Une approche de méta-modélisation formelle des méthodes de conception des systèmes automatisés de production

Laurent Piétrac, Bruno Denis

Journées Doctorales d'Automatique, JDA'99, Nancy (France), pp. 285-288, sept. 1999

Une approche de méta-modélisation formelle des méthodes de conception des systèmes automatisés de production

Laurent Piétrac[1] et Bruno Denis[2]

[1] LAI, INSA de Lyon, bat. 303, 20 av. Albert Einstein 69621 VILLEURBANNE CEDEX

e-mail : Laurent.Pietrac@lai.insa-lyon.fr

[2] LURPA, ENS de Cachan, 61 av. du président Wilson 94235 CACHAN CEDEX

e-mail : Denis@lurpa.ens-cachan.fr

Résumé

La conception des systèmes automatisés de production (SAP) passe par la construction de modèles. La qualité du système conçu dépend de la qualité des modèles produits, et donc des langages et méthodes utilisés pour construire ces modèles. De nombreux travaux cherchent à améliorer la qualité de ces langages et de ces méthodes grâce à la méta-modélisation. Cependant, tous ces travaux ne s'intéressent qu'à des aspects particuliers des langages et méthodes. Le travail présenté ici [8] a au contraire pour but de définir rigoureusement et complètement les langages et les méthodes de conception. Pour cela les différents aspects à méta-modéliser sont définis, permettant ainsi de choisir un langage fomel, le langage Z, couvrant l'ensemble des besoins. Notre approche est validée sur deux exemples types.

Mots-clés : *langage formel Z, méta-modélisation, validation*

1 Introduction

Les SAP sont des systèmes complexes, dont la qualité et la sûreté de fonctionnnement sont des exigences incontournables. Ces exigences ne peuvent être atteintes que si le processus de conception, et les modèles produits, sont eux-mêmes de qualité.

L'obtention de modèles de qualité nécessite le respect des définitions de **la syntaxe et la sémantique** du **langage** et de la **méthode de construction** utilisés. Ce respect est d'autant plus difficile à obtenir lorsque ce modèle simule le comportement dynamique du système : une définition d'un **joueur de modèles** doit alors être associé au langage. De même, cette qualité ne peut être obtenue que par l'utilisation de **méthodes de vérification** permettant de s'assurer de la cohérence interne des modèles, et de **méthodes de validation** de la cohérence entre le besoin et le modèle produit.

De plus, le processus de conception nécessite la construction de plusieurs modèles, et donc l'utilisation de **méthodes intégrées** multi-langages. Parmi celles-ci la définition de **méthodes d'importation et d'exportation** permet de construire des modèles cohérents entre eux.

Un modèle est donc le résultat de l'utilisation de plusieurs langages et méthodes qui doivent être parfaitement définis pour assurer la qualité des modèles produits.

2 Etat de l'art

Pour améliorer la qualité des modèles de conception, deux approches ont été explorées.

La première consite à définir l'ensemble du travail à réaliser, donc de définir un guide pour la conception de nouveaux systèmes. Le Centre Coopératif de Génie Automatique (CCGA) a ainsi défini une « carte des activités de développement d'un système automatisé de production ». Cette carte précise les résultats attendus dans les différentes étapes nécessaires à la conception d'un SAP, sans pour autant imposer les langages et méthodes à utiliser. Dans cette même voie, CIM-OSA propose unne classification suivant trois axes des modèles à construire : l'axe de génération, l'axe de dérivation et l'axe de particularisation. En se référant à ce « cube », les concepteurs peuvent vérifier que les modèles construits couvrent des aspects différents et complémentaires. Cependant, les langages et méthodes à utiliser ne sont pas non plus imposés. Cette approche permet donc de mieux structurer le travail à réaliser, sans pour autant définir les langages et méthodes utilisés pour la conception des SAP. Elle ne répond donc pas à notre attente.

Afin de mieux définir les langages utilisés et les méthodes intégrant ces langages, une nouvelle approche est apparue : la modélisation des modèles produits. Ces « modèles de modèles » sont souvent appelés méta-modèles. Cette deuxième approche est, à notre avis, la seule permettant de rendre plus rigoureux mais aussi plus évolutifs ces langages et méthodes, et ce, pour deux raisons essentielles. D'une part la construction de méta-modèles oblige à structurer la vision du langage ou de la méthode modélisée. D'autre part, les méta-modèles peuvent être modifiés en fonction des habitudes ou des nouveaux besoins des utilisateurs, tout en obligeant le méta-modélisateur à réfléchir à la conservation de la cohérence du méta-modèle produit.

La plupart des méta-modèles existants ne modélisent qu'une partie des langages et méthodes de conception. Par exemple, les méta-langages les plus fréquemment utilisés en France sont le langage NIAM [6] et les langages de type entité-relation [3]. De part les méta-langages utilisés, les méta-modè-

les produits ne permettent alors que d'exprimer la syntaxe et la sémantique du langage ou de la méthode intégrée [5]. De même, l'utilisation d'équations algébriques [1] ne permet de modéliser que le joueur de modèles.

Les travaux récents de Bon-Biérel [2] ont eu pour objectif de méta-modéliser la syntaxe et la sémantique de langages, ainsi que le joueur de modèles associs. La démarche choisie a été de construire deux méta-modèles avec deux méta-langages différents. Bien que plus d'aspects du langage et des méthodes soient modélisées, nous pensons que cette approche n'est pas satisfaisante. En effet, à travers plusieurs méta-modèles, certains aspects des langages et méthodes peuvent être modélisés plusieurs fois de manière incohérente, ou au contraire ne jamais l'être, entrainant alors des erreurs graves.

Sur le plan international, les travaux les plus avancés en méta-modélisation portent sur la construction de modèles. Les travaux de Soeki [11] portent sur la spécification de la méthode de construction d'un modèle, mais d'un point de vue abstrait. Par contre, Gee [4] psécifie l'aspect concret (visuel) des langages, sans imposer de démarche précise dans la méthode de construction.

3 Le choix du langage Z

Au contraire des approches précédentes, notre objectif est de pouvoir spécifier dans un méta-modèle aussi bien la syntaxe que la sémantique des langages de conception des SAP, ainsi que les méthodes associées à ces langages. De plus nous avons voulu aborder tous ces aspects avec un seul langage, pour éviter tous les problèmes d'intégration entre langages. Nous avons également souhaité valider et vérifer nos méta-modèles. Ces contraintes nous ont poussé à nous tourner vers un langage formel.

Le langage Z [12] a été choisi pour ses capacités à modéliser des données et des modifications de l'état du système modélisé. En outre, ce langage a connu des développements importants et fait l'objet d'une littérature scientifique conséquente. De plus, des logiciels de spécification en Z sont diffusés gratuitement : nous avons par exemple utilisé Z-EVES pour vérifier et valider nos méta-modèles [10].

Les premiers bénéfices de l'utilisation de Z pour la méta-modélisation ont été présentés dans [9]. Pour valider de manière plus exhaustive notre approche, nous l'avons utilisé sur deux exemples plus conséquents : un exemple de langage et un exemple de méthode multi-langages.

4 Méta-modélisation d'un langage

Le langage cible choisi est un langage nous permettant de montrer la supériorité de notre approche sur celles existantes : c'est le réseau de Petri généralisé [7]. En effet, pour définir rigoureusement ce langage, il est nécessaire de méta-modéliser sa syntaxe, sa sémantique mais aussi les méthodes associées de jeu et de construction de modèles : aucune approche actuelle ne permet de s'intéresser, à la foi et avec un seul méta-langage, à tous ces aspects.

La démarche d'élaboration d'un méta-modèle d'un langage est constituée de trois étapes (voir figure 2). La première étape consiste à construire le méta-modèle du langage à partir de sa définition. Ce méta-modèle doit ensuite être vérifié, ce qui constitue la deuxième étape. Enfin, troisième étape, ce méta-modèle doit être validé par rapport au besoin. Ces trois étapes sont ici rapidement présentées.

4.1 Définition

La définition proposée dans [7] est une définition formelle (présentée ci-après), complétée par du texte décrivant notamment les aspects dynamiques du langage.

Un réseau de Petri est un 5-uplet,
PN = (P, T, F, W, M_o) où :
P = $\{p_1, p_2, \ldots, p_n\}$ est un ensemble fini de places,
T = $\{t_1, t_2, \ldots, t_n\}$ est un ens. fini de transitions,
$F \subseteq (P \times T) \cup (T \times P)$ est un ensemble d'arcs
$W : F \to \{1, 2, 3, \ldots\}$ est la fonction poids,
$M_0 : P \to \{1, 2, 3, \ldots\}$ est le marquage initial,
$P \cap T = \emptyset$ et $P \cup T \neq \emptyset$.

4.2 Extrait du méta-modèle

Le méta-modèle décrit d'abord les types utilisés :

$[PLACE, TRANSITION]$

$USE ::= build \mid play$

Le schéma *PN* décrit ensuite les variables utilisées dans un modèle RdP. Certaines des variables utilisées correspondent à des variables issues de la définition formelle du RdP généralisé : *P*, *T*, *arcTP*, *WarcTP* ... D'autres correspondent à des variables, définies dans le texte qui accompagne cette définition formelle, utilisées pour la jeu des modèles : *M*, *enabled*. Enfin la variable *usePN* est utilisée pour définir les opérations de construction, opérations qui ne sont pas décritent dans [7], et qui sont donc définies en fonction de l'usage.

PN

$P : \mathbb{F}\ PLACE$
$T : \mathbb{F}\ TRANSITION$
$arcTP : TRANSITION \leftrightarrow PLACE$
$arcPT : PLACE \leftrightarrow TRANSITION$
$WarcTP : TRANSITION \times PLACE \pfun \mathbb{N}_1$
$WarcPT : PLACE \times TRANSITION \pfun \mathbb{N}_1$
$M0 : PLACE \pfun \mathbb{N}$
$M : PLACE \pfun \mathbb{N}$
$enabled : \mathbb{F}\ TRANSITION$
$usePN : USE$

$\mathrm{dom}(arcTP) \subseteq T \land \mathrm{ran}(arcTP) \subseteq P$
$\mathrm{dom}(arcPT) \subseteq P \land \mathrm{ran}(arcPT) \subseteq T$
$\mathrm{dom}(WarcTP) = arcTP \land \mathrm{dom}(WarcPT) = arcPT$
$\mathrm{dom}(M0) = P \land \mathrm{dom}(M) = P$

Le schéma *Play* décrit l'évolution des variables du schéma *PN* lors du tir d'une transition *Fired*? (exemple d'une méthode de jeu : les variables avec apostrophe correspondent à l'état après opération) :

Play

ΔPN
$Fired? : TRANSITION$

$usePN = play$
$Fired? \in enabled$
$P' = P \land T' = T$
$arcTP' = arcTP \land arcPT' = arcPT$
$WarcTP' = WarcTP \land WarcPT' = WarcPT$
$M0' = M0$
$\forall p : PLACE \mid p \in P \bullet$
$(p \in arcPT^{\sim}(\!| \{Fired?\} |\!) \cap arcTP(\!| \{Fired?\} |\!)$
$\land M'(p) = M(p) - WarcPT(p \mapsto Fired?)$
$+WarcTP(Fired? \mapsto p)) \lor$
$(p \in arcPT^{\sim}(\!| \{Fired?\} |\!) \setminus arcTP(\!| \{Fired?\} |\!)$
$\land M'(p) = M(p) - WarcPT(p \mapsto Fired?)) \lor$
$(p \in arcTP(\!| \{Fired?\} |\!) \setminus arcPT^{\sim}(\!| \{Fired?\} |\!)$
$\land M'(p) = M(p) + WarcTP(Fired? \mapsto p)) \lor$
$(p \notin arcPT^{\sim}(\!| \{Fired?\} |\!) \cup arcTP(\!| \{Fired?\} |\!)$
$\land M'(p) = M(p))$
$enabled' = \{t : TRANSITION \mid (\forall p : PLACE \bullet$
$(p,t) \in arcPT' \land WarcPT'(p,t) < M'(p))$
$\lor t \notin \mathrm{ran}(arcPT') \bullet t\}$
$usePN' = usePN$

4.3 Vérification

Pour vérifier une spécification en langage Z, deux propriétés sont à vérifier (en plus du bon typage des opérations) :

- l'état spécifié ne comporte pas de contradiction. Ceci peut être établi en démontrant le théorème d'initialisation : $\exists PN' \bullet PNinitial$
- les opérations sont totales, c'est à dire quelles sont toujours définies.

Par exemple, le calcul de la précondition de l'opération de jeu donne : $\mathrm{pre}\ Play \mathrel{\widehat{=}} usePN = play \land Fired? \in enabled$

Cette opération n'est pas totale. Pour la compléter, il faut alors créer d'autres schémas permettant de tenir compte des cas non prévus. Un nouveau schéma, équivalent à tous ces schémas, permet donc de tenir compte de tous les cas. Ceci n'est pas montré ici par manque de place, mais a été mené à son terme dans [8].

4.4 Validation

Pour tester nos opérations, nous avons utilisé le logiciel Z-EVES [10].

Pour valider l'opération de jeu du réseau, nous avons jugé nécessaire de disposer d'un modèle RdP généralisé comportant au moins une transition validée, et quatre places, chacune correspondant à un cas différent de calcul de marquage. Ce modèle RdP généralisé minimum est présenté figure 1.

Afin de simplifier la validation, nous avons modifié l'état initial afin qu'il décrive directement le RdP présenté.

Nous avons testé le schéma *Play* en créant un nouveau schéma : $Test2 \mathrel{\widehat{=}} PNinitial \fatsemi StartPlay \fatsemi Play$

Nous avons testé ce schéma avec la valeur $Fired? := t2$. Le résultat obtenu a bien été $enabled' = \{\}$ et $M' = \{(p1, 0), (p2, 0), (p3, 1), (p4, 1)\}$.

Cet exemple, entre autres, nous a permis de valider notre méta-modèle par rapport à notre besoin, et ceci par la simple instanciation des ensembles définis.

PNinitial

PN'
$t1, t2 : TRANSITION$
$p1, p2, p3, p4 : PLACE$

$P' = \{p1, p2, p3, p4\}$
$T' = \{t1, t2\}$
$arcTP' = \{(t1, p2), (t2, p3), (t2, p4)\}$
$arcPT' = \{(p1, t1), (p2, t2), (p4, t2)\}$
$WarcTP' = \{((t1, p2), 1), ((t2, p3), 1), ((t2, p4), 1)\}$
$WarcPT' = \{((p1, t1), 1), ((p2, t2), 1), ((p4, t2), 1)\}$
$M0' = \{(p1, 0), (p2, 1), (p3, 0), (p4, 1)\}$
$M' = \{(p1, 0), (p2, 1), (p3, 0), (p4, 1)\}$
$enabled' = \{\}$
$usePN' = build$

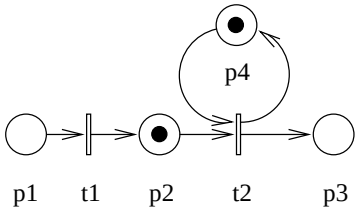

FIG. 1 – *Exemple utilisé de modèle RdP généralisé*

4.5 Conclusion

Notre approche nous a permis d'intégrer dans un seul méta-modèle des aspects aussi différents que la syntaxe et la sémantique du langage, les opérations de jeu et de construction des modèles. De plus, l'utilisation du langage formel Z nous a permis de valider et de vérifier le méta-modèle, et donc de nous assurer de sa rigueur et de sa cohérence.

5 Méta-modélisation d'une méthode

La méta-modélisation d'une méthode multi-langages consiste d'abord à méta-modéliser chacun des langages qui la compose. Pour chaque langage, comme pour l'exemple précédent, il faut construire le méta-modèle du langage et de méthodes associées, puis vérifier et valider ce méta-modèle. Les méta-modèles de chaque langage sont ensuite intégrés au sein du méta-modèle de la méthode. Ceci nécessite l'ajout d'ensembles ou de relations permettant cette intégration. Cela peut aussi nécessiter la modification de certains éléments des méta-modèles originaux. Le méta-modèle obtenu doit également être vérifié et validé.

Dans [8], la méthode multi-langages testée est une méthode intégrant une classe particulière de RdP et des équations différentielles. Ce test nous a permis de confirmer que notre approche permet de méta-modéliser aussi bien un langage, que les méthodes associées et les méthodes intégrées. Notre approche permet donc la méta-modélisation de l'ensemble des langages et méthodes utilisées pour la conception

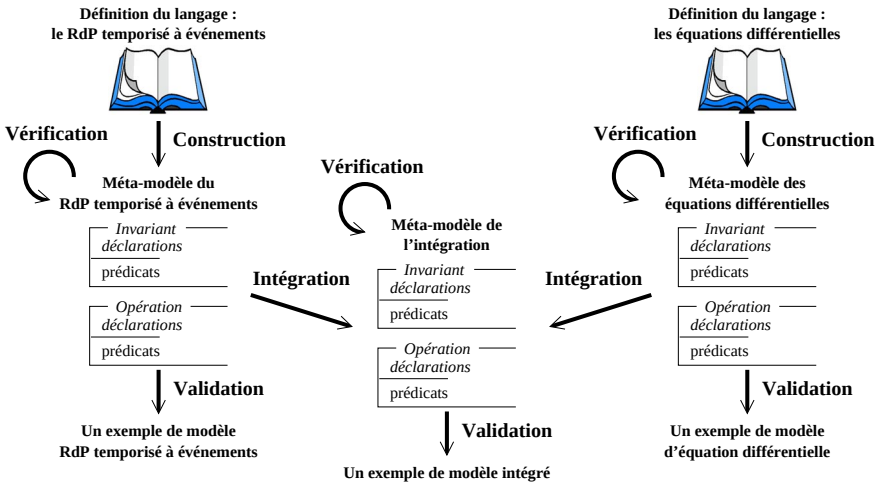

FIG. 2 – *Démarche de méta-modélisation de l'intégration*

des SED. Cet exemple nous permet en outre de valider notre approche sur une méthode de conception de systèmes hybrides, qui sont des systèmes particulièrement ardus à concevoir.

6 Conclusion

Nos travaux ont montré la faisabilité et l'intérêt de la méta-modélisation formelle de l'activité de modélisation des SAP. L'utilisation du langage Z permet de spécifier, dans un même méta-modèle, aussi bien la syntaxe et la sémantique des langages, que les méthodes associées à ces langages. Cette intégration de tous les aspects de l'activité de modélisation garantit la cohérence des méta-modèles, ce que ne permettent pas les autres approches existantes. L'utilisation d'un langage formel comme méta-langage permet de valider et de vérifier les méta-modèles, assurant ainsi de leur rigueur et de leur respect des besoins.

Références

[1] Bon-Bierel (Evelyne). – *Méta-modèles du Grafcet*. – Rapport de recherche, Université de Nancy I - ENS de Cachan, 1994.

[2] Bon-Bierel (Evelyne). – *Contribution à l'intégration des modèles de systèmes de production manufacturière par méta-modélisation*. – Thèse de PhD, Université de Nancy I, 19 novembre 1998.

[3] Couffin (Florent). – *Modèles de données de référence et processus de spécialisation pour l'intégration des activités de conception en Génie Automatique*. – Thèse de PhD, Ecole Normale Supérieure de Cachan, 1997.

[4] Gee (David M.). – Formal specification of visual languages. – 1995. http://lion.unn.ac.uk/ davidg/papers/fspec.ps.Z.

[5] Kiefer (Francois). – *Contribution à l'ingénierie intégrée des systèmes de production : formalisation des mécanismes d'intégration entre modèles et applications sur site industriel*. – Cachan, France, Thèse de PhD, Ecole Normale Supérieure de Cachan, Janvier 1996.

[6] Lhoste (Pascal). – *Contribution au génie automatique : concepts, modèles, méthodes et outils*. – Thèse, Université de Nancy I, 1994.

[7] Murata (Tadao). – Petri nets : Properties, analysis and applications. *Proceedings of the IEEE*, vol. 77, n° 4, 1989.

[8] Piétrac (Laurent). – *Apport de la méta-modélisation formelle pour la conception des systèmes automatisés de production*. – Thèse de PhD, Ecole Normale Supérieure de Cachan, 1999.

[9] Piétrac (Laurent), Denis (Bruno) et Jean-Jacques (Lesage). – Formalization of the design of control systems. *In : Sixth International Symposium on Robotics and Manufacturing (ISRAM'96), Second World Automation Congress (WAC'96)*. – Montpellier, France, 27–30 Mai 1996.

[10] Saaltink (Mark). – *The Z/EVES system*. – Rapport technique, Ottawa, Ontario, Canada, ORA Canada, 1995. ftp://ftp.ora.on.ca/pub/doc/z-eves-draft.ps.Z.

[11] Saeki (Motoshi). – A meta-model for method integration. *Information and software technology*, vol. 39, 1998, pp. 925–932.

[12] Spivey (J. M.). – *La notation Z*. – Masson and Prentice Hall, 1994. traduit de l'anglais par M. Lemoine.