ŞABLOANE DE PROIECTARE GOF FORME SPECIALIZATE A PRINCIPIILOR DE PROIECTARE

Autori: Sergiu STAVENSCHI, Vadim ANDRONACHI
Conducător ştiinţific: prof. univ. Dumitru CIORBĂ

Universitatea Tehnică a Moldovei
Email: stavenschi@gmail.com, andronachi.vadim@gmail.com, diciorba@yahoo.com

Abstract: *Proiectarea unui sistem software flexibil, reutilizabil şi elegant, implică un efort sporit şi mai mult timp la introducerea de nivele abstracte crescând considerabil complexitatea codului. La crearea ierarhiilor, structurilor flexibile bine echilibrate ne îndrumă principiile obiect orientate, care posedă un nivel înalt de abstractizare, comparativ şabloanelor de proiectare (gof). Principiile de proiectare sunt reflectate de şabloanele de proiectare, astfel aceste principii determină construcţiile primare a şabloanelor. Această lucrare urmăreşte identificarea formelor abstracte (principiilor) a şabloanelor GOF pentru clasificarea şabloanelor conform principiilor de proiectare facilitând înţelegerea lor.*

Cuvinte cheie: *principii, şabloane de proiectare, proiectare obiect orientată, responsabilităţi.*

1 Descrierea principiilor de proiectare obiect orientat

1.1 Single Responsibility Principle (SRP)

Niciodată nu trebuie să fie mai mult de un motiv pentru schimbarea unei clase [1].
Principiu unicei responsabilităţi ţine cont de următoarele afirmaţii:

a) clasă trebuie să deţină o singură responsabilitate.
b) Trebuie să aibă un singur motiv pentru schimbare.
c) Un obiect trebuie să folosească delegarea pentru a realiza operaţii ce nu ţin de responsabilitatea sa.
d) responsabilitate trebuie localizată într-un singur obiect şi nu duplicate în mai multe obiecte.

1.2 Open Close Principle (OCP)

Entităţile software (clase, module, funcţii) trebuie să fie deschise pentru extensie, dar închise pentru modificare [2]. Realizarea unui design flexibil presupune un timp suplimentar şi efort cheltuit pentru abstractizare crescând complexitatea codului.

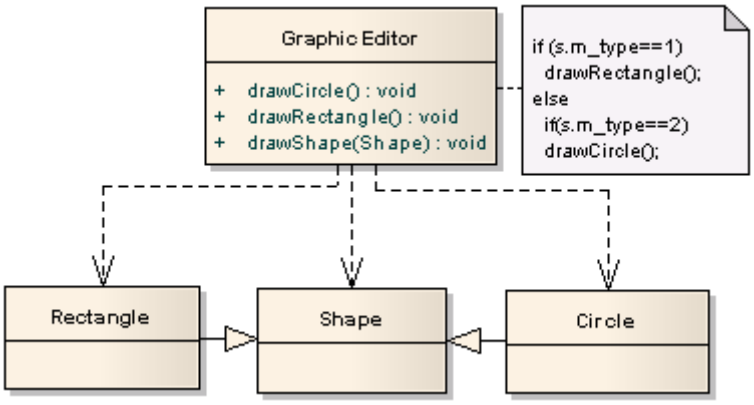

Figura 1 - Încălcarea Principiului Open Close [6]

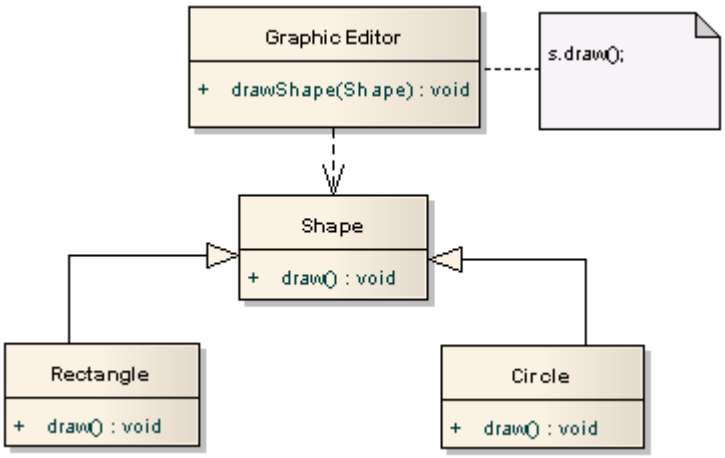

Figura 2 - Respectarea Principiului Open Close [6]

1.3 Liskov Subtitution Principle (LSP)

Principiu afirmă că dacă un modul program foloseşte o clasă de bază atunci referința la clasa de bază poate fi înlocuită cu o clasă derivată fără a afecta funcționalitatea modulului program.
Funcții care utilizează pointeri sau referințe la o clasă de bază trebuie să poată să utilizeze obiecte de la clase derivate fără ca să le cunoască [3].
Când este încălcat acest principiu, funcțiile care operează pe pointeri sau referințe la clasele de bază va trebui să verifice tipul obiectului actual pentru a vă asigura că acestea pot funcționa corect pe ea şi trebuie să ştie despre toate derivatele clasei de bază.
O astfel de funcție încalcă principiu Open Close deoarece aceasta trebuie să fie modificată ori de câte ori o nouă derivată e creată din clasa de bază.
Acest principiu poate fi interpretat din două aspecte. Primul aspect cuprinde crearea ierarhiilor obiect orientate corecte, principiul în acest caz ghidează procesul de creare a ierarhiilor corecte din punct de vedere a compatibilității dintre elementele sub-ordonate, care au la bază una şi aceeaşi clasă (clasă abstractă sau interfață). Al doilea aspect cuprinde crearea modelele corecte, care utilizează ierarhii, principiul în acest caz ghidează procesul de utilizare a ierarhiilor astfel încât modelul trebuie să fie compatibil cu orişice tip sub-ordonat, fără a cunoaşte tipul concret.

1.4 Interface Segregation Principle (ISP)

Clienții nu ar trebui să fie obligați să depindă de interfețe care nu le utilizează [4].
Sunt de preferat mai multe interfețe specifice client decât o interfață generală.

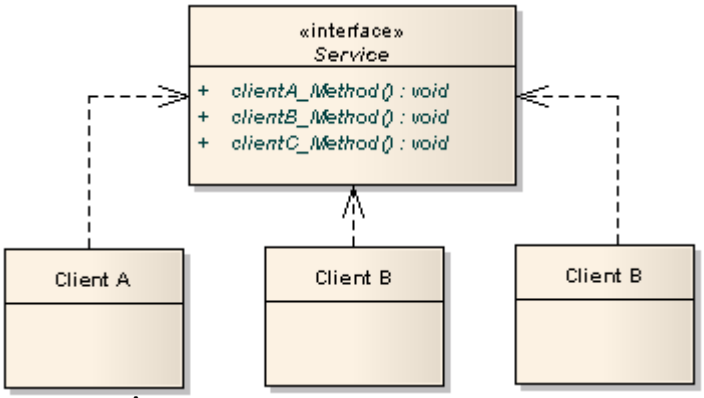

Figura 3 - Încălcarea Principiului Interface Segregation [8]

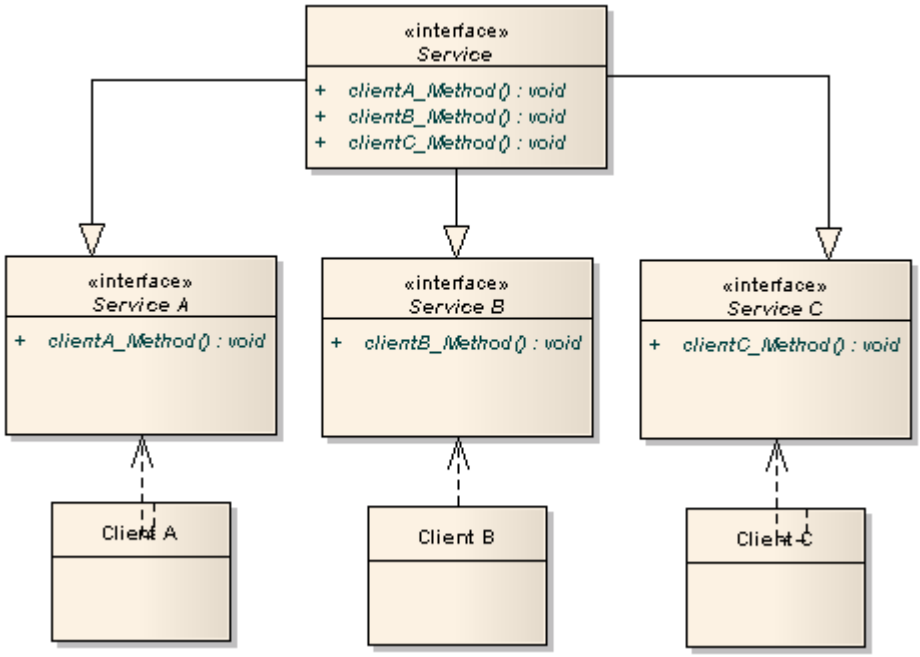

Figura 4 - Respectarea Principiului Interface Segregation [8]

1.5 Dependency Inversion Principle (DIP)

Modulele de nivel înalt nu trebuie să depindă de modulele de nivel jos. Ambele trebuie să depindă de abstracții [5]. Abstracțiile nu trebuie să depindă de detalii. Detaliile trebuie să depindă de abstracții [5].
Dependența de lucruri abstracte şi nu de lucruri concrete. Scopul a acestui principiu este să decupleze componentele de un nivel înalt de componentele de nivel jos aşa că reutilizarea cu diferite componente de nivel jos implementate să devină posibil.

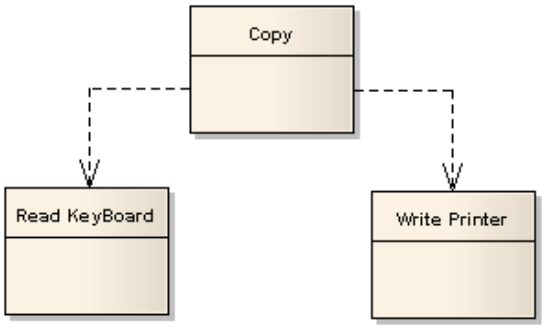

Figura 5 - Încalcarea Principiului Dependency Inversion [5]

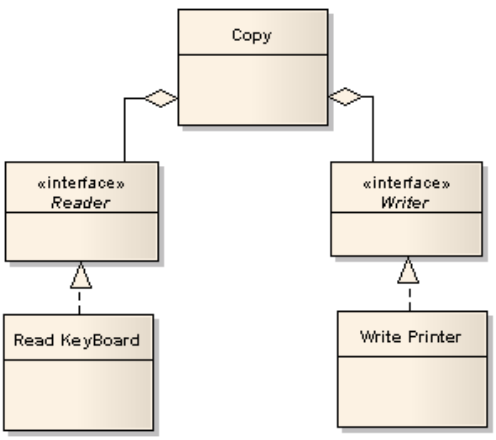

Figura 6 - Respectarea Principiului Dependency Inversion [5]

2 Clasificarea şabloanelor de proiectare conform principiilor de proiectare

Clasificarea şabloanelor GOF conform principiilor de proiectare catalogate în tabelul 1 oferă o mai bună înțelegere a entităților analizate. O clasificare reuşită ne va oferi o caracterizare suficientă a unui şablon doar după apartenența la o grupă sau alta. Clasificarea propusă evidențiază construcțiile primare (principiile) a şabloanelor de proiectare. În unele şabloane sunt prezente mai multe construcții primare ca exemplu este AbstractFactory însemnând că un pattern poate să conțină mai multe forme abstracte.

Tabelul 1 – Şabloanele de proiectare forme specializate a principiilor obiect orientat

SRP	OCP	LSP	DIP
Singleton	Adapter (Intentie)	Abstract Factory	Builder
Façade	Decorator(Intentie)	Factory Method	Prototype
Memento		Adapter	Bridge
Mediator(Intentie)		Proxy	Composite
		Iterator	Decorator
		Chain of Responsibility	Flyweight
		Composite(Intentie)	Command
			Interpreter
			Mediator
			Observer
			State
			Strategy
			Template Method
			Visitor

3 Exemplificarea raționamentelor de clasificare

3.1 Principiul inversării dependențelor (DIP)

Când principiu inversării dependențelor este aplicat aceasta înseamnă că clasele de nivel înalt nu lucrează direct cu clasele de nivel jos, ele folosesc interfețele ca un strat abstract.
Şabloanele specializate a acestui principiu sunt: Factory Method [7], Builder, Prototype [7], Bridge, Flyweight, Command, Interpreter, Iterator, Mediator, Observer, State, Strategy, Template Method [7], Visitor.

3.1.1 Builder

Entitățile principiului dip implicate în şablonul Builder sunt Director fiind ca stratul superior al structurii, nivelul abstract Builder şi ConcreteBuilder stratul de nivel jos, legătura directă dintre Director şi ConcreteBuilder este ruptă de interfața Builder.

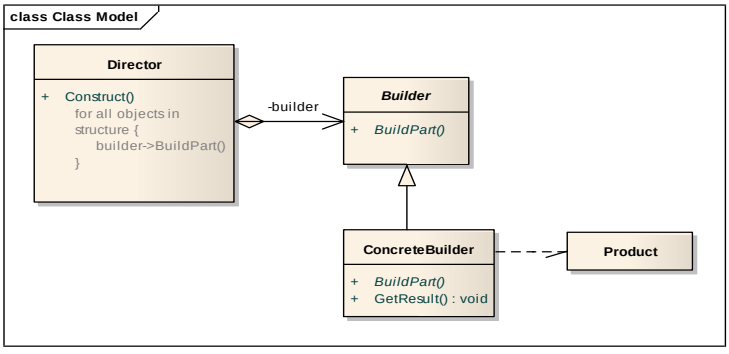

Figura 7 – Structura şablonului Builder

3.1.2 Prototype

Prezența principiului inversării dependențelor este vizibil între entitatea *Client* şi entitățile *ConcretePrototype* separat de un strat abstract numit *Prototype*, în aşa mod este făcut decuplarea dintre clasa de nivel înalt (*Client*) şi clasele de nivel jos (*ConcretePrototype*) prin intermediul stratului abstract (*Prototype*), aceasta specializare este arată în figura 8.

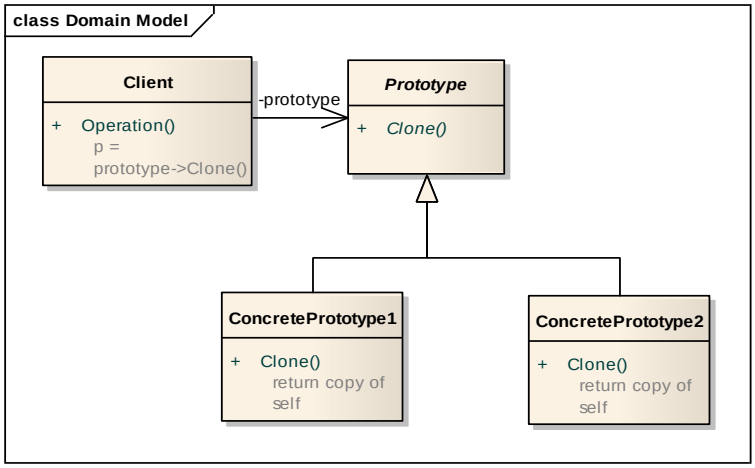

Figura 8 – Structura şablonului Prototype

3.1.3 Bridge

Bridge din figura 9 este specializarea principiului inversării dependențelor format dintre clasa de nivel înalt *Abstraction* şi clasele de nivel jos *ConcreteImplementor* separat de clasa abstractă *Implementor*, in aşa fel este creată o dependență tranzitivă clasa *Abstraction* dependentă de clasele *ConcreteImplementor* fiind posibil reutilizarea clasei *Abstraction* cu diferite clase derivate din *Implementor*.

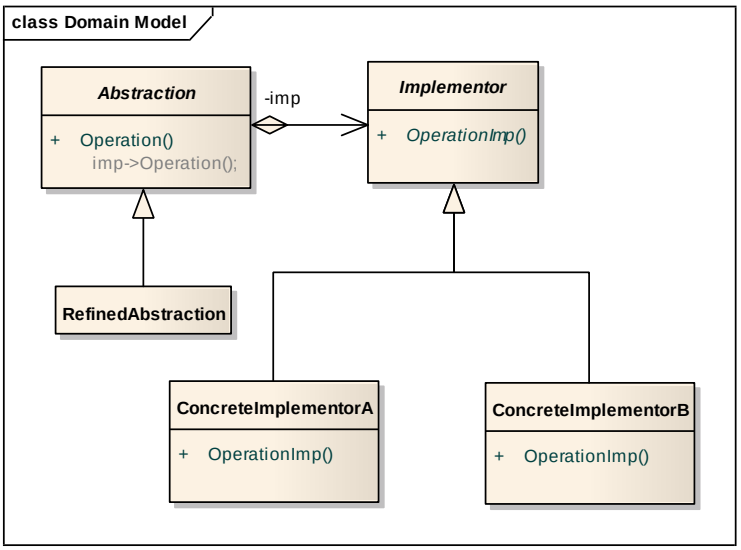

Figura 9 - Structura şablonului Bridge

3.2 Principiul Open Close

Şabloanele specializate a acestui principiu sunt: Adapter(Intenție), Decorator(Intenție).

3.2.1 Decorator

Modelul şablonului *Decorator* are intenția de a ataşa responsabilităţi obiectelor în mod dinamic promovînd principiul Open Close, cu referință la [6].

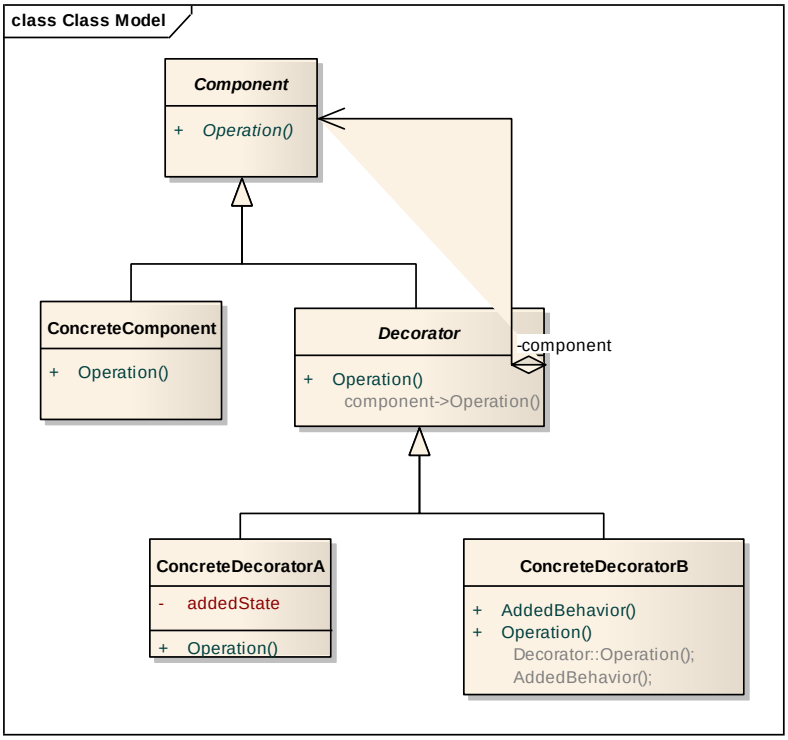

Figura 10 – Structura şablonului Decorator

3.2.2 Adapter

Adapter converteşte interfaţa unei clase la interfaţa care o aşteaptă clienţii acesteia, în acest fel modelul nu violează principiul Open Close el vizează extensia entităţii *Adaptee* fără a modifica comportamentul său.

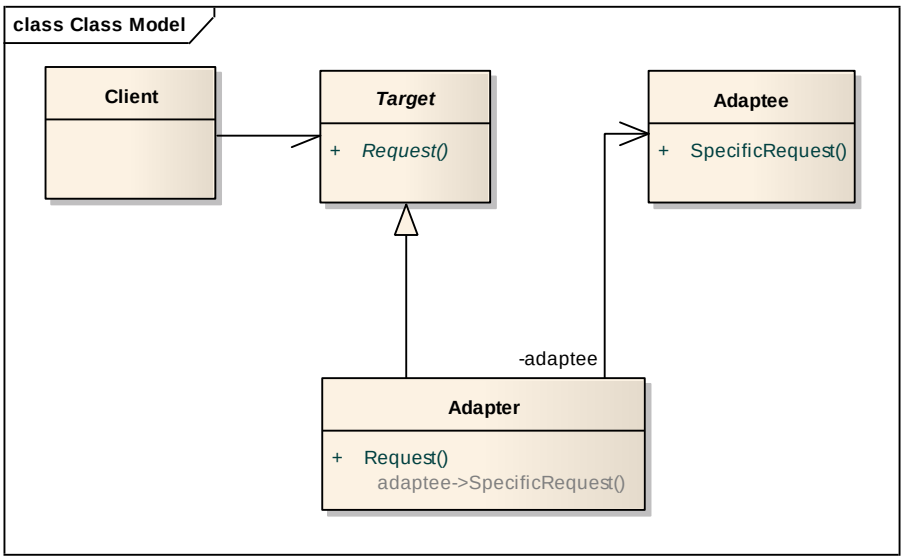

Figura 11 - Structura şablonului Adapter

3.3 Principiul unicei responsabilităţi

Principiu Single Responsibility fiind mai mult o convenție de formare a unor structuri bine echilibrate, fiecare obiect având o responsabilitate evidențiind în mod clar şablonul High Cohesion descris de Craig Larman.
Şabloanele specializate a acestui principiu sunt: Singleton, Façade, Memento şi Mediator(intenție).

3.3.1 Singleton

Singleton este bazat pe principiu singurei responsabilități deoarece şablonul spune că are responsabilitatea de a garanta existența unei singure instanțe a unei clase.

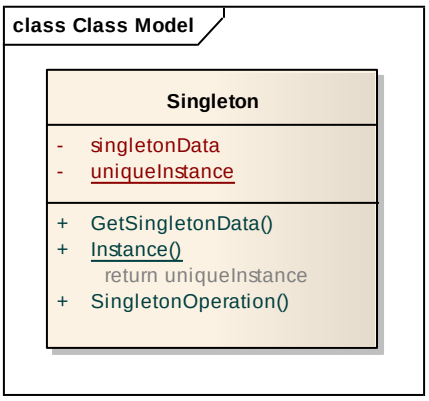

Figura 12 - Structura şablonului Singleton

3.3.2 Memento

Memento captează şi extern-alezează starea internă a unui obiect fără a afecta principiul încapsulării, având unica responsabilitate de a încapsula starea obiectului în alt obiect pentru o refacere ulterioară.

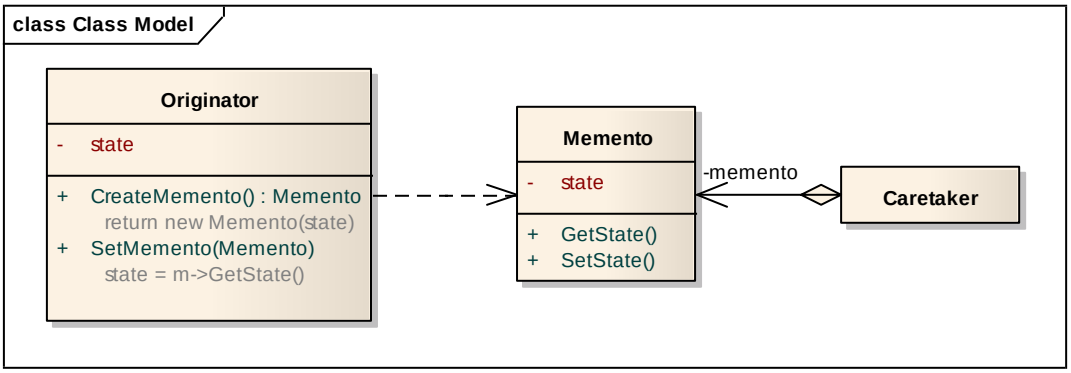

Figura 13 - Structura şablonului Memento

3.4 Principiul substituției Liskov

O clasă de bază trebuie să poată fi substituită cu orice clase derivate din ea.
Şabloanele specializate a acestui principiu sunt: Factory Method, Adapter, Proxy, Iterator, Chain of Responsibility, Abstract Factory, Composite(Intentie).

Abstract Factory

AbstractFactory prezentat în figura 14 furnizează o interfață pentru crearea familiilor de obiecte înrudite sau dependente, fără specificarea claselor lor concrete. Clasele *ConcreteFactory* şi *ConcreteProduct* extind clasele abstracte *AbstractFactory* şi *Product* fără a schimba funcționalitățile claselor de bază obținîndu-se clase complet substituibile pentru clasele de bază acest fapt demonstrează ca principiul Substituției Liskov este formă abstractizată a şablonului *AbstractFactory*.

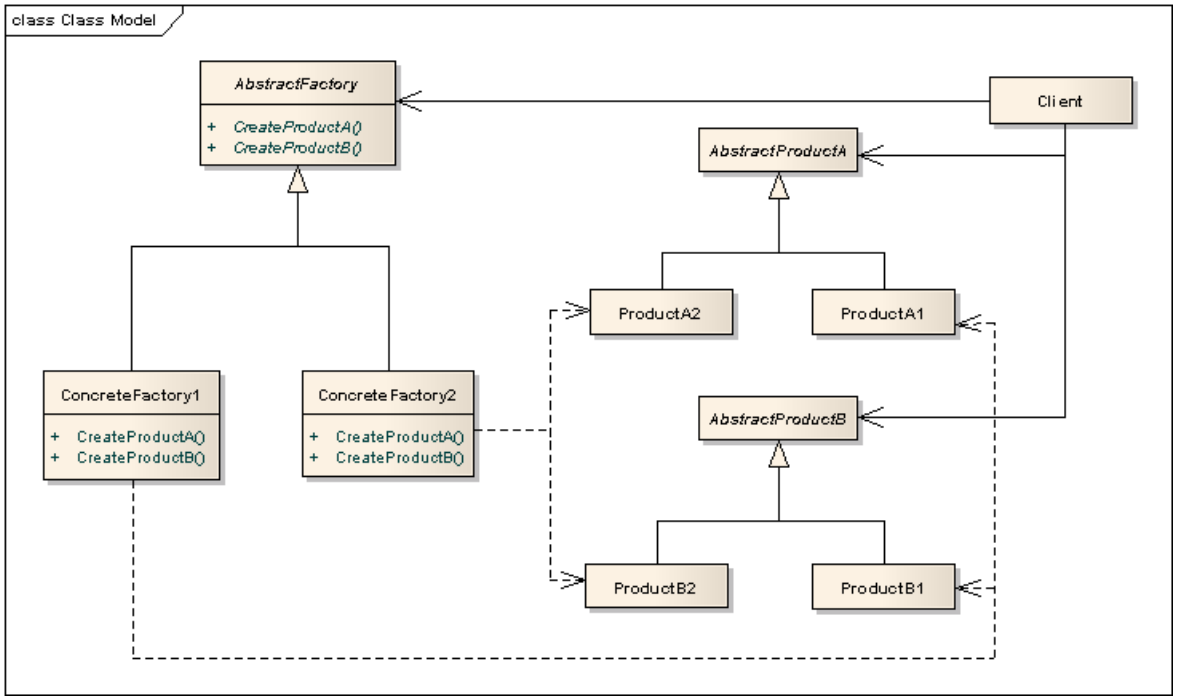

Figura 14 - Structura şablonului AbstractFactory

3.5. Principiul segregării interfețelor

Forme specializate a acestui principiu nu sânt, dar acest principiu este aparent la aplicarea şablonului Adapter atunci când segregăm interfețe încărcate [10].

4 Corelația dintre raționamentele lui Zimmer şi clasificarea

Formele specializate a principiilor de proiectare obiect orientată este posibil de demonstrat cu ajutorul lucrării lui Walter Zimmer numită “Relationships between Design Patterns”.

În lucrarea sa Zimmer defineşte 3 categorii de relații dintre şabloane [9]:

a) X utilizează Y în soluțiile sale;
b) X poate fi combinat cu Y;
c) X este similar cu Y.

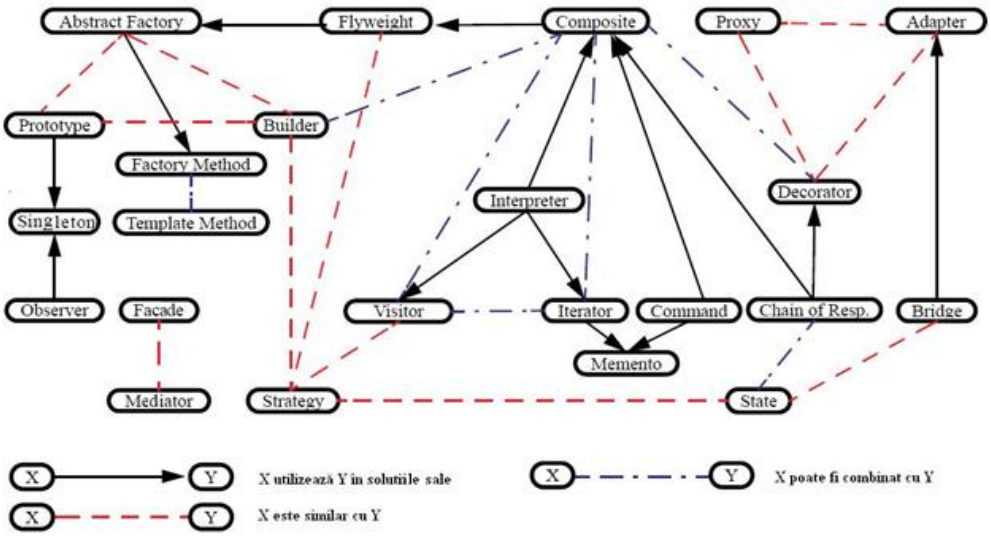

Figura 15 - Clasificarea relațiilor a şabloanelor de proiectare [9]

Zimmer determină că şablonul Adapter şi Decorator se află într-o relație „modelul X este similar cu modelul Y" prezentată în figura 13, acest fapt demonstrează că toate aceste 3 patterne sunt forme specializate a principiului Open Close.
Cunoscînd că structura lui Builder reflectă principiu inversării dependențelor putem să concluzionăm că şabloanele Prototype, Builder, Strategy, Flyweight, State, Bridge sunt forme specializate a acestui principiu deoarece Builder se află într-o relație de tipul "şablonul X este similar cu şablonul Y" cu toate şabloanele expuse mai sus.

5 Concluzie:

Această lucrare a făcut posibil determinarea formelor abstracte (principiilor) a şabloanelor de proiectare GOF argumentând prin aparențele construcțiilor primare a principiilor în şabloane şi pentru unele şabloane demonstrate şi cu ajutorul relațiilor Zimmer. La clasificarea şabloanelor GOF conform principiilor de proiectare a fost stabilite 3 forme specializate a principiului unicei responsabilități şi 1 după intenție, 2 după intenție a principiului deschis închis, 6 a principiului substituției Liskov şi 1 după intenție, 14 a principiului inversării dependențelor, principiu segregării interfețelor nu este reflectat în nici un şablon de proiectare dar în unele contexte este aparent la implementarea şablonului adaptor.

6 Bibliografie:

1 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.objectmentor.com/resources/articles/srp.pdf
2 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.objectmentor.com/resources/articles/ocp.pdf
3 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.objectmentor.com/resources/articles/lsp.pdf
4 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.objectmentor.com/resources/articles/isp.pdf
5 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.objectmentor.com/resources/articles/dip.pdf
6 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.oodesign.com/open-close-principle.html
7 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.oodesign.com/dependency-inversion-principle.html
8 *Robert Martin, Design Principles and Design Patterns*[Resursa electronică].-Regim acces: http://www.scribd.com/doc/6752666/Principles-and-Patterns
9 Walter Zimmer, *Relationships between Design Patterns*.
10 *Robert Martin, The Principles of OOD* [Resursa electronică].-Regim acces: http://www.oodesign.com/interface-segregation-principle.html