblesse des micronoyaux réside dans la lenteur d'exécution de ses tâches due aux nombreux mécanismes de communication utilisés,
- Systèmes extensibles : En effet, les applications installées dans ces systèmes peuvent améliorer l'architecture des systèmes d'exploitation. Ainsi, l'accès et la modification du noyau sont permis à toutes les applications du système extensible. Généralement, ces modifications permises du noyau apportent des gains de performances et des

améliorations de fonctionnement du système d'exploitation et des applications. Les modifications du système d'exploitation sont permises à toutes applications voire même prioritaires. En effet, le système réflexif est obligé de s'appuyer sur des politiques de sureté et fiabilité des codes intégrés sur son noyau.

1.2 Définition des concepts

1.2.1 Architecture logicielle embarquée

L'architecture logicielle embarquée est la partie centrale et incontournable d'un système embarqué. En effet, l'architecture logicielle embarquée d'un système comprend un ensemble d'éléments structurés pour le bon fonctionnement. Elle **[Fat13B]** comprend :

- Les composants : représentent les briques de base de l'architecture
- Les interfaces : sont associées à un composant et permettent d'accéder aux propriétés de celui-ci
- Les connecteurs : permettent de spécifier les liaisons entre les composants et les interfaces
- La configuration : représente un assemblage de composants, reliés aux interfaces par des connecteurs
- Le style architectural : représente toutes les contraintes imposées aux configurations
- L'ADL (Architecture Description Language) : est un langage qui définit les différents éléments décrits dernièrement

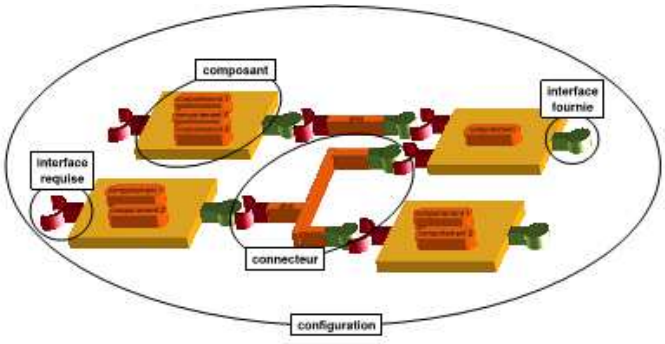

Figure 1: Représentation graphique d'une architecture logicielle embarquée

1.2.2 Configuration

La configuration d'un système embarqué représente l'architecture de ses composants afin d'accomplir une tâche. En effet, un composant est tout élément intervenant dans l'exécution d'une fonctionnalité du système. Chaque composant peut être divisé en sous composants. Cette

décomposition peut s'opérer jusqu'à l'obtention de composant indivisible c'est-à-dire atomique. Une configuration est bien définie pendant l'étape d'analyse et de conception du système. Cette étape est élaborée par les concepteurs du système grâce aux théories basées sur les modèles (MDA, MDE, …)

1.2.3 Reconfiguration

De nos jours, les systèmes embarqués font partie de notre quotidien. Afin de satisfaire les besoins de l'homme grandissants et exigeants, le système embarqué est obligé de s'adapter, de contourner les obstacles de fonctionnement, de donner les résultats escomptés dans un respect de délai malgré les aléas de l'environnement d'exécution. Ainsi, ce système intelligible qui assure son bon fonctionnement dans les meilleures conditions possibles de sécurité, d'intégrité, de fiabilité, de rapidité est appelé système reconfigurable. En effet, une reconfiguration [Guo08] est l'ensemble des modes d'adaptation d'un système en fonction des exigences des utilisateurs, des environnements d'exécution sans entraver à la qualité des résultats escomptés dans le respect des échéances fixés. Ainsi, nous proposons les différents types de changements et d'adaptation qui peuvent se dérouler dans les systèmes embarqués à temps réel

1.3 Les Types de changement dans un système

Afin d'assurer une bonne exécution, les systèmes doivent effectuer certains changements à divers niveaux : structure, déploiement, interface, implémentation.

1.3.1 Les changements de structure

Les changements de structure modifient la configuration structurelle du système. Les changements structuraux de base sont : l'ajout, la suppression de composants ou la modification d'interconnexions entre composants. Dans la figure ci-dessous, le changement de structure s'est effectué au niveau de la modification de la connexion Con1. Ainsi, les requêtes qui quittent C3 sont dirigées vers C5.

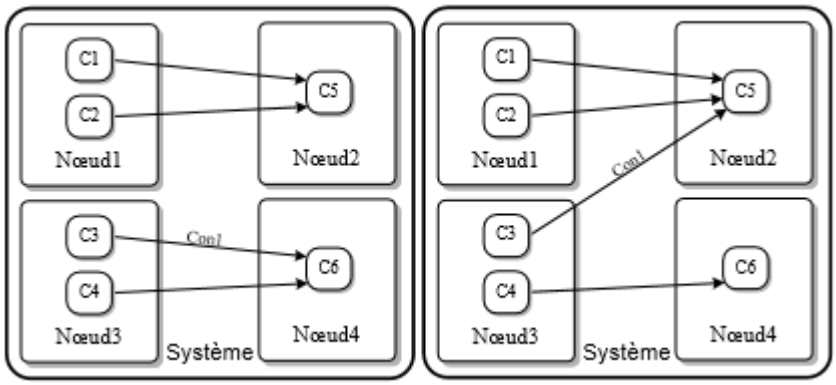

Figure 2: Exemple de changement de structure : Redirection de la connexion Con1 vers le composant C5 [Mou12]

1.3.2 Les changements de déploiement

Les changements de déploiement déplacent un composant dans un autre nœud d'exécution mais ne modifient jamais la structure initiale des connexions. Dans la figure 2 ci-dessous, le composant C6 se retrouve dans un nouveau nœud Noeud3. Cependant, les interconnexions restent inchangées.

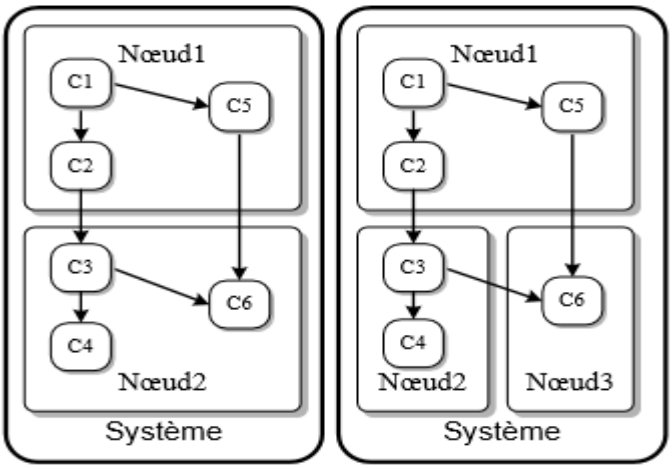

Figure 3: Exemple de changement de déploiement : Déplacement du composant C6 du Nœud2 vers le Noeud3 [Mou12]

1.3.3 Les changements d'interface

Les changements d'interface consistent à la modification de la signature des services fournis (figure 3 - A) par un paramètre (c) ou celle de l'ensemble des services fournis (figure 3 - B). En effet la signature d'un service comprend son nom, ses paramètres, leur type.

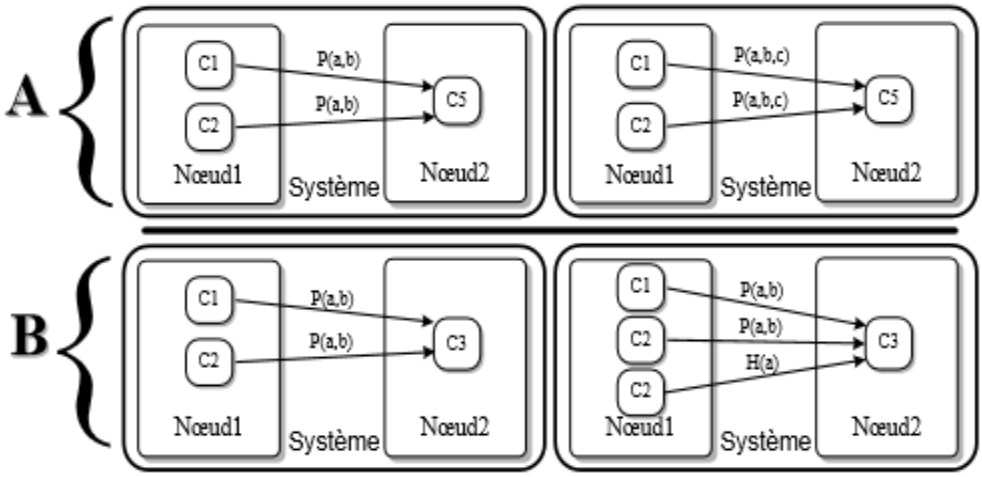

Figure 4: Exemples de changement d'interface [Mou12]

1.3.4 Les changements d'implémentation

Les changements d'implémentation modifient le code des composants et restructure leurs données sans entraver à leurs interfaces. Cependant, ils n'altèrent pas les placements, ni les connexions des composants.

1.4 Les catégories d'adaptation

Une étape importante d'un système reconfigurable est le plan de reconfiguration. Il détermine les changements (énoncés dans la section **1.3**) à effectuer sur le système en fonction des paramètres temporels, environnementaux et aux exigences de son utilisateur.
Ainsi, nous rencontrons ces différents plans de reconfiguration du système :

1.4.1 Le plan constructible

Le plan constructible permet au concepteur de définir la configuration initiale à l'aide d'un langage de description et son plan de reconfiguration avec un langage de modification. En effet, ce langage de modification devra supporter l'activation, la désactivation et le déplacement des composants et des interconnexions. Dans ce contexte, le programmeur implémente des configurations alternatives. Ainsi, le gestionnaire de reconfiguration choisit la **configuration adéquate. (Cf. figure 4- A)**

1.4.2 Le plan prédéfini

Le concepteur élabore plusieurs configurations alternatives, prédéfinies du système. Ces configurations sont bien conçues, vérifiées et validées. A l'exécution, le système

choisit la configuration la mieux adaptée à son contexte en s'appuyant sur une fonction d'aptitude connue. Ainsi, une maitrise de chaque configuration et des plans de reconfigurations vers les autres configurations sont obligatoires **(Cf. figure 4 - B)**

1.4.3 Le plan intelligent

Contrairement au plan précédent, le plan intelligent propose une série illimitée de configurations possibles. Le gestionnaire de reconfiguration choisit une configuration en fonction des modèles de performance. En effet, ce plan de reconfiguration n'est pas utilisé dans les systèmes à temps réel strict à cause de l'analyse effectué durant leur exécution. **(Cf. figure 4 - C)**

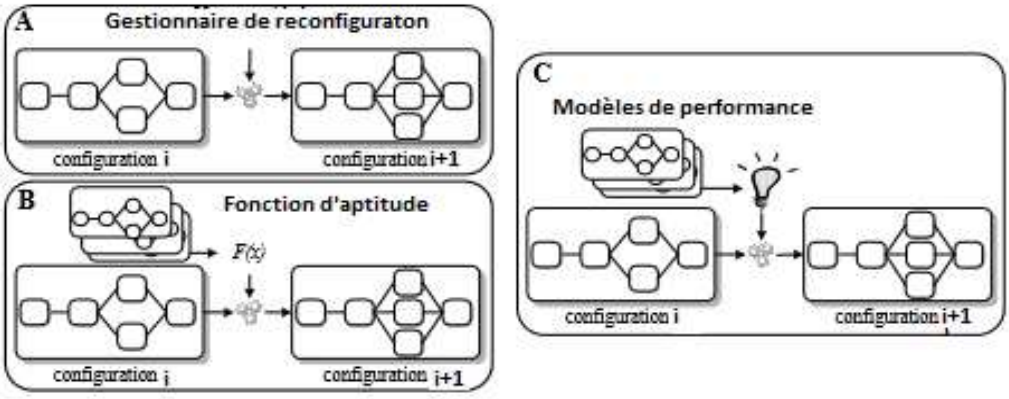

Figure 5: Les différentes catégories d'adaptation : Plan constructible (Figure 4 - A), Plan prédéfini (Figure 4 - B), Plan intelligent (Figure 4 - C) [Mou12]

Dans la section suivante **1.5**, nous allons nous intéresser à une nouvelle science novatrice nommée Intelligence Artificielle. Cette science introduit les techniques d'adaptation des systèmes embarqués sous forme d'algorithmes bien structurés.

1.5 L'intelligence artificielle

Ce point est dédié à une science innovante qui porte le nom d'intelligence artificielle abrégé I.A (A.I en Anglais Artificial Intelligence). Jadis, le mathématicien américain Norbert Wiener **[Web01**] introduisit une nouvelle science appelé Cybernétique **[Web02]**. En effet, la cybernétique **[Nor14] [Cél04]** fut défini comme l'ensemble des règles régissant les domaines naissants tels que l'automatique, l'électronique, les théories mathématiques appliquées à l'informatique. A partir de cette science naquirent certains domaines tels que les sciences cognitives, l'IA. L'origine de l'IA est probablement liée à l'article Imitation Game **[Tur50] [Web03]** du génie informaticien Alan Turing **[Web04]** en 1936. Ce test rebaptisé test de turing **[Web05]** en 1950 consistait à mettre en évidence l'intelligence avec laquelle une machine

arrive à soutenir une conversation avec l'homme. Ce dernier ne sent pas le caractère automate de la machine. Cette machine a su adopter le comportement humain. A partir de ce test naquit le concept de « machine consciente ». Ce concept fut introduit et défendu par Turing dans plusieurs conférences internationales :

- La conférence « L'intelligence de la machine, une idée hérétique »,
- Dans la conférence qu'il donne à la BBC 3e programme le 15 mai 1951 « Les calculateurs numériques peuvent-ils penser ? »
- La discussion avec M.H.A. Newman, Sir Geoffrey Jefferson et R.B. Braithwaite les 14 et 23 janvier 1952 sur le thème « Les ordinateurs peuvent-ils penser? »

Cette science a été développée par l'imminent chercheur John McCarthy **[Web06]** de l'université Stanford avec son collègue et cofondateur Marvin Lee Minsky **[Web07]** du MIT. En effet, l'IA est une science dédiée aux robots (nanordinateurs) et aux logiciels afin de les rendre intelligibles via des algorithmes implémentés en eux. Ces algorithmes dictent l'exécution des tâches accomplies par ces robots et logiciels de cette nouvelle ère. Jadis, ces tâches n'étaient accomplies que par les êtres humains dues à leur complexité. Leur exécution sollicite un certain niveau d'intelligence, de perception (sensibilité auditive, visuelle, tactile ou d'autres capteurs) et de raisonnement critique. Ainsi, l'IA est à la suite d'une évolution de plusieurs domaines de recherche qui sont :

- Recherches opérationnelles **[Web08]** dans les années 60,
- Supervision **[Web09]** dans les années 70,
- Aide à la décision **[Web10]** dans les années 80,
- Data mining **[Web11]** dans les années 90.

1.5.1 Différence entre l'intelligence artificielle forte et l'intelligence artificielle faible

Le concept d'IA faible cherche à créer des machines de plus en plus autonomes afin de réduire les couts de supervisions. En effet, ces supervisions des machines sont effectuées par des hommes. Les chercheurs de l'IA faible conçoivent des algorithmes capables de résoudre des problèmes d'une classe bien déterminée. Ainsi, ces machines dotées d'IA faible simulent l'intelligence. En effet, ELIZA **[Web12] [Web13]** est un programme d'IA faible créée par Joseph Weizenbaum **[Web14]**. ELIZA simule le comportement (jeux de questions, réponses) d'un psychothérapeute rogérien avec son patient. Dans son ouvrage « *Computer Power and Human Reason »,* Joseph Weizenbaum affirme que ces programmes informatiques simulent leurs intelligences. En effet, son ELIZA répond souvent par « Je comprends ». Cependant, le programme en réalité ne comprend pas son interlocuteur.

Le concept d'IA forte définit une machine capable non seulement d'effectuer des tâches de façon intelligente, d'éprouver une réelle conscience de soi, de vrais sentiments et de comprendre chacune de ses actions. De nos jours, les chercheurs défendent l'hypothèse de créer une intelligence consciente sur un support matériel. Selon les scientifiques, l'obstacle à créer une machine aussi consciente et intelligente que l'être humain ne réside pas sur l'aspect matériel mais plutôt sur la capacité de créer des algorithmes performants.

1.5.2 Méthodes implémentés par l'IA

En effet, l'IA développe des méthodes mathématiques permettant de rendre le système intelligent grâce à leur étape d'apprentissage par les expériences. Les méthodes les plus utilisées sont :

- Les arbres de décision **[Qui86] [Bre84],**
- Les réseaux de neurones,
- La régression multiple **[Fat13],**
- La logique floue **[Bou07]**.

En effet, les méthodes qui sont les arbres de décision, les réseaux de neurones et la logique floue feront l'objet de recherche approfondie dans la suite.

a) *Les arbres de décision*

L'arbre de décision est un outil d'aide à la décision qui connait un énorme succès. Il intervient dans des domaines variés (datamining ou fouille de données, sécurité, médecine, …). L'arbre de décision est une représentation lisible mettant en évidence les variations d'une variable à prédire (variable cible, variable de sortie) qui est la racine en fonction des variables descriptives (variables d'entrée, variables explicatives). Ces variables constituent les différentes branches à partir desquelles sortent les feuilles qui sont les résultats escomptés. Son succès est dû à:

1. Sa lisibilité de représentation : En effet, l'arbre de décision met en évidence la variable cible (nœud initial), ses relations entre les autres variables (nœuds fils) et ses différentes valeurs ou classes possibles (feuilles).
2. Sa capacité de sélection rapide des critères discriminants de la variable à prédire malgré l'importance de la base à explorer (la base échantillon).

   La figure ci-dessous met en évidence l'architecture d'arbres de décision.

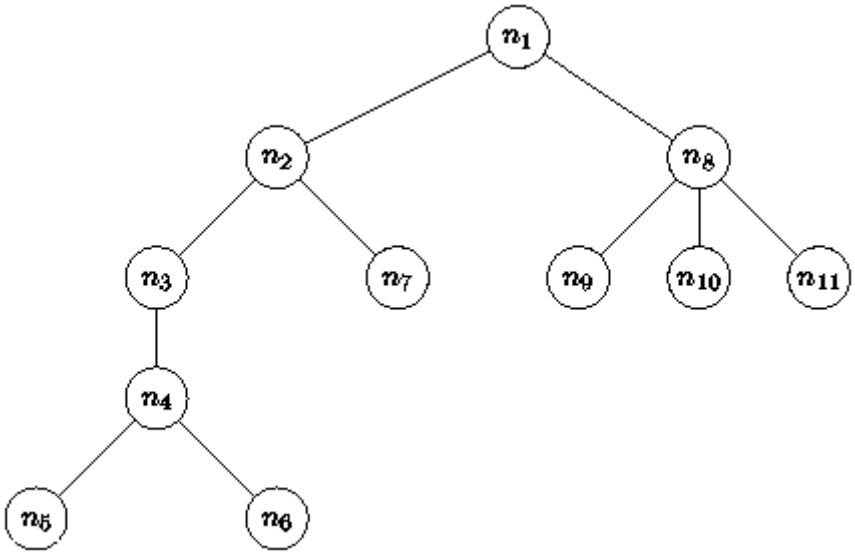

Figure 6: Exemple d'arbre de décision

Dans la **figure 6** ci-dessus, n_1 est la racine de l'arbre. Les nœuds n_2, n_8, n_3 et n_4 représentent les nœuds intermédiaires. Les nœuds n_5, n_6, n_7, n_9, n_{10} et n_{11} sont les nœuds terminaux (ou feuilles) de l'arbre.

On distingue deux types d'arbres de décision :

1. L'arbre de classification (Cluster Tree) permet de prédire à quelle classe la variable cible appartient,
2. L'arbre de régression (Regression Tree) permet de prédire la valeur numérique exacte de la variable cible.

En effet, l'apprentissage supervisé est appliqué pour l'obtention d'arbre de décision.

Dans la suite, nous allons nous intéresser aux algorithmes de construction d'arbre de décision [Chau00].

L'arbre de décision est obtenu à partir des big data des entreprises, des systèmes embarqués dotés de capteurs (température, présence, …) grâce à des algorithmes.

Ainsi, cet étape d'exploration d'informations et de recherche de liaisons entre variable cible et variables explicatives est appelé apprentissage.

En effet, le résultat de tout apprentissage est une fonction sous la forme :

Formule 1: Fonction Résultante d'apprentissage

$$Y= \beta_0+ \beta_1X_1+ \beta_2X_2+ \ldots + \beta_nX_n \text{ avec}$$

Y: Variable cible

$ß_0$: Terme Constante

$ß_1X_1, ß_2X_2, .., ß_nX_n$: Variables explicatives

Les algorithmes de construction d'arbres de décision sont:

- ID3,

- CHAID (**CH**i-squared **A**utomatic **I**nteraction **D**etector),
- CART.

Ces algorithmes s'appuient sur des méthodes de sélection des critères de segmentation. En effet, ces méthodes permettent de diviser l'arbre du sommet vers ses feuilles afin de déterminer pour un nœud donné sa variable d'entrée. Cette variable d'entrée choisie correspond à celle qui maximise le critère donné. Les méthodes les plus utilisés sont le coefficient de Gini et l'entropie de Shannon. Ainsi, nous allons essayer de décrire chacune des algorithmes énoncés précédemment.

- **ID3**

ID3 est un algorithme de construction d'arbre de décision à partir de la connaissance de la variable à prédire appelée *attributCible*, des variables descriptives notées *attributsNonCibles*, des enregistrements de la base d'apprentissage (Big data) nommés *exemples*. En effet, le big data est utilisé comme l'échantillon représentatif. Les variables utilisées sont aussi bien numériques que qualitatives. C'est un algorithme récursif très apprécié qui s'appuie sur l'entropie de Shannon afin de choisir la variable non cible qui sera le prochain nœud. En effet, cette variable maximisera le gain d'information. L'algorithme ID3 est décrit comme suit :

```
fonction ID3 (exemples, attributCible, attributsNonCibles)
  si exemples est vide alors /**** Nœud terminal avec Erreur****/
    retourner un nœud Erreur
  sinon si attributsNonCibles est vide alors /***Nœud terminal ****/
    retourner un nœud ayant la valeur la plus représentée pour attributCible
  sinon si tous les exemples ont la même valeur pour attributCible alors /* Nœud terminal */
    retourner un nœud ayant cette valeur
  sinon /****Nœud intermédiaire ****/
    attributSélectionné = attribut maximisant le gain d'information parmi attributsNonCibles /*** Choix du nouveau attributCible dans la liste des attributsNonCibles via le Calcul du gain d'information à partir des formules de calcul d'entropie de Shannon et de gain d'information pour l'ensemble des variables non cibles restantes***/
    attributsNonCiblesRestants = suppressionListe(attributsNonCibles, attributSélectionné) /**Suppression de l'attribut sélectionné dans la liste des attributsNonCibles**/
    nouveauNœud = nœud étiqueté avec attributSélectionné
    pour chaque valeur de attributSélectionné faire
      exemplesFiltrés = filtreExemplesAyantValeurPourAttribut(exemples, attributSélectionné, valeur)
      nouveauNœud->fils(valeur)=ID3(exemplesFiltrés, attributCible, attributsNonCiblesRestants) /** Appel de la fonction récursive***/
    finpour
    retourner nouveauNœud
```

Figure 7: Algorithme ID3 de la construction d'abre de décision

A partir de cet algorithme, vont naitre d'autres versions plus performantes connues sous le nom de C4.5, C5

- **CART**

CART est une méthode d'obtention d'arbre de classification. En effet, il s'appuie sur la logique de l'ID3. Cependant, l'arbre obtenu est binaire (chaque nœud ne peut avoir que 2 fils) et la sélection du critère de segmentation s'effectue avec le coefficient de Gini

- **CHAID**

CHAID est un algorithme de construction d'arbre de décision. Cet algorithme se divise en 3 étapes :

1. Choix des variables explicatives,
2. Identification des classes de la variable cible,
3. Sélection de la variable de segmentation.

L'apprentissage est effectué sur un échantillon dit représentatif des données de la base. Plus le problème est complexe plus l'arbre est grand (plus le nombre de branches et de feuilles est grand). Dans certains cas de construction d'arbres, le nombre de feuilles est égal au nombre

d'enregistrements de la base. Ainsi, vient le problème de surapprentissage (en Anglais over-fitting).

En effet, le surapprentissage donne au système un ensemble de données (échantillon) et ses résultats. Face à cette situation, le système perd ses capacités de prédiction et ne se limite qu'à ses résultats énoncés. Ainsi, le système est incapable de donner des résultats d'un nouvel échantillon de données.

Afin d'éviter le surapprentissage, sont mises en place des conditions d'arrêt de construction de nouveaux nœuds appelé le pré-élagage. A cette étape, l'homogénéité du groupe d'individus et le nombre d'individus réduits sont estimés atteints. Ainsi, il n'est plus nécessaire de diviser à nouveau le groupe mais de lui attribuer sa valeur ou classe (création de feuille). Une second méthode permet d'éviter le surapprentissage est appelée le post-élagage. Ce dernier comprend deux étapes :

- Une première construction d'un arbre avec ses feuilles est d'abord effectuée à partir d'une fraction de l'échantillon choisi,
- L'arbre obtenu est réduit à partir de la second fraction de données afin d'améliorer les performances.

Les arbres de décision ne peuvent représenter que la fonction logique AND. Ceci est une des limites de cette méthode d'IA.

b) *Les réseaux de neurones*

Le réseau de neurones constitue une méthode mathématique fortement inspirée du fonctionnement des neurones biologiques. En effet, le réseau de neurone applique la logique d'induction plus connue sous le nom d'apprentissage afin de présenter ses résultats. Cette méthode mathématique résulte d'une suite de recherches approfondies et enrichies :

- Le premier article de ce domaine intitulé « What's the frog's eye tells to the frog's brain » en français « ce que l'oeil de la grenouille dit au cerveau de la grenouille » fut publié par les neurologues Warren McCulloch et Walter Pitts. A partir de cet article, naquit le concept de neurone formel. Ce dernier est comparable au neurone biologique qui reçoit et véhicule des informations (provenant de neurones d'entrées) via ses connexions appelées synapses affectées par des poids synaptiques. Une somme pondérée est appliquée aux valeurs de sortie de neurones. Ensuite, une fonction d'activation introduite sur la somme obtenue permet d'avoir une sortie. Cette sortie est effectivement le résultat du neurone formel. Ces dernières étapes énoncées constituent la phase d'apprentissage. En effet, le réseau de neurones s'appuie sur l'apprentissage afin d'opérer rapidement à une classification à partir d'une valeur seuil. Ainsi surgissent des questions fondamentales:

Par quelle règle s'appuie-t-on pour avoir les coefficients (poids) synaptiques ?

Comment déterminer la fonction d'activation ?

Comment évaluer la valeur seuil qui permet d'opérer à une classification?

- Le physiologiste canadien Donald Hebb**[Web15]** apporta des réponses à ces questions via son article *The organization of bevahiour* **[Web16]**. Ainsi naquit la célébre règle de Hebb qui permet de déterminer les coefficients synaptiques en fonction de l'activité qui relie les neurones. La règle fut résumée par « des neurones qui stimulent en même temps, sont des neurones qui se lient ensemble.» (« *Cells that fire together, wire together*.»). Cette théorie s'appuie sur une hypothèse en neuroscience qui défend que l'apprentissage crée ou renforce les liaisons entre neurones du cerceau. Ainsi, ces liaisons créées rendent l'exécution de la tâche plus fiable, plus rapide.
- Cette règle fut le déclic spectaculaire des réseaux de neurones. Ainsi, naquit le modèle du perceptron de Franck Rosenblatt en 1957. Ce modèle fut le premier système capable d'apprendre par expérience. Ses limites furent mises en évidence par Marvin Lee Minsky et Seymour Papert dans un article. Ainsi, le modèle de **perceptron** était incapable de résoudre les problèmes non linéaires. En effet, le perceptron est la représentation le plus simplifiée d'un réseau de neurones. Dans sa version plus simple, le perceptron comprend deux entrées et une sortie. Il ne permet que de résoudre des problèmes linéaires.
- En 1982, John Joseph Hopfield proposa un nouveau modèle des réseaux de neurones
- En 1986, Rumelhart offrit une nouvelle ère aux réseaux de neurones avec son modèle de perceptron multicouches capables de traiter des problèmes non linéaires. Ce modèle se base sur la rétropropagation du gradient de l'erreur. Ainsi, les réseaux de neurones devinrent le domaine de recherche le plus prisé de l'IA.

A partir de ces recherches, vont naitre différents types de réseaux de neurones qui sont :

1. Les perceptrons multicouches :

   Le perceptron multicouche est la structure la plus simplifiée et la plus utilisée des réseaux de neurones. Il comprend une couche d'entrée, une couche de sortie et une ou plusieurs couche(s) cachée(s). Un neurone (cellule) n'est relié qu'aux neurones de couches précédentes. La fonction d'activation utilisée est la somme pondérée. Les figures ci-dessous illustrent un perceptron avec deux entrées et des sorties (sans couche cachée) (fig : Perceptron monocouche avec 2 entrées et 2 sorties) et un perceptron multicouche avec deux entrées et des sorties (avec une couche cachée) (fig : perceptron avec deux entrées et des sorties avec une couche cachée)

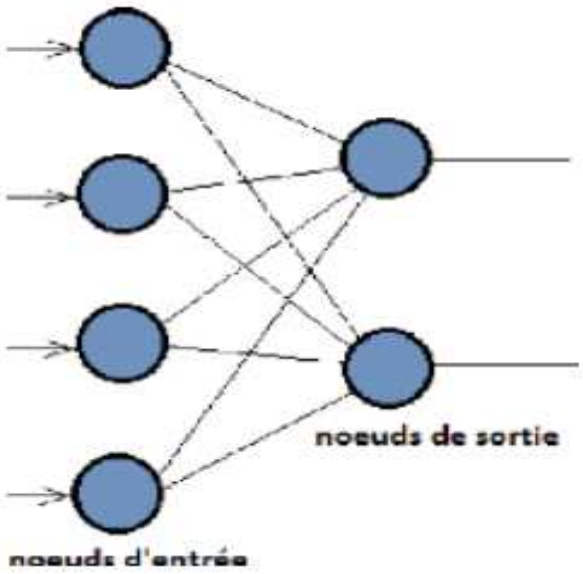

Figure 8: Perceptron monocouche avec 2 entrées et 2 sorties

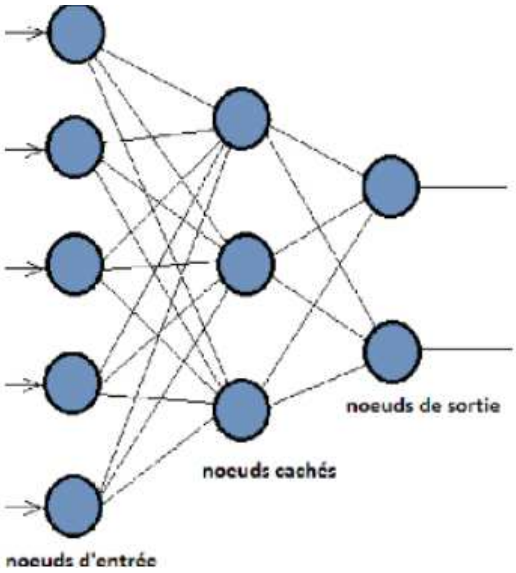

Figure 9: perceptron avec deux entrées et des sorties avec une couche cachée

Ainsi, le fonctionnement simplifié d'un perceptron (avec n neurones d'entrées et une sortie) se résume ainsi :

- La somme pondérée des valeurs d'entrées et des coefficients synaptiques afin d'obtenir une valeur totale d'entrée,
- Le choix d'une fonction d'activation appliquée à la valeur totale d'entrée afin d'aboutir à la valeur ou résultat de sortie.

NB :

La fonction d'activation peut être :

- une fonction à seuil : On applique la comparaison entre la somme pondérée des entrées et des poids synaptiques et le seuil choisi. Ainsi, si la somme est supérieure au seuil la sortie est égale à 1

Sinon la sortie est égale à 0.

La figure ci-dessous illustre l'évolution de la courbe d'une fonction à seuil

Figure 10: Courbe de la fonction à seuil

- une fonction linéaire par morceaux : On applique différentes fonctions linéaires sous la forme y= ax+b à la somme pondérée.

  La figure ci-dessous montre l'évolution de la courbe d'une fonction linéaire par morceaux

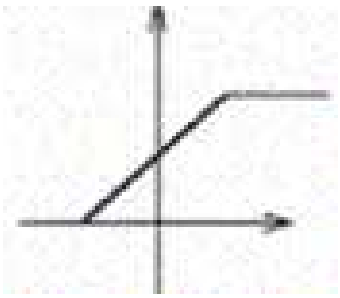

Figure 11: Courbe de la fonction linéaire par morceaux

- une fonction sigmoïde : On utilise la fonction sigmoïde sous la forme $y=1/1+e^{-x}$ à la somme pondérée.

  La figure ci-dessous montre l'évolution de la courbe d'une fonction sigmoïde.

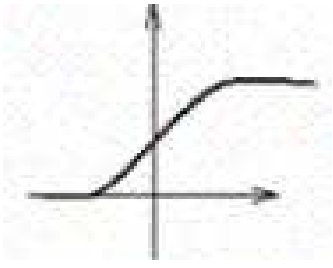

Figure 12 : Courbe de la fonction sigmoïde

- une fonction gaussienne : On implémente la fonction gaussienne sous la forme $y= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ à la somme pondérée.

La figure ci-dessous montre l'évolution de la courbe d'une fonction gaussienne.

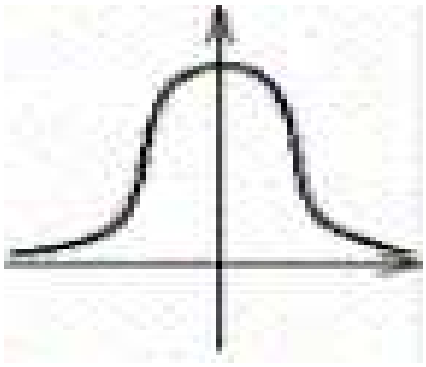

Figure 13 : Courbe de la fonction gaussienne

2. Les réseaux de kohonen ou cartes de kohonen

Cette famille de réseau de neurones présente une couche d'entrées connectée à une couche de sorties dite couche de compétition. Dans cette couche, chaque sortie cherche à être le vainqueur. En effet, le neurone vainqueur possède la sortie la plus proche de la valeur escomptée et minimise le plus l'erreur (différence entre la valeur obtenue et celle souhaitée).La famille de réseau de kohonen peut être divisée en trois sous familles qui sont :

- VQ (*Vector Quantization*) : Introduite par Grossberg (1976), la quantification vectorielle permet de retrouver les différents groupes qui forment un ensemble de données,
- SOM (*Self Organizing Map*) : Issus des travaux de Faussett (1994) et Kohonen(1995), les SOMs permettent de classifier des ensembles de données. Ils permettent de catégoriser en deux dimensions et d'identifier les différents groupes d'un ensemble de données,
- LVQ (*Learning Vector Quantization)* : Proposée par Kohonen (1988), LVQ permet de classifier des données par recherche du plus proche voisin.

La figure14 ci-dessous montre l'architecture d'une carte de kohonen

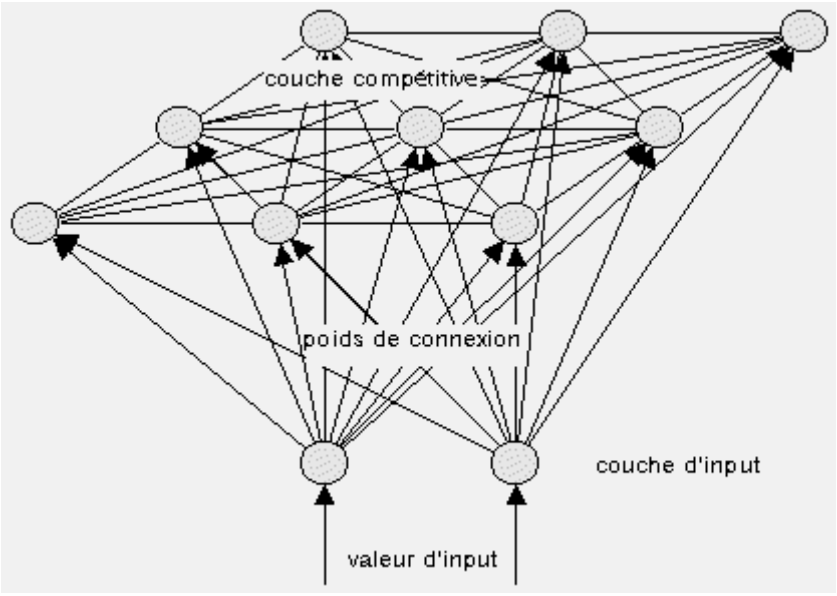

Figure 14 : Une carte de Kohonen avec ses valeurs d'entrées, ses connexions synaptiques et ses valeurs de sortie (Couche compétitive)

3. Les réseaux de Hopfield

Cette famille de réseau est la plus complexe. Chaque neurone est relié à tous les autres neurones. Les changements de valeurs des cellules s'opèrent en cascade jusqu'à l'obtention d'un état dit stable. Ces réseaux sont bien adaptés à la reconnaissance des formes. La figure 13 ci-dessous montre l'architecture de réseau de Hopfield

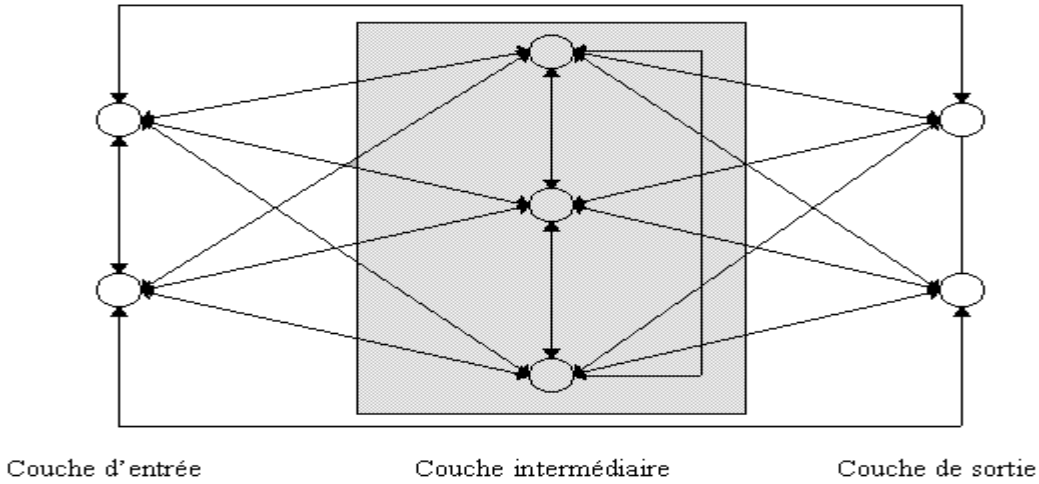

Figure 15: L'architecture de réseau de Hopfield

En effet, l'apprentissage constitue une étape sensible des réseaux de neurones. A cette étape, les coefficients synaptiques sont modifiés suivant les différents algorithmes. Ainsi, l'apprentissage peut avoir une durée assez longue. Cependant, une fois les poids synaptiques trouvés ; le calcul du résultat de sortie pour un nouvel échantillon va être rapide.

A partir des réseaux neurones, nous distinguons les différents types d'apprentissage :

- L'apprentissage supervisé : Ce type d'apprentissage consiste à modifier les coefficients synaptiques afin de comparer la valeur calculée et celle attendue. En effet, l'arrêt de l'apprentissage s'effectue quand la valeur calculée est égale ou proche de celle escomptée (cette valeur est bien sûr celle de la base d'échantillon). En effet, les algorithmes du rétropropagation du gradient et quickprop permettent d'accomplir l'apprentissage supervisé,
- L'apprentissage par renforcement : Ce type d'apprentissage est considéré comme un apprentissage supervisé. L'apprentissage met en évidence la corrélation entre les valeurs d'entrée et de sortie via une estimation d'erreur. Le réseau devra tenter de maximiser un index de performance qui lui est donné. Le système devra être capable de savoir si la réponse donnée est bonne ou non. Cependant, la réponse escomptée n'est pas connue d'avance,
- L'apprentissage non supervisé : Pour cette catégorie d'apprentissage, la valeur de sortie attendue est inconnue, seules les entrées sont disponibles,
- L'apprentissage hybride : Dans ce cas, l'apprentissage est effectué premièrement via les deux premières approches (L'apprentissage supervisé et l'apprentissage par renforcement). Ensuite, l'apprentissage du reste de l'échantillon s'effectue via l'apprentissage non supervisé.

Dans la suite, nous nous intéressons aux différents algorithmes d'apprentissage utilisés dans les réseaux de neurones qui sont :

- Rétropropagation du gradient,
- Quickprop.

La rétro propagation est une méthode qui permet de corriger les coefficients synaptiques des neurones. Cette correction des poids synaptiques s'effectue de la dernière couche vers la première. Au début, les coefficients sont initialisés avec des valeurs aléatoires. A partir des valeurs prises de la base test, on procède à des itérations successives. Chaque itération permet de modifier les coefficients synaptiques afin d'atteindre ou rapprocher la valeur de sortie avec celle du test. En effet, cette dernière est la valeur attendue.

c) *La logique floue*

Introduite en 1965 par Lotfi Zadeh, la logique floue (en anglais Fuzzy Logic) **[Bou95]** est aussi un outil de l'IA qui vient étendre la logique classique. Cette logique floue est appliquée à plusieurs domaines **[Bou98]** :

- L'aéronautique avec des solutions d'aide au pilotage,
- Le milieu bancaire avec des outils de gestion des prêts attribués aux clients,
- La médecine avec des appareils d'aide au diagnostic des patients,

- La météorologie avec des appareils de prévision de température, d'humidité …,
- La gestion de la circulation routière.

En effet, la logique floue introduit un nouvel ensemble nommé ensemble flou **[Zad65]** et de nouvelles propriétés intéressantes **[Zad68]**. Elle permet de représenter certaines incertitudes et imprécisions **[Kli88]** et facilite l'implémentation de ces dernières dans des systèmes intelligents.

Un ensemble flou E est caractérisé par une fonction d'appartenance $f_E(x)$ permettant d'attribuer à chaque élément noté x une valeur réelle comprise entre [0;1]. En effet, cette valeur réelle représente le degré d'appartenance de x dans E. Cependant, pour un ensemble classique sa fonction d'appartenance n'admet que deux valeurs soit 0, soit 1.

En logique floue, un sous ensemble (ou partie) de E notée A est associée à une fonction caractéristique. Ainsi, Elle prend la valeur 0 si x qui est élément de E n'appartient pas à A. Elle vaut 1 si x appartient à A. Cependant elle prend une valeur comprise entre [0;1] si x appartient à A avec un degré d'appartenance noté $\mu_A(x)$.

Ainsi, découlent certaines propriétés qui sont :

- Le noyau d'une partie floue A représente l'ensemble des éléments x appartenant totalement à A. Leur fonction d'appartenance équivaut à 1

Formule 2: Le noyau d'une partie floue A de E

$$n(A) = \{x \in E \mid \mu_A(x) = 1\}$$

- Le support d'une partie floue A représente l'ensemble des éléments x n'appartenant quasiment pas à A. Leur fonction d'appartenance est différente de0

Formule 3: Le support d'une partie floue A de E

$$supp(A) = \{x \in E \mid \mu_A(x) > 0\}$$

- La hauteur d'un sous ensemble flou A de E est défini par :

Formule 4: La hauteur d'une partie floue A de E

$$h(A) = \sup\{\mu_A(x) \mid x \in E\}$$

- La sous partie A de E est caractérisée par l'ensemble de ses α-coupes. En effet, une α-coupe est l'ensemble des éléments x appartenant à A dont leur degré d'appartenance est supérieur ou égal à α.

Formule 5 : Fonction α-coupe de A

$$\propto -coupe(A) = \{x \in E \,|\mu_A(x) \geq \propto\}$$

- Soient les fonctions caractéristiques notées μ_i de la réunion des i sous ensembles de E constituant la famille I. Ainsi, la fonction caractéristique μ(x) de leur réunion est obtenue :

Formule 6 : Fonction caractéristique de la réunion des sous ensembles de E

$\mu(x) = \sup\{\mu_i(x), i \in I\}$, ce qui sera noté $\mu = \bigvee_{i \in I} \mu_i$

- La fonction caractéristique μ(x) de l'intersection de ces sous ensembles est :

Formule 7 : Fonction caractéristique de l'intersection des sous ensembles de E

$\mu(x) = inf\{\mu_i(x), i \in I\}$, ce qui sera noté $\mu = \bigwedge_{i \in I} \mu_i$

- La fonction d'appartenance du complémentaire d'une sous partie de E associée est calculée ainsi : 1- μ

  Avec μ la fonction d'appartenance de la sous partie de E

  Dans un ensemble classique, le noyau et le support sont confondus (car la fonction caractéristique de l'ensemble ne peut prendre que deux valeurs 0 ou 1)

  Le système doté d'une logique floue doit pouvoir prendre des mesures (valeurs numériques) et les collecter dans une base de connaissance. Ces mesures prises doivent être interprétés afin de faciliter le comportement du système. La base de connaissance permet de bien définir les différentes étapes d'obtention du système à logique floue qui sont : la fuzzification, l'inférence floue et la défuzzification.

  Afin d'implémenter un système doté d'une logique floue, on devra suivre:

  La fuzzification : constitue la première étape qui consiste à transformer les valeurs numériques en valeurs linguistiques.

  Par exemple nous avons la variable température qui est à 20°C (valeur numérique) à un instant t de la journée. Cette variable prend des valeurs linguistiques (degré d'appartenance) aux ensembles flous: Très froid, Froid,

Tempéré, Chaud, Très chaud. En effet, chaque valeur linguistique est obtenue via une fonction d'appartenance définie par le concepteur du système.

L'inférence floue : aussi appelé moteur d'inférence du système. Elle est la deuxième étape d'implémentation de notre modèle flou. Cette étape consiste à mettre en évidence les relations existant entre variables linguistiques et les résultats qui en découlent. Cette étape permet au concepteur de concevoir l'ensemble des règles qui régissent le fonctionnement du système. En effet, les règles définies sont sous la forme d'enchainements de SI condition ALORS Action. Afin d'énoncer correctement les règles, le concepteur utilise les opérateurs ET, OU, NON de la logique floue.

L'opérateur ET en logique floue représente l'intersection entre deux ensembles flous. L'opérateur ET peut être traduit ainsi :

- L'opérateur de minimalité : a ET b = min (a ; b)
- L'opérateur produit : a ET b= a.b La formule de calcul de produit d'ensembles flous est la suivante :

Formule 8 : L'opérateur ET en logique floue

$$a\ ET\ b = \gamma.\min\ (a,b) + (1-\gamma).\frac{a+b}{2}$$

L'opérateur OU (en logique floue) correspond à l'union de deux ensembles flous. Il peut être traduit ainsi :

- L'opérateur de maximalité : a OU b= max (a ; b)
- L'opérateur de produit La formule d'obtention du produit est la suivante :

Formule 9: L'opérateur OU en logique floue

$$a\ OU\ b = \gamma.\max(a,b) + (1-\gamma).\frac{a+b}{2}$$

Le paramètre γ est une valeur comprise entre 0 et 1 définie par le concepteur du système flou.

L'opérateur NON en logique floue définit le complément de l'ensemble flou.

Formule 10 : L'opérateur NON en logique floue

NON a= 1-a

Ainsi, le résultat de chaque relation entre variables linguistiques est une valeur linguistique.

La défuzzification : représente la troisième et dernière étape qui consiste à obtenir une valeur numérique à partir des valeurs de la variable linguistique. En effet, ces valeurs sont obtenues à partir des règles de l'étape précédente. Ainsi, le but de la défuzzification est de combiner ces valeurs afin d'obtenir une valeur numérique unique.

La défuzzification se résume en deux étapes :

D'abord on applique la fonction OU logique sur les valeurs de la variable linguistique. Enfin, on obtient un résultat linguistique à transformer en valeur numérique via les méthodes de défuzzification prédéfinies :

La première méthode est celle de la moyenne des maximas qui consiste à repérer d'abord la fonction d'appartenance ayant la variable linguistique la plus élevée. Ensuite, on calcule la moyenne des abscisses ayant leurs ordonnées supérieures ou égales à cette valeur élevée. L'expression de la moyenne des maximas est la suivante :

Formule 11 : L'expression de la moyenne des maximas

$$valeur = \frac{\int_s x.dx}{\int_s dx} \text{ Avec } s = \{x, \mu(x) = \sup(\mu(x))\}$$

La seconde étape est celle du centre de gravité qui consiste à calculer l'abscisse correspondant au centre de gravité de la fonction d'appartenance. La méthode de calcul du centre de gravité est la suivante :

Formule 12 : La méthode de calcul du centre de gravité

$$valeur = \frac{\int_s \mu(x)x.dx}{\int_s \mu(x).dx}$$ Avec s le domaine de la fonction d'appartenance

De nos jours, la logique floue est implémentée dans des outils **[Gui11]** qui sont performants.

1.5.3 Langages de programmation de l'IA

A ses débuts, certains langages comme Lisp **[Web17]** et Prolog **[Web18]** semblaient plus adaptés à la conception de programmes dotés d'IA. Le premier programme informatique d'IA nommé ELIZA était écrit sous 3 pages de SNOBOL **[Web19]**. Avec l'évolution des langages de programmation, l'IA va de plus en plus se tourner vers C, C++, Java. En effet, ces deux derniers langages sont évolués et s'appuient sur le concept objet.

1.5.4 Domaines d'application

De nos jours, l'IA intervient dans différents domaines :

- Automobile : Les voitures d'aujourd'hui présentent plus de fonctionnalités (détection explicite de pannes pouvant survenir),
- Aéronautique : Les accidents se font de plus en plus rares grâce aux avions sophistiqués et aux tours de contrôle bien équipés,
- Médecine : Les appareils médicaux aident au diagnostic grâce à leurs fonctionnalités de plus en plus développées,
- Jeux : les jeux actuels sont de plus en plus proches du monde réel,
- Secteur bancaire : Les programmes informatiques utilisés ne se limitent plus au stockage, transfert, traitements de données. Ils interviennent dans les simulations et prévisions de résultats, les prises de décisions. Ces derniers permettent d'introduire une nouvelle science appelée le Datamining **[Agr96] [Bent04]** (fouille de données).

1.6 Conclusion

De nos jours, les systèmes embarqués à temps réel sont conçus avec différentes configurations alternatives qui répondent aux divers contextes d'exécution. Ces contextes représentent les exigences des utilisateurs, de l'environnement, du temps. En effet, ces systèmes sont dits reconfigurables. Un système reconfigurable comprend un plan de reconfiguration (constructible, prédéfini ou intelligent) qui permet de définir les configurations possibles, les lois de choix de la configuration adéquate, les méthodes de transitions entre configurations. Ce système doit accomplir des tâches complexes d'où l'importance de leurs algorithmes qui y sont implémentés. Ces algorithmes (Arbres de décision, Réseaux de neurones, Logique floue, …) introduisent une nouvelle science appelée Intelligence Artificielle. L'IA permet aux systèmes à vocation non informatique de remplir leurs tâches intelligemment.

Chapitre 2 : Analyse et conception de système embarqué à temps réel

2.1 Introduction

Ce chapitre est consacré à l'étude des méthodes d'analyse et de conception des systèmes embarqués. L'analyse de ces systèmes doit prendre en compte les fonctionnalités en fonction des exigences des utilisateurs, de l'environnement dans le respect des délais d'exécution. En effet, l'implémentation de toute solution logicielle passe par une étape d'analyse et de conception permettant d'énoncer, d'étudier les besoins des utilisateurs du système. Cette étape tout étant préparatoire permet de respecter les cahiers de charges proposés (les attentes des utilisateurs).

2.2 L'ingénierie dirigée par les modèles

En 2005, l'OMG introduit l'approche MDA qui permet d'assurer la mise en œuvre et la maintenance des systèmes logiciels. Cet approche s'appuie sur la séparation des spécifications du système indépendantes aux plateformes appelés PIM et celles liées aux plateformes PSM . MDA introduit deux notions importantes qui sont :

1. la transformation de modèles initiaux en modèles dérivés grâce au langage standard QVT.
2. la génération de code ou de documentation à partir de modèles initiaux grâce au second langage standard MOFM2T.

Par la suite, les concepteurs introduisent l'IDM (en Anglais MDE) **[Ben06]**. Cette approche respecte les concepts de MDA. De nos jours, les systèmes embarqués à temps réel (système composé de matériel, de logiciel, environnement d'exécution) répondent mieux aux besoins des utilisateurs. En effet, leur réalisation passe par une phase dite d'analyse et de conception. Cette étape est primordiale car permettant aux concepteurs de traduire l'ensemble des besoins, des contraintes (physiques, temporelles) en des modèles (diagrammes en UML). En effet, UML est l'un des langages les plus prisés.

2.2.1 Le langage de modélisation UML (Unified Modeling Language)

UML **[Web20]** est un langage de modélisation et d'analyse de solutions informatiques orientées objet **[Chri92]**. UML est un langage d'analyse à usage général, il peut être utilisé dans plusieurs domaines (la santé, la finance, la télécommunication, …). UML est né à la fin 1995 à la suite de

l'unification des méthodes de plusieurs scientifiques qui sont : Grady Booch, James Rumbaugh et Ivar Jacobson. Au fur et à mesure, UML est amélioré et connut un essor exponentiel. UML est utilisé par plusieurs plateformes (CORBA, Java 2 Entreprise Edition J2EE, Microsoft .Net). UML s'appuie sur l'approche objet. Un objet est toute entité conceptuelle, matérielle ayant un nom, des caractéristiques et un comportement. UML présente différents diagrammes qui sont :

- Diagramme de cas d'utilisations met en évidence les fonctionnalités du système tel que l'acteur les voit.
- Diagramme de classes définit l'ensemble des classes et associations (relations existantes entre classes) du système. En effet, une classe est une entité informatique permettant de représenter un élément dans le réel. Elle comprend un nom, un ensemble d'attributs et des méthodes
- Diagramme d'interactions permet d'établir un pont entre le diagramme de cas d'utilisations et le diagramme de classes. Il montre comment des instances du système communiquent pour réaliser une fonctionnalité bien définie.
- Diagramme d'activités représente l'enchaînement des actions aboutissant à l'exécution d'une fonctionnalité du système
- Diagramme de séquences étudie la succession chronologique des messages envoyés entre acteurs afin d'accomplir une fonctionnalité du système
- Diagramme de collaboration identifie les acteurs et les messages échangés entre eux afin d'exécuter une fonctionnalité du système
- Diagramme d'états-transitions met en évidence les différents états du système pendant son exécution
- Diagramme de composants est une vue de type statique illustrant l'implémentation d'un système selon les choix de réalisation. Ainsi un diagramme de composants comprend:
    1. des descriptions des implémentations du système (les composants)
    2. des regroupements de ces implémentations (les modules)
    3. des relations entre ces implémentations (les dépendances)
- Diagramme de structure composite décrit la structure interne d'une ou de plusieurs classes. Dans ce type de diagramme, la classe comprend un ensemble de parties reliées par des connecteurs. Une partie possède un type et une multiplicité. Le type d'une partie peut être un composant. En effet, un composant est la représentation d'une partie physique du système. Il implémente des services utilisables par d'autres composants.
- Diagramme de machine à états représente les différents états (réponses) que peut avoir une classe face aux différents événements perçus. Il met e évidence les différents

états et transitions d'une instance d'objet. La réponse d'une instance de classe dépend de son état initial. En effet, ce diagramme peut relier une instance de classe à des cas d'utilisations, des comportements, des collaborations afin d'y définir ses différents états.

- Diagramme de profil permet d'étendre les fonctionnalités d'UML. Ce diagramme facilite la représentation des exigences de certains domaines (Télécommunications, systèmes sur puce, …).
- Diagramme de déploiement montre la disposition physique des différentes ressources matérielles (nœuds) qui composent le système. Il met en évidence la répartition des composants (cf. diagramme de composants) à l'intérieur des nœuds. C'est un diagramme de type statique.

  Nous présentons ci-dessous le schéma structurel des diagrammes d'UML (Figure 15).

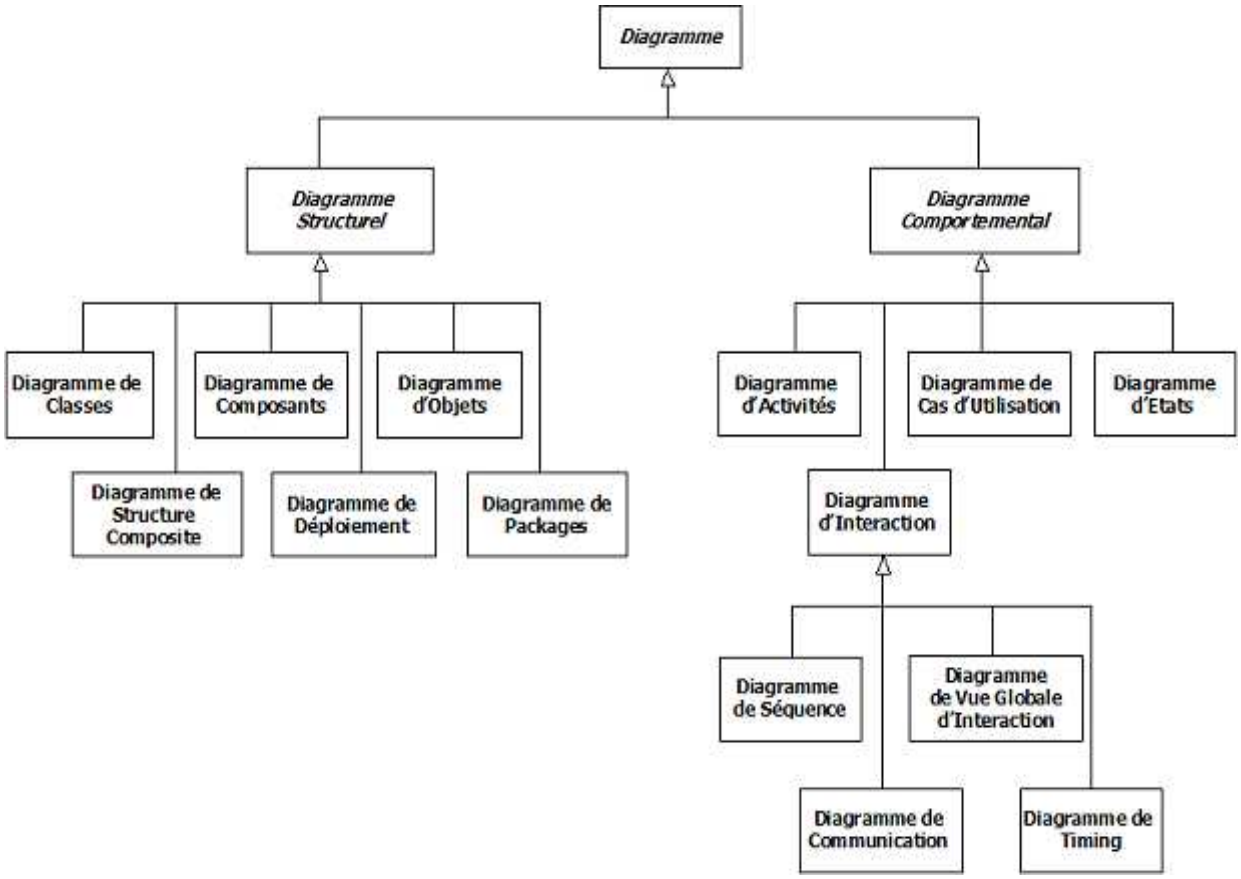

Figure 16: Schéma structurel des diagrammes d'UML

En effet, UML propose plusieurs profils **[Web20]** d'analyse en adéquation avec les besoins de ses utilisateurs. Ces profils d'UML sont :

- TelcoML est développé afin de modéliser les services avancés et intégrés de télécommunications.
- SoCP est destiné à l'analyse des systèmes sur puce

- MARTE est conçu pour l'analyse et la conception des systèmes embarqués à temps réel.

Ce dernier profil fera l'objet de l'analyse de notre système à implémenter.

2.2.2 UML profil MARTE

SysML (Systems Modeling Language) est un langage de modélisation des systèmes complexes (matériel, logiciel et contraintes physiques). En effet, SysML est un profil d'UML plus adapté prenant en compte les contraintes sophistiquées liées à des lois physiques **[Dav14]**. Un système embarqué à temps réel **[Rob13]** comprend le matériel, le logiciel et l'environnement. Une des limites de SysML réside sur son incapacité à représenter l'influence de l'environnement sur l'exécution du système. Ainsi, Edward A. Lee **[Web21]** qualifie les méthodes actuelles incapables de modéliser les systèmes embarqués et à temps réel dans son article **[Edw05]**. Ces derniers modèles représentent mal les notions de concurrence et de temps. Ainsi, en février 2005 OMG introduit UML profil MARTE. En effet, MARTE **[Mod11]** remplace UML profil SPT. MARTE fournit des modèles pour les étapes de spécification, de conception et de vérification/validation. Ainsi, MARTE permet de :

- Fournir une modélisation unifiée prenant en compte l'aspect matériel et logiciel du système
- Faciliter la construction de modèles contenant des informations quantitatives respectant les caractéristiques du matériel et du logiciel MARTE comprend :
    1. Un paquetage appelé MARTE Foundations regroupant l'ensemble des packages qui assurent les deux fonctionnalités : la conception (design) et l'analyse (analysis)
    2. Un autre paquetage appelé MARTE design model qui représente l'ensemble des sous-profils de conception
    3. Un troisième paquetage appelé MARTE Analysis model qui regroupe les sous-profils de l'analyse
- Enfin le dernier paquetage appelé MARTE Annexes qui définit l'ensemble des sous-profils utilitaires et sa bibliothèque utile

Ainsi, MARTE permet à UML de représenter les propriétés non fonctionnelles du système à concevoir. Ces propriétés peuvent être soit le temps de réponse, la consommation d'énergie, la taille de mémoire. La modélisation avec MARTE est basée sur les stéréotypes.

Nous présentons l'architecture du profil MARTE UML

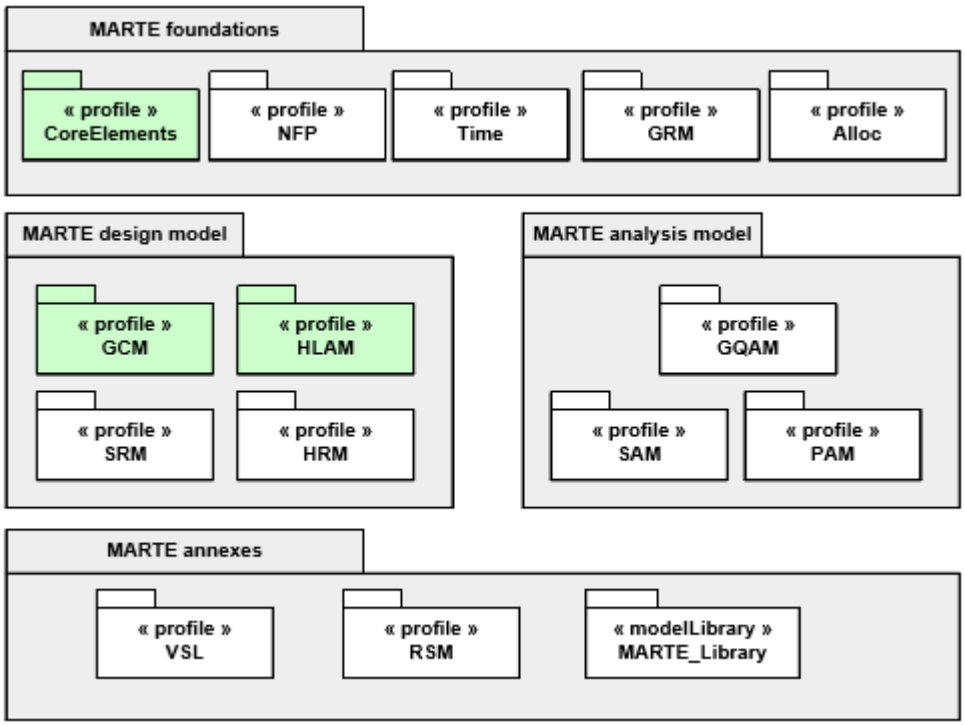

Figure 17 : Architecture du profil MARTE UML

Le package le plus haut appelé MARTE foundations comprend :

Le profil CoreElements définit les concepts qui caractérisent les modes de fonctionnement d'un système. Un mode de fonctionnement identifie le segment opérationnel dans l'exécution du système qui se caractérise par une configuration définie. Son CommonBehavior permet de décrire les éléments de base du comportement et d'exécution de système (comportement modal). En effet, le comportement modal est lié à un mode opérationnel qui est capable de gérer certains éléments :

- Un état d'un système opérationnel qui peut être géré par des mécanismes de reconfiguration et par des affectations de ressources afin qu'il puisse fournir des services de qualité.
- Un système opérationnel capable de gérer ses états (Démarrage, Arrêt, Lancement) et ses conditions de transitions

Le profil NFP (NonFunctional Properties) permet d'identifier les concepts dits propriétés non fonctionnelles. Ces propriétés représentent l'ensemble des caractéristiques quantitatives ou qualitatives des composants du système. Ce sont des mesures de constantes comme la température, la vitesse, la distance, des expressions mathématiques …

Le profil Time introduit des concepts de représentation du temps dans les modèles standard d'UML (diagrammes de classes, diagrammes d'états machines, …).

Le profil GRM (Generic Ressources Modeling) met en évidence la conception des composants (hardware et software) du système temps réel et embarqué

Le profil Alloc (Allocation Modeling) s'intéresse à l'allocation des ressources utilisées pendant l'exécution du système embarqué à temps réel

Ce package est divisé en deux sous packages appelé MARTE design model et MARTE analysis model afin d'apporter plus de précisions aux concepts utilisés dans le mode des systèmes embarqués.

Le sous-package MARTE design model se compose :

Le profil GCM (Generic Component Model) apporte les concepts additionnels utilisés dans la représentation des artefacts des systèmes embarqués à temps réel.

Le profil HLAM (High Level Application Model) définit le haut niveau des

Concepts utilisés dans les systèmes embarqués à temps réel

Le profil SRM (Software Resource Model) s'intéresse à la modélisation des interfaces des programmes de logiciels multitâches.

Le profil HRM (Hardware Resource Model) introduit les concepts de la modélisation de la partie matérielle des systèmes embarqués à temps réel

Le sous-package MARTE analysis model comprend :

Le profil GQAM (Generic Quantitative Application Model) représente les caractéristiques (temps, performance, disponibilité, …) des systèmes embarqués à temps réel

Le profil SAM (Schedulability Analysis Model) et le profil PAM (Performance Analysis Model) facilitent les annotations des concepts de performance et d'enchainement des tâches sur les modèles standards d'UML.

Enfin, le dernier sous-package MARTE annexes comprend le VSL (Values Expression Language), RSM package et sa bibliothèque. En effet, le VSL est le langage qui permet aux modèles de spécifier les propriétés, les contraintes des systèmes sous forme d'annotations.

En effet, MARTE ne s'intéresse qu'aux reconfigurations comportementales des systèmes embarqués temps-réel. Ces systèmes proposent de nombreuses reconfigurations possibles. Afin de modéliser les différentes reconfigurations, MARTE est obligé d'énumérer tous les modes et les transitions du système.

2.2.3 AADL

AADL est un langage de description d'architecture permettant de décrire l'architecture des systèmes embarqués à temps-réel. Il est un standard international publié par la SAE. AADL repose principalement sur la notion de 'composant'. Un composant représente toute partie

matérielle ou logicielle d'un système. Ainsi, AADL permet de décrire la partie matérielle et logicielle d'un système embarqué temps-réel. Il permet de décrire des systèmes embarqués temps réel en assemblant des blocs développés séparément. Pour cela, il utilise une syntaxe textuelle aussi bien qu'une représentation graphique. AADL permet aussi de spécifier les reconfigurations dynamiques des systèmes embarqués via des machines à états composées par des modes et des transitions entre eux. En effet, les événements nommés event ports définis au niveau des composants déclenchent toujours un changement de mode. La transition d'un mode à un autre nécessite les activations/désactivations de sous-composants et/ou de connexions. AADL s'intéresse aussi bien à la modélisation des reconfigurations comportementales qu'aux reconfigurations matérielles. Les reconfigurations du système sont représentées en AADL sous la forme d'énumération de modes de fonctionnement. Ainsi, le concepteur en AADL doit accomplir un travail fastidieux si son système propose de nombreuses reconfigurations.

2.2.4 Spécification PEARL et le profil UML-RT

La spécification PEARL permet d'assurer la programmation des applications temps réel automatiques. PEARL est étendue par des constructions permettant la description des configurations matérielles et logicielles. Elle s'appuie sur le mapping des modules logiciels sur des composants matériels. Elle introduit des attributs définissant des informations sur le temps permettant de spécifier des systèmes temps réel. En se basant sur les concepts de PEARL nait le profil UML-RT. En effet, UML-RT permettant d'assurer la modélisation des systèmes embarqués et la gestion des reconfigurations des architectures logicielles et matérielles. UML-RT utilise le diagramme de séquence pour modéliser les différentes reconfigurations et plus précisément les transitions d'une configuration à une autre. En UML, le gestionnaire de reconfiguration est modélisé selon les trois aspects suivants :

- L'aspect comportemental qui présente les modes du système et les conditions sous lesquelles le système change de mode,
- L'aspect structurel qui définit trois niveaux d'abstraction : le niveau station, le niveau configuration et le niveau collection,
- L'aspect fonctionnel qui fournit les fonctionnalités définissant les communications et les interfaces de l'application.

Afin de modéliser les reconfigurations par le profil UML-RT, des fonctionnalités de contrôle ont été implantées. Ces fonctionnalités assurent la vérification des contraintes d'entrées/sorties, des contraintes des états (configurations), des contraintes temporelles et le traitement des exceptions. Le profil UML-RT de la spécification PEARL permet de définir les reconfigurations dynamiques de ces systèmes par des transactions entre un ensemble de

configurations. Cependant, les concepts définis par la spécification restent limités. Ils ne permettent ni de spécifier des propriétés et des contraintes non-fonctionnelles de ces systèmes ni des contraintes temporelles complexes. De plus, la spécification des reconfigurations consiste à énumérer toutes les configurations possibles.

Dans la sous-section **3.3**, l'étude des algorithmes d'ordonnancement dans un système embarqué est primordiale pour la conception. En effet, ces algorithmes permettent au système d'assurer sa cohérence fonctionnelle.

2.3 Les algorithmes d'ordonnancement

Les algorithmes d'ordonnancement permettent de vérifier les contraintes temporelles (e.g. respect des échéances des tâches) et les contraintes de ressources (e.g. consommation CPU, utilisation de mémoire, ..). Un algorithme d'ordonnancement représente une politique d'exécution des tâches périodiques sur un processeur. Il aide à garantir l'exécution des tâches sur un processeur tout en assurant le respect de leurs échéances et à vérifier la consommation du processeur. Les politiques d'ordonnancement temps réel sont de deux types : l'ordonnancement par priorité statique et l'ordonnancement par priorité dynamique. Dans la suite, nous présentons un aperçu de ces deux types d'ordonnancement ainsi que les algorithmes les plus utilisés.

2.3.1 Ordonnancement par priorité statique

L'ordonnancement par priorité statique consiste à fixer les priorités des tâches avant de commencer leur exécution. Ces priorités seront préservées tout au long de l'exécution. Parmi les algorithmes d'ordonnancement par priorité statique, nous citons les deux algorithmes RMS et DMS.

Pour RMS, la tâche ayant la période la plus petite a la priorité la plus grande. Donc, les priorités des différentes tâches seront fixées dès le départ.

Dans RMS, l'ordonnancement des tâches sur un processeur est garanti si la **condition 1** est vérifiée c'est à dire si le taux d'utilisation du processeur est inférieur à un seuil défini en fonction du nombre des tâches périodiques. Cette condition est ci-dessous. Donc les tâches peuvent parfois être ordonnancées même si la condition ci-dessous n'est pas satisfaite.

Formule 13 : La première condition à vérifier pour l'ordonnancement des tâches

$$\sum_{i=0}^{n}(C_i/P_i) \leq n(2^{1/n} - 1) \quad 1$$

n : nombre de tâches périodiques,

Ci : pire temps d'exécution de la tâche i,

Pi : période de la tâche i.

Pour DMS, la tâche ayant l'échéance la plus petite a la priorité la plus grande. Il a la même condition d'ordonnançabilité que celui de RMS.

2.3.2 Ordonnancement par priorité dynamique

L'ordonnancement par priorité dynamique consiste à modifier les priorités des tâches durant leur exécution. Parmi les algorithmes d'ordonnancement par priorité dynamique, nous citons les deux algorithmes EDF et LLF.

Pour EDF, la tâche ayant l'échéance la plus proche est la tâche la plus prioritaire. Les priorités des tâches changent généralement au cours de l'exécution du système puisque les temps les séparant de leurs échéances peuvent changer.

Dans EDF, les tâches ne sont ordonnançables que si la **condition 2** est vérifiée : le processeur ne doit pas être surchargé.

Formule 14 : La deuxième condition à vérifier pour l'ordonnancement des tâches

$$\sum_{i=0}^{n}(C_i/P_i) \leq 1 \qquad 2$$

n : nombre de tâches périodiques,

Ci : pire temps d'exécution de la tâche i,

Pi : période de la tâche i.

Pour LLF, la tâche ayant la marge (laxity) la plus petite est la plus prioritaire. La marge L est définie par la formule ci-dessous.

Formule 15 : Calcul de la marge de tâche

$$L(t) = D - t - T_r(t) \qquad 3$$

L : marge de la tâche

D : échéance de la tâche

Tr : temps d'exécution restant de la tâche

t : temps courant du système

Dans LLF, la condition d'ordonnançabilité est la même que celle de EDF.

Ensuite, nous présentons des frameworks conçus pour assurer la vérification de certaines propriétés non-fonctionnelles au niveau modèle de conception.

2.4 Frameworks de vérification

Plusieurs frameworks assurant la vérification à base de modèles pour des systèmes embarqués ont été proposés. Dans la suite, nous présentons des frameworks assurant la vérification des propriétés temporelles et/ou des propriétés liées aux ressources.

2.4.1 Cheddar

Cheddar **[Sin04]** est un framework écrit en Ada et fournissant des outils pour vérifier des critères de performance. Il vérifie, par exemple, si une application temps réel respecte ses contraintes temporelles. Cheddar assure les tests de faisabilité et la simulation des systèmes mono-processeurs ou multi-processeurs. Les tests de faisabilité consistent à comparer le facteur d'utilisation d'un processeur avec une borne donnée, comparer le pire temps d'exécution de chaque tâche avec son échéance, trouver une borne pour le facteur d'utilisation du tampon dans le cas d'un tampon partagé entre des tâches périodiques ordonnancées par un ordonnanceur de priorité fixe ou de priorité dynamique.

La simulation consiste à simuler l'exécution du système et à effectuer l'analyse d'ordonnancement pour vérifier certaines propriétés. A partir de l'analyse d'ordonnancement, ce framework permet d'obtenir les temps de réponse des tâches, les temps de blocage sur des ressources partagées, le nombre de préemptions et les échéances non respectées. De plus, il permet de détecter les inter-blocages et les inversions de priorité. En effet, ce framework supporte la plupart des algorithmes d'ordonnancement (RMS, DMS, EDF et LLF). Cheddar est un framework ouvert et flexible. En effet, il interagit facilement avec des outils CASE par l'envoi et la réception des fichiers XML. De plus, il peut être étendu afin d'exécuter des ordonnancements spécifiques, faire des analyses spécifiques, ou ordonnancer des tâches avec des modèles particuliers d'activation. Il est donc facile d'écrire et de tester des extensions sur ce framework sans une connaissance approfondie du langage Ada. Cependant, Cheddar vérifie certaines propriétés non-fonctionnelles pour des systèmes ayant seulement des tâches périodiques. Il ne vérifie pas les systèmes ayant des tâches sporadiques et/ou apériodiques. Cheddar ne permet pas non plus de vérifier certaines propriétés liées aux ressources comme la consommation mémoire et la consommation de la bande passante.

2.4.2 TASM Toolset

TASM Toolset implante les caractéristiques du langage TASM à travers trois composants principaux :

- Un éditeur permettant l'édition du texte de base et la représentation syntaxique des fonctionnalités en utilisant le langage TASM.
- Un simulateur permettant la visualisation du comportement dynamique des spécifications TASM y compris les dépendances du temps et la consommation des ressources.
- Un analyseur assurant la vérification des propriétés de base des spécifications TASM comme la consistance.

Le langage TASM vise à capturer trois aspects du comportement d'un système temps réel : le comportement fonctionnel, le comportement temporel, et la consommation d'énergie. Afin de vérifier les propriétés temporelles, TASM Toolset intègre le vérificateur de modèle UPPAAL. Pour cela, il effectue une traduction de la spécification TASM vers les automates temporisés. En vue de vérifier la consistance de la spécification, TASM Toolset intègre le solveur SAT4J SAT pour vérifier une formule booléenne obtenue à partir des règles sur des machines à états. Cependant, TASM Toolset ne permet pas de vérifier les propriétés liées aux ressources comme la consommation CPU, la consommation mémoire et la consommation de la bande passante. En outre, la spécification en utilisant le langage TASM est compliquée et elle exige des compétences supplémentaires.

2.4.3 ModSyn

ModSyn est un framework basé sur l'IDM assurant la vérification formelle des systèmes embarqués. Il permet de vérifier des propriétés logiques fonctionnelles et temporelles et de vérifier aussi l'exécution correcte de ces systèmes. Ce framework permet de générer automatiquement un réseau d'automates temporisés à partir des modèles UML présentant la spécification fonctionnelle d'une application embarquée par un diagramme de classes et des diagrammes de séquence. En effet, les modèles UML sont transformés vers un modèle commun d'application défini par le méta-modèle IAMM proposé. Ensuite, ce modèle commun d'application est transformé vers un modèle de réseau d'automates temporisés conforme au méta-modèle LTAMM (Label Timed Automata Meta-model) proposé. La transformation entre les modèles est implantée avec le langage Xtend du framework Open ArchitectureWare. Afin de vérifier certaines propriétés temporelles et fonctionnelles du système, une représentation du réseau d'automates temporisés est obtenue automatiquement comme une entrée pour des outils de vérification formelle comme UPPAAL.

Après la modélisation des systèmes embarqués à temps-réel dynamiquement reconfigurables et la vérification de certaines propriétés non-fonctionnelles au niveau modèle de conception, une phase de génération de code est demandée afin de concevoir facilement ces systèmes.

2.4.4 VERTAF

VERTAF est un framework supportant la vérification des systèmes embarqués temps réel. Il intègre trois techniques : la réutilisation basée composant, la synthèse formelle (ordonnancement, génération de code), et la vérification formelle. Il assure la génération de code à partir des modèles UML. La phase de conception utilise trois diagrammes UML: le diagramme de classes, le diagramme d'états-transitions et le diagramme de séquence. Afin d'assurer l'ordonnancement, les diagrammes d'UML sont traduits vers des réseaux de Petri de

type RTPN ou CCPN. A partir des diagrammes de séquence, les modèles RTPN et CCPN sont automatiquement générés. Pour vérifier l'ordonnancement, VERTAF a recours à deux algorithmes d'ordonnancement :

- Si le système est temps réel, un ordonnancement quasi dynamique (QDS) est appliqué sur des modèles RTPN spécifiant le système. Il prépare le système pour être généré comme un seul noyau exécutif temps réel avec un ordonnanceur.
- Si le système n'est pas temps réel, un ordonnancement quasi statique étendu (EQSS) est appliqué sur des modèles CCPN spécifiant le système. Il prépare le système pour être généré comme un ensemble de tâches qui peuvent être ordonnancées et envoyées par un RTOS comme MicroC/II OS ou Linux ARM.

Pour assurer la vérification formelle, des automates temporisés étendus (ETAs) seront générés à partir des diagrammes d'états-transitions. Afin de vérifier certaines propriétés, ces automates sont des entrées pour le vérificateur de modèles SGM. VERTAF vérifie deux classes de propriétés : des propriétés du système comme l'absence d'interblocage et la vivacité et des propriétés définies par l'utilisateur à partir de la spécification UML en utilisant le langage déclaratif OCL. En effet VERTAF n'assure pas la vérification des propriétés liées aux ressources comme par exemple la consommation mémoire, la consommation CPU ou la consommation de la bande passante car il ne s'appuie que sur des simples diagrammes d'UML.
Dans la suite, nous décrivons quelques intergiciels dédiés aux systèmes embarqués.

2.5 Intergiciels dédiés aux systèmes embarqués

Face à la complexité croissante des systèmes embarqués, des intergiciels offrant un ensemble de fonctionnalités pour faciliter leur développement sont disponibles. Dans ce cadre, plusieurs intergiciels ont été proposés pour couvrir les besoins des systèmes embarqués. Dans cette section, nous présentons ceux qui sont les plus proches de notre approche.

2.5.1 DynamicTAO

Dynamic TAO est une extension de l'intergiciel TAO [Dou01] qui est un ORB reconfigurable, portable, flexible, extensible et basé sur les patrons de modélisation orientés objet. En effet, TAO utilise la stratégie de modélisation par des patrons afin de séparer les différents aspects du moteur interne de l'ORB. Il est dédié aux applications temps réel statiques. Dynamic TAO a été introduit pour supporter des applications adaptatives qui s'exécutent sur des environnements dynamiques. Il repose sur un ORB conforme à CORBA. Il implante les mécanismes de reconfiguration dynamique pour des systèmes répartis et performants. Il

implante aussi les mécanismes de concurrence, de sécurité et de supervision au cours de l'exécution. Dynamic TAO est un ORB réflexif. En effet, il permet la supervision et la reconfiguration de son moteur interne. Il assure la réflexivité par l'exportation d'une interface permettant de transférer des composants à travers des systèmes répartis, charger/décharger des modules dans l'ORB au cours de l'exécution, superviser et modifier l'état de sa configuration. Afin d'assurer facilement la reconfiguration dynamique, les composants du système sont regroupés dans des bibliothèques chargées et utilisées au cours de l'exécution. Dans dynamic TAO, la réflexivité est assurée à travers un ensemble d'entités appelées composants configurateurs. Pour chaque processus en cours d'exécution sur dynamic TAO, une instance du composant configurateur est créée. Dynamic TAO préserve la cohérence du système lors du déchargement des composants. Il vérifie que toutes les invocations du composant sont traitées avant le déchargement et il assure la préservation de l'état du composant par l'envoi d'une partie de l'état interne du composant déchargé vers un nouveau composant. Dynamic TAO assure aussi la sécurité lors des reconfigurations dynamiques. Pour cela, il offre une architecture de sécurité flexible par l'injection d'agents assurant la supervision et la reconfiguration. Il assure la reconfiguration dynamique ainsi que la supervision et la cohérence de ces systèmes. Il peut alors facilement s'adapter à l'environnement d'exécution du système. Cependant, DynamicTAO ne supporte que des reconfigurations architecturales. De plus, DynamicTAO n'est pas bien adapté pour des systèmes temps réel durs (critiques). Il n'assure pas la vérification des contraintes temporelles pendant et après des reconfigurations.

2.5.2 CIAO

CIAO [Ven07] est un intergiciel supportant des systèmes TR2E basés sur les composants. Il propose une implantation libre du modèle de composants LwCCM et de la spécification RT-CORBA. Il offre des mécanismes permettant la spécification, l'implantation, l'assemblage et le déploiement des composants. De plus, CIAO supporte les contraintes additionnelles sur les temps d'initialisation du système et les fonctions disponibles (comme par exemple la liaison et le chargement dynamique). En effet, un framework de déploiement et de configuration a été intégré dans l'intergiciel CIAO afin d'assurer la gestion du déploiement et de la configuration des composants de QoS ainsi que des services de l'intergiciel. Par ailleurs, CIAO assure une robuste QoS grâce à la séparation de l'implantation des composants du déploiement et de la configuration. En effet, il a recours aux techniques à base d'aspects afin d'assurer la séparation et la composition des aspects temps réel et des préoccupations propres à la configuration. Afin d'assurer les défis de performance et de faisabilité de déploiement et de reconfiguration du système, CIAO étend la définition du conteneur d'un composant et les capacités de représentation et de manipulation des méta-données du composant. Il permet de simplifier et

d'automatiser le (re)déploiement et la (re)configuration des systèmes embarqués dynamiquement reconfigurables. Cependant, CIAO ne prend pas en charge l'ordonnancement des tâches apériodiques, ni la nature des mécanismes de reconfigurations des systèmes. CIAO ne supporte que le modèle de composants LwCCM

2.5.3 SwapCIAO

SwapCIAO **[Jai05]** est une extension de l'intergiciel CIAO visant à supporter la reconfiguration dynamique des systèmes embarqués temps réel dynamiquement reconfigurables. Il permet la mise à jour dynamique des implantations de composants grâce aux extensions fournies par le modèle de composants LwCCM. La reconfiguration dynamique dans SwapCIAO consiste à faire une mise à jour des implantations de composants. Elle se fait selon les étapes suivantes : la désactivation de l'implantation du composant, le retrait de l'implantation du composant de la plate-forme d'exécution et la modification de l'implantation du composant. En effet, SwapCIAO assure :

- des mises à jour continues et consistantes en s'assurant que le composant à reconfigurer a terminé les invocations en cours et que les nouvelles invocations sont bloquées jusqu'à la fin de la mise à jour du composant.
- la transparence des mises à jour en garantissant la redirection des nouvelles invocations vers la nouvelle implantation du composant.
- la reconnexion des composants par la restauration des connexions après la mise à jour de l'implantation du composant. Ces connexions sont déjà stockées dans des descripteurs XML lors du déploiement de l'application.

Ainsi, SwapCIAO est un intergiciel flexible, performant et adapté aux systèmes embarqués dynamiquement reconfigurables. Cependant, la maintenance de SwapCIAO est difficile. Ceci peut expliquer son non-implémentation.

2.5.4 FLARe

FLARe est un intergiciel qui étend TAO. Il supporte les applications réparties temps réel molles. Il permet aussi la gestion de la tolérance aux pannes et le recouvrement des exigences des systèmes temps réel mous. FLARe assure la reconfiguration dynamique selon la disponibilité des ressources. Il offre des mécanismes de reconfiguration qui consistent en la redirection des clients en cas de panne (surcharge de processeur). Afin de fournir des mécanismes de tolérance aux pannes, FLARe utilise la réplication passive qui consiste à traiter toutes les requêtes par un seul serveur appelé primaire. Lorsque ce dernier tombe en panne, il est remplacé par un autre serveur

Cependant, il ne supporte pas les systèmes temps réel durs. De plus, il ne gère que les tâches périodiques et sa maintenance est très complexe.

2.5.5 PolyORB_HI

PolyORB_HI (PolyORB-High Integrity) **[Tho04]** est un intergiciel permettant de générer du code à partir des modèles AADL pour les systèmes embarquées à temps -réel. Il est inspiré de l'architecture de PolyORB et développé dans le cadre des recherches de TELECOM Paris. Il utilise l'architecture schizophrène et ses services canoniques afin d'assurer la communication entre plusieurs plates-formes hétérogènes. PolyORB_HI est conforme au Profil Ravenscar **[Jag02]**. Il respecte les restrictions proposées par ce profil qui restreint l'usage de certaines constructions des langages de programmation (ADA, C, etc.) afin d'assurer l'ordonnancement du système ainsi que l'absence d'interblocage. Par ailleurs, il n'implante pas les mécanismes assurant la reconfiguration dynamique des systèmes TR2E.

Grâce à sa faible empreinte mémoire, PolyORB_HI représente un intergiciel idéal pour les systèmes embarqués. En effet, PolyORB_HI a une empreinte mémoire estimée à ≈ 70KB.

Dans la suite, nous présentons quelques méthodologies et processus de développement des systèmes embarqués.

2.6 Processus et frameworks de développement

Pour faire face à la complexité croissante de la conception et du développement des systèmes embarqués, de nouveaux processus de développement ont été proposés. Ces processus offrent des méthodologies permettant de spécifier ces systèmes en prenant en considération leurs contraintes non-fonctionnelles. Ces méthodologies permettent d'une part de diminuer la complexité de la mise en œuvre de ces systèmes et d'autre part de réduire le coût de développement. Dans la suite, nous décrivons des processus de développement et frameworks proposés pour la spécification et le développement des systèmes embarqués.

2.6.1 ModES

ModES est une approche basée sur l'IDM. Elle présente une méthodologie et offre un ensemble d'outils de conception, d'estimation et de génération de code permettant de concevoir des systèmes embarqués. Cette méthodologie consiste, en une première phase, à transformer des modèles d'application et des modèles de plateforme spécifiés par des langages de modélisation (UML) en des modèles conformes respectivement au méta-modèle d'application interne IAMM et au méta-modèle de plate-forme interne IPMM. Le mapping entre ces derniers modèles (l'allocation de l'application sur la plate-forme), qui est conforme au méta-modèle de mapping

MMM, est assuré dans une deuxième phase. Dans une troisième phase, les modèles sont transformés en des modèles d'implantation conformes à un métamodèle d'implantation IMM. Finalement, une partie du code source, des scripts de synthèse matérielle, et des scripts de déploiement du système sont générés à partir des modèles d'implantation. Des outils d'analyse sont aussi utilisés afin de fournir des estimations sur les propriétés physiques du système (e.g. cycles d'exécution, consommation d'énergie, empreinte mémoire, etc.) au niveau modèle de mapping. ModES définit donc quatre nouveaux méta-modèles : un méta-modèle IAMM pour la spécification haut niveau de l'application, un méta-modèle IPMM pour la spécification de la plate-forme d'exécution, un méta-modèle de mapping présentant les règles d'allocation de l'application sur la plate-forme, et un méta-modèle d'implantation permettant la génération de code. L'évaluation des implantations possibles au cours du processus de conception en se basant sur des estimations précises de chaque séquence de transformation est assurée au niveau modèle de mapping. La transformation d'un modèle vers un autre est définie en utilisant le langage de transformation de modèles QVT qui est un standard de l'OMG. En effet, ModES ne propose pas d'intergiciel permettant la génération de code à partir des modèles. Il ne propose qu'un processus de développement des systèmes embarqués statiques.

2.6.2 Framework dirigé par les modèles

Ce framework basé sur les architectures dirigées par les modèles a été proposée afin de concevoir des systèmes embarqués temps réel. Dans cette approche, un modèle indépendant de la plateforme (PIM) est modélisé en utilisant le langage UML et le sous-profil HLAM du profil MARTE. Le modèle PIM est transformé en un modèle spécifique à la plate-forme (PSM) par des règles de transformation. Pour chaque plate-forme cible, pour obtenir le modèle PSM à partir du modèle PIM, il faut avoir des règles de transformation spécifiques. L'apport de cette approche est qu'elle offre des règles de transformation génériques permettant d'obtenir un modèle PSM à partir d'un modèle PIM. Ces règles de transformation permettent d'obtenir différents PSM à partir d'un PIM spécifié en utilisant les deux sous-profils HLAM et SRM du profil MARTE. Enfin, les générateurs de code permettent d'obtenir le code du système attendu à partir du PSM.

Cependant, cette approche ne prend pas en considération les reconfigurations dynamiques dans les systèmes embarqués temps réel. En outre, la modélisation du PIM repose seulement sur les deux sous-profils HLAM et SRM du profil MARTE.

2.6.3 CoSMIC

CoSMIC **[Chm03]** est une suite d'outils proposés pour la composition et le déploiement des applications embarquées à temps réel. Cette suite d'outils permet tout d'abord la modélisation et l'analyse des fonctionnalités des applications embarquées temps-réel reconfigurables et les exigences de qualité de service (QoS). Afin d'offrir et appliquer la qualité de service, elle permet de générer les méta-données de déploiement spécifiques au CCM **[Com]** pour les intergiciels CIAO et QuO. Un processus de développement des applications embarquées temps-réel reconfigurables basé sur la suite d'outils CoSMIC est proposé afin de modéliser les exigences et les politiques d'adaptation nécessaires pour la gestion de la QoS de ces applications. Ce processus permet l'assemblage des composants logiciels de l'intergiciel en assurant une compatibilité entre leurs différents paramètres de qualité de service. Après la modélisation de l'assemblage et du déploiement des composants en utilisant le langage CADML, des plans de déploiement sont générés. Ces plans de déploiement décrivent les différentes possibilités de déploiement pour une telle application. CoSMIC utilise aussi un langage spécifique OCML pour la configuration des paramètres et des contraintes. Il génère les méta-données de configuration d'intergiciels des applications. Cependant, CoSMIC utilise différents langages pour la configuration et le déploiement des composants des applications embarquées à temps-réel reconfigurable ainsi que l'application et le middleware (CADML, OCML). Le concepteur doit donc avoir des compétences sur ces langages afin de modéliser correctement son application. De plus, CoSMIC ne prend pas en considération les reconfigurations dynamiques dans les applications à temps-réels. En outre, CoSMIC repose sur l'intergiciel CIAO qui ne supporte pas les spécificités des systèmes critiques. En effet, CIAO se base sur l'intergiciel TAO qui assure une allocation dynamique de la mémoire. De plus, CIAO ne supporte que le modèle de composant CCM.

2.6.4 Ocarina

Ocarina **[Las09]** est un ensemble d'outils permettant de concevoir des systèmes embarqués temps réel répartis. Il offre un framework assurant le développement, la configuration et le déploiement des systèmes embarqués à temps-réels. Ce framework consiste, tout d'abord, à modéliser des systèmes embarqués à temps-réels statiques en utilisant le langage de description d'architecture AADL. A partir des modèles AADL et comme le montre la figure ci-dessous, Ocarina assure les analyses lexicales, syntaxiques et sémantiques afin de garantir la conformité des modèles à la grammaire AADL. Si les analyses sont faites avec succès, une instanciation du système est nécessaire. Grâce à son organisation en bibliothèques logicielles, Ocarina permet aussi de faire manipuler des modèles AADL par d'autres outils comme par

exemple l'outil d'analyse statique d'ordonnancement Cheddar. A partir des modèles AADL, les générateurs de code d'Ocarina produisent une grande partie du code applicatif ainsi qu'une couche de l'intergiciel consacrée à des besoins spécifiques à l'application. Cette dernière rassemble les services canoniques fortement personnalisés pour l'application. Ocarina permet de générer des constructions des langages de programmation Ada et C qui sont conformes aux restrictions du profil Ravenscar à partir des entités AADL. Pour cela, deux versions de l'intergiciel PolyORB_HI ont été proposées, une en Ada et l'autre en C. L'intergiciel PolyORB_HI présente la couche minimale qui implante les services canoniques faiblement personnalisables par l'application. Une autre couche de l'intergiciel présentant les services canoniques fortement personnalisables est générée avec le code applicatif de l'application. Ocarina assure automatiquement le déploiement et la configuration de l'application à l'aide des informations extraites à partir des modèles AADL. Cependant, le framework Ocarina traite seulement les systèmes embarqués à temps réels statiques et ne prend pas en considération les systèmes dynamiquement reconfigurables. De plus, l'utilisation du langage AADL pour la modélisation des systèmes embarqués à temps réels fournit une modélisation à un niveau concret (threads, processeurs, etc.) et pas à un haut niveau d'abstraction. La figure 18 ci-dessous illustre l'architecture globale du framework Ocarina.

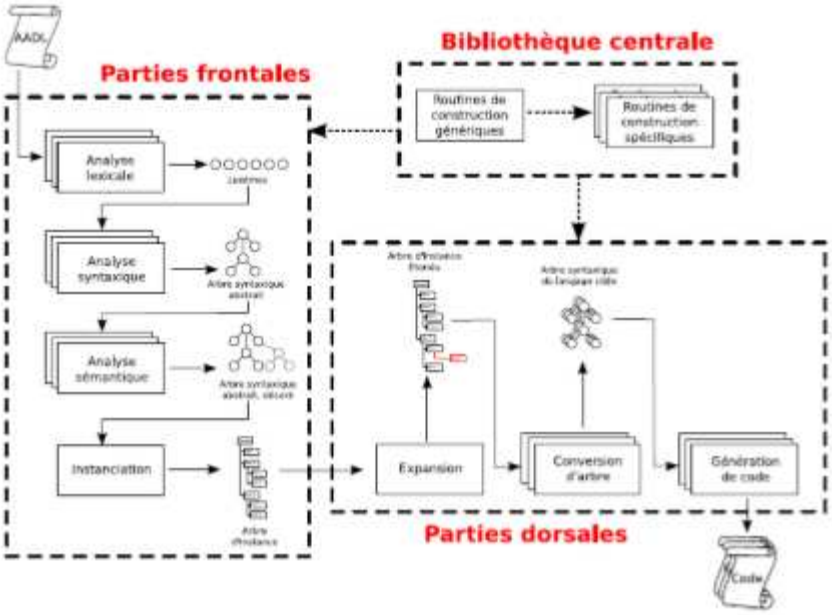

Figure 18 : Architecture globale d'Ocarina

2.6.5 TimeAdapt

TimeAdapt est un framework permettant d'assurer des reconfigurations dynamiques pour des systèmes embarqués. Il repose sur trois phases. La première phase consiste à fournir des moyens permettant la spécification de l'ensemble des actions de reconfiguration à effectuer. La

deuxième phase permet l'estimation du temps d'exécution de ces actions. Et la troisième phase assure l'exécution des actions de reconfiguration dynamique. En effet, ce framework possède un gestionnaire de reconfiguration. Ce dernier est responsable de la réception des actions de reconfiguration spécifiées en utilisant un langage de reconfiguration permettant d'exprimer les contraintes de reconfiguration. Chaque reconfiguration doit respecter ses contraintes temporelles (limites de temps) afin d'être exécutée. En effet, le temps d'exécution de chaque reconfiguration dépend des conditions environnementales et structurelles de l'application.

Pour cela, un test d'admission se charge de calculer la probabilité pour qu'une reconfiguration donnée respecte ses contraintes temporelles. Si la valeur de la probabilité calculée dépasse un seuil donné, la reconfiguration sera exécutée après la génération des constructions de reconfiguration en langage d'exécution. Elle sera considérée comme une tâche temps réel de haute priorité et les actions de reconfiguration seront donc exécutées. Sinon (la valeur de la probabilité calculée ne dépasse pas un seuil donné), les contraintes temporelles de la reconfiguration doivent être modifiées afin d'exécuter la reconfiguration plus tard. En effet, TimeAdapt supporte le modèle de composant UML2.

En outre, le langage utilisé pour spécifier les actions et les contraintes de reconfiguration a des inconvénients. D'une part, les utilisateurs de TimeAdapt doivent donc bien maîtriser ce langage afin d'utiliser ce framework. D'autre part, ce langage ne permet pas de spécifier les parties logicielles et matérielles des systèmes embarqués ainsi que l'allocation de la partie logicielle sur la partie matérielle de chaque système.

2.7 Conclusion

Ainsi, MARTE est le profil d'UML qui permet d'élaborer la conception des systèmes embarqués en représentant ses différents aspects matériels, logiciels et temporels. Une limite fondamentale de MARTE est la conception de la reconfiguration matérielle de ces systèmes complexes. En effet, la reconfiguration d'un système embarqué met en évidence l'existence des différents modes de fonctionnement, des transitions répondant aux exigences (humaines, temporelles, non-fonctionnelles). Cependant, AADL est le langage de description des systèmes embarqués temps-réel s'appuyant sur les concepts :

- Composant qui représente toute entité matérielle ou logicielle
- Mode qui constitue chaque configuration possible du système
- Transition qui identifie les techniques de passage d'une configuration à une autre

Enfin, PEARL s'appuie sur le profil UML-RT afin de modéliser les systèmes embarqués à temps-réel. UML-RT propose des concepts définissant les contraintes temporelles, non fonctionnelles.

En effet, le concepteur qui utilise l'un de ces méthodes de modélisation devra énumérer l'ensemble des configurations possibles du système à concevoir.

Le concepteur devra choisir son algorithme d'ordonnancement des tâches du système. L'algorithme d'ordonnancement permet d'assurer le bon fonctionnement avec le respect des échéances des tâches et celui de l'utilisation des ressources (processeur, mémoire, …). Ainsi, la conception des systèmes embarqués temps-réel est un travail fastidieux **[Jur06] [And08]**.

Par ailleurs, la plupart des travaux existants permettent de vérifier des propriétés non-fonctionnelles pour des systèmes statiques et n'assurent pas la vérification de toutes les propriétés que nous visons.

TASM Toolset permet de vérifier des propriétés liées aux ressources (consommation mémoire, consommation CPU et consommation de la bande passante) ainsi que l'absence d'interblocage en utilisant le langage TASM. Mais il ne permet pas de vérifier le respect des échéances et l'absence de famine. Le respect des échéances peut être vérifié par les automates temporisés ou par les algorithmes d'ordonnancement. Nous pouvons vérifier seulement le respect des échéances des tâches non préemptives en utilisant les automates temporisés.

L'absence d'interblocage et l'absence de famine sont, quant à elles, généralement vérifiées en utilisant les automates temporisés et les vérificateurs de modèles. Par contre, les automates temporisés sont limités par le problème d'explosion d'états et nécessitent des modèles avec un nombre restreint d'états.

Dans la suite, nous décrivons quelques intergiciels dédiés aux systèmes embarqués. Ces intergiciels facilitent la génération de code pour la conception des systèmes embarqués. En effet, un processus de développement des systèmes embarqués temps-réel dynamiquement reconfigurable fait l'objet d'étude complète dans la suite de notre document.

Deuxième partie: contributions

Chapitre 3: Processus de développement des systèmes embarqués dynamiquement reconfigurables

3.1 Introduction

Dans ce chapitre, nous proposons des solutions qui permettent de faciliter la mise en œuvre des systèmes embarqués à temps réel dynamiquement reconfigurables. Ces solutions introduisent :

- Des concepts de modélisation des composants matériels, logiciels du système,
- Un langage de modélisation des contraintes structurelles, non fonctionnelles, d'allocation,
- Un framework de vérification de ces contraintes,
- Un intergiciel dédié aux systèmes embarqués pour la génération de code,
- Un framework de développement de systèmes embarqués.

Ainsi dans la section **3.2**, nous présentons un processus de développement des systèmes embarqués qui permet d'assurer la modélisation des composants matériels et logiciels, la vérification des contraintes non fonctionnelles et enfin la génération de code.

La section **3.3** propose de nouveaux concepts de modélisation qui permettent de spécifier les propriétés logicielles, matérielles du système, leurs contraintes structurelles, non-fonctionnelles et d'allocation.

3.2 Description de notre Processus de développement

Ce processus de développement décrit dans la figure ci-dessous prend en compte les contraintes structurelles, celles non-fonctionnelles, celles d'allocation, celles d'exigence de délai.

En effet, il s'appuie sur une démarche bien précise. Cette démarche est la suivante :

1. La phase de modélisation qui comprend

- La spécification des reconfigurations possibles du système : identifie les tâches pouvant être accomplies par le système. En effet, chaque tâche requiert l'exécution d'une partie logicielle par l'allocation d'une partie matérielle
- La spécification de la partie logicielle : met en évidence l'ensemble des composants qui constituent cette sous-entité du système
- La spécification de la partie matérielle : s'intéresse aux composants matériels du système tel que bus, mémoire, processeur, microcontrôleur, …

- La spécification de l'allocation logiciel/matériel : clarifie l'allocation de chaque configuration logicielle sur une architecture matérielle correspondante

2. La phase de vérification, de validation des modèles : décrit la vérification de la prise en compte des contraintes non-fonctionnelles, des contraintes de délais sur les modèles
3. La phase de pré-génération : décrit les politiques (règles à respecter) de génération du code du système à partir des modèles de haut niveau obtenus
4. La phase de génération de code : s'oriente sur la génération automatique de code à partir des modèles obtenus
5. L'étude de performance : s'intéresse aux critères de performance de notre système (temps de réponse des tâches, Nombre moyen de tâches en attente, …)

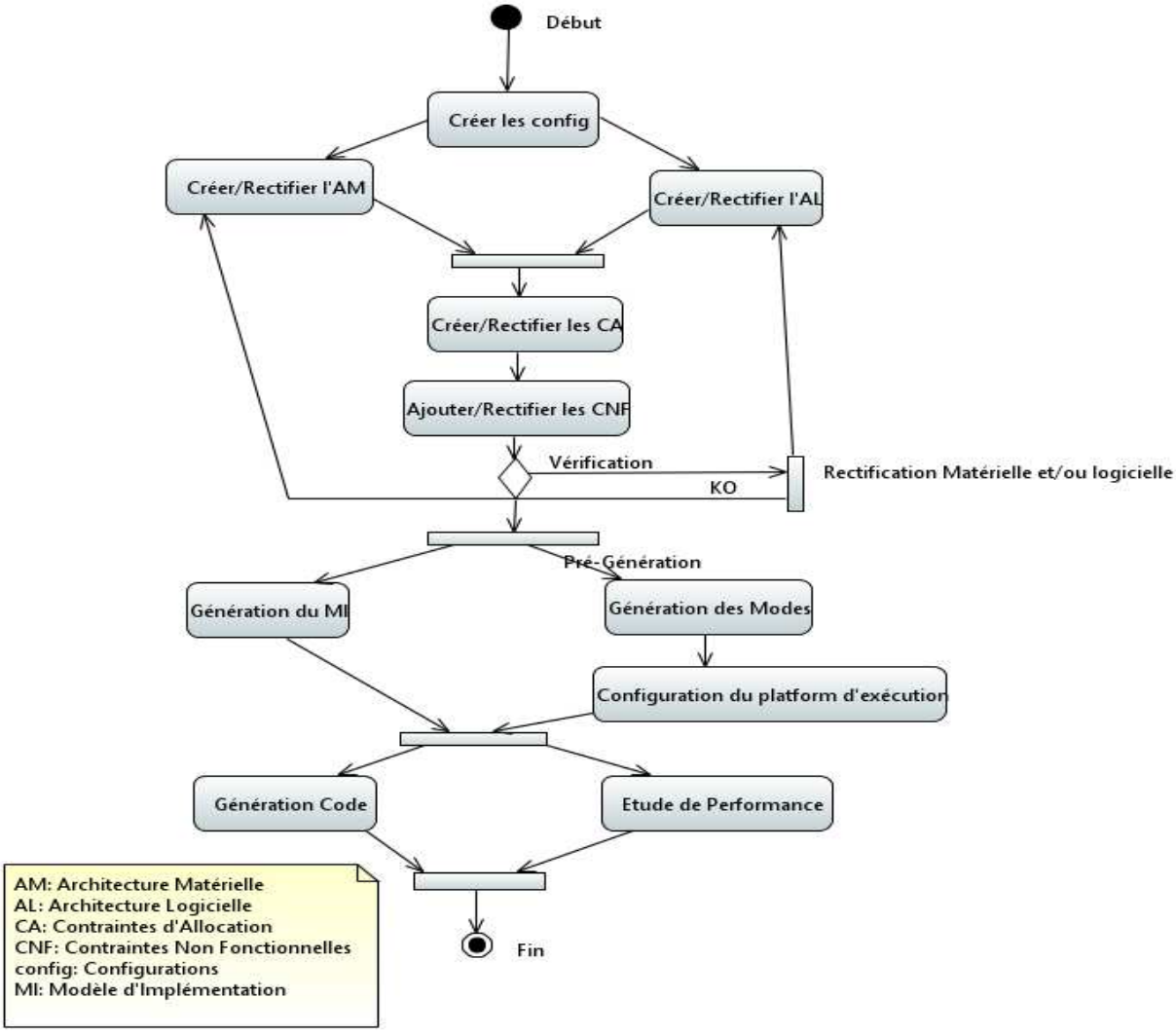

Figure 19 : Processus de Développement des systèmes embarqués temps réel dynamiquement reconfigurables

3.3 Concepts de modélisation

En effet, ce processus de développement s'appuie sur des concepts de modèle, méta-modèle et de transformation de modèles afin de concevoir les systèmes embarqués temps-réel dynamiquement reconfigurables.

Ces concepts s'appuient sur les méta-modèles déjà existants :

- MARTE : nous utilisons ses concepts afin de définir les architectures matérielles, logicielles, les contraintes structurelles, non-fonctionnelles, d'allocation
- UML : ses machines à états permettent de représenter les modes (configurations) et les relations entre elles (reconfigurations) du système

Notre processus intègre de nouvelles méta-modèles :

Le premier méta-modèle permet de regrouper les modes du système à concevoir en entité plus globale selon certains critères. Ce méta-modèle est appelé MetaMode

Le dernier est intégré dans l'implémentation

3.3.1 MetaMode, MetaTransition, Mode, Transition

En effet, un système embarqué temps réel dynamiquement reconfigurable présente plusieurs fonctionnalités. Chaque fonctionnalité constitue un mode de fonctionnement du système appelée Mode. Chaque Mode sollicite une architecture logicielle exécutée sur un support matériel. Cette exécution par le support matériel constitue l'allocation logicielle sur l'architecture matérielle.

En effet, le MetaMode est un concept qui permet de caractériser les différentes configurations d'un système en évitant leur énumération. Un MetaMode comprend un ensemble de composants structurés liés par des connexions potentielles et des contraintes structurelles et non fonctionnelles à respecter .Un MetaMode comprend plusieurs modes du système. Tous ces modes se trouvant dans un même MetaMode s'exécutent dans une architecture matérielle commune. Ces modes respectent la structure et les contraintes de leur MetaMode. La relation qui permet de passer d'un mode à un autre est appelé Transition. La relation entre MetaModes est appelé MetaTransition. Ainsi, la figure ci-dessous (diagramme de classes) illustre parfaitement les relations existantes entre MetaMode, Mode, MetaTransition et Transition.

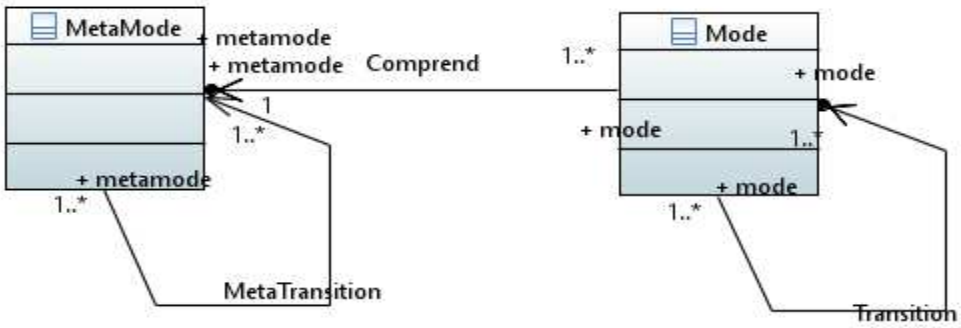

Figure 20 : Relations entre MetaMode, MetaTransition, Mode, Transition

Lorsqu'un événement survient (représentant la Transition), un ensemble de changements (reconfigurations) est appliqué sur le mode courant afin de passer au nouveau Mode du MetaMode cible. Le choix du Mode cible du nouveau MetaMode est effectuée selon certaines politiques. En effet, chaque MetaMode sollicite une instance de l'architecture matérielle du système.

3.3.2 Illustration des concepts : Une machine à laver Hi-Tech

Dans cette sous-section, nous illustrons les concepts via une machine à laver Hi-Tech. En effet, cette machine fait partie des systèmes embarqués de la nouvelle génération. Elle présente différents modes de fonctionnement :

Mode « **Marche complète »** : Le système effectue le lavage via une instance de l'**Essoreur**, une instance du **Laveur**, une instance du **Rinceur**

Mode « **Marche rapide »** : Le système effectue le lavage avec deux instances du **Laveur**, une instance du **Rinceur**

Mode « **Veille Simple »**: Le système arrête son fonctionnement via une instance du **Bloqueur** et une instance de **Veilleuse**

Mode « **Veille prolongée »** : Le système sauvegarde son état actuel et arrête son fonctionnement grâce à une instance de **Sauve-Etat**, une instance de **Bloqueur** et deux instances de **Veilleuse**

Le tableau 1 ci-dessous liste les Modes du système

Tableau 1: Les Modes du Système

Modes	Descriptions
Marche Complète	Ce mode permet au système d'essorer, de laver et de rincer
Marche Rapide	Ce mode permet au système de laver et de

	Rincer rapidement
Veille Simple	Ce mode permet au système de se bloquer à son état actuel et d'être inactif rapidement
Veille Prolongée	Ce mode permet de sauvegarder l'état actuel du système, de le bloquer et le rendre inactif

Le tableau 2 ci-dessous montre les Transitions

Tableau 2 : Les Transitions du système

Les Transitions	Modes Départ-Modes Cible
Activate-VeilleS1	Marche Complète-Veille Simple
Activate-VeilleP1	Marche Complète-Veille Prolongée
Activate-VeilleS2	Marche Rapide-Veille Simple
Activate-VeilleP2	Marche Rapide-Veille Prolongée
Desactivate -VeilleS1	Veille Simple- Marche Complète
Desactivate-VeilleP1	Veille Prolongée- Marche Complète
Desactivate-VeilleS2	Veille Simple- Marche Rapide
Desactivate-VeilleP2	Veille Prolongée- Marche Rapide

La figure 21 ci-dessous met en évidence la machine à états de la machine à laver avec ses différents Modes et ses Transitions

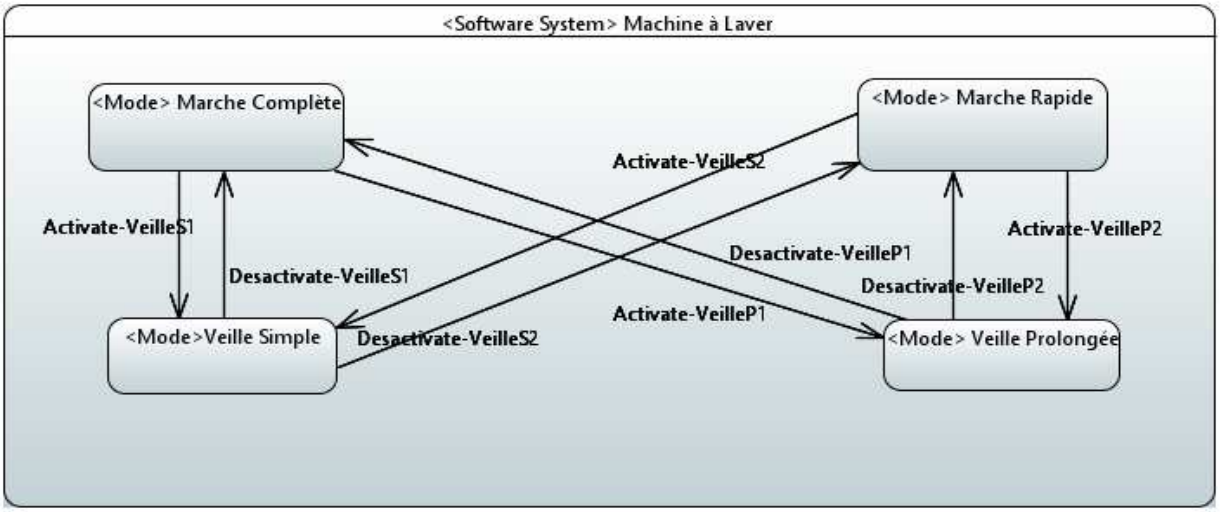

Figure 21 : Le système Machine à laver et ses Modes et Transitions

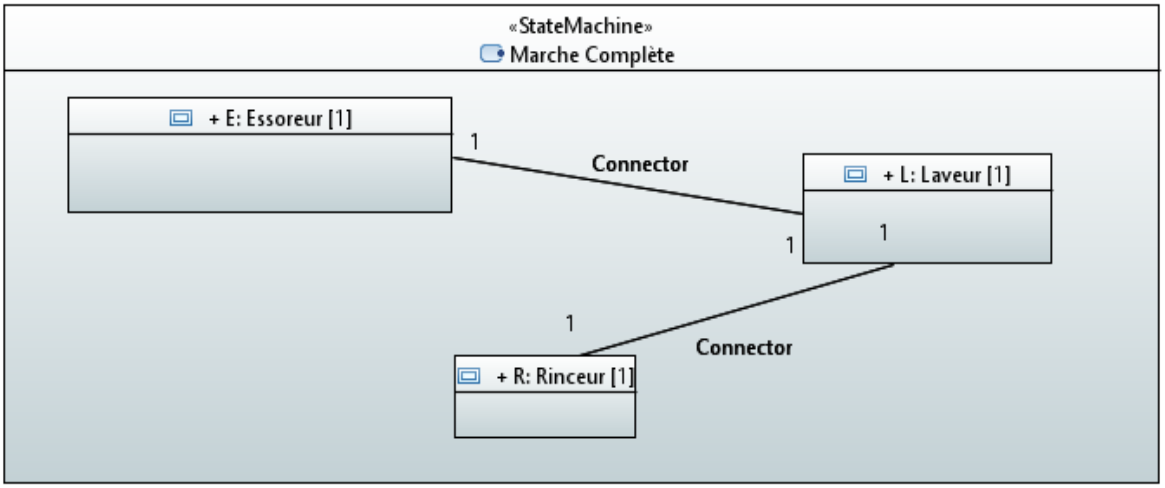

Figure 22 : Le Mode « Marche Complète » et ses composants

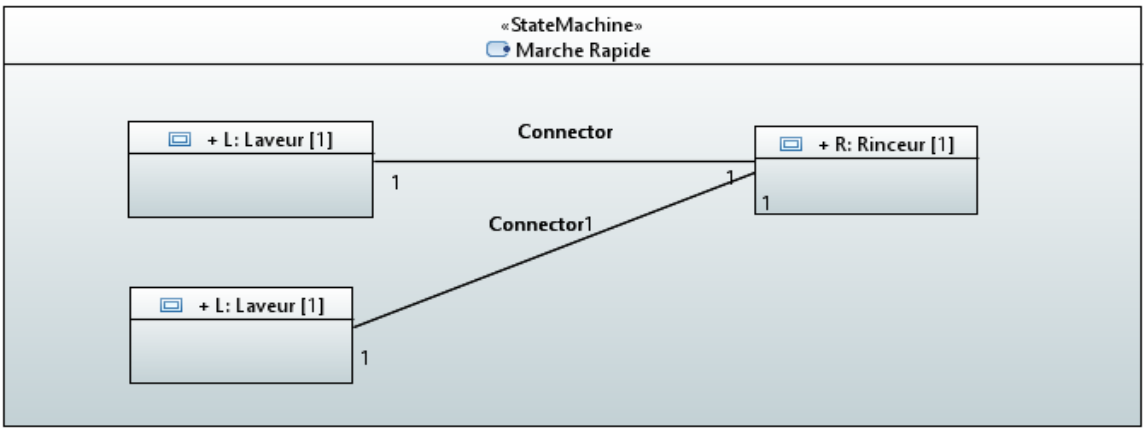

Figure 23 : Le Mode « Marche Rapide » et ses composants

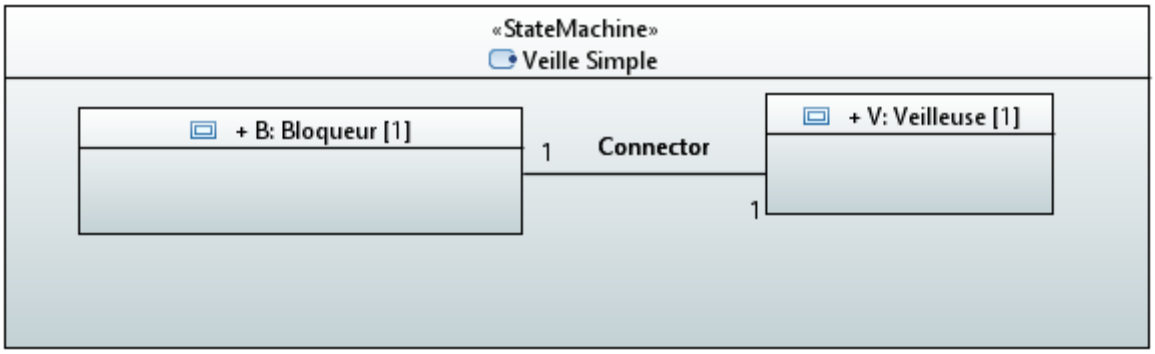

Figure 24 : Le Mode « Veille Simple » et ses composants

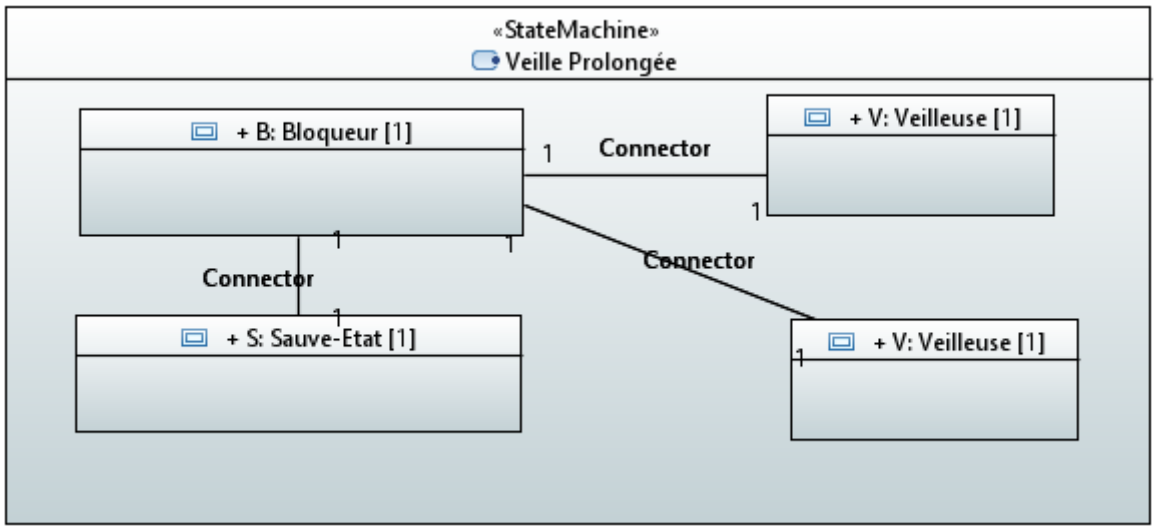

Figure 25 : Le Mode « Veille Prolongée» et ses composants

Afin de simplifier les Modes de notre machine à laver, nous proposons de les regrouper en deux MetaModes qui sont :

- MetaMode Marche : permet de spécifier les composants, les connexions potentielles et les contraintes structurelles, non fonctionnelles et de l'allocation qui caractérisent les différents types de « Marche » du système
- MetaMode Veille : met en évidence les caractéristiques des différents types de « Veille » du système

  La figure ci-dessous met en évidence le MetaMode « Marche »

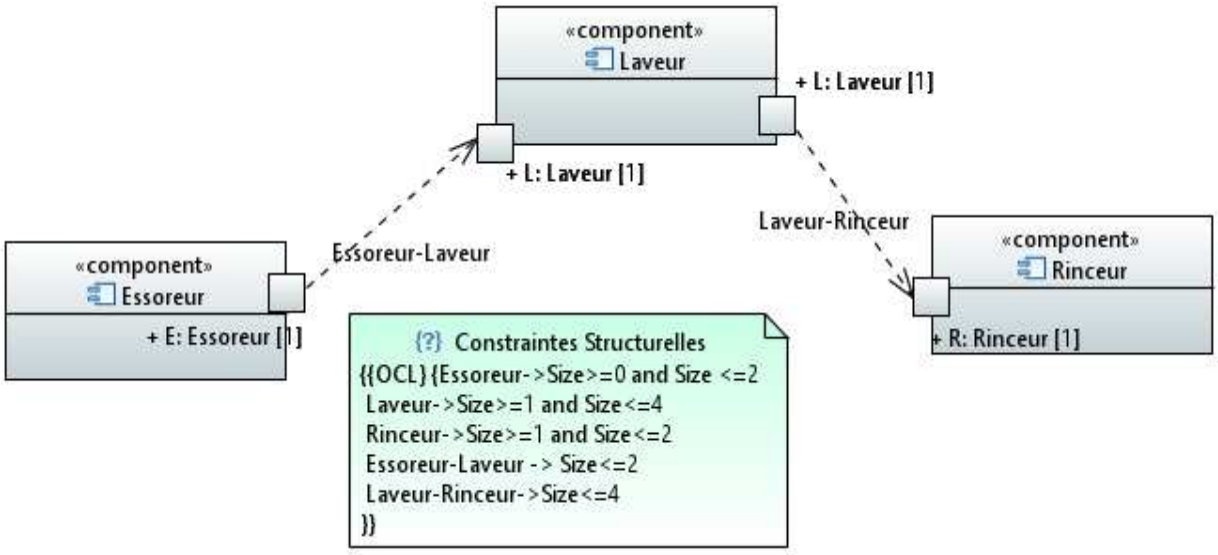

Figure 26 : Le MetaMode « Marche » du système

La figure ci-dessous met en évidence le MetaMode « Veille »

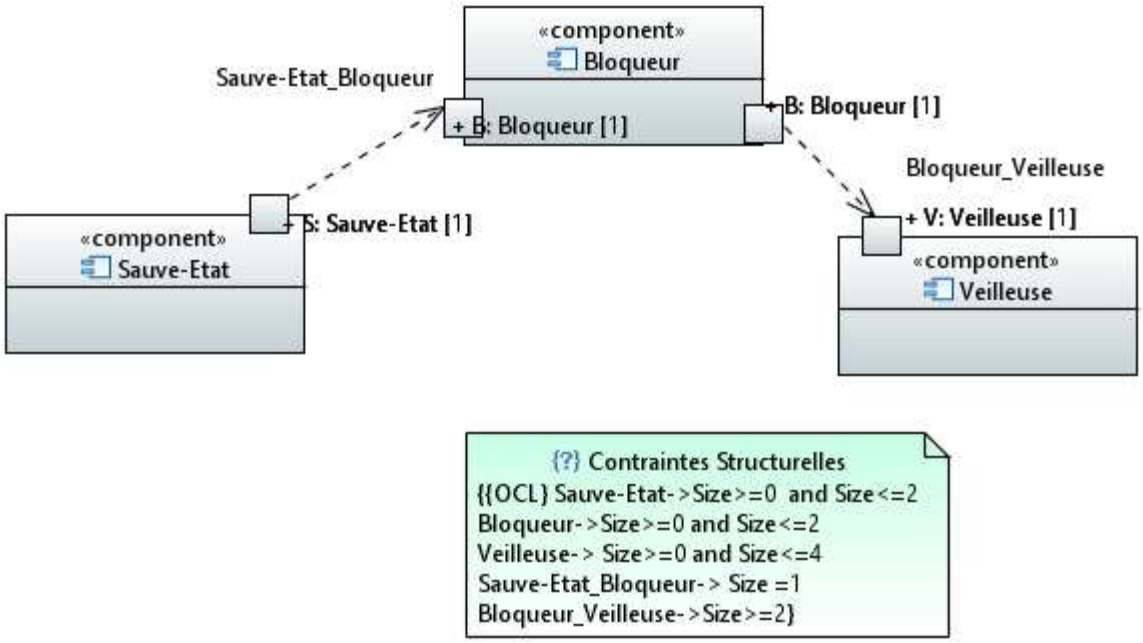

Figure 27 : Le MetaMode « Veille » du système

La figure ci-dessous montre la machine à états des MetaModes, MetaTransitions du système qui sont :

- MetaMode **Marche**
- MetaMode **Veille**

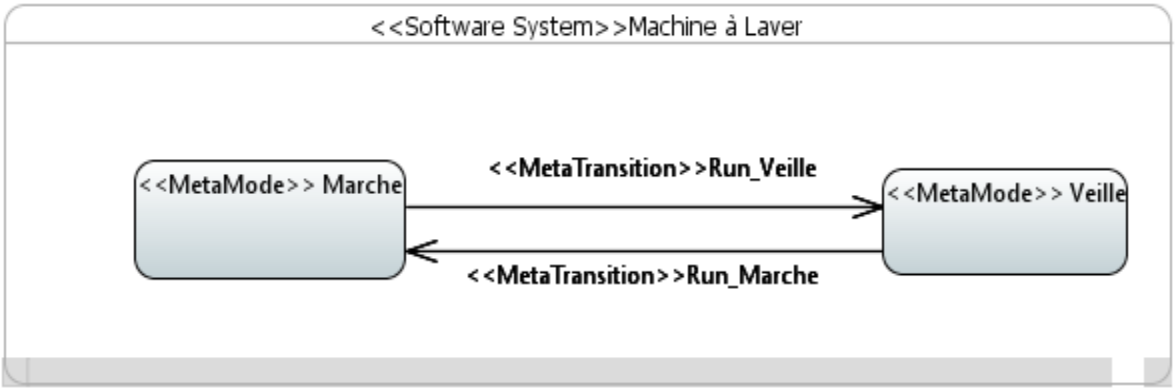

Figure 28 : Machine à états des MetaModes du système

En effet, chaque MetaMode doit être alloué sur une partie de l'architecture matérielle.

La figure ci-dessous illustre l'allocation du MetaMode **Marche** sur l'architecture matérielle.

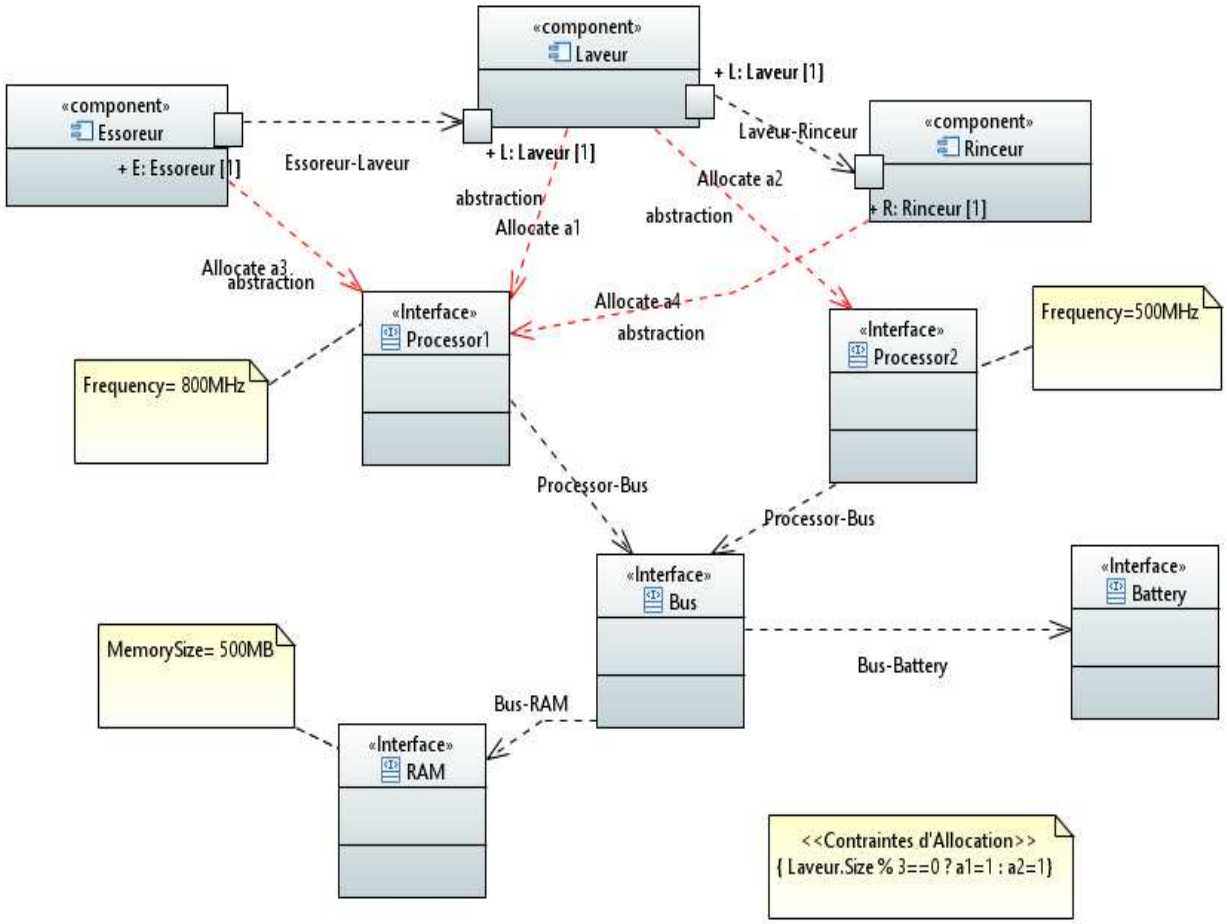

Figure 29 : L'allocation du MetaMode « Marche » sur l'architecture matérielle

En effet, cette figure met en évidence les différents composants structurés pouvant intervenir dans la MetaMode « **Marche »** du système. Les contraintes d'allocation sur l'architecture matérielle sont aussi visibles.

Le tableau ci-dessous spécifie les contraintes dites non fonctionnelles des composants structurés du système de machine à laver Hi-Tech

Tableau 3 : Les propriétés non fonctionnelles des composants structurés

Composants Structurés	Nature	Période (ms)	Pire Temps d'Exécution (ms)	Utilisation Mémoire(MB)
Essoreur	Périodique	8000	4500	0,50
Laveur	Sporadique	1500	8000	0 ,85
Rinceur	Sporadique	2000	9000	0,75

En effet, tous ces concepts présentés se trouvent dans un méta-modèle appelé M4DRES (Modeling For Dynamically Reconfigurable Embedded Systems).

3.3.3 M4DRES (Modeling For Dynamically Reconfigurable Embedded Systems)

M4DRES est un nouveau méta-modèle qui permet de spécifier les architectures matérielles, logicielles, les contraintes structurelles, non fonctionnelles, les politiques de reconfiguration du système.

Le diagramme de classes ci-dessous nous permet de représenter ces nouveaux concepts de M4DRES et leurs relations.

En effet, M4DRES présente la méta-classe **« Software System » est** caractérisé par ses propriétés qui sont :

- MemoryUs : indique l'utilisation de la mémoire par le système
- CpuUs : spécifie l'utilisation de processeur(s)
- BandWitdhUs : attribue une valeur à l'utilisation de la bande passante

Cette dernière méta-classe comprend plusieurs MetaModes. Ces derniers sont représentés par la méta-classe **« MetaMode »**. Un MetaMode a plusieurs méta-classes **« MetaTransition »**. Une méta-classe « MetaMode » requiert plusieurs composants structurés qui sont identifiés en méta-classe. Chaque MetaMode qui est une tâche bien déterminée du système a sa nature (soit périodique, soit apériodique, soit sporadique). Ainsi la nature est édifiée par une méta-classe nommée **« NatureMetaMode »**. Les contraintes structurelles, non-fonctionnelles et d'allocation sont liées à leur MetaMode. En effet, chaque type de contrainte est spécifié en méta-classe.

Chaque MetaMode est composé de plusieurs méta-classes **« Mode »** et **« Transition »**.

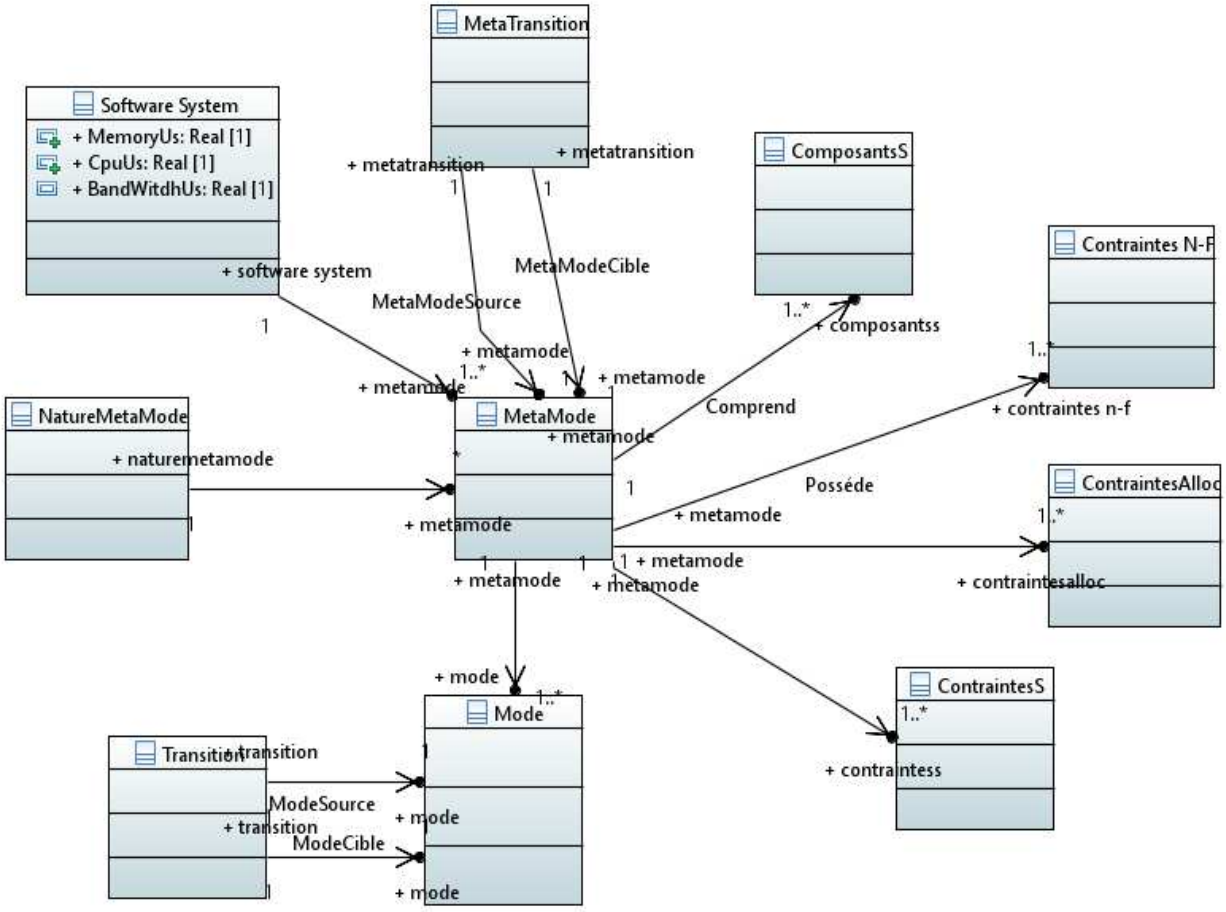

Figure 30 : Le méta-modèle M4DRES (Modeling For Dynamically Reconfigurable Embedded Systems)

En effet, les contraintes non fonctionnelles du système déjà spécifiées doivent être vérifiées par un framework approprié.

3.4 Vérification des contraintes non fonctionnelles

3.4.1 Introduction

Dans la section précédente, nous avons présenté le processus de développement des systèmes embarqués dynamiquement reconfigurables, le nouveau méta-modèle M4DRES et ses concepts. Ainsi, cette section est consacrée à la vérification des propriétés dites non fonctionnelles précédemment présentées (**Tableau3**).

Ainsi, la sous-section 3.4.2 nous permet de définir et de recenser les différents types de propriétés non fonctionnelles.

La sous-section 3.4.3 est dédiée à la consommation du CPU et au respect des échéances

La sous-section 3.4.4 étudie la consommation de la mémoire et de la bande passante.

La sous-section 3.4.5 définit l'absence de l'interblocage et de la famine dans ces systèmes. La sous-section 3.4.6 est dédiée à l'illustration par le système Machine à laver HI-Tech

Enfin, cette section aura des conclusions décisives pour la conception des systèmes embarqués dynamiquement reconfigurables.

3.4.2 Les propriétés non fonctionnelles

Dans cette section, nous présentons la phase de vérification de notre processus de développement (**cf Figure19**). A cette étape, les composants structurés, les contraintes structurelles, non fonctionnelles, d'allocation sont déjà définies. Pour ce faire, nous devons utiliser un framework qui nous permettra d'effectuer des vérifications sur les modèles M4DRES obtenus. Ces vérifications s'intéressent à la consommation du CPU, au respect des échéances des tâches identifiées, à la consommation de la mémoire, de la bande passante, l'absence d'interblocage et de famine. Une fois, ces contraintes vérifiées et validées, le concepteur peut passer à l'étape de génération de code. Dans le cas échéant, le concepteur est obligé de revenir sur ses modèles jusqu'à leurs validations.

Ainsi, ces propriétés sont vérifiées pour toutes les configurations (MetaModes) du système. La vérification de ces propriétés sur les Modes qui constituent les MetaModes est quasi-impossible car leur énumération totale est infaisable. Afin d'effectuer la vérification, nous allons nous intéresser qu'au pire cas d'exécution repéré sur une instance (Mode) du MetaMode à étudier. Pour ce faire, les tâches qui composent les Modes sont divisées en:

Tâche périodique: Elle est caractérisée par un intervalle de temps constant entre deux activations consécutives. Elle est définie par son échéance Ep, sa période Pp et son pire temps d'exécution PTEp.

Tâche sporadique : Elle est déclenchée par un événement à un instant aléatoire. Elle est caractérisée par un délai minimal entre deux activations successives qui est noté Psp. Elle est aussi définie par son échéance Esp et un pire temps d'exécution PTEsp.

Tâche apériodique : Elle est déclenchée par des évènements à un instant quelconque. Cet événement déclencheur est considéré comme temporel. Elle est caractérisée par une date d'arrivée et un pire temps d'exécution PTEap.

3.4.3 La consommation du CPU et le respect des échéances des tâches

Pour vérifier la consommation du CPU et le respect des échéances des tâches du système, nous utilisons le framework Cheddar et l'algorithme d'ordonnancement RMS.

Après étude approfondie, nous constatons que la consommation du CPU et le respect des échéances des tâches se partagent le même pire cas d'exécution d'un MetaMode donné.

La vérification de la consommation du CPU consiste à comparer le facteur d'utilisation du processeur par rapport à une borne bien déterminée. Cette borne est obtenue grâce à

l'algorithme d'ordonnancement choisi (l'algorithme choisi est RMS). En effet, l'utilisation de chaque processeur du système devra respecter la formule ci-dessous :

Formule 16 : La condition à respecter par l'utilisation du processeur

$$\sum_{i=0}^{n} (PTEi / Pi) \leq n(2^{1/n} - 1)$$

n : Nombre de tâches périodiques

PTE_i : Pire Temps d'Exécution de la tâche i

P_i : Période de la tâche i

La vérification avec Cheddar est effectuée sur une méta-période du système. En effet, la méta-période représente le PPCM (Plus Petit Commun Multiple) des périodes des tâches périodiques. Après chaque méta-période le système se retrouve à son état initial. Ainsi, Cheddar pourra vérifier le respect des échéances des tâches et la consommation CPU. Cheddar pourra identifier les tâches qui ne respectent pas leurs échéances. Ainsi, si la vérification émet un résultat positif durant la méta-période, le respect de ces deux propriétés sera possible durant toute l'exécution du système. Cheddar et l'algorithme RMS ne s'intéressent qu'aux tâches périodiques du système. Ainsi, les tâches apériodiques et sporadiques sont considérées comme des tâches périodiques ($P_{ap}=P_{sp}=P_p$; $E_{sp}=E_{ap}= E_p$).

3.4.4 La consommation de la mémoire et de la bande passante

En effet, le pire cas d'exécution d'un Mode du système est celle qui sollicite le maximum d'instances des composants structurés. Chaque composant structuré a une propriété appelée « MemorySize». Cette propriété nous renseigne sur l'empreinte mémoire de la tâche. La vérification en consommation mémoire est opérée au niveau des nœuds où s'exécutent plusieurs tâches. En effet, la taille de mémoire des tâches en cours d'exécution doit être inférieure à celle du nœud. Ceci nous permet de nous intéresser aux différents types de tâches rencontrées dans un nœud :

- Des tâches périodiques et/ou sporadiques : La somme des empreintes mémoires de ces tâches devra être inférieure à la taille de la mémoire du nœud. Dans le cas échéant, un dépassement de mémoire est signalé.
- Des tâches apériodiques : Comme le cas précédent, la consommation du nœud est calculée par la somme des tâches exécutées. Si cette somme dépasse la mémoire du nœud, nous assistons à un chevauchement des intervalles de temps des tâches apériodiques. Ainsi, la comparaison s'opère entre l'empreinte mémoire maximale d'un

des intervalles obtenus et la taille de mémoire du nœud. Si l'empreinte mémoire maximale est inférieure à la taille de la mémoire du nœud, la vérification est un succès.

- Des tâches périodiques et/ou sporadiques et des tâches apériodiques : Dans ce cas, l'empreinte mémoire est obtenue par la somme des mémoires des tâches périodiques et/ou sporadiques et celle des tâches apériodiques. Si cette somme est inférieure à la mémoire du nœud, la vérification est réussie.

Le pire cas d'exécution de la consommation de la bande passante dans un système embarqué consiste :

- au nombre maximal de connexions logicielles
- à l'exécution simultanée de ces connexions logicielles contenues sur un même bus

  En effet, un système embarqué distribué comprend plusieurs nœuds reliés entre eux par des bus. Chaque nœud comprend un ou plusieurs CPU(s) qui sont connectés entre eux par des bus. Chaque composant logiciel est lié à un ou plusieurs CPU(s). Les composants logiciels sont reliés entre eux par des connexions.

  Il faut assurer l'exécution de toutes les tâches et l'échange de données sans débordement au niveau de la bande passante.

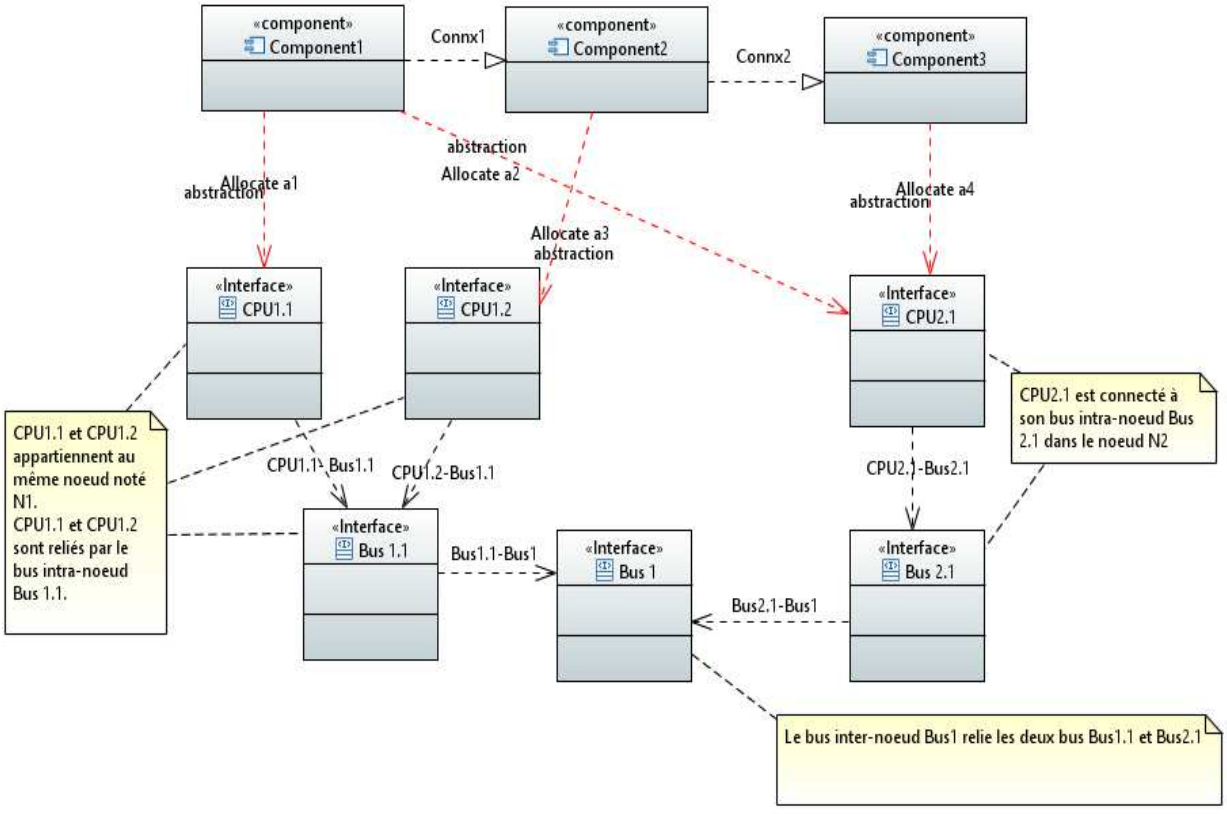

Figure 31 : Les connexions logicielles sur les CPUs, Les bus intra-nœuds et inter-nœuds

Ainsi, nous proposons un algorithme qui vérifie la consommation des bandes passantes des bus du système.

```
SomBCon=0 // Somme des bandes passantes des connexions logicielles projetées sur un bus
BBus // Bande passante du bus
BPC[i]// Bande passante d'une connexion logicielle par instance de CPU
Pour chaque couple de CPUs Faire
  Pour chaque connexion logicielle Faire
    SomBCon= SomBCon + (BPC[i]* nbreInstances)
    Si  SomBCon>BBus
    Sortir ('Débordement de  la Bande Passante')
  FSI
FPOUR
```

Figure 32: Algorithme de vérification de la consommation des bandes passantes des bus du système

En effet, l'algorithme ci-dessous permet de déterminer la présence (ou l'absence) de débordement de la bande passante d'un bus intra nœud.

L'algorithme s'intéresse à un bus et son couple de CPUs interconnectés appartenant à un seul nœud. Ainsi, la somme des bandes passantes des connexions logicielles appliquées sur chaque CPU notée *SomBCon.* Cette somme est obtenue à partir de la bande passante de chaque connexion *BPC[i]* multipliée par le nombre maximal d'instances *nbreInstances*. La somme *SomBCon* ainsi incrémentée est comparée à la bande passante du bus *BBus.*

Enfin, nous nous intéressons à la consommation de la bande passante dans les bus inter-nœuds. Un bus inter-nœud permet de relier plusieurs bus sortants (appartenant aux nœuds bien sûr). La vérification de ces bus inter-nœuds s'intéresse à la somme des bandes passantes des connexions logicielles locales et distantes projetées et la compare à sa bande passante.

3.4.5 L'absence de l'interblocage et de la famine

En effet, ces propriétés du système mettent en évidence sa sûreté et sa vivacité.

L'absence d'interblocage (Deadlock freedoom) traduit que le système ne se bloque pas dans une situation où plusieurs processus s'attendent les uns par rapport aux autres.

Cependant, l'absence de famine (Livelock freedoom) définit que le système travaille tout en ne produisant pas de résultat ainsi il n'avance pas.

L'évaluation de ces propriétés sur un système est quasi-impossible car nous devons opérer la vérification sur toutes les configurations (Modes) des MetaModes. En effet, l'énumération de

toutes les Modes du système est ardue. Cependant, le pire cas d'exécution de chaque MetaMode est inestimable dans ces cas de propriétés.

Afin d'assurer l'absence d'interblocage et de famine, certaines restrictions sont imposées aux développeurs de ces systèmes pendant sa construction. Ces restrictions sont introduites par le profil Ravenscar et définissent le partage des objets par les différentes tâches périodiques et sporadiques du système. En effet, les tâches apériodiques sont déclenchées aléatoirement. Ainsi, ces objets partagés sont protégés contre les accès concurrents. Cependant, un problème de blocage peut se produire pendant l'attente des tâches pour accéder aux objets partagés. Ainsi, le profil Ravenscar résout ce problème par l'utilisation du protocole PCP **[Lui90]**.

3.5 Conclusion

Dans ce chapitre, un nouveau méta-modèle M4DRES est introduit avec la définition des concepts tels que MétaMode et Métatransition. Le M4DRES s'intéresse aux architectures matérielles, logicielles, aux contraintes structurelles, non fonctionnelles, aux politiques de reconfiguration du système.

Nous avons introduit un nouveau processus de développement des systèmes embarqués qui s'appuie sur M4DRES pour effectuer la modélisation et sur le framework Cheddar pour la vérification des différentes propriétés non fonctionnelles (la consommation de la mémoire, le respect des échéances des tâches, la consommation de CPU et de la bande passante, l'absence d'interblocage et de famine).

En effet, le framework de vérification Cheddar devra être associé au méta-modèle M4DRES durant le processus de développement. Ainsi, si la vérification d'une de ces propriétés échoue, le framework de modélisation est évoqué afin de procéder à la rectification.

Ainsi, le chapitre suivant est consacré à la plateforme d'exécution et les générateurs de code associés.

Chapitre 4 : Intergiciel dédié aux Systèmes embarqués à temps réel dynamiquement reconfigurables Et Evaluation de nos contributions

4.1 Introduction

A la suite d'une étude approfondie sur la modélisation et la vérification, ce chapitre est dédié à la proposition d'une plateforme d'exécution adaptée aux systèmes embarqués à temps réel dynamiquement reconfigurables. La section 4.2 est consacrée à l'étude d'une plateforme d'exécution privilégié PolyORB_HI, ses caractéristiques, ses utilités. Ensuite, la section 4.3 s'intéresse à la description de notre plateforme proposée.

Par ailleurs, la section 4.4 est dédiée à l'évaluation de nos contributions par rapport aux solutions existantes.

Enfin, la section 4.5 nous permet de conclure ce chapitre.

4.2 La plateforme d'exécution PolyORB_HI

PolyORB_HI est un intergiciel dédié aux systèmes Temps réel répartis dynamiquement reconfigurables. Il est inspiré de l'architecture schizophrène et il introduit des services intergiciels canoniques pour implanter la communication entre plusieurs plates-formes hétérogènes. En outre, il supporte la séparation des préoccupations. Ses services offrent des mécanismes de communication permettant de gérer la transmission de l'information entre les nœuds d'une application répartie.

4.2.1 Services canoniques de PolyORB_HI

PolyORB_HI offre divers services à ses applications qui sont :

- Le service adressage : s'occupe de la gestion des références des entités externes à l'intergiciel
- Le service activation : s'intéresse à l'activation de l'entité concrète nécessaire pour traiter une requête reçue par une entité réceptrice
- Le service exécution : supporte l'exécution du traitement requis par la réception d'un message. Il se situe au dessus du service d'activation et il utilise l'entité concrète activée par le service d'activation pour exécuter le traitement d'un message reçu
- Le service liaison : permet la construction des ressources de communication requises pour dialoguer avec une entité distante ou locale

- Le service typage : s'occupe de la gestion des types de données transmises entre les nœuds et fournies à l'application
- Le service représentation : assure la transmission correcte des données à travers le réseau. Pour cela, il contrôle l'emballage des ressources, dans un tampon de communication, par l'émetteur et le déballage de ces ressources par le récepteur
- Le service interaction : s'occupe de la gestion de l'interaction entre deux nœuds logiques. Pour cela, il existe plusieurs modes d'interaction (par exemple le mode synchrone et le mode asynchrone)
- Le service protocole : dicte les étapes nécessaires pour la transmission d'un message entre deux entités,
- Le service transport : assure l'ouverture, la fermeture et l'utilisation des canaux de communication pour la transmission d'un message.

4.2.2 Familles des services canoniques

Afin d'assurer un bon fonctionnement, les services proposés par PolyORB_HI sont divisés en grandes familles illustré dans le Tableau4:

- Les services faiblement personnalisables: ils sont présents avec la même configuration dans toutes les applications. Ils constituent le noyau de l'intergiciel PolyORB_HI minimal
- Les services fortement personnalisables: Cependant, ces services sont changeants selon les besoins de l'application. Ils peuvent être paramétrés selon les fonctionnalités de l'application.
- Les services composés : En effet, ces services peuvent être à la fois fortement et faiblement personnalisable. Ainsi, on assiste à la génération d'autres nouveaux services.

Tableau 4 : Répartition des services canoniques de PolyORB_HI [Bec08]

Services canoniques	Nouveaux Services	Degré de personnalisation
Adressage	Adressage	Très fort
Liaison	Liaison	Très fort
Activation	Activation	Très fort
Exécution	Parallélisme Exécution	Très faible Très fort
Typage	Typage	Très fort
Représentation	Représentation élémentaire Représentation avancée	Très faible Très fort

Interaction	Interrogation Interaction	Très faible Très fort
Protocole	Protocole	Très faible
Transport	Couche basse de transport Couche haute de transport	Très faible Très fort

Le tableau 4 ci-dessus illustre la répartition des services canoniques de PolyORB_HI en fonction de leur degré de personnalisation.
Ainsi, l'adressage, la liaison, le typage et l'activation constituent les services fortement personnalisables. Cependant, le protocole qui dicte les étapes de transmission de message entre les entités est le seul service faiblement personnalisable. Ensuite, l'exécution, la représentation, l'interaction et le transport représentent les services composés.

4.2.3 Support des constructions de systèmes embarqués dynamiquement reconfigurables

En effet, l'intergiciel PolyORB_HI fournit certains avantages qui sont :

- Les processus légers : Ils constituent des mécanismes qui sont proposés permettant de faciliter la création des tâches des systèmes critiques
- Les interfaces : Ils représentent des mécanismes offerts permettant aux différentes entités ou code d'une application d'effectuer de façon déterministe des opérations au niveau des interfaces des processus légers
- La protection des données partagées : Elle permet de garantir l'accès aux données partagées par les différentes entités via des méthodes de verrouillage et de déverrouillage.

4.2.4 Faible empreinte mémoire

En effet, seuls les services faiblement personnalisés sont implantés dans l'intergiciel minimal PolyORB_HI. Cependant, les services fortement personnalisés se retrouvent dans le code applicatif de l'application. Ainsi, PolyORB _HI a une propriété intéressante pour les systèmes embarqués qui est sa faible empreinte mémoire.

4.3 Description de notre intergiciel dédié pour les systèmes embarqués dynamiquement reconfigurables

Suite aux limites des intergiciels énoncés, nous proposons une nouvelle plateforme nommée ESDRES (Execution Support For Dynamically Reconfigurable Embedded Systems) qui devra

faciliter l'exécution des systèmes dynamiquement reconfigurable. En effet, la plateforme ESDRES est inspirée de PolyORB_HI.

4.3.1 Fonctionnalités attendues

La plateforme ESDRES devra faciliter les reconfigurations dynamiques du système en assurant la cohérence et la supervision. Cet intergiciel devra aussi respecter les contraintes temps réel avec une faible empreinte mémoire.

En effet, l'intergiciel ESDRES permet de :

- Assurer la supervision du système en vérifiant au-cours de l'exécution le comportement du système, l'architecture matérielle et mettant à jour les variables partagées
- Respecter les contraintes temps réel
- Garantir la cohérence du système durant et après les reconfigurations
- Faciliter la communication entre les composants hétérogènes du système en s'appuyant sur l'architecture schizophrène et ses services canoniques

4.3.2 Modèle de notre intergiciel

A partir de ces fonctionnalités énoncées précédemment, nous proposons un modèle sous diagramme de classes de notre plateforme ESDRES.

Tableau 5 : Les classes de l'intergiciel ESDRES et leurs descriptions

Classes	Descriptions
Evenement	Cette classe identifie l'ensemble des événements pouvant influer sur l'exécution du système
ReconfigDyn	Cette classe présente les méthodes AddTask, RemoveTask, … du système qui sont déclenchées par un ou plusieurs événements. Ces méthodes assurent les reconfigurations du système
Architecture-Inst	Cette classe représente l'architecture de l'application à chaque instant de son exécution
Observer	Cette classe observe l'architecture du système à chaque instant de son exécution
MetaMode	Cette classe s'intéresse aux différents

	MetaModes du système.
Mode	Cette classe identifie les Modes du système. Chaque Mode appartient à un seul MetaMode
Tâche	Cette classe représente les différentes tâches du système
Tâche Périodique	Cette classe ne s'intéresse qu'aux tâches périodiques du système
Tâche Apériodique	Cette classe ne s'intéresse qu'aux tâches apériodiques du système
Tâche Sporadique	Cette classe ne s'intéresse qu'aux tâches sporadiques du système
InPort	Cette classe recense les ports d'entrée affectés aux tâches
OutPort	Cette classe recense les ports de sortie affectés aux tâches
Routeur-Port	Cette classe identifie les routeurs qui facilitent la communication entre tâches
Structure-Données	Cette classe s'intéresse aux structures de données assurant la communication entre tâches

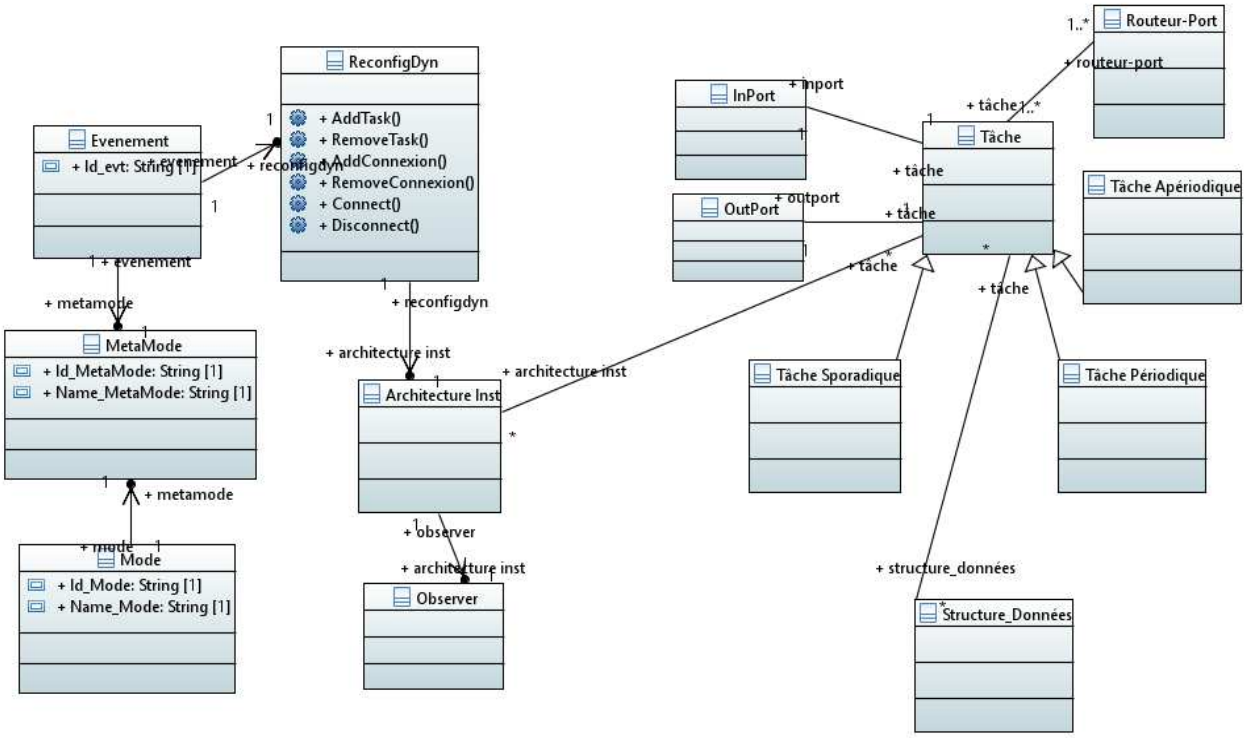

Figure 33: Modèle de l'intergiciel ESDRES

En effet, cet intergiciel devra faciliter à l'application les reconfigurations architecturales suivantes :

- Connecter des nœuds : créer un canal de communication entre deux nœuds,
- Déconnecter des nœuds : supprimer un canal de communication entre deux nœuds,
- Ajouter une connexion : créer un canal de communication entre deux composants,
- Supprimer une connexion : supprimer un canal de communication entre deux composants, Ajouter un composant : créer et déployer un composant sur une plate-forme d'exécution et le connecter avec les autres composants,
- Supprimer un composant : supprimer le composant et ses connexions avec les autres composants,
- Migrer un composant : déplacer un composant d'un nœud vers un autre nœud.

La plateforme ESDRES pourra également assurer les reconfigurations comportementales de l'application ci-dessous :

- Mettre à jour les propriétés d'un composant : en leur affectant des nouvelles valeurs.
- Remplacer un composant : remplacer un composant par un autre.

4.4 L'évaluation de nos contributions

Cette section est consacrée à l'étude de comparaison détaillée de nos contributions et des solutions existantes. A cette étape où toutes nos contributions sont dévoilées. Cette étude de comparaison qui est établie sur des critères pertinents est faisable.

Le tableau (Tableau 6) ci-dessous nous permet de comparer notre méta-modèle M4DRES par rapport au profil UML MARTE existant.

Tableau 6: Comparaison de M4DRES et MARTE

Outils de Modélisation des systèmes embarqués	Forte Présence d'UML	Représentation des Propriétés Non Fonctionnelles	Modélisation simple des reconfigurations
M4DRES	√	√	√
MARTE	√	√	×

Le tableau (Tableau 7) ci-dessous met en évidence la comparaison de l'intergiciel proposé ESDRES par rapport aux autres existants.

Tableau 7: Comparaison de l'intergiciel ESDRES et des autres existants

Intergiciels dédiés aux systèmes embarqués	Systèmes à temps réel statiques	Systèmes à temps réel dynamiques	Faible Empreinte Mémoire
ESDRES	√	√	√
PolyORB_HI	√	×	√
CIAO	√	√	×
DynamicTAO	√	√	×

Le tableau (Tableau 8) s'intéresse à la comparaison de notre processus de développement par rapport aux autres utilisés.

Tableau 8: Comparaison de notre processus proposé et des autres existants

Processus et Frameworks de développement	Intergiciel Proposé	Systèmes à temps réel statiques	Systèmes à temps réel dynamiques
Notre processus	√ ESDRES	√	√
ModES	×	√	×
Ocarina	√ PolyORB_HI	√	×
TimeAdapt	×	√	√

4.5 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle plateforme d'exécution qui est une extension de PolyORB_HI. Cet intergiciel assure les différentes reconfigurations du système tout en le supervisant et respectant sa cohérence durant son exécution.

Nous avons élaboré une étude comparative de nos différentes contributions proposées par rapport aux solutions existantes. Cette étude de comparaison met en évidence l'avancée positive de chacune de nos contributions dans la réalisation des systèmes embarqués.

Dans le chapitre suivant, nous allons nous intéresser aux différents outils qui nous permettront de respecter notre processus de développement proposé.

Chapitre 5 : Outillage du processus de développement

5.1 Introduction

Ce chapitre permet d'identifier les différents outils qui nous permettent de réussir les différentes étapes du processus de développement évoqué dans le chapitre 3.

La section **5.2** nous permet d'introduire l'architecture de la suite des outils proposés. Ensuite, la section **5.3**, nous explicitons les outils de modélisation du système à concevoir. Cependant la section **5.4** est dédiée aux outils de vérification des différentes propriétés non fonctionnelles. Enfin, la section **5.5** s'intéresse aux outils de génération d'une grande partie code du système.

5.2 L'architecture de la suite d'outils

Afin de respecter les étapes de notre processus de développement, notre suite d'outils s'appuie essentiellement sur la plate-forme Eclipse et ses plugins.

La suite d'outils est constituée d'un ensemble d'entités:

- L'entité de modélisation : comprend l'éditeur UML Eclipse Papyrus et les profils MARTE et M4DRES
- L'entité de vérification : est composé du framework Cheddar et de l'outil greatspn
- L'entité de génération : comprend le plugin à base d'ATL pour la génération de modèle à modèle et le plugin à base d'Acceleo pour la génération de modèle à code
- Enfin, la plate-forme d'exécution: est une extension de l'intergiciel PolyORB_HI

5.3 Les outils de modélisation

Depuis son apparition en 2008, Eclipse Papyrus Project est utilisé dans d'énormes projets de réalisation de systèmes embarqués d'envergure mondiale.

L'éditeur des modèles le plus fiable et open source est Eclipse Papyrus. Eclipse Papyrus donne l'opportunité à ses utilisateurs de modéliser, d'analyser leurs systèmes via UML et ses profils SysML, RobotML, MARTE et mieux la création d'autres profils à partir de ceux existants.

Ainsi, notre méta-modèle M4DRES est un profil créé en s'appuyant sur UML, MARTE et leurs spécifications (cf. **Figure 34**).

Afin de définir les contraintes non fonctionnelles et d'allocation, M4DRES s'appuie sur les sous profils VSL et NFP et la librairie des types Basic NFP_types de MARTE.

5.3.1 L'environnement Eclipse Papyrus

En effet, l'environnement de modélisation d'Eclipse Papyrus facilite à ses utilisateurs la création de modèles à partir des concepts d'UML, SysML préinstallés. Il permet aussi de créer de nouveaux profils à partir de ceux existants, d'installer des plugins de vérification, des outils de génération de modèles comme ATL de code comme Acceleo.

La figure ci-dessous présente l'environnement d'Eclipse Papyrus. Ainsi, le modèle est présenté au milieu. Cependant, la palette d'en haut nous présente les menus de l'éditeur et la palette de droite du modèle nous propose l'ensemble des concepts de profils à utiliser. La palette d'en bas présente les propriétés de composant sélectionné du modèle, les erreurs log, les problèmes … La palette à gauche du modèle nous montre l'arborescence de nos projets existants.

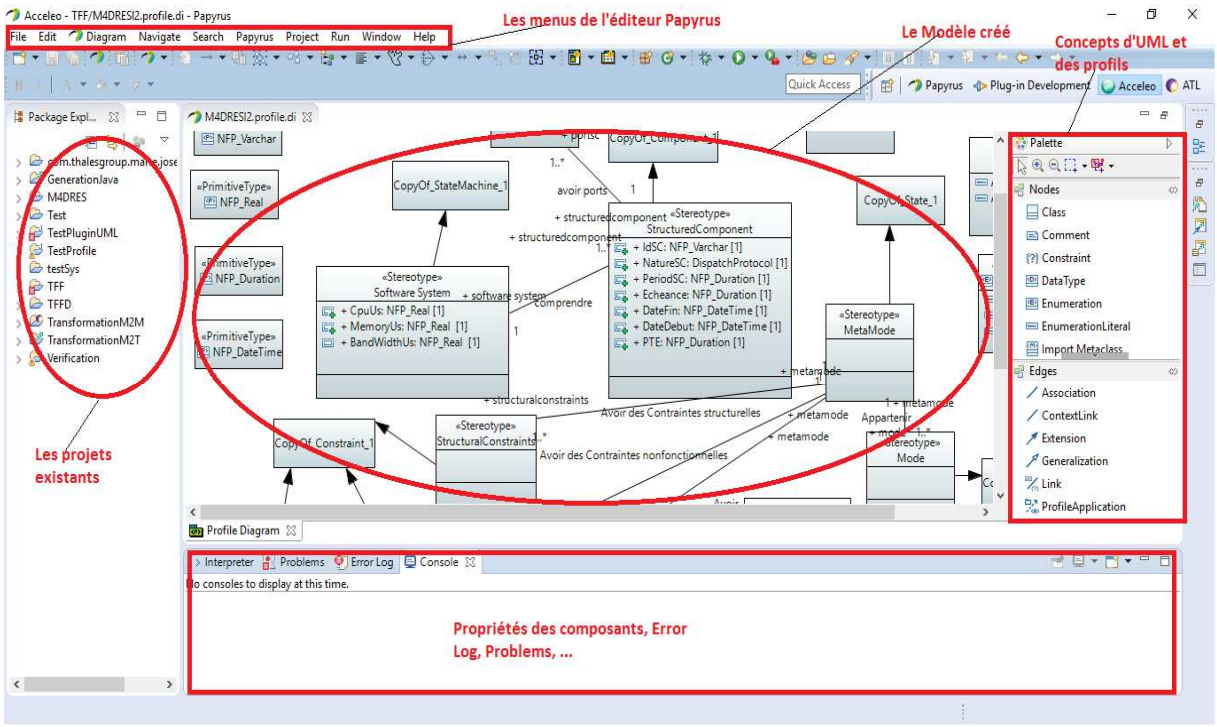

Figure 34 : Environnement Eclipse Papyrus

5.3.2 Le profil M4DRES dans Eclipse Papyrus

En effet, le profil M4DRES que nous avons proposé facilite la modélisation des systèmes embarqués dynamiquement reconfigurables.

La structure de notre profil M4DRES est illustrée par la figure ci-dessous (**Figure 35**)

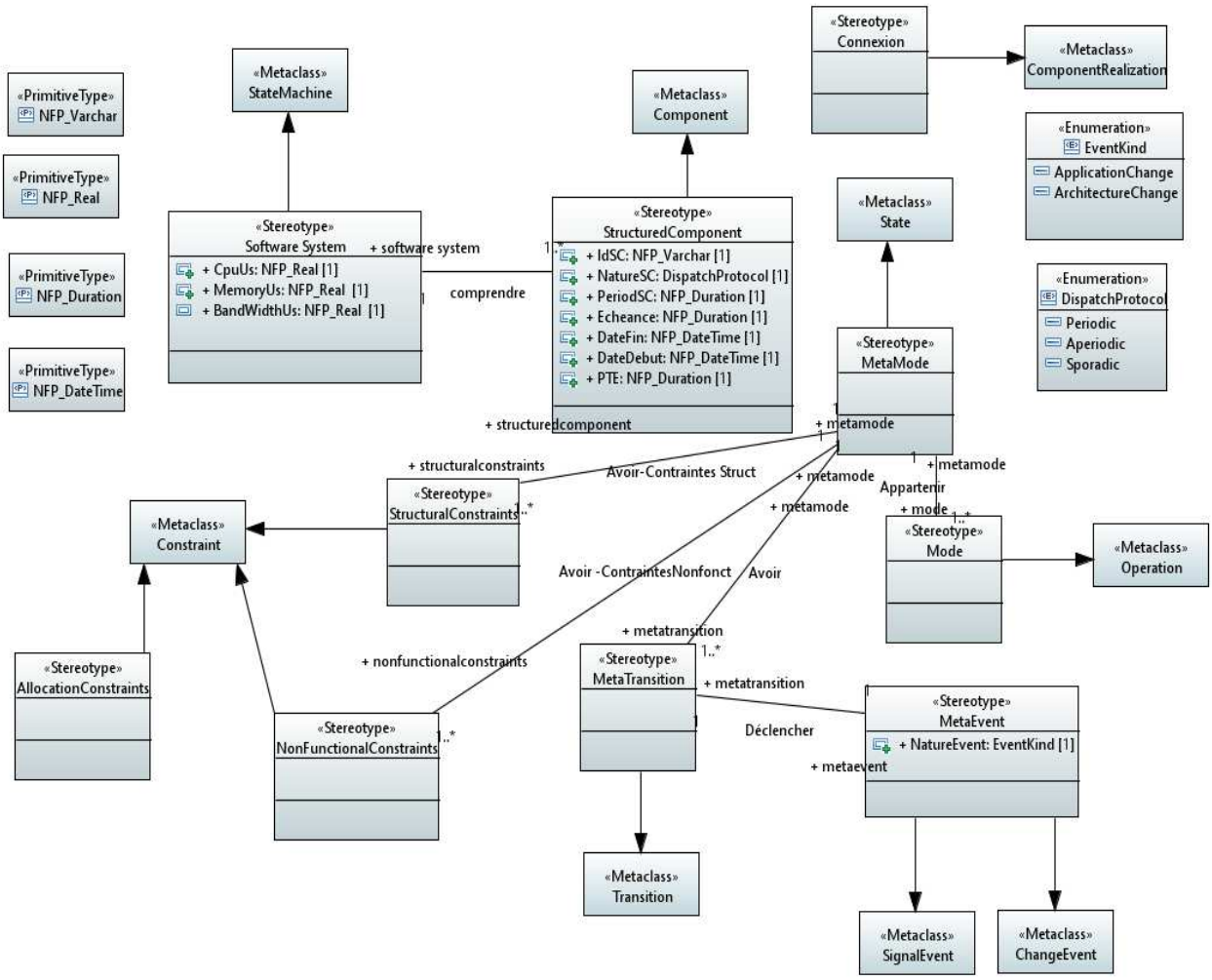

Figure 35 : La structure du profil M4DRES

Ce profil M4DRES comprend :

- Le stéréotype **Software System**
    1. Description : Il représente la machine à états des différentes reconfigurations dynamiques du système. Ce stéréotype comprend différents MetaModes et MetaTransitions.
    2. Méta-classe étendue : StateMachine
    3. Propriétés :
        - ✓ CpuUs mesure l'utilisation de chaque processeur. Il est de type NFP_Real de la librairie des types de MARTE
        - ✓ MemoryUs qui est la consommation de chaque mémoire est de type NFP_Real de la librairie des types de MARTE
        - ✓ BandWidthUs constitue l'utilisation de chaque bande passante. Il est de type NFP_Real de la librairie des types de MARTE

- Le stéréotype **MetaMode**
    1. Description : Il représente l'ensemble des MetaModes du système.
    2. Méta-classe étendue : State
    3. Propriétés :
- Le stéréotype **StructuredComponent**

1. Description : Il permet d'identifier les composants structurés du système. En effet un composant structuré représente une ou plusieurs tâches élémentaires.

2. Méta-classe étendue : Component
3. Propriétés:
    - ✓ IdSC : il permet d'affecter un identifiant au composant structuré. Son type est NFP_Varchar de la librairie des types de MARTE
    - ✓ NatureSC : Il identifie la nature de la tâche. En effet, une tâche peut être à l'une de ces états : périodique, apériodique, sporadique. Son type est identifié par l'énumération DispatchProtocol
    - ✓ EcheanceSC : Il identifie l'échéance de chaque composant structuré. Son type est NFP_Duration de la librairie des types de MARTE
    - ✓ PeriodSC : Il identifie la période octroyée à chaque composant structuré. Son type est NFP_Duration de la librairie des types de MARTE
    - ✓ DateDebut : Il identifie la date de début de chaque composant structuré. Son type est NFP_DateTime de la librairie des types de MARTE
    - ✓ DateFin : Il identifie la date de fin de chaque composant structuré. Son type est NFP_DateTime de la librairie des types de MARTE
    - ✓ PTE : Il représente le pire temps d'exécution de la tâche. Son type est NFP_Duration de la librairie des types de MARTE

- Le stéréotype **Mode**

1. Description : Il identifie les différents Modes du système. Chaque Mode appartient à un MetaMode précis.

2. Méta-classe étendue : Operation
- Le stéréotype **StructuralConstraint**

1. Description : Il identifie les différentes contraintes structurelles attribuées à un MetaMode.

2. Méta-classe étendue : Constraint
- Le stéréotype **NonFunctionalConstraint**

1. Description : Il représente l'ensemble des contraintes non fonctionnelles d'un MetaMode.
2. Méta-classe étendue : Constraint

- Le stéréotype **AllocationConstraint**
  1. Description : Il identifie les différentes contraintes d'allocation d'un MetaMode.
  2. Méta-classe étendue : Constraint
- Le stéréotype **MetaTransition**
  1. Description : Il constitue les relations existantes entre MetaModes.
  2. Méta-classe étendue : Transition
- Le stéréotype **MetaEvent**
  1. Description : Il représente chaque événement pouvant susciter un changement du système.
  2. Méta-classe étendue : SignalEvent, ChangeEvent
  3. Propriétés :
     - ✓ NatureEvent : Il représente la nature de l'événement déclencheur. Son type est identifié par l'énumération EventKind.
- Le stéréotype **Connexion**
  1. Description : Il représente la liaison entre composants structurés. En effet, il illustre le passage d'un composant structuré à un autre.
  2. Méta-classe étendue : Component Realisation

5.4 Les outils de vérification

Dans cette section, nous nous intéressons aux outils pouvant assurer la vérification des propriétés non fonctionnelles de modèles obtenus et l'évaluation de performance du système. En effet, cette étape est réalisable via l'utilisation du framework Cheddar et de l'outil greatspn. En effet, le framework Cheddar s'intéresse à l'ordonnançabilité et au respect des échéances des tâches du système, au respect de consommation mémoire, de CPU, l'absence de famine…Cependant, l'outil greatspn s'intéresse aux délais des tâches, à la disponibilité des ressources.

5.5 Les outils de génération

En effet, la génération est l'une des étapes les plus importantes dans la réalisation de systèmes embarqués dynamiquement reconfigurables. Elle permet de générer automatiquement, à partir des modèles obtenus, une grande partie du code du système. Elle nécessite l'utilisation de deux outils qui sont :

- ATL (Atlas Transform Language) qui assure la transformation de modèles M4DRES vers les modèles d'implantation via des règles de transformation
- Acceleo qui permet de générer une grande partie du code à partir des modèles d'implantation

5.6 Conclusion

En effet, ce chapitre met en évidence les différents outils qui nous aident à respecter notre processus de développement des systèmes embarqués dynamiquement reconfigurables.

En effet, Eclipse papyrus est notre environnement de modélisation qui nous permet de créer un nouveau profil M4DRES adapté à ce type de systèmes. Ce profil s'appuie sur UML et les sous-profils de MARTE afin de représenter les reconfigurations dynamiques, ses différentes propriétés (contraintes structurelles, non fonctionnelles, d'allocation, …).

Ensuite, la vérification de ces propriétés est effectuée grâce au framework Cheddar et à l'outil GreatSPN afin d'apporter des résultats d'analyse. Ces résultats peuvent susciter des rectifications si nécessaires.

Enfin, la génération automatique de la grande partie du code du système est effectuée via les outils ATL et Acceleo.

Ainsi, le **chapitre 6** nous permet d'utiliser un exemple à titre illustratif de notre processus de développement proposé.

Chapitre 6 : Etude de cas -Ecran Publicitaire Intelligent

6.1 Introduction

Ce chapitre nous permet d'illustrer à partir d'un cas pratique tous les concepts étudiés dans les chapitres précédents. En effet, ce cas pratique s'intéresse à la conception d'un écran nommé 'Ecran Publicitaire Intelligent' qui est un système embarqué dynamiquement reconfigurable à temps-réel.

Sa conception est une des moins simples car elle passe par plusieurs étapes :

- La modélisation : qui permet de définir les composants, les tâches du système et leurs caractéristiques via nos concepts de modélisation
- La vérification des propriétés non fonctionnelles : qui permet de vérifier l'ordonnancement et le respect des échéances des tâches, la consommation de CPU(s), la consommation de la mémoire, …
- La génération du code : qui facilite le développement de la partie logicielle du système

6.2 Description du système

Le système baptisé EPI est essentiellement composé :

- D'un écran qui affiche des publicités selon certains critères qui sont : l'âge, le sexe de l'auditoire, la luminosité ambiante, la distance séparant l'écran de la personne
- D'un contrôleur du système utilisable par une personne
- Des capteurs qui reçoivent les mesures des critères déjà énoncés

En effet, EPI permet d'afficher la catégorie de publicités qui devrait intéresser l'auditoire. Par exemple : Si la personne est un jeune homme, EPI va afficher des publicités de musique, sport, études…

Si la personne est une dame de la trentaine, EPI va afficher des publicités de produits cosmétiques, de salles d'habits, d'appareils électroménagers (lave-linge, sèche-linge, micro-onde, réfrigérateurs, …) de la dernière génération.

EPI affiche les informations selon la distance séparant de la personne et de la luminosité ambiante. Par exemple : Plus la personne est proche, plus EPI affiche des informations nettes et claires. Pendant la nuit (Nette diminution de la luminosité ambiante), EPI affiche les informations de façon à éclairer un champ plus large.

Ainsi, EPI fonctionne selon les exigences de ses utilisateurs (qui représentent l'audience) et son environnement d'exécution. EPI est dynamiquement reconfigurable.

Soit le tableau ci-dessous qui simule le fonctionnement du système en fonction des informations recueillis par les capteurs

Tableau 9 : Données de fonctionnement du système EPI

Sexe	Age	Distance (m)	Luminosité	Catégorie Pub	Affichage
Féminin	20	1,50	Faible	1	Fort
Masculin	18	1,50	Moyen	2	Moyen
Masculin	70	0,20	Fort	5	Faible
Féminin	80	0,90	Faible	6	Fort
Féminin	25	0,40	Faible	4	Fort
Féminin	45	0,50	Faible	6	Fort
Masculin	50	2	Fort	3	Faible
Masculin	60	3	Fort	3	Faible
Masculin	15	2	Faible	1	Fort
Masculin	18	0,50	Moyen	1	Moyen
Féminin	13	0,25	Faible	2	Fort
Féminin	22	0,50	Fort	2	Faible
Féminin	65	3	Fort	6	Faible
Féminin	30	3	Moyen	4	Moyen
Masculin	17	0,50	Fort	1	Faible
Masculin	48	0,75	Moyen	3	Moyen
Masculin	75	3	Faible	5	Fort
Masculin	60	2,5	Fort	3	Faible
Masculin	25	2,5	Moyen	3	Moyen
Masculin	47	0,5	Faible	3	Fort
Masculin	60	1,5	Fort	3	Faible
Féminin	12	0,5	Moyen	2	Moyen
Féminin	19	0,75	Moyen	2	Moyen
Féminin	40	0,2	Faible	4	Fort
Masculin	24	3	Fort	1	Faible

Le schéma simplifié ci-dessous met en évidence le système EPI, son audience (homme, femme, Vieux, …) et les paramètres perçus par ses capteurs.

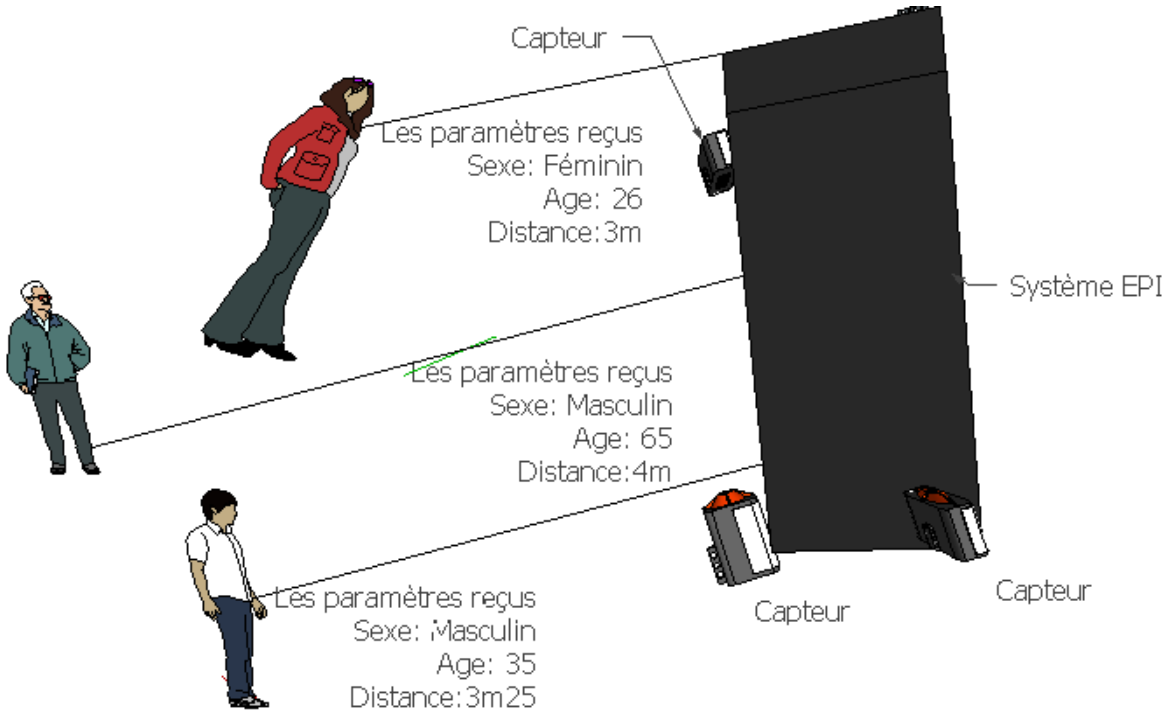

Figure 36: Vue simplifiée du système EPI et son audience

En effet, notre système EPI est simplifié et fonctionne selon deux principaux modes de fonctionnement:

- Mode Self : EPI récupère ses critères via ses capteurs, les traiter, choisit son programme de films publicitaires et l'exécute
- Mode Control : L'utilisateur contrôle EPI avec sa poste de contrôleur. Il effectue ses choix de critères. Ainsi, EPI propose son programme de films publicitaires. Une fois, le programme validé par l'utilisateur, EPI l'exécute.

Le passage d'un mode de fonctionnement à un autre s'opère à partir de la poste de contrôleur.

6.3 Modélisation du système EPI avec M4DRES

M4DRES est notre nouveau profil qui nous aide dans l'analyse et la conception de ce système décrit. Le système embarqué non adaptif présente une configuration unique. Ainsi, ce système sollicite un ordonnancement rigide des fonctions. Cependant, le système embarqué reconfigurable propose plusieurs modes de fonctionnement selon ses conditions d'exécution (exigences de l'utilisateur, de l'environnement, de délai) et des techniques de transitions d'un état initial vers celui final.

En effet, MARTE éprouve des difficultés à réaliser la conception des systèmes embarqués dynamiquement reconfigurables.

Ainsi, les concepts introduits avec M4DRES nous permettent de concevoir les fonctionnalités de notre système dans le respect des contraintes imposées.

A partir du cahier de charges, EPI comprend deux MetaModes qui sont :

- MetaMode « Self_EPI » : le système effectue ses tâches sans l'intervention d'un contrôleur
- MetaMode « Control_EPI » : l'utilisateur commande la plupart des tâches de EPI via son poste de contrôleur

  Ces deux MetaModes sont liés par deux MetaTransitions qui sont :

- MetaTransition « Self-Control » : EPI passe du MetaMode MetaMode « Self_EPI » vers le MetaMode « Control_EPI »
- MetaTransition « Control-Self » : EPI passe du MetaMode « Control_EPI » vers le MetaMode « Self_EPI »

  La figure ci-dessous illustre les MetaModes et MetaTransition du système EPI

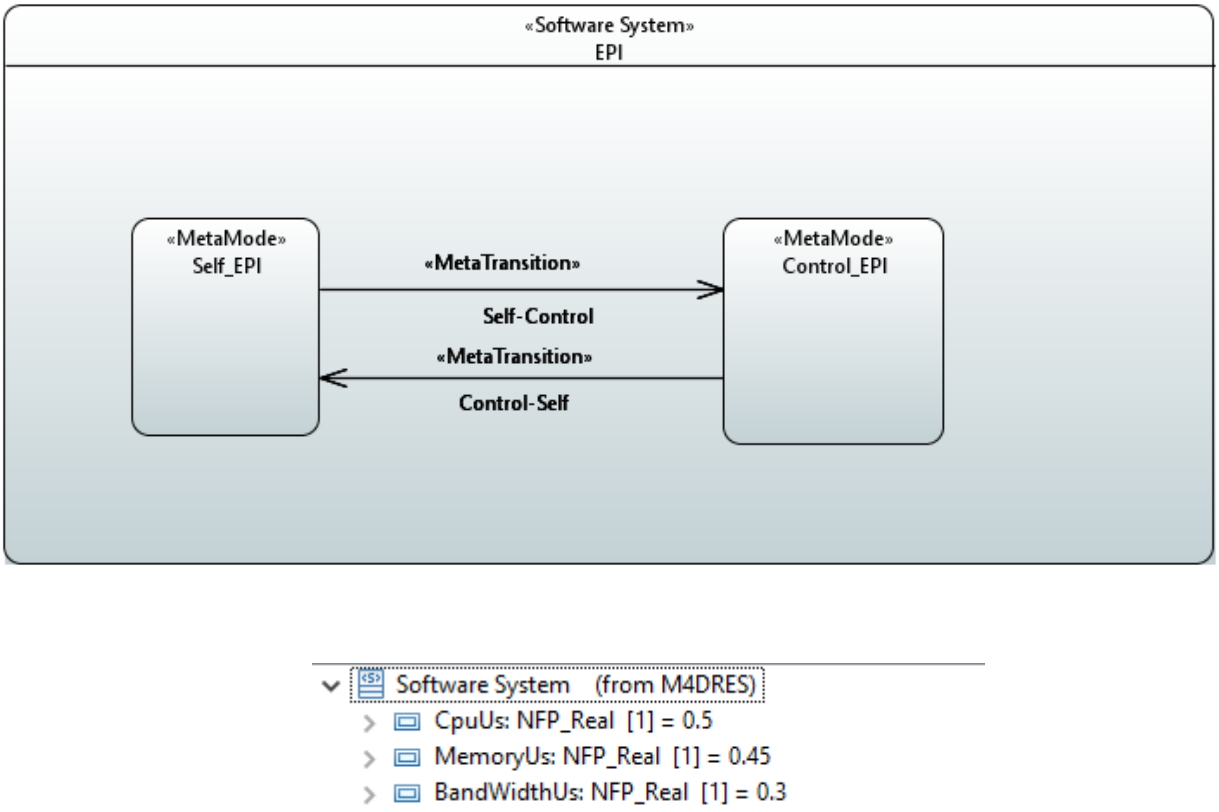

Figure 37 : Machine à états du système EPI

Ainsi, un MetaMode est un stéréotype qui étend un état. Un MetaTransition est un stéréotype qui étend une transition.

Cependant, le système EPI est représenté par un stéréotype Software System qui étend l'élément d'UML machine à états.

EPI présente ses caractéristiques qui sont :

- CpuUs : Cette propriété est à 50% et représente le taux maximal de consommation de chaque processeur du système.
- MemoryUs : Elle identifie le taux maximal de consommation de chaque mémoire du système. Sa valeur est à 45%.
- BandWidthUs : Cette propriété exprime le taux maximal d'utilisation de chaque bus du système. Sa valeur est à 30%.

Dans la suite, nous nous intéressons à l'architecture logicielle et celle matérielle du système.

6.3.1 Architecture logicielle du système

Dans cette sous-section, nous identifions les composants de chaque MetaMode du système. Pour chaque MetaMode identifié, nous nous intéressons aux composants, à leurs propriétés spécifiques, leurs contraintes structurelles.

MetaMode Self-EPI

L'architecture du MetaMode Self-EPI est composée de 6 composants qui sont :

- Le composant CaptCR pour récupérer les valeurs des critères à partir des capteurs du système
- Le composant TraitCR pour traiter ces valeurs de critères
- Le composant ChoicePrgm pour choisir l'enchainement des publicités
- Le composant StartPrgm pour démarrer le programme
- Le composant FollowPrgm pour respecter l'enchainement des publicités
- Le composant StopPrgm pour signaler la fin du programme

En effet, un composant est un stéréotype appelé **StructuredComponent** qui étend **Component**. Chaque composant est caractérisé par les valeurs de ses propriétés et ses contraintes structurelles.

La figure ci-dessous illustre l'architecture logicielle du MetaMode Self-EPI et les contraintes structurelles des composants. En effet, les propriétés structurelles sont représentées par le stéréotype **StructuralConsraint** qui étend **Constraint** (à gauche de la figure ci-dessous).

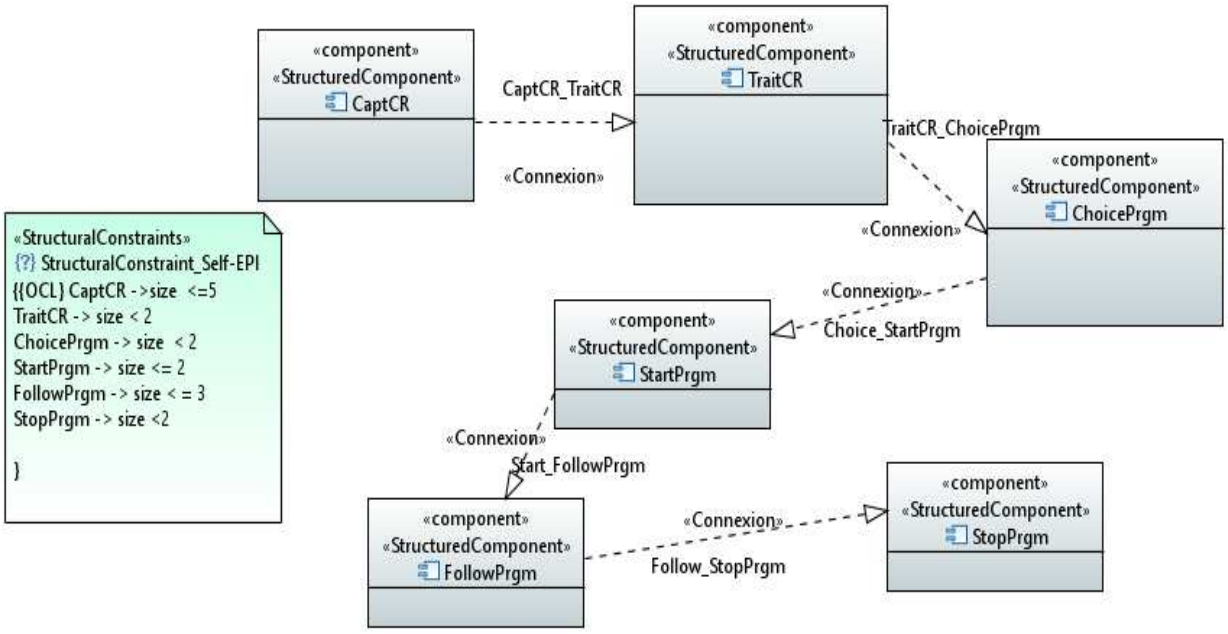

Figure 38 : L'architecture du MetaMode Self-EPI

Le tableau ci-dessous met en évidence les propriétés non fonctionnelles des composants structurés du MetaMode Self-EPI

Tableau 10 : Propriétés non fonctionnelles du MetaMode Self-EPI

Composants Structurés	Nature	Période (ms)	Pire Temps d'exécution (ms)	Consommation Mémoire(MB)
CaptCR	Périodique	500	30	2
TraitCR	Sporadique	100	75	2
ChoicePrgm	Sporadique	100	50	2.5
StartPrgm	Sporadique	100	30	1
FollowPrgm	Sporadique	100	80	2
StopPrgm	Sporadique	100	20	0 .6

MetaMode Control-EPI

L'architecture du MetaMode Control-EPI est semblable à celle du MetaMode Self-EPI avec quelques différences. En effet, Le composant CaptCR est remplacé par le composant ListCR qui présente les différents critères à l'utilisateur. Le composant ChoiceCR remplace TraitCR et permet d'enregistrer le choix de l'utilisateur.

La figure ci-dessous représente l'architecture du MetaMode Control-EPI

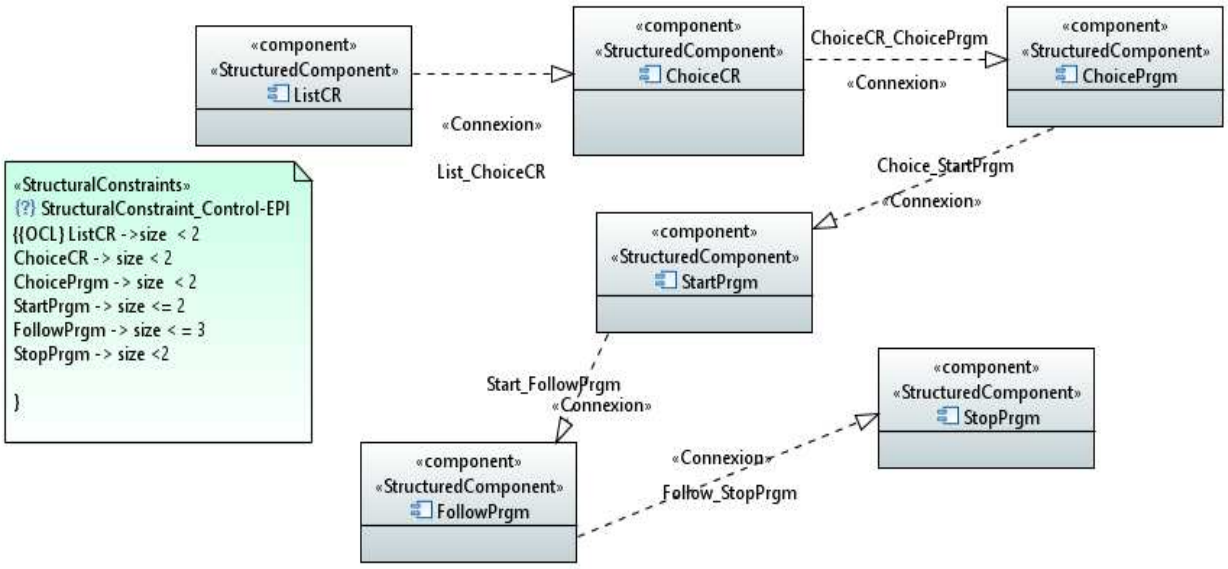

Figure 39 : L'architecture du MetaMode Control-EPI

Le tableau 8 ci-dessous met en évidence les propriétés non fonctionnelles des composants structurés du MetaMode Control-EPI

Tableau 11 : Propriétés non fonctionnelles du MetaMode Control-EPI

Composants Structurés	Nature	Période (ms)	Pire Temps d'exécution (ms)	Consommation Mémoire (MB)
ListCR	Périodique	500	50	2
ChoiceCR	Sporadique	100	20	0 .8
ChoicePrgm	Sporadique	100	30	4
StartPrgm	Sporadique	100	30	1
FollowPrgm	Sporadique	100	80	2
StopPrgm	Sporadique	100	20	0 .6

6.3.2 Architecture matérielle du système

Dans cette sous-section, nous identifions l'architecture matérielle de notre système EPI. Cette architecture simplifiée comprend quatre entités principales qui sont : la plateforme des capteurs, l'écran, le contrôleur, le système central.

En utilisant le sous-profil HRM de MARTE, chaque entité comprend différents composants qui sont : mémoires, processeurs, bus. Nous identifions aussi les valeurs des propriétés de chaque composant utilisé.

Le tableau ci-dessous met en évidence les différentes entités du système et les caractéristiques de leurs composants électroniques :

Tableau 12 : Entités du système et leurs caractéristiques

Entités	Processeur		RAM	Bus
	Nombre	Fréquence		
Plateforme des Capteurs	1 1	900 MHz 750 MHz	80MB	300Mb/s
Système Central	1 1	800MHz 900MHz	50MB 80MB	500Mb/s 300Mb/s
Ecran	1	750 MHz	100MB	400Mb/s
Contrôleur	1	800MHz	40MB	300Mb/s

La **figure 38** ci-dessous illustre l'architecture matérielle du système central d'EPI.

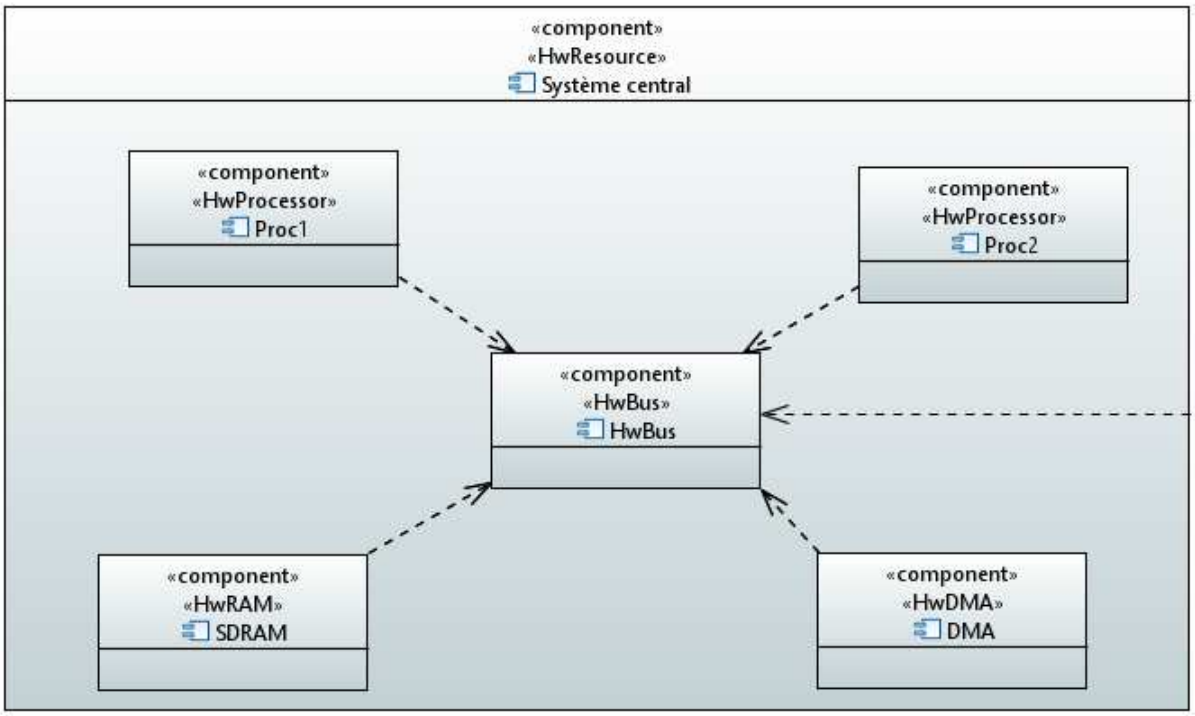

Figure 40 : L'architecture matérielle du système central

6.3.3 Allocation

Dans cette sous-section, nous nous intéressons à l'allocation des MetaModes sur l'architecture matérielle décrite. Ainsi, le MetaMode Control-EPI s'appuie sur une architecture matérielle bien définie et des caractéristiques d'allocation. La **figure 41** ci-dessous s'intéresse à l'allocation du MetaMode Control-EPI et ses caractéristiques

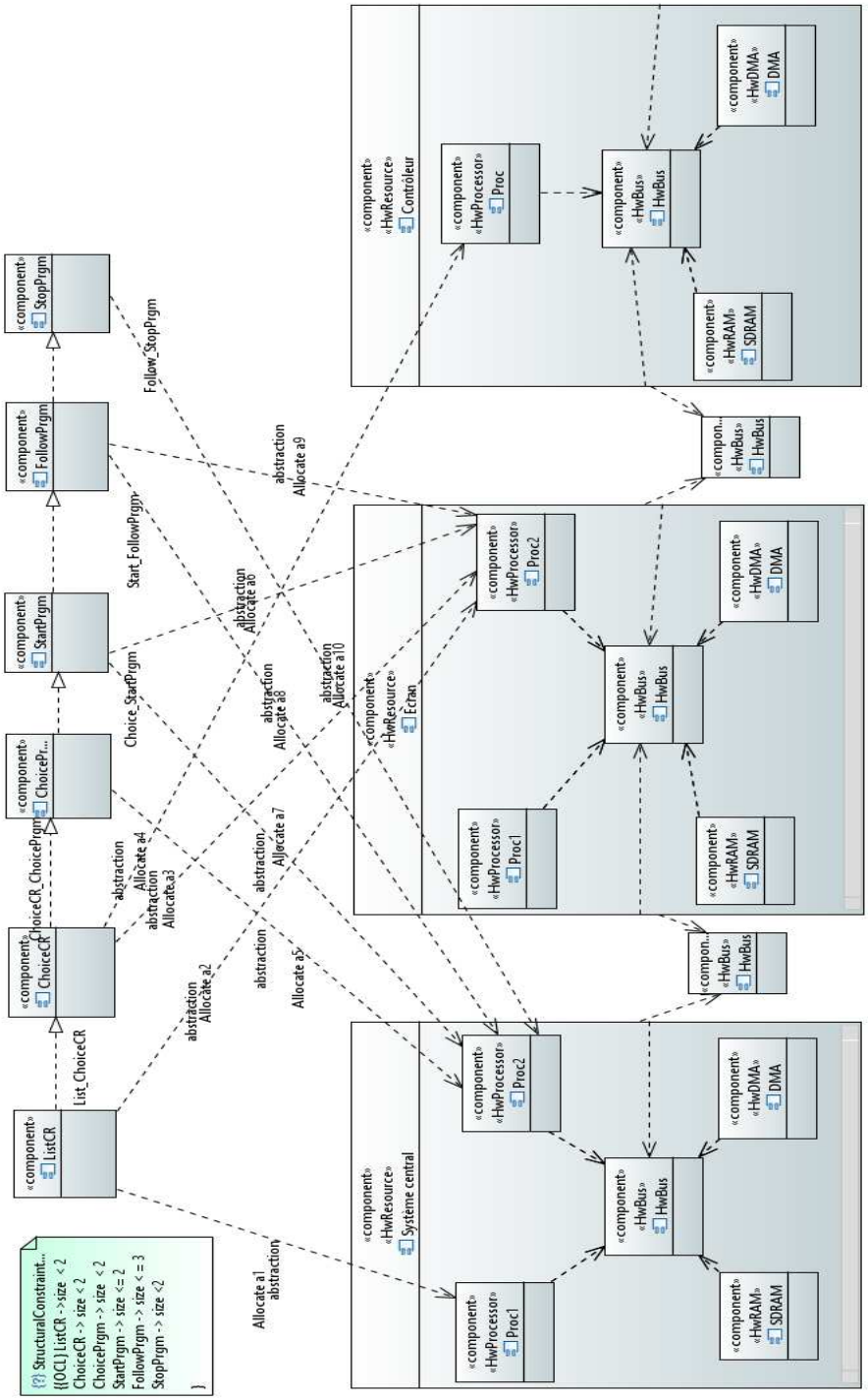

Figure 41 : L'allocation du MetaMode Control-EPI à l'architecture matérielle des composants matériels

6.4 Vérification des propriétés non fonctionnelles du système

En effet, cette section est consacrée à la vérification du respect des propriétés non fonctionnelles du système avec notre framework Cheddar. Ainsi, l'évaluation de performance de notre système fait l'objet d'un chapitre entier **Chapitre 8.**

Le fichier de simulation du framework Cheddar est un projet .xml (cf **Figure 42**). Ce fichier renferme les caractéristiques (nom, protocole, priorité, capacité, …) des différents composants (processeur, tâche, …) de notre système EPI.

```
<?xml version="1.0" encoding="utf-8"?>
<cheddar>
 <core_units>
  <core_unit id="id_29">
   <object_type>CORE_OBJECT_TYPE</object_type>
   <name>core_Controleur</name>
   <scheduling>
    <scheduling_parameters>
     <scheduler_type>POSIX_1003
_HIGHEST_PRIORITY_FIRST_PROTOCOL</scheduler_type>
     <quantum>0</quantum>
     <preemptive_type>PREEMPTIVE</preemptive_type>
     <capacity>0</capacity>
     <period>0</period>
     <priority>0</priority>
     <start_time>0</start_time>
    </scheduling_parameters>
   </scheduling>
   <speed>0.00000</speed>
  </core_unit>
  <core_unit id="id_28">
```

Figure 42 : Format du fichier descriptif .xml de Cheddar

Les simulations effectuées avec Cheddar nous permettent de vérifier nos contraintes non fonctionnelles.

L'absence d'interblocage, l'ordonnancement et le respect des échéances des tâches au niveau du Système central peuvent faire l'objet de simulation (Cf. **Figure 43**).

Scheduling simulation, Processor cpu_SC1 :

- Task response time computed from simulation :

  ChoicePrgm => 4/worst 4/best 4.00000/average

  ListCR => 6/worst 6/best 6.00000/average

- No deadline missed in the computed scheduling : the task set is schedulable.

Scheduling simulation, Processor cpu_SC1 :

- Blocking time from simulation :

  ChoicePrgm => 0/worst 0/best 0.00000/average

  ListCR => 0/worst 0/best 0.00000/average

Scheduling simulation, Processor cpu_SC2 :

- Task response time computed from simulation :

  StartPrgm => 2/worst 2/best 2.00000/average

  StopPrgm => 1/worst 1/best 1.00000/average

- No deadline missed in the computed scheduling : the task set is schedulable.

Scheduling simulation, Processor cpu_SC2 :

- Blocking time from simulation :

  StartPrgm => 0/worst 0/best 0.00000/average

Figure 43 : Vérification d'interblocage, de l'ordonnancement et le respect des échéances des tâches au niveau du Système Central avec Cheddar

En effet, la vérification ci-dessous met en évidence certaines erreurs de modélisation. Elle nous permet de revenir la modélisation du composant matériel Ecran. Ce composant peut se limiter à un seul processeur.

La consommation des CPU peut être vérifiée (Cf **Figure 44**).

Address space Memory Footprint Analysis :

ea_SC1 => Stack = 6

ea_SC2 => Stack = 2

ea_Ecran => Stack = 2

ea_Controleur => Stack = 1

Figure 44 : La consommation des CPU avec le framework Cheddar

Les adresses espaces ea_SC1, ea_SC2, ea_Ecran et ea_controleur sont respectivement allouées aux processeurs cpu_SC1, cpu_SC2, cpu_Ecran et cpu_controleur.

Ainsi, la consommation totale de chaque CPU par leurs tâches affectées est bien obtenue.

6.5 Construction automatique d'une partie de notre système EPI

En effet, cette étape nous permet d'exécuter la transformation M2M des différents composants de notre système EPI via notre outil ATL. Elle correspond aussi la transformation M2T de notre système via l'outil Acceleo.

L'étape de la transformation M2M nous a permis de générer un fichier .xmi. Ce fichier ci-dessous montre les différents composants, leurs tâches allouées et les relations existantes entre eux.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MyEPI:EPI2 xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:MyEPI="www.myepi.com"
xsi:schemaLocation="www.myepi.com ../MetaModels/MyEPI_MetaModel.ecore" Name="EPI">
  <Components Id_Component="Capteurs" Name_Component="Capteurs">
    <Component_PThread Name_PThread="CaptCR"/>
  </Components>
  <Components Id_Component="Systeme_Central" Name_Component="Système Central">
    <Component_SThread Name_SThread="ChoicePrgm"/>
    <Component_SThread Name_SThread="StartPrgm"/>
    <Component_SThread Name_SThread="FollowPrgm"/>
    <Component_SThread Name_SThread="StopPrgm"/>
  </Components>
  <Components Id_Component="Ecran" Name_Component="Ecran">
    <Component_PThread Name_PThread="ListCR"/>
  </Components>
  <Components Id_Component="Controleur" Name_Component="Contrôleur">
    <Component_SThread Name_SThread="ChoiceCR"/>
  </Components>
  <Relation Connexions="//@Components.0 //@Components.1"/>
  <Relation Connexions="//@Components.1 //@Components.2"/>
  <Relation Connexions="//@Components.1 //@Components.3"/>
</MyEPI:EPI2>
```

Figure 45 : Fichier généré 'MyEPI_Model2.xmi' ouvert sous éditeur texte simple

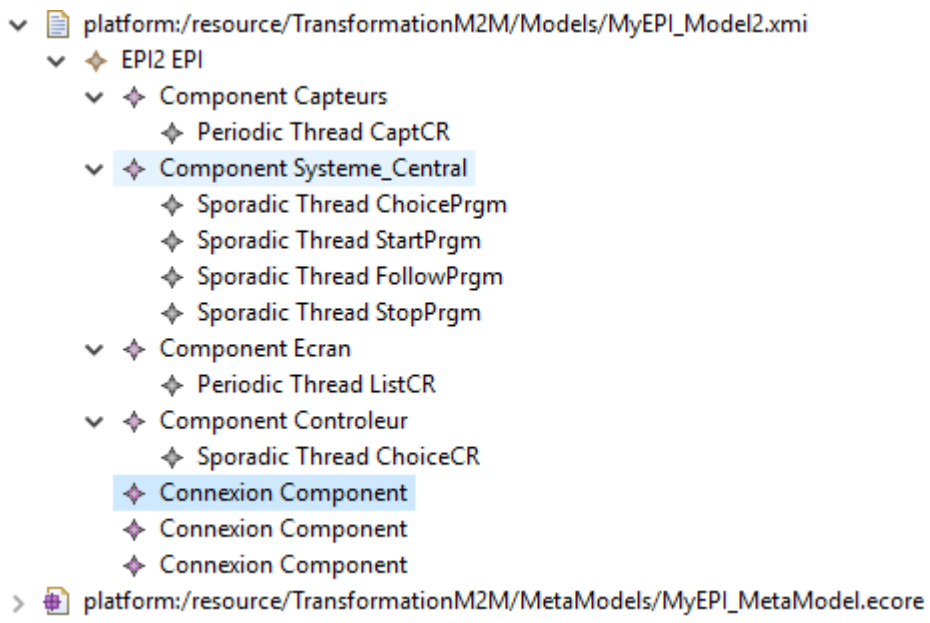

Figure 46 : Fichier généré 'MyEPI_Model2.xmi' ouvert sous éditeur des fichiers Ecore

Enfin, l'étape qui s'en suit est la génération automatique du code de notre système EPI avec l'outil Acceleo. En effet, cette génération de code permet de faciliter la mise en œuvre du système. Ainsi, la classe la plus fondamentale EPI2 (cf. **Figure 47**) est générée simultanément avec d'autres classes du système EPI.

```
public class EPI2 {

    java.lang.String Name;

    Component Components;
    Connexion_Component Relation;

    public java.lang.String getName () {
        return this.Name;
    }

    public void setName (java.lang.String Name) {
        this.Name = Name;
    }

    public Component getComponents () {
        return this.Components;
    }
    public Connexion_Component getRelation () {
        return this.Relation;
    }

    public void setComponents (Component Components) {
        this.Components = Components;
```

Figure 47 : Fichier 'EPI2.java' généré

6.6 Conclusion

D'abord, nous nous sommes intéressés à l'analyse et conception de notre système simplifié nommé EPI. L'analyse d'EPI nous a permis de modéliser les deux principaux MetaModes du système, son architecture logicielle, son architecture matérielle, l'allocation des MetaModes sur les composants matériels. Ensuite, vient l'étape de vérification des contraintes non fonctionnelles du système avec notre framework Cheddar. Enfin, la transformation M2M des différents composants de notre EPI et la génération automatique de code nous ont permises de clore ce chapitre.

Chapitre 7 : Evaluation de performances

7.1 Introduction

De nos jours, les systèmes embarqués à temps réel se trouvent face à leur évaluation de performance. En effet, cette évaluation s'intéresse à certains critères qui sont :

- Les temps de réponse de systèmes en fonction du nombre d'utilisateurs, en fonction des types de tâches, des types de ressources (processeurs, mémoires, …)
- Le taux d'utilisation d'un système
- Les temps d'attente
- La disponibilité des ressources d'un système

Ainsi, nous essayerons de nous intéresser dans la section **7.2** aux méthodes, concepts, modèles à notre portée afin d'élaborer l'évaluation de performance des systèmes à élaborer.
La section **7.3** met en évidence notre modèle de système implémenté, ses résultats de simulations et interprétations. Enfin, la section **7.4** nous permet de conclure ce chapitre.

7.2 Méthodes d'évaluation de performance des systèmes

L'évaluation de performance du système embarqué est une étape décisive dans sa conception. Elle permet au concepteur de :

- Améliorer à temps son système conçu avant son implémentation
- Prévoir les limites du système
- Le comparer à d'autres systèmes existants

Elle s'effectue par une modélisation du système. En effet, le modèle n'est qu'une abstraction du système qui essaie de se rapprocher le plus possible de la réalité.
Dans la suite de cette section nous nous intéressons aux différentes méthodes de modélisation de système.

7.2.1 Réseaux de file d'attente

Le réseau de file d'attente comprend un ensemble de files d'attentes interconnectées. En effet une file d'attente est caractérisée par la durée moyenne entre les arrivées et le taux d'un service moyen.
On distingue deux types de réseau de file d'attente :
Réseaux monoclasses : Les clients qui circulent dans le réseau sont indifférenciables.
Réseaux multiclasses: Les clients de ce type de réseau appartiennent à des classes différentes. En effet, ces classes se caractérisent par des propriétés divergentes au niveau des clients :
Taux d'arrivée différents, taux de services différents, ordonnancement différents.

Les réseaux monoclasses peuvent être ouverts ou fermés. Cependant, dans les réseaux multiclasses une classe de clients est soit ouverte ou fermée.

La complexité des systèmes à concevoir provoque l'utilisation de nouveaux formalismes. Ainsi, ces formalismes permettent de mettre en évidence toutes les propriétés de ces systèmes.

7.2.2 Réseaux de pétri (RdP)

Le RdP est un formalisme introduit par Carl Adam Pétri en 1962. Il permet de modéliser les systèmes avec concurrence, parallélisme, synchronisation. En effet, les réseaux de pétri sont utilisés dans plusieurs domaines (Informatique distribué, Télécommunications, Transport, …). Ce formalisme a intégré plusieurs améliorations (Introduction de notion de temps, de classes, de notion stochastique, de notion d'arc inhibiteur, …).

Un RdP est représenté graphiquement par différents composants qui sont :

- Place : sous forme de cercle, constitue une condition ou ressource du système.
- Jeton : sous forme de petit disque représenté toujours à l'intérieur d'une place. Il représente une valeur spécifique d'une condition ou d'une ressource du système.
- Transition : sous de rectangle ou trait continu, représente une activité du système. Il permet de d'exécuter une activité et changer les valeurs de condition ou de ressource du système.
- Arc : sous forme de flèche orienté, représente l'interconnexion entre place et transition.

Il relie toujours place et transition (l'arc ne relie jamais deux places ou deux transitions)

Un RdP est défini par le quadruplet (P, T, Pré, Post, Inh).

P= {p1, p2, …, pn} l'ensemble des places

T= {t1, t2, …, tn} l'ensemble des transitions

En effet, ces deux ensembles vérifient ces propriétés $P \cup T = \emptyset$ $P \cap T = \emptyset$

Les matrices Pré, Post, Inh sont des matrices d'incidence avant, arrière et inhibitrice.

7.2.3 Réseaux de pétri stochastiques (SPN)

En effet, la notion de temps peut être intégrée dans les composants du RdP :

- Jeton : le temps correspond à la durée pendant laquelle le jeton est disponible à accomplir un franchissement
- Place : le temps identifie le séjour obligatoire de(s) jeton(s) dans une place avant utilisation
- Transition : le temps détermine la durée d'exécution d'une activité

- Arc : le temps correspond au délai de traversé d'un jeton d'une place à une transition (ou d'une transition à une place)

Ainsi, dans un SPN chaque transition ti a un délai de franchissement w_i.

Un SPN est ainsi défini :

SPN= {P, T, Pré, Post, m_0, W} avec m_0 le marquage initial du réseau

W= {w_0, w_1, …, w_n} l'ensemble des délais de franchissement des transitions

Ainsi, l'utilisation de transitions immédiates (dont leur délai de tir est nul) dans un SPN introduit un GSPN (General Stochastic Petri Net).

Au fur et à mesure, le formalisme a subi des modifications qui permettent au concepteur de se rapprocher du système réel et de son comportement. Ainsi, nous allons nous intéresser à une amélioration bénéfique du GSPN.

7.2.4 Réseaux de pétri stochastiques Bien Formés (SWN)

En effet, le SWN est le formalisme qui permet d'introduire la notion de classe de couleurs dans les ressources du système. La classe de couleurs est un regroupement d'objets d'une ressource partageant des propriétés communes. Ainsi, Chaque ressource du système peut avoir une à plusieurs classes de couleurs. En effet, une classe de couleurs comprend une à plusieurs sous-classes statiques. Une classe statique comprend un nombre défini d'objets ayant les mêmes caractéristiques. Le SWN permet aux concepteurs de se rapprocher de la réalité du système, d'y effectuer des simulations, de prévoir le comportement et d'y apporter des modifications au besoin. Grâce à l'environnement Greatspn, le concepteur peut concevoir son système en SWN et récupérer ses résultats de simulation afin de les interpréter. Dans la section 8.3, nous nous intéressons à notre SWN du système EPI et ses résultats de simulations.

7.3 Modèle SWN et calculs de performance du système EPI

A partir de la description des composants d'EPI, nous essayons d'identifier l'ensemble des places (**Tableau13**) et l'ensemble des transitions (**Tableau14**) de notre modèle.

Tableau 13 : Liste des places du système

PLACES (RESSOURCES, ETATS DU SYSTEME)	DESCRIPTION
User	L'utilisateur du système
Cmde	Le système est en attente de la disponibilité de ses ressources (processeurs)
Processor1	Le processeur de type 1 plus performant qui exécute la tâche T1 définie
Processor2	Le processeur de type 2 moins performant qui exécute la tâche T2 définie
Task	La tâche à executer
InProcessingTask1	L'exécution en-cours de tâche de type T1
InProcessingTask2	L'exécution en-cours de tâche de type T2
DetecteStateProc	

Tableau 14 : Liste des transitions du système

TRANSITIONS (ACTIVITIES)	DESCRIPTIONS	RESSOURCES NECESSAIRES
Activate	Activation par une touche, un piston (Par exemple: Activer sur la touche pour démarrer le système)	User
SynchronizeTask1	Activation de l'exécution de tâche T1	Cmde+ Processor1+Processor2
SynchronizeTask2	Activation de l'exécution de tâche T2	Cmde+ Processor2
EndTaskT1	Fin de l'exécution de tâche T1	InProcessingTask1
EndTaskT2	Fin de l'exécution de tâche T2	InProcessingTask2
Detect_Task	Detection de nouvelle tâche	DetectStateProc
StopProcedure	Fin de l'exécution de la procédure	DetectStateProc

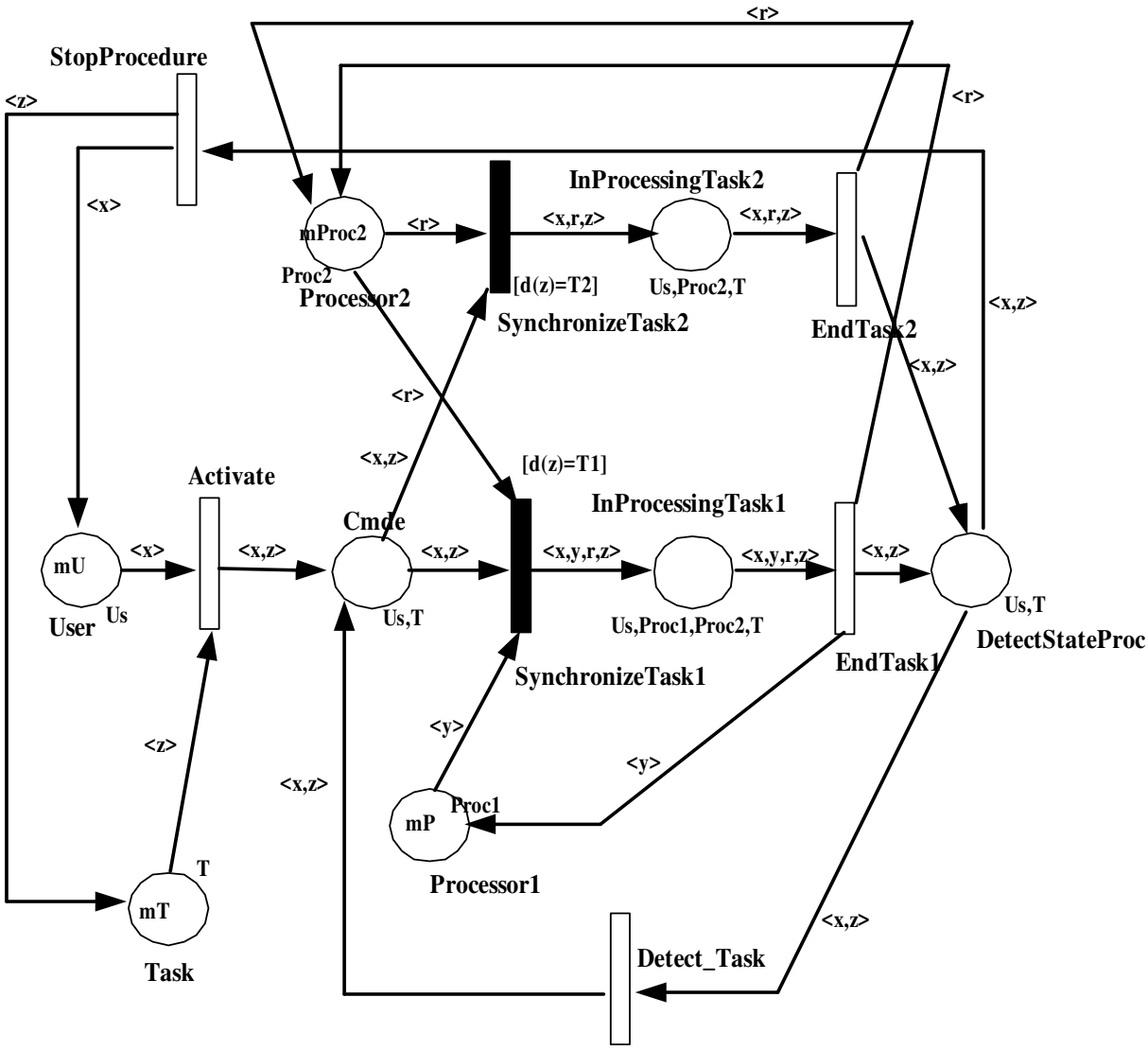

Figure 48: SWN du système EPI

L'utilisateur active le système (l'ensemble est représenté par la place User et la transition Activate)

Les propriétés de notre SWN sont :

- P= {User, Cmde, Task, Processor1, Processor2, InProcessingTask1, InProcessingTask2, DetectStateProc }
- T= {Activate, SynchronizeTask1, SynchronizeTask2, EndTask1, EndTask2, Detect_Task, StopProcedure }
- M0= {mU, mT,0, mProc1, mProc2, 0, 0, 0}

Nous allons nous intéresser à la variation des temps de réponse moyen des tâches périodiques en fonction du nombre d'utilisateurs

Tableau 15 : Résultats de simulations du système (Tâches périodiques)

Nombre de jetons 'User'	Débit de 'Activate'	Nombre moyen de jetons 'User'	Nombre moyen de jetons 'Cmde'	Nombre moyen de jetons 'InProcessingTask1'	Nombre moyen de jetons 'DetectStateProc'	Temps de réponse (Exécution de tâche périodique en ms)
4	0,647	0,0002	0,01	0,484	0,657	1,778
8	1,038	0,0004	0,052	0,726	1,04	1,984
16	1,366	0,0006	0,658	0,941	1,394	2,191
20	0,779	0,0006	0,815	0,533	0,776	2,72
24	1,383	0,0007	2,027	0,95	1,392	3,159

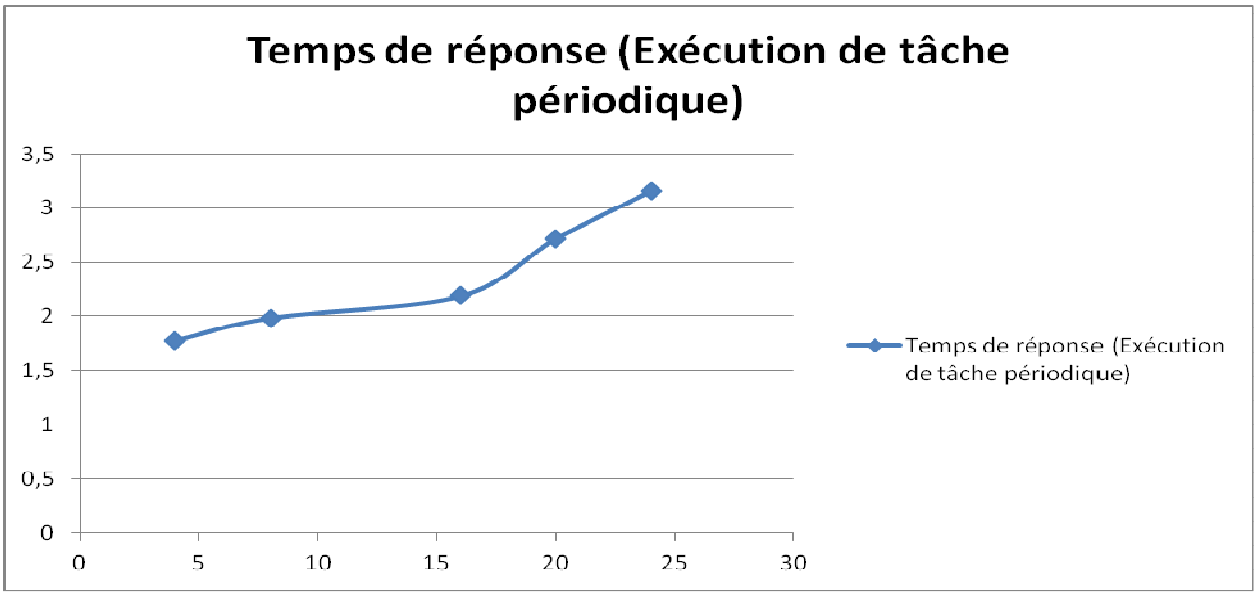

Figure 49 : Evolution du temps de réponse en ms (Tâche périodique)

Nous allons nous intéresser à la variation des temps de réponse moyen des tâches sporadiques en fonction du nombre d'utilisateurs

Tableau 16 : Résultats de simulations du système (Tâches sporadiques)

Nombre de jetons 'User'	Débit de 'Activate'	Nombre moyen de jetons 'User'	Nombre moyen de jetons 'Cmde'	Nombre moyen de jetons 'InProcessingTask2'	Nombre moyen de jetons 'DetectStateProc'	Temps de réponse (Exécution de tâche sporadique en ms)
4	0,779	0,0002	0	2,131	0,711	4,008
8	1,082	0,0004	0	3,124	1,051	4,258
16	1,615	0,0006	0,704	4,685	1,926	4,529
20	1,915	0,0006	2,549	5,427	1,892	5,153
24	1,824	0,0007	2,813	5,008	1,8	5,274

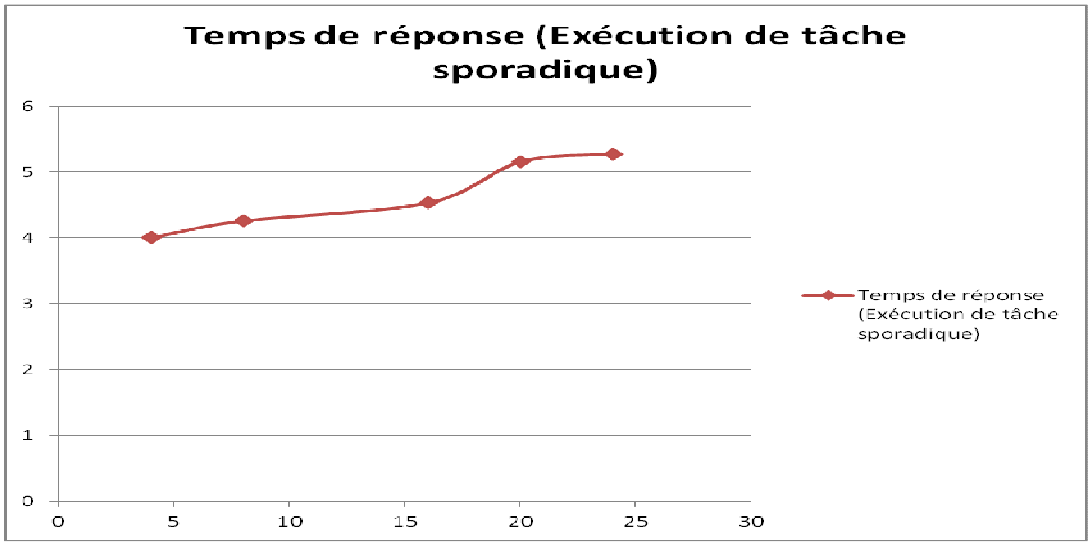

Figure 50 : Evolution du temps de réponse en ms (Tâche sporadique)

Les résultats ci-dessus (**Figure49** et **Figure50**) nous démontrent que l'exécution d'une tâche périodique est plus rapide que celle d'une tâche sporadique. En effet, la tâche périodique sollicite plus de ressources matérielles (1 processeur plus performant et 1 processeur moyen) pour son exécution.

Tableau 17 : Résultats des transitions (Dectect_Task et StopProcedure)

N.User	X(DetectTask)	X(StopP)
4	2,702	1,356
8	3,916	2,374
16	5,476	2,981
20	5,53	3,095
24	5,584	3,209

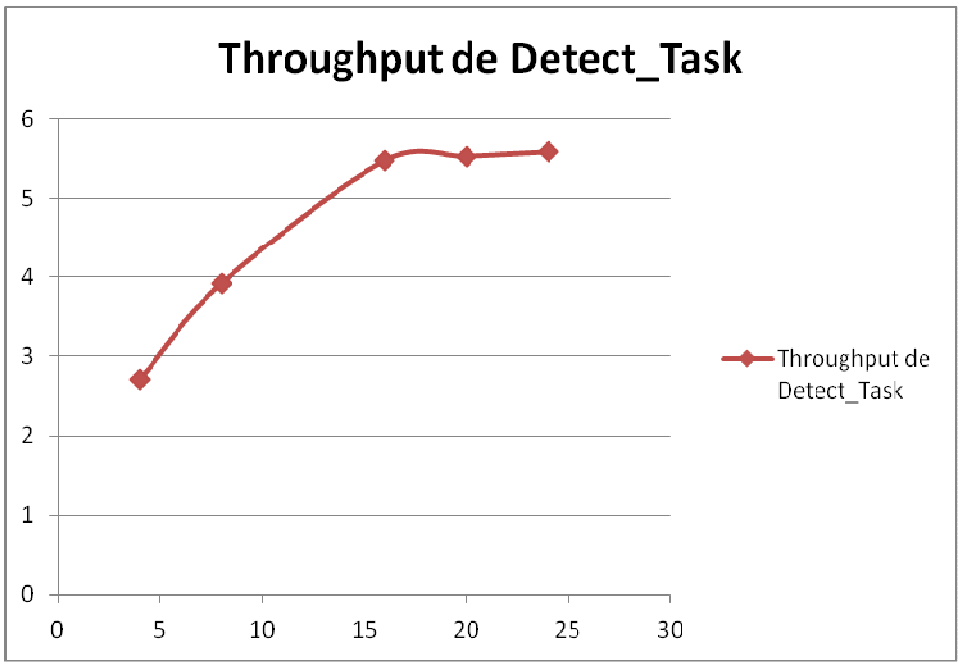

Figure 51 : Evolution de la Transition 'Detect_Task' Throughput

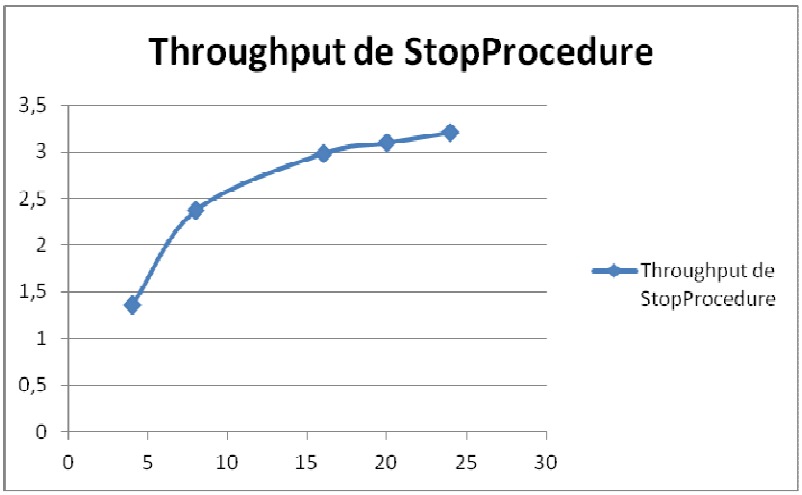

Figure 52 : Evolution de la Transition 'StopProcedure' Throughput

A partir de ces résultats (**Figure51** et **Figure 52**), le throughput de 'Detect_Task' est plus important que celui de 'StopProcedure'. Ainsi, notre EPI se retrouve fréquemment dans un scénario où la procédure est en-cours d'exécution.

Enfin, nous pourrons nous intéresser à l'évolution du nombre moyen de tâches en attente d'exécution dans la place 'Cmde' en fonction du nombre d'utilisateurs.

Tableau 18 : Evolution des tâches en attente

N.User	Nbre Moyen Tâches en Attente
4	0,01
8	0,052
16	1,362
20	3,364
24	4,84

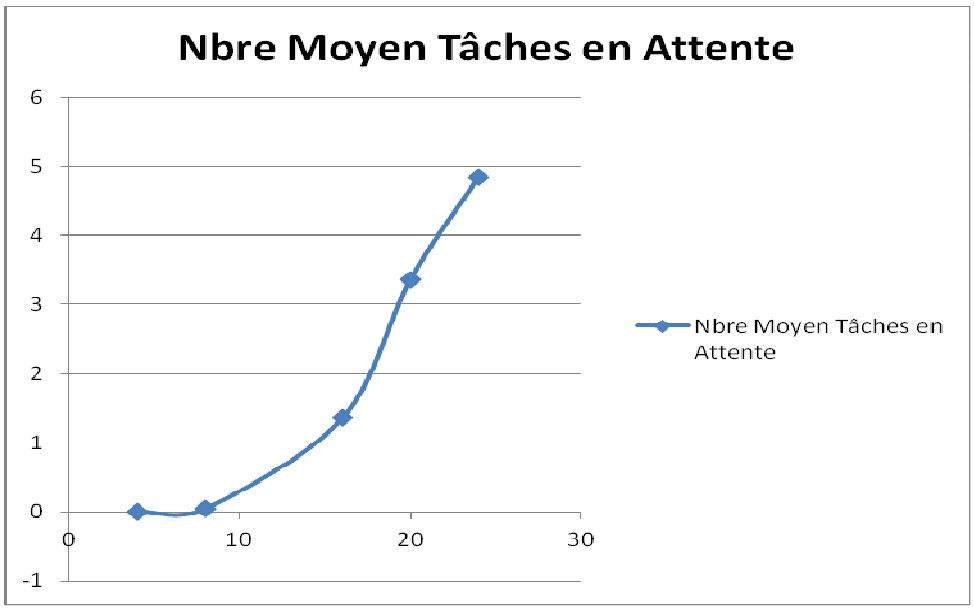

Figure 53 : Evolution des tâches en attente dans 'Cmde'

A partir des résultats de **Figure 53**, nous constatons une évolution croissante considérable des tâches en attente d'exécution. L'indisponibilité des ressources (processeurs) provoque cette croissance. En effet, ces ressources sont utilisées dans l'exécution des tâches en-cours.

7.4 Conclusion

Dans ce chapitre, nous avons établi l'étude de performance de notre système EPI. Cette étude nous permet d'avoir une appréhension claire du fonctionnement du système face à des contraintes d'utilisation (nombre d'utilisateurs, disponibilité des ressources, …). Ainsi, nous nous intéressons à la fiabilité et disponibilité **[Fat13B]** de notre système qui constitue un point focal dans sa conception.

Conclusion générale et Perspectives

Dans ce chapitre, nous élaborons un rappel de notre problématique et revoyons nos principales contributions et les perspectives qui y sont appréhendés.

Rappel de problématique

De nos jours, les systèmes embarqués temps-réels font partie de notre quotidien. Notre monde est caractérisé par l'utilisation de ces systèmes plus connus sous le nom d'objets connectés. Ainsi, les industries de conception de ces systèmes se trouvent face à la demande croissante et exigeante des utilisateurs. En effet, la conception de ces systèmes est une tâche minutieuse car ils sont dotés de fonctionnalités soumises à des contraintes non fonctionnelles à respecter (Consommation CPU, échéances des tâches, famine des ressources…). Le concepteur de ces systèmes devra modéliser l'architecture matérielle, celle logicielle soumise à des contraintes. Il s'intéressera aussi à l'allocation des composants logiciels sur la partie matérielle. En effet, les contraintes devront être prises en compte et validées dans la conception.
La reconfiguration dynamique de ces systèmes constitue une étape transitoire du système qui lui permet de passer d'un état à un autre et vice-versa. Cette étape transitoire devra respecter les contraintes imposées au système. Ainsi, le concepteur met en évidence les différents états du système, ses transitions.

Contributions

Dans le cadre de cette thèse, nous avons élaboré un processus de développement des systèmes embarqués temps réel s'appuyant sur l'ingénierie dirigée par les modèles. Ainsi, notre première contribution est l'introduction un nouveau méta-modèle M4DRES (Modeling For Dynamically Reconfigurable Embedded Systems) avec de nouveaux concepts : MetaMode et MetaTransition. En effet, un MetaMode représente un état du système à un instant de son fonctionnement. Cependant, un MetaTransition permet de modéliser le passage d'un état à un autre. M4DRES est une extension d'UML.
Notre processus comprend 4 grandes étapes qui sont : la modélisation, la vérification, l'étude de performance, la génération de code.

- La modélisation permet au concepteur d'élaborer des modèles qui mettent en évidence l'architecture matérielle, celle logicielle, l'allocation de la partie logicielle sur les

composants matériels miniaturisés et les contraintes imposées. Elle est faite grâce aux concepts de notre méta-modèle M4DRES,

- La vérification s'intéresse au respect des contraintes non fonctionnelles (respect des délais des tâches, consommation de la mémoire, du CPU, famine des ressources, …) avec l'outil cheddar. Le non-respect d'une de ces contraintes impose la révision de nos modèles précédemment obtenus,
- L'étude de performance est réalisée avec les réseaux stochastiques bien formés via l'outil GreatSPN. Elle est une étape importante dans la conception qui nous permet de prévoir les limites du système et d'y apporter des corrections au besoin,
- Enfin, la génération de code avec l'outil Acceleo permet de faciliter la mise en œuvre du système.

La deuxième contribution consiste à la plateforme d'exécution ESDRES (Execution Support For Dynamically Reconfigurable Embedded Systems) qui est une extension de PolyORB_HI. Elle permet d'assurer la cohérence du système durant son exécution (en-cours d'exécution, avant et après la reconfiguration).

La troisième contribution constitue le nouveau processus de développement des systèmes embarqués qui intègre les deux premières précédentes.

Perspectives

A partir de nos travaux réalisés, nous pourrons identifier l'absence d'un unique outil ou environnement de conception des systèmes embarqués qui pourra effectuer les différentes étapes de notre processus de développement proposé.

Ainsi, nos futurs travaux de recherche pourront s'intéresser à l'implémentation d'outil graphique de développement des systèmes embarqués dynamiquement reconfigurables. Cet outil devra respecter les étapes de développement passant de la modélisation des composants matériels et logiciels jusqu'à la génération automatique de code de ces objets connectés.

Bibliographie

[Agr96] R Agrawal, H Mannila, R Srikant. Fast discovery of association rules. In Advances in Kowledge Discovery and Data Mining, pages 307–328, 1996

[And08] Andreas Rasche and Andreas Polze. ReDAC dynamic reconfiguration of distributed component-based applications with cyclic dependencies. In Proceedings of the 2008 11th IEEE Symposiumon Object Oriented Real-Time Distributed Computing, pages322–330, Washington, DC, USA, 2008. IEEE Computer Society

[Bec08] Bechir Zalila. Thèse : Configuration et déploiement d'applications temps-réel réparties embarqués à l'aide d'un langage de description d'architecture. Ecole Nationale Supérieure des Télécommunications, 2008

[Ben06] Benoit Baudry Jean-Marc Jézéquel, Sébastien Gérard Lavoisier. Le génie logiciel et l'IDM: une approche unificatrice par les modèles, Hermes-science, 2006

[Bent04] F. Bentayeb, J .Darmont, C Udréa. Efficient integration of data
mining techniques in dbmss. In 8th International Database Engineering and Appli-
cations Symposium (IDEAS 04), Portugal, 2004

[Bou07] B. Bouchon-Meunier. La logique floue, Que-sais-je? PUF, N° 2702, 2007

[Bou95] B. Bouchon-Meunier. La logique floue et ses applications, Addison Wesley ed., 1995

[Bou98] B. Bouchon-Meunier, L. Foulloy et M. Ramdani. Logique floue – exercices corrigés et exemples d'applications, Cépaduès ed. , 1998

[Bre84] Breiman L., Friedman J., Olshen R., and Stone C. Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA, 1984

[Cél04] Céline Lafontaine. L'Empire cybernétique. Des machines à penser à la pensée machine, Le Seuil, Paris, 2004

[Chau00] J H Chauchat , R Rakotomalala. A new sampling strategy for building decision trees from large databases. In 7th Conference of the International Federation of Classification Societies (IFCS 00), Belgium, 2000

[Chm03] Chmidt Douglas, Gokhale Aniruddha, Natarajan Balachandran, Neema Sandeep, Bapty Ted, Parsons Jeff, Gray Jeff, Nechypurenko Andrey, and Wang Nanbor. An mda generative tool for distributed real-time and embedded component middleware and applications. In Proceedings Of The Oopsla 2002 Workshop On Generative Techniques In The Context Of Model Driven Architecture. ACM, 2003

[Chri92] C.Christment, G. Cabanac, K. Pinel-Sauvagnat, O. Teste, M. Tuffery. Bases de données orientées-objet, Collection Informatique, Paris, 1992

[Com] Component Model Specification, http://www.omg.org/spec/#MW.

Unified Modeling Language (OMG UML), Superstructure, V2.4. Object Modeling Group, OMG, January 2011

[Dav14] Davide Brugali. Simulation, modeling, and programming for autonomous robot. In proceedings of 4th International Conference , SIMPAR 2014, Bergamo, Italy, October 20-23, 2014, Springer

[Dou01] Douglas C. Schmidt and Chris Cleeland. Applying a pattern language to develop extensible ORB middleware. Cambridge University Press, 2001

[Edw05] Edward E. Lee. Absolutely positively on time: what would it take? In Embedded Systems Column, IEEE Computer, 2005

[Fat13] Fatou Faye, Mbaye Sene. Datamining Tool: Multiple regression and Logistic regression in a Web platform of a datawarehouse. In Advanced Materials Research, Vol.2385 (694), pages 2299-2307 Trans Tech, 2013

[Fat13B] Fatou Faye, Mbaye Sene. Reliability and Availability of Embedded Software Architectures: A Survey. In Applied Mechanics and Materials, Vol.2594 (373), pages 1612-1617 Trans Tech, 2013

[Fat16] Fatou Faye, Mbaye Sene. Static Reconfiguration and Performance
Evaluation of embedded software architectures. In 2nd IEEE International Conference on Engineering and Technology (ICETECH), 17th & 18th March 2016, Coimbatore, TN, India

[Guo08] Guo Y, Sierszecki K, Angelov C. A (re)configuration mechanism for resource-constrained embedded systems. In Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference pages 1315–1320, IEEE Computer Society, Washington DC, USA, 2008

[Gui11] Serge Guillaume and Brigitte Charnomordic. Learning interpretable Fuzzy Inference Systems with FisPro. In International Journal of Information Sciences, doi:10.1016/j.ins.2011.03.025 , 181(20), pages 4409-4427, 2011

[Jag02] Jagun Kwon, Andy Wellings, and Steve King. Ravenscar-Java: a High Integrity Prole for Real-Time Java. In Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande, pages 131-140, New York, NY, USA, 2002

[Jai05] Jaiganesh Balasubramanian, Balachandran Natarajan, Douglas C. Schmidt, Aniruddha S. Gokhale, Je Parsons, and Gan Deng. Middleware Support for Dynamic Component Updating. In Proceedings of the OTM Conferences, pages 978-996, Springer, 2005

[Jur06] Juraj Polakovic, Ali Erdem Ozcan, and Jean-Bernard Stefani. Building Reconfigurable Component-Based OS with THINK. In Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pages 178–185, IEEE Computer Society ,Washington, DC, USA, 2006

[Kli88] G. Klir and T. Folger. Fuzzy sets, uncertainty and information, Prentice Hall ed., 1988.

[Las09] Lasnier G, Zalila B, Pautet L, Hugues J. Ocarina: an environment for aadl models analysis and automatic code generation for high integrity applications. In Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies pages237–250, Springer: Brest, France, 2009

[Lui90] Lui Sha, Ragunathan Rajkumar and John P. Lehoczky. Priority Inheritance Protocols : An Approach to real-time Synchronization. In IEEE transactions on computers, Vol.39 pages 1175-1185, September 1990.

[Mod11] Modeling and Analysis of Real Time Embedded Systems - version 1.1, Object Management Group,- formal/2011-06-022, June 2011

[Mou12] Mouhamed-Lamine Boukhanoufa. Thèse: Adaptabilité et reconfiguration des systèmes temps-réel embarqués, Université de Paris-Sud, Paris XI, 2012

[Nor14] Norbert Wiener, Pierre-Yves Mistoulon. Cybernétique et société : L'usage humain des êtres humains, Editions du Seuil, Paris, 2014

[Qui86] Quinlan, J. R. Induction of decision trees. In Machine Learning, Vol. 1 pages 81–106, 1986

[Rob13] Rob Pooley, Jennifer Coady, Christoph Schneider, Henry Linger, Chris Barry. Information Systems Development Reflections, Challenges and New Directions, NY Springer, 2013

[Sin04] F. Singhoff, J. Legrand, L. Nana, L. Marcé. Cheddar: a Flexible Real Time Scheduling Framework. In proceedings of International Conference on Special Interest Group on Ada (SIGAda), ACM, Atlanta, Georgia, USA, November 2004, DOI : http://dx.doi.org/10.1145/1032297.1032298

[Tur50] Turing A.M. Computing machinery and intelligence, Mind, 59, 433-460, 1950.

[Tho04] Thomas Vergnaud, Jérôme Hugues, Laurent Pautet, and Fabrice Kordon. PolyORB : A Schizophrenic Middleware to Build Versatile Reliable Distributed Applications. In Proceedings of the 9th Ada-Europe International Conference on Reliable Software Technologies, pages 106-119. Springer, June 2004

[Ven07] Venkita Subramonian, Gan Deng, Christopher Gill, Jaiganesh Balasubramanian, Liang-Jui Shen, William Otte, Douglas C. Schmidt, Aniruddha Gokhale, and Nanbor Wang. The Design and Performance of Component Middleware for QoS-enabled Deployment and Configuration of DRE Systems. In Journal of Systems and Software, Vol. 80 pages 641-794, May 2007

[Zad65] L. A. Zadeh. Fuzzy Sets. In Information and Control, Vol. 8 pages 338-358, 1965

[Zad68] L. A. Zadeh. Fuzzy Algorithm. In Information and Control, Vol. 12 pages 94-102, 1968

Webographie

[Web01] http://fr.wikipedia.org/wiki/Norbert_Wiener: Bibliographie du père de la cybernétique Norbert Wiener

[Web02] http://fr.wikipedia.org/wiki/Cybern%C3%A9tique: Définition de la cybernétique

[Web03] http://www.loebner.net/Prizef/TuringArticle.html: Article Imitation Game d'Alan Turing

[Web04] http://fr.wikipedia.org/wiki/Alan_Turing: Bibliographie du génie informaticien de l'intelligence artificielle Alan Turing

[Web05] https://fr.wikipedia.org/wiki/Test de Turing: Test de Turing

[Web06] http://fr.wikipedia.org/wiki/John_McCarthy: Bibliographie du père de l'intelligence artificielle John McCarthy

[Web07] http://fr.wikipedia.org/wiki/Marvin_Minsky: Bibliographie du cofondateur de l'intelligence artificielle Marvin Minsky

[Web08] https://fr.wikipedia.org/wiki/Recherche_op%C3%A9rationnelle: Définition de la recherche opérationnelle

[Web09] https://fr.wikipedia.org/wiki/Supervision_(informatique): Définition de la supervision en informatique

[Web10] https://fr.wikipedia.org/wiki/Aide_%C3%A0_la_d%C3%A9cision: Définition de l'aide à la décision

[Web11] https://fr.wikipedia.org/wiki/Exploration_de_donn%C3%A9es:Définition du Data Mining

[Web12] http://fr.wikipedia.org/wiki/ELIZA: Définition du programme informatique nommé ELIZA

[Web13] http://eliza.levillage.org/: Programme informatique ELIZA en français

[Web14] http://fr.wikipedia.org/wiki/Joseph_Weizenbaum : Bibliographie du père d'ELIZA Joseph Weizenbaum

[Web15] http://fr.wikipedia.org/wiki/Donald_Hebb: Bibliographie du physiologiste Donald Hebb

[Web16] http://deeplearning.cs.cmu.edu/pdfs/Hebb.1949.pdf: The organization behavior de Donald Heeb

[Web17] http://fr.wikipedia.org/wiki/Lisp Définition du premier langage impératif et fonctionnel Lisp

[Web18] http://fr.wikipedia.org/wiki/Prolog Définition du langage de programmation logique Prolog

[Web19]http://fr.wikipedia.org/wiki/String_Oriented_Symbolic_Language Définition du langage SNOBOL

[Web20] http://www.omg.org/spec/#Profile: Différents profils d'UML

[Web21] http://ptolemy.eecs.berkeley.edu/~eal/biog.html Bibliographie du chercheur Edward

A. Lee

Annexe

Dans cette partie, nous faisons des rappels sur des notions d'algorithmes et de lois mathématiques utilisés dans notre document.

Le coefficient de Gini

Le coefficient de Gini est une mesure statistique mettant en évidence la dispersion d'un critère sur une population donnée. Sa valeur varie entre 0 et 1, où 0 traduit l'égalité parfaite et 1 signifie l'inégalité totale.

Afin de déterminer le coefficient de Gini pour n tranches de population ayant des revenus différents, on utilise la formule de Brown :

Formule 17 : Le coefficient de Gini (La méthode de Brown)

$$G = 1 - \sum_{k=0}^{n-1} (X_{k+1} - X_k)(Y_{k+1} + Y_k)$$

Où X est la part cumulée de la population et Y la part cumulée du revenu

Afin de déterminer le coefficient de Gini pour n personnes ayant des revenus différents yi, on applique la formule ci-dessous :

Formule 18 : La deuxième méthode de calcul du coefficient de Gini

$$G = \frac{2\sum_{i=1}^{n} i y_i}{n\sum_{i=1}^{n} y_i} - \frac{n+1}{n}$$

Autre formulation du coefficient de G*ini* plus adaptée au contexte:

Formule 19 : Calcul du coefficient de Gini plus adapté au contexte

$$Gini(p) = 1 - \sum_{k=1}^{c} P^2(k|p)$$

D'où $P(k|p)$: Proportion des individus appartiennent à la classe k sachant qu'ils sont associés au nœud p

C : le nombre de classes

L'entropie de Shannon

L'entropie de Shannon permet de quantifier l'information contenue ou émise par une source donnée. En effet, si une source d'information émet toujours le même symbole'*', son entropie est nulle.

La formule du calcul de l'entropie notée H est la suivante :

Formule 20 : Calcul de l'entropie de Shannon

$$H_b(X) = -E\left[log_b P(X = x_i)\right] = \sum_{i=1}^{n} P_i \, log_b\left(\frac{1}{P_i}\right) = -\sum_{i=1}^{n} P_i \, log_b P_i$$

Avec

E : Espérance mathématique

log_b : Logarithme base b. En général b=2

X : la variable aléatoire

n : le nombre de symboles

xi : les différents symboles

Pi : la probabilité d'apparition de chaque symbole

Autre formulation de l'entropie de Shannon plus adaptée au contexte :

Équation 21 : Calcul de l'entropie plus adapté au contexte

$$\mathbf{Entropie(p)} = -\sum_{k=1}^{c} \mathbf{P(k|p)\ln P(k|p)}$$

D'où $P(k|p)$: Proportion des individus appartiennent à la classe k sachant qu'ils sont associés au nœud p

C : le nombre de classes

Le Gain d'information

Afin d'obtenir le gain d'information d'une variable explicative par rapport à une autre variable cible (nœud initial de l'arbre notée place P), nous appliquons la formule ci-dessous :

Équation 22 : Calcul du gain de l'information d'une variable explicative

$$Gain(p,t) = i(p) - \sum_{j=1}^{n} P_j \, i\,(P_j)$$

Avec t : la variable cible (le test)

n : le nombre de classes possibles

i : la fonction de mélangeance

P_j : la proportion des individus de la position p à une position p_j

Un intergiciel schizophrène

Un intergiciel est dit schizophrène lorsqu'il permet la cohabitation simultanée de plusieurs paradigmes de répartition au sein d'une même instance et met en place un mécanisme efficace d'interaction entre ces paradigmes (personnalités).

L'intergiciel POLYORB

POLYORB est la première implantation de l'architecture schizophrène. Il propose cinq personnalités applicatives (CORBA, DSA, AWS, MOMA et DDS) et trois personnalités protocolaires (GIOP, SOAP et SRP).

Liste des publications

Publication 1

- **Auteurs**: Fatou FAYE & Mbaye SENE
- **Titre:** *Static Reconfiguration and Performance Evaluation of embedded software architectures*
- **Publiée dans:** Proc. 2nd IEEE Conference and Technology (ICETECH)
- **Lieu** : India
- **Année de publication:** March 2016

Publication 2

- **Auteurs:** Fatou FAYE & Mbaye SENE
- **Titre:** *Datamining tool: Multiple regression and logic regression in a Web platform of datawarehouse*
- **Publié dans:** Journal of Advanced Materials Research, Vol. 694-697
- **Pages :** 2299-2307
- **Editeur:** Trans Tech Publication Switzerland
- **Année de publication:** 2013

Publication 3

- **Auteurs:** Fatou FAYE & Mbaye SENE
- **Titre:** *Reliability and Availability of embedded software architectures: a survey*
- **Publié dans:** The 2013 International Conference on Mechatronics, Robotics and Automation (ICMRA), China, Vol. 373-375,
- **Pages:** 1612-1617
- **Editeur:** ICMRA
- **Année de publication:** 2013