Spécification de modèles de données orientés objet dans le domaine technique : le langage EXPRESS[1]

Guy Pierra, Yamine Ait-Ameur

Laboratoire d'Informatique Scientifique et Industrielle, ENSMA
B.P. 109 - 86960 FUTUROSCOPE Cedex
Tél. (33) 49-49-80-63 FAX (33) 49-49-80-64 E-mail pierra@ensma.univ-poitiers.fr

RÉSUMÉ : EXPRESS est un langage de spécification pour l'échange de données récemment normalisé. On montre dans cet article que ce langage peut également être utilisé pour spécifier de façon complète, puis développer de façon systématique les "modèles" des systèmes de XAO, ou les bases de données actives

ABSTRACT : EXPRESS is a specification language for data exchange recently published as an International Standard. We argue that this language is well suited for specification and systematic design of CAx system models and active data base.

MOTS-CLES : XAO, modélisation de données, spécifications formelles, BDOO, Base de données active, EXPRESS.

KEY-WORD : CAx, Information Modeling, Formal Specification, OODB, Active Data Base, EXPRESS.

1 *paru dans la* Revue d'Automatique et de Productique Appliquée (RAPA), 1995, n° 8, vol. 2-3, pp. 241-246.

Spécification de modèles de données orientés objet dans le domaine technique : le langage EXPRESS

Guy Pierra, Yamine Ait-Ameur

Laboratoire d'Informatique Scientifique et Industrielle, ENSMA
B.P. 109 - 86960 FUTUROSCOPE Cedex
Tél. (33) 49-49-80-63 FAX (33) 49-49-80-64 E-mail pierra@ensma.univ-poitiers.fr

RÉSUMÉ : EXPRESS est un langage de spécification pour l'échange de données récemment normalisé. On montre dans cet article que ce langage peut également être utilisé pour spécifier de façon complète, puis développer de façon systématique les "modèles" des systèmes de XAO, ou les bases de données actives

ABSTRACT : EXPRESS is a specification language for data exchange recently published as an International Standard. We argue that this language is well suited for specification and systematic design of CAx system models and active data base.

MOTS-CLES : XAO, modélisation de données, spécifications formelles, BDOO, Base de données active, EXPRESS.

KEY-WORD : CAx, Information Modeling, Formal Specification, OODB, Active Data Base, EXPRESS.

1. Introduction

Les langages de spécification formelle sont de plus en plus utilisés, tant pour la conception de logiciel que pour la modélisation des données. Cette dernière application est particulièrement importante dans le domaine de l'ingénierie et de la productique où le développement des systèmes intégrés de production passe par l'intégration de sous-systèmes existants à travers des modèles de données communs permettant le partage et l'échange.

La généralisation de l'approche dite orientée objet pose le problème de la modélisation des données sous un jour nouveau. Suivi à la lettre, le principe de B. Meyer [MEY 88] qui consiste à "mettre le programme dans les données" reviendrait à remettre en cause la possibilité de définir et de spécifier des modèles de données, sans spécifier en même temps les programmes qui les utilisent. Ce manque de séparation est d'ailleurs caractéristique de la plupart des méthodes dites orientées objet [RUM 91] [COA 93].

Confronté à ce problème, au cours de sa démarche visant à définir une nouvelle génération de format d'échange entre systèmes intégrés de production, le projet STEP, en charge du Standard ISO 10303, a été amené à définir un nouveau langage de modélisation des données, le langage EXPRESS [SCH 94]. L'objet de cet article est de montrer à partir d'un exemple significatif, que les caractéristiques de ce langage permettent de l'utiliser, non seulement pour définir un format d'échange, mais également, lors de la conception d'un système de XAO, pour spécifier, sous une forme directement prototypable, la base de données (le "modèle") de ce système, et ce, non pas indépendamment mais *séparément* des programmes qui permettent de l'exploiter.

La présentation est organisée comme suit. On présente d'abord l'exemple traité. Cet exemple étant un cas particulier de graphe, la section 3 présente une spécification générique de la notion de graphe. Cette présentation permet d'illustrer les principales caractéristiques du langage. La section 4 exploite le caractère modulaire d'une spécification EXPRESS pour spécialiser cette ressource, et ébaucher le modèle complet. La section 5 discute le passage à l'implémentation par raffinement de la spécification. On montre, en conclusion, que la séparation formelle effectuée par EXPRESS entre modèle de donnée (actif) et application coïncide avec des propositions récentes sur les bases de données actives.

2. Présentation du cas traité

La méthode PERT est une méthode très utilisée pour réaliser la planification temporelle des projets de grande taille. Elle consiste à identifier les *tâches* qui constituent le projet, les rapports *d'antériorité* entre tâches et les *étapes* significatives du projet. Elle permet alors : (1) de calculer la durée minimum du projet, (2) d'identifier les tâches critiques, (3) de mettre en évidence les "marges" de manœuvre qui existent pour choisir les dates de déroulement des tâches non critiques.

L'exemple ci-dessous montre un exemple très simplifié de réseau PERT correspondant à un projet d'informatisation dont la phase d'analyse est déjà achevée. Chaque tâche contient une durée (estimée) et chaque étape est associée à une date au plut tôt (t_i), depuis le début du projet, à une date au plus tard (T_i), pour ne pas retarder le projet, et une latitude (l_i) correspondant à la marge. Les contraintes d'antériorité sont représentées à la fois par des tâches réelles, et des tâches fictives, de durée nulle (en pointillé). L'analyse du réseau permet de mettre en évidence les événements critiques ($l_i = 0$) et les tâches critiques (en gras).

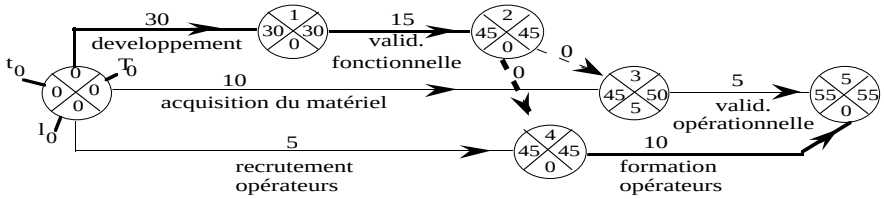

Figure 1 : *Exemple de réseau PERT*

Un réseau PERT étant un cas particulier de graphe, et EXPRESS permettant la spécification modulaire, il paraît judicieux de définir d'abord la ressource générique que constitue un graphe.

3. Représentation d'un graphe

Un graphe G est défini par la description de deux ensembles. Un ensemble fini de sommets X, et un ensemble d'arcs U reliant ces sommets : G = (X,U) ; $U \subset X \times X$. Lorsque (i, j) $\in U$, i est dit un *prédécesseur* de j, et j un *successeur* de i. Il est habituel de noter $\Gamma : X \rightarrow 2^X$, la fonction (multi-application) qui à tout sommet associe ses successeurs, et $\Gamma^{-1} : X \mapsto 2^X$, la fonction qui lui associe ses successeurs. Les trois représentations les plus classiques d'un graphe sont alors définies par (X,U), (X,Γ) ou (X,Γ^{-1}). Ces trois représentations sont équivalentes du point de vue mathématique, mais non du point de vue informatique.

Comme tout concept, cette notion de graphe recouvre en fait trois classes de connaissances. Des connaissances *structurelles*, qui appréhendent le monde à travers des catégories (hiérarchisées) et des relations entre ces catégories. Un graphe comporte des sommets et des arcs, un arc relie deux sommets. Des connaissances *descriptives* qui associent à chaque catégorie les types de propriétés qui la décrivent. Des connaissances *procédurales* qui s'expriment sous forme d'opérations associées à chaque catégorie. On sait, par exemple, passer de la représentation (X,U) à la représentation (X,Γ).

Conformément à l'approche objet, la connaissance structurelle est modélisée en EXPRESS sous forme de hiérarchies (de type) d'entités associées à un mécanisme de factorisation/héritage. EXPRESS suit une approche orientée type [ATK 90] : les types sont définis dans un schéma statique, et il n'y a pas de notion de metaclasse. On peut cependant en travaillant au niveau meta [AIT 95], modéliser le schéma lui-même. Une entité est décrite par des attributs, et une relation s'exprime soit par l'intermédiaire d'attributs, soit en réifiant la relation sous forme d'entité. La principale originalité d'EXPRESS est au niveau de la représentation de la connaissance procédurale. Comme PARDES [ETZ 93], EXPRESS dispose seulement de deux abstractions de haut niveau qui expriment les *invariants des données* [GAL 95], respectivement sous forme fonctionnelle et assertionnelle. La *dérivation* permet de calculer de façon fonctionnelle un attribut à partir des autres éléments du modèle. La *contrainte d'intégrité*, locale (clause WHERE) ou globale (GLOBAL) s'exprime sous forme de prédicat. Un langage de programmation, de type impératif, permet d'exprimer ces invariants à l'aide de toute fonction calculable.

Du point de vue syntaxique, l'ensemble des constructions, qui visent à représenter un univers, sont regroupées dans un SCHEMA. Du point de vue sémantique, ce schéma est associé à un *modèle* qui en constitue le domaine d'interprétation. Un graphe peut, par exemple, se spécifier sous la forme suivante.

```
SCHEMA graph_schema;
ENTITY graph
       sommets       : SET [0:?] OF sommets;
       arcs          : SET [0:?] OF arcs;
END_ENTITY;
```

```
ENTITY arc;                          ENTITY sommet;
 init : sommet;                      DERIVE
 final : sommet;                      succ : SET [0:?] OF sommet
UNIQUE                                :=sommets_finaux(SELF.sortant);
 init, final;                         pred: SET [0:?] OF sommet
INVERSE                               :=sommets_initiaux(SELF.entant);
 dans : graph FOR arcs;              INVERSE
WHERE                                 dans : graph FOR sommets
 même graphe :                        sortant : SET [0:?] OF arc
 (SELF.dans:=:SELF; init.dans)                  FOR init;
 AND                                  entrant : SET [0:?] OF arc
 (SELF.dans:=:SELF.final.dans);                 FOR final;
END_ENTITY;                          END_ENTITY;

FUNCTION sommets_finaux (A: SET OF arc) : SET OF sommet;
LOCAL I : INTEGER; RESUL : SET OF sommet := []; END_LOCAL;
REPEAT I IN LOBOUND(A) TO HIBOUND(A);
   RESUL := RESUL + A.final
END_REPEAT;
END_FUNCTION; -- sommets_finaux
END_SCHEMA; --graph_schema
```

Figure 4 : *Un exemple de modèle EXPRESS pour la notion de graphe*

La clause UNIQUE, contrainte d'intégrité globale au modèle mais exprimable localement, indique qu'il ne peut pas exister deux arcs ayant le même couple d'extrémité. La clause INVERSE permet d'associer à toute relation, la relation inverse. Du point de vue sémantique, il s'agit d'une méthode de dérivation implicite qui évalue, pour chaque instance, l'ensemble des instances qui y font référence dans un rôle donné. Ce lien inverse possède deux utilisations. D'une part, il permet d'exprimer la cardinalité d'une relation inverse (tout sommet appartient à un graphe et un seul). D'autre part, il permet d'accéder à partir d'une instance, par la notation pointée, à toutes les instances qui y font référence (par exemple pour calculer l'attribut "succ"). Enfin des fonctions prédéfinies permettent d'accéder à l'extension des classes d'entités, de les parcourir de façon indicée (LOBOUND...HIBOUND) ou non (clause QUERY), d'accéder au type d'une entité.

4. Modélisation du réseau PERT

Un réseau PERT est une spécialisation d'un graphe qui possède deux caractéristiques :

1 - C'est un réseau : il existe un seul sommet sans prédécesseur appellé entrée, un seul sommet sans successeur, appellé sortie, le graphe est acyclique (on ne peut revenir au même point en suivant le sens des arcs) et connexe (si l'on ne tient pas compte de l'orientation, tous les sommets sont connectés).

2 - C'est un graphe PERT : du point de vue structurel, deux sommets particuliers représentent le début et la fin du projet ; du point de vue descriptif, sommets et arcs sont associés à des attributs sémantiques et géométriques ; du point de vue procédural, pour les sommets intérieurs, les différentes dates peuvent se dériver de celles associées aux prédécesseurs et successeurs.

L'ensemble de ces contraintes est représentable en EXPRESS. La figure ci-dessous présente la modélisation d'un réseau PERT final par spécialisation d'un graphe général. La spécialisation permet : (1) de *spécialiser les attributs* (la notation \ permet de référencer un attribut hérité), (2) de *restreindre les cardinalités* (un réseau a au moins 1 sommet), (3) *d'ajouter des contraintes* (le graphe est un réseau).

```
SCHEMA PERT_final_schema;
REFERENCE FROM graph_schema (graph, sommet, arc);
ENTITY reseau_PERT
SUBTYPE OF graph;
   entré : etape_debut;
   sortie : etpae_fin;
   nœuds : SET [0:?] OF etape_interieur;
DERUVE
SELF\graph.sommet: SET [1:?] OF etape:=[entree] + [sortie] + nœuds;
SELF\graph.arc: SET [0:?] OF tache;
WHERE
    acyclic: is_acyclic (SELF\graph.sommet, SELF\graph.arc);
END_ENTITY;
...
```

Figure 5 : *Spécialisation d'un graphe en un réseau PERT*

Les trois catégories d'étapes (début, intermédiaire et fin) sont alors définies par trois sous-types différents de sommets qui capturent leur différence structurelle, descriptive (toute étape intérieure a au moins un successeur et un prédécesseur) et procédurale (la date au plus tôt de l'entrée se dérive en la constante 0, celle d'un sommet intérieur se calcule par $t_i = \max[t_j + d_{ij}, (i,j) \in U]$. *La totalité des propriétés d'un réseau PERT peuvent donc être capturées dans sa spécification EXPRESS* et ce, *sans que les programmes de création, modification et traitement* du réseau (par exemple pour générer le planning du projet) *ne soient conçus simultanément.*

5. Du modèle à l'implémentation

Un modèle EXPRESS est un modèle d'analyse. Il est bien connu [COA 93] [RUM 91] que la phase de conception modifie un tel modèle. EXPRESS présente néanmoins pour la phase de conception trois caractéristiques intéressantes. (1) La spécification est exécutable : il est possible de générer automatiquement des classes C++ qui prototypent le modèle [SCH 94]. (2) Les modifications à faire subir (performance [COA 91] [RUM 91], états partiellement cohérents,...) peuvent s'exprimer sous forme d'un nouveau schéma EXPRESS dont la comparaison à la spécification initiale permet la validation. (3) Le codage d'une spécification EXPRESS se limite à un transcodage systématique (les WHERE rules sont programmées en invariants de classe levant des exceptions de même nom, les entités associés à une clause UNIQUE sont associées à une entité de collection, les attributs non dérivés, direct ou inverse, définissent les paramètres de la primitive d'instanciation de façon à assurer l'intégrité référentielle, ..). La figure ci-dessous donne le codage Ada de la primitive de création d'un sommet défini par le schéma

graph_schéma (sa structure, de type, *limited private*, se déduit également de façon systématique du schéma final).

```
FUNCTION create_sommet (dans : IN graph;
                        sortant : IN pointeur_liste_arc := NIL;
                        entrant : IN pointeur_liste_arc := NIL)
                        RETURN pointeur_sommet;
```

Figure 6 : *Codage en Ada de la spécification*

6. Conclusion

Nous avons, dans cet article, montré comment le langage EXPRESS pouvait être utilisé pour spécifier, de façon très complète, puis concevoir et développer de façon systématique un "modèle" de données, orienté objet, pour un système technique. A la question de savoir ce qui, dans un système objet, relève du modèle de données et ce qui relève de l'application, EXPRESS apporte une réponse formelle : dans un modèle de données orienté objet la connaissance procédurale qui relève du modèle de donnée est celle qui se formalise en termes de méthode de dérivation d'attribut, ou en termes de contrainte, locale ou globale, d'intégrité. Notons que cette réponse coïncide avec une nouvelle approche proposée [ETZ 93] [GAL 95] pour la spécification des *bases de données* dites *actives.*

[AIT 95] AIT-AMEUR Y., BESNARD F., GIRARD P., PIERRA G., POTIER JC., *Formal Specification and Metaprogramming in the EXPRESS Language* , Software Engineering and Knowledge Engineering SEKE'95 (IEEE - ACM Sigsoft), Rockville, USA, June 1995.

[ATK 90] ATKINSON M., BANCILHON F., DITTRICH K., MAIER D., ZOO NIK S., *The Object-Oriented Database System Manifesto. Deductive and Object-Oriented Databases,*, Elsevier Science Publishers B.V. (North-Holland), pp. 223-240, 1990.

[COA 93] COAD P., YOURDON E. *Conception Orientée Objet*, Prentice-Hall, 1993.

[ETZ 93] ETZION O. *Pardes - A Data-Driven)Oriented Active Database Model*, SIGMOD Record, Vol. 22, n° 1, pp. 7-14, March 1993.

[GAL 95] GAL A., ETZION O. *Maintaining Data-Driven Rules in Databases*, IEEE, pp. 28-38, Juanuary 1995.

[MEY 88] MEYER B. *Object-oriented Software Construction*, Prentice-Hall International series in Computer Science, Prentice-Hall, Hemel Hempstead (1988).

[RUM 91] RUMBAUGH J. *Object-Oriented Modeling and Design*, Prentice-Hall International Editions, 1991.

[SCH 94] SCHENCK D., WILSON P. *Information Modeling the EXPRESS Way*, Oxford University Press, 1994.