Modelarea fiabilitatii sistemelor de programe cu arhitectura complexa

Lect. Marian CRISTESCU
Universitatea "Lucian Blaga" Sibiu

Early quality prediction at the architecture design stage is highly desired by software managers and developers, as it provides a means for making design decisions and thereby facilitates effective development processes. If a design flaw, especially in large-scale software system, is detected in the implementation or testing phase, it is more difficult and more expensive to make changes than at the architecture design stage. Software architecture analysis aims at investigating how architecture meets its quality requirements based on the structure and the correlation among the components of the software. It not only facilitates component-based software development, but also provides a means for early quality prediction. Therefore, the quality of component-based software can be predicted by using software architecture analysis.

***Keywords**: software architecture, architectural style, markov model, reliability estimation, methodologies.*

Arhitectura software descrie structura unui sistem de programe la nivel abstract. Aceasta consta dintr-un set de componente, conectori si configuratii, iar sablonul repetitiv care caracterizeaza configuratiile componentelor si conectorilor este denumit tip arhitectural.

Deoarece exista multe tipuri arhitecturale utilizatorii trebuie sa-si aleaga tipul potrivit, sau sa-si modeleze configuratii ale tipurilor selectate, pentru a-si proiecta arhitectura în conformitate cu propriile specificatii. O metoda sau un model de predictie sau evaluare a fiabilitatii unui sistem de programe complex ofera un mijloc prin care proiectantii pot configura arhitectura care se potriveste cel mai bine cu cerintelor lor de calitate.

În [CHEU80] este descris un model de fiabilitate bazat pe componente, dezvoltat de Cheung, care ia în considerare fiabilitate fiecarei componente si profilul operational. În acest model este folosita o diagrama de stare care prezinta comportamentul sistemului. O stare reprezinta executia unei singure componente si probabilitatea de tranzitie dintr-o stare în alta este obtinuta din profilul operational al sistemului.

Fiabilitatea unui sistem de programe depinde de secventa de executie a starilor si de fiabilitatea fiecarei stari. Pe baza proprietatilor lanturilor Markov se considera ca tranzitia dintre stari este un proces Markov. Aceasta înseamna ca acele componente care sunt executate în starea urmatoare depind doar de componentele starii curente si de componentele starii urmatoare si nu au nici o dependenta fata de alte stari anterioare starii curente.

Diagrama de stare este un graf directionat în care fiecare nod S_i reprezinta o stare si tranzitia de la starea S_i la S_j este reprezentata de un arc directionat (S_i, S_j). R_i exprima fiabilitatea lui S_i si P_{ij} este probabilitatea de tranzitie din S_i în S_j. Pe baza diagramei de stare s-a definit matricea de tranzitie M, prezentata în figura 1 si valoarea unei intrari $M(i,j)$, care este calculata ca $R_i x P_{ij}$,. Aceasta indica ajungerea cu succes la starea S_j de la S_i,- executia corecta a lui S_i si aparitia tranzitiei de la S_i la S_j.

În continuare este descris modul de calcul al fiabilitatii pa baza matricei de tranzitie. Fie $S=\{S_1,S_2,\ldots,S_n\}$, setul de stari din diagrama de stare, unde S_1 este starea initiala si S_n starea finala. $M^k(i,j)$ reprezinta probabilitatea de a ajunge în starea S_j din S_i prin k tranzitii.

$$M = \begin{array}{c} \\ S_1 \\ S_2 \\ \vdots \\ S_i \\ \vdots \\ S_{n-1} \\ S_n \end{array} \begin{array}{c} \begin{array}{ccccccc} S_1 & S_2 & .. & S_i & .. & S_{n-1} & S_n \end{array} \\ \begin{bmatrix} 0 & R_1P_{12} & .. & R_1P_{1j} & .. & R_1P_{1(n-1)} & R_1P_{1n} \\ R_2P_{21} & 0 & .. & R_2P_{2j} & .. & R_2P_{2(n-1)} & R_2P_{2n} \\ \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ R_iP_{i1} & R_iP_{i1} & .. & 0 & .. & R_iP_{i(n-1)} & R_iP_{in} \\ \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ R_{n-1}P_{(n-1)1} & R_{n-1}P_{(n-1)2} & .. & R_{n-1}P_{(n-1)i} & .. & 0 & R_{n-1}P_{(n-1)n} \\ R_nP_{n1} & R_nP_{n2} & .. & R_nP_{ni} & .. & R_nP_{n(n-1)} & 0 \end{bmatrix} \end{array}.$$

Fig.1. Matricea de tranzitie

Fiabilitatea R, pentru ajungerea de la starea S_i la starea S_j cu k tranzitii este:

$$R = M^k(i, j) \times R_j \qquad (1)$$

De la starea initiala S_1 pâna în starea finala S_n, numarul de tranzitii k variaza de la *0* la ¥, unde *0* înseamna ca starea initiala este si stare finala si ¥ înseamna ca între stari poate sa apara o secventa de ciclare. S-a considerat T o matrice astfel încât:

$$T = I + M + M^2 + M^3 + \ldots = \sum_{k=0}^{\infty} M^k = \frac{I}{I - M} = (I - M)^{-1} \qquad (2)$$

unde I este matricea identica de dimensiuni n x n. Fiabilitatea generala a sistemului este data de relatia:

$$R = T(1, n) \times R_n \qquad (3)$$

$$\text{cu } T(1,n) = (-1)^{n+1} \frac{|E|}{|I - M|} \qquad (4)$$

unde $|I - N|$ este determinantul matricei $(I - N)$ si $|E|$ este determinantul matricei ramase dupa excluderea celui de-al noualea rând si a primei coloane din matricea $(I - N)$.

Modelul lui Cheung este bazat pe o singura stare initiala si o singura stare finala, pe când un sistem de programe complex are un set de stari initiale $I=\{S_{i1}, S_{i2}, \ldots, S_{im}\}$ si un set de stari finale $F=\{S_{f1}, S_{f2}, \ldots, S_{fn}\}$. În acest caz s-a modificat problema starilor initiale si finale multiple si s-a obtinut o singura stare initiala si o singura stare finala, prin introducerea unei stari preinitiale si a unei stari postfinale în diagrama de stare. Figura 2 prezinta modul în care o diagrama de stare cu stari initiale si finale multiple, din dreptunghiul desenat punctat, este convertita la o diagrama de stare cu o singura stare initiala si una finala. S-a adaugat o stare preinitiala S^I si un arc directionat (S^I, S_{ij}), cu probabilitatea de tranzitie P_{1j} preluata din profilul operational, pentru fiecare $j=1,2,\ldots,m$. Similar, s-a creat o stare postfinala S^F si un arc directionat (S_{fj}, S^F) cu probabilitatea de tranzitie 1 pentru fiecare $j=1,2,..,n$; fiabilitatile ambelor stari S^I si S^F sunt fixate la valoarea 1.

Pentru un sistem de programe cu o arhitectura complexa, acest model de fiabilitate a fost îmbunatatit pentru a lua în considerare caracteristicile diferitelor tipuri arhitecturale. Tipurile arhitecturale studiate includ grupuri de tip secvential-înlantuit, apel si revenire, paralel-filtru de înlantuire si tolerant la erori. Celelalte tipuri arhitecturale sunt percepute ca extensii ale acestora; de exemplu, stilul client–server este similar cu stilul apel si revenire, dar trebuie sa ia în considerare fiabilitatile conectorilor.

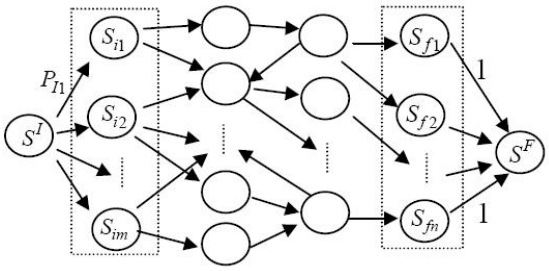

Fig .2. Diagrama de stare cu o stare preinitiala si o stare postfinala

Pentru utilizarea modelului Markov a fost introdusa o transformare, pentru fiecare tip arhitectural, dintr-o viziune arhitecturala într-o viziune de stare. Matricea de tranzitie M a fost rafinata, pe baza viziunii de stare transformata si s-a obtinut fiabilitatea sistemului de programe software bazata pe tip. În cadrul modelului R_i reprezinta fiabilitatea componentei C_i si P_{ij} reprezinta probabilitatea tranzitiei din componenta C_i în componenta sa succesoare C_j. Pentru a lua în considerare fiabilitatea conectorilor, P_{ij} a fost ajustata la o valoare egala cu probabilitatea de tranzitie originala înmultita cu fiabilitatea conectorului corespunzator.

- **Tipul grupuri secventiale/înlantuite.** Atât tipul grupurilor secventiale cât si cel înlantuit ruleaza în ordine secventiala. Ele împart aceeasi viziune arhitecturala si viziune de stare. Cu toate acestea, în tipul grupurilor secventiale iesirile unei componente sunt produse doar dupa ce toate intrarile sunt procesate complet, iar în tipul înlantuit, iesirile pot fi produse înainte de consumarea completa a intrarilor.
- **Tipul paralel/filtre de înlantuire.** Într-un mediu de executie secvential, doar o singura componenta este executata la un moment dat. Totusi, într-un mediu de executie concurenta, componentele ruleaza, de obicei simultan.

Comportamentul unui sistem de programe tinând seama de procesul de executie poate fi modelat prin folosirea unui lant Markov în care o stare este definita ca fiind executia mai multor componente. Într-un tip arhitectural paralel sau cu filtre de înlantuire, mai multe componente pot fi executate concurent, dupa cum este prezentat în suprafata punctata a figurii 3.a. Diferenta dintre aceste doua tipuri este ca procesarea paralela se foloseste în mediile multi-procesor, în timp ce tipul filtre de înlantuire este întâlnit în cazul unui singur procesor cu mai multe procese. Figura 3.b este diagrama de stare a tipurilor arhitecturale paralel/filtre de înlantuire, unde executiile componentelor C_2 pâna la C_{k-1} sunt continute în starea S_{p1} care este un element al setului de stare paralela S_p.

În figura 3.a sunt un numar *k* de componente dintre care $l = k-2$ componente ruleaza concurent în aceeasi stare; astfel, numarul total al starilor este $k-l+1$. Din cauza caracteristicilor tipului paralel, probabilitatile tranzitiei de la componenta C_1 catre componentele C_2, C_3,…,si C_{k-1} sunt toate egale cu P_{12}, care este acum probabilitatea de tranzitie de la starea S_1 la starea S_{p1}.

Pentru simplificare, se introduce $\{S_i\}$ care va returna numarul de rânduri sau numarul de coloane ale variabilei de stare S_i într-o matrice. Intrarea $M(\{S_{p1}\},\{S_k\})$ necesita ca toate componentele de la C_2 la C_{k-1} din starea S_{p1} sa ruleze cu succes si, în final sa ajunga în starea S_k.

Deoarece fiabilitatile componentelor si probabilitatile de tranzitie sunt toate independente una de cealalta, rezulta ca valoarea lui $M(\{S_{p1}\},\{S_k\})$ este egala cu $\prod_{n=2}^{k-1} R_n P_{nk}$ care este produsul fiabilitatilor tuturor componentelor din aceasta stare si a probabilitatilor de tranzitie de la componentele $C_2, C_3, \ldots,$ si C_{k-1} spre componenta C_k, în ordine.

Pentru un numar *k* de componente, matricea de tranzitie poate fi obtinuta prin:

$$\begin{cases} M(i,j) = R_i P_{ij}, S \notin S_p \\ M(i,j) = \prod_{C_n in S_i} R_n P_{nj}, S_i \in S_p \\ M(i,j) = 0 \end{cases} \quad (5)$$

pentru $1 = i, j = |S| \ldots$ *si* $1 = n = k$ si unde S_i nu poate ajunge în starea S_j.

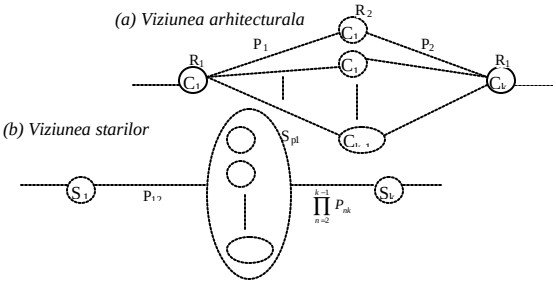

Fig.3. Tipul paralel sau filtre înlantuite

▪ **Toleranta la erori.** Se face presupunerea ca toate componentele de acoperire au aceleasi probabilitati de tranzitie catre componenta urmatoare ca si componenta primara. Daca se presupune ca exista un numar *k* de componente dintre care $l=k-4$ componente ruleaza ca tolerante la erori în aceeasi stare rezulta ca numarul total de stari este $k-l+1$ (figura 4).

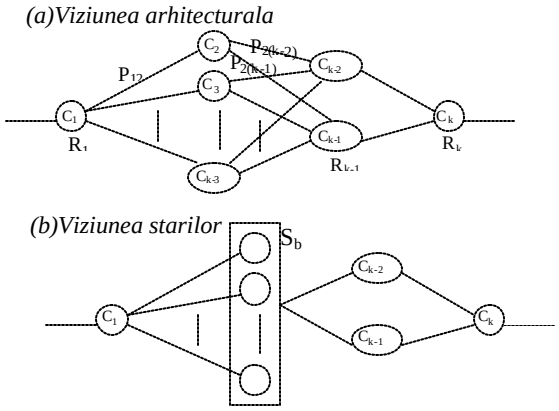

Fig.4. Toleranta la erori

Caracteristicile concurente ale tipului tolerant la erori sunt similare cu tipul paralel, unde probabilitatile de tranzitie de la componenta C_1 la componentele C_2, C_3,…, si C_{k-3} sunt toate egale cu P_{12}, care este acum probabilitatea de tranzitie de la starea S_1 la S_{b1}. Cu toate acestea, starea S_3 îsi îmbunatateste fiabilitatea doar când starea S_2 esueaza. Similar, starea S_4 îsi mareste fiabilitatea când ambele stari S_2 si S_3 esueaza. Astfel, când se calculeaza fiabilitatea atingerii starilor S_{k-1} si S_{k-2} din S_1, nu se cunoaste momentul când starea S_2 este întotdeauna activa, atunci când starea

S_2 esueaza dar starea S_3 este activa, si asa mai departe. Prin inductie, intrarea $M(1,\{S_{b1}\})$ este egala cu:

$$M(1,\{S_{b1}\}) = R_2 + \sum_{n=3}^{k-3}\left(\left(\prod_{m=2}^{n-1}(1-R_m)\right)R_n\right) \quad (6)$$

deoarece se presupune ca toate componentele de acoperire au aceleasi probabilitati de tranzitie ca si componenta primara catre fiecare componenta de tip subsecventa; valorile lui $M(\{S_{b1}\}, \{S_{k-1}\})$ si $M(\{S_{b1}\},\{S_{k-2}\})$ sunt egale cu $R_{b1}P_{2(k-1)}$ si respectiv $R_{b1}P_{2(k-2)}$.

Bibliografie :

[CHEU80] - Cheung R.C., A User-Oriented Software Reliability Model. *IEEE Transactions on Software Engineering*, 6(2):118, March 1980;

[GOLT98] - Gokhale S.S., Lyu M.R. and Trivedi K.S., Reliability Simulation of Component-Based Software Systems, In *Proceedings of ISSRE'98*, pp 192-201;

[MEDV97] - Medvidovic N., Oreizy P., Robbins J.E., and Taylor R. N., Reuse of off-the-shelf components in c2-style archtiectures, In *Proceedings of 19th International Conference on Software Engineering*, May 1997;

[SPIT98] - Spitznagel B. and Garlan D., Architecture based performance analysis, In Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering, 1998;

[TAYL96] - Taylor R.N., Medvidovic N. and Anderson K.M., A component and message-based architectural style for gui software, *IEEE Transactions on Software Engineering*, June 1996;