Mecanisme pentru reutilizarea de clase in tehnologia orientata-obiect

Ciprian-Bogdan Chirila

chirila@cs.upt.ro

1. Introducere

Utilizarea unei relatii de mostenire inversa (MI) intre clase poate usura atingerea scopului de reutilizabilitate, adaptabilitate si extensibilitate a sistemelor software orientate obiect. Ideea de MI a aparut initial in lumea bazelor de date obiectuale pentru omogenizarea schemelor si apoi a fost utilizata in reorganizarea ierarhiilor de clase orientate-obiect.

In prima etapa a proiectului am propus spre realizare o semantica a acestei relatii de MI pentru limbajul Eiffel. Semantica descrie pe baza de exemple, gramatici si reguli modalitatile de utilizare a acestei relatii in cadrul limbajului Eiffel. Definirea semanticii se imparte in trei parti. Prima parte se va ocupa de generalitatile privind aceasta relatie, posibilitatea de a crea ierarhii simple de clase folosind MI. A doua parte se refera la adaptarile potentiale ce pot fi oferite de aceasta relatie intre clase si limitarile ce apar. A treia parte se va referi la analiza problemelor de legare dinamica si constrangerile ce apar in legatura cu membrii mosteniti invers. Tot in cadrul primei etape se va dezvolta si o relatie de reutilizare de tip like-type (LT) auxiliara si se va adapta semantic pentru limbajul Eiffel, datorita faptului ca in practica relatia MI combinata cu LT sa ofere flexibilitate sporita in manipularea ierarhiilor de clase.

In a doua etapa se va face o evaluare a abordarii pe baza de studii de caz cu caracter general. Totodata se va initia dezvoltarea unui prototip care sa permita utilizarea practica a relatiilor MI si LT in limbajul Eiffel. Se va alege o solutie de implementare ce va fi dezvoltata. Se va propune un model al clasei sursa a relatiei MI si un alt model pentru relatia LT. Pentru obtinerea unui sistem software executabil se propune scrierea unui translator pentru transformarea unui sistem software scris in limbajul Eiffel extins, folosind MI si LT intr-un sistem cu comportament identic, scris in limbajul Eiffel pur (fara MI si LT).

2. Stadiul actual al mecanismelor pentru reutilizarea de clase

Proiectul propus are ca obiective reutilizarea, adaptarea si evolutia sistemelor software. Dintre acestea am selectat sistemele software orientate-obiect datorita faptului ca fac parte dintr-un domeniu ce prezinta un mare potential de cercetare si datorita raspandirii largi pe care acestea o au in industrie. Pentru atingerea obiectivelor se propune utilizarea unei relatii de mostenire inversa si a unei relatii Like-Type.

Stadiul actual al cercetarii in domeniu dezvaluie o serie de mecanisme sau tehnici ce sunt destinate adaptarii si evolutiei sistemelor software obiectuale. Dintre cele mai importante vom alege spre discutie cele ce au ca scop reutilizarea la nivel de clasa: mostenirea intre clase simpla si multipla, relatia like-type, rolurile, view-urile, mixin-urile, traits-urile.

Mostenirea intre clase este unul din conceptele centrale pe care se bazeaza intreaga paradigma obiectuala. Unii cercetatori considera ca diversitatea implementarii acestui concept conduce la o diversitate a paradigmelor obiectuale [Fro02]. Mostenirea este un mecanism incremental ce transforma o superclasa in subclasa prin augmentare. O data cu relatia de mostenire intre clase, e implicata de regula si relatia de subtipizare, superclasa este supertip, iar subclasa subtip. Scopul mostenirii intr-un sistem obiectual poate fi reutilizarea (date sau cod) [Dao02] sau clasificarea (organizarea in ierarhie) [Bla02].

Mostenirea poate fi simpla sau multipla dupa numarul superclaselor implicate in relatie. Mostenirea multipla poate fi utilizata in combinarea a doua concepte sau pentru reutilizarea de cod si date. Mostenirea multipla, depinzand de implementare, poate introduce o serie de probleme, conflicte pentru care in literatura s-au propus diverse solutii [CMR02,Cut06]. Cea mai cunoscuta este problema unificarii sau a duplicarii unui membru mostenit multiplu pe mai multe cai, aceasta problema fiind cunoscuta in literatura si problema mostenirii in diamant [Mey97,Mey02].

Relatia like-type este o relatie intre clase prin care se poate obtine o clasa destinatie in urma unor operatii de adaugare sau chiar anulare de membri din clasa sursa. Relatia like este cea mai generala relatie posibila intre clase si le include pe toate celelalte existente.

Rolurile sunt abstractiuni ale preocuparilor si formalizeaza separatia intre acestea. Programarea bazata pe roluri permite descompunerea obiectelor in mai multe roluri. Rolurile se pot atasa sau detasa liber in timpul vietii unui obiect. Ca implementare in literatura exista mai multe solutii. Una dintre acestea implica utilizarea tiparelor de proiectare pentru roluri care sunt de fapt tehnici ad-hoc de organizare a claselor in vederea integrarii mai facile a rolurilor. Alte abordari propun modele ce suporta evolutia adaptiva, descrierea separarii preocuparilor, reutilizarea avasata a mediilor ce contin rolurile [TUI05].

Mixin-urile sunt derivate din programarea generica si sunt clase ce au ca parametru generic superclasa lor. Ideea de baza este de a specifica o extensie fara ca fi obligati sa specificam si entitatea ce va fi extinsa. Aceasta este echivalent cu a specifica doar subclasa lasand superclasa ca parametru ce va fi specificata ulterior. Avantajul consta in faptul ca o singura clasa este folosita pentru specificarea unei extensii incrementale valida pentru o varietate de alte clase. Mixin layers [Sma02] reprezinta un model derivat din mixin cu scopul de a implementa roluri si colaborari.

Traits-urile [Sch02,Sch03] sunt mecanisme simple pentru organizarea sistemelor orientate-obiect. Un trait este un set de metode parametrizabil ce poate fi asamblat in clase, reprezentand entitatea primitiva de reutilizare. Folosind traits-uri clasele sunt organizate tot in ierarhii de clase bazate pe mostenire simpla dar acestea pot fi utilizate si in a specifica diferenta incrementala intre subclasa si superclasa. Pentru acest mecanism, mostenirea nu reprezinta un operator de compozitie ca si in cazul mostenirii multiple sau a mixin-urilor, el avand proprii sai operatori de compozitie.

Pentru a atinge scopul de adaptare si evolutie a sistemelor orientate-obiect, se propune utilizarea unui mecanism ce are la baza unul din conceptele centrale ale paradigmei obiectuale si anume relatia de mostenire intre clase. Mecanismul propus pentru dezvoltare este relatia de mostenire inversa intre clase. Alaturi de aceasta relatie se incearca si utilizarea relatiei like-type pentru a permite o manipulare mai flexibila a ierarhiilor de clase.

Relatia de mostenire inversa a aparut pentru prima data in literatura in lumea bazelor de date obiectuale datorita necesitatii de a reutiliza impreuna diverse scheme de baze de date. Ideea de baza consta in a crea o clasa de baza, care sa contina toti membrii comuni ai subclaselor si ierarhia rezultata sa fie echivalenta cu una construita cu ajutorul mostenirii clasice sau directe. Necesitatea de a omogeniza doua sau mai multe scheme a condus la aparitia unor mecanisme de adaptare a acestora. Unul dintre mecanisme presupune redenumirea unor campuri in vederea obtinerii unui nume comun pentru acesta. Alt mecanism propune prevederea de rutine de transformare a unor campuri numerice ce provin din scheme diferite si care au scale diferite. Implementarea acestor mecanisme este facuta pe baza unor metaclase care incapsuleaza in ele eventualele rutine de conversie necesare. Evident ca aceste adaptari sunt limitate doar la valori numerice, dar ideea este buna, iar mecanismul poate fi imbunatatit.

Ulterior aceasta relatie de mostenire inversa a fost integrata si in contextul programarii orientate pe obiecte. In [Ped89] unde relatia este numita generalizare, se analizeaza din doua puncte de vedere: cel al factorizarii interfetei si implementarii. Mai sunt prezentate si aspecte favorabile ale mostenirii inverse singulare, caz in care se face factorizarea de la o singura clasa sursa.

In [Sak02], unde aceeasi relatie este numita ex-mostenire, se critica abordarea din [Ped89]. Sunt discutate aspectele legate de factorizarea membrilor comuni, de posibilitatile de a factoriza interfata si implementarea acestora. Mai sunt prezentate si doua tipuri de conflicte ce pot aparea in cadrul factorizarii: conflictele de nume si conflictele de valoare. Conflictele de nume apar cand doi membri cu semantica diferita au acelasi nume, iar conflictele de valoare se intalnesc in cazul cand doi membri cu aceesi semantica au nume diferite. Tot in acest context sunt prezentate si efectele combinarii mosteniri inverse cu mostenirea clasica directa.

Lucrarea [LHQ94] prezinta cateva aspecte ale mostenirii inverse. Abordarea are ca obiect mostenirea inversa in contextul limbajului Eiffel. Sunt discutate problemele legate de conflictele de nume ale carei posibile solutii este redenumirea. De asemenea mai sunt prezentate problemele legate de asertiuni si de modalitatea in care acestea pot fi modificate la nivelul superclasei astfel incat mostenirea inversa sa fie valida. Probleme legate de legarea dinamica sunt formulate, dar nu se propune nici o solutie.

Lucrarea [Chi04a] prezinta mecanismul de factorizare al relatiei MI si potentialele problemele care pot aparea in contextul utilizarii acestei relatii in proiectarea ierarhiilor de clase.

Lucrarea [Chi04b] dezvolta idei in jurul mecanismului de factorizare cu rolul de a adapta membrii exmosteniti din subclase.

3. Contributii

Pentru a aduce contributii la cele mai recente idei din literatura, proiectul se imparte in doua mari obiective: elaborarea semanticii mostenirii inverse, a relatiei Like-Type si dezvoltarea prototipului care sa permita utilizarea acestor relatii in practica.

3.1.Notatii

MO = mostenire obisnuita clasica, poate si simpla sau multipla

MM = mostenire multipla

MI = mostenire inversa

LT = relatie de reutilizare like-type

3.2. Motivatii

Relatia de mostenire inversa prezinta mai multe aspecte ce contribuie la reutilizarea claselor, in particular ne putem adresa claselor ce provin din ierarhii ce au fost dezvoltate in diferite contexte. Un alt tip de reutilizare oferit de mostenirea inversa il reprezinta adaptarea unei ierarhii existente pentru un context special.

Folosind relatia de mostenire inversa proiectarea ierarhiilor se face intr-o maniera naturala, pentru ca este mult mai natural sa se proiecteze subclasele prima data, dupa care sa se factorizeze membrii comuni si sa se identifice superclasele.

Totodata mostenirea inversa poate ajuta la captarea functionalitatilor comune din diverse clase, oferind posibilitatea ca acestea sa fie utilizate uniform prin intermediul unei clase de baza create ulterior proiectarii ierarhiei din care provin clasele originale. Totodata evolutia in sus a unei ierarhii de clase poate fi realizata cu ajutorul acestei relatii.

Deseori apar situatii in care unele ierarhii de clase sunt proiectate fara a se tine cont de eventualele necesitati ulterioare ce pot aparea. Insertia unei clase in interiorul unei ierarhii poate fi realizata cu ajutorul relatiei de mostenire inverse. O asemenea reorganizare de clase poate fi utila si in contextul adaptarii unei ierarhii de clase pentru o situatie particulara.

Combinand relatiile de mostenire inversa si Like-Type, se poate extrage comportament din clase deja existente pentru a fi reutilizat in contextul altor clase noi. O astfel de facilitate poate conduce totodata la crearea unui nou tip. Mai mult, daca se utilizeaza si relatie de mostenire clasica se poate ajunge la descompunerea si recompunerea de clase. Se pot extrage comportamente diverse din clase diferite care apoi pe baza mostenirii multiple sa fie compuse intr-o clasa unica.

Privind alegerea unui limbaj de programare in contextul caruia sa fie definita aceasta relatie s-a tinut cont de mai mult aspecte. Este important ca noua relatie care se va integra in limbaj sa respecte filosofia acestuia. Totodata este necesar sa se respecte simetria limbajului, adica sa nu se permita constructia de ierarhii folosind extensia de limbaj ce nu au echivalent in limbajul original. Analiza s-a facut in contextul limbajelor Java, C++, Eiffel. Printre argumentele in favoarea limbajului Eiffel mentionez: prezenta mecanismului de redenumire care poate fi usor extins in cadrul MI pentru a facilita factorizarea membrilor comuni, lipsa posibilitatii de supraincarcare a metodelor garanteaza unicitatea de nume pentru membri, ceea ce faciliteaza mult o eventuala implementare, existanta mecanismului de mostenire multipla.

In [Sak02] se mentioneaza ca acest concept nu este dezvoltat complet in literatura si ca nu exista nici un limbaj de programare care sa-l implementeze.

3.3. Raportarea MI la alte mecanisme similare

In primul rand mostenirea inversa trebuie raportata la mostenirea obisnuita (MO) [Mey02]. In situatiile in care se doreste reorganizarea unei ierarhii o solutie ar fi rescrierea ierarhiei folosind mostenirea obisnuita. Dezavantajele acestei solutii sunt legate de obligativitatea de intretinerea codului sursa modificat. Exista insa si posibilitatea ca sursele sa nu poate fi modificate din motive de copyright. De asemenea in cazurile cele mai defavorabile codul sursa poate chiar sa nu fie disponibil, problema fiind si mai acuta.

Comparand relatiile MI, LT si MM cu mecanismele de traits se pot face cateva asemanari si deosebiri. Partea comuna se refera la capabilitatile de a compune o clasa. Mecanismele de traits compun clasele pe baza traits-urilor iar cu MM se pot compune clasele, in contextul in care superclasele au fost create cu MI si LT. Dezavantajul mecanismelor traits este legat de imposibilitatea aplicarii compunerii pe clase de biblioteca deja existente care nu au fost proiectate in acest scop.

Tinand cont de stadiul actual al acestei relatii in literatura [LHQ94,Sak02], si avand in vedere faptul ca nu exista un model complet sau o implementare intr-un limbaj de programare, se pot aduce urmatoarele contributii:

(i) definirea unui model al acestui concept prin definirea unei semantici a acestei relatii de MI care sa functioneze in limabjul Eiffel,

(ii) proiectarea unui prototip care implementeaza acest concept, facilitand reutilizarea in practica a ierarhiilor de clase.

4. Elaborarea semanticii pentru relatia de mostenire inversa in Eiffel

In prima parte a proiectului ne propunem sa gasim o semantica pentru relatia de mostenire inversa intre clase. Pentru aceasta vom porni de la semantica limbajului Eiffel. Dupa modelul acesteia [Mey02], folosind exemple, gramatici si reguli se va crea semantica mostenirii inverse. De asemenea se vor prezenta si conditiile initiale de echivalenta si substituabilitate la nivel conceptual intre mostenirea inversa si cea directa sau clasica.

4.1. Crearea unei clase folosind mostenirea inversa

In aceast modul se va defini mostenirea inversa simpla si multipla. Se vor studia posibilitatile de factorizare ale membrilor subclaselor. De asemenea se va tine cont de tipul membrilor factorizati daca sunt atribute sau metode, daca sunt abstracti sau concreti, ce se intampla cu implementarea si in ce conditii poate fi ea factorizata. Un alt aspect important il reprezinta ce sa intampla cu relatia de subtipizare intre superclasa si subclasa in contextul MI. Aici am putea discuta eventual de necesitatea introducerii unei relatii duale de MI una conforma si alta non-conforma. Contributia in aceasta etapa se refera la definirea regulilor semantice pentru definirea relatiei in contextul limbajului Eiffel.

4.2. Mecanismele de adaptare a membrilor ex-mosteniti

In aceasta modul se va discuta despre mecanismele de redefinire, abstractizare si redenumire, care exista deja in contextul mostenirii clasice si cum acestea se vor modifica pentru mostenirea inversa. Adaptarile propuse pentru analiza sunt cele de scala, de ordine a parametrilor, de numar al parametrilor. De asemenea se vor analiza si adaptarile clasice, covariante ale limbajului referitor la signatura atributelor si a metodelor. Adaptarile pot aparea si la nivelul genericitatii. La interactiunea intre MI si genericitate pot aparea situatii care necesita stabilirea unor reguli semantice. De asemenea declaratiile ancorate specifice Eiffel vor suferi un impact la introducerea relatiei MI in limbaj. O atentie deosebita trebuie acordata mecanismului de verificare a corectitudinii care se constituie in paradigma de design pe baza de contract. Referirea se face direct catre asertiunile din subclase ce necesita a fi adaptate la o eventuala factorizare a membrului de care ele sunt atasate. Contributiile originale in aceasta etapa se pot referi la mecanismele care se pot dezvolta, unele pornind de la idei deja existente iar altele putand fi noi in totalitate.

4.3. Probleme legate de legarea dinamica si constrangerile impuse de aceasta

Se vor analiza potentialele problemele ce pot aparea in diferite situatii in ierarhiile de clase. Ar fi interesant de analizat situatiile de mostenire multipla prin relatii de MI a unor membri cu decendent comun sau fara descendent comun. O alta posibilitate de cercetare este importanta ordinii de creare a claselor in potentiale ierarhii de clase proiectate cu MI si mostenire clasica. Se vor studia constrangerile implicate de starea unor membri afectati de cuvinte cheie gen: precursor, assign, obsolete, once. In interactiunea MI cu agentii din Eiffel este un punct de analiza destul de interesant. O alta clasa de constrangeri este cea de la nivelul superclasei relatiei de MI.

Tot in acest modul se va propune o semantica pentru relatia like-type (LT) la integrarea in Eiffel, care ar avea potential in facilitarea reorganizarii ierarhiilor de clase. Potentialele contributii pot rezulta din solutiile care se ofera in solutionarea problemelor datorate interactionarii intre noua relatie si limbajul de programare.

4.4. Implementarea relatiei MI

Prin implementare se intelege creare unui set de reguli de translatare a relatie MI in relatii de MO echivalente avand in vedere echivalenta ierarhiilor obtinute. Aceste transformari se refera la cod fiind la baza scrierii prototipului.

4.5. Evaluarea abordarii

Pe baza unor studii de caz cu caracter general se va face o evaluare a abordarii de fata. Se vor cauta diverse situatii complexe in care solutia data de MI combinata eventual cu LT si MO este simpla si expresiva. Se va arata ca pe baza regulilor semantice si a celor de implementare, ierarhiile initiale se pot translata in ierarhii echivalente din punct de vedere al comportamentului.

4.6. Elaborarea prototipului

Dezvoltarea prototipului presupune mai multe faze. In prima faza se va proiecta modelul superclasei pentru mostenirea inversa apoi in a doua faza se va realiza un translator ce incapsuleaza transformarile de cod.

4.7. Modelul superclasei pentru mostenirea inversa

Pe baza gramaticii elaborata in partea de semantica, se va proiecta un nou model de clasa care sa cuprinda toate elementele noi de semantica introduse de MI. In model vor fi incluse elemente prin care sa se specifice atributele ce se doresc a fi factorizate, rutinele de transformare in cazul adaptarilor, informatii legate de unele decizii ce ar putea fi luate cand se trateaza legarea dinamica.

4.8. Proiectarea translatorului

Translatorul are ca scop eliminarea relatiei de MI dintr-un proiect Eiffel si inlocuirea ei cu mostenirea clasica. In acest fel se poate obtine un sistem executabil ce poate fi compilat cu un compilator de Eiffel, dupa care poate fi executat. Avand in vedere ca relatia de MI trebuie conceputa de asa natura incat sa poata fi substituita cu mostenire clasica, la nivel conceptual, rezulta ca sansele ca la transformari sa apara dificultati sunt minimizate.

Mecanismul de factorizare va presupune mutarea membrilor factorizati in superclasa. Evident ca mecanismele de adaptare vor presupune adaugarea de cod aditional de conversie in clasele afectate de MI. Regulile din modulul ce se ocupa cu legarea dinamica ar putea conduce la transformari care sa schimbe modificatorii unor membri.

5. Concluzii

Mecanismul de reutilizare de clase bazat pe mostenirea inversa evident nu rezolva toate problemele de reutilizare ale tehnlogiei orientate obiect. Pentru o reutilizare mai eficienta se pot folosi tehnici de tipul programarii orientate pe aspecte. Evident complexitatea acestora este mult peste cea conceptului de mostenire inversa. Din momentul in care se cunoaste conceptul de mostenire clasic este mult mai usor de a inversa logica acestuia si a intelege mostenirea inversa. Totodata mostenirea inversa ne ajuta sa intelegem mai bine conceptul de mostenire clasica.

Bibliografie

[Bla02] Andrew P. Black. A Use for Inheritance. In Proceedings of the Inheritance Workshop at ECOOP 2002, Malaga, Spain: Information Technology Research Institute, University of Jyvaskyla, Finland.

[Chi04a] Ciprian-Bogdan Chirila, Pierre Crescenzo, Philippe Lahire, Dan Pescaru, Emanuel Tundrea. Factoring Mechanism of Reverse Inheritance, International Conference on Technical Informatics CONTI 2004, Periodica Politechnica, Transactions on Automatic Control and Computer Science Vol.49 (63), ISSN 1224-600X, pp. 131-136, Timisoara, Romania, May 2004.

[Chi04b] Ciprian-Bogdan Chirila, Pierre Crescenzo, Philippe Lahire. A Reverse Inheritance Relationship Dedicated to Reengineering: The Point of View of Feature Factorization, MASPEGHI Workshop at ECOOP 2004, MechAnisms for SPEcialization, Generalization and inHerItance Oslo, June 15, 2004.

[CMR02] Yania Crespo and Jos Manuel Marques and Juan Jos Rodryguez. On the Translation of Multiple Inheritance Hierarchies into Single Inheritance Hierarchies, In European Conference on Object-Oriented Programming, 2002.

[Cut06] Tom van Cutsem. Eiffel and C++ a comparison between Object Oriented Languages, (unpublished), http://prog.vub.ac.be/~tvcutsem/publications/oo_comparison.pdf

[Dao02] Michel Dao and Marianne Huchard, Therese Libourel and Cyril Roume. Evaluating and Optimizing Factorization in Inheritance Hierarchies, Proceedings of the Inheritance Workshop at ECOOP 2002, Malaga, Spain, June, 2002.

[Fro02] Peter H. Frohlich, Inheritance Decomposed, In Proceedings of the Inheritance Workshop at ECOOP 2002, Malaga, Spain, June, 2002.

[LHQ94] Ted Lawson, Christine Hollinshead, and Munib Qutaishat. The potential for reverse type inheritance in Eiffel. In Technology of Object-Oriented Languages and Systems (TOOLS'94), 1994.

[Mey02] Bertrand Meyer. Eiffel: The language. http://www.inf.ethz.ch/~meyer/, September 2002.

[Mey97] Bertrand Meyer. Object-Oriented Software Construction 2nd ed., Prentice Hall, 1997.

[Ped89] C. H. Pedersen. Extending ordinary inheritance schemes to include generalization. In Conference proceedings on Object-oriented programming systems, languages and applications, pages 407-417. ACM Press, 1989.

[Sak02] Markku Sakkinen. Exheritance - Class generalization revived. In Proceedings of the Inheritance Workshop at ECOOP 2002, Malaga, Spain, June 2002

[Sch02] Nathanael Scharli, Stephane Ducasse, Oscar Nierstrasz. Classes = Traits + States + Glue (Beyond mixins and multiple inheritance), In Proceedings of the International Workshop on Inheritance, Malaga, Spain, June, 2002.

[Sch03] Nathanael Scharli, Stephane Ducasse, Oscar Nierstrasz and Andrew Black. Traits: Composable Units of Behavior, In Proceedings of the Inheritance Workshop at ECOOP 2003, Darmstadt, Germany, July, 2003.

[Sma02] Yannis Smaragdakis and Don Batory. Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs, In ACM Trans. Softw. Eng. Methodol., vol. 11, no 2, issn 1049-331X, pp. 215-255, http://doi.acm.org/10.1145/505145.505148, ACM Press, New York, NY, USA, 2002.

[TUI05] Tetsuo Tamai, Naoyasu Ubayashi and Ryoichi Ichiyama, An Adaptive Object Model with Dynamic Role Binding, ICSE '05: Proceedings of the 27th International Conference on Software Engineering, issn 1-59593-963-2, pp. 166-175, St. Louis, Missouri, USA, ACM Press, New York, NY, USA, May, 2005.