Vers une relation d'héritage générique

Christophe Jalady

Encadrants: Philippe Lahire, Pierre Crescenzo, Michel Gautero

Rapporteur: Didier Parigot

Université de Nice-Sophia Antipolis

Laboratoire I3S - Projet OCL

Plan de la soutenance

1. Rappel des motivations
2. Etat de l'art
3. Une relation d'héritage générique : approche intuitive
4. Rappels des problèmes
5. Spécialisation de l'usage de l'héritage
   (a) Taxinomie de l'héritage
   (b) Le concept d'annotation
6. Synthèse et conclusion

Rappel : Motivations

- Trois difficultés :
  - Adaptation d'un composant logiciel [HÖL93, TL94]
  - Modification du comportement d'un composant
  - Ajout de fonctionnalités à un composant [HO92]

- Insuffisance de la relation d'héritage :
  - Extension uniquement par ajout de descendants
  - Incohérence avec la notion d'indépendance de représentation

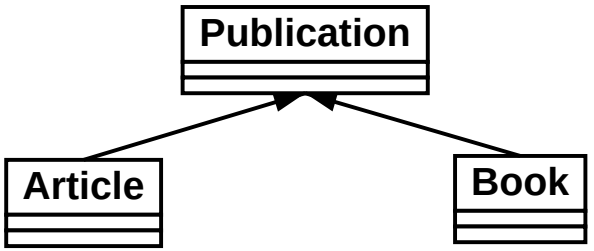
$\Rightarrow$ Amélioration de l'expressivité de l'héritage

Le langage BETA

Deux principaux concepts :

- Redéfinition d'une propriété de manière à garantir une *compatibilité structurelle*.
- Concept de classe interne *virtuelle* permettant la généricité .

Exemple :
Ajout d'un attribut *"numéro"* à toute une hiérarchie de type *Publication*, *Article*, *Book*.



Le langage BETA : exemple

```
1  Publication: class Record
2            (# Author: ...; Title: ...; Date: ... #) ;
3
4  Book: class Publication
5      (# ... #) ;
6
7  Article: class Publication
8         (# ... #) ;
9
10
11 PubGen: class
12       (# GenType: virtual class Publication;
13          ArcPub: class GenType;
14                  (# ArcNo: @integer #) ;
15           New: proc
16                (# RN: @integer; R: ^ArcPub
17                   enter RN
18                   do new ArcPub[] -> R[];
19                      RN-> R.ArcNo;
20                   exit R[]
21           #) ;
22 #) ;
23
24 BookGen: class PubGen
25        (# Gentype: extended class Book #) ;
26
27 ArticleGen: class PubGen
28           (# Gentype: extended class Article #) ;
```

Le langage BETA : exemple

```
1  Publication: class Record
2            (# Author: ...; Title: ...; Date: ... #) ;
3
4  Book: class Publication
5      (# ... #) ;
6
7  Article: class Publication
8        (# ... #) ;
9
10
11 PubGen: class
12       (# GenType: virtual class Publication;
13          ArcPub: class GenType;
14                  (# ArcNo: @integer #) ;
15           New: proc
16                (# RN: @integer; R: ^ArcPub
17                   enter RN
18                   do new ArcPub[] -> R[];
19                      RN-> R.ArcNo;
20                   exit R[]
21           #) ;
22 #) ;
23
24 BookGen: class PubGen
25        (# Gentype: extended class Book #) ;
26
27 ArticleGen: class PubGen
28           (# Gentype: extended class Article #) ;
```

Le langage BETA : exemple

```
1   Publication: class Record
2              (# Author: ...; Title: ...; Date: ... #) ;
3
4   Book: class Publication
5       (# ... #) ;
6
7   Article: class Publication
8          (# ... #) ;
9
10
11  PubGen: class
12         (# GenType: virtual class Publication;
13            ArcPub: class GenType;
14                    (# ArcNo: @integer #) ;
15             New: proc
16                  (# RN: @integer; R: ^ArcPub
17                     enter RN
18                     do new ArcPub[] -> R[];
19                        RN-> R.ArcNo;
20                     exit R[]
21           #) ;
22  #) ;
23
24  BookGen: class PubGen
25         (# Gentype: extended class Book #) ;
26
27  ArticleGen: class PubGen
28             (# Gentype: extended class Article #) ;
```

Le langage BETA : exemple

```
1  Publication: class Record
2            (# Author: ...; Title: ...; Date: ... #) ;
3
4  Book: class Publication
5      (# ... #) ;
6
7  Article: class Publication
8         (# ... #) ;
9
10
11 PubGen: class
12       (# GenType: virtual class Publication;
13          ArcPub: class GenType;
14                  (# ArcNo: @integer #) ;
15           New: proc
16                (# RN: @integer; R: ^ArcPub
17                   enter RN
18                   do new ArcPub[] -> R[];
19                      RN-> R.ArcNo;
20                   exit R[]
21          #) ;
22 #) ;
23
24 BookGen: class PubGen
25        (# Gentype: extended class Book #) ;
26
27 ArticleGen: class PubGen
28           (# Gentype: extended class Article #) ;
```

Le langage BETA : exemple

```
1  Publication: class Record
2            (# Author: ...; Title: ...; Date: ... #) ;
3
4  Book: class Publication
5      (# ... #) ;
6
7  Article: class Publication
8        (# ... #) ;
9
10
11 PubGen: class
12       (# GenType: virtual class Publication;
13          ArcPub: class GenType;
14                  (# ArcNo: @integer #) ;
15           New: proc
16                (# RN: @integer; R: ^ArcPub
17                   enter RN
18                   do new ArcPub[] -> R[];
19                      RN-> R.ArcNo;
20                   exit R[]
21           #) ;
22 #) ;
23
24 BookGen: class PubGen
25       (# Gentype: extended class Book #) ;
26
27 ArticleGen: class PubGen
28         (# Gentype: extended class Article #) ;
```

Le langage BETA : exemple

```
1  Publication: class Record
2            (# Author: ...; Title: ...; Date: ... #) ;
3
4  Book: class Publication
5      (# ... #) ;
6
7  Article: class Publication
8         (# ... #) ;
9
10
11 PubGen: class
12       (# GenType: virtual class Publication;
13          ArcPub: class GenType;
14                  (# ArcNo: @integer #) ;
15           New: proc
16                (# RN: @integer; R: ^ArcPub
17                   enter RN
18                   do new ArcPub[] -> R[];
19                      RN-> R.ArcNo;
20                   exit R[]
21           #) ;
22 #) ;
23
24 BookGen: class PubGen
25        (# Gentype: extended class Book #) ;
26
27 ArticleGen: class PubGen
28           (# Gentype: extended class Article #) ;
```

La séparation des préoccupations

Paradigme de programation : séparation de fonctionnalités.

- Permet de se concentrer sur l'algorithme principal et de ne pas avoir de code "pollueur".
- Modification/adaptation non invasive.
- Besoin d'un langage pour spécifier l'intégration (ex : AspectJ).
- Intégré aux paradigmes Objet ou Fonctionnel.

La séparation des préoccupations : exemple

```
1  abstract aspect Trace {
2
3      protected static void traceEntry(String str, Object o) {
4        stream.println("Entering " + str + ": " + o.toString());
5      }
6
7
8      abstract pointcut myClass(Object obj);
9
10     pointcut myConstructor(Object obj): myClass(obj) && execution(ne
11     pointcut myMethod(Object obj): myClass(obj) &&
12         execution(* *(..)) && !execution(String toString());
13
14     before(Object obj): myConstructor(obj) {
15         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
16     }
17     before(Object obj): myMethod(obj) {
18         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
19     }
20 }
21
```

La séparation des préoccupations : exemple


```
1  abstract aspect Trace {
2
3      protected static void traceEntry(String str, Object o) {
4        stream.println("Entering " + str + ": " + o.toString());
5      }
6
7
8      abstract pointcut myClass(Object obj);
9
10     pointcut myConstructor(Object obj): myClass(obj) && execution(ne
11     pointcut myMethod(Object obj): myClass(obj) &&
12         execution(* *(..)) && !execution(String toString());
13
14     before(Object obj): myConstructor(obj) {
15         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
16     }
17     before(Object obj): myMethod(obj) {
18         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
19     }
20 }
21
```

La séparation des préoccupations : exemple

```
1  abstract aspect Trace {
2
3      protected static void traceEntry(String str, Object o) {
4        stream.println("Entering " + str + ": " + o.toString());
5      }
6
7
8      abstract pointcut myClass(Object obj);
9
10     pointcut myConstructor(Object obj): myClass(obj) && execution(ne
11     pointcut myMethod(Object obj): myClass(obj) &&
12         execution(* *(..)) && !execution(String toString());
13
14     before(Object obj): myConstructor(obj) {
15         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
16     }
17     before(Object obj): myMethod(obj) {
18         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
19     }
20 }
21
```

La séparation des préoccupations : exemple


```
1  abstract aspect Trace {
2
3      protected static void traceEntry(String str, Object o) {
4        stream.println("Entering " + str + ": " + o.toString());
5      }
6
7
8      abstract pointcut myClass(Object obj);
9
10     pointcut myConstructor(Object obj): myClass(obj) && execution(ne
11     pointcut myMethod(Object obj): myClass(obj) &&
12         execution(* *(..)) && !execution(String toString());
13
14     before(Object obj): myConstructor(obj) {
15         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
16     }
17     before(Object obj): myMethod(obj) {
18         traceEntry("" + thisJoinPointStaticPart.getSignature(), obj)
19     }
20 }
21
```

Rappel : Problématique

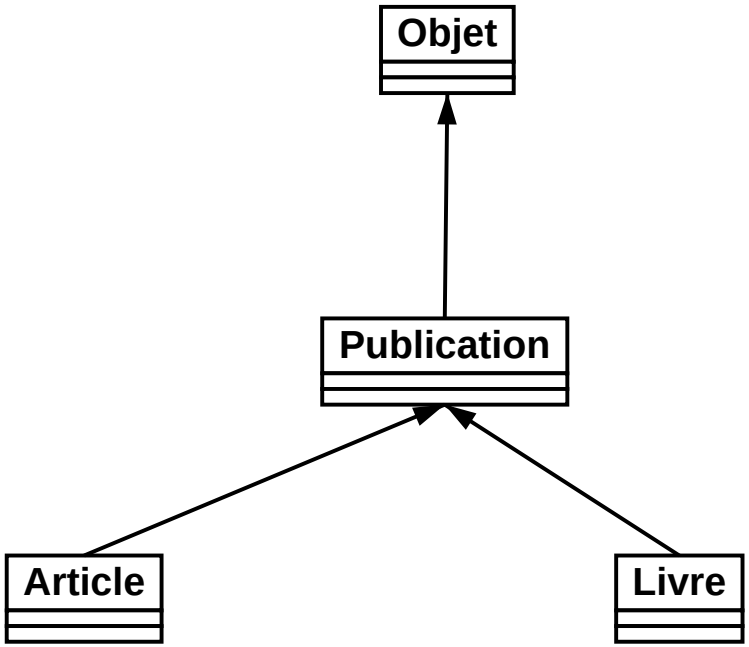
Objectif :
Ajouter la possibilité de modifier la cible de la relation d'héritage

- Donner une possibilité de redéfinir la cible d'héritage de manière externe au composant

- Donner une possibilité de paramétrer la superclasse. Ajout possible d'un paramètre formel désignant la cible de la relation d'héritage

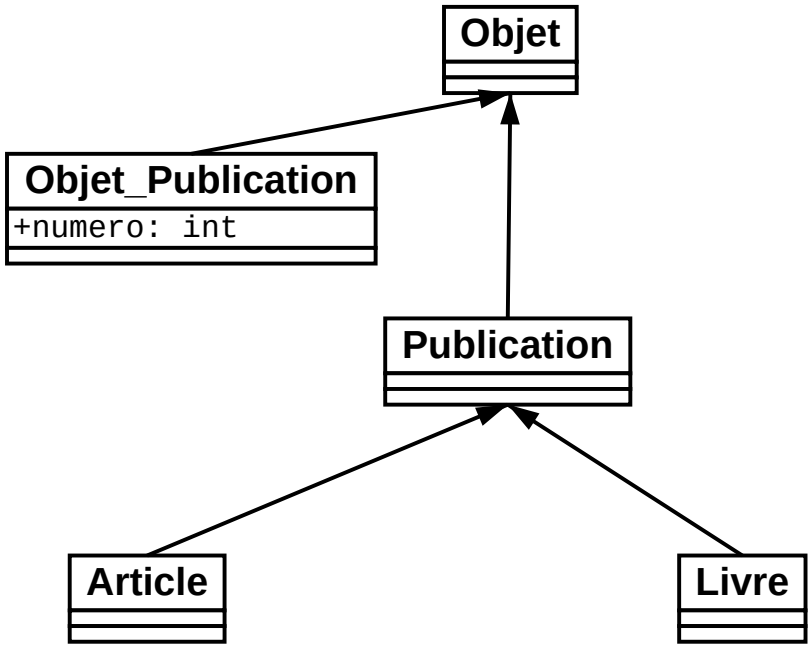
Proposition : Approche intuitive

Exemple d'ajout d'un nouvel attribut à une hiérarchie : cas de la **redéfinition**.



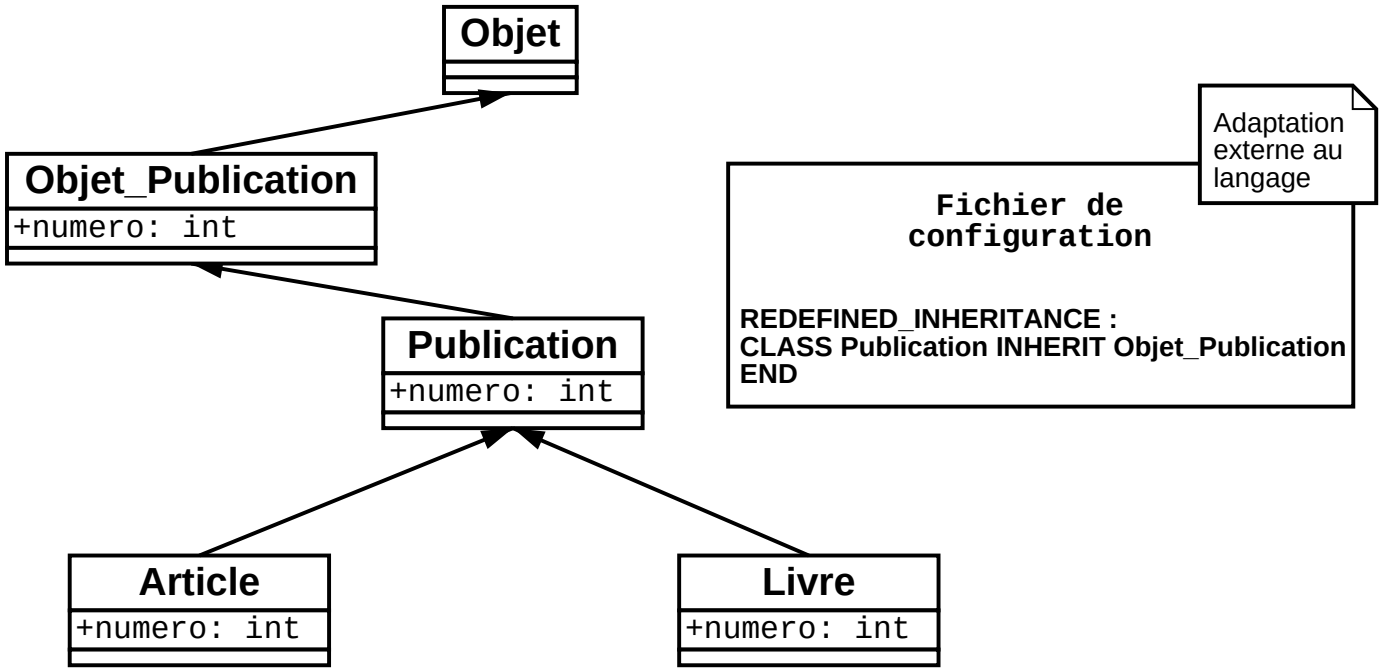
Proposition : Approche intuitive

Exemple d'ajout d'un nouvel attribut à une hiérarchie : cas de la **redéfinition**.



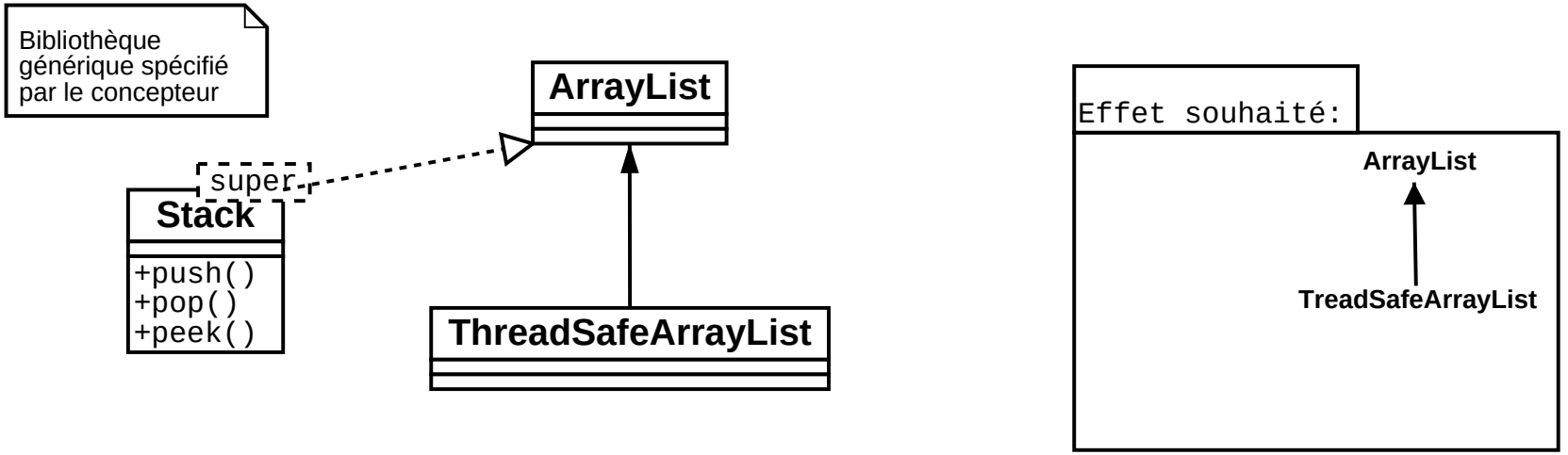
Proposition : Approche intuitive

Exemple d'ajout d'un nouvel attribut à une hiérarchie : cas de la **redéfinition**.



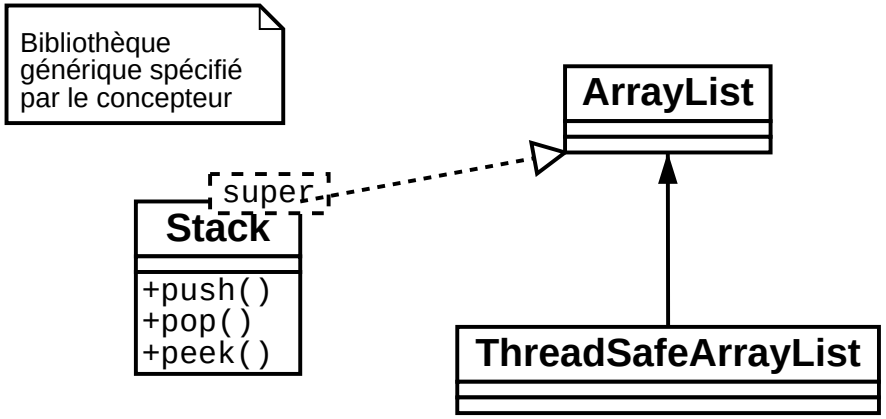
Proposition : Approche intuitive

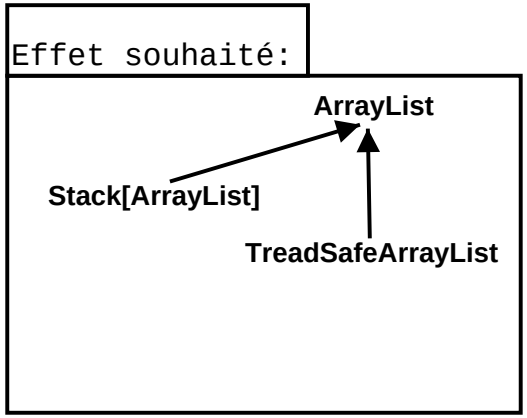
Exemple d'adaptation d'un composant : cas de l'**héritage générique**.



Proposition : Approche intuitive

Exemple d'adaptation d'un composant : cas de l'**héritage générique**.

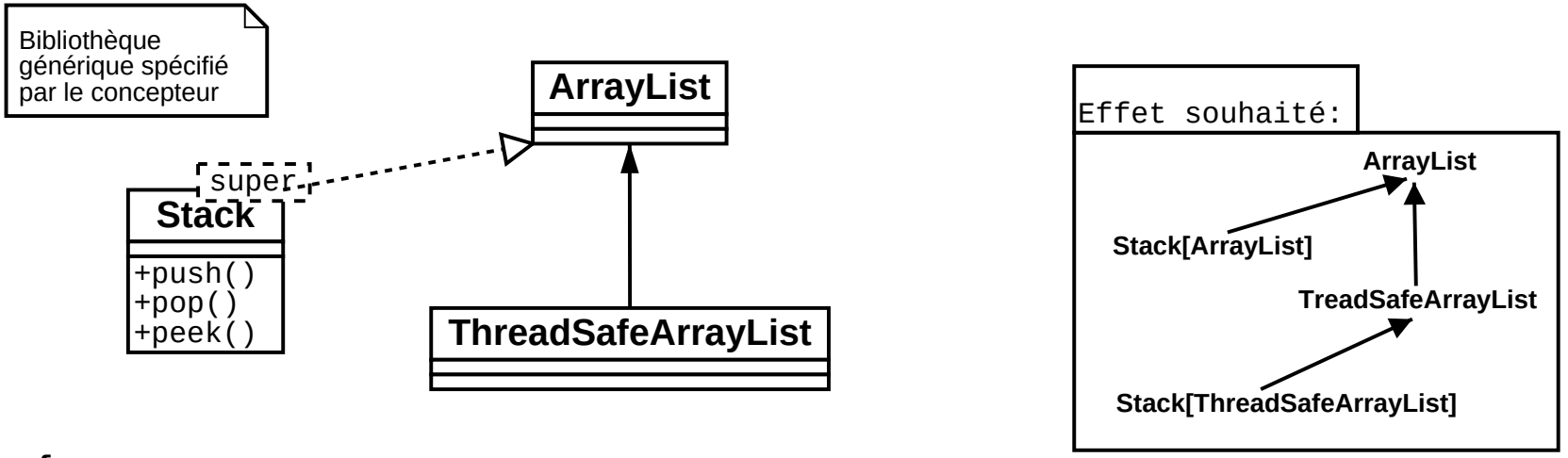





```
{
 ...
myStack1 : Stack[ArrayList];
...
}
```

Proposition : Approche intuitive

Exemple d'adaptation d'un composant : cas de l'**héritage générique**.




```
{
 ...
myStack1 : Stack[ArrayList];
...
myStack2 : Stack[ThreadSafeArrayList];
...
}
```

Adaptation interne au langage

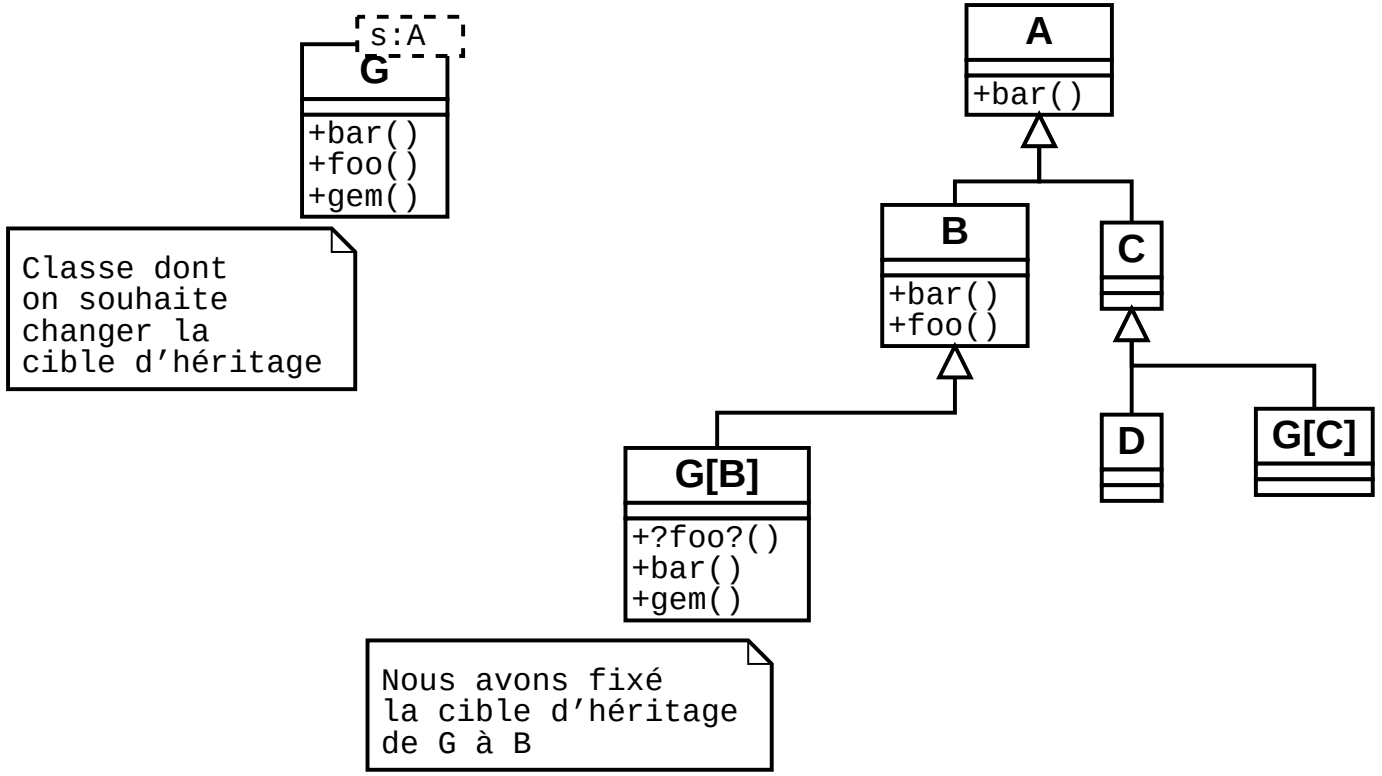
Rappel : Problèmes engendrés

Trois problèmes principaux sont relevés :

- Conflits de nom.
- Variance des redéfinitions.
- Cohérence des assertions.

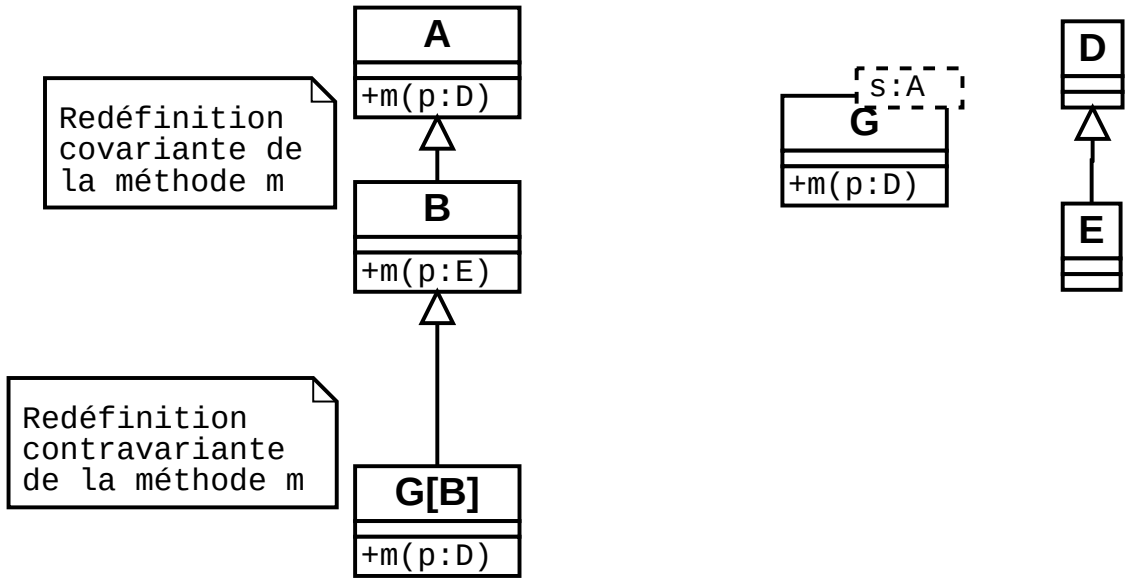
Rappel : Problèmes engendrés

Conflits de nom :



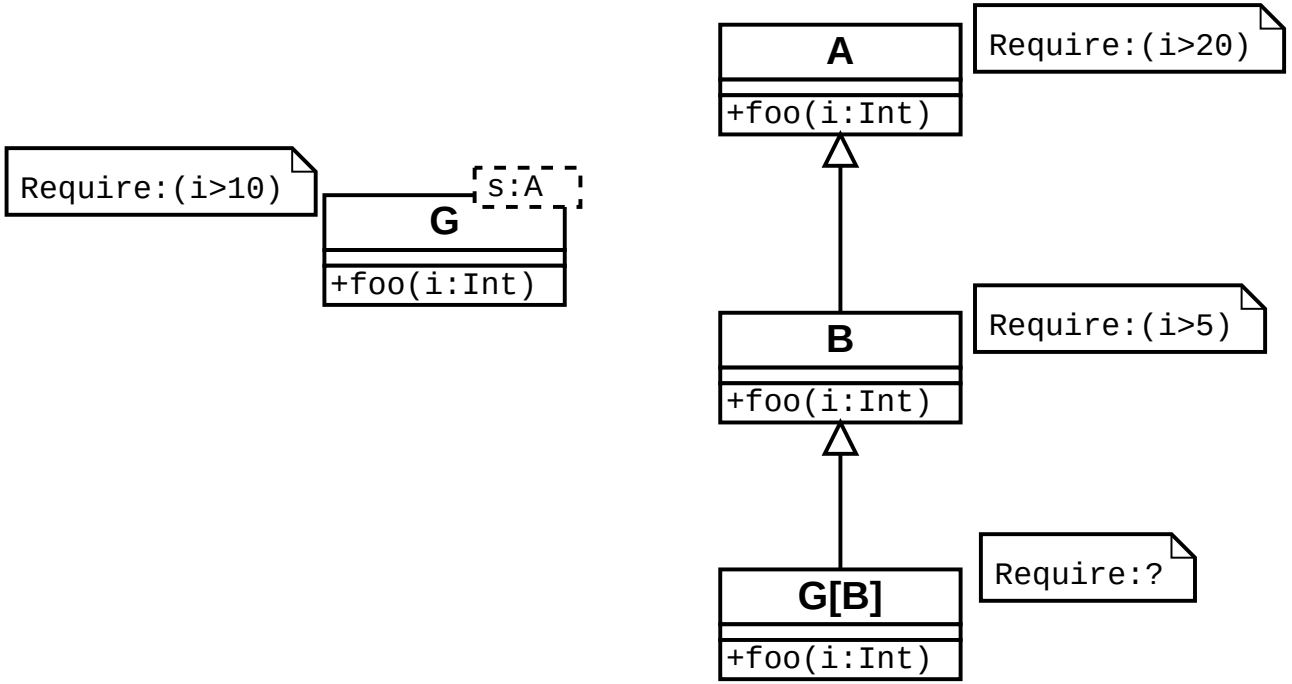
Rappel : Problèmes engendrés

Variance de la redéfinition :



Rappel : Problèmes engendrés

Cohérence des assertions :



Analyse

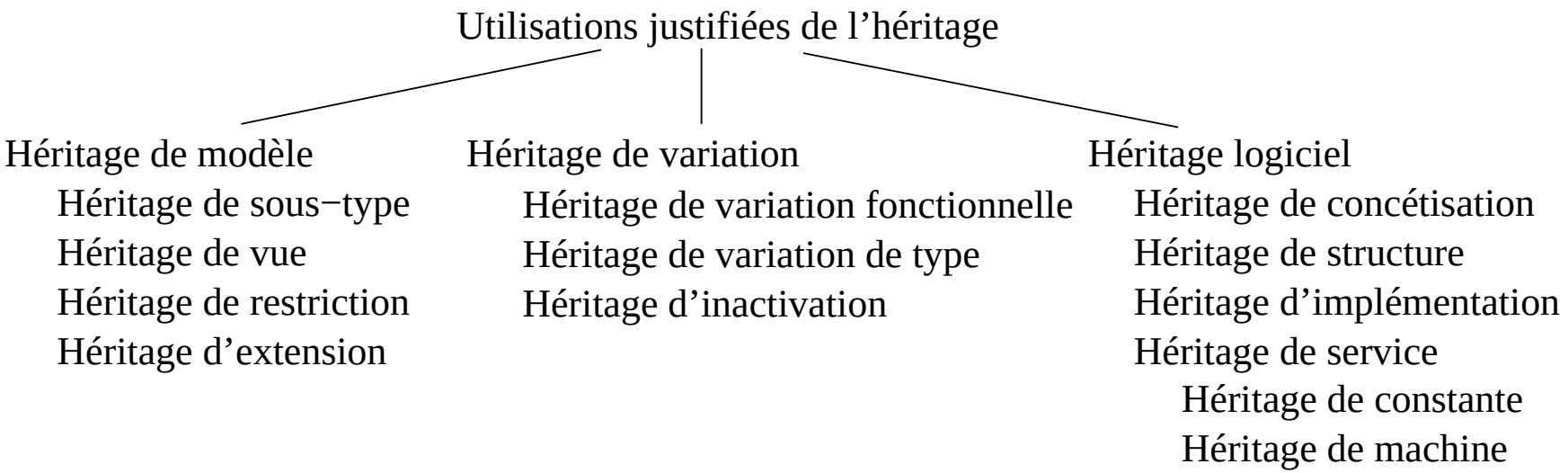
Problèmes engendrés par :

- *G* ne connait pas la sémantique réelle de la classe dont elle hérite.
- L'héritage est manipulé pour de nombreuses utilisations.

$\Rightarrow$ Proposition : prévention des problèmes par spécification de l'usage de l'héritage.

Taxinomie de l'héritage

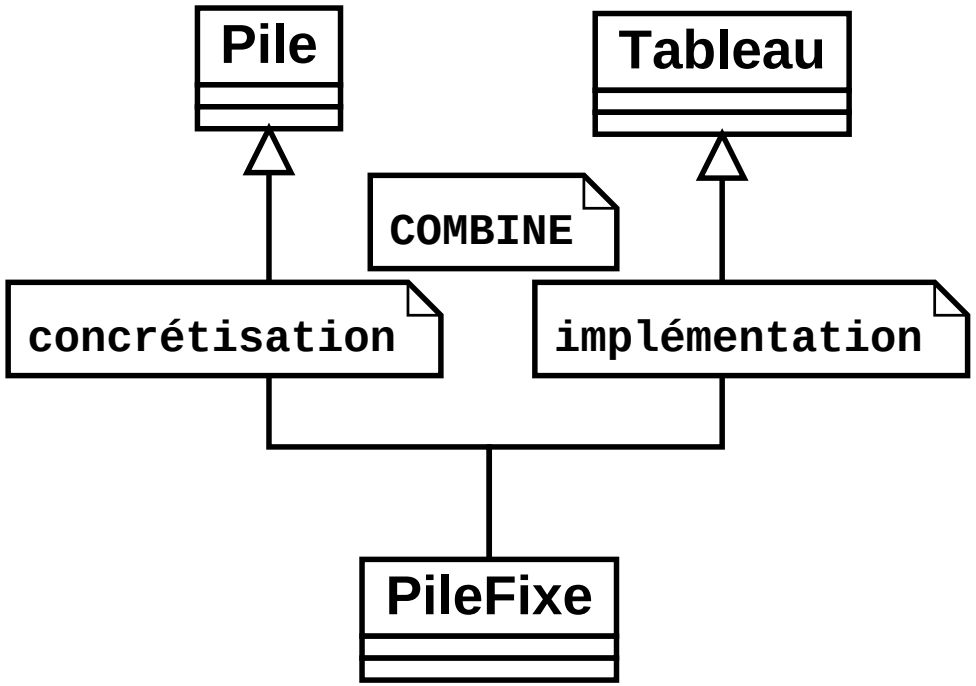
Taxinomie proposée par Bertrand Meyer.



Taxinomie de l'héritage

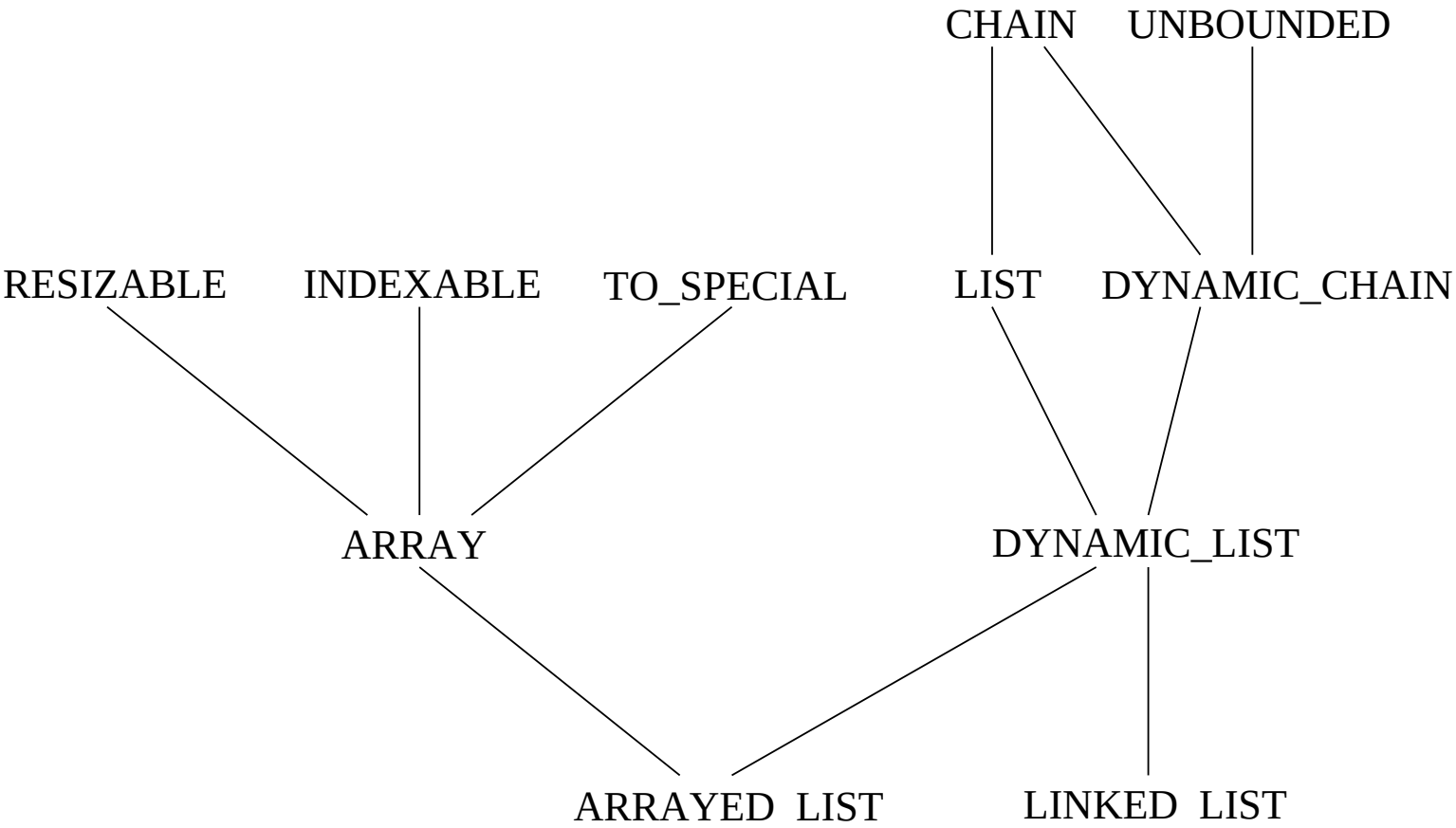
Xavier Girod propose une autre classification en ajoutant :

- ***héritage combiné*** : dépendance entre les différents types de relations lors d'un héritage multiple



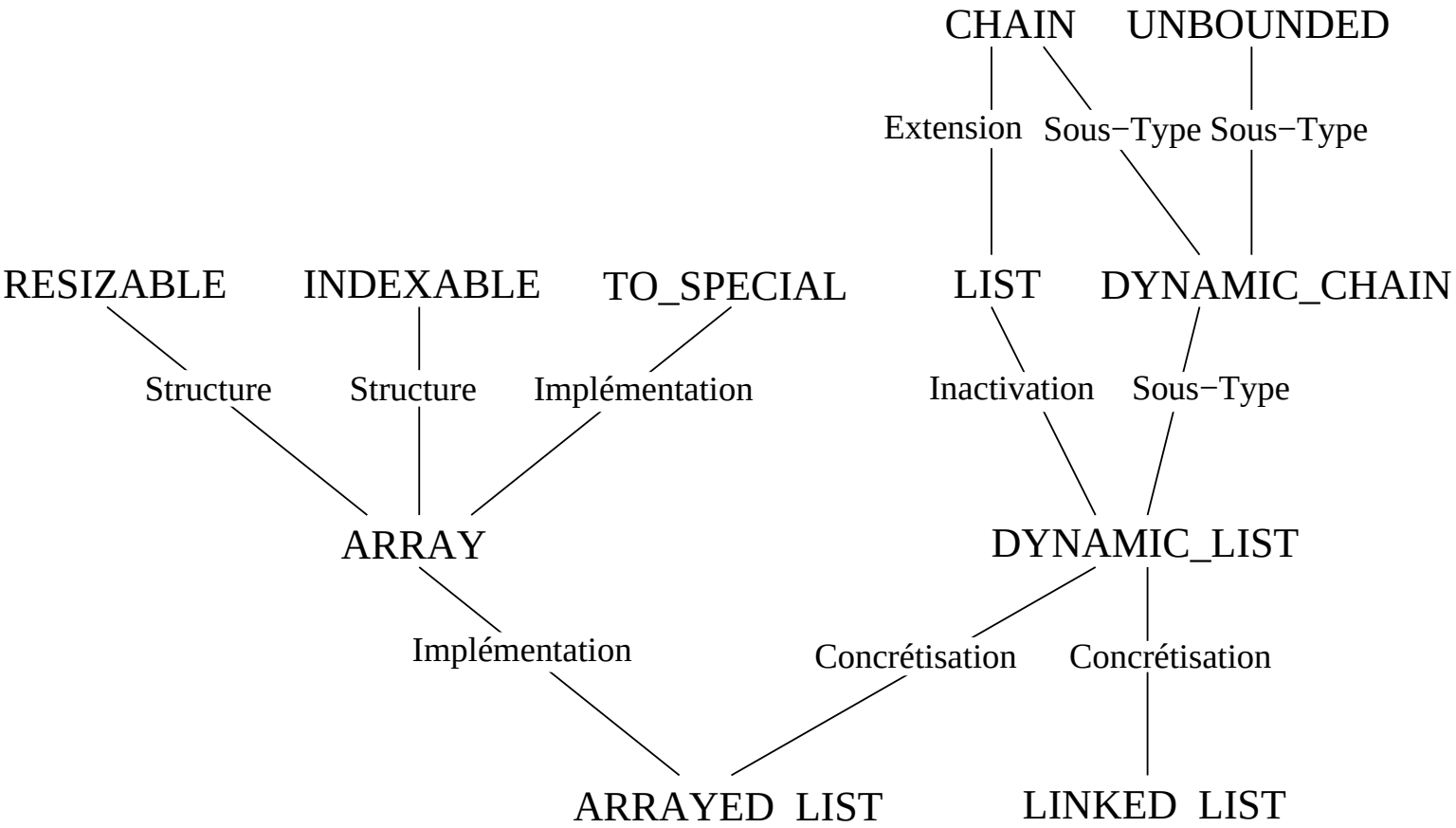
Exemple sur une hiérarchie

Extrait d'une bibliothèque de classes (*EiffelStudio 5.1*) :

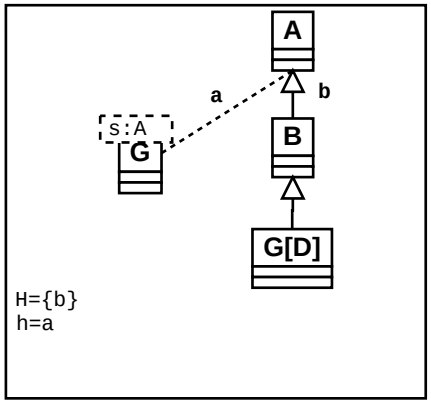


Exemple sur une hiérarchie

Extrait d'une bibliothèque de classes (*EiffelStudio 5.1*) :

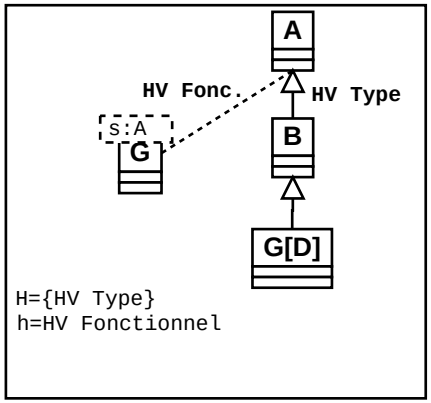


Problèmes et incohérences potentiels



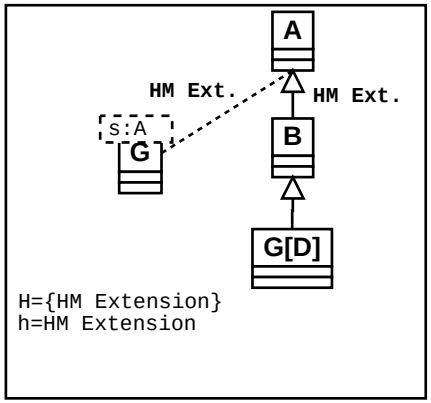
h \ ∃x∈H | x est	HV fonctionnelle	HV type	HV inactivation	HL concrétisation	HL structure	HL implémentation	HLS constante	HLS machine	HM sous-type	HM vue	HM restriction	HM extension
HV fonctionnelle	assertion	covariance	I1								assertion	
HV type	assertion	covariance									assertion	
HV inactivation	assertion	covariance	I2								assertion	
HL concrétisation	assertion	covariance		I4	I3	I3 I9					assertion	
HL structure	assertion	covariance		I5	I7 I3	I3 I9	I7				assertion	
HL implémentation	assertion	covariance	I3	I6	I7	I7	I7	I6	I6		assertion	
HLS constante	assertion	covariance	I3	I6	I7	I7					assertion	
HLS machine	assertion	covariance			I7	I7					assertion	
HM sous-type	assertion	covariance				I7	I7			I8	assertion	
HM vue	assertion	covariance				I7	I7				assertion	
HM restriction	assertion	covariance									assertion	
HM extension	assertion	covariance									assertion	conflits de nom

Problèmes et incohérences potentiels



∃x∈H \ x est	HV fonctionnelle	HV type	HV inactivation	HL concrétisation	HL structure	HL implémentation	HLS constante	HLS machine	HM sous-type	HM vue	HM restriction	HM extension
HV fonctionnelle	assertion	covariance	I1								assertion	
HV type	assertion	covariance									assertion	
HV inactivation	assertion	covariance	I2								assertion	
HL concrétisation	assertion	covariance		I4	I3	I3 I9					assertion	
HL structure	assertion	covariance		I5	I7 I3	I3 I9	I7				assertion	
HL implémentation	assertion	covariance	I3	I6	I7	I7	I7	I6	I6		assertion	
HLS constante	assertion	covariance	I3	I6	I7	I7					assertion	
HLS machine	assertion	covariance			I7	I7					assertion	
HM sous-type	assertion	covariance				I7	I7			I8	assertion	
HM vue	assertion	covariance				I7	I7				assertion	
HM restriction	assertion	covariance									assertion	
HM extension	assertion	covariance									assertion	conflits de nom

Problèmes et incohérences potentiels



∃x∈H | x est / h	HV fonctionnelle	HV type	HV inactivation	HL concrétisation	HL structure	HL implémentation	HLS constante	HLS machine	HM sous-type	HM vue	HM restriction	HM extension
HV fonctionnelle	assertion	covariance	I1								assertion	
HV type	assertion	covariance									assertion	
HV inactivation	assertion	covariance	I2								assertion	
HL concrétisation	assertion	covariance		I4	I3	I3 I9					assertion	
HL structure	assertion	covariance		I5	I7 I3	I3 I9	I7				assertion	
HL implémentation	assertion	covariance	I3	I6	I7	I7	I7	I6	I6		assertion	
HLS constante	assertion	covariance	I3	I6	I7	I7					assertion	
HLS machine	assertion	covariance			I7	I7					assertion	
HM sous-type	assertion	covariance				I7	I7			I8	assertion	
HM vue	assertion	covariance				I7	I7				assertion	
HM restriction	assertion	covariance									assertion	
HM extension	assertion	covariance									assertion	conflits de nom

Limite de cette approche

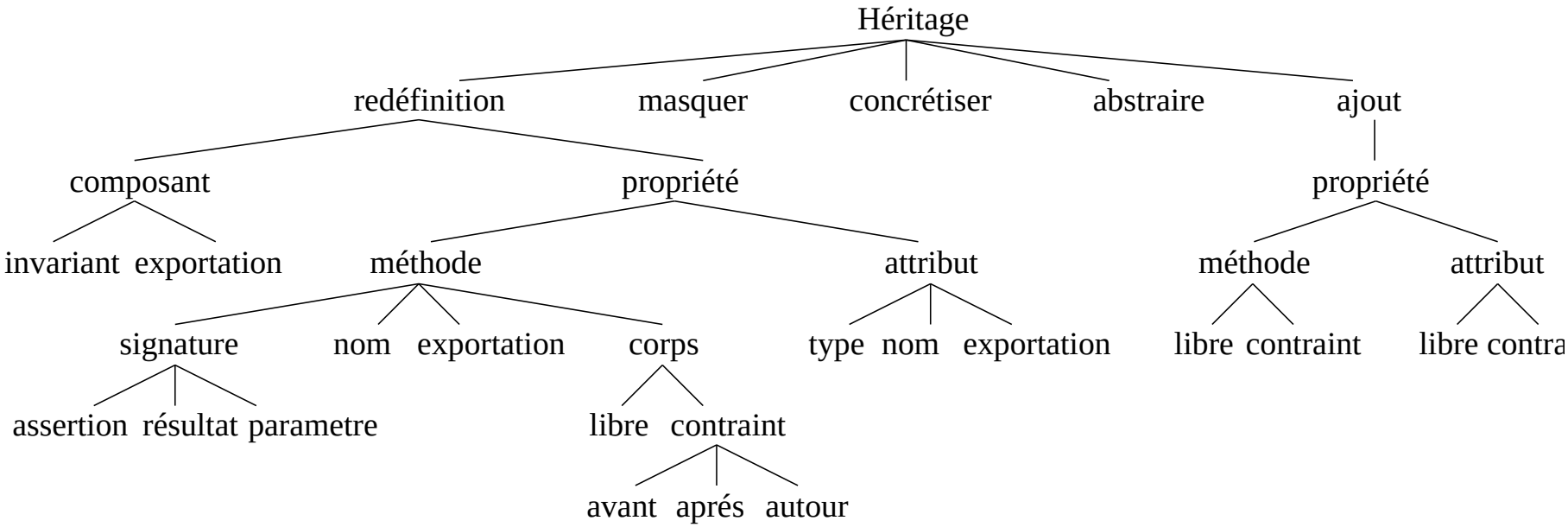
Ne prévient pas les problèmes car :

- Les cases n'étant pas notées comme incohérentes ne sont pas sûres.
- Ces relations décrivent *pourquoi* on utilise l'héritage.

$\Rightarrow$ Nous devons préciser *comment* on utilise l'héritage.

Une autre classification

Classification des possibilités offertes au travers de la relation d'héritage :



Vers le concept d'annotation

Nous avons défini un ensemble d'actions possible par l'utilisation de l'héritage.

- Une sous-classe peut décrire ce qu'elle fait par rapport à sa super-classe (*à la* "redefine" de Eiffel)
- Une classe peut décrire un ensemble d'actions *autorisé* ou *prohibé* pour ses futures sous-classes.

Généralisation de ces descriptions sous le concept d'*annotation* présentées dans [PC03].

Synthèse

Nous avons proposé de pouvoir modifier la cible de la relation d'héritage pour permettre soit :

- A l'utilisateur d'adapter un composant à ses besoins (redéfinition de la cible d'héritage)
- Au concepteur de créer des composants plus adaptables (héritage générique)

Mais des incohérences à résoudre :

$\Rightarrow$ Les prévenir par une explicitation de l'usage de l'héritage : concept d'annotation.

Conclusion et perspective

- La redéfi nition de la cible d'héritage permettrait une meilleure adaptation d'un composant.
- Les annotations préviendraient des incohérences mais sont diffi ciles à gérer ("lourdeur", spécifi cations explicites ...)

D'autres questions sont cependant ouvertes :

- Où effectuer la spécifi cation de la nouvelle cible ? (de manière interne au langage ou de manière externe)
- Peut-on spécifi er différentes cibles dans différentes parties d'une même application ? (quelle compatibilité a-t-on alors)
- Doit-on pouvoir spécialiser la classe générique en fonction de son paramètre ? (quelles adaptations de la classe par rapport à sa cible sont autorisées)

Références

[CRE01] Pierre CRESCENZO. *OFL : Un modèle pour paramétrer la sémantique opérationelle des langages à objets. Application aux relation inter-classes*. PhD thesis, Université de Nice-Sophia Antipolis - UFR Sciences, 2001.

[HÖL93] Urs HÖLZLE. Integrating independently-developed components in object-oriented languages. In *ECOOP'93*, 1993.

[HO92] William HARRISON Harold OSSHER. Combination of inheritance hierarchies. In *OOPSLA'92*, 1992.

[PC03] Philippe LAHIRE Pierre CRESCENZO, Christophe JALADY. Annotations des classes et des relations d'héritage : un mécanisme unifi é pour améliorer les facteurs de qualité des bibliothèques de classes. Soumission à MASPEGHI 2003, 2003.

[TL94] Munid QUTAISHAT Ted LAWSON, Christine HOLLINSHEAD. The potential reverse type inheritance in eiffel. *TOOLS Europe 94*, 1994.