P^{2}E : Une solution outillée dédiée à la gestion des évolutions des profils UML

Fadoi Lakhal, Hubert Dubois
CEA, LIST, 91191 Gif-sur-Yvette Cedex, France
fadoi.lakhal@cea.fr,hubert.dubois@cea.fr

Dominique Rieu
Laboratoire d'informatique de Grenoble
Equipe SIGMA
220, Rue de la Chimie, BP 53
38041 Grenoble Cedex 9
dominique.rieu@imag.fr

Abstract

Pour définir un nouveau langage de modélisation, le mécanisme d'extension d'UML basé sur les profils est régulièrement utilisé comme moyen de description de sa syntaxe abstraite. Comme pour toute syntaxe de langage, le profil évolue au fil du temps. Ces évolutions ont alors un impact direct sur les modèles (qui sont associés au profil) de telle sorte qu'il faut alors adapter ces modèles à la nouvelle version du profil. Le coût d'une adaptation manuelle peut devenir plus important que le coût d'une redéfinition totale des modèles. Il apparaît inévitable de traiter ces évolutions de profils en fonction de leur impact sur les modèles. Dans ce travail, nous nous sommes intéressés à la réduction de ce coût d'adaptation en fonction du type d'évolution détectée et en fonction de son impact sur les modèles. Nous proposons dans cet article un processus d'adaptation automatisé implémenté dans l'outil P^{2}E (Papyrus Profile Evolution). P^{2}E permet de détecter les changements entre deux versions d'un profil, de classer ces derniers en fonction de leur impact et, finalement, d'offrir une migration des modèles en adéquation avec chaque type d'évolution.

UML profiles are a frequently used alternative to describe the abstract syntax of modeling languages. As any language abstract syntax, a profile evolves through time. These evolutions have a direct impact on the models (associated to the profile) so as they need an adaptation to fit to the new profile version. The manual adaptation cost of these models may be more important than redefining models from scratch. It seems unavoidable to treat theses profile evolutions according to their impacts on the models. To reduce this cost, we provide an automated process of models adaptation. In this paper, we present the P^{2}E tool (Papyrus Profile Evolution) which implements our adaptation process. The P^{2}E tool has the ability to detect the changes between two profiles versions, to classify them according to their impacts and to offer an online model migration for each kind of evolution.

MOTS-CLES : profil UML, évolution, classification d'évolution, classification d'impact, migration des modèles.
KEYWORDS: UML profile, evolution, evolution classification, impact classification, models migration.

1 Introduction

Pour définir un nouveau langage de modélisation, le mécanisme d'extension d'UML basé sur les profils [7] est régulièrement utilisé comme moyen de description de sa syntaxe abstraite. Comme pour toute syntaxe abstraite d'un langage, un profil évolue pour différentes raisons : amélioration de la clarté du langage, ajouts de nouveaux concepts, etc. Un profil UML se définit grâce un faible nombre d'éléments de construction (Stéréotype, Extension, etc.), ceci offre l'avantage de réduire la complexité de la gestion de ses évolutions. Or, une évolution d'un profil peut avoir un impact direct sur les modèles l'utilisant, avec pour conséquence que

ces modèles ne deviennent inutilisables voire corrompus. Il apparait donc nécessaire d'offrir un outil de gestion d'évolution qui soit capable de mesurer l'impact de ces changements puis d'adapter les modèles à la nouvelle version du profil. Pour cela nous présentons l'outil P^{2}E (développé comme un plugin Papyrus [1]). P^{2}E (Papyrus Profile Evolution) implémente une approche d'adaptation des modèles se divisant en deux principales phases :

- l'adaptation des modèles de manière à conserver la conformité des modèles vis-à-vis du profil évolué. Un modèle doit être conforme à une syntaxe abstraite, celle-ci est décrite dans notre cas comme un profil UML. Cette étape correspond dans P^{2}E à une opération de migration automatisée.
- l'amélioration possible des modèles pour représenter au mieux le système à modéliser. Une évolution d'un profil peut être due à une amélioration des concepts dont les modèles doivent profiter. Dans P^{2}E, cette étape correspond à une opération d'optimisation réalisée en interaction avec le concepteur des modèles.

Après avoir présenté les principaux outils de migration existants en section 2, nous proposons en section 3 une classification des évolutions des profils en fonction de leurs impacts sur les modèles. En section 4, nous présentons le processus d'adaptation sur un exemple utilisant P^{2}E. Finalement, nous résumons notre travail avant d'ouvrir ses perspectives.

2 Outils existants

Aucun outil ne permet de traiter les modèles impactés par une évolution d'un profil UML, les outils existants concernent le traitement des modèles suite à une évolution d'un métamodèle. Nous avons néanmoins regardé si ils étaient adaptés à cet objectif.

Hermandosfer et al. propose dans [5] un outil appelé COPE. Le système enregistre tous les changements atomiques détectés entre deux versions d'un métamodèle et associe à chaque changement une opération de migration (spécifiée manuellement par le concepteur des modèles).[5] ne propose pas d'étape de classification. De plus, nous pensons que le concepteur des modèles n'est pas systématiquement acteur de l'évolution.

Cicchetti at al. propose dans [2] un outil basé sur l'exécution séquentielle de deux transformations. La première transformation consiste à obtenir un métamodèle de différence à partir d'un métamodèle source. Ce métamodèle sert alors de grammaire pour spécifier les changements dans un modèle de différence. Ce dernier sert alors d'entrée à la seconde transformation pour créer les opérations de migration correspondantes. Comme [5], cette approche ne traite que les changements atomiques. Une autre limitation étant la spécification du modèle de différence par le concepteur des modèles. Celui-ci doit être apte à détecter ces différences. [5] ne propose pas d'étape de classification mais réutilise la classification proposé par Grushko et al.

Levendovszky at al. dans [6] définissent un langage appelé "Model Change Language" (MCL). MCL permet d'établir des relations de correspondance entre deux concepts issus d'un même métamodèle. A chaque relation de correspondance est attachée une règle de migration. L'établissement de ces relations de correspondance et la définition des règles de migration doivent être spécifiées par le concepteur des modèles. Cette approche ne définie pas d'étape de classification.

Grushko et al. dans [1] se base sur une opération de migration spécifiée manuellement et ne traitant que des changements atomiques. Les changements détectés sont cependant classifiés selon trois catégories: les changements non cassants "Non breaking changes" (leur impact

[1] www.papyrusuml.org

fait que les modèles instances restent des instances de la nouvelle version du métamodèle); les changements cassants mais pouvant être résolus "Breaking and resolvable" (les modèles ne sont plus des instances du métamodèle évolué mais la migration de ces modèles est complètement automatisable); les changements cassants et ne pouvant être résolus "Breaking and non resolvable" (la migration n'est pas possible automatiquement et elle nécessite obligatoirement l'intervention du concepteur des modèles).

Notre objectif étant d'offrir une approche d'adaptation la plus automatisée possible, le modèle de différence doit être obtenu de manière automatique. Pour offrir une adaptation et une optimisation les mieux adaptées, l'étape de classification nous semble incontournable. Enfin, le traitement de ces changements devra se traduire par une opération de migration la plus automatisée.

3 Classification de l'impact des évolutions d'un profil

La classification des évolutions est une étape clé pour gérer au mieux leurs impact sur les modèles. Nous avons donc commencé par établir les évolutions possible d'un profil UML. L'expérimentation a été menée sur un profil contenant 17 stéréotypes, 17 liens de généralisations, 37 propriétés, 2 opérations, 5 stéréotypes SysML spécialisés et 5 métaclasses UML étendues. Nous nous sommes limités à l'étude des évolutions les plus rencontrées pour ces six éléments de constructions d'un profil. Nous avons ainsi identifié 84 évolutions possible de ces éléments (évolutions composites ou atomiques). De cette expérimentation, nous obtenons quatre catégories d'évolutions :

- **CATEGORIE 1** : *Impact-free category.* Cette catégorie correspond aux évolutions dont l'impact sur la conformité des modèles est nul. Les modèles n'ont pas à s'adapter à la nouvelle version du profil (aucune opération de migration). Grushko considère ces évolutions comme étant *Non-Breaking*. Dans notre approche, nous considérons que ce type d'évolution impacte cependant la qualité de représentation des modèles. L'ajout de concepts optionnels n'est pas insignifiant pour le concepteur des modèles qui peut alors soit améliorer la clarté des modèles soit satisfaire au mieux les exigences du système modélisé. Dans ce cas, nous suggérons de transmettre à ce dernier, des messages d'amélioration permettant d'identifier les éléments des modèles pouvant être améliorés.

- **CATEGORIE 2** : *Automatic evolution category.* Dans cette catégorie, une opération de migration est nécessaire pour maintenir la conformité des modèles avec le profil évolué. Celle-ci peut être faite de manière complètement automatisée. Prenons l'exemple de l'ajout d'une propriété dont la valeur d'initialisation est renseignée, la migration sera automatique mais toute les instances de cette propriété doivent-elles être égales à cette valeur d'initialisation? Des préconisations d'améliorations sont alors transmises au concepteur des modèles afin de l'informer sur des vérifications à réaliser ultérieurement.

- **CATEGORIE 3** : *Monitored evolution category.* Ce sont les évolutions qui nécessitent une opération de migration assistée (semi-automatisée). Ce type d'évolution contraint l'opération de migration car certaines informations manquent pour rendre complètement automatisable l'opération. Ces contraintes bloquantes ne peuvent être résolues que par interaction avec le concepteur des modèles. Prenons l'exemple de l'ajout d'une propriété dont la valeur d'initialisation n'a pas été définie, quelle sera alors la valeur de la propriété instanciée dans un modèle?

- **CATEGORIE 4** : *Manual evolution category.* Cette catégorie concerne les évolutions dont la migration ne peut pas être automatisée et qui nécessitent une opération de migration manuelle effectuée par le concepteur des modèles. Par exemple, une référence obligatoire ciblant un stéréotype devenu abstrait. Celui-ci est alors spécialisé par deux sous stéréotypes actifs. Quel sous stéréotype, la référence doit elle choisir? Il s'agit dans ce cas de créer un nouvel élément,la description du système diffèrera selon l'instanciation d'un concept plutôt qu'un autre. Les messages d'alertes permettront dans ce cas d'identifier les éléments du modèle devant être traités manuellement pour permettre le processus d'adaptation.

4 Processus d'adaptation des modèles dans P^{2}E

L'approche implémentée dans P^{2}E se divise en quatre étapes : la détection automatique des changements entre deux versions d'un profil (1), leur classification selon les catégories décrites en section 3 (2), l'opération d'adaptation mise en œuvre par une opération de migration (3) et finalement le suivi des messages pour optimiser la description des modèles (4). Pour illustrer notre approche, nous nous sommes intéressés à l'évolution du langage EAST-ADL2 [3] . EAST-ADL2 est un standard utilisé dans le domaine de l'automobile pour décrire des systèmes à un haut niveau d'abstraction. Ce langage est représentatif de notre approche du fait que les concepteurs ont tout d'abord défini un métamodèle pour décrire la syntaxe abstraite du langage puis ont fait le choix de l'implémenter sous forme d'un profil UML.

4.1 Étape 1 : Génération du modèle de différence

Depuis 2010, le standard EAST-ADL2 a évolué en trois versions différentes. Pour détecter les différences entre deux de ces versions, nous avons fait le choix de réutiliser la technologie `EMFCompare` [4] . Nous avons étendu son mécanisme pour qu'il soit en mesure de détecter tout type de changement et également de générer en sortie un modèle de différence structuré selon notre métamodèle de différence. Ce dernier permet d'identifier le type de l'élément évoluant, l'opération d'évolution, etc. Entre la version 2.0 et 2.1, 580 éléments ont été ajoutés, 529 éléments ont été supprimés et 82 concepts modifiés, soit au total 1191 changements détectés. Considérons l'un de ces 1191 changements, dans la version 2.0 d'EAST-ADL2 (figure 1 (a)), les fonctions d'un système sont représentées par le concept `ADLFunctionType`. Elles peuvent être hiérarchiques et communiquent entre elles via des ports appelés `ADLFlowPort`. Ce concept est abstrait (n'est pas instanciable) et spécialisé par trois sous stéréotypes instanciables : `ADLInFlowPort`, `ADLOutFlowPort` et `ADLInOutFlowPort`. Dans la version 2.1 (figure 1 (b)), la direction des ports est désormais décrite par une nouvelle propriété `direction` appartenant au stéréotype `FunctionFlowPort`. L'énumération `EADirectionKind` précise les trois directions possibles (`in`, `out`, `inout`). La figure 2 présente le modèle de différence de façon schématique à l'issue de l'évolution.

4.2 Étape 2 : Classification des changements détectés

L'étape de classification consiste à réorganiser les modèles de différence selon les catégories ci-dessus. Le modèle de différence (figure 2) montre que trois opérations d'évolution ont été effectuées sur cet extrait de profil. L'opération de suppression peut être complètement automatisée, l'opération de modification est également automatisée (cf catégorie 2). L'opération d'addition n'est pas automatisable si l'ensemble de ses caractéristiques n'est pas renseigné.

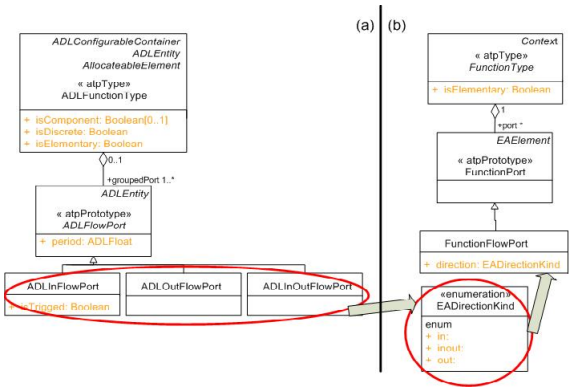

Figure 1: Concept de port d'EAST-ADL2

Opération d'évolution	Element
Suppression	ADLInFlowPort ADLOutFlowPort ADLInOutFlowPort
Modification	ADLFlowPort *abstrait → actif*
Addition	Property « direction » *Type : Enumeration EADirectionKind* *Owner : ADLFlowPort* Enumeration « EADirectionKind » *Literals: in, out, inout*

Figure 2: Modèle de différence simplifié

Or, la valeur d'initialisation de la propriété n'est pas précisée. Dans cet exemple, nous remarquons que la décomposition d'une évolution globale en opérations atomiques peut réduire l'automatisation de l'opération de migration du modèle. Dans notre approche, nous proposons de faire des recherches sur le modèle de différence pour trouver des patrons d'évolutions composites. Nous avons ainsi défini un catalogue regroupant l'ensemble des évolutions d'un profil sous la forme de patrons. A chaque patron, nous lui avons attribué la catégorie correspondante. Cet exemple correspond à une instance d'un patron d'évolution appelé `Suppression de sous stéréotypes devenant un type énuméré`. En effet, les trois opérations fusionnées deviennent une seule évolution composite à traiter. Le type de chaque port peut être associé (mapping) à un littéral et servir implicitement de valeur d'initialisation. Un patron composite permet d'automatiser une migration qui ne serait pas automatisée si prise de façon atomique. Plus largement, l'attribution d'une catégorie à chaque patron détecté sera faite par filtrage en accord avec le catalogue de patrons d'évolution.

4.3 Étapes 3 et 4 : Opération de migration et d'optimisation

Pour les évolutions de type *Impact-free*, aucune opération de migration n'est nécessaire, cependant des messages d'améliorations sont générés pour un traitement à l'étape 4.
Pour les évolutions *Automatic* et *Monitored*, nous générons automatiquement une règle de transformation spécifique au patron d'évolution détecté. En effet, nous avons étudié chaque patron d'évolution et défini dans le catalogue une règle de migration. Ceci nous permet ainsi de traiter des patrons non atomiques (ce qui n'est pas le cas des outils existants). Les règles décrites dans ce catalogue prennent également en compte les contraintes bloquantes devant être résolues par interaction avec le concepteur des modèles. A l'issue de la génération de ces transformations, des messages d'amélioration cohérente avec la classification sont alors créés.
Pour les évolutions de type *Manual*, nous avons pour objectif de réduire leur occurrence en proposant des solutions automatisées pour quelques cas. Si aucune solution n'est envisageable, des messages d'alertes sont générés pour remonter au concepteur l'ensemble des éléments nécessitant une migration manuelle et ainsi compléter le processus d'adaptation.
Pour notre exemple, le modèle utilisant les concepts d'`ADLFunctionType` et d'`ADLInFlowPort` d'EAST-ADL2 dans la version 2.0 est représenté par la figure 3 : l'élément `Engine` stéréotypé `ADLFunctionType` est la fonction `moteur` du système à modéliser, elle possède un port particulier stéréotypé `ADLInFlowPort` représentant les valeurs recueillies par la pédale d'accélérateur.

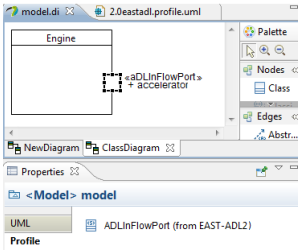

Figure 3: Modèle avant adaptation

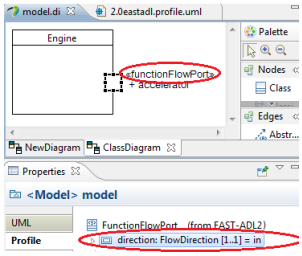

Figure 4: Modèle après adaptation

L'opération de migration qui consiste à instancier la règle de migration est composée 1/ d'un mapping entre les trois sous stéréotypes et les littéraux (exemple : `ADLInFlowPort` = `in`). 2/ du remplacement du stéréotype `ADLInFlowPort` par `FunctionFlowPort`. 3/ de la création d'une nouvelle propriété `direction`. 4/ du renseignement de celle-ci par la valeur spécifique. Nous n'avons pas traité pour cet exemple le renommage des concepts. La figure 4 illustre le résultat du processus d'adaptation appliqué sur le modèle en figure 3.

L'étape d'optimisation n'est pas détaillée pour cet exemple.

5 Conclusion et Perspectives

Dans cet article, nous avons présenté l'outil P²E implémentant les premiers résultats pour la gestion des impacts des évolutions d'un profil UML dans l'outil Papyrus. Nous avons établi une classification des évolutions d'un profil UML selon leur impact sur les modèles (préliminaire à l'étape de migration) puis selon le type de message d'améliorations à transmettre au concepteur des modèles (préliminaire à l'étape d'optimisation). Nous avons considéré les évolutions les plus rencontrées de six éléments de construction d'un profil UML, l'objectif étant cependant d'offrir aux concepteurs des profils la possibilité d'étendre le catalogue de patrons d'évolution pour des évolutions particulières. P²E devra être complété par l'implémentation d'une technique de filtrage pour la détection des patrons d'évolution composites.

References

[1] Richard F. Paige Boris Gruschko, Dimitrios S. Kolovos. Towards synchronizing models with evolving metamodels. In *ECSMR'07*, 2007.

[2] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Automating co-evolution in model-driven engineering. EDOC '08, pages 222–231, Washington, DC, USA, 2008.

[3] EAST-ADL, 2010. `http://www.atesst.org/home/`.

[4] EMFCompare, 2011. `http://wiki.eclipse.org/emf/compare`.

[5] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. Automatability of coupled evolution of metamodels and models in practice. In *MODELS'08*, pages 645–659, Berlin, Heidelberg, 2008.

[6] Tihamer Levendovszky, Bernhard Rumpe, Bernhard Schätz, and Jonathan Sprinkle. Model evolution and management. In *MBEERTS'07*, pages 241–270, 2007.

[7] Bran Selic. A systematic approach to domain-specific language design using uml. In *ISORC '07*, pages 2–9, Washington, DC, USA, 2007.