Klara Podbreznik

PRIMERJAVA IZBRANIH RAZVOJNIH ORODIJ S PODPORO JEZIKU SDL

Diplomsko delo

Maribor, april 2009

Diplomsko delo univerzitetnega študijskega programa

PRIMERJAVA IZBRANIH RAZVOJNIH ORODIJ S PODPORO JEZIKU SDL

Študent: Klara Podbreznik

Študijski program: Univerzitetni študijski program Telekomunikacije

Mentor: doc. dr. Boštjan VLAOVIČ, telekomunikacije

Somentor: asist. dr. Aleksander VREŽE, telekomunikacije

Maribor, april 2009

UNIVERZA V MARIBORU

_____

(ime fakultete oz. visoke strokovne šole)

Številka: _____

Datum: _____

SKLEP O DIPLOMSKEM DELU

1. _____, študent-ka univerzitetnega (visokošolskega strokovnega) študija _____, izpolnjuje pogoje, zato se mu dovoljuje izdelati diplomsko delo.

2. Tema diplomskega dela je s področja katedre/oddelka/inštituta _____ _, pri predmetu _____.
   Mentor-ica: _____
   Somentor-ica: _____

3. Naslov diplomskega dela: _____ _____

4. Vsebina diplomskega dela: _____ _____ _____ _____ _____ _____

5. Diplomsko delo je potrebno izdelati skladno z »Navodili za izdelavo diplomskega dela« in ga oddati v ____ izvodih do _____ v referatu za študentske zadeve.

Predstojnik katedre/oddelka/inštituta: Dekan-ica:

_____ _____

Mentor:

_____

ZAHVALA

Najprej se zahvaljujem mentorju doc. dr. Boštjanu Vlaoviču, da me je sprejel pod svoje mentorstvo in me uspešno vodil v pravo smer. Zahvaljujem se tudi somentorju, asistentu dr. Aleksandru Vrežetu, za vso pomoč pri diplomi.

Posebna zahvala velja staršem, ki sta mi skozi mojo študijsko pot stala ob strani in mi nudila finančno in moralno podporo.

PRIMERJAVA IZBRANIH RAZVOJNIH ORODIJ S PODPORO JEZIKU SDL

Ključne besede: SDL, Telelogic SDL Suite, ObjectGEODE, Cinderella, Safire Professional, razvojno orodje, telekomunikacijski sistem, specifikacija

UDK: 004.436:621.39(043.2)

Povzetek

V diplomskem delu so predstavljena izbrana razvojna orodja s podporo jeziku SDL (Specification and Description Language) – Telelogic SDL Suite, ObjectGEODE, Cinderella in Safire Professional. Orodja smo preverjali z uporabo poenostavljenega protokola V.76, ki je namenjen vzpostavitvi podatkovne povezave in prenosu podatkov med dvema uporabnikoma. V večini orodij smo uvozili obstoječ opis sistema, le v orodju Cinderella smo sistem zgradili tudi sami. V orodju Safire Professional sistema ni bilo mogoče uvoziti, zaradi prevelikega odstopanja od standarda Z.100, tudi lastna izgradnja sistema ni bila uspešna. Delovanje sistema smo praviloma preverili s simulacijo štirih testnih scenarijev. Ob tem smo predstavili uporabniško izkušnjo ter prednosti in slabosti posameznega orodja s poudarkom na preverjanju skladnosti s standardom Z.100.

COMPARISON OF SELECTED DEVELOPMENT TOOLS WITH SDL LANGUAGE SUPPORT

Key words: SDL, Telelogic SDL Suite, ObjectGEODE, Cinderella, Safire Professional, development tool, telecommunication system, specification

UDK: 004.436:621.39(043.2)

Abstract

In the diploma work there are four development tools for supporting SDL (Specification and Description Language) evaluated and presented - Telelogic SDL Suite, ObjectGEODE, Cinderella and Safire Professional. These tools were evaluated with the use of simplified example protocol for establishing a connection between two modems and data transfer on the established link. In the first three tools the example was imported. The fourth tool – Safire Professional was not standard compliant enough, so neither the import of the example nor the development of the new one was not possible. The correct operation was verified with a simulation of four scenario. The visual appearance, strengths and weaknesses for each application as well as the compliance with standard Z.100 were presented by these scenario.

VSEBINA

KAZALO SLIK

KAZALO TABEL

1. UVOD

Sistemi s sočasnostjo so zelo kompleksne narave. Trenutne razmere na trgu krajšajo razvojne roke in s tem še povečujejo pomembnost metodologije razvoja in s tem povezane izbire razvojnega okolja. Razvoj, predvsem pa vzdrževanje, zahtevata dosti finančnih, tehničnih in človeških virov. Programske komponente veljajo za najdražji in najbolj kritični del sistemov s sočasnostjo in v telekomunikacijah predstavljajo več kot 70% cene sistema [1].

Ob razvoju stremimo k ponovni uporabi komponent sistema in metodologiji, ki omogoča čim bolj enostavno vzdrževanje sistema. Zato je izredno pomembna pravilna izbira razvojnega in podpornega okolja. Izkazalo se je, da za razvoj telekomunikacijskih sistemov, še posebno protokolov, zelo primeren formalni jezik za specifikacijo in opis sistemov SDL (Specification and Description Language), ki je standardiziran v okviru sektorja za telekomunikacije v ITU (International Telecommunications Union) kot standard Z.100 [3]. Podatki v SDL-ju sledijo teoriji abstraktnega podatkovnega tipa, obnašanje sistema pa se poda z razširjenimi končnimi avtomati. Standard formalno določa tekstovno in grafično predstavitev. Slednja je koristna pri zasnovi sistema in študiju delovanja, pri vnosu manjših sprememb v velike projekte, pa se razvijalci pogosto zatečejo k tekstualni obliki.

Ob tehnološkem napredku se neredko zamenja tudi platforma, zato je zaželeno da razvojni sistem omogoča čim bolj enostaven prehod in primerno podporo s strani proizvajalca. Pri gradnji sistema, ki je opisan v jeziku SDL, se praviloma v prvem koraku generira koda v izbranem programskem jeziku. Ta vključuje t. i. SDL stroj in zagotavlja delovanje, ki je skladno s standardom Z.100. Navadno se pri sistemih v telekomunikacijah uporabljajo realno-časovni operacijski sistemi, različne podatkovne baze, ter gonilniki strojne opreme. Ker so nekateri deli sistema razviti neposredno v jeziku C, takšen primer lahko predstavlja razpoznavalnik za protokolni sklad ali gonilnik za uporabljeno strojno opremo, je izredno pomembno, da orodje podpira tudi vključevanje takšne kode. Navadno se za to uporabi mehanizem zunanjih operatorjev ob definiciji abstraktnega podatkovnega

tipa. Zaradi delov, ki niso opisani z jezikom SDL, postane opis sistema pol-formalen. Ocenjujemo, da je za podjetje izbira primernega orodja strateškega pomena.

Na spletu je mogoče najti kar nekaj orodij s podporo jeziku SDL. Namen diplomskega dela je predstaviti nekaj orodij s podporo jeziku SDL, preveriti skladnost s standardom Z.100 in primerjati njihove lastnosti. Orodja se bodo preverjala s pomočjo primera specifikacije protokola V.76, ki je bil objavljen v knjigi Lurenta Doldija [2]. Primer se bo za posamezno orodje po potrebi prilagodil posebnim zahtevam oz. omejitvam orodja. Preverili se bodo podprti formati zapisa, podpora uvozu in izvozu specifikacij v standardnih formatih, podpora različnim operacijskim sistemom, pregled pridruženih orodij za simulacijo in testiranje ter ostale lastnosti, ki prispevajo k uporabnosti orodja za potrebe razvojnega procesa. Izdelali bomo pregledno primerjavo izbranih orodij, na podlagi katere se bo lahko potencialni uporabnik odločil, katero orodje mu ustreza.

1.1 Struktura diplomskega dela

Diplomsko delo je razdeljeno na 5 poglavij. V prvem poglavju so predstavljeni nameni, cilji in teze diplomskega dela. V drugem poglavju je opisan jezik SDL. Podrobnost opisa smo prilagodili potrebam diplomskega dela. Jezik je opisan tako podrobno, da je diplomsko delo razumljivo tudi nepoznavalcem jezika SDL. V tretjem poglavju je predstavljena uporabljena specifikacija protokola V.76. V četrtem poglavju so opisana orodja, ki smo jih našli. Podrobneje so opisana orodja, ki smo jih izbrali za primerjavo, in vtis o delu s posameznim orodjem. V petem poglavju je podan sklep o diplomskem delu.

2. PREDSTAVITEV JEZIKA SDL

2.1 SDL

SDL je standardiziran jezik za specifikacijo in opis sistemov. Pripravil ga je sektor za telekomunikacije v okviru ITU, takrat znan kot CCITT (Comité Consultatif International Téléraphique et Téléphonique). Prva verzija jezika je izšla leta 1976, nato so sledile izboljšave v letih 1980, 1984, 1988, 1992, 1996 in 2000. Da se zagotovi stabilno delovanje sistemov, so vsi zapisi sistemov veljavni tudi v kasnejših verzijah standarda [3]. Standardu so dodelili ime Z.100.

Znano je, da je za uspešen razvoj sistema ključna izdelava natančne specifikacije in modela sistema. To pa zahteva primeren specifikacijski jezik, ki zadošča naslednjim potrebam:

- dobro definiran skupek konceptov,
- jasna, natančna in jedrnata specifikacija,
- temeljita in natančna osnova za analizo specifikacij,
- osnova za določanje, ali implementacija ustreza specifikaciji in
- osnova za določanje skladnosti specifikacije.

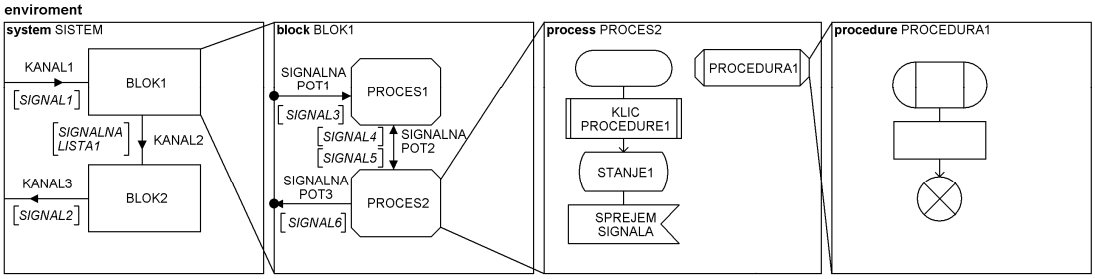
SDL je zasnovan tako, da ustreza vsem naštetim zahtevam [1]. Je formalen, objektno orientiran grafični specifikacijski jezik. Z jezikom lahko natančno opišemo strukturo, obnašanje in podatke sistemov s sočasnostjo, kar odpravi dvoumnosti in zagotavlja integriteto sistema, natančen opis pa nam omogoča formalnost jezika. Vsak simbol ima točno določeno semantiko.

Sistem je sestavljen iz naslednjih komponent:

- struktura sistema – gradniki: `SISTEM, BLOK, PROCES, PROCEDURA`,
- komunikacija – gradniki: `SIGNAL, SIGNALNA POT, KANAL`,
- obnašanje sistema – gradniki: `PROCES, PROCEDURA, STORITEV`,
- podatki – vnaprej definirani podatkovni tipi in abstraktni podatkovni tipi (ADT),
- dedovanje – opis relacij in specializacija.

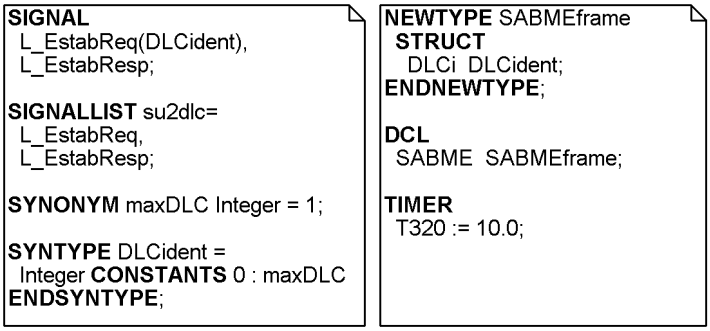
Pri opisu sistema s SDL navadno začnemo z neformalnim opisom na zelo visokem nivoju abstrakcije. Ta specifikacija nudi osnovo za nadaljnji razvoj. Med razvojem se sistem opisuje vedno bolj podrobno. V končni fazi razvoja so formalno opisani vsi deli sistema. Formalno opisane sisteme lahko s posebnimi orodji tudi simuliramo, verificiramo ter avtomatsko generiramo izvršilno kodo za različne ciljne sisteme [4].

Struktura sistema je razdeljena na štiri glavne nivoje (`SISTEM`, `BLOK`, `PROCES`, `PROCEDURA`). Struktura sistema je prikazana na sliki 2.1.



Slika 2.1: Pogled strukture SDL sistema

Najvišji nivo arhitekture predstavlja objekt `SISTEM`. Sistem je grafično predstavljen kot pravokotnik, v tekstovnem zapisu se vse kar sistem vsebuje nahaja med `system` <ime sistema> in `endsystem` <ime sistema>. Vse, kar je zunaj sistema (pravokotnika), je okolje (Slika 2.1). Za SDL je okolje črna škatla, ki spoštuje omejitve sistema. Sistem je lahko zaprt ali odprt. Odprt sistem z okoljem komunicira. Sistemi v telekomunikacijah so odprte narave [4]. Sistem vsebuje tudi deklaracijsko območje, kjer se definirajo novi podatkovni tipi, signali, signalne liste, spremenljivke in ostalo. Deklaracijsko območje je predstavljeno na sliki 2.2. Vsebina deklaracijskega območja je v grafičnem in tekstovnem zapisu enaka.



Slika 2.2: Deklaracijsko območje

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 `PROCESOV`. Blok tako predstavlja primeren način za združevanje procesov v logične skupine kot tudi mejo za vidnost podatkov in podatkovnih tipov [3]. Poznamo pa tudi blokovni tip, ki je grafično predstavljen kot pravokotnik z dvema črtama (slika 3.3), definiramo ga na nivoju sistema ali bloka. Uporablja se, ko imamo več blokov z enako funkcionalnostjo, da definiramo enake bloke na enem mestu. Kličemo ga tako da zraven imena bloka navedemo še ime blokovnega tipa (slika 3.2).

Procesi si znotraj bloka in njegovim okoljem (vse, kar je zunaj bloka, je za blok okolje) izmenjujejo sporočila s pomočjo signalov. Procesi so med seboj povezani s signalnimi potmi (npr. SIGNALNA POT2 na sliki 2.1). Bloki so med seboj povezani s kanali (npr. KANAL2 na sliki 2.1). 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. Signalne poti so vedno brez zakasnitev, kanali pa so lahko zakasnjeni (puščice so na sredini) ali nezakasnjeni (puščice so na koncu črte). V tekstovnem zapisu se kanalu, ki nima zakasnitve, doda za imenom beseda `nodelay`. Kanali in signalne poti prenašajo signale, ti pa so podani v oklepajih poleg grafičnega simbola za kanal oz. signalno pot. Če sta kanal ali signalna pot dvosmerna, so signali našteti za vsako smer. Da se izognemo večkratnemu naštevanju signalov, jih združimo v signalne liste (slika 2.2) in jih uporabimo pri deklaraciji kanalov in signalnih poti. Kanali in signalne poti so povezani preko vrat (`gate`). Vsaka vrata imajo svoje ime in predstavljajo točko povezave.

Signal je najnižji objekt pri komunikaciji in je definiran z imenom. V primeru, da prenaša podatke, so poleg imena v oklepaju definirani tudi podatkovni tipi prenašanih podatkov (slika 2.2). Ko je signal poslan, so parametri zamenjani z dejanskimi vrednostmi. V signalnih listah, kanalih ali signalnih poteh se uporabi samo ime signala, parametri so avtomatsko pridruženi. Signal je lahko definiran na nivoju sistema, bloka ali procesa. Signali, ki so definirani na določenem hierarhičnem nivoju, se lahko uporabljajo tudi na vseh nižjih nivojih. Tako se lahko signal, ki je definiran znotraj procesa, uporablja samo med različnimi primerki 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 [4].

Nižje od nivoja blokov imamo nivo, ki ga sestavljajo procesi. Procesi določajo obnašanje sistema, opisujemo jih s pomočjo razširjenih končnih avtomatov – EFSM (Extended Finite State Machines). 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 [9]. Procesi lahko komunicirajo med seboj, če jih povezujejo komunikacijske poti. 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). Izjema so prioritetni signali, ki jih proces sprejme ne glede na njihov položaj v vhodni vrsti. Ob prehodu med stanji se lahko izvedejo različne akcije. Izvrši se lahko računska operacija, starta časovnik, pošlje signal itd. Proces je lahko ustvarjen ob zagonu sistema ali pa je ustvarjen in zaključen med delovanjem sistema. Obstaja lahko več kot en primerek procesa. Izvajanje procesa se zaključi ob uporabi konstrukta izhod. Vsak proces ima edinstven procesni identifikator (PID - Process Identification Number), ki nedvoumno določa proces. S sklicevanjem na PID je mogoče poslati signale posameznemu primerku procesa. Vrednost podatkovnega tipa PID je lahko konstanta ali predefinirana spremenljivka:

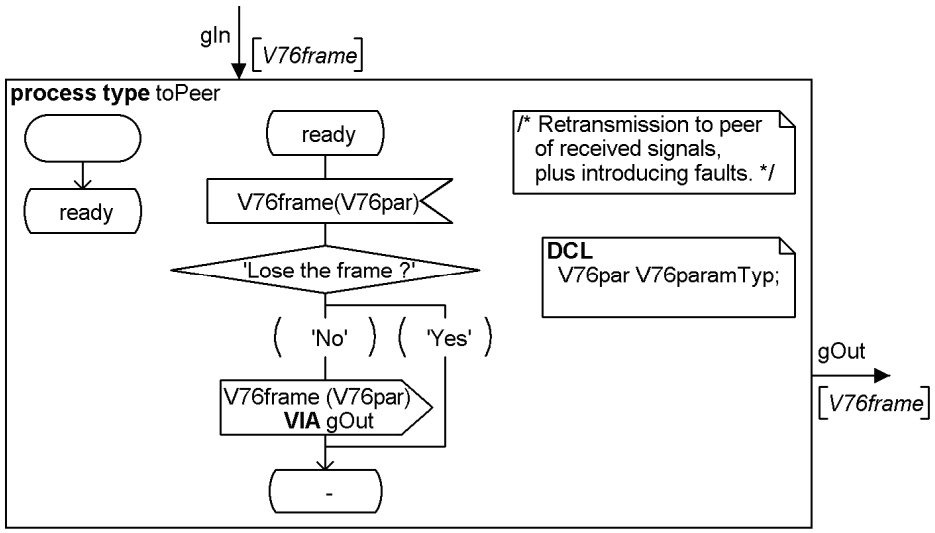
- SELF – lastni PID procesa,
- PARENT – PID procesa, ki je ta proces kreiral,
- SENDER – PID procesa, ki je poslal signal, kateri je sprožil zadnji prehod in
- OFFSPRING – PID zadnje kreiranega procesa [4].

Najnižji nivo sistema sestavljajo `PROCEDURE`. Procedure omogočajo združevanje kompleksnih delov v celoto z enkratno definicijo in večkratno uporabo. Procedura je lahko definirana znotraj procesa ali znotraj druge procedure (procedura1 na sliki 2.1). Definicija procedure lahko vsebuje tudi formalne parametre. Formalni parametri vsebujejo sezname podatkovnih tipov parametrov. Procedura bere vhodne signale iz vhodne vrste procesa, v katerem je definirana. Praviloma je dostopna le v procesu v katerem je definirana. Klic procedure se lahko izvrši med prehodom med dvema stanjema [4].

SDL dopušča dve možnosti opisa podatkov: ASN.1 (Abstract Syntax Notation One) in abstraktne podatkovne tipe (Abstract Data Type, ADT). Slednjega bomo v nalogi uporabili za definicijo novih podatkovnih tipov in operacij med njimi. Standard določa naslednje predefinirane podatkovne tipe:

- INTEGER – decimalna cela števila,
- REAL – realna, decimalna števila brez iracionalnih,
- NATURAL – decimalna naravna števila vključno z ničlo,
- CHARACTER – ASCII znaki,
- CHARSTRING – niz znakov,
- BOOLEAN – logični podatkovni tip,
- PID – identifikacijska številka procesa,
- TIME – absolutni čas sistema,
- DURATION – relativni čas sistema,
- ARRAY – seznam in
- STRUCT – struktura [4].

Za opis sistema v jeziku SDL obstajata dva sintaktično ekvivalentna zapisa: grafični in tekstovni. Na sliki 2.3 vidimo procesni tip `toPeer`, spodaj pa je prikazan njegov tekstovni zapis.



Slika 2.3: Procesni tip `toPeer`

```
PROCESS TYPE toPeer
   COMMENT 'A process type';
  GATE gIn
```

```
    IN WITH V76frame;
  GATE gOut
    OUT WITH V76frame;
  /* Retransmission to peer
     of received signals,
     plus introducing faults. */
  DCL
    V76par V76paramTyp;
  START ;
    NEXTSTATE ready;
  STATE ready;
    INPUT V76frame(V76par);
      DECISION 'Lose the frame ?';
        ( 'No' ):
          OUTPUT V76frame (V76par)
          VIA gOut;
        ( 'Yes' ):
      ENDDECISION;
      NEXTSTATE -;
   ENDSTATE;
ENDPROCESS TYPE;
```

Uporaba jezika SDL pa ni omejena samo na telekomunikacijske sisteme, saj je prisoten tudi pri razvoju medicinske opreme, nadzornih sistemih železnic, avtomobilski in letalski industriji itd. [1]. Slovenska telekomunikacijska industrija temelji na različici SDL-92, zato smo tudi mi v diplomskem delu izbrali to različico.

3. SPECIFIKACIJA PROTOKOLA V.76

Primer, ki smo ga uporabili, je poenostavljena različica protokola, ki ga opisuje ITU standard V.76 in temelji na "Link Access Procedure for Modems" (LAPM). Standard opisuje protokol za vzpostavitev podatkovne povezave (Data Link Connection, DLC) med dvema modemoma in prenos podatkov preko vzpostavljene povezave. Sistem je predviden za dva uporabnika (Service User, SU), A in B, ki komunicirata prek protokola V.76. Hkrati lahko med uporabnikoma obstaja več povezav, uporabnik A lahko uporablja eno povezavo (DLC 0) za prenos zvoka do ali od uporabnika B in med tem, ko je ta povezava aktivna, lahko vzpostavi še eno povezavo (DLC 1) za prenos podatkov do ali od uporabnika B [2].

3.1 Specifikacija protokola V.76

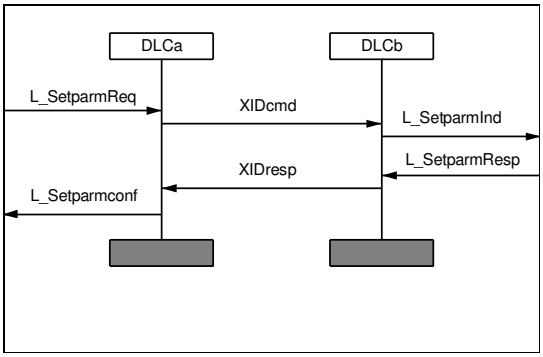
Komunikacija med uporabnikoma ponavadi poteka tako, da uporabnik pošlje zahtevo ("request") nivoju, ki je zadolžen za komunikacijo, ta pošlje uporabniku na ponorni strani naznanitev ("indication"), če višji sloj sprejme zahtevo odgovori z odgovorom ("response"), ki je nato preveden v potrdilo ("confirm") na strani pošiljatelja zahteve. Vsi primitivi, ki se uporabljajo za komunikacijo so predstavljeni v tabeli 1.

Tabela 1: Primitivi za komunikacijo med uporabnikom in protokolom, ki je zadolžen za komunikacijo [10].

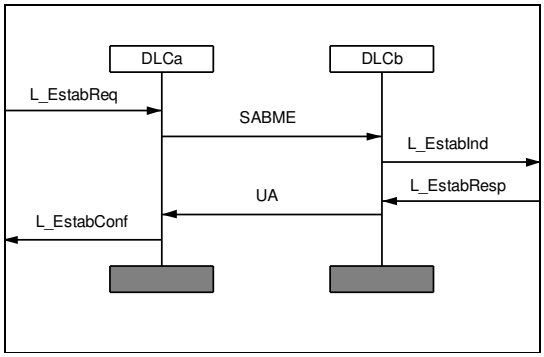
Storitev	Primitiv	Tip
Vzpostavitev podatkovne povezave med uporabnikoma	L_Establish	request, indication, response, confirm
Prenos podatkov v ERM (Error Recovery Mode)	L_Data	request, indication
Prenos podatkov v UNERM (Unacknowledged Non-Error Recovery Mode)	L_UnitData	request, indication
Sprostitev podatkovne povezave	L_Release	request, indication

Prenos kontrolne informacije	L_Signal	request, indication, response, confirm
Nastavitev vrednosti parametrov	L_SetParam	request, indication, response, confirm
Povratna povezava	L_Test	request, indication

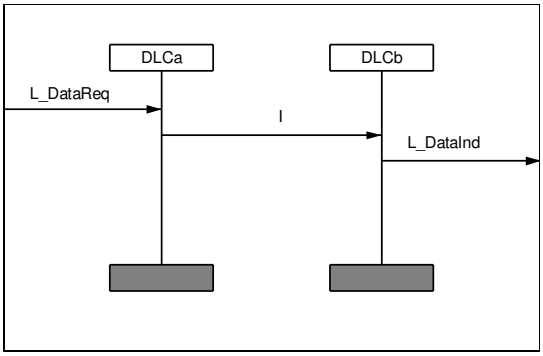
Slika 3.1 prikazuje štiri diagrame MSC (Message Sequence Charts) 0, za različne faze poteka komunikacije med uporabnikoma.



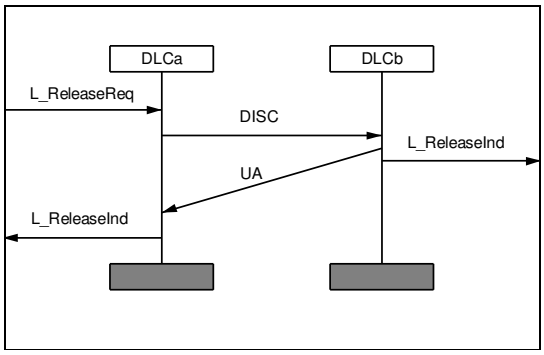
Potrditev prisotnosti



Vzpostavitev povezave



Prenos informacijskega okvirja



Prekinitev povezave

Slika 3.1: Diagrami MSC

Po prejetju signala `L_SetparamReq` (primitiv `L_SetParam`) od uporabnika, primerek procesa (`DLCa`) pošlje ukaz `XIDcmd` (eXchange IDentification command frame) primerku procesa `DLC` na sprejemni strani (`DLCb`). `DLCb` po prejetju `XIDcmd` pošlje

signal L_SetparamInd uporabniku, ki mora odgovoriti s signalom L_SetparamResp. DLCb nato pošlje ukaz XIDresp primerku procesa DLCa, ki potem obvesti uporabnika na svoji strani s signalom L_SetparamConf, s tem je potrditev prisotnosti zaključena.

Drugi diagram prikazuje vzpostavitev povezave. Po prejemu signala L_EstabReq (primitiv L_Establish) od uporabnika na strani A poskuša V.76 vzpostaviti novo povezavo. Proces DLCa najprej resetira števec ponovnih oddaj, nato pošlje ukaz SABME (Set Asynchronous Balanced Mode Extended) in nastavi časovnik T320. Proces DLCb na sprejemni strani po prejemu ukaza SABME preveri, ali lahko uporabnik na sprejemni strani sprejme novo povezavo (L_EstabInd). Po prejemu pozitivnega odgovora (L_EstabResp) se tvori nova povezava DLC, proces DLCb pa mora:

- odgovoriti z neoštevilčeno potrditvijo - UA (Unnumbered Acknowledge) in
- smatrati, da je DLC povezan, in vstopiti v povezano stanje.

Po prejemu UA mora pošiljatelj ukaza SABME ustaviti časovnik T320 in smatrati povezavo DLC za vzpostavljeno, uporabnika pa obvesti s signalom L_EstabConf.

Če uporabnik ne more sprejeti povezave, mora DLCb odgovoriti na ukaz SABME z ukazom DM (Disconnect Mode).

Po prejetju odgovora DM mora pošiljatelj ukaza SABME obvestiti uporabnika, da vzpostavitev DLC ni uspela.

Če časovnik T320 poteče pred sprejemom UA ali DM, mora DLC ponovno zagnati časovnik, ponovno poslati ukaz SABME in povečati števec ponovnih oddaj. Število ponovnih oddaj določa števec N320, ki ima vrednost nastavljeno na 3. Če se ponovna oddaja ukaza SABME zgodi N320-krat in DLC še vedno ne dobi odgovora, mora obvestiti uporabnika, s signalom L_ReleaseInd, da vzpostavitev povezave ni uspela.

Ko je povezava vzpostavljena, se lahko prične prenos informacijskih okvirjev. Proces DLCa prejme od uporabnika signal L_DataReq (primitiv L_Data), skladno s pravili V.76 se vsebina posreduje prejemniku z ukazom I, ko pa proces DLCb na drugi strani sprejme ukaz I, mora poslati informacijski okvir uporabniku na sprejemi strani s pomočjo signala L_DataInd.

Prekinitev povezave lahko zahteva katerikoli uporabnik s signalom L_ReleaseReq. Po sprejemu zahteve po prekinitvi proces DLC na strani pošiljatelja zahteve pošlje ukaz

`DISC`, `DLC`-ju na drugi strani povezave. Po sprejemu ukaza `DISC`, `DLC` odgovori z ukazom `UA` in uporabniku na svoji strani pošlje signal `L_ReleaseInd` ter se ubije. Ko pošiljatelj ukaza `DISC` sprejme odgovor `UA` ali `DM`, kar pomeni, da je drugi `DLC` že prekinil povezavo, še sam prekine povezavo in o tem obvesti uporabnika na svoji strani. Primer je podrobneje predstavljen v [2].

4. ORODJA

4.1 Telelogic SDL Suite

4.1.1 Uvod

Telelogic SDL Suite je orodje podjetja Telelogic, ki je bilo ustanovljeno leta 1983, s sedežema v Malmö na Švedskem in v Irvine v Kaliforniji. Podjetje ima 22 podružnic po vsem svetu in je vodilno podjetje na področju grafičnih programskih orodij za opisovanje telekomunikacijskih sistemov [8]. Leta 2008 ga je kupilo podjetje IBM.

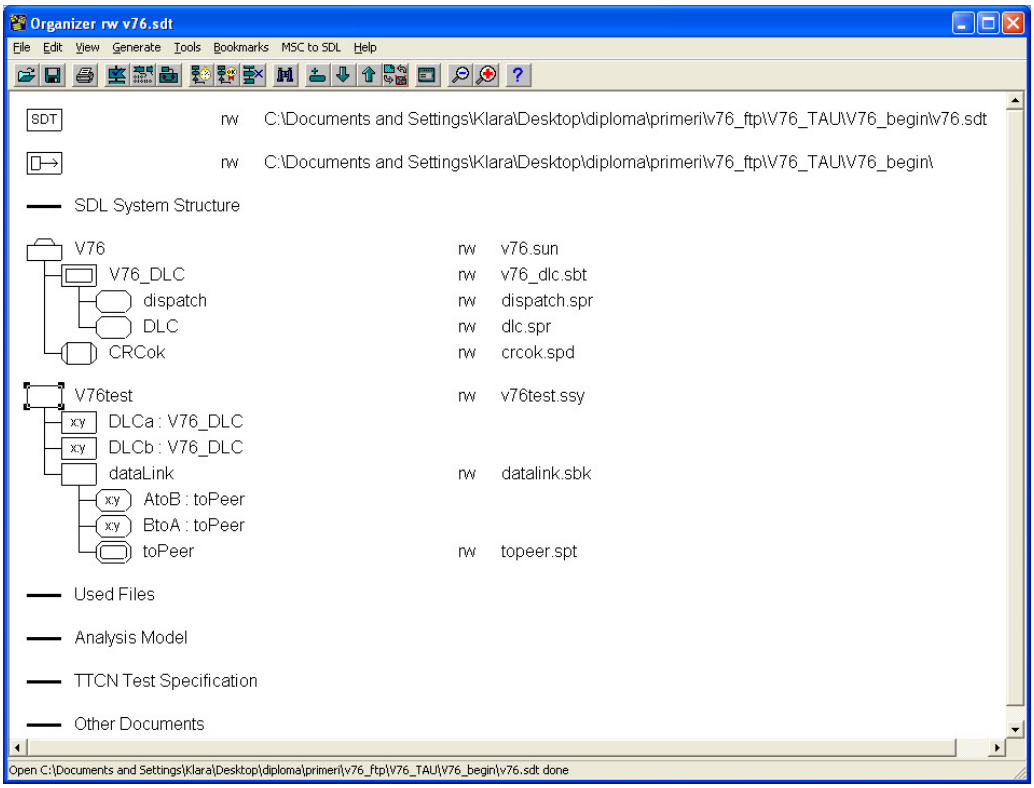
SDL Suite 4.6 je namenjen gradnji in specifikaciji sistemov s sočasnostjo. Orodje podpira jezike SDL, MSC in UML. SDL Suite je najbolj primeren za telekomunikacijsko industrijo, ker je veliko standardov za komunikacijske protokole napisanih v SDL, uporablja pa se tudi v avtomobilski industriji in pri razvoju medicinskih naprav. Grafična sintaksa programa je dovolj preprosta, da sta struktura in obnašanje sistema razumljiva tudi tistim, ki niso razvijalci, hkrati pa dovolj podrobna, da je mogoče opisati vse podrobnosti sistema.

SDL Suite shrani projekt v datoteko s končnico .sdt, poleg tega orodje shrani sistem (.ssy), vsak blok (.sbk) in vsak proces (.spr) v svojo datoteko, ki imajo različne končnice in jih je možno tudi odpreti vsako zase. Če pa želimo .pr zapis, pa moramo izbrati "Convert to PR", kar nam ustvari eno PR (Phrase Representation) datoteko. To datoteko je nato možno pretvoriti nazaj v GR (Graphical Representation) in jo uvoziti s pomočjo "Import SDL". Orodje omogoča pretvorbo med formati GR in CIF (Common Interchange Format), kjer pa povratna pretvorba iz CIF v GR ne deluje. Če želimo urejati kodo v tekstualnem načinu, moramo v izbranem urejevalniku urejati PR datoteko in jo nato uvoziti nazaj v program, pri čemer ohranimo funkcionalnost, grafična podoba pa se spremeni [12].

4.1.2 Grafična podoba

Orodje SDL Suite je sestavljeno iz več oken. Na primer, osnovno okno je "Organizer" (Slika 4.1), kjer je vidna hierarhija sistema in kjer kličemo druge funkcije. Če želimo zraven še urejati kakšen proces, imamo odprto tudi okno "SDL Editor" (Slika 4.2), kjer grafično sestavljamo sistem, bloke, procese itd. s pripadajočima oknoma za simbole in za

tekst. Če želimo sistem analizirati, nam morebitne napake in opozorila izpiše v oknu "Organizer Log". Veliko število oken deluje malce nepregledno.

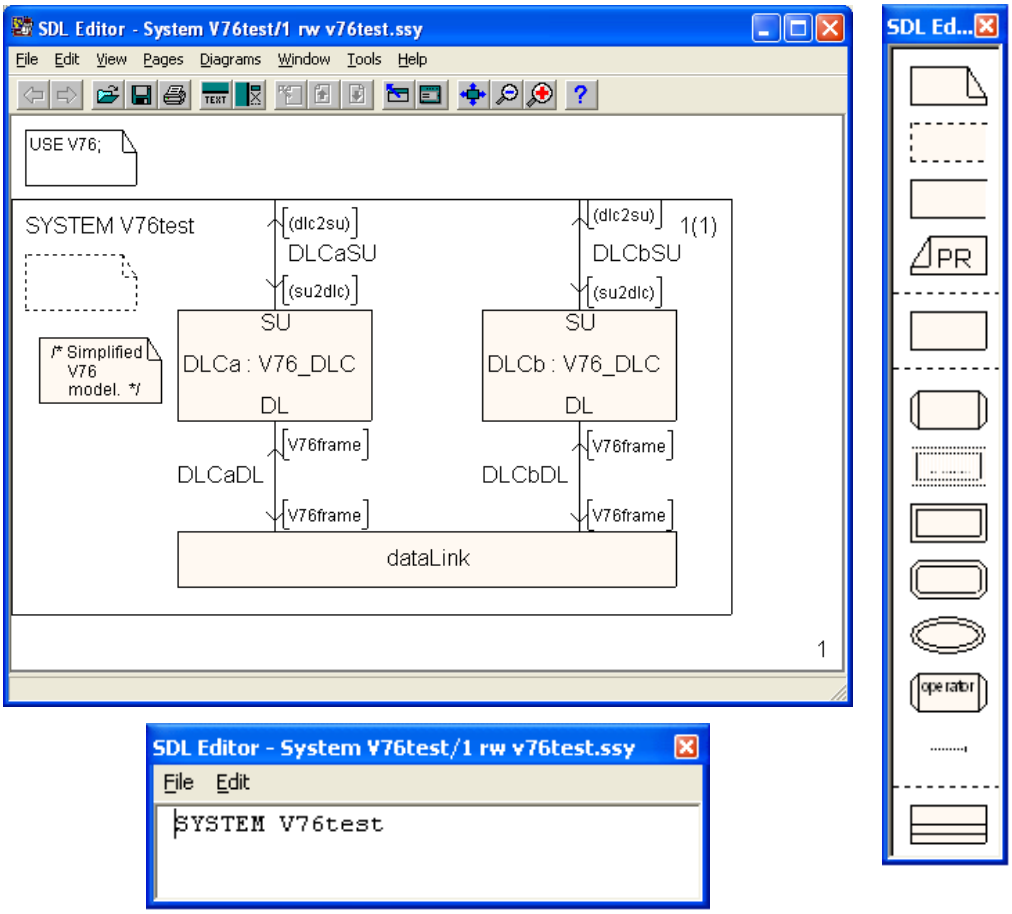


Slika 4.1: Pogled okna "Organizer" orodja Telelogic SDL Suite

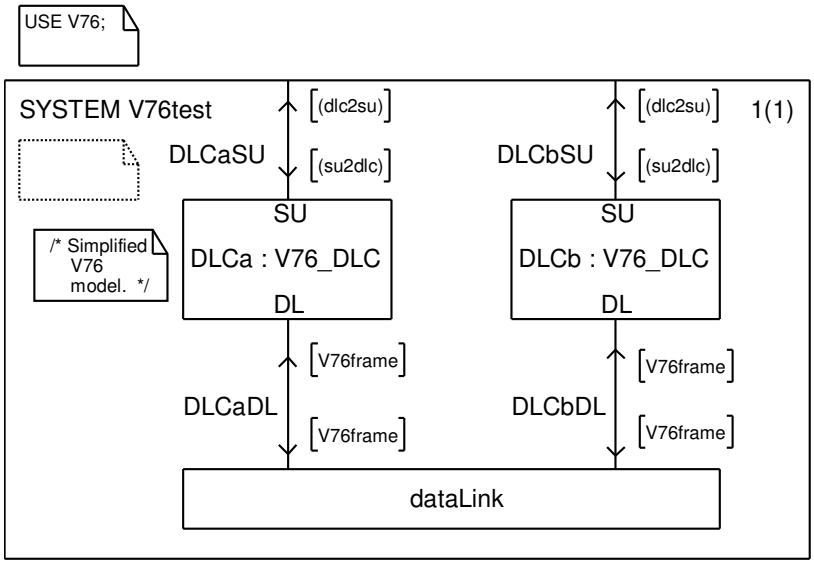
4.1.3 SDL opis protokola V.76

Opis protokola je povzet po knjigi Laurenta Doldija "Validation of Communications Systems with SDL" [2]. Opisan primer je dostopen na naslovu: ftp://ftp.wiley.co.uk/pub/books/ldoldi/.

Slika 4.3 prikazuje grafično specifikacijo testnega sistema. Sistem `V76test` vsebuje dva primera protokola V.76, `DLCa` in `DLCb`, ki sta blokovna tipa `V76_DLC` (slika 4.6). Storitev podatkovne povezave jima zagotavlja blok `dataLink`. Blok `dataLink` predstavlja poenostavljen sloj podatkovne povezave, prikazan je na sliki 4.11. Bloka `DLCa` in `DLCb` komunicirata z uporabniki preko kanalov `DLCaSU` in `DLCbSU`. Kanaloma sta pridruženi signalni listi `su2dlc` in `dlc2su`, ki določata katere signale uporabnik pošilja oz. sprejema (slika 4.4).

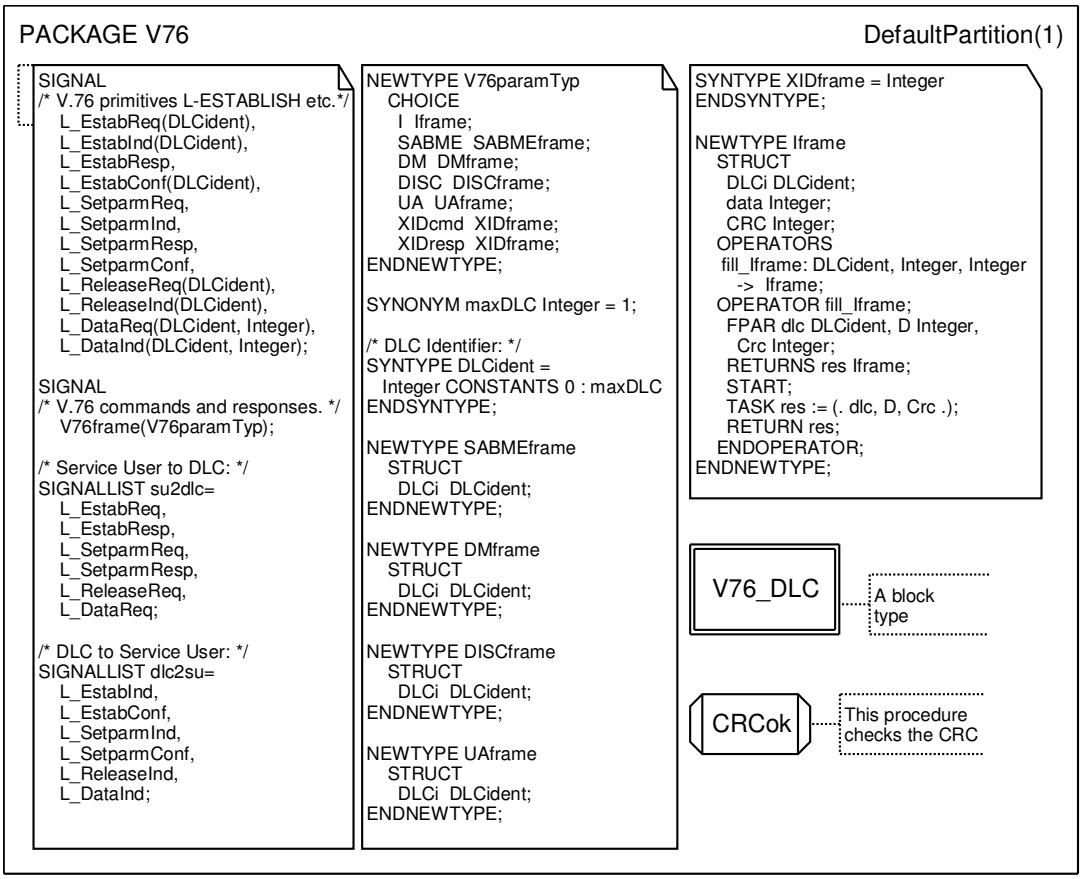


Slika 4.2: Pogled okna “SDL Editor” in pripadajočih oken orodja

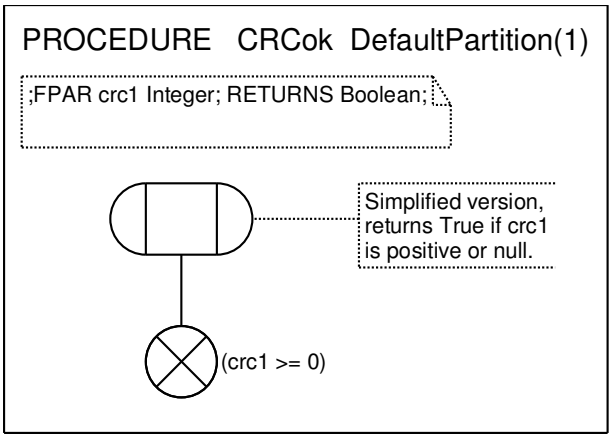


Slika 4.3: Grafični opis specifikacije sistema `V76test`

Blokovni tip `V76_DLC` in definicije njegovih signalov so vsebovane v paketu `V76` (slika 4.4) in uvožene v sistem `V76test`. Paket `V76` vsebuje tudi deklaracije podatkovnih tipov, ki so uporabljeni kot signalni parametri, in proceduro `CRCok` (slika 4.5).



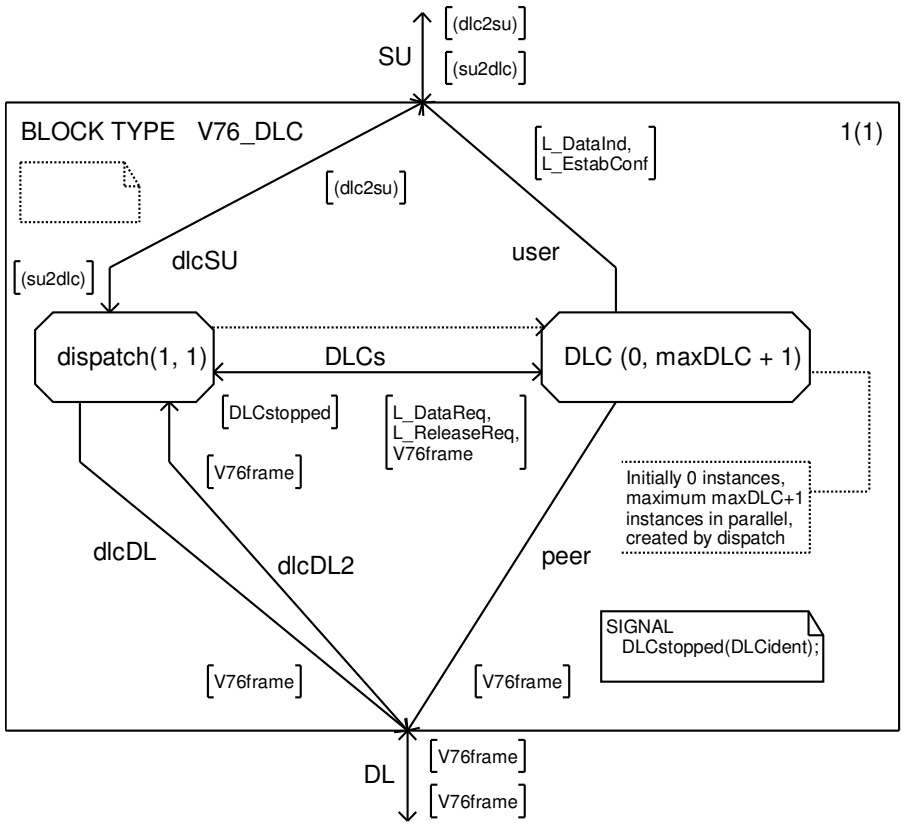
Slika 4.4: Grafični opis specifikacije paketa `V76`



Slika 4.5: Grafični opis specifikacije procedure `CRCok`

Proceduro se kliče kadar je sprejet `I` (information) okvir, da se preveri pravilnost podatkov.

Blokovni tip `V76_DLC` je predstavljen na sliki 4.6. Vsebuje dva procesa: `dispatch`, ki je kreiran že ob zagonu sistema in ima maksimalno en primerek, in proces `DLC`, ki na začetku nima nobenega primerka, maksimalno število primerkov pa je `maxDLC+1`. `MaxDLC` je sinonim, definiran v paketu `V76` in je enak 1 (slika 4.4).

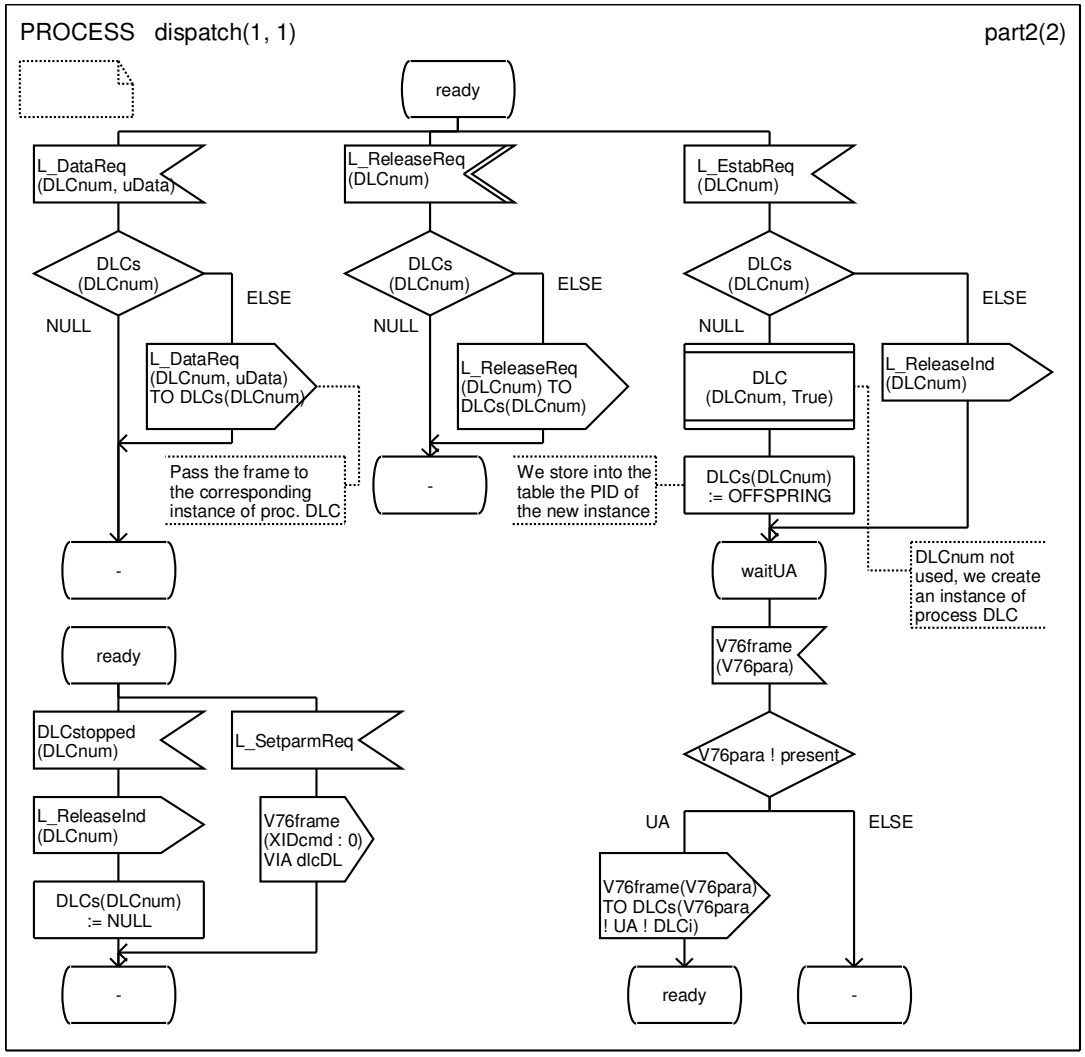


Slika 4.6: Grafični opis specifikacije blokovnega tipa `V76_DLC`

Črtkana črta na sliki 4.6 od procesa `dispatch` proti procesu `DLC` pomeni, da je `dispatch` tisti proces, ki lahko kreira primerke procesa `DLC`. Proces `DLC` se tvori za vsako vzpostavljeno podatkovno povezavo.

Proces `DLC` sprejema signale samo preko procesa `dispatch`. Proces `dispatch` pošilja prejete signale naprej ustreznemu primerku procesa `DLC`, s pomočjo parametra številka `DLC`. `DLCident` je `DLC` številka, ki označuje tok podatkov na nižjem nivoju

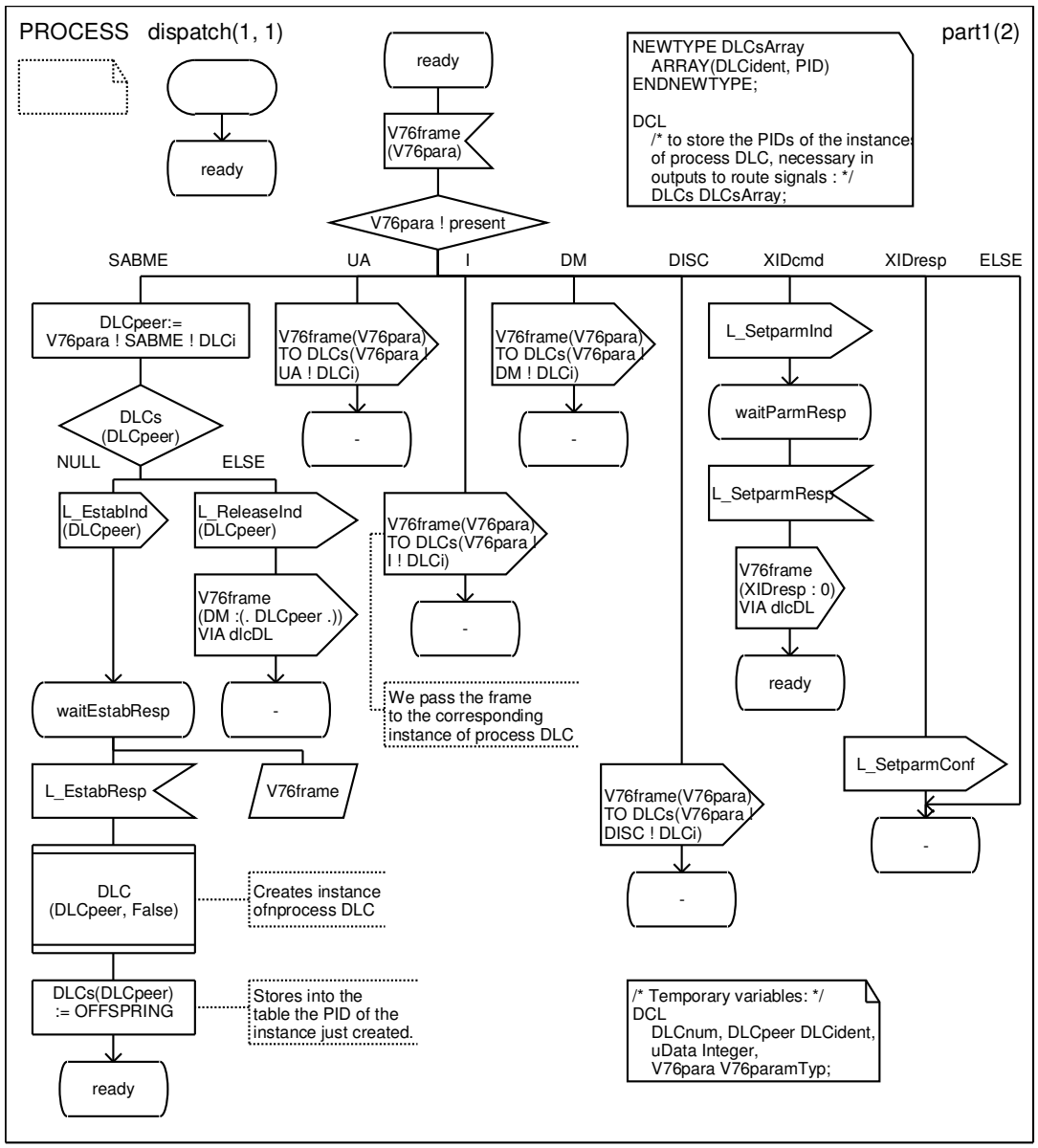
DLC. DLCident je SYNTYPE tipa celo število in je med 0 in maxDLC, ker je maxDLC enak 1, lahko ima DLCident vrednosti 0 in 1. Ta parameter je bil dodan signalom L_EstabReq, L_EstabInd, L_ReleaseReq, L_ReleaseInd, L_DataReq in L_DataInd.



Slika 4.7: Grafični opis specifikacije drugega dela procesa dispatch

Proces dispatch, prikazan na sliki 4.7, po sprejemu signala L_EstabReq, preveri ali je številka DLC (DLCnum) prosta in ustvari nov primerek procesa DLC s parametri DLCnum in True. Pid novo ustvarjenega primerka procesa se shrani v polje DLCs na indeks DLCnum. Polje je deklarirano in prikazano na sliki 4.8. To polje rabimo, da lahko potem s pomočjo številke DLC (DLCnum) dobimo Pid procesa, ki mu moramo

posredovati signal. Po kreiranju novega procesa se proces `dispatch` postavi v stanje `waitUA`. Če v tem stanju sprejmemo signal `V76frame`, ki vsebuje `UA`, ga pošljemo ustreznemu primerku procesa `DLC`, ki ga določimo s pomočjo polja `DLCs` (slika 4.7).

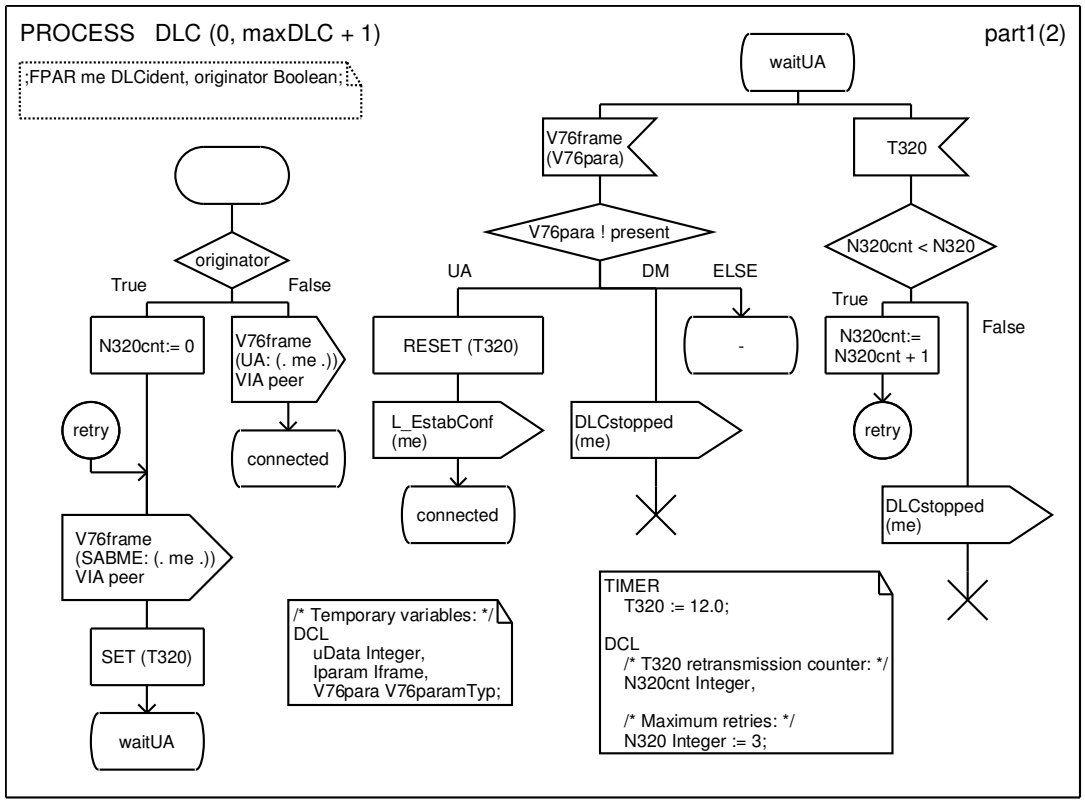


Slika 4.8: Grafični opis specifikacije prvega dela procesa `dispatch`

Slika 4.8 nam kaže prvi del procesa `dispatch`, ko sprejmemo signal `V76frame` in uporabimo simbol za odločitev, da razberemo kateri element je prisoten v sprejetem parametru `V76para`. Če je polje `present` enako `SABME` testiramo številko DLC (ki smo

jo prejeli v `V76para`) ali ustreza prostemu DLC v polju `DLCs`. `Null` v polju pomeni da ne obstaja noben primerek procesa DLC, če je tako pošljemo signal `L_EstabInd` uporabniku in počakamo na odgovor. Po prejetju odgovora `L_EstabResp`, ustvarimo nov primerek procesa s parametri `DLCpeer` (njegova DLC številka) in `False` (to nam pove, da ta stran povezave ni poslala zahteve za povezavo `L_EstabReq`). Nato se shrani `Pid` kreiranega procesa v polje `DLCs`, proces `dispatch` pa se postavi v stanje `ready`.

Sliki 4.9 in 4.10 prikazujeta vsebino procesa `DLC`, dinamično kreiranega s procesom `dispatch`. Proces `DLC` na začetku preveri kakšno vrednost ima parameter `originator`. V primeru da ima vrednost `True` pošlje signal `SABME` (preko povezave `peer` slika 4.6), v primeru `False` pošlje `UA` (prav tako preko `peer`) in se postavi v stanje `connected` (povezava je vzpostavljena). Po oddaji `SABME`, se postavi v stanje `waitUA` in po sprejemu potrditve `UA` odda signal `L_EstabConf`, povezava je tako vzpostavljena.

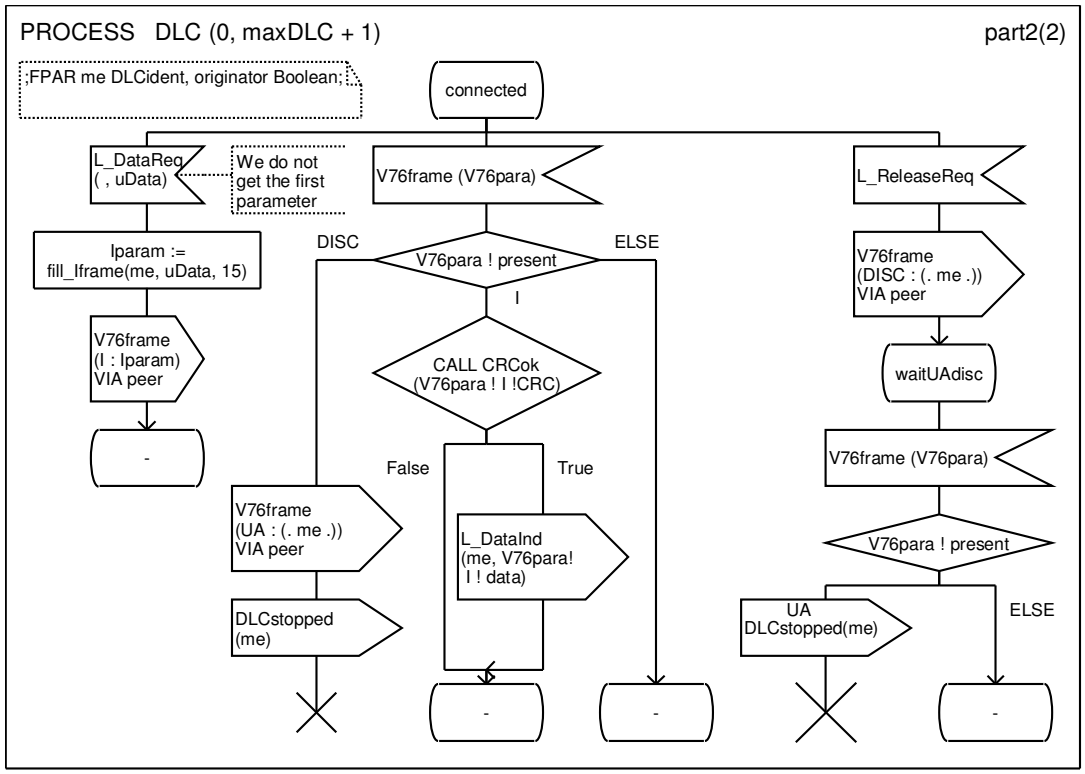


Slika 4.9: Grafični opis specifikacije prvega dela procesa `DLC`

Proces `DLC` lahko v stanju `connected` (slika 4.10) sprejme signale:

- `L_DataReq` – za prenos podatkov preko `I` okvirjev,
- `L_ReleaseReq` – zahteva za prekinitev povezave, konča se s simbolom stop in
- `V76frame`, ki lahko vsebuje `DICS` ali `I` okvir.

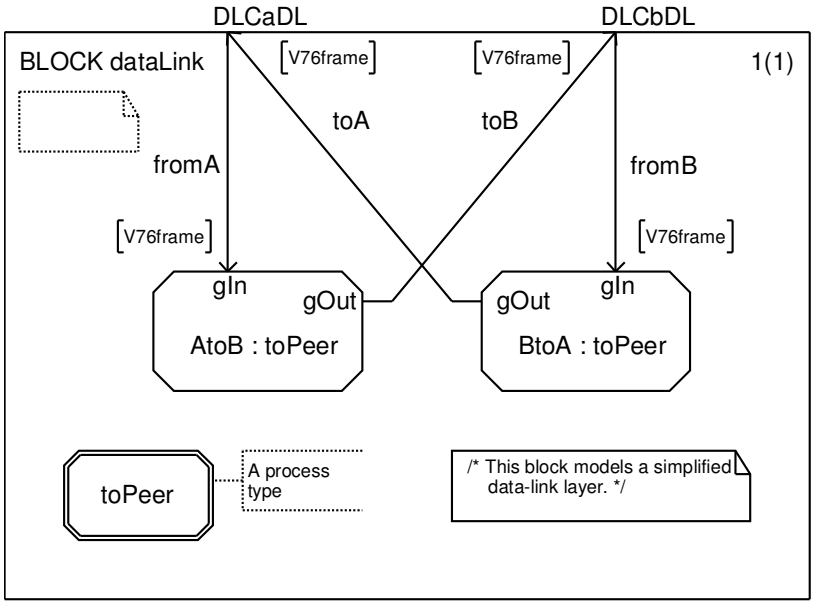
Ko je sprejet ukaz `DISC`, se pošlje signal `DLCstopped` procesu `dispatch`, ki posodobi polje `DLCs`.



Slika 4.10: Grafični opis specifikacije drugega dela procesa `DLC`

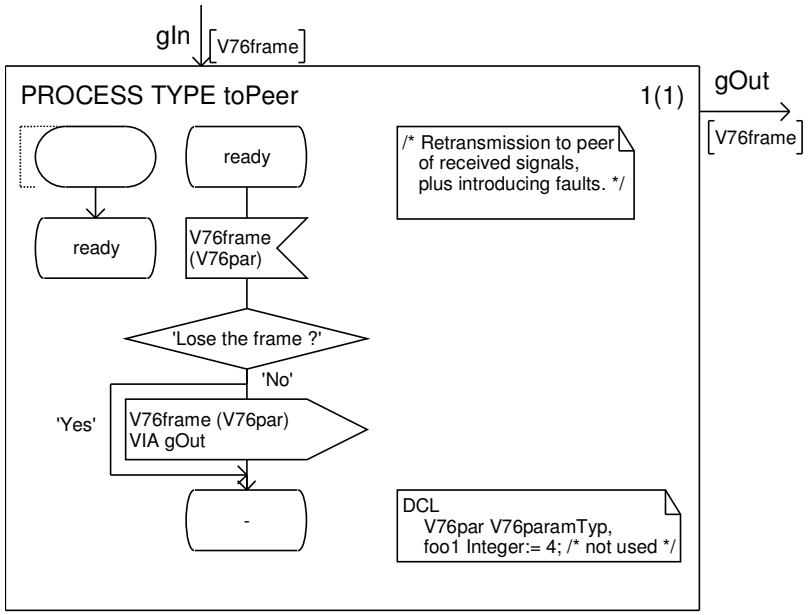
Slika 4.11 prikazuje vsebino bloka `dataLink`, katerega naloga je prenesti (ali pa izgubiti) okvirje `V76frame` od ene `DLC` do enakovredne `DLC` na drugi strani. Za to funkcijo bi zadostoval en proces, vendar v SDL ne moremo pošiljati prejetih signalov vratom, če sta obe strani identični, ker bi imeli težave z naslavljanjem in bi bilo potrebno shranjevati PID-e. Enostavna rešitev je, da uporabimo en proces za vsako smer prenosa:

- `AtoB` prejema signale od `DLCa` in jih pošilja naprej `DLCb` in
- `BtoA` prejema signale od `DLCb` in jih pošilja naprej `DLCa`.



Slika 4.11: Grafični opis specifikacije bloka `dataLink`

Procesni tip `toPeer`, slika 4.12, smo dodali, da smo lahko generirali napake za testiranje robustnosti sistema. Ko simulator pride do takšnega simbola, vpraša uporabnika za odgovor, če uporabnik izbere `'No'` je signal `V76frame` prenesen, drugače je izgubljen.



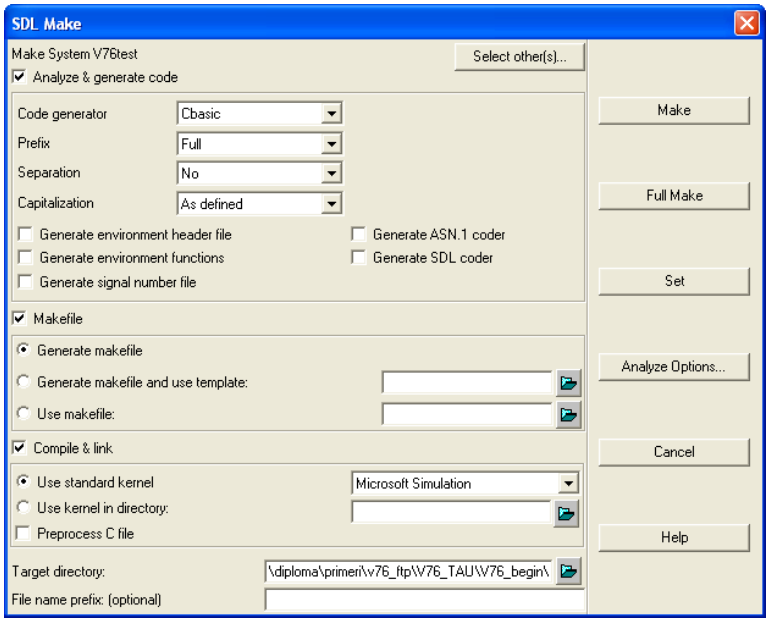
Slika 4.12: Grafični opis specifikacije procesnega tipa `toPeer`

4.1.4 Simulacija

Telelogic SDL Suite ima orodje za simulacijo razdeljeno na:

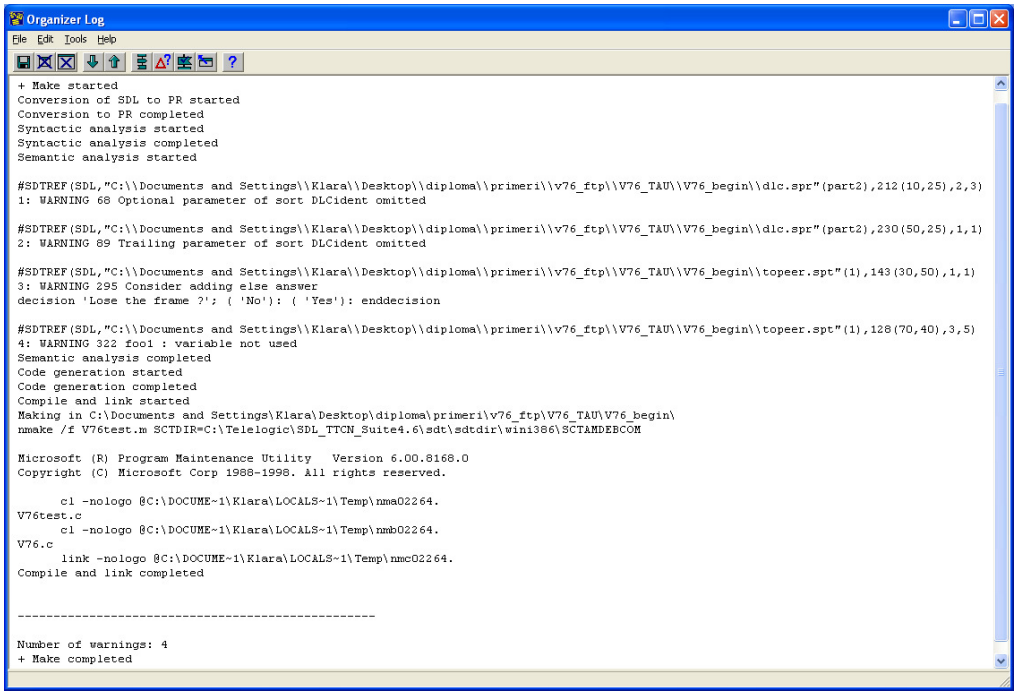
- "Simulator": interaktivna simulacija in avtomatska simulacija - kadar sta pripravljena dva prehoda hkrati, se bo izvedel tisti, ki je bil pripravljen prvi.
- "Validator": interaktivna, naključna in obširna simulacija - kadar sta pripravljena dva prehoda hkrati se lahko izvede katerikoli od njiju, izbira je odvisna od načina simulacije. Pri interaktivni uporabnik določi kateri prehod se izbere, pri naključni je prehod odvisen od naključne številke, pri obširni pa imamo na voljo obe možnosti.

V nadaljevanju je opisan postopek simulacije štirih scenarijev, ki so prikazani na sliki 3.1 (preverjanje prisotnosti, vzpostavitev povezave, prenos podatkov in prekinitev povezave). Uporabljeni sistem [11] odpremo v oknu "Organizer" (slika 4.1). Izberemo sistem `V76test` in "Generate > Make", pojavi se okno "SDL Make" (slika 4.13)



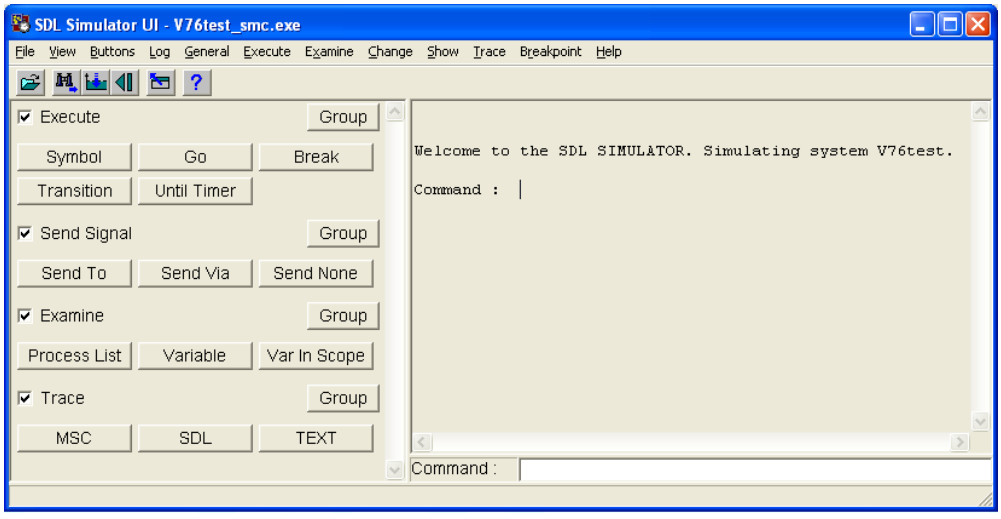
Slika 4.13: Okno "SDL Make"

V oknu "SDL Make" izberemo "Microsoft Simulation" in pritisnemo gumb "Full Make". Pojavi se nam okno "Organizer Log" (slika 4.14). Če je naš SDL model pravilen SDL Suite zgenerira kodo C, nato prevede C datoteke in jih poveže s simulatorjevo knjižnico in tako ustvari izvršljivo datoteko "v76test_smc.exe".



Slika 4.14: Okno “Organizer Log” po ukazu “Full Make”

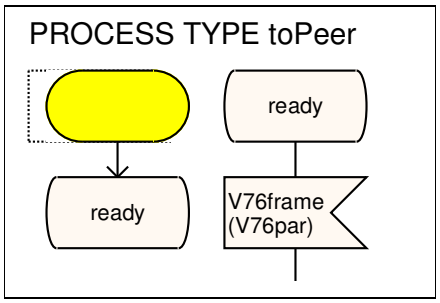
Sedaj lahko začnemo simulacijo z izbiro ikone “Simulate” v oknu “Organizer” (slika 4.15).



Slika 4.15: Okno “Simulator”

Na začetku so omogočeni štirje prehodi, ker model V.76 vsebuje štiri primerke procesa: `AtoB:1`, `BtoA:1`, `<<Block DLCa>> dispatch` in `<<Block DLCb>> dispatch`. Da lahko začnemo pošiljati signale je potrebno vse štiri procese postaviti v

stanje “ready”. Ob izbiri “SDL” v skupini “Trace” (slika 4.15) nam bo simulator ob izbiri gumba “Transition” (skupina “Execute”), pokazal simbol za izvršitev (slika 4.16).

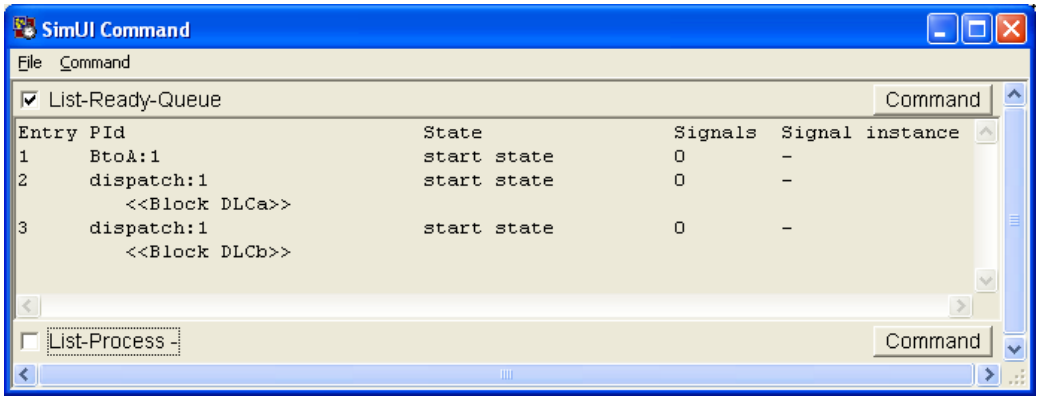


Slika 4.16: Označen simbol za izvršitev

Po izvedenem začetnem prehodu v prvem primerku procesa `AtoB` iz stanja `start` v stanje `ready`, se nam v desnem delu okna “Simulator” izpiše:

```
*** TRANSITION START
*       PId     : AtoB:1
*       State   : start state
*       Now     : 0.0000
*** NEXTSTATE  ready
```

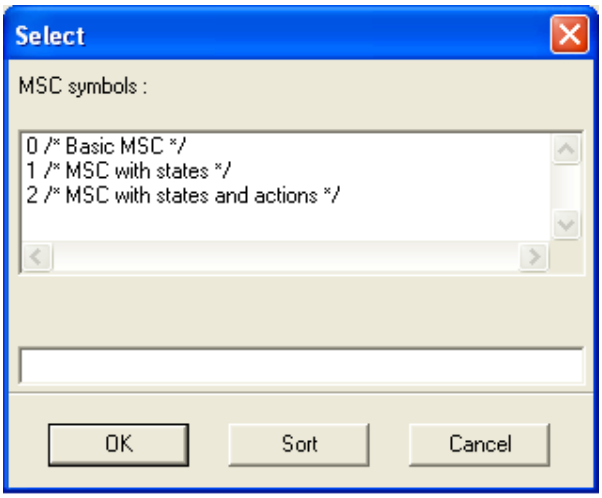
Če želimo imeti pregled nad procesi, ki čakajo na izvršitev, v oknu “Simulator” izberemo “View > Command Window”. V oknu z ukazi vidimo tri primerke procesa (`BtoA`, `dispatch` v bloku `DLCa` in `dispatch` v bloku `DLCb`), ki so v stanju `start` in čakajo na prehod (slika 4.17).



Slika 4.17: Okno “Command”

Pred nadaljevanjem simulacije moramo tudi preostale procese postaviti v stanje `ready`. V oknu "Command" vidimo da je čakalna vrsta sedaj prazna, SDL model sedaj pričakuje zunanje signale.

Ker želimo MSC diagrame primerjati z diagrami iz slike 3.1, pripravimo simulator za sledenje izvajanja z uporabo diagramov MSC. V simulator vpišemo ukaz: "Define-MSC-Trace-Channels on" (ali krajše "d-m-t-c on"), sedaj simulator namesto enega primerka okolja generira en primerek okolja za vsak zunanji SDL kanal (`DLCaSU` in `DLCbSU`). V oknu "Simulator" izberemo "Trace > MSC Trace: Start", odpre se okno "Select" (slika 4.18), kjer izberemo "Basic MSC". Sedaj je simulator pripravljen preveriti skladnost sistema z diagrami MSC, ki so predstavljeni na sliki 3.1.

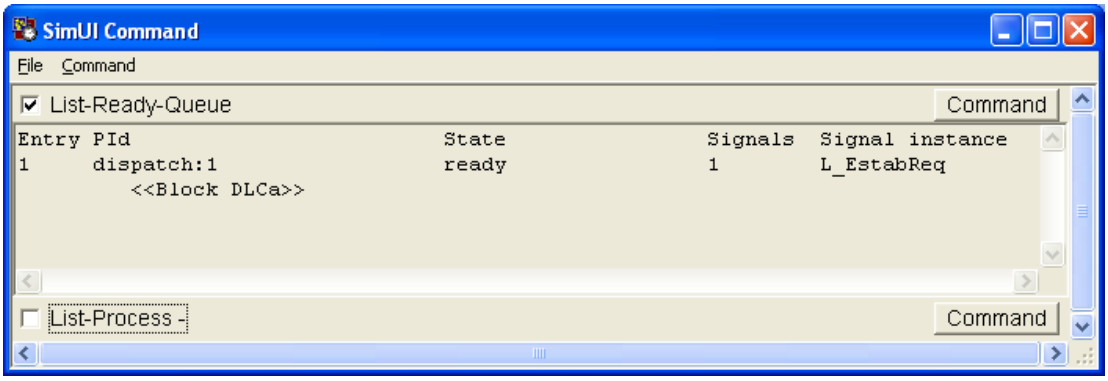


Slika 4.18: Okno "Select"

Prvi scenarij, ki ga bomo preverili, je vzpostavitev povezave. V oknu "Simulator" izberemo gumb "Send To" in iz liste signalov, ki so na voljo izberemo signal `L_EstabReq`. Odpre se novo okno v katerem parametru `DLCident` signala `L_EstabReq` določimo vrednost 0. Nato izberemo ciljni proces `<<Block DLCa>> dispatch`. V oknu "Command" opazimo, da čakalna vrsta sedaj vsebuje proces `dispatch` bloka `DLCa`, ki čaka na sprejem signala `L_EstabReq` (slika 4.19).

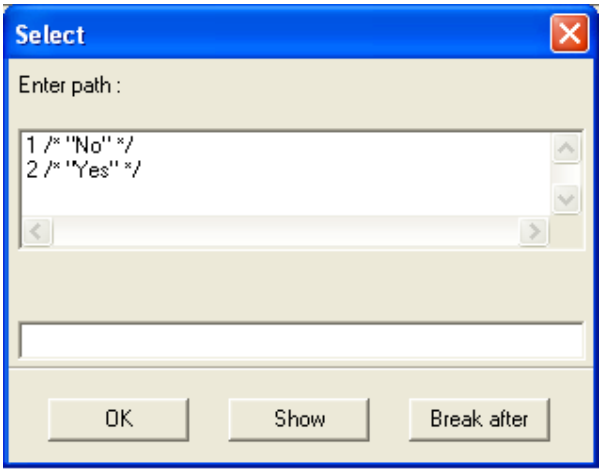
Prehode izvršujemo z izbiro gumba "Transition" v skupini "Execute". Ob sprejemu signala se odpre okno "Select", v katerem lahko izberemo dve možnosti. Z izbiro `1`

`/*˝No˝*/` bo signal `v76frame` prenesen, z izbiro `2 /*˝Yes˝*/` se signal izgubi (slika 4.20).



Slika 4.19: Okno "Command" s procesom `dispatch`

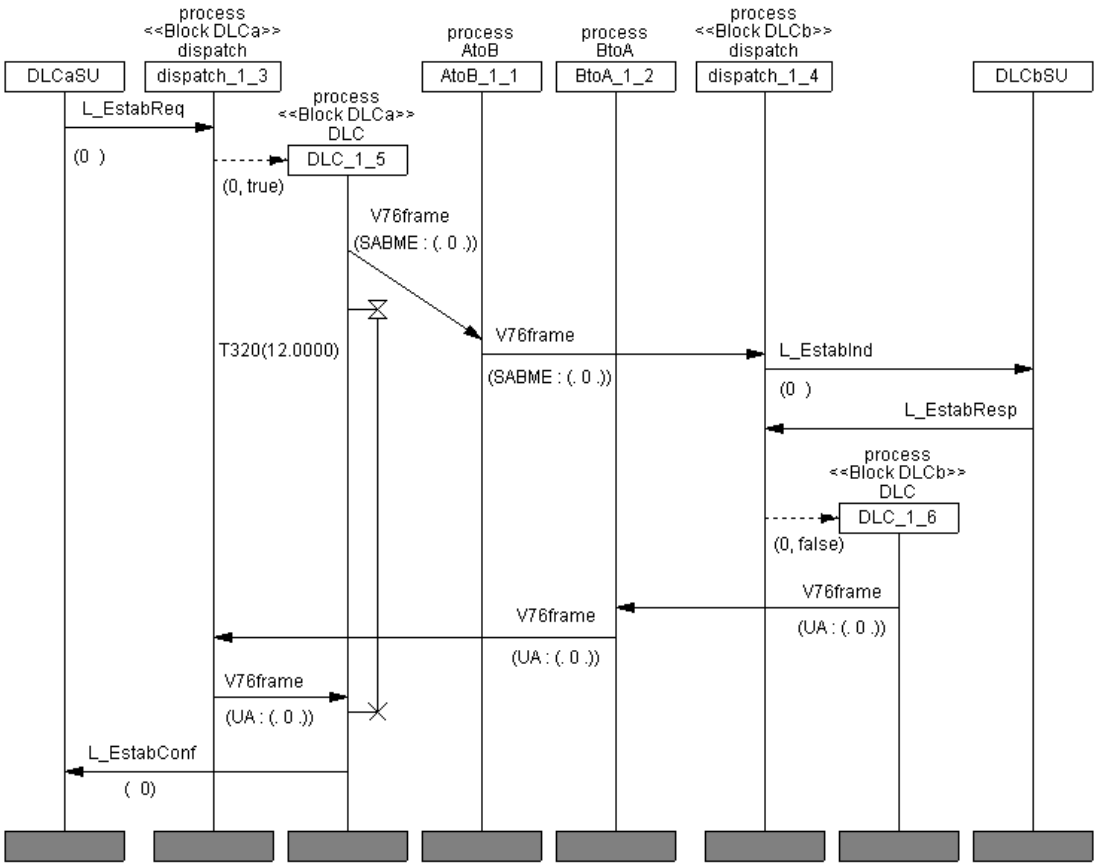
Ker želimo uspešno vzpostavitev povezave izberemo 1. Sedaj je procesni tip `toPeer` poslal signal, procesu `dispatch` na strani B. `Dispatch` po sprejemu signala, pošlje signal `L_EstabInd` uporabniku na svoji strani.



Slika 4.20: Okno "Select"

Proces DLC na strani A je zagnal časovnik T320. V izogib poteku časovnika ne nadaljujemo z izvajanjem prehodov, temveč pošljemo signal `L_EstabResp` procesu `dispatch` na strani `B`. Signal pošljemo z izbiro "Send To". Po izbiri `L_EstabResp`, določimo za ciljni proces `<<Block DLCb>> dispatch`. V nadaljevanju scenarija

nekajkrat izberemo “Transition”. Pri tem z izbiro `1 /*˝No˝*/` poskrbimo, da se signal ne izgubi. Ko se na diagramu MSC izriše izhodni signal `L_EstabConf`, vemo, da sta oba DLC-ja povezana. Slika 4.21 prikazuje diagram MSC, ki je bil tvorjen tekom simulacije.

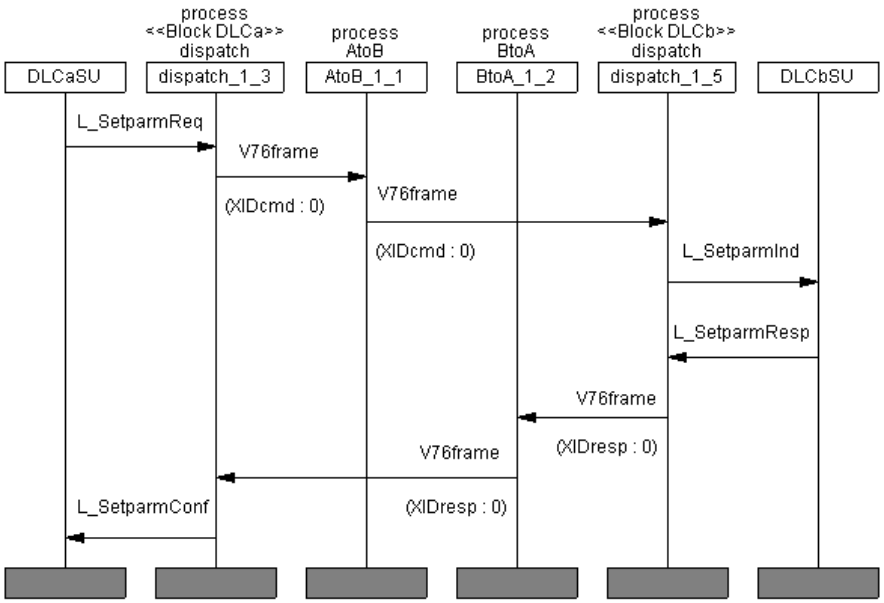


Slika 4.21: MSC vzpostavitev povezave

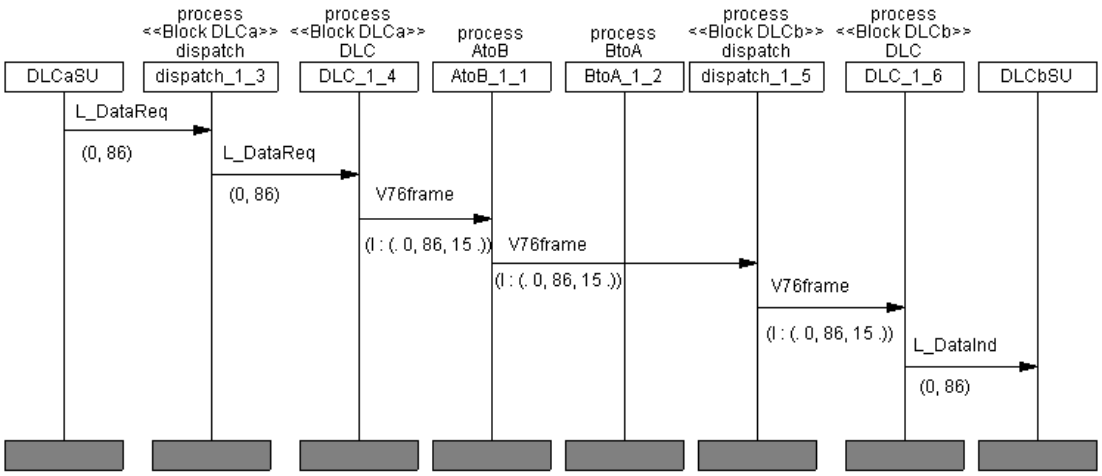
Z enakim postopkom izvedemo tudi preostale tri scenarije. Ob preverjanju prisotnosti najprej brezizgubno pošljemo signal `L_SetParmReq` procesu `dispatch` bloka `DLCa`. Po oddaji signala `L_SendParmInd` procesa `dispatch` bloka `DLCb`, brezizgubno pošljemo signal `L_SetParmResp` procesu `dispatch` bloka `DLCb`. Po sprejemu signala `L_SetParmConf` je prisotnost preverjena. Diagram MSC, ki je bil tvorjen tekom preverjanja prisotnosti je prikazan na sliki 4.22.

Prenos informacijskega okvirja poteka tako, da pošljemo procesu `dispatch` bloka `DLCa` signal `L_DataReq`, ki mu nastavimo vrednosti parametrov `DLCident` na `0` in

`Integer` na `86`. Tudi tokrat izberemo brezizgubno pošiljanje. Informacijski okvir je uspešno poslan. Diagram MSC poteka prikazuje slika 4.23.

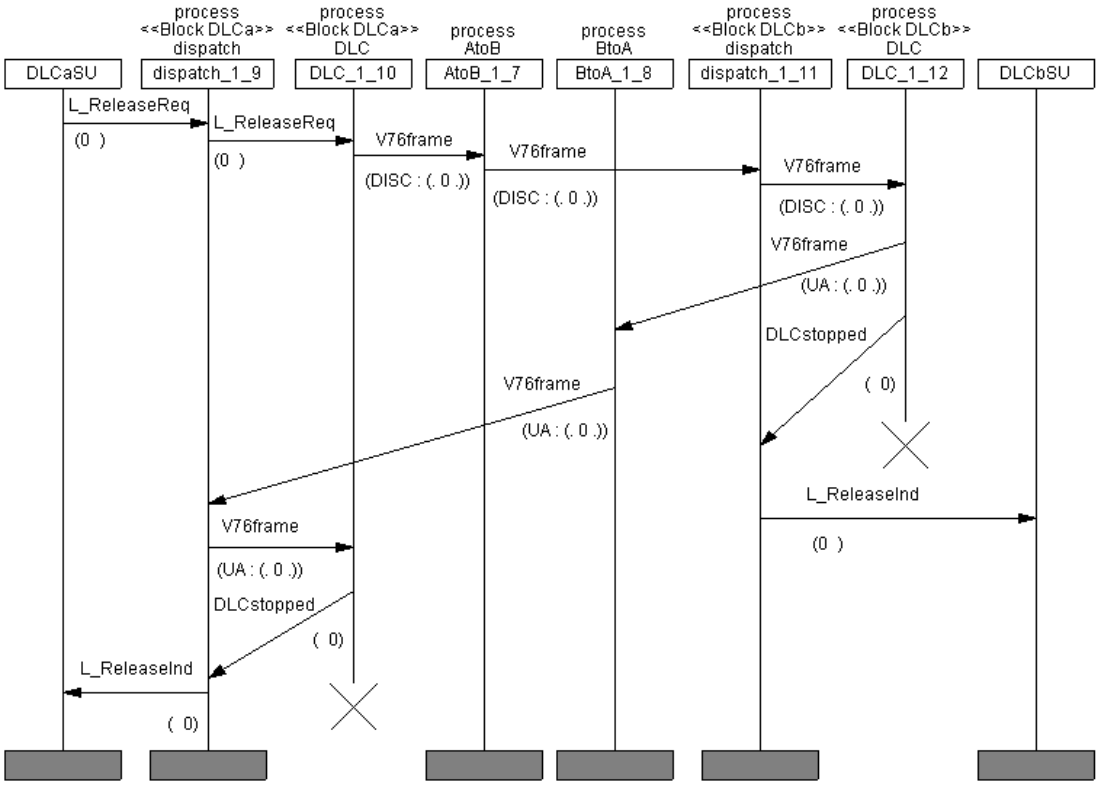


Slika 4.22: MSC preverjanje prisotnosti



Slika 4.23: MSC prenosa informacijskega okvirja

Za prekinitev povezave brezizgubno pošljemo procesu `dispatch` bloka `DLCa` signal `L_ReleaseReq` s parametrom `0`. Povezava je uspešno prekinjena, diagram MSC prikazuje slika 4.24.



Slika 4.24: MSC prekinitev povezave

4.1.5 Komentar orodja

Orodje Telelogic SDL Suite deluje v okoljih Windows in Unix. Pri našem preizkušanju programa, smo orodje uporabljali le v okolju Windows. Orodje je sestavljeno iz več oken, kar deluje malce nepregledno. Vsebuje pregleden priročnik za uporabo in kar nekaj primerov uporabe, kar omogoča novemu uporabniku lažje spoznavanje orodja.

Orodje SDL Suite podpira le grafični (SDL-GR) način zapisa specifikacije SDL. Sestavljen je iz več binarnih datotek in tekstovne datoteke s končnico ".sdt" (datoteka SDT), ki vsebuje informacijo o vseh binarnih datotekah, ki so del sistema. Urejanje kode v tekstualnem zapisu je mogoče samo z zunanjim urejevalnikom besedila. Urejamo ".pr" zapis, ki smo ga izvozili z izbiro "Convert to PR". Datoteko PR lahko v orodje uvozimo na dva načina:

1. brez ohranitve grafične predstavitve ali
2. z ohranitvijo grafične predstavitve [12].

Pri gradnji sistema orodje ne povezuje dodanih gradnikov, nas pa opozori z rdečim podčrtajem pri simbolih, ki zahtevajo besedilo in ga še nimajo. Pri premikanju elementa se premakne samo izbran element, ne pa tudi tisti ki so nanj povezani. Orodje omogoča samo en korak razveljavitve. Podpira komentarje CIF, prav tako podpira diagrame MSC in vsebuje orodje za simulacijo (opisano v poglavju 4.1.4). Izbiramo lahko med več možnostmi tiskanja specifikacije. SDL Suite vsebuje tudi orodji "Validator UI" in "Target Tester", ki sta namenjena testiranju in validaciji sistema, vendar ju zaradi njune obsežnosti nismo preverjali.

4.2 ObjectGEODE

4.2.1 Uvod

Trenutno se orodje ObjectGEODE ne trži, saj se je podjetje Telelogic ob prevzemu podjetja Verilog odločilo ponuditi le lastno orodje SDL Suite. Ker je razvojno orodje še vedno prisotno v mnogih podjetjih, smo ga vključili tudi v naš pregled orodij.

Orodje omogoča specifikacijo in opis sistemov z jeziki SDL, MSC in UML (Unified Modeling Language). Uporablja se v celotnem razvojnem ciklu – od načrtovanja sistema do generiranja kode in testiranja z uporabo simulacije. V vseh urejevalnikih se sintaktične napake v notaciji takoj zaznajo, uporabniku pa je podana razlaga o predvidenih vzrokih. To razvijalcu omogoča hitro učenje uporabljene notacije med samo uporabo programa. K večji učinkovitost prispevajo tudi algoritmi, ki zagotavljajo, da urejevalnik ponudi razvijalcu le tiste gradnike, ki so v trenutni točki sistema skladni s standardi.

Prehajanje med SDL, MSC in UML pogledi je omogočeno preko povezav. Spremembe nastavitev posameznega pogleda se poznajo tudi na drugih pogledih. Vsi urejevalniki podpirajo standardno povleči in spusti opcijo, kot tudi funkciji razveljavi in uveljavi ter funkcijo iskanja.

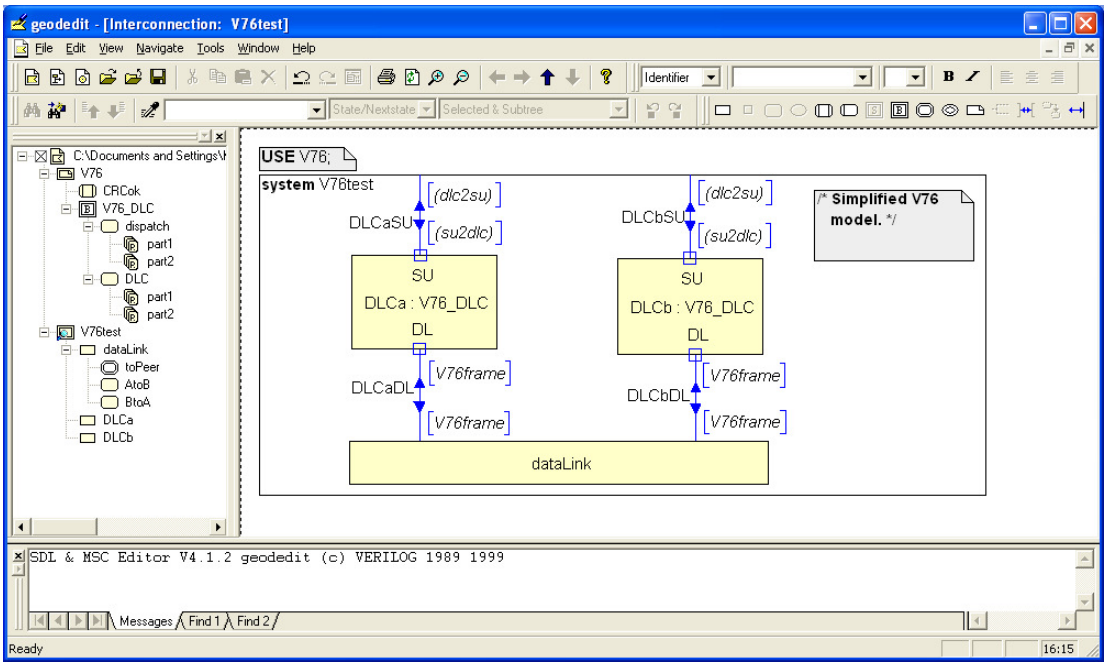
Hierarhična arhitektura sistema temelji na SDL gradnikih sistem, blok in proces. Različni konstrukti med seboj komunicirajo z uporabo signalov. Ti se prenašajo preko signalnih poti in kanalov. Proces SDL je opisan s pomočjo razširjenega končnega avtomata. Orodje podpira enostavno prehajanje med standardnim grafičnim in tekstovnim opisom sistema, saj je slednji uporabljen kot osnovni format zapisa sistema. Dodatne informacije o grafičnih lastnostih posameznega gradnika sledijo standardu Z.106 - “Common Interchange Format for SDL” [6].

Za opis scenarijev izvajanja sistema se uporabljajo diagrami MSC. Vsak diagram MSC predstavlja scenarij izmenjave signalov med komponentami sistema ali z okoljem ter se lahko uporabi ob preverjanju pravilnosti delovanja sistema z uporabo simulacije.

Orodju je priložena pregledna uporabniška dokumentacija, kjer so navedena tudi morebitna odstopanja od standardov in dodatne razširitve. Med pomembnejše razširitve gotovo sodi podpora mehanizmu zunanjih operatorjev ob definiciji abstraktnega podatkovnega tipa. S tem je omogočeno vključevanje zunanje programske kode.

4.2.2 Grafična podoba

Orodje ObjectGEODE ima pregleden grafični vmesnik. Ob zagonu urejevalnika "SDL & MSC Editor" se odpre okno delovnega okolja (slika 4.25). Okno delovnega okolja vsebuje menijsko vrstico, orodno vrstico, paleto objektov in polje sporočil. Na levi strani okna je prikazana hierarhija sistema, desno od nje se nahaja zgradba sistema. Orodje nam omogoča poljubno postavitev omenjenih podoken.



Slika 4.25: Pogled okna "SDL & MSC Editor"

4.2.3 SDL opis protokola V.76

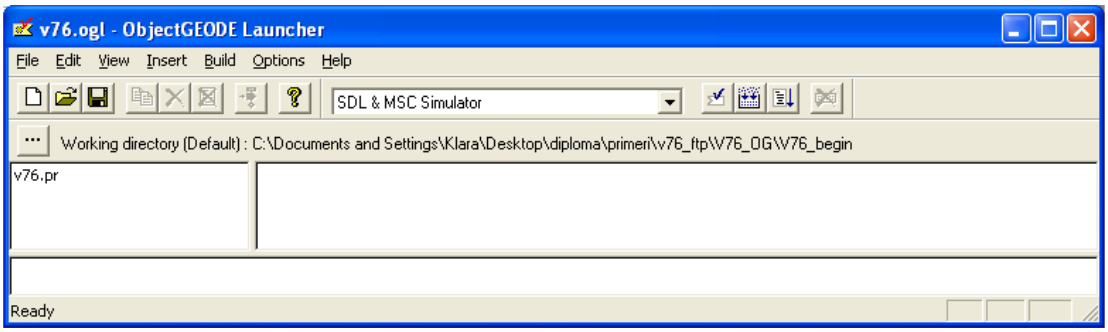
Opis protokola je povzet po knjigi Laurenta Doldija "Validation of Communications Systems with SDL" [2]. Opisan primer je dostopen na naslovu: ftp://ftp.wiley.co.uk/pub/books/ldoldi/.

Ker je zgradba sistema v orodju ObjectGEODE enaka zgradbi v orodju Telelogic SDL Suite, je na tem mestu ne bomo ponavljali, saj je že podana v poglavju 4.1.3.

4.2.4 Simulacija

V nadaljevanju je opisan postopek simulacije štirih scenarijev, ki so prikazani na sliki 3.1 (preverjanje prisotnosti, vzpostavitev povezave, prenos podatkov in prekinitev povezave). Uporabimo model, ki nam je na voljo na internetu [11] in ga odpremo v oknu

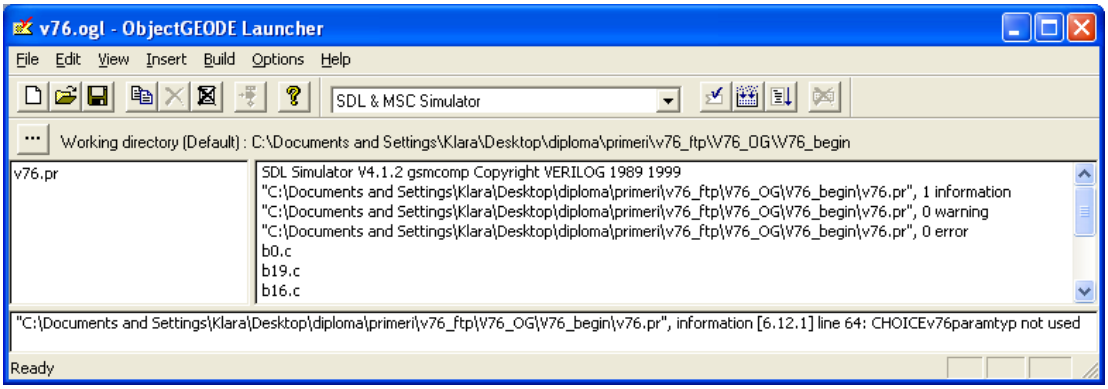
"SDL & MSC Editor" (slika 4.24). V oknu "SDL Editor" izberemo "Tools > SDL & MSC Simulator", pojavi se nam okno "ObjectGEODE Launcher", slika 4.26.



Slika 4.26: Pogled okna "ObjectGEODE Launcher"

Levi del okna vsebuje ime naše datoteke (v76.pr). Vedno ko v oknu "SDL Editor" kaj spremenimo, moramo SDL datoteko shraniti. V oknu "ObjectGEODE Launcher" izberemo "Build", preveri se ali je SDL model pravilen in če je, se prevede v kodo C, ki se shrani v mapo .geodesm. Zatem so C datoteke povezane s knjižnico, zgenerira se izvršilna datoteka "v76.sim".

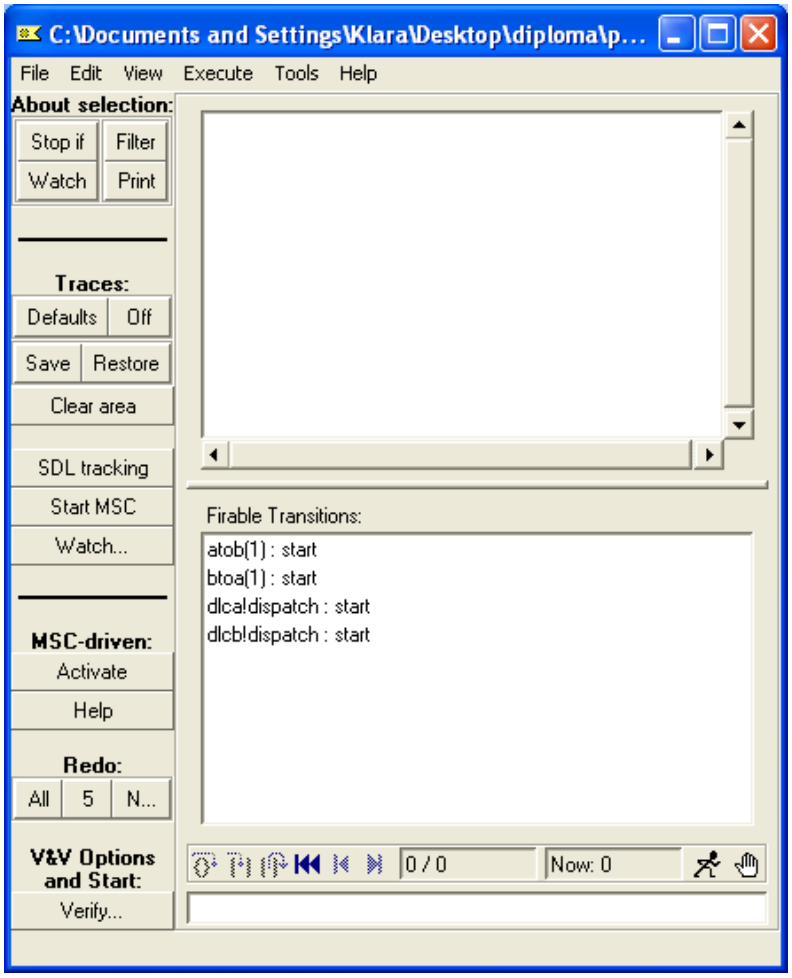
Za zagon simulatorja izberemo "Execute" (slika 4.27). Odpre se glavno okno simulatorja (slika 4.28).



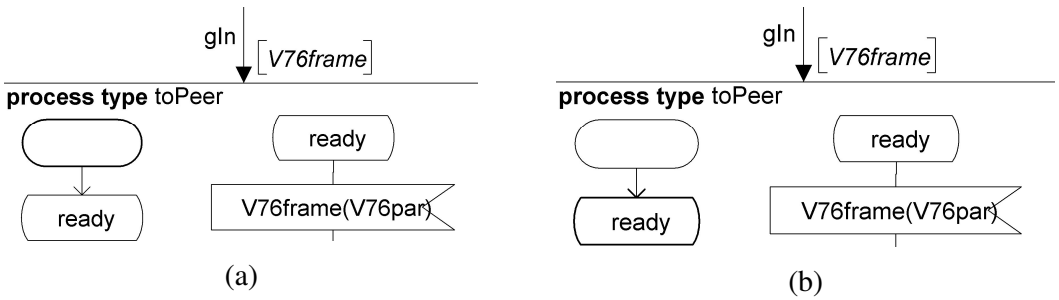
Slika 4.27: Pogled okna "ObjectGEODE Launcher"

V spodnjem oknu simulatorja vidimo štiri prehode, ki čakajo na izvršitev: `AtoB(1)`, `BtoA(1)`, `DLCa!dispatch` in `DLCb!dispatch`. Da lahko začnemo s pošiljanjem signalov, se morajo izvesti vsi štirje prehodi. Z izbiro opcije "SDL tracking" (slika 4.28) se nam odpre okno, v katerem lahko spremljamo kje v SDL modelu se nahajamo med

simulacijo. Izberemo prvi prehod, ki čaka na izvršitev. "SDL Editor" nam prikaže proces `AtoB(1)`, s poudarjenim simbolom "start", kot prikazuje slika 4.29 (a). Ob dvokliku na izbran prehod (atob(1):start), se izvede prehod iz simbola `start` v stanje `ready` (slika 4.29 (b)).



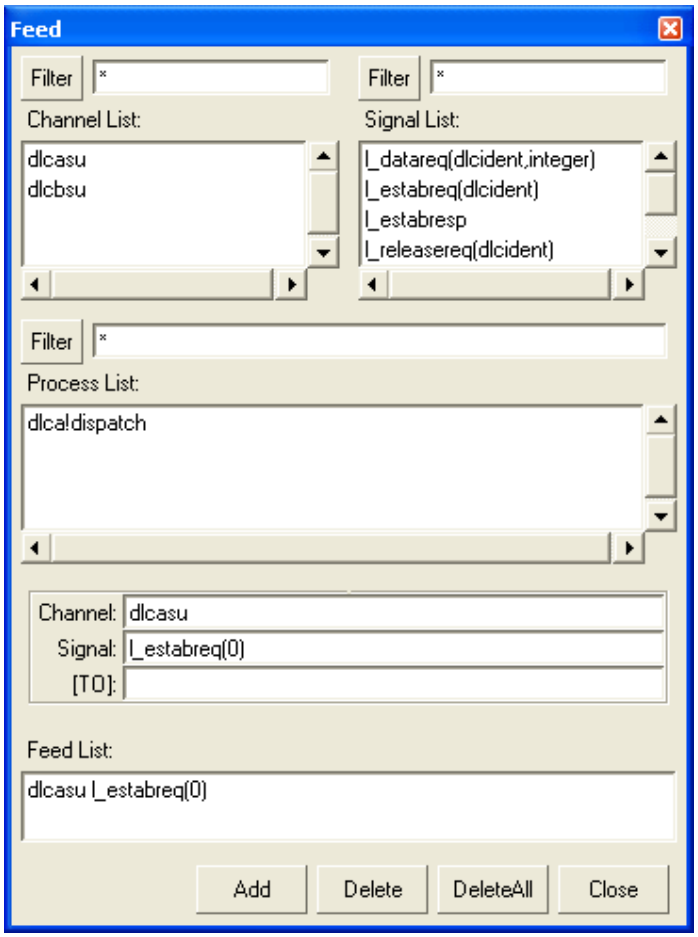
Slika 4.28: Glavno okno simulatorja



Slika 4.29: Prehod iz začetnega stanja v stanje "ready"

Na enak način izvedemo preostale inicializacijske prehode. Sedaj SDL sistem pričakuje signale iz okolja. Prve korake simulacije shranimo v datoteko start.scn, z ukazom "save start.scn". Naslednjič ko bomo zagnali simulacijo se bodo začetni prehodi izvedli avtomatsko in model bo pripravljen sprejemati signale iz okolja.

Za pošiljanje signalov iz okolja uporabimo ukaz "feed". V simulatorju izberemo "Edit > Feed" (slika 4.30). V tem oknu dodamo zunanje signale s parametri, ki jih bomo kasneje med simulacijo potrebovali. Dodamo jih tako, da izberemo kanal (npr. `dlcasu`), signal (npr. `l_estabreq(dlcident)`) in če signal nosi podatke, signalu dodamo vrednost parametra (npr. `l_estabreq(0)`). Primer dodanega signala prikazuje slika 4.30.



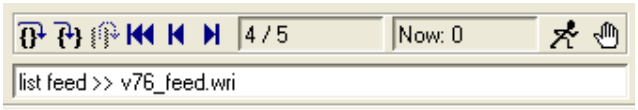
Slika 4.30: Okno "Feed"

Z enakim postopkom dodamo še naslednje signale:

- `dlcasu l_setparmreq ( ),`

- `dlcasu l_datareq (0,86),`
- `dlcasu l_releasereq (0),`
- `dlcbsu l_estabresp ( )in`
- `dlcbsu l_setparamresp ( ).`

Seznam dodanih signalov lahko shranimo tako z ukazom "list feed >> v76_feed.wri" (slika 4.31), tako se izognemo večkratnemu vnašanju signalov.



Slika 4.31: Shranjevanje liste signalov

Ob zagonu simulatorja lahko poženemo zagonsko datoteko, ki vsebuje seznam signalov in izvede začetne prehode. V izbranem urejevalniku besedila naredimo datoteko v76.startup, ki vsebuje:

```
source v76_feed.wri
source start.scn
```

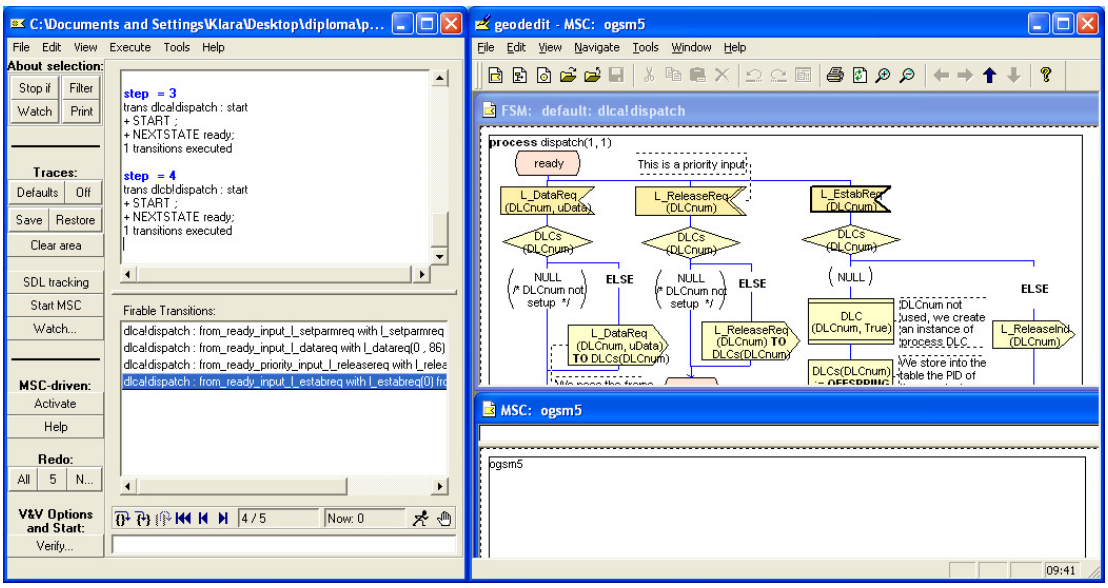
Shranimo jo v mapo v kateri se nahaja sistem, ki ga simuliramo. Ob naslednjem zagonu simulatorja se avtomatsko izvrši datoteka v76.startup. Izvedejo se začetni prehodi in naloži se seznam signalov.

Za sledenje izvajanja simulacije z uporabo diagramov MSC izberemo "Start MSC" (slika 4.32).

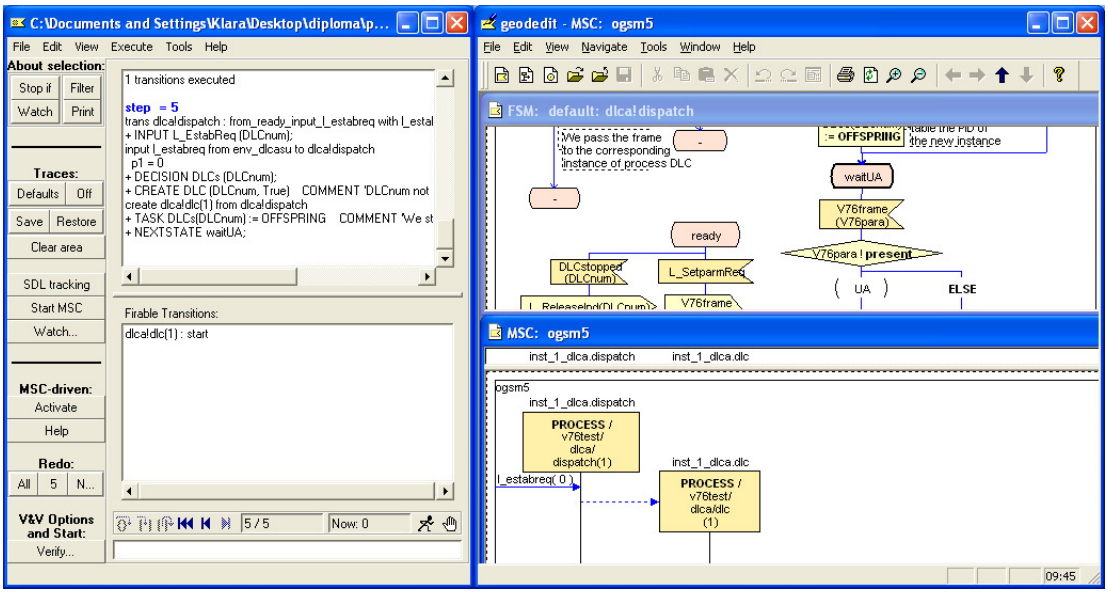
Najprej smo simulirali vzpostavitev povezave. V oknu simulatorja "Firable Transitions" z dvoklikom izberemo prehod:

- `trans dlca!dispatch : from_ready_input_l_estabreq with L_estabreq(0) from env_dlcasu.`

Simulator pošlje signal `L_EstabReq`, ki ga sprejme proces `dispatch` in izvede prehod v stanje `waitUA` (slika 4.32). V MSC diagramu vidimo, da je proces `dispatch` na strani `A` sprejel signal `L_EstabReq`, s parametrom 0 in je kreiral proces `DLC` (slika 4.33).



Slika 4.32: Okno “Editor” in “Simulator”



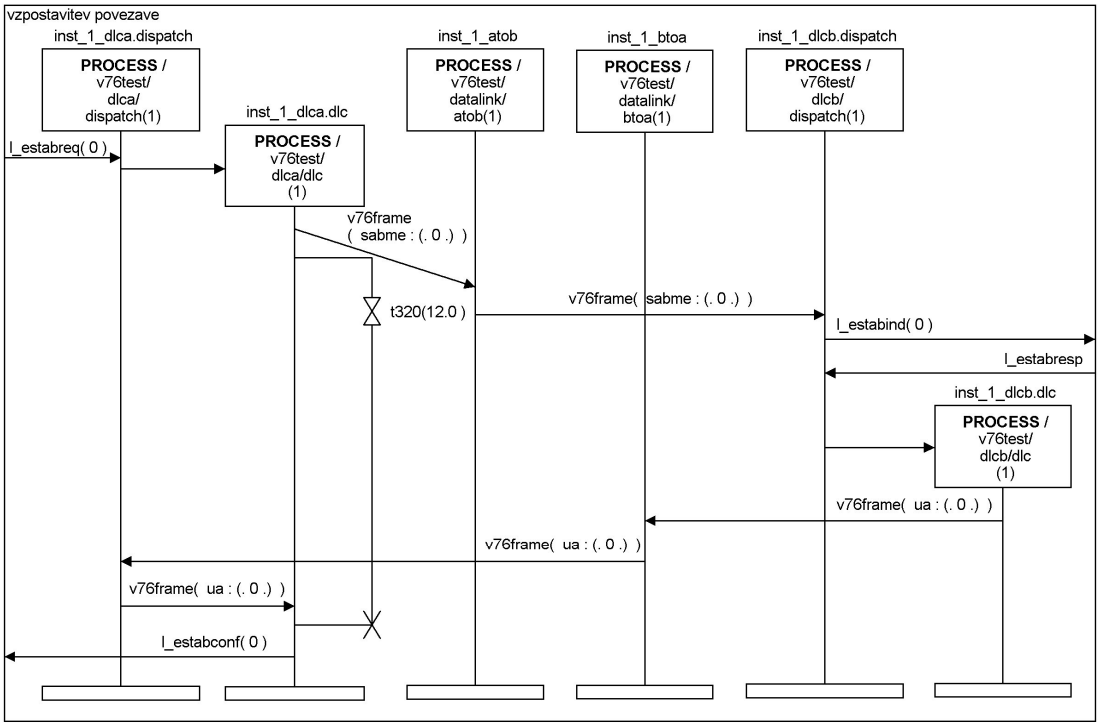
Slika 4.33: Okno “Editor” in “Simulator” po prvem prehodu

Celotna vzpostavitev se izvede z naslednjimi prehodi:

- `trans dlca!dlc(1) : start,`
- `trans atob(1) : from_ready_input_v76frame,`
- `trans atob(1) : decision_lose_the_frame('No'),`
- `trans dlcb!dispatch : from_ready_input_v76frame,`
- `trans dlcb!dispatch : from_waitestabresp_input_l_estabresp with l_estabresp from env_dlcbsu,`

- `trans dlcb!dlc(1) : start,`
- `trans btoa(1) : from_ready_input_v76frame,`
- `trans btoa(1) : decision_lose_the_frame('No'),`
- `trans dlca!dispatch : from_waitua_input_v76frame,`
- `trans dlca!dlc(1) : from_waitua_input_v76frame.`

Po izvedeni zadnji transakciji je povezava uspešno vzpostavljena. Slika 4.34 prikazuje diagram MSC uspešne vzpostavitve povezave.



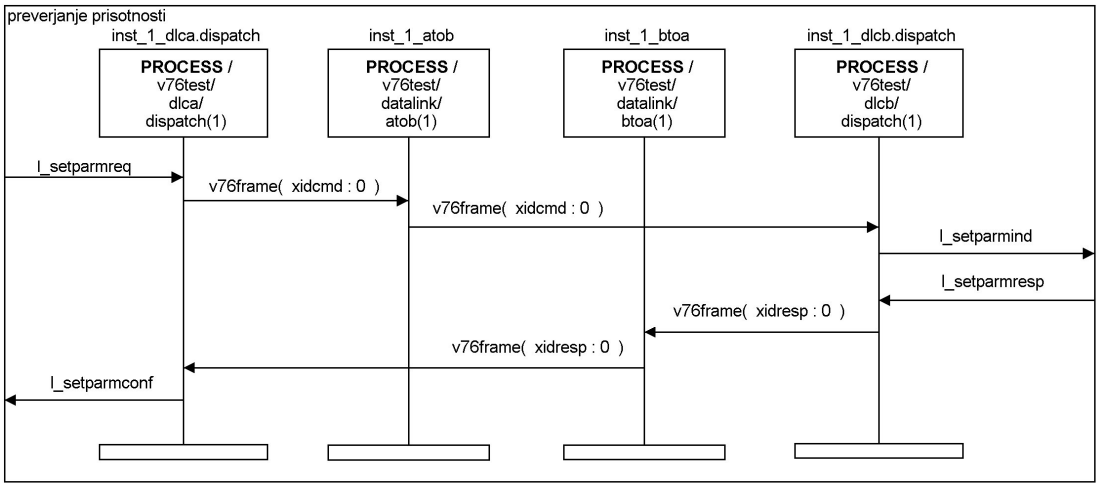
Slika 4.34: MSC vzpostavitev povezave

Po enakem postopku izvedemo tudi preverjanje prisotnosti:

- `trans dlca!dispatch : from_ready_input_l_setparmreq with l_setparmreq from env_dlcasu,`
- `trans atob(1) : from_ready_input_v76frame,`
- `trans atob(1) : decision_lose_the_frame('No'),`
- `trans dlcb!dispatch : from_ready_input_v76frame,`
- `trans dlcb!dispatch : from_waitparmresp_input_l_setparmresp with l_setparmresp from env_dlcbsu,`
- `trans btoa(1) : from_ready_input_v76frame,`
- `trans btoa(1) : decision_lose_the_frame('No'),`

- `trans dlca!dispatch : from_ready_input_v76frame.`

Rezultat je diagram MSC, ki je predstavljen na sliki 4.35.

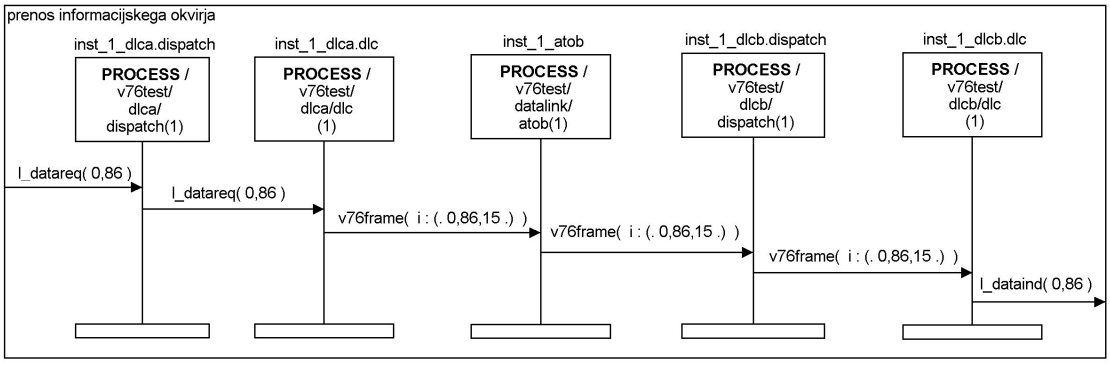


Slika 4.35: MSC preverjanje prisotnosti

Po uspešno vzpostavljeni povezavi, lahko sledi prenos informacijskega okvirja. Pošljemo signal `I_DataReq (0, 86)` s parametrom 86. Potek simulacije je naslednji:

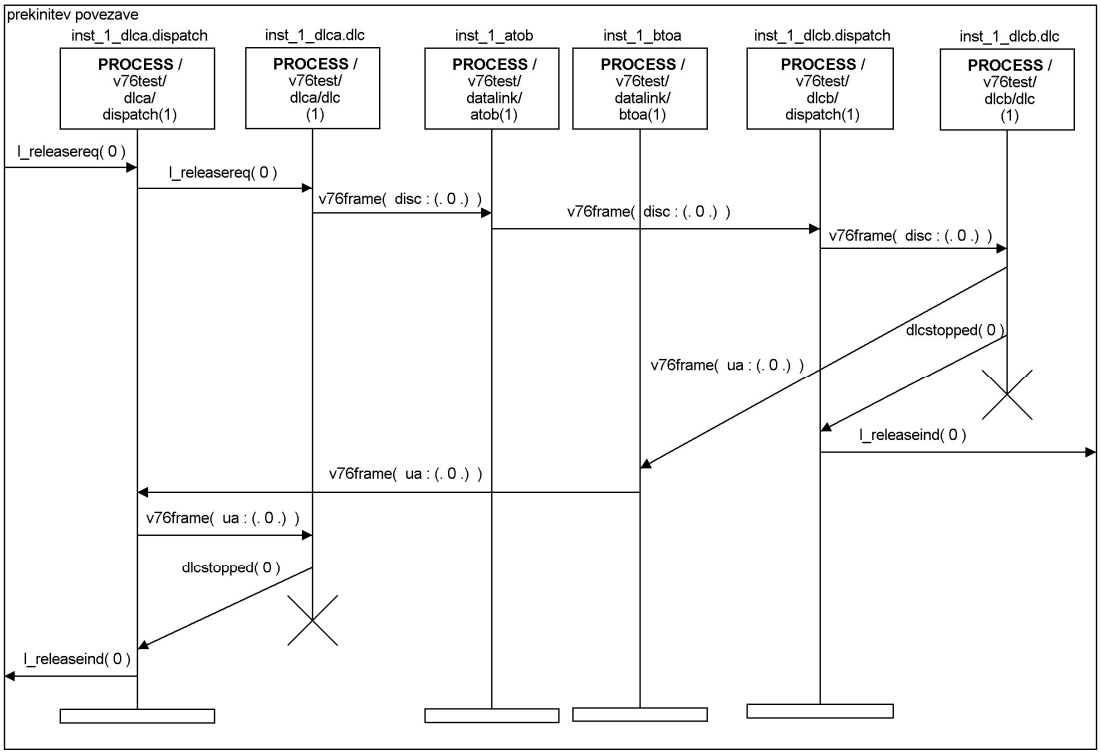
- `trans dlca!dispatch : from_ready_input_l_datareq with l_datareq(0 , 86) from env_dlcasu,`
- `trans dlca!dlc(1) : from_connected_input_l_datareq,`
- `trans atob(1) : from_ready_input_v76frame,`
- `trans atob(1) : decision_lose_the_frame('No'),`
- `trans dlcb!dispatch : from_ready_input_v76frame,`
- `trans dlcb!dlc(1) : from_connected_input_v76frame.`

Slika 4.36 prikazuje diagram MSC uspešnega prenosa podatkov.



Slika 4.36: MSC prenos informacijskega okvirja

Slika 4.37 prikazuje diagram MSC prekinitve povezave, kjer uporabnik na strani A zahteva prekinitev.



Slika 4.37: MSC prekinitev povezave

Za prekinitev povezave izvedemo naslednje prehode:

- `trans dlca!dispatch : from_ready_priority_input_l_releasereq with l_releasereq(0) from env_dlcasu,`
- `trans dlca!dlc(1) : from_connected_input_l_releasereq,`
- `trans atob(1) : from_ready_input_v76frame,`
- `trans atob(1) : decision_lose_the_frame('No'),`
- `trans dlcb!dispatch : from_ready_input_v76frame,`
- `trans dlcb!dlc(1) : from_connected_input_v76frame,`
- `trans dlcb!dispatch : from_ready_input_dlcstopped,`
- `trans btoa(1) : from_ready_input_v76frame,`
- `trans btoa(1) : decision_lose_the_frame('No'),`
- `trans dlca!dispatch : from_ready_input_v76frame,`
- `trans dlca!dlc(1) : from_waituadisc_input_v76frame,`
- `trans dlca!dispatch : from_ready_input_dlcstopped.`

4.2.5 Komentar orodja

Orodje ObjectGEODE deluje v okoljih Windows, Solaris in Unix. Orodje smo uporabljali le v okolju Windows. Orodje ima pregleden grafični vmesnik, vsebuje izčrpen priročnik za uporabo, prav tako vsebuje primere uporabe.

Specifikacija SDL se v orodju ObjectGEODE shrani v tekstovno datoteko s končnico ".pr", tako je tudi omogočeno enostavno prehajanje med tekstovnim in grafičnim opisom sistema. Po privzetem datoteka vsebuje zapis o grafični predstaviti, za kar se uporabljajo komentarji CIF. Poleg privzetega formata je specifikacijo SDL mogoče shraniti še v treh drugih formatih: SDL za izvoz ("SDL for Export"), vgnezden SDL brez komentarjev CIF ("Nested SDL Without CIF") in SDL brez vgnezdenja in brez komentarjev CIF ("Flat SDL Without CIF"). Datoteke vseh naštetih formatov imajo končnico ".pr".

Gradnja sistema v orodju ObjectGEODE je učinkovita. Orodje nam ponudi le tiste gradnike, ki so v trenutni točki sistema skladni s standardom in jih tudi sam poveže. Sintaktične napake v notaciji so takoj zaznane, uporabniku pa je podana razlaga o predvidenih vzrokih. Orodje podpira diagrame MSC in nam omogoča tiskanje specifikacije.

4.3 Cinderella

4.3.1 Uvod

Podjetje Cinderella I/S so ustanovili leta 1995 v Kopenhagnu, na Danskem. Prva različica komercialnega orodja Cinderella SDL je bila izdana aprila 1998. Ciljno skupino predstavljajo podjetja, ki se ukvarjajo z razvojem telekomunikacijskih sistemov, informacijskimi tehnologijami, proizvodnjo elektronike, ipd. Orodje uporabljajo tako majhni, kot največji svetovni proizvajalci telekomunikacijskih sistemov, kakor tudi številne univerze in izobraževalne ustanove [5].

Cinderella SDL temelji na jeziku SDL in se izvaja na operacijskem sistemu Windows. Uporablja se za grafično modeliranje in razvoj sistemov, ki temeljijo na pošiljanju sporočil. Omogoča analizo in simulacijo sistema SDL, uporabo standardiziranih notacij SDL, MSC in ASN.1 ter uvažanje in izvažanje projektov v/iz Telelogic Tau in ObjectGEODE.

Orodje ima integriran grafični urejevalnik, analizator in simulator. Grafični urejevalnik omogoča zapis specifikacije. S pomočjo analizatorja SDL je omogočeno sprotno popravljanje omejene množice napak. Simulacijo sistema lahko izvedemo s pomočjo simulatorja, ki omogoča simulacijo celotnega sistema ali samo izbranega dela. Orodje omogoča vključitev dopolnilnih orodij:

- "Cinderella Slipper" – generiranje in razhroščevanje kode C,
- "Cinderella Site" – generiranje kode C++ in
- "Cinderella MSC" – generiranje testnih diagramov MSC in izvoz diagramov MSC v podprte formate (npr. Telelogic Tau) [5].

Orodje omogoča hierarhičen prikaz specifikacije, kjer lahko preklapljamo med različnimi pogledi specifikacije. Hkrati je lahko odprtih več oken, kjer lahko v vsakem oknu določimo različen pogled. Dodatno lahko uporabnik vpliva na barvo grafične predstavitve konstruktov. Orodje podpira tudi format CIF [6], kar prispeva k lažji izmenjavi specifikacij med različnimi orodji s podporo jeziku SDL.

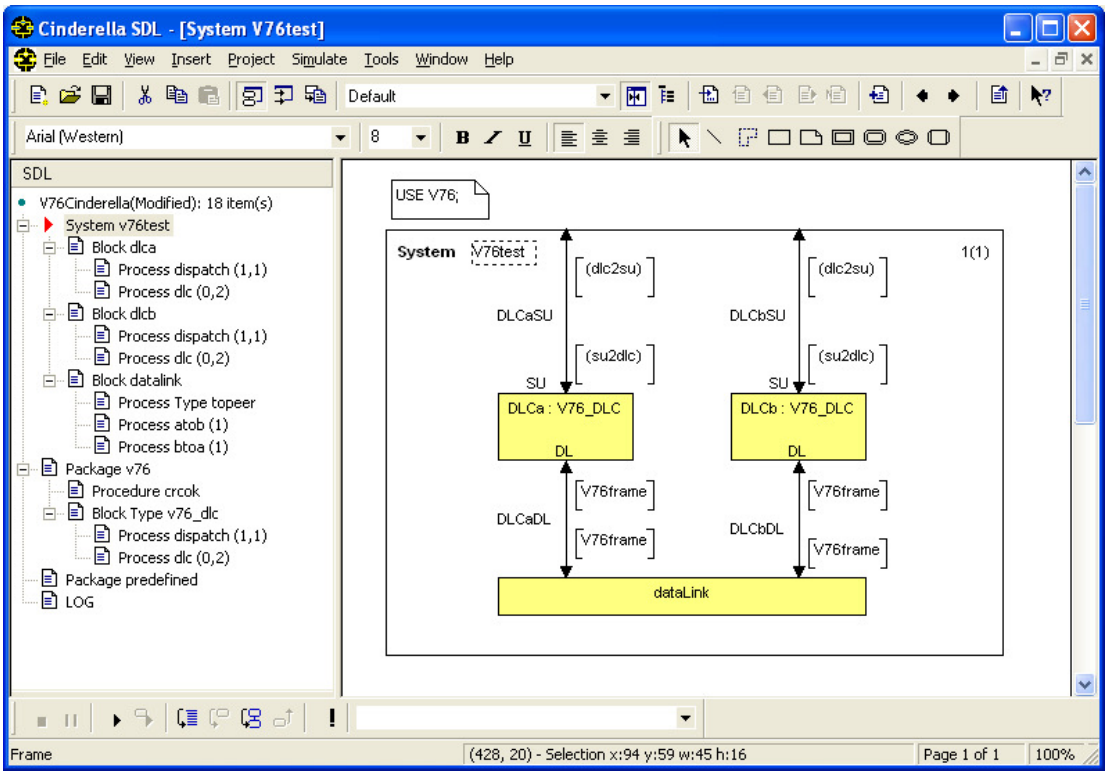
Projekt se shrani v binarnem formatu s končnico ".cbf" (Cinderella Binary Format). Zato sistema ne moremo urejati z urejevalniki besedila. Orodje omogoča izvoz opisa sistema v naslednje tekstualne formate:

- format SDL PR, končnica ".pr",
- format CIF, končnica ".cif" in
- format STF (SDL Textual Format), končnica ".stf".

Pri ponovnem uvozu izvoženega tekstualnega opisa in uporabi opcije “PR to GR” se grafični prikaz sistema ne ohrani, funkcionalnost sistema pa ostane nespremenjena.

4.3.2 Grafična podoba

Uporabniški vmesnik orodja Cinderella je pregleden. Slika 4.38 prikazuje delovno okno orodja. Na levi strani je prikazana hierarhija sistema, na desni pa je prikazana zgradba sistema. V zgornjem delu okna se nahajajo orodna vrstica s simboli, orodna vrstica za oblikovanje in glavna orodna vrstica. V spodnjem levem kotu okna je orodna vrstica za simulacijo. Orodje omogoča poljubno vključitev in postavitev vseh orodnih vrstic.

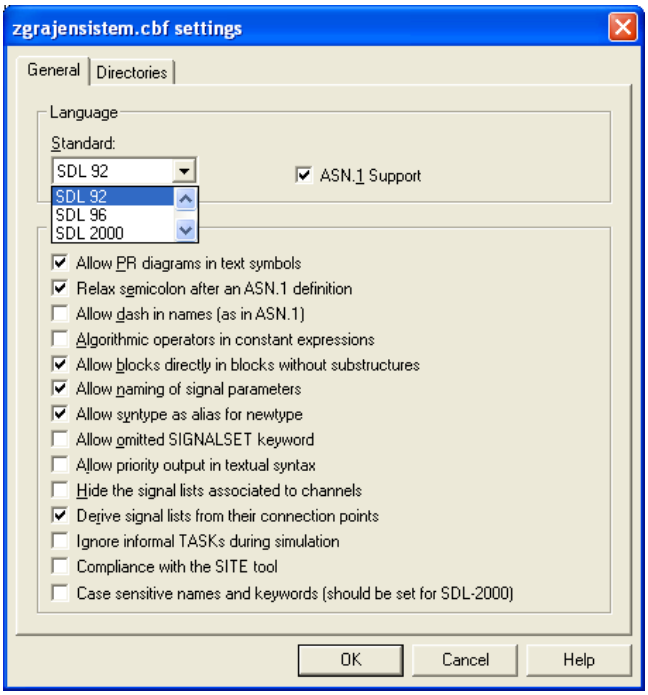


Slika 4.38: Uporabniški vmesnik programa Cinderella

4.3.3 SDL opis protokola V.76

Opis protokola je povzet po knjigi [2]. Primer lahko uvozimo v orodje Cinderella, če ga v orodju ObjectGEODE shranimo kot “SDL for Export”. Ker simulacija ni delovala po pričakovanjih, smo se odločili za samostojno gradnjo sistema in sprotno preverjanje delovanja.

Pred začetkom gradnje sistema v meniju “Project > Settings” izberemo želeno različico SDL. Na voljo imamo SDL-88, SDL-92, SDL-96 in SDL-2000 (slika 4.39). Tudi pri tem orodju smo izbrali SDL-92.



Slika 4.39: Okno z nastavitvami

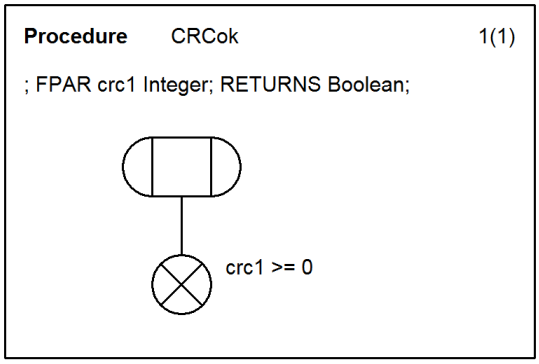
V meniju “Tools > Options” lahko nastavimo nivo analize: 1 – preverja kako so povezani grafični simboli, 5 – celotna analiza sistema, navzkrižno preverjanje in priprava na simulacijo, pri tej stopnji se preverja ali so vsi signali, ki so definirani dejansko kasneje tudi uporabljeni, ali se signal, ki ga nek proces pošlje drugemu procesu, lahko sprejme,… Izberemo lahko na koliko minut nam avtomatsko shranjuje sistem, določimo barve simbolov, določimo mapo iz katere želimo vključevati datoteke, itd.

Gradnja sistema je dokaj preprosta. Za razvijalca bi lahko bilo moteče, da ko izberemo gradnik in ga želimo poimenovati, ni jasno razvidno ali lahko v gradnik pišemo ali ne. Med gradnjo sistema so na izbiro vsi gradniki, orodje nas omejuje pri povezovanju le-teh in tako zagotavlja skladnost s standardom. Gradnike povezujemo tako, da z desnim gumbom miške kliknemo nanj, izberemo “connect” in naslednji gradnik. Orodje nas ves čas gradnje sistema opozarja na napake, npr. če dodamo nov gradnik, takoj opozori, da ni povezan in poimenovan. Preverjajo se tudi signalne poti in kanali. Orodje nas takoj opozori, če

nimamo definiranih signalov, ki se prenašajo in določenih vrat na katera se signalna pot oz. kanal povezuje. Pri poimenovanju gradnikov v grafični predstavitvi je moteče, da ni možno linearno nastaviti velikosti gradnika in pogosto se zgodi, da sega besedilo čez rob gradnika. Ta težava bo vidna tudi na uporabljenih slikah.

Zgradba sistema v orodju Cinderella je enaka zgradbi v orodju Telelogic SDL Suite. Na tem mestu bomo omenili le dele, pri katerih smo ob gradnji sistema opazili posebnosti. Opis celotnega sistema je podan v poglavju 4.1.3.

Pri proceduri `CRCok` je potrebno definirati nov formalni parameter `crc1` tipa `Integer` in določiti tip podatka, ki ga procedura vrača. V meniju "View" omogočimo "Heading", prikažejo se nam okvirčki "Virtuality", "Exportspec", "Qualifier" in "Heading". V okvir "Heading" vpišemo definicijo procedure. Specifikacija procedure `CRCok` je prikazana na sliki 4.40.

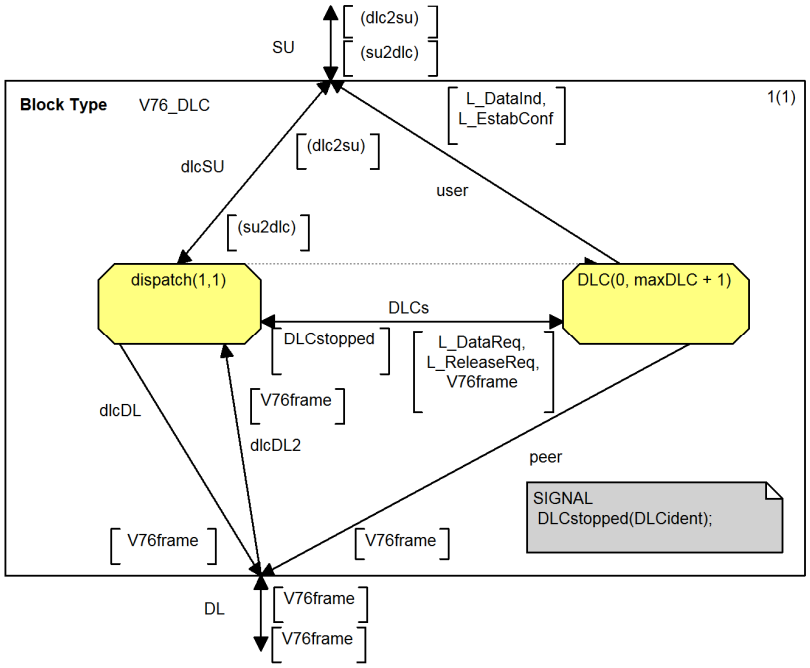


Slika 4.40: Procedura `CRCok`

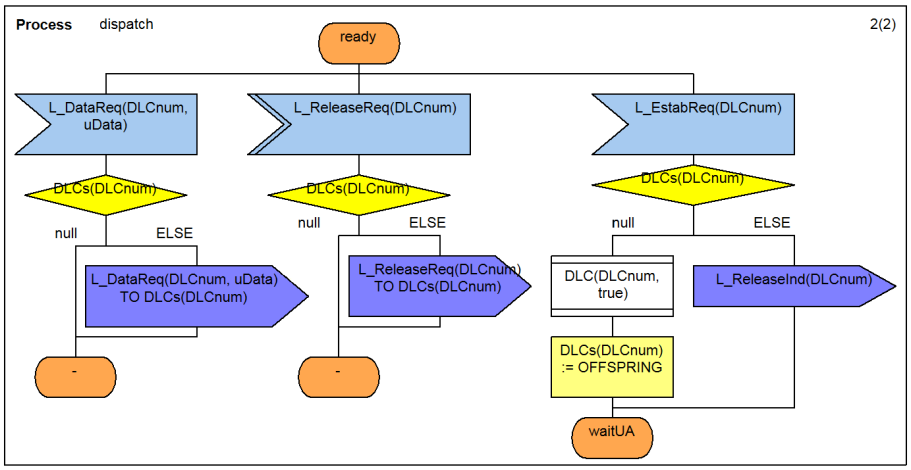
Blokovni tip `V76_DLC`, vsebuje procesa `dispatch` in `DLC`, ki se preko signalnih poti povezujeta na vrata `SU` in `DL` (slika 4.41). Ko ustvarimo novo signalno pot od procesa do okolja, nam program za vsako pot ustvari svoja vrata. Ker sta na vrata `SU` povezani dve signalni poti, je potrebno ena vrata izbrisati in prestaviti končno točko povezave na druga vrata. Ob tem nam orodje izpiše opozorilo, v katerem sprašuje ali naj združi končni točki. V kolikor nam program ne ponudi združitve končnih točk, bo v grafičnem prikazu izgledalo, kot da sta obe signalni poti povezani na ista vrata vendar bo ena signalna pot brez povezave. Da je signalna pot brez povezave nam sistem javi, kot napako.

Orodna vrstica s simboli ne vsebuje konstrukta prioritetni vhod. Njegovo obnašanje lahko opišemo s pomočjo konstrukta vhod, kjer je potrebno ustrezno nastaviti prioriteto

sprejema signala. z izbiro “Attributes > High Priority”. Ob tem se spremeni tudi grafični prikaz konstrukta – signal `L_ReleaseReq(DLCnum)` na sliki 4.42.



Slika 4.41: Blokovni tip `V76_DLC`

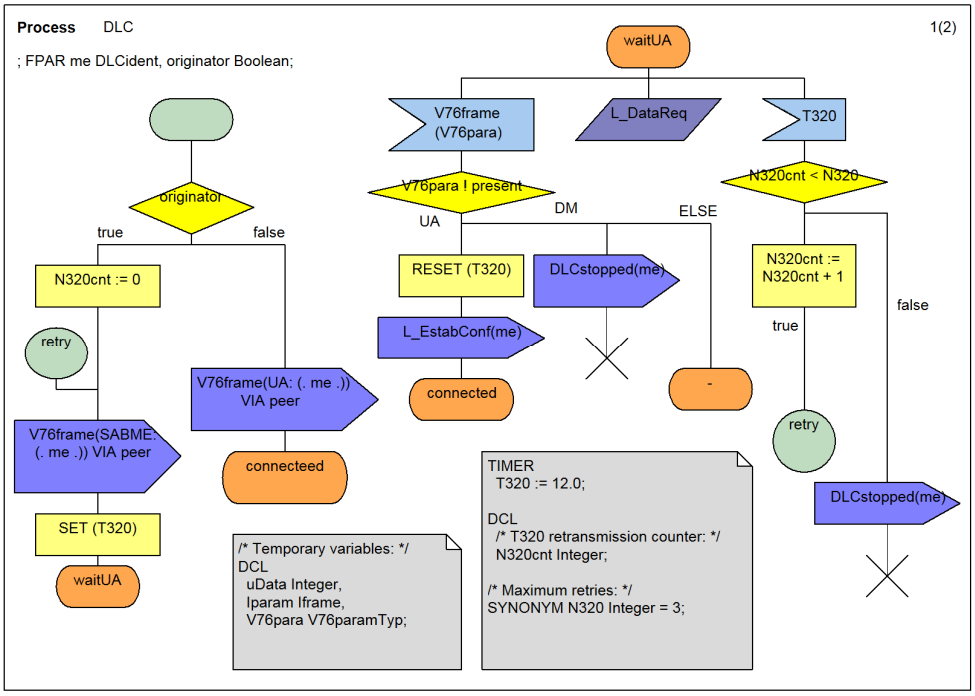


Slika 4.42: Grafični opis specifikacije drugega dela procesa `dispatch`

Cinderella dodatno ne vsebuje konstruktov `nextstate` in `join`, vendar se jih da implementirati z uporabo konstruktov `state` namesto `nextstate` in `label` namesto `join`. Konstrukta `nextstate` in `join` sta prikazana na sliki 4.43, ki prikazuje

specifikacijo prvega dela procesa `DLC`. Konstrukt `nextstate` je poimenovan "waitUA" in "connected", konstrukt `join` pa je poimenovan "retry".

Orodja ne podpira grafičnega konstrukta za opis makrojev. Makre je mogoče opisati samo v tekstovni obliki, na kar opozarja tudi dokumentacija orodja [13]. Ker naš testni primer ne vsebuje makrov tega nismo podrobneje preverjali.



Slika 4.43: Grafični opis specifikacije prvega dela procesa `DLC`

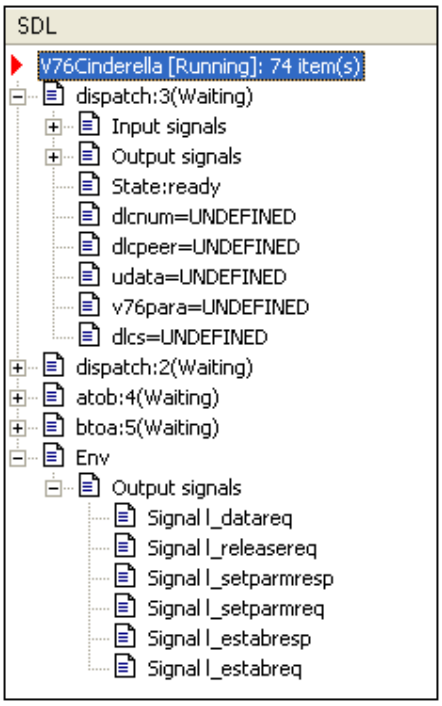
4.3.4 Simulacija

Orodje Cinderella vsebuje simulator, ki je preprost za uporabo in omogoča simulacijo delov sistema že med samo gradnjo. Simulacija lahko zajema celoten sistem, dele sistema ali en sam proces [5]. Med simulacijo lahko obnašanje sistema opazujemo z MSC diagrami.

Simulacijo lahko poženemo iz orodne vrstice "Simulate > Start" (F5) ali z izbiro gumba "play" v orodni vrstici za simulacijo.

Ko zaženemo simulacijo protokola V.76, se procesi `dispatch:3` iz bloka `DLCa`, `dispatch:2` iz bloka `DLCb` in `atob:4`, `btoa:5` iz bloka `dataLink` postavijo v stanje `ready` in so pripravljeni na sprejem signala iz okolice. V hierarhičnem pogledu

specifikacije, simulator doda še okolje (Env). Okolje ima pod "Output signals" naštete signale, ki jih simulator pripravi glede na možne sprejeme v sistem (slika 4.44). Signali, ki jih lahko pošljemo iz okolice so: `l_datareq`, `l_releasereq`, `l_setparmresp`, `l_setparmreq`, `l_estabresp` in `l_estabreq`.



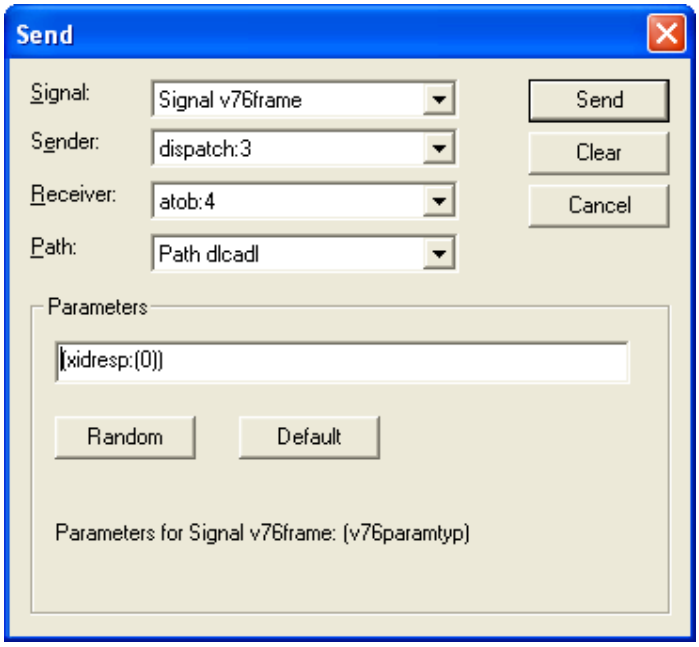
Slika 4.44: Hierarhičen pogled po zagonu simulacije

Tekom simulacije lahko sledimo izvajanju sistema tudi z uporabo diagrama MSC ("View > MSC"). Diagrama ne moremo spreminjati, lahko ga le natisnemo. Zato na prikazanih diagramih MSC imena signalov segajo čez črte.

Signal pošljemo z izbiro "Send signal" (slika 4.45). Iz seznama signalov izberemo signal, pošiljatelja, prejemnika in pot. V kolikor signal nosi parametre nastavimo tudi te. Če vrednosti parametrov niso navedene, se signal pošlje brez njih. Simulacija se ustavi v točki, kjer prejemnik signala za nadaljevanje potrebuje manjkajoče parametre.

Najprej smo simulirali preverjanje prisotnosti obeh uporabnikov (slika 3.1). Signal `l_setparmreq` pošljemo procesu `dispatch:3` iz bloka `DLCa`, preko povezave `dlcasu`. Po prejetju signala `l_setparmreq` proces `dispatch:3` pošlje signal

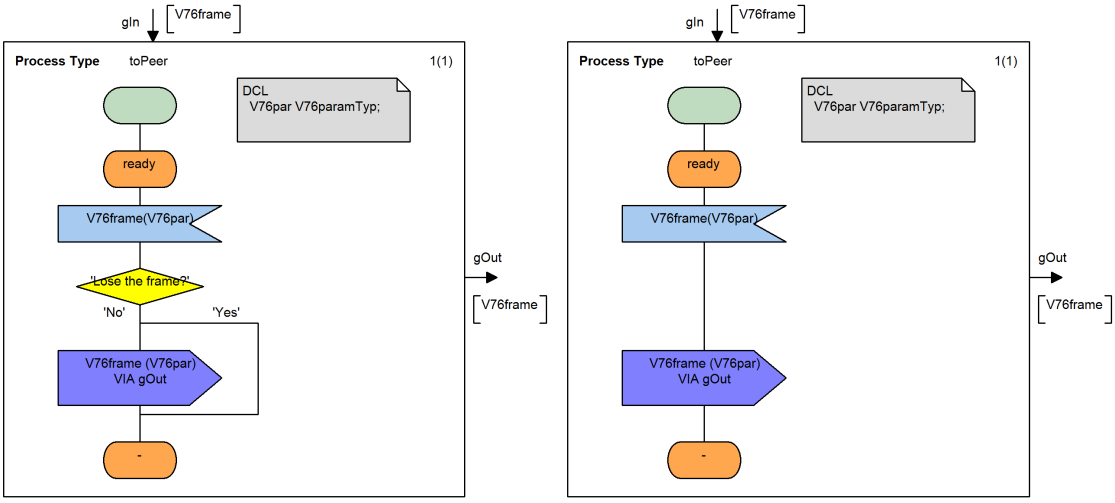
`v76frame(xidcmd(0))` procesu `atob:4` iz bloka `dataLink`. Izbrane parametre prikazuje slika 4.45.



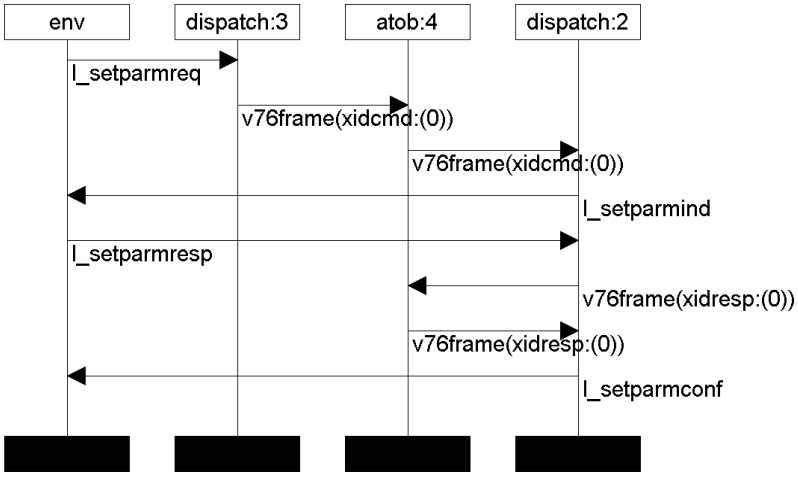
Slika 4.45: Okno "Send"

Simulator v procesnem tipu `toPeer` iz bloka `dataLink` javi napako, da nobena veja ne ustreza odločitvi. Zato smo procesni tip priredili tako, da prejeti signal vedno posreduje naprej (slika 4.46). S tem smo onemogočili testiranje robustnosti sistema, saj se signal nikoli ne izgubi. Lahko bi se odločili in uporabili nedeterministično izbiro, s konstruktom odločitev in rezervirano besedo "any". Vendar potem ne bi mogli zagotoviti, da bi se signal vedno posredoval dalje. V dokumentaciji orodja [13] nismo našli rešitve, ki bi omogočala uporabniku izbiro po kateri poti naj se simulacija nadaljuje.

V popravljenem sistemu ponovno zaženemo simulacijo in pošljemo signal za preverjanje prisotnosti `l_setparmreq` procesu `dispatch:3` iz bloka `DLCa`. Po prejetju zahteve za preverjanje prisotnosti proces `dispatch:2` iz bloka `DLCb` pošlje okolju signal `l_setparmind`. Simulacija se ustavi, saj sistem čaka na signal iz okolja. Prisotnost potrdimo s signalom `l_setparmresp`, ki ga iz okolja pošljemo procesu `dispatch:2` iz bloka `DLCb`. Slika 4.47 prikazuje potek izvajanja simulacije.

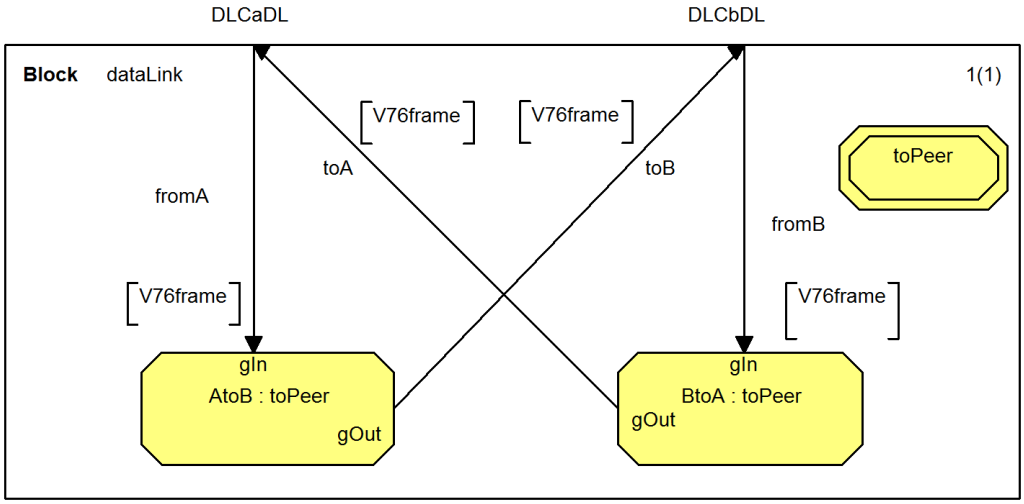


Slika 4.46: Procesni tip `toPeer` pred in po spremembi



Slika 4.47: MSC preverjanja prisotnosti

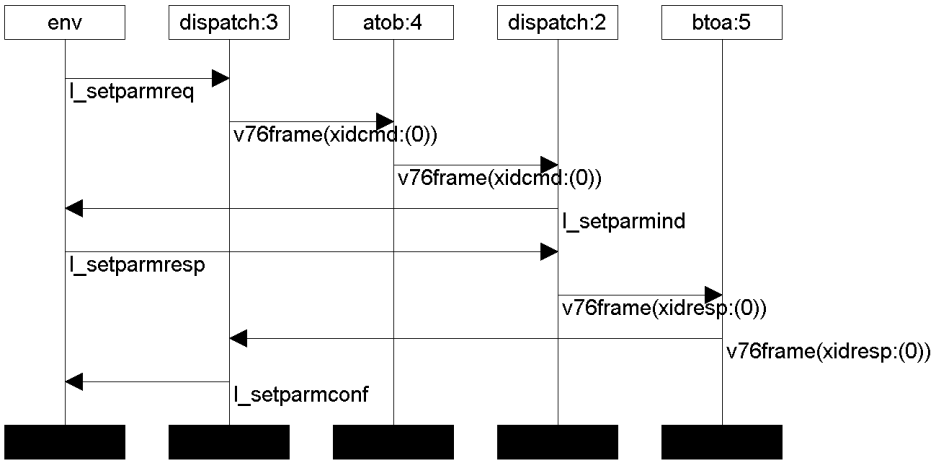
Diagram MSC (slika 4.47) prikazuje preverjanje prisotnosti, ki ni skladno s pričakovanji, saj bi moral proces `dispatch:2` iz bloka `DLCb`, signal `V76frame(xidresp:(0))` poslati procesu `btoa:5` in ne procesu `atob:4`. Ob pogledu na sliki 4.38 (prikazuje sistem `V76test`) in 4.48 (prikazuje blok `dataLink`) vidimo, da vodi kanal `DLCbDL` od bloka `DLCb`, do bloka `dataLink`. Kanal se v bloku nadaljuje v signalno pot `fromB`, ki vodi proti procesu `BtoA:toPeer`. Nobena povezave ne vodi od bloka `DLCb` do procesa `AtoB:toPeer`. Da bi preverili ali je mogoče poslati signal od bloka `DLCb`, procesu `BtoA:toPeer` smo izvedli vodeno simulacijo.



Slika 4.48: Blok `dataLink`

Za vsak poslan signal določimo kdo ga pošlje (`from`), kdo ga prejme (`to`) in po kateri poti naj se pošlje (`via`). Signali so si sledili v naslednjem vrstnem redu:

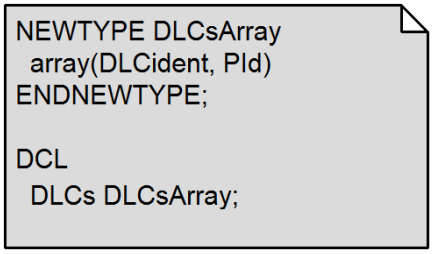
- `l_setparmreq from Env to dispatch:3 via dlcasu,`
- `v76frame (xidcmd:(0)) from dispatch:3 to atob:4 via dlcadl ,`
- `v76frame (xidcmd:(0)) from atob:4 to dispatch:2 via dlcbdl,`
- `l_setparmind from dispatch:2 to Env via dlcbsu,`
- `l_setparmresp from Env to dispatch:2 via dlcbsu,`
- `v76frame (xidresp:(0)) from dispatch:2 to btoa:5 via dlcadl,`
- `v76frame (xidresp:(0)) from btoa:5 to dispatch:3 via dlcadl,`
- `l_setparmconf from dispatch:3 to Env via dlcasu.`



Slika 4.49: Pravilen MSC preverjanja prisotnosti

Slika 4.49 prikazuje pravilen diagram MSC preverjanja prisotnosti.

Naslednji scenarij, ki smo ga želeli preveriti, je vzpostavitev povezave. Procesu `dispatch:3` iz bloka `DLCa` pošljemo signal `l_estabreq`, vrednost parametra `dlcident` nastavimo na 0. Simulator javi napako v drugem delu procesa `dispatch:3`, kjer po sprejetju signala `l_estabreq(DLCnum)` sledi odločitev `DLCs(DLCnum)` (slika 4.42). Simulator bi moral v tabeli preveriti ali je številka `DLCnum` prosta in v primeru da je ustvariti nov primerek procesa `DLC`. Namesto tega javi napako, da vrednost spremenljivke ni določena.



Slika 4.50: Deklaracija tabele `DLCs` tipa `DLCsArray`

Te napake nismo uspeli odpraviti. Predvidevamo, da je napaka v simulatorju, ker se obnaša enako pri uvoženem in zgrajenem sistemu. Oba sistema, pa v orodju ObjectGEODE delujeta pravilno. Pri preizkušanju simulatorja na manjših primerih smo dodatno ugotovili, da prioriteta signala pri sprejemu ni upoštevana.

Ostalih dveh scenarijev (prenos informacijskega okvirja in prekinitev povezave) nismo preverili, ker nismo zagotovili uspešne vzpostavitve povezave.

4.3.5 Komentar orodja

Orodje Cinderella deluje v okolju Windows. Orodje ima pregleden grafični vmesnik in omogoča hierarhičen prikaz specifikacije. Vsebuje pregleden priročnik za uporabo, za lažje razumevanje orodja vsebuje tri primere uporabe. Omogočeno je tudi tiskanje specifikacije.

Specifikacija se shrani v binarnem formatu s končnico ".cbf". Zato sistema ne moremo urejati z urejevalniki besedila. Orodje omogoča izvoz opisa sistema kot datoteko SDL PR, datoteko CIF ali datoteko STF. Če urejamo tekstualno predstavitev v urejevalniku besedila in jo nato uvozimo nazaj v orodje grafični prikaz ni več enak, je pa funkcionalno pravilen.

Orodje omogoča nekaj uporabniških nastavitev, npr. izbira barve simbolov, nivo analize in avtomatsko shranjevanje specifikacije. Pri gradnji sistema so na izbiro vsi gradniki, glede na sintaktična pravila pa smo omejeni pri njihovem povezovanju.

Omogočeno je uvažanje specifikacije iz drugih orodij. V našem primeru analizator ni javil napak, simulacija pa kljub temu ni delovala. Orodje omogoča vključitev orodij: "Cinderella Slipper", ki podpira generiranje in razhroščevanje kode C, "Cinderella Site", ki podpira generiranje kode C++ in "Cinderella MSC", ki omogoča generiranje testnih diagramov MSC in izvoz diagramov MSC v drug format (npr. Telelogic Tau) [5].

4.4 Safire Professional

4.4.1 Uvod

Podjetje Solinet iz Stuttgarta je prvo različico orodja Safire Professional izdalo leta 1994. Od takrat sodelujejo z največjimi svetovnimi proizvajalci testne opreme in ponudniki omrežne infrastrukture. Leta 2006 so jih prevzeli Northern Venture Ltd. s sedežem v Limassolu na Cipru. Prodajne, podporne in razvojne oddelke imajo v ZDA, Veliki Britaniji, Nemčiji, Indiji, Grčiji, Tajvanu in Siciliji [7].

Safire Professional je razvojno okolje optimizirano za implementacijo, validacijo in opazovanje signalizacijskih sistemov. Za opazovanje obnašanja signalizacijskih sistemov ima orodje vgrajen tudi protokolni analizator.

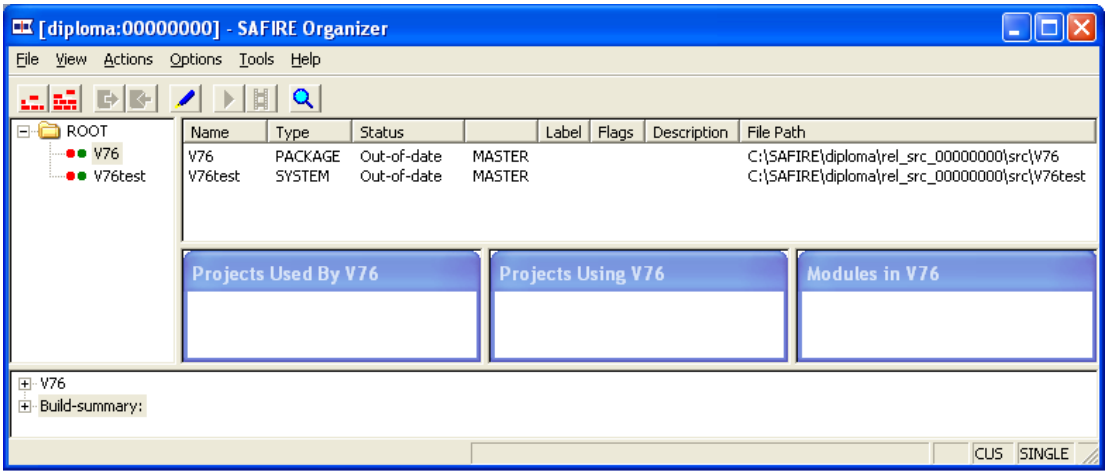
Orodje je zasnovano modularno in ima pregleden grafični vmesnik. V osnovnem oknu, "SAFIRE Organizer", so prikazani vsi sistemi, ki se nahajajo v izbrani mapi. S pomočjo modula "SAFIRE Editor" urejamo specifikacijo sistema. Postopek opisa specifikacije sistema je nekoliko drugačen, kot smo vajeni pri ostalih orodjih. Vsak konstrukt specifikacije dodamo s pomočjo padajočih menijev, kar je nekoliko zamudno opravilo. Povezave med konstrukti se tvorijo avtomatično. Enako velja za njihovo razporeditev po delovni površini. Orodje ne omogoča premikanje konstruktov.

S pomočjo modula "SAFIRE Organizer" lahko kodo urejamo tudi v tekstualnem načinu. Pri tem se uporabi sistemski urejevalnik teksta (npr. beležnica). Modul "SAFIRE Editor" omogoča pregled tekstualnega zapisa specifikacije, ne omogoča pa spreminjanja specifikacije.

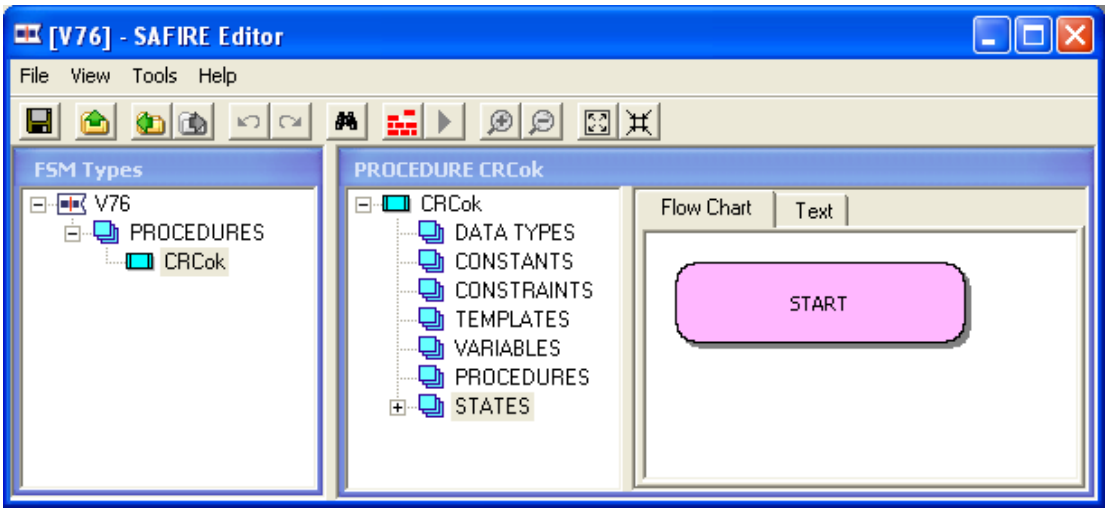
4.4.2 Grafična podoba

Delovno okolje orodje Safire Professional je sestavljeno iz več oken. V osnovnem oknu "SAFIRE Organizer" (slika 4.51), se nahaja mapa "ROOT" in vsi projekti, ki so shranjeni v tem arhivu. V oknu " SAFIRE Organizer" lahko dodajamo sisteme, pakete, knjižnice, protokole,... Na tem nivoju vidimo ob imenu projekta dve piki. Če sta obe zeleni, to pomeni, da je sistem pravilno zgrajen. V primeru, da je katera od njiju rdeča, pomeni, da je bil sistem spremenjen in še ni bil zgrajen ("build") ali pa da sistem vsebuje napake, ki se izpišejo na dnu okna. Za urejanje posameznega sistema (oz. paketa, knjižnice,…) izberemo opcijo "Edit". Odpre se nam okno "SAFIRE Editor" (slika 4.52).

Na levi strani je prikazana hierarhija sistema. V urejevalniku imamo vse urejeno po mapah, ki ločujejo uvožene pakete, podatkovne tipe, konstante, procedure, FSM, opis obnašanja bloka, idr. Načrtovanje sistema poteka drugače kot pri preostalih orodjih, saj nimamo orodne vrstice z gradniki. Te dodajamo z izbiro opcije vstavi ("insert"). Ob tem orodje prikaže gradnike, ki jih na izbranem mestu v sistemu lahko uporabimo. Orodna vrstica orodja ni prilagodljiva, saj nimamo vpliva na izbiro prikaznih elementov orodja. Orodje ne omogoča tiskanja specifikacije, vendar jo lahko natisnemo z zunanjim urejevalnikom besedila. Omogočeno je kopiranje diagramov, ki jih prav tako lahko natisnemo z uporabo zunanjega programa.



Slika 4.51: Pogled okna "SAFIRE Organizer"

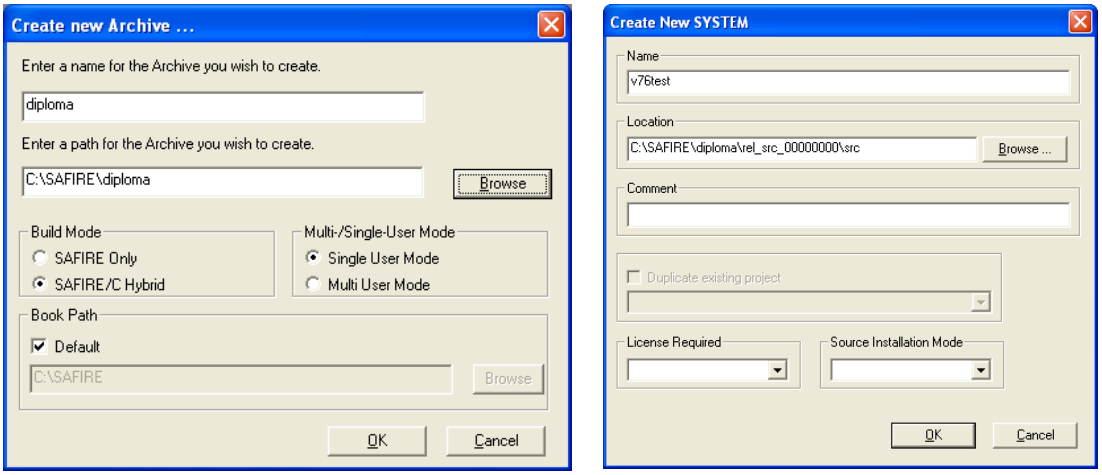


Slika 4.52: Pogled okna "SAFIRE Editor"

4.4.3 SDL opis protokola V.76

Opis protokola je povzet po knjigi [2]. Orodje Safire Professional ne omogoča uvažanja, zato smo protokol V.76 zgradili sami. V nadaljevanju je opisana gradnja sistema in morebitne omejitve programa. Uporabili smo različico programa v20.05.16.01.

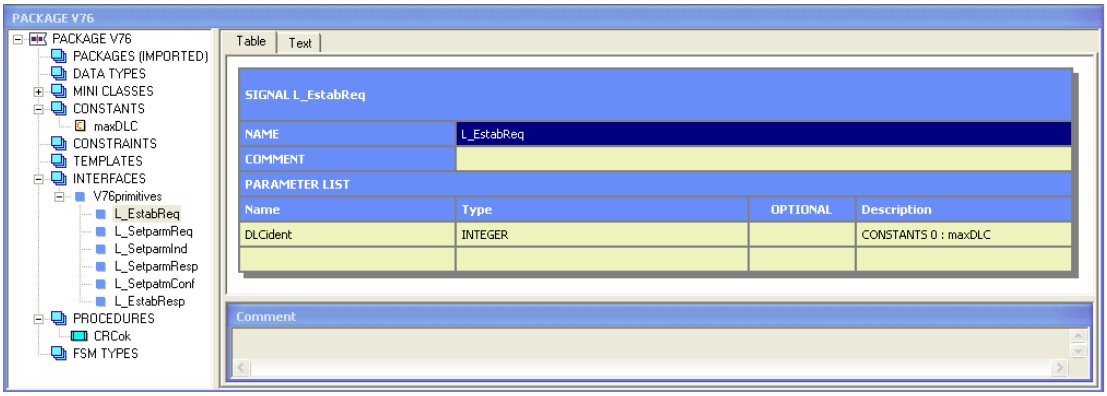
Najprej tvorimo arhiv, ki bo vseboval projekt in v katerem bo interna koda. Izberemo ime projekta in mapo, kjer ga želimo ustvariti (slika 4.53). V enem arhivu se lahko nahaja več projektov, ki jih združimo v mape.



Slika 4.53: Okni “Create new Archive” in “Create New SYSTEM”

V oknu “SAFIRE Organizer” (slika 4.51) imamo mapo “ROOT”, če odpremo obstoječi arhiv, se v mapi nahajajo tudi drugi projekti. V mapi “ROOT” izberemo “Create > System” (slika 4.53) in v okno vpišemo ime sistema (`V76test`). V oknu “SAFIRE Organizer” z izbiro novega sistema in opcije “Edit” odpremo “SAFIRE Editor” (slika 4.52). Na enak način, kot sistem, dodamo tudi paket `V76`. Paket `V76`, vsebuje definicije signalov, deklaracije podatkovnih tipov, proceduro `CRCok` in blokovni tip `V76_DLC` in je uvožen v sistem `V76test`.

Orodje Safire Professional za definicijo signalov in podatkovnih tipov ne uporablja definicijskega območja. Signale dodajamo v oknu “SAFIRE Editor”, v padajočem meniju pod “INTERFACES” izberemo “Insert > INTERFACE”, jo poimenujemo, nato lahko vpisujemo signale. Signal dodamo z izbiro “insert > SIGNAL”. Slika 4.54 prikazuje opis signala `L_EstabReq`, ki nosi parameter `DLCident`, tipa celo število (“INTEGER”). Parametre dodamo signalu z izbiro “insert > PARAMETER”.



Slika 4.54: Pogled definicije signala

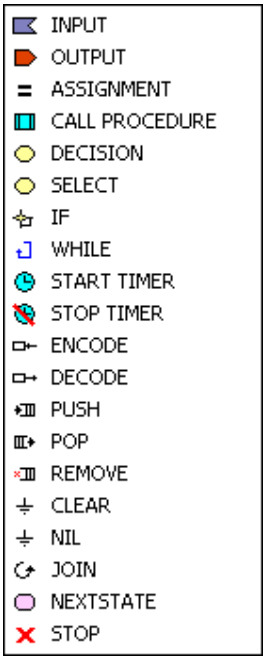
Orodje ne omogoča definicije uporabniških podatkovnih tipov `SYNTYPE`, `SYNONYM` in `NEWTYPE`. Podatkovne tipe, ki so bili definirani kot `SYNONYM` [2], definiramo kot konstante, tiste, ki so definirani kot `SYNTYPE`, pa definiramo v okviru parametrov signala (slika 4.54). Za podatkovni tip `NEWTYPE` nismo našli rešitve.

Orodje Safire Professional ne podpira blokov, temveč samo blokovne tipe. Prav tako ne podpira procesov, temveč le procesne tipe. Paket `V76` vsebuje blokovni tip `V76_DLC`. Dodamo ga z izbiro "insert > FSM Block Definition". Blokovni tip `V76_DLC` vsebuje dva procesa: `dispatch` in `DLC`. Procesni tip dodamo z izbiro "insert > FSM Definition". Vsak FSM tip potrebuje en primerek procesa, ki ga dodamo v padajočem meniju pod "BEHAVIOR BLOCK > FSM INSTANCES". To ni skladno z našim primerom, saj proces `DLC` ob zagonu sistema nima predvidenega nobenega primerka procesa, število primerkov pa je omejeno z `maxDLC+1`. Tega v orodju Safire Professional ni mogoče definirati.

Za povezovanje blokovnih in procesnih tipov orodje uporablja vrata, ki jih definiramo v padajočem meniju pod "GATES", povezave med njimi pa pod "BEHAVIOR BLOCK > CONNECTIONS". V pomoči [14] je navedeno, da morajo imeti vrata na obeh straneh povezave enak nabor signalov, kar pomeni, da so vse povezave dvosmerne in se vsi signali, ki so našteti lahko pošiljajo v obe smeri. Za naš sistem to pomeni, da moramo dvosmerne povezave, ki imajo za vsako smer različen nabor signalov razdeliti v dve povezavi z enakim naborom signalov.

Obnašanje sistema definiramo z diagrami prehajanja stanj. Naredimo ga za vsak FSM posebej. Gradnja diagrama prehajanja stanj, je v primerjavi z ostalimi predstavljenimi orodji zelo zamudna. Gradnike dodajamo v padajočem meniju pod "STATES" z izbiro

"insert" in izbiro ponujenih opcij (slika 4.55). Orodje nas omejuje pri izbiri naslednjega gradnika in nam ponudi samo tiste, ki so v dani situaciji dovoljeni. Ne omogoča premikanja simbolov. Vse kar dodamo, orodje samodejno razporedi in nariše povezave.



Slika 4.55: Nabor gradnikov v orodju Safire Professional

Med gradnjo sistema smo našli nekaj omejitev programa oz. odstopanj od standarda Z.100. Orodje ne podpira shranjevanja signala, namesto stanja zvezdica, uporablja "any state", namesto vhod zvezdica, uporablja "input any" in namesto naslednje stanje "-", uporablja "no state change". Nikjer v dokumentaciji niti v programu nismo zasledili makrov. Orodje ne podpira prioritetnega vhoda, namesto tega uporablja prioritetna vrata. V oknu "SAFIRE Editor" imamo možnost grafičnega in tekstovnega pogleda specifikacije, vendar lahko specifikacijo urejamo le v grafičnem delu. V oknu "SAFIRE Organizer" lahko izberemo opcijo "Edit as Text", kar nam omogoči urejanje tekstovnega zapisa z zunanjim urejevalnikom besedila. Vnesene spremembe se poznajo tudi v orodju Safire Professional. Orodje ne podpira komentarjev CIF. V oknu "Safire Product Browser" imamo možnost, da si iz njihove spletne strani naložimo nekaj primerov uporabe.

Velika pomanjkljivost orodja je tudi ta, da ne podpira uvažanja in izvažanja datotek v druga orodja. Program sicer shrani projekt v tekstovni datoteki s končnico ".fsm", vendar

ko smo poskusili projekt zgrajen v orodju Safire Professional uvoziti v ObjectGEODE, kot tekstovno datoteko, je le-ta javil sintaktične napake. Kar smo pričakovali, saj se tekstovni zapis orodja Safire Professional razlikuje od standarda Z.100.

Za izgradnjo celotnega sistema se nismo odločili, ker bi bile potrebne prevelike spremembe sistema. To se nam ni zdelo smiselno, saj je namen diplomskega dela predstaviti orodje in preveriti skladnost s standardom Z.100, ker se je tu pojavilo preveliko odstopanje gradnje nismo nadaljevali.

4.4.4 Komentar orodja

Orodje Safire Professional deluje v okolju Windows. Grafični vmesnik je pregleden, vsebuje pa zelo skop priročnik za uporabo. S spletne strani orodja, si je mogoče naložiti nekaj preprostih primerov uporabe.

Orodje podpira urejanje specifikacije samo v grafičnem načinu zapisa. Urejanje kode v tekstualnem zapisu, je mogoče le z zunanjim urejevalnikom besedila, omogočen pa je pregled tekstualnega zapisa specifikacije. Tekstovna datoteka se nahaja v arhivu v mapi "sdl" s končnico ".fsm". Orodje ne omogoča uvoza ali izvoza specifikacije. Poizkusili smo preimenovati datoteko ".fsm" v ".pr" in jo uvoziti v orodje ObjectGEODE, vendar to ni bilo mogoče, ker zapis specifikacije odstopa od standarda Z.100.

Pomanjkljivost orodja je, da ne omogoča tiskanja specifikacije in ne podpira diagramov MSC. Omogočeno je kopiranje diagramov, ki jih lahko natisnemo z uporabo zunanjega programa.

Uporabniške nastavitve niso podprte, orodna vrstica se ne da prilagajati, premikati ne moremo niti konstruktov po delovni površini.

5. SKLEP

Izbira primernega razvojnega orodja je za podjetje strateškega pomena. V diplomskem delu smo predstavili čim več lastnosti posameznega orodja, pri tem pa smo imeli v mislih potencialnega uporabnika orodja.

Vsa predstavljena orodja s podporo jeziku SDL so komercialna, uporabljali pa smo jih v okolju Windows. Ocenjujemo, da sta najprimernejši orodji SDL Suite in ObjectGEODE. Razvoj slednjega je bil ob prevzemu s strani podjetja Telelogic ustavljen. Orodji imata pregleden grafični vmesnik, podpirata uvoz in izvoz specifikacije ter vsebujeta pregleden priročnik za uporabo. Orodje Cinderella je manj napredno. Ima pregleden grafični vmesnik, je preprosto za uporabo ter podpira uvoz in izvoz specifikacije, kar smo preverili z orodjem ObjectGEODE. Obravnavani sistem smo uspeli zgraditi in uvoziti brez sintaktičnih napak. Pri simulaciji smo naleteli na težave, ki jih nismo uspeli odpraviti. Odzivnost tehnične podpore ocenjujemo kot dobro, saj so nam odgovorili v enem dnevu. Orodje Safire Professional po naši oceni za delo z jezikom SDL ni primerno, saj ni skladno s standardom Z.100. Razvoj orodja je aktiven. Odzivnost podpore je tudi zelo dobra.

Vsako izmed predstavljenih orodij ima svoje prednosti in slabosti. Izbira najprimernejšega orodja je odvisna tudi od zahtev uporabnika. Ocenjujemo, da je za zahtevnejše uporabnike najprimernejše orodje SDL Suite (primerno je tudi ObjectGEODE, vendar se ne trži več), za manj zahtevne, pa bi morda zadoščali orodji Cinderella in Safire Professional.

LITERATURA

[1] The International Engineering Consortium, *Specification and Description Language (SDL)*, [Spletni vir]. Naslov: http://www.iec.org/online/tutorials/sdl/index.html. [Dostopano: 12. 1. 2009].

[2] L. Doldi, *Validation of Communications Systems with SDL: The Art of Simulation and Reachability Analysis*, John Wiley & Sons, Ltd, 2003.

[3] ITU-T, *Recommendation Z.100 – Specification and Description Language (SDL)*, ITU-T, 2007.

[4] B. Vlaovič, *Generator klicev za telefonsko centralo MLB SI2000 V5*, diplomsko delo univerzitetnega študija. FERI, Maribor, 1999.

[5] Cinderella Software, [Spletni vir]. Naslov: http://www.cinderella.dk/. [Dostopano: 12. 1. 2009].

[6] ITU-T, *Recommendation Z.106 – Common interchange format for SDL*, ITU-T, 2002.

[7] Safire World, [Spletni vir]. Naslov: http://www.safire-world.com/. [Dostopano: 12. 1. 2009].

[8] Telelogic, [Spletni vir]. Naslov: http://www.telelogic.com/index.cfm?. [Dostopano: 12. 6. 2008].

[9] B. Vlaovič, *Avtomatska tvorba modelov s sondami iz specifikacije sistema v jeziku SDL*, doktorska disertacija, Maribor 2004.

[10] ITU-T, *Recommendation V.76 – Generic muliplexer using V.42 LAPM – based procedures*, ITU-T, 1996.

[11] L. Doldi, [Spletni vir]. Naslov: ftp://ftp.wiley.co.uk/pub/books/ldoldi/. [Dostopano: 10. 12. 2008].

[12] A. Kokol, *Študija prenosljivosti SDL specifikacije med izbranimi orodji*, diplomsko delo univerzitetnega študija, FERI, Maribor, 2009.

[13] Cinderella Software, *Cinderella Help*.

[14] Safire World, *Safire Editor Userman.*

ITU-T, *Recommendation Z.120 – Message Sequence Chart (MSC)*, ITU-T, 1999.

PRILOGE

A Naslov študenta

Klara Podbreznik
Novi trg 11
3000 Celje
e-pošta: klara9@gmail.com

B Kratek življenjepis

Rojena:	18. 9. 1983	v Celju
Šolanje:	1990 – 1998	Osnovna šola Primoža Trubarja Laško Trubarjeva ulica 20, 3270 Laško
	1998 – 2002	Gimnazija Celje - Center Kosovelova ulica 1, 3000 Celje
	2002 – 2009	Univerzitetni študijski program Telekomunikacije, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko Smetanova ulica 17, 2000 Maribor

C Izjava o istovetnosti tiskane in elektronske verzije diplomskega dela

UNIVERZA V MARIBORU

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV AVTORJA

Ime in priimek avtorja:	Klara Podbreznik
Vpisna številka:	93511718
Študijski program:	Telekomunikacije
Naslov zaključnega dela:	Primerjava izbranih razvojnih orodij s podporo jeziku SDL
Mentor:	doc. dr. Boštjan Vlaovič
Somentor:	asist. dr. Aleksander Vreže

Podpisana Klara Podbreznik izjavljam, da sem za potrebe arhiviranja oddal-a elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna elektronski verziji, ki sem jo oddal-a za objavo v Digitalno knjižnico Univerze v Mariboru. Podpisani-a izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zagovora, naslov zaključnega dela) na spletnih straneh in v publikacijah UM.

Kraj in datum: Podpis avtorja:

Maribor, 2. 4. 2009 _____