Motif pour la métamodélisation: Interfaçage entre contextes

Alain Plantec, Vincent Ribaud

Motif pour la métamodélisation

Interfaçage entre contextes

Alain Plantec — Vincent Ribaud

EA3883, LISyC, Université de Bretagne Occidentale,
C.S. 93837, 29238 Brest Cedex 3
{alain.plantec,vincent.ribaud}@univ-brest.fr

RÉSUMÉ. *Les métamodèles sont très souvent conçus à partir de la définition de contextes dans lesquels sont spécifiés les différents éléments du langage modélisé. Les langages proposent alors un ou plusieurs mécanismes d'interfaçage entre contextes permettant la réutilisation d'élements d'un contexte cible depuis un contexte source. Interfaçage entre contextes est un motif pour l'implantation de la réutilisation inter-contextes*

ABSTRACT. *Metamodels are often designed with contexts definition which include language elements definitions. Target context elements reuse from a source context are implemented using one or several interfacing mechanism. Contexts interfacing is a pattern for such implementation*

MOTS-CLÉS : *métamodèle, contexte, interfaçage, réutilisation*

KEYWORDS: *metamodel, context, interfacing, reuse*

1. Motivation

La plupart des langages de modélisation ou de programmation permettent la spécification d'éléments de modélisation au sein d'un contexte. Ce principe est à la base de la notion de visibilité des concepts. Un élément est implicitement visible dans le contexte auquel il appartient : il peut ainsi être référencé par les autres éléments de modélisation du même contexte. On parle de *paquetage* en *java*, d'*espace de nommage* en *C++* ou de *schéma* en *EXPRESS*.

Le développement modulaire, la compilation séparée et plus récemment l'ingéniérie des modèles permettent de mettre en oeuvre la réutilisation de concepts. Le principe très majoritairement retenu est d'indiquer, dans le contexte utilisateur, les concepts réutilisés. Les contextes et les concepts réutilisables sont nommés.

La déclaration des concepts utilisés s'effectue :

– soit en indiquant le nom du contexte, tous les concepts du contexte utilisé sont alors implicitement inclus ;

– soit en indiquant une liste de noms de concepts par contexte utilisé, les concepts sont alors explicitement inclus.

Un concept utilisé pouvant lui même référencer d'autres concepts, par transitivité, ces autres concepts peuvent devenir implicitement inclus.

L'interfaçage peut avoir pour conséquence d'introduire une ambiguité de référence lorsque qu'un concept importé porte le même nom qu'un concept local ou qu'un concept importé par ailleur. Suivant le langage et suivant l'environnement hôte, ce problème est principalement géré par le nommage complet du chemin incluant le nom des contextes parents ou par le renommage du concept utilisé.

2. Indication d'utilisation

Interfaçage entre contextes est de portée large puisqu'il peut s'appliquer aussi bien pour des modèles textuels que pour des modèles graphiques. Le motif permet la constitution de graphes reliant les différents contextes d'un modèle tout en autorisant la gestion des relations inter-contextes cycliques.

Ce motif est principalement utilisé pour bénéficier de la réutilisation entre différents contextes. La notion d'*interfaçage* permet à un environnement de localiser les concepts effectivement utilisés et ainsi de distinguer les concepts locaux des concepts externes.

Pour la compilation, la génération de code ou pour le remodelage cette distinction et indispensable pour que le code ou le modèle synthétisé puisse refléter de l'organisation modulaire des spécifications d'entrée. Un *Interfaçage entre contextes* est utilisé par un compilateur pour construire la table des symboles exploitée par l'éditeur de liens. Un générateur de code exploite un *Interfaçage entre contextes* pour implanter

les imports et organiser le code généré. Pour la rétro ingéniérie, l'organisation modulaire d'un programme peut se traduire directement dans l'environnement de modélisation. Les graphes obtenus peuvent être analysés (arbres des dépendances, références croisées ...) et réorganisés.

Le motif est utile pour la vérification des modèles et l'élaboration de messages d'erreur intelligibles. En effet, la réutilisation ainsi représentée spécifie une intention de réutilisation. A l'issue d'un processus de résolution, l'intention peut se révéler insoluble. Les métadonnées disponibles permettent alors de disposer de suffisamment d'informations pour construire des rapports d'erreur intelligibles.

Dans un environnement, l'aide à la saisie peut tirer partie de ce motif pour proposer les symboles par modules. Seul les symboles externes réellement importés sont proposés en tenant compte du renommage éventuel.

3. Structure

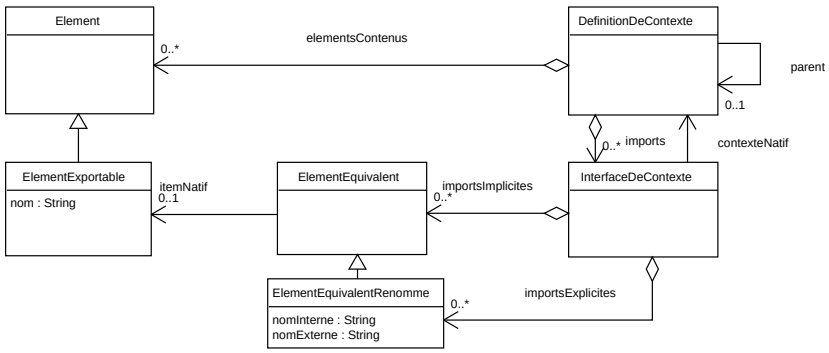

Figure 1. *Structure du motif "Interfaçage entre contextes"*

4. Constituants

4.1. *Element*

Représente tout élément de modèle spécifié dans un contexte.

4.2. *ElementExportable*

Représente un *Element* qui peut être exporté, soit, réutilisé par un contexte source. Très généralement, la déclaration d'un import repose sur un identifiant (*nom*), constituant une clé non ambiguë d'accès à l'élément dans le contexte auquel il appartient.

4.3. *DefinitionDeContexte*

Un *DefinitionDeContexte* constitue le contexte des éléments modélisés (*elements-Contenus*). Un *DefinitionDeContexte* peut être éventuellement lui même déclaré dans un *DefinitionDeContexte* parent.

4.4. *InterfaceDeContexte*

Une *InterfaceDeContexte* permet d'accéder aux éléments importés (*imports*) depuis un contexte source. Elle référence un contexte cible (*contextNatif*) dont tout ou partie des *ElementsExportable* sont réutilisés par le contexte source.

4.5. *ElementEquivalent*

Un *ElementEquivalent* représente une intention de réutilisation implicite d'un *ElementsExportable*. Une réutilisation implicite étant calculée, elle est résolue par construction. Via un *ElementEquivalent*, il est donc toujours possible à l'*InterfaceDeContexte* d'accéder à l'*ElementExportable* réutilisé.

La création d'un *ElementEquivalent* est effectué lorsqu'un *DefinitionDeContexte* source spécifie la réutilisation globale de tous les *ElementExportable* d'un *Definition-DeContexte* cible ou par le traitement des imports implicites induits par transitivité.

4.6. *ElementEquivalentRenomme*

Un *ElementEquivalentRenomme* représente une intention de réutilisation explicite d'un *ElementExportable*. *nomExterne* constitue la clé qui identifie l'*ElementExportable* dans le *DefinitionDeContexte* cible. *nomInterne* représente la clé qui identifie l'*ElementExportable* dans le *DefinitionDeContexte* source.

L'absence de renommage peut être indiqué soit par l'absence de *nomInterne*, soit par l'égalité des deux clés *nomExterne* et *nomInterne*.

Un *ElementEquivalentRenomme* peut ne pas être résolu dans le *DefinitionDeContexte* cible. Dans ce cas, un rapport d'erreur peut utiliser la clé *nomExterne*.

5. Exemple

L'exemple de la figure 2 présente un métamodèle simplifié du langage de modélisation EXPRESS (ISO 1994b) du standard ISO STEP (ISO 1994a) :

– le contexte (*context_definition*) comprend la définition de types (*named_type*), d'entités (*entity_definition*), de procedures et de functions (*statements_container*) ;

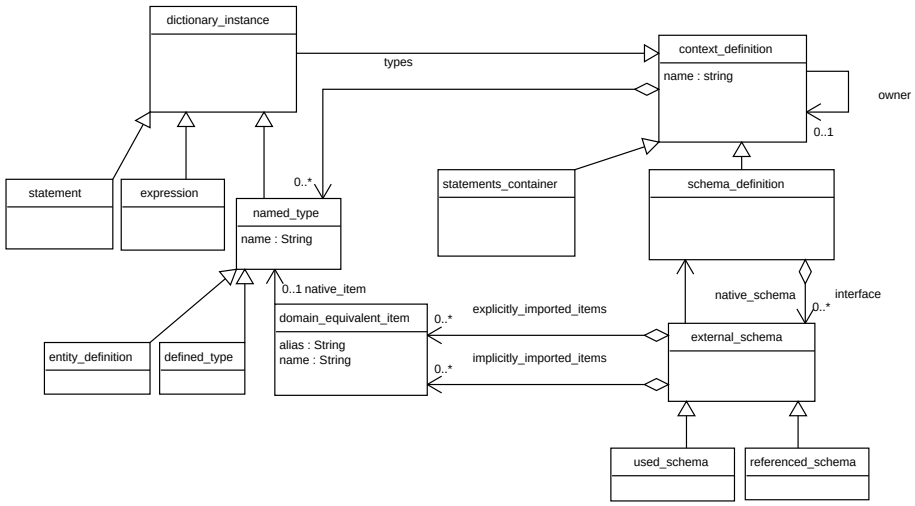

Figure 2. *Extrait d'un métamodèle simplifié d'EXPRESS*

– le schéma (*schema_definition*) est le contexte racine au niveau duquel il est possible d'importer les définitions d'autres schémas et joue donc le rôle de *DefinitionDeContexte*.

– *external_schema* joue le rôle de *InterfaceDeContexte* ; EXPRESS dispose de deux mécanismes de réulisation représentés par *used_schema* et *referenced_schema* qui spécialisent *external_schema* ;

– l'intention d'une réutilisation représenté par *domain_equivalent_item* peut être précisé avec un alias pour indiquer un renommage ; Dans cet exemple, *domain_equivalent_item* joue à la fois le rôle de *ElementEquivalent* et *ElementEquivalentRenomme*.

– la réutilisation est implicite ou explicite.

La norme précise l'algorithme de calcul des réutilisations implicites et de résolution des références. Ces algorithmes s'appuient sur les *external_schema* qui doivent dans un premier temps être eux même résolus.

Les procédures et les fonctions (les *statements_container*) d'un schéma peuvent aussi être réutilisés. Cette possibilité n'apparaît pas dans le métamodèle simplifié.

6. Bibliographie

ISO, *Part 1 : Overview and fundamental principles*. 1994a.

ISO, *Part 11 : EXPRESS Language Reference Manual*. 1994b.