# MODELE ORIENTATE OBIECT PENTRU MANAGEMENTUL DATELOR ÎN GEOGRAPHICAL INFORMATION SYSTEM (GIS)

**nat. Angela Ioniță,**
**nat. Tudor Macovei,**

*nstitutul de Cercetări în Informatică*

**lezumat:** Odată cu lărgirea ariei de aplicabilitate a GIS s-a produs și ezvoltarea tehnicilor computerizate, astfel încît viitoarele sisteme vor ere integrarea GIS cu "remote sensing data" [1]. Pe de altă parte, experții n ingineria programării au investigat o noua tehnică: abordarea orientată biect [2]. Această abordare este aplicabilă, fie pentru integrarea "remote ensing" cu GIS, fie pentru managementul de date grafice și non-grafice. n particular, orientarea obiect permite integrarea diverselor abordări sub ın concept universal, cum ar fi reprezentarea grafică, informație tematică, chimbările temporale și caracteristicile bazelor de date clasice memorare și regăsire de date, partajare de date, consistență etc.) [3].

**Cuvinte cheie:** baze de date orientate obiect, modele orientate obiect, noștenire, genericitate, Geographical Information System (GIS).

## 1. Introducere

Experții în GIS au promovat abordarea orientată obiect pentru GIS. Aceasta este o tehnică avansată, bazată pe definirea tipurilor de obiecte în combinație cu operațiile corespunzătoare, într-o manieră modulară ce se caracterizează prin claritatea codului și ușurința ntreținerii lui.

Conceptul de obiect a căpătat o mare importanță în lomeniul bazelor de date [19]. Sistemele standard de baze le date, construite pe modele de date, ca de exemplu, nodelul relațional, sînt o soluție bună pentru aplicațiile din omeniul business, GIS, CAD/CAE. Programarea plicațiilor implică o comunicare între limbajul de iterogare al sistemului și un limbaj extern de programare. ceasta are o serie de dezavantaje: programatorul trebuie să vețe două limbaje de programare diferite, comunicația te la un moment dat tuplu și cele două limbaje de ogramare au contexte de lucru diferite.

Paradigma orientării obiect este un mod bun de zolvare a acestor probleme. Caracteristicile limbajelor entate obiect [4] sînt noțiunile de obiect, clasa și ıştenire. Opus tuplului din modelul relațional, obiectul o identitate care este independentă de valoarea lui. easta este folosită, de obicei, la partajarea și actualizarea ectelor. Clasele factorizează structura comună a ectelor, iar moștenirea este un foarte bun instrument de delare, care permite să se reutilizeze obiectele sau clasele tente și să se definească obiecte și clase specializate noi , [6], [7]).

Fundamentele teoretice ale limbajelor orientate obiect ([8], [9], [10]) există deja. Aceste modele formalizează facilitățile standard ale orientării obiect (moștenire și generalizare) (polimorfism de tip). Obiectele bazelor de date sînt foarte bine structurate și sînt construite folosind constructori ca set și tuplu. Au fost propuse deja modele de date care manipulează valori foarte structurate ([11], [12], [13]). Aceste modele nu iau în considerare noțiunea de identitate a obiectului sau mecanismul de moștenire. Există și alte modele pentru baze de date orientate obiect. FAD [11] este un model de date cu constructori de seturi și tuple și cu identități de obiecte. FAD nu modelează moștenirea tipurilor. O altă abordare interesantă este propusă în LOGIN [15] unde este introdusă relația de moștenire într-un context de programare logică. Obiectele acestui model sînt limitate la tuple și nu există identitate de obiect, deși partajarea obiectelor este schițată prin introducerea variabilelor.

Prezenta abordare propune următoarele facilități pentru model:

- o viziune uniformă asupra structurilor de tip si a obiectelor: nu se face distincție între obiecte și structuri de tip. Obiectele sînt văzute ca tipuri cu o singură valoare;
- construcția tipurilor generice: conform cu [13], unele aplicații nu sînt ușor de manipulat folosind moștenirea, dar sînt ușor de dezvoltat folosind generalizarea fără constrîngeri;
- o semantică de incluziune a mulțimilor pentru relația de moștenire cu aplicație la tipurile generice și non-generice;
- moștenire multiplă.

Să încercăm mai întîi să introducem cele mai importante noțiuni ale modelului, folosind exemple. Entitățile constau dintr-un identificator (numele entității) și o valoare. Valorile pot fi valori simple (de exemplu -2, 0, 3.1415, "Muzeul de Istorie" sau tipuri (integer, real, string). De observat că integer reprezintă o valoare de bază ca și simbolul 1. Intuitiv, valoarea integer reprezintă mulțimea tuturor întregilor (în baza de date), în timp ce 1 reprezintă numai unul dintre acești întregi, și anume 1. Cele două valori pot fi considerate, atît ca tipuri (structuri tip), pentru că 1 poate fi văzut ca un tip cu o singură valoare, cît și ca obiecte. Din acest motiv se introduce noțiunea de entitate, care acoperă noțiunile de structură tip și de obiect.

Se pot acum construi entități de complexitate arbitrară folosind constructorii de set si de tuplu. O valoare de tip set este un termen ca $\{i_1, i_2\}$, unde $i_1$ si $i_2$ sînt identificatori de entități. Date fiind entitățile ($i_1$, 1) și ($i_2$, -2), termenul $\{i_1, i_2\}$ reprezintă un set/toate seturile care conține/conțin întregii 1 si -2. Valorile tuplu se pot construi în același mod: [nume: $i_3$, vechime: $i_4$] este un termen de tip tuplu. De notat că toate cîmpurile unui tuplu poartă un nume.

Se pot construi astfel entităţi de complexitate oricît de mare. Se vor adopta următoarele convenţii (valabile numai în contextul acestui articol): valorile entitătilor de baza (de exemplu **1**) sînt scrise întărit, în timp ce identificatorii sînt subliniaţi. Vom considera în continuare următoarele entităţi:

(integer, **integer**)
(string, **string**)
(constructie, [nume: string, vechime: integer])
(clădire,[nume: string, vechime: integer, suprafaţă: integer])
(cartier, {clădire})

(constr_nume, **'Muzeul de Arta'**)
(clădire_nume, **'Posta Centrală'**)
(patruzeci, **40**)
(cincimii, **5000**)
(constr, [nume: constr_nume, vechime: patruzeci])
(clădire,[nume:clădire_nume, vechime: patruzeci, suprafată: cincimii])

Deşi s–au definit (şi s-au folosit) toate aceste entităti într-un mod uniform, intuitiv se poate face distincţie între primele cinci entităţi (integer,... cartier) şi ultimele şase (constr_nume,.., clădire). Entităţile, ca de exemplu constructie, reprezintă setul tuturor construcţiilor din baza de date, adică setul tuturor entităţilor tuplu avînd cîmpurile corespunzătoare. constr este o astfel de entitate. În mod similar, entitatea constr reprezintă setul tuturor entitătilor tuplu avînd numele constr_nume şi vechimea patruzeci.

Construcţie se poate considerà o structură tip, iar constr, un obiect, dar se poate considera constr şi ca structura tip avînd în acest caz o singură valoare. Aceasta abordare va genera o semantică uniformă pentru tipuri şi obiecte.

Între valori, se poate defini noţiunea de rafinare. Se consideră ordonarea prin ≤ Iată cîteva exemple de rafinare:

1. **30** ≤ **integer**. Aceasta arată că orice entitate cu valoarea 30 este o entitate cu o valoare întreagă;
2. **'Muzeul de Artă'** ≤ **string**, pentru acelaşi motiv;
3. [nume:string, vechime:integer, suprafaţă:integer]≤ [nume: string, vechime: integer]. Această relaţie este mai "subtilă" decît cele anterioare. Ea arată că tripletele, ca de exemplu "clădire", pot fi considerate cazuri speciale de perechi, cum ar fi "construcţie". Interpretarea pentru tuple este similară celei din [17].

Se pot observa pe aceste exemple, intuitiv, relaţiile de apartenenţă şi incluziune. Inegalitatile 1 şi 2 sînt relaţii de apartenenţă. Ultima este o relaţie de incluziune.

Acum se doreşte modelarea unui tip de generalizare numită generalizare fără constrîngeri. În acest scop, se vor introduce variabile în termenii ce definesc entităţile. Expresiile următoare sînt exemple de entităţi generice:

(ab_tuplu, [a: x, b: y])

(ab_=_tuplu, [a: x, b: y])

Intuitiv, entitatea cu numele ab_tuplu reprezintă entităţile de valoare [a: x, b: y] pentru toate entităţile posibile x şi y din baza de date. Variabilele x şi y sînt abstracţii ale identificatorilor de entităţi. Se poate gînd acum ab_tuplu ca o notaţie pentru toate entităţile tuplu avînd două cîmpuri 'a' şi 'b'. Entitatea generică ab_=_tuplu este o notaţie pentru toate entităţile tuplu cu două cîmpuri 'a' şi 'b' avînd aceeaşi valoare.

Unei entităţi generice i se poate aplica o substituţie, adică variabilele pot fi substituite prin identificatori. De exemplu, aplicarea substituţiei
<x/real, y/real> entităţii cu numele ab_tuplu conduce la entitatea (ab_tuplu(integer, real), [a: integer, b: real]).

## 2. Entităţile şi semantica lor

### 2.1. Sintaxa

Considerăm cîteva valori simple: (**1**, **2**, .., **3.14**, **'Muzeul de Istorie'**,.., **integer**, **real**, **string**). Vom folosi constructorii de set şi tuplu pentru a construi valorile structurate pornind de la cele de bază. Se presupun date următoarele patru mulţimi disjuncte:

- o mulţime numărabilă $C$, reuniunea dintre mulţimea întregilor N, a realilor R, a stringurilor S şi a mulţimii {nil, any, integer, real, string}. În mod formal N (respectiv R şi S) sînt mulţimile numeralelor ce reprezintă întregii (respectiv realii şi şirurile). Pentru simplificare nu vom face distincţie între valori şi reprezentarea lor;
- o mulţime numărabilă infinită $\Re$ de simboluri, atributele. Elementele lui $\Re$ sînt nume pentru cîmpurile tuplu;
- o mulţime numărabilă infinită $\nu$ de simboluri, variabilele. Elementele lui $\nu$ acoperă identificatorii. Ele sînt notate prin x, y şi z cu sau fără indici;
- o mulţime numărabilă $L$ de simboluri numite simboluri fundamentale. Simbolurile fundamentale se pot extinde astfel: un identificator valoric este o expresie $i(e_1,...,e_n)$ unde i este un identificator fundamental din $L$ şi $e_1,...,e_n$ sînt fie variabile, fie identificatori fundamentali, fie identificatori valorici. Identificatorul fundamental i este numit rădăcina identificatorului valoric. Se notează cu $L^*$ setul tuturor identificatorilor (care vor fi scrişi subliniat).

**Observaţie**: În continuare, termenul de "identificator" se va referi fie la identificatori fundamentali, fie la identificatori valorici. Se poate defini acum noţiunea de termen.

**Def. 2.1:** Se notează cu $T$ mulțimea tuturor termenilor definiți astfel:

- orice element $v \in C$ este un termen numit termen de bază;
- termenii structurați ca set sînt expresii de forma: $\{e_1,...,e_n\}$, unde $e_i$ sînt variabile sau identificatori;
- termenii structurați ca tuplu sînt expresii de forma: $[a_1: e_1,..., a_n: e_n]$, unde $a_1,..,a_n$ sînt atribute care aparțin lui $\Re$, iar $e_i$ sînt variabile sau identificatori.

Se numește grad(t) numărul variabilelor care apar în termenul t. Un termen t este generic, dacă gradul lui este mai mare decît 0, altfel el este termen fundamental.

Exemple de termeni fundamentali:

- integer
- [nume: string, vechime: integer]
- {integer, set(real)}

Primul termen reprezintă mulțimea tuturor întregilor. Al doilea, reprezintă mulțimea tuturor tuplelor avînd un atribut nume, ale cărui valori sînt șiruri, și un atribut vechime, ale cărui valori sînt întregi. Ultimul termen reprezintă toate multimile (heterogene) conținînd întregi și mulțimi de reali.

Exemple de termeni generici:

- {x}
- {integer, x}
- [a: x, b: x]
- {nume: string, copii: set(real)}

Primul termen reprezintă "mulțimea" generică omogenă. Intuitiv, aplicarea substituțiilor acestui termen va produce toate mulțimile omogene posibile. Al doilea termen reprezintă o mulțime eterogenă generică, ce conține întregi. Al treilea reprezintă un tuplu generic avînd două cîmpuri ale aceleiași valori. Se observă că expresii ca {integer, integer} sînt termeni valizi, prin urmare, semantica lui {integer, integer} este aceeași cu semantica lui {integer}.

Putem defini acum noțiunea de entitate.

**Def. 2.2:** O entitate este o pereche e=(id, t), unde id este un identificator și t este un termen din $T$ astfel încît:

- t este un termen fundamental și id este un identificator fundamental din $L$, în acest caz e se numește entitate fundamentală;
- t este un termen generic și id este un identificator valoric $i(e_1,...,e_p)$, astfel încît orice variabilă care apare în t, va apare și în id. În acest caz, e se numește entitate generică.

În mod natural, se definesc și noțiunile de entitate de bază, structurată ca set și structurată ca tuplu. Dacă e=(id,t) este o entitate, atunci identifier(e) este identificatorul id, iar term(e) este termenul t. Figura 1 prezintă cîteva exemple de entități. Se vor folosi substituții ale variabilelor. O substituție σ este o expresie $<x_1/i_1,...,x_n/i_n>$, unde $x_j$ sînt variabile și $i_j$ sînt identificatori. Ca de obicei, dacă t este un termen generic și σ o substituție, se notează t. σ termenul obținut prin înlocuirea în t a variabilelor prin identificatorii corespunzători lui σ. Dacă i este un identificator și σ o substituție, se notează i.σ identificatorul obținut prin înlocuirea în i a variabilelor cu identificatorii corespunzători lui σ. În sfîrșit, dacă e = (i,t) este o entitate și σ este o substituție, se notează e. σ = (i. σ,t. σ). Iată cîteva exemple de substituții:

(set(x), {x}).<x/integer> = (set(integer), {integer})
(set(x), {x}).<x/real> = (set(real), {real})
(vechime_tuplu(x), [vechime:x]).<x/integer> =
= (vechime_tuplu(integer), [vechime:integer])

Introducem noțiunea de mulțime consistentă de entități.

**Def. 2.3** Fie ε o mulțime de entități. ε este consistentă dacă și numai dacă sînt îndeplinite următoarele condiții:

- nu există două entități cu aceeași rădăcină identificator;
- pentru orice entitate din ε și pentru orice identificator id al rădăcinii i care apare în term(e), există o entitate în ε care are i ca rădăcină identificator;
- pentru orice entitate (id,t) din ε, dacă id este un identificator valoric, atunci id conține numai variabile distincte.

Intuitiv, într–o mulțime consistentă de entități, fiecare entitate are un unic identificator, nu are "pointeri de balansare", iar identificatorii entității generice conțin numai variabile. De exemplu, (set(x), {x}) este o entitate generică putînd să apară într-o mulțime consistentă de entități. Deci, (set(integer), {integer}) nu poate face parte dintr-o mulțime consistentă de entități pentru că nu definește "complet" entitatea set. Acesta este rezultatul substituției <x/integer> în entitatea (set(x), {x}).

Figura 1 este un exemplu de mulțime consistentă de entități. Se observă că această construcție permite autoreferirea entităților. De asemenea, entitățile sînt folosite pentru a modela tipurile și obiectele și în același mod se pot defini tipurile cu autoreferire (de exemplu "construcție" în figura 1), ca și obiectele cu autoreferire. Un aspect important al definiției entităților este genericitatea. De exemplu, entitatea (set(x), {x}) din figura 1 este o entitate generică reprezentînd orice tip de mulțimi (omogene). Mai mult, termenii fundamentali și generici sînt manipulați într-un mod uniform. Originalitatea acestui model constă în faptul că nu se face distincție între structurile de tip si obiecte. Aceste două noțiuni sînt subsumate noțiunii de entitate. De exemplu, (unu, 1) și (integer, **integer**) sînt tratate la fel.

În această secțiune vom defini semantica modelului propus (în continuare, pentru a deosebi termenii structurați ca set și mulțimea din metalimbaj, ultima se noteaza întarit). Fie ε o mulțime consistentă de entități. Pentru a defini interpretările termenilor din ε se folosește o extensie a lui ε cu toate substituțiile posibile ale entităților din ε. Se definește ε* după cum urmează:

$$\varepsilon^* = \varepsilon \cup e \,.\, \sigma$$

unde $e \in \varepsilon$ și σ este o substituție.

Un termen t este definit pe ε dacă el conține numai identificatori (sau substituții ale acestor identificatori) ai entităților din ε. În următoarea definiție se va da o interpretare pentru toți termenii definiți pe ε Interpretarea unui termen fundamental este un subset al lui ε*. Pentru un termen generic t, interpretarea acestuia este o funcție care mapează toate substituțiile σ pe interpretarea substituției corespunzătoare lui t.

**Def. 2.4** Fie ε o mulțime consistentă de entități. Fie t un termen definit pe ε. Interpretarea $I_\varepsilon(t)$ se definește astfel:

**Termeni fundamentali**:

Termeni de bază:

$$I_\varepsilon\,(nil) = (i,nil) \in \varepsilon$$

$$I_\varepsilon\,(any) = \varepsilon^*$$

$$I_\varepsilon\,(v) = I_\varepsilon(nil) \cup (i,v) \in \varepsilon$$

dacă v este un simbol al lui *C*.

$$I_\varepsilon(integer) = I_\varepsilon(nil) \cup (i,n) \in \varepsilon \wedge n \in N$$

$$I_\varepsilon(real) = I_\varepsilon(nil) \cup (i,r) \in \varepsilon \wedge r \in R$$

$$I_\varepsilon(string) = I_\varepsilon(nil) \cup (i,s) \in \varepsilon \wedge s \in S$$

$$I_\varepsilon(boolean) = I_\varepsilon(nil) \cup (i,b) \in \varepsilon \wedge b \in B$$

Termeni structurați ca set:

dacă $t = \{i_1,..,i_n\}$, unde $i_i$ este un identificator

$$I_\varepsilon(t) = I_\varepsilon(nil) \cup (i \cdot \sigma, v \cdot \sigma) \in \varepsilon^*$$

unde $v \cdot \sigma = \{j_1, ... j_q\}$ și

$$j_k \in \bigcup_{e=1}^{n} I_\varepsilon\,(term(i_e))$$

Termeni structurați ca tuplu:

dacă $t = [a_1{:}i_1,.., a_n{:}i_n]$, unde $i_i$ este un identificator

$$I_\varepsilon(t) = I_\varepsilon(nil) \cup (i \cdot \sigma, v \cdot \sigma) \in \varepsilon^*$$

unde $v \cdot \sigma$ este un termen structurat ca tuplu $[a_1{:}j_1,...,a_n{:}j_n,...,a_{n+p}{:}j_{n+p}]$ astfel încît pentru toți $k \in 1,...,n$

$$j_k \in I_\varepsilon(term(i_k))$$

**Termeni generici**:

Dacă t este un termen generic de grad p, $I_\varepsilon(t)$ este o funcție de la mulțimea tuturor substituțiilor, în $2\varepsilon^*$ astfel încît:

$$I_\varepsilon(t)\,(\sigma) = I_\varepsilon\,(t \cdot \sigma)$$

1. (nil,**nil**)
2. (integer,**integer**)
3. (string,**string**)
4. (construcție,[nume:string,vechime:integer, proprietar:construcție])
5. (clădire,[nume:string,vechime:integer proprietar:construcție, suprafața:integer])
6. (cartier, {clădire})
7. (sector, {cartier})
8. (vechime_tuplu(x), [vechime:x])
9. (set(x), {x})
10. (unNume, **'Muzeul de Arta'**)
11. (altNume, **'Poșta Centrala'**)
12. (treizeci, **30**)
13. (patruzeci, **40**)
14. (cincimii, **5000**)
15. (constr, [nume:unNume, vechime:treizeci, proprietar:nil])
16. (oClădire, [nume:altNume, vechime:patruzeci, proprietar:nil, suprafața:cincimii])
17. (unCartier, {cladire})

**Figura 1 - O mulțime consistentă de entități**

Să considerăm mulțimea consistentă din figura 1. Următorul tabel da interpretarea pentru cîțiva termeni:

| | t | $I_\varepsilon(t)$ |
|---|---|---|
| 1 | **30** | nil, treizeci |
| 2 | **integer** | nil, treizeci, patruzeci, cincimii |
| 3 | **string** | nil, unNume, altNume |
| 4 | [nume:string, vechime:integer] | nil, constr, oClădire |
| 5 | {construcție} | nil, unCartier, set(constr), set(oClădire) |
| 6 | {integer, string} | nil, set(treizeci), set(unNume) |
| 7 | [vechime:x] | <x/integer> → nil,constr,oClădire, vechime_tuplu(nil) vechime_tuplu(treizeci) vechime_tuplu(patruzeci) vechime_tuplu(cincimii) <x/real> → nil, vechime_tuplu(nil) |

Acest tabel arată că pentru termenii fundamentali, ca **integer**, interpretarea este mulțimea identificatorilor lui ε care reprezintă entitățile cu o valoare întreagă.

### 2.3. Ordonarea asupra termenilor

Introducem acum o ordonare numită rafinare. Relația de rafinare ne permite să afirmăm că 1 este un întreg sau că o clădire este o construcție. Interpretarea pentru tupluri, similară cu cea din [17], ne permite să afirmăm că un tuplu [a:i, b:j] este o rafinare a tuplului [a:i]. Această proprietate este foarte importantă datorită faptului ca relația de rafinare (pentru termenii fundamentali) este definită ca incluziune a interpretărilor.

**Def. 2.5** Fie ε o mulțime consistentă de entitați. Fie v și w doi termeni definiți pe ε. Se definește relația de rafinare $v \leq_{\varepsilon} w$ astfel:

**Termeni fundamentali**:

$v \leq_{\varepsilon} w$ dacă şi numai dacă

$$I_{\varepsilon}(v) \subseteq I_{\varepsilon}(w)$$

**Termeni generici**: Fie $x_1,\dots,x_p$ variabilele lui v şi $y_1,\dots,y_q$ variabilele lui w.

$v \leq_{\varepsilon} w$ dacă şi numai dacă există o funcţie parţială injectivă $\theta$ de la $\{1,..,q\}$ la $\{1,..,p\}$ astfel încît pentru orice identificatori $i_1,\dots,i_p$, $j_1,\dots j_q$ verificînd $i_{\theta(\alpha)} = j_{\alpha}$ ori de cîte ori $\theta(\alpha)$ este definită, avem:

$$I_{\varepsilon}(v. < x_1/i_1,\dots,x_p/i_p>) \subseteq I_{\varepsilon}(w. < y_1/j_1,\dots,y_p/j_p>)$$

În cazul termenilor fundamentali, relaţia de rafinare corespunde chiar cu incluziunea interpretărilor. Definirea entităţilor generice se poate explica astfel: v este o rafinare a lui w dacă unele variabile ale lui w se pot mapa pe unele variabile ale lui v astfel încît, atunci cînd se aplica substituţiile la ambii termeni cu aceleaşi entităţi în variabilele corespunzătoare, ordonarea se păstrează. Această mapare este o mapare variabilă asociată cu v şi w. Definirea rafinării pentru termenii fundamentali este un caz special al definirii pentru termenii generici. S-a făcut o separare a celor două cazuri pentru a face definiţia cît mai clară.

Ordonarea definită mai sus depinde de mulţimea $\varepsilon$ Este necesară o ordonare generală asupra termenilor, independentă de mulţimea consistentă de entităţi la care ne referim. Această ordonare va fi folosită pentru a modela relaţia de moştenire.Dacă $\varepsilon$ este o mulţime consistentă de entităţi şi t este un termen, închiderea lui t în $\varepsilon$ (notată $\bar{t}$) este cea mai mică submulţime consistentă $\varepsilon'$ a lui $\varepsilon$ astfel încît orice identificator care apare în t identifică o entitate în $\varepsilon'$ De exemplu, închiderea {construcţie} în mulţimea de entităţi din figura 1 este:

(integer, **integer**),
(string, **string**),
(construcţie, [nume:string, vechime:integer, proprietar:construcţie])

**Def. 2.6** Fie $\varepsilon$ o mulţime consistentă de entităţi, fie v şi w doi termeni definiţi pe $\varepsilon$; v rafinează w (notat $v \leq w$) dacă şi numai dacă pentru toate seturile consistente de entităţi $\varepsilon'$ care conţin închiderile lui t şi t',avem $v \leq \bar{\varepsilon}$ . w.

Figura 2 arată o reprezentare grafică a unor legături de rafinare si de substituţie. Săgeţile simple reprezintă legăturile de rafinare, iar cele duble reprezintă legăturile de substituţie. O trăsătura importantă a acestui model constă în faptul că termenii fundamentali şi generici sînt manipulaţi în acelaşi mod. Aceasta conduce la grafuri generale de rafinare şi substituţie, ca în figura 2. În această figură, există mai multe drumuri de la termenul "[a:x]" la termenul "[a:integer, b:real]". Un singur drum substituie întîi x cu integer in "[a:x]" şi apoi îl rafinează în "[a:integer, b:real]". Celălalt drum rafinează întîi termenul generic [a:x] în termenul generic "[a:x, b:y]" şi apoi substituie x cu integer si y cu real.

De notat că se modelează de asemenea moştenirea multiplă. De fapt, un termen poate fi o rafinare a mai multor termeni. Figura 3 zugrăveşte o reprezentare grafică a unui exemplu de moştenire multiplă. Termenul construcţie este o rafinare a termenilor bloc şi vilă. Ambii termeni sînt la rîndul lor rafinări ale termenului construcţie.

[a:x]

[a:x,b:y] [a:x,b:integer]

[a:integer]

[a:integer,b:real] [a:2]

**Figura 2 - Rafinare şi substituţie**

casă

bloc vilă

construcţie

casa = [nume:string, vechime:integer]
bloc = [nume:string, vechime:integer, stare:string]
vila = [nume:string, vechime:integer, proprietari:set(x/string)]
construcţie = [nume:string, vechime:integer, stare:string, proprietari:set(x/string)]

**Figura 3 - Moştenire multiplă**

## 3. Caracterizarea sintactică a rafinării

În această secţiune vom încerca să dăm o caracterizare a relaţiei de rafinare. Considerăm mai întîi cazul termenilor fundamentali.

**Teorema 3.1.** Fie $\varepsilon$ o mulţime consistentă de entităţi.

Termeni fundamentali de bază:

**nil** $\leq$ t pentru toţi termenii t,
t $\leq$ **any** pentru toţi termenii t,
n $\leq$ **integer** pentru toţi $n \in N$,
r $\leq$ **real** pentru toţi $r \in R$,
s $\leq$ **string** pentru toţi $s \in S$.

Termeni fundamentali structuraţi ca set:

Fie $t=\{e_1,\dots,e_n\}$ si $t'=\{e_1',\dots,e_p'\}$ doi termeni definiţi pe $\varepsilon$.

$t \le t'$ dacă și numai dacă pentru toți $i \in \{1,..,n\}$, există $j \in \{1,..,p\}$ astfel încît $term(e_i) \le term(e_j')$.

Termeni fundamentali structurați ca tuplu:

Fie t și t' doi termeni definiți pe $\varepsilon$.

$t \le t'$ dacă și numai dacă pentru toate cîmpurile $a_i$ ale valorii $e_i$ în t, există un cîmp $a_i$ de valoare $e_i'$ în t' astfel încît $term(e_i) \le term(e_i')$.

**Comentariu**: Această teoremă arată că relația de rafinare pentru tuple este la fel ca în [17].

**Demonstrație** : Demonstrația acestei teoreme se face în doi pași. Presupunem că ne ocupăm de o mulțime consistentă de entități $\varepsilon$.

Demonstrarea caracterizării sintactice a ordonării pentru termeni fundamentali:

**Lema 3.1.1** Pentru toți termenii t, t' și t" astfel încît $I_\varepsilon(t") \subseteq I_\varepsilon(t) \cup I_\varepsilon(t')$ avem fie $I_\varepsilon(t") \subseteq I_\varepsilon(t')$ fie $I_\varepsilon(t") \subseteq I_\varepsilon(t)$.

**Demonstrație lemă**: Cazul termenilor fundamentali este trivial. Se poate vedea ușor că t si t' au aceeași structură. Distingem următoarele două cazuri:

Termeni structurați ca tuplu:

Se presupune că t și t' au numai două atribute astfel încît t=[a:i, b:j] si t'=[a':i', b':j']. Fie t" tuplul care satisface lema, deci a este un atribut al lui t". Dacă nu, $I_\varepsilon(t")$ nu va fi inclus în $I_\varepsilon(t) \cup I_\varepsilon(t')$. Să presupunem că b' nu apare în t", atunci $I_\varepsilon(t") \cap I_\varepsilon(t')=\emptyset$ și $I_\varepsilon(t") \subseteq I_\varepsilon(t)$. Invers, să presupunem că b și b' sînt în t". Atunci t"=[a:i", b:j", b':k"]. Din ipoteza: $I_\varepsilon(term(i")) \subseteq I_\varepsilon(term(i)) \cup I_\varepsilon(term(i'))$. Aplicînd recursiv lema obținem, de exemplu,

$$I_\varepsilon(term(i")) \subseteq I_\varepsilon(term(i))$$

Presupunem că:

$$I_\varepsilon(term(i")) \not\subseteq I_\varepsilon(term(i'))$$

atunci:

$$I_\varepsilon(term(j")) \subseteq I_\varepsilon(term(j))$$

Dacă nu, se poate produce un identificator x astfel încît termenul [a:x, b:x, b':z] este în $I_\varepsilon(t")$, dar nu este în $I_\varepsilon(t')$, nici în $I_\varepsilon(t)$. Rezultă deci, $I_\varepsilon(t") \subseteq I_\varepsilon(t)$.

Termenii structurați ca set:

Presupunem că $t=\{s_1,...,s_r\}$, $t'=\{s_1',...,s_q'\}$ și $t"=\{s_1",...,s_p"\}$ avem pentru toți i $s_i" \le s_j$ sau $s_i" \le s_j'$. Se mai presupune că există un element $s_i"$ astfel încît $s_i" \le s_j$ și un element $s_j"$ astfel încît $s_j" \le s_l'$. Dar $s_i" \not\le s_k'$ pentru toți k și $s_j" \not\le s_l$ pentru toți l. Atunci fie x în $I_\varepsilon(term(s_i"))$ și y în $I_\varepsilon(term(s_j"))$. Rezultă că {x, y} este în $I_\varepsilon(\{s_i", s_j"\})$ și deci și în $I_\varepsilon(t")$ Pe de altă parte {x, y} nu este nici în $I_\varepsilon(t)$., nici în $I_\varepsilon(t')$. Am obținut o contradicție. În consecință avem, de exemplu, $s_i" \le s_j$ pentru toți i și $I_\varepsilon(t") \subseteq I_\varepsilon(t)$. q.e.d.

Folosind lema anterioară vom demonstra teorema 3.1.

Cazul termenilor fundamentali este trivial.

Termenii structurați ca tuplu:

"$\Rightarrow$" este trivială

Dacă $t \le t'$, atunci din definiția interpretării rezultă că orice atribut în t' este în t. Dacă nu, se poate găsi usor un termen structurat ca tuplu care este consistent cu $I_\varepsilon(t) \subseteq I_\varepsilon(t')$. Fie un atribut al lui t' definit pe $e_i'$; atunci a apare în t și este definit pe $e_i$.

Dacă, $term(e_i) \not\le term(e_i')$ atunci se poate obține o entitate care este în $I_\varepsilon(e_i)$ și nu este în $I_\varepsilon(e_i')$. Pentru aceasta se poate produce o entitate structurată ca tuplu care va fi în $I_\varepsilon(t)$ și nu în $I_\varepsilon(t')$.

Termenii structurați ca set:

"$\Rightarrow$" este trivială

Fie $\{e_1,...,e_n\} \le \{e_1',...,e_p'\}$; atunci, conform definiției interpretării termenilor structurați ca set, avem

$$I_\varepsilon(e_1) \cup ... \cup I_\varepsilon(e_n) \subseteq \cup I_\varepsilon(e_1') \cup ... \cup I_\varepsilon(e_p')$$

Deci pentru toți i,

$$I_\varepsilon(e_1) \subseteq \bigcup_j I_\varepsilon(e_j')$$

și conform lemei, rezultă:

$$I_\varepsilon(e_i) \subseteq I_\varepsilon(e_j') \qquad \text{q.e.d.}$$

Teorema 3.1 arată că relația de rafinare pentru tuple este similară celei din [17]. Următoarea teoremă este o extindere la cazul termenilor generici. Ca și în cazul definiției rafinării, teorema 3.2 subsumează teorema 3.1, care a fost dată pentru mai multă claritate. Vom folosi următoarea notație: dacă v și w sînt doi termeni definiți pe $\varepsilon$, $v <_\theta w$ înseamnă că pentru toate evaluările $\sigma$ și $\sigma'$ corespunzînd pe $\theta$, definit ca în 2.5,avem $v.\ \sigma \le v.\ \sigma'$.

**Teorema 3.2** Fie $\varepsilon$ o mulțime consistentă de entități. Fie t și t' doi termeni definiți pe $\varepsilon$. Avem $t \le t'$ dacă și numai dacă există o mapare $\theta$, ca în definiția 2.5, astfel încît $t \le_\theta t'$. Mai mult, fie $\theta$ o astfel de mapare. Avem $t \le_\theta t'$ dacă și numai dacă este adevărat unul din cazurile:

termeni structurați ca set:

$t=\{e_1,...,e_n\}$, $t'=\{e_1',...,e_p'\}$ și pentru orice e în term(t):

- ori e este o variabilă $x_k$ și există o variabilă $e'=y_l$ în term(t') astfel încît $\theta(l)=k$,
- ori e este un identificator $i(x_1,..,x_p)$ și există $e'=j(y_1,..,y_q)$ în term(t') astfel încît $term(e) \le_\theta term(e')$.

Termeni structurați ca tuplu:

pentru orice cîmp a de valoare e din t, există un cîmp a de valoare e' în t' astfel încît

- ori e este o variabilă $x_k$ si e' este o variabilă $y_l$ astfel încît $\theta(l)=k$,
- ori e este un identificator $i(x_1,..,x_p)$, e' este un identificator $j(y_1,..,y_p)$, astfel încît $term(e) \leq_\theta term(e')$.

**Demonstrație**: Începem cu "⇒". Demonstrația acestei implicații se poate face prin inducție. Echivalența $t \leq t'$ dacă și numai dacă există o mapare, θ astfel încît $t \leq_\theta t'$ devine evidentă pornind de la definiția lui ≤. Echivalența este numai o definiție pentru $\leq_\theta$. Fie ε o mulțime consistentă de entități. Fie t și t' doi termeni definiți pe ε si θ o mapare ca în definiția 2.5. Se pot defini două substituții astfel:

- pentru orice variabilă x în t se construiește o entitate fundamentală $(i_x,T_x)$ și se defineste $\sigma_0(x)=i_x$ ;
- pentru orice variabilă x' în t' astfel încît $\theta(x')$ este definit, se definește $\sigma'_0(x') = \sigma_0(\theta(x))$;
- pentru celelalte variabile x' în t' se construiește o entitate fundamentală $[i'_{x'},T'_{x'}]$ și se definește $\sigma'_0(x) = i'_x$.

Se presupune că termenii fundamentali $T_x$ sînt perechi necomparabile. Se pot construi termeni folosind, de exemplu, termeni tuplu fundamentali avînd fiecare un cîmp distinct. Se observă că aceste substituții sînt substituții valide pentru t și t' care respectă θ. Acestea vor fi folosite în următoarea lemă:

**Lema 3.1.2** Dacă v și v' sînt doi termeni care implică unele dintre variabilele din t, respectiv t', atunci:

$$v \cdot \sigma_0 \leq v' \cdot \sigma'_0 \Rightarrow v \leq_\theta v'$$

**Demonstrație lemă**: Se va face demonstrația pentru termeni structurați ca set. Demonstrația pentru termeni tuplu și de bază este similară. Dacă v este un termen structurat ca set și $v=\{..,x,..\}$, atunci există o componentă s' a lui v' astfel încît $\sigma_0(x) \leq \sigma'_0(s')$. Termenii $T_x$ sînt perechi necomparabile, deci s' este θ(x). Dacă i(...) este un identificator valoric în v, atunci există o componentă s' a lui v' astfel încît:

$$term(i(...)) \cdot \sigma_0 \leq term(s').\sigma'_0$$

Folosind lema pentru term(i(...)) și term(s') rezultă:

$$term(i(...)) \leq_\theta term(s')$$

:eea ce termină demonstrația inductivă a lemei.

q.e.d.

Acum sîntem pregătiți să demonstrăm partea ›rincipală a teoremei. Fie t și t' doi termeni. resupunem că avem $t \leq_\theta t'$, conform definiției lui $t \leq \theta$:

$$\forall\sigma,\ \forall\sigma',\ t.\sigma \leq t'.\sigma'.$$

Cum $t.\sigma$ si $t'.\sigma'$ sînt termeni fundamentali se poate losi lema 3.1.1. Conform structurii lui t si t', se vor stinge trei cazuri:

:meni structurați ca set:

dacă $t=\{s_1,...,s_n\}$ si $t'=\{s_1',...,s_n'\}$, atunci, conform teoremei 3.1 pentru toți σ și σ' pentru toți $s_i$ în t, există $s_j'$ în t' astfel încît:

$$term\ (s_i\ .\ \sigma) \leq term(s'_j\ .\ \sigma')$$

Aplicînd acest rezultat la $\sigma_0$ si $\sigma'_0$, se va obține ca pentru toți $s_i$ în t, există $s_j'$ în t' astfel încît

$$term\ (s_i\ .\ \sigma_0) \leq term(s'_j\ .\ \sigma'_0)$$

Folosind acum lema 3.1.2, rezultă că pentru toți $s_i$ în t, există $s_j'$ în t' astfel încît:

$$term\ (s_i) \leq_\theta term\ (s'_j)$$

Prin urmare:

- ori $s_i$ este o variabilă și $s_j'$ este o variabilă astfel încît $\theta(s'_j) = s_i$;
- ori $s_i$ este un identificator valoric și prin definiția lui $\leq_\theta$, $term(s_i) \leq_\theta term(s_j')$.

Demonstrația pentru termeni tuplu și de bază se poate face la fel, folosind lema 3.1.2 si caracterizarea termenilor fundamentali.

q.e.d.

Să încercăm să aplicăm această teoremă la următoarea rafinare:

{set(x), clădire} ≤ {construcție, set(y)}

Se consideră o variabilă de mapare θ astfel încît θ (x)=y și, conform teoremei 3.2, se știe că rafinarea se păstrează dacă și numai dacă:

1. {x} ≤ {y} (desi term(set(x))={x}) si
2. [nume:<u>string</u>, vechime:<u>integer</u>, proprietar:<u>construcție</u>, suprafața:<u>integer</u>] ≤ [nume:<u>string</u>, vechime:<u>integer</u>, proprietar:<u>construcție</u>]

Se mai folosește o dată teorema 3.2 pe cele două rafinări de mai sus. Prima este trivială, cu o mapare a lui y pe x. Ultima rafinare are loc dacă și numai dacă:

1. **string** ≤ **string** și
2. **integer** ≤ **integer** și
3. term(<u>construcție</u>) ≤ term(<u>construcție</u>)

Toate aceste instrucțiuni sînt evident adevărate și se poate spune că toate rafinările se păstrează [18].

Această teoremă conduce la o caracterizare sintactică a rafinărilor. Singura dificultate în construirea unui algoritm de testare apare în cazul entităților cu auto-referire. Acest algoritm ar trebui să includă un mecanism de detecție a ciclării.

## 4. Concluzii

Acest articol a prezentat un model de date orientate obiect, care încorporează construcții de mulțimi și tuple, identitatea obiectelor împreună cu moștenire și genericitate și cîteva exemple simple din domeniile de aplicabilitate ale

GIS. Trăsăturile principale ale acestui model sînt: tratarea uniformă a tipurilor şi a obiectelor, construcţia tipurilor generice, o semantică a incluziunii mulţimilor pentru relaţiile de moştenire şi moştenire multiplă.

Nu se face distincţie între obiecte şi structuri de tip. Se defineşte un cadru uniform în care aceste concepte sînt văzute ca entităţi construite din valori simple şi constructori de set şi de tuplu. Termenii sînt interpretaţi folosind mulţimi consistente de entităţi. Mai mult, este permisă folosirea variabilelor în construirea entităţilor pentru a modela genericitatea. Datorită uniformităţii acestui model, relaţia de rafinare asupra entităţilor (aşa numita moştenire link) este aplicată, atît entităţilor generice, cît şi entităţilor fundamentale.

Acest model înglobează toate stadiile, de la entităţile fundamentale, ca "1" sau "[nume:cladire, vechime:40]", la entităţile generice ca "[nume:x, vechime:y]", incluzînd entităţi ca "[nume:string, vechime:integer]" sau entităţi parţial instanţiate ca "[nume:string, vechime:40]". Semantica entităţilor structurate ca tuplu nu se foloseşte în lumea bazelor de date, dar urmăreşte propunerea din [14], care dă o semantică simplă a incluziunii mulţimilor pentru relaţia de rafinare între entităţile structurate ca tuplu. Mai mult, această relaţie de rafinare modelează moştenirea multiplă. Acest model este o extensie a lucrării [15] şi conduce la un sistem formal pentru limbajele orientate obiect generice.

În acest articol s-au considerat doar aspectele structurale ale tipurilor. În paradigma Orientării Obiect, tipurile încapsuleaza atît structura cît şi comportamentul obiectelor. Aceste rezultate pot fi extinse uşor în maniera din [16] pentru a lua în considerare aspectele comportamentale ale tipurilor. În [6], funcţiile (metode în terminologia obiectelor) nu sînt obiecte ale modelului. Se intenţionează extinderea lucrării prin adăugarea unui constructor de metode, ceea ce va permite considerarea funcţiilor ca entităţi şi implementarea în MDL (Modular Development Language de la MICROSTATION-INTERGRAPH [20]).

## Bibliografie


1. **MANFRED, E., EDWARDS, G., BEDARD, Y.**: Integration of remote sensing with GIS: a necessary evolution, PE&RS, nr.11, 1989.
2. **EGENHOFER, M.J., FRANK, A.U.**: Object-Oriented software engineering considerations for future GIS. In: Sec. Int. GIS Symposium, Baltimore, 1989.
3. **GONG, J.**: Object-Oriented models for thematic data management in GIS. In: EGIS '90, 1990, pp. 494-503.
4. **WEGNER, P.**: The Object-Oriented Classificat Paradigm, MIT Press Cambridge, 1987.
5. **GOLDBERG, A., ROBSON, D.**: Smalltalk 80, Language and Implementation, Addison-Wesl 1983.
6. **LECLUSE, C., RICHARD, P.**: $O_2$, an Obj Oriented Data Model, TR 1987. In: Proc. of ACM-SIGMOD, Chicago, 1988.
7. **MAIER, D., OTIS, A., PURDY, P.**: Developm of an Object Oriented DBMS, IEEE, Special Iss on Object Oriented Systems, vol 8:4, 1985.
8. **BRUCE, K.B.**: An Algebraic Model of Subtypes Object Oriented Language, SIGPLAN Notices, v 21:40, 1986.
9. **CARDELLI, L., WAGNER, P.**: On Understandi Types, Data Abstraction and Polymorphism, AC Computing Surveys, vol.17:40, 1985.
10. **MILNER, R.**: A Theory of Type Polymorphism Programming Languages, JCSS, vol.17, 1978.
11. **ABITEBOUL, S., BEERI, C.**: On the Power Language for Manipulating Complex Object. In: I Workshop on Theory and Applications of Nes Relations and Complex Objects, Darmstadt, 1987.
12. **PISTOR, P.**: A Database Language for Sets, Li and Tools, IBM Wiss. Zents Heidelberg, 85/10/004,1985.
13. **SCHEK, H.**: A Basic Relational NF2 Algeb Processor. In: Proc. of the Int. Conf. on Pound. Data Org. Kyoto, Japon, mai 1985, pp.173-182.
14. **BANCILBON, F. ş.a.**: FAD a Powerful and Sim Database Language. In: Proc. of the 13th Conf. VLDB, Brighton, 1987.
15. **AIT-KACI, H., NASR, R.**: LOGIN: A Lo Programming Language with Built-in Inheritan Journal of Logic Programming, 1986.
16. **MEYER, B.**: Genericity versus Inheritan OOPSLA, Portland, Oregon, 1986.
17. **CARDELLI, L.**: A Semantics of multi Inheritance, Semantics of Data Types, Lecture No in Computer Science, 1984.
18. **LECLUSE, C., RICHARD, P.**: Modelli Inheritance and Genericity in Object Orient Databases, TR nr. 18-88, 1988.
19. **GRAY, K.G., KULKA, N.W., Paton**: Obje Oriented Databases Englewood Cliffs A seman Data Model Approach, Prentice Hall, New Jers 1992.
20. * * *: MICROSTATION-MDL, INTERGRAP 1992 (în dotarea lab. GeMaSOFT-ICI).