République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Cycle de formation doctorale dans la discipline Ingénierie des Systèmes Informatiques
Bechir Zalila
Habilitation Universitaire
N° d'ordre: 06/2022

Habilitation Universitaire

présentée et soutenue publiquement le 28 mai 2022

pour l'obtention de l'

Habilitation Universitaire

dans la discipline : Ingénierie des Systèmes Informatiques

par

Bechir Zalila

La Modélisation Architecturale au Service des Systèmes Temps réel Répartis Embarqués - Contributions à la Configuration Dynamique, la Tolérance aux Pannes, l'Optimisation et la Vérification Formelle

Composition du jury

Président :	Mohamed Abid	Professeur, ENIS
Rapporteurs :	Rafik Bouaziz	Professeur Émérite, FSEGS
	Leila Ben Ayed	Professeur, ENSI
Examinateurs :	Khalil Drira	Directeur de Recherche, CNRS
	Mohamed Jmaiel	Professeur, CRNS

À mes parents, Mohamed et Radhia,
À Sahar et Eya.

© 2022 Bechir Zalila

Remerciements

Ce mémoire, fruit de plus d'une décennie de travaux de recherche, n'a pu voir le jour sans la contribution, l'aide et le support de personnes exemplaires. Cette partie est une modeste reconnaissance à ces personnes.

Je tiens tout d'abord à remercier Prof. Mohamed Jmaiel, directeur général du Centre de Recherche en Numérique de Sfax et chef du laboratoire de Recherche en Développement et Contrôle d'Applications Distribuées pour m'avoir accueilli dans son équipe alors que j'étais encore un jeune assistant venant de l'étranger et ne connaissant personne dans le paysage universitaire Tunisien. Son aide, ses conseils et l'opportunité qu'il m'a offerte en me permettant de participer à l'encadrement de thèses de doctorat et de mastères de recherche sous sa direction sont la cause directe des contributions objets de ce mémoire. Je le remercie également pour les conseils et les remarques durant la phase d'élaboration de ce manuscrit et pour avoir accepté d'être le parrain de cette habilitation universitaire.

Je tiens ensuite, à remercier M. Ahmed Hadj Kacem, professeur à la Faculté de Sciences Économiques et de Gestion de Sfax, pour les conseils et l'aide inestimables qu'il m'a fournis durant les dernières phases de cette habilitation et qui ont facilité l'aboutissement de ce travail.

Je remercie vivement M. Rafik Bouaziz, professeur émérite à la Faculté de Sciences Économiques et de Gestion de Sfax et Mme Leila Ben Ayed, professeur à l'École Nationale des Sciences de l'Informatique pour l'honneur qu'ils m'ont fait en acceptant d'être les rapporteurs de ce mémoire et pour les remarques pertinentes qu'ils m'ont données. Je remercie également M. Mohamed Abid, professeur à l'École Nationale d'Ingénieurs de Sfax et M. Khalil Drira, directeur de recherche au Centre National de la Recherche Scientifique (CNRS, France) pour avoir accepté de faire partie du jury de cette habilitation universitaire.

Les contributions décrites dans ce mémoire sont le fruit d'un travail d'une équipe exemplaire de doctorantes et d'étudiantes en mastère de recherche dont les travaux étaient d'une importance capitale pour la concrétisation des idées de départ. Je remercie profondément Mmes Fatma Krichen, Wafa Gabsi, Rahma Bouaziz, Hana Mkaouar, Sihem Loukil, Amal Gassara, Alvine Boyae Belle, Amal Ghorbel et Dorra Ktari pour la qualité exceptionnelle de leur participation à ces contributions.

Les thèses de doctorats auxquelles j'ai participé n'auraient pas l'impact souhaité sans le support de mes collègues et amis de plusieurs institutions d'enseignement supérieur étrangères, qui n'ont épargné aucun effort pour que ces travaux aboutissent aux résultats souhaités. Merci donc, de tout mon cœur à Jérôme Hugues, chercheur senior au Software Engineering Institute (Carnegie Mellon University) et à Frank Singhoff, professeur à l'Université de Bretagne Occidentale, pour leur aide et support.

Passer toutes ces années de travail au sein du Département de Génie Informatique et de Mathématiques Appliquées de l'ENIS et du laboratoire de recherche ReDCAD m'a fait connaître beaucoup de collègues qui, au fil du temps, sont devenus mes amis. Je les remercie tous, et particulièrement M. Mourid Marrakchi pour son aide et ses conseils.

Je ne peux clore cette partie de remerciements sans penser à l'âme de mon collègue et ami, feu Maher Ben Jemaa. Mon interaction avec lui pendant les années où il a été mon chef de département m'ont fait connaître un homme exceptionnel qui n'épargnait aucun effort pour aider ses collègues et ses étudiants. Paix à son âme.

Enfin, la totalité de ma reconnaissance et de mes pensées vont à mes parents Mohamed et Radhia, mon épouse Sahar et ma fille Eya qui n'ont jamais cessé de m'encourager et de m'apporter du support, surtout durant les dernières phases tumultueuses de la rédaction de ce mémoire. Il est donc tout naturel que ce manuscrit leur soit dédié.

© 2022 Bechir Zalila

La Modélisation Architecturale au Service des Systèmes Temps réel Répartis Embarqués - Contributions à la Configuration Dynamique, la Tolérance aux Pannes, l'Optimisation et la Vérification Formelle

Bechir ZALILA

ملخص

في أطروحة التأهيل الجامعي هذه، نصف المساهمات الرئيسية لأنشطتنا البحثية. تندرج هذه الأنشطة ضمن موضوع هيكلة البرمجيات في سياق آني، مضمن وموزع (TR^2E). تم وصف أربع مساهمات رئيسية : إعادة التكوين الديناميكي، التسامح مع الخطأ، التحسين متعدد الأهداف، وأخيرًا التحقق الشكلي من الأنظمة TR^2E . تستند جميع هذه المساهمات إلى لغات وصف الهيكل باعتبارها الركيزة الأساسية للنمذجة، نظرًا للقوة التعبيرية القوية لهذه اللغات، والتي تجعل من الممكن وصف البنية الكلية للنظام قبل استخدام تقنيات تحويل النماذج وإنشاء الكود لأداء الإجراءات المختلفة المتعلقة بكل من هذه المساهمات. نحن نستخدم هذه التقنيات على نطاق واسع لتحقيق أقصى قدر من الأتمتة وتقليل مقدار الكود المكتوب يدويا من قبل المطور.

الكلمات المفاتيح : الأنظمة الموزعة، الأنظمة المضمنة، الأنظمة الآنية، لغات وصف الهيكل، التكوين الديناميكي، التسامح مع الخطأ، توليد الكود، تحسين، تحقق شكلي، OCARINA ، POLYORB-HI ، AADL ، UML .

Résumé

Dans ce mémoire d'habilitation universitaire, nous décrivons les principales contributions de nos activités de recherche. Ces activités s'inscrivent dans la thématique des architectures logicielles dans un contexte temps réel réparti embarqué (TR^2E). Quatre contributions principales sont décrites : la reconfiguration dynamique, la tolérance aux pannes, l'optimisation multi-objectifs et enfin la vérification formelle des systèmes TR^2E. Toutes ces contributions se fondent sur les langages de description d'architectures (ADLs) comme pilier principal de modélisation, et ce, en raison du pouvoir d'expression très puissant de ces derniers, ce qui permet de décrire l'architecture globale d'un système avant d'utiliser des techniques de transformation de modèles et de génération de code pour réaliser les différentes actions relatives à chacune des contributions. Nous utilisons intensivement ces techniques afin de maximiser l'automatisation et réduire la quantité de code écrit à la main par le développeur.

Mots-clés: Systèmes répartis, systèmes embarqués, systèmes temps réel, langages de description d'architectures, configuration dynamique, tolérance au pannes, génération de code, optimisation, vérification formelle, OCARINA, POLYORB-HI, AADL, UML.

Abstract

In this academic habilitation report, we describe the main contributions of our research activities. These activities fall within the topic of software architectures in a distributed real-time embedded (DRE) context. Four main contributions are described : dynamic reconfiguration, fault tolerance, multi-objective optimization and finally formal verification of DRE systems. All these contributions are based on architecture description languages (ADLs) as the main modeling mean, due to the very powerful expressiveness of these languages, which makes it possible to describe the overall architecture of a system before using model transformation and code generation techniques to carry out the various actions relative to each contribution. We use these techniques extensively to maximize automation and reduce the amount of manually produced code by the developer.

Keywords: Distributed systems, embedded systems, real-time systems, architecture description languages, dynamic configuration, fault tolerence, code generation, optimisation, formal verification, OCARINA, POLYORB-HI, AADL, UML.

Table des matières

Introduction générale **1**
1 Contexte général 1
2 Contributions à la thématique ADLs dans un contexte TR²E 3
2.1 Reconfiguration dynamique 4
2.2 Tolérance aux pannes 4
2.3 Optimisation multi-objectifs 4
2.4 Vérification formelle 5
3 Plan du mémoire 5

1 Reconfiguration dynamique des systèmes TR²E **7**
1.1 Contexte général 8
1.2 Approche MDE pour la reconfiguration des systèmes TR²E 9
1.3 Architecture reconfigurable pour les systèmes TR²E 11
1.3.1 Méta-modèle de l'architecture 12
1.3.2 Profil UML pour les systèmes TR²E reconfigurables 13
1.4 Intergiciel pour les systèmes TR²E reconfigurables 14
1.4.1 Reconfiguration dynamique 14
1.4.2 Cohérence 15
1.4.3 Temps réel dur 15
1.4.4 Conformité au profil Ravenscar 16
1.5 Stratégie de génération de code 16
1.5.1 Méta-modèle d'implantation 17
1.5.2 Profil UML d'implantation 17
1.5.3 Synthèse de code 18
1.5.4 Outillage 18
1.6 Étude de cas 20
1.7 État de l'art 22
1.8 Conclusion et perspectives 25

2 Tolérance aux pannes dans les systèmes TR²E **26**
2.1 Contexte général . . . 27
2.1.1 Problématique et objectifs . . . 27
2.1.2 Notions de base sur la tolérance aux pannes . . . 28
2.2 Approche pour la tolérance au pannes dans les systèmes TR²E . . . 30
2.3 Extension d'un langage d'aspect pour les systèmes TR²E . . . 33
2.4 Génération de code pour les systèmes TR²E tolérants aux pannes . . . 35
2.5 Étude de cas . . . 37
2.5.1 Description . . . 37
2.5.2 Résultats . . . 38
2.6 État de l'art . . . 38
2.6.1 Modélisation de la tolérance aux pannes . . . 40
2.6.2 Tolérance aux pannes en utilisant la programmation orientée aspect 41
2.7 Conclusion et Perspectives . . . 42

3 Optimisation multi-objectifs des systèmes TR²E **44**
3.1 Contexte général . . . 45
3.1.1 Problématique et objectifs . . . 45
3.1.2 Contributions . . . 46
3.2 Exploration MOO de conceptions des systèmes TR²E . . . 48
3.2.1 Description de la solution . . . 50
3.2.2 Formulation d'une PAES pour l'affectation . . . 51
3.2.3 Règles d'affectation des fonctions aux threads . . . 55
3.2.4 Tests de faisabilité sur les solutions candidates . . . 56
3.3 Prototype implanté dans CHEDDAR . . . 57
3.4 Études Expérimentales . . . 59
3.5 État de l'art . . . 60
3.6 Conclusion et perspectives . . . 63

4 Vérification formelle des systèmes TR²E **65**
4.1 Contexte général . . . 66
4.2 Contributions . . . 67
4.3 Modèle de tâches LNT . . . 68
4.3.1 Un modèle de tâches compatible avec Ravenscar . . . 69
4.3.2 Mapping de l'ordonnancement . . . 69
4.3.3 Mapping des communications . . . 76
4.3.4 Composition et synchronisation . . . 77

© 2022 Bechir ZALILA

4.3.5 Discussion . . . 78
4.4 Transformation d'un modèle AADL vers LNT . . . 79
4.4.1 Règles de transformation . . . 80
4.4.2 Outillage . . . 81
4.5 Expérimentations . . . 83
4.5.1 Modélisation . . . 84
4.5.2 Génération de code . . . 84
4.5.3 Vérification formelle . . . 85
4.5.4 Analyse des résultats . . . 85
4.6 État de l'art . . . 86
4.7 Conclusion et perspectives . . . 87

Conclusion générale et perspectives **88**
1 Rappel des contributions et des résultats . . . 88
1.1 Processus de reconfiguration dynamique pour les systèmes TR^2E . . . 88
1.2 Processus de construction de systèmes TR^2E tolérants aux pannes . . . 89
1.3 Optimisation multi-objectifs des systèmes TR^2E . . . 89
1.4 Vérification formelle des systèmes TR^2E . . . 89
2 Perspectives . . . 90
2.1 Cours terme . . . 90
2.2 Moyen terme . . . 90
2.3 Long terme . . . 91

Bibliographie **92**

Liste des publications de l'auteur **100**

© 2022 Bechir Zalila

Liste des illustrations

1.1 Modélisation par MétaModes 10
1.2 Processus de la méthodologie de développement de systèmes TR²E reconfigurables 11
1.3 Méta-modèle de l'architecture à composants reconfigurables 12
1.4 Dépendances du profil RCA4RTES 13
1.5 Description du profil UML d'implantation 18
1.6 Génération de code et de modes 19
1.7 Framework de modélisation pour les systèmes TR²E reconfigurables 19
1.8 Machine à états du système GPS 21
1.9 Configuration GPS non sécurisé 22
1.10 Configuration GPS sécurisé 22

2.1 Classification des pannes 31
2.2 Processus de développement proposé 32
2.3 Nouvelle architecture pour le compilateur AspectAda 34
2.4 Processus de génération de code 36
2.5 Description globale du thermostat de l'Isolette 37
2.6 Modèle AADL du système Isolette 38
2.7 Modèle intermédiaire répliqué du système Isolette 39

3.1 Front de Pareto pour deux objectifs de minimisation 49
3.2 Description de l'approche 50
3.3 Représentation en chromosome d'une solution d'affectation 53
3.4 Représentation normalisée en chromosome d'une solution d'affectation . . 54
3.5 Affectation initiale 1–1 d'un exemple 55
3.6 Affectation après mutation et fusion de sections critiques 56
3.7 Affectation après mutation et suppression d'une section critique 56

4.1 Représentation graphique LNT 68
4.2 Automate d'états d'une tâche 71
4.3 Algorithme d'ordonnancement : tâche prête 74
4.4 Algorithme d'ordonnancement : boucles 75
4.5 Exemple de représentation graphique du processus **MAIN** 79
4.6 Processus MDE basé sur AADL 80
4.7 Règles de transformation AADL vers LNT 81
4.8 Chaînes d'outils Ocarina/CADP 82

Liste des exemples de code

4.1 Squelette d'une tâche LNT 71
4.2 Squelette de l'ordonnanceur LNT 73
4.3 Squelette d'un connecteur LNT 77
4.4 Type et canal LNT pour composer **TASK** et **SCHEDULER** 78
4.5 Exemple de processus LNT **MAIN** 78

Liste des tableaux

1.1 Propriétés non fonctionnelles du système GPS 21

2.1 Classification des pannes 29

Introduction générale

Sommaire

1 Contexte général 1
2 Contributions à la thématique ADLs dans un contexte TR²E 3
2.1 Reconfiguration dynamique 4
2.2 Tolérance aux pannes 4
2.3 Optimisation multi-objectifs 4
2.4 Vérification formelle 5
3 Plan du mémoire 5

1 Contexte général

Construire des systèmes informatiques est devenu de nos jours une tâche très fastidieuse en raison de la complexité toujours croissante de ces systèmes s'exécutant sur un matériel toujours plus performant. Cette complexité est davantage plus grande dans certains domaines tels que le temps réel et l'embarqué du fait que des contraintes supplémentaires (temps de réponse, empreinte mémoire, etc.) s'ajoutent à l'application. Lorsque le système à construire doit être distribué (réparti), des problèmes et contraintes supplémentaires viennent s'ajouter à la liste, déjà longue, des besoins à satisfaire [Vinoski, 2002].

Afin de simplifier la conception des systèmes Temps réel, Répartis, Embarqués (notés dans la suite de ce document systèmes TR²E), l'utilisation de la modélisation et de l'ingénierie guidée par les modèle (*Model Driven Engineering*, MDE) est devenue une nécessité [Fondement and Silaghi, 2004]. Passer par un modèle du système au lieu de développer directement ce dernier avec un langage de programmation offre une vision globale de l'application, permet au travers de certains outils d'effectuer des analyses avancées sur cette application et donne enfin au développeur la possibilité de générer automatiquement une large portion du code source de son application évitant ainsi de commettre des erreurs dues au codage manuel.

Les langages de description d'architecture (*Architecture Description Languages*, ADL), et particulièrement les ADLs concrets [Medvidovic and Taylor, 2000], se distinguent parmi les autres langages de modélisation par leur pouvoir d'expression permettant de décrire des aspects à haut niveau de l'application, leur aptitude à subir des analyses avancées comme l'analyse d'ordonnancement et enfin la possibilité de production automatique de code à partir du modèle. C'est donc sans une grande surprise que les ADLs concrets sont, jusqu'à aujourd'hui, au cœur des processus de développement dans des organisations

comme l'Agence Spatiale Européenne, et sont utilisés pour construire des systèmes critiques dans lesquels des vies humaines sont en jeu [Kordon *et al.*, 2013].

Nous avons utilisé les langages de description d'architectures dans notre travail de thèse [Zalila, 2008] pour effectuer la configuration et le déploiement automatiques des systèmes TR²E. Dans le travail susmentionné, les systèmes considérés étaient des systèmes statiques n'ayant pas de support pour la configuration dynamique. De plus, nous nous sommes concentrés uniquement sur la génération automatique de code et nous avons laissé l'analyse et le support de certains aspects, tels que la tolérance aux pannes et la vérification formelle, comme perspectives puisque ces dernières étaient en dehors du cadre de notre travail.

Chacune des perspectives listées à la fin de notre mémoire de thèse était en soi un projet de recherche à part entière et pouvait constituer, une fois réalisée une contribution importante à la thématique des architectures logicielles dans un contexte TR²E. Il était donc tout à fait naturel de commencer par nous intéresser à la continuation de ce travail au début de notre carrière de recherche et d'essayer d'apporter des éléments de résolution à chacune de ces perspectives. Nous avons donc décidé de nous attaquer à quatre défis dans le cadre de nos activités de recherche post-thèse.

Le premier défi a été d'implanter la reconfiguration dynamique dans les systèmes TR²E. La problématique principale de cette implantation était de réconcilier le dynamisme requis pour avoir la reconfiguration dynamique avec la conformité aux contraintes des systèmes critiques, qui nécessite un certain caractère statique du système. Nous avons résolu cette problématique en introduisant de nouveaux concepts lors de la modélisation de ce genre de systèmes.

Le second défi auquel nous nous sommes intéressés était l'implantation de la tolérance aux pannes dans les systèmes TR²E. La difficulté majeure était de supporter la tolérance aux pannes dès le stage de modélisation à cause de l'absence de support technologique pour cela. Nous avons surmonté cette difficulté en implantant une extension orientée aspect (paradigme de choix pour implanter la tolérance au pannes) dans un langage de programmation et aussi dans un langage de description d'architectures.

Le troisième défi, auquel nous avons fait face, était l'optimisation de l'affectation des fonctions aux threads dans les systèmes temps réel qui, jusque là, était une tâche chronophage et sujette à erreur qui devait être effectuée manuellement par le concepteur. Il s'agit d'un problème d'optimisation multi-objectifs assez difficile car, pour le résoudre, il faut réconcilier des objectifs qui sont parfois contradictoires. De plus, plus le nombre de ces objectifs est grand, plus le temps pour trouver la solution idéale sera long également. Ce problème a été résolu en créant et en implantant une méta-heuristique permettant cette optimisation et en réalisant un ensemble d'études expérimentales permettant de minimiser le nombre d'objectifs à considérer.

Enfin, le dernier défi auquel nous nous sommes intéressés a été d'implanter la vérification formelle des modèles architecturaux des systèmes TR²E. La problématique principale de ce défi était que les langages de description d'architectures n'étaient pas des formalismes adaptés à la vérification formelle. La solution que nous avons implantée était de réaliser un processus basé sur la transformation de modèles et la génération de code permettant d'automatiser l'analyse de ce genre de systèmes.

Nous avons sélectionné ces quatre contributions que nous avons jugées importantes car elle s'inscrivaient dans la continuité de notre travail de thèse. De plus, ces contributions, quoi que relativement indépendantes chacune de l'autre, partagent un même *fil conducteur* :

© 2022 Bechir Zalila

— Elles utilisent les architectures logicielles comme moyen principal de modélisation. En effet, les ADLs et particulièrement les ADLs concrets permettent de réaliser deux objectifs que l'on pensait contradictoires avant l'apparition de cette famille de langages : avoir un modèle abstrait d'un système tout en pouvant y intégrer des détails assez fins pour permettre d'effectuer des analyses très poussées du même modèle. Le même langage de description d'architecture peut jouer, selon le détail des constructions utilisées, le rôle de PIM (*Platform Independant Model*) ou de PSM (*Platform Specific Model*). Le passage entre ces deux rôles peut s'effectuer manuellement, ou mieux encore, par une simple transformation automatique de modèle. Pour cette raison, les ADLs sont des citoyens de première classe dans toutes les contributions décrites dans ce mémoire.

— Elles utilisent l'approche MDE (*Model Driven Engineering*) comme guide du processus de réalisation d'un système. En effet, les ADLs étant un cas particulier de langages de modélisation, nous pouvons profiter de toutes les possibilités de la MDE (transformation de modèles, etc.) et surtout du fait que plusieurs outils existent déjà pour réaliser ces tâches là.

— Elles s'appuient extensivement sur la génération automatique de code pour, d'une part, minimiser les risques d'erreurs dues au codage manuel et, d'autre part, réduire le temps et par la suite le coût du développement d'un système TR²E. Nous avons acquis un savoir faire respectable dans cette technique puisque nous avons travaillé dessus depuis notre projet de mastère de recherche [Zalila, 2005], durant lequel nous avons élaboré, entre-autres, une nouvelle technique de génération automatique de code fondée sur la conversion d'arbres syntaxiques abstraits (AST). Cette technique, contrairement à la génération de code *à la volée*, simplifie les générateurs de code et les rend extensibles tout en permettant d'atteindre des objectifs qui semblaient jusque là inatteignables. Pour cette raison, les contributions décrites dans ce mémoire, ont adopté cette même technique chaque fois que du code devait être généré.

— Elles se situent toutes dans un contexte temps réel embarqué et dans un contexte réparti pour la majorité d'entre elles. Ceci à permis d'utiliser les résultats et les savoirs acquis dans certaines contributions pour réaliser d'autres.

Dans la section suivante, nous résumons les quatre contributions de nos activités de recherche pendant les dix dernières années.

2 Contributions à la thématique ADLs dans un contexte TR²E

Dans cette section nous énonçons les contributions majeures que nous avons réalisées dans le cadre nos activités de recherche. Forts d'une expérience respectable dans la thématique des modèles architecturaux, de la transformation de modèles et surtout dans la génération automatique de code, nous avons décidé, comme précisé ultérieurement, de nous attaquer à plusieurs perspectives, laissées ouvertes à la fin de notre thèse. Les contributions ci-dessous se situent à mi-chemin entre **l'indépendance** (suffisamment pour constituer chacune un projet de recherche doctorale à part entière) et la **complémentarité** (suffisamment pour que les chercheurs travaillant dessus ne soient pas des *îlots* et puissent collaborer ensemble dans la réalisation de sous-contributions qui leur sont d'un intérêt commun). Nous estimons que cet objectif a été réalisé. Ces contributions ont été donc le fruit d'un travail de recherche d'équipe dans le cadre de travaux de thèses dont

© 2022 Bechir Zalila

nous avons participé à l'encadrement ou de mastères de recherche effectués sous notre encadrement. Pour chaque contribution, les auteurs des travaux de recherche correspondants sont cités au début du chapitre qui lui est relatif.

2.1 Reconfiguration dynamique

Le caractère statique des applications, quoique très important et obligatoire dans certains domaines critiques comme l'aéronautique ou l'espace, ne permet pas de supporter des fonctionnalités importantes comme la reconfiguration dynamique des systèmes TR²E. Une première contribution, a été donc, de concevoir une méthodologie de développement des systèmes TR²E dynamiquement reconfigurables en utilisant l'approche MDE et plus précisément les modèles architecturaux. Ceci a nécessité l'introduction de nouvelles notions, jusque là inexistantes comme le concept de MétaMode et l'implantation d'un nombre d'outils (méta-modèles, profils UML, etc.). Il fallait, par la suite, s'assurer que l'introduction de la reconfiguration dynamique dans un contexte temps réel ne compromettait pas l'exactitude du système construit. Ceci a été effectué, en partie, en testant la conformité de notre modèle à certains profils d'exécution existants comme le profil Ravenscar. Cette contribution a été validée par une étude de cas décrivant un système TR²E dynamiquement reconfigurable. Elle a donné lieu à un nombre de publications dans des revues et de conférences de renommée internationale.

2.2 Tolérance aux pannes

Une fois la reconfiguration dynamique dans les systèmes TR²E maîtrisée, il était tentant d'utiliser le savoir-faire acquis afin d'implanter la tolérance aux pannes (TP) dans ces systèmes. La seconde contribution décrite dans ce mémoire est donc, l'implantation d'un processus de construction de systèmes TR²E tolérants aux pannes en utilisant les ADLs. Pour dompter les problématiques liées à cette contribution, plusieurs sous-contributions ont du être réalisées. Premièrement, la TP étant une préoccupation transversale d'un système, utiliser la Programmation Orientée Aspect (POA) pour l'implanter était le choix naturel. Toutefois, les outils de modélisation existants ne supportaient pas ce paradigme de programmation. Une première sous-contribution a été d'implanter le support de la POA pour enrichir un langage de description d'architecture. Deuxièmement, le contexte de cette contribution étant les systèmes TR²E critiques, le langage de programmation Ada comme cible de la génération de code était un choix quasi-obligatoire. Cependant, l'extension orientée aspect pour ce langage était encore à un stade embryonnaire de développement et son utilisation était non conforme aux exigences des systèmes TR²E. Une seconde sous-contribution a été de compléter l'implantation de cette extension et de la rendre conforme à ces exigences. Finalement, pour compléter cette contribution, des enrichissements et de nouveaux outils ont été implantés afin d'achever la réalisation de tout le processus de développement. Cette contribution a été validée avec plusieurs études de cas dont la plus importante et volumineuse a été reprise dans ce mémoire. Elle a donné également lieu à un nombre de publications dans des revues et des conférences de renommée internationale.

2.3 Optimisation multi-objectifs

Après la reconfiguration dynamique et la tolérance aux pannes, nous nous sommes intéressés à une autre problématique qui touche les systèmes TR²E et plus particulière-

© 2022 Bechir Zalila

ment, les systèmes TR²E critiques : l'affectation des fonctionnalités aux threads du système. Jusque là, chaque fonction implantée (généralement périodique) était affectée à un thread. Multiplier le nombre de threads va également multiplier le nombre de changements de contexte, occuper plus de mémoire à cause de la multiplication des piles d'exécution et augmenter les temps de blocage de chaque thread lorsqu'il y a des ressources partagées. Par ailleurs, affecter toutes les fonctions à un seul thread revient à jouer soi-même le rôle d'ordonnanceur et diminue considérablement la flexibilité du système et son extensibilité. Il s'agit donc de trouver le bon compromis qui consiste à affecter certaines fonctions au même thread. Puisque les critères à satisfaire sont multiples et souvent contradictoires, il s'agit d'un problème d'optimisation multi-objectifs (MOO). Nous avons développé une approche MOO basée sur les algorithmes évolutionnaires (MOEA) et plus particulièrement sur une stratégie évolutionaire archivée (PAES) afin d'aider le concepteur d'une architecture logicielle à trouver l'affectation des fonctions aux threads qui réponde le plus aux besoins et aux contraintes de son application. Cette approche a inclus un ensemble d'études empiriques qui nous a permis de sélectionner un nombre minimal de fonctions-objectifs non redondantes afin d'accélérer la recherche de telles solutions architecturales. Cette contribution, ainsi que les études empiriques associées ont donné lieu à un nombre de publications dans une revue et des conférences de renommée internationale.

2.4 Vérification formelle

Nous nous sommes enfin intéressés à la question de la vérification formelle des architectures TR²E qui était également une perspective de notre travail de thèse et de la contribution mentionnée à la section 2.2. Notre approche était de ne pas *"réinventer la roue"* et d'utiliser des outils de vérification existants ayant déjà fait leur preuve. Il fallait donc convertir le modèle architectural en une spécification formelle. Le modèle architectural, tout seul, étant insuffisant pour produire une spécification formelle riche, il fallait que ce dernier soit décoré par des éléments comportementaux. Notre dernière contribution a consisté donc, tout d'abord à définir un modèle formel de concurrence temps réel pour le langage LNT qui est une algèbre de processus enrichie avec des éléments de la programmation impérative. Ce modèle est conçu pour être modulaire et compréhensible afin d'être facilement étendu et utilisé dans les approches MDE. Nous prenons principalement en charge les tâches périodiques et sporadiques, qui sont connectées de manière asynchrone et exécutées simultanément par un ordonnanceur préemptif à priorité fixe. Le second volet de cette contribution a consisté à définir une approche MDE basée sur le langage AADL et intégrant la vérification formelle lors de la phase de modélisation. Cela permet la détection précoce de problèmes plus profonds pouvant entraîner de graves erreurs dans le système final ou augmenter le coût de correction si découverts plus tard. Cette vérification est censée être automatique et transparente pour simplifier et encourager la pratique de méthodes formelles en génie logiciel. Cette contribution a été validée avec plusieurs études de cas. Elle a également donné lieu à un nombre de publications dans des revues et des conférences de renommée internationale.

3 Plan du mémoire

Ce mémoire est organisé comme suit :

© 2022 Bechir Zalila

— Dans le chapitre 1, nous présentons la première contribution : la reconfiguration dynamique dans les systèmes TR^2E en utilisant les modèles architecturaux.

— Dans le chapitre 2, nous présentons la seconde contribution : l'implantation d'un processus de construction des systèmes TR^2E tolérants aux pannes en utilisant les modèles architecturaux et la programmation orientée aspect.

— Dans le chapitre 3, nous présentons la troisième contribution : l'optimisation multi-objectifs des systèmes TR^2E critiques en utilisant un algorithme évolutionnaire afin d'aider le concepteur à trouver l'affectation idéale des fonctions aux threads du système.

— Dans le chapitre 4, nous présentons la quatrième contribution : la définition d'un modèle formel pour les systèmes TR^2E et la définition et implantation de règles de génération de code d'un langage de description d'architecture vers ce modèle formel.

— Le dernier chapitre donne une conclusion générale pour ce mémoire et énonce nos perspectives vers de futurs travaux de recherche.

© 2022 Bechir Zalila

Chapitre 1

Reconfiguration dynamique des systèmes TR²E

Sommaire

1.1 Contexte général 8
1.2 Approche MDE pour la reconfiguration des systèmes TR²E 9
1.3 Architecture reconfigurable pour les systèmes TR²E 11
  1.3.1 Méta-modèle de l'architecture 12
  1.3.2 Profil UML pour les systèmes TR²E reconfigurables 13
1.4 Intergiciel pour les systèmes TR²E reconfigurables 14
  1.4.1 Reconfiguration dynamique 14
  1.4.2 Cohérence 15
  1.4.3 Temps réel dur 15
  1.4.4 Conformité au profil Ravenscar 16
1.5 Stratégie de génération de code 16
  1.5.1 Méta-modèle d'implantation 17
  1.5.2 Profil UML d'implantation 17
  1.5.3 Synthèse de code 18
  1.5.4 Outillage 18
1.6 Étude de cas 20
1.7 État de l'art 22
1.8 Conclusion et perspectives 25

Dans ce chapitre, nous présentons la première contribution de ce travail de recherche : la reconfiguration dynamique des systèmes TR²E. Nous proposons une approche dirigée par les modèles permettant de construire des systèmes TR²E dynamiquement reconfigurables. La croissance constante de la complexité et l'autonomie requise pour la gestion des systèmes logiciels embarqués donnent à la reconfiguration dynamique une grande importance. De nouveaux défis pour appliquer la reconfiguration dynamique au niveau du modèle ainsi qu'au niveau du support d'exécution s'avèrent nécessaires.

Dans ce contexte, le développement des systèmes TR²E reconfigurables selon les processus traditionnels n'est pas applicable. De nouvelles méthodes sont nécessaires pour construire et fournir des architectures logicielles embarquées reconfigurables.

Le travail décrit dans ce chapitre présente une approche basée sur l'ingénierie dirigée par les modèles (MDE), permettant de concevoir des systèmes TR²E reconfigurables avec leur support d'exécution. Cette approche guide le concepteur afin de spécifier pas à pas son système en partant d'un modèle pour arriver à un autre plus raffiné jusqu'à ce que le modèle visé soit atteint. Ce modèle cible est lié à une plateforme d'exécution spécifique conduisant à la génération de la plus grande partie de l'implantation du système. Nous avons également développé un nouvel intergiciel qui supporte les systèmes TR²E reconfigurables.

La contribution décrite dans ce chapitre est le fruit d'un travail de recherche d'équipe. Il s'agit des résultats des travaux de thèse de doctorat de Mme Fatma Krichen [Krichen, 2013] réalisée dans le cadre d'une cotutelle entre l'Université de Sfax et l'Université de Toulouse, et des projets de mastères de recherches de Mmes Amal Gassara [Gassara, 2011], Alvine Boaye Belle [Belle, 2011] et Amal Ghorbel [Ghorbel, 2012].

1.1 Contexte général

Un système TR²E est constitué d'une partie logicielle et d'une partie matérielle. L'intégration de ces deux parties est l'une des tâches les plus importantes du développeur. La reconfiguration dynamique consiste à faire évoluer le système de sa configuration actuelle vers une autre configuration, et ce, pendant l'exécution. La configuration d'un système TR²E est l'action de paramétrage de chacun des composants de ce système et leur assemblage pour former une application opérationnelle [OMG, 2006]. Cette évolution peut être aussi bien architecturale que comportementale. La reconfiguration architecturale consiste à modifier la topologie du système en ajoutant ou en supprimant des composants ou des connexions. La reconfiguration comportementale consiste, quant à elle, à modifier le comportement d'un système, par exemple en mettant à jour ses propriétés non fonctionnelles ou encore en modifiant le comportement d'un ou plusieurs de ses composants.

Alors que l'une des propriétés fondamentales d'un système TR²E est l'autonomie, cette autonomie ne couvre que le comportement classique du système. Pour ce qui concerne la reconfiguration, la plupart des systèmes TR²E ne sont pas entièrement autonomes et requièrent l'intervention humaine pour répondre à certains événements demandant la reconfiguration. Cette intervention humaine est une possible source d'erreur et requiert une quantité considérable d'attention pour être effectuée avec succès.

Par ailleurs, il est parfois impossible d'interrompre l'activité d'un système TR²E critique afin de le reconfigurer. Par conséquent, la reconfiguration dynamique est requise pour construire des systèmes TR²E autonomes. Construire ce genre de systèmes nécessite un effort considérable et peut être sujet à erreur à cause de la complexité parfois très grande de ces systèmes. Pour réduire cette complexité, on passe généralement par un langage de modélisation de haut niveau afin de décrire l'architecture du système. Ceci nécessite la création de nouveaux concepts de modélisation, nécessaires pour spécifier les reconfigurations dynamiques de ces systèmes.

Dans un autre contexte, les ressources matérielles d'un système embarqué sont généralement limitées et leur utilisation doit être optimisée. Pour développer un système embarqué riche avec plusieurs fonctionnalités et des ressources matérielles à faible coût, les ressources matérielles doivent être allouées uniquement lorsque cela est nécessaire. Pour les architectures à base de composants, les composants doivent être remplacés/mis

© 2022 Bechir Zalila

à jour au moment de l'exécution pour être réutilisés et pour fournir différentes fonctionnalités.

La majorité des activités de recherche présentent des systèmes reconfigurables avec un nombre prédéfini de configurations. La plupart des travaux réalisés dans ce sens ont été proposés dans le cadre des deux normes : AADL (Architecture Analysis & Design Language) [SAE, 2017a] et MARTE (Modeling and Analysis of Real-Time Embedded Systems) [OMG, 2019]. Ces deux normes définissent les reconfigurations dynamiques en termes de modes et de transitions de modes. Un mode représente une configuration particulière tandis qu'une transition représente un événement, qui implique la reconfiguration du système d'un mode à un autre. Les reconfigurations sont décrites à l'aide de machines à états composées d'un nombre prédéfini de modes et de transitions entre eux. L'inconvénient majeur de ces deux normes est la nécessité de définir un nombre prédéfini de configurations.

Face à l'évolution exponentielle des exigences des systèmes TR²E reconfigurables, les développeurs ont un temps réduit pour prototyper et commercialiser leurs systèmes. Cette contrainte est un facteur important pour avoir un avantage concurrentiel. Pour cette raison, les développeurs doivent construire un système aussi rapidement que possible, garantissant les fonctionnalités requises. Pour faire face à la complexité croissante de la conception des systèmes, plusieurs approches de raffinement ont été proposées. La plus populaire est l'ingénierie dirigée par les modèles (MDE) [Whittle *et al.*, 2014]. En utilisant les langages de modélisation, les modèles représentent les principaux artefacts à construire et à maintenir. Dans le contexte MDE, le développement logiciel consiste à transformer un modèle en un autre plus raffiné jusqu'à ce que le modèle visé soit atteint. Ce modèle cible est lié à une plateforme spécifique et il est prêt à être exécuté.

Dans ce chapitre, nous fournissons une approche permettant de concevoir des systèmes TR²E reconfigurables. Pour cela, nous proposons une approche basée sur les modèles qui définit un ensemble d'étapes à suivre par le développeur. Un ensemble de règles de transformation permet la transformation d'un modèle vers un autre modèle. Contrairement à AADL et à MARTE, notre approche permet de spécifier un système reconfigurable avec un nombre de configurations non prédéfini. Pour cela, nous introduisons de nouveaux concepts capturant les reconfigurations dynamiques des systèmes TR²E. De plus, nous avons développé un nouvel intergiciel qui assure la reconfiguration dynamique ainsi que la surveillance et la cohérence des systèmes TR²E.

Ce chapitre est organisé comme suit. Dans la section 1.2, nous décrivons l'ensemble de notre méthodologie de développement pour concevoir des systèmes TR²E reconfigurables. La section 1.3 présente l'architecture à composants reconfigurables proposée pour l'approche (RCA4RTES) pour spécifier des systèmes TR²E reconfigurables. La section 1.4 décrit l'intergiciel proposé dédié aux systèmes TR²E reconfigurables. La stratégie de génération de code est décrite dans la section 1.5. La section 1.6 illustre l'efficacité de l'approche proposée à travers une étude de cas ayant des exigences de reconfiguration dynamique. Dans la section 1.7, nous passons brièvement en revue certains travaux connexes qui traitent du développement de systèmes TR²E. Enfin, la section 1.8 conclut ce chapitre et présente quelques perspectives.

1.2 Approche MDE pour la reconfiguration des systèmes TR²E

Pour résoudre les problèmes mentionnés précédemment, nous proposons une approche MDE pour concevoir des systèmes TR²E reconfigurables avec un nombre non pré-

© 2022 Bechir Zalila

défini de configurations. Pour cela, nous introduisons le nouveau concept de MétaMode qui capture et caractérise un ensemble de configurations (modes) au lieu de définir chacune d'entre elles. Le MétaMode est décrit par des composants structurés, des connecteurs liant ces composants ainsi que des contraintes non fonctionnelles et structurelles. Les modes appartenant à un MétaMode sont spécifiés par l'ensemble des instances de composants structurés et de connecteurs définis par le MétaMode en question et satisfaisant ses contraintes.

Notre approche définit des reconfigurations basées sur des politiques. Nous spécifions des reconfigurations dynamiques à l'aide de machines à états qui définissent un ensemble de MétaModes et des transitions entre eux. Une transition entre deux MétaModes représente un ensemble de reconfigurations entre les modes appartenant à ces MétaModes (comme le montre la figure 1.1). Lorsqu'un événement (représenté comme une transition entre deux MétaModes) survient, des reconfigurations (représentées comme une transition entre modes) sont appliquées sur le mode actuel pour atteindre l'un des modes appartenant au MétaMode cible. Les politiques de reconfiguration permettent de sélectionner automatiquement le mode cible. Les politiques de reconfiguration envisagées dans ce cadre sont l'optimisation de l'usage de mémoire, du processeur ainsi que de la bande passante réseau.

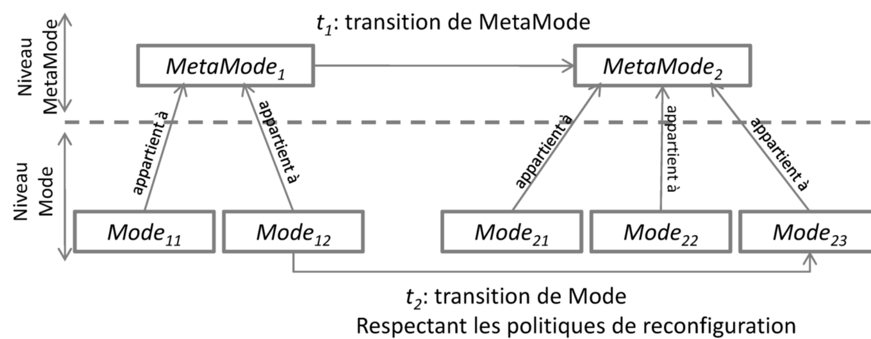

FIGURE 1.1 – Modélisation par MétaModes [Krichen, 2013]

La figure 1.1 illustre les concepts précédents avec un exemple très simple. Le concepteur spécifie les reconfigurations dynamiques de son système TR²E à l'aide d'une machine à états, qui contient deux MétaModes : $MetaMode_1$ et $MetaMode_2$. Une transition t_1 représente une reconfiguration entre ces deux MétaModes. La transition de mode t_2 est une des transitions possibles déduites de t_1 grâce aux politiques de reconfiguration. Le mode actuel $Mode_{12}$ est automatiquement remplacé par le mode $Mode_{23}$.

Par la suite, chaque MétaMode doit être alloué sur l'architecture matérielle. Cette dernière étant inchangée, l'allocation est définie en partant des modèles d'architectures logicielles (MétaModes) vers les supports d'exécution. Certaines contraintes d'allocation doivent être définies afin de préciser les politiques d'allocation. Ces politiques définissent la projection des modèles logiciels vers l'instance matérielle.

Tous les concepts décrits précédemment permettent de modéliser des systèmes TR²E reconfigurables. Pour effectuer la génération de code, les modes seront transformés en modèles d'implantation. Les modèles d'implantation permettent de générer du code en utilisant les routines de notre support d'exécution (intergiciel) pour les composants reconfigurables proposé pour les systèmes embarqués temps réel (RCES4RTES) [Krichen *et al.*, 2012b].

© 2022 Bechir ZALILA

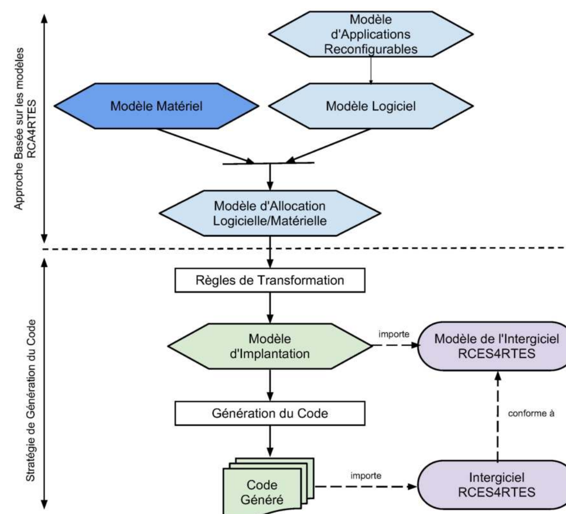

FIGURE 1.2 – Processus de la méthodologie de développement de systèmes TR²E reconfigurables [Krichen, 2013]

Nous définissons une méthodologie basée sur une approche MDE pour passer des modèles au code source, comme le montre la figure 1.2. Cette méthodologie comporte un processus qui se compose de cinq étapes à suivre par l'utilisateur :

(1) **Spécification de modèle d'application reconfigurable :** modélisation des reconfigurations dynamiques à l'aide de machines à états composées d'un ensemble de MétaModes et de transitions entre eux. Des politiques de reconfiguration doivent également être spécifiées pour sélectionner le mode cible. Ce mode est conforme au méta-modèle RCA4RTES [Krichen *et al.*, 2011],

(2) **Spécification de modèle logiciel** : modélisation des MétaModes du système où chacun est composé d'un ensemble de composants structurés, de connecteurs ainsi que de contraintes non fonctionnelles et structurelles. Le modèle logiciel est également conforme au méta-modèle RCA4RTES,

(3) **Spécification du modèle matériel** : modélisation de l'architecture matér ielle fixe en termes de composants matériels (tels que le processeur et le bus) à l'aide du profil MARTE [OMG, 2019],

(4) **Spécification du modèle de projection matériel/logiciel :** allocation des MétaModes du système sur l'architecture matérielle fixe spécifiée également à l'aide du méta-modèle RCA4RTES,

(5) **Génération automatique du modèle d'implantation :** génération de modèles qui représentent l'implantation du système. Ce modèle est conforme au méta-modèle d'implantation [Krichen *et al.*, 2012a]. À partir du modèle d'implantation, du code source est généré pour l'intergiciel RCES4RTES proposé [Krichen *et al.*, 2012b].

1.3 Architecture reconfigurable pour les systèmes TR²E

Dans cette section, nous décrivons les nouveaux concepts introduits pour spécifier les systèmes TR²E reconfigurables. Ces concepts permettent de spécifier un modèle d'ap-

© 2022 Bechir ZALILA

plication reconfigurable, un modèle logiciel et un modèle de projection matériel/logiciel comme mentionné à la fin de la section précédente. Pour cela, un nouveau méta-modèle, appelé RCA4RTES, a été proposé pour décrire ces nouveaux concepts et les relations entre eux. Comme implantation de ce méta-modèle proposé, un profil UML a également été proposé.

1.3.1 Méta-modèle de l'architecture

Dans ce qui suit, nous détaillons le méta-modèle RCA4RTES proposé [Krichen *et al.*, 2011] et illustré à la figure 1.3 pour spécifier les systèmes TR²E reconfigurables.

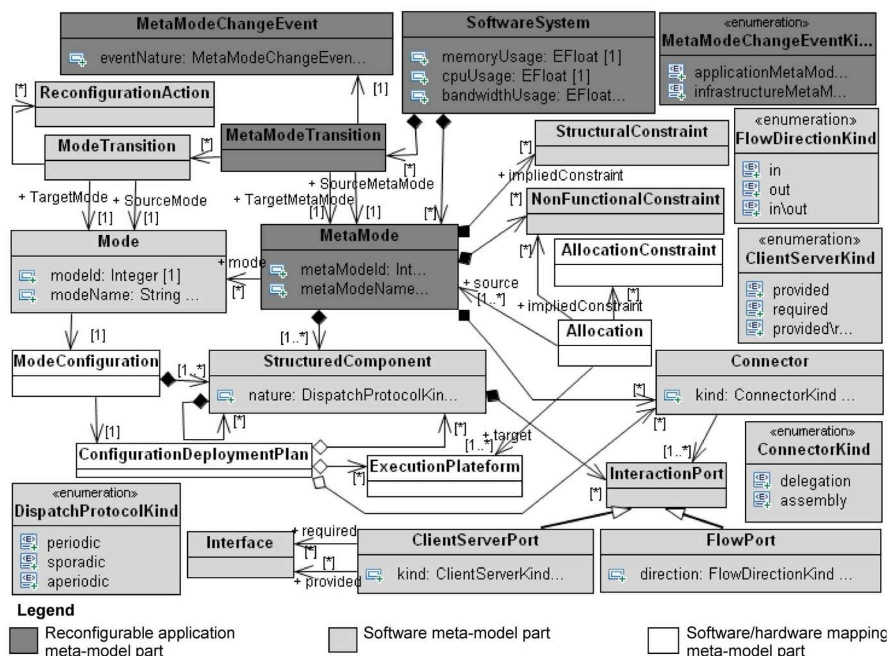

FIGURE 1.3 – Méta-modèle de l'architecture à composants reconfigurables [Krichen, 2013]

Pour spécifier les politiques de reconfiguration requises, nous introduisons trois propriétés (**`cpuUsage`**, **`memoryUsage`** et **`bandWidthUsage`**). Chaque propriété doit avoir une valeur pour indiquer le taux de consommation à ne pas dépasser par les ressources correspondantes.

Pour définir le méta-modèle qui est composé d'un ensemble de composants structurés, de connecteurs et de contraintes structurelles et non fonctionnelles, nous introduisons la méta-classe **`MetaMode`** et la méta-classe **`StructuredComponent`** qui est composée d'un ensemble de ports d'interaction. Chaque composant structuré peut être un thread périodique, sporadique ou apériodique, ou une composition de composants structurés. Pour décrire la communication entre les composants, nous introduisons la méta-classe **`Connector`** qui relie deux ou plusieurs ports d'interaction. Un connecteur peut être un connecteur de délégation (entre deux ports de sortie ou deux ports d'entrée) ou un connecteur d'assemblage (entre un port d'entrée et un port de sortie). Deux types de ports sont supportés : le port de flux et le port client/serveur. Un port de flux présenté par la méta-classe **`FlowPort`** a été introduit pour décrire la communication orientée *"flux de données"* entre les composants tandis qu'un port client/serveur présenté par la méta-classe

© 2022 Bechir ZALILA

ClientServerPort a été ajouté pour définir un paradigme de communication requête/-réponse entre les composants tels que les appels d'opérations ou encore les signaux.

Chaque MétaMode possède une ou plusieurs instances décrites à l'aide de la méta-classe **Mode**. Pour chaque mode, une configuration associe ce mode au plan de déploiement. Un plan de déploiement décrit une configuration par un ensemble de composants structurés, les connexions entre eux, leur configuration et leur affectation aux nœuds physiques. Nous introduisons la méta-classe **Allocation** pour spécifier l'allocation des MétaModes aux supports d'exécution (par exemple, l'allocation de modèles logiciels à une architecture matérielle fixe). Cette allocation possède des contraintes non fonctionnelles et d'allocation qui doivent être respectées. Notre méta-modèle définit trois types de contraintes :

— Les contraintes structurelles : liées à la topologie des architectures à base de composants,

— Les contraintes non fonctionnelles : spécifient des conditions sur les propriétés non fonctionnelles associées aux modèles (i.e., composants et connecteurs),

— Les contraintes d'allocation : spécifient les politiques utilisées pour l'allocation des modèles logiciels (MétaModes) à une architecture matérielle fixe. Les contraintes d'allocation sont décrites en utilisant VSL, le langage de spécification de valeur du profil MARTE [OMG, 2019].

1.3.2 Profil UML pour les systèmes TR²E reconfigurables

Pour gérer les exigences de reconfiguration des systèmes TR²E, un profil UML a été dérivé du méta-modèle RCA4RTES. La figure 1.4 montre les dépendances de ce profil. Il importe à la fois les profils NFP (*Non Functional Properties*) et VSL de MARTE pour spécifier les contraintes non fonctionnelles et d'allocation et les types NFP de base de la bibliothèque MARTE pour utiliser les types définis dans cette bibliothèque. La description complète du profil est donnée dans [Krichen, 2013].

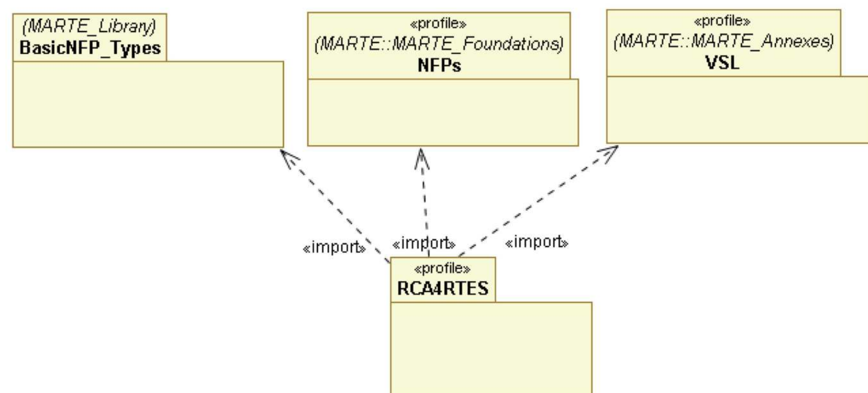

FIGURE 1.4 – Dépendances du profil RCA4RTES [Krichen, 2013]

Comme mentionné précédemment, un MétaMode caractérise l'état du système par un ensemble de composants structurés, de connecteurs et de contraintes structurelles et non fonctionnelles. Par conséquent, le stéréotype **MetaMode** étend la méta-classe **State** UML. Comme nous nous intéressons aux systèmes embarqués temps réel, chaque composant

© 2022 Bechir Zalila

structuré est considéré comme un thread ou un ensemble de threads. Pour définir et caractériser ces threads, nous définissons de nouvelles propriétés telles que la nature, la période, l'échéance, les temps de début et de fin d'exécution et enfin le pire temps d'exécution d'un thread (WECT).

1.4 Intergiciel pour les systèmes TR²E reconfigurables

L'intergiciel RCES4RTES [Krichen *et al.*, 2011] prend en charge les systèmes TR²E reconfigurables. La fonction centrale de cet intergiciel est la reconfiguration dynamique des systèmes TR²E basés sur des composants logiciels. RCES4RTES fournit également d'autres fonctions nécessaires au développement d'applications embarquées temps réel reconfigurables. Pour développer cet intergiciel, nous avons étendu PolyORB-HI [Zalila *et al.*, 2008], un intergiciel réalisé dans le cadre de notre travail de thèse [Zalila, 2008]. L'intergiciel étendu offre les fonctionnalités suivantes :

— Surveiller le système en supervisant, à l'exécution, la topologie et le comportement de l'architecture, en traçant l'exécution du système (par exemple, obtenir le nombre de composants et de connexions). La fonction de surveillance peut également être utilisée pour assurer la réflexivité du système. Il s'agit d'une nouvelle fonctionnalité réalisée dans le cadre de cette contribution,

— Préserver la cohérence du système pendant et après les reconfigurations car la reconfiguration peut conduire le système vers des états incohérents. Il s'agit, là aussi, d'une nouvelle fonctionnalité réalisée dans le cadre de cette contribution,

— Respecter les contraintes temps réel. En effet, sur chaque nœud du système, un thread de reconfiguration dynamique est automatiquement créé. Il s'agit d'un thread sporadique appliquant des actions de reconfiguration. Il est considéré comme un thread système, mais ordonnancé avec les autres threads présents. En utilisant ce thread sporadique, notre intergiciel peut facilement gérer les reconfigurations sans affecter l'exécution des threads systèmes et sans dépasser leurs délais. Là aussi, il s'agit d'une nouvelle fonctionnalité que nous avons réalisée dans le cadre de cette contribution,

— Assurer la communication entre plateformes hétérogènes en utilisant l'architecture schizophrène [Pautet, 2002] et ses services canoniques. Profitant de ces services et contrairement à plusieurs intergiciels existants, notre intergiciel RCES4RTES a une faible empreinte mémoire,

— Respecter les restrictions du profil Ravenscar [Burns *et al.*, 2004] pour garantir l'ordonnançabilité et l'absence d'interblocage et de famine dans le système construit.

Plus de détails sur les fonctionnalités de l'intergiciel RCE4RTES ainsi que leurs avantages seront donnés dans les sous-sections qui suivent.

1.4.1 Reconfiguration dynamique

L'intergiciel RCES4RTES effectue la reconfiguration dynamique des systèmes TR²E à travers deux types de reconfigurations : la reconfiguration architecturale et la reconfiguration comportementale. Il gère les reconfigurations architecturales suivantes : *Connexion de nœuds*, *Déconnexion de nœuds*, *Ajout d'une connexion*, *Suppression d'une connexion*,

© 2022 Bechir Zalila

Ajout d'un composant, *Suppression d'un composant* et *Migration d'un composant*. Il gère également les reconfigurations comportementales suivantes : *Mise à jour des propriétés d'un composant* et *Remplacement de composants*.

Il introduit également un ensemble de structures de données pour assurer les reconfigurations précédentes et mettre à jour l'état actuel de l'application en termes de nœuds, composants, connexions et ports :

(1) Une structure de données est définie dans chaque nœud pour décrire les connexions entre ce nœud et les autres nœuds et pour gérer l'interconnexion dynamique des nœuds. Chaque paire de nœuds ayant au moins une connexion entre leurs composants associés doit être connectée. La connexion et la déconnexion des nœuds nécessitent la mise à jour des structures de données correspondantes.

(2) Une structure de données est introduite pour mettre à jour au moment de l'exécution l'état de l'interconnexion entre les composants. Dans chaque nœud, cette structure de données représente les ports de destination de chaque port de composant déployé. L'ajout ou la suppression d'un connecteur entre deux ports nécessite la mise à jour des structures de données correspondantes.

(3) Deux structures de données sont définies dans chaque nœud pour l'ajout, la suppression et la migration de composants lors de l'exécution. La première contient tous les nœuds d'application avec leurs instances correspondantes de composants déployés, tandis que la seconde contient toutes les instances de composants d'application avec leurs ports associés. Lorsqu'un composant est ajouté, supprimé ou migré, ces deux structures de données sont mises à jour dans chaque nœud pour assurer la cohérence du système.

1.4.2 Cohérence

La cohérence est une propriété essentielle d'un système reconfigurable. Le système doit être dans un état correct pendant et après les reconfigurations pour éviter les pannes. Pour maintenir l'état correct d'un système, nous devons éviter la perte de messages entre les composants lors des reconfigurations. Chaque composant affecté par le processus de reconfiguration doit être verrouillé pendant la reconfiguration. Pour cela, nous définissons deux routines de verrouillage et de déverrouillage des composants. Le verrouillage de composant consiste à empêcher les composants sources (i.e. les composants qui envoient des requêtes au composant verrouillé) d'envoyer des requêtes et à achever le traitement de toutes les requêtes en cours. Le déverrouillage de composant consiste à libérer le verrou en permettant aux composants sources de lui envoyer des requêtes.

Pour éviter la perte de temps et minimiser autant que possible la durée de verrouillage, le verrouillage et le déverrouillage des composants doivent être effectués respectivement juste après la création de nouveaux composants et juste avant la suppression des composants dans les routines de reconfiguration.

1.4.3 Temps réel dur

Pour garantir le respect des contraintes temps réel dur, il faut comptabiliser le temps de reconfiguration T_{rd} lors de l'analyse du système. Il s'agit de la somme de la durée de verrouillage des composants T_b, du temps d'exécution des actions de reconfiguration T_{act}

© 2022 Bechir Zalila

et du temps de transfert de l'état du composant T_{state}. T_{state} n'est défini que dans le cas de la migration de composant d'un nœud à un autre tandis que T_{act} est le temps d'exécution du thread de reconfiguration dynamique. Pour chaque nœud, un thread sporadique (i.e. un thread de reconfiguration dynamique) est créé pour effectuer les reconfigurations sur ce nœud et pour informer les autres nœuds de ses reconfigurations.

T_b, T_{state} et T_{act} sont généralement proportionnels à la taille des données à traiter. T_{act} dépend également de la fréquence du processeur et de la couche de transport. Pour obtenir un temps de reconfiguration déterministe, on devrait utiliser une couche de transport déterministe telle que SPACEWIRE [ESA, 2008].

1.4.4 Conformité au profil Ravenscar

Le profil Ravenscar [Burns *et al.*, 2004] introduit des restrictions permettant l'ordonnançabilité et l'absence d'interblocage et de famine dans les systèmes embarqués temps réel. Pour garantir tout cela, le profil Ravenscar interdit l'utilisation de threads qui sont lancés à des moment arbitraires. Il recommande l'utilisation exclusive de threads périodiques et sporadiques. Par conséquent, l'ensemble des threads à analyser est fixe et possède des propriétés statiques. Le profil Ravenscar nécessite également des communications asynchrones. Les communications entre les threads ne devraient être assurées que par un ensemble statique d'objets partagés protégés, et l'accès à ces objets partagés protégés devrait être effectué à l'aide du protocole de plafonnement de priorité (*Priority Ceiling Protocol*, PCP) [Sha *et al.*, 1990].

Dans notre approche, nous nous intéressons à trois types de threads : les threads périodiques, sporadiques et apériodiques. Comme chaque thread apériodique a un instant d'arrivée, il peut être assimilé à un thread sporadique qui se lance une seule fois. Par conséquent, nous pouvons simplifier notre approche en considérant uniquement les threads périodiques et sporadiques.

Nous utilisons également des communications asynchrones entre les threads du système à l'aide du protocole PCP. Comme nous nous concentrons sur les systèmes distribués, le temps de construction et d'envoi des messages est non déterministe en raison de la non-fiabilité des couches de transport. Cela peut être une source de perte de message. Pour résoudre cette limitation, nous proposons d'utiliser une couche de transport fiable et conforme aux exigences des systèmes temps réel telle que SPACEWIRE [ESA, 2008]. Nous pouvons donc considérer notre système distribué comme un système local. Il faut également effectuer l'analyse statique de chaque MétaMode séparément étant donné que le nombre de threads peut différer d'un MétaMode à un autre.

L'intergiciel RCES4RTES est utilisé pour générer le code des systèmes TR^{2}E reconfigurables.

1.5 Stratégie de génération de code

Afin de générer la majeure partie d'une implantation d'un système, un nouveau méta-modèle et une implantation de ce nouveau méta-modèle ont été définis pour décrire le modèle d'implantation de chaque système. Dans les sous-sections suivantes, nous décrivons à la fois le méta-modèle ainsi que son profil implantation, et nous détaillons la synthèse du code.

© 2022 Bechir ZALILA

1.5.1 Méta-modèle d'implantation

Nous avons défini un nouveau méta-modèle qui permet la représentation de modèles d'implantation de systèmes TR²E reconfigurables. Comme les implantations des systèmes utilisent des routines définies dans l'intergiciel RCES4RTES, ce méta-modèle d'implantation importe le modèle UML de ce dernier.

Chaque implantation d'un système TR²E conforme à notre méta-modèle d'implantation possède un ensemble de processus. Pour cela, nous définissons à la fois les méta-classes **System** et **Process**. Les processus du système communiquent pour échanger des données. La méta-classe **Connector** décrit la communication entre deux processus (émetteur et récepteur) à travers les bus décrits par la méta-classe **Bus**. Chaque bus est caractérisé par un protocole de communication et de transport (énumérations **CommunicationProtocolKind** et **TransportProtocolKind**).

Un processus est composé d'un ensemble de threads définis par la méta-classe **Thread**. Ces threads peuvent être des threads périodiques, sporadiques ou apériodiques définis respectivement par les méta-classes **PeriodicThread**, **SporadicThread** ou **AperiodicThread** qui héritent respectivement des classes **PeriodicTask**, **SporadicTask** et **AperiodicTask** du méta-modèle de l'intergiciel RCES4RTES. Chaque thread est caractérisé par un ensemble de NFP (*Non Functional Properties*) telle que la priorité et possède un ensemble de ports d'entrée et de sortie dont les types sont respectivement les classes **PortIn** et **PortOut** du méta-modèle RCES4RTES. Ces ports permettent d'envoyer et de recevoir des données dont le type est la classe **GeneratedType** du modèle RCES4RTES et via un routeur de port (classe **PortRouter**). Le comportement de chaque thread sera ajouté par le développeur dans l'opération **threadJob** après la génération du code.

Chaque thread est alloué à un processeur choisi en fonction des contraintes d'allocation spécifiées au niveau de conception. La méta-classe **Processor** est caractérisée par la propriété **Frequency**. Tous les processeurs sont reliés par des bus définis par la méta-classe **Bus**.

Les méta-classes suivantes sont également définies : (i) Méta-classe **Deployment** représentant le déploiement du mode initial utilisant la classe **Context** du modèle de l'intergiciel RCES4RTES ; (ii) la méta-classe **TransportHighLevelImpl** gérant à la fois l'envoi et la réception de données pour chaque thread ; et (iii) Méta-classe **Activity** permettant le démarrage des threads système ainsi que le démarrage du thread qui effectue la reconfiguration dynamique du système (instance de la classe **ReconfigurationTrigger** du modèle de l'intergiciel RCES4RTES).

1.5.2 Profil UML d'implantation

Nous proposons un profil UML appelé Profil d'implantation (Figure 1.5) dérivé de notre méta-modèle d'implantation proposé précédemment. Nous définissons à la fois les stéréotypes **System** et **Process**, qui étendent la méta-classe **Package** d'UML pour définir le système comme un ensemble de paquetages. Comme défini dans notre méta-modèle d'implantation, chaque processus est composé d'un ensemble de threads. Pour cela, nous définissons les stéréotypes **PeriodicThread**, **SporadicThread** et **AperiodicThread**. Ces stéréotypes héritent du stéréotype **Thread**, qui étend la méta-classe **Class** UML. Nous définissons également les stéréotypes **Deployment**, **TransportHighLevelImpl** et **Activity**, qui étendent la même méta-classe.

© 2022 Bechir Zalila

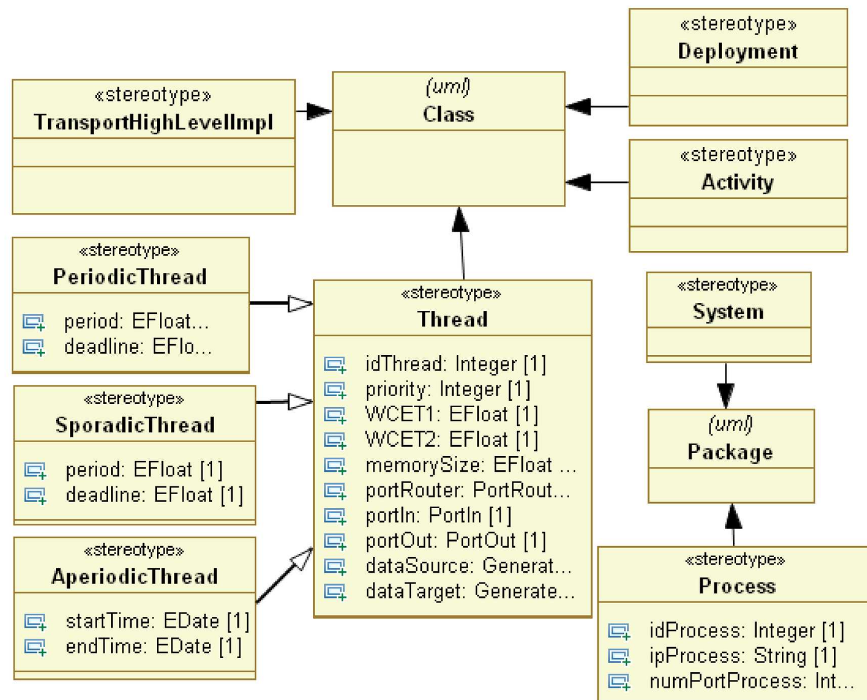

FIGURE 1.5 – Description du profil UML d'implantation [Krichen, 2013]

1.5.3 Synthèse de code

Dans notre approche et comme décrit dans la figure 1.6, nous proposons un générateur de code permettant de générer du code à partir de modèles RCA4RTES. Nous fournissons également un générateur permettant de produire l'ensemble des modes de chaque Méta-Mode conforme aux politiques de reconfiguration spécifiées par le concepteur.

Afin de générer du code, la transformation du modèle RCA4RTES en modèle d'implantation est assurée par des règles définies à l'aide d'Atlas Transformation Language [Jouault *et al.*, 2006]. Afin de générer des modes, nous avons élaboré un algorithme permettant de sélectionner des modes conformes aux politiques de reconfiguration spécifiées dans les modèles RCA4RTES. Cet algorithme permet d'ajouter directement les modes sélectionnés à l'intergiciel RCES4RTES sans passer par le modèle d'implantation [Krichen, 2013].

1.5.4 Outillage

Un framework de modélisation a été proposé pour concevoir des systèmes TR²E reconfigurables. La figure 1.7 montre les outils utilisés pour réaliser ce framework. Notre framework a été développé en utilisant la plateforme ECLIPSE [1]. Nous avons utilisé le plug-in Papyrus [2] comme éditeur graphique UML. Cet éditeur est un éditeur UML où l'on peut intégrer des profils UML. Dans notre méthodologie et afin de pouvoir spécifier des systèmes TR²E reconfigurables, nous avons intégré à la fois les profils RCA4RTES et MARTE.

1. http ://www.eclipse.org
2. http ://www.eclipse.org/modeling/mdt/papyrus

© 2022 Bechir ZALILA

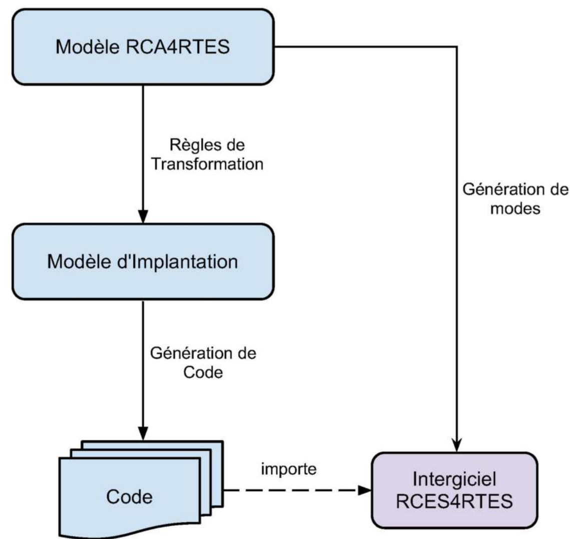

FIGURE 1.6 – Génération de code et de modes [Krichen, 2013]

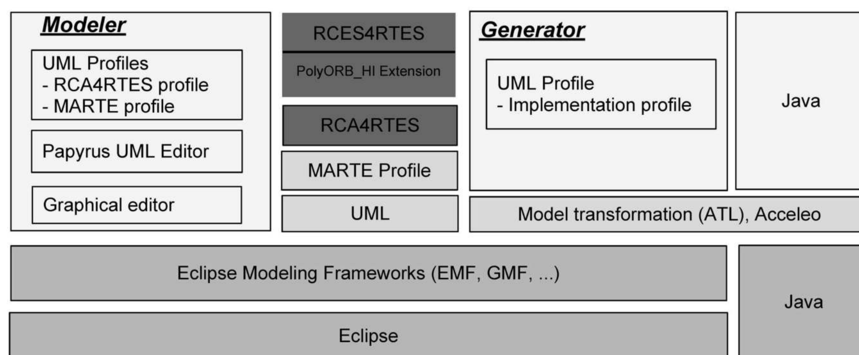

FIGURE 1.7 – Framework de modélisation pour les systèmes TR^{2}E reconfigurables [Krichen, 2013]

© 2022 Bechir Zalila

Nous avons utilisé le langage ATL [3] pour définir un ensemble de règles de transformation du modèle RCA4ERTES vers le modèle d'implantation, et le projet Acceleo [4] pour définir un ensemble de patrons permettant de générer du code à partir du modèle d'implantation.

Notre intergiciel a été implanté en utilisant la spécification temps réel pour Java (RTSJ). Il représente une extension de l'intergiciel PolyORB-HI existant. Nous avons étendu et mis à jour cet intergiciel pour ajouter des routines permettant de supporter les reconfigurations dynamiques ainsi que la vérification de la cohérence et la surveillance des systèmes TR^2E.

1.6 Étude de cas

Nous avons illustré notre approche basée MDE à l'aide d'une étude de cas autour du système de géolocalisation GPS [Krichen *et al.*, 2011]. GPS est un système de radionavigation qui fournit des signaux de navigation précis à n'importe quel endroit sur Terre. Il aide l'utilisateur à déterminer la route à suivre depuis son endroit actuel jusqu'à certaines destinations spécifiées en utilisant les informations fournies par un réseau de satellites. Le satellite envoie à la Terre un signal chiffré qui contient diverses informations utiles à la localisation et à la synchronisation. La base de contrôle envoie (reçoit) des informations aux (des) satellites afin de synchroniser leurs horloges respectives. Nous définissons les cas d'utilisation suivants : (i) GPS à fonctionnement non sécurisé : consiste en une utilisation traditionnelle (ou publique) d'un GPS; et (ii) GPS avec fonctionnement sécurisé : représente une utilisation restreinte d'un GPS avec certaines exigences de sécurité.

En appliquant notre méthodologie de développement décrite dans la section 1.2, nous commençons par définir une machine à états spécifiant les reconfigurations dynamiques par un ensemble de MétaModes et de transitions entre eux. Nous avons défini un diagramme de machine à états UML comme le montre la figure 1.8. Nous spécifions deux MétaModes du GPS : (i) le MétaMode GPS non sécurisé; et (ii) le MétaMode GPS sécurisé. Le passage d'un MétaMode à un autre est initié par l'occurrence d'événements. Par exemple, le passage du MétaMode GPS non sécurisé au MétaMode GPS sécurisé se produit lorsque le moniteur commande de passer à l'état sécurisé.

Les politiques de reconfiguration sont également définies en tant qu'étiquettes du stéréotype **`SoftwareSystem`** (Figure 1.8). Par exemple, la consommation de mémoire ne doit pas dépasser 40% de la taille de la mémoire physique. Les modes conformes aux politiques de reconfiguration spécifiées sont ensuite générés et ajoutés à notre intergiciel.

Dans la deuxième étape, chaque MétaMode doit être décrit, y compris les composants structurés, les connecteurs ainsi que les contraintes non fonctionnelles et structurelles. Par souci de simplicité, de nombreuses fonctionnalités de cette étude de cas ont été omises. Le satellite et la base de contrôle sont représentés par des composants de base (resp. composants **`GpsSatellite`** et **`GpsControlBase`**). Dans ce chapitre, nous décrivons uniquement l'architecture **`GPS_Terminal`** qui se compose de cinq composants pour **`Insecure MetaMode`** :

— Composant **`Position`** pour recevoir le signal satellite,

— Composant **`Receiver`** pour convertir le signal analogique en un signal numérique,

3. http ://www.eclipse.org/atl
4. http ://www.eclipse.org/acceleo

© 2022 Bechir Zalila

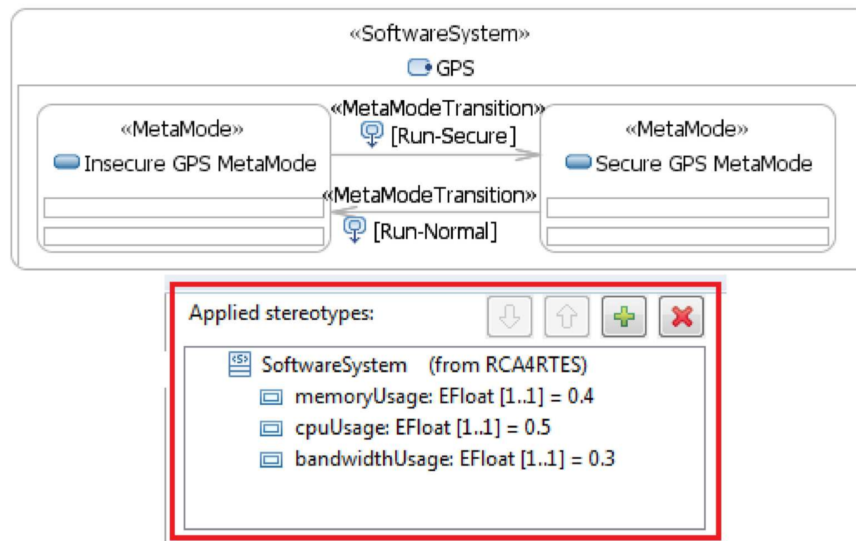

FIGURE 1.8 – Machine à états du système GPS [Krichen, 2013]

TABLE 1.1 – Propriétés non fonctionnelles du système GPS

Composant Structuré	Nature	Période Échéance	$WCET_1$	$WCET_2$	Taille mémoire
Receiver	sporadique	100 ms	20 ms	2 ms	0.9 MB
Position	sporadique	100 ms	20 ms	2 ms	0.5 MB
TreatmentUnit	sporadique	100 ms	20 ms	4 ms	0.75 MB
Decoder	sporadique	100 ms	20 ms	2 ms	0.1 MB
Encoder	sporadique	100 ms	20 ms	0	0.5 MB
GpsSatellite	périodique	400 ms	30 ms	0	0.9 MB
GpsControlBase	périodique	400 ms	30 ms	0	0.9 MB

— Composant **Decoder** pour décoder des informations numériques et séparer les informations pour calculer les informations de distance et de temps,

— Composant **TreatmentUnit** permettant de calculer la distance du satellite afin d'obtenir la position,

— Composant **Encoder** pour l'encodage des informations de temps et de position.

Le passage du MétaMode GPS non sécurisé au MétaMode GPS Sécurisé consiste à supprimer toutes les instances du composant **Position** et à ajouter des instances des composants **SecurePosition** et **AccessController** pour assurer la réception sécurisée et le contrôle du signal satellite. Le tableau 1.1 présente les propriétés des composants, obtenues à la suite d'une simulation.

Dans la troisième étape, nous spécifions l'architecture matérielle fixe suivie de l'affectation de chaque MétaMode à cette architecture. En particulier nous réalisons l'affectation du MétaMode GPS non sécurisé au matériel de terminal GPS et au matériel de satellite GPS. Nous utilisons le profil MARTE pour spécifier l'architecture matérielle. Les contraintes d'allocation décrivent les politiques d'allocation des modèles aux instances matérielles.

© 2022 Bechir Zalila

Par exemple, l’allocation d’instances du composant structuré **Encoder** doit être effectuée soit sur le processeur cpu_1 soit cpu_2 du terminal GPS.

Ensuite et afin de générer du code, le modèle d’implantation sera obtenu en appliquant des règles de transformation.

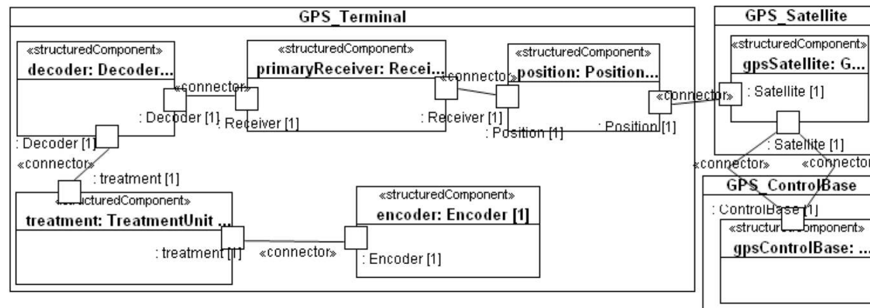

FIGURE 1.9 – Configuration GPS non sécurisé [Krichen, 2013]

Après avoir généré le code, et pour illustrer l’utilisation de notre intergiciel, nous définissons deux configurations de GPS : Configuration GPS non sécurisée (Figure 1.9) du MétaMode GPS non sécurisé comme configuration initiale et Configuration GPS sécurisée (Figure 1.10) du MétaMode GPS sécurisé. La transition entre ces deux métamodes donnera lieu à une transition entre deux modes dans l’implantation du système.

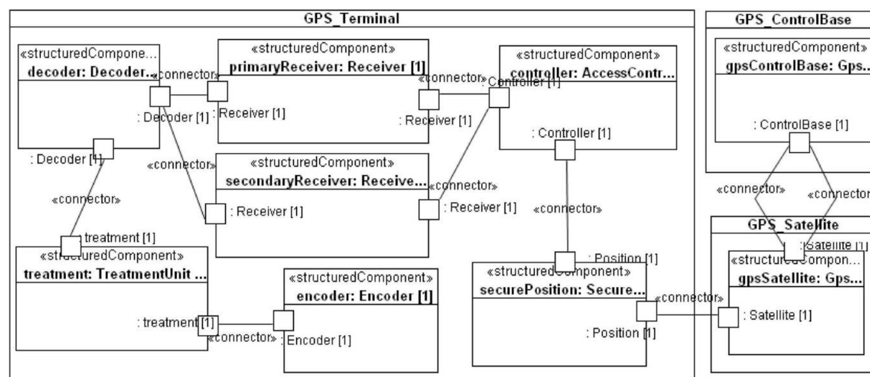

FIGURE 1.10 – Configuration GPS sécurisé [Krichen, 2013]

Notre intergiciel permet la reconfiguration dynamique en faisant passer le GPS de la configuration GPS non sécurisée à la configuration GPS sécurisée lors du lancement de l’événement. Après le passage d’une configuration GPS non sécurisée à une configuration GPS sécurisée, le système reste cohérent et conserve ses contraintes temporelles.

1.7 État de l’art

Dans cette section, nous passons en revue certains travaux connexes qui traitent du développement et de la reconfiguration de systèmes TR^{2}E, des modèles aux plates-formes d’exécution. Plusieurs travaux ont été menées pour concevoir des systèmes embarqués et notamment reconfigurables.

© 2022 Bechir ZALILA

AADL, (*Architecture Analysis & Design Language* [SAE, 2017a]) est un langage de description d'architecture, qui permet la spécification de systèmes TR²E en tant qu'assemblage de composants. Il permet de décrire à la fois les parties logicielles et matérielles d'un système. AADL permet également de spécifier des systèmes reconfigurables utilisant des machines à états composées de modes et de transitions entre mode. Un mode représente un état particulier (configuration) tandis qu'une transition représente un événement, ce qui permet la reconfiguration du système. Par rapport à notre approche, les modes en AADL sont prédéfinis statiquement. Cela réduit considérablement les possibilités de modélisation. AADL spécifie les systèmes embarqués à un bas niveau (thread, processeur, etc.), de sorte que la modélisation des reconfigurations est liée à une plate-forme spécifique.

MARTE [OMG, 2019] est un profil UML pour les systèmes temps réel embarqués inspiré du profil SPT [OMG, 2005]. Il permet la séparation des parties matérielles et logicielles des ressources de la plate-forme et la modélisation des NFP (*Non Functional Properties*). Il présente un ensemble de paquetages qui permettent de spécifier un système à plusieurs niveaux d'abstraction. De plus, MARTE permet de spécifier les reconfigurations comportementales des systèmes embarqués temps réel à l'aide de machines à états composées d'un ensemble de modes et de transitions entre eux. Contrairement à notre approche, MARTE ne supporte pas les systèmes distribués. Il permet de spécifier uniquement les reconfigurations comportementales du système. Il ne gère pas les reconfigurations architecturales.

Pour faire face à la complexité croissante de la conception de systèmes embarqués, plusieurs méthodologies de développement ont été proposées.

Ocarina est un framework qui permet de développer, de configurer et de déployer des systèmes TR²E à l'aide d'une approche basée sur les modèles [Hugues *et al.*, 2008; Lasnier *et al.*, 2009a]. Nous avons réalisé Ocarina dans le cadre de notre travail de thèse [Zalila, 2008] dans un contexte de systèmes TR²E critiques interdisant la reconfiguration dynamique. À l'aide d'Ocarina, les systèmes TR²E peuvent être spécifiés à l'aide du langage AADL. À partir du modèle, Ocarina permet d'effectuer une analyse d'ordonnancement et une vérification pour garantir la validité du modèle. Ensuite, une partie importante du code de l'application ainsi qu'une couche intergicielle dédiée aux besoins spécifiques de l'application seront générées automatiquement. Ocarina permet la génération automatique de code à partir de modèles AADL vers de multiples supports d'exécution. Le déploiement et la configuration de l'application sont effectués automatiquement par l'outil à l'aide d'informations extraites des modèles. Cependant, cet outil n'aborde pas la reconfiguration dans les systèmes TR²E. Par ailleurs, le langage AADL ne permet pas de spécifier les systèmes TR²E à un haut niveau d'abstraction mais plutôt à un bas niveau (thread, processeur, etc.) et cela demande plus de compétence et d'expertise pour spécifier ces systèmes.

Dans la même direction, une approche MDA pour résoudre les problèmes de réutilisabilité, de maintenabilité et de portabilité des logiciels en temps réel est proposée dans [Chehade *et al.*, 2011]. Les auteurs proposent un framework basé sur les modèles et un processus facilitant la conception et la mise en œuvre de systèmes temps réel embarqués. Premièrement, le modèle d'application doit être annoté avec le sous-profil High Level Application Modeling de MARTE [OMG, 2019]. Le modèle de plate-forme cible et le modèle de mapping doivent également être spécifiés. Ensuite, le modèle spécifique à la plate-forme généré est obtenu grâce à des règles de transformation génériques définies. Enfin, le code exécutable est généré. À la suite de ce processus, les concepteurs peuvent obtenir une architecture de système temps réel embarqué qui peut être utilisée pour plusieurs implantations de plates-formes. Cependant, cette approche ne prend pas en considération la reconfiguration des systèmes.

© 2022 Bechir Zalila

ModES [do Nascimento *et al.*, 2007] est également une approche basée sur MDE consacrée à la conception de systèmes embarqués. Elle définit un ensemble de méta-modèles représentant les éléments suivants : (i) **`application`** pour capturer des fonctionnalités au moyen de processus communicants ; (ii) **`platform`** pour indiquer les ressources matérielles/logicielles disponibles ; (iii) **`mappings`** l'application à la plate-forme ; (iv) et **`implentations`**, orientées vers la génération de code et la synthèse matérielle. La particularité de cette approche est que le méta-modèle **`mappings`** ne spécifie pas seulement l'affectation des processus applicatifs à des composants matériels fixes. Cette cartographie délimite également un espace de conception qui correspond à toutes les implantations possibles pouvant être obtenues par le choix de séquences de transformations entre modèles. Par conséquent, l'ensemble des transformations entre modèles implante les projections possibles de l'application à la plate-forme. La méthodologie ModES comprend un ensemble d'outils qui prennent en charge les tâches de conception basées sur des modèles à partir de la spécification jusqu'à la génération et la projection de composants matériels/logiciels. Cependant, cette approche ne prend pas en charge la reconfiguration des systèmes TR^{2}E.

Les approches présentées précédemment proposent des méthodologies de développement qui permettent de concevoir des systèmes embarqués temps réel. Cependant, ces approches ne prennent pas en charge les systèmes reconfigurables. Dans cette direction, TimeAdapt [Fritsch and Clarke, 2008] est une méthodologie de développement pour la conception de systèmes reconfigurables. Elle est conforme à une approche à trois niveaux fournissant les moyens de spécifier les actions de reconfiguration, d'estimer si leur exécution peut être effectuée dans un délai donné et de les exécuter en temps opportun. Chaque reconfiguration a une limite de temps basée sur les conditions environnementales et l'application structurelle. Un test d'admission calcule la probabilité que la reconfiguration donnée puisse respecter les limites de temps spécifiées. Si cette probabilité dépasse un seuil donné, la reconfiguration est ordonnancée sous la forme d'une tâche temps réel de haute priorité et ses actions de reconfiguration seront exécutées par le système. En cas de rejet d'une tâche de reconfiguration, la reconfiguration est reprogrammée avec un nouveau temps limite à un moment ultérieur. TimeAdapt prend en charge l'exécution de reconfigurations sur des applications temps réel basées sur des composants. Cependant, ce framework fournit un temps limite pour chaque reconfiguration. Si une reconfiguration dépasse son temps limite, elle ne sera pas exécutée.

Dans le même contexte, COMDES [Guo *et al.*, 2008] est un framework dédié à la spécification et à la configuration de systèmes temps réel embarqués. En utilisant ce framework, une application embarquée est construite à partir de composants réutilisables implantés sous forme de blocs fonctionnels exécutables. COMDES définit un processus de développement de systèmes embarqués depuis le niveau de conception jusqu'à la production de code d'application. Un système est modélisé à un haut niveau d'abstraction, puis le modèle de sortie sera transformé en un modèle COMDES qui sera généralement enrichi d'informations qui guident la génération de code. Enfin, le code généré est déployé et testé. Ce framework définit deux types de processus : le processus de configuration et le processus de reconfiguration. Le processus de configuration permet de rechercher des composants dans un dépôt de composants puis de les assembler pour configurer un modèle d'application. Un processus de reconfiguration permet d'ajouter, de supprimer et de mettre à jour des composants à l'exécution afin de mettre à jour l'application. Cependant, en utilisant le framework COMDES, le développeur dispose d'un nombre limité de composants pré-

© 2022 Bechir Zalila

construits qui sont stockés dans un dépôt de composants et ne peut pas ajouter, pendant l'exécution, un nouveau composant qui n'existe pas dans ce dépôt.

1.8 Conclusion et perspectives

Dans ce chapitre, nous avons proposé une approche MDE pour concevoir des systèmes TR²E reconfigurables. Nous avons développé une méthodologie de modélisation qui introduit de nouveaux concepts pour concevoir ces systèmes en utilisant des technologies de méta-modélisation et en suivant une approche MDE. Cette méthodologie comporte un processus qui permet de générer facilement une partie importante du code du système TR²E.

Afin d'éviter d'énumérer toutes les configurations possibles du système, nous avons introduit le nouveau concept de MétaMode. Les reconfigurations sont spécifiées à l'aide de machines à états ayant un ensemble de MétaModes et des transitions entre eux. Nous assurons également l'affectation de chaque MétaMode à l'architecture matérielle. Nous avons développé un nouvel intergiciel en tant qu'extension et mise à jour de PolyORB-HI fournissant un ensemble de routines effectuant les reconfigurations dynamiques et assurant la cohérence et la surveillance. Ensuite, nous avons étudié la génération de code s'exécutant à l'aide de cet intergiciel. Un modèle d'implantation est obtenu grâce à des règles de transformation puis un code est généré automatiquement.

Fournir une description de haut niveau dans le processus de construction des systèmes TR²E minimise les erreurs et les efforts du développeur. Notre framework de modélisation permet de spécifier des systèmes autonomes où les reconfigurations seront effectuées sans intervention humaine. Cependant, notre approche n'assure pas la validation sémantique avancée des modèles spécifiés. De plus, notre approche supporte les systèmes embarqués temps réel, mais elle ne supporte pas les systèmes critiques. Il faudrait ajouter des contraintes temporelles qu'il faut respecter pour passer d'un MétaMode à un autre et assurer la bonne exécution de ces systèmes.

Publications

Cette contribution a donné lieu à la publication d'**un** article de revue et de **sept** articles dans des conférences internationales. La liste exhaustive de nos publications est donnée à la fin de ce mémoire.

Perspectives

Comme travaux futurs, nous visons à proposer des modèles qui permettent de spécifier facilement des systèmes TR²E reconfigurables. Ensuite, ces patrons seront traduits en modèles conformes à notre méta-modèle RCA4RTES. De plus, nous visons à enrichir notre framework de modélisation et à ajouter de nouvelles routines à notre intergiciel pour prendre en charge la tolérance aux pannes (cf. chapitre 2). Nous visons également à analyser les systèmes TR²E reconfigurables dès la phase de modélisation en utilisant des formalismes formels (chapitre 4).

© 2022 Bechir Zalila

Chapitre 2

Tolérance aux pannes dans les systèmes TR²E

Sommaire

2.1 Contexte général 27
2.1.1 Problématique et objectifs 27
2.1.2 Notions de base sur la tolérance aux pannes 28
2.2 Approche pour la tolérance au pannes dans les systèmes TR²E 30
2.3 Extension d'un langage d'aspect pour les systèmes TR²E 33
2.4 Génération de code pour les systèmes TR²E tolérants aux pannes 35
2.5 Étude de cas 37
2.5.1 Description 37
2.5.2 Résultats 38
2.6 État de l'art 38
2.6.1 Modélisation de la tolérance aux pannes 40
2.6.2 Tolérance aux pannes en utilisant la programmation orientée aspect 41
2.7 Conclusion et Perspectives 42

Dans ce chapitre, nous détaillons la seconde contribution de notre activité de recherche : la tolérance aux pannes dans les systèmes TR²E. Cette contribution constitue une suite logique de celle du chapitre 1 puisque sa mise en œuvre nécessite un savoir-faire dans la configuration dynamique des systèmes TR²E critiques.

La construction des systèmes TR²E critiques est aujourd'hui une tâche extrêmement fastidieuse en raison des exigences de sécurité et de reconfiguration dynamique et de la complexité toujours croissante de ces systèmes. Pour cela, quelles que soient les précautions prises, l'apparition de fautes dans de tels systèmes est parfois inévitable. Ainsi, les développeurs doivent prendre en compte la présence de ces fautes dès le niveau de conception. Dans ce contexte, nous constatons le besoin de techniques assurant la sûreté de fonctionnement des systèmes TR²E dynamiquement reconfigurables. Nous nous concentrons sur la tolérance aux pannes dans ces systèmes pour leur permettre de fonctionner même en cas de pannes. Dans ce chapitre, nous définissons un processus de développement pour la modélisation et la génération de code supportant la tolérance aux pannes pour les systèmes TR²E en utilisant la programmation orientée aspect. Dans un premier temps, nous intégrons des éléments de tolérance aux pannes dès l'étape de modéli-

sation d'un système afin de profiter des fonctionnalités d'analyse, de preuve et de vérification possibles à ce stade en utilisant le langage AADL et son annexe de modèles d'erreur. Dans un second temps, nous étendons un langage orienté aspect existant et l'adaptons pour se conformer aux exigences temps réel. Enfin, nous définissons un processus de génération de code à la fois pour les préoccupations fonctionnelles et transversales comme la tolérance aux pannes et nous proposons une extension pour un intergiciel existant. Pour valider notre contribution, nous utilisons AADL et ses annexes pour concevoir un système tolérant aux pannes : une couveuse pour nourrissons.

La contribution décrite dans ce chapitre est le fruit d'un travail de recherche d'équipe. Il s'agit des résultats des travaux de thèse de doctorat de Mme Wafa GABSI [Gabsi, 2017] et des projets de mastères de recherches de Mmes Sihem LOUKIL [Loukil, 2010], Rahma BOUAZIZ [Bouaziz, 2012] et Dorra KTARI [Ktari, 2014].

2.1 Contexte général

Une évolution rapide des systèmes TR²E dynamiquement reconfigurables est constatée de nos jours suite à l'émergence de nouveaux besoins dans ce domaine. Dans ce type de systèmes, quelles que soient les précautions prises par les développeurs, l'apparition d'erreurs ne peut être évitée en raison de conditions internes ou externes telles que des défauts d'origine humaine, le vieillissement des équipements, les catastrophes naturelles, etc. Dans certains cas, les défauts peuvent entraîner de graves conséquences telles que la perte d'argent, du temps ou même de vies humaines.

2.1.1 Problématique et objectifs

Les systèmes TR²E doivent faire preuve d'une fiabilité élevée [Avizienis *et al.*, 2004]. La fiabilité est définie comme la capacité d'un système à fournir un service auquel on peut faire confiance à juste titre. Ce concept est entouré de divers attributs qui sont : la disponibilité, la sécurité, l'intégrité et la maintenabilité. Toutes ces propriétés doivent être satisfaites même en présence de menaces de sûreté que sont les fautes, les erreurs et les défaillances. Les moyens dont nous avons généralement besoin pour éviter différents types de pannes incluent la prévention des pannes, la prévision des pannes, la suppression des pannes et la tolérance aux pannes. La tolérance aux pannes (TP ou FT, *Fault Tolerence* en anglais), l'un des différents moyens de fiabilité, est définie comme la capacité d'un système à continuer à fournir les services offerts même en présence d'erreurs [Avizienis *et al.*, 2004].

Basée sur la redondance pour détecter et masquer les erreurs, la TP expose des problèmes majeurs dans le cadre de systèmes TR²E dynamiquement reconfigurables. Premièrement, la tolérance aux pannes dans les logiciels nécessite des tâches de reconfiguration dynamique au moment de l'exécution. Cela peut compromettre la prédictabilité du système. Dans le contexte des systèmes temps réel, ce sera une source de non-déterminisme. Deuxièmement, les mécanismes de TP sont basés sur une certaine redondance. Ils introduisent une complexité et un coût supplémentaires pour la fonctionnalité de base. Ainsi, nous devons vérifier que ces techniques n'introduisent pas un sur-coût très élevé en termes de temps d'exécution et d'empreinte mémoire. Afin de fournir une redondance puissante à moindre coût, nous devons définir les mécanismes de TP appropriés au système. De plus, dans le cadre de systèmes dynamiquement reconfigurables, nous devons

© 2022 Bechir ZALILA

prendre en compte de nouvelles considérations liées à la migration entre configurations. L'adaptation, dans ce cas, peut faire courir le risque d'une défaillance lors d'une reconfiguration. En outre, il existe un manque de techniques permettant de générer du code tolérant aux pannes conforme aux exigences temps réel à partir d'un modèle fiable. Jusque là, les modèles visent uniquement à vérifier formellement certaines propriétés par transformation de modèle.

Pour surmonter ces problèmes, nous présentons dans ce chapitre nos travaux consacrés au support de la tolérance aux pannes pour les systèmes TR²E dynamiquement reconfigurables. Pour ce type d'applications, nous intégrons les éléments TP dans la modélisation d'un système pour bénéficier des fonctionnalités d'analyse, de preuve et de vérification réalisables à ce stade. L'objectif principal de cette approche est d'améliorer la couverture des fautes, assurée par les stratégies de TP dédiées à certaines classes de fautes sélectionnées au niveau de la conception ainsi qu'au niveau de la mise en œuvre. La tolérance aux pannes étant une préoccupation transversale, nous avons décidé d'implanter les différents algorithmes en utilisant la programmation orientée aspect (AOP).

Nous avons défini un processus de développement pour la conception, la mise en œuvre et la génération de code de systèmes TR²E reconfigurables tolérants aux pannes. Nous utilisons le langage AADL pour modéliser notre système de base. Ensuite, nous offrons la possibilité d'étendre ce modèle avec des aspects TP en utilisant des annexes AADL telles que l'annexe de modèles d'erreur [SAE, 2015] et l'annexe comportementale [SAE, 2017b]. Nous proposons ensuite la génération de code pour les parties fonctionnelles et transversales en utilisant différents outils pour avoir enfin un code tolérant aux pannes. Nous avons choisi de générer du code applicatif dans le langage Ada car il est bien adapté pour implanter des systèmes TR²E. Pour des raisons de conformité, nous générons des aspects dans le langage AspectAda, un langage que nous avons étendu dans le cadre de cette contribution. Afin de valider notre approche, nous modélisons, comme étude de cas, une couveuse pour nourrissons en utilisant AADL et ses annexes.

Ce chapitre est organisé comme suit : dans la section 2.2, nous présentons notre approche à travers une description détaillée du processus de production. Ensuite, dans la section 2.3, nous détaillons notre première sous-contribution : extension d'un langage d'aspect pour les systèmes TR²E. La seconde sous-contribution de ce travail, la génération de code tolérant au pannes à partir des annexes AADL sera présentée dans la section 2.4. Dans la section 2.5, nous illustrons notre contribution en l'appliquant dans un exemple réel de système TR²E. Un bref état de l'art est donné dans la section 2.6. Enfin, la section 2.7 conclut ce chapitre et présente quelques perspectives. Mais avant cela, nous présentons, dans la sous-section 2.1.2 quelques notions de base sur la tolérance aux pannes.

2.1.2 Notions de base sur la tolérance aux pannes

Cette section présente les concepts de base (1) des menaces de fiabilité, (2) de la classification des pannes et (3) des techniques de tolérance aux pannes.

Menaces sur la fiabilité

Les menaces principales auxquelles peut faire face un système sont :

— *Faute* : défaut matériel ou logiciel physique ; tout événement, action ou circonstance entraînant une dégradation du service,

© 2022 Bechir Zalila

— *Erreur* : valeur incorrecte provoquant ou non une défaillance du système,
— *Défaillance* : écart du comportement observé du système par rapport aux spécifications.

La propagation des erreurs constitue un risque majeur pour la sûreté de fonctionnement. Une erreur est la manifestation d'une défaillance sur le système. Une défaillance se produit lorsqu'une erreur est propagée à l'interface de service et dévie le service de sa spécification correcte. De plus, étant donné qu'un système informatique est souvent composé de plusieurs sous-systèmes, une défaillance d'un sous-système peut provoquer une panne dans un autre sous-système ou dans le système global.

Classification des pannes

Les pannes sont classées selon huit points de vue de base, conduisant aux classes de pannes élémentaires [Avizienis *et al.*, 2004].

Dans notre contexte, nous nous concentrons sur un ensemble particulier de pannes composé de six classes élémentaires comme le montre le tableau 2.1.

TABLE 2.1 – Classification des pannes

Critère	Types	Description
Objectif	Mal intentionné	Introduit par un humain dans l'objectif de nuire au système
	Non-malveillant	Introduit sans objectif de nuisance
Frontière du système	Interne	Provenant de l'intérieur du système
	Externe	Provenant de l'extérieur du système
Cause phénoménologique	Naturelle	Causée par un phénomène naturel sans intervention humaine
	Homme	Résultant d'actions humaines
Phase de création ou d'occurrence	Conceptuelle	Se produisant au niveau de la conception
	Opérationnelle	Se produisant au niveau de l'exécution
Persistance	Permanente	Supposée exister en permanence
	Transitoire	Dont la présence est limitée dans le temps
Dimension	Logicielle	Originaire ou affectant des composants logiciels (programmes et données)
	Matérielle	Originaire ou affectant des composants matériels (capteurs, actionneurs, etc.)

Techniques de tolérance aux pannes

Afin d'éviter la défaillance du système, la TP est réalisée à l'aide des techniques suivantes :

(1) *Détection d'erreur* : consiste à détecter l'occurrence d'une erreur. Elle est soit concomitante, soit préventive :
    — Détection concomitante : réalisée lors de l'exécution régulière du système,

© 2022 Bechir Zalila

— Détection préventive : réalisée alors qu'une exécution régulière du service est suspendue et profitant du temps pendant lequel le processeur n'a pas de travail à exécuter,

(2) *Recouvrement du système* : consiste à remplacer l'état erroné du système par un autre état sûr. Cette phase repose sur deux mécanismes :

a) Traitement des fautes : empêche l'activation de la panne une fois de plus. Ceci peut être réalisé de différentes manières :

— **Diagnostic :** enregistrer et identifier la cause de l'erreur en termes de type et de localisation,

— **Isolation :** désactiver le défaut en assurant l'exclusion physique ou logique des composants défaillants,

— **Reconfiguration :** attribuer des tâches aux composants non défectueux,

— **Réinitialisation :** vérifier, mettre à jour et enregistrer les nouvelles configurations et données du système,

b) Traitement des erreurs : élimine les erreurs de l'état du système. Dans ce cas, nous devons utiliser la restauration, l'avancement ou la compensation ou combiner certaines d'entre elles dans des situations particulières :

— **Restauration :** faire passer le système de l'état erroné à un état stable antérieur enregistré,

— **Avancement :** corriger l'état du système en le déplaçant vers un nouvel état stable,

— **Compensation :** pour permettre de masquer l'état erroné, le système fournit une redondance suffisante dès le départ. La redondance est une pratique très fréquemment utilisée dans la conception de systèmes critiques ou tolérants aux pannes. Elle implique la réplication de différents composants matériels ou traitements logiques.

2.2 Approche pour la tolérance au pannes dans les systèmes TR²E

Dans le but d'assurer la tolérance aux pannes pour les systèmes TR²E dynamiquement reconfigurables, nous avons défini une approche de développement intégrant les mécanismes TP depuis la phase de modélisation.

Tout d'abord, nous avons sélectionné un sous-ensemble des pannes logicielles traitées dans le cadre de systèmes TR²E dynamiquement reconfigurables, comme le montre la figure 2.1. Les pannes représentées par des cases sombres sont celles que notre processus supporte.

Les pannes malveillantes étant liées à la sécurité et, par conséquent, à l'intention de l'utilisateur, sont en dehors du cadre de notre travail. Nous considérons uniquement les fautes non malveillantes dans notre processus. Nous nous concentrons sur les pannes internes et naturelles dans les systèmes TR²E dynamiquement reconfigurables. Nous considérons également les défauts conceptuels car nous visons à intégrer des éléments TP depuis le niveau de conception. En outre, comme nous concevons la TP pour des systèmes temps réel, le temps de réponse doit être borné et sa borne connue en avance. Par ailleurs, nous nous concentrons sur les deux types de persistance des pannes, permanentes et transitoires. Dans notre cas, nous ne gérons pas les pannes byzantines [Lamport *et al.*, 1982]

© 2022 Bechir Zalila

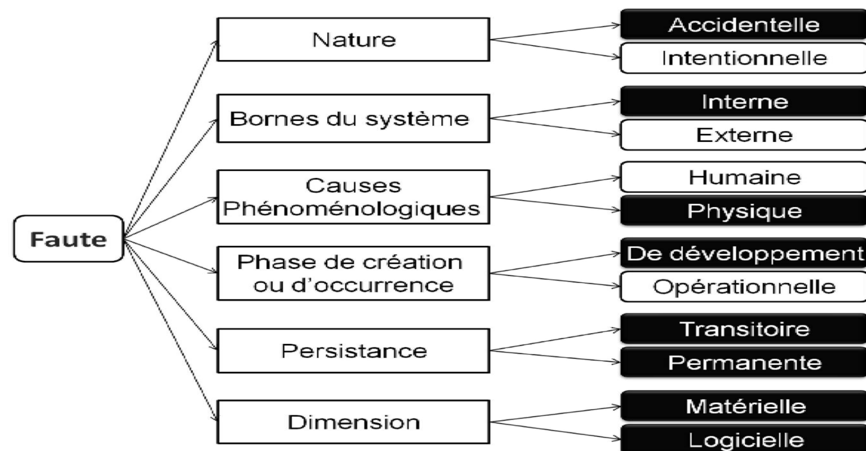

FIGURE 2.1 – Classification des pannes [Gabsi, 2017]

car dans les systèmes temps réel critiques, nous courons le risque de non respect des échéances si nous considérons de telles pannes.

Après avoir établi le classement des pannes, nous avons sélectionné le langage AADL (*Architecture Analysis & Design Language* [SAE, 2017a]) comme langage de description d'architecture pour intégrer les éléments TP au niveau de la conception, et ce, pour les raisons suivantes :

— AADL est un standard international permettant à la fois la représentation textuelle et graphique avec une sémantique d'exécution précise pour les systèmes TR²E. Il a été élaboré pour modéliser des systèmes critiques tels que les systèmes aéronautiques et spatiaux. De plus, AADL est un ADL concret [Medvidovic and Taylor, 2000] : tous les composants d'un modèle AADL correspondent à des entités du monde réel. Aussi, ce langage peut modéliser un système entier comprenant aussi bien des composants logiciels que matériels,

— AADL fournit un support pour décrire les modes opérationnels d'un système et gère par conséquent une certaine reconfiguration dynamique [5]. Les modes et transitions entre modes décrivent les processus de reconfiguration des composants existants. Un mode représente un état opérationnel qui se manifeste sous la forme d'une configuration de composants, de connexions et d'associations de valeurs de propriétés spécifiques au mode. Une configuration est définie statiquement et consiste en un ensemble de composants liés à des connexions. AADL peut être utilisé pour décrire le comportement dynamique de l'architecture d'exécution grâce aux concepts de mode et de transitions de mode. Ainsi, la commutation entre les configurations consiste en une transition entre les modes. Cela n'a besoin que de la spécification des contraintes de reconfiguration pour garantir que la transition de mode nous amène à un nouveau mode correct,

— AADL peut être étendu d'une manière standardisée avec des propriétés pour spécifier les caractéristiques critiques d'un composant dans son contexte architectural à l'aide de propriétés. Les annexes AADL sont également un autre moyen prévu par le

5. Il ne s'agit pas ici d'un support pour la reconfiguration dynamique aussi avancé que celui de notre contribution du chapitre 1. Les modes en AADL doivent être spécifiés d'une manière statique.

© 2022 Bechir ZALILA

standard pour permettre au concepteur d'étendre et de personnaliser la spécification de base AADL avec d'autres concepts spécifiés dans un langage autre que AADL. Les mécanismes de propriétés et d'annexes sont parmi les raisons principales qui nous ont guidés lors de notre choix de langage de modélisation pour les systèmes TR²E reconfigurables tolérants aux pannes.

En recherchant les stratégies de TP adéquates et les meilleures techniques de modélisation pour concevoir les pannes sélectionnées ultérieurement, nous avons décidé d'utiliser l'annexe de modèles d'erreur du standard AADL [SAE, 2015]. Cette annexe permet de modéliser non seulement différents types de pannes, mais également la propagation de ces pannes affectant les composants concernés. En effet, cette annexe permet de concevoir tout type de pannes, comportement défectueux, propagation de pannes, détection de panne mais aussi mécanismes de recouvrement. Le langage est conçu pour être extensible afin de prendre en charge les analyses des architectures d'exécution qu'il ne supporte pas par défaut. Pour étendre ce langage et décrire complètement le système, nous pouvons ajouter de nouvelles propriétés et notations spécifiques à l'analyse qui peuvent être par la suite associées aux composants.

Sur la base du langage AADL et de son annexe de modèles d'erreur, nous avons proposé notre processus de développement intégrant les exigences TP au stade de la modélisation, comme le montre la figure 2.2.

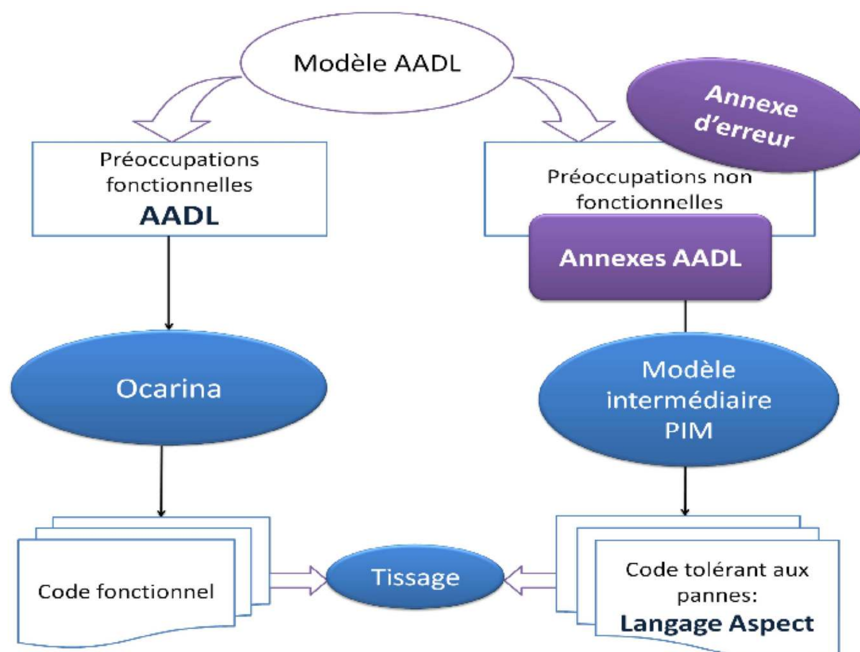

FIGURE 2.2 – Processus de développement proposé [Gabsi, 2017]

Nous utilisons AADL pour modéliser les besoins fonctionnels et non fonctionnels de notre système. En particulier, nous modélisons les exigences de TP que nous concevons en utilisant l'annexe de modèles d'erreur. Pour cela, le concepteur peut utiliser l'outil OSATE [Hecht *et al.*, 2011] qui propose une collection de paquetages de méta-modèles pour permettre au concepteur de modéliser son système central en utilisant le langage AADL. Par la suite, il peut y ajouter des propriétés et une spécification de TP en utilisant une annexe de modèles d'erreur.

© 2022 Bechir ZALILA

Une fois le modèle conçu en utilisant le langage AADL, un processus de génération automatique de code pour obtenir un code adéquat est déclenché en utilisant la suite d'outils OCARINA [Vergnaud *et al.*, 2006]. Cette suite d'outils permet de générer le code correspondant à la partie fonctionnelle d'une spécification AADL dans Ada, C et Java. Ensuite, les systèmes logiciels embarqués peuvent être déployés sur plusieurs plateformes. Nous utilisons OCARINA pour générer le code fonctionnel en langage Ada pour les raisons suivantes :

— Ada est adapté aux systèmes TR²E. Certains profils d'exécution du langage permettent d'avoir des systèmes analysables statiquement et peuvent permettre de prouver certaines propriétés avant l'exécution. Par exemple, le profil Ravenscar [Burns *et al.*, 2004] est un sous-ensemble du langage Ada destiné à modéliser la concurrence dans les systèmes temps réel critiques en imposant des restrictions au moment de la compilation.

— Ada peut être combiné avec diverses technologies pour prouver formellement certaines propriétés du code. SPARK [Barnes, 2003], un système d'annotations pour le langage Ada, fournit des assertions concernant l'état afin d'assurer une preuve et des vérifications par la suite.

La tolérance aux pannes étant une préoccupation transversale, il est tout à fait naturel d'utiliser la programmation orientée aspect pour permettre de spécifier certains aspects liés à la TP dans le modèle AADL. Pour ce faire, nous avons utilisé AO4AADL une annexe du langage AADL développée par notre équipe [Loukil *et al.*, 2013] pour permettre d'insérer dans le modèle des notions liées à la TP.

Dans les sections suivantes, nous décrivons les contributions réalisées afin de produire automatiquement du code source conforme aux exigences des systèmes temps réel à partir des modèles décrits en utilisant le langage AADL et décorés avec l'annexe de modèles d'erreur de ce langage.

2.3 Extension d'un langage d'aspect pour les systèmes TR²E

Comme nous l'avons précisé dans les sections précédentes de ce chapitre, nous avons décidé de mettre en œuvre séparément les préoccupations transversales avec la programmation orientée aspect [Kiczales *et al.*, 1997]. Pour des raisons de conformité avec le langage Ada, le code tolérant aux pannes ainsi que le code des préoccupations non fonctionnelles doivent être générés dans un langage d'aspect compatible avec Ada afin d'être tissés automatiquement dans le code fonctionnel généré. L'implantation existante de ce langage [Pedersen and Constantinides, 2005] présentait certains défauts et lacunes qui nous empêchait de l'utiliser directement. En effet, dans le cadre des systèmes temps réel, l'opération de tissage peut compromettre le déterminisme car elle utilise plusieurs constructions telles que l'allocation dynamique, les priorités dynamiques et l'aiguillage dynamique. Ces constructions rendent l'estimation du temps d'exécution difficilement prévisible et sont malheureusement très abondamment utilisées dans l'implantation existante d'AspectAda et doivent être évitées. Pour cela, nous avons étudié l'implantation existante et extrait ses limites et ses lacunes puis nous avons proposé une restructuration massive et une re-factorisation de la grammaire du langage et du compilateur [Gabsi *et al.*, 2013]. La nouvelle implantation d'AspectAda a été optimisée et étendue pour prendre en charge

© 2022 Bechir ZALILA

l'ensemble de la syntaxe du langage et, en même temps, être conforme aux contraintes des systèmes temps réel.

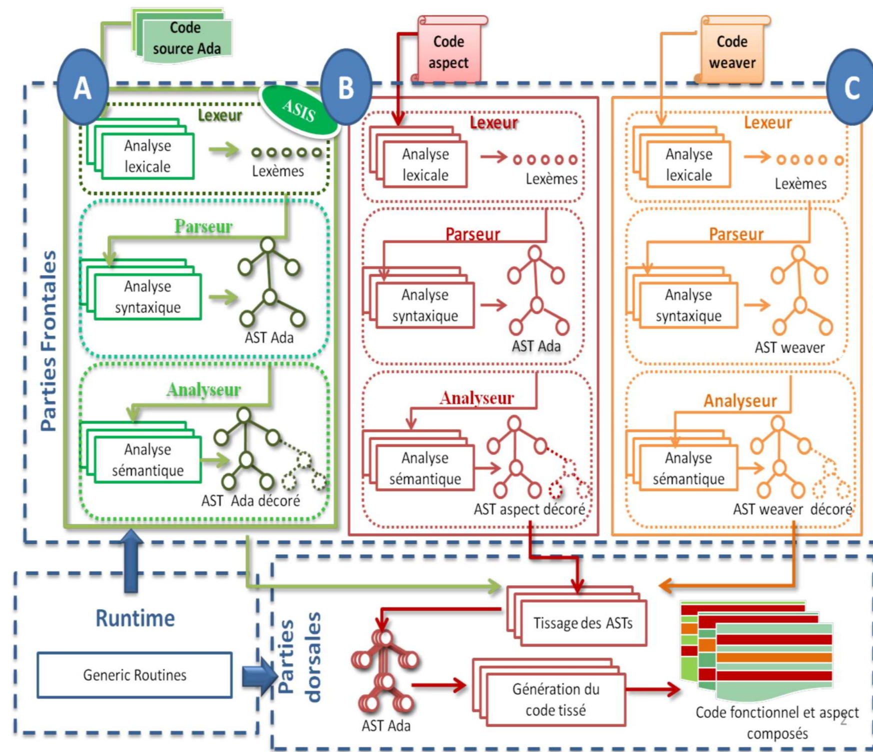

FIGURE 2.3 – Nouvelle architecture pour le compilateur AspectAda [Gabsi, 2017]

Nous avons commencé par étendre AspectAda pour exporter le contexte des *jointures*, des arguments et des valeurs renvoyées, vers le code des *advices*. Nous avons également enrichi sa grammaire pour inclure davantage de concepts du paradigme orienté aspect comme les deux types d'appel et d'exécution de jointures. Ensuite, nous avons défini les règles de tissage du code et de génération en fonction des exigences temps réel en évitant d'utiliser les constructions interdites dans le contexte de ces systèmes. Après cela, nous avons développé les APIs *aspect* et *tisseurs* et étendu la runtime AspectAda. Enfin, nous avons défini une nouvelle architecture pour le compilateur, qui est similaire à l'architecture des compilateurs modernes, comme le montre la figure 2.3. La nouvelle architecture du compilateur AspectAda se compose de trois parties :

— La bibliothèque centrale : définit un ensemble de routines utiles pour la construction et la manipulation d'arbres syntaxiques abstraits (*Abstract Syntax Tree*, AST),

— Les parties frontales (*frontends*) : permettent l'analyse lexicale, syntaxique et sémantique de trois types de code source : les aspects, le tisseur et le code fonctionnel Ada. Chaque partie frontale génère en sortie un AST décoré en utilisant des routines de la bibliothèque centrale,

— La partie dorsale (*backend*) : reçoit trois ASTs résultant de la partie frontale. Ensuite, elle parcourt ces différents AST pour produire un seul AST Ada final à partir duquel le code tissé Ada sera généré.

© 2022 Bechir ZALILA

Du point de vue implantation, nous avons réalisé le développement des trois parties frontales : Ada, la partie aspect et le tisseur. Nous avons également achevé le développement de la partie dorsale permettant la production du code Ada tissé. Cette contribution a donné lieu à la publication d'un article dans une revue internationale de renommée [Gabsi *et al.*, 2020]. Dans la section suivante, nous décrivons la démarche de génération de code à partir du modèle vers ce code orienté aspect Ada/AspectAda.

2.4 Génération de code pour les systèmes TR²E tolérants aux pannes

Notre solution tire profit des extensions possibles des modèles AADL. Elle consiste à enrichir le modèle de base avec l'annexe de modèles d'erreur (EMA) [SAE, 2015]. Cette annexe est dédiée aux erreurs de conception, aux comportements d'erreur, aux comportements des composants en présence d'erreurs, à la propagation d'erreurs ainsi qu'au recouvrement en cas d'erreurs dans le système.

Nous proposons dans cette section une approche qui consiste en la séparation des préoccupations au niveau du modèle et au niveau du code, garantissant la transformation du modèle enrichi de l'annexe de modèles d'erreur vers un langage de programmation orienté aspect. Ceci est basé sur l'enrichissement du code fonctionnel généré à partir de la spécification AADL de base, par des aspects grâce aux générateurs déjà implantés dans la suite d'outils OCARINA. Pour tirer parti des générateurs disponibles, nous avons étendu cette suite d'outils pour assurer la traduction des clauses de l'annexe de modèles d'erreur dans différents langages d'aspect tels que AspectAda dans le code fonctionnel généré en Ada.

Cette transformation n'est pas directe et passe par une étape intermédiaire de traduction du modèle d'erreur vers un modèle AO4AADL [Loukil *et al.*, 2013] comme le montre la figure 2.4. L'avantage de cette étape intermédiaire est de tirer profit de la partie AO4AADL déjà implantée dans la suite OCARINA et aussi d'avoir un modèle intermédiaire avec sa partie orientée aspect qui est complètement indépendant de la plateforme d'exécution (PIM, *Platform Independant Model*).

Notre approche, illustrée dans la figure 2.4, propose un processus de développement pour produire une application tolérante aux pannes réutilisable et modulaire en utilisant AADL et ses annexes à travers quatre étapes :

(1) Conception du modèle AADL de base : le concepteur commence par modéliser son système de base à l'aide d'AADL. A ce stade, il définit les types de composants, leurs implantations, leurs propriétés et établit des connexions pour décrire la hiérarchie des composants de son modèle.

(2) Conception du modèle d'erreurs : le concepteur enrichit son modèle de base en intégrant des stratégies de tolérance aux pannes et plus généralement des exigences de sûreté de fonctionnement au travers des paragraphes de modèle d'erreur (EMA, plus précisément, la version 2 de cette annexe standard, EMV2). À ce niveau, le concepteur est en mesure de corriger l'ensemble des erreurs attendues pouvant être détectées ou propagées via les ports en entrée ou en sortie. Le comportement du composant AADL durant une phase d'erreur est également décrit par des états et des transitions. Cette extension du modèle de base peut être vérifiée à l'aide d'OCARINA avant de passer à la génération de code. S'il y a des erreurs dans le modèle, le concepteur est guidé pour les corriger.

© 2022 Bechir ZALILA

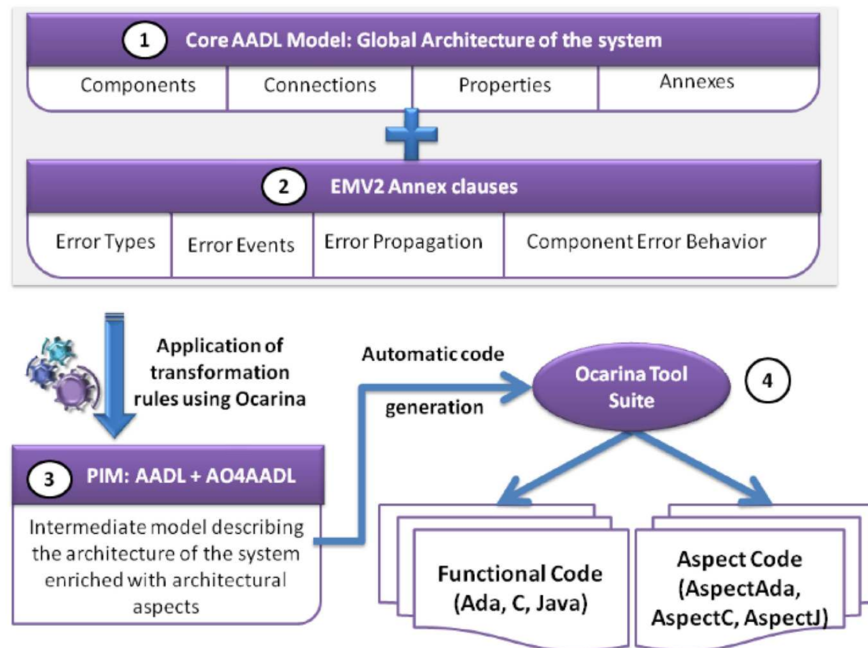

FIGURE 2.4 – Processus de génération de code [Gabsi, 2017]

(3) Génération d'un modèle intermédiaire indépendant de la plate-forme (PIM) : il s'agit d'un modèle AADL décrivant l'application, enrichi par des clauses AO4AADL pour décrire les aspects architecturaux. Ce modèle intermédiaire est généré suite à une transformation de modèle de l'annexe AADL EMV2 à notre annexe aspect AADL AO4AADL. Le but est d'obtenir un code d'aspect générique et de bénéficier des générateurs disponibles d'AO4AADL vers divers langages d'aspect, en particulier AspectAda. Les règles de transformation définies sont implantées dans la suite d'outils OCARINA. Ainsi, nous offrons à l'utilisateur la possibilité de transformer les clauses EMV2 en clauses d'aspect pour intercepter les erreurs dans le modèle de base, réparer les pannes, suivre les propagations d'erreurs et surveiller l'application en présence d'erreurs depuis le niveau de conception.

(4) Génération de code à la fois fonctionnel et d'aspect : une fois que le concepteur a franchi l'étape de conception, il produit du code fonctionnel à l'aide d'OCARINA. Le code fonctionnel est généré en Ada, C ou Java grâce aux générateurs disponibles selon le choix du concepteur. Après cela, les aspects AO4AADL doivent être générés dans un langage spécifique lié au code fonctionnel pour garantir la cohérence de l'application. Les spécifications AO4AADL sont générées dans AspectAda, AspectC ou AspectJ pour être tissées avec le code fonctionnel dans le langage correspondant.

Nous avons défini et implanté dans la suite d'outils OCARINA un ensemble de règles de transformation à partir de EMV2 vers AO4AADL. Ces règles couvrent les transformations suivantes :

— Transformation des types d'erreur : un type d'erreur indique le type de panne activée, d'erreurs propagées ou d'erreur provoquant une transition dans une machine d'état de comportement d'erreur d'un système ou d'un composant. Les types d'erreurs peuvent être organisés en hiérarchies de types d'erreurs ou en ensembles

© 2022 Bechir ZALILA

d'erreurs. Ces types donnent lieu à la génération de modes et d'événements dans le modèle.

— Transformation de la propagation d'erreur : une déclaration de propagation d'erreur spécifie que les erreurs de types spécifiques sont propagées dans ou hors d'un composant via une fonctionnalité ou une liaison. Pour cela, un point de propagation d'erreur au niveau du modèle doit être intercepté au niveau du code. Ainsi, il correspond à une déclaration d'une jointure dans le langage d'aspect. Dans ce cas, nous devons intercepter soit la valeur envoyée par un port de sortie, soit la valeur reçue par un port d'entrée.
— Transformation des réplications : les réplications donnent lieu à la génération, dans le modèle intermédiaire, de copies du composant répliqué ainsi que de toutes les connexions nécessaires à ce composant.

Ces règles de transformation ainsi que les règles de génération de code ont été abondamment décrites dans [Gabsi, 2017] et ont donné lieu auparavant à la publication d'un article dans une conférence internationale de renommée [Gabsi *et al.*, 2016]. Nous avons illustré et validé ces règles à travers une étude de cas : une couveuse de nouveaux nés. La description de cette étude de cas fera l'objet de la section 2.5.

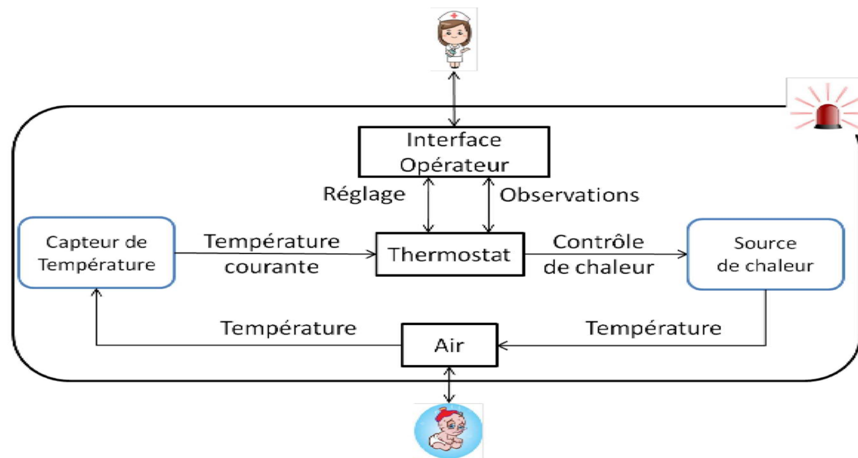

FIGURE 2.5 – Description globale du thermostat de l'Isolette [Gabsi, 2017]

2.5 Étude de cas

Pour valider notre approche, nous avons appliqué les règles de transformation proposées d'EMV2 à AO4AADL à un système médical critique. Il s'agit d'une couveuse pour nourrissons appelée Isolette. Cet exemple a été imaginé (dans sa version AADL annotée par des erreurs) dans [Larson *et al.*, 2013] afin d'illustrer les capacité d'expression de l'annexe de modèles d'erreur, sans donner aucune suite d'implantation après. Nous l'avons reprise et enrichie dans le cadre de notre contribution.

2.5.1 Description

Une sonde de température prend la valeur de la température de l'air et l'envoie au thermostat de l'Isolette. Ce dernier contrôle une source de chaleur afin de produire une

© 2022 Bechir ZALILA

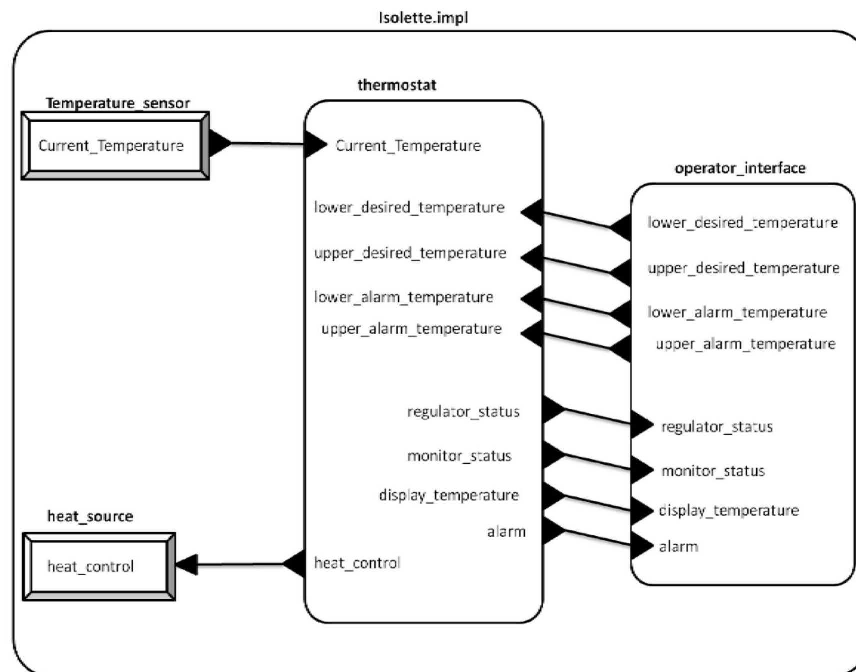

FIGURE 2.6 – Modèle AADL du système Isolette [Gabsi, 2017]

température de l'air dans une plage cible bien déterminée. Ceci est effectué afin de garantir que la température de l'air à l'intérieur de l'Isolette est toujours adaptée pour éviter de nuire au nourrisson. Cette plage est fixée par le clinicien via l'interface opérateur. Si la température détectée est trop chaude ou trop froide, le système Isolette déclenche une alarme par un sous-système distinct du thermostat de fonctionnement. La communication entre les différents composants du système d'Isolette est illustrée à la figure 2.5.

Une vision graphique globale du modèle AADL de l'Isolette est données dans la figure 2.6.

2.5.2 Résultats

Après avoir analysé le modèle avec OCARINA, nous appliquons nos règles de transformation de EMV2 vers AO4AADL, nous obtenons le modèle AADL intermédiaire dont une vision globale graphique est donnée dans la figure 2.7.

11cm

Ce modèle sera le point d'entrée pour les générateurs de code fonctionnel et d'aspect. Nous avons généré du code source Ada et AspectAda et vérifié à travers une simulation que notre application fonctionne est que les erreurs supportées ont bien été détectées et traitées. L'analyse détaillées des résultats de cette étude de cas est donnée dans [Gabsi, 2017].

2.6 État de l'art

Dans cette section, nous donnons un aperçu des travaux de recherche liés à la modélisation des mécanismes de tolérance aux pannes. Ensuite, nous présentons certains

© 2022 Bechir ZALILA

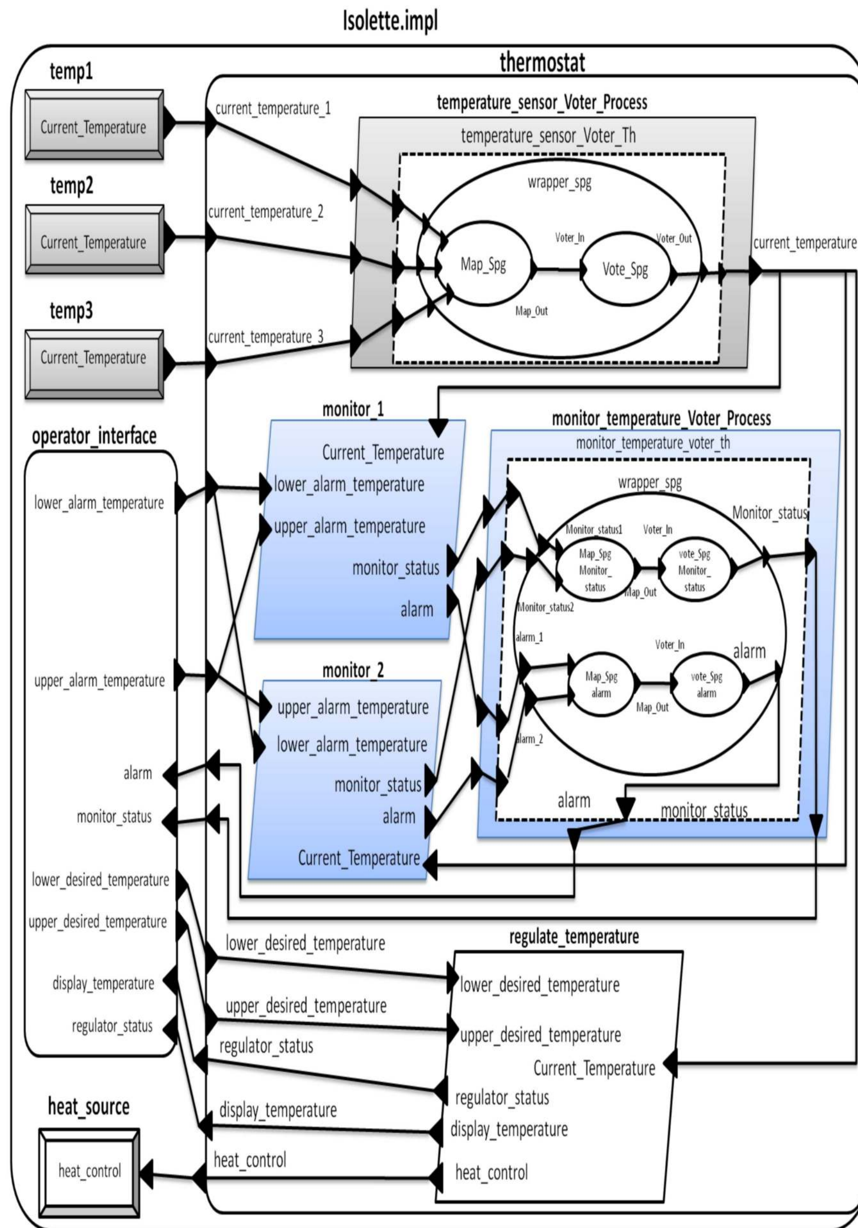

FIGURE 2.7 – Modèle intermédiaire répliqué du système Isolette [Gabsi, 2017]

travaux utilisant la programmation orientée aspect (POA) à des fins de TP étant donné que cette dernière est une préoccupation transversale.

2.6.1 Modélisation de la tolérance aux pannes

Les aspects de tolérance au panne ont souvent été introduits très tard dans le cycle de construction des systèmes TR²E et ce, jusqu'à ce que l'approche MDE soit suffisamment mature.

Dans [Chen and Kulkarni, 2011], les auteurs ont proposé la modélisation de certains types de pannes de manière uniforme en utilisant des systèmes de transition afin d'appliquer efficacement les techniques de vérification et de révision de modèle. Dans chacune des 31 catégories de la taxonomie des failles [Avizienis *et al.*, 2004], ils se concentrent sur la manière dont les pannes peuvent être modélisées et si cela est réalisable en utilisant des systèmes de transition. Après avoir étudié les catégories une à une, ils ont conclu qu'une telle modélisation était faisable mais non pratique pour 2 catégories. De plus, ce n'est pas faisable pour 11 catégories. Ces 13 (2 + 11) catégories regroupent toutes les combinaisons possibles de défauts internes. Elles présentent une augmentation significative des états atteignables et nécessitent la conception d'un comportement continu. Les 18 catégories restantes, pour lesquelles cette approche est faisable et pratique, sont toutes des pannes externes. Comme nous traitons les pannes internes en considérant les systèmes embarqués, nous ne pouvons pas utiliser les systèmes de transition comme technique de modélisation pour modéliser la TP des systèmes TR²E.

D'autres points de vue se sont concentrés sur la modélisation de techniques tolérantes aux pannes à l'aide de langages de description d'architectures comme AADL ou d'autres sur une approche axée sur les modèles pour définir un processus de développement tolérant aux pannes pour les systèmes distribués. Dans [Hamid *et al.*, 2008b; Hamid *et al.*, 2008a], les auteurs ont étendu un framework FT-CORBA en prenant en charge les mécanismes de tolérance aux pannes tels que la détection de pannes et la redondance. Ainsi, ils ont défini un processus de modélisation afin de pouvoir concevoir des composants TP et générer leur code correspondant. Ce processus de modélisation est basé sur les modèles de composants UML et CORBA (CCM) ainsi que sur les profils de qualité de service et de tolérance aux pannes (QoS et FT). Après cela, ce modèle peut être déployé et exécuté au moyen de fichiers descripteurs CCM. Pourtant, cette approche ne prend pas en charge la reconfiguration dynamique des composants TP. De plus, les auteurs n'ont pas évalué la mise en œuvre de leur framework avec différentes stratégies de TP. Ils ont limité leur test à un style de réplication unique qui est le style actif avec vote.

Dans [Gilles *et al.*, 2010], les auteurs ont utilisé AADL pour modéliser les systèmes TR²E critiques. Ils ont également utilisé l'annexe comportementale qui étend AADL avec raffinement des aspects comportementaux exprimés sous la forme d'un automate à transition d'état avec gardes et des actions utilisant des variables pour manipuler les données. Ils ont intégré les préoccupations de la TP en appliquant des modèles de réplication afin de mettre en œuvre des systèmes TR²E. Pour exprimer les capacités de modélisation d'AADL et de son annexe, les auteurs ont envisagé une conception simplifiée du modèle de réplication de sauvegarde principale (*Primary Backup Replication*, PBR). Ils ont conçu avec succès la stratégie PBR avec l'intégration d'un ensemble de règles sémantiques très strictes qui sont cohérentes avec le langage de base au niveau architectural. Cependant, contrairement à l'annexe de modèles d'erreur, l'annexe comportementale n'est pas assez riche et

© 2022 Bechir Zalila

sophistiquée pour la conception de composants tolérants aux pannes et son utilisation dans ce contexte est très fastidieuse.

Dans [Rugina, 2007], l'auteur a défini un nouveau processus de modélisation fiable qui décrit l'architecture d'un système TR²E et ses informations de fiabilité séparément en utilisant respectivement AADL et son annexe de modèles d'erreur. Dans ce contexte, l'auteur a défini un processus de modélisation basé sur la transformation de modèle afin de concevoir des systèmes fiables et sécurisés. Ensuite, elle a annoté le modèle architectural avec des constructions de l'annexe de modèles d'erreur. Après la définition des règles de transformation du modèles, l'auteur permet la génération de réseaux de Petri Stochastiques Généralisés (GSPN) à partir de modèles AADL pour les valider par rapport à sa spécification, sur la base de l'analyse et de la validation du modèle GSPN, après chaque itération. Néanmoins, cette approche n'offre pas la génération de code de mécanismes TP. Elle s'arrête à l'étape de vérification du modèle AADL, qui, quoique très importante, ne permet pas d'obtenir un système opérationnel tolérant aux pannes.

2.6.2 Tolérance aux pannes en utilisant la programmation orientée aspect

La TP étant une occupation transversale dans un système, il est tout à fait naturel de penser à la programmation orientée aspect pour l'implanter. Nous donnons dans cette partie un survol des travaux utilisant la POA pour implanter la TP.

Les auteurs dans [Alexandersson and Karlsson, 2011] ont proposé une étude d'injection de fautes qui estime la couverture de fautes de deux logiciels implantés avec POA. Ils ont étudié la vérification du flux de contrôle à double signature (*Double Signature Control Flow Checking*, DSCFC) et les mécanismes d'exécution redondante triple avec récupération en avant (*Triple Time Redundant execution with Forward Recovery*, TTRFR). Ils les ont également implantés en utilisant trois langages différents pour prouver l'importance des langages orientés aspect pour réduire les sur-coûts. Cependant, cette méthodologie est statique et par conséquent, non adaptée aux systèmes dynamiquement reconfigurables. En outre, elle présente le problème du *glue code* qui est généré par le tisseur du langage orienté aspect, qui n'est pas visible dans les sources du programme et qui échappe ainsi au contrôle du développeur et présente une source de panne possible en soit.

D'autres méthodologies tolérantes aux pannes ont été mises en œuvre à travers un framework pour les systèmes TR²E tolérants aux pannes basés sur l'AOP [Afonso *et al.*, 2008] : les blocs de récupération (*Recovery Blocks*, RB), les blocs de récupération distribués (*Distributed Recovery Blocks*, DRB) et la programmation en N-versions (*N-Version Programming*, NVP). La deuxième méthodologie est la version distribuée de la première qui est basée sur la restauration par annulation alors que la troisième méthodologie est basée sur le masquage des erreurs et le vote majoritaire pour sélectionner la bonne réponse. Cette dernière est préférée aux précédentes car elle est adaptée non seulement aux systèmes temps réel mais aussi aux systèmes distribués dynamiquement reconfigurables. Ceci est réalisé grâce à une communication transparente entre les nœuds et basée sur le paradigme publier/souscrire. Toutefois, cette méthodologie s'intègre mal dans une démarche MDE puisque le processus proposé ne prévoit pas de phase de modélisation.

Dans [Hameed *et al.*, 2009], les auteurs ont proposé un framework orienté aspect pour la détection d'erreurs, les mécanismes de recouvrement et un framework de conception de gestion des exceptions qui est basé sur des contrôles de plausibilité. Les modèles de conception orientés aspect apportés par ce framework offrent des avantages supplémentaires telle que la localisation du code de gestion des erreurs en termes de définition,

© 2022 Bechir Zalila

d'initialisation et d'implantation. Cette méthodologie garantit la réutilisation et la portabilité du framework grâce à la mise en œuvre séparée du traitement des erreurs avec injection de fautes permanentes. Néanmoins, elle n'est adaptée ni aux systèmes embarqués ni aux systèmes temps réel. En effet, elle utilise des capteurs *"best effort"* pour détecter les pannes.

Finalement, les auteurs dans [Szentivanyi and Nadjm-Tehrani, 2004] ont proposé d'utiliser la POA afin d'améliorer les performances des services TP dans FT-CORBA. Ils assurent une évaluation quantitative des gains en performance qui affirment que le sur-coût diminue au moment de l'exécution en le comparant aux autres implantations étudiées. Cette approche offre une implantation séparée des mécanismes de journalisation et de synchronisation. Elle apporte également une optimisation des intercepteurs dans l'infrastructure FT-CORBA à la gestion des exceptions au niveau applicatif. Cependant, comme la précédente, cette approche n'est adaptée ni aux systèmes temps réel ni aux systèmes embarqués car l'intergiciel FT-CORBA possède une empreinte mémoire très élevée. De plus, l'intégration d'aspects dans l'infrastructure FT-CORBA peut compromettre la sûreté de fonctionnement et rendre plus difficile la maintenance. Enfin, cette approche, comme toutes les précédentes, ne propose pas de processus de modélisation TP.

La principale différence entre les approches et les méthodoligies examinées ci-dessus et notre méthodologie réside dans la mise en accent sur :

— L'adaptabilité aux systèmes TR^2E dynamiquement reconfigurables,
— L'intégration de la POA dès la phase de modélisation et la génération automatique de code orienté aspect par la suite.

Dans notre travail, nous avons modélisé des éléments de tolérance aux pannes dans le respect des contraintes temps réel et des politiques de tolérance aux pannes. Par rapport aux profils UML QoS et FT, qui sont limités aux architectures FT-CORBA, notre travail offre à l'utilisateur la liberté de concevoir différents types d'erreurs et de modéliser des architectures logicielles et matérielles en utilisant AADL et son annexe de modèles d'erreur et de spécifier des comportements sémantiques en utilisant l'annexe comportementale. Il offre également les politiques de reconfiguration dynamique grâce à la spécification AADL des modes et des transitions entre modes.

2.7 Conclusion et Perspectives

Dans ce chapitre, nous avons présenté un processus de développement pour intégrer les préoccupations de tolérance aux pannes depuis l'étape de modélisation. Nous avons choisi le langage AADL pour modéliser notre système car il donne aux utilisateurs la possibilité de décrire des préoccupations fonctionnelles ainsi que des préoccupations transversales comme la tolérance aux pannes. Nous avons décidé de modéliser cette dernière à l'aide de l'annexe de modèles d'erreur (EMV2) qui propose la spécification de toutes les classes de pannes, leur propagation, leur détection ainsi que leur recouvrement. Après cela, nous avons proposé et implanté des générateurs de code. Nous avons considéré la programmation orientée aspect pour la mise en œuvre des exigences TP en étendant et en optimisant le langage d'aspect AspectAda par rapport aux exigences en temps réel. Nous avons implanté notre compilateur AspectAda et testé sa conformité aux contraintes temps réel. Nous avons par la suite enrichi la suite d'outils OCARINA pour générer un modèle

© 2022 Bechir ZALILA

AADL intermédiaire annoté par l'annexe AO4AADL et complété la génération de code pour produire du code source Ada/AspectAda.

Ensuite, pour valider notre méthodologie, nous avons présenté un modèle AADL pour une couveuse de nourrisson (Isolette) et testé nos générateur de code avec nos outils implantés.

Publications

Cette contribution a donné lieu à la publication de **trois** articles de revues, de **sept** articles dans des conférences internationales et d'**un** chapitre de livre. La liste exhaustive de nos publications est donnée à la fin de ce mémoire.

Perspectives

Comme futurs travaux à cette contribution, nous visons à implanter un processus de vérification formelle sur le modèle AADL intermédiaire afin de garantir l'exactitude de nos modèles. Par la suite, nous comptons achever le support de la génération de code C et Java qui est encore dans une phase embryonnaire. Enfin, les générateurs de code à partir de ce modèle doivent être certifiés afin que le code source généré soit lui aussi correct par construction.

© 2022 Bechir Zalila

Chapitre 3

Optimisation multi-objectifs des systèmes TR²E

SOMMAIRE

3.1 CONTEXTE GÉNÉRAL 45
3.1.1 Problématique et objectifs 45
3.1.2 Contributions 46
3.2 EXPLORATION MOO DE CONCEPTIONS DES SYSTÈMES TR²E 48
3.2.1 Description de la solution 50
3.2.2 Formulation d'une PAES pour l'affectation 51
3.2.3 Règles d'affectation des fonctions aux threads 55
3.2.4 Tests de faisabilité sur les solutions candidates 56
3.3 PROTOTYPE IMPLANTÉ DANS CHEDDAR 57
3.4 ÉTUDES EXPÉRIMENTALES 59
3.5 ÉTAT DE L'ART 60
3.6 CONCLUSION ET PERSPECTIVES 63

Après avoir supporté la reconfiguration dynamique et la tolérance aux pannes dans les systèmes TR²E, nous nous sommes penchés sur la possibilité d'optimiser ce type de systèmes afin de maximiser la charge de travail possible sur une configuration matérielle donnée, tout en garantissant le respect des contraintes temps réel du système. Jusque là, les modèles architecturaux traités contenaient d'une manière statique l'affectation des composants logiciels (essentiellement les threads) sur les composants matériels (les processeurs, les bus, etc.), mais aussi l'affectation des fonctionnalités dans chacun des threads, l'affectation des fonctions aux threads du système d'exploitation temps réel cible faisant partie intégrante du processus de conception. Or, trouver une affectation appropriée implique de nombreuses et importantes décisions de conception qui ont un impact important sur la qualité et parfois même sur l'ordonnançabilité du système.

Cependant, avec la complexité et l'échelle croissantes des systèmes TR²E actuels et le grand nombre de solutions de conception possibles, prendre des décisions de conception tout en équilibrant des critères de qualité contradictoires est une tâche très fastidieuse et constitue une source d'erreurs pour les concepteurs. Dans le cadre de la contribution décrite dans ce chapitre, nous proposons une méthode automatisée utilisant un algorithme

évolutif multi-objectifs guidé par une technique de regroupement architectural. Cette méthode permet aux concepteurs de chercher dans l'espace de conceptions possibles des solutions pouvant être ordonnancées en fonction de plusieurs critères de performance concurrents. Pour sélectionner les fonctions-objectifs adéquates pour notre optimisation, plusieurs expérimentations ont été réalisées afin de minimiser le nombre de fonctions à prendre en compte.

Cette contribution est le fruit d'un travail de recherche d'équipe. Il s'agit des résultats des travaux de thèse de doctorat de Mme Rahma Bouaziz [Bouaziz, 2018] réalisée dans le cadre d'une collaboration entre l'Université de Sfax et l'Université de Bretagne Occidentale.

3.1 Contexte général

Les systèmes TR²E critiques sont souvent conçus selon des architectures multi-threads. Ces threads interagissent généralement les uns avec les autres et partagent l'accès aux ressources du système. Dans un contexte temps réel, la communication et la synchronisation des threads doivent être traitées avec beaucoup de soin afin de préserver le déterminisme du système.

Les concepteurs sont censés fournir l'association appropriée des fonctions du système aux ressources matérielles qui exécuteront ces fonctions. De plus, ils doivent affecter les fonctions aux threads du système d'exploitation temps réel (RTOS), tout en s'assurant de la non-violation des contraintes du système aussi bien que de son bon comportement. La vérification des contraintes temporelles (analyse d'ordonnançabilité) et les exigences de communication et de synchronisation font partie intégrante du processus de conception.

Ce chapitre porte sur l'exploration de conceptions des systèmes temps réel critiques. De tels systèmes doivent répondre à des contraintes temporelles strictes imposées par leur environnement. Les défaillances, y compris le non-respect des contraintes de temps, pourraient entraîner de graves dommages matériels, voire des pertes de vies humaines.

3.1.1 Problématique et objectifs

Malgré les avancées significatives dans le développement de systèmes TR²E critiques, surtout avec l'utilisation des ADLs et les possibilités de génération de code et d'automatisation que ces derniers offrent, la modélisation de ces système reste une tâche délicate. En effet, lors de la conception d'architectures logicielles pour des systèmes TR²E critiques, les concepteurs doivent prendre de bonnes décisions concernant plusieurs critères de performance concurrents : l'amélioration d'un critère peut conduire à la dégradation d'un autre. Les critères de performance auxquels nous nous intéressons ici sont ceux issus du domaine de l'ordonnancement, tels que le nombre de préemptions, le nombre de changements de contexte, les laxités des threads, les temps de blocage de ces derniers, etc.

En effet, affecter les spécifications fonctionnelles à l'architecture logicielle est une tâche non triviale en raison du nombre de fonctions impliquées et du nombre important de solutions d'affectation possibles, allant des architectures mono-tâche aux architectures multi-tâches concurrentes. Étant donné un ensemble de fonctions, la taille de l'espace de solutions d'affectation des fonctions aux threads est définie par le nombre de Bell [Rota, 1964] qui croît d'une manière exponentielle par rapport au nombre de fonctions. Par conséquent, une recherche exhaustive et exacte parmi toutes les possibilités n'est pas envisageable à partir de quelques dizaines de threads.

© 2022 Bechir Zalila

Les décisions de conception ont une forte influence sur les performances du système final. Considérons les deux stratégies extrêmes d'affectation des fonctions aux threads :

(1) La première solution extrême est la solution de mise en œuvre à thread unique dans laquelle l'ensemble de la spécification fonctionnelle est exécutée dans le contexte d'un unique thread. Bien que cette solution soit bien adaptée aux applications fortement contraintes en mémoire, elle conduit à une conception moins flexible et plus coûteuse à modifier lorsque les fonctions du système évoluent.

(2) L'autre extrême est l'attribution individuelle des fonctions aux threads : que chaque fonction est attribuée à un seul thread. Contrairement à la première solution extrême, cette architecture se traduit par une conception beaucoup plus flexible puisque cette dernière offre une plus grande laxité globale du système permettant des modifications plus aisées de la conception actuelle ou son extension par des fonctions potentielles supplémentaires. La laxité (ou encore la marge) représente une mesure de la flexibilité disponible pour l'ordonnancement d'un seul thread. Elle est définie par le temps qui s'écoule entre la fin de l'exécution du thread et son échéance. Cependant, cette seconde solution conduit à un nombre de threads significativement plus élevé. Cela augmente la consommation de mémoire (par exemple la taille de la pile d'exécution pour chaque thread) en plus de la surcharge excessive de l'ordonnanceur due au changement de contexte ou aux préemptions.

Par conséquent, les architectes de l'application doivent explorer plusieurs alternatives d'architecture afin de sélectionner celles qui répondent au mieux au compromis entre les critères de performance sélectionnés. Cependant, trouver manuellement des solutions d'architecture appropriées prend du temps et peut constituer une source d'erreurs. Les facteurs qui empêchent les concepteurs de gérer efficacement les processus d'exploration et de prise de décision sont la complexité et la taille toujours croissantes des systèmes actuels, le grand nombre d'options de conception possibles, les exigences de conception strictes et les critères de performance souvent contradictoires.

La problématique que nous traitons dans ce chapitre est la suivante : *étant donné une spécification fonctionnelle, comment affecter les fonctions aux threads afin de produire les meilleures alternatives de conception au regard d'un ensemble de critères de performance parfois conflictuels ?* L'affectation de fonctions à des threads n'est pas une activité anodine. D'une part, elle pose généralement des problèmes de complexité combinatoire. D'autre part, son influence sur les performances de l'application résultante est difficilement prévisible. Par ailleurs, elle a un impact sur l'ordonnançabilité du système et nécessite donc, la vérification de la faisabilité des alternatives de conception. Ces problèmes motivent l'automatisation de l'exploration de la conception qui aiderait non seulement à prendre de meilleures décisions, mais aussi à réduire considérablement le temps du processus de conception.

3.1.2 Contributions

Le problème d'exploration de conceptions décrit ci-dessus est typiquement un problème d'optimisation multi-objectifs (MOO). Compte tenu du problème de complexité combinatoire dans ce genre de situations, nous utilisons des algorithmes évolutionnaires multi-objectifs (MOEA) [Deb, 2001]. Les MOEA sont des méta-heuristiques [6] qui permettent aux

6. Il est utile de rappeler ici que la différence entre heuristique et une méta-heuristique est que la seconde, contrairement à la première explore différemment l'espace de recherche pour augmenter les chances

© 2022 Bechir Zalila

concepteurs de trouver des solutions sous-optimales (ou presque optimales), appelées *Front de Pareto*, dans un délai raisonnable lorsque les méthodes exactes ne parviennent pas à gérer les problèmes à grande échelle en raison des besoins en ressources informatiques.

Nous proposons une méthode permettant aux concepteurs d'explorer des solutions ordonnançables et sous-optimales dans l'espace de recherche d'affectation de fonctions aux threads selon un ensemble de critères de performance conflictuels. La méthode proposée est une MOEA basée sur la stratégie d'évolution archivée de Pareto (*Pareto Archived Evolution Strategy*, PAES) [Knowles and Corne, 2000].

Nous considérons les systèmes TR²E critiques où toutes les échéances des threads doivent être respectées. La méthode d'exploration d'architecture est appliquée sur des systèmes constitués d'un ensemble de threads concurrents interagissant à travers des mémoires partagées protégées par des sémaphores. Nous nous appuyons sur un modèle de threads conventionnel basé sur le modèle classique de Liu et Layland [Liu and Layland, 1973]. Nous supposons des threads périodiques synchrones avec des délais implicites s'exécutant sur une plate-forme monoprocesseur sous une politique d'ordonnancement préemptif à priorité fixe. Nous ciblons les systèmes conformes au profil Ravenscar [Burns *et al.*, 2004], dans lequel les accès aux ressources partagées sont régis par le protocole de plafond de priorité (PCP) [Sha *et al.*, 1990] afin d'assurer la synchronisation des threads et leur exclusion mutuelle tout en évitant les interblocages [7] et les famines [8]. Le calcul des critères de performance adoptés dans ce travail ainsi que l'analyse d'ordonnancement des solutions d'affectation explorées sont effectués en utilisant CHEDDAR [Singhoff *et al.*, 2004], un framework d'analyse d'ordonnancement capable d'interpréter les modèles architecturaux AADL.

La méthode d'exploration et d'optimisation de conception multi-objectifs que nous proposons repose sur les contributions suivantes : (i) spécifier et formuler les composants PAES pour le problème traité, (ii) définir des règles qui, à partir d'une spécification fonctionnelle et d'une solution d'affectation candidate, déterminent l'architecture associée en termes de threads et de ressources partagées (y compris le calcul des paramètres temporels). L'espace de conception composé par les solutions possibles d'affectation des fonctions aux threads est exploré automatiquement à l'aide de l'algorithme PAES. Les alternatives de conception sont évaluées et comparées par rapport à un ensemble de critères de performance concurrents afin de générer celles qui répondent au mieux aux critères et aux exigences non fonctionnelles du système comme les contraintes de temps et la cohérence des données.

Plusieurs critères de performance, appelés objectifs, pourraient être impliqués pour guider l'exploration de la conception. La corrélation (c'est-à-dire les relations conflictuelles ou les relations de soutien) entre ces critères de performance n'est pas évidente et pourrait être contre-intuitive. Deux objectifs sont identifiés comme *redondants* lorsqu'ils se soutiennent : l'optimisation de l'un conduit à l'optimisation de l'autre. Considérer deux

de trouver un optimum obsolu et ne pas tomber dans le piège des optimums locaux. Les approches de méta-heuristiques sont souvent inspirés des phénomènes de la nature.

7. Un interblocage se produit dans un système lorsque tous ou une partie des threads de ce système sont impliqués dans un cycle de blocages mutuels permanent due à une mauvaise gestion du partage des ressources

8. Une famine se produit dans un système lorsque tous ou une partie des threads de ce système échouent d'une manière permanente à acquérir les ressources partagées nécessaires pour effectuer leur travaux respectifs sans pour autant être techniquement bloqués.

© 2022 Bechir ZALILA

objectifs redondants dans l'exploration de la conception n'est pas pertinent car cela augmentera inutilement la complexité du problème sans, pour autant, améliorer les solutions trouvées. Pour cette raison, nous avons effectué une étude empirique de 3 couples d'objectifs à l'issue de laquelle nous avons identifié 2 couples conflictuels pour plus de 70% des instances de tests étudiées et 1 couple redondants pour 54% des instances de tests étudiées. Toutefois, il faut noter que la corrélation entre deux objectifs dépend fortement du système et ne peut être considérée d'une manière absolue.

Pour évaluer et valider notre approche, nous avons conduit un ensemble d'expérimentations pour, d'une part, montrer que notre méthode est capable de produire des solutions très proches de celles calculées d'une manière exhaustive calculées sur des instances de petites tailles, et d'autre part, montrer que cette méthode produit des Fronts de Pareto prometteurs pour des instances de tailles plus grandes et plus complexes. Nous avons également mené des expérimentations pour illustrer l'influence du choix de la solution de départ sur la performance (en termes de convergence vers un optimum).

Le reste de ce chapitre est organisé comme suit : la section 3.2 présente la méthode proposée. Un aperçu du prototype implanté pour ce travail est donné dans la section 3.3. La section 3.4 détaille les expériences menées pour évaluer nos propositions de fonctions-objectifs et leurs résultats. La section 3.5 discute les travaux connexes et la section 3.6 conclut le chapitre.

3.2 Exploration MOO de conceptions des systèmes TR²E

De nombreux problèmes d'ingénierie impliquent plusieurs objectifs concurrents qui doivent être optimisés simultanément par rapport à un ensemble de contraintes. Par conséquent, les performances des solutions candidates doivent être évaluées selon plus d'un objectif [Coello Coello *et al.*, 2007]. Le résultat d'un algorithme MOO est une solution unique ou un ensemble de solutions, chaque solution représentant un compromis entre les objectifs.

La MOO a été appliquée dans de nombreux domaines, y compris la conception de produits et de processus, l'économie et la logistique. Lorsqu'une tentative d'amélioration d'un objectif conduit à la dégradation de l'autre, des décisions doivent être prises afin de définir des compromis entre les objectifs. Il existe rarement une solution unique offrant la meilleure valeur pour tous les objectifs. Au lieu de cela, un ensemble de solutions alternatives appelées solutions non dominées (ou Pareto-optimales) sont recherchées.

Une solution est Pareto-optimale par rapport à un ensemble d'objectifs s'il n'y a pas d'autre solution dans l'espace de recherche qui améliore tous les objectifs à la fois. Ces solutions forment l'ensemble ou le Front de Pareto. Comme le montre la figure 3.1, les vecteurs objectifs associés (points) correspondent au Front de Pareto qui représente le meilleur compromis pour les objectifs considérés.

Deux solutions candidates peuvent être comparées selon le concept de dominance de Pareto [Hruschka *et al.*, 2009] : une solution candidate c_1 domine une autre solution candidate c_2 si et seulement si (i) c_1 est strictement meilleur que c_2 pour au moins un des objectifs considérés et (ii) c_1 n'est pas pire que c_2 pour aucun des objectifs.

De nombreuses méta-heuristiques qui cherchent à approximer le Front de Pareto ont été développées pour résoudre les problèmes MOO [Bandyopadhyay and Saha, 2013]. Les points clés de ces algorithmes sont leur précision (distance des solutions qu'ils fournissent

© 2022 Bechir Zalila

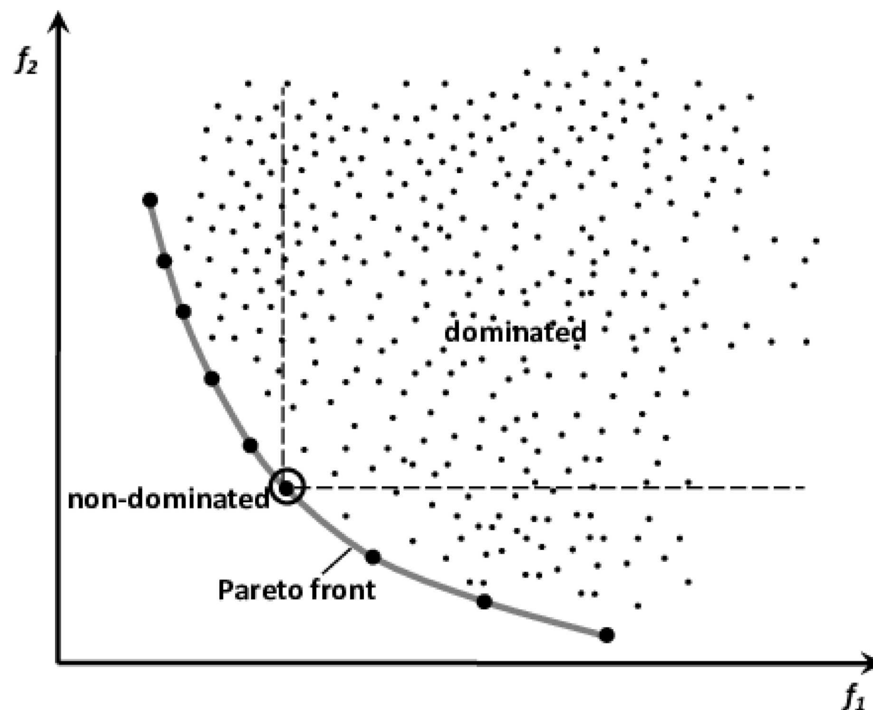

FIGURE 3.1 – Front de Pareto pour deux objectifs de minimisation [Bouaziz, 2018]

du Front de Pareto) et leur diversité (nombre et répartition de solutions dans l'espace objectif).

Une grande quantité de techniques MOO est dérivée d'algorithmes évolutionnaires multi-objectifs (MOEA). Les MOEA sont utiles dans les problèmes combinatoires qui présentent un espace très grand de recherche, lorsque les méthodes exactes pour MOO telles que la programmation linéaire multi-objectifs en nombres entiers ne sont pas applicables en raison de la quantité de ressources de calcul qu'elles nécessitent. Ils font partie des algorithmes évolutifs. Ces algorithmes d'optimisation méta-heuristiques sont inspirés de la nature, comme les colonie de fourmis et les algorithmes génétiques. Ces derniers utilisent des mécanismes inspirés de la biologie comme la mutation et le croisement afin d'affiner un ensemble de solutions candidates de manière itérative.

Dans notre approche, nous utilisons PAES (Pareto Archived Evolution Strategy [Knowles and Corne, 2000]) qui est une technique MOEA utilisant l'archivage. Elle sert à trouver un ensemble de solutions correctement réparties sur tout le spectre des compromis possibles entre les objectifs. Ceci permet d'effectuer l'exploration de conception par la suite. PAES manipule une solution unique à la fois. Ceci est un point clé dans l'exécution de fonctions d'évaluation très chronophages telles que celles issues de l'analyse d'ordonnancement. Son approche conceptuelle est assez simple en tant que procédure de recherche locale multi-objectifs. Nous avons détaillé son algorithme dans la section 2 de [Bouaziz *et al.*, 2018].

Dans la section 3.2.1, nous présentons un aperçu de notre méthodologie d'exploration de conception. Puis, nous détaillons la formulation PAES pour notre problème dans la section 3.2.2. Ensuite, dans la section 3.2.3 nous définissons un ensemble de règles permettant de calculer les paramètres d'une alternative de conception (en termes de threads, de ressources et de sections critiques) à partir d'une solution d'affectation candidate et de la

© 2022 Bechir Zalila

spécification fonctionnelle. Par la suite, l'impact de notre méthode d'affectation des fonctions aux threads sur l'ordonnançabilité du système est discuté dans la section 3.2.4.

3.2.1 Description de la solution

Comme décrit dans la figure 3.2, le point d'entrée de notre solution est la spécification fonctionnelle d'un système TR²E critique. Cette spécification définit les fonctions du système, leurs interactions et leurs caractéristiques temps réel.

A partir de cette spécification, une première architecture est proposée telle que la solution d'affectation 1–1 dans laquelle chaque fonction est affectée à un seul thread. Une analyse d'ordonnançabilité est réalisée sur la conception par rapport à la solution d'affectation 1–1 à l'aide de l'outil d'ordonnancement CHEDDAR. Si cette conception est ordonnançable, elle sera alors considérée comme la solution initiale de l'algorithme PAES. Dans le cas contraire, les paramètres de temporisation des fonctions doivent être adaptés.

Une fois une solution initiale trouvée, nous arrivons à la partie exploration et optimisation multi-objectifs. Cette dernière implique l'exécution de l'algorithme PAES.

À chaque itération, une solution de conception alternative (solution d'affectation candidate) est générée à partir de la solution actuelle en modifiant l'affectation d'une fonction à un thread d'une manière aléatoire via l'opérateur de mutation (❶ sur la figure 3.2). Des contrôles de faisabilité ❷ sont effectués sur chaque solution candidate afin de produire des conceptions qui remplissent les contraintes de temps et les contraintes d'affectation des fonctions aux threads (les détails et l'algorithme sont donnés dans [Bouaziz *et al.*, 2018]). Cette solution candidate est ensuite évaluée selon les fonctions-objectifs considérées ❸. Ensuite, d'autres étapes du PAES sont effectuées à savoir la comparaison et le classement des solutions, l'archivage et la sélection d'une solution courante pour l'itération suivante ❹.

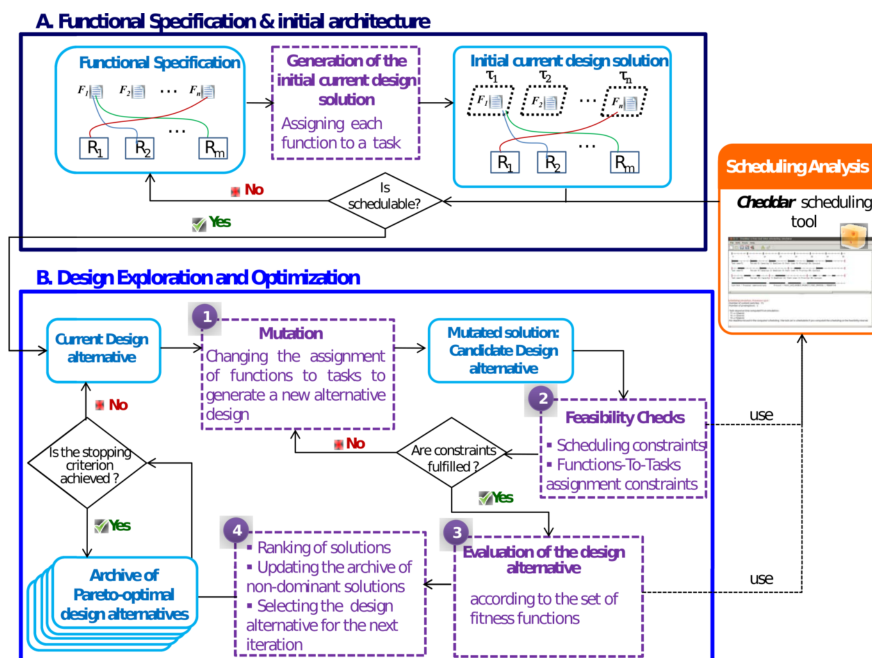

FIGURE 3.2 – Description de l'approche [Bouaziz, 2018]

© 2022 Bechir ZALILA

Les quatre étapes sont itérées jusqu'à ce que la condition de terminaison de PAES soit atteinte, par exemple, le nombre d'itérations. Le résultat de notre méthode est un ensemble d'alternatives de conception ordonnançables qui se rapprochent du Front de Pareto. À partir de ces solutions, les concepteurs choisiront la conception appropriée.

3.2.2 Formulation d'une PAES pour l'affectation

PAES repose sur la définition de plusieurs composants pour résoudre un problème de MOO particulier. Dans cette partie, nous spécifions les fonctions-objectifs. Nous donnerons aussi l'encodage des solutions en chromosomes, la solution initiale et l'opérateur de mutation afin de formuler notre problème.

Fonctions-objectifs

Compte tenu du modèle de système que nous considérons et du problème d'affectation des fonctions aux threads, plusieurs critères de performance de conception peuvent être définis comme des fonctions-objectifs pour piloter le processus d'exploration et d'optimisation multi-objectifs. Ici, nous listons quelques fonctions-objectifs possibles.

(f_1) **Minimisation des préemptions :** la minimisation du nombre de préemptions exprime la minimisation des sur-coûts de l'ordonnancement. Une préemption se produit lorsqu'un thread τ_i de priorité plus élevée est libérée pendant l'exécution d'un thread de priorité inférieure τ_j et lorsque l'ordonnanceur interrompt l'exécution de τ_j pour permettre au thread τ_i de s'exécuter [Buttazzo, 2011]. Une préemption provoque un changement de contexte qui nécessite d'exécuter plusieurs instructions du processeur et donc un sur-coût important sur le temps global d'exécution des threads. La fonction-objectif relative à cette métrique compte le nombre total de préemptions dans la simulation d'ordonnancement.

(f_2) **Minimisation des changements de contexte :** la minimisation des changements de contexte est également équivalente à la minimisation des sur-coûts de l'ordonnancement. Le contexte d'un thread se compose de son contexte mémoire et de son contexte processeur, par ex. les valeurs des registres du processeur, la valeur du compteur de programme, etc. Le basculement entre les threads consiste à sauvegarder le contexte du thread en cours d'exécution et à restaurer le contexte du thread sélectionné pour être exécuté. La fonction-objectif liée aux changements de contexte compte le nombre total de changements de contexte sur la séquence d'ordonnancement produite par simulation.

(f_3) **Minimisation du nombre de threads :** la minimisation du nombre de threads peut conduire à la minimisation à la fois du sur-coût en temps et en mémoire. En effet, un grand nombre de threads est l'un des facteurs induisant un nombre élevé de changements de contexte et nécessite des allocations mémoire supplémentaires pour la pile d'exécution des threads.

(f_4) **Maximisation de la laxité des threads :** la maximisation de la laxité des threads peut améliorer la flexibilité de la conception. En effet, plus les laxités des threads sont grandes, plus le modèle de conception peut prendre en charge des threads supplémentaires ou d'éventuels changements de paramètres des threads. La laxité (appelée aussi la marge) est le temps maximum qu'un thread peut être retardé sur son activation pour se terminer dans son délai [Buttazzo, 2011]. Pour un thread τ_i, elle est

définie comme la différence entre son échéance D_i et son temps de réponse au pire des cas $WCRT_i$. Le $WCRT_i$ d'un thread τ_i est le délai maximum entre son temps de libération et son temps d'achèvement [Audsley *et al.*, 1993]. PAES requiert que les fonctions-objectifs doivent êtres toutes des fonction de minimisation ou bien toutes de maximisation. Ainsi, cette fonction est transformée en une fonction de minimisation en prenant l'opposé de la somme des laxités et en lui ajoutant une constante pour qu'elle soit toujours positive.

(f_5) **Minimisation du WCRT [9] des threads :** dans le cadre des systèmes temps réel, nous avons toujours intérêt à minimiser le temps de réponse des threads.

(f_6) **Minimisation du WCBT [10] des threads :** le temps de blocage est induit par l'exclusion mutuelle des ressources. Cela provoque des latences dans l'exécution des threads.

(f_7) **Minimisation du nombre de ressources partagées :** la minimisation du nombre de ressources partagées permet de minimiser le nombre de sémaphores et donc de réduire le sur-coût en mémoire.

Toutes les fonctions citées ci-dessous sont détaillées dans [Bouaziz, 2018]. Bien entendu, ces fonctions ne peuvent pas toutes êtres utilisées car ceci augmente la dimension du problème et par conséquent le temps pour trouver une solution. Bien que certains des objectifs énumérés ci-dessus soient contradictoires, d'autres peuvent se comporter de manière non-conflictuelle. Dans le second cas, les objectifs se soutiennent mutuellement et sont désignés comme objectifs redondants. Un objectif est identifié comme redondant lorsque le Front de Pareto reste le même même si cet objectif est omis de l'ensemble d'objectifs d'origine [Gal and Hanne, 1999].

Nous notons que toutes les combinaisons d'objectifs parmi la liste donnée ci-dessus ne sont pas pertinentes puisque certaines paires d'objectifs sont évidemment non-conflictuelles, par exemple (f_1, le nombre de préemptions et f_2, le nombre de changements de contexte) ou encore (f_4, la laxité des threads et f_5, le WCRT des threads). Les objectifs de chacune de ces paires ne doivent pas être considérés ensemble pour guider le processus d'exploration de la conception. Néanmoins, il n'est pas intuitivement évident pour nous de prédire la relation entre toutes les paires redondantes de la liste de fonctions-objectifs ci-dessus. Pour cela, nous avons choisi trois objectifs (f_1, f_4 et f_6) et nous avons utilisé les expérimentations de la sections 3.4 pour enquêter sur les relations (conflit ou soutien) entre chaque couple possible de ces trois fonctions-objectifs.

Encodage en chromosomes

Nous représentons une solution au moyen d'un schéma de codage conventionnel en entiers. Avec un tel codage, un chromosome est défini comme un vecteur d'entiers de taille n. Dans notre cas, n représente le nombre de fonctions dans la spécification fonctionnelle. Un chromosome présente des informations sur l'affectation des fonctions aux threads. Chaque position du chromosome, appelée gène, désigne une fonction particulière. Autrement dit, la i[e] position correspond à la i[e] fonction. La valeur détenue par un gène représente l'indice du thread à laquelle la fonction correspondante est affectée. Par conséquent, un chromosome représente une solution formée de k threads ($k \leq n$), où

9. **Worst Case Response Time :** pire temps de réponse. Très souvent assimilé au WCET (Worst Case Execution Time, le pire temps d'exécution)
10. **Worst Case Blocking Time :** pire temps de blocage

© 2022 Bechir Zalila

chaque gène a une valeur dans $\{1, 2, \ldots, k\}$. Cela signifie que chaque fonction est affectée à l'un des threads $\{\tau_1, \tau_2, \ldots, \tau_k\}$.

La figure 3.3 montre un exemple de chromosome qui correspond à une solution d'affectation particulière S. La longueur du chromosome indique le nombre de fonctions. Dans cet exemple, la spécification fonctionnelle se compose de 8 fonctions. Le gène F_1 a une valeur égale à 2, ce qui signifie que la fonction F_1 est affectée au thread d'index 2 (τ_2). La solution modélisée par ce chromosome affecte F_4 à τ_1 ; F_1, F_3 et F_7 à τ_2 ; F_2 et F_6 à τ_3 ; F_5 et F_8 à τ_4.

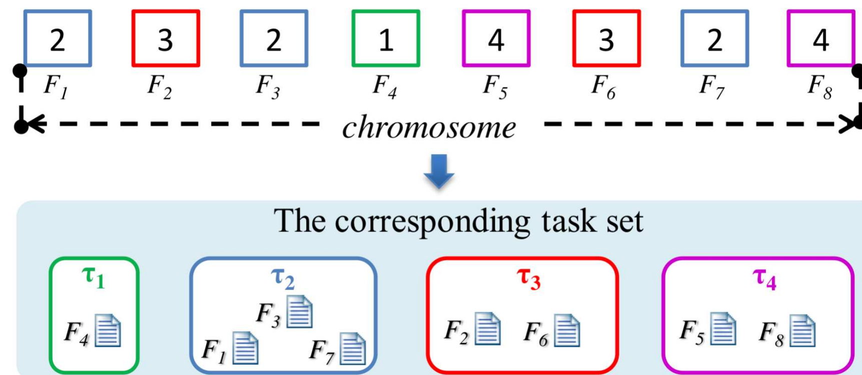

FIGURE 3.3 – Représentation en chromosome d'une solution d'affectation [Bouaziz, 2018]

Ce codage est assez simple mais il est redondant puisqu'une même solution peut être représentée par des chromosomes différents. Par exemple, la solution S codée par le chromosome représenté sur la figure 3.3 peut également être représentée par d'autres chromosomes, à savoir $[1, 2, 1, 3, 4, 2, 1, 4]$, $[3, 1, 3, 2, 4, 1, 3, 4]$, $[2, 4, 2, 3, 1, 4, 2, 1]$, etc. En d'autres termes, une solution peut être représentée par de nombreux chromosomes qui modélisent la même solution d'affectation indépendamment des indices de threads.

Pour réduire toutes les solutions d'affectation équivalentes à la même représentation chromosomique, nous normalisons la représentation chromosomique. Ceci s'effectue de la manière suivante : l'indice de thread de la fonction F_1 est toujours 1 (alors toutes les fonctions qui appartiennent au même thread que F_1 ont pour indice de thread 1). L'index de thread de F_2 est 2, à moins que F_2 soit affectée au même thread que F_1, dans ce cas on prend l'indice suivant, et ainsi de suite. La figure 3.4 montre la représentation chromosomique normalisée de la même solution d'affectation S de la figure 3.3.

La procédure de normalisation permet de comparer deux solutions. Elle doit être appliquée sur chaque chromosome d'une solution candidate générée par mutation. Elle permet aussi de réduire le nombre de solutions à explorer.

Solution initiale

PAES a besoin d'une première solution pour lancer la procédure de recherche. Nous avons mis en œuvre deux manières de générer cette solution initiale : la première manière est très simple et elle favorise l'objectif de laxité (f_4) tandis que la seconde est plus sophistiquée et a nécessité un peu plus d'effort (nécessite un pré-traitement des fonctions du système) et elle favorise l'objectif de temps de blocage (f_6) :

© 2022 Bechir ZALILA

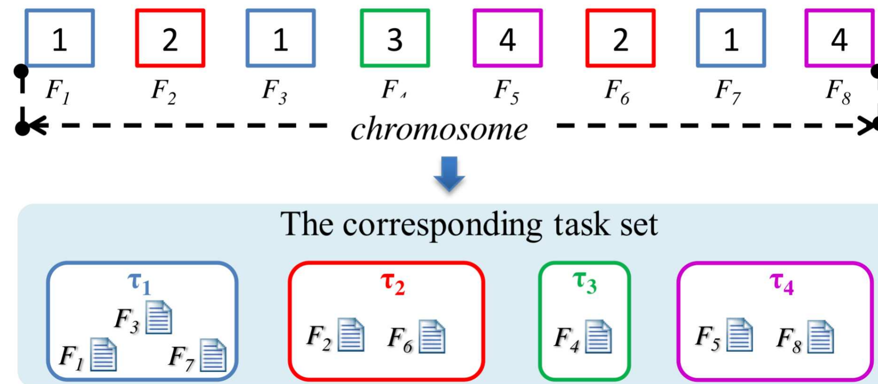

FIGURE 3.4 – Représentation normalisée en chromosome d'une solution d'affectation [Bouaziz, 2018]

(1) Une fonction par thread (appelée solution d'affectation 1–1) : Elle consiste à affecter chaque fonction à un seul thread, Le nombre de threads est égal au nombre de fonctions et le chromosome représentant cette solution sera $[1, 2, 3, \ldots, n]$, n étant le nombre de fonctions.

Les laxités des threads augmentent avec la granularité des fonctions à affecter aux threads. Cela signifie que la solution d'affectation 1–1 fournit la meilleure laxité globale du système, favorisant ainsi l'objectif de laxité (f_4). Ceci est garanti, très probablement au détriment d'autres fonctions-objectifs et cette solution n'est, le plus probablement, pas la meilleure.

(2) Solution initiale pré-traitée pour minimiser les temps de blocage (appelée solution pré-traitée) : lorsqu'on considère la solution d'affectation 1–1 comme solution initiale, cette dernière n'est biaisée qu'envers l'objectif de laxité. Dans le cas de systèmes à ressources partagées, cette première solution peut être très pessimiste quant au critère de temps de blocage (f_6). Un pré-traitement de la solution initiale peut être effectué avant d'exécuter le processus d'exploration PAES afin de promouvoir la fonction-objectif f_6. Une des contributions de [Bouaziz, 2018] a été le développement d'une méta-heuristique qui essaie de regrouper autant que possible les fonctions dépendantes au sein des mêmes threads minimisant ainsi les temps de blocage. Cette méta-heuristique prend en compte plusieurs paramètres et garantit que la solution qu'elle délivre est une solution ordonnançable.

Opérateur de mutation

Au lieu d'appliquer des opérateurs de mutation génériques qui peuvent ne pas être entièrement adaptés à notre problème (produire des solution non ordonnançables), nous avons mis en œuvre dans [Bouaziz *et al.*, 2018] un opérateur de mutation basé sur la connaissance. Cet opérateur de mutation est conçu sur la base des caractéristiques du problème d'affectation des fonctions aux threads, abordé dans cette contribution. Il choisit une position aléatoire dans le chromosome et change la valeur du gène associé en une nouvelle valeur. La mutation produit une nouvelle solution d'affectation alternative à partir de la solution actuelle en réaffectant une fonction aléatoire F_i à un thread choisi au hasard dans l'ensemble des threads harmoniques avec F_i (dont les périodes ont un

© 2022 Bechir Zalila

diviseur commun non trivial). Cette restriction de choisir seulement des threads harmoniques est faite afin de suivre les règles d'affectation des fonctions aux threads définies dans la section 3.2.3. De plus, l'opérateur de mutation est implanté de manière à générer uniquement des solutions ordonnançables qui satisfont les contraintes du système.

3.2.3 Règles d'affectation des fonctions aux threads

Afin de garantir des solutions candidates qui soient ordonnançables et correctes, une des contributions les plus importantes de [Bouaziz, 2018] a été d'établir un ensemble de règles qui régissent l'affectation des fonctions du système aux threads. Ces règles permettent de déterminer suite à une affectation les paramètres des threads du système afin de pouvoir déterminer si elles sont ordonnançables.

La règle principale qu'il faut respecter lors de l'affectation de fonctions aux threads, est de n'affecter à un thread que des fonctions harmoniques (dont les périodes sont multiples d'une période commune non triviale). A partir de cette règle, nous pouvons déterminer les caractéristiques d'un thread τ_i.

La période d'un thread τ_i est le PGCD des périodes des fonctions qui lui sont affectées. Ceci permet, de simplifier l'implantation de ce thread sous la forme d'une boucle. Le pire temps d'exécution du thread ($WCET_i$) est égal à la somme des $WCET$ des fonctions qui lui sont affectées. Enfin, l'échéance du thread est égale au min des échéances des fonctions affectées.

Pour minimiser les temps de blocage et le nombres de sections critiques, nous avons mis en œuvre dans [Bouaziz, 2018] un algorithme permettant de prendre en compte les ressources partagées entre les threads lors de la mutation et de fusionner les sections critiques consécutives chaque fois que cela est possible ou encore supprimer des sections critiques si les ressources correspondantes ne sont plus partagées. Ceci est illustré par un exemple dans les figures 3.5, 3.6 et 3.7.

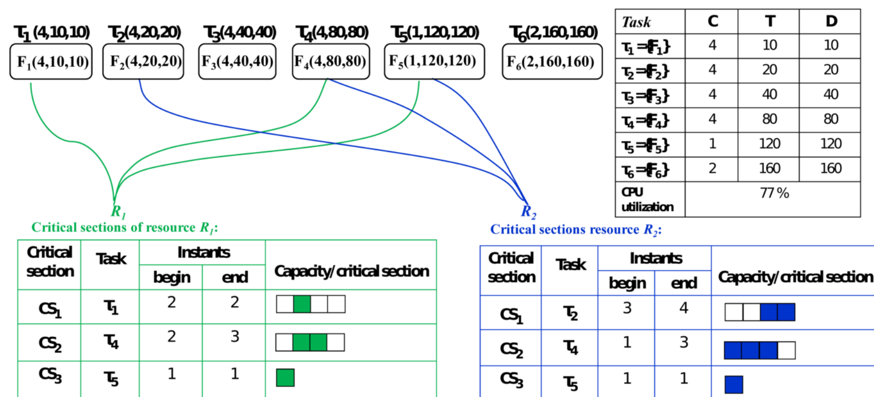

FIGURE 3.5 – Affectation initiale 1–1 d'un exemple [Bouaziz, 2018]

La conception initiale de cet exemple est donnée sur la figure 3.5. Elle se compose de 6 threads, chacune contenant une fonction selon la méthode 1–1. Comme le montre la figure 3.5, il existe deux ressources partagées R_1 et R_2 partagées chacune entre 3 des 6 fonctions.

En appliquant notre algorithme de mutation, une affectation candidate possible est celle montrée dans la figure 3.6. Dans cette nouvelle affectation, nous avons affecté les

© 2022 Bechir ZALILA

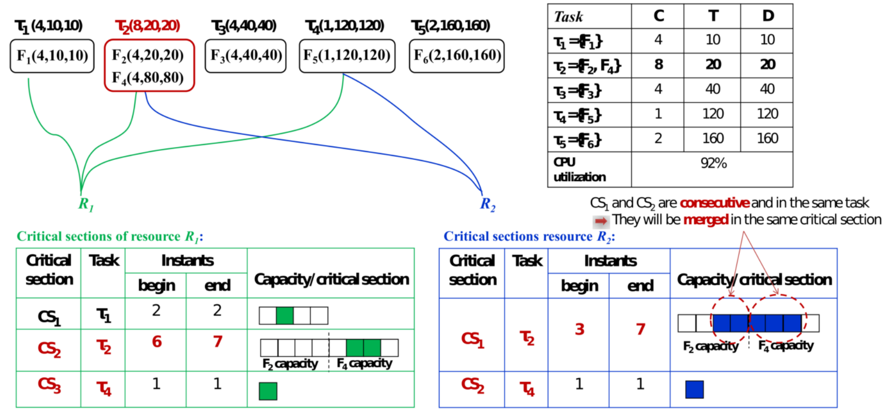

FIGURE 3.6 – Affectation après mutation et fusion de sections critiques [Bouaziz, 2018]

fonctions F_2 et F_4 (qui sont harmoniques) à un même thread et fusionné leur sections critiques relatives à la ressource R_2 puisque ces deux sections critiques se sont trouvée consécutives.

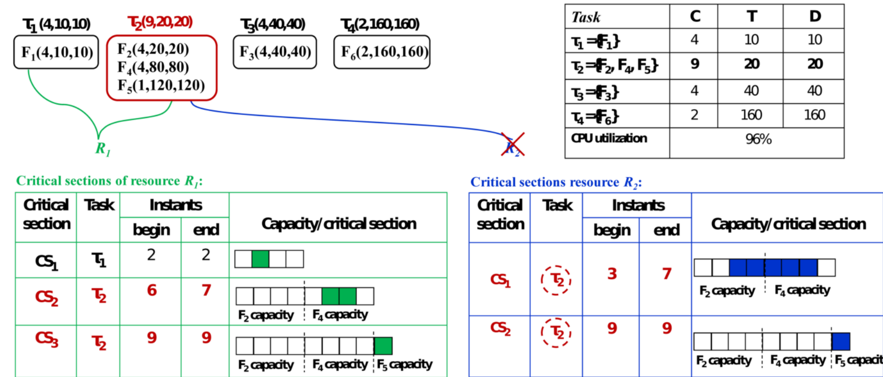

FIGURE 3.7 – Affectation après mutation et suppression d'une section critique [Bouaziz, 2018]

A partir de la solution donnée à la figure 3.6, une mutation possible consiste à affecter la fonction F_5 avec les fonctions F_2 et F_4 au thread τ_2 car F_5 est harmonique avec τ_2. Encore une fois, nous appliquons notre algorithme de mutation sur cette solution mutée et nous obtenons l'ensemble de ressources et les sections critiques comme indiqué sur la figure 3.7. Nous pouvons observer que la ressource R_2 n'est plus partagée. Ses sections critiques sont donc supprimées. Ceci causera à la fois la réduction des temps de blocage et de l'empreinte mémoire.

3.2.4 Tests de faisabilité sur les solutions candidates

Les contrôles de faisabilité sont utilisés pour éviter la génération de solutions candidates incorrectes après mutation. Nous distinguons deux types de contrôles de faisabilité : les vérifications des exigences temporelles et d'ordonnancement et les vérifications liées à l'affectation des fonctions aux threads. Dans le reste de cette partie, nous discutons

© 2022 Bechir ZALILA

d'abord de l'impact de la mutation et de la méthode d'affectation sur l'ordonnançabilité d'une solution de conception. Ensuite, nous montrons comment cela peut conduire à des activations inutiles de certains threads. Enfin, nous décrivons la manière de vérifier toutes les contraintes, permettant ainsi de nous prononcer sur la faisabilité d'une solution candidate donnée générée par mutation.

Impact sur l'ordonnançabilité

La manière d'affecter les fonctions aux threads influence le taux d'utilisation global du processeur et peux conduire, si ce taux dépasse 100%, on peut conclure avec certitude que la solution candidate est non ordonnançable. Il en est de même si un thread (ou plusieurs) ont la capacité qui dépasse l'échéance ($C > D$). Dans notre cadre, nous utilisons l'algorithme d'ordonnancement RMS [Liu and Layland, 1973], pour être sûr qu'un ensemble de threads est ordonnançable, il faut que le taux d'utilisation global du processeur soit $\leq 69\%$.

Pour les solutions candidates où le taux d'utilisations global du processeur est entre 69% et 100% et pour lesquelles on ne peux pas conclure à coup sûr de l'ordonnançabilité, une analyse d'ordonnançabilité doit être menée sur chaque alternative de conception explorée par mutation au cours du processus de recherche.

Contraintes d'affectation des fonctions aux threads

Le fait que la période d'un thread soit égale au PGCD des périodes des fonctions qui lui sont associées peut conduire à une situation dans laquelle ce PGCD soit différent de toutes les périodes des fonctions. Par exemples, deux fonctions F_1 et F_2 de périodes respectives 40 et 60, quand elles sont affectées à un thread τ_i, la période de ce thread sera égale à 20. Ceci conduit, lors de l'exécution du thread τ_i à des activations (et donc, des changements de contexte) inutiles. Pour cette raison, les solutions candidates contenant des threads présentant cette anomalie, seront rejetées.

Algorithme de test de faisabilité

Afin de s'assurer que la mutation ne génère que des solutions réalisables, un ensemble de vérifications de faisabilité est effectué sur chaque alternative de conception explorée. Ces vérifications de faisabilité ont été structurées sous la forme d'un algorithme (détaillé dans [Bouaziz, 2018]). Cet algorithme a par la suite été implanté dans le framework d'ordonnancement temps réel CHEDDAR.

3.3 Prototype implanté dans CHEDDAR

Dans cette section, nous présentons le prototype développé dans le cadre de la contribution de ce chapitre. Ce prototype a été implanté dans le framework CHEDDAR [Singhoff *et al.*, 2004]. CHEDDAR est un outil libre d'analyse d'ordonnancement temps réel. Il est conçu pour vérifier les contraintes temporelles des systèmes temps réel critiques. Il est écrit en utilisant le langage de programmation Ada.

Notre prototype couvre les aspects suivants :

© 2022 Bechir ZALILA

— La mise en œuvre de l'exploration architecturale basée sur PAES via l'affectation des fonctions aux threads. Elle inclut les fonctionnalités proposées dans nos précédents travaux [Bouaziz *et al.*, 2016] pour améliorer l'évolutivité et l'efficacité de notre méthode d'exploration des conceptions. Ces fonctionnalités sont une implantation parallèle de notre méthode et une nouvelle stratégie de sélection pour l'algorithme PAES.

— Le développement d'un générateur d'ensembles de fonctions avec ressources partagées pour réaliser les expérimentations qui nous permettrons de savoir si un couple de fonctions-objectifs est redondant ou pas.

Avec CHEDDAR, les concepteurs doivent spécifier l'architecture de l'application à analyser avec le langage AADL [SAE, 2017a], ou l'ADL spécifique de CHEDDAR appelé CHEDDAR-ADL. CHEDDAR-ADL est un ADL dédié à l'analyse d'ordonnancement temps réel. Il permet aux utilisateurs de définir, pour un système temps réel donné, les composants logiciels (e.g. threads, ressources partagées, messages, etc.), les composants matériels (e.g. processeurs, cœurs, caches, etc.) et les interactions entre eux. Les concepteurs peuvent utiliser l'éditeur graphique associé à CHEDDAR-ADL afin de construire facilement leurs modèles d'applications temps réel.

Deux méthodes différentes d'analyse d'ordonnancement sont supportées par CHEDDAR : les tests de faisabilité et les simulations d'ordonnancement. CHEDDAR prend en charge la plupart des algorithmes d'ordonnancement des systèmes TR^2E classiques et des tests de faisabilité pour différents types d'applications temps réel (monoprocesseur/multiprocesseur, threads/threads indépendants avec contraintes de précédance/threads partageant des ressources, etc.).

Dans le cadre de notre contribution, deux nouvelles bibliothèques ont été ajoutées à CHEDDAR :

— La bibliothèque **`Optimizers`** qui se compose de deux paquetages Ada qui définissent les sous-programmes de base et communs qui pourraient être réutilisés pour formuler tout problème avec l'algorithme PAES ou l'algorithme de méthode exhaustive respectivement. Le premier nécessite un opérateur de mutation, le second un moyen d'énumérer les solutions. Les deux sont également paramétrés avec des fonctions-objectifs.

— La bibliothèque **`Function_To_Task_Assignment`** fournit des composants qui sont spécifiques à notre domaine de problèmes d'optimisation. Elle contient deux paquetages :

(1) Le premier paquetage (manipulation des données chromosomiques) fournit des fonctions pour manipuler la représentation chromosomique personnalisée en fonction de l'affectation des fonctions aux threads. Par exemple, il permet aux utilisateurs de transformer une représentation chromosomique d'une solution candidate donnée en un modèle CHEDDAR-ADL en utilisant la bibliothèque CHEDDAR-ADL existante dans CHEDDAR. Il comprend les règles d'affectation des fonctions aux threads qui déterminent et calculent les paramètres des threads, des ressources et des sections critiques d'une solution candidate donnée. L'opérateur de mutation proposé pour notre problème est également implanté dans ce paquetage. Ce paquetage définit également une routine qui, à partir d'une solution initiale (avec représentation chromosomique), énumère toutes les affectations possibles des fonctions aux threads (encore une fois avec des repré-

© 2022 Bechir ZALILA

sentations chromosomiques). Cette méthode est utilisée pour mettre en œuvre la méthode exhaustive.

(2) Le deuxième paquetage (fonctions-objectifs et contrôles de faisabilité) est dédié à l'évaluation de solutions au moyen de fonctions-objectifs, et à la réalisation des contrôles de faisabilité sur les alternatives de conception. Ce paquetage utilise la bibliothèque de simulation d'ordonnancement afin d'effectuer l'analyse d'ordonnancement et de calculer les fonctions-objectifs à partir de la simulation résultante (par exemple, le nombre de préemptions, le nombre de changements de contexte, WCRT et WCBT des threads, etc.).

Afin de réaliser les différentes expériences avec un large éventail de configurations, nous avons choisi de générer des modèles d'entrée synthétiques, c'est-à-dire des ensembles de n fonctions interagissant via des ressources partagées. Ceci est fait à l'aide d'un générateur d'ensemble de fonctions que nous avons implanté afin de réaliser les expérimentations objet de la section 3.4.

3.4 Études Expérimentales

Dans cette section, nous présentons quatre expériences réalisées dans le but d'étudier certains aspects liés à notre problème ainsi que d'évaluer nos propositions. Tout d'abord, nous explorons la corrélation entre trois fonctions-objectifs sélectionnées dans la liste mentionnée dans la section 3.2.2. Ensuite, nous évaluons la précision (en termes de convergence et de capacité de couverture) de notre méthode d'exploration de conception basée sur PAES. Cette évaluation est réalisée sur des instances de problèmes de petite taille par comparaison avec une méthode de recherche exhaustive pour que cette dernière soit réalisable. Par la suite, nous évaluons l'efficacité de notre méthode pour des systèmes plus importants en étudiant la qualité des Fronts de Pareto produits pour différents niveaux de conflit de ressources. Enfin, nous analysons l'impact du choix de la solution initiale sur la convergence de la méthode d'exploration de conception basée sur PAES vers le Front de Pareto optimal.

Quatre expériences ont été réalisées :

(1) La première expérience est étude empirique de la corrélation entre les fonctions-objectifs. Comme précisé dans la section 3.2.2, nous avons retenu les 3 fonctions-objectifs f_1, f_4 et f_6. Le but de cette expérimentation est de vérifier, lors du passage de modèles à fonctions indépendantes à des modèles à ressources partagées, s'il faut considérer 2 objectifs ou rester à 3 objectifs. Ceci est réalisé en étudiant la relation de conflit entre chaque paire de ces 3 fonctions. L'issue de cette première expérimentation nous a indiqué qu'il faut, dans tous les cas, utiliser 3 fonctions-objectifs et ne pas se limiter à deux ou encore à une fonction.

(2) La seconde expérience a permis l'évaluation de la précision de la méthode d'exploration d'architecture basée sur PAES pour les instances de problèmes de petite taille. Elle vise à évaluer la convergence et la couverture de notre méthode d'exploration d'architecture basée sur PAES. Ceci nécessite la connaissance du vrai Front de Pareto. Nous proposons une méthode qui détermine le vrai Front de Pareto grâce à une recherche exhaustive parmi toutes les solutions d'affectation des fonctions aux threads (et c'est pour cette raison que la taille du problème doit être petite). La méthode exhaustive fonctionne comme suit :

© 2022 Bechir Zalila

(a) Énumérer toutes les solutions d'affectation possibles pour un ensemble de fonctions donné. Le nombre de solutions est égal au nombre de Bell [Rota, 1964] de la taille de l'ensemble de fonctions traité,
(b) Déterminer des solutions réalisables parmi toutes les solutions énumérées,
(c) Évaluer les solutions selon les fonctions-objectifs considérées,
(d) Enfin, calculer le Front de Pareto à partir de solutions réalisables selon le concept de dominance.

L'issue de cette expérimentation montre que PAES arrive à converger vers le vrai Front de Pareto dans 68% des instances générées.

(3) Dans la troisième expérience, nous nous sommes intéressés à évaluer l'efficacité de notre méthode en étudiant la qualité des ensembles générés de solutions pour différents niveaux de partage des ressources. Les ensembles de solutions obtenus dans les expériences en cours sont à évaluer quantitativement (au moyen du nombre de solutions non dominées trouvées) et qualitativement.

Cette expérience nous a montré que notre méthode permet d'explorer efficacement l'architecture des systèmes à ressources partagées en générant un ensemble prometteur de compromis de conception.

(4) Dans la quatrième expérience, nous avons étudié l'impact de la solution initiale sur les performances de la méthode d'exploration de conception basée sur PAES. Dans tous les ensembles d'expériences précédents, nous avons exécuté une méthode d'exploration de conception basée sur PAES en définissant la solution initiale sur la solution d'affectation 1–1. Cette dernière représente une solution de conception extrême impliquant le nombre maximum de threads qui favorise l'objectif de la laxité (f_4). La solution initiale d'affectation 1–1 est assez simple et définie à des coûts de calcul très faibles. Cependant, démarrer la recherche à partir d'une solution extrême peut empêcher l'algorithme d'atteindre toutes les solutions dans le vrai Front de Pareto. Pour cette raison, nous avons étudié dans la section 3.2.2 une autre solution initiale pré-traitée permettant de minimiser les temps de blocage.

Nous avons conduit un ensemble d'expérimentations visant à comparer les fronts générés en exécutant la méthode d'exploration de conception basée sur PAES tout en définissant la solution initiale d'abord sur une solution d'affectation 1–1 et ensuite sur une solution pré-traitée.

Les résultats de cette expérience nous ont permis de conclure qu'un bon choix de la solution initiale améliorerait les performances (en termes de précision et de convergence vers des résultats optimaux) de notre méthode d'exploration de conception. De plus, ces résultats prouvent que la méthode que nous proposons pour pré-traiter une solution initiale adéquate pour les systèmes à ressources partagées, est efficace pour les systèmes de petite taille. Cependant, l'efficacité de cette méthode diminue à mesure que les conflits de ressources et le nombre de fonctions dans le système augmentent.

3.5 État de l'art

Dans cette section, nous discutons des travaux connexes visant à piloter la conception d'architectures logicielles et l'ordonnancement des threads pour les systèmes temps réel

© 2022 Bechir Zalila

critiques. Tout d'abord, nous présentons des approches qui automatisent l'affectation de fonctions logicielles à une plate-forme spécifique de manière à respecter les contraintes non fonctionnelles. Certaines de ces approches s'intéressent à optimiser certains critères de performance du système conçu. Ensuite, nous passons en revue les travaux qui traitent de l'exploration de conceptions multi-critères à l'aide de techniques d'optimisation multi-objectifs dédiées. Enfin, nous introduisons quelques approches de conception d'architectures logicielles basées sur des techniques de regroupement.

Dans la littérature, de nombreuses approches ont été proposées pour piloter la conception de modèles architecturaux qui doivent répondre aux exigences temporelles, en transformant une spécification fonctionnelle en une architecture multi-tâches. Dans [Bartolini *et al.*, 2005], les auteurs ont développé des heuristiques qui génèrent le modèle architectural à partir d'un modèle fonctionnel de flux de données avec des propriétés de synchronisation. L'objectif principal de ce travail est d'automatiser le problème de projections (*mapping*) (i.e. du niveau fonctionnel au niveau architectural) tout en trouvant un compromis entre les deux solutions de mapping extrêmes (à savoir, la solution d'affectation 1–1 et la solution à un thread unique). Cependant, l'approche proposée étudie une alternative de conception unique, ce qui signifie que les aspects d'exploration et d'optimisation de la conception ne sont pas pris en compte.

Dans [Pagetti *et al.*, 2011], les auteurs ont fourni un framework pour l'intégration et le développement de systèmes temps réel embarqués. Avec ce framework, les concepteurs écrivent une spécification fonctionnelle avec des contraintes de dépendances en utilisant le langage *Prelude*. A partir de ce modèle, le framework proposé permet de générer un ensemble de threads pouvant être exécutés sur une architecture monoprocesseur. Les auteurs ont prouvé que l'implantation générée garantit le comportement correct du système ainsi que les contraintes temporelles décrites dans la spécification fonctionnelle. Néanmoins, dans ce travail, les fonctions sont attribuées aux threads selon une stratégie d'attribution 1–1. Cette stratégie d'affectation induit un grand nombre de threads qui, à leur tour (i) peuvent entraîner une surcharge excessive de l'ordonnanceur suite au changement de contexte (ii) et peuvent dépasser le nombre maximal de threads prises en charge par le RTOS cible. Dans notre travail, nous explorons des alternatives de conception grâce à une technique de regroupement qui permet d'affecter plus d'une fonction au même thread selon un ensemble de règles, ce qui permet de limiter le nombre de threads. Grâce à l'aspect multi-objectifs apporté par notre approche, le nombre de threads dans les conceptions produites (i.e. l'ensemble des conceptions non dominées) est équilibré à travers les deux objectifs contradictoires, à savoir (a) la minimisation des préemptions qui est renforcée lorsque plus les fonctions sont affectées à un thread et (b) la maximisation des laxités des threads qui pourraient être améliorées avec une affectation précise. Encore une fois, le travail cité n'étudie pas l'exploration de l'espace de conceptions.

Dans le même contexte, une méthodologie basée sur MARTE [Mraidha *et al.*, 2011] a été proposée. Elle permet une analyse d'ordonnancement aux premiers stades du cycle de vie du logiciel. Elle permet aux concepteurs de générer, à partir d'une spécification fonctionnelle exprimée avec le profil MARTE d'UML, un modèle de conception conforme aux exigences temporelles de la spécification fonctionnelle. L'exploration de l'espace de conceptions vers l'optimisation des critères de performance est également hors de portée du travail cité.

Les systèmes distribués dans le domaine de l'automobile utilisent souvent AUTOSAR. Dans ce framework [Fürst *et al.*, 2009], l'affectation des fonctions aux threads est connue par la projection d'entités exécutables aux threads et est identifiée comme une étape pri-

© 2022 Bechir ZALILA

mordiale dans la méthodologie de conception standard définie par AUTOSAR [Scheickl and Rudorfer, 2008]. Compte tenu de la complexité accrue des systèmes utilisés dans l'industrie automobile, plusieurs approches ont émergé visant à gérer cette complexité en automatisant l'intégration d'entités fonctionnelles définies par les exécutables sur une architecture matérielle composée d'un réseau de calculateurs. Parmi ces approches, on peut citer [Monot *et al.*, 2012] qui a proposé deux heuristiques pour projeter des entités exécutables à une architecture multi-cœurs. La première heuristique traite du mapping des exécutables vers les cœurs en tenant compte des dépendances inter-exécutables et des contraintes de localité tout en optimisant la charge d'un cœur. Pour les systèmes où la plupart des exécutables sont dépendants, la stratégie utilisée pour attribuer les exécutables aux cœurs peut entraîner l'attribution de la plupart d'entre eux à un seul cœur. La seconde heuristique permet de construire le séquençage des entités exécutables de chaque cœur en utilisant un thread par cœur. Encore une fois, une conception réalisable unique est évaluée. En conséquence, à la fois l'exploration de plusieurs alternatives réalisables et l'optimisation d'un ensemble de critères de performance conflictuels ne sont pas pris en compte.

Dans [Wozniak *et al.*, 2013; Mehiaoui *et al.*, 2012], les auteurs ont présenté une approche en deux étapes visant à optimiser la projection des entités exécutables aux threads par rapport à un ensemble de métriques d'optimisation telles que le temps de réponse de bout-en-bout, la consommation de mémoire, le débit du bus, etc. D'abord, les entités exécutables de données sont partitionnés sur des ECUs. Ensuite, les *runnables* de chaque ECU sont affectés à des threads. Le problème de projection est résumé et résolu à l'aide de deux stratégies et techniques d'optimisation différentes, à savoir (a) une stratégie exacte utilisant la programmation linéaire en nombres entiers mixtes (MILP) et (b) une approche méta-heuristique via un algorithme génétique (GA) pour faire face à l'évolutivité, sujet de la méthode exacte basée sur MILP. Contrairement à ces approches mono-critère, nous proposons une approche d'exploration de conception multi-critères basée sur une technique méta-heuristique dédiée au MOO.

Certaines contributions à la recherche ont été développées dans le cadre de l'exploration de conception en utilisant des techniques d'optimisation multi-objectifs. La plupart d'entre elles sont basées sur des MOEA. Dans [Rahmoun *et al.*, 2015b; Rahmoun *et al.*, 2015a], les auteurs ont proposé une méthode qui explore des alternatives d'architecture pour les systèmes embarqués temps réel, telles que les architectures produites répondent au mieux à un ensemble de propriétés non fonctionnelles conflictuelles. Cette méthode est basée sur la composition des transformations de modèles et la MOEA au moyen de la stratégie d'optimisation multi-objectifs NSGA-II [Deb *et al.*, 2002]. Par ailleurs, dans [Koziolek *et al.*, 2011], les auteurs ont développé un framework appelé Peropteryx. Ce framework aide les architectes logiciels au cours de la phase de conception à se rapprocher des architectures du Front de Pareto. Les processus d'exploration et de sélection sont guidés par les stratégies architecturales et la NSGA-II. En outre, AQOSA [Li *et al.*, 2011] est un autre framework générique qui fournit un processus automatisé d'exploration de conceptions basé sur un ensemble de MOEA, à savoir NSGA-II, SPEA2 et SMS-EMOA. Ces approches s'appuient sur des opérateurs génétiques génériques (mutation/croisement), qui pourraient nuire aux performances de la procédure de recherche et par conséquent entraver la méthode d'exploration de conception pour converger vers le Front de Pareto optimal. Contrairement à cette approche, nous définissons un opérateur de mutation basé sur la connaissance conformément aux caractéristiques du problème d'affectation des fonctions aux threads. De plus, tous ces frameworks ont adopté des MOEA basés sur la population, ce qui ne convient pas aux problèmes impliquant une évaluation de fonctions-objectifs coûteuse

© 2022 Bechir ZALILA

en calcul comme le problème abordé dans notre travail. Pour cela, nous avons choisi PAES comme MOEA car il gère une seule solution à chaque itération.

3.6 Conclusion et perspectives

Dans ce chapitre, nous avons proposé une méthode d'exploration de conception basée sur l'optimisation multi-objectifs pour des systèmes temps réel critiques. Cette méthode permet d'explorer l'espace de conception en étudiant de nombreuses alternatives d'affectation des fonctions aux threads afin de sélectionner celles qui répondent au mieux à un ensemble de critères de performance concurrents. Les conceptions finales sélectionnées appliquent le comportement du système et les exigences de synchronisation décrites dans la spécification fonctionnelle.

Une technique de regroupement est utilisée afin d'explorer l'espace de recherche de conception. Pour aborder la question combinatoire du problème traité, notre méthode s'appuie sur un algorithme évolutionnaire d'optimisation multi-objectifs, à savoir la stratégie d'évolution archivée de Pareto.

Notre méthode a été implantée et intégrée dans Cheddar, un framework d'ordonnancement temps réel existant. Le principal avantage de cette implantation est sa réutilisabilité et l'extensibilité de ses artefacts logiciels réalisés sous forme de bibliothèques. Par exemple, la bibliothèque d'optimiseurs pourrait être réutilisée dans différents problèmes ou étendue avec d'autres méthodes d'optimisation. De même, le moteur dédié à la spécification de l'affectation des fonctions aux threads est réutilisable dans l'instanciation de notre problème avec d'autres MOEA, etc.

Nous avons également présenté les résultats d'études expérimentales réalisées pour étudier certaines caractéristiques découlant de notre problème et évaluer l'efficacité de notre approche. Les expériences sont faites sur des instances de problèmes générées synthétiquement. Nous avons réalisé une étude empirique visant à étudier la corrélation entre des paires d'objectifs (c'est-à-dire des critères de performance). Nos expériences montrent que la corrélation dépend de l'instance du problème abordé. En outre, nous avons réalisé un ensemble d'expériences visant à explorer l'effet du choix de la solution initiale sur les performances de la méthode PAES. Les expériences pour cette évaluation ont été réalisées sur des instances de problèmes de petite taille. Cela nous a conduit à conclure que consacrer un certain effort de calcul à la construction d'une solution initiale adéquate est encouragé car cela contribue à améliorer la convergence dans l'optimisation, améliorant ainsi l'efficacité de la méthode d'exploration de conception.

Publications

Cette contribution a donné lieu à la publication d'**un** article de revue et de **deux** articles dans des conférences internationales. La liste exhaustive de nos publications est donnée à la fin de ce mémoire.

Perspectives

Comme perspectives, il serait très utile d'implanter un générateur automatique de code qui concrétise les solutions trouvées en transformant le modèle architectural initial en un modèle prenant en compte la nouvelle affectation des fonctions aux threads. Aussi, et

© 2022 Bechir Zalila

pour garantir plus l'exactitude des systèmes produits, il serait bénéfique d'intégrer cette approche dans un processus de vérification formelle des systèmes TR^2E.

© 2022 Bechir Zalila

Chapitre 4

Vérification formelle des systèmes TR²E

Sommaire

4.1 Contexte général . . . 66
4.2 Contributions . . . 67
4.3 Modèle de tâches LNT . . . 68
4.3.1 Un modèle de tâches compatible avec Ravenscar . . . 69
4.3.2 Mapping de l'ordonnancement . . . 69
4.3.3 Mapping des communications . . . 76
4.3.4 Composition et synchronisation . . . 77
4.3.5 Discussion . . . 78
4.4 Transformation d'un modèle AADL vers LNT . . . 79
4.4.1 Règles de transformation . . . 80
4.4.2 Outillage . . . 81
4.5 Expérimentations . . . 83
4.5.1 Modélisation . . . 84
4.5.2 Génération de code . . . 84
4.5.3 Vérification formelle . . . 85
4.5.4 Analyse des résultats . . . 85
4.6 État de l'art . . . 86
4.7 Conclusion et perspectives . . . 87

Dans les chapitres précédents, nous nous sommes principalement intéressés aux aspects architecturaux des systèmes TR²E spécifiés à l'aide des ADLs. Dans ce chapitre, nous nous intéressons à l'aspect comportemental de ces systèmes, et particulièrement à la manière de vérifier formellement un système TR²E modélisé avec un ADL et décoré avec des éléments comportementaux.

Les méthodes formelles sont devenues une pratique recommandée lors de la conception de logiciels temps réel critiques. Pour être formellement vérifié, un système doit être spécifié avec un formalisme spécifique tel que les réseaux de Petri, les automates ou encore les algèbres de processus, ce qui nécessite une expertise formelle et devient très vite complexe surtout avec les grands systèmes. Dans ce chapitre, nous proposons un processus de transformation d'un modèle AADL enrichi par une annexe comportementale vers un

modèle d'algèbre de processus et nous décrivons comment cette dernière est utilisée pour l'intégration d'une phase de vérification formelle dans un processus de développement de systèmes TR²E. Nous avons élaboré une chaîne d'outils complète pour la transformation automatique des modèles et la vérification formelle des spécifications AADL. Nous avons validé notre contribution sur plusieurs études de cas dont deux seront reprises dans ce chapitre.

La contribution décrite dans ce chapitre est le fruit d'un travail de recherche d'équipe. Il s'agit des résultats des travaux de thèse de doctorat de Mlle Hana MKAOUAR [Mkaouar, 2019] réalisée dans le cadre d'une collaboration entre l'Université de Sfax et l'Institut Supérieur de l'Aéronautique et de l'Espace de Toulouse.

4.1 Contexte général

Le génie logiciel dans les domaines critiques tels que le transport, la santé, l'aéronautique et l'espace est un sujet assez délicat en informatique. Dans un tel contexte, les concepteurs sont souvent confrontés à de grands systèmes TR²E avec des exigences diverses. Plusieurs approches (modélisation, vérification, génération de code et tests) se sont concentrées sur la simplification de ces constructions complexes avec plus d'abstraction dans la conception du système et l'automatisation dans les chaînes d'outils de développement [Vyatkin, 2013]. Parmi ces approches, on note l'approche Model-Driven Engineering (MDE). C'est une tendance de développement basée sur les langages de modélisation, la transformation de modèles, la production de documentation et la génération automatique de code. Théoriquement, les approches MDE visent à abstraire les représentations du système et à permettre une évaluation cohérente du système depuis la spécification jusqu'à l'application finale. Dans la pratique, les outils supportant l'approche MDE assurent principalement la génération, l'analyse et la simulation automatiques de modèles et de code.

D'autres approches prometteuses sont les méthodes formelles, qui font référence à des techniques et outils mathématiquement rigoureux pour la spécification et la vérification des systèmes. Au cours des dernières décennies, les méthodes formelles sont devenues l'une des techniques préconisées dans l'ingénierie logicielle critique. En effet, ils sont maintenant acceptés dans les processus de certification (par exemple, DO-333 [RTCA, 2011]) comme moyen d'obtenir la certification par les autorités. Pour ces raisons, l'intégration de méthodes formelles dans les approches MDE semble gratifiante.

Cependant, leur application nécessite une expertise pas toujours facile à acquérir : le système considéré doit être spécifié avec un formalisme spécifique comme les automates et les réseaux de Petri ou encore les algèbres de processus, basé sur une sémantique formelle décrite à l'aide d'approches mathématiques, à explorer par des outils d'analyse dédiés. Contrairement à cela, la sémantique des langages de modélisation architecturaux tels que AADL et MARTE est souvent donnée en langage naturel (c'est-à-dire des documents standards et manuels). Ce manque de sémantique formelle rend les langages de modélisation inappropriés pour la vérification formelle, ils ne peuvent pas être explorés directement par les outils d'analyse formelle. Par conséquent, il est utile de fournir une génération automatique du modèle architectural à la spécification formelle qui peut être utilisée et réutilisée pour encourager la pratique des méthodes formelles et aider les concepteurs lors de la vérification du système.

© 2022 Bechir ZALILA

Dans ce contexte, nous avons intégré une phase de vérification formelle dans une approche MDE basée sur le langage AADL (Architecture Analysis & Design Language) [SAE, 2017a]. AADL est un langage architectural industriel pour des domaines critiques tels que l'avionique, l'électronique automobile, la robotique et l'espace. Il est considéré comme un leader de la modélisation en temps réel et classé dans [Malavolta *et al.*, 2013] parmi les langages les plus utilisés dans l'industrie. C'est une norme standardisée par la SAE [11] et sa deuxième version a été publiée en 2009 et révisée en 2016.

Dans ce chapitre, nous proposons et évaluons une transformation du langage AADL vers le langage LNT (LOTOS New Technology) qui est une algèbre de processus enrichie par des éléments de la programmation impérative. La transformation, AADL2LNT est implantée dans la suite d'outils OCARINA [Lasnier *et al.*, 2009b], un environnement de développement pour la modélisation AADL que nous avons réalisé pendant notre travail de thèse [Zalila, 2008], afin de générer automatiquement une spécification LNT prête à être analysée avec CADP [Garavel *et al.*, 2013] (Construction and Analysis of Distributed Processes), une boîte à outils d'analyse, développée depuis le milieu des années 1980. De plus, un fichier de script, écrit en langage SVL (Script Verification Language) [Garavel and Lang, 2002], est également généré pour faciliter la vérification avec CADP. Ce script assure la génération de l'espace d'états de la spécification LNT et la vérification d'un ensemble de propriétés génériques pour garantir l'absence de problèmes graves tels que la détection des interblocages, le test d'ordonnancement et la détection des échecs de connexion. Enfin, la phase de vérification se termine par la génération de résultats d'analyse conviviaux, facilement interprétés par des concepteurs experts non spécialistes en formel afin de les assister lors de la correction des erreurs.

Le reste de ce chapitre est organisé comme suit : La section 4.2 donne un aperçu rapide de nos contributions dans le contexte de la vérification formelle des systèmes TR^2E. Dans la section 4.3, nous développons une projection formelle LNT pour un modèle de tâches temps réel. La section 4.4 détaille la transformation AADL2LNT. Les résultats expérimentaux sont discutés dans la section 4.5. La section 4.6 traite des travaux connexes. Enfin, une conclusion et des travaux futurs terminent le chapitre dans la section 4.7.

4.2 Contributions

Notre contribution se compose de deux parties principales. Dans la première, nous décrivons et justifions une projection formelle d'un modèle de tâches temps réel conforme au profil Ravenscar [Burns *et al.*, 2004] pour les systèmes temps réel critiques. Cette projection est basée sur un modèle de tâches conventionnel inspiré de Liu et Layland [Liu and Layland, 1973] avec une sémantique rigoureuse et des exigences fortes définies par le profil Ravenscar. Elle est conçue pour être modulaire et compréhensible afin d'être facilement étendue et utilisée dans les approches MDE. Nous prenons principalement en charge les tâches périodiques et sporadiques, qui sont connectées de manière asynchrone et exécutées simultanément par un ordonnanceur préemptif à priorité fixe. Différentes caractéristiques du modèle sont spécifiées avec le langage formel LNT [Champelovier *et al.*, 2018; Garavel *et al.*, 2017], qui est une algèbre de processus basée sur deux standards : LOTOS et E-LOTOS [LOTOS, 1989]. LNT fournit des opérateurs suffisamment expressifs pour les données et le comportement avec des notations conviviales pour simplifier l'écriture et l'ex-

11. SAE International, anciennement *Society of Automotive Engineers* : `https://www.sae.org/`

© 2022 Bechir ZALILA

tension. Le langage LNT est supporté par la boîte à outils CADP. Cette boîte à outils propose diverses méthodes formelles comme la vérification de modèles et la simulation. C'est une boîte à outils bien expérimentée, utilisée dans de nombreuses applications industrielles (par exemple, par Airbus [Garavel and Hautbois, 1993]).

Dans la deuxième partie, nous décrivons une approche MDE basée sur le langage AADL et intégrant la vérification formelle lors de la phase de modélisation. Cela permet la détection précoce de problèmes profonds pouvant entraîner de graves erreurs dans le système final. Cette vérification est censée être automatique et transparente pour simplifier et encourager la pratique de méthodes formelles en génie logiciel. Notre choix du langage AADL est principalement motivé par l'existence d'une annexe comportementale standard de ce langage [SAE, 2017b]. L'utilisation de cette annexe permet d'incorporer des éléments comportementaux dans l'architecture. Ce sont précisément ces éléments qui donneront lieu au modèle LNT sujet de la vérification.

4.3 Modèle de tâches LNT

Le langage LNT [Garavel *et al.*, 2017] combine des fonctionnalités issues des algèbres de processus et des langages de programmation avec une sémantique dynamique basée sur les règles formelles SOS (Structural Operational Semantics). Une spécification LNT se compose de deux parties : une partie données qui définit les types et les fonctions ; et une partie contrôle qui définit le comportement (au sein des processus). La partie données est un langage pleinement impératif en syntaxe et en sémantique. La partie contrôle comprend presque toute la partie données et ajoute des constructions pour le comportement comme le choix non déterministe, le parallélisme des processus et la communication.

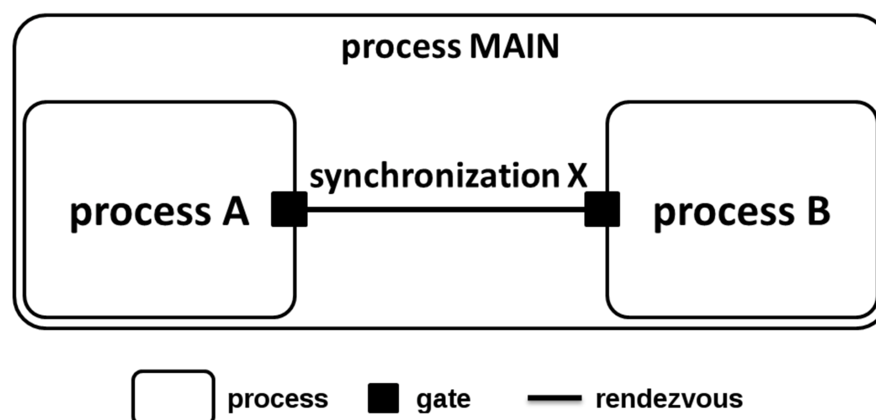

FIGURE 4.1 – Représentation graphique LNT [Mkaouar, 2019]

Dans une spécification LNT, le système est représenté par un ensemble de processus concurrents communiquants à travers des portes (*gates*) typés avec des canaux. Comme le montre la figure 4.1, un processus principal (*racine*) appelé impérativement **`MAIN`** doit être ajouté et définit le point d'entrée pour toute la spécification. La spécification représente une sémantique exécutable dans laquelle tous les processus parallèles commencent leur exécution et se terminent en même temps avec la possibilité de synchronisation à travers

des *rendez-vous* [12]. Nous avons donné dans [Mkaouar *et al.*, 2020a] un aperçu rapide de ce langage présentant tous les éléments nécessaires à notre modèle de tâches temps réel.

Dans cette première partie de notre contribution, nous visons à définir une projection formelle pour un système temps réel. Nous utilisons le langage LNT pour spécifier de manière générique un modèle de tâches standard. La syntaxe LNT combine des fonctionnalités de langages de programmation avec des primitives de concurrence adoptées à partir des algèbres de processus [Baeten, 2005], ce qui la rend appropriée pour spécifier des tâches concurrentes et gérer les analyses d'ordonnancement. La représentation formelle doit simuler l'ordonnancement, l'exécution et l'interaction des tâches. Idéalement, la tâche et l'ordonnanceur sont spécifiés séparément pour fournir un mapping compréhensible et modulaire.

Dans cette section, nous présentons d'abord le modèle de tâches considéré. Ensuite, nous développons le mapping proposé à travers trois parties principales : le mapping d'ordonnancement, la communication et la composition/synchronisation.

4.3.1 Un modèle de tâches compatible avec Ravenscar

Dans le contexte de la vérification, un système temps réel peut être considéré comme un ensemble de tâches coopératives et concurrentes s'exécutant à intervalles réguliers (tâches périodiques) ou suite à des événements spéciaux (tâches apériodiques ou sporadiques). Le système est ensuite abstrait en tant que modèle de tâches avec un ensemble de paramètres temps réel, masquant la complexité architecturale. Dans notre travail, nous nous appuyons sur un modèle de tâches conventionnel inspiré du modèle temps réel de Liu et Layland [Liu and Layland, 1973]. Formellement, on travaille avec un ensemble de k tâches notées $S = \tau_1, ..., \tau_k$ dont chaque τ_i est définie par deux paramètres C_i et T_i : C_i désigne la capacité ou WCET (Worst Case Execution Time); et T_i est considérée comme la période ou bien le délai relatif de chaque déclenchement.

De plus, des contraintes fortes sont également prises en compte puisque nous traitons des systèmes critiques qui nécessitent une certification pour être utilisés. Dans un tel contexte, mentionnons le profil Ravenscar [Burns *et al.*, 2004], qui est défini pour répondre aux exigences temps réel critiques (déterminisme, analyse d'ordonnancement, aptitude à la certification, etc.). Ce profil décrit un ensemble de restrictions des fonctionnalités de tâches Ada afin de permettre les analyses statiques pour les certifications des systèmes à haute intégrité. Le profil Ravenscar peut être appliqué au niveau du modèle en tant que sous-ensemble composé d'un ensemble statique de tâches en interaction, régies par un ordonnanceur préemptif à priorité fixe en plus d'autres contraintes définies en détail dans [Burns *et al.*, 2004].

Il a fallu donc sélectionner du langage LNT le sous-ensemble qui nous permet de modéliser des tâches en toute conformité avec le profile Ravenscar.

4.3.2 Mapping de l'ordonnancement

Le mapping d'ordonnancement concerne les représentations de la tâche et de l'ordonnanceur en LNT. Généralement décrites, les tâches sont projetées sur des processus LNT

12. Un rendez-vous est un mécanisme de synchronisation permettant à un thread d'attendre l'arrivée d'un autre thread à un niveau d'avancement précis avant de poursuivre, à son tour, son exécution. Certaines implantation des rendez-vous, permettent l'exécution d'un ensemble d'instructions avant la poursuite du travail des deux threads.

© 2022 Bechir Zalila

pour être exécutées simultanément, chaque tâche τ_i est représentée par un processus LNT, nommé **TASK**. Ces **TASK**s sont régies par un processus principal, nommé **SCHEDULER**, qui représente l'ordonnanceur : les **TASK**s sont synchronisées (via les portes et canaux LNT) avec le **SCHEDULER** pour être activées. Il faut noter que le langage LNT n'est pas une algèbre de processus temps réel. Il n'a pas d'opérateurs temporels et tous les processus parallèles démarrent et se terminent en même temps. Il existe des extensions temporisées pour les langages CADP (ET-LOTOS [Léonard and Leduc, 1998], RT-LOTOS [Courtiat *et al.*, 2000], etc.), mais actuellement, elles ne sont pas supportées par les outils existants. Néanmoins, l'utilisation du langage LNT reste suffisante pour notre propos, puisqu'il fournit une partie riche de données, utilisée pour spécifier des fonctionnalités temps réel et des algorithmes d'ordonnancement. Par conséquent, le temps fait partie du mapping LNT proposé et il est intelligemment inclus (si nécessaire) pour fournir des spécifications LNT avec des espaces d'états réduits. Nous définissons une variable **COUNTER** pour représenter le temps (une sorte de chronomètre logique pour compter les unités de temps), utilisée selon les besoins pour effectuer les calculs temporels (par exemple, expédition, préemption). De plus, nous définissons une variable **HYPERPERIOD** qui représente le PPCM de T_1, .., T_k, ainsi, le timer **COUNTER** reste borné (0 < **COUNTER** < **HYPERPERIOD**).

Dans ce qui suit, nous développons respectivement les définitions **TASK** et **SCHEDULER** LNT.

Mapping de TASK

Le processus **TASK** est conçu pour représenter la tâche comme une unité concurrente ordonnançable avec une séquence potentiellement infinie d'activations (appels ou jobs) par l'ordonnanceur. Pour spécifier le modèle de dispatching considéré, nous définissons un ensemble d'ordres d'activation projetés sur un type d'énumération LNT échangé entre **TASK** et **SCHEDULER** : **`T_Dispatch_Preemption`**, **`T_Preemption`**, **`T_Preemption_Completion`**, **`T_Dispatch_Completion`**, **`T_Completion`**, **`T_Error`** et **`T_Stop`**.

Un squelette du processus **TASK** est illustré dans l'exemple de code 4.1. Le processus déclare une porte LNT, nommée **ACTIVATION**, à synchroniser avec le processus **SCHEDULER**. Le comportement **TASK** est une boucle infinie dont le corps est une sélection non déterministe de choix afin de séparer les comportements d'exécution, d'erreur et de terminaison. Le comportement sélectionné est déterminé par la communication **ACTIVATION** avec ses différentes valeurs possibles.

L'automate à états d'une tâche (comme le montre la figure 4.2) est projeté dans la partie comportement d'exécution de **TASK**. La communication **ACTIVATION** définit les états de **TASK** : l'état courant est défini en fonction de l'ordre **SCHEDULER** reçu. Initialement, la **TASK** est supposée à l'état **Ready**. Elle est suspendue jusqu'à la réception d'un ordre **SCHEDULER** sur la porte **ACTIVATION**. Toutes les transitions de tâches de la figure 4.2 (**`resume`**, **`preempt`**, **`dispatch`** et **`complete`**) entre les états se traduisent par des suspensions au rendez-vous d'**ACTIVATION** avec **SCHEDULER**. À la réception d'un ordre d'activation (**`T_Dispatch_Completion`**, **`T_Preemption`**, **`T_Dispatch_Preemption`** et **`T_Preemption_Completion`**), **TASK** passe à l'état **Running**. Après l'exécution, la **TASK** envoie l'étiquette **`T_Completion`** au **SCHEDULER** signifiant que la **TASK** a accompli l'ordre d'activation et qu'elle n'est plus à l'état **Running**. À ce stade, en fonction de la commande reçue, **TASK** peut changer d'état comme suit :

© 2022 Bechir Zalila

Exemple 4.1 – Squelette d'une tâche LNT

```
process TASK [ACTIVATION: LNT_Channel_Dispatch,
-- other gate declarations
] is
   loop
      select
         select -- execution behavior
            -- a complete execution time
            ACTIVATION (T_Dispatch_Completion);
            []
            -- preemption
            ACTIVATION (T_Dispatch_Preemption);
            []
            ACTIVATION (T_Preemption)
            []
            ACTIVATION (T_Preemption_Completion);
         end select;
         ACTIVATION (T_Completion)
         []-- error behavior
         ACTIVATION (T_Error)
         []-- termination behavior
         ACTIVATION (T_Stop)
      end select
   end loop
end process
```

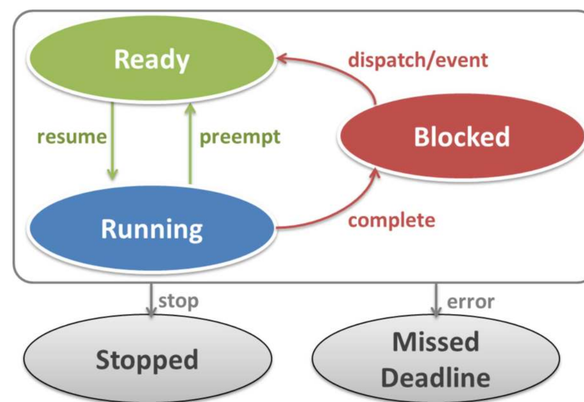

FIGURE 4.2 – Automate d'états d'une tâche [Mkaouar, 2019]

— **T_Dispatch_Completion** : la tâche démarre et termine l'exécution de la période en cours et passe à l'état **Blocked**,
— **T_Dispatch_Preemption** : la tâche démarre l'exécution dans la période courante mais avec une préemption, donc, revient à l'état **Ready**,
— **T_Preemption** : la tâche progresse en exécution mais sans atteindre le temps d'achèvement, revient donc à l'état **Ready**,
— **T_Preemption_Completion** : la tâche termine l'exécution de la période en cours et passe à l'état **Blocked**.

Le processus **TASK** peut également recevoir des ordres **T_Error** et **T_Stop**, qui sont respectivement utilisés pour marquer une échéance manquée et pour arrêter la simulation du système. Cela concerne les comportements d'erreur et de terminaison, ce qui conduit à définir deux états supplémentaires de tâche **Missed_Deadline** et **Stopped** inclus dans l'automate d'états de tâche comme le montre la figure 4.2, utilisés pour les fins de vérification.

Il est toutefois notable que les tâches périodiques et sporadiques sont représentées avec le même squelette LNT, tandis que la différence (mode de déclenchement) sera dans le mode d'activation contrôlé par le processus **SCHEDULER**. Tous les calculs temporels sont encapsulés dans le processus **SCHEDULER** qui garantit que les tâches périodiques sont exécutées avec des ordres réguliers, tandis que les tâches sporadiques reçoivent des ordres à des intervalles irréguliers en fonction de la réception d'événements d'invocation.

Dans cette contribution, nous supportons les communications asynchrones entre les tâches : une tâche peut être reliée à d'autres tâches. Au niveau du mapping LNT, les données et événements échangés sont projetés de manière générique à l'aide d'un type énuméré (étiquettes **DATA** et **EVENT**). Les connexions sont établies via les portes et canaux LNT. Ainsi, un processus **TASK** peut posséder de nombreuses portes selon les besoins de ses connexions. Les interactions **TASK** sont également contrôlées par des commandes **SCHEDULER** qui fixent les temps d'entrée et de sortie comme suit :

— **T_Dispatch_Preemption** (début du temps d'exécution) : la tâche reçoit des entrées,
— **T_Preemption_Completion** (fin d'une itération) : la tâche envoie des sorties,
— **T_Dispatch_Completion** (l'achèvement de l'exécution) : la tâche reçoit des entrées au moment du début et envoie des sorties au moment de la fin.

Mapping de SCHEDULER

Le processus **SCHEDULER** code l'algorithme d'ordonnancement pour simuler l'exécution des tâches. Il est synchronisé avec toutes les **TASK**s à travers les portes **ACTIVATION**. La construction de ce processus dépend de l'ensemble S des tâches et du protocole d'ordonnancement choisi. Conformément au profil Ravenscar, l'exécution des tâches est assurée par un ordonnancement préemptif dans lequel la priorité de chaque tâche est statique et l'ordonnanceur exécute toujours la tâche prête la plus prioritaire. A tout moment, si une tâche avec une priorité plus élevée devient prête, l'ordonnanceur exécute un changement de contexte en préemptant la tâche en cours d'exécution permettant à la tâche de priorité

© 2022 Bechir Zalila

plus élevée de reprendre l'exécution. Dans ce chapitre, nous considérons l'ordonnancement RMS (Rate Monotonic Scheduling [Liu and Layland, 1973]) avec un temps de changement de contexte négligeable [13]. Une tâche τ_i est définie par la donnée du couple (C_i, T_i) où l'indice i représente la priorité de la tâche, attribuée selon sa période T_i : la tâche ayant la plus petite période se trouve avoir la priorité la plus élevée. C_i est le WCET de la tâche. Pour ordonnancer τ_i, nous définissons également t^i_j pour la date de la j_{eme} activation et d^i_j pour la date de la j_{eme} échéance.

Exemple 4.2 – Squelette de l'ordonnanceur LNT

```
process SCHEDULER [
   ACTIVATION_1 : LNT_Channel_Dispatch, ...,
   ACTIVATION_k : LNT_Channel_Dispatch,
   NOTIFICATION_1 : LNT_Channel_Event, ...,
   NOTIFICATION_n : LNT_Channel_Event]
is
  var   -- initialization part
     S : LNT_Type_Task_Array,
     ..
  in
     S [i] := LNT_Type_Task (..);
     ..
     loop
        if (Counter < HYPERPERIOD) then
        -- operational part
           -- time allocation
           -- update task state
           -- task activation
           -- notification for sporadic task
        else -- termination part
           ACTIVATION_1 (T_Stop) ...
           ACTIVATION_K (T_Stop)
        end if
     end loop
  end var
end process
```

Dans l'exemple de code 4.2, nous incluons un squelette LNT du processus **SCHEDULER**. La déclaration de ce processus comporte k portes ($ACTIVATION_1$, ..., $ACTIVATION_k$) avec n portes supplémentaires si S contient des tâches sporadiques ($NOTIFICATION_1$, ..., $NOTIFICATION_n$), n étant le nombre de tâches sporadiques. Le comportement de **SCHEDULER** se compose de trois parties comme suit :

(1) Partie Initialisation : le processus **SCHEDULER** commence par un ensemble d'initialisations nécessaires aux calculs temporels, principalement le compteur **COUNTER** et l'ensemble de tâches S,

(2) Partie opérationnelle : cette partie implante l'algorithme d'ordonnancement. Tant que **COUNTER** n'a pas atteint l'**HYPERPERIOD**, **SCHEDULER** simule l'exécution des tâches à l'aide d'un algorithme que nous avons implanté et présenté dans [Mkaouar *et al.*, 2020a] (illustré graphiquement dans les figures 4.3 et 4.4),

(3) Partie d'arrêt : la terminaison des tâches n'est pas autorisée dans le profil Ravenscar, mais dans notre contexte de vérification formelle, nous définissons une terminaison

13. Il s'agit là d'une approximation réaliste car, dans le cas des priorités statique, le changement de contexte et la détermination de la tâche la plus prioritaire se résument à quelques instructions assembleurs seulement.

© 2022 Bechir Zalila

globale du système lorsque $COUNTER = HYPERPERIOD$[14]. Par conséquent, **SCHEDULER** envoie l'ordre **T_Stop** pour toutes les tâches pour marquer la fin de la simulation.

L'ensemble des tâches S est inclus statiquement sous la forme d'un tableau LNT. Les tâches sont indexées par leurs priorités fixées selon leurs périodes (T_1, ..., T_k), τ_1 (index 1) a la période la plus petite et la priorité la plus élevée et la tâche τ_k (index k) a la plus grande période et la plus basse priorité. Ce tableau représente les files d'attente des tâches prêtes dans le système : les tâches prêtes sont insérées/supprimées en tête/queue en fonction de leurs priorités dans la file d'attente prête et à tout moment l'ordonnanceur sélectionne la tâche ayant la priorité d'exécution la plus élevée. Dans notre mapping, nous utilisons une structure statique où les tâches ont des index fixes pour marquer leurs priorités, tandis que leurs états sont modifiables par le processus **SCHEDULER** lui-même. Chaque tâche τ_i est représentée par un tableau LNT contenant un ensemble de paramètres fixes et d'états mis à jour. Les tâches sporadiques sont ignorées jusqu'à la réception d'un événement (considérées comme bloquées). Par conséquent, **SCHEDULER** devrait être notifié pour chaque nouvel événement d'invocation entrant par les portes $NOTIFICATION_i$.

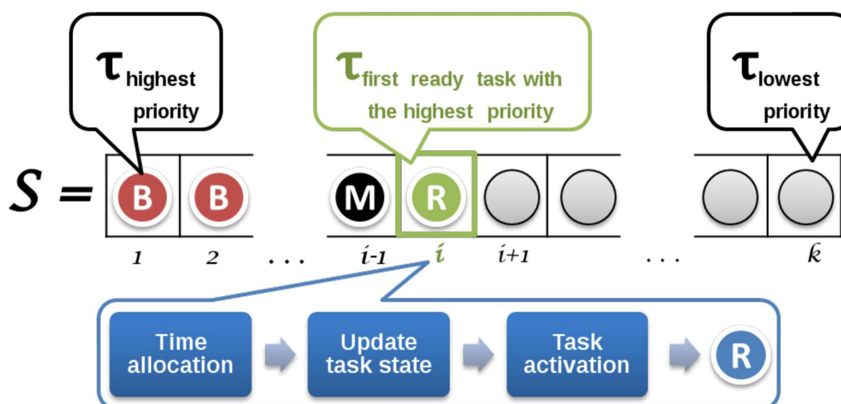

FIGURE 4.3 – Algorithme d'ordonnancement : tâche prête [Mkaouar, 2019]

Durant l'ordonnancement, le process **SCHEDULER** visite les tâches en boucle dans l'ordre de leurs priorités pour trouver et exécuter les tâches prêtes. De la priorité la plus élevée à la plus faible, chaque tâche est inspectée pour déterminer son état actuel (**Ready** ou **Blocked**), ainsi la première tâche τ_i à l'état **Ready**, est toujours la tâche prête avec la priorité la plus élevée comme le montre la figure 4.3. Formellement, l'ordonnanceur compare t^i_j et d^i_j avec la valeur **COUNTER**. Ainsi, il décide de l'état τ_i comme suit :

— **Blocked** : τ_i est une tâche sporadique inactive ou une tâche périodique en attente de sa prochaine période ($t^i_j > COUNTER$),

— **Missed_Deadline** : τ_i a raté son échéance ($d^i_j < COUNTER$). Dans ce cas, le message **T_Error** lui sera envoyé,

— **Ready** : τ_i est initialisée, préemptée ou activée ($t^i_j \leq COUNTER < d^i_j$).

Les tâches dans les états **Blocked** ou **Missed_Deadline** sont ignorées et le processus **SCHEDULER** passe à τ_{i+1}. Sinon, si τ_i est confirmée à l'état **Ready**, le processus **SCHEDULER** décide de l'exécution de τ_i qui passe à l'état **Running**.

14. Nous arrêtons la simulation à la valeur **HYPERPERIOD** de **COUNTER** car le comportement des tâches se répète à l'identique après cette valeur.

© 2022 Bechir ZALILA

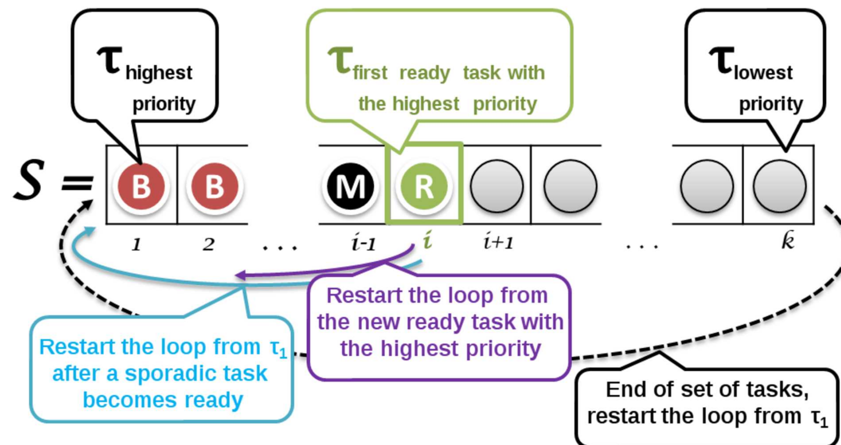

FIGURE 4.4 – Algorithme d'ordonnancement : boucles [Mkaouar, 2019]

A ce niveau, **SCHEDULER** peut exécuter τ_i entièrement, puis, il se déplace pour gérer d'autres tâches. Cela peut être suffisant pour un ordonnanceur non préemptif d'un ensemble de tâches périodiques. Cependant, dans notre contexte, une tâche avec une priorité plus élevée peut devenir prête à tout moment [15]. De même, une tâche sporadique avec une priorité plus élevée peut devenir prête suite à la réception d'un événement d'invocation. Ces cas peuvent conduire à préempter l'exécution de la tâche en cours d'exécution, ils doivent donc être pris en compte lors de l'ordonnancement. L'idée que nous avons implantée dans le comportement du processus **SCHEDULER** consiste à considérer à chaque évolution du système s'il y a des nouvelles tâches prêtes comme le montre la figure 4.4. Dans le reste de cette partie, nous expliquons ce comportement en détaillant ses trois étapes réalisant l'exécution d'une tâche à l'état **Ready** : allocation du temps, mise à jour de l'état de la tâche et activation de la tâche. De plus, nous incluons une section de vérification des tâches sporadiques ajoutée si S contient des tâches sporadiques.

Allocation du temps

Cette étape calcule le temps d'exécution de la période courante pour une tâche τ_i et prépare un ordre d'activation qui sera envoyé à l'étape d'activation de la tâche. Nous rappelons que τ_i est la tâche prête actuelle avec la priorité la plus élevée. Nous définissons la variable **Allocated_Time** pour calculer le temps d'exécution requis afin de réaliser l'exécution dans la période en cours, ou bien une partie de ce temps d'exécution, puisque τ_i peut être préemptée par une autre tâche plus prioritaire.

Mise à jour de l'état des tâches

À ce stade, τ_i est considéré à l'état **Running**, le processus **SCHEDULER** incrémente **COUNTER** avec **Allocated_Time** et met à jour le tableau des tâches pour les prochaines

15. Plus précisément, les tâches sporadiques sont déclenchées suite à la réception d'événements envoyés par d'autres tâches. Selon la sémantique AADL, les envois d'événements ne sont effectués qu'à la fin du traitement d'une tâche. Ceci simplifie beaucoup le travail (et l'implantation) de l'ordonnanceur puisque ce dernier ne doit vérifier si une tâche sporadique est éligible qu'à la fin du traitement de chaque tâche.

© 2022 Bechir ZALILA

activations. Dans le cas d'une préemption, **SCHEDULER** conserve l'état de la tâche et enregistre le temps exécuté de τ_i afin de compléter le reste ultérieurement. En cas de non-préemption, **SCHEDULER** prépare la tâche pour une nouvelle période.

Activation de la tâche

Le processus **SCHEDULER** envoie à τ_i son ordre courant avec la porte $ACTIVATION_i$. Il attend une notification **T_Completion** de τ_i puis passe au calcul de τ_{i+1}. En cas de dépassement de délai, le label **T_Error** sera le dernier ordre envoyé à τ_i, puisqu'elle sera ignorée dans la suite de la simulation.

Cas des tâches sporadiques

L'état global de l'ensemble des tâches peut changer après chaque activation de tâche (échange d'événements et de données, augmentation de **COUNTER**, etc.). En particulier, les tâches sporadiques peuvent être activées par les événements d'invocation et peuvent passer à l'état **Ready**. Elles doivent donc être prises en compte durant l'ordonnancement avec d'autres tâches périodiques. A cet effet, après chaque activation de tâche, le processus **SCHEDULER** consulte toutes les portes $NOTIFICATION_i$ et mets à jour l'état des tâches sporadiques le cas échéant.

4.3.3 Mapping des communications

Dans le cadre de cette contribution, les tâches peuvent être connectées les unes aux autres pour échanger de manière asynchrone des données ou des événements. Dans le cas sporadique, chaque tâche possède au moins une connexion de type événement nécessaire à son activation : la réception d'un événement d'invocation active la tâche sporadique qui peut passer à l'état **Ready**.

Les processus LNT communiquent par rendez-vous bloquants bidirectionnel sur leurs portes respectives. Le rendez-vous LNT sur une porte (*gate*) permet la synchronisation de n processus (plusieurs envois et réceptions en même temps). Dans notre cas, nous n'avons pas besoin d'une synchronisation aussi avancée entre les processus. On considère les portes de manière unidirectionnelle et on utilise uniquement la synchronisation entre paire de processus (émetteur et récepteur). Les connexions inter-tâches asynchrones ne peuvent pas être projetées directement avec les synchronisations des portes LNT car elles dénotent des rendez-vous bloquants et les rendez-vous sont interdits par le profil Ravenscar. Pour cette raison, nous ajoutons dans notre modèle concurrent un processus auxiliaire appelé **CONNECTOR** pour représenter la connexion au moyen des objets protégés Ravenscar. **CONNECTOR** a deux portes principales (**INPUT** et **OUTPUT**) et une variable pour enregistrer les données/événements qui circulent à travers lui.

Dans l'exemple de code 4.3, nous donnons le squelette du processus **CONNECTOR**. Le comportement du processus se compose de trois parties via une boucle infinie dont le corps est une instruction **select** pour séparer les données/événements d'envoi et de réception et la notification sporadique. Ainsi, une seule opération peut être exécutée à la fois et le choix est résolu par la possibilité de communication sur les portes.

© 2022 Bechir Zalila

Exemple 4.3 – Squelette d'un connecteur LNT

```
process CONNECTOR [
    INPUT: LNT_Channel_Port,
    OUTPUT: LNT_Channel_Port,
    NOTIFICATION: LNT_Channel_Event]
    (Queue_Size: Nat)
is
    loop
        select
            -- inputs of event/data part
            INPUT ()
            []
            -- output of event/data part
            OUTPUT ()
            []
            -- sporadic part
            -- needed to notify SCHEDULER
            -- when receiving a new event
            NOTIFICATION ()
        end select
    end loop
end process
```

Chaque connexion entre deux **TASK**s est projetée sur un **CONNECTOR** synchronisé (point de rendez-vous LNT [16]) avec l'émetteur sur **INPUT** et le récepteur sur **OUTPUT** qui assure l'atomicité des deux opérations (émission et réception) et l'unicité de **TASK** en attente à tout moment (respectivement, sur les portes **INPUT** et **OUTPUT**).

Les données sont enregistrées et conservées jusqu'à la prochaine réception. Chaque fois qu'une nouvelle entrée est reçue, la précédente est écrasée. En revanche, les événements sont mis en file d'attente dans une liste LNT avec une taille fixe définie. Nous utilisons une **FIFO** non bloquante, dans laquelle la nouvelle entrée entrante écrase les événements précédents en cas de débordement. Dans le cas d'une **FIFO** vide, la **TASK** reçoit un message **EMPTY** sans blocage.

Puisque nous considérons un événement d'invocation spécial pour l'activation de la tâche sporadique, nous ajoutons une troisième porte au processus **CONNECTOR** (nommée **NOTIFICATION**) à synchroniser avec le processus **SCHEDULER**. Nous utilisons cette porte pour avertir ce dernier processus de chaque nouvelle réception. Ainsi, il considère la tâche concernée parmi les tâches à ordonnancer.

4.3.4 Composition et synchronisation

Après avoir donné dans les sous-sections précédentes les modèles des 3 briques de base (**TASK**, **SCHEDULER** et **CONNECTOR**), nous expliquons dans cette partie comment ces briques doivent être composées et synchronisées afin de former un modèle de tâches concurrentes vérifiables formellement. Tous les processus LNT décrits doivent être composés (connectés) pour former le système principal. Cette étape est assurée par l'instruction de composition LNT pour le parallélisme et la synchronisation des processus **TASK**, **CONNECTOR** et **SCHEDULER**. Ainsi, nous assemblons l'ensemble du système dans le cadre du processus racine **MAIN**.

16. Il est utile de préciser ici qu'il s'agit d'un rendez-vous LNT qui n'implique pas le blocage des deux **TASK**s en même temps et par conséquent, il ne s'agit pas d'un rendez-vous entre tâches au sens synchrone du terme.

© 2022 Bechir Zalila

Exemple 4.4 – Type et canal LNT pour composer **TASK** et **SCHEDULER**

```
type LNT_Type_Dispatch is
    T_Dispatch_Completion ,
    T_Dispatch_Preemption ,
    T_Preemption ,
    T_Preemption_Completion ,
    T_Completion ,
    T_Error ,
    T_Stop
end type

channel LNT_Channel_Dispatch is
    (LNT_Type_Dispatch)
end channel
```

Exemple 4.5 – Exemple de processus LNT **MAIN**

```
process MAIN [
    ACTIVATION_1: LNT_Channel_Dispatch ,
    ACTIVATION_2: LNT_Channel_Dispatch ,
    NOTIFICATION_1: LNT_Channel_Event ,
    SEND_A: LNT_Channel_Port ,
    RECEIVE_A: LNT_Channel_Port
] is
    par
     ACTIVATION_1, RECEIVE_A->
     TASK_CONSUMER[ACTIVATION_1, RECEIVE_A]
     ||
     NOTIFICATION_1, SEND_A, RECEIVE_A->
     CONNECTOR[SEND_A, RECEIVE_A, NOTIFICATION_1]
     ||
     SEND_A, ACTIVATION_2->
     TASK_PRODUCER[ACTIVATION_2, SEND_A]
     ||
     ACTIVATION_1, ACTIVATION_2, NOTIFICATION_1->
     SCHEDULER[ACTIVATION_1, ACTIVATION_2,
               NOTIFICATION_1]
    end par
end process
```

Il est utile de préciser qu'un ensemble de types et de canaux LNT est utilisé pour les synchronisations **TASK**-**CONNECTOR**, **CONNECTOR**-**SCHEDULER** et **TASK**/**SCHEDULER**. Par exemple, nous incluons dans l'exemple 4.4 le type et le canal pour la synchronisation **TASK**-**SCHEDULER**.

Nous donnons également un exemple de modèle de tâche pour le processus **MAIN** dans l'exemple de code 4.5. Une représentation graphique de ce modèle est donnée dans la figure 4.5. Le modèle de tâche initial (τ_1, τ_2) consiste en une tâche périodique connectée à une autre tâche sporadique exécutée et un ordonnanceur (exemple classique producteur/consommateur). La spécification LNT obtenue contient cinq processus synchronisés dans le processus principal **MAIN**. La composition **par** (*parallèle*) est utilisée globalement pour les synchronisations suivantes :

— Les processus **TASK_CONSUMER** et **CONNECTOR** sont synchronisés sur la porte **RECEIVE_A**,

— Les processus **CONNECTOR** et **TASK_PRODUCER** sont synchronisés sur la porte **SEND_A**,

— Les processus **CONNECTOR** et **SCHEDULER** sont synchronisés sur la porte **NOTIFICATION_1**,

— Les processus **TASK_CONSUMER**, **TASK_PRODUCER** et **SCHEDULER** sont synchronisés sur les portes **ACTIVATION_1** et **ACTIVATION_2**.

4.3.5 Discussion

La spécification LNT obtenue (l'ensemble des définitions LNT composées dans le processus **MAIN**) représente une sémantique exécutable formelle pour un modèle de tâches temps réel, où les **TASK**s sont connectées via les **CONNECTOR** et ordonnancés par le processus **SCHEDULER**. Ce mapping est suffisamment flexible pour prendre en charge divers

© 2022 Bechir Zalila

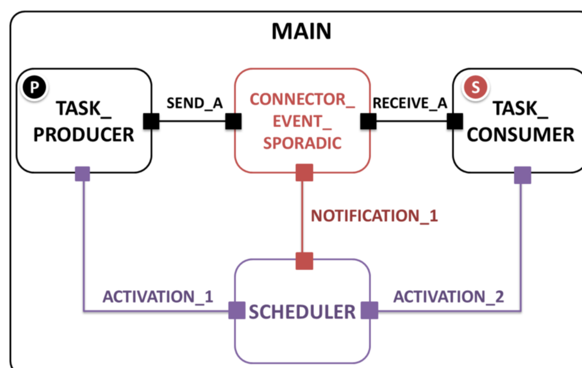

FIGURE 4.5 – Exemple de représentation graphique du processus **MAIN** [Mkaouar, 2019]

modèles de tâches avec des modifications mineures : tâches périodiques/sporadiques, tâches indépendantes/communiquantes et tâches préemptives/non préemptives.

De plus, les fonctionnalités temps réel sont conçues de manière modulaire, ce qui augmente l'extensibilité du mapping : chaque fonctionnalité peut être complétée ou étendue séparément. Par exemple, une tâche peut être facilement enrichie avec plus de comportement qui peut être spécifié dans un processus ou une fonction LNT distincts et simplement appelé dans le squelette **TASK**. Ceci facilitera beaucoup la génération automatique de spécification LNT.

De même, le mapping d'ordonnancement peut être étendu avec d'autres algorithmes d'ordonnancement puisque nous spécifions un **SCHEDULER** explicite qui encapsule tous les calculs d'ordonnancement. Par exemple, nous avons développé, en plus de RMS (*Rate Monotonic Scheduling*), l'algorithme d'ordonnancement EDF (*Earliest Deadline First*) basé sur un ordonnancement à priorités dynamiques. Le squelette *SCHEDULER* est conservé dans ce cas aussi. Les modifications peuvent être limitées dans la partie opérationnelle, principalement, l'opération d'attribution de temps. Les autres manipulations (mise à jour de l'état de la tâche, activation de la tâche et vérification de la tâche sporadique) peuvent être conservées, et ainsi les processus **TASK** et **CONNECTOR** n'ont besoin d'aucune modification. Ceci, est aussi un grand atout pour une approche fondée principalement sur la génération automatique de code comme nous allons voir dans la prochaine section.

4.4 Transformation d'un modèle AADL vers LNT

Dans cette section, nous discutons le second volet de notre contribution, à savoir l'applicabilité de notre mapping LNT décrit dans la section précédente en tant que pratique d'ingénierie logicielle. Nous visons à intégrer la vérification formelle dans un processus MDE basé sur le langage AADL. Comme le montre simplement la figure 4.6, tout au long du processus de développement, la représentation du système prend différentes formes (spécification écrite, modèle architectural, code source) sur de nombreuses phases (spécification, modélisation et génération de code). Lors de la phase de modélisation, la vérification formelle du modèle AADL semble utile et complémentaire aux analyses syntaxiques et sémantiques traditionnelles. À cette fin, nous définissons la transformation (AADL2LNT) du modèle AADL initial en une spécification LNT basée sur le mapping proposé. Cette trans-

© 2022 Bechir Zalila

formation permet de générer automatiquement une spécification LNT conforme à la boîte à outils CADP et prête à être vérifiée formellement.

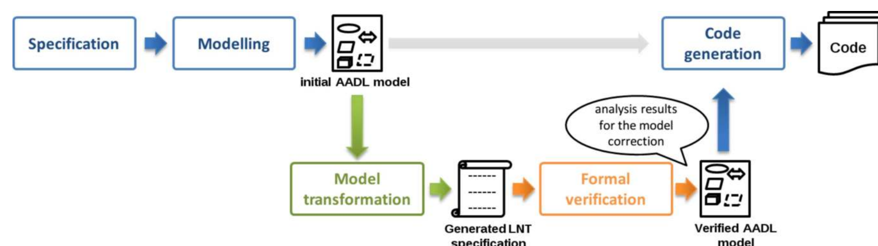

FIGURE 4.6 – Processus MDE basé sur AADL [Mkaouar, 2019]

Dans le reste de cette section, nous décrivons les règles de transformation AADL2LNT. Enfin, nous décrivons nos implantations permettant la définition d’une chaîne d’outils pour un processus de développement basé sur le langage AADL.

4.4.1 Règles de transformation

Le langage AADL décrit différents concepts des systèmes TR²E avec une sémantique riche détaillée dans sa norme [SAE, 2017a]. Le langage couvre de nombreux aspects importants (exigences temporelles, comportements de défauts et d’erreurs, partitionnement temporel et spatial, propriétés de sécurité, etc.) qui ne peuvent pas être entièrement analysés en une seule approche. Selon nos objectifs de vérification, nous définissons un sous-ensemble AADL dont les éléments sont représentés dans la figure 4.7. De plus, la prise en compte du profil Ravenscar nécessite des restrictions supplémentaires appliquées au niveau du modèle, ce qui signifie que le sous-ensemble AADL doit être conforme aux exigences du profil.

Dans notre contexte, nous visons à définir une sémantique formelle exécutable du modèle AADL vu comme un ensemble de tâches communicantes dans un contexte temps réel. Cette abstraction permet différentes alternatives de vérification, telles que l’analyse d’ordonnancement, la simulation d’exécution de thread et la vérification des propriétés de communication. Par la suite, le modèle AADL subira une transformation pour produire une spécification LNT.

La transformation de modèle joue un rôle crucial dans MDE pour divers objectifs (modélisation, optimisation et analyse). C’est le mécanisme de génération d’un modèle cible basé sur des informations extraites d’un modèle source. Cette opération est basée sur notre mapping LNT et nécessite un ensemble de nouvelles définitions dans ce langage pour couvrir le sous-ensemble AADL considéré. La transformation AADL2LNT est décrite avec un ensemble de règles de correspondance entre AADL et LNT (résumées dans la figure 4.7 et détaillées dans [Mkaouar, 2019] et [Mkaouar *et al.*, 2020a]).

Le mapping LNT (présenté dans la section 4.3.1) est utilisé comme suit : nous traduisons chaque composant de thread AADL en un processus **TASK**; nous représentons les ports AADL par les portes LNT; les connexions de ports AADL sont projetées via des processus **CONNECTOR**; et le processeur AADL devient le processus **SCHEDULER**. La figure 4.7 représente graphiquement ces règles de transformation de base, qui sont appliquées en quatre étapes :

© 2022 Bechir ZALILA

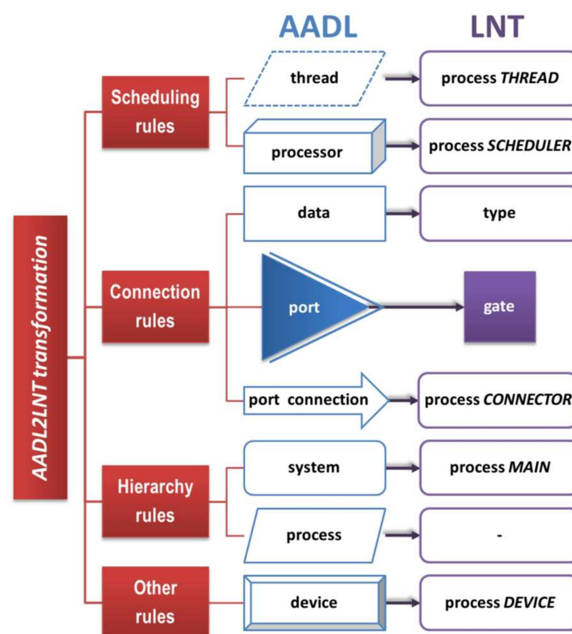

FIGURE 4.7 – Règles de transformation AADL vers LNT [Mkaouar, 2019]

(1) **Règles d'ordonnancement :** Ces règles décrivent les transformations des composants actifs AADL (threads, processeurs) en des structures LNT,

(2) **Règles de connexions :** Ces règles décrivent les transformation des composants passifs AADL (données) et des connexions AADL en structures LNT,

(3) **Règles hiérarchie :** Ces règles décrivent comment les structures LNT construites doivent êtres composées ensembles. Il est notable ici que les composants AADL de type **`process`** ne donnent lieu à aucune génération du coté LNT. Un processus AADL est défini comme étant la zone d'adressage et tout le comportement est modélisé dans les composants AADL de type **`thread`**,

(4) **Autres Règles :** Ces règles décrivent d'autres transformations qui ne se classent pas parmi les 3 précédentes (essentiellement les devices AADL).

4.4.2 Outillage

La description de la transformation du modèle étant définie, nous arrivons à la phase d'implantation pour fournir une génération automatique de la spécification LNT à partir d'un modèle AADL donné. Une description détaillée de nos implantations a été publié dans [Mkaouar *et al.*, 2018]. Dans cette partie, nous décrivons brièvement la chaîne d'outils obtenue, représentée sur la figure 4.8, basée sur OCARINA [Lasnier *et al.*, 2009b] pour la modélisation architecturale et la génération de code, et CADP [Garavel *et al.*, 2013] pour la vérification formelle.

— OCARINA est une suite d'outils libre pour la modélisation et la génération de code à partir de modèles AADL développée depuis 2004 et déployée sur GitHub dans le cadre du projet OpenAADL [17]. OCARINA peut être utilisée comme un compilateur autonome

17. **`https://github.com/OpenAADL/ocarina`**.

© 2022 Bechir ZALILA

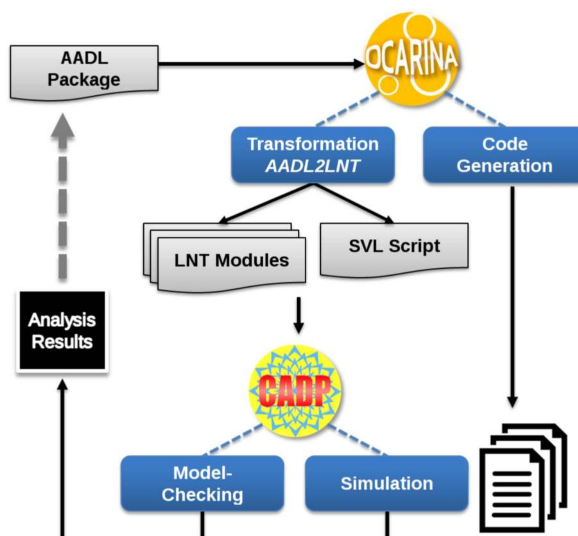

FIGURE 4.8 – Chaînes d'outils OCARINA/CADP [Mkaouar, 2019]

pour le langage AADL avec le support de certaines annexes ARINC653, EMV2 et REAL. La suite d'outils est adaptée pour une approche MDE car elle fournit les analyses de base (syntaxique et sémantique), des manipulations avancées de modèles, une vérification formelle (réseaux de Petri) et une génération de code (vers POLYORB-HI le runtime AADL),

— CADP [18] est une boîte à outils pour la conception et la vérification de systèmes concurrents, développée depuis 1986. Elle est disponible avec des licences académiques et commerciales. En plus de LNT, elle prend en charge de nombreux autres langages d'entrée tels que LOTOS, FSP et EXP. Elle fournit également un langage de script SVL (Script Verification Language) [Garavel and Lang, 2002] pour la description des scénarios d'analyse. La boîte à outils propose un ensemble complet d'outils pour la spécification, la simulation interactive, la vérification (model-checking, contrôle d'équivalence, etc.), l'évaluation des performances, etc. Pour faire face aux systèmes complexes, CADP propose un ensemble de techniques de vérification telles que l'analyse d'atteignabilité, la vérification à la volée et la vérification distribuée.

Extension d'OCARINA

La transformation AADL2LNT proposée a été intégrée à la suite d'outils OCARINA. Le compilateur d'OCARINA est conçu avec une architecture modulaire distinguant trois parties : une bibliothèque centrale (un ensemble de routines de base utilisées par les autres parties), les parties frontales (pour les analyses de modèles) et les parties dorsales (pour les manipulations et générations de modèles). Différentes manipulations de modèles sont traitées à l'aide des AST (*Abstract Syntax Tree*) qui sont la représentation interne des modèles (AADL, annexes et autres langages). La transformation AADL2LNT est intégrée dans le dépôt GitHub officiel d'OCARINA.

Généralement décrite, la transformation du modèle AADL2LNT est implantée sous la forme d'une nouvelle partie dorsale d'OCARINA. Le modèle AADL doit d'abord être ana-

18. `http://cadp.inria.fr`.

© 2022 Bechir ZALILA

lysé avec succès avec la partie frontale, puis transformé en un arbre syntaxique. Nous n'utilisons pas de langages de transformation de modèle pour notre génération. Nous appliquons directement les règles de transformation sur l'arbre syntaxique pour former un autre arbre syntaxique LNT de la même manière décrite dans le chapitre 2. Ensuite, cet AST est parcouru afin de produire les fichiers de code source (**`*.lnt`**). En plus des modules LNT, un fichier de script (**`demo.svl`**) est également généré, contenant un ensemble d'opérations spécifiées dans le langage SVL pour *orchestrer* la phase d'analyse. Ce fichier est généré pour chaque système AADL.

Vérification formelle avec CADP

La vérification formelle permet aux concepteurs de prouver qu'un système satisfait ses exigences. Les techniques de vérification sont appliquées sur un modèle mathématique abstrait du système (espace d'états du système), construit selon la sémantique du langage de spécification considéré. De plus, les propriétés à vérifier (exigences-système) doivent être spécifiées sous forme de graphes ou de propriétés logiques temporelles, en utilisant des formalismes spécifiques. Ensuite, la vérification peut être effectuée par les outils d'analyse. Dans notre travail, nous traitons principalement de la technique de model-checking, qui consiste à vérifier si le système satisfait une propriété donnée spécifiée avec une logique temporelle. Nous n'entrons pas dans les détails de cette vérification car ceci est en dehors du cadre de cette contribution.

Sur la base des sorties générées par Ocarina une phase de vérification formelle peut être effectuée par la boîte à outils CADP à l'aide du script SVL qui guide la compilation de la spécification LNT et la vérification d'un ensemble de propriétés comportementales et temporelles. La spécification LNT générée est traduite dans un système à transitions étiquetées (*Labeled Transition System*, LTS) pour être explorée à l'aide des vérificateurs de modèles CADP. Pour automatiser cette opération, nous incluons les propriétés à vérifier dans le script SVL. À l'aide de la déclaration de propriétés SVL, nous définissons un ensemble de propriétés génériques pour vérifier certaines exigences des systèmes temps réel telles que l'absence d'interblocages, l'ordonnançabilité, l'absence de problèmes de communication (par exemple, la perte de données) et de file d'attente (par exemple, le débordement de tampons).

Nous avons ainsi élaboré une chaîne d'outils permettant une vérification automatique et transparente du modèle AADL. La transformation est effectuée par la ligne de commande de l'outil Ocarina, puis, le script SVL généré est simplement invoqué pour commencer la phase de vérification avec la boîte à outils CADP. Enfin, les résultats de l'analyse aident les concepteurs dans la correction et l'amélioration du modèle. Cette opération peut être appliquée de manière itérative après chaque modification, tout au long du processus de développement, jusqu'à la génération de l'application finale.

4.5 Expérimentations

La transformation AADL2LNT a été testée avec divers exemples. Dans cette section, nous donnons un rapide retour sur expérience à propos de deux études de cas : FCS (*Flight Control System*, système de contrôle de vol) et LFR (*Line Follower Robot*, robot suiveur de ligne), pour illustrer les tâches périodiques/sporadiques et la transformation du modèle de connexions données/événements et l'analyse formelle. Nous parlons de la phase de

© 2022 Bechir Zalila

modélisation. Ensuite, nous exposons sommairement nos résultats expérimentaux sur la génération de code avec OCARINA et l'analyse formelle avec CADP. Les résultats détaillés de ces études de cas sont publiées dans [Mkaouar *et al.*, 2020a] et [Mkaouar *et al.*, 2020b].

4.5.1 Modélisation

Cette phase consiste à définir les modèles de tâches (paramètres temps réel, connexions, etc.) et à modéliser l'ensemble du système avec le langage AADL.

Système de contrôle de vol

Nous considérons cet exemple pour illustrer les tâches périodiques et les connexions de données. FCS est un système TR²E avionique critique pour le contrôle d'un avion. Ce système contrôle l'altitude, la trajectoire et la vitesse d'un avion. Nous considérons une version simplifiée, composée de 7 tâches périodiques ($S_{FCS} = \{\tau_1...\tau_7\}$) qui collaborent en échangeant des données, afin d'envoyer un retour d'informations à la surface et de contrôler les différents paramètres de vol.

Brièvement décrit, le modèle FCS se compose d'un premier sous-ensemble de tâches (FL (Feedback Law), FF (Feedback Filter) et AP (Acceleration Position Acquisition)), qui est exécuté à une période de $10ms$ pour acquérir l'état du système (angles, position, accélération) et calculer un résultat, afin d'envoyer l'ordre final pour contrôler les paramètres de vol. Un sous-ensemble de tâches de la boucle de pilotage (PL (Piloting Law) et PF (Piloting Filter)) est exécuté à une période de $40ms$ pour déterminer l'accélération à appliquer. Enfin, un sous-ensemble de tâches de boucle de navigation (NL (Navigation Law) et NF (Navigation Filter)) est exécuté à une période de $120ms$ pour déterminer la position à atteindre.

Robot suiveur de ligne

LFR est notre deuxième étude de cas pour tester des tâches sporadiques et des connexions de données-événements. Ce robot est une machine qui suit une ligne noire sur une surface blanche. Ce système utilise des capteurs pour détecter la ligne et des unités de contrôle pour prendre des décisions de mouvement et commander les moteurs de droite et de gauche (pilotant chacun une des roues). Les capteurs du robot contrôlent régulièrement le suivi de la ligne noire et envoient des informations aux unités de contrôle. Les moteurs ne sont commandés (pour s'allumer/s'éteindre) que si le robot perd la ligne, ce qui consiste en une action non périodique. Ainsi, le comportement du moteur serait mieux modélisé avec des tâches sporadiques.

Le modèle de tâche LFR ($S_{LFR} = \{\tau_1...\tau_6\}$) considère les côtés droit/gauche du robot de manière similaire. Chaque coté contient 3 tâches échangeant des données-événements : une tâche périodique pour la détection, une tâche périodique pour le contrôle et une tâche sporadique pour allumer/éteindre le moteur.

4.5.2 Génération de code

La complexité de la transformation AADL2LNT dépend principalement du nombre de threads AADL et de connexions entre ports : à partir de modèles avec des threads indépendants, on obtient des modèles LNT simples sans les synchronisations **CONNECTOR**.

© 2022 Bechir ZALILA

Alors qu'avec les modèles hautement connectés, le nombre de processus requis augmente considérablement et la complexité de la spécification LNT sous-jacente augmente également.

La génération automatique avec Ocarina couvre toutes les étapes nécessaires pour la transformation d'AADL vers LNT et élimine sa complexité, notamment dans le mapping **SCHEDULER** qui est moins générique par rapport aux autres processus générés (**THREAD_***, **DEVICE_*** et ***_CONNECTOR**). Une autre difficulté éliminée avec notre génération AADL2LNT réside dans la phase de composition et de synchronisation : le mapping du processus **MAIN** semble délicat puisque nous traitons beaucoup d'instances de processus et de portes, notamment pour le mapping des connexions de ports, dans lesquelles différentes déclarations hiérarchiques (au niveau du processus et du système) sont effectuées au niveau de **MAIN** qui est généré automatiquement par Ocarina.

4.5.3 Vérification formelle

Après la génération AADL2LNT, diverses analyses peuvent être effectuées par la boîte à outils CADP. Dans notre travail, nous utilisons deux techniques formelles importantes : la simulation (par exemple avec le simulateur OCIS) et le model-checking. La phase d'analyse est décrite dans le script **demo.svl** et consiste en deux étapes :

(1) Génération de l'espace d'états : il s'agit d'une étape importante pour permettre la vérification des spécifications LNT. Une traduction de LNT en LOTOS est d'abord appliquée. Ensuite, une génération d'un LTS (système de transitions étiquetées) est effectuée.

(2) La vérification : Après la génération de l'espace d'états, nous arrivons à la phase de model-checking. Comme mentionné précédemment, nous utilisons la déclaration de propriétés SVL pour spécifier les propriétés à vérifier. Cette description peut être pilotée par un ensemble de paramètres. Elle contient un ensemble d'énoncés de vérification, tel que l'énoncé de vérification de logique temporelle qui intègre une formule de logique temporelle à vérifier.

4.5.4 Analyse des résultats

Un des problèmes majeurs dans les approches formelles AADL existantes concerne l'utilisabilité des résultats d'analyse obtenus. Dans notre travail, nous fournissons une sortie à la fois simple et conviviale, ce qui est facilement interprété par des concepteurs experts non spécialistes dans la vérification formelle.

Sur la base d'une définition traçable de la spécification LNT et des propriétés vérifiées SVL, nous préservons les informations du modèle AADL initial qui se distingue facilement dans les résultats affichés. La traçabilité est d'abord garantie lors de la génération du modèle : la spécification LNT conserve tant que cela est possible les noms des composants AADL et des connexions de ports, qui sont utilisés dans les règles de nommage des variables, portes et processus LNT. De plus, l'utilisation de propriétés paramétrées et commentées favorise la traçabilité et donne des résultats compréhensibles. Les paramètres sont utilisés pour représenter les connexions de ports et les threads par leurs noms AADL initiaux. Ainsi, chaque connexion de threads ou de ports peut être vérifiée séparément et ainsi, les pannes sont rapidement localisées dans le modèle AADL.

© 2022 Bechir Zalila

4.6 État de l'art

La littérature regorge d'approches formelles tournant autour des ADLs en général et du langage AADL en particulier. Dans cette section, nous étudions les travaux basés sur une transformation de modèle AADL afin de réaliser une vérification formelle. Ces approches sont regroupées en trois familles, selon la nature du modèle d'entrée AADL (avec ou sans annexes et dépendant de la nature de l'annexe).

La première famille comprend des transformations de modèles AADL sans annexes. Ces approches se contentent de la sémantique AADL décrite dans sa norme [SAE, 2017a] qui est suffisante pour simuler formellement le système et vérifier un ensemble de propriétés comportementales telles que l'absence d'interblocage et de famine. La seconde famille est spécialisée dans le comportement et l'analyse des erreurs en utilisant des modèles AADL enrichis de son annexe de modèles d'erreur EMA [SAE, 2015]. Ceci permet l'analyse de la performance et de la sûreté de fonctionnement comme dans [Rugina *et al.*, 2008] et [Bozzano *et al.*, 2010]. La troisième famille, à laquelle nous appartenons, représente les travaux sur le langage AADL avec son annexe comportementale (BA) [SAE, 2017b]. Dans ce qui suit, nous détaillerons quelques transformations de cette dernière famille.

Dans [Chkouri *et al.*, 2009], les auteurs définissent une transformation dans le langage BIP, puis le modèle BIP est transformé en modèles non-temporisés pour permettre le model-checking et la simulation avec le framework BIP. Le sous-ensemble AADL que les auteurs ont sélectionné prend en charge les connexions périodiques/sporadiques de threads et les ports événements/données. Cependant, le mapping utilise un simple ordonnanceur non préemptif et le papier ne développe pas la transformation des actions de l'annexe comportementale.

Dans l'environnement TOPCASED [Berthomieu *et al.*, 2009], le modèle AADL est transformé en modèle intermédiaire *Fiacre*. Ensuite, ce modèle est compilé dans un système de transitions temporisées abstrait pris en charge par l'outil Tina pour la vérification de modèle. Une seconde version de ce travail est présentée dans [Bodeveix *et al.*, 2015]. Elle traite d'un sous-ensemble synchrone AADL visant à valider la transformation.

Comme la majorité des transformations formelles AADL, notre proposition vise à définir une sémantique exécutable formelle d'un sous-ensemble AADL pour permettre la simulation et le model-checking des propriétés comportementales et temporelles. Pourtant, notre approche se distingue par les points suivants : (1) le sous-ensemble AADL considéré se compose à la fois de composants AADL logiciels et matériels avec un ensemble significatif de propriétés AADL temporelles. De plus, (2) différentes sections comportementales sont prises en charge et projetées vers des constructions du langage LNT. Nous nous concentrons sur (3) l'ordonnancement préemptif des threads AADL et (4) la communication asynchrone. Enfin, (5) nous prenons en charge les threads périodiques ainsi que les threads sporadiques. Ce sous-ensemble couvre les fonctionnalités fondamentales en temps réel qui peuvent être utilisées dans des applications plus réalistes que dans des approches synchrones et non préemptives. De plus, nous utilisons LNT comme langage cible car il est une entrée directe idéale pour la boîte à outils CADP offrant une variété de méthodes pour la vérification formelle.

© 2022 Bechir Zalila

4.7 Conclusion et perspectives

Dans ce chapitre, nous avons décrit notre contribution à la vérification formelle dans le contexte des systèmes temps réel. Nous avons principalement présenté un modèle formel de tâches temps réel utilisant le langage LNT. Cette proposition a été appliquée et testée dans le cadre d'une approche MDE basée sur le langage de description d'architectures AADL. Nous avons présenté une transformation automatique des modèles AADL en spécifications LNT et nous l'avons implantée dans la suite d'outils OCARINA. Cette génération permet la vérification formelle du modèle AADL avec la boîte à outils CADP. Notre travail a été illustré avec divers systèmes temps réel.

Nous avons fourni une sémantique exécutable formelle considérant principalement l'ordonnancement et la communication qui sont indispensables pour une analyse utile des systèmes temps réel. Notre proposition apporte des résultats significatifs face aux défis de vérification formelle (temps d'analyse et explosion de l'espace d'états) ce qui est encourageant. A ce niveau, une étape fondamentale a été franchie.

Publications

Cette contribution a donné lieu à la publication de **deux** articles de revues et de **deux** articles dans des conférences internationales. La liste exhaustive de nos publications est donnée à la fin de ce mémoire.

Perspectives

Comme travaux futurs, nous envisageons d'étendre le mapping d'ordonnancement pour prendre en charge l'ordonnancement multicœurs. Une autre direction concerne l'aspect réseau et la prise en compte des composants AADL **`bus`** et **`virtual bus`**. Enfin, notre transformation de modèle doit être certifiée afin de montrer que le passage du modèle abstrait au modèle de vérification préserve la sémantique.

© 2022 Bechir ZALILA

Conclusion générale et perspectives

Sommaire

1 Rappel des contributions et des résultats 88
  1.1 Processus de reconfiguration dynamique pour les systèmes TR²E . . 88
  1.2 Processus de construction de systèmes TR²E tolérants aux pannes . 89
  1.3 Optimisation multi-objectifs des systèmes TR²E 89
  1.4 Vérification formelle des systèmes TR²E 89
2 Perspectives 90
  2.1 Cours terme 90
  2.2 Moyen terme 90
  2.3 Long terme 91

Dans les chapitres précédents, nous avons présenté les contributions de nos activités de recherche. Dans ce chapitre, nous rappelons ces contributions. Ensuite, nous présentons les futures pistes de recherche que nous comptons explorer.

1 Rappel des contributions et des résultats

Nos activités de recherche des dix dernières années s'inscrivent dans la thématique des langages de descriptions d'architectures (ADLs) dans un contexte TR²E.

1.1 Processus de reconfiguration dynamique pour les systèmes TR²E

Nous avons proposé dans le chapitre 1 une approche dirigée par les modèles permettant de construire des systèmes TR²E dynamiquement reconfigurables. Nous avons développé un framework de modélisation introduisant de nouveaux concepts (comme le concept de MétaMode) pour concevoir ces systèmes en utilisant des technologies de méta-modélisation et en suivant un processus d'ingénierie dirigée par les modèles. Ce processus permet de concevoir et de générer facilement une grande partie du code source du système TR²E. Ce processus a été validé par une étude de cas décrivant un système TR²E dynamiquement reconfigurable.

Cette contribution est le fruit d'un travail de recherche d'équipe. Il s'agit principalement des résultats des travaux de thèse de doctorat de Mme Fatma Krichen réalisée dans le cadre d'une convention de cotutelle entre l'Université de Sfax et l'Université de Toulouse, et des projets de mastères de recherches de Mmes Amal Gassara, Alvine Boaye Belle et

Amal GHORBEL. En outre, ces travaux ont donné lieu à des publications dans une revue et plusieurs conférences internationales [19].

1.2 Processus de construction de systèmes TR²E tolérants aux pannes

Dans le chapitre 2, nous avons présenté un processus de développement pour intégrer les préoccupations de tolérance aux pannes depuis l'étape de modélisation. Nous avons utilisé, pour cela, le langage AADL pour modéliser notre système car il donne aux utilisateurs, grâce à son mécanisme d'annexes, la possibilité de décrire des préoccupations fonctionnelles ainsi que des préoccupations transversales comme la tolérance aux pannes. Nous avons également eu recours à la programmation orientée aspect pour la mise en œuvre des exigences de tolérance aux pannes en étendant et en optimisant le langage d'aspect AspectAda par rapport aux exigences temps réel.

Cette contribution est le fruit d'un travail de recherche d'équipe. Il s'agit principalement des résultats des travaux de thèse de doctorat de Mme Wafa GABSI et des projets de mastères de recherches de Mmes Sihem LOUKIL, Rahma BOUAZIZ et Dorra KTARI. En outre, ces travaux ont donné lieu à des publications dans des revues, plusieurs conférences internationales et également sous la forme d'un chapitre de livre [19].

1.3 Optimisation multi-objectifs des systèmes TR²E

Dans le chapitre 3, nous nous sommes intéressés à l'optimisation de l'affectation des fonctions d'une application TR²E aux threads d'un système. Pour ce faire, nous avons proposé une méthode automatisée utilisant un algorithme évolutif multi-objectifs guidé par une technique de classification architecturale. Cette méthode permet aux concepteurs de chercher dans l'espace de conceptions des solutions ordonnançables de plusieurs critères de performance concurrents pour une affectation quasi-idéale des fonctions aux threads. Nous avons également élaboré un ensemble d'études empiriques qui ont permis de sélectionner un nombre minimal des fonctions-objectifs non redondantes afin d'accélérer la recherche de telles solutions architecturales.

Cette contribution est le fruit d'un travail de recherche d'équipe. Il s'agit principalement des résultats des travaux de thèse de doctorat de Mme Rahma BOUAZIZ réalisée dans le cadre d'une collaboration entre l'Université de Sfax et l'Université de Bretagne Occidentale. En outre, ces travaux ont donné lieu à des publications dans une revue et plusieurs conférences internationales [19].

1.4 Vérification formelle des systèmes TR²E

Dans le chapitre 4, nous avons proposé une approche générative pour la vérification formelle des systèmes TR²E. Cette approche est basée sur des règles de transformation d'un langage de description d'architectures (AADL) vers un modèle formel concurrent à base d'algèbres de processus (LNT). L'objectif de cette approche a été de profiter de l'existence d'outils de vérification formelle et d'intégrer leur utilisation dès le stade de modélisation d'un système TR²E. Cela a permis la détection précoce de problèmes plus profonds pouvant entraîner de graves erreurs dans le système final ou augmenter le coût de correction si découverts tard. Cette vérification est censée être automatique et transparente pour simplifier et encourager la pratique de méthodes formelles en génie logiciel.

19. La liste exaustive de nos publications est donnée en annexe de ce mémoire.

© 2022 Bechir ZALILA

Cette contribution est le fruit d'un travail de recherche d'équipe. Il s'agit principalement des résultats des travaux de thèse de doctorat de Mlle Hana MKAOUAR réalisée dans le cadre d'une collaboration entre l'Université de Sfax et l'Institut Supérieur de l'Aéronautique et de l'Espace de Toulouse. En outre, ces travaux ont donné lieu à des publications dans des revues et plusieurs conférences internationales [19].

2 Perspectives

L'expérience et le savoir-faire que nous avons acquis durant nos travaux de thèse et durant les dix dernières années de travaux de recherche dans le domaine de la conceptions de systèmes TR²E à l'aide des ADLs nous permettent de nous intéresser à de nouveaux horizons très prometteurs en utilisant les contributions décrites précédemment dans ce domaine. Dans cette section, nous présentons des perspectives à cours terme et à moyen terme pour nos activités de recherche. Nous donnons également un aperçu sur la directions dans laquelle nous comptons orienter nos activités de recherche sur le long terme.

2.1 Cours terme

Sur le cours terme, nous envisageons de nous intéresser à la perspective suivante liée aux systèmes temps réel.

Optimisation des protocoles de partage de ressources en temps réel

Jusque là, nous avons été de purs consommateurs pour les profils d'exécution pour les systèmes temps réel (comme le profil Ravenscar) et particulièrement pour le protocole de partage de ressources qu'il utilise (*Priority Ceiling Protocol*, PCP). Le protocole PCP garantit certes l'absence d'interblocages dans les systèmes qui le respectent, mais il est réputé pour son *pessimisme* dans le sens où certains threads se trouvent bloqués alors qu'ils pourraient ne pas l'être. Nous envisageons de proposer une optimisation à ce protocole afin de réduire les temps de blocage inutiles de threads tout en conservant les mêmes garanties du protocole PCP. Il s'agira de raffiner la condition qui plafonne les priorités des threads partageant une ressource afin de n'augmenter la priorité que pour les threads susceptibles de causer un interblocage s'ils se trouvent suspendus par l'ordonnanceur.

2.2 Moyen terme

Sur le moyen terme, nous envisageons de nous intéresser aux perspectives suivantes, pouvant constituer chacune un sujet de thèse de doctorat à part entière.

Configuration et reconfiguration automatiques de systèmes TR²E à micro-contrôleurs

L'avènement récent de l'Internet des Objets (*Internet of Things*, IoT) et l'apparition d'une grande variété de cartes programmables basées sur des architectures matérielles performantes comme **ATmega** (présente surtout sur les cartes **Arduino**), **STM32**, **ESP8266**, **ESP32** ou encore **RISC-V** a, certes, simplifié la conception de systèmes embarqués et a accéléré leur prototypage, surtout grâce à des IDE [20] qui masquent la complexité du développement pour ces architectures. Cependant, le code produit en utilisant ces environnements de développement n'est pas optimisé pour certains besoins critiques (consomma-

20. IDE : Integrated Development Environment

© 2022 Bechir ZALILA

tion d'énergie, vitesse CPU, utilisation mémoire). Nous proposons d'utiliser les langages de description d'architectures pour permettre d'intégrer des besoins non fonctionnels et de produire du code qui soit plus optimisé aux besoins d'une architecture matérielle donnée.

Génération automatique d'images Linux-embarqué optimisées

Aujourd'hui, la grande majorité des systèmes embarqués avec un besoin massif en concurrence et des communications à travers le réseau utilisent le système d'exploitation libre Linux. Contrairement aux micro-ordinateurs pour lesquels il existe des distributions Linux toutes prêtes à être utilisées, les architectures embarquées, du fait de leur très grande variété et hétérogénéité, nécessitent la reconstruction complète à partir du code source d'un système qui leur soit dédié (après une phase de configuration pas toujours facile). Pour simplifier cette tâche de configuration et de production d'une image d'un système Linux-embarqué, beaucoup d'outils existent et facilitent considérablement cette tâche. Le plus utilisé aujourd'hui est le projet libre **Yocto**. Cependant, utiliser **Yocto** nécessite une station de travail de développement avec des performances extrêmement hautes (en terme de mémoire, de vitesse de microprocesseur et d'espace de stockage). Ceci empêche aujourd'hui l'utilisation de cet outil dans l'enseignement par exemple à cause des coûts exorbitants des équipements nécessaires. Nous proposons d'explorer l'utilisation d'une autre approche basée sur les langages de description d'architectures, de profiter de leur pouvoir d'expression afin de réduire les performances requises pour construire une image Linux-embarqué pour une plateforme embarquée donnée (**ARM**, **RISC-V**, **x86**, etc.).

2.3 Long terme

Sur le long terme, nous envisageons de rassembler les processus des quatre contributions décrites dans ce mémoire au sein d'une même plateforme pour assister (particulièrement lors de la phase d'affectation des fonctions aux threads) les concepteurs à décrire et à optimiser des systèmes TR^{2}E dynamiquement reconfigurables tolérants au pannes, ensuite de les vérifier formellement dès la phase de modélisation. Cette plateforme, notamment avec les contributions à moyen terme, pourra être utilisée dans des domaines critiques comme celui des véhicules autonomes ou encore celui de la santé afin d'accélérer et de réduire le coût du prototypage des systèmes TR^{2}E. Elle pourra, également être utilisée dans des domaines moins critiques comme celui de l'agriculture intelligente afin de permettre une gestion automatisée des parcelles agricoles tout en optimisant l'utilisation d'eau. Un pays souffrant d'un stress hydrique chronique comme la Tunisie pourra profiter considérablement de ces résultats.

Les architectures aussi bien logicielles que matérielles ont considérablement simplifié la modélisation des systèmes TR^{2}E et, par l'intermédiaire d'outils dédiés, ont permis d'effectuer diverses analyses et générations automatiques de code pour ces systèmes. Nos contributions décrites dans ce mémoire sont venues ajouter quelques pierres à cet édifice.

© 2022 Bechir Zalila

Bibliographie

[Afonso *et al.*, 2008] Francisco Afonso, Carlos Silva, Nuno Brito, Sergio Montenegro, and Adriano Tavares. Aspect-oriented fault tolerance for real-time embedded systems. In *ACM Proceedings of the AOSD workshop on Aspects, components, and patterns for infrastructure software*, New York, USA, 2008. 41

[Alexandersson and Karlsson, 2011] Ruben Alexandersson and Johan Karlsson. Fault injection-based assessment of aspect-oriented, implementation of fault tolerance. *International Conference on Dependable Systems and Networks*, pages 303–314, 2011. 41

[Audsley *et al.*, 1993] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5) :284–292, September 1993. 52

[Avizienis *et al.*, 2004] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33, 2004. 27, 29, 40

[Baeten, 2005] Jos C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3) :131–146, 2005. 69

[Bandyopadhyay and Saha, 2013] Sanghamitra Bandyopadhyay and Sriparna Saha. Some single- and multiobjective optimization techniques. In *Unsupervised Classification*, pages 17–58. Springer Berlin Heidelberg, 2013. 48

[Barnes, 2003] John Barnes. *High Integrity Software : The SPARK Approach to Safety and Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. 33

[Bartolini *et al.*, 2005] C. Bartolini, G. Lipari, and M. Di Natale. From functional blocks to the synthesis of the architectural model in embedded real-time applications. In *Proceedings of the* 11th *IEEE Real Time and Embedded Technology and Applications Symposium*, pages 458–467, March 2005. 61

[Belle, 2011] Alvine Boaye Belle. Conception et Développement d'un Support d'Exécution pour Systèmes TR^{2}E Dynamiquement Reconfigurables. Master's thesis, Université de Sfax (ENIS), aug 2011. 8

[Berthomieu *et al.*, 2009] Bernard Berthomieu, Jean-Paul Bodeveix, Christelle Chaudet, Silvano Dal Zilio, Mamoun Filali, and François Vernadat. Formal verification of AADL specifications in the Topcased environment. In *International Conference on Reliable Software Technologies*, pages 207–221. Springer, 2009. 86

[Bodeveix *et al.*, 2015] Jean-Paul Bodeveix, Mamoun Filali, Manuel Garnacho, Régis Spadotti, and Zhibin Yang. Towards a verified transformation from AADL to the formal component-based language FIACRE. *Science of Computer Programming*, 106 :30–53, 2015. 86

[Bouaziz *et al.*, 2016] Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, and Mohamed Jmaiel. Efficient parallel multi-objective optimization for real-time systems software design exploration. In *Proceedings of the* 27th *International Symposium on Rapid System Prototyping*, October 2016. 58

[Bouaziz *et al.*, 2018] Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, and Mohamed Jmaiel. Multi-objective design exploration approach for Ravenscar real-time systems. *Real-Time Systems*, 54(2) :424–483, 2018. 49, 50, 54

[Bouaziz, 2012] Rahma Bouaziz. Extension et Adaptation d'un Langage d'Aspects pour les Systèmes Temps-Réel. Master's thesis, Université de Sfax (ENIS), jul 2012. 27

[Bouaziz, 2018] Rahma Bouaziz. *Multi-objective Optimization and Design Space Exploration of Critical Real-Time Systems*. PhD thesis, Université de Sfax (ENIS), jul 2018. 45, 49, 50, 52, 53, 54, 55, 56, 57

[Bozzano *et al.*, 2010] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5) :754–775, 2010. 86

[Burns *et al.*, 2004] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar Profile in high integrity systems. *Ada Lett.*, XXIV :1–74, June 2004. 14, 16, 33, 47, 67, 69

[Buttazzo, 2011] Giorgio Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011. 51

[Champelovier *et al.*, 2018] David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Frédéric Lang, Christine McKinty, Vincent Powazny, Wendelin Serwe, and Gideon Smeding. Reference Manual of the LNT to LOTOS Translator. 2018. 67

[Chehade *et al.*, 2011] W. El Hajj Chehade, Ansgar Radermacher, François Terrier, Bran Selic, and Sébastien Gérard. A model-driven framework for the development of portable real-time embedded systems. In *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 45–54, Las Vegas, Nevada, USA, 2011. IEEE Computer Society. 23

[Chen and Kulkarni, 2011] Jingshu Chen and Sandeep Kulkarni. Effectiveness of transition systems to model faults. In *ACM/IEEE the 5th International Conference on Distributed Smart Cameras*, Gent, Belgium, 2011. 40

[Chkouri *et al.*, 2009] M Yassin Chkouri, Anne Robert, Marius Bozga, and Joseph Sifakis. Translating AADL into BIP-application to the verification of real-time systems. In *Models in Software Engineering*, pages 5–19. Springer, 2009. 86

[Coello Coello *et al.*, 2007] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007. 48

[Courtiat *et al.*, 2000] J-P Courtiat, Celso AS Santos, Christophe Lohr, and B Outtaj. Experience with rt-lotos, a temporal extension of the lotos formal description technique. *Computer Communications*, 23(12) :1104–1123, 2000. 70

[Deb *et al.*, 2002] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm : NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2) :182–197, 2002. 62

© 2022 Bechir Zalila

[Deb, 2001] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*, volume 16. John Wiley & Sons, 2001. 46

[do Nascimento *et al.*, 2007] Francisco Assis M. do Nascimento, Marcio F. S. Oliveira, and Flavio Rech Wagner. ModES : Embedded Systems Design Methodology and Tools based on MDE. In *Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 67–76, Washington, DC, USA, 2007. IEEE Computer Society. 24

[ESA, 2008] ESA. Spacewire - links, nodes, routers and networks. Technical report, July 2008. Technical report. 16

[Fondement and Silaghi, 2004] Frédéric Fondement and Raul Silaghi. Defining model driven engineering processes. In *Proceedings of the 7th International Conference on the Unified Modeling Language (UML)*, Lisbon, Portugal, 2004. 1

[Fritsch and Clarke, 2008] S. Fritsch and S. Clarke. TimeAdapt : timely execution of dynamic software reconfigurations. In *Proceedings of the 5th Middleware doctoral symposium*, pages 13–18, New York, NY, USA, 2008. ACM. 24

[Fürst *et al.*, 2009] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. Autosar–a worldwide standard is on the road. In *the* 14th *International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, 2009. 61

[Gabsi *et al.*, 2013] Wafa Gabsi, Rahma Bouaziz, and Bechir Zalila. Towards an aspect oriented language compliant with real time constraints. In *WETICE - AROSA*, pages 68–73, Hammamet, Tunisia, 2013. IEEE Computer Society. 33

[Gabsi *et al.*, 2016] Wafa Gabsi, Bechir Zalila, and Mohamed Jmaiel. EMA2AOP : from the AADL error model annex to aspect language towards fault tolerant systems. In *14th IEEE International Conference on Software Engineering Research, Management and Applications, SERA 2016, Towson, MD, USA, June 8-10, 2016*, pages 155–162, 2016. 37

[Gabsi *et al.*, 2020] Wafa Gabsi, Bechir Zalila, and Mohamed Jmaiel. Extension and Adaptation of an Aspect Oriented Programming Language for Real-time Systems. *International Journal of Business and Systems Research*, 14(2) :139–161, march 2020. 35

[Gabsi, 2017] Wafa Gabsi. *Tolérance aux pannes pour les systèmes temps-réel distribuées : de la modélisation à l'implantation*. PhD thesis, Université de Sfax (ENIS), avr 2017. 27, 31, 32, 34, 36, 37, 38, 39

[Gal and Hanne, 1999] Tomas Gal and Thomas Hanne. Consequences of dropping nonessential objectives for the application of mcdm methods. *European Journal of Operational Research*, 119(2) :373–378, 1999. 52

[Garavel and Hautbois, 1993] Hubert Garavel and Rene-Pierre Hautbois. An experiment with the LOTOS formal description technique on the flight warning computer of airbus 330/340 aircrafts. In *Proc. of the first AMAST International Workshop on Real-Time Systems*. Citeseer, 1993. 68

[Garavel and Lang, 2002] Hubert Garavel and Frédéric Lang. SVL : a scripting language for compositional verification. In *Formal Techniques for Networked and Distributed Systems*, pages 377–392. Springer, 2002. 67, 82

[Garavel *et al.*, 2013] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011 : a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2) :89–107, 2013. 67, 81

[Garavel *et al.*, 2017] Hubert Garavel, Frédéric Lang, and Wendelin Serwe. From LOTOS to LNT. In *ModelEd, TestEd, TrustEd*, pages 3–26. Springer, 2017. 67, 68

[Gassara, 2011] Amal Gassara. Vérification Formelle des Propriétés non Fonctionnelles des RTES Dynamiquement Reconfigurables. Master's thesis, Université de Sfax (ENIS), jul 2011. 8

[Ghorbel, 2012] Amal Ghorbel. Génération de Code pour les Systèmes Embarqués Temps-Réel Dynamiquement Reconfigurables selon l'Approche MDA. Master's thesis, Université de Sfax (ENIS), sep 2012. 8

[Gilles *et al.*, 2010] Lasnier Gilles, Robert Thomas, Pautet Laurent, and Kordon Fabrice. Behavioral Modular Description of Fault Tolerant Distributed Systems with AADL Behavioral Annex. In *10th Annual International Conference on New Technologies of Distributed Systems (NOTERE), 2010*, pages 17 –24, june 2010. 40

[Guo *et al.*, 2008] Yu Guo, Krzysztof Sierszecki, and Christo Angelov. A (Re)Configuration Mechanism for Resource-Constrained Embedded Systems. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1315–1320, Washington, DC, USA, 2008. IEEE Computer Society. 24

[Hameed *et al.*, 2009] Kashif Hameed, Rob Williams, and Jim Smith. Aspect oriented software fault tolerance. In *Proceedings of the World Congress on Engineering WCE London, U.K.*, 2009. 41

[Hamid *et al.*, 2008a] Brahim Hamid, Ansgar Radermacher, Agnes Lanusse, Christophe Jouvray, Sébastien Gérard, and François Terrier. Designing fault-tolerant component based applications with a model driven approach. In *SEUS*, 2008. 40

[Hamid *et al.*, 2008b] Brahim Hamid, Ansgar Radermacher, Patrick Vanuxeem, Agnes Lanusse, and Sebastien Gerard. A fault-tolerance framework for distributed component systems. In *EUROMICRO-SEAA*, 2008. 40

[Hecht *et al.*, 2011] Myron Hecht, Alexander Lam, and Chris Vogl. A Tool Set for Integrated Software and Hardware Dependability Analysis Using the Architecture Analysis and Design Language (AADL) and Error Model Annex. In *ICECCS*, pages 361–366, 2011. 32

[Hruschka *et al.*, 2009] Eduardo Raul Hruschka, Ricardo J. G. B. Campello, Alex A. Freitas, and André C. Ponce Leon F. De Carvalho. A survey of evolutionary algorithms for clustering. *Trans. Sys. Man Cyber Part C*, 39(2) :133–155, 2009. 48

[Hugues *et al.*, 2008] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. From the Prototype to the Final Embedded System Using the Ocarina AADL Tool Suite. *ACM Transactions in Embedded Computing Systems (TECS)*, 7(4) :1–25, jul 2008. 23

[Jouault *et al.*, 2006] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. ATL : a QVT-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM, 2006. 18

[Kiczales *et al.*, 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997. 33

© 2022 Bechir Zalila

[Knowles and Corne, 2000] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2) :149–172, 2000. 47, 49

[Kordon *et al.*, 2013] Fabrice Kordon, Jérôme Hugues, Agusti Canals, and Alain Dohet. *Embedded Systems : Analysis and Modeling with SysML, UML and AADL*. Wiley-ISTE, UK, apr 2013. 2

[Koziolek *et al.*, 2011] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. Peropteryx : automated application of tactics in multi-objective software architecture optimization. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 33–42. ACM, 2011. 62

[Krichen *et al.*, 2011] Fatma Krichen, Brahim Hamid, Bechir Zalila, and Mohamed Jmaiel. Towards a model-based approach for reconfigurable distributed real time embedded systems. In *Proceedings of the 5th European Conference on Software Architecture*, pages 295–302, Essen, Germany, September 2011. Springer. 11, 12, 14, 20

[Krichen *et al.*, 2012a] Fatma Krichen, Amal Ghorbel, Bechir Zalila, and Brahim Hamid. An MDE-based approach for reconfigurable DRE systems. In *Proceedings of the 21st IEEE International Conference on Collaboration Technologies and Infrastructures*, pages 78–83, Toulouse, France, juin 2012. IEEE Computer Society. 11

[Krichen *et al.*, 2012b] Fatma Krichen, Bechir Zalila, Mohamed Jmaiel, and Brahim Hamid. A Middleware for Reconfigurable Distributed Real-time Embedded Systems. In *Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications SERA (selected papers)*, Studies in Computational Intelligence, pages 81–96, Shanghai, China, 2012. Springer. 10, 11

[Krichen, 2013] Fatma Krichen. *Architectures logicielles à composants reconfigurables pour les systèmes Temps Réel Répartis Embarqués (TR^2E).* PhD thesis, Université de Toulouse et Université de Sfax (ENIS), sep 2013. 8, 10, 11, 12, 13, 18, 19, 21, 22

[Ktari, 2014] Dorra Ktari. Définition et Implantation de Règles de Génération de Code d'un Langage de Modélisation d'Erreurs vers un Langage de Programmation. Master's thesis, Université de Sfax (FSEGS), nov 2014. 27

[Lamport *et al.*, 1982] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3) :382–401, 1982. 30

[Larson *et al.*, 2013] Brian Larson, John Hatcliff, Kim Fowler, and Julien Delange. Illustrating the AADL Error Modeling Annex (V.2) Using a Simple Safety-critical Medical Device. In *Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '13, pages 65–84, New York, NY, USA, 2013. ACM. 37

[Lasnier *et al.*, 2009a] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérôme Hugues. Ocarina : An environment for aadl models analysis and automatic code generation for high integrity applications. In *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*, pages 237–250, Brest, France, 2009. Springer. 23

[Lasnier *et al.*, 2009b] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérome Hugues. Ocarina : An environment for AADL models analysis and automatic code generation for

© 2022 Bechir Zalila

high integrity applications. In *Reliable Software Technologies–Ada-Europe 2009*, pages 237–250. Springer, 2009. 67, 81

[Léonard and Leduc, 1998] Luc Léonard and Guy Leduc. A formal definition of time in lotos. *Formal Aspects of Computing*, 10(3) :248–266, 1998. 70

[Li *et al.*, 2011] Rui Li, Ramin Etemaadi, Michael TM Emmerich, and Michel RV Chaudron. An evolutionary multiobjective optimization approach to component-based software architecture design. In *IEEE Congress on Evolutionary Computation*, pages 432–439. IEEE, 2011. 62

[Liu and Layland, 1973] Chang L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1) :46–61, 1973. 47, 57, 67, 69, 73

[LOTOS, 1989] ISO LOTOS. ISO/IEC. LOTOS : A formal description technique based on the temporal ordering of observational behaviour. International Organization for Standardization Information Processing Systems Open Systems Interconnection, 1989. 67

[Loukil *et al.*, 2013] Sihem Loukil, Slim Kallel, Bechir Zalila, and Mohamed Jmaiel. AO4AADL : aspect oriented extension for AADL. *Central Europ. J. Computer Science*, 3(2) :43–68, 2013. 33, 35

[Loukil, 2010] Sihem Loukil. Extension d'un Langage de Description d'Architecture pour la Programmation Orientée Aspect. Master's thesis, Université de Sfax (ENIS), jul 2010. 27

[Malavolta *et al.*, 2013] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Anthony Tang. What industry needs from architectural languages : A survey. *Software Engineering, IEEE Transactions on*, 39(6) :869–891, 2013. 67

[Medvidovic and Taylor, 2000] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Software Engineering*, 26(1) :70–93, 2000. 1, 31

[Mehiaoui *et al.*, 2012] Asma Mehiaoui, Sara Tucci-Piergiovanni, J Babau, and Laurent Lemarchand. Optimizing the deployment of distributed real-time embedded applications. In *Proceedings of the* 18th *International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 400–403. IEEE, 2012. 62

[Mkaouar *et al.*, 2018] Hana Mkaouar, Bechir Zalila, Jérôme Hugues, and Mohamed Jmaiel. An ocarina extension for AADL formal semantics generation. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1402–1409. ACM, 2018. 81

[Mkaouar *et al.*, 2020a] Hana Mkaouar, Bechir Zalila, Jérôme Hugues, and Mohamed Jmaiel. A formal approach to AADL model-based software engineering. *International Journal on Software Tools for Technology Transfer*, 22(2) :219–247, april 2020. 69, 73, 80, 84

[Mkaouar *et al.*, 2020b] Hana Mkaouar, Bechir Zalila, Jérôme Hugues, and Mohamed Jmaiel. Towards a Formal Specification for an AADL Behavioural Subset Using the LNT Language. *International Journal of Business and Systems Research*, 14(2) :162–190, march 2020. 84

[Mkaouar, 2019] Hana Mkaouar. *A Formal Approach for Real-time Systems Engineering*. PhD thesis, Université de Sfax (ENIS), feb 2019. 66, 68, 71, 74, 75, 79, 80, 81, 82

[Monot *et al.*, 2012] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multisource software on multicore automotive ecus—combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10) :3934–3942, 2012. 62

© 2022 Bechir Zalila

[Mraidha *et al.*, 2011] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum : a MARTE-based Methodology for Schedulability Analysis at Early Design Stages. *ACM SIGSOFT Software Engineering Notes*, 36(1) :1–8, 2011. 61

[OMG, 2005] OMG. UML Profile for Schedulability, Performance, and Time Specification. http ://www.omg.org/technology/documents/formal/schedulability.htm, January 2005. 23

[OMG, 2006] OMG. Deployment and Configuration of Component-based Distributed Applications Specification, v4.0. http ://www.omg.org/cgi-bin/doc ?formal/06-04-02, April 2006. 8

[OMG, 2019] OMG. UML Profile for MARTE : Modeling and Analysis of Real-Time Embedded Systems. https ://www.omg.org/spec/MARTE/, April 2019. 9, 11, 13, 23

[Pagetti *et al.*, 2011] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3) :307–338, 2011. 61

[Pautet, 2002] Laurent Pautet. *Intergiciels schizophrènes : une solution à l'interopérabilité entre modèles de répartition*. Habilitation à diriger des recherches, Université Pierre et Marie Curie – Paris VI, jan 2002. 14

[Pedersen and Constantinides, 2005] Knut H. Pedersen and Constantinos Constantinides. Aspectada : aspect oriented programming for ada95. *Ada Letters*, XXV(4) :79–92, 2005. 33

[Rahmoun *et al.*, 2015a] S. Rahmoun, E. Borde, and L. Pautet. Multi-objectives refinement of aadl models for the synthesis embedded systems (mu-ramses). In *Proceedings of the 20th International Conference on Engineering of Complex Computer Systems*, pages 21–30, December 2015. 62

[Rahmoun *et al.*, 2015b] Smail Rahmoun, Etienne Borde, and Laurent Pautet. Automatic selection and composition of model transformations alternatives using evolutionary algorithms. In *Proceedings of the 9th European Conference on Software Architecture Workshops*, page 25. ACM, 2015. 62

[Rota, 1964] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5) :498–504, may 1964. 45, 60

[RTCA, 2011] RTCA. RTCA/DO-333 : formal methods supplement to DO-178C and DO-278A. 2011. 66

[Rugina *et al.*, 2008] Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. The ADAPT Tool : From AADL Architectural Models to Stochastic Petri Nets through Model Transformation. *CoRR*, abs/0809.4108, 2008. 86

[Rugina, 2007] Ana Elena Rugina. *Dependability modeling and evaluation From AADL to stochastic Petri netss*. PhD thesis, 2007. 41

[SAE, 2015] SAE. *Architecture Analysis & Design Language (AADL) Annex E : Error Model Annex*, September 2015. Available at `https://www.sae.org/standards/content/as5506/1a/`. 28, 32, 35, 86

[SAE, 2017a] SAE. *Architecture Analysis & Design Language*, Jan 2017. Available at `https://www.sae.org/standards/content/as5506c/`. 9, 23, 31, 58, 67, 80, 86

[SAE, 2017b] SAE. *Architecture Analysis & Design Language (AADL) Annex D : Behavior Model Annex*, august 2017. Available at `https://www.sae.org/standards/content/as5506/3/`. 28, 68, 86

[Scheickl and Rudorfer, 2008] Oliver Scheickl and Michael Rudorfer. Automotive real-time development using a timing-augmented autosar specification. *Proceedings of the* 4th *International Congress on Embedded Real-Time Systems*, 2008. 62

[Sha *et al.*, 1990] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9) :1175–1185, 1990. 16, 47

[Singhoff *et al.*, 2004] F. Singhoff, J. Legrand, L. Nana Tchamnda, and L. Marcé. Cheddar : a Flexible Real Time Scheduling Framework. *ACM Ada Letters journal, 24(4) :1-8, ACM Press. Also published in the proceedings of the ACM SIGADA International Conference, Atlanta, 15-18 November, 2004*, Nov 2004. 47, 57

[Szentivanyi and Nadjm-Tehrani, 2004] Diana Szentivanyi and Simin Nadjm-Tehrani. Aspects for improvement of performance in fault-tolerant software. *Pacific Rim International Symposium on Dependable Computing*, pages 283–291, 2004. 42

[Vergnaud *et al.*, 2006] Thomas Vergnaud, Bechir Zalila, and Jérôme Hugues. Ocarina : a Compiler for the AADL. Technical report, Telecom Paristech - France, 2006. 33

[Vinoski, 2002] Steve Vinoski. Middleware "Dark Matter". *IEEE Internet Computing*, 6(5) :92–95, 2002. 1

[Vyatkin, 2013] Valeriy Vyatkin. Software engineering in industrial automation : State-of-the-art review. *Industrial Informatics, IEEE Transactions on*, 9(3) :1234–1249, 2013. 66

[Whittle *et al.*, 2014] Jon Whittle, John Hutchinson, and Mark Rouncefield. The State of Practice in Model-Driven Engineering. *IEEE Software*, 31(3) :79–85, 2014. 9

[Wozniak *et al.*, 2013] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and Sébastien Gerard. An optimization approach for the synthesis of autosar architectures. In *The* 18th *IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–10. IEEE, 2013. 62

[Zalila *et al.*, 2008] B. Zalila, L. Pautet, and J. Hugues. Towards Automatic Middleware Generation. In *11th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'08)*, pages 221–228, Orlando, Florida, USA, may 2008. 14

[Zalila, 2005] Bechir Zalila. *Optimisation, Déterminisme et Asynchronisme de Souches et Squelettes CORBA pour Systèmes Répartis Temps-réel*. Master's thesis, Université Pierre & Marie Curie, Paris VI, sep 2005. 3

[Zalila, 2008] Bechir Zalila. *Configuration et déploiement d'applications temps-réel réparties embarquées à l'aide d'un langage de description d'architecture*. PhD thesis, École Nationale Supérieure des Télécommunications, nov 2008. 2, 14, 23, 67

© 2022 Bechir Zalila

© 2022 Bechir Zalila