Milan ZORIĆ
ELEKTROTEHNIČKI FAKULTET
ZAGREB, Unska 3

PRIMJENA MODELA DISKRETNOG AUTOMATA U PROCESU TESTIRANJA PROGRAMSKE PODRŠKE ELEKTRONIČKIH KOMUTACIJA

THE USE OF FINITE-STATE MODEL IN SPC EXCHANGE SOFTWARE TESTING

SADRŽAJ - U radu se analiziraju neki aspekti izgradnje kvalitetne programske podrške komutacijskih sustava. Posebna pažnja posvećena je ranim fazama gdje se ukazuje na potrebu modeliranja. Analizira se odnos diskretnog automata i SDL opisa. Prikazana je struktura signalizacijskog podsistema ETC 960. Opisani su postupci testiranja primjereni ovakvoj koncepciji.

ABSTRACT - In this paper certain aspects of advanced software development in communication systems are analysed. Special attention has been paid to early phases where the need for modelling is stressed. Relations between finite-state automata and SDL description are analysed. Signalling subsystem structure in the ETC 960 switching system is shown. Appropriate testing procedures are described.

1. UVOD

Modernizacijom telekomunikacijske mreže, te postepenim prelaskom na digitalnu mrežu integriranih usluga, mijenjaju se zahtjevi koji utječu na način izgradnje i logičku organizaciju komutacijskih sistema. Dok su raniji sistemi bili suočeni s problemom cijene sklopova, pa stoga i bili gradjeni tako da sklopovi budu maksimalno iskorišteni, danas je cijena izgradnje programske podrške dominantan faktor. S druge strane prvi SPC komutacijski sistemi sadržavali su oko 150 usluga dok je taj broj tokom sedamdesetih godina narastao na oko 450 sa trendom daljeg porasta. Uz to pokazuje se da je pored ostalog neophodno:

- skratiti vrijeme razvoja,
- omogućiti korisniku upravljanje i administriranje komutacijskih resursa,
- kontinuirano pratiti razvoj novih sklopova, te ugradjivati jeftinije komponente u nove verzije sistema,
- unaprijediti metode dokumentiranja usluga, što je potrebno za uspješnu prodaju, održavanje, planiranje tarifa itd,
- osigurati visoku pouzdanost u cijelom (vrlo dugom životnom ciklusu).

Svi ovi zahtjevi utjecali su s jedne strane na strukturu programske podrške, a s druge strane na sam proces izgradnje programske podrške. U ovom radu bit će detaljnije obradjeni neki aspekti izgradnje programske podrške vezani uz primjenu modela diskretnog automata.

2. ZNAČAJ SPECIFIKACIJA

Nepouzdanost programske podrške dobrim je dijelom posljedica grešaka u programskoj podršci. Greške je moguće klasificirati na više načina koji se medjusobno preklapaju, pri čemu neke polaze od pojavnih oblika, dok druge nastoje razotkriti suštinu nastanka greške. Na ovom mjestu koristit ćemo klasifikaciju koju daju Goodenough i Gerhart [1]:

1. greške u definiranju zahtjeva (zahtjevi ne održavaju stvarnu potrebu)
2. greške u dizajnu (neuspjeh u zadovoljenju postavljenog zahtjeva)
3. greške u dokumentaciji (dokumentacija ne reprezentira dizajn)
4. greške u konstrukciji (neuspjeh da se zadovolji zahtjev zbog greške u pisanju programa).

Očito je da se samo jedna kategorija grešaka odnosi na program u užem smislu. Dodamo li ovome podatak iz više studija, koji govori da se do 2/3 grešaka u programskoj podršci može pripisati nekompletnim i nekonzistentnim specifikacijama, postaje jasno zašto se specifikacije kao takve moraju detaljnije analizirati.

Specifikacije pojedinih funkcija komutacijskih sistema bile su često dane opisno, što je rezultiralo slabo organiziranom, a često obimnom, dokumentacijom. Takve specifikacije su u pravilu pisane za svaki razvoj posebno, te je stil, sadržaj i format znatno varirao. Često je slučaj nepotpunih ili nejasnih specifikacija, te raznih opisa specifičnih za dani sistem. Univerzalni kompromis izmedju općeg i konkretnog u praksi je rješavan korištenjem različitih pojmova i načina njihovog označavanja, kako za različite klase sistema, tako i za različite aspekte unutar istog sistema. Apstraktne funkcije sistema mogu uvijek biti opisane (i najčešće jesu) kroz specifičnu implementaciju. To nikako ne može biti zadovoljavajuće. Potrebno je pronaći pojmove koji nisu ovisni o implementaciji koji bi omogućavali modeliranje neovisno o implementaciji. Samo tako moguće je shvatiti svojstva sistema sama po sebi, te napraviti dobro fundiran izbor strategije implementacije.

Iz navedenog je evidentno da postoji potreba za modelima koji su neovisni o implementaciji, koji definiraju šta sistem treba napraviti, te modelima koji su vezani za implementaciju koji govore kako sistem radi. Različiti modeli mogu imati različit način označavanja, te biti osnovani na različitim pojmovima, no, bolje je ako to nije slučaj.

Modeliranjem sistema potrebno je dati kako statičke aspekte sistema (struktura), tako i dinamičke (ponašanje). Statički aspekti se jednostavno modeliraju, dok dinamički aspekti zahtijevaju dodatni napor čitaoca jer treba shvatiti kako nešto radi dinamički čitajući statički opis. Statička prezentacija dinamike mora biti bazirana na nekim pretpostavkama o načinu rada. Barem minimum znanja o tim pretpostavkama potreban je čitaocu da bi mogao generirati dinamičku sliku iz statičkog opisa. Ovo često nije posebno naglašeno, ali je za preciznu komunikaciju

neophodno da autor i čitalac polaze od istih pretpostavki.

Bilo koja tehnika modeliranja ima dva aspekta:

- skup osnovnih pojmova, ili apstrakcija na kojima se baziraju modeli i na osnovi kojih su modeli razumljivi (semantika),
- notacija za opis modela (sintaksa).

Bez preciznog definiranja oba aspekta, kompletni i jasni modeli se ne mogu izgraditi.

Skup osnovnih pojmova mora biti pažljivo izabran, jer to odredjuje snagu i ograničenja tehnike modeliranja. Način označavanja mora biti tako izabran da odgovara čitaocu modela, tako da se opisane ideje prenose efikasno i nedvosmisleno. Izbor osnovnih pojmova je posebno značajan jer odražava i utječe na naš način razmišljanja o sistemu. Kroz historiju, napredak znanosti i tehnologije je bio usko vezan sa ljudskom sposobnošću da nadje osnovna svojstva te ih koristi u apstraktnim modelima realnih sistema. Stoga, osnovni pojmovi moraju predstavljati sredstva razumijevanja sistema, a ne barijeru. U svakom slučaju skup osnovnih pojmova mora nas osloboditi trivijalnosti i pomoći u konstruktivnom razmišljanju o važnim aspektima sistema.

Možemo reći da su u procesu razvoja programske podrške precizne specifikacije neophodne iz slijedećih razloga:

1. da se opiše problem koji treba riješiti,
2. za komunikaciju medju ljudima,
3. da se programer oslobodi potrebe da zna kako radi ostatak sistema,
4. za podršku razvoju programske podrške sa više verzija,
5. za rasčlanjivanje problema u obradive podprograme,
6. da se omogući verifikacija medjukoraka u odlučivanju o dizajnu.

3. MODELIRANJE

Sekvencijalni procesi koji se odvijaju u komutacijskim sistemima mogu biti interpretirani kao usmjereni graf čiji čvorovi odgovaraju stanjima, a čije grane odgovaraju prijelazima. Izbor terminologije je proizvoljan. Terminologija procesa može biti korištena kada se misli na dinamičku interpretaciju, a terminologija grafa se može koristiti u statičkom smislu. Stoga je proces jedinka koja prelazi (putuje) grafom stanja, a graf stanja definira sve moguće putove koje proces može prijeći.

Procesi definirani programima mijenjaju stanja izvodjenjem instrukcija pisanih u programu. Analiza procesa zahtijeva da grafovi stanja budu transformirani tako da im struktura bude sačuvana. U grafu stanja struktura koja treba biti sačuvana je skup grana. Transformacije grafa procesa kreću se izmedju dva ekstremna slučaja. Prvi je takva transformacija u kojoj je jedina preostala komponenta programski brojač. Ovaj oblik možemo nazvati i opisom procesa preko aktivnosti

koje generiraju odredjene situacije. Sama stanja su često izrazito implicitna i neuočljiva. To je slučaj sa najvećim brojem opisa u dijagramima toka i uobičajenim općim programskim jezicima. Kako bi mogao razumjeti opis, čitalac mora formirati mentalnu sliku stanja integracijom aktivnosti, što može predstavljati složen zadatak. U okviru ovakvog pristupa opisivanju procesa intelektualna kontrola se želi zadržati uvodjenjem strukturnog programiranja, koje nameće ograničenja na strukturu sekvenci akcija, u cilju što lakšeg rekonstruiranja stanja. Opisi orjentirani na stanja su drugi ekstrem i radikalno su drugačiji. Proces se opisuje preko niza dobro definiranih stanja. U tom slučaju moguće je koncentrirati se na stanja i uvjete pod kojima se prelazi iz jednog stanja u drugo.

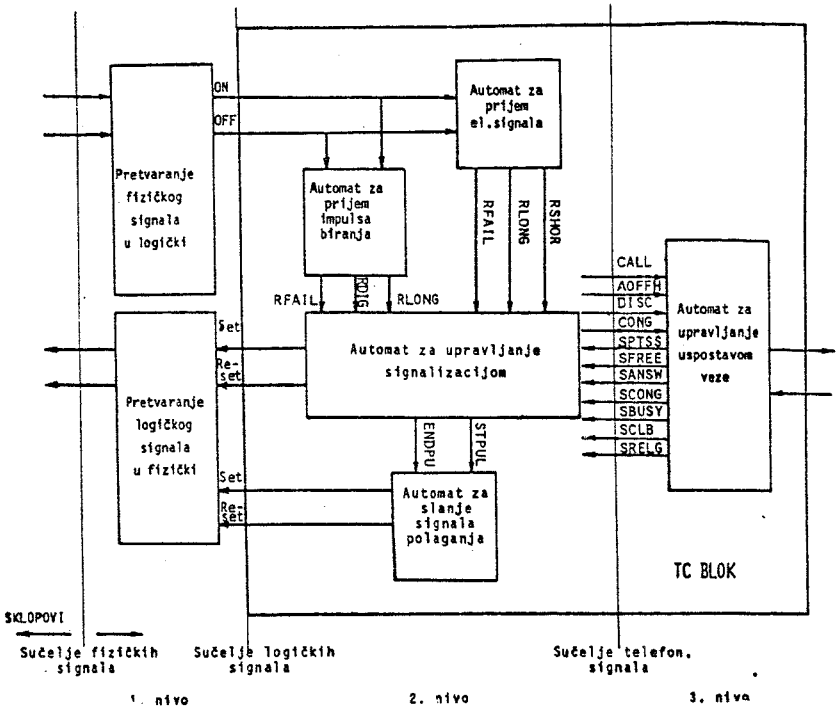
Ukoliko je broj stanja konačan, a prijelazi iz stanja u stanje su odredjeni samo stanjem i ulaznim dogadjajem možemo proces modelirati diskretnim automatom. Diskretni automat posjeduje važno svojstvo da se radi o dobro definiranoj apstraktnoj mašini definiranoj preko matematičke funkcije, te stoga ima jaku teoretsku osnovu. Postoje formalizirane metode analize i sinteze. To je veoma važno ako se razmišlja o verifikaciji, kontroli sintakse, automatizaciji dizajna ili logičkoj simulaciji. Dizajn baziran na diskretnom automatu je jednostavan i jasan. Moguće je postići povoljan nivo apstrakcije, pri čemu ono bitno dolazi u prvi plan, a detalji se razradjuju kasnije. Dizajn može biti optimiran obzirom na različite zahtjeve kao što su efikasnost korištenja vremena/prostora, podobnost za održavanje itd. Modeliranje je potpuno nezavisno od implementacije, a realizacija je moguća u svim relevantnim tehnologijama.

Sredina izmedju ova dva ekstrema, uz nastojanje da se ipak bude što bliže diskretnom automatu, predstavlja pristup korišten u jeziku SDL. Ovdje je uveden koncept odluke koji omogućava da novo stanje ne bude samo funkcija starog stanja i ulaznog dogadjaja, već i vrijednosti varijable na osnovu koje se donosi odluka. Moguće je pretvoriti SDL opis u diskretni automat (i obratno), kao što je to opisano u [2], ali je za to potrebno:

- da je broj odluka mali,
- da varijabla na osnovu koje se vrši grananje poprima mali broj vrijednosti.

4. STRUKTURA MODELA SIGNALIZACIJSKOG PODSISTEMA ETC 960

Razrada strukture signalizacijskog podsistema ETC 960 opisana je detaljnije u [2]. U osnovi sistem se dekomponira na niz asinhronih automata rasporedjenih u odgovarajuće nivoe, koji medjusobno komunicirajući putem signala realiziraju funkcije sistema (slika 1). Svaki automat definiran je odgovarajućim SDL dijagramom, te predstavlja podlogu za programsku implementaciju i testiranje.



Slika 1: Struktura modela signalizacijskog podsistema za D3 signalizaciju - dolazni promet

5. TESTIRANJE MODELA

Svakim testiranjem programske podrške utvrdjuje se da li programi odstupaju od specifikacija. Obzirom da u fazi realizacije modela (tj. kodiranja programa) očekujemo manje problema ukoliko je model dobro napravljen poželjno je prije pisanja programa utvrditi da li model odgovara specifikaciji. Ukoliko je model diskretni automat dokazano je [8] da je navedeno moguće postići. U nastavku će biti na neformalan način izložena metoda testiranja dizajna. (Precizniji opis u [6]).

Pretpostavimo da specifikacija implicite sadrži u sebi diskretni automat. Da bi model vjerno prikazivao specifikaciju, automat impliciran specifikacijom i automat koji čini model, moraju biti ekvivalentni. T.S. Chow [8] je dokazao da se ekvivalentnost tih automata može pokazati tako da za odgovarajući skup ulaznih sekvenci usporedjujemo izlazne sekvence koje dobijemo iz automata koji čini model i onih koje proizlaze iz specifikacije. Ukoliko nema razlika, automati su sigurno ekvivalentni. Spomenuti skup ulaznih sekvenci moguće je generirati na odgovarajući način iz modela. Broj test sekvenci moguće je jednostanim metodama minimizirati.

Opisana metoda zahtjeva da diskretni automat bude potpuno specificiran, minimalan, da počinje u fiksnom stanju te da su sva stanja dostupna. Takodjer slijedeće stanje mora ovisiti isključivo o prethodnom stanju i ulazu. Da se pokazati da su svi ovi zahtjevi realni te da ih je moguće zadovoljiti. Uz odredjene manipulacije moguće je obuhvatiti i takve slučajeve gdje nisu svi zahtjevi zadovoljeni. Ukoliko su svi zahtjevi zadovoljeni, opisanom metodom se sigurno otkrivaju greške u sekvenciranju izlaza diskretnog automata, tj. greške u upravljačkoj strukturi. Na ovaj način nije moguće testirati realizaciju pojedinih operacija koje čine izlaze diskretnog automata.

Test podaci generirani na osnovu modela mogu se koristiti i prilikom pripreme podataka za testiranje napisanih programa, pri čemu se opet može kontrolirati samo upravljačka struktura programa.

Očito je da proces testiranja modela nije moguće u potpunosti automatizirati. Usporedbą izlaza automata i izlaza dobivenog na bazi specifikacije ne može biti automatizirana. Generiranje test podataka moguće je automatizirati što je napravljeno. Programski sistem razvijen u tu svrhu opisan je u [7].

6. KONTROLA NAPISANOG KODA

Polazeći od ispravnog dizajna pristupa se pisanju programa. Preslikavanje dizajna u program mora biti jednostavno i jednoznačno. Mora postojati 1 - 1 korespondencija izmedju elemenata dizajna i implementacije u programu. Procesu u SDL dijagramu treba odgovarati modul programa, stanju odgovara stanje višeg programskog jezika ili vrijednost varijable koja nosi informaciju o stanju u nižem jeziku. Upravljačka struktura automata realizira se preko tabela, dok izlazi diskretnog automata (prijelazi u SDL dijagramu) predstavljaju izvršne procedure, (na primjer: procedurni programi u ETC 960).

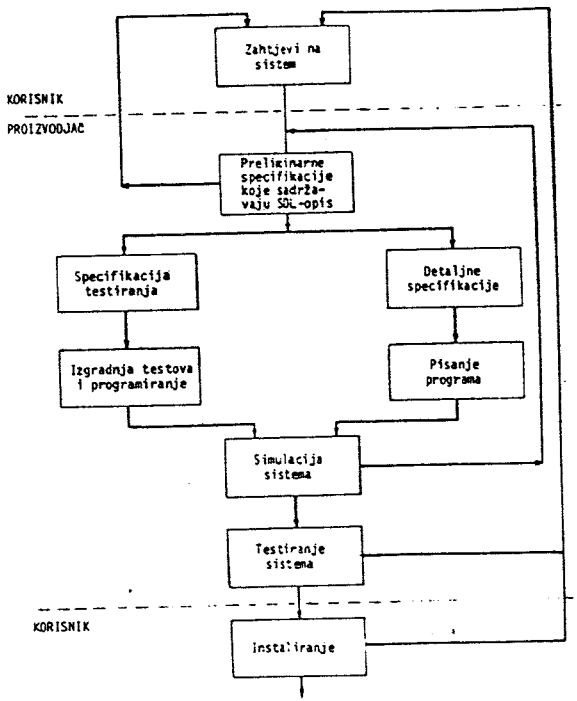
Nakon što je program napisan potrebno je da drugi član tima izvrši kontrolu napisanog. Kontrolira se sintaktička ispravnost napisanog programa te posebno njegova podudarnost sa modelom. Pored toga, kontrolira se čitljivost te poštivanje dogovorenih načina označavanja i dokumentiranja. Ovaj dio je neobično bitan za kasnije održavanje sistema, te eventualne modifikacije.

7. TESTIRANJE REALIZIRANOG AUTOMATA

Pristup izgradnji sistema preko automata bitno utječe na način testiranja te njegovu kompleksnost. Automat može biti testiran izolirano od svoje stvarne okoline tako da se simuliraju ulazne sekvence i kontroliraju izlazni signali, što se obično provodi na računalu opće namjene. Ovisno o karakteru programske jedinice nekada je to moguće uz pomoć interpretera visokog programskog jezika u kome je program realiziran, a nekada je potrebno osigurati simulaciju sklopovske

okoline sa kojom program treba suradjivati. Modul može biti doveden u sva stanja te usmjeren na sve putove kroz program. Osnova za pripremu ovih testova su SDL dijagrami. Pri tome se testovi odabiru na taj način da se barem jednom prodju sve grane u SDL dijagramu. Definiranje testova treba biti radjeno paralelno sa razvojem programske podrške, što je prikazano na slici 2. Poželjno je da se ovaj postupak automatizira jer se time uz povećanje brzine i preciznosti testiranja može postići:

- rigorozna verifikacija podudarnosti sa specifikacijom
- mogućnost neregresivnih testova na sukcesivno razvijenim verzijama programa
- mogućnost ponovnog korištenja većine testova u slučaju novog razvoja
- mali broj grešaka koje preostaju nakon ovoga koraka.



Slika 2: Elementi procesa razvoja programske podrške

8. ZAKLJUČAK

Iako se može reći da su danas identificirane sve faze u životnom ciklusu programske podrške, te da se kvaliteta nastoji ugraditi u svaku od tih faza, ipak nam se čini da treba još više naglasiti utjecaj ranih faza u razvoju programske podrške na ukupnu kvalitetu proizvoda. Interaktivni sistemi, koji pružaju

podršku u pisanju, dokumentiranju i testiranju programske podrške u znatnoj mjeri doprinose poboljšanju kvalitete i produktivnosti, ali ne pružaju isti nivo podrške na nivou dizajna. Prelazak na strojno obradivi zapis o dizajnu, na primjer programski oblik SDL, omogućio bi eksperimentiranje na nivou dizajna, te automatizaciju kontrole ekvivalentnosti automata danog specifikacijom i implementiranog u programu. Time bi bilo osigurano da u više faze testiranja mogu proći neotkrivene samo greške na sučeljima sa drugim podsistemima, i to prvenstveno one koje nastaju zbog opterećenja ili preopterećenja u realnom vremenu.

LITERATURA

[1] R.T.Yeh, Ed.: "Current Trends in Programming Methodology" Vol II Program Validation, Prentice-Hall, Inc. Englewood Chiffs, New Yersey 1977.

[2] M.Zorić: "Primjena modela diskretnog automata u procesu testiranja programske podrške elektroničkih komutacija", Elektrotehnički fakultet u Zagrebu, magistarski rad.

[3] R.T.Yeh, Ed.: "Applied Computation Theory: Analysis, Design, Modelling" Prentice-Hall, Inc. Englewood, New Yersey 1976.

[4] S.R.Traves, K.Caves: "New Approach to Signalling System Specification" ISS 79 Paris pp 1292-1300.

[5] R.Braek: "Unified System Modelling and Implementation" ISS 79 Paris pp 1180-1187.

[6] M.Zorić: "Testiranje programske podrške pomoću modeliranja diskretnim automatom", JUREMA 1981.

[7] B.Mikac,M.Zorić: "Dijagnostika i raspoloživost elemenata integrirane komunikacijske mreže" YUTEL, Ljubljana 1981.

[8] T.S.Chow: "Testing Software Design Modeled by Finite-State Machines" IEEE Trans. on Software Engineering, Vol SE 4 No 3 May 1978.

[9] A.Gill: "Introduction to the Theory of Finite-State Machines", McGraw-hill, New York 1962.

[10] M.P.Vasilevski: "Failure Diagnosis of Automata" Kibernetika No 4, pp 98--108 July-Aug. 1973.

[11] R.C.Linger, H.D. Mills, B.I.Witt: "Structured Programming" Addison-Wesley Publ. Comp. Reding Massachusets 1979.