Modelarea fiabilitatii software-ului orientat pe obiecte

Lect. Marian CRISTESCU
Universitatea „Lucian Blaga" Sibiu

Reliability is one of the most important features of critical software system that are becoming the foundations of human life and progress.In this paper foundations of the properties of object-oriented software measures are presented. The criteria for the properties of object-oriented software measures are characterized with several binary operations between objects, classes, methods etc.

Keywords: *Software reliability, software measurement,object-oriented software measures, metrics, measurement theory, model, concatenation operation.*

1 Introducere

Crearea de puncte de oprire implica salvarea unor informatii de stare despre executia unui program pe un mediu de stocare stabil, iar daca este necesar, programul poate fi relansat în executie începând de la starea înregistrata la punctul de oprire.

Punctele de oprire au fost folosite în mod traditional pentru îmbunatatirea timpului de executie pentru programele cu timpi de rulare îndelungati în prezenta erorilor. Schemele, mai sofisticate, de toleranta la erori care folosesc procesari repetate, contin adesea puncte de oprire pe care le folosesc pentru salvarea si transferarea informatiilor de stare a programului catre procesele de rezerva.

Atât programerea multidirectionala (**M**ulti-**T**hreading) cât si cea orientata pe obiecte (**OO**) au câstigat o popularitate imensa în practica software, si foarte multe aplicatii sunt construite folosind aceste metode. Toleranta la erori si performanta acestora sunt factori esentiali pentru organizatiile care le folosesc. Este usor de realizat extinderea bibliotecilor traditionale ale punctelor de oprire pentru ca acestea sa tina seama de mai multe directii în cadrul aceluiasi proces. Dar, o astfel de biblioteca a punctelor de oprire setata multidirectional (MT), care lucreaza la nivel de proces, trebuie sa salveze toate directiile la nivelul unui punct de verificare, si sa le restaureze pe toate la revenire. Se poate demonstra ca acest mecanism al punctelor de oprire limiteaza flexibilitatea si gradul de sofisticare a schemelor de toleranta la erori.

Pentru exemplificare se poate considera un sistem distribuit care foloseste pentru toleranta la erori algoritmul distribuit al punctelor de oprire si revenire. Ca urmare a comunicarii procesele sistemului devin interdependente. Daca unul dintre procese esueaza algoritmul determina care dintre procesele sistemului sunt dependente de procesul care a esuat, si acestea trebuie rulate din urma de la punctul de oprire anterior. Pentru a demonstra utilitatea obiectelor de tip puncte de oprire în cadrul tehnicii care urmareste stabilirea punctelor de oprire si revenire în programele de tip MT-OO se apeleaza la doua argumente:

- investigarea modului în care pentru grupuri izolate de obiecte si directii comunicante, dintr-un program aflat în executie, pot fi stabilite puncte de oprire pe grup (în mod selectiv);
- investigarea felului în care unele directii dintr-un program aflat în executie pot fi rulate înapoi, în mod selectiv, în timp ce altele continua sa fie executate, timp în care se mentine consistenta starii generale a programului.

În vederea demonstrarii modului cum pot fi construie nivele înalte ale schemelor de toleranta la erori se folosesc algoritmi selectivi, aratând astfel ca modelele conversationale, propuse initial pentru coordonarea procesului de recuperare dupa erori a entitatilor concurente care colaboreaza, pot fi implementate în vârful algoritmilor se-

lectivi. Rezultatele obtinute prin aplicarea acestor tehnici pot indica faptul ca aceste scheme selective functioneaza, de cele mai multe ori, mai bine decât tehnicile bazate pe "puncte de oprire pe nivele de procesare" si "schemele de recuperare" atunci când nivelele concurente ale procesului sau numarul erorilor este mare.

2. Modelul unui program OO

Bibliotecile pentru puncte de oprire existente presupun ca modelul proceselor este întotdeauna: *cod + date + multime + stiva.* În locul acestui model poate fi folosit un model mai sofisticat al programului.

Cei ce dezvolta software proiecteaza directiile de creare si terminare a unui program împreuna cu crearea si distrugerea obiectelor-date si cu interactiunea dintre obiecte si directii. În timpul executiei apar interactiuni si dependentele între directii si obiecte care sunt în concordanta cu modul de proiectare a programului. Cu acest model, informatia de stare a programului este împartita între obiecte, astfel daca fiecare obict stie cum sa-si testeze propria stare, atunci testarea unui astfel de program înseamna invocarea metodelor de testare a obiectelor si salvarea starii curente a directiilor. Punctele de testare fiind referite ca puncte de testare a obiectelor, ca atare ele înregistreaza mai mult starea programului decât starea processului.

Ipoteze

1. Toate datele din program sunt încapsulate de obiecte;
2. Aceste obiecte nu au date membre partajate.

Deci, fara nici o modificare, aceasta tehnica se poate aplica programelor care nu au date globale non-obiectuale sau nu urmaresc recuperarea datelor de felul acesta, daca ele exista.

Definitii

Consideram un proces care consta în N_0 obiecte si N_{th} directii.

- *Punct de testare globala*: punctul de testare care contine toate cele N_0 obiecte si N_{th} directii ale procesului.
- *Punct de testare selectiva*: un punct de testare format din punctele de testare ale unui grup de obiecte comunicante si directii ale procesului. Atunci când toate obiectele si directiile programului sunt incluse grupul 1, atunci punctul de testare selectiva este acelasi cu punctul de testare globala.
- *Informatia despre interactiunea directii-obiecte*: poate fi privita ca o multime de N_{th} perechi (TH_i, O_j) pentru orice $i \hat{I} (1,N_{th})$, $j \hat{I} (1,N_o)$; O_j este obiectul care a fost accesat prin TH_i atunci când informatia este generata.
- *Graficul dependentei obiectului*: un grafic nedirectionat al obiectelor din sistem, cu o latura unind oricare doua obiecte, daca cel putin unul dintre obiecte a invocat vreo metoda care apartine celuilaltx în intervalul de timp scurs de la ultimul punct de testare.
- *Setul de date de referinta* (ale unei directii): setul de obiecte care au fost referite pe o directie în intervalul de timp scurs de la ultimul punct de testare. Fiecare directie din sistem are propriul sau set de date; acest set de date poate intersecta cu altele. Intersectia reprezinta datele partajate între directii.
- *Setul de obiecte de revenire si directii*: set de obiecte si directii care este necesar a fi aduse înapoi la punctul de testare precedent în timpul procesului de reluare a executiei programului.

Cerinte de consistenta

În mod normal, mediile de programare multi-directionale nu ofera nici o garantie asupra executiei relative a directiilor, si astfel, nu exista instructiuni care sa poata fi considerate drept evenimente în executia diferitelor directii. Programatorii "controleaza executia" si "stabilesc o anumita ordine în program" prin adaugarea unor puncte de sincronizare ca si semafoare si a conditiilor variabile. Aceste puncte de sincronizare forteaza o ordine partiala a evenimentelor în executie pe doua sau mai multe directii. Pentru atingerea acestui deziderat directiile trebuie sa ajunga la un punct comun de sincronizare. Un punct

comun de sincronizare poate fi privit de asemenea, în acest model de program, ca un obiect comun în seturile de date de referinta. Prin urmare, în contextul modelului de program, punctele de sincronizare pot fi generalizate ca obiecte comune în seturile de date de referinta; deci:

a) Toate directiile, care participa la sincronizarea într-un anumit punct al executiei lor, au seturi de date care se intersecteaza (cel putin, obiectul de sincronizare este comun în seturile lor de date de referinta).

b) Directiile care nu participa la nici o sincronizare cu alte directii, nu au definita nici o ordine a actiunilor, în timpul executiei, care sa tina seama de actiunile altor directii. Aceste directii nu trebuie sa aiba seturi de date de referinta intersectate cu cele ale altor directii (daca lucreaza pe seturi de date distincte).

Algoritmul de revenire nu este necesar sa restaureze programul (directii si obiecte) la o stare (punct de revenire) despre care se stie ca a aparut în timpul executiei normale a programului; dar trebuie sa readuca programul într-o stare care respecta ordinea impusa de program si de secventa de executie individuala a directiilor. Aceasta stare este cunoscuta sub denumirea de *linie de recuperare*.

Fie TH1, TH2 doua directii oarecare, iar O_n un obiect al procesului. O *linie de recuperare* este un set de puncte de revenire pentru aceste directii si pentru obiect care satisfac urmatoarele trei cerinte:

1. Setul contine exact câte un punct de recuperare pentru fiecare directie si obiect implicat în revenire;
2. Daca programul indica faptul ca actiunea a_1 din executia lui TH1 *precede* actiunea a_2 din executia lui TH2, atunci linia de recuperare este stabilita astfel încât sa nu existe nici o actiune a_1 în executia lui TH1 dupa punctul de revenire al lui TH1, al carui corespondent *a2* din executia lui TH2 sa apara *înainte* de punctul de revenire al lui TH2.
3. Daca TH1 executa actiunile a_3 & a_4 aspra obiectului O_n astfel încât a_3 precede pe a_4, si punctul de revenire al Th1 *succede pe* a_3 dar *precede* pe a_4, atunci punctul de revenire al lui O_n este astfel plasat încât, oricare ar fi actiunile a_3 si a_4, starea obiectului în punctul de revenire reflecta efectele actiunii a_3 dar nu si ale actiunii a_4.

Desi nu exista nici o restrictie asupra felului în care punctele de testare ar trebui stabilite, natura punctelor de testare afecteaza algoritmul de revenire. Cerinta pentru un punct de testare ales selectiv este ca, odata ce un punct de testare este initiat dintr-o pereche obiect-directie, toate celelalte obiecte si directii a caror stare au o anumita ordine, definita prin perechea de origine, trebuie incluse în punctul de testare.

Puncte de testare si revenire alese selectiv

Pentru stabilirea unor astfel de puncte este necesar a fi urmarite urmatoarele aspecte:

- sistemul minim de executie pentru înregistrarea dependentei si a altor informatii despre executia programului;
- procedura pentru plasarea punctelor selective de testare utilizând informatia si punctele de reper pentru dezvoltarea algoritmului de recuperare;
- algoritmul selectiv de revenire, care este bazat pe punctele de reper.

Sistemul de executie

Alaturi de ipotezele modelului program, pentru sistemul de executie apar si alte ipoteze:

3. Toate datele din program sunt încapsulate de obiecte. Obiectele nu partajeaza datele membre comune, si nu exista nici o data non-obiect în program;
4. Fiecare obiect comunica doar prin invocarea metodelor celorlalte obiecte. Nu exista alta cale de a accesa date încapsulate de catre alt obiect.

Sistemul de executie retine urmatoarele date:

a) Graficul dependentei obiectelor: Acesta este un grafic indirect în care fiecare nod reprezinta un obiect. Un segment este adaugat între doua noduri când unul dintre ele invoca o metoda a celuilalt. Graficul trebuie refacut continuu pe durata executiei de

catre un mecanism de urmarire a dependentelor, care se activeaza ori de câte ori un obiect este accesat (de exemplu #4, aceasta înseamna un apel de metoda). Obiectele unite printr-un segment pot fi înlaturate, doar atunci când ambele sunt salvate într-un punct de testare.

b) Lista directiilor: Intrarile sunt adaugate si înlaturate din aceasta lista atunci când directiile se bifurca sau se termina. Aceasta lista este utilizata si pentru a colecta informatii despre interactiunile directie-obiect. Fiecare element al listei este o pereche de forma *(id directie, id obiect)*. Când aceste informatii sunt necesare, cu ajutorul mecanismului de detectare a dependentelor, fiecare directie activa analizeaza obiectul asupra caruia lucreaza si îsi completeaza intrarea din lista.

c) Simbolul creatiei: Fiecare obiect poarta cu sine un simbol al procesului de creatie, *viz*, descris prin numarul celui mai recent punct de testare plasat lânga locul unde obiectul a fost creat.

Informatia referitoare la interactiunea dintre directie si obiect poate fi cautata pentru a descoperi obiectele asociate cu directia data. Cum toate obiectele comunicante sunt conectate în grafic, o cautare în grafic (începând cu obiectul obtinut astfel) gaseste obiectele în setul de date de referinta ale directiei date. Obiectul care initiaza punctul de testare ales selectiv este numit obiect primar; directia asociata lui se numeste directie primara a punctului de testare ales selectiv. Dependentele dintre obiectele unui grup din interiorul graficului sunt resetate dupa ce graficului i s-au adaugat puncte de testare, iar monitorizarea dependentelor începe dupa aceasta, precum si procesul de executia continua dupa punctul de testare.

Procedura punctelor de testare are doua faze:

1. testarea obiectelor;
2. testarea directiilor.

Pentru salvarea unei stari cât mai consistente a obiectelor comunicante ale unui grup, este necesar sa nu se modifice nici un obiect atâta timp cât testarea grupului este în plina desfasurare. Acest lucru poate fi asigurat prin doua metode:

- blocarea: toate directiile se suspenda pe perioada primei parti;
- neblocarea: toate directiile se executa în aceasta etapa dar trebuie sa existe certitudinea ca nu va fi modificat nici un obiect care nu a fost modificat înca.

Pornind de la presupunerea ca se folosesc doar algoritmi cu blocare, prin compararea metodelor s-a ajuns la concluzia ca dezavantajele blocarii nu sunt exagerat de mari si pentru aceasta nu este necesara adaugarea unor metode mai sofisticate care sa utilizeze tehnica neblocarii. Directiile sunt blocate numai pe perioada în care se desfasoara testarea obiectelor în anumite puncte. De-a lungul perioadei în care directiile îsi salveaza starea nu este necesar sa se faca blocari, iar executia unor directii poate fi continuata în timp ce altele îsi salveaza starea. Din ratiuni similare, procedurile de revenire pot fi de asemenea blocante, sau neblocante.

3. Puncte de testare selective. Algoritmul punctelor de testare selective

Notatii utilizate în cadrul algoritmului: G - setul punctelor de testare a axului; O_p - obiect primar; O_n - un obiect al procesului; TH_p - directia primara a punctului de testare selectiv.

Etapa 1

1. Se initializeaza $G = TH_p$;
2. Se trimite un semnal CHECKPOINT catre toate celelalte directii. Asta face ca acestea sa "genereze informatia referitoare la interactiunea dintre directii si obiecte" si sa se "suspende pâna la aparitia unui semnal END_CHECKPOINT" (nici o directie nu salveaza informatie de stare, în acest moment);
3. Lansarea unei cautari în graficul dependentei obiectelor, începând de la punctul O_p. Pentru toate punctele O_n descoperite în timpul procesului de cautare:

a. Se apeleaza O_n.**checkpoint()**;

b. Se elimina toate marginile din graficul de dependenta a obiectelor care conecteaza

O_n cu alte obiecte care au fost deja incluse în punctul de testare;
c. Se verifica informatiile despre interactiunea dintre directii si obiecte, pentru a descoperi un TH_n asociat obiectului. Daca este gasit, atunci $G = G \cup TH_n$.
4. Se retine informatia despre interactiunea dintre directii si obiecte într-o mediu de stocare;
Sfârsitul *etapei* 1.
Etapa 2
5. Se trimite un semnal END_CHECKPOINT catre toate directiile. Dupa receptionarea acestui semnal, fiecare directie cauta *G*. Daca este inclus, atunci îsi salveaza propria stare în punctul de testare si îsi continua activitatea. Când toate directiile din *G* si-au salvat starea, testarea în acel punctul este terminata.
Sfârsit *etapa* 2.
Sfârsit algoritm.

4. Algoritmul de recuperare

Notatii utilizate: *R* - multimea obiectelor care participa la procesul de revenire; *G* - multimea directiilor care participa la procesul de revenire; C_i - punct de testare selectiv; S_i - multimea obiectelor si directiilor din C_i; R_i - multimea obiectelor care vor fi reântoarse la C_i; T_i - multimea directiilor care vor fi reântoarse la C_i; *F* - setul final de reveniri: toate R_i & T_i; O_s - obiect care necesita o revenie, simbolul procesului de creare este retinut de catre fiecare obiect; C_n - ultimul punct de testare selectiv luat în considerare.
1. Trimite un semnal RECOVER catre toate directiile active în cadrul procesului. Acest lucru le determina sa genereze informatiile referitoare la interactiunea directie-obiect si sa se suspende pâna la semnalul END_RECOVER;
2. Genereaza *R* prin lansarea unei cautari în graficul de dependenta a obiectelor, începând de la O_s. *F* este initial goala, iar procesul de revenire porneste de la cel mai recent punct de testare, $i=n$;
3. Atâta timp cât $R \neq \emptyset$ executa:
a. Pentru toate $O_n \in R$, *Daca* intervalul de timp alocat procesului de creare este mai mare decât *i, atunci* este distrus O_n si este înlaturat din *R*; *Sfârsit*;
b. Determina obiectele si directiile care vor fi reântoarse la C_i si le înlatura din *R*:
$R_i = R \cap S_i$, $T_i = T \cap S_i$
$R = R - R_i$, $T = T - T_i$
c. Reactualizeaza: $F = F \cup R_i \cup T_i$.
d. Salt la punctul de testare precedent: $i = i - 1$.
4. Ruleaza înapoi toate obiectele continute în *F*;
5. Trimite un semnal END_RECOVER catre toate directiile. Dupa acest semnal, fiecare directie verifica daca este inclusa în *F*. Daca da, aceasta se autoexecuta înapoi catre o stare gasita într-un punct de testare mentionat prin intermediul propriei sale intrari în *F*.
Sfârsit algoritm.
"Corectitudinea starii" rezultata dintr-un proces de revenire care utilizeaza algoritmul descris anterior presupune ca fiecare punct de testare selectiv formeaza o linie de recuperare consistenta pentru grupul de directii continut în el. Când mai multe puncte de testare sunt examinate în cautarea unei linii de recuperare consistente pentru *R*, grupurile care revin la puncte de testare diferite nu au în mod garantat dependente unul fata de celalalt. Astfel, nu va apare situatia în care un obiect revine la o stare care *precede* starea unui alt obiect dependent. Din acest motiv, nu pot fi facute reveniri în cascada.

5. Implementarea modelului conversational pentru procesele MT-OO

Algoritmii selectivi furnizeaza un mecanism simplu de puncte de testare, pe care pot fi construite alte scheme de toleranta la erori, de nivel mai înalt. În cele ce urmeaza este prezentata o metoda care adapteaza algoritmii selectivi pentru a implementa un model conversational pentru programele MT-OO si analizeaza eficienta schemelor selective folosind ca suport de testare un server de comert electronic.
Conform [RAN97], modelul conversational este un mecanism de structurare a software-ului foarte cunoscut care a fost dez-

voltat pentru simplificarea procesului de recuperare, în urma aparitiei defectelor, a coordonatelor proceselor concurente. O conversatie implica de obicei doua sau mai multe procese si constituie o *limita* pentru parametri *timp si spatiu bidimensional*, pe care entitatile care interactioneaza nu o pot depasi. Datorita limitarii activitatilor în anumite limite, este evitat efectul de domino în procesul de revenire. Înainte de începerea colaborarii, toate entitatile concurente care colaboreaza intra formal în conversatie. Intrarea în conversatie forteaza pe cel care intra sa treaca printr-un punct de testare. Încheierea conversatiei este un test de acceptanta. Testul poate fi local sau global, si trebuie sa fie trecut de catre toti participantii. Daca oricare dintre participanti nu reuseste sa treaca testul, este declansata o conversatie de revenire, care readuce toti participantii în punctele în care se aflau în momentul când au intrat în procesul de testare.

Pentru a realiza astfel de aplicatii sunt luate în considerare conversatiile din cadrul proceselor MT-OO, unde directiile sunt entitatile participante.

Desi anumite implementari originale ale modelului conversational, necesita ca participantii si datele folosite în fiecare conversatie sa fie cunoscute în momentul compilarii, conversatiile pot sa fie formate si în mod dinamic (nu este necesara nici o informatie despre participanti sau despre datele conversatiilor în momentul compilarii).

Bibliografie

[KAN95] - Kan S.H., "*Metrics and Models in Software Quality Engineering*", Addison Wesley, 1995;

[LOR94] - Lorenz M., Kidd J., " *Object-Oriented Software Metrics*", Prentice Hall, 1994;

[RAN97] - Randell B., "*System structure for software fault-tolerance*", IEEE Transaction of Software Engineering, vol SE-1, no.2, 1997, p.220-232;

[ROS97] - Rosenberg L., Hammer T., "*Metrics for Object Oriented Environment*", EFAITP/AIE Third Annual Software Conference, 1997.