Generic System 2

Analyse Conceptuelle

Nicolas Feybesse

2009 - 2010

Table des matières

1 Analyse conceptuelle

1.1 Information et structure

Dans un système d'information orienté objet, on distingue le type de l'information de l'information en elle-même. Le type d'information renseigne sur la structure de l'information sous-jacente. Prenons un exemple très simple : une voiture en particulier, la bmw de votre voisin, ne représente pas la même chose que le concept de « voiture » en général qui agrège en quelque sorte l'ensemble des voitures qui existent.

A partir de ces observations, il semble possible de distinguer le « système », qui désigne une représentation de la « réalité », du « meta-système » qui désigne la structure du système pre-cité.

Une des grandes facultés de la pensée humaine est de ne faire qu'une faible distinction dans la désignation des objets du système de ceux du meta-système. Le même mot « voiture » peut ainsi désigner une instance particulière de la classe « voiture », ou la classe elle-même selon qu'il est employé dans tel ou tel contexte. Une des facultés cognitives très intéressante est cette capacité qu'a la pensée, de savoir de façon implicite si «voiture » désigne une instance de voiture dans tel cas ou la classe des instances de voiture dans tel autre.
Nous pourrions dire simplement que la pensée humaine sait bien souvent à quel « **niveau de transcendance**» ou « **niveau d'instanciation** »est désigné un objet a tout instant en fonction du contexte.

Il découle de cela plusieurs choses :

- L'information semble pouvoir être organisée selon des « **niveaux de transcendance**» et ceux-ci ne se limitent pas forcément au nombre de 2. (Rien n'empêche de considérer la structure de la structure de l'information).
- Les niveaux de transcendance, s'ils semblent pouvoir être distingués les uns des autres, semblent toutefois obéir à des règles de gestion assez similaires.
- Il existe des interactions entre les niveaux de transcendance, puisque par exemple la classe « voiture » regroupe l'ensemble des « instances de voiture » et que des objets de différents niveaux de transcendance peuvent donc intervenir dans une même proposition.

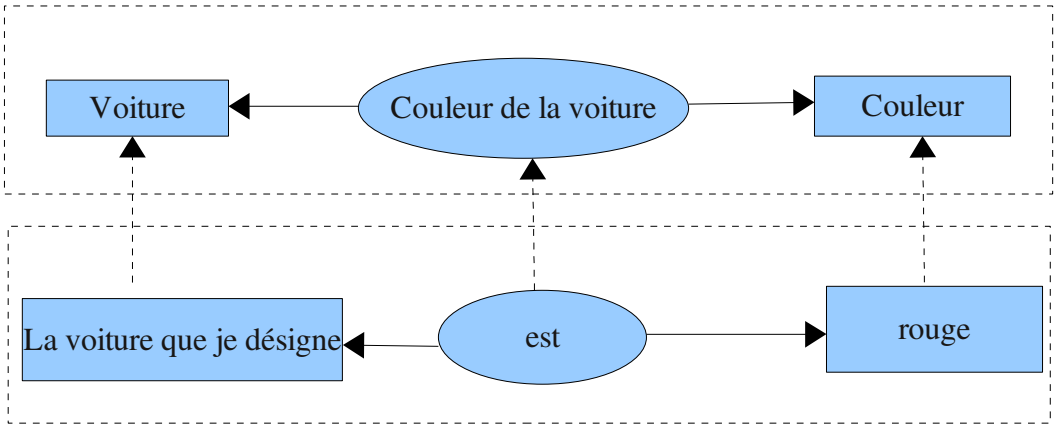
Generic System propose de représenter l'information selon un graphe dont les règles de construction tentent de reproduire le plus canoniquement possible les principes évoqués précédemment.

1.2 Interactions simples et repésentation

A chaque fois que l'on exprime sa pensée, il semble que l'on fasse interagir des objets entre eux. Si je dis que ***cette voiture est rouge***, il semble que je désigne deux objets : « telle voiture » et la couleur « rouge ». En disant que ***cette voiture est rouge***,je les relie entre eux à l'aide du mot « est » qui est le représentant ici de l'interaction que je souhaite exprimer entre ces deux objets.

« Cette voiture » est clairement une instance de la classe « voiture ».
« Rouge » est une instance de la classe « couleur ».
Mais ce n'est pas tout : « est » peut être considéré ici comme instance implicite d'un relation qui relierait la classe voiture à la classe couleur.

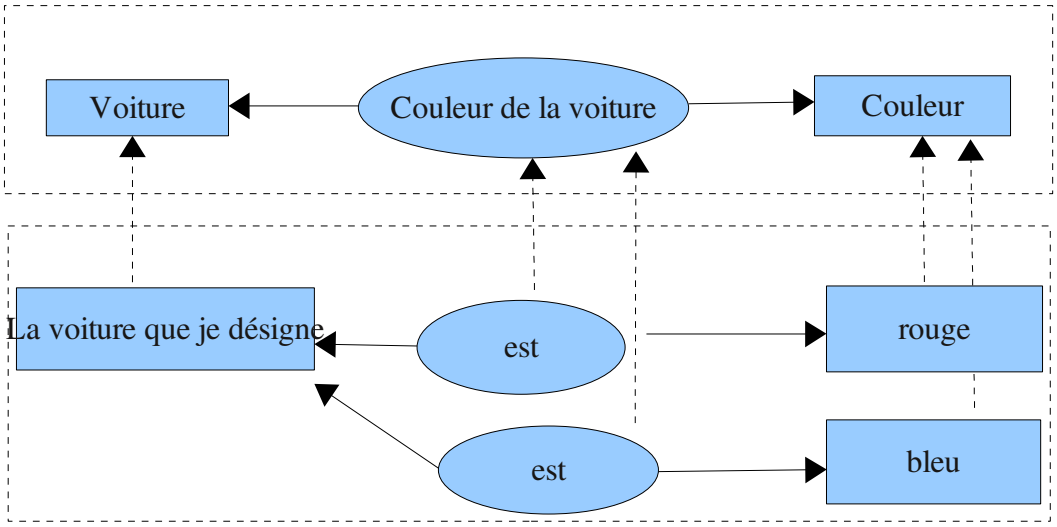
Cela pourrait être simplement représenté pas le schéma suivant :



Dans ce schéma, les objets « voiture », « couleur de la voiture » et « couleur » transcendent les objets « la voiture que je désigne», « est » et « rouge ». Les objets « Voiture », « couleur de la voiture » et « couleur » définissent clairement dans cet exemple la structure des objets « la voiture que je désigne », « est » et « rouge » pour la raison simple que chacun d'entre ces derniers en sont les instances.

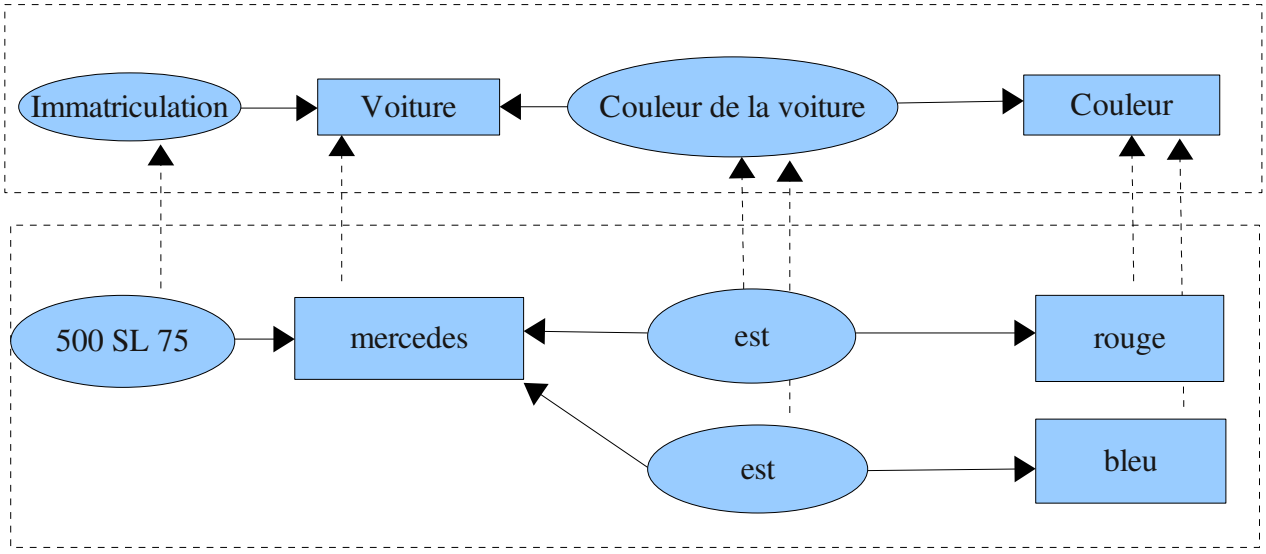
Les flèches en pointillés représentent ici des relations « d'instanciation » d'un niveau de transcendance à un autre.

Si je dis : ***cette voiture est rouge et bleu***, on pourrait représenter cette proposition de la façon suivante :

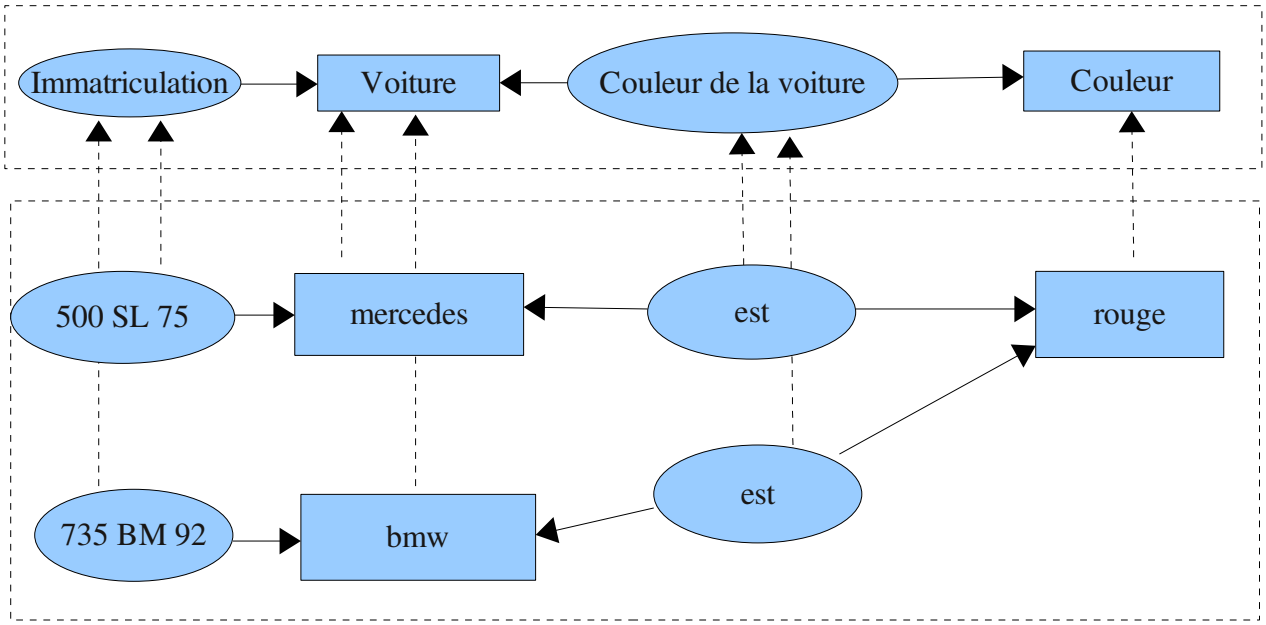


On constate ici que les interactions de « un vers plusieurs » sont simplement représentées.

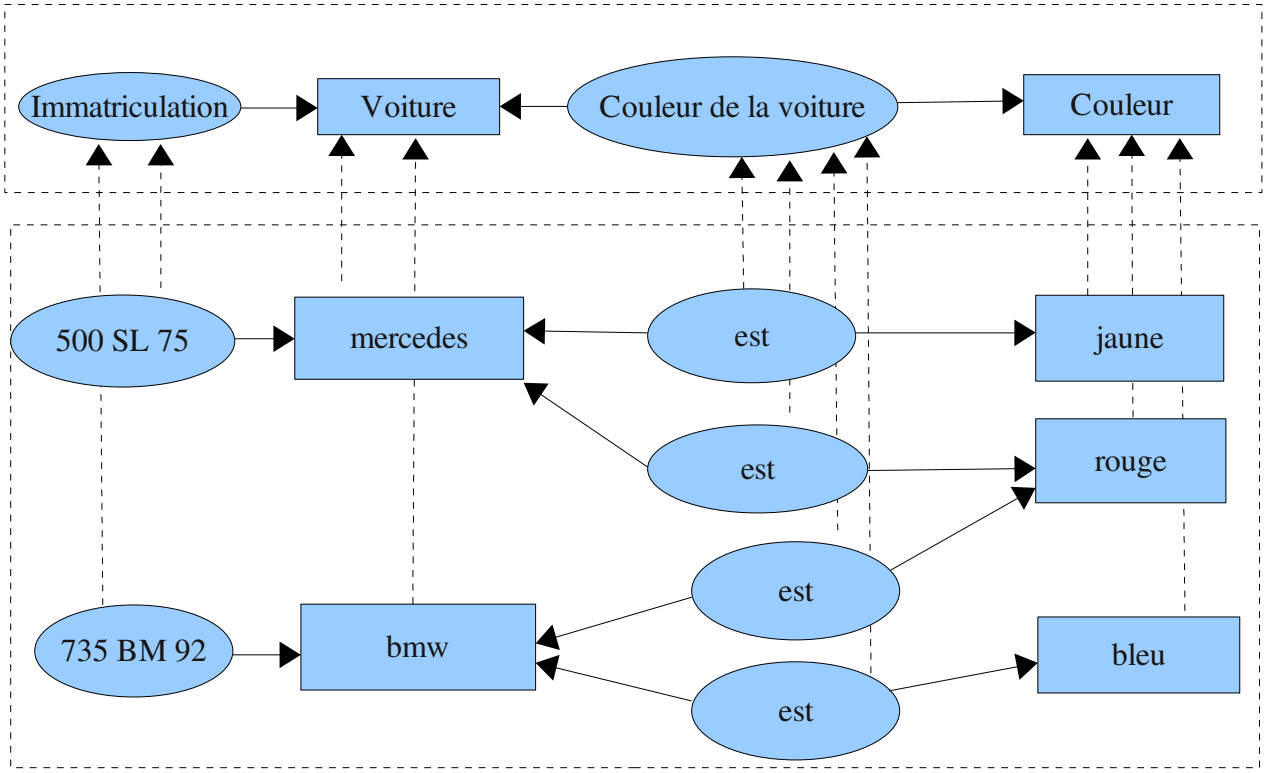
Si je dis :***la mercedes 500 SL 75 est rouge et bleue***, on pourrait représenter cette proposition comme ceci :



De même, si ***la bmw immatriculée 735 BM 92 est rouge et la mercedes 500 SL 75 est également rouge***, le schéma correspondant est celui-ci :



Enfin, si ***la bmw immatriculée 735 BM 92 est rouge et bleue et la mercedes 500 SL 75 est rouge et jaune***, on peut représenter les choses comme ceci :



On constate donc que les interactions « plusieurs vers un » sont symétriquement aussi simplement représentables que les interactions de « un vers plusieurs ». La représentation des relations « plusieurs vers plusieurs » ne pose elle non plus, pas de problème particulier.

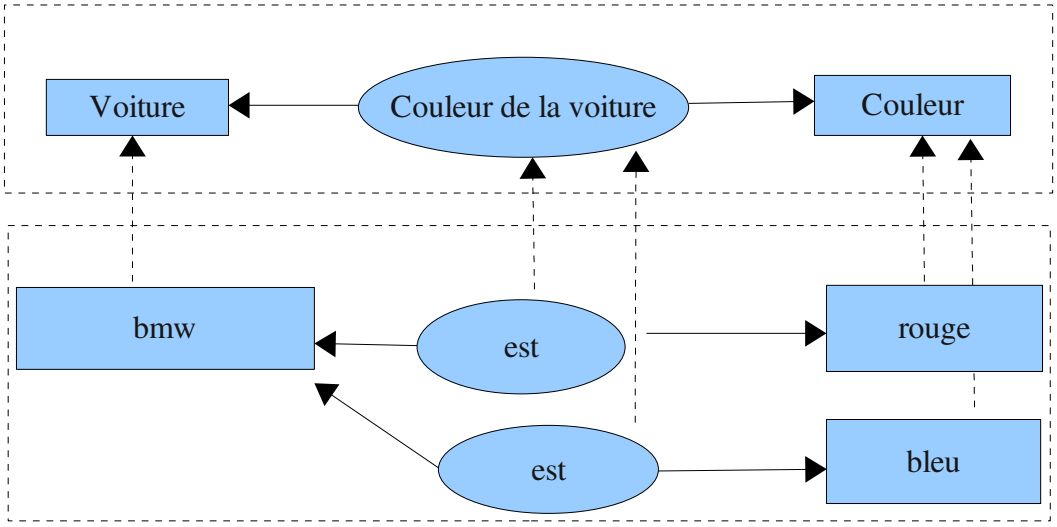
1.3 Ancêtres et dépendances

Après ces quelques exemples, il nous est possible de préciser la signification des flèches dans ces schémas : Lorsqu'une flèche va d'un objet D vers un objet A, cela signifie que l'objet D s'appuie sur l'objet A. L'objet D est construit sur la base de l'objet A, et ainsi l'objet D ne peut exister que si l'objet A existe déjà. La construction de D est en quelque sorte conditionnée par la construction préalable de A.

Par convention on dira que :

- l'objet A est **l'ancêtre** de l'objet D
- l'objet D est **une dépendance** de l'objet A.

Par exemple, dans le schéma ci-dessous, l'objet « Couleur de la voiture » a pour ancêtres les objets « Voiture » et « Couleur ». De même, l'objet « Couleur » a pour dépendances les objets <Couleurs de la voiture » , « rouge » et « bleu ».

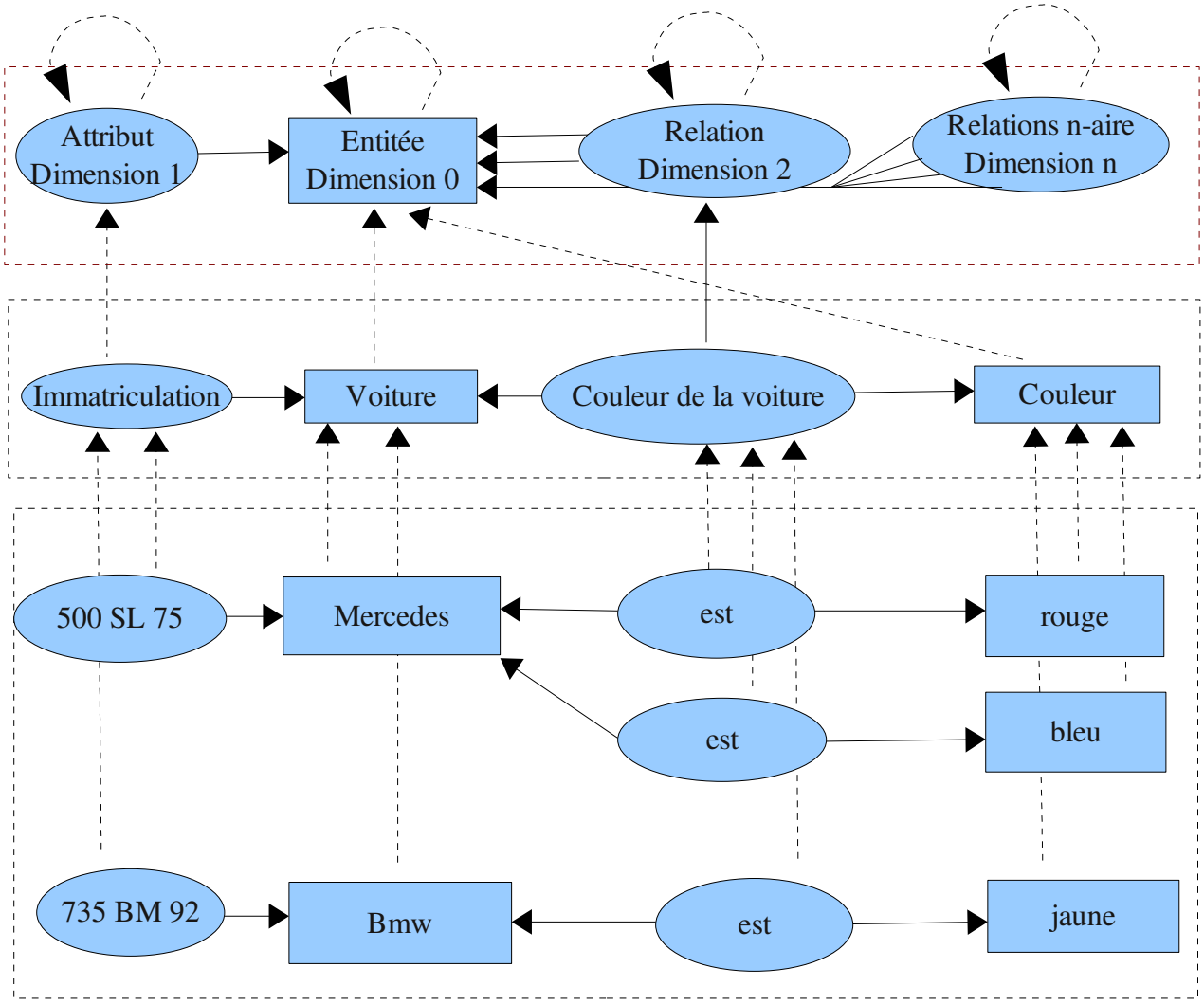


Ainsi, on constatera que les ancêtres sont les objets cibles des flèches sortantes de l'objet considéré et que les dépendances sont les bases des flèches qui entrent dans l'objet considéré.
L'existence d'un objet est conditionné par l'existence de ses ancêtres : c'est une des règles fondamentales qui régissent le système de représentation de Generic System.

1.4 Objets fondamentaux et rang

Nous avons vu que les objets représentés dans les schémas précédents, peuvent être regroupés par des niveaux de transcendance qui sont représentés par les rectangles en pointillés.

Voici ce que donne la représentation du méta-système du méta-système (en rouge) qu'on appellera par convention l'**hyper-système**.



Les objets de l'hyper-système sont appelés par convention les **objets fondamentaux**.
Ils y en a un nombre infini : Les types, les attributs, les relations, les relations ternaires, les relation quaternaires etc …

On peut constater que

- Les instances de l'objet **Type** n'ont qu'**un seul ancêtre** : leur « méta ».
- Les instances de l'objet **Attribut** ont **deux ancêtres** : leur méta et un sous-jacent.
- Les instances de l'objet **Relation** ont **trois ancêtres** : leur méta et 2 sous-jacents
- ....

On nommera par convention ce **nombre d'ancêtres (horsmis le meta)** la **dimension** d'un objet.
Par convention le meta n'est pas compté dans la dimension car il est toujours renseigné.

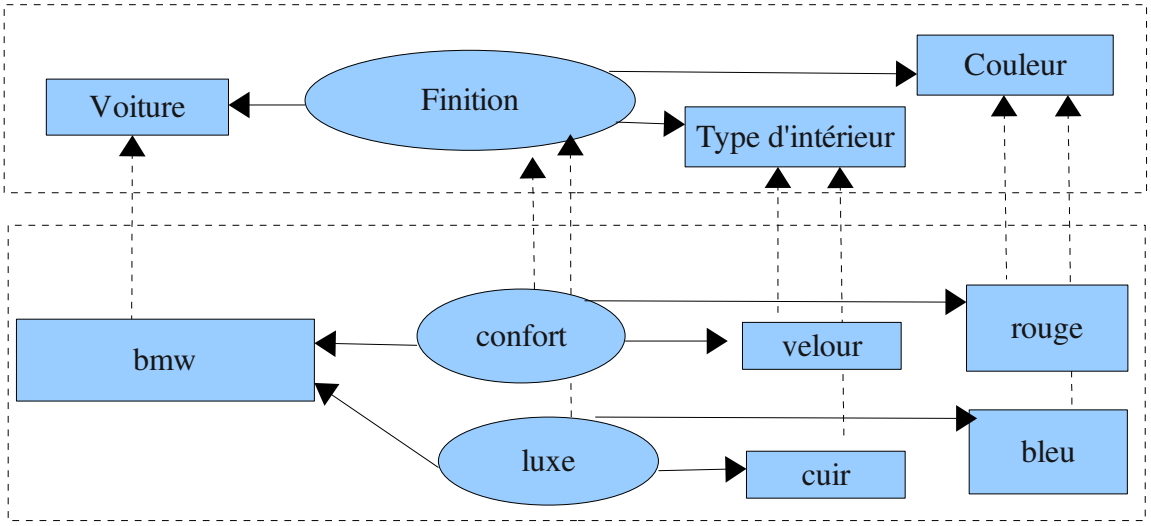
On nommera **niveau d'instanciation (ou espace transcendantal) de rang n** l'ensemble des objets de même rang du graphe issus de**s** instanciations successives pour chaque objet fondamental.

L'**hyper-système** est le niveau d'instanciation 0 et est constitué d'objets de **rang 0**. Il sert au fonctionnement de Generic System.
Le **meta-système** est le premier niveau d'instanciation et est constitué d'objets de **rang 1** : il représente concrètement la **structure de l'information (le contenant).**
Enfin le deuxième niveau d'instanciation est constitué d'objets de **rang 2**, il correspond à la représentation de **la réalité (le contenu)**
Le niveau 3, philosophiquement plus complexe à appréhender, n'est pas utilisé pour l'instant, et pourrait s'apparenter à de l'**information sensorielle ou émotionnelle,** le résultat perceptif d'un ensemble de « capteurs ».

1.5 Relations n-aires

Les relations n-aires sont nativement gérées par le modèle de Generic System.

Voici un exemple de relation ternaire.

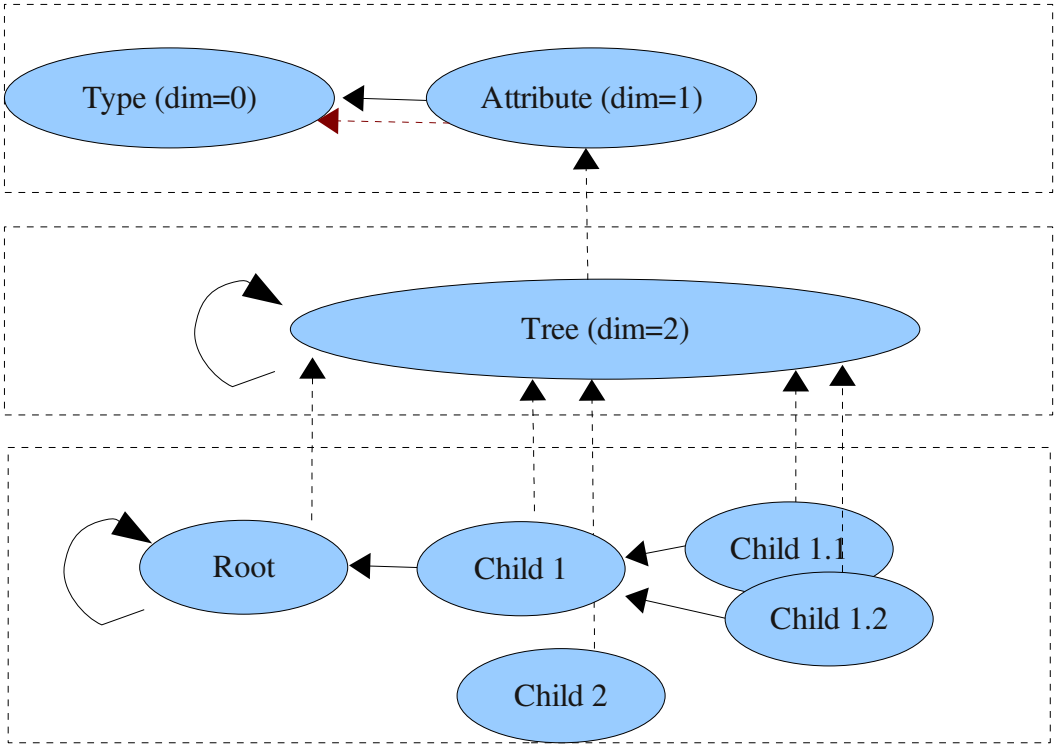


1.6 Structures arborescentes

Les structures arborescentes sont nativement gérées par le modèle de Generic System.

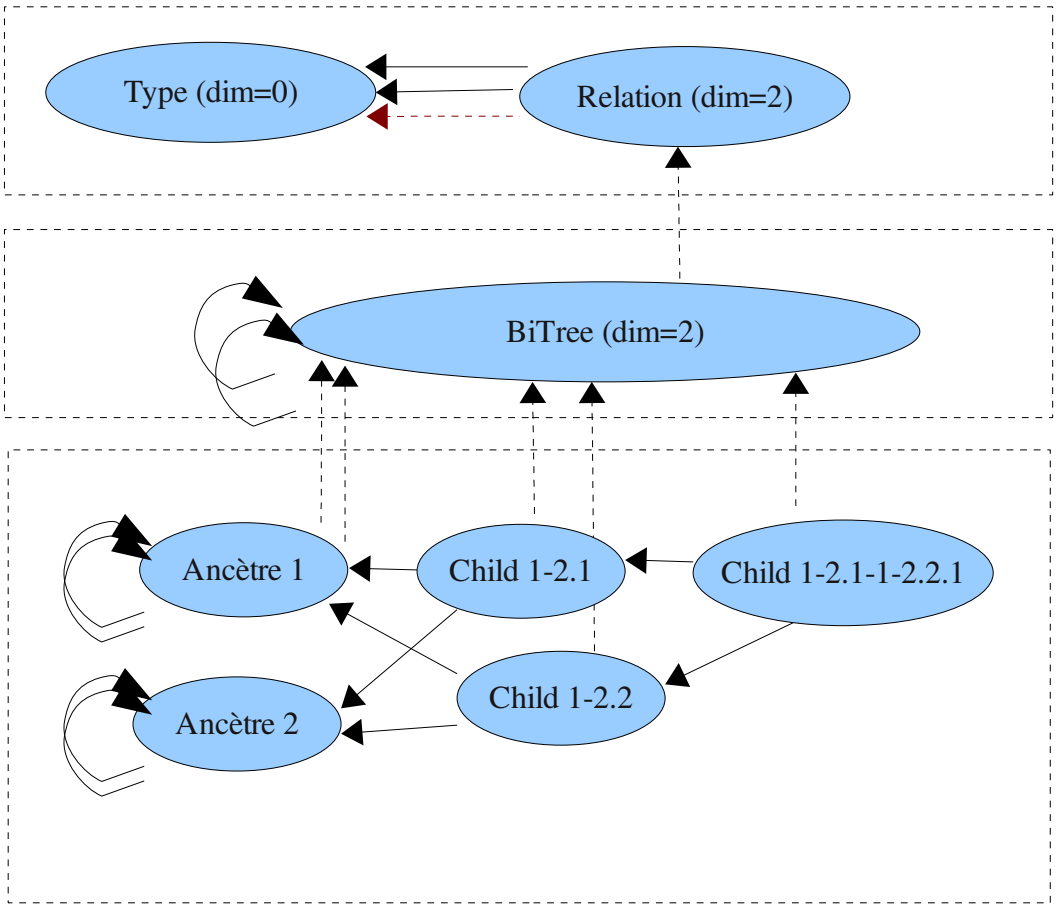
Un arbre est simplement vu comme un attribut de lui-même et dont les nœuds enfants (valeurs d'attribut) reposent sur un autre nœud parent (valeurs d'attribut)

La racine de l'arbre est un noeud particulier dont le parent est lui-même.



1.7 Arbre n-aires

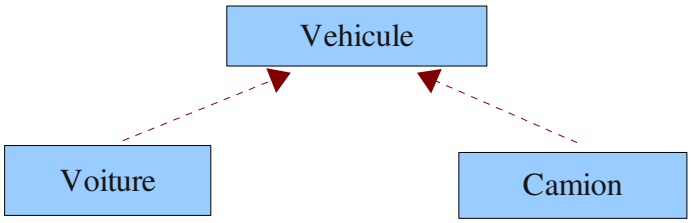
Un arbre n-aire est un arbre dont chacun des nœuds a « n » parents. Ces arbres sont vu comme des relation n-aires de dimention n+1.
Voici par exemple un arbre généalogique (binaire)



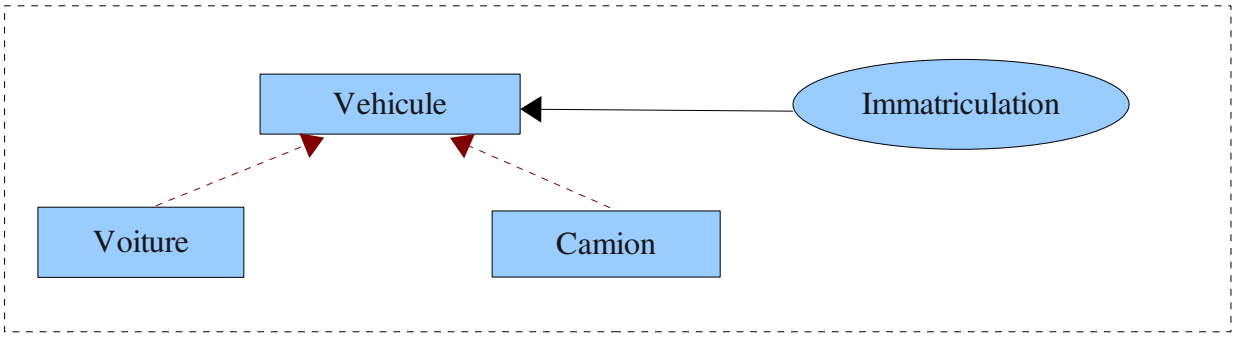
1.8 Héritage

Tout système de représentation objet fait intervenir la notion d'héritage. Generic System ne déroge pas à cette règle et en étend même le principe si on le compare à ce qui se fait usuellement.

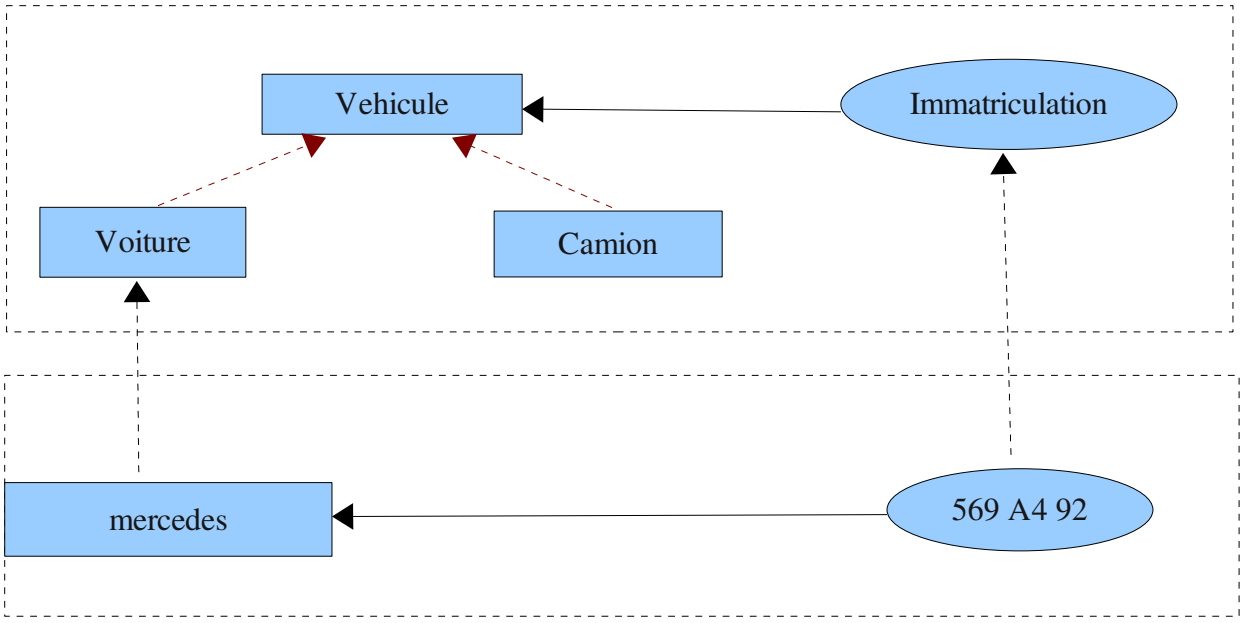
Pour illustrer simplement l'héritage, prenons un exemple simple de trois classes :



Imaginons désormais que nous paramétrions un attribut « immatriculation » sur « véhicule »
Cela donnerait cette représentation :



Si on instancie une voiture avec une plaque, cela donne cela :



Cette situation est tout à fait réglementaire et autorisée dans Generic System.

1.9 Concept de standard

Dans Generic System, tout objet du « meta-système » possède une instanciation un peu particulière dans le système nommée « standard », et dont tout autre instanciation du même meta hérite (directement ou indirectement).

Cette instance porte en quelque sorte, toute les informations habituelles dont tout autre instance du même type (et des types hérités) doivent hériter.

1.10 L'héritage naturel des objets de dimension n envers ceux de dimension inférieure et multiplicité.

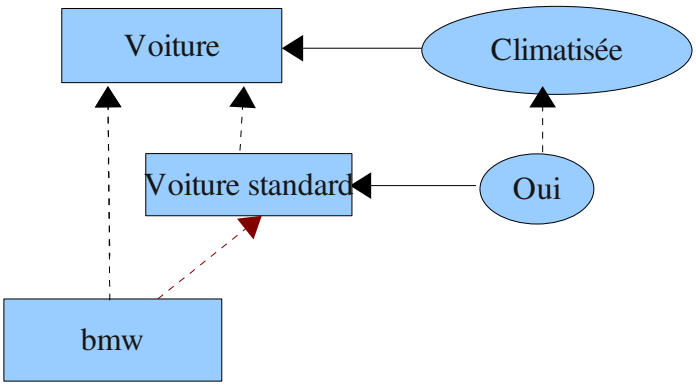
Dans Generic System et contrairement à la norme UML, les attributs, les relations et les relations n-aires sont considérés eux mêmes comme des entités. Ainsi, l'attribut d'une classe peut avoir plusieurs valeurs puisqu'en fait dans Generic System ces valeurs d'attribut sont représentés (comme des entités) par des instanciations différentes de l'attribut et qui ont potentiellement le même sous-jacent l'instance de la classe à laquelle l'attribut se rapporte.
De même, le principe s'étend au relations binaires et n-aires puisque chaque instanciation de ces relations porte une valeur d'instanciation.
Dans Generic System, l'attribut n'est en fait que la vision unaire d'un relation n-aire, une relation est la vision binaire d'une relation n-aire. Dans la me logique, un type est la vision 0-aire d'une relation n-aire.

1.11 L'information flottante et redéfinition partielle

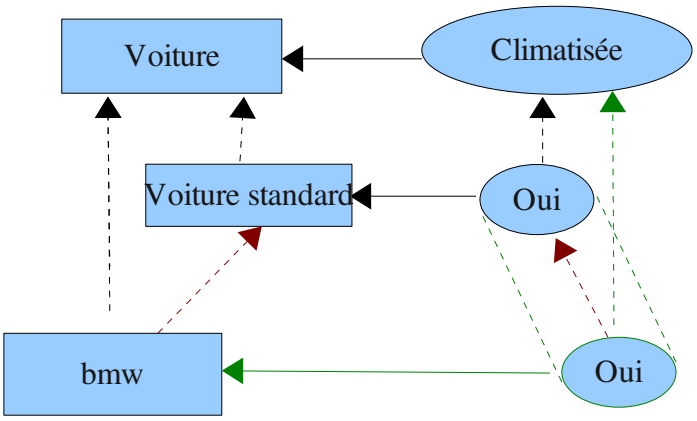
Représenter une information se rapportant à un ensemble d'instances d'une classe est un besoin récurrent lors de la conception d'un système de représentation d'information.
Les systèmes de gestion de base de données traditionnels, ou même plus généralement les outils de persistance orientés objets du marché n'apportent aucune réponse à ce besoin pourtant évident.

Generic System introduit à cet effet, le concept d'information « flottante » pour pallier cette carence. Prenons un exemple :

Toute voiture est climatisée

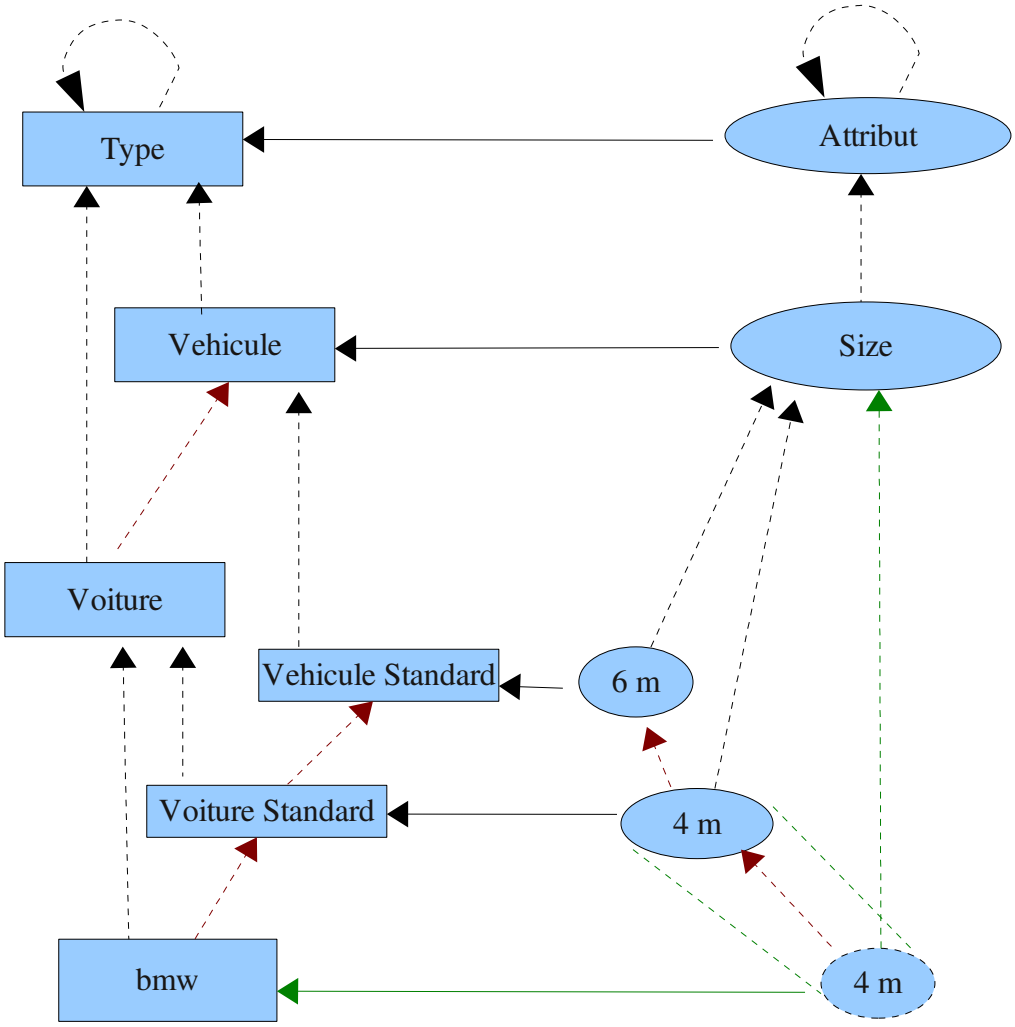


Sur le schéma ci-dessus, on constate que la « bmw » est climatisée, parce que la voiture standard l'est. L'information est en quelque sorte « projetée » de la « voiture standard » sur la « bmw » du fait qu'elle hérite de la voiture standard.



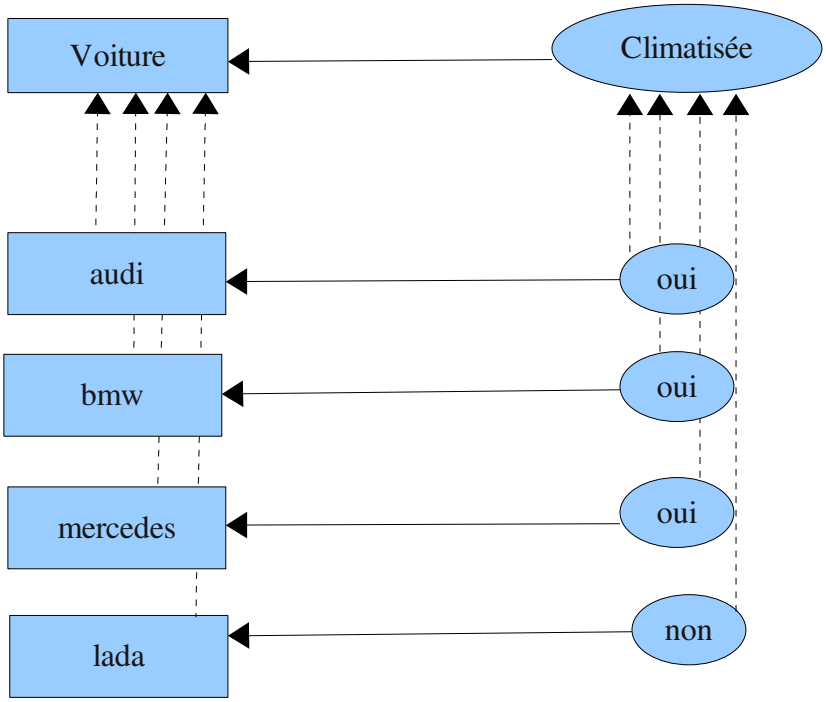
Cette information n'a pas besoin d'être représentée car Generic System déduit automatiquement cette information.

Autre exemple de redéfinition partielle :

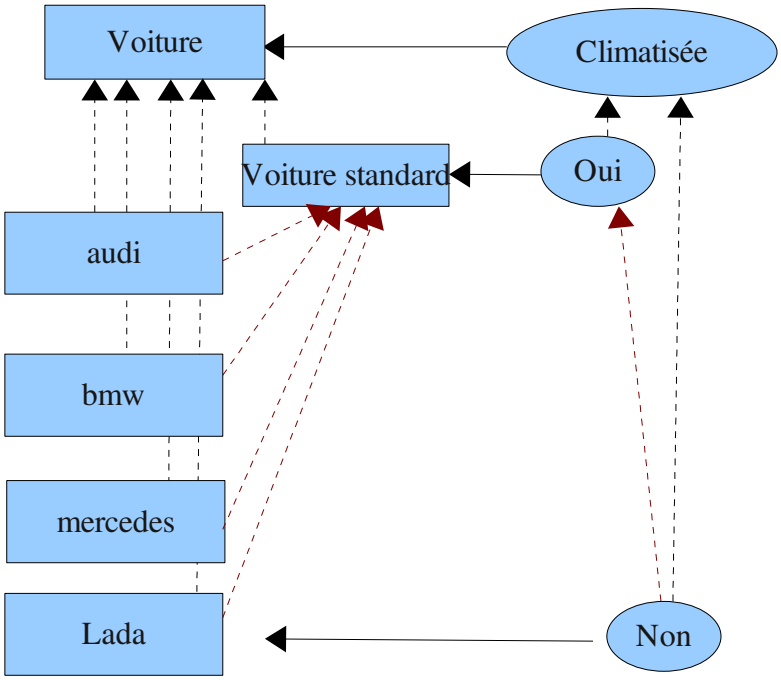


Autre cas de figure :

Toute voiture est climatisée sauf la « lada »

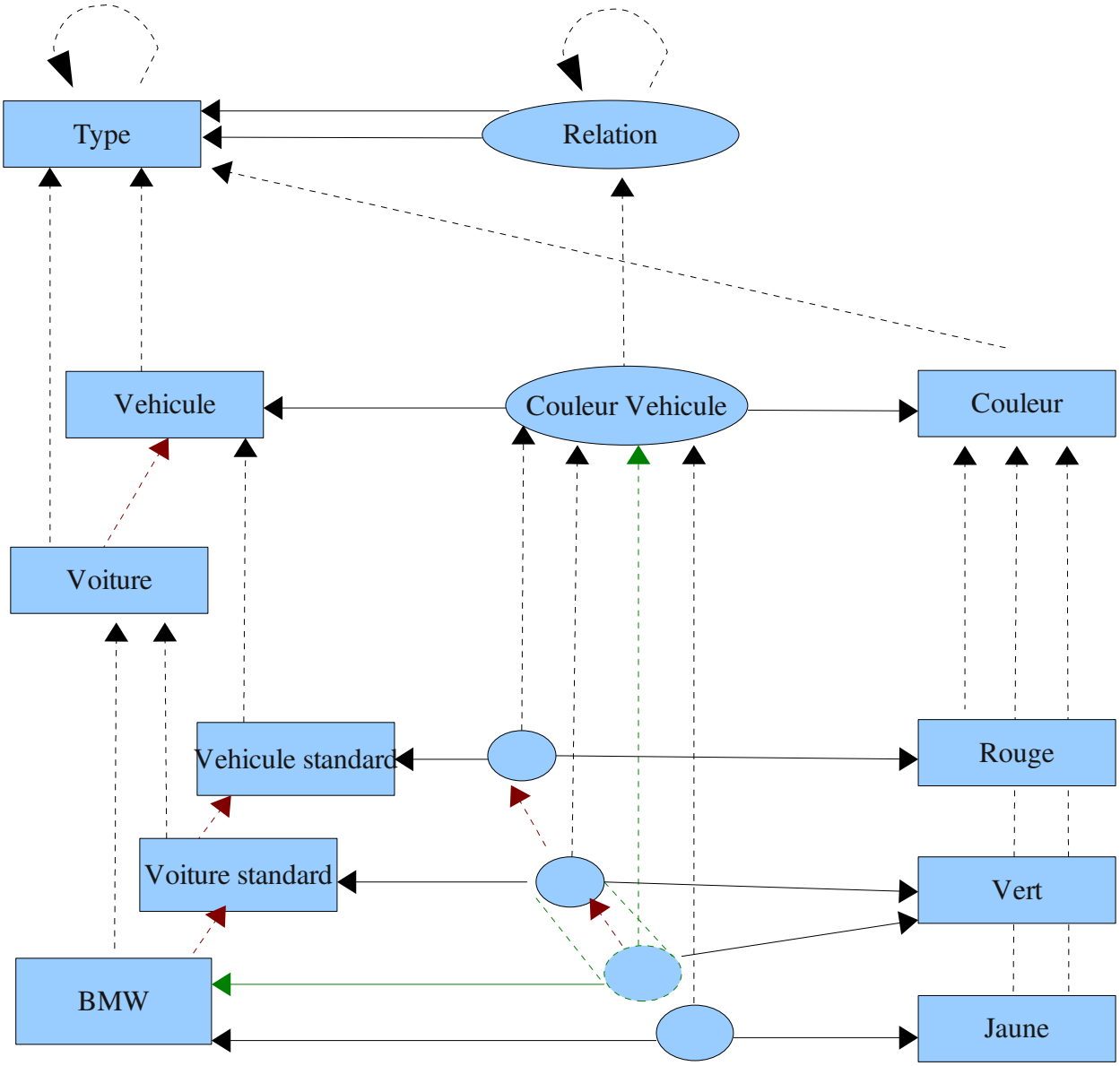


Sur ce modèle, on comprendra aisément que si la plupart des instances de voitures sont climatisées, alors, c'est le fait de ne pas l'être qui doit être représenté.
C'est ce que propose Generic System avec le mécanisme de redéfinition partielle (override) :



Ici on peut s'apercevoir que l'information est stockée par différence avec l'habituel. On comprendra aisément que dans de nombreux situations une économie substantielle d'espace de stockage peut être réalisé en ne représentant que ce qui est exceptionnel.

Cela est vrai pour les attributs (dimension 2) mais aussi pour les relations de dimension n :



La bmw est dans cet exemple à la fois verte et jaune, verte de manière standard, et jaune aussi de manière individuelle.

1.12 L'axe des ancêtres et des dépendances

Tout objet du graphe de Generic System possède autant d'ancêtres que sa dimension.
Ces ancêtres sont indexés du premier au n-ième. Le concept d'axe représente cet index.
L'ancêtre d'un objet D, l'est toujours pour un index donné. Dans ce cas, l'objet D devient dépendance de l'ancêtre considéré toujours pour cet axe donné.

1.13 Contraintes fondamentales

Les concepts fondamentaux et les définitions ayant étés présentées, il convient désormais d'énoncer les contraintes qui sont toujours respectées afin de garantir la cohérence de l'information contenue dans le système :

Contrainte fondamentale n°1
A instance de B = > dim(A) = dim(B)

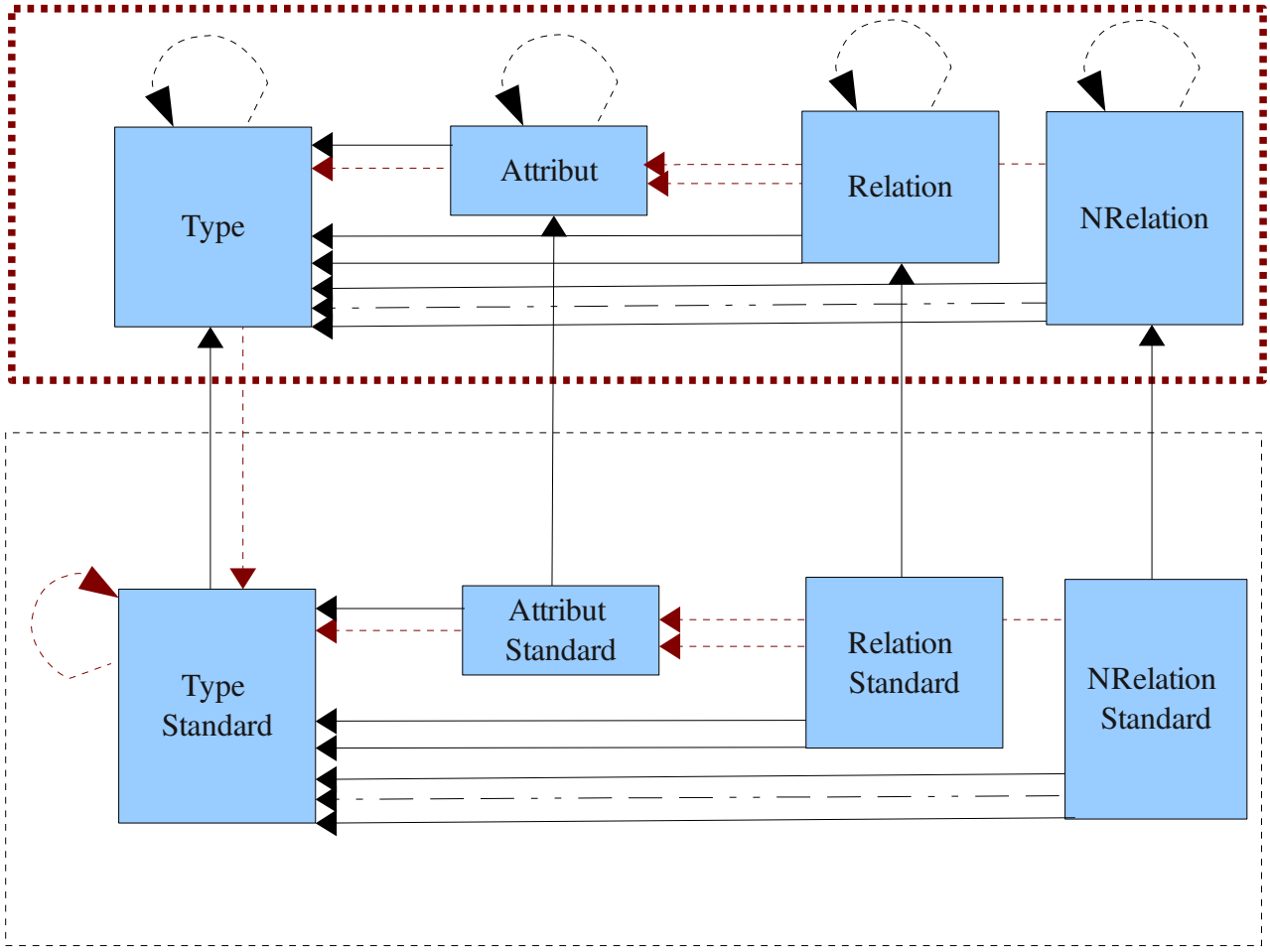
Contrainte fondamentale n°2
A hérite de B = > dim(A) >= dim(B)

Contrainte fondamentale n°3 (meta rule)
quelque soit l'axe < dim(A),
A instance de B => Ancêtre(A,axe) est instance de Ancêtre (B,axe)

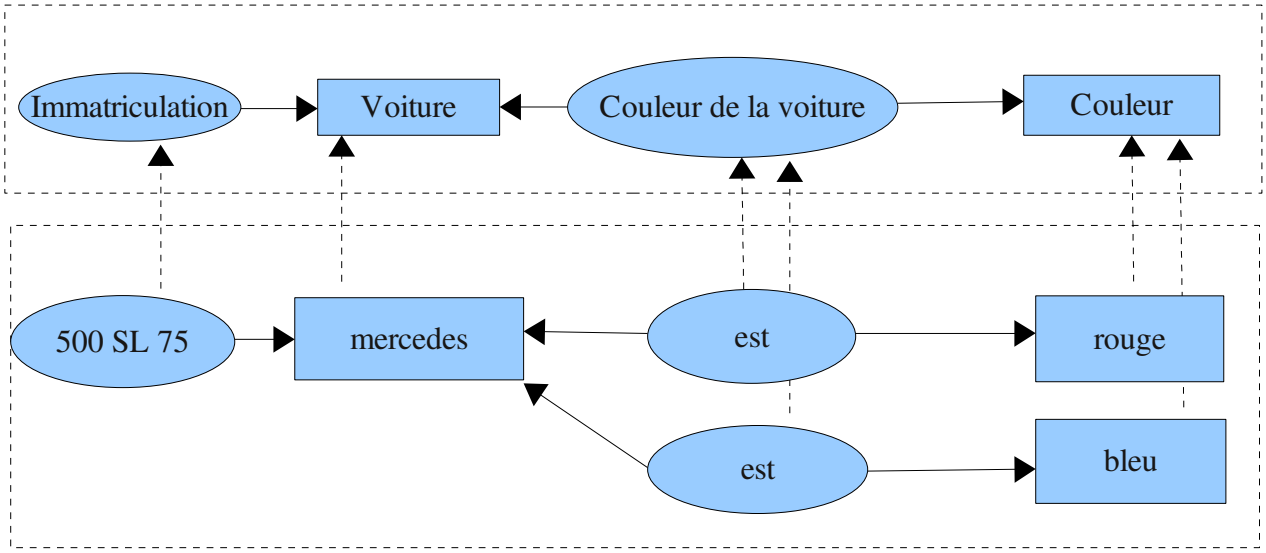
Contrainte fondamentale n°4 (super rule)
quelque soit l'axe < dim(A),
A hérite de B => soit Ancêtre(A,axe) hérite de Ancêtre (B,axe), soit Ancêtre (B,axe) n'existe pas

1.14 Hyper System

Suite à ne de nombreuses expérimentations, nous sommes parvenus à élaborer un modèle cohérent pour l'hyper-système :



1.15 Exemple simple d'utilisation de l'API



Ce schéma peut être très simplement créé par le code java suivant :

```
@Test
public class DocTest extends AbstractSessionTest {

   public void Test() {
      Type voiture = metaType.newInstance("voiture");
      Type couleur = metaType.newInstance("couleur");
      Relation voiture_couleur = metaRelation.newInstance("voiture-couleur", voiture, couleur);
      Attribute immatriculation = metaAttribute.newInstance("immatriculation", voiture);
      Entity mercedes = voiture.newInstance("mercedes");
      Entity rouge = couleur.newInstance("rouge");
      Entity blue = couleur.newInstance("blue");
      Link mercedes__rouge = voiture_couleur.newInstance("mercedes <---> rouge", mercedes, rouge);
      Link mercedes__blue = voiture_couleur.newInstance("mercedes <---> blue", mercedes, blue);
      Value voiture_500_SL_75 = immatriculation.newInstance("500 SL 75", mercedes);
   }
}
```

On peut constater que la structure de l'information est manipulée de la même manière que l'information elle-même.

1.16 Conclusion

Les principes cognitifs ont toujours largement influencé la conception de Generic System. Ainsi, on s'apercevra que l'ensemble des « interactions imaginables » peuvent être représentées dans Generic System.
Les possibilités qu'offrent les diagrammes de classe de la norme UML trouvent exhaustivement une représentation naturelle dans Generic System. Cela se vérifie en particulier en ce qui concerne les relations d'héritage et les relations n-aires mais aussi en ce qui concerne les structures arborescentes.

Avec les concepts de standard et d'héritage, il devient possible de représenter l'information de manière très optimisée, par différence avec l'habituel. Les perspectives de concevoir des algorithmes « intelligents » qui factorisent l'information essentielle sont très importantes. Celle de concevoir des algorithmes qui structurent automatiquement l'information « par analogies» le sont tout autant.
Ces deux axes de recherches, très liés à l'intelligence artificielle, vont naturellement guider nos recherches dans les années à venir.