Sélection automatique de tests de faisabilité à l'aide de patrons de conception

Vincent Gaudel, Frank Singhoff, Alain Plantec
Université de Bretagne Occidentale/LISyC
Université Européenne de Bretagne
20, av Le Gorgeu 29238 BREST Cedex 3, France
Email: {Vincent.Gaudel, singhoff, plantec}@univ-brest.fr

Pierre Dissaux, Jérôme Legrand
Ellidiss Technologies
24, quai de la douane 29200 BREST, France
Email: {pierre.dissaux, Jerome.Legrand}@ellidiss.com

***Résumé*—Cet article traite de la vérification de modèles d'architectures de systèmes embarqués temps réel par la théorie de l'ordonnancement temps réel. Il présente une approche s'articulant autours de la définition de patrons de conception spécifiques aux architectures temps réel et permettant une vérification automatisée d'ordonnançabilité. La vérification se base sur des tests de faisabilité existants, mais dont la sélection est déduite de la conformité du système à un patron de conception et de ses propriétés. Ces contrôles de conformité sont intégrés dans Cheddar, un environnement de vérification de modèles d'architectures AADL (*Architecture Analysis and Design Langage*). Nous exposons, ici, la méthode de conception d'architectures temps réel du point de vue du concepteur souhaitant employer cette approche.**

I. Introduction

Un système temps réel est dit critique si son dysfonctionnement peut avoir pour conséquence d'entraîner des dégâts sur des personnes ou pour l'environnement. La validation d'un tel système est donc très importante. La théorie de l'ordonnancement fournit des outils pour la validation des architectures temps réel avec notamment des méthodes analytiques appelées *tests de faisabilité*. Cependant, leur utilisation implique une expertise approfondie : en effet, les tests de faisabilité ne sont utilisables que dans certaines conditions bien précises. Suivant le critère de performance évalué et suivant les caractéristiques de l'architecture, un tel test peut ne pas être applicable. Ainsi, déterminer quels tests de faisabilité sont applicables est une tâche complexe et couteuse. En définitive, on observe que la théorie de l'ordonnancement est peu utilisée.

Nous proposons une caractérisation des architectures par le biais de patrons de conception qui permettent d'automatiser la sélection des tests de faisabilité. Nous avons défini cinq patrons basés sur les protocoles de communication et de synchronisation entre les tâches. Les patrons de conception sont modélisés par des ensembles de conditions sur les propriétés des modèles d'architectures. Nous analysons les modèles d'architecture pour vérifier le respect de ces conditions : on vérifie que les modèles sont conformes aux patrons. Dans le cas où la conformité à un patron de conception est confirmée, nous sommes capables de déterminer une liste de tests de faisabilité applicables. Cette liste est dépendante du patron de conception utilisé ainsi que de contraintes sur les propriétés de l'architecture.

Nous avons réalisé un prototype capable de sélectionner les tests de faisabilité applicables à un modèle d'architecture AADL conforme à un patron de conception. Ce prototype a été conçu comme une extension de Cheddar, logiciel permettant de concevoir de telles architectures et de leur appliquer les différents tests de faisabilité sélectionnés. Cet article présente notre approche au travers du point de vue du concepteur souhaitant vérifier un modèle d'architecture. Nous montrons comment modéliser les patrons de conception et leur mise en correspondance avec un ensemble de tests de faisabilité. Nous exposons la méthode d'utilisation de notre méthode du point de vue concepteur en partie II, puis la façon dont nous modélisons les patrons de conception en partie III. Enfin, nous étudions les travaux connexes en partie IV, avant de conclure partie V.

II. Processus

A. Le problème de la vérification des systèmes critiques

La théorie de l'ordonnancement temps réel permet au concepteur d'un système d'analyser le comportement temporel d'un ensemble de tâches avec des méthodes algébriques appelées tests de faisabilités. Par exemple, Liu et Layland [1] définissent des tâches réalisant périodiquement un traitement par trois paramètres : le délai critique (D_i), la période d'activation (P_i), la capacité (C_i). à chaque fois qu'un tâche i est activée, elle doit réaliser un travail dont le temps d'exécution est borné par C_i. Ce travail doit être complété avant D_i unités de temps après l'instant d'activation de la tâche. De ces paramètres ont été conçu différents types de tests de faisabilité : des tests basés sur le facteur d'utilisation processeur, sur le temps de réponse pire cas, etc [1]. Par exemple, Liu *et al.* proposent un moyen de calculer le facteur d'utilisation processeur par :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

La somme des rapports de la capacité des tâches sur leur période doit être inférieure ou égale à 1. Dans le cas où cette somme est supérieure à 1, la charge de calcul est trop importante vis-à-vis de la capacité du processeur. Le système ne peut donc pas être ordonnancé de façon à respecter les

contraintes temporelles des tâches. Cependant, ce test n'est applicable que dans un contexte bien précis : les tâches doivent être périodiques et indépendantes et le protocole d'ordonnancement doit être *Earliest Deadline First* préemptif.

L'application d'un test de faisabilité à un système requiert donc que ce dernier possède des propriétés architecturales données. Nous nommerons ces contraintes *"contraintes d'applicabilité"* dans la suite de l'article. Elles caractérisent les propriétés des entités inhérentes aux systèmes temps réel comme la périodicité des tâches, le protocole d'ordonnancement utilisé, le protocole de communication, etc.

B. Définition des patrons de conception

Comme nous le précisions précédemment, la classification des différents systèmes analysables est réalisée par le biais de cinq patrons de conception : *Synchronous data-flow*, *Ravenscar*, *Blackboard*, *Queued buffer* et *Unplugged*. Chaque patron de conception propose une solution architecturale à un problème de synchronisation entre les tâches en définissant un protocole de communication inter-tâches [2], [3]. *Synchronous data-flow* est le patron de conception le plus simple. La communication entre les tâches s'effectue par zones de mémoire. Les tâches lisent ces zones à leur réveil, et écrivent des données sur ces mêmes zones lors de leur terminaison. Les réveils et terminaisons sont disjoints, il n'y a donc pas de besoin de synchronisation. Pour *Ravenscar*, les données sont partagées de façon asynchrone, sous le contrôle du protocole d'héritage de priorités. Ravenscar permet aux tâches de partager des données protégées par des sémaphores. Ces dernières peuvent être utilisées pour construire différents protocoles de synchronisation tels que les sections critiques, lecteurs-écrivains, producteurs-consommateurs, etc. *Blackboard* implante le patron de conception lecteurs-écrivains : seule la dernière donnée produite est accessible aux tâches. *Queued buffer* implante le patron de conception producteurs-consommateurs. L'accès aux messages est géré par un protocole FIFO. Le dernier patron, *Unplugged*, modélise l'absence de communication entre les tâches. Il existe bien sûr de nombreux autres paradigmes possibles de synchronisation qui puissent justifier une telle spécification. Dans un premier temps, nous estimons que ces cinq patrons sont suffisants pour valider notre approche [4].

C. La méthode du point de vue concepteur

La figure 1 décrit le cycle de conception de modèles d'architectures temps réel incluant l'outil de sélection de tests. Le concepteur propose un modèle d'architecture en AADL (1). L'outil de sélection analyse le modèle d'architecture et vérifie sa conformité à un patron conception (2) (les cinq patrons sont vérifiés un à un). L'outil propose alors une liste de tests de faisabilité spécifique au système (4), avant de les exécuter (6). Lors de chacune des étapes 2, 4 et 6, le concepteur est susceptible de revenir sur la phase de conception. Dans un premier temps lors de la vérification de la conformité du modèle d'architecture à un patron : si le système n'est pas conforme à un patron, l'outil n'est donc pas en mesure de fournir une liste de tests de faisabilité, et propose à la place un ensemble de métriques pour assister le concepteur (3). Dans un second temps, lors de l'étape (4) : si la liste de tests de faisabilité proposée ne le satisfait pas (besoin d'appliquer un test en particulier par exemple). L'outil propose alors un ensemble de tests faisabilité différents et leurs contraintes d'applicabilité non-respectées (5). Enfin, après l'analyse d'ordonnançabilité : dans le cas où le système n'est pas ordonnançable ou si les tests faisabilité ne sont pas en mesure de valider l'ordonnancement (7).

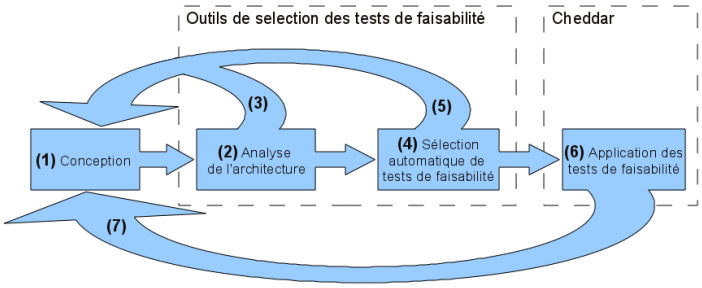

Fig. 1. Schéma des étapes de la méthode du point de vue concepteur.

Comme nous venons de l'exposer, outre le choix des tests de faisabilité à appliquer, il est utile de raisonner sur le modèle d'architecture afin de pouvoir fournir au concepteur des informations sur des modifications potentielles permettant une meilleure vérification de son modèle d'architecture. Un outil implantant ces contrôles de conformité à un patron de conception doit donc répondre à plusieurs questions : le modèle d'architecture est-il conforme à un patron de conception ? Si non, quelle est l'importance des modifications à apporter au modèle d'architecture pour l'être ? Si oui, quelle est la liste des tests de faisabilité applicables ? Quelles sont les autres listes de tests de faisabilité non-retenues et les modifications à apporter au modèle d'architecture pour pouvoir les appliquer ? Les tests de faisabilité sélectionnés suffisent-ils à prouver l'ordonnançabilité ? Le modèle du système est-il ordonnançable, etc.

D. Générer le code du prototype

Afin d'évaluer notre approche, nous avons réalisé un prototype intégré à Cheddar [4], un environnement d'analyse de performances d'applications concurrentes temps réel. Cheddar est capable de gérer des modèles d'architectures AADL [5] et implante déjà de nombreux tests de faisabilité. Notons que Cheddar est en partie constitué de code généré à partir du méta modèle des entités qu'il manipule. La modélisation et la génération de code est réalisée grâce au méta-atelier Platypus [6]. Le méta modèle utilisé est rédigé EXPRESS, un langage de modélisation de données. Comme précisé dans la figure 1, le prototype implante les étapes 2, 3, 4 et 5 de la méthode du point de vue concepteur. Le code de l'outil (en Ada) a été rédigé manuellement pour le patron de conception Synchronous data-flow. L'objectif à cours terme est de générer la partie du prototype propre aux patrons de conception (les

évaluations de contraintes d'applicabilité entre autres). De cette façon nous pouvons inclure au prototype de nouveaux patrons pour lesquels le code sera généré automatiquement. Cette modélisation est également réalisée en EXPRESS et est exposée plus en détails dans la partie suivante[1].

III. Modélisation des patrons de conception

Avec pour objectif la génération automatique du code du prototype, nous proposons une modélisation des contraintes à évaluer pour déterminer les tests de faisabilité applicables aux modèles d'architectures. Dans cette partie, nous présentons les entités modélisées puis la façon dont nous en déduisons les tests de faisabilité applicables.

A. Déterminer les tests applicables

Nous avons défini, pour chacun des patrons, un nombre important de cas d'application de tests de faisabilité (126 pour Synchronous data-flow). Chacun de ces cas d'application correspond à une liste de tests de faisabilité applicables. Pour déterminer le cas d'application auquel correspond un modèle d'architecture, nous vérifions successivement les contraintes relatives aux environnements de déploiement, aux patrons de conception et aux cas d'application de tests de faisabilité. Nous avons modélisé chacune de ces entités séparément. La figure 2 représente l'ensemble des architectures temps réel. Chacun des cadres représente un sous ensemble d'architectures conformes à une entité (plus le cadre est foncé, plus le nombre de contraintes est important).

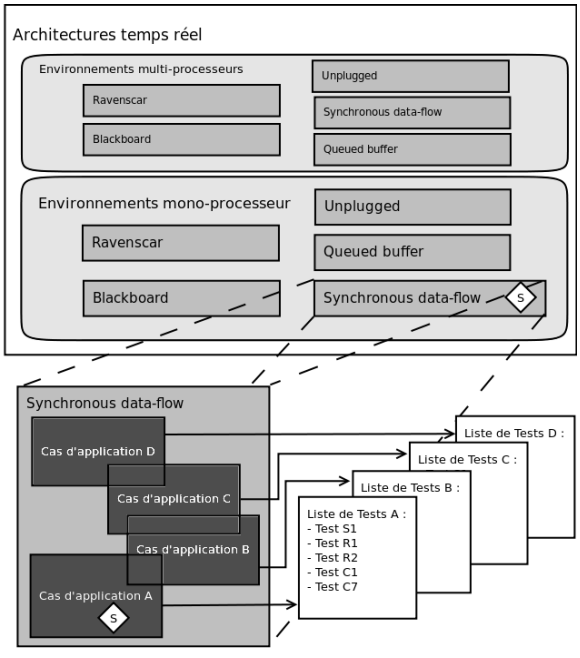

Fig. 2. Caractériser des architectures temps réel d'un point de vue ensembliste.

[1]Le prototype est disponible à l'adresse : http://beru.univ-brest.fr/svn/CHEDDAR-2.0/

B. Structure de la modélisation

Afin de pouvoir raisonner sur le contenu des modèles AADL [5], Cheddar utilise son propre méta modèle des architectures. Un système est alors représenté par plusieurs ensembles d'entités : des tâches, des ressources, des dépendances, des processeurs, des buffers, des réseaux et des espaces d'adressage. Le méta modèle de Cheddar est rédigé en EXPRESS et nous fournit tous les outils pour réaliser la modélisation de nos patrons. La figure 3 donne un extrait du méta modèle de Cheddar. Un tâche générique est définie par sa capacité, son échéance, son instant d'activation et sa priorité, entre autres. Une tâche périodique est définie comme une générique pour laquelle on précise la période et le jitter.

```
SCHEMA Tasks;
  ENTITY Generic_Task
  ... Capacity : Natural;
    Deadline : Natural;
    Start_Time : Natural;
    Priority : Priority_Range; ...
  END_ENTITY; ...
  ENTITY Periodic_Task SUBTYPE OF ( Generic_Task );
    Period : Natural_type;
    Jitter : Natural_type;
  END_ENTITY; ...
END_SCHEMA;
```

Fig. 3. Extrait du méta modèle de Cheddar : la définition d'une tâche générique et d'une tâche périodique.

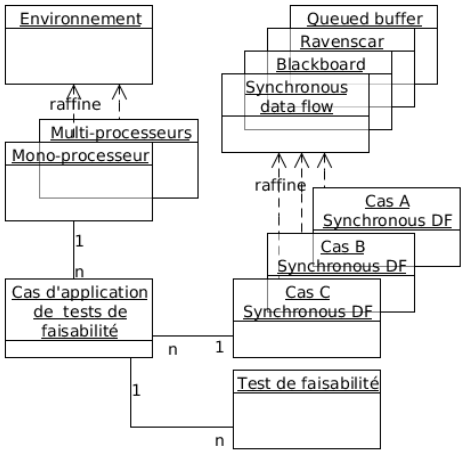

Fig. 5. Modèle d'association.

Pour chacune des entités présentées partie III-B, nous exprimons à l'aide du méta modèle de Cheddar l'ensemble de ses contraintes. Dans le cas de Synchronous data-flow, par exemple, les contraintes sont les suivantes : (1) toutes les tâches sont périodiques ; (2) le système ne contient ni tampon, ni données partagées ; (3) le protocole de partage de données entre les tâches ne peut être que soit *immediate connexion*, soit *sampled connexion* soit *delayed connexion*[2] ; (4) il existe un unique espace d'adresse par processeur. La modélisation de ces

[2]Protocoles de communication AADL [5]

```
RULE all_tasks_are_periodic FOR ( generic_task );
WHERE
  R1 : SIZEOF ( QUERY ( t <* generic_task | NOT ( 'TASKS.PERIODIC_TASK' IN TYPEOF ( t ) ) ) ) = 0;
END_RULE;
```

Fig. 4. Extrait de la spécification de Synchronous data-flow en EXPRESS

contraintes est réalisée en EXPRESS, ce qui nous permettra de générer le code. La figure 4 contient le modèle de la contrainte numéro (1), extraite de la spécification de Synchronous data-flow en EXPRESS. Elle est exprimée sous la forme d'une règle EXPRESS vérifiant que parmis toutes les tâches, le nombre de tâches non périodiques est égal à zéro.

Enfin, nous modélisons les associations entre les entités de caractérisation énoncées précédemment et les listes de tests de faisabilité. Comme le décrit la figure 5, une association contient la spécification d'un environnement, d'un cas d'application ainsi que les tests de faisabilité applicables.

IV. Travaux connexes

Plusieurs approches explorent également le domaine de la vérification de contraintes sur des systèmes temps réel. Gilles *et al.* proposent un langage de contraintes pour AADL appelé REAL (Requirement Enforcement Analysis Language)[7]. REAL est développé par Télécom-Paris-Tech et ISAE et devrait être adopté comme une annexe du standard AADL. REAL permet d'exprimer différents types de contraintes directement sur des architectures AADL. Les auteurs ont montré qu'il pouvait être utilisé pour exprimer des contraintes d'applicabilité comme celles que nous cherchons à modéliser. Une autre approche de spécification pouvant être corrélée aux patrons de conception se trouve dans la méthode HOOD et dans la définition de HRT-HOOD qui correspond au patron Ravenscar. Panuzio *et al.* proposent un processus d'ingénierie basé sur le méta modèle RCM (Ravenscar Computational Model). Des vérifications de performances sont faites avec l'environnement MAST [8]. MAST implante, comme Cheddar, plusieurs tests de faisabilité [9]. Enfin, PPOOA propose une approche similaire à nos patrons de conception [10]. PPOOA est implanté comme une extension d'UML et fournit un ensemble de mécanismes de synchronisation prédéfinis. De plus, les auteurs soulignent l'importance d'appliquer des tests de faisabilité à un modèle de système lors de la phase de conception, et utilisent Cheddar pour ce faire.

Chacune des méthodes présentées précédemment étudie la validation d'un ensemble d'architectures temps réel avec un ensemble fixe de patrons de conception. Dans notre approche, nous souhaitons que le concepteur puisse spécifier de nouveaux patrons tout en générant automatiquement l'outil de validation.

V. Conclusion

Dans cet article, nous avons présenté une approche permettant d'automatiser la vérification d'ordonnançabilité d'architectures temps réel. Nous avons proposé une méthode de conception du point de vue du concepteur. Elle permet, grâce à l'utilisation de patrons de conception spécifiques aux architectures de systèmes temps réel de déduire les tests de faisabilité applicables. On montre ainsi que l'on permet à des concepteurs, non-experts de la théorie de l'ordonnancement, de bénéficier des multiples techniques de validation existantes pour les architectures temps réel. Nous avons réalisé un prototype s'intégrant à l'environnement Cheddar qui permet d'utiliser cette méthode. La suite de nos travaux s'oriente vers deux problématiques. La première est l'analyse de systèmes plus complexes résultants de la composition des patrons de conception. Cela implique une étude approfondie des tests de faisabilité, aboutissant sur une nouvelle formalisation. Cette dernière doit définir des règles d'applicabilité des tests de faisabilité pour les combinaison de plusieurs patrons de conception. La seconde consiste à mettre en place un protocole simple afin de définir de nouveaux patrons. Enfin, l'étude d'une métrique sur les patrons de conception est en cours.

Remerciements

Nous tenons à remercier le Conseil Régional de Bretagne et Ellidiss Technologies pour leur financement du projet.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[2] P. Dissaux and F. Singhoff, "Stood and Cheddar : AADL as a Pivot Language for Analysing Performances of Real Time Architectures." Proceedings of the European Real Time System conference. Toulouse, France, Jan. 2008.

[3] A. Plantec, F. Singhoff, P. Dissaux, and J. Legrand, "Enforcing applicability of real-time scheduling theory feasibility tests with the use of design-patterns," in *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation-Volume Part I*. Springer-Verlag, 2010, pp. 4–17.

[4] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand, "Investigating the usability of real-time scheduling theory with the Cheddar project," *Real-Time Systems*, vol. 43, no. 3, pp. 259–295, 2009.

[5] SAE, "Architecture Analysis and Design Language (AADL) AS 5506," The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 1.0, Tech. Rep., Nov. 2004.

[6] "Platypus Technical Summary and download," http://cassoulet.univ-brest.fr/mme/, 2007.

[7] O. Gilles and J. Hugues, "Expressing and enforcing user-defined constraints of AADL models," in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2010, pp. 337–342.

[8] M. Panunzio and T. Vardanega, "A metamodel-driven process featuring advanced model-based timing analysis," *Reliable Software Technologies–Ada Europe 2007*, pp. 128–141, 2007.

[9] G. Harbour, G. Garcia, P. Gutierrez, D. Moyano *et al.*, "MAST: Modeling and analysis suite for real time applications," in *Real-Time Systems, 13th Euromicro Conference on, 2001.* IEEE, 2002, pp. 125–134.

[10] J. Fernández Sánchez and G. Mármol Acitores, "Modelling and Evaluating Real-Time Software Architectures," *Reliable Software Technologies–Ada-Europe 2009*, pp. 164–176, 2009.