Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Ecole Doctorale des Sciences Pour l'Ingénieur ▪ Université de Poitiers

THESE

Pour l'obtention du grade de
DOCTEUR DE L'UNIVERSITE DE POITIERS
(Faculté des Sciences Fondamentales et Appliquées)
(Diplôme National — Arrêté du 25 Avril 2002)

Ecole Doctorale : Science Pour l'Ingénieur
Secteur de Recherche : INFORMATIQUE et APPLICATION

Présentée par :

Mourad EL HADJ MIMOUNE

****

Contribution à la modélisation explicite et à la représentation des données de composants industriels : application au modèle PLIB

****

Directeurs de Thèse : ***Guy Pierra, Yamine Ait-Ameur***

Soutenue le : 12 Juillet 2004

Devant la commission d'Examen

JURY

Rapporteurs :	Anne-Françoise Cutting-Decelle	Professeur, Université d'Evry Val d'Essonne, Evry
	Parisa Ghodous	Maître de Conférences, HDR, Université Claude Bernard Lyon1, Lyon
Examinateurs :	Yves Bertrand	Professeur, Université de Poitiers, Poitiers
	Danielle Boulanger	Professeur, Université Jean Moulin Lyon 3, Lyon
	Yamine Ait-Ameur	Professeur, Université de Poitiers, Poitiers
	Guy Pierra	Professeur, ENSMA, Futuroscope

Je souhaite, avant toute chose, dédier ce mémoire
à ma famille pour m'avoir soutenu et permis de le réaliser
ainsi qu'à ma femme, pour son amour et son soutien
dans les moments difficiles et de doute que rencontre tout doctorant.

Remerciements

Par l'intermédiaire de ces quelques lignes, je tiens à adresser mes plus sincères remerciements à Guy Pierra pour m'avoir accueilli au sein de son laboratoire et pour m'avoir permis de mener à bien ces travaux. Je le remercie pour ses conseils techniques et théoriques, ainsi que pour son encadrement tout en m'accordant une grande confiance et une souplesse de manoeuvre.

Je tiens également à remercier Yamine Ait-Ameur pour son encadrement étroit de ce travail, sa grande disponibilité et pour ses conseils avisés.

J'adresse également mes remerciements aux membres du jury, Yves Bertrand pour avoir accepté de présider le jury, Danielle Boulanger pour avoir accepté de faire partie de ce jury, mais également à Anne-Françoise Cutting-Decelle et Parisa Ghoudous pour les remarques critiques et constructives qu'elles ont observées et pour le temps précieux qu'elles ont consacré en tant que rapporteurs de cette thèse.

Je remercie Eric Sardet et Jean Claude Potier pour leurs aides et conseils.

Que serait une thèse sans les collègues de bureau et autres membres du laboratoire pour animer les pauses et les repas. Aussi je remercie tous les membres du laboratoire, et plus particulièrement les deux Laurent, Fabrice, Dago, Ladjel, Frédérique, Manu et les deux Michael pour tous les bons moments passés en leur compagnie.

Merci à ma mère, mon père, mes grand parents et ma femme pour leur soutien permanent.

Merci également à tous ceux qui m'ont aidé et soutenu de près ou de loin pour réaliser ce travail.

Table des matières

Glossaire

API	Access Programming Interface
BSU	Basic Semantic Unit
CAD-LIB	Computer Aided Design-LIBrary (project)
CAO	Conception Assistée par Ordinateur
CFAO	Conception et Fabrication Assistée par Ordinateur
FAO	Fabrication Assistée par Ordinateur
GDT	Gestion des Données Techniques
GMAO	Gestion de Maintenance Assistée par Ordinateur
GPAO	Gestion de Production Assistée par Ordinateur
IAO	Ingénierie Assistée par Ordinateur
IGES	Initial graphical Exchange System
KADS	Knowledge Acquisition Design Systems
LDD	Langage de Définition de Données
LMD	Langage de Manipulation de Données
NBS	National Bureau of Standard
PL/SQL	Procedural Language/Structured Query Language
PLIB	Parts LIBrary
SET	Standard d'Echange et de Transfert
SGBD	Système de Gestion de Bases de Données
SGBDR	Système de Gestion de Bases de Données Relationnelles
SGBDRO	Système de Gestion de Bases de Données Relationnelles-Objet
SGDT	Système de Gestion de Données Techniques
SI	Système d'Information
SQL	Structured Query Language
STEP	STandard for Exchange of Product model data
VDA-FS	Verband der Automobilindustrie Flaechen-Schnittstelle - Grafikstandard
XAO	Désigne tout ce qui peut être assisté par ordinateur (AO)

Introduction générale

Motivation

L'évolution qu'a connue l'informatique au cours des dernières décennies, en particulier à travers la programmation orientée objet, a permis le développement de nouveaux modèles de données et de nouveaux outils capables de représenter et de gérer les données informatiques décrivant les produits complexes dans les entreprises industrielles. Ces données peuvent être classées en deux catégories. Les données de produits visent les produits conçus et/ou fabriqués dans les entreprises industrielles. Les descriptions qu'elles représentent permettent de donner une vision analytique d'un produit nécessaire au suivi de son développement. Ces données accompagnent donc le produit depuis sa description sur des plans et modèles jusqu'à sa mise en service. Mais, beaucoup de produits contiennent des assemblages de produits préexistants. Pour l'entreprise utilisatrice, la description de ces produits préexistants est différente de celle utilisée dans l'entreprise qui les produit. Il ne s'agit pas de fabriquer ces produits mais de les retrouver, de les comparer et de les choisir. Pour l'entreprise utilisatrice, ces produits réutilisables sont appelés *composants*. Les données décrivant ces composants constituent la seconde catégorie des données techniques. Elles sont appelées données de composants. Ainsi, deux modèles de données techniques distincts ont été développés pour représenter chacune des catégories citées ci-dessus.

Dans le domaine des composants, les travaux de recherche lancés par le projet européen CAD-LIB ont abouti au développement d'une nouvelle norme de modélisation des catalogues électroniques de composants préexistants. Il s'agit de la norme PLIB (ISO-13584). Le modèle PLIB a été développé de manière à atteindre les objectifs suivants : représenter de façon compacte des données de composants, faciliter leur échange entre fournisseurs et concepteurs et permettre la comparaison des composants similaires en vue de leur insertion dans un produit particulier. Deux parties ont été développées dans le modèle PLIB afin qu'il réponde aux objectifs cités auparavant : le modèle de dictionnaire et le modèle de contenu. Le modèle de dictionnaire permet la représentation de l'ontologie du domaine visé en décrivant les concepts de ce domaine (classes = familles de composants, attributs = propriétés de composants, …) et en caractérisant de façon unique et non ambiguë chacun de ces concepts par un identifiant universel. Son existence facilite la comparaison des composants similaires à travers la comparaison de valeurs de leurs propriétés ("*modélisation par propriétés*"). Quant au contenu, il définit une structure permettant la représentation implicite des instances existant pour chaque famille décrite au niveau du dictionnaire en vue de leurs échanges entre systèmes hétérogènes. Comme dans les catalogues techniques, les instances sont décrites implicitement par des contraintes et propriétés afin de rendre la représentation compacte et de faciliter la sélection.

Concernant les données de produits, les différentes données techniques représentant les produits durant leur cycle de vie et selon différents points de vue sont gérées et stockées dans des systèmes de gestion de données techniques (SGDT). Ces données ont la particularité d'être explicites : chaque produit est *individuellement et explicitement décrit*. De plus, elles contiennent de nombreuses représentations, chacune est créée par un système particulier (CAO, outils de calculs éléments finis, …) qui est le seul à pouvoir les interpréter. Pour ce faire, les SGDT permettent la gestion des contenants de ces représentations et l'encapsulation des contenus. Ils gèrent également l'organisation des structures de produit et leurs nomenclatures (décomposition récursive d'un produit en sous-ensembles jusqu'à atteindre les composants élémentaires). La gestion des contenants concerne la description de toutes ces représentations par des attributs (date de création d'un fichier CAO, nom du créateur, date de modification, …..) ainsi que l'établissement de liens entre les différentes données techniques (lier un dessin de définition d'un sous-ensemble au sous-ensemble lui-même, par exemple). Quant à l'encapsulation des contenus, elle est relative à la gestion, sous forme de fichiers informatiques totalement soumis au contrôle du SGDT, de tous les documents : plans CAO et différentes représentations d'un produit. Ce système de contrôle est appelé «coffre fort ». La moindre modification apportée à n'importe quelle donnée technique nécessite le passage par une procédure de modification (gestionnaire de tâches ou *workflow*) et est soumise à une approbation. En ce qui concerne la gestion de la nomenclature, STEP et les SGDT doivent garantir à chaque intervenant que sa vision du produit est présente. Ainsi, plusieurs structures de produit sont gérées par celui-ci permettant la représentation de la nomenclature de produit pendant une phase particulière de son cycle de vie.

Comme pour les données de composants qui ont fait l'objet d'un développement normatif PLIB, des modèles de produits normalisés ont été développés avec le standard STEP facilitant ainsi la représentation et l'échange des données de produits entre les différents SGDT utilisés par des partenaires développant, par exemple, un même produit.

Le modèle de données PLIB et le modèle des SGDT et a fortiori de STEP sont voisins car ils traitent tous les deux des données de produits et sont utilisés dans le même domaine d'application. La différence majeure entre les deux modèles est que le premier représente les produits d'une manière implicite, c'est-à-dire par des contraintes sur un ensemble de propriétés, et le second les représente d'une manière explicite, c'est-à-dire en utilisant un identifiant par produit (ou assemblage), auquel sont associés des attributs et des représentations. La représentation des composants par propriétés vise à favoriser la sélection et le choix des composants en vue de les utiliser dans un contexte particulier. En revanche, la modélisation explicite des produits par des identifiants permet la modélisation des produits au cours de leur développement depuis la période qui précède leur existence (étude de marché), en passant par la phase où ils n'existent que sur papier, dans des plans ou des modèles informatiques jusqu'à leur production et leur mise en service.

Malgré la similitude du domaine d'application des deux modèles et l'utilisation simultanée des données de produits et des données de composants dans les entreprises industrielles, la structure des deux modèles est très différentes et il n'est facile (1) ni de référencer un composant à partir d'un modèle de produit, (2) ni de gérer dans un même univers informatique données de produit et données de composants.

- Du point de vue de référencement, un composant n'existe, dans le modèle implicite de représentation de PLIB, que comme le résultat d'un processus de sélection. Il n'existe pas a priori dans le modèle. Il est donc difficile pour le SGDT de le référencer.

- Du point de vue de la gestion des données, les données de produits sont décrites de façon structurelle (entités) et descriptive (attribut) correspondant à la structure classique d'une base de données. Les données de composants sont décrites sous une forme implicite et procédurale (contraintes, fonctions, …) qui relève d'un système de gestion spécifique.

De ce fait, l'intégration des deux modèles parait naturelle et indispensable. C'est précisément l'objectif que nous poursuivons.

Contribution de la thèse

L'objectif des travaux de recherche amorcés par cette thèse est d'intégrer le modèle de données PLIB au sein d'un SGDT et de définir un processus d'intégration favorisant le passage d'une manière automatique d'un modèle à un autre et plus précisément le transfert du modèle et des instances PLIB dans le modèle du SGDT cible.

Comme les SGDT sont toujours basés sur des SGBD, le problème se ramène pour une grande partie au choix d'une stratégie d'implémentation du modèle PLIB au sein d'un SGBD "traditionnel". Ceci nous a conduit à créer un gestionnaire de données de composants basé sur un SGBD et à étudier l'intégration et le référencement des composants à partir de ce gestionnaire et des données de produits gérés par un SGDT.

A partir de l'étude des différentes approches d'intégration possibles, *une nouvelle représentation des instances des contenus des catalogues a été développée*. Cette représentation consiste à décrire de manière explicite ces instances ; c'est-à-dire par une énumération exhaustive des valeurs de leurs propriétés et non par des contraintes et des fonctions comme c'est le cas dans le modèle de représentation implicite.

L'implémentation du modèle de données PLIB dans une base de données cible a abouti au *développement d'une nouvelle architecture de représentation de données dans cette base de données*. Elle vise à représenter au sein d'une même base de données la description ontologique d'un domaine (description des concepts du domaine et leur identification universelle) et les données concernant les instances de ce domaine.

En ce qui concerne la mise en œuvre, notre travail s'est concentré sur le développement des approches d'intégration automatiques basées sur la technologie EXPRESS. Ainsi, deux approches de génération de code ont été étudiées et mises en œuvre : la génération de code basée sur la compilation des modèles de données EXPRESS et la programmation événementielle appliquée au langage EXPRESS. La génération de code basée sur les techniques de compilation utilise le principe des grammaires hors contexte. La programmation événementielle appliquée au langage EXPRESS est basée sur le principe des grammaires attribuées pour générer les

programmes de bases de données cibles. Ce type de programmation a été réalisé en utilisant les attributs dérivés du langage EXPRESS qui permettent chacun de calculer un fragment du code final généré. Ceci a permis de mettre en évidence une caractéristique très importante du langage EXPRESS : le fait de pouvoir intégrer dans un modèle de données EXPRESS une méthode permettant son implémentation au sein d'une base de données cible (ou autres modèles de données cibles).

Organisation de la thèse

Nous structurons ce mémoire de la manière suivante. Tout d'abord, dans le premier chapitre, nous définissons les deux grandes approches de modélisation des données de produits à savoir la modélisation explicite par représentation et la modélisation implicite par propriétés. La première est utilisée pour la modélisation de produits (pendant tout leur cycle de vie) et la seconde pour la modélisation des composants. Comme les données de produits sont gérées par des SGDT, nous présentons ces systèmes ainsi que l'organisation et la structuration de ces données. Le modèle PLIB représentant les données de composants est également présenté dans ce chapitre. Ce modèle est constitué de deux parties. Le dictionnaire de données contient l'ontologie (les classes, les attributs et leurs identifications universelles) du domaine représenté. Le contenu définit les instances du domaine conformément à la description ontologique. Nous discutons également dans ce premier chapitre les différents modèles de base de données et en particulier les SGBDRO qui seront utilisés par la suite pour implémenter le modèle PLIB dans une base de données cible.

Dans le chapitre 2 nous présentons l'approche implicite de représentation des contenus des données de composants. C'est l'unique approche qui existait au démarrage des travaux de cette thèse. Elle consiste à représenter les instances par des expressions et des contraintes d'intégrité. Dans cette approche, les instances n'ont une existence effective qu'à l'issue du processus de sélection. Puis, nous détaillons les trois approches étudiées pour l'implémentation du modèle de données PLIB ainsi défini dans une base de données cible. La première de ces approches vise à implémenter le modèle PLIB au niveau des entités du modèle EXPRESS, c'est-à-dire une traduction systématique du modèle PLIB au sein de la base de données cible (avec une représentation des instances du contenu par des contraintes et des expressions au sein de la base de données cible). Ceci génère une base de données complexe. Pour pallier ce problème, nous avons étudié une deuxième approche qui consiste à extraire à partir de l'ontologie le schéma de la base de données cible sans représenter l'ontologie et puis à représenter explicitement chaque instance de composants comme un tuple. Cette dernière est appelée l'implémentation au niveau des instances. Finalement, nous avons étudié une troisième approche dite hybride. Celle-ci vise, d'une part, à représenter la description ontologique du domaine au sein de la base de données cible comme dans la première approche, et, d'autre part, à représenter des contenus comme dans la seconde approche. Ainsi, le dictionnaire de données PLIB (le modèle de l'ontologie) sera implémenté au niveau du modèle EXPRESS et le contenu (les instances) seront implémentés comme des tuples.

Dans le chapitre 3, nous étudions la conversion de données conformes au modèle implicite PLIB en données conforme au modèle explicite. Cette conversion nécessite principalement des évaluations d'expressions car celles-ci sont la base de la

représentation implicite. Cette conversion permet donc de générer les instances du contenu en calculant les valeurs de leurs propriétés. Nous appliquons ensuite des filtres (contraintes logiques) pour éliminer les instances superflues ne répondant pas aux contraintes définies dans le modèle. Nous discutons finalement, dans ce chapitre, de l'implémentation de la connaissance procédurale représentée dans le modèle PLIB au sein d'une base de données cible. Deux approches sont étudiées : la représentation interne (au sein de la base de données) et la représentation externe (au sein d'un applicatif) de cette connaissance.

Dans le chapitre 4, nous détaillons le modèle de représentation explicite des données de composants. Dans cette représentation, toutes les instances sont énumérées de manière exhaustive. Chaque instance y est représentée par une liste de couples (propriété, valeur). Nous détaillons également dans ce chapitre les deux méthodes d'implémentation étudiées. Il s'agit, en fait, de méthodes de génération de code SQL correspondant au modèle PLIB. La première est la méthode de génération de code utilisant l'analyse syntaxique basée sur les grammaires hors contexte. La seconde est la méthode de génération de code utilisant la programmation événementielle basée sur les grammaires attribuées.

Finalement, nous étudions dans le chapitre 5 une mise en œuvre des approches présentées dans les chapitres précédents à travers l'implémentation d'une maquette dans le SGBDRO PostgreSQL. Nous précisons les correspondances choisies entre les concepts du langage EXPRESS et du modèle PLIB, d'une part, et le langage SQL de PostgreSQL, d'autre part. Nous discutons également dans ce chapitre les deux approches de connexion possible entre un gestionnaire PLIB et un SGDT. Dans la première approche les données de produits et les données des composants sont représentées dans une seule base de données intégrée. Dans la seconde approche les données de produits et celles des composants sont représentées dans deux bases de données distinctes. Dans ce cas, la connexion est réalisée au travers de références externes représentés par des identifiants universels. Nous terminons ce mémoire par une conclusion générale et par les perspectives ouvertes par les travaux de cette thèse.

Chapitre 1

Modélisation et gestion de données de produits et de composants

Résumé. *Dans les différents domaines industriels, le patrimoine informationnel constitue un capital très important pour les entreprises. Il comprend toutes les données, plans, représentations, etc. axés autour du produit. Ces informations permettent la description du produit. Elles sont appelées les données techniques génériques de produit. Elles peuvent être classifiées selon deux visions. La vision analytique permet la modélisation du produit en terme de représentation donnant chacune une vue de ce produit dans une phase particulière de son cycle de vie (conception, production, maintenance, ...) et selon un point de vue particulier (études, méthodes, ...). La vision synthétique du produit permet la modélisation du produit par des propriétés en vue de faciliter sa sélection pour l'utiliser dans la conception d'autres produits. Un produit ainsi réutilisable est également appelé un composant. On distingue donc parmi les données techniques génériques les données techniques de produit correspondant à la vision analytique et les données techniques de composants correspondant la vision synthétique. Les données techniques de produits sont gérées par des systèmes développés pour cette fin, appelés Systèmes de Gestion de Données Techniques (SGDT). Le principe de l'entreprise étendue et le développement de l'ingénierie concurrente ont augmenté les besoins d'un échange de données de ce type. La norme STEP a précisément été développée pour décrire des modèles standards pour ces données et offrir un format neutre d'échange. Les données techniques de composants ont également fait l'objet d'une normalisation dans la norme PLIB. Nous présentons dans ce chapitre les deux grandes approches de modélisation de données, à savoir, la modélisation par représentations et la modélisation par propriétés. Nous discutons également la convergence entre les deux modèles de données et la nécessité de leur intégration. Etant donné que les SGDT sont des systèmes basés sur des SGBD, l'implémentation de PLIB dans un SGBD faciliterait énormément l'intégration de ces deux modèles.*

1 Introduction

L'usage de l'informatique dans les domaines techniques a nécessité l'élaboration de différents modèles. Ces modèles permettent d'approcher le monde technique en général, et les produits en particulier, selon différents points de vue. Ces modèles permettent la description des données techniques génériques des produits. Toutefois, les produits peuvent être perçus selon deux grands aspects. D'une part, la vision analytique du produit permet sa description par des représentations en utilisant

différents modèles techniques (géométriques, éléments finis, …) [Pierra 93] dans les différentes phases de son cycle de vie (c'est-à-dire pendant son développement). Chaque représentation offre une vue particulière du produit. Les données techniques génériques correspondant à cette vision sont appelées données techniques de produits. Notons qu'il s'agit là d'une approche "produit" et non pas d'une approche "production" (cette dernière traite la gestion de process, l'élaboration de plan directeur de production, …). D'autre part, la vision synthétique du produit permet de le décrire en terme de propriétés favorisant sa comparaison avec d'autres produits semblables et donc sa sélection en vue de son utilisation pour concevoir et fabriquer un autre produit (c'est-à-dire après son développement). Les données techniques génériques correspondant à cette seconde vision sont appelées données techniques de composants.

La modélisation de produit permet de définir un modèle informatique (traitable par ordinateur) d'un produit. [Minsky 86] définit un modèle comme suit :

pour un observateur B, un objet M est un modèle d'un objet A s'il aide B à répondre aux questions qu'il se pose sur A.

Un modèle est une représentation de l'univers de discours selon un point de vue particulier. Cette représentation du monde réel peut être faite à des niveaux d'abstraction plus au moins élevés selon l'utilisation que l'on veut faire de ces modèles. Dans ce qui suit, nous détaillerons les deux possibilités de modéliser les produits (par des représentations et par des propriétés) et nous parlerons des différents niveaux d'abstraction pour ces modèles.

Ce chapitre est présenté de la manière suivante. Dans la partie 2 nous présentons la première approche de modélisation des données techniques génériques de produits. Il s'agit de l'approche de modélisation par représentation. Celle-ci correspond à la vision analytique des produits. Nous donnons d'abord les définitions de quelques concepts utilisés dans le domaine de gestion de données techniques de produits (désignées également par données de produits). Puis, nous discutons des problèmes liés à l'échange de ces données entre les différents acteurs et des normes et standards développés pour faciliter cet échange. La plus importante de ces normes est la norme STEP (ISO-10303). Celle-ci sera décrite donc avec plus de détails dans cette partie. Nous mettons l'accent plus précisément sur le schéma de gestion de données techniques (GDT) de STEP. Dans la deuxième partie nous présentons la seconde approche de modélisation des données techniques génériques de produits à savoir la modélisation par propriétés. Celle-ci correspond à la vision synthétique des produits. Nous présentons d'abord l'approche de modélisation par propriétés d'une manière générale. Puis nous discutons de l'importance des données techniques de composants (désignées également par données de composants) dans les Systèmes d'Information (SI) des entreprises et la nécessité de leur modélisation. Nous discutons également de l'importance de l'intégration de ces données entre des systèmes hétérogènes et les problèmes qui peuvent bloquer cette intégration. Ceci nous amène donc à justifier le choix qui a été porté à l'approche de modélisation par propriétés pour modéliser les données de composants. Dans la partie 4 nous présentons le modèle de données de composants PLIB (ISO-13584). En effet, il s'agit d'une norme internationale basée sur l'approche de modélisation par représentation. Elle apporte des réponses concrètes aux problèmes d'intégration évoquées ci-dessus. Finalement, dans la partie 5 nous nous

discutons de l'intégration entre les données de produits et les données de composants. Nous présentons les convergences et les divergences entre les deux modèles et l'approche d'intégration adoptée. Celle-ci passe par une implémentation du modèle de données PLIB dans un SGBD. C'est pour cette raison que différents SGBD sont présentés pour le choix du plus avantageux pour réaliser cette intégration. En conclusion, pour intégrer les données de produits et les données de composants, nous avons adopté une stratégie basée sur le développemnt d'une architecture de base de données implélentant le modèle PLIB. Le modèle de la base de données choisi pour cette implémentation est le modèle relationnel-objet.

2 Modélisation de produits basée sur des représentations

2.1 Introduction

Le développement de nouveaux produits passe par plusieurs phases et nécessite l'intervention de plusieurs métiers. La perception du produit dans chaque phase de son cycle de vie est différente. Les informations sur les produits sont représentées sous différents formats selon les outils informatiques que l'on utilise lors de chaque phase. Elles permettent, entre autres, de répondre à certaines questions que l'on se pose sur le produit (quelle est la forme du produit, quel est le modèle de calcul de sa structure, quelles sont les étapes de sa fabrication, …). Chaque vue de produit est, en fait, perçue à travers un certain nombre de représentations. Chacune de ces représentations est créée en utilisant des outils spécifiques et en dépend directement.

2.2 Données techniques de produits

2.2.1 Définition des données techniques de produits

Les données techniques de produits sont les données produites par les systèmes informatiques spécifiques des études, de l'ingénierie, de la fabrication, de la gestion des projets, de la gestion de qualité, etc. Elles présentent la caractéristique très particulière d'être chacune associée à un outil particulier (exemple : système CAO, FAO, code de calcul éléments finis, …) qui, seul, permet d'en interpréter le contenu. Elles sont donc hétérogènes. Parmi plusieurs types de données techniques, on peut citer les données CAO, IAO (Ingénierie Assistée par Ordinateur), GPAO (Gestion de Production Assistée par Ordinateur), GMAO (Gestion de Maintenance Assistée par Ordinateur), etc. Il entre dans cette définition la nomenclature de produit (décomposition itérative d'un produit en composants et assemblages). Elle constitue une structure de base sur laquelle viennent s'ajouter les différentes représentations décrivant chaque composant ou assemblage. Les données techniques d'un produit peuvent être définies de la manière suivante : il s'agit d'une structure arborescente représentant la composition d'un produit et servant de support à l'organisation de toutes les données de produit.

Les données techniques constituent le patrimoine informationnel d'un produit depuis sa conception jusqu'à sa destruction [Randoing 95]. Elles décrivent les produits par des représentations donnant chacune une définition particulière d'une vue de

produit durant tout son cycle de vie. Cette description est réalisée par le rattachement de ces données à une structure arborescente (nomenclature) du produit. Nous définissons dans le paragraphe suivant les différents concepts qui rentrent dans la description de cette structure.

2.2.2 L'exemplaire, l'article et l'objet technique

L'exemplaire est un objet physique utilisé dans la composition d'un produit. Chaque exemplaire possède un identificateur appelé généralement numéro de référence ou de numéro de série. L'exemplaire, appelé également occurrence dans PLIB représente le composant utilisé dans un contexte particulier. Un roulement sélectionné, pour être inséré dans un assemblage précis par exemple, est une occurrence.

L'article est une abstraction de l'exemplaire. Il représente un élément dans la structure représentant un produit. Cette structure est appelée nomenclature. L'article représente un composant interchangeable ; une vis parmi l'ensemble des vis sans tête par exemple.

La norme française AFNOR [NF X 60-012] définit l'article de la manière suivante :

"l'article est un bien identifié en tant que tel, constituant un élément de nomenclature ou de catalogue".

L'interprétation de la définition diffère d'une fonction à une autre dans une entreprise. Dans les bureaux d'études, l'article est considéré comme étant un composant physique ou logique d'un produit. Par exemple, une vis, un assemblage boulonné ou bien un module logiciel sont tous des articles. En fait, chaque décomposition du produit représentant un point de vue technique pertinent du produit est un article [Maurino 93].

Dans la phase de production, l'article est considéré comme étant tout élément impliqué dans la définition du processus de production. Ainsi, tout composant, sous-ensemble matière, outillages, éprouvettes, … qui sont nécessaires à la fabrication du produit sont des articles.

Nous avons précisé ci-dessus que les articles constituent des éléments de la nomenclature. La structuration des produits ne se limite pas aux articles mais elle s'étend aux documents décrivant les articles. La nomenclature peut donc être définie comme une structuration de l'ensemble des articles et des documents qui les décrivent. Le concept employé pour structurer les différentes vues de produits est celui d'objet technique. Un objet technique est un élément constitutif d'une nomenclature de produit [Maurino 93]. Il désigne ainsi aussi bien les articles que les documents et constitue une abstraction de ces deux éléments.

2.2.3 Les dossiers techniques d'un produit

Nous avons parlé au début de ce chapitre de la vision analytique donnée par la modélisation par représentations. En effet, cette vision vient du fait même de la structuration des données techniques en objets techniques (articles et documents), liens de composition et de représentation, etc. Le rassemblement de ces éléments dans des ensembles cohérents (dossiers) permet de les traiter d'une façon naturelle en situation opérationnelle et donne cette vision analytique du produit [Randoing 95]. Un dossier technique donne une vue particulière des données techniques du produit. Le dossier technique d'un article, appelé également dossier de définition, peut regrouper, par exemple :

- Un sous-ensemble des propriétés de l'article (identification, définition),
- la nomenclature d'étude de l'article (liens de composition),
- l'ensemble des documents de définition attachés à l'article,
- les informations concernant la gestion du dossier lui-même (autorisation d'accès, statut, …).

Un dossier technique peut contenir également d'autres dossiers si un article contient des sous articles (les dossiers techniques de chaque composant de l'article) de sorte que la structure de composition définit la structure de l'ensemble des documents qui leur sont attachés. Des modèles génériques pour les dossiers techniques d'un produit sont créés dans le Système de Gestion de Données Techniques (Dossiers de définition, de fabrication, de maintenance, …). Les différents dossiers du produit sont donc obtenus par instanciation de ces modèles génériques.

2.2.4 Echange et partage de données techniques

Les innovations informatiques ont beaucoup aidé les entreprises à réduire la lourdeur de gestion et de manipulation des documents en papier. De plus, l'utilisation des outils informatiques de XAO a facilité la tâche des développeurs. Ces derniers gagnent du temps en réutilisant des parties des dessins techniques pour représenter un autre dessin ou bien pour passer automatiquement, par exemple, d'un modèle 3D à un modèle éléments finis.

La représentation des données CAO, par exemple, diffère d'un système à un autre. L'échange de données entre différents systèmes pose des problèmes de compatibilité à différents niveaux: géométrie, sémantiques, …[Bouazza 95b]. Cet échange devient d'autant plus complexe que les modèles mathématiques utilisés pour représenter des entités géométriques sont différents.

Le principe de l'entreprise étendue, qui a émergé ces dernières années, et le développement de la sous-traitance (externalisation) ont donné plus d'importance à l'échange et au partage de données de produits entre les différents partenaires. Un sous-traitant fournit aux donneurs d'ordre non seulement le produit mais également les

différents plans et documents décrivant ce produit. Cela est dû à l'utilisation intensive des différents outils informatiques pour représenter ces produits (modélisation 2D, 3D, calcul éléments finis, simulation cinématique, …).

Généralement, les partenaires n'utilisent pas les mêmes systèmes informatiques pour modéliser leurs données. Chacun choisit celui qui convient le plus à ses activités et à ses intérêts. Dès lors, le problème d'incompatibilité des données issues des différents systèmes est posé. Ce problème comporte deux dimensions : celle de l'échange et celle du partage des données techniques.

Echange de données: l'échange de données consiste à échanger des fichiers contenant des données codées selon différents formats. Souvent, les sous-traitants sont obligés d'acquérir le même système que leurs donneurs d'ordre uniquement pour l'utiliser comme un système intermédiaire pour pouvoir passer ensuite à leurs propres systèmes. L'utilisation des données d'un système dans un autre nécessite un passage par un pré-processeur qui joue le rôle du traducteur. C'est la raison pour laquelle des normes d'échange ont été développées.

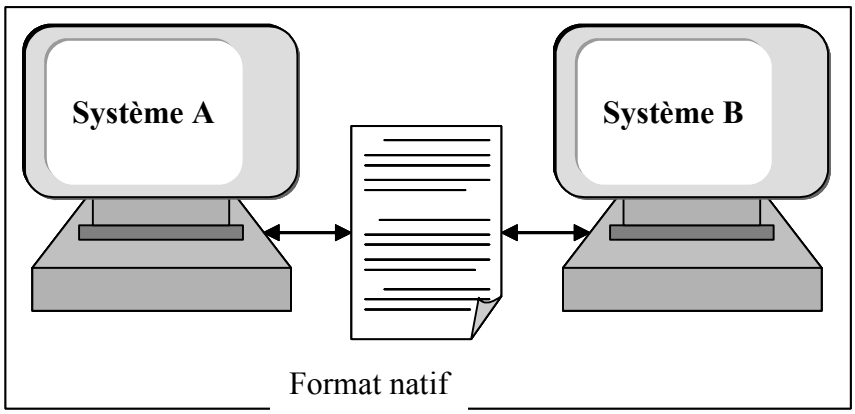


Figure 1: échange de données entre deux systèmes

Partage de données: Le développement de l'ingénierie concurrente a poussé à trouver une solution d'intégration des applications. Cette solution propose de stocker les données dans une base commune sous un format permettant de pouvoir les utiliser sur différents systèmes. Cela permet de faciliter l'implémentation des systèmes qui gèrent les accès aux données dans un contexte d'intervenants multiples. De plus, il conduit à une fédération des données afin qu'elles soient disponibles pour chacun de ces intervenants et sans se soucier des applications qu'ils utilisent.

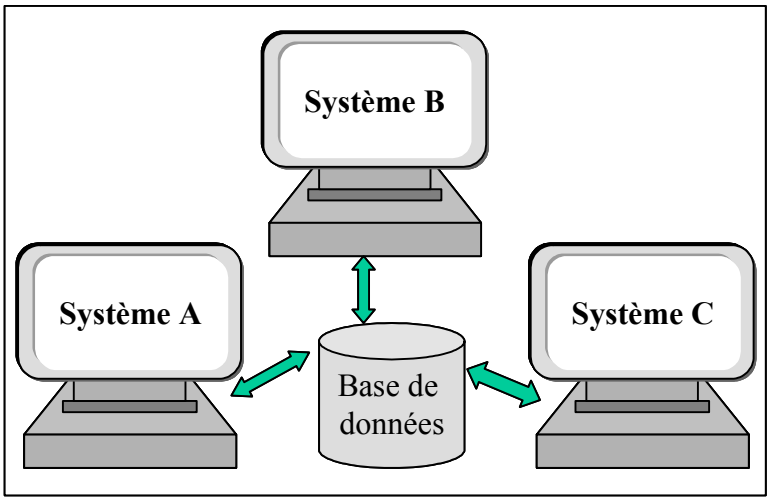


Figure 2 : Le partage de données

2.2.5 Développement de standards d'échange

La recherche d'un format commun pour les données issues de la XAO est un souci récurrent des industriels dans le monde entier depuis les années 70. La mise en œuvre d'échange et de partage des données a amené les industriels, surtout ceux qui utilisent beaucoup la sous-traitance (l'industrie automobile, aérospatiale), à créer des standards. Ces standards ont beaucoup servi aux différents partenaires pour mener à bien leurs projets. La plupart d'entre eux sont des standards nationaux; on peut citer la norme SET (Standard d'Echange et de Transfert), IGES (Initial graphical Exchange System), la norme allemande VDA-FS, …

2.2.5.1 Norme IGES

En 1979, aux Etats Unis, les utilisateurs et les éditeurs de systèmes CAO se sont mobilisés pour créer le premier standard d'échange de données CAO. Au printemps 1980 le National Bureau of Standard (NBS) avait formé l'organisation IGES (Initial Graphics Exchange Specification) [IGES 80], [Kemmerer 99]. IGES est un standard d'échange, dont le principal objectif est l'échange des dessins techniques. Il est utilisé principalement dans le domaine de mécanique et d'aéronautique. Au début, il contenait des entités de base telle que la géométrie, l'annotation et la structure. Les nouvelles versions ont intégré les méthodes éléments finis et les systèmes électriques. Le transfert des modèles solides a été ajouté dans la version 4.0 [Foulard 98].

2.2.5.2 Norme SET

SET (Standard d'Echange et de transfert) [SET 89] est un standard français lancé en 1983 par Aérospatiale. Cette dernière avait besoin d'une base de données commune pour des données issues de différents systèmes de CAO. Le standard représente une solution aux exigences relatives à l'échange de données entre différents systèmes de CFAO, et au besoin d'archiver ces données. La première version normalisée est apparue en 1985 (norme AFNOR expérimentale Z68.300.85.08). Elle conteint:

- des spécifications détaillées du domaine de la mécanique,

- des informations supplémentaires concernant la structure des données et les concepts utilisés,

- des règles et des recommandations concernant les spécifications qui assurent la cohérence dans le futur développement.

En 1987 est apparu le groupe opérationnel GOSET dont les principaux objectifs étaientt de développer SET, d'apporter une aide technique aux utilisateurs de la norme, de valider les interfaces entre les logiciels de CFAO et la norme et de participer aux travaux de normalisation nationaux et internationaux dans le domaine des échanges de données CFAO.

2.2.5.3 Norme VDA-FS

Le standard allemand, "Verband der Automobilindustrie Flaechen-Schnittstelle - Grafikstandard", (VDA-FS) [VDA 86] a été développé dans l'industrie automobile pour l'échange des modèles surfaciques. Il a été normalisé en 1985 par DIN (l'organisme de normalisation allemande). Cette norme était basée sur IGES mais a ajouté en particulier un format d'échange de fichiers de calcul à celle-ci [Kemmerer 99].

2.2.5.4 Standard DXF

DXF est un standard dit de fait. Il est développé par la société AUTODESK qui édite le logiciel AUTOCAD. Ce standard a pour objectif de sauvegarder les modélisations de DAO (Dessin Assisté par Ordinateur) d'AUTOCAD dans des fichiers ASCII. Le succès qu'a eu le logiciel AUTOCAD a rendu ce standard largement utilisable pour échanger des modélisations de DAO principalement dans le monde de la micro-informatique. Ce standard permet d'échanger sous forme numérique des plans de dessin industriel c'est-à-dire des entités géométriques bi-dimensionnelles, des annotations de dessin et des hachures.

2.2.6 La norme internationale STEP

Les normes nationales et standard industriel ont permis la résolution de certains problèmes d'échange et de partage de données. Toutefois, la création d'une norme internationale plus élaborée s'avérait intéressante. La communauté internationale voulait regrouper dans cette norme tous les avantages des standards nationaux (IGES, PDDI, SET et VDA-FS) tout en palliant les inconvénients et limitations de ces derniers. Elle a été conçue de telle manière à être facilement extensible et réutilisable. Après un certain nombre de rencontres des acteurs internationaux, un consensus a été établi sur les grandes lignes de ce standard. L'objectif principal de ce standard est de représenter, sans aucune ambiguïté les données liées à l'ingénierie et aux produits, dans un format neutre, interprétable par tout système informatique, sans perdre cependant l'intégrité des données durant tout le cycle de vie d'un produit [Kemmerer 99] [Pierra 00]. Ce standard international est connu sous l'acronyme STEP (STandard for Exchange of Product model data) et est enregistré au sein l'Organisation Internationale de Normalisation ISO sous la référence ISO 10303 [ISO 10303-1:1994].

Pour atteindre son objectif, STEP a :

- développé une nouvelle méthodologie de modélisation de données, pour assurer leur indépendance de tout système informatique. Cette approche comporte [Pierra 00]:
  - la définition d'un langage de spécification de données, le langage EXPRESS [ISO 10303-11:1994],
  - la définition d'un format neutre d'échange et de stockage des données décrites dans ce langage [ISO 10303-21:1994].
- établi des procédures permettant aux experts de chaque domaine technique de définir des modèles propres à leur domaine de spécialisation. Un certain nombre de modèles relevant d'aspects communs ont été également définis.

Ainsi, plusieurs protocoles d'application, basés sur le langage EXPRESS, ont été définis au sein de STEP. Un protocole d'application définit toutes les données requises dans un domaine d'application particulier. L'approche suivie pour l'élaboration d'un protocole d'application permet ainsi que celui-ci soit une véritable capitalisation du savoir-faire des experts du domaine d'application visé. Il décrit les informations pertinentes dans un domaine technique donné ainsi que la structuration de ces données. Il spécifie également les différents sous-ensembles implantables (« classe de conformité ») ainsi que la manière de les mettre en œuvre. Pour être conformes à STEP, les systèmes logiciels doivent être en mesure d'interpréter toutes les données définies dans une classe de conformité d'un protocole d'application. Cela assure que toutes les données peuvent être traduites à l'entrée ou à la sortie de cette structure sans aucune perte.

2.2.7 Structure de STEP

L'architecture de STEP est basée sur les deux niveaux de modélisation permettant de pratiquer, entre tous les protocoles d'application, des structures de modélisation et des modèles génériques.

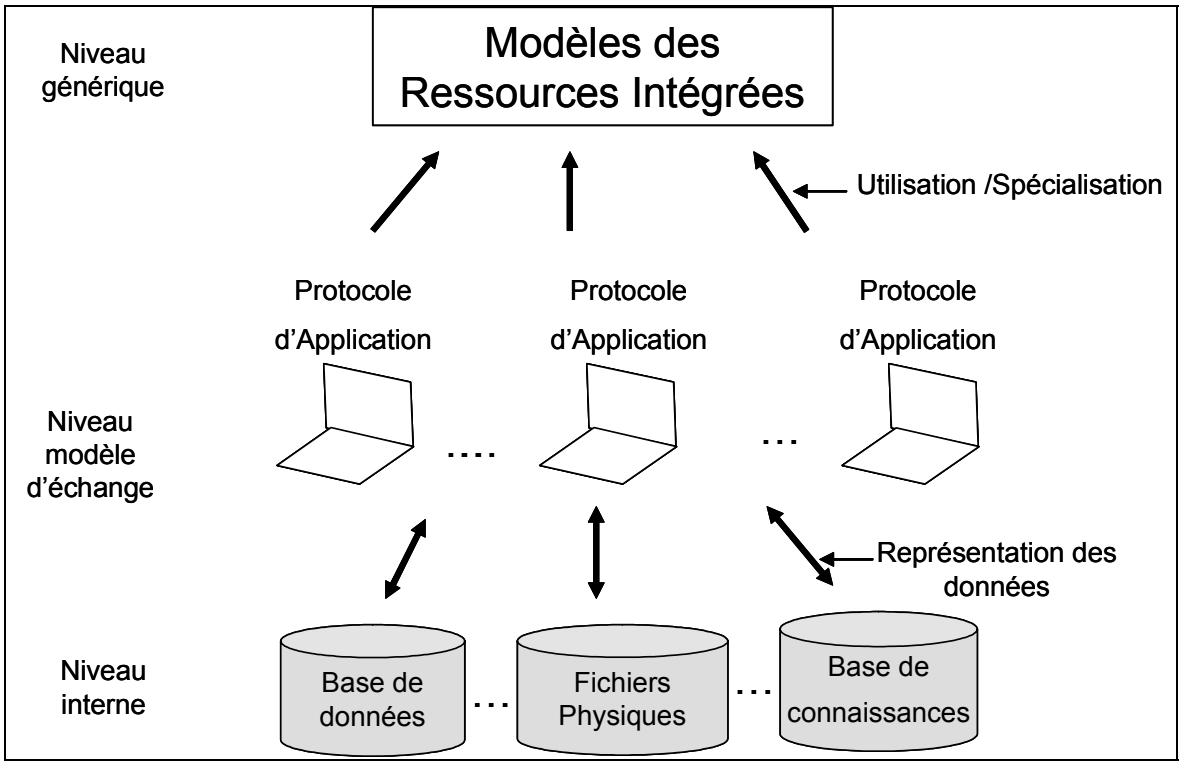


Figure 3 : L'architecture de STEP

Le niveau "échange de données effectif" du standard STEP est représenté par les Protocoles d'application. Chaque protocole d'application définit toutes les données requises dans un domaine d'application particulier. Pour être conformes à STEP, les systèmes logiciels doivent être en mesure d'interpréter les données définies par un protocole d'application. Cela assure que toutes les données peuvent être traduites à l'entrée ou à la sortie de cette structure. Chaque protocole d'application est un standard qui définit un modèle d'échange pour un domaine d'activité donné (exemple : [ISO 10303-214:2003] pour le domaine de l'industrie automobile).

Le niveau générique est représenté par les modèles des Ressources Intégrées. Ils comprennent les modèles génériques indépendants de tout contexte d'application. Ils sont destinés à être utilisés et spécialisés par tous les protocoles d'application. Ils constituent des « briques » élémentaires de données de STEP destinées à être utilisées par tous les protocoles d'application. Ils visent ainsi à faciliter l'interopérabilité entre les différentes applications implémentant ces protocoles d'application.

Le niveau physique (représentation effective de données) est défini par les méthodes d'implémentation. La norme STEP définit automatiquement pour tout modèle d'échange :

- une structure de fichier ASCII permettant l'échange des données [ISO 10303-21:1994], et
- une interface normalisée d'accès aux données conforme à ce modèle (SDAI, …) [ISO 10303-22:1997].

2.2.8 Description des données : Le langage EXPRESS

La description de données est la représentation d'un "univers du discours" par un modèle de données graphique ou textuel. Au début du développement de STEP, en 1986, aucune notation de modélisation de données ne s'avérait à la fois suffisamment précise et suffisamment expressive pour l'objectif visé. Un nouveau langage a donc dû être développé [Pierra 00]. Il s'agit du langage de modélisation de données EXPRESS. Son objectif principal est la description de modèles d'informations dans le domaine technique, en vue de l'échange de données représentant de façon fiable et non ambiguë ces informations [ISO 10303-11:1994] [ISO 10303-11:2000] [Bouazza 95a] [Pierra 00] (nous avons utilisé la version EXPRESS ISO 10303-11:1994 dans nos travaux). A la différence des langages de modélisation de données jusque là développés (Entité/Association [Chen 76], OMT [Rumbaugh et al 91], NIAM [Nijssen 81], …), EXPRESS permet, en plus de la représentation simplifiée (graphique) des données destinées à l'échange entre concepteurs humains, une représentation textuelle non ambiguë qui est traitable par machine.

EXPRESS est un langage de modélisation et de spécification des données normalisé (ISO10303). Il a été défini principalement pour représenter les modèles de données dans les domaines techniques et est à présent largement utilisé pour la modélisation des données dans différents domaines. De plus, il peut être utilisé pour la spécification de plusieurs applications dans le domaine de développements informatiques [Ait-Ameur et al 00]. Il a été définit dans le contexte du standard STEP pour assurer à la fois, l'indépendance des données de tout système et leur complétude, et pour augmenter leur richesse. La définition du langage EXPRESS [ISO 10303-11:1994] et du format d'échange et de stockage de données [ISO 10303-21:1994] qui accompagne ce langage a augmenté l'efficacité de l'échange.

Un format d'échange d'instances des modèles de données EXPRESS a été défini dans la partie 21 du standard STEP [ISO 10303-21:1994]. Cette spécification définit la manière de coder, sous forme d'un fichier de caractères, une population d'instances d'entités conforme à un modèle défini en EXPRESS. EXPRESS contient également une représentation graphique en parallèle de la représentation textuelle : il s'agit du formalisme EXPRESS-G. Ce dernier permet une représentation synthétique d'un modèle de données EXPRESS. De plus, il peut être utilisé dans les phases préliminaires de conceptions de modèles de données. Des détails complémentaires sur le langage EXPRESS sont donnés dans l'annexe.

2.3 Le schéma GDT de STEP

Nous avons parlé dans les paragraphes précédents des protocoles d'application et de leur intérêt dans la norme STEP. Parmi les protocoles d'applications de STEP, il y en a quatre qui s'intéressent particulièrement à la gestion des nomenclatures de produit (structure de composition) et/ou des documents associés ; C'est-à-dire à la structure des données dans un SGDT. Il s'agit en fait des protocoles d'application 203 (conception tri-dimensionnelle de pièces et composants mécaniques avec contrôle de configuration) [ISO 10303-203:1994], 212 (Conception et installation électrotechnique) [ISO 10303-212:2001], 214 (données de base pour les processus d'étude du matériel automobile) [ISO 10303-214:2003] et 232 (présentation des données techniques : informations de base et échange de données) [ISO 10303-232:2001].

Afin de faciliter l'interopérabilité entre ces protocoles d'application, le schéma GDT de STEP (STEP PDM Schema) [Kindrick et al 00] a été développé par ProSTEP[1] en collaboration avec PDES, Inc.[2]. Ce schéma définit l'ensemble des entités STEP qui interviennent dans la structuration d'un SGDT :

- structure de composition du produit (nomenclature),
- gestion des représentations et documents associés aux différents niveaux de cette structure.

Ce schéma permet de faciliter l'échange de l'ensemble des informations stockées au sein d'un SGDT. Il décrit entre autres [Kindrick et al 00]:

- les propriétés des articles (la masse, le prix),
- la gestion des modifications et,
- des flux de travail (effectivity and workflows management).

La figure suivante montre la place du schéma GDT dans la norme STEP.

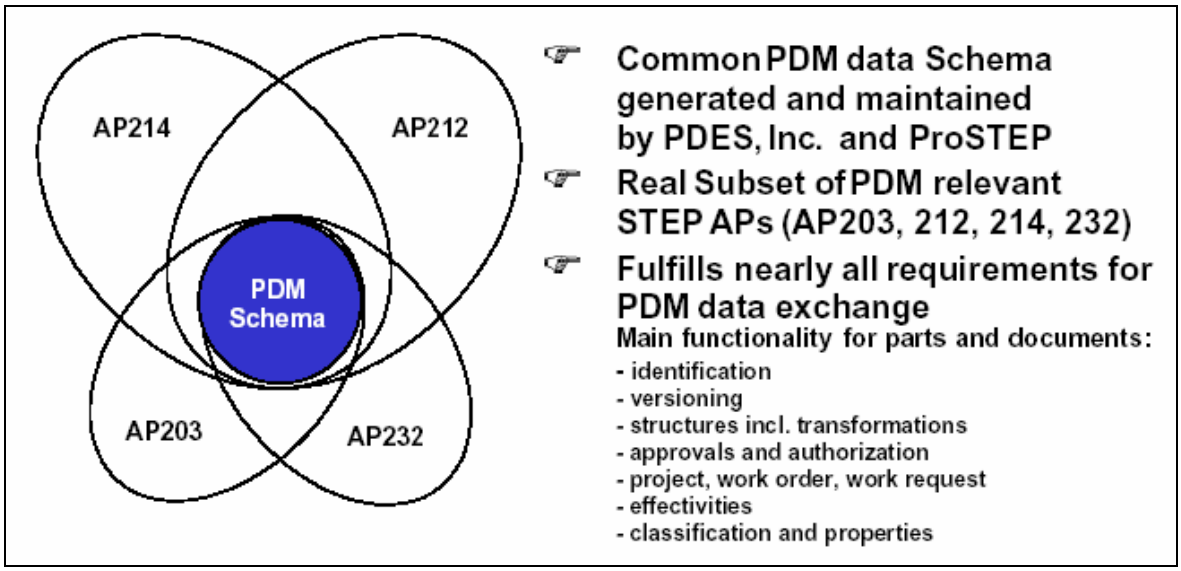


Figure 4 : la place de du schéma GDT dans la norme STEP [Kindrick et al 00]

[1] ProSTEP est une association pour la promotion des normes internationales de données de produit (STEP), Allemagne.

[2] PDES, Inc. est un consortium global d'industrie, de gouvernement, et de membres académiques travaillant ensemble pour avancer le développement et la mise en œuvre de la technologie STEP, USA.

2.4 Structuration des données techniques dans le schéma GDT de STEP

2.4.1 Le produit

L'entité centrale qui constitue la base d'information dans le schéma GDT de la norme STEP est le produit (*Product).* Dans ce schéma, tous les objets techniques (y compris les documents) sont des produits (*part as product).* Un objet peut être un composant simple ou bien un assemblage (d'une complexité variable). La caractérisation d'un "produit" dans le schéma GDT de STEP est basée sur les trois concepts suivants :

- L'identification : elle comporte l'identification principale du produit, permettant une identification absolue du produit commune à toutes les versions et les étapes du cycle de vie (nom, num_id), une identification des versions (identifier les modifications apportées à un produit) et finalement une définition des vues (pour un domaine particulier ou une phase particulière du cycle de vie),

- le contexte d'information : celui-ci permet de définir le contexte d'application de l'information représentant le produit (domaine d'application, contexte de définition du produit, étapes du cycle de vie, etc.),

- la classification : elle permet la distinction des instances de l'entité "*Product*" considérées comme objets et celles considérées comme documents. Cette classification permet également de différencier les différents types d'objets (composants de base, assemblages ou bien composants standards).

2.4.2 Représentations du produit

Dans le schéma GDT de STEP, une représentation d'un produit modélise un point de vue particulier sur ce produit. Ces représentations sont appelées *Part Properties*. Il s'agit d'une définition d'une propriété spécifique du produit qui reflète une caractéristique physique (masse, matière, …) ou tout autre représentation de celui-ci (coût, qualité, représentation 2D, …). Chaque propriété du produit est associée à une représentation qui peut être une simple valeur (valeur de la masse ou du coût) ou quelques chose de plus complexe comme que la représentation 2D ou un modèle éléments finis.

2.4.3 Document

Tout au long du cycle de vie d'un produit, des document sont créés pour le décrire. Un document peut être défini comme un ensemble cohérent d'informations visant à décrire un article ou un ensemble d'articles (assemblage) [Maurino 93]. Ces informations sont structurées soit en texte ou sous un autre format compréhensible par un homme ou une machine (dessin, programme d'une machine-outil à commande numérique, …). Cela comprend donc les documents simples (information structurée en texte ou en représentation graphique compréhensible par un être humain) et les documents techniques telles qu'elles sont définies dans la section 2.2.1. Comme nous

l'avons précisé dans la section 2.4.1 dans le schéma GDT de STEP, l'entité centrale du modèle est le "*Product*". Les documents sont donc représentés par cette même entité en utilisant le concept *"document as product"*. Un document possède une identification principale (ID, fichier d'attachement, version, …), un contexte de définition (usage du document au sein du produit, le type de document (document électronique ou papier), …) et finalement la classification du produit [Kindrick et al 00]. En effet, c'est cette dernière qui permet de distinguer entre une entité "*Product*" représentant un article de celle représentant un document.

2.4.4 Liens entre les objets techniques (articles et documents)

2.4.4.1 Liens de composition

Le lien de composition sert à lier un objet à ses constituants. Ce lien permet d'établir et de gérer la nomenclature des produits qui est une fonctionnalité très importante d'un SGDT. Ces liens de composition sont porteurs d'informations telles que :

- l'identification du lien : type de lien, type de structure (fonctionnelle, technique, industrielle, …), type de configuration décrite (de référence, produite, en service, …),
- la qualité du composant entrant dans le composé (lien entre articles ou exemplaire) (les articles sont interchangeables les exemplaires ne le sont pas),
- la validité du lien (date de création, date de péremption, …).

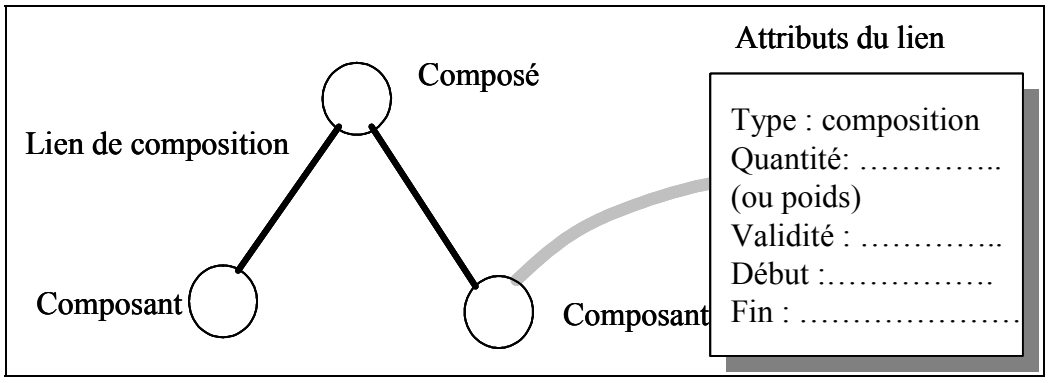


Figure 5 : lien de composition [Maurino 93]

Les attributs de lien sont représentés par des classes de lien. Un lien de composition est une instance de cette classe.

2.4.4.2 Lien d'association

Un document est associé à l'objet qu'il décrit par un lien d'association appelé également lien de représentation. Ce lien porte en particulier des informations sur la date de sa création, sa date de péremption, … Le lien d'association permet de trouver

tous les documents qui servent à décrire un article. L'association des documents à des articles permet de donner une représentation multi-vue d'un produit et ainsi de donner à chaque intervenant sa vision du produit.

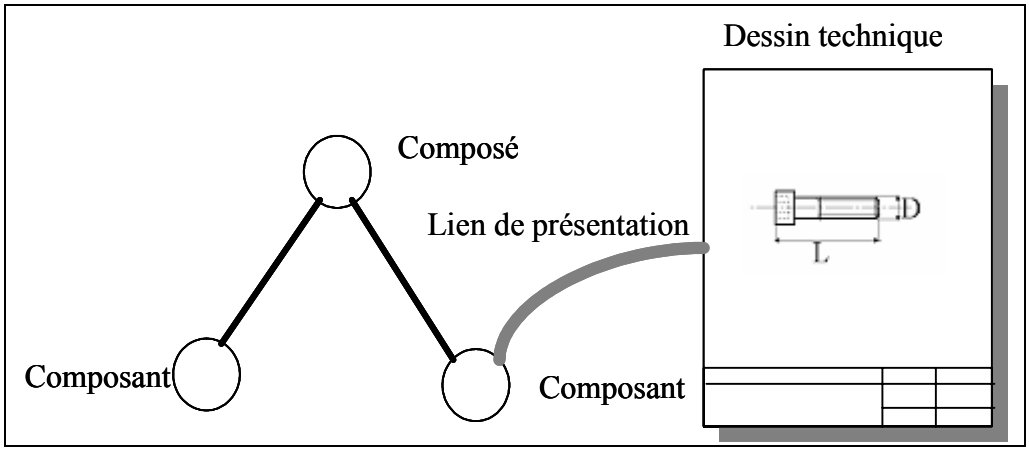


Figure 6 : lien d'association ou de représentation [Maurino 93]

2.4.5 Différentes structures des produits

Comme nous l'avons signalé ci-dessus, l'élaboration d'un produit nécessite l'implication de plusieurs métiers et l'intervention de plusieurs acteurs. Chaque acteur manipule une vue particulière du produit qui correspond à ses besoins spécifiques.

L'objet technique et le plus grand diviseur commun que connaissent les différents acteurs sur le produit. Une structure du produit le décrit par des niveaux successifs de décompositions en objets techniques d'un point de vue particulier [Maurino 93]. On peut considérer le produit du point de vue de sa fonction, de sa définition (dessin technique), de son processus de fabrication (nomenclature et gammes de production et d'assemblage), ou de son processus de mise en œuvre et de maintenance. La structure principale qui est formellement gérée par toutes les entreprises est la nomenclature de production, appelée également structure industrielle. Ces structures de produit définissent des modèles de produits par des représentations.

2.4.5.1 Structure fonctionnelle

C'est la description des différents niveaux du produit en fonctions (allumage (bougies, batterie, …), transmission (boite de vitesses, …)). Cette structure permet d'ordonner et de hiérarchiser les fonctions du produit. La hiérarchisation a pour but le classement des fonctions du produit d'une manière logique en établissant des relations de dépendance entre elles. Cette structure permet l'expression de besoins en termes de fonctions attendues, sans évoquer les solutions techniques susceptibles de les résoudre.

Les objectifs de l'élaboration d'une structure fonctionnelle sont énoncés par la norme NF X 50-151 (expression fonctionnelle de besoins et cahier des charges fonctionnel) :

- l'offre de produits compétitifs et innovants au moyen d'une expression de besoins formulée en termes d'obligation de résultats, plutôt qu'en termes de moyens,

- utilisation en interne de toutes les compétences de l'entreprise nécessaires pour évaluer ce besoin,

- l'expression d'une manière unifiée du besoin par des acteurs ayant des attributions et des visions différentes du produit,

- l'incitation au dialogue entre les différents partenaires et l'élaboration d'une relation plus claire entre le client et le fournisseur,

- la progression technique vers la solution la mieux adaptée au besoin,

- la démonstration par le fournisseur, à chaque étape de l'élaboration du produit, que la solution proposée y répond effectivement,

- la possibilité de comparer des offres ou des produits répondant au même besoin.

2.4.5.2 Structure Technique

La structure technique, appelée également nomenclature d'étude, permet une vision de produit qui facilite l'organisation des travaux de conception du produit et offre des réponses et des solutions techniques à l'expression fonctionnelle des besoins et aux exigences techniques du produit. Elle définit l'architecture technique du produit telle quelle est définie dans les phases préliminaires de conception [Maurino 93].

2.4.5.3 Structure Industrielle

La structure industrielle, appelée également nomenclature de production, est la description des étapes successives d'élaboration du produit à partir de matières, composants et sous-ensembles [Maurino 93]. Elle est utile pour identifier les étapes de production et les gammes associées, pour élaborer le plan directeur de production, pour lancer et suivre les ordres de fabrication et pour planifier les approvisionnements et suivre les livraisons (ces éléments font partie de l'approche "production" non traitée dans ce document). Elle permet d'obtenir une vision hiérarchique du produit qui s'appuie sur les articles (composés et composants) et qui prend en compte l'organisation de la production.

2.4.5.4 Structure Logistique

La logistique est l'ensemble des opérations et des procédures permettant d'assurer la disponibilité du produit dans de bonnes conditions, au bon moment, au bon endroit, au bon client et au coût le plus juste. L'analyse du processus logistique permet d'éliminer la duplication des efforts, d'éliminer les freins, de réduire les délais et

l'accumulation de stocks. L'ASLOG (Association française pour la logistique) définit la logistique comme étant "*l'art et la manière de mettre à disposition un produit donné au bon moment, au bon endroit, au moindre coût et avec la meilleure qualité*" [http://www.aslog.org/].

La structure logistique du produit est une description arborescente des éléments de soutien logistique, hiérarchisés en fonction d'un échelon de maintenance du produit [Maurino 93]. Elle décrit les opérations de remplacement des composants échangeables d'un produit (ou d'un assemblage). Par exemple, l'assemblage roues et freins d'une voiture peut être décomposé en pièces démontables et non démontables. A chaque niveau de décomposition, on spécifie pour un assemblage démontable les opérations de démontage. Ceci aboutit naturellement à une description de l'ordre de démontage et de montage permettant l'échange de plaquettes de freins par exemple.

2.5 Systèmes de Gestion des Données Techniques : SGDT

Durant cette dernière décennie, une nouvelle organisation industrielle a été mise sur pied afin de réduire les délais de mise sur le marché des produits manufacturés. Il s'agit de l'ingénierie simultanée (ou Concurrent Engineering). Cette organisation ne peut être déployée qu'avec une maîtrise totale de la documentation technique (données techniques) et de la base documentaire (documents simples) qui accompagnent le développement du produit (voir section 2.4.3). Or, cette maîtrise de l'information technique nécessite des outils adaptés et efficaces. C'est pour gérer les informations inhérentes au processus de développement, qu'un outil connu sous le nom de SGDT a été développé. Ce système est un progiciel, une "grosse boîte à outils" basée sur un Système de Gestion de Base de Données (SGBD) [Foulard 98]. Les SGDTs permettent la structuration de l'information technique et, en outre, ils en facilitent l'accès, en assurent la sécurité et gèrent ses évolutions.

Un SGDT comporte un modèle de produits et les documents techniques qui y sont associés. Un modèle de produit est décrit par des nomenclatures d'articles auxquels sont liés les documents qui les décrivent (fichiers CAO, image, modèle de calcul éléments finis, …). Il comporte également un schéma de fonctionnement informationnel de l'entreprise. Ce schéma comprend les acteurs industriels, leurs droits et devoirs, ainsi que les procédures de validation et de modification des documents (exemple : un dessin technique ne doit pas être transmis à la fabrication avant la validation du responsable des études). Ce dernier point constitue l'aspect dynamique de la gestion des données techniques de produit. Il est assuré par le module de gestion des flux de travail (*workflow management*).

2.5.1 Fonctionnalités d'un SGDT

Les principales fonctionnalités d'un SGDT sont :

- le stockage des données élémentaires dans un coffre fort "Data vault". L'objectif de ce "*vault*" est d'assurer que les données sont à jour, correctes et protégées contre les accidents ou les malveillances, de garantir que les données contrôlées par le SGDT ne sont accessibles qu'à travers ce logiciel (fonctions "Check-in" "Check-out"), de gérer les versions (les données ne

sont validées "prêt à l'emploi" qu'à travers une procédure bien définie), de garantir la traçabilité des données et finalement offrir une transparence aux utilisateurs (ne pas se préoccuper de la localisation des données). Par ailleurs, les documents non électroniques (plans, schémas, …) peuvent être gérés par des références externes,

- la gestion de la structure produit. Elle est basée sur la nomenclature produit. Elle permet à chaque utilisateur d'avoir une vue "métier" du produit (conception, ordonnancement/fabrication, maintenance, vente) (voir section 2.4.5),

- la structuration de l'instruction d'un dossier et de sa chronologie (workflow). Ceci comprend la définition des processus et flux utilisés pour modifier les données, la gestion des approbations et autorisations selon des règles et procédures prédéfinies (les acteurs sont clairement identifiés par le système qui garantit la signature électronique),

- la visualisation des données par activation des programmes spécifiques permettant de les interpréter (exemple : activer un outil CAO pour visualiser un plan 2D).

Nous avons présenté dans les sections précédentes l'approche de modélisation des données techniques de produit par des représentations. Celles-ci sont adaptées à la gestion de ces données pendant les phases de son cycle de vie (de son développement jusqu'à sa mise sur le marché). Beaucoup de produits contiennent des assemblages de produits préexistants Pour l'entreprise utilisatrice, la description de ces produits préexistants est différente de celle utilisée dans l'entreprise qui les produit. Il ne s'agit pas de fabriquer ces produits mais de les retrouver, de les comparer et de les choisir. Pour l'entreprise utilisatrice, ces produits réutilisables sont appelés *composants*. L'utilisation d'un composant particulier dans la conception d'un nouveau produit est basée sur un choix. Ce choix, effectué sur un ensemble de composants similaires, est réalisé selon des critères de sélection définis par le fournisseur lui-même en tenant compte du contexte dans lequel ce composant sera inséré. Ceci nécessite donc la modélisation de ces produits (i.e. composants) d'une manière plus adaptée à ces nouvelles exigences afin de faciliter leur comparaison. C'est l'approche de modélisation par propriétés qui est la plus adaptée pour répondre à l'objectif visé. Nous présentons dans les paragraphes suivants les aspects de cette approche.

3 Modélisation des données de produit basée sur les propriétés

3.1 Introduction

Nous avons donné dans les paragraphes précédents la définition de modèle par [Minsky 86]. Celui-ci le définit comme un moyen de répondre à des questions précises sur l'univers de discours modélisé. Il est clair que cette deuxième manière de modéliser les données (par propriété) vise à donner une autre vision du produit. Il s'agit de la vision synthétique utile pour comparer des produits similaires. En effet, la conception

d'un nouveau produit se base souvent sur la sélection et l'utilisation de produits pré-existants : les composants. Dans cette phase du cycle de vie d'un produit (conception) le concepteur a donc besoin de comparer un ensemble de composants voisins afin de choisir celui qui répond le mieux à ses attentes. Le choix du concepteur peut aussi bien se baser sur des propriétés intrinsèques des composants (diamètre d'un roulement) que sur des propriétés dépendant de son contexte d'insertion (durée de vie d'un roulement dans des conditions particulières d'utilisation). Dans ce qui suit, nous parlerons du modèle de données des composants.

Nous commençons dans cette partie par présenter les principes de la modélisation par propriétés d'une manière générale. Celle-ci est structurée en quatre niveaux : les instances, les modèles, les méta-modèles, les méta-méta-modèles. Puis nous discutons de l'importance des données de composants dans les SI des entreprises ainsi que de la nécessité de leur échange entre les différents acteurs impliqués dans ces entreprises. Nous discutons également de la particularité de ces données et des problèmes freinant leur échange. Nous cherchons ainsi à présenter l'intérêt à utiliser l'approche de modélisation par propriétés pour représenter les données de composants et donner les grande ligne de cette approche pour faciliter la compréhension du modèle PLIB. Celui-ci sera détaillé dans la quatrième partie.

3.2 Modélisation par propriétés : principe et définition

La modélisation par propriétés est une des caractéristiques de l'approche objet dont l'origine remonte aux années 60 avec le langage SIMULA [Birtwistle et al 73], [Dahl et al 66]. Cette approche a été utilisée sous le nom de Property Driven Model dans la fin des années 70 à l'Université de Technologie de Compiègne [Barthès et al 79]. Elle a été également influencée par le modèle de "frames" de M. Minsky [Minsky 75] (ce sont des granules de connaissances plus importantes que les nœuds d'un réseau sémantique), et le modèle sémantique d'Abrial [Abrial 74]. Cette approche considère que le monde est constitué d'entités (les objets) ayant des propriétés qui sont également des objets [Shen 95a].

L'objectif initial de la modélisation par propriétés était de pouvoir accommoder un grand nombre d'objets changeant dynamiquement et de les stocker de façon permanente dans une base de données (appelée VORAS). Cette modélisation comporte une structure de schéma à deux niveaux (objet, attribut), dans lequel les attributs eux-mêmes sont des objets. Les autres facettes possibles de l'attribut deviennent alors des attributs de l'objet attribut.

Ainsi le monde réel est considéré comme étant constitué d'objets ou d'entités, ces entités elles-mêmes possédant un certain nombre d'attributs ou propriétés. On distingue deux types d'attributs objets :

- propriétés de structure : elles pointent sur d'autres objets
- propriétés terminales : elles pointent directement sur une valeur

Les propriétés et les modèles (les classes) sont à leur tour représentés comme des objets. De plus, certaines propriétés peuvent être partagées entre les objets.

La modélisation par propriétés fait apparaître quatre niveaux :

- les instances,
- les modèles (les classes), formant une hiérarchie IS-A,
- le méta-modèle,
- le méta-méta-modèle.

Ces mêmes niveaux ont été utilisés par l'OMG (*Object Management Group)* pour définir le MOF *(Meta Object Facility)* [OMG 02]. La figure suivante montre ces différents niveaux de modélisation des données.

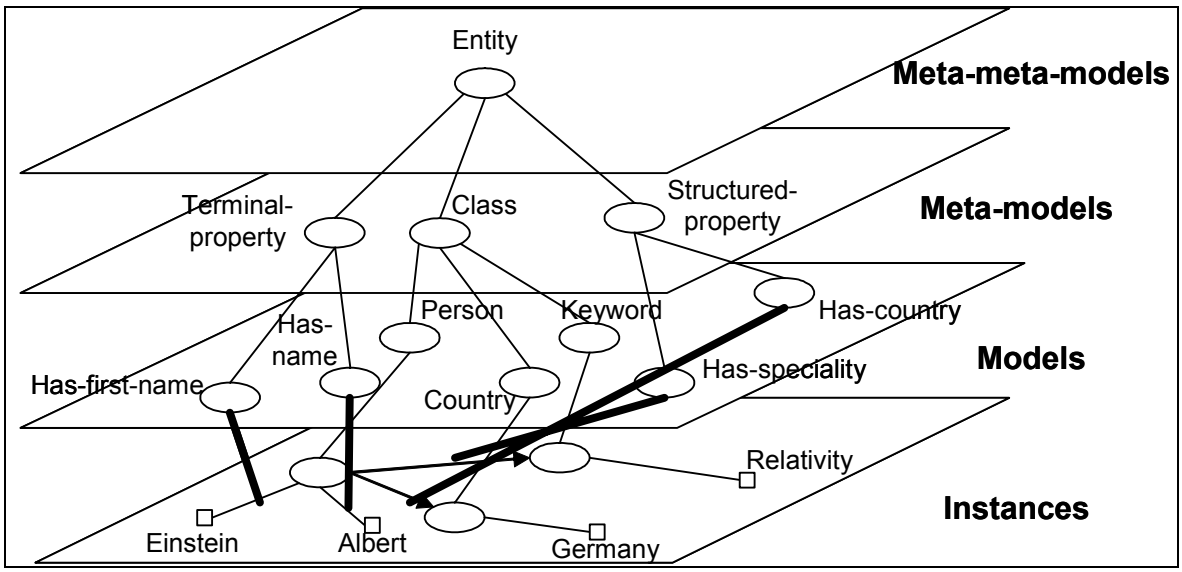


Figure 7 : les quatre niveaux de modélisation par propriétés [Shen 95a]

Il est clair que la modélisation par propriétés est une approche beaucoup plus adaptée à la recherche d'un composant ayant des caractéristiques données que la modélisation basée sur des représentations. Nous discutons dans les paragraphes suivants les besoins correspondant à la représentation des composants.

3.3 Importance des données de composants dans le SI de l'entreprise

Dans la plupart des domaines de l'ingénierie (mécanique et éléctronique en particulier), les produits à concevoir sont essentiellement des assemblages de composants techniques préexistants [Moranne 86]. Dans ces domaines, une grande partie de la connaissance concerne la connaissance sur les composants [Paasilia et al. 93]. Cette connaissance est détenue essentiellement par les fournisseurs des composants. Elle correspond au savoir-faire du fournisseur sur ses composants :

critères de sélection d'un composant, cas d'utilisation, comportement dans un contexte d'utilisation particulier, etc.

Les outils informatiques d'ingénierie (IAO) sont de plus en plus utilisés dans les processus d'ingénierie au sein des entreprises et permettent une augmentation nette de la productivité. Actuellement, dans une majorité des cas, les informations descriptives des composants sont disponibles sur des documents papier (normes, catalogues fournisseurs) [Maurino 93]. Dans le cas où ces informations sont représentées d'une manière numérique, cette représentation se limite généralement à des bases d'articles qui ne contient qu'une partie infime de ces informations (une désignation, un prix, un fournisseur). L'apparition de l'ingénierie concourrante (ou simultanée) et l'utilisation des outils informatiques pendant les différentes phases du cycle de vie des produits ont nécessité le développement d'une représentation informatisée des catalogues et des normes. Cette représentation informatisée doit être basée naturellement sur des modèles de données bien adaptés. Ces modèles de données doivent répondre aux besoins liés à l'utilisation des données de composants : à savoir leur sélection et leur intégration au niveau des modèles des produits. Ils doivent permettre la représentation du savoir faire des fournisseurs sur les composants pour le mettre à disposition des concepteurs. De plus, les modèles de composants doivent favoriser leur sélection.

3.4 Echange et partage de description de composants

Comme nous l'avons précisé précédemment, la conception de nouveaux produits se base principalement sur la sélection et l'assemblage de composants préexistants. Lors de cette étape les concepteurs se basent sur le savoir-faire des fournisseurs pour réaliser leurs sélections. Les données concernant les composants doivent donc être échangées dans cette optique. Avec l'utilisation des outils de XAO, les concepteurs souhaitent pouvoir récupérer sous forme informatique toutes les informations sur les composants (avec leurs représentations selon différents points de vue, dessin 2D, par exemple) afin de les insérer directement dans leurs modèles de produit. Ainsi, le fournisseur ne doit pas seulement livrer les composants mais également toutes les connaissances qu'il possède à leur sujet.

L'approche traditionnelle d'échange de connaissance réunit la connaissance et son opérationnalisation. C'est l'approche utilisée, par exemple, dans les systèmes experts [Farreny 85] dans le milieu des années 70. La connaissance de l'expert est codée directement dans un langage d'exécution (par exemple, le langage PROLOG [Colmerauer et al. 83] utilisant un formalisme de représentation basé sur la logique du premier ordre) puis, cette connaissance est activée par l'utilisateur dans un contexte opératoire défini par le langage d'exécution [Sardet 99]. Avec l'hétérogénéité des systèmes de XAO dans lesquels on souhaite intégrer la connaissance du fournisseur, cette approche n'est pas possible. L'approche la plus récente, utilisée en particulier dans les travaux KADS (Knowledge Acquisition and Design Support) [Schreiber et al 92], sépare la connaissance et son opérationnalisation. La connaissance est modélisée de façon explicite à l'aide d'un ou de plusieurs modèles de données. Son opérationnalisation, c'est-à-dire son exploitation par un utilisateur dans un cadre spécifique, ajoute à cette connaissance un environnement informatique permettant sa mise en œuvre. C'est cette approche qui doit être suivie pour permettre l'échange et le partage des descriptions de composants dans un univers hétérogène. Pour mettre en œuvre cette approche, on a appliqué le principe de la modélisation par propriétés pour

définir les différents modèles permettant la représentation des catalogues de composants.

3.5 Particularité des données de composants

On s'intéresse particulièrement ici aux composants des fournisseurs, c'est-à-dire ceux qui sont acquis de l'extérieur de l'entreprise. Ces composants ont les particularités suivantes :

- Ils sont décrits dans des catalogues,
- un catalogue regroupe les composants par familles, les propriétés de description de l'ensemble des composants d'une famille sont spécifiques de cette famille,
- les différents composants de chaque famille sont alors décris par les valeurs de ses propriétés, ces valeurs étant définies soit explicitement (par une table) soit implicitement (par un ensemble de tables, des opérateurs de combinaison et/ou des formules de calcul),
- les différentes familles sont organisées selon une hiérarchie de classes. Les propriétés peuvent être factorisées à différents niveaux de cette hiérarchie.

De cette classification, en prenant une perspective base de données, il apparaît que :

1- la description d'une famille constitue le modèle des instances qui lui appartiennent,

2- lors de l'échange d'un catalogue, on échange à la fois les modèles et les instances,

3- les instances peuvent être décrites implicitement (par des formules de calcul, par exemple). Elles n'existent pas alors explicitement dans l'échange.

Ces particularités mettent en évidence les trois difficultés de représentation et de gestion des données de composants :

1- le système de gestion de composants doit permettre de gérer à la fois les modèles (de familles) et les instances (de composant),

2- si chaque instance de composant est représentée comme une instance d'une relation, le système de gestion doit être capable d'intégrer dynamiquement dans son schéma global les schémas relationnels des nouvelles familles devant être représentées,

3- les données décrites implicitement au niveau du format d'échange doivent pouvoir être représentées dans le système receveur.

Nous détaillons dans la section suivante les différents problèmes que pose l'intégration dynamique de données hétérogènes. Nous introduisons ensuite une vue synthétique du modèle PLIB qui permet de représenter et d'échanger simultanément modèles et instances et que nous utiliserons donc dans cette thèse pour résoudre et alimenter le système de gestion à développer.

3.6 Problèmes d'intégration de données hétérogènes

La nécessité d'échange de données de composants entre systèmes hétérogènes se heurte à différents problèmes liés à l'existence de plusieurs représentations déconnectées les unes des autres. Nous discutons dans ce qui suit les différentes facettes de ces problèmes.

3.6.1 Diversités de dénomination

Cette diversité vient du fait que les mêmes concepts peuvent avoir des noms différents dans des bases de données différentes. Ainsi, dans chaque base de données, entités, tables et attributs sont identifiées par des noms. Des noms différents peuvent être utilisés pour désigner le même concept, ou, inversement, des concepts différents peuvent être désignés par le même nom. Ceci constitue le premier frein à l'échange des données de composants entre systèmes hétérogènes.

3.6.2 Diversités de modélisation conceptuelle

Dans le même domaine, et en supposant que les noms sont identiques, la structure du modèle conceptuel peut changer. Un modèle peut ne pas présenter de spécialisation particulière par rapport aux concepts et aux noms existants dans l'autre système, il peut au contraire avoir donné lieu a spécialisation. De plus, s'il y a spécialisation, on peut s'être basé sur différents critères pour faire la spécialisation. Par exemple, pour une voiture, on peut spécialiser selon l'origine (française, étrangère), le type (voiture de course, tout terrain ou touristique), etc. Concernant le choix d'attributs, chaque catégorie d'utilisateur et donc chaque schéma s'intéresse à un sous-ensemble particulier des attributs possibles d'une entité.

3.6.3 Diversités d'implémentation et représentation des données

Une troisième difficulté pour l'échange de données de composants est celle de la représentation des données. En effet, plusieurs systèmes sont utilisés (notamment des SGBD) pour représenter les données d'un domaine particulier. Ainsi, dans le même domaine, et en supposant que les mêmes concepts (entités et attributs) sont utilisés, la diversité de représentation résulte de la multitude de choix permettant de définir le schéma final de la base de données. Elle peut avoir trois causes: le souci d'éviter la redondance (l'aspect normalisation), l'ordre dans lequel les attributs sont représentés dans les tables, et enfin le caractère spécifique de chaque SGBD qui entraîne une

diversité d'implémentation (représentation directe de l'héritage dans un SGBDRO, et la simulation de celui-ci dans un SGBDR). Nous discutons dans le paragraphe suivant la manière dont ces besoins et difficultés concernant la modélisation de composants sont traités dans le modèle PLIB.

4 Le modèle de données de composants PLIB

Nous avons posé dans le paragraphe précédent les problèmes de la modélisation des données de composants. Nous avons présenté également les différentes sources d'hétérogénéité qui peuvent exister entre les modèles de données et qui entravent ainsi l'échange et le partage de ces données entre systèmes. Afin de pallier ces problèmes et afin d'offrir une représentation qui réponde aux exigences de données de composants, le modèle PLIB a été développé. Nous présenterons dans les paragraphes suivants le modèle de données de composants défini par PLIB.

4.1 Elément de base du modèle PLIB pour représenter les modèles de composants

4.1.1 Définition d'une ontologie pour le domaine de composants : le dictionnaire sémantique

Nous avons vu que parmi les difficultés de l'échange de données de composants se trouve la diversité de dénomination. Les problèmes liés à ce type de diversité sont réglés par l'établissement d'une ontologie et d'un mécanisme de nomination. L'ontologie permet la définition des concepts utilisés dans le domaine de composants d'une manière formelle et universelle. Cette définition comprend d'une part la description de chaque concept utilisé (classes et propriétés) pour éviter toute ambiguïté, et, d'autre part, une organisation des ces concepts dans une structure bien définie afin d'en définir très précisément le sens. Dans cette structure, les classes sont organisées dans une hiérarchie de classes avec héritage simple et sont liées aux propriétés qui les définissent via l'identifiant de celles-ci. Les classes et leurs propriétés sont définies en parallèle. Les types de valeurs des propriétés sont également définis dans l'ontologie. Ainsi, chaque propriété est associée à un domaine (la classe dans laquelle elle s'applique) et un co-domaine constitué d'un type de données qui définit d'une manière précise le domaine de valeurs réel de la propriété.

4.1.2 Identification des concepts (le mécanisme du BSU)

Une fois les concepts définis, il faut les identifier afin d'éviter les problèmes liés aux différences de noms. Dans le modèle de données PLIB, les symboles qui sont utilisés pour représenter chaque concept, et qui permettent de le référencer, sont appelés les "unités sémantiques atomiques" (Basic Semantic Units ou BSU). Un schéma universel de nomination a donc été défini [ISO 13584-42:1998]. Ce schéma permet d'identifier trois catégories de concepts : les sources de l'information (*supplier_BSU*) (permettant ensuite l'utilisation conjointe de plusieurs ontologie), les entités (*class_BSU*) et les propriétés (*property_BSU*).

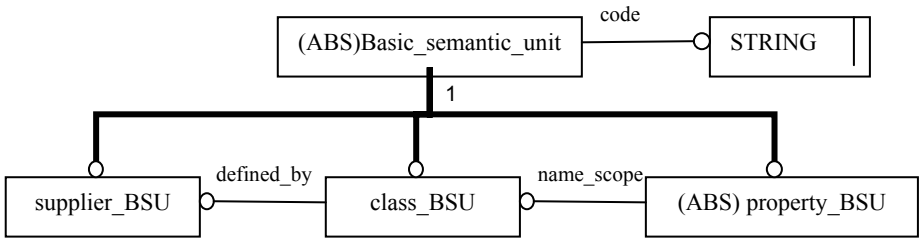


Figure 8 : identification universelle des concepts [ISO 13584-42:1998]

L'identifiant de chacun des concepts hérite d'un attribut code. L'identification d'une source d'information est un simple code, mais la manière d'affecter ce code, défini dans la norme [ISO 13584-26:2000], assure son unicité universelle. L'identifiant d'une entité est constitué d'un code et d'une référence à la source d'information qui l'a défini (attribut *defined_by*). Il est donc suffisant que chaque source d'information assure l'unicité des codes d'entités qu'elle attribue pour assurer une unicité globale des instances de *class_BSU*.

Enfin, l'identification d'une propriété est constituée d'un code et d'une référence à la *class_BSU* de la classe où elle est définie. Il est donc suffisant pour assurer l'unicité de cet identifiant que la source d'information attribue des codes de propriétés uniques pour chacune des classes que cette source d'information définit.

Outre le domaine de composants, cette même approche peut être utilisée dans d'autres domaines où des ontologies "standard" sont susceptibles d'être définies pour les entités et les attributs faisant l'objet de consensus. Dans le cas particulier des bibliothèques de composants industriels, de telles ontologies existent effectivement (exemple : la norme [IEC 1360-4:1995]), ou sont en cours de développement pour différents domaines de composants.

4.1.3 Représentation des données indépendamment des modèles conceptuels particuliers : méta-modèle

Afin d'éviter les problèmes liés à la diversité de modélisation conceptuelle, le modèle PLIB utilise la méta-modélisation. Ceci permet de modéliser le contenu d'un langage à objet (classes, attributs et contraintes) indépendamment du langage les manipulant. Cette manière de représenter les données permet à la fois d'accepter les diversités structurelles (exemple : composant de même type décrit par des propriétés différentes) et de les implanter dans des différents langages et formalismes cibles, facilitant ainsi leur portabilité. De plus, grâce à l'approche méta-modélisation, le modèle peut être échangé avec les données. Lors d'une importation d'une bibliothèque de composants, le système receveur peut n'importer que les données concernant les composants, ou bien importer en plus leurs modèles. Le premier cas s'applique lorsque le système receveur contient déjà le modèle de données. Dans le cas contraire, l'importation du modèle permet au système receveur d'interpréter les données importées.

Le schéma ci-dessous donne une version un peu simplifiée du méta-schéma générique d'échange de modèles, en supposant, par exemple, que seuls les types entier, chaîne et référence existent. L'instanciation d'un tel méta-schéma donne lieu à un modèle de données à part entière.

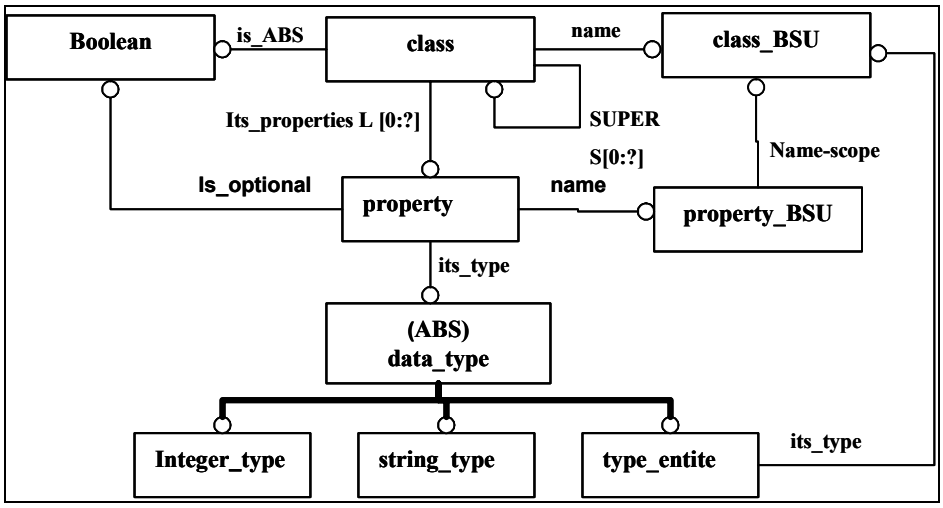


Figure 9 : méta-schéma générique de représentation des modèles [El Hadj Mimoune et al 03]

Dans ce schéma, chaque classe est identifiée par un identificateur absolu (*class_BSU* défini dans la Figure 8*)*. Elle possède un ensemble de propriétés et elle est caractérisée par un ensemble d'attributs. L'attribut *super* permet la représentation de l'héritage. Une classe peut être définie comme abstraite (attribut *is_ABS*). Chaque propriété est identifiée par un identificateur absolu (*property_BSU* également défini dans la Figure 8) et associée à un type (l'attribut *its_type*. Une propriété peut être obligatoire ou bien optionnelle (l'attribut *is_optinal*).

4.1.4 Typologie des propriétés

Afin de faciliter la sélection des composants, et de permettre de représenter le contexte dans lequel certaines propriétés ont été mesurées, le modèle permet la représentation de différentes catégories de propriétés. Le modèle PLIB distingue trois catégories de propriétés : propriétés caractéristiques, paramètres de contexte et propriétés dépendantes du contexte.

4.1.4.1 Propriétés caractéristiques

Cette catégorie de propriétés représente les propriétés intrinsèques des composants. Elles permettent la description des composants eux-mêmes et les valeurs de ces propriétés permettent de distinguer les instances les unes des autres. Par exemple, la propriété *diamètre* est une propriété caractéristique d'une vis.

4.1.4.2 Paramètres du contexte

Cette seconde catégorie de propriétés permet de caractériser le contexte dans lequel sera inséré un composant (par exemple, la charge réelle qu'un roulement va supporter dans un cas d'utilisation particulier). Ceci permet de représenter le savoir-faire des fournisseurs sur les composants qu'ils fournissent en donnant les grandeurs spécifiques caractérisant un problème particulier. Ainsi, les concepteurs, se basant sur ces grandeurs, peuvent résoudre d'une manière efficace les problèmes de conception et de choix de composants.

4.1.4.3 Propriétés dépendantes du contexte

Cette troisième catégorie de propriétés permet de caractériser les comportements des composants dans un contexte particulier. En effet, les valeurs de ces propriétés sont fonction des paramètres de contexte. Par exemple, la durée de vie d'un roulement à billes dépend directement et fondamentalement des forces axiales et radiales qui s'appliquent, ainsi que de sa vitesse de rotation.

4.1.5 Les sélecteurs de classes

La structuration en hiérarchie de classes des familles de composants (généralisation/spécialisation) constitue un moyen naturel d'accès aux familles cherchées. Malheureusement, cette structuration ne peut être réalisée que selon une seule perspective : celle qui est définie par la hiérarchie. Par exemple, on peut hiérarchiser des familles de roulements soit par leurs types (roulements à billes, à rouleaux, …) soit par leur mode de lubrification. Pour permettre les deux types d'accès on peut hiérarchiser ces familles par types et représenter le mode de lubrification comme propriété de classe ; c'est-à-dire ayant une valeur unique pour toute une classe. C'est ce que l'on appelle un sélecteur de classe. La valeur d'une telle propriété permet à l'utilisateur de sélectionner la (ou les) classe(s) dans laquelle (lesquelles) il veut sélectionner les composants.

4.2 Le méta schéma générique de représentation des instances de composants : le modèle de contenu PLIB

4.2.1.1 Méta modélisation des instances

Afin de permettre d'échanger les données de composants indépendamment de leurs modèles conceptuels particuliers, un méta modèle d'instances a été développé (le contenu). Ce modèle fournit la possibilité de décrire les composants eux-mêmes à travers les valeurs de leurs propriétés. Contrairement au modèle de l'ontologie qui décrit les familles de composants en intention, le contenu de PLIB permet une description en extension des propriétés.

Le modèle utilisé pour représenter des instances est un schéma générique susceptible de représenter n'importe quelle instance d'une famille décrite au niveau de l'ontologie. La première approche de représentation des instances, seule existante pendant le démarrage de cette thèse, est la représentation implicite. Une instance y est représentée, outre l'identificateur universel de sa classe par des formules de calcul, des expressions, des listes de listes, etc. Cette approche définit également des méthodes de sélection précises permettant la reconstitution des instances. Celles-ci n'ont pas d'existence effective dans le modèle d'échange et sont calculées lors de leur sélection. Nous présentons en détails cette approche de représentation dans le chapitre 2.

Les travaux de recherche que nous avons menés pendant l'avancement de cette thèse ont mis en évidence les difficultés de la mise en œuvre des données de composants ainsi représentées (première approche). Ceci est dû à la complexité de cette représentation. Une nouvelle approche de représentation a été alors développée et normalisée (norme [ISO 13584-25:2004]). Il s'agit de l'approche de représentation

explicite des instances. Dans cette approche, chaque instance est représentée par des couples (propriété, valeur). Ses instances sont décrites dans des fichiers physiques du langage EXPRESS qui sont représentés dans un format neutre (voir annexe). Chaque propriété est identifiée par son identifiant universel, et la valeur est représentée comme une valeur appartenant à l'un des types de base. Des détails concernant cette approche de représentation des instances sont donnés dans le chapitre 4.

4.2.1.2 Référence entre le contenu et l'ontologie

Comme nous l'avons précisé auparavant, l'existence de deux méta-modèles (représentation des modèles et les instances) offre la possibilité d'échanger les données et leurs modèles d'une manière complètement indépendante. Les références à l'ontologie au niveau du contenu sont réalisées à travers des identifiants universels (les BSU). De cette manière, les systèmes qui manipulent les mêmes bibliothèques de composants (même ontologie) peuvent facilement interpréter les données du contenu sans avoir besoin d'échanger l'ontologie.

5 Intégration des données de produit et de données de composant

Nous avons présenté dans les parties 2 et 4 deux approches de modélisation de données complexes des entreprises industrielles : une pour les données de produits et l'autre pour les données de composants. D'une manière assez générale, et notamment dans les domaines de la mécanique et de l'électronique, les produits sont constitués de composants préexistants. Actuellement, les informations concernant les données de composants ne sont représentées que très partiellement dans les systèmes d'informations (SI) contrôlés par les SGDTs. En effet, les composants sont généralement stockés dans des bases de données dites bases d'articles. Dans ces bases, les informations qui y sont représentées sont les mêmes pour tous les composants. Elles se limitent donc à des informations de base sur tous les composants (la référence du composant, son fournisseur, …), et d'une propriété de type chaîne de caractères souvent appelée "désignation" et qui contient sous forme textuelle, et souvent plus ou moins codée (concaténation du code du composant avec les valeurs de ses propriétés d'identification, par exemple), toute la description technique du composant. Cette solution n'offre que des méthodes de sélection très rudimentaires, et a donc pour effet de provoquer la multiplication de composants similaires.

Au contraire, l'utilisation de bibliothèques de composants informatisées selon le modèle PLIB permettrait de représenter d'une manière complète tout le savoir-faire des fournisseurs de composants. L'intégration de données de composants et de données de produits permettrait donc de mettre ce savoir-faire à la disposition des concepteurs de nouveaux produits. Ces concepteurs pourraient par exemple facilement sélectionner un composant pour l'insérer dans leurs modèles de produit. Ils pourraient également substituer un composant interchangeable à un autre. Ceci serait en particulier très efficace dans la phase de maintenance lorsque le composant défectueux doit être remplacé. Il le serait également dans la phase de production pour substituer un composant manquant dans le stock par son substitué.

5.1 Convergences et divergences entre les deux modèles

Les deux modèles décrivant les composants et les produits possèdent aussi bien des convergences que des divergences.

5.1.1 Convergences

De point de vue du domaine couvert, les deux modèles visent à représenter un même univers du discours concernant des objets techniques (vis, voitures, …). Les deux modèles décrivent des objets matériels (produit ou composant) en utilisant une approche objet. Dans les deux modèles visant à représenter les données de composants ou les données de produits, plusieurs représentations des objets techniques sont fournies. Le modèle PLIB permet également d'associer aux définition par propriétés (présentés dans les paragraphes 3 et 4) des représentations techniques, schématiques ou modèles de calcul éléments finis définies par une autre catégorie de classe, éventuellement associée à des fichiers externes. Le modèle de données de produit permet différentes représentations d'un produit tout au long de son cycle de vie différentes vues).

5.1.2 Divergences

Les données de composants sont décrites d'une manière dynamique (les valeurs de certaines propriétés varient en fonctions du contexte d'insersion des composants). Cette description permet la représentation des comportements de composants dans des contextes d'insertion différents (durée de vie d'un roulement par exemple). Tandis que les données de produits sont décrites d'une manière statique (représentation de produit essentiellement). Un autre point de divergence concerne la modélisation des données de composants et celles des produits : d'un coté, la modélisation des données de composants est basée en grande partie sur des propriétés. D'un autre coté, la modélisation des données de produits est basée sur des représentations. De plus la norme PLIB permet de décrire et d'échanger des ontologies pour les composants (les familles de composants et leurs propriétés) en plus des données concernant les composants eux même (contenu de la bibliothèque de composants).

Le principal point de divergence identifié est que les données de produits sont représentées d'une manière explicite. Les données de composants telles qu'elles sont définies dans la partie 24 de la norme PLIB [ISO 13584-24:2003] sont représentées, elles, d'une manière implicite.

5.2 Approche d'intégration

Nous avons montré dans les paragraphes précédents que les SGDT sont basés sur des SGBD sur lesquels ont été ajouté des fonctionnalités spécifiques à la gestion des données techniques. Ce constat impose de choisir un SGBD comme un moyen d'intégration des données de composants et des données de produits. Ainsi, l'étape la plus déterminante de l'intégration devient celle de l'implémentation du modèle PLIB dans une base de données cible. C'est pour cette raison qu'une grande partie de ce

travail est orientée sur ce point là. Nous présenterons, dans ce qui suit, différents types de bases de données et nous en choisirons une pour implémenter le modèle PLIB.

5.3 Différents types de bases de données

Depuis l'avènement de l'informatique, des systèmes d'organisation de données ont été développés. Les fichiers (systèmes sommaires d'organisation des données), les SGBD hiérarchiques et les SGBD réseaux ont dominé le monde de la gestion des données jusqu'au milieu des années 80. Puis les SGBD Relationnels (SGBDR) ont dominé le marché des bases de données et continuent à le dominer. Entre temps des SGBD orientés objet (SGBDOO) ont fait leur apparition. Pour des raisons de performances en particulier, les SGBDOO n'ont pas pu réellement percer. Conscients des avancées apportées par l'approche orientée objet, les développeurs des SGBDR ont commencé une intégration des principes de l'approche orienté objet dans leurs systèmes. Une nouvelle génération de SGBD a alors vu le jour : il s'agit des SGBDRO. Quelque soit le modèle choisi, tout système de gestion de bases de données doit posséder un certain nombre de caractéristiques permettant une gestion appropriée des données qu'il renferme. Selon [Graham 91]

, [Delobel et al 91] et [Benzaken et al 93], tout système de gestion de bases de données doit posséder les caractéristiques suivantes :

- la capacité à gérer des données persistantes,
- la capacité à gérer un volume important de données et à accéder à une grande quantité de données de façon performante,
- le support d'un modèle de données abstrait (conceptuel) qui permet la structuration des données représentées dans le système et l'établissement de liens entre ces données,
- le support d'un langage de haut niveau ou langage de requêtes. "*Ces langages de requêtes peuvent être utilisés de façon interactive* [dynamique] *par des utilisateurs pour consulter une base de données ou faire des modifications* " [Delobel et al 91],
- le support pour la sécurité des données. Un SGBD doit être capable de protéger les données qu'il contient contre les "agressions" extérieures. Ces agressions peuvent être de nature physique (pannes d'équipements ou de logiciels) ou de nature humaine (accès non permis à des données, manipulations malveillantes ou non, etc.),
- la garantie de la cohérence de données par rapport à la réalité qu'elles représentent et la vérification de leur consistance. Ceci est réalisé par l'application de contraintes d'intégrité ainsi que par le typage fort des données.

- le partage des données entre plusieurs utilisateurs. Le SGBD offre les mécanismes de contrôle de ce partage de données, de détection des conflits éventuels d'accès, et donne les moyens de les résoudre.

5.3.1 Bases de données relationnelles

Le modèle relationnel s'est avéré très adapté aux applications de gestion avec son système de modélisation formé de tables dont les colonnes prennent des valeurs alphanumériques. Les concepts du modèle relationnel (tables, relations, opérateurs, etc. sont simples et aisément compréhensibles par n'importe qui. De plus, la longue expérience dont bénéficient maintenant les SGBD relationnels a permis de les optimiser et donc de les rendre très performants. En outre, le modèle relationnel est basé sur un modèle mathématique très cohérent : il s'agit de l'algèbre relationnelle.

Un autre point fort du modèle relationnel est l'existence de SQL, un langage standardisé permettant la création et la manipulation de bases de données. Le langage SQL est bien adapté aux architectures client-serveur, il intègre la gestion des transactions, indispensable lors d'accès concurrents aux données ainsi que pour leur sécurité.

Par contre ce modèle est critiqué pour sa pauvreté sémantique qui exige en particulier dans bien des cas d'éclater entre plusieurs relations la description d'une entité et qui limite les attributs à des types de base.

5.3.2 Bases de données objet

De nouvelles méthodes de modélisation telles que OMT [Rumbaugh et al 91], UML [Jacobson et al. 99], EXPRESS [ISO 10303-11:1994] ont été développées ces dernières décennies. Toutes ces méthodes sont basées sur une approche orientée objet. Cette nouvelle approche a permis une représentation de modèles de données plus proche du monde réel. Elle a été exploitée en premier lieu dans des langages de programmation orientés objet, ce qui explique que ce type de base de données a été très peu utilisé au moins dans les grande entreprises industrielles pour gérer les données de produits. Le plus grand inconvénient, dans ces cas, est que les objets disparaissent une fois les programmes exécutés. Les bases de données à objets sont venues combler cette faille et permettre un stockage permanent des objets (persistance d'objets). Plusieurs systèmes de gestion de bases de données objet ont été développés tels que O2, OBJECTSTORE, etc. Ces bases de données supportent tous les concepts de la modélisation orientée objet et intègrent les fonctionnalités offertes par les SGDBR (sécurité de données, cohérence, …). Un langage de manipulation d'objets a été par ailleurs développé pour garantir la sélection, la mise à jour des données, etc. Il s'agit du langage OQL (Object Query Language) [Bancilhon 94]. Par contre ces bases de données n'ont pas réussi à offrir le même niveau d'efficacité dans le traitement des requêtes que le modèle relationnel.

5.3.3 Bases de données relationnelles-objet

Les SGBDOO n'ont pas pu faire leur place sur le marché des SGBD. Mais malgré

le succès des SGBDR et leur performance, ceux-ci ne sont pas sans insuffisances. Parmi ces insuffisances, le caractère atomique des attributs et l'absence de représentation de la relation de généralisation/spécialisation. Pour pallier ces problèmes, et bénéficier des avancées apportées par l'approche orientée objet, les développeurs des SGDBR ont commencé, dès les années 90, à étendre leurs systèmes pour supporter le paradigme objet. Ceci permet de bénéficier des atouts des deux modèles. Dans ces systèmes, les objets sont stockés dans des tables relationnelles, ce qui permet d'en indexer les accès. L'avantage de ces systèmes par rapport aux SGBDOO est la rapidité d'accès et la compatibilité ascendante avec les SGBDR. De plus, leur langage de requêtes est basé sur la norme SQL3 [ISO 9075-2:1999] qui est une extension du langage SQL à des concepts objet. Tous les SGBDR, tels que Oracle, PostgreSQL, Informix sont en train de migrer progressivement vers des SGBDRO.

Un autre point fort de ces systèmes est l'intégration de langages procéduraux (PL/SQL) permettant le renforcement des contraintes susceptibles d'être exprimées sur des données et le stockage de certains traitements simultanément avec les données. Par contre, l'intégration des concepts objet se fait de façon différente, et dans un ordre différent selon les systèmes. Cette introduction provoque une nouvelle incompatibilité entre les systèmes SGBDRO disponibles dans le commerce.

5.3.4 L'approche choisie : objet relationnelle

Notre objectif étant de permettre la gestion intégrée des composants et des produits au sein des entreprises industrielles, et ces derniers étant en règle générale gérés par des systèmes basés sur des SGBDR (et des SGBDRO), choisir une plateforme différente ne pourrait que nuire à l'atteinte de notre objectif. Pour implémenter le modèle PLIB, nous avons donc choisi le modèle relationnel-objet. En effet, ce système offre dès a présent l'héritage de table, facilitant la représentation de l'héritage au sens de l'approche objet, largement utilisé dans le modèle PLIB. Ce choix nous permet également de profiter des performances du modèle relationnel qui est plus mature et plus efficace. De plus, la place que commencent à occuper les SGBDRO permet à cette étude d'aboutir à la réalisation d'un prototype de système opérationnel et crédible.

6 Conclusion

Nous avons présenté dans ce chapitre deux approches pour la modélisation des données de produits. La première approche est l'approche de modélisation par représentation qui vise particulièrement les produits conçus et/ou fabriqués dans l'entreprise. Dans cette approche, le produit possède plusieurs représentations. Chaque représentation modélise un point de vue particulier du produit pour les différents acteurs et dans diverses phases de son cycle de vie. La seconde approche est l'approche de modélisation par propriétés. Elle apparaît nécessaire pour représenter les données de composants. En effet, ceux-ci doivent pouvoir être retrouvés et comparés facilement, puis choisis pour être utilisés dans la conception et la fabrication de produits. Or la méthode de recherche la plus efficace est celle effectuée par l'intermédiaire de valeurs de propriétés.

Les données de produit ont été visées par des travaux de normalisation dans les instances nationales et internationales de standardisation. Le standard STEP vise la représentation sous forme de données de produits conçus et/ou fabriqués dans l'entreprise (appelé "données de produits"). La description sous forme de données de produits destinées à être réutilisées que nous appelons des composants fait l'objet de la norme PLIB. Elle utilise la seconde approche de modélisation. L'objectif de notre travail est d'intégrer ces deux modèles qui sont à la fois voisins et différents. Puisque les données de produits sont gérées par des SGDT (Systèmes de Gestion de Données Techniques), eux même basés sur des SGBDR (et des SGBDRO), il apparaît clairement que l'intégration entre les deux modèles (modèles de produits et ceux des composants) suppose de pouvoir représenter les données définies par le modèle PLIB dans des SGBDR ou des SGBDRO. De plus, l'implémentation du modèle PLIB dans un SGBDR ou SGBDRO permettra l'émergence d'un système de gestion de données de composants (ou gestionnaire de données de composants) jusque là inexistant. L'avantage de ce gestionnaire est qu'il sera compatible avec la norme PLIB.

Nous avons identifié deux problèmes principaux à résoudre. Le premier concerne l'intégration entre le modèle de données de produits et le modèle de données de composants. Le second concerne la réduction des divergences constatées entre les deux modèles et en particulier celle liée à la modélisation explicite des données de produits et la modélisation implicite des données de composants.

Dans le chapitre 2 nous présentons en détail le modèle de représentation des instances de composants basé sur l'approche implicite. Nous présentons également différentes approches possibles pour implémenter ce modèle PLIB dans un SGBD (choisi comme moyen d'intégration). Dans le chapitre 3, nous détaillons les différentes étapes d'implémentation en suivant l'approche d'implémentation la plus avantageuse parmi les trois approches étudiées au chapitre 2. En avançant dans l'implémentation les difficultés d'implémentation liées au second problème identifié deviennent plus évidentes. Ainsi, la nécessité de représenter les données de composants d'une manière explicite deviendra plus pressante. Partant de ce constat, un nouveau modèle de représentation explicite des instances (méta-schéma d'instance) a été développé et normalisé dans la norme PLIB. Nous présentons dans le chapitre 4 et 5 ce modèle puis son implémentation dans un SGBD.

Chapitre 2

Représentation implicite des données de composants : modèle et implémentation

Résumé : *Dans certains cas, le nombre d'instances possibles dans une famille de composants peut être très important voir même infini. C'est le cas, par exemple, d'une famille de composants ayant une propriété dépendant d'un paramètre de contexte dont les valeurs appartiennent à un domaine continu. La factorisation de l'information dans un modèle implicite consiste donc à modéliser la manière dont les valeurs d'une telle propriété sont calculées plutôt que de fournir toutes les valeurs possibles (ce qui serait impossible). Cette approche offre un moyen très puissant pour modéliser les instances sans avoir à les énumérer. Elle permet également, dans les cas évoqués ci-dessus, d'éviter une explosion combinatoire des données. Cependant, elle est complexe à implémenter et à mettre en œuvre dans un contexte de base de données.*

Dans ce chapitre nous étudions les différentes possibilités de représentation d'une population décrite sous forme implicite de PLIB au sein d'une base de données cible. Deux méthodes sont d'abord étudiées. Elles consistent respectivement à représenter le modèle EXPRESS d'une manière intégrale et systématique ou, au contraire, à ne représenter effectivement que les instances de composants sous forme de n-uplets, le schéma de la base de données étant généré à partir du dictionnaire. Aucune de ces deux méthodes ne s'avérant satisfaisante, nous élaborons alors une troisième méthode, qualifiée d'approche hybride, où le dictionnaire est représenté au niveau méta (au niveau des descripteurs), et les composants sous forme de n-uplet relationnel. Les données, leur schéma, et leur spécification sont donc représentés au même niveau et accessibles simultanément dans une même requête. C'est cette approche qui sera ensuite retenue dans le cadre de ce travail et qui a d'ailleurs été ajoutée au modèle PLIB où deux méthodes de description des instances sont désormais disponibles : la description implicite et la description explicite.

1 Introduction

Nous avons vu que dans PLIB, les composants sont décrits à deux niveaux différents : au niveau de l'ontologie (dictionnaire) et au niveau instances (contenu). La description de l'ontologie permet l'identification des concepts (les familles des composants et leurs propriétés), la définition de leur structure (hiérarchie de classes) ainsi que les types de données des propriétés.

Le second niveau de description de composants est le niveau des instances. Dans ce niveau, les instances sont représentées d'une manière implicite. Ceci est réalisé en définissant les domaines de valeurs des propriétés, les relations qui existent entre ces propriétés (dépendance fonctionnelle et dérivation) et les méthodes de génération et de validation des instances. La génération et la validation d'instances passent par le contrôle de leur conformité vis-à-vis de la description de leurs classes au niveau de l'ontologie ainsi que par le contrôle des contraintes d'intégrité représentées dans le modèle implicite.

En effet, dans le cas d'une modélisation par représentation implicite, les instances licites d'une famille de composants sont représentées par des tables éclatées, des fonctions de dérivation, des opérateurs relationnels et des prédicats de validation d'instances. Une telle représentation constitue une caractérisation puissante permettant de représenter les instances ainsi que leur comportement dans leur contexte d'insertion.

Dans le cas de la durée de vie d'un roulement par exemple, celle-ci dépend directement des charges axiales et radiales exercées. On imagine bien que ces charges peuvent avoir un nombre important voire une infinité de valeurs. Pour représenter ce roulement, la solution préconisée est de représenter de telles propriétés par des fonctions permettant de les exprimer et de les calculer en se basant sur les valeurs des paramètres de contexte (également représentés dans le modèle) et éventuellement des valeurs d'autres propriétés.

La description implicite du contenu d'une bibliothèque de composants est modélisée dans la partie 24 de la norme PLIB [ISO 13584-24:2003]. Ce document inclut plusieurs schémas, formalisés en EXPRESS. Ils permettent la description du contenu effectif des familles de composants (des classes) décrites en intention au sein du dictionnaire de données PLIB (au niveau de l'ontologie). Cette description du contenu est représentée implicitement par des propriétés caractéristiques. Des ressources génériques permettant cette description y sont également définies. Elles permettent de modéliser :

- les domaines de valeur des propriétés,
- les variables et le mécanisme permettant de leur assigner des valeurs,
- les tables et les opérations algébriques sur celles-ci,
- des expressions permettant de représenter la connaissance procédurale,
- et donc, la description implicite des instances (les extensions de classe).

Nous détaillons dans ce chapitre les différents concepts représentés dans le modèle implicite de PLIB. Nous donnons ensuite les détails concernant l'mplémentation d'un tel modèle dans une base de données cible. Nous avons étudié trois approches d'implémentation. C'est l'approche hybride, consistant à implémenter l'ontologie PLIB au niveau des entités EXPRESS et le contenu par des n-uplets, qui a été retenue.

2 Première approche de modélisation du contenu des bibliothèques de composants : modèle implicite de PLIB

2.1 La classe, l'instance et l'occurrence

Le monde réel est vu à travers de hiérarchies des classes. Cela constitue une organisation naturelle du savoir humain [Coad et al 92] [Minsky 86]. La classe permet de rassembler un certain nombre d'objets ayant des caractéristiques communes (même structure et même comportement) afin de faciliter leur gestion [Gardarin et al 94].

Dans le modèle PLIB, les composants sont définis à trois niveaux : la classe, l'instance et l'occurrence. La classe constitue un modèle général pour les objets techniques (i.e. composants). Chaque classe définit les instances licites, c'est-à-dire celles qui correspondent à des objets matériels existants. Cette définition, en général assez complexe dans le cas des composants en ingénierie [Pierra 90], est effectuée sous forme de contraintes d'intégrité qui caractérisent les instances licites.

Une fois l'instance choisie et insérée dans un modèle de produit particulier, l'information qui la caractérise change de nature. Une telle entité est appelée occurrence. Elle est porteuse de nouvelles informations. En effet, certaines informations, appelées caractéristiques d'insertion, décrivent l'espace de modélisation du produit auquel l'instance appartient (une matrice de positionnement par exemple) ou caractérisent le contexte dans lequel l'occurrence est insérée (charge exercée sur un roulement) [Pierra 95]. *A contrario*, les propriétés indépendantes du contexte d'insertion peuvent être factorisées au niveau d'une instance référencée éventuellement par plusieurs occurrences. D'un point de vue informatique, l'occurrence est très importante dans les modèles de produits car elle permet de gérer la maintenance du produit notamment lors de son entretien (changement d'une pièce après un temps d'utilisation défini dans un contexte d'utilisation particulier ou lors d'un changement de pièces défectueuses).

L'instance et l'occurrence correspondent respectivement à l'article et à l'exemplaire dans les SGDT [Maurino 93]. D'un point de vue modélisation, l'instance trouve un sens dans le modèle de données de composants, alors que l'occurrence intervient plutôt dans le modèle de données de produits. En effet, elles sont toutes les deux des représentations d'objets techniques selon deux points de vue différents.

2.2 Contraintes d'intégrité

Comme nous l'avons précisé ci-dessus, la représentation implicite est fondée sur un ensemble de contraintes d'intégrité. Une contrainte d'intégrité appelée également invariant de données est une propriété définie formellement dans le modèle. Le modèle PLIB distingue deux types de contraintes :

1- les contraintes fonctionnelles,

2- les contraintes logiques.

Les contraintes fonctionnelles sont des propriétés calculées par le système en utilisant des fonctions (propriétés dites "dérivées"). Les contraintes logiques sont les propriétés vérifiables par le système (contraintes assertionnelles décrites par des prédicats). Ce type de contrainte n'exprime pas des fonctions mais des relations au sens le plus général.

Ces contraintes, exprimées par des fonctions ou des assertions logiques, constituent la connaissance procédurale de la représentation implicite des données de composants. Afin de favoriser l'échange entre des systèmes hétérogènes, elles sont représentées par la technique de méta-programmation (*cf.* section 2.7.3). Dans une telle représentation, chaque classe est associée à un ensemble de contraintes permettant de décrire d'une manière implicite l'ensemble de ses instances (définition d'un ensemble en extension). Nous présenterons dans les sections suivantes les modèles formels de PLIB permettant la représentation des contraintes d'intégrité.

2.3 Représentation des propriétés dans le modèle implicite

Comme nous l'avons précisé dans le chapitre 1, la modélisation par propriétés constitue une large partie de la modélisation de données de composants. Nous avons abordé dans ce même chapitre la classification des propriétés d'un point de vue ontologique ou sémantique. Effectivement, au niveau de l'ontologie, les propriétés représentent les caractéristiques importantes permettant la description des composants dans le contexte le plus général de leur utilisation (propriétés caractéristiques, propriétés dépendantes du contexte et paramètres de contexte) (cf. Chapitre 1). En revanche, dans ce paragraphe, la classification des propriétés se fonde sur un autre point de vue à savoir, la représentation de populations de classes décrites au niveau de l'ontologie (associer des valeurs à des propriétés et valider les instances). Cette classification, que nous détaillerons ci-dessous, constitue un choix de représentation visant, d'une part, à définir un cadre efficace pour la représentation implicite des instances, et, d'autre part, à offrir un moyen efficace pour la sélection de ces instances. Dans la partie 24 de la norme PLIB [ISO 13584-24:2003], une typologie précise a été élaborée. Elle offre une base solide pour le développement de gestionnaires de bibliothèques de composants. En effet, cette typologie joue le rôle de protocole entre les fournisseurs d'informations (i.e. de composants) et les développeurs de gestionnaires de bibliothèques de composants. Elle définit pour les uns, la manière de représenter les informations concernant les composants (plus spécifiquement la population des classes) et, pour les autres, le processus à suivre pour sélectionner ces mêmes composants.

Les propriétés pour lesquelles les valeurs sont fournies, dans une description implicite de l'extension de classes, sont classifiées dans trois catégories [ISO 13584-24:2003]:

Les propriétés sélectionnables : dont les valeurs doivent ou peuvent être fournies par l'utilisateur lors de la sélection d'une instance (i.e. un composant). Ces propriétés sont choisies parmi les propriétés caractéristiques de l'objet (propriétés non dépendantes de contexte *cf.* chapitre 1) et les paramètres de contexte. Les propriétés choisies parmi la première catégorie sont des propriétés d'indentification de la classe. Elles permettent d'identifier chaque instance sans aucune ambiguïté.

Les propriétés d'identification : elles représentent un sous ensemble des propriétés sélectionnables. Elles permettent l'identification non ambiguë de chaque composant. Elles jouent le rôle des clés primaire dans les base de données relationnelles.

Les propriétés dérivées : les valeurs de telles propriétés dépendent de certaines valeurs de propriétés sélectionnables et éventuellement de certaines valeurs d'autres propriétés dérivées. Dans ce dernier cas (dépendance de propriétés dérivées), la dépendance doit être acyclique. Les propriétés dérivées peuvent être soit des propriétés caractéristiques (longueur non filetée d'une vis qui est la soustraction de la longueur filetée de la longueur totale), soit des propriétés dépendantes du contexte (durée de vie d'un roulement) ou encore des paramètres de contexte.

2.4 Domaines des propriétés

Les domaines des propriétés sont définis d'une manière globale dans la partie dictionnaire (au niveau de l'ontologie). Dans cette partie, chaque propriété est associée à un type de données qui représente, à un niveau d'abstraction élevé, le domaine de valeurs que peut prendre cette propriété. Or, dans la partie du modèle qui décrit le contenu, c'est l'extension de classe qui est définie implicitement. Il est nécessaire de rendre les domaines de valeurs des propriétés plus précis et de les limiter aux seules valeurs décrivant des instances licites (qui correspondent à des objets techniques existants).

Dans le cas d'un roulement par exemple, le domaine de valeurs de son diamètre intérieur est défini dans le dictionnaire comme étant un réel associé à une unité de mesure. Alors que dans la partie du contenu, ce domaine est détaillé par une liste de valeurs qui représentent les diamètres caractérisant des roulements existants.

En général, le domaine de valeurs d'une variable (associé à une propriété dans le cas de données de composants) peut être soit complètement indépendant de tout autre domaine de valeurs, soit directement dépendant d'un ou de plusieurs autres domaines de valeurs. Dans la partie 24 de PLIB, le schéma EXPRESS (ISO13584_domain_resource_schema) traite de la modélisation des domaines de valeurs [ISO 13584-24:2003]. Ce schéma permet :

- la représentation des domaines des valeurs autorisées pour une ou plusieurs variable(s),
- la représentation de la dépendance entre les domaines de certaines variables avec les domaines de valeurs d'autres variables,
- la représentation de la dérivation d'un ensemble de valeurs d'une variable à partir de domaines de valeurs d'autres variables lorsqu'il y a une dépendance fonctionnelle entre la première variable et les autres,
- la représentation des dépendances entre les propriétés d'un produit et/ou d'un composant.

La représentation des instances est fondée sur des concepts généraux (domaines de valeurs, dérivation, prédicat, …). Chaque catégorie de ces concepts est modélisée par PLIB dans un schéma appelé "*resource*". Ces ressources sont indépendantes vis-à-vis du modèle de données de composants et pourraient être utilisées sans problème pour d'autres modèles de données. Le schéma défini ci-dessus (*ISO13584_domain_resource_schema*) est utilisé pour représenter à la fois les valeurs licites des propriétés de familles de composants et les fonctions de dérivation permettant de calculer les valeurs d'autres propriétés dites « propriétés dérivées ».

2.5 Représentation de la dépendance entre les propriétés

D'une manière générale, les domaines de valeurs des propriétés d'une classe pourraient être liés les uns aux autres par des relations. Deux concepts mathématiques différents, permettant d'exprimer les relations entres les valeurs de deux ensembles différents de variables, ont été utilisés dans PLIB [ISO 13584-24:2003]. Le concept le plus général est celui de la relation mathématique : le tuple, constitué par une liste ordonnée de valeurs des différentes variables. Il doit appartenir à un ensemble de tuples qui peuvent dépendre des valeurs des autres variables. La représentation de cette dépendance dans le modèle PLIB est expliquée en détail dans les sections 2.7.2.2 et 2.7.2.3. Cette relation est appelée « dépendance relationnelle » entre les deux ensembles de variables.

Afin d'illustrer ce concept, considérons une relation R[a, b, c, d, e, f], dont les valeurs du sous-ensemble de propriétés [d, e, f] dépendent des valeurs du sous-ensemble des attributs du complémentaire [a, b, c]. La dépendance relationnelle se traduit par le fait que pour chaque tuple constitué des valeurs des propriétés [a, b, c] correspond un ensemble de tuples constitués des valeurs des propriétés [d, e, f]. Il faut préciser qu'aucun des deux sous-ensembles ne constitue une clé primaire pour la relation R. Le sous-ensemble constituant une clé primaire peut contenir des propriétés appartenant aux deux sous-ensembles cités ci-dessus (par exemple a et b).

Le second concept permettant de représenter une relation entre deux ensembles de variables est la fonction. Lorsque les valeurs du premier sous-ensemble sont définies, les valeurs du second sous-ensemble sont fixées. En d'autres termes pour chaque tuple du premier sous-ensemble correspond essentiellement un tuple du second sous-ensemble. Une telle relation est appelée « dépendance fonctionnelle ». La cette dernière est un cas particulier de la dépendance relationnelle lorsque l'ensemble des tuples permis devient un singleton.

Dans la représentation implicite, les relations entre les différents domaines de variables sont représentées par plusieurs fragments d'information. Elles sont décrites principalement par l'entité *domain_restriction*. La Figure 10 présente la structure de cette entité.

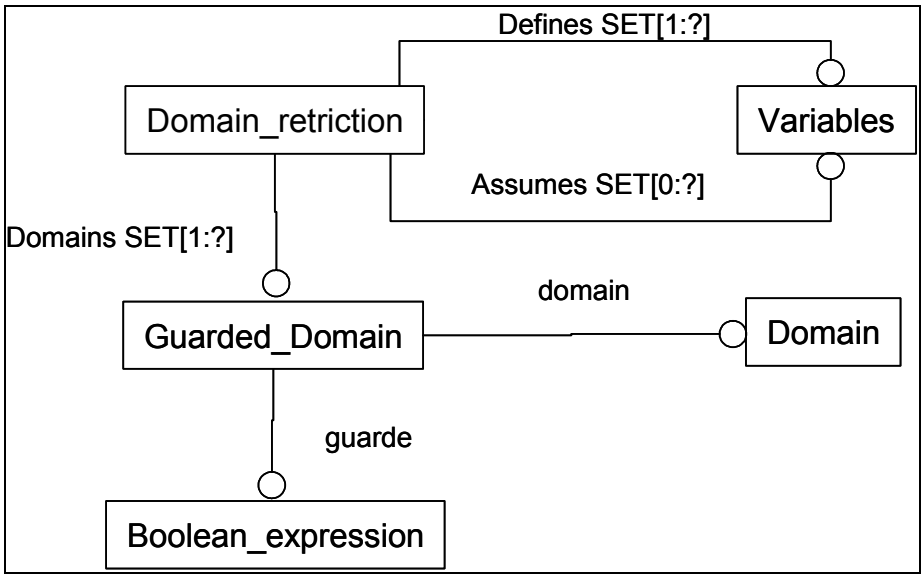


Figure 10 : représentation de la dépendance relationnelle dans la représentation implicite [ISO 13584-24:2003].

Les variables dont les valeurs sont sources de la dépendance sont représentées par l'attribut *assumes* de cette entité. Les variables dont les valeurs dépendent de la relation utilisée sont représentées par l'attribut *defines* (voir 2.7.2.2). Ainsi, dans le processus de sélection des composants, les valeurs des variables *assumes* sont données par l'utilisateur et celles de *defines* sont calculées par le système (au travers de dérivation). Par ailleurs, des gardes peuvent compléter cette définition pour contraindre le domaine de valeurs. Ces domaines sont alors représentés par l'attribut *domains* qui sont des domaines associés à des expressions booléennes représentant les gardes (*guarded_domain*).

2.6 Description de l'extension des classes dans une représentation Implicite

Comme nous l'avons précisé auparavant, le modèle de données PLIB fournit, d'une part, la description des composants (identification des concepts et définition des classes et leurs propriétés) au niveau de l'ontologie, et, d'autre part, une structure permettant la représentation et l'échange des instances. Une telle structure est appelée « schéma du contenu de PLIB ». En effet, chaque classe possède une définition en intention (structure de classes, propriétés, …) et une définition en extension (population ou ensemble d'instances). Dans le modèle implicite, les instances d'une famille de composants (i.e. une classe) sont modélisées d'une manière implicite par des propriétés caractéristiques sans décrire explicitement chaque instance de cette classe. Dans ce modèle, il est possible de reconstituer des instances, en se basant sur des valeurs de propriétés représentées par des domaines fournis (des listes éclatées, des tables et des fonctions de dérivation). Des contraintes permettent de vérifier que chaque instance représentée dans le modèle d'échange correspond bien à la définition de sa classe au niveau de l'ontologie, et qu'elle remplit bien les contraintes d'intégrité définies (au niveau du contenu) sur cette classe.

Dans la spécification formelle du modèle implicite, l'extension d'une classe est représentée par l'entité EXPRESS *implicit_model_class_extension.* Cette entité permet de caractériser les instances d'une famille de composants (définition en extension). La

Figure 11 montre la description, en langage EXPRESS-G, de l'extension de classe dans le modèle implicite.

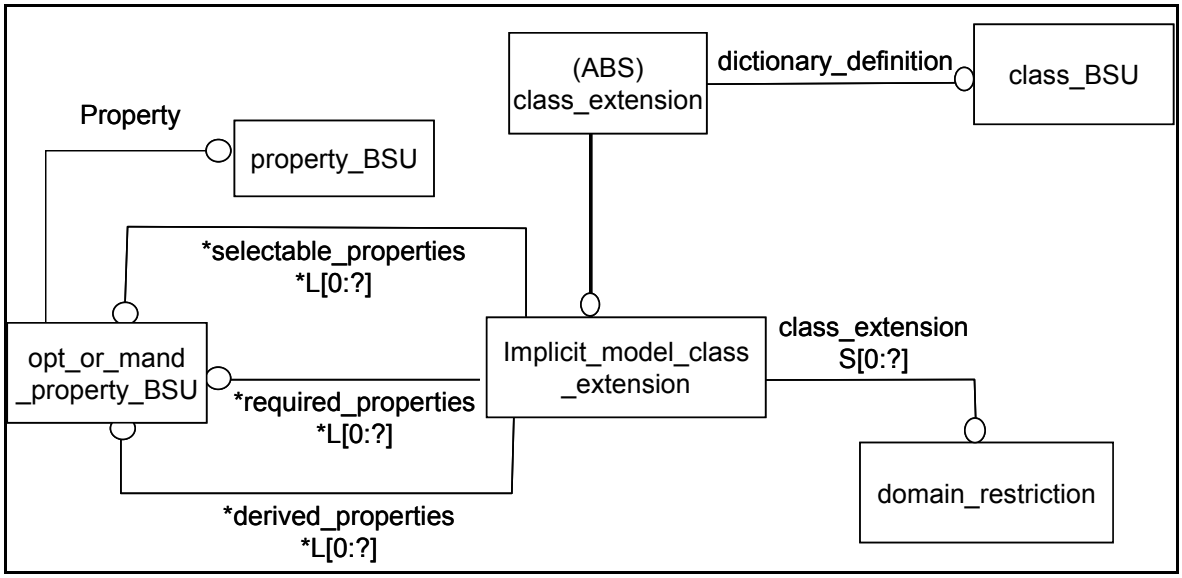


Figure 11 Modèle EXPRESS-G simplifié de l'extension de classe dans une représentation implicite [ISO 13584-24:2003].

L'entité *implicit_model_class_extension* référence l'identifiant universel (i.e. *class_BSU*) de la classe (du niveau de l'ontologie) pour laquelle elle décrit la population. Cette référence est réalisée par l'attribut *dictionary_definition* hérité de l'entité *class_extension* (le mot clé *ABS* indique que la classe est abstraite). En effet, cette référence établit le lien entre la population d'une classe et sa description au niveau du dictionnaire de données (au niveau de l'ontologie). Les différentes propriétés (sélectionnables, dérivées et exigées) sont définies comme étant des *opt_or_mand_property_BSU*. Cette entité est la spécification des propriétés au niveau du contenu. Elle permet entre autre d'exprimer si la valeur d'une propriété est optionnelle ou obligatoire. Elle référence la description de propriété au niveau du dictionnaire (de l'ontologie) via l'identifiant universel de la propriété (*property_BSU*). Par ailleurs, l'extension de la classe proprement dite (i.e. ensemble d'instances) est représentée par un ensemble de domaines de valeurs à travers l'attribut *class_extension*. La représentation de ces domaines dans le modèle implicite de PLIB a été présentée dans la section 2.5.

2.7 Ressources utilisées dans la modélisation implicite de PLIB

Dans les catalogues de composants sur papier, les familles de composants sont décrites naturellement à travers des tables, des expressions de formules et des schémas (illustrations). Dans ce qui suit, nous détaillerons pour ces concepts le modèle ou le mécanisme permettant de les reproduire dans la norme PLIB sous forme de modèles qui peuvent être traités automatiquement.

2.7.1 Structure de tables dans le modèle implicite de PLIB

D'une manière générale, l'ensemble des populations des familles de composants est représenté dans des tables (réelles ou virtuelles i.e. calculées à partir d'autres tables), appelées « tables de définition » de famille de composants. Dans ces tables, une ligne représente un composant (ou une instance) et une colonne représente une propriété de composant [Pierra 94]. Dans certains cas, de telles tables peuvent être très volumineuses. Par conséquent, ces tables sont représentées à la fois par un ensemble de tables élémentaires et par des opérations de jointures de tables permettant la construction de la table globale. A la différence d'un modèle de bases de données relationnelles, où la table constituant la définition en extension d'une relation est l'entité de base pour contenir les données, dans le langage EXPRESS, utilisé par PLIB, cette notion n'existe pas. La structure de table a dû être représentée dans le modèle implicite de PLIB. Un ensemble de listes de valeurs a permis de représenter cette structure de table. Chaque liste représente une colonne de table et est associée à une propriété d'une famille de composants. Des règles de jointure permettant la construction de la table globale y sont également définies.

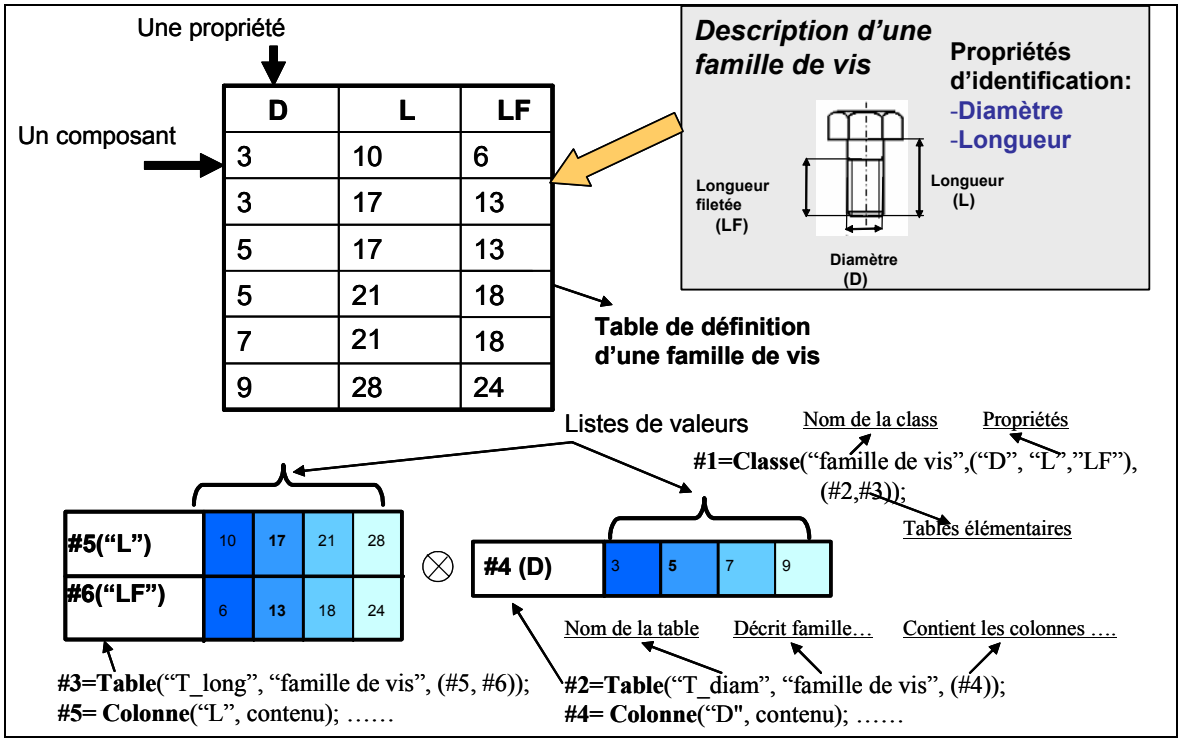


Figure 12 : Représentation simplifiée des tables dans le modèle PLIB

La Figure 12 présente d'une manière très simplifiée la structure de table dans la représentation implicite du contenu PLIB. D'abord, la table de définition d'une famille de vis est représentée par deux tables élémentaires *T_diam, T_long* en plus des règles de jointure (non représentées sur la figure). Chaque table, identifiée par un identifiant universel (représenté par un nom de table sur la figure), fait référence à la famille de composants qu'elle décrit via son identifiant universel. Une table est constituée d'un ensemble de colonnes représentées par des listes de valeurs. Chaque colonne référence la propriété pour laquelle elle décrit le contenu.

2.7.2 Filtres et fonctions de dérivation

Comme nous l'avons précisé dans les paragraphes précédents, dans le modèle implicite de PLIB décrivant les instances, les données de composants sont représentées à l'aide de tables élémentaires, de fonctions, d'expressions, etc. Il est évident qu'en procédant ainsi pour représenter des données, il faut définir des règles précises permettant de reconstruire la table globale et ainsi les instances. Cette reconstruction de table doit donc se limiter à ne fournir que des composants licites (c'est-à-dire ceux qui existent réellement). Ces règles permettent de restreindre les domaines de définition des propriétés et d'éliminer les composants superflus générés, par exemple, par un produit cartésien entre les tables élémentaires.

2.7.2.1 Les filtres

Les filtres sont des contraintes d'intégrité supplémentaires, présentées au niveau de la description des contenus (instances), exprimées sur les données représentées par le modèle implicite. Ils sont représentés par des expressions logiques et permettent de restreindre le nombre d'instances possibles. Ils offrent également un support au processus de sélection de composants particuliers par l'utilisateur. Ainsi, en appliquant ces filtres, le nombre d'instances offertes à l'utilisateur lors de sa sélection est réduit. Par conséquent, l'utilisateur est guidé dans son choix en réduisant au fur et à mesure le nombre de composants à choisir. Un filtre est représenté de la même façon qu'un domaine par l'entité *domain_restriction*. Il permet d'exprimer différents types de contraintes allant des plus simples aux plus complexes (par exemple $d > 5$ et/ou $d = d\_e$ où d et d_e sont respectivement le diamètre d'une vis et le diamètre d'un écrou). L'utilisation de valeurs des différentes variables dans une contrainte ainsi que la complétude du modèle représentant les expressions permet de représenter et d'échanger différents types de contraintes (cf. 2.7.3).

2.7.2.2 Les dérivations par tables

Dans le modèle PLIB les dérivations par tables permettent de représenter en extension une dépendance fonctionnelle identifiée entre propriétés. Deux propriétés dépendant fonctionnellement l'une de l'autre peuvent être représentées à l'aide d'une table dont la source de dépendance constitue la clé. Ce type de représentation permet de factoriser l'information. Lors d'une action de sélection, l'utilisateur fournit les valeurs des propriétés indépendantes. Les valeurs des propriétés dépendantes sont dérivées à partir de ces tables.

La Figure 13 illustre la représentation de la dérivation par tables dans le modèle implicite. Elle montre comment la propriété « longueur filetée » d'une famille de vis peut être dérivée à partir de la propriété indépendante « longueur ».

Le domaine des valeurs licites de la longueur filetée est représenté par l'entité *Functional_domain_restriction*. L'attribut *Domain* de l'entité *Guarded_domain* permet de représenter cette dérivation. L'attribut *Defines* de cette même entité (*Domain_restriction*) permet de représenter la propriété dérivée (la longueur fileté) tandis que l'attribut *assumes* représente la(les) propriété(s) permettant de dériver cette propriété (dans ce cas la longueur).

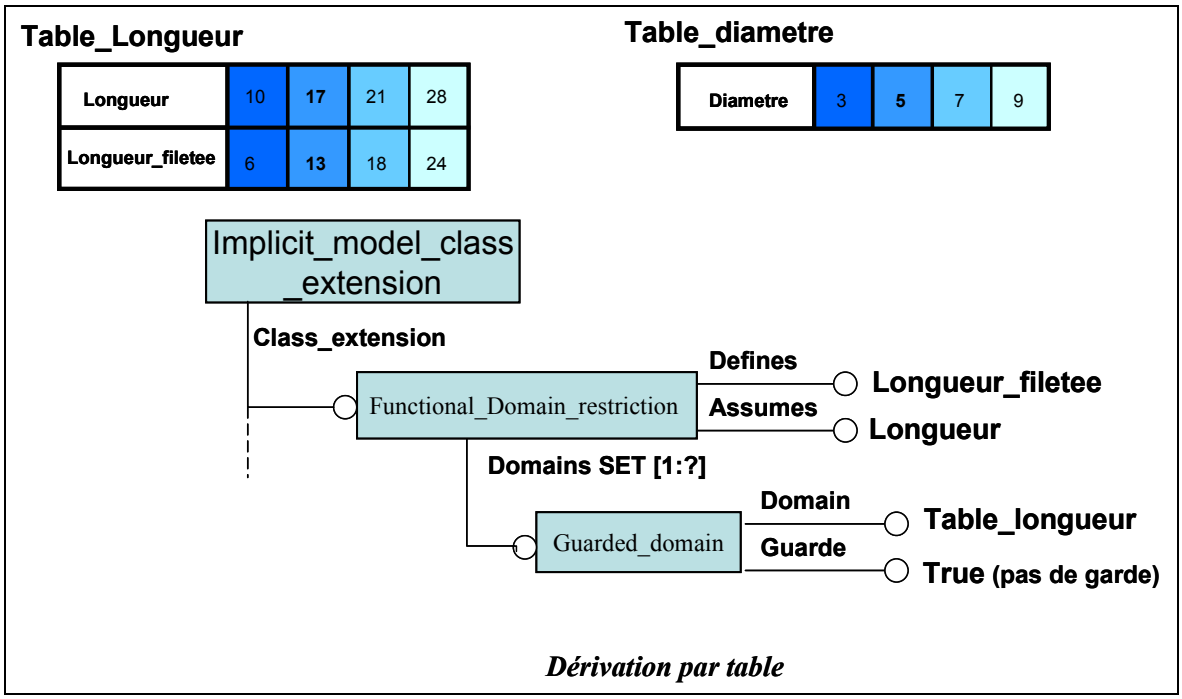


Figure 13 : la représentation de la dérivation par table.

Les contraintes sur le domaine de valeurs de la propriété dérivée (la longueur filetée) sont représentées par des gardes (l'attribut *Guard* de l'entité *Guarded_domain*). Le domaine de cette propriété dérivée pourrait ainsi être partitionné en un ou plusieurs domaines gardés (*Guarded_domain)*. Les gardes[1] sont représentés par des expressions booléennes vraies pour toute valeur licite. Ces expressions booléennes sont formellement représentées dans le modèle (cf. 2.7.3) (Ici, il existe une seule table de dérivation de sorte que la garde soit "*True*").

2.7.2.3 Les dérivations par fonction

Outre la possibilité de modéliser les domaines de valeurs par des tables, les valeurs de propriétés dérivées peuvent être calculées par une fonction à partir des valeurs d'autres propriétés. Lorsque les dérivations sont exprimées de cette manière, le modèle de données bénéficie des deux avantages suivants:

- le nombre de données stockées (pour les propriétés dérivées) est réduit à une simple fonction. Cette dernière sera évaluée lors de la sélection;

- une représentation dynamique des données, permettant d'accroître le pouvoir de sélection de ces données, est fournie grâce au recours à l'évaluation tardive des propriétés caractéristiques (dérivation et contraintes) au moment de la sélection.

[1] Garde est une traduction du terme anglais « *guard* » utilisé dans la norme PLIB pour désigner des contraintes appliquées sur un domaine donné pour le restraindre.

La Figure 14 nous montre un exemple de représentation, dans le modèle implicite, d'une propriété dérivée à partir d'une fonction. Il s'agit de la hauteur de la tête d'une vis. Elle est calculée, par une simple expression, à partir du diamètre de la vis. Dans ce cas, nous avons recours à l'entité *Functional_domain_restriction,* sous type de l'entité *Domain_restriction,* pour représenter cette information.

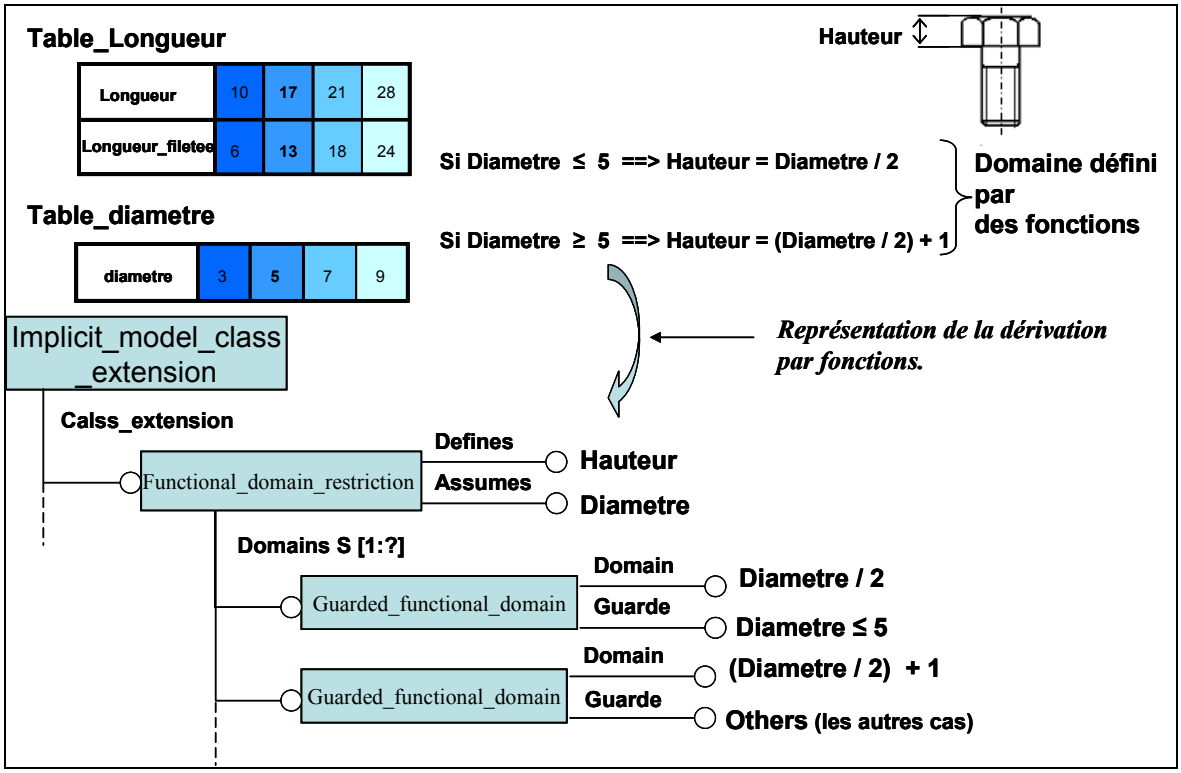


Figure 14 : représentation de la dérivation par fonction dans le modèle implicite.

Comme pour la dérivation par tables, la propriété dérivée est représentée par l'attribut *Defines*. Les propriétés, à partir desquelles celle-ci est calculée, sont représentées par l'attribut *Assumes* (voir la Figure 14). Les fonctions de dérivation (deux dans le cas de la Figure 14) sont évaluées selon le résultat de l'évaluation des gardes. Ainsi, sur notre exemple, si le diamètre est inférieur à 5, alors la première expression (« diamètre/2 ») sera évaluée, sinon (modélisée par l'entité *Others*) la seconde fonction (« diamètre/2+1 ») sera évaluée. Comme pour les expressions booléennes, les fonctions et les expressions arithmétiques ont été formellement modélisées dans le modèle PLIB (cf. section 2.7.3).

2.7.3 Méta-modélisation des expressions

Lors des échanges de données de composants, la connaissance procédurale (représentée en général par des expressions) accompagnant ces données doit être également échangée. L'approche développée dans PLIB pour modéliser la connaissance procédurale est basée sur la technique de la méta-programmation. Cette technique a été appliquée au langage EXPRESS dans [Ait-Ameur et al 95a], [Ait-Ameur et al 95b], [Pierra et al 96]. D'un point de vue structurel, à l'image des interpréteurs dans les langages de programmation, ces expressions sont modélisées par un graphe direct acyclique où les nœuds sont des opérateurs et les feuilles sont des

variables ou des littéraux. Cette représentation correspond à un arbre de syntaxe abstraite dans la théorie de compilation [Aho 77].

L'exemple de la Figure 15 montre une expression simple modélisée par un arbre et le format d'échange correspondant. Dans le modèle PLIB, chaque concept mathématique utilisé dans une expression est modélisé par une entité. Cette modélisation est une hiérarchie de classes réalisée par des relations de généralisation/spécialisation. Dans le modèle d'échange, chaque instance est associée à un identifiant (#2, …) et est décrite par les valeurs de ses attributs ou des références aux valeurs de ces attributs (voir Figure 15)

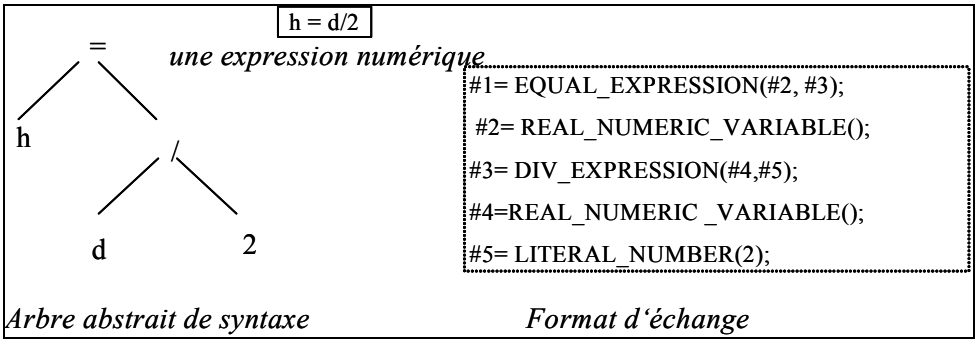


Figure 15 : modélisation de la connaissance procédurale par méta-programmation.

Sur l'exemple de la Figure 15, les variables *h* et *d* sont représentées par les instances #2 et #4 associées à une sémantique précise définie hors de cette représentation. En effet, ces variables peuvent être associées à la hauteur et au diamètre d'une vis pour représenter la fonction de dérivation de la Figure 14, par exemple.

La représentation des expressions par méta-programmation permet de représenter et d'échanger de la connaissance procédurale. Cette représentation est indispensable pour représenter les relations portant sur des ensembles très grands ou infinis. Dans la représentation des catalogues de composants, elle a pour objectif de représenter une infinité ou un grand nombre de valeurs de propriétés par une simple expression. En conséquence, pour PLIB, elle permet de :

- représenter la dérivation de propriétés qui dépendent d'autres propriétés en utilisant des fonctions,
- exprimer des contraintes d'intégrité sur les données représentées. Parmi ces contraintes, nous citons les filtres et les gardes définis précédemment.

3 Implémentation du modèle implicite de PLIB dans un SGBD

Afin de manipuler, de sélectionner, de modifier ou de référencer à partir d'un SGDT un composant, il est nécessaire d'implémenter les spécifications (décrites en EXPRESS) du modèle PLIB pour décrire un gestionnaire de composants. Nous discutons, dans cette section, des différentes options d'implémentation de ce gestionnaire.

Avant de détailler les différentes approches étudiées pour réaliser cette implémentation, nous tenons à rappeler quelques points importants du modèle à implémenter (le modèle PLIB). Deux niveaux distincts caractérisent ce modèle : le niveau du dictionnaire ou de l'ontologie et le niveau du contenu. Le niveau de l'ontologie vise à décrire d'une manière formelle tous les concepts (classes et propriétés) tout en les munissant d'un identifiant universel. Dans la partie "ontologie", les concepts ainsi décrits représentent principalement la connaissance structurelle et descriptive des données de composants (décrire la structure de classes en même temps que leurs propriétés). Le niveau du contenu vise, en premier lieu, à caractériser les instances des familles de composants. Une partie significative de ce schéma représente la connaissance procédurale sur les données de composants. De plus, la représentation implicite du contenu des bibliothèques de composants est fortement orientée contraintes. Les contraintes d'intégrité y sont formellement représentées par des prédicats (tel que les gardes exprimés sur les domaines, les filtres, …). La dérivation entre les propriétés y est également formellement représentée par des expressions et des tables.

3.1 Implémentation d'un schéma EXPRESS dans un SGBD

L'implémentation du modèle conceptuel de PLIB, c'est-à-dire l'implémentation des spécifications décrites en EXPRESS dans un système de gestions de base de données nécessite la représentation des différents concepts du langage EXPRESS dans ce système.

Les différents éléments du langage de modélisation de données EXPRESS peuvent être représentés ou simulés dans un SGBD. Afin de faciliter la compréhension du reste de ce chapitre, nous présentons d'une manière sommaire l'implémentation ou la traduction d'un schéma EXPRESS dans un modèle de SGBD cible (relationnel, relationnel-objet, …). Une entité EXPRESS est représentée par une relation et ses attributs par ses colonnes. L'héritage entre les entités d'EXPRESS peut être traduit directement par l'héritage des tables dans les SGBD relationnel-objet ou simulé dans les SGBD purement relationnels. Les types de données de base du langage EXPRESS tels que STRING, INTEGER, REAL, … sont représentés par des types du langage SQL. Les types plus complexes du langage EXPRESS tels que les types *select* et les types énumérés sont représentés par des relations. La correspondance entre les concepts du langage EXPRESS et ceux du modèle d'une base de données cible (relationnel, relationnel-objet) sera détaillée dans le chapitre 5 (mise en œuvre de l'approche).

La suite de cette section présente les différentes approches d'implémentation d'un système de gestion de composants fondé sur PLIB. Nous utilisons les représentations du modèle de données EXPRESS décrites précédemment. Trois approches peuvent être envisagées. Nous les qualifions respectivement de "Méta", d'"instance" et d'"hybride".

3.2 Première approche : Implémentation au niveau des descripteurs (niveau "méta")

La première approche consiste à considérer le modèle PLIB comme n'importe quel autre modèle EXPRESS puis, de le traduire directement dans le modèle de la base de données cible (relationnel ou relationnel-objet). Cette traduction utilise les règles de représentation, d'un modèle de données EXPRESS dans un SGBD relationnel ou relationnel-objet, ébauchées ci-dessus dans le pargraphe 3.1 et détaillées dans le chapitre 5. Dans le reste de ce pargraphe, nous donnons le modèle d'implémentation de cette approche et ainsi qu'un exemple pour illustrer cette implémentation.

3.2.1 Modèle d'implémentation de la première approche

Etant donné que les spécifications du modèle PLIB décrivent un méta-modèle, sa représentation systématique au niveau du SGBD cible produit exactement le même niveau de représentation (un méta-modèle). Ainsi, chaque entité du méta-modèle (ou chaque descripteur), qu'elle appartienne au dictionnaire (à l'ontologie) ou bien à son contenu, sera représentée par une table au niveau du SGBD. Une telle représentation privilégie le modèle de données EXPRESS sur le modèle de la base de données cible au sein même de cette dernière. La Figure 16 montre le modèle de l'implémentation engendrée par cette approche. La figure montre les deux parties du modèle PLIB (l'ontologie et son contenu) après leur implémentation dans une base de données cible. Dans cette approche, la spécificité de chacun des deux modèles (du niveau de l'ontologie et du niveau du contenu) n'a pas été considérée : la sémantique représentée par chaque modèle (où l'un des niveaux s'apparente à un "modèle" de l'autre) n'a pas été prise en compte. Par conséquent, le modèle de la base de données cible n'est finalement qu'une transcription à l'identique du modèle de données de PLIB (voir Figure 16).

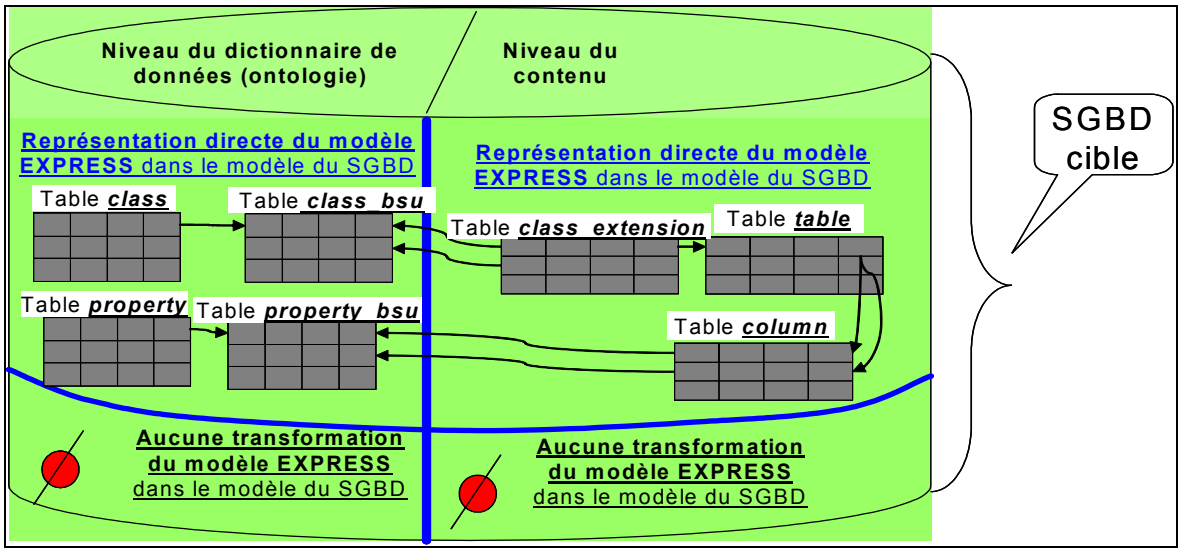


Figure 16 : modèle de l'implémentation de l'approche au niveau des descripteurs

Comme le montre la Figure 16, les capacités de la base de données cible ne sont que très peu utilisées dans le modèle de l'implémentation (implémentation des types des données nommés par des types de la base de données cible, par exemple). Ceci est vrai aussi bien pour le dictionnaire de données que pour son contenu. Pour la partie du

contenu, cette prédominance du modèle EXPRESS par rapport au modèle de la base de données cible se voit, par exemple, dans la représentation d'une table dans cette base de données. En effet, dans cette approche, une table est représentée telle qu'elle a été représentée dans le modèle PLIB (EXPRESS) et non pas telle qu'elle le doit dans la base de données cible (voir Figure 19) : elle est représentée par plusieurs relations (car elle l'est ainsi dans PLIB). Les deux exemples de la Figure 17 (niveau de l'ontologie) et la Figure 19 (niveau du contenu) montrent comment une telle représentation peut être réalisée.

3.2.2 Exemple d'implémentation de la première approche

La Figure 17 présente l'implémentation par un modèle relationnel du dictionnaire PLIB simplifié. Dans ce modèle, les familles de composants sont représentées par des classes (des instances de la relation *Class)* décrites par un ensemble de propriétés (des instances de la relation *Property*). Chaque classe est identifiée par un identifiant universel (un *BSU* représenté par l'attribut *Cl_id* de la relation *Class*). Chaque propriété est également identifiée par un identifiant universel (un *BSU* représenté par l'attribut *Prop_id* de la relation *Property*) et est associée à un type de données (*Data_type)* représentant son domaine de valeurs. Les liens entre les différentes entités EXPRESS sont représentés par des associations dans le schéma relationnel (traduites, selon la cardinalité, en clés étrangères ou en tables d'association au niveau de la base de données implémentant ce schéma). Ainsi, par exemple, les classes et leurs propriétés sont liées par la clé étrangère *Describe_by* de la relation *Property*. Les tables obtenues, par l'implémentation de ce schéma relationnel dans une base de données cible, représentent des éléments du méta-modèle PLIB. Nous les désignerons par le mot "méta-table". Elles correspondent aux méta-descripteurs du dictionnaire de données PLIB (exemple : l'entité *Class,* l'entité *Property,* …).

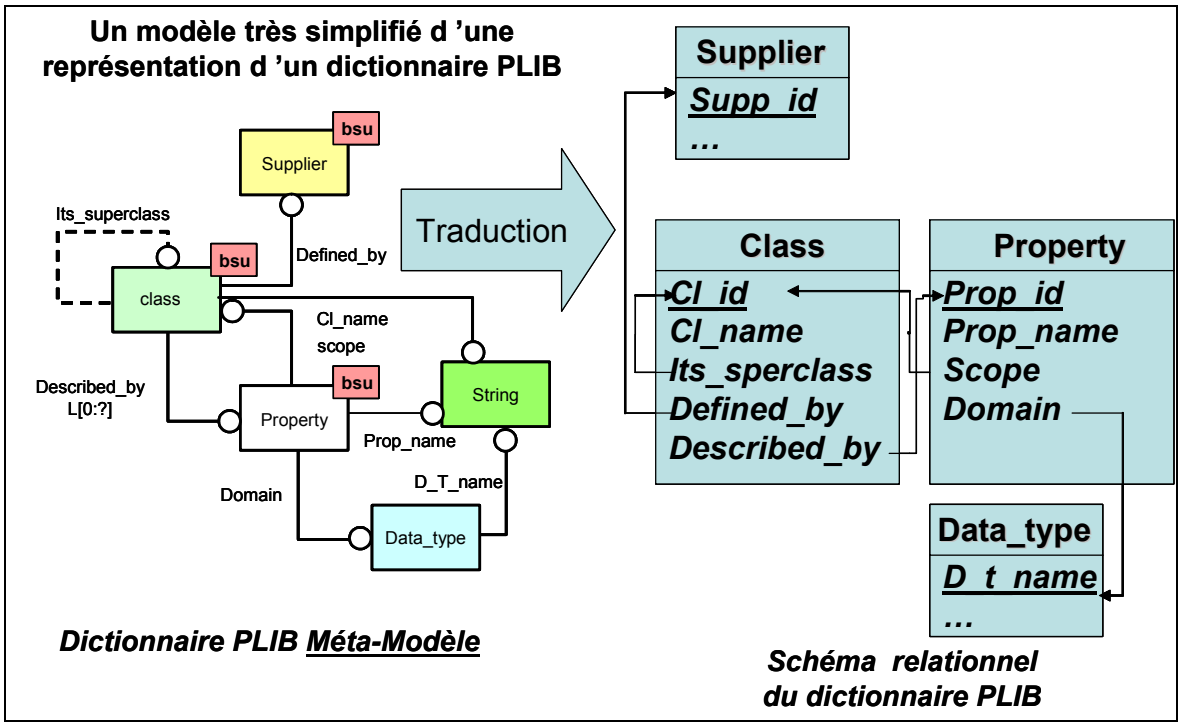


Figure 17 : Implémentation du dictionnaire au niveau méta par un schéma relationnel

La Figure 18 montre un exemple de description d'une famille de composants dans les méta-tables du modèle obtenu par l'implémentation du schéma relationnel de la Figure 17 dans une base de données cible. Ainsi, une classe (ou famille de composants) est représentée dans la base de données par plusieurs lignes ou instances de ces méta-tables. Par exemple, la classe *Vis_a_tete,* définie par le fournisseur (ou la source) d'information *LISI* (identifier par *F123L6*), est décrite dans les méta-tables *Class* et *Property*. Cette classe est identifiée par l'identifiant universel *Meca1546* et est une sous-classe de *Vis* (identifiée par *Meca1545*). Les propriétés de cette classe sont décrites dans la méta-table *Property*.

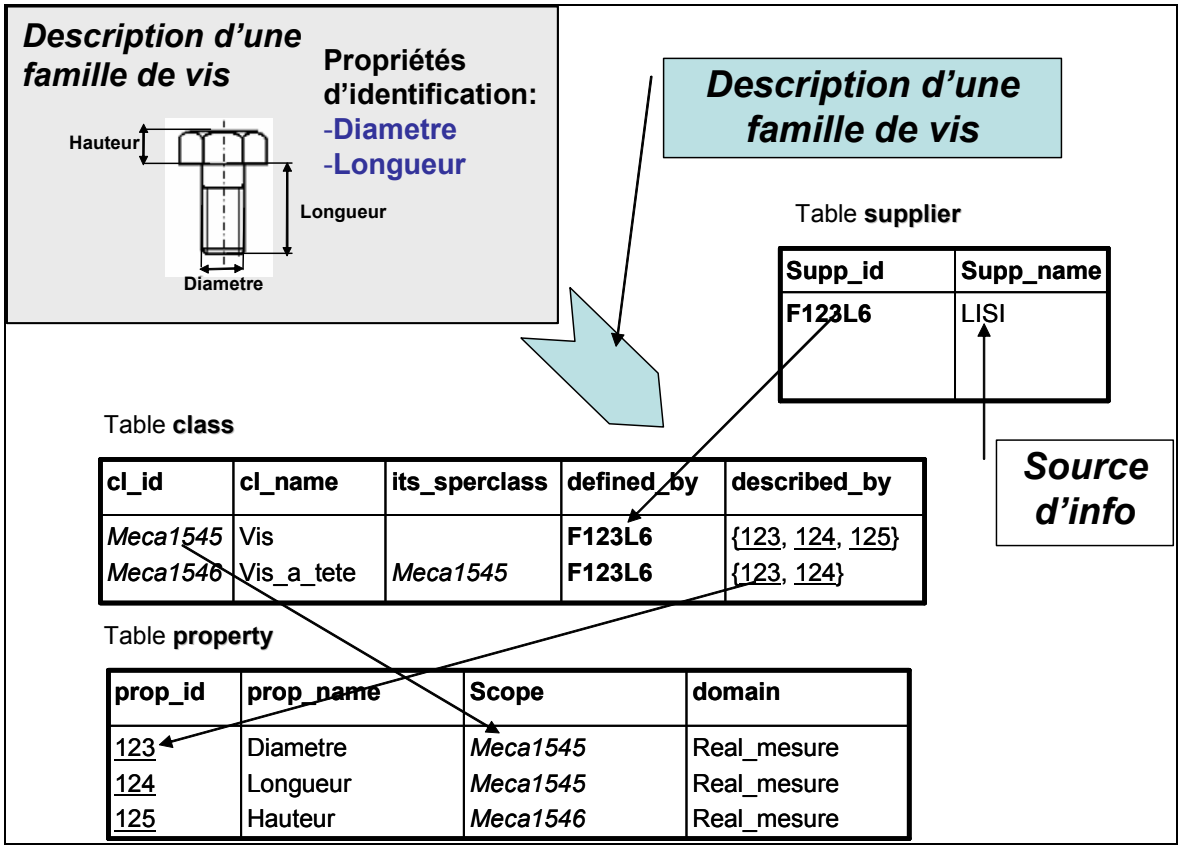


Figure 18 : Exemple d'un dictionnaire PLIB dans une base de données cible

La suite de ce paragraphe montre l'utilisation de la même approche pour implémenter le modèle de description des contenus. La Figure 19 montre l'implémentation d'un modèle simplifié du contenu d'une bibliothèque de composants, représenté suivant l'approche de description implicite du modèle PLIB.

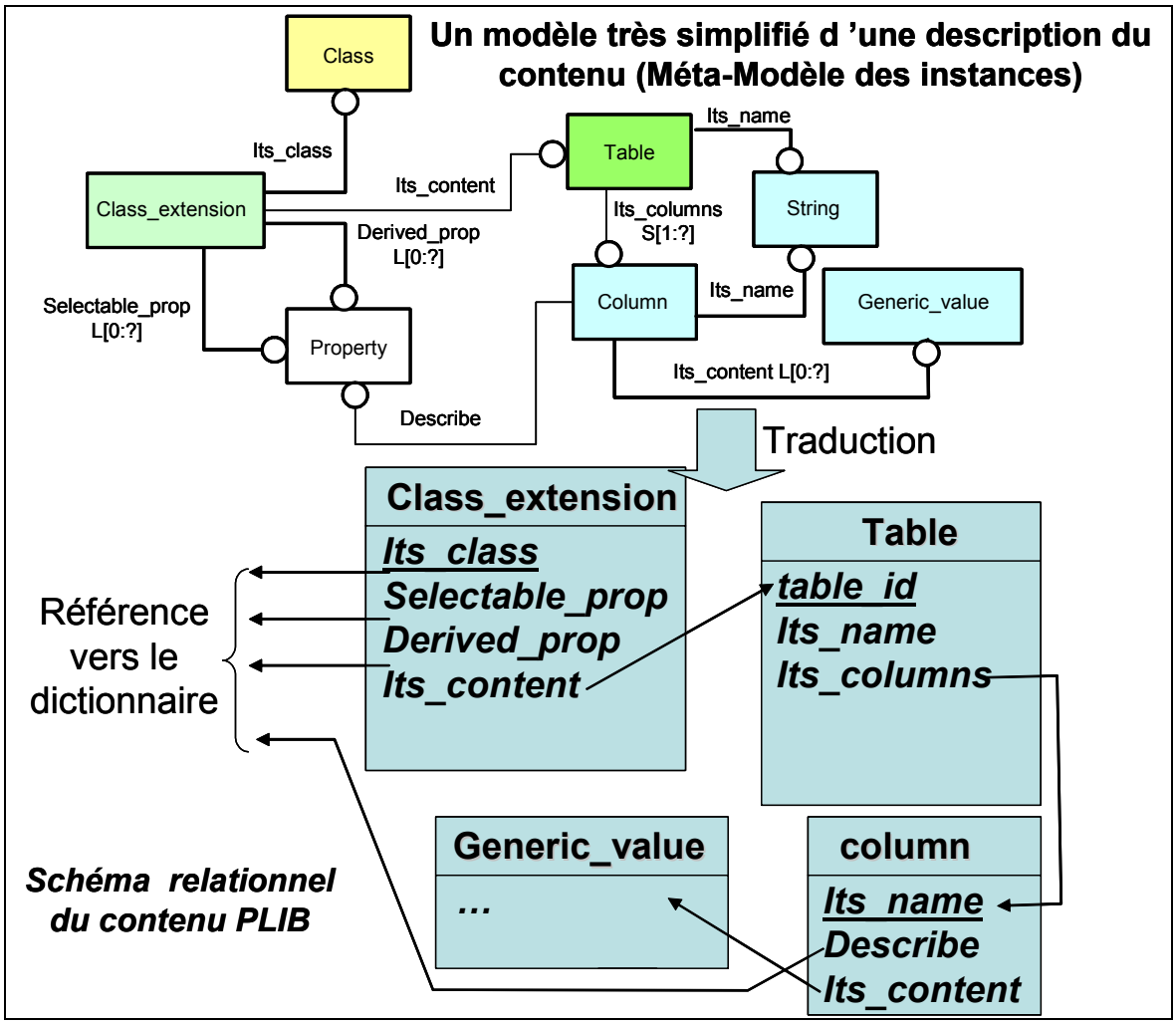


Figure 19 : Implémentation du contenu implicite au niveau méta par un schéma relationnel

Comme nous l'avons décrit en section 2.6, l'entité *Class_extension* décrit les instances des classes spécifiées au niveau du dictionnaire de données par des tables éclatées, des fonctions de dérivation et des règles de jointure. Dans le modèle implicite, ces tables sont représentées par des listes de listes. De la même façon que dans le schéma EXPRESS du méta-modèle de représentation des instances, dans l'approche "méta" (c'est-à-dire conforme au modèle EXPRESS), les tables contenant les extensions de classes sont représentées par des listes de listes dans le modèle de la base de donnée cible (relationnelle ou raltionnelle-objet). Dans notre exemple, l'implémentation est représentée par un schéma relationnel (voir Figure 19). Comme pour l'implémentation du dictionnaire, les liens entre les entités EXPRESS sont représentés par des associations entre les relations les représentant dans le schéma relationnel (clés étrangères dans la base de données cible).

La Figure 20 donne un exemple d'implémentation du schéma relationnel de la Figure 19 dans une base données cible particulière. Cet exemple montre la représentation d'un contenu d'une famille de vis dans cette base de données. Ce contenu est représenté par une description de l'extension de la famille des vis (ses propriétés sélectionnables, ses propriétés dérivées, la(les) table(s) définissant les domaines de propriétés, …).

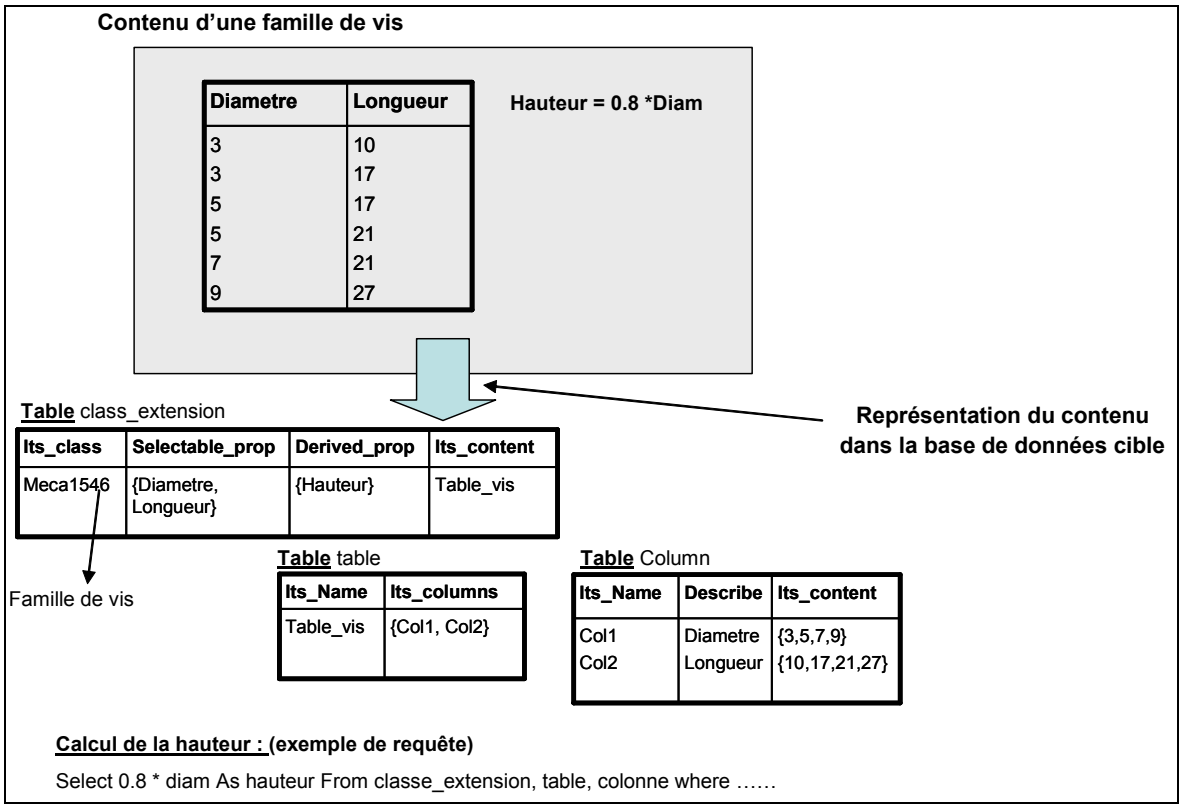


Figure 20 : Exemple de représentation d'un contenu PLIB dans une base de données cible

L'extension d'une classe et la table de son contenu (*Table_vis*) sont liées par une clé étrangère (l'attribut *Its_content*) de la méta-table *Class_extension* vers la méta-table *Table*. Chaque instance de la méta-table *table* (une table spécifique) référence ses colonnes par des clés étrangères vers la méta-table *Column*. Une colonne est décrite par un nom, le lien vers la propriété pour laquelle elle décrit le contenu ainsi que la liste des valeurs que peut prendre celle-ci. La propriété dérivée *Hauteur* de la vis peut être calculée, par exemple, par une requête de sélection en effectuant des opérations relationnelles sur les différentes méta-tables décrivant l'extension d'une classe (les méta-tables *Class_extension, Table, Column,* …). Ceci traduit l'expression en EXPRESS. Les règles de jointure permettant la reconstitution des instances à partir de ces méta-tables ne sont pas représentées sur cet exemple.

3.2.3 Faisabilité

Ainsi dans l'approche "méta", il n'y a pas de différence, d'un point de vue structurel, entre le dictionnaire de données et son contenu. Les deux modèles sont représentés au même niveau, par codage direct des schémas EXPRESS (les deux sont modélisés par des entités décrites par des attributs, etc.). En fait, les deux modèles sont implémentés comme n'importe quel autre schéma EXPRESS. En réalité, la différence majeure entre les deux modèles réside dans la sémantique représentée par chacun d'eux et donc les méthodes d'accès qui seront nécessaires. D'une part, le dictionnaire de données enregistre essentiellement la connaissance structurelle et descriptive tout en offrant un moyen d'identifier d'une manière universelle chaque concept. Les accès se font surtout par suivi de chemin (réalisation sur clé). D'autre part, le modèle du contenu représente l'ensemble des instances, et va nécessiter des accès de type algèbre relationnelle.

Ainsi, vu le mode de modélisation, l'exploitation de chaque niveau au sein de la base de données est fondamentalement différente selon que l'on s'intéresse au dictionnaire ou à son contenu. Le dictionnaire de données implémenté dans la base de données cible constitue des méta-données permettant de représenter les schémas conceptuels des bases de données des composants. Les opérations nécessaires sur ces méta-données sont la consultation et des classes et de leurs propriétés. De telles opérations sont assez simples à réaliser dans une base de données par des requêtes SQL. En fait, ces classes et propriétés étant représentées comme des instances du méta-modèle (par des tables), la consultation des informations les concernant peut être effectuée par des requêtes SQL simples. Ainsi, l'exploitation d'une implémentation du méta-modèle, représentant le dictionnaire de données, par traduction d'EXPRESS ne pose pas de problème particulier.

Lorsque nous traitons du modèle du contenu décrivant les instances (i.e. les composants), l'approche est différente. En effet, il ne suffit pas de traduire le schéma dans la base de données mais il faut également offrir, en plus, les moyens de son exploitation (c'est-à-dire de sélections). Ce schéma EXPRESS decrivant le contenu étant très complexe (représentation implicite des contenus de bibliothèques de composants), son implémentation directe dans une base de données cible (relationnelle ou relationnelle-objet) est encore plus complexe. Cette complexité est due, d'une part, à la multiplicité des entités EXPRESS (dans le modèle PLIB) permettant de représenter des concepts de base existant dans un SGBD (dans le modèle implicite de PLIB, une table relationnelle est représentée par les entités *table, column,* etc.), et, d'autre part, à la multiplication des relations nécessaires à la représentation des concepts simples du langage EXPRESS (l'union de types, par exemple, nécessite plusieurs relations pour le représenter). De plus, au niveau du contenu, le modèle implicite contient une multitude de contraintes de validation et d'expressions représentées formellement dans le modèle. La conception d'un programme qui vérifie automatiquement ces contraintes et calcule ces expressions n'est pas triviale et est très coûteuse.

Que ce soit pour le dictionnaire de données ou le contenu, la traduction automatique ne concerne que l'aspect structurel. C'est-à-dire, les entités et leurs attributs. En revanche, lorsque il s'agit de contraintes exprimées sur les modèles (directement dans le langage EXPRESS), et surtout celles qui sont validées en utilisant les fonctions et les procédures du langage EXPRESS, leur implémentation reste délicate et demande une implémentation de toutes les fonctions prédéfinies du langage EXPRESS et une génération de code (par exemple en C ou en PL/SQL).

3.2.4 Difficultés

Les difficultés que nous avons recensées sur la mise en œuvre de cette approche sont les suivantes :

1- Traduction des différentes contraintes du modèle PLIB, exprimées en langage EXPRESS, dans le modèle relationnel (les contraintes exprimées par EXPRESS pouvant être très complexes).

En ce qui concerne ce premier point et comme nous l'avons précisé dans le chapitre 1, le langage EXPRESS possède un grand pouvoir d'expression des contraintes. Ce pouvoir est renforcé par l'utilisation de fonctions et procédures comme dans un langage de programmation. Il offre également un certain nombre de fonctions prédéfinies (exemple: fonctions TYPEOF, USEDIN, …) permettant de renforcer plus son pouvoir d'expression. Compte tenu de la différence entre le langage EXPRESS et les langages de la famille SQL, une traduction manuelle des fonctions et des procédures, écrites dans le langage EXPRESS (notamment celles qui sont utilisées pour exprimer les contraintes), est nécessaire. C'est précisément la traduction de certaines fonctions prédéfinies (telles que TYPEOF, USEDIN, …) qui est difficile à réaliser. Pourtant la notion de procédures stockées a considérablement augmenté le pouvoir d'expression du langage SQL. Cette partie de SQL est basée sur les concepts de SQL augmentés par l'ajout de structures de contrôle conditionnelles et itératives. Elle permet la réalisation des calculs complexes sur les données de la base. Néanmoins, le pouvoir d'expression du langage SQL reste nettement inférieur à celui d'EXPRESS et ne permet pas la traduction de toutes les contraintes de ce langage. Parmi ces contraintes, on peut citer celles qui sont exprimées sur les instances (introduites par le mot clé **WHERE** du langage EXPRESS) ou sur des classes (contrainte globale introduite par le mot clé **RULE**) et qui utilisent des fonctions prédéfinies (telle que la fonction **TYPEOF**) ou des opérateurs (tel que l'opérateur **QUERY**) qui n'ont pas d'équivalent dans le langage SQL (i.e. PL/SQL).

2- Elaboration du programme permettant l'interrogation des instances au niveau du contenu lorsque celui-ci est représenté par une base de données relationnelle ou relationnelle-objet. Cette implémentation ne permet absolument pas de tirer profit, à ce niveau, du moteur relationnel existant : tout doit être re-programmé sur des listes de listes.

3- Complexité de l'évaluation des expressions décrites, par méta-programmation en EXPRESS, dans la base de données cible.

Pour ce qui concerne l'évaluation d'expressions, nous avons précisé que dans le modèle de données PLIB, les expressions sont méta-modélisées. Puisque dans cette première approche les méta-modèles sont directement représentés dans le modèle de la base de données cible, il va de soi qu'un moyen d'évaluation de ces expressions doit être mis en place. Cette évaluation n'est pas seulement complexe, à cause de la présence de plusieurs instances EXPRESS pour représenter une expression, mais également très coûteuse en temps car l'évaluation nécessite de nombreuses requêtes ou jointures de tables.

3.2.5 Avantages

Cette approche présente les avantages suivants :

1- Représentation de la définition ontologique des données de composants en parallèle avec les données de composants.

2- Possibilité de générer les données de composants à partir de la base de données dans un fichier d'échange (i.e. fichier physique EXPRESS) conforme au modèle PLIB.

3- Possibilité d'échanger les données de composants stockées dans la base de données avec d'autres systèmes hétérogènes.

4- Traduction du modèle de données PLIB à l'aide d'un outil systématique permettant de traduire tout modèle EXPRESS dans une base de données cible (relationnelle, relationnelle-objet, …). De ce fait, aucune connaissance particulière sur le modèle de données PLIB n'est requise pour passer directement à un schéma de la base de données cible.

3.2.6 Inconvénients

Les inconvénients de cette approche sont les suivants :

1- Complexité de la sélection des composants et de leur mise à jour. La représentation implicite du contenu n'exploite pas le moteur relationnel, ce qui nécessite de programmer le moteur de requêtes même pour les requêtes qui peuvent être très simples à la base.

2- Réduction du pouvoir d'interrogation de la base de données ainsi que des opérations algébriques supportées par un SGBD. Cela peut s'expliquer par la nécessité de traduire toutes les requêtes de base du langage SQL *« select * from table »* par des opérations de jointure, de produits cartésiens, etc. (car dans un modèle implicite, une table est représentée par des listes de listes).

3- Difficulté de représenter la connaissance procédurale telle qu'elle est représentée dans le modèle PLIB et surtout de l'exploiter par un langage comme SQL. En effet, si SQL permet d'interroger une base de données, il n'est pas adapté à la programmation d'outils capables d'interpréter cette connaissance lorsqu'elle est représentée dans un modèle relationnel éclaté entre un grand nombre de tables. Effectivement, il est plus simple de représenter une expression par une requête, que par un arbre de syntaxe éclaté sur plusieurs tables [Challal 03], [Ait-Ameur et al 03].

3.3 Deuxième approche : implémentation au niveau des instances

L'objectif de cette deuxième approche est d'éviter les problèmes liés à la complexité de la représentation des instances dans une base de données lorsqu'elles sont décrites par un méta-modèle (cf. section 3.2). En effet, d'une part la notion de table est bien définie dans les SGBD et toute redéfinition de cette notion par elle-même ne fait qu'ajouter un niveau de complexité supplémentaire. D'autre part, l'utilisation d'un méta modèle d'instances dans un SGBD nécessite la re-programmation de toutes les opérations algébriques déjà bien définies dans le langage SQL et bien intégrées dans tous les SGBDs.

Ainsi, la représentation au niveau "instances", proposée dans ce paragraphe, consiste à extraire d'abord le schéma de la base de données à partir du dictionnaire de données (sans le représenter), et ensuite à peupler ce schéma par transformation des instances représentées au niveau du contenu. Le traitement distingue deux niveaux, celui du dictionnaire et celui de son contenu. Le schéma de la base de données

représentant un catalogue de composants conforme au modèle PLIB est d'abord extrait du dictionnaire présenté comme étant une instanciation du modèle du dictionnaire PLIB [ISO 13584-42:1998]. La population de ce schéma est ensuite réalisée à partir de la représentation du contenu implicite [ISO 13584-24:2003], qui représente la population ou les instances de composants devant être représentées dans la base de données cible, d'où l'appellation « représentation au niveau d'instances » (les instances du modèle de dictionnaire (méta-modèle) correspondent au schéma de la base de données, les instances du modèle du contenu correspondent à la population de la base de données).

3.3.1 Modèle d'implémentation de la seconde approche

L'implémentation du modèle de données PLIB, en suivant cette approche, donne le modèle d'implémentation de la Figure 21. Le méta-modèle représentant le dictionnaire de données PLIB est d'abord instancié en EXPRESS (dans un fichier physique). Le modèle obtenu par cette instanciation (un dictionnaire particulier = une ontologie) est considéré comme un schéma conceptuel de données étendu. Il sera ensuite représenté dans la base de données cible. Compte tenu de la possibilité de traitement par machine des modèles EXPRESS, la génération du schéma de la base de données cible peut être réalisée d'une manière complètement automatique. Une fois le schéma de la base de données cible créé, le dictionnaire de données n'aura pas de rôle particulier à jouer dans cette base de données. L'implémentation de cette approche donne une grande importance au modèle de la base de données cible par rapport à celle du modèle EXPRESS. En effet, ce dernier est seulement utilisé pour générer le premier. Les exemples de la section 3.3.2 montrent un exemple de création d'un schéma de la base de données cible à partir d'un dictionnaire PLIB particulier, et le peuplement de cette base à partir d'un fichier physique représentant les instances (niveau du contenu) conforme à ce dictionnaire.

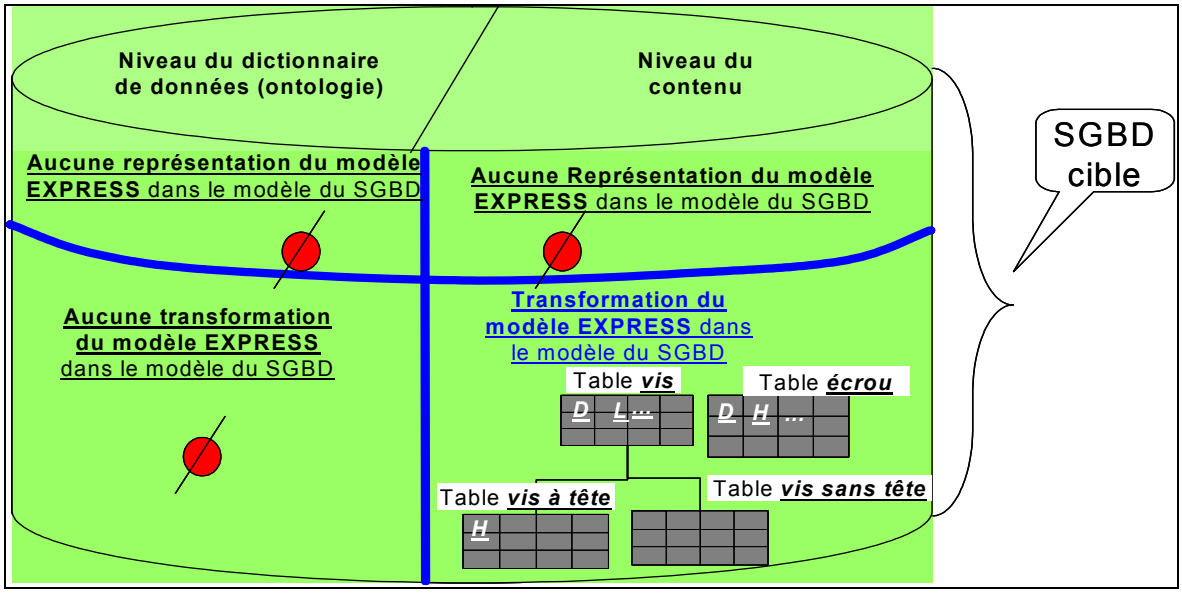


Figure 21 : modèle de l'implémentation de la seconde approche (niveau des instances)

3.3.2 Exemple d'implémentation de la seconde approche

Nous donnons dans cette section un exemple d'implémentation d'une bibliothèque de composants mécanique dans une base de données cible, en suivant l'approche décrite ci-dessus. Cet exemple est composé de deux parties : la partie traitant du schéma de la base de données (qui correspond au niveau du dictionnaire) et la partie traitant de la population de la base de données (qui correspond au niveau du contenu). La Figure 22 illustre la création du schéma de la base de données cible à partir d'un dictionnaire de données PLIB particulier. Ce dictionnaire est représenté dans un fichier physique. Le schéma relationnel créé à partir de ce dictionnaire est fondé sur un modèle relationnel-objet (héritage de tables). La classe *vis,* instance de la méta-classe *class,* est représentée par la table *vis*. Les propriétés de cette classe sont les colonnes de cette table. L'héritage de classes (représenté par l'attribut *its_superclass* dans le méta-modèle de classes) est représenté par un héritage de tables. Les noms des tables et des colonnes sont extraits à partir des noms des classes et des propriétés représentées dans le fichier physique. Une correspondance entre les types de données modélisés dans le dictionnaire de données PLIB et la base de données cible a été représentée (*Real_type* correspond au type *real* de la base données cible). Cette correspondance est représentée d'une manière informelle dans le programme de traduction.

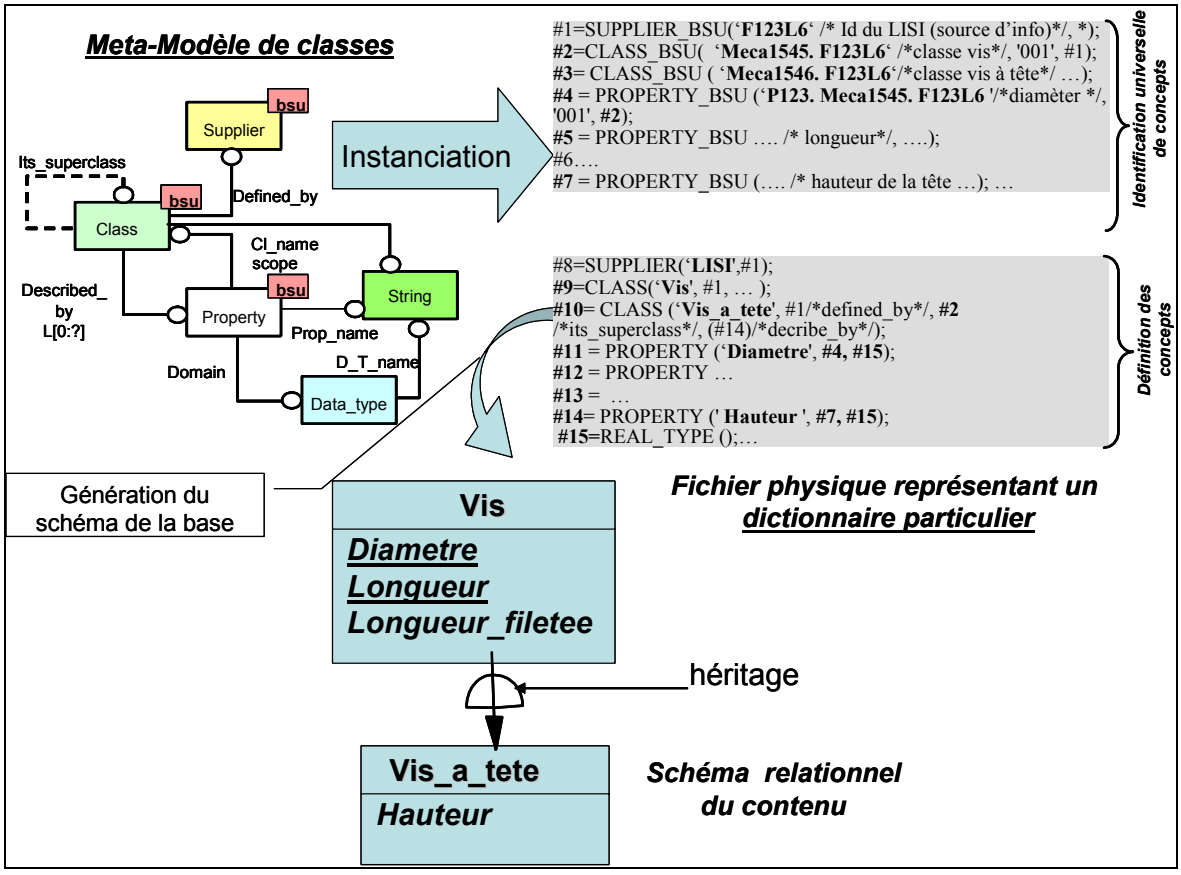


Figure 22 : Création du schéma de la base de données à partir d'une instanciation du dictionnaire PLIB

La Figure 23 montre le processus de représentation des composants (instances du niveau contenu) dans la base de données dont le schéma a été généré en suivant l'approche décrite ci-dessus en Figure 22. Ce processus nécessite l'explicitation (le dépliage) des données représentées d'une manière implicite. Cette phase n'est pas

représentée de façon complète dans cet exemple et fera l'objet d'une explication détaillée dans le chapitre 3. On suppose que dans la Figure 23, les données sont fournies par une simple liste de listes.

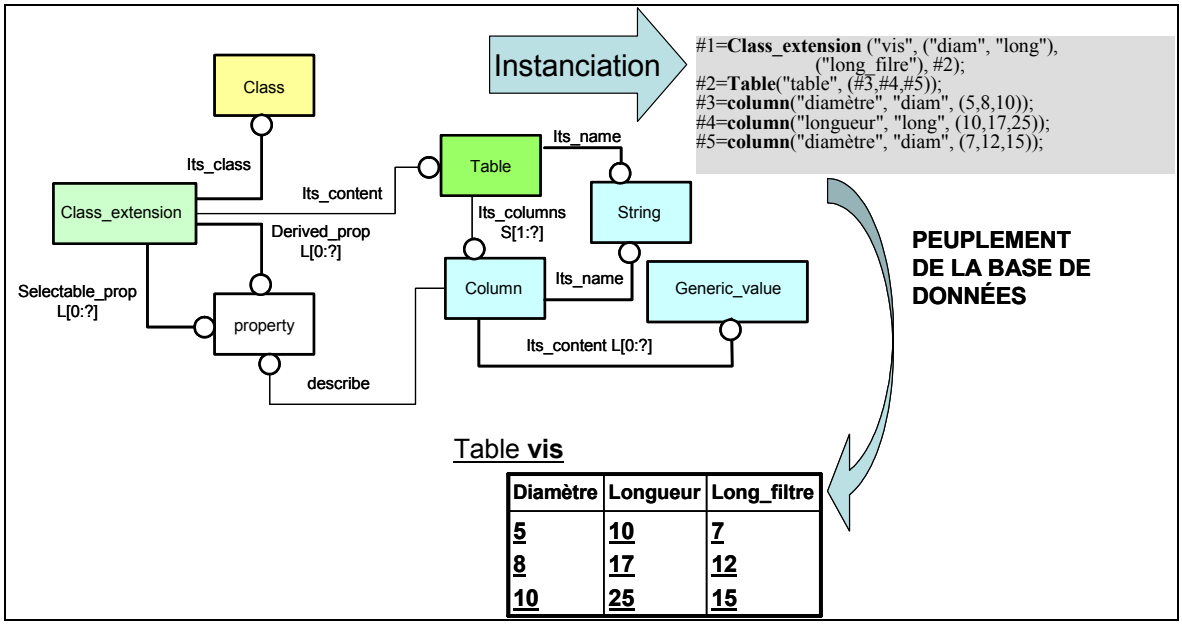


Figure 23 : Peuplement de la base de données à partir d'une instanciation du modèle de contenu.

La Figure 23 comprend trois parties. La première, située à gauche de la figure, représente une partie du méta-schéma générique du niveau du contenu de PLIB pour la représentation et l'échange des instances. L'instanciation de ce méta-schéma constitue la population d'une bibliothèque de composants décrite par une ontologie au niveau du dictionnaire de données PLIB. La deuxième partie qui se situe à droite de la figure représente cette population sous forme d'un fichier physique conformément à la spécification du langage EXPRESS défini par la norme STEP [ISO 10303-21:1994] (voir annexe). La dernière partie (en bas de la figure) représente la base de données dont le schéma est créé à partir de l'ontologie (voir Figure 22). La population (instances de composants) est importée à partir du contenu (représenté par le fichier physique). Le contenu importé subit un traitement permettant de convertir les données représentées d'une manière implicite dans un format explicite relationnel (dans la base de données). Ce traitement de conversion est décrit en détail dans le chapitre 3.

3.3.3 Faisabilité

L'approche consistant à représenter les données au niveau "instance" a ses limites et ne peut représenter qu'une partie des informations supportées par la représentation implicite. Ces limites sont atteintes lorsque la reconstitution des tables de définitions des familles de composants, représentées d'une manière implicite, conduit à une explosion combinatoire du nombre d'instances générées. De nombreux domaines d'application CAO possèdent des familles de composants ayant des propriétés dont les valeurs dépendent directement des contextes d'insertion. Ces derniers peuvent avoir un domaine de valeurs très grand (voire infini), et engendrer un très grand nombre d'instances.

Une solution possible pour l'implémentation suivant cette approche est de se limiter à ne représenter que les propriétés caractéristiques qui ne dépendent pas des contextes d'insertion. Bien qu'elles fassent partie intégrante des propriétés décrivant les familles de composants, les propriétés dépendantes du contexte ainsi que les paramètres de contexte ne sont pas représentés directement dans la table relationnelle. Ces propriétés peuvent être représentées de différentes manières, comme par exemple les procédures stockées, les vues, etc. Ceci sera détaillé dans le chapitre 3.

3.3.4 Difficultés

Les difficultés de la mise en œuvre de cette approche apparaissent à deux niveaux : choix de schéma de la base de données et extraction du contenu, compte tenu de la complexité de la représentation implicite des données de composants présente au niveau du contenu.

La création du schéma de la base de données cible pose le premier problème. En effet, le "schéma" est d'abord représenté au niveau de l'ontologie par la description des familles de composants et leurs structures (il s'agit d'une instanciation du dictionnaire de données PLIB). Dans cette définition ontologique, les familles de composants sont décrites dans le contexte le plus général de leur utilisation. Chaque famille de composants est munie d'un grand nombre de propriétés. En général, le contenu n'utilise qu'une partie de ces propriétés. Si un autre contenu correspondant à la même famille de composants est chargé ultérieurement, il se peut que les propriétés retenues ne soient pas les mêmes. Lors de l'importation de nouvelles données vers la base de données cible à partir des fichiers physiques échangés, une incohérence entre les données stockées dans la base de données cible et celles importées peut apparaître. Cela est dû à la différence entre les propriétés représentées du modèle importé et celles représentées dans la base de données cible. Ce problème peut être résolu en ajoutant, autant que de besoins, de nouvelles colonnes aux tables existantes initialisées à **NULL** pour les instances existantes. Remarquons que cela suppose que le dictionnaire reste accessible.

Au niveau du contenu, la représentation de la connaissance procédurale (contraintes et dérivations) présente également des difficultés. La connaissance procédurale étant méta-programmée dans le modèle PLIB, son implémentation nécessite, par conséquent, la définition d'outils permettant la conversion de ces méta-données en programmes compilables dans le système cible (des fonctions, procédures, des triggers, des vues, … dans le SGBD). Ainsi compilée, cette connaissance ne peut être en aucun cas échangée avec d'autres systèmes et sert exclusivement à une utilisation locale dans le système pour lequel a été générée.

3.3.5 Avantages

Les avantages qu'offre cette approche sont les suivants :

- la simplicité du schéma de description de données dans le modèle de la base de données cible (une table par famille),

- une meilleure performance dans la gestion de la base de données cible. Elle est due à l'utilisation directe des opérations algébriques (jointure, produit cartésien, ….) effectuées sur les relations par le moteur relationnel. En fait, l'utilisateur bénéficie pleinement des fonctionnalités et des performances des SGBD cibles.

3.3.6 Inconvénients

Cette approche présente néanmoins plusieurs inconvénients :

- perte du pouvoir d'expression et de modélisation des données de composants offert par le modèle PLIB. Cela est dû en partie à l'absence de représentation des paramètres de contexte et des propriétés dépendantes du contexte ;

- absence de la représentation de la description ontologique des familles de composants définie par le modèle de données PLIB. Une perte d'information très importante en résulte : ainsi tout échange de données avec une autre source d'information est compromis (on ne peut pas lire des valeurs de propriété qui figurent dans l'ontologie mais que l'on n'a pas retenues au niveau du schéma);

- faible pouvoir de sélection et de choix de composants, qui constitue une partie importante de l'exploitation du modèle de données. Les composants seront sélectionnés en se basant seulement sur les propriétés caractéristiques ;

- absence de représentation formelle de la connaissance procédurale dans le modèle relationnel.

3.4 Troisième approche : l'approche hybride

Afin de pallier les problèmes dus aux deux approches précédentes, nous avons proposé d'associer les deux approches [El Hadj Mimoune et al 00] en définissant une troisième approche : l'approche hybride. Cette approche consiste à représenter l'ontologie (dictionnaire de données) au niveau du méta-modèle en utilisant la première approche, et le contenu décrivant les composants au niveau des instances en utilisant la seconde approche. Ainsi, la base de données cible contiendra aussi bien les descriptions ontologiques (dictionnaire de données) que leurs données des composants. Par conséquent, le modèle conceptuel de la base de données sera représenté au même niveau que les données qu'il vise à représenter.

L'objectif de cette approche est de préserver les informations véhiculées par le modèle PLIB lors de son implémentation dans une base de données cible (relationnelle, relationnelle-objet, objet, …). En procédant de la sorte, nous souhaitons mettre en œuvre un gestionnaire de données de composants basé sur un SGBD qui bénéficie des performances de ce dernier et qui garde le pouvoir de modélisation de PLIB grâce à la représentation simultanée du modèle conceptuel et des données dans la même base de données.

3.4.1 Modèle de l'implémentation de la troisième approche

Dans la mesure où cette approche est une association entre les deux autres approches (l'implémentation au niveau "méta" et l'implémentation au niveau des instances), son modèle d'implémentation sera une combinaison des deux modèles d'implémentation des approches associées. Le niveau du dictionnaire est implémenté au niveau des descripteurs comme il l'est dans la première approche. Le niveau du contenu est implémenté au niveau des instances comme dans la deuxième approche. La Figure 24 montre le modèle d'implémentation de cette approche dans une base de données cible.

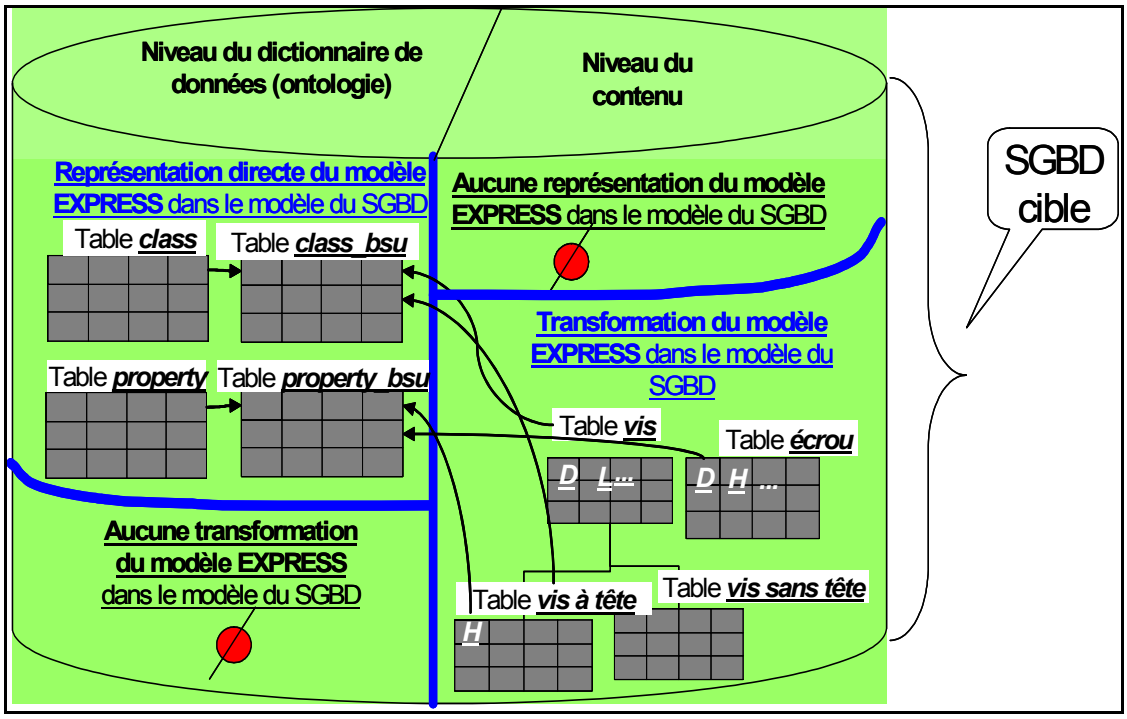


Figure 24 : Modèle de l'implémentation de l'approche hybride

Au niveau du dictionnaire de données le schéma de la base de données est presque l'identique au modèle EXPRESS. Par contre, au niveau du contenu, le modèle de la base de données cible est très différent. Comme le montre la Figure 24, les classes et les propriétés du niveau du contenu référencent les descriptions ontologiques du niveau du dictionnaire.

L'exemple d'implémentation du dictionnaire de données PLIB dans la première approche (cf. paragraphe 3.2.2, Figure 17, Figure 18) illustre bien l'implémentation du dictionnaire dans l'approche hybride. L'implémentation du niveau du contenu de cette approche peut être illustrée par l'exemple d'implémentation de l'approche "instances" (paragraphe 3.3.2). Le schéma de la base de données cible (représentant le niveau du contenu PLIB) sera créé à partir du modèle conceptuel représenté au niveau du dictionnaire au sein de la même base de données, puis les instances converties en n-uplets explicites pour remplir les tables.

3.4.2 Faisabilité

Cette approche se divise en deux parties : la partie concernant la représentation du niveau du dictionnaire de données PLIB et la partie concernant la représentation du niveau du contenu.

Puisque cette approche est une association entre les deux premières approches, les difficultés rencontrées proviennent des deux approches dont elle est issue. Les difficultés de la partie du niveau du dictionnaire de données sont celles qui le concernent dans la première approche. Les difficultés de la partie du niveau du contenu sont celle de la seconde approche.

Pour pouvoir implémenter le modèle du contenu, trois étapes sont nécessaires :

1- générer un schéma relationnel de représentation des composants (instances du niveau du contenu) sous forme explicite (plus simple),

2- convertir les données représentées dans le modèle implicite dans cette nouvelle représentation,

3- conserver un lien entre la nouvelle représentation et certains éléments de la représentation implicite. Ce lien permet de traiter le cas où la représentation implicite est indispensable pour représenter toutes les informations souhaitables (surtout le comportement de composants mécaniques dans leurs contextes d'insertion).

En revanche, la représentation de la connaissance procédurale est beaucoup plus difficile et nécessite une étude plus approfondie. Nous nous sommes contentés de représenter certains aspects de cette connaissance, plus particulièrement en ce qui concerne la première partie : le dictionnaire de données. Nous pouvons citer, par exemple, la représentation des attributs dérivés du langage EXPRESS ainsi que d'autres contraintes simples (unicité par exemple). Une intervention du programmeur pour coder certaines fonctions (de dérivations ou utilisées dans des contraintes) est nécessaire.

3.4.3 Difficultés

Les difficultés posées par cette approche sont de deux natures différentes :

1- difficultés posées par l'implémentation du dictionnaire. Elles concernent essentiellement la représentation des contraintes du langage EXPRESS au niveau de la base de données (cf. section 3.2.4), et la réécriture des fonctions EXPRESS prédéfinies.

2- difficultés liées à l'implémentation du contenu. Elles concernent :

- la complexité du modèle implicite qui se traduit par la complexité de son implémentation,

- la difficulté de représenter toutes les informations véhiculées par le modèle implicite (connaissance procédurale, …),

3.4.4 Avantages

Les avantages de cette approche sont multiples :

1. représentation uniforme de la définition ontologique et du contenu du dictionnaire dans le même SGBD cible,

2. possibilité d'interroger la base de données, soit au niveau ontologique, soit au niveau du contenu,

3. utilisation de la performance des SGBD pour gérer les données de composants sans perdre le pouvoir de modélisation de PLIB (représenter les données et leurs définitions ontologiques dans la même base de données),

4. facilité d'échange et de partage de données avec les systèmes qui implémentent la même ontologie quelle que soit la structure des schémas locaux dans les différents SGBD,

5. création dynamique du schéma au niveau du contenu de la base de données lors de l'importation d'un contenu d'une bibliothèque de composants. Il est créé à partir de son modèle ontologique représenté physiquement dans la même base de données,

6. simplification des modifications de structure de la base de données cible, dans le cas où les données importées sont décrites par des propriétés absentes du schéma du contenu dans la base de données cible,

7. et surtout ce qui était un des objectifs de notre thèse : facilité d'intégration des données de composants dans un SGDT.

3.4.5 Inconvénients

Les inconvénients liés à cette approche sont les suivants :

1- insuffisance des langages des bases de données classiques pour l'interrogation de la base de données au niveau ontologique, afin d'exploiter les définitions du dictionnaire ;

2- impossibilité d'intégrer certaines informations contenues dans une représentation implicite, lorsque celles-ci s'expriment sous forme de fonctions ayant des domaines contenus.

3.5 Conclusion

Nous avons présenté dans ce chapitre les différentes approches envisageables pour implémenter le modèle de données de PLIB dans une base de données cible (relationnelle, relationnelle-objet, …). Nous avons également étudié les avantages et les inconvénients de chaque approche. La première approche qui représente à l'identique le modèle PLIB au sein de la base de données présente un inconvénient majeur : la nécessité de redévelopper complètement un moteur relationnel fonctionnant sur des listes de listes pour pouvoir ensuite sélectionner des composants. La seconde approche se contente de représenter les instances. Elles sont représentées dans des tables relationnelles pour résoudre le problème d'exploitation d'instances posé par la première approche. Elle néglige quant à elle un aspect fondamental du modèle de donnée PLIB. Il s'agit de la définition ontologique qui permet de définir et d'identifier universellement chaque concept utilisé (classes, propriétés, …). L'ontologie, ainsi définie, permettant de faciliter l'échange et le partage de données entre systèmes hétérogènes, sa représentation effective au sein de la base de données apparaît indispensable. Enfin, la troisième approche que nous avons élaborée est l'approche hybride. Elle consiste à représenter le dictionnaire (l'ontologie) conformément à la première approche, et le contenu conformément à la deuxième, et représenter les liens entre ces deux niveaux. Cette approche permet de représenter aussi bien le niveau du dictionnaire que le niveau du contenu de la manière la plus efficace. C'est l'approche hybride que nous retiendrons dans la suite de ce travail et dont nous réaliserons une implémentation effective dans une base de données cible. Cette proposition d'approche hybride a été soumise aux groupes de normalisation en charge du modèle PLIB et a entraîné une évolution de ce dernier. A l'approche implicite de représentation des contenus, seule existante lors du démarrage de nos travaux, a ainsi été ajoutée une deuxième approche de représentation explicite des instances (norme [ISO 13584-25:2004]). Dans cette représentation explicite, chaque instance est représentée comme une liste de couples (propriété, valeur), très proche en fait des n-uplets du modèle relationnel. La représentation explicite sera détaillée dans le chapitre 4. Par ailleurs, les différentes dimensions de l'implémentation du modèle implicite selon l'approche hybride seront détaillées dans le chapitre 3.

Chapitre 3

Implémentation du modèle implicite en suivant l'approche hybride : conversion des instances de ce modèle en n-uplets

Résumé.
Dans ce chapitre, nous présentons l'implémentation du modèle de données PLIB en suivant l'approche hybride (approche retenue pour une implémentation effective). Nous présentons également l'architecture de la base de données cible résultant de cette implémentation. Celle-ci est basée sur trois niveaux logiques : le niveau de la métabase, le niveau de l'ontologie et le niveau des instances (le contenu). Le premier niveau est directement lié au fonctionnement du SGBD et les deux autres résultent de l'implémentation du modèle PLIB. Ainsi, la représentation d'une bibliothèque de composants dans la base de données cible dépasse la simple représentation de données de composants. Elle s'étend à la représentation de la définition ontologique de ces données au sein de la même base de données. La représentation implicite des composants étant complexe, une conversion de données représentées implicitement vers une représentation explicite (plus simple) est nécessaire. Nous détaillons dans ce chapitre les différentes étapes par lesquelles passe cette conversion. L'évaluation des expressions (très utilisées dans la représentation implicite) se situe au centre de ce processus de conversion. Nous détaillons également les différentes méthodes permettant de réaliser cette évaluation. Enfin, cette implémentation débouche sur un gestionnaire de données de composants dont le modèle est présenté dans ce chapitre. Nous discutons les différentes possibilités d'interroger les données représentées dans ce gestionnaire.

1 Introduction

Dans le chapitre précédent, nous avons étudié la représentation implicite de données de composants dans le modèle de données PLIB. Nous y avons présenté les différents concepts et mécanismes permettant cette représentation. Les différentes approches d'implémentation des données de composants représentés d'une manière implicite dans une base de données cible ont été également étudiées. Trois approches ont été considérées. Comme nous l'avons précisé dans le chapitre 2, l'approche hybride est celle qui présente le plus d'avantages et permet d'éviter les inconvénients des deux autres approches. Cette approche consiste à représenter l'ontologie (dictionnaire de

données) au niveau du méta-modèle (représentation directe des entités EXPRESS), et le contenu décrivant les composants au niveau des instances (par des n-uplets). Elle a donc été retenue pour une implémentation effective. La complexité de la représentation implicite constitue un des problèmes qui entravent la mise en œuvre de l'approche hybride (cf. section 3.4.3 du chapitre 2). Afin de pallier cette difficulté, nous avons proposé de procéder à une conversion des données implicites en données explicites. Nous détaillons dans ce chapitre cette conversion des données. Par ailleurs, la connaissance procédurale qui occupe une place importante dans la représentation implicite des données de composants doit être également prise en compte lors de ce processus de conversion. L'évaluation des expressions (i.e. connaissance procédurale) constitue donc une étape importante de ce processus. Nous présentons ici différentes méthodes permettant d'évaluer les expressions : évaluation par programme générique, évaluation par attributs dérivés du langage EXPRESS (ou auto-évaluation) et évaluation par programmes générés. Enfin, l'implémentation des données converties dans une base de données cible permet la mise en place d'un gestionnaire de données de composants. Le modèle global de ce gestionnaire fondé sur un SGBD est détaillé dans ce chapitre. Enfin, nous présentons les différents niveaux d'interrogation des données au sein de ce gestionnaire.

2 Schéma global de la base de données

L'approche que nous avons adoptée, pour implémenter le modèle PLIB dans un modèle de base de données cible, nous conduit à une nouvelle architecture de base de données. Cette architecture consiste à représenter dans un modèle de données unique aussi bien les données que leurs descriptions conceptuelles à travers le modèle d'ontologie défini par le dictionnaire PLIB (Figure 25).

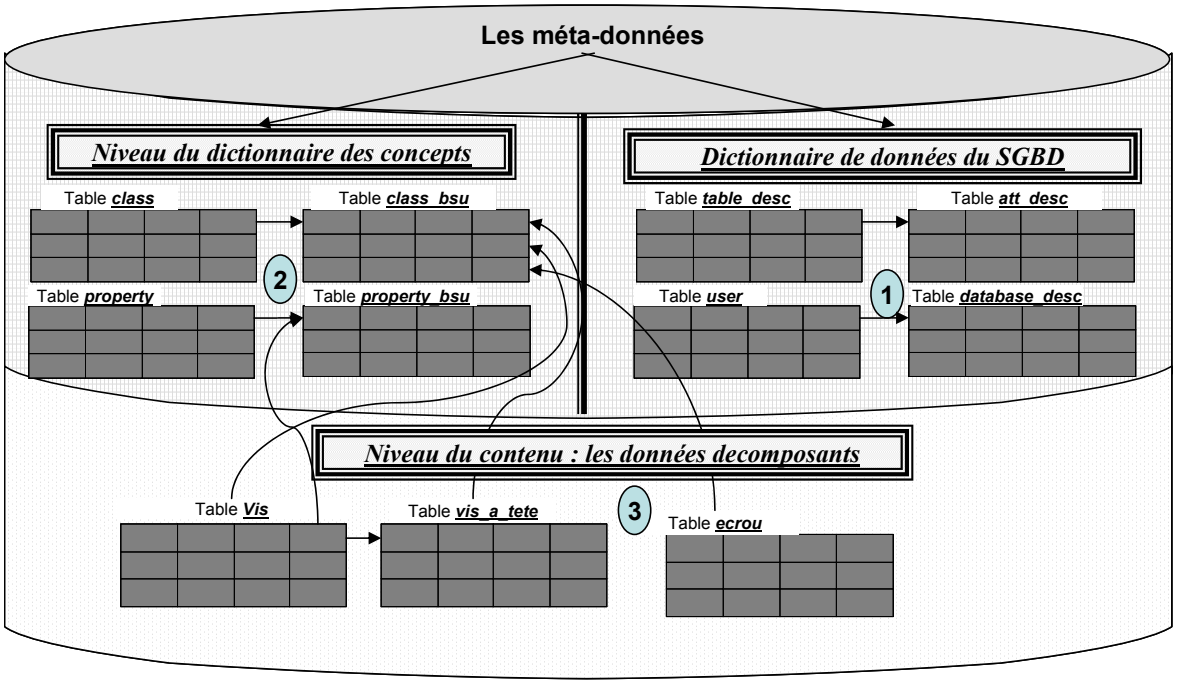


Figure 25 : L'architecture de la base de données dans une implémentation de l'approche hybride

Comme le montre la Figure 25, le modèle général de données est constitué de trois parties logiques.

La première partie constitue le cœur du SGBD et est appelée la métabase [Gardarin 01]. Elle contient les méta-données (c'est-à-dire des informations sur les données) de la base de données, et comprend, entre autres, les informations concernant les tables (leurs noms, leurs attributs, leurs contraintes), les types, les utilisateurs (leurs noms, leurs droits, …), etc. Cette première partie est constituée d'un ensemble de tables qui sont mises à jour automatiquement à chaque modification du schéma de la base de données, c'est-à-dire, à chaque exécution d'une requête de définition de données (utilisation du Langage de Définition de Données (LDD) de SQL). Le schéma de la métabase est créé par le système de gestion de la base de données cible lors de son installation et reste inchangé pendant toute la durée de son utilisation.

La deuxième partie contient la définition ontologique de l'univers du discours représenté dans la base de données. Les concepts (classes et attributs) utilisés dans la base de données y sont clairement identifiés et définis (schéma conceptuel de données). Dans cette partie, la définition ontologique des données sera représentée de la même façon que les données elles-mêmes (lignes de tables de la base de données). Nous détaillons davantage dans le chapitre 4 des différents avantages qu'offre cette nouvelle architecture. A la différence de la partie 1 (voir Figure 25) de l'architecture de la base de données qui conserve le schéma de la base de données tel qu'il a été **implémenté**, la partie 2 permet la représentation du schéma de la base de données tel qu'il a été **conçu**. En d'autres termes, elle contient l'ontologie et le schéma conceptuel de la base de données. Cette représentation explicite du modèle conceptuel permet d'accéder à des connaissances définies dans ce modèle conceptuel qui sont souvent perdues dans le modèle logique de la base de données. Les tables illustrées dans la Figure 25 représentent l'implémentation du méta-modèle du dictionnaire de données PLIB (i.e. le modèle de l'ontologie).

La troisième et dernière partie du modèle général de données contient les données relatives aux composants (les instances de l'univers de discours représenté). Elle représente l'implémentation du niveau du contenu du modèle PLIB (des composants) dans la base de données cible. Cette implémentation est réalisée au niveau des "instances" conformément à la troisième approche d'implémentation (l'approche hybride). Le schéma de la base de données constitue, pour cette partie, une traduction du modèle conceptuel de données des instances, telles qu'elles sont représentées dans le modèle PLIB, en modèle de la base de données cible (relationnel ou relationnel objet).

Pour résumer, nous proposons une implémentation sous forme d'une nouvelle architecture de base de données dite "base de données à base ontologique" et comprenant trois niveaux : la métabase, le modèle conceptuel (l'ontologie) et le contenu. Nous nous intéressons particulièrement dans la suite aux bases de données à ontologie où le niveau de l'ontologie est représenté par le modèle de dictionnaire PLIB.

3 Approche détaillée de l'implémentation du modèle implicite

Comme nous l'avons mentionné dans le paragraphe 3.4.5 du chapitre 2, l'implémentation, en suivant une approche hybride, des éléments représentant le contenu dans le modèle implicite conduit à une perte d'informations. De plus, l'explicitation des instances représentées d'une manière implicite peut générer un nombre très important voire infini d'instances. Donc, une restriction du nombre de ces instances, représentées dans la base de données cible, s'impose. Cette restitution pourra donc conduire à une perte des informations représentées.

Dans la suite de ce paragraphe, nous donnons les différentes étapes suivies pour l'implémentation du modèle implicite de PLIB dans un SGBD cible (relationnel ou relationnel-objet). Nous détaillons les différentes approches proposées afin de conserver un maximum d'informations, représentées dans le modèle implicite, après son implémentation dans cette base de données cible.

3.1 Différentes étapes menant à l'implémentation du modèle implicite

La représentation de bibliothèques de composants, conformes à une description ontologique correspondant au modèle implicite de PLIB dans une base de données cible (ici une base de données relationnelle) passe par les étapes suivantes :

1- création du schéma relationnel (i.e. le modèle logique) représentant le niveau du dictionnaire de données (le méta-modèle de l'ontologie) dans la base de données cible (partie 2 dans l'architecture de la base de données),

2- peuplement de ce schéma par les descriptions ontologiques des catalogues de composants que l'on souhaite représenter dans la base de données,

3- création du schéma relationnel (i.e. schéma logique) représentant le modèle de contenu (le méta-modèle des instances de composants) de PLIB,

4- peuplement de ce schéma par les instances représentant les composants proprement dits (le contenu d'une bibliothèque de composants).

Nous détaillons dans les pargraphes suivants chacune de ces étapes.

3.1.1 Schéma relationnel représentant le niveau du dictionnaire

L'étape 1 consiste à traduire le méta-modèle représentant le dictionnaire de données dans un modèle de la base de données cible (modèle relationnel, par exemple) (voir Figure 17). Cette traduction se solde par la création d'un schéma relationnel (ou relationnel-objet selon le SGBD cible) représentant le dictionnaire de données conforme à PLIB. Ce schéma représente à l'identique le méta-modèle de données EXPRESS représentant le dictionnaire de données PLIB. Il est créé en suivant des règles d'implémentation des concepts du langage EXPRESS dans le modèle de la base

de données cible (relationnel ou relationnel-objet). Ces règles seront détaillées dans le chapitre 5.

Lorsque le schéma du dictionnaire est créé dans la base de données, une ou plusieurs descriptions ontologiques peuvent y être insérées. Dans le cas des données de composants, chaque ontologie constitue le modèle d'un catalogue de composants. De manière générale, dans l'architecture que nous proposons, une ontologie peut être considérée comme un schéma conceptuel étendu de la base de données. En effet, elle définit tous les concepts (classes et propriétés) représentés. Elle peut définir également des concepts (classes et propriétés) qui ne sont pas représentés dans le schéma logique de la base de données. La représentation effectuée dans la base de données permet de représenter au même niveau les données de composants (instances) et leurs modèles conceptuels. Ceci permet de préserver les informations contenues dans ce dernier et qui disparaissent souvent au moment de l'implémentation.

3.1.2 Insertion d'une description ontologique dans la base de données cible

La deuxième étape du processus menant à l'implémentation du modèle PLIB dans la base de données cible consiste à insérer une ou plusieurs description(s) ontologique(s) dans le schéma de l'ontologie créé précédemment. Chaque ontologie constitue une instanciation du méta-modèle que représente le dictionnaire de données PLIB. Elle permet de définir, pour un univers du discours particulier, les différents concepts qui y sont utilisés (classes, propriétés). Chaque concept y est identifié d'une manière universelle permettant ainsi d'éliminer toute ambiguïté dans les références à des ontologies PLIB.

L'insertion ou l'importation d'une description ontologique dans la base de données cible (plus précisément dans le schéma de l'ontologie créé dans la première étape), est réalisée à partir d'un fichier physique (i.e. fichier d'instances d'un modèle EXPRESS) contenant cette ontologie. La possibilité qu'offre le langage EXPRESS pour un traitement automatique des modèles ainsi que de leurs instanciations permet de mettre au point un processus complètement automatique pour réaliser cette importation. En effet, nous avons procédé à la génération de commande d'insertion de données (ou plus généralement des commandes de manipulation de données du langage SQL) directement à partir du modèle du dictionnaire (i.e. de l'ontologie) et de son fichier d'instanciation. Nous donnerons plus de détails sur les différentes méthodes utilisées pour la génération automatique de code dans le chapitre 4.

3.1.3 Création du schéma du contenu

La description ontologique constitue un schéma conceptuel étendu pour les données de composants. Chaque classe y est décrite par l'ensemble de propriétés possibles dans le contexte le plus large de son utilisation. Cependant, le concepteur de la base de données peut choisir de ne représenter qu'un sous ensemble de ces propriétés dans sa base de données. Dans ce cas, le schéma du contenu ne constitue qu'une implémentation partielle du schéma conceptuel. Cela permet de donner à chaque utilisateur le choix de représenter les données qui sont pertinentes pour son application et de personnaliser ainsi sa base de données pour cette utilisation.

Pour créer le schéma du contenu, deux *scenarii* sont possibles :

1- Si le contenu de la base de données est déjà représenté dans un fichier physique (données fournies par une source d'information), la génération du schéma se base, d'une part, sur la description ontologique, et, d'autre part, sur le contenu (fichier physique) pour en extraire les propriétés représentées. Un mécanisme de nomination universel permet de lier les valeurs des instances du niveau du contenu aux descriptions du niveau de l'ontologie.

2- Si l'utilisateur désire ne représenter qu'une partie des propriétés représentées dans les fichiers du contenu fourni il doit, alors, interagir avec le système pour choisir les propriétés qu'il souhaite représenter. Ainsi, avant la création du schéma à partir de la description au niveau de l'ontologie, le système propose à l'utilisateur, pour chaque classe, la liste des propriétés représentées (au niveau du contenu). Les propriétés désirées sont sélectionnées. Puis, le système crée le schéma du contenu en se basant sur ces informations et sur le schéma conceptuel étendu représenté au niveau du dictionnaire de données (ontologie) (partie 1 de la base de données). Les règles de traduction du schéma conceptuel dans la base de données cible sont également utilisées. Ces règles sont présentes essentiellement dans le programme de génération des schémas d'une manière informelle.

En outre, une autre variante de ce deuxième scenario peut se présenter ainsi : l'utilisateur possède déjà une base de composants (le schéma du contenu est déjà créé) et il souhaite importer des données de composants similaires produites par un autre fournisseur. Dans ce cas, il a le choix, soit de modifier son schéma initial (essentiellement ajout de colonnes pour les tables existantes) soit de conserver ce dernier et de passer directement à l'importation de données décrite dans l'étape suivante (cf. section 3.1.4). S'il choisit la première option, son gestionnaire (le système) procède à la modification du schéma initial du contenu en se basant sur le contenu représenté sous la forme du fichier physique et du schéma du contenu existant. Il compare donc, d'une manière automatique, les propriétés représentées au sein du fichier physique à celles représentées dans la base de données, puis, génère des commandes de modification du schéma du contenu. Ces commandes permettent d'ajouter des colonnes, correspondant aux propriétés non représentées dans la base de données (mais fournies dans le nouveau fichier), aux tables déjà existantes dans la base de données (i.e. correspondant au schéma du contenu).

Par ailleurs, si on considère que ce gestionnaire est utilisé par un fournisseur d'information (i.e. de composants), pour créer le contenu de son catalogue de composants, ce dernier doit choisir, parmi les propriétés disponibles pour chaque classe au niveau de l'ontologie, celles qu'il souhaite représenter. Puis, une fois que les propriétés à représenter sont choisies, la création automatique du schéma du contenu peut démarrer sur la base de ces informations et sur l'ontologie représentée au sein de ce même gestionnaire.

3.1.4 Insertion du contenu

La dernière étape de ce processus consiste à insérer, dans la base de données, le contenu d'une bibliothèque de composants représentée sous forme de fichier physique conforme à la norme [ISO 10303-21:1994]. L'insertion ou l'importation des instances de composants est réalisée d'une manière automatique à partir des fichiers physiques contenant les instances du contenu. En effet, une génération automatique de code (des commandes d'insertion de données dans le langage de la base de données cible) est effectuée en utilisant un outil qui manipule les modèles et les données EXPRESS (voir chapitre 4).

L'accomplissement de cette dernière étape présente cependant des difficultés liées à la complexité de la représentation implicite des données de composants. De ce fait, une conversion des données représentées d'une manière implicite dans un modèle plus simple s'impose. Cette conversion sera introduite comme une nouvelle étape entre la deuxième et la troisième étape. Son but est de reconstruire, pour chaque famille de composants représentée dans le contenu, la liste exhaustive de toutes ses instances (table de définition) à partir des différentes tables élémentaires (cf. section 4). Nous utilisons également dans cette conversion les contraintes et les fonctions de dérivation définies dans le modèle (pour valider les instances et calculer les propriétés dérivées).

4 Conversion du modèle implicite de PLIB en modèle explicite

D'après les éléments évoqués ci-dessus, la conversion des données de composants représentées d'une manière implicite est nécessaire avant tout passage à la base de données cible. Cette conversion permet d'offrir une représentation simplifiée des données de composants et donc, facile à exploiter et à importer dans la base de données cible. Nous détaillons dans cette section les différentes phases du processus de conversion ainsi que les différents moyens pour l'effectuer.

4.1 Calcul des instances : les étapes

La conversion des données de composants décrites implicitement (par des contraintes sur les propriétés caractéristiques) vers une représentation explicite (énumération des instances sous forme de listes de couples (propriété, valeur)) passe par les étapes suivantes :

1. calcul du produit cartésien des domaines de valeurs des propriétés d'identification (pour constituer le domaine de définition d'une famille de composants (i.e. une classe)),

2. application des contraintes d'intégrité (filtres) sur ce domaine de définition pour supprimer les valeurs non autorisées,

3. calcul des propriétés dérivées en utilisant les valeurs des propriétés d'indentification (après application des contraintes d'intégrité) et en utilisant les fonctions de dérivations représentées dans le modèle,

4. représentation des instances calculées dans une table globale appelée table de définition de familles de composants par une jointure entre le domaine de définition de la famille de composants et les domaines des propriétés dérivées.

A l'issue de ces étapes, chaque famille de composants représentée dans le modèle implicite sera décrite par une table unique où chaque ligne représente une instance licite. Les données, représentées de cette manière, pourront être facilement importées par la suite vers une base de données cible (relationnelle ou relationnelle-objet).

4.2 Exploitation de la connaissance procédurale

Une grande partie de la conversion d'une représentation implicite vers une représentation explicite se fait sur l'exploitation de la connaissance procédurale. Celle-ci constitue le cœur de la représentation implicite, et il faut évaluer les différentes expressions (méta-programmes) représentant les différentes contraintes d'intégrité du modèle implicite (contraintes fonctionnelles et assertionnelles). Cela permet de calculer les propriétés dérivées, de valider les contraintes appliquées sur les domaines (les gardes) et de valider les contraintes appliquées sur les instances (contraintes d'unicité ou d'autres restrictions possibles représentées par des filtres). Trois approches sont possibles pour réaliser cette évaluation :

1- parcours des arbres de syntaxe représentant les expressions (les méta-programmes) par un programme générique qui calcule et renvoie le résultat final,

2- utilisation de la programmation événementielle avec le langage EXPRESS. Elle consiste à embarquer dans chaque entité d'expression sa propre fonction d'évaluation. Cela peut se réaliser en complétant les entités représentant des expressions par des attributs dérivés. La programmation événementielle sera détaillée dans le chapitre 4,

3- génération pour chaque expression (méta-programme) d'un programme compilable équivalent, dans un langage de programmation particulier, puis exécution de ces différents programmes pour évaluer le résultat final.

Nous détaillons chacune de ces approches dans les paragrahes suivants.

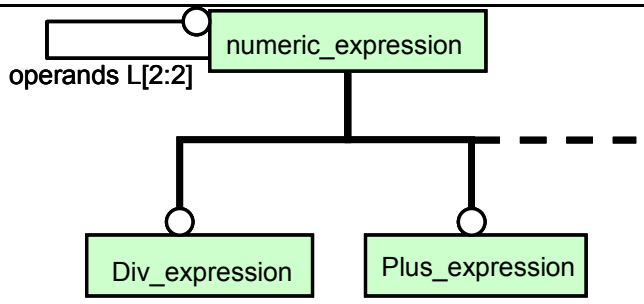
4.2.1 Evaluation générique des expressions

Dans le modèle PLIB implicite, toute la connaissance procédurale est représentée par méta-programmation. Nous avons vu dans la section 2.7.3 du chapitre précédent que la représentation des expressions (les méta-programmes) était basée, d'un point de vue structurel, sur des arbres de syntaxe abstraite.

L'évaluation générique est une des approches possibles pour l'évaluation des méta-programmes. Elle est basée sur l'utilisation d'un seul programme générique pouvant évaluer n'importe quel type d'expression. Ce programme peut être écrit dans n'importe

quel langage de programmation pouvant accéder à une base de données EXPRESS (à des instances des méta-programmes modélisés en EXPRESS). Cet accès est réalisé soit directement (cas d'un langage utilisant les notations EXPRESS : EXPRESS-C [Staub 94] par exemple), soit à travers une API (Application Programming Interface) (cas d'un langage différent d'EXPRESS : C++, JAVA, …).

Ce programme générique parcourt l'arbre représentant les expressions en profondeur, de la gauche vers la droite. L'évaluation commence par les feuilles et remonte jusqu'à la racine. Ce programme est exécuté d'une manière récursive. Le programme d'évaluation peut être une fonction, par exemple, constituée de plusieurs blocs permettant d'évaluer chacun un type précis d'expression (*plus_expression, mult_expression,* …) (voir Figure 26).



Modèle des expressions

```
Function generic_evaluation (exp : in expression) return generic_value;
…
If 'div_expression' in typeof(exp) then
        resultat = generic_evaluation(operand[1]) / generic_evaluation( operand[2]);
Elsif …,
```

Un bloc d'une fonction d'évaluation générique écrite en EXPRESS-C

Figure 26 : exemple d'évaluation d'une expression simple par une fonction générique

La Figure 26 donne un exemple de modèle d'une expression simple et d'une partie d'une fonction générique permettant d'évaluer cette expression. Les opérandes d'une expression peuvent être eux-mêmes des expressions (un arbre). La fonction générique est donc appelée d'une manière récursive jusqu'au renvoi final des résultats de l'expression à évaluer. Cette fonction reçoit en paramètre d'entrée une expression et renvoie une valeur générique, de type *generic_value,* qui est le résultat de l'évaluation de cette expression. Le type *generic_value* est une union de types composée de types numériques et booléens (pour évaluer aussi bien les expressions numériques que les expressions booléennes). Le résultat peut donc appartenir à un type parmi cette union (il peut être un réel, un entier ou un booléen). La fonction **TYPEOF** permet de connaître le type de l'expression à évaluer (*div_expression, plus_expression,* …) et ainsi d'appeler la bonne composante d'évaluation (voir Figure 26).

4.2.2 Auto évaluation des expressions

La deuxième approche pour l'évaluation des expressions par l'auto évaluation est basée sur la programmation événementielle avec l'utilisation des attributs dérivés du langage EXPRESS. Elle consiste à doter chaque expression de sa propre fonction d'évaluation. Chaque expression dans le modèle est complétée par un attribut dérivé, permettant de calculer sa valeur. Or, les attributs dérivés n'apparaissent pas dans les fichiers d'échange et donc les instances du modèle initial pourront effectivement être vues comme des instances du modèle modifié. Si le système lit les expressions échangées comme étant des instances du modèle enrichi par des attributs dérivés, une simple consultation de ces attributs pourra envoyer le résultat d'évaluation d'une expression. Nous présentons dans la Figure 27 le nouveau modèle de représentation des expressions enrichi (par des attributs dérivés). Cette figure présente une partie du modèle global représentant les expressions (une partie des expressions numériques). L'attribut *the_value* est rajouté à la racine comme étant un attribut explicite de type *generic_value*. Cet attribut est redéfini dans les sous-expressions en restreignant son type et en le déclarant comme attribut dérivé. A ce niveau de la structure, une fonction permettant l'évaluation de l'expression est associée à l'attribut dérivé. Il faut noter que le langage EXPRESS-G ne permet pas de représenter les fonctions ou les expressions calculant les valeurs de ces attributs. Les fonctions utilisées pour calculer les attributs dérivés et représentées sur la figure sont donc des extraits du modèle EXPRESS textuel complet.

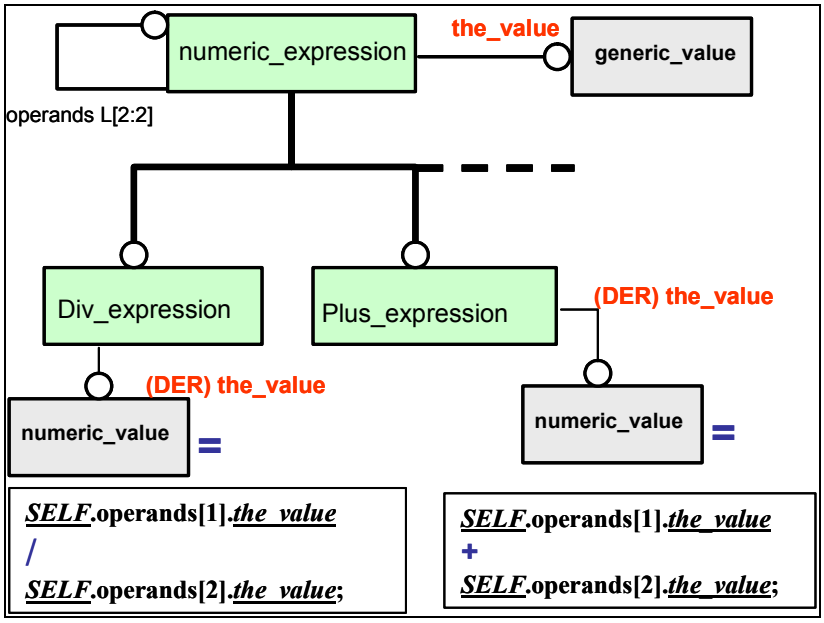


Figure 27 : modèle des expressions enrichi des attributs dérivés

4.2.3 Evaluation par programmes compilables générés

La troisième approche d'évaluation d'expressions méta-programmées est fondée sur la génération de programmes compilables représentant une traduction des méta-programmes, représentant la connaissance procédurale, dans un langage compilable (Ada, C++, Java, …) ou interprété (PL/SQL, JavaScript, …). Le(s) programme(s) généré(s) seront ensuite compilé(s) et/ou exécuté(s). La Figure 28 présente un exemple de génération d'une fonction en langage PL/SQL permettant de calculer une

expression simple. En réalité, l'évaluation des expressions par cette méthode passe par deux étapes :

1- la génération de fonctions ou de procédures correspondant aux différents opérateurs pouvant figurer dans une expression (génération de sous-programmes),

2- la génération de programmes principaux faisant appel à ces fonctions avec les valeurs des variables échangées dans le modèle.

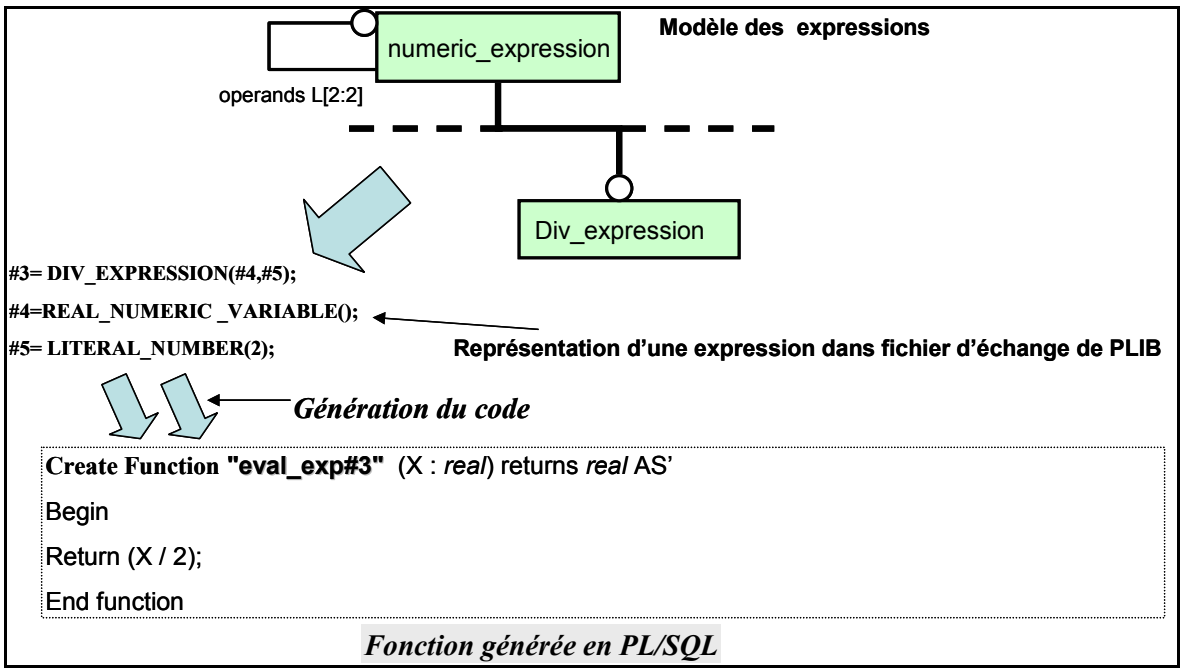


Figure 28 : génération de programmes à partir de méta-programmes

La Figure 28 illustre un exemple de génération d'une fonction d'évaluation pour l'expression identifiée par #3 dans le modèle d'échange. La fonction *"eval_exp#3"* (les guillemets sont utilisés pour permettre l'utilisation de caractères spéciaux (#) dans PL/SQL) peut être générée d'une manière complètement automatique. La variable *real_numeric_variable,* représentée par une variable *X* (qui est un paramètre d'entrée de la fonction d'évaluation), pourra ensuite (appel de la fonction) être associée à l'identifiant universel d'une propriété. Ainsi, l'expression peut permettre le calcul, par exemple, de la propriété hauteur d'une vis en fonction de son diamètre (h= d/2).

Si cette approche peut être utilisée pour évaluer des expressions, elle peut également l'être pour la génération de contraintes dans la base de données cible. Cela est possible car les expressions sont utilisées aussi bien dans la représentation des contraintes fonctionnelles (dérivation par exemple) que dans des contraintes assertionnelles (prédicats) (cf. sections 2.2 et 2.7.2 du chapitre 2). Techniquement, la mise en œuvre de la génération de contraintes n'est pas différente de celle de l'évaluation des expressions. En effet, la contrainte est générée en deux parties :

1- la génération d'une fonction représentant l'expression attachée à la contrainte. Prenons l'exemple d'une contrainte exprimée sur la

propriété *diamètre* d'une vis (*diamètre* > 5 , par exemple), la fonction qui peut être générée pour cette expression est la suivante (en PL/SQL) :
Create function diametre_restriction (x: real) returns Boolean as ' return ("eval_expr#6"(x)) ;
end_function ;
où *"eval_expr#6"* est une expression représentant la contrainte (d>5) générée d'une manière automatique (voir Figure 28) ;

2- l'association de cette fonction à une contrainte **CHECK** (une grande partie des SGBDs permet l'utilisation de fonctions booléennes dans ce type de contrainte) déclarée sur la table représentant la famille de vis dans la base de données cible, par exemple.

4.2.4 Association des variables dans les expressions à leurs valeurs dans le modèle PLIB

Nous avons vu dans les paragraphes précédents trois approches permettant d'évaluer des expressions représentées dans le modèle implicite de PLIB. Dans ces paragraphes, nous avons étudié l'évaluation des expressions sans rentrer dans les détails concernant l'interprétation des variables. En effet, dans le modèle de représentation des expressions, chaque variable est attachée à une sémantique particulière dans le modèle général qui définit l'interprétation qui doit en être faite (associer une variable à une propriété d'une famille de composants, par exemple).

Les variables sont représentées dans le modèle des expressions comme étant des sous-types des expressions. La Figure 29 montre un modèle simplifié de représentation de variables dans le modèle des expressions de PLIB.

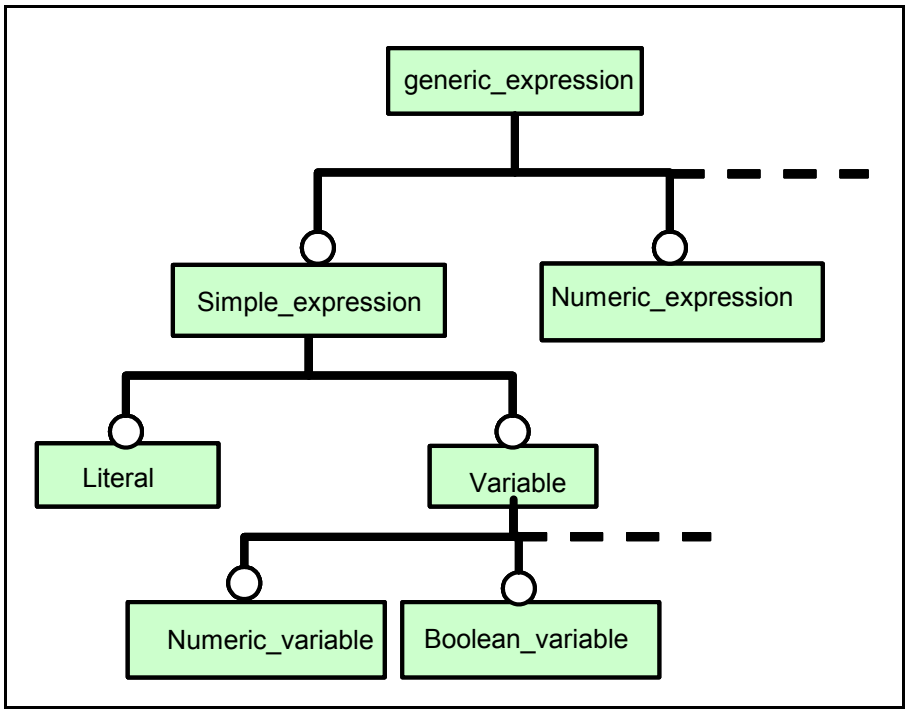


Figure 29 : représentation des variables dans le modèle des expressions de PLIB.

Chaque variable a trois aspects différents [ISO 13584-20:1998]:

1. une représentation syntaxique qui est un symbole utilisé pour construire une expression,

2. un type de données qui définit le domaine de valeurs de la variable,

3. une sémantique qui définit sa signification et permet de récupérer sa valeur lors d'une évaluation.

Une variable particulière est représentée comme une instance de l'entité *variable*. Chaque instance est associée à un identificateur (représenté par '#' suivi d'un numéro dans les fichiers physiques) qui constitue le symbole représentant la variable dans une expression.

Les domaines de variables sont représentés par un typage fort effectué par le sous-typage de l'entité *variable* (en *numeric_variable, boolean_variable, etc.*). Le type de données de chaque expression (i.e. variable) est vérifié lors de l'échange des expressions.

Pour déterminer sa signification dans le modèle d'échange, chaque variable doit être associée à une sémantique particulière dans le modèle de PLIB. Cette sémantique est représentée par une entité abstraite *variable_semantics* qui doit être sous-typée à des sémantiques particulières selon l'utilisation de la variable (associer une variable à une propriété d'une famille de composants, par exemple). Cette association permet de doter la variable d'une valeur au moment de l'évaluation d'une expression.

La relation sémantique est assurée par une entité d'association intermédiaire : il s'agit de l'entité *environnement*. La Figure 30 illustre la représentation de cette relation sémantique dans le modèle des expressions de PLIB [ISO 13584-20:1998].

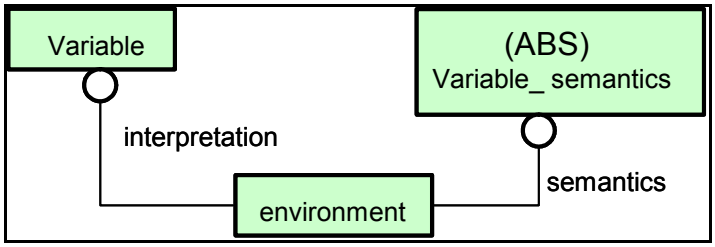


Figure 30 : Association sémantique entre une variable et la sémantique qui y est attachée dans le modèle d'échange

Nous avons donné ci-dessus la représentation formelle de l'association entre les variables et la sémantique à laquelle elles sont attachées. Le lien formel entre chaque variable et la sémantique qu'elle représente permet d'associer facilement des valeurs à ces variables et donc évaluer les expressions.

4.2.5 Comparaison entre les trois approches d'évaluation

Nous avons décrit dans les paragraphes précédents trois approches possibles permettant l'exploitation de la connaissance procédurale et particulièrement l'évaluation des expressions. Les trois approches présentent des avantages et des inconvénients, et chacune est plus ou moins adaptée à une utilisation particulière.

La première approche est générique (évaluation par programme générique pouvant accéder à une base de données EXPRESS) et ne nécessite pas de changement dans le modèle initial. Elle l'utilise plutôt comme une ressource extérieure pour lire et manipuler la connaissance échangée. Toutefois, le programme générique reste figé une fois qu'il est mis en place. Chaque extension du modèle initial nécessite donc une mise à jour du programme générique, c'est-à-dire l'ajout de lignes de codes permettant l'évaluation de la ou des expressions rajoutées. Cela fait de cette méthode une méthode statique. Elle est bien adaptée si le modèle EXPRESS n'évolue pas. Elle l'est moins tant que le modèle reste susceptible d'évoluer, ce qui était le cas pendant notre étude (la norme PLIB n'était pas encore figée)

La deuxième approche nécessite des changements dans le modèle initial. Ces changements se résument à l'ajout d'attributs dérivés pour chaque entité représentant une expression dans le modèle. Une fois que le modèle est enrichi par les attributs dérivés, l'évaluation des expressions se résume à une simple consultation d'une valeur d'un attribut d'une entité, représentant une expression dans le modèle. L'avantage est qu'aucun programme supplémentaire n'est nécessaire pour effectuer cette tâche. A chaque ajout d'une nouvelle expression, celle-ci contient, naturellement, sa propre fonction d'évaluation. Cette méthode d'évaluation est donc dynamique et évolutive.

La troisième approche est plus longue et nécessite le passage par plusieurs étapes avant d'arriver au résultat final, et en particulier la génération de toutes les fonctions prédéfinies du langage EXPRESS. Cependant, cette approche peut être très intéressante pour la génération et le stockage de contraintes complexes dans les bases de données. En effet, les contraintes complexes, définies dans le modèle implicite, peuvent être générées en utilisant la même approche puis stockées dans la base de données cible. Les propriétés des familles de composants représentées par des contraintes fonctionnelles (fonctions) peuvent être directement représentées par des fonctions générées en suivant la même approche puis stockées au sein de la base de données cible.

Parmi ces trois approches, nous avons choisi la seconde pour évaluer les expressions du modèle implicite lors de la conversion des données. En effet, cette approche, très modulaire, supporte facilement des évolutions du modèle, ce qui était encore le cas au début de notre étude. De plus, cette approche est très commode à mettre en œuvre car une fois que le modèle des expressions est enrichi par des attributs dérivés, l'évaluation d'une expression s'effectue de la manière la plus simple : par consultation d'une valeur d'un attribut. Ainsi, l'évaluation de chaque expression devient inhérente au modèle des expressions et son évolution (par ajout de nouvelles expressions) n'altère en rien cette méthode d'évaluation.

4.3 Génération de la table de définition d'une famille de composants

Les opérations permettant la génération des instances explicites (construction de la "table de définition" d'une famille de composants) vont d'une simple jointure ou produit cartésien jusqu'au filtrage et validation de contraintes. La construction de cette table suit les étapes suivantes (voir la section 4.1) :

1- calcul du domaine de définition des propriétés d'identification de la famille de composants,

2- filtrage de ce domaine en appliquant les contraintes d'intégrité,

3- calcul des propriétés dérivées, et enfin

4- jointure entre le domaine de définition calculé en 1 et les domaines des propriétés dérivées.

Nous donnons dans les sections suivantes plus de précisions sur chacune de ces étapes.

4.3.1 Domaine de définition des propriétés d'identification

Nous appelons domaine de définition des propriétés d'identification d'une classe ou d'une famille de composants le produit cartésien de tous les domaines de valeurs des propriétés d'identification. Ce produit est représenté sous forme de listes de listes (représentant les tables dans le modèle implicite). Il correspond à un ensemble de tuples. Chacun de ces tuples permet d'identifier une instance unique de la classe décrite par ses propriétés. Le calcul de ce domaine de définition constitue le point de départ de l'opération de conversion des instances implicites en instances explicites. Comme nous l'avons précisé dans le chapitre précédent, les domaines de valeurs de certaines propriétés peuvent être soumis à des gardes (des conditions portant sur les domaines d'autres propriétés). Ils permettent de limiter les valeurs d'une propriété par rapport aux valeurs d'autres propriétés. Lors du produit cartésien, ces gardes sont évaluées à chaque étape de ce produit. Ainsi, l'élimination des valeurs illicites du domaine de définition des propriétés d'indentification se fait au fur et à mesure du calcul du produit cartésien. A l'issue de cette opération, nous obtenons l'ensemble des valeurs des clés primaires de la table de définition de la famille de composants.

Les gardes étant représentés par des expressions, leur validation passe par l'évaluation de ces expressions. Pour réaliser cette évaluation, la deuxième approche (programmation évenementielle par attributs dérivés) est utilisée (cf. paragraphe 4.2).

4.3.2 Filtrage du domaine de définition

Dans le chapitre 2, nous avons mentionné que les filtres permettent d'imposer des contraintes supplémentaires sur le modèle de données pour restreindre davantage le nombre d'instances licites (par exemple : tous les diamètres et longueurs sont autorisés,

sauf si d ≤ 5, alors l ≤ 17). Le filtrage du domaine de définition est l'application de contraintes d'intégrité définies à travers les filtres pour éliminer ces instances illicites (voir Figure 31).

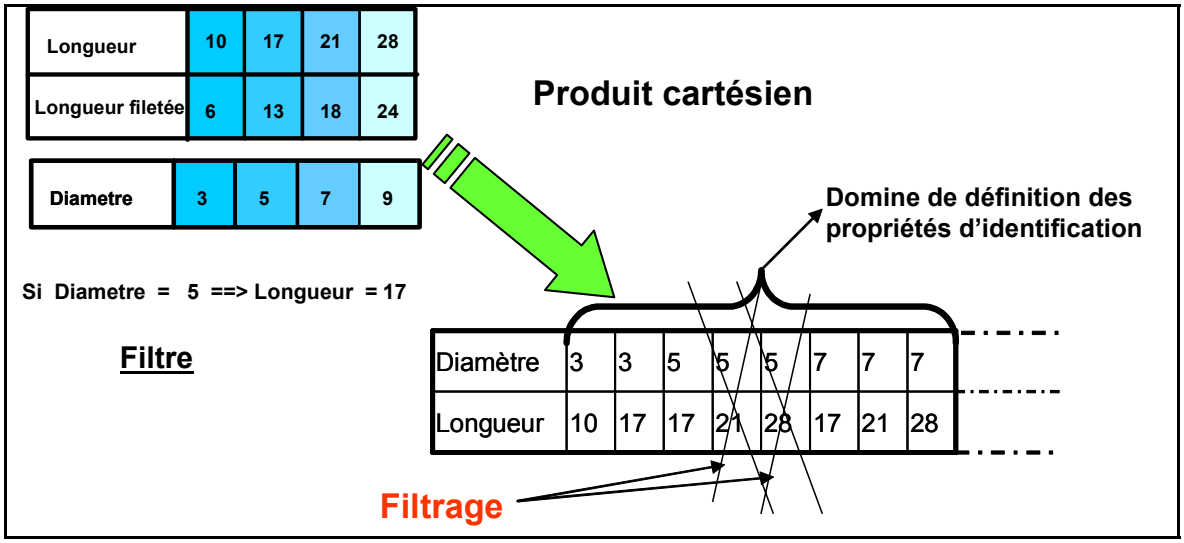


Figure 31 : application des filtres sur le domaine des propriétés d'identification

La Figure 31 illustre un exemple simple de filtrage de données. En effet, le produit cartésien entre les deux domaines de valeurs des propriétés *Diametre* et *Longueur* produit un ensemble de tuples identifiant des instances non autorisées ((5,21) et (5,28)). Pour éliminer ces instances non autorisée, les concepteurs du modèle PLIB implicite ont ajouté des filtres (voir Figure 31). Le filtrage de données consiste alors, d'une part, à appliquer les contraintes définies par ces filtres sur tous les tuples obtenus par le produit cartésien, et, d'autre part, à supprimer tous ceux qui ne remplissent pas ces contraintes (les tuple barrés sur la Figure 31).

Les contraintes, représentant les filtres, sont évaluées comme des expressions (voir paragaraphe 4.2) de la même façon que les gardes. En effet, chaque filtre est associé à une expression booléenne qui définit la contrainte.

4.3.3 Calcul des propriétés dérivées

Une fois que le domaine de définition d'une famille de composants est calculé et filtré, le calcul des propriétés dérivées peut commencer. Ces propriétés étant représentées par des expressions dans le modèle implicite de PLIB, ce calcul revient à évaluer ces expressions. La deuxième approche (programmation évenèmentielle par attributs dérivés) est utilisée (cf. paragraphe 4.2). pour cette évaluation. Nous avons utilisé, pour notre part, l'approche de l'auto évaluation des expressions pour calculer les valeurs des propriétés dérivées.

4.3.4 Jointure entre les domaines des propriétés d'identification et le domaine des propriétés dérivées

L'étape finale pour la construction de la table globale est la jointure entre le domaine de définition des propriétés d'identification et les domaines des propriétés dérivées. Cette jointure peut en effet être réalisée en parallèle du calcul des valeurs des propriétés dérivées. Elle permet de construire au fur et à mesure la table de définition d'une famille de composants ; c'est-à-dire la description explicite, sous forme de table, de l'ensemble de ses instances. Elle est réalisée sur la (les) propriété(s) qui constitue(nt) le domaine de l'expression associée à chaque propriété dérivée. Ces propriétés sont représentées par l'attribut *Assumes* de l'entité *Domain_restriction* représentant chaque dérivation (voir Figure 14).

4.4 Limites de la conversion de données

D'une manière générale, la représentation implicite des données est plus riche et couvre un éventail d'informations et de connaissances plus large que ce que l'on peut représenter par une simple énumération. Lorsqu'un domaine de valeurs d'une propriété est infini, le nombre d'instances générées par le calcul peut l'être également. Dans ce cas, l'énumération complète de ces instances dans une représentation explicite est impossible. Ceci arrive dans deux cas :

- lorsque l'ensemble de composants potentiels est infini. C'est le cas lorsque le composant peut être défini sur mesure. Nous ne pourrons donc pas représenter ce cas-là,

- lorsque l'on veut représenter les comportements des composants dans des domaines continus:

  En effet, la représentation des comportements des composants s'effectue à travers la représentation de paramètres de contexte et de propriétés dépendant du contexte. Ceux-ci peuvent avoir, dans certains cas, des domaines de valeurs continus (la charge exercée sur un roulement, par exemple). Pour pouvoir effectuer la conversion de données dans tous les cas, nous avons donc choisi de ne pas représenter les propriétés dépendant du contexte et les paramètres de contexte. De cette manière, les données converties représentent effectivement toutes les instances (sachant que nous ne traitons pas les composants sur mesure) mais ne représente pas leurs comportements. Notons qu'à la suite de l'identification de cette difficulté une restriction a été introduite dans le modèle PLIB lui-même pour que les comportements puissent être exprimés en un point unque de mesure (exemple : à la température de 20°, la résistance = 12 Ω). Ce type de fonctions discrètes réduites à un seul point qui permet de préciser le contexte particulier d'une mesure peut être représenté avec notre approche.

5 Gestionnaire de bibliothèques de composants PLIB fondé sur un SGBD

Nous avons donné dans le paragraphe 2 l'architecture globale de la base de données obtenue par l'implémentation de l'approche hybride. Nous définissons dans ce paragraphe l'architecture globale du gestionnaire de bibliothèques de composants basé sur cette implémentation. Elle est illustrée par la Figure 32. Le but essentiel de ce gestionnaire est d'offrir un système puissant pour le stockage, la sélection et la sécurité des données de composants. Le stockage et la sécurité de données sont assurés par le SGBD. La sélection des données est assurée soit par un module externe à la base de données, soit par le SGBD lui-même, selon la nature des données recherchées et selon l'approche utilisée pour la représentation des données correspondant à la connaissance procédurale (voir le paragraphe 5.2).

5.1 Initialisation de la base de données

Les étapes nécessaires pour représenter, à l'intérieur de la base de données, une bibliothèque PLIB représentée sous forme de fichier physique (i.e. conforme à la norme [ISO 10303-21:1994]), sont les suivantes (voir Figure 32) :

1- les données sont chargées en mémoire à partir d'une base de données EXPRESS. Afin d'interpréter et d'accéder à ces données, le modèle de données implicite de PLIB (Schémas EXPRESS de [ISO 13584-24:2003]) est utilisé ;

2- une fois les données chargées, elles sont converties dans le format explicite. La conversion des données représentées d'une manière implicite dans une représentation explicite est réalisée par un outil supportant EXPRESS (Ecco Toolkit [Staub et al 96] par exemple), puis le processus mis en œuvre est celui décrit dans le paragraphe 4.3 ;

3- importation de l'ontologie décrivant les données converties (si celle-ci n'est pas déjà représentée dans la partie 2 de la base de données) selon la méthode décrite en sections 3.1.1 et 3.1.2 ;

4- les instances sont alors rangées dans des tables de définition des familles de composants (une table par famille de composants) et insérées dans la base de données cible (dans la partie 3).

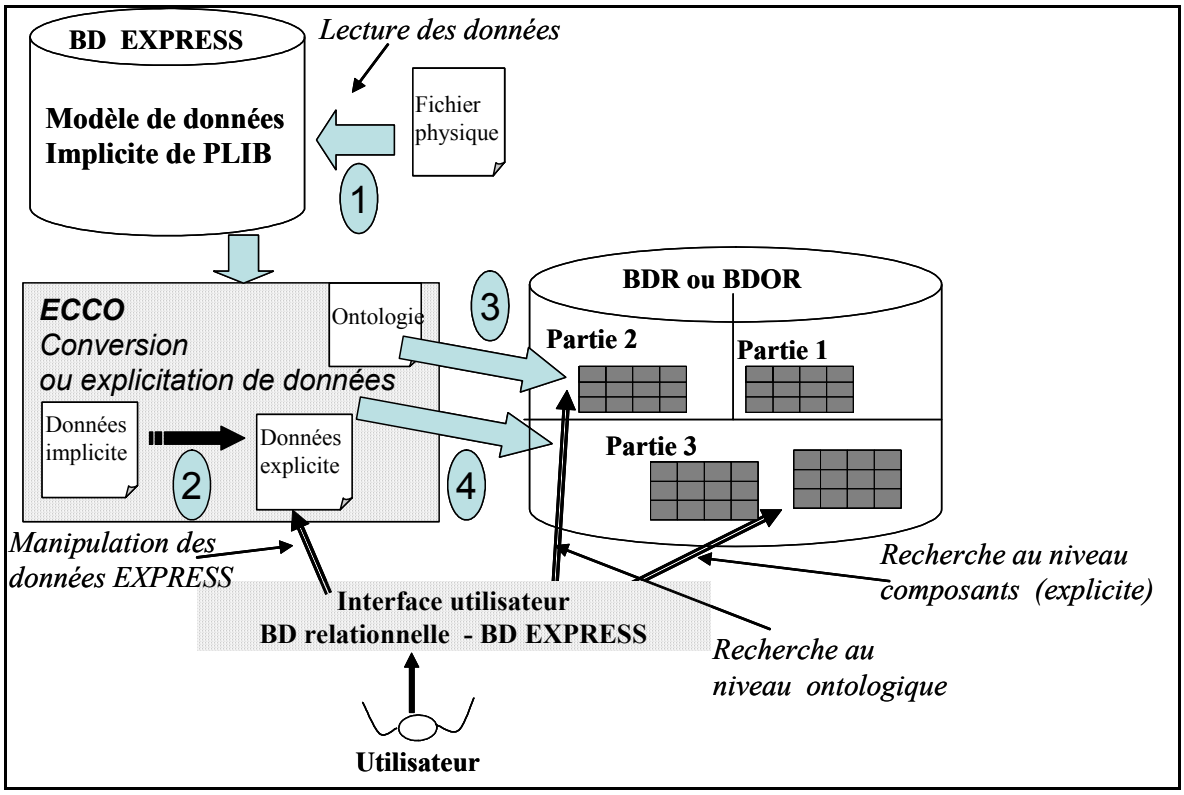


Figure 32 : Architecture globale du gestionnaire de bibliothèque de composants conforme au modèle implicite de PLIB

5.2 Sélection de composants : trois niveaux d'interrogation des données dans le gestionnaire des données de composants

Parmi les fonctionnalités essentielles du gestionnaire de données de composants figure la fonction sélection des composants. Elle constitue une activité importante dans le processus d'ingénierie. Afin de faciliter cette tâche, des outils permettant la sélection des composants dans la base de données doivent être créés. Dans l'architecture proposée, trois niveaux d'interrogation de la base de données ont été identifiés (voir Figure 32):

1- interrogation au niveau explicite (en se basant essentiellement sur des propriétés caractéristiques non dépendantes du contexte des composants),

2- interrogation au niveau de l'ontologie (indépendamment du schéma du contenu des bases de données cible).

3- interrogation au niveau des comportements de composants (i.e. niveau implicite) (en se basant sur des paramètres de contexte de leur insertion).

L'interrogation de la base de données au niveau des instances explicites est une interrogation classique d'une base de données. A ce niveau, les requêtes sont des requêtes simples et sont écrites directement dans le langage de manipulation de données de la base de données cible (langage SQL par exemple). C'est d'ailleurs une des raisons du choix de l'approche hybride.

L'interrogation au niveau de l'ontologie ouvre un nouvel horizon dans la sélection de données dans les SGBD. A ce niveau, les requêtes ne dépendent pas de l'implémentation du modèle de données dans un système particulier. L'intérêt d'une telle approche est que les mêmes requêtes peuvent être utilisées pour interroger des données dans des systèmes hétérogènes, implémentant la même ontologie. Elle permet également d'écrire des requêtes sur une base de données sans se soucier du schéma d'implémentation de cette base (qu'une classe soit représentée dans une ou plusieurs tables n'affecte en rien la requête). Elle peut jouer un rôle important pour l'interrogation des bases de données fédérées ou distribuées [Bellatreche 03]. Les requêtes à ce niveau nécessitent l'utilisation d'un nouveau langage : le langage CQL (Class Query Language) est en cours de développement pour permettre ce type de requêtes [Murayama 00], [Murayama 03].

Finalement, l'interrogation au niveau des comportements des composants (en se basant sur le modèle implicite) vise à offrir à l'utilisateur, après implémentation, les mêmes possibilités de sélection des composants que celles offertes par le modèle implicite lui-même (par des paramètres de contexte essentiellement). A cause de la perte d'information dans le modèle de la base de données cible résultant à l'ussue de la conversion des données (voir 3.4), et afin de préserver toutes les informations véhiculées par le modèle implicite après son implémentation, deux approches, permettant la représentation de la connaissance procédurale dans le gestionnaire de données de composants, ont été étudiées :

1- Représenter dans le modèle de la base de données cible les données que l'on peut expliciter (cf. pragraphe 4.4) et garder un lien vers le modèle implicite et la base de données EXPRESS (cf. Figure 32) pour produire les autres données non représentées (représentation externe de la connaissance procédurale) ;

2- Transférer la connaissance procédurale (représentant les propriétés d'une manière implicite) en programme compilable (ou interprétable) et la représenter au sein de la base de données cible (représentation interne de la connaissance procédurale).

Ces deux possibilités de représentation de la connaissance procédurale sont détaillées dans le paragraphe 5.3 suivante.

5.3 Représentation de la connaissance procédurale dans le gestionnaire de données de composants

5.3.1 Représentation externe de la connaissance procédurale

La représentation externe de la connaissance procédurale consiste à conserver un lien entre le SGBD implémentant le modèle PLIB (après sa conversion bien sûr) et le système EXPRESS (i.e ECCO Toolkit) contenant la représentation de la même bibliothèque de composants selon le modèle implicite de PLIB (cf. Figure 32). Ce lien permet alors l'exploitation de la connaissance procédurale représentée dans le modèle implicite. Parmi les informations représentées d'une manière procédurale, nous

pouvons donner l'exemple des propriétés dépendant du contexte et les paramètres de contexte (cf. paragraphe 2.7.3). Ces derniers peuvent avoir un nombre très important (voire infini) de valeurs dans certains cas (durée de vie d'un roulement qui dépend de la charge exercée sur celui-ci). Lors d'un processus de sélection, l'utilisateur donne la valeur du paramètre de contexte (exemple : charge exercée sur un roulement) qui sera utilisée par le système, avec éventuellement d'autres données correspondant aux caractéristiques du composant recherché extraites du SGBD (exemple : diamètre du roulement). La propriété qui dépend du contexte est alors calculée par le système EXPRESS à partir de l'expression représentée dans le modèle implicite. Dans ce cas, la passerelle établie entre le modèle de la base de données cible (le SGBD) et le modèle implicite (système EXPRESS) permet une interrogation simultanée des deux modèles (le modèle de la base de données cible et le modèle implicite de PLIB). Une maquette conforme à cette approche est actuellement en cours de développement dans le laboratoire.

5.3.2 Représentation interne de la connaissance procédurale

La représentation interne de la connaissance procédurale consiste à la stocker et la gérer au sein même du SGBD. Dans ce cas, le gestionnaire de données de composants devient entièrement autonome et n'a plus aucun lien avec un système EXPRESS représentant cette connaissance. La connaissance procédurale est représentée par des procédures stockées et des contraintes au sein de la base de données cible. Les contraintes fonctionnelles (des expressions permettant de calculer les propriétés dérivées) seront représentées par des fonctions ou des procédures, associées à des vues. Les contraintes assertionnelles (expressions représentant des prédicats) sont représentées si besoin est, c'est-à-dire si elles sont pertinentes dans le modèle explicite et non pas utilisées seulement pour gérer la factorisation de l'information dans le modèle implicite, par des contraintes au sein de la base de données cible (contraintes vérifiées par des fonctions et activées par des triggers (Un trigger définit une action qui est déclenchée par l'arrivée d'un événement. : une mise à jour, par exemple).

5.3.3 Traitement des paramètres de contexte

Pour pouvoir effectuer des sélections de composants basées sur des paramètres de contexte, il faut représenter les paramettre de contexte par des variables et les propriétés dépendantes de contexte seront représentées par des fonctions. Selon la nature du paramètre de contexte représenté, ses valeurs pourront être représentées soit dans une table (si ses valeurs sont limitées et sont fournies dans le modèle d'échange), soit par une simple variable de fonction (domaine continu).

Ces programmes (fonctions et procédures) sont utilisés lors d'une sélection pour évaluer une propriété dépendant du contexte. Dans ce cas, des structures permettant la présentation de données (une classe avec toutes ses propriétés, par exemple) pourraient être créées (utilisation de vues, par exemple).

Les fonctions et procédures représentant les propriétés dépendant du contexte peuvent être générées à partir des expressions les représentant dans le modèle implicite en utilisant l'approche décrite dans le paragraphe 4.2.3.

5.4 Représentation de la connaissance structurelle et descriptive dans le gestionnaire

Les aspects structurels (classes) et descriptifs (attributs) sont représentés à deux niveaux différents. Le premier est le niveau de l'ontologie. En effet, l'ontologie étant créée pour définir les concepts (classes et attributs) de l'univers du discours traité (ici le domaine de données de composants), les classes sont d'abord décrites à ce niveau. Elles sont modélisées dans une hiérarchie de classes avec un héritage simple. Ceci constitue la représentation de la connaissance structurelle. Chaque instance est associée à un ensemble de propriétés (représentation de la connaissance descriptive). Ainsi, les deux types de connaissance sont représentés de la même manière que dans le modèle de données PLIB car le dictionnaire de données est représenté, dans la base de données cible, à l'identique du modèle EXPRESS le définissant.

Le second niveau de représentation de ces deux types de connaissance se trouve dans le schéma du contenu. Toutes les classes ayant un contenu sont représentées par des tables (leurs tables de définition) éventuellement associées à des vues si l'on représente de façon interne la connaissance procédurale (cf. section 5.3.2). Concernant les classes abstraites et la hiérarchie, elles peuvent être soit directement représentées par une hiérarchie de tables dans des bases de données relationnelles-objet, soit simulées dans les bases de données purement relationnelles. La connaissance descriptive est représentée par les colonnes (attributs) de tables représentant des familles de composants. Soulignons que la représentation des classes abstraites (et donc sans population) n'est pas obligatoire dans le schéma du contenu. Ce dernier peut très bien ne représenter que les classes feuilles dans le contenu. Dans notre cas, nous avons choisi de représenter la structure hiérarchique complète afin de permettre la sélection générique des composants. Nous discuterons plus en détail la représentation que nous proposons pour la connaissance structurelle dans le schéma du contenu dans le chapitre 5.

6 Conclusion

Nous avons présenté dans ce chapitre un modèle d'implémentation de l'approche hybride dans un SGBD cible. Cette implémentation a conduit à dégager une nouvelle architecture de base de données. Elle consiste à représenter, dans la même base de données, les données concernant les composants (ou toute autre population d'entités représentée selon le modèle PLIB) ainsi que leur description ontologique (modèle conceptuel étendu). Cette architecture présente des points très intéressants qui offrent des ouvertures sur la structuration des bases de données relationnelles ou relationnelles-objet. En effet, le modèle de données PLIB, même s'il visait initialement à modéliser des données de composants, pourrait être parfaitement utilisé dans d'autres domaines (intégration de bases de données hétérogènes [Bellatreche 03], commerce éléctronique [Aklouf et al 03], …). La raison de cette généralité vient de l'esprit même du modèle de données PLIB. En réalité, il offre un méta-modèle permettant de représenter tout univers du discours dont les concepts peuvent être représentés dans une hiérarchie de classes décrites par un ensemble de propriétés et reliées entre elles par des relations pouvant être représentées par des attributs.

Au-delà des domaines pour lesquels des ontologies existent ou pourraient être définies, cette architecture peut être vue simplement comme une représentation simultanée d'un modèle conceptuel des données et de son implémentation dans la même base de données. Cela permet de garder toute la richesse sémantique que présente un modèle conceptuel, richesse qui est perdue après passage aux schémas logiques et physiques de données. Cette perte d'information est due aux diverses restrictions imposées par les modèles logiques et physiques de données lors de l'implémentation des modèles conceptuels (exemple : normalisation, restriction sur l'héritage ou le référencement, etc.).

Comte tenu la complexité du modèle implicite et de la difficulté de le représenter dans les SGBD actuels (essentiellement relationnels et relationnels-objets), sa conversion dans un modèle plus simple et explicite s'avérait indispensable. Nous avons présenté, dans ce chapitre les différentes étapes de cette conversion. L'étape la plus déterminante de cette conversion est celle de l'évaluation des expressions (exploitation de la connaissance procédurale). Trois approches permettant l'évaluation des expressions méta-programmées ont été présentées : l'utilisation de programmes d'évaluation générique, l'évaluation par attributs dérivés du langage EXPRESS (l'auto évaluation) et finalement la compilation avec la génération automatique de programmes cibles à partir de ces expressions. La conversion des données de composants représentées d'une manière implicite aboutit à une représentation de toutes les instances de classes par des n-uplets. Cette représentation est partielle car chaque instance n'est représentée que par ses propriétés ayant un domaine discret et fini (en particulier, certaines propriétés dépendantes du contexte ne peuvent ainsi pas être représentées) ; mais elle permet néanmoins de représenter toutes les instances existantes dès lors que celles-ci sont en nombre fini.

On a étudié ensuite l'architecture du gestionnaire de données de composants résultant de cette implémentation. Cette architecture permet de prendre en charge les différentes tâches que doit effectuer ce gestionnaire à savoir la représentation, le stockage, la sécurisation et la sélection de données. La représentation de données dans la base de données cible est effectuée selon l'approche hybride discutée dans le chapitre précédent. En ce qui concerne la connaissance procédurale (expressions et fonctions), nous avons dégagé deux approches possibles permettant de la représenter. La première consiste à établir un lien entre la base de données cible et un système EXPRESS capable d'interpréter directement un modèle EXPRESS. La seconde consiste à représenter l'aspect procédural au sein même de la base de données cible par des fonctions et des procédures stockées et activées par des vues et des triggers. Le stockage et la sécurisation de données sont assurés par le SGBD utilisé. Pour la sélection de données, nous avons présenté trois niveaux d'interrogation de données dans ce gestionnaire : le niveau d'instance où les requêtes sont effectuées directement dans le langage supporté par la base de données cible (requêtes classiques en SQL) ; le deuxième niveau concerne la sélection au niveau de concepts ou de l'ontologie qui suggère le développement d'un nouveau langage de requêtes étendant SQL pour permettre d'accéder simultanément aux valeurs des données et à leurs modèles conceptuels ; le troisième niveau, dans lequel un attribut (le paramètre de contexte) est utilisé pour calculer de façon dynamique les instances correspondant à une requête. Ce dernier niveau vise à offrir à l'utilisateur, après implémentation, les mêmes possibilités de sélection des composants que celles offertes par le modèle implicite lui-même.

Nous avons vu dans ce chapitre les difficultés que pose l'implémentation du modèle implicite et nous avons présenté des solutions pour y faire face. Une conversion des données de composants représentées d'une manière implicite en données représentées d'une manière explicite était alors nécessaire avant toute implémentation. Ainsi, à chaque importation de données PLIB vers la base de données cible, il faut faire cette conversion. La représentation implicite des composants permet une représentation compacte des instances. Cependant, cette représentation est très complexe et elle rend l'implémentation du modèle PLIB très coûteuse (à cause du processus de conversion). Si l'utilisation de cette représentation est souhaitable dans certains domaines, elle ne l'est pas pour tous. En effet, dans certains domaines d'application, les comportements de composants sont très peu ou pas représentés. Et, même s'ils sont représentés, les valeurs des paramètres de contexte se réduisent à un contexte de mesure. C'est le cas, par exemple, du domaine de l'électronique. Dans ce domaine, les paramètres de contexte correspondent aux conditions dans lesquelles une propriété donnée a été mesurée. Chaque paramètre ne possède donc qu'une valeur unique. Dans ces cas, les avantages de la représentation implicite sont faibles par rapport à la complexité de cette même représentation. De ce fait, continuer à représenter les données de cette manière ne fait que ralentir les processus de transfert de données du modèle PLIB vers un modèle de base de données cible en raison des conversions appliquées sur les données représentées d'une manière implicite. Afin d'éviter la complexité de cette représentation, une nouvelle approche de représentation de données de composants a été développée. Il s'agit de la représentation explicite des contenus des bibliothèques de composants. Elle consiste à représenter les composants dans le fichier d'échange en procédant à une énumération exhaustive de toutes les instances dans le modèle d'échange comme nous avons proposé jusqu'ici de le faire dans la base de données. Dans cette représentation, aucune expression, contrainte ou méthode de calcul n'est représentée. Toutes les instances sont représentées par des listes comprenant leurs valeurs. Cette représentation a été le fruit de la réflexion menée lors de l'implémentation de la représentation implicite (en particulier la conversion de données). Nous étudions dans le chapitre suivant le modèle de données permettant d'échanger directement des composants selon la représentation explicite.

Chapitre 4

Représentation explicite des données de composants : modèle et implémentation

Résumé.
Nous présentons dans ce chapitre une autre approche de représentation de données de composants : la représentation explicite. C'est la complexité de la représentation implicite et la nécessité de convertir les données de composants conforme à cette représentation avant de pouvoir les implémenter dans une base de données cible qui nous ont amené à adopter cette nouvelle représentation (explicite). Nous présentons d'abord le modèle permettant la représentation explicite. Les données de composants seront désormais échangées directement dans ce modèle. Puis elles devront être enregistrées dans la base de données cible. Nous discutons ensuite les différentes approches étudiées pour implémenter ce modèle de données explicite dans une base de données cible (relationnelle ou relationnelle-objet). Il s'agit de méthodes de génération du code SQL correspondant au modèle EXPRESS de PLIB et ses instances.

1 Introduction

Nous présentons dans ce chapitre la représentation explicite de composants. Celle-ci permet une représentation simple des instances. En fait, chaque instance est représentée par une énumération de couples (propriété, valeur). En ce qui concerne les propriétés dépendant du contexte (représentées par des fonctions dans le modèle implicite), nous avons fait le choix de représenter une seule valeur pour chaque paramètre de contexte. Ainsi, on ne modélise que le contexte de mesure d'une propriété (par exemple : la résistance électrique d'un composant électronique mesurée à 25 °C) et non pas la possibilité de sélectionner des composants en fonction du contexte d'insertion. Il faut noter que les descriptions des concepts (description au niveau de l'ontologie) sont les même pour les deux représentations (représentation implicite et représentation explicite des contenus). Il s'agit de deux approches différentes de représentation des données de composants décrites par un même dictionnaire de données (par une même ontologie).

Nous présentons également dans ce chapitre deux méthodes d'implémentation possibles. Il s'agit, en fait, de méthodes de génération de code SQL correspondant au modèle PLIB. La première est la méthode de génération de code utilisant l'analyse syntaxique basée sur les grammaires hors contexte. La seconde est la méthode de génération de code utilisant la programmation événementielle appliquée au langage

EXPRESS (attributs dérivés) basée sur les grammaires attribuées. Ces méthodes sont analogues à celles évaluée au chapitre précédent pour évaluer les expressions.

2 Nouvelle approche de représentation des données de composants : la représentation explicite

2.1 Représentation des propriétés dans le modèle explicite

Contrairement au modèle implicite, dans le modèle explicite aucune opération, contrainte, expression, méthode n'est modélisée, ni représentée.

Toutes les propriétés des classes sont associées à des valeurs explicites. Même si les valeurs des propriétés dépendant du contexte sont définies par une fonction, celles-ci doivent être énumérées de façon explicite en associant chaque valeur de telle propriété à la valeur de chaque paramètre de contexte dont elle dépend. Cette énumération se fait en se basant sur le choix d'un seul point du graphe représentant cette fonction. Par exemple, les valeurs d'une propriété "longueur" d'un ressort sont aussi nombreuses que les valeurs des charges exercées sur celui-ci dans des contextes d'utilisation différents. On peut alors choisir de représenter, par exemple, la longueur du ressort quand il ne subit aucune charge. Ceci donne seulement un contexte de mesure d'une valeur. Cette manière de représenter les propriétés dépendant du contexte est d'ailleurs très utilisée dans le domaine de l'électronique.

Le choix de représentation des propriétés dans le modèle explicite est complètement différent de celui effectué dans le modèle implicite (propriétés libres, propriétés dérivées, ….). Dans ce dernier, le souci de représenter les données dans une structure favorisant la sélection des composants par paramètre de contexte était fortement pris en compte. En revanche, dans le modèle explicite la sélection de composants par paramètres n'est pas possible, mais toutes les propriétés deviennent sélectionnables car leurs valeurs sont toutes représentées explicitement. Une seule caractéristique des propriétés est représentée dans le modèle d'extension de classe : il s'agit des propriétés d'identification qui constituent la clé identifiant chaque composant d'une famille en tant qu'article.

2.2 Dépendance fonctionnelle entre différentes propriétés d'une classe

Nous avons vu qu'un sous ensemble de propriétés représentant une classe sont définies comme étant des propriétés d'identification (voir le paragraphe 2.3 du chapitre 2). Le reste des propriétés dépendent fonctionnellement des propriétés d'identification. Cette dépendance fonctionnelle est représentée explicitement dans le modèle. Pour chaque instance toutes les valeurs des propriétés qui la décrivent sont fournies y compris celles des propriétés qui ne font pas partie des propriétés d'identification. Ainsi, cette dépendance fonctionnelle est représentée de la même façon que dans les SGBD (relationnels ou relationnels-objet). Cette notion de clé permet, lors d'une mise à jour de la base de données (nouvelle version d'un catalogue, par exemple), de reconnaître les instances qui sont nouvelles et celles qui existaient déjà, même si les propriétés qui les décrivent ne sont pas exactement les mêmes.

2.3 Extension de classe en représentation explicite

Dans la représentation explicite, les instances d'une famille de composants sont décrites d'une manière exhaustive. Pour chaque famille de composants, l'ensemble de ses instances est regroupé dans une structure unique. Cette structure modélise l'extension de classe et est représentée sur la Figure 33 par l'entité *Explicit_parts_family_extension*. L'extension de chaque famille de composants (classe) est une énumération de toutes les instances licites. Cette représentation est très proche de celle utilisée dans les SGBD relationnels ou relationnels objets pour décrire les extensions des différentes relations. Effectivement, dans ces SGBD, les extensions sont décrites dans des tables relationnelles où chaque instance est représentée par un n-uplet.

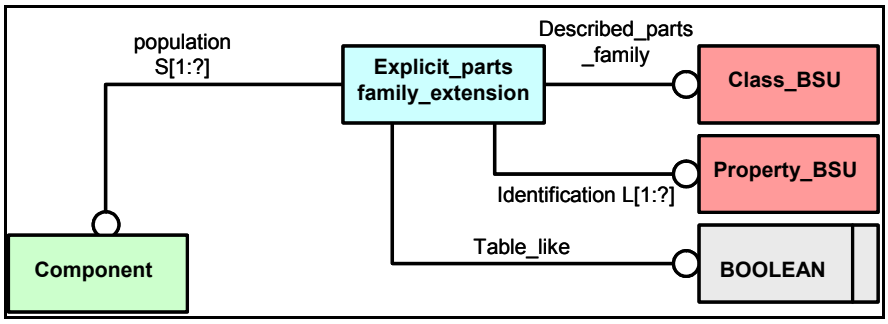


Figure 33 : représentation explicite d'une famille de composants

Chaque extension de classe référence sa classe (la description ontologique de cette classe) via son identifiant universel défini au niveau de l'ontologie (référence du BSU via l'attribut *Describe_parts_family*). L'ensemble des propriétés d'identification est répertorié par l'attribut *Identification*. Ces propriétés d'identification permettent l'identification non ambiguë de chaque instance. La population effective d'une famille de composants, qui est un ensemble d'instances (ou de composants), est représentée par l'attribut *Population*. L'attribut *Table_like* permet de préciser si toutes les instances sont décrites par les même propriétés et dans le même ordre (structure n-uplet d'une table ; c'est-à-dire que ces instances peuvent être représentées dans une table) ou si chaque instance est représentée par un certain nombre de propriétés, différemment des autres instances (structure hétérogène pouvant difficilement être représentée dans une table sauf à y mettre un grand nombre de valeur **NULL**).

2.4 Description d'une instance dans le modèle explicite

La représentation explicite des instances présente les avantages suivants :

1- faciliter la sélection des composants (des instances) en évitant les calculs : le calcul (ou la reconstitution) des instances à partir des expressions, des contraintes et des règles de jointures représentants ces instances.

2- simplifier le processus d'échange de données de composants entre des bases de données hétérogènes (car le processus de conversion des données représentées en mode implicite en données représentées en mode explicite n'est plus nécessaire),

3- réduire la complexité d'implémentation du modèle PLIB dans une base de données cible. Ceci passe, d'une part, par l'élimination de l'étape de conversion de données, et, d'autre part par la non représentation des expressions (représentant essentiellement les contraintes fonctionnelles). En effet, les bases de données ne sont pas très adaptées à la représentation de ce type information (i.e., la connaissance procédurale) [Ait-Ameur et al 03], [Challal 03].

La Figure 34 illustre la représentation d'instances dans le modèle explicite.

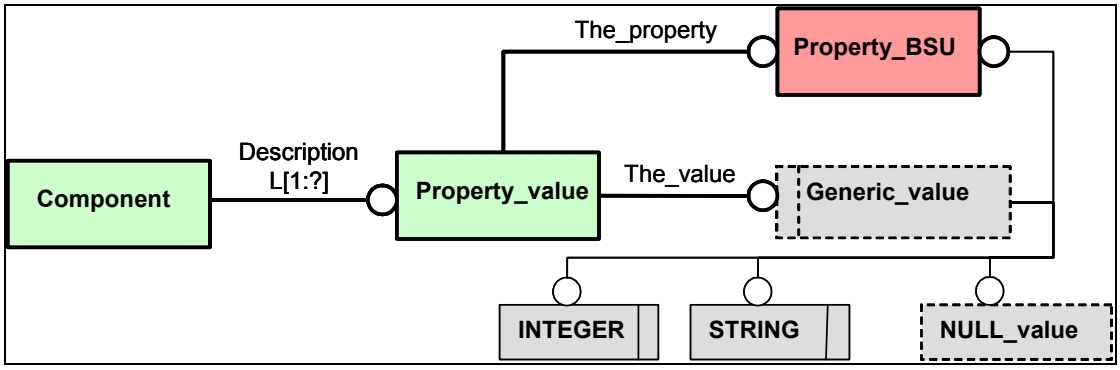


Figure 34 : représentation d'une instance dans le modèle explicite

Dans cette figure, une instance (ou un composant), représentée par l'entité *Component,* est décrite par une liste de couples *Property_value.* Chaque instance de *Property_value* possède une valeur et référence l'identifiant universel (*Property_BSU*) de la propriété associée à cette valeur. Ainsi, l'ordre dans lequel les propriétés sont représentées au niveau du contenu n'a aucune importance, puisque chaque couple représente à la fois la propriété (par une référence au *Properrty_BSU*) et sa valeur. Ceci offre une structure générique pour représenter les instances indépendamment de leurs modèles de définition. Elle peut devenir ainsi un support adapté à l'échange de données entre systèmes hétérogènes.

2.5 Association de valeurs aux propriétés

Nous avons indiqué ci-dessus que chaque instance est représentée par une liste de couples (propriété, valeur). La représentation d'une telle information permet d'associer la valeur d'une propriété à la propriété elle-même au sein du même fragment d'information dans le modèle d'échange. L'un des concepts importants du langage EXPRESS permettant la modélisation de ce type d'information est l'union de types représentée par le type *Select.* En effet, ce type permet de représenter un attribut d'une entité dont la valeur peut appartenir à un type parmi différents types. Le type de cet attribut est donc déclaré comme étant une union de types. A l'instanciation, la valeur de l'attribut peut appartenir à un type parmi les types de cette union.

Dans la Figure 34, le type *generic_value* est un type *select* qui est une union de types de valeurs que peut prendre une propriété d'une instance. Dans ce modèle (simplifié) une propriété peut prendre la valeur **NULL** (*Null_value*), une valeur entière (*Integer*), une chaîne de caractères (*String)* ou encore une autre entité (composition). Ce mode de typage permet d'avoir une structure unique pour représenter toutes les instances possibles des classes décrites dans le modèle.

La particularité du domaine représenté (les catalogues de composants) a nécessité une catégorisation particulière des propriétés ne se trouvant pas dans les modèles orientés-objets habituels. Il s'agit des propriétés applicables, visibles et représentées. Nous précisons que si l'ontologie représente toutes les propriétés possibles (visibles et applicables) pour chaque famille de composants, seule une partie de ces propriétés est utilisée réellement pour décrire un catalogue particulier : les propriétés représentées (i.e. associées à des valeurs). En effet, ce sont ces propriétés qui auront des valeurs effectives dans le modèle d'échange (associées à des valeurs). Les propriétés applicables ont le même statut que les propriétés représentées dans des modèles orientés-objets habituels (chaque propriété définie à une racine est applicable à toutes les sous-classes). Une propriété déclarée visible pour une classe à un certain niveau de la hiérarchie n'est pas forcément applicable à toutes ses sous-classes. Elle est en effet visible pour toutes les sous-classes mais applicable seulement pour une partie d'entre elles. Les propriétés représentées constituent un sous-ensemble de propriétés applicables (représentées au niveau de l'ontologie) choisies par un utilisateur particulier pour décrire réellement des composants par des valeurs dans son système. Par exemple, un utilisateur peut choisir de ne pas représenter la masse d'un composant électronique même si cette propriété fait partie intégrante des propriétés applicables à une famille de composants au niveau de l'ontologie.

2.6 Références classe-instance

Comme pour la représentation implicite, les extensions de classes référencent leur description au niveau de l'ontologie. Ainsi, dans le modèle PLIB les relations sémantiques se font par l'intermédiaire des identifiants universels (les BSU). La relation entre l'extension d'une classe et sa description au niveau de l'ontologie garantit la conformité de cette extension par rapport à sa description au niveau du dictionnaire. Pour chaque classe, les instances sont regroupées dans une même structure (représentée par l'entité *Explicit_parts_family_extension* sur la Figure 33) mais décrites chacune indépendamment. Le mécanisme de référencement permet à la fois de représenter le contenu (les instances) séparément du modèle qui les représente (l'ontologie) ou de les représenter conjointement. Ceci permet, en particulier, un échange du contenu sans avoir besoin d'échanger l'ontologie si les deux systèmes implémentent déjà la même ontologie.

2.7 Représentation des paramètres de contexte dans le modèle explicite

Nous avons précisé lors de la discussion sur la conversion de données implicites que les propriétés dépendant du contexte n'étaient pas prises en considération dans cette conversion car leur domaine de valeurs pouvait être continu, donc infini. Dans la représentation explicite, on a fait le choix de représenter les paramètres de contexte et les propriétés dépendantes du contexte. Cependant, chaque paramètre de contexte doit être associé à une seule valeur. On distingue deux types de domaines pouvant être représentés par PLIB :

- les domaines où les comportements de composants sont décrits par des paramètres de contexte dont le domaine de valeurs est continu. Par exemple, dans le domaine de la mécanique : la longueur d'un ressort dépend de la force (traction, compression) qu'il subit,

- les domaines où les paramètres de contexte sont seulement utilisés pour préciser le contexte d'une mesure de valeur d'une propriété. Par exemple, dans le domaine de l'électronique : la résistance d'un transistor dépend de la température de l'environnement, on fournit seulement cette valeur à une température de 25 °C.

La représentation du deuxième cas ne présente aucun problème, en revanche pour le premier cas (domaine continu) deux choix se présentent :

- choisir une valeur représentative du domaine continu et la représenter ainsi que la valeur de la propriété dépendant du contexte qui lui correspond (exemple : résistance mesurée à 25 °C, longueur du ressort quand il ne subit aucune force),

- choisir la représentation implicite (par des expressions) dans le cas où on veut avoir plus une description complète de fonctions de dépendance (voir chapitre 3).

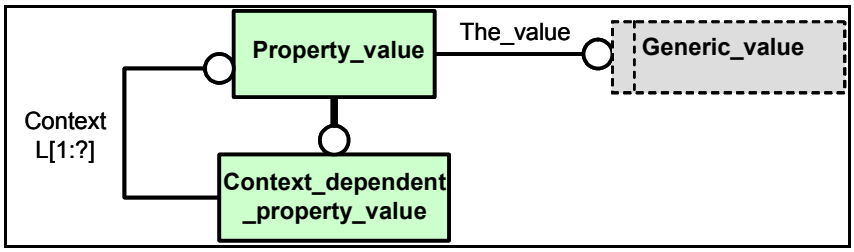


Figure 35 : représentation des comportement des composants dans des contextes d'insertion dans le modèle explicite.

La Figure 35 montre la partie du modèle explicite permettant de représenter les propriétés dépendantes du contexte ainsi que les paramètres de contexte dont elles dépendent. Les propriétés dépendantes du contexte sont représentées par l'entité *Context_dependent_property_value* (une sous-classe de l'entité *Property_value*). Le paramètre du contexte est représenté par une instance de l'entité *Property_value* comme n'importe quelle propriété. La propriété dépendante de contexte est liée au(x) paramètre(s) de contexte dont elle dépend par l'attribut *Context* de l'entité *Context_dependent_property_value*. De cette manière chaque valeur d'une propriété dépendant du contexte est liée directement à la valeur du ou des paramètre(s) de contexte dont elle dépend même si, en fait, cette dépendance devrait être représentée par une fonction (on ne représente qu'un point du graphe de cette fonction).

2.8 Représentation des agrégats : nécessité des agrégats dans le modèle explicite

Dans les bases de données relationnelles, les valeurs des attributs des relations doivent être des valeurs atomiques (principe de 1ère forme normale). Avec l'arrivée du modèle objet, la représentation des propriétés de classes ayant pour valeurs un agrégat de valeurs (atomiques ou d'agrégats) est devenue courante. Le modèle de données PLIB offre cette possibilité. Comme pour la représentation des instances ayants des valeurs atomiques, le modèle du contenu de PLIB (modèle des instances) définit un schéma permettant la représentation et l'échange des agrégats (collections de valeurs).

Dans ce paragraphe, nous présentons la partie du modèle explicite permettant la représentation d'agrégats. Dans ce modèle, les agrégats sont modélisés conformément à la définition des types de données des agrégats dans le langage EXPRESS. Les quatre types d'agrégats du langage EXPRESS (**LIST, SET, BAG ET ARRAY**) sont représentés dans le modèle des agrégats. Nous précisons la particularité de chaque type d'agrégat dans le langage EXPRESS [ISO 10303-11:2000] :

- type **SET** : ou "ensemble" permet le regroupement non ordonné et sans duplication d'éléments d'un même type. Il correspond à la notion d'ensemble dans le sens mathématique du terme.

- type **BAG** : ou "sac" permet le regroupement non ordonné et avec duplication possible d'éléments d'un même type. Il s'agit d'une généralisation du type **SET**.

- type **ARRAY** : ou "tableau" permet de spécifier des tableaux d'éléments homogènes caractérisés par le même type (ressemble aux tableaux utilisés dans les langages de programmation). Contrairement aux autres types d'agrégats, dans le type **ARRAY** les éléments sont caractérisés par un index de type entier et par une dimension (définie à l'aide de borne *inférieure* et *supérieure*) qui restent inchangé pendant toute la durée de vie d'un tableau. Un tableau peut être creux, c'est-à-dire avoir des cases dont les valeurs sont indéterminées (représentée par "?" (équivalent à la valeur **NULL** dans les bases de données)).

- type **LIST** *:* ou "liste" permet le regroupement ordonné d'éléments d'un même type pour lesquels l'emplacement est important. Contrairement au type **ARRAY** la liste n'admet pas d'éléments optionnels (ne peut pas être creuse). Par ailleurs, les bornes de la liste n'indiquent pas, comme dans le tableau, la première et la dernière position des éléments de la liste, mais le nombre maximum et minimum des valeurs qu'elle peut comporter.

Dans le modèle de données PLIB les agrégats sont modélisés à deux niveaux : au niveau du dictionnaire et au niveau du contenu. Au niveau du dictionnaire de données, un type d'agrégat a été défini pour permettre d'associer une propriété d'une classe à ce type. A ce niveau il s'agit seulement de définir d'une manière globale le domaine d'une propriété (son type seulement). L'extension apportée au dictionnaire de données pour représenter ces types (i.e. les agrégats) est donnée Figure 36.

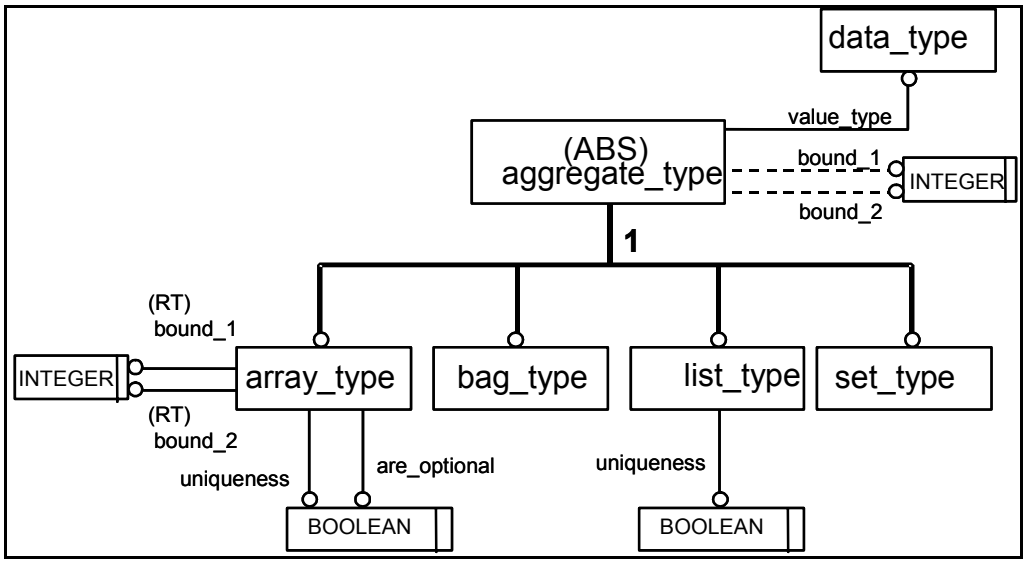


Figure 36 : représentation des types d'agrégats au niveau du dictionnaire [ISO 13584-25:2004]

Au niveau du contenu une structure permettant l'énumération des valeurs des propriétés de type agrégats a été définie dans le modèle explicite de PLIB. De cette manière, toutes les instances possibles d'un modèle à objet peuvent être représentées. La Figure 37 présente le modèle formel permettant de représenter les valeurs de type agrégat.

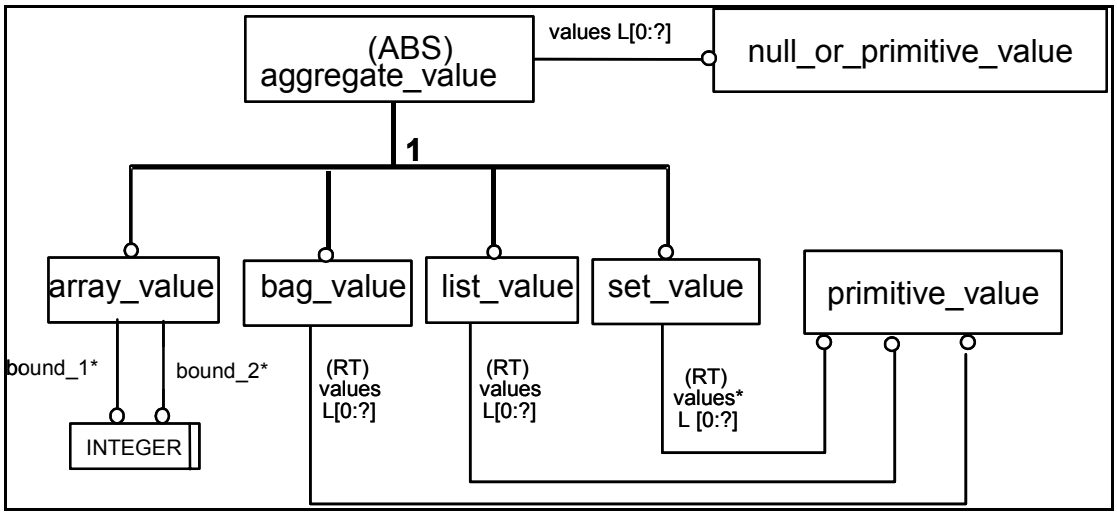


Figure 37 : représentation des valeurs des agrégats au niveau du contenu explicite [ISO 13584-25:2004]

3 Implémentation du modèle explicite de PLIB dans une base de données cible

L'implémentation du modèle explicite de PLIB dans une base de données cible est réalisée en suivant l'approche hybride. Cette approche est celle qui a été retenue parmi les différentes approches possibles permettant l'implémentation du modèle PLIB dans une base de données cible (voir paragraphe 3.4 du chapitre 2). De plus, le

développement de la représentation explicite était motivé par la simplification de l'implémentation du modèle PLIB en suivant justement cette approche hybride. Le modèle de données PLIB sera représenté à deux niveaux différents : le niveau de l'ontologie et le niveau du contenu. Comme pour la représentation implicite, la représentation du dictionnaire reste la même que pour la représentation implicite. Nous présentons dans ce paragraphe les différents aspects de l'implémentation du contenu.

Nous discutons dans cette paragraphe de l'aspect théorique de l'intégration du modèle de données PLIB dans un modèle d'une base de données cible. Cet aspect théorique comprend essentiellement les approches d'automatisation étudiées pour l'intégration des deux modèles représentant respectivement l'ontologie et le contenu. L'aspect pratique sera détaillé dans le chapitre 5 (mise en œuvre de l'implémentation).

3.1 Les différentes étapes d'implémentation

L'implémentation du modèle explicite de PLIB suit les mêmes étapes que celles de l'implémentation du modèle implicite (cf. paragraphe 3.1 du chapitre 3). Ces étapes sont les suivantes : i) la création du schéma de la base de données du niveau de l'ontologie, ii) le peuplement de ce schéma par une ou plusieurs descriptions d'ontologies, iii) la création du schéma de contenu conforme à la description des concepts au niveau de l'ontologie, iv) et finalement, le peuplement du schéma de contenu par des instances.

3.2 Intégration automatique d'un modèle EXPRESS dans une base de données cible

L'intégration automatique des instances d'un modèle EXPRESS dans une base de données cible sous-entend la génération automatique du code correspondant au modèle EXPRESS, dans le langage supporté par la base de données cible. Cette génération de code concerne deux niveaux différents : la génération du schéma de la base de données cible (commandes de LDD (Langage de Définition de Données) de SQL) et la génération du stockage des instances (commandes de LMD (Langage de Manipulation de Données) du SQL). Plusieurs approches sont possibles pour procéder à la génération du code. Nous en avons étudié deux : l'une est basée sur les grammaires hors contexte et l'autre sur les grammaires attribuées. Nous étudions d'abord le cas de la génération de schémas et de leurs instances pour un modèle de données représenté directement dans le langage EXPRESS. Le second cas étudié concernera des modèles de données issus de méta-modèles.

3.2.1 Génération de code par analyse syntaxique basée sur une grammaire hors contexte

Cette première approche de génération de code s'inspire de la compilation des langages définis par une grammaire hors contexte. Généralement, la compilation des programmes aboutit à la génération de code représentant ces programmes. Cette génération de code passe par les étapes suivantes [Aho et al 72], [Aho et al 86] :

- analyse lexicale,
- analyse syntaxique,
- analyse sémantique,
- production de code.

L'analyse lexicale permet de décomposer le programme en unités lexicales (transformer le programme en une suite de mots). Après cette étape vient l'analyse syntaxique qui permet de construire l'arbre de syntaxe du programme. Cette analyse syntaxique se base sur la grammaire hors contexte, qui définit le langage, pour construire l'arbre syntaxique. Puis, une fois cet arbre construit, son analyse sémantique permet d'éliminer toutes les erreurs d'ordre sémantique (incompatibilité de types, par exemple). Finalement la génération du code correspondant au programme compilé est réalisée en parcourant l'arbre de syntaxe.

Nous nous sommes basé sur cette approche pour générer automatiquement le code SQL correspondant à l'implémentation du modèle de données EXPRESS (dans notre cas le modèle PLIB) dans une base de données cible, puis la représentation des instances dans cette base de données cible. Cette génération est réalisée à partir du modèle EXPRESS de PLIB et des fichiers physiques représentant les instances de ce modèle. Le point commun entre la méthode que nous utilisons pour générer du code SQL et la compilation des langages définis par une grammaire hors contexte réside au niveau de la représentation des données et des programmes. Lors d'une compilation d'un programme, on construit un arbre de syntaxe le représentant. De la même façon les données représentant les instances du modèle (représentées dans un fichier physique en format texte) suivent les mêmes étapes pour aboutir à un arbre d'instances. A la différence de la méthode de compilation de langages, la construction de l'arbre d'instances se base conjointement sur une grammaire hors contexte et sur le modèle de données lui-même. Tout d'abord, la compilation du schéma EXPRESS permet de construire un arbre de syntaxe du modèle. Elle permet d'une part de vérifier l'aspect sémantique de modèle et d'autre part de faciliter l'accès aux informations concernant le modèle lui-même (nom des entités, leurs attributs, …) en parcourant l'arbre de syntaxe construit lors de la compilation. C'est cet accès aux modèles (via l'arbre de syntaxe) qui permet ensuite la fabrication de l'arbre d'instances.

L'exploitation des deux arbres (l'arbre de syntaxe du modèle EXPRESS et l'arbre d'instances) permet la génération du code SQL. Cette génération est réalisée à l'aide d'un programme générique qui parcourt les deux arbres. Le parcours de l'arbre de syntaxe du modèle permet la génération des commandes du LDD de SQL permettant la création du schéma de la base de données correspondant au modèle de données EXPRESS. Le parcours de l'arbre des instances permet la génération des commandes de LMD de SQL permettant le peuplement de la base de données dont le schéma a été créé dans l'étape 1.

Pour mettre en œuvre cette méthode nous avons utilisé un outil permettant de manipuler des modèles EXPRESS et de lire des fichiers physiques (outil ECCO Toolkit). En effet, par les procédures de compilation citées ci-dessus cet outil construit

l'arbre du modèle et l'arbre d'instances. Le programme générique de génération de code que nous avons développé accède à ces deux arbres (pour pouvoir les parcourir) via une API (Application Programming Interface).

3.2.2 Génération de code par programmation événementielle appliquée au langage EXPRESS

La programmation événementielle est une méthode de programmation qui consiste à produire des programmes qui réagissent à des événements particuliers. Chaque évènement est suivi d'une action précise du programme si sa condition de déclenchement est remplie. Les évènements peuvent être déclenchés soit par un utilisateur soit par d'autres programmes. Cette technique de programmation est utilisée en programmation graphique, dans les IHM ou dans les programmes de contrôle de procédés par exemple.

L'application de cette technique de programmation au langage EXPRESS est d'une nature différente. Dans le cas du langage EXPRESS cette programmation n'a rien à voir avec la gestion ou la génération des interfaces graphiques. En fait, elle est basée sur le principe de la grammaire attribuée. Une grammaire attribuée est connue comme étant une méthode déclarative et structurée de spécification d'un calcul sur une structure arborescente [Deransart et al 88a], [Deransart et al 88b], [Engelfriet 84]. Les calculs sont décomposés sur chaque composante (constructeur) de la structure. Ce calcul est réalisé en enrichissant chacune de ces composantes d'une spécification locale du calcul à l'aide d'un ou plusieurs attributs appelés règles sémantiques.

L'application de cette technique sur le langage EXPRESS consiste à ajouter à chaque nœud du modèle EXPRESS (chaque entité) un attribut dérivé permettant de générer le code représentant les instances de cette entité dans le langage de la base de données cible. Cette approche utilise les attributs dérivés du langage EXPRESS pour générer des programmes. Chaque attribut dérivé défini contient un fragment du programme global. Ce fragment est constitué à partir des informations que portent les instances (à partir des valeurs d'autres attributs). La simple consultation de ces attributs (considérée comme événement) et l'assemblage des différentes parties (les valeurs des attributs dérivés) permettent la génération d'un ou de plusieurs programmes [Ait-Ameur et al 00]. La présence des instances du modèle (enrichi par les attributs dérivés) constitue une condition pour la poursuite des actions de la génération.

Nous avons utilisé cette approche de programmation pour générer du code SQL permettant de réaliser l'intégration des instances d'un modèle EXPRESS dans une base de données cible. L'application de cette méthode sur les modèles de données EXPRESS n'altère en rien les fichiers physiques. En effet, les valeurs des attributs dérivés n'apparaissent pas dans ces fichiers. Les instances du modèle d'origine peuvent être lues également comme des instances du modèle enrichi par des attributs dérivés. Une fois que ces instances sont lues comme étant des instances du modèle enrichi, la génération de code devient une simple action de consultation d'une valeur d'un attribut dans le modèle.

La Figure 38 donne un exemple simple montrant la génération du code SQL à partir d'un schéma EXPRESS simple.

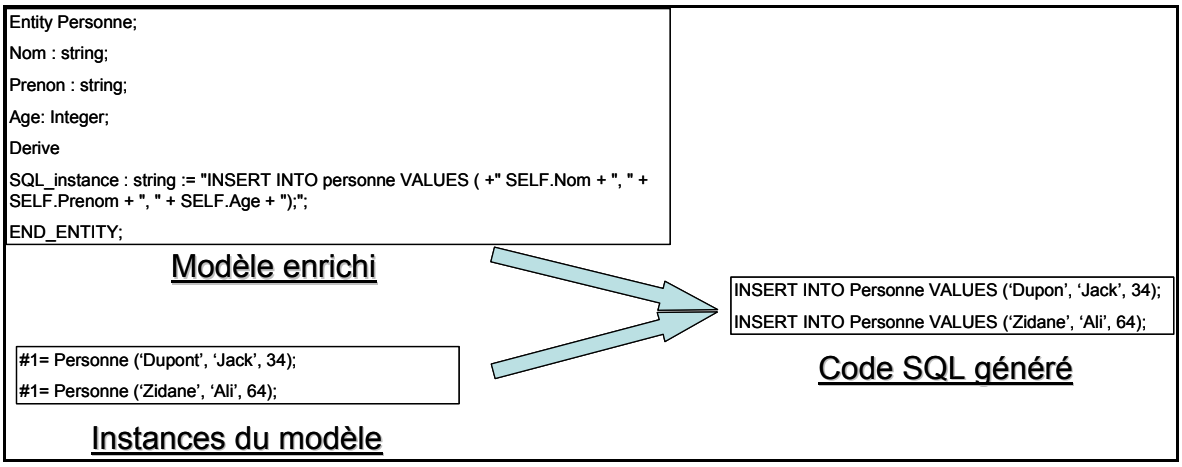


Figure 38 : Génération de code SQL pour l'insertion des instances d'un modèle EXPRESS dans une base de données cible.

Cette figure présente une partie d'un modèle EXPRESS qui a été enrichi par des attributs dérivés. L'attribut dérivé *SQL_instance* a été rajouté à l'entité *personne*. Cet attribut permet la génération des commandes SQL permettant l'insertion des instances de l'entité *personne* au niveau de la base de données. Le code généré est constitué de la concaténation des valeurs des attributs dérivés calculées pour chaque instance du modèle. Ainsi, pour les deux instances de la Figure 38, deux commandes SQL ont été générées.

3.3 Comparaison des deux approches de génération de code

Nous avons présenté ci-dessus deux approches permettant la génération automatique de code pour une base de données cible pour réaliser l'intégration des données PLIB (représentées par des modèles EXPRESS) dans cette base de données. La première approche, basée sur une grammaire hors contexte, présente les avantages suivants :

- elle ne nécessite pas de changement au niveau des modèles de données,
- une fois le programme de compilation (voir paragraphe 3.2.1 ci-dessus) mis en place la génération du code est simple.

Cette approche présente également des inconvénients :

- elle nécessite la création d'un programme pour chaque modèle de données,
- elle peut être très coûteuse en terme d'espace dès lors que les fichiers d'instances deviennent gros,

La seconde approche développée présente également des avantages et des inconvénients. Parmi les avantages nous citons :

- elle ne nécessite aucun programme global,

- elle est très efficace en espace car le calcul est réparti sur chaque entité et s'effectue de façon "paresseuse", c'est-à-dire lorsque l'instance est lue,

- elle est simple à mettre en œuvre et les modèles peuvent être enrichis par des attributs dérivés dès leur conception,

- elle permet d'intégrer dans le modèle lui-même son implémentation dans différents systèmes,

- on ne change pas de langage et de technologie car tout est fait avec EXPRESS.

Parmi les inconvénients nous citons :

- la nécessité de modifier les modèles de données (en ajoutant des attributs dérivés),

- l'impossibilité de redéfinir un attribut dérivé lors d'une spécialisation d'entité.

3.4 Conclusion : approche choisie

Parmi les deux approches de génération de code présentées ci-dessus, nous avons choisi d'utiliser l'approche de la programmation événementielle appliquée au langage EXPRESS. Le choix de cette approche est motivé d'une part par les avantages qu'elle apporte, et, d'autre part, par le souhait d'étudier de façon plus approfondie les capacités particulières du langage EXPRESS. Ces capacités consistent à offrir en plus de la modélisation de données, les moyens nécessaires et suffisants pour implémenter des modèles EXPRESS.

3.5 Application de la programmation événementielle à des modèles et des méta-modèles EXPRESS

Nous avons précisé ci-dessus que la programmation événementielle appliquée au langage EXPRESS est réalisée par l'enrichissement des modèles EXPRESS par des attributs dérivés lors de la lecture d'une instance. L'évènement qui déclenche l'action est la consultation de l'attribut dérivé. La condition de poursuite de cette action (qui est le calcul de la valeur de l'attribut dérivé) est la présence d'une instance. Les valeurs qui peuvent être utilisées pour calculer les attributs dérivés proviennent des autres attributs qui sont donc renseignés dans des fichiers physiques. Cette approche a permis la génération d'un code SQL pour intégrer la population du modèle de données EXPRESS au niveau de la base de données cible. En revanche, la création du schéma de la base de données cible n'est pas possible en utilisant la programmation événementielle appliquée directement au modèle de données lui-même. En effet, les informations concernant le modèle lui-même ou les méta-données (nom d'une entité ou de ses attributs, …) ne sont pas renseignés dans un fichier physique (i.e. fichier d'instances) représentant l'instanciation de ce même modèle. Le code généré à partir de

l'enrichissement d'un modèle ne peut concerner que les commandes de manipulation de données (LMD) permettant de créer les instances dans la base de données cible.

L'application de cette approche pour la création du schéma de la base de données nécessite le passage à un niveau supérieur, le niveau du méta modèle qui décrit le modèle de données lui-même. Le modèle de données est représenté dans un fichier physique comme une instanciation du méta modèle. Par conséquent, la génération des schémas de bases de données cibles peut être réalisée en utilisant la programmation évènementielle d'EXPRESS appliquée aux méta-modèles (i.e. enrichir le méta modèle par des attributs dérivés). La Figure 39 présente une partie du méta-modèle du formalisme EXPRESS enrichi par des attributs dérivés.

```
ENTITY entity_info
      Name: STRING; -- le nom de l'entité
      Attributes : LIST OF UNIQUE attribute;

Derive
       SQL_representation : STRING := "CREATE TABLE " + SELF.Name + "(" +
SQL_attr_representation (SELF.attributes) + ")"; -- la fonction SQL_attr_representation parcourt
la liste des attributs et renvoit une chaine de caractères composé les noms des attributs suivis de
leurs types SQL séparés par des virgules (attributes[1].name + ' ' + attribute[1].SQL_att_type +
', ' + attributes[2].name ….
END_ENTITY;

ENTITY  attribute
       Name      : STRING;     -- le nom de l'attribut
       Attr_type  : type_info;  -- le type de l'attribut

Derive
     SQL_att_type : STRING := att_typpe_function (SELF.attr_type) ; -- le type SQL
correspondant au type EXPRESS de l'attribut.
END_ENTITY;

ENTITY type_info
    Category : type_category;  -- la catégorie de base du type  (string_type, integer_type etc.)
END_ENTITY;
….
```

Figure 39 : le méta-modèle (ou méta-schéma) du formalisme EXPRESS enrichi par des attributs dérivés

Dans la Figure 39 l'entité *entity_info* représente les méta-données d'une entité telle qu'elle est représentée dans le langage EXPRESS. De la même façon, les entités *attribute* et *type_info* représentent respectivement les méta-donnés d'un attribut et d'un type tels qu'ils sont représentés dans le langage EXPRESS. Le méta-modèle de départ a été enrichi par des attributs dérivés permettant la génération des schémas de bases de données cibles correspondant à n'importe quel modèle EXPRESS (considéré comme une instanciation du méta-modèle). Ainsi l'attribut *sql_représentation* de l'entité *entity_info* permet de générer une commande SQL permettant de créer une table correspondant à une entité quelconque d'un modèle de données EXPRESS. La valeur de cet attribut dérivé est calculée à l'aide d'une expression permettant de concaténer des chaînes de caractères représentant les mots clés du langage SQL avec d'autres chaînes de caractères concernant les instances du méta-modèle récupérées à partir des valeurs des attributs (attribut *name* de l'entité *entity_info* sert à insérer le non la table dans la commande générée). La fonction *SQL_attr_representation* permet la génération des colonnes des tables représentant les instances du méta-modèle. Chaque valeur de

l'attribut ainsi obtenue correspond à une commande SQL permettant la création d'une table. L'exécution de l'ensemble des commandes générées permet la création d'un schéma de base de données correspondant au modèle conceptuel de données représenté comme une instanciation du méta-modèle.

Notons que la génération automatique du schéma, quelle que soit la méthode utilisée, présente un inconvénient majeur. Il n'est pas possible de tenir compte des spécificités du modèle pour optimiser son exploitation dans les SGBD cible (par exemple : éviter de trop grand nombre de jointures).

Nous détaillons dans le paragraphe suivant l'implémentation réalisée pour les deux générations nécessaires :

1- génération du schéma de la base de données (LDD),

2- génération du contenu de la base de données (LMD),

4 Détails de l'implémentation : application de la programmation événementielle au modèle PLIB

Nous avons proposé dans les paragraphes précédents deux approches permettant la génération de code SQL à partir de modèles EXPRESS. Nous avons présenté les avantages et les inconvénients de chacune d'entre elle. Nous avons choisi l'approche de la programmation événementielle pour l'appliquer sur le modèle PLIB. Rappelons que l'architecture globale de la base de données adoptée pour implémenter le modèle explicite de PLIB est constitué de trois niveaux différents : le niveau de la métabase, le niveau de l'ontologie et le niveau du contenu (voir la Figure 44). Nous discutons dans les paragraphes suivants des différentes facettes de l'application de cette approche sur le modèle PLIB. Plus précisément, nous détaillons les différentes étapes menant à la création de cette architecture. La métabase, représentant la partie 1 de la Figure 44, étant créé automatiquement par le SGBD, elle ne sera cependant pas traitée dans ce qui suit. Les deux autres niveaux de cette architecture (le niveau de l'ontologie et le niveau du contenu) seront chacun implémentés en deux étapes : la création du schéma de la base de données PLIB et son peuplement par les instances. Ces dernières représentent des modèles de catalogues de composants quand il s'agit du peuplement de l'ontologie (partie 2 de la Figure 44) et des instances de composants quand il s'agit du peuplement du contenu de PLIB (partie 3 de la Figure 44). Comme pour la représentation implicite, le niveau de l'ontologie est implémenté au niveau des entités du modèle EXPRESS.

4.1 Schéma relationnel du dictionnaire (ontologie)

Le schéma relationnel associé au dictionnaire est la traduction directe du modèle EXPRESS représentant le dictionnaire de données PLIB dans un modèle de la base de données cible (relationnelle ou relationnelle-objet). Le modèle du dictionnaire PLIB est représenté par le fascicule 42 de la norme PLIB (voir le chapitre 1). Nous avons d'abord étudié la création de ce schéma par une approche manuelle. Il s'agit d'une approche classique utilisée pour transformer les modèles conceptuels de données en

modèles relationnels ou relationnels-objet selon la base de données cible. La transformation du modèle de l'ontologie se base sur des règles de traduction d'un schéma EXPRESS dans un modèle relationnel ou relationnel objet, et ces règles sont ensuite appliquées manuellement. Cette méthode présente des avantages et des inconvénients. le principal de ces avantages est l'optimisation de certaines requêtes en évitant certaines opérations algébriques qui deviendraient nécessaires dans le cas d'une implémentation systématique du modèle. Cette optimisation de requêtes est le résultat d'une modification du modèle à l'occasion de l'application de règles. Elle consiste à représenter, dans certains cas, plusieurs entités par une même table. Prenons l'exemple de l'entité *class* qui décrit les familles de composants. Cette entité est décrite par un ensemble d'informations (les noms de la classe (longs, courts avec leur traduction éventuelle dans plusieurs langues), des descriptions textuelles pour donner plus d'informations sur la classe avec des traductions éventuelles dans plusieurs langues, ….) représentées par l'entité *Item_names* associée à l'entité *Class*. Dans ce cas, la modification du modèle consiste à remonter tous les attributs de l'entité *Item_names* pour les représenter directement dans la table représentant l'entité *class* dans la base de données cible. De cette manière, les deux entités *Class* et *Item_names* seront représentées par la seule table *class* dans le schéma de la base de données cible.

Cette transformation permet de simplifier le schéma de la base de données représentant le modèle de données exprimé en EXPRESS et, par conséquent, de simplifier sa manipulation et sa gestion dans la base de données cible. Effectivement, la modification du modèle ne peut être appliquée d'une manière générique ; c'est-à-dire qu'elle ne peut être généralisée sur toutes les entités qui sont associées l'une à l'autre. Elle ne peut donc être réalisée d'une manière automatique. Chaque cas est particulier, et seul un concepteur averti et qui connaît bien le modèle peut faire des choix de modification sur telle ou une telle entité.

Parmi les inconvénients de cette approche, mentionnons la nécessité d'une intervention humaine pour réaliser la traduction du modèle EXPRESS représentant le dictionnaire de PLIB. Ceci est d'autant plus délicat que le nombre d'entités représentées dans ce modèle est important. De plus chaque entité peut donner lieu à plusieurs traductions d'implémentation ce qui donne lieu à une démarche moins systématique. Compte tenu de la complexité du modèle, des erreurs de traduction peuvent apparaître. De plus, l'application de cette approche rend plus difficile toute automatisation des étapes suivantes à savoir le peuplement de la base de données créée ou la création, à partir de cette même base de données, des fichiers physiques correspondant aux modèles EXPRESS implémentés.

L'étude de l'approche manuelle a été motivée par deux raisons. La première est que le modèle de données représentant le dictionnaire PLIB est un modèle stable et qui ne varie que très peu dans le temps. Ainsi, une fois le schéma du dictionnaire créé la première fois il reste inchangé. La seconde raison est que l'utilisation de cette approche permet de simplifier le schéma de la base de données cible et par conséquent de simplifier la gestion des données. D'un point de vue scientifique, cette approche n'apporte pas de nouveaux résultats. Nous avons donc étudié la possibilité de réaliser l'intégration du dictionnaire PLIB dans une base de données cible d'une manière automatique. Cette intégration automatique est réalisée par l'application de la programmation événementielle au langage EXPRESS. Le paragraphe suivant la présente.

4.1.1 Création du schéma de l'ontologie dans la base données cible par programmation événementielle

Contrairement à d'autres langages de modélisation de données (en majorité des langages graphiques) le langage EXPRESS contient également une représentation textuelle du modèle facilitant ainsi son traitement d'une manière automatique. Cette particularité du langage EXPRESS nous a permis de traiter le cas du passage d'un modèle conceptuel (ici un modèle EXPRESS) à un schéma relationnel ou relationnel-objet d'une façon complètement automatique. Il faut noter que certains outils de conception de bases de données tels que *PowerAMC* [Sybase 04], *Rational Rose* [Noonan 03] et *Oracle Designer* [Kenneth 99] permettre d'enrichir les modèles graphiques par des informations sémantiques afin de permettre la génération automatique du modèle relationnel ou relationnel-objet selon le type de base de données cible visée.

Le modèle du dictionnaire PLIB constitue un modèle conceptuel de données. Ce modèle étant représenté directement dans le langage EXPRESS, la génération du modèle relationnel ou relationnel-objet équivalent dans la base de données cible peut se réaliser en appliquant la programmation événementielle (appliquée au langage EXPRESS). Ceci implique donc l'utilisation du méta modèle du formalisme EXPRESS enrichi par des attributs dérivés décrit dans le paragraphe 3.5.

Avant de procéder à la génération du code représentant le schéma du dictionnaire dans la base de données cible, une transformation du modèle EXPRESS (du dictionnaire) en un fichier physique (d'instances) du méta modèle EXPRESS est nécessaire. Le modèle du dictionnaire pourra alors être lu comme n'importe quel fichier, physique ce qui permettra de générer du code SQL permettant la création du schéma de l'ontologie au niveau de la base de données cible. Cette transformation du modèle peut être réalisée en utilisant une API d'accès aux méta-données fournies par un outil gérant les modèles EXPRESS (ECCO Toolkit, par exemple).

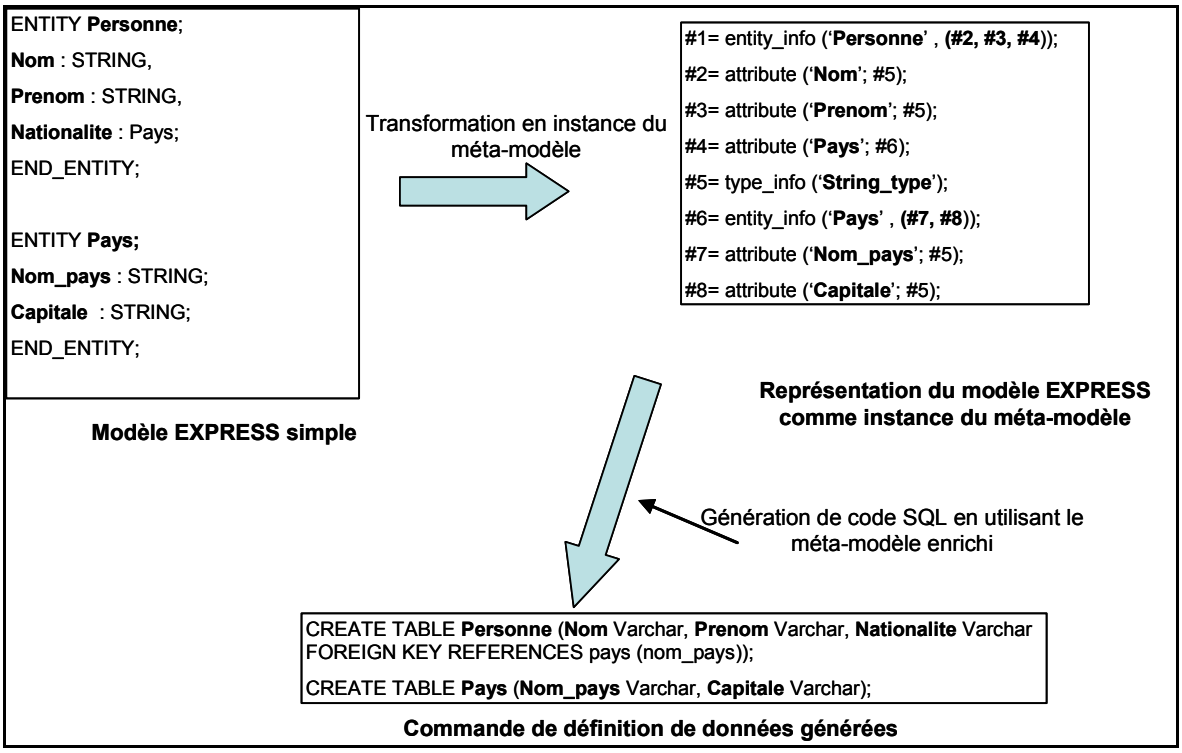


Figure 40 : Utilisation du méta-schéma EXPRESS enrichi pour la génération du code de définition de données correspondant au modèle EXPRESS simple

La Figure 40 montre un exemple simple de génération de schéma de base de données qui correspond à un modèle EXPRESS simple. Dans ce modèle deux entités sont représentées : *personne* et *pays*. La transformation de ce modèle en instances du méta-modèle (voir Figure 39) a donné le fichier physique à droite de la figure ci-dessus. La lecture de ce fichier par un outil EXPRESS qui connaît le méta-modèle EXPRESS enrichi par des attributs dérivés permet la génération du code SQL par une simple consultation des attributs dérivés. Ce code généré permet la définition du schéma de la base de données cible correspondant au modèle de données du départ (modèle représenté à gauche de la figure ci-dessus).

4.1.2 Peuplement du dictionnaire de données PLIB

Après la génération du schéma du dictionnaire dans la base de données cible vient le peuplement par une description ontologique particulière. Cette description ontologique constitue une instanciation du modèle du dictionnaire de données. Comme pour la génération du modèle relationnel ou relationnel objet correspondant au dictionnaire, le peuplement de la base de données est réalisé en appliquant également la programmation événementielle. Dans ce cas, c'est le modèle du dictionnaire qui doit être enrichi par des attributs dérivés. A chaque entité de ce modèle on ajoute un attribut dérivé permettant la génération d'une commande SQL pour insérer une instance de cette même entité dans la base de données cible. La première étape est de lire le fichier physique représentant l'ontologie avec le modèle enrichi. Puisque les valeurs des attributs dérivés n'apparaissent pas dans les fichiers physiques, cette lecture ne pose pas problème. Une fois cette lecture réalisée par un outil permettant la manipulation des données EXPRESS (Ecco Toolkit par exemple), la génération du

code contenant les commandes de manipulation de données du langage SQL (pour insérer les instances) devient une simple consultation des attributs dérivés ajoutés.

La Figure 41 montre une partie de la génération des commandes d'insertion de données (LMD) en utilisant la programmation évènementielle. Nous voyons sur cette figure trois parties distinctes : un modèle simplifié du dictionnaire de données enrichi (partie 1 de la Figure 41), une instanciation de ce modèle qui représente une partie d'un dictionnaire particulier (l'identification des concepts n'est pas représentée) (partie 2 de la Figure 41) et le code généré à partir des deux parties précédentes (partie 3 de la Figure 41).

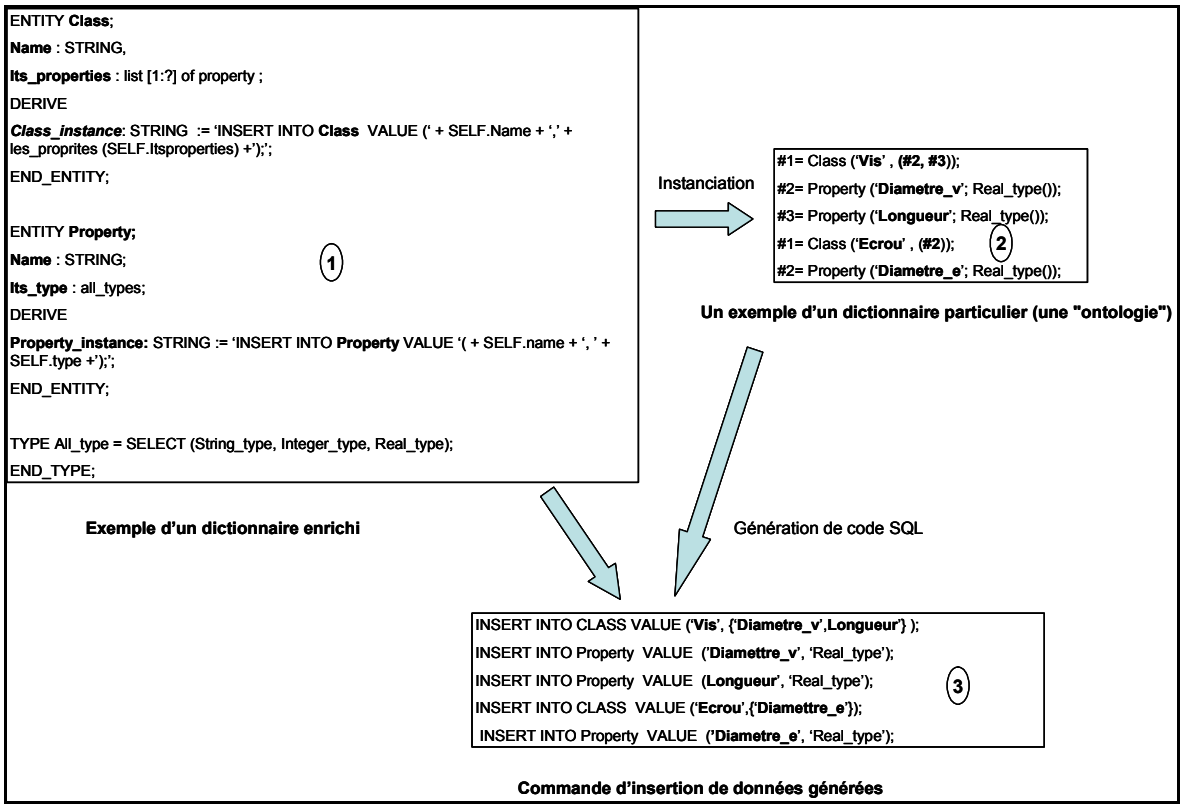


Figure 41 : génération de code LMD de SQL pour l'insertion d'une ontologie au niveau de la base de données cible.

Le fichier physique est tou d'abord lu comme étant une instanciation du modèle enrichi. Pour chaque instance lue une commande d'insertion de données est générée en consultant la valeur de l'attribut dérivé de cette instance. Le regroupement de ces différents fragments du code généré constitue un programme SQL permettant l'intégration automatique de toutes les instances représentées dans le fichier physique au sein de la base de données cible.

4.1.3 Schéma relationnel ou relationnel-objet du contenu

Le schéma du contenu est le schéma qui permet le stockage des données effectives des composants. Le modèle conceptuel de ce schéma est représenté comme étant une instanciation du modèle du dictionnaire PLIB. La génération automatique d'un tel schéma est réalisée par programmation événementielle en langage EXPRESS. Dans ce

cas c'est le modèle du dictionnaire de données qui doit être enrichi par des attributs dérivés. A la différence de la génération des commandes permettant l'insertion d'un dictionnaire de données (paragraphe 4.1.2) où pour chaque instance du dictionnaire on génére une commande d'insertion, quand il s'agit de la création du schéma du contenu les informations contenues dans plusieurs instances servent à la création d'une seule commande. En plus du modèle du dictionnaire enrichi, nous avons besoin du modèle de données des instances (du contenu) pour créer le schéma de la base de données du contenu.

En effet, au niveau du dictionnaire, chaque famille de composants est décrite par un maximum de propriétés possibles. Celles-ci ne sont représentées que d'une manière partielle au niveau du contenu (selon les besoins des utilisateurs) (cf. paragraphe 3.1.3 du chapitre 3). L'accès au modèle de données permet donc de savoir quelles sont les propriétés représentées effectivement dans le contenu afin de créer le schéma de la base de données correspondant.

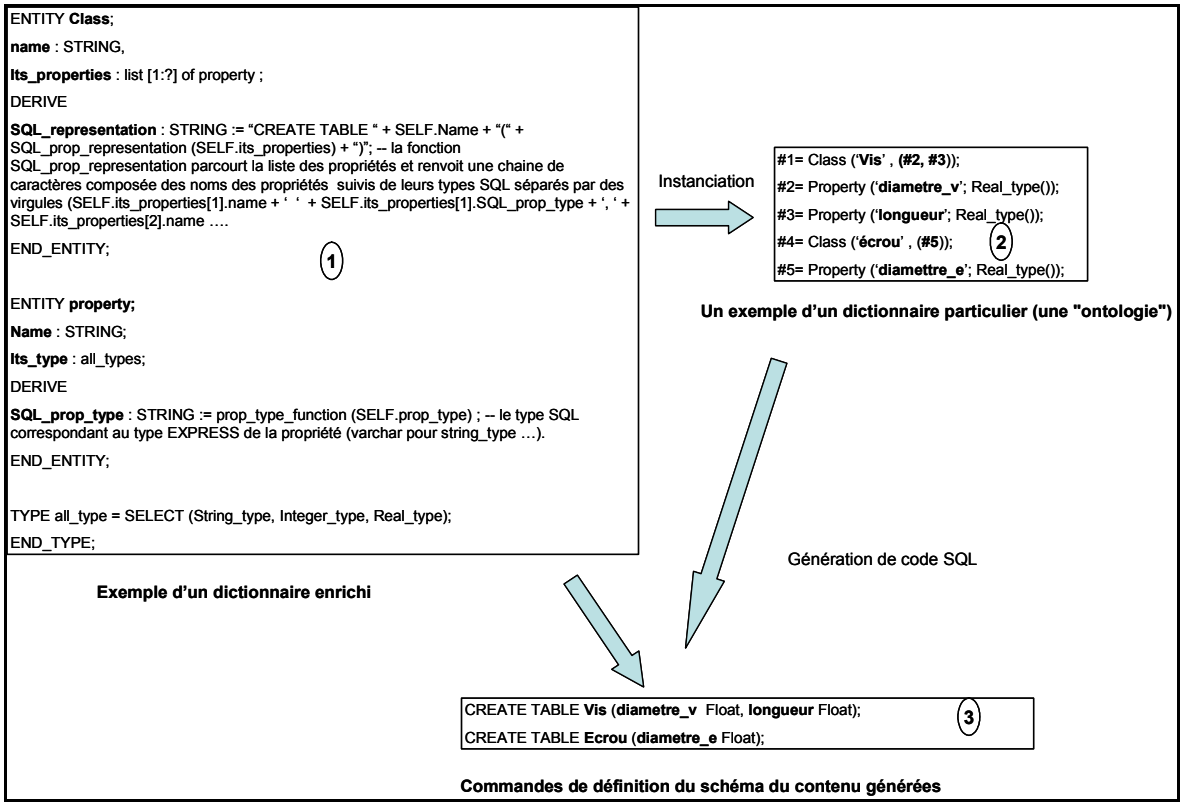


Figure 42 : génération de code LDD de SQL pour la création du schéma du contenu à partir d'une ontologie.

La Figure 42 illustre en partie la génération du code SQL permettant la création du schéma du contenu dans la base de données cible en faisant, pour simplifier, l'hypothèse que toutes les propriétés définies dans l'ontologie sont effectivement représentées dans les instances. Ce schéma correspond au modèle conceptuel de données représenté par une ontologie décrivant des vis et des écrous (partie 2 de la Figure 42). La lecture de cette ontologie par un outil EXPRESS connaissant le modèle du dictionnaire de PLIB enrichi par des attributs dérivés (parie 1 de la Figure 42) permet la génération du code SQL. Ce code contient des commandes de définition de données (CREATE TABLE …) qui représente chacune la valeur de l'attribut dérivé

SQL_representation de l'entité *class*. L'exécution de ces commandes dans un SGBD cible permet la création des tables qui contiendront le contenu d'un catalogue de composants (i.e les instances des vis et des écrous pour cet exemple).

4.1.4 Peuplement du contenu

La dernière étape de l'intégration du modèle de données PLIB consiste à peupler la base de données cible par un contenu d'un catalogue de composants (représenté par PLIB). Nous considérons que ce contenu existe déjà sous un format d'échange de données (i.e. dans un fichier physique) conformément aux instances du modèle explicite de PLIB représenté dans partie 25 de PLIB [ISO 13584-25:2004]. Cette étape n'est indispensable que lors d'une importation de données de composants à partir d'un catalogue existant. Dans le cas d'une utilisation du gestionnaire de données de composants en vue de créer des catalogues de composants (par un fournisseur de composants, par exemple), le peuplement de cette partie s'effectue au fur et à mesure de la création de composants directement par des requêtes SQL. La génération du code SQL pour importer les instances est réalisée en appliquant la programmation évènementielle sur le modèle du contenu (ou modèles des instances). La Figure 43 illustre la génération des commandes d'insertion des données du langage SQL à partir d'un fichier physique contenant les instances du contenu PLIB.

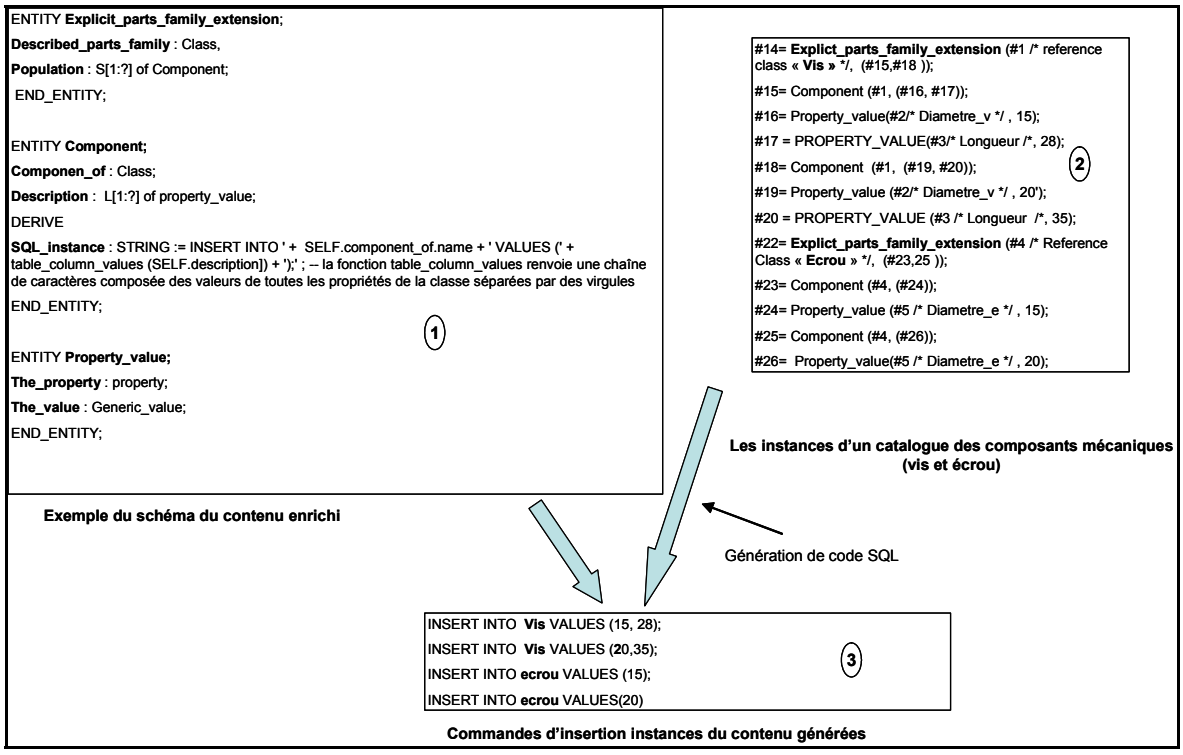


Figure 43 : Génération de commandes d'insertion de données pour peupler le schéma du contenu

Sur cette figure, la partie 1 représente une partie du modèle des instances (ou du contenu) PLIB enrichi par des attributs dérivés. L'attribut dérivé *SQL_instance* permet la génération, pour chaque instance des composants, d'une commande SQL permettant son insertion dans la base de données cible. Cette commande est le résultat d'une concaténation entre des mots clés du SQL (INSERT INTO …) avec les valeurs des propriétés de la classe concernée. Le référencement des concepts (*Class* et *property*)

définis au niveau de l'ontologie se réalise au travers des identifiants universels (*BSU*) de ces concepts. Néanmoins, pour des raisons de simplification, nous avons fait référence aux concepts eux-mêmes (l'attribut *Component_of* référence *Class*). La partie 2 de la Figure 43 représente un extrait du fichier physique des instances du contenu. Dans cette partie, *#1* et *#4* référencent respectivement les classes *vis* et *Ecrou* telles qu'elles sont représentées dans la partie 2 de la Figure 42. Ceci est valable également pour les propriétés des ces classes à savoir *Diametre_v, Longueur* et *Diametre_e* qui référencent respectivement #2, #3 et #5. Le code généré constitue l'ensemble des valeurs de l'attribut dérivé *SQL_instance* calculées pour chaque composant du contenu représenté dans ce fichier physique. Finalement, l'exécution des ces commandes au sein du SGBD cible permet l'importation les données du contenu PLIB dans cette base de données.

5 Gestionnaire de données de composants basé sur le modèle explicite de PLIB

5.1 Introduction

Nous présenterons dans ce paragraphe le gestionnaire de PLIB résultant de l'implémentation du modèle explicite au sein d'une base de données cible. Nous commençons d'abord par la présentation de son architecture composée de trois parties distinctes. Celle-ci comprend les deux parties importantes du modèle PLIB : l'ontologie et le contenu et la métabase du SGBD contenant les informations sur les tables, leurs colonnes, les contraintes, les clés étrangères, etc. nécessaires à la gestion de la base de données. Ensuite, nous détaillons les approches possibles de sélection des données de composants dans ce gestionnaire. Deux niveaux d'interrogation des données sont identifiés pour ce gestionnaire. Le premier niveau concerne la sélection des données directement au niveau du contenu, le second la sélection basée sur l'ontologie.

5.2 Architecture du gestionnaire de données de composants représentées dans le modèle explicite de PLIB

A l'inverse du gestionnaire de données de composants étudié dans du chapitre 3, ce gestionnaire est géré entièrement au sein de la base de données cible. Ainsi, lorsque les traductions du paragraphe précédent sont effectuées, aucune liaison avec le modèle EXPRESS de PLIB n'est requise, car toutes les informations représentées dans le modèle explicite de PLIB sont simples et peuvent être représentées sans difficultés dans la base de données cible.

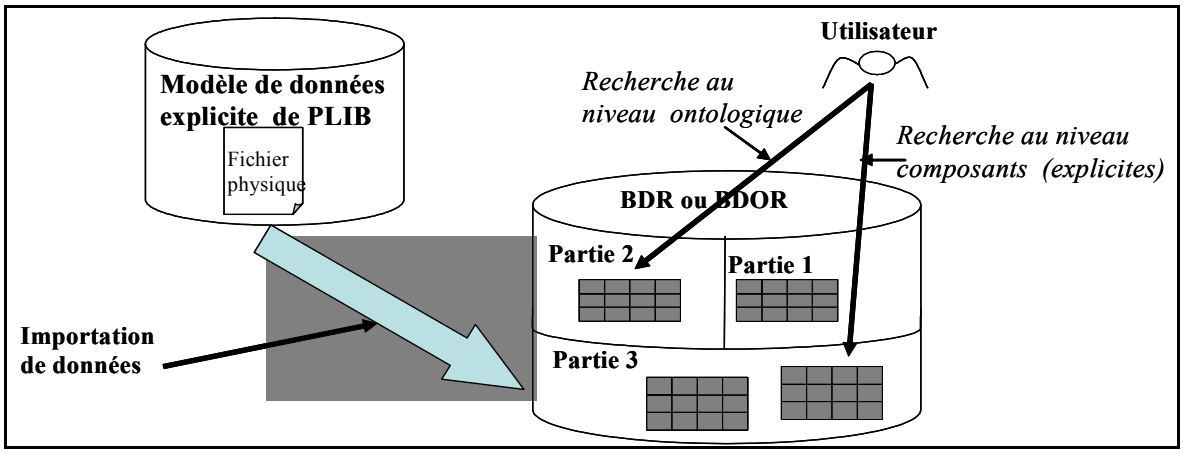


Figure 44 : Gestionnaire de données de composants basé sur l'implémentation du modèle explicite de PLIB

La Figure 44 illustre l'architecture du gestionnaire de données de composants fondée sur le modèle explicite. Ce gestionnaire est basé sur une architecture de base de données à trois niveaux. La première partie (partie 1) contient les méta-données relatives à la gestion de la base de données cible. Elle contient toutes les informations correspondant aux tables, à leurs attributs, aux contraintes et aux triggers, etc. La deuxième partie (partie 2) contient l'ontologie qui décrit le domaine de données représenté. Elle identifie tous les concepts qui y sont utilisés. Elle permet de stocker les modèles conceptuels de données au sein même de la base de données.La troisième partie (partie 3) contient les données proprement dites du domaine de données représenté. Dans notre cas, il s'agit du contenu des catalogues de composants conformes au modèle PLIB. La création des schémas des deuxième et troisième parties et leur peuplement ont été détaillés dans les paragraphes précédents.

5.3 Interrogation et sélection des données de composants

Le gestionnaire de composants comporte deux niveaux d'interrogation et de sélection de la base de données cible : la sélection des composants au niveau du contenu et la sélection des données au niveau de l'ontologie. Le premier niveau d'interrogation et de sélection de données est simple. A ce niveau, les requêtes ressemblent à toutes les requêtes d'une base de données relationnelle ou relationnelle-objet. Ces requêtes sont écrites directement dans un langage de requêtes supporté par la base de données cible (exemple : le langage SQL). A ce niveau de représentation de données, les requêtes peuvent être basées indifféremment sur des propriétés caractéristiques ou sur des paramètres de contexte, c'est-à-dire que les composants peuvent être sélectionnés par rapport à leurs propriétés caractéristiques (diamètre d'une vis, par exemple) ou par rapport à leur contexte d'insertion (charge de cisaillement appliqué sur la vis, par exemple). Il faut rappeler que dans le modèle explicite, les domaines de valeurs représentant les paramètres de contexte sont très limités (une seule valeur).

Le second niveau d'interrogation de la base de données est l'interrogation au niveau de l'ontologie. Ce niveau d'interrogation de données nécessite le développement d'un langage d'interrogation de bases de données spécifique permettant d'effectuer ce type d'interrogation. Une première spécification d'un tel langage a été proposée avec le

langage CQL (Class Query Language) [Murayama 00], [Murayama 03]. L'avantage de ce langage est que les requêtes présentent la particularité d'être indépendantes de tout système implémentant l'ontologie. En effet, les requêtes formulées au niveau de l'ontologie sont complètement transparentes par rapport à l'implémentation du contenu. Si par exemple, pour des raisons d'implémentation, une classe de composant a été représentée par plusieurs tables dans une base de données purement relationnelle et par une seule classe (table à objets) dans une base de données relationnelle-objet, la requête permettant de récupérer les informations concernant celle-ci est la même pour les deux systèmes. Cela suppose évidement que chaque système implémente le langage CQL.

Les requêtes CQL peuvent être de deux natures différentes : le premier niveau concerne exclusivement le niveau de l'ontologie, le second concerne le niveau du contenu. Pour le premier niveau, il s'agit des requêtes demandant des informations sur les concepts représentés (les classes et leurs propriétés par exemple : requêtes adressant la partie 2 de la base de données). Au second niveau, les requêtes portent sur la sélection de composants au niveau du contenu (partie 3 de la base de données cible).

6 Conclusion

La réflexion sur l'implémentation du modèle implicite de PLIB dans une base de données cible a abouti au développement d'une nouvelle approche de représentation des contenus des catalogues de composants, l'approche explicite. Celle-ci consiste à énumérer explicitement pour chaque famille de composants toutes ses instances. Nous avons présenté dans la première partie de ce chapitre les différents concepts de cette représentation. Nous avons ensuite abordé, dans la deuxième partie, l'implémentation d'une représentation explicite des données de composants PLIB dans une base de données cible. Nous avons étudié la génération de code (langage SQL) correspondant au modèle de données PLIB (écrit en langage EXPRESS) pour réaliser son intégration dans une base de données cible. Deux approches sont analysées : la génération de code par analyse syntaxique et la génération de code par programmation événementielle appliquée au langage EXPRESS. La première approche est basée sur l'utilisation d'une grammaire hors contexte, la grammaire hors contexte utilisée est celle qui définit le langage EXPRESS. La programmation événementielle est fondée sur le principe de grammaire attribuée. Les calculs sur une structure arborescente sont répartis sur chaque nœud de cette structure. En programmation événementielle, la génération de code est réalisée sur chaque nœud (entité) du modèle EXPRESS, par l'intermédiaire de l'ajout d'attributs dérivés à chaque entité du modèle EXPRESS. Ces attributs sont appelés également règles sémantiques dans la grammaire attribuée. Chaque attribut dérivé calcule donc un fragment du code correspondant à la représentation d'une instance de l'entité pour laquelle il est défini dans une base de données cible. L'assemblage de tous les fragments constitue un code complet de traduction du modèle EXPRESS vers un modèle de la base de données cible.

Nous avons présenté les avantages et les inconvénients de chaque approche et nous avons choisi la seconde approche pour l'appliquer au modèle PLIB, ce qui fait apparaître deux niveaux d'application. Le niveau génération des schémas de bases de données et le niveau génération des instances pour les peupler. La génération des

instances est réalisée par un enrichissement du modèle de données lui-même. Mais avant de pouvoir insérer les données au niveau de la base de données cible, il faut d'abord créer le schéma correspondant de la base de données. Nous avons utilisé la méta-modélisation. Le méta-modèle représentant le schéma du contenu est donné par le modèle PLIB : il s'agit du modèle du dictionnaire, enrichi des attributs dérivés pour permettre la génération du schéma du contenu. La génération du schéma du dictionnaire a pour objet de générer un schéma de base de données correspondant à un modèle représenté directement dans le langage EXPRESS. Nous avons alors utilisé le méta modèle du formalisme EXPRESS. Ce méta-modèle décrit les concepts du langage EXPRESS (entité, attribut, type, …) par EXPRESS lui-même. Ce méta-modèle du formalisme a été lui même enrichi par des attributs dérivés permettant la génération de commandes de définition de données (LDD de SQL) pour créer un schéma de base de données correspondant à un modèle EXPRESS. Le modèle à traduire doit être d'abord représenté comme une instanciation du méta modèle (dans un fichier physique) de ce formalisme avant d'être traduit en code SQL.

Nous avons présenté ensuite dans ce chapitre les détails d'implémentation du modèle PLIB. Nous avons commencé par la création du schéma du dictionnaire (permettant de stocker des ontologies) puis son peuplement par une ontologie particulière. L'étape suivante a permis de créer le schéma du contenu à partir de l'ontologie insérée. Une fois ce schéma créé, nous l'avons peuplé par un contenu de catalogue de composants. Toutes ces étapes ont été réalisées d'une manière complètement automatique en utilisant la technique de la programmation événementielle du langage EXPRESS.

Dans la dernière partie de ce chapitre nous avons présenté le gestionnaire de données de composants issu de l'implémentation du modèle explicite de PLIB dans la base de données cible. Nous y avons présenté également les différents niveaux d'interrogation de la base de données au sein de ce gestionnaire. Il s'agit de l'interrogation au niveau de l'ontologie et de l'interrogation au niveau du contenu. Notre maquette d'implémentation du gestionnaire de données PLIB a permis de valider complètement nos propositions. Elle a été considérablement facilitée grâce au développement du modèle explicite de PLIB. La mise en œuvre des outils de sélection des données au niveau de ontologie reste à finaliser en fonction des spécifications définitives du langage CQL.

Chapitre 5

Un exemple de mise en œuvre: implémentation de PLIB dans le SGBDRO PostgreSQL

Résumé
Nous avons présenté dans le chapitre précédent le modèle explicite de PLIB et les différentes approches permettant son intégration automatique dans une base de données cible. Nous présentons dans ce chapitre une représentation particulière de ce modèle explicite dans une base de données relationnelle-objet : PostgreSQL. Nous présentons, d'une part, les détails d'implémentation du modèle de données PLIB (dictionnaire et contenu) dans PostgreSQL et, d'autre part, quelques méthodes de sélection de composants. La représentation du dictionnaire de données revient à représenter les concepts du langage EXPRESS dans PostgreSQL, car celui-ci est implémenté directement au niveau des descripteurs (traduction directe de son modèle EXPRESS dans PostgreSQL). L'implémentation du contenu est un peu différente. Ceci est dû au concept de propriétés visibles et applicables définies dans le modèle PLIB et à notre choix de représentation des instances (propriétés représentées). La sélection des composants est également discutée dans ce chapitre. Finalement nous présentons deux approches de connexion entre le gestionnaire de données de composants et un SGDT : l'intégration interne au sein d'une seule base de données et l'intégration externe dans deux bases de données différentes.

1 Introduction

Nous présentons dans ce chapitre la mise en œuvre des approches développées dans les chapitres précédents dans un système de gestion de bases de données particulier. Nous avons choisi pour ce faire d'utiliser un SGBD relationnel-objet car, d'une part, il conserve la puissance du modèle relationnel qui est basé sur des modèles mathématiques solides et bien maîtrisés, et, d'autre part, il facilite la représentation des données en utilisant des concepts issus de la modélisation orientée objets. Nous avons choisi le SGBDRO PostgreSQL pour réaliser cette implémentation.

Comme le modèle de données PLIB est modélisé dans le langage EXPRESS, et qu'il modélise les catalogues de composants en se basant sur des concepts orientés objets, la représentation des concepts EXPRESS dans PostgreSQL couvre en grande

partie la représentation de PLIB dans PostgreSQL. Nous décrivons dans ce chapitre les solutions possibles pour réaliser l'implémentation de la majorité de ces concepts et analysons les solutions les plus avantageuses.

La représentation des concepts EXPRESS dans PostgreSQL permet directement la représentation du dictionnaire PLIB. Pour le contenu, son implémentation ne diffère pas beaucoup de celle du dictionnaire car la plupart des concepts du langage EXPRESS sont repris dans le modèle PLIB (héritage, polymorphisme, …). Cependant, afin de favoriser une meilleure méthode de sélection de données, une particularité du modèle du contenu concerne la représentation des familles de composants et de leurs propriétés (visibles, applicables et représentées). En effet, les familles de composants ne peuvent être instanciées qu'au niveau des classes feuilles ; dès lors la question de la représentation complète de la hiérarchie se pose. Nous avons présenté, dans le paragraphe 3.1.3 du troisième chapitre, la description d'une solution concernant la création du schéma du contenu en se basant conjointement sur l'ontologie et le contenu. Nous donnons dans ce chapitre les approches étudiées pour représenter ces différentes catégories de propriétés dans PostgreSQL (qui supporte la forme classique de l'héritage).

Le dernier maillon de la chaîne d'intégration du modèle PLIB et d'un SGDT est la connexion entre les deux modèles. Nous proposons deux types de connexion. Le premier type est la connexion interne. Celle-ci suppose la représentation, au sein d'une même base de données, des données des composants et de celles des produits. Cette base de données sera gérée par le SGDT lui-même. Les approches développées dans cette thèse, pour représenter un gestionnaire de données de composants basé sur le modèle PLIB seront utilisées pour créer ce gestionnaire au sein même du SGDT. Dans ce cas, le référencement des composants à partir de structures des produits est réalisé par des pointeurs internes au sein d'une unique base de données intégrée. Le second type de connexion est la connexion externe. Dans ce cas, les données des composants et celles des produits sont représentées dans deux bases de données séparées. Le gestionnaire de données de composants est un système indépendant (représenté dans PostgreSQL), mais piloté par le SGDT à l'instar de ce qui se passe avec la plupart des systèmes utilisés dans les entreprises (CAO, GPAO, etc.). Dans ce cas, le référencement des composants dans les structures des produits gérées par le SGDT est réalisé par des identifiants externes au SGDT.

2 Choix du SGBD

Comme nous l'avons vu au chapitre 1, les SGBD relationnels-objet nous semblent les mieux adaptés à la représentation des données de composants PLIB. En effet, ces systèmes regroupent deux caractéristiques importantes des systèmes de gestion des données. D'une part, ils sont basés sur des modèles relationnels qui reposent sur des théories solides (l'algèbre relationnelle), et, d'autre part ils conjuguent avec cela la puissance des concepts de la modélisation orientée objet.

Nous avons étudié deux SGBD relationnels-objet (Oracle et PostgreSQL). Notre choix s'est porté sur PostgreSQL. En effet, ce système est un SGBDRO qui représente une grande partie des concepts objets, possède un programme source ouvert et est

disponible avec une licence gratuite. Son programme source ouvert donne la possibilité d'opérer d'éventuels changements sur le système lui-même (un seul changement a été réalisé pour permettre de lever la contrainte sur le nombre maximum de caractères (31 au départ) que peut prendre un nom de table ou d'attribut). De plus, ce système intègre les mécanismes de gestion de l'héritage et offre un langage procédural puissant : le PGPLGSQL. Ce langage nous permet de renforcer les contraintes d'intégrité de la base de données et de calculer les attributs dérivés ainsi que les attributs inverses. Toutefois, il y a certaines lacunes dans la gestion du polymorphisme et de l'intégrité référentielle qui en découle. Celle-ci doit être prise en compte par le concepteur de la base de données. Nous détaillerons dans les sections suivantes le SGBDRO PostgreSQL.

2.1 Présentation de PostgreSQL

A l'origine PostgreSQL était un projet de recherche de bases de données dirigé par le Professeur Michael Stonebraker de l'Université de Berkeley (US). Il fut créé pour faciliter les recherches sur les Systèmes de Gestion de Base de Données. Sa fonction principale était de fournir une plate-forme de test pour l'implémentation de nouvelles idées dans la recherche en matière de base de données. De nombreux étudiants et professeurs de Berkeley ont contribué à la mise en place du prédécesseur de PostgreSQL appelé Ingres au début de son développement 1977-1985 [Momjian 01].

A la même université, un serveur de base de données relationnel-objet appelé Postgres a été développé (1986-1994). En 1994, Andrew Yu et Jolly Chen ont ajouté à Postgres le langage SQL supportant des concepts objets (héritage, par exemple). Le résulta de ce projet fut appelé Postgres95. Son code fut alors délivré sur le Web afin que le produit puisse être utilisé et développé librement. Postgres95 était un descendant public et open source du code original de Berkeley. Aujourd'hui il ne constitue plus un programme de recherche mais il est devenu la base de développement de certains SGBD de commerce (Informix, par exemple).

A partir de 1996, il devint clair que le nom de Postgres95 n'était plus adapté. Un nouveau nom, PostgreSQL, fut choisi pour refléter la relation entre le Postgres originel et les versions plus récentes supportant SQL.

Le modèle de données PostgreSQL peut être considéré comme un modèle «relationnel étendu» ou «relationnel objet» [Momjian 01]. Il s'appuie sur le modèle relationnel mais apporte les extensions suivantes :

- les classes,
- l'héritage,
- les types de données utilisateurs (tableaux, structures, listes..),
- les fonctions,

Cela permet de qualifier PostgreSQL de système de gestion de base de données «relationnel-objet» (*ORDBMS*).

2.1.1 Architecture de PostgreSQL

PostgreSQL utilise le protocole basé sur des messages pour communiquer entre le client et le serveur. Ce protocole est implémenté sur TCP/IP et sur le *Domain Sockets* de l'*UNIX*. Dans le jargon des bases de données, Postgres utilise l'approche dite "un processus par utilisateur" utilisée dans les modèles client/serveur. Une session PostgreSQL utilise les processus *UNIX* suivants :

- un processus serveur, qui gère les fichiers de la base, accepte les connexions à la base depuis les applications client et exécute les actions sur la base pour le compte des clients. Le programme serveur est appelé *Postmaster,*

- l'application cliente qui désire effectuer des opérations sur la base. Les applications clientes peuvent être de nature différentes : ce peut être un outil texte, une application graphique, un serveur web qui accède à la base pour afficher des pages web, ou un outil de maintenance de bases de données spécialisé. Certaines applications clientes sont fournies par PostgreSQL, la plupart sont développées par les utilisateurs.

En tant qu'applications client/serveur typiques, les clients et le serveur peuvent se trouver sur des hôtes différents. Dans ce cas ils communiquent par une connexion réseau TCP/IP. Le serveur PostgreSQL peut traiter de multiples connexions simultanées depuis les clients. Pour cela il démarre un nouveau processus ("*Forks*") pour chaque connexion.

2.1.2 Concepts et organisation des données dans PostgreSQL

2.1.2.1 Dictionnaire de données et métabase

L'ensemble des descriptions des objets manipulés, des contraintes et des opérations est habituellement stocké sous une forme particulière dans un fichier ou un ensemble de fichier appelé *dictionnaire de données* [Gardarin 01]. La plupart des systèmes de gestion de bases de données permettent de manipuler ce dictionnaire, par exemple pour accéder au libellé d'un item, changer un libellé, ajouter ou supprimer un élément dans une liste de réponses possibles, ajouter ou supprimer une procédure. L'ensemble des attributs possible d'un item peut lui-même être considéré comme un objet abstrait doté de propriétés. Il est alors possible de constituer une base de données décrivant la base de données et de gérer cette base abstraite de la même façon que les données proprement dites. On appelle *métabase* une telle base de données [Gardarin 01].

2.1.2.2 Classes et objets

Dans les SGBD orientés objets en général, les objets sont la représentation des entités du monde réel. Chaque objet complexe est construit à partir de valeurs de ses propriétés caractéristiques. On appelle classe, le regroupement des objets ayant des propriétés caractéristiques communes. Une classe représente donc une abstraction commune de ces objets. PostgreSQL considère les tables créées dans la base de données comme des classes. Chaque tuple créé dans la base de données est considéré comme un objet. L'identité d'objets, qui est une caractéristique importante du modèle

orienté objet, est garantie par le système lui même. En effet, des propriétés implicites tel que le **OID** (Object IDentifier) sont ajoutés pour chaque table créée, permettant l'identification des objets indépendamment des clés primaires qui peuvent être définies par les concepteurs de la base de données. Par contre la notion de méthodes, définies sur des classes et des objets, n'existe pas dans PostgreSQL.

2.1.2.3 Héritage

Il s'agit d'un des concepts centraux de la modélisation orientée objet. La technique d'héritage simplifie la création d'une nouvelle classe (ou type) en donnant au développeur la possibilité de réutiliser des classes existantes et d'ajouter simplement de nouveaux éléments. Des hiérarchies de classes peuvent être créées, elles permettent la spécialisation des classes à partir des classes racines et facilitent le parcours du modèle de données représenté. PostgreSQL intègre ce concept en le représentant par un héritage entre tables (i.e. classes) (voir paragraphe 3.1.1.4 ci-dessous).

2.1.2.4 PL/SQL

Le langage PL/SQL, appelé PGPLSQL dans le jargon **P**ost**G**reSQL, est un langage L4G (un langage de quatrième génération), fournissant une interface procédurale au SGBD PostgreSQL. Le langage PL/SQL intègre parfaitement le langage SQL en lui apportant une dimension procédurale [PostgreSQL 03].

En effet, le langage SQL est un langage déclaratif non procédural permettant d'exprimer des requêtes dans un langage relativement simple. En contrepartie il n'intègre aucune structure de contrôle permettant par exemple d'exécuter une boucle itérative.

Au contraire, le langage PL/SQL permet de manipuler de façon complexe les données contenues dans une base PostgreSQL en transmettant un bloc de programmation au SGBD au lieu d'envoyer une simple requête SQL. De cette façon les traitements sont directement réalisés par le système de gestion de bases de données au niveau du serveur. Cela a pour effet notamment de réduire le nombre d'échanges à travers le réseau et donc d'optimiser les performances des applications.

De plus, le langage PL/SQL permet de faire appel à des procédures externes, écrites dans un autre langage (généralement le langage C).

2.1.2.5 Contraintes d'intégrité

Le fait de définir une structure d'une base de données ne suffit pas à garantir l'exactitudea de l'information présente dans cette base. Pour aider au maintien de la cohérence dans la base, il faut spécifier des règles édictant des propriétés que doivent respecter les données de la base. Ce sont les contraintes d'intégrité qui garantissent cette tâche.

Une contrainte d'intégrité est une assertion qui doit être vérifiée à tout moment par les données contenues dans la base de donnée. Il existe deux types de contraintes d'intégrité [Gardarin 01]:

- les contraintes d'intégrité structurelles,
- les contraintes d'intégrité non-structurelles,

Les contraintes d'intégrité structurelles sont directement liées au modèle. Une contrainte structurelle spécifique à un modèle exprime une propriété fondamentale d'une structure de données du modèle [Gardarin 01]. Elles sont généralement statiques. On peut distinguer les contraintes structurelles suivantes :

- **contrainte de clé primaire.** Elle constitue la principale contrainte structurelle. Elle est constituée d'un ensemble minimal d'attributs permettant d'identifier d'une manière unique chaque tuple de la relation considérée,

- **contrainte référentielle.** Une contrainte de référence portant sur une relation *A* impose que la valeur d'un groupe d'attributs apparaisse comme valeur de clé primaire dans une autre relation *B*. Une contrainte de référence impose donc que tout tuple de la relation *A* qui se réfère à la relation *B* se réfère à un tuple existant effectivement dans la relation *B*. Cette contrainte est également appelée contrainte de clé étrangère.

- **contrainte du domaine.** Ce type de contrainte permet de restreindre la plage de valeurs d'un domaine. En général un domaine est défini par un type et un éventuel domaine de variation spécifié par une contrainte de domaine. Une contrainte de domaine peut simplement préciser la liste des valeurs permises (définition en extension) ou une plage de valeurs (contrainte en intention).

- **contrainte de non nullité.** Dans le vocabulaire des SGBD une valeur **NULL** signifie l'absence de valeur. Une contrainte **NOT NULL** spécifie que la valeur d'un attribut doit être renseignée.

Les contraintes d'intégrité non inhérentes au modèle de données sont regroupées dans la classe de contraintes non structurelles. La plupart traitent de l'évolution des données à la suite des mises à jour. Elles sont souvent appelées **contraintes comportementales**. [Gardarin 01] définit une contrainte comportementale comme une « *contrainte d'intégrité exprimant une règle d'évolution que doivent vérifier les données lors de mises à jour* ». Dans le cadre de notre travail ce type de contraintes n'a pas été étudié, et nous n'avons donc pas étudié la capacité du PostgreSQL à supporter ce type de contraintes.

3 Représentation de concepts EXPRESS dans le SGBD PostgreSQL

Nous présentons les correspondances entre les concepts d'EXPRESS et de PostgreSQL. Pour chaque concept EXPRESS nous présentons les différentes

possibilités de le représenter en PostgreSQL. Puis, nous choisissons la représentation qui nous semble la plus avantageuse. Cette correspondance entre les concepts permet de réaliser par la suite l'intégration du modèle PLIB dans PostgreSQL.

3.1.1 Les entités

3.1.1.1 Catégorisation

Pour représenter une entité EXPRESS dans une base de données PostgreSQL, on utilise une table. Le nom de la table est celui de l'entité (voir le Tableau 1). Les attributs de l'entité sont représentés par des colonnes de tables. La représentation des types des attributs est détaillée dans le paragraphe 3.2. Le Tableau 1 présente un exemple de traduction d'une entité EXPRESS simple (*personne*) par une commande SQL permettant la création de la table qui la représente au sein de PostgreSQL.

EXPRESS	PostgreSQL
ENTITY personne; son_nom : **STRING**; son_prenom : **STRING**; **END_ENTITY**;	**CREATE TABLE** personne (son_nom : **VARCHAR**, son_prenom : **VARCHAR**);

Tableau 1 : Représentation d'une entité dans la base de données PostgreSQL

3.1.1.2 Identification d'instances

En EXPRESS, chaque instance du modèle est identifiée par un identificateur unique au sein d'un fichier d'instances. Cet identificateur est géré d'une manière automatique et transparente par le système gérant des données EXPRESS. Cependant, il ne figure nullement comme un attribut d'une quelconque entité. Il est donc nécessaire de trouver une équivalence de cet identificateur au sein de la base de données PostgreSQL. Deux solutions sont alors possibles :

- utilisation de l'**OID** PostgreSQL : comme nous l'avons signalé ci-dessus, la base de données gère elle-même un identificateur unique au sein de la base de données. Cependant, dans les versions actuelles du système, l'utilisation de ces **OID** est réservée exclusivement au système lui-même. L'utilisateur ne peut pas le référencer comme une clé étrangère par exemple ;
- création d'une nouvelle colonne d'identification pour chaque table créée : on crée une séquence, un nombre qui s'auto-incrémente à l'instanciation. Puis, la colonne sera liée à cette séquence par une contrainte. La colonne étant de type entier (**INTEGER**) et créée par l'utilisateur, son utilisation par ce dernier est complètement libre (elle peut être référencée comme une clé étrangère, par exemple).

C'est la seconde solution que nous avons choisie. Pour assurer une unicité globale de cet identificateur au niveau de la base de données, nous avons créé une super-classe dont toute table créée dans la base de données doit hériter. Cette classe (table), appelée *id_table,* est mono-colonne. Le nom de la colonne est « RID » (Row IDentifier) de type **SERIAL8** (valeur max 2^{63}- 1). Le Tableau 2 présente cette table d'identification.

PostgreSQL
CREATE TABLE ID_table (RID **SERIAL8);**

Tableau 2 : Table d'identification

En déclarant un attribut de type **SERIAL8**, PostgreSQL crée automatiquement une séquence, un index et une contrainte **NOT NULL** sur cet attribut. L'index permet un accès rapide aux données par une méthode b-tree. Cet index crée également une contrainte d'unicité sur cet attribut. Dans PostgreSQL seules les contraintes **NOT NULL** et **CHECK** ainsi que les valeurs par défaut sont héritées. Les contraintes d'unicité ainsi que les index ne le sont pas (elles sont appliquées exclusivement sur les tables pour lesquelles elles ont été déclarées). Pour garantir une optimisation d'accès à toutes les tables par leur RID, un index est créé sur cet attribut (le RID) pour chaque table créée.

3.1.1.3 Représentation des entités abstraites dans PostgreSQL

Dans le langage EXPRESS une entité déclarée abstraite ne peut être instanciée qu'à travers ses sous-types. Nous avons donc représenté cette abstraction par une règle de PostgreSQL appliquée sur la table représentant une entité abstraite. Cette règle permet en effet d'empêcher l'insertion d'instance pour la table déclarée abstraite (Tableau 3).

EXPRESS	PostgreSQL
ENTITY personne **ABSTRACT SUPERTYPE OF** (**ONEOF** (etudiant, enseignant)); son_nom : **STRING;** son_prenom : **STRING;** **END_ENTITY;**	**CREATE RULE** abstract_personne **AS** **ON** INSERT **TO** personne **DO** **INSTEAD** nothing**;**

Tableau 3 : représentation d'une entité EXPRESS abstraite dans PostgreSQL

3.1.1.4 Hiérarchisation

PostgreSQL prend en charge une forme d'héritage. Il est possible de représenter l'héritage entre entités par un héritage entre tables au moyen du mot clé **INHERITS**. L'héritage multiple est également supporté. Comparé au langage EXPRESS où une super classe déclare ses sous-classes, en PostgreSQL, seules les super-classes sont déclarées dans les sous-classes. Le tableau suivant illustre un exemple de représentation d'héritage dans PostgreSQL.

EXPRESS	PostgreSQL
ENTITY personne **SUPERTYPE OF** (**ONEOF**(etudiant,salarie)); son_nom : **STRING**; son_prenom : **STRING**; son_prenom2 : **OPTIONAL STRING**; **END_ENTITY;** **ENTITY** etudiant **SUPERTYPE OF** (etudiant_salarie) **SUBTYPE OF** (personne); sa_classe : **STRING**; **END_ENTITY;** **ENTITY** salarie **SUPERTYPE OF** (etudiant_salarie) **SUBTYPE OF** (personne); son_salaire : **FLOAT**; **END_ENTITY;** **ENTITY** etudiant_salarie **SUBTYPE OF** (etudiant, salarie); **END_ENTITY;**	**CREATE TABLE** personne (son_nom : **VARCHAR,** son_prenom : **VARCHAR,** son_prenom2 : **VARCHAR** **);** **CREATE TABLE** etudiant (sa_classe : **VARCHAR** **) INHERITS (**personne**);** **CREATE TABLE** salarie (son_salaire : **FLOAT** **) INHERITS (**personne**);** **CREATE TABLE** etudiant_salarie (**) INHERITS (**etudiant, salarie**);**

Tableau 4 : représentation de l'héritage dans PostgreSQL.

3.1.1.5 Association entre classes

Une fois les différentes tables modélisées, il faut pouvoir les associer entre elles. Comme cela est le cas dans les bases de données relationnelles, PostgreSQL permet d'utiliser les clés étrangères. Cependant cette utilisation ne permet pas le polymorphisme. Ceci est dû au fait que les contraintes d'intégrité référentielle implémentées en PostgreSQL sont restées à leur état de base de données relationnelles. En effet, l'introduction des concepts de la modélisation orientée objets au sein de PostgreSQL suit une évolution progressive. En attendant que tous ces concepts soient implémentés dans PostgreSQL, deux solutions ont été donc étudiées pour résoudre ce problème :

- la première solution consiste à utiliser un RID générique dans la table référençante et d'y adjoindre une table de référencement. Le RID générique permet alors de faire référence à n'importe quelle instance dans la base de données. La table de référencement, quant à elle, permet de préciser cette association. En effet, cette table possède trois colonnes : la première permet de représenter le nom de la table référençante, la deuxième permet de représenter l'attribut de l'association et la dernière représente le nom de la table associée ;

- la seconde solution consiste à ajouter deux colonnes pour chaque table "référençante". La première colonne contient le RID de l'instance référencée et la seconde contient le nom de la table dans laquelle est insérée cette instance.

Parmi les deux solutions proposées, nous avons choisi la seconde, celle-ci nous semble présenter le plus d'avantages. D'abord, elle permet de regrouper dans le même tuple l'instance référencée et la table associée (ce qui permet d'optimiser les requêtes lors des jointures entre tables associées en diminuant le nombre de jointures). De plus, elle permet de partager les informations relatives aux associations sur plusieurs tables contrairement à la première solution qui les regroupe toutes dans une seule table.

3.1.2 Représentation des différents types d'attributs

Les attributs en EXPRESS sont de trois types : explicite, dérivé et inverse (voir annexe).

- Attribut explicite

Un attribut explicite (ou libre) est un attribut dont les valeurs doivent être fournies lors d'une implémentation du modèle (instanciation) pour créer une instance d'une entité [ISO 10303-11]. Toutefois, un attribut explicite peut être déclaré optionnel (mot clé **OPTIONAL**). Dans ce cas, la valeur de cet attribut n'est significative que pour certaines instances de l'entité (i.e. il peut être non valué pour certaines instances et porte alors le signe **$** dans le fichier d'instances). Dans PostgreSQL, les attributs explicites d'une entité sont représentés par les colonnes de la table représentant cette entité. Si aucune contrainte n'est exprimée sur la colonne représentant un attribut explicite, celle-ci peut contenir une valeur mais peut aussi ne pas être valuée (la valeur **NULL)**. Ceci représente donc un attribut optionnel du langage EXPRESS. En revanche, si l'attribut est obligatoire, il faut donc spécifier, au moyen d'une contrainte **NOT NULL**, que le champ doit être impérativement valué.

EXPRESS	PostgreSQL
ENTITY personne **SUPERTYPE OF** (**ONEOF**(etudiant,salarie)); son_nom : **STRING**; son_prenom : **STRING**; son_prenom2 : **OPTIONAL STRING**; **END_ENTITY;**	**CREATE TABLE** personne (son_nom : **VARCHAR NOT NULL,** son_prenom : **VARCHAR NOT NULL,** son_prenom2 : **VARCHAR);**

Tableau 5 : représentation des attributs explicites d'EXPRESS dans PostgreSQL

- Attribut dérivé

La valeur d'un attribut dérivé est calculée par le système au moyen d'une fonction. Un attribut dérivé représente une contrainte fonctionnelle (cf. paragraphe 2.2 du chapitre 2). Deux solutions de représentation au niveau de la base de données ont été étudiées :

- créer une colonne dans la table dont la valeur est calculée par une fonction. Une contrainte doit être créée pour assurer la cohérence de l'information lors des mises à jour. Cette contrainte peut être assurée par un trigger qui met à jour ce champ à chaque mise à jour de la table. Les champs peuvent alors être référencés par d'autres attributs dans d'autres tables. Cette représentation permet alors une vérification a posteriori de la contrainte fonctionnelle représentée par l'attribut dérivé.

- utiliser une vue qui calcule les attributs à chaque accès à la vue. Les attributs dérivés sont représentés comme étant des "attributs" de la vue. Dans cette deuxième représentation la contrainte fonctionnelle représentée par l'attribut dérivé est établie par le calcul.

Dans les deux cas, il faut l'utilisation de fonctions de dérivation (pour vérifier la valeur de l'attribut dans le premier cas et pour la calculer dans le second cas). On distingue deux types de fonctions de dérivation : les expressions simples et les fonctions complexes. Les expressions simples ne font pas appel à des fonctions génériques du langage EXPRESS et utilisent seulement des opérateurs simples pour les calculer (*, +, concaténer deux chaînes de caractères, ….). Les fonctions plus complexes sont celles définies en utilisant le "constructeur" de fonction d'EXPRESS (**FUNCTION**). Celles-ci peuvent utiliser des fonctions génériques du langage EXPRESS telles que **USEDIN, TYPEOF,** etc. ou les accès pointés. Ces cas de figure n'ont pas été pris en compte dans cette implémentation d'autant plus que ces types de fonctions n'ont été que rarement utilisés pour le calcul des attributs dérivés. Dans notre implémentation, nous avons retenu la seconde solution. Elle permet d'éviter le grand nombre de vérifications inutiles déclenchés par les triggers lors des mises à jour qui n'affectent pas les valeurs des attributs dérivés. De plus, une vue a un statut quasi identique à une table dans une base de données : c'est une table virtuelle. De ce fait, toutes les opérations réalisées sur une table peuvent l'être sur une vue. Elle peut même être matérialisée pour des besoins de performance de la base de données.

- Attribut inverse

Un attribut inverse permet de décrire le rôle d'une association, déclarée par un attribut explicite dans une autre entité (i.e. l'entité "référençante"), au niveau de l'entité associée.

Puisque la valeur de cette attribut dépend des instances d'une autre entité (l'entité référençante), celui-ci peut être représenté, à l'image de l'attribut dérivé, de deux manières différentes :

- utilisation d'une colonne qui contient la listes des instances référençant une instance quelconque de la table associée. Les valeurs de cette colonne doivent être mises à jour à chaque insertion (ajout d'une nouvelle instance) au niveau de la table référençante. Cette mise à jour peut être réalisée par un trigger ;

- représentation de l'attribut inverse dans une vue. Sa valeur est donc calculée par une fonction. Une fonction générique appelé **USEDIN**, traduction de la fonction **USEDIN** prédéfinie dans le langage EXPRESS, a été définie dans PostgreSQL. Cette fonction permet de calculer, à partir du nom de la table référençante et de son attribut d'association, la liste des instances de cette table référençante qui référence chaque instance de la table associée.

Pour les mêmes raisons justifiant le choix de la représentation des attributs dérivés, nous avons retenu la solution utilisant les vues pour représenter les attributs inverses.

3.2 Représentation des types

En EXPRESS, un type peut appartenir à cinq catégories. Il peut être simple (ou atomique) ou être un agrégat. Il peut ensuite être défini par restriction à partir d'un type de base ou d'un autre type sur lequel on peut ajouter des contraintes pour restreindre ses valeurs. Il peut être également un type énuméré qui définit un domaine limité regroupant un ensemble fixe de valeurs. La dernière catégorie de type utilisateur est l'union de types. C'est un type spécifique à EXPRESS déclaré à l'aide du mot clé **SELECT**. Les valeurs d'un tel type peuvent en effet appartenir à un type quelconque parmi ceux qui le constituent.

Nous proposons dans ce qui suit une représentation de chacun de ces types dans PostgreSQL.

3.2.1 Types atomiques

Les colonnes (attributs) des tables sont définies par des types (des domaines de valeurs). Il faut donc associer à chaque type atomique d'EXPRESS un type de PostgreSQL. Le tableau suivant identifie des correspondances entre les types atomiques d'EXPRESS et les types de PostgreSQL.

Types EXPRESS	Types PostgreSQL
BINARY	**BIT**
BOOLEAN	**BOOLEAN** + contrainte NOT NULL
LOGICAL	**BOOLEAN** sans contrainte NOT NULL, la valeur NULL de PostgreSQL sera utilisée pour représenter UNKNOWN d'EXPRESS
NUMBER	**NUMERIC**
REAL	**FLOAT**
INTEGER	**INTEGER**
STRING	**CHAR(N)** (Chaîne de caractères de longueur N fixe) **VARCHAR(N)** (Chaîne de caractères de longueur N variable) **TEXT** (Chaîne de caractères sans longueur fixée)

Tableau 6 : Représentation des types simples EXPRESS dans PostgreSQL

En ce qui concerne la représentation du type **STRING**, **CHAR(N)** représente le type **STRING(N) FIXED**, **VARCHAR(N)** représente **STRING(N)** et **TEXT** représente **STRING**.

3.2.2 Types définis par restriction

Les types définis par restriction sont utilisés pour représenter des domaines de valeurs contraints. Ces types sont créés soit à partir de types de base, soit à partir d'autres types définis. La représentation de ces types dans PostgreSQL peut se faire de deux façons différentes :

- par des types utilisateurs définis dans PostgreSQL,
- directement par les types de base de PostgreSQL.

La première proposition consiste à créer pour chaque type défini un type utilisateur dans PostgreSQL. En effet, le SGBD PostgreSQL permet la création de tels types en utilisant la commande **CREATE TYPE**. Une fois que ces types sont définis, ils sont utilisés comme n'importe quel autre type atomique. La création d'un type utilisateur PostgreSQL nécessite la création du même type dans le langage C et de deux fonctions écrite en langage C également. La première fonction, appelée **INPUT_FUNCTION** convertit le type de sa représentation utilisée dans la base de données vers une représentation interne, représenté en langage C, où la valeur peut être validée. La seconde fonction, appelée **OUTPUT_FUNCTION,** réalise l'opération inverse (c'est-à-dire transformer la valeur de sa représentation interne vers une représentation externe). La représentation interne est la seule forme compréhensible par le système pour la création de fonctions ou d'opérateurs utilisant ces types utilisateurs. L'utilisation du langage C permet de créer des types définis assez variés. On peut ainsi représenter des types contraints, mais également des enregistrements (RECORDs). Il existe cependant deux inconvénients à l'utilisation de ces types :

- le recours à un langage extérieur autre que SQL (ici C) pour créer un nouveau type. Cela a pour conséquence de compliquer l'intégration automatique d'un modèle EXPRESS dans une base de données PostgreSQL. Cette complexité est d'autant plus importante que toutes les fonctions créées dans le langage C doivent être créées et compilées séparément avant la création définitive des types ;

- la nécessité de créer pour chaque nouveau type tous les opérateurs (égalité, accesseurs, sélecteurs, modificateurs, testeurs, etc.) pour pouvoir l'utiliser comme les autres types définis dans PostgreSQL.

La seconde solution proposée consiste à représenter chaque type défini par son type atomique de base en respectant l'équivalence de types évoquée dans le Tableau 6. Ceci s'effectue au fur et à mesure de la création des tables représentant les entités d'un modèle de données EXPRESS. Ainsi, pour chaque attribut ayant pour type un type défini, la colonne correspondante à cet attribut aura pour type le type de base de ce type défini. Si le type représenté est contraint, cette contrainte peut être créée sur chaque colonne contenant une valeur de ce type. Cette contrainte peut être exprimée par une seule fonction PL/PGSQL, représentant la **WHERE RULE** du type EXPRESS, et attachée à chaque colonne par une contrainte **CHECK**.

Parmi ces deux solutions, nous avons adopté la seconde approche, car elle présente moins d'inconvénients que la première approche. Elle favorise également une intégration automatique du modèle EXPRESS dans PostgreSQL.

3.2.3 Type énuméré

Un type énuméré est un type construit dont les valeurs sont définies par une liste. Le type énuméré peut être représenté de deux manières différentes. La première consiste à utiliser l'approche précédente, c'est-à-dire un type de base qui est un **VARCHAR.** Une fonction peut y être associée pour vérifie la conformité des valeurs représentées à celles qui sont définies dans le type énuméré d'EXPRESS (les valeurs possibles).

La deuxième manière de représenter des types énumérés consiste à utiliser une table, ayant pour nom celui du type énuméré. Cette table est utilisée pour stocker les valeurs possibles de ce type. Elle contient en plus de la colonne qui contient les valeurs du type, une autre colonne. Cette dernière permet de mémoriser l'ordre des valeurs dans la liste utile dans le cas d'une comparaison. C'est cette deuxième approche qui a été adoptée pour représenter les types énumérés dans PostgreSQL. L'exemple suivant illustre bien cette approche.

EXPRESS	PostgreSQL
TYPE t_classe = **ENUMERATION OF** (A1, A2, A3); **END_TYPE;** **ENTITY** etudiant; ... sa_classe :t_classe; **END_ENTITY;**	**CREATE TABLE** t_classe(sa_valeur **VARCHAR PRIMARY KEY,** ordre **INTEGER**); **INSERT INTO** t_classe **VALUES** ('A1', 1); **INSERT INTO** t_classe **VALUES** ('A2', 2); **INSERT INTO** t_classe **VALUES** ('A3', 3); **CREATE TABLE** etudiant (... sa_classe **VARCHAR NOT NULL** **REFERENCES** t_classe (sa_valeur) **);**

Tableau 7 : Représentation de types énumérés par une table

3.2.4 Union de Types (SELECT)

Un type **SELECT** est une union de types. Nous avons représenté le type **SELECT** par une table ayant un nombre de colonnes égal au nombre de types constituant ce type. Chaque colonne a pour nom le nom du type dont elle contient les valeurs. Un attribut d'une entité EXPRESS ayant pour type un type **SELECT** sera représenté par une référence à la table représentant ce type **SELECT**. Chaque valeur de l'attribut représenté sera rangée dans une colonne de la table représentant son type (le type **SELECT**) selon le type effectif de la valeur. Si un type **SELECT** est une union de deux types (*integer_type* et *value_code_type*), une valeur de ce type sera stockée dans la colonne *integer_type* si son type effectif est le type *integer_type* et vice versa (sachant que la colonne *integer_type* est une colonne de la table représentant le type select dans PostgreSQL).

Pour garantir la cohérence des valeurs d'un attribut dont le type est un **SELECT**, une contrainte d'intégrité est exprimée sur la table représentant ce type. Cette contrainte permet l'instanciation d'une seule colonne à la fois.

EXPRESS	PPostgreSQL
TYPE VALUE_TYPE = SELECT(VALUE_CODE_TYPE, INTEGER_TYPE); **END_TYPE;**	**CREATE TABLE** TYPEVALUE_TYPE **(** VALUE_CODE_TYPE **VARCHAR** **CHECK(**ISVALUE_CODE_TYPE**(**VALUE_CODE_TYPE E**)),** INTEGER_TYPE INTEGER **CHECK(**ISINTEGER_TYPE**(**INTEGER_TYPE**)),** **CHECK (((**VALUE_CODE_TYPE **IS NULL) AND (**INTEGER_TYPE **IS NOT NULL)) OR ((**VALUE_CODE_TYPE **IS NOT NULL) AND (**INTEGER_TYPE **IS NULL))));**

Tableau 8: Représentation de types SELECT par une table

3.2.4.1 Représentation des agrégats

Les agrégats ou les collections en EXPRESS correspondent aux types **LIST**, **BAG**, **SET** et **ARRAY** (voir paragraphe 2.8 du chapitre 4). PostgreSQL, comme la majorité des SGBDRO, supporte les types collections (NFNF ou NF2 (Non First Normal Form : support d'attributs complexes et multivalués)). Contrairement au langage EXPRESS, PostgreSQL ne supporte qu'un seul type d'agrégats : il s'agit du type **ARRAY**. En effet, en PostgreSQL le type **ARRAY** représente un tableau ordonné d'élément de même type. Pour prendre en considération cette disparité, nous avons proposé une solution générique. Elle consiste à représenter chaque agrégat comme un **ARRAY** de références vers les valeurs effectives de cet agrégat qui sont rangées dans une table. Cette solution n'était que partielle. En effet, les listes de taille évolutive n'ont pas été traitées. De plus, les contraintes liées à chaque type d'agrégat (ordonné ou pas, avec duplication ou pas, …) n'ont pas été représentées.

3.2.5 Représentation des contraintes

Les types de contraintes EXPRESS sont au nombre de quatre : la contrainte locale (**WHERE RULE**), la contrainte d'unicité (**UNIQUE**), la contrainte de cardinalité (**INVERSE**) et la contrainte globale (**RULE**). Pour plus de détails sur les contraintes EXPRESS se reporter à l'annexe.

3.2.5.1 Contrainte locale (WHERE RULE)

Cette règle s'applique aux instances d'une entité pour vérifier ou contraindre les valeurs de certains attributs (sur lesquels s'applique la contrainte). Avec la contrainte **CHECK** de PostgreSQL, on spécifie une contrainte qui sera vérifiée à chaque saisie et à chaque mise à jour (**INSERT ET UPDATE**) d'une instance. Cette contrainte peut faire appel à une fonction booléenne définie en PL/SQL. Une contrainte locale EXPRESS peut être représentée par une contrainte **CHECK** de PostgreSQL. L'expression représentant cette contrainte peut être représentée soit par une expression directe soit par l'intermédiaire d'une fonction PL/SQL. Le tableau suivant montre un exemple simple de représentation de règle locale EXPRESS dans PostgreSQL.

EXPRESS	PostgreSQL

ENTITY salarie
SUBTYPE OF (personne);
  son_salaire : **REAL**;
WHERE
  wr : salaire **>=** 0.0 ;
END_ENTITY;
	CREATE TABLE salarie (
	  son_salaire : **FLOAT,**
	  CONSTRAINT wr **CHECK** (son_salaire>=0)
	) INHERITS (personne**);**

Tableau 9 : Représentation d'une contrainte locale EXPRESS en PostgreSQL

Cependant, des problèmes peuvent se poser pour la représentation de certaines contraintes locales, car celles-ci peuvent être exprimées à l'aide de fonctions génériques telle que **TYPEOF** et/ou à l'aide de l'opérateur **QUERY** du langage EXPRESS assez difficile à représenter en PostgreSQL. Ce type de contraintes n'a pas été pris en compte dans notre implémentation. Le développement de fonctions équivalentes à celles des fonctions génériques d'EXPRESS doit précéder tout implémentation complète de ce type de contraintes.

3.2.5.2 Contrainte d'unicité (UNIQUE)

En EXPRESS la contrainte d'unicité s'applique à l'ensemble des instances d'une entité. Elle assure l'unicité de la valeur d'un (ou plusieurs) attribut(s) sur la population de l'entité où elle est déclarée mais également dans toutes les sous-entités. En PostgreSQL, la contrainte d'unicité s'applique uniquement à la table pour laquelle elle a été définie, car les contraintes ne sont pas héritées par les sous-tables. Cela signifie que la contrainte d'unicité validera uniquement les instances de la table mère mais elle ne vérifie pas l'ensemble des instances de toutes les tables de la hiérarchie. Nous pouvons donc trouver, par exemple, une instance dans la table mère et une instance de la table fille avec la même valeur d'attribut.

Pour assurer une unicité sur l'ensemble de la population d'une hiérarchie de tables, nous proposons l'utilisation d'une contrainte **CHECK** exprimée au niveau de la table de plus haut niveau. Cette contrainte est attachée à une fonction qui vérifie, à chaque insertion d'une nouvelle instance, si la valeur de l'attribut déclaré unique existe déjà pour une autre instance dans la hiérarchie. Dans PostgreSQL, Les contraintes sur les attributs (**CONSTRAINT … CHECK …**) sont, avec la contrainte **NOT NULL**, les seules à être transmises par l'héritage entre tables ; la contrainte sera alors propagée de la table vers ses sous-tables. Pour vérifier les valeurs des instances, la fonction utilise une requête (**SELECT**) mais au lieu d'interroger seulement la table où s'applique la fonction (**SELECT * FROM TABLE**), elle interroge toutes les tables dérivées (**SELECT * FROM TABLE***).

EXPRESS	PostgreSQL

ENTITY personne
SUPERTYPE OF
(**ONEOF**(etudiant,salarie));
  son_nom : **STRING**;
  son_prenom : **STRING**;
  son_prenom2 : **OPTIONAL**
STRING;
  son_num_SS : **STRING**;
UNIQUE
  ur1 : son_num_SS;
END_ENTITY;
	CREATE TABLE personne (
	  son_nom : **VARCHAR NOT NULL,**
	  son_prenom : **VARCHAR NOT NULL,**
	  son_prenom2 : **VARCHAR,**
	  son_num_SS : **VARCHAR NOT NULL,**
	  CONSTRAINT ur **CHECK**
	(IsUnique('personne','son_num_SS',
	son_num_SS))
	);

Tableau 10: Contrainte d'unicité POSTGRESQL

Afin de rendre la solution plus simple à utiliser, nous avons créé des fonctions génériques. Il faut préciser qu'il faut autant de fonctions génériques (nommées *IsUnique*) qu'il y a de types. Chaque fonction générique possède trois paramètres dont deux sont "génériques". Ces deux paramètres sont des chaînes de caractères qui servent à transmettre respectivement le nom de la table et le nom de l'attribut sur lequel la contrainte est appliquée. Le dernier paramètre permet de transmettre la valeur de l'attribut à tester. Il est donc du même type. C'est pour cette raison qu'il faut autant de fonctions que de types de données. A partir de ces deux chaînes de caractères et de la valeur de l'attribut une commande SQL est construite puis exécutée pour vérifier l'existence d'une instance ayant la même valeur pour cet attribut.

3.2.5.3 Contrainte de cardinalité (inverse)

L'attribut **INVERSE** d'une entité *B* permet de recenser toutes les instances d'une entité *A* référençant via un attribut une instance donnée de l'entité *B*. Il permet également de restreindre leur nombre. Nous appelons l'entité *A* entité de la relation directe et l'entité *B* entité de la relation inverse. La fonction EXPRESS **USEDIN** permet de faire ce recensement sans pour autant représenter la contrainte de cardinalité inverse. Pour représenter l'attribut inverse en **POSTGRESQL,** nous avons représenté la fonction **USEDIN** qui nous renvoie dans une liste (un **ARRAY**) les entités concernées. Un attribut inverse peut donc être calculé en utilisant cette fonction dans une vue (cf. paragraphe 3.1.2). Si la cardinalité de la relation inverse est limitée à un nombre fixe d'instances référencées, une contrainte sera donc ajoutée sur la table de la relation directe pour limiter le nombre de références à des instances de la table de la relation inverse. Cette contrainte peut être introduite soit par une contrainte **CHECK** vérifiée à l'aide d'une fonction soit par un trigger qui se déclenche au moment de l'insertion des tuples dans la table de relation directe. La contrainte de cardinalité inverse sera représentée dans la table de la relation directe (représentant l'entité A dans cet exemple).

Le problème qui peut se poser pour représenter des contraintes de cardinalité liées aux attributs inverses dans PostgreSQL est celui des tests systématiques des données rangées dans la base de données PostgreSQL. Une contrainte de cardinalité, de type 1 :1, telle que par exemple la contrainte suivante : "un point ne peut être que le centre d'un seul cercle", ne peut pas être représentée de cette manière. En effet, la vérification tardive de contraintes dans le langage EXPRESS permet de tester le modèle après son instanciation complète. Dans le cas d'une base de données, la vérification de son intégrité est réalisée a priori. Dans cet exemple précis la création d'un tuple de la table *cercle* nécessite l'existence d'au moins une instance de la table *point*. La contrainte posée par l'attribut inverse nécessite également l'existence d'un *cercle* pour lequel ce *point* est le centre. Deux solutions sont possibles pour ce problème :

1- changer la cardinalité cercle/point de 1 :1 à 0 : 1,

2- utiliser un mécanisme de transactions. En effet, une transaction est constituée d'un ensemble de commandes qui, exécutées séparément, peuvent produire des incohérences dans une base de données, mais l'exécution de l'ensemble des commandes est tout à fait cohérente avec la base de données (principe du tout ou rien).

Nous avons retenu la deuxième solution car elle permet une représentation fidèle du modèle EXPRESS.

3.2.5.4 Contrainte globale (rule)

Une contrainte globale du langage EXPRESS permet de vérifier une propriété sur l'ensemble des instances d'une ou de plusieurs entités. Dans PostgreSQL le principe de contrainte globale n'existe pas. Nous avons proposé deux solutions pour représenter les contraintes globales. Néanmoins, les difficultés liées à la représentation de certaines fonctions EXPRESS dans PostgreSQL, et qui sont assez répandues dans les contraintes globales, ont fait que ces solutions n'ont pas été développées.

Représentation d'une contrainte globale par une contrainte check

Cette approche consiste à coder chaque règle globale par une fonction PGPLSQL qui renvoie ***True*** si l'expression est évaluée à vrai et ***false*** dans le cas contraire. Cette fonction sera donc utilisée dans la contrainte **CHECK** sur la table concernée par cette contrainte. Si la contrainte globale vise plusieurs entités du modèle EXPRESS cette même fonction traitera toutes les tables soumises à cette contrainte.

Représentation d'une contrainte globale par des triggers

Cette approche consiste à associer à chaque table visée par une contrainte globale un trigger. Ce trigger se déclenche lors de chaque insertion ou mise à jour sur les tables. Si le test de la contrainte est évalué à vrai le nouveau tuple est accepté sinon il est rejeté, car la contrainte est définie par une fonction du trigger. Si la contrainte globale vise plusieurs tables, il suffit de créer pour chaque table un trigger attaché à la même fonction de contrainte. Nous avons retenu la première solution car la deuxième réalise un très grand nombre de vérifications inutiles.

3.2.5.5 Intégrité référentielle

L'intégrité référentielle est une fonctionnalité du SGBD permettant de garantir l'intégrité et la cohérence de la base de données. Elle assure la cohérence des relations (des associations) entre les tables d'une base de données. Dans notre approche, cette cohérence doit être assurée par d'autres mécanismes que ceux existant dans PostgreSQL, qui ne permet pas le polymorphisme. L'intégrité référentielle doit être donc garantie par le concepteur de la base de données. Cette intégrité référentielle doit être garantie par un mécanisme qui permet, d'une part, de vérifier l'existence des références insérées dans toutes les sous-tables (la hiérarchie) de la table associée, et, d'autre part, de garantir, selon les règles définies pour l'association, l'intégrité de la base de données lors de la suppression d'une instance référencée. L'intégrité référentielle dans le premier sens, c'est-à-dire garantir l'existence d'une instance référencée, peut être réalisée de deux façons différentes :

1- utilisation d'une contrainte **CHECK** vérifiée à l'aide d'une fonction,

2- utilisation d'un trigger.

En revanche, pour vérifier l'intégrité dans l'autre sens, c'est-à-dire éviter d'avoir des références à des instances qui n'existent plus (après une suppression par exemple), nous utilisons les triggers. Chaque trigger est défini pour une table. Les événements possibles sont **INSERT** pour l'insertion, **UPDATE** pour la mise à jour et **DELETE** pour la suppression. Chaque trigger est associé à une fonction qui s'exécute au moment de son déclenchement.

Pour des raisons d'efficacité et de simplicité, nous avons choisi d'utiliser des triggers pour garantir l'intégrité référentielle dans les deux sens, par la création d'un trigger chacune des tables associées. Le trigger créé sur la table référençante se déclenche lors de l'insertion d'une instance pour vérifier l'existence de l'instance référencée. Il permet, soit de valider cette instance, si l'instance référencée existe dans la hiérarchie de la table associée, et de l'insérer, soit de rejeter l'instance et empêcher son insertion dans le cas

contraire. Pour le second trigger créé sur la table associée, plusieurs possibilités sont envisagées selon les règles imposées à la relation. En effet, lors de la suppression d'une instance référencée, on peut soit mettre à jour toutes les références vers celle-ci à la valeur **NULL**, soit réaliser une suppression en cascade de toutes les instances référençant l'instance à supprimer (pour plus de détails voir le paragraphe 4.1).

Pour alléger la gestion de l'intégrité référentielle nous avons préféré écrire des fonctions qui génèrent automatiquement ces triggers et les fonctions qu'ils utilisent. Les triggers et les fonctions associées sont créés dynamiquement à l'aide de la fonctions suivante : *Tab_Integrity_Constraint.*

La fonction *Tab_Integrity_Constraint* a la signature suivante :

Tab_Integrity_Constraint (**VARCHAR**, **VARCHAR**, **VARCHAR**) **RETURNS BOOLEAN** ; Le premier paramètre reçoit le nom de la table référençante. Le deuxième représente le nom de l'attribut d'association entre les deux tables et le troisième le nom de la table associée. Pour illustrer cela, prenons l'exemple précédent d'association entre les deux tables *A* et *B* (*A* référence *B* via *ab*). Cette fonction est appelée : *Tab_Integrity_Constraint* ('a','ab','b') ; sa première action est de créer une fonction qui permet de vérifier l'existence d'une référence (RID) dans la table A et toutes ses sous tables. Elle renvoie vrai si la référence existe et faux sinon. Dans la troisième étape une fonction de trigger est créée. Cette fonction fait appel à la précédente. En effet une fonction de trigger aurait suffit pour créer un trigger. Mais celle-ci est spécifique à chaque trigger (donc à chaque table). Pour simplifier la création des triggers pour les sous-classes on crée à chaque fois une nouvelle fonction de trigger qui appelle la même fonction créée pour sa superclasse. Cette fonction générique construit d'abord les textes du code SQL correspondant aux commandes de création de fonctions et de triggers, puis les exécute afin de créer effectivement ces fonctions et triggers.

3.2.6 Table de correspondance

Pour garantir l'intégrité des références lors du passage d'un modèle EXPRESS (fichiers physiques) à une base de données PostgreSQL, une table de correspondance entre les instances EXPRESS et leurs correspondants au niveau de PostgreSQL est créée. Cette table, appelée **INSTANCES_IDENTIFICATION,** possède les deux champs suivants : **EXPRESS_ID** et **POSTGRES_RID**. Elle est mise à jour au fur et mesure de l'insertion des instances dans la base de données PostgreSQL.

EXPRESS_ID	**POSTGRES_RID**
#1 **#2**	*243455* *243456*

Tableau 11 : exemple d'instance de la table de correspondance INSTANCES_IDENTIFICATION

Pour référencer les instances deux solutions sont envisageables :

1. utiliser cette table d'une manière temporaire. C'est-à-dire la peupler pendant une session juste pour mémoriser les correspondances entre les **OID** EXPRESS et les RID PostgreSQL. Son contenu sera donc écrasé à la fin de chaque session d'utilisation. Dans ce cas les références stockées dans la base de données sont bien les RIDs des instances référencées. La table d'identification d'instances ne sera donc utilisée que pour récupérer les bonnes références. Une fois toutes les références insérées, cette table peut être vidée de son contenu.

2. la deuxième solution consiste à garder le contenu de la table d'une manière persistante. Les références entre les tables passent donc par l'intermédiaire de cette table. Il faut noter que l'**OID** EXPRESS n'est pas unique dans tous les fichiers d'échange (on peut trouver le même **OID** #1 dans deux fichiers physiques différents). Pour pallier ce problème, il faudra augmenter cette table par l'ajout d'autres colonnes incluant les informations définies dans les en-têtes des fichiers physiques.

Dans les deux cas, une fonction permettant de récupérer le RID de la référence sera créée. Elle récupère soit le RID de l'instance référencée (premier cas) soit le RID du tuple de la table intermédiaire contenant cette référence (deuxième cas). Le paramètre d'entrée de cette fonction est l'**EXPRESS_ID** qui prend ses valeurs dans la colonne, portant le même nom, de la table de correspondance **INSTANCES_IDENTIFICATION** (voir Tableau 11). Pour des raisons de simplicité nous privilégions la première solution.

Lors de la génération de code SQL à partir d'un modèle EXPRESS, à chaque insertion d'une instance dans la base de données on génère des commandes d'insertion des références croisées, pour mettre à jour la table des références **INSTANCES_IDENTIFICATION**.

3.3 Représentation des fonctions et des procédures

PostgreSQL offre la possibilité de coder des fonctions dans une base de données. Cette fonctionnalité permet :

- le renforcement de l'intégrité de la base de données (traitement dépendant exécuté dans un même bloc) par l'écriture de contraintes d'intégrité complexes ;

- l'augmentation de la performance de la base de données (réduction du trafic réseau entre le client et le serveur), car les procédures sont exécutées directement sur le serveur ;

- la réutilisation du code (code partagé).

Dans PostgreSQL les fonctions peuvent être écrites dans les différents langages suivants : SQL, PL/PGSQL, PL/TCL et PL/PERL. Pour éviter la compilation externe nous avons préféré utiliser que les langages SQL et PL/PGSQL pour écrire nos fonctions (cf. paragraphe 3.2.4 du chapitre 2).

Les fonctions de dérivation ainsi que celles qui sont utilisées dans l'expression de contraintes seront codées par le concepteur de la base de données. La représentation de procédures n'étant pas permise dans PostgreSQL, celles-ci doivent être représentées par une ou plusieurs fonctions.

4 Représentation des instances du niveau contenu dans PostgreSQL

D'un point de vue structurel, le modèle du contenu ne diffère pas beaucoup de celui du dictionnaire. En effet, les principaux concepts du langage EXPRESS sont représentés par le modèle PLIB. Cependant, des concepts plus spécifiques au modèle PLIB ont été rajoutés. Parmi lesquels l'organisation de l'héritage et le polymorphisme. Nous détaillons dans le paragraphe suivant ses deux points.

4.1 Polymorphisme

La représentation du polymorphisme est identique à celle utilisée au niveau du dictionnaire. En effet, l'héritage de classes est représenté par l'héritage de tables. L'association entre les tables est représentée d'une manière analogue à celle donnée dans le paragraphe 3.1.1.5. Cependant, une différence subsiste au niveau des règles que l'on peut imposer à ces relations. Cela est dû au fait qu'à ce niveau deux types de relations sont possibles :

1- la relation d'association,

2- la relation de composition.

La différence majeure entre ces deux types de relations concerne la suppression. Dans une relation d'association, la suppression d'une instance référençante n'induit pas la suppression de l'instance référencée. En revanche, dans une relation de composition la suppression d'une instance référençante (i.e. un composé) induit la suppression, en cascade, de toutes les instances référencées (i.e. les composants). Cette différence de sémantique ne provoque pas une différence au niveau de la représentation de ces deux types de relations. La différence apparaît en fait au niveau des contraintes d'intégrité appliquées sur chacune des relations. Nous avons donné dans la paragraphe 3.2.5.5 la solution retenue pour assurer l'intégrité référentielle : elle consiste à utiliser des triggers. Cette même solution est adaptée pour garantir l'intégrité référentielle dans le cas d'une relation de composition.

4.2 Tables représentant les familles de composants

Au niveau du contenu, l'instanciation des familles de composants est limitée aux classes feuilles de la hiérarchie. Les tables représentant les familles de composants peuvent être représentées de deux manières différentes. La première représentation consiste à se limiter à la représentation des classes feuilles uniquement. La seconde consiste à représenter la hiérarchie complète des familles de composants mais en limitant l'instanciation aux classes feuilles. Cette seconde solution facilite le processus

de sélection de composants (sélection directe sur le contenu sans passer par le dictionnaire de données) en procédant d'une manière hiérarchique. La présence des classes intermédiaires à différents niveaux (entre la classe du haut niveau et les classes feuilles) a pour objectif la structuration de données. Elle nécessite la prise en compte de la représentation des propriétés à différents niveaux de la hiérarchie (propriétés visibles en particulier) (voir Figure 45).

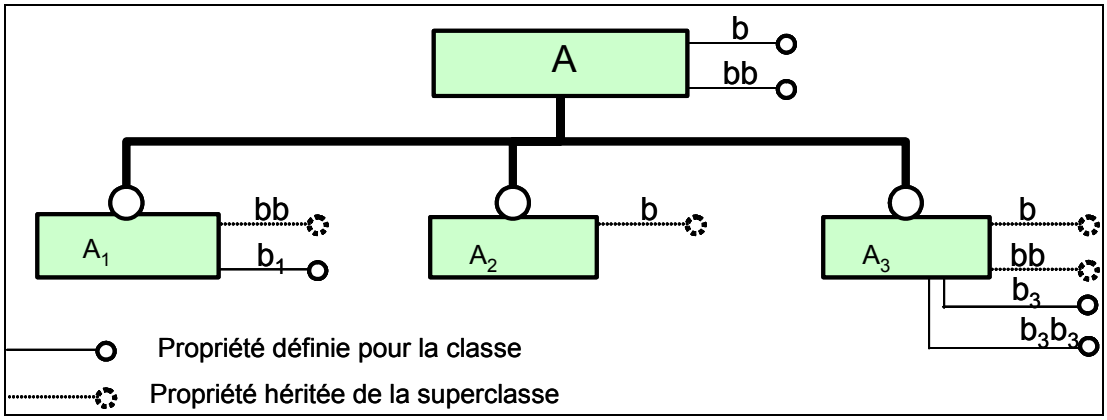


Figure 45 : représentation des propriétés visibles et des propriété applicables dans le modèle PLIB

Comme le montre la Figure 45, les deux propriétés *b* et *bb* sont définies pour la classe *A*. Ces propriétés ne sont pas héritées par toutes les sous-classes de *A* (la propriété *b* est héritée par A_2, A_3 et la propriété *bb* est héritée par A_1, A_3). Dans le modèle PLIB ces deux propriétés sont déclarées comme des propriétés visibles. C'est-à-dire qu'elles sont visibles par toutes les sous-classes. Elles ne sont applicables qu'au niveau des sous-classes pour lesquelles elles présentent une signification. Par exemple, si on considère la famille des vis, les propriétés diamètre, longueur et hauteur de la tête sont significatives pour la plupart des sous-classes des vis. Cependant, les vis sans tête ne possèdent pas, par définition, une hauteur de la tête. Cette dernière propriété doit être déclarée comme étant une propriété visible au niveau de la classe racine. Elle sera déclarée comme applicable pour quelques sous-classes telles que les vis à tête carré, les vis à tête hexagonale, etc. Ces informations sont formellement représentées au niveau du dictionnaire de données PLIB. Cependant, le mécanisme d'héritage habituel, et en particulier celui de PostgreSQL, ne permet pas de représenter de telles informations. L'adoption de cette approche nécessite donc de ne représenter, au niveau de la racine, que les propriétés communes à toutes les sous-classes. De plus, ces propriétés doivent être incluses dans la liste des propriétés fournies dans le contenu (propriétés effectivement représentées dans les fichiers physiques) (cf. paragraphe 4.1.3).

Le choix de la solution à utiliser pour représenter les familles de composants doit être établi en se basant sur un autre aspect : celui de la sélection de composants. Nous discutons dans le paragraphe suivant de cet aspect.

4.3 Sélection des composants au niveau du contenu

Nous avons donné dans le paragraphe précédent deux manières différentes pour la représentation des familles de composants. C'est en fonction de ces représentations

que la sélection des composants s'organise. A chaque approche de représentation des familles de composants correspond une méthode de sélection spécifique.

La première méthode de sélection est la sélection générique. Elle est utilisée lorsque les familles de composants ne sont représentées que par les classes feuilles (instanciables). Cette sélection générique vise à garder d'une certaine manière la possibilité d'une sélection hiérarchique même si cette dernière n'est pas physiquement représentée au niveau du contenu. Pour ce faire, les requêtes doivent être calculées lors de la sélection des composants. En effet, ce calcul des requêtes se base d'une part sur la requête initiale formulée par l'utilisateur et d'autre part sur les informations du schéma du contenu rangées au niveau de l'ontologie. Les étapes de ce calcul sont les suivantes :

1- calcul des sous-familles supportant les propriétés incluses dans la clause "**WHERE**" de la requêtes,

2- calcul des propriétés appartenant à toutes ces classes (intersection entre les schémas des différentes tables représentant ces classes),

3- génération d'une requête par sous-famille,

4- consolidation du résultat de ces différentes sous-requêtes par une union.

A la fin de ces étapes les instances répondant aux conditions de la sélection seront regroupées dans une seule table virtuelle. Afin de pouvoir connaître la classe effective de chaque instance, PostgreSQL prévoit une solution consistant à ajouter automatiquement pour chaque table créée une colonne appelée **OIDTABLE**. Cette colonne permet de récupérer, d'une manière dynamique, l'**OID** de la table d'une instance dans une requête. Pour obtenir le nom de la table, il suffit de faire une jointure avec la méta-table *pg_class* qui décrit les tables (leurs noms, leurs attributs, …) au niveau de la métabase (partie 1 de l'architecture de la base de données). Cette première sélection générique est une sélection dynamique.

La seconde approche de sélection est statique. Elle peut s'appliquée à un contenu dont les familles de composants sont représentées selon la seconde approche donnée dans le paragraphe précédent (hiérarchie de tables). A la différence de l'approche dynamique de sélection, dans cette approche la plupart des calculs des propriétés et des sous-familles sont effectués pendant la création du schéma du contenu. Ce sont ces calculs qui aboutissent à une représentation hiérarchique des données (calcul de toutes les propriétés communes à toutes les sous-classes pour les représenter au niveau de la classe racine). Une fois la hiérarchie de classes représentée directement dans PostgreSQL, les requêtes deviennent plus simples si les sélections portent sur les propriétés communes représentées au niveau de la racine. En fait, PostgreSQL intègre très bien la sélection hiérarchique, ce qui permet d'optimiser les requêtes car elles ne sont pas calculées à chaque sélection. Cela aide à améliorer la performance du gestionnaire PLIB. L'inconvénient de cette seconde approche est qu'elle ne permet pas de réaliser des sélections hiérarchiques en se basant sur des propriétés visibles qui ne sont pas représentées dans la table racine.

Pour ne pas empêcher les sélections basées sur les propriétés visibles représentées effectivement au niveau du contenu, nous avons conjugué les deux solutions précédentes en gardant la représentation hiérarchique de familles de composants. Les requêtes s'exécutent selon la seconde approche si elles ne comportent pas de propriétés visibles. Dès lors qu'une requête comporte des propriétés visibles (non représentées à la racine de la hiérarchie) le système a recours à la première approche. Nous pouvons qualifié cette approche d' "approche mixte" ou de méthode de sélection "semi dynamique".

Nous avons signalé dans le paragraphe précédent que le choix de la représentation des familles de composants au niveau du contenu est motivé également par la simplicité de sélection des composants. En choisissant une sélection semi dynamique, le choix d'une représentation hiérarchique des familles de composants doit être reél. Il a également l'avantage de tirer profit de l'efficacité du mécanisme d'héritage dans PostgreSQL.

5 Connexion avec un SGDT

L'objectif des travaux de cette thèse était l'intégration du modèle PLIB dans un SGDT. Nous avons présenté, dans les chapitres précédents, les différentes approches possibles pour implémenter le modèle PLIB dans une base de données cible. Ensuite, une fois que les méthodes d'implémentation choisies, nous avons choisi PostgreSQL pour les mettre en œuvre. Maintenant que nous avons abouti à une implémentation complète d'une maquette (dictionnaire et contenu), la dernière étape est son intégration à un SGDT. Deux approches peuvent être suivies pour réaliser cette intégration :

1- l'intégration des deux modèles dans une même base de données,

2- l'intégration des deux modèles à partir de bases de données différentes.

La première approche consiste à implémenter le modèle PLIB au sein même de la base de données gérée par le SGDT. L'implémentation de ce modèle sera réalisée d'une manière similaire à celle employée lors de son implémentation dans PostgreSQL (en l'adaptant aux spécificités du système cible). De cette manière les composants peuvent être facilement référencés au sein des structures des produits gérées par le SGBD. Ce référencement est réalisé à l'aide de pointeurs internes.

La seconde approche consiste à représenter le modèle PLIB dans une base de données différente que celle utilisée par le SGDT, telle que PostgreSQL, par exemple. Le gestionnaire de données sera soumis au contrôle global du SGDT à l'instar des autres systèmes utilisés dans les entreprises tels que les systèmes CAO, GPAO, etc. Le référencement des composants au sein des structures de produits est réalisé par l'intermédiaire d'identificateurs externes. Ces identificateurs sont en fait les identificateurs de composants au sein du gestionnaire de données de composants.

6 Conclusion

Nous avons présenté dans ce chapitre la mise en œuvre de l'implémentation du modèle PLIB, développée dans les chapitres précédents. Le système PostgreSQL a été choisi pour réaliser cette mise en œuvre. Après avoir donné les raisons de ce choix, nous avons commencé par la description et la discussion de représentation des différents concepts du langage EXPRESS au sein de PostgreSQL. Le modèle PLIB étant représenté en langage EXPRESS, la représentation des différents concepts du langage EXPRESS définit directement l'implémentation du dictionnaire PLIB dans PostgreSQL. Nous avons discuté à cet effet les différentes solutions possibles pour réaliser l'implémentation de la majorité des concepts d'EXPRESS dans PostgreSQL.

Les solutions apportées pour la modélisation des concepts EXPRESS sont les mêmes que celles utilisées pour la représentation des concepts PLIB. En effet, une majorité de ces concepts est commune. Cependant, le contenu comporte un certain nombre de concepts absents dans EXPRESS. Nous avons décrit ces concepts spécifiques au modèle PLIB qui concernent essentiellement les relations entre les familles de composants et la représentation de leurs propriétés. Pour les relations entre propriétés nous avons identifié deux types de relations : la relation d'association et la relation de composition. Nous avons également donné les solutions appropriées à chacune de ces relations. Pour la représentation des propriétés nous avons choisi d'utiliser la hiérarchie de tables de PostgreSQL pour représenter les familles de composants, et donc de ne représenter que les propriétés fournies communes à toutes les sous-classes, au niveau des classes racines. Les propriétés visibles sont prises en compte dans la sélection de composants.

La sélection de composants constitue un point important dans la création du gestionnaire des données de composants. Nous avons donc présenté brièvement ce point. La possibilité de sélection de composants en se basant sur les propriétés visibles doit être garantie car elle est très répandue dans ce domaine (données de composants). La sélection se déroule de deux manières différentes selon l'existence ou non de propriétés visibles dans les requêtes. Les familles de composants étant représentées dans une hiérarchie de tables, les sélections ne portant pas sur les propriétés visibles sont exécutées d'une manière directe (statique). En revanche, si une requête contient des propriétés visibles, une procédure de calcul dynamique de requête est déclenchée. Elle permet de générer et de lancer des sous-requêtes sur les différents sous-classes ayant en commun ces propriétés visibles puis de consolider les résultats issus de ces sous requêtes. Cette dernière approche est une approche dynamique. L'approche globale de sélection conjugue l'approche statique et l'approche dynamique de sélection. Elle est dite semi-dynamique.

Nous avons auusi évoqué deux approches permettant l'intégration du modèle PLIB dans un SGDT. La première approche est l'intégration interne au SGDT. Elle consiste à représenter les données de composants au sein même de la base de données contenant les données de produits et gérée par le SGDT. Cette implémentation se fait en se basant sur l'approche de représentation du modèle PLIB au sein d'une base de données cible développée dans ce mémoire. La seconde approche est l'intégration externe au SGDT. Elle consiste à créer un gestionnaire de données de composants indépendant (implémenté sur PostgreSQL, par exemple) et le mettre sous le contrôle du SGDT (à l'aide d'une interface entre les deux systèmes). Dans le premier cas, les

références des composants au sein des structures de produits sont assurées par des liens internes, dans l'autre cas, ce sont des identificateurs externes qui l'assurent.

Conclusion générale

Ce chapitre conclut nos travaux sur l'intégration des modèles de données de composants et des modèles de données de produits. Nous y récapitulons notre contribution et présentons quelques perspectives.

Contributions

Dans notre étude, nous sommes parti d'une situation dans laquelle les données de composants étaient représentées selon le modèle PLIB en suivant une approche de modélisation implicite par *propriétés* et, dans laquelle les données de produits étaient représentées par des modèles explicites où chaque produit ou composant est individuellement identifié et associé à une modélisation analytique par *représentations*. En étudiant différentes approches d'intégration entre les données de composants et les données de produits, nous avons développé les points suivants :

1. proposition d'un modèle explicite et conversion

Le modèle PLIB comprend deux parties principales, le modèle de dictionnaire de données (ou modèle de l'ontologie) et le modèle de contenu (modèle des instances). La première version de ce dernier modèle utilisait au départ une approche implicite de représentation des instances (représentation des instances par des expressions et des contraintes d'intégrité).

Nous avons constaté que l'implémentation du modèle implicite de PLIB était complexe et coûteuse. Ceci nous a amené à proposer une représentation explicite permettant de rapprocher la représentation des instances de celle utilisée dans les bases de données : c'est-à-dire par des n-uplets. Elle permet également de se rapprocher de la représentation des articles dans les SGDT où chacun est identifié par un numéro d'article (*part number*). Cette approche consiste à représenter les instances par une énumération des couples (propriété, valeur). Toutes les instances y sont énumérées d'une manière exhaustive. Cette approche de représentation a été ensuite intégrée dans la norme internationale PLIB parue sous le numéro de [ISO 13584-25:2004].

Nous avons ensuite montré comment cette représentation explicite pouvait être générée en calculant les formules et expressions décrivant les propriétés de composants puis en appliquant les contraintes d'intégrité associées implicitement au composant. La tâche principale dans la conversion qui aboutit à la représentation explicite est celle de l'évaluation des expressions (numériques pour le calcul des valeurs des propriétés et logiques pour l'application des contraintes d'intégrité). Pour réaliser cette évaluation, trois approches ont été étudiées : l'évaluation par programme générique, externe aux

modèles (qui récupère l'information à traiter en exploitant les modèles et en parcourant les fichiers des instances correspondants), la programmation évènementielle appliquée au langage EXPRESS (utilisation des attributs dérivés d'EXPRESS) et l'évaluation par génération de programmes compilables à partir des modèles et des fichiers d'instances. Nous suggérons finalement de retenir la deuxième approche qui a l'avantage de permettre une programmation modulaire.

2. conception d'une architecture de gestion

La représentation explicite simplifie la tâche de représentation des données de composants PLIB dans une base de données cible. Nous avons alors étudié plusieurs architectures de gestion des données. L'approche hybride a été retenue. Celle-ci permet à la fois la représentation de l'ontologie au niveau des entités du modèle EXPRESS (niveau du méta-modèle) et du contenu au niveau des instances (chaque instance des composants devient un tuple). Ceci a permis le développement d'une nouvelle architecture de base de données permettant, en plus de la représentation des instances (représentation ordinaire des données dans les SGBD), une représentation des schémas conceptuels de ces données à travers l'ontologie. En effet, l'ontologie, telle qu'elle est définie par le modèle PLIB, constitue une extension d'un schéma conceptuel dans lequel chaque concept est identifié par un GUI. Chaque classe y est décrite par le maximum de propriétés possibles dans le contexte le plus général d'utilisation. Cette description étendue d'une classe n'implique pas nécessairement la représentation effective de toutes ses propriétés au niveau du contenu. Les deux avantages essentiels de cette approche sont, d'une part, la possibilité de représentation, dans la base de données cible, de toutes les informations portées habituellement par les schémas conceptuels (absents dans les implémentations classiques), et, d'autre part, comme cela a été prouvé depuis, d'aider à l'intégration automatique de bases de données hétérogènes.

3. Implémentation

Pour valider notre proposition d'architecture, nous avons étudié plusieurs approches pour réaliser un passage automatique du modèle PLIB vers la base de données cible. Toutes sont basées sur le principe de la génération automatique du code SQL correspondant au schéma de la base de données cible. Nous avons finalement réalisé une maquette d'implémentation dans le SGBD PostgreSQL ce qui a permis de montrer la faisabilité de nos propositions.

4. Intégration des données de produits et des données de composants

Dans l'intégration finale des données de produits et de composants, chaque composant étant désormais explicite, il devient alors possible de référencer les identifiants des composants dans les modèles de produits. Ce référencement peut s'effectuer dans deux contextes différents :

- l'intégration des deux modèles dans une seule et unique base de données (celle gérée par le SGDT lui-même),

- l'intégration des deux modèles dans deux bases de données distinctes.

Le référencement peut être représenté par des références internes (pointeurs, clés étrangères, …) dans le premier cas et par des références externes (identifiant universel) dans le second.

Perspectives

Ayant proposé une implémentation du modèle PLIB au sein d'un SGBD, le seul langage de requêtes permettant sa manipulation, à l'heure actuelle, est le langage du SGBD cible, c'est-à-dire le langage SQL. Afin d'exploiter la disponibilité de l'ontologie, nous proposons le développement et l'intégration d'un langage de requêtes qui permettrait la réalisation des requêtes au niveau de l'ontologie elle-même. Ce langage permettrait l'interrogation de la base de données des composants sans se soucier de la manière dont ces données sont implémentées.

Nous proposons également de faire une étude sur l'optimisation de la représentation choisie en vue de favoriser l'optimisation de requêtes exprimées dans ce langage. L'optimisation de la sélection des composants nécessiterait également une interface graphique permettant la génération automatique de requêtes.

Le développement d'une interface (API) entre le gestionnaire PLIB (SGBD-PLIB) et le SGDT est également nécessaire dans le cas d'une intégration dans deux bases de données distinctes. Cette API pourrait être générée automatiquement pour *la partie ontologie* (à partir du schéma EXPRESS correspondant). Elle devra être définie "à la main" pour *la partie contenu* qui fait l'objet d'une représentation différente du schéma EXPRESS.

Une des plus importantes perspectives est le développement de l'approche de base de données basée sur une ontologie pour étendre son utilisation à d'autres domaines. Elle peut être utilisée à des fins d'échanges de données entre bases de données hétérogènes ou pour permettre l'intégration de modèles de données hétérogènes.

Bibliographie

[Abrial 74] J. R. Abrial, "*Data semantics*", In Klimbie and Koffeman, editors, Data Base Management. North-Holland Publ., 1974.

[Aho 77] A. Aho, J. Ullman, "*Principles of Compiler Design*", Addition-Wesley Publishing Company, 1977.

[Aho et al 72] A.V. Aho, J.D. Ullman, "*The Theory of Parsing, Translation and Compiling*", Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[Aho et al 86] A.V. Aho, R. Sethi, J.D.Ullman, "*Compilers -- Principles, Techniques and Tools*", Addison-Wesley, 1986.

[Ait-Ameur et al 00] Y. Aït-Ameur, G. Pierra, E. Sardet, "*An object approach to represent behavioural knowledge in heterogeneous information systems*", Proceeding of the International Conference on Object-Oriented Information Systems, London, pp. 315-339, 2000.

[Ait-Ameur et al 03] Y. Ait-Ameur, G. Pierra, "*Representation of context free languge using data oriented model*", Rapport de recherche, LISI/ENSMA, 2004.

[Ait-Ameur et al 95a] Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, J.-C. Potier, "*Specification and Metaprogramming in the EXPRESS Language*", Proceeding of the International Conference on Software Engineering and Knowledge Engineering SEKE'95, Rockville, 1995.

[Ait-Ameur et al 95b] Y. Ait-Ameur, G. Pierra, E. Sardet, "*Using the EXPRESS Language for Metaprogramming*", Proceeding of the The 3rd International Conference of EXPRESS User Group EUG'95, Grenoble, 1995.

[Aklouf et al 03] Y. Aklouf, G. Pierra, Y. Aït-Ameur, H. Drias, "*PLIB Ontology For B2B Electronic Commerce*",.Proceedings of CE'2003, Special track on Data Integration in Engineering, Madeira, Portugal, edited by R. Jardim-Gonçalves and J. Cha and A. Steiger-Garçao, Madeira, Portugal, UNINOVA, A.A. Balkema, pp. 269-278, 2003.

[Ambler 00] Scott W. Ambler, "*Mapping Objects To Relational Databases*", An AmbySoft Inc. White Paper, http://www.ambysoft.com/mappingObjects.html, 2000.

[Arens et al 93] Y. Arens, C. Chee, C.-N Hsu, C.A Knoblock, "*Retrieving and Integrating Data from Multiple Information Sources*", International Journal on Intelligent and Cooperative Information Systems. In press, 1993.

[Atzeni et al 97] P. Atzeni, R. Torlone, "*MDM: a Multiple-Data-Tool for the Management of Heterogeneous Database Schemes*", Proceeding of the International Conference on Management Data and Symposium on Principals of Database Systems, pp.528-531, ACM SIGMOD, Tucson, Arizona,1997.

[Bacha et al 02] R. Bacha, B. Yannou, "*A Methodology for Process Information Modelling : Towards a Process Technical Data Management*", Integrated Design and Manufacturing in Mechanical Engineering, Kluwer Academic Publishers, Dordrecht/Boston/London, Chedmail, 2002.

[Baker 97] N. Baker, J-M Le Goff., "*Meta Object Facilities and their Role in Distributed Information Management Systems*". Proc of the EPS ICALEPCS97 conference, Beijing, China, November 1997.

[Bancilhon 94] F. Bancilhon, G. Ferran, "*OQL: an Object Query Language for All Seasons*", Proc. of Intl. Symp. on Advanced Database Technologies and Their Integration, pp. 253--263, Nara, Japan, October 1994.

[Barthès et al 79] Jean-Paul Barthes, Michel Vayssade, Monika ZNAMIEROWSKA, "*Property driven databases*", Proc. 6th IJCAI, Tokyo, 1979.

[Bellatreche 03] L. Bellatreche, G. Pierra, D. Nuyen Xuan, H. Dehainsala, "*An Automated Information Integration Technique using an Ontoly-based Database Approach*", Proceeding of the 10th ISPE International Conference On Concurrent Engineering: Research And Applications, A.A. Balkema Publishers, pp. 217-224, Madeira, Portugal, 2003.

[Bellatreche 04] L. Bellatreche, G. Pierra, D. Nuyen Xuan, H. Dehainsala, "*Intégration de sources de données autonomes par articulation a priori d'ontologie*", Acted du XXII^{e} INFORSID, Inforsid, pp. 283-298, Biarritz, France, 2004.

[Benchimol, 93] G. Benchimol, "*L'entreprise étendue*", Ed. Hermes, Paris, 1993.

[Benzaken et al 93] V. Benzaken, A. Doucet, "*Bases de données orientées objet: origines et principes*", Armand Colin, 126 pp, 1993.

[Birtwistle et al 73] G. Birtwistle, O. Dahl, B. Myhrhaug, K. Nygaard, "*Simula begin*", Petrocelli Charter, New York, 1973.

[Bodington 00] R. Bodington, L. Lämmer, "*Integrating the Enterprise using the PDM Schema and PDM Enablers*", proceeding of PDT Days, pp. 13-20, QMS edition, Netherlands, 2000.

[Booch 86] G. Booch, "*Software engineering with ADA*", Benjamin/Cummings Publishing Compagny Inc., Menlo Park, 1986.

[Booch 91] G. Booch; "*Object Oriented Design*", Redwood City, Calif.: Bejamin/Cummings, 1991.

[Bouazza 95a] M. Bouazza, "*Le langage EXPRESS*", Editions Hermès, 1995.

[Bouazza 95b] M. Bouazza, "*la Norme STEP*", Editions Hermes, 1995.

[Bouzeghoub et al 97] M. Bouzeghoub, G. Gardarin, P. Valduriez, "*Les objets*", Editions Eyrolles, 1997.

[Castano et al 97] S. Castano, V. De Antonellis, "*Semantic Dictinnary Design for Database Interoperability*", Proceeding of the 13th International Conferce on Data Engeneering, IEEE Computer Society Press, Birmingham, UK, 1997.

[Challal 03] S. Challal, "*Etude de la représentation procédurale dans une base de données ontologique, application à la base de données PLIB*", DEA T3IA, Université de Poitiers, 2003.

[Chambolle 98] F. Chambolle, A. Bernard, F. Petit, P. Germain-Lacour, J. Bocqet, "*STEP AP214 et la gestion de la diversité dans l'industrie automobile*", In journée PRIMECA-SGDT : Offres, Méthodes et Outils, Grenoble - France, 1998.

[Chambolle 99] F. Chambolle, "*Un modèle produit piloté par les processus d'élaboration : application au secteur automobile dans l'environnement STEP*", Thèse de Doctorat, l'Ecole Centrale de Paris, 1999.

[Chen 76] Peter Pin-Shan Chen, "*The Entity-Relationship Model. Toward Unified View of Data*", ACM Transaction on Database Systems, 1(1) : 9-36, 1976.

[Chochon 03] J. Chochon, Y. Ait-Ameur, G. Pierra, J.-C. Potier, "*Reducing parts diversity in product design: a data centered approach"*, Proceeding of the 10th ISPE International Conference On Concurrent Engineering: Research And Applications, A.A. Balkema Publishers, pp. 311-318, Madeira, Portugal, 2003.

[CIMdata 97] CIMdata Inc., "*Product Data Management : the definition. An introduction to concepts, Benefits, and Terminology*", (http://www.cimdata.com/), 1997.

[CIMdata, 95] CIMdata Corp., "*CIMdata Glossary*", (http://www.cimdata.com/), 1995.

[Coad et al 92] P. Coad, E. Yourdon, "*Object-oriented analysis*", Prentice-hall ed, 1992.

[Colmerauer et al. 83] A. Colmerauer, H. Kanoui, M. Van Caneghem, "*PROLOG, bases théoriques et développements actuels*", Techniques et Sciences Informatiques, 2(4): 271-311, 1983.

[Dahl et al 66] O. Dahl, K. Nygaard, *"Simula, an Algol-based simulation language"*, Comm. ACM, Vol 9, pp. 671-678, 1966.

[Dehainsala 04] H. Dehainsala, "*Base de données à base ontologique*", Acted du XXII^{e} INFORSID, Inforsid, pp. 539-540, Biarritz, France, 2004.

[Delmal 00] Pierre Delmal, "*SQL2 - SQL3 Applications à Oracle*", De Boeck Université, 2000.

[Delobel et al 91] C. Delobel, C. Lécluse, P. Richard, "*Bases de données: des systèmes relationnels aux systèmes à objets*", InterEditions, 460 pp, 1991.

[Deransart et al 88a] Pierre Deransart, Martin Jourdan, Bernard Lorho, "*Attribute Grammars: Definitions, Systems and Bibliography*", volume 323 of Lect. Notes in Comp. Sci. Springer-Verlag, August 1988.

[Deransart et al 88b] Pierre Deransart, Martin Jourdan, "*Attribute Grammars and their Applications (WAGA)* ", volume 461 of Lect. Notes in Comp. Sci., Paris, Springer-Verlag, September 1990.

[Dittrich 97] K. R. Dittrich, A. Geppert, "*Object-Oriented DBMS and Beyond*", Conference on Current Trends in Theory and Practice of Informatics, pp. 275-294, 1997.

[Duris 95] Etienne Duris, Didier Parigot, Martin Jourdan, "*Mises à jour destructives dans les grammaires attribuée*", INRIA, rapport de recherche N° 2686, Octobre 1995.

[El Hadj Mimoune et al 00] M. El Hadj Mimoune, Y. Ait-Ameur, G. Pierra, J.-C. Potier, "*Integration of component descriptions in product data management systems*", Proceeding of the ISPE International Conference on Concurrent Engineering "CE2000", Lyon, 2000.

[El Hadj Mimoune et al 01a] M. El-Hadj Mimoune, Y. Ait-Ameur, G. Pierra, "*Modélisation du contenu des catalogues de composants industriels : de la représentation implicite à la représentation explicite*", Proceeding of the ISPS'2001, pp. 15-26, Alger, 2001.

[El Hadj Mimoune et al 01b] M. El-Hadj Mimoune, G. Pierra, Y. Aït-Ameur, "*Une approche pour l'échange entre bases de données hétérogènes basée sur des méta-modèles génériques exprimés en langage EXPRESS*", Actes de la Journée de Travail Bi-Thématique du GDR-PRC I3, pp. 229-246 Lyon, 13 Décembre, 2001.

[El Hadj Mimoune et al 03] M. El-Hadj Mimoune, G. Pierra, Y. Aït-Ameur, "*An Ontology-based approach for exchanging data between heterogeneous database systems*", ICEIS'03, vol. 4, ESEO, pp. 512-524, Angers, 2003.

[Engelfriet 84] J. Engelfriet, "*Attribute grammar : attribute evaluation methods*", In B. Lorho, editor, Methods and Tools for Compiler Construction, pp. 103-138, Cambridge University Press, 1984.

[Farreny 85] H. Farreny, "*Les systèmes experts, principes et exemples*", Editions CEPADUES, 1985.

[Feru et al 98] F. Feru, C. Viel, "*Echanger avec le protocole d'application 203 de STEP : Echange et partage de données CAO et GDT*", Association GOSET, 1998.

[Fiallet et al 92] F. Fiallet, J-M Augier, M-J Bellosta, "*Les langages de requêtes des systèmes de gestion de bases de données orienté-objet*", dans le livre : Les base de données de données avancées, pp. 196-211, Imad Saleh, edition HERMES, 1992.

[Flater et al 99] David Flater, K.C. Morris, "*Testability of Product Data Management Interfaces*," NISTIR 6429, National Institute of Standards and Technology, Gaithersburg, MD, 1999.

[Foulard 98] C. Foulard, "*L'Entreprise Communicante*", Editions Hermès, pp. 255-265, 1998.

[Gardarin 01] G. Gardarin, "*Bases de données*", Editions Eyrolles, Paris, 2001.

[Gardarin et al 94] G. Gardarin, M. Bouzeghoub, and P. Valduriez, "*Les Objets*", EYROLLESE, 1994.

[Graham 91] I. Graham, "*Object Oriented Methods*", Addison-Wesley Publishing Company, 1991.

[Habrias 88] H. Habrias, *"Le modèle relationnel binaire : La méthode NIAM"*, Eyrolles, 1988.

[Henriques et al 02] Pedro Henriques, Maria Varanda Pereira, Marjan Mernik, Mitja Lenic, Enis Avdicausevic, Viljem Zumer, "*Automatic Generation of Language-based Tools*", Electr. Notes Theor. Comput. Sci. 65(3): 2002.

[Herbst 94] A. Herbst, *"Long-Term Database Support for EXPRESS Data",* In 7th Int'l. Working Conf. on Scientific and Statistical Database Management, IEEE Computer Society Press, pp. 207-216, Charlottesville, VA, 1994.

[Höfling 99] Björn Höfling, "*Reuse of STEP product data for the generation of technical documentation*", in Nada Matta, Rose Dieng (eds.), Proceedings of the Workshop on Knowledge Management and Organizational Memories, at the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, 1999.

[Horntvedt et al 00] O. Horntvedt, J. Storvik, "*Product Data Management in Small and Medium Sized Companies*", proceeding of PDT Days, pp. 255-261, QMS edition, Netherlands, 2000.

[Huau 00] Pascal Huau, "*Opportunities for use of the STEP Automotive Application Protocol (AP214) in Space Industry*", proceeding of PDT Days, pp. 59-66, QMS edition, Netherlands, 2000.

[IEC 1360-4:1995] International Electrotechnical Commission, IEC 1360- 4, "*Standard Data Element Types With Associated Classification Scheme for Electric Components*", , 1995

[IGES 80] IGES, "*Initial Graphics Exchange Specification*",: ANSI Y 14.26M, ANSI – American National Standard Institute, USA, 1980.

[ISO 10303-1:1994] ISO 10303-1, "*Industrial automation systems -- Product data representation and exchange -- Part 1: Overview and Fundamental Principles*", ISO/IEC, Geneva, Switzerland, 1994.

[ISO 10303-11:1994] ISO 10303-11, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual*", ISO/IEC, Geneva, Switzerland, 1994.

[ISO 10303-11:2000] ISO 10303-11, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual*", ISO/IEC, Geneva, Switzerland, 2000.

[ISO 10303-203:1994] ISO 10303-203, "*Industrial automation systems - Product data representation and exchange -- Part 203: Application protocol: Configuration Controlled Design*", ISO/IEC, Geneva, Switzerland, 1994.

[ISO 10303-21:1994] ISO 10303-21, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure (Physical file)* ", ISO/IEC, Geneva, Switzerland, 1994.

[ISO 10303-212:2001] ISO 10303-212, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 212: Application protocol: Electrotechnical design and installation*", ISO/IEC, Geneva, Switzerland, 2001.

[ISO 10303-214:2003] ISO 10303-214, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 214: Application protocol: Core data for automotive mechanical design processes*", ISO/IEC, Geneva, Switzerland, 2003.

[ISO 10303-22:1997] ISO 10303-22, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 22: Implementation methods: Standard Data Access Interface*", ISO/IEC, Geneva, Switzerland, 1997.

[ISO 10303-224:2001] ISO 10303-224, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 224: Application protocol: Mechanical product definition for process planning using machining features*", ISO/IEC, Geneva, Switzerland, 2001.

[ISO 10303-232:2001] ISO 10303-232, "*Industrial automation systems and integration -- Product data representation and exchange -- Part 232: Application protocol: Technical data packaging core information and exchange*", ISO/IEC, Geneva, Switzerland, 2001.

[ISO 13584-20:1998] ISO 13584-20, "*Systèmes d'automatisation industrielle et intégration -- Bibliothèque de composants -- Partie 20: Ressource logiques: Modèle logique d'expressions*", ISO 13584-20, edited by ISO, Geneva, 1998.

[ISO 13584-24:2003] ISO/IS 13584-24 G. PIERRA, Y. AIT-AMEUR, E. SARDET, "*Industrial Automation Systems and Integration, Parts Library, Logical Model of Supplier Library* ", ISO 13584-24, edited by ISO, Geneva, ISO (642 p.), 2003.

[ISO 13584-25:2004] ISO/IS 13584-25 Y. Ait-Ameur, G. Pierra, "*Industrial Automation Systems and Integration, Parts Library, Logical model of supplier library with aggregate values and explicit content*", ISO 13584-25, ISO (107 p), Geneva, 2004.

[ISO 13584-26:2000] ISO 13584-26, "*Industrial automation systems and integration -- Parts library -- Part 26: Logical resource: Information supplier identification*", ISO 13584-25, ISO, Geneva, 2000.

[ISO 13584-42:1998] ISO 13584-42, G. Pierra, "*Insdustrial Automation Systems and Integration- Part Library-Methodology for Structuring -Parts Famillies*", ISO, Geneva, 1998.

[ISO 9075:1992] ISO 9075, "*Information technology -- Database Language -- SQL*", ISO/IEC JTC1/SC21 Doc. 9075 N5739, Geneva, Switzerland, 1992.

[ISO 9075-2:1999] ISO 9075-2, "*Information technology -- Database languages -- SQL --Part 2: Foundation (SQL/Foundation)*", ISO/IEC 9075:1999(E), Geneva, Switzerland, 1999.

[ISO 9075-4:1999] ISO 9075-4, "*Information technology -- Database languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM)*", ISO/IEC 9075:1999(E), Geneva, Switzerland, 1999.

[Jacobson et al. 99] I. Jacobson, G. Booch and J. Rumbaugh, "*The Unified Software Development Process*", Addison-Wesley Eds, 1999.

[Kemmerer 99] S. Kemmerer, "*STEP: The Grand Experience*", (Editor of) NIST Special Publication 939, National Institute of Standards and Technology, Gaithersburg, MD, 1999.

[Kenneth 99] A. Kenneth, "*Oracle Designer Generation*", McGraw-Hill, ISBN: 0078824753, 1999.

[Kindrick et al 00] Jim Kindrick, Markus Hauser, Rogerio Barra, "*Usage Guide for the STEP PDM Schema*", PDM Implementor Forum, http://www.pdm-if.org/, 2000.

[Lämmer 00] Lutz Lämmer, Bodo Machner, "*Standards as enabler for PDM in virtual enterprises*", proceeding of PDT Days, pp. 233-240, QMS edition, Netherlands, 2000.

[Lampierre et al 95] S Lampierre, F Coufin, J.-P. Frachet, "*Méta-modélisation pour l'Intégration par les Données des Activités d'Ingénierie des Systèmes Industriels*", Acte de conférences du Congrès International de Génie Industriel, Montréal, CDN, 1995.

[Lang 88] B. Lang, "*Parsing incomplete sentences*", In Proc. Of the 12th International Conference on Computational Linguistics, volume 1, pp 365-371, Budapest, 1988.

[Lin 96] J. Lin, M.S. Fox, T. Bilgic, "*A requirement Ontology for Engineering Design*", Concurrent Engineering: Research and Applications, Vol 4(3), pp279-291, September 1996.

[Loffredo 98] David Loffredo, "*Efficient Database Implementation of EXPRESS Information Models*", PhD Thesis, Rensselaer Polytechnic Institute Troy, New York, 1998.

[Maurino 93] M. Maurino, "*La gestion des données techniques - – Technologie du concurrent engineering* ", Edition Masson, 1993.

[Meier 02] R. Meier and V. Cahill, "*Taxonomy of Distributed Event-Based Programming Systems*", in Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02). pp. 585-588, Vienna, Austria, 2002.

[Minsky 75] M. L. Minsky, "*A framework for representing knowledge*", In Winston ed. The Psychology of computer vision, McGraw Hill, New-York, 1975.

[Minsky 86] M. Minsky, "*The Society of Mind*", Simon & Schutster Inc. ed. New York, 1986.

[Mizoguchi, 02] Y. Mizoguchi, H. Murayama, N. Minamino, "*Class Query Language and its application to ISO13584 Parts Library Standard*", In Proceeding of the ECEC'2002 9th European Concurrent Engineering Conference, Concurrent Engineering: System integration for profit, pp. 128-135, Modena, Italy, 2002.

[Momjian 01] B. Momjian, "*PostgreSQL : introduction and concepts*", Addison-Wesly, 2001.

[Moranne 86] J.M. Moranne, "*Conception assistée par ordinateur d'ensembles mécaniques avec recherche d'une bonne solution : le logiciel : SICAM*", Proceedings of MICAD'86, Hermès, pp. 41-71, Paris, 1986.

[Morris 90] K.C. Morris, "*Translating Express to SQL: A User's Guide*", NISTIR 4341, National Institute of Standards and Technology, Gaithersburg, MD, http://www.mel.nist.gov/msidlibrary/summary/pubs90.htm , 1990.

[Murayama 00] H. Murayama, "*InterLIB : an integrated database framework for effective collaboration of Parts Library and EPISTLE class Library*", proceeding of PDT Days, pp. 141-146, QMS edition, Netherlands, 2000.

[Murayama 03] H. Murayama , O. Lemarchand, Y. Mizoguchi, "*Modeling Product Ontology with CQL*", Proceeding of the 10th ISPE International Conference On Concurrent Engineering: Research And Applications, A.A. Balkema Publishers, pp. 279-286, Madeira, Portugal, 2003.

[NaiRuo 00] L. NaiRuo, Y. XiaoHu. D. JinXiang, L. ShanPing, "*Research of a STEP-based tool for Product Data*", proceeding of PDT Days, pp. 363-384, QMS edition, Netherlands, 2000.

[NF X 60-012] Norme AFNOR, "*Termes et définitions des éléments constitutifs et de leurs approvisionnements pour les biens durables*" Norme expérimentale, 1982.

[Nijssen 81] G.M. Nijssen, "*An architecture for knowledge base systems*", Proc. SPOT-2 conf., Stockholm, 1981.

[Noonan 03] J. G. Noonan, "*IBM Rational Rapid Developer: Automated Construction*", White paper, diponible sur : http://www-306.ibm.com/software/rational/info/literature/design.jsp, 2003.

[OMG 00] Object Management Group, "*Product Data Management Enablers Specification*", OMG document, http://www.omg.org/technology/documents/formal/ product _data_management_enablers.htm, 2000.

[OMG 02] Object Management Group, "*Meta Object Facility (MOF) Specification*", Version 1.4, OMG document, disponible sur : http://www.omg.org/technology/documents/formal/mof.htm, Avril 2002.

[OMG 99] OMG Manufacturing Domain Task Force, "*STEP and OMG Product Data Management Specifications : A Guide for Decision Makers*, OMG Document (whitepaper), disponible sur :

http://www.omg.org/homepages/mfg/mfgppepdm.htm, octobre 1999.

[Paasilia et al. 93] P. Paasilia, A. Aatonen, A. Riitahuta, "*Automatic component selection*", in Kooij, C., MacConaill, P.A. and Bastos J. (eds), IOS Press, 1993.

[PDMIC 02] Product Data Management Information Center, "*Glossary of Product Data Mangement Related Terms*", (http://www.pdmic.com/glossary/index.html) - mise à jour du 12/07/2002.

[Peltier 03] Mikaël Peltier, "*Techniques de transformation de modèles basées sur la méta-modélisation*", Thèse de Doctorat, Université de Nantes, 2003.

[Pierra 00] G. Pierra, "*Représentation et échange de données techniques*", Mec. Ind., vol. 1, Elsevier SAS, pp. 397-414, 2000.

[Pierra 03] G. Pierra, "*Context-explication in conceptual ontologies: The PLIB approach*", Proceeding of the 10th ISPE International Conference On Concurrent Engineering: Research And Applications, A.A. Balkema Publishers, pp. 243-254, Madeira, Portugal, 2003.

[Pierra 90] G. Pierra, "*An object oriented approach to ensure portability of CAD standard part library*", presented at Eurographics'90, Montreux, 1990.

[Pierra 93] G. Pierra, "*A Multiple Perspective Object Oriented Model for engineering Design*", in: New Advance in Computer Aided Design & Computer Graphics, X. Zhang, Ed. International Academic Publishers, pp. 368-373, Beijing, China, 1993.

[Pierra 94] G. Pierra, "*Modelling classes of prexisting components in a CIM perspective: The ISO 13584/ENV 400014 approach*", revue internationale de CFAO et d'Infographie, vol. 9, pp. 435-454, 1994.

[Pierra 95] G. Pierra, "*Classes d'objets techniques et données de produit dans les systèmes d'IAO*", presented at Actes Quinzièmes Journées Internationales IA 95, Montpelier, 1995.

[Pierra et al 04] G. Pierra, H. Dehainsala, Y. Ait-Ameur, L. Bellatreche, J. Chohon, M. El-Hadj Mimoune, "*Base de Données à Base Ontologique : le modèle OntoDB*", à aparatre dans le proceeding de la conférence Bases de Données Avancées (BDA'2004), 21 pp, Farance, 2004.

[Pierra et al 96] G. Pierra, Y. Ait-Ameur, F. Besnard, P. Girard, J.-C. Potier, "*A general framework for parametric product model within STEP and Parts Library*", presented at European PDT Days, London, 1996.

[Plantec 99] A. Plantec, "*Utilisation de la norme STEP pour la spécification et la mise en œuvre de générateurs de code*", thèse, Université de Rennes 1, 1999.

[PostgreSQL 03] The PostgreSQL Global Development Group, "*PostgreSQL 7.4.3 Documentation*", disponioble sur: http://www.postgresql.org/docs/7.4/static/index.html, 2003.

[Randoing 95] J. M. Randoing, "*Les SGDT*", Edition Hermès, 1995.

[Rozenknop 02] A. Rozenknop, "*Une grammaire hors-contexte valuée pour l'analyse syntaxique*", Proceeding de la 9ème Conférence Annuelle sur le Traitement Automatique des Langues Naturelles, V(1), Association pour le Traitement Automatique des Langues (ATALA), pp. 95-104, Nancy, 2002.

[Rumbaugh et al 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorrensen, "*Object-Oriented Modelling and Design*", Prentice Hall, International Edition, 1991.

[Sardet 01] E. Sardet, G. Pierra, H. Murayama, Y. Oodake, Y. Ait-Ameur, "*Simplified Representation of Parts Library : Model, Practice and Implementation*", proceeding of PDT Days, Brussels, QMS edition, 2001.

[Sardet 99] E. Sardet, "*Intégration des approches modélisation conceptuelle et structuration documentaire pour la saisie, la représentation, l'échange et l'exploitation d'informations. Application aux catalogues de composants industriels*", thèse de doctorat, Université de Poitiers, 1999.

[Schenk et al 94] D. Schenk, P. Wilson, "*Information Modelling the Express Way*", Oxford University Press, 1994.

[Schreiber et al 92] T. Schreiber, B. Wielinga, J. Breuker, "*KADS: A Principled Approach to Knowledge-Based System Development*", Academic Press, London, Forthcoming, 1992.

[SET 89] SET : Z 68-300, "*Industrial Automation – External Representation of Product Definition Data – Data Exchange and Transfert Standard Specification*", AFNOR- Association Française de NORmalisation, France, 1989.

[Shen 95a] W. Shen, J.P. Barthes, "*Description and Applications of an Object-Oriented Model PDM, Modeling Complex Data for Creating Information: Real and Virtual Objects*", eds., Dubois J.E. & Gershon N., Spinger Verlag, pp 15-24, 1995.

[Shen 95b] W. Shen, J.P. Barthes, "*An object-oriented approach for engineering design product modelling*", in Knowledge Intensive CAD-1. Eds., T. Tomiyama, M. Mäntylä, S. Finger, Otaniemi 1995/TKO-C75, pp. 245-261, 1995.

[Staub 94] Günter Staub, *"PISA Information Modelling Language: EXPRESS-C"*, Technical Report, RPK Karlsruhe, Karl-sruhe, 1994.

[Staub et al 96] Günter Staub, Markus Maier, "*ECCO Tool Kit: User Reference Manual*", May 1996.

[Sybase 04] Sybase, "*PowerAMC™ Référentiel Mise en route*", Version 10.0, DC31031-01-1000-01, disponible sur http://download.sybase.com/pdfdocs/amc1000f/rlmr.pdf, 2004

[Tollenaere 99] M. Tollenaere, "*Internet, Intranet, SGDT : couvertures fonctionnelles et complémentarités*", Journée PRIMECA, IFMA, Clermont-Ferrand, 10 juin 1999.

[VDA 86] VDA-FS, "*Vereinung Deutsche Autombilindustie Flächen Schnittstelle : DIN 66301*", DIN-Deutsches Instiut für Normung, Germany, 1986.

[Wilkes 00] W. Wilkes, J. Bröking, "*MERCI : Integration of EXPRESS based and XML based component information representation*", proceeding of PDT Days, pp. 127-134, QMS edition, Netherlands, 2000.

[Yan et al 01] Ling Ling Yan, Renée J. Mille, Laura M. Haas, Ronald Fagin, "*Data-driven understanding and refinement of schema mappings*"**,** Proceedings of the 2001 ACM SIGMOD international conference on Management of data, pp. 485 – 496, ACM Press, California, United States, 2001.

Liste des figures

Liste des tableaux

Annexe : Le langage EXPRESS

1 Introduction

La modélisation de l'information a pour but d'identifier et de décrire les différents types d'information (i.e. des éléments de connaissance tel que le diamètre que possède une vis) concernant un aspect précis du monde réel qui sont nécessaires pour que l'état de celui-ci puisse être représenté et communiqué efficacement [Sardet 99]. La structuration de l'information permet son appréhension et son intelligibilité par un être humain ou par une machine. De nombreux formalismes se sont développés pour permettre cette structuration. De l'approche ensembliste à l'approche objet, et de la représentation graphique à la représentation textuelle, leurs pouvoirs d'expression ont progressé pour approcher le raisonnement humain dans la manière d'identifier et de distinguer les éléments traités et les associations qui les lient.

Une étude bibliographique nous fournit déjà un large éventail de formalismes : entité/association [Chen 76], OMT [Rumbaugh et al 91], NIAM [Nijssen 81], [Habrias 88]. La référence en terme de modélisation dans le domaine technique est le langage EXPRESS [Schenk et al 94] [ISO 10303-11:1994]. Il s'inscrit dans le prolongement des travaux sur les formulaires cités ci-dessus. EXPRESS a été développé dans les années 80 dans le cadre du projet STEP (*STandard for the Exchange of Product model data* ISO 10303). Il n'est pas seulement un langage de modélisation conceptuelle utilisable pour les échanges entre humains. Il est également un langage de définition de données (LDD) [Herbst 94] permettant de spécifier les données devant être générées et valider pour les machines. Son objectif initial est la normalisation des modèles de données pour pouvoir les utiliser sur des plates-formes hétérogènes et surtout pouvoir les échanger et les réinterpréter sans difficulté. Il a été décrit initialement pour définir des modèles de données dans les domaines techniques. Il est à présent largement utilisé pour la modélisation des données dans différents domaines [Ait-Ameur et al 00], [Plantec 99]. Il a été défini pour rendre plus précis et traitable par machine de tels modèles, et pour représenter beaucoup plus librement les contraintes sur les données et donc les conditions de leur intégrité. A la différence des langages de programmation orientée objet, EXPRESS n'est pas destiné à modéliser des systèmes informatiques (les objets ne possèdent pas de méthodes et la connaissance procédurale s'exprime exclusivement par des contraintes d'intégrité ou des fonctions de dérivation).

Contrairement aux autres formalismes de modélisation, EXPRESS est avant tout un formalisme textuel associé à une version graphique nommée EXPRESS-G. Le format

textuel en fait toute sa puissance d'expression par rapport aux autres formalismes : il est compilable et génère des modèles de données pour les langages C++, JAVA, ...

2 Les concepts

EXPRESS est un langage de modélisation orienté objet possédant les grandes caractéristiques des langages à objet [Booch 91] : abstraction, héritage et polymorphisme. Un modèle de données EXPRESS [ISO 10303-11:1994], [Bouazza 95a] est constitué d'un ensemble de modules appelés SCHEMAs. Chaque schéma décrit un ensemble d'entités qui représentent les objets de l'univers du discours. Un schéma contient deux parties. La première est constituée d'un ensemble d'entités structurées selon une approche orientée objet avec héritage multiple. La seconde est constituée d'un ensemble de fonctions, procédures et règles globales permettant de décrire la partie procédurale du modèle de données.

2.1 Les entités

Une entité désigne un élément identifiable dans le domaine étudié et possédant une existence effective et autonome. Elle est une représentation d'une catégorie d'objets. Elle correspond à une classe dans les langages à objet.

Le langage EXPRESS étant un formalisme de modélisation orienté objet, il permet donc de modéliser la connaissance structurelle sous une forme de hiérarchies d'entités associées à un mécanisme de généralisation/spécialisation. Le langage EXPRESS autorise différents types d'héritage, à savoir l'héritage simple, l'héritage multiple ou encore l'héritage répété. La mise en œuvre de ce mécanisme d'héritage a pour conséquence que toute définition d'attributs et/ou de contraintes d'intégrité définis dans une super-classe sera héritée dans la sous-classe. On peut enfin noter que la déclaration d'une telle relation entre deux entités doit obligatoirement être spécifiée dans la sous-classe.

Le principe de catégorisation de l'information permet de factoriser les informations relatives à des classes dans des super-classes (héritage). Les super-classes peuvent cependant ne jouer qu'un rôle fédérateur : leur instanciation n'est effective qu'au travers de l'instanciation de l'une (ou plusieurs) de ses sous-classes. Ce concept d'entité non instanciable, ou entité abstraite, peut être décrit en EXPRESS. L'exemple du tableau suivant (Tableau 12) présente une arborescence d'entités : deux sous-classes, *étudiant* et *enseignant,* héritent des caractéristiques d'une super-classe non instanciable, *personne*.

```
ENTITY Personne
ABSTRACT SUPERTYPE OF (ONEOF(Etudiant,
Enseignant));
END_ENTITY;

ENTITY Etudiant
SUBTYPE OF (Personne);
END_ENTITY;

ENTITY Enseignant
SUBTYPE OF (Personne);
END_ENTITY;
```

Tableau 12 : Notion d'héritage en EXPRESS

Une classe est donc exprimée par le mot-clé **ENTITY**. La classe *personne* est définie comme étant une super-classe abstraite. On peut noter que la super-classe spécifie son type d'instanciation. En effet, le mot-clé **ONEOF** signifie que la classe *personne* ne pourra être instanciée qu'au travers de la classe *étudiant* ou bien de la classe *enseignant*. EXPRESS permet d'étendre ce type de règle. Une super-classe peut être instanciée par l'ensemble de ses sous-classes (mot-clé **AND**), ou bien par un sous-ensemble de ses sous-classes (mot-clé **ANDOR**).

2.2 Les attributs

La description de catégories d'objets (entités ou classes) peut être enrichie par l'adjonction de caractéristiques à l'aide des attributs. Ils modélisent la connaissance descriptive. Chaque attribut est défini par un label et possède un domaine (où il prend ses valeurs) défini par un type de données. Ce dernier peut être simple (tel qu'un entier, une chaîne de caractères, …) ou bien un agrégat (telles que les listes (**LIST**), les ensembles (**SET**), …) ou encore défini par l'utilisateur en utilisant le mot clé **TYPE** ou bien finalement une entité du modèle de données.

Dans le langage EXPRESS, on distingue trois catégories d'attributs :

- **les attributs libres ou explicites :** sont utilisés pour représenter les propriétés qualifiées de fondamentales, i.e., les propriétés intelligibles et stables d'une catégorie d'objets d'un domaine particulier. Ils sont indépendants de tout autre attribut ; de plus, ils peuvent être associés à une valeur optionnelle ;

- **les attributs dérivés ou calculés :** dépendent fonctionnellement d'autres attributs. Ils sont calculés à partir des attributs explicites et/ou d'autres attributs dérivés ;

- **les attributs inverses :** permettent la représentation de la cardinalité de la relation réciproque ou inverse d'une relation donnée.

Le Tableau 13 enrichit l'exemple du Tableau 12 en utilisant des types simples définis dans la section suivante (2.3).

```
ENTITY Personne
ABSTRACT SUPERTYPE OF (ONEOF (Etudiant,
Enseignant));
  son_nom : STRING;
  son_prenom : STRING;
DERIVE
  Initial : STRING : = calcul_initial
(SELF.son_nom) ;
END_ENTITY;

ENTITY Etudiant
SUBTYPE OF (Personne);
  ses_notes :SET OF [1 :10] OF REAL;
END_ENTITY;

ENTITY Enseignant
SUBTYPE OF (Personne);
  son_salaire : OPTIONAL REAL;
END_ENTITY;
```

Tableau 13 : Définition d'attributs en EXPRESS

La classe *personne* est caractérisée par deux attributs libres, *son_nom et son_prenom* de type chaîne de caractères *(string)*, et un attribut dérivé, *initiale*, de type chaîne de caractères, dont la valeur est calculée par une fonction, *calcul_initial* (qui n'est pas définie ici), prenant en entrée la valeur de l'attribut *son_nom*. On peut noter que le paramètre de la fonction (**SELF**.*son_nom*) utilise **SELF** pour signifier que *son_nom* est un attribut de la classe courante. La notation pointée permet d'accéder aux attributs des classes. Enfin, on a affecté d'une part un attribut nommé *ses_notes* à la classe *étudiant* (attribut de type ensemble d'au plus dix réels), et, d'autre part, caractérisé la classe *enseignant* par un attribut *son_salaire*, de type réel, dont la valeur est optionnelle.

2.3 Les types

Le langage EXPRESS permet de distinguer les instances d'entités des valeurs. Une valeur ne possède pas d'identification et est représentée de façon explicite. Les types de valeurs autorisés, dits types simples ou atomiques, sont les types chaînes de caractères (**STRING**), numériques (**REAL, BINARY, INTEGER**) ou logiques, (**LOGICAL** et **BOOLEAN**).

Les agrégats sont des sacs (**BAG**), des ensembles (**SET**), des listes (**LIST**) ou des tableaux (**ARRAY**). Les trois premiers agrégats peuvent êtres bornés explicitement (définition du nombre minimal et maximal de valeurs). Pour le type tableau, on doit définir sa plage d'indexation. Le tableau suivant illustre les types cités ci-dessus.

TYPES SIMPLES	
BINARY	**0, 1**
BOOLEAN	**TRUE** ou **FALSE**
LOGICAL	**TRUE, FALSE** ou **UNKNOWN**
NUMBER	Union des types **INTEGER** et **REAL**
REAL	Nombre réel

INTEGER	Nombre entier
STRING	Chaîne de caractères
TYPES COLLECTIONS	
LIST	Collection ordonnée d'éléments avec répétition
SET	Collection non ordonnée d'éléments sans répétition
BAG	Collection non ordonnée d'éléments avec répétition
ARRAY	Tableau de dimension finie

Tableau 14 : Les types simples et les agrégats dans EXPRESS

De plus, il est possible de construire des types utilisateurs (**TYPE**) permettant de donner un nom différent à un type existant avec la possibilité de le contraindre, rendant ainsi le modèle de données plus intelligible. Un type ainsi nommé peut être un type entité. Le langage EXPRESS définit également deux constructeurs de types : le type énuméré et le type sélection. Le premier permet de définir un type de données qui décrit un ensemble de valeurs qui sont des identificateurs, le second spécifie une union de types.

```
TYPE t_jour_semaine= ENUMERATION
OF (lundi, mardi, mercredi,
jeudi, vendredi);
END_TYPE;
TYPE t_jour_weekend = ENUMERATION
OF(samedi, dimanche);
END_TYPE;
TYPE t_jour = SELECT
(jour_semaine, jour_weekend);
END_TYPE;
TYPE t_salaire = REAL;
WHERE
SELF > 0;
END_TYPE;
```

Tableau 15 : Quelques déclarations de type en EXPRESS

Un type de données *t_jour* est défini comme étant l'union des types énumérés *t_jour_semaine* et *t_jour_weekend*. Enfin, un type appelé *t_salaire* est décrit comme étant un réel pour lequel une contrainte spécifiant que le type réel est limité aux réels strictement positifs a été définie.

2.4 Les contraintes

Le langage EXPRESS offre un langage de description de contraintes très élaboré. Dans les sections suivantes, nous présentons les mécanismes d'expression de ces contraintes. En particulier, nous différencierons les contraintes s'appliquant aux instances de classes des contraintes s'appliquant aux classes. Pour des explications complémentaires, il est souhaitable de consulter [ISO 10303-11:1994], [Schenk et al 94], [Bouazza 95a].

2.4.1 Contraintes sur les instances

Les contraintes sur les instances stipulent des règles auxquelles chacune des instances de la classe où elles sont décrites devront se plier.

Contraintes d'unicité

EXPRESS permet de définir des instances de classes comme devant être caractérisées par des attributs possédant des valeurs uniques dans un contexte d'instanciation donné. Cela peut être comparé à la clé primaire dans le cadre des modèles relationnels. Cette contrainte est introduite par le mot-clé **UNIQUE**, et peut s'appliquer à un ou plusieurs attributs de la classe.

De tels ensembles d'attributs peuvent, si besoin, être utilisés comme clé d'identification des différentes instances. Ceci n'empêche pas chaque instance d'être supposée identifiée par un identificateur unique attribué par le système (identificateur apparaissant d'ailleurs explicitement dans les formats d'échanges de populations, voir section 3).

Contraintes de cardinalité

L'expression de relations dans le langage EXPRESS se fait au moyen d'attributs, ces mêmes attributs ayant pour type une classe. La relation de cardinalité directe se fait dans l'expression de cet attribut, en le définissant soit obligatoire, soit optionnel, ou encore de type collection. De même, la classe référencée peut définir une relation de cardinalité inverse sur un attribut qui la référence. Cette cardinalité inverse est introduite par la description d'un attribut dans une clause **INVERSE**. Tout comme les attributs traditionnels, cet attribut sera caractérisé par un type (la classe sur laquelle la relation de cardinalité s'applique), et par une référence sur l'attribut qui réalisait la relation directe. Le nombre des instances référençant une autre instance peut être restreint par l'attribut inverse. Ainsi, on peut préciser, par exemple, qu'une voiture n'appartient qu'à un seul propriétaire.

Contraintes d'instances

Le langage EXPRESS permet de définir des règles de cohérence des objets issus d'une opération d'instanciation de classe. Ces règles sont des expressions booléennes auxquelles toutes les instances de la classe devront obéir. Ces règles (**WR** dans le Tableau 16) sont introduites via une clause **WHERE**, et s'appliquent à un ou plusieurs attributs. On peut, par exemple, assurer via de telles règles que dans certains cas, une valeur doit être fournie pour un attribut déclaré optionnel. Notons enfin que de telles clauses **WHERE** peuvent être utilisées pour contraindre des types de données utilisateurs (**TYPE**).

2.4.2 Contraintes sur les classes

EXPRESS permet de définir des contraintes sur tous les objets issus d'instanciations successives d'une classe particulière. Par exemple, une telle contrainte pourrait exprimer une cardinalité maximale sur le nombre d'instances créées à partir d'une classe. Ces contraintes, dites contraintes globales, sont définies dans une clause

appelée **RULE**. Ces règles globales ne sont pas déclarées au sein des entités mais sont définies séparément.

2.4.3 Contraintes ensemblistes

Enfin, le langage EXPRESS permet de décrire des contraintes de type ensembliste. Les éléments d'une collection peuvent être parcourus un à un, et peuvent être vérifiés comme répondant à des critères particuliers.

Les contraintes ensemblistes permettent de représenter à la fois l'utilisation du quantificateur universel '∀' :

$\forall x \in X \mid p(x)$

- où x : ensemble d'instances de type X ;
- et p(x) : une proposition logique se rapportant à x.

Et celle de quantificateur existentiel '∃' :

$$\exists x \in X \mid p(x) = \neg(\forall x \in X \mid \neg p(x))$$

Ces contraintes sont utilisées soit dans des clauses **WHERE**, soit dans des clauses **RULE**, et sont introduites par le mot-clé **QUERY**. Enfin, soulignons que toutes ces contraintes sont évaluables, car l'ensemble des instances d'une classe est un ensemble fini.

2.5 Exemple de modèle de données contraint

Nous allons enrichir le modèle de données que nous avons entrepris de décrire en lui ajoutant de nouveaux types, de nouveaux attributs et un certain nombre de contraintes d'intégrité (Tableau 16).

```
SCHEMA Etablissement;
TYPE t_salaire = REAL;
WHERE
  SELF > 0;
END_TYPE;

TYPE t_num_ss = STRING(13) FIXED;
WHERE
  WR: t_num_ss[1] = '1'
  OR t_num_ss[1] = '2';
END_TYPE;

TYPE t_cours = ENUMERATION OF(
  math, info, histoire, sport);
END_TYPE;

TYPE t_nom = STRING;
END_TYPE;

ENTITY personne
ABSTRACT SUPERTYPE OF
  ONEOF (etudiant, enseignant);
  num_ss: t_num_ss;
  son_nom : t_nom;
DERIVE
  initiale: STRING :=
calcul_initial(SELF.so_nom);
UNIQUE
  UR: num_ss;
END_ENTITY;
```

```
TYPE t_annee = ENUMERATION OF(
  A1, A2, A3);
END_TYPE;

ENTITY etudiant
SUBTYPE OF personne;
  Ses_notes: SET[0:10] OF REAL;
  suit: SET[1:4] OF COURS;
WHERE
  WR: SIZEOF(QUERY(X <*
SELF.ses_notes |(X < 0) OR (X > 20)))
= 0;
END_ENTITY;

ENTITY enseignant
SUBTYPE OF personne;
  Son_salaire: OPTIONAL t_salaire;
  enseigne: SET[1:?] OF cours;
END_ENTITY;

ENTITY cours
  intitule: t_cours;
  annee: t_annee;
INVERSE
  est_suivi_par: SET[0:30] OF
      etudiant FOR suit;
  est_enseigne_par: enseignant
      FOR enseigne;
END_ENTITY;
END_SCHEMA ;
```

Tableau 16 : Un modèle de données contraint

Nous avons créé quatre nouveaux types de données :

- *t_nom*, un type défini basé sur le type **STRING** ;
- *t_num_ss*, une chaîne de caractères possédant exactement 13 caractères, pour laquelle le premier caractère est soit '1', soit '2' ;
- *t_cours*, un type énuméré spécifiant les cours disponibles ;
- *t_année*, un type énuméré définissant les trois années pour lesquelles les cours sont dispensés.

De plus, une nouvelle classe, *cours*, a été créée, ainsi que deux propriétés, *enseigne* et *suit*, respectivement dans les classes *enseignant* et *étudiant*, ceci afin de mettre en évidence l'expression de relations de cardinalités bilatérales. Ainsi, nous avons exprimé une relation entre *étudiant* et *cours*, pour laquelle un *étudiant* doit suivre au moins un et au plus quatre *cours*, et qu'un *cours* peut être suivi par au plus trente *étudiant*s. De même, un *cours* est dispensé par un (et un) seul *enseignant*, et un *enseignant* peut intervenir dans plus d'un *cours*.

Nous avons aussi apposé à la classe *personne* une contrainte d'unicité sur un nouvel attribut, *num_ss*. Enfin, la classe *étudiant* a été enrichie d'une contrainte spécifiant que

tous les éléments de l'ensemble appelé *ses_notes* doivent être des valeurs réelles comprises entre 0 et 20 inclus.

2.6 Les procédures et les fonctions

Le langage EXPRESS enrichit son pouvoir d'expression en proposant aux utilisateurs un ensemble de fonctions prédéfinies (par exemple **QUERY** (requête sur les instances), **SIZEOF** (taille d'une collection), …), la possibilité de définir leurs propres procédures et fonctions en vue du calcul des attributs dérivés, ou encore le moyen d'exprimer des contraintes d'intégrité.

La définition des fonctions et procédures repose sur un langage impératif structuré, proche de Pascal, incluant déclarations de types et structures de contrôle. Notons que EXPRESS autorise la récursivité, d'où une très grande souplesse d'utilisation.

L'exemple suivant (Tableau 17) exprime la fonction *calcul_initial(...)* évoquée dans Tableau 13.

```
FUNCTION     calcul_initial(nom:
t_nom): STRING;
      RETURN(nom[1]);
END_FUNCTION;
```

Tableau 17 : Un exemple de déclaration de fonction

2.7 Modularité

Toute description de modèles de données se fait en EXPRESS dans le cadre d'un **SCHEMA** qui représente l'univers du discours auquel le modèle de données tente de donner une représentation abstraite.

De plus, tout schéma peut importer des classes décrites dans le cadre d'autres schémas via des clauses d'importation de différents types :

- **USE** : les entités importées ont le même statut que les entités définies localement, ce qui implique qu'elles peuvent être instanciées isolément ; un tel mode de référence peut être comparé à un mécanisme de *réutilisation* de l'information ;

- **REFERENCE** : les entités importées ne peuvent être instanciées que dans le cadre d'une relation avec une entité du modèle courant ; un tel mode de référence peut être comparé à un mécanisme de *partage* de l'information.

Tout comme le concept de modularité [Booch 86], cette facilité permet de découper toute modélisation en un ensemble de modèles de données, cohérents entre eux, et visant chacun à résoudre un problème de modélisation particulier. Le Tableau 18

donne une vue globale de la définition d'un modèle de données et de l'importation de classes provenant d'autres schémas.

SCHEMA S1; ... **END_SCHEMA;** **SCHEMA S2;** ... **END_SCHEMA;**	**SCHEMA S;** **USE FROM S1;** **REFERENCE FROM S2;** ... **END_SCHEMA;**

Tableau 18 : Importations de schémas

3 Représentation des instances : le fichier physique

Une population de données conforme à un modèle de données EXPRESS est représentée au sein d'un fichier physique (Tableau 19). Celui-ci est un fichier texte répondant à une syntaxe stricte [ISO 10303-21:1994] précisant en particulier les règles de traduction de définitions EXPRESS pour décrire des instances. Un exemple commenté d'instances du modèle de données que nous avons décrit dans le Tableau 16 est présenté ci-après :

```
ISO-10303-21;


HEADER;
FILE_DESCRIPTION(('CECI EST UN TEST','LE FICHIER CONTIENT LA
DESCRIPTION DE ...'),'1')
FILE_NAME('NOM','2002-08T15:12:30','M.EL-HADJ MIMOUNE', 'LISI/ENSMA',
'PREPROC_VERSION','SYSTEME','AUTORIS');
FILE_SCHEMA('ETABLISSEMENT_SCHEMA');
END_SEC;
DATA;
#1 = ETUDIANT ('1700975121457', /* num_ss, hérité de la classe
personne*/
       'Dupont',           /* son_nom, hérité de la classe
personne */
       (10.0,13.5,15.0,9.0),/* ses_notes*/
       (#5,#6,#7)) /*les cours suivis par l'étudiant*/
#2 = ETUDIANT ('2700286054018', 'Durand', (12.0,18.5,15.0),
(#5,#6..));
#3 = ENSEIGNANT ('1541211100004', 'Martin', $, (#5, #6));
#4 = ENSEIGNANT ('1600366015259', 'Dupont', $, (#6)));
#5 = COURS(.MATH., A1);
#6 = COURS(.INFO., A2);
..
END_DATA;
END-ISO-10303-21;
```

Tableau 19 : Un exemple de fichier physique

Une instance fait référence à l'entité dont elle est issue par son nom et un identifiant (#i), suivi d'un nombre unique et contient la suite de valeurs de ses attributs entre parenthèses. Chaque attribut respecte son type défini dans le modèle. Les attributs dérivés, tout comme les attributs inverses, sont calculés par le système receveur et ne

sont pas présents dans le fichier. Un agrégat se présente sous la forme d'une liste d'attributs comprise entre parenthèses. Un attribut déclaré optionnel ne possédant aucune valeur est représenté par le symbole $.

4 L'express-G

L'EXPRESS-G est une représentation graphique de la représention textuelle du langage EXPRESS. Il permet une représentation synthétique d'un modèle de données EXPRESS. De plus, ce formalisme peut être utilisé dans les phases préliminaires de conceptions de modèles de données. EXPRESS-G permet une représentation des concepts structurels et descriptifs du modèle de données par une annotation graphique, ce qui augmente sa lisibilité et sa compréhensibilité. Par contre, les aspects procéduraux (dérivation et contraintes) ne peuvent être représentés. L'exemple suivant illustre une représentation EXPRESS-G du modèle de données du Tableau 16.

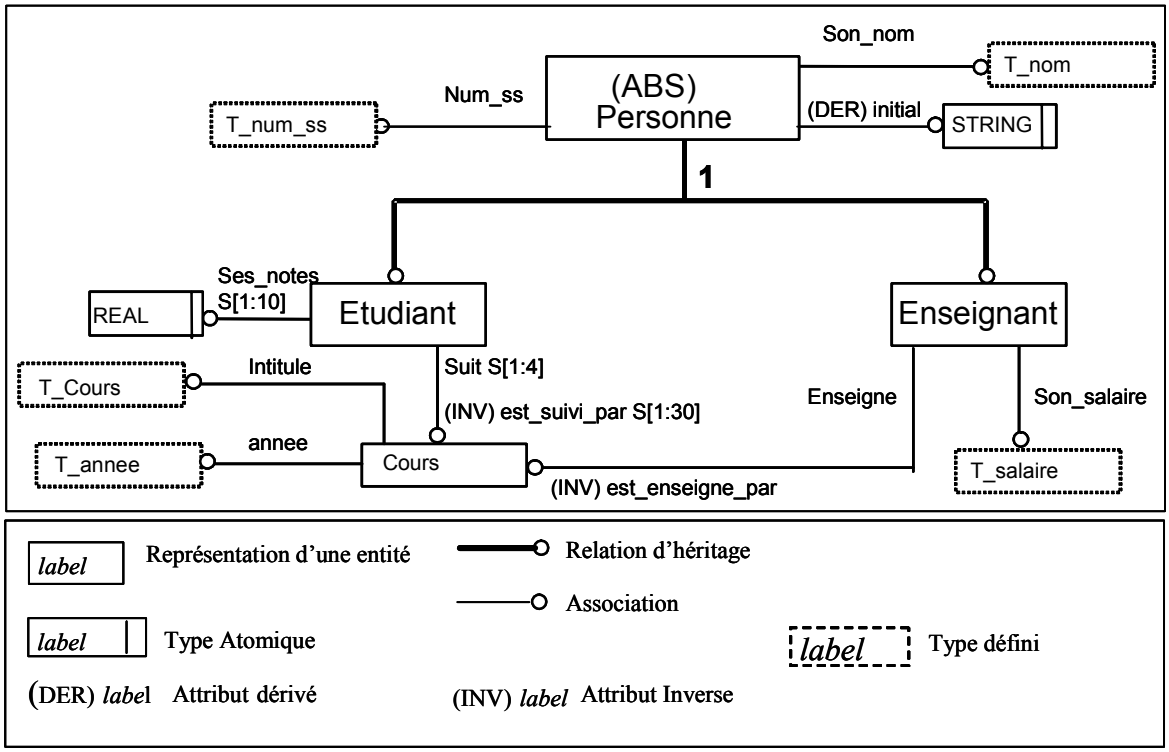


Figure 46 : représentation graphique EXPRESS-G du schéma etablissement

5 Conclusion

Avec le pouvoir d'expression et les nombreuses fonctions que propose le langage EXPRESS, la modélisation de la connaissance selon les trois points de vue (structurel, descriptif et procédural) est totalement possible. Ajouter des fonctions au moyen d'un langage procédural proche du PASCAL vient offrir aux possibilités déjà présentes un pouvoir d'expression important. De plus, les fonctions prédéfinies telle que **QUERY** permettent l'expression des contraintes complexes faisant ainsi d'EXPRESS un langage de modélisation puissant. C'est avant tout pour ce pouvoir d'expression que ce langage a été créé pour le projet STEP et la modélisation des produits mais est aussi utilisé à

travers d'autres projets similaires tels que PLIB pour la modélisation des composants. Par ailleurs, l'expression des contraintes fonctionnelles (via les attributs dérivés) permet de développer la programmation événementielle appliquée à EXPRESS. Ceci permet d'avoir la possibilité d'intégrer au sein du modèle même les méthodes de son implémentation dans différents modèles cibles.

Contribution à la modélisation explicite et à la représentation des données de composants industriels : application au modèle PLIB

Présentée par :

Mourad EL HADJ MIMOUNE

Directeurs de Thèse

Guy Pierra et Yamine Ait-Ameur

Résumé. De nouveaux modèles de données ont été développés ces dernières années pour représenter les produits complexes des entreprises industrielles. Le modèle PLIB (ISO-13584) représente implicitement les composants répétitifs par l'intermédiaire de contraintes décrivant les composants de chaque famille. La norme STEP (ISO-10303), et les Systèmes de Gestion des Données Techniques (SGDT), décrivent explicitement chaque produit de l'entreprise par l'ensemble de ces constituants. L'objet de cette thèse est d'étudier une architecture de base de données permettant de représenter de façon homogène ces deux types de modèles. L'approche proposée repose sur un modèle explicite des composants, sur la possibilité de générer cette représentation à partir d'une représentation implicite, et sur une architecture opérationnelle de base de données permettant de représenter à la fois les données et leur description conforme à une ontologie PLIB.

Mots clés : Base de données, Ontologie, Bibliothèque de composants, PLIB, SGDT, données de produits.
