Clasificări şi formalizări pentru şabloane de proiectare

drd. Dumitru CIORBĂ, *Universitatea Tehnică a Moldovei, diciorba@yahoo.com*
prof. dr. Victor BEŞLIU, *Universitatea Tehnică a Moldovei, vbesliu@yahoo.com*

***Abstract* — Utilizarea tehnicilor formale sau semi-formale în procesele de proiectare, în cel mai apropiat timp poate duce la apariţia unor instrumente automatizate de *design* şi *refactoring* a sistemelor obiect-orientate. Iar prin cercetările noastre, aplicând metode *pattetn-driven design* şi *genetic programming,* noi ne dorim să contribuim la accelerarea apariţiei unor astfel de sisteme, care vor cere implicarea minimă a proiectanţilor. Clasificările şi formalizările şabloanelor de proiectare ne vor facilita înţelegerea şi utilizarea lor. Astfel în acest articol ne propunem să trecem în revistă cele mai actuale clasificări şi formalizări, precum şi să prezentăm un formalism ce na va permite facil şi intuitiv să reprezentăm formal şabloanele, dar şi unificarea lor.**

***Cuvinte cheie* — analiză şi proiectare obiect-orientată, clasificarea şabloanelor de proiectare, COMET, formalizarea şabloanelor de proiectare**

I. INTRODUCERE

Tehnicile programării genetice utilizate în cadrul cercetărilor implică găsirea unei modalităţi de codificare (reprezentare) a şabloanelor de proiectare. Astfel se justifică încercarea noastră de a înţelege şabloanele şi de a găsi modalitatea optimală pentru formalizarea lor.

În cadrul cercetărilor efectuate, după o analiză a clasificărilor şi formalizărilor implicate în studiu, noi propunem o „regrupare" a clasificărilor şi a formalismelor în scopul înţelegerii lor mai bune. La fel, în acest articol ne propunem să prezentăm notaţia pentru formalizarea structurilor OO, şi vom încerca să argumentăm necesitatea propriului formalism pentru reprezentarea şabloanelor de proiectare.

II. CLASIFICĂRI ALE ŞABLOANELOR DE PROIECTARE

Ce face ca şabloanele să fie atât de diferite? Care este relaţia dintre ele? Ce factori caracterizează fiecare şablon? Căutarea răspunsurilor ne va duce într-un mod sau altul la necesitatea clasificării şabloanelor. În primul rând, clasificarea ne va oferi o înţelegere mai bună a entităţilor analizate. În cel de-al doilea rând ne va permite o organizare mai bună, care implicit facilitează înţelegerea lor.

Savantul rus Voevodin V. V. în cartea sa *Параллельные вычисления* [1] menţiona importanţa clasificării invocând o analogie cu legea periodicităţii apărută în 1869, care a permis ordonarea elementelor chimice şi a indicat spre existenţa unor „pete albe".

Astfel, poate mai mult intuitiv, dar putem susţine că o „clasificare" reuşită ne va oferi o caracterizare suficientă a unui şablon, doar după apartenenţa la o grupă sau alta, mai mult decât atât, ne poate indica (prin „petele albe") spre anumite şabloane ce pot fi obţinute nu numai experimental, ci şi teoretic. Aparent este ceva utopic, dar legea periodicităţii în 1869 părea la fel.

Nu ne propunem o analiză exhaustivă a clasificărilor existente, dar vom menţiona pe cele care ne-au atras atenţia prin simplitatea şi originalitatea lor în cadrul cercetării. Iar aici am menţiona că unele clasificări nu sunt chiar explicit prezentate de autori.

Diversele interpretări ale cercetătorilor şi modul de aplicare deseori ale aceluiaşi criteriu şi determină apariţia multor clasificări. Multitudinea celor din urmă nicidecum nu complică analiza şabloanelor, ba dimpotrivă, ele ne permit să analizăm şabloanele din diverse perspective:

Clasificări bazate pe scop/intenţie

- Clasificarea primară GoF de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides [2];
- Concretizarea clasificării GoF de Alan Shalloway şi James R. Trott [3];
- Clasificarea şabloanelor GoF de Steven Metsker [4];
- Clasificarea şabloanelor GoF de Nija Shi şi Ronald A. Olsson [5];

Clasificări bazate pe analiza structurii claselor

- Clasificarea şabloanelor GoF de Wolfgang Pree utilizind meta-şabloanele [6] (aplicate şabloanelor GoF în *The UML Profile for Framework Architectures* [7]);
- Clasificarea şabloanelor GoF după similitudini structurale de Vince Huston [8];

Clasificări bazate pe relaţii între şabloane

- Clasificarea şabloanelor GoF de Walter Zimmer bazată de relaţii primare între şabloane [9].

Astăzi există şi încercări multiple, mai mult sau mai puţin reuşite, de formalizare a şabloanelor de proiectare, care ne permit descrierea exactă şi "matematică" a şablonului.

III. FORMALIZĂRI ALE ŞABLOANELOR DE PROIECTARE

Specificarea formală a şabloanelor de proiectare, prin descrieri precise, ne permite să găsim relaţiile dintre şabloane, să validăm aplicaţiile conform specificaţiilor determinate de şabloane, să elaborăm instrumentar ce ar facilita utilizarea lor, etc.

Şirul formalizărilor este în continuă creştere. Doar lucrările [10] şi [11] ne descriu împreună în jur de 20 de formalizări diferite. Şi toate încearcă să acopere problemele legate de automatizarea proceselor de proiectare, pe care limbajul de modelare UML de sine-stătător nu le poate

acoperi. Printre aceste majore deficienţe ale limbajului UML se poate enumăra imposibilitatea de operare pe mulţimi de nivel înalt (cum ar fi mulţimi de mulţimi), numărul limitat de relaţii între clasificatori, precum imposibilitatea definirii explicite a relaţiilor între metodele claselor/obiectelor.

Natura formalismelor este diferită, iar uneori autorii recurg la forme complexe de reprezentare a şabloanelor:

Formalisme matematico-logice:

- BPSL (Balanced Pattern Specification Language) – Toufik Taibi, David Chek Ling Ngo [12]
- Modelul formal al şabloanelor bazat pe RSL (RAISE specification language) - Andrés Flores, Alejandra Cechich, Gabriela Aranda [10]
- Reprezentarea şabloanelor de proiectare utilizând logica de ordinul întâi (instrumentarul utilizează PROLOG) – J. Dong, P.S.C. Alencar şi D.D. Cowan [13]
- Sistemul formal pentru detectarea şabloanelor de proiectare (SPQR - *system for pattern query and recognition*) utilizînd ρ-calculus şi semantici formale ce descriu relaţiile dintre entităţile programabile – Jason Smith şi David Stotts [14]
- Limbajul formal LePUS (LanguagE for Pattern Uniform Specification), specificaţiile căruia (formule text sau diagrame vizuale) determină relaţiile dinte *participanţi* în termenii calculul predicativ – Epameinondas Gasparis [10]

Formalisme bazate pe meta-modele UML:

- DPML (Design pattern modeling language) – David Mapelsden, John Hosking, John Grundy [15]
- RBML (Role-Based Metamodeling Language) – D. Kim, R. France, S. Ghosh, E. Song [16]
- Constraint Diagrams – Anthony Lauder şi Stuart Kent [17]

Formalisme bazate pe semantici mixte (formale şi informale)

- Ontologia ODOL OWL (Object Design Ontology Layer) - Jens Dietrich şi Chris Elgar [10]
- Notaţia URN (User Requirements Notation) asociată de GRL (Goal-Oriented Requirement Language) şi de notaţia UCM (Use Case Map)– Gunter Mussbacher, Daniel Amyot şi Michael Weiss [10]

Formalisme convenţionale – extensii de limbaje:

- Tehnica gramaticilor extinse – G. Hedin [11]
- Tehnica „extinderii” compilatorului: PEC (Pattern Enforcing Compiler) – H.C. Lovatt, A.M. Sloane şi D.R Verity[10]

IV. NOTAŢIA COMET (CLASS-OBJECT-METHOD)

În urma analizei formalismelor prezentate mai sus, noi am conchis că ele nu sunt cele mai bune soluţii pentru reprezentarea şabloanelor de proiectare în maniera în care ne dorim să exprimăm formal problema de unificare a şabloanelor de proiectare. Astfel aş enumăra cumulativ următoarele inconveniente:

1. Abstractizarea prea mare: nu toţi programatorii vor dori să înveţe acest formalism (în deosebi dacă nu va conţine structuri OO explicite);
2. Detalizarea prea mare: nu va fi posibil de a oferi soluţii *cross*-limbaj (de exemplu PEC este o metodă exclusivă JAVA);
3. Metode imperfecte de lucru cu mulţimile de clase/obiecte/metode (sau chiar lipsa lor): aici ar fi ca şi excepţie grupul de formalisme matematico-logice.
4. Nu toate metodele definesc explicit natura relaţiei dintre entităţi, care poate fi generală între clase, specifică între instanţe, determinată de invocări de metode, etc.
5. Notaţii complexe: sintaxa notaţiilor trebuie să fie clară oricărui dezvoltator de programe OO, căci formalizarea se doreşte a fi efectuată asupra unor structuri OO;
6. Notaţiile grafice sunt uneori complet străine limbajului UML, deci utilizarea metodelor formale pe scară industrială este îngreunată.

Aceste şi alte inconveniente justifică elaborarea unei noi notaţii şi implicit unui nou formalism care ar permite exprimarea a unor structuri OO simple şi mai puţin complexe, cum sunt şabloanele de proiectare.

Nu ne propunem aici să enumerăm toate principiile notaţiei COMET [18] elaborate în cadrul cercetărilor noastre, dar esenţa constă în respectarea relaţiilor OO asistate de notaţii specifice, care intuitiv sunt clare oricărui dezvoltator de programe OO. Astfel notaţia se bazează pe următoarele patru principale elemente:

1. Acquaintance
   - astfel sunt exprimate relaţii de tipul *Association, Aggregation, Composite*
   - `C1@C2`
2. Hierarchy
   - astfel sunt exprimate relaţiile de moştenire sau realizare (de interfeţe)
   - H_2`(S1,S2)`
3. Alias
   - ne permite lucrul cu mulţimile de entităţi, utilizat de fapt pentru a „marca” şi selecta entităţile din structuri;
   - $\alpha, \beta, \ldots$
4. Predicates
   - ne vor permite specificarea relaţiilor dintre entităţi în cadrul unor paranteze pătrate după selectorul elementului (entitate, relaţie, alias).

Pentru ilustrarea notaţiei fie structura obiect-orientată din fig. 1: unde avem o relaţie de moştenire dintre *Class12* şi *NewClass12*, conform căreia două metode sunt redefinite în clasa descendentă, precum şi o relaţie de asociere direcţionată, natura căreia nu este definită, dintre clasa *NewClass12* şi clasa *Client*.

Astfel, expresia ce ar reprezenta structura descrisă ar fi următoarea:

$$\alpha[M1,M2](H(S)),\ C_{Client}@\ \alpha S \qquad (1)$$

Prin α vom s-a marcat ierarhia (H) structurii noastre, unde avem doar o singură clasă descendentă (*S*) şi două metode redefinite în ea (M1 – *Operation1*, M2 – *Operation2*), iar α_S nu va fi decît clasa *NewClass12*.

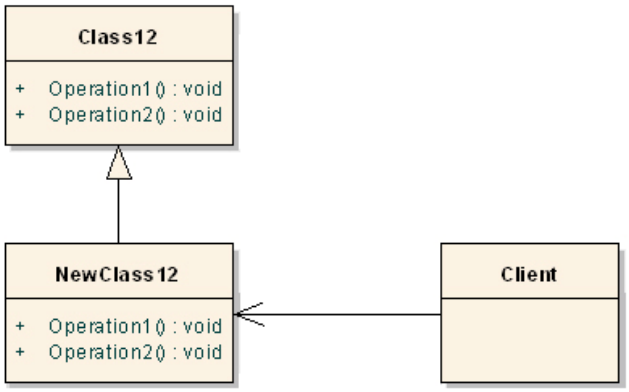

Fig. 1 – Diagrama de clasă a unei structuri OO simple

V. FORMULE COMET PENTRU EXPRIMAREA UNIFICĂRII DE ŞABLOANE

În acest punct final am putea menţiona că expresiile COMET sunt succinte astfel încît formularea problemei de unificare a şabloanelor de proiectare *Strategy* (formula 1) şi *Template Method* (formula 2) [2] poate fi redusă la formule de tipul formulei 3.

$$\beta(C[M\{\alpha M\}]@\alpha[M](H_2(S1,S2))) \quad (1)$$

$$\gamma(\alpha[M2,\ M3](H1[M1\{M2,M3\}](S1))) \quad (2)$$

$$\beta + \gamma \rightarrow \ldots \quad (3)$$

REFERINŢE

[1] Voevodin V. V., Voevodin V. Vl. *Параллельные вычисления,* BHV-Peterburg, 2004

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995

[3] Alan Shalloway, James R. Trott, *Design Patterns Explained*, 2000

[4] Steven Metsker, Design patterns. Java WorkBook, 2002

[5] Nija Shi, Ronald A. Olsson, Reverse Engineering of Design Patterns from Java Source Code, Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006

[6] Wolfgang Pree, *Meta Patterns - A Means for Capturing the Essentials of Reusable Object-Oriented Design*, 1994

[7] Marcus Fontoura, Wolfgang Pree, Bernhard Rumpe *The UML Profile for Framework Architectures*, 2001

[8] Vince Huston, Design patterns, http://www.vincehuston.org/dp/, 2006

[9] Walter Zimmer, *Relationships Between Design Patterns, Pattern Languages of Program Design*, 1995

[10] Toufik Taibi (red.), *Design Patterns Formalization Techniques*, IGI Publishing, London, 2007

[11] Aline Lúcia Baroni, Yann-Gaël Guéhéneuc, Hervé Albin-Amiot, *Design Patterns Formalization (Research Report)*, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Nantes, 2003

[12] Toufik Taibi, David Chek Ling Ngo, *Formal Specification of Design Patterns - A Balanced Approach*, Journal Of Object Technology, Vol. 2, No. 4, 2003

[13] J. Dong, P.S.C. Alencar, D.D. Cowan, *Ensuring structure and behavior correctness in design composition*, In Proceedings of the 7th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (pp. 279-287), 2000

[14] Jason McC. Smith, David Stotts, *SPQR: Flexible Automated Design Pattern Extraction From Source Code*, 18th IEEE International Conference on Automated Software Engineering (ASE'03), 2003

[15] David Mapelsden, John Hosking, John Grundy, *Design Pattern Modeling and Instantiation using DPML*, Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications, Sydney, 2002

[16] D. Kim, R. France, S. Ghosh, E. Song, *A UML-Based Metamodeling Language to Specify Design Patterns,* Proceedings of the Workshop on Software Model Engineering (WiSME) at UML, San Francisco, 2003

[17] Anthony Lauder, Stuart Kent, *Precise Visual Specification of Design Patterns*, Proceedings of ECOOP'98, 1998

[18] Dumitru Ciorbă, *A method based on GP for pattern-driven specification and design (doctoral thesis presentation)*, Doctoral Intensive Summer School on Evolutionary Computing in Optimization and Data Mining, Iaşi, 2009