Software cu componente pentru sisteme de control în depozite[1]

Prof.dr. Hans CZAP,
Universitatea Trier, Germania

Dezvoltarea sistemelor software în acord cu necesitatile specifice ale clientilor poate fi substantial simplificata prin utilizarea unor componente realizate si standardizate anterior. Problema prinicipala a acestui asa-numit software cu componente (component software) consta într-o reprezentare adecvata a interfetei sistemului.
Respectând cele mai tehnice aspecte ale unui sistem de control într-un depozit (SCS – Store Control System), vom arata cum în sistemul de software cu componente SCS-Composer considerat, situatia aplicatiei a fost modelata cu scopul de a construi interfata specifica sistemului. De asemenea, vom arata cum s-a realizat accesul la componentele software.
Cuvinte cheie: *sisteme software, componente software, sisteme de control.*

1. Introducere

Micile companii producatoare de software pot avea succes numai daca opereaza pe o piata specializata, ori daca au suficienta flexibilitate pentru a dezvolta aplicatii specifice clientilor. Instrumentul de utilizare a componentelor mentionat, SCS-Composer, suporta dezvoltarea aplicatiilor specifice clientilor, atât timp cât ele au aceeasi functionalitate de baza.
Ca exemplu tipic poate fi considerat un sistem SCS. Acest tip de sisteme controleaza aspectele cele mai tehnice ale miscarii bunurilor într-un depozit. Subfunctii tipice constau din puncte de livrare, selectia unitatilor de transport necesare, posibila reîmpachetare, determinarea pozitiei de depozitare, depozitarea etc. si în final furnizarea spre centrele de distributie.
In continuare vom prezenta caracteristicile tipice ale SCS cu scopul de a ilustra variabilitatea conditiilor de utilizare a acestora. Paragrafele urmatoare arata arhitectura interfetei sistem a SCS-Composer si se concentreaza asupra structurii obiectelor si a modului cum se dezvolta o aplicatie folosind SCS-Composer.

2. Caracteristici ale SCS

Functionalitatea de baza a tuturor SCS este comparabila, la nivelul caracteristicilor principale si macro-proceselor. De exemplu, în toate cazurile se gasesc functii ca puncte de livrare, controlul calitatii, depozitare s.a.m.d. Dar privind lucrurile mai apropiat de situatia specifica a clientului, apar suficiente aspecte diferite, ceea ce conduce la SCS complet distincte.
Diferentierea bunurilor se produce în functie de marime si tipul materialului (articole de dimensiuni foarte mari, mari si mici, baloturi, butoaie, bidoane, sticle, bunuri în vrac), de temperatura de depozitare (locuri cu temperatura normala, locuri racoroase, congelatoare), pericole (pericol de explozie, toxicitate etc.), serii de productie si durata.
În plus apar diferentieri în functie de rafturi, caracteristicile lor tehnice, amplasarea si daca depozitarea este automatizata sau

[1] Versiune în limba româna, realizata de prof.dr. Constantin Apostol, a lucrarii**: Component Software for Store Control Systems.** Lucrarea originala a fost primita la redactie în luna mai 1999 si revizuita în octombrie 1999.

manuala. De asemenea, vehiculele folosite pentru transport si containerele folosite pentru depozitare si transport difera de la caz la caz (pentru detalii, de comparat Jünemann (1989), Grochla (1978), Rupper (1988) si Rupper-Scheuchzer (1988)).
Multe diferente apar, de asemenea, în raport de functionalitate. Asigurarea calitatii ne va servi ca exemplu.
Pentru realizarea controlului calitatii, în anumite cazuri este necesara prelevarea si deplasarea unei probe. In cazul ca acest control de calitate ia ceva timp, bunurile ramase sunt etichetate ca blocate si plasate în depozit. In alte cazuri întreaga livrare este depozitata dupa etichetare si proba pentru realizarea controlului calitatii va fi luata din depozit într-o alta perioada. Depinzând de rezultatul controlului calitatii, se produce deblocarea sau întreaga livrare va fi trimisa înapoi. Deoarece controlul calitatii poate fi distructiv sau nu proba însasi necesita un tratament special.
Asa cum arata acest exemplu, anumite functii ale unui SCS nu sunt independente de altele. Functia '*pune în depozit*' se aplica în conditiile determinate de diversele variante rezultate din '*controlul calitatii*'. Lucrurile stau la fel pentru functia '*extrage din depozit*'. De asemenea, în diferite versiuni pot exista primitive ale unui SCS ca functia '*gaseste o locatie în depozit pentru a extrage bunuri*'. In general, locatia în depozit va fi aleasa astfel încât sa se minimizeze deplasarile vehiculului de transport intern. In cazul unor bunuri perisabile sau al unor produse chimice sau farmaceutice din aceeasi serie de productie data de expirare va fi caracteristica determinanta.
Aceste interdependente între functiile unui SCS contribuie esential la cresterea complexitatii. Interdependentele si un volum mai mic al pietei de desfacere a produsului software împiedica dezvoltarea unei solutii standardizate care sa poata fi personalizata prin ajustarea parametrilor. De asemenea, acest segment de activitate nu este foarte sensibil la pret. Costurile unui SCS se înscriu între 300 si 500 de mii de DEM în timp ce costurile întregii investitii într-un sistem de depozitare se exprima în zeci de milioane de DEM. În concluzie, pot functiona relativ usor firme de software care sa faca dezvoltari de solutii individuale.

3. Bazele de cunostinte ale SCS-Composer: Modelul de referinta *'depozit '*

3.1. Structuri de baza: Obiecte, relatiile *'parte a'* si *'varianta a'*

Baza de cunostinte a SCS-Composer este modelata printr-un graf conceptual ierarhic (Sowa(1984)), ale carui noduri constau din *obiecte* iar muchiile din relatii *'parte d* (figura 1) si *'varianta a'* (figura 2).

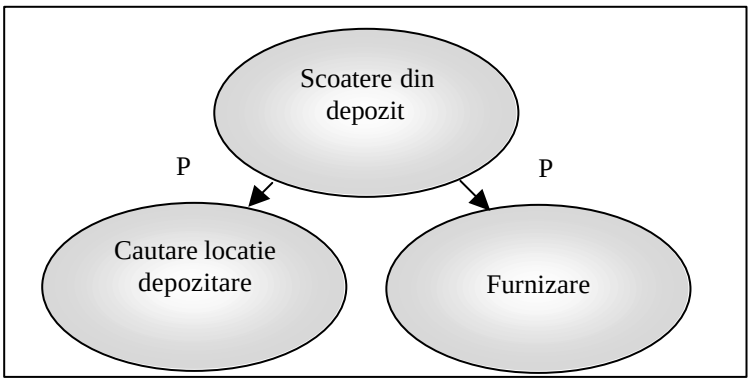

Fig.1. Relatia *'parte a'*

În plus, baza de cunostinte consta din *reguli de productie*:

- *Obiecte* reprezinta unitati organizationale, concepte sau procese.

▪ Relatia *'parte of'* reprezinta agregarea, adica un nod-parinte este o agregare a nodurilor-fiu asociate. Deci, în acord cu inferenta, existenta nodului-parinte implica existenta tuturor nodurilor-fiu.

▪ Relatia *'varianta a'* reprezinta specializarea, adica instante alternative ale nodului parinte. În acord cu inferenta, existenta nodului-parinte implica existenta a cel putin unuia dintre nodurile-fiu.

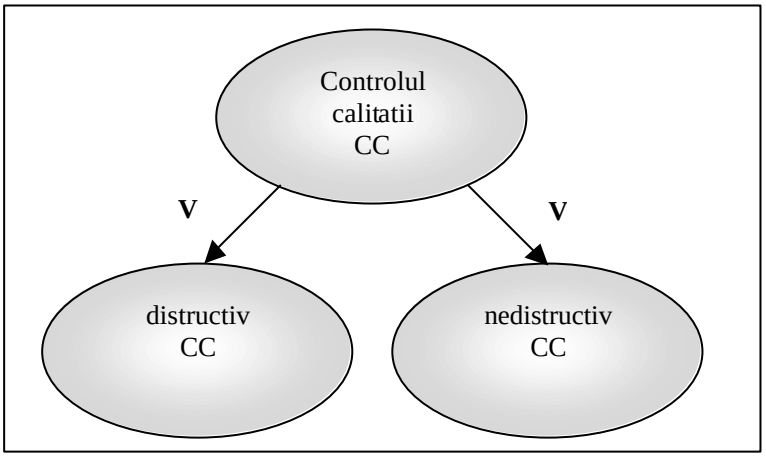

Fig.2. Relatia *'varianta a'*

Aspectele structurale ale modelului de referinta *'depozit'* sunt descrise prin relatiile *'parte a'* si *'varianta a'*.

▪ Regulile de productie sunt folosite pentru a reprezenta relatiile neierarhice dintre obiecte. Exemple:

IF întreprindere de productie (ca 'varianta a' obiectului întreprindere) AND depozit produse finite (ca 'varianta a' obiectului depozit) THEN livrare de bunuri din productie (ca 'varianta a' obiectului livrare de bunuri)

Regulile de productie exprima domenii de cunoastere generalizate. Daca în timpul procesului de modelare a unei întreprinderi concrete sunt înregistrate obiectele *'întreprindere de productie'* si *'depozit de produse finite'*, existenta obiectului (respectiv procesului) *'livrare de bunuri din productie'* va fi inferentiata. În plus, vor fi folosite reguli pentru a asigura consistenta modelului structural cu modelul procesului.

Expresii de tipul *'if A then B or C '* pot fi modelate folosind relatia *'varianta a'* considerându-l pe A ca nod-parinte si pe B si C ca noduri-fiu.

3.2 Arhitectura SCS-Composer

Structurile de baza identificate în ultimul paragraf sunt combinate cu scopul de a reprezenta interfata sistemului, respectiv de a stabili baza de cunostinte a SCS-Composer. La nivelul superior (figura 3) SCS-Composer are partile *'model referinta'* si *'obiecte'* (cu sensul de *obiecte de programare*). Modelul de referinta consta din cunostinte specifice domeniului, necesare pentru a reprezenta aspectele structurale ale organizatiei-depozit (*structura*) si caracteristici legate de proces pentru a modela fluxurile de activitati (*model proces*).

Submodelul *'caracteristicile întreprinderii'* permite diferentierea întreprinderilor comerciale de întreprinderile de productie ca si de diferite canale de distributie si permite formularea unor reguli de productie importante: ***if*** *'întreprindere comerciala'* ***then not*** *'depozit de produse neterminate'* la fel ca si ***if not*** *'depozit de produse neterminate'* ***then*** *'livrare a bunurilor din productie'*.

De asemenea, regulile de productie leaga adesea aspecte statice ale modelului struc-

tural *'structura'* de cele procedurale ale modelului procesului: ***if*** *<tip definit de material>* ***then*** *'control calitate distructiv'.*

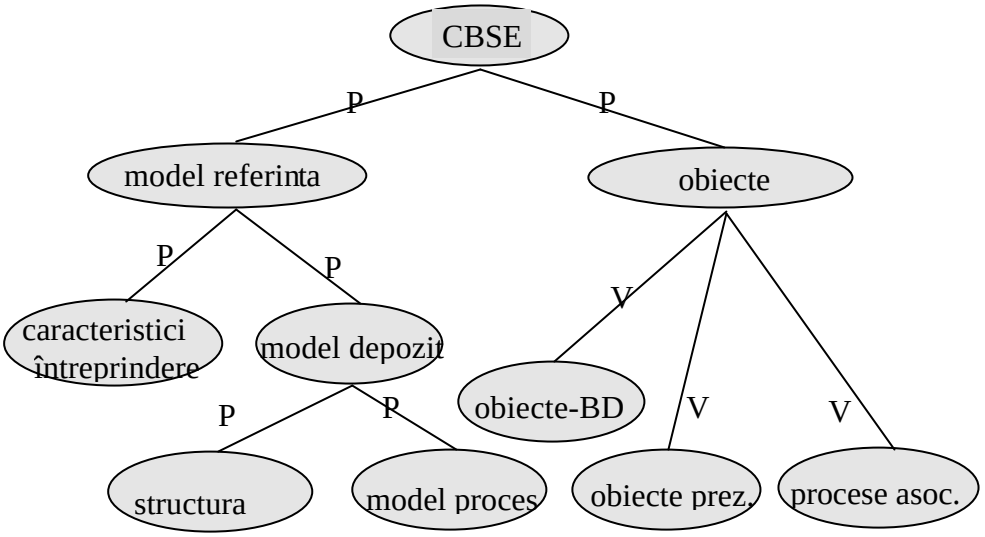

Fig.3 Structura principala a SCS-Composer

4. Obiecte

SCS-Composer permite dezvoltarea aplicatiilor administrând si memorând obiecte separate, selectându-le pe cele potrivite în functie de necesitatile determinate ale utilizatorului si realizând un control al consistentei.

4.1. Tipuri

Ca si obiectele oricarui limbaj de programare orientata obiect, cele ale SCS-Composer constau din metode si date. In plus, obiectele baze de date ('*obiecte BD*' în figura 3), adica proceduri memorate sau trigger-e, ca si obiectele de prezentare orientate spre utilizator ('*obiecte prez.'* în figura 3), sunt dezvoltate printr-o sectiune ce înregistreaza evenimentele ce se traduc într-un apel de metoda.

O metoda a unui obiect SCS-Composer poate sa cheme o metoda a altui obiect SCS-Composer. Pentru a administra aceste referinte, fiecare obiect este identificat printr-o eticheta a clientului si numele functiei. Numele functiei corespunde unui obiect al modelului de referinta.

Obiectele SCS-Composer sunt diferentiate în:

Obiecte baze de date pentru a reprezenta metode independente de context. De exemplu, daca stocul unui produs din depozit scade sub nivelul de siguranta se initiaza reaprovizionarea.

- *Obiecte de prezentare* pentru a realiza interfata cu utilizatorul.
- *Procese asociate* ('*procese asoc.'* în figura 3) care reprezinta fluxul concret al contolului în timplu executiei si realizeaza înlantuirea proceselor.

4.2. Structura obiectelor

Orice obiect SCS-Composer contine urmatoarele atribute, respectiv tabele:

- *Tip obiect* (asa cum este mentionat în paragraful 4.1)
- *Data creare*
- *Eticheta client/proiect.* Obiectele cu functionalitate similara difera în functie de eticheta clientului/proiectului. Deoarece aceasta eticheta este parte a cheii obiectului, pot fi create anumite proiecte implicite '$standard_1$',.., '$standard_n$' cu scopul de a memora diferite versiuni ale unui obiect, care nu este legat de un anumit client, respectiv proiect.

- *Nume obiect.* Exista doua versiuni ale numelui oricarui obiect. *Versiunea extinsa* contine un sir arbitrar de pâna la 50 de caractere si va fi aleasa cât mai descriptiva cu putinta. *Numele scurt* consta din exact 8 caractere continând indicative (fiecare de doi baiti) pentru *grupul de functii* caruia îi apartine obiectul, *functia* în cadrul grupului, *tipul obiectului* (ca în 4.1) si în final un *numar de ordine* de doi baiti. Acest nume scurt este folosit cu rol de cheie primara pentru a referi obiectul în proiectul curent. Combinatia (*eticheta client/proiect, nume scurt*) serveste la identificare în cadrul SCS-Composer.
- *Pseudo-codul obiectului.* Acesta este un câmp lung descriind codul obiectului. Uzual, înaintea folosirii într-un nou proiect orice obiect necesita anumite ajustari, care trebuie facute de programatori. Deoarece ca mediu de lucru pentru programare este folosit un limbaj de generatia a 4-a (4GL), etapa trecerii de la pseudo-cod la codul real de programare nu a fost considerata esentiala.
- *Tabela evenimente-actiuni.* Aceasta tabela leaga evenimentele posibile de metodele care sunt executate în cazul aparitiei evenimentelor. Evenimentele posibile în cazul unor *obiecte de prezentare* sunt clicuri cu mouse-ul, apasari de taste etc., iar în cazul *obiectelor baze de date* acestea sunt evenimente-trigger.

Actiunea de realizat poate fi o metoda a obiectului în cauza. In acest scop, orice metoda consta din *numele metodei* (de pâna la 8 caractere), cu care este identificata în cadrul obiectului si un *câmp lung* pentru a defini metoda (în pseudo-cod). De asemenea, actiunea referita de *tabela evenimente-actiuni* poate fi o metoda a unui obiect diferit. In acest caz, referinta consta din *numele scurt al obiectului* si *numele metodei*.

4.3. Conectivitatea obiectelor

Interdependentele diferitelor obiecte sunt reprezentate prin doua mecanisme diferite. Asa cum se descrie în sectiunea mai generala anterioara, orice obiect corespunde unui nod în graful structural. Astfel, relatia structurala parinte-fiu poate fi de tipul *'varianta a'* sau de tipul *'parte a'*.

Relatia *'varianta a'* permite referirea simpla a obiectelor cu o functionalitate asemanatoare. Relatia *'parte a'*.dintre obiecte în cadrul SCS-Composer este interpretata prin bine-cunoscutul mecanism al *mostenirii* din programarea orientata obiect, adica obiectul-fiu poate referi orice metoda a obiectului-parinte.

Pe lânga aceasta structura de graf, în interiorul oricarui obiect *tabela evenimente-actiuni* permite apelul metodelor apartinând unor obiecte diferite. In acest mod sunt realizate legaturi arbitrare între diferite obiecte.

Componenete ale SCS-Composer

O componenta în SCS-Composer corespunde unui obiect, care este memorat ca o unitate separata. Deoarece obiectele sunt identificate prin (*eticheta client/proiect, nume functie*), unde numele functiei se refera la un obiect al *modelului de referinta*, exista o relatie 1:n între obiectele modelului de referinta si obiectele de implementare specifice *programului depozit* (=*componenta depozit*).

Intr-adevar, modelul de referinta actioneaza ca un tezaur, descriind conceptul oricaruia dintre obiectele sale prin calea catre obiect. Deoarece, în general, exista mai mult de o implementare a unui obiect al modelului de referinta, pentru diverse scopuri sunt memorate cuvinte-cheie aditionale. Legaturile dintre cuvintele-cheie si obiecte sunt memorate într-o tabela separata.

6. Lucrul cu SCS-Composer

Realizarea unui sistem de control în depozit SCS într-un context dat presupune înregistrarea particularitatilor clientului. Pentru a realiza aceasta, modelul de referinta serveste ca o lista de control, unde

fiecare element trebuie etichetat cu ADEVARAT sau FALS.
In acest mod sunt înregistrate caracteristicile bunurilor, tehnologia de depozitare si detaliile procedurale dorite. Modelul general de referinta este croit dupa *'modelul client'*.
Procesul de adaptare este realizat printr-un mecanism de inferenta care opereaza asupra unei tabele simple având ca atribute *(posibile) fapte* si *asertiuni*. *(Posibilele) fapte* se coreleaza printr-o relatie 1:1 cu fiecare nod al modelului de referinta,iar *asertiunile* stabilesc pur si simplu daca fiecare *fapt* este *adevarat*, *fals* sau înca *necunoscut*. Initial fiecare fapt are valoarea *necunoscut*.
In timpul înregistrarii situatiei clientului, orice *fapt necunoscut*, adica nodurile modelului de referinta, sunt marcate cu *adevarat* sau *fals*. Mecanismul de inferenta foloseste dependentele modelului de referinta si regulile de productie pentru a simplifica sarcina de a raspunde pentru toate faptele necunoscute:

- Regulile de productie mentionate anterior permit sa se marcheze nodul corespunzator al modelului de referinta daca conditia este satisfacuta.
- Daca un obiect al modelului de referinta a fost marcat cu *adevarat*, toate nodurile situate pe calea de acces la el trebuie sa fie marcate cu *adevarat*.
- Daca relatia parinte-fiu este de tipul *'varianta a'* si nodul parinte a fost marcat cu *adevarat*, cel putin unul din fiii sai trebuie sa fie de asemenea *adevarat*.
- Daca relatia parinte-fiu este de tipul *'parte a'* si nodul parinte a fost marcat cu *adevarat*, toti fiii sai trebuie sa fie de asemenea *adevarat*.
- Analiza situatiei clientului poate adesea sa se concretizeze într-o descriere incompleta. Faptele absente pot genera cereri ulterioare costisitoare sau, chiar mai rau, necesitatea reconsiderarii software-ului livrat. Modelul de referinta al SCS-Composer implica înregistrarea completa a faptelor, adica orice *fapt* (si nodul marcat corespunzator) trebuie sa aiba *asertiunea adevarat* sau *fals*.

'Modelul client' consta din functionalitatea de baza a obiectelor necesare (obiecte de interfata) si interdependentele lor. Astfel se determina functia si structura aplicatiei clientului. Mai ramâne sa se selecteze pentru orice obiect de interfata cel mai adecvat obiect de implementare, adica *'obiect'* în terminologia SCS-Composer. Pentru realizarea acestei sarcini se va folosi lista proprietatilor tuturor acestor obiecte.Selectia finala si adaptarea la situatia specifica a clientului este efectuata manual prin inspecatrea codului.
Folosirea SCS-Composer în practica este similara cu ciclul de rationament bazat pe cazuri, asa cum este descris de Aamodt/ Plaza (1994). Acesti autori disting în acord cu cazurile fazele RETRIEVE, REUSE, REVISE si RETAIN. Numai faza RETAIN, adica extinderea bazei de cunostinte si a bazei de componente necesita unele explicatii. In situatia SCS-Composer particularitatile specifice clientului vor determina, în general, o descriere functionala a unui obiect si a implementarii ulterioare care trebuie înregistrata ca o 'componenta noua' în *programul înregistrat* (baza de componente). In cazuri mai simple acest obiect nou este o *'varianta a'* unui obiect existent. In cazurile mai complicate, trebuie adaugate ierarhii suplimentare *'modelului de referinta'* si *'modelului program'*.

7. Starea SCS-Composer

SCS-Composer este un prototip dezvoltat în strânsa coordonare cu o casa de software specializata în SCS. Dar situatiile de aplicare a SCS-Composer nu este limitata la sistemele de control în depozite. Una dintre caracteristicile de proiectare a fost de a realiza un instrument de dezvoltare de software independent de aplicatie. Pentru memorarea componentelor si a bazei de cunostinte SCS-Composer foloseste un

sistem de gestiune a bazelor de date (SGBD) relational standard, suplimentat de cod integral C/C++. In prezent, acesta este considerat un inconvenient important, deoarece un SGBD orientat obiect ar facilita în mod cert mostenirea si inferenta. Dar, în perioada de realizare a fazei de proiectare, nu a fost disponibil un SGBD orientat obiect de încredere.

Bibliografie

Aamodt A.; Plaza E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications, Vol. 7, Nr.1, 1994, p.39-59.

Czap, H.; Grimm, Ch.: SCS-Composer: Ein Fallbasiertes CASE-Tool zur Entwicklung kundenspezifischer Anwendungssysteme am Beispiel von Lagerführungssystemen. Wirtschaftsinformatik, Heft 1/96, 32-38.

Czap, H.: Case Based Software Engineering CBSE. The Example of a Store Control System. In: Classification and Knowledge Organization, Proceedings der 20. Jahrestagung der Ges. für Klassifikation, R. Klar und O. Opitz Hrsg., Reihe: Studies in Classification, Data Analysis and Knowledge Organization, H. H. Bock, O. Opitz, M. Schader Managing Editors, Berlin, Heidelberg u.a. 1997, S. 245 - 252.

Grochla, E.: Materialwirtschaft. Das materialwirtschaftliche Optimum im Betrieb. Wiesbaden 1978.

Jünemann, R.: Materialfluß und Logistik - Systemtechnische Grundlagen mit Praxisbeispielen. Berlin u.a. 1989.

Rupper, P.: Logistische Systeme. Köln, 1988.

Rupper, P.; Scheuchzer, R. (Hrsg.): Lager- und Transportlogistik. Planung, Steuerung und Kontrolle im Transport und Lagerbereich. Zürich 1988

Sowa, J. F.: Conceptual Structures. Information Processing in Mind and Machine. Reading Massachusetts, Menlo Park California u.a. 1984

Szyperski C.: Component Software, Beyond Object-Oriented Programming. ACM Press Books, Addison Wesley, Harlow Reading u.a., reprinted (twice) 1998.