Un procédé de validation des métamodèles par les métadonnées

Alain Plantec, Vincent Ribaud

Un procédé de validation des métamodèles par les métadonnées

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

La norme STEP a pour objet de standardiser des modèles de données par métier. Ce standard a développé un procédé de validation de la conformité des modèles dans lequel l'instanciation est un composant essentiel : un modèle est valide si un ensemble d'instances conformes au modèle est jugé «sémantiquement correct» par un expert du domaine. Appliqué à la métamodélisation, cette approche permet de construire et de valider les métamodèles par raffinements successifs. Cette méthode est utilisée dans l'environnement Platypus qui intègre un outillage STEP dans un système Squeak.

1 Introduction

Dans le domaine de la gestion des données, les modèles sont élaborés par cycle de conception-mise en œuvre. Un modèle est considéré comme pertinent si sa mise en œuvre permet de gérer correctement les données du domaine. Si ce n'est pas le cas, le modèle évolue puis sa mise en œuvre jusqu'à ce que la gestion des données soit jugée sensée par un familier du domaine.

Le point de vue dominant sur la manière d'élaborer des métamodèles reproduit celui employé pour des modèles : il faut mettre en œuvre les métamodèles dans des outils et expérimenter des modèles pour valider les métamodèles.

La norme *STEP* a pour objet de produire des modèles de données standards par métier. Son but est de résoudre les problèmes d'échange de données et des modèles afférents. Ceci addressant des questions techniques et méthodiques, la norme s'est dotée d'une part, d'un socle technologique de description des données et d'autre part, d'un procédé d'élaboration de modèles métiers, fondés sur l'usage de ces données dans les applications. Cette méthode met donc en avant la nécessité de disposer de jeux de données valides tout au long de l'élaboration des modèles.

Platypus [15] est un environnement *Squeak* [17] de modélisation et de transformation intégrant un socle technologique *STEP*. Il est historiquement issu d'une convention *CIFRE* entre l'*UBO* et la société *Thales-IS* [14].

Face à la question des méthodes et outils de métamodélisation, nous répondons avec l'approche *STEP* : le procédé de construction d'un métamodèle est fortement couplé à la disponibilité de modèles valides et sensés. C'est l'existence de ces modèles dans le méta-environnement

qui permet de raffiner le métamodèle. Cet article présente un tel procédé de métamodélisation. Son originalité est de prendre en considération non seulement le besoin d'un environnement pour gérer les métadonnées mais aussi d'intégrer une démarche incrémentale habituelle dans le cadre de la spécification de modèles complexes.

2 La norme *STEP*

La norme *STEP* (STandard for the Exchange of Product model data), référencée comme standard *ISO 10303* [7], a pour objectif de produire des modèles de données standards par métier, appelés *protocole d'application*. Les grandes sociétés industrielles comme *General Motors* ou *Dassault* sont à l'origine des travaux de normalisation pour résoudre leurs problèmes d'échange de données (voir par exemple [6] concernant l'utilisation de *STEP* dans le secteur automobile). Pour ce faire, la norme s'est dotée de parties technologiques dont les constituants principaux sont une méthode de description, le langage de modélisation des données *EXPRESS* [8] et une méthode de mise en œuvre, le format d'échange de données neutre *STEP* [9]. Les évolutions récentes de cette technologie tendent vers un rapprochement avec les outils de l'*OMG*, avec la standardisation de passerelles conceptuelles et techniques entre *EXPRESS-STEP* et *UML-XMI*.

2.1 Le langage *EXPRESS*

Les *protocoles d'application* de la norme sont spécifiés avec *EXPRESS*, langage de modélisation pour la spécification de schémas de données. Ce langage est textuel et repose sur une grammaire de type LL(1). Bien qu'il intègre certaines caractéristiques des langages à objets typés comme *C++* ou *Java* et de langages de définition et de manipulation de données comme *SQL*, il est spécifié indépendamment d'un système ou d'un langage cible particulier :

- un schéma se compose de types d'entités et de contraintes globales ;
- une entité décrit un ensemble d'objets qui partagent des propriétés et des contraintes communes ; ainsi, la spécification d'une entité comprend des attributs explicite, dérivés, inverses et la définition de contraintes locales ou de règles d'unicité ;
  - un attribut explicite est une propriété dont la valeur est stockée (échangée au sens *STEP*), sa valeur est obligatoire par défaut ;
  - un attribut dérivé comprend une expression qui permet de calculer sa valeur ;
  - un attribut inverse permet d'expliciter les cardinalités et de naviguer dans le sens inverse de l'association ;
  - une contrainte permet de spécifier une règle d'intégrité locale à l'entité ;
  - une règle d'unicité permet de spécifier une liste d'attributs dont la combinaison des valeurs doit être unique pour toutes les instances de l'entité ;
- une contrainte globale permet de contraindre simultanément toutes les instances d'une ou plusieurs entités.

EXPRESS autorise l'héritage multiple (les ambiguités sont réglées par indication du chemin d'accès aux attributs) et la réutilisation entre schémas. La figure 1 montre des exemples d'entités et des instances correspondantes au format *STEP*.

2.2 La méthode *STEP*

STEP apporte non seulement une technologie pour la modélisation et l'échange des données mais aussi une méthode de modélisation. En effet, une de ses spécificités importantes

```
ENTITY cartesian_point SUBTYPE OF ( point );
  coordinates : LIST [ 1 : 3 ] OF REAL;
DERIVE
  dim : dimension_count := dimension_of ( SELF );
END_ENTITY;

ENTITY direction SUBTYPE OF ( geometric_representation_item );
direction_ratios : LIST [ 2 : 3 ] OF REAL;
WHERE
 wr1 : (SIZEOF(QUERY( tmp <* direction_ratios | (tmp <> 0))) > 0);
END_ENTITY;

ENTITY axis2_placement_3d SUBTYPE OF (geometric_representation_item);
 location : cartesian_point;
 axis : OPTIONAL direction;
 ref_direction : OPTIONAL direction;
DERIVE
 p : LIST [3:3] OF direction := build_axes(axis,ref_direction);
WHERE
 wr1 : (SELF\placement.location.dim = 3);
 wr2 : ((NOT EXISTS (axis)) OR (axis.dim = 3));
 wr3 : ((NOT EXISTS (ref_direction)) OR (ref_direction.dim = 3));
 wr4 : ((NOT EXISTS (axis)) OR (NOT EXISTS (ref_direction)) OR
        (cross_product (axis,ref_direction).magnitude > 0));
END_ENTITY;
```

```
#1001=DIRECTION('dir1',(1.0,0.0,0.0));
#1050=DIRECTION('dir10',(0.0,0.0,1.0));
#1053=CARTESIAN_POINT('cp13',(0.75,0.5,-0.25));
#1054=AXIS2_PLACEMENT_3D('ap5',#1053,#1050,#1001);
```

FIG. 1 – L'encadré supérieur présente trois entités extraites du protocole d'application standard AP203 utilisé pour la modélisation d'objets en 3D. Ces deux entités spécifient le concept de placement d'un axe en 3D utile à la définition d'une surface élémentaire (comme un plan, un cylindre, un cône ou une sphère). Une direction est un point en deux ou trois dimensions, qui est l'extrémité du vecteur de direction (l'origine du vecteur étant l'origine de l'espace). Pour être valide, une instance de direction doit respecter la contrainte d'intégrité suivant : la requête calculant si une des coordonnées de la direction est différente de 0 doit avoir au moins une ligne résultat (autrement dit, l'extrémité de la direction ne peut être l'origine). Un placement se constitue d'un point (*location*), éventuellement complété d'un axe (*axis*) et d'une direction (*ref_direction*). Les trois directions de l'axe sont données par l'attribut *p* dont la valeur est calculée par la fonction *build_axes*. Pour être valide, une instance doit respecter quatre contraintes d'intégrité (à titre d'exemple la première indique que le point de référence doit être en trois dimensions). L'encadré inférieur présente l'encodage *STEP* d'une instance valide de *axis2_placement_3d* ainsi que les instances avec lesquelles elle est en relation.

est d'intégrer la notion d'instance au niveau modélisation. A tout schéma *EXPRESS* correspond un langage d'instanciation valide. L'interprétation d'un schéma, sa sémantique dans un contexte précis, est donnée par un ensemble d'instances considérées comme valides si toutes les contraintes (types, cardinalités, règles d'optionalité, règles d'intégrité locales et globales) sont validées.

La spécification d'un *protocole d'application* est une activité d'analyse très complexe. Une version standardisée est l'aboutissement d'une série de versions successives. Il s'agit d'un processus incrémental pendant lequel se succèdent des étapes de spécification et de validation. Dans la méthode *STEP*, l'activité de validation procède par mise en situation d'exemples réels. Cette activité est décrite par les méthodes de test de conformité de la norme [10] dans laquelle l'instanciation est utilisée comme support de validation *lexicale*, *structurelle* et *sémantique*. Au

sens *STEP*, la validation *lexicale* et *structurelle* se rapporte directement à l'évaluation automatique des règles des schémas alors que la validation *sémantique* est plus le fait de l'expert qui évalue la conformité des données vis à vis des règles du domaine. Un schéma est déclaré valide par rapport aux interprétations données pour des lots d'instances constituant des jeux d'essais standards.

Il s'agit d'une différence importante avec, par exemple la norme *XML*, qui considère la validité des instances par rapport à un *schéma XML* ou une *DTD* mais qui n'intègre pas de méthode de validation des modèles par les données comme c'est le cas dans *STEP*.

3 Modélisation des métadonnées

3.1 Définitions

3.1.1 Métainformation

Une métainformation, comme par exemple un commentaire dans le code source d'un programme ou le type d'une variable, permet de décrire des informations. Une métainformation est différemment exploitable selon la nature de la sémantique qu'elle spécifie. Dans [18], Michael Uschold décrit quatre catégories de sémantique dans le cadre d'un «continuum sémantique» illustré par la figure 2.

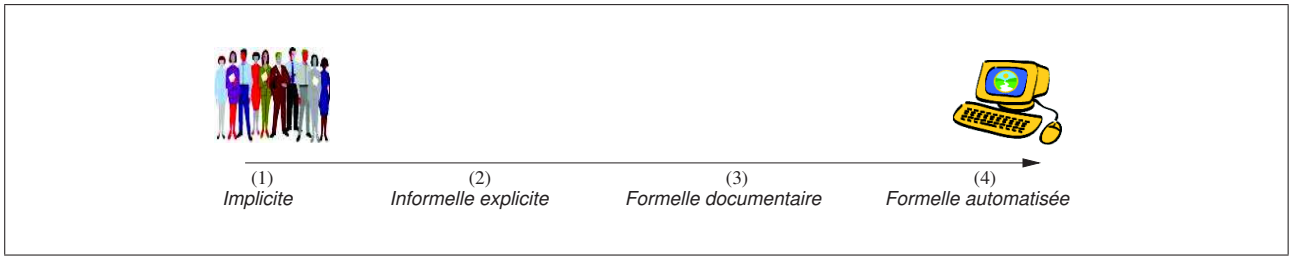

FIG. 2 – Une sémantique peut être implicite (1) si le sens porté fait l'objet d'un consensus intellectuel dans un groupe de personnes sans retranscription sur aucun support ; une sémantique informelle (2) est en plus retranscrite ; enfin la sémantique peut être explicitée de manière formelle dans un but documentaire (3) ou pour être exploitée automatiquement (4).

3.1.2 Métadonnée

Comme dans [4], nous appelons métadonnée une métainformation dont la sémantique est spécifiée formellement par des règles relatives à une donnée dans un contexte particulier, comme un environnement dans un certain domaine par exemple. La sémantique d'une métadonnée peut comprendre la description des concepts manipulés dans le contexte, des contraintes de validité, des règles de représentation, de transformation, de configuration et de la documentation [3]. Il s'agit donc de sémantique formelle, documentaire ou automatisée.

3.1.3 Métamodélisation

Une métadonnée est caractérisée formellement dans un modèle de donnée appelé métamodèle. La métamodélisation est une activité complexe où l'on cherche à cerner les caractéristiques d'un domaine au travers de la production de métamodèles [11]. Cette définition a deux implications importantes : il n'y a pas de représentation unique et définitive d'un domaine et une activité de métamodélisation est toujours liée à un domaine.

3.1.4 Méta-environnement

Un méta-environnement est un outil qui sert à produire d'autres outils. Il permet de spécifier et d'exploiter des métamodèles pour le développement d'environnements spécifiques. Les fonctions principales d'un méta-environnement sont :

- l'édition et la documentation de méta-modèles et de contraintes de validité associées ;
- la spécification et la mise en œuvre de générateurs de code produisant tout ou partie des environnements ;
- la production des environnements comprenant les composants de vérification sémantique des modèles gérés sur la base des contraintes spécifiées dans le méta-modèle.

3.2 Métamodélisation avec Platypus

3.2.1 L'environnement *Platypus*

Platypus [15] est un méta-environnement. Il met en œuvre au sein d'un système *Smalltalk Squeak* [17], un environnement de métamodélisation basé sur la norme *STEP*. Les métamodèles sont spécifiés avec *EXPRESS*. Dans un métamodèle, les entités spécifient les concepts du domaines (du langage) à modéliser. Les contraintes précisent leur sémantique localement ou globalement. Les transformations sont exprimées par des attributs dérivés [14, 12].

Platypus dispose d'un analyseur sémantique et d'un interpréteur pour *EXPRESS* qui permettent une validation au sens *STEP*. Les validations lexicales et structurelles sont effectuées par l'analyse du métamodèle et l'évaluation des règles. Pour la validation sémantique, l'expert du domaine analyse les résultats au travers de rapports d'analyse et du résultat du calcul des attributs dérivés.

Squeak [17] est un système *Smalltalk* complet libre, portable et très activement maintenu. Le système est instrumenté par une interface de développement très riche et de nombreux outils comme des explorateurs de code, un générateur de compilateurs, un gestionnaire de versions... La compilation incrémentale permet aux applications d'être très aisément testées en cours de développement et favorise une mise en œuvre très rapide.

L'intégration dans *Squeak* permet de compléter très efficacement les tests et l'exploitation des résultats. En effet, l'utilisateur peut, par exemple, produire les lots d'instances de référence pour les tests et construire des composants spécifiques pour exporter ou visualiser les résultats des transformations.

3.2.2 Orientation «donnée» des métamodèles

Suivant le domaine envisagé, les modèles exploités pour la construction des applications peuvent être de natures très différentes comme par exemple les modèles de la norme *UML*. Par contre, le paradigme de la modélisation des données convient et est souvent employée pour la spécification de la sémantique abstraite : sont spécifiés des types d'entités composés d'attributs, des relations, des règles d'intégrité et de transformation [5].

Par exemple, dans [16], les auteurs utilisent le diagramme des classes de la norme *UML* pour présenter une partie de la sémantique des métamodèles. Dans [12], est présentée une méthode utilisant *EXPRESS* pour modéliser les métamodèles exploités pour échanger des données entre bases de données hétérogènes.

Notre environnement *Platypus* utilise des schémas *EXPRESS* pour modéliser les métamodèles exploités. La figure 3 montre une partie du métamodèle de *SQL* concernant les tables. La sémantique est complétée par des lots d'instances conformes aux différentes règles. Les valeurs

```
SCHEMA sql_dictionary;

  ENTITY base_type
    ABSTRACT SUPERTYPE OF(ONEOF(real_type, integer_type, string_type));
  END_ENTITY;

  ENTITY table;
    name : STRING;
    columns : LIST OF column;
  END_ENTITY;

  ENTITY column;
    name : STRING;
    domain : base_type;
    primary : BOOLEAN;
    foreign : OPTIONAL column;
  INVERSE
    owner : table FOR columns;
  END_ENTITY;

END_SCHEMA;
```

```
#30=STRING_TYPE();
#31=REAL_TYPE();
#64=TABLE('direction',(#70));
#70=COLUMN('name',#30,.T.,$));
#90=TABLE('direction_ratios',(#91,#92));
#91=COLUMN('parent',#30,.F.,#70);
#92=COLUMN('value',#31,.F.,$);
```

FIG. 3 – L'encadré supérieur contient une partie du métamodèle de *SQL*. L'entité *table* ne contient que des attributs explicites, notamment la liste de ses colonnes. L'entité *column* comprend l'attribut inverse *owner* qui, par calcul, donne la table propriétaire de la colonne. L'encadré inférieur présente des instances correspondant à la mise en œuvre en *SQL* de l'entité *direction* de la figure 1. On remarque que l'attribut *direction_ratios* est traduit dans une table séparée reliée par une clé étrangère à la table *direction* (via la colonne *parent* de la table *direction_ratios* dont l'attribut *foreign* pointe sur la colonne *name* de la table *direction*).

des attributs explicites étant données, on peut calculer celles des attributs inverses et dérivés ainsi que les résultats d'évaluation des contraintes.

3.2.3 Rôles et spécificité des métamodèles

Les travaux actuels autour du *MDA* renforcent les prérogatives des modèles et des métamodèles en les considérant comme des constituants actifs du processus de production des applications. A ce niveau d'abstraction sont considérées comme activités essentielles, la vérification et la transformation de modèles :

- la vérification consiste en la validation de règles ou contraintes spécifiées dans le métamodèle ;
- par l'application de règles définies dans le métamodèle sous la forme d'attributs dérivés, la transformation de modèle permet la production d'une réalisation à partir des métadonnées ; une réalisation est typiquement un autre modèle ou toute ou partie du code d'une application.

De plus, la multiplicité des outils et environnements existants implique un besoin d'environnements interopérables. Les modèles doivent pouvoir être échangés. L'échange de modèle procède soit à partir d'une forme structurée lisible (un schéma *EXPRESS* par exemple) soit à partir des

métadonnées sérialisées (un fichier *XML* ou *XMI* par exemple).

3.2.4 Architecture

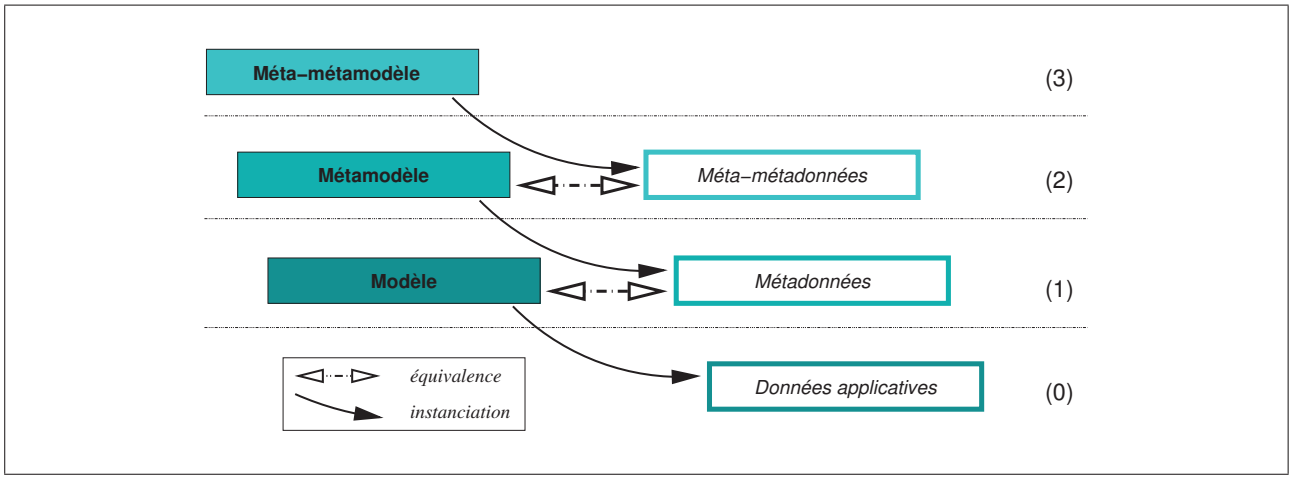

FIG. 4 – L'architecture de modélisation se compose de quatres niveaux. Le niveau APPLICATION (0) comprend les données applicatives, instances des modèles qui décrivent l'application souvent suivant différents points de vue (les modèles d'UML par exemple). Le niveau MODÈLE (1) comprend les métadonnées qui décrivent les données applicatives. Le niveau MÉTAMODÈLE (2) se compose des méta-métadonnées qui décrivent la structure et la sémantique des métadonnées. Le niveau MÉTA-MÉTAMODÈLE (3) décrit la structure et la sémantique des méta-métadonnées

On se situe dans le cadre d'une architecture à quatre niveaux comme celle exploitée par le *MDA* [13, 1]. Comme le montre la figure 4, pour chaque niveau, les caractéristiques suivantes sont récurrentes :

- les données d'un niveau $n-1$ constituent une instanciation du modèle du niveau n qui en spécifie la sémantique ;
- les données et le modèle d'un même niveau n sont «sémantiquement» équivalents.

Conceptuellement, l'intérêt fondamental de l'architecture *MDA* provient des différents niveaux d'abstraction : chaque niveau n de cette architecture permet de spécifier un système du niveau $n-1$ en se focalisant sur ce qui est essentiel au système et en s'abstrayant des spécificités de mise en œuvre comme le langage ou l'architecture cible [2].

4 Validation des métadonnées avec Platypus

4.1 Validation des métadonnées

La validation d'une métadonnée est le fruit d'un processus. Ce processus de validation est le fait d'une interprétation informelle ou automatisée suivant le cas, en référence au «continuum sémantique» présenté dans la figure 2. Réutilisant la terminologie employée dans *STEP*, nous distinguons trois niveaux de validation :

- le niveau *lexical* concerne la vérification de règles liées aux valeurs (le nom d'une classe commence toujours par une majuscule, ou le nombre de caractères d'un identifiant ne dépasse pas une certaine taille par exemple) ; l'attribut dérivé *sql_name* ci-dessous met en œuvre la règle lexicale indiquant l'insensibilité de *SQL* à la casse pour le nom des tables ;

```
ENTITY table ...
```

```
  DERIVE
    sql_name : uppercase(SELF.name);
END_ENTITY;
```

– le niveau *structurel* concerne la composition, les règles d'intégrité et l'exécutabilité des transformations ; la règle structurelle ci-dessous contraint le nom d'une table à être unique ;

```
ENTITY table ...
  UNIQUE
    sql_name;
END_ENTITY;
```

– le niveau *sémantique* concerne la vérification de l'adéquation des métadonnées aux règles du domaine ; à ce niveau, la cohérence des résultats d'évaluation des règles et des transformations par rapport au domaine est examinée.

Pour les niveaux *lexical* et *structurel*, la validation est automatisable. Par contre, le niveau *sémantique* nécessite l'intervention d'un expert qui s'appuie sur toutes les formes de sémantiques (d'implicite à automatisée) pour porter un jugement de l'adéquation entre la métadonnée et le métamodèle. Les résultats possibles sont soit que le métamodèle est cohérent et la méta-donnée est invalide, soit l'inverse. Le processus de validation des métadonnées peut donc aboutir à l'invalidation du métamodèle.

4.2 Un exemple de validation par les méta-données

En *SQL*, une clé étrangère doit référencer une clé primaire ou unique. Pour simplifier l'exemple, on ne considère que des clés mono-colonne et on ne traite pas le cas des clés uniques. Cette règle s'exprime par la contrainte locale ci-dessous :

```
ENTITY column ...
  WHERE
    (EXISTS(foreign) AND (foreign.primary)) OR NOT EXISTS (foreign);
END_ENTITY;
```

Considérons le cas de trois tables A, B, C. C est dépendante de B via une clé étrangère, ainsi que B vis à vis de A. Le code *SQL* est donné ci-dessous.

```
CREATE TABLE A ( idA NUMBER PRIMARY KEY );
CREATE TABLE B ( idB NUMBER PRIMARY KEY, idA NUMBER REFERENCES A );
CREATE TABLE C ( idC NUMBER PRIMARY KEY, idB NUMBER REFERENCES B );
```

Platypus fournit un éditeur de métadonnées. Les métadonnées de l'exemple sont présentées ci-dessous :

```
#100=STRING_TYPE();
#10=TABLE('A',(#11));
#11=COLUMN('idA',#100,.T.,$);
#20=TABLE('B',(#21,#22));
#21=COLUMN('idB',#100,.T.,$);
#22=COLUMN('idA',#100,.F.,#11);
#30=TABLE('C',(#31,#32));
#31=COLUMN('idC',#100,.T.,$);
#32=COLUMN('idB',#100,.F.,#21);
```

A tout moment, on peut vérifier la conformité des métadonnées avec le métamodèle. Selon les règles exprimées précédemment, ce lot d'instance est jugé valide. Cependant, on peut introduire un cycle de dépendance entre clés, par exemple en altérant la table A pour qu'elle dépende de la table C (code *SQL* ci-dessous)

```
ALTER TABLE A ADD idC NUMBER REFERENCES C;
```

Ce qui correspond aux modifications suivantes :

```
#10=TABLE('A',(#11,#12));
#12=COLUMN('idC',#100,.F.,#31);
```

Le lot d'instances est toujours jugé valide pourtant un praticien de *SQL* reconnait l'invalidité de la situation de cycle. Le métamodèle est donc incohérent vis à vis du domaine. Cela nécessite l'introduction d'une nouvelle règle d'intégrité vérifiant la non existence de cycle de dépendance. Cette règle est exprimée ci-dessous à l'aide d'une contrainte globale exécutant une fonction récursive de parcours de l'arbre de dépendances :

```
RULE no_cycle_in_foreign FOR (table);
LOCAL
   visited : SET OF table := [];
END_LOCAL;
WHERE
  SIZEOF(QUERY(tbl <* table | NOT no_cycle (tbl, visited))) =0;
END_RULE;

FUNCTION no_cycle (tested : table; visited : SET OF table): BOOLEAN;
LOCAL
  with_foreign : SET OF column;
  curr : column;
END_LOCAL;
  IF ( tested IN visited ) THEN
    RETURN ( false );
  ELSE
    visited := visited +tested;
    with_foreign := QUERY ( c <* tested.columns | EXISTS(c.foreign));
    REPEAT no := LOINDEX(with_foreign) TO HIINDEX(with_foreign);
      curr := with_foreign[ no ];
      IF NOT ( no_cycle (curr.owner, visited)) THEN
        RETURN ( false );
      END_IF;
    END_REPEAT;
    RETURN ( true );
  END_IF;
END_FUNCTION;
```

Après introduction de cette règle, la vérification invalide le lot d'instances considéré.

4.3 Cycle de validation

La validation met en jeu plusieurs étapes : édition des schémas (métamodèles), analyse *EXPRESS*, construction d'instances (métadonnées), interprétation, production de rapports d'analyse et de résultats de calcul, analyse des rapports par l'expert, évolution des schémas et/ou des instances (cf. figure 5).

Les schémas sont produits grâce à un éditeur structuré, inspiré des browsers de langage objets. Plusieurs vues du(des) schéma(s) sont disponibles : textuelle, organisée par type de construction du langage, structurée par le graphe d'héritage. La figure 6 présente une vue de *Platypus* avec trois espaces de travail superposés. Chaque espace de travail constitue à gauche, d'un arbre comprenant les différents éléments du schéma *EXPRESS* ou du lot d'instances et à droite de l'environnement d'édition et de compilation de l'élément sélectionné.

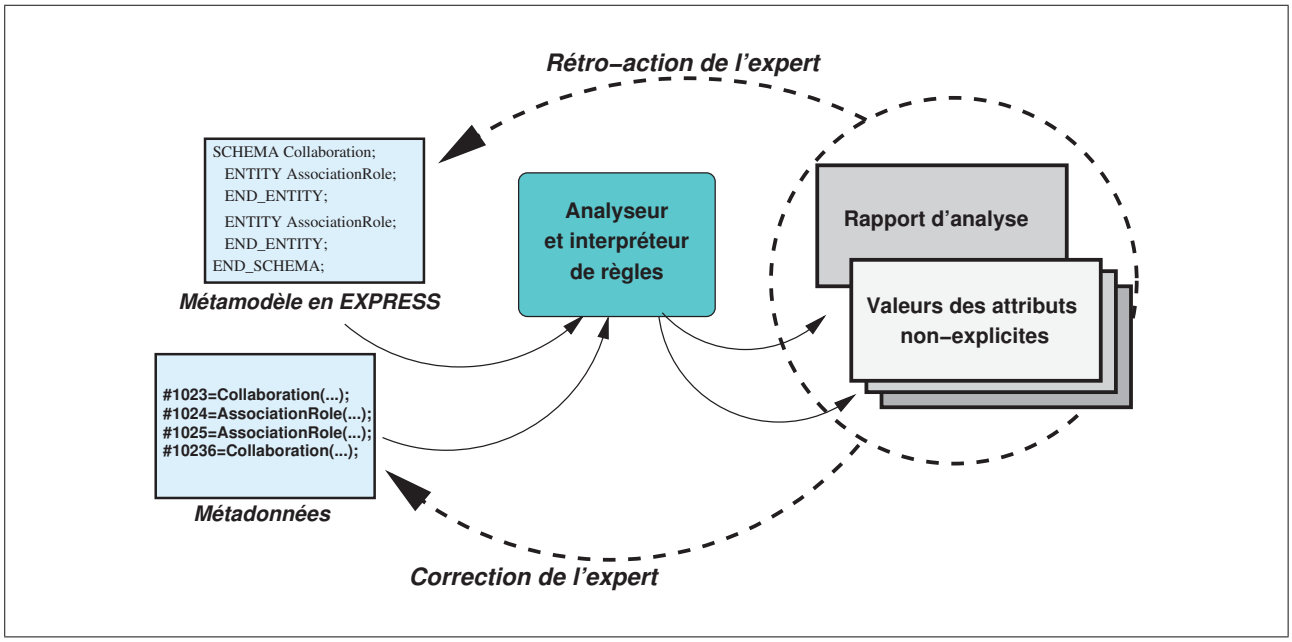

FIG. 5 – Cycle de validation dans *Platypus*

L'analyseur d'*EXPRESS* vérifie la bonne construction des schémas et produit une représentation interne.

Les instances peuvent être construites de plusieurs manières :

- par import d'un fichier d'échange *STEP* ;
- par édition directe dans l'environnement qui fournit une aide minimale à la saisie ;
- tirant partie de l'intégration de *Platypus* dans *Squeak*, par des outils dédiés comme par exemple, des compilateurs produits en partie automatiquement, des passerelles de transformation depuis différents formats, des éditeurs basés sur des dialogues ; ces outils permettent la production de lots d'instances servant de jeux de test pour la validation des métamodèles.

L'interpréteur met en relation la représentation interne et les métadonnées, il vérifie la conformité des instances, évalue les règles et calcule les valeurs des attributs non-explicites.

Un rapport d'analyse trace les erreurs de conformité des instances par rapport aux entités et fournit le résultat des évaluations des règles locales et globales.

A partir de ces rapports et résultats de calcul, l'expert du domaine raffine les schémas, les règles et les procédures de calcul. Cette activité de remodelage s'apparente à celle d'un programmeur qui réorganise son modèle de données à partir d'une revue de ses algorithmes et de leurs résultats.

5 Conclusion

Le domaine de l'ingénierie dirigée par les modèles connait actuellement un regain d'intérêt important notamment autour des travaux de l'*OMG* concernant le *MDA*. La norme *STEP*, standard pour la normalisation de modèles de données par métier a pour objectif de faciliter les échanges de données entre applications et systèmes hétérogènes. Ce standard *ISO* a développé une méthode pour la validation des modèles par les données. Cet article discute de l'utilisation de cette méthode pour les métamodèles : les métadonnées peuvent être exploitées pour la vérification des métamodèles par l'automatisation de l'évaluation des règles d'intégrité et de procédures de transformation spécifiées dans les métamodèles. Cette méthode est outillée par

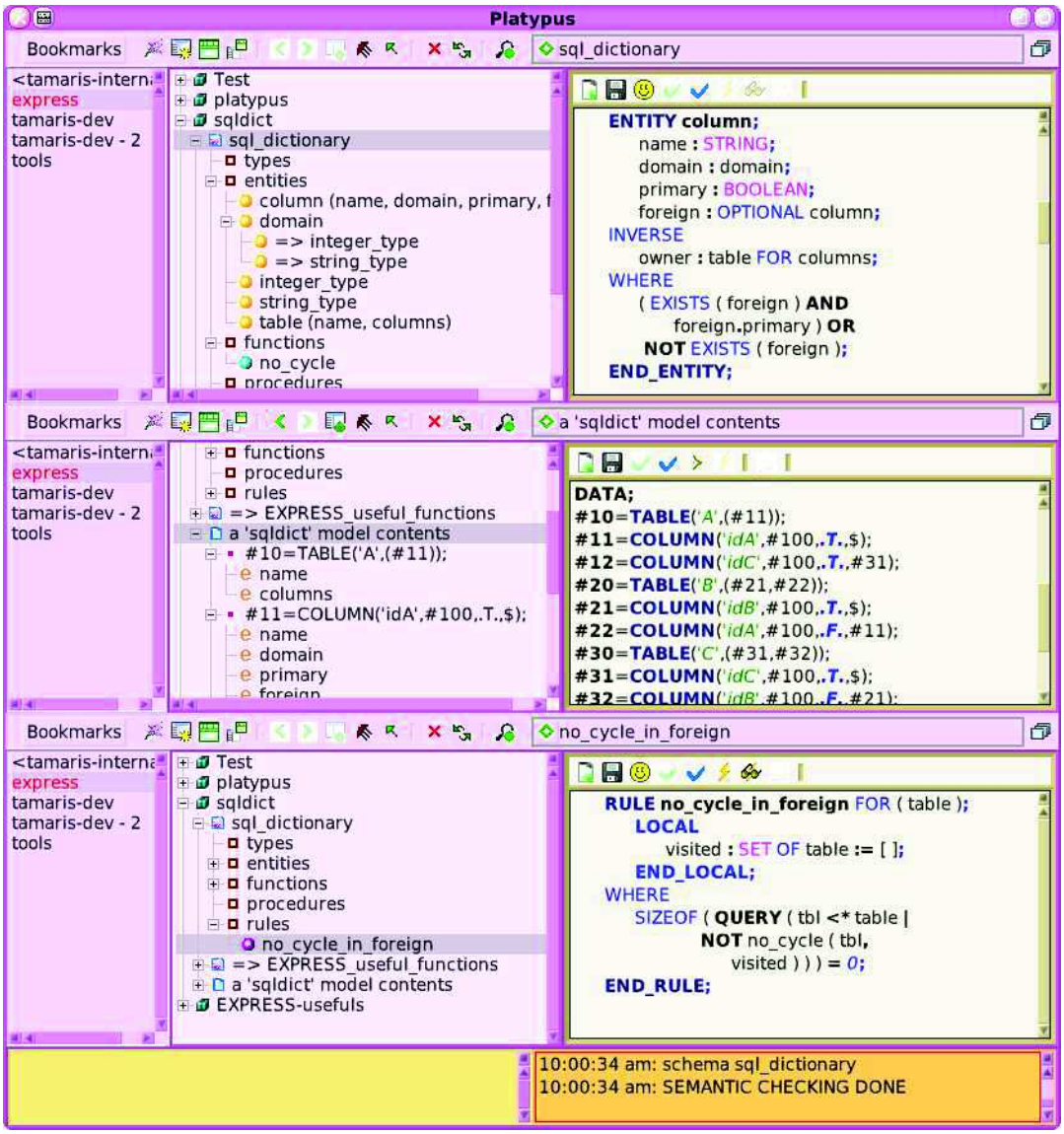

FIG. 6 – Une vue générale des éditeurs de *Platypus*

l'environnement *Platypus* qui intègre une mise en œuvre de *STEP* dans un système *Squeak*.

Références

[1] A. Belangour, J. Bézivin, and M. Fredj. Towards platform independence : a MDA organization. In *Workshop on Information Technology, Rabat, Morocco, March 17-19*, 2003.

[2] Alan Brown. An introduction to Model Driven Architecture - Part I : MDA and today's systems. http ://www-106.ibm.com/developerworks/rational/library, 12 January 2004.

[3] Miguel de Miguel, Jean Jourdan, and Serge Salicki. Practical experiences in the application of MDA. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings*, volume 2460 of *LNCS*, pages 128–139. Springer, 2002.

[4] Erik Duval, Wayne Hodgins, Stuart Sutton, and Stuart L. Weibel. Metadata principles and practicalities. *D-Lib Magazine*, 8(4), 2002.

[5] EBU. EBU tech 3295 - The EBU Metadata Exchange Schema - P_Meta 1.0. http ://www.ebu.ch/fr/technical/publications/tech3000_series/tech3295, 2003.

[6] LE STANDARD INTERNATIONAL STEP ET SON PROTOCOLE D'APPLICATION 214. http ://www.galia.com/.

[7] ISO 10303-1. *Part 1 : Overview and fundamental principles*, 1994.

[8] ISO 10303-11. *Part 11 : EXPRESS Language Reference Manual*, 1994.

[9] ISO 10303-21. *Part 21 : Clear Text Encoding of the Exchange Structure*, 1994.

[10] ISO TC184/SC4/WG6 N29. *Part 31 : Conformance testing methodology and framework : general concepts*, 1992.

[11] metamodel.com. http ://www.metamodel.com, 2005.

[12] Mourad El-Hadj Mimoune, Guy Pierra, and Yamine Ait-Ameur. An ontology-based aproach for exchanging data between heterogeneous database systems. In *ICEIS 2003 : Proceedings of the 5th International Conference On Enterprise Information Systems*, Angers - France, 2003. École Supérieure d' Électronique de l' Ouest.

[13] OMG. Model Driven Architecture. http ://www.omg.org/mda, 2003.

[14] Alain Plantec. *Exploitation de la norme STEP pour la spécification et la mise en œuvre de générateurs de code*. PhD thesis, Université de Rennes I, 35065 Rennes cedex, France, 1999.

[15] Platypus : site internet. http ://cassoulet.univ-brest.fr :8000/squeak, 2004.

[16] P. Selonen, K. Koskimies, and M. Sakkinen. How to make apples from oranges in uml. In *HICSS '01 : Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3*, page 3054, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Squeak : site internet. http ://www.squeak.org, 2005.

[18] Michael Uschold. Where are the semantics in the semantic web ? *AI Mag.*, 24(3) :25–36, 2003.