Utilisation de l'approche par composants pour la conception d'applications temps réel

Jean-Paul Etienne et Samia Bouzefrane
Laboratoire CEDRIC, Conservatoire National des Arts et Métiers
292 rue Saint Martin
75141 Paris Cedex 03
jeanpaul.etienne@auditeur.cnam.fr, samia.bouzefrane@cnam.fr

Abstract

L'introduction de l'approche par composants dans le développement des systèmes temps réel permet de faciliter leur conception en les construisant par assemblage de composants préexistants, d'accélérer leur développement et leur déploiement par le principe de réutilisation logicielle et de faciliter leur évolution en offrant une séparation claire entre spécification et implémentation des composants. Dans cet article, nous avons voulu montrer comment concevoir une application temps réel en utilisant cette approche. Pour cela, nous avons défini un modèle de composants offrant une description explicite des différents types d'entités logicielles (actives, passives, mécanismes de communication) que nous rencontrons habituellement dans les systèmes temps réel, qui permet de décrire explicitement les besoins temporelles de chaque entité ou de l'ensemble des composants du système et qui offre des moyens permettant d'analyser et de valider l'assemblage tant au niveau fonctionnel que temporel. L'ordonnançabilité de l'assemblage de composants temps réel a été analysée à l'aide des automates temporisés.

Mots Clés : *Système temps réel, ordonnancement temps réel, approche par composants, automates temporisés.*

1. Introduction

Comme tous les domaines de l'informatique, le domaine du temps réel n'est pas épargné par le besoin de développer de plus en plus vite des systèmes de plus en plus complexes. Traditionnellement, l'utilisation de langages de bas niveau était de rigueur dans le développement de tels systèmes afin de garantir un contrôle total de leur comportement. Cependant, au fil des années, cette complexité accrue, résultant de l'évolution des besoins et l'arrivée de nouvelles technologies, a rendu le développement de ces systèmes beaucoup plus difficile. Vient alors le besoin d'introduire de nouvelles méthodologies qui faciliteraient la conception et la gestion des systèmes temps réel en se basant notamment sur « l'augmentation du niveau d'abstraction »afin d'alléger la complexité logicielle et sur « la réutilisation du logiciel »pour accélérer le développement et faciliter la certification des logiciels. L'objectif de cet article est de proposer une méthodologie pour la conception d'applications temps réel à base de composants. Cette méthodologie permet :

- de suivre toutes les étapes du cycle de développement, de l'expression des besoins jusqu'au déploiement du système temps réel sur une plate-forme d'exécution et
- d'apporter un cadre formel pour une conception rigoureuse et sûre des systèmes temps réel.

La suite de cet article est organisée comme suit. Nous commençons par rappeler ce qu'est une approche par composants dans la section 2 pour présenter ensuite notre architecture à base de composants temps réel dans la section 3 en définissant différents types de composants. Dans la section 4, nous distinguons différents types de contrats qui correspondent aux spécifications associées aux composants temps réel. La section 5 présente les différents niveaux de compatibilité à vérifier en vue d'assembler les composants d'une application. Dans la section 6, nous étudions le comportement temporel de l'architecture en analysant son ordonnançabilité à l'aide d'automates temporisés. Enfin, nous concluons dans la section 7.

2. L'approche par composants

Le développement à base de composants a pris un essor considérable depuis ces quinze dernières années. L'introduction de cette approche dans le développement des systèmes informatiques permet notamment de :

- réduire la complexité des systèmes en les concevant par assemblage de composants préexistants.
- accélérer le développement et le déploiement des systèmes car, si les composants ont des spécifications bien-définies et une implémentation conforme à ces spécifications, ils peuvent être

réutilisés efficacement dans différentes applications temps réel (du moment que leurs spécifications satisfont les exigences des applications) et
- faciliter l'évolution des systèmes, car la notion de séparation entre spécification et implémentation permet aux composants d'être mis à jour ou remplacés sans aucun redéploiement du système tout entier.

Du fait des bénéfices apportés, cette approche se trouve être la méthodologie de choix pour répondre aux besoins rencontrés dans la conception des systèmes temps réel. Cependant, les principaux modèles de composants disponibles sur le marché (CORBA/CCM de l'OMG, (D)COM/COM+ de MicroSoft et Enterprise JavaBeans de SUN) sont rarement utilisés en temps réel en raison de leurs besoins gourmands en mémoire, du manque de support pour les propriétés temporelles et leur comportement imprévisible. Néanmoins dans le monde académique, plusieurs modèles de composants sont proposés. Nous pouvons citer : la technologie AutoComp de [6], le modèle RTCOM du projet ACCORD [1] qui utilise le paradigme de la programmation par aspects, le modèle SaveCCM du projet SAVE [4], PECOS [9] et ROBOCOP mais aussi des technologies orientées composants utilisées dans l'industrie telles que KOALA [10] et RUBUS [3]. Le but principal de ce travail est de proposer une méthodologie de conception permettant la construction de systèmes temps réel par assemblage de composants. Le travail qui est présenté dans cet article consiste à :
- déterminer quelles doivent être les caractéristiques d'un composant temps réel. Cela nous permettra dans un premier temps d'établir une spécification bien définie pour les composants et dans un second temps de déterminer comment vérifier la compatibilité de leurs spécifications lors de la phase d'assemblage,
- établir une relation entre le domaine structurel des composants et le domaine temporel des tâches et
- établir des méthodes visant à vérifier l'ordonnançabilité ainsi que le bon fonctionnement d'un assemblage de composants temps réel.

3. L'architecture à composants

Nous présentons dans les paragraphes suivants, les éléments faisant partie de notre architecture logicielle temps réel.

3.1. Entité de base : le composant

Le composant est une entité logicielle de calcul, interagissant avec son environnement uniquement au moyen d'interfaces à des fins de composition et de réutilisation. Ces interfaces vont non seulement décrire les services offerts par le composant mais aussi ceux dont il requiert afin de pouvoir remplir son rôle. De ce fait, le composant sera constitué à la fois d'interfaces offertes et d'interfaces requises. Une interface est définie par un nom, un sens (requise ou offerte) et une liste d'opérations qu'elle comprend. Une opération est définie par un nom, une liste de paramètres, des propriétés fonctionnelles définies en terme de pré et de post-conditions, le type du paramètre en sortie ainsi que des propriétés temporelles (exemple, pire temps d'exécution).

3.2. Types de composants

Notre modèle de composants définit trois types de composants comme dans [7].

3.2.1 Composant actif

Un composant actif est un composant ayant son propre cycle d'exécution. En jargon temps réel, cela équivaut à une tâche. Au niveau temporel, le composant actif comprend aussi des propriétés faisant référence à sa périodicité, son échéance et sa priorité. D'un point de vue fonctionnel, le composant actif n'offre pas de services mais fait appel aux services offerts par les autres types de composants. Pour se faire, il comprend uniquement des interfaces requises. En plus des interfaces fonctionnelles, le composant actif est aussi composé d'une interface de contrôle utilisée pour réguler son cycle d'exécution. Cette interface permet notamment d'initier son exécution, dans le cas où son réveil dépend d'un événement ou d'un autre composant actif, et de le réveiller suite à sa mise en attente de ressource.

3.2.2 Composant passif

Un composant passif représente généralement des libraires de programmes ou des modules spécialisés utilisés par les composants actifs. Dans le cas où le composant passif est utilisé par plusieurs composants actifs, l'accès à ses opérations visibles est contraint par des mécanismes d'exclusion mutuelle (moniteurs ou sémaphores) afin de garantir la cohérence de son état interne. Afin de rendre explicite ces mécanismes d'exclusion mutuelle, chaque composant passif comprend une interface offerte contenant les opérations permettant son verrouillage et son déverrouillage.

3.2.3 Connecteur : Composant de communication

La notion de connecteur regroupe l'ensemble des mécanismes de communication distante (exemple, le protocole RPC) permettant aux composants d'interagir entre eux. Un connecteur peut être perçu comme un type de composant spécial. Il est utilisé dans notre architecture pour modéliser l'interaction entre les composants actifs distants.

3.3. Exemple d'illustration

Nous présentons dans la figure 1 l'architecture d'un modèle producteur-consommateur temps réel. L'architecture est composée de trois composants actifs, deux de

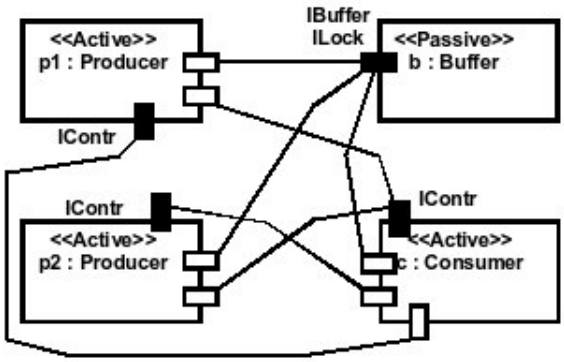


FIG. **1. Description de l'architecture**

type producteur et un de type consommateur, interconnectés à travers un composant passif représentant une file ayant une taille bornée. Les trois composants actifs sont reliés au composant *Buffer* par le biais de ses deux interfaces *IBuffer* et *ILock*. Les composants de type *Producer* sont périodiques tandis que l'exécution du composant *Consumer* est dépendante du comportement de ces derniers. Les producteurs réveillent le consommateur via son interface de contrôle *IControl* une fois qu'ils ont stocké leurs données à travers le composant *Buffer*. Dans le cas où la file est pleine, les deux producteurs se mettent en attente, jusqu'à ce qu'ils soient réveillés par le consommateur via leurs interfaces de contrôle respectives.

4. Spécification des interfaces ou Contrat

Etant donné qu'un composant est une boîte noire, il est primordial de l'équiper de spécifications bien définies afin de comprendre précisément non seulement ce qu'il permet d'accomplir mais aussi les besoins dont il requiert afin qu'il puisse remplir son rôle correctement. Ces spécifications nous permettent de décrire les propriétés structurelles, comportementales et temporelles du composant. Lors de la phase d'assemblage, les spécifications (ou contrats) offertes et requises des composants vont être confrontées entre elles afin de déterminer si leurs contraintes d'utilisation sont valides une fois que les composants sont mis en relation. Nous définissons les contrats suivants : un *contrat syntaxique* qui permet de décrire les signatures des services, c'est-à-dire le nom du service, les noms et types de paramètres d'entrées sorties. Un *contrat assertionnel*, basé sur les travaux de Meyer [8], qui définit les propriétés fonctionnelles des opérations à l'aide d'un formalisme de type pré-post condition et invariant. Par exemple, dans la spécification de l'interface *IBuffer* du composant *Buffer*, l'opération *get* ne peut être exécutée que si la file n'est pas vide.

$$long\ get()\ \{\ Pre : \neg\ empty()\ ;\ Post : true\ \}$$

Un *contrat d'interaction* qui décrit le comportement d'un composant et son interaction avec son environnement en terme de transitions sur ses opérations visibles (offertes et requises). Enfin, un *contrat temporel* qui prend en compte les aspects temporels relatifs à l'exécution et à l'interaction entre composants. Ainsi, ce type de contrat peut être greffé soit sur le contrat assertionnel ou sur le contrat d'interaction, afin de décrire des propriétés temporelles suivant les propriétés fonctionnelles des opérations ou suivant les différents flots d'exécution ou d'interaction entre composants. Par exemple, une fois qu'un composant *Producer* a stocké une donnée, le composant *Consumer* doit effectuer le retrait avant 10ms. Cela pourrait se traduire dans une logique temporelle temporisée comme suit :

$$G((p1.put \vee p2.put) \rightarrow F_{<10}(c1.get))$$

5. La compatibilité

Valider un assemblage de composants temps réel revient à vérifier que toute paire de composants à assembler est compatible de différents points de vues : syntaxique, comportemental, interactionnel et temporel.

5.1. Compatibilité syntaxique

Une compatibilité syntaxique entre opérations offertes et requises se ramène à déterminer si les types des paramètres d'entrée ainsi que ceux des paramètres de sortie sont dans une relation de sous-typage [2].

5.2. Compatibilité comportementale

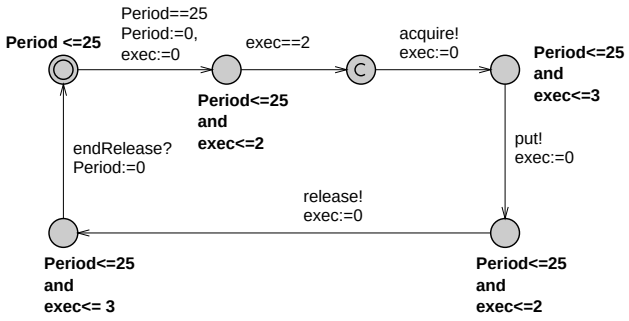
A ce niveau, l'objectif est de déterminer si les fonctionnalités définies par l'opération offerte sont bien celles attendues par l'opération requise. Suivant un formalise à la pré-post condition, la compatibilité fonctionnelle entre opérations offertes et requises consiste à vérifier si la pré-condition de l'opération offerte satisfait celle de l'opération requise et que la post-condition de cette dernière satisfait celle de l'opération offerte.

5.3. Compatibilité interactionnelle

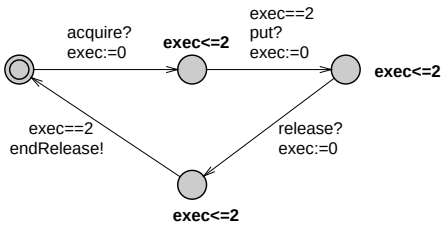
Dans le cas d'une compatibilité interactionnelle, l'objectif est de déterminer si les enchaînements d'appels et de requêtes de méthodes entre composants ne sont pas conflictuels. Ceci peut être vérifié en utilisant divers formalismes (CSP, CCS, FSP, automates communicants).

5.4. Compatibilité temporelle

D'un point de vue temporel, une compatibilité entre composants consiste à établir si les propriétés temporelles (pire temps d'exécution (WCET), échéance) des opérations offertes et requises des composants sont compatibles. Ces propriétés temporelles, insérées au niveau des contrats assertionnels et interactionnels, nous permettent de vérifier la compatibilité temporelle des opérations en assurant que les valeurs temporelles associées aux opérations requises sont supérieures ou égales à celles des opérations offertes. Dans l'exemple de la Figure 2, nous considérons une description en automates temporisés des contrats d'interaction simplifiés des composants *Producer* et *Buffer* augmentés avec



(a) Producer



(b) Buffer

FIG. **2. Contrats d'interaction des composants Producer et Buffer**

des contraintes temporelles. Le composant *Producer* est exécuté toutes les 25 unités de temps. A chaque réveil, il effectue une exécution interne pendant 2 unités de temps avant d'entamer une série d'interactions avec le composant *Buffer*. Les contraintes temporelles relatives à l'exécution des actions sont spécifiées comme des invariants sur les places succédant aux transitions de ces dernières.

6. Analyse d'ordonnançabilité

Dans toutes les analyses que nous avons exposées jusqu'à présent, nous n'avons en aucun cas statuer sur le problème d'ordonnançabilité de l'assemblage de composants, même dans le cas où nous avons pris en compte les contraintes temporelles des composants. Cette omission est justifiée car dans le cas contraire, nous aurions eu à faire lors des analyses à des espaces d'états trop grands à cause des contraintes d'ordonnançabilité. Pour cela, nous avons choisi d'effectuer une abstraction des spécifications fonctionnelles de l'assemblage de composants afin de considérer uniquement les composants actifs du système, les propriétés temporelles associées à leur exécution ainsi que les actions de contrôle régissant leur exécution, offrant donc uniquement une vision temporelle du système. Nous effectuons l'analyse d'ordonnançabilité de l'assemblage de composants à l'aide d'automates temporisés. Le choix d'un tel formalisme a été motivé d'une part par le fait qu'il nous permet de considérer des modèles d'exécution beaucoup plus complexes et d'autre part parce qu'il nous permet d'établir des estimations qui sont beaucoup plus précises que celles fournies par des analyses classiques. Le formalisme fourni par les automates temporisés permet de modéliser explicitement les comportements des tâches ainsi que leurs interactions. L'analyse d'ordonnançabilité est transformée en un problème d'atteignabilité en effectuant une exploration exhaustive de tous les comportements possibles de l'ensemble des tâches. Nous effectuons cette modélisation à l'aide de l'outil UPPAAL [5].

7. Conclusion

Dans cet article, nous avons voulu montrer comment concevoir une application temps réel en utilisant l'approche par composants. Pour atteindre cet objectif, nous avons montré comment caractériser un composant temps réel tout en mettant en évidence les différents types de composants constituant notre modèle. En vue d'assembler les composants d'une application temps réel, nous avons proposé d'une part différents niveaux de compatibilité à vérifier et d'autre part l'analyse d'ordonnançabilité de cet assemblage à l'aide des automates temporisés.

Références

[1] J. A. Tesanovic, D. Nystrom and C. Norstrom. Towards aspectual component-based development of real-time systems. *Proc. of the 9th Intern. Conf. of Real-Time and Embedded Computing Systems and Applications*, 2003.

[2] L. Cardelli. Structural subtyping and the notion of power type. *Conf. Record of the 15th Annual ACM Symp. on Principles of Programming Languages, California*, pages 70–79, January 1988.

[3] C. N. D. Isovic. Components in real-time systems. *Proc. of the 8th Inter. Conf. on Real-Time Computing Systems and Applications, Japan*, 2002.

[4] I. C. H. Hansson, M. Akerholm and M. Torngren. Saveccm : a component model for safety-critical real-time systems. *EuroMicro Conference, Special Session Component Models for Dependable Systems, Rennes, France*, September 2004.

[5] http ://www.uppaal.com/.

[6] M. A. K. Sandstrom, J. Fredriksson. Introducing a component technology for safety critical embedded real-time systems. *Intern. Symp. On Component-based Software Engineering (CBSE7), Springer Verlag, Scottland*, May 2004.

[7] L. L. M. Diaz, D. Garriddo and J. Troya. Integrating real-time analysis in a component model for embedded systems. *Proc. of the 30th EuroMicro Conference*, 2004.

[8] B. Meyer. *Object Oriented Software Construction, 2nd edition*. Prentice Hall, Englewood Cliffs NJ, 1997.

[9] C. Z. P. Muller, C. Stich. Components@work : Component technology for embedded systems. *27th International Worshop on Component-based Software Engineering, EUROMICRO*, 2001.

[10] J. K. R. Van Ommering, F. Van der Linden. The koala component model for consumer electronics software. *IEEE Computer, Vol. 33, NÂ° 3*, pages 78–85, March 2000.