Jedno rešenje prevodioca SDL dijagrama

Bogdan D. Trivunović, Miroslav V. Popović, Ivan S. Velikić,
Robert I. Pečkai – Kovač, Milan M. Ačanski, Slavko M. Svirčević

***Sadržaj* — U radu je prikazana realizacija jednog rešenja prevodioca SDL (Specification and Description Language) dijagrama u C++ klase zasnovane na FSM biblioteci razvijenoj na Katedri za računarsku tehniku i računarske komunikacije, Fakulteta tehničkih nauka u Novom Sadu. Rešenje prevodioca je obezbedilo i potrebnu informaciju u XML (Extensible Markup Language) obliku za lociranje greške u polaznom projektu na osnovu otkaza u realnom vremenu.**

***Ključne reči* — FSM, IDE, prevodilac SDL dijagrama, reaktivni sistemi, SDL, sistemi konačnih automata.**

I. Uvod

PREVODILAC SDL dijagrama nudi brz, jednostavan i pregledan način zadavanja automata u sistemu konačnih automata, kao i njihovo prevođenje u C++ klase.

U toku projektovanja i programske realizacije prevodioca vođeno je računa o činjenici da bi prevodilac trebalo da bude deo šireg rešenja integrisanog razvojnog okruženja (IDE) sistema konačnih automata [1].

Prevodilac obuhvata alate za unos SDL dijagrama, generisanje C++ koda i generisanje XML dokumenta sa informacijama potrebnim za lociranje greške u polaznom projektu na osnovu otkaza u realnom vremenu. Prevodilac SDL dijagrama omogućava i upravljanje projektnim informacijama, proširenje i lako dodavanje preostalih alata iz koncepta IDE.

U radu je prikazan koncept IDE za reaktivne sisteme, uloga prevodioca SDL dijagrama u tom sistemu, kao i veze prevodioca sa ostalim alatima u IDE. Ukratko je opisana biblioteka FSM, kao i proces projektovanja rešenja od opisa sistema, pravljenja prototipa korisničke sprege do definisanja klasa i načina saradnje među klasama. Prikazane su osnovne odrednice programske realizacije. Ukratko su prikazani i način ispitivanja, kao i praktične koristi realizacije ovog prevodioca i naredni koraci u njegovom razvoju.

Rad je delimično podržan u okviru projekta TR-6136B Ministarstva za nauku i zaštitu životne sredine Republike Srbije.

Bogdan D. Trivunović, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (telefon: 381-21-4801131; faks: 381-21-450721; e-mail: Bogdan.Trivunovic@MicronasNIT.com).

Miroslav V. Popović, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: Miroslav.Popovic@MicronasNIT.com).

Ivan S. Velikić, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: Ivan.Velikic@MicronasNIT.com).

Robert I. Pečkai-Kovač, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: Robert.PeckaiKovac@rt-rk.com).

Milan M. Ačanski, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: Milan.Acanski@rt-rk.com).

Slavko M. Svirčević, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: Slavko.Svircevic@krt.neobee.net).

II. Prevodilac SDL dijagrama kao deo IDE za reaktivne sisteme

Savremena inženjerska praksa, veoma često, susreće se sa problemom realizacije sistema konačnih automata. Koncept IDE za reaktivne sisteme izlaže ideju integrisanja svih potrebnih alata za unošenje automata, ispitivanje i verifikaciju sistema, prevođenje dijagrama u C++ kod i otklanjanje grešaka, sa ciljem poboljšanja i ubrzanja procesa razvoja reaktivnih sistema. Uzimajući u obzir svu šarenolikost primene reaktivnih sistema, kao i činjenicu da se oni veoma često nalaze kao kritični sistemi u uslovima u kojima ne sme doći do otkaza, jasna je želja za realizacijom specijalizovanog razvojnog okruženja koje će uključivati razvojnu i fazu testiranja sistema. Takvo okruženje predstavlja korak ka funkcionalnim, korektnim i pouzdanim programskim realizacijama reaktivnih sistema.

Kroz opis razvojnog ciklusa jednog sistema konačnih automata, mogu se uočiti osnovne komponente IDE. Razvojni ciklus počinje unošenjem SDL dijagrama za sva stanja svih automata u sistemu. Automati se predstavljaju u grafičkom obliku SDL-a, uz dodatni C++ kod pridružen pojedinim blokovima. Centralna komponenta IDE je SDL editor u kojem je moguć unos i izmene dijagrama, kao i definisanje vremenskih kontrola za sve automate.

Koncept predviđa upotrebu spoljašnjeg alata za prevođenje i povezivanje C++ koda. Važno je i obezbediti vezu između izlaza prevodioca i povezivača sa SDL dijagramima. Ovu vezu predstavlja dokument koji sadrži izveštaje o upozorenjima i greškama, a pomoću kojeg je moguće pozicioniranje na SDL blok ili dodatni C++ kod odgovoran za nastalu grešku ili upozorenje.

IDE mora da obezbedi i sledeće module i funkcije:

- integraciju sa sistemima za kontrolu verzija,
- mogućnost otklanjanja grešaka „korak po korak“,
- modul za automatsko generisanje skupa testova,
- MSC (Message Sequence Charts) editor, za unos korisnički definisanih testova u obliku MSC dijagrama,
- komponentu za upravljanje projektom (Project Menager).

Zahvaljujući preciznoj dekompoziciji problema i jasno izdvojenim modulima i podeljenim poslovima po modulima u konceptu IDE, lako je bilo izolovati podsistem IDE koji obuhvata alate za unos SDL (Specification and Description Language) dijagrama, alate za prevođenje dijagrama u C++ kod i alate za pravljenje dokumenta u XML (Extensible Markup Language) obliku i uočiti sve veze sa ostalim podsistemima, zahvaljujući čemu je obezbeđena mogućnost dodavanja i preostalih alata u konačan sistem, integrisano razvojno okruženje za sisteme konačnih automata.

III. BIBLIOTEKA FSM

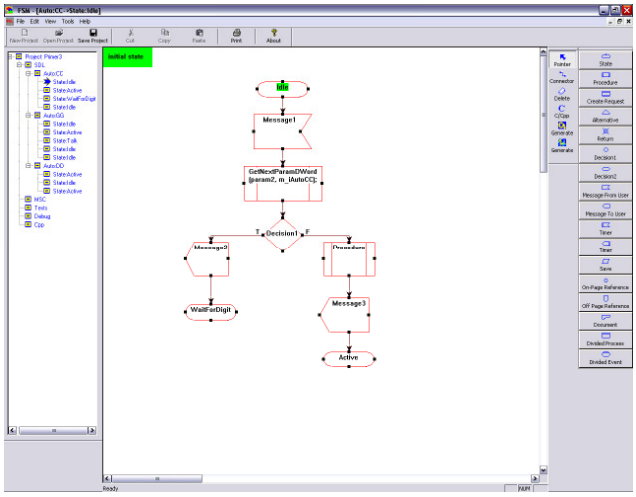
Rezultat prevođenja SDL dijagrama su C++ klase zasnovane na FSM biblioteci razvijenoj na Katedri za računarsku tehniku i računarske komunikacije, Fakulteta tehničkih nauka u Novom Sadu [2], [3]. Osnovna namena ove biblioteke je realizacija konačnih automata koji se koriste za implementaciju telekomunikacionih protokola. Automati se povezuju u sistem automata koji rukuje resursima potrebnim za funkcionisanje protokola. Implementacija određenog programskog rešenja svodi se na realizaciju funkcija prelaza. Funkcija prelaza se sastoji od procedura koje obrađuju primljenu poruku u datom stanju automata na osnovu kojih automat prelazi u novo stanje i generiše odgovarajuće poruke drugim automatima. Za ispravan rad sistema potrebno je inicijalizovati sve elemente FSM jezgra koji će biti korišćeni. Osnovne komponente jezgra su realizovane kao klase koje daju osnovnu funkcionalnost automatima i sistemu automata. Funkcionalnost automata, koji su potrebni za konkretan problem, se ostvaruje nasleđivanjem osnovne klase FiniteStateMachine. U izvedenoj klasi potrebno je realizovati virtuelne funkcije, realizovati nove funkcije koje opisuju specifičnosti konkretnog problema ili redefinisati neke nasleđene funkcije kako bi se izmenila osnovna funkcionalnost bazne klase. Instanca klase FSMSystem je objekat koji predstavlja sistem automata. Zaštićeni atributi ove klase predstavljaju resurse koji stoje na raspolaganju automatima u sistemu automata. Osnovni zadaci ove klase su rukovanje porukama, rukovanje memorijom i rukovanje vremenskim kontrolama.

IV. PROJEKTOVANJE REŠENJA

Nakon obavljene analize problema i potvrđene izvodljivosti projekta napisan je dokument sa opisom sistema. Ova dva početna koraka projektovanja su obavljena i prikazana u [1]. U daljem tekstu su, u najkraćim crtama, prikazani naredni koraci u fazi projektovanja rešenja. Analizom opisa sistema u tom dokumentu dolazi se do specifikacije zahteva, koja predstavlja konačnu predstavu sistema predviđenog za realizaciju.

Uneti dijagrami prikazuju funkcije prelaza iz jednog stanja u drugo na osnovu određene pobude. Dijagrami su sačinjeni od SDL blokova povezanih usmerenim vezama. Za svaki od blokova potrebno je omogućiti unošenje odgovarajućih parametara karakterističnih za dati blok, kao i lokalnih promenljivih funkcije prelaza, atributa automata, sistemskih i globalnih promenljivih. Svakom automatu je pridružen i niz vremenskih kontrola. Komunikacija delova sistema se odvija slanjem i primanjem poruka pa je neophodno obezbediti definisanje poruka, dodelu jedinstvenih identifikacija porukama, kao i vremenskim kontrolama, automatima i poštanskim sandučićima svakog automata. Potrebno je omogućiti i postavljanje početnog stanja za svaki automat. Zahtevan je i lak pristup i izmena podataka o automatima, dodavanje i uklanjanje dijagrama prelaza i celokupnih automata. Generisani kod bi trebalo da sadrži i potrebne komentare koji će omogućiti lakše pozicioniranje i otklanjanje prijavljene greške.

Po ustanovljavanju specifikacije zahteva izrađen je prototip korisničkog okruženja, sl.1.



Sl. 1. Prikaz prototipa korisničkog okruženja

Osnovne komponete prototipa su: okvirni prozor sa odgovarajućim menijima (u zavisnosti od trenutno aktivnog alata), radna površina za unos dijagrama, palete sa alatima (koje je moguće prikazati i ukloniti sa radne površine), prozora za upravljanje komponentama projekta (uključuje i stablo sa komponentama projekta i omogućava pozivanje dijaloga za prikaz i izmenu informacija i parametara segmenata sistema, kao i dovođenje prozora sa zahtevanim dijagramom u prvi plan).

Za prikazivanje C++ i XML koda se predviđa, u ovoj verziji alata, upotreba spoljašnjeg programa, kojem će biti prosleđena putanja do dokumenta sa kodom. Za unos, prikaz i izmenu raznih parametara predviđeno je nekoliko desetina prozora.

Analizom zahteva dolazi se do nekoliko grupa klasa, kao osnovnih jedinica strukturnog modela sistema:

- klase koje predstavljaju gradivne elemente SDL dijagrama, izvedene iz osnovne klase CShape,
- klase dijalog prozora koje prikazuju, preuzimaju i dozvoljavaju izmenu informacija o sistemu i komponentama sistema, izvedene iz klasa CDialog ili CpropertyPage,
- klase koje predstavljaju automat, stanje, bafer, vezu na dijagramu, vremensku kontrolu, identifikacije, liniju, oblik, promenljivu, izvedene

iz osnovne klase CObject i

- ostale klase, među kojima su klase za rad sa XML-om, okviri prozora, klase za prevođenje dijagrama u C++ kod, osnovne klase dokumenta i aplikacije.

U narednoj fazi je bilo potrebno utvrditi odgovornosti i način saradnje svake od klasa. Pod pojmom odgovornosti podrazumevan je skup koji čine informacije koje ta klasa poseduje i usluge koje ona može obaviti. Vođeno je računa o ravnomernoj raspodeli odgovornosti po klasama sistema.

Generisanje modela sistema prikazanog kroz skup UML (Unified Modeling Language) dijagrama predstavlja završni korak projektovanja rešenja.

Prikaz dijagrama saradnje i objašnjenje svih veza u tom dijagramu, kao i prikaz UML modela sistema prevazilazi obim rada, s obzirom da se prevodilac sastoji od velikog broja klasa. Iz istog razloga je izostalo i nabrajanje svih projektovanih i realizovanih klasa i opisi njihovih atributa i karakterističnih metoda.

V. PROGRAMSKA REALIZACIJA

U fazi programske realizacije dosledno su ispoštovane odluke donesene u fazi projektovanja. Realizovano je preko 60 klasa. Korišćen je programski model dokument/prikaz, kao osnov za razvoj MFC (Microsoft Foundation Class) aplikacije. MFC aplikacije standardno koriste ovaj model u kojem se podaci razdvajaju od prikaza tih podataka i najvećeg dela korisnikove interakcije sa podacima.

Za implementaciju SDL-a po ITU preporuci [4] odlučeno je da se unose dijagrami prelaza, a skup takvih dijagrama sa istim početnim stanjem precizno definiše određeno stanje. Dijagrami prelaza su jednostavniji, jasniji i pregledniji od dijagrama celokupnog stanja.

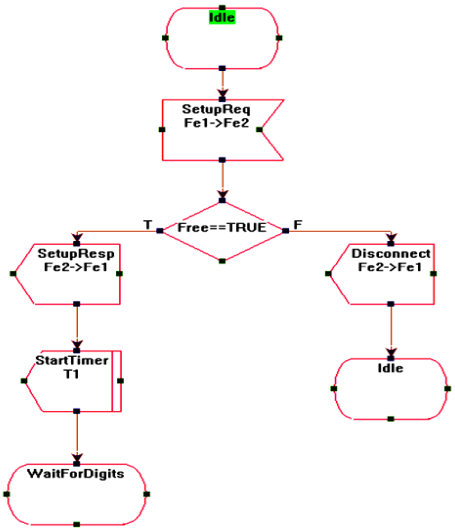
Odvojeni objekti prikaza vode računa o predstavljanju podataka, počevši od njihovog iscrtavanja u prozoru do korisnikovog označavanja i izmene tih podataka. Prikaz preuzima podatke od dokumenta i dokumentu prosleđuje informaciju o eventualnoj izmeni podataka.

Generisanje C++ koda zasnovano je na šablonima klasa isporučenim od strane FSM biblioteke, koji su u postupku prevođenja, a na osnovu analize unetih dijagrama, popunjavani odgovarajućim konstantama, promenljivima, funkcijama prelaza, inicijalizacijama vremenskih kontrola i opisima sistema poruka i poštanskih sandučića.

Prikaz podataka u vidu stabla, „karakteristična“ je i za SDL dijagrame, kao i za XML prikaz. Ova činjenica ukazuje na to da je generisanje XML koda predstavljalo jedan od lakših segmenata realizacije prevodioca.

Manipulaciji objektima SDL dijagrama i komfornosti korisničkog okruženja je posvećena velika pažnja. Sa ciljem projektovanja što komfornije grafičke korisničke sprege poštovane su opšte prihvaćene konvencije o projektovanju i organizaciji menija sa alatima, kontekstnih menija i dijalog prozora. U realizaciji unosa SDL dijagrama omogućeno je pozicioniranje i pomeranje SDL blokova, promena njihove veličine, dodavanje veza, kao i dodavanje i uklanjanje blokova, dijagrama prelaza, kao i celih automata, promena imena elemenata i dodavanje C++ koda blokovima.

Primer SDL dijagrama, zadatog upotrebom SDL editora, kao sastavnog dela SDL prevodioca, dat je na sl.2. Svaki od blokova, dijagram, kao i sistem u celini poseduje niz dodatnih informacija koje se postavljaju u odgovarajućim dijalog prozorima.



Sl. 2. Prikaz SDL dijagrama

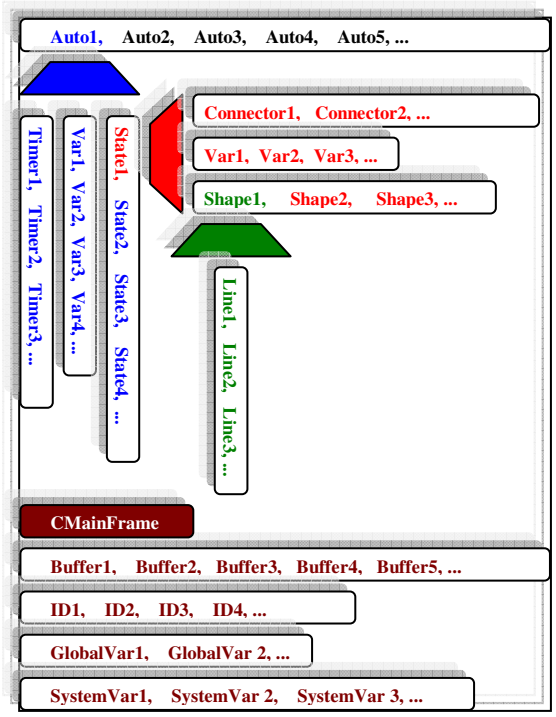
Za SDL dijagram prikazan na sl.1. prevodilac će generisati sledeću funkciju prelaza.

```
void CCClass::IdleSetupReq(){
//oblik MESSAGEFROMUSER
//kod za oblik DECISION1
if (Free == TRUE){
  //kod za oblik MESSAGETOUSER
  PrepareNewMessage (LEN, MSG_CC_ID_SETUP_RESP);
  SendMessageLeft();
  //kod za oblik TIMER
  StartTimer(TMR_CC_T1);
  //kod za oblik STATE
  SetState(WaitForDigits);
  }
else{
  //kod za oblik MESSAGETOUSER
  PrepareNewMessage (LEN, MSG_CC_ID_DISCONNECT);
  SendMessageLeft();
  //kod za oblik STATE
  SetState(Idle);
  }
}
```

Višedimenzionalni dinamički nizovi pokazivača predstavljaju osnovu uređenja podataka u prevodiocu. Na sl.3. prikazana je struktura višedimenzionalnog dinamičkog niza pokazivača. Klasa CMainFrame sadrži nizove pokazivača na objekte tipa CBuffer, CID i dva niza pokazivača na objekte tipa CVariable, za globalne i sistemske promenljive. Klasa CFSMDoc sadrži niz pokazivača na automate. Svaki objekat klase CAutomata sadrži niz pokazivača na funkcije prelaza koje opisuju stanja u tom automatu, niz pokazivača na vremenske kontrole pridružene tom automatu i niz pokazivača na promenljive koje se javljaju kao atributi automata.

Svaki objekat klase CState sadrži niz pokazivača na oblike koji čine dijagram funkcije prelaza, niz pokazivača

na veze koje povezuju oblike u dijagramu i niz pokazivača na promenljive koje se javljaju kao lokalne promenljive funkcije prelaza. Svaki objekat klase CShape sadrži niz pokazivača na linije kao gradivne elemente prikaza oblika. Isto važi i za objekte klase CConnector.



Sl. 3. Višedimenzionalni dinamički nizovi pokazivača predstavljaju osnovu uređenja podataka u prevodiocu

VI. Ispitivanje

Faza ispitivanja prevodioca prožimala se sa samom programskom realizacijom. Zahvaljujući modularnosti rešenja i preciznom poznavanju očekivanih izlaza bilo je moguće po završetku svake programske celine u realizaciji ispitati sve njene metode. Po završenoj programskoj realizaciji prevodilac je ispitivan skupom testnih slučajeva, pri čemu je uočeno da je ispitivanje realnim praktičnim primerima moralo biti dopunjeno velikim brojem testnih slučajeva. Funkcije prelaza kod realnih sistema često su veoma jednostavne i svode se na niz blokova: stanje, ulazna poruka, izlazna poruka, novo stanje. Na takav način se ne proveravaju sve mogućnosti prevodioca.

VII. Zaključak

Realizovan je unos SDL dijagrama, kao i niza potrebnih parametara sistema. Realizovani prevodilac obezbeđuje prevođenje SDL dijagrama u C++ klase zasnovane na FSM biblioteci, kao i potpuni opis SDL dijagrama u XML obliku, a sa ciljem omogućavanja lociranja greške u polaznom projektu na osnovu otkaza u realnom vremenu.

Spoj grafičkog korisničkog razvojnog okruženja sa mogućnostima C++ programskog jezika nudi komforno radno okruženje za razvoj. Sa druge strane, dalji rad na razvoju alata za ispitivanje, kao i međusobna veza alata za ispitivanje i automatskih testova sa SDL dijagramima i generisanim kodom trebalo bi da omogući lako otklanjanje grešaka. Sve ovo bi trebalo da poboljša i ubrza proces razvoja sistema konačnih automata.

Naredni koraci na usavšavanju prevodioca SDL dijagrama bi mogli biti: uvođenje bloka za verifikaciju dijagrama, čime bi se omogućilo ukazivanje na greške nastale nepravilnim unosom dijagrama, odnosno nepoklapanjem unetog dijagrama sa pravilima koji proističu iz SDL standarda i uvođenje sistema upravljanja verzijama.

Literatura

[1] I. Velikić, M. Popović, V. Kovačević, "A Concept of an Integrated Development Environment for Reactive Systems", ECBS, Brno, 2004.

[2] Katedra za računarsku tehniku i računarske komunikacije, FTN, Priručnik radnog okruženja za pisanje protokola, Novi Sad.

[3] I. Velikić, „Jedno rešenje sistemskih podloga programske podrške centra za distribuciju poziva", Magistarski rad, Novi Sad, 2001.

[4] www.itu.int – International Telecommunication Union

Abstract

This paper presents one solution of SDL (Specification and Description Language) diagrams' compiler in the C++ classes based on FSM library. The FSM Library is developed on the Faculty of Technical Science, University of Novi Sad. The compiler translates input SDL diagrams into the output XML (Extensible Markup Language) document, which contains the debugging information.

ONE SOLUTION OF SDL DIAGRAMS' COMPILER

Bogdan D. Trivunović, Miroslav V. Popović,
Ivan S. Velikić, Robert I. Pečkai – Kovač,
Milan M. Ačanski, Slavko M. Svirčević