UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Boštjan Vlaovič

GENERATOR KLICEV ZA TELEFONSKO CENTRALO MLB SI2000 V5

Diplomska naloga

Maribor, september 1999

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO
2000 Maribor, Smetanova ul. 17

Diplomska naloga univerzitetnega študijskega programa

GENERATOR KLICEV ZA TELEFONSKO CENTRALO MLB SI2000 V5

Študent: Boštjan Vlaovič
Študijski program: univerzitetni, Elektrotehnika
Smer: Elektronika

Mentor: izr. prof. dr. Zmago Brezočnik
Somentor: doc. dr. Monika Kapus-Kolar
Lektorica: Vojka Leskovšek, prof.

Maribor, september 1999

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO
2000 Maribor, Smetanova ul. 17

Številka: DE/XZ-97
Datum: 3.10.1997

SKLEP O DIPLOMSKI NALOGI

1. **Boštjan Vlaovič**, absolvent višješolskega študija elektrotehnike, izpoljnjuje pogoje za izdelavo diplomske naloge.

2. **Tema diplomske naloge** je s področja Inštituta za elektroniko pri predmetu MIKRORAČUNALNIŠKI SISTEMI II.

   MENTOR: izr. prof. dr. Zmago BREZOČNIK
   SOMENTOR: doc. dr. Monika KAPUS-KOLAR

3. **Naslov diplomske naloge**

   GENERATOR KLICEV ZA TELEFONSKO CENTRALO MLB SI2000 V5

4. **Vsebina diplomskega dela**
   Proučite delovanje signalizacije za analogne telefonske priključke v centrali MLB SI2000 V5. Izvedite in preizkusite programski generator klicev, realiziran v jezikih SDL in C, ki naj bo umeščen tako, da bodo posegi v obstoječo programsko opremo čm manjši. Uporabniški vmesnik naj omogoča dostop do generatorja klicev na daljavo.

5. Diplomsko nalogo izdelajte skladno z ”Navodila za izdelavo diplomske naloge”in jo oddajte v 3 izvodih do 15/9-1999

PREDSTOJNIK INŠTITUTA DEKAN

ZAHVALA

Na prvem mestu se zahvaljujem podjetju IskraTEL d.o.o, Kranj, ki mi je omogočilo izdelavo diplomskega dela. V času večmesečnega sodelovanja z njimi so me sprejeli kot enakovrednega člana svojega razvojnega oddelka.

Za vso pomoč pri izdelavi diplomske naloge se zahvaljujem svojemu mentorju izr. prof. Zmagu Brezočniku in somentorici doc. dr. Moniki Kapus-Kolar.

Zahvaljujem se vodji razvojnega oddelka za aplikacijsko programsko Opremo (RDSA) IskraTEL-a g. Jožetu Gašpariču za izbiro zanimive tematike ter korektno in prijazno vodenje. Še posebna zahvala pa velja g. Simonu Molki ter njegovi ekipi RDSA7, katere del sem postal v razvojnem obdobju projekta.

Prijateljici Katji Leskovšek se zahvaljujem za njeno podporo in potrpežljivost v mesecih razvoja izdelka in času izdelave diplomske naloge.

Hvaležen sem tudi bratu Janku Vlaoviču, ki je v vseh letih mojega študija poskrbel za marsikatero moje opravilo.

Nenazadnje se zahvaljujem tudi staršema, Evi Dolničar in Miodragu Vlaoviču, ter nju-nima partnerjema Petru Šivicu in Metki Vlaovič za vso podporo in pomoč, ki so mi jo nudili v času študija.

GENERATOR KLICEV ZA TELEFONSKO CENTRALO MLB SI2000 V5

Ključne besede: ***telekomunikacije, testiranje, SI2000, SDL, generator klicev, telefonska centrala***

UDK: *621.395:621.39*

Povzetek

Diplomsko delo predstavlja razvoj programske opreme za telefonsko centralo MLB SI2000 V5. Razvili smo programski Generator klicev*, ki omogoča razvijalcem enostavno in temeljito testiranje svoje programske kode. Do sedaj so razvijalci programske opreme uporabljali samo zunanje testirne naprave. Zaradi visoke cene in skromnega števila naprav jih v podjetju večino časa uporabljajo verifikatorji. Velik poudarek pri razvoju je bila prostorska neodvisnost uporabnika* Generatorja klicev*. Tako lahko* Generator klicev *uporabljamo iz poljubnega delovnega okolja, ki je povezano s telefonsko centralo preko omrežja ethernet. S tem smo omogočili tudi testiranje že vgrajenih central na oddaljenih mestih (Rusija, Turčija, . . .). Uporaba torej ni omejena samo na maketni center podjetja. Ta* Generator klicev *lahko s pridom uporabljajo tudi serviserji. Uporabniški vmesnik nam zagotavlja katerikoli spletni brskalnik. S tem smo zagotovili tudi neodvisnost od operacijskega sistema, ki ga uporablja uporabnik* Generatorja klicev*. V diplomskem delu je na kratko predstavljena centrala SI2000, razvojni jezik SDL ter načrtovanje in tehnična izvedba* Generatorja klicev*. Na koncu dela je predstavljen tudi praktični primer uporabe.*

CALL GENERATOR FOR SWITCH NODE MLB SI2000 V5

Key words: ***telecommunications, testing, SI2000, SDL, call generator, switch node***

UDK: *621.395:621.39*

Abstract

This graduation work introduces development of software for the switch node MLB SI2000 V5. Software Call Generator *has been developed for easy and thorough testing of software developers' program code. Until now developers had to use external testing machines. There are not enough machines for every developer to use, due to their high cost. They are in the hands of validation and testing personnel most of the time. One of the main development objectives was geographical independence of the* Call Generator *user.* Call Generator *can be used from any place in the world. The only demand is Ethernet connectivity with the switch node that we want to test. This enables testing of already installed distant switch nodes in Russia, Turkey, etc. We are not limited to testing areas in the company.* Call Generator *can be also successfully used by the service department. Any web browser can be used for the user interface. This way operating system independence is guaranteed. In the graduation work, switch node SI2000 V5, SDL programming language, planning and development of* Call Generator *are presented. There is also a practical illustration of its use presented in the last part of the document.*

VSEBINA

SEZNAM SLIK

SEZNAM TABEL

UPORABLJENE KRATICE

A/D — *Analog to Digital (analogno-digitalni pretvornik)*

ADT — *Abstract Data Type (abstraktni podatkovni tip)*

ASMI — *Analog Subscriber Module Interface (vmesnik analognega naročniškega modula)*

BRA ISDN — *Basic Rate Access ISDN (osnovni ISDN dostop)*

CAS — *Channel Associated Signaling (signalizacija po dodeljenem kanalu)*

CCITT — *Comité Consultatif International Téléraphique et Téléphonique (mednarodni posvetovalni odbor za telegrafijo in telefonijo)*

CCS — *Common Channel Signalling (signalizacija po skupnem kanalu)*

CDA — *Communications Controller Version A (komunikacijski krmilnik verzije A)*

CLB — *Central Line Module Version B (krmilnik linijskega modula)*

CUG — *Closed User Grups (zaprte uporabniške skupine)*

CVA — *Central VME Processor Unit Version A (krmilni procesor verzije A)*

CVC — *Central VME Processor Unit Version C (glavna vtična enota verzije C)*

D/A — *Digital to Analog (digitalno-analogni pretvornik)*

DECT — *Digital Enhanced Cordless Telecommunications (digitalne izboljšane brezvrvične telekomunikacije)*

DID — *Direct Inward Dialing (izbiranje lokalnih številk)*

DSP — *Digital Signal Processor (digitalni signalni procesor)*

DSS1 — *Digital Subscriber Signaling System No. 1 (digitalna naročniška signalizacija št. 1)*

DTMF — *Dual Tone Multiple Frequency (dvotonsko večfrekvenčno)*

EFSM — *Extended Finite State Machines (razširjeni končni avtomati)*

FIFO — *First In First Out (prvi noter prvi ven)*

HDLC — *High-level Data Link Control (visokonivojsko krmiljenje podatkovne povezave)*

ISDN — *Integrated Services Digital Network (digitalno omrežje z integriranimi storitvami)*

ITU — *International Telecommunication Union (mednarodna telekomunikacijska unija)*

MCA — *Central Module Version A (centralni modul verzije A)*

MLB — *Line Module Version B (linijski moduli verzije B)*

MN — *Management Node (upravljalna enota)*

MSC — *Message Sequence Charts (diagrami poteka sporočil)*

PABX — *Private Automatic Branch Exchange (zasebna avtomatska naročniška centrala)*

PCM — *Pulse Code Modulation (impulzno kodna modulacija)*

PID — *Process Identification Number (identifikacijska številka procesa)*

PLB — *Power and RC Unit Version B (enota napajalnika in generatorja poziv-nega toka verzije B)*

PRA — *Primary Rate Access (primarni ISDN dostop)*

pSOS+ — *Plug-in Silicon Operating System*

SAx — *Analog Subscriber Unit Version x (enota za priključevanje analognih naročnikov verzije x)*

SBx — *Basic Rate Access Subscriber Unit Version x (enota za osnovni naročniški dostop verzije x)*

SCSI — *Small Computer Standard Interface*

SDL — *Specification and Description Language (specifikacijski in opisni jezik)*

SN — *Switch Node (komutacijsko vozlišče)*

SQL — *Structured Query Language (strukturiran povpraševalni jezik)*

SSN7 — *Signaling System Number 7 (sistem signalizacije številka 7)*

TAx — *Analog Trunk Unit Version x (enota z analognimi prenosniki verzije x)*

TPx — *Trunk Primary Rate Access Version x (enota za naročniški simetrični primarni dostop)*

VF — *visoka frekvenca*

VME — *Virtual Memory Extender*

1 UVOD

Namen naloge je predstaviti razvoj *Generatorja klicev* za linijski modul telefonske centrale SI2000.

1.1 Splošno o sistemu SI2000

Telekomunikacijski sistem SI2000 je sodoben podatkovno krmiljen komutacijski sistem namenjen povezovanju analognih in digitalnih ISDN (Integrated Services Digital Network) naročnikov med seboj in v telefonsko omrežje. SI2000 je večmodulna telefonska centrala [19]. Modularnost omogoča prilagodljivost na različne konfiguracijske zahteve. S tem omogoča ekonomično rast omrežja ter prilagodljivost različnim trgom. Poznamo pet različnih osnovnih konfiguracij sistema SI2000:

1. SI2000 velika konfiguracija (large),
2. SI2000 mala konfiguracija (small),
3. SI2000 dostopovno vozlišče (access),
4. SI2000 namenska konfiguracija (dedicated),
5. SI2000 zasebna avtomatska naročniška centrala (PABX — Private Automatic Branch Exchange).

Velika konfiguracija lahko deluje kot končna, lokalna ali vozliščna centrala. Sestavljena je iz centralnega modula in linijskih modulov. Upravljanje in nadzor sta centralizirana. Dovoljuje priklop do 20.000 analognih naročnikov ali 10.000 ISDN naročnikov z osnovnim dostopom (BRA (Basic Rate Access) ISDN). Seveda je mogoča tudi kombinacija obeh.

Mala konfiguracija deluje kot majhna končna centrala. Sestavljena je iz linijskega modula (MLB — Line Module Version B), ki mu je dodana programska oprema z dodano funkcionalnostjo centralnega modula. Prav tako omogoča centralizirano upravljanje in nadzor.

Omogoča priklop 640 analognih naročnikov ali 320 ISDN naročnikov z osnovnim dostopom (BRA ISDN). S spremembo programske opreme se lahko pretvori v dostopovno vozlišče.

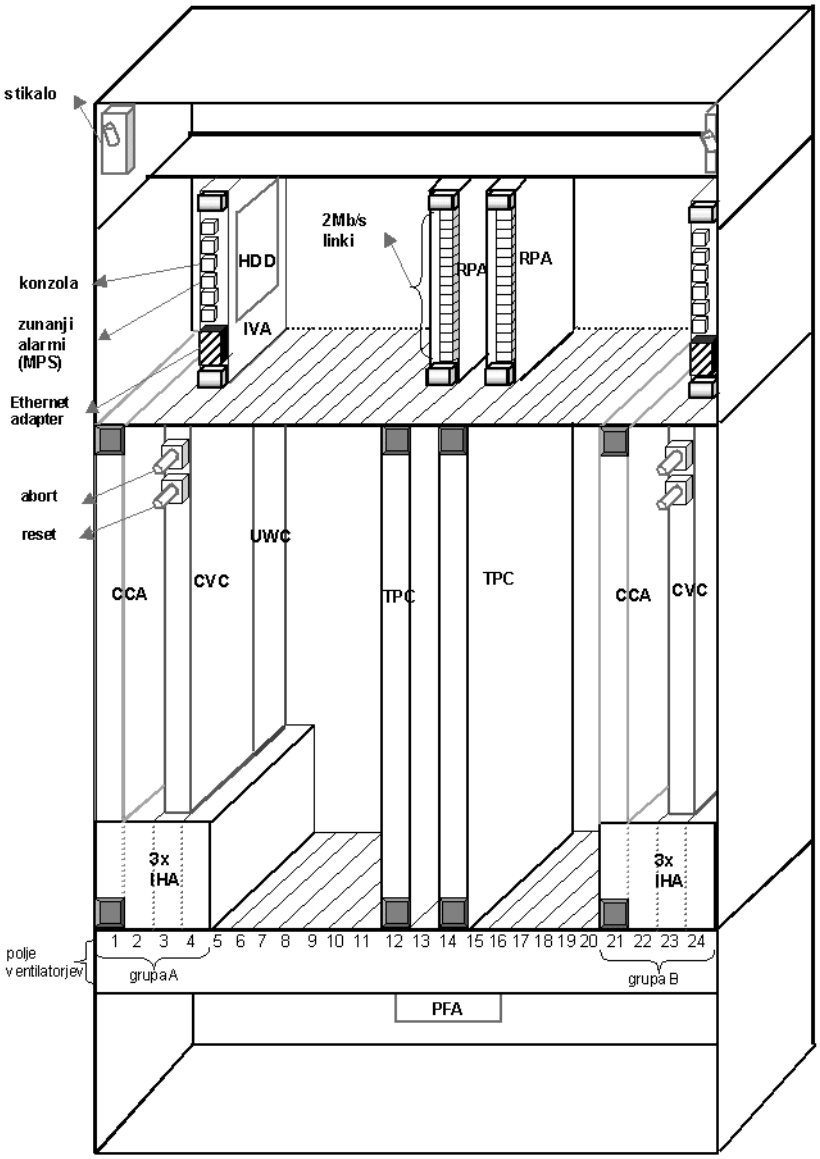
Dostopovno vozlišče tvori le del telefonske centrale. Osnovni del predstavlja linijski modul (MLB). Postavi se lahko v bližnjo soseščino naročnikov. Torej je lahko fizično oddaljen od osnovne konfiguracije telefonske centrale, katere del predstavlja. Omogoča priključitev analognih naročnikov, ISDN naročnikov z osnovnim dostopom (BRA ISDN), dostopovnega vozlišča in PABX central.

Namenska konfiguracija je namenjena podjetjem in institucijam, ki potrebujejo zanesljivo in kvalitetno povezavo med oddaljenimi deli. Omogoča ISDN storitve in podpira sistem brezvrvične telefonije DECT (Digital Enhanced Cordless Telecommunications). Funkcionalno je enaka veliki konfiguraciji. Podpira torej do 20.000 analognih ali 10.000 ISDN naročnikov z osnovnim dostopom (BRA ISDN).

Zasebna centrala ali PABX omogoča neprekinjeno delovanje podjetja. Nahaja se na najnižjem hierarhičnem nivoju namenskih konfiguracij ali kot neodvisna PABX. Osrednji del je linijski modul (MLB) z dodano programsko opremo, ki nudi PABX storitve. Dovoljuje združevanje določenega števila telefonskih priključkov v manjše zaprte skupine (CUG — Closed User Groups), podpira izbiranje lokalnih številk (DID — Direct Inward Dialing), omogoča priklop klicnega centra in zagotavlja tarifiranje lokalnih klicev. Integrirana je lahko tudi v Centrex okolje, ki ga tvori ponudnik javnih telekomunikacijskih storitev.

Materialno opremo sistema SI2000 V5 sestavljajo modul MCA (Central Module version A), moduli MLB in moduli AXM. Centralni modul MCA sestavljajo različne vtične in natične enote (glej sliko 1.1).

MCA modul je krmiljen s procesorjem, ki je v sklopu glavne vtične enote CVC (Central VME (Virtual Memory Extender) Processor Unit Version C). Ta skrbi tudi za nalaganje programske opreme in lokalno podatkovno bazo. V bazo se zapisujejo vse informacije, ki so potrebne za delovanje sistema. ISDN funkcionalnost omogoča modul MLB, ki ga priključimo na modul MCA s signalizacijo SSN7 (Signaling System Number 7) ali DSS1 (Digital Subscriber Signaling System No. 1) — PRA (Primary Rate Access). Modul MLB se lahko uporabi tudi samostojno za razširitve sistema SI2000 V4 s funkcionalnostjo SSN7 in za priključevanje ISDN naročnikov.

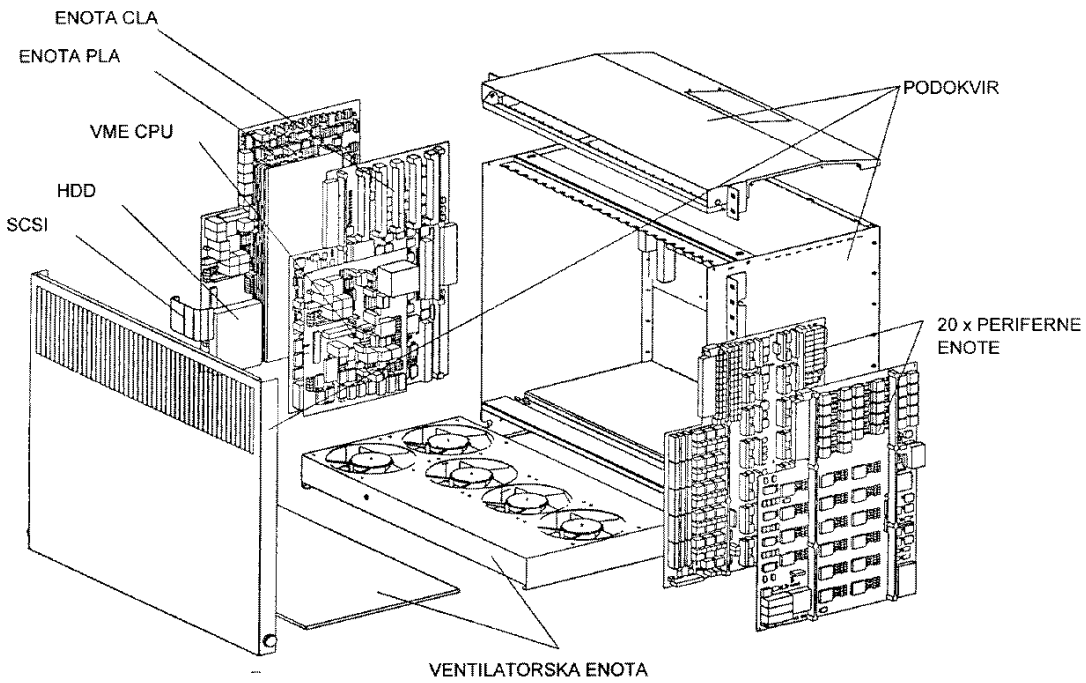


Slika 1.1: Pogled na podokvir modula MCx z vtičnimi enotami

Za upravljanje in shranjevanje konfiguracijskih podatkov se uporablja Management Node (MN). Ti podatki vsebujejo informacijo o naročnikih, tarifiranju, nadzoru delovanja sistema ter prometne in statistične meritve. Začetno nalaganje, administriranje in nadzor centrale opravlja nadzorni računalnik, ki je preko etherneta povezan s krmilnim računalnikom centrale na vtični enoti CVA (VME processor Unit version A).

1.2 Linijski modul verzije B — MLB

Linijski modul verzije B sestavljajo različne vtične in natične enote (glej sliko 1.2). Natična enota na zadnji strani okvirja ima 24 vtičnih mest (glej sliko 1.3). Prvi dve vtični mesti sta rezervirani za krmilnik linijskega modula (CLB — Central Line Module Version B). Naslednji mesti zasedata enoti napajalnika in generatorja pozivnega toka (PLB — Power and RC Unit version B). Preostalih 20 vtičnih mest je namenjenih perifernim vtičnim enotam.



Slika 1.2: Pogled na podokvir linijskega modula — MLB

Vsi konektorji, razen konektorja za CLB, so enaki. Zato je vrstni red vstavljanja perifernih enot poljuben.

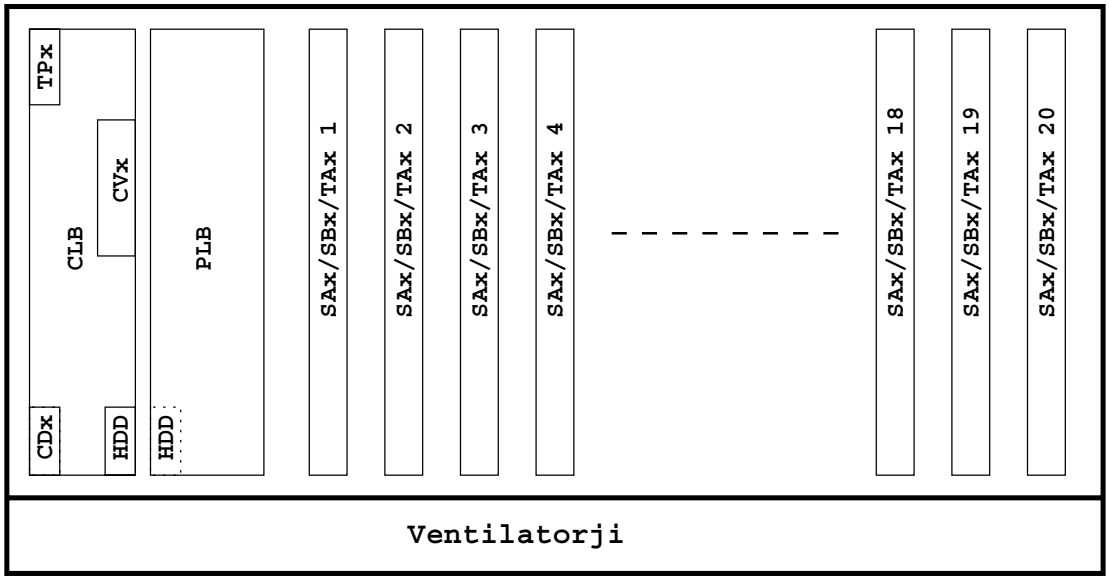
Modul ima naslednje vtične enote:

CVA krmilni procesor,

CDA komunikacijski krmilnik (Communications Controller Version A),

PLB enota napajalnika in generatorja pozivnega toka,

CLB krmilnik linijskega modula,



Slika 1.3: Razporeditev vtičnih enot v linijskem modulu — MLB

SAx enota za priključevanje analognih naročnikov (Analog Subscriber Unit, Version x),

TPx enota za naročniški simetrični primarni dostop (Trunk Primary Rate Access Version x),

SBx enota za naročniski osnovni dostop (Basic Rate Access Subscriber Unit, Version x),

TAx enota z analognimi prenosniki (Analog Trunk Unit, Version x).

Zaradi velike gostote elementov na vtičnih enotah je za normalno delovanje potrebno prisilno hlajenje. Ventilatorji so vgrajeni pod vtičnimi enotami po celi dolžini podokvirja. Število vrtljajev ventilatorjev se krmili s krmilnim procesorjem CVA preko napajalne enote PLB.

Krmilnik linijskega modula — CLB

Krmilnik linijskega modula je osnovna vtična enota linijskega modula (MLB). CLB sestavljajo naslednje enote:

- krmilni procesor (CVA),
- komunikacijski krmilnik (CDA),

- vmesnik za 2 Mb simetrični dostop (TPD),
- trdi disk.

CLB vsebuje komutacijsko polje s 4096 vhodi in enakim številom izhodov. Le-to je enostopenjski polnopropustni časovni multipleks. Komutacijsko polje je opremljeno s PCM (Pulse Code Modulation) RAM pomnilnikom, ki opravlja funkcijo pomnenja podatkov. Komutacijsko polje upravlja glavni procesor preko VME vmesnika.

Na komutacijsko polje se priključujejo:

- centralna procesna enota preko VME vmesnika,
- serijski vmesnik za povezavo s perifernimi enotami (16 Mb/s),
- komunikacijski krmilnik,
- vmesnik za primarni dostop.

Serijska vodila med enoto CLB in perifernimi enotami so povezana po principu zvezde, kar močno zmanjšuje medsebojne vplive med enotami v primeru izpadov in povečuje zanesljivost sistema. Serijski vmesnik povezuje:

- komutacijsko polje s PCM govornimi kanali perifernih priključkov,
- glavni procesor s periferijo.

Krmilni procesor (CVA) MVME162LX power PC je preko adapterja med VME in lokalnim vodilom vezan na posamezne funkcijske sklope osnovne enote (CLB). CVA izvaja naslednje naloge:

- V modulu opravlja večino kontrolnih funkcij, tako signalizacij, kot procesiranje poziva ter krmiljenje periferije.
- Vse aplikacije tečejo v okviru operacijskega sistema pSOS+ (Plug-in Silicon Operating System), ki skrbi za pravilno in pravočasno razporeditev opravil posameznim zvezam.
- S terminalom za upravljanje in vzdrževanje (MN) komunicira preko RS232 ali vodila ethernet.

- Preko vmesnika SCSI (Small Computer Standard Interface) krmili trdi disk.

Komunikacijski krmilnik (CDA) se uporablja za skaniranje analognih naročnikov in CAS (Channel Associated Signaling) signalizacij. Izvaja tudi nadokvirno sinhronizacijo, skrbi za sprejem in oddajo frekvenčne signalizacije, konference ter podatkovne komunikacije po protokolu HDLC (High-level Data Link Control) (32 krmilnikov DSS1, No.7). Enote CDA vsebujejo DSP (Digital Signal Processor) procesor za procesiranje tonskih signalizacij in komunikacijski procesor 68MH360 za procesiranje digitalnih signalizacij tipa CAS, CCS (Common Channel Signaling) in HDLC.

Vmesnik za primarni simetrični dostop (TPD) je namenjen za povezavo z digitalnim omrežjem (PRA, A&CAS, A&CCS, ASMI vmesniki (Analog Subscriber Module Interface)). Funkcije, ki jih opravlja TPD:

- omogoča do dvanajst standardnih priključkov hitrosti 2048 kb/s tipa A ali dvanajst priključkov za primarni ISDN dostop (PRA),
- izloča linijski takt iz vsakega od 12 priključkov in izbira sinhronizacijski izvor,
- zbiranje in posredovanje prekinitev preko VME vmesnika na enoto CLB,
- identifikacijo enote.

Enota napajalnika in generatorja (PLB) skrbi za napajanje posameznih sklopov modula MLB. Zagotavlja tudi pozivni tok analognim naročniškim enotam SAx. Iz vhodne napetosti 48 V se v enoti PLB generirajo naslednje napetosti in signali:

- $\pm 5\ V$,
- $\pm 12\ V$,
- nastavljiva linijska napetost med $-20\ V$ in $-50\ V$ ($-34\ V$),
- pozivna napetost $85\ V$ z odstopanjem $+5\ \%$ in $-20\ \%$ (25 ali 50 Hz) pri izhodnem toku 0–250 *mA*,
- pomožne napetosti za naročniška vezja.

Enota PLB je lahko opremljena s tarifnim generatorjem za generiranje tarifnih frekvenc 12 kHz ali 16 kHz.

Periferne enote se vstavljajo v podokvir linijskega modula z leve proti desni za centralnima enotama. Kot smo že omenili, je v modulu prostora za 20 perifernih vtičnih enot. Zvezdna arhitektura povezav zmanjšuje medsebojni vpliv in dovoljuje zamenjavo periferne enote pod napetostjo. Vsaka periferna enota ima vmesnik za povezavo s krmilnikom linijskega modula.
Lastnosti vmesnika:

- prenosna hitrost je 16 Mb/s dvosmerno,
- 4-žični prenos (ura, okvir, vodilo navzgor, vodilo navzdol),
- osnova je 8-bitno podatkovno vodilo,
- protokol na vodilu je tipa gospodar — suženj.

Vsaka periferna enota ima tudi vgrajen senzor za pravočasno odkrivanje pregrevanja enote in izpada varovalke.

Analogna naročniška enota (SAA) je namenjena za dvožilno (a/b) priključevanje analognih naročnikov. Opremljena je z 32 vmesniki za analogne telefone s tonskim ali impulznim izbiranjem. Vsak vmesnik ima vgrajeno A/D (analog to digital) in D/A (digital to analog) pretvorbo. Povezava periferne enote z delilnikom je dvožilna. Za priklop oddaljenih telefonov je opremljena z releji za priklop višje napajalne napetosti.

Analogna naročniška enota (SAB) je lahko uporabljena v javnem in v PBX sistemu. Enota ima fiksno impedanco naročniškega vezja $600\ \Omega$. Za napajanje kratkih naročniških linij se uporablja linijska napetost (napetostno napajanje), pri večji upornosti naročniške zanke se napajalni mostič samodejno preklopi na baterijsko napetost (tokovno napajanje).

Analogna naročniška enota (SAC) je tretja verzija naročniških enot. Namenjena je za uporabo v javnem sistemu. Nazivna impedanca je v omejenem obsegu lahko realna ali kompleksna. Je programsko nastavljiva. Za napajanje linije se poleg baterijske napetosti uporablja še znižana linijska napetost. Linijska napetost je programsko nastavljiva od $-18\ V$ do $-48\ V$. Preklop med napajalnima izvoroma se izvede avtomatsko v odvisnosti od upornosti naročniške zanke. Naročniško vezje s pomočjo testne točke registrira vse spremembe, ki

se odvijajo na naročniškem priključku (dvig, izbiranje, polaganje, kalibrirna tipka, dvig v fazi poziva). Vsako naročniško vezje je opremljeno z A/D in D/A pretvornikom, ki poleg pretvorbe omogoča programsko nastavljivost vhodnega in izhodnega slabljenja ter vhodne impedance.

ISDN naročniška enota (SBA) je opremljena s 16 S vmesniki za osnovni dostop (BRA). Vmesnik S ima dva programsko nastavljiva načina delovanja:

1. Vmesnik S v načinu LT-S je namenjen za povezavo med različnimi ISDN terminali in sistemom.

2. Vmesnik S v načinu LT-T je uporabljen kot štirižični ISDN prenosnik.

Povezava med S vmesnikom in delilnikom je štirižična. Na osnovni dostop se preko pasivnega vodila lahko veže do 8 različnih ISDN terminalov. V primeru, da je to 8 ISDN telefonov, se dva napajata s periferne plošče, ostali pa preko dodatnega zunanjega vira.

ISDN naročniška enota (SBC) je opremljena s 16 naročniškimi vezji z U vmesniki. Vmesnik U ima dva programsko nastavljiva načina delovanja:

1. V LT načinu je namenjen za povezavo med ISDN terminali in sistemom.

2. V NT načinu se uporablja za testiranje povezave LT-NT.

Na enoti SBC se nahajata dva DC/DC pretvornika napetosti. Prvi iz baterijske napetosti generira napetost ($+5\ V$) za napajanje SBC. Drugi pretvornik zagotavlja linijsko napetost v mejah od $90\ V$ do $106\ V$. Zagotovljen je linijski tok od $50\ mA$ do $60\ mA$ po liniji. Napajanje za posamezno linijo se lahko vklaplja in izklaplja programsko. Povezava med vmesnikom U in delilnikom je dvožična.

Enota analognih javnih prenosnikov (TAA) je opremljena s šestnajstimi prenosniškimi vezji. Vsako prenosniško vezje ima vgrajeno A/D in D/A pretvorbo. Povezava med javnim prenosnikom in delilnikom je dvožična. Z dodatno opremo lahko prenosnik sprejema tarifne signale $12\ kHz$ oziroma $16\ kHz$.

Enota analognih prenosnikov (TAB) je opremljena z osmimi dvosmernimi prenosniškimi vezji. Namenjena je priključevanju VF (visokofrekvenčnih) naprav na sistem SI2000. Med prenosniškim vezjem in delilnikom je v odvisnosti od signalizacije največ osemžična povezava. Za govor se uporabljajo štiri žice z vhodno impedanco 600 Ω. Preostale žice se

uporabijo za signalizacijo. Na enoti so realizirani VF sprejemniki in oddajniki za frekvenčne signalizacije. VF sprejemniki so sestavljenni iz VF A/D pretvornika na osnovni plošči in iz DSP vezja, ki je realizirano na posebni enoti. Le-to je namenjeno potrebam signalnega procesiranja na perifernih ploščah.

2 PROGRAMSKA OPREMA SI2000 V5

V grobem lahko programsko opremo sistema SI2000 V5 razdelimo na programsko opremo za SN (Switch Node) in programsko opremo za MN. Programska oprema za SN je pisana s pomočjo jezika SDL (Specification and Description Language) in se prevaja za operacijski sistem pSOS+, ki je kot operacijski sistem za delovanje v realnem času prisoten na vseh procesorskih ploščah v sistemu. Za razvoj SDL kode uporabljajo v IskraTEL-u orodja francoskega proizvajalca Verilog. Operacijski sistem za nadzorni računalnik je Windows NT.

2.1 Programska oprema za SN

Razdeljena je na dva večja sklopa. Prvi je pisan za krmilni procesor (CVA), drugi pa za komunikacijski krmilnik (CDA). Takšna razdelitev je povsem razumna, saj sta to ločena procesorja.

SDL

Specification and **D**escription **L**anguage je standardiziran formalni jezik za specifikacijo in opis sistemov. Razvil in standardiziral ga je CCITT (Comité Consultatif International Téléraphique et Téléphonique). Danes je zapisan kot standard ITU (International Telecommunication Union) Z.100. Začetek razvoja jezika sega v leto 1972, prva verzija pa je bila izdana leta 1976 (oranžna knjiga). Sledile so verzije v letih 1980 (rumena knjiga), 1984 (rdeča knjiga) in 1988 (modra knjiga). Zadnji dve omenjeni verziji sta vsebovali mnoge razširitve in se smatrata kot prvi zreli verziji. IskraTEL trenutno uporablja za razvoj programske opreme SDL-88. Verziji iz leta 1988 so sledile še nove, objektno orientirane verzije v letih 1992 in 1996.

Stabilnost jezika je pri opisih velikih sistemov posebnega pomena. Zato so vsi sistemi, ki so opisani s SDL-88, pravilni tudi v novejših verzijah jezika. Za natančnejši pregled priporočamo [1, 2, 3, 4, 5, 15] .

Glavne lastnosti SDL-ja:

- Primernost za sisteme, ki tečejo v realnem času.
- Grafična predstavitev programske kode.
- Model je zasnovan na med seboj komunicirajočih procesih. Procesi so opisani s pomočjo razširjenih končnih avtomatov — EFSM (Extended finite state machines).
- Jezik je objektno orientiran.
- Uporaben je vse od zapisa zahtev do končne implementacije.

Pri opisu sistema s SDL-jem navadno začnemo z neformalnim opisom na zelo visokem nivoju abstrakcije. Ta specifikacija nudi osnovo za nadaljnji razvoj. Med razvojem se sistem opisuje vedno bolj podrobno. V končni fazi razvoja so formalno opisani vsi deli sistema. Formalno opisane sisteme lahko s posebnimi orodji tudi simuliramo, verificiramo ter avtomatsko generiramo izvršilno kodo za različne ciljne sisteme.

Zgoraj našteto lahko nudi jezik, ki ima:

- nedvoumno, jasno, natančno in konsistentno specifikacijo,
- osnovo za analizo dovršenosti in pravilnosti specifikacije,
- osnovo za določanje skladnosti implementacije s specifikacijo,
- osnovo za določanje skladnosti specifikacij med seboj in
- uporabna računalniška orodja za razvoj, vzdrževanje, analizo in simulacijo specifikacij.

Vse to SDL ima, zato ga je mogoče široko uporabljati v telekomunikacijah. Uporablja se tudi v mnogih drugih panogah, od letalske industrije do sistemov za pakiranje.

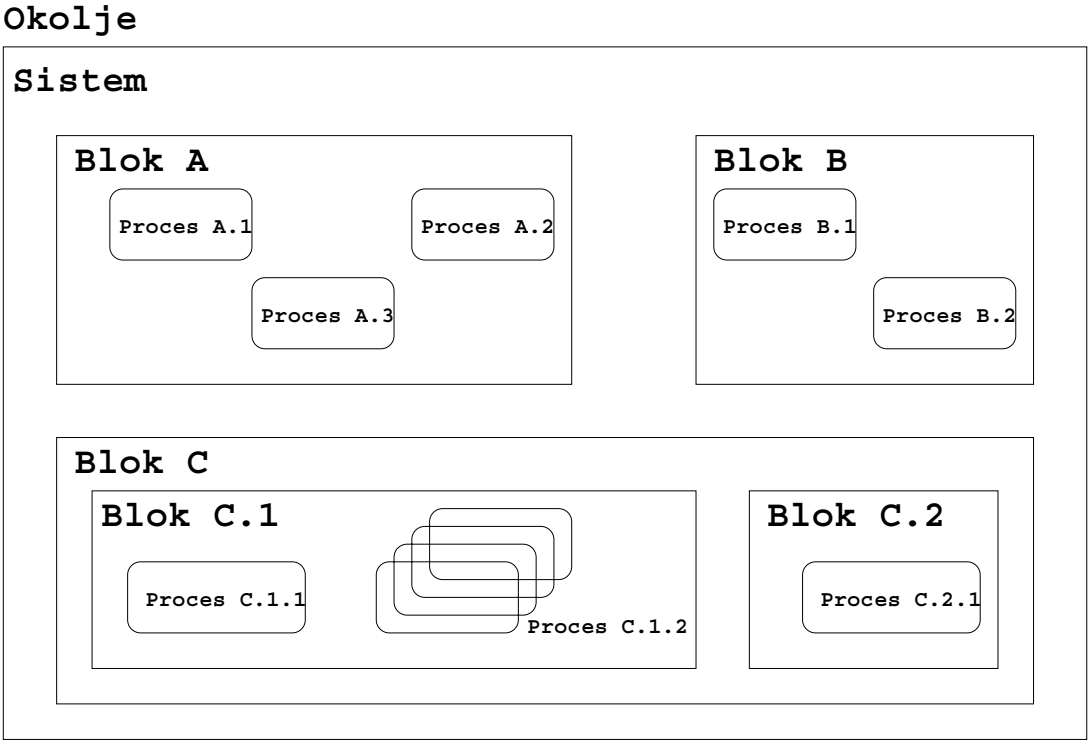
Objekt, ki ga želimo opisati, imenujemo `SISTEM`. Pri opisu sistema s SDL-jem nas prvenstveno zanima njegovo obnašanje. Torej, kaj sistem počne in pri kakšnih pogojih. Vse, kar je zunaj sistema, predstavlja okolje. SDL smatra okolje za črno škatlo, ki spoštuje omejitve sistema. Sistem je lahko zaprt ali odprt. Odprt sistem z okoljem komunicira. Sistemi v telekomunikacijah so odprte narave.

Zaradi lažjega razumevanja besedila, ki sledi, bomo preleteli osnovne gradnike in koncepte SDL-ja.

Za opis objektov v SDL-ju obstajata dva sintaktično ekvivalentna zapisa: grafični in tekstovni. Opis objektov lahko razdelimo na tri sklope:

1. arhitektura sistema,
2. opis procesov,
3. določitev podatkovnih tipov.

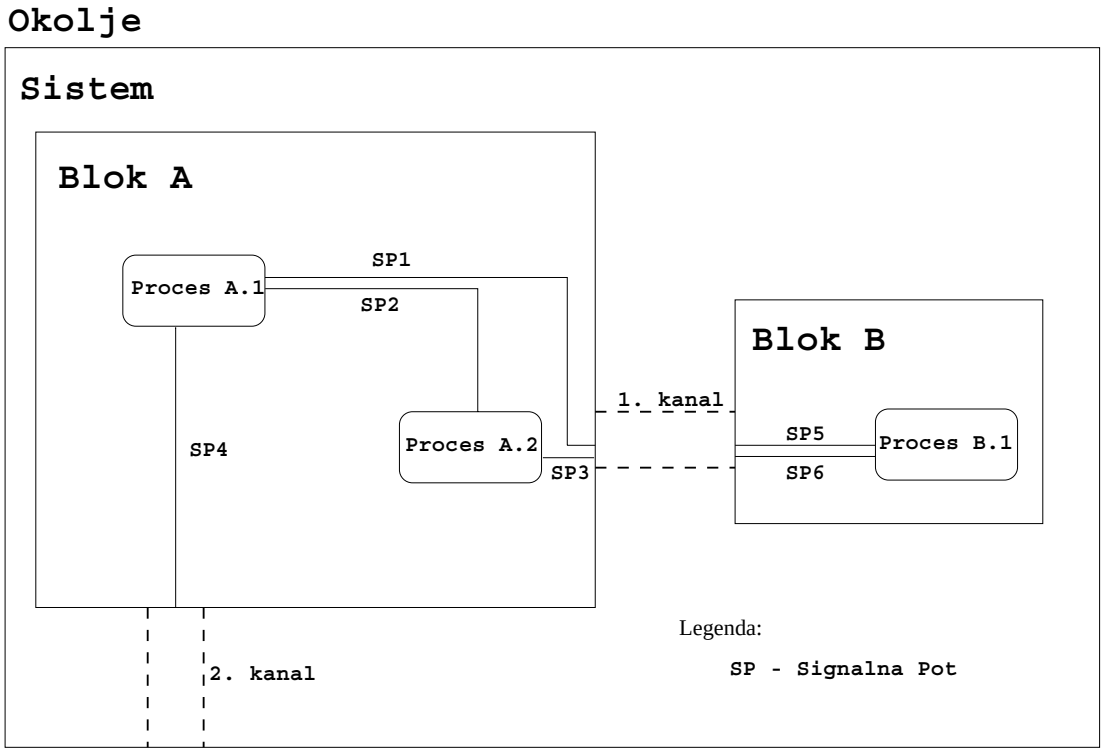
Najvišji nivo arhitekture predstavlja objekt `SISTEM`. Sistem je grafično predstavljen kot pravokotnik. Sistem je to, kar SDL specifikacija poskuša definirati. Vse, kar je zunaj sistema (pravokotnika), je okolje (glej sliko 2.1).



Slika 2.1: Struktura SDL sistema

Sistem razgradimo na manjše enote, ki jih imenujemo `BLOK`. Blok predstavlja svojevrsten podsistem. S tem olajšamo opis velikih sistemov. Bloki so med seboj neodvisni. Vsak blok je lahko nadalje razdeljen na podbloke ali pa vsebuje enega ali več `PROCES`ov. Blok tako predstavlja primeren način za združevanje procesov v logične skupine, kot tudi mejo za vidnost podatkov in podatkovnih tipov.

Znotraj bloka, kot tudi z okoljem[1], si procesi izmenjujejo sporočila s pomočjo signalov. Slika 2.2 prikazuje komunikacijo med tremi procesi in okoljem. Procesi so med seboj povezani s signalnimi potmi. Bloki so med seboj povezani s kanali. Kanal lahko prenaša več signalnih poti. Tudi sistem in okolje sta povezana s kanalom. Teoretično zadostuje že en kanal, praktično pa je kanalov toliko, kolikor je logičnih povezav sistema z okoljem. Kanali in signalne poti so lahko enosmerni ali dvosmerni.



Slika 2.2: Prikaz komunikacije znotraj SDL sistema

Signal je najnižji objekt pri komunikaciji in je definiran z imenom. V primeru, da prenaša podatke, so poleg imena definirani tudi podatkovni tipi prenašanih podatkov. Signal je lahko definiran na nivoju sistema, bloka ali procesa. Signali, ki so definirani na določenem hierarhičnem nivoju, se lahko uporabljajo tudi v vseh nižjih nivojih. Tako se lahko signal, ki je definiran znotraj procesa, uporablja samo med različnimi instancami tega procesa. Signal, ki je definiran na nivoju sistema, pa je uporaben znotraj celotnega sistema, torej povsod. V praksi je pametno upoštevati princip lokalnosti in signal definirati na najnižjem možnem nivoju.

[1] Vse, kar je zunaj mej bloka, je za blok okolje.

Obnašanje sistema določajo procesi. Procese opisujemo s pomočjo razširjenih končnih avtomatov. Vsak proces je sestavljen iz več stanj. Spremembo stanja povzroči prispeli signal. Vse signale proces pobira iz pridružene vhodne vrste. Vsi signali, ki jih proces sprejme, se postavijo v vhodno vrsto. Signali se obdelujejo po principu FIFO (First In First Out). V novejših verzijah SDL-ja poznamo tudi prioritetne signale. Ob prehodu med stanji se lahko izvedejo različne akcije. Izvrši se lahko računska operacija, starta časovnik, pošlje signal, ...

Med procesi se pogosto prenaša večje število signalov. Zato jih lahko združimo v signalne liste. S tem se predstavitev sistema poenostavi. Zmanjšajo pa se tudi potrebni posegi v opise pri nadaljnjem razvoju sistema, saj se ob spremenjeni signalni listi lahko vse popravi na mestu definicije.

V primeru, ko proces opisuje obnašanje objekta, ki se pojavlja v več kot enem primerku (npr.: telefonski aparat pri opisu telefonske centrale), opisa ni potrebno ponavljati. Takšnemu procesu se dovoli obstoj v več primerkih. Vsi primerki temeljijo na generični definiciji procesa, vsak primerek pa ima svojo podatkovno strukturo. S tem se množijo tudi signalne poti, ki proces povezujejo z okoljem. Primer takšnega procesa je `Proces C.1.2` na sliki 2.1.

Procese med seboj ločimo s PID-i. PID (Process Identification Number) nedvoumno določa proces. V primeru procesa `Proces C.1.2`, ko imamo več primerkov enakega procesa, ima vsak primerek svoj PID. Tako lahko pošiljamo signale samo izbranim primerkom procesa.

Vsak proces lahko vsebuje `PROCEDURE` in `STORITVE`. Procedure so podobne proceduram v drugih programskih jezikih. Omogočajo združevanje kompleksnih delov v celoto z enkratno definicijo in večkratno uporabo. Procedura je lahko definirana znotraj procesa ali znotraj druge procedure. Definicija procedure lahko vsebuje tudi formalne parametre. Formalni parametri vsebujejo sezname spremenljivk, povezanih s tipom spremenljivk. Procedura bere vhodne signale iz vhodne vrste procesa, v katerem je definirana. Vidna je znotraj objekta, kjer je definirana. Klic procedure se lahko izvrši znotraj prehoda med dvema stanjema.

Storitve omogočajo še natančnejšo razgradnjo procesa. Storitev razvojno orodje ne podpira, zato se v našem opisu ne pojavljajo.

Deklaracije spremenljivk in časovnikov se nahajajo v programskih deklaracijah. SDL vsebuje naslednje vnaprej definirane podatkovne tipe:

- INTEGER — decimalna cela števila,
- REAL — realna, decimalna števila brez iracionalnih,
- NATURAL — decimalna naravna števila vključno z ničlo,
- CHARACTER — ASCII znak,
- CHARSTRING — niz znakov,
- BOOLEAN — logični podatkovni tip,
- PID — identifikacijska številka procesa,
- TIME — absolutni čas sistema,
- DURATION — relativni čas sistema,
- ARRAY — seznam,
- STRUCT — definicija nove strukture.

Vrednost spremenljivk PID je lahko konstanta (NULL) ali predefinirana spremenljivka:

- SELF — lastni PID procesa,
- PARENT — PID procesa, ki je ta proces kreiral,
- SENDER — PID procesa, ki je poslal signal, kateri je sprožil zadnji prehod,
- OFFSPRING — PID zadnjega kreiranega procesa.

Poleg teh lahko razvijalec definira svoje podatkovne tipe in operacije nad njimi. Za definicijo novih tipov SDL uporablja pristop abstraktnega podatkovnega tipa (ADT — Abstract Data Type).

Vsi procesi v sistemu tečejo vzporedno in neodvisno drug od drugega. Pri klicu procedure se izvajanje procesa ustavi v točki klica. Ko se program vrne iz procedure, se izvajanje nadaljuje v točki za klicem procedure. Procedure znotraj procesa torej ne tečejo vzporedno.

Iz opisanega je razvidno, da lahko isti sistem opišemo na različne načine. Kako sistem razdelimo, je odvisno od različnih kriterijev:

- definiranje še obvladljivih blokov glede na velikost in funkcionalnost,
- skladnost z dejansko programsko in strojno opremo,
- naravna funkcionalna delitev,
- ponovna uporabnost,
- minimalno število povezav,
- zdrava pamet.

2.2 Testiranje programske opreme

V procesu razvijanja programske kode za telekomunikacijske sisteme je še posebno pomembno enostavno preverjanje funkcionalnosti napisane kode. Za te namene obstaja posebna računalniška in strojna oprema.

Zunanji generatorji klicev delujejo kot množica telefonskih aparatov v eni škatli. Izvajajo vnaprej določene scenarije klicev. S priključitvijo na zunanje priključke centrale omogočajo preizkušanje centrale kot črne škatle. Pri tem znajo preverjati pravilnost poteka linijskih in registrskih signalov na naročniškem ali prenosniškem priključku. Analogne signalizacije priključka se v splošnem delijo na linijski in registrski del. V fazi vzpostavljanja in rušenja zveze sodelujeta oba dela, pri čemer se gradnja zveze začne z linijskimi signali (zaseganje — SZR, potrditev zaseganja — SZR_ACK, pripravljenost na sprejem registrskih signalov, ...). Sledi registrska faza, ki v grobem pomeni prenos številčne informacije (cifer), nato pa ponovno linijska faza (signal končane registrske faze — EOS, signal javljanja — ANSW, signal sproščanja naprej — CLRF, signal sproščanja nazaj — CLRB, dokončni signal sproščanja — RLSG). Našteti signali so dokaj splošni, nekatere signalizacije ne vsebujejo vseh naštetih, druge imajo tudi nekatere dodatne. Pri analogni naročniški signalizaciji prištevamo v linijski del dvig — OFF_HOOK, polog — ON_HOOK, ponovni klic registra s kalibrirano prekinitvijo — CF in ponovni klic registra z ozemljitveno tipko — GND. Registrski del pa so signali cifer, ki so lahko tonfrekvenčni (DTMF — Dual Tone Multiple Frequency) ali impulzni (izbiranje s prekinjanjem tokovne zanke).

Zunanji generatorji klicev preverjajo tudi kakovost govorne zveze. Pomanjkljivosti v funkcionalnem smislu praktično nimajo. Problem je običajno v tem, da jih je omejeno število. Zato ga razvijalec, ko ga potrebuje, nima vedno pri roki. V *IskraTEL*-u jih v glavnem uporabljajo sodelavci v integracijski in verifikacijski skupini. Razvijalci jih pretežno uporabljajo pri iskanju vzrokov napak ali v posebnih primerih preizkušanja delovanja programske opreme. Za generatorje so zadolženi sodelavci verifikacijske skupine. Ostali si jih v dogovoru z njimi lahko občasno sposodijo. Ker se razvijalci zelo pogosto selijo po raznih maketah, je prestavljanje generatorjev in kablov vsaj nadležno in nepraktično.

Vse to nakazuje potrebo po alternativni rešitvi problema testiranja funkcionalnosti programske kode. Če želimo, da bo testiranje lahko izvajal vsak razvijalec, ki bo imel na voljo testno centralo, je najprimerneje, da realiziramo modul za testiranje v centrali sami. Pri tem je potrebno paziti, da modul ne bo vplival na delovanje centrale. Prav tako se ne smejo izvajati nobeni posegi v obstoječo programsko opremo centrale, saj bi s tem vnašali spremembe v okolje, ki ga želimo testirati. Cilj je torej programska rešitev, ki bo razvijalcu omogočala testiranje funkcionalnosti programske kode. S tem se razvijalcu omogoči, da natančneje preveri pravilnost delovanja svoje kode, preden jo uradno izda v programsko knjižnico in s tem v javno uporabo.

Prednosti programskega generatorja klicev:

- Dostopnost vsakomur ob vsakem času (razvijalci, integratorji, verifikatorji, serviserji).
- V kombinaciji z zunanjimi generatorji lahko dosežemo skoraj poljubne obremenitve centrale (tudi največjih konfiguracij).
- Potencialno ga je mogoče uporabljati tudi ob montaži central na objektih (pri kupcu) pred vključitvijo v redno obratovanje. S tem lahko do določene mere preverimo delovanje programskega paketa in vsebine podatkovne baze na terenu.
- Po potrebi je prilagodljiv posebnim zahtevam (ker je naš izdelek in imamo popoln nadzor nad njim).

Pomanjkljivosti programskega generatorja klicev:

- nezmožnost preizkusa vseh delov centrale kot črne škatle, saj se vključuje na programskih in ne fizičnih vmesnikih,

- dodatno obremenjuje procesor,
- omejene zmožnosti preverjanja klicev in zvez (na primer preverjanje govornih zvez),
- verjetnost napak v generatorju znotraj sistema je morda večja od tiste pri zunanjih generatorjih, saj zunanje generatorje poleg proizvajalca preverja bistveno večja in raznolika množica uporabnikov.

3 ANALIZA ZAHTEV

Pred dejanskim kodiranjem *Generatorja klicev* je bilo potrebno izvršiti natančno analizo delovanja sistema na nivoju signalizacije analognega naročnika [18, 8]. To smo naredili s pomočjo simulatorja. Beležili smo vse signale, ki so se sprožili med enostavno vzpostavitvijo zveze med dvema telefonoma. Te signale smo nato s pomočjo programa, ki smo ga napisali, pretvoril v velik MSC (Message Sequence Charts) diagram. Ker je bilo signalov ogromno, smo diagram natisnili na več listov formata A3. Nato smo sledili signalom in si poskušali pojasniti način delovanja telefonske centrale. Na koncu smo prišli do naslednjega spoznanja: če umestimo *Generator klicev* zadosti nizko v programski strukturi telefonske centrale, bo potrebno poznati samo signale na najnižjem nivoju signalizacije. To močno zmanjša potrebno poznavanje signalizacije, saj je na najnižjem nivoju vse dokaj enostavno.

3.1 Umestitev Generatorja klicev v obstoječo programsko opremo

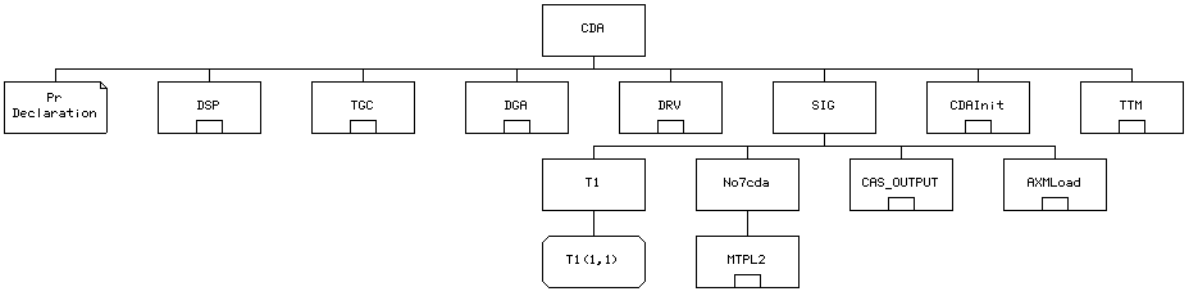
Programska oprema telefonske centrale je razdeljena na dva glavna bloka:

- CVA in
- CDA.

Blok CVA se izvaja na procesorju MVME162LX, CDA pa na procesorju Motorola 68MH360. Torej se programski opremi znotraj omenjenih blokov izvajata na različni strojni opremi. V nadaljevanju bomo v primeru, ko bomo imeli v mislih razlike, ki jih prinaša strojna oprema, govorili o CVA in CDA strani, ko se bomo sklicevali na programsko opremo, pa bomo govorili o blokih CVA in CDA.

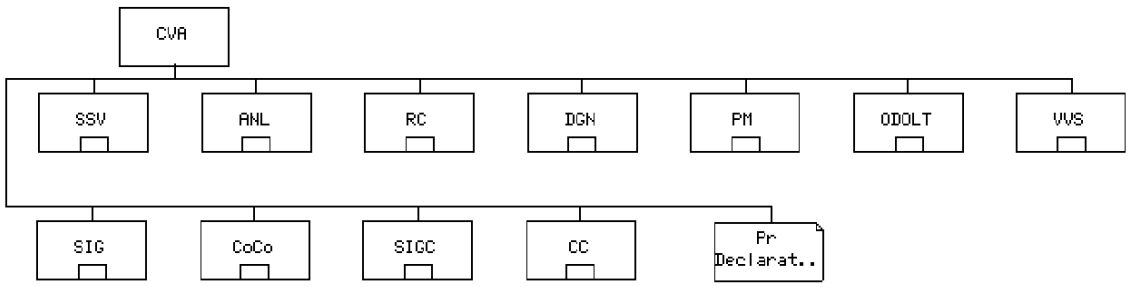
Testiranje analognih telefonskih priključkov izvaja blok CDA. Programsko strukturo bloka CDA prokazuje slika 3.1.

Podbloka, ki nas zanimata, se imenujeta SIG in DRV. Že sami imeni namigujeta, da vsebujeta opise za signalizacijo in gonilnike strojne opreme. Najnižji nivo programske opreme

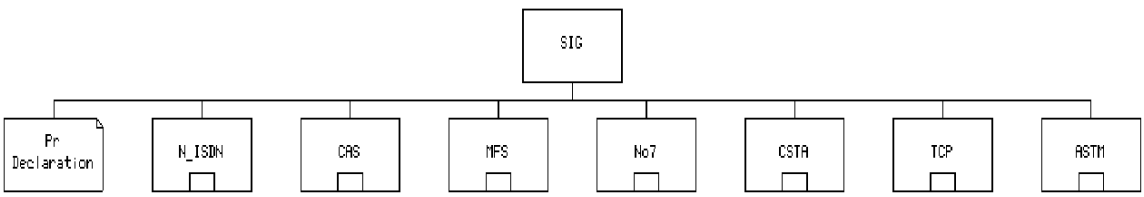


Slika 3.1: Struktura CDA bloka

so gonilniki. Le-ti skrbe za pravilno komunikacijo med strojno in programsko opremo telefonske centrale. Operator, ki skanira stanje analognih priključkov, se imenuje `scandrv`. To je ADT operator napisan v programskem jeziku C. Vsake 4 ms preveri stanja na vhodih analognih priključkov telefonske centrale in morebitne spremembe pošlje v obliki posebnega paketa bloku CAS. Blok CAS se nahaja v bloku CVA, natančneje, v podbloku SIG[2] (glej sliki 3.2 in 3.3). Že samo ime nam pove, da vsebuje kodo za CAS signalizacijo.



Slika 3.2: Struktura bloka CVA



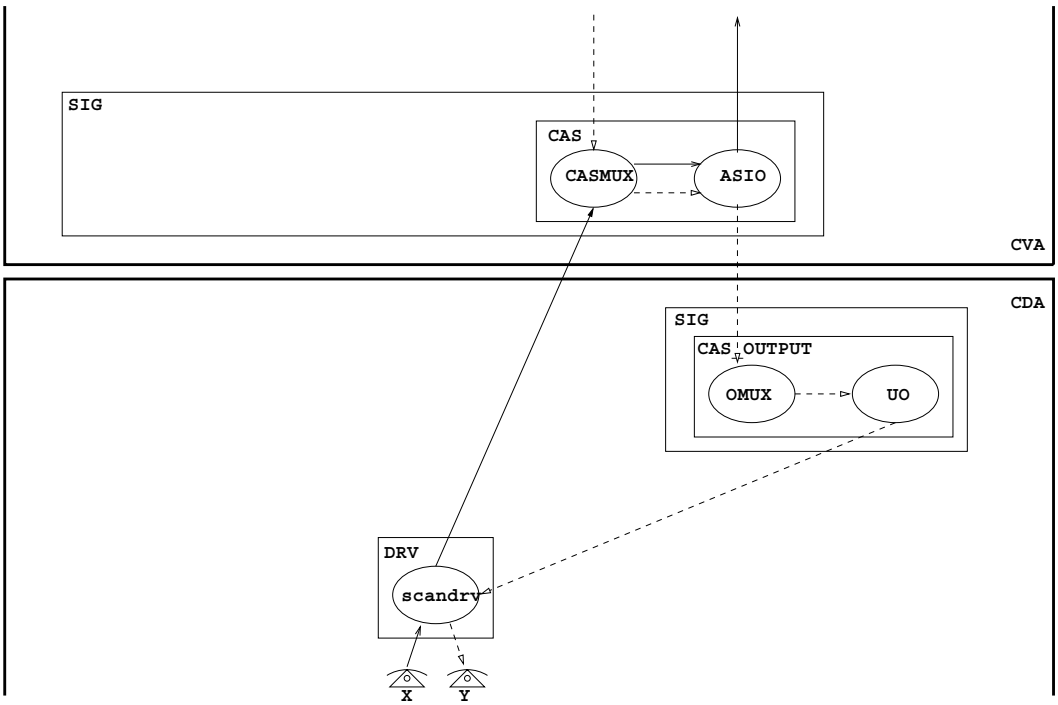
Slika 3.3: Struktura podbloka CVA/SIG

Na tem in na višjih nivojih se nahaja programska koda, ki skrbi za pravilno detekcijo izbiranja, vzpostavitev zveze, tarifiranje in vsa ostala opravila, ki so vezana na posamezen

[2] Bloka CVA/SIG in CDA/SIG sta čisto različna bloka.

telefonski klic, torej programska koda, ki jo želimo testirati. pSOS+ meri čas v TICK-ih. Na strani CVA dobi programska oprema TICK vsakih 10 ms, na strani CDA pa vsaki 2 ms. Naravno mesto za generator klicev bi torej bil nivo `scandrv`-ja. S tem bi lahko zagotovili enako časovno natančnost, kot jo omogoča `scandrv` ob normalnem delovanju centrale. Ker pa ne želimo spreminjati programske kode za vsak test posebej, potrebujemo dostop do baze podatkov. Ta je mogoč samo s strani CVA, torej je potrebno del programske opreme *Generatorja klicev* postaviti tudi na stran CVA.

Najprej si oglejmo, kako poteka vzpostavljanje klica v programski opremi telefonske centrale (glej sliki 3.4 in 3.5).



Slika 3.4: Struktura bloka SIG na strani CVA

1. `scandrv` zazna spremembo v tokokrogu telefonskega priključka X.

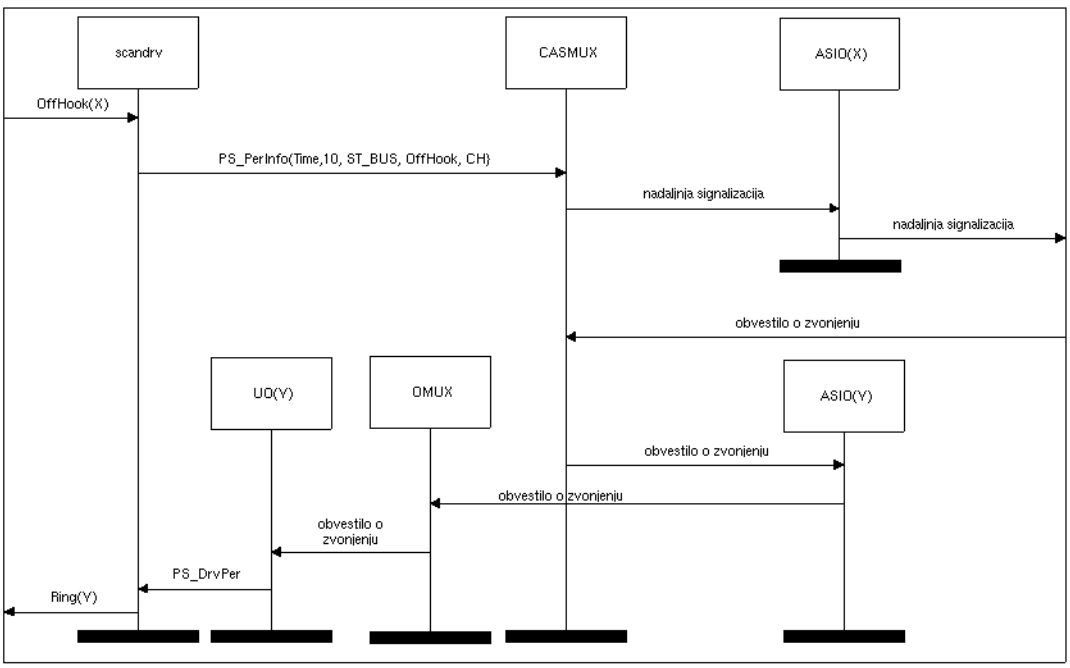
2. To spremembo zapakira v paket PerInfo. S signalom PS_PerInfo se paket pošlje bloku CVA.

3. Signal PS_PerInfo sprejme proces `CASMUX`, ki se nahaja v bloku CVA/SIG/CAS.

4. `CASMUX` obdela prejeto informacijo in ustrezno ukrepa.

Ko je na izvoru izbrana pravilna telefonska številka, se sproži proces obveščanja klicanega uporabnika o prispelem klicu.
Le-to sproži naslednje akcije:

1. `CASMUX` iz višjenivojske programske opreme prejme zahtevo po zvonjenju.
2. `CASMUX` preda zahtevo procesu `ASIO`, ki skrbi za telefonski priključek Y.
3. `ASIO` obvesti o zahtevi proces `OMUX`, ki se nahaja v bloku CDA/SIG/ CAS_OUTPUT.
4. `OMUX` o zahtevi obvesti proces `UO`, le-ta pa zahtevo posreduje `scandrv`-ju.
5. `scandrv` sproži indikacijo poziva na ponorni strani zveze (zvonjenje).

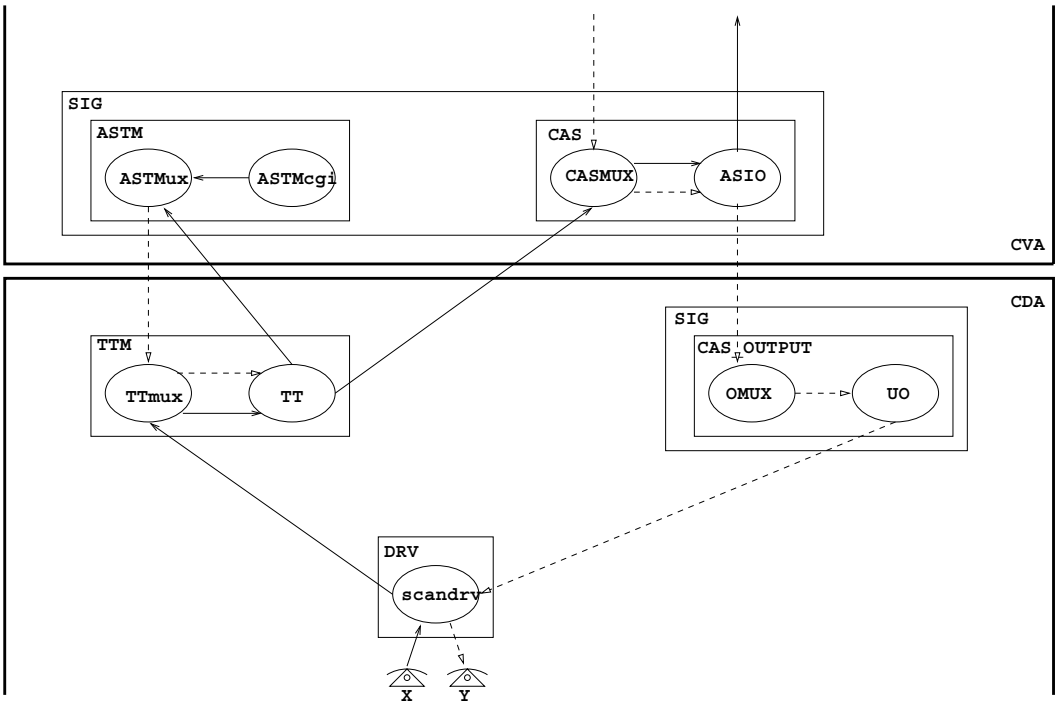


Slika 3.5: Pregled signalov pri vzpostavljanju zveze

Programska oprema znotraj bloka je organizirana tako, da ima vsak blok navadno en nadzorni proces. Nadzorni proces ima v imenu besedico *MUX* in skrbi za tvorjenje ter upravljanje ostalih procesov bloka. Zgornji primer prikazuje takšen način dela. Ko `scandrv` zazna spremembo na priključku številka X, o temu obvesti nadzorni proces bloka `CAS`. `CASMUX` nato preveri, če ima priključek X že tvorjen svoj primerek procesa `ASIO` (Analog Subscriber

I/O). Le-ta se tvori za vsak aktivni priključek posebej. ASIO(X) nato skrbi za telefonski priključek X. O vseh dogodkih obvešča programsko opremo na višjem nivoju. Le-ta poskrbi za pravilno usmerjanje in tarifiranje klica ter nudi dodatne storitve (konferenca, preusmerjanje, ...).

Ko programska oprema najde ponorni telefonski priključek, pošlje ukaz za indikacijo poziva CASMUX-u. Le-ta jo posreduje ASIO-u priključka Y. Če ta še ne obstaja, ga tvori. ASIO(Y) nato posreduje zahtevo procesu OMUX znotraj bloka CDA/SIG/CAS_OUTPUT. Ta preveri, če je za telefonski priključek Y že tvorjena instanca procesa UO. Ko instanca UO(Y) obstaja, ji posreduje zahtevo za indikacijo poziva. UO obvesti scandrv, ta pa sproži akcije na strojni opremi in telefon Y zazvoni.



Slika 3.6: Umestitev PO *Generatorja klicev*

To je opis delovanja programske opreme na najnižjem nivoju signalizacije. Ker smo želeli posege v obstoječo programsko opremo telefonske centrale minimizirati, smo se odločili za programsko shemo, ki jo prikazuje slika 3.6. Dodali smo podbloka CVA/SIG/ASTM in CDA/TTM.

Edini poseg v obstoječo programsko opremo je manjša sprememba scandrv-ja. Ker za delovanje potrebujemo informacije, ki pridejo po povratni poti iz UO-ja, mora scandrv-er

vse signale poslati tudi procesu `TTmux`. Le-ta nato s pomočjo lokalnih zapisov ugotovi, če je omenjeno sporočilo namenjeno kateremu izmed testnih priključkov.

Slika 3.6 prikazuje potek signalov samo za klice, ki jih vzpostavlja *Generator klicev*. Če bi želeli zajeti tudi klice, ki se vzpostavljajo regularno, torej izvirajo iz dejanskih naročniških priključkov, bi morali dodati še povezavo med `scandrv`-jem in `CASMUX`-om (glej sliko 3.4). Takšna organizacija signalizacije nas v ničemer ne omejuje. Če želimo, lahko del priključkov telefonske centrale testiramo s pomočjo *Generatorja klicev*, drugi del priključkov testiramo s pomočjo zunanjih testnih naprav, tretji del telefonskih priključkov pa je dan v uporabo naročnikom. S tem je dosežen eden izmed glavnih ciljev umestitve programske opreme *Generatorja klicev* v obstoječo celoto. Vsa signalizacija v smeri telefonskega priključka, ki izvira iz *Generatorja klicev*, se preko `scandrv`-ja tudi dejansko izvede. Tako klicani telefoni med testiranjem zvonijo. Ker se noben del programske opreme ne zaveda, da signalizacija prihaja iz modula *Generatorja klicev*, se lahko zveza vzpostavi tudi z ročnim dvigom slušalke klicanega telefona.

3.2 Opis SDL strukture Generatorja klicev

Blok ASTM

`ASTM` je okrajšava za *Analog Subscriber Test Module*. Vsebuje dva procesa. `ASTMux` je manager in omogoča dostop do baze podatkov, zbira prometne podatke in skrbi za pravilno delovanje *Generatorja klicev*. `ASTMcgi` vsebuje operatorje, ki omogočajo nadzor *Generatorja klicev* s pomočjo spletnega pregledovalnika. Centrala vsebuje HTTP strežnik z vmesnikom CGI. Tako lahko s programi CGI pošiljamo posameznim procesom poljubne signale.

Blok TTM

Blok `TTM` (Test Telephone Module) prav tako vsebuje dva procesa. `TTmux` je manager in koordinira vse prihajajoče signale. V nasprotju s procesi, ki smo jih srečali do sedaj, se lahko število procesov `TT` med delovanjem sistema spreminja. Vsi pa so povezani s procesom `TTmux` na identičen način. Tudi sami procesi so identični, v svojem delovanju pa neodvisni. `TT`-ji torej predstavljajo posamezne testne telefone.

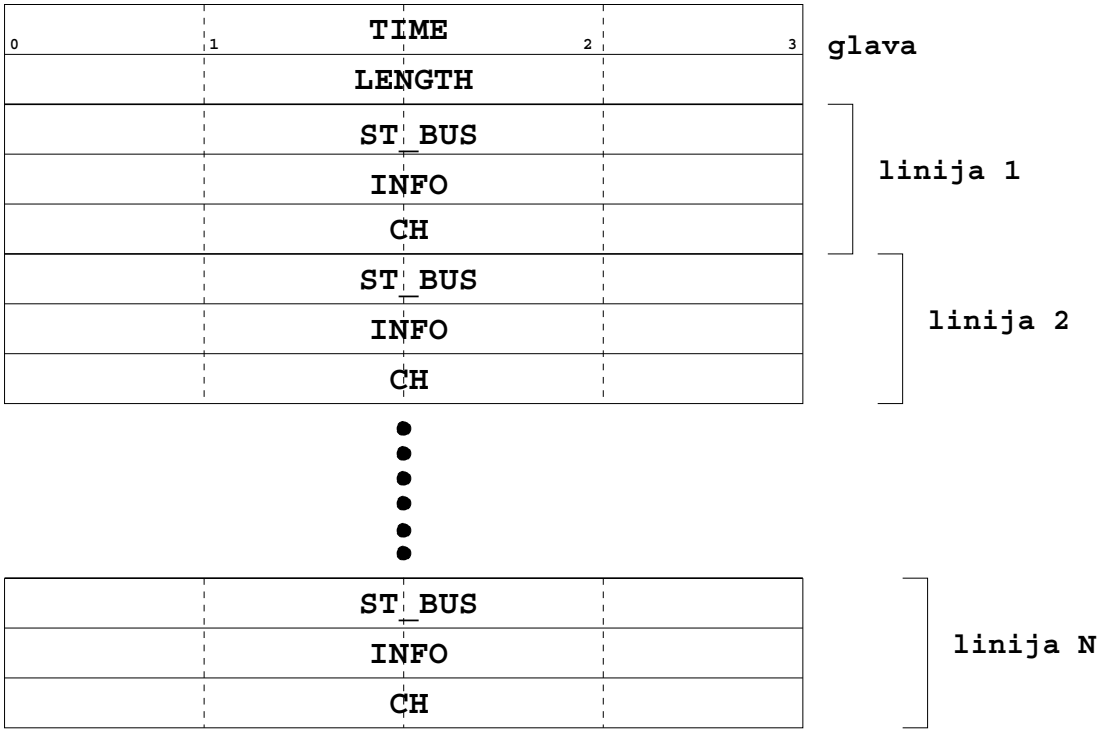
3.3 Pregled signalizacije

Oglejmo si potek signalizacije in signalov v enakem vrstem redu kot poteka vzpostavitev zveze (glej sliki 3.4 in 3.5). Ob dvigu slušalke in izbiranju `scandrv` zazna spremembo v tokokrogu. Uporabljali bomo impulzno izbiranje. V primeru tonskega izbiranja (DTMF) prihajajo tonfrekvenčni signali po govornem kanalu. Na zahtevo procesa `ASSC` proces `MFMUX` in blok `CoCo` rezervirata in povežeta govorni kanal in sprejemnik signalov DTMF. Sprejemnik teče na procesorju DSP. Le-ta s postopki FFT razpoznava dvofrekvenčne signale in sprejete cifre posreduje procesom SDL (preko procesov `MFMUX` in `DTMF` v bloku `ASSC`). `ASSC` je torej stična točka, ki združuje sprejete signale od `ASIO`-ja ali `DTMF`-ja. Uvajanje funkcionalnosti tonskih signalov v *Generator klicev* na MLB bi zahteval dodatne povezave generatorja s procesi MF, v podatkovno strukturo *Generatorja klicev* bi bilo potrebno dodati parameter načina izbiranja (impulzno/dtmf) ter upoštevati strukturo govornih kanalov. Doslej tega posega namenoma nismo predvideli in je vprašanje, če bi bilo to sploh smiselno. Mislimo, da vsaj trenutno ne, saj je nadaljnja signalizacija enaka kot pri impulznem izbiranju. Natančnejši opis procesov, povezanih z DTMF izbiranjem, smo izpustili, saj niso povezani z *Generatorjem klicev*.

Spremembe na vseh naročniških priključkih, ki so se zgodile v zadnjih štirih milisekundah, se zapakirajo v paket PerInfo. Strukturo paketa prikazuje slika 3.7.

Glava paketa vsebuje informacijo o času zajetja podatkov ter dolžino celotnega paketa. Dolžina paketa je odvisna od števila sprememb v zadnjih štirih milisekundah. Vsak zapis je dolg 16 bitov. Za vsako linijo, na kateri je prišlo do spremembe, vsebuje paket tri polja. `ST_BUS` predstavlja številko govorne povezave. Vsaka govorna povezava vsebuje 32 govornih kanalov. Za nadzorno signalizacijo se uporablja ločena povezava enake hitrosti. Vsak govorni kanal ima svojo zaporedno številko (0–31). Zaporedne številke govornih kanalov določa polje `CH`. Polje `INFO` določa spremembo na liniji.
Spremembo povzroči:

- sklenitev tokokroga,
- prekinitev tokokroga,
- pritisk ozemljitvene tipke,
- pritisk kalibrirne tipke.



Slika 3.7: Struktura paketa PerInfo

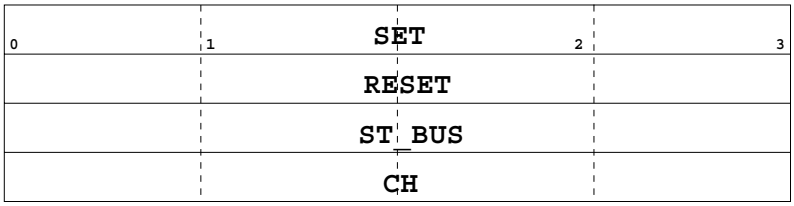
CASMUX te informacije obdela in izvede ustrezne akcije. V primeru *Generatorja klicev* se poleg omenjenih posegov v `scandrv` spremeni tudi komunikacija s CASMUX-om[3]. Namesto da bi se vse spremembe, ki se zgode v zadnjih štirih milisekundah, zapakirale v skupen paket PerInfo, pošiljajo TT-ji CASMUX-u svoje spremembe individualno. Posledica takšnega početja je večja obremenjenost povezave med CDA in CVA. S tem zmanjšamo potencialno uporabnost *Generatorja klicev*, saj ga ne moremo uporabiti preko zmogljivosti omenjene povezave. Za vsako izbrano cifro ”0”na številčnici telefona se namreč pošlje kar dvajset paketov PerInfo. Kljub predvidenim omejitvam smo se odločili za izvedbo v tej obliki[4]. V primeru testnih telefonov so torej paketi PerInfo vedno dolgi deset oktetov, saj beležijo samo spremembe posameznih telefonov. Vsak telefon lahko v določenem trenutku povzroči samo eno spremembo. Zato se vedno prenaša informacija samo za eno linijo (glej sliko 3.7). S tem smo definirali signalizacijo s strani naročniških priključkov in testnih telefonov navzgor.

Testiranje programske opreme telefonske centrale pa mora zajeti tudi povratne informa-

[3] To pomanjkljivost bomo v naslednji verziji *Generatorja klicev* odpravili.

[4] Po zagotovilih sistemske skupine dodatna obremenitev ne bi smela bistveno vplivati na povezavo med CDA in CVA ter na samo delovanje telefonske centrale.

cije. Torej informacije, ki so namenjene ponornemu telefonskemu priključku. Kot kažeta sliki 3.4 in 3.6, se je ta del signalizacije spremenil samo v tem, da `scandrv` vse pakete, ki jih pošljejo `UO`-ji, posreduje tudi `TTmux`-u. Paket, ki ga pošiljajo `UO`-ji, se imenuje `DrvPer`, signal, ki ga nosi, pa PS_DrvPer. Struktura paketa je vidna na sliki 3.8.



Slika 3.8: Struktura paketa DrvPer

Paket vsebuje informacijo o ciljnem telefonskem priključku (polji `ST_BUS` in `CH`) in polji `SET` in `RESET`. Polji `SET` in `RESET` vsebujeta šest informacijskih bitov. Prvo polje postavlja, drugo pa briše funkcije, ki jih določajo informacijski biti. Tabela 3.1 prikazuje strukturo teh dveh polj.

Tabela 3.1: Struktura polj `SET` in `RESET`

BIT	POMEN
0	x
1	x
2	ACT
3	TEST
4	OBR
5	MT
6	POZ
7	BL

Razlaga posameznih ukazov:

- BL : 0 — normalni linijski tok
  1 — znižan linijski tok
- MT : 0 — mirovno stanje
  1 — oddaja tarifnega impulza
- TEST : 0 — preklop na pozitivno vodilo (pozivanje)
  1 — preklop na testno vodilo (test linije)
- Možna stanja na naročniškem linijskem vmesniku:

zap. št.	OBR	ACT	POZ	status
1	0	0	1	pozivanje
2	0	1	0	aktivno stanje
3	1	1	0	aktivno stanje z obr. pol.

Za nas je predvsem zanimiv bit POZ, saj vklaplja in izklaplja zvonjenje. Ko telefonska centrala vklopi zvonjenje telefonskega naročnika, ima oktet `RESET` same ničle, oktet `SET` pa ima naslednjo vsebino: 01000000. Kot smo že omenili, `scandrv` preusmerja vse signale PS_DrvPer, tudi procesu `TTmux`. Le-ta s pomočjo ADT operatorjev analizira sprejeto informacijo. Najprej preveri, če `ST_BUS` in `CH` določata testni priključek. Če temu ni tako, se sprejeti signal ignorira. Če pa je sprejeta informacija namenjena testnemu telefonskemu priključku, se posreduje procesu `TT`, ki ta priključek zastopa. Proces `TTmux` mora torej voditi evidenco o vseh obstoječih `TT` procesih. To mu omogoča preslikavo med parom (`ST_BUS`, `CH`) in PID-om pripadajočega `TT` procesa. Ko proces `TT` sprejme informacijo o zvonjenju, se scenariju primerno odzove. Vse svoje akcije sporoči `CASMUX`-u, tako kot pri procesu izbiranja telefonske številke.

S tem smo opisali osnovni princip komunikacije strukture *Generatorja klicev* z obstoječo programsko opremo telefonske centrale. Opisali smo tudi spremembe v signalizaciji, ki jih *Generator klicev* vnaša.

4 GENERATOR KLICEV

Po analizi zahtev in delovanja programske opreme telefonske centrale ter tehtnem premisleku in posvetovanju z razvijalci smo se lotili razvoja *Generatorja klicev*.

4.1 Zapisi v bazi podatkov

Prvi korak dejanske implementacije *Generatorja klicev* je bil razmislek o bazi podatkov [11, 17, 12, 14]. Določiti je bilo potrebno dve vrsti podatkov:

1. konfiguracijske podatke in
2. prometne podatke.

Konfiguracijski podatki morajo čim natančneje določati delovanje generatorja kot celote, prav tako pa tudi delovanje posameznih `TT`-jev. Prometni podatki pa bodo namenjeni kasnejši analizi izvajanja testnih scenarijev. Tvorjenje zapisov v bazi podatkov nam omogoča opis lastnosti testnih telefonov in scenarijev s pomočjo obstoječega orodja `irtsql`. Ta uporablja kot svojo sintakso množično razširjeni jezik SQL (Structured Query Language). Takšni posegi v bazo so večini razvijalcev že poznani, zato so še posebej primerni.

Podatke smo razdelili v tri ločene tabele:

- skupni podatki generatorja,
- podatki za posamezen `TT`,
- akcije.

Prvi dve tabeli vsebujeta poleg konfiguracijskih podatkov tudi prometne podatke. Polja s prometnimi podatki bodo v tabelah osenčena. Ta polja se med izvajanjem testnega scenarija spreminjajo. Skupni podatki generatorja se nahajajo v tabeli

Tabela 4.1: Vsebina tabele `call_gen_param`

StartTime (Start_Date) (Start_Time)	restart_period Enota: 1 s	EndTime (End_Date) (End_Time)	reset_trafic_data 1 : reset podatkov	write_trafic_data 1 : zapis podatkov	action_pause_unit Enota: 1 ms
num_gen_starts	LastStart (last_start_date) (last_start_time)				

`call_gen_param` (tabela 4.1). Vrednosti, ki so napisane z drobnim tiskom, dodatno opisujejo zapis v bazi.

Natančnejši opis polj:

- *StartTime* vsebuje datum in čas zagona generatorja. Sestavljen je iz dveh polj:

  1. *start_date*: Format zapisa je YYYYMMDD (npr.: 19750124),
  2. *start_time*: Format zapisa je HHMM (npr.: 0750).

- *restart_period* je čas v sekundah. Določa periodo ponovnih zagonov generatorja. Razvijalec se lahko odloči, da se njegov testni scenarij začne izvajati od začetka vsakih N sekund. Privzeta je vrednost 0, ki izklopi ponovne zagone scenarija.

- *EndTime* določa datum in uro zaustavitve *Generatorja klicev*. Sestavljen je iz dveh polj:

  1. *end_date*: Format zapisa je YYYYMMDD (npr.: 19760907),
  2. *end_time*: Format zapisa je HHMM (npr.: 0545).

- *reset_traffic_data* lahko zavzame samo dve vrednosti:

  0 — ne resetiraj podatkov,
  1 — ob ponovnem zagonu resetiraj podatke.

- *write_traffic_data* lahko prav tako zavzame samo dve vrednosti:

  0 — ne zapiši na disk,
  1 — ob ponovnem zagonu zapiši prometne podatke na disk.

- *action_pause_unit* je celo število. Je večkratnik 1 ms in določa osnovno časovno enoto za računanje pavze med posameznimi akcijami v testnem scenariju. Primer[5]: *action_pause_unit*=10 in *in_call_pause*=12 $\longmapsto$ pavza traja 120 ms.

- *num_gen_starts* je števec zagonov testnih scenarijev. To število je ob regularnem zaključku testnega scenarija enako številu restartov + 1.

- *LastStart* označuje datum in čas zadnjega zagona testnega scenarija. Ob neregularnem zaključku testnega scenarija nam ta zapis olajša iskanje napake. Sestavljen je iz dveh polj:

  1. *last_start_date* hrani datum. Format zapisa je YYYYMMDD (npr.:19990915),
  2. *last_start_time* hrani čas zagona. Format zapisa je HHMM (npr.: 1015).

S tem smo opisali podatke, ki so skupni vsem `TT`-jem. Sledi opis tabele s podatki za posamezen `TT` (tabela 4.2).

Tabela 4.2: Vsebina tabele `test_telephone`

tt_id	stbus_id	stbus_CI	tt_delay enota: 100 ms	call_duration_h enota: 100 ms	call_duration_l enota: 100 ms
short_pause_h enota: 100 ms		short_pause_l enota: 100 ms	long_pause_h enota: 1 s	long_pause_l enota: 1 s	
num_out_calls		num_succ_dials	num_in_calls	num_acc_calls	

Za vsak primerek virtualnega telefona beležimo konfiguracijske in prometne podatke. Tabela `test_telephone` ima torej toliko zapisov, kolikor priključkov testiramo.

Natančnejši opis polj:

- *tt_id* predstavlja identifikacijsko številko `TT`-ja. Hkrati je tudi zaporedna številka `TT`-ja v našem scenariju.

- *st_bus_id* je številka `STBUS` priključka, na katerem se nahaja `TT`.

- *stbus_ci*[6] je številka kanala, na katerem se nahaja priključek znotraj `STBUS`-a.

[5] Za razumevanje naslednjega primera je potrebno poznati tudi tabelo `tt_action`

[6] Takšno ime je bilo izbrano zaradi obstoječih definicij znotraj baze podatkov.

- *tt_delay* predstavlja čas, ki ga bo `TT` čakal pred pričetkom izvajanja testnega scenarija. Ta čas bo čakal tudi ob ponovnem zagonu generatorja[7].

  Vpisano celo število predstavlja mnogokratnik 100 ms. Zaloga vrednosti je v bazi podatkov omejena na 0 – INTEGER. To nam omogoči asinhroni pričetek izvajanja akcij posameznih `TT`-jev. Če bi vsi `TT`-ji začeli izbirati v istem trenutku, to ne bi bila korektna predstavitev realnega okolja. Obremenjenost centrale bi v takšnem primeru bila mnogo večja od normalne. To nam odpira dodatne možnosti pri testiranju zmogljivosti centrale ob visokih obremenitvah.

- *call_duration_h* in *call_duration_l* določata časovne okvirje trajanja testne povezave v primeru, ko želimo da se le-ta spreminja naključno. V primeru, ko sta vrednosti v obeh zapisih enaki, je dolžina trajanja zveze točno določena. Določitev trajanja povezave nam omogoča nasilno prekinitev izvajanja scenarija in s tem simulacijo neregularnih dogodkov. Tako lahko zvezo prekinemo že med izbiranjem ali sredi pogovora.

- *short_pause_h* in *short_pause_l* določata meji za izračun kratke pavze. Osnovna enota je 100 ms. Celo število v zapisu predstavlja mnogokratnik te vrednosti. Ti polji sta namenjeni primeru naključne izbire pavze med akcijami. Zaradi večje prilagodljivosti smo se odločili za dva tipa pavz. Kratko in dolgo. Dolga je opisana v naslednji točki. Katera izmed možnosti se bo uporabila, določajo polja za določitev pavze med akcijami (*in_action_pause* in *out_action_pause*) v tabeli `tt_action`. Če je izbrana vrednost 0, se izbere naključna vrednost znotraj mej kratke pavze. Če sta polji z zgornjo in spodnjo mejo enaki, se ta vrednost privzame kot želena pavza.

- *long_pause_h* in *long_pause_l* določata meji za izračun dolge pavze med akcijami. Osnovna enota je 1 s. Celo število v zapisu predstavlja mnogokratnik te vrednosti. Veljajo enaka pravila kot za kratko pavzo. Naključna vrednost v teh mejah se izračuna, ko uporabimo v zapisih *in_action_pause* in *out_action_pause* vrednost 1.

- *num_out_calls* je prometni podatek. V to polje se zapisuje število uspešno zaključenih izvornih klicev. Uspešna zaključitev klica predstavlja regularen zaključek scenarija.

[7] Če smo v tabeli `call_gen_param` v polje *restart_period* zapisali pozitivno od nič različno število.

- *num_succ_dials* je prav tako prometni podatek. Hrani število uspešno končanih izbiranj. To nam omogoča natančnejši pregled poteka klica. Če se klic zaradi kateregakoli vzroka ne zaključi v skladu s scenarijem (npr.: poteče časovnik določen z zapisoma *call_duration_h* in *call_duration_l*), izbira številke pa je bila končana, se ta števec poveča. Števec uspešno zaključenih klicev, ki se hrani v zapisu *num_out_calls*, pa ostane nespremenjen.

- *num_in_calls* hrani informacijo o številu dohodnih klicev posameznega TT-ja.

- *num_acc_calls* hrani informacijo o številu sprejetih klicev. V primeru, ko izvorni TT čaka na sprejem klica le tako dolgo, da uspe telefon na ponorni strani zazvoniti samo dvakrat, ponorni TT pa je nastavljen tako, da klic sprejme po tretjem zvonjenju, ne pride do uspešne vzpostavitve zveze, saj izvorni TT prehitro odneha. V tem primeru se števec *num_in_calls* poveča, *num_acc_calls* pa ne.

Poleg osnovnih konfiguracijskih podatkov ima vsak TT pridruženo še tabelo s scenarijem. Scenarij določa akcije TT-ja. Vsak TT deluje kot realni telefon. Torej lahko deluje kot izvorni in ponorni telefon. Tudi v istem scenariju. S tem smo omogočili pisanje scenarijev, ki pokrivajo vse primere dejanske uporabe. Akcije posameznega TT-ja so shranjene v tabeli tt_action (tabela 4.3). Vsak TT lahko ima poljubno število akcij[8].

Tabela 4.3: Vsebina tabele tt_action

tt_id	tt_action_num	in_call_action	in_call_pause	out_call_action	out_call_pause	called_tt_id

Natančnejši opis polj:

- *tt_id* je številka, ki nedvoumno določa TT. V tabeli tt_action je zapisan scenarij *Generatorja klicev*. Scenarij je sestavljen iz podscenarijev, ki opisujejo obnašanje posameznih TT-jev. Podscenarije ločimo med seboj s pomočjo *tt_id*-ja. Če ima določeni TT 10 akcij, vsebuje polje *tt_id* v desetih zapisih njegovo identifikacijsko številko[9]. Teh deset zapisov določa podscenarij obravnavanega TT-ja. Naslednji zapis v tabeli

[8] Trenutno so programsko omejene na 100.

[9] glej tudi tabelo 4.2

vsebuje podatke za naslednji TT. Tako med sabo ločimo akcije posameznih TT-jev (podscenarije). Dovoljeni nabor vrednosti je med 1 in CV_MaxTTNo=640[10].

- *tt_action_num* vsebuje zaporedno številko akcije TT-ja. V prejšnjem primeru bi se spreminjala od 1 do 10. Ko se zamenja identifikacijska številka TT-ja, torej ob začetku akcij za naslednji TT, polje ponovno vsebuje enico.

- *in_call_action* in *in_call_pause* predstavljata ukazni par za akcije ob dohodnem klicu. TT sledi tem akcijam, ko se nahaja na ponorni strani. Prvo polje je akcija, drugo pa predstavlja pavzo pred izvajanjem naslednje akcije. Pavza je lahko naravno število večje od 1. Prvi dve števili (0 in 1) sta rezervirani za generiranje naključnih pavz[11]. Ničla povzroči generiranje kratke pavze (reda 100 ms), enica pa generira dolgo pavzo (reda 1 s). Poleg tega poznamo še eno izjemo. Pri ukazu *WaitForRing* prevzame polje pavze vrednost števila zvonjenj pred dvigom telefonske slušalke.

- *out_call_action* in *out_call_pause* predstavljata ukazni par akcij za primer, ko je TT izvorni telefon.

- *called_tt_id* je identifikacijska številka (*tt_id*) klicanega telefona. Trenutno se to polje še ne uporablja. Namenjeno je nadaljnjemu razvoju. Načrtuje se namreč tudi slišno preverjanje pravilnosti vzpostavljene zveze.

Polja za akcije in pavze so lahko prazna oz. NULL, saj je s privzetimi vrednostmi poskrbljeno za pravilno delovanje *Generatorja klicev*. Konkreten primer bo predstavljen v nadaljevanju.

S tem smo opisali tabele, ki smo jih tvorili v bazi podatkov. Vrednosti se lahko vnašajo s pomočjo SQL jezika s katerekoli delovne postaje v lokalnem ethernet omrežju. Tako od razvijalca ne zahtevamo, da se nahaja v maketnem centru. Dejansko se lahko razvijalec nahaja na poljubni lokaciji v svetu. Edini pogoj je dostop do lokalnega mrežja, na katerega je priključena testirana telefonska centrala. Testiranja se torej lahko izvajajo tudi na oddaljenih, že inštaliranih telefonskih centralah. Tudi ob normalnem testiranju programske opreme, ki

[10] Predpona CV_ pove, da je to konstantna vrednost. Vrednost dobimo, če pomnožimo število vtičnih mest MLB modula s številom priključkov na modul: $20 * 32 = 640$

[11] Glej razlago polj tabele test_telephone.

se navadno izvaja v maketnem centru, nudi oddaljen pristop precej prednosti. Med ostalim je maketni center zaradi množice central hrupen, zato je razvojni oddelek primernejši za izvajanje analize funkcionalnosti programske opreme. Do sedaj to ni bilo mogoče.

Eden izmed ciljev v fazi načrtovanja je bil oddaljen dostop do centrale med testiranjem. Vnos podatkov torej ustreza tej zahtevi.

4.2 Pregled rezultatov testiranja

Zelo pomemben del testiranja je seveda pregled rezultatov. Tudi ta del smo želeli narediti čimbolj prijazen. Zaradi razmaha svetovnega spleta smo se odločili za vmesnik uporabiti spletni brskalnik. Centrala nam to omogoča, saj ima spletni strežnik, ki podpira CGI vmesnik. S tem tudi omogoča pregled rezultatov iz kateregakoli računalnika, ki ima dostop do ethernet omrežja, na katerem se nahaja testirana centrala. Če smo želeli, da so podatki dostopni CGI programom, jih je bilo potrebno preslikati v skupni pomnilniški prostor. Prometni podatki se tako beležijo na dveh mestih. V pomnilniških tabelah in v bazi podatkov. Pomnilniške tabele so prikazane v tabeli 4.4. Da razbremenimo vodilo za dostop do diska in CVA procesor, smo se odločili, da se bodo podatki v realnem času beležili samo v pomnilniških tabelah. V bazo podatkov pa se bodo zapisali v naprej izbranih časovnih intervalih ter ob zaključku scenarija.

Tabela 4.4: Pomnilniške tabele s prometnimi podatki

Gen_Data:

StartTime	RestartPeriod	EndTime	ResetTrafficData	WriteTrafficData	ActionPauseUnit

GenNoOutCalls	GenNoSuccDial	GenNoInCalls	GenNoAccCalls	NumGenStarts	LastStart

TT_Data:

TT_id				
NoOutCalls	NoSuccDial	NoInCalls	NoAccCalls	Status

Trenutno se v bazo podatkov preslikujejo samo skupni prometni podatki. V pomnilniku pa se beležijo prometni podatki za vsak `TT` posebej. Tako lahko spremljamo izvajanje scenarijev v realnem času s pomočjo spletnega brskalnika.

Natančnejši opis polj tabele Gen_Data:

- *StartTime* beleži čas zagona scenarija.
- *RestartPeriod* hrani podatke o periodi ponovnega starta scenarija.
- *EndTime* hrani podatke o času zaključka izvajanja scenarija.
- *ResetTrafficData* določa, ali se bodo prometni podatki ob ponovnem startu scenarija resetirali ali ne (0 — ne resetira, 1 — resetira).
- *WriteTrafficData* določa, ali se bodo prometni podatki ob ponovnem startu scenarija zapisali na disk ali ne (0 — ne zapiši na disk, 1 — zapiši na disk).
- *ActionPauseUnit* hrani informacijo o osnovni enoti pavze med akcijami[12].
- *GenNoOutCalls* hrani vsoto vseh začetih odhodnih klicev v scenariju. Tu se beležijo uspešne in neuspešne zveze.
- *GenNoSuccDial* hrani vsoto vseh uspešnih izbiranj.
- *GenNoInCalls* hrani vsoto vseh ponornih klicev. Na tem mestu se beležijo vsi ponorni klici. Tudi tisti, ki niso sprejeti na ponorni strani.
- *GenNoAccCalls* hrani vsoto vseh sprejetih klicev na ponorni strani.
- *NumGenStarts* beleži število ponovnih startov scenarija.
- *LastStart* beleži čas zadnjega starta scenarija.

Natančnejši opis polj tabele TT_Data:

- *TT_id* je ključ tabele. Določa, kateremu `TT`-ju pripadajo prometni podatki.
- *NoOutCalls* hrani vsoto vseh začetih odhodnih klicev `TT`-ja. Tu se beležijo uspešne in neuspešne zveze.
- *NoSuccDial* hrani vsoto vseh uspešnih izbiranj `TT`-ja.
- *NoInCalls* hrani vsoto vseh dohodnih klicev `TT`-ja. Tu se beležijo vsi dohodni klici. Tudi tisti, ki niso sprejeti.

[12] glej tudi tabelo `call_gen_param` (tabela 4.1)

- *NoAccCalls* hrani vsoto vseh klicev, ki jih `TT` sprejme.
- *Status* beleži trenutni status telefona. Telefon je lahko v naslednjih stanjih:
  - brezdelen — idle (0),
  - kličem — dialing (2),
  - zvonim — ringing (3),
  - govorim — talking (4).

  Številčna vrednost v oklepaju predstavlja zapis v tabeli.

Vse zapise v bazi podatkov in v pomnilniški tabeli smo prilagodili tako, da so cela števila. S tem smo poenostavili izdelavo dodatnih programov za analizo podatkov in zmanjšali količino prostora, ki ga tabele zavzamejo na disku in v pomnilniku.

Za natančnejšo analizo vseh pridobljenih podatkov se lahko napišejo dodatni programi, ki bodo zajemali podatke iz baze podatkov. V nadaljnjem razvoju se bo omogočil tudi zapis na disk. S tem bi lahko shranjevali podatke za vsako izvajanje scenarija posebej. To bi nam omogočilo primerjavo rezultatov in detekcijo nedeterminističnih napak.

4.3 Izbira akcij

Tabela 4.5 prikazuje akcije, ki jih lahko izvaja posamezen `TT`. Prikazuje tudi njihov številčni ekvivalent, ki se uporablja pri zapisu v bazo podatkov.

Uporaba znakov * in # omogoča izvajanje dopolnilnih storitev. Dopolnilne storitve so zapisane v [6]. Tudi tu *Generator klicev* ne postavlja nobenih omejitev.

4.4 Opis delovanja Generatorja klicev s SDL

Za opis *Generatorja klicev* smo uporabili orodje GEODEDIT V2.2.4 francoskega podjetja Verilog. Ta verzija podpira standard SDL-88.

Skladno z odločitvami, ki smo jih sprejeli v fazi analize problema, je bilo najprej potrebno določiti SDL strukturo obeh blokov (glej sliki 3.6 in 4.1). Blok `ASTM` se bo nahajal na strani CVA, blok `TTM` pa na strani CDA. Blok `ASTM` smo razdelili na procesa `ASTMux` in `ASTMcgi`, blok `TTM` pa na `TTmux` in `TT`.

Tabela 4.5: Akcije, ki jih lahko izvaja testni telefon

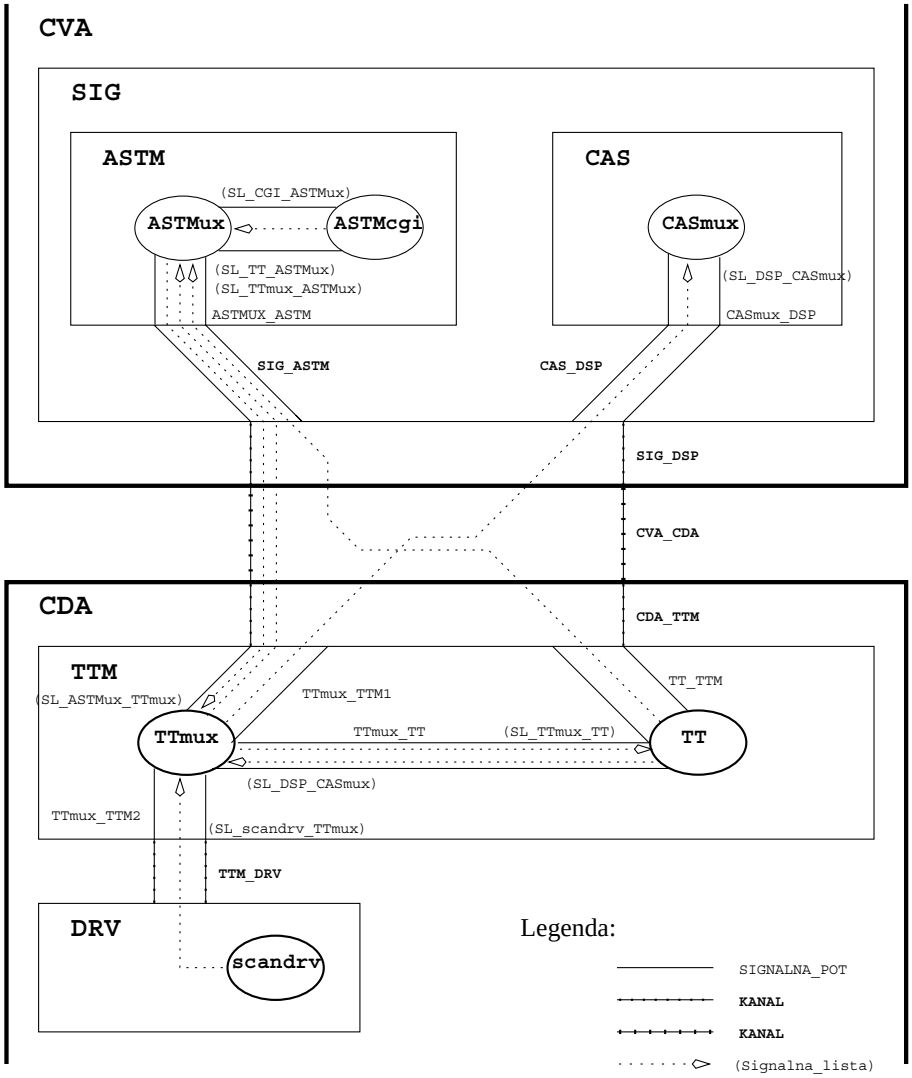
Število	Pomen
1	Izbira tipke 1
2	Izbira tipke 2
3	Izbira tipke 3
4	Izbira tipke 4
5	Izbira tipke 5
6	Izbira tipke 6
7	Izbira tipke 7
8	Izbira tipke 8
9	Izbira tipke 9
10	Izbira tipke 0
11	Izbira tipke *
12	Izbira tipke #
33	RR — Kalibrirna tipka.
20	OffHook — Dvig slušalke.
30	OnHook — Polog slušalke.
40	GNDOn
50	GNDOff
66	EndOfSelection — Konec izbire telefonske številke.
99	EndOfScenario — Konec scenarija.
80	NOP (No operation) — Nič se ne izvede. Uporabno za vnašanje dodatnih pavz v scenarij.
100	WaitForRing — Čakaj na klic. Pavza tega ukaza pove, koliko zvonjenj naj `TT` čaka pred sprejetjem klica z dvigom slušalke.

Definicija signalov, signalnih poti in kanalov

Procesi komunicirajo med sabo s pomočjo signalov[13]. Signale je dobro združiti v signalne liste, saj se na ta način olajša dodajanje in brisanje signalov. Pri opisu signalnih poti in

[13] Obstajajo tudi druge možnosti, vendar jih nismo uporabljali.

kanalov je namreč potrebno navesti vse signale, ki jih le-ta prenaša. Slika 4.1 prikazuje kanale, signalne poti in signalne liste, ki jih uporablja *Generator klicev*.



Slika 4.1: Komunikacija med procesi

Sledi kratek opis signalov. Njihova uporaba bo opisana v nadaljevanju.

- `PS_Line` nosi informacijo o posameznem testnem priključku,
- `PS_LineAction` vsebuje en zapis tabele `tt_action`,
- `PS_CDAconfirm` je signal, ki ga `TTmux` uporablja za potrditev sprejema podatkov,
- `PS_Stop` zaustavi *Generator klicev*,
- `PS_Start` zažene *Generator klicev*,

- `PS_Restart` povzroči ponovni zagon scenarija,
- `PS_Status` je namenjen obveščanju `ASTMux`-a o stanju testnih telefonov,
- `PS_PerInfo` nosi spremembe na testnem priključku,
- `PS_Drv_Per` nosi povratne informacije,
- `PS_RingTT` oznanja zvonjenje.

Naštete signale smo združili v naslednje signalne liste:

- SIGNALLIST SL_ASTMux_TTmux=
  `PS_Line`,
  `PS_LineAction`,
  `PS_Restart`,
  `PS_Stop`,
  `PS_Start`;
- SIGNALLIST SL_TTmux_ASTMux=
  `PS_CDAconfirm`;
- SIGNALLIST SL_TT_ASTMux=
  `PS_Status`;
- SIGNALLIST SL_DSP_CASmux=
  `PS_PerInfo`;
- SIGNALLIST SL_scandrv_TTmux=
  `PS_Drv_Per`;

Imena signalnih list so tvorjena iz imen procesov, ki jih povezujejo. S tem poskrbimo za pregledost. Pri definiciji signalnih poti in kanalov za potrebe *Generatorja klicev* sedaj zadostuje samo ena signalna lista za posamezno smer komunikacije. V našem primeru smo z uporabo signalnih list privarčevali samo pri komunikaciji procesov `ASTMux` in `TTmux`. Zagotovili pa smo si nemoten nadaljnji razvoj, saj v primeru dodajanja ali brisanja signalov ne bo potrebno spreminjati definicije signalnih poti in kanalov.

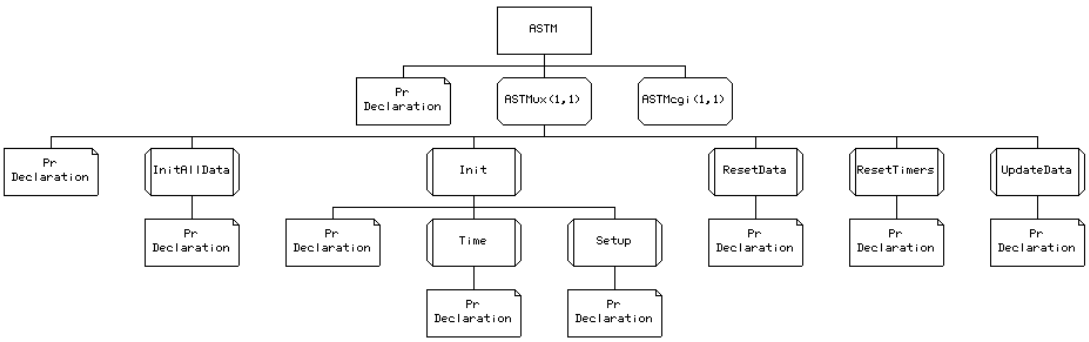
Ker so vsi podatki, ki so potrebni za zagon *Generatorja klicev*, shranjeni v bazi podatkov, s to pa lahko komunicira samo programska oprema na strani CVA, je smiselno nadaljevati z opisom bloka `ASTM`.

Opis bloka ASTM

Blok `ASTM` je sestavljen iz dveh procesov:

1. `ASTMux(1,1)` je upravljalni proces (manager).
2. `ASTMcgi(1,1)` je proces, ki bo skrbel za CGI komunikacijo z *Generatorjem klicev*[14].

Prva enica v oklepaju pove začetno število primerkov procesa ob startu centrale, druga pa predstavlja maksimalno število primerkov procesa. V primeru, ko drugo število ni navedeno, omejitev o številu procesov med delovanjem sistema ni. To v praksi ne drži, saj se skozi nastavitve operacijskega sistema pSOS+ te vrednosti za procese v splošnem omejijo. S popravkom nastavitev pa se lahko doseže poljubno, končno veliko, število procesov. V mislih moramo imeti delovanje v realnem času.
Celotna razgradnja bloka `ASTM` je vidna na sliki 4.2.



Slika 4.2: Struktura bloka ASTM

Poleg procesov je na drugem hierarhičnem nivoju tudi pravokotnik z deklaracijami. Tu se definirajo signali, seznami signalov, novi tipi in konstante, za katere želimo, da so vidni znotraj celotnega bloka. Trenutno vsebuje samo deklaracijo seznama signalov, ki se bodo

[14] Trenutno še ni realiziran.

prenašali med procesoma v tem bloku. Ker blok `ASTMcgi` še ni realiziran, vsebuje samo t.i. ”dummy” proces, ki ne počne ničesar.

Podobno kot na nivoju bloka ima svoje deklaracije tudi posamezen proces. Tu se definirajo spremenljivke procesa, konstante, časovniki, novi tipi in ADT operatorji. Vse, kar je definirano znotraj procesa, je njemu lokalno. Tako lahko v različnih procesih definiramo spremenljivke z enakimi imeni.

Proces lahko zaradi preglednosti in pametnejše razgradnje razdelimo na procedure. Proces `ASTMux` ima pet procedur. Ob zagonu procesa se najprej izvede prehod iz začetnega (start) stanja v prvo stabilno stanje. Ob tem prehodu se lahko izvedejo različne akcije (inicializacije spremenljivk, pošljejo izbrani signali, . . .).

Proces `ASTMux` najprej pokliče proceduro `InitAllData`. Kot namiguje že samo ime, ta procedura izvrši inicializacijo vseh spremenljivk procesa. Nato s pomočjo ADT operatorja `CGImInitData` inicializiramo obe pomnilniški tabeli na nič. Kako se definirajo ADT operatorji, bomo opisali kasneje. Za tem se izvede procedura `Init`[15]. Ta preveri, če je to prvi ali ponovni zagon scenarija, in temu primerno ukrepa. Če je to prvi zagon, je potrebno prebrati podatke iz baze podatkov in jih prenesti programski opremi, ki se nahaja na strani CDA. Da minimiziramo obremenjenost povezave med stranema CVA in CDA, smo uvedli preprost protokol prenosa z rokovanjem. Podatki so razdeljeni v skupine po `TT`-jih. Pred prenosom se preštejejo vrstice vhodnih in izhodnih akcij posameznega `TT`-ja. S tem optimiziramo čas prenosa podatkov, saj prenesemo samo toliko vrstic tabele, kolikor jih je dejansko uporabljenih. Sicer bi bilo potrebno za vsak `TT` prenašati naksimalno število vrstic[16]. Po prenosu vseh podatkov se preverijo osnovni podatki generatorja. Podatkom primerno se nastavijo časovniki za:

- osveževanje baze podatkov,
- začetek scenarija,
- zaključek scenarija.

V primeru, da je prišlo med prenosom do napake, se proces `ASTMux` postavi v stanje napak. V stanju napak ignorira vse sprejete signale in napako javlja z izpisom na konzolo.

[15] glej prilogo B, Pogled: 8 / Stran: 12 do 17

[16] Trenutno je to programsko omejeno na 100.

S tem je proces ASTMux opravil prvi del svojega dela. Uporabili smo ga za branje zapisov iz baze podatkov. Prebrane zapise je bilo potrebno zapakirati v dogovorjene pakete ter jih poslati procesu TTmux. Sedaj proces ASTMux čaka na signale iz bloka CDA. Posamezni TT-ji ga namreč obveščajo o svojih akcijah. ASTMux jih beleži v pomnilniških tabelah. Ob poteku časovnika za osvežitev baze podatkov skupne prometne podatke zapiše na disk. Kateri podatki se zapišejo v bazo podatkov in kateri so dostopni samo v pomnilniku, je razvidno iz primerjave tabel 4.1, 4.2 in 4.4. TT-ji obveščajo ASTMux o spremembah s pomočjo signala PS_Status. Le-ta vsebuje samo dva podatka, identifikacijsko številko testnega priključka in karakteristično število. Procedura UpdateData s pomočjo prejetega karakterističnega števila zazna spremembo stanja posameznega TT-ja in osveži zapise v pomnilniški tabeli. Zapisi se osvežujejo s pomočjo ADT operatorjev CGImGenData in CGImTTData. Prvi skrbi za zapis skupnih prometnih podatkov, drugi pa za zapis prometnih podatkov posameznega TT-ja. Skupni prometni podatki predstavljajo vsoto prometnih podatkov posameznih TT-jev.

Proces ASTMux nadzira tudi izvajanje scenarija. Ob poteku časovnika za ponovni zagon scenarija izvede proceduro Init in o tem obvesti procese znotraj bloka CDA. Kljub temu, da nadzor s pomočjo spletnega pregledovalnika še ni realiziran, smo pri razvoju procesa ASTMux to že imeli v mislih. Zato so predvidene akcije v ASTMux-u že pripravljene. Drugi signali, ki se začnejo s predpono PS_, bodo prihajali s strani procesa ASTMcgi. Realizirati je tako potrebno samo še pošiljanje signalov SDL procesom s pomočjo spletnega pregledovalnika. Zaradi dodatne zahtevnosti se tega v prvi verziji *Generatorja klicev* nismo lotili. Ti signali so:

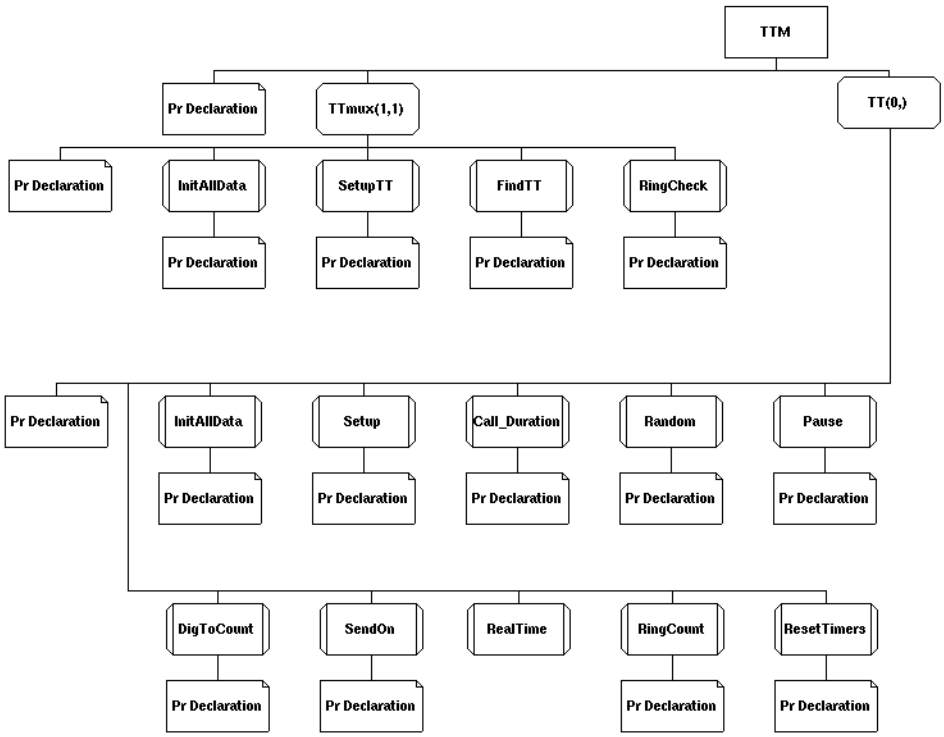
- PS_Start poskrbi za zagon generatorja v trenutku, ko si to zaželi razvijalec.
- PS_Restart poskrbi za ponovni zagon scenarija.
- PS_Stop ustavi generator.

Proces ASTMux vsebuje še dve preprosti proceduri, ki skrbita za resetiranje prometnih podatkov in časovnikov. S tem smo opisali osnovne naloge procesa ASTMux in na grobo razložili njegovo delovanje. Kot smo opazili, je njegova glavna naloga komunikacija z bazo podatkov in spletnim brskalnikom ter zbiranje in urejanje prometnih podatkov. Naloge, ki jih opravlja, zahtevajo prisotnost na strani CVA. V prihodnjem razvoju se bo realiziral tudi zapis

v datoteko na disku, ki ne bo odvisen od osveževanja baze podatkov. To bi nam omogočilo N-kratno izvajanje scenarija in hranjenje prometnih podatkov za vsakega posebej. S tem bi lahko preverjali ponovljivost rezultatov in s tem zanesljivost centrale brez posredovanja razvijalca.

Opis bloka TTM

Srce *Generatorja klicev* se nahaja v bloku `TTM`. Sestavljen je iz dveh procesov. Proces `TTmux` je manager. Kot kaže slika 4.3, vedno obstaja samo en primerek tega procesa.



Slika 4.3: Struktura bloka TTM

Ob zagonu generatorja prejema iz bloka CVA informacije o potrebnih testnih telefonih. Temu primerno tvori za vsak testni telefon svoj primerek procesa `TT`. Še pred pričetkom komunikacije s stranjo CVA se požene procedura `InitAllData`, ki nastavi vse spremenljivke na privzete vrednosti. Poleg tega poskrbi tudi za lokalno tabelo, v katero se bodo zapisovale identifikacijske številke priključkov (`TT_id` in identifikacijske številke pripadajočih procesov `TT` (PID-i)). Zatem se proces postavi v prvo stabilno stanje `J010_Ready` in čaka na nadaljnje dogodke. Ob zagonu generatorja pride s strani CVA že omenjeni si-

gnal PS_Line. Le-ta nosi informacije o prvem testnem telefonu (TT-ju). Tu je prisotna tudi informacija o številu akcij, ki jih bo obravnavani TT izvajal. To potrebujemo za optimizacijo prenosa podatkov. Ko se signal obdela, se proces postavi v naslednje stabilno stanje. V stanju J020_Actions sedaj pričakuje signale, ki bodo vsebovali informacijo o pripadajočih akcijah. Vsaka vrstica iz tabele tt_action (tabela 4.3) se prenese s svojim signalom PS_LineAction. S tem razbremenimo povezavo med stranema CVA in CDA, ki je med procesom inicializacije centrale še posebno obremenjena. Vsak sprejeti signal potrdimo s signalom PS_CDAConfirm. S tem smo uvedli najpreprostejše potrjevanje pravilnosti sprejema. Ko se vse akcije sprejmejo brez težav, kličemo proceduro SetupTT. Ta vsak novi TT zapiše v lokalno tabelo, ki je indeksirana po identifikacijski številki TT-ja (TT_id). Tabela hrani za vsak TT podatke o:

- STbus-u,
- CH-ju in
- PID-u.

Ti zapisi nam bodo pri posredovanju signalov pomagali poiskati pravi proces. S strani scandrv-ja pridejo signali naslovljeni na kombinacijo (STbus, CH). S pomočjo tabele lahko tako preverimo, če je prispeli signal namenjen kakšnemu od testnih telefonov. Torej izvršimo neke vrste filtriranje signalov. Če pravega testnega telefona (TT-ja) ne najdemo, signal ignoriramo, sicer pa posredujemo signal na pravi naslov (PID). Procesom, ki med delovanjem sistema obstajajo v več primerkih, lahko pošljemo signal samo, če poznamo njihovo identifikacijsko številko. PID proces enoumno določa. S tem smo se že navezali na naslednjo nalogo procesa TTmux. Prva naloga je torej pravilno tvorjenje procesov TT. Vsak proces TT predstavlja en testni telefon. Lokalno ima shranjene vse potrebne informacije za predvideno obnašanje. TTmux teh informacij ne hrani. Hrani samo tabelo s preslikavami med parom (STbus, CH) in PID-om procesa TT, ki predstavlja telefonski priključek na tem naslovu. Druga naloga procesa TTmux pa je sprejemanje signalov s strani scandrv-ja. Le-ti prihajajo s signalom PS_Drv_Per. Ta signal nosi informacijo o spremembah na posameznem priključku, ki jih narekuje telefonska centrala. Signali prihajajo iz procesa UO (glej sliko 3.6). Paket, ki ga nosi signal, je viden na sliki 3.8. Tudi o informaciji, ki jo nosi, smo že govorili. Zato tega na tem mestu ne bomo ponavljali. Edina trenutno aktualna funkcija

tega signala je detekcija zvonjenja. Pri tem nam pomaga procedura `RingCheck`. Ta najprej s pomočjo ADT operatorja `upkpert` pretvori sprejeto informacijo v celo število. Procedura to število primerja z vnaprej definiranimi vrednostmi. Definicije so vidne v Prilogi B (Pogled:25 / Stran: 42).

Če je zaznan signal zvonjenja, se preveri, ali je namenjen kateremu izmed testnih telefonov. Brskanje po lokalni tabeli izvrši procedura `FindTT`. Detekcija zvonjenja je seveda nujno potrebna za pravilno izvajanje scenarija. V primeru, da je prišla indikacija zvonjenja za testni telefon, se primerek procesa obvesti s signalom `PS_RingTT`.

`TTmux` sprejema s strani CVA tudi ukaze za začetek, ponovni zagon in zaključek scenarija. O tem nato s pomočjo lokalne tabele o obstoječih `TT`-jih obvesti tudi njih. S tem smo opisali delovanje upravljalnega procesa bloka `TTM`.

Preostane nam še opis procesa `TT`, ki dejansko predstavlja testni telefon in izvaja akcije, ki jih narekuje scenarij. S slike 4.3 je razvidno, da je ta proces najobširnejši. Podobno kot pri ostalih se tudi tu na začetku izvede procedura `InitAllData`. Ta poskrbi za inicializacijo vseh uporabljenih spremenljivk. Takoj za tem se s pomočjo procedure `Setup` izvede analiza prispelih akcij. Prešteje se število izvornih in ponornih akcij. Izvorne akcije se uporabljajo pri klicanju, ponorne pa pri sprejemu klica. S tem je prehod v prvo stabilno stanje zaključen. `TT` je sedaj pripravljen na izvajanje scenarija. Ko ga `TTmux` obvesti, naj začne z izvajanjem scenarija, se najprej izvrši procedura `Call_Duration`. Ta s pomočjo časovnika `TM_Call_Duration` določi dovoljeno dolžino klica. Dolžina klica je lahko točno določena s strani razvijalca ali pa naključno izbrana v mejah, ki jih določi razvijalec. Za natančnejši opis možnosti si oglejte opis tabele `test_telephone` (tabela 4.2). Nato se zažene časovnik `TM_Start`. Dolžina čakanja pred pričetkom izvajanja akcij je odvisna od zapisa *tt_delay* v tabeli `test_telephone`. Po poteku časovnika se izvajanje scenarija dejansko prične. Privzet je izvorni scenarij. V SDL opisu se to določa s pomočjo spremenljivke `V_Scenario`, ki lahko zavzame tri vrednosti. Če se nahajamo v izvornem scenariju, ima ta spremenljivka prirejeno konstanto z imenom `CV_Outgoing`, če se nahajamo v ponornem scenariju, pa privzame vrednost konstante `CV_Incoming`. V primeru, ko `TT` ni v nobenem od scenarijev (ko čaka na klic), vsebuje vrednost konstante `CV_None`. Prebere se prva akcija in prva pavza. Pri pavzi imamo tri možnosti. Pri pisanju scenarija se je lahko razvijalec odločil za točno določeno dolžino pavze ali pa za eno izmed dveh naključno izbra-

nih vrednosti. Pri pravilni določitvi pavze nam pomaga procedura `Pause`. Če je vrednost pavze enaka nič, nam procedura generira kratko pavzo (reda 100 ms). Če je vrednost enaka ena, nam generira dolgo pavzo (reda 1 s). Drugače se privzame vrednost, ki si jo je izbral razvijalec. Tudi na meje kratke in dolge pavze lahko vpliva razvijalec (glej razlago tabele 4.3).

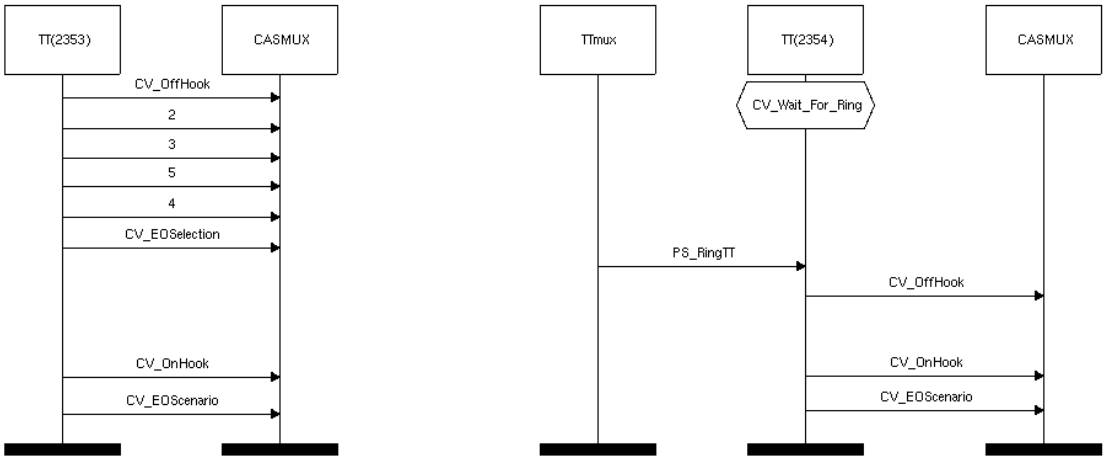
Sedaj imamo določeno akcijo in pavzo prvega ukaza. Naslednji korak je izvedba akcije. Vse možne akcije so že bile naštete v tabeli 4.5. Tem so bile prirejene naslednje konstante:

- `CV_WaitForRing`,
- `CV_OffHook`,
- `CV_OnHook`,
- `CV_NOP`,
- `CV_RR`,
- `CV_EOScenario`,
- `CV_EOSelection`,
- `CV_GNDON`,
- `CV_GNDOFF`.

Če je prva akcija enaka `CV_EOScenario`, vemo, da `TT` ne bo izvajal nobenih izvornih akcij. Zato samo še resetiramo časovnik `TM_Call_Duration` in se postavimo v osnovno stabilno stanje (`J010_Ready`). `TT` v tem primeru ne izvaja nobenih akcij, dokler ne dobi indikacije o zvonjenju. Ko pride do zvonjenja, pa se začne izvajati ponorni scenarij.

Za nazorni opis delovanja procesa `TT` bomo opisali scenarij, ki ga prikazuje slika 4.4.

To je najenostavnejši scenarij, ki si ga lahko zamislimo. `TT`(2353) kliče `TT`(2354). Število v oklepaju predstavlja klicno številko testnega telefona. Zaradi boljše preglednosti si bomo najprej ogledali delovanje `TT`-ja, ki izvaja izvorni scenarij. Oba `TT`-ja delujeta neodvisno drug od drugega. Oba sledita akcijam, ki so zapisane v scenariju posameznega `TT`-ja. V primeru, da bi `TT`(2354) bil v trenutku, ko ga kliče `TT`(2353), že v pogovoru, `TT`(2353)



Slika 4.4: Testni scenarij

tega ne bi zaznal, saj nimamo neposredne povezave med navideznima telefonoma[17]. Takšni primeri so kasneje opazni v statistiki klicev.

Kot je razvidno iz grafične predstavitve scenarija, je prva akcija izvornega `TT`-ja CV_OffHook. Najprej je torej potrebno dvigniti navidezno telefonsko slušalko. Za to poskrbi procedura `SendOff`. Procedura s pomočjo ADT operatorja `pkinfo` zapakira informacijo o trenutnem času, dolžini paketa, spremembi na liniji ter naslovu linije (STbus, CH) v paket PerInfo. Tega nato s signalom `PS_PerInfo` pošlje procesu `CASMUX`. Pri pulznem izbiranju se večina dogodkov opisuje s sklepanjem in prekinjanjem tokokroga. Zrcalna slika procedure `SendOff` je procedura `SendOn`. Ta pošlje `CASMUX`-u obvestilo o prekinitvi tokokroga oz. pologu slušalke. Vsako akcijo spremlja tudi pavza[18]. Tako se tudi v tem primeru pred izvajanjem naslednje akcije počaka želeni čas. Za to poskrbi časovnik `TM_Interdig`. Ta časovnik se vedno sproži na koncu akcije. Sedaj je izvorni `TT` pripravljen na izbiranje želene telefonske številke. Vsa števila, ki so manjša od 13, `TT` prepozna kot izbiro tipke na navideznem telefonu. Števila od ena do deset predstavljajo numerični del tipkovnice (tipke 1, 2, 3, 4, 5, 6, 7, 8, 9, 0), enajst in dvanajst pa funkcijski tipki * in #[19]. Po dvigu slušalke se `TT` nahaja v osnovnem stanju pripravljenosti. Po poteku časovnika `TM_InterDig` se izračuna realni čas. Za tem takoj poteče časovnik `TM_Start`, ki sproži izvajanje naslednje akcije. V našem primeru je to prva cifra izbranega `TT`-ja. S pomočjo procedure `DigToCount` se

[17] To funkcionalnost bo prinesel nadaljnji razvoj *Generatorja klicev*.

[18] glej tabelo 4.3.

[19] glej tabelo 4.5.

nastavi števec V_Counter, ki nadzira število prekinitev tokokroga. Za tem program skoči v stanje J020_Pulse. Tu izmenično izvaja prehoda TM_Off in TM_On. Za vsak prehod zmanjšamo V_Counter. Ko pride števec do nič, vemo, da smo z izbiro trenutne številke zaključili. Za zaključek trenutne akcije zaženemo še časovnik TM_Interdig. Recimo, da želimo med izbiranji številk imeti eno sekundo pavze. To dosežemo s pravilnimi vrednostmi v poljih out_action_pause in action_pause_unit. Do vrednosti pavze pridemo z naslednjim izračunom:
out_action_pause * action_pause_unit = 100 * 10 ms = 1 s.
Na enak način se izvedejo izbire preostalih številk. Zato, da omogočimo štetje uspešno izbranih klicnih števil, smo uvedli posebno akcijo CV_EOSelection. Pavza, ki sledi potrditvi uspešne izbire telefonske številke, predstavlja dolžino pogovora. Po poteku časovnika se testnemu telefonu pošlje zahteva za polog slušalke. Za konec se uspešnost scenarija potrdi z ukazom CV_EOScenario. Tudi ta ukaz je namenjen za zapis prometnih podatkov. S tem je izvajanje scenarija zaključeno. Izvorni scenarij se začne ponovno izvajati po poteku časa, ki je pridružen zadnji akciji.

Programska oprema na višjem nivoju sedaj poišče klicani telefonski priključek in mu po standardni poti pošlje zahtevo po zvonjenju. Proces scandrv zahtevo posreduje TTmux-u. Ta ugotovi, da je signal namenjen testnemu telefonu. TTmux poišče PID procesa in mu posreduje signal. TT, ki predstavlja ponorni telefonski priključek, tako sprejme signal PS_RingTT. Ker je to prvi signal zvonjenja, TT še ni v ponornem scenariju. V_Scenario je enak konstanti CV_None. TT stopi v ponorni scenarij in prične s štetjem zvonjenj. Ob začetku štetja zvonjenj se zažene tudi časovnik TM_WaitForRing, ki določa maksimalni čas čakanja. Ko ponorni TT sprejme predvideno število zvonjenj, se postavi v stanje pogovora. Procesu ASTMux se pošljeta dva signala PS_Status. Prvi sporoči novo stanje testnega telefona, drugi pa oznani uspešno sprejeti klic (poveča se števec NoAccCalls). Naslednji ukaz zahteva dejanski dvig slušalke in s tem tudi fizično potrditev sprejema zveze. Pavza v akciji dviga slušalke predstavlja dolžino pogovora za ponorno stran. Po zahtevanem času se izvrši zahteva po pologu slušalke (CV_OnHook). Kot zadnji ukaz se, podobno kot v izvornem scenariju, izvede CV_EOScenario. Le-ta poskrbi za pravilno osvežitev števca uspešno zaključenih scenarijev za ponorni TT.

S tem smo zaključili z opisom delovanja *Generatorja klicev*. Opisali smo osnovne na-

loge posameznih procesov ter pojasnili, kako se izvajajo. Za natančnejši pregled delovanja predlagamo pregled priloge B.

4.5 Realizacija ADT operatorjev

V nasprotju s programskimi jeziki, kjer nove podatkovne tipe sestavljamo iz obstoječih primitivov, je v specifikacijskem jeziku, kot je SDL, pomembno, da opis podatkovnih tipov ni vezan na implementacijske podrobnosti. Opis lastnosti abstraktnega podatkovnega tipa je potrebno opisati na formalen način. Opis ADT-jev je statičen, torej se s časom ne spreminja. ADT definira množico vrednosti, ki jih spremenljivka tega tipa lahko zavzame (literale), množico operatorjev in opis lastnosti operatorjev. Algebra, s katero so podatki modelirani, vsebuje imenovane vrste (designated sorts) in množico operatorjev, ki preslikavajo med različnimi sortami. Vsaka vrsta je množica vrednosti, ki jih generira množica operatorjev te vrste. Formalen opis operatorjev je za vsak netrivialen podatkovni tip zelo kompliciran. Zahteva precej matematičnega znanja in izkušenj za dokazovanje pravilnosti in popolnosti enačb, kot tudi izbire osnovnih konstruktorjev. Konstruktorji so podmnožica operatorjev, s katerimi lahko generiramo vse vrednosti sorte.

Verilog je problem komplicirane definicije ADT operatorjev rešil tako, da je omogočil definicijo operatorjev v ločenih C datotekah. Izvede se preslikava med podatkovnimi tipi v SDL-u in podatkovnimi tipi jezika C. Funkcije, ki so definirane v jeziku C, se prevedejo. Prevedena koda se nato poveže z drugim delom aplikacije, ki je pisan v SDL-u, v enotno izvršilno datoteko [13, 20]. S tem se izdelava ADT operatorjev močno poenostavi. Hkrati pa se izgubi implementacijska neodvisnost, saj so programi v C-ju vezani na obstoječo strojno opremo. To nas v tem primeru ne moti, saj razvijamo programsko kodo za znani ciljni sistem. V primeru, da se strojna oprema spremeni do te mere, da obstoječa programska koda ne bo delovala, je potrebno ponovno napisati samo ADT operatorje. Preostali SDL opis, ki predstavlja večino vloženega dela, lahko ostane nespremenjen. S tem se doseže kompromis med praktičnim in hitrim razvojem ter neodvisnostjo od strojne opreme.

Pri pisanju ADT operatorjev s pomočjo programskega jezika C, kot to omogoča Verilogovo orodje Geode, se je potrebno držati določenih pravil.

V SDL kodi definiramo nov tip, ki vsebuje želene ADT operatorje. Za vsak tip se ob prevajanju SDL sintakse v C-jevsko programsko kodo avtomatsko generira datoteka

u_t_ime_tipa.h. Tej datoteki je pridružena datoteka u_t_ime_tipa.c s programsko kodo v jeziku C. To datoteko tvori razvijalec. Tipe torej definiramo tako, da vsebujejo vse operatorje, ki jih želimo združiti znotraj ene programske datoteke.
Primer definicije tipa:

```
NEWTYPE T_IME_TIPa
  LITERALS L_IME_TIPA
  OPERATORS
  ADTop1: T_Tip_vh_sprem_1, T_Tip_vh_sprem_2 -> T_Tip_izh_sprem COMMENT '#c_extern';
  ADTop2: -> c_uchar COMMENT '#c_extern';
ENDNEWTYPE T_IME_TIPa COMMENT '#c_export';
```

S tem zapisom smo definirali dva ADT operatorja. Prvi ima dva vhodna parametra tipa T_Tip_vh_sprem_1 in T_Tip_vh_sprem_2, drugi pa nima vhodnih spremenljivk. V primeru, ko operator kot rezultat svojega dela ne vrača nobenega rezultata, imamo t.i. "void" funkcijo. C ne dopušča t.i. "void" funkcij, zato izberemo kot tip izhodne vrednosti u_char, ki zasede najmanj pomnilniškega prostora. Poleg že omenjenih datotek je potrebno tvoriti še u_t_ime_tipa.mk. To datoteko potrebuje prevajalnik. Vsaka *.c datoteka mora imeti svojo *.mk datoteko. Da se bo sistem zavedal novega ADT operatorja, je potrebno dodati zapise tudi v datoteko ime_projekta.umk. Če želimo ADT operator uporabljati tudi v SDL simulatorju, moramo dodati zapise tudi v datoteko ime_projekta.cmd.

Verilog je poskrbel tudi za preslikavo SDL tipov in struktur v programski prostor jezika C. Poudariti velja, da je velikost črk pomembna.
Veljajo naslednje preslikave:

Zaradi večje preglednosti in lažjega razumevanja delovanja *Generatorja klicev* bomo večino uporabljenih ADT operatorjev na kratko opisali. S pregledom priložene programske kode in SDL opisa se da razbrati natančno delovanje. Za razumevanje delovanja bo zadoščal kratek opis.

ADT operatorji za dostop do baze podatkov

1. V_Gen=DBeGenConf(V_Err) iz baze podatkov prebere tabelo 4.1 (Call_gen_param). Vsebino zapiše v spremenljivko naslednje strukture:

Tabela 4.6: Preslikava med SDL in C

SDL definicija	C preslikava
NEWTYPE T_Adt	GU_T_ADT
SYNONYM CV_Max	sym_cv_max
NEWTYPE T_Oper OPERATOR Func:	 gu_T_OPER_func
STRUCT sktruktura F_SPREM1 T_tip1 F_SPREM2 T_tip2	(GU_T_STRUCT *p) p → fd_f_sprem1 p → fd_f_sprem2

```
NEWTYPE T_Gen
  STRUCT
    F_StartTime         T_Time;
    F_Restart_Period    NATURAL;
    F_EndTime           T_TIME;
    F_Action_Pause_Unit NATURAL;
    F_ResetTrafficData  BOOLEAN;
    F_Write TrafficData BOOLEAN;
ENDNEWTYPE T_Gen;
```

2. `V_TT:=DBeTTConf(V_IndexNo, V_Err)` iz baze podatkov prebere tabelo 4.2 (`test_telephone`). Vsebino zapiše v spremenljivko naslednje stukture:

```
NEWTYPE T_TT
  STRUCT
    F_TT_id           T_TT_id;
    F_STbus_id        T_STbus;
    F_STbus_ci        T_CI;
    F_TT_delay        NATURAL; /* 100 ms */
    F_Call_Duration_H NATURAL; /* 100 ms */
```

```
        F_Call_Duration_L NATURAL; /* 100 ms */
        F_Short_Pause_H   NATURAL; /* 100 ms */
        F_Short_Pause_L   NATURAL; /* 100 ms */
        F_Long_Pause_H    NATURAL; /* 1 s */
        F_Long_Pause_L    NATURAL; /* 1 s */
    ENDNEWTYPE T_TT;
```

3. `V_Actions:=DBeTTActions(V_IndexNo, V_Err)` iz baze podatkov prebere tabelo 4.3 (`tt_action`). `V_Actions` je `array [T_ActionNo | T_Action]`. Ključ dostopa do posamezne akcije je njena zaporedna ševilka.
   `T_Action` je struktura:

```
    NEWTYPE T_TT
      STRUCT
        F_OutAction NATURAL;
        F_OutPause  NATURAL;
        F_InAction  NATURAL;
        F_InPause   NATURAL;
        F_RecLine   T_TT_id;
    ENDNEWTYPE T_Action;
```

4. `V_Dummy:=DBmASTMData(V_IndexLast, V_TTData, V_GenData!F_Num_Gen_Starts, V_GenData!F_LastStart, V_Err)` vrne brezpomensko celoštevilčno vrednost.
   Naloga operatorja je shranjevanje prometnih podatkov generatorja v bazo podatkov.
   `V_TTData` je `array [T_TT_id | T_Data]`.
   Struktura `T_Data`:

```
    NEWTYPE T_Data
        F_NoOutCalls NATURAL;
        F_NoSuccDial NATURAL;
        F_NoInCalls  NATURAL;
        F_NoAccCalls NATURAL;
```

```
    F_Status     T_Status; /* Tega ne zapisujemo v bazo */
ENDNEWTYPE T_Data;
```

ADT operatorji za CGI vmesnik

5. `V_Dummy:=CGImInitData` inicializira obe pomnilniški tabeli.

6. `V_Dummy:=CGImGenData(V_Gen, V_GenData, V_Err)` zapiše trenutne nastavitve *Generatorja klicev* (`V_Gen`) in skupne prometne podatke (`V_GenData`) v pomnilniške tabele. S tem omogočimo pregled teh podatkov s spletnim pregledovalnikom.

   Spremenljivki imata naslednjo strukturo:

```
NEWTYPE T_Gen
    F_StartTime         T_TIME;
    F_Restart_Period    NATURAL;
    F_EndTime           T_TIME;
    F_Action_Pause_Unit NATURAL;
    F_ResetTrafficData  BOOLEAN;
    F_WriteTrafficData  BOOLEAN;
ENDNEWTYPE T_Data;
NEWTYPE T_GenData
    F_GenNoOutCalls NATURAL;
    F_GenNoInCalls  NATURAL;
    F_GenNoAccCalls NATURAL;
    F_GenNoSuccDial NATURAL;
    F_Num_Gen_Starts NATURAL;
    F_LastStart     T_TIME;
ENDNEWTYPE T_Data;
```

7. `V_Dummy:=CGImTTData(V_TT_id, V_Err)` zapiše spremenljivko `V_Data` v pomnilniško tabelo s prometnimi podatki `TT`-jev.

   Spremenljivka ima naslednjo strukturo:

```
NEWTYPE T_GenData
    F_NoOutCalls   NATURAL;
    F_NoSuccDial   NATURAL;
    F_NoInCalls    NATURAL;
    F_NoAccCalls   NATURAL;
    F_Status       T_Status;
ENDNEWTYPE T_Data;
```

Drugi ADT operatorji

1. `DURATION:=DiffTime(T_TIME, T_TIME)` vrne razliko v sekundah. Tip `T_TIME` ima naslednjo strukturo:

```
NEWTYPE T_TIME
    F_WDay  T_WDay; /* c_uchar */
    F_Day   T_Day; /* c_uchar */
    F_Month T_Month; /* c_uchar */
    F_Year  T_Year; /* NATURAL */
    F_Hour  T_Hour; /* c_uchar */
    F_Min   T_Min; /* c_uchar */
    F_Sec   T_Sec; /* c_uchar */
    F_100ms T_100 ms; /* c_uchar */
ENDNEWTYPE T_TIME;
```

2. `INTEGER:=Int_AND(INTEGER, INTEGER)` je operator, ki izvede logično operacijo IN nad biti dveh spremenljivk tipa `INTEGER`. Tip `INTEGER` je na ciljnem sistemu dolg štiri besede.

Tu smo opisali samo ADT operatorje, ki so nastali med razvojem *Generatorja klicev*. Vsi ostali uporabljeni operatorji so na ciljnem sistemu že obstajali.

4.6 Primer praktične uporabe

V tem razdelku bomo ponovno opisali že poznani primer scenarija. Tu se ne bomo poglabljali v delovanje *Genaratorja klicev*. Opisali bomo celoten postopek, ki ga je potrebo izvesti pri praktični uporabi. Do sedaj se še nismo dotaknili uporabe baze podatkov, zato bomo prve vrstice namenili temu.

Konfiguracija Generatorja klicev

Vse nastavitve *Generatorja klicev* se zapišejo v bazo podatkov. To lahko storimo na dva načina. Prvi, elegantnejši, je uporaba programa `irtsql` in SQL sintakse. Drugi, hitrejši za manjše popravke, pa je neposredno urejanje datotek baze podatkov. Za enostavnejše vnašanje in boljši pregled predlagamo, da si razvijalec za vsak scenarij napiše skupino skript. Predlagamo naslednje:

- Skripte za vpis v bazo podatkov.
- Skripte za pregled zapisov v bazi podatkov.
- Skripte za brisanje podatkov.

Pri uporabi *Generatorja klicev* smo uporabili naslednje skripte:

1. Skripta za skupne nastavitve *Generatorja klicev*. Skripto smo poimenovali `gen`.

```
irtsql $1 "UPDATE call_gen_param SET start_date=19990202"
irtsql $1 "UPDATE call_gen_param SET start_time=505"
irtsql $1 "UPDATE call_gen_param SET end_date=19990416"
irtsql $1 "UPDATE call_gen_param SET end_time=1600"
irtsql $1 "UPDATE call_gen_param SET restart_period=0"
irtsql $1 "UPDATE call_gen_param SET reset_traffic_data=0"
irtsql $1 "UPDATE call_gen_param SET write_traffic_data=0"
irtsql $1 "UPDATE call_gen_param SET action_pause_unit=10"
```

   Trenutno je potrebno za vsak ponovni zagon generatorja popraviti zapise v tabeli `call_gen_param` in resetirati centralo. Ko bo realizirano tudi upravljanje s pomočjo

spletnega pregledovalnika, reset telefonske centrale ne bo več potreben. Trenutno je za vsak ponovni zagon potrebno popraviti datum oz. uro starta generatorja. V primeru, da želimo zagnati generator takoj po zagonu centrale, zadostuje, da v polju za zagon generatorja nastavimo uro oz. datum v preteklost.

2. Skripta za nastavitev testnih priključkov. Skripto smo poimenovali `att2`. Število v imenu skripte predstavlja število uporabljenih testnih priključkov. V našem enostavnem primeru uporabljamo samo dva testna telefona.

```
att $1 '(1,14,1,1,0,0,100,10,100,10,0,0,0,0)'
att $1 '(2,14,2,2,0,0,100,10,100,10,0,0,0,0)'
```

Vidimo, da skripta uporablja osnovno skripto za dodajanje testnih priključkov (`att`).

```
irtsql $1 "INSERT INTO test_telephone VALUES $2"
```

3. Skripta za vpis akcij posameznih TT-jev[20] (ttaction2):

```
irtsql $1 "INSERT INTO tt_action VALUES (1,1,99,NULL,20,300,0)"
irtsql $1 "INSERT INTO tt_action VALUES (1,2,NULL,NULL,2,100,NULL)"
irtsql $1 "INSERT INTO tt_action VALUES (1,3,NULL,NULL,3,100,NULL)"
irtsql $1 "INSERT INTO tt_action VALUES (1,4,NULL,NULL,5,100,NULL)"
irtsql $1 "INSERT INTO tt_action VALUES (1,5,NULL,NULL,4,100,NULL)"
irtsql $1 "INSERT INTO tt_action VALUES (1,6,NULL,NULL,66,1500,NULL
irtsql $1 "INSERT INTO tt_action VALUES (1,7,NULL,NULL,30,100,NULL)
irtsql $1 "INSERT INTO tt_action VALUES (1,8,NULL,NULL,99,100,NULL)
irtsql $1 "INSERT INTO tt_action VALUES (2,1,100,2,99,NULL,NULL)"
irtsql $1 "INSERT INTO tt_action VALUES (2,2,20,500,NULL,NULL,NULL)
irtsql $1 "INSERT INTO tt_action VALUES (2,3,30,100,NULL,NULL,NULL)
irtsql $1 "INSERT INTO tt_action VALUES (2,4,99,300,NULL,NULL,NULL)
```

4. Skripta za brisanje akcij za posamezen TT. Poimenovali smo jo rmaction. Tu je potrebno poleg IP naslova centrale podati tudi identifikacijsko število TT-ja, katerega akcije želimo zbrisati iz baze podatkov.

```
irtsql $1 "DELETE FROM tt_action WHERE tt_id=$2"
```

5. Skripta za prikaz nastavitev Generatorja (vgen):

```
irtsql $1 "SELECT * FROM call_gen_param"
```

6. Skripta za prikaz nastavitev posameznega TT-ja (vtt):

```
irtsql $1 "SELECT * FROM call_gen_param"
```

7. Skripta za prikaz akcij posameznega TT-ja (vtt):

```
irtsql $1 "SELECT * FROM tt_action WHERE tt_id=$2"
```

[20] glej tabelo 4.3.

8. Skripta za prikaz vseh podatkov *Generatorja klicev* (`vall`):

```
irtsql $1 "SELECT * FROM call_gen_param "
irtsql $1 "SELECT * FROM test_telephone "
irtsql $1 "SELECT * FROM tt_action "
```

Pred uporabo skript za vnos konfiguracijskih in prometnih podatkov je pametno preveriti trenutno stanje vseh zapisov (`vall`). Če so trenutni zapisi neprimerni, jih je potrebno iz baze podatkov odstraniti.

Za obravnavani primer nam `vall` izpiše:

```
vall maketa7
start_date|start_time|restart_period|end_date|end_time|reset_traffic_data|
write_traffic_data|action_pause_unit|num_gen_starts|last_start_date|
last_start_tim e|
19990202|505|0|19990916|1600|0|0|10|1|19000101|0|
tt_id|stbus_id|stbus_ci|tt_delay|call_dur_h|call_dur_l|short_pause_h|
short_pause_l|long_pause_h|long_pause_l|num_out_calls|num_succ_dials|
num_inc_calls|num_ acc_calls|
1|14|1|1|0|0|100|10|100|10|0|0|0|0|
2|14|2|2|0|0|100|10|100|10|0|0|0|0|
tt_id|tt_action_num|in_call_action|in_action_pause|out_call_action|
out_action_ pause|called_tt_id|
1|1|99|NULL|20|300|0|
1|2|NULL|NULL|2|100|NULL|
1|3|NULL|NULL|3|100|NULL|
1|4|NULL|NULL|5|100|NULL|
1|5|NULL|NULL|4|100|NULL|
1|6|NULL|NULL|66|1500|NULL|
1|7|NULL|NULL|30|100|NULL|
1|8|NULL|NULL|99|100|NULL|
2|1|100|2|99|NULL|NULL|
2|2|20|500|NULL|NULL|NULL|
2|3|30|100|NULL|NULL|NULL|
2|4|99|300|NULL|NULL|NULL|
```

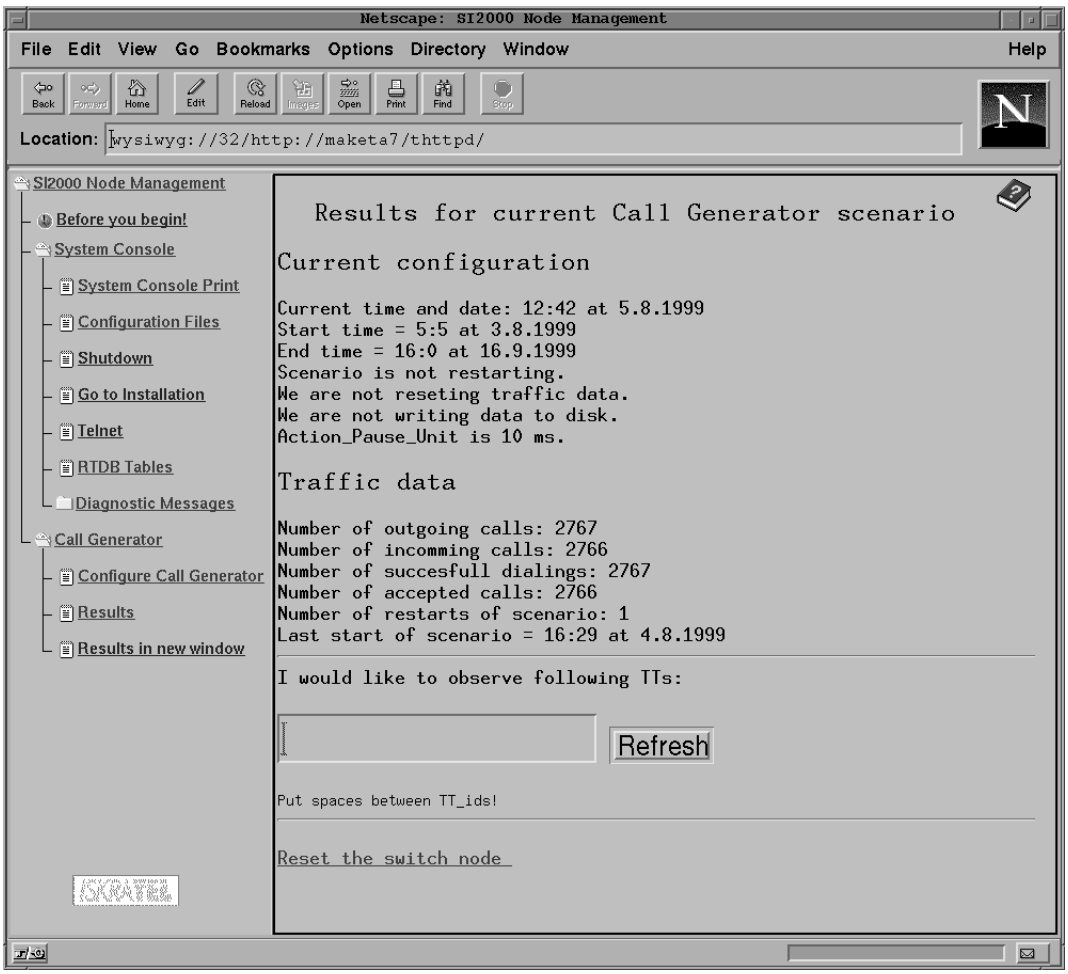
Kot edini ukazni parameter se pri večini opisanih skript uporablja IP naslov testirane telefonske centrale. Ko delamo s centralo, ki se nahaja v maketnem centru, lahko uporabimo kar njeno ime (npr.: maketa7).

Ko so vsi zapisi v bazi podatkov pravilni, resetiramo centralo. Pri ponovnem zagonu se *Generator klicev* nastavi podatkom primerno. Ob želenem datumu in času dneva se *Generator klicev* zažene in prične z izvajanjem scenarija. Rezultate opazujemo s pomočjo spletnega pregledovalnika s kateregakoli računalnika na omrežju.

V našem primeru smo izvajali teste na centrali *maketa7*. Da pridemo do rezultatov, moramo v polje za vnos URL naslova vpisati naslednjo vrstico:

`http://maketa7/httpd/`

V osnovnem oknu (glej sliko 4.5), ki vsebuje tudi kazalo, je najbolje opazovati skupne podatke.



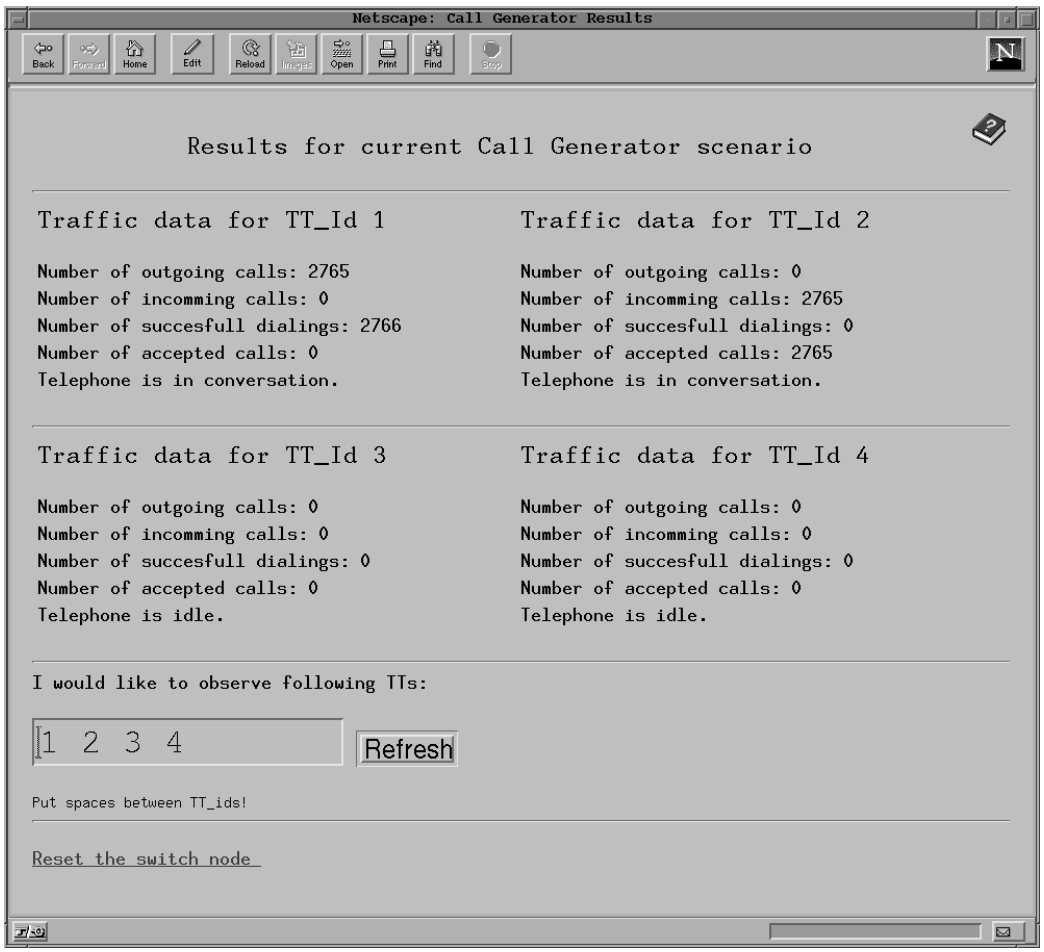
Slika 4.5: Skupni prometni podatki

Osnovno okno je razdeljeno na dva dela. Na levi strani se nahaja kazalo, na desni pa je prikazana trenutna izbira. Z izbiro povezave *Call Generator* se odpre podmeni z naslednjimi

izbirami:

- Configure Call Generator: ker ta del programa še ni realiziran, se prikažejo skupni prometni podatki;
- Results: v desnem delu okna prikaže prometne podatke;
- Results in new window: prometne podatke prikaže v novem oknu.

Prometne podatke posameznih TT-jev pa je primerneje opazovati v novem oknu, ki ima več prostora za prikaz podatkov. Program je narejen tako, da se prometni podatki v primeru sodega števila izbranih TT-jev prikazujejo v dveh stolpcih. Tako lahko opazujemo izvorni in ponorni TT vzporedno. S tem dobimo nazoren pogled na dogajanje testnih priključkov. Izgled okna prikazuje slika 4.6.



Slika 4.6: Pregled delovanja izbranih TT-jev

5 SKLEP

Izdelali smo programski *Generator klicev* za IskraTEL-ovo centralo MLB SI2000 V5. Delo sem opravljal na sedežu podjetja IskraTEL v Kranju. Ob izdelavi *Generatorja klicev* sem sodeloval pri vseh stopnjah razvoja — od ideje, ki je prišla s strani g. Jožeta Gašpariča, do končne izvedbe. Prve dni sem se posvetil spoznavanju organizacijske sheme podjetja. Pridružil sem se razvojni ekipi z imenom RDSA7. Vodja ekipe je g. Simon Molka, ki ima bogate izkušnje z razvojem programske opreme v telekomunikacijah. Vsi sodelavci so me zelo toplo sprejeli in mi vedno z veseljem pomagali. Pri tako velikem projektu, kot je razvoj programske opreme za telefonsko centralo, je kjučnega pomena dobra organiziranost in timsko delo. Prva dva tedna sem porabil za spoznavanje programske opreme telefonskih central SI2000 V5. Pomagal sem si s SDL simulatorjem, ki mi je omogočil natančno analizo vseh signalov. Nato sem začel razmišljati o najboljši možni realizaciji zastavljene naloge. Pri tem sem kot glavno orodje uporabljal svinčnik in papir. Nastala je množica skic delovanja in različnih umestitev *Generatorja klicev* v obstoječo programsko opremo. Vse sem natančno analiziral s pomočjo g. Simona Molke. Brez njegovih izkušenj bi spregledal marsikatero podrobnost. Ob samem usklajevanju in razgovoru o funkcionalnih zahtevah sem še natančneje spoznal organiziranost programske opreme. Skozi množico sestankov smo usklajevali in preverjali svoje zamisli tudi z ostalimi razvojnimi skupinami — predvsem s sistemsko skupino, ki je avtor procesa `scandrv`. Rezultat usklajevanj je bila skupno sprejeta umestitev *Generatorja klicev* v obstoječo programsko opremo. Le-ta je bila rahlo drugačna od tiste, ki smo jo predvideli z našo skupino. Zato ima prva verzija *Generatorja klicev* manjšo pomanjkljivost: pri prevelikem številu testnih telefonov se preobremeni povezava med CDA in CVA stranjo. V tem trenutku je ta pomanjkljivost praktično že odpravljena. Namesto da bi `TT`-ji pošiljali pakete neposredno `CASMUX`-u, jih pošiljajo svojemu managerju (`TTmux`). Le-ta sedaj vse spremembe, ki se zgodijo v zadnjih štirih milisekundah, zapakira v paket `PerInfo` in ga pošlje `CASMUX`-u, enako, kot to počne `scandrv` za običajne naročniške priključke.

Temu je sledila izvedba zamisli s pomočjo SDL-ja. Teoretične osnove SDL-ja sem pridobil že med študijem na fakulteti. Pri predmetu ”Programska oprema v telekomunikacijah” smo skozi strokovne članke spoznali osnovno idejo SDL-ja. Tako mi samo spoznavanje SDL-ja ni vzelo dosti časa.

Zaradi velike količine razvijalcev se je pri delu potrebno držati nekaterih pravil. Vedno moraš imeti v mislih dejstvo, da se bo tvoja programska koda v prihodnosti popravljala. Verjetnost, da je ne boš popravljal sam, je velika. Kljub temu, da je *Generator klicev* že uporaben, še ni dosegel svoje končne oblike. Razvoj še vedno teče. Mojo programsko kodo torej v tem trenutku dopolnjujejo in spreminjajo drugi razvijalci. Od trenutka, ko je *Generator klicev* prvič zaživel na telefonski centrali v maketnem centru, sem pogosto uporabljal tudi sledilnik signalov in razhroščevalnik [16]. Ravno v maketnem centru sem preživel večino zadnjega razvojnega obdobja. V tem času sem tesno sodeloval z Simono Turk in ostalimi. Prvi odziv vseh sodelujočih je bil zelo pozitiven. Predvsem jim je bila všeč enostavna konfiguracija in prijeten pregled rezultatov.

Prva verzija *Generatorja klicev* je bila zaključena s tem diplomskim delom. Njegov razvoj pa se že nadaljuje. Spreminja se način komunikacije med CDA in CVA stranjo, dodaja se upravljanje s pomočjo spletnega pregledovalnika, pišejo se programi za avtomatsko generiranje scenarijev, ... V kratkem bo *Generator klicev* dobil veliko število uporabnikov, ki bodo zagotovo izrazili željo po dodatnih funkcijah. Razvoj se torej še ne bo zaključil. Verjamem, da bo *Generator klicev* približal testiranje programske opreme vsem razvijalcem in jim s tem olajšal njihovo delo. Poleg osebnega zadovoljstva sem pri izdelavi diplomskega dela dobil tudi mnogo neprecenljivih delovnih izkušenj, ki mi bodo pomagale pri nadaljnjem strokovnem izpopolnjevanju. Zato vsem, ki razmišljate o sodelovanju s podjetji pri realizaciji svoje diplomske naloge, to toplo priporočam.

LITERATURA

[1] CCITT specification and description language (SDL), 1993. ITU-T Recommendation Z.100
Series Z: Programming Languages.

[2] Initial algebra model and SDL predefined data, 1993. ITU-T Recommendation Z.100 Annexes C and D
Series Z: Programming Languages, Specification and Description Language (SDL).

[3] SDL methodology guidelines, SDL bibliography, 1993. ITU-T Recommendation Z.100 Appendices I and II
Series Z: Programming Languages, Specification and Description Language (SDL).

[4] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with Applications from Protocol Specification*. Prentice Hall International (UK) Ltd., 1991. ISBN = 0-13-785890-6.

[5] Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. *SDL Formal Object-oriented Language for Communicating Systems*. Prentice Hall Europe, 1997. ISBN = 0-13-621384-7.

[6] Mira Frelih. *Uvod v uporabo dopolnilnih storitev*. IskraTEL d.o.o, Kranj, Slovenija. Official code = KSS2302BA-EDL-010.

[7] Michael Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley Longman, Inc., eighth edition, 1997.

[8] Stanislav Kajzer and Sašo Erzin. *Enota analognih naročniških vezij verzija C SAC*. IskraTEL d.o.o, Kranj, Slovenija. Official code = UTA5011AA-PML-020.

[9] Linda Lamb and Arnold Robbind. *Learning the vi Editor*. O'REILLY, sixth edition, 1998.

[10] Leslie Lamport. *A Document Preparation System LaTeX User's Guide and Reference Manual*. Addison Wesley Longman, Inc., sixth edition, 1997.

[11] Marjan Markun. *Diagnostične funkcije za dostop do podatkovne baze*. IskraTEL d.o.o, Kranj, Slovenija. Official code = FUN323000-PCL-020.

[12] Dejan Potočnik. *Pregledovanje dinamičnih podatkov*. IskraTEL d.o.o, Kranj, Slovenija. Št. zapisa v SOPRAN-u = 68510.

[13] Ana Robnik. *Ročno pisanje C kode za ADT operatorje in zunanje procese ter taske*. IskraTEL d.o.o, Kranj, Slovenija. Št. zapisa v SOPRAN-u = 63671.

[14] Ana Robnik and Matjaž Dolenc. *Baza simbolov za verzijo pet in več*. IskraTEL d.o.o, Kranj, Slovenija. Št. zapisa v SOPRAN-u = 62955.

[15] Ana Robnik and Uroš Srakar. *Jezik SDL in orodje GEODE*. IskraTEL d.o.o, Kranj, Slovenija, Marec 1996. Gradivo za tečaj "Programiranje za SI2000".

[16] Sani Rus. *Iskanje napak v SDL opisu aplikacije XRAY+ debugger*. IskraTEL d.o.o, Kranj, Slovenija. Št. zapisa v SOPRAN-u = 61721.

[17] Mirko Šaranovič and Tomaž Mohorič. *Podatkovna baza v realnem času*. IskraTEL d.o.o, Kranj, Slovenija. Št. zapisa v SOPRAN-u = 65924.

[18] Besnik Shantri, Simon Molka, Niko Vidic, Marko Koželj, and Vlasta Vidic. *Programska oprema signalizacij*. IskraTEL d.o.o, Kranj, Slovenija. Official code = FUN277000-PDL-010.

[19] Meta Slatnar. *Montažni priročnik SI2000 V5*. IskraTEL d.o.o, Kranj, Slovenija. Official code = KSS1180G0-EDL-01A.

[20] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1998.

PRILOGA A: ASTM OPERATORJI

```
/*#########################################################################*/
/*#                                                                       #*/
/*#                                                                       #*/
/*#            Copyright (c) 1998 IskraTEL                                #*/
/*#                                                                       #*/
/*#                                                                       #*/
/*#                                                                       #*/
/*# Name        : u_t_cgiastm.c                                           #*/
/*#                                                                       #*/
/*# Description : Functions for writing CGI data to common memory         #*/
/*#                                                                       #*/
/*# Code        : KHSD - XAG6703                                          #*/
/*#                                                                       #*/
/*# Date        : Oct 1998                                                #*/
/*#                                                                       #*/
/*# Author      : Bostjan Vlaovic RDSA7                                   #*/
/*#                                                                       #*/
/*# Translation : in makefile u_t_difftime.mk                             #*/
/*#                                                                       #*/
/*# Remarks     :                                                         #*/
/*#                                                                       #*/
/*# Revisions   :                                                         #*/
/*#                                                                       #*/
/*#########################################################################*/
/*****************************************************************/
/*          UNIX - SCCS  VERSION DESCRIPTION                     */
/*****************************************************************/
static char unixid[] = "%W% %D% u_t_cgiastm.c";

/*****************************************************************/
/*          L I B R A R I E S                                    */
/*****************************************************************/

# ifdef SIM
# include <time.h>
# include <string.h>
# endif

# include <stdio.h>
# include "u_t_cgiastm.h"
# include "u_t_gettime.h"



#ifdef RTE_PSOS
# include "htmlutil.h"
#include <psos.h>
#include <drv_intf.h>
#include <pna.h>
#include <prepc.h>
#include "sys_conf.h"
#include "bsp.h"
#include <bspfuncs.h>
#include <gsblk.h>
#include "g2_appli.h"
#include "r_cmsc.h"
#include "htmlutil.h"
#endif /* RTE_PSOS */
#include "rte_conf.h"



# ifdef SIM
   /*
    * SET THE VALUES FROM THE GEODE/RTE
   */
#  ifdef __STDC__
#   define ANSI
#  else
```

```
#   define NOANSI
#  endif
# endif


/*****************************************************************/
/*         SYMBOLIC CONSTANTS                                    */
/*****************************************************************/

/*-------------*/
/* HTML macros */
/*-------------*/

#define CENTER          w3_printf("<CENTER>\n");
#define _CENTER         w3_printf("</CENTER>\n");
#define BR              w3_printf("<BR>\n");
#define _BR             w3_printf("</BR>\n");
#define HR              w3_printf("<HR>\n");
#define _HR             w3_printf("</HR>\n");
#define H1              w3_printf("<H1>\n");
#define _H1             w3_printf("</H1>\n");
#define H2              w3_printf("<H2>\n");
#define _H2             w3_printf("</H2>\n");
#define TR              w3_printf("<TR>\n");
#define _TR             w3_printf("</TR>\n");
#define PRE             w3_printf("<PRE>");
#define _PRE            w3_printf("</PRE>\n");
#define RIGHT           w3_printf("<P ALIGN=RIGHT>\n");
#define _RIGHT          w3_printf("</P>\n");

# ifdef RTE_PSOS
#       define YEAR(date)       ((date >> 16) & 0xffff)
#       define MONTH(date)      ((date >> 8) & 0xff)
#       define DAY(date)        (date & 0xff)
#       define HOUR(time)       ((time >> 16) & 0xffff)
#       define MINUTE(time)     ((time >> 8) & 0xff)
#       define SECOND(time)     (time & 0xff)
# endif

/*****************************************************************************/
/*                          P R O T O T Y P E S                              */
/*****************************************************************************/
void  cgconf();
void  cgres();


/*****************************************************************************/
/*          GLOBAL  DATA  STRUCTURES                                         */
/*****************************************************************************/

typedef struct
{
GU_T_TIME starttime;
unsigned int RestartPeriod;
GU_T_TIME endtime;
unsigned int ActionPauseUnit;
SDL_BOOLEAN ResetTrafficData;
SDL_BOOLEAN WriteTrafficData;
unsigned int GenNoOutCalls;
unsigned int GenNoInCalls;
unsigned int GenNoAccCalls;
unsigned int GenNoSuccDial;
unsigned int NumGenStarts;
GU_T_TIME laststart;
}GEN_DATA_T;

typedef struct
{
unsigned int NoOutCalls;
unsigned int NoSuccDial;
unsigned int NoInCalls;
```

```
unsigned int NoAccCalls;
unsigned int Status;
}TT_DATA_T;


/*****************************************************************/
/*          SYMBOLIC  CONSTANTS                                  */
/*****************************************************************/

#define MAX_TT_DATA 100

/*****************************************************************/
/*          DATA  DECLARATIONS                                   */
/*****************************************************************/

GEN_DATA_T gen_data;
TT_DATA_T tt_data[MAX_TT_DATA];

/*****************************************************************/
/*          F U N C T I O N S                                    */
/*****************************************************************/

#ifndef __STDC__
uchar gu_T_CGIASTM_cgiminitdata()
#else  /* __STDC__ */
uchar gu_T_CGIASTM_cgiminitdata(void)
#endif /* __STDC__ */
{
   int i;
   gen_data.RestartPeriod = 0;
   gen_data.starttime.fd_f_wday = 1;
   gen_data.starttime.fd_f_day = 1;
   gen_data.starttime.fd_f_month = 1;
   gen_data.starttime.fd_f_year = 1999;
   gen_data.starttime.fd_f_hour = 0;
   gen_data.starttime.fd_f_min = 0;
   gen_data.starttime.fd_f_sec = 0;
   gen_data.starttime.fd_f_100ms = 0;
   gen_data.endtime.fd_f_wday = 0;
   gen_data.endtime.fd_f_day = 0;
   gen_data.endtime.fd_f_month = 0;
   gen_data.endtime.fd_f_year = 0;
   gen_data.endtime.fd_f_hour = 0;
   gen_data.endtime.fd_f_min = 0;
   gen_data.endtime.fd_f_sec = 0;
   gen_data.endtime.fd_f_100ms = 0;
   gen_data.ActionPauseUnit = 0;
   gen_data.ResetTrafficData = FALSE;
   gen_data.WriteTrafficData = TRUE;
   gen_data.GenNoOutCalls = 0;
   gen_data.GenNoInCalls = 0;
   gen_data.GenNoAccCalls = 0;
   gen_data.GenNoSuccDial = 0;
   gen_data.NumGenStarts = 0;
   gen_data.laststart.fd_f_wday = 0;
   gen_data.laststart.fd_f_day = 0;
   gen_data.laststart.fd_f_month = 0;
   gen_data.laststart.fd_f_year = 0;
   gen_data.laststart.fd_f_hour = 0;
   gen_data.laststart.fd_f_min = 0;
   gen_data.laststart.fd_f_sec = 0;
   gen_data.laststart.fd_f_100ms = 0;

   for ( i = 0 ; i< MAX_TT_DATA ; i++ )
     {
       tt_data[i].NoOutCalls = 0;
       tt_data[i].NoSuccDial = 0;
       tt_data[i].NoInCalls = 0;
       tt_data[i].NoAccCalls = 0;
       tt_data[i].Status = 0;
     }
```

```
   return(0);
}


#ifdef NOANSI
uchar gu_T_CGIASTM_cgimgendata(gen, gendata, gc_pt_3)
 GU_T_GEN *gen;
 GU_T_GENDATA *gendata;
 GU_T_ERR *gc_pt_3;
#else
uchar gu_T_CGIASTM_cgimgendata(GU_T_GEN *gen , GU_T_GENDATA *gendata, GU_T_ERR *gc_pt_3)
#endif
{
  /*memcopy((void*) & (gendata.starttime),(void*) & (gen->fd_f_starttime), sizeof(GU_T_TIME));*/
  gen_data.starttime = gen->fd_f_starttime;
  gen_data.RestartPeriod = gen->fd_f_restart_period;
  gen_data.endtime = gen->fd_f_endtime;
  gen_data.ActionPauseUnit = gen->fd_f_action_pause_unit;
  gen_data.ResetTrafficData = gen->fd_f_resettrafficdata;
  gen_data.WriteTrafficData = gen->fd_f_writetrafficdata;
  gen_data.GenNoOutCalls = gendata->fd_f_gennooutcalls;
  gen_data.GenNoInCalls = gendata->fd_f_gennoincalls;
  gen_data.GenNoAccCalls = gendata->fd_f_gennoacccalls;
  gen_data.GenNoSuccDial = gendata->fd_f_gennosuccdial;
  gen_data.NumGenStarts = gendata->fd_f_num_gen_starts;
  gen_data.laststart = gendata->fd_f_laststart;
  return(0);
}


#ifdef NOANSI
uchar gu_T_CGIASTM_cgimttdata(TT_id, data, gc_pt_3)
GU_T_TT_ID TT_id;
GU_T_DATA *data;
GU_T_ERR *gc_pt_3;
#else
uchar gu_T_CGIASTM_cgimttdata(GU_T_TT_ID TT_id, GU_T_DATA *data, GU_T_ERR *gc_pt_3)
#endif
{
  tt_data[TT_id].NoOutCalls = data->fd_f_nooutcalls;
  tt_data[TT_id].NoSuccDial = data->fd_f_nosuccdial;
  tt_data[TT_id].NoInCalls = data->fd_f_noincalls;
  tt_data[TT_id].NoAccCalls = data->fd_f_noacccalls;
  tt_data[TT_id].Status = data->fd_f_status;

  return(0);
}
```

```
/*#########################################################################*/
/*#                                                                       #*/
/*# FUNCTION    : cgconf()                                                #*/
/*#                                                                       #*/
/*# IN par      :                                                         #*/
/*#                                                                       #*/
/*# OUT par     : 0 - OK                                                  #*/
/*#                                                                       #*/
/*# Description : CGI program for Call Generator configuration            #*/
/*#                                                                       #*/
/*# Author      : Bostjan Vlaovic RDSA7                                   #*/
/*#                                                                       #*/
/*#########################################################################*/

#ifdef NOANSI
void cgconf( argc, argv, envp )
                                        int argc;
                                        char *argv[];
                                        char *envp[];
#else
void cgconf(int argc, char *argv[], char *envp[])
#endif
{

 char Command[10];
 unsigned long   date;
 unsigned long   time;

 start_Document( "Call Generator Results", "/thttpd/docs/cghelp.html");
 CENTER
 H1
 w3_printf("Configuring Call Generator");
 _H1
 _CENTER

 H2
 w3_printf("Current configuration");
 _H2
 w3_printf("Current time and date: %d:%d at %d.%d.%d <BR>", HOUR(time), MINUTE(time), DAY(date),\
 MONTH(date), YEAR(date));
 w3_printf("Start time = %d:%d at %d.%d.%d<BR>", gen_data.starttime.fd_f_hour, gen_data.starttime.fd_f_min,
 gen_data.starttime.fd_f_day, gen_data.starttime.fd_f_month, gen_data.starttime.fd_f_year);
 w3_printf("End time = %d:%d at %d.%d.%d<BR>", gen_data.endtime.fd_f_hour, gen_data.endtime.fd_f_min,i\
 gen_data.endtime.fd_f_day, gen_data.endtime.fd_f_month, gen_data.endtime.fd_f_year);
 if (gen_data.RestartPeriod == 0){
    w3_printf("Scenario is not restarting. <BR>");
 }
 else {
    w3_printf("Scenario is restarting every %d seconds! <BR>", gen_data.RestartPeriod);
 }
 if (gen_data.ResetTrafficData == FALSE){
    w3_printf("We are not reseting traffic data. <BR>");
 }
 else {
    w3_printf("We are reseting trafic data for each restart of scenario! <BR>");
 }
 if (gen_data.WriteTrafficData == FALSE){
    w3_printf("We are not writing data to disk.<BR>");
 }
 else {
    w3_printf("We are writing data to disk.<BR>");
 }
 w3_printf("Action_Pause_Unit is %d ms. <BR>", gen_data.ActionPauseUnit);

 H2
 w3_printf("Traffic data");
 _H2
 w3_printf("Number of outgoing calls: %d <BR>", gen_data.GenNoOutCalls);
 w3_printf("Number of incomming calls: %d <BR>", gen_data.GenNoInCalls);
 w3_printf("Number of succesfull dialings: %d <BR>", gen_data.GenNoSuccDial);
```

```
 w3_printf("Number of accepted calls: %d <BR>", gen_data.GenNoAccCalls);
 w3_printf("Number of restarts of scenario: %d <BR>", gen_data.NumGenStarts);
 w3_printf("Last start of scenario = %d:%d at %d.%d.%d<BR>", gen_data.laststart.fd_f_hour,\
 gen_data.laststart.fd_f_min, gen_data.laststart.fd_f_day, gen_data.laststart.fd_f_month,i\
 gen_data.laststart.fd_f_year);

}
```

```
/*##########################################################################*/
/*#                                                                        #*/
/*# FUNCTION    : cgres()                                                  #*/
/*#                                                                        #*/
/*# IN par      : NONE                                                     #*/
/*#                                                                        #*/
/*# OUT par     : 0 - OK                                                   #*/
/*#                                                                        #*/
/*# Description : CGI function for display of Call Generator results       #*/
/*#               in WWW browser                                           #*/
/*#                                                                        #*/
/*# Author      : Bostjan Vlaovic RDSA7                                    #*/
/*#                                                                        #*/
/*##########################################################################*/

#ifdef NOANSI
void cgres( argc, argv, envp )
                                        int argc;
                                        char *argv[];
                                        char *envp[];
#else
void cgres(int argc, char *argv[], char *envp[])
#endif
{

 char Command[10],tmp[3];
 int tt_id, tt_id1, no_tt, i, i1, i2, i_tmp, size;
 int    tt[100];

  unsigned long   date,
                  time,
                  ticks;

 tm_get (&date, &time, &ticks);

/*Reading TT_id's */

 size=strlen(argv[1]);
 i_tmp=0;
 no_tt=0;
 i2=0;
 if(argc>1){
  for(i=0;i<size;i++){
   if(argv[1][i] == ' '){ /*Space is delimiter between TTs*/
    for(i1=i_tmp;i1!=i;i1++){
     tmp[i2]=argv[1][i1];
     i2++;
    }
    i2=0;
    no_tt++;
    tt[no_tt]=atoi(tmp);
    strcpy(tmp,"");
    i_tmp=i+1;
   }
  }
  for(i=i_tmp;i!=size;i++){
   tmp[i2]=argv[1][i];
   i2++;
  }
  no_tt++;
  tt[no_tt]=atoi(tmp);
 }
/* Starting HTML document */

 start_Document( "Call Generator Results", "/thttpd/docs/cghelp.html");
 CENTER
 H1
 w3_printf("Results for current Call Generator scenario");
 _H1
 _CENTER
```

```
 w3_printf("<FORM action='/thttpd/cgi-bin/cgres'>");

if (no_tt == 0){
 H2
 w3_printf("Current configuration");
 _H2
 w3_printf("Current time and date: %d:%d at %d.%d.%d <BR>", HOUR(time), MINUTE(time), DAY(date),\
 MONTH(date), YEAR(date));
 w3_printf("Start time = %d:%d at %d.%d.%d<BR>", gen_data.starttime.fd_f_hour, gen_data.starttime.fd_f_min,\
 gen_data.starttime.fd_f_day, gen_data.starttime.fd_f_month, gen_data.starttime.fd_f_year);
 w3_printf("End time = %d:%d at %d.%d.%d<BR>", gen_data.endtime.fd_f_hour, gen_data.endtime.fd_f_min,\
 gen_data.endtime.fd_f_day, gen_data.endtime.fd_f_month, gen_data.endtime.fd_f_year);
 if (gen_data.RestartPeriod == 0){
    w3_printf("Scenario is not restarting. <BR>");
 }
 else {
    w3_printf("Scenario is restarting every %d seconds! <BR>", gen_data.RestartPeriod);
 }
 if (gen_data.ResetTrafficData == FALSE){
    w3_printf("We are not reseting traffic data. <BR>");
 }
 else {
    w3_printf("We are reseting trafic data for each restart of scenario! <BR>");
 }
 if (gen_data.WriteTrafficData == FALSE){
    w3_printf("We are not writing data to disk.<BR>");
 }
 else {
    w3_printf("We are writing data to disk.<BR>");
 }
 w3_printf("Action_Pause_Unit is %d ms. <BR>", gen_data.ActionPauseUnit);

 H2
 w3_printf("Traffic data");
 _H2
 w3_printf("Number of outgoing calls: %d <BR>", gen_data.GenNoOutCalls);
 w3_printf("Number of incomming calls: %d <BR>", gen_data.GenNoInCalls);
 w3_printf("Number of succesfull dialings: %d <BR>", gen_data.GenNoSuccDial);
 w3_printf("Number of accepted calls: %d <BR>", gen_data.GenNoAccCalls);
 w3_printf("Number of restarts of scenario: %d <BR>", gen_data.NumGenStarts);
 w3_printf("Last start of scenario = %d:%d at %d.%d.%d<BR>", gen_data.laststart.fd_f_hour,\
 gen_data.laststart.fd_f_min, gen_data.laststart.fd_f_day, gen_data.laststart.fd_f_month,\
 gen_data.laststart.fd_f_year);
}

else if((no_tt%2)==0){
 for(i=1;i<no_tt;i=i+2){
  tt_id=tt[i];
  tt_id1=tt[i+1];
  w3_printf("<HR>");
  w3_printf("<TABLE ALIGN=justify WIDTH=\"100%%\">");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
  H2
  w3_printf("Traffic data for TT_Id %d", tt_id);
  _H2
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
  H2
  w3_printf("Traffic data for TT_Id %d", tt_id1);
  _H2
  w3_printf("</TD></TR>");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of outgoing calls: %d <BR>", tt_data[tt_id].NoOutCalls);
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of outgoing calls: %d <BR>", tt_data[tt_id1].NoOutCalls);
  w3_printf("</TD></TR>");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
```

```
  w3_printf("Number of incomming calls: %d <BR>", tt_data[tt_id].NoInCalls);
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of incomming calls: %d <BR>", tt_data[tt_id1].NoInCalls);
  w3_printf("</TD></TR>");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of succesfull dialings: %d <BR>", tt_data[tt_id].NoSuccDial);
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of succesfull dialings: %d <BR>", tt_data[tt_id1].NoSuccDial);
  w3_printf("</TD></TR>");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of accepted calls: %d <BR>", tt_data[tt_id].NoAccCalls);
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
  w3_printf("Number of accepted calls: %d <BR>", tt_data[tt_id1].NoAccCalls);
  w3_printf("</TD></TR>");
  w3_printf("<tr ALIGN=left VALIGN=middle>");
  w3_printf("<TD WIDTH=\"50%%\">");
   switch(tt_data[tt_id].Status){
        case syn_cv_idle:
  w3_printf("Telephone is idle.");
  break;
        case syn_cv_dialing:
  w3_printf("Telephone is dialing.");
  break;
        case syn_cv_ringing:
  w3_printf("Telephone is ringing.");
  break;
        case syn_cv_talking:
  w3_printf("Telephone is in conversation.");
  break;
   }
  w3_printf("</TD>");
  w3_printf("<TD WIDTH=\"50%%\">");
   switch(tt_data[tt_id1].Status){
        case syn_cv_idle:
  w3_printf("Telephone is idle.");
  break;
        case syn_cv_dialing:
  w3_printf("Telephone is dialing.");
  break;
        case syn_cv_ringing:
w3_printf("Telephone is ringing.");
break;
        case syn_cv_talking:
  w3_printf("Telephone is in conversation.");
  break;
   }
  w3_printf("</TD></TR>");
  w3_printf("</TABLE>");
  w3_printf("<P>");
 }
}
else{
 for(i=1;i<no_tt+1;i++){
 tt_id=tt[i];
 H2
 w3_printf("Traffic data for TT_Id %d", tt_id);
 _H2
 w3_printf("Number of outgoing calls: %d <BR>", tt_data[tt_id].NoOutCalls);
 w3_printf("Number of incomming calls: %d <BR>", tt_data[tt_id].NoInCalls);
 w3_printf("Number of succesfull dialings: %d <BR>", tt_data[tt_id].NoSuccDial);
 w3_printf("Number of accepted calls: %d <BR>", tt_data[tt_id].NoAccCalls);
  switch(tt_data[tt_id].Status){
       case syn_cv_idle:
                w3_printf("Telephone is idle.");
                break;
       case syn_cv_dialing:
```

```
                w3_printf("Telephone is dialing.");
                break;
        case syn_cv_ringing:
                w3_printf("Telephone is ringing.");
                break;
        case syn_cv_talking:
                w3_printf("Telephone is in conversation.");
                break;
  }
 }
}

 w3_printf("<HR>");
 w3_printf("I would like to observe following TTs: <P> <INPUT TYPE=text SIZE=15 NAME=TT VALUE=\"");
 if (no_tt>0){
  for(i=1;i<no_tt;i++){
  w3_printf("%d ",tt[i]);
  }
  w3_printf("%d",tt[i]);
  w3_printf("\">");
  w3_printf(" <INPUT TYPE='submit' VALUE='Refresh' target='TT'> </FORM><P>");
  w3_printf("<SMALL>Put spaces between TT_ids! </SMALL>");
  w3_printf("<HR>");
 }
 else{
  w3_printf("\">");
  w3_printf(" <INPUT TYPE='submit' VALUE='Refresh' target='TT'> </FORM><P>");
  w3_printf("<SMALL>Put spaces between TT_ids! </SMALL>");
  w3_printf("<HR>");
 }
 w3_printf("<BR> <A HREF=\"/thttpd/cgi-bin/reset\">Reset the switch node <A>");

end_Document();
}
```

```
/*#########################################################################*/
/*#                                                                       #*/
/*#                                                                       #*/
/*#            Copyright (c) 1998 IskraTEL                                #*/
/*#                                                                       #*/
/*#                                                                       #*/
/*#                                                                       #*/
/*# Name        : u_t_difftime.c                                          #*/
/*#                                                                       #*/
/*# Description : Operator for returning difference between two times     #*/
/*#               T_TIME in seconds                                       #*/
/*#                                                                       #*/
/*# Code        : KHSB - XAG6701                                          #*/
/*#                                                                       #*/
/*# Date        : Oct 1998                                                #*/
/*#                                                                       #*/
/*# Author      : Bostjan Vlaovic RDSA7                                   #*/
/*#                                                                       #*/
/*# Translation : in makefile u_t_difftime.mk                             #*/
/*#                                                                       #*/
/*# Remarks     :                                                         #*/
/*#                                                                       #*/
/*# Revisions   : Oct 95                                                  #*/
/*#                                                                       #*/
/*#########################################################################*/
/*************************************************************************/
/*          UNIX - SCCS  VERSION DESCRIPTION                              */
/*************************************************************************/
static char unixid[] = "@(#)KHSB.c 1.1 99/01/29 u_t_difftime.c";

/*****************************************************************/
/*          L I B R A R I E S                                    */
/*****************************************************************/

# ifdef SIM
#  include<string.h>
# endif

# include "u_t_difftime.h"
# include<stdio.h>
# include<time.h>

/*****************************************************************/
/*          SYMBOLIC  CONSTANTS                                  */
/*****************************************************************/

# ifdef SIM
#  ifdef __STDC__
# define ANSI
# else
# define NOANSI
# endif
# endif

/*****************************************************************/
/*              EXTERN FUNCTIONS  DECLARATIONS                   */
/*****************************************************************/
/*
# ifdef RTE_PSOS
        extern int mktime();
extern int difftime();
# endif
*/
/*****************************************************************/
/*          F U N C T I O N S                                    */
/*****************************************************************/

#ifdef  NOANSI
SDL_NATURAL gu_T_DIFFTIME_difftime(time1,time2)
  GU_T_TIME *time1;
```

```
  GU_T_TIME *time2;
#else
SDL_NATURAL gu_T_DIFFTIME_difftime(GU_T_TIME *time1, GU_T_TIME *time2)
#endif
{
        SDL_NATURAL diff;
   int wday;

# ifdef RTE_PSOS
diff=(time2->fd_f_day-time1->fd_f_day)*3600*24+(time2->fd_f_hour-time1->fd_f_hour)*3600+\
(time2->fd_f_min-time1->fd_f_min)*60+(time2->fd_f_sec-time1->fd_f_sec);
#else
diff=0;
# endif
return(diff);
}
```

PRILOGA B: SDL OPIS GENERATORJA KLICEV

IZJAVA

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom g. Simona Molke, dipl. inž., in mentorja izr. prof. dr. Zmaga Brezočnika.

NASLOV ŠTUDENTA

Boštjan Vlaovič
Na zelenici 9
3000 Celje
Tel.: 041/769-386
e-mail: vlaovic@computer.org

KRATEK ŽIVLJENJEPIS

Rojen: 24. januar 1975
Šolanje: 1981-1989 Prva osnovna šola Celje
1989-1993 Srednja tehniška šola Celje
1993-1999 Fakulteta za elektrotehniko, računalništvo in informatiko