Praxeme Institute

Composant PxM-41 « Modus : La méthodologie Praxeme »

La dérivation du modèle "métier" en modèle logique des données

Objectif

Le modèle "métier" incorpore ce que les méthodes antérieures nommaient "modèle conceptuel des données" (MCD). Les possibilités d'expression d'un modèle UML obligent à compléter les règles de passage du modèle objet vers le modèle logique des données (MLD).

Ce document rassemble ces règles et indique les décisions à prendre en matière d'architecture de données, dans l'aspect logique de la Topologie du Système Entreprise.

Contenu

- Orientations pour l'architecture des données
- Synthèse des décisions relatives aux données
- Le travail préalable sur le modèle sémantique
- La dérivation du modèle sémantique vers le MLD
- Les décisions sur le MLD
- Éléments pour la modélisation physique des données

Rédacteur Dominique VAUQUIER

Version 1.3, le 9 juillet 2009

✉ 21, chemin des Sapins – 93160 NOISY-LE-GRAND – France
dominique.vauquier@praxeme.org
☎+33 (0)6 77 62 31 75

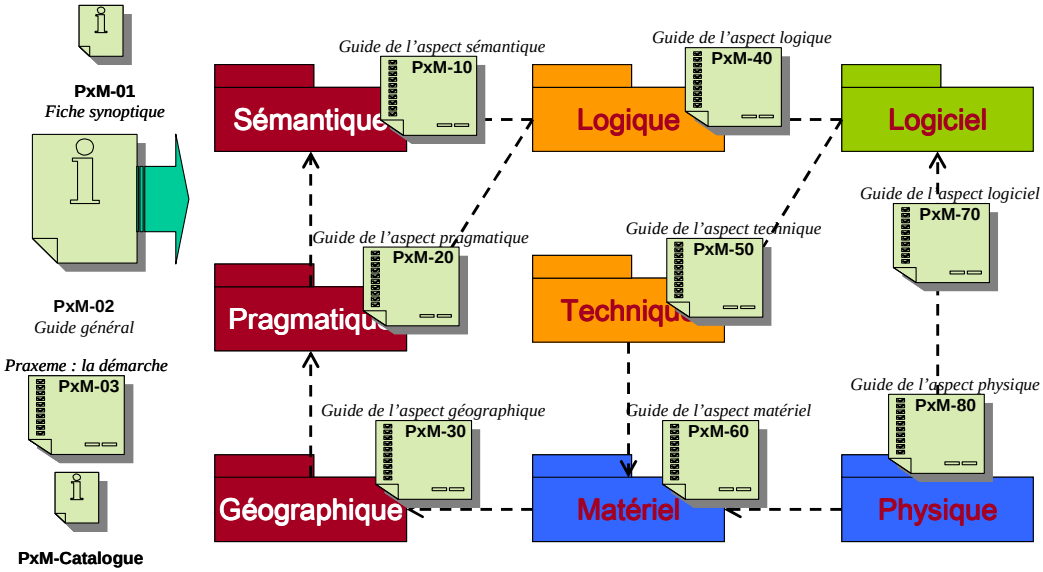
Éléments de configuration

Situation du composant

Positionnement dans la documentation

La méthodologie Praxeme est structurée selon les aspects de la Topologie du Système Entreprise. Le *Guide général* explique cette approche.

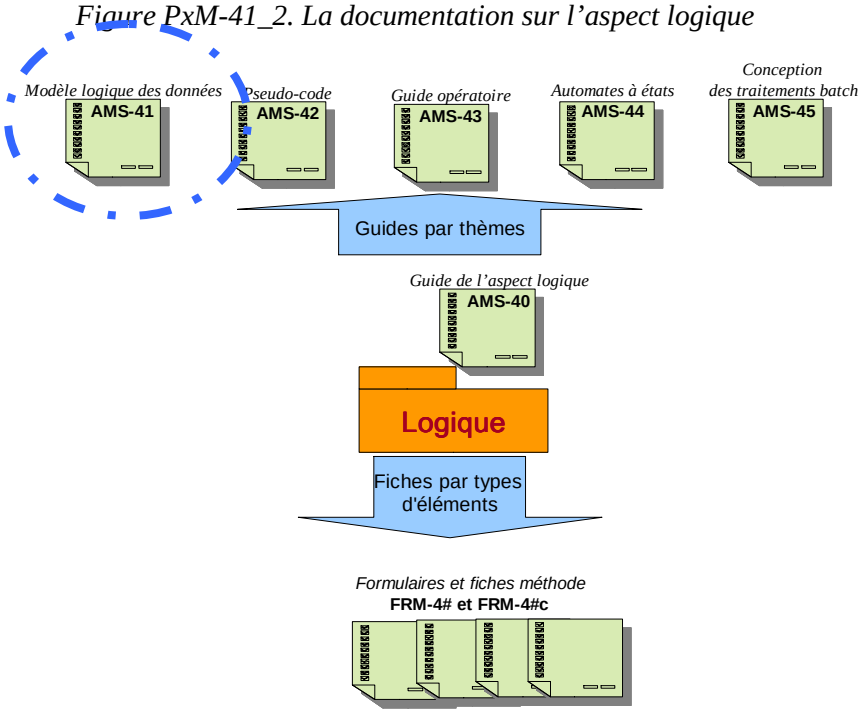
Figure PxM-41_1. Architecture du référentiel méthodologique



Documentation pour l'aspect logique

PxM-41 complète le « Guide de l'aspect logique » (PxM-40).

Figure PxM-41_2. La documentation sur l'aspect logique



Éléments de configuration (suite)

L'historique

Indice	Date	Rédacteur	Contenu
1.0	22/06/05	DVAU	Version élaborée pour le projet « Règlements » de la SMABTP (financement SMABTP, validation par Cellule Architecture Technique et Jean-Luc TREMOUILLE)
1.1	19/12/06	DVAU	Intégration dans Praxeme. Compléments.
1.2	13/04/07	DVAU, AJER	Compléments sur l'architecture des données et prise en compte des 7 principes de modélisation des données.
1.3	1/07/2009	DVAU	Relecture avant traduction et utilisation sur le programme Multi-Access du groupe AXA.
1.3			Version actuelle du document

Validation

Ont participé à la revue de ce document : Pierre BONNET (Orchestra Networks), David LAPETINA (SMABTP), Jean-Luc TREMOUILLE (SMABTP), Anthony JERVIS (Accenture).

Disponibilité

Ce document est disponible sur le site Praxeme et utilisable dans les conditions définies page suivante. Les éléments sources (documents et graphiques) peuvent être obtenus sur demande.

Propriétaire

Le référentiel Praxeme a été élaboré dans le cadre de l'initiative pour une méthode publique. Les contributeurs se réunissent au sein du « Collège des contributeurs » qui oriente les travaux en fonction des préoccupations des entreprises et organismes. L'association *Praxeme Institute* fait évoluer le fonds commun.

Toute suggestion ou souhait d'évolution sont les bienvenus (à adresser à l'auteur).

Licence

Conditions d'utilisation et de diffusion

Droits et devoirs

Ce document est protégé par une licence « Creative Commons » , résumée ci-dessous. Le terme « création » s'applique au document lui-même. L'auteur original est :

- Dominique VAUQUIER, pour le document ;
- l'association *Praxeme Institute*, pour l'ensemble de la méthodologie Praxeme.

Nous vous demandons de citer l'un et/ou l'autre, selon que vous extrayez une citation directe ou que vous vous référez aux principes généraux de la méthodologie Praxeme.

Sommaire

Table des figures

Exergue

« *La logique n'est pas une théorie, mais une image réfléchie du monde.* »

Ludwig Wittgenstein, *Tractatus logico-philosophicus*

Remerciements

Les recommandations rassemblées ici proviennent, pour l'essentiel, de deux sources :

- notre héritage merisien ;
- la méthode classes-relations.

Pour la première, reconnaissons notre dette à l'égard des créateurs de Merise, Hubert TARDIEU, Arnold ROCHFELD, René COLLETI, Yves TABOURIER. Pour la seconde, c'est Philippe DESFRAY qu'il faut remercier pour avoir perfuser un peu de rigueur dans la littérature « objet ».

Introduction

Dériver le MLD à partir des modèles amont

Objectif du document

Ce document capitalise sur la conception des bases de données, à partir d'un modèle objet. Il tente une synthèse des règles qui permettent de passer des modèles « métier » – orientés objets – au modèle logique des données – relationnel. Les modèles « métier » qui vont être dérivés sont :

- le modèle sémantique, bien sûr, essentiellement modèle des objets « métier » ;
- dans le modèle pragmatique, le modèle des objets de nature organisationnelle ou administrative.

Les règles de dérivation exposées ici partent d'un modèle de classes pour aboutir à un MLD (modèle logique de données), en style relationnel. Les autres types de SGBD ne sont pas envisagés ici.

Principe

La philosophie de la méthodologie Praxeme repose sur la séparation des niveaux de préoccupation (cf. *Guide général*, référence PxM-02).

Aspects amont

La conception du MLD prend, en entrée :

- le modèle sémantique, essentiellement sous la forme de modèle de classes ;
- le modèle pragmatique, s'il comprend un modèle de classes exprimant les notions de l'organisation (acteur, rôle, habilitation, structure, dossier et autres objets de nature organisationnelle).

La limitation au modèle de classes s'explique par le but de la conception : focalisée sur les données persistantes, celle-ci ne s'occupe que de la statique du modèle. Il lui faudra tout de même considérer la présence d'un automate à états et lui garantir la persistance.

Aspects aval

Le maître mot de la conception des données est la dérivation. Cette orientation se conforme au standard MDA[1]. Les modèles sémantique et pragmatique font abstraction des technologies. Avec le MLD, les choix techniques commencent à poindre : il s'agit, au moins, de prendre en compte le *style* du SGBD retenu. En l'occurrence, il s'agira d'un SGBD relationnel simple.

Les précisions concernant l'éditeur, la version et les possibilités du SGBD appartiennent à la technologie. Elles conditionneront la conception du modèle *physique* des données. Cet acte, motivé par l'optimisation de la base, est évoqué dans la dernière partie mais dépasse l'intention de ce document.

La nécessité d'un MLD

La nécessité d'un MLD fait, parfois, l'objet de discussions.

Pourquoi ne pas générer directement le modèle physique de données, dernier stade de la chaîne ? Parce que de nombreuses décisions sur le modèle des données peuvent être prises à un niveau logique, donc indépendamment du détail des choix techniques. Ce modèle sera donc plus largement utilisable.

Pourquoi, dans l'aspect logique, un modèle de données en plus du modèle des services et du modèle des flux ? Parce que ces modèles, quoique proches structurellement, répondent à des impératifs différents. Leurs règles de conception et d'optimisation peuvent différer. De plus, le MLD est un produit indispensable pour traiter des thèmes comme l'historisation ou l'intégrité des données, qui ont pu être délaissés par la modélisation sémantique.

Origine

Ce document résume le savoir-faire acquis sur plusieurs projets, principalement pour Celesio, SMABTP, EDF DOAAT (voir page des références sur www.praxeme.org).

[1] MDA : *Model Driven Architecture*, standard de l'OMG préconisant la séparation des modèles. Les modèles amont (PIM : *platform-independant models*) sont indépendants des choix techniques, donc assez stables et transportables dans plusieurs systèmes. Ils se dérivent, par application de choix techniques, en des PSM, *platform-specific models*.

Orientations pour l'architecture des données

Le service fournit le seul moyen d'accéder aux informations

Le style de SGBD

Par opportunité, ce document ne s'intéresse qu'au style relationnel, limité au SQL standard. Il évoque tout de même, à propos de l'héritage, les possibilités du modèle relationnel étendu, mais ne détaille pas les possibilités avancées des SGBDR du marché. Il faudrait, pour cela, faire le point sur les règles de dérivation qui exploitent ces possibilités et qui définiraient un autre style, disons le « relationnel étendu ».

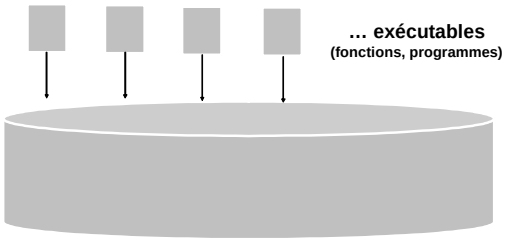
Les données et les services

L'architecture des données subit l'influence de l'approche choisie pour l'architecture du système.

Approche fonctionnelle

Dans l'approche fonctionnelle, la transformation du modèle conceptuel en MLD se fait en bloc, sans fracture, sans structure. Elle peut appliquer les règles de dérivation de façon quasi-automatique, sans autre considération de structure. Le SGBD assume un rôle prépondérant et le concepteur peut exploiter toutes les possibilités de la technologie relationnelle.

Figure PxM-41_3. Architecture des données dans l'approche fonctionnelle



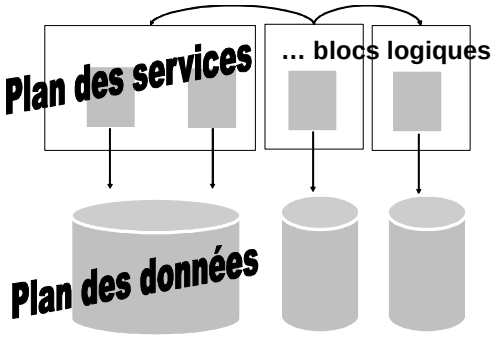


Figure PxM-41_4. Architecture des données dans l'approche services

Approche orientée services

Le SGBD ne saurait prendre en charge toutes les contraintes portant sur les données. Les services sont conçus pour fournir le seul accès valide aux informations et garantir toutes les contraintes.

L'architecture logique se dessine, donc, comme un plan des services qui surplombe et masque le plan des données.

Le paragraphe suivant détaille les conséquences de cette orientation.

Les conséquences

L'architecture de services cherche à définir des composants autonomes (« Blocs logiques », machines, ateliers, etc.). Leur autonomie couvre la gestion des données, rassemblées comme les traitements selon le critère de la sémantique. Le découpage du modèle sémantique en domaines d'objets et la structuration du modèle pragmatique vont façonner le modèle logique des données, comme ils le font du plan des services.

Par ailleurs, la conception des services s'impose des exigences :

- Chaque table est sous la responsabilité exclusive d'une machine logique[2].
- Les besoins d'information sont satisfaits par les services, uniquement.

Ces règles bannissent tout accès direct à la base de données, en dehors des machines logiques propriétaires des données.

Approche SOA radicale

Si on va jusqu'au bout de la logique SOA, on cherche à développer des constituants logiques autonomes. Dans les termes de Praxeme, l'atelier logique correspond à ce qui sera, dans l'aspect physique, l'unité de déploiement. Qu'un atelier se comporte comme un composant autonome, cela signifie que l'on peut prendre le composant logiciel qui en dérive et le déplacer sur tel ou tel nœud de l'architecture matérielle. Il faut, donc, que l'atelier embarque au moins ses ressources en données. La conséquence est celle-ci : dans une approche SOA radicale, les bases de données sont découpées selon l'architecture logique et délimitées par les frontières des ateliers logiques. Ceci conduit à des schémas de bases réduits à quelques tables. Le périmètre de ces bases se limite à celui des ateliers logiques.

Sans aller jusque là, l'architecture des données peut reconduire les habitudes avec des schémas plus larges, dimensionnés par rapport aux fabriques. Mais, même dans ce cas, il serait bon de s'interdire les manipulations qui débordent des ateliers et de refuser les jointures entre tables qui dépendent d'ateliers différents. Les jointures sont admises à l'intérieur des ateliers logiques, mais bannies entre les ateliers. Elles sont remplacées par des appels de services qui découplent les ateliers et les laissent évoluer en toute indépendance.

[2] En fait, en appliquant nos procédés de conception logique, deux machines pilotent une table : la machine élémentaire et la machine ensembliste (cf. *Guide de l'aspect logique*, référence PxM-40).

Orientations pour l'architecture des données (suite)

Comment tailler la base de données

L'échelle de la base de données

L'intention est de développer des composants assez autonomes pour faciliter le déploiement. En effet, l'architecture logique prépare le système en vue d'un déploiement complexe. Il ne s'agit pas seulement de penser le système d'information d'une entreprise, mais aussi d'anticiper la constitution d'un système de systèmes, englobant en un tout cohérent les systèmes partenaires[3].

Au niveau logique, l'unité de déploiement est l'Atelier logique, c'est-à-dire l'agrégat de niveau intermédiaire entre la Fabrique logique (équivalent en taille à un « domaine ») et la Machine logique (assemblage de services atomiques)[4].

Option extrême Une application maximaliste de l'approche SOA militerait en faveur de la délimitation des bases de données à l'échelle des ateliers logiques. De cette façon, l'atelier constituerait une parfaite unité de déploiement, emportant les données en même temps que les traitements.

Évidemment, cette option fragmente considérablement l'architecture des données. Sa validité dépend du niveau de maille retenu pour l'atelier : si les ateliers sont trop petits, limités à une table ou deux, l'échelle n'est plus compatible avec celle d'une base de données.

Même sans tomber dans ce travers, cette option d'architecture des données conduit à sous-exploiter la technologie des SGBDR, puisqu'elle réduit la portée des jointures et des contraintes d'intégrité.

Option intermédiaire Moins radicale, l'option intermédiaire choisit la Fabrique logique comme périmètre de définition des bases de données. On retrouve une pratique assez répandue, qui délimite la base par la notion de domaine. Plusieurs changements notables interviennent, toutefois :

- Le modèle sémantique est structuré en domaine d'objets, non en domaines fonctionnels. Le graphe d'architecture est donc différent. Même la définition de la donnée change : ceci se montre avec évidence dans le cas des notions génériques (personne, par exemple).
- Le souci de la non-redondance des données et de la réutilisation motive une architecture tendancielle, idéale, dans laquelle une même information ne figurerait qu'une seule fois (par exemple, l'adresse, qu'elle soit celle d'un client, d'un intervenant ou d'un expert ; ou l'objet, assuré ou endommagé).

Option dégradée Les options précédentes bouleversent la conception habituelle de l'architecture des données. Elles soulèvent la question des jointures et contraintes de longue portée, que l'on ne peut plus confier au SGBD, et, plus difficile encore, la question des transactions distribuées. En effet, quand le modèle des données s'architecture pour correspondre au modèle des services, les transactions fonctionnelles ont plus de chances de recouvrir plusieurs bases de données.

[3] On parle de SIIO : système d'information inter-organisationnels.

[4] Pour plus de précisions sur ces notions, se référer au *Guide de l'aspect logique*, PxM-40.

Orientations pour l'architecture des données (suite)

Le travail de conception

La structuration du MLD

Le modèle logique est structuré de la façon suivante :

- D'abord en strates : « Métier », « Organisation » et, éventuellement, « Interaction »[5].
- À l'intérieur de chaque strate : en Fabriques logiques[6].

Si on applique l'option intermédiaire décrite précédemment, le concepteur introduira un modèle logique de données dans chaque Fabrique. Pour cela, il crée un paquetage pour le MLD, à l'intérieur de la Fabrique et au même niveau que les Ateliers de celle-ci.

Si, au contraire, on applique l'option « radicale », alors le MLD sera décrit au sein de l'Atelier, au même niveau que les Machines logiques.

Les Ateliers ont un droit de regard sur le MLD de leur Fabrique (et uniquement de celle-ci, les autres informations leur étant fournies soit par passage de paramètres, soit par appel de services vers les autres Ateliers). Cette visibilité est explicitée par des relations d'utilisation ou dépendances montrées sur le diagramme de positionnement des Ateliers.

La vérification du MLD

Il va de soi que le MLD vérifie les formes normales définies par l'algèbre relationnelle. Elles sont à l'œuvre, déjà, dans la modélisation sémantique car elles permettent une meilleure expression du savoir « métier ».

Nous nous intéressons, ici, aux changements induits par l'approche orientée services sur la conception et la vérification du MLD.

Les dépendances que le concepteur indique sur le modèle (voir ci-dessus) accordent des droits trop importants. Toutes les machines de l'Atelier ont, formellement, la main sur toutes les tables. Le concepteur doit donc s'assurer que le principe d'encapsulation est respecté, c'est-à-dire :

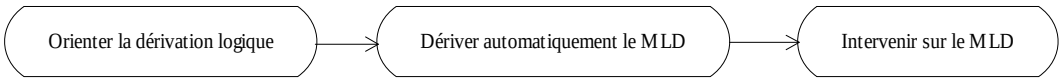
- que chaque Machine logique n'accède qu'à un nombre réduit de tables : les tables dont elle est explicitement propriétaire (une table principale, éventuellement des tables satellites telles que les types et les tables d'historisation) ;
- que chaque table n'est manipulée que par les deux Machines propriétaires, une Machine élémentaire qui charge les données d'une ligne, et une Machine ensembliste qui recherche et crée les lignes.

[5] La matérialisation de la strate « Interaction » dans le modèle logique permet de traiter le dialogue homme-machine, d'un point de vue logique, c'est-à-dire indépendamment de la technologie de l'interface et de la conception ergonomique.

[6] Dans la strate « Métier », les Fabriques logiques correspondent aux domaines d'objets du modèle sémantique ; dans la strate « Organisation », elles couvrent un organisme, un contexte de gestion (par exemple, des entreprises partenaires). Les Ateliers logiques de la strate « Organisation » traduisent les domaines fonctionnels. Un Atelier particulier est AL_Organisation qui offre les services transverses liés à la description de l'organisation, aux règles organisationnelles et au contrôle des habilitations.

Synthèse des décisions relatives aux données

La chaîne de transformation



	Orienter la dérivation	Dériver le MLD	Intervenir sur le MLD
Objectif	Apporter l'information nécessaire pour diriger la dérivation.	Produire automatiquement une première description des tables.	Produire le MLD définitif, avec tous les détails requis.
Support	Le modèle objet (sémantique ou partie du modèle pragmatique)	Le modèle logique des données	Idem.
Forme	Classes et annotations sur les classes et attributs.	Tables : classes stéréotypées « table »[7].	Idem + détails sur les colonnes et les PK et FK[8].
Décision	1. Demander la génération automatique de l'identifiant quand nécessaire. 2. Choisir une option de transformation de l'héritage. 3. Donner un nom pour les tables générées à partir des associations et, parfois, à partir des classes[9].[10]	Étape automatique (la seule décision de cette étape porte sur le périmètre de la base de données ; c'est une décision d'architecture des données, discutée dans les pages précédentes).	1. Renommer les « oid[11] » générés. 2. Décrire le contenu des identifiants. 3. Compléter la transformation dans tous les cas spéciaux (associations qualifiées et ternaires)[12]. Reprendre les rôles (colonnes et FK). 4. Supprimer les tables générées pour les codifications et modifier le type des codes. 5. Préciser les formats de colonnes.
Interventions complémentaires			• Les variables d'états. • Les données externes au domaine.

[7] Cette étape suppose l'utilisation du module SQL de l'outil UML.

[8] *Primary keys* (identifiants de tables) et *foreign keys* (pointeurs sur les identifiants d'autres tables).

[9] Le nom de la classe a été choisi pour rester au plus près du langage métier. Quand il contient des caractères incompatibles avec l'univers des SGBD, il est nécessaire d'associer à la classe, le nom de la table résultante.

[10] Ces décisions se marquent par des annotations sur le modèle (et non par modification du modèle lui-même).

[11] « oid » = *object identifier*.

[12] Les transformations automatiques, actuellement pratiquées, ne couvrent pas toutes les possibilités d'expression d'un modèle UML, surtout sémantique.

Synthèse des décisions relatives aux données

La distribution des décisions

Thème	Aspect sémantique	Aspect logique	Aspect physique
Persistance	La persistance des classes et des attributs se décide dès le modèle sémantique[13].	On peut décider de stocker en table les attributs dérivés[14].	Génération et optimisation du schéma physique de la base de données (DDL[15]).
Identification	Parmi les propriétés des classes, il peut s'en trouver qui soient identifiantes[16]. Elles sont désignées comme telles. Mais, sur le modèle sémantique, on n'inscrit pas d'identifiants artificiels qui n'auraient pas de valeur sémantique.	Les éventuelles propriétés identifiantes sont reprises en *primary keys* dans le MLD. Les tables qui dérivent de classes sans identifiant sont dotées d'un identifiant artificiel. Dans tous les cas, le concepteur définit le contenu de l'identifiant, sa construction.	Le compteur automatique (identifiant physique des lignes de la table) peut être utilisé comme solution d'identification.
Héritage	Le modèle sémantique recourt à l'héritage pour ordonner des concepts sous la forme de classifications.	Le style de SGBD cible peut comporter une solution naturelle de l'héritage[17]. Dans le cas contraire, les arbres d'héritage sont transformés selon l'une des options possibles, au cas par cas[18].	Optimisation pour mieux lier les tables impliquées dans la classification.

[13] En informatique de gestion, les informations décrites sous la forme d'attributs dans le modèle sémantique sont quasiment toujours persistantes. Par défaut, l'outil de modélisation désigne les classes et attributs comme persistants.

[14] Les attributs dérivés (au sens d'UML) se comportent comme des opérations : leurs valeurs sont obtenues par calcul ou navigation (par exemple, l'âge de la personne – obtenu à partir de la date de naissance). Un MLD normalisé ne retiendrait pas ces attributs car ce serait introduire une dépendance fonctionnelle entre les données. Par souci d'optimisation, le concepteur peut décider, tout de même, d'inscrire l'attribut dans le support de stockage, sous la forme d'une colonne. Cette solution est surtout intéressante quand l'attribut dérivé produit une information consolidée (calcul sur des ensembles d'objets) ou qu'il nécessite une navigation complexe.

[15] DDL : *Data Definition Language*. Ce sont les « create table », etc.

[16] Nous parlons bien de « propriétés » : il peut s'agir d'attributs, bien sûr, mais aussi d'opérations. Par exemple, en toute rigueur, le numéro de sécurité sociale devrait être restitué sous la forme d'une opération ou, mieux, d'un attribut dérivé, agrégeant une partie de la date de naissance, le sexe, etc. De cette façon, on évite la dépendance fonctionnelle. Ceci n'enlève rien à la capacité de ce numéro pour identifier la personne.

[17] La technologie du relationnel étendu ou de l'objet-relationnel fournit une solution pour l'héritage, en définissant des types et sous-types.

[18] Voir dans la suite de ce document.

Thème	Aspect sémantique	Aspect logique	Aspect physique
Association	Cardinalités, contraintes formelles : classes associatives, associations qualifiées, ternaires, agrégations et compositions, contraintes {*ordered*}, etc.	Le concepteur applique les règles de transformation fixées pour chacun des cas.	
Codification[19]	Le modèle sémantique recourt à des *data types* (UML), des codifications et des énumérations pour décrire les valeurs de référence, codifiées.	Le modèle logique propose un dispositif unique pour traiter les codifications. Il fixe aussi les relations entre les Machines logiques et ce dispositif.	L'architecture physique décidera du mode de déploiement de la solution pour la gestion des codifications (duplication des tables ou des structures…).
Historisation	Le modèle sémantique marque les notions particulières soumises à contraintes temporelles. Pas de solution générale[20].	Une solution généralisée d'historisation peut être élaborée au niveau logique.	La solution technique peut fournir une solution générale à l'historisation et à la sauvegarde, au niveau physique.
Multi-systèmes, multilinguisme	Impact sur la définition des codifications[21]. La devise est encapsulée dans la classe Montant qui assure les conversions nécessaires.	La solution pour les codifications tient compte des exigences multi-systèmes et multi-langues. Le traitement de la devise ajoute des colonnes dans les tables comportant des montants.	
Format des données	Le modèle sémantique décrit les attributs à partir des types atomiques, sans autres précisions, sauf cas exceptionnels[22].	Le MLD apporte toutes les précisions nécessaires sur les formats de données. Longueur, types pivots compatibles avec la technologie cible…	Conversion des formats pivots dans les termes du SGBD retenu. Éventuellement, contraintes sur les volumes (longueur des chaînes, textes, blobs…).
Types complexes	Attributs de nature : types (*data types*) ou classes.	Conversion en colonnes « simples ».	
États	Les automates à états modélisent le cycle de vie des objets métier.	Le MLD prévoit une colonne pour conserver l'état courant. Cas particuliers : états résumés et états concurrents (voir p. 23).	

[19] Les codifications sont des valeurs significatives, conventionnelles et limitées à des ensembles pré-définis. Elles associent, souvent, un code à un libellé intelligible. On les appelle, parfois, « tables de référence ». Cette expression présente deux défauts : d'une part, le premier terme, « table », dénote une solution technologique ; d'autre part, le terme « référence » renvoie à d'autres usages (dans les approches d'urbanisation, la donnée référentielle porte plutôt sur les objets métier, partagés entre les applications : par exemple, la personne).

[20] Le souci de l'historisation conduit à introduire la date dans la structure même du modèle et à créer de nombreuses associations ternaires. Plutôt que de sophistiquer ainsi le modèle, cette question n'est pas traitée dans la modélisation sémantique mais réapparaît dans le MLD.

[21] La dimension multi-systèmes découle de la volonté stratégique d'ouvrir le système aux partenaires.

[22] Ce n'est qu'exceptionnellement que le format précis d'une information revêt une valeur sémantique. Exemple : le numéro de sécurité sociale. Contre-exemple : la limitation d'un nom de famille à une certaine longueur n'a rien de sémantique.

Le travail préalable sur le modèle sémantique

Les considérations logiques ne doivent pas refluer sur le sémantique

Principe

La chaîne de transformation prévoit des interventions sur le modèle sémantique. Sans ces interventions, la transformation en modèle logique de données ne fonctionne pas.

Toutefois, on n'autorise de telles interventions que dans la mesure où elles n'altèrent pas la structure du modèle sémantique. Dans l'outil de modélisation, il s'agira d'introduire des annotations (*tagged values*), introduites dans l'outil de modélisation par le biais d'un profil UML, par exemple.

Les informations d'ordre logique

Ces informations qui vont diriger la dérivation du modèle de classes en MLD, sont les suivantes :

- la persistance désirée pour la classe ou l'attribut (cette décision est de nature sémantique) ;
- les noms à donner aux tables, si la reprise du nom de l'élément de modélisation (classe ou association) ne convient pas[23] ;
- l'option pour la transformation de l'héritage (ce thème est détaillé plus loin) ;
- la demande de générer automatiquement un identifiant (« oid » : *object identifier*).

L'identification

La génération du MLD ne fonctionne que si chaque classe comporte un identifiant. En l'absence d'un identifiant, les connexions entre tables (*foreign keys*) ne peuvent pas être construites. Dans le modèle sémantique, il y a deux façons de préparer l'identifiant des tables du MLD :

1. La classe peut contenir un ou des attributs identifiants. Ce sont des propriétés naturelles de la classe qui prennent une valeur distinctive sur chaque objet. On élimine, bien sûr, les attributs artificiels qui auraient pu être créés pour identifier les objets. Ils n'ont pas leur place dans le modèle sémantique[24].

2. Lorsque la classe n'a pas de propriété naturellement identifiante, le modélisateur demande, par annotation, la génération automatique d'un oid. La colonne résultante sera ensuite définie par le concepteur, sur le MLD.

Les classes imbriquées

Le modélisateur a pu utiliser l'imbrication des classes, dans le modèle sémantique, à seule fin de l'ordonner pour faciliter son exploitation. Cet usage dévoyé de l'imbrication pose problème lors de la génération du MLD. L'imbrication des classes a un sens précis, liés aux langages de programmation.

Il est donc nécessaire de supprimer les imbrications du modèle sémantique ou, si cette commodité est maintenue, de préciser la dérivation à adopter dans ce cas.

---

[23] Ce cas se présente systématiquement quand on s'impose des règles de nommage des tables. Le modèle sémantique, quant à lui, privilégie la facilité de lecture et la restitution de l'univers du discours. Il est nécessaire, au moins, de donner un nom pour les tables correspondant aux associations. En effet, les associations sont presque toujours nommées par un verbe au présent de l'indicatif, de façon à reconstituer les phrases du discours.

[24] Cas limite : en informatique de gestion, beaucoup de numéros (de client, de dossier…) tiennent ce rôle d'identifiant artificiel. Quand le modélisateur a un mandat pour innover sur la sémantique, il retire ces numéros du modèle sémantique. En analyse (observation sans invention), dans la mesure où ces propriétés sont connues des acteurs du système, le modélisateur les reprend dans le modèle.

Le travail préalable sur le modèle sémantique (suite)

Remarques complémentaires

Les devises

La classe Montant encapsule le traitement de la devise.

Ses attributs sont privés et l'information n'est manipulée que par des opérations qui possèdent un paramètre pour définir la devise dans laquelle travaille le demandeur.

Il est prudent de conserver la date de valorisation. Elle peut être nécessaire pour les conversions.

Le reste du modèle peut faire référence à la classe Montant, comme à un type atomique, c'est-à-dire sans passer par une association.

La dérivation du modèle en MLD transforme les attributs de type Montant en trois colonnes : pour la valeur, le code devise et la date de valorisation.

Le code devise lui-même renvoie à la table de codification.

La convention de nommage peut reprendre le nom de l'attribut pour la première colonne, ce même nom complété par « cd_devise » et « dt_valeur ».

La dérivation du modèle sémantique vers le MLD

Généralités

Directive principale

Le passage du modèle sémantique au modèle logique cherche à préserver le plus possible la lisibilité. Le concepteur limite donc les transformations appliquées aux noms des éléments du modèle (noms des classes qui deviennent les noms des tables ; noms des attributs dqui deviennent les noms des colonnes…). La dérivation exploite non seulement les noms des classes et des attributs, mais aussi les noms de rôles et de qualificateurs trouvés sur les associations.

Les conventions

Au niveau du projet ou, mieux, au niveau de la DSI, des conventions concernant les noms sont établies. Par exemple :

- « TB_ » : préfixe des noms de tables ;
- « id_ » : préfixe pour les identifiants ;
- « cd_ » : préfixe pour les codes ;
- « FK_ » : pour « foreign key » ;
- « PK_ » : pour « primary key » (généré par l'outil mais non retenu comme nom de colonne).

Dans cet exemple, on choisit d'utiliser les majuscules quand il s'agit de nommer les tables et les minuscules pour les noms de colonnes.

Contraintes sur les noms

Outre les conventions évoquées ci-dessus, les noms des éléments du modèle logique de données doivent vérifier les conditions suivantes :

- Ils sont conformes au standard SQL (on s'interdit les mots réservés du SQL ; les caractères spéciaux sont éliminés).
- Ils ne perturbent pas la génération automatique par le module SQL de l'outil UML (on peut avoir des surprises).

Les noms de tables ne sont pas astreints à refléter les noms de Machines. Il peut y avoir dissociation entre le modèle logique des données et le modèle des services.

La dérivation du modèle sémantique vers le MLD (suite)

La dérivation de l'héritage

Le relationnel étendu

Les SGBD relationnels étendus incorporent des dispositifs qui s'apparentent à la logique objet. La notion de sous-table donne une solution élégante et directe à l'héritage. Il s'agit de créer une table correspondant à la classe mère et autant de sous-tables qu'il y a de sous-classes.

Cette facilité est, pourtant, rarement autorisée par les responsables dans les DSI qui craignent de ne pas maîtriser ce comportement ou, simplement, qui n'ont pas pris la peine de l'évaluer.

Les trois options classiques

Quand on s'interdit d'utiliser le typage proposé par le SGBD, l'héritage présent dans le modèle « métier » (sémantique ou pragmatique) doit être transformé et mis à plat dans les tables.

Il y a trois options pour cette transformation :

- une table par classe ;
- une table par classe concrète ;
- une table pour l'ensemble des classes.

Une table par classe

La première option consiste à définir autant de tables qu'il y a de classes dans l'arbre d'héritage, que ces classes soient concrètes ou abstraites[25].

Cette option est la plus rigoureuse puisqu'elle évite la redondance des attributs et respecte la première forme normale : toute colonne reçoit une et une seule valeur pour toutes les lignes de la table.

Cette rigueur a un coût : le nombre de tables et les relations entre ces tables. Toutes ces tables entretiennent des relations qui reproduisent l'arbre d'héritage. Ces relations sont matérialisées par les clefs étrangères sur l'identifiant. L'identifiant est défini une première fois sur la table correspondant à la classe racine. Toutes les autres tables possèdent le même identifiant.

Sur la table racine, il est nécessaire, aussi, d'ajouter une colonne pour indiquer le type des occurrences. La valeur de cette colonne permettra de recomposer toute l'information de l'objet, en explorant les « sous-tables » associées à ce type. L'information peut, en effet, être disséminée sur plusieurs tables, si la classification présente plusieurs niveaux.

Une table par classe concrète

L'option intermédiaire permet de réduire le nombre de tables dans les cas où l'arbre d'héritage comporte des classes abstraites.

La contrepartie est la suivante :

- Les attributs des classes mères sont traduits en colonnes qui sont dupliquées sur plusieurs tables. D'où redondance structurelle.
- Les associations sur les niveaux hauts de l'arbre d'héritage doivent être « descendues » sur plusieurs tables, ce qui complique le modèle des données.

Une seule table

Quand les sous-classes portent relativement peu d'attributs et qu'il y a une grande proportion d'attributs communs aux classes concrètes, on peut être tenté de réunir toute l'information dans une seule table.

[25] Une classe est dite concrète quand elle peut être instanciée (on crée des objets à partir d'elle) ; elle est abstraite dans le cas contraire. Les classes racines d'héritage sont souvent abstraites, mais ce n'est pas toujours le cas. À n'importe quel niveau de l'héritage, une classe peut être concrète. En revanche, une classe feuille de l'arbre d'héritage n'aurait aucune raison d'être abstraite, quand le modèle est terminé.

Cette option ne respecte pas la première forme normale puisque les colonnes traduisant les attributs des sous-classes ne seront pas alimentées pour toutes les occurrences. Ceci complique l'enregistrement des attributs booléens. Pour le reste, cette solution simplifie le modèle des données.

Les critères de choix

Cette discussion montre que l'on ne peut pas arrêter une option qui s'applique de façon universelle. Le concepteur décide au cas par cas, en soupesant les avantages et les inconvénients :

- Quelle est la proportion des attributs communs par rapport aux attributs spécifiques (classes mères / classes filles) ?
- Y a-t-il beaucoup d'associations et comment sont-elles connectées aux classes de l'arbre d'héritage ?
- Accepte-t-on la normalisation et la sophistication nécessaire pour les valeurs booléennes ?
- Quels types de requêtes faudra-t-il construire ?

L'option 1 (une table par classe), pour rigoureuse qu'elle soit, entraîne une grande sophistication dans la composition des requêtes. Ce n'est pas forcément rédhibitoire puisque cette complexité est masquée par la Machine logique, propriétaire des tables.

Le type

Dans tous les cas où le MLD ne traduit pas tel quel l'arbre d'héritage, il faut que l'on puisse déterminer le type de chaque occurrence. Cette exigence conduit, le plus souvent, à ajouter une colonne pour donner le type (la sous-classe). Cette information permet de vérifier l'intégrité des données : ne remplir que les colonnes qui ont un sens pour le type, etc. Le type peut être traité comme une codification (code + libellé).

Les solutions intermédiaires

Pour un arbre d'héritage donné, le concepteur apprécie les critères indiqués ci-dessus. Si l'arbre est étendu (large et profond), le concepteur peut combiner les options : couvrir plusieurs classes concrètes par une seule table, tout en traitant le reste selon l'option 1, par exemple.

Ce travail doit être mené au cas par cas.

Bien que la logique de transformation de l'héritage se retrouve aussi pour les Machines logiques, les décisions à prendre sur le plan des données et celles sur le plan des services sont indépendantes. Les exigences structurelles et formelles diffèrent sur les deux plans. Par exemple, une information facultative se tolère plus facilement dans le contrat d'un service que dans le MLD.

La dérivation du modèle sémantique vers le MLD (suite)

La dérivation des associations

L'exploitation de la sémantique

On souhaite conserver, dans les modèles aval (logique et suivants), le plus possible de la sémantique et préserver les capacités d'expression qui ont été dégagées lors de la modélisation sémantique. Cette remarque s'applique au modèle des données comme au modèle des services. Les deux paragraphes suivants répondent à ce souhait.

Le nom de la table La table résultant de la dérivation provient soit d'une classe, soit d'une association.

Dans le cas d'une classe, la table peut reprendre le nom de la classe, tel quel ou additionné d'un préfixe (par exemple 'TB') qui évitera les confusions dans les discussions.

Dans le cas d'une association, le nom ne se dérive pas directement. En effet, nos règles de modélisation sémantique privilégient le verbe au présent de l'indicatif, pour nommer les associations. Le souci est de restituer le langage du métier et de pouvoir lire le modèle. Par exemple, le Sinistre (classe) *affecte* (association) un Objet (classe).

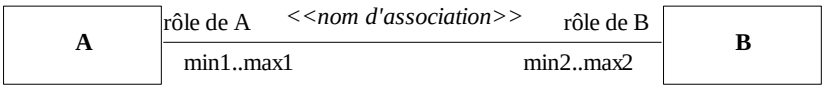
Il ne serait pas seyant de nommer les tables d'après le verbe. On préférera un nom qui peut s'obtenir par la « substantification » du verbe[26].

Les rôles Les colonnes contenant les clefs étrangères devraient être nommées par les noms de rôles quand ils existent sur l'association.

Ainsi, quand une table TB_B contient une clef étrangère vers une table TB_A, le nom le plus significatif pour cette colonne , est le nom de rôle de l'association, du côté de la classe A.

Dans la figure ci-dessous, il s'agit de « rôle de A ». Par exemple, « client », rôle de Personne dans une association à Contrat.

Figure PxM-41_5. Le dessin d'une association binaire



L'orientation des associations

L'architecture logique et, au niveau sémantique, le découpage en domaines d'objets introduisent des contraintes qui s'imposent au modèle logique de données. Le MLD doit en tenir compte (voir § sur l'architecture des données). **Ces contraintes d'architecture priment sur les règles de transformation données ci-après.**

[26] Ce nom est introduit, avant dérivation, sous la forme d'une annotation attachée à l'association grâce au module SQL de l'outil UML. L'intérêt d'annoter le modèle « métier » réside dans la conservation des décisions et la possibilité de générer plusieurs fois le modèle de données.

La dérivation du modèle sémantique vers le MLD (suite)

La dérivation des associations binaires

Les associations binaires ordinaires

Les associations binaires se dérivent selon les mêmes règles qui ont été formalisées par la méthode Merise (passage dur MCD au MLD) ou les approches entités-relations. Elles sont exposées dans les paragraphes suivants mais ne s'appliquent que sous la contrainte de ne pas violer l'orientation de l'association.

Les cardinalités Ces règles tiennent compte de la cardinalité de l'association. Dans la figure de la page précédente, min1 et max1 représentent, respectivement, le nombre minimum et le nombre maximum d'instances de la classe A qui peuvent être reliées à une même instance de la classe B, par l'association. Les valeurs que peuvent prendre les cardinalités sont les valeurs habituelles, notées en UML '0', '1' et '*' (pour 'n'). On peut aussi leur assigner une valeur spécifiée de façon absolue (un nombre) ou relative (le nom d'un paramètre)

Les combinaisons Dans les associations binaires, les cardinalités sont indiquées sur chaque branche. Compte tenu de la symétrie des rôles, ceci produit *dix* combinaisons qui sont détaillées dans le tableau suivant.

La dérivation

Les notions suivantes sont nécessaires pour exposer les règles de dérivation.

La transformation normalisée La transformation normalisée préserve les formes normales du modèle.

Le modèle sémantique applique, déjà, les trois premières formes normales, au minimum. Cette exigence formelle contribue à la qualité sémantique du modèle puisqu'elle pousse à dégager et isoler les concepts qui pourraient se cacher dans des concepts plus englobants.

La dénormalisation On pourra s'autoriser, sur justification, de « dénormaliser » le modèle des données. Le prix à payer est l'introduction de dépendance ou de redondance, et, le plus souvent, l'autorisation d'avoir des colonnes non renseignées.

Il est à noter que, pour certains attributs faiblement sémantiques (champs informatifs, commentaires… champs facultatifs), le modèle des classes, souvent, ne respecte pas strictement la première forme normale[27].

La table intermédiaire (ou relation) La « table intermédiaire », mentionnée dans le tableau, est la table relationnelle qui assure la connexion entre les deux tables « métiers ». Elle contient une clef étrangère vers chacune des tables reliées. L'identifiant de la table intermédiaire – sa clef primaire – s'obtient en concaténant les clefs étrangères. Cette mesure vérifie la définition de l'association qui interdit d'enregistrer plusieurs fois le même couple.

Cette définition, ainsi que la construction, s'appliquent quelle que soit l'arité[28] de l'association.

[27] La première forme normale (1FN) stipule que tout attribut doit avoir une et une seule valeur, pour chaque instance et à tout moment de son existence.

[28] L'arité d'une association est le nombre de branches. Sa valeur minimale est 2, puisque, même dans le cas d'une association réflexive, il y a deux rôles (cf. Raymond Devos, « Vous voyez bien qu'il y a deux bouts ! »). En théorie, il n'y a pas de valeur maximale. En pratique, la majorité des associations sont binaires, surtout si la modélisation sémantique n'a pas été poussée à son terme. La modélisation sémantique a le souci de dégager les déterminations des concepts. Ceci lui impose de recourir à des associations n-aires.

La dérivation du modèle sémantique vers le MLD (suite)

La dérivation des associations binaires (suite)

Figure PxM-41_6. Combinaisons des cardinalités pour une association binaire

Cas	min1	max1	min2	max2	Commentaire	Transformation normalisée	Transformation avec dénormalisation ou aménagement
1	0	1	0	1	Existence indépendante des objets reliés	Table intermédiaire	Dénormalisation possible : cas2 ou cas 3[29] .
2	0	1	1	1	L'objet B existe indépendamment de l'objet A. Mais tout A est relié à un B.	TB_A contient la référence à TB_B.	Si un B n'existe que pour un A : composition d'instance (voir p. 22).
3	1	1	1	1	Les deux notions, représentées par les classes A et B, sont fortement liées.	TB_A contient la référence à TB_B et réciproquement.	On peut se poser la question de l'opportunité de fusionner les 2 tables[30].
4	0	1	0	*	Existence indépendante des objets. A paraît dominant.	Table intermédiaire	Dénormalisation possible : appliquer cas 6[31].
5	0	1	1	*	Idem cas 4.	Idem cas 4 + contrainte de TB_A vers table inter.	Idem cas 4.
6	1	1	0	*	B est subordonné à A. Cas où A désigne le type de B.	TB_B contient la référence à TB_A.	Cette clef peut ou non appartenir à l'identifiant de TB_B.
7	1	1	1	*	Idem cas 6.	Idem cas 6.	
8	0	*	0	*	Indépendance des concepts. Vraie relation sémantique[32].	Table intermédiaire avec référence vers A et B.	Aucune autre solution n'est possible.
9	0	*	1	*	B semble plus capté dans l'orbite de A.	Idem cas 8 + contrainte de TB_A vers TB_B.	Aucune autre solution n'est possible.
10	1	*	1	*	Les deux notions ont un lien plus fort[33].	Idem cas 8 + contrainte dans les 2 sens.	Aucune autre solution n'est possible.

[29] Cette dénormalisation entraîne que les clefs étrangères peuvent ne pas être renseignées.

[30] La décision prend en compte le volume des données dans chaque table, le cycle de vie des objets (est-il indépendant ? les objets A et B vivent-ils sur le même rythme ? processus de sauvegarde…) et l'architecture des données (orientation des associations imposées par l'architecture logique, possibilité de localiser les tables sur des nœuds différents).

[31] Si on accepte que la référence à TB_A dans la table TB_B puisse ne pas être renseignée.

[32] L'association *__* est réellement sémantique car on est sûr qu'elle ne cache pas de dépendance dans la définition des concepts. Un modèle sémantique doit présenter une forte proportion de telles associations ; en contrepartie, il contiendra de nombreuses classes associatives.

[33] Il faudrait se demander ce que cache cette contrainte exprimée par les deux cardinalités minimales.

La dérivation du modèle sémantique vers le MLD (suite)

Les particularités sur les associations

Les associations réifiées

Une association est « réifiée[34] » quand on lui associe une classe. Une telle classe est dite « classe associative »[35].

Dérivation

Quand une association est réifiée et que la classe associative possède des attributs, elle est dérivée en une table intermédiaire, quelles que soient les cardinalités de l'association.

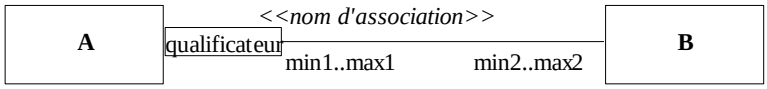
Le nom de la table reprend le nom de la classe associative (et non le nom de l'association).

Les associations qualifiées

Quand l'association porte un qualificateur, de nouvelles possibilités de dérivation apparaissent et il faut placer le qualificateur sur une table.

Le qualificateur fait, généralement, diminuer la cardinalité opposée (côté de B sur la figure ci-dessous).

Figure PxM-41_7. Le dessin d'une association qualifiée



Les grands traits de la solution

Selon les combinaisons de cardinalités, trois solutions types se présentent pour localiser l'information portée par le qualificateur :

1. La table intermédiaire[36] porte, en plus des identifiants, une colonne correspondant au qualificateur.
2. Une des tables reçoit le qualificateur qui sera valorisé sur chaque ligne, ainsi qu'une clef étrangère vers l'autre table.
3. La table TB_A comporte une structure qui contient les clefs étrangères pour toutes les valeurs du qualificateur. Cette structure peut être un tableau avec autant de positions qu'il y a de valeurs du qualificateur. Si les tableaux ne sont pas possibles, la table comporte autant de colonnes que de valeurs du qualificateur ; ces colonnes sont toutes des clefs étrangères vers TB_B.

Dans le cas où il y aurait plusieurs qualificateurs du côté de A, la table intermédiaire comporte autant de colonnes que de qualificateurs. La solution 2 est ajustée de la même façon. La solution 3 ne fonctionne qu'avec des tableaux multi-dimensionnels.

---

[34] « Réification » est un terme qui nous vient de la philosophie médiévale (du latin « *res facere* », faire une chose). Le terme a acquis une signification différente dans les méthodes de modélisation (cf. Jacques Ferber). La réification est l'acte essentiel de la modélisation : la décision que prend le modélisateur d'isoler une portion du réel pour le représenter sous la catégorie de la classe.

[35] En Merise, il s'agissait de la « relation porteuse ». UML (et l'approche objet, en général) apporte deux changements notables : d'abord, le fait que cette classe associative peut porter des opérations aussi bien que des attributs ; ensuite, la possibilité de lier cette classe par des associations (la relation de relations n'était pas possible en Merise).

[36] Cf. définition p. 16.

La dérivation du modèle sémantique vers le MLD (suite)

Les règles de dérivation pour les associations qualifiées

Les cas

Le tableau ci-dessous précise les règles de dérivation pour l'association qualifiée binaire. Les cas sont plus nombreux puisque la présence du qualificateur brise la symétrie de l'association.

Les solutions mentionnées dans le tableau sont :

1. La table intermédiaire portant le ou les qualificateurs.
2. Une des tables contient une colonne donnant la valeur du qualificateur et une autre qui est la clef étrangère. S'il y a plusieurs qualificateurs : il faut autant de colonnes que de qualificateurs mais une seule clef étrangère.
3. Une structure reflétant les valeurs du ou des qualificateurs et contenant les références à l'autre table.

Généralement, le qualificateur prend un nombre fixe de valeurs (un type énuméré, une position dans un domaine fini…). Si ce n'est pas le cas (réel, date…), la solution 3 n'est pas possible.

Figure PxM-41_8. Dérivation des associations qualifiées

Cas	min1	max1	min2	max2	Transformation normalisée	Transformation avec dénormalisation ou aménagement	Complément ou alternative
1	0	1	0	1	Table intermédiaire avec une colonne pour le qualificateur.	Dénormalisation possible : cas 2.	Sol. 1. Sol. 3 si on accepte colonne non renseignée.
2	0	1	1	1	TB_A comporte n colonnes, chacune référençant TB_B.		Solution 3 (tableau). Solution 1 possible.
3	0	1	0	*	Table intermédiaire avec une colonne pour le qualificateur.	Aucune autre solution possible.	Solution 1.
4	0	1	1	*	Idem cas 3 + contrainte de TB_A vers table inter.	Idem.	Solution 1.
5	1	1	0	1	TB_B avec attribut pour le qualificateur et réf. TB_A.	Les autres solutions sont possibles.	Solution 2. 1 et 3 possibles.
6	1	1	1	1	Idem cas 2 + contrainte TB_B vers TB_A	Les solutions 2 et 3 sont également envisageables.	Si solution 3 : une référence vers B n'existe qu'1 fois[37].
7	1	1	0	*	TB_B contient la référence à TB_A et le qualificateur.	On peut préférer la transformation 4.	Le nom de la colonne dépend de son positionnement.
8	1	1	1	*	Idem cas 3 + contrainte de TB_A vers table inter.		Solution 1.
9	0	*	0	1	Table intermédiaire portant le qualificateur.	Dénormalisation possible : solution 3.	Solution 1 (normalisée) ou 3.

[37] Il n'est pas possible d'avoir plusieurs occurrences de la même référence à une ligne de TB_B à l'intérieur d'une même ligne de TB_A, même avec des valeurs différentes du qualificateur.

Cas	min1	max1	min2	max2	Transformation normalisée	Transformation avec dénormalisation ou aménagement	Complément ou alternative
10	0	*	1	1	Idem cas 2.		Solution 3.
11	0	*	0	*	Table intermédiaire portant le qualificateur.	Pas d'autre solution. En fait, la contrainte vaut pour chaque valeur du qualificateur.	Solution 1. Vérifier adéquation entre nom de table et colonne
12	0	*	1	*	Idem cas 3 + contrainte de TB_A vers table inter.		
13	1	*	0	1	Cas 1 + contrainte sur TB_B.	Dénormalisation possible : appliquer cas 2..	Solution 1 ou 3.
14	1	*	1	1	Idem cas 2 + contrainte TB_B vers TB_A		Également : une référence vers B n'existe qu'1 fois[38].
15	1	*	0	*	Table intermédiaire et contrainte de TB_B	Pas d'autre solution.	Solution 1
16	1	*	1	*	Idem cas 15 + contrainte TB_A et TB_B / table inter.	Pas d'autre solution.	Solution 1.

[38] Il n'est pas possible d'avoir plusieurs occurrences de la même référence à une ligne de TB_B à l'intérieur d'une même ligne de TB_A.

La dérivation du modèle sémantique vers le MLD (suite)

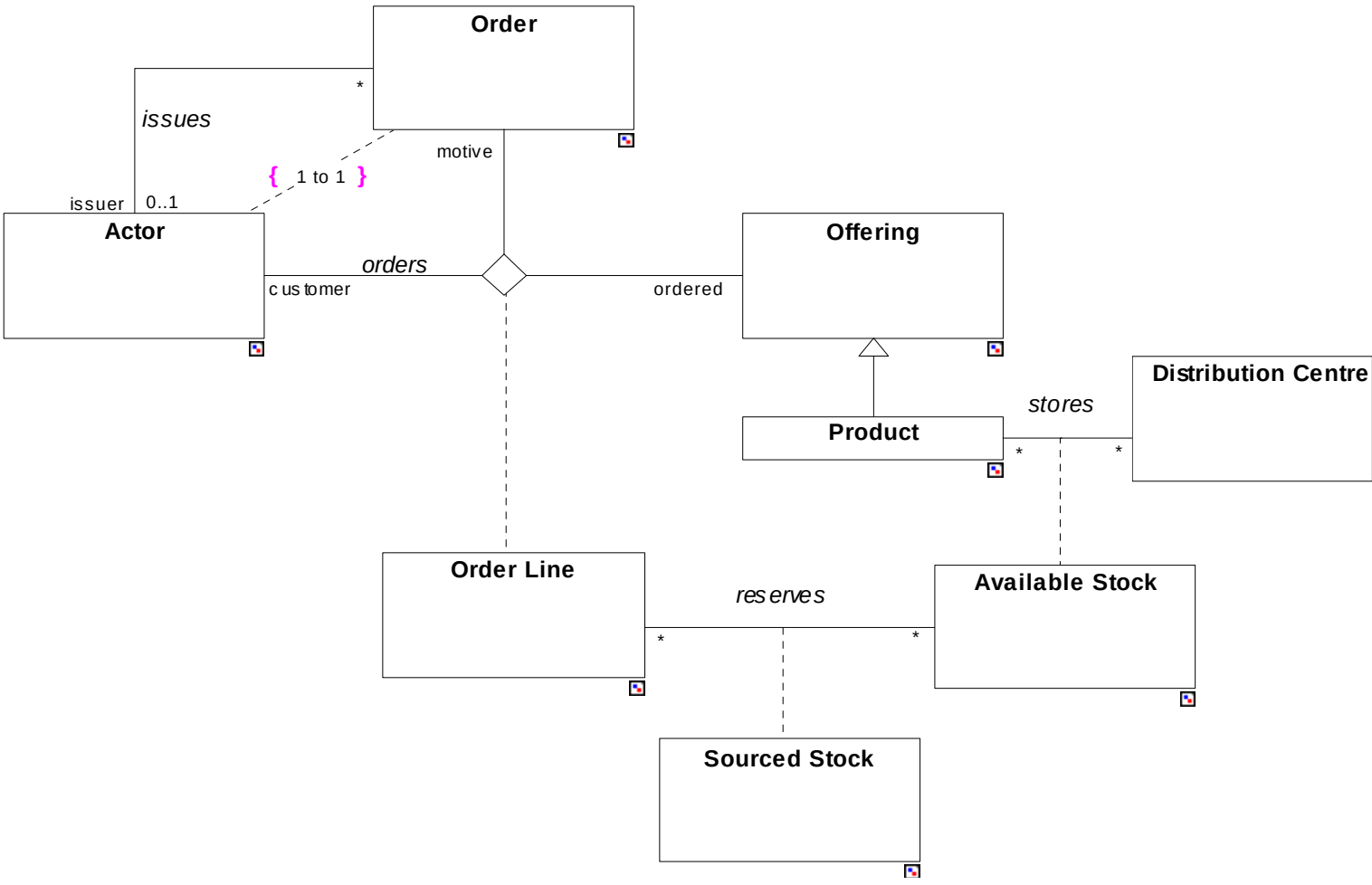
La dérivation des associations n-aires

Les associations n-aires

Le cas le plus fréquent est l'association ternaire, c'est-à-dire une association qui relie trois classes. Les modèles sémantiques comportent, généralement, de telles associations. La figure ci-dessous en donne un exemple.

Illustration Un acteur commande une offre (produit ou service). Ce faisant, il est considéré comme un client (ce que dit le rôle sur l'association). Si le modèle restituait cette phrase sous la forme d'une association binaire, on ne pourrait instancier qu'une fois le lien entre un acteur donné et une offre donnée (ce serait la ruine assurée !). La bonne représentation est donc l'association ternaire, qui introduit la motivation de l'acte et la date. C'est la classe « commande » qui constitue le troisième terme de l'association.

Figure PxM-41_9. Exemple d'association ternaire : « orders *»*



La dérivation du modèle sémantique vers le MLD (suite)

La dérivation des associations n-aires (suite)

La dérivation

On a du mal à représenter les cardinalités des associations n-aires sur le diagramme de classes. James Rumbaugh conseille de les traiter en commentaire du modèle.

La table intermédiaire

Pour la dérivation, cela n'a pas d'incidence car, quelles que soient les cardinalités de l'association, une table est créée pour l'association.

Cette table intermédiaire comporte autant de clefs étrangères qu'il y a de branches dans l'association.

L'identifiant de la table s'obtient par la concaténation de ces clefs étrangères.

Le cas de la classe associative

Dans l'exemple de la page précédente, l'association ternaire est réifiée par le concept de ligne de commande. La contrainte qu'exprime cette association est qu'il ne peut y avoir, dans une commande qu'une seule ligne portant sur un produit donné[39].

La prise en compte des contraintes

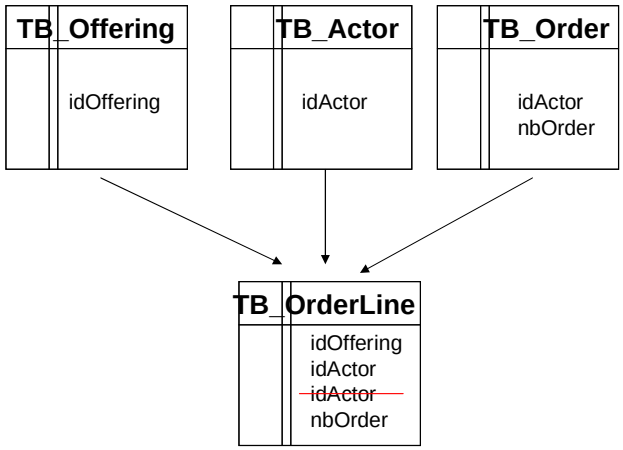
Le diagramme de classes de la page précédente montre une contrainte entre la Commande et l'Acteur. Cette contrainte précise que toutes les lignes rassemblées dans une même commande ne concernent qu'un et un seul acteur.

Si on a opté pour un identifiant de la commande relativement au client, alors l'application de la règle de dérivation énoncée ci-dessus double la colonne contenant l'identifiant de l'acteur. On retouchera donc le modèle résultant en retirant cette colonne.

Ainsi constituée, la clef primaire de la table intermédiaire suffit à vérifier la contrainte d'intégrité.

La figure ci-dessous illustration la transformation de la relation ternaire du modèle précédent, en tenant compte de la contrainte.

Figure PxM-41_10. Modèle logique des données pour la table « Order line »



Plus généralement, la commande est identifiée par un numéro propre et le problème ne se pose pas.

[39] Cette contrainte ne s'impose pas à l'interface homme-machine, laquelle peut très bien assouplir la saisie, avant que l'information soit consolidée.

La dérivation du modèle sémantique vers le MLD (suite)

Autres détails du modèle des classes

Les agrégations et compositions

L'agrégation n'est rien d'autre qu'une association. Elle ne réclame pas de règles de dérivations particulières.

La composition (au sens strict[40]) exprime une contrainte plus forte : la dépendance du composant à l'égard de la classe propriétaire. Cette circonstance incite le concepteur à absorber l'information du composant dans la table correspondant au propriétaire. La condition reste la cardinalité de l'association, du côté du composant (cf. règle sur les associations).

La contrainte {*ordered*}

Il est possible de mentionner, sur une branche de l'association, le mot réservé « *ordered* »[41]. Il n'a de sens que dans le cas d'une cardinalité multiple (pas forcément indéterminée). Cette contrainte stipule que les objets dans la liste sont ordonnés.

La dérivation conduit à ajouter un numéro de séquence (s'il n'existe pas déjà) à l'endroit où sont conservés les liens :

- soit sur la table intermédiaire (cas 1, 4, 8 du tableau p. 15) ;
- soit sur la table correspondante (cas 6 du même tableau).

La dérivation des associations : dénormalisation

La composition d'instances

Dans le cas d'une association dont les cardinalités sont 0-1__1-1, la dérivation normalisée est traitée par le cas 2. Cependant, il peut se trouver que, lorsqu'une instance de B est liée à une instance de A, elle ne puisse être liée à rien d'autre. D'autres instances, non liées à des instances de A, pourront avoir une existence autonome ou dépendre d'autres classes. Il s'agit de la composition d'instances.

Dans ce cas, la table TB_A pourrait absorber la table TB_B. On a affaire à une application dénormalisée car la même structure de données (l'ensemble des colonnes dérivées de la classe B) se trouvera dupliquée : elle se place dans la table TB_A et dans d'autres tables ou dans une table TB_B pour enregistrer les autres instances.

Ce cas est limite mais le raisonnement se transpose au niveau de l'architecture des données. B peut tenir le rôle d'un concept générique, par exemple Personne dans le domaine Réalité. La structure de données dérivée peut être dupliquée dans le MLD des Fabriques utilisatrices (domaine « Clientèle », domaine « RH »…).

[40] Elle est représentée sur le diagramme des classes UML par une ligne terminée par un losange noir (placé du côté du propriétaire, la classe composite).

[41] À ne pas confondre avec le nom de rôle « *ordered* » (commandé) utilisé dans le diagramme de la page précédente. Ce nom de rôle exprime un élément de la sémantique, alors que le mot réservé {*ordered*} (normalement entre accolades) impose une contrainte structurelle.

La dérivation du modèle sémantique vers le MLD (suite)

La dérivation des codifications

Le lien avec le dispositif pour les codifications

Les attributs des classes sémantiques dont la nature désigne une codification (type ou énumération) sont dérivés en colonnes dont la nature est : chaîne de caractères. Ces colonnes contiennent les valeurs d'identifiant sur la table Codification (pseudo-identifiant id_codification).

Voir, plus loin, « Le dispositif pour les codifications », p. 24. La longueur de la chaîne de caractères est donnée par ce chapitre.

La colonne conserve le code sans la langue. En effet, la langue dans laquelle la codification doit être exprimée dépend du contexte d'utilisation, non de l'objet lui-même.

La dérivation des automates à états

Une ou plusieurs colonnes

Afin de conserver l'état dans lequel l'objet s'est positionné lors de la dernière exécution, il est nécessaire d'ajouter une colonne dans la table qui lui correspond.

Plusieurs cas se présentent :

- Des états englobent des états plus fins : dans ce cas, on peut choisir soit d'ajouter une deuxième colonne (une par niveau de profondeur dans l'automate), soit de ne conserver que les valeurs détaillées.
- L'automate comporte des compartiments concurrents : dans ce cas, il n'est d'autre solution que de définir une colonne pour chacun des compartiments simultanés.
- L'automate peut être doté d'une mémoire (historisation d'état, indiquée par un 'H' dans un cercle sur les états historisables) : là aussi, il est nécessaire de prévoir une colonne supplémentaire.

Les valeurs d'états se distribuent ensuite sur les colonnes. L'affectation des valeurs aux colonnes peut ne pas être triviale. Mais seule la Machine logique a connaissance du mécanisme.

Les décisions sur le MLD

Le dispositif pour les codifications

Introduction

Le modèle sémantique mentionne des codifications sous la forme de types d'objets, de codifications (association d'un code et d'un libellé explicite) ou d'énumérations (codifications dont les valeurs sont figées, i.e. listes fermées et stables).

Dans tous ces cas, la structure de l'information est la même, ainsi que les exigences multi-systèmes et multi-langues. En conséquence, l'architecture logique propose un dispositif unique qui permet de réutiliser des services transverses et d'éviter la multiplication de petites tables.

Des services transverses

Les services et la structure de données sont définis dans la fabrique FL_Utilitaires. C'est dire que, logiquement, le dispositif de codifications est unique et réutilisable. Le dispositif prend la forme d'une machine ML_Codification. Elle se range dans l'Atelier AL_Thesaurus[42]. Cet atelier a pour vocation d'assurer la compréhension du système et de son comportement. Il regroupe les codifications et les messages (d'information et d'erreur). C'est l'organe de la communication avec les acteurs humains et, à ce titre, son contrat comprend le multilinguisme.

Il pourrait embrasser, également, les dispositifs d'aide en ligne, de dictionnaire-index et toute l'intelligence que l'on met dans le système (d'où le nom Thesaurus).

Les services de codification reçoivent le contexte de gestion. Ce paramètre permet de filtrer les codifications. Ils reçoivent, également la langue à utiliser.

Questions d'architecture

Par ailleurs, l'architecture logique obéit à une exigence : les Ateliers logiques, en tant qu'unités de déploiement, doivent être autonomes, autant que faire se peut.

L'architecte peut décider que tout Atelier logique gère ses propres codifications. Une telle décision augmente l'autonomie des Ateliers. D'un atelier, il ne sort aucun code s'il n'est accompagné du libellé qui lui donne son sens[43].

À cette fin, l'architecte logique lie tous les ateliers à AL_Thesaurus par une dépendance stéréotypée « import », signifiant par là qu'ils incorporent les services de cet atelier.

Quant à la question de la localisation des données, elle trouve sa réponse dans le choix d'architecture des données (voir p. 4). Chaque base de données doit posséder sa table de codification, en sorte qu'en ouvrant une seule base l'atelier trouve toute l'information dont il a besoin pour répondre (dans 90% des cas). Si le MLD est à l'échelle de la Fabrique logique, on aura un support des données de codification pour tous les ateliers de la fabrique.

[42] Autres noms candidats pour cet atelier : AL_Communication, AL_Contrôle, AL_Terminologie…

[43] C'est une solution exigeante. Une autre solution confie à la MLO le soin de demander la traduction code/expression. L'argument est que cette traduction ne sert que dans l'interaction avec l'humain.

Les décisions sur le MLD (suite)

Le dispositif de codification : modèle

Le modèle

Le diagramme de classes ci-dessous exprime la sémantique nécessaire aux codifications.

La contrainte impose que : une codification est autorisée pour un contexte de gestion si celui-ci utilise le type de codification auquel la codification est rattachée.

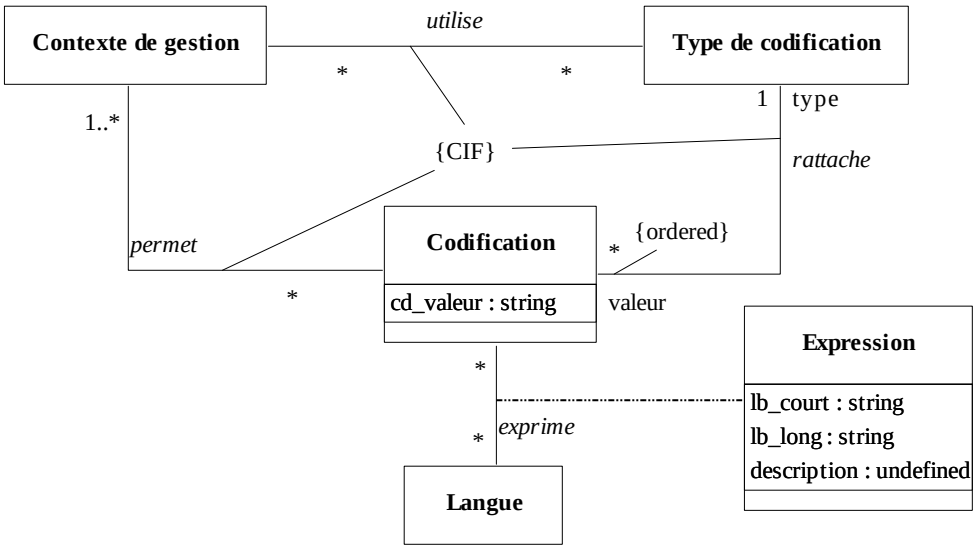
Le modèle prévoit que certaines codifications s'appliquent à plusieurs contextes de gestion (à la condition bien sûr, que ces contextes reconnaissent les mêmes types de codifications).

Pour un type de codification, les valeurs de codification sont ordonnées. Cette contrainte permet de classer les libellés présentés par l'interface homme-machine.

Les expressions (libellés et commentaires) supposent la langue connue : d'où la classe associative.

Les opérations fournies par le dispositif doivent permettre de filtrer les listes de codification par type et contexte.

Figure PxM-41_11. Le diagramme des classes pour le dispositif de codification



Les décisions sur le MLD (suite)

Le dispositif pour les codifications (suite)

La structure de la table

Le tableau ci-dessous donne le détail des colonnes de la table TB_Codification.

Colonne	Définition	Format
id_codification	Identifiant de la table. Il obéit à un format donné ci-après.	String [8]
cd_contexteGestion	Code identifiant le contexte de gestion (partenaire de l'entreprise étendue).	String [3]
cd_typeCodification	Désigne le type de codification (plusieurs lignes portent la valeur de type[44]).	String [2]
cd_valeur	Code de la codification (indépendant de la langue).	String [3]
cd_langue	Code de la langue dans laquelle le libellé est attendu[45].	String [2]
no_sequence	Permet d'ordonnancer les codifications dans la liste fournie pour un type.	Short integer
dt_validiteDebut	Date à partir de laquelle la codification s'applique (la période de validité inclut cette date). Si non renseignée : valide jusqu'à date de fin.	Date
dt_validiteFin	Date jusqu'à laquelle la codification s'applique (la période de validité inclut cette date). Si non renseignée : valide à partir de date de début, sans limite.	Date
lb_court	Libellé court pour la codification, pour la composition des *combo boxes*…	String [64]
lb_long	Libellé long, plus explicite.	String [256]
description	Un texte donnant plus d'explication « métier ». Pour interface homme-machine.	Text
commentaire_admin	Information liée à l'administration des données (origine, application, etc.).	Text

Tables complémentaires

Les tables suivantes aideront l'administrateur des données. L'atelier AL_Thesaurus devrait présenter une interface supplémentaire dédiée à l'administrateur.

- Type de codification : donne la signification de cd_typeCodification.
- Langue : traduit le code langue.
- Contexte de gestion : enregistre les partenaires.

NB : Ces tables sont des codifications : l'information pourrait être traitée dans la même table TB_Codification. Tout le dispositif serait couvert par une seule table.

[44] Cette table ne respecte pas la normalisation des données. Il y a, en effet, une dépendance entre les composantes de l'identifiant et les autres colonnes. Cette entorse se justifie pour faciliter l'exploitation de la table. Par exemple, une requête sur cd_typeCodification et cd_langage permet de rassembler les codes applicables.

[45] Un même contexte de gestion peut couvrir plusieurs langues. Par exemple, en Belgique : français et flamand.

L'identification des codifications

L'identifiant id_codification permet de lier les tables « métier » à la table TB_Codification. Ce sont ses valeurs qui sont conservées dans les colonnes des tables, comme clefs étrangères. Il constitue le code complet de la codification, contexte compris. En revanche, il n'identifie pas parfaitement les lignes de la table TB_Codification. En effet, il lui manque la langue. Pour une même valeur de id_codification, on peut avoir plusieurs lignes : c'est la même codification, mais les libellés sont exprimés dans des langues différentes.

La table TB_Codification est identifiée par l'ensemble des colonnes qui composent le pseudo-identifiant id_codification, plus la langue.

L'identifiant

La table TB_Codification est identifiée par l'ensemble des colonnes suivantes, qui constituent son véritable identifiant :

- cd_contexteGestion ;
- cd_typeCodification ;
- cd_valeur ;
- cd_langue.

Le pseudo-identifiant

Le pseudo-identifiant id_identifiant identifie la codification, mais pas la ligne dans la table. C'est une chaîne de caractères qui concatène :

- cd_contexteGestion ;
- cd_typeCodification ;
- cd_valeur.

Du point de vue de l'administration des données, id_codification n'est pas un identifiant mais il peut être intéressant d'en faire un index, pour accélérer les recherches.

Une autre possibilité : créer un deuxième identifiant comprenant id_codification et cd_langue.

Une solution plus puriste : on se passe de cette colonne (qui introduit des dépendances fonctionnelles), à charge pour les services de ML_Codification de décortiquer la chaîne reçue en paramètre et de retrouver les éléments d'information significatifs.

Relations dans le modèle logique

Que contient la FK dans les tables « métier » ? Le point est traité p. 23. L'intérêt d'imposer une cif entre les FK des tables métier et id_codification est modeste puisque, de toute façon, le SGBD ne peut pas vérifier la pertinence de la valeur par rapport au type de la codification associé à la colonne. La contrainte ne serait que partielle parce que la table englobe tous les types de codification.

C'est la machine ML_Codification qui fait les contrôles.

Les décisions sur le MLD (suite)

Autres décisions

L'historisation

Les besoins d'historisation et de traçabilité sont traités sur le MLD. Ce sujet n'est pas abordé ici. Notons seulement que la notation et l'outil UML peuvent être utilisés pour indiquer les besoins d'historisation. Ceci peut se faire sous la forme d'une annotation attachée soit à une classe, soit à un attribut, dans les modèles « métier ».

Dans les cas où ces besoins revêtent une valeur « métier », le renseignement devrait être reporté sur le modèle sémantique. Ceci laisse envisager une transformation automatique vers le MLD, tenant compte de l'historisation. Il est alors nécessaire d'ajouter la date, qui contribue à l'identifiant.

La construction des identifiants

Il peut arriver que le modèle sémantique désigne des propriétés identifiantes sur les classes. Ceci arrive rarement et le modèle sémantique suppose acquise « l'identité d'objet ». C'est donc au modèle logique des données que revient la tâche de fixer, systématiquement, les identifiants des tables.

L'identifiant se définit :

- soit à partir de données significatives, présentes sur la table (on préférera alors la construction d'identifiants composites, définis comme agrégeant plusieurs colonnes) ou référant les identifiants d'autres tables ;
- soit artificiels (compteur ou valeur arbitraire).

La construction des identifiants doit prendre en compte la dimension multi-systèmes.

Les formats

Le modèle sémantique ne définit que des natures d'attributs. Si un attribut obéit à des règles syntaxiques, elles sont données en commentaire. Le MLD doit préciser le format :

- type compatible avec le SGBD ;
- longueur pour les chaînes.

Il reprend les contraintes définies sur les attributs et, chaque fois que cela est possible, il les confie au SGBD (champ obligatoire, plage de valeurs…).

Les champs documentaires

Le concepteur peut ajouter des colonnes qui permettront aux acteurs du système d'enregistrer des commentaires, en plus des données de gestion (chaînes de caractères, textes, liens vers des notes…). Ces ajouts peuvent être systématiques.

Les attributs et associations dérivés

UML permet de définir un attribut ou une association comme « dérivé », c'est-à-dire comme pouvant être déduit du reste du modèle, soit par calcul, soit par navigation.

La dérivation automatique du MLD devrait exclure ces éléments, car, dans le modèle des données, ils introduiraient une dépendance fonctionnelle et un risque de contradiction de l'information.

Toutefois, pour des raisons d'optimisation, le concepteur s'autorise parfois à alourdir le MLD avec des colonnes ou des clefs qui correspondent à ces éléments. La dérogation doit être exceptionnelle. On peut envisager une telle décision encore sur le modèle physique des données. Le moment est, d'ailleurs, plus opportun puisqu'à ce stade on aura une meilleure connaissance du comportement du système.

Éléments pour la modélisation physique des données

Les décisions de l'aspect physique

Avertissement

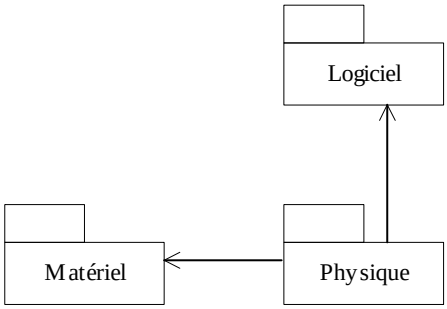
Le document sur le modèle logique des données ne devrait pas couvrir l'aspect physique. Cependant, il a semblé utile de lui ajouter ce complément afin de montrer comment se distribuent les décisions relatives aux données, sur tout le processus. Le souci est de sérier les décisions et d'éviter que des préoccupations d'un niveau aval ne viennent polluer un modèle en amont.

L'aspect physique

Le modèle physique des données se range dans la représentation de l'aspect physique. La figure ci-dessous rappelle le positionnement de cet aspect dans le cadre général. Le modèle physique fait référence, à la fois, à l'aspect logiciel et à l'aspect matériel. Dans le premier, on trouve les composants logiciels ; dans le deuxième, les « nœuds » de l'architecture matérielle, c'est-à-dire les machines physiques.

La conception de l'architecture physique consiste à localiser les composants logiciels sur l'architecture matérielle. Le guide PXM-80 décrit ce travail.

Figure PxM-41_12. Le positionnement de l'aspect physique dans la topologie du Système d'action



Le déploiement

L'architecture physique fixe les décisions de déploiement. En ce qui concerne l'architecture des données, la réflexion couvre :

- la délimitation des bases de données physiques[46] ;
- la création de plusieurs bases pour le même MLD, avec éventuellement des nuances sur le MPD[47] ;
- la dynamique de duplication / synchronisation des données ;
- l'importation conformément aux indications de l'architecture logique[48] ;
- l'optimisation (emploi des schémas, espaces de tables, index…).

[46] La dérivation naturelle est, bien sûr, de faire une base de données par MLD. Mais, on peut, pour diverses raisons s'écarter de l'isomorphisme et faire déborder le MPD sur plusieurs MLD. Par exemple au stade du prototype, on peut construire le MPD à partir du MLD de la fabrique centrale et en agrégeant des tables provenant d'autres fabriques logiques. Dans la conception de la cible, ce type de décision peut être motivé soit par l'optimisation, soit par des contraintes géographiques ou techniques.

[47] Ces nuances peuvent provenir de l'utilisation de SGBD différents ou de la différence entre les configurations physiques.

[48] C'est le cas avec les données issues de la FL_Utilitaires (codification, messages…), quand l'architecte logique choisit l'importation.

Éléments pour la modélisation physique des données (suite)

Les éléments de l'architecture des données, au niveau physique

Les éléments de l'architecture physique

L'architecture physique se représente avec le diagramme de déploiement.

Nœud Les nœuds de l'architecture sont les machines physiques, dont les serveurs de bases de données.

Connexion Entre les nœuds, l'architecture établit des connexions en indiquant le protocole de communication.

Les éléments propres aux bases de données

Schéma Le schéma spécifie la structure d'une base de données.

Il contient la définition des tables, colonnes et de leurs relations et contraintes. Un schéma est un composant. En tant que tel, il figure dans le modèle logiciel. Le schéma peut se manifester sous deux formes équivalentes : une représentation (le MPD) et sa traduction en DDL[49].

Base de données La base de données est un composant physique, localisé sur un nœud, réservant de l'espace et qui sert de support de masse, c'est-à-dire qui assure la persistance des données.

Elle est construite conformément à un schéma (ou créée à partir d'un DDL).

Espace de table La « *tablespace* » est une unité de stockage. Elle se construit comme une portion d'une base de données, réservant un certain volume de données. Ce volume est délimité par le choix de tables et d'index[50].

Dans l'architecture physique, l'espace de table est une partie d'une base que l'on pourra localiser indépendamment (sur une partition).

Partition Il s'agit d'une partition physique : un espace réservé sur un support de stockage.

Index L'index est un mécanisme fourni par le SGBD et qui permet d'accélérer les recherches en tables.

Les index sont ajoutés par l'administrateur des données, à partir du constat ou de l'estimation des performances du SGBD.

La conception du MPD

Le MPD est un élément de l'architecture physique. Sa conception repose sur la connaissance pointue du SGBD retenu[51]. Il exprime des décisions de localisation et d'optimisation.

---

[49] *Data Description Language*.

[50] In E.J. NAIBURG, R.A. MAKSIMCHUK, *UML for database design*, Ed. Addison-Wesley, p. 285 : « *tablespace : a construct representing an amount of storage space that is to be allocated to tables, indexes, and so on*. »

[51] Par opposition au MLD qui ne supposait que la connaissance des principes de la technologie choisie (ici, relationnel classique) et des bonnes pratiques de la conception des données (formes normales, règles de dérivation).

Limite du MLD

Le modèle de données ne saurait assumer tout ce qu'exprime le modèle « métier », sans même parler des opérations et des automates. En effet, l'association entre classes montre un comportement dynamique qui échappe au MLD. Si deux instances sont reliées par un lien d'association et que l'une d'entre elles est supprimée, alors le lien aussi disparaît. Il faut en tirer les conséquences sur la base de données. C'est le genre de comportement que l'on peut confier aux SGBD actuels. On peut aussi préférer l'inscrire dans les services. Cette dernière solution est plus souple et permet de vérifier d'éventuelles contraintes.

Index