Nouvelle Approche Globale par Composants AADL pour le Développement des Systèmes Adaptatifs de Contrôle Industriel

Farid Adaili

ÉCOLE POLYTECHNIQUE DE TUNISIE, TUNISIE
CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS, FRANCE

École Doctorale : École Doctorale Sciences Appliquées

Laboratoires : LISI/INSAT, Tunisie et CEDRIC/CNAM, France

THÈSE DE DOCTORAT

présentée par : **Farid ADAILI**

soutenue le : **04 Mars 2017**

pour obtenir le grade de : **Docteur de l'École Polytechnique de Tunisie**

Spécialité : **Électronique et Technologie de L'Information et de la Communication**

Nouvelle Approche Globale par Composants AADL pour le Développement des Systèmes Adaptatifs de Contrôle Industriel

THÈSE dirigée par

M. Mohamed KHALGUI	*MdC/HDR, INSAT, Tunisie*
Mme. Samia BOUZEFRANE	*MdC/HDR, CNAM, France (Co-encadrante)*
Mme. Olfa MOSBAHI	*MA, INSAT, Tunisie (Co-encadrante)*

RAPPORTEURS

M. Mohamed ABID	*Prof, ENIS, Université de Sfax, Tunisie*
M. Alain PLANTEC	*Prof, Université de Bretagne Occidentale, France*

EXAMINATEURS

M. Nicolas TREVES	*Prof, CNAM, France*
Mme. Maryline CHETTO	*Prof, Université de Nantes, France*
M. Laurent GEORGE	*Prof, ESIEE, Paris Est marne la vallée, France*

Remerciements

Il est toujours délicat de remercier l'ensemble des personnes qui ont contribué à l'aboutissement de mes travaux de recherche. Que ceux qui ne sont pas mentionnés ne m'en tiennent pas rigueur.

Je tiens tout d'abord à remercier mon directeur de thèse Monsieur **Mohamed KHALGUI**, Maître de Conférences/HDR à INSAT Tunisie et ma Co-Encadrante Madame **Samia BOUZEFRANE**, Maître de Conférences/HDR au CNAM France, qui ont su canaliser mes travaux de recherches doctorales, l'aide et les conseils qu'ils n'ont jamais cessés de me donner m'ont été d'une valeur inestimable, ce dont je leur suis très reconnaissant.

Mes très sincères remerciements à ma Co-encadrante Madame **Olfa MOSBAHI**, pour toute l'attention qu'elle m'a portée, sa disponibilité et son soutien inconditionnel au cours de ces années de recherches. Merci infiniment Madame OLFA.

Je remercie M. **Mohamed ABID**, professeur à ENIS-Sfax Tunisie et M. **Alain PLANTEC**, professeur à l'Université de Brest pour avoir accepté d'être les rapporteurs de ce mémoire. Les remarques pertinentes qu'ils ont émis ont permis de le consolider.

Je remercie également Mme. **Maryline CHETTO**, Professeur à l'Université de Nantes, M. **Laurent GEORGE**, Professeur à ESIEE, Paris Est marne la vallée, M. **Nicolas TRÊVES**, Professeur au CNAM-Paris pour l'intérêt qu'ils ont porté à mes travaux et pour avoir accepté de faire partie du jury de ma soutenance de thèse.

Je remercie également tous les membres du laboratoire LISI-INSAT pour leur accueil et leur disponibilité, surtout le directeur Monsieur **Moncef GASMI**. Mes collègues et chers amis surtout Aymen GAMMOUDI, Med Oussama BEN SALEM... Ils n'ont jamais cessé de m'encourager et de m'apporter du support.

Je remercie également tous les membres du Laboratoire CEDRIC-CNAM qui m'ont accueilli et qui m'ont permis de m'épanouir dans mes recherches et pour leur sympathie et leur bonne humeur. Ils ont contribué à une atmosphère propice pour travailler et se sentir bien.

Je n'oublie pas non plus tous mes amis.

Enfin, merci du fond du cœur à toute ma famille ainsi qu'à tous mes amis pour leur soutien inestimable et les bons moments partagés.

Glossaire

— I : Interface

— M_p : Ensemble des méthodes

— $P_\Sigma \subseteq I$: Interfaces d'entrées

— $R_\Sigma \subseteq I$: Interfaces de sorties

— β_Σ : les méthodes qui interagissent avec les interfaces du composant

— CS : Un composant selon

— Θ : l'ensemble des signatures du composant

— AX : l'ensemble des axiomes du composant

— B : l'ensemble des comportements atomiques du composant

— $(GL, \oplus)$: Un monoïde commutatif

— COM : Component Object Model

— $DCOM$:Distributed Component Object Model

— JB : Composant JavaBeans

— ADL : Langage de Description d'Architecture

— $ACME$: Architecture Description Interchange Language

— $EADLs$: Executable Architecture Definition Language

— $POSET$: Partial Ordered event Sets

— $AADL$: Architecture Analysis and Design Language

— $RA2DL$: Reconfigurable AADL

— SAE : Society of Automotive Engineers

— $WCET$: Worst Case Execution Time

— TR^2E : Temps-Réel Réparties Embarqués

— IMA : Integrated Modular Avionics

— TPN : Time Petri Net

— BF : Blocs Fonctionnels

— FSM : Machine à États Finis

— SCI : Systèmes de Contrôles Industriels

— IEM : Input Event Module

— $IEDB$: Input Event Data Base

— DM : Data Module

— DDB : Data Data Base

— ALM : Algorithms Module

— $ALDB$: Algorithms Data Base

— OEM : Output Event Module

— AL : Niveau Architecture

— CL : Niveau Composition

— DL : Niveau Données

— β : Module Contrôlé de RA2DL

— R : Module de Contrôle de RA2DL

— Ψ_{RA2DL} : Ensemble de tous les algorithmes de RA2DL

— ξ_{RA2DL} : Sous-ensemble des algorithmes

— ASM : Architecture State Machine

— S_{AL} : État dans ASM

— S^i_{AL} : Un état particulier dans un niveau architectural

— O : Ensemble de n états dans S_{AL}

— δ : Fonction de transition au niveau architecture

— CSM : Composition State Machine

— S_{CL} : État dans CSM

— S^j_{CL} : Un état particulier dans un niveau composition

— $\Gamma(\delta_{RA2DL})$: Ensembles des modèles d'exécutions des algorithmes

— P : Ensemble de m états dans $\mathbf{S}_{CL}$

— γ : Fonction de transition au niveau composition

— DSM : Data State Machine

— S_{DL} : État dans DSM

— S^k_{DL} : Un état particulier dans un niveau données

— Q : Ensemble de k états dans $\mathbf{S}_{DL}$

— ϑ : Fonction de transition au niveau données

— BSM : Behavior State Machine

— RA : Reconfiguration Architecture

— $CondA$: Condition de passage à ASM

— RC : Reconfiguration Composition

— $CondC$: Condition de passage à CSM

— RD : Reconfiguration Données

— $CondD$: Condition de passage à DSM

— $flex_R$: Seuil de flexibilité donné par un expert

— $Flex(RA2DL)$: Équation de la flexibilité

— R^i_{exe} : Scénario de reconfiguration i accepté

— R^i : Totalité des scénarios de reconfigurations

— $AlgFlex$: Algorithme de calcule de flexibilité

— $RA2DL - Coordinator(CR)$: Composant de coordination

— MR : Middleware de reconfiguration

— RT : Jeton de reconfiguration

— RF : Flux de reconfigurations

— DA : Adresse du composant RA2DL

— V : Variable binaire

— PF : Facteur de priorité

— IRF : Port d'entrées

— ORF : Port de sorties

— EC : Contrôleur d'exécution

— ST : Jeton de synchronisation

— S : Sémaphore

— CM : Coordination Matrix

— $Reconfiguration^a_{ia,ja,ka}$:Ensemble des scénarios de reconfigurations

— ia : Reconfiguration au niveau architecture

— ja : Reconfiguration au niveau composition

— ka : Reconfiguration au niveau données

— $RA2DL - Pool$: Conteneur des composants RA2DL

— id_pool : Identifiant de RA2DL-Pool

— UT : Table des utilisateurs

— id_user : Identifiant d'utilisateur

— $password\_user$:Mot de passe d'utilisateur

— $service\_user$: Services réservées au utilisateur

— RT : Table de reconfigurations

— id_reconf : Identifiant de reconfiguration

— id_RA2DL : Identifiant d'un composant RA2DL

— ST : Table de sécurité

— $privilege\_user$: Privilèges des utilisateurs

— AAA : Authentication, Authorization et Accounting

— S : Services offerts par les composants RA2DL

— M_{sc} : les droits de chaque paire (S et C)

— $TCTL$: Timed Compotational Tree Logic

— BMS : Body-Monitoring System

— M : impulsions envoyé par le Radar

— N : Objets détectés par le Radar

— f_i : Fréquence de l'antenne

— O_j : Un objet détecté

— r_i : Direction

— d_i : Distance

— s_i : Surface

— C : Paramètre du Radar

— H_Res : Haute résolution

— L_Res : Faible résolution

— $condition\_weather$: Condition climatique

— $wind\_speed$: Vitesse du vent

— $M1$, $M2$: Moteur 2 et Moteur 2

— EV_T1, EV_T2 : Thread 1 et Thread 2

— ci : Variable d'envoi des paquets

— Pi : Paquet numéro i

— Se : Taille du paquet à l'envoi

— Sr : Taille du paquet à la réception

— (S_f) : Fréquence d'envoi du paquet

Résumé

Cette thèse de doctorat propose une approche globale intitulée RA2DL pour le développement d'une architecture embarquée, logicielle et matérielle reconfigurable pour les systèmes de contrôle industriels à base de composants AADL. La reconfiguration est une opération souvent utile permettant la modification des composants AADL logiciels et matériels afin de les adapter dynamiquement à leur environnement d'exécution. Dans cette thèse, nous définissons des composants dynamiques et flexibles intitulés RA2DL à partir des composants statiques AADL qui peuvent encoder plusieurs comportements relatifs à plusieurs scénarios de reconfiguration d'une façon dynamique. Un scénario de reconfiguration est défini comme une opération dynamique et automatique permettant l'ajout-suppression-mise à jour des algorithmes/ ports d'un composant donné. Dans ce cadre, un composant RA2DL sera la superposition de plusieurs sous-composants de même nature mais avec différentes reconfigurations ou caractéristiques au niveau des interfaces ou des implémentations. Nous modélisons cette approche à l'aide des automates temporisés pour spécifier les comportements des nouveaux composants R2ADL d'un système donné selon trois niveaux : architecture, composition et données. A l'aide du modèle checker UPPAAL, la vérification de cette approche est nécessaire pour la validation des différents scénarios de reconfiguration et notamment en ce qui concerne la vérification des propriétés temporelles, et en particulier les propriétés d'atteignabilité et de vivacité avec la logique TCTL. Une extension sera étudiée aussi pour le cas des systèmes distribués qui doivent garantir une cohérence de reconfiguration synchrone entre des composants RA2DL distribués. Notre étude portera également sur une contribution garantissant la sécurité de reconfiguration de RA2DL qui s'articule sur le concept du Pool. Enfin, nous présentons l'implémentation ainsi que la simulation des études des cas reconfigurables avec RA2DL pour montrer

l'applicabilité de l'approche proposée.

Mots clés : Approche par composant, Système de contrôle industriel, Système temps-réel embarqué, Reconfiguration, AADL, RA2DL, Modélisation, Vérification, Sécurité, Synchronisation, Simulation.

Abstract

This thesis focuses on a global approach named RA2DL that we propose for the development of embedded architecture, reconfigurable hardware and software for industrial control systems based on AADL components. The reconfiguration is a useful operation for changing the AADL software and hardware components to adapt dynamically to the execution environment. We define a new form of dynamic and flexible components called RA2DL from static components AADL which can encode several behaviors in multiple reconfiguration scenarios dynamically. A reconfiguration scenario is defined as a dynamic and automatic operation allowing the add-remove-update algorithms / ports for a given component. In this context, a RA2DL component will be the superposition of several subcomponents of the same type but with different characteristics or reconfigurations at the interfaces or implementations levels. We model this approach by using timed automata to specify the behavior of new R2ADL components of a given system on three levels : architecture, composition and data. Using the model checker UPPAAL, the verification of this approach is necessary for the validation of different reconfiguration scenarios especially regarding verification of temporal properties, and particularly the reachability and liveness properties with logic TCTL. An extension will also be investigated in the case of distributed systems that need to ensure coherence of synchronous reconfiguration between distributed RA2DL components. Our study will also include a contribution ensuring security in RA2DL which is based on the concept of the Pool. Finally, we present the implementation and simulation of reconfigurable cases studies with RA2DL to show the applicability of the proposed approach.

Keywords : Component-based approach, Industrial control system, Real-time embedded system, Reconfiguration, AADL, RA2DL, Modeling, Verification, Security, Synchronization, Simulation.

Table des matières

Liste des tableaux

Table des figures

Première partie

Introduction Générale

Chapitre 1

Introduction Générale

1.1 Rappels des concepts fondamentaux

Dans un premier temps, nous rappelons les principaux concepts de bases qui seront utilisés dans le cadre de cette thèse et les spécificités de chaque concept. Puis, nous nous focalisons sur la problématique de la reconfiguration soumise à des ensembles des contraintes. Ensuite, nous exposons la solution proposée et nous présentons les objectifs adaptées à la résolution de cette problématique.

Les systèmes embarqués : Ce sont des systèmes électroniques et informatiques intégrant à la fois des composants logiciels et matériels, sont souvent utilisés en raison de la diversité de leurs modes d'usage : Les systèmes avioniques intègrent des sous-systèmes électroniques comme les systèmes de navigation, radars, les systèmes de génération et de distribution électrique, etc. Les systèmes de télécommunications utilisent de nombreux systèmes embarqués des commutateurs téléphoniques aux téléphones mobiles. Les réseaux informatiques utilisent également des routeurs basés sur des contrôleurs embarqués et des ponts de réseau pour acheminer les données. La miniaturisation des composants logiciels, matériels et l'accroissement de la puissance de calcul dans les domaines de l'industrie sont aujourd'hui concernés par cette révolution technologique.

Composant : Dans le cadre de notre travail, nous nous basons sur la technologie intitulé *"Composant"* pour répondre aux besoins de cette révolution. En effet, plusieurs travaux de recherche sur l'approche par composant visent le développement des nouveaux types ou modèles de composants pour développer des systèmes ou des applications inté-

grants des composants. Ces composants sont très utiles car ils s'adaptent aux différents changements des besoins tels que la consommation d'énergie, la fiabilité, la performance et l'occupation physique, etc. Ces changements sont souvent en relation avec les environnements de fonctionnement et les révolutions du système : par exemple, changer le mode d'exécution ou minimiser la performance d'un composant. Le composant devrait modifier ses comportements d'une façon automatique au moment d'exécution pour répondre aux changements des caractéristiques de son environnement de l'exécution. Nous prenons les exemples de l'occurrence d'une panne matérielle ou logicielle, un scénario de reconfiguration, une modification de l'architecture et un changement du mode d'exécution qui obligent d'adapter le comportement d'un composant.

Modélisation des composants logiciels : Les travaux de recherche sur les systèmes embarqués visent le développement de modèles, de langages et d'outils pour la conception de systèmes embarqués. Ces derniers sont construits de manière modulaire à partir de composants. Jusqu'alors, les composants avaient été envisagés comme une décomposition structurelle du système à concevoir. Le défi actuel est de prendre en compte d'autres types de spécifications comme les spécifications structurelles et fonctionnelles. Les systèmes embarqués sont à la fois soumis à des contraintes de capacités (mémoire, bande passante), et à des exigences non fonctionnelles (exécution sous contraintes de temps réel). L'objectif principal dans ce domaine est d'étudier des modèles formels permettant de décrire ces systèmes et leurs contraintes (conception, spécification), de les construire (programmation, simulation, synthèse, exécution), et de les analyser (validation, vérification).

Composant AADL : (Architecture Analysis and Design Language) Standardisé par la sociétie SAE (International Society of Automotive Engineers) en 2004 [FG12] pour l'analyse et de description d'architecture des systèmes, qui est destiné à l'industrie des systèmes embarqués. AADL décrit plusieurs composants qui modélisent une partie du système. Certains composants sont matériels (bus, processor, memory ...), d'autres logiciels (process, thread, subprogram, ...). AADL fournit une description sous forme de texte, XML et graphique pour décrire des architectures matérielles et logicielles.

Automates temporisés : Introduits par Alur et Dill dans [AD94], constituent un modèle classique dont la sémantique s'exprime en terme de système de transitions tem-

porisé. Ils sont formés d'une structure de contrôle finie et manipulent un ensemble fini de variables réelles appelées horloges pour spécifier les contraintes de temps.

Logiques temporelles : Ces logiques ont été proposées pour la spécification des systèmes réactifs, pour la première fois en 1977 par Amir Pnueli [Pnu77]. Ce sont des formalismes adaptés pour énoncer des propriétés faisant intervenir la notion d'ordonnancement dans le temps, par exemple : *un radar envoie un signal à la détection d'un objet.*

UPPAAL : Un outil permet d'analyser des systèmes formés d'une collection d'automates temporisés qui communiquent par des variables entières partagées et des synchronisations par messages. Les automates temporisés d'Uppaal sont une variante des automates temporisés d'Alur et Dill [AD94]. En plus des horloges, les automates d'Uppaal manipulent des variables entières bornées. Les transitions de ces automates sont étiquetées par des gardes, des remises à zéro, des mises à jour de certaines variables entières et des étiquettes de synchronisation $m?$ et $m!$ où m est un canal.

Reconfiguration dynamique : De nos jours, l'un des défis les plus importants est le compromis entre la performance et la réponse rapide aux changements du marché et aux besoins des clients. L'une des directions les plus prometteuses pour résoudre ces problèmes est la reconfiguration dynamique. La reconfiguration dynamique. Cette fonctionnalité se réfère au processus de modification de la structure et du comportement du système au cours de son exécution. Être reconfigurable est important afin de réagir rapidement à l'évolution des besoins soudains et imprévisibles avec un coût et un risque minimum.

Les exigences des systèmes de contrôle industriels ont progressé de plus en plus en termes de flexibilité et d'agilité. En fait, les systèmes statiques sont conçus pour garder la même structure et le même comportement tout au long de leur existence. Sinon, ces systèmes n'ont pas la capacité de répondre à un événement externe. Ces systèmes ne peuvent pas faire face aux conditions fluctuantes qui touchent leur environnement d'exécution comme des pannes de machines, les changements d'objectifs, l'intégration d'une nouvelle machine, la modification de la demande de l'utilisateur, etc. L'une des solutions pour répondre à ces exigences est de rendre ces systèmes capables de s'adapter dynamiquement à l'environnement et à utiliser cette capacité d'adaptation afin de réduire leurs coûts et améliorer leur performance sans perturbations. En fait, selon l'état de leurs environ-

nements, les systèmes de contrôle industriels peuvent chercher rapidement et de manière rentable la nouvelle configuration et la mise en œuvre de la reconfiguration sans être mis hors ligne. Il s'avère donc nécessaire de recourir à une méthode pertinente et rigoureuse pour modéliser ces systèmes afin de tenir compte de leur caractère reconfigurable, de leur conception et à être en mesure de maintenir leur cohérence au cours de leur évolution. La reconfiguration est le changement du comportement du système pour l'adapter à son environnement. Il existe deux types de reconfiguration : matérielle qui correspond à l'activation et la désactivation d'un composant du système et logicielle qui correspond à l'ajout ou la suppression des tâches.

Dans ce contexte, nous nous intéressons à améliorer les performances et réduire les coûts au niveau des systèmes embarqués à base de composants à travers leur adaptation au changement de leurs environnements. Ce changement est bien la reconfiguration que nous visons à l'implémenter au niveau d'un composant AADL qui se fait selon trois niveaux : architecturale, composition et données. Un système reconfigurable est modélisé à un moment donné par un ensemble de tâches. La reconfiguration architecturale engendre un ensemble de modifications sur l'architecture du système par l'ajout, la suppression des tâches ou l'arrêt et la reprise de l'exécution d'autres. La reconfiguration de composition permet la modification de la composition des tâches pour une architecture donnée. La reconfiguration des données modifie les valeurs des variables pour une composition donnée.

Dans ce chapitre introductif, nous présentons le contexte général de notre étude, nous synthétisons les problématiques posées dans cette thèse ainsi que les contraintes. Nous détaillerons les solutions et les objectifs de notre approche que nous exposerons dans la suite de ce mémoire.

1.2 Problématique

De nos jours, un des problèmes majeurs pour adapter dynamiquement des systèmes de contrôle industriel pour les rendre reconfigurables est la reconfiguration de son architecture applicative construite par des composants logiciels interagissant avec une architecture matérielle qui représente le support d'exécution construit par des composants matériels

communicants entre eux. Nous regroupons ici les problèmes que posent la reconfiguration du système de contrôle industriel selon quatre axes complémentaires, à savoir : la détection des besoins de reconfiguration d'un système de contrôle industriel à base des composants, la mise en oeuvre et l'application des scénarios de reconfigurations, la vérification et l'analyse formelle du comportement d'un système reconfigurable après chaque reconfiguration, la coordination et la synchronisation entre les composants reconfigurables au niveau d'une architecture distribuée, et enfin le problème de sécurité des composants reconfigurables.

1.2.1 Détection des besoins de reconfiguration

Un premier problème concernant la reconfiguration dynamique des systèmes de contrôle industriel à base des composants est représenté au niveau de la reconfiguration de ces composants localement, qui consiste à détecter d'une part les conditions imprévisibles du changement de l'environnement à l'exécution de ces composants qui nécessitent des reconfigurations du composant, et d'autre part de s'assurer que le composant concerné est dans un état qui permet d'appliquer la reconfiguration demandée de façon sûre et automatique sans perturber l'exécution du système.

Par exemple : suite à une panne d'un capteur d'un système radar, il faut détecter les besoins de reconfiguration comme le changement climatique ou l'augmentation de vitesse du vent afin de proposer des scénarios de reconfigurations qui ne violent pas le fonctionnement du capteur. Puis la mise en oeuvre des mécanismes de reconfiguration.

1.2.2 Vérification formelle du comportement

La reconfiguration est dédiée essentiellement à la modification du comportement interne qui complique considérablement l'exécution du système ou du composant et qui permet la perturbation de son fonctionnement qui touche essentiellement son architecture, sa structure et ses données. En effet, il est nécessaire de suspendre certains algorithmes du système pour leur permettre de se reconfigurer. Si ces algorithmes sont modifiés par la reconfiguration, on ne peut les suspendre qu'une fois qu'ils ont fourni leurs résultats ; sans quoi il est probable que le comportement du système devient incohérent. Nous appelons architecture flexible, l'architecture qui supporte les actions de la reconfiguration.

Cette dernière permet de réaliser la reconfiguration de façon sûre et automatique. Nous devrons également nous intéresser aux problèmes que pose la vérification du comportement après chaque reconfiguration. Nous devrons également proposer une méthode d'analyse et vérifié le comportement du système reconfigurable, afin de vérifier que les scénarios de reconfigurations vérifient un certain nombre de propriétés.

1.2.3 Communication et synchronisation des composants reconfigurables

Le troisième problème réside au niveau de la coordination et la communication entre les composants reconfigurables. Au-delà de la mise en œuvre pratique des reconfigurations fiables des composants, et comment garantir la cohérence des reconfigurations dans une architecture distribuée à base des composants reconfigurable automatiquement, nous devons également proposer un protocole de coordination dans le cas où les reconfigurations sont synchrones afin de reconfigurer globalement le système. Une matrice de coordination est proposée qui permet la gestion de la coordination et la synchronisation d'une façon sûre afin de vérifier certaines propriétés de sûreté de cohérence, coordination correcte et ne viole pas l'aspect fonctionnel du système.

1.2.4 Sécurité des composants reconfigurables

Un système reconfigurable est ouvert à toutes formes d'attaques extérieures et des utilisations malveillantes dans le contexte d'un réseau ouvert se traduisant par une modification ou dysfonctionnement illégale. Par exemple, un scénario de reconfiguration peut être utilisé pour attaquer le système ou un composant. Au-delà de la mise en oeuvre pratique d'une sécurisation fiable du système pour garantir un niveau très élevé de sécurité, nous devrons également proposer une méthode pour sécuriser le système ou ses composants.

Ces différents problèmes seront fortement raffinés lorsque nous présenterons l'étude détaillée des besoins auquel nous allons répondre (Partie 2).

1.3 Contraintes de Reconfiguration

La reconfiguration matérielle et logicielle du système [LCD+00] peut être contrainte par des propriétés architecturales, de flexibilité, de ressources et de sécurité, etc. Elle est également contrainte d'assurer la performance requise par le système reconfigurable. Parmi les contraintes de reconfiguration, nous pouvons citer les suivantes :

- La reconfiguration du système repose elle-même sur l'ensemble des reconfigurations de ses composants et les possibilités de coordination et de synchronisation fournies. Cette reconfiguration doit, en particulier, garantir le respect des contraintes fonctionnelles, temporelles et opérationnelles. La question qui se pose est alors, comment peut-on effectuer la reconfiguration d'un système à base des composants par l'adaptation de ces comportements au moment de l'exécution, et plus particulièrement, comment reconfigurer un de ses composants (matériels ou logiciels) en respectant toutes les contraintes du cahier des charges ?

- La reconfiguration des composants est dynamique et se fait d'une manière automatique pendant l'exécution du système. Les composants concernés par les reconfigurations peuvent être implicitement construits lors de fonctionnement du système (On-line) [CE00], ou des composants déjà spécifiés dès la conception et avant la mise en marche du système (Off-line) [The04]. La question qui se pose est donc, comment trouver un bon compromis entre les composants on-line et off-line ?

- La reconfiguration des composants rend l'exécution du système un peu délicate vue la synchronisation et la cooridnation entre ces composants reconfigurables [MLCR08]. En effet, plusieurs incompatibilités peuvent apparaitre entre les scénarios de reconfigurations et les règles de contrôle d'exécution. A titre d'exemple, nous pouvons citer la possibilité de deux scénarios de reconfigurations contradictoires qui apparaissent au même temps, un pour changer les constantes ou les variables d'un tel algorithme d'un composant et l'autre pour supprimer le même algorithme.

- Un autre problème concernant la reconfiguration dynamique des systèmes à base de composants consiste à sécuriser des activités de reconfiguration [BDF+11]. Cela concerne d'une part les composants concernés par les scénarios de reconfiguration, et d'autre part

le système qui dois reste fonctionnel et dans un état qui permet de mettre en œuvre l'adaptation de ses comportements.

1.4 Objectif et approche

1.4.1 Objectif

L'objectif général de notre travail de thèse est d'améliorer la productivité des systèmes de contrôles industriels qui sont équipés de composants matériels et logiciels en vue de les rendre reconfigurables et adaptatifs aux changements de leur environnent d'exécutions, en termes de qualité et de productivité. Nous devrons donc automatiser la reconfiguration et la production de ses composants composites. Pour améliorer la qualité de la production, nous nous appuierons sur des techniques de reconfiguration dynamique et automatique des composants de type *AADL* du système, ainsi que sur l'utilisation de protocoles propres à la mise en œuvre de coordination entre les composants du système dans une architecture distribuée. Pour améliorer la sécurité du système, nous utiliserons des techniques de sécurisation automatique des composants reconfigurables. Pour vérifier la qualité de notre approche, nous utiliserons des techniques de vérification formelle.

1.4.2 Approche Proposée

La solution envisagée par cette thèse s'attache à décrire une méthodologie globale représentée dans la Figure 1.1 qui repose sur six étapes : la spécification, la vérification, la conception, la sécurité, la flexibilité et l'implémentation, nous énumérons les détails de chaque étape ci-après pour le développement d'un système de contrôle industriel reconfigurable. Nous nous plaçons plus particulièrement dans le cadre des systèmes embarqués utilisant les approches par composants. Nous nous appuyons pour cela sur les travaux menés autour des composants AADL afin de proposer un composant AADL reconfigurable intitulé **RA2DL**.

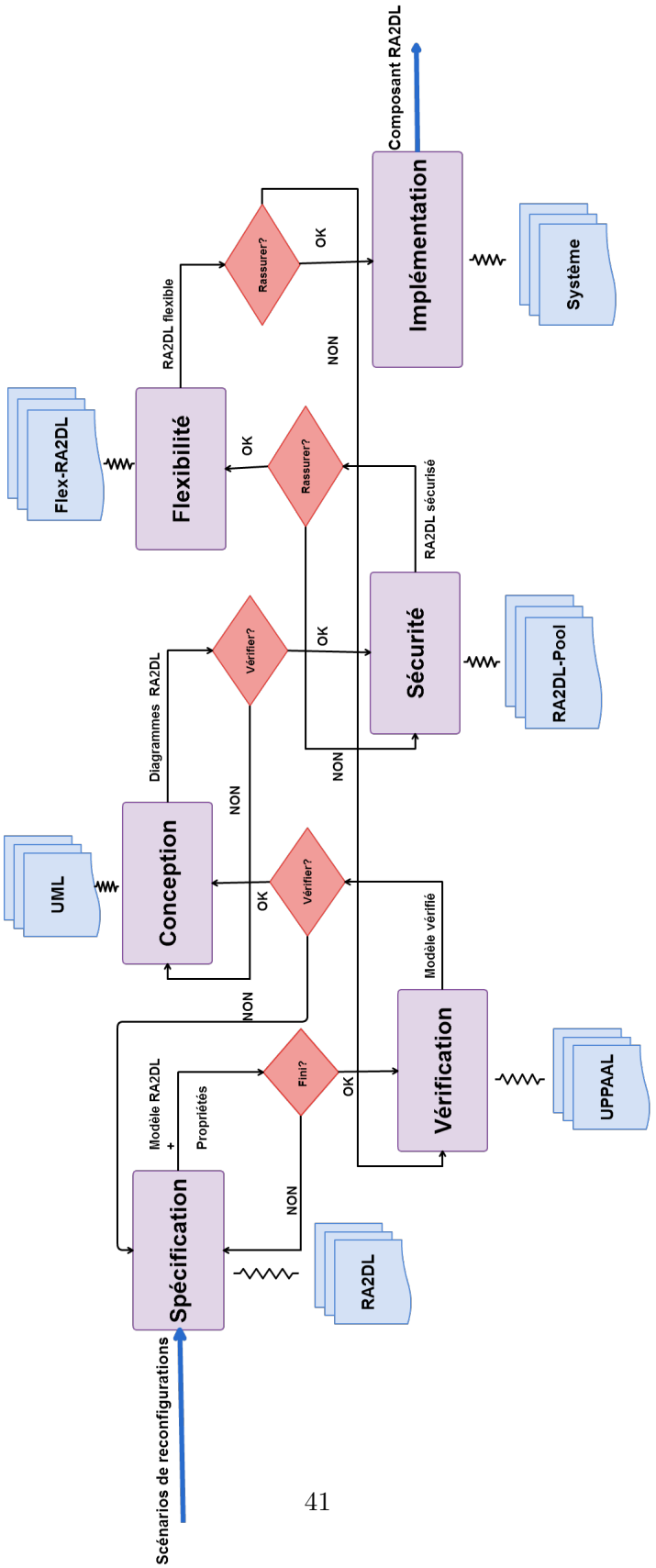

FIGURE 1.1 – Méthodologie RA2DL.

1.4.2.1 Spécification du composant RA2DL

La première étape consiste à spécifier les caractéristiques locales du nouveau composant RA2DL. D'abord, nous regroupons les formes de reconfiguration en trois modes : architecturales, compositions (structures) et données. Ensuite, nous proposons une nouvelle architecture qui s'adapte aux trois formes de reconfigurations, c'est une architecture flexible répartie en quatre modules fondamentaux. Le premier module est dédié aux événements d'entrées, le deuxième est consacré aux algorithmes internes du composant, le troisième est réservé pour les événements de sorties et le quatrième dédie pour les données du composant. Enfin, nous formalisons les comportements de RA2DL en tenant compte ces trois formes de reconfigurations [AMKB15b] [AMKB15c].

1.4.2.2 Vérification du composant RA2DL

La deuxième étape concerne la vérification des propriétés de sûreté, de vivacité, d'atteignabilité, de sécurité, etc. de RA2DL, en utilisant le modèle checker UPPAAL [Mer01] basé sur les automates temporisés pour la modélisation des composants et la logique TCTL (Timed Compotational Tree Logic) [ACD90] pour vérifier les propriétés de comportement, de synchronisation et de la sécurité de RA2DL. [AMKB15a].

1.4.2.3 Conception du composant RA2DL

La troisième étape consiste à représenter l'ensemble des besoins fonctionnels et non fonctionnels de RA2DL au niveau de la conception. Il s'agit de créer une représentation simplifiée par un modèle et un méta-modèle du composant RA2DL en utilisant le langage de modélisation unifié (UML) afin de montrer les vues statiques et dynamiques de RA2DL par un ensemble de diagrammes.

1.4.2.4 Sécurité du composant RA2DL

La quatrième partie est consacrée à la sécurisation du composant RA2DL. Nous proposons dans ce contexte une méthode qui s'articule autour de la notion du *Pool*, qui est un conteneur représenté par une super-classe qui vise à regrouper des composants RA2DL

selon un facteur de similarité afin d'affecter les différentes politiques ou des mécanismes de sécurité selon le niveau de sensibilité de ces composants [AMKB16].

1.4.2.5 Flexibilité du composant RA2DL

La cinquième étape est dédiée aux mesures de la flexibilité de RA2DL, qui est un facteur crucial qui permet de calculer les taux d'acceptation de scénarios de reconfiguration. Ce facteur est comparé à un seuil $flex_R$ donné par un expert. Pour mesurer la flexibilité d'un tel composant RA2DL, nous proposons une approche intitulée $Flex-RA2DL$ qui s'appuie sur un algorithme $AlgFlex$ dans le but de calculer cette flexibilité.

1.4.2.6 Implémentation du composant RA2DL

La sixième étape est réservée à l'implémentation du composant RA2DL, par la mise en œuvre de l'ensemble de ses modules sur une carte Arduino et la réalisation d'un outil complet intitulé $RA2DL-Tool$ qui réalise toutes ces fonctions de la spécification à l'implémentation finale. L'application de notre approche est testée sur trois études de cas concrètes afin d'en valider les performances et montré l'applicabilité de notre approche dans le domaine de l'industrie.

1.5 Organisation du Manuscrit

Le manuscrit de thèse est organisé en sept chapitres structurés comme suit :

- Chapitre 2 : dresse un état de l'art sur l'approche par composant (l'approche online et off-line) et l'ingénierie dirigée par les modèles IDM. Après la définition du système de contrôle industriel, nous décrivons une étude détaillée du langage AADL et comment un composant de ce langage peut être exploité pour appliquer la reconfiguration locale au niveau du composant. Nous exposons les méthodes de reconfigurations statiques et dynamiques existantes nous détalions dans ce cadre les méthodes de la reconfiguration automatique des systèmes temps-Réels embarqués. Nous discutons aussi les limites des solutions existantes afin de synthétiser toutes ces approches.

- Chapitre 3 : comporte la première contribution de notre thèse représentée par une

approche qui analyse la reconfiguration d'un composant AADL et la transformation en un composant RA2DL. Cette approche vise à étudier les formes de reconfigurations possibles s'appliquant au RA2DL afin de proposer une architecture flexible. Nous proposons la formalisation du composant RA2DL avec ces comportement, la modélisation de RA2DL avec des automates temporisés et le calcul du facteur de flexibilité de RA2DL. Avant de conclure nous illustrons un exemple simple pour montre le passage d'un composant AADL à un composant RA2DL reconfigurable.

- Chapitre 4 : présente des extensions possibles à un composant RA2DL pour faciliter son exécution dans une architecture distribuée pour garantir une cohérence avec des reconfigurations synchrones. Nous étudions également c'est quoi un réseau des composants RA2DL au niveau d'une application distribuée avec la proposition et la modélisation d'un modèle d'exécution et d'un réseau de RA2DL distribué. Un simple exemple sera mis en place pour comprendre les contributions de ce chapitre.

- Chapitre 5 : propose une approche de sécurisation du composant RA2DL. Cette approche tient compte du groupement des composants RA2DL dans un conteneur intitulé $RA2DL - Pool$ en fournissant deux mécanismes de sécurité pour chaque $Pool$: authentification et contrôle d'accès.

- Chapitre 6 : consacré à l'implémentation du composant RA2DL. Le composant s'appuie sur un outil $RA2DL - Tool$ pour simuler l'exécution et les fonctionnalités de reconfigurations de la spécification à l'implémentation de RA2DL. Pour montrer les variétés d'applications, nous présentons trois études de cas sur lesquelles nous appliquons les contributions inventées au niveau de cette thèse. Nous montrons en particulier l'utilité et les gains de l'approche de reconfiguration proposée d'AADL par des évaluations statistiques.

- Chapitre 7 : nous concluons la thèse par un récapitulatif des contributions et des discussions avec des perspectives et des extensions possibles pour montrer l'utilité de notre approche dans nos futurs travaux de recherche.

Deuxième partie

État de l'art

Chapitre 2

État de l'Art

2.1 Introduction

Dans ce chapitre, nous présentons un état de l'art concernant l'ingénierie dirigée par les modèles et sur les approches par composants. Dans un premier temps nous définissons qu'est-ce-qu'un système de contrôle industriel et c'est quoi la différence entre un système de contrôle industriel et les systèmes temps-réels embarqués ? Par la suite, nous citons des travaux d'ordre général sur les composants et nous abordons les types des composants $On - line$ et $Off - line$ ainsi que la sécurité de l'approche par composants. Ensuite, nous présentons un état de l'art sur le langage de description et d'analyse d'architecture (AADL) avec lequel nous avons restreint notre champ d'étude, nous donnons un panorama des principes d'AADL et les catégories de ses composants. Enfin, avant de conclure, nous présentons l'état de l'art dans le domaine de reconfiguration et plus particulièrement dans les systèmes temps-réels embarqués. Tout naturellement nous commençons par décrire les méthodes de reconfigurations statiques, par la suite, nous donnons les différentes méthodes de reconfigurations dynamiques, nous exposons aussi les méthodes de reconfigurations automatiques des systèmes temps-réels comme la reconfiguration mono-processeur et multi-processeurs. Notre contribution se situe dans la lignée de la reconfiguration automatique et dynamique des approches par composants. L'objectif est de construire un composant reconfigurable pour une application à base des composants distribuée, et de l'enrichir jusqu'à obtenir une application totalement reconfigurable.

Les principales contributions de cette thèse portent sur la reconfiguration des compo-

sants AADL matériels ou logiciels. Ces composants sont connectés entre eux pour former une architecture distribuée. Une description AADL consiste en un ensemble de déclarations de composants qui peuvent être instanciées pour former la modélisation d'une architecture du système global. AADL est particulièrement intéressant car il est orienté vers le développement d'applications par des composants et illustre donc bien la réponse à un besoin exprimé pour la reconfiguration par composant ou un réseau de composants soumis à des contraintes de fiabilité forte. En revanche, AADL définit les moyens de liaison entre les spécifications de ces composants et l'expression du comportement des différents composants.

2.2 Système de Contrôle Industriel

Dans cette section, nous présentons quelques définitions pour faire la différence entre un système embarqué, un système temps-réel et un système de contrôle industriel.

Système embarqué. Selon la définition de Wayne Wolf dans [Wol08] « *Un système embarqué est défini comme un système électronique et informatique autonome, souvent temps réel, spécialisé dans une tâche bien précise.* » Le terme désigne aussi bien (i) le matériel informatique représenté par certains périphériques qui sont essentiellement un processeur dédie au calcul et le contrôle de l'ensemble du système, des capteurs et des actionneurs et (ii) le logiciel utilisé constitue l'ensemble des programmes décrits dans un langage spécifique. Les ressources de ce système sont limitées. Cette limitation est également de niveau spatial à cause de l'encombrement réduit et de niveau énergétique à cause de la consommation restreinte.

Système temps-réel. Un système temps-réel (STR) est tout système qui doit respecter des contraintes essentiellement temporelles, l'exactitude des applications ne dépend pas seulement des résultats d'exécutions mais elle est liée aussi au facteur temps durant lequel ce résultat est produit. Dans ce cas, si les contraintes temporelles de l'application ne sont pas respectées, alors le systéme peut sebir des défaillances. Les contraintes temporelles sont divisées en deux types [Ngo08] : (i) *contraintes strictes :* à chaque occurrence d'une tache, on associe une échéance que le système doit impérativement respecter. Sans respect de ces

contraintes strictes, le système peut avoir des conséquences catastrophiques. (ii) *contraintes relatives :* le système se compose de tâches ayant des échéances, comme dans le cas des systèmes à contraintes strictes. Cependant, une violation d'une contrainte temporelle est tolérable et a un impact sur la qualité de service que l'on cherche à minimiser.

Système de contrôle industriel. Le système de contrôle industriel (SCI) fortement évolué ces dernières années est un système qui réunit un ensemble d'activités de types techniques et technologiques destinées essentiellement à l'automatisation des procédés et de systèmes de production industrielle. Les systèmes de contrôle industriel sont des réseaux de commande et de contrôle conçus pour soutenir les processus industriels. Ces systèmes sont utilisés pour surveiller et contrôler divers processus et opérations, tels que la distribution de gaz et d'électricité, l'eau, le raffinage du pétrole et le transport ferroviaire.

2.3 Ingénierie Dirigée par les Modèles et la Validation

Nous exposons dans cette section l'intérêt de l'ingénierie dirigée par les modèles IDM pour tirer profit de certains aspects : IDM en faveur de la qualité et de l'adéquation des ateliers produits par rapport au besoin, mais aussi pour couvrir le plus largement possible les besoins en termes de vérification et de validation.

2.3.1 Ingénierie Dirigée par les Modèles

L'ingénierie dirigée par les modèles IDM (en anglais, MDE : Model Driven Engineering) est un concept qui désigne la généralisation de l'approche MDA (Model Driven Architecture) [Sch06] et la séparation ente les spécifications fonctionnelles d'un système structuré en modèles et les spécifications de son implémentation sur une plate-forme donnée. Les avantages de l'IDM sont très nombreux dans la communauté des systèmes informatiques tels que :

— Une meilleure maîtrise de la complexité croissante des systèmes informatiques et leur réutilisation,

— La réalisation du même modèle sur plusieurs plates-formes grâce à des projections standardisées,

— L'indépendance vis à vis des évolutions technologiques,

— L'interopérabilité des applications tout en permettant l'évolution des plates-formes et techniques,

— La mise en œuvre de l'IDM est entièrement basée sur les modèles et leur transformation.

L'approche IDM consiste à manipuler différents modèles de l'application, depuis une description abstraite jusqu'à une représentation correspondant à l'implémentation effective du système. Le processus IDM se décompose en trois étapes, représenté par la Figure 2.1 :

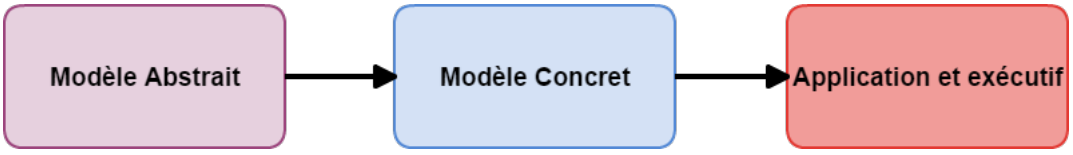

FIGURE 2.1 – Processus de l'approche IDM.

- Modèle Abstrait : ce modèle est raffiné afin d'ajouter différentes informations non fonctionnelles demeurant indépendantes de la plateforme d'exécution qui sera effectivement utilisée telle que la gestion de la sécurité,

- Modèle Concret : ce modèle prend en compte les spécifications propres à la plateforme d'exécution, il peut être également raffiné afin de prendre en compte différents paramètres liés à l'environnement d'exécution,

- Application et exécutif : le modèle concret est ensuite utilisé pour générer une application exécutable basée sur les spécifications à partir desquelles le modèle abstrait est construit.

L'IDM propose une démarche intégrée permettant de rassembler tous les éléments pour la description d'une application. De cette façon, IDM vise la mise en place de différentes étapes de raffinement depuis la conception de l'application jusqu'à la production de code exécutable.

Dans le contexte de l'IDM, les modèles sont manipulés via l'utilisation de *langages* [Pla12]. C'est pourquoi un certain nombre de langages permettant cette description sont apparus et offrent à ce jour un certain nombre de fonctionnalités. La plupart des approches

guidées par des modèles se basent soit sur :

— **UML** (Unified Modeling Language) [MG00] c'est un langage de modélisation unifié à usage général avec des profils comme MARTE (Modeling and Analysis of Real-Time and Embedded) [Mal08],

— **ADL** (Architecture Description Language)[ZFZ10] c'est un langage de description d'architecture dédié à des domaines particuliers, il permet la modélisation d'une architecture conceptuelle d'un système logiciel et/ou matériel et fournit une syntaxe concrète et une structure générique *(framework)* conceptuelle pour caractériser les architectures. Parmi les ADLs, le langage d'analyse et de description d'architectures (Architecture Analysis and Design Language) AADL fournit une notation textuelle et graphique standardisée pour décrire des architectures matérielles et logicielles. Le succès d'AADL dans l'industrie est justifié par son support avancé à la fois pour la modélisation d'architectures reconfigurables et pour la conduite d'analyses. En particulier, le langage a été conçu pour être extensible afin de permettre des analyses qui ne sont pas réalisables avec le langage de base. Dans cette optique, une annexe au standard AADL a été définie ABS (Annex Behavior Specification) [BF$^+$07] pour compléter les descriptions d'architecture avec du comportement,

— **BIP** (Behavior Interaction Priority)[BBS06] : c'est un langage spécifique pour répondre à des besoins particuliers de modélisation et d'analyse, par exemple Fractal [BCL$^+$06] et Ptolemy [EJL$^+$03].

2.3.2 Vérification et Validation

L'enjeu technologique et scientifique est l'étude de méthodes et outils facilitant le développement rigoureux et à coût maîtrisé des systèmes embarqués. La vérification représente un souci majeur pour un grand nombre de systèmes embarqués. D'où l'importance de la validation de ces systèmes [DPDP11], c'est-à-dire test, vérification et certification. Il y a donc un besoin réel et pressant de développer des méthodes et outils efficaces pour la validation des systèmes embarqués. Car la moindre faille sera exploitée par des malveillants potentiels . Les définitions les plus adéquates selon [TDH$^+$04], [Car02] et [Mac05] pour les termes de vérification et de validation sont :

- vérification : est un ensemble d'étapes qui permet d'assurer que les modèles sont correctement formés et développés selon les bonnes pratiques. Les pratiques à ce niveau peuvent être l'aboutissement d'un ensemble de techniques comme la technique de modélisation.

– **validation :** est un processus qui permet d'affirmer que le modèle est cohérent par rapport à son intention en termes de méthodes utilisées et de résultat obtenu ; le but ultime est d'obtenir un modèle efficace, c'est-à-dire un modèle qui adresse le bon problème, qui fournit des informations pertinentes sur le système modélisé et que le modèle est finalement réellement utilisé.

2.3.3 Exemples de reconfigurations et IDM

Dans cette sous-section, nous présentons des exemples les plus proches de notre contexte de travail et qui traitent principalement le développement des systèmes adaptables dans le cadre de l'ingénierie dirigée par les modèles.

2.3.3.1 MADAM

MADAM (mobility and adaptation-enabling middleware) [FHS+06] est projet qui permet de fournir aux ingénieurs logiciels des moyens adéquats pour développer des applications mobiles adaptatives.

Avec MADAM, le modèle du système est une composition des composants types. Chaque composant type peut être atomique ou composite. Pour chaque composant type, plusieurs implémentations peuvent être définies. Pour distinguer entre les alternatives d'implémentations d'un composant dans les modèles d'architecture de MADAM, ils annotent des composants avec des propriétés (*properties*). Les propriétés sont étroitement liées à des éléments de contexte. De cette façon, ils représentent les dépendances entre les implémentations de composants et leurs contextes. Ces dépendances sont représentées sur des propriétés indicatrices de fonctions (*property predictor functions*), ces propriétés attribuent des valeurs constantes aux propriétés du composant type afin de faire la liaison entre les implémentations du composant et les éléments du contexte d'adaptation, cf. figure 3.1. Le middleware automatise la dérivation d'une variante d'un contexte spécifique à l'exécution,

c.-à-d. les configurations sont calculées en cours d'exécution.

MADAM met l'accent sur tous les éléments de l'adaptabilité des systèmes, notamment : le contexte, la variabilité, le comportement adaptatif du système par rapport à son environnement. Cependant, la modélisation des opérations de reconfiguration, les caractéristiques temporelles du système et l'analyse temporelle du système pour son comportement adaptatif ne sont pas traitées dans le projet MADAM.

2.3.3.2 DiVA

DiVA (Dynamic Variability in complex Adaptive systems) [FDB+08] traite la notion de l'adaptabilité des systèmes dans une approche IDM en focalisant sur un traitement efficace du nombre de configurations possibles, qui peut croître de façon exponentielle avec chaque nouvelle dimension de variabilité. Dans cette approche on modélise : le contexte, la variabilité, le comportement adaptatif du système par rapport à son environnement. Cependant, la modélisation : des opérations de reconfiguration, les caractéristiques temporelles du système et l'analyse temporelles du système pour son comportement adaptatif, restent des points ouverts non traités dans le projet DiVA.

2.3.3.3 CEA-Frame

CEA-Frame (Construction and Execution of Adaptable applications) [LSO+07] une approche dédiée pour la construction et l'exécution des applications reconfigurables qui offre principalement :

1- Des méthodes de spécification des variantes d'une application en combinant les techniques d'IDM et la modélisation orientée aspect.

2- Un mappage pour générer les éléments du système liés à la plateforme à partir de spécifications indépendant de la plateforme.

CEA-Frame présente une partie pour l'instanciation et l'exécution de ses applications. En conclusion, cette approche modélise le contexte, la variabilité du système et son comportement adaptatif, mais ne modélise pas les opérations de reconfiguration et ne prends pas en compte l'influence de ces opérations sur les caractéristiques temporelles du système.

Dans cette section nous avons présenté des travaux autour du développement des systèmes adaptables dans le cadre de l'IDM. Ces travaux ne traitent pas le cas des systèmes de contrôles industriels dans le cadre de l'IDM.

l'IDM de nos jours, ne cesse de faire face à des applications de plus en plus complexes qui doivent évoluer rapidement, à moindre frais ainsi que dans des courts délais. Et ce en répondant aux exigences croissantes des utilisateurs et au nombre grandissant de fonctionnalités à intégrer jusqu'à l'obtention du produit final. Pour aider les développeurs à suivre cette évolution, de nouvelles techniques de construction de logiciels destinées à faire face à ces contraintes ont vu le jour. C'est le cas de l'approche par composants, qui vise à faciliter le développement d'applications à partir de l'assemblage de briques logicielles/ matérielles prédéfinies appelées composants. Nous expliquons dans la section suivante les intérêts des approches par composants.

2.4 Approches par Composants

Il est incontestable que le développement des applications à base de composants est devenue la norme depuis une vingtaine d'années. Nous présentons dans cette section les concepts de base des composants, puis nous dressons un état de l'art du domaine, nous décrivons les différents modèles à composant de type On-line (En ligne) et Off-line (Hors ligne). Ensuite, nous exposons les méthodes existants de sécurité des approches par composants. Enfin, nous concluons la section par une synthèse discute les limites de ces approches dans cette thèse pour mieux passionner notre approche.

2.4.1 Concept du Composant

De nos jours, il existe beaucoup de définitions dans la littérature du terme « *composant* ». Il est presque impossible de donner toutes ces définitions, mais nous nous essayons dans cette section de donner les définitions les plus utilisées par la communauté des systèmes embarqués.

Selon la définition de Larousse [1] un composant est un « *Élément standard utilisé dans*

1. http ://www.larousse.fr/dictionnaires/francais/composant/17736

la construction de produits industriels de série tels que machines, véhicules, circuits électriques et électroniques, appareils électroménagers, portes, fenêtres, etc. (En électronique, on distingue les composants passifs et les composants actifs) ».

Dans le rapport de Microsoft [Mic95], un composant est défini comme un fragment d'un logiciel qui est réutilisable, sous une forme binaire qui peut être lié avec d'autres composants provenant d'autres fournisseurs.

Szyperski dans [Szy02] définit un composant comme une unité de composition qui possède des interfaces contractualisées et un contexte de dépendance exprimé explicitement. Ensuite, il ajoute qu'un composant peut être déployé indépendamment.

Ivica Crnkovic dans son livre [Crn02] spécifie que chaque composant possède un ensemble d'interfaces permettant de décrire les connections possibles avec d'autres composants d'un système, ainsi qu'un code exécutable pouvant être associé au code d'autres composants.

Aoyoma dans [Aoy98], précise la différence entre le concept d'un *composant* et le concept d'un *objet*. Il considère qu'un composant devrait être capable de se connecter et de s'exécuter avec d'autres composants et/ou des frameworks, ce dernier peut être composé au moment de l'exécution sans compilation. De plus, il doit séparer l'interface de l'implémentation et cacher les détails de l'implémentation afin que les composants puissent être composés sans connaître les détails de l'implémentation.

Les caractéristiques principales servant à décrire un composant sont soulignées dans les définitions citées ci-dessous :

- *Réutilisable :* représente le principal objectif d'utilisation d'un composant. Il s'agit d'utiliser de nouveau le composant pour diverses applications,

- *Interaction* : désigne l'interaction et la communication entre les composants par l'intermédiaire de leurs interfaces.

Ivica Crnkovic dans [Crn04], définit deux types de composants pour les systèmes embarqués. Le premier réservé pour les petits systèmes non fonctionnels où l'interface du composant récapitule ses propriétés qui sont visibles de l'extérieur. Le deuxième type est dédié aux larges systèmes où les contraintes de ressources ne sont pas les principales pré-

occupations et la complexité et l'interopérabilité jouent un rôle beaucoup plus important.

Dans notre thèse, nous définissons le modèle d'un composant (Figure 2.2) comme une boîte noire qui exécute un ensemble de fonctionnalités et qui est caractérisée par :

- **Une implémentation :** est présentée sous la forme des algorithmes intégrants des programmes exécutables qui supportent ses fonctionnalités et fournissent des services.

- **Entrées/sorties :** représentent les interfaces d'un composant, assurant les interactions avec d'autres composants de son environnement, ces interfaces possèdent deux types de ports, un pour les événements et l'autre pour les données.

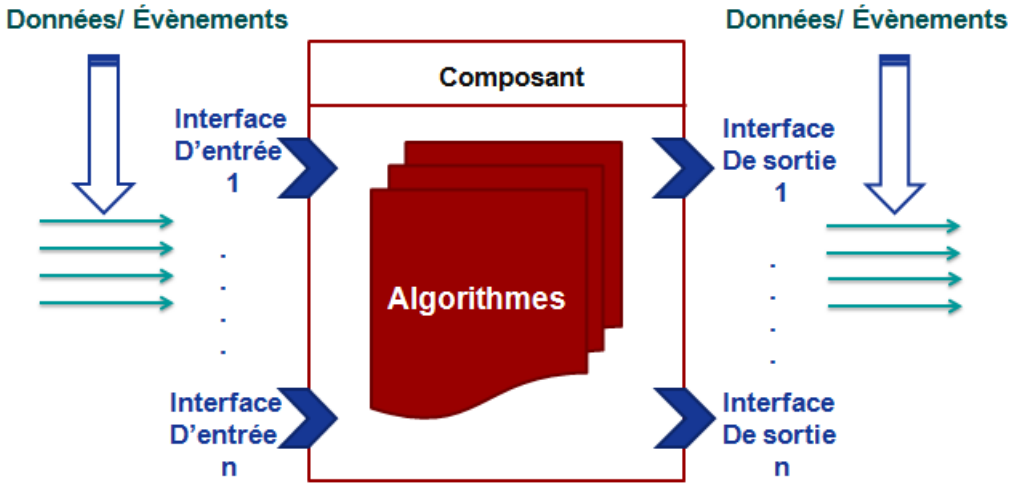

FIGURE 2.2 – Modèle du composant proposé.

Il existe plusieurs types de formalisation de composants dans la littérature. Nous présentons ci-après quelques formalismes bien connus :

Les auteurs dans [MSKf05] ont formalisé un composant comme une unité qui fournit et demande des services à d'autres composants. Il offre les services aux autres composants par les interfaces de sortie et les demande par les interfaces d'entrée. Chaque interface notée I est associée à un port qui interagit avec un ensemble de méthodes noté M_p. La formalisation du composant est représentée comme suit :

$$\Sigma = (P_\Sigma, R_\Sigma, \beta_\Sigma) \text{ où,}$$

- $P_\Sigma \subseteq I$: Interfaces d'entrées,

- $R_\Sigma \subseteq I$: Interfaces de sorties,

- $\beta_\Sigma : P_\Sigma \cup R_\Sigma \rightarrow M_p$: les méthodes qui interagissent avec les interfaces du composant.

Un composant CS selon [Fil02] est formalisé de cette façon :

$$CS = (\Theta, AX) \text{ où,}$$

- Θ : représente l'ensemble des signatures du composant qui sont des interfaces d'entrées/sorties du composant,

- AX : désigne l'ensemble des axiomes du composant, qui sont des formules décrivant l'ensemble de ses comportements.

Sifakis dans [Sif05], donne une formalisation algébrique d'un composant CA comme suit :

$$CA = (\mathbf{B}, GL, \oplus, \cong) \text{ où,}$$

- $\mathbf{B}$: représente l'ensemble des comportements atomiques du composant,

- $(GL, \oplus)$: représente un monoïde commutatif,

- $\cong$: est compatible avec la propriété de séparation structure-comportement.

Dans toutes les formalisations des composants citées auparavant, la spécification de la reconfiguration d'un composant n'est pas prise en compte. En effet, le changement des modes des composants n'est pas suffisamment formalisé pour spécifier une telle reconfiguration. De ce fait, il n'est pas possible de formaliser tous les comportements d'un composant reconfigurable en utilisant ces formalisations.

2.4.2 Composants des systèmes embarqués

Plusieurs travaux orientés composants dans le domaine des systèmes embarqués ont été conçus pour gérer la progression et la portabilité du code entre les différents systèmes. Il existe des propositions généralistes, ainsi que des implémentations pour des systèmes embarquées spécifiques. Tels que :

- Koala : C'est un modèle d'un composant développé essentiellement pour programmer les logiciels embarquées avec Philips [VOvdLKM00]. Dans Koala, les auteurs ont cherché à séparer le code métier de la configuration spécifique à la plate-forme. Ainsi, les développeurs du code métier encapsulent le code dans des composants et ne font aucune hypothèse sur les configurations dans lesquelles leurs composants sont utilisés. De même, les concepteurs de la configuration ne sont pas autorisés à changer le fonctionnement interne d'un composant Koala en fonction des spécificités de la plate-forme.

L'implémentation de Koala a été faite en langage C. La plupart des connexions entre les composants sont statiques et connues au moment de la configuration. Mais, malheureusement, Philips ne fournit pas l'accès au code source des composants logiciels de ses plates-formes.

- TAO : C'est un modèle d'un composant implémenté par CORBA [SLM98], dédie essentiellement pour les systèmes temps réel. Dans l'objectif de fournir un canevas qui contrôle les politiques et les mécanismes utilisés par CORBA en assurant un ordonnancement adapté.

Le modèle à composants CORBA a été optimisé pour assurer le bon fonctionnement sur des systèmes sur puce. Les optimisations cherchent à réduire l'empreinte mémoire et le temps d'exécution. En particulier, les améliorations portent sur la diminution de la taille des structures de composants, principalement sur la réduction de code des talons et des squelettes , ainsi que sur des optimisations du compilateur.

- RUNES : Les composants dans RUNES [CCM05] sont structurées dans des *canevas*. Ce dernier est un groupe de composants ayant une contrainte commune gérée comme une unité indépendante de déploiement et d'exécution. Les canevas peuvent être reconfigurés lors de l'exécution (ajout et suppression des composants). RUNES propose deux implémentations pour les plates-formes embarquées : l'une en langage Java et l'autre en langage C. Les composants RUNES sont interconnectés à l'aide d'interfaces et réceptacles.

Nous constatons que les approches à bases de composants sont présentes dans divers domaines des systèmes embarqués et utilisées pour des applications différentes : le multimédia et la communication, et abordent des aspects divers, dont le déploiement et la

qualité de service. Malheureusement, aucune des approches ne cible pas la reconfiguration des systèmes de contrôles industriels comme l'un de ses objectifs.

2.4.3 Composant On-line

Nous décrivons dans cette sous-section une étude des composants à caractère $On-line$, c'est-à-dire les composants qui sont accessibles pour interagir, composer et coordonner avec d'autres composants au moment de l'exécution tels que :

1) Component Object Model (COM) : est une technologie de Microsoft [Cor93] pour les systèmes d'exploitation utilisés par les programmeurs pour créer des composants logiciels réutilisables. COM est une architecture logicielle qui permet aux applications d'être construites à partir de composants logiciels binaires. COM est l'architecture sous-jacente qui constitue la base des services de logiciels de niveau supérieur, comme ceux qui sont fournis par OLE *(Object Linking and Embedding)*. OLE est un ensemble de services qui couvrent divers aspects de la fonctionnalité du système couramment nécessaire, y compris des documents composés, des contrôles personnalisés, transferts de données, et d'autres intéractions logicielles. DCOM (Distributed Component Object Model)[Lud03] est une extension des composants COM qui fournit un ensemble d'interfaces permettant aux clients et aux serveurs de communiquer dans le même ordinateur (Windows 95).

2) Composant .NET. : un composant *.NET* développé par Microsoft comme une extension au composant DCOM [Gro02] permet de fournir une interface programmable qui est accessible par les applications (applications souvent appelées clients). L'interface de **.NET** se compose d'un certain nombre de propriétés, de méthodes et d'événements qui sont exposés par les classes contenues dans le composant. En d'autres termes, un composant **.NET** est un ensemble compilé de classes qui prennent en charge les services fournis par le composant. Les classes exposent leurs services à travers les propriétés, les méthodes et les événements qui composent l'interface du composant. Au moment de l'exécution, un composant **.NET** est appelé et chargé en mémoire pour être utilisé par d'autres applications. Les composants **.NET** sont le plus souvent construits et testés en tant que projets **.NET** indépendants et ne font pas nécessairement partie d'un autre projet [BFJLT02] [KK06].

Discussions. Pour reconfigurer un composant .Net ou COM au moment de l'exécution, ce composant doit avoir un *manifeste* pour les informations d'activation nécessaires. Pour créer un manifeste d'application il faut avoir un éditeur XML avec l'identification du propriétaire du manifeste. Cependant, le processus de création d'un manifeste est très compliqué. En effet, l'approche de développement adaptée dans COM ou .Net utilise des syntaxes différentes pour décrire chacun des aspects d'une application. Par exemple l'insertion de l'en-tête doit être à chaque début du fichier. Ceci requiert la consolidation de toutes les méthodes et les descriptions des attributs d'un manifeste. Cette solution rend difficile la mise en œuvre des méthodes de reconfiguration d'un système de contrôle industriel à base de composants COM ou .Net.

3) Composant JavaBeans (JB) : les composants JavaBeans [CDF+98] sont des classes Java qui peuvent être facilement réutilisées et composées ensemble dans une application. Toute classe Java qui suit certaines conventions de conception est un composant JavaBeans. Les conventions de conception de composants JavaBeans régissent les propriétés de la classe et les méthodes publiques qui donnent accès aux propriétés. Une propriété du composant JavaBeans peut être : Lecture / écriture, lecture seule ou écriture seule.

Discussion. Des contributions pour reconfigurer un composant JB sont proposées dans [RAC+02]. Une reconfiguration paramétrique est manipulée par un composant JB d'une manière comme de la planification du changement dans les systèmes informatiques par la modification des valeurs des paramètres. Ces modifications peuvent être apportées au comportement du système sans avoir modifié l'un de ses composants logiciels exécutables. Ce type de reconfiguration pourrait être coordonné sur plusieurs nœuds. Par exemple, un paramètre peut être utilisé pour contrôler si un canal de communication est chiffré ou non. Ce qui nécessite la communication des composants afin de coordonner leur réponse à une modification de ce paramètre. La principale limitation de ces composants est qu'ils ne permettent pas de déduire l'architecture et la composition interne du composant afin de garantir la fiabilité de ses fonctionnements.

Synthèse.

En conclusion, pour les applications à base de composants On-line, le même composant peut être mis à jour ou reconfiguré à plusieurs applications. De plus, elles ne possèdent

pas la solution nécessaire pour la spécification d'interactions entre les composants des applications. De plus, l'utilisation de ce type du composant pour le développement d'un tel système de contrôle industriel adaptatif reconfigurable n'est pas possible.

2.4.4 Composant Off-line

Plusieurs travaux ont conduit à la définition des composants à caractère off-line, qui permet de spécifier l'application sur divers dispositifs pour les systèmes de contrôle critiques. Ce sont des composants à assembler avant la mise en œuvre en particulier dans le domaine des architectures logicielles afin d'en montrer leurs inter-connections et garantir a-priori les propriétés requises.

Nous nous intéressons dans cette partie aux composants proposés par le langage de description d'architecture (ADL) [CSLS99], qui permet de spécifier le composant comme un élément central sur lequel repose une architecture embarquée, qui intègre l'aspect conceptuel et formel pour décrire une architecture complète.

Un composant selon ADL est défini à l'aide de deux parties : Une partie extérieure qui correspond à ses interfaces d'entrées/sorties. Une autre partie intérieure qui correspond à l'implantation permettant la description des fonctionnements internes du composant. En général, un composant ADL est défini à l'aide de plusieurs paramètres tels que : Interface, type, contraintes, sémantique et propriétés non fonctionnelles [Chk10].

- **Interface :** permet de décrire les liens et les contraintes du composant pour communiquer avec l'extérieur. L'interface possède des ports d'entrées/sorties des événements/-données.

- **Type :** chaque composant possède un ou plusieurs types qui reflètent son implantation ainsi que l'ensemble des fonctionnalités fournies par le composant aux autres composants de l'architecture.

- **Contraintes :** sont les règles et les protocoles qui définissent les modes d'utilisation du composant et son interaction avec d'autres composants extérieurs.

- **Sémantique :** permet de spécifier le dynamisme interne et externe du composant avec un modèle abstrait qui doit respecter les différents niveaux de raffinement du composant.

- **Propriétés non fonctionnelles :** Ce sont des propriétés en relation avec la sécurité, la performance et la portabilité du composant, , définies pour assurer la séparation entre la spécification et les aspects fonctionnels d'un composant.

Nous présentons ci-après les principaux langages de description d'architecture les plus utilisées par la communauté des systèmes embarqués pour décrire une architecture complète du système :

1) ACME. Le projet ACME [GMW97b] [GMW97a] lancé début 1995, dans le but de fournir un langage commun ACME (Architecture Description Interchange Language) pouvant être utilisé pour assurer l'échange de descriptions architecturales entre une variété d'outils de conception architecturale.

Le langage ACME fournit trois fonctions principales :

1) Échange architectural : en fournissant un format d'échange générique pour des conceptions architecturales. ACME permet aux développeurs d'outils d'architecture d'intégrer facilement leurs outils avec d'autres outils complémentaires.

2) Extensible pour les outils de conception et d'analyse de l'architecture : L'origine de ACME comme langage d'échange générique, permet aux outils développés en ACME d'être compatible avec plusieurs outils des langages d'architecture de description existants sans aucun effort de développement supplémentaire.

3) Description de l'architecture : ACME a émergé comme un langage utile de description d'architecture dans son propre droit. Il fournit un ensemble simple de constructions pour décrire la structure architecturale, les types, les styles architecturaux et les propriétés des éléments architecturaux.

Discussion. Les auteurs dans [JBCG05] ont proposé des extensions de reconfigurations pour les composants ACME par la proposition de quatre constructions suivantes : (i) *on (<condition>) do <actions>* : cette construction permet au programmeur ACME d'exprimer en temps-réel les conditions *(condition clause)* dans lesquelles est programmée la reconfiguration capable de changer l'action *(actions clause).* (ii)*detach <role> from <port>* qui permet de supprimer un attachement entre un rôle et un port et *remove < elem >* : supprime un composant, connecteur ou représentation existante. (iii) *depen-*

dencies < *statements* > *:* cette construction permet l'expression des dépendances d'exécution entre les éléments architecturaux. (iv) : *active property* < *propertydefinition* > : ceci est un type spécial de la propriété qui ne peut être fixé aux ports et aux rôles du composant. Cependant, la reconfiguration choisie pour ACME ne permet pas d'imposer et de contraindre la sémantique d'exécution du composant. La reconfiguration d'un système de contrôle industriel à base des composants ACME nécessite de mettre en œuvre des méthodes d'analyse qui devront garantir les propriétés de comportement d'exécution de ces composants.

2) Darwin. est un langage de description de structures de logiciels qui a été inventé depuis 1991, sous diverses formes syntaxiques. Il a été développé à l'origine comme un langage de configuration pour le projet REX [SN92] à base de langage antérieur de configuration CONIC [MKS89]. Ce langage permet la séparation entre la structure du programme et le comportement algorithmique. Darwin encourage les approches par composant ou par object à programmer la structure dans laquelle un composant se cache derrière le comportement d'une interface bien définie. Les programmes sont construits par des instances, des types et des composants en contraignant leurs interfaces ensemble. La forme générale d'un programme Darwin est donc représentée comme un arbre dans lequel la racine et tous les nœuds intermédiaires sont des composants composites ; les feuilles sont des composants primitifs qui expriment un comportement par opposition aux aspects structurels [DoCM97].

Les interfaces des composants sont constituées d'un ou plusieurs éléments dont chacun représente un service fourni ou requis par ce composant. La spécification d'un service est un aspect inhabituel, mais extrêmement puissant du langage car elle réduit non seulement les dépendances implicites de composants sur leur environnement, mais elle augmente également la réutilisation de chaque composant. Les primitives du composant représentent des unités algorithmiques avec des interfaces bien définies.

Darwin possède à la fois une représentation textuelle et graphique. La forme textuelle est plus riche, car elle permet la spécification des conditions et les itérations qui sont évaluées lorsque la configuration est élaborée au moment de l'exécution. La forme graphique décrit essentiellement la structure après l'état-évaluation. Pour des raisons de portabilité,

le compilateur de Darwin est écrit en Java. Il est découplé de son environnement par une définition abstraite de son générateur de code.

Discussion. Darwin ne gère pas l'évolution pendant l'exécution de ces composants. Il assure la cohérence entre configurations et spécifications. Darwin gère les changements de configuration sous formes transactionnelles sur le système au moment de l'exécution. En effet, Darwin ne conserve pas la trace de changements de ces composants comme des listes d'opérations de changement tel que l'ajout, suppression ou modification dans les représentations des architectures.

3) Wright : en tant que langage de description d'architecture, Wright est utilisé pour fournir une précision significative et abstraite de la spécification et l'analyse d'architecture avec l'algèbre de processus, à la fois pour les architectures de systèmes logiciels individuels et les familles de systèmes.

Wright sert également à l'exploration des abstractions architecturales elles-mêmes. En particulier, le travail avec Wright est focalisé sur le concept de types connecteurs explicites, la vérification automatique des propriétés architecturales, et la formalisation des styles architecturaux [All97].

Wright définit le composant par une interface et un comportement, où l'interface possède des ports pour l'interaction avec d'autres composants de son environnement à l'aide de connecteurs qui possèdent des configurations spéciales, le comportement supporte les fonctionnalités d'une application donnée [AG94].

4) Rapide : le langage Rapide développé par David Luckham à Stanford [Luc96], permet la spécification des systèmes distribués à grande échelle avec une nouvelle technologie. Cette dernière est basée sur une nouvelle génération de langages informatiques appelés EADLs (Executable Architecture Definition Language) qui est un ensemble d'outils innovants favorisant l'utilisation de EADLs dans le développement évolutionnaire avec une analyse rigoureuse des systèmes à grande échelle.

Rapide [LKA+95] s'articule sur le principe de POSET (Partial Ordered event Sets), conçu pour soutenir le développement des systèmes à grande échelle à base de composants, par la définition de l'architecture en tant que framework de développement. Ainsi,

Rapide adopte un nouveau modèle d'exécution basé sur des événements pour les systèmes distribués. Un composant selon Rapide est défini par une interface qui gère l'interaction avec d'autres composants, et un module d'exécution qui possède le code exécutable et un ensemble des sous-composants.

Discussion. Les deux langages Rapide et Wright permettent la modélisation d'une architecture avec la concurrence des taches. Un travail a été fait pour assurer la transformation des spécifications avec Rapide en des spécifications avec Wright. Mais, à cause des différences de sémantique entre les deux langages, seule la partie structurelle de l'architecture à été transférée. L'inconvénient de ces deux langages est qu'ils ne précisent pas la correspondance entre le composant matériel et le composant logiciel. C'est pour cette raison que nous ne pouvons pas reconfigurer une application réelle basée sur la description de ces deux langages.

5) Aesop. est développé par David Garlan [Gar95] pour souligner les styles architecturaux. Aesop est également un ensemble d'outils permettant l'assemblage des environnements pour la conception et l'analyse d'architecture logicielle, ainsi, elle permet de définir plusieurs styles pour une architecture. Un composant selon Aesop est une unité logique fonctionnelle, représentée par une interface qui possède des ports pour l'interaction extérieure où chaque port a un rôle à exécuter.

En utilisant les informations fournies par les descriptions du style et une infrastructure partagée commune à tous les environnements Aesop, chacun de ces environnements prend en charge :

1) une palette de conception pour les types d'éléments correspondant au vocabulaire du style,

2) vérifier que les compositions des éléments de conception satisfont les contraintes topologiques du style,

3) spécification optionnelle de la sémantiques des éléments,

4) une interface qui permet à des outils externes d'analyser et de manipuler les descriptions architecturales,

5) plusieurs visualisations spécifiques d'information architecturale conjointement avec

un ou plusieurs éditeurs graphiques pour les manipuler.

Discussion. Aesop [MT00] fournit des invariants stylistiques, un composant peut également être limité via des attributs. La reconfiguration peut fournir une plus vaste extension de sous-type de composants par l'application de comportement créant un sous-style d'un style architectural donné. Une sous-classe Aesop doit fournir un strict comportement du sous-type pour les opérations qu'il réalise, mais peut également introduire plus de sources de défaillance à l'égard de sa super-classe. En revanche, avec Aesop on ne peut pas générer automatiquement des environnements d'exécution de ces composants. Ainsi, les paramètres d'hétérogénéité entre les composants ne sont pas pris en compte.

6) C2 : est l'un des quelques langages de description d'architecture [MRT99] qui supporte le dynamisme de l'architecture sans contrainte. Une architecture avec C2 est représentée par un réseau de composants en exécution d'une manière concurrente. En C2, les composants et les connecteurs sont des entités de première classe qui possèdent un haut et un bas, le passage de messages est utilisé pour relier des composants les uns aux autres par des connecteurs avec certaines restrictions telles que, le haut d'un composant doit être obligatoirement connecté au bas d'un seul connecteur et le bas d'un composant par le haut d'un seul connecteur. En outre, les connecteurs peuvent effectuer un filtrage des messages.

C2 [MORT96] permet la liaison dynamique des composants aux connecteurs. Toutes les possibilités de reconfiguration d'une architecture sont autorisées. Une caractéristique intéressante de C2 est la reconfiguration dynamique qui est effectuée en utilisant la transmission de messages entre les composants, de la même manière que la communication ordinaire entre eux. En particulier, un certain nombre de problèmes ne sont pas résolus dans leur reconfiguration, à savoir, comment désactiver les composants qui sont en cours d'exécution sans causer la perte de messages, ou la dépendance entre les composants. Ainsi que les problématiques d'initialisation d'un composant nouvellement ajouté, et la cohérence de l'architecture à travers les changements de structure.

7) UniCon. organisé autour de deux constructions symétriques : composants et connecteurs dont chacun possède une partie de la spécification et une autre d'implémentation. Les composants représentent des unités de calcul et de données dans un logiciel du système. Ils sont utilisés pour organiser le calcul et les données dans les parties qui ont une sémantique

et des comportements bien définis. Les connecteurs représentent des classes d'interactions entre les composants. Ils sont utilisés pour assurer la communication et les interactions entre les composants [Wil06].

Les composants UniCon sont spécifiés par une interface. L'interface définit les engagements de calcul que le composant peut faire et les contraintes sur la façon dont le composant peut être utilisé. Il fournit également les garanties concernant la performance et le comportement du composant. L'interface contient trois types d'informations : le type du composant, les propriétés, les services fournis et requis du composant.

Les connecteurs UniCon sont spécifiés à l'aide d'un protocole. Le protocole définit les intéractions permises entre un ensemble de composants et fournit des garanties de ces interactions. Il contient trois types d'informations : le type du connecteur, les propriétés, les services fournis et requis par le connecteur.

Discussion. Les rôles et les coordinations entre les composants UniCom et C2 sont prédéfinis au niveau du protocole, par la définition des attributs pour chaque rôle ainsi que pour les interfaces des utilisateurs. Ce qui mène à un problème d'adaptation de façon automatique et dynamique en cas de faute ou panne qui vont apparaitre lors des changements imprévisibles de son environnement d'exécution.

8) Meta-H : fournit les moyens d'exprimer, d'analyser et d'implémenter une architecture en temps réel pour les systèmes embarqués. L'architecture se réfère à la manière dont les composants d'un système sont connectés et contrôlés. Ainsi, Meta-H décrit les détails des composants (algorithmes) et le champ d'application pour caractériser l'architecture pour prédire le comportement du système et permettre un changement rapide dans les composants. Meta-H [VK00] possède un ensemble d'outils développés autour d'un langage de spécification, de la même catégorie comme certains outils commerciaux (Rational Rose, Aonix Software-Through-Pictures, Cayenne TeamWork) pour spécialiser la spécification, l'analyse et la vérification des systèmes embarqués.

Discussion. La reconfiguration dynamique de Meta-H s'articule autour des modes de fonctionnements, qui sont des configurations. Par exemple (Figure 2.3), l'activation du processus où les connexions, changements de modes *arrêter*et *démarrer* des sous-ensembles

de processus et changer les schémas de connexions de messages et d'événements et les connexions d'événements créent un diagramme hiérarchique de transition de mode. Ce type de reconfiguration permet d'avoir une meilleure connaissance sur les fonctionnements du système dans un contexte spécifique. Cependant, dans certains cas d'applications pour des systèmes complexes, l'utilisation de ce type de reconfiguration est insuffisante pour prouver l'ensemble des propriétés de sûreté que doit vérifier le système.

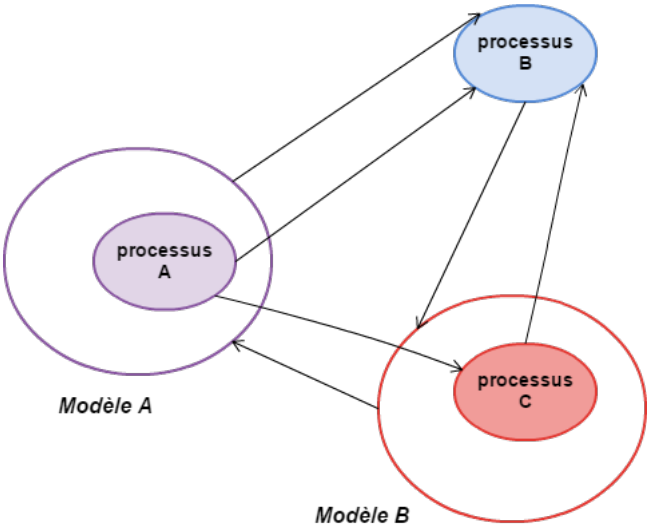

FIGURE 2.3 – Changement de modes avec Meta-H.

9) AADL. *Architecture Analysis and Design Language* (AADL) [FGH06] est un langage de description d'architecture normalisé par SAE[2]. AADL a été développé en 1990 pour le domaine de l'avionique, il était connu comme Avionique architecture Description Language. AADL est utilisé pour modéliser l'architecture logicielle et matérielle d'un système temps réel embarqué sous forme de composants logiciels et matériels, afin d'avoir un modèle unique et facile pour les outils d'analyse ayant une seule représentation unique du système. AADL [HAS06] spécifie les caractéristiques des composants du système à l'aide de propriétés. AADL permet la spécification et l'analyse du système temps réel embarqué de haute fiabilité, les systèmes complexes et les capacités de performance des systèmes. Il permet aussi la répartition des composants logiciels sur des composants matériels. Nous reviendrons à la section 2.5 avec une étude détaillée du langage AADL. Nous choisissons

2. Society of Automotive Engineers

de travailler avec le langage AADL dans le cadre de cette thèse parce-qu'il possède un ensemble d'avantages tels que :

- AADL offre la possibilité de décrire l'architecture complète du système embarqué pour une meilleure maîtrise de sa complexité,

- Possède une bibliothèque riche pour la réutilisation des applications basées sur AADL,

- La possibilité de vérifier des propriétés,

- La précision dans la modélisation qui facilite la conception d'un tel système,

- Automatise la production de l'application (génération de code...),

- Facilite l'analyse (test d'ordonançabilité, vivacité, etc...),

- La possibilités d'effectuer des corrections sur l'architecture.

⇒ Nous constatons selon ces avantages qu'un composant AADL représente un cadre utile pour étudier les composants dans le cadre de notre thèse.

2.4.5 Sécurité des Approches par Composants

Les systèmes à base de composants évoluent d'une manière générique. De ce fait, la sécurité devient une préoccupation de plus en plus importante pour ces systèmes et leurs composants. La sécurité est une propriété émergente, il est donc insuffisant de sécuriser un composant à part. Mais, l'ensemble du système doit être sécurisé où tous les composants concernés doivent collaborer pour assurer la sécurité du système.

UML est un langage de modélisation et de conception standard. Il y a eu plusieurs approches basées sur UML pour modéliser la sécurité, $UMLsec$ [JÖ2] qui permet d'exprimer la sécurité des informations pertinentes avec les diagrammes dans la spécification du système, $SecureUML$ [LBD02] basé sur le contrôle d'accès qui est basé sur les rôles pour spécifier des contraintes d'autorisation l'accès aux composants.

Les auteurs dans [CLWK00] proposent un modèle d'assurance de qualité QA (Quality Assurance) pour les logiciels à base de composants qui couvre l'analyse, la spécification, la vérification, la conception, l'implémentation et le test de l'architecture du système à base des composants.

Le travail dans [RKK07] présente une approche fondée sur la dépendance itérative pour la modélisation de la fiabilité d'un système en utilisant AADL. Cette approche est une partie d'un framework complet qui permet l'analyse de la sûreté de fonctionnement et l'évaluation des modèles AADL pour appuyer l'analyse des logiciels et l'architecture du système dans divers domaines.

AADL recourt à l'architecture MILS4 [HFM08] pour valider la sécurité de la conception des systèmes. MILS4 [AfHOT06] utilise deux mécanismes pour diviser un système sécurisé par la partition et la séparation en couches. L'architecture MILS isole les processus dans des partitions qui définissent une collection d'objets de données, le code et les ressources du système peuvent être évalués séparément. Chaque partition est divisée en trois couches : couche de séparation du noyau, couche Middleware des services et couche d'application dont chacune est responsable de son propre domaine de sécurité.

2.4.6 Vérification Formelle par Model-Checking

2.4.6.1 Vérification Formelle

La vérification formelle d'un système consiste à modéliser le système et ses interactions puis à prouver un ensemble de propriétés sur le modèle. Le modèle d'un système est une représentation mathématique du comportement du système. Une propriété est un énoncé qui exprime une spécification voulue pour le système, par exemple dans le cadre d'un système Radar on veut vérifier que le délai d'envoi des signaux à l'antenne dans les mauvaises conditions climatiques ne dépasse pas 2ms. Deux logiques sont fréquemment utilisées pour la vérification formelle :

- La logique CTL : (Computation Tree Logic) a été définie au début des années 1980 dans [QS82], [CES86]. Elle est interprétée sur des structures de Kripke, c'est-à-dire des automates finis dont les états sont étiquetés par des propositions atomiques. Elle permet d'exprimer des propriétés sur ces états, faisant intervenir l'arbre des exécutions issues de cet état.

- La logique TCTL : (Timed Computation Tree Logic)représente une extension proposée dans [ACD93] de la logique CTL, afin d'énoncer des propriétés temporisées, i.e

qui font intervenir des informations quantitatives sur le temps.

2.4.6.2 Model-Cheking

Le model-checking [CE82], [QS82] (vérification du modèle) est une approche automatisée permettant de vérifier qu'un modèle de système est conforme à ses spécifications. Le comportement du système est formellement modélisé, via des automates, réseaux de Petri, algèbres de processus, . . . et les spécifications, exprimant les propriétés attendues du système, sont formellement exprimées par exemple via des formules de logiques temporelles.

En pratique, les propriétés du système sont souvent classées en deux grandes catégories informelles. Les propriétés de *sûreté* énonce qu'une situation particulière ne peut être atteinte. Les propriétés de *vivacité* énonce quelque chose de mauvais (ou de bon) qui finira par se produire [BBF$^+$13].

La Figure 2.4, tirée de [Saa11], donne un aperçu simplifié du cycle d'utilisation du modelchecking.

Le cycle peut se diviser en trois phases :

1. Modélisation formelle du comportement du système,

2. Expression formelle des propriétés attendues,

3. Si une propriété n'est pas satisfaite, un contre-exemple est produit qui décrit un scénario possible d'erreur (i.e de violation de la propriété). L'analyse de celui-ci aide à apporter les corrections nécessaires que ce soit sur la modélisation du comportement du système ou sur l'expression formelle des propriétés attendues.

Ce cycle est répété jusqu'à ce que toutes les formules, c'est à dire toutes les spécifications, soient vérifiées.

2.4.6.3 UPPAAL

Cet outil a été développé en 1995, conjointement par l'université d'UPPsala (UPP) en Suède et l'université d'AALborg (AAL) au Danemark [LPY97], spécifié pour la modélisation, la simulation et la vérification des systèmes temps réel modélisés par des automates temporisés avec variables entières bornées, actions urgentes, etc... Les propriétés

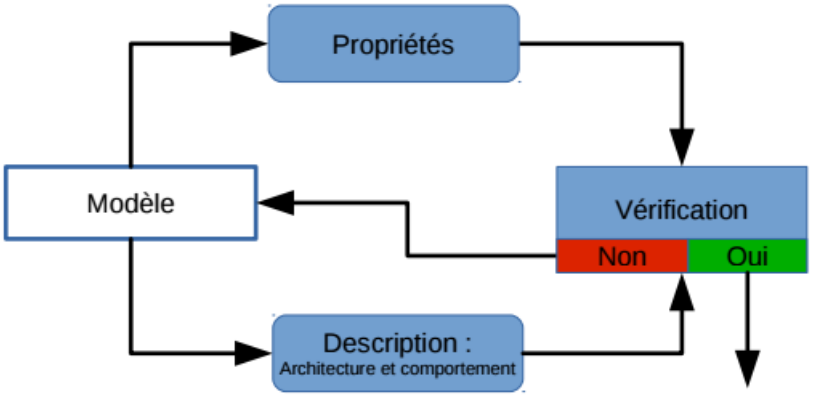

FIGURE 2.4 – Cycle d'utilisation du model-checking.

qui peuvent être vérifiées sont principalement des propriétés d'accessibilité, de vivacité ou d'états bloquants. La logique utilisée par UPPAAL est uniquement un fragment de TCTL qui ne permet pas de vérifier l'ensemble des formules TCTL, les opérateurs de temps ne pouvant pas être imbriqués. L'algorithme implémenté dans UPPAAL est essentiellement un algorithme d'analyse en avant.

Les avantages d'utilisation d'UPPAAL par rapport aux autres outils sont d'une part son interface graphique, très conviviale et d'autre part, son module de simulation qui permet lors de la phase de modélisation de faire des tests du modèle et de détecter d'éventuelles erreurs de modélisation. En effet, une des caractéristiques de UPPAAL est qu'il basé sur une communication asynchrone par canaux. D'autre part, un composant ne peut pas recevoir un signal s'il n'a pas été émis par un autre composant. Cela oblige à respecter l'évolution logique entre les différents états d'un signal.

Le vérificateur (menu 'Verifier') d'UPPAAL permet de vérifier les propriétés sous forme TCTL. Nous pouvons vérifier une ou plusieurs propriétés à la fois. Nous pouvons insérer de nouvelles propriétés ou supprimer des propriétés. Nous pouvons aussi basculer l'affichage pour voir les propriétés ou leurs commentaires respectifs dans la liste. Quand une propriété est sélectionnée, nous pouvons modifier sa définition ou ajouter des commentaires pour documenter ce que signifie cette propriété informellement.

2.4.7 Synthèse

Bien que les langages cités dans les paragraphes précédents n'utilisent pas la même terminologie, ils permettent de spécifier un composant qu'il soit en ligne ou hors ligne à l'aide de son interface, son type et ses interactions avec les autres composants d'un système.

Les approches proposées pour les ADLs sont plutôt statiques, la reconfiguration et la flexibilité dynamique du composant ou du système à base des composant sont peu ou pas du tout exprimées.

Ces approches ne permettent pas de faire évoluer une architecture d'un composant vers une nouvelle architecture reconfigurable et flexible non définie au moment de la création du composant. C'est pourquoi nous proposons, dans le chapitre 3, un modèle qui prend en compte les concepts communs de ces différentes évolutions et reconfigurations, la proposition d'une approche pour la gestion des reconfigurations cohérentes et synchrones dans le chapitre 4 et l'analyse de sécurité de ces composants reconfigurables dans le chapitre 5.

2.5 Langage de Description et d'Analyse d'Architectures (AADL)

AADL (Architecture Analysis & Design Language) [FG12] est développé par Honeywell en 1990 et la société SAE (Society of Automotive Engineers) qui a publié son standard international depuis 2001 et la version 1.0 en 2004. AADL est un langage conçu pour la description et l'analyse d'architectures des systèmes temps-réels embarqués à base du langage Meta-H. Nous exposons dans cette section les principes de AADL en tenant compte de ses composants, ses ports, ses outils existants, les annexes tel que l'annexe comportemental et l'annexe de gestion des erreurs et les modes en AADL. Nous finissons cette section par une synthèse dans laquelle nous discutons les travaux au tour de la reconfiguration d'AADL pour mieux positionner notre contribution.

2.5.1 Principes du Langage

AADL [AL04] est un langage de description d'architecture utilisé pour modéliser l'architecture logicielle et matérielle d'un système temps-réel embarqué avec des propriétés

de son comportement tels que la politique d'ordonnancement et les allocations des ressources. La description selon le standard AADL est représentée par trois syntaxes ; textuelle pour contrôler les détails du système, XML pour faciliter l'interopérabilité entre les outils existants et la représentation graphique afin d'avoir une vision globale sur la totalité du système. Par cette diversité de syntaxe, AADL permet de faciliter la conception d'un tel système par l'automatisation de la production et la simplicité d'analyse des comportements de l'application.

AADL définit trois types des composants : composants matériels pour décrire la plateforme d'exécution, composants logiciels pour modéliser la partie applicative du système et composants système qui englobent les composants matériels et logiciels. La connectivité entre ces composants forme une architecture AADL distribuée pour offrir une grande souplesse de la modélisation.

2.5.2 Définition des Composants AADL

Un composant AADL est défini par une interface *(component type)* désignée par les ports d'entrées/sorties et une implémentation *(component implementation)* qui définit la structure interne du composant avec les sous-composants, les algorithmes... etc. Une interface peut correspondre à une ou plusieurs implémentations.

AADL définit plusieurs catégories de composants, classés en trois grands types :

1) Composants logiciels *(software components)* : AADL possède cinq types de composants illustrés dans la Figure 2.5 avec la syntaxe graphique :

FIGURE 2.5 – Composants logiciels d'AADL.

i) Composant Processus : Un processus représente un espace mémoire protégé dans lequel s'exécutent des composants threads, où la protection est fournie à partir d'autres composants tels que les sous-programmes ou les données. Chaque processus possède des propriétés tels que les informations du fichier source, les protocoles d'ordonnancement et

les contraintes de liaison, ainsi que les contraintes au niveau de ses interfaces (types) et son implantation.

ii) Composant Thread : Un thread est une unité comme un processus léger en exécution séquentielle dans un code source qui s'exécute dans un espace d'adressage virtuel et protégé d'un processus. Un thread représente aussi un chemin d'exécution. Les propriétés du thread sont utilisées pour représenter ses attributs, telles que la période, la politique de déclenchement (périodique, apériodique, sporadique), l'échéance, etc.

ii) Composant Groupe-threads : Un groupe-threads est une abstraction d'un composant pour organiser logiquement les threads, les données et les processus afin de former des hiérarchies. Mais, il ne représente pas un espace d'adressage virtuel ou une unité d'exécution.

iv) Composant Données : Un composant données en AADL représente des données statiques (par exemple, des données numériques ou texte) et les types de données dans un système. Plus précisément, un composant données est utilisé pour représenter : les types de données d'application, la sous-structure de types de données qui peuvent être stockées ou échangées entre les composants, et les instances de données.

v) Composant Sous-programme : Un sous-programme représente un fragment de code exécutable d'une manière séquentielle, qui est appelé par des composants avec ou sans paramètres pour fournir ou demander des services.

2) Composants Matériels *(Execution Platform Components)* : AADL offre quatre types de composants matériels qui représentent des ressources de calcul dans un système. Figure 2.6 montre la représentation de ces quatre composants avec la syntaxe graphique.

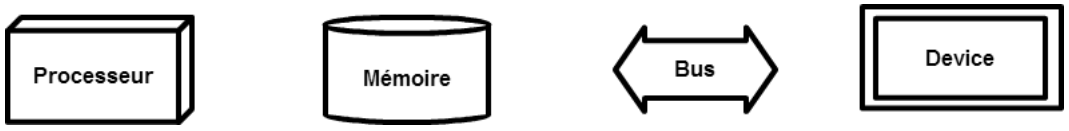

FIGURE 2.6 – Composants Matériels d'AADL.

i) Composant Processeur : un composant processeur est le responsable d'exécuter et l'ordonnancement des composants threads. Les processeurs peuvent exécuter les threads qui sont déclarés au niveau des applications logicielles du système ou threads des

composants accessibles à partir de ces processeurs.

ii) Composant Mémoire : un composant mémoire représente les composants de stockage pour les données et le code exécutable : RAM, ROM, disque dur, ect. Ce composant possède des propriétés telles que la taille du texte et le nombre de mots.

iii) Composant Bus : un composant bus représente les matériels et les protocoles de communication associés qui permettent l'interaction entre les divers composants de la plateforme d'exécution. Les bus peuvent être connectés directement à d'autres bus pour représenter les communications complexes inter-réseaux. Ainsi, les connexions entre les différents composants peuvent être assurées par une séquence de bus ou un bus de processeurs intermédiaires.

iv) Device : un composant device représente les éléments dont la structure interne n'est pas visible avec des comportements complexes et qui possède une interface pour se connecter à l'extérieur. Il peut s'agir par exemple de capteurs.

3) Composants Systèmes *(System Components)* : la Figure 2.7 montre la représentation d'un composant système avec la syntaxe graphique. Le composant système est un composant sous forme d'un bloc logique d'entités qui regroupent les composants matériels et logiciels pour faciliter la structuration de l'architecture. Le composant système ne possède pas de comportements sémantiques sur ses composants.

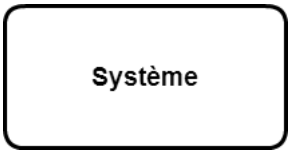

FIGURE 2.7 – Composant système d'AADL.

2.5.3 Structure Interne des Composants AADL

La structure interne des composants AADL est spécifiée au niveau de la déclaration des implantations des composants qui est représentée par les sous-composants internes et l'appel de sous-programmes :

- *Sous-composants* : L'implantation d'un composant spécifie une structure interne en terme de sous-composants, où la déclaration d'un sous-composant doit être associée à

Composant	Sous-Composants
Donnée	Données
Sous-Programmes	-
Thread	Données
Processus	Threads, Données
Processeur	Mémoires
Device	-
Bus	-
Mémoire	Mémoires
Système	Données, Threads, Processus, Processeurs, Mémoires, Bus, Devices, Systèmes

TABLE 2.1 – Compositions des composants AADL.

une famille de trois familles, une implantation et un type de composant pour modéliser une architecture AADL arborescente avec des règles de composition des composants. Le tableau 2.1 montre la synthèse des compositions possibles.

Dans notre travail nous considérons les sous-composants d'un composant AADL comme des algorithmes internes.

- *Appels de sous-programmes :* Les appels de sous-programmes se fait en séquences d'appels dans les sous-programmes ainsi que les composants threads avec une syntaxe semblable à la déclaration d'un sous-composant. L'appel d'un sous-programme en AADL correspond aux principes d'appels d'une procédure ou fonctions au niveau des langages de programmation : chaque sous-programme appelé est implicitement instancié dans le composant mémoire du composant processus, et peut-être appelé plusieurs fois avec le même composant.

2.5.4 Ports d'interface

Les éléments d'interface sont déclarés dans la section des caractéristiques *features* du composant avec la déclaration du type. Les chemins d'interaction (les connexions) entre les éléments d'interface sont déclarés explicitement dans les implémentations de composants.

AADL définit deux catégories de ports qui correspondent à l'interaction entre les interfaces de composants (Figure 2.8). Les ports simples sont répartis en trois types.

1. Ports simples : un port simple représente une interface de communication pour l'échange des données, des événements, ou les deux (données d'événement) entre les com-

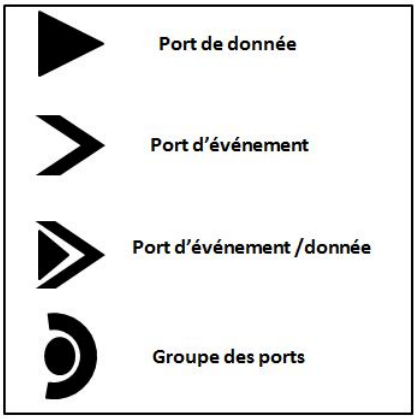

FIGURE 2.8 – Ports d'AADL.

posants. Les ports sont classés en trois types comme suit :

- **les ports d'événements (Event ports) :** sont des ports réservés pour la transmission des événements qui correspondent à des signaux qui peuvent également déclencher l'exécution des sous-programmes, des threads, des processeurs, ou des devices qui peuvent être en file d'attente,
- **les ports de données (Data ports) :** sont des ports réservés pour la transmission des données. Ils ne déclenchent rien à la réception, ils sont représentés par des variables avec la source texte. La structure de la variable/tableau est définie par le type de données *[data classifier]*. Les connexions entre les ports de données sont immédiates ou différées,
- **les ports d'événements/données (Event/Data ports) :** sont les ports qui possèdent les fonctionnalités des deux premiers types de ports pour la transmission de messages avec des files d'attente. Ils permettent la transmission des données tout en générant un événement à la réception.

2. Groupes des ports (port Group) : les groupes des ports sont définis pour faciliter la manipulation de ports associés. Ils sont déclarés comme des composants.

2.5.5 Outils AADL Existants

AADL possède plusieurs outils de modélisation et de description d'architecture :

- OSATE2 : est la référence d'implémentation de la version 2.0 d'AADL [FG06]. Il supporte le langage de base avec AADL complété avec des plugins supplémentaires pour étendre ses capacités de modélisation et d'analyse : plugins OSATE2 pour la validation, un framework OSATE2 ARINC 653 et un modèle d'erreur Annex version 2.0.

- AADL Inspector : est un framework d'un modèle de traitement pour AADL, qui fournit un outil léger et extensible pour effectuer une analyse statique et dynamique des architectures AADL. Il permet aussi de se connecter facilement avec un outil de vérification, de compilation ou de génération de code pour AADL.

AADL Inspector [DMR+14] fournit les caractéristiques suivantes :

- Analyse syntaxique d'AADL(aadlrev),

- Analyse des règles statiques,

- Analyse des paramètres architecturaux,

- Analyse de l'ordonnançabilité avec l'outil CHEDDAR,

- Simulation dynamique des modèles AADL, l'intégration du moteur de simulations multi-agent MARZIN.

- Ocarina : est un outil utilisé pour analyser et construire des applications à partir de descriptions AADL. Grâce à son architecture modulaire, ocarina peut également être utilisé pour ajouter des fonctions AADL pour les applications existantes [LZPH09] [HZPK08]. Ocarina prend en charge les fonctionnalités de AADL 1.0 et AADL 2.0 suivantes :

- Analyse syntaxique de modèles AADL pour les deux versions 1.0 et 2.0,

- Vérification sémantique,

- Génération de codes avec les générateurs de code suivants : PolyORB-HI/Ada, PolyORB-HI/C et POK,

- Modèle de vérification en utilisant les réseaux de Petri,

- Calcul de WCET avec l'outil Bound-T de Tidorum Ltd,

- Analyse de l'ordonnançabilité de modèles AADL avec l'outil CHEDDAR et MAST.

2.5.6 Annexe d'AADL

Les annexes AADL sont un moyen d'associer des informations aux composants d'une description contrairement aux propriétés. Dans ces annexes sont déclarées les extensions du langage sous forme conceptuelle et syntaxique afin de spécifier le comportement des composants.

L'annexe comportementale d'AADL est décrite à l'aide des éléments de base suivants :

- Variable d'état *(state variable)* : Cet élément permet de déclarer les variables dans la section d'initialisation,

- État *(State)* : cet élément permet de déclarer les états de chaque composant,

- Transition *(Transition)* : cette transition permet de décrire les transitions de l'état initial vers l'état final. La transition peut avoir une garde (une condition booléenne par exemple) et une action à exécuter si la garde est vraie.

La garde de l'annexe comportementale d'AADL peut avoir trois valeurs :

– la garde est toujours vraie,

– la garde sous forme d'une expression,

– la garde sous forme d'une expression et d'un port (event).

2.5.7 L'annexe de gestion des erreurs en AADL

L'annexe du modèle d'erreurs en AADL intitulée *EMV2* décrit dans [GZJ16] est une extension standard du langage AADL dans le but de supporter la modélisation des erreurs au niveau d'architecture et pour analyser la sécurité de manière automatique. Cette annexe facilite la conception de comportements d'erreurs qui peuvent être utilisés pour plusieurs modélisations et analyses d'activités. Grâce à cette annexe, l'utilisateur peut annoter une architecture du système embarqué exprimée dans un noyau AADL pour décrire les types de pannes, le comportement des défauts d'un composant AADL et la propagation de panne affectant des composants connexes ou hiérarchiques.

EMV2 se focalise sur la modélisation de l'architecture d'erreurs selon trois niveaux d'abstraction :

— **Propagation de flux d'erreurs :** pour chaque composant, le concepteur indique des types spécifiques d'erreurs qui se propagent entre les composants lors de la coordination entre eux. Le flux des propagations d'erreurs de l'entrée aux sorties peut également être décrite par le concepteur pour spécifier s'il est la source d'erreurs, le récepteur d'erreurs ou le chemin d'erreurs,

— **Comportement d'erreur du composant :** pour chaque composant (type ou implémentation), le concepteur peut associer une machine d'états de comportement d'erreurs à un ensemble d'états et de transitions qui se produisent dans des conditions spécifiées. Les transitions de comportement d'erreurs peuvent être déclenchées par des propagations d'erreurs entrantes, des événements de détection d'erreurs, des événements de récupération et des événements de réparation à divers points de propagation d'erreurs,

— **Comportement d'erreur composite :** Il vise à fournir le mappage entre la spécification de comportement d'erreurs du composant abstrait du système de base et la spécification du comportement de ses sous-systèmes.

De plus, *EMV2* [DF14] introduit le concept de type d'erreur pour caractériser les défauts, les échecs et les propagations. Il comprend un ensemble de types d'erreurs prédéfinies comme point de départ pour l'identification systématique de différents types de propagations d'erreurs fournissant une ontologie de propagation d'erreurs. Les utilisateurs peuvent adapter et étendre cette ontologie à des domaines spécifiques.

2.5.8 Les Modes et les transitions de mode en AADL

Les modes des composants AADL permettent de décrire des modes de fonctionnement de l'architecture d'un composant AADL. Ces modes sont spécifiés dans les implantations des composants et représentent les états opérationnels du logiciel, du matériel, et des composants compositionnels dans le système modélisé. Ceux-ci s'énoncent comme des configurations des composants et des connexions entre ces composants. Donc, l'ensemble de composants *threads* considérés *active*, c'est-à-dire le composant prêt à répondre aux expéditions, et les connexions qui sont disponibles pour transférer des données et des évènements entre ces composants représentent le mode courant du système. [BEN11]

Un changement de mode d'un composant AADL peut changer l'ensemble des composants actifs et connexions entre les composants. Dans ce cas, le changement de comportement est spécifié comme un diagramme d'états transitions, où les états représentent les modes et les transitions déclenchés par les événements.

Ces événements proviennent d'un port d'entrée d'événement *(event port)*. Ils peuvent avoir l'effet d'activation ou de désactivation des *threads* pour l'exécution, de changement du chemin de connexion entre les *threads*, et de changements des caractéristiques internes des composants. A un moment donné, un seul mode peut être actif et seuls les threads actifs peuvent exécuter leur code.

2.5.9 Discussion

À cause de la diversité des domaines d'application d'AADL, diverses méthodes de reconfigurations sont proposées au fil du temps. Nous discutons dans cette sous-section les travaux existants qui traitent la reconfiguration au niveau d'un composant AADL.

Selon [Zal08] le langage AADL s'appuie sur des composants de types *concrets*, ce qui permet d'utiliser ces composants pour configurer et analyser les systèmes TR^2E (Temps-Réel Réparties Embarqués) critiques, mais cela pose un vrai problème pour la réutilisation des composants AADL dans d'autres domaines d'application variés. Dans le cadre d'un projet de Thales, ce problème a été résolu en proposant des composants *génériques* de type CCM (CORBA Component Model) [EK02] pour la décomposition fonctionnelle qui sont des composants qui sont encapsulés par des composants AADL [Bor09]. Ces solutions ne rendent pas le processus de reconfiguration et d'adaptation d'un composant AADL dynamique et automatique sensible aux changements demandés par son environnement d'exécution. Notre but consiste à proposer un processus de reconfiguration automatique suivant des scénarios ayant pour résultat un ensemble de reconfigurations prêtes à être exécutées sans violé le fonctionnement du système qui génèrent automatiquement du code et la reconfiguration automatique des composants AADL en fonction des propriétés du modèle.

Etienne Borde dans sa thèse [Bor09] propose une approche qui s'appuie sur la notion de changement de mode de fonctionnement divisé en trois étapes afin d'améliorer la

productivité des systèmes TR^2E. La première étape consiste à analyser la spécification du système et la logique des mécanismes de changement de mode. La deuxième étape consiste à vérifier que les applications logicielles respectent un certain nombre de propriétés de sûreté de fonctionnement. La troisième étape consiste à automatiser la productivité de l'ensemble des applications logicielles du système avec des techniques de génération de code pour améliorer la qualité du code produit et respecter les règles de programmation des systèmes critiques. Il propose dans [BFHP09] une approche de conception qui repose sur les techniques de génération de code dans le but d'implémenter des systèmes adaptatifs et critiques. Cette approche consiste à représenter le comportement dynamique du système sous forme d'un langage d'architecture orienté composant COAL (Component-Oriented Architecture Language) par :

1) Énumérer les modes de fonctionnement du système,

2) Représenter les changements de mode du système en communication avec le mode automates,

3) Préciser lesquels des caractéristiques de l'architecture sont valides ou non dans un mode donné.

Puis ces informations seront interprétées afin de produire le code correspondant pour l'adaptation spécifique.

La reconfiguration selon [QWL14] est définie comme l'activité transitoire entre deux états /modes du système (Multimodes) par la transformation d'un composant AADL en Petri Net (PN). Cette reconfiguration est appliquée sur IDM (Integrated Modular Avionics). Une extension est proposée dans [SAZ11] par l'ajout d'un mécanisme de reconfiguration dynamique de multimodes du système par la transformation d'un modèle AADL vers Time Petri Net (TPN).

Dans ces travaux, la reconfiguration s'articule essentiellement autour des changements des modes prédéfinis avant la mise en marche du système. En cas d'apparition d'une panne ou d'un problème donné, le système navigue entre ces modes déjà prédéfinis. La question qui se pose : si le système couvre tous les modes et ne trouve pas un mode pertinent, dans ce cas le système peut subir des blocages.

Un autre inconvénient posé par l'approche de Etienne Borde concerne la reconfiguration locale du système sans prendre en compte les composants matériels car il se focalise essentiellement sur les composants logiciels. De plus, la reconfiguration locale au niveau des composants AADL est absente et ne traite pas le problème de la sécurité due par la reconfiguration.

La faiblesse des travaux de [Zal08] réside dans la reconfiguration statique des composants. De plus, ces composants étant dédiés à des applications spécifiques, ils ne peuvent pas assurer la réutilisation de tels composants pour d'autres applications.

Les différences majeures entre la logique de reconfiguration et la logique de changement de mode dans le contexte des composants AADL qui empêchent l'utilisation des solutions de modes décrits dans la sous-section 2.5.8 sont :

- Tous les modes d'un composant AADL sont chargés en mémoire, ce qui peut causer la fragmentation de l'espace mémoire avec un temps de recherche d'espace libre dans un bloc de mémoire fragmenté qui peut être difficilement prévisible,

- Il est difficile de garantir une initialisation correcte d'une mode,

- Les changements de modes ne traitent pas des changements imprévisibles de l'exécution d'un composant ou d'un algorithme. Ceci peut causer un problème lors du calcul du pire temps d'exécution (worst case exécution time, WCET) dans un système de contrôle industriel,

- Absence de la flexibilité de l'architecture qui supporte toutes les formes de reconfigurations possibles, et qui traite de la sécurité d'un composant reconfigurable,

- Absence d'une architecture flexible qui rend les interactions entre les composants reconfigurables souples qui requiert par conséquent moins de tests pour vérifier le bon fonctionnement du système.

Le nouveau composant AADL reconfigurable que nous proposons dans cette thèse, bien qu'il offre un processus dynamique de reconfiguration locale, assure la cohérence et l'interaction entre ses composants reconfigurables au niveau d'une architecture distribuée. Nous avons opté pour les reconfigurations synchrones, comme nous le verrons dans la suite. Ces besoins de capacités de reconfiguration sont très significatifs dans le contexte

TABLE 2.2 – Comparaison entre les solutions de reconfiguration d'AADL.

Solution	Méthode	Statique/ Dynamique	Flexibilité	Sécurité	Générique
Configuration [Zal08]	Intergiciel	Statique	Non	Non	Non (Concrets)
COAL [Bor09]	Changement de mode	Dynamique	Non	Sûreté de fonctionnement	Oui
AADL vers TPN [SAZ11]	Multimodes	Dynamique	Non	Non	Non
RA2DL	Reconfiguration	Dynamique/ automatique	Oui	Oui	Oui

d'un système de contrôle industriel, qui impose un accroissement constant du nombre de fonctionnalités que doivent remplir ce système tout en respectant des contraintes de performance et de sécurité de ces composants reconfigurables.

Le Tableau 2.2 fait une comparaison entre les méthodes existantes de reconfiguration d'AADL avec notre méthode *RA2DL*. Cette comparaison porte tout d'abord sur la méthode de reconfiguration. Nous nous limitons dans cette comparaison aux types de reconfiguration statique ou dynamique. La comparaison porte aussi sur l'étude de la flexibilité et la sécurité du composant AADL reconfigurable. Finalement, pour chaque méthode étudiée, nous précisons s'il s'agit d'une méthode générique ou non.

2.6 Méthodes de Reconfigurations Existantes

La reconfiguration d'un système embarqué assure la distinction entre les fonctions essentielles et non essentielles au niveau des énoncés dans le cahier des charges du système. Ainsi, les concepteurs et les programmeurs peuvent spécifier les propriétés qu'ils veulent.

Nous montrons dans cette section les deux types de reconfigurations existants dans la littérature pour reconfigurer un système temps-réel embarqué.

2.6.1 Définition

L'action de la reconfiguration définie par tous les mécanismes et scénarios de reconfigurations ayant lieu au moment où un système est en cours d'exécution, permet de changer

l'architecture, la structure et le comportement du système afin de s'adapter à son environnement d'exécution. Cette reconfiguration se produit conjointement avec des processus normaux du système [SK06].

D'autres travaux ont utilisé la reconfiguration pour garantir la fiabilité du système dans une variété d'utilisation, par exemple fournir un contrôle de fonctionnalités du système lors de sa dégradation. Shelton et Koopman dans [SK04] ont étudié l'identification et l'application de façon alternative des fonctionnalités utiles qu'un système pourrait fournir en cas de défaillances des composants matériels. Leur travail est focalisé sur les besoins de reconfiguration et peut donc être utilisé dans la conception des systèmes reconfigurables, mais ne garantit pas l'assurance du système.

Khalgui et Hanisch dans [KH09] ont proposé une approche pour développer un système de contrôle embarqué qui est reconfigurable et sécurisé à base de blocs fonctionnels (BF) selon le standard IEC61499. La reconfiguration s'articule sur une sémantique : c'est l'amélioration automatique de la performance du système lors de son exécution. Si un scénario de reconfiguration est appliqué au moment de l'exécution, le réseau de BF est totalement modifié.

Nous définissons la reconfiguration, comme une opération souvent utile permettant la modification des composants logiciels et matériels d'un système embarqué à base de composants d'une façon automatique, afin de les adapter dynamiquement au changement de son environnement d'exécution.

Dans la communauté de la reconfiguration des systèmes embarqués, il existe deux politiques de reconfigurations, la reconfiguration statique appliquée quand le sytème n'est pas mis en marche et la reconfiguration dynamique appliquée quand le système est en cours d'exécution.

2.6.2 Reconfiguration Statique

La reconfiguration statique est toute reconfiguration appliquée par l'utilisateur avant la mise en marche du système. Elle est moins utilisée de nos jours.

Les auteurs proposent dans [ASM05a] le concept des blocs fonctionnels réutilisables

pour implémenter une large gamme de systèmes embarqués où chaque bloc est statiquement reconfigurable sans aucune re-programmation. Ceci est accompli par la mise à jour de la structure des données du support quand le code exécutable ne change pas et n'est pas enregistré dans une mémoire permanente. Le tableau d'états de transitions se compose de plusieurs sorties sous forme de diagrammes de décision binaires qui représentent le prochain état des divers états et les actions de contrôles associées.

Une approche de reconfiguration statique appliquée sur FPGA [LEPI10], permet d'optimiser les erreurs dans le contexte des systèmes embarqués temps réel distribués, utilisée pour sécuriser des applications critiques par la mise en place des mécanismes de détection des erreurs avec deux algorithmes d'optimisation des erreurs matérielles et logicielles, afin de minimiser le coût d'ordonnancement.

Les auteurs dans [Val00] on étudié l'implémentation des circuits de commande qui sont construits à partir de modèles réutilisables avec un noyau re-programmable qui est décomposé en blocs. La fonctionnalité d'un circuit de commande se présente sous la forme d'une spécification comportementale d'une machine à états finis (FSM) afin de construire un FSM basé sur des blocs programmables avec une taille modeste.

Une reconfiguration statique dans [LT07] permet qu'aucun paquet ne reste pas dans le réseau pendant la reconfiguration du matériel. Ainsi, aucune action supplémentaire ne doit être fournie pour acheminer la sortie correcte après la reconfiguration. Avant d'effectuer la reconfiguration du matériel, le réseau doit être vide. Cela se fait par un mode de fonctionnement spécial.

2.6.3 Reconfiguration Dynamique

La reconfiguration dynamique est une action qui consiste à modifier le comportement du système pendant son exécution par la modification de ses composants logiciels ou matériels du système, avec l'introduction du concept de flexibilité et la dynamique du système. Il existe deux types de reconfiguration dynamique : reconfiguration manuelle et reconfiguration automatique.

- *Reconfiguration Manuelle :* repose sur l'intervention de l'utilisateur pour appliquer

des scénarios de reconfiguration. Les auteurs proposent dans [RSS+07] une méthodologie complète basée sur l'intervention humaine pour reconfigurer dynamiquement des systèmes de contrôle. Ils présentent une expérimentation intéressante montrant le changement dynamique d'un algorithme des blocs fonctionnels par les utilisateurs sans perturber l'ensemble du système. Les auteurs utilisent dans [TDF04] *Real-time-UML*, comme un méta-modèle entre les modèles de conception et leurs modèles d'implémentation pour soutenir la reconfiguration dynamique à base d'utilisateurs des systèmes de contrôle.

- *Reconfiguration Automatique :* C'est la reconfiguration la plus utilisée de nos jours. C'est une technique souvent utile pour modifier le comportement d'un tel système par l'intervention automatique des programmes ou des agents intelligents dès l'occurrence des défauts logiciel/matériel pour améliorer la performance du système. Les auteurs proposent dans [Bre02], une approche de reconfiguration à base d'agents pour sauver l'ensemble du système en cas de panne au moment de l'exécution. Les auteurs proposent dans [AV10] un agent à base d'ontologies pour effectuer des reconfigurations de système afin d'adapter ses comportements à l'évolution de son environnement d'exécution. Ils se sont intéressés à l'étude des reconfigurations des systèmes de contrôle dans le cas où des défauts matériels peuvent se produire au moment de l'exécution.

Khalgui et Hanisch proposent dans [KH08], une approche basée sur la reconfiguration dynamique par la proposition d'une architecture à base d'agents représentés par des machines d'états pour gérer toutes les reconfigurations possibles du système.

2.6.4 Reconfiguration Automatique des Systèmes Temps-Réels Embarqués

La reconfiguration signifie des changements qualitatifs des structures, des fonctionnalités et des algorithmes du système de contrôle comme une réponse aux changements qualitatifs du système physique contrôlé ou de l'environnement. Cela pourrait être causé par des échecs (partiels) [HMKC16], des pannes (répartitions) [LMKT16], ou même par des interventions humaines. Quand un problème matériel arrive dans le système physique contrôlé [GBKC16], un scénario de reconfiguration est automatiquement appliqué dans le système de contrôle qui doit réagir en changeant sa mise en œuvre d'une instance de

tâches à une deuxième. Dans cette section, nous allons nous intéresser à la reconfiguration mono/multi processeur(s).

2.6.4.1 Reconfiguration Mono-processeur

Plusieurs études ont été consacrées au développement des systèmes temps réel reconfigurables [KMLH11a] [GKA12]. L'auteur dans [WKK+15] met l'accent sur la reconfiguration temps réel des systèmes embarqués sous les contraintes d'énergie. L'auteur a proposé un agent en vue de réduire la consommation d'énergie après l'application d'un scénario de reconfiguration. La recherche dans [ASM05b] suggère les tâches réutilisables pour implémenter un large éventail de systèmes où chaque tâche est statiquement reconfigurée sans aucune re-programmation. Différentes solutions sont proposées pour les reconfigurations de WCETs, les délais et les périodes de tâches qui sont planifiés par FP ou EDF [GC10].

2.6.4.2 Reconfiguration Multi-Processeurs

Plusieurs travaux ont traité de la reconfiguraion au niveau des systèmes multi-processeurs [KMLH11b] [SE15]. Les auteurs dans [JSHP11] proposent un nouveau système de gestion de puissance dynamique pour les MPSoCs adaptés aux applications multimédias. Les auteurs dans [ASC+13] présentent un cadre global pour l'optimisation et l'estimation de la puissance/ l'énergie des systèmes multi-processeurs hétérogènes. Dans ce cadre, une méthodologie de modélisation de puissance est proposée et une plate-forme ouverte est développée. Cette méthodologie prend en compte tous les aspects des systèmes embarqués ; le logiciel, le matériel, et le système d'exploitation. Les auteurs dans [AEK06] décrivent une architecture NoC pour multi-processeurs reconfigurable sur puce afin de minimiser l'énergie.

2.6.5 Synthèse

La comparaison entre les méthodes de reconfigurations existantes dans le domaine de systèmes temps réel embarqués est donnée dans le Tableau 2.3. Cette comparaison est basée sur : la méthode utilisée, l'architecture utilisée pour la reconfiguration, la sécurité de la reconfiguration et l'étude des comportements.

TABLE 2.3 – Comparaison entre les méthodes de reconfigurations.

Familles des solutions	Méthode	Architecture	Sécurité	Comportement
Contrôle de fonctionnalités [SK04]	Alternative (Statique)	Matériels	Non	Non
Blocs fonctionnels (BF) [KH09]	Sémantique	Standard IEC61499	Oui	Oui
FPGA [LEPI10]	Optimisation des erreurs	Applications Critiques	Oui	Non
Modules Réutilisables [Val00]	Re-programmation	Circuit de commande	Non	Oui (FSM)
Real-time-UML [TDF04]	Dynamique manuelle	systèmes de contrôle.	Non	Non
Agents [AV10]	Ontologies	Matériels	Oui	Non
Mono-Processeur [WKK+15]	Sous contraintes	Système Embarqué	Non	Oui
Multi-processeurs [KMLH11b]	Adaptation Dynamique	MPSoCs	Non	Non

2.7 Limites des Solutions Existantes

Dans cette section, nous citons les limites des solutions existantes. Nous présentons deux problèmes cruciaux : (l) l'inadéquation de la plupart de ces solutions de reconfigurations des approches par composants dans le contexte des systèmes de contrôles industriels (SCI) et (2) les reconfigurations sont toujours effectues au niveau locale du composant et la distribution de reconfiguration au niveau d'une architecture distribuée n'est pas supporté dans les travaux cités auparavant. Nous proposons dans cette thèse, une méthodologie RA2DL représentée à l'aide d'un processus global complètement automatique pour reconfigurer dynamiquement un composant AADL localement puis reconfigurer la totalité du système de contrôle industriel à base des composants RA2DL selon des scénarios de reconfigurations et les changements de l'environnement de l'exécution.

2.7.1 Inadéquation avec les Systèmes de Contrôle Industriel

L'amélioration de la productivité pour les systèmes SCI demeure un objectif incontournable pour justifier la qualité des résultats. Toutefois, la prise en compte des besoins d'utilisateurs pour le rendre un système reconfigurable à travers ces composants est une tâche très importante. Par ailleurs, la reconfiguration de ses composants tels que *Darwin*,

$Meta-H$ ou *Rapid* est effectuée le plus souvent au moment de l'exécution selon des scénarios de reconfigurations, tels que l'allocation dynamique de la mémoire ou le remplacement d'un composant par un autre. Le comportement d'un composant reconfigurable est choisi dynamiquement en fonction des propriétés et changements de l'exécution de l'application. Cette méthode à l'avantage de rendre le code de l'application plus simple du point de vue du concepteur du composant reconfigurable et permet de construire rapidement les comportements du composant. Cependant cette approche présente deux inconvénients majeurs qui empêchent son utilisation dans le contexte des systèmes SCI :

1. Il est difficile de garantir la coordination correcte entre les composants reconfigurables comme $Wright$ ou $C2$ en terme de sécurité à cause des nombreuses failles de sûreté de fonctionnement pour certaines approches par composants.

2. L'utilisation de l'approche composant complique l'analyse du flux de reconfiguration entre les composants reconfigurables. Ainsi, la synchronisation devient de plus en plus complexe ce qui demande par conséquent un nombre supplémentaire de tests pour vérifier la bonne reconfiguration de la totalité du système.

2.7.2 Reconfiguration Locale du Composant

La plupart des approches existantes que nous avons décrites dans la section 2.4, à savoir les composants On-line et Off-line, plus particulièrement $AADL$ sont principalement utilisés pour reconfigurer le système afin d'adapter son comportement global aux changements de son environnement d'exécution sans tenue compte de particularité de la reconfiguration locale d'un composant à part . Notre objectif dans cette thèse est de prendre en compte toutes les formes de reconfiguration au niveau architectural, structurel (composition) et données d'un seul composant AADL à part et qu'il soit compatible au formalisme interne proposé. Toutefois il faut que ce formalisme possède la caractéristique et les changements des comportements interne du composant selon des reconfigurations différents .

2.8 Synthèse

La question cruciale à poser au niveau de cette synthèse est quel type de composants devrions-nous choisir pour développer des systèmes industriels flexibles et reconfigurables ? La réponse est un peu difficile, car chaque type des composants possède des avantages et des limites.

Comme nous pouvons le constater à travers cet état de l'art, le concept d'AADL comme langage fédérateur avec sa bibliothèque riche et son annexe comportementale regroupe des points importants qui permettent de décrire précisément toutes les descriptions d'une architecture d'un système SCI à la fois par des composants matériels et logiciels. AADL possède un haut niveau d'abstraction permettant la spécification formelle et la vérification des descriptions architecturales. Aussi AADL dispose à la fois d'une description graphique, et d'une description textuelle de ses composants. Le graphique a l'avantage de permettre une description facile pour une lecture de haut niveau du système. Le textuel est plus précis, et permet potentiellement de représenter plus d'informations sur les composants et le fonctionnement du système. AADL est un langage extensible à plusieurs variétés des domaines pour leur capacité à générer du code à partir de spécifications formellement vérifiables.

Aucun des langages et techniques présentés dans ce chapitre ne décrit un processus de reconfiguration dynamique et automatique intégrant à la fois la vérification, la flexibilité, la modélisation et la sécurisation d'un composant AADL reconfigurable qui soit dédié aux systèmes de contrôle industriel. Il existe certes des éléments dans certains des travaux décrits qui vont nous être d'une grande utilité pour mener à bien ce travail de thèse. Le changement de modes et l'annexe de gestion d'erreurs, par exemple, semblent être de très bons éléments pour créer un composant AADL reconfigurable et générer le reste des composants reconfigurables dédiés à un système de contrôle industriel. En effet, la définition des services canoniques de reconfiguration et l'organisation par couche de la reconfiguration offrent des moyens efficaces pour reconfigurer les différents composants flexibles afin de les optimiser.

Nous voulons dans cette thèse proposer de nouvelles solutions techniques pour les dé-

veloppements des systèmes de contrôle industriel reconfigurables à base de composants logiciels/matériels. Nous avons opté pour l'utilisation du langage AADL qui permet à la fois de modéliser précisément la partie logicielle et la matérielle des systèmes de contrôle industriel. Nous-nous sommes intéressés aux reconfigurations automatiques des composants AADL pour gérer automatiquement les modifications au moment de l'exécution.

2.9 Conclusion

Dans ce chapitre, nous avons présenté dans une première étape les concepts du système embarqué, système temps-réel et système de contrôle industriel. Dans la deuxième étape, nous avons exposé les méthodes de l'ingénierie dirigée par les modèles. Dans la troisième étape, nous avons proposé un étude bibliographique des composants on-line et off-line existants autour de l'approche par composant. Ainsi, nous avons défini et formalisé le concept de composant selon les modes d'applications les plus reconnus et couramment utilisés, nous avons décrit les différentes techniques qui visent à sécuriser les approches par composants. L'objectif est de prendre en compte les différentes approches académiques et industrielles utilisant les composants.

Dans une quatrième étape, nous avons exposé l'intérêt du langage de description et d'analyse d'architecture AADL. L'objectif de cette étude est de présenter les principes d'AADL décrivant les caractéristiques et les types de ces composants dans leurs structures internes et les interactions extérieures avec des ports pour assurer la composition d'une architecture à base de composants AADL. Nous avons exposé également les outils existants, l'annexe d'AADL, l'annexe de gestion des erreurs et les modes et les transitions de modes en AADL. A la fin de cette étapes nous avons discuté les travaux effectuées pour la reconfiguration des composant AADL pour positionner notre contribution.

Dans la cinquième étape, nous avons présenté les différentes méthodes de reconfigurations existantes de nos jours en les classant en deux grandes familles : reconfiguration statique et reconfiguration dynamique. Aussi nous avons présenté deux méthodes de reconfigirations automatiques des systèmes temps-réels embarqués : Mono-processeur et Multi-processeur. A la fin nous avons synthétisé ces méthodes

Enfin, nous avons montré les limites des solutions existantes, au niveau de l'inéquation des approches existantes avec la reconfiguration d'un système SCI, ainsi que le manque de reconfiguration locale au niveau des composants AADL.

AADL est le langage qui permet de développer une approche globale par composants pour mettre en place un système adaptatif de contrôle industriel flexible, reconfigurable dynamiquement et sécurisé. En effet, AADL offre un ensemble de catégories de composants, réparties en trois grandes familles : logiciels, matériels et systèmes. Pour tous ces raisons AADL représente un cadre utile à la reconfiguration du système de contrôle industriel.

Troisième partie

Contributions

Chapitre 3

Première Contribution : RA2DL : Nouveau Composant AADL Reconfigurable

3.1 Introduction

Comme nous l'avons évoqué dans l'état de l'art, la technologie AADL de l'approche par composant a été proposée essentiellement pour analyser et décrire les éléments architecturaux d'un système embarqué par un ensemble des composants. Dans ce chapitre, nous introduisons dans un premier temps la démarche à suivre afin de transformer un composant AADL classique en un composant reconfigurable dynamiquement. Dans un second temps, nous étudions les différentes formes de reconfigurations possibles appliquées à un composant AADL en utilisant des scénarios de reconfiguration. Dans un troisième temps, nous présentons une nouvelle architecture constituée de plusieurs modules pour gérer toutes les formes de reconfigurations. La modélisation de ce nouveau composant est représenté par une automates qui modélise le passage d'un état du composant à un autre selon les niveaux hiérarchiques de reconfigurations en tenant compte la formalisation de chaque niveau et ses comportements. L'étude de la flexibilité et comment tester si le composant est flexible ou non sera une autre approche à traiter au niveau de cette contribution.

Notre objectif dans ce chapitre est de mettre en place un composant intitulé $RA2DL$ qui est un composant à la base AADL mais reconfigurable et flexible. La solution proposée par ce dernier doit garantir un fonctionnement adaptatif pour faciliter l'exécution d'un

composant RA2DL avec la multi-utilisation au niveau d'un système de contrôle industriel. En effet, la mise en place d'un composant RA2DL reconfigurable dépend des éléments suivants :

- Des formes de reconfigurations architecturales, structurelles (ou de composition) et de données,

- De la nouvelle architecture avec un contrôleur RA2DL dédiée aux reconfigurations,

- De la formalisation et la modélisation de ses comportements selon des niveaux hiérarchiques pour mieux maitriser la complexité de la modélisation.

Par la suite, nous présentons dans ce chapitre les différentes formes de reconfigurations qui peuvent effectués par un composant RA2DL. Nous proposons l'architecture du composant RA2DL avec les trois modules de contrôles dans la troisième section. Une formalisation de RA2DL sera exposée dans la quatrième section. Nous nous intéressons dans la section cinq à la modélisation de RA2DL. Au niveau de la section six , nous exposons une étude sur la flexibilité d'un composant RA2DL. Avant de finir le chapitre, nous présentons dans la section sept un simple exemple de reconfiguration d'un composant AADL et le transformé en un composant RA2DL pour mieux retenue les contributions de ce chapitre.

3.2 Formes de Reconfigurations

La reconfiguration dynamique d'un composant AADL nécessite des formes de reconfigurations qui doivent respecter les changements de l'environnement d'exécution. De ce fait, AADL permet de définir trois formes possibles de reconfigurations, chose permettant ainsi d'atteindre des niveaux de performance très élevés. Le principal intérêt d'un composant AADL reconfigurable est de pouvoir changer à tout moment son architecture, sa structure interne et ses données.

3.2.1 Reconfiguration d'Architecture

Le composant AADL possède sa propre architecture proposée par SAE, cependant la reconfiguration dynamique envisageable est une solution architecturale qui permet de reconfigurer l'architecture du composant AADL afin de présenter des solutions architectu-

rales qui répondent au changement de son environnement d'exécution.

Selon la norme des fonctionnements d'un composant AADL, les principales reconfigurations architecturales sont :

- Ajout d'un algorithme, activation d'un port d'entrée-sortie ou d'un port de données/événements,

- Suppression d'un algorithme, désactivation d'un port d'entrée-sortie ou d'un port de données/événements non utilisées ou non importantes selon leurs priorités.

Nous appelons la reconfiguration architecturale toute reconfiguration qui permet de changer l'architecture d'un composant AADL en une autre architecture.

3.2.2 Reconfiguration de Structure (Composition)

La composition interne d'un composant AADL est représentée par une implantation qui définit la structure interne du composant en termes d'interactions entre les sous-composants, les algorithmes et les ports. La plupart des composants peuvent contenir des compostions différentes pour former l'architecture d'une application basée sur des composants AADL.

La reconfiguration de structure d'un composant AADL permet de changer la structure interne d'un composant au niveau de ses sous-composants, ses algorithmes et leurs connexions pour enrichir les descriptions AADL avec les caractéristiques non architecturales.

Dans un environnement d'exécution instable, la reconfiguration de structure devient une des clés de réponse garantissant une réactivité et une adaptation aux contraintes externes. Les principales reconfigurations de structure d'un composant AADL sont :

- La modification de l'ordre d'exécution des algorithmes ou des sous-composants selon une architecture donnée,

- L'altération des entrées-sorties de données/d'évènements selon une priorité.

3.2.3 Reconfiguration de Données

Le standard AADL permet d'exécuter des données internes. La validité de ces données est définie d'une part par la catégorie du composant et d'autre part par les données déjà définies par les algorithmes internes qui peuvent être une donnée dans la source texte, composée ou non. Une donnée (*Data*) peut contenir des programmes et offrir des accès à ceux-ci. Elle peut également contenir d'autres données et obtenir un type composé (comme une structure en langage C). La procédure de reconfiguration de données d'un composant AADL se concentre essentiellement sur :

- La modification des données en entrées provenant des ports d'entrées selon un scénario de reconfiguration,

- Le changement des valeurs des variables d'un algorithme selon les reconfigurations demandées.

3.3 Architecture de RA2DL

Le standard AADL ne définit pas une méthode de reconfiguration de ses composants selon les formes de reconfigurations cités auparavant. Pour ceci, nous instaurons un nouveau formalisme d'un composant nommé **RA2DL**, il s'agit d'une extension d'un composant AADL qui est reconfigurable. Dans ce cadre, une nouvelle architecture flexible sera proposée et mise en place pour répondre à l'ensemble des formes de reconfiguration de RA2DL. Cette architecture est essentiellement décrite par deux modules, le premier est un **module de contrôle** former par un ensemble de fonctions de reconfigurations appliquées sur RA2DL. Le deuxième est un **module contrôlé** qui désigne l'ensemble des algorithmes et événements/données d'entrées-sorties.

Figure.3.1 illustre l'architecture du module contrôlé de RA2DL qui est répartie en quatre modules à démêler ci-après.

3.3.1 Module d'Entrée des Événements (IEM)

Le module (IEM) (Input Event Module) est conçu spécifiquement pour manipuler les reconfigurations des événements d'entrées (IE) à travers les ports d'entrées des événe-

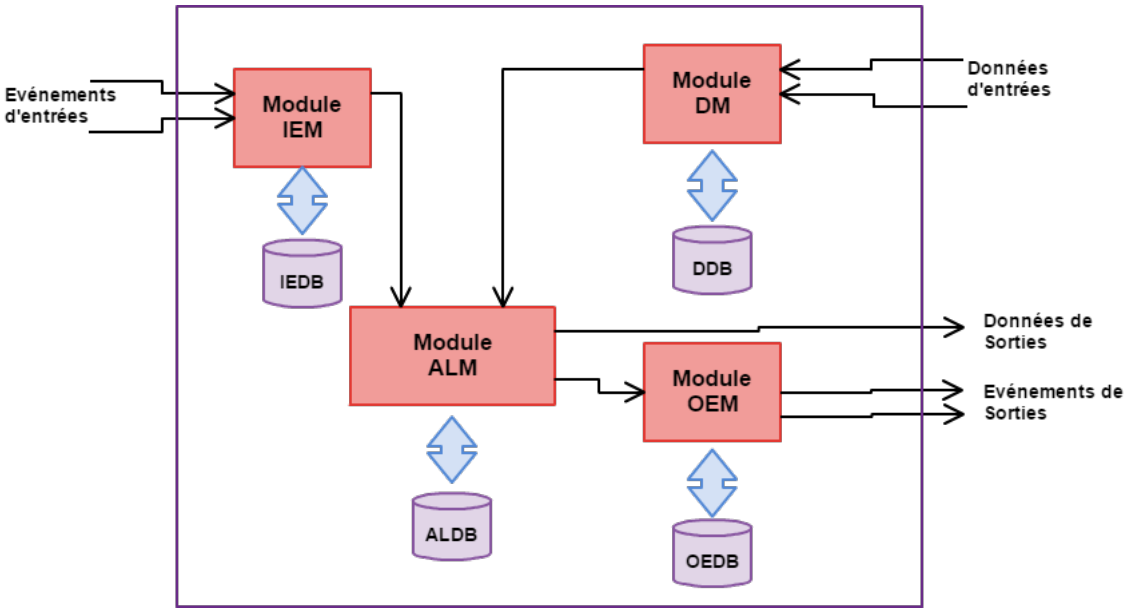

FIGURE 3.1 – Architecture du composant RA2DL.

ments, qui sont mémorisées dans une petite base de données ($IEDB$). IEM est défini et activé à un moment donné suite à un scénario de reconfiguration qui touche les événements d'entrées (IE) d'un sous-ensemble d'événements pour exécuter les algorithmes correspondants d'un composant RA2DL.

3.3.2 Module des Données (DM)

Le module (DM) (Data Module) est un module dédié particulièrement pour gérer les reconfigurations des données (*data*) dans un composant RA2DL aux niveaux des portes d'entrées des données ou les dionées des algorithmes internes, en cohérence avec le reste des modules de RA2DL. DM possède une petite base de données (DDB) pour enregistrer en temps-réel les données exécutées ou reconfigurées.

3.3.3 Module des Algorithmes (ALM)

Le module (ALM) (Algorithms Module), permet de distinguer entre les scénarios de reconfigurations des algorithmes d'un composant RA2DL en relation avec le module IEM et en cohérence avec l'ajout/suppression d'un tel algorithme suite un évènement en entrée. Les reconfigurations possibles au niveau de ALM peuvent être l'activation/désactivation

d'un algorithme existant ou l'ajout et la suppression d'un algorithme. ALM contient une base de données ($ALDB$) qui permet d'enregistrer les algorithmes.

3.3.4 Module de Sortie des Événements (OEM)

Le module OEM (Output Event Module), concerne la reconfiguration des événements de sortie (OE) et les ports de sortie des événements. Ils sont mémorisés dans une petite base de données ($OEDB$). OEM est défini et activé à un moment donné suite à un scénario de reconfiguration d'un sous-ensemble d'événements après l'exécution des algorithmes correspondants d'un composant RA2DL.

Notons enfin que pour assurer une interaction correcte entre les modules d'un composant RA2DL et pour couvrir toutes les formes de reconfigurations possibles, les événements de reconfigurations définissent un modèle d'exécution pour appliquer les reconfigurations requises selon les besoins des utilisateurs et aussi selon le changement et l'évolution de l'environnement d'exécution.

3.4 Formalisation de RA2DL

Nous visons, dans cette section, à proposer un nouveau formalisme du composant RA2DL, dont l'objectif d'adapter son comportement lors de son exécution à son environnement selon les besoins définis par les utilisateurs. La reconfiguration de RA2DL se fait selon trois niveaux hiérarchiques :

- Niveau Architecture (AL) **:** Nous définissons en AL toutes les architectures possibles qui peuvent être implémentées par le composant RA2DL au moment de l'exécution. Une architecture dans AL est un ensemble d'algorithmes qui effectuent des activités de contrôle. Un scénario de reconfiguration peut changer l'architecture du composant par l'ajout et aussi la suppression d'un algorithme. Pour chaque architecture AL, nous devons définir un modèle d'exécution pour l'algorithme correspondant.

- Niveau Composition (CL) **:** Pour chaque architecture dans AL nous associons une composition CL des algorithmes correspondants, qui représente l'ordre d'exécution

des algorithmes d'un composant RA2DL.

- Niveau Données (DL) **:** Pour chaque composition d'une architecture d'algorithmes au niveau AL, nous définissions les valeurs de données possibles des variables des algorithmes qui correspondent à un DL.

Grâce à cette formalisation hiérarchique selon trois niveaux d'un composant RA2DL, il peut manipuler tous les scénarios de reconfigurations possibles d'un composant RA2DL.

Un composant RA2DL est formalisé comme suit :

$$RA2DL = (\beta, R) \text{ ou :}$$

β : représente le module contrôlé de RA2DL, et **R** : décrit le module de contrôle défini selon une structure hiérarchique comme suit :

3.4.1 Niveau Architectural (AL)

Selon les changements imprévisibles de l'architecture d'un composant RA2DL et suite à la satisfaction d'une condition particulière, il est possible d'ajouter, supprimer ou changer les comportements internes du composant RA2DL au niveau des quatre modules IEM, OEM, ALM et DM. Nous notons par Ψ_{RA2DL} l'ensemble de tous les algorithmes possibles impliqués dans les différentes implémentations d'un composant. Soit ξ_{RA2DL} qui représente le sous-ensemble des algorithmes impliqués dans une implémentation particulière à un temps donné t dont $\xi_{RA2DL} \subseteq \Psi_{RA2DL}$.

Nous modélisons le niveau architectural AL par une machine à états finis (Figure 3.2)notés par ASM de l'ensemble des états S_{AL} tel que chaque état de $\mathbf{S}_{AL}$ correspond à une implémentation particulière des modules IEM, OEM, ALM et DM.

$$\mathbf{S}_{AL} = (\Psi_{RA2DL}, \mathbf{O}, \delta) \text{ , Où :}$$

- **O** est un ensemble de n états dans $\mathbf{S}_{AL}$ (**O**=$\{ \mathbf{S}^i_{AL} \mid i \in [1..n] \}$),

- δ est une fonction de transition de : Ψ_{RA2DL} x **O** $\rightarrow$ Ψ_{RA2DL} x **O**.

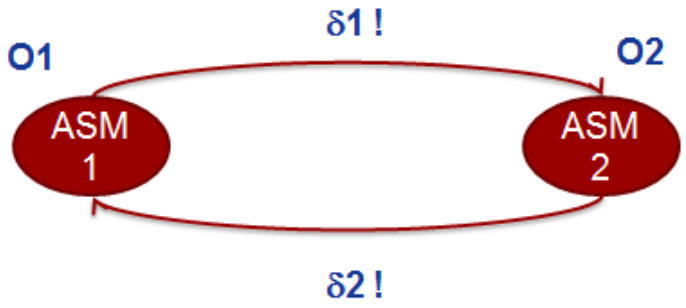

FIGURE 3.2 – Niveau Architecture.

3.4.2 Niveau Composition (CL)

Le niveau composition ou structure (Composition Level (CL)) conserve la même architecture d'un composant RA2DL mais change seulement la composition ou l'ordre d'exécution de ses algorithmes, les événements d'entrées-sorties afin d'adapter le composant à son environnement d'exécution. CL est formalisé par une machine à états finis CSM (Figure 3.3), tel que chaque CSM correspond à un état particulier dans un niveau architectural $\mathbf{S}_{AL}$, pour chaque état $\mathbf{S}^{i}_{AL}$ de $\mathbf{S}_{AL}$. Nous définissons dans le second niveau hiérarchique CL un état particulier noté par $\mathbf{S}^{j}_{CL}$. Pour chaque état $\mathbf{S}^{j}_{CL}$ de $\mathbf{S}_{CL}$, nous fixons une composition particulière d'un sous-ensemble d'algorithmes et des événements d'entrées-sorties. Cette composition étudie a priori chaque algorithme du composant afin d'obtenir un modèle d'exécution déterministe pour chaque composant RA2DL. Nous notons par $\Gamma(\delta_{RA2DL})$ tous les ensembles possibles des modèles d'exécutions des algorithmes δ_{RA2DL} à un niveau de composition (CL).

$$\mathbf{S}_{CL} = (\Gamma(\delta_{RA2DL}), \mathbf{P}, \gamma) \text{ , où :}$$

- $\mathbf{P}$ est un ensemble de m états dans $\mathbf{S}_{CL}$(P= { $\mathbf{S}^{j}_{CL} \mid j \in [1..m]$ }),

- γ est une fonction de transition de : $\Gamma(\delta_{RA2DL})$ x $\mathbf{P} \rightarrow \Gamma(\delta_{RA2DL})$ x$\mathbf{P}$.

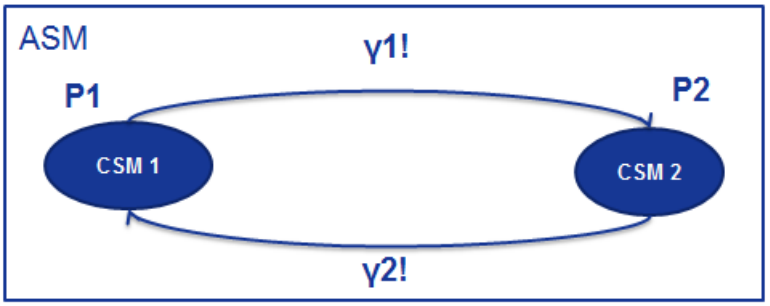

FIGURE 3.3 – Niveau Composition.

3.4.3 Niveau de Données (DL)

Au niveau des données (Data Level (DL) d'un composant RA2DL, un scénario de reconfiguration $R^{i,j}_{CL}$ au niveau composition CL est une transition de l'état S^{i}_{CL} à un autre état S^{j}_{CL} de S_{CL}. La reconfiguration d'un composant RA2DL qui touche le troisième niveau DL correspond à la mise à jour des données du composant. Nous définissons pour chaque état S^{i}_{AL} de S_{AL} et pour chaque état S^{j}_{CL} de S_{CL} une nouvelle machine à état S_{DL}, où chaque état correspond à des nouvelles valeurs affectées aux données d'une telle composition d'une architecture de RA2DL.

Le niveau DL est dédié à la reconfiguration des données du composant RA2DL. Il est formalisé par une machine à états finis de données DSM (Figure 3.4) où chaque état correspond à des valeurs particulières de données. Nous définissons pour chaque état $\mathbf{S}^{i}_{AL}$ de $\mathbf{S}_{AL}$ et pour chaque état $\mathbf{S}^{j}_{CL}$ de $\mathbf{S}_{CL}$ une nouvelle machine à état $\mathbf{S}^{k}_{DL}$ où chaque état correspond à des nouvelles valeurs de données.

$$\mathbf{S}_{DL} = (\Gamma(\mu_{RA2DL}), \mathbf{Q}, \vartheta) \text{ , où :}$$

- $\mathbf{Q}$ est un ensemble k d'état de composition de $\mathbf{S}_{DL}(\mathbf{Q}=\{ \mathbf{S}^{k}_{DL} \mid k \in [1..n] \})$,

- ϑ est une fonction de transition de : $\Gamma(\mu_{RA2DL})$ x $\mathbf{Q} \rightarrow \Gamma(\mu_{RA2DL})$ x$\mathbf{Q}$.

3.4.4 Comportement de RA2DL

Pour analyser le module de contrôle β d'un composant RA2DL, nous caractérisons l'exécution des algorithmes correspondants par un pire cas de temps d'exécution WCET

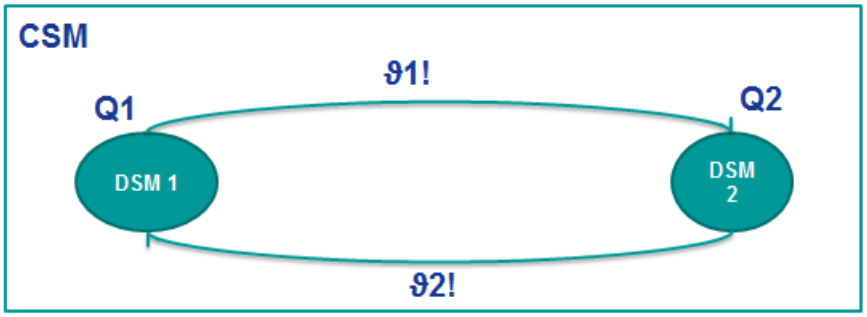

FIGURE 3.4 – Niveau Données.

(Worst Case Execution Times) ou respectivement par un meilleur cas de temps d'exécution BCET (Best Case Execution Times). De plus, nous considérons que les événements de sortie peuvent être envoyés simultanément ou même l'exclusion en fonction des besoins des utilisateurs. Afin de valider le comportement temporel d'un composant RA2DL, nous nous intéressons uniquement aux scénarios de reconfiguration en entrée. Nous supposons une synchronisation complète entre les événements et les données. En effet, lorsqu'un événement se produit à l'entrée, toutes les données associées se produisent en même temps au niveau des entrées correspondantes. Les différents scénarios de reconfigurations appliqués par les différents contrôleurs définissent tous les comportements possibles dans le module contrôle β. Dans la Figure 6.9, nous désignons les comportements d'un composant RA2DL par des machines à états finis notés par BSM où chaque état correspond à un comportement particulier après chaque scénario de reconfiguration.

Un exemple qui montre le changement de comportements du composant RA2DL selon des scénarios de reconfigurations au niveau d'un système Radar illustré dans la Figure 6.9 du chapitre, montre que chaque branche représente un comportement.

3.5 Modélisation de RA2DL

Nous proposons dans cette section la modélisation des différents niveaux d'un composant RA2DL où nous utilisons le modèle checker UPPAAL [Mer01] pour modéliser les états de chaque niveau. Nous modélisons le module de contrôle de RA2DL tel que le niveau architectural représenté par ASM dans lequel chaque état correspond à une architecture

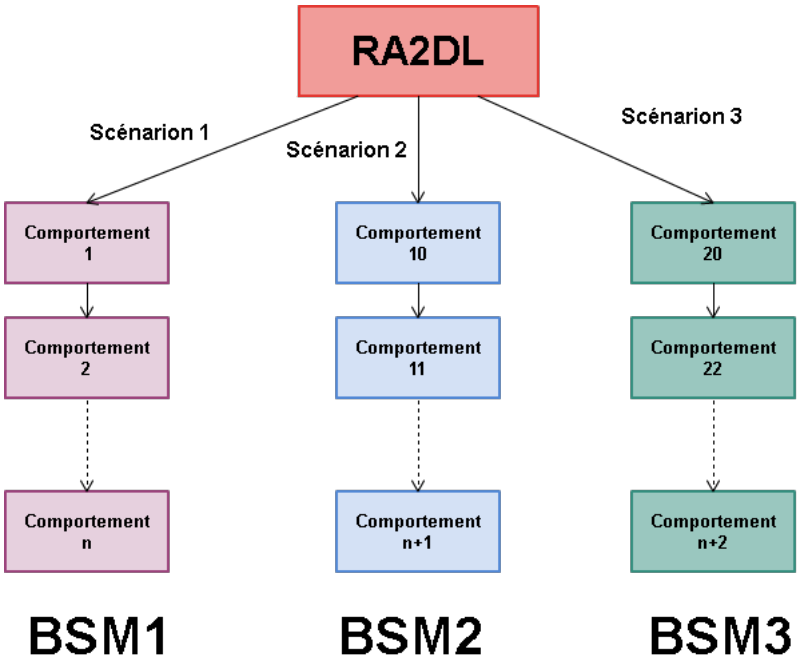

FIGURE 3.5 – Comportement de RA2DL.

particulière du composant RA2DL qui exécute une reconfiguration d'architecture RA. Par conséquent, chaque transition de ASM correspond à un ajout ou suppression des algorithmes et des événements d'entrées-sorties. Selon une condition de passage $CondA$ un état de ASM correspond à une machine à états particulier du niveau composition notée CSM. Cette machine à états dédiée à la reconfiguration de composition RC, spécifie toutes les formes de composition d'algorithmes et des événements d'entrées-sorties pour être activé dans cet état du premier niveau d'architecture. Dès que la condition $CondC$ est validée, un état de la composition correspond à une machine à états dans le niveau données notée par DSM est activée pour reconfigurer les données RD, elle spécifiée toutes les valeurs possibles de données dans le composant RA2DL après la vérification de la condition $CondD$.

Figure 3.6 représente la modélisation et les liens entre les divers niveaux du composant RA2DL. Cette modélisation est décrite par un automate où chaque état représente un niveau de reconfiguration d'un composant RA2DL et les liens montrent les conditions de passage d'un état à un autre :

- *Start* : : l'état de début l'application de la reconfiguration,

- *ASM* : l'état qui représente la reconfiguration de l'architecture par l'application de l'action *ApplyRA*! ,

- *CSM* : l'état qui représente la reconfiguration au niveau composition ou structure par l'application de l'action *ApplyRC*! ,

- *DSM* : l'état qui représente la reconfiguration au niveau donnée par l'application de l'action *ApplyDC*! ,

- $X == CondA$: représente la condition de passage d'un scénario de reconfiguration au niveau architectural selon la transition CR,

- $X == CondC$: représente la condition de passage d'un scénario de reconfiguration d'un niveau architectural au niveau composition selon la transition CR,

- $X == CondD$: représente la condition de passage d'un scénario de reconfiguration d'un niveau architectural ou composition au niveau donnée selon la transition DR.

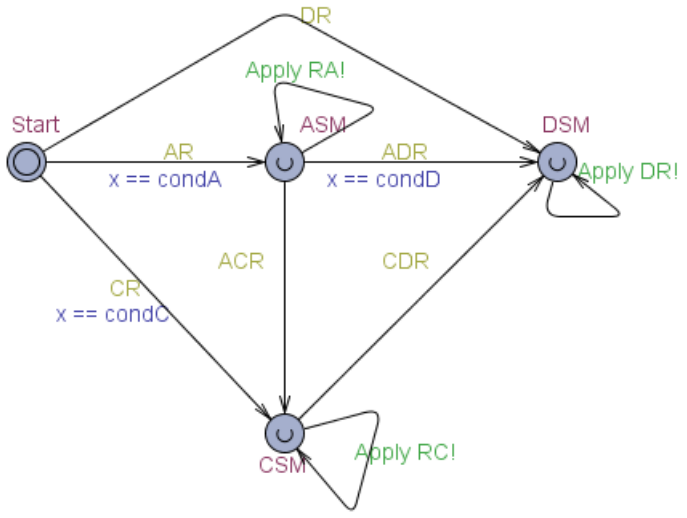

FIGURE 3.6 – Modélisation de RA2DL.

3.6 Flexibilité de RA2DL

Après avoir étudié les formes de reconfigurations possibles appliquées au composant RA2DL, nous passons à l'étude et au test de flexibilité, qui est un paramètre important à calculer théoriquement pour savoir si le composant RA2DL accepte et répond à tous les scénarios de reconfigurations possibles ou non ? Ce facteur est comparé à un seuil $flex_R$ donné par un expert où $0 \leq \text{flex}_R \leq 1$.

Dans ce cas, nous proposons une approche intitulée $Flex - RA2DL$, qui calcule la flexibilité d'un composant RA2DL selon la reconfiguration demandée. Les types de reconfigurations cruciales qui peuvent être rencontrées par un composant RA2DL : ajout, modifier ou supprimer des algorithmes, des ports ou même un autre composant RA2DL. Il y a aussi des reconfigurations moins cruciales telles que le changement de connexion entre les composants. Le calcul de flexibilité d'un composant RA2DL est une technique pour calculer les réponses certaines aux scénarios de reconfigurations selon l'importance de la reconfiguration demandée, afin de produire le coût de la reconfiguration totale d'un composant RA2DL.

L'équation de la flexibilité de RA2DL noté par $Flex(RA2DL)$ est calculée en fonction de tous les scénarios de reconfigurations reçues par RA2DL :

$$Flex(RA2DL) = \sum_{i=0}^{n} \frac{R_{exe}^{i}}{R^{i}} \leq flex_R. \text{ où :}$$

- R_{exe}^{i} : Représente le scénario de reconfiguration i accepté parmi toutes les scénarios à exécuter par le composant RA2DL,

- R^{i} : Désigne la totalité des scénarios de reconfigurations du composant RA2DL au niveau architecture, composition ou donnée, avec ($R^{i} \in \{AL, CL, DL\}$).

L'algorithme 1 appelé $AlgFlex$ est proposé dans l'ordre de calculer la flexibilité de chaque composant RA2DL. Tout d'abord, il lit les paramètres d'entrée de la première reconfiguration R_1. Ensuite, il calcule la flexibilité de R^{i} (i= 1) selon la fonction $Flex(RA2DL)$ s'il accepte ($Flex(RA2DL) \leq flex_R$) alors il passe à l'étape de l'exécution, Si non, il rejette ($Flex(RA2DL) > flex_R$) et passe à la reconfiguration suivante. Les mêmes calculs seront appliqués pour tous les scénarios de reconfigurations.

Algorithm 1 Algorithme de flexibilité de RA2DL ($AlgFlex$).

Require: $Flex(RA2DL)$: Flexibility de RA2DL.
  INPUT : R^i
Ensure: $Newreconfiguration(R^i)$
  for each ($R^i \in \{AL, CL, DL\}$) **do**
    if RA2DL $\longleftarrow R^i$ **then**
    $R^i \leftarrow R^i_{exe}$
    else
    $R^i \leftarrow R^{i+1}$
    end if

    Calculate $Flex(RA2DL)$
    Read $flex_R$
    for each $Flex(RA2DL)$ **do**
    if ($Flex(RA2DL) <= flex_R$) **then**
    $Execution$
    else
    R^{i+1}
    end if
    end for
    OUTPUT : $Flex(RA2DL)$
    =0

3.7 Exemple

Pour mieux comprendre les contributions de ce chapitre, nous faisons recours a un simple exemple d'un composant AADL intitulé *processor* (Figure 3.7) qui est un composant matériel qui exécute trois algorithmes *(Alg1, Alg2 et Alg3)*. Il comporte deux ports d'entrées d'événements *(IEvn1 et IEvn2)*, un port d'entrée de données *(IDn1)* et deux ports de sorties d'événements *(OEvn1 et OEvn2)*.

Le Tableau 3.1 résume les différents scénarios de reconfigurations demandées avec la fonction de chaque scénario et le type de reconfiguration (architectural, composition ou donnée).

Après l'application des scénarios de reconfigurations demandées avec le composant *processor* qui devient un composant reconfigurable avec un *Contrôleur*, la Figure 3.8 illustre la nouvelle architecture du composant $R - processor$.

Pour calculer le facteur de flexibilité du composant $R - processor$, nous supposons que

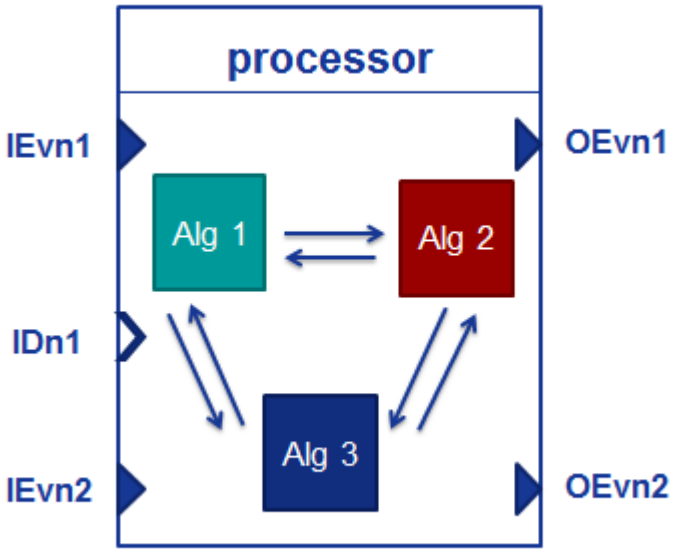

FIGURE 3.7 – Composant processor.

l'expert a attribué pour le composent un seuil $flex_R = 0.5$ et que le composant n'exécute pas les deux reconfigurations *(IRF2 et IRF5)*. Dans ce cas, l'équation $Flex(RA2DL)$ calculée en fonction de l'algorithme $AlgFlex$ est égale à :

$$Flex(RA2DL) = 3/5 = 0.6 > 0.5$$

Selon la fonction $Flex(RA2DL)$ le composant $R-processor$ est un composant flexible qu'on peut implémenter.

TABLE 3.1 – Scénarios de reconfigurations.

Scénario de reconfiguration	Fonction	Type
IRF 1	Ajout d'un algorithme Alg4	Architecture
IRF 2	Activation d'un port IEvn 3	Achitecture
IRF 3	Supprime l'algorithme Alg1	Architecture
IRF 4	Change l'ordre de l'exécution de Alg2 et Alg3	Structure
IRF 5	Change le valeur de variable du port de donnée IDn1	Donneé

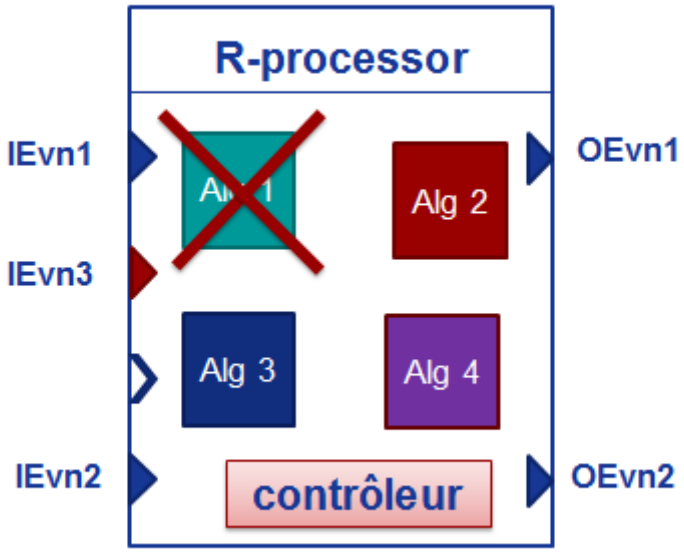

FIGURE 3.8 – Composant R-processor.

3.8 Conclusion

Dans ce chapitre, nous avons proposé une approche pour valider la reconfiguration d'un composant AADL, spécifiée comme un nouveau composant intitulé RA2DL qui est composé par deux modules : module contrôle et module contrôlé.

Dans la norme de RA2DL, une nouvelle architecture bien dédiée est proposée répartie en quatre modules : un module d'entrée des événements (IEM), un module de données (DM), un module d'algorithmes (ALM) et un module de sortie d'événements (OEM).

Nous avons aussi défini une formalisation des principales formes de reconfigurations de RA2DL. Une formalisation nous a permis de définir les reconfigurations possibles structurées en trois niveaux hiérarchiques : niveau architectural (AL) pour reconfigurer l'architecture de RA2DL, niveau composition réservé pour la reconfiguration de la structure interne de RA2DL et le niveau de données consacré à la reconfiguration des données de RA2DL.

Ensuite, nous avons mis en œuvre la modélisation de RA2DL. Une modélisation vise à présenter les trois niveaux de RA2DL par des machines à états qui montrent l'interaction

entre les divers niveaux en respectant les conditions de reconfigurations dans chaque niveau.

Afin de montrer la validité de notre composant et d'évaluer l'exécution des reconfigurations, nous avons proposé une mesure de flexibilité pour chaque composant RA2DL qui est un paramètre à calculer à chaque reconfiguration.

Nous avons utilisé une étude de cas simple représentée par un composant AADL *processor*, comme exemple pour appliquer nos contributions et produire un composant RA2DL $R - processor$.

Le chapitre suivant explique la manière dont les composants RA2DL sont améliorés au niveau de l'exécution pour participer à la reconfiguration d'une architecture distribuée d'un système de contrôle industriel. Nous y ajoutons le modèle d'exécution qui permet de corriger l'exécution des composants RA2DL afin de produire des composants applicatifs dédiés à l'application d'une architecture distribuée. Ces composants sont paramétrés avec ceux d'un composant de coordination intégrant une matrice de coordination pour gérer la synchronisation et la cohérence de reconfiguration distribuée et produire une application prête à être exécutée.

Chapitre 4

Contribution 2 : Nouveau concept : Réseau des Composants RA2DL

4.1 Introduction

Dans le chapitre précédent, nous avons présenté l'approche détaillée du composant RA2DL avec les formes de reconfiguration possibles et une architecture bien dédiée à la reconfiguration. Nous nous intéressons dans ce chapitre, à l'amélioration de la qualité d'exécution de ce dernier dans le but d'avoir un composant correctement exécutable, mais aussi le rôle du composant RA2DL dans l'architecture distribuée. Nous avons vu que certains reconfigurations de l'architecture distribuée sont faiblement personnalisables en fonction des propriétés de l'architecture distribuée. Ces reconfigurations ne devraient pas être générés automatiquement. D'autres reconfigurations sont fortement synchrones. Elles sont souvent en relation avec les environnements de fonctionnement du système. Les environnements sont, quant à eux, produits d'une manière automatique à partir des caractéristiques de l'architecture. Nous proposons également une nouvelle approche qui consiste à gérer la communication et la cohérence de reconfiguration entre les composants RA2DL au niveau d'une architecture distribuée qui s'adaptent aux différents changements des besoins tels que la stabilité, la performance, l'occupation physique, etc. Nous supposons que les éléments en entrée dans ce chapitre sont :

* Un modèle d'exécution pour corriger et améliorer l'exécution d'un composant RA2DL.

Ce modèle est fragmenté en trois niveaux : un middleware de reconfiguration pour gérer les scénarios (ou flux) de reconfigurations, un contrôleur d'exécution pour exécuter les reconfigurations souhaitées et un middleware de synchronisation dédiée pour les reconfigurations synchrones.

* Un support d'exécution distribué pour une architecture distribuée reconfigurable. Nous considérons une matrice de coordination qui sera attribuée à chaque composant de coordination, nommé $RA2DL - Coordinator$ qui connectent les différents composants RA2DL considérés afin de gérer les coordinations et les reconfigurations entre eux.

Dans la section suivante de ce chapitre, nous présentons c'est quoi un réseau de RA2DL ? Nous définissons dans la section d'après, le modèle d'exécution d'un composant RA2DL avec ses trois couches. Ensuite, nous exposons le modèle d'exécution pour les architectures RA2DL distribuées. Puis, nous modélisons le réseau et le modèle d'exécution d'un RA2DL. Enfin nous présentons un simple exemple de distribution de trois composant RA2DL pour expliquer notre contribution.

4.2 Réseau de RA2DL

Dans cette section, nous inventons un nouveau concept nommé **Réseau RA2DL** dans le quel nous précisons la manière dont notre composant RA2DL supporte la coordination et la communication dans une architecture distribuée : Il s'agit d'un ensemble des composants RA2DL reliés entre eux pour échanger des données, des événements et des reconfigurations. Une caractéristique essentielle des réseaux de RA2DL est l'implication de différents composants RA2DL qui entrent en jeu pour fournir des services ou des reconfigurations aux autres composants sur le même réseau :

- Les composants RA2DL fournissent des ressources aux composants prestataires,

- Les composants prestataires de reconfigurations s'appuient sur une ou plusieurs plateformes pour fournir une reconfiguration aux composants finaux,

- Les composants RA2DL demandent des reconfigurations temps-réel aux autres composants par l'envoi des événements de reconfigurations.

En termes d'enjeux, bien qu'offrant une série d'avantages pour l'utilisation d'un réseau

de composants RA2DL, un composant RA2DL comporte cependant des risques importants dans la mesure où ces reconfigurations sont désormais détenues par d'autres composants. La production et la diffusion d'un scénario de reconfiguration sont également faits par le composant cible. Pourtant, les reconfigurations se déplacent vers des composants centralisés avec une énorme puissance de traitement et de stockage de données. Cela comporte des implications importantes pour les composants du réseau RA2DL qui perdent non seulement le contrôle sur leurs propres reconfigurations, mais aussi la possibilité de déterminer la façon dont celles-ci seront utilisées par les composants eux-mêmes ou par d'autres composants.

Les réseaux RA2DL rassemblent et relient entre eux les composants RA2DL connectés en réseau de façon à créer une infrastructure reconfigurable, dynamique et performante. Les composants participant au réseau mettent une partie de leurs ressources à la disposition des autres composants connectés au réseau qui vont gérer le partage et la redistribution de ces ressources. Étant donné que ces composants sont tous égaux entre eux, ils participent tous de la même manière au réseau : tous ont une responsabilité égale dans la reconfiguration et l'adaptation du réseau aux changements de leur environnement d'exécution sans aucune distinction.

Dans le but d'offrir une bonne performance à ce réseau, nous regroupons l'ensemble des contraintes portant sur la distribution de RA2DL. Ces contraintes viennent soit du support l'exécution de RA2DL, soit des caractéristiques du réseau de RA2DL. Il est nécessaire de respecter ces contraintes pour le bon fonctionnement d'un système à base de réseaux de composants RA2DL. Du point de vue temps d'exécution et gestion de reconfigurations, ces contraintes permettent de restreindre le nombre de cas possibles des reconfigurations inutiles.

4.3 Modèle d'Exécution d'un RA2DL

Selon les fonctionnalités du composant RA2DL citées dans le chapitre précédent, RA2DL présente quelques lacunes et certaines faiblesses au niveau de son exécution qui demeure non résolue, comme la gestion des flux de différentes reconfigurations provenant

de l'extérieur, la synchronisation de reconfiguration entre les composants, la coordination et la distribution dans un réseau RA2DL. Dans cette section, nous enrichissons RA2DL par un modèle d'exécution qui corrige l'exécution lors d'un échec au niveau de l'exécution de RA2DL qui provoque un effet inattendu pendant son exécution et assure une meilleure répartition entre les composants RA2DL synchrones au niveau d'une architecture distribuée. La Figure 4.1 présente le modèle d'exécution composé de trois couches : un middleware de reconfiguration, un contrôleur d'exécution et un middleware de synchronisation.

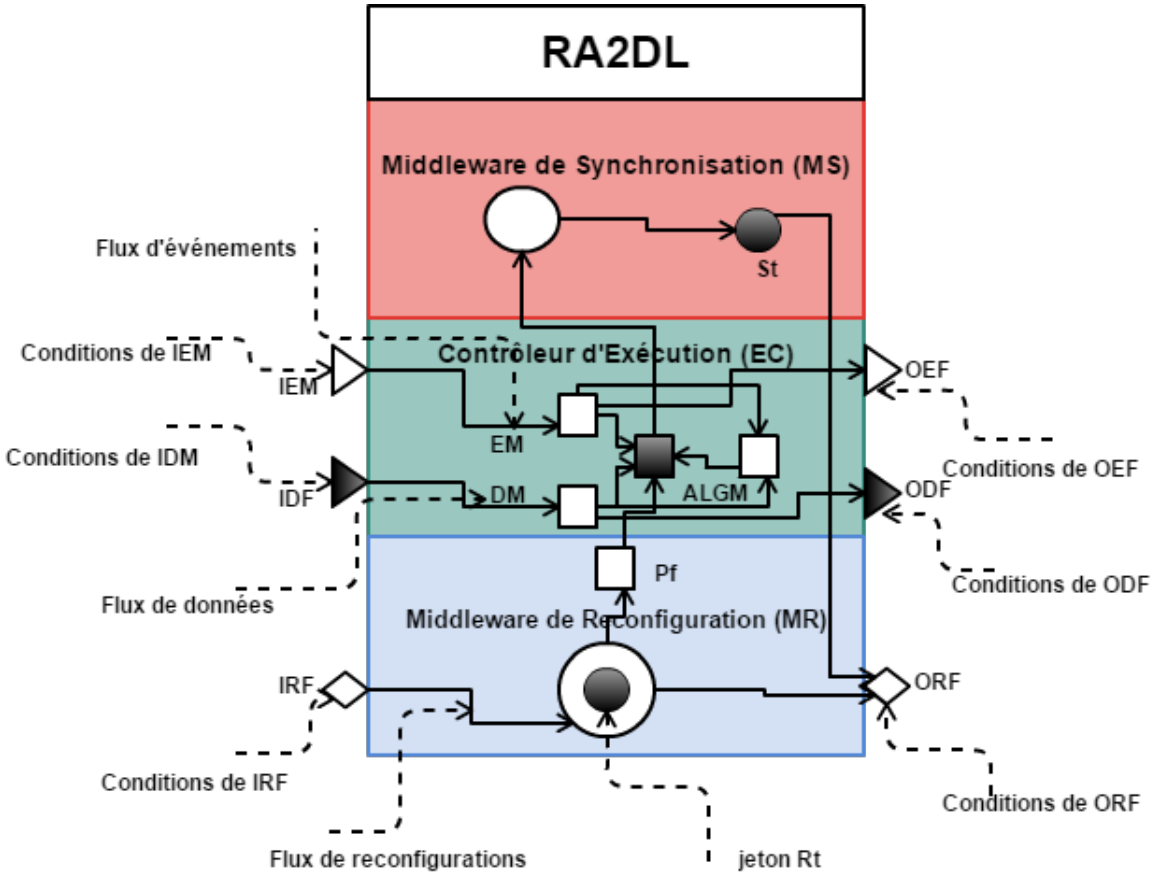

FIGURE 4.1 – Modèle d'exécution de RA2DL.

4.3.1 Couche 1 : Middleware de Reconfiguration

Le middeleware de reconfiguration (Middleware Reconguration Layer) MR est la première couche à contacter pour l'utilisation d'un composant RA2DL selon un jeton de reconfiguration RT. En premier lieu, MR est dédié essentiellement à la réception des flux

de reconfigurations (RF) provenant des ports d'entrées (IRF), où chaque RF porte un jeton RT contenant les informations nécessaires comme l'adresse DA du composant RA2DL cible pour effectuer la reconfiguration, une variable binaire V (V= 1 si la reconfiguration est synchrone avec un autre composant, V= 0 si la reconfiguration n'est pas synchrone) et une valeur pour le facteur de priorité PF donné par l'utilisateur pour ordonner les flux RF selon le PF le plus grand. En second lieu, ce middleware joue le rôle d'un manager de RA2DL ; il décide si le RF est associé à son composant ou non : s'il accepte la reconfiguration, alors elle sera envoyée aux couches suivantes. Dans le cas ou il n'est pas associé au composant courant, il passe alors au composant successeur par le port de sortie ORF. Dans le cas où MR reçoit plusieurs reconfigurations contradictoires et concurrentes, il décide quelle reconfiguration sera acceptée selon le facteur de priorité PF.

4.3.2 Couche 2 : Contrôleur d'Exécution

Le contrôleur d'exécution (Execution Controller) EC est la couche responsable de l'application de la reconfiguration acceptée par le middleware MR. EC possède deux ports d'entrées-sorties dont le premier est réservé aux flux de données et le second aux flux d'événements. EC contient aussi les algorithmes Alg de RA2DL.

Le contrôleur d'exécution est supposé être codé selon trois niveaux hiérarchiques, (a) niveau architectural (noté AL), (b) niveau composition (noté CL) et niveau données (noté DL). Nous définissons au niveau AL toutes les architectures possibles qui peuvent implémenter RA2DL dynamiquement. Chaque architecture de RA2DL est représentée par un ensemble d'algorithmes qui contrôlent les activités de RA2DL. Un scénario de reconfiguration peut changer l'architecture du composant par l'ajout d'un nouvel algorithme ou par la suppression d'un autre existant. Pour chaque architecture de AL, nous avons besoin de définir un modèle d'exécution pour les algorithmes correspondants. Une composition, définie au niveau CL, affecte une priorité à chaque algorithme dans chaque architecture et pour chaque composition de l'algorithme correspondant. Nous précisons également au niveau des données DL toutes les reconfigurations possibles qui peuvent reconfigurer les valeurs correspondantes aux données manipulées au moment de l'exécution.

4.3.3 Couche 3 : Middleware de Synchronisation

La reconfiguration de chaque composant RA2DL au niveau d'un réseau de composants est indépendante de tous les autres composants et la sortie générée par un composant RA2DL peut être l'entrée d'un autre réseau. Dans la technologie RA2DL, les composants doivent être exécutés de façon asynchrone. Cependant, étant donné que les composants RA2DL sont indépendants à chaque exécution, les états finaux dans chaque système à base des composants sont les mêmes dans le cas où les deux reconfigurations asynchrones ou synchrones.

Le middleware de synchronisation possède un jeton de synchronisation ST. Si la reconfiguration est synchrone avec un autre composant RA2DL, alors le jeton ST est envoyé avec l'adresse du composant cible pour l'impliquer dans la reconfiguration et mettre en attente l'état du sémaphore S comme le montre le pseudo-code ci-après pour les deux composants (RA2DL 1 et RA2DL 2). Si la reconfiguration est asynchrone, alors cette couche n'est pas concernée.

```
Semaphore (RA2DL 1, RA2DL 2)
RA2DL 1(s:)
RA2DL 2(s):
n(s) := n(s) - 1;
if n(s)<0 then;
State(RA2DL 1) := blocked;
State(RA2DL 2) := blocked;
enter (RA2DL 1, f(s))
enter (RA2DL 2, f(s))
```

4.4 Modèle d'Exécution pour les Architectures RA2DL Distribuées

Nous présentons ici une méthode pour appliquer ce que nous avons proposé auparavant avec le composant RA2DL sur une architecture distribuée à base de composants RA2DL. Afin de simplifier la compilation des reconfigurations des composants, cette méthode se base sur un modèle d'exécution pour les architectures RA2DL distribuées qui est centré sur un composant RA2DL de coordination qui gère la communication entre les composants utilisant une matrice de coordination.

4.4.1 Définition

Les architectures distribuées sont devenues le paradigme dominant pour toutes les applications embarquées. Pour ces raisons, nous avons orienté notre approche vers les applications distribuée à base des composants RA2DL. Cette distribution est assurée par un modèle d'exécution. Le modèle d'exécution d'une architecture RA2DL est basé sur des interactions entre des composants reconfigurables distribués, ce qui est une contrainte importante lorsqu'on essaye de faire coopérer des composants reconfigurables hétérogènes. Le principe de ce modèle est de représenter comme une interface pour tous les composants afin de gérer les transmissions des requêtes, ordonner les flux des reconfigurations, assurer les connexions et mémoriser l'état de chaque composant. Ce modèle est basé sur un composant RA2DL de coordination nommé $RA2DL-Coordinator$ (CR). Néanmoins, la coordination entre les modèles d'exécution dans cette architecture distribuée est extrêmement obligatoire parce que toute reconfiguration automatique incontrôlée appliquée avec un RA2DL peut conduire à des problèmes critiques. Pour garantir la reconfiguration distribuée en toute sécurité, nous définissons le concept d'une matrice de coordination (CM) (Coordination Matrix) qui définit des scénarios de reconfiguration correctes. La création d'un composant $RA2DL-Coordinator$ contenant un CM, traitant plusieurs instructions en même temps au sein du même composant, résout ce dilemme pour les architectures distribuées. Le modèle d'exécution qui en résulte supporte alors des interactions asynchrones basées sur la gestion des scénarios de reconfiguration.

4.4.2 Architecture RA2DL Distribuée

Soit Sys un système distribué à base des composants RA2DL composé de n composants $(RA2DL_1, ..., RA2DL_n)$. Soit m le nombre de composants de coordination qui assurent la communication entre ces n composants. On note dans ce qui suit par $Reconfiguration^{a}_{ia,ja,ka}$ l'ensemble des scénarios de reconfigurations appliqué par $RA2DL_n$ (a∈[1..n]) où :

* a : correspond au composant a qui applique la reconfiguration,

* ia : correspond à la reconfiguration au niveau architecture,

* ja : correspond à la reconfiguration au niveau composition,

* ka : correspond à la reconfiguration au niveau donnée.

Soient :

- ASM : La machine à états qui modélise tous les scénarios possibles au niveau architecture,

- ASM_{ia} : représente un état de la machine à état ASM,

- $cond^a_{ia}$: représente l'ensemble des conditions pour atteindre l'état ASM_{ia},

- CSM : La machine à états qui modélise tous les scénarios possibles au niveau composition,

- CSM_{ja} : représente un état de la machine à état CSM,

- $cond^a_{ja}$: représente l'ensemble des conditions pour atteindre l'état CSM_{ja},

- DSM : La machine à états qui modélise tous les scénarios possibles au niveau donnée,

- DSM_{ka} : représente un état de la machine à état DSM,

- $cond^a_{ka}$: représente l'ensemble des conditions pour atteindre l'état DSM_{ka},

Pour gérer les reconfigurations distribuées cohérentes qui garantissent des comportements sécurisés de l'ensemble du système Sys, nous définissons la notion de matrice de coordination notée par CM *(Coordination Matrix)* de taille (n, 3) avec n : le nombre des composants et 3 représente les niveaux hiérarchiques d'un tel composant RA2DL (architecture, composition et donnée). La Figure 4.2 définit les scénarios de reconfigurations cohérents à appliquer simultanément par les différents composants RA2DL. CM est une matrice où chaque ligne $a(a \in [1..n])$ correspond à un scénario de reconfiguration $Reconfiguration^a_{ia,ja,ka}$ appliqué par un composant $RA2DL_a$ et la colonne représente le niveau hiérarchique

$$CM[a,1] = ia; CM[a,2] = ja; CM[a,3] = ka$$

$\Rightarrow$ Si $ia = 1$ (idem pour ja et ka). Alors le composant correspondant applique la reconfiguration. Sinon (ia=0). Alors, le composant n'applique pas la reconfiguration.

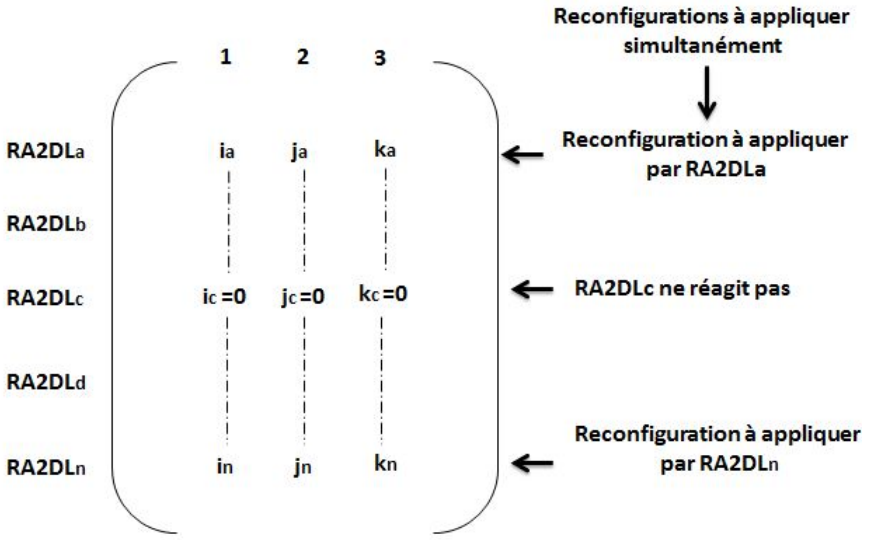

FIGURE 4.2 – Matrice de coordination.

4.4.3 Coordination Entre les Composants RA2DL Distribués

Pour coordonner et communiquer entre des composants RA2DL qui sont distribués, nous mettons en place une architecture pour les systèmes de contrôle suivant le standard RA2DL, pour gérer les reconfigurations automatiques réparties entre des composants. Afin de garantir un comportement cohérent de l'ensemble du système distribué, nous définissons $RA2DL-coordinator$ (noté par $CR(n')$) où n' représente le sous ensemble de composant ($n' \subseteq n$) qui manipule la matrice de coordination CM de n' pour contrôler les restes des composants de l'architecture (RA2DL^a $\in$[1..n]) tel que le scénario suivant : quand un composant RA2DL^a (a$\in$[1..n]) veut appliquer un scénario $Reconfiguration^a_{ia,ja,ka}$ (dans des conditions bien définies), alors il envoie la requête suivante à CR(n') pour l'obtention d'une autorisation.

$$request(RA2DL^a, CR(n'), Reconfiguration^a_{ia,ja,ka})$$

Quand CR(n')) reçoit la requête qui correspond à une matrice de coordination particulière $CM \in n'$ et si CM a la plus haute priorité entre toutes les matrices de n', alors $CR(n')$ informe le composant RA2DL cible de réagir simultanément avec le composant CR. L'information envoyée pour le composant $CR(n')$ est comme suit :

Pour chaque $RA2DL^b$, $b \in [1..n] \setminus \{a\}$ and $CM[b..i] \neq 0 \; \forall \; i \in [1..3]$:

$reconfiguration\ ((CR\Omega(Sys)), RA2DL_b, Reconfiguration^b_{(CM[b,1],CM[b,2],CM[b,3])})$

La Figure 4.3 montre un exemple de communication dans une architecture distribuée entre deux composants RA2DL 1, et RA2DL 2, dont la coordination est assurée par le composant CR avec la matrice CM.

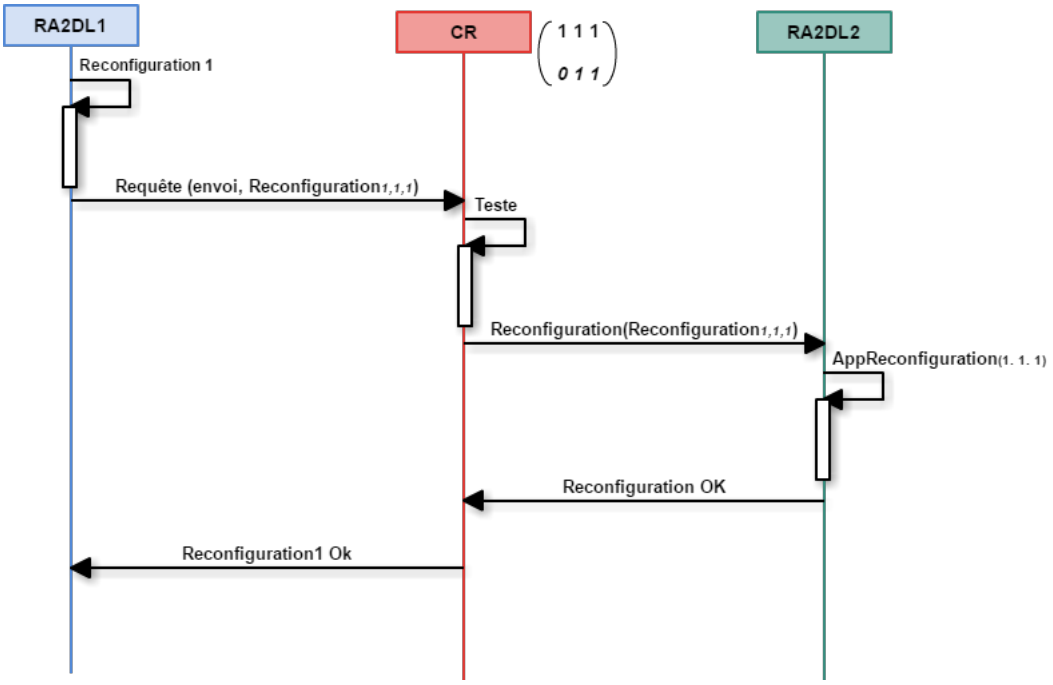

FIGURE 4.3 – Coordination entre $RA2DL_1$ et $RA2DL_2$.

4.5 Modélisation d'un Réseau de RA2DL

Dans cette section, nous expliquons comment est modélisé le modèle d'exécution d'un composant RA2DL. Nous expliquons aussi comment un composant RA2DL navigue entre les divers états de ce modèle.

4.5.1 Modélisation du Modèle d'Exécution

La modélisation du modèle d'exécution de RA2DL en trois couches est représentée par la machine d'états de la Figure 4.4.

La description des états du modèle d'exécution est ci-après : l'état *start* qui représente

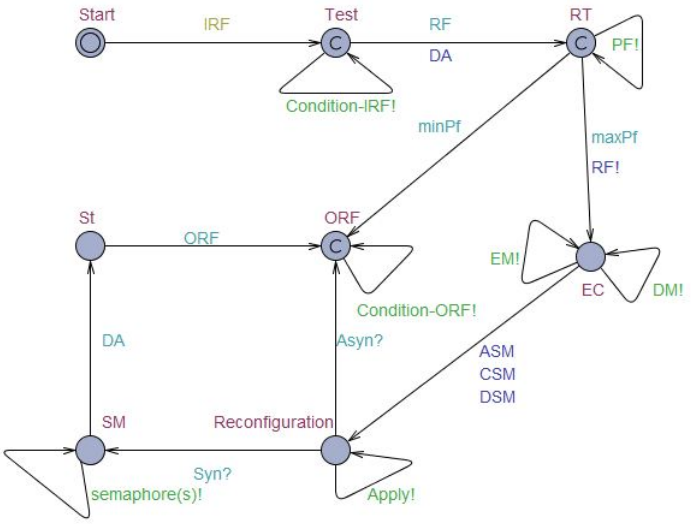

FIGURE 4.4 – Modélisation d'un modèle d'exécution de RA2DL.

le déclencheur pour commencer l'application de reconfigurations avec RA2DL. L'état *test* pour contrôler si la condition d'acceptation d'entrée du flux de reconfguration notée par $condition - IRF$ est vérifiée ou non. Si la condition est bien vérifiée, le composant accepte l'entrée du flux de reconfiguration. L'état RT pour tester la priorité de reconfiguration selon le facteur de priorité noté par PF et les informations portées par le jeton de reconfiguration RT. L'état noté EC correspond à la couche contrôleur du modèle d'exécution de RA2DL avec EM pour les événements et DM pour les données. L'état $Reconfiguration$ permet d'appliquer la requête de la reconfiguration demandée, à ce niveau un autre test qui aura lieu pour voir si la reconfiguration demandée est synchrone ou asynchrone. L'état SM représente l'état du middleware de synchronisation. Si la reconfiguration est synchrone avec un autre composant, alors le sémaphore S se met en attente au niveau de l'état *waiting*. L'état St correspond à l'envoie de la reconfiguration à un autre composant. L'état ORF correspond à l'état final du modèle d'exécution.

4.5.2 Modélisation de la Coordination Entre les Composants RA2DL

Nous avons exposé précédemment le modèle d'exécution de RA2DL. La Figure 4.5 décrit la modélisation de la coordination entre les composants RA2DL. Pour une meilleure explication de la communication, nous illustrons cette modélisation par un exemple de

trois composants RA2DL : $RA2DL - server$ pour l'envoi, $RA2DL - Receiver$ pour la réception et le composant de coordination $RA2DL - coordinator$.

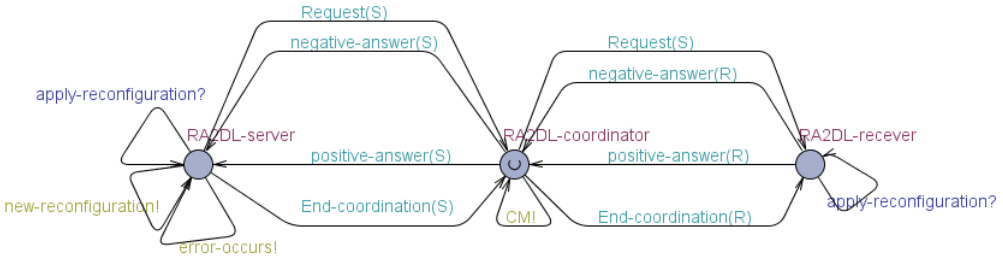

FIGURE 4.5 – Modélisation de la coordination entre RA2DL.

$RA2DL - server$ envoie une requête de reconfiguration au composant $RA2DL - coordinator$ qui teste selon la matrice de coordination CM si elle convient au composant $RA2DL - Receiver$ ou non.

4.6 Exemple

Nous faisons recours à un simple exemple pour expliquer les approches proposées dans ce chapitre, nous supposons une application distribuée composée de trois composants RA2DL communicants (RA2DL 1, RA2DL 2 et RA2DL3) comme dans la Figure 4.6.

Le processus d'exécution et de reconfiguration dans cette architecture est décrit comme suit :

— Le composant RA2DL 1 joue le rôle de récepteur d'un ensemble de scénarios des reconfigurations (IRF1 , IRF2 et IRF3) où chaque scénario porte un jeton portant les informations dans le Tableau 4.1,

TABLE 4.1 – Information du jeton.

Scénario de reconfiguration	Composant concerné	Synchrone/ Asynchrone	Facteur de priorité (PF)
IRF 1	RA2DL 2	Asynchrone	2
IRF 2	RA2DL 3	Architecture	1
IRF 3	RA2DL 1	Synchrone	3

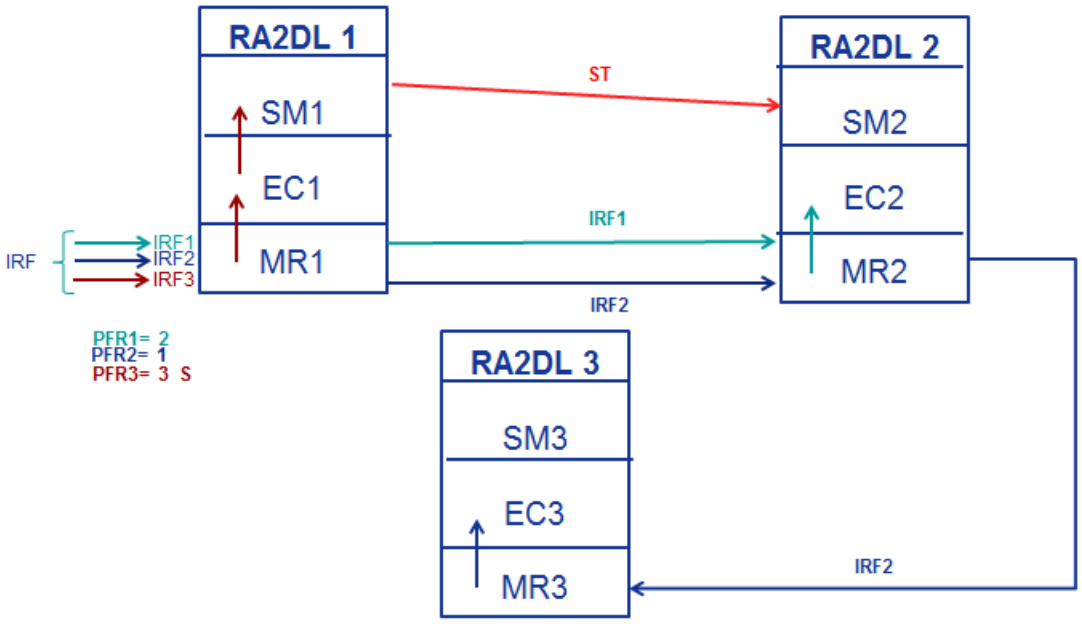

FIGURE 4.6 – Exemple d'application distribuée.

— Le middleware de reconfiguration du composant RA2DL 1 qui accepte le scénario $IRF3$ (le plus prioritaire) pour l'exécuter au niveau de la couche contrôleur d'exécution. Ainsi, l'exécution de cette reconfiguration est synchrone avec le composant RA2DL 2,
— Le scénario $IRF3$ passe par le middleware de synchronisation,
— Un jeton de synchronisation (ST) envoyé au composant RA2DL 2,
— Le composant RA2DL 2 se met en attente au niveau sémaphore S jusqu'a la fin de la reconfiguration,

```
State(RA2DL 2) := blocked;
```

— Le scénario $IFR1$ sera exécuté au niveau du composant RA2DL 2,
— Le scénario $IFR3$ sera exécuté au niveau du composant RA2DL 3,
— Le composant CR gère la coordination et la reconfiguration entre les trois composants selon la matrice de coordination.

4.7 Conclusion

Dans ce chapitre, nous avons présenté les différentes étapes d'un processus d'exécution d'un composant RA2DL à partir de son modèle d'exécution. Il s'agit d'un modèle de RA2DL écrit selon les restrictions d'exécution que nous avons spécifiées et il est réparti en trois couches, la première permet de gérer les flux des reconfigurations, la deuxième pour contrôler l'exécution et la troisième est réservée pour les reconfigurations synchrones. Nous avons présenté les différentes analyses qu'il est possible d'effectuer sur ce modèle pour garantir le fonctionnement correct d'un composant RA2DL. La deuxième contribution de ce chapitre porte sur la proposition d'un modèle d'exécution pour les architectures RA2DL distribuées. Dans cette proposition, nous avons exposé les différents scénarios de communications entre les composants RA2DL avec un composant de coordination et une matrice de coordination. Nous avons présenté ensuite une modélisation du modèle en fonction des états de composant RA2DL. A la fin de ce chapitre nous avons montré un simple exemple d'une architecture distribuée composée de trois composants RA2DL.

Le chapitre suivant explique la manière dont les composants RA2DL restent en mode sécurisé au niveau d'une architecture distribuée. Nous y présentons notre méthode $RA2DL-Pool$ qui permet de sécuriser les composants RA2DL afin de produire une architecture sécurisée dédiée à l'application SCI. Ces composants sont paramétrés avec des mécanismes de sécurité pour produire une application prête à être exécutée.

Chapitre 5

Contribution 3 : RA2DL-Pool : Nouvelle Approche pour la Sécurisation du Composant RA2DL

5.1 Introduction

Dans les chapitres précédents, nous avons défini un processus dynamique et automatique pour reconfigurer un composant RA2DL à partir de la description d'architecture d'un composant AADL. Nous avons aussi défini l'architecture d'un réseau de composants RA2DL fondée sur un modèle d'exécution. Ce modèle couplé au processus de distribution, permet de générer automatiquement une coordination de reconfiguration dédiée aux besoins spécifiques de chaque architecture distribuée.

L'objectif de ce chapitre est de proposer une approche pour sécuriser un composant RA2DL dans une architecture distribuée ouverte pour les attaques malveillantes qui n'étaient pas possibles avant. Ces situations soulèvent de nouveaux défis sur la façon de gérer la sécurité afin de concevoir une architecture à base de composants RA2DL qui est plus résistante aux attaques et moins vulnérable.

Face aux nouveaux défis de sécurité pour les systèmes reconfigurables à base de composants RA2DL, nous proposons une méthode alliant une nouvelle architecture sécurisée avec l'application des mécanismes de sécurité dédiées.

Une autre évolution pour sécuriser les composants RA2DL repose sur un nouveau concept intitulé $RA2DL - Pool$ qui représente un conteneur qui regroupe les composants RA2DL selon un facteur de similarité entre eux. La sécurité d'une application à base de composants RA2DL revient alors à décider non seulement la sécurité d'un composant RA2DL mais aussi la sécurisation des containers $RA2DL - Pool$.

Dans la seconde partie de ce chapitre, nous considérons que les composants RA2DL d'un système de contrôle industriel sont déjà regroupés dans des $RA2DL - Pool$. Nous proposons deux mécanismes de sécurité avérés pour accéder à chaque $RA2DL - Pool$, le premier est le mécanisme d'authentification et le deuxième est le contrôle d'accès.

5.2 RA2DL-Pool : Nouvelle Extension pour la Sécurité du Composant RA2DL

Dans cette section, nous donnons une vue globale du conteneur $RA2DL - Pool$. D'abord, nous détaillons les motivations qui nous ont poussé à créer un $RA2DL-Pool$. Ensuite, nous décrivons les éléments de son architecture. Enfin, nous exposons l'applicabilité de $RA2DL - Pool$ au niveau d'une architecture distribuée.

5.2.1 Définition de RA2DL-Pool

Pour consolider la sécurité d'un composant RA2DL contre les attaques extérieures et les utilisations malveillantes, nous recourons à une nouvelle approche de sécurité. En effet, l'utilisation d'une application à base de RA2DL repose sur la communication entre plusieurs composants ce qui influe sur le niveau de sécurité garanti par l'application. De ce fait, nous proposons une méthode qui assure le groupement des composants RA2DL selon un facteur de similarité tel que les composants qui ont : les mêmes tâches, reconfiguration, relation...etc. Nous allons les rassembler dans un conteneur nommé $RA2DL-Pool$ (Figure 5.1).

Il s'agit d'un caisson conçu pour le transport des marchandises, $RA2DL-Pool$ est une classe abstraite destinée à grouper les composants RA2DL pour faciliter la manipulation et la sécurité de toute utilisation ou exploitation des composants RA2DL et qui offre

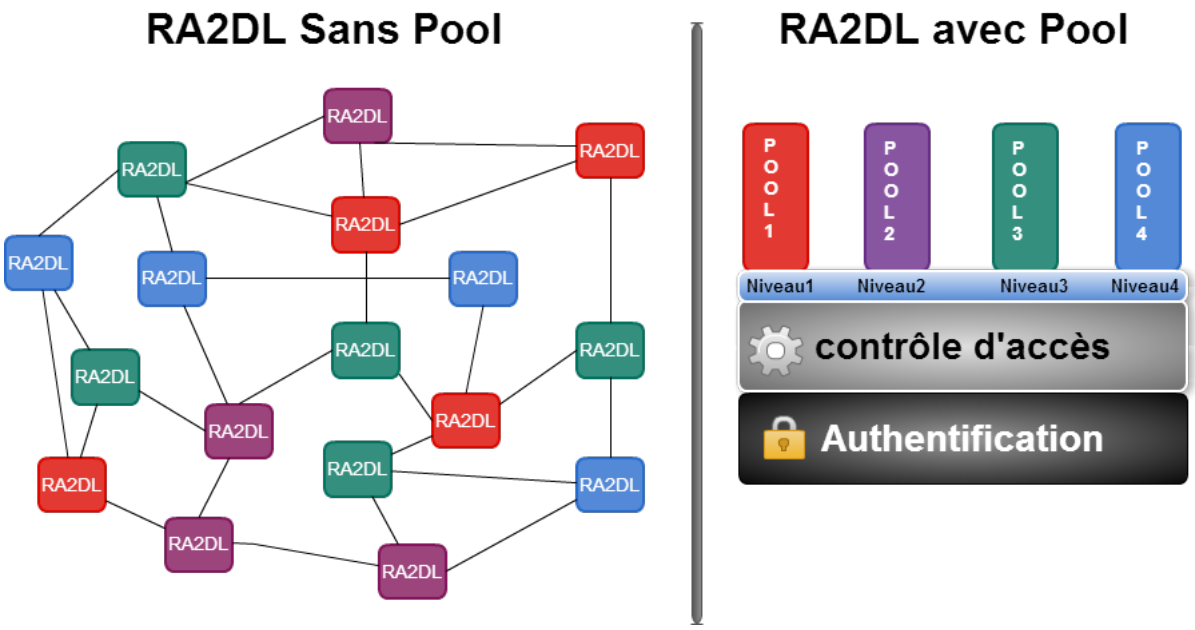

FIGURE 5.1 – RA2DL avec et sans Pool.

différents services de sécurité de ses composants, où chaque $RA2DL-Pool$ a un niveau de sensibilité de l'information de ses composants. Il détient des méthodes bien définies pour regrouper les composants RA2DL ensemble.

5.2.2 Architecture de RA2DL-Pool

Afin de pouvoir pleinement définir l'intérêt pour lequel nous avons proposé $RA2DL-Pool$, il est nécessaire de présenter son architecture avec ses éléments composites. La Figure 5.2 représente le diagramme de classes de l'architecture de $RA2DL-Pool$, qui montre les connexions entre les éléments suivants :

- **Contrôleur :** noté *Controller* c'est la partie cruciale du conteneur qui possède toutes les méthodes pour pouvoir gérer les composants à l'intérieur telle que l'authentification et la reconfiguration. Ainsi, il se présente comme une interface entre $RA2DL-Pool$, l'utilisateur, les autres conteneurs et les composants RA2DL. Chaque *contrôleur* est identifié par *id_pool*, qui est l'identifiant du conteneur.

- **Table :** Contient trois principales tables. D'abord, la table des utilisateurs notée (UT) réservée aux utilisateurs qui possèdent un identifiant (*id_user*), le mot de passe

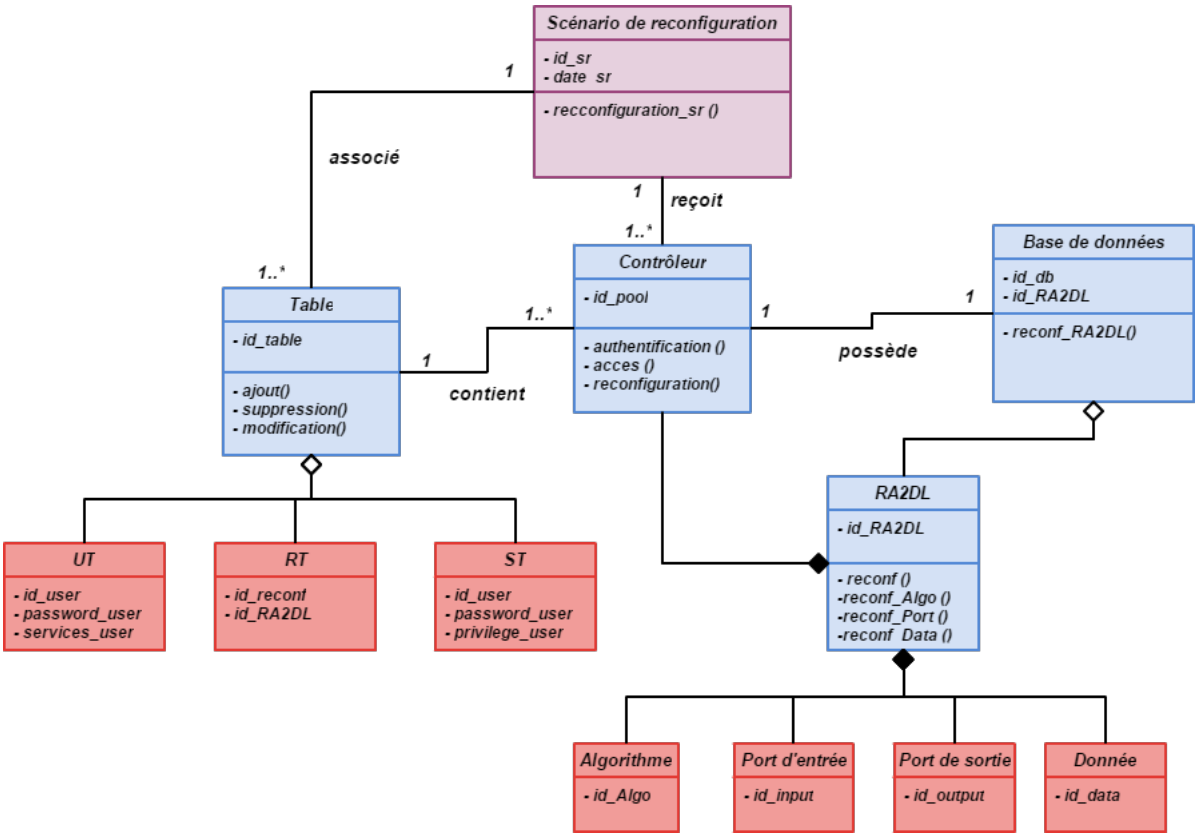

FIGURE 5.2 – Diagramme de classe de RA2DL-Pool.

(*password_user*) et les services (*service_user*) qui sont accessibles par l'utilisateur. Ensuite, la table de reconfiguration (*RT*) qui est une table dédiée spécialement à la gestion des scénarios de reconfiguration. Cette table contient l'identifiant de la reconfiguration (*id_reconf*) et l'identifiant du composant (*id_RA2DL*). Enfin, la table de sécurité (*ST*) qui contient les privilèges des utilisateurs (*privilege_user*),

- **Base de données :** C'est une base de données qui permet d'enregistrer les composants RA2DL dans $RA2DL - Pool$ de façon organisée et hiérarchisée.

- **Scénario de reconfiguration :** Définit l'ensemble des scénarios de reconfigurations réalisés avec $RA2DL - Pool$ ou avec ses composants RA2DL. Chaque scénario appliqué sera en relation avec les trois tables (UT, RT et ST) déjà décrites,

- **RA2DL :** Représente le composant RA2DL avec ses algorithmes et ses ports d'entrées/ sorties,

Méthode	Description
getRA2DL ()	renvoie le nombre de composants
Component-getRA2DL(int position)	retourne la position du composant
Component⌈⌉ getRA2DL ()	retourne un tableau de tous les composants
RA2DL-add(Component RA2DL, int position)	ajoute à *position* un composant
add (Component RA2DL, RA2DL constraints)	ajoute des contraintes à un composant
public void remove (int index)	supprime un composant de la position *index*
remove (RA2DL component)	supprime un composant
removeAll ()	supprime tous les composants de $RA2DL-Pool$
boolean isAncestorOf (RA2DL)	vérifie si le composant RA2DL est un parent
addContainerListener (pool)	ajoute un *listener* pour $RA2DL-Pool$
removeContainerListener (pool)	supprime un *listener* de $RA2DL-Pool$
processEvent(RA2DLEvent e)	reçoit les événements d'un RA2DL
addNotify ()	crée un père d'un tel composant qu'il
removeNotify ()	supprime le père d'un composants qu'il contient
Insetsgetinsets()	reçoit les $RA2DL-Pool$ actifs
list()	liste des composants

TABLE 5.1 – Méthodes de RA2DL-Pool.

Le tableau 5.1 clarifie les méthodes de $RA2DL-Pool$, qui sont des méthodes inspirées du langage Java et qui sont applicables aussi à ses composants.

5.2.3 Architecture Sécurisée à Base de RA2DL-Pool

La sécurité est l'une de nos priorités lors de la conception de notre architecture à base de RA2DL qui utilise principalement des communications entre les composants en utilisant $RA2DL-Pool$, nous allons minimiser la latence de la communication, de la reconfiguration et aussi résoudre des problèmes de sécurité tels que le reniflage de coordination et la falsification des reconfigurations.

Figure 5.3 présente un diagramme de classes d'une architecture sécurisée à base de $RA2DL-Pool$ avec l'aspect statique de relation entre un composant RA2DL et le conteneur $RA2DL-Pool$ et qui ne fournit pas d'information à propos de son comportement.

5.3 Mécanismes de Sécurité de RA2DL

La sécurité des conteneurs et ses composants RA2DL se cantonne généralement à garantir les droits d'accès aux données et aux composants en mettant en place des mécanismes

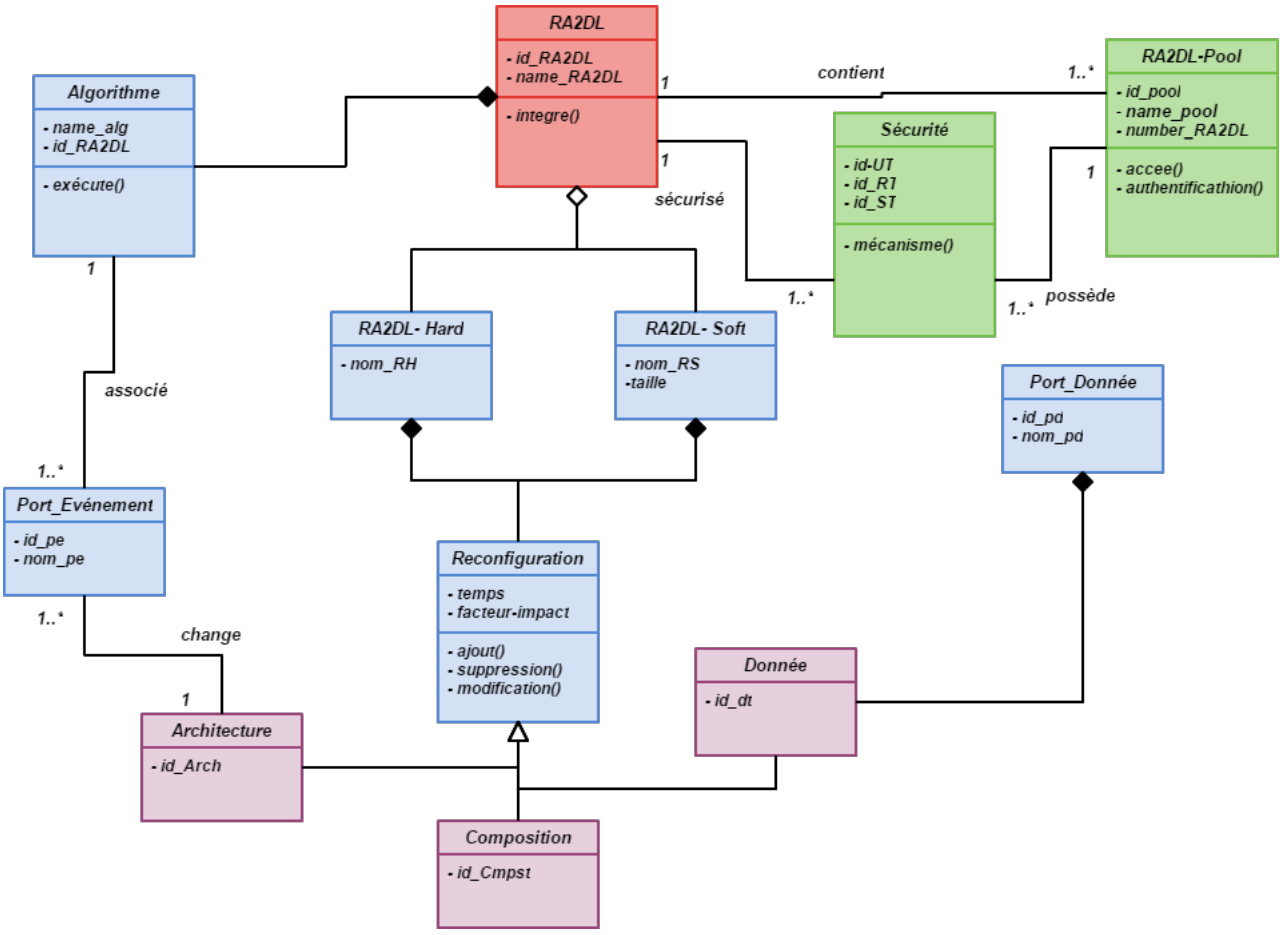

FIGURE 5.3 – Architecture sécurisée à base de RA2DL-Pool.

d'authentification et de contrôle d'accès permettant de s'assurer que les utilisateurs ou les composants spécifiques possèdent uniquement les droits qui leur ont été octroyés.

Les mécanismes de sécurité mis en place pour RA2DL peuvent néanmoins minimiser les attaques et des mauvaises exploitations des composants. Les consignes et les règles deviennent de plus en plus sécurisées au fur et à mesure de l'évolution de l'architecture. Ainsi, la sécurité de RA2DL mise en place n'empêche pas les reconfigurations des autres composants qui leur sont nécessaires, et garantit l'utilisation du composant en toute confiance.

Pour cette raison de sécurité, il est nécessaire de définir deux mécanismes de sécurité pour RA2DL, le premier est un mécanisme d'authentification et le deuxième est un mécanisme de contrôle d'accès dans le but d'avoir une politique solide de sécurité de RA2DL.

5.3.1 Mécanisme d'Authentification

L'authentification est le premier rempart aux attaques informatiques, pour ce là, nous proposons un mécanisme d'authentification où tous les utilisateurs des composants RA2DL ou $RA2DL - Pool$ doivent s'authentifier pour accéder aux services réservés.

Le mécanisme d'authentification est une méthode qui constitue une sécurité relativement fiable lorsqu'elle est bien utilisée en relation avec la table des utilisateurs (UT). Pour implémenter ce mécanisme, nous recourons au protocole **RADIUS** *(Remote Authentication Dial-In User Service)* développé par l'entreprise Livingston [YLY07] qui est un protocole de réseau centralisé, (AAA) Authentication, Authorization et Accounting utilisé pour la gestion des utilisateurs qui se connectent et utilisent un service de réseau.

Le diagramme de séquence de la Figure 5.4 décrit le principe de fonctionnement du mécanisme d'authentification avec le protocole RADIUS selon les quatre étapes suivantes :

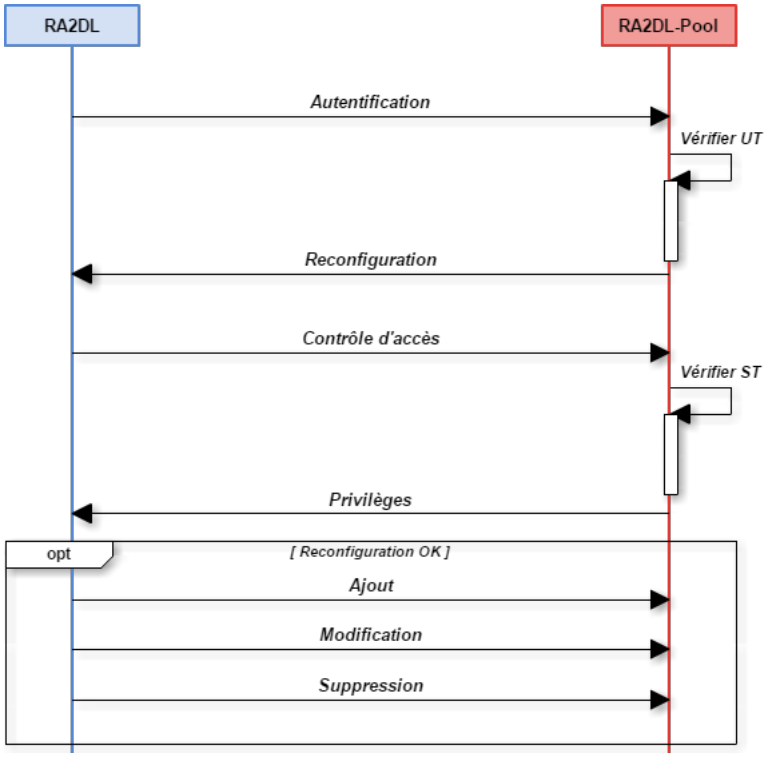

FIGURE 5.4 – Diagramme de séquence du mécanisme d'authentification.

1) Le *Contrôleur* exécute la requête de connexion au $RA2DL - Pool$ et la table utili-

sateurs (UT) récupère les informations de la connexion,

2) Le *Contrôleur* transfère les informations au composant RA2DL cible,

3) Le composant cible reçoit la demande de connexion du *Contrôleur*, la contrôle et renvoie les informations de configuration nécessaires à l'utilisateur pour accepter ou refuser l'accès.

4) Le *Contrôleur* envoie à l'utilisateur un message d'erreur en cas d'échec d'une authentification.

5.3.2 Mécanisme de Contrôle d'Accès

Le mécanisme de contrôle d'accès vient juste après le mécanisme d'authentification dont l'objectif est de définir les droits, les interdictions pour accéder au $RA2DL - Pool$. Deux tables seront utilisées avec ce mécanisme, la table de sécurité notée par (ST) et la table de reconfiguration notée par (RT).

Le contrôle d'accès est formalisé par le triple (S, C, M_{sc}) où, S représente les services offerts par les composants RA2DL ou les conteneurs $RA2DL - Pool$, C représente le composant RA2DL ou le conteneur $RA2DL - Pool$ et M_{sc} note les droits de chaque paire (S et C). Le diagramme d'activité de la Figure 5.5 met en évidence l'activité de ces deux mécanismes et des tests afin de parvenir à créer un composant RA2DL sécurisé.

5.4 Modélisation de RA2DL-Pool

Pour respecter les différentes contraintes de sécurité et les mécanismes que nous avons proposés, nous modélisons le conteneur $RA2DL-Pool$ avec ses aspects sécuritaires (Figure 5.6) par une machine à états qui décrit l'ensemble des états de notre méthode :

L'état *start* correspond au démarrage de l'interrogation ou la connexion avec $RA2DL-Pool$, l'état *Controller* représente le premier contact avec $RA2DL - Pool$ où la vérification de *id_user* et *password_user* avec la table des utilisateurs (UT) est obligatoire. Si l'authentification est acceptée et les coordonnées d'utilisateur sont vérifiées, alors il passe à l'état $Reconfiguration$ qui englobe tous les scénarios de reconfigurations. Après la vérification des paramètres suivants : (i) *id_sr* pour les IDs des scénarios de reconfiguration,

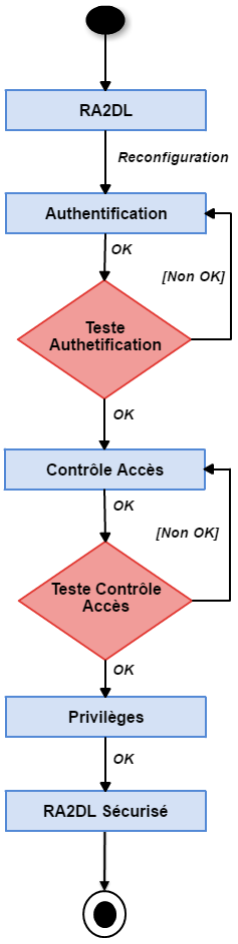

FIGURE 5.5 – Diagramme d'activité de deux mécanismes.

(ii) *privilege_user* pour les privilèges des utilisateurs dans la table de sécurité (ST) et (iii) id_reconf pour les IDs des reconfigurations de la table de reconfigurations (RT). Si tous ces IDs sont acceptés, alors l'utilisateur peut appliquer sa reconfiguration avec le composant réservé selon id_RA2DL. L'état *database* est associé pour faciliter le processus de reconfiguration des composants RA2DL.

Nous proposons les cinq propriétés suivantes afin de vérifier avec la logique TCTL la sécurité des composants RA2DL.

- **Propriété 1 :** pour chaque connexion avec un RA2DL-Pool, nous devrions vérifier l'authentification de l'utilisateur en utilisant la table UT. Cette propriété est décrite à l'aide de la formule suivante : *(Controller[].check id_user)AND (UT[].check passeword_user)*,

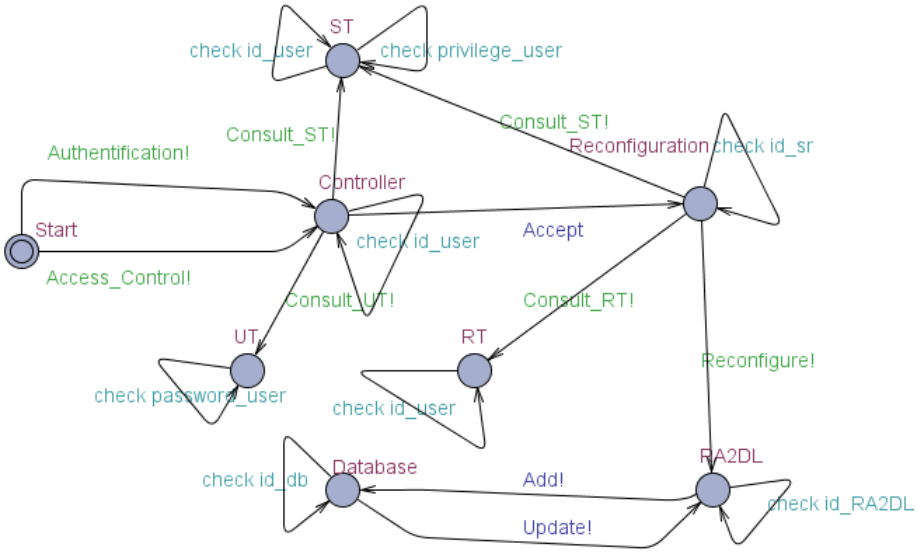

FIGURE 5.6 – Modélisation de RA2DL-Pool.

- **Propriété 2 :** avant l'exécution de tout scénario de reconfiguration, il est important de vérifier si la reconfiguration est enregistrée dans la table de reconfiguration (RT). Cette propriété est décrite à l'aide de la formule suivante : *(Reconfiguration[].check id_sr) AND (RT[].check id_reconf)*,

- **Propriété 3 :** la propriété concerne la vérification du mécanisme de contrôle d'accès. Cette propriété est décrite à l'aide de la formule suivante : *(Reconfiguration[].Reconfigure ! ⇒ RA2DL[].check id_RA2DL) AND (ST[].check privilege_user)*,

- **Propriété 4 :** chaque composant RA2DL doit impérativement être enregistré dans une base de données *Database* pour faciliter la manipulation des composants RA2DL et pour minimiser le temps d'exécution. Cette propriété est décrite à l'aide de la formule suivante : *RA2DL[].save ⇒ Database[].check id_db*,

- **Propriété 5 :** *RA2DL − Pool* fonctionne sans aucun blocage. Cette propriété est décrite par la formule suivante : *A[] not deadlock*.

La vérification de ces cinq propriétés est résumée dans le tableau 5.4 .

Propriété	Résultat	Temps (sec)	Mémoire (Mo)
Propriété 1	Vérifié	10.52	5.72
Propriété 2	Vérifié	9.12	4.82
Propriété 3	Vérifié	5.32	3.20
Propriété 4	Vérifié	13.25	6.56
Propriété 5	Vérifié	8.23	4.37

TABLE 5.2 – Résultat de vérification de RA2DL-Pool.

5.5 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode pour sécuriser le composant RA2DL intitulée $RA2DL - Pool$. Cette méthode a pour but de regrouper les composants RA2DL selon des points communs entre eux. L'objectif principal de l'utilisation de cette méthode de regroupement, est de sécuriser la reconfiguration et la connexion entre les composants RA2DL selon des méthodes bien définies.

Ensuite, nous avons mis en œuvre deux mécanismes de sécurité pour $RA2DL - Pool$. Ces mécanismes visent à vérifier l'authentification des accès extérieurs et de contrôler l'accès aux composants de $RA2DL - Pool$.

Nous avons modélisé la méthode proposée avec une machine d'états afin de montrer la validité de notre solution et d'évaluer les différents états de $RA2DL - Pool$ en termes de mécanisme de sécurité.

Dans les chapitres précédents, nous avons présenté dans un premier temps les méthodes de transformation d'un composant AADL statique vers un composant RA2DL reconfigurable. Nous avons ensuite montré comment les composants RA2DL sont déployés et reconfigurés en fonction des caractéristiques de l'application d'une architecture distribuée d'un système de contrôle industriel. Ceci achève la partie des contributions de cette thèse qui constitue l'étude théorique. Le prochain chapitre sera consacré à l'implantation d'un prototype $RA2DL - Tool$, ainsi qu'à l'applicabilité de ce prototype à trois études de cas différentes pour montrer l'applicabilité de notre approche.

Chapitre 6

Étude de Cas et Expérimentation

Introduction

Dans les chapitres précédents de ce mémoire, nous avons présenté une nouvelle méthodologie globale de reconfiguration d'un composant AADL afin de donner un composant RA2DL reconfigurable et flexible. Nous avons montré comment exploiter RA2DL pour produire une application distribuée reconfigurable, flexible et sécurisée. Nous avons aussi montré comment communiquer les composants RA2DL dans une architecture reconfigurable et distribuée dans les normes de sécurité.

Dans ce chapitre, nous présentons tous d'abord l'architecture détaillée d'un outil intitulé $RA2DL - Tool$ qui permet de réaliser l'ensemble des fonctionnalités d'un composant RA2DL de la spécification à l'implémentation. Ensuite, nous décrivons les principales contributions de la méthodologie proposée : la reconfiguration d'un composant AADL selon des scénarios de reconfiguration afin de donner une architecture flexible RA2DL, le calcul de la flexibilité du RA2DL $Flex - RA2DL$, la vérification des propriétés de RA2DL par la logique TCTL (Timed Compotational Tree Logic) [ACD90], la sécurité de RA2DL avec les containers $RA2DL - Pool$ et la déploiement.

Dans ce chapitre, nous appliquons la méthodologie proposée sur trois études de cas concrètes afin de valider les propriétés et d'évaluer la performance des implantations réalisées. Nous présentons un système Radar constitué de plusieurs composants AADL afin de les transformer en des composants RA2DL. Nous proposons également l'application du réseau IEEE 802.11 Wireless LAN comme une étude de cas afin de montrer la distribution

et la synchronisation des composants RA2DL. Pour montrer la variété d'application de l'approche proposée, nous exposons l'étude de cas du système de surveillance du corps *(Body-Monitoring System (BMS))* avec lequel nous traitons la sécurité et les mécanismes de sécurité des conteneurs $RA2DL - Pool$.

6.0.1 Outil RA2DL-Tool

Nous proposons dans cette section le prototype que nous avons développé au cours de cette thèse intitulé $RA2DL - Tool$. Nous avons utilisé le langage de programmation Java pour implanter l'outil $RA2DL - Tool$ de traduction d'un composant AADL vers un composant RA2DL. Cet outil est sous forme d'un plugin dédié à la plateforme open source Eclipse. Il prend comme entrée un modèle AADL (. aaxl) conforme au métamodèle AADL et génère un modèle RA2DL conforme au métamodèle RA2DL. Le nombre de lignes de code pour ce prototype est environ de 8000 lignes de code. L'architecture globale de $RA2DL - Tool$ est présenté au niveau de la Figure 6.1 contient les éléments suivants :

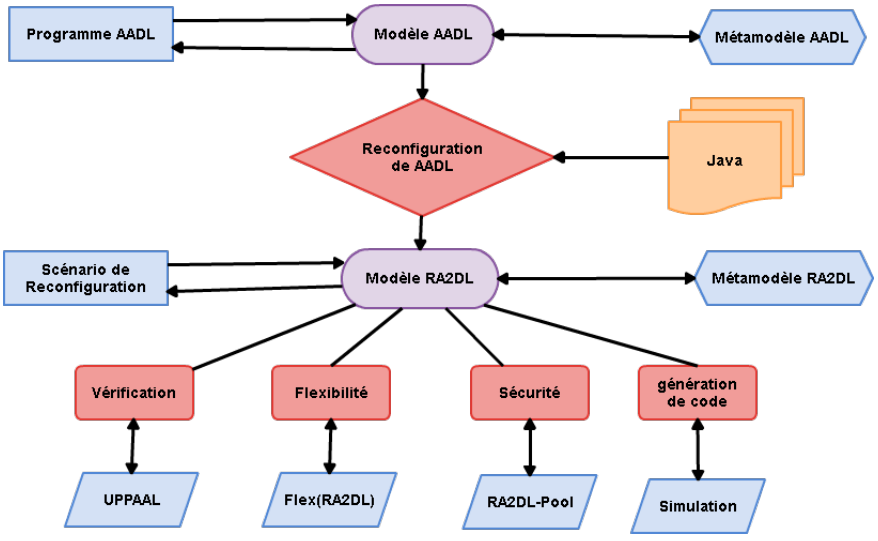

FIGURE 6.1 – Architecture globale de RA2DL-Tool.

— *Scénario de reconfiguration* : Représenté par un éditeur de texte qui permet la description des scénarios de reconfigurations. Nous l'avons développé en utilisant le langage Java et nous l'avons inséré au niveau de la plateforme Eclipce,

— *Métamodèle RA2DL* : Nous avons développé un méta-modèle RA2DL sous la plate-

forme Eclipse/EMF. Dans ce contexte, il est recommandé, par exemple pour la génération automatique d'éditeurs de modèles et d'avoir un tel élément de base d'un composant RA2DL,

— *Reconfiguration de AADL* : Utilisé pour transformer la description d'un composant AADL en Java afin de l'adapter à l'architecture d'un composant RA2DL. Nous avons développé un transformateur automatique sous forme d'un plugin pour la plateforme open source Eclipse, pour produire un composant RA2DL. Il prend en entrée un modèle AADL (.aaxl) conforme au méta-modèle AADL et génère un modèle RA2DL conforme au méta-modèle RA2DL,

— *Vérification* : C'est le module responsable de la vérification formelle des propriétés avec l'outil $RA2DL - Tool$, la vérification est assurée à l'aide du model checker UPPAAL et en utilisant la logique TCTL (Figure 6.2).

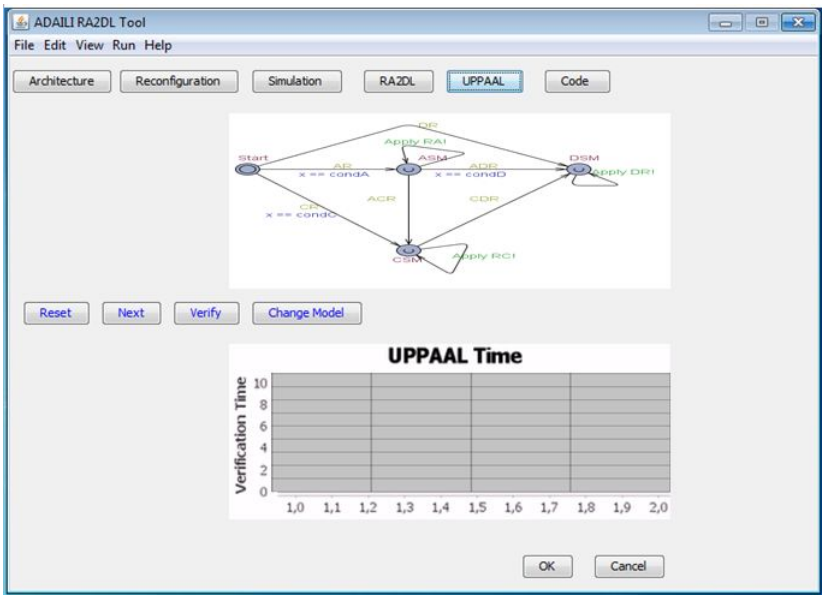

FIGURE 6.2 – Interface de vérification de RA2DL-Tool.

— *Flexibilité* : Pour calculer la flexibilité d'un composant RA2DL ainsi que la flexibilité de la totalité du système à base des composants RA2DL à l'aide de l'algorithme $Flex(RA2DL)$ et la fonction de flexibilité présentée au niveau du chapitre 3 (Figure 6.3).

— *Sécurité* : C'est la partie responsable de la sécurité des composants RA2DL en

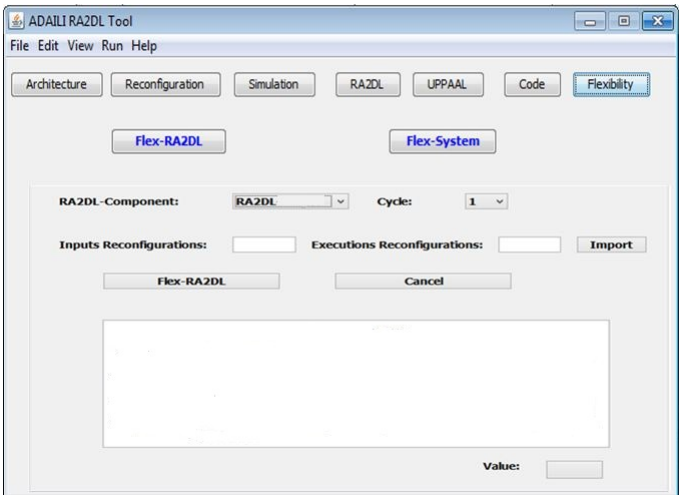

FIGURE 6.3 – Interface de calcul de la flexibilité.

relation avec les conteneurs $RA2DL - Pool$. Elle offre la possibilité à l'utilisateur de regrouper les composants RA2DL dans les $RA2DL - Pool$ et d'appliquer les mécanismes de sécurité proposés comme l'authentification et le contrôle d'accès.

— *Génération de code* : C'est une autre fonctionnalité assurée par l'outil $RA2DL - Tool$. Il s'agit de générer le code des composants RA2DL dédiés à l'application et conduit à la production d'une configuration dédiée à l'application pour un système à base des composants RA2DL préexistant pour faciliter le problème de correspondance entre les composants et aide à la réutilisabilité des composants à des autres applications. Nous utilisons pour la génération de code le processus évolutif (Figure 6.4) : les prototypes deviennent progressivement des composants RA2DL, qui sont raffinés à plusieurs niveaux lors de ses évolutions. Le dernier prototype est le composant RA2DL final. Les raffinements peuvent être fournis à partir de différents niveaux du processus (Spécification, vérification, modélisation et génération de code).

Les composants RA2DL sont générés selon les demandes des utilisateurs pour la version en langage C pour STM32F4 avec l'environnement *Coocox* et Arduino (Figure 6.5).

Après le lancement de la transformation d'un composant AADL vers un composant RA2DL, nous générons plusieurs fichiers automatiquement :

— Une représentation textuelle et XML du composant AADL,

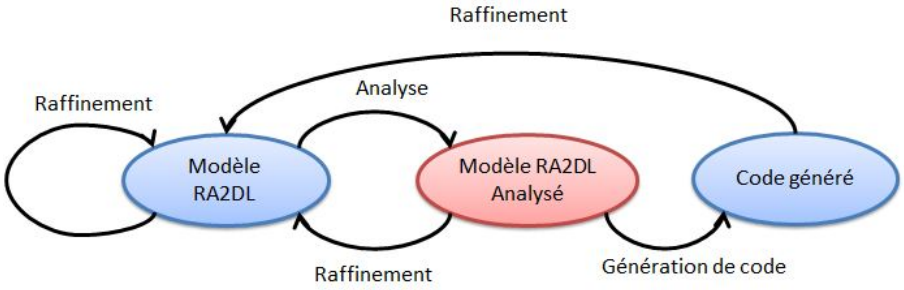

FIGURE 6.4 – Processus de génération de code évolutif.

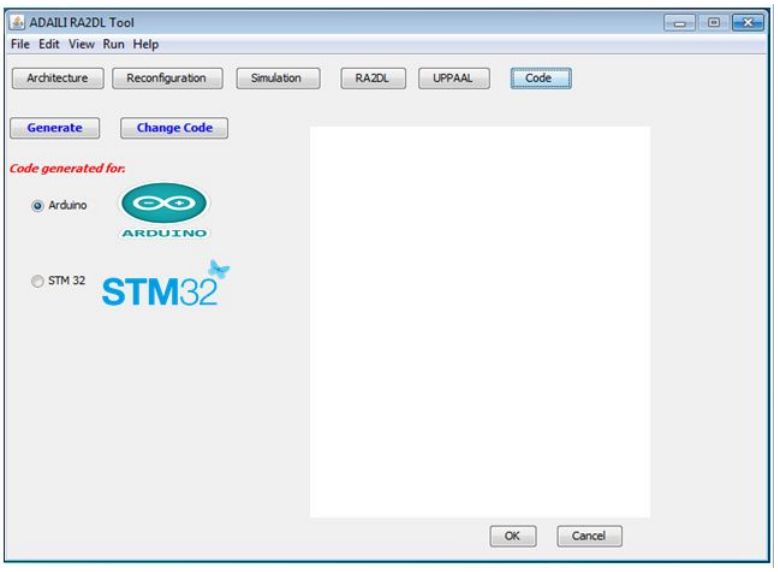

FIGURE 6.5 – Génération de code du RA2DL.

— Une représentation textuelle et XML du composant RA2DL,

— Un système à base de composant RA2DL exécutable,

— Des fichiers à l'aide du générateur de code fourni par le prototype $RA2DL - Tool$.

La génération de ces derniers ne nécessite aucune intervention humaine qui permette de faciliter l'exploration des composants RA2DL par des utilisateurs et la réutilisation des fichiers RA2DL générés, et qui permet de rendre aisé le cycle de développement d'une application reconfigurable distribuée.

6.1 Études de Cas

Trois études de cas ont été réalisées pour prouver la bonne applicabilité de l'approche de reconfiguration proposée. En premier lieu, un système radar sera utilisé comme une étude de cas présentant une application des reconfigurations et transformation d'AADL en RA2DL et pour montrer l'utilisation de la flexibilité. La deuxième étude de cas illustrant l'application du modèle d'exécution, la synchronisation et la distribution des composants RA2DL, concerne l'application du réseau IEEE 802.11 Wireless LAN. Nous expliquons à la fin de cette section comment nous avons appliqué la sécurité de RA2DL sur un système de surveillance du corps.

6.1.1 Système Radar

Nous utilisons comme une étude de cas un système radar à base de composants AADL représenté par l'outil STOOD [Dis04] Comme décrit dans [HS09] et détaillé dans l'archive de Ocarina [1].

La Figure 6.6 montre le diagramme de classes du radar constitué de plusieurs composants AADL répartis en deux catégories : matériels et logiciels :

1) Les composants matériels sont représentés par :

* Antenne (*Antenna*) : son rôle est de diffuser l'onde électromagnétique vers la cible avec le minimum de perte. Sa vitesse de déplacement, rotation et/ou balancement, ainsi que sa position, en élévation comme en azimut, sont généralement asservies mécaniquement, mais parfois aussi électroniquement,

* Processeur (*Processor*) : représente la partie d'exécution dans le radar,

* Mémoire (*Memory*) : sert à sauvegarder les espaces d'adressage,

* Bus (*Bus*) : assure la communication entre l'antenne et le processus principal stocké dans la mémoire sans gêner le fonctionnement interne des autres composants,

* Moteur (*Motor*) : sert lors de l'émission de l'antenne dans différentes directions dans le but de déterminer l'angle de rotation mesuré dans le plan vertical.

1. http ://aadl.telecom-paristech.fr

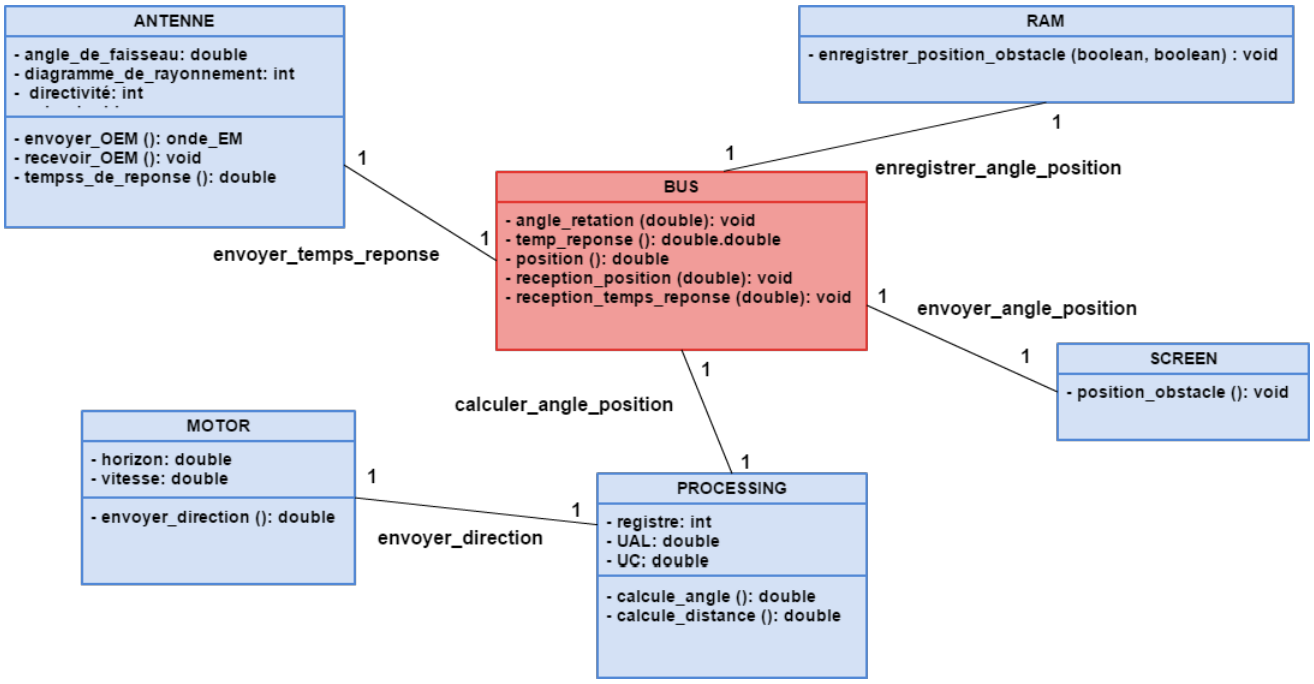

FIGURE 6.6 – Diagramme de classes du radar.

2) Les composants logiciels sont attribués au composant principal processus (*processing*) qui héberge les composants suivants (Figure 6.7) :

$$transmitter \rightarrow angle\_controller \rightarrow receiver \rightarrow analyser \rightarrow display.$$

* *transmitter* : le composant responsable qui envoie les signaux du radar à l'antenne,

* *angle_controller* : le composant qui calcule l'angle du radar,

* *receiver* : reçoit toutes les informations de l'antenne,

* *analyser* : compare les signaux émis et reçus pour effectuer la détection, la localisation et l'identification des objets,

* *display* : affiche les objets détectés sur l'écran radar.

Le composant (*processing*) possède deux données en entrée : *get_angle* pour repérer la position du moteur, *receive_pulse* pour mesurer la donnée venant de l'objet détecté. (*processing*) possède aussi deux événements en sortie : *to_screen* un événement pour l'écran et *send_pulse* pour envoyer les impulsions. Chaque composant interne possède également des données/événements en entrées/sorties pour supporter son interaction avec

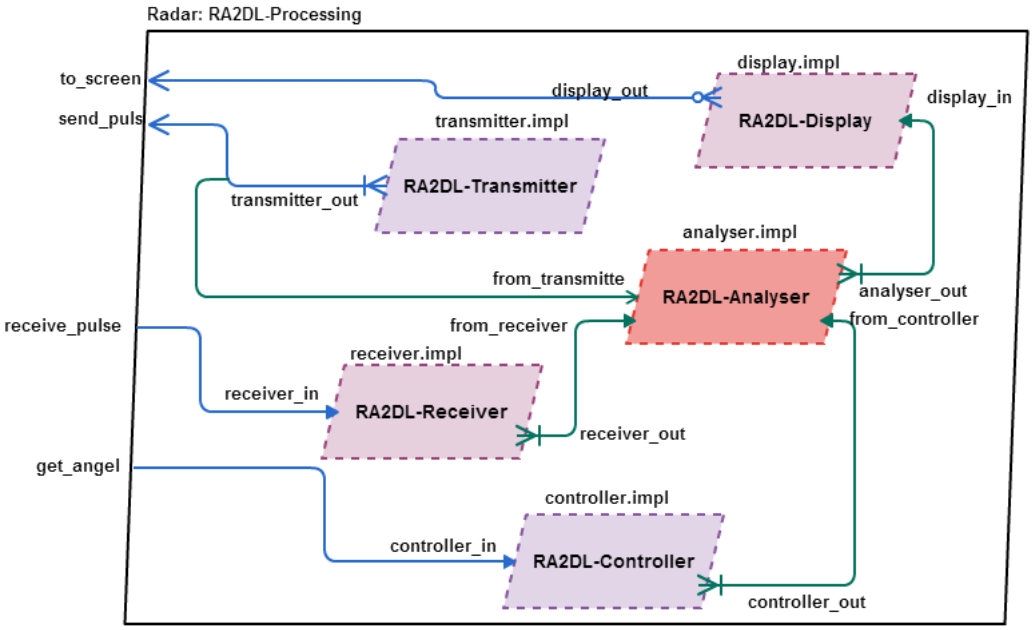

FIGURE 6.7 – Composants logiciels du système Radar.

d'autres composants.

Le diagramme de séquences représenté par la Figure 6.8 permet de montrer les interactions des composants du système radar dans le cadre d'un scénario de reconfiguration.

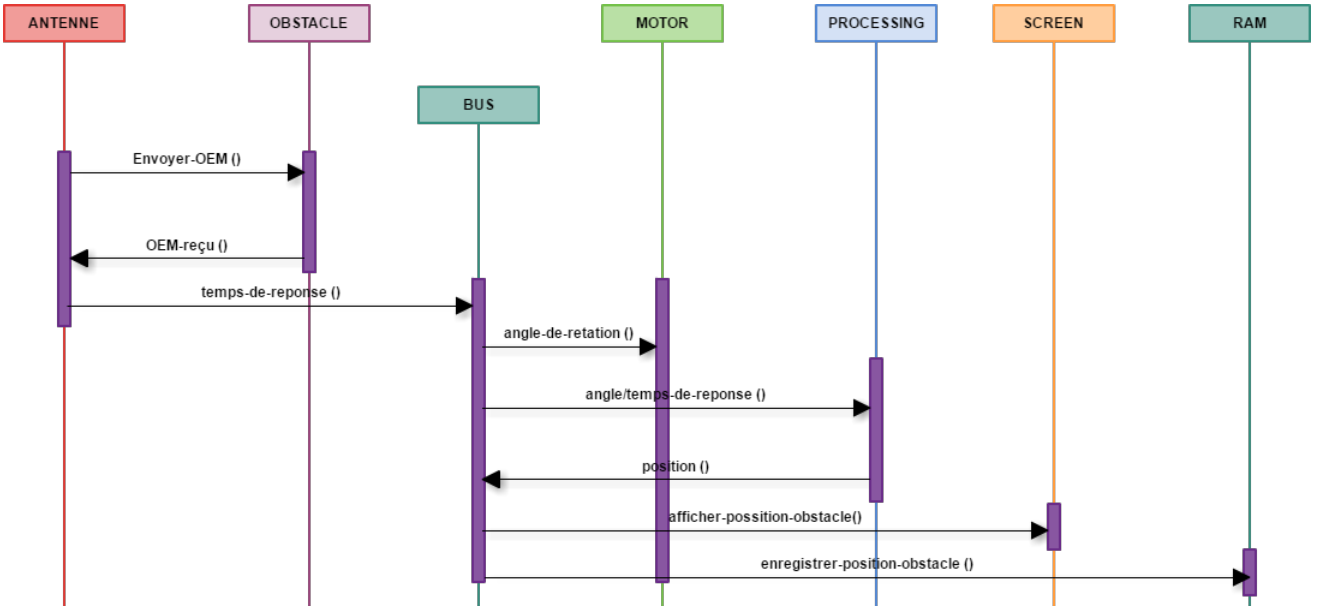

FIGURE 6.8 – Diagramme de séquence du système radar.

Bien que le radar soit bien testé, il manque la flexibilité et il ne possède pas les actions de reconfigurations qui peuvent adapter son comportement au moment de l'exécution en cas de panne, ou lorsque l'environnement du radar évolue et nécessite des changements utiles dans le comportement du radar. La reconfiguration est nécessaire pour les systèmes modernes et représente un nouveau défi pour l'étude de cas du radar.

6.1.1.1 Reconfigurations du Système Radar

Nous exposons certains scénarios de reconfigurations qui peuvent adapter le radar à son environnement au moment de l'exécution, supposons que le radar envoie M impulsions et détecte N objets à un moment donné. Notons par :

1) p_i le $i-th$ impulsion ($i \in [1..M]$) à envoyer de l'antenne avec une fréquence f_i.

2) O_j le $j-th$ l'objet détecté par le radar ($j \in [1..N]$), caractérisé par sa direction r_i, sa distance d_i du radar et sa surface s_i.

3) C en m^2, est un paramètre statique possédé par chaque radar à utiliser pour le traitement des zones. Ce paramètre est égal à H_Res si le radar fonctionne avec une haute résolution, sinon il est égal à L_Res en cas de faible résolution.

4) *condition_weather* est un paramètre booléen qui est égal à 0 en cas de mauvais temps (Neige ou pluie) ou 1 dans le cas où il fait beau.

5) *wind_speed* est un paramètre qui représente la vitesse du vent.

Nous supposons que le radar dispose de deux moteurs $M1$ et $M2$ pour faire tourner l'antenne avec deux vitesses en fonction de la vitesse du vent et de deux threads permettant l'émission d'impulsions avec deux périodes en fonction des conditions météorologiques : le premier EV_T1 envoie les impulsions chaque 6 ms, tandis que la seconde EV_T2 envoie les impulsions chaque 2 ms.

Nous notons que le calcul de l'angle peut être effectué avant la réception des signaux, ou bien dans le cas où l'on veut optimiser les performances du radar. Le calcul de l'angle avant la réception des signaux est effectué lorsque le trafic est faible, sinon il doit être effectué chaque fois qu'une impulsion est envoyée à partir de l'antenne.

Nous proposons ci-après cinq scénarios de reconfigurations à appliquer avec le système

radar :

1. **Reconfiguration 1 : S'il** existe un objet O_j ($j \in [1..N]$) tel que $s_j < C$, **Alors** le composant *processing* reconfigure le paramètre C de L_Res à H_Res pour détecter le maximum possible des objets,
2. **Reconfiguration 2 : Si** $condition\_weather == 1$, **Alors** les impulsions sont envoyées périodiquement de l'antenne par le thread EV_T1 toutes les 6 ms,
3. **Reconfiguration 3 : Si** $condition\_weather == 0$, **Alors** les impulsions sont envoyées périodiquement de l'antenne par le thread EV_T2 toutes les 2 ms,
4. **Reconfiguration 4 : Si** $wind\_speed >= 100\ km/h$, **Alors** le premier moteur $M1$ tourne avec un vitesse 45 tr/mn. Nous supposons dans ce cas un composant logiciel particulier $Rotat1$ exécuté pour contrôler le premier moteur,
5. **Reconfiguration 5 : Si** $wind\_speed < 100\ km/h$, **Alors** le second moteur $M2$ tourne avec une vitesse de 30 tr/mn. Nous supposons dans ce cas un composant logiciel particulier $Rotat2$ exécuté pour contrôler le second moteur.

Nous spécifions dans la Figure 6.9, les différents comportements de la partie contrôlée que nous pouvons suivre pour tous les scénarios de reconfigurations. Nous distinguons cinq branches de différents comportements. **Branch 1** spécifie le comportement du système lorsque la *Reconfiguration 1* est appliquée ($s_j < C$), **Branch 2** spécifie le comportement du système lorsque la *Reconfiguration 2* est appliquée ($condition\_weather == 1$), **Branch 3** spécifie le comportement du système lorsque la *Reconfiguration 3* est appliquée ($condition\_weather == 0$). **Branch 4** spécifie le comportement du système lorsque la *Reconfiguration 4* est appliquée ($wind\_speed > 100\ km/h$), et **Branch5** spécifie le comportement du système lorsque la *Reconfiguration 5* est appliquée ($wind\_speed < 100\ km/h$).

6.1.1.2 Modélisation du Système Radar

Nous exposons trois architectures de RA2DL dans le système radar représentées par l'automate de la Figure 6.10 et réparties en trois états (ASM1, ASM2 et ASM3) :

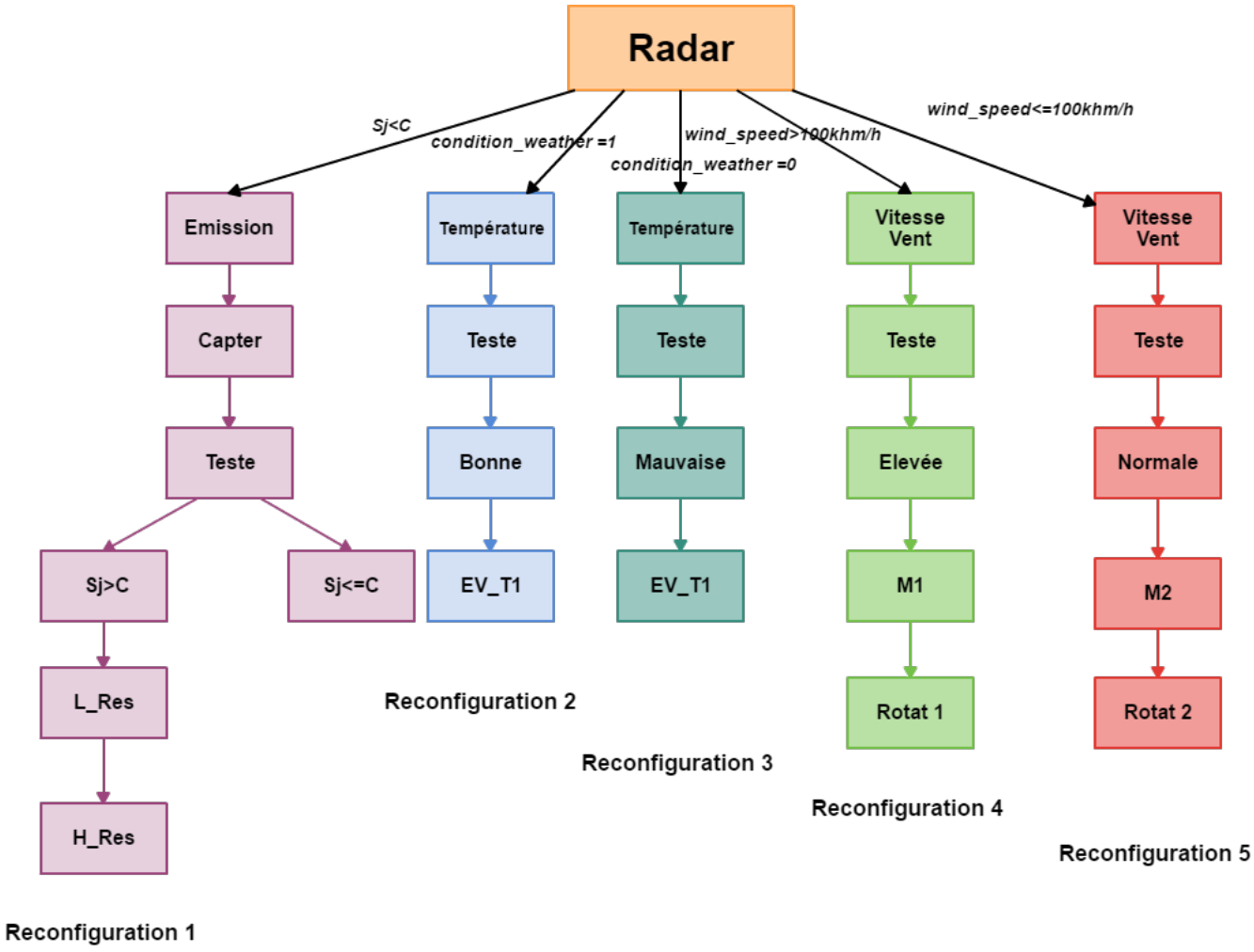

FIGURE 6.9 – Comportement du radar.

- Première architecture (ASM1) : quand la météo est parfaite et il fait beau ($IEM = condition\_weather == 1$), alors nous implémentons RA2DL selon la première architecture ($ASM1$). Dans cette architecture, nous distinguons au niveau de la Figure 6.11 deux compositions, le calcul de l'angle peut être effectué avant ou après la réception des signaux. Dans ce cas, le composant a deux compositions $CSM1$ et $CSM2$. Chaque composition se caractérisé à l'aide des intervalles de temps $T_1 = 20sec$ et $T_2 = 60sec$,

- Deuxième architecture (ASM2) : quand la météo devient imparfaite ($IEM = condition\_weather == 0$ et $DM = wind\_speed < 100\ km/h$), alors nous implémentons RA2DL selon la deuxième architecture,

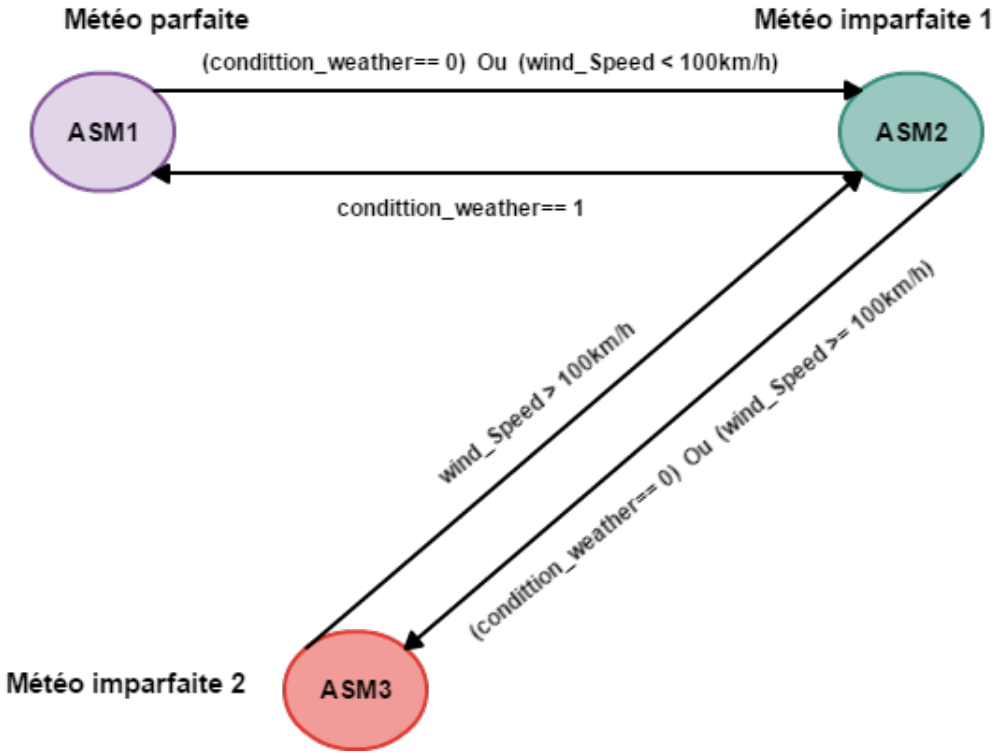

FIGURE 6.10 – Automate d'architecture du radar.

FIGURE 6.11 – Automate de composition de l'architecture ASM1.

- Troisième architecture (ASM3) : quand la météo devient parfaite et la vitesse du vent est élevée ($IEM = condition\_weather == 0$ et $DM = wind\_speed >= 100\ km/h$), alors nous implémentons RA2DL selon la troisième architecture.

Nous modélisons dans la Figure 6.12, le module contrôleur de RA2DL par une machine d'états de telle sorte que le niveau d'architecture est spécifié par ASM, dans lequel chaque état correspond à une architecture particulière du composant. Par conséquent, chaque

transition de ASM correspond à une activation ou désactivation des algorithmes et des événements d'entrées-sorties. Un état ASM correspond à une machine d'états dont le niveau de composition est noté CSM. Cette machine d'état spécifie toutes les formes de composition des algorithmes et des événements d'entrées-sorties à activer dans cet état du premier niveau de l'architecture. Un état du niveau de la composition correspond à une machine d'état dans le niveau donné DSM qui spécifie toutes les valeurs possibles de données d'un composant RA2DL.

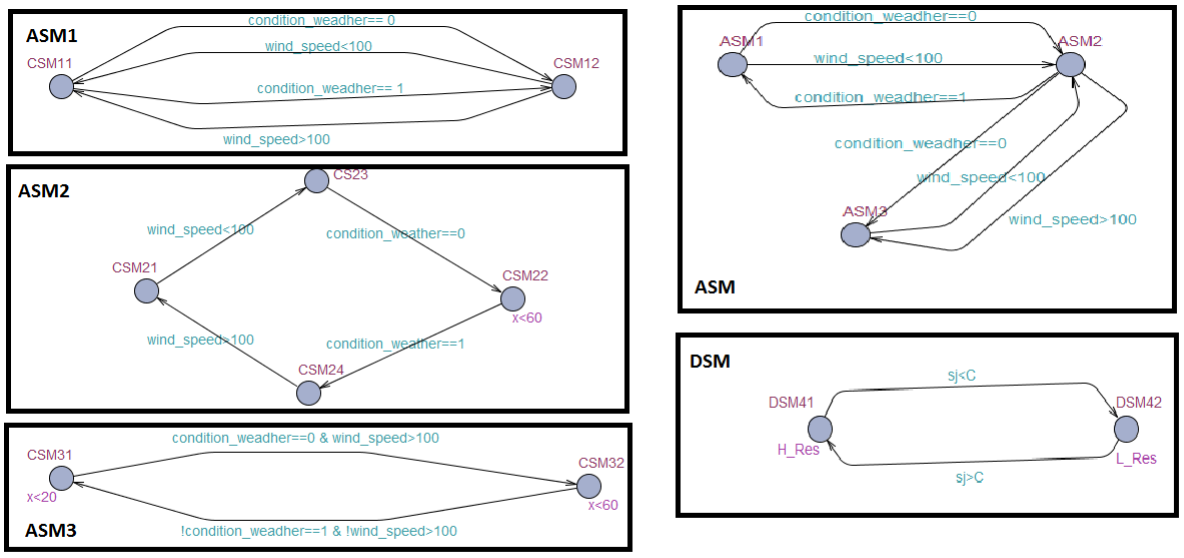

FIGURE 6.12 – Modélisation du module contrôleur.

La modélisation du module contrôlé est décrite dans la Figure 6.13 par cinq états : *Reconfiguration 1, Reconfiguration 2, Reconfiguration 3, Reconfiguration 4, and Reconfiguration 5* . Nous avons d'abord commencé par *Reconfiguration 1* qui correspond au composant *processing* lorsque la condition $S\_j < C$ est satisfaite. Dans ce cas, le composant *processing* reconfigure le paramètre C de L_Res vers H_Res. Si la situation météorologique est normale alors la condition $condition\_weather == 0$ est satisfaite. Dans ce cas, le radar passe à l'état $Reconfiguration3$ et par la suite, les impulsions sont envoyées périodiquement à partir du composant *antenna* par le thread EV_T2 chaque $2ms$ à cet état. Si $wind\_speed >= 100km/h$ alors le radar passe à l'état $Reconfiguration4$ où le premier composant moteur $M1$ tourne avec une vitesse de $45tr/mn$ et le composant $Rotat1$ est exé-

cuté pour contrôler le premier moteur. Sinon, lorsque la condition $wind\_speed < 100km/h$ est satisfaite alors le radar passe à l'état $Reconfiguration5$ quand le deuxième composant moteur $M2$ tourne avec une vitesse de $30tr/mn$ et le composant $Rotat2$ est exécuté pour contrôler le deuxième moteur. La même chose est répétée pour la $Reconfiguration2$ lorsque la condition $condition\_weather == 1$ est satisfaite.

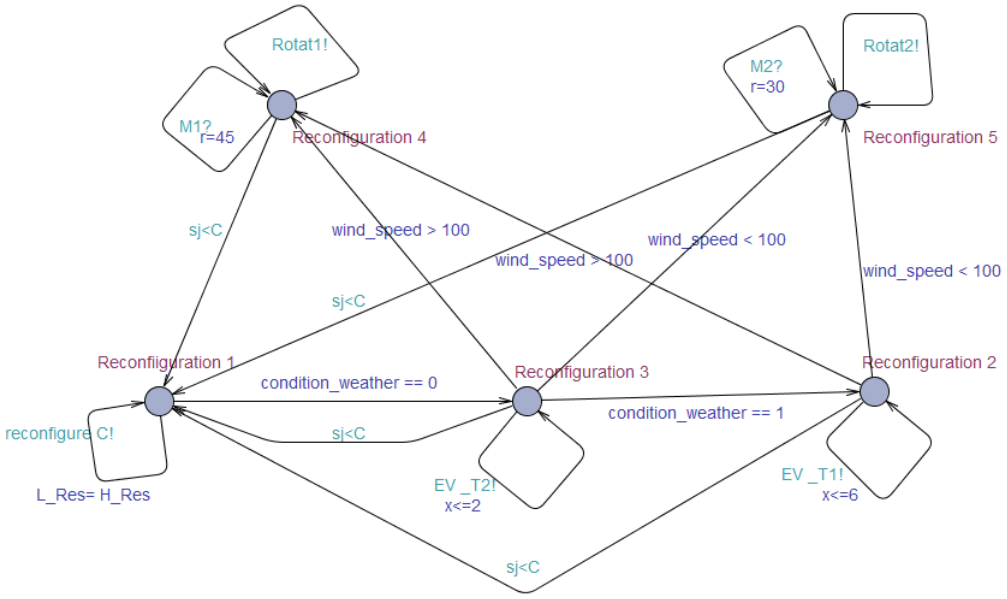

FIGURE 6.13 – Modélisation du module contrôlé.

6.1.1.3 Vérification des Propriétés du Radar

Nous vérifions l'exactitude du comportement du système radar après un scénario de reconfiguration afin d'éviter toute exécution imprévisible. Plus particulièrement, nous avons considéré les propriétés d'accessibilité, sécurité, vivacité et blocage avec la logique TCTL.

Les propriétés d'accessibilité sont vérifiées si un état du système donné est accessible à tout moment et et énoncent qu'une certaine situation peut être atteinte :

— **Propriété 1 :** Le système radar devrait fonctionner dans toutes les conditions météorologiques. Cette propriété est décrite par la formule suivante : $A[]RA2DL.(Reconfiguration2$ *et* $Reconfiguration3$ *et* $Reconfiguration4$ *et* $Reconfiguration5)$,

— **Propriété 2 :** Le moteur $M1$ doit tourner avec le thread $Rotat1$ lorsque la condition $wind\_speed >= 100km/h$ est satisfaite. Cette propriété est décrite par la formule

suivante : $A[]RA2DL.Reconfiguration4.M1$,

— **Propriété 3 :** Le moteur $M2$ doit tourner avec le thread $Rotat2$ lorsque la condition $wind\_speed < 100km/h$ est satisfaite. Cette propriété est décrite par la formule suivante : $A[]RA2DL.Reconfiguration5.M2$.

Les propriétés de sécurité suivantes doivent être prises en compte par tous les états accessibles, pour vérifier que quelque chose de mauvais ne se produit jamais au moment de reconfiguration du système :

— **Propriété 4 :** En cas de mauvaises conditions climatiques, le moteur $M1$ doit tourner avec une vitesse bien définie égale à 45 tr/mn. Cette propriété est décrite par la formule suivante : $A[]RA2DL.Reconfiguration4.r=45$,

— **Propriété 5 :** En cas de bonnes conditions climatiques, le moteur $M2$ doit tourner avec une vitesse bien définie égale à 30 tr/mn. Cette propriété est décrite par la formule suivante : $A[]RA2DL.Reconfiguration5.r=30$.

Les propriétés de vivacité sont vérifié en analysant l'ensemble de toutes les traces possibles du système et qui sont spécifiées comme suit :

— **Propriété 6 :** Vivacité Bornée : Un RA2DL doit reconfigurer l'envoi du signal avec un minimum de 2 secondes pour la Reconfiguration 3 et un maximum de 6 secondes pour la Reconfiguration 2. Cette propriété est décrite par la formule suivante : $A[](RA2DL.ASM1 \Rightarrow RA2DL.Reconfiguration3.x <=2)\ and\ (RA2DL.ASM2 \Rightarrow RA2DL.Reconfiguration2.x <=6)$,

— **Propriété 7 :** Chaque fois que la condition $wind\_speed >= 100km/h$ est vérifiée, le moteur $M1$ doit éventuellement tourner. Cette propriété est décrite par la formule suivante : $RA2DL.Reconfiguration3 \Rightarrow RA2DL.Reconfiguration4$,

— **Propriété 8 :** Chaque fois que la condition $wind\_speed < 100km/h$ est vérifiée, le moteur $M2$ doit éventuellement tourner. Cette propriété est décrite par la formule suivante : $RA2DL.Reconfiguration3 \Rightarrow RA2DL.Reconfiguration5$.

La propriété de blocage est décrite comme suit :

— **Propriété 9 :** Le système radar est pourvu de blocage. Cette propriété est décrite par la formule suivante : $A[]$ *not deadlock*.

Propriétés	Résultat	Temps (sec)	Mémoire (Mo)
Propriété 1	Vérifié	16.37	4.45
Propriété 2	Vérifié	4.48	4.03
Propriété 3	Vérifié	12.20	4.20
Propriété 4	Vérifié	10.34	4.20
Propriété 5	Vérifié	3.44	4 ;03
Propriété 6	Vérifié	8.50	4.20
Propriété 7	Vérifié	13.16	4.45
Propriété 8	Vérifié	7.12	4.20
Propriété 9	Vérifié	4.23	4.03

TABLE 6.1 – Résultats de vérifications des propriétés du radar.

La vérification de ces propriétés est résumée dans le tableau 6.1.

6.1.1.4 Implémentation et Simulation du Radar

Nous avons implémenté le radar avec les composants RA2DL représentés dans la Figure 6.14, que nous avons développés avec un micro-contrôleur Arduino Uno possédant un processeur ATmega32 (8 bits) et une mémoire SRAM 2KB. L'antenne est représentée par un capteur à ultrasons hc-SR04. Le moteur est représenté par Servomoteur df05bb avec une alimentation de 160mA (4.8V) et avec une vitesse de 60 degrés/ 0,17 seconde.

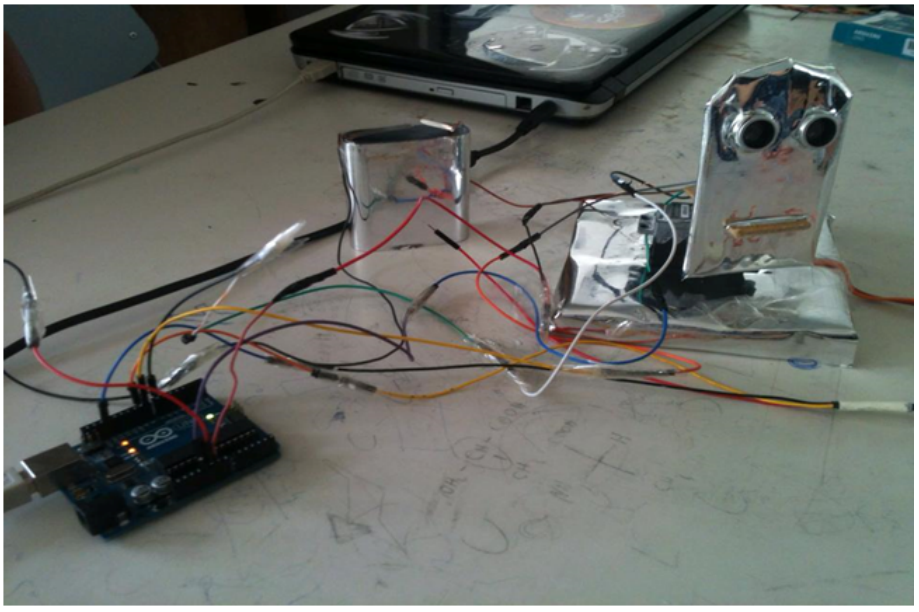

FIGURE 6.14 – Implémentation du système radar.

Pour analyser la faisabilité de notre système dans chaque reconfiguration, nous appliquons une reconfiguration lorsque la météo devient imparfaite et nous changeons les moteurs (Figure 6.15).

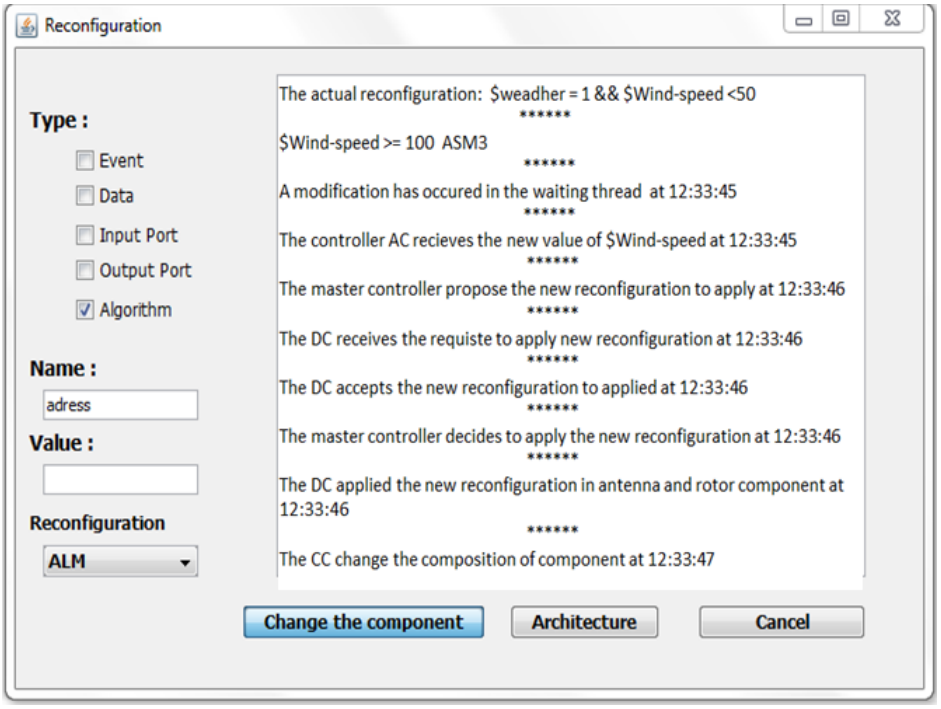

FIGURE 6.15 – Exemple de reconfiguration.

Le résultat de cette reconfiguration est affiché sur l'écran du radar dans la Figure 6.16.

6.1.2 IEEE 802.11 Wireless LAN

Afin d'évaluer la distribution et la synchronisation de RA2DL, nous avons mené une étude de cas basée sur un réseau de capteurs sans fil IEEE 802.11 [CCG98] (Figure 6.17). Ce scénario représente une collection de nœuds de capteurs modélisés à l'aide des deux composants RA2DL : $RA2DL-Sender$ and $RA2DL-receiver$, reliés par un canal décrit par le composant $RA2DL-channel$, qui est à son tour connecté à un autre réseau câblé. La description des composants de cette étude de cas est la suivante :

RA2DL-sender : est destiné à envoyer des paquets dans le réseau. Il commence par un paquet de données prêt à envoyer, et détecte $RA2DL-channel$. Si le canal reste libre, alors le composant $RA2DL-sender$ entre dans sa période de vulnérabilité et commence

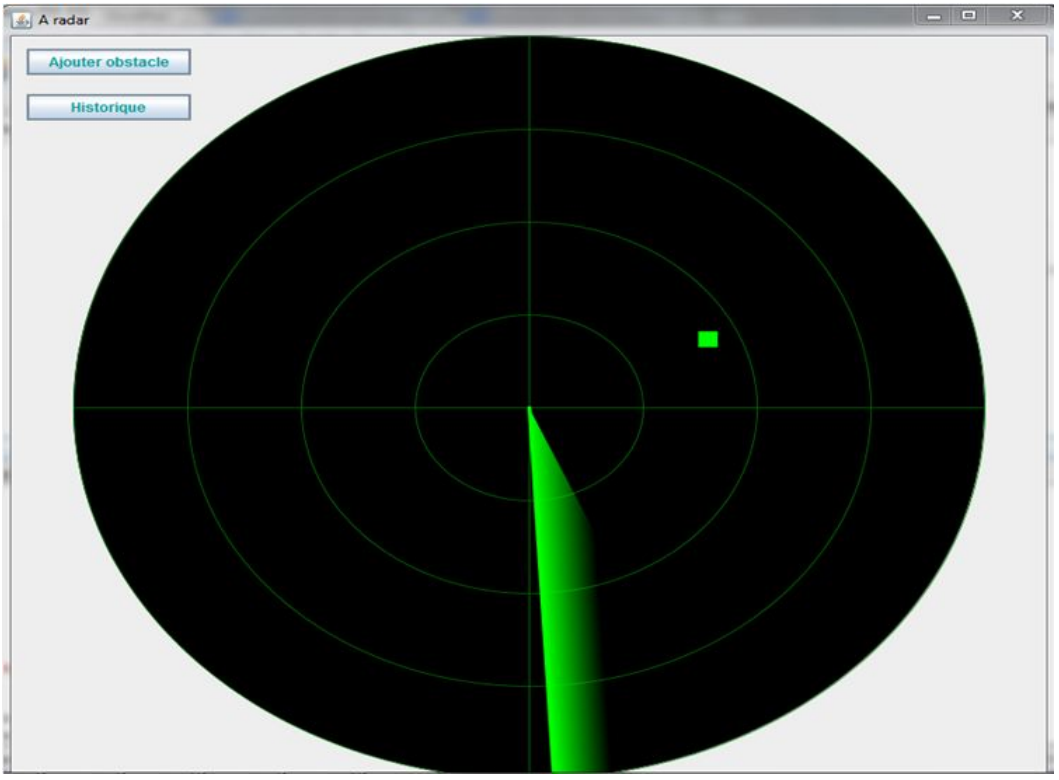

FIGURE 6.16 – Interface de l'écran d'affichage.

à envoyer les paquets (événement send), sinon mis en attente (backoff) par l'intermédiaire d'une transition urgente. Le temps requis pour envoyer un paquet est non-déterminée (T_{min} et T_{max}) où, T_{min} représente le temps minimum d'envoi et T_{max} le temps maximum.

RA2DL-channel : joue le rôle d'intermédiaire dans le réseau, en tant que canal de transmission de paquets entre les composants du réseau. Le succès de la transmission dépend de la collision, il est enregistré en réglant la valeur de la variable d'état $c1$ de $RA2DL - channel$, le composant $RA2DL - sender$ teste immédiatement l'occupation du composant $RA2DL - channel$ (représenté par l'état d'urgence TESTE C). Si le canal est occupé, le $RA2DL - sender$ passe à l'état d'attente, Sinon il attend un accusé de réception. Si le paquet a été envoyé correctement ($status = 1$), alors il attend l'envoi de l'accusé de réception. D'un autre côté, si le paquet n'a pas été transmis correctement ($status = 2$), alors la destination $RA2DL - receiver$ ne fait rien. Dans ce cas, $RA2DL - sender$ reste en attente.

RA2DL-receiver : représente le composant destiné à recevoir les paquets délivrés par

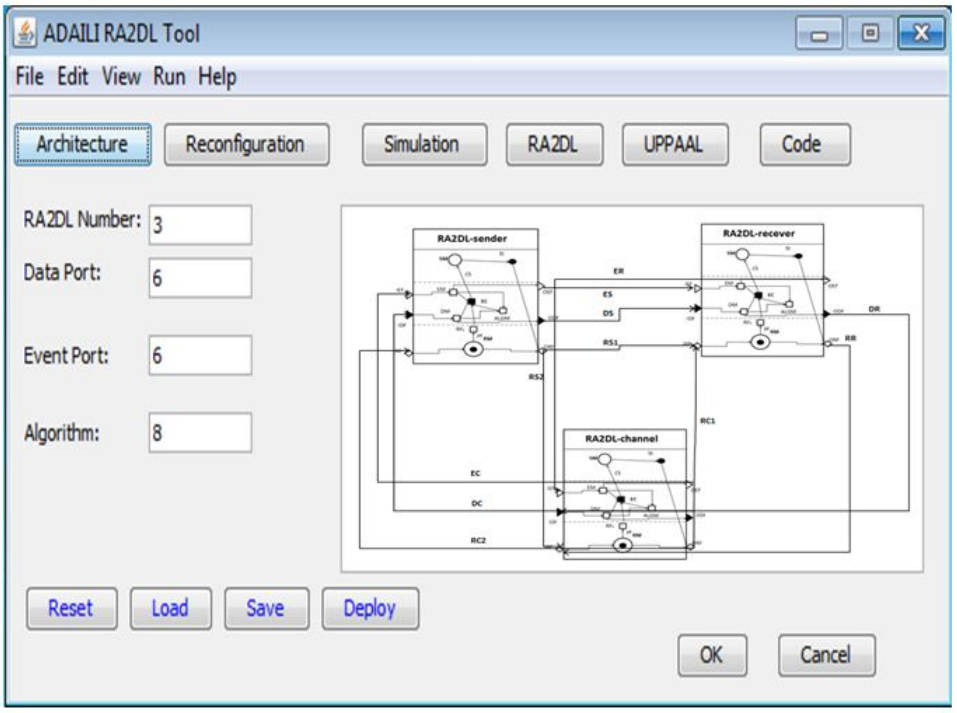

FIGURE 6.17 – Composants RA2DL du réseau de capteurs sans fil.

le composant $RA2DL-sender$. Les messages du composant $RA2DL-sender$ doivent être transmis à travers le réseau. Si le réseau est occupé, alors ces messages doivent être stockés dans le composant $RA2DL-channel$ et ils retardent leur traitement avec l'augmentation de l'espace mémoire du composant $RA2DL-channel$.

L'expérimentation de cette étude de cas avec le RA2DL classique présente un ensemble de problèmes :

i) Problème de la gestion du flux des reconfigurations. Si un composant reçoit plusieurs scénarios de reconfigurations en même temps,

ii) Problème d'exécution pour résoudre l'impasse et l'ambiguïté lorsque l'exécution d'une reconfiguration de chaque composant RA2DL se produit,

iii) Problème de synchronisation lorsque les reconfigurations sont synchrones avec d'autres composants,

iv) Problème de coordination lorsque les composants RA2DL sont interconnectés et communiquent à l'aide de flux de reconfiguration, des données et des événements.

Le composant $RA2DL - channel$ reçoit un ensemble de flux de reconfigurations en entrée au moment de l'exécution à partir des composants $RA2DL - sender$ et $RA2DL - receiver$. Par exemple, en changeant la fréquence d'envoi notée par (S_f) pour donner une maximisation ou une minimisation. $RA2DL - channel$ possède deux variables notées $c1$ et $c2$ qui enregistrent respectivement le statut du paquet envoyé par les deux composants $RA2DL - sender$ et $RA2DL - receiver$, et qui sont mis à jour à la fois quand une station commence à envoyer un paquet (*event send*) ou termine l'envoi d'un paquet (*event finish*). Ces variables ont l'interprétation suivante : $ci = 0$: rien envoyé par la station $i, ci = 1$: Paquet étant envoyé correctement de la station i, $ci = 2$: Paquet étant envoyé brouillé de la station i.

Nous montrons certains scénarios de reconfigurations qui peuvent adapter et coordonner les composants RA2DL de cette application à son environnement au moment de l'exécution. Nous supposons à un moment donné que le $RA2DL - sender$ envoie M paquets reçus par $RA2DL - receiver$. Notons par :

i) Pi (i $\in$[1..M]) (M représente l'ensemble des baquets) le paquet à envoyer à partir du $RA2DL - receiver$ avec une fréquence Fi et une vitesse de transformation (TS) entre T_{min} et T_{max}. Chaque Pi possède une taille Se pendant l'envoi du $RA2DL - sender$ et Sr une taille à la réception du $RA2DL - receiver$.

ii) Pi passe le composant $RA2DL - channel$ qui possède une variable booléenne C (0 si occupé et 1 si libre)

iii) Nous supposons que nous avons deux threads permettant l'émission de paquets avec deux périodes selon les conditions de canal : le premier envoie le paquet chaque 6 *sec* lorsque le canal est occupé $(C = 0)$ alors que le second envoie chaque 2 *sec* lorsque le canal est libre $(C = 1)$.

Nous proposons les cinq scénarios de reconfigurations suivants :

1. **Reconfiguration 1 :** Si $RA2DL - sender$ envoie le paquet par $RA2DL - channel$ vers $RA2DL - receiver$, alors le contenu du paquet ne doit pas modifier ou changer le composant $RA2DL - channel$.
2. **Reconfiguration 2 :** Si $RA2DL - channel$ reçoit des reconfigurations contradic-

toires pour minimiser ou maximiser la fréquence d'envoi (S_f), alors le composant $RA2DL-channel$ reconfigure le paramètre (S_f),

3. **Reconfiguration 3 :** Si $RA2DL-channel$ est occupé ($C=0$), alors le paquet doit envoyer périodiquement du $RA2DL-receiver$ par le thread ($EV-T1$) toutes les 6 sec,

4. **Reconfiguration 4 :** Si $RA2DL-channel$ est libre ($C=1$) alors le paquet doit envoyer périodiquement du $RA2DL-receiver$ par le thread ($EV-T2$) toutes les 2 sec,

5. **Reconfiguration 5 :** Si la taille du paquet Se est grande au moment de l'envoi, alors Se doit compresser Sr à l'aide le composant $RA2DL-recevier$.

Nous supposons quatre scénarios de reconfigurations : $RS1$, $RS2$, RR et RC. $RS1$ pour amplifier la fréquence f_s du composant $RA2DL-channel$, $RS2$ pour minimiser la taille du paquet du composant $RA2DL-receiver$, RR pour changer la variable cc et RC pour changer la taille du paquet au niveau du composant $RA2DL-receiver$. Les informations véhiculées par le jeton RT sont décrites dans le tableau 6.2.

	DA (Adresse du composant)	Synchrone	Asynchrone	PF (Facteur de priorité)
RS1	RA2DL-receiver	0	1	1
RS2	RA2DL-channel	1	0	3
RR	RA2DL-channel	0	1	2
RC	RA2DL-sender	0	1	4

TABLE 6.2 – Informations du jeton.

La reconfiguration du composant $RA2DL-channel$ est synchrone et dépend des composants $RA2DL-sender$ et $RA2DL-receiver$, lorsque $RA2DL-channel$ a des problèmes de collision. Dans ce cas, un flux de reconfiguration synchrone est envoyé aux composants $RA2DL-sender$ et $RA2DL-receiver$ pour informer d'une nouvelle reconfiguration à appliquer. Automatiquement, $RA2DL-sender$ et $RA2DL-receiver$ utilisent un sémaphore (S).

```
Semaphore (RA2DL-sender, RA2DL-receiver)
RA2DL-sender(s:)
RA2DL-receiver(s):
n(s) := n(s) - 1;
if n(s)<0 then;
State(RA2DL-sender) := blocked;
State(RA2DL-receiver) := blocked;
enter (RA2DL-sender, f(s))
enter (RA2DL-receiver, f(s))
```

Pour la communication IEEE 802.11 Wireless LAN, nous distinguons trois types de composants participants :

— $RA2DL - sender$ (station 1) : il démarre la communication et quand une erreur spécifique se produit dans le réseau, le composant associé $RA2DL - receiver$ essaie de la corriger, décide de la nécessité de reconfiguration dans l'ensemble du réseau (les autres composants RA2DL doivent être conscients de cette modification) et informe le composant coordinateur ($RA2DL - coordinator$).

— $RA2DL - coordinator(CR)$: il est le principal composant qui vise à assurer la coordination entre les différents composants RA2DL. Lorsqu'il reçoit une demande de reconfiguration, il cherche dans sa liste des composants RA2DL le composant cible qui devrait être informé. Il envoie une requête à ce composant de la liste et attend sa réponse.

— $RA2DL - receiver$ (station 2) : il reçoit une requête de reconfiguration à partir du composant $RA2DL - coordinator$. Dans ce cas, il vérifie la possibilité d'appliquer la reconfiguration. S'il est possible, il envoie une réponse positive, sinon il envoie une réponse négative.

Le diagramme de séquence de la Figure 6.18 montre la coordination entre ces trois stations lorsque la taille des paquets est élevée. Dans ce cas, le composant CR utilise la matrice CM pour compresser la taille des paquets au niveau du composant $RA2DL - receiver$.

Nous vérifions cinq propriétés pour un comportement correct du modèle d'exécution et une bonne coordination entre les composants RA2DL du système après tout scénario de reconfiguration afin d'éviter toute exécution imprévisible. Ces propriétés sont exprimées par des formules selon la logique TCTL.

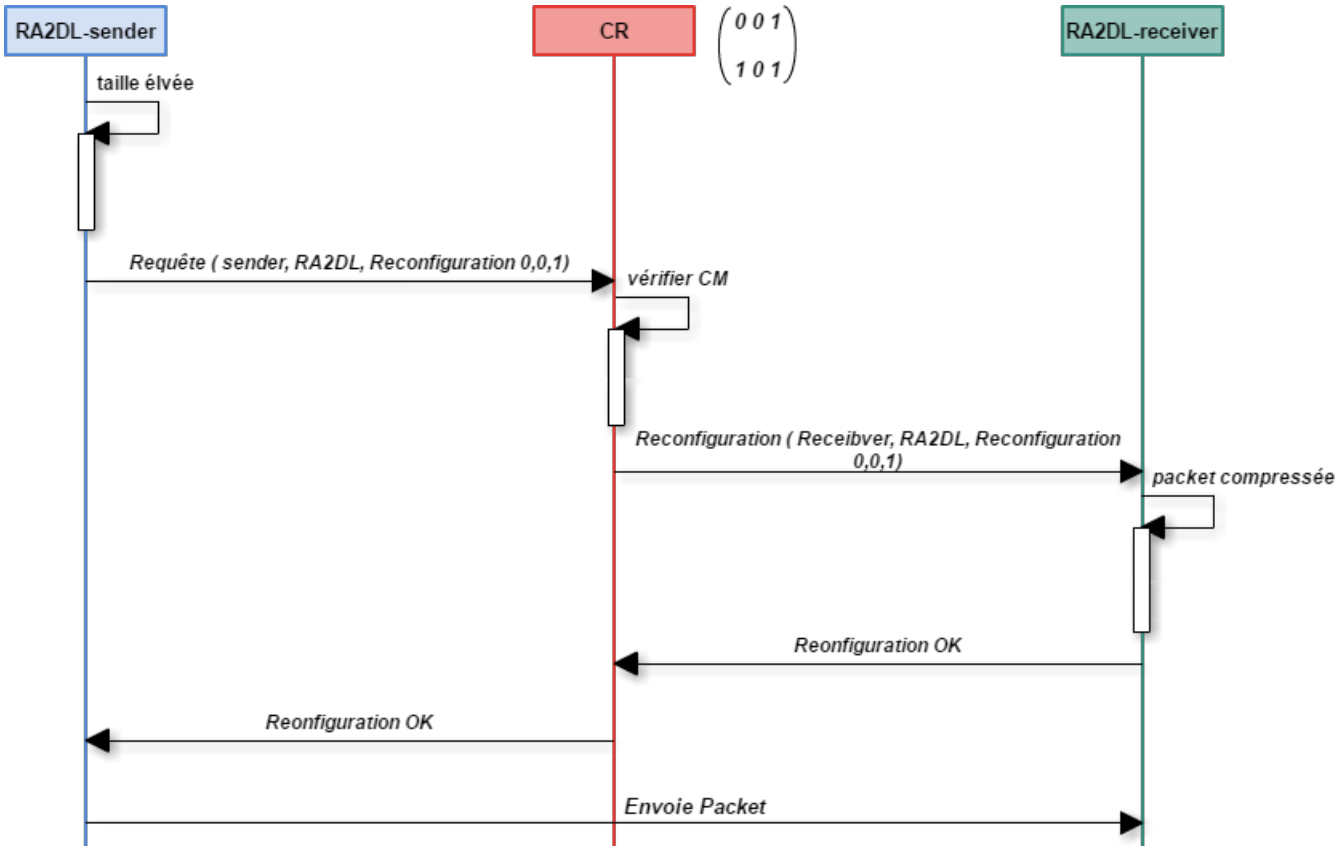

FIGURE 6.18 – Coordination entre RA2DL-sender et RA2DL-receiver.

Propriété 1 : le modèle d'exécution doit gérer les reconfigurations concurrentes selon le facteur de priorité PF du jeton et décide quelle reconfiguration doit être exécutée en premier. Cette propriété est décrite par la formule suivante : $RT[].PF!.maxPF(RF)$,

Propriété 2 : si la reconfiguration est synchrone, le middleware de synchronisation SM doit nécessairement envoyer un jeton pour bloquer le composant cible dans le sémaphore (S). Cette propriété est décrite par la formule suivante : $SM[].semaphore(s) \Rightarrow St[]$,

Propriété 3 : cette propriété signifie que lorsqu'une erreur se produit dans RA2DL, le composant $RA2DL - coordinator$ doit être informé. Cette propriété est décrite par la formule suivante : $RA2DL - server[].error - occurs \Rightarrow RA2DL - coordinator$,

Propriété 4 : cette propriété signifie que nous ne pouvions pas recevoir deux notifications différentes du composant $RA2DL - coordinator$ en même temps. Cette propriété est décrite par la formule suivante : $NOT(RA2DL - sender[]$ AND $RA2DL - recever[])$,

Propriété 5 :le réseau est pourvu de blocage. Cette propriété est décrite par la formule suivante : $A[]not\ deadlock$.

La vérification de ces propriétés est résumée dans le tableau 6.3.

Propriété	Résultat	Temps (sec)	Mémoire (Mo)
Propriété 1	Vérifié	12.03	5.34
Propriété 2	Vérifié	5.9	3.56
Propriété 3	Vérifié	10.23	5.78
Propriété 4	Vérifié	11.34	4.23
Propriété 5	Vérifié	7.24	3.96

TABLE 6.3 – Résultat de la vérification du réseau.

Pour simuler le comportement d'une architecture distribuée à base de composants RA2DL, nous développons un prototype avec l'outil $RA2DL-Tool$ appelé *ECReconf*. D'abord, nous présentons l'interface graphique du modèle d'exécution avec ses trois couches (Figure 6.19).

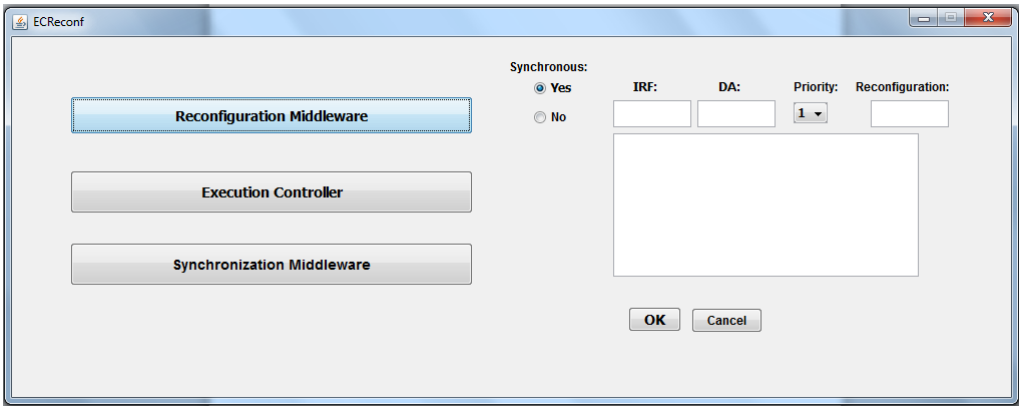

FIGURE 6.19 – Interface du modèle d'exécution.

Ensuite, nous montrons la simulation de la coordination et de la communication entre les différents composants RA2DL du réseau. La Figure 6.20 présente un exemple de coordination et de reconfiguration à la fois, le composant $RA2DL-coordinator$ applique la reconfiguration pour minimiser la taille du paquet Se. Cette reconfiguration est appliquée au niveau du composant $RA2DL-sender$ au moment de l'envoi afin de compresser le paquet pour avoir une taille plus petite Sr à la réception par le composant $RA2DL-receiver$.

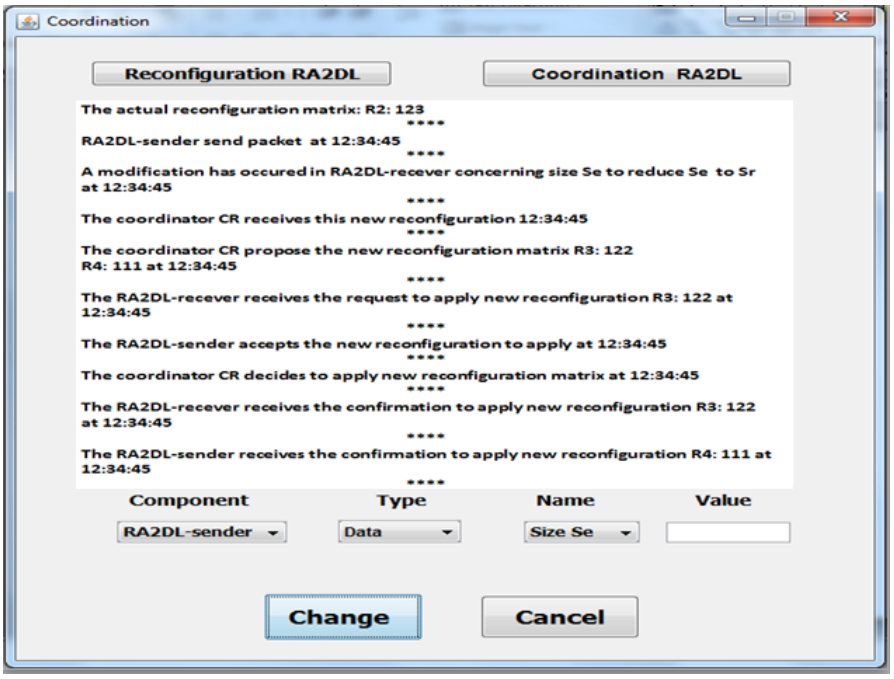

FIGURE 6.20 – Exemple de coordination et de reconfiguration.

6.1.3 Système de surveillance du corps

Au cours des dernières années, il y a eu une augmentation significative du nombre et de la variété des dispositifs portables de surveillance de la santé tels que les moniteurs d'impulsions simples, moniteurs d'activité et moniteurs Holter portables, aux capteurs implantables qui sont sophistiqués et coûteux. Le système de surveillance du corps (The Body-Monitoring System (BMS)) [HSS03] (Figure 6.21) est désigné comme un dispositif mobile qui est capable de collecter des données mesurées et de réagir selon les instructions établies par un superviseur. Le système est constitué d'un réseau de surveillance de corps. Afin de reconnaître l'état de la personne surveillée, l'unité du moniteur est connecté aux différents capteurs du corps et aux dispositifs d'entrée/sortie en utilisant soit des techniques de communication filaire ou sans fil. Les données provenant de tous les capteurs sont collectées, stockées et analysées en temps réel, en fonction de l'analyse des actions qui peut être effectuée. Un ordinateur est utilisé comme une interface entre le réseau de surveillance du corps, et un logiciel spécifique permettant à un superviseur de configurer l'unité de surveillance de la personne surveillée pour connecter les capteurs et les dispositifs d'entrée

/ sortie.

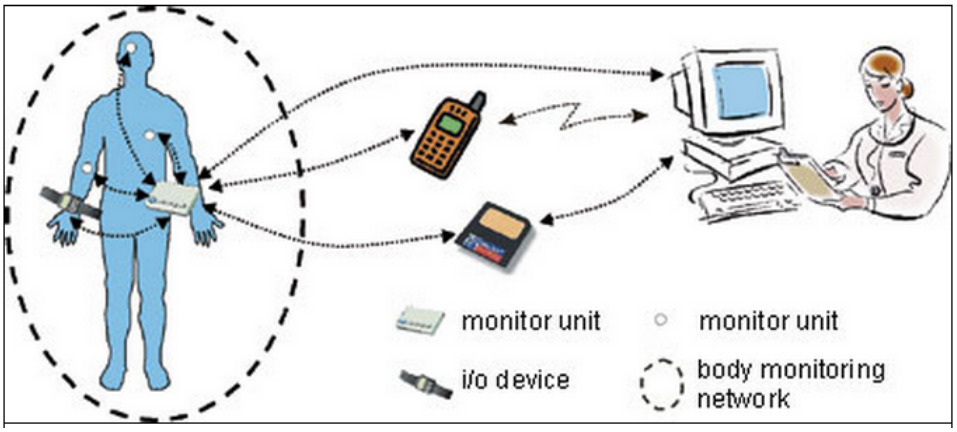

FIGURE 6.21 – Système de surveillance du corps [Bie02].

L'unité de surveillance se compose d'un module de communication responsable de la connexion qui contrôle les capteurs, et collecte le pré-traitement des données mesurées, un module de stockage permet de stocker les données collectées, et un module d'interprétation pour contrôler le comportement de l'unité de surveillance conformément aux instructions définies par un superviseur.

Pour sécuriser le système de surveillance du corps, nous devons tenir compte de ces étapes :

i) Regrouper dans RA2DL-Pool, les composants RA2DL en fonction des caractéristiques similaires,

ii) Attribuer pour chaque RA2DL-Pool un niveau de sécurité (en fonction du degré d'importance des composants RA2DL qu'ils contiennent),

iii) Affecter à chaque RA2DL-Pool un mécanisme de sécurité.

Nous regroupons les composants RA2DL du système BMS sur cinq RA2DL-Pools comme le montre le diagramme d'objets de la Figure 6.22 :

i) **RA2DL-Pool 1 :** comprend les composants RA2DL suivants : *RA2DL-G* pour la détection du glucose, *RA2DL-C* pour la détection de chlorure et *RA2DL-W* pour la détection de l'eau,

ii) **RA2DL-Pool 2 :** inclut les composants RA2DL suivants : *RA2DL-L* pour la dé-

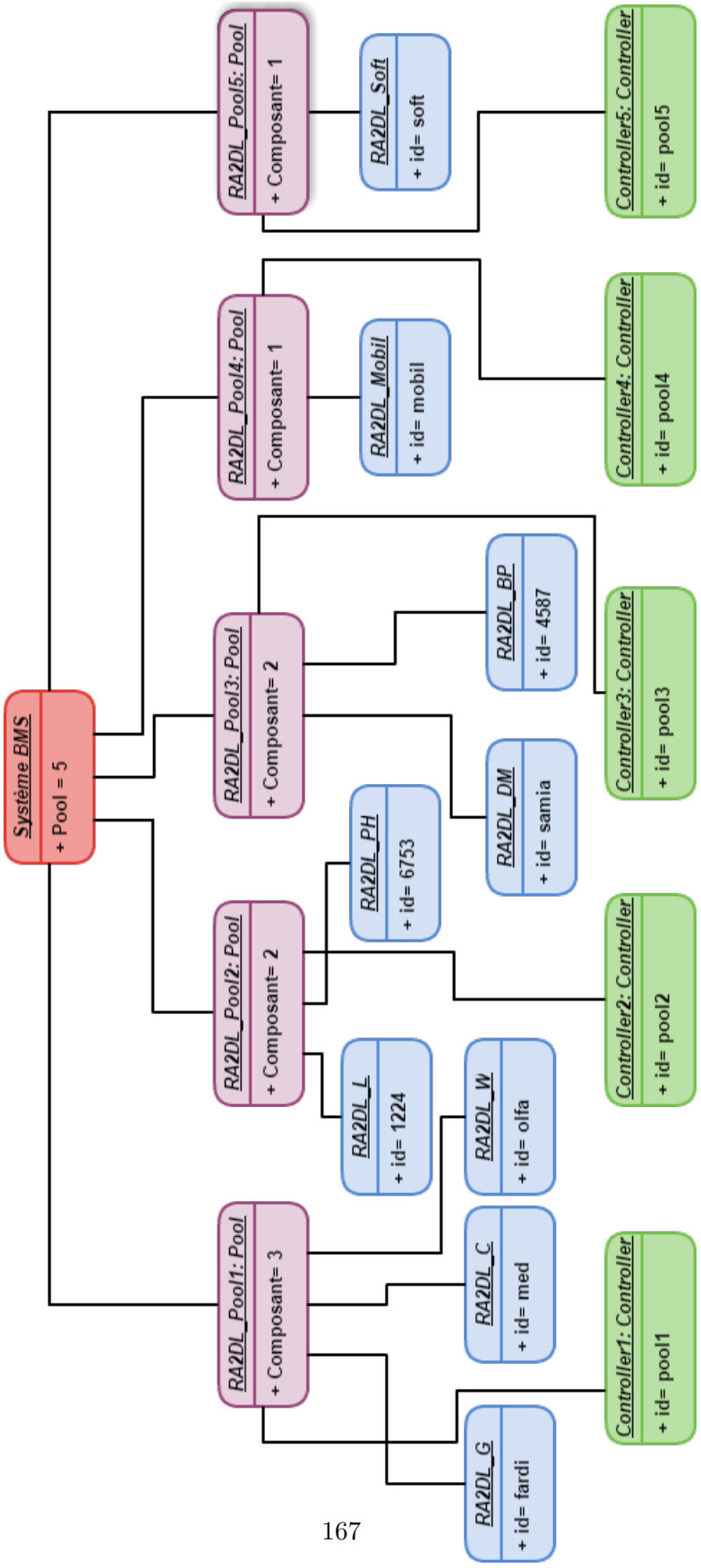

FIGURE 6.22 – Diagramme d'objet du système BMS.

tection de lactate et *RA2DL-PH* pour la détection de PH,

iii) **RA2DL-Pool 3 :** contient les composants RA2DL suivants : *RA2DL-DM* pour la détection du diabète et *RA2DL-BP* pour la pression artérielle,

iv) **RA2DL-Pool 4 :** contient le dispositif d'affichage qui est le composant *RA2DL-Mobil*,

v) **RA2DL-Pool 5 :** contient *RA2DL-Soft* pour la transmission de données avec un protocole jusqu'à *RA2DL-Mobil*.

Nous implémentons un module au niveau de l'outil $RA2DL-tool$ pour regrouper les composants RA2DL dans des conteneurs $RA2DL-Pool$ afin de sécuriser notre système. Nous supposons cinq Pools avec leurs paramètres tels que le nombre de composants RA2DL dans chaque Pool, Worst Case Execution Time (WCET), et les mécanismes d'authentification et de contrôle d'accès.

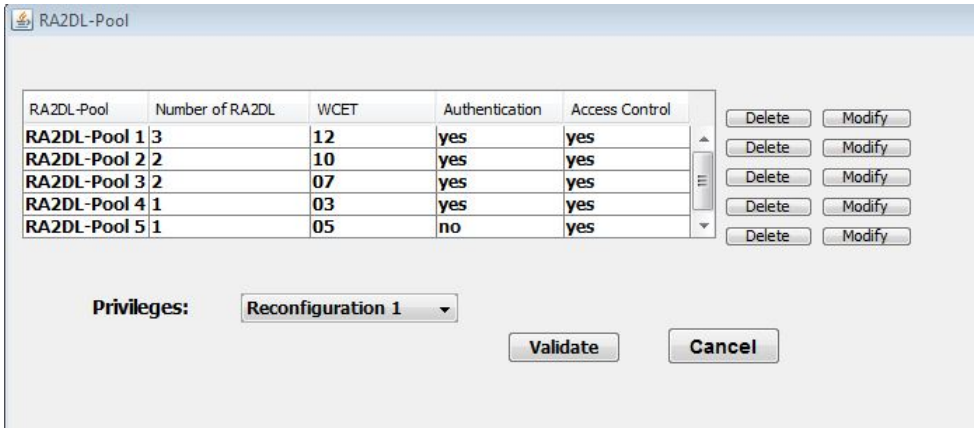

FIGURE 6.23 – RA2DL -Pool du système BMS.

La Figure 6.24 montre le test de connectivité des différents Pools selon les mécanismes d'authentification et la vérification de la reconfiguration entre les différents composants RA2DL dans chaque Pool.

L'application de notre approche pour sécuriser le système BMS est illustrée dans le tableau 6.4, où nous donnons un niveau de sécurité (S.L) pour les cinq Pools en fonction de la sensibilité des composants existants. Dans le système BMS, $RA2DL-Pool3$ est le plus sécurisé et $RA2DL-pool1$ est le moins sécurisé. Les deux mécanismes de sécurité

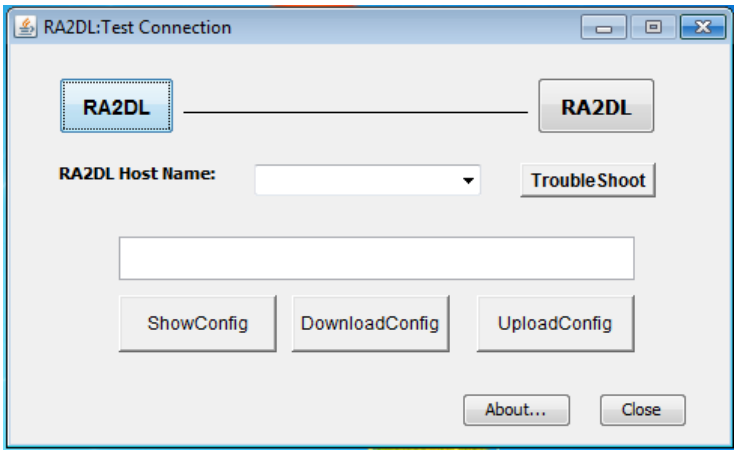

FIGURE 6.24 – Test de mécanisme d'authentification.

sont appliqués aux cinq Pools.

Pool	Niveau de sécurité (S.L)	Authentification	Contrôle d'accès	Sécurité
RA2DL-Pool 1	1	Non	Oui	Oui
RA2DL-Pool 2	2	Oui	Non	Oui
RA2DL-Pool 3	6	Oui	Oui	Oui
RA2DL-Pool 4	5	Oui	Oui	Oui
RA2DL-Pool 5	5	Oui	Oui	Oui

TABLE 6.4 – Exemple d'exécution.

6.2 Évaluation de RA2DL et synthèse

Dans cette section nous présentons les résultats en termes de reconfiguration et d'exécution des études de cas avec les composants RA2DL. Nous présentons aussi les différentes statistiques et évaluations des gains en termes de temps d'exécution. Ensuite nous présentons une synthèse qui synthétise l'évaluation de nos contributions et expérimentations de cette thèse.

6.2.1 Évaluation de RA2DL

À partir d'un composant AADL, nous avons utilisé l'outil $RA2DL-Tool$ pour produire un ensemble de composants RA2DL reconfigurables et flexibles. Ces composants exécutent des scénarios de reconfigurations dédiés à chaque module du composant. Nous donnons

ci-après les résultats obtenus avec la méthodologie proposée :

* Les composants AADL deviennent automatiquement reconfigurables et flexibles aux changements de l'environnement d'exécution au moment de l'exécution,

* Une nouvelle architecture RA2DL flexible qui remédie aux nombreuses limitations de reconfiguration est mise en place selon trois formes de reconfiguration : architecturale, composition et donnée,

* Les reconfigurations sont distribuées avec des composants RA2DL synchrones au niveau d'une architecture distribuée,

* RA2DL répond au besoin légitime de sécurité d'exécution.

La Figure 6.25 montre que grâce aux composants RA2DL du système radar, nous arrivons à maximiser le nombre d'objets détectés en garantissant une bonne qualité de détection. Ainsi, le radar fonctionne dans toutes les conditions climatiques et qu'il accepte tous formes de reconfigurations demandées.

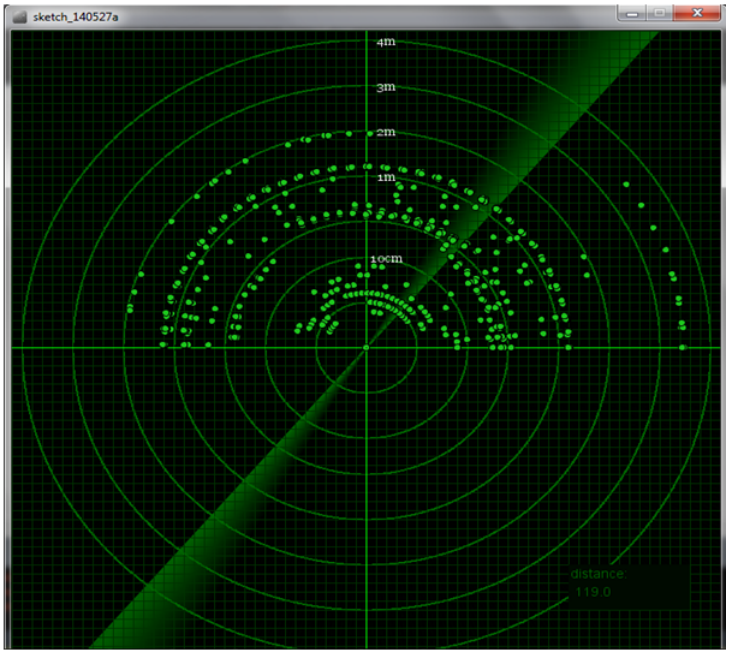

FIGURE 6.25 – Objets détectés du système radar.

Nous présentons par la suite les mesures que nous avons effectuées afin d'évaluer le temps d'exécution moyen correspondant à chaque des politiques de reconfiguration que

nous avons présentées précédemme

Nous montrons dans la Figure 6.26 la comparaison du gain de temps d'exécution entre le nouveau composant RA2DL avec deux autres composants AADL classiques (AADL1 et AADL2) [HS09] au niveau du système radar. Par la suite, nous ajoutons quelques événements de reconfigurations et nous calculons le temps d'exécution. Nous notons qu'avec le composant RA2DL, nous gagnons en terme de temps d'exécution. Ce temps moyen d'exécution devrait dépendre fortement de la charge du processeur de l'application en cours de reconfiguration. Nous avons donc simulé la charge du processeur en ajoutant dans le code fonctionnel des composants AADL des instructions qui permettent d'occuper le processeur pendant un temps d'exécution. Par ailleurs, nous stimulons les mécanismes de reconfiguration du composant RA2DL par l'envoi à chaque 100 millisecondes un scénario de reconfiguration pour reconfigurer son architecture, structure ou donnée. Cette solution est également efficace pour maintenir un minimum de temps d'exécution après l'application des scénarios de reconfigurations (25 scénarios). Les allures de la courbe 6.26 dans le cas expérimental et théorique sont très similaires.

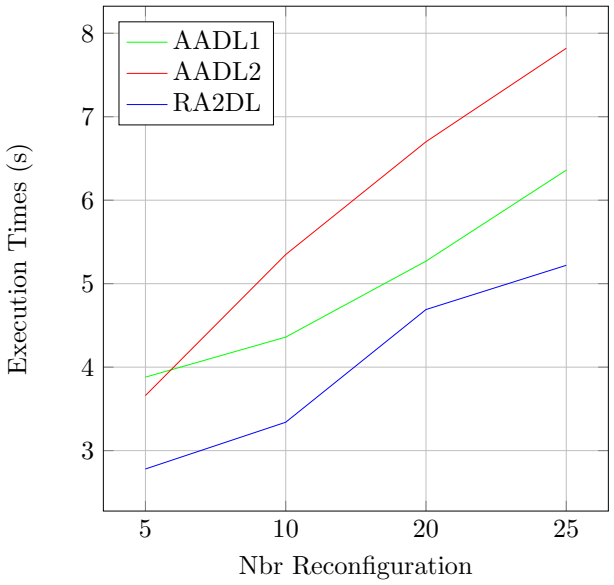

FIGURE 6.26 – Comparaison des temps d'exécution entre (RA2DL) et (AADL1 et AADL2).

.

Le résultat de la simulation exposée dans la Figure 6.27 montre qu'après les corrections

de l'exécution de RA2DL, l'utilisation du nouveau composant RA2DL pour l'application du réseau IEEE 802.11 Wireless LAN avec les corrections apportés par les extensions proposées au niveau du chapitre 4 comme la gestion des reconfigurations distribuées et synchrone et l'exécution des fonctionnalités du composant RA2DL devient plus performante et correcte à l'exécution qu'avant (avant la correction). Ce résultat permet de confirmer que nos expérimentations fournissent des résultats conformes aux attentes théoriques du chapitre 4.

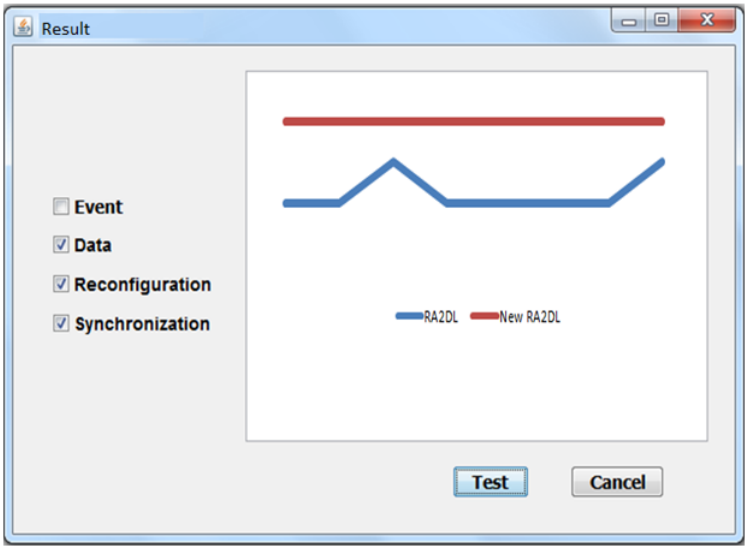

FIGURE 6.27 – Résultat de la simulation du modèle d'exécution de RA2DL.

Dix évaluations sont appliquées aux deux mécanismes qui sont répartis sur deux volets : le premier un composant RA2DL sans Pool et le deuxième le composant RA2DL inclue dans un $RA2DL - Pool$ du système BMS. On mesure ici le temps qui sépare la réception d'un scénario de reconfiguration et la réception de la réponse correspondante. Nous montrons dans la Figure 6.28, les résultats de l'évaluation. Nous nous sommes intéressés aux gains de temps de réponse pour les composants RA2DL sécurisés (avec RA2DL-Pool) et les composants RA2DL non sécurisés (sans RA2DL-Pool). Nous obtenons donc un temps de réponse avec RA2DL-Pool inférieur au temps des composants sans RA2DL-Pool. Ces résultats montrent le déterminisme des mécanismes de sécurité pour minimiser le temps de réponse.

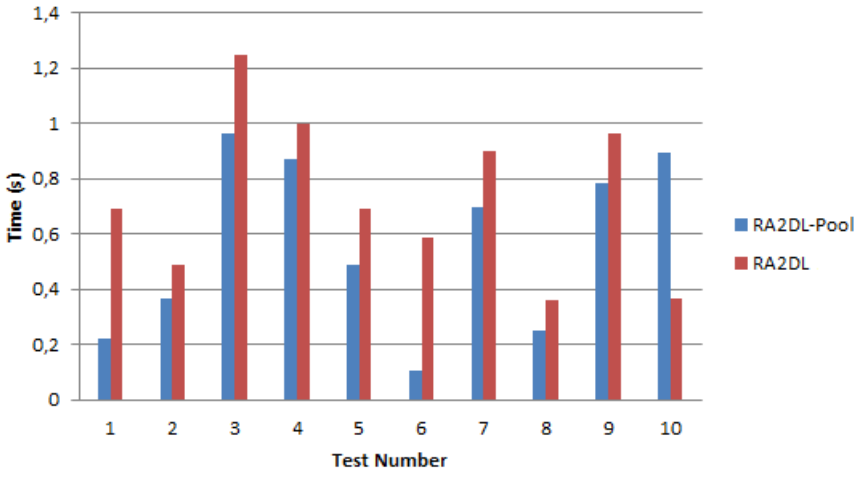

FIGURE 6.28 – Évaluation du temps de RA2DL sans et avec Pool du système BMS.

6.2.2 Synthèse

Nous exposons dans cette sous-section les gains apportés par cette thèse et les résultats expérimentaux en termes de performance, de productivité et l'amélioration de la qualité des systèmes de contrôle industriel en fonction des composants RA2DL :

- Augmentation de la productivité : Les résultats que nous avons présentés dans ce chapitre confirment la validité de notre méthode de reconfiguration RA2DL. Nous avons ainsi montré qu'il est possible d'augmenter la productivité du système de contrôle industriel, rappelons que la politique de reconfiguration et de sécurité respecte les règles de réalisation des systèmes de contrôles industriels.

- Amélioration de la qualité des systèmes de contrôles industriels : Les résultats présentés dans ce chapitre ont également permis de montrer l'analysabilité des spécifications de RA2DL, aussi bien du point de vue sécurité et distribution. Outiller l'analyse du comportement d'un composant RA2DL et du système de contrôle industriel devient beaucoup plus facile qu'avant, et peut ainsi être utilisé en cours de réalisation du système. Ceci permet d'augmenter fortement la qualité des résultats produits, puisque le comportement de celui-ci est vérifié. Les résultats numériques obtenus autour de la réalisation des études de cas, en termes de temps d'exécution, montrent la validité de notre implémentation de l'outil $RA2DL - Tool$ de reconfiguration et illustrent l'utilité de disposer de ce outil.

En conclusion de cette section, l'ensemble des résultats empiriques et expérimentaux que nous avons présentés dans cette thèse confirment que notre contribution répond à l'objectif que nous nous étions fixé au début : fournir des composants RA2DL reconfigurables à partir des composants AADL qui améliorent l'efficacité des équipes de développement industriel en matière de qualité et de productivité.

Conclusion

Dans ce chapitre, nous avons présenté notre suite d'outils $RA2DL - Tool$ qui permet de réaliser l'ensemble des fonctionnalités d'un composant RA2DL développé en Java. $RA2DL - Tool$ possède une architecture similaire à un compilateur moderne et permet à partir d'un modèle AADL de produire automatiquement du code vers un composant RA2DL reconfigurable ou bien de transformer les modèles AADL vers des formalismes destinés à la reconfiguration, sécurité, vérification et génération de code d'un composant flexible.

Nous avons ensuite présenté l'application de notre approche sur trois études de cas déférentes pour montrer l'appui du composant RA2DL sur les divers domaines. Nous avons présenté également, les résultats expérimentaux de la méthodologie proposée.

Nous avons pris le système radar comme une étude de cas classique du domaine embarqué et nous l'avons adopté aux composants RA2DL pour le rendre reconfigurable. Nous avons enrichi RA2DL par un modèle d'exécution en tenant compte du concept de synchronisation et de distribution que nous appliquons sur un réseau sans fil sous de la norme IEEE 802.11. Nous avons pu consolider RA2DL par une politique de sécurité ($RA2DL - Pool$) pour garantir la sécurité de fonctionnement d'un composant RA2DL, cette politique a été vérifiée également sur un système de surveillance du corps.

Nous avons donné des statistiques et des mesures sur le fonctionnement de RA2DL en termes de gain de temps d'exécution et de la fiabilité pour les trois études de cas proposées. En se basant sur ces études de cas, nous avons mis en évidence l'apport des différentes approches proposées dans cette thèse. Le premier apport concerne la flexibilité et la reconfiguration des composants AADL. Le deuxième concerne l'applicabilité de RA2DL

pour les systèmes adaptatifs pour avoir un système flexible et fiable. Ceci nous permet de dire que nous avons accompli les objectifs de notre travail de thèse qui consistent à construire automatiquement des composants RA2DL reconfigurables dédiés aux besoins des systèmes industriels adaptatifs.

Quatrième partie

Conclusion et perspectives

Chapitre 7

Conclusion et perspectives

Dans les chapitres précédents, nous avons présenté la problématique de notre travail de thèse : reconfiguration dynamique-automatique d'un système industriel adaptatif à base de composants AADL. Ensuite, nous avons proposé et implanté des solutions concrètes pour résoudre cette problématique. Pour appliquer ces solutions, nous avons présenté trois études de cas. Dans ce chapitre, nous rappelons les contributions de ce travail de thèse. Ainsi, nous présentons les résultats de ces contributions permettant de produire des systèmes reconfigurables à base des composants RA2DL. Enfin, nous présentons les perspectives pour étendre et poursuivre ce travail sur différents axes de recherche.

7.1 Contributions

Afin d'exposer d'une façon efficace les caractéristiques des approches par composants, nous présentons dans cette thèse les composants off-line et ceux de on-line avec les caractéristiques et les concepts de chaque approche. Nous avons opté pour l'utilisation des composants de type off-line et plus particulièrement les composants AADL pour reconfigurer un système de contrôle embarqué.

La reconfiguration est une opération souvent utile permettant le changement du comportement des composants logiciels et matériels afin de les adapter dynamiquement à l'environnement d'exécution. Notre objectif est d'avoir des composants flexibles qui peuvent encoder différents comportements possibles suite à l'application des scénarios de reconfiguration. Dans ce cas, un composant logiciel ou matériel sera la superposition de plusieurs

sous-composants de même nature mais avec différentes configurations ou caractéristiques au niveau des interfaces ou des implémentations.

Dans **le chapitre 3**, nous avons spécifié soigneusement les formes de reconfigurations possibles qu'on peut effectuer avec un composant RA2DL réparties sous trois formes : architecture, composition et données. De plus, nous avons proposé une architecture de RA2DL qui répond aux trois formes de reconfiguration. Nous avons modélisé et vérifié tous les comportements possibles du composant RA2DL en utilisant le modèle checker UPPAAL.

Dans **le chapitre 4**, nous avons considéré la distribution des composants RA2DL sur plusieurs architectures distribuées. Cette architecture reçoit des stimulus flux de reconfigurations de ces composants. Pour respecter les contraintes d'interaction entre les RA2DL, nous avons proposé une approche qui gère la coordination et la communication entre les composants en utilisant un composant de coordination qui possède une matrice de coordination. De plus, nous avons considéré la correction de l'exécution d'un composant RA2DL à l'aide de trois couches : la première pour gérer les flux de reconfigurations, la deuxième pour contrôler l'exécution et appliquer la reconfiguration et la troisième traite des reconfigurations synchrones entre les composants RA2DL.

Dans **le chapitre 5**, nous avons consolidé RA2DL à l'aide d'une méthode garantissant sa sécurité. Nous avons proposé une approche qui regroupe les composants RA2DL sous formes de conteneurs *Pool* selon un facteur de similarité entre les composants. De plus, nous avons combiné les pools avec deux mécanismes de sécurité, un mécanisme d'authentification pour garantir le processus de confirmation d'une identité entre les pools et un mécanisme de contrôle d'accès pour sécuriser et gérer les demandes de ressources des pools.

Dans **le chapitre 6**, nous avons traité trois études de cas différentes afin d'appliquer l'approche proposée. La première étude de cas concerne un système radar composé d'un ensemble des composants RA2DL où chaque composant applique un ensemble de scénarios de reconfiguration selon les trois modes d'architecture, composition et données. Nous avons eu recours à une deuxième étude de cas IEEE 802.11 Wireless LAN pour montrer la coordination entre les composants RA2DL en utilisant le composant de coordination avec la matrice de coordination. La troisième étude de cas concerne un système de surveillance

du corps que nous avons appliqué pour mettre en évidence la sécurité de RA2DL avec le concept de Pool. Les trois études de cas proposées montrent l'utilité de l'approche proposée en termes de temps d'exécution, de flexibilité et de sécurité.

Durant cette thèse, nous avons développé un environnement logiciel graphique RA2DL complet permettant la spécification à l'aide des automates temporisés, la vérification des propriétés en s'appuyant sur la logique TCTL (Timed Compotational Tree Logic) à l'aide du modèle checker UPPAAL, la conception des différents composants logiciels et matériels RA2DL d'un système reconfigurable avec UML. Cet environnement permet la simulation de cette architecture pour visualiser et tester dynamiquement des scénarios de reconfigurations.

7.2 Synthèse

Cette thèse a présenté la mise en application de notre méthode pour la reconfiguration dynamique et automatique des composants AADL pour les systèmes de contrôles industriels. Nous avons illustré celle-ci à l'aide de trois cas d'études exprimant les caractéristiques des composants AADL reconfigurables.

Après une première analyse préliminaire du premier cas d'étude d'un système radar à base des composants AADL, nous avons pu appliquer notre méthode de reconfiguration et mener des améliorations importantes sur le processus de fonctionnement du système. Notre méthode automatique de reconfiguration nous a permis de détecter une anomalie de reconfiguration du système au moment de son fonctionnement. La correction de cette anomalie à partir des spécifications initiales de l'utilisateur nous a permis d'obtenir un modèle complet et de valider le système. A partir de ce dernier, nous avons reconfiguré les composants du système respectant la flexibilité de l'architecture d'un composant RA2DL. Puis, nous avons exécuté, vérifié et analysé les nouveaux comportements du système avec succès.

Une fois que le composant RA2DL est spécifié, la reconfiguration distribuée et la synchronisation entre des composants RA2DL sont utiles pour comprendre le fonctionnement d'une application distribuée. Cela nous a permis de vérifier et appliquer notre méthode sur

un cas d'étude d'un réseau IEEE 802.11 Wireless LAN formé par des nœuds distribués. Ceci permet d'augmenter fortement la qualité de transmission des trames dans le réseau, puisque le comportement de celui-ci est formellement reconfigurable.

Nous avons utilisé un système de surveillance du corps comme troisième cas d'étude pour tester et prouver le bon fonctionnement de notre approche de sécurité. Cette étude de cas a été choisi pour valider leur adéquation aux problématiques posées et pour valider les solutions que nous avons réalisées. Nous avons effectué des améliorations de sécurité sur les composants RA2DL pour sécuriser les applications modélisées en RA2DL.

Ceci nous permet de dire que nous avons accompli les objectifs que nous nous étions fixés durant nos travaux de thèse, qui consistent à construire automatiquement des composants RA2DL reconfigurables dédiés aux besoins des systèmes de contrôles industriels.

7.3 Perspectives

Malheureusement, nous n'avons pas pu développer toutes les fonctionnalités et les reconfigurations possibles d'un système adaptatif de contrôle industriel à base de composants RA2DL. Ces manques viennent principalement du fait que nous n'avons pas eu le temps de poser et réaliser l'ensemble des caractéristiques de notre contribution. Ainsi, des approches supplémentaires pourraient être réalisées pour renforcer le bon fonctionnement d'un composant RA2DL ou un système à base de RA2DL. Comme guise de perspectives nous proposons :

Amélioration de RA2DL : Des travaux sont en cours pour supporter la nouvelle version d'un composant RA2DL amélioré pour offrir plusieurs corrections touchant la syntaxe et la sémantique de RA2DL selon les nouvelles approches par composants comme les composant RA2DL virtuelles. C'est un composant qui permet de raffiner la reconfiguration d'une architecture matérielle ou logicielle de façon virtuelle.

Solution pour d'autres montages : la réutilisation de RA2DL pour d'autres montages afin de tester la mise en œuvre et la reconfiguration des composants dans le contexte des systèmes embarqués. Ce qui peut servir à l'évaluation des techniques visant à automatiser la reconfiguration des composants RA2DL en amont du processus de développement.

Cette solution peut être utilisée pour faciliter la mise en œuvre de l'ensemble de l'approche, depuis la spécification de la reconfiguration possible jusqu'à l'implantation finale du système.

Optimisation des modèles RA2DL : les approches de cette thèse ont permis de produire un composant RA2DL selon les modèles des composants AADL. Cependant, le modèle RA2DL lui-même devrait être optimisé en fonction du besoin d'adaptation de l'architecture. Il s'agit de fusionner les composants RA2DL, ceux ayant les mêmes contraintes de reconfiguration en un seul composant afin d'optimiser le nombre de composants et de reconfiguration et de verrous logiciels et matériels dans le système. L'implantation de ces optimisations par des méthodes d'optimisation stochastiques peut être une nouvelle proposition d'un sujet de thèse.

Bibliographie

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *Proceedings., Fifth Annual IEEE Symposium on eLogic in Computer Science, 1990. LICS'90,*, pages 414–425. IEEE, 1990.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1) :2–34, May 1993.

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2) :183–235, 1994.

[AEK06] Balal Ahmad, Ahmet T Erdogan, and Sami Khawam. Architecture of a dynamically reconfigurable noc for adaptive reconfigurable mpsoc. In *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, pages 405–411. IEEE, 2006.

[AfHOT06] Jim Alves-foss, W. Scott Harrison, Paul Oman, and Carol Taylor. The mils architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, 2 :239–247, 2006.

[AG94] Robert Allen and David Garlan. Formalizing architectural connection. In *Proceedings of the Sixteenth International Conference on Software Engineering*, pages 71–80, Sorrento, Italy, May 1994.

[AL04] AS-2C Architecture Analysis and Design Language. *Architecture Analysis & Design Language (standard SAE AS5506),*. Society of Automotive Engineers (SAE), September 2004.

[All97] Robert Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

[AMKB15a] Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, and Samia Bouzefrane. New solutions for useful execution models of communicating adaptive ra2dl. In *14th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMET'2015)*, volume 532 of *Communications in Computer and Information Science*, pages 87–101, Naples, Italy, September 2015.

[AMKB15b] Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, and Samia Bouzefrane. Ra2dl : New flexible solution for adaptive aadl-based control components. In *Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems. PECCS 2015*, pages 247–258, Angers, Loire Valley, France, February 2015.

[AMKB15c] Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, and Samia Bouzefrane. Ra2dl : New reconfigurable aadl component. In *Journées Doctorales de l'Ecole Polytechnique de Tunisie (JDEPT' 15)*, La Marsa, Tunisia, April 2015.

[AMKB16] Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, and Samia Bouzefrane. Ra2dl-pool : New useful solution to handle security of reconfigurable embedded systems. In *Proceedings of the 11th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2016)*, pages 102–111, Rome, Italia, April 2016.

[Aoy98] Mikio Aoyama. New age of software development : How component-based software engineering changes the way of software development ? In *1st Workshop on Component Based Software Engineering*, 1998.

[ASC+13] Rabie Ben Atitallah, Eric Senn, Daniel Chillet, Mickael Lanoe, and Dominique Blouin. An efficient framework for power-aware design of heteroge-

neous mpsoc. *IEEE Transactions on Industrial Informatics*, 9(1) :487–501, 2013.

[ASM05a] Christo Angelov, Krzysztof Sierszecki, and Nicolae Marian. Design models for reusable and reconfigurable state machines. In *in L.T. Yang et al. (Eds.) : Proc. of EUC 2005, LNCS 3824*, pages 152–163, 2005.

[ASM05b] Christo Angelov, Krzysztof Sierszecki, and Nicolae Marian. *Design Models for Reusable and Reconfigurable State Machines*, pages 152–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[AV10] Yazen Alsafi and Valeriy Vyatkin. Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. *Robotics and Computer-Integrated Manufacturing*, 26(4) :381 – 391, 2010.

[BBF+13] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and software verification : model-checking techniques and tools*. Springer Science & Business Media, 2013.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, pages 3–12. Ieee, 2006.

[BCL+06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java. *Software : Practice and Experience*, 36(11-12) :1257–1284, 2006.

[BDF+11] Benoit Badrignans, Jean-Luc Danger, Viktor Fischer, Guy Gogniat, and Lionel Torres. *Security Trends for FPGAS : From Secured to Secure Reconfigurable Systems*. Springer Science & Business Media, 2011.

[BEN11] Malika BENAMMAR. Une approche basée architecture pour la spécification formelle des systèmes embarqués. Master's thesis, Mentouri University Constantine Algeria, 2011.

[BF+07] J Bodeveix, M Fllali, et al. The aadl behavior annex-experiments and roadmap. In *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems, IEEE Computer Society, Washington, DC*, pages 377–382, 2007.

[BFHP09] Etienne Borde, Peter H. Feiler, Grégory Haïk, and Laurent Pautet. Model driven code generation for critical and adaptative embedded systems. *SIGBED Rev.*, 6(3) :10 :1–10 :5, October 2009.

[BFJLT02] Benoit Baudry, Franck Fleurey, Jean-Marc Jézéquel, and Yves Le Traon. Automatic test case optimization using a bacteriological adaptation model : application to .net components. In *17th IEEE International Conference on Automated Software Engineering, 2002.*, pages 253–256, Edinburgh, United Kngdm, September 2002.

[Bie02] Mria Bielikov. A body-monitoring system with eeg and eog sensors. *Journal of ERCIM News No. 49*, October 2002.

[Bor09] Etienne Borde. *Configuration and Reconfiguration of Critical and Distributed Real-Time Embedded Systems.* Theses, Télécom ParisTech, December 2009.

[Bre02] Martyn. Norrie Douglas H. Brennan, Robert W. Fletcher. A holonic approach to reconfiguring real-time distributed control systems. In Vladimír Mařík, Olga Štěpánková, Hana Krautwurmová, and Michael Luck, editors, *In Multi-Agent Systems and Applications : MASA01.*, pages 323–335, Berlin, Heidelberg, 2002. Berlin Heidelberg.

[Car02] John S Carson. Model verification and validation. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 1, pages 52–58. IEEE, 2002.

[CCG98] Federico Cali, Marco Conti, and Enrico Gregori. Ieee 802.11 wireless lan : capacity analysis and protocol enhancement. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communica-*

tions Societies. Proceedings. IEEE, volume 1, pages 142–149 vol.1, Mar 1998.

[CCM05] Paolo Costa, Geoff Coulson, and Cecilia Mascolo. The runes middleware : A reconfigurable component-based approach to networked embedded systems. In *In Proc. of 16 th International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC05*, pages 11–14. IEEE Press, 2005.

[CDF+98] Christopher F Codella, Donna N Dillenberger, Donald Francis Ferguson, R D Jackson, Thomas A Mikalsen, and Ignacio Silva-Lepe. Support for enterprise javabeans in component broker. *IBM Systems Journal*, 37(4) :502–538, Octobre 1998.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.

[CE00] Krzysztof Czarnecki and Ulrich W Eisenecker. Generative programming. *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen*, page 15, 2000.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, April 1986.

[Chk10] Mohamed Yassin Chkouri. *Modelling real-time embedded systems using AADL for the automatic generation of applications formally verified.* Theses, Université Joseph-Fourier - Grenoble I, April 2010.

[CLWK00] Xia Cai, M. R. Lyu, Kam-Fai Wong, and Roy Ko. Component-based software engineering : technologies, development frameworks, and quality assurance schemes. In *Proceedings. Seventh Asia-Pacific Software Engineering Conference, APSEC*, pages 372–379, 2000.

[Cor93] Microsoft Corporation. The component object model, 1993.

[Crn02] Ivica Crnkovic. *Building Reliable Component-Based Software Systems*, volume 1580533272. Artech House, Inc., Norwood, MA, USA, 2002.

[Crn04] Ivica Crnkovic. Component-based approach for embedded systems. In *Ninth International Workshop on Component-Oriented Programming (WCOP)*, Oslo, Norway, 2004.

[CSLS99] Domenico Cavaliere, Françoise Simonot-Lion, and Ye-Qiong Song. Modélisation d'architectures d'applications - moyens et formalismes. Interne 99-R-217 || cavaliere99a, 1999. Rapport interne.

[DF14] Julien Delange and Peter Feiler. Architecture fault modeling with the aadl error-model annex. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 361–368, Aug 2014.

[Dis04] Pierre. Dissaux. Using the aadl for mission critical software development. In *2nd European Congress ERTS, Embedded Real Time Software*, Toulouse- France, 2004.

[DMR+14] Pierre Dissaux, Olivier Marc, Stéphane Rubini, Christian Fotsing, Vincent Gaudel, Frank Singhoff, Alain Plantec, Vuong Nguyen-Hong, and Hai-Nam Tran. The smart project : Multi-agent scheduling simulation of real-time architectures. In *Embedded Real Time Software and Systems*, pages –, Toulouse, France, Feb 2014.

[DoCM97] Technology Department of Computing, Imperial College of Science and London SW7 2BZ UK. Medicine, 180 Queens Gate. The darwin language version 3d, September 1997.

[DPDP11] Rémi Delmas, Thomas Polacsek, David Doose, and Anthony Fernandes Pires. Idm : Vers une aide ? la conception. In *INFORSID*, pages 147–162, 2011.

[EJL+03] Johan Eker, Jörn W Janneck, Edward A Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming

heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1) :127–144, 2003.

[EK02] Wolfgang Emmerich and Nima Kaveh. Component technologies : Java beans, com, corba, rmi, ejb and the corba component model. In *Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002.*, pages 691–692. IEEE, 2002.

[FDB+08] Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jézéquel. Modeling and validating dynamic adaptation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 97–108. Springer, 2008.

[FG06] Peter H Feiler and Aaron Greenhouse. Osate plug-in development guide-. *CMU. Pittsburgh*, 2006.

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL : An Introduction to the SAE Architecture Analysis & Design Language.* Addison-Wesley Professional, 1st edition, 2012.

[FGH06] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl) : An introduction. Technical report, DTIC Document, 2006.

[FHS+06] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *IEEE Softw.*, 23(2) :62–70, March 2006.

[Fil02] Juliana Kuster Filipe. A logic-based formalization for component specification. *Journal of Object Technology*, 1(3) :231–248, 2002.

[Gar95] David Garlan. An introduction to the aesop system. *D. Garlan*, 11, 1995.

[GBKC16] Aymen Gammoudi, Adel Benzina, Mohamed Khalgui, and Daniel Chillet. Real-time scheduling of reconfigurable battery-powered multi-core platforms. In *28th IEEE International Conference on Tools with Artificial In-*

telligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016, pages 121–129, 2016.

[GC10] Laurent George and Pierre Courbin. Reconfiguration of uniprocessor sporadic real-time systems : the sensitivity approach. *IGI-Global Knowledge on Reconfigurable Embedded Control Systems : Applications for Flexibility and Agility*, pages 167–189, 2010.

[GKA12] Hamza Gharsellaoui, Mohamed Khalgui, and Samir Ben Ahmed. New optimal preemptively scheduling for real-time reconfigurable sporadic tasks based on earliest deadline first algorithm. *International Journal of Advanced Pervasive and Ubiquitous Computing (IJAPUC)*, 4(2) :65–81, 2012.

[GMW97a] David Garlan, Robert Monroe, and David Wile. Acme : An architecture description interchange language. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, pages 7–. IBM Press, 1997.

[GMW97b] David Garlan, Robert T. Monroe, and David Wile. Acme : An architecture description interchange language. In *Proceedings of CASCON'97*, pages 169–183, Toronto, Ontario, November 1997.

[Gro02] Michael Groh. Creating components in .net. PC Productivity Solutions, Microsoft, February 2002.

[GZJ16] Wafa Gabsi, Bechir Zalila, and Mohamed Jmaiel. Ema2aop : From the aadl error model annex to aspect language towards fault tolerant systems. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 155–162, June 2016.

[HAS06] Feiler Peter H, Lewis Bruce A, and Vestal Steve. The sae architecture analysis and design language (aadl) a standard for engineering performance critical systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applica-*

tions, 2006 IEEE International Symposium on Intelligent Control, pages 1206–1211, Munich, Germany, Oct 2006.

[HFM08] Jörgen Hansson, Peter H Feiler, and John Morley. Building secure systems using model-based engineering and architectural models. *CrossTalk : The Journal of Defense Software Engineering*, 21(9) :12, 2008.

[HMKC16] Wiem Housseyni, Olfa Mosbahi, Mohamed Khalgui, and Maryline Chetto. Real-time scheduling of reconfigurable distributed embedded systems with energy harvesting prediction. In *20th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2016, London, United Kingdom, September 21-23, 2016*, pages 145–152, 2016.

[HS09] Jèrome. Hugues and Frank Singhoff. Développement de systèmes à l'aide d'aadl- ocarina/cheddar. In *Tutoriel présenté à lécole d'été temps réel*, September 2009.

[HSS03] D. Husemann, R. Steinbugler, and B. Striemer. Body monitoring using local area wireless interfaces. Google Patents, Avril 2003. US Patent App. 10/406,865.

[HZPK08] Jerome Hugues, Bechir Zalila, Laurent Pautet, and Fabrice Kordon. From the prototype to the final embedded system using the ocarina aadl tool suite. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(4) :42, 2008.

[JÖ2] Jan Jürjens. Umlsec : Extending uml for secure systems development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, UK, 2002. Springer-Verlag.

[JBCG05] A. Joolia, T. Batista, G. Coulson, and A. T. A. Gomes. Mapping adl specifications to an efficient and reconfigurable runtime component platform. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 131–140, 2005.

[JSHP11] Haris Javaid, Muhammad Shafique, Jörg Henkel, and Sri Parameswaran. System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia. In *Proceedings of the International Conference on Computer-Aided Design*, pages 616–623. IEEE Press, 2011.

[KH08] M. Khalgui and H. M. Hanisch. Dynamic reconfiguration of two benchmark production systems. In *6th IEEE International Conference on Industrial Informatics 2008*, pages 307–314, July 2008.

[KH09] Mohamed Khalgui and Hans-Michael Hanisch. Reconfiguration of industrial embedded control systems. *Behavioral Modeling for Embedded Systems and Technologies : Applications for Design and Implementation : Applications for Design and Implementation*, page 318, 2009.

[KK06] Deuk Kyu Kum and Soo Dong Kim. A systematic method to generate .net components from mda/psm for pervasive service. In *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, pages 324–331, Seattle, WA, Aug 2006.

[KMLH11a] M. Khalgui, O. Mosbahi, Z. Li, and H. M. Hanisch. Reconfigurable multiagent embedded control systems : From modeling to implementation. *IEEE Transactions on Computers*, 60(4) :538–551, April 2011.

[KMLH11b] Mohamed Khalgui, Olfa Mosbahi, Zhiwu Li, and Hans-Michael Hanisch. Reconfiguration of distributed embedded-control systems. *IEEE/ASME Transactions on Mechatronics*, 16(4) :684–694, 2011.

[LBD02] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. Secureuml : A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, London, UK, UK, 2002. Springer-Verlag.

[LCD+00] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, and Jon Stockwood. Hardware-software co-design of embedded reconfigu-

rable architectures. In *Proceedings of the 37th Annual Design Automation Conference*, pages 507–512. ACM, 2000.

[LEPI10] Adrian Lifa, Petru Eles, Zebo Peng, and Viacheslav Izosimov. Hardware/software optimization of error detection implementation for real-time embedded systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 41–50, New York, NY, USA, 2010. ACM.

[LKA+95] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using rapide. *IEEE Transactions on Software Engineering*, 21(4) :336–354, Apr 1995.

[LMKT16] Wafa Lakhdhar, Rania Mzid, Mohamed Khalgui, and Nicolas Trèves. Milp-based approach for optimal implementation of reconfigurable real-time systems. In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1 : ICSOFT-EA, Lisbon, Portugal, July 24 - 26, 2016.*, pages 330–335, 2016.

[LPY97] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2) :134–152, 1997.

[LSO+07] Sten A Lundesgaard, Arnor Solberg, Jon Oldevik, Robert France, JanOyvind Aagedal, and Frank Eliassen. Construction and execution of adaptable applications using an aspect-oriented and model driven approach. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 76–89. Springer, 2007.

[LT07] Daniel Lüdtke and Dietmar Tutsch. Lossless static vs. dynamic reconfiguration of interconnection networks in parallel and distributed computer systems. In *Proceedings of the 2007 Summer Computer Simulation Conference*, SCSC '07, pages 717–724, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[Luc96] David C Luckham. Rapide : A language and toolset for simulation of distributed systems by partial orderings of events. Technical report, Stanford, CA, USA, 1996.

[Lud03] Frank. Luders. Adopting a software component model in real-time systems development. In *28th Annual NASA GoddardSoftware Engineering Workshop, 2003.*, pages 114–119, Greenbelt, MD, USA, Dec 2003.

[LZPH09] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérome Hugues. Ocarina : An environment for aadl models analysis and automatic code generation for high integrity applications. In *Reliable Software Technologies–Ada-Europe 2009*, pages 237–250. Springer, 2009.

[Mac05] Charles M Macal. Model verification and validation. In *Proceedings of the Workshop on Threat anticipation : social science methods and models, The University of Chicago and Argonne National Laboratory*, 2005.

[Mal08] Frédéric Mallet. Clock constraint specification language : specifying clock constraints with uml/marte. *Innovations in Systems and Software Engineering*, 4(3) :309–314, 2008.

[Mer01] Stephan Merz. *Modeling and Verification of Parallel Processes : 4th Summer School, MOVEP 2000 Nantes, France, June 19–23, 2000 Revised Tutorial Lectures*, chapter Model Checking : A Tutorial Overview, pages 3–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[MG00] Pierre-Alain Muller and Nathalie Gaertner. *Modélisation objet avec UML*, volume 514. Eyrolles Paris, 2000.

[Mic95] Microsoft. The component object model specification. *Microsoft Corporation and Digital Equipment Corporation*, Version 0.9, October 1995.

[MKS89] J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in conic. *IEEE Transactions on Software Engineering*, 15(6) :663–675, Jun 1989.

[MLCR08] J Marco Mendes, Paulo Leitão, Armando W Colombo, and Francisco Restivo. Service-oriented control architecture for reconfigurable production systems. In *6th IEEE international conference on industrial informatics*, pages 744–749. IEEE, 2008.

[MORT96] Nenad Medvidovic, Peyman Oreizy, Jason E. Robbins, and Richard N. Taylor. Using object-oriented typing to support architectural design in the c2 style. *SIGSOFT Softw. Eng. Notes*, 21(6) :24–32, October 1996.

[MRT99] Nenad Medvidovic, David S. Rosenblum, and Richard N. Taylor. A language and environment for architecture-based software development and evolution. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 44–53, New York, NY, USA, 1999. ACM.

[MSKf05] Sotiris Moschoyiannis, Michael W. Shields, and Juliana Kster-filipe. Formalising well-behaved components. In *In Proceedings Formal Aspects of Component Software (FACS), satellite workshop of FME03*, pages 121–142, 2005.

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1) :70–93, January 2000.

[Ngo08] Khanh Hieu Ngo. *Aide au développement de systèmes temps réel à laide dun langage graphique flots de donnèes.* PhD thesis, Université Poitiers, Novembre 2008.

[Pla12] Alain Plantec. Faciliter la vérification et la validation de méta-modèles dans le cadre de lingénierie dirigée par les modèles : une approche agile, outillée et orientée données. 2012.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on*

International Symposium on Programming, pages 337–351, London, UK, UK, 1982. Springer-Verlag.

[QWL14] Z. Quan, S. Wang, and Liu. Ima reconfiguration modeling and reliability analysis based on aadl. In *IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014*, pages 664–668, June 2014.

[RAC+02] Matthew J. Rutherford, Kenneth M. Anderson, Antonio Carzaniga, Dennis Heimbigner, and Alexander L. Wolf. Reconfiguration in the enterprise javabean component model. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment*, CD '02, pages 67–81, London, UK, UK, 2002. Springer-Verlag.

[RKK07] Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. Architecting dependable systems iv. chapter A System Dependability Modeling Framework Using AADL and GSPNs, pages 14–38. Springer-Verlag, Berlin, Heidelberg, 2007.

[RSS+07] Martijn N. Rooker, Christoph Sünder, Thomas Strasser, Alois Zoitl, Oliver Hummer, and Gerhard Ebenhofer. Zero downtime reconfiguration of distributed automation systems : The cedac approach. In *3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, HoloMAS '07, pages 326–337, Berlin, Heidelberg, 2007. Springer-Verlag.

[Saa11] Rodrigo Saad. *Parallel model checking for multiprocessor architecture.* PhD thesis, INSA de Toulouse, 2011.

[SAZ11] Dajiang Suo, Jinxia An, and Jihong Zhu. Aadl-based modeling and tpn-based verification of reconfiguration in integrated modular avionics. In *Proceedings of the 18th Asia-Pacific Software Engineering Conference*, APSEC '11, pages 266–273, Washington, DC, USA, 2011. IEEE Computer Society.

[Sch06] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2) :25, 2006.

[SE15] Mohammad Salehi and Alireza Ejlali. A hardware platform for evaluating low-energy multiprocessor embedded systems based on cots devices. *IEEE Transactions on Industrial Electronics*, 62(2) :1262–1269, 2015.

[Sif05] Joseph Sifakis. A framework for component-based construction. In *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*, pages 293–299, Sept 2005.

[SK04] Charles P. Shelton and Philip Koopman. Improving system dependability with functional alternatives. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN '04, pages 295–, Washington, DC, USA, 2004. IEEE Computer Society.

[SK06] Elisabeth A. Strunk and John C. Knight. Dependability through assured reconfiguration in embedded system software. *IEEE Trans. Dependable Secur. Comput.*, 3(3) :172–187, July 2006.

[SLM98] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The design of the tao real-time object request broker. *Comput. Commun.*, 21(4) :294–324, April 1998.

[SN92] Crane Stephen and Dulay Naranker. Constructing multi-user applications in rex. In *CompEuro '92 . 'Computer Systems and Software Engineering',Proceedings.*, pages 365–370, The Hague, Netherlands, May 1992.

[Szy02] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[TDF04] K. Thramboulidis, G. Doukas, and A. Frantzis. Towards an implementation model for fb-based reconfigurable distributed control applications. In *Proceedings. Seventh IEEE International Symposium on Object-Oriented*

Real-Time Distributed Computing, 2004., pages 193–200, Vienna, Austria, May 2004.

[TDH+04] Ben H Thacker, Scott W Doebling, Francois M Hemez, Mark C Anderson, Jason E Pepin, and Edward A Rodriguez. Concepts of model verification and validation. Technical report, Los Alamos National Lab., Los Alamos, NM (US), 2004.

[The04] Mike Thelwall. *Link analysis : An information science approach*. Elsevier Inc., 2004.

[Val00] Sklyarov Valery. Synthesis of control circuits with dynamically modifiable behavior on the basis of statically reconfigurable fpgas. In *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design*, SBCCI '00, pages 353–, Washington, DC, USA, 2000. IEEE Computer Society.

[VK00] Steve Vestal and Jonathan Krueger. Technical and historical overview of metah. Technical report, Honeywell Technology Center, January 2000. Minneapolis, MN 55418-1006.

[VOvdLKM00] Rob Van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *Computer*, 33(3) :78–85, March 2000.

[Wil06] Michael D. Wilder. The uniconc optimizing unicon compiler. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 756–757, New York, NY, USA, 2006. ACM.

[WKK+15] X. Wang, I. Khemaissia, M. Khalgui, Z. Li, O. Mosbahi, and M. Zhou. Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks. *IEEE Transactions on Automation Science and Engineering*, 12(1) :258–271, Jan 2015.

[Wol08] Wayne Wolf. *Computers As Components, Second Edition : Principles of*

Embedded Computing System Design. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2008.

[YLY07] Eun-Jun Yoon, Wan-Soo Lee, and Kee-Young Yoo. Secure pap-based RADIUS protocol in wireless networks. In *Proceedings of Third International Conference on Intelligent Computing, ICIC 2007, Qingdao, China, August 21-24, 2007.*, pages 689–694, 2007.

[Zal08] Bechir Zalila. *Configuration et déploiement d'applications temps-réel réparties embarquées à l'aide d'un langage de description d'architecture*. Theses, Télécom ParisTech, Nov 2008.

[ZFZ10] Wei Zuo, Jinfu Feng, and Jiaqiang Zhang. Model transformation from xuml pims to aadl psms. In *International Conference on Computing, Control and Industrial Engineering (CCIE), 2010*, volume 1, pages 54–57. IEEE, 2010.