Ingénierie basée sur les modèles : quelques défis actuels

FABRICE KORDON

Résumé : L'ingénierie dirigée par les modèles (IDM ou MDE pour *Model-Driven Engineering*) est une approche de plus en plus présente dans le monde industriel. L'objectif est de centrer le développement de systèmes sur des modèles qui accompagnent leur conception et servent de base à leur développement. Il d'agit donc d'une phase de mise en place systématique de processus de fabrication logicielle instrumentés en vue de mieux maîtriser la fiabilité des systèmes ainsi produits.

À l'issue d'une longue lignée de travaux, l'IDM a établi des techniques permettant : 1) de définir des langages ou notations semi-formelles au moyen de méta-modèles ; 2) d'exprimer des règles de transformations entre méta-modèles ; 3) d'exprimer des contraintes de cohérence sur des modèles ; 4) d'exprimer des procédés de fabrication. Ces techniques atteignent une maturité à travers des implémentations reconnues, par exemple au sein de l'environnement Eclipse. Cependant, de nouveaux défis sont en train d'apparaître et deux d'entre eux sont mis en perspective ici : la description et la manipulation de modèles formels, d'une part, la prise en compte des méthodes formelles dans un processus de développement, d'autre part.

Mots-clefs : ingénierie dirigée par les modèles, méthodes de développement, méthodes formelles

1. INTRODUCTION

Le développement logiciel a, ces dernières années, subi de profondes mutations avec l'arrivée d'une nouvelle discipline informatique : l'ingénierie dirigée par les modèles. Cela confirme l'importance que prennent les modèles sur les programmes (même si cela semble moins avancé Outre-Atlantique).

Lorsqu'ils sont bien construits avec des notations à la sémantique rigoureusement définie, les modèles offrent l'avantage de permettre une forme de raisonnement plus poussée que la simple exécution ou simulation. On peut en extraire des informations structurelles ou comportementales, éventuellement à l'aide de modèles mathématiques. Ces opérations sont plus difficiles à réaliser sur des programmes, ce qu'illustre bien la thématique du « program checking » qui utilise des représentations intermédiaires formelles — des modèles — comme base pour calculer les propriétés recherchées.

Le développement récent de cette discipline consacre donc une longue tradition de travaux sur l'usage de modèles dans le développement logiciel et matériel. L'objectif est d'en retracer ici des étapes importantes avant de poser quelques défis qui s'ouvrent à la communauté.

2. UNE VISION HISTORIQUE DE L'INGÉNIERIE DIRIGÉE PAR LES MODÈLES

L'ingénierie dirigée par les modèles n'est pas une idée neuve, même si pendant longtemps, elle a pris des formes utilisant des modèles sans les mettre explicitement en avant. On parlait en particulier de « prototypage » [12] ; ce terme est d'ailleurs resté longtemps en vigueur dans certaines communautés (par exemple, dans le développement de systèmes matériels).

L'approche développée dans cette section se veut historique mais en aucun cas exhaustive, ce qui semble impossible au vu du très grand nombre de travaux dans le domaine. L'idée est d'identifier quelques étapes clefs correspondant à différentes visions de ce qui est devenu l'ingénierie des modèles

2.1 PREMIERS PAS

Le couple lex/yacc [9] peut-être vu comme un outil d'aide au développement de compilateurs basé sur le modèle de Backus Naur pour exprimer un langage informatique. Dans ce cas, le

modèle est utilisé uniquement pour générer du code au sein duquel s'insèrent des actions décrites par l'utilisateur. Cependant, des versions plus récentes de compilateurs de compilateurs comme ANTLR [13] en déduisent la gestion des erreurs de syntaxe. Cette lignée d'outils toujours utilisés constitue un premier niveau d'aide au développement logiciel au moyen d'une abstraction dédiée à un domaine (DSL — Domain Specific Language — avant l'heure).

Nous pouvons également citer HOOD (Hierarchic Object-Oriented Design), beaucoup plus proche du développement, qui est utilisé par l'Agence Spatiale Européenne dès la fin des années 1980. L'objectif est de regrouper en un seul « modèle » graphique la structure d'une application et les annotations qui en constituent le code métier. Des générateurs peuvent produire les sources de l'application finale en insérant le code métier au sein de la partie automatiquement déduite de la structure de la spécification orientée objet. On accède donc à une sorte de programmation mi-visuelle, mi-textuelle, qui permet de mieux appréhender les systèmes complexes. Des outils d'analyse ont également été expérimentés pour effectuer certaines analyses [11]. Plus de vingt ans après sa création, HOOD est toujours utilisé dans certains domaines et des Ateliers de Génie Logiciel supportant la méthode associée sont toujours commercialisés.

2.2 Quelques exemples précurseurs

Citons comme premier précurseur l'outil SCADE, opérationnel dès 1994 après environ dix années de recherches. Cet outil prend en compte trois dimensions importantes qui sont aussi des bases de l'ingénierie des modèles : la modélisation, la génération de programmes sur différentes cibles ainsi que des capacités d'analyse, possible grâce à la notation choisie qui repose sur des bases mathématiques [3]. Plus de quinze ans après, la société qui commercialise SCADE, Esterel Technologies, est mondialement connue.

On peut aussi citer bien d'autres travaux émergeant dès les années 1980, par exemple autour des réseaux de Petri qui permettent également une analyse comportementale et la génération de code [5, 7, 15]. Cette dernière n'est cependant pas suffisamment efficace, car les réseaux de Petri n'offrent que des mécanismes de bas niveau : des entités plus complexes (file de messages, piles, listes, etc.) ne sont pas identifiables et il est difficile de leur associer des patrons de génération de code.

Enfin, d'autres outils sont expérimentés ponctuellement dans des projets d'envergure comme Parallel PROTO [4] qui fit l'objet d'un contrat du département de la défense américaine avec les Rome Laboratories. L'objectif était de développer un modèle du système d'information de la défense américaine afin d'en étudier la robustesse. Les modèles étaient simulables mais aucune génération automatique de programme ne semble avoir été à l'ordre du jour.

2.3 Problématiques de l'ingénierie dirigée par les modèles

Comme nous l'avons vu, les éléments clefs de l'ingénierie des modèles sont en gestation depuis longtemps. Les travaux dans la lignée de ceux que nous avons cités ont progressivement mis en avant l'importance de l'architecture du système [10], qu'elle soit logicielle, matérielle ou qu'elle comporte les deux aspects. De là, une nouvelle notion émerge : l'abstraction de l'architecture d'exécution. Ainsi, on obtient des modèles dont la sémantique est neutre (PIM pour *platform-independent* model) et d'autres qui s'appuient sur un environnement d'exécution (PSM pour *platform-specific* model). Ces deux niveaux de conception permettent d'ébaucher une méthode. L'idée est de commencer par les aspects propres du système en se focalisant sur les fonctions qu'il doit réaliser (niveau PIM) avant de se concentrer sur son déploiement dans un environnement d'exécution donné, voire d'en explorer plusieurs (niveau PSM) [2].

Mais parallèlement à ces aspects liés au développement, l'ingénierie dirigée par les modèles a élaboré un certain nombre de techniques nécessaires pour faciliter l'élaboration de passerelles entre modèles [2]. La notion de méta-modèle, *framework* pour la définition de nouveaux langages et notations, est une première étape nécessaire. L'expression de règles de transformation entre méta-modèles permet de définir des correspondances entre les notions offertes par deux méta-modèles. Cela permet de produire à coût raisonnable des outils de transformation permettant de projeter une spécification dans une autre notation (par exemple, un modèle UML vers du code). Enfin, plus récemment, il est devenu important d'exprimer des procédés de fabrication pour construire des méthodes en associant des notations (méta-modèles) et des outils à une « façon de procéder » dans le développement de systèmes complexes [1].

3. De nouveaux défis

On trouve donc actuellement dans l'ingénierie des modèles deux aspects complémentaires. Le premier, plus technique, permet de décrire et de manipuler des modèles, de les échanger et de les projeter sur d'autres notations. Le second est d'ordre méthodologique et vise à insérer les techniques ainsi produites dans un processus de développement qu'il faut lui-même décrire (à l'aide de modèles par exemple).

Parallèlement, on observe une émergence, lente mais inexorable, de l'usage des méthodes

formelles qui sortent progressivement du marché de niche où elles étaient cantonnées. Cela est dû à la nature de plus en plus complexe et de plus en plus critique des systèmes que nous développons et à la maturité des techniques et outils qui permettent désormais de traiter des problèmes industriels comme un intergiciel [15]. La taille de certaines applications les lie aussi à un nombre d'utilisateurs important que des effets inattendus et potentiellement indésirables ne doivent pas inquiéter, sous peine de les en éloigner. Dans ce cadre, les méthodes formelles, naturellement basées sur des modèles, et l'ingénierie des modèles devront tôt ou tard s'intégrer. Cela engendre deux défis.

3.1 Premier défi : manipuler des concepts formels

Le premier défi se rapporte aux aspects plus techniques de l'ingénierie des modèles. Il concerne la gestion d'une sémantique formelle. Pour certaines classes d'applications critiques, ces techniques sont déjà mises en œuvre dans un cadre de développement dirigé par les modèles. Cependant, les modèles formels ont des sémantiques basées sur des notions mathématiques qu'il est difficile de capturer sous la forme de méta-modèles focalisés sur les aspects syntaxiques. De plus, certaines caractéristiques de ces modèles formels sont difficiles à capturer de manière purement structurelle et requièrent l'usage encore mal maîtrisé de langages de contraintes comme OCL [12]. Enfin, l'usage de mécanismes d'héritage n'est pas du tout trivial dans ce cadre précis.

Des travaux dans ce sens ont été effectués pour le standard ISO/IEC pour les réseaux de Petri et proposent une approche intéressante dans le cadre d'une sémantique rendue délicate par l'existence de différentes variations de cette notation formelle [6]. Ces travaux sont décrits dans l'article de L. Hillah et L. Petrucci : « Standardisation des réseaux de Petri : état de l'art et enjeux futurs » figurant dans le présent numéro.

3.2 Deuxième défi : intégration des méthodes formelles dans une approche IDM

Le second défi est d'ordre méthodologique. Il concerne la mise en œuvre de méthodes formelles dans le cadre du développement de systèmes. Ce point est particulièrement délicat lorsqu'il faut raisonner sur les modèles tout au long du processus de conception et de fabrication. Pour le maîtriser, il faut à la fois disposer d'un langage pivot adéquat servant de base à la modélisation et pouvant être projeté sur différentes notations formelles à des fins d'analyse.

L'une des difficultés réside dans le nombre élevé de résultats théoriques applicables dans un cadre donné seulement et performant dans des conditions précises. Il faut savoir comment et quand orchestrer ces différents algorithmes et modèles dans un processus qui serait alors dirigé non plus par les seuls modèles mais par les aspects liés à la vérification : [8] parle de VDE pour *verification-driven engineering*.

Une expérience intéressante dans ce cadre a été menée avec AADL [14] dans le domaine des systèmes embarqués, souvent critiques de par leur nature. AADL est au départ un langage de description d'architecture. Des mécanismes (attributs, annexes linguistiques) permettant de préciser des caractéristiques structurelles ou comportementales sont connectés sur cette architecture pour préciser la sémantique de ses composantes. En ce qui concerne la connexion avec des méthodes d'analyse formelle et de génération automatique de code, AADL est bien plus précis que ne peut l'être UML mais son domaine d'application est en revanche moins large. Ces travaux sont décrits dans l'article de X. Renault et J. Hugues : « Définition d'une famille de patrons de transformation pour l'analyse de modèles AADL » figurant dans le présent numéro.

4. Références

[1] R. Bendraou, J.-M. Jézéquel, M.-P. Gervais et X. Blanc :*A comparison of six UML-based languages for software process modeling* ; in IEEE Transactions on Software Engineering, à paraître

[2] X. Blanc : *MDA en action* ; Eyrolles, 2005

[3] A. Bouali, J.-P. Marmorat, R. de Simone et H. Toma : *Verifying synchronous reactive systems programmed in ESTEREL* ; in Conference on Formal Techniques in Real-Time and Fault-Tolerant Systems, pp 463-466, 1996

[4] C. Burns : *Parallel PROTO - A prototyping tool for analyzing and validating sequential and parallel processing software requirements ;* 2nd International Workshop on Rapid System Prototyping, Raleigh-Durham, États-Unis, juin 1991

[5] J.M. Colom, M. Silva et J.L. Villarroel : *On software implementation of Petri Nets and colored Petri Nets using high-level concurrent languages* ; 7th Workshop on Application and Theory of Petri Nets, pp. 207, 1986.

[6] L. Hillah, E. Kindler, F. Kordon, L. Petrucci et N. Tre?ves : *A primer on the Petri Net Markup Language and ISO/IEC 15909-2 »* ; in Petri Net Newsletter (présenté initialement au 10th International Workshop on Practical Use of Colored Petri Nets and the CPN Tools – CPN'09), 76, pp. 9–28, octobre 2009)

[7] F. Kordon : *Prototypage de systèmes parallèles à partir de réseaux de Petri colorés, application au langage Ada dans un environnement centralisé ou reparti* ; Thèse de l'Université Pierre et Marie Curie (Paris 6), mai 1992

[8] F. Kordon, J. Hugues et X. Renault : *From model-driven engineering to verification-driven engineering* ; in 6th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2008), volume 5287 of Lecture Notes in Computer Science, pages 381–393, Capri, Italie, octobre 2008, Springer-Verlag

[9] J. Levine, T. Mason et D. Brown : *Lex & Yacc*, 2e édition ; O'Reilly & Associates, 1992.

[10] N. Medvidovic et R. Taylor : *A classification and comparison framework for software architecture description languages* ; IEEE Transactions on Software Engineering, 26, pp.70–93, 2000.

[11] M. Paludetto : *Sur la commande de procédés industriels : une méthodologie basée objets et réseaux de Petri* ; Thèse de l'Université Paul Sabatier, Toulouse, décembre 1991

[12] OMG : *« Object Constraint Language Specification, Version 2.0* », http://www.omg.org/technology/documents/formal/ocl.htm

[13] T. Parr : *The Definitive ANTLR Reference: Building Domain-Specific Languages* ; The Pragmatic Programmers, 2007

[14] SAE : *Architecture Analysis & Design Language v2.0 (AS5506)* ; septembre 2008

[15] Y. Thierry-Mieg, S. Baarir, A. Duret-Lutz et F. Kordon : *Nouvelles techniques de model-checking pour la vérification de systèmes complexes* ; Génie Logiciel, n°69, pages 17-23, juin 2004

[16] R. Valette, M. Courvoisier, J.-M. Bigou et J. Alburkerque : *A Petri net-based programmable logic controller* ; 1st International Conference on Computer Application in Production and Engineering, Amsterdam, avril 1983

[17] R. Vonk : *Prototypage : l'utilisation efficace de la technologie CASE* ; Masson & Prentice Hall, Paris, 1992

165 mm

Standardisation des réseaux de Petri : état de l'art et enjeux futurs

LOM M. HILLAH ET LAURE PETRUCCI

Résumé : La communauté travaillant sur les réseaux de Petri a constaté relativement tôt le manque d'homogénéité dans les définitions formelles et les outils. En effet, il existe une grande variété de types de réseaux de Petri, avec des différences parfois significatives (par exemple les réseaux temporels) ou de simples extensions (les réseaux avec arcs inhibiteurs). De plus, chaque outil utilise sa propre version de réseau de Petri et implémente des techniques d'analyse qui lui sont propres. La vérification de systèmes modélisés avec des réseaux de Petri nécessite l'utilisation de diverses techniques de vérification, donc des outils associés différents. Pour que cette vérification soit réalisable dans la pratique, la conception d'un format d'échange de réseaux de Petri entre ces outils est un enjeu majeur.

La norme ISO/IEC-1590 présentée dans cet article a pour objectif de répondre à cette problématique. Elle est structurée en trois parties. La première partie concerne les définitions du formalisme, donnant ainsi une sémantique formelle à plusieurs types usuels de réseaux de Petri : les réseaux places/transitions et les réseaux de haut niveau (colorés). Cette partie a été publiée comme norme internationale en décembre 2004. Un amendement définissant les réseaux symétriques devrait être bientôt publié. La seconde partie, parue en novembre 2009, se concentre sur la syntaxe et l'élaboration d'un langage d'échange de réseaux de Petri. Ce langage, PNML (Petri Net Markup Language) s'appuie sur les concepts introduits dans la première partie de la norme. Sa conception repose sur des métamodèles UML, puis une transformation en XML.Un des enjeux majeurs de la seconde partie a été la flexibilité et l'extensibilité du langage PNML, en vue des travaux sur la troisième partie de la norme. Cette partie, dont le développement vient de démarrer, porte sur les extensions de réseaux de Petri. Ces dernières concernent la structuration des modèles (modularité et hiérarchie) et les mécanismes nécessaires à l'intégration de nouveaux types de réseaux de Petri et à l'ajout de nouveaux éléments (tels que les arcs inhibiteurs).

Mots-clés : PNML, normalisation, réseaux de Petri, métamodélisation, UML, MDE.

1. INTRODUCTION

Les réseaux de Petri (ci-après abrégés par RdP) constituent une famille de formalismes mathématiquement fondés, adaptés à la modélisation de systèmes parallèles et asynchrones [1]. Ils permettent l'analyse de différents aspects comportementaux d'un système (discret, stochastique, temporisé), soit sur des questions relatives à l'expressivité, soit sous l'angle de la décidabilité pour résoudre des problèmes de vérification [2]. Il en existe de ce fait plusieurs types, dont les sémantiques d'exécution diffèrent parfois de manière significative (par exemple, réseaux temporels opposés aux réseaux places/transitions ordinaires), ou représentent de simples extensions d'autres types (réseaux places/transitions avec arcs inhibiteurs).

L'écosystème des réseaux de Petri pourrait donc être caractérisé par les volets suivants :

- les grandes familles à sémantique discrète, à temps continu ou stochastique,
- les types au sein de chaque famille, ainsi que leurs extensions,