Modélisation objet avec UML

Pierre-Alain Muller
Nathalie Gaertner

Deuxième édition 2000
Cinquième tirage 2004

ISBN : 2-212-11397-8
EYROLLES

3

La notation UML

La notation UML est une fusion des notations de Booch, OMT, OOSE et d'autres notations. UML est conçue pour être lisible sur des supports très variés comme les tableaux blancs, les feuilles de papier, les nappes de restaurants, les écrans d'ordinateurs, etc. Les concepteurs de la notation ont recherché avant tout la simplicité ; UML est intuitif, homogène et cohérent. Les symboles embrouillés, redondants ou superflus ont été éliminés en faveur d'un meilleur rendu visuel.

UML se concentre sur la description des artefacts de modélisation logicielle, plutôt que sur la formalisation du processus de développement lui-même : elle peut ainsi être utilisée pour décrire les éléments logiciels, obtenus par l'application de différents processus de développement. UML n'est pas une notation fermée : elle est générique, extensible et configurable par l'utilisateur. UML ne recherche pas la spécification à outrance : il n'y a pas une représentation graphique pour tous les concepts imaginables ; en cas de besoin, des précisions peuvent être apportées au moyen de mécanismes d'extension et de commentaires textuels. Une grande liberté est donnée aux outils pour le filtrage et la visualisation d'information. L'usage de couleurs, de dessins et d'attributs graphiques particuliers est laissé à la discrétion de l'utilisateur.

Ce chapitre propose un survol de la sémantique et de la notation des éléments de modélisation d'UML, décrits de manière précise dans le document de spécification d'UML[1].

[1] Object Management Group. *Unified Modeling Language Specification V 1.3*, juin 1999.

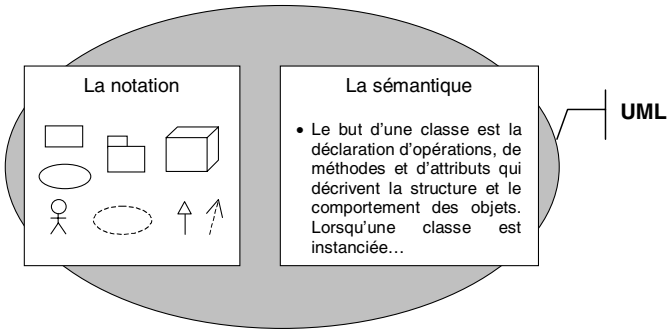


Figure 3–1.
UML définit une notation et une sémantique.

Ce chapitre a pour objectif d'introduire les principaux concepts de modélisation et de montrer leur articulation au sein de la notation UML. Les éléments de visualisation et les éléments de modélisation sont présentés conjointement, en se servant de la notation comme d'un support pour faciliter la présentation de la sémantique. UML définit neuf sortes de diagrammes pour représenter les différents points de vue de modélisation. L'ordre de présentation de ces différents diagrammes ne reflète pas un ordre de mise en œuvre dans un projet réel, mais simplement une démarche pédagogique qui essaie de minimiser les prérequis et les références croisées.

Les diagrammes d'UML

Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. UML définit des diagrammes structurels et comportementaux pour représenter respectivement des vues statiques et dynamiques d'un système.

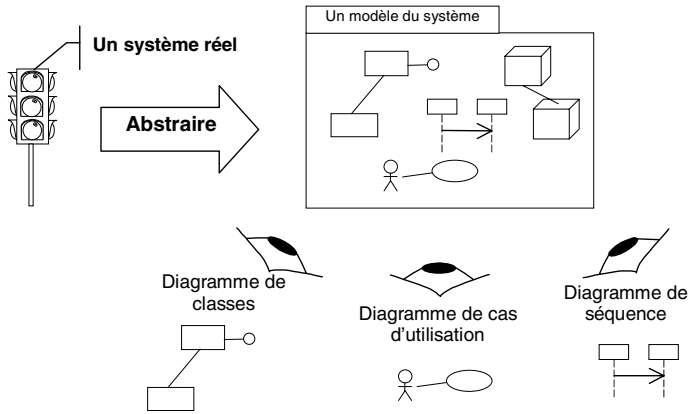
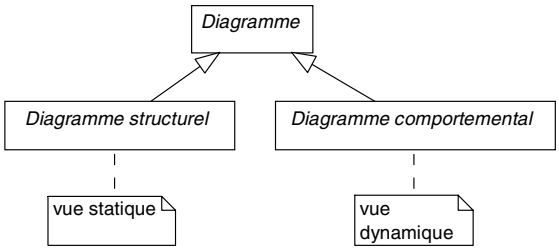


Figure 3–2.
Les diagrammes offrent différentes vues sur le modèle.

Figure 3–3.
UML propose des diagrammes structurels et comportementaux.



Au total UML définit neuf types de diagrammes, quatre structurels et cinq comportementaux.

Figure 3–4.
Les quatre types de diagrammes structurels.

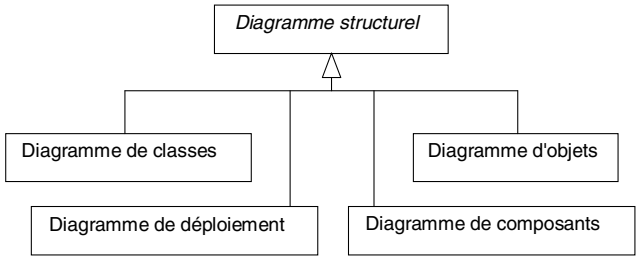
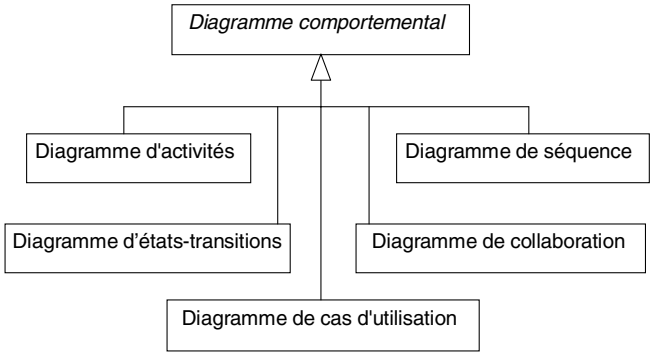


Figure 3–5.
Les cinq types de diagrammes comportementaux.



Un diagramme contient des attributs de placement et de rendu visuel qui ne dépendent que du point de vue. La plupart des diagrammes se présentent sous la forme de graphes, composés de sommets et d'arcs.

Les diagrammes contiennent des éléments de visualisation qui représentent des éléments de modélisation. Un élément peut apparaître dans différents diagrammes pour modéliser une même abstraction selon différents points de vue.

De plus, plusieurs éléments, éventuellement issus de paquetages distincts, peuvent être représentés dans un diagramme donné, même en l'absence de relations de visibilité entre ces paquetages[2].

Les diagrammes peuvent montrer tout ou partie des caractéristiques des éléments de modélisation, selon le niveau de détail utile dans le contexte d'un diagramme donné.

Par exemple, au lieu de modéliser les différents types de diagrammes d'UML à l'aide des trois figures précédentes, il est possible de se satisfaire du premier diagramme très général ou faire la fusion des trois.

En général, un diagramme représente un aspect ou un point de vue particulier. Les diagrammes peuvent également rassembler des informations liées entre elles, pour montrer par exemple les caractéristiques héritées par une classe.

Il est important de bien choisir les diagrammes, les abstractions et les niveaux de détails représentés par ces diagrammes pour modéliser au mieux le système, ses caractéristiques et pour mettre en évidence les points essentiels et délicats à prendre en compte.

Voici, la finalité des différents diagrammes d'UML :

- *les diagrammes d'activités* représentent le comportement d'une méthode ou d'un cas d'utilisation, ou un processus métier ;
- *les diagrammes de cas d'utilisation* représentent les fonctions du système du point de vue des utilisateurs ;
- *les diagrammes de classes* représentent la structure statique en termes de classes et de relations ;
- *les diagrammes de collaboration* sont une représentation spatiale des objets, des liens et des interactions ;
- *les diagrammes de composants* représentent les composants physiques d'une application ;

[2] Car l'utilisateur peut tout voir !

- *les diagrammes de déploiement* représentent le déploiement des composants sur les dispositifs matériels ;
- *les diagrammes d'états-transitions* représentent le comportement d'un classificateur ou d'une méthode en terme d'états ;
- *les diagrammes d'objets* représentent les objets et leurs liens et correspondent à des diagrammes de collaboration simplifiés, sans représentation des envois de message ;
- *les diagrammes de séquence* sont une représentation temporelle des objets et de leurs interactions.

Les diagrammes de collaboration et les diagrammes de séquence sont appelés diagrammes d'interaction. Les diagrammes d'états-transitions sont également appelés *Statecharts*[3] (nom donné par leur auteur, David Harel).

Des diagrammes personnels peuvent être définis si nécessaire ; toutefois, il est probable que les neufs types de diagrammes standard suffisent à la modélisation des différents aspects de la majorité des systèmes.

Concepts de base

Il est pratique de représenter la sémantique des éléments de modélisation d'UML selon le formalisme d'UML.

Ce type de représentation récursive pose cependant le problème *de l'œuf et de la poule*, principalement lors de l'apprentissage. C'est pourquoi il est conseillé, en première lecture, de considérer les diagrammes et les éléments des diagrammes comme des exemples de la notation, plutôt que de chercher à approfondir leur compréhension.

Quoi qu'il en soit, les diagrammes montrent des vues simplifiées du métamodèle afin de rendre le texte accessible au plus grand nombre de lecteurs.

Les éléments communs

Les éléments sont les briques de base du métamodèle UML.

Ces éléments définissent la sémantique des modèles qui vont être créés ; ils se placent ainsi dans le niveau M2 de l'architecture de métamodélisation de l'OMG.

---

[3] Harel, D. *Statecharts : A Visual Formalism for Complex Systems*. Science of Computer Programming, vol. 8, 1987.

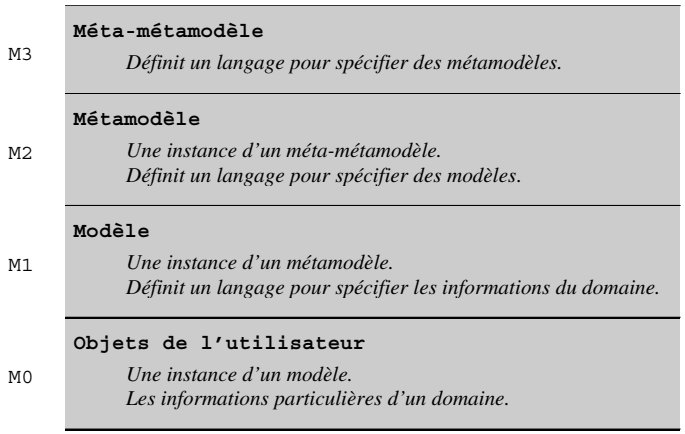


Figure 3–6. *Architecture de métamodélisation de l'OMG, composée de quatre couches.*

Un exemple d'utilisation de cette architecture en quatre couches peut être le suivant :

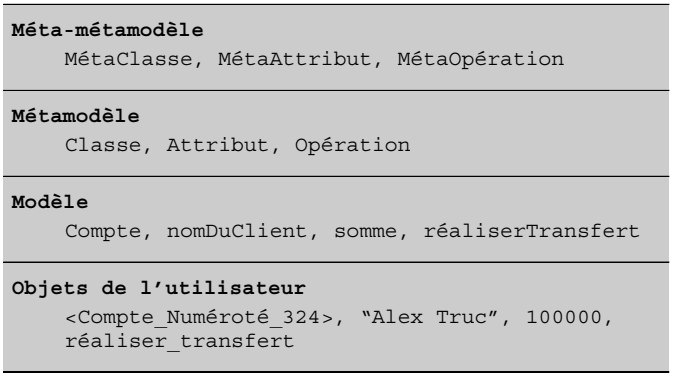


Figure 3–7. *Exemple d'utilisation de l'architecture de métamodélisation de l'OMG.*

Les éléments d'UML comprennent les *éléments de modélisation* et *les éléments de visualisation* (par exemple, la notion de classe et l'icone associée). Un modèle donné contient des *instances* d'éléments de modélisation, pour représenter les abstractions du système en cours de modélisation (par exemple, une classe **`Compte_Numéroté_324`**). Ainsi, pour chaque modèle décrit dans la suite de l'ouvrage, il faudrait systématiquement préciser le terme *instance de* suivi du nom d'un élément de modélisation. Les phrases de description deviendraient longues et difficiles à comprendre. Par souci de clarté, la désignation explicite concernant l'instanciation d'un élément de modélisation ne sera pas effectuée mais sera sous-entendue. Les éléments de modélisation, ou plus précisément les instances d'éléments de modélisation, sont regroupés en paquetages. Un modèle est une abstraction d'un système, représenté par une arborescence de paquetages.

Il est développé en fonction de besoins ou d'objectifs et représente uniquement les aspects pertinents du système au regard de ces objectifs. Un élément de visualisation procure une projection textuelle ou graphique d'un ensemble d'éléments de modélisation. L'ensemble des éléments de visualisation (ou de représentation) forme la notation, qui permet la manipulation des éléments de modélisation.

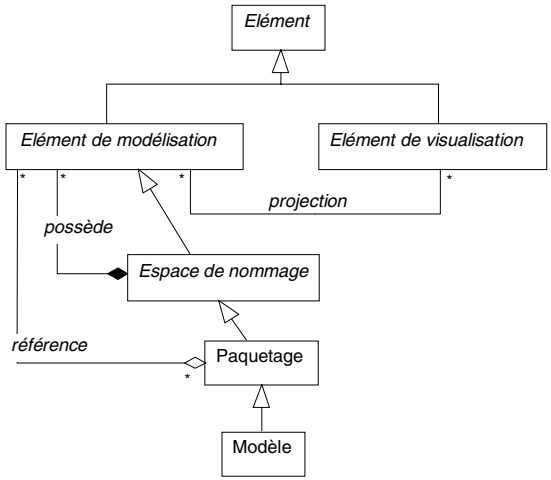


Figure 3–8.
Extrait du métamodèle. Représentation des deux grandes familles d'éléments qui forment le contenu des modèles.

Les mécanismes communs

UML définit un petit nombre de mécanismes communs qui assurent l'intégrité conceptuelle de la notation. Ces mécanismes communs comprennent les commentaires, les mécanismes d'extension (stéréotypes, contraintes et valeurs marquées), la relation de dépendance et les dichotomies **`(type, instance)`** et **`(type, classe)`**. Chaque élément de modélisation possède une spécification qui contient la description unique et détaillée de toutes ses caractéristiques.

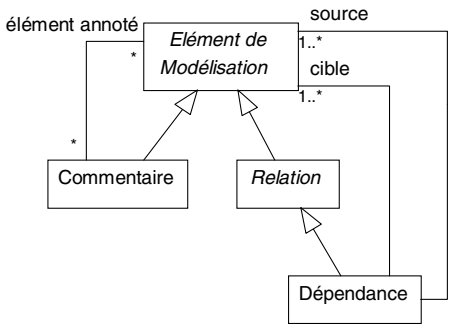


Figure 3–9.
Extrait du métamodèle. Représentation des commentaires et des dépendances.

Les stéréotypes, les valeurs marquées et les contraintes permettent l'extension et la spécialisation d'UML. Les stéréotypes spécialisent les classes du métamodèle, les valeurs marquées étendent les attributs des classes du métamodèle et les contraintes sont des relations sémantiques entre éléments de modélisation qui définissent des conditions que doit vérifier le système.

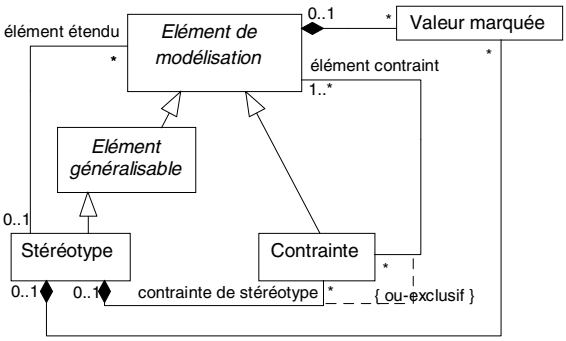


Figure 3–10. *Extrait du métamodèle. Représentation des mécanismes d'extension.*

Les stéréotypes

Le concept de stéréotype est le mécanisme d'extensibilité le plus puissant d'UML. Un stéréotype introduit une nouvelle classe dans le métamodèle par dérivation d'une classe existante (le nom d'un stéréotype nouvellement défini ne doit pas être déjà attribué à un autre élément du métamodèle). Un stéréotype permet la métaclassification d'un élément d'UML. Les utilisateurs (méthodologistes, constructeurs d'outils, analystes et concepteurs) peuvent ainsi ajouter de nouvelles classes d'éléments de modélisation au métamodèle, en plus du noyau prédéfini par UML.

Un stéréotype est utilisé pour définir une utilisation particulière d'éléments de modélisation existants ou pour modifier la signification d'un élément. L'élément stéréotypé et son parent non stéréotypé ont une structure identique mais une sémantique différente. Ainsi, les instances d'un élément **`E`** stéréotypé ont les mêmes propriétés (attributs, opérations, associations) que les instances de l'élément **`E`** ; elles peuvent toutefois avoir des propriétés supplémentaires via des contraintes ou des valeurs marquées, associées à leur stéréotype. Chaque élément stéréotypé peut s'appuyer sur d'autres stéréotypes, essentiellement par le biais du mécanisme de généralisation entre stéréotypes.

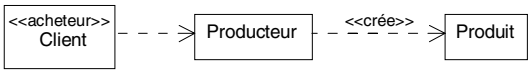
Les stéréotypes permettent aux utilisateurs d'UML d'opérer l'extension contrôlée des classes du métamodèle. Un élément spécialisé par un stéréotype **`S`** est sémantiquement équivalent à une nouvelle classe du métamodèle, nommée elle aussi **`S`**. À l'extrême, toute la notation aurait pu être construite à partir des

deux classes **`Truc`** et **`Stéréotype`**, les autres concepts en dérivant par stéréotypage de la classe **`Truc`**.

Les concepteurs d'UML ont recherché l'équilibre entre les classes incluses d'origine et les extensions apportées par les stéréotypes. Ainsi, seuls les concepts fondamentaux sont exprimés sous la forme de classes distinctes. Les autres concepts, dérivables de ces concepts de base, sont traités comme des stéréotypes. Les stéréotypes prédéfinis dans le standard UML sont décrits en annexe A1.

Le nom du stéréotype est placé entre guillemets avant le nom de l'élément auquel il s'applique.

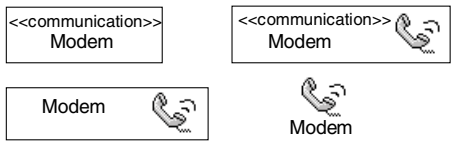
Figure 3–11.
Représentation d'une classe et d'une dépendance stéréotypées.



Il est à noter qu'UML fait la distinction entre stéréotype et mot clé bien que les deux concepts se basent sur un nom placé entre guillemets. Le stéréotype définit une nouvelle métaclasse et possède donc un nom qui n'est pas déjà utilisé dans le métamodèle UML, alors que le mot clé référence une classe du métamodèle.

Une icone peut être associée à un stéréotype pour être utilisée à la place ou avec le nom du stéréotype (aucune icone n'est prédéfinie par UML). Une distinction visuelle est ainsi possible.

Figure 3–12.
Différentes représentations pour un élément de modélisation stéréotypé.



Les stéréotypes sont à utiliser avec modération pour que les concepteurs ne soient pas les seuls à comprendre les modèles et que lesdits modèles soient compatibles avec les différents outils de modélisation.

Les valeurs marquées

Une valeur marquée est une paire **`(nom, valeur)`** qui ajoute une nouvelle propriété à un élément de modélisation. Cette propriété peut représenter une information d'administration (auteur, date de modification, état, version…), de génération de code ou une information sémantique utilisée par un stéréotype. En général, la propriété ajoutée est une information annexe, utilisée par les concepteurs ou les outils de modélisation. La spécification d'une valeur marquée prend la forme : **`nom = valeur`**. Une valeur marquée est indiquée entre accolades. Il est possible d'utiliser des valeurs marquées prédéfinies (voir annexe A1) ou personnalisées (par exemple, version ou auteur).

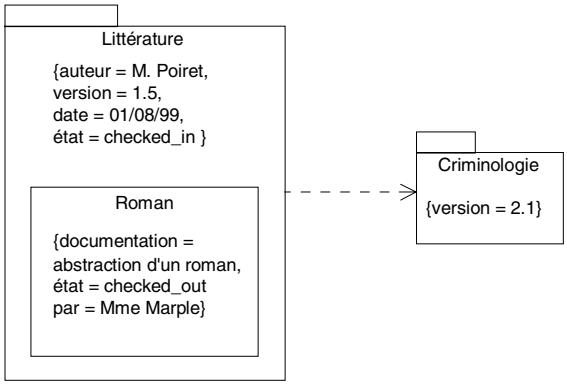


Figure 3–13.
Utilisation de valeurs marquées personnelles (auteur, version, date, état et par) et standard (documentation).

Pour les propriétés de type booléen, la valeur par défaut est vrai et il n'est pas nécessaire de l'indiquer.

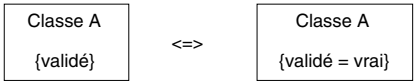


Figure 3–14.
Les valeurs marquées booléennes ont une valeur vrai par défaut.

Les contraintes

Une contrainte est une relation sémantique quelconque entre éléments de modélisation qui définit des propositions devant être maintenues à vraies pour garantir la validité du système modélisé. UML permet d'exprimer des contraintes de manière informelle, en langage naturel ou en pseudo-code. Pour exprimer des contraintes de manière formelle, le langage de contraintes OCL (Object Constraint Language) peut être utilisé. OCL est décrit plus en détail dans une section ultérieure. Une contrainte est définie par une expression booléenne prédéfinie (voir annexe A1) ou personnalisée. Chaque contrainte est indiquée entre accolades et placée près de l'élément (stéréotypé ou non) auquel elle est associée. Une contrainte peut être associée à plusieurs éléments par des relations de dépendance.

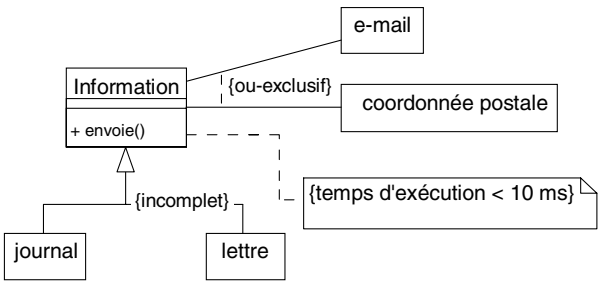


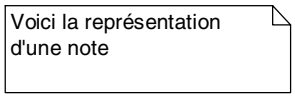
Figure 3–15.
Représentation de contraintes.

Une expression entre accolades placée dans une note (voir section suivante) est une contrainte attachée aux éléments connectés à la note. Par exemple, **{ temps d'exécution < 10ms }** est une contrainte attachée à l'opération **envoie**.

Les notes

Une note est un symbole graphique qui contient des informations. En général, ces informations sont sous forme textuelle.

Figure 3–16.
Représentation d'une note.



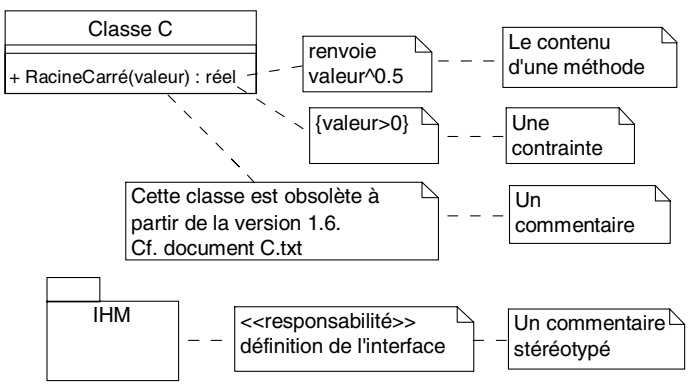
L'utilisation d'une note permet de présenter des commentaires, des contraintes, le contenu de méthodes ou des valeurs marquées.

Un commentaire fournit des explications utiles, des observations de diverses natures ou des renvois vers des documents de description plus complets. Par défaut, ils ne véhiculent pas de contenu sémantique.

Les deux stéréotypes prédéfinis **<<besoin>>** et **<<responsabilité>>** s'appliquent aux commentaires. Un besoin spécifie un comportement désiré ou des caractéristiques souhaitées pour un élément alors qu'une responsabilité spécifie un contrat ou une obligation d'un élément lors de son association avec d'autres éléments.

Chaque note est reliée à un ou plusieurs éléments de modélisation par une ligne en traits pointillés.

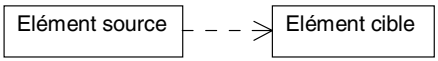
Figure 3–17.
Représentation de différentes notes reliées à des éléments de modélisation.



La relation de dépendance

La relation de dépendance définit une relation d'utilisation unidirectionnelle entre deux éléments de modélisation, appelés respectivement source et cible de la relation. Elle indique une situation dans laquelle un changement au niveau de la cible implique un changement au niveau de la source. Une relation de dépendance est représentée par une ligne en traits pointillés avec une flèche du côté de l'élément cible.

Figure 3–18.
Représentation d'une dépendance.



Les notes et les contraintes peuvent également être l'élément source d'une relation de dépendance. Chaque relation de dépendance peut être stéréotypée si désiré avec des stéréotypes standard (tels que **<<instance de>>**, **<<importe>>**, **<<liaison>>** ou **<<étend>>**, par exemple) ou personnalisés.

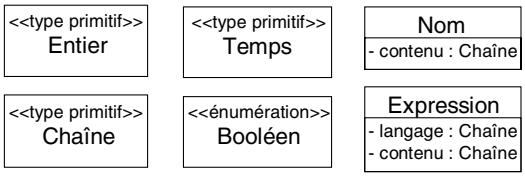
Dichotomies `(type, instance)` et `(type, classe)`

De nombreux éléments de modélisation présentent une dichotomie **`(type, instance)`**, dans laquelle le type dénote l'essence de l'élément, et l'instance avec ses valeurs correspond à une manifestation de ce type. De même, la dichotomie **`(type, classe)`** correspond à la séparation entre la spécification d'un élément qui est énoncé par le type et la réalisation de cette spécification qui est fournie par la classe.

Les types primitifs

Le standard UML définit un certain nombre de types primitifs, utilisés pour la modélisation d'UML dans le métamodèle. La figure suivante présente un extrait du métamodèle, en particulier certains de ces types.

Figure 3–19.
Extrait du métamodèle. Représentation de types primitifs du métamodèle UML.



Les types suivants sont des types primitifs du métamodèle UML :

- *Booléen.* Un booléen est un type énuméré qui comprend les deux valeurs **`vrai`** et **`faux`**.
- *Entier.* Dans le métamodèle, un entier est un élément compris dans l'ensemble infini des entiers {…, -2, -1, 0, 1, 2, …}.

- *Expression.* Une expression est une chaîne de caractères qui définit une déclaration à évaluer. Le nom du langage d'interprétation peut être spécifié (OCL par exemple) ou omis. S'il est omis, l'interprétation est hors de la portée d'UML.
- *Nom.* Un nom est une chaîne de caractères qui permet de désigner un élément. Des noms composés peuvent être formés à partir de noms simples, selon la règle suivante :

```
nom_composé ::= nom_simple { '.' nom_simple }
```

  Le nom d'un élément peut être qualifié par le nom du paquetage qui le contient ou qui le référence, selon la syntaxe suivante :

```
nom_qualifié ::= qualificateur "::" nom_simple

qualificateur ::= nom_de_paquetage
                                  [ "::" qualificateur ]
```

- *Chaîne.* Une chaîne est une suite de caractères, désignée par un nom.
- *Temps.* Un temps est une valeur qui représente un temps absolu ou relatif. Un temps possède une représentation sous forme de chaîne. La syntaxe est hors de la portée d'UML.

Un utilisateur peut définir ses propres types primitifs (tels que des booléens, des réels, des chaînes de caractères) et des types énumérés particuliers. Les stéréotypes standard **<<type primitif>>** et **<<énumération>>** ainsi que des stéréotypes personnels sont appliqués aux classes pour spécifier des types particuliers. Les littéraux d'énumération sont spécifiés dans le compartiment des attributs. Des contraintes pour restreindre des plages de valeurs peuvent également être utilisées.

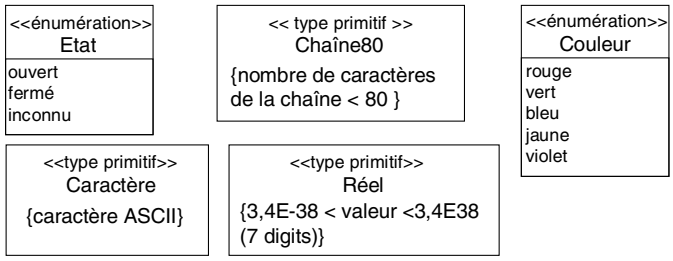
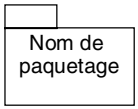


Figure 3–20.
Exemple de types primitifs définis par l'utilisateur.

Les paquetages

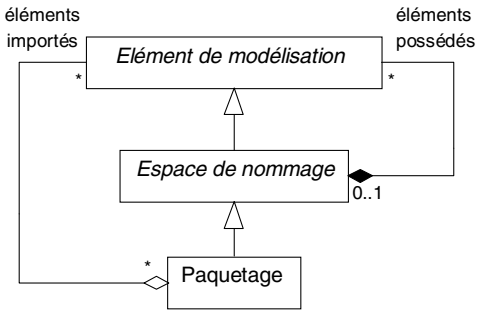
Les paquetages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de modélisation. Chaque paquetage est représenté graphiquement par un dossier.

Figure 3–21.
Représentation des paquetages.



Un paquetage *possède* des éléments de modélisation et peut également en *importer*.

Figure 3–22.
Extrait du métamodèle. Un paquetage est un élément de modélisation qui possède et importe d'autres éléments de modélisation.



La relation de possession est une composition entre les éléments de modélisation et leur paquetage associé ; elle implique la destruction de tous les éléments de modélisation contenus dans un paquetage lorsque ce dernier est détruit. De plus, un élément de modélisation ne peut appartenir qu'à un seul paquetage. Les paquetages divisent et organisent les modèles de la même manière que les répertoires organisent les systèmes de fichiers. Il est ainsi possible de gérer des modèles complexes composés d'un grand nombre d'éléments.

Les paquetages peuvent contenir ou référencer des paquetages, des classificateurs, des associations, des généralisations, des dépendances, des collaborations, des machines à états et/ou des stéréotypes. Chaque paquetage correspond à un sous-ensemble du modèle et possède, selon le modèle, des classes, des objets, des relations, des composants, des cas d'utilisation ou des nœuds, ainsi que les diagrammes associés ou d'autres paquetages.

La décomposition en paquetages n'est pas l'amorce d'une décomposition fonctionnelle ; chaque paquetage est un regroupement d'éléments selon un critère purement logique. Ce critère n'est pas défini par UML. L'objectif de la décomposition en paquetage est d'avoir une cohérence forte entre éléments d'un même paquetage et un couplage faible entre paquetages. En général, le contenu d'un paquetage est constitué d'éléments qui forment un tout cohérent et qui sont proches sémantiquement ; différentes vues d'un système sont souvent définies dans différents paquetages. La forme générale du système (l'architecture du système) est exprimée par la hiérarchie de paquetages et par le réseau de relations de dépendance entre paquetages.

Un paquetage est lui-même un élément de modélisation. Il peut ainsi contenir d'autres paquetages ; les niveaux d'emboîtement étant illimités. Un niveau donné peut contenir un mélange de paquetages et d'autres éléments de modélisation, de la même manière qu'un répertoire peut contenir des répertoires et des fichiers. Le paquetage de plus haut niveau est le paquetage racine de l'ensemble d'un modèle ; il constitue la base de la hiérarchie de paquetages. Le stéréotype **<<racine>>** permet de le désigner.

Graphiquement, le contenu d'un paquetage est placé dans le dossier qui représente le paquetage ou à l'extérieur du dossier en utilisant des lignes pour relier les éléments de modélisation du contenu au dossier du paquetage ; dans ce dernier cas, un cercle avec un signe + est attaché du côté du paquetage conteneur.

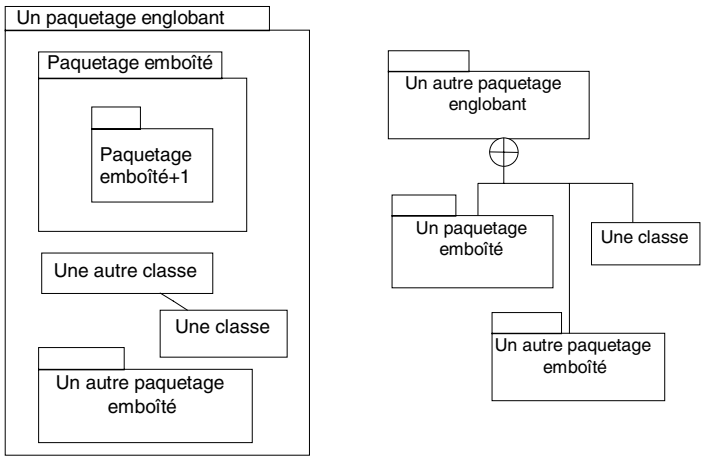


Figure 3–23. *Exemple de représentation mixte. Un paquetage contient des éléments de modélisation, accompagnés éventuellement d'autres paquetages.*

Par souci de clarté, le contenu d'un paquetage n'est pas toujours représenté dans son intégralité. Lorsqu'une partie ou la totalité du contenu est représentée, le nom du paquetage est placé dans l'onglet du dossier. Il est possible de préciser la visibilité des éléments contenus dans un paquetage en faisant précéder les noms des éléments de modélisation d'un symbole de visibilité (**+** pour public, **-** pour privé ou **#** pour protégé).

En outre, les paquetages peuvent être stéréotypés ou posséder des valeurs marquées et des contraintes. UML définit les quatre stéréotypes standard suivants pour les paquetages :

- **`Façade`**. Un tel paquetage définit une vue simplifiée d'un ensemble d'autres paquetages. En général, un paquetage stéréotypé **`Façade`** ne possède pas d'éléments ; il référence uniquement certains éléments des autres paquetages.
- **`Framework`**. Un tel paquetage contient une charpente applicative.

- **`Souche`**. Un tel paquetage fournit uniquement une partie publique.
- **`Racine`**. Il s'agit du paquetage de plus haut niveau dans une hiérarchie de paquetages.

Espace de nommage

Chaque paquetage définit un espace de nommage. Cela signifie que tous les éléments contenus dans un paquetage se distinguent par leur appartenance au paquetage englobant. Deux éléments de modélisation, contenus dans deux paquetages différents, peuvent ainsi porter le même nom. En revanche, les éléments contenus dans un paquetage (exception faite des associations et des généralisations) possèdent un nom unique.

Les éléments de modélisation situés au niveau le plus haut sont par défaut contenus dans un paquetage racine. Ils doivent également avoir un nom unique.

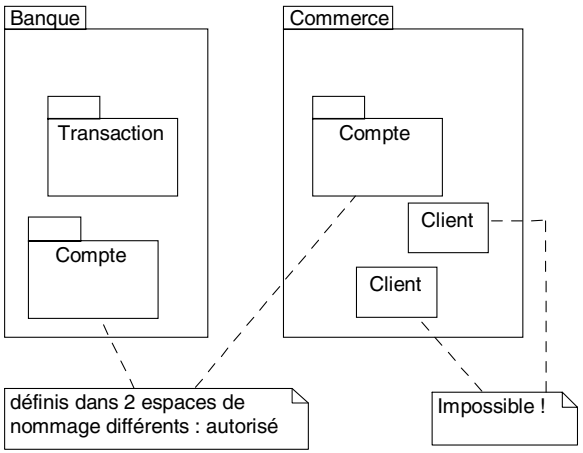


Figure 3–24. *Un paquetage définit un espace de nommage.*

Deux éléments de modélisation ayant le même nom, contenus dans des paquetages différents, ne sont pas identiques. Par exemple, le paquetage **`Compte`** contenu dans le paquetage **`Banque`** n'est pas le même que le paquetage **`Compte`** du paquetage **`Commerce`**. L'identification d'un élément est obtenue par son nom qui est unique dans le contexte du paquetage englobant. On parle alors de *nom simple*. Le *nom complet* préfixe le nom simple avec des informations liées à la hiérarchie des paquetages englobants.

L'opérateur **`::`** permet de séparer les différents noms de la hiérarchie de paquetages.

Par exemple, l'expression **`Zoo::Mammifère`** désigne le paquetage **`Mammifère`** défini dans le paquetage **`Zoo`**.

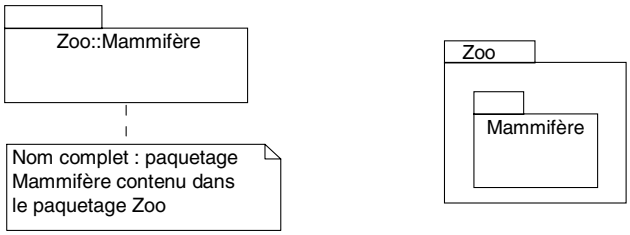


Figure 3–25. *Nom complet et nom simple.*

Le nombre de préfixes d'un nom complet n'est pas limité. De plus, le nom complet est utilisé aussi bien pour les paquetages que pour les autres éléments de modélisation.

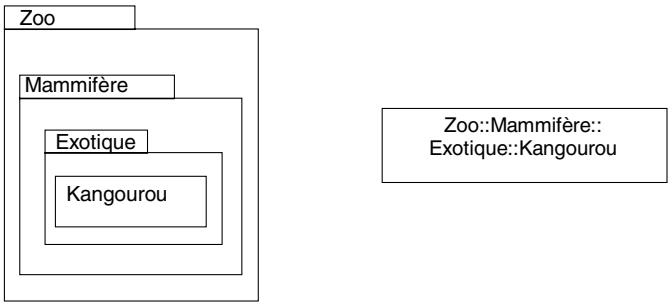


Figure 3–26. *Le nom complet précise le « chemin » à travers les paquetages englobants, jusqu'à l'élément identifié.*

Il est à noter qu'un paquetage ne peut pas être instancié ; il s'agit uniquement d'un conteneur qui définit un espace de nommage, sans existence dans le monde réel.

Dépendances entre paquetages

Les éléments contenus dans un paquetage emboîté voient les éléments contenus dans leur paquetage ou dans les paquetages englobants. Ils peuvent être associés entre eux à travers les niveaux d'emboîtement. L'accès d'un paquetage englobant au contenu d'un paquetage emboîté est impossible de prime abord. Pour avoir accès à des éléments qui ne sont pas accessibles par défaut, il faut définir une relation de dépendance entre les paquetages concernés. Une relation de dépendance entre paquetages doit exister dès que deux éléments issus de deux paquetages différents sont associés, hormis les cas de dépendances implicites (généralisation de paquetages et emboîtement de paquetages).

Les dépendances entre paquetages représentent des relations entre paquetages et non entre éléments individuels. Elles se représentent dans les diagrammes de classes, les diagrammes de cas d'utilisation et les diagrammes de composants. Une relation de dépendance est orientée du paquetage source (client) vers le paquetage cible (fournisseur). Les dépendances sont ainsi unidirectionnelles.

UML définit deux stéréotypes standard associés aux paquetages : **<<importe>>** et **<<accède>>**. Les relations de dépendance ainsi stéréotypées impliquent que les éléments du paquetage destination sont visibles dans le paquetage source ; toutefois, il n'y a pas de relation de possession entre ces paquetages.

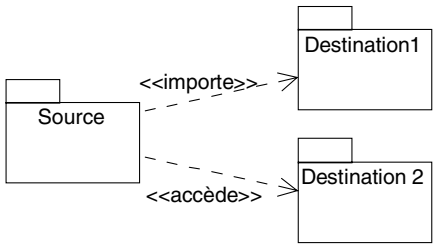


Figure 3–27. *Représentation de l'importation et de l'accès à des éléments de paquetage au moyen de relations de dépendance stéréotypées.*

La dépendance stéréotypée **<<importe>>** ajoute les éléments du paquetage destination à l'espace de nommage défini par le paquetage source. Il faut vérifier l'unicité des noms. Des alias peuvent être définis lors de l'importation pour éviter des conflits de noms. Par la suite, il est possible de distinguer les éléments importés sans utiliser leur nom complet. La dépendance stéréotypée **<<accède>>** permet de référencer des éléments du paquetage destination. Il est nécessaire de mentionner leur nom complet. Les stéréotypes **<<accède>>** et **<<importe>>** s'apparentent respectivement aux clauses de contexte **with** et **use** du langage Ada. Chaque élément contenu dans un paquetage possède un paramètre qui signale si l'élément est visible ou non à l'extérieur de ce paquetage. Les valeurs prises par le paramètre sont : public (symbolisé par **+**), protégé (symbolisé par **#**) ou privé (symbolisé par **-**). Les éléments publics sont accessibles de l'extérieur du paquetage et constituent les exportations du paquetage. Ceux qui sont privés ou protégés ne sont pas accessibles, même s'il existe une dépendance stéréotypée.

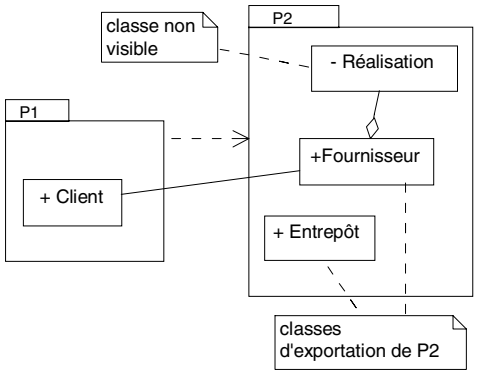


Figure 3–28. *Dans le cadre d'une dépendance, toutes les classes publiques sont accessibles et uniquement celles-là.*

Il est possible de redéfinir la visibilité des éléments contenus dans un paquetage une fois qu'ils sont importés. La dépendance **<<importe>>** peut être transitive. Par défaut, un élément importé (au moyen d'une dépendance **<<importe>>**) dans un paquetage est invisible à l'extérieur de ce paquetage. Toutefois, il est possible de rendre l'élément importé public de sorte qu'il devienne visible à l'extérieur du paquetage qui réalise l'importation. La dépendance **<<accède>>** n'est pas transitive. L'accès dans un paquetage à des éléments rendus accessibles par une précédente dépendance **<<accède>>**, ou leur importation, est impossible.

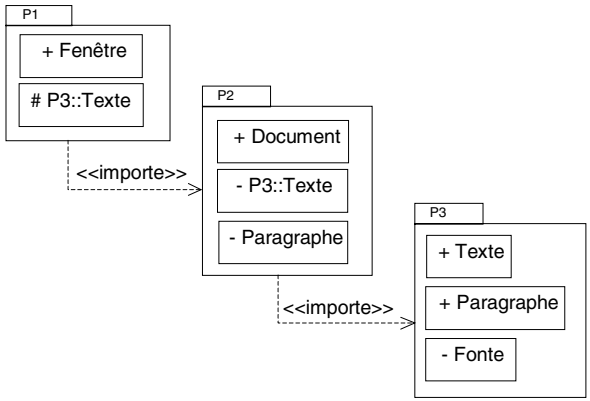


Figure 3–29. *Dans* **P1**, *la classe* **Texte** *de* **P3** *est accessible transitivement.*

Les relations de dépendance entre paquetages entraînent des relations d'obsolescence entre les éléments de modélisation contenus dans ces paquetages. Pour des raisons de compilation lors de la réalisation, il est fortement conseillé de faire en sorte que les dépendances entre paquetages forment uniquement des graphes acycliques. Il faut donc éviter la situation suivante :

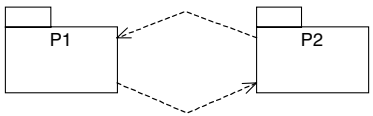


Figure 3–30. *Exemple de dépendance circulaire entre paquetages, à éviter.*

De même, il convient d'éviter les dépendances circulaires transitives.

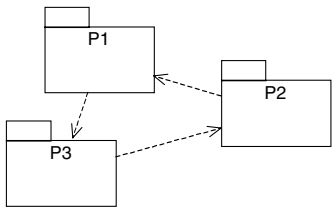


Figure 3–31. *Exemple de dépendance circulaire transitive, à éviter.*

En règle générale, il est possible de réduire les dépendances circulaires en éclatant un des paquetages incriminés en deux paquetages plus petits, ou en introduisant un troisième paquetage intermédiaire. Certains paquetages sont utilisés par tous les autres paquetages. Ces paquetages regroupent par exemple des classes de base comme des ensembles, des listes, des files, ou encore des classes de traitement des erreurs. Ces paquetages peuvent posséder la valeur marquée {**global**} qui les désigne comme paquetages globaux. Il n'est pas nécessaire de montrer les relations de dépendance entre ces paquetages et leurs utilisateurs ce qui limite la charge graphique des diagrammes.

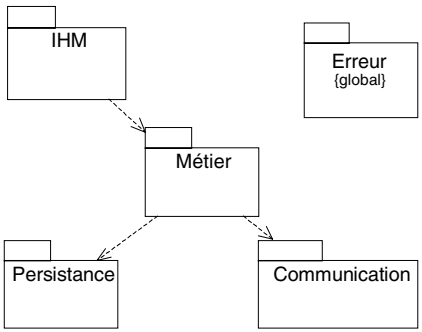


Figure 3–32.
Afin de réduire la charge graphique, les dépendances concernant les paquetages visibles globalement ne sont pas portées dans les diagrammes.

Généralisation

Un paquetage peut participer à une relation de généralisation. La généralisation des paquetages est comparable à celle des classes, présentée par la suite. Il est possible d'ajouter ou de spécialiser des éléments de paquetages en modélisant des relations de généralisation entre paquetages. Les éléments publics et protégés sont alors accessibles dans les paquetages de spécialisation ; les éléments privés non.

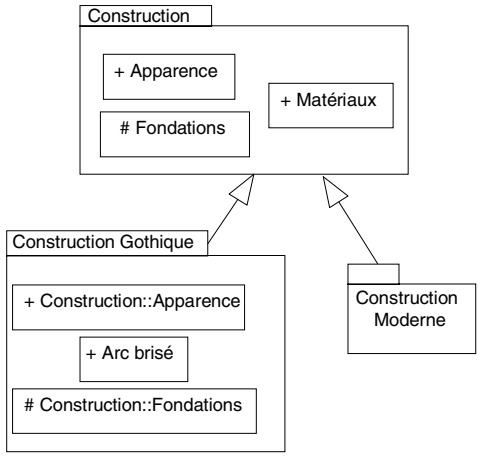


Figure 3–33.
Représentation de la généralisation de paquetages.

Le paquetage **`Construction Gothique`** ajoute une classe **`Arc brisé`** et intègre les classes **`Fondations`** et **`Apparence`**. Le principe de substitution s'applique : tout paquetage de spécialisation (tel que **`Construction Gothique`**) peut être utilisé en lieu et place du paquetage qu'il spécialise (tel que **`Construction`**).

Les diagrammes de classes

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Outre les classes, ils présentent un ensemble d'interfaces et de paquetages, ainsi que leurs relations.

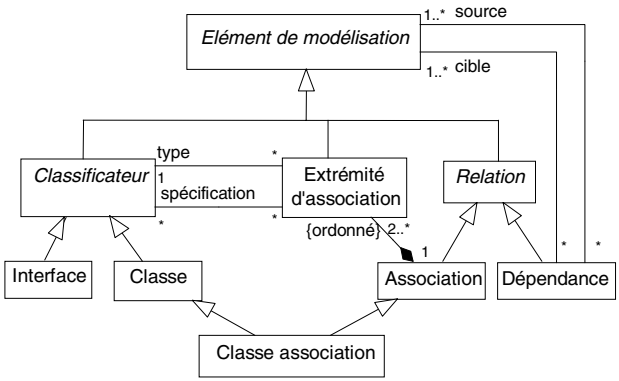


Figure 3–34. *Extrait du métamodèle. Relations entre les éléments de modélisation des diagrammes de classes.*

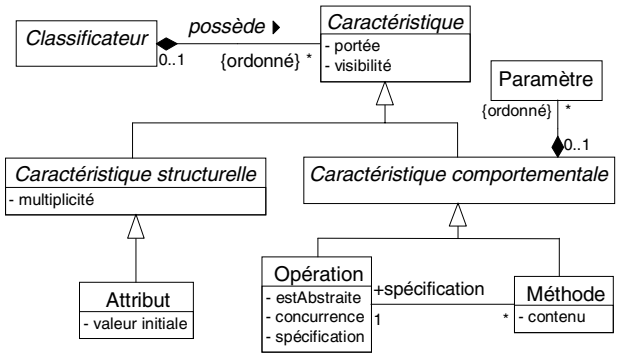


Figure 3–35. *Extrait du métamodèle. Caractéristiques de la classe* **`Classificateur`**.

De même qu'une classe décrit un ensemble d'objets, une association décrit un ensemble de liens ; les objets sont instances des classes et les liens sont instances des associations. Un diagramme de classes n'exprime rien de particulier sur les liens d'un objet donné, mais décrit de manière abstraite les liens potentiels d'un objet vers d'autres objets.

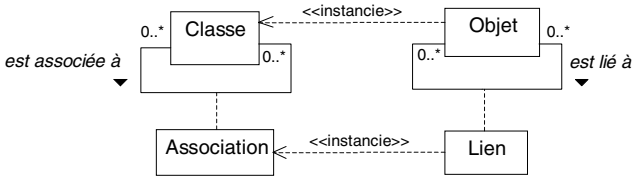


Figure 3–36.
Correspondances entre, d'une part, les classes et leurs associations dans les diagrammes de classes, et, d'autre part, les objets et leurs liens dans les diagrammes d'objets.

Pour un système donné, un ensemble de diagrammes de classes est défini ; chaque diagramme de classes se focalise sur un seul aspect afin de rester simple et intelligible.

Les classes

Les classes sont représentées par des rectangles compartimentés. Le premier compartiment contient le nom de la classe qui est unique dans le paquetage qui contient cette classe. En général, le nom d'une classe utilise le vocabulaire du domaine. Le nom de la classe doit exprimer ce que la classe est, et non ce qu'elle fait. Une classe n'est pas une fonction, une classe est une description abstraite – condensée – d'un ensemble d'objets du domaine de l'application ; elle définit leur structure, leur comportement et leurs relations. Deux autres compartiments sont généralement ajoutés ; ils contiennent respectivement les attributs et les opérations de la classe. Il est possible de définir des compartiments supplémentaires, par exemple pour lister les responsabilités ou les exceptions.

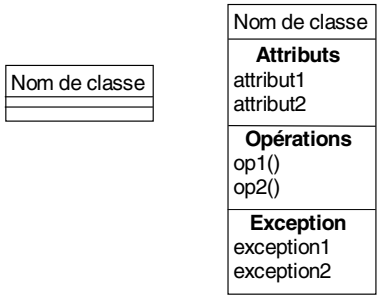
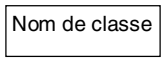


Figure 3–37.
Représentation graphique des classes.

La norme UML précise que les noms des classes sont indiqués en gras. Toutefois, dans cet ouvrage ils ne sont pas présentés ainsi. Une classe contient toujours au moins son nom. Lorsque le nom n'a pas encore été trouvé, ou que le choix n'est pas encore arrêté, il est conseillé de faire figurer un libellé générique, facile à identifier, comme **`A_Définir`**.

Les compartiments d'une classe peuvent être supprimés (en totalité ou en partie) lorsque leur contenu n'est pas pertinent dans le contexte d'un diagramme. La suppression des compartiments reste purement visuelle ; elle ne signifie pas qu'il n'y a pas d'attribut ou d'opération.

Figure 3–38.
Représentation graphique simplifiée par suppression des compartiments.

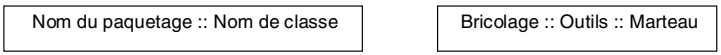


Il est possible de mentionner un nom complet pour indiquer le paquetage contenant la classe. La syntaxe utilisée est alors :

```
nom paquetage '::' nom de la classe
```

Il est également possible de préciser le nom du paquetage par rapport aux paquetages englobant en utilisant "`::`" comme séparateur.

Figure 3–39.
Représentation des classes avec nom complet.



Le rectangle qui symbolise la classe peut contenir un stéréotype et des propriétés. UML définit les stéréotypes de classe (et de classificateur) suivants :

- **`<<classe implémentation>>`**. Il s'agit de l'implémentation d'une classe dans un langage de programmation.
- **`<<énumération>>`**. Il s'agit d'une classe qui définit un ensemble d'identificateurs formant le domaine de valeur d'un type.
- **`<<métaclasse>>`**. Il s'agit de la classe d'une classe, comme en Smalltalk.
- **`<<powertype>>`**. Une classe est un métatype : ses instances sont toutes des sous-types d'un type donné.
- **`<<processus>>`**. Il s'agit d'une classe active qui représente un flot de contrôle *lourd*.
- **`<<thread>>`**. Il s'agit d'une classe qui représente un flot de contrôle *léger*.
- **`<<type>>`**. Il s'agit d'une classe qui définit un domaine d'objets et les opérations applicables à ces objets.
- **`<<utilitaire>>`**. Il s'agit d'une classe réduite au concept de module et qui ne peut être instanciée.

Un mot clé, indiqué entre guillemets, peut être utilisé pour renforcer la description d'une classe. La différence entre un mot clé et un stéréotype réside dans le fait que le mot clé apparaît dans le métamodèle ; il ne s'agit donc pas d'une extension du langage UML. La liste des mots clés qui apparaissent fréquemment dans le compartiment du nom d'une classe sont :

- **`<<acteur>>`**. La classe modélise un ensemble de rôles joués par un acteur intervenant dans le système.
- **`<<interface>>`**. La classe contient uniquement une description des opérations visibles.

- **<<signal>>** ou **<<exception>>** (cas particulier de signal). La classe modélise des éléments de type signal. En général, la classe est liée via des relations de dépendance stéréotypées <<**envoie**>>, aux opérations qui génèrent les signaux.

Les propriétés désignent toutes les valeurs attachées à un élément de modélisation, comme les attributs, les associations et les valeurs marquées. Une valeur marquée est une paire **(attribut, valeur)** définie par l'utilisateur ; elle permet de préciser par exemple des informations de génération de code, d'identification ou de références croisées.

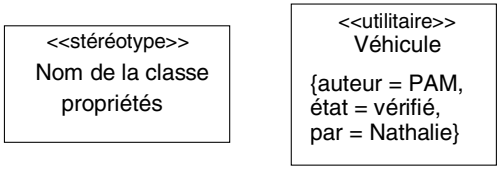


Figure 3–40.
Les compartiments des classes peuvent contenir un stéréotype et des propriétés, définis librement par l'utilisateur.

Les attributs et les opérations

Les attributs et les opérations peuvent figurer de manière exhaustive ou non, dans les compartiments des classes. En général, il vaut mieux limiter la visualisation des attributs et des opérations en se focalisant sur les éléments pertinents dans un diagramme donné. Par exemple, un nom est donné en analyse mais la spécification de la visibilité et le choix du type sont souvent différés. Il est parfois utile de préciser que la liste d'éléments d'un compartiment n'est pas complète ; dans ce cas, une ellipse (**...**) est placée en fin de liste.

Par convention, le deuxième compartiment contient les attributs et le troisième contient les opérations. Lorsque cet ordre est différent ou que des compartiments supplémentaires sont ajoutés, il est possible de nommer ces compartiments pour lever toute ambiguïté.

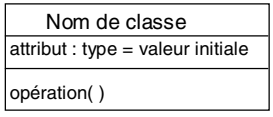


Figure 3–41.
Des attributs et des opérations dans les compartiments de la classe.

L'utilisateur classe à sa guise les éléments d'un compartiment (ordre alphabétique, regroupement par catégories grâce à des stéréotypes ou par type de visibilité, etc.), mais cet agencement n'apporte aucune information de modélisation supplémentaire.

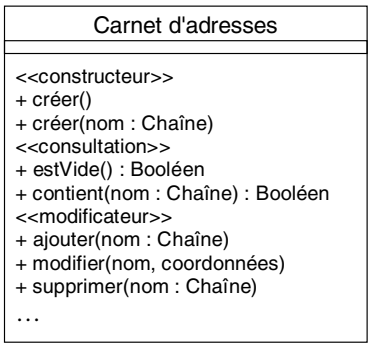


Figure 3–42. *Exemple de classification d'opérations.*

La syntaxe retenue pour la description des attributs est la suivante :

```
Visibilité Nom_Attribut ["[" Multiplicité "]"] ":"
    Type_Attribut ['=' Valeur_Initiale]
    { '{' Propriété '}' }
```

avec

```
Visibilité  :=  '+' | '-' | '#'
Multiplicité:=  (Intervalle | nombre)
    [',' Multiplicité]
Intervalle  :=  Limite_Inférieure'..'Limite_Supérieure
Limite_Inférieure := entier_positif | 0
Limite_Supérieure := nombre
nombre ::= entier_positif | '*' (illimité)
```

`Type_Attribut` : spécification du type de l'attribut

`Valeur_initiale` : expression qui définit la valeur initiale d'un objet nouvellement créé

`Propriété` : une propriété et sa valeur

Le type des attributs peut être une classe (**`Rectangle`**, **`Cercle`**…), un type primitif (**`Entier`**, **`Chaîne`**…) ou une expression complexe dont la syntaxe n'est pas précisée par UML (**`tableau[temps] de Points`**, etc.). UML exprime la mutabilité des attributs au moyen de la propriété **`modification`**, dont les trois valeurs prédéfinies sont :

- **`gelé`** : attribut non modifiable (l'équivalent du **`const`** de C++) ;
- **`variable`** : propriété par défaut qui définit un attribut modifiable ;
- **`ajoutUniquement`** : seul l'ajout est possible (pour des attributs avec multiplicité supérieure à 1).

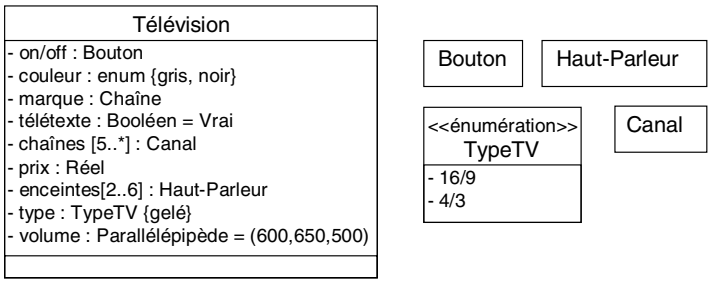


Figure 3–43.
Exemple de divers types d'attributs.

La visibilité et le type de l'attribut existent toujours, mais ne sont éventuellement pas représentés dans une classe d'un diagramme donné.

Il est à noter que les attributs d'une classe peuvent également être représentés par composition, ce qui permet de prendre en compte les attributs non valués (lorsque la multiplicité du côté de l'agrégat débute à 0).

Dans ce cas, la multiplicité est indiquée à proximité de l'extrémité d'association concernée, et la visibilité est mentionnée avec le rôle concerné.

Il est possible de compléter progressivement cette description des attributs, lors de la transition de l'analyse à la conception, par exemple en précisant les types ou les valeurs initiales.

Des propriétés redondantes sont parfois spécifiées lors de l'analyse des besoins. Les attributs dérivés offrent une solution pour allouer des propriétés à des classes, tout en indiquant clairement que ces propriétés sont dérivées d'autres propriétés déjà définies.

Les attributs dérivés sont indiqués en faisant précéder leur nom d'un **/**. Dans l'exemple suivant, la classe **Rectangle** possède les attributs **Longueur** et **Largeur**, ainsi qu'un attribut dérivé **Surface**, qui peut être construit à partir des deux autres attributs.

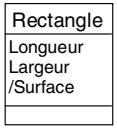


Figure 3–44.
Exemple d'attribut dérivé. La surface d'un rectangle peut être déterminée à partir de la longueur et de la largeur.

Ce principe de dérivation est également vrai pour des associations.

Le mécanisme de construction d'un élément dérivé à partir d'autres éléments peut être spécifié avec une dépendance stéréotypée **<<dérive>>** ; souvent, seule une note, exprimant la manière dont l'élément dérivé est calculé, est précisée.

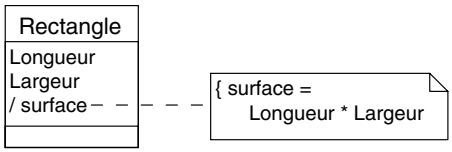


Figure 3–45.
Spécification du mécanisme de construction d'un attribut dérivé.

Au moment de la conception, l'attribut dérivé **`/Surface`** est transformé en une opération **`Surface()`** qui encapsule le calcul de surface. Cette transformation peut toutefois être effectuée sans attendre, dès lors que la nature dérivée de la propriété **`surface`** est détectée.

Figure 3–46.
Transformation de l'attribut dérivé en une opération de calcul de la surface.

Le compartiment des opérations liste les opérations définies par la classe et les méthodes qu'elle fournit (une opération est un service qu'une instance de la classe peut réaliser alors qu'une méthode est l'implémentation d'une opération). La syntaxe retenue pour la description des opérations est la suivante :

```
Visibilité Nom_Opération "(" Arguments ")"
':' Type_Retourné { "{" Propriété "}" }
```

Avec

```
Visibilité ::=  '+' | '-' | '#'
Arguments  ::= Direction Nom_Argument ':'Type_Argument
  ['=' Valeur_Par_Défaut] [','Arguments]
Direction  ::= in | out | inout
```

`Type_Retourné` : précise le type retourné par l'opération. Une liste de types peut être définie pour indiquer plusieurs valeurs de retour.

`Propriété` : une propriété et sa valeur

Toutefois, étant donné la longueur du libellé, il est permis d'omettre les arguments des opérations, la visibilité et/ou le type retourné dans certains diagrammes. La direction d'un argument d'une opération est par défaut **`in`** ; les trois directions définies par UML sont :

- **`in`** : l'argument est un paramètre en entrée seule et non modifié par l'exécution de cette opération.
- **`out`** : l'argument est un paramètre en sortie seule ; l'appelant peut ainsi récupérer des informations.
- **`inout`** : l'argument est un paramètre en entrée-sortie, passé à l'opération et modifiable.

La liste des propriétés prédéfinies applicables aux opérations est la suivante :

- **{requête}** indique que l'opération n'altère pas l'état de l'instance concernée (la propriété est de type booléen ; la valeur **vrai** peut être omise).
- **{concurrence = valeur}** où valeur est séquentiel, gardé (l'intégrité de l'objet est garantie par un mécanisme de synchronisation externe) ou concurrent.
- **{abstrait}** indique une opération non implémentée dans la classe. Le nom d'une opération abstraite peut également être rendu par une fonte en italique (la propriété est de type booléen ; la valeur vrai peut être omise).
- **{estFeuille}** indique que l'opération ne peut pas être redéfinie dans une sous-classe (la propriété est de type booléen).
- **{estRacine}** indique que l'opération est définie pour la première fois dans une hiérarchie de classes (la propriété est de type booléen).

Les opérations peuvent également être stéréotypées. Le stéréotype, indiqué entre guillemets, est placé avant l'indicateur de visibilité de l'opération.

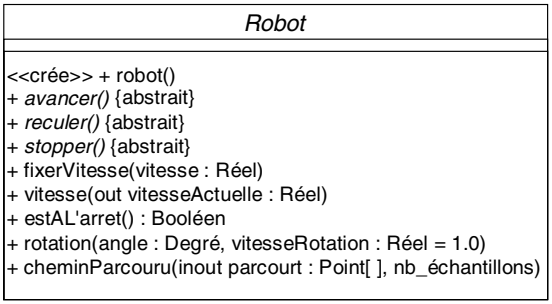


Figure 3–47. *Exemple de représentation des opérations.*

Il est possible de donner des informations sur les méthodes au moyen de notes attachées aux opérations. Les pré-conditions et post-conditions peuvent être ajoutées au moyen de contraintes stéréotypées ou d'expressions en OCL (langage de contraintes défini par UML et présenté plus loin dans cet ouvrage).

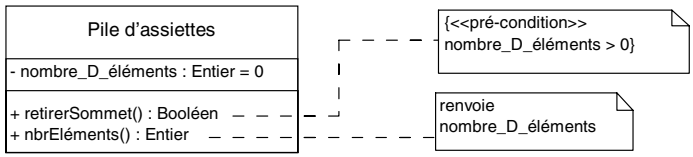


Figure 3–48. *Représentation du comportement d'une opération et de sa pré-condition.*

En outre, la réception d'un signal par une classe peut être indiquée comme une opération. Dans le compartiment des opérations, le signal, qui peut être réceptionné, est précisé en utilisant la syntaxe d'une opération et en faisant précéder sa signature par le mot clé **<<signal>>**. La réponse, provoquée par la réception du signal, est ensuite représentée dans un automate d'états.

Visibilité et portée des attributs et des opérations

UML définit trois niveaux de visibilité pour les attributs et les opérations :

- *public* : l'élément est visible pour tous les clients de la classe ;
- *protégé* : l'élément est visible pour les sous-classes de la classe ;
- *privé* : l'élément est visible pour la classe seule.

L'information de visibilité ne figure pas toujours de manière explicite dans les diagrammes de classes, ce qui ne veut pas dire que la visibilité n'est pas définie dans le modèle. Par défaut, le niveau de visibilité est symbolisé par les caractères **+**, **#** et **-**, qui correspondent respectivement aux niveaux **`public`**, **`protégé`** et **`privé`**. Pour regrouper des attributs ou des opérations ayant une même visibilité, il est possible d'utiliser un mot clé **`public`**, **`protégé`** ou **`privé`**, puis de faire suivre la liste des opérations ou attributs concernés par cette visibilité. Des visibilités particulières peuvent également être spécifiées sous forme d'une propriété de type chaîne de caractères. Certains attributs et opérations peuvent être visibles globalement, dans toute la portée lexicale de la classe (par défaut, la portée est celle de l'instance). Ces éléments, également appelés attributs et opérations de classe, sont représentés comme un objet, avec leur nom souligné. La notation se justifie par le fait qu'un attribut de classe apparaît comme un objet partagé par les instances d'une classe. Par extension, les opérations de classe sont aussi soulignées.

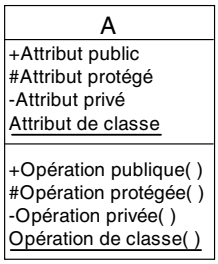


Figure 3–49.
Représentation des différents niveaux de visibilité des attributs et des opérations.

Les compartiments supplémentaires

D'autres compartiments peuvent être ajoutés, en plus des trois compartiments « standard » (nom, attributs et opérations). Il est ainsi possible de préciser les responsabilités de la classe (issues du concept des cartes CRC[4]), les événements qu'elle génère et les exceptions qu'elle accepte. Aucune syntaxe spéciale n'est spécifiée par UML pour la définition de ces informations supplémentaires.

[4] K. Beck, W. Cunningham L. A laboratory for teaching object-oriented thinking, *SIGPLAN Notices* vol. 24 (10), October 1989.

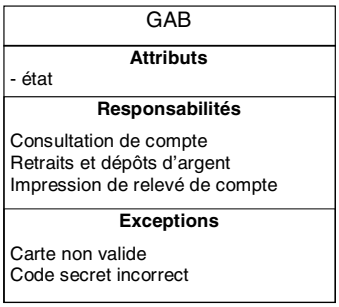


Figure 3–50.
Les responsabilités et les exceptions d'une classe sont représentées dans des compartiments supplémentaires.

Les signaux (et les exceptions) réceptionnés par une classe peuvent également être indiqués comme des opérations. Il est possible de préciser un signal traité par une classe dans le compartiment des opérations en utilisant la syntaxe d'une opération et en faisant précéder sa signature par le mot clé **<<signal>>**. D'autre part, les événements générés par la classe peuvent être modélisés avec une relation de dépendance stéréotypée **<<envoie>>** ayant pour source la classe qui émet le signal ou l'exception, et comme cible une classe stéréotypée **<<signal>>** ou **<<exception>>**. La définition d'une exception ou d'un signal se fait avec une classe stéréotypée **<<exception>>** ou **<<signal>>**.

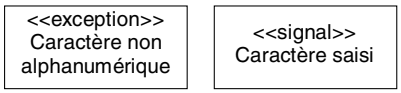


Figure 3–51.
Définition d'une exception et d'un signal.

Les interfaces

Une interface décrit le comportement visible d'une classe, d'un composant, d'un sous-système (décrits plus loin dans l'ouvrage), d'un paquetage ou d'un autre classificateur. Ce comportement visible est défini par une liste d'opérations ayant une visibilité **public** ; contrairement à un type, aucun attribut ou association n'est défini dans une interface. UML représente les interfaces au moyen de petits cercles reliés par un trait à l'élément qui fournit les services décrits par l'interface.

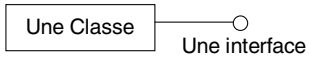


Figure 3–52.
Représentation d'une interface au moyen d'un petit cercle relié à la classe qui fournit effectivement les services.

Les interfaces peuvent également se représenter au moyen de classes avec le mot clé **<<interface>>**. Cette notation permet de faire figurer les services de l'interface dans le compartiment des opérations. La classe qui *réalise* cette interface est alors indiquée par une relation de réalisation (représentée par une flèche en traits pointillés et à extrémité triangulaire). Une même classe peut réaliser plusieurs interfaces et une interface peut être réalisée par plusieurs classes.

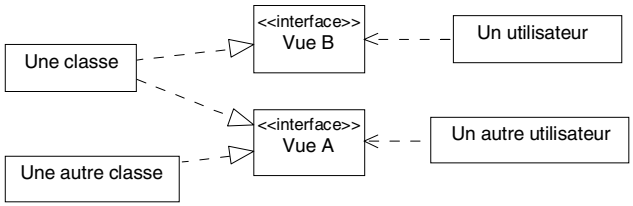


Figure 3–53.
Exemple de représentation des interfaces au moyen de classes stéréotypées.

La notation au moyen de classes **<<interface>>** permet également de représenter des relations de généralisation entre interfaces. La hiérarchie des interfaces n'est pas forcément la même que celle des classes qui les réalisent. Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par un ou plusieurs éléments. L'interface offre ainsi une alternative à la généralisation multiple.

Les dépendants d'une interface utilisent tout ou partie des services décrits dans l'interface. Le diagramme suivant illustre la modélisation de deux interfaces **Crédit** et **Assurance** d'une classe **Banque**. Une relation de réalisation indique que la classe **Banque** réalise l'interface **Assurance**.

Cette notation permet de représenter clairement la séparation entre la description du comportement et sa réalisation.

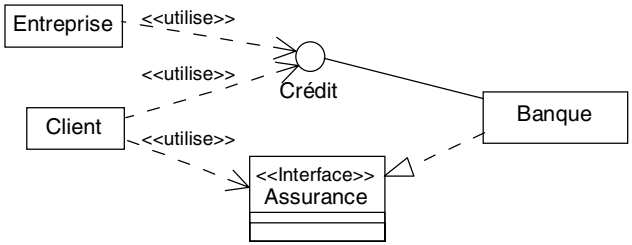


Figure 3–54.
Représentation des deux interfaces d'une classe **Banque**.

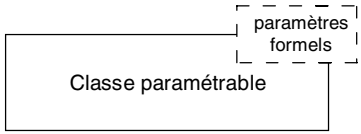
Une interface possède uniquement la spécification d'un comportement visible, sous forme d'un ensemble d'opérations (pas d'attributs et pas d'associations), et ne fournit aucune implémentation de ses services. Cette dernière caractéristique implique que des éléments de modélisation accèdent ou utilisent une interface mais que les éléments de modélisation ne sont pas visibles pour l'interface.

Les classes paramétrables

Les classes paramétrables, aussi appelées classes templates, sont des modèles de classes. Elles correspondent aux classes génériques d'Eiffel et aux templates de C++. Une classe paramétrable ne peut être utilisée telle quelle. Il convient d'abord de *lier* les paramètres formels à des paramètres effectifs (le terme

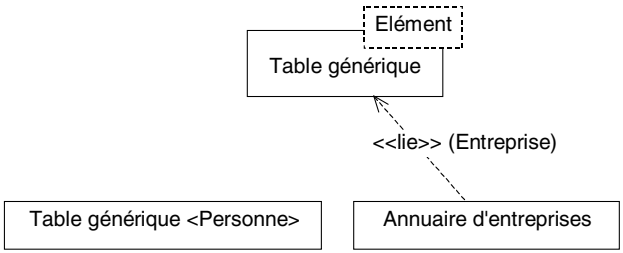
instancier est également utilisé) afin d'obtenir une classe paramétrée qui pourra à son tour être instanciée afin de donner des objets. Lors de l'instanciation, les paramètres effectifs personnalisent la classe réelle obtenue à partir de la classe template. Les classes paramétrables permettent de construire des collections universelles, typées grâce aux paramètres effectifs.

Figure 3–55.
Représentation graphique des classes paramétrables.



Ce type de classes n'apparaît généralement pas en analyse, sauf dans le cas particulier de la modélisation d'un environnement de développement. Les classes paramétrables sont surtout utilisées en conception détaillée, pour incorporer par exemple des composants réutilisables. La figure suivante représente deux possibilités de représentation de l'instanciation d'une table générique, pour réaliser un annuaire de personnes et pour réaliser un annuaire d'entreprises.

Figure 3–56.
Exemple de représentation de l'instanciation d'une classe paramétrable.



Le paramètre générique formel figure dans le rectangle pointillé de la classe paramétrable alors que le paramètre générique effectif est 1) placé entre des symboles **< >** et accolé au nom de la classe obtenue par instanciation ou 2) défini comme arguments d'une relation de dépendance (orientée de l'instance paramétrée vers la classe paramétrable) avec le mot clé **<<lie>>**.

La déclaration des paramètres formels suit le format :

```
Nom_Paramètre ':' Type ['=' Valeur_Par_Defaut]
```

La déclaration de liaison entre une classe paramétrable instanciée et une classe paramétrable répond au format :

```
Nom_Template '<' Valeur {',' Valeur} '>'
```

`Valeur` : une instance du `Type` défini dans la déclaration des paramètres formels.

Plusieurs paramètres génériques peuvent être définis pour une classe paramétrable, comme l'illustre la figure suivante :

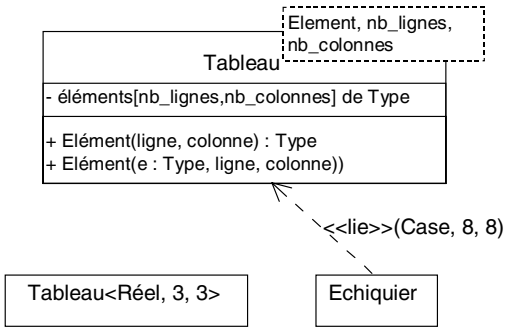


Figure 3–57. *Représentation d'une classe paramétrable avec plusieurs paramètres génériques et son instanciation.*

Les classes utilitaires

Il est parfois utile de regrouper des éléments (les fonctions d'une bibliothèque mathématique, par exemple) au sein d'un module, sans pour autant construire une classe complète. La classe utilitaire permet de représenter de tels modules et de les manipuler graphiquement comme les classes conventionnelles.

Les attributs et les procédures contenus dans une classe utilitaire deviennent des variables et des procédures globales ; la portée n'a pas besoin d'être spécifiée puisque les attributs et opérations sont automatiquement visibles dans la portée lexicale de classe. Le stéréotype **`<<Utilitaire>>`** spécialise les classes en classes utilitaires.

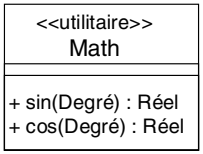


Figure 3–58. *Exemple de représentation graphique d'une classe utilitaire.*

En C++, une classe utilitaire correspond à une classe qui ne contient que des membres statiques (fonctions et données).

Les associations

Les associations représentent des relations structurelles entre classes d'objets. Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées. Une association compte au moins deux extrémités d'association, reliées à des classificateurs.

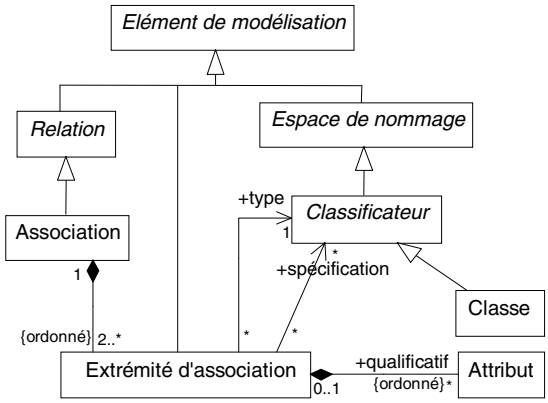


Figure 3–59. *Extrait du métamodèle. Représentation des principales caractéristiques des associations.*

La plupart des associations sont binaires, c'est-à-dire qu'elles connectent deux classes. Les associations se représentent en traçant une ligne entre les classes associées.

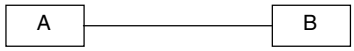


Figure 3–60. *Une ligne entre des classes représente une association.*

Les associations peuvent être représentées par des traits rectilignes ou obliques, selon les préférences de l'utilisateur. L'usage recommande de se tenir le plus possible à un seul style de traits afin de simplifier la lecture des diagrammes au sein d'un projet.

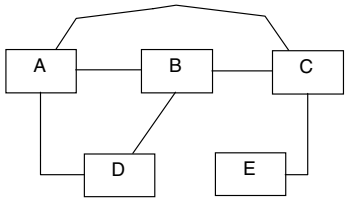


Figure 3–61. *Les associations peuvent être représentées par des traits rectilignes ou obliques selon les préférences de l'utilisateur.*

En règle générale, il faut éviter que les associations se croisent. Lorsqu'il est impossible d'éviter des croisements, il est possible de représenter un demi-cercle au point d'intersection (comme dans les schémas électriques). Les instances d'une association sont des tuples des instances des classificateurs reliés par cette association.

Chaque valeur de tuple doit être *unique* ; cela signifie que deux instances ne sont jamais reliées par plusieurs liens instances de la même association. UML impose ainsi une sémantique d'ensemble aux relations.

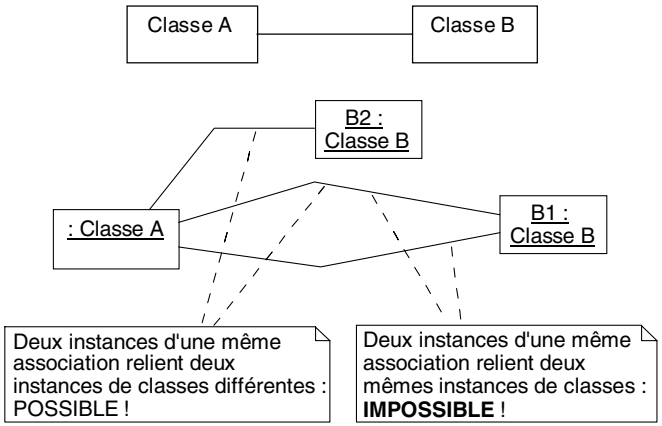


Figure 3–62. *Plusieurs liens instances d'une même association ne peuvent pas relier les deux mêmes instances.*

Arité des associations

La plupart des associations sont dites binaires car elles relient deux classes. Des arités supérieures peuvent cependant exister et se représentent alors au moyen d'un losange sur lequel arrivent les différentes composantes de l'association.

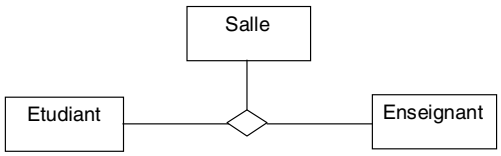


Figure 3–63. *Exemple de relation ternaire qui matérialise un cours.*

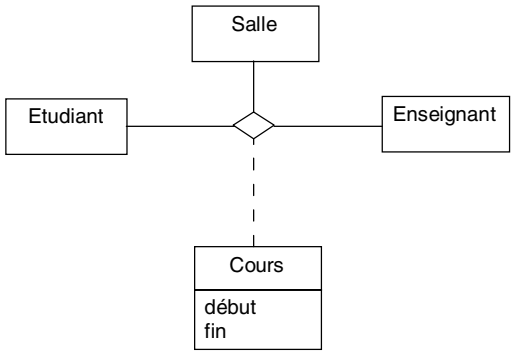


Figure 3–64. *Exemple de relation ternaire avec une classe-association pour la définition de deux attributs* **`début`** *et* **`fin`** *de cours.*

Généralement, les associations n-aires peuvent se représenter en promouvant l'association au rang de classe et en ajoutant une contrainte qui exprime que les multiples branches de l'association s'instancient toutes simultanément, en un même lien. Dans l'exemple suivant, la contrainte est exprimée au moyen d'un stéréotype (non standard) qui précise que la classe **`Cours`** réalise une association ternaire.

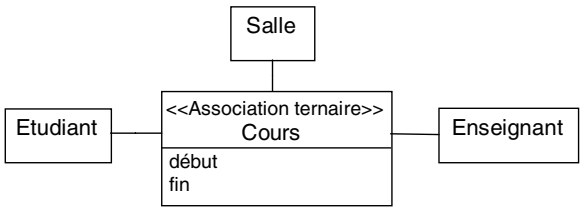


Figure 3–65.
Représentation d'une association ternaire au moyen d'une classe stéréotypée.

Comme pour les associations binaires, il est possible de nommer les extrémités d'une association n-aire pour décrire le rôle des classes dans l'association (voir la section *Rôle des extrémités d'association*). La difficulté de trouver un nom différent pour chaque extrémité d'une association n-aire est souvent le signe d'une association d'arité inférieure.

Nommage des associations

Les associations peuvent être nommées ; le nom de l'association figure alors au milieu de la ligne qui symbolise l'association, plus précisément au-dessus, au-dessous ou sur la ligne en question.

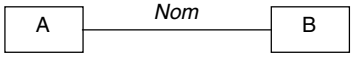


Figure 3–66.
Les associations peuvent être nommées afin de faciliter la compréhension des modèles.

Sans en faire une règle systématique, l'usage recommande de choisir une forme verbale active (comme *travaille pour*) *ou* passive (comme *est employé par*) en guise de nom d'association.

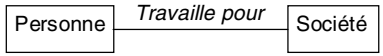


Figure 3–67.
Une forme verbale active en guise de nom d'association.

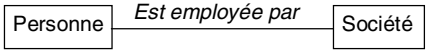
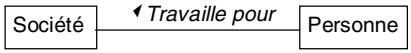


Figure 3–68.
Une forme verbale passive en guise de nom d'association.

Dans les deux cas, le sens de lecture du nom peut être précisé au moyen d'un petit triangle dirigé vers la classe désignée par la forme verbale et placé à proximité du nom de l'association. Par défaut, le sens de lecture est de gauche à droite. En général la précision du sens est donnée uniquement lorsque le sens de lecture est ambigu. Dans un but de simplification, le petit triangle peut être remplacé par les signes **<** et **>** disponibles dans toutes les fontes.

Figure 3–69.
Précision du sens de lecture du nom d'une association.

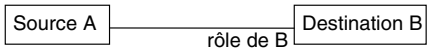


Les associations entre classes expriment principalement la structure statique ; de ce fait, le nommage par des formes verbales qui évoquent plutôt le comportement s'écarte un peu de l'esprit général des diagrammes de classes. Comme le nommage des associations, le nommage des extrémités des relations permet de clarifier les diagrammes, mais d'une manière plus passive, plus en phase avec la tonalité statique des diagrammes de classes.

Rôles des extrémités d'association

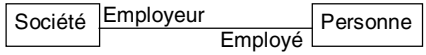
L'extrémité d'une association possède un nom (hérité de **`Elément de modélisation`**, voir la **Figure 3.59**). Ce nom, aussi appelé rôle, décrit comment une classe source voit une classe destination au travers de l'association. Il nomme également le passage d'une instance de la classe source à une ou plusieurs instances de la classe destination.

Figure 3–70.
Représentation du rôle de la classe **`Destination B`** *vu par la classe* **`Source A`**.



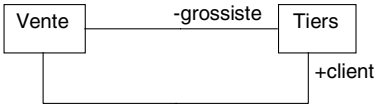
Le rôle représente un *pseudo-attribut* de la classe source (utilisé comme un attribut) et doit avoir un nom unique dans l'ensemble des noms d'attributs et pseudo-attributs de la classe source. Par exemple, dans la figure suivante, **`Employeur`** est un pseudo-attribut de **`Personne`**. Un rôle exprime comment une classe voit une autre classe au travers d'une association. Chaque association binaire possède deux rôles, un à chaque extrémité. Un rôle se présente sous forme nominale. Le nom d'un rôle se distingue du nom d'une association, car il est placé près d'une extrémité de l'association.

Figure 3–71.
Dans cet exemple, la personne voit la société comme son employeur, et la société voit la personne comme son employé.



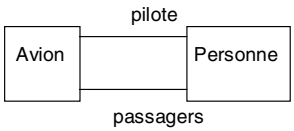
Une indication de visibilité (voir la section *Visibilité des attributs et des opérations*) peut être précisée. Elle est placée avant le rôle pour préciser la visibilité du rôle à l'extérieur de l'association. Par défaut, si rien n'est précisé, le rôle est considéré comme **`public`**.

Figure 3–72.
Lorsque les instances de **`Tiers`** *jouent le rôle de* **`grossiste`**, *elles sont* `privées` *; elles sont accessibles uniquement à l'instance de vente à laquelle elles sont associées. En observant un objet* **`Vente`**, *il est possible de voir le client mais pas le grossiste.*



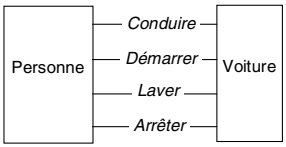
Il est possible de nommer à la fois les associations et les extrémités d'associations ; toutefois, l'usage mélange rarement les deux dénominations. Il est fréquent de commencer par décorer une association par un verbe, puis de se servir plus tard de ce verbe pour construire un substantif verbal qui définit le rôle correspondant. Le nommage des rôles tend ainsi à être plus important que celui de l'association ; le premier étant structurel (pseudo attribut), alors que le second est plutôt documentaire. La plupart du temps, le nommage des rôles se limite en fonction de la navigabilité de l'association. Lorsque deux classes sont reliées par une seule association, le nom des classes suffit souvent à caractériser le rôle ; le nommage des rôles prend tout son intérêt en présence de relations réflexives ou lorsque plusieurs associations relient deux classes. Dans ce dernier cas, il n'y a aucune corrélation par défaut entre les objets qui participent à ces relations. Chaque association exprime un concept distinct. Dans l'exemple suivant, il n'y a pas de relation entre les passagers et le pilote.

Figure 3–73.
Exemple d'associations multiples entre un avion et des personnes.



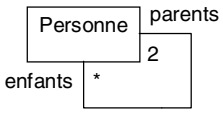
La présence d'un grand nombre d'associations entre deux classes peut être suspecte. Chaque association augmente le couplage entre les classes associées ; or, un fort couplage peut être le signe d'une mauvaise décomposition. Il est également fréquent que les débutants représentent plusieurs fois la même association en la nommant avec le nom d'un des messages qui circulent entre les objets instances des classes associées.

Figure 3–74.
Exemple de confusion entre associations et messages. Une seule association est suffisante pour permettre à une personne de conduire, démarrer, laver et arrêter sa voiture.



Les associations peuvent également relier une classe à elle-même, comme dans le cas des structures récursives. Ce type d'association est appelé association réflexive. Une telle configuration signifie qu'une instance d'une classe est liée à une autre instance de cette classe. Les rôles prennent à nouveau toute leur importance car ils permettent de distinguer les instances qui participent à la relation. L'exemple suivant montre la classe des personnes et la relation qui unit les parents et leurs enfants.

Figure 3–75.
Toute personne possède deux parents, et de zéro à plusieurs enfants. Le rôle de chaque extrémité d'association est essentiel à la clarté du diagramme.

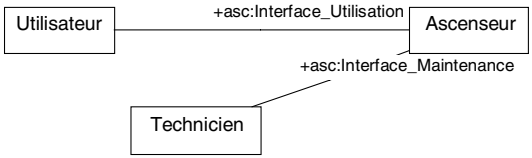


Typage des rôles

Typer un rôle d'une association consiste à définir l'interface que réalise le rôle (une classe peut réaliser plusieurs interfaces). Ainsi le typage des rôles est également appelé spécification d'interface. En général, la classe du côté du rôle fournit un surensemble de services par rapport à ce que l'association nécessite. Le typage permet alors de restreindre ce sur-ensemble. Sans spécification du type de rôle, l'ensemble des fonctionnalités de la classe est mis à la disposition de l'association. Le typage d'un rôle respecte la syntaxe suivante :

```
Nom_du_rôle ':' Nom_de_l'interface
```

Figure 3–76.
Exemple de typage du rôle **`asc`**.



Multiplicité des associations

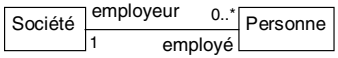
Chaque extrémité d'une association peut porter une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. La multiplicité est une information portée par l'extrémité d'association, sous la forme d'une expression entière.

Figure 3–77.
Valeurs de multiplicité conventionnelles.

1	Un et un seul
0..1	Zéro ou un
N	N (entier naturel)
M .. N	De M à N (entiers naturels)
*****	De zéro à plusieurs
0 .. *	De zéro à plusieurs
1 .. *	D'un à plusieurs

L'exemple suivant rend compte du fait que plusieurs personnes travaillent pour une même société.

Figure 3–78.
Ce modèle supprime le chômage ! Chaque personne travaille pour une société, chaque société emploie de zéro à plusieurs personnes.



La multiplicité de valeur **`1`** du côté de la classe **`Société`** n'est pas très réaliste car elle signifie que toutes les personnes ont un emploi. La multiplicité de valeur **`0..*`** du côté de la classe **`Personne`** signifie qu'une société emploie de zéro à plusieurs personnes. Une valeur de multiplicité supérieure à **`1`** implique une

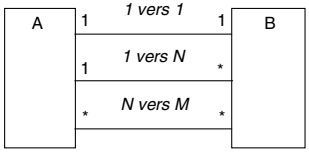
collection d'objets. Cette collection n'est pas bornée dans le cas d'une valeur ***** ; en d'autres termes, plusieurs objets participent à la relation, le nombre maximum d'objets n'étant pas fixé. Le terme *collection*, plus général que liste ou ensemble, est employé afin d'éviter toute supposition sur la structure de données qui contient les objets.

Les valeurs de multiplicité expriment des contraintes liées au domaine de l'application, valables durant toute la vie des objets. Le respect des valeurs de multiplicité peut être assuré statiquement ou dynamiquement. Dans le cas dynamique, il ne faut pas considérer les multiplicités durant les régimes transitoires, comme lors de la création ou de la destruction des objets. Une multiplicité de valeur **1** indique qu'en régime permanent, un objet possède obligatoirement un lien vers un autre objet ; pour autant, il ne faut pas essayer d'en déduire le profil de paramètres des constructeurs et systématiquement prévoir un paramètre de la classe de l'objet lié. Les valeurs de multiplicité n'impliquent rien de particulier sur l'ordre de création des objets. L'exemple suivant montre deux classes connectées par une association de multiplicité **1** vers **1**, sans rien supposer sur le profil de paramètres des constructeurs d'objets. Dans l'exemple suivant, il n'est pas nécessaire de disposer d'une voiture pour construire un moteur et inversement.

Figure 3–79.
La multiplicité d'une association est le reflet d'une contrainte du domaine.

Il est important de déterminer les valeurs de multiplicité optimales pour trouver le bon équilibre, d'une part, entre souplesse et possibilité d'extension, et d'autre part, entre complexité et efficacité. En analyse, seule la valeur de la multiplicité est réellement importante. En revanche, en conception, il faut choisir des structures de données (pile, file, ensemble...) pour réaliser les collections qui correspondent aux multiplicités de type **1..*** ou **0..***. La surestimation des valeurs de multiplicité induit un surcoût en taille de stockage et en vitesse de recherche. De même, une multiplicité qui commence à zéro implique que les diverses opérations doivent contenir du code pour tester la présence ou l'absence de liens dans les objets. Les valeurs de multiplicité sont souvent employées pour décrire de manière générique les associations. Les formes les plus courantes sont les associations **1** vers **1**, **1** vers **N** et **N** vers **M**, représentées sur la figure suivante.

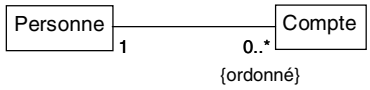
Figure 3–80.
Description générique des associations selon les valeurs de multiplicité.



Contraintes sur les associations

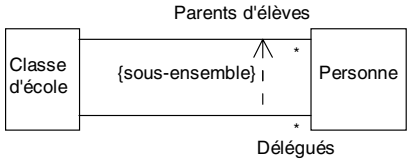
Toutes sortes de contraintes peuvent être définies sur une relation ou sur un groupe de relations. La multiplicité présentée dans le paragraphe précédent est une contrainte sur le nombre de liens qui existent éventuellement entre deux objets. Les contraintes se représentent dans les diagrammes par des expressions placées entre accolades. La contrainte **{ordonné}** peut être placée sur le rôle pour spécifier qu'une relation d'ordre décrit les objets de la collection ; dans ce cas, le modèle ne spécifie pas comment les éléments sont ordonnés, mais seulement que l'ordre doit être maintenu durant l'ajout et ou la suppression des objets par exemple.

Figure 3–81.
La contrainte **{ordonné}** *indique que la collection des comptes d'une personne est ordonnée.*



La contrainte **{sous-ensemble}** indique qu'une collection est incluse dans une autre collection. La contrainte est placée à proximité d'une relation de dépendance entre deux associations (la flèche de la relation de dépendance indique le sens de la contrainte). L'exemple suivant montre que les délégués de parents d'élèves sont des parents d'élèves.

Figure 3–82.
La contrainte **{sous-ensemble}** *indique que les objets qui participent à la relation* **Délégués** *participent également à la relation* **Parents d'élèves**.



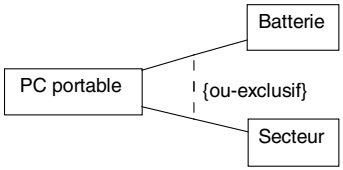
Cette contrainte peut également être exprimée avec le langage de contraintes OCL (voir la section *OCL*) :

```
context Classe d'école inv :
   self.Parents d'élèves→includesAll (self.Délégués)
```

Avec OCL, il est ainsi possible de préciser que, *pour chaque instance* **de Classe d'école**, les personnes qui jouent le rôle de **Délégués** font aussi partie des personnes qui jouent le rôle de **Parents d'élèves**. La contrainte **{ou-exclusif}** (**{xor}** est également utilisée) indique que, pour un objet donné, une seule association parmi un groupe d'associations est valide.

Cette contrainte évite l'introduction de sous-classes artificielles pour matérialiser l'exclusivité. Dans le diagramme de classes, cette contrainte est placée à proximité d'un trait en pointillés qui relie les associations (au minimum deux) dont les instances sont mutuellement exclusives.

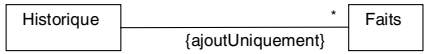
Figure 3–83.
Introduction d'une contrainte **{ou-exlusif}** *pour distinguer deux associations mutuellement exclusives.*



La contrainte **{ou-exclusif}** permet d'éviter l'introduction de sous-classes artificielles comme c'est le cas dans les constructions à base de généralisation. Une extrémité d'association possède une propriété **modification**. Cette propriété peut être précisée dans un modèle en utilisant la notation des valeurs marquées (le nom de la propriété est parfois omis et seule la valeur est alors notée). Les valeurs que peut prendre cette propriété sont :

- **variable** : l'instance de la classe à cette extrémité d'association peut être modifiée. Il s'agit de la valeur par défaut.
- **gelé** : l'instance de la classe à cette extrémité d'association est non modifiable.
- **ajoutUniquement** : il est possible d'ajouter des instances et non de les retirer (pour une multiplicité supérieure à 1).

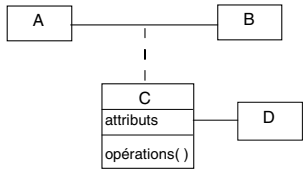
Figure 3–84.
Précision de la propriété de modification d'une extrémité d'association.



Les classes-associations

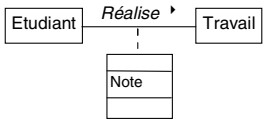
Il est possible de représenter une association par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association. Une classe de ce type, appelée classe associative ou classe-association, possède à la fois les caractéristiques d'une classe et d'une association, et peut à ce titre participer à d'autres relations dans le modèle. La notation utilise une ligne pointillée pour attacher une classe à une association. Dans l'exemple suivant, l'association entre les classes **A** et **B** est représentée par la classe **C**, elle-même associée à la classe **D**.

Figure 3–85.
Exemple d'une classe-association.



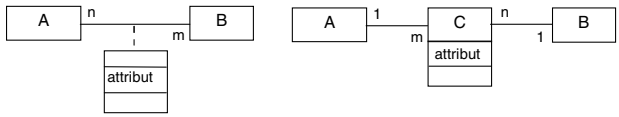
Une association qui contient des attributs sans participer à des relations avec d'autres classes est appelée association attribuée. Dans ce cas, la classe attachée à l'association ne porte pas de nom spécifique.

Figure 3–86.
Représentation d'une association attribuée.



En conception, une classe-association peut être remplacée par une classe intermédiaire et qui sert de pivot pour une paire d'associations.

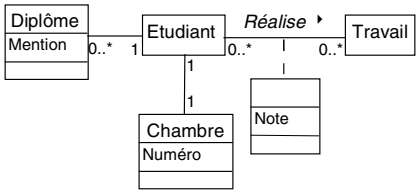
Figure 3–87.
Transformation d'une classe-association en classes et en associations.



Placement des attributs selon les valeurs de multiplicité

La réification des associations prend tout son intérêt pour les associations **N** vers **M**. Pour les associations **1** vers **1**, les attributs de l'association peuvent toujours être déplacés dans une des classes qui participent à l'association. Pour les associations **1** vers **N**, le déplacement est généralement possible vers la classe du côté **N** ; toutefois, il est fréquent de promouvoir l'association au rang de classe pour augmenter la lisibilité ou en raison de la présence d'associations vers d'autres classes. L'exemple suivant illustre les différentes situations.

Figure 3–88.
Exemples de placement des attributs selon les valeurs de multiplicité.



L'association entre la classe **Etudiant** et la classe **Travail** est de type **N** vers **M**. La classe **Travail** décrit le sujet, la solution apportée par l'étudiant n'est pas conservée.

Dans le cas des contrôles de connaissances, chaque étudiant compose individuellement sur un travail donné et la note obtenue ne peut être stockée ni *dans* un étudiant en particulier (car il effectue de nombreux travaux dans sa carrière), ni *dans* un travail donné (car il y a autant de notes que d'étudiants).

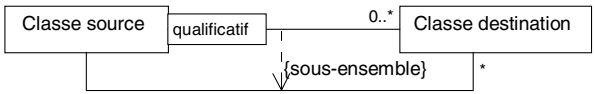
La note est un attribut de la relation entre la classe des étudiants et la classe des travaux. En fin d'année, chaque étudiant reçoit un diplôme avec une mention qui dépend de sa performance individuelle.

La relation entre le diplôme et l'étudiant est personnalisée car un diplôme ne concerne qu'un étudiant donné. La mention devient un attribut du diplôme. La mention n'est pas stockée *dans* l'étudiant pour deux raisons : elle ne qualifie pas un étudiant et un étudiant peut obtenir plusieurs diplômes. Chaque étudiant possède une chambre et une chambre n'est pas partagée par plusieurs étudiants. L'association entre les étudiants et les chambres est du type **`1`** vers **`1`**. Le numéro est un attribut de la classe **`Chambre`** puisqu'un numéro caractérise une chambre.

Qualification des associations

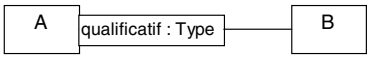
La qualification d'une association, aussi dénommée restriction d'une association, consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association. La restriction est réalisée au moyen d'un tuple d'attributs particuliers, appelé qualificatif ou clé, qui est utilisé conjointement avec un objet de la classe source.

Figure 3–89.
Représentation des associations qualifiées.



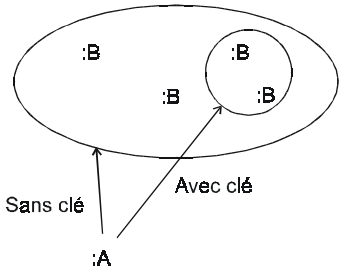
Le qualificatif est placé sur l'extrémité d'association au niveau de la classe source, dans un compartiment rectangulaire. Le qualificatif appartient pleinement à l'association et non aux classes associées.

Figure 3–90.
Qualification d'une association au moyen d'un attribut particulier appelé qualificatif ou clé (le type du qualificatif peut être précisé).



L'instanciation d'une association qualifiée définit le nom des objets source et destination, et la valeur du qualificatif. Ainsi, chaque instance de la classe **`A`**, accompagnée de la valeur du qualificatif, identifie un sous-ensemble des instances de **`B`** qui participent à l'association. La qualification partitionne l'ensemble d'arrivée et réduit ainsi la multiplicité de l'association. La paire **`(instance de A, valeur du qualificatif)`** identifie un sous-ensemble des instances de **`B`**.

Figure 3–91.
Une restriction réduit le nombre d'instances qui participent à une association.



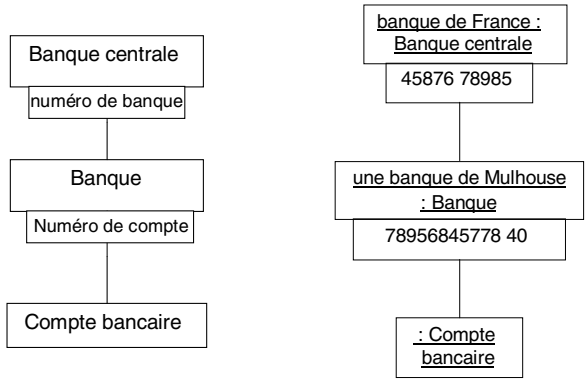


Figure 3–92.
L'instanciation d'une association qualifiée requiert une valeur de qualification.

La restriction d'une association peut être opérée en combinant les valeurs des différents attributs qui forment le qualificatif.

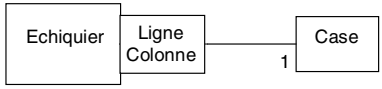


Figure 3–93.
Combinaison d'une ligne et d'une colonne pour identifier une case de l'échiquier.

Bien que l'identifiant d'objets semble trouver sa place tout naturellement dans un attribut de classe, une association qualifiée est souvent une approche plus judicieuse.

L'identifiant désigne un objet parmi un ensemble d'instances d'une même classe. Les associations qualifiées se prêtent bien à cette identification lorsqu'elles conduisent à une unique instance de la classe destination.

Une adresse de courrier électronique est un exemple d'identifiant. Pour un réseau donné (l'Internet par exemple), toute personne peut être identifiée de manière unique avec son adresse électronique.

Une même personne peut également posséder plusieurs adresses électroniques (professionnelles et personnelles). L'adresse électronique n'est ni un attribut de la personne ni du réseau (nul besoin d'avoir une adresse électronique sans être connecté à un réseau et inversement) mais appartient à l'association.

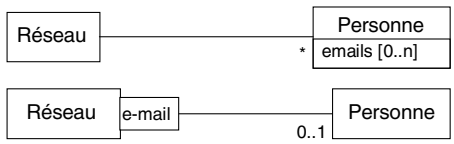


Figure 3–94.
Une association qualifiée est plus adaptée à modéliser un identifiant qu'un attribut de classe.

Dans l'exemple précédent, la qualification de l'association conduit à une multiplicité de 0 ou 1, ce qui facilite la navigation de la classe **`Réseau`** vers la classe **`Personne`** et rend inutile la déclaration d'un attribut **`e-mails`** de multiplicité 0 ou plusieurs dans la classe **`Personne`**. Avec la qualification, chaque association entre une instance de **`Réseau`** et une instance de **`Personne`** implique l'existence d'au moins une adresse électronique pour l'instance de **`Personne`**.

L'idéal, pour modéliser les identifiants via une association qualifiée, est de trouver un objet qui modélise le *contexte* dans lequel l'identifiant est unique. Dans le contexte d'une rue, un numéro de maison est unique ; dans le contexte d'un hôtel, les numéros de chambres sont uniques.

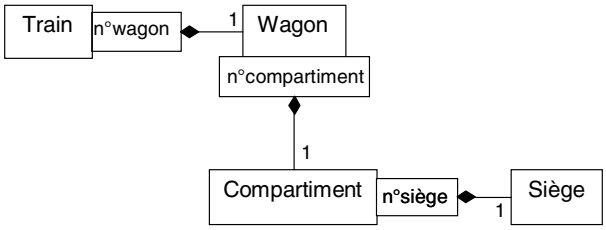


Figure 3–95.
Un même numéro de siège peut exister dans différents compartiments ; mais dans le contexte d'un compartiment donné, il est unique ; le compartiment est unique dans le contexte du wagon et le wagon l'est aussi dans le contexte du train.

Cette construction peut se réaliser par des clés composites dans les bases de données relationnelles. Les numéros d'identification (numéro de Sécurité sociale, numéro d'immatriculation de voiture...) et autres identifiants codés sont souvent des qualificatifs associés à un contexte. En général, la multiplicité du côté de la classe destination est de 0 ou 1.

L'association qualifiée peut également être utile lorsqu'elle restreint l'ensemble des instances de la classe destination à une cardinalité supérieure à 1. Dans ce cas, le qualificatif sert d'index pour partitionner l'ensemble des instances de la classe destination. L'association qualifiée peut se transformer en classe-association puisque le qualificatif est un attribut de l'association et non un attribut d'une des deux classes participant à l'association. Une association qualifiée est toutefois plus précise qu'une classe-association puisqu'elle détermine un contexte et permet de modéliser une restriction de l'ensemble des instances de la classe destination.

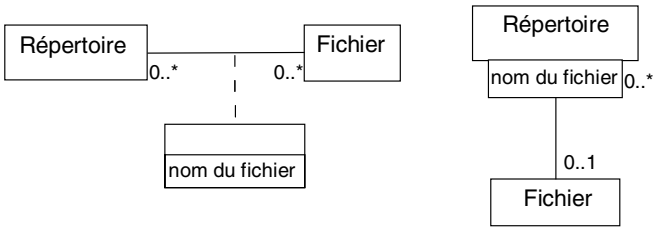
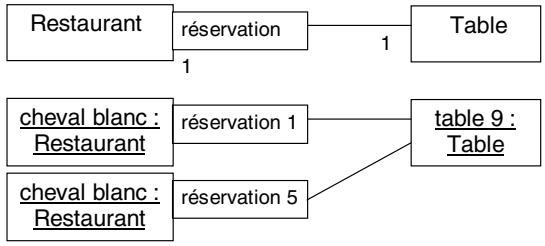


Figure 3–96.
Classe-association ou association qualifiée.

Avec une classe-association, un lien entre deux objets ne peut avoir qu'une valeur pour chaque attribut de la classe-association (les instances d'une association sont des tuples ; sans doublons) ; avec une qualification il peut y avoir plusieurs valeurs de qualification pour une association d'objets donnée. Par exemple, si la modélisation des fichiers contenus dans un répertoire est faite à l'aide d'une association qualifiée, il est possible de définir plusieurs noms de fichier pour un même fichier de ce répertoire ; avec une classe-association, cela est impossible.

La qualification permet également de modéliser de manière compacte des situations qui requièrent une sémantique de sac à provision, à partir des relations qui imposent une sémantique d'ensemble. Par exemple, des réservations d'une même table dans un restaurant donné :

Figure 3–97.
Représentation des réservations d'une table dans un restaurant donné.

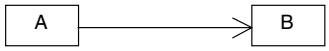


La navigabilité

Les associations décrivent le réseau de relations structurelles qui existent entre les classes et qui donnent naissance aux liens entre les objets instances de ces classes. Les liens peuvent être vus comme des canaux de navigation entre les objets. Ces canaux permettent de se déplacer dans le modèle et de réaliser les formes de collaboration qui correspondent aux différents scénarios.

Par défaut, les associations sont navigables dans les deux directions. Dans certains cas, une seule direction de navigation est utile ; l'extrémité d'association vers laquelle la navigation est possible porte une flèche. L'absence de flèche signifie que l'association est navigable dans les deux sens. Dans l'exemple suivant, les objets instances de **A** voient les objets instances de **B**, mais les objets instances de **B** ne voient pas les objets instances de **A**.

Figure 3–98.
L'association entre les classes **A** *et* **B** *est uniquement navigable dans le sens* **A** *vers* **B**.



Une association navigable uniquement dans un sens peut être vue comme une demi-association. Cette distinction est souvent réalisée pendant la conception, mais peut très bien apparaître en analyse, lorsque l'étude du domaine révèle une

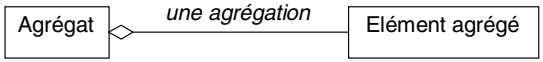
dissymétrie des besoins de communication. Il peut être intéressant de réduire la navigabilité afin de diminuer les couplages et les dépendances entre classes ; surtout si ces classes sont situées dans des paquetages différents puisqu'il faudrait alors mettre en place des visibilités mutuelles entre ces paquetages.

Les agrégations

Une agrégation représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité. Quelle que soit l'arité, l'agrégation ne peut concerner qu'un seul rôle d'une association.

L'agrégation se représente en ajoutant un petit losange du côté de l'agrégat.

Figure 3–99.
Représentation d'une agrégation.



L'agrégation permet de modéliser une contrainte d'intégrité et de désigner l'agrégat comme garant de cette contrainte.

À travers une telle contrainte d'intégrité, il est possible de représenter par exemple :

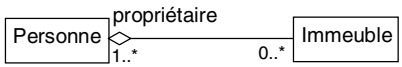
- la propagation des valeurs d'attributs d'une classe vers une autre classe ;
- une action sur une classe qui implique une action sur une autre classe, comme la copie profonde ;
- une subordination des objets d'une classe à ceux d'une autre.

Les agrégations peuvent être multiples, comme les associations. Tant qu'aucun choix de réalisation n'est effectué, il n'y a aucune contrainte particulière sur les valeurs de multiplicité que peuvent porter les extrémités d'une agrégation.

Cela signifie, en particulier, que la multiplicité du côté de l'agrégat peut être supérieure à 1. Ce type d'agrégation correspond par exemple au concept de copropriétaire.

Le diagramme suivant montre que des personnes peuvent être copropriétaires des mêmes immeubles sur lesquels elles possèdent des droits conjoints.

Figure 3–100.
Exemple d'agrégat multiple.



La notion d'agrégation ne suppose aucune forme de réalisation particulière.