Action LeTool

Environnement de Programmation Orienté-Objets et Interfaces Visuelles

Localisation : *Sophia-Antipolis*

Mots-clés : assistance à l'utilisateur, environnement de programmation, langage à objets, qualité du logiciel, CLOS, Common Lisp, interface homme-machine, psychologie cognitive, ergonomie cognitive, production de document, conception par objets, programmation par objets, programmation par objets colorés, programmation visuelle.

1 Composition de l'équipe

Responsable scientifique

Henry Borron, chargé de recherche

Assistante de projet

Marie-Line Ramfos

2 Présentation du projet

Après avoir participé à deux projets ESPRIT, l'un portant sur la conception d'un langage orienté objet typé et de son environnement de programmation, l'autre sur la mise au point d'un éditeur de documents en phase brouillon, l'action s'est donnée deux objectifs : (1) définir une nouvelle forme de programmation par objets, la programmation par « objets colorés » ; (2) tester celle-ci en vraie grandeur en l'utilisant pour implémenter l'outil d'aide à la rédaction précédemment défini.

Une caractéristique commune aux deux parties est l'approche choisie : l'émiettement (du code ou du texte) par l'utilisateur et sa reconstruction par l'ordinateur au moyen de règles ; une autre est le type d'interface, fortement visuelle, de l'environnement proposé à l'utilisateur pour sa tâche de conception (programmation ou rédaction).

Comme l'an passé, le travail effectué cette année a porté essentiellement sur la première partie.

3 Actions de recherche

Mots-clés : assistance à l'utilisateur, environnement de programmation, langage à objets, qualité du logiciel, Lisp, CLOS, Common Lisp, conception par objets, programmation par objets, programmation

par objets colorés, programmation visuelle, interface homme-machine, psychologie cognitive, ergonomie cognitive.

Participant : Henry Borron

Notre travail s'est développé selon deux axes : le premier, de nature purement informatique, a porté successivement sur une spécification détaillée du formalisme proposé, le concept de "mixin" et l'héritage ; le second, moins classique, explore d'une part les aspects ergonomiques du formalisme visuel, d'autre part les implications cognitives du type de programmation envisagé.

3.1 Formalisme

Considérant que la programmation par objets s'est arrêtée au milieu du gué, notre approche en reprend l'idée de base (responsabiliser les structures de données) et la pousse à fond en faisant dépendre la réponse d'un objet à un message non seulement de la classe de cet objet mais aussi de son état. À chaque classe est donc attaché un graphe décrivant les possibles états d'une instance de cette classe et les transitions entre ceux-ci induites par les requêtes externes (messages ou fonctions génériques). Un tel graphe est appelé « graphe coloré », le terme de couleur évoquant métaphoriquement la notion d'état. Les états sont généralement décrits dans un espace à N dimensions : ceci accroît la modularité et permet de lutter contre l'explosion combinatoire du nombre d'états. Trois types de constructions peuvent lier les noeuds d'un graphe (i.e. états ou contributions à des états) : sélection (OU), conjonction (ET) et décomposition (d'un état en contributions). La sémantique de chaque type de construction peut être exprimée grâce à une sémantique plus simple, celle des « transitions réflexes » liant un noeud à un autre noeud (une fois armée, une telle transition fait feu si la condition d'arrivée est vraie). Pour simplifier l'expression des « transitions régulières » (dues à des requêtes externes), on est amené à introduire un héritage local le long des transitions réflexes. Spécification externe du comportement des instances d'une classe, un graphe coloré peut être augmenté par la définition de représentations en mémoire (une représentation par noeud du graphe) et de micro-méthodes (une pré-méthode et une post-méthode par transition). Ceci induit une distinction claire entre spécification et implémentation, ainsi qu'un haut degré de modularité (plusieurs implémentations différentes peuvent être définies pour une même spécification). Les changements de représentations sont gérés automatiquement par le système. L'héritage local le long des transitions réflexes s'étend aux micro-méthodes et aux représentations en mémoire.

3.2 Mixins

Notre proposition généralise la notion de mixin qui existe dans certains langages à objets sophistiqués. Nous ne lui connaissons aucun équivalent dans un espace d'états à N-dimensions. Alors qu'une classe décrit le comportement d'objets instantiables similaires, les mixins spécifient des suppléments de comportements indépendants, mais non instantiables en tant que tels. Un mixin est donc destiné à être attaché à une classe (dite de base) pour produire une nouvelle classe (dite dérivée). L'opération en question est appelée dérivation. Après avoir défini la forme générale d'un mixin (une sélection à multiples niveaux), nous avons montré qu'il était essentiel de distinguer la topologie de chaque transition au point d'attache de la classe de base (transition circulaire, transition sortante ou transition impure, i.e. sortante mais avec une possibilité de retour au point d'attache via des transitions réflexes). Chaque cas implique certaines contraintes pour les transitions similaires dans le mixin. A partir de là, on peut définir des règles par défaut qui permettent une expression simplifiée des transitions dans un mixin. A chaque mixin sont attachées ses contraintes d'emploi (i.e. les transitions requises au point d'attache de la classe de base). Il est possible de définir des mixins paramétrés pour couvrir différents cas par des mixins bien adaptés et donc plus simples. Ceci permet de prendre en compte les cas limites –comme une pile ne pouvant contenir qu'un seul élément– tout en préservant l'efficacité : le choix du mixin effectif est fait à la création de l'instance.

3.3 Héritage

En termes de spécifications, deux mécanismes existent pour créer une classe nouvelle à partir de classes existantes. Le premier, appelé composition, combine les graphes colorés de plusieurs classes de base (superclasses) ; toutes les dimensions distinctes sont combinées (produit cartésien). Le second mécanisme est la dérivation ; en ce cas, il y a normalement ajout d'une ou plusieurs dimensions (celles du mixin) seulement en un seul point de l'espace de base. Une hiérarchie de classes résulte de ces mécanismes.

En termes d'implémentations, notre proposition développe le concept d'héritage dans une hiérarchie par extension progressive du concept d'héritage local. Appelée « héritage par dimensions », son principe consiste en une recherche des items (micro-méthodes, représentations en mémoire) selon chaque dimension « impliquée ». Concernant les représentations en mémoire, les dimensions non abstraites sont toutes impliquées ; de plus, les contraintes habituelles rendent la solution aisée. Concernant les micro-méthodes, les dimensions impliquées sont celles de la transition correspondante. Etant donnée une dimension impliquée, la recherche est faite selon la linéarisation des classes affectant cette dimension. Le résultat est un « diagramme d'invocation » listant les méthodes selon toutes les dimensions impliquées et, s'il y a lieu, selon les dimensions utilisées (par une telle méthode) mais non impliquées (par la transition) : par exemple, pour réaliser une impression, la transition correspondante (définie dans la classe *Objet*) implique seulement la dimension *objet*, tandis que la méthode fait généralement appel à toutes les autres dimensions. Si une certaine condition (dite de « régularité ») est vérifiée, alors le diagramme d'invocation peut être mis sous la forme d'une liste d'arbres : chaque arbre est dû à une ou plusieurs listes de méthodes, listes initialement parallèles (elles traversent les mêmes classes), puis progressivement divergentes. Une hiérarchie est « régulière » vis-à-vis de ses méthodes m si toutes les méthodes m de degré supérieur ou égal à K qui existent selon une même dimension satisfont les mêmes K dimensions. (Cette condition est vérifiée en pratique.) L'ordre original des méthodes selon chaque dimension est éventuellement remanié de manière à privilégier les méthodes les plus spécialisées par rapport à celles qui le sont moins (règle de prévalence des méthodes combinant un plus grand nombre de dimensions), d'où la structure en liste d'arbres. Il est bien sûr possible de masquer explicitement des méthodes, d'ailleurs d'une manière plus fine que dans les langages à objets traditionnels. Les méthodes ainsi ordonnées et retenues sont combinées selon deux combinaisons a priori distinctes : la première est utilisée le long d'une même branche ; la seconde, quand des branches divergent. Ceci constitue la technique de base. L'ajout de mots-clés aux déclarations des méthodes permet des combinaisons plus sophistiquées. Non seulement notre proposition généralise ainsi celle de CLOS au cas des objets ayant des états, mais elle permet en outre un style purement déclaratif : le style impératif ("call-next-method"), non modulaire (nécessité d'examiner le corps des méthodes), est évacué. Concernant la linéarisation, nous avons été amenés à définir une nouvelle propriété, la « congruence », qui implique que la linéarisation de toute sous-hiérarchie de racine *C0* et de sommet *C* soit une sous-liste de la linéarisation de la hiérarchie de racine *C0*. Nous avons montré que l'algorithme de linéarisation de LOOPS était congruent et qu'un algorithme de linéarisation aveugle ne pouvait être à la fois monotone et congruent. Nous avons également élaboré un nouvel algorithme de linéarisation monotone.

3.4 Ergonomie

L'ergonomie du formalisme visuel a été étudiée en termes de dimensions cognitives (expression des rôles, notamment). Les règles pratiques de conception et d'utilisation ont été détaillées. Concernant les représentations alternatives, l'interprétation dans un espace cartésien a été explicitée ; de plus, une comparaison avec les graphes hiérarchiques de David Harel a été faite.

3.5 Psychologie cognitive

Assez récemment, la psychologie cognitive a établi deux points importants : (1) l'activité de programmation, comme toute activité de conception, est de nature opportuniste (travaux de Guindon et alii) ; (2) la programmation par objets traditionnelle convient bien à une démarche descendante mais beaucoup

moins à une démarche opportuniste (travaux de T.R.G. Green et alii). De fait, les environnements de programmation correspondants sont nécessairement mal adaptés aux tâches cognitives de l'utilisateur et il en résulte un coût économique. Nous avons montré que notre approche était susceptible d'entraîner une amélioration importante sur ce point. Ceci est notamment lié au fait qu'un graphe coloré décrit en un seul endroit –et non pas de manière dispersée dans une hiérarchie– le comportement global des instances d'une classe (le graphe inclut les comportements hérités des superclasses et autres ancêtres de la classe considérée).

4 Actions nationales et internationales

4.1 Actions nationales

Nous avons dépeint les aspects cognitifs de notre approche à l'équipe de psychologie de la programmation conduite par Françoise Détienne (U.R. de Rocquencourt).

4.2 Actions internationales

4.2.1 Moyen-Orient

Nous avons été invités à présenter notre travail (formalisation d'objets multi-dimensionnels, héritage par les dimensions) à David Harel (Weizmann Institute of Science, Israël) et à l'équipe recherche et développement d'Ilogix (novembre 1996). Notre mécanisme d'héritage est en effet adaptable au formalisme des graphes hiérarchiques utilisés pour décrire les systèmes réactifs ; la comparaison des deux formalismes est également un point d'intérêt.

5 Diffusion des résultats

5.0.2 Actions d'enseignement

Nous avons à nouveau fait un cours sur les protocoles métaobjets en CLOS à l'ENTPE de Lyon (janvier).

5.0.3 Participation à des colloques

Nous avons décrit les aspects cognitifs de la programmation par objets colorés (ergonomie du formalisme, impact sur l'activité de programmation) à la conférence ERGO-IA (10 octobre 1996).

6 Publications

Communications à des congrès, colloques, etc.

[159] H. BORRON, «Colored-Object Programming : Ergonomic and Cognitive Issues», *in : ERGO-IA'96 Proceedings*, Biarritz, France, 1996.

Rapports de recherche et publications internes

[160] H. BORRON, «Color Graphs, a visual formalism for synthesizing the behaviour of objects», *rapport de recherche n°2876*, Sophia-Antipolis, avril 1996.

[161] H. BORRON, «Colored-Object Programming : About the Programming Activity», *rapport de recherche n°2880*, Sophia-Antipolis, avril 1996.

[162] H. BORRON, «Colored-Object Programming : About the Visual Formalism», *rapport de recherche n°2879*, Sophia-Antipolis, avril 1996.

[163] H. BORRON, «Colored-Object Programming : Inheritance by Dimensions», *rapport de recherche n°2878*, Sophia-Antipolis, avril 1996.

[164] H. BORRON, «Colored-Object Programming : Mixin and Derivation, two conjoint concepts for a rigorous handling of independent supplementary behaviours», *rapport de recherche n°2877*, Sophia-Antipolis, avril 1996.

7 Abstract

Major works done this year about Colored Object Programming deal with both computer science and cognitive psychology. Concerning the first aspect, we first precisely and concisely described our formalism, then we elaborated the concept of mixin and finally what we call "inheritance by dimensions". This last point extends local inheritance to class inheritance. It results in a sophisticated and purely declarative mechanism (no "call-next-method" inside method bodies). Concerning the cognitive aspect, we showed that our proposal is likely to support opportunistic programming in a much better way than traditional object oriented programming, hence a possibly important impact from an industrial point of view.