Le réseau sémantique universel (1ère partie)

Des concepts semblant sans rapport entre eux présentent parfois de grandes similitudes. Voici une synthèse - très simplifiée - de vingt années de réflexion sur divers modèles de données et de traitements.
L'auteur vous prie de l'excuser pour la densité - et parfois le manque d'explications ou de rigueur -, mais il est difficile de condenser en quelques pages ce qui nécessitera une publication plus complète (dans un proche avenir).

Introduction

De 1977 à 1982, j'étais, chez IBM, un des responsables du projet "Advanced Data Dictionary Utilities". Il s'agissait d'un produit maison, dont j'étais un des principaux artisans, qui transformait le dictionnaire de Données standard en un véritable AGL avant la lettre. Le support SGBD, bien que hiérarchique (DL/1), était exploité un maximum au travers de liens logiques, donnant une véritable émulation entité-relation. Malgré le grand nombre de couches logicielles (TSO / Application / Dictionnaire / IMS / DL/1 / MVS), l'importance de la mémoire consommée (de 3 à 6 Mo, valeur énorme pour l'époque) et la faible puissance - comparativement à maintenant - de la machine (un 370/168 de 1 Mips, équivalent d'un PC 286), nous parvenions à traiter 1000 - oui, mille - requêtes par minute, performance pas encore atteinte en 1996 par tous les SGBD relationnels en exploitation.

Ce niveau de performance, associé à une grande souplesse d'utilisation, m'a convaincu que l'avenir du stockage et du traitement de l'information passait par le modèle entité-relation. Des essais plus récents (1987) sur un premier prototype ont donné des résultats dépassant mes attentes (la première fois, j'ai cru que le traitement s'était interrompu prématurément) : chargement de masse d'entités avec identifiants 20 fois plus rapide qu'un simple chargement - non indexé - sous Dbase III, temps d'accès en lecture séquentielle indexée de moins d'une milliseconde sur un PC 8086, etc.

J'ai bien sûr continué d'étudier le domaine, et notamment comment intégrer un maximum d'informations dans la base afin d'automatiser au maximum les traitements (incorporation de code dans les données)[1].

Par ailleurs, il y a une dizaine d'années, je m'étais attaqué à l'élaboration du métamodèle de ce qu'on appelait alors "système expert", autrement dit un système à base de connaissances. Je cherchais notamment à permettre aisément les chaînages - avec des calculs de probabilité - tant avant qu'arrière, afin de réaliser un produit qui tournerait sur "mon" SGBD (permettant un nombre de faits et de règles illimités). Le résultat graphique de cette modélisation ressemblait beaucoup à un Modèle Conceptuel de Données, tout en fonctionnant - à la réversibilité près - comme un Modèle Conceptuel de Traitements.

Ces dernières années, j'ai donc eu la curiosité d'effectuer des comparatifs entre divers réseaux sémantiques, et j'en ai conclu qu'il devait être possible de définir une modélisation commune et de définir ainsi un système de base de données entité-relation[2] sur lequel on pourrait faire tourner les moteurs les plus variés.
Voici le résultat de ces cogitations.

---

[1] *Ce modèle a d'ailleurs été étudié en 1990 par IBSI, lors de la recherche d'un SGBD pouvant accueillir la version Windows de son AGL Conceptor, et - malheureusement - non retenu.*

[2] *Ou entité-association pour ceux qui préfèrent, c'est exactement la même chose.*

Les modèles analysés

Il n'est évidemment pas question d'effectuer un rapprochement entre les dizaines - sinon centaines - de sujets et de méthodes de modélisation. D'autant plus que nombre d'entre eux, s'appliquant à des domaines relativement proches, ne diffèrent que par le symbolisme et quelques détails particuliers.

Cette étude n'avait d'intérêt que si on pouvait effectivement rapprocher des systèmes a priori complètement différents, notamment en ce qui concerne leurs natures et leurs applications. Les systèmes mis en présence sont les suivants :

- Modèle Conceptuel de Données (MCD) de Merise, dans lequel on peut naviguer avec un moteur de base de données ;
- Modèle Conceptuel de Traitements (MCT) de Merise ;
- Système à Base de Connaissance (SBC), parcouru et modifié par un moteur d'inférences ;
- Tableur, mis à jour par un moteur d'algorithmes ;
- Système NeuroMimétique (SNM) ;
- Modèle de Traitements en Temps Réel (TTR) ;
- Programmation Orientée Objets (POO) telle qu'elle est mise en œuvre avec des outils comme Delphi et C++ Builder (Borland), Visual Basic et Visual C++ (Microsoft), Visual Objects (Computer Associates), NSDK (Nat Systèmes), Optima++ (Powersoft), etc.

Nous allons voir que tous ces systèmes peuvent être représentés avec un même formalisme et stockés dans un modèle unique de base de données.

Les points communs

Tous ces systèmes ont en commun qu'ils peuvent être - et sont généralement - modélisés par un réseau de nœuds reliés entre eux :

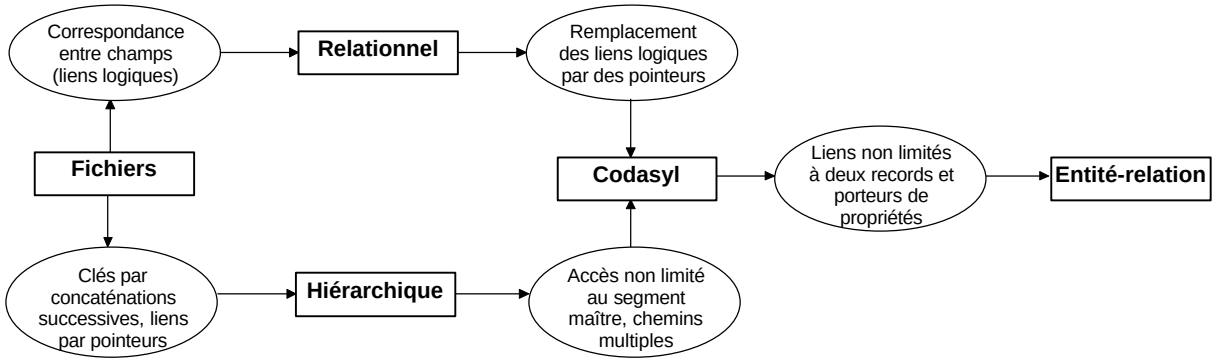
Modèle	Nœud	Lien	Particularités	Navigation
MCD	Objet	Relation	Cardinalités, contraintes ensemblistes	Base de données
MCT	Événement / Résultat	Opération	Synchronisation	Enchaînement séquentiel
SBC	Fait	Règle	Opérateurs logiques, priorités	Chaînage avant et/ou arrière
Tableur	Cellule (valeur)	Cellule (formule)	Disposition en feuilles, lignes, colonnes	Calcul et Propagation
SNM	Neurone	Axone	Synapses	Activation et Propagation
TTR	Donnée	Traitement	Capteurs et effecteurs	Algorithmes
POO	Composant / Classe	Héritage, Méthode	Méthodes incorporées aux classes	Traitement des messages

Comment nœud et lien ont-ils été distingués ?

Tout simplement selon la règle de base : un lien relie des nœuds, un nœud n'a pas nécessairement plusieurs liens :

- dans un MCT, un événement peut être initial et un résultat final, et ils ont une existence propre, alors qu'une opération se situe obligatoirement entre (au moins) un événement et (au moins) un résultat ;
- dans un SBC, un fait a une existence propre et peut être initial ou final, alors qu'une règle ne sert qu'à passer d'un fait à un autre ;
- dans un tableur, la valeur de chaque cellule a une existence propre, alors que les formules ne servent qu'à obtenir une valeur à partir d'une ou plusieurs autres ;
- dans un SNM, chaque neurone a un état spécifique, les axones (avec synapses et dendrites) ne servant qu'à les relier ;
- un TTR est analogue à un MCT ;
- dans une application objet, tout ce qui relie deux composants est un lien.

Certains pourraient critiquer que le seul représentant des SGBD pris en compte est le modèle entité-relation, et qu'il en existe d'autres. Je leur répondrai que ce modèle est le plus puissant, et que tous les autres en sont des sous-ensembles, comme le montre le diagramme ci-dessous :



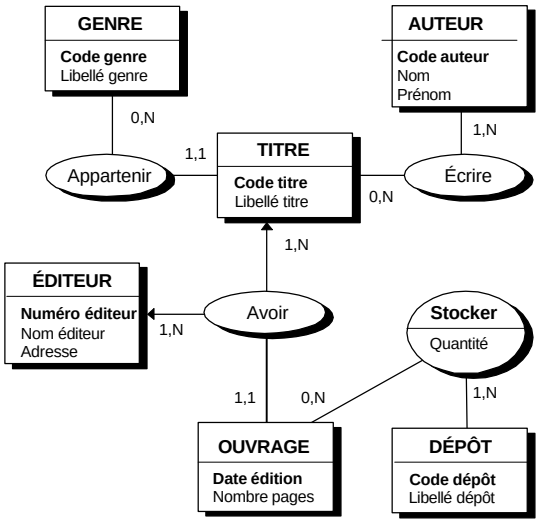
Nota : le succès actuel - et temporaire - du modèle relationnel est uniquement dû à la simplicité de sa mise en œuvre.

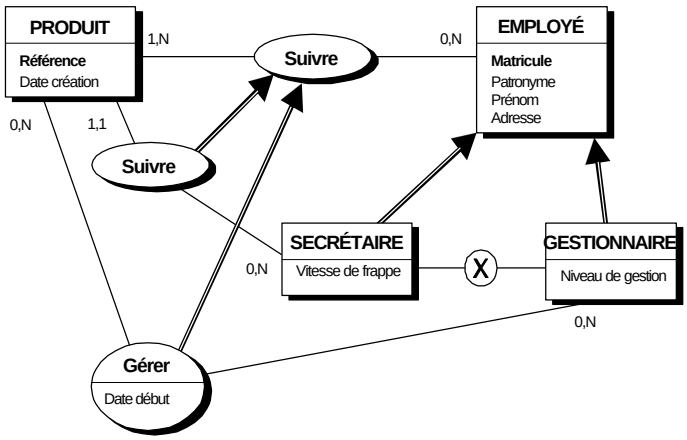
Les spécificités de chaque système

Le Modèle Conceptuel de Données

- Un **objet** contient de 1 à N **propriétés**, dont au moins une sert d'identifiant. Deux individus (= occurrences d'objet) ne peuvent avoir la même valeur d'identifiant.
- Une **relation** a de 2 à N **pattes**, chacune de celles-ci associant la relation à un objet (ou à une autre relation dans OOM - Orientation Objet dans Merise). À chaque patte est associée un couple de cardinalités (mini-maxi) correspondant au nombre d'occurrences de la relation pouvant être rattachées à un même individu. Une relation peut avoir des propriétés.
- Une **propriété** est une information de base. C'est une donnée indivisible dans Merise, qui peut (dans OOM) être itérative ou un agrégat (struct du C).
- Les **contraintes** ensemblistes sont un ensemble de règles définissant les possibilités d'évolution des objets et relations, que ce soit en création, mise à jour ou suppression : inclusion, exclusion, simultanéité, unicité, stabilité, etc.).
- L'**héritage** permet de définir des sous-types d'objets et de relations (généralisation par héritage simple ou multiple) et des classes (spécialisation par prédicat), en affinant les modèles avec une approche objet.

Le formalisme d'un MCD Merise est bien connu. En voici deux exemples :



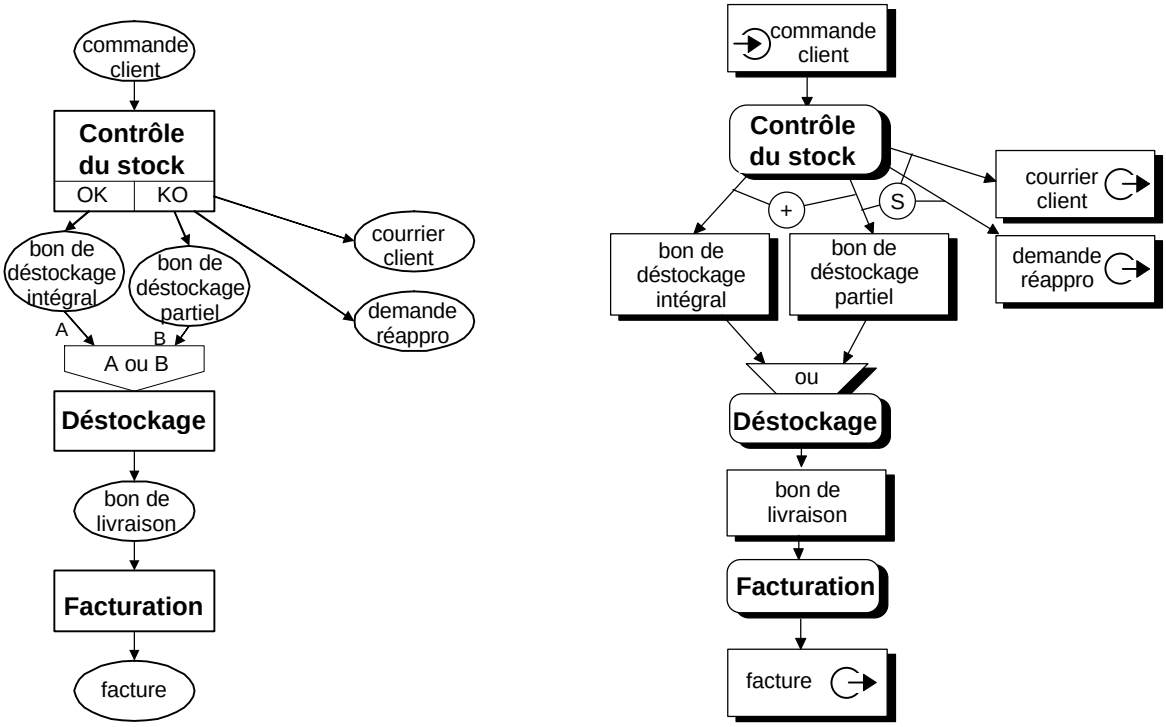


C'est ce type de formalisme - avec des enrichissements - qui sera utilisé pour l'unification des différents systèmes. Si des Merisiens puristes (dont je fus) trouvent à redire aux modèles ci-dessus, qu'ils sachent que notre objectif ici n'est pas de faire du Merise pur et dur, mais de le faire évoluer, même au prix de quelques "entorses" (c'est vrai que le suivi par la secrétaire est une relation restreinte du suivi par un employé, mais nous verrons plus loin qu'en POO le seul changement de cardinalité - qui masque un changement de méthode - correspond bien à un héritage).

Le Modèle Conceptuel de Traitements

- Un **événement** (en entrée d'une opération) ou un **résultat** (en sortie d'une opération) contient de 1 à N **propriétés**. Dans la suite, il sera sous-entendu que événement≡utilisation et résultat≡création/mise à jour.
- Une **opération** est constituée de **règles de gestion**.
- Une **synchronisation** précise les contraintes de déclenchement d'une opération.
- Une **règle de gestion** est une fonction qui utilise, crée ou modifie des propriétés d'un événement|résultat, d'un objet ou d'une relation.

Le formalisme Merise d'un MCT est également bien connu. L'exemple ci-après (à gauche) est associé au même en formalisme unifié (à droite).



Si l'intérêt de ce changement de formalisme n'apparaît pas très évident ici[3], il prendra tout son sens plus loin. Bien sûr, s'agissant d'un ensemble dynamique, la signification des symboles n'est pas rigoureusement la même que dans un MCD : notamment, les "pattes" sont des chemins temporels et non spatiaux, les relations étant des actions ; par contre, les événements et résultats sont bien des objets, même s'ils n'ont qu'une durée de vie limitée.

De plus, une véritable implémentation devra descendre au niveau des règles de gestion, puisque ce sont elles qui manipulent les propriétés. Aussi, pour la suite, nous ne considérerons les opérations que comme des sous-systèmes constitués de règles de gestion reliant des objets.

Le Système à Base de Connaissances

- Un **fait** est une information ayant un état ou une valeur, obtenue par l'application d'une règle.
- Une **règle** est une condition, pouvant être associée à un algorithme, qui, à partir d'une ou plusieurs conditions ou prédicats, détermine l'état ou la valeur d'un ou plusieurs faits.
- Une **prémisse** ou un **prédicat** est une condition, dont le résultat est généralement booléen.
- Une **action** est le traitement déclenché quand les prémisses qui lui sont associées sont vérifiées.
- La **priorité** permet de définir, parmi les diverses règles activables, celles qui doivent l'être en premier (celles qui sont déclenchées immédiatement - priorité la plus haute - sont parfois appelées **démons**).
- Un **chaînage avant** est un traitement qui, à partir de valeurs initiales d'un ou plusieurs faits, parcourt les règles pour en déduire l'état ou la valeur d'autres faits (raisonnement déductif).
- Un **chaînage arrière** est un traitement qui, à partir de valeurs connues d'un ou plusieurs faits, parcourt les règles pour déterminer l'état ou la valeur d'autres faits pour qu'ils soient cohérents avec les faits connus (raisonnement inductif).

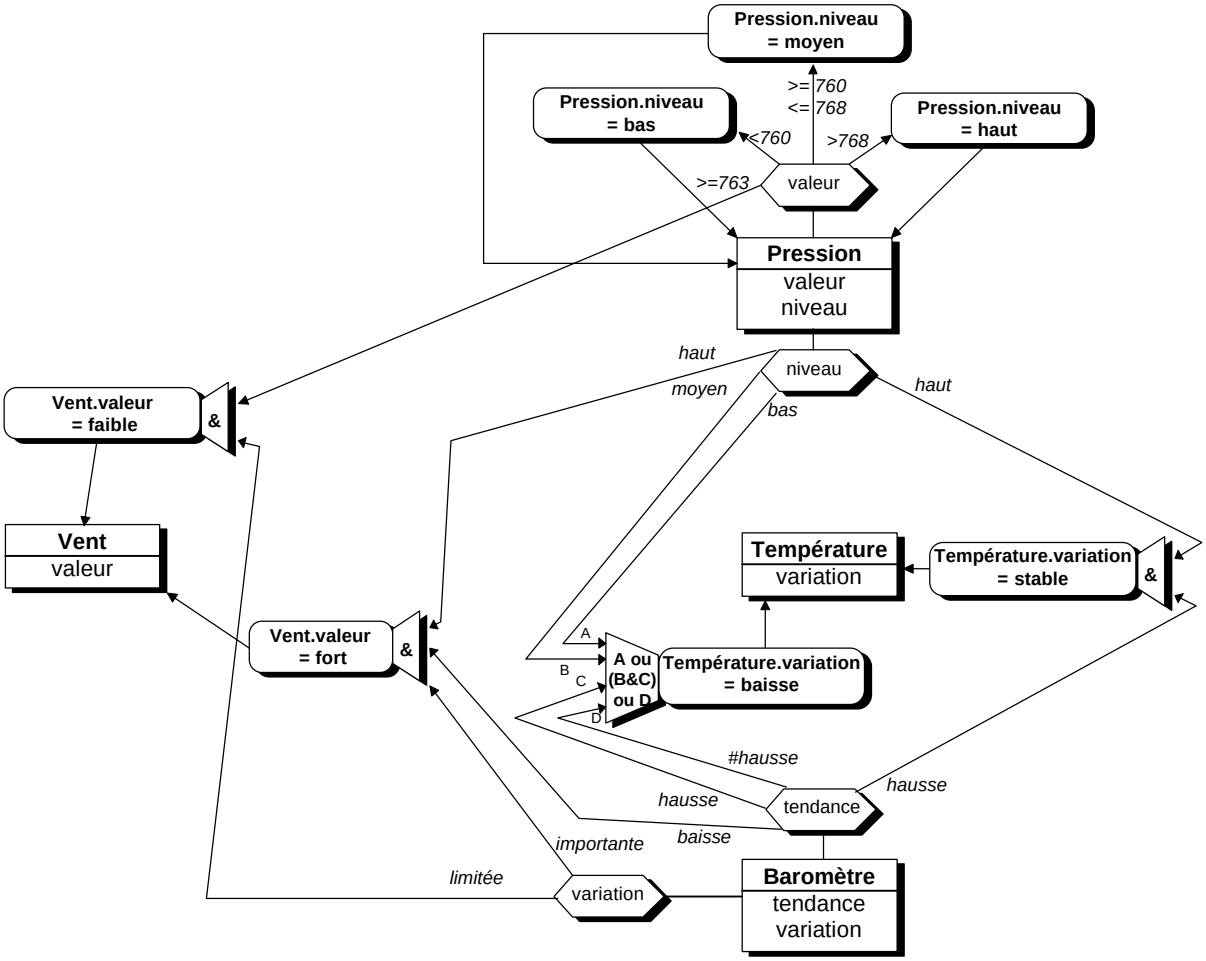
---

[3] *Bien que certaines méthodes utilisent le même formalisme pour les traitements et les données ; exemple, SADT avec ses actigrammes et ses datagrammes.*

De plus, il existe plusieurs niveaux de logique :

- la logique d'ordre **0**, où les faits sont booléens et uniques (`Si température en hausse...`);
- la logique d'ordre **0+**, où les faits peuvent constituer des populations (`Si X parent de Y...`);
- la logique d'ordre **1**, qui permet en plus de générer des relations entre objets
  (`Si terrain A au-dessus de terrain B, alors A plus récent que B...`).
- la logique d'ordre **2** ? On m'en a parlé, mais je n'en ai jamais rien lu. Elle ne sera donc pas abordée ici[4].

En ce qui concerne le formalisme, il en existe à peu près autant que de produits, aucune normalisation n'étant intervenue à ma connaissance. Voici donc un exemple - très - simple en formalisme unifié :



À noter que les objets modélisés ne représentent pas des populations, mais les individus eux-mêmes. Ceci sera explicité plus loin dans cet article[5].

L'exemple ci-dessus n'est évidemment pas représentatif d'une modélisation de SBC en général : il faut - entre autres - différencier les prédicats simples (agissant et/ou utilisant des propriétés individuelles) des prédicats génériques (agissant sur des populations).

---

[4] *Toute information sur ce sujet sera la bienvenue.*

[5] *Pour ceux qui remarqueraient que l'objet Pression n'est pas en deuxième forme normale puisque niveau dépend de valeur, je répondrai : ben oui.*

Le Tableur

- Une **cellule** est l'élément de base d'un tableur. Elle ne peut contenir qu'une valeur, et est identifiée par ses coordonnées (feuille, ligne, colonne).
- Une **formule** est un algorithme, associé à la cellule qui en recevra le résultat. Les données utilisées peuvent être des constantes ou des valeurs stockées dans d'autres cellules.
- Un **calcul** est la résolution des algorithmes, qui peut se faire, soit en temps réel (au fur et à mesure des entrées de données), soit en différé (sur demande).

On retrouve ici une structure très proche d'un SBC (cellule ≡ fait, formule ≡ règle, calcul ≡ chaînage avant). Des utilitaires divers (scénarios, valeur cible...) fournissent, par approximations successives, un succédané de chaînage arrière. En fait, la principale différence entre les deux systèmes est que, dans un SBC, un fait conserve sa valeur si aucune règle ne le modifiant n'est déclenchée, alors qu'une cellule de tableur doit toujours être calculable à partir de sa formule - sauf utilisation de macros ou procédures.

La disposition en lignes et colonnes a également un rapport très proche avec les SGBD : une table relationnelle n'est pas construite autrement.

Inutile ici de montrer une modélisation de tableur, puisqu'il est possible de les utiliser tant comme SGBD que pour générer des SBC simples. D'ailleurs, compte tenu des similitudes avec les SGBD et SBC, je considérerai pour la suite qu'un tableur est une version simplifiée de l'un et/ou de l'autre, et donc que la modélisation de ceux-ci entraîne ipso facto celle des tableurs.

Le Système NeuroMimétique

- Un **neurone** est l'élément constitutif de base d'un SNM. Il ne contient - dans les modèles simples - que deux informations : son état (activé ou non) et son seuil d'activation.
- Un **axone** est le lien entre deux neurones, l'un transmetteur, l'autre récepteur.
- Une **synapse** est le point d'entrée dans un neurone depuis un axone. Elle porte une information élémentaire : activante ou inhibante.
- Un **capteur** est un point d'entrée dans un réseau de neurones. C'est un événement externe, relié à des neurones au travers d'axones et synapses.
- Un **effecteur** est un point de sortie d'un réseau de neurones. C'est un résultat externe, relié à des neurones transmetteurs.
- L'**apprentissage** est un processus de mise à niveau des seuils des neurones par itérations successives, pour obtenir en sortie le résultat attendu par rapport aux entrées (il existe aussi un mode d'apprentissage non supervisé, dans lequel on laisse le réseau s'auto-organiser).

L'ensemble neurone.seuil + synapse joue ici un rôle analogue à celui de la synchronisation dans un MCT.

Par ailleurs, un SNM fonctionne comme un SBC, sauf que faits (≡ neurones) et règles (≡ axones) sont indifférenciés et que l'ensemble peut boucler sur lui-même jusqu'à atteinte d'un état stable - ce qui existe cependant aussi dans certains SBC à système dynamique. Il peut disposer également d'un taux de bruitage (variations aléatoires du seuil d'activation) destiné à accélérer la convergence du système vers un état stable.

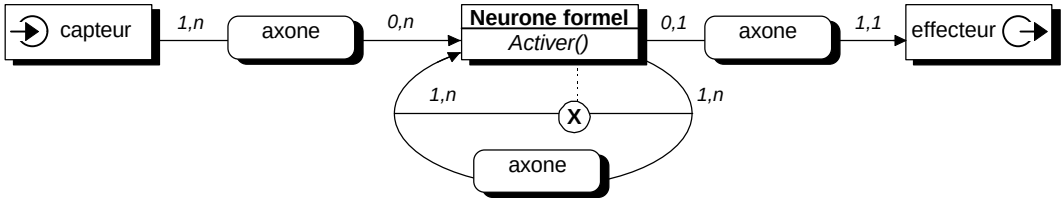
Tous les neurones étant a priori identiques, seuls comptant les informations introduites en entrée et les valeurs récupérées en sortie, une modélisation n'aurait guère de sens, sauf pour montrer le mode de couplage : rétroactivité ou non (≡ contrainte ensembliste), niveau de couplage (≡ cardinalité des relations).

On peut ainsi considérer un SNM comme un cas particulier de SBC à logique d'ordre 0+, la seule règle étant :

Neurone.actif = (ΣValeurs(synapses) + F(bruit) > Neurone.seuil)

...et devant être considérée comme une méthode de l'objet Neurone (Activer() dans l'exemple ci-dessous).

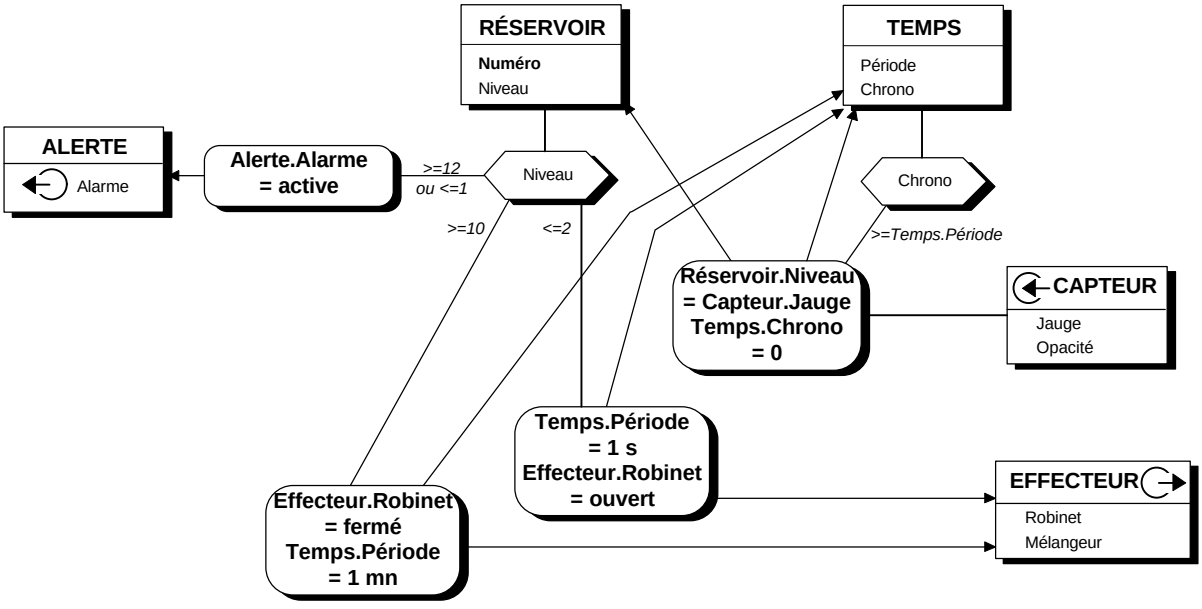
Voici, à titre d'exemple, la modélisation d'un SNM dans le cas le plus général :



Le Traitement Temps Réel

À l'exception d'une prise en compte explicite des périphériques (capteurs, effecteurs...), un Traitement Temps Réel se modélise - en formalisme unifié - exactement comme un MCT, éliminant par la même occasion l'un des reproches faits à Merise, à savoir son inadaptation au monde industriel.

En voici pour preuve un exemple de traitement - très simple -, à savoir le contrôle du niveau d'un réservoir et de son remplissage (la propriété Temps.Chrono est asservie à l'horloge et s'incrémente donc automatiquement) :



Tout comme celui du SBC, les objets du schéma ci-dessus sont directement des individus.

La Programmation Orientée Objet

- Un **composant** (ou **objet**, ou **classe**) est, soit un élément visible et manipulable (champ de saisie, champ mémo, image, navigateur de base de données, etc.), soit un élément regroupant des traitements et des données (table relationnelle, requête SQL, etc.).
  Il y a généralement une très importante arborescence d'héritage entre les divers types d'objets disponibles dans un générateur d'application.
  Ainsi, tous les objets visuels ont en commun des propriétés telles que leur taille et leur position dans la fenêtre.

- Une **propriété** est une information appartenant à un composant et accessible, soit en lecture, soit en écriture, soit les deux.
  Certaines propriétés cachent en fait des relations (par exemple, la propriété Table d'un composant Navigateur[6], renseignée avec le nom de la table, génère en fait un lien entre le navigateur et la table manipulée à travers lui).

- Une **méthode** est une fonction permettant de lire ou de modifier une ou plusieurs propriétés. C'est une règle de gestion interne à l'objet.
  Il y a parfois redondance entre propriété et méthode ; exemple (Delphi) : `Bouton.Hide` a exactement le même effet que `Bouton.Visible := faux`.
  En réalité, les deux coexistent : la méthode `Hide` cache le bouton, puis met la propriété `Visible` à `faux`.
  L'erreur du créateur de l'objet est d'avoir, dans le second cas, masqué la méthode derrière une apparente affectation de valeur : la propriété `Visible` peut être initialisée, mais ne devrait normalement être accessible à l'exécution qu'en lecture, sa mise à jour étant l'exclusivité des méthodes `Hide` et `Show`.

- Un **événement** est une action quelconque (généralement en provenance de l'utilisateur, mais parfois aussi déclenchée par une fonction) qui déclenche l'exécution d'une fonction associée. Cette fonction est toujours un lien entre deux objets (l'un d'entre eux pouvant être un événement externe au sens MCT, comme le fait de cliquer sur un bouton ou d'appuyer sur une touche de fonction).

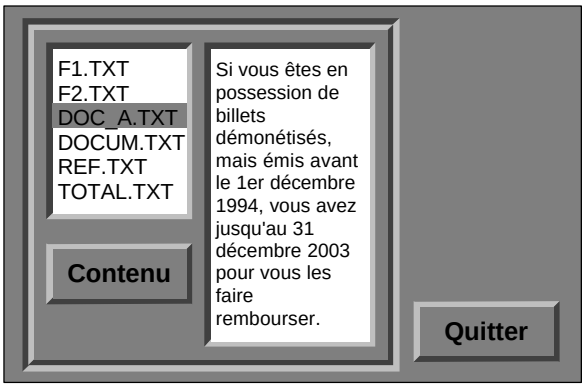
En pratique, la mise en œuvre d'une application à l'aide de composants visuels revient plus ou moins à faire de la programmation Temps Réel, et les développeurs qui programment un bouton "Interrompre" ou une jauge permettant de visualiser l'avancement d'un traitement, ou ceux qui créent des jeux vidéo, font - dans une certaine mesure - du temps réel comme Monsieur Jourdain faisait de la prose...

D'où l'apparente inadaptation de Merise au développement d'applications en POO.

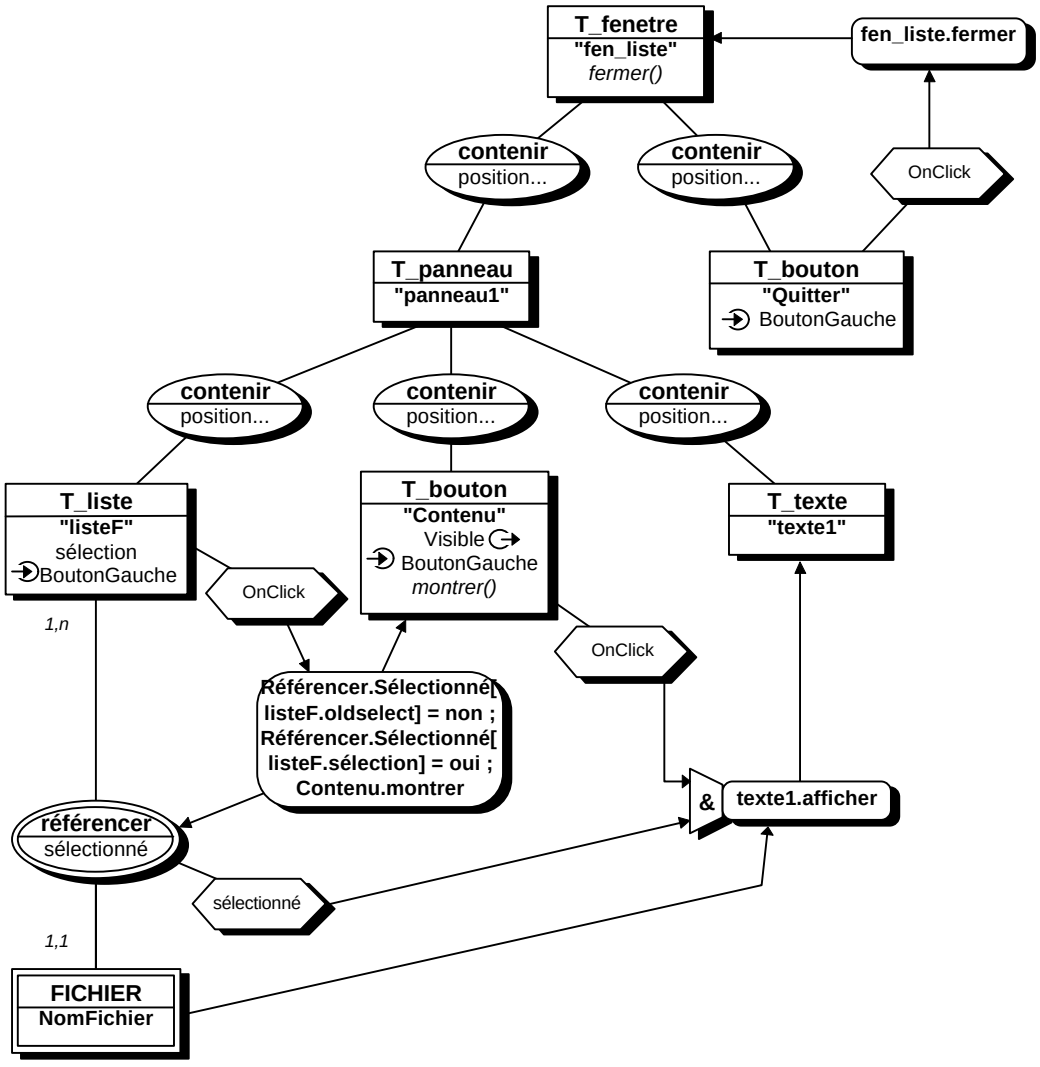
---

[6] *Navigateur = ensemble de boutons correspondant aux fonctions premier, précédent, suivant, dernier, ajouter, supprimer, modifier, enregistrer, annuler et ayant cet aspect :* 

Comment modéliser ? Eh bien, exactement comme un traitement Temps Réel : prenez l'exemple de la fenêtre ci-dessous, composée de 6 éléments : une liste (1), un texte (2), un bouton (3), les trois contenus dans un panneau (4), un autre bouton (5), l'ensemble dans la fenêtre elle-même (6).



Sachant que le bouton Contenu n'apparaît qu'après sélection d'un élément de la liste, et qu'il permet d'afficher le contenu du fichier sélectionné, on peut modéliser le tout comme suit :



Tout comme celui du SBC et du TTR, les objets du schéma ci-dessus sont directement des individus, on peut d'ailleurs le constater par la présence de deux T_bouton. Les modèles n'ont pas été représentés.

On peut remarquer les capteurs des objets listeF, Contenu et Quitter (détection des actions faites avec la souris par l'opérateur, à l'origine des événements OnClick).

Formalisme et Vocabulaire

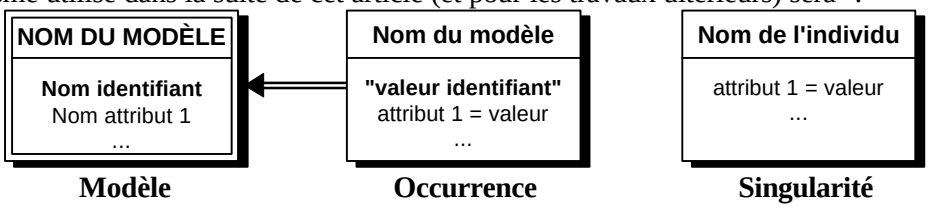
Maintenant que nous avons vu qu'il était possible de modéliser pratiquement n'importe quoi avec un formalisme équivalent, il s'agit à présent de rassembler les différents composants de chacun de ces modèles pour élaborer un ensemble cohérent. Si le formalisme de base est celui du MCD, le formalisme unifié sera nécessairement enrichi pour prendre en compte des concepts supplémentaires[7].
Prenons les divers concepts recensés, et voyons, catégorie par catégorie, ce qui les différencie.

Les objets

Cette catégorie regroupe objet (pour un MCD) ; événement, résultat (pour un MCT) ; fait (pour un SBC) ; neurone, capteur, effecteur (pour un SNM). Tous sont porteurs de propriétés.
On peut diviser cet ensemble en plusieurs sous-catégories :

- Les **populations**, ensembles d'individus ayant les mêmes propriétés, différenciés par leur identifiant, et manipulées par des règles de gestion communes à tous les individus d'une même population. C'est la règle générale dans un MCD, un SNM ou un SBC d'ordre 0+. Dans une modélisation, ce type d'objet n'est représenté que par son modèle ; on ne montre pas les individus.
- Les **occurrences**, ensembles d'individus ayant les mêmes propriétés, différenciés par leur identifiant, mais pouvant être manipulés par des règles de gestion spécifiques pour chaque individu. C'est une règle courante dans un SBC (exemple : dans un ensemble de métriques, chacune - fait - est mise à jour par une ou plusieurs questions spécifiques - règles), et la généralité en POO - c'est pourquoi la modélisation nécessite de représenter les individus et non seulement les modèles.
- Les **singularités**, individus spécifiques directement identifiés par leur type (ex : SBC d'ordre 0 et/ou paramètres). Il n'y a dans ce cas évidemment aucune distinction entre le modèle et l'individu.
- Les **modèles**, qui permettent un certain niveau de généralisation : tous les individus héritent de ses attributs et relations.

Le symbolisme utilisé dans la suite de cet article (et pour les travaux ultérieurs) sera[8] :



Les attributs

Nous parlons ici des différentes informations stockées dans les objets et dans certaines relations. Elles sont également de plusieurs types :

- Les propriétés simples, correspondant à Merise "classique", que nous appellerons désormais plutôt **attributs**, car en langage courant les propriétés d'une chose sont généralement des actions (par exemple, les propriétés d'un médicament sont des actions, alors que sa composition et son conditionnement sont des caractéristiques statiques).
- Les **faits**, ou propriétés facultatives, c'est-à-dire pouvant avoir un état non renseigné (sa valeur n'a pas été fournie), absent (sa valeur n'existe pas), inconnu (sa valeur est inconnue), flou (sa valeur est fournie avec une certaine probabilité d'exactitude, ou une fourchette, ou contenue dans une liste de plusieurs valeurs)[9]. Nous les identifierons en les faisant précéder du symbole : ≈. (Exemple : ≈Téléphone)

---

[7] *Comme le lecteur attentif s'en sera déjà aperçu dans les exemples présentés, et bien que le symbolisme proposé n'y ait pas toujours été intégralement pris en compte.*

[8] *La distinction majuscules/minuscules n'est pas significative.*

[9] *Par exemple, si l'attribut "n° de téléphone" n'est pas renseigné, est-ce parce qu'on ne l'a pas fourni, parce qu'on n'a pas pu l'obtenir (liste rouge...), ou parce que la personne n'a pas le téléphone ?*

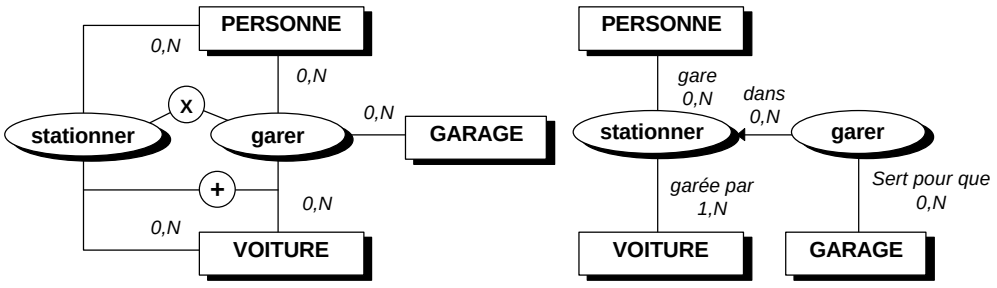
- Les **tableaux**, ou propriétés itératives (exemple : n° de téléphone). Nous les identifierons en les faisant suivre d'une paire de crochets : **[ ]**.
  Exemple : Prénoms[ ]
- Les **agrégats**, propriétés décomposables en plusieurs composantes (exemple : adresse, décomposable en n°, rue, code postal, ville). On peut bien sûr avoir des agrégats contenant des tableaux, et des tableaux d'agrégats. Nous identifierons un agrégat, soit en le faisant suivre de l'énumération de ses composantes encadrée par des accolades :
  Exemple : Agrégat
  { attribut_a
  attribut_b
  ... }
  ou, si on ne détaille pas, en le faisant suivre par : {...}.
  Exemple : Agrégat {...}
- Les **méthodes**, traitements spécifiques à l'objet et permettant l'accès à un ou plusieurs attributs (en lecture et/ou en mise à jour). Elles peuvent masquer des attributs (mise à jour), ou, au contraire, faire apparaître des attributs virtuels (en lecture), valeurs renvoyées par la méthode, qui les a calculées à partir d'autres attributs. Nous les identifierons en les faisant suivre d'une paire de parenthèses, contenant éventuellement la liste des paramètres transmis. Pour bien les différencier des attributs "données", il est recommandé de les mettre en italiques :
  Exemples : *Cacher()*
  Afficher(oui½non)
  Nota : dans un héritage, une méthode peut être redéfinie. Dans ce cas, elle remplace la méthode de l'objet parent.

Les notions de **capteur** et d'**effecteur** sont plus complexes. En effet, en POO notamment, un objet peut être à la fois capteur et effecteur (un bouton est capteur par ses méthodes générant des événements OnClick, etc.; il est effecteur par ses caractéristiques d'affichage - position, visible, etc.). Ces notions doivent donc être déportées au niveau des attributs, sauf si la totalité de l'objet est concerné - cas classique des événements/résultats externes d'un MCT. Nous identifierons un capteur par le symbole : ⊙→, et un effecteur par le symbole : ↻→, placés devant le nom de l'attribut, ou devant le nom de l'objet si tous ses attributs sont des capteurs ou des effecteurs.

Les associations

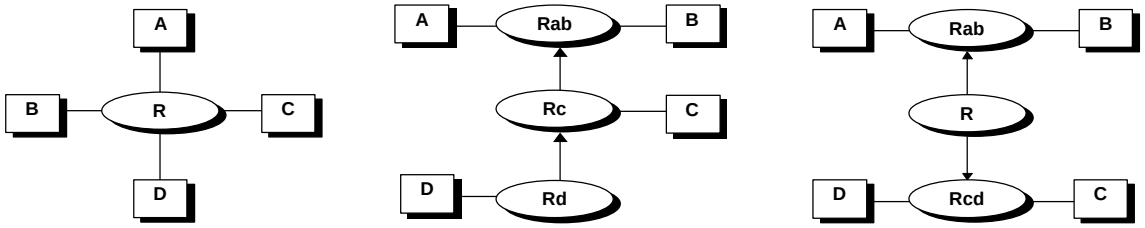
Le terme d'**association** recouvre les relations statiques, c'est-à-dire celles qui relient les objets "spatialement". Ce sont les relations classiques des MCD et SGBD.
Pour pouvoir aisément différencier les catégories - et aussi parce qu'il est extrêmement complexe, lourd et peu pratique de mettre en œuvre des relations physiques sur plus de deux objets -, nous ramènerons toutes les associations à deux pattes, grâce aux associations d'associations. En effet, dans une relation ternaire ou davantage, il est toujours possible de classer les différents objets par ordre d'importance, et donc de définir une paire qui servira d'association de base, à partir de laquelle viendront se greffer d'autres associations, jusqu'à obtention du nombre de pattes complet, ce qui peut être facteur d'économie.

Par exemple, la version de droite n'est-elle pas plus simple que celle de gauche ?



En pratique, nous verrons que comme la mise en œuvre oblige à nommer les pattes, ce nommage même permet de définir une sémantique mettant en évidence les priorités (dans l'exemple ci-dessus, on dit couramment qu'une personne gare sa voiture dans un garage, et rarement qu'une personne dans un garage gare sa voiture[10]; ou alors, éventuellement, qu'une personne utilise un garage pour garer sa voiture : le verbe principal définit la relation mère).

S'il y a plus de trois pattes, plusieurs possibilités existent, mais l'une d'entre elles est certainement sémantiquement meilleure :

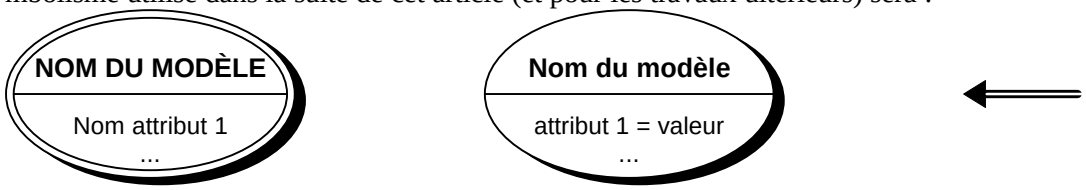


Ceci... **correspond-il à ceci...** **ou plutôt à cela ?**

Les différentes sous-catégories d'associations sont donc :

- Les **populations** relient deux populations d'objets, ou une population avec une occurrence ou une singularité. Dans une modélisation, ce type d'association n'est représenté que par son modèle.
- Les **particularités**, associations reliant des occurrences et/ou des singularités. Il n'y a dans ce cas pas nécessairement de modèle.
- Les **modèles**, qui permettent un certain niveau de généralisation.
- Les **héritages**, qui relient des objets (ou des associations) divers et correspondent toujours à une cardinalité (1,1) côté enfant et (0,1) côté parent :
    - une population peut hériter d'une ou plusieurs autres populations ;
    - une occurrence peut hériter du modèle d'une population ou d'une autre occurrence ;
    - une singularité peut hériter d'un modèle quelconque ou d'une autre singularité.
    - l'héritage peut être une spécialisation, auquel cas un prédicat s'interpose entre l'objet parent et l'objet enfant.

Le symbolisme utilisé dans la suite de cet article (et pour les travaux ultérieurs) sera :



Modèle **Particularité** **Héritage (parent ← enfant)**

Les actions

Les **actions** sont des relations "temporelles". Elles définissent - comme leur nom l'indique - des actions qui sont déclenchées lorsque les conditions définies par leurs prémisses sont remplies. Elles sont orientées de la cause vers l'effet, c'est pourquoi leurs pattes sont toujours fléchées.
Elles relient des modèles, des occurrences ou des singularités ; jamais des individus de populations (ceux-ci disposent des actions de leur modèle).

Toute action est particulière : il n'y a pas de modèle d'action. Dans un "héritage", une action redéfinie remplace l'action-parent, mais ne l'enrichit pas (ni spécialisation, ni généralisation).

---

[10] *Sauf si on veut imiter Monsieur Jourdain.*

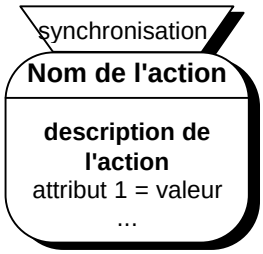
Ce qui différencie essentiellement une action d'une méthode, en dehors du fait qu'une méthode est interne à un objet - mais une action peut également ne concerner qu'un seul objet (voir par exemple Pression.niveau dans l'exemple de SBC présenté dans les pages précédentes) -, c'est qu'une méthode s'exécute quand elle est appelée (par une autre méthode ou par une action), alors qu'une action est déclenchée par un événement :

- externe (souris, clavier, capteur, etc.) ;
- ou interne (attribut modifié, objet ou association créé, etc.).

Ce déclenchement est soumis à un certain nombre de contraintes :

- prémisses et/ou prédicats vrais ;
- présence simultanée et/ou exclusive de prémisses et/ou prédicats, c'est-à-dire une synchronisation ;
- priorité d'exécution.

Les prédicats et prémisses sont reliés à l'action par une patte, la priorité est un attribut de l'action. La synchronisation nécessite un symbolisme particulier. Nous utiliserons donc le formalisme suivant[11] :

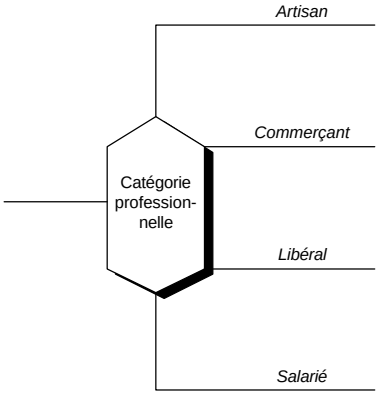


La synchronisation n'est bien sûr à représenter que si elle existe.

Les prédicats

Les **prédicats** constituent en quelque sorte une classe à part, puisqu'ils sont utilisés tant pour les données (héritage par spécialisation) que pour les actions (dans un SBC, l'ensemble prédicats+synchronisation constitue la partie si de la règle, le résultat de l'action en étant la partie alors).

Un prédicat concerne un attribut (appartenant à un objet ou à un association, ou éventuellement à une action), qui est susceptible de prendre plusieurs valeurs, ou un événement. Nous le formaliserons comme ceci :



objet ou association propriétaire de l'attribut ← **→ objet enfant ou action utilisatrice**

À chaque patte sortante est associée la valeur du prédicat (éventuellement précédée d'un opérateur de comparaison : = # < ≤ > ≥). Cette patte est toujours fléchée, comme nous le verrons plus bas.
La seule patte non fléchée d'un prédicat est celle qui le relie à l'objet ou association propriétaire de l'attribut.

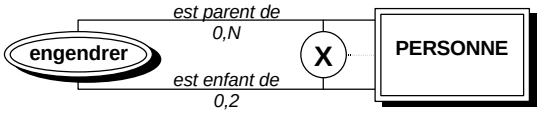
---

[11] *Remarque aux Merisiens, qui peuvent utiliser indifféremment l'ellipse ou le rectangle arrondi pour dessiner des relations : ici, il y a une distinction fondamentale entre une association et une action : le formalisme n'est pas indifférent.*

Les pattes

Les pattes sont les liens qui, dans un diagramme, relient entre eux objets, associations et actions. Recensons leurs divers types, afin de pouvoir les identifier au niveau du formalisme :

- **Objet-Objet** : il s'agit nécessairement d'un héritage. Ce type de lien est en fait une association non représentée par une ellipse, car elle n'a pas besoin de nom, et ses cardinalités sont figées (voir plus haut : associations). Sa représentation est ⟹, orientée vers l'objet parent.
- **Objet-Prédicat** : il s'agit, soit d'un prédicat sur un attribut de l'objet, auquel cas on utilise un trait simple (——), soit d'un héritage. Dans ce cas, sa représentation est ⟹, orientée vers le prédicat.
- **Objet-Association** : patte classique d'un MCD. Elle porte des cardinalités et doit être nommée avec un verbe (dont l'objet est le sujet). Ce nommage, qui peut surprendre les Merisiens (mais pas les adeptes de certaines autres méthodes) est pourtant indispensable dans le cas des relations réflexives :



Elle n'est fléchée (flèche simple ⟶) que pour indiquer un identifiant relatif, et pointe dans ce cas sur l'objet portant l'identifiant principal.

- **Objet-Action** ou **Prédicat-Action** : patte indiquant un attribut utilisé ou modifié par une action. Elle est fléchée (⟶), la pointe indiquant le sens du traitement (vers l'action pour une utilisation - source -, vers l'objet pour un résultat - cible -). On peut différencier la cible en la pointant avec ⟶⟶ (double flèche).
- **Association-Association** : il s'agit, soit d'un héritage, auquel cas la double flèche (⟹) pointe vers l'association parent, soit d'une association d'associations, qu'on symbolise avec une patte simplement fléchée (⟶), la pointe indiquant l'association recevant ainsi une patte supplémentaire (facultative ou obligatoire, selon la cardinalité). Elle correspond à une identification relative, l'association source ayant besoin de l'identifiant de l'association pointée pour être complètement identifiée. Elle est nommée (terme utilisé pour construire la phrase quand on parcourt le réseau dans le sens inverse de la flèche) et porte des cardinalités.
- **Association-Action** : tout comme Objet-Action, il s'agit d'utiliser ou de modifier un attribut de l'association. Même sémantique, même formalisme (⟶ ou ⟶⟶).
- **Association-Prédicat** : tout comme Objet-Prédicat, on a ici affaire à un héritage. Même sémantique, même formalisme (—— depuis l'association parent, ⟹ depuis l'association enfant).
- **Action-Action** : une action utilise ou modifie un attribut d'une autre action. Il est ici nécessaire de savoir si la patte indique l'utilisation par l'action cible d'un attribut de l'action source, ou de la modification d'un attribut de l'action cible par l'action source. L'option retenue est de formaliser le résultat d'une action par une patte à double flèche ⟶⟶.

Remarque importante : dans le cas d'un lien association-association correspondant à un héritage, ceci n'est valide que si l'association-enfant est reliée à un objet-enfant d'un objet-parent relié à l'association-parent (pour plus de clarté, voir le second exemple de MCD au début de l'article : Gérer⟹Suivre vs Gestionnaire⟹Employé).

Les contraintes

Un dernier point à aborder est celui des **contraintes** ensemblistes. Merise permet de les symboliser simplement[12]:

- Le premier groupe consiste en règles reliant des objets, des associations ou des pattes différents : inclusion –(I)–, exclusion –(X)–, totalité –(T)–, partition –(+)–, simultanéité –(S)–, unicité –(1)–.

[12] *Ne voulant pas refaire dans cet article un cours Merise complet, je renvoie les lecteurs qui ne connaissent pas ces concepts aux sources documentaires citées ci-après.*

- Le second groupe (aussi appelé contraintes de stabilité) s'applique directement à une relation, une patte ou un attribut. Le symbole se place sur la patte (relation définitive –Ⓓ– ou patte verrouillée –Ⓥ–) ou après l'attribut (propriété constante Ⓒ).
- D'ailleurs, sémantiquement, les cardinalités sont également des contraintes ensemblistes.

Ces contraintes ensemblistes masquent en fait des méthodes et/ou des actions : quand - par exemple - on essaie de créer une troisième occurrence de relation dont la cardinalité est (0,2), il y a bien rejet quelque part, un événement erreur généré par une action, ou un code retourné par une méthode.

Ces actions, qu'on peut considérer comme banalisées (car leur algorithme n'est pas rattaché à un ensemble particulier), seront, lors d'une mise en œuvre, programmées "en dur", leur définition faisant simplement partie du descriptif du SGBD. Ceci sera étudié plus en détail dans la seconde partie de cet article.

Conclusion de la 1ère partie

Nous avons donc vu que n'importe quel réseau sémantique, qu'il s'agisse de données, de traitements, d'actions... peut être modélisé dans un formalisme unique, avec cette particularité qu'on mélange parfois des entités et des modèles. C'est pourquoi la notion de métamodèle devient incontournable, et que la formalisation d'un tel réseau demande la mise en œuvre d'une nouvelle méthodologie, aucune méthode actuelle ne couvrant un champ aussi vaste[13].

Ceci amène à une question simple : si on peut tout rassembler dans un seul modèle, n'est-il pas possible de tout réaliser avec une seule base de données ? Ma réponse est oui, et implique les contraintes et avantages suivants dans un tel système :

- Les traitements sont stockés avec les données, donc pas de programmation (ou du moins limitée au descriptif des méthodes et actions) et portabilité totale possible.
- Il est nécessaire de disposer d'un moteur de navigation (SGBD) d'une part, et d'un moteur de traitement (inférences, algorithmes) d'autre part.
- Le grand nombre d'informations modifiables simultanément et indépendamment rend obligatoire l'usage d'un véritable système d'exploitation multitâche (ce qui élimine MS-DOS, Windows 3.x et - actuellement - Apple).
- Un langage de requêtes relationnel comme SQL est absolument inadapté à un véritable système entité-relation. Un nouveau standard est donc à définir.
- Enfin[14], comme il est impensable de modéliser un système en représentant l'intégralité des occurrences des entités n'appartenant pas à des populations, nous devrons travailler avec un méta-modèle permettant de définir les conteneurs.

Tout ceci sera étudié dans la seconde partie, et mis en application dans un prototype qui servira à réaliser un outil d'aide au choix d'un AGL. ▲

(à suivre)

Jean-Luc Blary

---

13 Avis aux amateurs...

14 En conclusion supplémentaire : je crains d'abuser légèrement des notes de bas de page. (Ah, vous aviez remarqué ?)

Au sommaire de la 2ème partie

- ✓ Définition d'un métamodèle permettant la génération automatique
- ✓ Mise en pratique du modèle sémantique universel sous forme d'un SGBD
- ✓ Perspectives d'avenir

Sources documentaires

MERISE, support de cours
ABOUHAIR G.
IBSI, Paris 1988, 1991, 1992

De la modélisation systémique au réseau sémantique
BRES P-A., ROCHFELD A., TABOURIER Y., SIBERTIN-BLANC C.
Conférence-débat de l'AFCET, Paris 15/11/1990

EPHIBASE, SGBD entité-relation orienté objet, dossiers de conception et prototype
BLARY J-L.
EPHITEQ, Caëstre 1989-1990, 1993, 1995, 1996

SGBD avancés : bases de données objet, déductives, réparties
GARDARIN G. VALDURIEZ P.
Eyrolles, Paris 1990

ODESYS, Outil d'aide à l'évolution du Système d'Information, dossiers de conception
BLARY J-L., THELLIEZ Ph.
EPHITEQ, Caëstre 1993-1996

Merise vers une modélisation orientée objet
MOREJON J.
Les Éditions d'Organisation, Paris 1994

MERISE & OBJETS, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1995

Réseaux de neurones
NADAL J-P.
Armand Colin, Paris 1993

MERISE, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1996

Moteurs de systèmes experts
VOYER R.
Eyrolles, Paris 1987

☛*Intéressés, critiques, puristes, adversaires, partisans... pour en savoir plus ou participer :*

- ☎ *0.660.602.702*
- 📠 *0.328.402.702*
- 💻 *jlblary@nordnet.fr*
- ✉ *EPHITEQ - Château Vallée - 59190 CAËSTRE*