11. Les associations sous la loupe

Après avoir, dans les deux chapitres précédents, présenté la méthode que nous préconisons pour analyser les données et concevoir un modèle de la base, nous allons ici étudier plus en détail la spécification des associations. Cette, occasion nous permettra, comme dans les chapitres suivants, de traiter d'autres exemples et de passer en revue certains problèmes qui se posent constamment dans le contexte de l'activité qui nous concerne ici.

Examen exhaustif des associations possibles

Lorsqu'on établit un modèle de données, il vaut la peine d'examiner en détail tous les types d'objet découverts et se poser la question du sens éventuel d'associations entre toutes les paires de types d'objet recensés, même si ces associations ne découlent pas directement de l'inventaire des données.

Nous allons voir en outre qu'il est quelques fois nécessaire d'étudier aussi des associations entre objets d'un même type ainsi que des associations entre objets de trois types ou même davantage.

Il est pourtant clair qu'on ne s'amusera pas à garder dans le modèle des associations auxquelles ont ne peut pas rattacher un sens, des attributs ou une utilité pour le système envisagé. Le but de l'examen préconisé ici est d'assurer que rien n'a été oublié. Souvent on découvre en effet des aspects importants du problème ou de sa solution en se posant la question: "quel serait le sens possible d'une association entre type A et type B". Nous en donnons quelques exemples dans le cours du chapitre.

Cardinalités d'une association

Nous distinguons les types d'associations suivants en fonction de leurs cardinalités:

Associations de type *un à un* (reliant un objet exactement à un autre)

Associations de type un à plusieurs (reliant un objet à plusieurs autres)

Associations de type plusieurs à plusieurs (reliant plusieurs objets à plusieurs autres)

Des exemples de chacun de ces cas sont présentés ci-dessous.

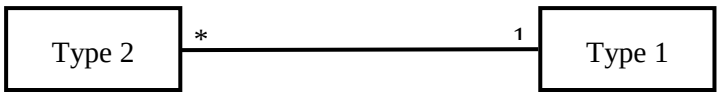
Précisons tout de suite que, dans le modèle relationnel pur, on ne décrit que des associations de type *un à plusieurs*.

Les associations de type *un à un* laissent le concepteur dans la situation étrange de ne pas disposer de solution définie, comme nous allons le voir.

Les associations de type *plusieurs à plusieurs* entraînent à l'implémentation une violation de la 1ère forme normale, nous l'avons déjà vu. Nous avons mentionné le fait que de telles violations sont aujourd'hui admises, mais déconseillées, sauf peut être pour des cas extrêmement simples.

Dans cet ouvrage (et dans la pratique), **dès qu'on est confronté à une association de type *plusieurs à plusieurs*, on crée un nouveau type d'objet associatif qui décompose cette association en deux associations de type *un à plusieurs*.**

Associations de type *un à plusieurs*



Chaque objet *type 2* est associé obligatoirement à un et un seul objet *type 1*. Chaque objet *type 1* peut être associé à 0, 1 ou plusieurs objets *type 2*.

Type 1 (#-clé1, …

Type 2 (#-clé2, …, *#-clé1*, …

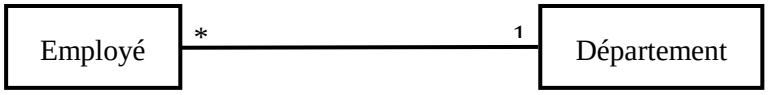
Une association de type *un à plusieurs* se spécifie en ajoutant la clé primaire du côté "un" comme attribut clé étrangère du côté "plusieurs" de l'association.

Ici également on pourrait définir un type d'objet associatif. Il n'est pas nécessaire. Les attributs éventuels propres à l'association des deux objets sont stockés du côté plusieurs.

Exemple

Dans une entreprise, nous devons décrire à quel département sont rattachés les employés, ceci avec leur date d'entrée dans ce département.

On nous certifie qu'un employé n'est rattaché qu'à un département. Par contre un département contient évidemment plusieurs employés.



Appliquant les règles ci-dessus, nous écrivons:

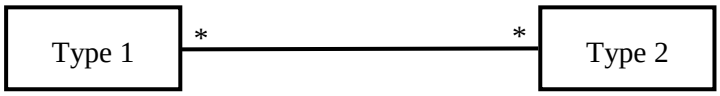
Département (#-dept, nom dept, …

Employé (#-emp, nom, prénom, … , *#-dept*, date entrée, …)

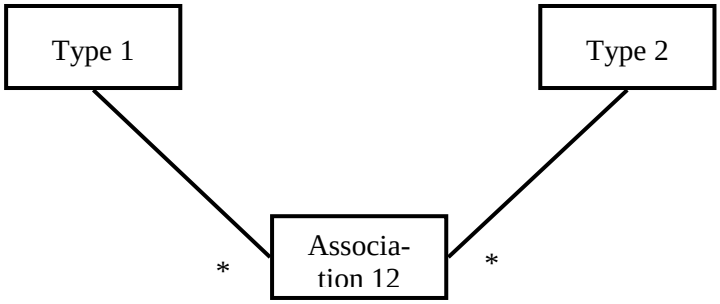
Le département apparaît comme clé étrangère dans l'employé et l'attribut qui lie l'employé au département (la date d'entrée) figure également dans l'employé.

Cette solution empêcherait par exemple un employé d'occuper deux demi-postes dans deux départements. Mais c'était bien l'hypothèse de départ.

Associations de type *plusieurs à plusieurs*



Règle: Dans un tel cas, il faut créer un type d'objet associatif (de liaison) et l'associer par des associations de type *un à plusieurs* aux deux types originaux.



Type 1 (#-type 1, …

Type 2 (#-type 2, …

Association 12 (clé, … , *#-type 1*, *#-type 2*, …

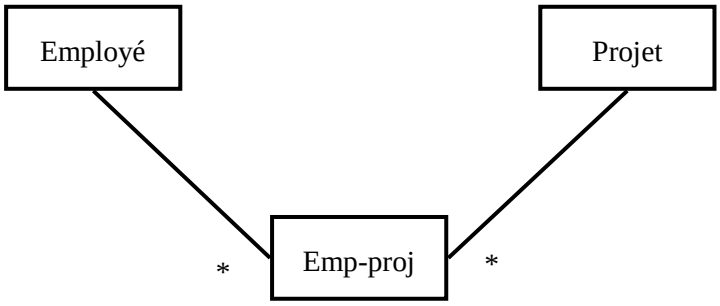
Conformément à notre règle concernant les associations de type *un à plusieurs*, les clés primaires des deux types originaux apparaissent comme clés étrangères dans le type associatif. À chaque objet *type 1* ou *type 2* peuvent être associés zéro, un ou plusieurs objets *association 12*, mais chaque objet *association 12* est associé exactement à un objet *type 1* et à un objet *type 2*.

Que choisir comme clé d'un type associatif? Plusieurs possibilités sont à notre disposition:

- Un nouvel attribut totalement neutre
- La concaténation des clés des deux types associés, si cette valeur est unique par définition
- La concaténation des clés des deux types associés plus un autre attribut, la date par exemple, pour rendre cette clé unique.

Comme toujours, il faut bien sûr contrôler l'absence de violations d'une des formes normales. Et si l'on choisit comme clé primaire un regroupement d'attributs, vérifier également l'absence de violations de la 2ème forme normale (dépendances partielles, donc d'une partie de la clé, voir chapitre 6).

Exemple



Un employé peut travailler à plusieurs projets et un projet est traité par plusieurs employés. Il s'agit d'enregistrer chaque jour le temps que les employés consacrent aux projets.

Employé (#-employé, nom, prénom, sexe, date de naissance, …

Projet (#-projet, nom projet, …

Emp-proj (*#-employé*, *#-projet*, date, temps, …

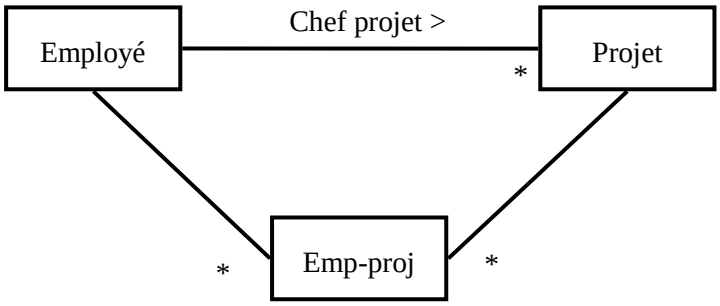
Le regroupement *#-employé*, *#-projet* n'est pas unique puisqu'un même employé peut travailler à un même projet à des dates différentes. Si on rajoute la date, le regroupement devient unique.

Nous avons donné au type associatif le nom *emp-proj*, concaténant les deux noms. Rien ne nous empêche de donner un nom plus utile à ce type d'objet, *heures* ou *rapport*, par exemple. Rien ne nous empêche non plus de donner à ce type un attribut neutre comme clé primaire.

Rapport (#-rapport, *#-employé*, *#-projet*, date, temps, …

Des violations de formes normales? Ce serait le cas si *nom employé* ou *nom projet* apparaissaient dans *rapport*, quel que soit le choix de la clé primaire. Si l'on choisit la clé concaténée, ils ne dépendraient que d'une partie de la clé primaire (violation de la 2ème FN) et si l'on choisit la solution présentée ils dépendraient d'un attribut n'étant pas la clé.

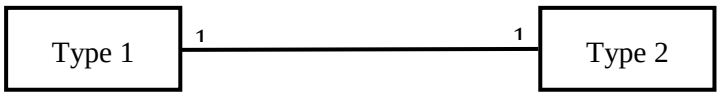
Nous avons mentionné au début du chapitre la nécessité de se pencher sur toutes les associations possibles. Une association directe entre *employé* et *projet* aurait elle un sens? Oui, c'est bien sûr la façon de décrire qu'un projet possède un chef qui est évidemment un employé.



Projet (#-projet, nom projet, *#-employé chef*, …

Associations de type *un à un*

Nous l'avons déjà dit: il n'existe pas vraiment de solution pour spécifier que deux objets sont exclusivement liés l'un à l'autre. On peut bien sûr le représenter:



Mais la seule façon de l'imposer serait de faire figurer dans la spécification de chacun des deux types la clé primaire de l'autre comme clé étrangère.

Type 1(#-clé 1, …, *#-clé 2*, …

Type 2(#-clé 2, …, *#-clé 1*, …

Cette solution a pour désavantage qu'on ne peut pas créer le premier objet d'une paire en une seule opération, puisque son partenaire n'existe pas encore dans le système. Il faut donc laisser la clé étrangère vide, saisir le deuxième objet avec, dans la clé étrangère, la valeur de la clé primaire du premier, puis revenir compléter le premier objet. Nous n'avons jamais vu d'implémentation d'une telle solution dans la pratique.

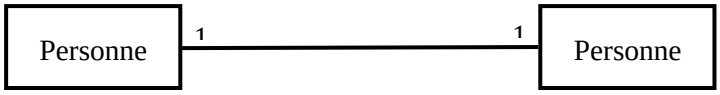
À part la description de couples mariés (dans une société monogame, évidemment!), le besoin de décrire de telles associations n'existe à notre avis pas vraiment. Appliquée à cet exemple, la solution présentée ci-dessus ferait figurer chez l'homme le numéro de personne de la femme et chez la femme le numéro de personne de l'homme. Nous reviendrons plus bas sur la description possible de couples.

Au chapitre 10, nous avons pourtant eu un exemple d'association de type *un à un* et sommes restés sur une solution insatisfaisante: l'entreprise produit exactement une facture pour chaque livraison. Il est évidemment possible de programmer des règles pour empêcher la saisie de plusieurs factures sur un seul bulletin de livraison, mais la structure de la base de données ne peut en elle-même l'empêcher.

Associations entre objets d'un même type

Rien n'empêche de définir des associations entre objets d'un même type. C'est même une situation assez courante. Et l'association peut aussi bien être de type *un à un*, *un à plusieurs* ou *plusieurs à plusieurs*. Tout ce que nous avons dit précédemment dans ce chapitre concernant la manière de traiter ces différents types d'associations reste évidemment valable. Nous donnons un exemple pour chaque cas.

Pour l'association de type *un à un*, nous reprenons celui de l'union monogame.

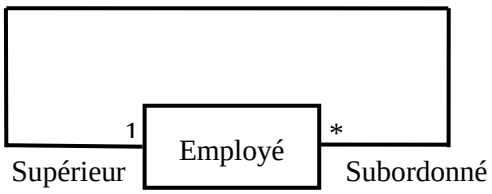


Personne (#-personne, nom, prénom, sexe, date naissance, *#-personne conjoint*)

Nous avons déjà donné notre opinion au sujet de cette solution, qui ne permet en outre pas de faire figurer proprement des attributs de la relation tels que la date du mariage.

Nous présenterons une meilleure solution plus loin.

Comme exemple d'association de type *un à plusieurs* d'objets d'un même type, nous présentons la description d'une hiérarchie, ici de personnes.



Employé (#-employé, nom, prénom, …, *#-employé supérieur*, …

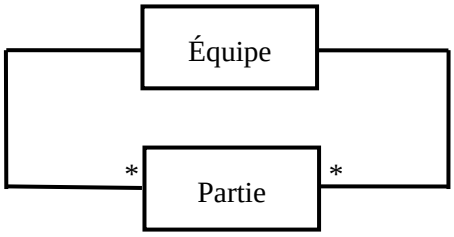
Chaque employé contient ainsi la référence vers son supérieur, chez qui on trouve la référence vers son supérieur à lui, etc. Le seul employé chez qui cette clé étrangère est vide est évidemment le PDG de la société.

Notons qu'une même solution pourrait s'appliquer aux hiérarchies de postes dans une entreprise, à des hiérarchies de catégories d'articles ou de régions.

Ajoutons toutefois que des hiérarchies à plusieurs niveaux sont difficiles à gérer en pratique. Il faut par exemple empêcher la saisie de boucles (indiquer par exemple comme supérieur de A une personne qui est subordonnée à B, lui-même subordonné de A).

Nous présentons quatre exemples d'associations de type *plusieurs à plusieurs* entre objets d'un même type, prouvant que de telles associations existent bien en pratique.

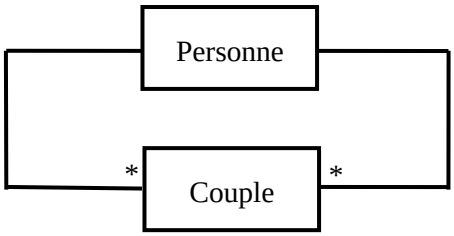
Premier exemple: parties (matchs)



Équipe (#-équipe, nom équipe, …

Partie (#-partie, *#-équipe 1, #-équipe 2,* date, stade, résultat

Second exemple: couples



Personne (#-personne, nom, prénom, sexe, date naissance, date décès, …

Couple (#-union, *#-personne 1*, *#-personne 2*, date début, date fin, …

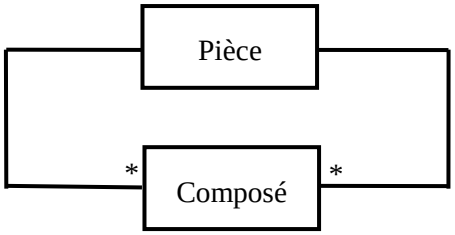
Cette fois la monogamie n'est plus imposée par le modèle! Plus pratique, la solution nous donne la possibilité de décrire des mariages successifs pour les personnes divorcées ou veuves. Et les attributs de l'association trouvent leur place dans le type associatif.

Troisième exemple: liste de pièces

En fabrication, on parle de liste de pièces ou de nomenclature pour décrire les composants d'un article. En général, une liste de pièces comprend plusieurs niveaux (les composants sont eux-mêmes des composés, etc.). En accédant à la liste de pièces de haut en bas on trouve les composants d'un composé. En y accédant de bas en haut, on trouve les composés dans lesquels ces composants dans lesquels ils interviennent.

On peut alors poser des questions du type:

- Combien de vis d'un certain modèle sont utilisées dans certaine machine (par exemple si on veut en construire 100)?
- Dans quelles machines une vis d'un certain type est-elle utilisée?



Pièce (#-pièce, désignation, prix de revient

Composé (*#-pièce composé*, *#-pièce composant*, quantité, …

Pièce comprend tout: le produit fini, les composants intermédiaires à tout niveau et les pièces les plus élémentaires (exemple: l'Airbus complet, le réacteur, la dernière vis).

En accédant à *composé* par la première partie de la clé primaire on obtient les composants (décomposition), en accédant par la seconde partie de la clé primaire, on obtient les composés comprenant un certain composant (utilisation).

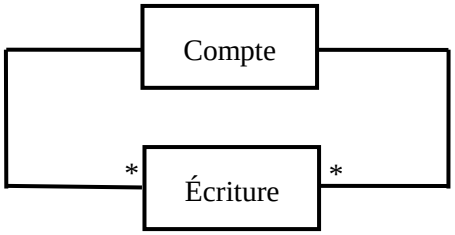
En voici une illustration (les numéros de pièces ont été remplacés par les désignations et la description des vélos est arbitraire)

Composé	Composant	Quantité
Vélo sport	Cadre sport	1
Vélo ville homme	Cadre ville homme	1
Vélo ville femme	Cadre ville femme	1
VTT	Cadre VTT	1
Vélo sport	Roue sport	2
Vélo ville homme	Roue ville	2
Roue ville	Jante ville	1
Roue ville	Rayon	24
Roue ville	Moyeu	1

Contrairement à l'exemple des employés présenté plus haut (supérieur – subordonné), il ne s'agit dans ce cas pas d'une hiérarchie, mais d'un réseau, puisqu'un composant peut être utilisé dans plusieurs composés.

Quatrième exemple: comptabilité

Voici une façon originale de représenter une comptabilité dans une base de données:



Compte (#-compte, désignation)

Écriture (#-écriture, *#-compte débit*, *#-compte crédit*, date, montant)

Avantages: une écriture par transaction (débit/crédit dans la même écriture), on voit immédiatement de quel compte à quel compte a lieu la transaction. L'équilibre des transactions (égalité débit/crédit) est assuré par définition. Les extraits de compte s'obtiennent en accédant par les clés étrangères.

Désavantages: pas de contreparties multiples, pas de notion de pièce comptable.

Des essais pour tester réellement ce système ont paraît-il été effectués dans une banque. Mais la solution nous paraît surtout utile pour décrire par exemple des mouvements de stock.

Mouvement de stock (#-transaction, *#-article*, quantité, date, *de stock*, *à stock*)

Exemple: 100 *vis M3* de *Magasin* à *Montage*.

Terminons cet exemple en montrant tout de même la structure des données d'une comptabilité traditionnelle:

Compte (#-compte, désignation)

Pièce comptable (#-pièce, date)

Écriture (#-écriture, *#-pièce*, code crédit/débit, montant)

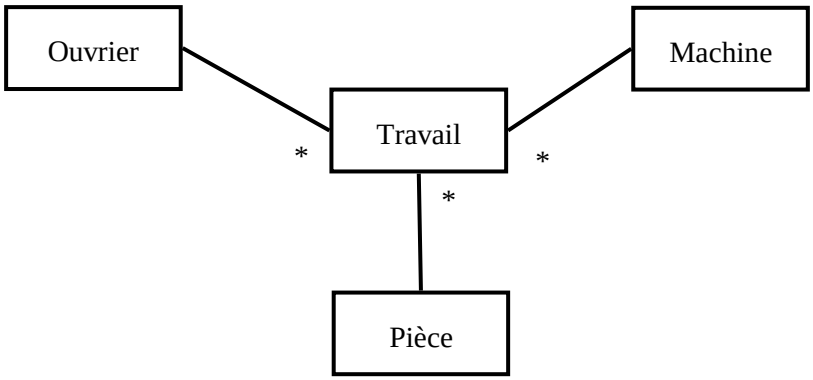
Au programmeur pourtant de vérifier que crédits et débits sont équilibrés au sein de la pièce!

Associations entre trois types d'objets

Rien n'empêche de définir des types d'objets associatifs reliant trois types d'objets ou davantage.

Passons immédiatement à un exemple, à nouveau tiré de la fabrication.

On désire ici enregistrer les travaux exécutés par des ouvriers travaillent à des machines pour produire des pièces.



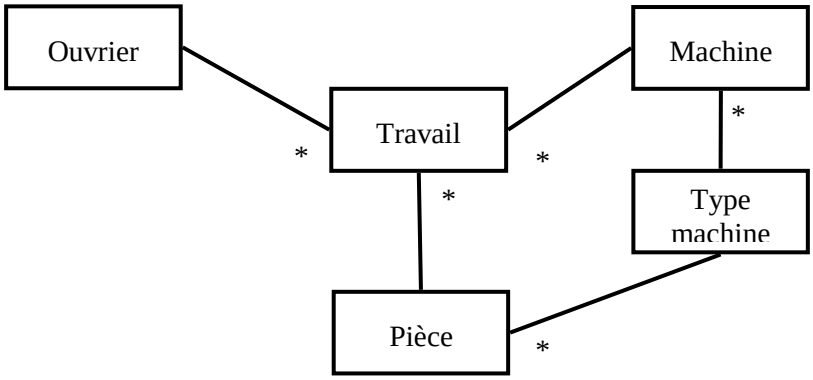
Ouvrier (#-ouvrier, nom, prénom, …

Machine (#-machine, …

Pièce (#-pièce, libellé, …

Travail (#-travail, *#-ouvrier*, *#-pièce*, *#-machine*, date, temps nécessaire, …

Étendons cette solution. Une association entre machine et pièce aurait-elle un sens? Oui, si nous admettons que l'usine se serve de différents types de machines et que chaque pièce doive être usinée sur un type spécifique de machine.



Ouvrier (#-ouvrier, nom, prénom, …

Type machine (#-type machine, …

Machine (#-machine, *#-type machine*, …

Pièce (#-pièce, libellé, *#-type machine*, …

Travail (#-travail, *#-ouvrier*, *#-pièce*, *#-machine*, date, temps nécessaire, …

Nous laissons au lecteur le soin de développer encore cet exercice en décrivant par exemple les certifications des ouvriers sur certains types de machines.

Violations de 1ère forme normale, oui ou non?

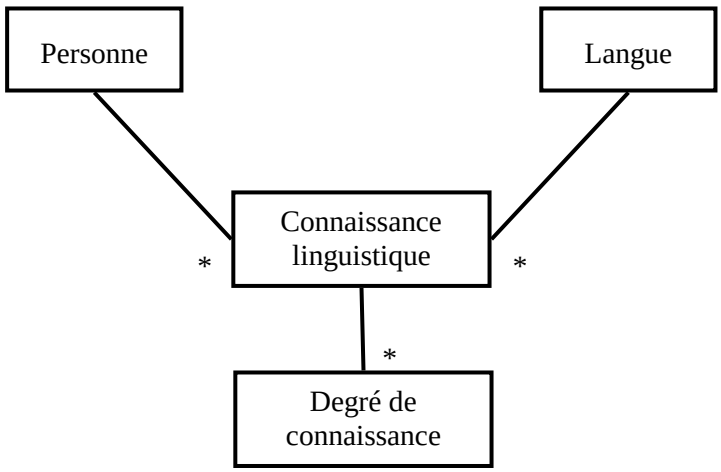
Nous avons déjà mentionné à plusieurs reprises que l'interdiction stricte de tableaux d'attributs dans la définition d'un type d'objet appartient au passé et que certains SGBD du marché sont capables de les gérer parfaitement. Nous avons aussi affirmé que, à notre avis, il fallait limiter cette possibilité aux cas les plus simples.

À titre d'exemple, nous étudions la façon de décrire les connaissances linguistiques d'un ensemble de personnes. Nous avons déjà créé un type d'objet *langue* (peut-être intégré à la table des codes).

Personne (#-personne, nom, prénom, sexe, date naissance, …

Langue (#-langue, libellé langue, …

Une personne peut évidemment parler plusieurs langues et une langue être parlée par plusieurs personnes. Il s'agit bien d'une association de type *plusieurs à plusieurs* qui nécessiterait normalement un type associatif. Nous avons aussi codifié les degrés de connaissance de langues sous forme d'une nouvelle table dans la table des codes, ceci dans le but d'éviter la saisie de tout et n'importe quoi dans ce champ.



Connaissance linguistique (*#-personne, #-langue, #-degré de connaissance*, …

En utilisant les possibilités offertes par certains SGBD et en violant la 1ère forme normale dans sa définition traditionnelle, on peut éviter de créer ce type associatif et écrire:

Personne (#-personne, nom, prénom, …, tableau (*#-langue, #-degré connaissance*), …

SQL-92 permet de définir avec *array* des tableaux de dimension fixe, de prévoir par exemple 4 langues au maximum. Oracle permet de définir des variables de type *varray* pouvant contenir un nombre variable (mais également limité) d'éléments.

Quelle solution est la plus appropriée? Cela dépend certainement du problème à traiter, mais, pour notre part, nous optons plutôt pour l'approche traditionnelle, donc la définition d'un type associatif.