Modelarea riguroasa cu OCL a sistemelor software

Asist. Anca Ioana ANDRONESCU
Catedra de Informatica Economica, A.S.E. Bucuresti

The Unified Modeling Language (UML) has been for several years the leading standard for modeling of software systems. Object Constraint Language (OCL), the expression language that comes with UML, allows to formally specify constraints on a UML model. The goal of the present article is to show that expressions written in a precise, mathematically based language like OCL offer a number of benefits over the use of diagrams to specify a software system.
***Keywords**: software systems, UML, Object Constraint Language, formal specification, precise modeling*.

Caracteristici de baza ale OCL

Object Constraint Language (OCL) a fost definit în cadrul limbajului UML ca standard pentru specificarea riguroasa a expresiilor care pot adauga informatii esentiale modelelor orientate obiect si altor artefacte ale modelarii obiectuale. În versiunea 1.1 a limbajului UML informatiile au fost proiectate pentru a exprima constrângeri. O constrângere este definita ca „*o restrictie aplicata uneia sau mai multor valori ale unui sistem sau a unei parti dintr-un sistem sau model orientat obiect*". Odata cu versiunea 2.0 a limbajului UML s-a ajuns însa la concluzia ca aceste informatii aditionale trebuie sa contina mai mult decât constrângeri si sa poata defini interogari, referi valori si stabili conditii sau reguli ale afacerii într-un model.

Pâna la includerea sa în standardul UML în 1997, OCL a evoluat de la un limbaj de expresii pentru metoda Syntropy la un limbaj de modelare a afacerilor folosit de IBM.

Cele mai importante caracteristici ale OCL sunt:

- Este un *limbaj de expresii*; aceasta este o garantie a faptului ca expresiile OCL nu vor avea efecte secundare asupra celorlalte elemente ale modelului. Când o expresie OCL este evaluata, ea va returna o valoare care nu poate schimba nimic în cadrul modelului. Astfel o stare a sistemului nu se va schimba niciodata datorita evaluarii unei expresii OCL, chiar daca expresia este folosita pentru a exprima schimbarea unei stari.
- Este un *limbaj de modelare* si nu un limbaj de programare; nu este posibil sa se scrie logica sau fluxuri de control în OCL. Deoarece este un limbaj de modelare, nu orice element din OCL va putea fi direct executabil. Toate problemele legate de implementare sunt în afara scopului limbajului si nu pot fi exprimate în OCL.
- Este un *limbaj formal*; o caracteristica remarcabila a OCL o reprezinta fundamentul sau matematic. Limbajul se bazeaza pe teoria matematica a multimilor si pe calculul predicatelor, dar notatiile nu folosesc simboluri matematice. OCL nu intentioneaza sa înlocuiasca limbaje formale existente (precum VDM sau Z), ci sa fie o alternativa mai usor de utilizat pentru introducerea formalismului în modelarea sistemelor software.
- Este un *limbaj bazat pe tipuri*, astfel încât fiecare expresie OCL are un anumit tip, iar pentru a fi bine formata expresia OCL trebuie sa satisfaca anumite reguli de conformitate a tipurilor.

Modelarea sistemelor software cu OCL

Modelarea si în special modelarea sistemelor software este sinonima cu procesul de realizare a diagramelor. Cele mai mult modele sunt formate dintr-o multime de elemente de modelare vizuala, uneori însotite de text în limbaj natural. Informatiile continute într-un astfel de model au tendinta de a fi incomplete, informale, imprecise si uneori chiar inconsistente. Multe din erorile care apar în cadrul modelarii sunt cauzate de limitele pe ca-

re le impun diagramele folosite în model. O diagrama UML (cum ar fi diagrama de clase) nu este de cele mai multe ori suficient de rafinata pentru a surprinde toate aspectele relevante ale unei specificatii. Apare astfel nevoia descrierii unor constrângeri aditionale referitoare la obiectele continute în model. În general, constrângerile sunt specificate în limbaj natural si practica a aratat ca aceasta modalitate de descriere va genera întotdeauna ambiguitati. Pentru a descrie constrângeri clare, lipsite de ambiguitati au fost dezvoltate asa-numitele limbaje formale. Dezavantajul limbajelor formale traditionale este acela ca sunt greu de utilizat de catre persoane care nu au o formatie matematica solida. OCL umple acest gol fiind limbaj formal, dar în acelasi timp, usor de citit si de scris.

Sa consideram exemplul din figura 1 în care asocierea dintre clasele Zbor si Persoana, prin care se exprima faptul ca un grup de persoane sunt pasageri ai unui zbor, are multiplicitatea „multi" de partea clasei Persoana. Astfel se sugereaza ca numarul de pasageri ai unui zbor este nelimitat.

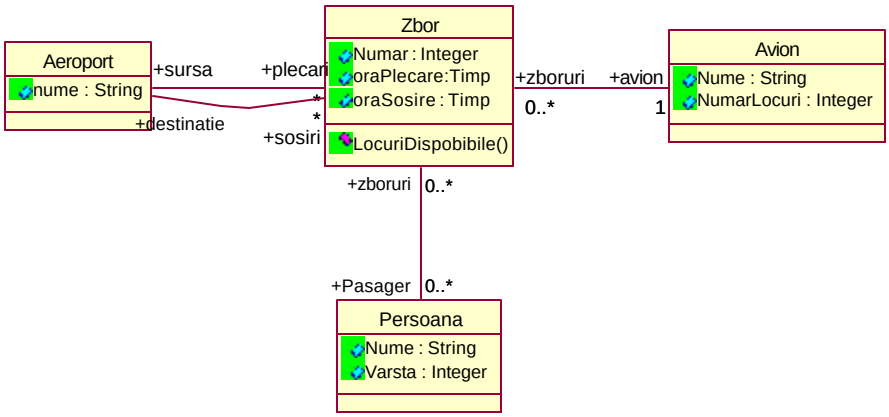

Fig. 1. Diagrama de clase pentru un sistem de evidenta a pasagerilor pe curse aeriene

În realitate, însa, numarul de pasageri trebuie restrictionat la numarul de locuri din avionul asociat zborului respectiv. Deoarece este imposibila exprimarea acestei restrictii prin intermediul diagramei de clase, o modalitate eficienta de specificare a acestei multiplicitati consta în adaugarea urmatoarei constrângeri OCL:

```
context Zbor
inv: Pasager → size( ) <= Avi-
on.NumarLocuri
```

Expresiile scrise în limbaje precise, fundamentate matematic (precum OCL) ofera numeroase beneficii atunci când sunt folosite împreuna cu diagramele prin care se modeleaza un sistem. Specificatiile scrise în limbaj formal nu genereaza ambiguitati si ajuta la realizarea unui model mai precis si mai detaliat. Cel mai mare avantaj este acela ca expresiile nu mai pot fi interpretate în mod diferit de membrii echipei de dezvoltare si ca pot fi verificate cu ajutorul unor instrumente automate pentru a fi siguri ca ele sunt corecte si consistente în raport cu alte elemente ale modelului. Nu în ultimul rând, prin generarea automata de cod, se vor produce secvente de cod mult mai complete si mai apropiate de versiunea finala a implementarii sistemului.

Pentru a scoate în evidenta importanta limbajului OCL în proiectarea riguroasa a sistemelor software sa analizam exemplul din figura 2 care modeleaza un sistem simplu de acordare de credite si care contine trei clase: Persoana, Bun si Credit împreuna cu asocierile dintre ele.

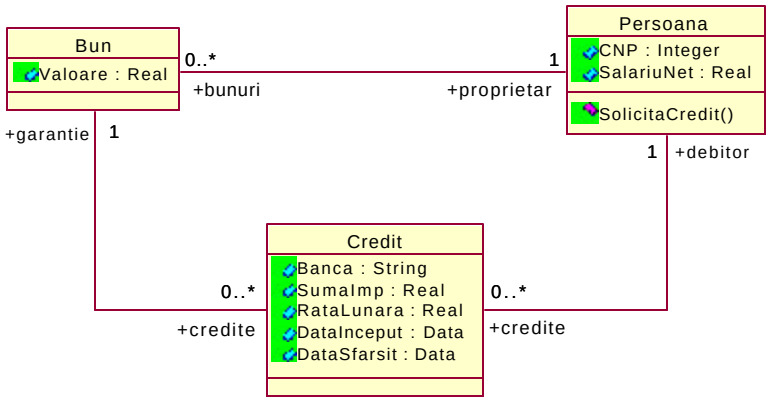

Fig. 2. Diagrama de clase pentru un sistem de acordare de credite

Urmatoarele reguli vor trebui aplicate acestui model:

1. o persoana poate obtine un credit numai daca bunul pe care îl depune drept garantie îi apartine;
2. data de început a perioadei de creditare trebuie sa fie anterioara datei de sfârsit;
3. Codul Numeric Personal (CNP) trebuie sa fie unic pentru toate persoanele;
4. o persoana poate solicita un nou credit numai daca suma lunara pe care o plateste pentru toate creditele contractate nu depaseste 30% din salariul sau net lunar.

O diagrama UML nu poate evidentia astfel de informatii. Daca regulile mentionate mai sus nu sunt documentate corespunzator, diferite persoane care citesc acest model pot face presupuneri diferite privind cerintele sistemului, ceea ce va conduce la o întelegere incorecta si implicit la o implementare incorecta a acestuia. Scrise în limbaj natural regulile ramân ambigue si lasa loc interpretarilor. Îmbogatind însa modelul cu expresii OCL asociate acestor reguli, putem obtine o descriere mai precisa si mai completa a sistemului. Pentru exemplul de mai sus cele patru reguli exprimate în limbaj natural pot fi transformate în urmatoarele reguli OCL:

```
1. context Credit
   inv:garantie.proprietar = debitor
2. context Credit
   inv:DataInceput < DataSfarsit
3. context Persoana
   inv:Persoana::allInstances( ) →
isUnique(CNP)
4. context Persoana:
SolicitaCredit(suma:Real;
garantie:Bun)
   pre: self.Credite.RataLunara→
sum( ) <= self.SalariuNet * 0.3
```

Includerea în modelul UML a regulilor sub forma unor expresii scrise în OCL prezinta o deosebita importanta deoarece:

- creste precizia modelului si implicit scade posibilitatea de a interpreta gresit aceste reguli;
- erorile sunt descoperite din fazele timpurii ale dezvoltarii sistemului, atunci când costul de solutionare a lor este relativ redus;
- ofera o modalitate eficienta si unitara de comunicare între analistii si programatorii care dezvolta sistemul.

În prezent exista disponibila o mare varietate de instrumente care pot translata OCL în cod scris într-un limbaj de programare, pot simula date de test, pot verifica consistenta modelelor etc. Un instrument de dezvoltare nu poate interpreta reguli scrise în limbaj natural. Regulile scrise în OCL includ toate informatiile necesare pentru a trece la generarea automata de cod. În acest mod implementarea se face mai rapid si mai eficient, ducând la cresterea nivelului de maturitate al procesului de dezvoltare al sistemului software.

Concluzii

OCL este un limbaj textual, precis, folosit pentru a completa limbajele grafice utilizate în modelarea sistemelor orientate obiect. El permite descrierea unor constrângeri asociate

modelului, care nu ar fi putut fi exprimate cu ajutorul notatiilor standard ale diagramelor. Cel mai important aspect referitor la OCL este acela ca el constituie o parte a standardului de modelare UML. Acest avantaj a facut ca limbajul sa devina înca de la aparitia sa mai popular decât alte limbaje de specificare formala precum VDM sau Z.

În exemplul prezentat am vazut ca si un model simplu, continând doar trei clase poate avea nevoie de multe informatii suplimentare scrise în OCL pentru a deveni complet, consistent si precis.

Bibliografie

- M. Fowler, S. Kendal, *UML Distilled – Applying the Standard Object Modeling Language*, Addison-Wesley, 1997
- Object Management Group, *Object Constraint Language Specification în OMG Unified Modeling Language Specification*, iunie 1999, capitolul 7
- C. Richter, *Designing Flexible Object-Oriented Systems with UML*, Macmillan Technical Publishing, 1999
- J. Warmer, A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, Addison Wesley, 2003
- http://www.klasse.nl/OCL
- www.omg.org