Le réseau sémantique universel (2ème partie)

L'intégration de quelques concepts permet d'étendre considérablement les possibilités de modélisation. Voici la suite de l'analyse de cette intégration.
L'auteur vous renouvelle ici ses excuses pour la densité et la non-exhaustivité de cette étude (et pour les égratignures faites à MERISE), mais il s'agit d'une suite d'articles à considérer comme un condensé – et un résumé – d'une étude plus complète qui sera publiée dans un proche avenir.

Rappel de la 1ère partie

Nous avons vu dans la première partie de cet article (La Lettre n° 26) qu'il était possible d'utiliser un seul et unique symbolisme pour modéliser pratiquement n'importe quel système, ceci ayant été présenté avec les sept systèmes suivants :

- Modèle Conceptuel de Données (MCD) de Merise ;
- Modèle Conceptuel de Traitements (MCT) de Merise ;
- Système à Base de Connaissance (SBC) ;
- Tableur (considéré comme un croisement entre une Base de Données et un SBC) ;
- Système NeuroMimétique (SNM) ;
- Modèle de Traitements en Temps Réel (TTR) ;
- Programmation Orientée Objets (POO).

Bien que le symbolisme utilisé soit - à la base - celui des MCD de Merise, il a été sensiblement élargi, prenant en compte les concepts suivants :

- **Objets** (modèles, populations, occurrences, singularités) ;
- **Attributs** (propriétés simples, faits, tableaux, agrégats, méthodes ; appartenant à des objets, des associations, ou des actions) ;
- **Associations** (modèles, populations, particularités, héritages) ;
- **Actions** (les traitements) ;
- **Prédicats** (attributs contribuant à un héritage par spécialisation ou au déclenchement d'actions) ;
- **Pattes** (reliant entre eux objets, associations, actions, prédicats).

Notre objectif, maintenant, est d'élaborer un métamodèle - c'est-à-dire modéliser les modèles - qui sera capable de permettre la définition de tout modèle. Ce métamodèle jouera tout à la fois les rôles de dictionnaire, générateur, concepteur, etc.

À partir de ce métamodèle, l'étape suivante consistera à élaborer un modèle physique qui, implanté en tant que SGBD entité-relation, permettra de concevoir, prototyper et réaliser n'importe quelle application sans programmation.

Définition

Pour bien comprendre l'objectif actuel, il est indispensable de bien définir ce qu'est un métamodèle.

Dans un modèle, on trouve par exemple des objets, contenant des attributs. Dans un métamodèle, on trouvera la description d'un objet-type (modèle d'objet), stockée sous forme d'un ensemble de composants (entités "objet" et "attributs", reliées par des associations "posséder").

En résumé, un métamodèle est un modèle dans lequel sont définies des entités permettant de décrire un modèle, et ce selon le même mécanisme[1 et 2].

Enrichissement du symbolisme

L'élaboration d'un métamodèle prenant en compte des concepts très élargis, il est nécessaire et/ou utile de légèrement enrichir le symbolisme défini dans la première partie de ce dossier.

Héritage simple et multiple

L'héritage simple n'ayant été précédemment que survolé, et l'héritage multiple non abordé, nous allons maintenant en faire une analyse plus complète, car indispensable pour définir notre métamodèle.
Dans la pratique, ces notions ne figurent que dans deux des systèmes étudiés : le MCD et la POO, et ce avec des implémentations différentes :

- Dans un MCD, si un objet conceptuel hérite d'un autre objet conceptuel, on a toujours affaire à un même individu. De même, un objet conceptuel ne peut hériter de plusieurs objets que si ces derniers ont une même base sémantique, c'est-à-dire qu'ils héritent tous, à quelque degré que ce soit, d'un objet de base unique. Par exemple, un *Enseignant-Chercheur* pourra hériter des objets *Enseignant* et *Chercheur*, car ceux-ci héritent tous deux (héritage simple) de l'objet *Personne*. Dans tous les cas, ces quatre objets conceptuels ne constituent qu'un seul et unique individu, mais avec des vues différentes. On a bien affaire ici à un **héritage sémantique**.
- A contrario, en POO, une classe peut être constituée par héritage d'une ou plusieurs autres classes, même si celles-ci n'ont aucun rapport sémantique entre elles. Ceci n'est possible que parce qu'on ne définit en fait que des modèles, et que les objets générés seront toujours physiquement distincts. Par exemple, si on définit une classe C qui hérite des deux classes A et B, ceci permettra à un objet C d'être vu comme un objet A ou comme un objet B, mais il n'existera aucune occurrence de A ni de B pouvant être vue comme un objet C. Nous appellerons ce type d'héritage : **héritage génétique**, car l'objet héritier bénéficie des attributs de son ou ses parents, et reste un individu distinct. C'est d'ailleurs le cas entre une occurrence d'objet et son modèle.

Il s'agit donc bien de deux concepts distincts, et nous verrons plus loin qu'ils coexistent dans notre modélisation unifiée, sachant que l'héritage Merise concerne les objets et les individus, et l'héritage POO les modèles, notamment au niveau des agrégats.

Il est donc nécessaire de pouvoir les différencier au premier coup d'œil, et donc d'enrichir le symbolisme déjà mis en place :

- Un héritage sémantique reste symbolisé par une association (1,1)-(0,1) représentée par une flèche à double tracé (), orientée vers l'objet parent. En cas d'héritage multiple, il y a une flèche par parent.
- Un héritage génétique se différenciera de l'héritage sémantique par l'utilisation d'une flèche à double tracé également, mais doublement barré (), pour montrer que les individus sont différents, les cardinalités de ce type de lien étant également (1,1)-(0,1).

La contrainte d'égalité

Merise propose indifféremment l'usage du symbole –(s)– ou –(=)– pour symboliser une *contrainte de simultanéité*, c'est-à-dire présence simultanée d'au moins une occurrence de chacune des entités concernées. L'expérience a montré que le second symbole pouvait prêter à confusion.

Nous proposons donc de différencier la signification de ce symbole –(=)– : nous l'utiliserons désormais pour matérialiser une *contrainte d'égalité*, c'est-à-dire impliquant d'avoir de part et d'autre le même nombre d'occurrences (désolé pour les puristes de Merise qui l'ont choisi au lieu du premier).

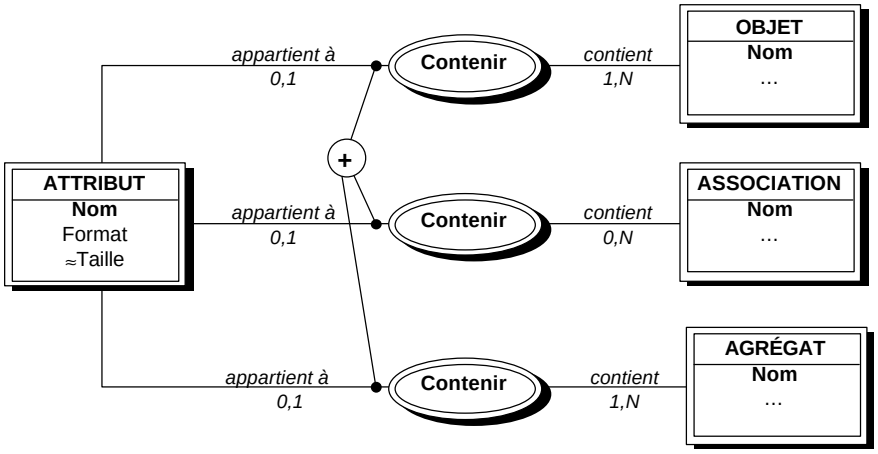
---

[1] *Dans le respect de Merise, qui est une méthode réflexive, c'est-à-dire qui peut se modéliser elle-même.*

[2] *…et je suis reparti pour vous submerger de notes de bas de page (mais c'est ça ou les apartés entre parenthèses) !*

Les pattes alternatives

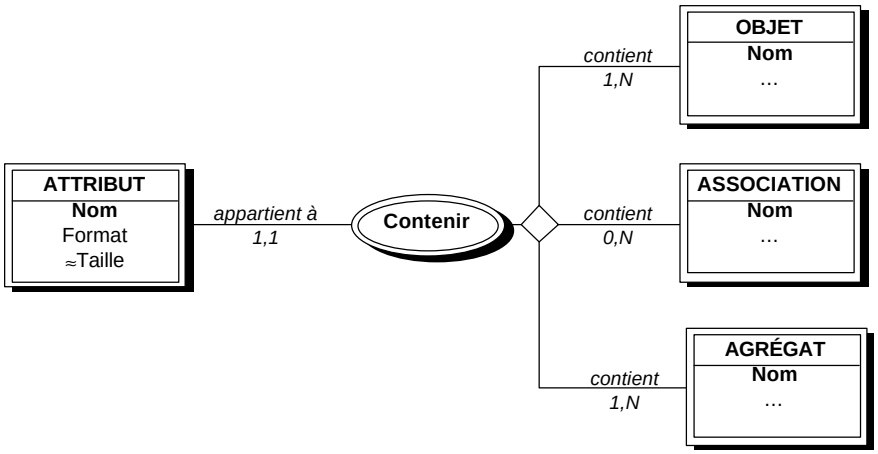
Sous ce nom "barbare" se cache une notion très simple, que nous allons illustrer immédiatement. Par exemple, sachant qu'un attribut appartient soit à un et un seul objet, soit à une et une seule association, soit à un et un seul agrégat, nous devrions symboliser cet ensemble comme ceci :



Mais en fait, les trois associations :

- sont sémantiquement équivalentes pour l'attribut ;
- contiendront des informations semblables ;
- une et une seule d'entre elles existera.

Il semblerait donc intéressant de mieux formaliser cet état de choses, et d'utiliser la représentation ci-dessous :



Ce symbolisme permet de mettre la véritable cardinalité du lien d'appartenance de l'attribut (1,1), de ne modéliser qu'une seule association, et d'éliminer le besoin d'une contrainte ensembliste explicite (la cardinalité (1,1) implique qu'un seul des trois objets de droite sera relié à l'attribut).

Bien sûr, il ne s'agit ici que d'une facilité de représentation, et il est toujours nécessaire de définir chacune des associations possibles, car il s'agit bien d'autant d'entités distinctes.

Le réseau récursif

Certaines associations relient les mêmes types d'objets, soit directement (OBJET hérite de OBJET), soit indirectement (ATTRIBUT appartient à TYPE, TYPE définit ATTRIBUT ...). Dans certains cas, il est impératif qu'un parcours au travers de ces associations n'aboutisse jamais sur un objet déjà rencontré dans ce même parcours : si A hérite de B, B hérite de C, en aucun cas C ne peut hériter de

A[3], car nous avons en fait une arborescence. La simple contrainte d'exclusion –(x)– entre les pattes n'interdisant que d'avoir le même individu de part et d'autre, il faut pouvoir être plus explicite.

Un second cas de figure est encore plus restrictif. En effet, si - par exemple, pour un héritage génétique - A hérite de B et C, C hérite de D, B hérite de E, il serait gênant que D hérite également de E, car on se retrouverait avec les mêmes attributs en double dans A[4]. Il faut donc dans ce cas interdire de pouvoir retrouver le même objet, même par un autre chemin, c'est-à-dire qu'il ne peut exister qu'un seul chemin pour aller de X à Y utilisant cette association, quel que soit le nombre d'étapes.

Nous utiliserons donc la notation –(x→)– (flèche simple) pour expliciter l'interdiction de boucler sur un chemin en allant toujours dans le même sens, et la notation –(x↔)– (flèche double) pour expliciter l'interdiction de revenir sur le même objet quel que soit le chemin parcouru.

Éventuellement, le même type de formalisme pourra servir avec d'autres contraintes que l'exclusion (notamment l'inclusion –(I)–).

Les clones

Il ne s'agit plus ici d'enrichir réellement le symbolisme, mais d'améliorer la lisibilité : au-delà d'un certain degré de complexité, on en vient à avoir des pattes qui se croisent et s'entrecroisent, transformant le schéma en sac de nœuds. Il serait donc intéressant, pour éviter cela, de prévoir une possibilité de duplication de certains objets.

Attention : il ne s'agit pas d'augmenter le nombre des objets, mais uniquement de les faire apparaître plusieurs fois dans le modèle. Comme il est indispensable que le lecteur du modèle soit informé de la chose, et ne s'étonne pas de voir apparemment plusieurs objets porter le même nom, nous ajouterons à tout objet dupliqué un symbole[5] :
❑ dans l'original, pour signaler qu'il a été dupliqué ;
❑ dans chaque copie, pour signaler qu'il s'agit d'un duplicata.

Pour être plus lisible encore, il est inutile de répéter les attributs d'un objet dans un clone : le symbole de copie peut aussi signifier : pour plus d'information, voir l'original.

Le métamodèle

Tout le formalisme nécessaire étant maintenant défini, nous pouvons commencer à élaborer ce modèle étape par étape, en prenant les différents types d'entités une par une et en les raccrochant à ce qui aura été défini.

Le lecteur observateur remarquera dans ce qui suit qu'en fait, parmi les différents systèmes analysés dans la première partie de ce dossier, les concepts modélisés proviennent en quasi-totalité de trois d'entre eux seulement : le MCD Merise, le Système à Base de Connaissance (ou système-expert), et la Programmation Orientée Objet.

Les attributs

Un attribut s'identifie par son nom, et il a un format (entier, flottant, caractère, binaire, blob[6],...) et éventuellement une taille. S'il est de type agrégat, il contient d'autres attributs ; on a donc une relation récursive **contenir**, dont les pattes sont nommées *contient* et *contenu dans*. Dans cette association, on placera des attributs tels que le nom de l'attribut dans l'agrégat, le nombre d'occurrences et le numéro de la première occurrence si on inclut un tableau, la facultativité de l'attribut.

---

[3] *Nous parlons bien entendu d'héritage sémantique ou génétique, et pas d'héritage testamentaire (quoique même dans ce cas... si A et B se désignent mutuellement comme héritiers, un seul des deux pourra hériter au final...).*

[4] *Ceci est autorisé en C++, mais implique une gymnastique pour définir si les attributs en double sont sémantiquement les mêmes ou pas, et dans ce dernier cas il faut procéder à un renommage. Bjarne Stroustrup (créateur du C++) lui-même déconseille ceci.*

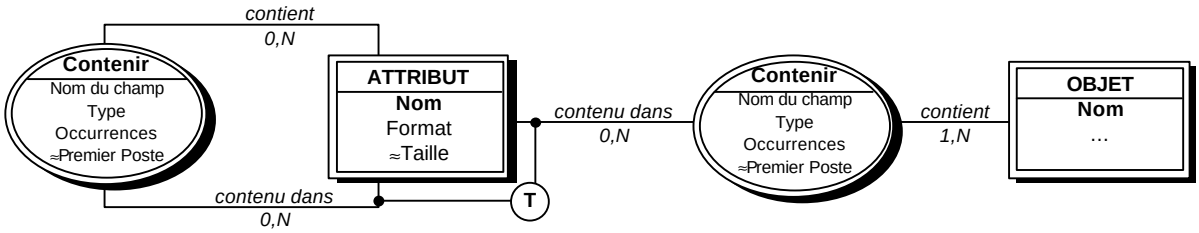
[5] *Certains AGL utilisent déjà cette technique, et pour les mêmes raisons (mais pas nécessairement le même symbolisme, ce qui est secondaire).*

[6] *BLOB = Binary Large Object, pouvant être un texte, un son, une image, etc.*

Un objet pouvant être considéré - du point de vue des attributs - comme un agrégat, on trouvera également ces attributs dans l'association objet-attribut.

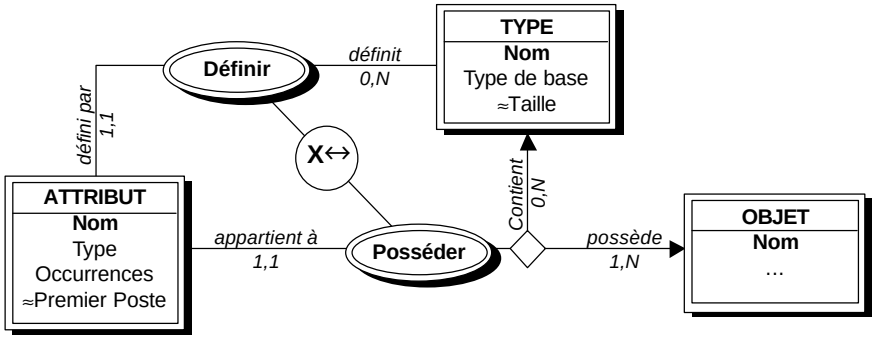
Un attribut n'ayant d'utilité que s'il est utilisé quelque part, il doit obligatoirement être associé à au moins un agrégat ou un objet (ou une association, nous compléterons plus loin).

Il semble donc que l'amorce du métamodèle devrait être :



Mais si on regarde d'un peu plus près, on s'aperçoit qu'on a déjà tout faux : logiquement, rien n'empêche d'utiliser dans un objet ou un agrégat deux (ou plus) attributs semblables[7] (exemple : *adresse de livraison* et *adresse de facturation*), ce que ne permet pas le modèle ci-dessus.

L'attribut *Nom du champ* est en fait un identifiant relatif [8]de l'attribut par rapport à son contenant. C'est pourquoi nous devons distinguer les notions d'attribut "physique" et d'attribut "type" (équivalent sémantique du *typedef*, *struct* ou *class* du C++, ou du *type* ou *class* du Pascal). Nous appellerons tout simplement celui-ci **TYPE**, et nous modéliserons l'amorce du métamodèle, finalement plus simple que le précédent, comme ceci :



Cette première ébauche appelle, toujours au niveau des attributs, plusieurs compléments.

a) Tout d'abord, dans un grand nombre de cas, un attribut simple (ou un fait) ne peut contenir n'importe quoi : son contenu fait partie d'une liste de valeurs et/ou exclut certaines valeurs, est compris dans une ou plusieurs fourchettes, doit avoir un format précis (masque), etc. Nous définirons donc un autre type d'objet pour le métamodèle : la **VALEUR**.

   Tout individu **VALEUR** contient un ensemble de données permettant de valider le contenu d'un attribut, et n'a (en principe) d'utilité que s'il est effectivement associé à au moins un attribut (association **Vérifier**, dont les pattes sont nommées *vérifie* et *vérifié par*).

   Nous verrons en détail la définition de cette entité lors de la mise en œuvre pratique[9].

b) Ensuite, il est utile d'introduire ici la notion d'héritage génétique entre types (un type agrégat *A* contenant un autre agrégat *B* nécessitera de référencer les attributs de ce dernier en *A.B.x*, alors qu'un agrégat héritant d'un autre agrégat disposera directement d'attributs identiques : *A.x*).

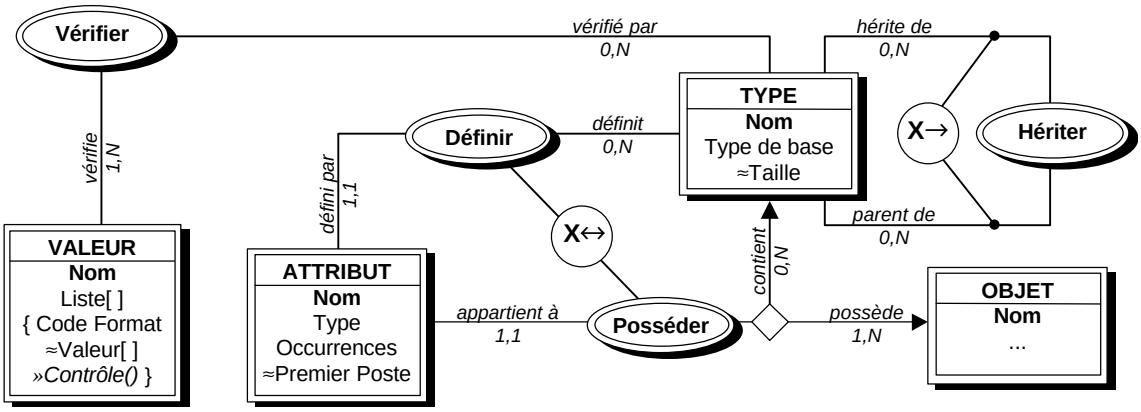
---

[7] *"Semblable est ici utilisé dans son sens mathématique : deux triangles semblables ont exactement la même forme, mais peuvent être de tailles différentes, donc non identiques). Deux attributs semblables ont les mêmes caractéristiques, mais pas le même usage.*

[8] *De par leur implémentation, certains AGL du commerce ne permettent pas cette identification relative, tant pour les atributs que pour les associations, obligeant ainsi à donner des noms différents à des concepts parfois semblables. Nous déplorons cette restriction, qui ne fait pas partie de Merise.*

[9] *Notons cependant immédiatement qu'une étude complète entraînerait des associations et des contraintes entre objets VALEUR : un paramètre défini selon un certain type, avec des valeurs définies, peut très bien correspondre avec un attribut d'un type légèrement différent, si son format de base est compatible et que ses valeurs possibles sont incluses dans celles du paramètre…*

Nous en arrivons donc à une deuxième ébauche de métamodèle :



On utilise ici pour définir l'héritage entre types une relation normale, et non pas le symbole d'héritage, car nous sommes dans un métamodèle, donc nous définissons ce qui permettra de gérer les modèles : la relation **TYPE hérite de TYPE** signifie que les attributs du type parent feront également partie du type héritier.

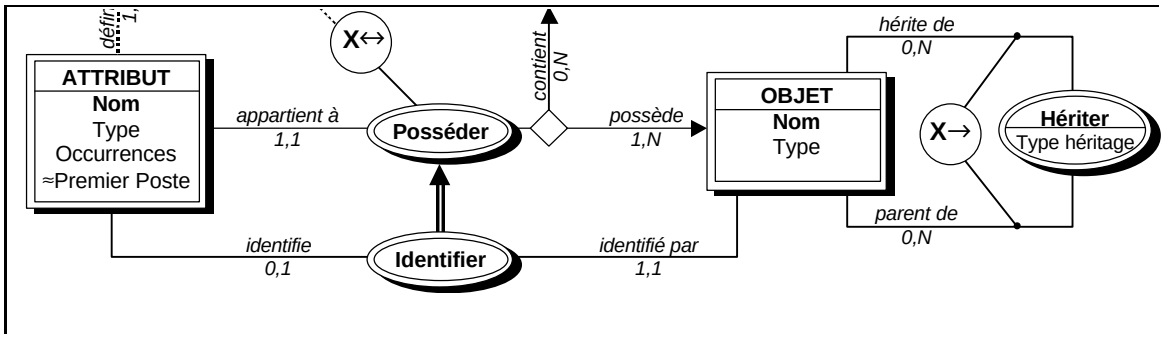
Anecdotiquement, notez que l'objet **VALEUR** nous fait ici la totale quand aux enrichissements sur les attributs : nous avons un tableau d'agrégats Liste[ ]{...} contenant notamment une méthode facultative *»Contrôle()* !

Par ailleurs, on constatera que les méthodes, bien qu'appartenant à des objets, des agrégats ou des associations, tout comme les attributs "données", n'ont pas été prises en compte ici. C'est normal, puisqu'il s'agit en fait d'une forme d'action : contrairement à une action "ordinaire", dont les entrées et sorties sont parfaitement définies, une méthode appartient à une entité dans ce sens qu'une partie de ses entrées/sorties ne concerne que des attributs de cette entité, le reste étant paramétré (paramètres en entrée ou résultat seront associés à diverses entités, selon les cas). Ceci sera donc étudié après les actions.

Les objets

Tout objet a un nom et un type (modèle, occurrence, singularité). Il contient un ou plusieurs attributs, dont l'un est l'identifiant (par rapport à Merise standard, nous n'accepterons qu'un seul attribut comme identifiant, quitte à utiliser un agrégat).

Il peut également hériter des attributs et relations d'un ou plusieurs autres objets, cet héritage pouvant être sémantique ou génétique. Ceci nous permet de compléter notre métamodèle comme suit[10] :



Évidemment, la contrainte d'exclusion de la relation **OBJET hérite de OBJET** dépend du type d'héritage :  pour un héritage sémantique, pour un héritage génétique. Une représentation complète au niveau du symbolisme est ici impossible, puisque concernant un chemin complet.

[10] *Les pointillés indiquent qu'on ne montre qu'une partie du schéma.*

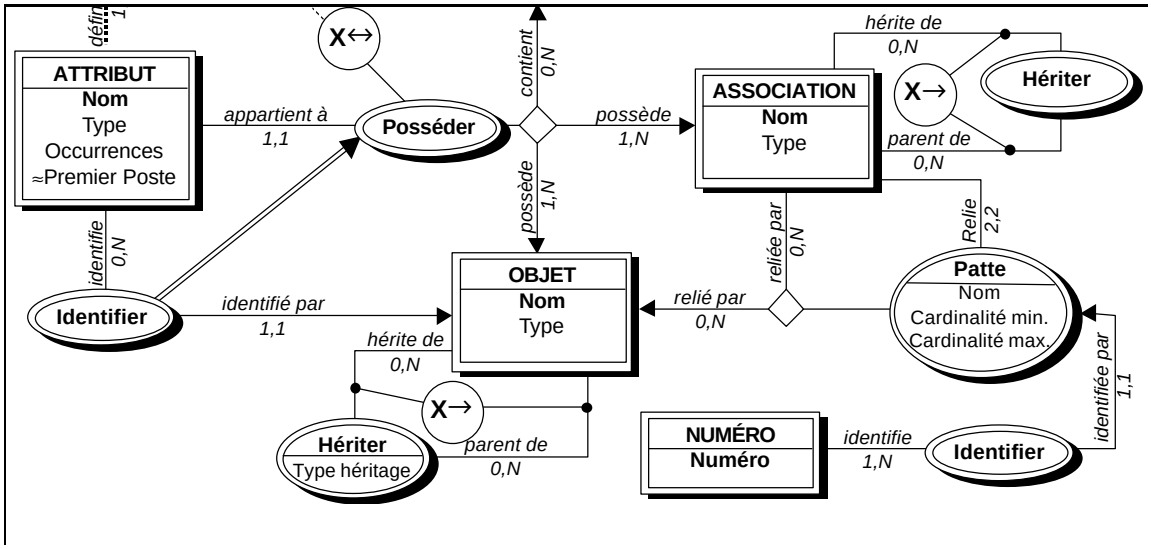
Les associations et les pattes

Comme les objets, les associations ont un nom et un type. Elles peuvent contenir des attributs, et chacune est reliée par deux pattes (ayant un nom et des cardinalités) à deux objets et/ou associations. Elles peuvent hériter d'une ou plusieurs autres associations (sous réserve qu'elles relient les mêmes objets et dérivent toutes d'une même association de base : c'est la même restriction que pour les objets).

La spécificité d'une association, c'est qu'elle a toujours deux pattes, pas plus, pas moins[11]. De plus, ces pattes peuvent toutes deux relier le même objet (association réflexive) et/ou porter le même nom : par exemple l'association associé entre deux personnes est symétrique : si Pierre est associé de Paul, on a en sens inverse Paul est associé de Pierre[12].

L'association Patte peut donc exister en deux exemplaires entre une même association et un même objet ou association. Ceci est interdit puisque tout identifiant doit être unique. Pour permettre la modélisation, nous allons donc devoir utiliser un objet fictif différenciant ces pattes. Sa population se composera en tout et pour tout de deux individus, nommés 1 et 2 (ou A et B, Gauche et Droite, Vrai et Faux… ceci est sans importance, du moment qu'il en existe exactement deux).

Ce qui nous permet d'enrichir le métamodèle (après une petite réorganisation spatiale) ainsi :



Il serait intéressant de pouvoir prendre en compte les pattes alternatives dans ce métamodèle (pour pouvoir ne faire qu'une seule définition), mais nous garderons ceci pour une étude ultérieure plus approfondie, donc hors du cadre de cet article.

Les prédicats

Il existe deux sortes de prédicats : ceux qui déterminent les héritages sémantiques par spécialisation, et ceux qui permettent de déclencher des méthodes et actions.

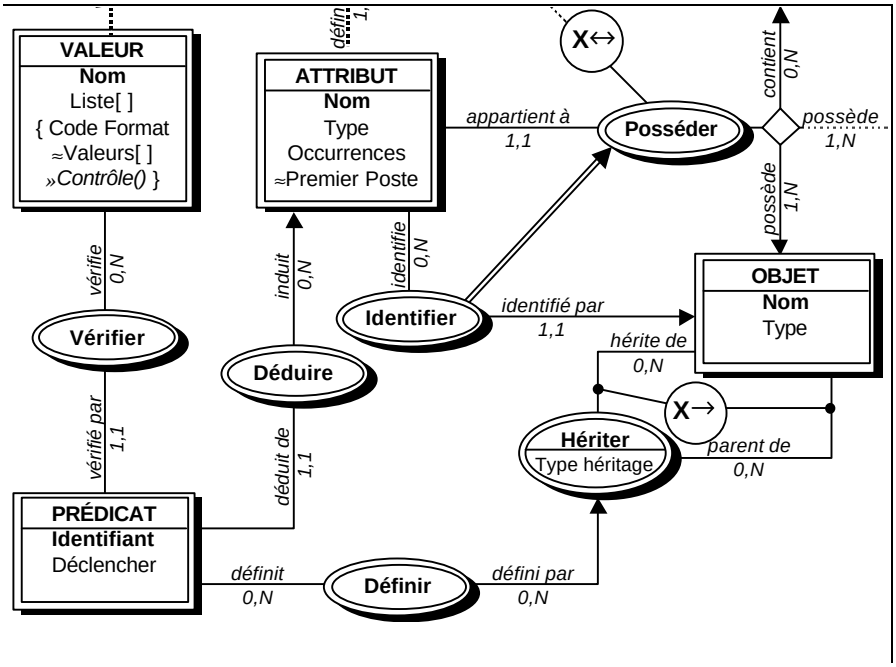
Le second cas étant indissociable des méthodes et actions, occupons-nous ici uniquement du premier cas : un ou plusieurs attributs déterminent, selon leur(s) valeur(s), à quelle classe se rattache un objet.

Dans notre métamodèle, nous allons donc définir un nouveau type d'objet : **PRÉDICAT**, relié d'une part à **ATTRIBUT** et d'autre part à l'association **Hériter** :

---

11 Nous avons montré dans la 1ère partie que toute association pouvait être réduite à 2 pattes, en utilisant les associations d'associations.

12 En théorie, on pourrait même avoir une association réflexive symétrique sur le même objet…



Les cardinalités s'expliquent aisément : un héritage n'est pas nécessairement défini par un prédicat ; a contrario, plusieurs attributs – et donc autant de prédicats – peuvent être nécessaires[13].

L'association **Vérifier** avec **VALEUR**, qui permet d'associer une liste de valeurs à un prédicat, est sémantiquement légèrement différente de son homonyme avec **TYPE**, car elle définit en fait un sous-ensemble des valeurs associées au type de l'attribut. Les occurrences de **VALEUR** concernées ne sont donc pas identiques, mais il existe entre elles une contrainte, dans le détail de laquelle nous n'entrerons pas maintenant[14].

Les contraintes ensemblistes

Arrivés à ce point, nous avons à peu près parcouru l'ensemble des concepts statiques. Je dis bien à peu près, car il y manque notamment les diverses contraintes ensemblistes (inclusion, exclusion, totalité, unicité, etc.). Celles qui mettent en cause des associations ne seront pas traitées ici[15].

Certaines contraintes (contraintes dites "de stabilité") peuvent cependant être prises en compte immédiatement :

- Une patte verrouillée est définie par un attribut à ajouter dans l'association **Patte**.
- Un attribut constant (non modifiable) est défini par un attribut à ajouter dans l'objet **ATTRIBUT**.
- Une relation définitive (dont toutes les occurrences doivent être créées en un seul processus) est définie par un attribut à ajouter dans l'objet **ASSOCIATION**.

Les actions

Nous arrivons maintenant à une partie légèrement plus complexe. Jusqu'ici, nous n'avons modélisé que des données, c'est-à-dire des choses relevant exclusivement du domaine "statique". Maintenant, nous nous attaquons aux traitements.

Nous pouvons décomposer les actions en trois familles, que nous étudierons l'une après l'autre :

- les **actions** proprement dites, correspondant à des règles de gestion, et dont l'ensemble des données manipulées et des règles de déclenchement est parfaitement défini ;
- les **méthodes**, actions dont ne sont conceptuellement définis que des liens avec des données appartenant à un objet (ou agrégat, ou association…), qui peuvent avoir des paramètres en

---

13 En théorie, une synchronisation est également nécessaire, mais nous ne nous en occuperons pas ici.

14 Les amateurs pourront essayer de modéliser cette contrainte (objets "objet valeur" associés à des objets "attribut valeur", etc.). Réponse dans la 3ème partie de ce dossier.

15 Mais le seront dans l'ouvrage plus complet à venir.

entrée et/ou renvoyer une donnée en sortie, paramètres et valeur renvoyée ne correspondant pas à des attributs connus lors de la définition.

- Les **routines**, méthodes complètement paramétrées n'ayant aucun lien conceptuel en-dehors du format de leurs entrées-sorties – c'est-à-dire non liées à un objet spécifique.

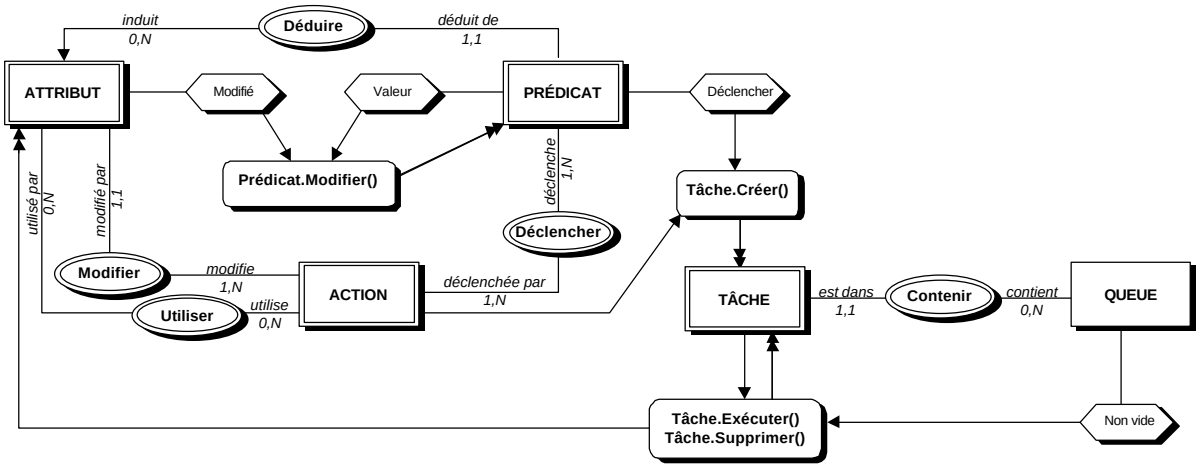
Les actions ordinaires

Rappelons que si les objets et associations peuvent constituer des occurrences ou des populations issues de modèles, il n'en est pas de même des actions[16], car toute action associée à des attributs d'un modèle est automatiquement applicable à chaque individu issu de ce modèle et n'est donc pas dupliquée.

Par contre, un attribut utilisé comme prédicat a une action individuelle. Pour mieux comprendre ceci, détaillons le fonctionnement d'un traitement dans le mode prédicat-action :

1) Un attribut d'un objet est modifié suite à l'exécution d'une action.
2) Si cet attribut fait l'objet d'un ou plusieurs prédicats[17], chacun d'eux est analysé pour vérifier si la nouvelle valeur a un impact sur une ou plusieurs actions (par exemple, si un prédicat déclenche une action avec la condition >100, la modification de la valeur de l'attribut de 80 à 90 n'aura aucun effet : on ne déclenche une action qu'à bon escient).
3) Si c'est le cas, un objet **TÂCHE** est créé (ou mis à jour dans le cas d'une action à plusieurs prédicats synchronisés) pour chaque action, et associé aux individus dont il utilise ou met à jour des attributs. Si l'ensemble des conditions de déclenchement sont remplies, cet objet est mis en queue (création d'une association avec une singularité[18] **QUEUE**, en tenant compte de la priorité de l'action).
4) Quand la tâche parvient en tête de liste, elle est exécutée puis supprimée (le résultat de son exécution pouvant déclencher l'activation d'autres actions).

Essayons donc de modéliser ce cheminement[19] au moyen d'objets, associations, prédicats et actions :



Comme vous pouvez le constater, une fois qu'un traitement est bien décrit, rien de plus simple que de le modéliser avec notre formalisme unifié, c'est-à-dire MCD Merise enrichi. Il ne reste plus, en effet, qu'à compléter le premier jet ci-dessus pour l'intégrer au métamodèle, après avoir répondu à quelques questions complémentaires :

- Quels sont les attributs de ces entités ?
- Le **PRÉDICAT** ci-dessus peut-il être conceptuellement fusionné avec le **PRÉDICAT** définissant l'héritage par spécialisation ?
- Si un métamodèle d'action contient des actions, comment celles-ci seront-elles définies et gérées dans la pratique ?

---

16 Si l'objet ACTION défini ici est un modèle, c'est parce que nous sommes dans un méta-modèle.

17 En principe, toute information devant avoir une origine et une utilité, tout attribut devrait être prédicat de quelque chose, la chaîne s'achevant sur un effecteur (par exemple, un libellé modifié déclenchera l'affichage de son contenu sur l'écran de l'utilisateur…).

18 Voir 1ère partie : une singularité est un objet dont la population est limitée à un et un seul individu, qui n'a donc besoin d'aucun identifiant en dehors de son type.

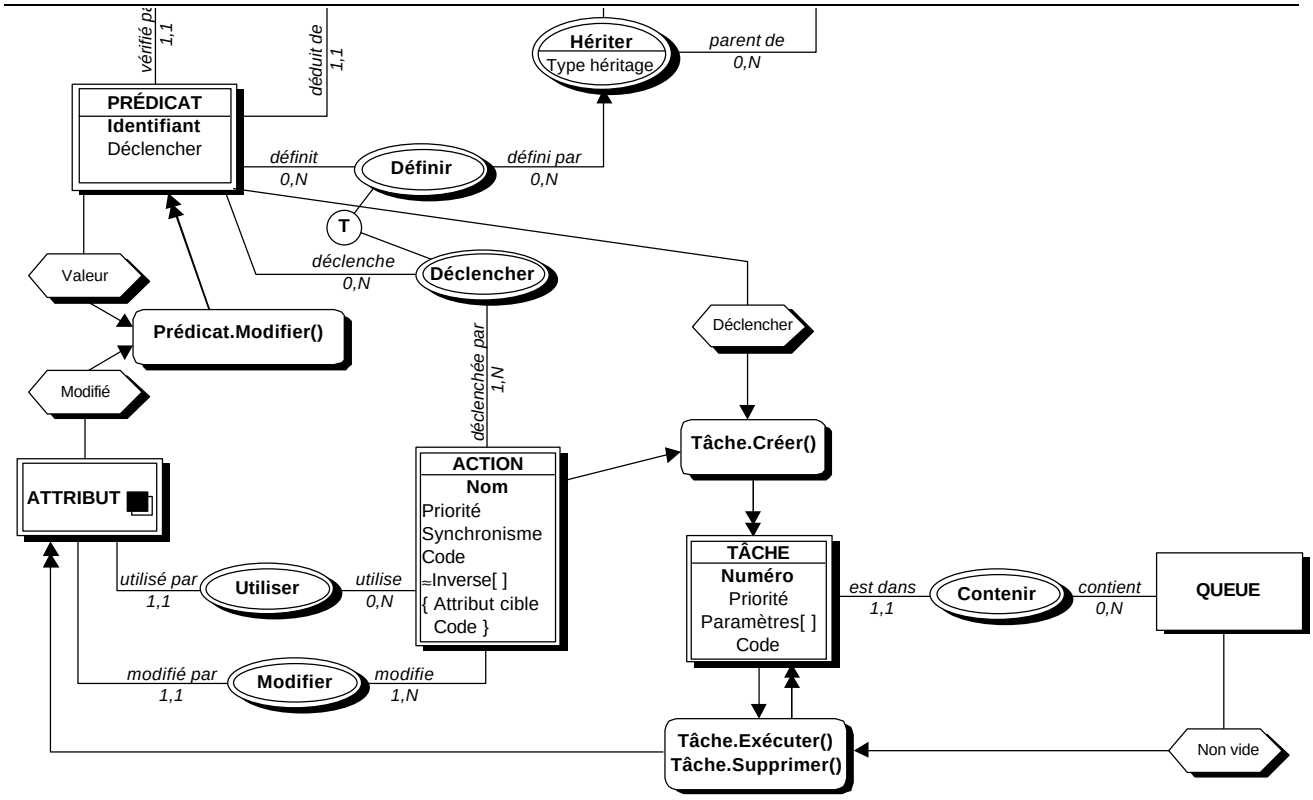
19 Tout le monde a évidemment reconnu le principe de fonctionnement d'un moteur d'inférence de Système à Base de Connaissances.

- À quoi servent les priorités ?

Essayons de répondre à ces questions :

- Une action n'a que peu d'attributs : son identifiant ; son code source[20] (qu'à la saisie une méta-action contrôle et utilise pour générer les liens avec les entités impliquées) ; éventuellement des codes source de fonctions inverses (pour permettre une recherche à partir du résultat voulu) ; une synchronisation (pour les déclenchements dépendant de plusieurs paramètres) ; une priorité ; et c'est à peu près tout, si on excepte la possibilité d'attributs internes (paramètres constants et/ou données intermédiaires utilisées lors de l'appel de méthodes et/ou routines).
- Une tâche reçoit un certain nombre d'attributs, qu'il faudrait conceptuellement définir comme des associations avec l'ensemble des informations nécessaires à son déroulement : attributs manipulés, code, etc. Pour ne pas compliquer le modèle, nous nous contenterons, en incorporant le schéma ci-dessus au métamodèle, de spécifier ces attributs.
- Un prédicat n'a qu'un identifiant relatif par rapport à l'attribut qu'il représente et un attribut booléen indiquant que la ou les actions associées doivent être déclenchées. Par contre, il est associé à un objet **VALEUR**, qui permet de connaître les conditions de déclenchement. Nous retrouvons effectivement les caractéristiques du prédicat d'héritage.
- Quant aux actions présentes dans le schéma ci-dessus, elles seront directement programmées.
- La queue, elle, n'a besoin d'aucun attribut.
- Les priorités permettent de gérer l'importance des actions, notamment dans un contexte temps réel : toutes les actions de priorité zéro seront exécutées avant toutes les autres, c'est-à-dire en principe immédiatement[21].

Complétons le modèle obtenu et intégrons-le au métamodèle en cours :



Ce schéma appelle quelques commentaires :

- Pour commencer, certains objecteront que **ACTION** et **TÂCHE** ne sont pas tout-à-fait conceptuels. C'est vrai, mais entrer dans le détail d'une conceptualisation complète augmenterait fortement le volume du présent article, sans ajouter grand-chose à sa compréhension. En d'autres termes, l'imperfection est volontaire.

---

20 Pourquoi pas en Java ?

21 Ce qui est fondamental si on veut pouvoir faire du temps réel.

- Nous avons mis qu'un attribut ne peut être modifié que par une et une seule action. Ceci constitue une condition nécessaire pour éviter les risques de non-intégrité[22].
- Il a été dit qu'une action pouvait contenir des attributs de travail. Nous intégrerons cette association dans le métamodèle complet (voir à la fin de cet article).
- La contrainte de totalité entre Définir et Déclencher nous dit qu'un prédicat sert obligatoirement à quelque chose…
- Les actions semblent être constituées de méthodes ? Pourquoi pas ? Pour plus de détails sur l'implémentation des méthodes, voir ci-dessous.
- Que se passe-t-il si une action (déclenchée 2 fois) modifie deux fois un attribut avant qu'une action subséquente ait pu impacter la première modification ? Voyons avec un exemple : si l'attribut passe de 80 à 110, puis à 90 :
  * si l'action déclenchée devait déclencher une alarme pour dépassement de la valeur 100, il faudrait logiquement l'annuler ;
  * si elle est simplement chargée de totaliser le nombre de dépassements de la valeur 100, elle doit être maintenue ;

  Bref, tout dépend du contexte, et ce cas sera étudié dans un autre ouvrage à venir.

Une question posée page précédente et à laquelle nous n'avons pas encore répondu : comment modéliser une action qui utiliserait d'autres actions ? Eh bien, c'est simple : il faut d'une part définir des objets recevant les résultats intermédiaires, et d'autre part que les sous-actions soient des méthodes ou des routines.

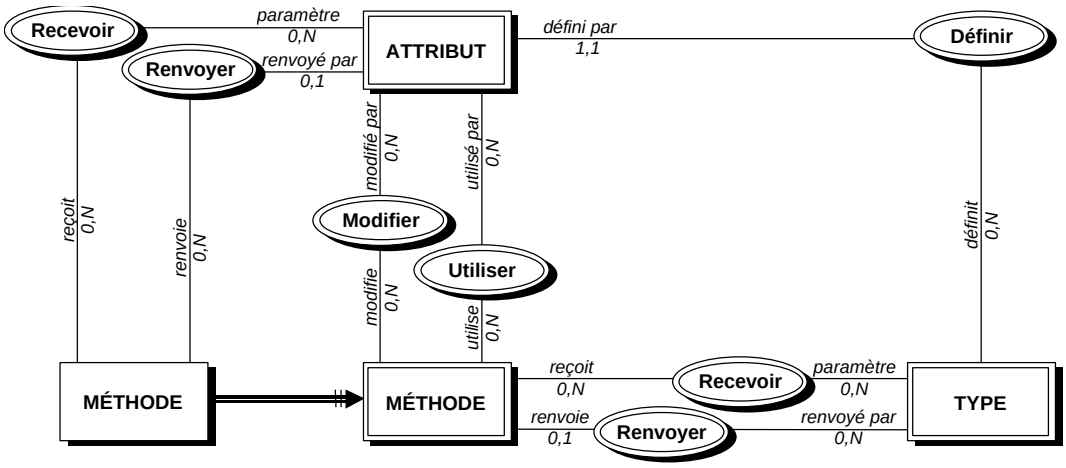
Les méthodes et routines

Maintenant que nous savons modéliser les actions, il est possible d'analyser les variantes que constituent une méthode ou une routine.

Qu'est-ce qui différencie une méthode ou une routine d'une action ? Simplement le fait que toutes ses entrées-sorties ne sont pas rattachées à des attributs précis lors de sa définition : on n'en connaît que le format. Ceci implique donc qu'on ne peut associer une méthode (partiellement) ou une routine (entièrement) à des attributs – via des prédicats –, mais seulement à des types.

Ce n'est que dans un deuxième temps, quand on définira une utilisation pour cette méthode ou routine, qu'on lui associera réellement des attributs. À la différence des actions, méthodes et routines sont donc prédéfinies comme modèles, les individus étant des occurrences.

Voyons ceci à l'aide d'une première modélisation :



Commentons ceci :

- Une méthode-type peut recevoir des paramètres et renvoyer une valeur. Elle peut également utiliser et/ou modifier des attributs dans un ou plusieurs objets (une routine, elle, n'a aucune association **Modifier** ni **Utiliser**).

---

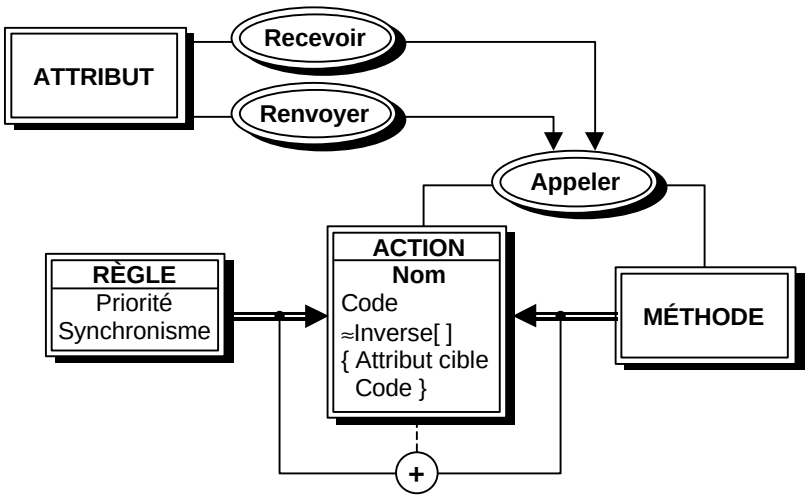
22 Condition loin d'être toujours respectée en informatique "traditionnelle"…

- Dans une occurrence de méthode, il faut remplacer les types par des attributs, qui doivent être de type (et de valeurs possibles) compatibles. Les occurrences de **Recevoir** et **Renvoyer**, pour chaque occurrence de méthode, doivent correspondre individu par individu avec celles de son modèle.
- Il faut également tenir compte (ce qui n'a pas été fait ici) que plusieurs paramètres peuvent être du même type, et qu'un même attribut peut servir pour plusieurs paramètres : ceci nécessite un identifiant complémentaire pour les associations **Recevoir**.
- Nous retrouvons les associations **Modifier/Utiliser ATTRIBUT** déjà définies pour les actions.

Le bilan de cette modélisation ? C'est qu'une méthode ou une routine peut être considérée comme une variété d'action pouvant posséder des paramètres et/ou renvoyer une valeur, et qui n'est déclenchée que lors de l'exécution d'une action/méthode/routine appelante[23]. Comme elle n'est déclenchée que par une action ou une méthode déjà en cours d'exécution, il est inutile de lui définir une priorité, puisqu'elle doit être exécutée immédiatement, sous peine d'interrompre l'action en cours.

De plus, une occurrence de méthode n'ayant d'existence que dans la mesure où elle est appelée par une action (le modèle définit la méthode, l'occurrence est son utilisation), on n'a pas affaire dans le métamodèle à un objet, mais bien à une association.

Nous allons donc compléter comme suit la définition d'un objet **ACTION**, une action ordinaire devenant une **RÈGLE** (de par son rôle en tant que règle de gestion ou règle d'un Système à Base de Connaissances), et les occurrences de méthodes étant tout simplement l'association **Appeler**, sur laquelle viendront se greffer **Recevoir/Renvoyer ATTRIBUT** :



Il s'agit bien ici d'héritage sémantique, sachant que toute action est soit une règle, soit une méthode, mais pas les deux, et que :

- Une règle a des prédicats et des attributs cibles, pas une méthode.
- Une méthode peut avoir des paramètres et/ou un résultat en retour, et est appelée par une règle ou une méthode.

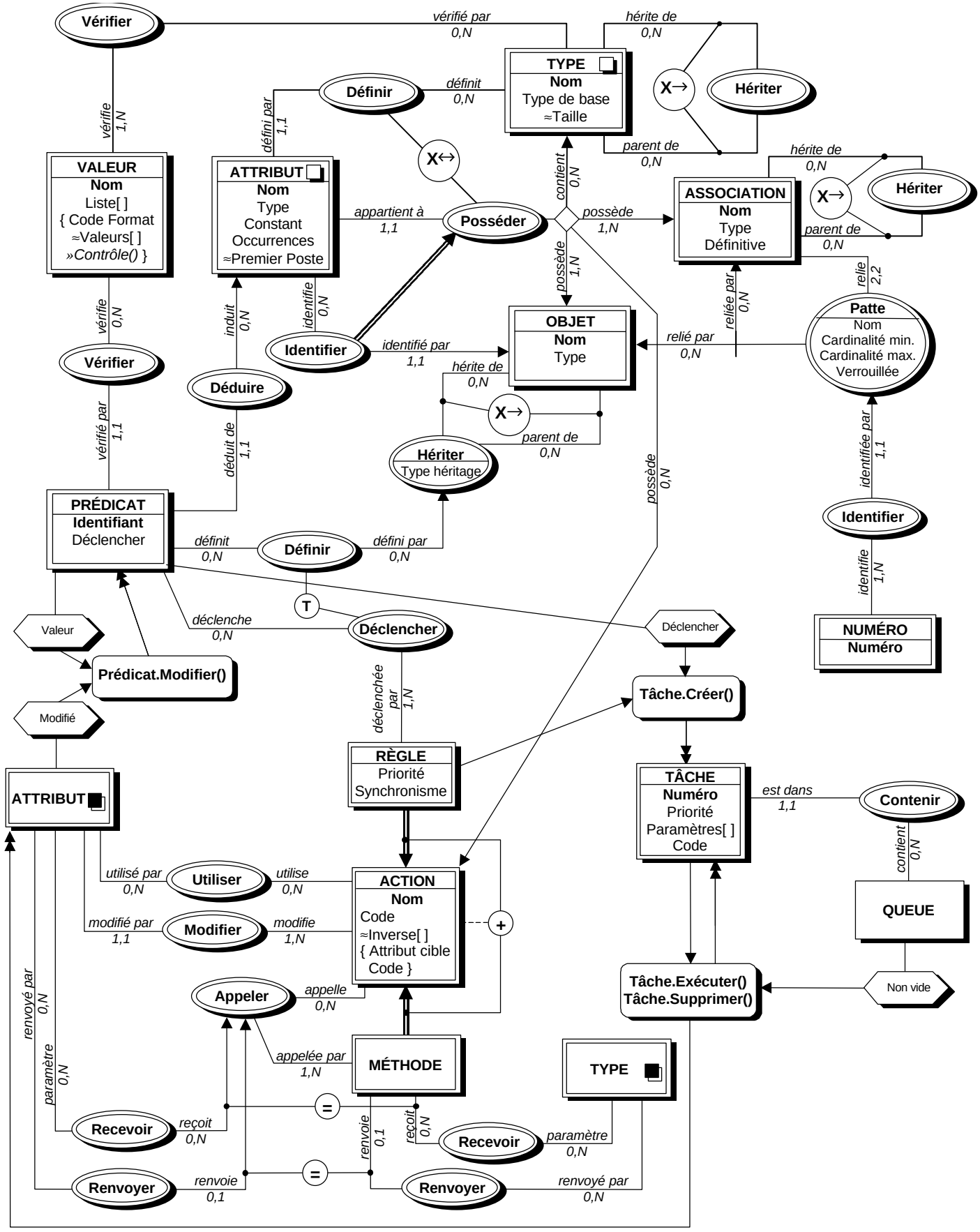
Finalement, c'est plutôt simple, non ?

Il est cependant recommandé, dans ce style de conception, de limiter au maximum l'arborescence des appels à des méthodes et/ou routines en définissant plusieurs règles simples de préférence à une règle complexe: plus une action s'exécute rapidement, mieux c'est.

23 Sachant qu'une action et une méthode peuvent appeler des méthodes et/ou des routines, mais qu'une routine ne peut appeler que des routines.

Synthèse

Après avoir intégré les méthodes, nous avons donc obtenu un métamodèle – incomplet, on le sait, mais cependant significatif – qui est celui-ci :



Et voici la liste des attributs :

Nom	Description	Format
ACTION.Code	Descriptif du traitement à effectuer.	Code source compilable et/ou interprétable (Java ?)
ACTION.Inverse[ ]	Tableau des traitements inverses possible (retrouver une valeur origine en connaissant le résultat). Autant de postes possibles que d'attributs en entrée.	Tableau d'agrégats
ACTION.Inverse[ ].Attribut cible	Identifie l'attribut source devenu cible dans le traitement inverse (pour une méthode, la définition cible un type).	Pointeur
ACTION.Inverse[ ].Code	Descriptif du traitement inverse à effectuer.	Code source compilable et/ou interprétable (Java ?)
ACTION.Nom	Nom de la règle de gestion.	Texte
ASSOCIATION.Définitive	Si toute la population vs objet commun doit être créée en une fois.	Booléen
ASSOCIATION.Nom	Nom "Merise" de l'association.	Texte
ASSOCIATION.Type	Modèle, occurrence, particularité.	Code
ATTRIBUT.Constant	Si attribut non modifiable.	Booléen
ATTRIBUT.Nom	Nom de la donnée dans l'objet ou agrégat qui la contient.	Texte
ATTRIBUT.Occurrences	Nombre de postes si l'attribut est un tableau (min. 1 = pas un tableau).	Numérique
ATTRIBUT.Premier Poste	Numéro d'ordre du 1er poste dans un tableau (peut être différent de 0 ou 1).	Numérique
ATTRIBUT.Type	Attribut obligatoirement renseigné ou pas.	Booléen
Hériter.Type héritage	Héritage sémantique ou génétique.	Booléen
OBJET.Nom	Nom de l'objet.	Texte
OBJET.Type	Modèle, occurrence, singularité.	Code
Patte.Cardinalité max.	Nombre maximal d'occurrences possibles pour que le système soit cohérent.	Numérique
Patte.Cardinalité min.	Nombre minimal d'occurrences possibles pour que le système soit cohérent.	Numérique
Patte.Nom	Nom de la patte, utilisé quand on vient de l'objet ou association connecté.	Texte
Patte.Numéro	Différencie les pattes dans le cas d'une association réflexive (dans le modèle : NUMÉRO.Numéro).	2 valeurs
Patte.Verrouillée	Si l'association ne peut être supprimée sans que l'objet ou association rattaché soit supprimé également.	Booléen
PRÉDICAT.Déclencher	Indique si la ou les actions dépendantes de ce prédicat devraient être déclenchées ou si l'héritage spécialisé est valide pour l'objet.	Booléen
PRÉDICAT.Identifiant	Permet de différencier les différents prédicats concernant un même attribut.	Texte
RÈGLE.Priorité	Urgence d'exécution.	Numérique

Nom	Description	Format
RÈGLE.Synchronisme	Combinaison des prédicats pour permettre un déclenchement.	Formule booléenne
TÂCHE.Code	Descriptif du traitement à effectuer.	Code compilé et/ou pré-interprété
TÂCHE.Numéro	Identifiant de la fonction dans la queue.	Quelconque
TÂCHE.Paramètres	Valeurs en entrée et destination du résultat.	Tableau/Agrégat
TÂCHE.Priorité	Urgence d'exécution (hérité de l'action).	Numérique
TYPE.Nom	Nom du type.	Texte
TYPE.Taille	Taille de la donnée (selon type de base)	Numérique
TYPE.Type de base	Format de base de la donnée : (caractère - chaîne de caractères fixe ou variable - entier court, standard, long, signé ou non – nombre avec décimales fixes – flottant – complexe – booléen - texte formaté – image - etc.)	Code
VALEUR.Liste[ ]	Liste des valeurs possibles pour un attribut	Tableau d'agrégats
VALEUR.Liste[ ].Code Format	Usage en contrôle des valeurs associées (valeur(s) autorisée(s) - valeur(s) interdite(s) - valeur minimale et/ou maximale, masque, routine…)	Code
VALEUR.Liste[ ].Contrôle()	Routine permettant si nécessaire des contrôles complexes	Routine
VALEUR.Liste[ ].Valeurs	Liste des valeurs associées au type de contrôle.	Selon TYPE
VALEUR.Nom	Nom du contrôle	Texte

Conclusion de la 2ème partie

Nous avons donc vu qu'il était possible de définir, sous la forme – et avec les techniques – d'un MCD Merise amélioré – un métamodèle sémantique universel, qu'il suffit de renseigner pour pouvoir générer une modélisation de n'importe quelle application, qu'elle soit de gestion, objet, temps réel, système expert...

Bien sûr, ce qui a été élaboré tout au long du présent article est loin d'être absolument complet, et prête à de nombreuses critiques. Cet état de choses est délibéré, pour plusieurs raisons :

- Une modélisation complète nécessite une étude beaucoup plus approfondie – une LETTRE entière n'y suffirait pas.
- J'ai conçu, pour ce genre de modélisation, un certain nombre d'éléments dont je tiens à la paternité et que j'estime donc prématuré de présenter au public. Cette restriction sautera dès lors qu'existera un produit opérationnel mettant en œuvre ce concept. Que les frustrés prennent leur mal en patience ou viennent participer au projet.

Dans la suite – et fin – de ce dossier, à paraître dans LA LETTRE n°28, nous verrons justement comment mettre en œuvre ce métamodèle[24] pour réaliser un véritable Système d'Information intégré, permettant tout à la fois de concevoir, modéliser, prototyper (en RAD, s'il vous plaît !) et mettre en service n'importe quelle application informatisable.▲

(à suivre)

Jean-Luc Blary

24 …À quelques astuces près – pour les mêmes raisons que ci-dessus, ce qui frustrera encore une fois plusieurs d'entre vous.

Au sommaire de la 3ème partie

- ✓ 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

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

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

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

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

Le Réseau Sémantique Universel (1ère partie)
BLARY J-L.
La Lettre de l'ADELI n°26, janvier 1997

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

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

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

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

Le langage C++
STROUSTRUP B.
Addison-Wesley, Paris 1992

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*