Manuscrit auteur, publié dans "11èmes Journées Francophones sur les Approches Formelles dans l'Assistance au Développement de Logiciels, Grenoble : France (2012)"

hal-00756230, version 1 - 22 Nov 2012

IDCM : un outil d'analyse de composants et d'architectures dédié à la construction incrémentale

Hong-Viet Luong[1], Anne-Lise Courbis[2], Thomas Lambolais[2], Thanh Liem Phan[2]

1. Laboratoire Ampère – UMR 5005 – France {hong-viet.luong@insa-lyon.fr}
2. LGI2P – Ecole des mines d'Alès – France {prenom.nom@mines-ales.fr}

1. Introduction

Pour faire face à la complexité de modélisation des systèmes à logiciel prédominant, nous préconisons une démarche incrémentale de développement de modèles. Il s'agit de permettre aux concepteurs de définir pas à pas un modèle du système à concevoir en partant d'une spécification incomplète, souvent indéterministe, qui sera affinée, corrigée et complétée. L'objectif est de permettre de suivre une démarche de mise au point de modèles similaire à celle préconisée pour le développement de programmes, enchaînant des cycles de « proposition évaluation correction ». La spécification initiale s'exprime sous forme d'une machine d'états ou d'un assemblage de composants. Certains composants peuvent être des boîtes noires dont seul le protocole d'utilisation est connu (cas de composants sur étagère), d'autres peuvent être en cours de développement. Le développement incrémental se fait selon deux axes : un axe horizontal permettant d'ajouter de nouvelles fonctionnalités par un enrichissement des interfaces et l'ajout éventuel de composants ; un axe vertical permettant d'ajouter des détails de façon à obtenir en fin de processus un modèle pouvant donner lieu à une génération automatique de code. Nos travaux se distinguent des travaux classiques de raffinement par la possibilité d'enrichir le modèle au cours de son développement par de nouvelles fonctions, ce qui correspond à l'axe horizontal. Il est ainsi possible de partir d'un modèle ne prenant en compte que quelques fonctionnalités du futur système. Les techniques d'évaluation proposées entre deux étapes s'assurent que la vivacité du système est préservée (pas d'ajout de blocage ni de refus) ou que la sûreté est maintenue lors des développements verticaux (pas d'ajout de fonctions interdites).

Nous nous sommes basés sur les relations de comparaison de modèles proposées sur les LTS [1, 2, 3] (Labelled Transition Systems). Notre première tâche a été de définir un transformateur de modèles UML en LTS. La seconde tâche a été de proposer et d'implanter des algorithmes de vérification des relations de conformité définies sur les LTS [4, 5]. L'outil que nous présentons, appelé IDCM (*Incremental Development of Conforming Models*), regroupe les fonctionnalités de transformation et de comparaison de modèles. Nous montrons l'apport de chacune de ces relations à partir de modèles simples réalisés sous TopCased.

2. Cas d'étude

Le système que nous avons choisi pour illustrer notre travail [6] a la spécification suivante : il réceptionne une tâche à traiter (service offert IN) et la restitue après son traitement (service requis OUT). Sa spécification comportementale de haut niveau peut être représentée par une machine d'états à deux états : attente d'une tâche et traitement en cours. La transition de l'attente au traitement se fait par la réception d'une tâche (IN) et le retour à l'état d'attente se fait en émettant la tâche traitée (OUT). Le traitement de la tâche peut être fait en interne par le composant de réception (composant appelé TaskManagement) ou sous-traité à un autre composant (appelé TaskTreatment), qui doit lui aussi être spécifié. Les conditions de sous-traitance des tâches ne sont pas définies. On suppose que c'est le module TaskManagement qui décide selon des critères qui lui sont propres (surcharge de travail, tâches difficiles etc.). On fait l'hypothèse que le système est sans mémoire et qu'une seule tâche est traitée à la

fois. La Figure 1 montre l'architecture initiale du système où apparaissent les composants TaskManagement et TaskTreatment. Nous définissons pour chaque composant une machine d'états (cf. Figure 2) que nous allons affiner et compléter. Nous montrerons au cours des étapes de modélisation les problèmes détectés.

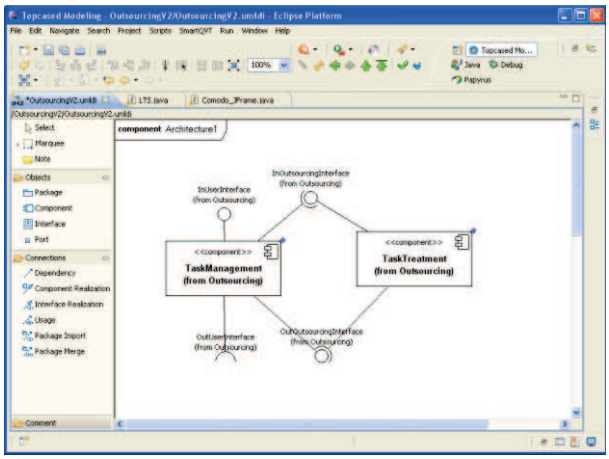

Figure 1. Modélisation d'une architecture

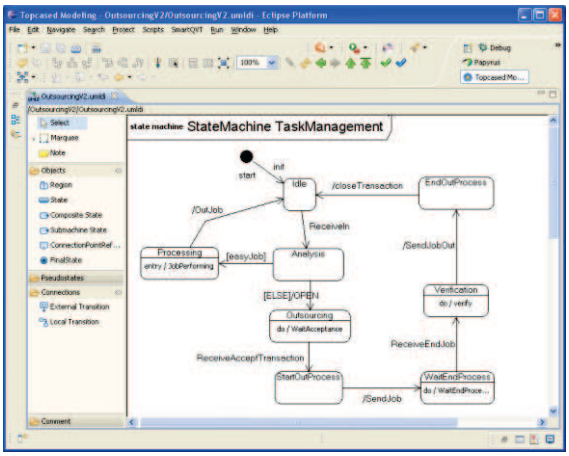

Figure 2. Modélisation comportementale d'un composant

3. Transformation des modèles UML en LTS

La transformation des machines d'états en LTS est un programme JAVA qui implante des règles compositionnelles de transformation définies en fonction de configurations spécifiques : état avec activité, transition avec trigger, transition avec garde, transition avec action. La transformation procède au masquage de toutes les actions non contrôlables et non observables. Seuls restent visibles les éléments des interfaces fournies et requises de chaque composant.
La transformation d'une architecture en LTS nécessite deux étapes. La première étape, automatique et implantée dans IDCM, consiste à générer à partir de la description UML de l'assemblage, un code écrit en formalisme Exp 2.0 [7]. Le langage Exp offre l'avantage de modéliser le système sous forme de réseaux d'automates communicants, réseau dans lequel la communication est décrite aisément grâce à la notion de vecteurs de synchronisation. La deuxième étape consiste à générer le modèle LTS correspondant au code Exp. Cette transformation est réalisée automatiquement par Exp OPEN [7]. On dispose alors du modèle LTS du comportement de chaque composant ainsi que de celui de l'architecture.

4. Relations de comparaison de modèles comportementaux

Les relations que nous avons implantées dans IDCM (cf. Figure 3) permettent de comparer deux modèles obtenus au cours d'un processus de construction incrémentale. Lorsqu'une erreur est détectée (cf. Figure 4), IDCM indique une trace d'actions contrôlables et observables problématique. Le modélisateur peut alors analyser cette trace sur la machine d'états ou l'assemblage de machines pour comprendre son erreur et apporter des corrections. La figure 5 récapitule dans une vue ensembliste ces relations [5]. On désigne par *modèle de référence*, le dernier modèle validé et *modèle courant,* celui qui est à valider. Les relations ont les caractéristiques suivantes :

- conf, relation de conformité : cette relation assure que toute action refusée par le modèle courant l'est aussi par le modèle de référence [2]. C'est une relation permettant de vérifier si le modèle courant est une implantation *conforme* du modèle de référence, tel que cela est défini

hal-00756230, version 1 - 22 Nov 2012

dans la méthodologie de test de conformité proposée par l'ISO. Elle pose cependant problème: elle n'est pas transitive, propriété nécessaire pour la construction incrémentale. Elle pourra être utilisée en tant que relation d'implantation et non comme relation de raffinement. Nous avons proposé un algorithme pour la vérifier dans [4].

- confrest, relation de raffinement : cette relation traduit le fait que toute implantation du modèle courant est aussi une implantation du modèle de référence.
- ext, relation d'extension : c'est une relation de conformité pour laquelle le modèle courant peut avoir plus de traces que le modèle de référence, tout en restant conforme. Cette relation est intéressante pour la construction incrémentale selon l'axe horizontal : elle permet d'ajouter des fonctionnalités en cours de développement et elle est transitive. Elle est aussi une relation de raffinement.
- red*, relation de réduction : cette relation assure que le modèle courant est un raffinement du modèle de référence et qu'il n'a pas plus de traces. Elle permet de vérifier que le modèle courant assure les fonctionnalités obligatoires définies dans le modèle de référence. C'est une relation appropriée pour la construction incrémentale selon l'axe vertical.

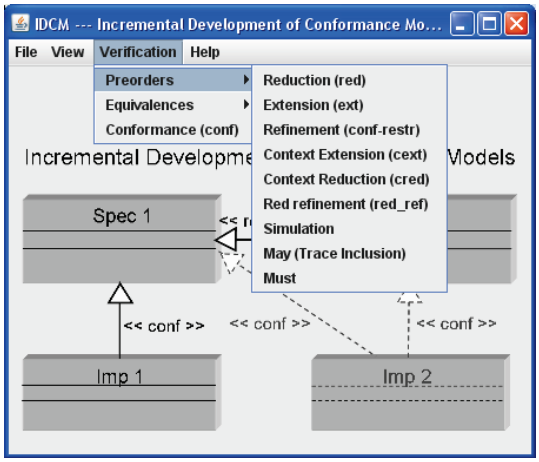

Figure 3. Analyse de conformité de composants

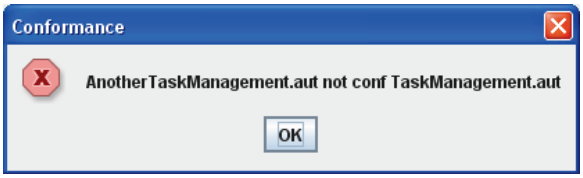

Figure 4. Diagnostic de conformité de composants

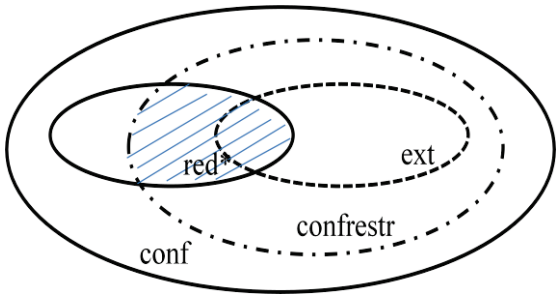

Figure 5. Vue ensembliste des relations de conformité implantées dans IDCM

Ces relations possèdent une propriété particulièrement intéressante : quel que soit leur ordre d'utilisation, c'est-à-dire quelle que soit la stratégie choisie par le modélisateur, le modèle final sera conforme au modèle initial. Les relations permettent donc de mettre en œuvre différentes méthodes de développement telles que :

hal-00756230, version 1 - 22 Nov 2012

- la méthode conventionnelle : les spécifications sont élaborées de façon complète (développement de l'axe horizontal) avant de commencer la réalisation (développement de l'axe vertical).
- la méthode prudente : on réalise une partie de la spécification pour étudier par exemple sa faisabilité ou avoir un retour des utilisateurs (développement de l'axe vertical), puis on complète la réalisation en fonction des évolutions demandées (développement de l'axe horizontal).
- la méthode agile : on construit une première version du système par des développements horizontaux et verticaux successifs. Le modèle d'implantation obtenu sera candidat pour être la spécification initiale de la version suivante du produit.

5. Travaux en cours

Nos travaux en cours portent sur la vérification incrémentale d'architectures : il s'agit de définir les conditions pour lesquelles la substitution par un nouveau composant d'un composant déjà défini au sein d'un assemblage ne perturbe pas le fonctionnement du système. Nous pouvons montrer sur des exemples simples que la relation de conformité entre composants n'est pas suffisante. Nous travaillons sur l'implantation de relations de conflit définies dans [8] permettant la détection d'interblocages et de livelocks. Ce sont des relations de congruence compatibles avec les opérateurs de masquage et synchronisation, adaptées à la vérification incrémentale.

Les relations présentées ci-dessus permettent d'analyser la conformité d'architectures, puisque nous pouvons générer automatiquement le LTS d'une architecture. Mais dans le cas de systèmes avec de nombreux composants, la complexité des algorithmes limitent les cas d'application. A l'heure actuelle, IDCM permet de traiter en un temps raisonnable des LTS dont le nombre d'états est de l'ordre du million, avant minimisation.

6. Bibliographie

[1] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[2] E. Brinksma, G. Scollo, *Formal notions of implementation and conformance in LOTOS*, Tech. report INF-86-13, Twente University of Technology, Department of Informatics, Enschede, Netherlands, December 1986.

[3] G. Leduc, *Conformance relation, associated equivalence, and minimum canonical tester in LOTOS*, PSTV XI, North-Holland, 1991, p.249-264.

[4] H.-V. Luong, T. Lambolais, A.-L. Courbis, *Implementation of the Conformance Relation for Incremental Development of Behavioural Models*, Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, 2008, Volume 5301/2008, p.356-370.

[5] T. Lambolais, A.-L. Courbis, H.-V. Luong, *Raffinement de modèles comportementaux UML, vérification des relations d'implantation et d'extension sur les machines d'états*, AFADL 2009.

[6] T. Lambolais, A.-L. Courbis, H.-V. Luong, T.-L. Phan , *Interoperability Analysis of Systems*, 18th IFAC World Congress, Milano, September 2011, p. 7879-7884

[7] F. Lang, *Exp OPEN 2.0: A flexible tool integrating partial order compositional and on-the-fly verification methods*, Lecture Notes in Computer Sciences, 2005, Volume 3771/2005, p.70-88.

[8] R. Malik, D. Streader, S. Reeves, *Conflicts and fair testing*, International Journal of Foundations of Computer Science 17(4), p.797-813.

hal-00756230, version 1 - 22 Nov 2012