Analyse des documents graphiques : une approche par reconstruction d'objets

15 juin 2007

Annexe B : Les langages O.O

Cette Annexe présente une introduction sur les langages Orienté(s)-Objet (O.O). Ces derniers sont basés sur le concept d'objet introduit par [Nygaard 63] dans le langage de programmation Simula. Depuis lors de nombreux langages de programmation O.O ont été développés (Java, C++, Smalltalk, Modula, ...), ils sont aujourd'hui largement utilisés en informatique pour le développement d'applications [Zamir 99]. Ces langages ont également été appliqués à la représentation des connaissances [Euzenat 98]. Ils servent dans ce cas à la représentation objet des connaissances et non plus à l'écriture des programmes [Valtchev 99]. Dans tous les cas les langages O.O emploient différents concepts communs : objet, classe, instance, héritage, spécialisation, et agrégation. Nous les introduisons dans les définitions (1) et (2) suivantes.

Définition 1 ***Objet, classe et instance :***

Un objet est un ensemble d'attributs et de méthodes opérant sur ces attributs.

Une classe est un modèle définissant les attributs et méthodes des objets.

Une instance est un objet défini par une classe.

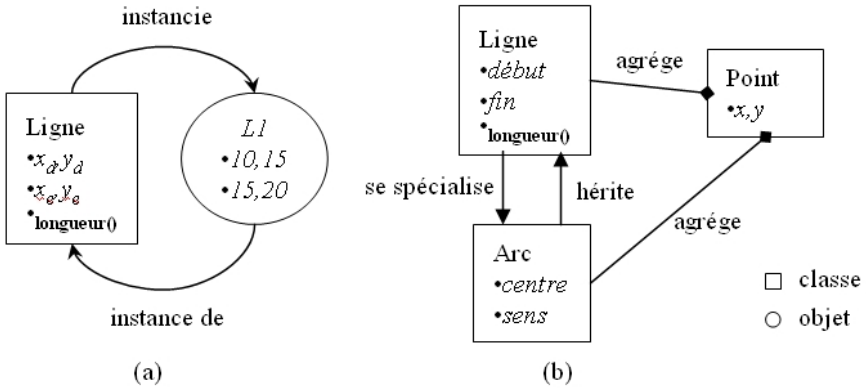
Définition 2 ***Héritage, spécialisation et agrégation :***

L'héritage est une technique de partage des attributs et méthodes entre classes.

La spécialisation est la relation inverse de l'héritage.

L'agrégation est une relation de composition entre objets instances de classes.

Les figures (a) et (b) suivantes illustrent ces différentes définitions. La figure (a) présente l'exemple d'un objet ($L1$) instance d'une classe ($Ligne$) composée d'attributs correspondant aux coordonnées de début (x_b,y_b) et de fin (x_e,y_e) de la ligne. Une méthode **longueur()** permet alors de retourner la distance euclidienne entre ces deux points. La figure (b) présente la mise en oeuvre des relations d'héritage et d'agrégation. La classe ($Ligne$) a été redéfini comme l'agrégation de deux classes ($Point$). Cette classe ($Ligne$) se spécialise également en classe (Arc) par agrégation d'un point supplémentaire ($centre$) et la définition de l'attribut ($sens$). Les classes ($Ligne$) et (Arc) sont respectivement appelée classe mère et fille.



(a) objet et instance (b) héritage et agrégation

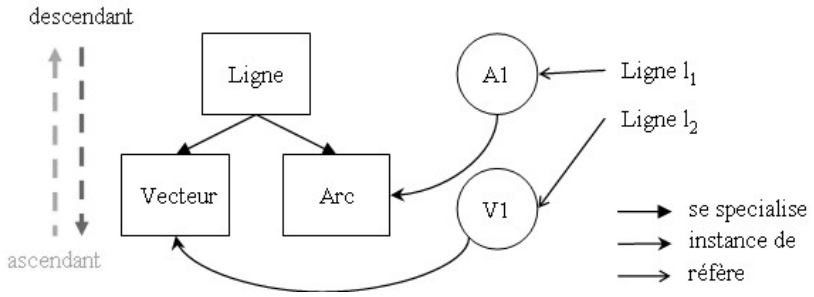
Un concept fondamental des langages O.O vient alors se greffer aux précédents : celui de polymorphisme[1]. Nous le présentons dans la définition (3) suivante. Le polymorphisme repose sur deux notions fondamentales : celle de compatibilité ascendante et celle de ligature dynamique. Nous présentons chacune de ces notions par la suite.

Définition 3 ***Polymorphisme :*** *Concept de la théorie des types selon lequel une référence peut désigner des objets de différentes classes liées par une super-classe commune. Par conséquent, tout objet désigné par cette référence peut répondre à un certain ensemble d'opérations communes.*

Nous présentons tout d'abord dans la définition (4) la notion de compatibilité ascendante. La figure suivante l'illustre à travers un exemple. Dans cet exemple une classe ($Ligne$) est spécialisée en deux classes filles ($Vecteur$) et (Arc). Ces classes sont alors qualifiées de classes descendantes de ($Ligne$), et la classe ($Ligne$) de classe ascendante aux classes ($Vecteur$) et (Arc). Deux objets (A_1) et (V_1) sont alors instanciés par ces deux classes. Ces deux objets sont désignés dans le programme par deux références (l_1) et (l_2) de type ($Ligne$). Les références sont alors des variables dans le programme désignant les objets. Il y a ici compatibilité ascendante : les relations de spécialisation entre la classe ($Ligne$), (Arc) et ($Vecteur$) permettent de manipuler les objets (l_1) et (l_2) comme des objets de classe ($Ligne$).

Définition 4 ***Compatibilité ascendante :*** *Il existe une conversion implicite d'une référence v_1 (vers à un objet de classe T_v) en une référence v_2 (vers à un objet d'une classe T_n ascendante de T_v).*

[1] Nous abordons ici le polymorphisme dit d'héritage (ou d'inclusion), nous reportons le lecteur à [Lafon 92] pour une présentation des autres catégories de polymorphisme.



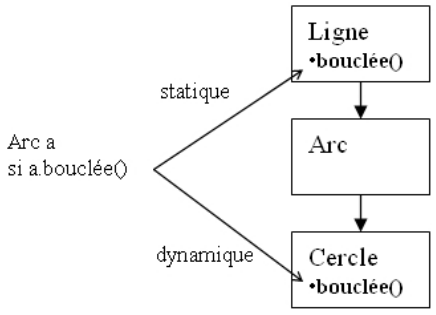
Compatibilité ascendante

Une notion fondamentale vient se greffer à celle de compatibilité ascendante : celle de ligature dynamique. Nous la présentons dans la définition (5) suivante, la figure suivante l'illustre à travers un exemple. Dans cet exemple une classe (*Ligne*) est spécialisée en classe (*Arc*) à son tour spécialisée en classe (*Cercle*). La classe (*Cercle*) re-définie la méthode **bouclée()** de la classe (*Ligne*). Une référence (a) de type (*Arc*) est alors déclarée dans le programme principale. Cette dernière est alors utilisée pour l'appel de la méthode **bouclée()**. Ce type d'appel à cheval entre deux niveaux de classe permet d'illustrer la notion de ligature dynamique. À la compilation la référence (a) réfère par ligature dite statique la méthode **bouclée()** de la classe (*Ligne*). À l'exécution, elle réfère par ligature dite dynamique la méthode **bouclée()** de la classe (*Cercle*).

Définition 5 ***Ligature dynamique :*** *Dans un appel de la forme $v.f()$ ou v est une référence déclarée et supposée de classe T_n et $f()$ une méthode de T_n. Cette déclaration est telle que pour $\{T_0, \ldots, T_n, \ldots, T_v\}$ (avec $\{T_0, T_{n-1}\}$ les classes ascendantes à T_n, et $\{T_{n+1}, \ldots, T_v\}$ les classes descendantes à T_n) le choix de $f()$ est déterminé ainsi :*

À la compilation : *on détermine dans $\{T_0, \ldots, T_n\}$ la meilleure signature de $f()$, on parle de ligature statique.*

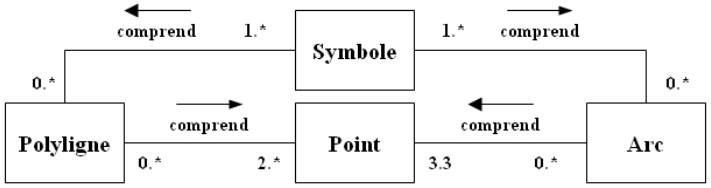
À l'execution : *on détermine dans $\{T_n, T_v\}$ la meilleure signature de $f()$, on parle de ligature dynamique.*



Ligature dynamique

Les langages O.O servent à l'écriture de programmes afin d'implémenter des systèmes informatiques. Cependant cette étape d'implémentation est généralement englobée dans un cycle plus large lié à réalisation d'un système. Ce cycle se décompose en quatre étapes principales [André 01] : l'analyse des besoins, la conception, l'implémentation et la validation. De façon à assister le programmeur dans la réalisation de son système différentes méthodes[2] dites d'analyse et de conception ont été définies ces quinze dernières années comme SADT, UML, OMT, OOSE, OBJECTORY, ... Dans le cadre de cette Annexe nous nous intéresserons uniquement à l'étape de conception[3] associée à ces méthodes. Cette dernière est généralement basée sur l'utilisation de formalismes à base de graphes. À partir de de ces formalismes les concepteurs définissent les modèles de leurs systèmes. Cette définition peut se dérouler en différentes étapes selon la méthode employée, nous nous intéresserons ici plus particulièrement à la méthode UML[4] [Sigaud 05]. En effet, UML est distinguable des autres méthodes de par sa large utilisation liée à sa standardisation par l'OMG[5] en 1997.

UML repose sur l'utilisation de différents diagrammes : de cas, de classes, de séquences, d'états, ... Dans le cadre de cette Annexe nous nous limiterons[6] à la présentation du diagramme de classes. La figure suivante en donne un exemple. Dans celui-ci quatre classes sont modélisées {*symbole*,*polyligne*,*point*,*arc*}. Les relations entre ces classes sont nommées associations en UML. Ces associations sont orientées et attribuées selon une relation ($\overrightarrow{comprend}$). De cette façon cet exemple décrit qu'un objet instance de la classe *symbole* comprendra des objets instance des classes *polyligne* et *arc* à leur tour composés d'objets instance de la classe *point*. Un seul attribut est défini pour les besoins de l'exemple mais les diagrammes de classes UML peuvent en compter plusieurs. Ces associations sont également attribuées par des cardinalités (∗.∗). Celles-ci définissent les limites inférieure et supérieure entre les objets instances des classes décrites par l'association. Par exemple un objet de classe symbole comprend de zéro à plusieurs (0,∗) objets de classe *polyligne* ou *arc*, mais ces derniers seront toujours inclus dans au moins un objet symbole (1,∗). De cette manière cet exemple défini des ensembles de symboles dans lesquels les primitives graphiques {*polyligne*,*arc*} peuvent être partagées entre les symboles.



Exemple de diagramme de classes UML

[2] Celles-ci sont définies généralement que pour sous-ensemble du cycle de réalisation.
[3] Nous reportons le lecteur à [André 01] sur les autres étapes.
[4] Unified Modeling Language.
[5] Object Management Group : http ://www.omg.org/
[6] Nous reportons le lecteur à [Sigaud 05] pour une présentation des autres diagrammes.

Bibliographie

[André 01] P. André & A. Vailly. Conception des systèmes d'information. Editions Ellipses, ISBN : 272980479X, 2001.

[Euzenat 98] J. Euzenat. *Représentation des Connaissances par Objets.* In Langages et Modèles à Objets : Etat et Perspectives de la Recherche, pages 293–319. Editions INRIA, ISBN : 2-7261-1131-9, 1998.

[Lafon 92] M.B. Lafon. Les langages à objets, principes de base, techniques de programmation. Armand Colin, ISBN : 2-200-42051-X, 1992.

[Nygaard 63] K. Nygaard. *SIMULA - an Extension of ALGOL to the Description of Discrete-Event Networks.* In IFIP Congress, volume 62 of *Information Processing*, pages 520–522, 1963.

[Sigaud 05] O. Sigaud. *Introduction à la Modélisation Orientée Objets avec UML.* Laboratoire d'Informatique de Paris VI (LIP6), France, 2005.

[Valtchev 99] P. Valtchev. *Construction Automatique de Taxonomies pour l'Aide à la Représentation de Connaissances par Objets.* Thèse de Doctorat, Université de Joseph Fourier, Grenoble, France, 1999.

[Zamir 99] S. Zamir & al. Handbook of object technology. CRC Press, ISBN : 0849331358, 1999.

Table des figures

Concepts orientés objets 1
Compatibilité ascendante 2
Ligature dynamique 2
Exemple de diagramme de classes UML 3

Table des matières

Annexe B : Les langages O.O **0**

Bibliographie **4**

Table des figures **5**