Composants avec Propriétés Temporelles

Sébastien Saudrais[*] — **Olivier Barais**[*] — **Noël Plouzeau**[*]

[*] *IRISA France, Projet Triskell*[1]
{saudrais, barais, plouzeau}@irisa.fr

RÉSUMÉ. *Cet article propose une approche pour le développement d'applications mettant en avant le support de propriétés temporelles pour les composants logiciels. Notre processus s'adresse aux développeurs qui (1) construisent des applications à partir de composants avec des contraintes de temps et de performance et/ou (2) conçoivent des composants logiciels pour ce type d'applications. A partir des spécifications temporelles de l'application, le processus vérifie l'assemblage des composants et génère un connecteur en Giotto préservant les propriétés temporelles.*

ABSTRACT. *This paper proposes a software development approach that emphasizes support for time properties in software components. Our software process is best suited to developers that (1) build applications from components with timing and performance concerns and/or (2) design software components for this kind of applications. From the timed specifications, the process verifies components assembly and generates a Giotto connector preserving timed properties.*

MOTS-CLÉS : *Composants, QoS, MDE, temps-réel*

KEYWORDS: *Components,QoS, MDE, real-time*

1. Ce travail est financé par ARTIST2, le réseau d'excellence de la conception de systèmes embarqués.

1re soumission à *CAL06*, le 6 juin 2006.

1. Introduction

Depuis plusieurs années, les modèles de composants apportent une aide aux architectes et aux développeurs de systèmes d'information. Les composants fournissent une abstraction utile des modules de code et de leurs interactions. Toutefois, les modèles de composants ont différentes significations selon les personnes. Le plus souvent, les composants logiciels sont vus comme des modules logiciels sophistiqués. Ils sont développés en utilisant des langages conventionnels de programmation et sont définis dans des cadres logiciels tels que .Net, EJB ou CCM. Dans le reste du papier nous appelons ce type de composant des COTS(Commercial Off-The-Shelf).

Dans la communauté temps-réel, ces modèles abstraits COTS ne sont pas suffisants car ils ne prennent pas en compte le temps et l'ordonnancement. Des modèles abstraits ont été développés mais ne sont pas facilement utilisables. Par exemple, CQML est un langage lexical pour la spécification de Qualité de Service(QdS) [AAG 01]. Intégrable à UML et utilisable à différents niveaux d'abstraction, il ne peut cependant pas être utilisé efficacement dans un processus de développement logiciel du fait du manque d'outils associés. A l'autre extrémité du processus de développement les développeurs d'applications temps-réel s'appuyent de plus en plus sur la composition de sous-systèmes pour construire leurs applications. Les langages de modélisation correspondants (et bien sûr leurs métamodèles associés fournissant la sémantique) sont souvent des langages spécifiques au domaine de l'architecture, appelés *Langages de Description d'Architecture*. Qinna est une architecture de QdS pour composant [TOU 05]. La QdS est intégrée dans l'architecture mais pas dans le modèle. En d'autres termes, les composants temps-réel partagent le concept de composition avec les composants COTS, mais mettent en avant la justesse comportementale et temporelle, là où les COTS mettent en avant la réutilisation, l'adaptabilité et la richesse des fonctionnalités.

Notre travail est motivé par le besoin de combler les lacunes de ces deux approches. Nous voulons préserver les techniques de vérification des composants temps-réel, tout en conservant l'architecture des COTS. Par exemple, nous voulons appliquer la composition formelle des spécifications en conservant les implémentations conventionnelles. Pour ces raisons, nous nous appuyons sur deux couches différentes de modèles : une couche orientée pour la validation formelle et une couche pour l'implémentation pratique. Ces couches sont interconnectées par la transformation de modèles, suivant l'*Ingénierie dirigée par les modèles*.

Le reste de l'article est organisé comme suit. La section 2 présente un aperçu de notre approche. La section 3 décrit notre chaîne de validation. Enfin la section 4 conclut et donne les travaux futurs.

2. Un aperçu de notre approche

Notre approche repose sur un sous-ensemble approprié d'UML 2.0, adapté et étendu pour la conception d'une architecture à composants avec des fonctionnalités

liées au temps. Le sous-ensemble choisi inclut des diagrammes d'architecture avec des spécifications structurelles d'interface de composant. Notre processus de conception d'un composant repose sur les artefacts suivants : une spécification de service, une spécification de composant, une implémentation abstraite du composant et enfin une implémentation concrète du composant.

2.1. *Spécification de service*

La spécification de service décrit ce qui est requis pour l'exécution d'un service : définitions des types et des opérations utilisées. Les opérations peuvent avoir des contraintes tels que les types, les pré et postconditions, les propriétés comportementales et relatives aux temps. Dans le monde UML 2.0, un service est associé avec la définition d'une interface qui spécifie la méthode qui peut être exécutée. Notre modèle de composants doit supporter la spécification de propriétés comportementales et temporelles et garder les formalismes souvent utilisés pour les niveaux de contrat 2 (logique de prédicat) et 3 (logique temporelle classique) tel que définis dans [BEU 99]. Pour cela, nous choisissons TCTL (Timed Computation Tree Logic) [ALU 93], qui est une extension de CTL avec des opérateurs temporels quantitatifs. Les contrats temporisés en TCTL sont capables d'améliorer les contraintes temporelles sur les évenements de service. Un cas classique et très commun est la propriété de temps de réponse : le temps entre l'invocation du service et la fin du service doit être inférieur à une valeur connue à l'avance. Cet propriété peut être écrite en TCTL :

$$invocation \Rightarrow \forall\Diamond_{<temps\ de\ reponse} reponse \qquad [1]$$

2.2. *Spécification du composant*

Une spécification du composant est un recueil des services fournis par celui-ci : elle donne des contraintes supplémentaires sur la coordination et les interactions entre les services. Une spécification de composant définit l'ensemble minimum de fonctionnalités et de propriétés que l'implémentation du composant doit proposer.

2.3. *Implémentation abstraite du composant*

La prochaine étape dans le cycle de conception d'un composant est l'implémentation abstraite du composant. Une implémentation abstraite du composant doit adhérer à une spécification de composant pour implémenter un ensemble de services ; cette implémentation abstraite publie des informations additionnelles spécifiques à l'implémentation, telles que l'ensemble des services requis pour effectuer ses tâches et les bornes des propriétés quantitatives des services fournis. Ces bornes dépendent généralement des propriétés quantitatives de l'environnement. Une implémentation abstraite cache tous les détails spécifiques à la plate-forme : c'est une description

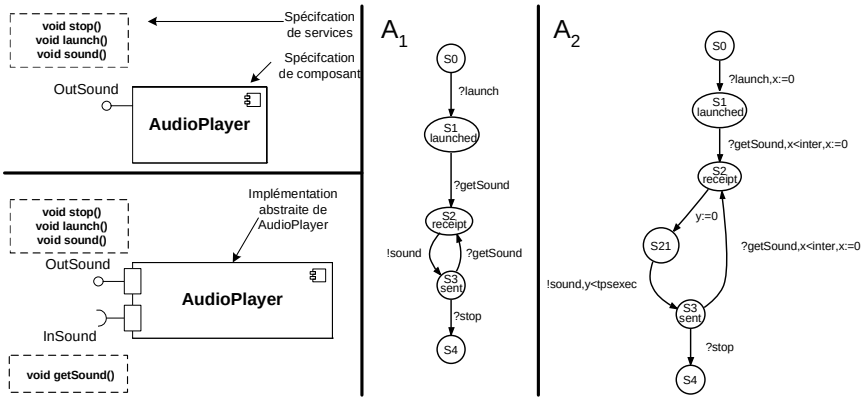

Figure 1. *Exemple du composant AudioPlayer*

adaptée en vue de la validation formelle et pour le calcul des propriétés de la composition.

Le comportement temporel d'un composant permet de définir les interactions du composant avec son environnement. Ce comportement est représenté par un automate temporisé [ALU 94]. L'automate décrit les séquences de messages qui peuvent être échangés entre le composant et son environnement. Un automate temporisé est un automate étendu avec un ensemble fini d'horloges, utilisé pour exprimer les contraintes temporelles. Une transition est tirable seulement si la contrainte temporelle qui lui est associée est satisfaite par les valeurs courantes des horloges.

La figure 1 illustre le modèle avec un exemple de composant $AudioPlayer$. La partie gauche montre la représentation structurelle du composant en UML 2.0. Le centre de la figure montre un automate A_1 décrivant tous les comportements possibles $AudioPlayer$[1]. Nous introduisons du temps dans A_1 et nous obtenons A_2.

2.4. *Implémentation concrète du composant*

Une implémentation concrète du composant est une entité au niveau code qui est exécutable sur un système d'exécution pour composants. Elle doit fournir les services de ses implémentations abstraites de composant, avec les propriétés associées. Elle est réalisée en utilisant les langages de programmation traditionnels, tels que Java et C++, compatibles avec le cadre logiciel Giotto [HEN 03]. Comme détaillé dans la figure 2, nous définissons trois niveaux dans l'implémentation concrète : fonctionnel, interaction temporelle et plate-forme. Les composants métiers générés à partir de la spécification des services et de l'implémentation abstraite se trouvent au niveau

1. Dans la figure 1, nous ne représentons pas la réponse à chaque transition pour simplifier l'automate. Les communications sont synchrones.

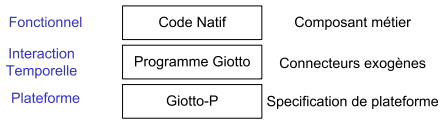

Figure 2. *Modèle d'implémentation concrète de composant*

fonctionnel. Notre implémentation concrète de composant utilise le paradigme des *connecteurs exogènes* [ARB 03, LAU 05] qui encapsulent complètement le contrôle entre les composants dans des connecteurs. Les connecteurs exogènes générés à partir des automates temporisés et des contraintes de temps se trouvent au niveau de l'interaction temporelle. Ces connecteurs sont générés par transformation de modèles en partant du méta-modèle des automates temporisés vers le méta-modèle de Giotto. Finalement, au niveau plate-forme, nous trouvons la spécification des plates-formes comme la topologie des CPU et réseaux ainsi que de la performance.

3. Chaîne de validation

Notre approche consiste à fournir une suite complète d'outils de validation qui aide l'architecte à chaque étape du développement logiciel. Cette suite est construite à partir des trois étapes illustrées dans la figure 3 : la vérification de la cohérence de l'architecture pour les concepts de temps, cohérence entre la conception et les exigences et finalement, la cohérence de l'implémentation concrète sur une plate-forme.

Cohérence de la composition des composants connectés. Dans le modèle de composants, le comportement de chaque composant primitif est défini par un automate temporisé. Le comportement d'un composite est obtenu par synchronisation des automates temporisés de ses fils. Dans ce contexte, IFx [GRA 05] permet de vérifier les conditions de la cohérence comme les interblocages ou les blocages temporels. IFx génére les traces de diagnostic qui aident l'architecte à construire et déverminer l'application en utilisant le simulateur.

Cohérence exigence/architecture. Le comportement de chaque composant primitif est modélisé par un automate temporisé et les exigences sont exprimées dans les contrats temporisés en TCTL. A partir de ces données, Kronos [BOZ 98] vérifie si un automate temporisé satisfait une formule TCTL ou non.

Cohérence de l'implémentation concrète. Notre approche vise à supprimer le fossé entre les techniques utilisés par les développeurs pour implémenter leur application et le modèle utilisé par le concepteur pour spécifier et analyser leur système. L'utilisation des techniques de transformation de modèles garantira que l'implémentation concrète a les mêmes contraintes temporelles que la spécification et l'implémentation abstraite. Au niveau de l'implémentation concrète, la couche "connecteurs exogènes" est responsable de la vérification du respect des contraintes. Ensuite, l'utilisation de Giotto comme cible d'implémentation concrète permet à l'architecte de vérifier si la

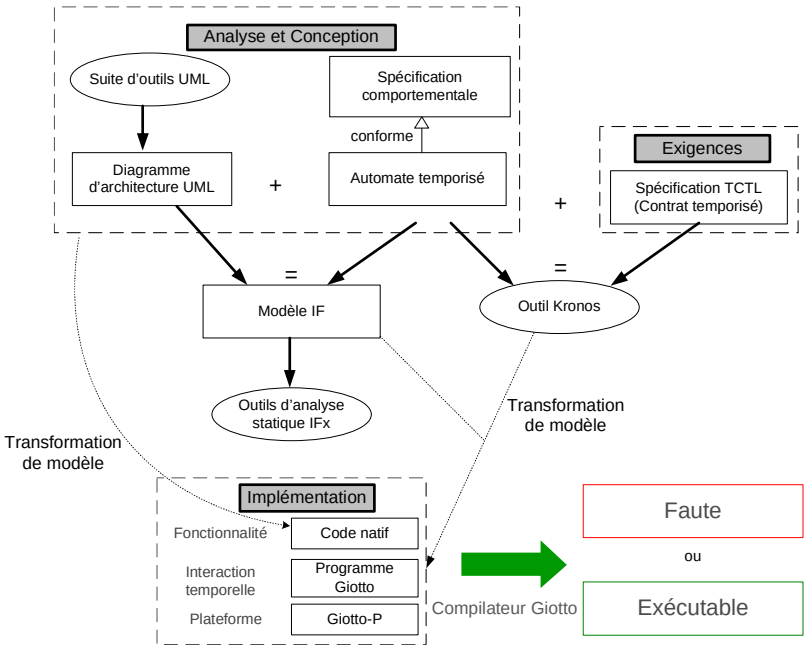

Figure 3. *Chaîne de validation*

spécification de la plate-forme est suffisament contrainte pour obéir aux contraintes temporelles.

4. Conclusion et perspectives

Le fossé entre le modèle de spécification et le modèle d'implémentation est souvent une source d'erreurs lors de la conception et l'implémentation d'un système temps-réel. Dans cet article, nous avons proposé une approche unifiée pour concevoir et implémenter des systèmes à base de composants. Cette approche a pour but d'assister les architectes dans le développement de systèmes temps-réel en leur fournissant un ensemble d'outils qui vérifie l'exactitude de ces systèmes. L'approche est basée sur le standard UML 2.0 étendu pour concevoir les services fournis par composant, pour spécifier le composant et pour donner une première implémentation abstraite du système. De plus, en utilisant l'ingénierie dirigée par les modèles, l'approche fournit des possibilités de génération de code qui sépare clairement le niveau fonctionnel, les interactions temporelles et le niveau plate-forme.

Les travaux futurs consistent à prouver l'exactitude du processus de transformation. La preuve devra inclure la vérification que le mécanisme de composition, au niveau de l'implémentation concrète, est valide par rapport au mécanisme de com-

position abstrait en vue de préserver les résultats obtenus par validation de la phase d'implémentation abstraite. De plus, nous devons prouver la cohérence de la transformation de modèle exprimée en Kermeta [MUL 05]. Enfin, nous validerons notre approche sur une étude de cas industrielle.

5. Bibliographie

[AAG 01] AAGEDAL J. Ø., « Quality of Service Support in Development of Distributed Systems », PhD thesis, Department for Informatics, University of Oslo, juin 2001.

[ALU 93] ALUR R., COURCOUBETIS C., DILL D., « Model-Checking in Dense Real-time », *Information and Computation*, vol. 104, n° 1, 1993, p. 2-34.

[ALU 94] ALUR R., DILL D., « A theory of timed automata », *Theor. Comput. Sci.*, vol. 126, n° 2, 1994, p. 183–235, Elsevier Science Publishers Ltd.

[ARB 03] ARBAB F., RUTTEN J., « A Coinductive Calculus of Component Connectors », *Lecture Notes in Computer Science*, vol. 2755, 2003, p. 34 - 55.

[BEU 99] BEUGNARD A., JÉZÉQUEL J.-M., PLOUZEAU N., WATKINS D., « Making Components Contract Aware », *Computer*, vol. 32, n° 7, 1999, p. 38–45, IEEE Computer Society Press.

[BOZ 98] BOZGA M., DAWS C., MALER O., OLIVERO A., TRIPAKIS S., YOVINE S., « Kronos : A model-checking tool for real-time systems », *Proc. 1998 Computer-Aided Verification, CAV'98*, vol. 1427 de *Lecture Notes in Computer Science*, Vancouver, Canada, June 1998, Springer-Verlag.

[GRA 05] GRAF S., HAUGEN O., OBER I., SELIC B., Eds., *Specification and Validation of UML models for Real Time and Embedded Systems : An STTT special section*, vol. under press de *STTT, Journal for Software Tools for Technology Transfer*, Juin 2005.

[HEI 05] HEINEMAN G., I.CRNKOVIC, SCHMIDT H., STAFFORD J., SZYPERSKI C., WALLNAU K., Eds., *Component-Based Software Engineering, 8th International Symposium, CBSE 2005, St. Louis, MO, USA, Mai 14-15, 2005, Proceedings*, vol. 3489 de *Lecture Notes in Computer Science*, Springer, 2005.

[HEN 03] HENZINGER T., KIRSCH C., HOROWITZ B., « Giotto : A Time-triggered Language for Embedded Programming », *Proceedings of the IEEE*, vol. 91, n° 1, 2003, p. 84–99.

[LAU 05] LAU K.-K., E. P. V., WANG Z., « Exogenous Connectors for Software Components. », Heineman et al. [HEI 05], p. 90-106.

[MUL 05] MULLER P.-A., FLEUREY F., JÉZÉQUEL J.-M., « Weaving Executability into Object-Oriented Meta-languages. », BRIAND L. C., WILLIAMS C., Eds., *MoDELS*, vol. 3713 de *Lecture Notes in Computer Science*, Springer, 2005, p. 264-278.

[TOU 05] TOURNIER J., BABAU J., OLIVE V., « Qinna, a Component-Based QoS Architecture. », Heineman et al. [HEI 05], p. 107-122.