Une approche de vérification d'exécution de modèles par contrats

Eric Cariou* — **Cyril Ballagny**** — **Alexandre Feugas***** — **Franck Barbier***

* *Université de Pau et des Pays de l'Adour /* LIUPPA*, BP 1155, 64013 Pau Cedex*
{Eric.Cariou, Franck.Barbier}@univ-pau.fr
** SOFTEAM*, Objecteering Software, 8 Parc Ariane, 78284 Guyancourt Cedex*
Cyril.Ballagny@softeam.fr
*** INRIA *Lille-Nord Europe /* LIFL CNRS UMR *8022 / Université de Lille 1*
Cité scientifique, Bât. M3, 59655 Villeneuve d'Ascq Cedex
Alexandre.Feugas@inria.fr

RÉSUMÉ. *Un des buts fondateurs de l'ingénierie des modèles est la manipulation des modèles en tant qu'éléments logiciels productifs. L'exécution de modèles est notamment un moyen de remplacer l'écriture du code. Nous nous intéressons dans cet article à la vérification d'exécution de modèles. Nous utilisons pour cela une approche par contrat pour spécifier une sémantique d'exécution pour un méta-modèle donné. Nous montrons qu'une sémantique d'exécution est un prolongement naturel d'un méta-modèle rigoureusement défini et est formée de niveaux complémentaires, de la définition d'éléments statiques et dynamiques ainsi que de spécifications d'exécution de ces éléments. Nous utilisons des contrats de transformation de modèles pour vérifier la cohérence de l'évolution dynamique d'un modèle pendant son exécution.*

ABSTRACT. *One of the main goal of model-driven engineering is the manipulation of models as exclusive software artifacts. Model execution is in particular a means to substitute models for code. We focus in this paper on verifying model executions. We use a contract-based approach to specify an execution semantics for a meta-model. We show that an execution semantics is a seamless extension of a rigorous meta-model specification and composed of complementary levels, from static element definition to dynamic elements, execution specifications as well. We use model transformation contracts for controlling the dynamic consistent evolution of a model during its execution.*

MOTS-CLÉS : *IDM, exécution de modèles, vérification, contrats*

KEYWORDS: *MDE, model execution, verification, contracts*

1. Introduction

Un des buts fondateurs de l'ingénierie des modèles (IDM) est de considérer les modèles comme les éléments principaux et productifs pour le développement d'applications. Cela peut être réalisé par la génération de code à partir d'un modèle mais également par directement exécuter un modèle. Cela nécessite de disposer de techniques et de langages pour exécuter des modèles mais il faut également s'assurer que cette exécution est cohérente. Dans cet article, nous présentons une approche de vérification d'exécution de modèles. La conception et la programmation par contrats (Meyer, 1992, Beugnard *et al.,* 1999) est une approche bien connue de vérification d'exécution d'éléments logiciels. Nous proposons de l'appliquer à la vérification d'exécution de modèles, dans un contexte IDM.

Il existe de nombreuses approches de vérification d'exécution de modèles. Elles ont quasiment toutes pour caractéristique de nécessiter un autre espace technologique que celui de méta-modélisation (UML, MOF, Ecore ...) afin d'en utiliser les techniques et outils de vérification. Par exemple, (Combemale *et al.,* 2009) utilise des réseaux de Pétri temporisés pour faire de la vérification sur des modèles Ecore. Si ces espaces technologiques offrent des outils performants et puissants, notamment basés sur du *model-checking*, cela pose néanmoins deux problèmes. Le premier est qu'il faut définir une sémantique translationnelle pour passer d'un espace technologique à un autre (transformer par exemple un modèle Ecore en un réseau de Pétri). Le second est que le concepteur doit maîtriser cet espace technologique supplémentaire. Nous proposons dans cet article une approche par contrat permettant de définir une sémantique complète d'exécution en restant dans l'espace technologique de méta-modèlisation. Avant de détailler cela dans la section 3, nous montrons dans la section suivante qu'une exécution de modèle se ramène à une série de transformations. Cela permet alors d'utiliser des techniques de vérification de transformations de modèles.

2. Exécution et transformations de modèles

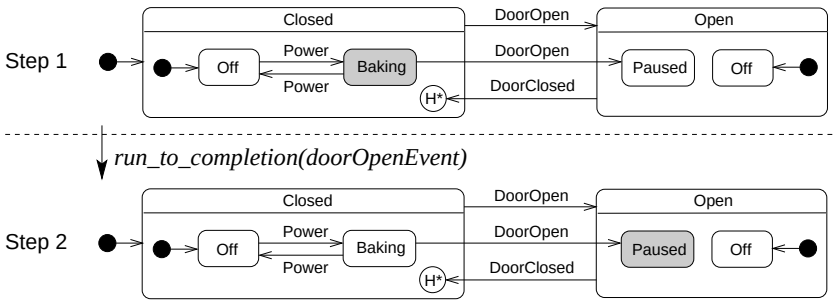

Figure 1. *Exemple d'exécution d'une machine à états UML*

La figure 1 représente deux étapes de l'exécution d'une machine à états UML. Cette machine modélise le fonctionnement d'un four à micro-onde, avec deux états composites principaux selon que sa porte est ouverte ou fermée. A la première étape, l'état actif est *Baking* de l'état *Closed* (l'état actif est représenté avec un fond gris). Le pas d'exécution consiste ici à traiter l'occurence de l'événement *DoorOpen* qui implique de suivre la transition allant de *Baking* à *Paused*. Cet état devient donc le nouvel état actif.

Le pas d'exécution est associé à l'opération *run_to_completion* qui traite l'événement et fait évoluer le modèle (cette opération est attachée à un méta-élément ou est juste virtuelle pour discrétiser les pas d'exécution du modèle au sein du moteur d'exécution). Ici, c'est l'état actif qui est modifié. Le modèle après l'appel de l'opération est donc différent du modèle avant. En d'autres termes, le modèle est transformé. Ici, la transformation consiste simplement à changer l'état actif du modèle, mais on peut imaginer d'autres cas d'exécutions plus complexes. Par exemple, on peut raffiner à l'exécution l'état *Baking* en un état composite comportant plusieurs niveaux de puissance de cuisson. Ce genre de modification structurelle à l'exécution est ce qui peut typiquement être réalisé lors d'une auto-adaptation à un contexte (Ballagny *et al.,* 2009). Ainsi, un pas d'exécution est concrètement réalisé par une transformation. Une exécution de modèles (plusieurs pas) consiste donc en une série de transformations.

3. Définition d'une sémantique d'exécution pour la vérification

3.1. *Classification de niveaux de sémantique*

La définition d'un méta-modèle rigoureux consiste en un diagramme structurel définissant les concepts du méta-modèle et leurs relations. Ce diagramme n'étant pas suffisant pour décrire l'ensemble des contraintes entre et sur les éléments, il faut lui ajouter des contraintes exprimées dans un langage dédié, comme OCL. Ces règles supplémentaires sont appelées les *well-formedness rules* (règles de bonne formation).

Comme déjà détaillé par d'autres travaux, notamment (Combemale *et al.,* 2009), exécuter un modèle nécessite que son méta-modèle définisse des éléments statiques et dynamiques pour pouvoir spécifier l'état complet du modèle pendant son exécution. En nous basant sur ces travaux, nous proposons une classification propre de niveaux de spécification complémentaires. Un méta-modèle contient trois niveaux :

Méta-modèle structurel C'est la définition standard d'un méta-modèle sans prendre en compte aucune considération d'exécution. Il définit les éléments statiques avec leurs règles de bonne formation. Pour une machine à états, on y trouvera par exemple les concepts d'état ou de transition.

Méta-modèle exécutable [1] Ajout d'éléments dynamiques avec leurs règles de bonne formation afin de pouvoir modéliser entièrement l'état d'un modèle lors de son

1. Nous utilisons ici le raccourci de méta-modèle exécutable mais ce sont des modèles conformes à ce méta-modèle qui sont exécutables, pas le méta-modèle lui même.

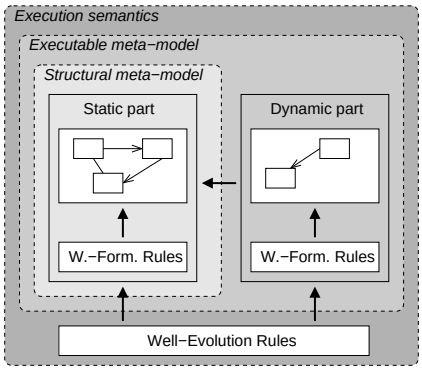

Figure 2. *Niveaux de sémantique pour la définition d'un méta-modèle*

exécution. Pour une machine à états, on y trouvera par exemple la notion d'état actif.

Sémantique d'exécution Ajout de la définition précise de la façon dont le modèle doit évoluer pendant son exécution, via des règles de bonne évolution – *well-evolution rules*. Concrètement, les spécifications des opérations d'exécution, comme *run_to_completion*, peuvent contenir toutes ces règles puisque les opérations d'exécution discrétisent l'exécution d'un modèle.

La complémentarité de ces trois niveaux est représentée sur la figure 2. Pour un DSL donné, son méta-modèle exécutable est a priori unique (une seule façon de représenter l'état d'un modèle à l'exécution). Par contre, il est possible de définir plusieurs sémantiques d'exécution. Cela permet de gérer des variantes de sémantiques d'exécution.

3.2. *Contrats pour une sémantique d'exécution*

De manière traditionnelle, une approche par contrats consiste à spécifier des éléments logiciels via des invariants et leurs opérations via des pré et des post-conditions. Dans un contexte d'exécution de modèles, ces invariants peuvent être les règles de bonne formation ou d'autres contraintes supplémentaires sur les éléments statiques et dynamiques, et les opérations à spécifier sont les opérations d'exécution. Ainsi, on peut appliquer directement une approche par contrats pour spécifier une sémantique d'exécution. Notons que cette spécification se fait en restant dans l'espace technologique de méta-modélisation. Par exemple, si on définit un méta-modèle en Ecore, augmenté de contraintes OCL pour une spécification rigoureuse, il est possible de décrire toute la sémantique d'exécution en Ecore et OCL car OCL permet de spécifier les couples de pré et post-conditions des opérations d'exécution.

Dans (Cariou *et al.,* 2009), nous proposons une approche par contrats pour vérifier des transformations de modèles. Comme une exécution de modèle est une série de transformations de modèles, nous utilisons ces contrats pour la vérification des opérations d'exécution. Nous avons également montré qu'un couple de pré/post-condition peut-être exprimé sous la forme de trois invariants. Cela est intéressant puisqu'une spécification par couple de pré et post-condition est contraignante au niveau de leur évaluation : par principe, c'est le moteur d'exécution qui doit les vérifier et cela ne se fait donc qu'à l'exécution. Avec des invariants, il est également possible de les vérifier a posteriori, si le moteur enregistre les modèles après chaque pas d'exécution.

Ainsi, il est possible d'utiliser ces contrats dans plusieurs buts, comme de la simulation en générant une trace d'exécution qui sera vérifiée une fois l'exécution simulée terminée. Ils peuvent également servir d'oracle pour des techniques de *model-checking* ou être utilisés à l'exécution pour vérifier la validité d'une auto-adaptation du modèle.

4. Conclusion

Nous avons proposé l'application d'une approche par contrats à l'exécution de modèles. Cette approche a l'avantage de pouvoir spécifier entièrement une sémantique axiomatique d'exécution en tant qu'extension naturelle de la définition standard d'un méta-modèle. Une version étendue de cet article est proposée dans (Cariou *et al.,* 2011). Elle décrit plus en détail le cadre général d'expression des contrats ainsi que son application concrète aux machines à états UML[2].

5. Bibliographie

Ballagny C., Hameurlain N., Barbier F., « MOCAS : A State-Based Component Model for Self-Adaptation », *IEEE SASO '09*, IEEE Computer Society, 2009.

Beugnard A., Jézéquel J.-M., Plouzeau N., Watkins D., « Making Components Contract Aware », *IEEE Computer*, vol. 32, n° 7, p. 38-45, 1999.

Cariou E., Ballagny C., Feugas A., Barbier F., « Contracts for Model Execution Verification », *ECMFA '11*, vol. 6698 of *LNCS*, Springer, p. 3-18, 2011.

Cariou E., Belloir N., Barbier F., Djemam N., « OCL Contracts for the Verification of Model Transformations », *OCL Workshop at MoDELS '09*, vol. 24, EC-EASST, 2009.

Combemale B., Crégut X., Garoche P.-L., Xavier T., « Essay on Semantics Definition in MDE – An Instrumented Approach for Model Verification », *Journal of Software*, vol. 4, n° 9, p. 943-958, 2009.

Meyer B., « Applying "Design by Contract" », *IEEE Computer (Special Issue on Inheritance & Classification)*, vol. 25, n° 10, p. 40-52, 1992.

2. Cette expérimentation, utilisant notre moteur MOCAS d'exécution de machines à états UML (`http ://mocasengine.sourceforge.net/`), ainsi qu'un DSL de machines à états simplifiées et ses contrats sont disponibles en ligne :
`http ://web.univ-pau.fr/%7Ecariou/contracts/`