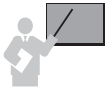
Identification et incidence sur la réification

Cette section concerne le processus d'identification des entités et des associations pour les inconditionnels de la méthode Merise. Les concepteurs UML feront l'analogie avec leur notation favorite.

Les écrits qui suivent sont un résumé des remarques formulées par Dominique Nanci qui a été un acteur dans l'élaboration du formalisme entité-relation qui est devenu entité-association, puis utilisé dans la méthode Merise. Par ailleurs, Christophe Sibertin-Blanc avait présenté un article entièrement dédié à cette problématique aux journées de l'AFCET en novembre 1990.

Identification absolue d'une entité



L'identifiant est composé d'un ou plusieurs attributs dont les valeurs permettent de désigner de manière unique chaque occurrence de l'entité. On parle d'identifiant simple quand l'attribut est unique (ex : `immatAvion` pour l'entité `Avion`). On parle d'identifiant composé quand il est composé de plusieurs attributs (ex : `nom+prénom+dateNaissance` pour l'entité `Etudiant`).

Pour assurer une identification absolue, tous les attributs composant l'identifiant de l'entité doivent être intrinsèques à l'entité (sémantiquement rattachés à cette entité). L'entité disposant d'une identification absolue modélise un objet disposant d'une identité propre, d'une indépendance de désignation. On peut faire référence à ses occurrences sans besoin d'autres entités. C'est la modélisation la plus simple, basique de Merise, que l'on cherche souvent à réaliser quitte à « inventer » des identifiants artificiels.

Identification relative

Dans certaines situations, l'identification d'une entité nécessite de se positionner relativement par rapport à une autre entité. Par exemple, l'entité `LigneCommande` s'identifie assez naturellement par le numéro de ligne par rapport à sa commande. Cela sous-entend que les entités `Commande` et `Ligne` sont associées par une relation d'appartenance.

Bien qu'intrinsèque à l'entité `Ligne`, le numéro ne peut être un identifiant absolu. Relativement à sa commande (identifiée par exemple par un numéro de commande), il joue alors le rôle d'identification d'une occurrence de l'entité `Ligne`.

L'identification relative d'une entité est donc construite à partir d'un ou plusieurs attributs intrinsèques à l'entité et d'autres entités auxquelles elle est reliée par des associations binaires de cardinalités 1,1.

Nous avons vu que l'outil Win'Design utilise le symbole (R) sur le lien d'association de cardinalités (1,1) connecté à l'entité faible. Nous verrons au chapitre 4 que l'identifiant relatif est noté au niveau du lien de l'association.

Cette notion d'identification relative a connu une histoire. Elle est abordée dans l'équipe de recherche initiale et figure dans [TAR 79b] sous une forme « confidentielle » de quelques lignes. L'ouvrage de base sur Merise [TAR 83], faisant explicitement référence pour une présentation plus détaillée du formalisme entité-relation au précédent ouvrage, passe sous silence cet aspect. Exit donc l'identification relative… À partir de 1987, Arnold Rochfeld et José Morejon, dans plusieurs articles, réhabilitent cette notion qui s'intègre désormais complètement dans la Merise/2 ainsi que dans la plupart des outils associés.

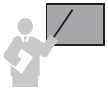
Identification d'une association

Par définition, une association n'a pas d'identifiant propre, son identification est construite par la composition (concaténation) des identifiants des entités connectées. C'est donc un abus de langage que de parler de l'identifiant d'une association, et encore plus comme certains auteurs, de faire figurer ces propriétés explicitement dans la relation.

Cette définition de l'identification d'une association implique qu'à un *n*-uplet d'occurrences des entités qui sont connectées (matérialisées par les identifiants), il n'y a qu'une seule occurrence de l'association. On peut le noter ainsi : E1 x E2 x… $\rightarrow$ A.

Par contre, il n'a jamais été dit que la réciproque était systématique ! Ce n'est pas parce qu'un concept perçu est identifiable par la composition d'identifiants d'entités qu'il doit obligatoirement être modélisé comme une association. Certes, c'est souvent le cas mais cela peut aussi être modélisé *via* le concept des entités faibles.

Identifiant alternatif



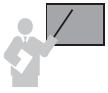
Une entité peut présenter plusieurs identifiants potentiels. Celui retenu est appelé identifiant principal, les autres, identifiants alternatifs.

Cependant cette situation, compte tenu de la difficulté de trouver un identifiant, reste assez exceptionnelle (sauf dans le cas où cohabitent un identifiant naturel et un identifiant artificiel).

Cette notion apparaît plus souvent dans des graphes d'héritage entre entités. Les sous-entités (entités sous-type), qui généralement préexistent, sont souvent dotées d'identifiant absolu. La sur-entité (l'entité sur-type) possède également son propre identifiant. Dans ce cas, les identifiants des sous-types sont qualifiés *d'alternatifs* car ils possèdent par héritage l'identifiant de leur sur-type.

Considérons l'exemple des entités sous-type `Assure` (d'identifiant absolu `numINSEE`) et `Cotisant` (d'identifiant absolu `numURSSAF`) qui héritent de l'entité `Adhérent` (d'identifiant `numAdherent`). L'identifiant alternatif de l'entité `Assure` devient `numINSEE` et celui de l'entité `Cotisant` devient `numURSSAF`. L'identifant absolu de ces deux entités devient, par héritage, celui de l'entité sur-type à savoir `numAdherent`.

Entité faible



On qualifie d'entité faible, une entité dont l'identification est réalisée relativement à plusieurs (2 à *n*) associations, sans disposer d'attribut intrinsèque intervenant dans l'identification. Son identification est donc identique à celle d'une association. Cependant, ce n'est pas parce que cette entité est identifiée comme une association qu'elle en est une.

Le choix de modélisation en entité ou association n'est pas un problème déductif mais perceptif. C'est avant tout l'intérêt sémantique du concepteur qui décide d'abord de modéliser un concept en tant qu'entité (ou association) et qui se pose ensuite la question de son identification. Cette modélisation par une entité faible correspond à une évolution dans le discours du concepteur.

Au début, le concept à modéliser apparaît comme une forme verbale (une association) : *un avion est affrété par une compagnie*. Puis progressivement, la forme verbale devient substantive : *pour chaque affrètement…, les vols affrétés…* Ce groupe nominal (cet objet) intervient à son tour dans des associations.

Dans l'exercice concernant les affrètements d'avions dont vous trouverez l'énoncé plus bas, on ne peut pas parler d'entité faible car le vol dispose d'une propriété composant l'identification relative (la date du vol). En revanche, un cas d'entité faible se présente dans les exemples d'associations d'agrégation à propos de la mission des véhicules qui visitent des chantiers.

Dans la sémantique du discours, sous-tendue par la structure grammaticale, on est passé progressivement d'une association (groupe verbal) à un objet (groupe nominal). Ce phénomène est appelé réification ou substantification. La modélisation conceptuelle, dans l'esprit de Merise, est avant tout une vision sémantique : la perception en entité ou association prime sur la manière de l'identifier. Ce n'est pas l'identification qui détermine la modélisation, mais le choix de modélisation qui implique l'identification.

Le tableau suivant résume l'évolution de la modélisation en entité ou relation (terme utilisé initialement pour parler d'association).

Tableau 1.8 Classification des entités

Relation	Association entre des entités, identification déduite par construction, aucune existence propre, position de groupe verbal
Entité faible	Objet d'intérêt, obtenu souvent par substantification en position de groupe nominal , identification relative multiple sans attribut identifiant intrinsèque
Entité relative	Objet d'intérêt, identification relative simple ou multiple avec toujours un attribut identifiant intrinsèque
Entité dépendante	Objet d'intérêt, identification absolue, impliquée dans une ou plusieurs associations avec une cardinalité 1,1, objet autonome
Entité	Objet d'intérêt, identification absolue, objet indépendant

Exemple récapitulatif

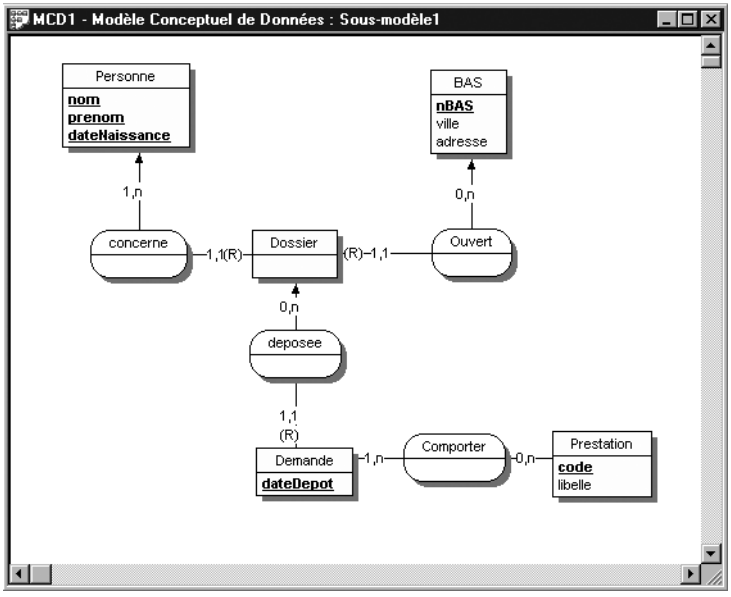
La gestion de l'Aide Sociale (AS) relève de l'autorité des Conseils Généraux (CG). Chaque CG a construit son système d'information. Les personnes souhaitant bénéficier des prestations de l'AS déposent leur dossier auprès du Bureau d'Aide Social (BAS) de leur municipalité.

Les personnes sont identifiées par leurs nom, prénom, date et lieu de naissance. Les BAS sont identifiés par un numéro départemental. Ces personnes déposent, dans le cadre de leur dossier, des demandes (identifiées par la date de dépôt) dont chacune comporte les prestations demandées. Normalement, une personne ne doit ouvrir qu'un dossier. La mise en commun des dossiers au niveau départemental a révélé que dans certaines grandes villes disposant de plusieurs BAS, certaines personnes disposaient, illégalement, de plusieurs dossiers.

Solution Merise/2

La modélisation 1-93 fait apparaître une entité faible `Dossier` et une entité relative `Demande`. L'identifiant relatif de `Dossier` est (`nom`, `prenom`, `dateNaissance`, `nBAS`). L'identifiant relatif de `Demande` est (`nom`, `prenom`, `dateNaissance`, `nBAS`, `dateDepot`).

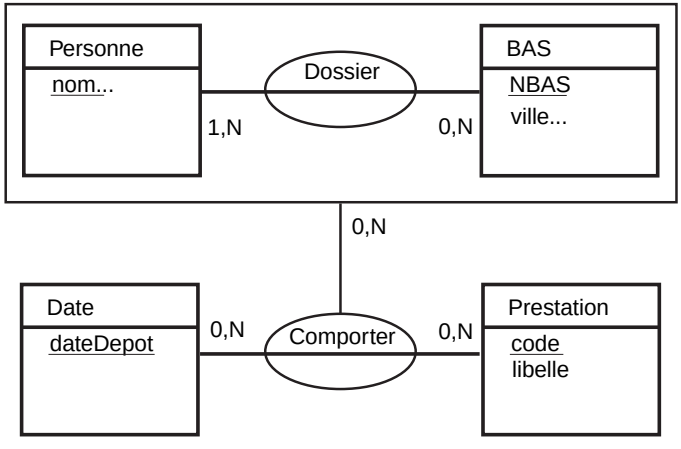
Figure 1-93 *MCD Win'Design*



Solution par les associations d'agrégation

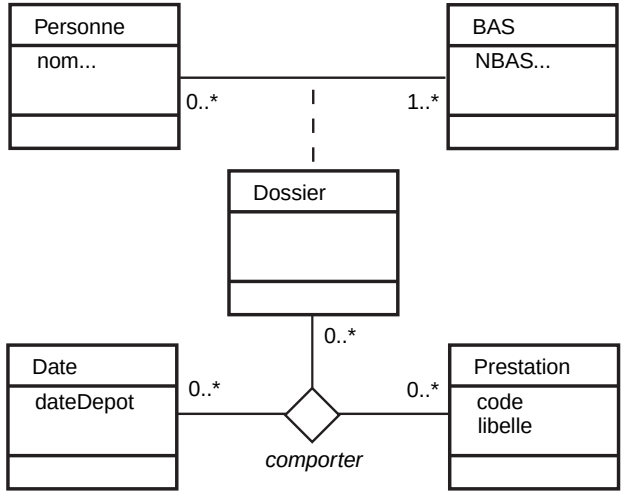
En utilisant les associations d'agrégation, avec lesquelles il est plus facile de faire l'analogie avec le formalisme UML, on obtient le schéma 1-94. Au niveau logique, on économiserait une relation (il n'est pas nécessaire de formuler une demande si elle n'est pas rattachée à une prestation).

Figure 1-94 *Solution avec les association d'agrégation*



Le diagramme UML 1-95 permet de visualiser l'analogie entre les deux formalismes.

Figure 1-95 *Solution avec UML*



Aspects temporels

Il est très fréquent d'avoir à prendre en compte des aspects temporels au niveau conceptuel. On peut classifier trois modélisations : le moment, la chronologie et l'historisation.

Évitez de nommer un attribut temporel par un mot réservé du langage informatique (`date`, `month`, `day`, etc.). Préférez des identificateurs plus parlants (exemple : `dateNaissance`, `dateVente`, `moisArrivee`, etc.)

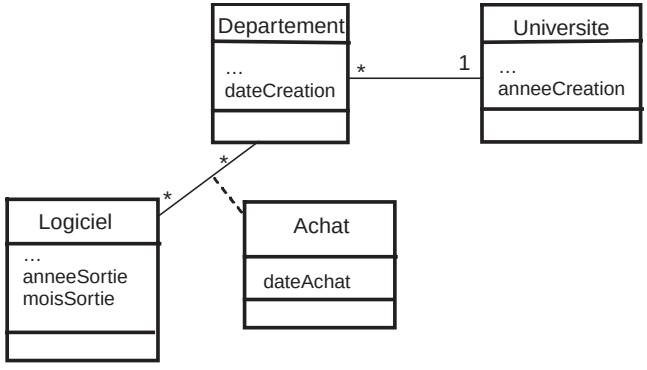
Modélisation d'un moment

Il s'agit de représenter des informations indiquant un moment (le plus souvent une date). Ce moment est représenté sous diverses formes : *jour*, *mois*, *jour*/*mois*, *année*, *mois*/*année*, *jour*/*mois*/*année*, avec ou sans les heures, minutes et secondes, etc.). Le plus souvent cet attribut sera positionné dans une classe.

L'exemple 1-96 décrit quelques attributs temporels. On peut supposer que `anneeCreation` et `anneeSortie` seront au format *année* (entier sur 4 positions), `dateCreation` au format *mois*/*année*, `moisSortie` au format mois (entier sur 2 positions) et `dateAchat` au format *jour*/*mois*/*année*. Ce dernier attribut se trouve dans la classe-association car un département

peut acheter à plusieurs dates, de même un logiciel peut être vendu à plusieurs dates. La modélisation indique ici qu'un département n'achète pas plusieurs fois le même logiciel, et cette date correspond à l'achat d'un logiciel par un département.

Figure 1-96 *Modélisation d'un moment*

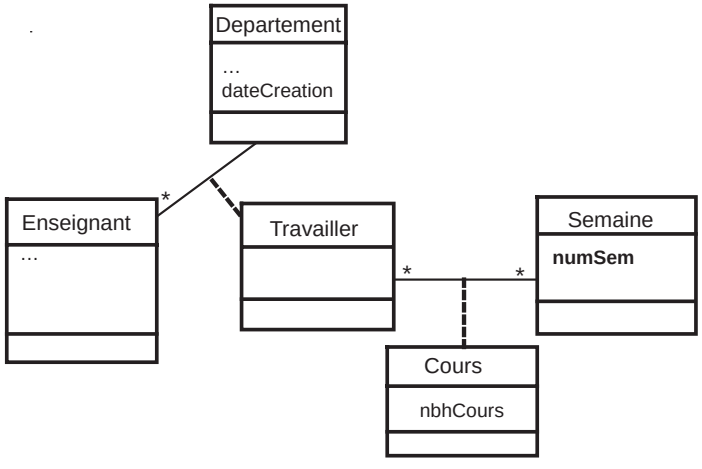


Modélisation de chronologie

Il s'agit de modéliser des situations où le temps intervient de manière régulière (plannings, emploi du temps, prévisions, statistiques, etc.). Le plus souvent cet attribut sera considéré en tant que classe. Les formats de cette donnée peuvent aussi être divers (*jour*, *mois*, *jour*/*mois*, etc.).

L'exemple 1-97 décrit l'attribut de chronologie `numSem` qui identifie la classe `Semaine`. Ici on suppose que des enseignants peuvent travailler dans plusieurs départements et que l'on souhaite connaître le volume de leurs cours pour chaque semaine.

Figure 1-97 *Modélisation d'une chronologie*



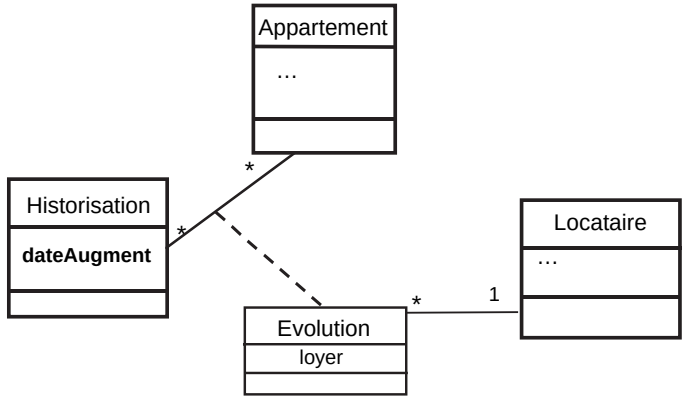
Modélisation de l'historisation

Ce type de modélisation est à utiliser quand on désire conserver les valeurs antérieures d'un attribut. Citons quelques exemples comme le montant d'un loyer (on désire connaître son évolution) ou la valeur du coefficient bonus/malus d'un contrat d'assurance. En contre-exemple, le nombre de passagers d'un vol donné sera plutôt traité comme une chronologie (exemple traité dans les exercices).

Alors que Merise/2 met à disposition un artifice pour noter l'historisation (symbole H sur un lien d'association), il est possible de noter avec UML cette modélisation en ajoutant un attribut de type moment.

L'exemple 1-98 décrit l'historisation des montants de loyer des appartements L'attribut `dateAugment` identifie la classe `Historisation`.

Figure 1-98 *Modélisation d'une historisation*



La démarche

Les sources d'information à utiliser lors de la modélisation peuvent être diverses : énoncé en langage naturel (c'est le cas des exemples du livre présentés à l'aide d'un texte introductif), interviews, observation du système, états de sortie papier, copies d'écran, etc.

Dans tous les cas, on doit se ramener à des phrases élémentaires en prenant garde d'éviter de formuler des propositions incomplètes, redondantes, contradictoires ou fausses.

Décomposition en propositions élémentaires

Une proposition élémentaire est assimilée à la composition *sujet-verbe-complément*. Il est courant d'avoir à reformuler certaines phrases lues ou entendues. Ainsi la proposition « Un

avion est caractérisé par une immatriculation, un nombre de places assises et appartient à une compagnie. » devra être décomposée en :

- « Un avion est caractérisé par une immatriculation. »
- « Un avion est caractérisé par un nombre de places assises. »
- « Un avion appartient à une compagnie. »

Un autre exemple, la proposition « Le prix de vente d'un avion devra être proportionnel au nombre de places assises. » devra être décomposée en :

- « Un avion a un prix de vente. »
- « Un avion est caractérisé par un nombre de places assises. »
- « Le prix de vente est proportionnel au nombre de places assises. »

Propositions incomplètes

Il ne doit pas exister de proposition incomplète. Ainsi une seule proposition mettant en jeu trois entités « Un logiciel est installé par un département sur un serveur. » risque d'être incomplète. En effet, il faut préciser ce fait en considérant les entités deux-à-deux.

- « Un département peut-il installer tout logiciel? »
- « Un serveur peut-il accueillir tout logiciel? »
- « Un département peut-il utiliser tout serveur? »

Je doute fort que la réponse soit « oui » à chacune de ces propositions. L'association ternaire représentant l'installation devra donc soit être décomposée, soit inclure une contrainte. Pour les associations de degré supérieur, il est possible de généraliser ce raisonnement.

Propositions redondantes

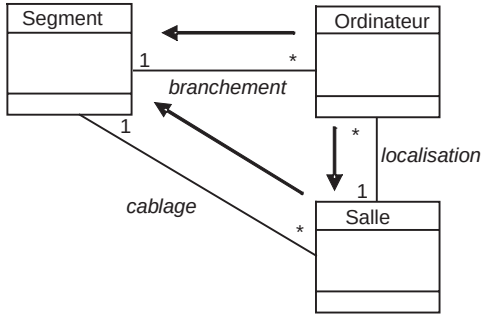
Il faut éviter de conserver des propositions redondantes. Parmi les trois propositions suivantes, une est redondante car déduite des deux autres.

- « Un ordinateur est branché à un segment réseau. »
- « Un ordinateur est situé dans une salle. »
- « Seul un segment réseau passe dans une salle. »

La proposition redondante est la première (qui induirait l'association `branchement` de l'exemple 1-99). En effet, tout ordinateur étant situé dans une salle où passe un seul segment, il n'est pas nécesaire de relier l'ordinateur à son segment, car il suffit de connaître la salle pour en déduire cette information.

L'explication plus formelle est donnée par les dépendances fonctionnelles. Le schéma 1-99 n'est pas en troisième forme normale en effet la dépendance fonctionnelle `ordinateur` $\rightarrow$ `segment` n'est pas directe car elle est déduite des deux autres (`ordinateur` $\rightarrow$ `salle` puis `salle` $\rightarrow$ `segment`).

Figure 1-99 *Exemple de transitivité*



Propositions réductibles

Étudions un autre exemple avec la proposition « Un client réalise des achats de produits chez des fournisseurs. ». Décomposons de manière intuitive de sorte à extraire des propositions plus élémentaires.

- « Un achat est effectué par un client. »
- « Un achat concerne plusieurs produits. »
- « Un achat s'effectue chez un fournisseur. »

On peut en déduire que `Client`, `Produit` et `Fournisseur` seront des entités (classes). `Achat` sera l'association (classe-association) reliant `Fournisseur` au couple (`Client,Produit`).

Propositions complexes irréductibles

En considérant à nouveau la proposition « Un logiciel est installé par un département sur un serveur. » et en supposant que les réponses soient « oui » à chacune des propositions binaires, alors l'association représentant l'installation sera ternaire (sans contrainte).

De même, concernant l'exemple précédent, si toutes les propositions suivantes sont vraies, aucune décomposition ne sera possible et l'association représentant l'achat sera ternaire (sans contrainte).

- « Un client peut acheter tout produit. »
- « Un fournisseur dispose de tout produit. »
- « Un client peut se servir chez tout fournisseur. »

Chronologie des étapes

En théorie, la construction d'un schéma conceptuel suit les étapes suivantes :

- Recensement des entités (classe).

- Disposition des attributs dans chaque entité (classe).
- Détermination d'un identifiant par entité (classe).
- Recensement des associations et définition du degré : binaire, classe-association ou *n*-aire.
- Disposition d'éventuels attributs dans chaque classe-association.
- Mise en place des contraintes.
- Vérification par normalisation.

Attention à ne pas surestimer le degré d'une association.
Ne pas réfléchir en terme de fonctionnement ou en termes de traitement, mais simplement exprimer des faits.

Bilan

Alors que les aspects statiques du niveau conceptuel (entité, classe, association, classe-association, héritage) se traduisent naturellement sous SQL2 ou SQL3, les contraintes nécessitent un effort de programmation le plus souvent sous la forme de contraintes SQL de vérification `CHECK` ou de déclencheurs. L'aspect dynamique du niveau conceptuel n'est pas natif aux bases de données et il restera à programmer l'encapsulation par les méthodes.

Les méthodes et outils de conception objet qui permettent de transiter du conceptuel au physique (spécifications UML traduites automatiquement en classes C++ ou Java) sont bien plus adaptés à l'élaboration de programmes que de schémas de bases de données. Les principaux inconvénients de ces démarche sont les suivants :

- le souci d'intégrité des objets stockés n'est pas suffisamment pris en compte ;
- la traduction des associations au niveau physique n'est ni clairement décrite, ni totalement maîtrisée.

UML 2 ou Merise/2 ?

Après avoir examiné différents formalismes : le modèle de Chen, le MCD de Merise/2 et la notation UML, on est en droit de savoir quel formalisme convient le plus à la modélisation d'une base de données. La notation UML 2 est bien adaptée à la modélisation d'une base de données (et si elle est utilisée à bon escient !) pour les raisons suivantes :

- les bases de données sont majoritairement relationnelles, mais certaines migreront vers l'objet ;
- les concepts fondamentaux de Merise/2 peuvent être représentés dans le diagramme de classes d'UML qui offre en plus la possibilité de définir des stéréotypes et des contraintes personnalisées ;

- la majorité des contraintes sur les associations *n*-aires sont implicites si on utilise les classes-associations ;
- les concepteurs travaillent dans un même environnement (comme Eclipse), avec la possibilité d'interfacer plus facilement les langages évolués C++ ou Java.

Bien qu'il reste encore des fonctionnalités absentes des outils utilisant la notation UML, notamment en ce qui concerne les contraintes et des associations *n*-aires, les versions à venir de ces produits devraient être de plus en plus performantes.

Cette édition fait encore référence au modèle entité-association dans les prochains chapitres pour que les inconditionnels y trouvent leur compte.

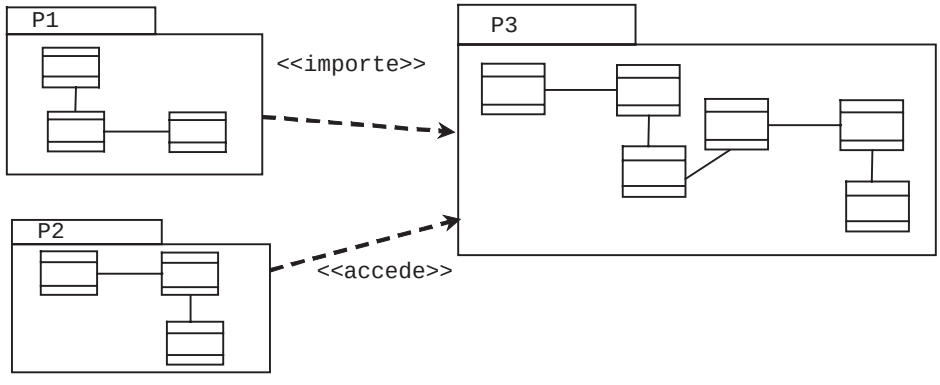
Quelques règles à respecter avec UML

La spécification UML 2 n'est pas très restrictive: *« Lines may be drawn using various styles, including orthogonal segments, oblique segments, and curved segments. The choice of a particular set of line styles is a user choice. Generalizations between associations are best drawn using a different color or line width than what is used for the associations. »*

L'adoption des règles suivantes vous permettra de construire des diagrammes de classes UML les plus lisibles et évolutifs possible :

- Évitez de croiser les liens d'association (si c'est toutefois inévitable, utilisez la notation du « pont » électrique).
- Tracez les liens d'association horizontalement ou verticalement.
- Concernant les associations *un-à-plusieurs*, essayez de placer la classe qui contient une multiplicité *en haut ou à gauche de la classe qui contient une multiplicité 1.
- N'hésitez pas à construire des classes grandes et fines ou, au contraire, des classes petites et larges afin d'éviter les croisements de liens d'association.

Figure 1-100 *Utilisation de paquetages*



- Utilisez la notion de paquetages pour former des sous-schémas (regroupement de classes reliées entre elles) dans le cas de diagrammes volumineux. L'exemple 1-100 illustre deux relations de dépendance entre paquetages. Le paquetage source est dit client, le paquetage cible est dit fournisseur. Le stéréotype `<<importe>>` permet d'ajouter des éléments du paquetage fournisseur au paquetage client. Le stéréotype `<<accede>>` permet de référencer des éléments du paquetage fournisseur.

Et après ?

Il est utile d'établir plusieurs niveaux de conception, car ils permettront de séparer les spécifications formelles de l'implémentation et rendront les schémas indépendants des matériels et des logiciels, dans la mesure du possible. Nous énonçons au chapitre suivant les règles d'évolution pour définir un modèle de données (niveau logique) à partir d'un diagramme conceptuel.

Exercices

Composer les schémas conceptuels de type Merise/2 ou UML 2 afin de modéliser les faits suivants. Pour faciliter les corrections qui se trouvent sur le site d'accompagnement, utilisez les noms d'attributs qui sont indiqués en police `courrier`. Les exercices sont de difficulté croissante.

Exercice 1.1 Inscription des étudiants

Lors de son inscription en début d'année scolaire, chaque étudiant remplit une fiche sur laquelle il indique certains renseignements comme son numéro d'identification nationale (`ninsee`), ses nom et prénom (`nom`, `prenom`), son adresse (`adresse`) et la liste des unités de valeurs (UV) qu'il s'engage à suivre (8 au plus sur les 15 possibles). Un code lui est automatiquement attribué (`codetu`).

Une UV est caractérisée par un code (`codeuv`) et un intitulé (`intuv`). Par exemple le code *UV3* identifie *Électronique numérique*. Chaque UV est placée sous la responsabilité d'un enseignant identifié par ses initiales (`initens`) et caractérisé par un nom (`nomens`), un numéro de bureau (`bureauens`) et un numéro de téléphone (`telens`). Cet enseignant se rend disponible un jour de la semaine (`jsem`) et durant une plage horaire précise (`hrens`) afin de donner tout renseignement concernant les UV qu'il dirige.

- Que faut-il modifier pour qu'un enseignant se rende disponible à différents moments (un seul créneau par UV qu'il dirige) ?
- Que faut-il modifier pour que différents créneaux soient disponibles par UV qu'il dirige ?

Exercice 1.2 Gestion des stages

Une école de journalistes veut organiser des stages à l'intention des étudiants en dernière année (`codetu`, `nom`, `prenom`, `ninsee`). Elle fait appel à des entreprises (`codent`, `adressent`) qui proposent des stages. Les stages sont caractérisés par un code stage, une durée, une date de début et de fin (`nstage`, `duree`, `ddeb`, `dfin`).

Chaque stage est consacré à un thème d'étude (`nomtheme`) répertorié par un code (`codetheme`). Exemple : le code *eco1* identifie le thème *économie générale*. Les étudiants indiquent les stages qu'ils souhaitent suivre en priorité en leur attribuant un numéro de préférence (`npriorite`). Le chef d'établissement décide ensuite des attributions. Chaque stagiaire est placé sous la responsabilité d'un enseignant (utiliser l'entité `Enseignant` du précédent exercice).

On désire obtenir des statistiques selon le thème du stage effectué par les étudiants. On désire stocker les diplômes (`codedip`, `nomdip`) qu'ont obtenus les étudiants (bacs et diplômes de premier cycle). Les éventuelles mentions aux diplômes (`mentiondip`) doivent aussi être stockées.

Précisez la nature de chaque stage par un pourcentage associé à chaque thème (par exemple, le stage *c1* est composé de 20 % de *eco1* et 80 % de *maketing2*).

On désire connaître les antécédents des étudiants avant l'obtention de leur diplôme final. En particulier, il sera utile de connaître pour chaque étudiant le nombre d'années passées (`nbanpasse`) dans les différents établissements fréquentés pour avoir un diplôme (utiliser l'entité `Etablissement` avec `codetab`, `nometab`, `nomdirecteur`).

Exercice 1.3 Affrètement d'avions

Un aéroport toulousain désire gérer les compagnies, leurs avions et les vols affrétés. Une compagnie est caractérisée par un code et un nom (`comp`, `nomComp`). Chaque avion est désigné par une immatriculation (`immatriculation`), un type (`typeAvion`), une capacité (`capacite`). Un avion est la propriété d'une compagnie.

Un avion peut être affrété par une compagnie à différentes dates (`dateVol`), même plusieurs fois par jour par différentes compagnies. Pour chaque affrètement il faudra stocker le nombre de passagers transportés (`nbPax`) et le coût du vol pour la compagnie (`cout`). On ne pose pas de contrainte, donc, chaque compagnie peut affréter n'importe quel avion à n'importe quel moment. On suppose toutefois qu'une compagnie ne peut pas affréter le même avion plusieurs fois dans la même journée.

L'aéroport décide de stocker les caractéristiques de chaque type d'avion : le code de la désignation commerciale, le nombre maximum de passagers (`npMax`) et la désignation commerciale (`nomAvion`). Exemple : l'A320 peut transporter au maximum 180 passagers et se dénomme « Airbus A320 ».

- Comment décrire la contrainte UML 2 qui interdit à une compagnie d'affréter des avions qui ne lui appartiennent pas ?
- Décrire la modification à faire pour qu'une compagnie puisse affréter un avion plusieurs fois dans la même journée.

Exercice 1.4 Gestion des partiels

Compléter le schéma de la scolarité (exercice 1.1) en tenant compte de nouvelles spécifications :

En cours d'année, dans le cadre du contrôle continu des connaissances, chaque étudiant subit deux partiels pour chaque UV qu'il a choisie. En fin d'année il passe un examen obligatoire. Pour chaque épreuve (partiel ou examen) on désire mémoriser :

- un numéro chronologique qui identifie chaque épreuve quelle que soit la matière (`ncont`) ;
- la date (`datecont`), l'heure de début (`debcont`) et sa durée (`dureecont`) ;

- le type de l'épreuve (`typecont`).

Après correction des copies, il est affecté une note à chaque étudiant (`notetu`). Une UV est décernée sous réserve d'une moyenne supérieure à 10.

Les épreuves sont surveillées par un ou plusieurs enseignants et peuvent se dérouler simultanément dans différentes salles. Les salles sont caractérisées par un code (`codesal`) et un nombre de places (`capacité`). Les enseignants sont caractérisés par un code, un nom et un grade. Chaque enseignant rédige un rapport concernant sa surveillance sous la forme d'une note de quelques lignes (`rapport`). Un enseignant peut passer d'une salle à l'autre durant un contrôle. On désire connaître le temps de présence (`tempspresence`) passé dans chaque salle au total.

Exercice 1.5 Castanet Télécoms

Castanet Télécoms désire utiliser une base de données afin de gérer les factures de ses abonnés. Il est nécessaire de pouvoir stocker dans la base de données tous les éléments relatifs au calcul d'une facture. On suppose que les différentes consommations sont automatiquement importées d'un autocommutateur (par exemple, une communication d'une heure trente sera automatiquement transformée en 1,5 dans un champ numérique de la base de données).

On ne souhaite pas conserver l'historique de l'année en cours, mais seulement les consommations courantes de périodes de deux mois. Tous les abonnés de Castanet Télécoms reçoivent une facture tous les deux mois. Il est nécessaire de connaître les différentes consommations (locales, nationales, Internet, vers mobiles et autres appels). Les coordonnées (nom et adresse) d'un abonné devront être connues (pour lui envoyer ses factures). L'abonné peut choisir le débit de sa ligne pour chaque période de deux mois (1, 2, 8 ou 20 Mo/s) – bien sûr, le montant de la facture sera modifié en conséquence.

Les abonnés peuvent souscrire un contrat pour Internet. Plusieurs formules de contrat sont possibles :

- la formule 'F1', qui coûte 45 euros par mois avec 3 heures de connexion comprises, l'heure supplémentaire étant taxée 19 euros ;
- la formule 'F2', qui coûte 95 euros par mois avec un nombre d'heures de connexion illimité ;
- d'autres formules toutes basées sur un montant par mois.

Castanet Télécoms fait appel à des fournisseurs de services Internet pour prendre en charge ces formules. Ainsi, il est possible qu'une même formule soit proposée par différents prestataires et que des abonnés soient connectés à différents fournisseurs de services pour la même formule de contrat. Castanet Télécoms désire conserver les dates de premier contact et de fin de contrat avec chaque prestataire Internet pour les différentes formules qu'il a proposées. Il est aussi nécessaire de conserver l'URL, l'adresse et le nom du responsable pour tous les prestataires Internet. Un abonné peut également opter pour un téléphone portable. Une fois le contrat signé, un numéro de portable est attribué au client et un combiné portable lui est donné. Le type du combiné peut être basique, moderne ou de luxe. L'abonné peut choisir trois numéros privilégiés pour lesquels une réduction importante sera calculée en fonction des durées d'appel, lors de l'édition de la facture.

Les attributs à prendre en compte sont les suivants : `numAbonne`, `nomAbonne`, `adresseAbonne`, `numPortable`, `typeCombine`, `telPrefere`, `dureePreferee`, `numTel`, `consoLocale`, `consoNationale`, `consoInternet`, `consoAutres`, `debitLigne`, `consoMobiles`, `formule`, `prixParMois`, `heureSupp`, `URL`, `adresseFourni`, `responsable`, `premierContact`, `finContrat`.

Exercice 1.6 Voltige aérienne

Compétition journalière

Les organisateurs du championnat du monde de voltige aérienne décident de concevoir une base de données relationnelle afin de gérer la compétition. Celle-ci se déroulera sur période de 9 jours, elle rassemblera 25 avions et 30 compétiteurs. Chaque voltigeur pourra utiliser au plus trois avions différents lors de la compétition (même des avions qui n'appartiennent pas à son club de voltige). Pour cela, il devra notifier aux juges avant le début de la compétition la liste des avions qu'il envisage d'utiliser. Il ne sera donc pas possible pour un voltigeur de voler sur n'importe quel avion. On suppose qu'un pilote ne vole qu'une fois au plus par jour.

Chaque vol de compétition comprend 7 figures qui seront notées distinctement par les juges. On suppose que l'on ne stocke pas la date des vols (la base est vidée chaque matin). Il faudra stocker en plus les éléments suivants :

- numéro, nom et aérodrome de base de chaque club de voltige (exemples : LFBO pour Blagnac, LFCL pour Lasbordes, LFCI pour Albi…) ;
- immatriculation, type des avions et rattachement au club propriétaire ;
- identité du voltigeur (numéro et nom du compétiteur, pays et club de voltige) ;
- pour chaque figure de voltige référencée dans un ouvrage de référence (qui en comporte plus de 1 500), il faudra connaître son nom, sa cotation maximale (note qui varie de 5 à 10), la date de création et le numéro de la page du livre la décrivant ;
- les inventeurs des figures doivent être également mémorisés avec leur nom et le type de personnage (militaire, pilote civil ou autre).

Les attributs sont les suivants : `nclubVoltige`, `aerodromeBase`, `nomClub`, `immatriculation`, `typeAvion`, `ncomp`, `nomComp`, `pays`, `nfigure`, `nomFigure`, `noteMax`, `dateCreation`, `pageManuel`, `ninventeur`, `nomInventeur`, `typePilote`, `note`.

Historique des compétitions

Il est nécessaire maintenant de stocker les résultats pour chaque jour de la compétition. On suppose toujours qu'un pilote ne vole qu'une fois au plus par jour.

Chaque vol de compétition est soit un programme connu, un des deux programmes inconnus ou un libre intégral (où les compétiteurs peuvent faire 12 figures). Lors du libre intégral, les juges doivent avoir reçu de la part du compétiteur avant le vol la liste des figures avec leur chronologie.

Les attributs à ajouter sont les suivants : `numeroChrono`, `dateVol` et `typeProgramme`.

Chapitre 2

Le niveau logique : du relationnel à l'objet

Nous ne cesserons pas d'explorer -
L'aboutissement de toutes nos quêtes -
Sera d'atteindre l'endroit d'où nous étions partis -
Et pour la première fois de le reconnaître.

The Magus, J. Fowles, Jonathan Cape, 1967

Ce chapitre se divise en trois parties. La première décrit le modèle relationnel et traite de la normalisation. La deuxième décrit les concepts objet présents au niveau logique. La troisième, enfin, expose les règles permettant de dériver un schéma relationnel ou objet à partir d'un diagramme entité-association ou UML.

Modèle relationnel

Bien que tous les outils permettent de modéliser un schéma logique relationnel, aucun ne peut offrir un processus de normalisation automatique. Il est donc nécessaire de maîtriser à la fois le modèle de données et les principes de normalisation afin de concevoir des bases de données cohérentes.

Historique, généralités

Le modèle relationnel a vu le jour dans les années 1970 avec les travaux de E.F. Codd [COD 69, 70]. Il est fondé sur de solides bases théoriques, car il propose des opérateurs issus de la théorie des ensembles. Par ailleurs, le processus de normalisation permet de construire des bases de données sans redondance d'informations, tout en préservant l'intégrité des données lors d'ajouts, de modifications ou de suppressions d'enregistrements.