Comprendre la nature exécutable des modèles

Eric Cariou, Olivier Le Goaer, and Franck Barbier

Université de Pau et des Pays de l'Adour / LIUPPA, France
prenom.nom@univ-pau.fr

Résumé

En ingénierie dirigée par les modèles (IDM), le concept de « i-DSML » (*interpreted Domain Specific Modeling Language*) fait référence à des modèles exécutables qui sont interprétés par un moteur d'exécution. Tandis que de nombreux travaux ont techniquement discuté des éléments constitutifs d'un i-DSML, peu d'entre eux se sont attachés à répondre à la question originelle : quels sont les modèles qui sont exécutables par nature et ceux qui ne le sont pas ? Dans cet article, nous proposons d'y répondre en établissant deux critères discriminants. À la lumière de ces critères, nous reconsidérons quelques DSML bien connus.

Mot-clés : IDM, exécution de modèles, i-DSML

Abstract

Within the model-driven engineering field (MDE), the concept of "i-DSML" (interpreted Domain Specific Modeling Language) refers to executable models which are interpreted through an engine. While several works discussed the key ingredients of a i-DSML, few of them answered the original question : what is the class of models that are executable by nature and those that are not ? In this article, we try to answer it by proposing two discriminating criteria. Based on the latters, we reconsider some well-known DSML.

Keywords : MDE, model execution, i-DSML

1 Introduction

L'ingénierie dirigée par les modèles (IDM) a vocation à rendre les modèles productifs pour le développement de logiciels. Dans ce contexte, on peut définir des langages de modélisation dédiés à une classe de problèmes avec les DSML (*Domain Specific Modeling Languages*). Certains DSML peuvent également définir des modèles particuliers dans le sens où ils sont exécutables via un moteur. On parle alors de i-DSML [5] (*interpreted-DSML*). L'exécution n'est donc plus l'apanage exclusif des langages de programmation et concerne aussi les langages de modélisation.

L'exécution de modèles a été étudiée par de nombreux travaux qui ont établi un consensus sur les éléments constitutifs d'un i-DSML et donc sur comment réaliser de l'exécution de modèles. Cela permet de répondre à la question « comment définir un i-DSML ». Dans cet article, nous tentons de prendre le problème dans un autre sens en répondant à la question « comment savoir si un modèle est exécutable » ou, dit autrement, de déterminer si un DSML est ou non un i-DSML.

Dans la section suivante, nous rappelons les éléments constitutifs et associés à un i-DSML. Dans la section 3 nous donnons deux critères permettant de déterminer si un modèle donné est exécutable et les appliquons sur quelques DSML connus.

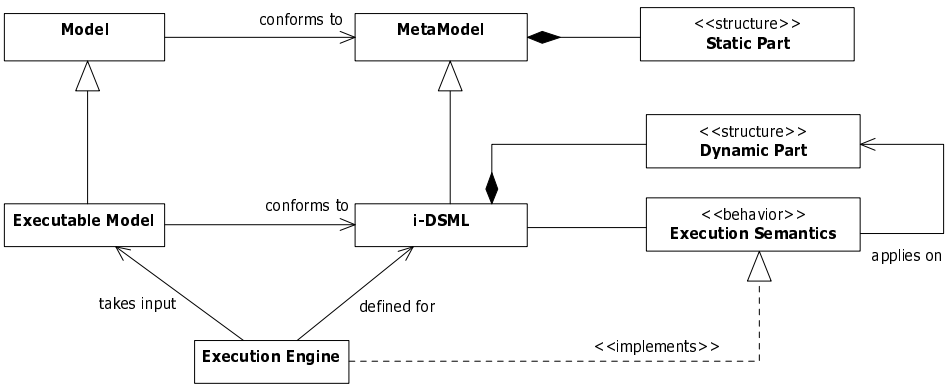

FIGURE 1 – Cadre conceptuel définissant l'exécution de modèles

2 Caractérisation de l'exécution de modèles

L'exécution de modèles a été étudiée et caractérisée par plusieurs travaux, notamment [2, 3, 4, 5, 6, 7]. Ces travaux établissent un consensus sur ce qu'est l'exécution de modèles et comment définir un i-DSML. La figure 1 reprend notre caractérisation de [4]. Un i-DSML est un DSML particulier pour lequel le méta-modèle contient deux types d'éléments : les éléments dits statiques qui décrivent le contenu statique du modèle et les éléments dynamiques qui permettent de préciser dans quel état est le modèle à un pas d'exécution donné. Pour une machine à états par exemple, la partie statique définit les états et les transitions de la machine alors que la partie dynamique précise quels sont les états actifs courants. Une sémantique d'exécution est associée à un i-DSML. Elle précise comment faire évoluer le modèle au fil du temps pendant son exécution en modifiant les éléments de la partie dynamique du modèle. Pour une machine à états, elle précise comment les transitions sont suivies par rapport aux états actifs courants et à un événement généré. Cette sémantique d'exécution est implémentée par un moteur d'exécution. Il prend en entrée un modèle exécutable (conforme par principe à un i-DSML) et a la charge d'exécuter ou interpréter ce modèle, c'est-à-dire de le faire évoluer, générant ainsi une série de pas d'exécution.

Notons que cette caractérisation inclut le fait que l'état courant du modèle est stocké dans le modèle lui-même. Cela n'est pas forcément nécessaire, l'information sur l'état courant du modèle pouvant être gérée directement par le moteur d'exécution. Par exemple, PauWare [1] est un moteur d'exécution de machines à états UML en Java. La spécification UML n'a pas prévu de mémoriser dans le modèle quels sont les états actifs à un pas d'exécution donné. C'est donc ici le moteur PauWare qui gère la liste de ces états actifs. Disposer de l'état courant dans le modèle offre néanmoins l'avantage de pouvoir enregistrer une trace d'exécution complète du modèle pendant son exécution en disposant de toute l'information sur l'état du modèle à chaque pas d'exécution. On pourra par exemple grâce à cela réaliser une reprise sur panne ou une vérification sur la trace obtenue.

Nous pouvons noter au passage un paradoxe autour d'UML. La spécification UML n'a pas prévu à la base de pouvoir exécuter les différents types de diagrammes, notamment les comportementaux qui sont pourtant a priori exécutables. Ainsi, pour les machines à états par exemple, il n'existe pas de partie dynamique dans le méta-modèle UML (dans [3] nous avons

1. `http://www.pauware.com`

proposé une extension dans ce but). Par contre, le diagramme de classes, non exécutable lui, est étonnement livré avec sa partie dynamique : le diagramme d'objets. En effet, bien que ce diagramme n'ait pas été prévu pour de l'exécution de modèles, il peut néanmoins servir à représenter un état courant à un moment donné pour les instances d'un diagramme de classes. D'ailleurs, on retrouve ces idées d'état et de temps dans la présentation des diagrammes d'objets dans la spécification UML (1.3 et supérieure) de l'OMG [2] : "A static object diagram is an instance of a class diagram ; it shows a snapshot of the detailed state of a system at a point in time".

3 Caractérisation de la nature exécutable d'un modèle

La caractérisation de l'exécution donnée ci-avant ne permet pas de savoir explicitement quels sont les modèles qui peuvent être exécutés et ceux qui ne le peuvent pas. Cela relève d'une réflexion implicite et empirique de la part de l'ingénieur logiciel. Nous allons ici détailler deux critères qui nous semblent caractériser la nature exécutable d'un modèle :

— Le comportement du système est localisé dans le modèle ;
— Il est possible de déterminer le pas suivant et un état initial.

Ces deux critères permettent en conséquence de savoir si on peut définir une sémantique d'exécution et un moteur l'implémentant pour un DSML donné. En d'autres termes de savoir si ce DSML est un i-DSML.

3.1 Localisation du comportement du système

Supposons un système logiciel quelconque qui utilise un modèle durant son exécution (c'est-à-dire au *runtime*). La question est de savoir quel est le rôle de ce modèle par rapport au fonctionnement ou au comportement du système. Un système réalise des actions « métier ». Par exemple, un ascenseur ouvre et ferme sa porte, enroule ou déroule son câble pour atteindre un certain étage tandis qu'un système de réservation de voyage en ligne insère des données clients en base de données ou appelle des Web services de compagnies de transport. Le comportement du système consiste à déterminer quand exécuter ces actions et selon quelles conditions.

Dans ce contexte, la question est de savoir si les décisions de réaliser ces actions sont dictées par le modèle lui-même ou bien si elles proviennent d'ailleurs et auquel cas le modèle est considéré comme de simples données d'entrée aidant à prendre les décisions. Par exemple, les actions de l'ascenseur peuvent être déclenchées par rapport aux transitions et états d'une machine à états ou les appels des Web services sont déclenchés selon une orchestration BPEL. Dans ces deux cas, le comportement du système est pleinement défini par le modèle. A contrario, dans l'esprit des *models@run.time* [1], on pourrait disposer à l'exécution d'un modèle représentant pour un ascenseur le niveau d'usure des pièces, le nombre d'usages journalier, etc. Ce modèle sera modifié et/ou interrogé par les actions métier mais ne servira pas à les déclencher. Dans ce cas, il ne s'agit pas d'un modèle exécutable.

En conséquence, si un modèle définit le comportement du système, alors le système logiciel le prenant en entrée et le manipulant peut être qualifié de moteur d'exécution.

3.2 Des pas d'exécution déterminables

Comme vu dans la section 2, l'exécution de modèles consiste à réaliser des pas d'exécution ; un pas faisant passer le modèle d'un état à un autre. Il faut donc qu'intrinsèquement le modèle

2. `http://doc.omg.org/formal/2000-03-01.pdf`, section 3-20, page 278

permette d'identifier des états différents. Mais être capable de définir l'état courant d'un modèle ne suffit pas bien qu'étant nécessaire. Nous avons noté par exemple que le diagramme d'objets peut être vu comme la partie dynamique d'un diagramme de classes, mais cela ne suffit pas à rendre le diagramme de classes exécutable pour autant car il faut en plus savoir comment modifier le diagramme d'objets au fil du temps. Or cela, le diagramme de classes ne permet pas de le déterminer (sauf à lui associer des diagrammes comportementaux (séquence, machines à états, etc.) et/ou le contenu des méthodes des classes avec fUML [3] par exemple, mais dans ce cas, ce n'est pas le diagramme de classes seul qui est considéré).

Par conséquent, le second critère de la nature exécutable des modèles est donc que l'on puisse réaliser des pas d'exécution. On retrouvera cette caractéristique par exemple pour un diagramme d'activité, une machine à états ou un réseau de Petri. Dans ces modèles, il existe des éléments de modélisation dédiés à cela (typiquement des transitions à suivre entre les états, activités ou places). Dans d'autres cas, le ≪ calcul ≫ du prochain pas se fait sans élément dédié comme pour un modèle SBVR [4] (*Semantics of Business Vocabulary and Business Rules*) où le moteur parcourt l'ensemble des règles pour déterminer celles à suivre, comme pour tout langage déclaratif.

Il existe un pas particulier, celui qui démarre l'exécution du modèle en le plaçant dans son état initial. Il faut donc que le modèle définisse un point de départ. On peut se permettre une analogie avec les langages de programmation qui exigent la définition d'une opération particulière *main()* qui dit par où commencer l'exécution du programme. Dans une machine à états, un diagramme d'activité ou bien encore un réseau de Petri, ce point de départ est explicite car un ou plusieurs éléments de modélisation sont dédiés à cela. Avoir un point d'arrivée est optionnel car certaines exécutions n'ont pas vocation à se terminer un jour.

La possibilité de pouvoir faire passer le modèle d'un état à un autre est une caractéristique fondamentale puisque sans cela, il n'est pas possible de définir une sémantique d'exécution. La sémantique d'exécution ayant en effet pour but de faire évoluer le modèle.

3.3 Exemples de quelques DSML

Langage de modélisation	**Comportement du système**	**État courant**	**Pas d' exécution**	**Exécutable ?**
BPEL/BPMN	Oui	Externe	Explicite	Oui
Cas d'utilisation UML	Oui	Interne	Aucun	Non
Diagramme de classes UML	Non	Interne	Aucun	Non
Diagramme de composants UML	Non	Aucun	Aucun	Non
Machines à états UML	Oui	Externe	Explicite	Oui
SBVR	Oui	Externe	Implicite	Oui
Diagramme de séquence UML	Oui	Externe	Explicite	Oui

Table 1 – Caractérisation de la nature exécutable de quelques DSML connus

La table 1 applique nos critères sur quelques DSML connus dont certains ont déjà été évoqués précédemment et en nous basant sur leurs spécifications par l'OMG (on considère un type de diagramme UML comme un DSML ici). La première constatation est déjà que pour les modèles qui sont exécutables (machines à états, diagramme de séquence, BPMN, ...) jamais le méta-modèle n'inclut de partie dynamique. La gestion de l'état courant est donc externe et laissée

3. `http://www.omg.org/spec/FUML/`
4. `http://www.omg.org/spec/SBVR/`

à la charge du moteur d'exécution. Paradoxalement, des modèles non exécutables comme le diagramme de classes ou les cas d'utilisation ont eux une partie dynamique définie (on retrouve dans le méta-modèle UML le concept d'instance de classes ou d'instance de cas d'utilisation). Le diagramme de cas d'utilisation est particulier dans le sens où il est généralement considéré comme étant exécutable car étant un diagramme de comportement. Néanmoins, la description des cas d'utilisation est très informelle et il n'y a pas de possibilité de savoir comment déclencher les cas ni comment les exécuter concrètement. La notion de pas d'exécution ne fait pas sens ici, les cas d'utilisation ne sont donc pas exécutables.

4 Conclusion

Dans cet article, nous avons commencé par rappeler ce qu'est un i-DSML sous un angle technique (partie dynamique, sémantique d'exécution, moteur, etc.). Puis, nous nous sommes interrogés, d'un point de vue cette fois-ci plus fondamental, sur la nature exécutable d'un modèle. Nous avons dégagé deux critères qui permettent de déterminer si un modèle peut être exécutable, c'est-à-dire si son DSML peut être considéré comme un i-DSML. Ces deux critères sont nécessaires mais pas forcément suffisants. Il s'agit de pouvoir réaliser des pas d'exécution pour un modèle (incluant un pas initial) ainsi que de placer le « cœur » du système (son comportement) dans le modèle. Ces deux caractéristiques sont requises pour pouvoir définir une sémantique d'exécution et le moteur qui l'implémente pour un i-DSML donné. Enfin, nous avons appliqué ces critères sur quelques DSML connus.

Cette caractérisation de la nature exécutable d'un modèle n'est peut-être pas exhaustive mais ces deux critères sont de notre point de vue les principaux. Nous allons par la suite étudier plus en profondeur un ensemble d'i-DSML pour affiner cette caractérisation.

Références

[1] Gordon S. Blair, Nelly Bencomo, and Robert B. France. Models@run.time. *IEEE Computer*, 42(10) :22–27, 2009.

[2] Erwan Breton and Jean Bézivin. Towards an understanding of model executability. In *Proceedings of the international conference on Formal Ontology in Information Systems (FOIS '01)*. ACM, 2001.

[3] Eric Cariou, Cyril Ballagny, Alexandre Feugas, and Franck Barbier. Contracts for Model Execution Verification. In *Seventh European Conference on Modelling Foundations and Applications (ECMFA 2011)*, volume 6698 of *LNCS*, pages 3–18. Springer, 2011.

[4] Eric Cariou, Olivier Le Goaer, Franck Barbier, and Samson Pierre. Characterization of Adaptable Interpreted-DSML. In *9th European Conference on Modelling Foundations and Applications (ECMFA 2013)*, volume 7949 of *LNCS*, pages 37–53. Springer, 2013.

[5] Peter J. Clarke, Yali Wu, Andrew A. Allen, Frank Hernandez, Mark Allison, and Robert France. *Formal and Practical Aspects of Domain-Specific Languages : Recent Developments*, chapter 9 : Towards Dynamic Semantics for Synthesizing Interpreted DSMLs. IGI Global, 2013.

[6] Benoit Combemale, Xavier Crégut, and Marc Pantel. A Design Pattern to Build Executable DSMLs and associated V&V tools. In *The 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*. IEEE, 2012.

[7] Grzegorz Lehmann, Marco Blumendorf, Frank Trollmann, and Sahin Albayrak. Meta-Modeling Runtime Models. In *Models@run.time Workshop at MoDELS 2010*, volume 6627 of *LNCS*. Springer, 2010.