

OKRUŽENJE ZA REALIZACIJU PROTOKOLA

- ◆ Programska realizacija protokola
- ◆ Jezgro (Kernel)
- ◆ FSM klase

KOMUNIKACIONI PROTOKOL ČINE:

- ◆ Skup poruka sa definisanim formatom (oblikom).
- ◆ Opis obrade pojedinih poruka – skup reakcija komunikacionog procesa.
- ◆ Način obrade grešaka – reakcija na izvanredne događaje.

PROTOKOLI KAO AUTOMATI SA KONAČNIM BROJEM STANJA

- ◆ U programskom smislu protokoli se mogu predstaviti kao automati sa konačnim brojem stanja (FSM – Finite State Machine).

JEZGRO (KERNEL) SISTEMSKE PROGRAMSKE PODRŠKE

- ◆ Osnovna ideja se sastoji u stvaranju jedinstvene podloge za realizaciju automata, komunikaciju između automata i pružanje jedinstvene sprege automata sa sistemom.
- ◆ Cilj je da se automati jednostavno razvijaju i da budu nezavisni od korišćenog operativnog sistema i drugih automata.

PREDNOSTI

- ◆ Razvoj komunikacionih protokola je jednoobrazan, pojednostavljen i smanjen je broj grešaka pri kodiranju.
- ◆ To sve zajedno dovodi do bržeg razvoja željene grupe automata.
- ◆ Vreme koje su projektanti ranije trošili na kodovanje sada mogu da posvete projektovanju i testiranju automata.

REALIZACIJA DETERMINISTIČKIH AUTOMATA

- ◆ Da bi se olakšalo rešavanje problema komunikacionih protokola, predstavljenih skupom konačnih determinističkih automata, uvedene su klase koje obezbeđuju mehanizme rada automata.

PODRŽANI MEHANIZMI RADA AUTOMATA

- ◆ Prelazak iz stanja u stanje.
- ◆ Pozivanje odgovarajuće funkcije obrade.
- ◆ Promena stanja.
- ◆ Sadejstvo više konačnih determinističkih automata.

SISTEMSKE PODLOGE ČINE

- ◆ Rutine jezgra dostupne protokolu.
- ◆ Osnovna klasa za realizovanje konačnih determinističkih automata.
- ◆ Osnovnu klasu sistema konačnih determinističkih automata.

RUTINE JEZGRA (KERNELA)

- ◆ Rukovanje baferima.
- ◆ Rukovanje porukama.
- ◆ Rukovanje vremenskim kontrolama.

KLASA ZA REALIZOVANJE POJEDINAČNIH AUTOMATA OBEZBEĐUJE (1/2)

- ◆ Praćenje stanja.
- ◆ Definisanje funkcija prelaza.
- ◆ Funkcionisanje automata, određivanje funkcije prelaza u zavisnosti od stanja automata i koda poruke.
- ◆ Rukovanje sadržajem poruka, proveru ispravnosti poruka, formiranje novih poruka i pristup parametrima poruka.

KLASA ZA REALIZOVANJE POJEDINAČNIH AUTOMATA OBEZBEĐUJE (2/2)

- ◆ Razmenu poruka između automata.
- ◆ Rukovanje memorijom, zauzimanje i oslobađanje bafera za poruke.
- ◆ Rukovanje vremenskim kontrolama, pokretanje i zaustavljanje vremenikih kontrola.

KLASA ZA REALIZOVANJE GRUPE AUTOMATA OBEZBEĐUJE (1/2)

- ◆ Inicijalizaciju objekata automata, formiranje tabela koje definišu funkcije prelaza, tj. u kojem stanju na koju poruku treba izvršiti određenu funkciju prelaza.
- ◆ Raspoređivanje poruka, određivanje objekta automata, koji treba da obradi poruku, na osnovu podataka u zaglavlju poruke.

KLASA ZA REALIZOVANJE GRUPE AUTOMATA OBEZBEĐUJE (2/2)

- ◆ Definisane politike dodele vremena obrade, tj. koliko poruka iz kog reda sa porukama će biti obrađeno u jedinici vremena.
- ◆ Dodeljivanje objekata automata obradi. U slučaju slanja poruke nedefinisanim objektu, sistem određuje objekat koji će obraditi poruku.

DEFINICIJA APSTRAKTNOG AUTOMATA

$W = (X, Y, S, \delta, \lambda, S_0)$, gde su:

$X = (X_1, X_2, \dots, X_n)$ skup ulaznih signala (ul. azbuka)

$Y = (Y_1, Y_2, \dots, Y_m)$ skup izlaznih signala (izl. azbuka)

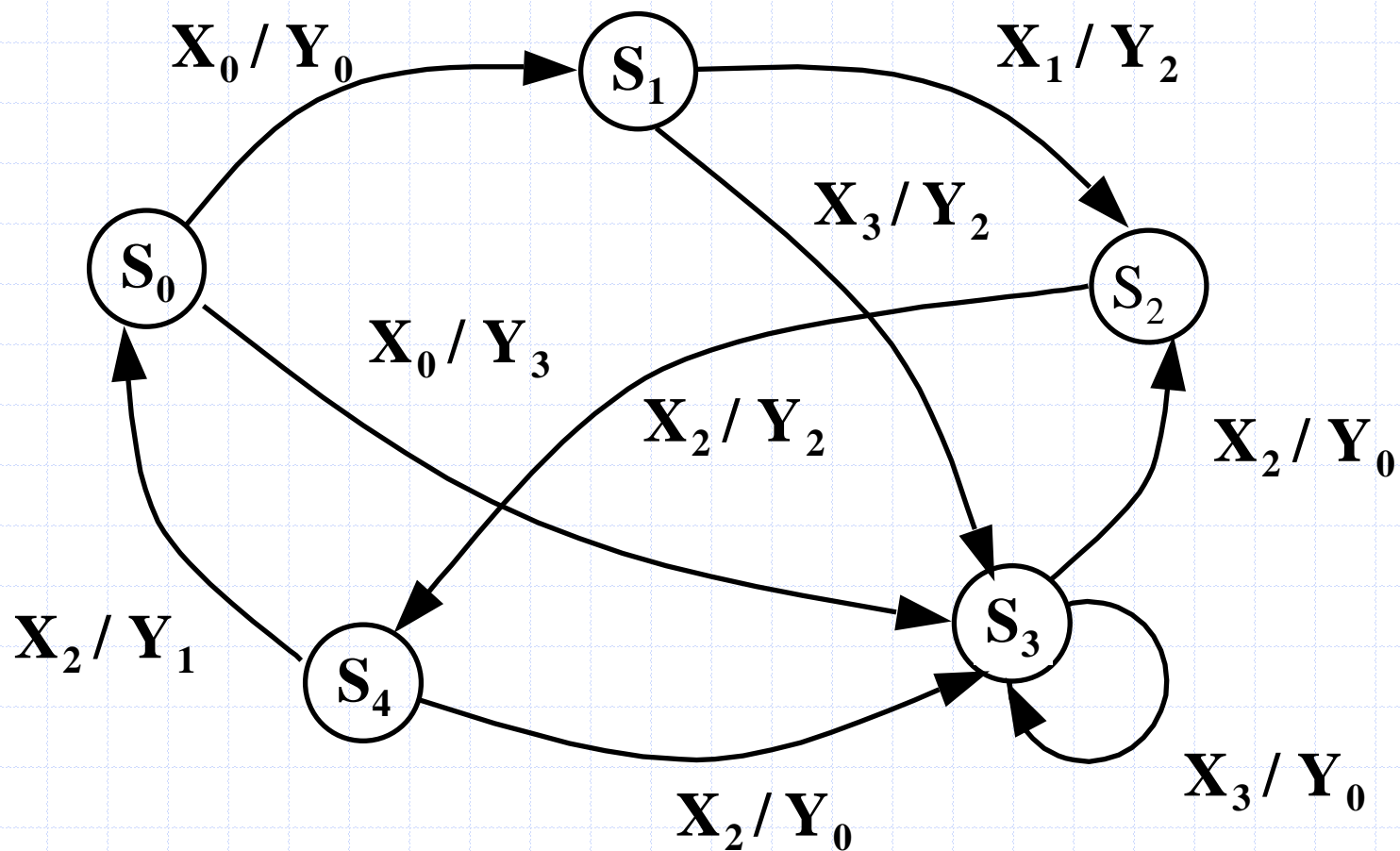
$S = (S_1, S_2, \dots, S_k)$ skup stanja (azbuka stanja)

S_0 je početno stanje.

δ je funkcija prelaza koja preslikava skup $S \times X \rightarrow S$

λ - funkcija izlaza koja preslikava skup $S \times X \rightarrow Y$

PRIMER



PRETHODNA STRUKTURNA REALIZACIJA

```
switch(stanje) {  
  case stanje_1:  
    switch(kod_poruke) {  
      case poruka_1:  
        //obrada poruke  
        break;  
      case poruka_2:  
        //obrada poruke  
        break;  
      case poruka_3:  
        //obrada poruke  
        break;  
      default:  
        //obrada neočekivane poruke  
        break;  
    }  
    break;  
    .....  
  case stanje_n:  
    switch(kod_poruke) {  
      case poruka_1:  
        //obrada poruke
```


Određivanje funkcije obrade poruke

```
PROC_FUN_PTR FiniteStateMachine::GetProcedure(uint16 event,  
uint16 connection)
```

```
{
```

```
.....
```

```
.....
```

```
    uint8 state = m_States[connection]; //odredi stanje konkretnog  
    automata
```

```
.....
```

```
.....
```

```
    for (unsigned i=0; i<m_PStates[state]->NumOfBranches; i++){  
        //odredi funkciju koja treba da obradi poruku
```

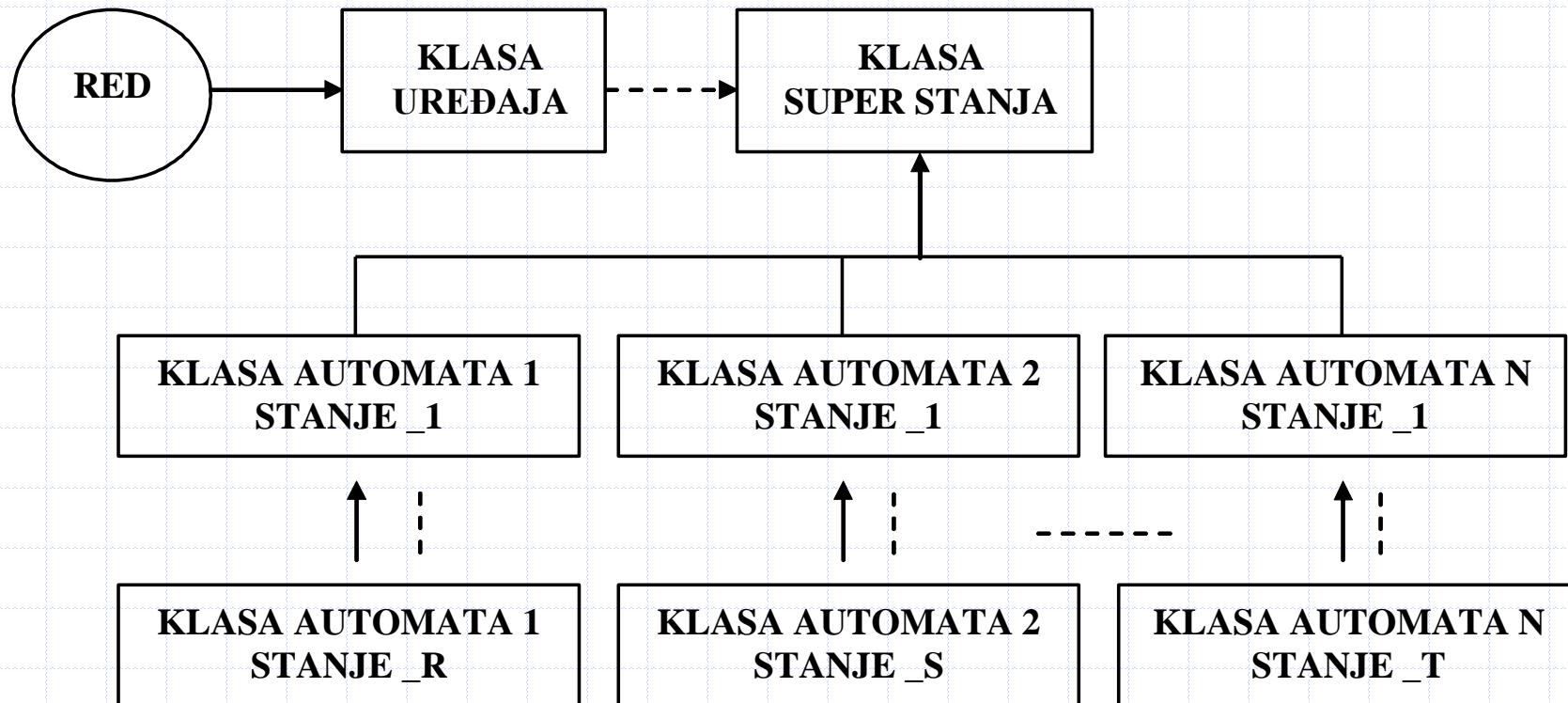
```
        if (m_PStates[state]->PBranch[i].EventCode == event)  
            return m_PStates[state]->PBranch[i].ProcPtr;
```

```
    }
```

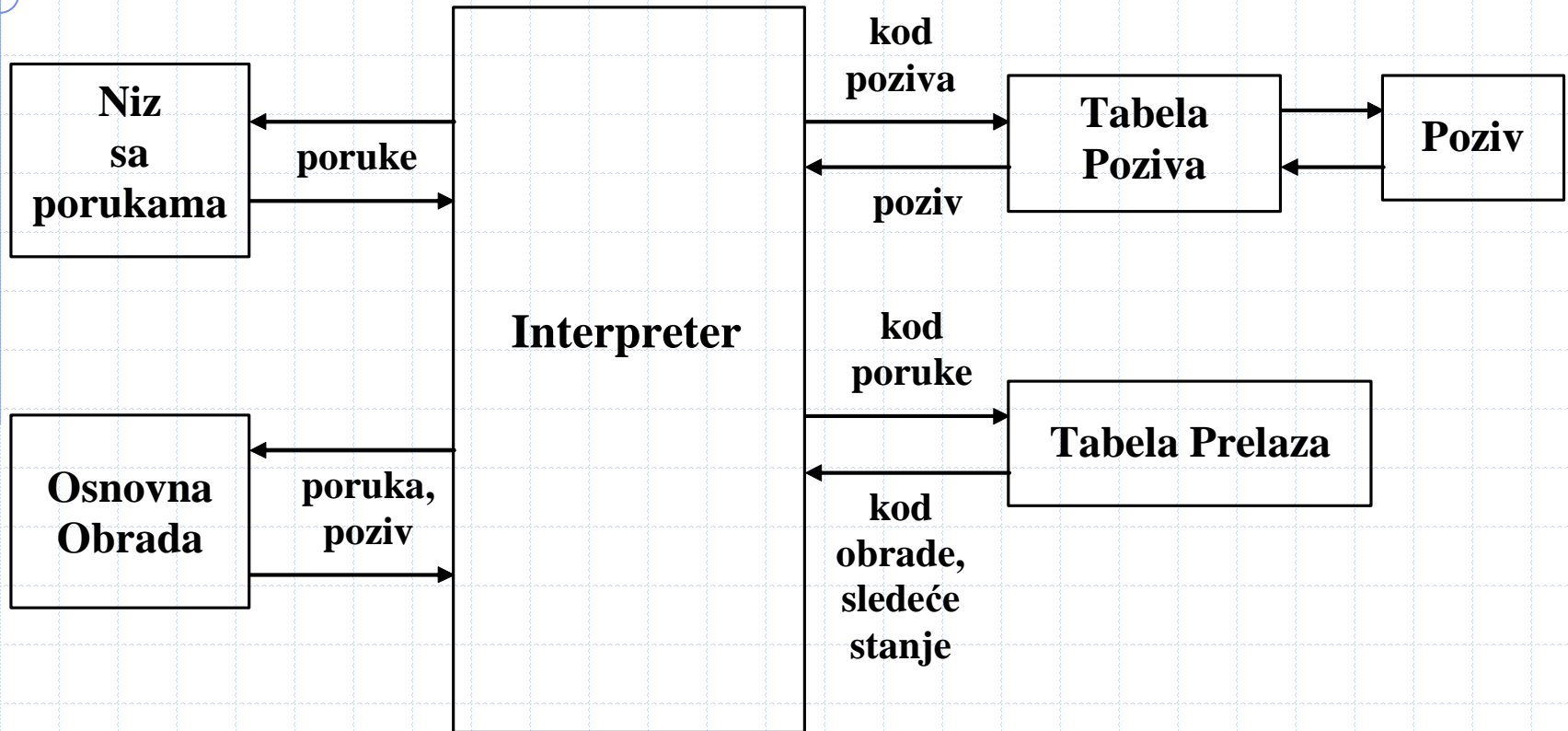
```
    // Ako funkcija nije defnisana, vrati default funkciju za obradu poruka  
    return m_PStates[state]->UnexpectedEventProcPtr;
```

```
}
```

Šema nasljedjivanja u Harellovom pristupu



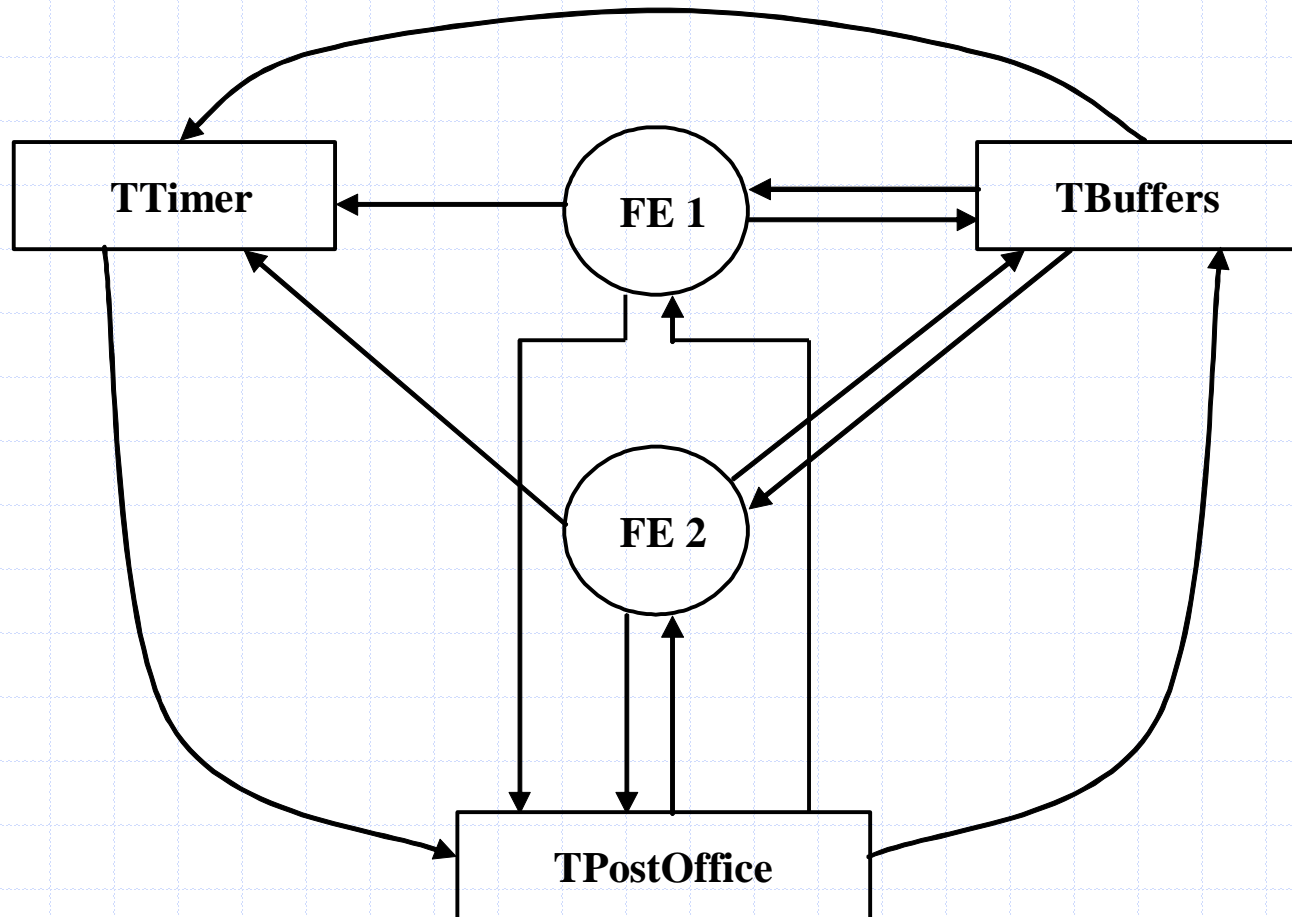
Pristup konfiguracione datoteke



ZAJEDNIČKE SISTEMSKE RUTINE JEZGRA

- ◆ Rukovalac memorijom.
- ◆ Rukovalac porukama.
- ◆ Rukovalac vremenskim kontrolama.

STRUKTURA PROGRAMSKE PODRŠKE



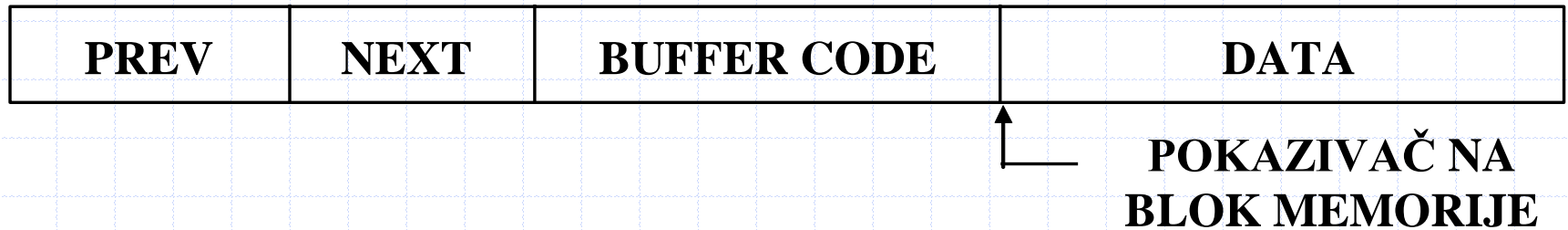
UKOVALAC MEMORIOM

- ◆ Rukovalac memorijom se sastoji iz dve klase *TBuffers* i *TBuffersQueue*.
- ◆ Objekat klase *TBuffers* je objekat koji sistem vidi, tj. preko njega se zauzimaju i oslobađaju baferi. Klasa *TBuffersQueue* je sadržana (agregacija) u klasi *TBuffers*.
- ◆ *TBufferQueue* je lista bafera određene veličine, a klasa *TBuffers* je klasa koja rukuje baferima raznih veličina.

Objekat klase *TBuffers* sadrži N objekata klase *TBuffersQueue*



Struktura bafera



Prev

- Pokazivač na prethodni element u listi.

Next

- Pokazivač na sledeći element u listi.

Buffer Code

- Indeks liste bafera u nizu listi objekata Tbuffers kojoj bafera pripada.

UKOVALAC PORUKAMA

- ◆ Rukovalac porukama ima za zadatak da privremeno sačuva poruke kako bi one kasnije bile predate odredišnom procesu radi obrade.
- ◆ Važna karakteristika rukovaoca porukama je da zadrži hronološki redosled poruka sa jedne strane i da obezbedi obradu poruka po prioritetima sa druge strane.

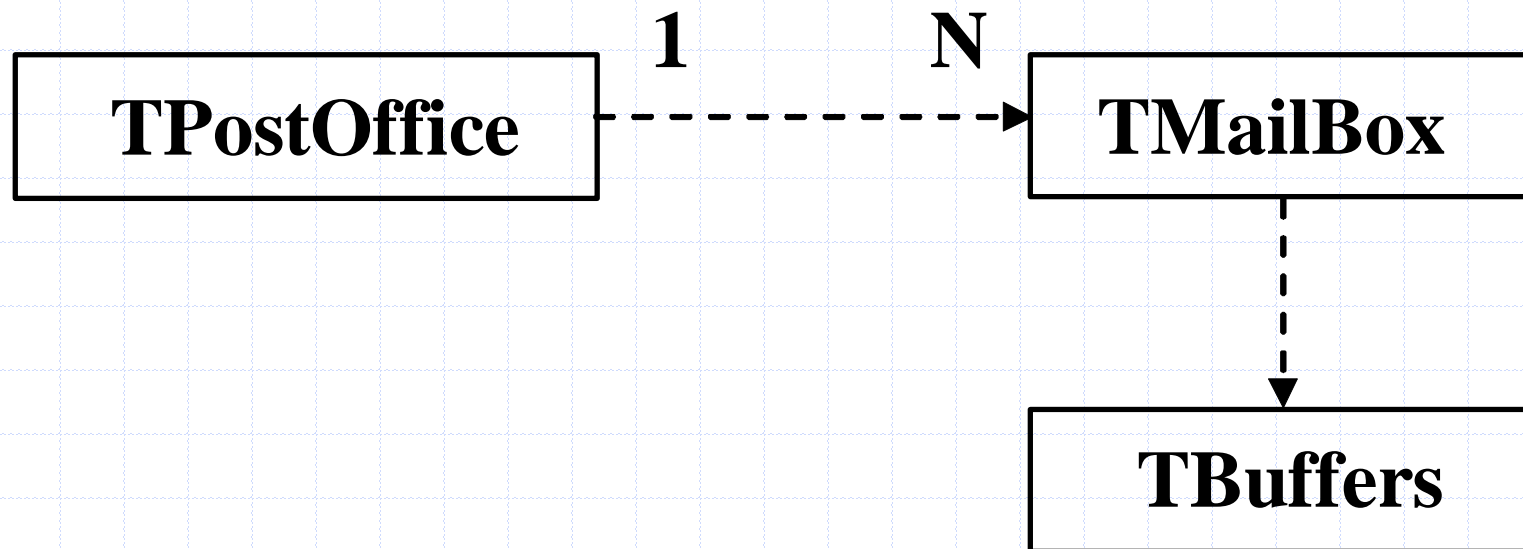
PRIORITET PORUKA

- ◆ Osnovni parametri određivanja prioriteta su hronološki redosled poruka, od kog automata je poruka, za koji automat, a mogu se definisati i prioriteti po vrsti poruka.
- ◆ Politika dodele prioriteta je složena sa stanovišta određivanje formule po kojoj se prioritet računa i ulančavanja poruke na odgovarajuće mesto u listi.

REALIZACIJA RUKOVAOCA PORUKAMA

- ◆ Realizacija rukovaoca porukama je vrlo slična realizaciji rukovaoca memorijom.
- ◆ Uopšteno posmatrano postoji jedna klasa čiji objekat je predstavnik rukovaoca porukama u sistemu, koji u sebi sadrži klasu liste sa porukama, kao niz objekata.

Klase rukovaoca porukama



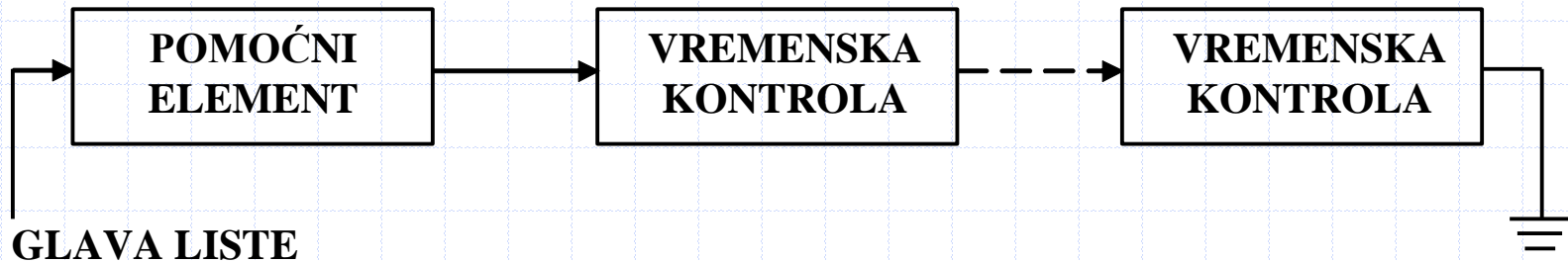
Funkcija *Get* klase *TMailBox*

```
uint8* TMailBox::Get(){
    uint8 *info;
    info = GetFromHead();
    if( info == NULL) return NULL;
    while ( *( (uint16* )(info + MSG_CODE)) == DISCARDED) {
        Buff->RetBuffer(info);
        info = GetFromHead();
        if(info == NULL) break;
    }
    return info;
}
```

RUKOVALAC VREMENSKIM KONTROLAMA

- ◆ Održava listu vremenskih kontrola.
- ◆ Sama lista pokrenutih vremenskih kontrola je delta lista složena po vremenu isteka kontrole.
- ◆ Prva vremenska kontrola u listi je ona za koju se čeka da prva istekne.

Delta lista vremenskih kontrola



Funkcija Zaustavljanja vremenske kontrole

```
void TTimer::StopTimer(uint8 *timerMessage){  
    if (timerMessage == NULL) return;  
    *( (uint16 *) (timerMessage + MSG_CODE) ) = DISCARDED;  
}
```


KLASE ZA REALIZACIJU AUTOMATA

- ◆ Klasa za realizaciju automata, pored mehanizma samog automata, definiše i jedinstvenu spregu za rukovanje porukama, vremenskim kontrolama, sadržajem poruka i memorijom.
- ◆ Sve ovo je nezavisno od upotrebljenih sistemskih rutina u realnom vremenu.

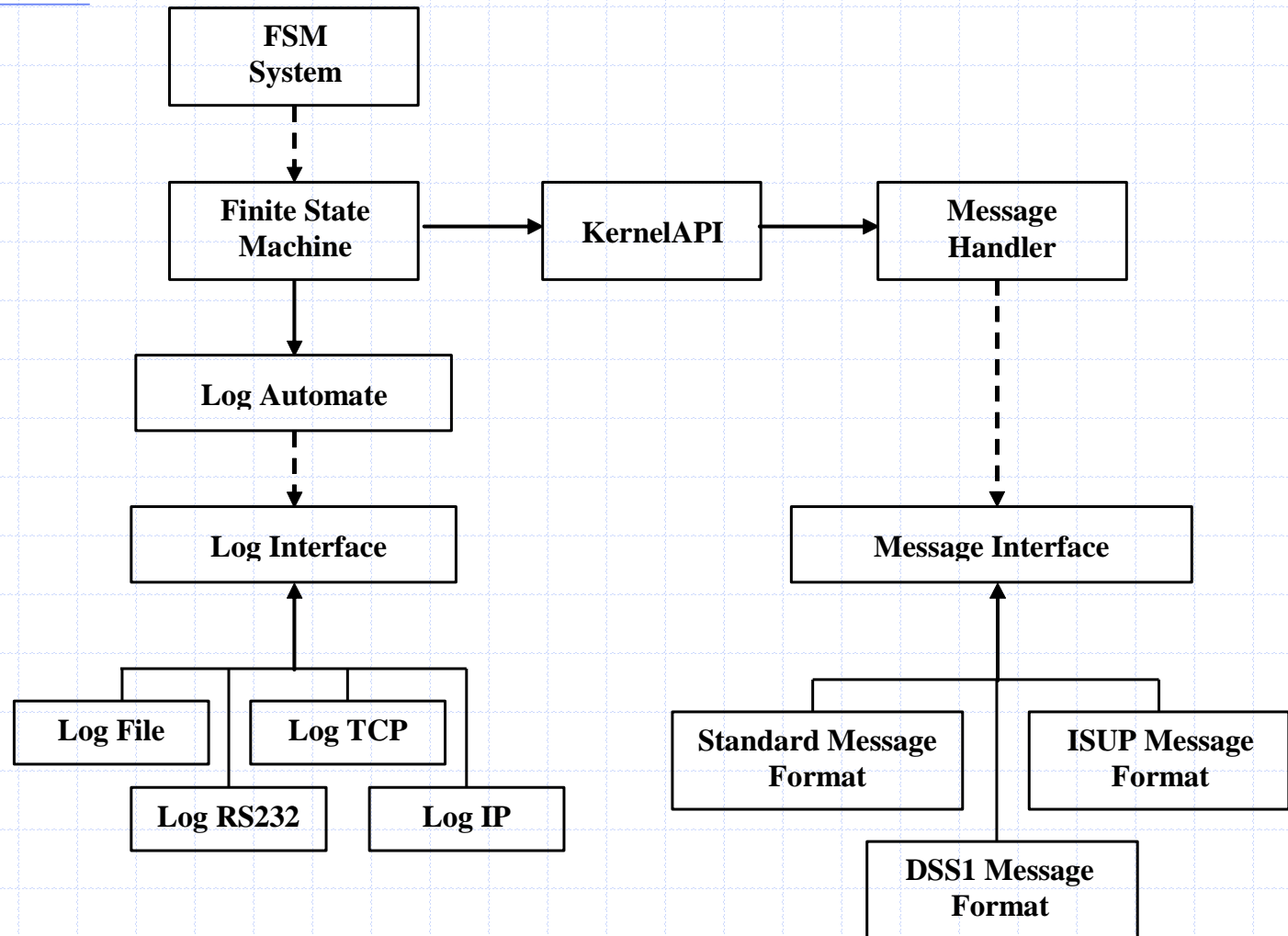
KLASE ZA REALIZACIJU

- ◆ ***MessageHandler*** – za rukovanje sadržajem poruka.
- ◆ ***ProgrammingInterface*** – za prilagođenje jezgru (kernel).
- ◆ ***LogAutomate*** – za praćenje događanja u sistemu (logovanje).
- ◆ ***FiniteStateMachine*** – osnovna klasa za realizaciju automata.

DODATNE KLASSE

◆ ***LogInterface*** i ***MessageInterface*** koje definišu spregu koju sistem determinističkih automata koristi za praćenje događanja u sistemu, odnosno rukovanje sadržajem poruka.

Hijerarhija klasa konačnih determinističkih automata



Klasa *MessageHandler*

- ◆ Ima za zadatak da definiše spregu između automata i poruka.
- ◆ Sprega se sastoji iz dva dela:
 - 1) Rukovanje zaglavljem poruke.
 - 2) Rukovanje sadržajem poruke.

Definicija zaglavlja poruke

```
const uint8 MSG_FROM_AUTOMATE           = 0;
const uint8 MSG_TO_AUTOMATE             = 1;
const uint8 MSG_CODE                    = 2;
const uint8 MSG_OBJECT_ID_FROM          = 4;
const uint8 MSG_OBJECT_ID_TO            = 8;
const uint8 CALL_ID                     = 12;
const uint8 MSG_INFO_CODING              = 16;
const uint8 MSG_LENGTH                  = 17;
const uint8 MSG_INFO                    = 19;
const uint8 MSG_HEADER_END               = MSG_INFO;
const uint8 MSG_HEADER_LENGTH = MSG_HEADER_END;
//parametri potrebni za vremenske kontrole
const uint8 TIMER_MBX_ID                 = MSG_FROM_GROUP;
const uint8 TIMER_COUNT                   = MSG_OBJECT_ID_FROM;
```

Rukovanje sadržajem poruka

- ◆ Svaki parametar u ma kom formatu da je kodovan, može se opisati sa tri podatka:
 - 1) Kod parametra.
 - 2) Dužina parametra.
 - 3) Sam parametar (engl. payload).

Deklaracija funkcija za rad sa sadržajem poruke

//izvršavaju se nad primljenom porukom

uint8 *GetParam(uint8 paramCode);

bool GetParamByte(uint8 paramCode, BYTE ¶m);

bool GetParamWord(uint8 paramCode, WORD ¶m);

bool GetParamDWord(uint8 paramCode, DWORD ¶m);

//izvršavaju se nad novoformiranom porukom

**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);

bool RemoveParam(uint8 paramCode);

Sprežna klasa *KernelAPI*

- ◆ Sprežna klasa (*KernelAPI*) definiše jedan generalizovani skup funkcija za rad sa jezgrom, koji sakriva implementaciju jezgra od automata.

Funkcije klase *KernelAPI*

- ◆ Funkcije inicijalizacije.
- ◆ Funkcije za rad sa memorijom.
- ◆ Funkcije za rukovanje porukama.
- ◆ Funkcije vremenikih kontrola.

Funkcije za rad sa memorijom

```
uint8 *GetBuffer(uint32 length);  
void RetBuffer(uint8 *buff);  
bool IsBufferSmall(uint8 *buff, uint32 length);  
uint32 GetBufferLength(uint8 *buff);
```

Funkcije za razmenu poruka

```
void Discard(uint8* buff);  
void SetMessageFromData();  
void SendMessage(uint8 mbxId);  
void SendMessage(uint8 mbxId, uint8 *msg);  
void SendMessageLeft();  
void SendMessageRight();  
void ReturnMsg(uint8 mbxId);
```

Funkcije vremenskih kontrola

```
uint8 *StartTimer(uint16 code, uint32 count, uint8 *info=0);  
void StopTimer(uint8 *timer);  
bool IsTimerRunning(uint8 *timer);
```

Klasa *FiniteStateMachine*

◆ je osnovna klasa svakog automata u sistemu.

Deklaracija strukture stanja automata

```
struct SState {  
    SState (uint16 maxNumOfProceduresPerState);  
    ~SState ();  
    bool      StateValid;    //if true, data are valid  
    unsigned short NumOfBranches; //number of branches  
    //procedure for proccessing unexpected message  
    PROC_FUN_PTR  UnexpectedEventProcPtr;  
    SBranch*      PBranch;  //pointer to data for each branch  
};
```

Deklaracija strukture grane automata

```
struct SBranch {  
    uint16    EventCode;    // code of event, message code  
    PROC_FUN_PTR ProcPtr;  // pointer to the processing function  
};
```


Deklaracije raznih podataka automata

```
class FiniteStateMachine : public KernelAPI, LogAutomate {  
private:
```

```
.....
```

```
    uint16 NumOfStates;           // Number of different FSM states  
    uint16 NumOfTimers;          // Number of different timers  
    uint16 MaxNumOfProcPerState; //  
    SState *States[MAX_STATE_NO]; // States with procedure ptrs  
    uint32 ConnectionId;         // Id of current connection  
    uint32 CallId;               // id of current call  
    //automate state dependent data  
    uint8 State;                 //state of automate
```

Funkcije i podaci automata

```
//Left automate DATA, used for current call
uint8 LeftMbx;           //left mbx id
uint8 LeftAutomate;      //left automate group
uint8 LeftGroup;         //left object type
uint32 LeftObjectId;     //number of left object
//Right automate DATA, used for current call
uint8 RightMbx;          //right mbx id
uint8 RightAutomate;     //right automate group
uint8 RightGroup;        //right object type
uint32 RightObjectId;    //number of right object
inline void SetLeftMbx(uint8 mbx);
inline void SetLeftAutomate(uint8 automate);
inline void SetLeftObject(uint8 group);
inline void SetLeftObjectId(uint32 id);
inline void SetRightMbx(uint8 mbx);
inline void SetRightAutomate(uint8 automate);
inline void SetRightObject(uint8 group);
inline void SetRightObjectId(uint32 id);
inline uint8 GetLeftMbx();
inline uint8 GetLeftAutomate();
inline uint8 GetLeftGroup();
inline uint32 GetLeftObjectId();
inline uint8 GetRightMbx();
inline uint8 GetRightAutomate();
inline uint8 GetRightGroup();
inline uint32 GetRightObjectId();
```

Funkcije inicijalizacije i kontrole automata

```
virtual void Initialize(void) = 0;
void InitEventProc(uint8 state, uint16 event, PROC_FUN_PTR fun);
void InitUnexpectedEventProc(uint8 state, PROC_FUN_PTR fun);
virtual void NoFreeInstances() = 0;
void FreeFSM();
PROC_FUN_PTR GetProcedure(uint16 event);
FiniteStateMachine(
    uint16 numOfTimers = DEFAULT_TIMER_NO,
    uint16 numOfState = DEFAULT_STATE_NO,
    uint16 maxNumOfProceduresPerState =
        DEFAULT_PROCEDURE_NO_PER_STATE);
~FiniteStateMachine();
void Main(void);
virtual void Process(uint8 *msg);
```

Korišćenje sistema automata

◆ Primeri na vežbama.