

Primer 1 - Izrada konačnog determinističkog automata

Funkcionalnost automata je ostvarena u klasi *FiniteStateMachine* iz koje se izvodi klasa svakog tipa automata u sistemu. Klasa *FSMSystem* u sebi sadrži funkcionalnost kontrole i funkcionisanja sistema konačnih determinističkih automata.

Upotreba sistemskih podloga za razvoj konačnih determinističkih automata zahteva da se izvede klasa specifičnog automata iz klase *FiniteStateMachine*, tj. realizuju konkretni tipovi raznih automata.

Realizacija konkretnog tipa automata zahteva da se u izvedenoj klasi definišu sledeće funkcije:

- *GetMessageInterface* – Vraća pokazivač na objekat koji ume da radi sa tekućom porukom. Ako se pojavi poruka u nepoznatom formatu, treba da obradi grešku (obično generiše izuzetak (*exception*)).
- *SetDefaultHeader* – Funkcija postavlja parametre unutar zaglavlja poruke na vrednosti koje su uobičajene za dati tip automata.
- *GetMbxId* – Vraća kod reda u koji se uobičajeno ulančavaju poruke za tip automata koji se razvija.
- *GetAutomate* – Vraća kod tipa automata koji se razvija, obično se koristi za postavljanje podataka u zaglavlju poruke.
- *SetDefaultFSMData* – Poziva se kada treba postaviti podatke instance automata na standardne vrednosti.
- *NoFreeInstances* - Funkcija koja treba da se realizuje u slučaju da automat podržava mehanizam slanja nepoznatom automatu. Ova funkcija će biti pozvana kada nema više slobodne instance automata za novu obradu.
- *Initialize* – Funkcija koja inicijalizuje automat. Treba da inicijalizuje sve strukture potrebne za funkcionisanje automata, kao i podatke specifične za dati tip automata.

U nastavku je dat primer deklaracije klase konačnog determinističkog automata (*AutoExample*) izvedene iz klase *FiniteStateMachine*.

```
class AutoExample : public FiniteStateMachine {
private:
    enum AutoExampleStates { AUTO_STATE0, AUTO_STATE1, AUTO_STATE2 };

    StandardMessage StandardMsgCoding;

    //abstract functions
    MessageInterface *GetMessageInterface(uint32 id);
    void SetDefaultHeader(uint8 infoCoding);

    inline uint8 GetMbxId();
    inline uint8 GetAutomate();
    inline uint8 GetObject(){return 0;} //van upotrebe

    inline void SetDefaultFSMData();
    virtual void NoFreeInstances();

public:
    AutoExample();
    ~AutoExample(){};

    void Initialize(void);
};
```

Primer definicije apstraktnih funkcija klase *FiniteStateMachine* u okviru izvedene klase *AutoExample*.

```
#define AUTOEXAMPLE_MBX_ID    1
#define AUTOEXAMPLE_FSM      1

MessageInterface * AutoExample::GetMessageInterface(uint32 id)
{
    if(id == 0) return &StandardMsgCoding;
    throw TErrorObject( __LINE__, __FILE__, 0x01010400);
}

void AutoExample::SetDefaultHeader(uint8 infoCoding)
{
    SetMsgInfoCoding(infoCoding);
    SetMessageFromData();
}

uint8 AutoExample::GetMbxId()
{
    return AUTOEXAMPLE_MBX_ID;
}

uint8 AutoExample::GetAutomate()
{
    return AUTOEXAMPLE_FSM;
}

void AutoExample::NoFreeInstances()
{
    switch( GetMsgFromAutomate() ){
        case FE2_FSM:
            //report error
            break;
        case FE4_FSM:
            //report error
            break;
        default:
            //report error
            break;
    }
}

void AutoExample::SetDefaultFSMData() {}

void AutoExample::Initialize(void)
{
    SetState(AUTO_STATE0);
    SetDefaultFSMData();
}
```

Zadatak 1

Na osnovu prethodnog primera realizovati klasu *AutoExample* koja predstavlja konačni deterministički automat. Funkcionalnost automata će biti dodata u nastavku vežbe.

Primer 2 – Inicijalizacija sistema

Klasa *FSMSystem* u sebi sadrži osnovu za izvedbu grupe konačnih determinističkih automata. Obavlja sledeće zadatke:

- Vodi računa o tipovima automata u sistemu.
- Određuje instancu automata koja treba da obradi poruku.
- Realizuje prioritete redova poruka.
- Vodi računa o slobodnim instancama automata.
- Realizuje mehanizam slanja nepoznatom automatu.
- Inicijalizuje instance automata.

Pri stvaranju objekta klase *FSMSystem* potrebno je navesti broj tipova automata koji postoji u datom sistemu, kao i broj poštanskih sandučića u sistemu.

```
FSMSystem SystemObject(4/*Broj automata*/, 6/*Broj postanskih sandučića*/);
```

Zatim je porebno navesti veličinu i broj bafera koji će biti korišćeni. Moguće je definisati bafere različitih veličina. To omogućava funkcija *InitKernel*. Prvi parametar je broj različitih tipova bafera. Drugi i treći parametar su pokazivači na nizove. Prvi niz sadrži broj bafera određenog tipa, dok drugi sadrži njihovu veličinu. Poslednji parametar predstavlja broj režijskih poštanskih sandučića korišćen od strane sistema.

```
uint32 buffersCount[8] = { 704, 416, 4 };  
uint32 buffersLength[8] = { 128, 256, 324 };  
uint8 buffClassNo = 3;
```

```
SystemObject.InitKernel( buffClassNo, buffersCount, buffersLength, 5 );
```

U cilju realizacije praćenja događaja bez obzira na mesto zapisa događaja, razvijena je klasa *LogInterface*, koja definiše skup funkcija za registrovanje događaja.

Zahvaljujući informaciji sadržanoj u zaglavlju poruka i objektu klase automata, zabeleženi podaci se mogu lako pretražiti i analizirati kako bi se otkrile greške.

U klasi *FiniteStateMachine* u svim relevantnim tačkama izvršavanja rada automata, pozivaju se odgovarajuće funkcije zapisa događaja.

Iz klase *LogInterface* su izvedene dve klase *UdpLogInterface* i *LogFile*. U navedenim klasama su realizovane funkcije deklarisanе u klasi *LogInterface*. Funkcije *LogFile* klase su realizovane tako da se svi događaji upisuju u tekstualnu datoteku (*log* datoteka). U datoteku se upisuju sledeći podaci:

- Tip automata kome se šalje poruka
- Identifikator automata kome se šalje poruka
- Stanje automata pre obrade poruke
- Tip automata koji je poslao poruku
- Identifikator automata koji je poslao poruku
- Kod poruke
- Sadržaj poruke
- Stanje automata posle obrade poruke

Podaci kao što su: tip automata, stanje automata, kod poruke, se predstavljaju određenim brojnim vrednostima. Ukoliko bi *log* datoteka sadržala samo takve brojne vrednosti, na osnovu nje bi bilo vrlo teško pratiti nastale promene. Zbog toga, klasa *LogFile* koristi inicijalizacionu datoteku (*log.ini*) za prevođenje brojnih vrednosti u odgovarajuće tekstualne poruke i takvu informaciju upisuje u *log* datoteku. U nastavku je dat primer inicijalizacione datoteke klase *LogFile*.

```
[PROJECT]
ProjectName=VCC server
numOfAutoamtes=50
;kodautomata = ime
[AUTOMATES]
0= TIMER SYSTEM
1= FE1 & FE5
2= FE2
3= FE4
4= FE5
;redni broj sloga= kod poruke, ime poruke, kod kodiranja poruke, (kod
;automata,...1)
[MESSAGES]
0=0x00cc, AGENT SET STATE, 0
1=0x1301, L3_ALERTING, 1
2=0x1302, L3_CALL_PROCEEDING, 1
3=0x1307, L3_CONNECT, 1
4=0x130f, L3_CONNECT_ACKNOWLEDGE, 1
;redni broj sloga= kod autoamta, kod vremenske kontrole, automat
[TIMERS]
0= 14, 0, Lapd_T200
1= 14, 1, Lapd_T203
2= 15, 0, L3_T302
```

Slika 1: Primer inicijalizacione datoteke klase *LogFile*

Pri stvaranju objekta klase *LogFile* potrebno je navesti naziv datoteke u koju će biti zapisani događaji, i naziv inicijalizacione datoteke. Podrazumevani nazivi su *default.log* i *log.ini*. Pomoću finkcije *SetLogInterface* sistemu se prijavljuje klasa čije će se funkcije koristiti za praćenje događaja u sistemu.

```
LogFile logObj("default.log", "log.ini");
LogAutomateNew::SetLogInterface(&logObj);
```

Sledeći korak u inicijalizaciji sistema je prijava svih objekata sistemu, pri čemu se specificira za pojedine tipove automata, da li da podrže mehanizam slanja nepoznatom automatu ili ne. Za to se koristi funkcija *Add* klase *FSMSystem*. Funkcija *Add* je predefinisana funkcija. Pri prvom pozivu funkcije za određeni tip automata, pored pokazivača na objekat automata i identifikatora automata potrebno je proslediti broj automata datog tipa kao i parametar koji govori da li navedeni tip automata podržava slanje nepoznatom automatu. Pri prijavljivanju svakog narednog automata datog tipa potrebno je pozvati *Add* funkciju koja ima samo dva parametra (pokazivač na objekat automata i identifikator tipa automata).

```
SystemObject.Add(&felife5[0], FE1FE5_FSM, 100, true);
for(int i=1; i<100; i++) SystemObject.Add(&felife5[i], FE1FE5_FSM);
```

Poslednji korak u inicijalizaciji predstavlja pokretanje sistema. Sistem se pušta u rad pozivom funkcije *Start* klase *FSMSystem*.

```
SystemObject.Start();
```

¹ Navode se svi tipovi automata koji mogu prihvatiti datu poruku

Zadatak 2

Na osnovu prethodnog primera realizovati sistem od tri automata tipa `AUTOEXAMPLE_FSM`.

Primer 3 – Načini slanja poruke

Najčešći oblik funkcije za slanje poruke automatu je *SendMessage(uint8 mbxId, char *msg)*, koja ulančava poruku u odgovarajući red. Oblik funkcije zahteva od programera da prvo formira celu poruku, a zatim da je pošalje odgovarajućem automatu.

Pored samog sadržaja, poruka sadrži i zaglavlje. Sadržaj zaglavlja poruke je definisan automatom koji šalje i automatom koji prima poruku. Podaci o automatu koji šalje poruku su uvek isti, dok podaci o automatu koji prima poruku mogu biti različiti.

U nastavku je dat primer slanja poruke *CHANGE_STATE* prvom objektu automata tipa *AUTOEXAMPLE_FSM*. Slanju poruke prethodi formiranje poruke za šta je zadužena funkcija *PrepareNewMessage*. Prvi parametar funkcije je njena veličina da bi se za njeno pravljenje zauzeo odgovarajući bafer. Ukoliko veličina poruke nije unapred poznata, funkciji je moguće proslediti vrednost nula (0), jer funkcije za dodavanje parametara proveravaju veličinu zauzetog bafera i ukoliko on nije dovoljno veliki zauzimaju novi.

```
PrepareNewMessage(0x00, CHANGE_STATE);  
SetMsgToAutomate(AUTOEXAMPLE_FSM);  
SetMsgObjectNumberTo(1);  
SendMessage(AUTOEXAMPLE_MBX_ID);
```

Funkcije rukovanja sadržajem poruke omogućavaju dodavanje/čitanje parametra poruke. Za dodavanje parametra se koriste sledeće funkcije:

```
uint8 *AddParam(uint8 paramCode, uint8 paramLength, uint8 *param);  
uint8 *AddParamByte(uint8 paramCode, BYTE param);  
uint8 *AddParamWord(uint8 paramCode, WORD param);  
uint8 *AddParamDWord(uint8 paramCode, DWORD param);
```

Navedene funkcije omogućavaju dodavanje parametara različite veličine. Funkcija *AddParam* dodaje parametar proizvoljne veličine. U njoj se pored koda parametra prosleđuje njegova veličina i pokazivač na parametar koji se dodaje. Sve navedene funkcije vraćaju pokazivač na datu poruku.

Analogno funkcijama za dodavanje parametara postoje i funkcije za njihovo čitanje:

```
uint8 *GetParam(uint8 paramCode);  
bool GetParamByte(uint8 paramCode, BYTE &param);  
bool GetParamWord(uint8 paramCode, WORD &param);  
bool GetParamDWord(uint8 paramCode, DWORD &param);
```

Primer slanja poruke sa parametrom:

```
PrepareNewMessage(0x00, CHANGE_STATE);  
AddParamByte(PARAM1, 0x01);  
AddParamWord(PARAM2, 0x11);  
SetMsgToAutomate(AUTOEXAMPLE_FSM);  
SetMsgObjectNumberTo(1);  
SendMessage(AUTOEXAMPLE_MBX_ID);
```

U praksi se pokazalo kao čest slučaj da neki tip automata komunicira sa druga dva tipa automata, u SDL dijagramima uvek označavaju kao levi i desni automat u komunikaciji. Zbog toga su obezbeđene funkcije za slanje levom i desnom automatu u komunikaciji (*SendLeft*, *SendRight*). Pre slanja poruka levom ili desno automatu potrebno je odrediti levi, odnosno, desni automat. Za to se koriste sledeće funkcije :

```
inline void SetLeftMbx(uint8 mbx);  
inline void SetLeftAutomate(uint8 automate);  
inline void SetLeftObjectId(uint32 id);  
  
inline void SetRightMbx(uint8 mbx);  
inline void SetRightAutomate(uint8 automate);  
inline void SetRightObjectId(uint32 id);
```

Podaci potrebni za popunjavanje zaglavlja o levom i desnom automatu se u svakom trenutku mogu promeniti u slučaju da je promenjen automat u komunikaciji.

Primer korišćenja funkcije *SendMessageLeft*:

```
//Definisanje levog automata u komunikaciji  
SetLeftMbx(AUTOEXAMPLE_MBX_ID);  
SetLeftAutomate(AUTOEXAMPLE_FSM);  
SetLeftObjectId(1);  
...  
//Priprema i slanje poruke  
PrepareNewMessage(0x00, CHANGE_STATE);  
SendMessageLeft();
```

Zadatak 3

Na osnovu prethodnog primera realizovati funkciju članicu klase *AutoExample* koja šalje poruku *CHANGE_STATE* istom objektu (izvor i odredište poruke je isti). Funkciju pozvati na kraju inicijalizacije automata (u funkciji *Initialize*).

Primer 4 - Mehanizam obrade poruke

Za svako stanje automata potrebno je realizovati funkcije prelaza, kao članice specijalizovane klase. Registrovanje funkcije koja će se pozvati po prijemu određene poruke u određenom stanju vrši se pomoću funkcije *InitEventProc*.

Funkcija:	Opis:
<i>InitEventProc</i>	Inicijalizuje niz struktura grana stanja automata. Za svaku granu funkcija se poziva jednom. Strukture podataka o granama, stanja automata se redom popunjavaju. Grane bi trebalo inicijalizovati po učestanosti njihovog pozivanja, pošto se zbog algoritma pretraživanja niza brže nalazi grana koja je pre inicijalizovana.
Prametri:	Opis:
<i>uint8 state</i>	Stanje automata na koje se navedena funkcija prelaza odnosi.
<i>uint16 event</i>	Poruka po čijem prijemu će se navedena funkcija izvršiti.
<i>PROC_FUN_PTR fun</i>	Adresa funkcije koja će biti izvršena u datim uslovima.

Primer definicije jednostavne funkcije koja menja stanje automata po prijemu poruke prikazan je u nastavku.

```
void AutoExample::S0_ChangeState(void)
{
    SetState(AUTO_STATE1);
}
```

Ukoliko navedena funkcija treba da se izvrši po prijemu poruke *CHANGE_STATE* u stanju *AUTO_STATE0* potrebno je pozvati funkciju *InitEventProc* sa sledećim parametrima:

```
InitEventProc(AUTO_STATE0,
              CHANGE_STATE,
              (PROC_FUN_PTR) &AutoExample::S0_ChangeState);
```

Uobičajeno je da se poziv funkcije *InitEventProc* nalazi u inicijalizaciji datog automata odnosno u funkciji *Initialize*.

U konstruktoru klase automata potrebno je navesti broj vremenskih kontrola koje automat koristi, broj stanja automata i maksimalan broj funkcija prelaza po stanju.

```
AutoExample::AutoExample() : FiniteStateMachine(
    0, // vremenskih kontrola koje automat koristi
    2, // broj stanja automata
    1) // maksimalan broj funkcija prelaza po stanju
{
    //Ostatak inicijalizacije...
}
```

Po prijemu poruke, na osnovu njenog zaglavlja, moguće je saznati koji automat je poslao poruku. Za to se koriste sledeće funkcije:

```
inline uint8  GetMsgFromAutomate();
inline uint8  GetMsgFromGroup();
inline uint32 GetMsgObjectNumberFrom();
```

Zadatak 4

Na osnovu prethodnog primera realizovati mehanizam obrade poruka. Automat treba iz inicijalnog stanja `AUTO_STATE0` da pređe u stanje `AUTO_STATE1`, po prijemu poruke `CHANGE_STATE`.

Primer 5 – Korišćenje vremenskih kontrola

Signalizacija o isteku vremenske kontrole je realizovana korišćenjem mehanizma za rukovanje porukama. Inicijalizacija vremenske kontrole vrši se pozivom funkcije *InitTimerBlock*.

Funkcija:	Opis:
<i>InitTimerBlock</i>	Funkcija koja vezuje određenu poruku za određenu vremensku kontrolu. Po isteku vremenske kontrole automatu će biti prosleđena navedena poruka
Prametri:	Opis:
<i>uint16 tmrId</i>	Identifikator vremenske kontrole.
<i>uint32 count</i>	Vreme isteka vremenske kontrole.
<i>uint16 signalId</i>	Kod poruke koja će biti prosleđena automatu po isteku vremenske kontrole.

Potrebno je registrovati funkciju koja će biti izvršena po isteku vremenske kontrole, odnosno po prijemu odgovarajuće poruke. Za to koristimo ranije navedenu funkciju *InitEventProc*. Time je inicijalizacija vremenske kontrole završena. Uobičajeno je da se navedena procedura izvrši u okviru *Initialize* funkcije.

Nakon inicijalizacije moguće je koristiti vremensku kontrolu pozivom funkcija

```
void StartTimer(uint16 tmrId, uint8 *info=0);  
void StopTimer(uint16 tmrId);  
void RestartTimer(uint16 tmrId, uint8 *info=0);
```

Napomena: Nakon isteka vremenske kontrole potrebno je pozvati funkciju *StopTimer* ili funkciju *RestartTimer* ukoliko je potrebno ponovo pokrenuti vremensku kontrolu.

U nastavku je dat primer korišćenja vremenskih kontrola.

```
#define TIMER1_ID          1  
#define TIMER1_COUNT      20  
#define TIMER1_EXPIRED    0x20  
  
void AutoExample::S0_TIMER1_Expired()  
{  
    RestartTimer(TIMER1_ID);  
    SetState(AUTO_STATE1);  
}  
  
void AutoExample::S1_TIMER1_Expired()  
{  
    StopTimer(TIMER1_ID);  
    SetState(AUTO_STATE0);  
}  
  
void AutoExample::Initialize(void)  
{  
    InitEventProc(AUTO_STATE0, TIMER1_EXPIRED,  
        (PROC_FUN_PTR)&AutoExample::S0_TIMER1_Expired);  
  
    InitEventProc(AUTO_STATE1, TIMER1_EXPIRED,  
        (PROC_FUN_PTR)&AutoExample::S1_TIMER1_Expired);  
  
    InitTimerBlock(TIMER1_ID, TIMER1_COUNT, TIMER1_EXPIRED);  
}
```

```
    StartTimer(TIMER1_ID);  
}
```

Zadatak 5

Na osnovu prethodnog primera realizovati vremensku kontrolu čije je vreme isteka 5s. Po isteku vremenske kontrole automat prelazi u stanje `AUTO_STATE1`. Vremenska kontrola se pokreće pri inicijalizaciji automata (u funkciji *Initialize*).

Zadatak 6

Na osnovu SDL i MSC dijagrama trećeg primera prve vežbe realizovati automate koji za razmenu paketa koriste mehanizam potvrde paketa.