Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Darko Božić

Analiza in zgled uporabe programskega orodja "CPN Tools" za postavljanje modelov dinamičnih sistemov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

izr. prof. dr. Miha Mraz
MENTOR

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01861/2012
Datum: 04.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DARKO BOŽIĆ**

Naslov: **ANALIZA IN ZGLED UPORABE PROGRAMSKEGA ORODJA »CPN TOOLS« ZA POSTAVLJANJE MODELOV DINAMIČNIH SISTEMOV**

DYNAMICAL SYSTEMS MODELLING WITH CPN TOOLS AND COLOURED PETRI NETS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kandidat naj v svojem delu predstavi delovanje javno razpoložljivega programskega orodja »CPN Tools«, ki je namenjeno modeliranju dinamičnih sistemov na osnovi barvnih Petrijevih mrež. Pri tem naj kandidat oriše pomen barvnih Petrijevih mrež, osnove uporabe orodja »CPN Tools«, v nadaljevanju pa zgradi model komunikacijskega protokola »drsečega okna« in predstavi simulacijske rezultate modela navedenega protokola.

Mentor:

prof. dr. Miha Mraz

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Darko Božić, z vpisno številko ***63060044****, sem avtor diplomskega dela z naslovom:*

Analiza in zgled uporabe programskega orodja "CPN Tools" za postavljanje modelov dinamičnih sistemov

S svojim podpisom zagotavljam, da:

- *sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Mihe Mraza,*
- *so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela*
- *soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".*

Podpis avtorja:

— Darko Božić, Ljubljana, september 2012.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Darko Božić

Analiza in zgled uporabe programskega orodja "CPN Tools" za postavljanje modelov dinamičnih sistemov

POVZETEK

Namen diplomske naloge je pokazati, kako se s pomočjo računalniškega orodja *CPN Tools* in grafičnega jezika barvnih Petrijevih mrež izdela model, ki predstavlja nek načrtovani sistem. Barvne Petrijeve mreže so nadgradnja Petrijevih mrež na ta način, da je Petrijevim mrežam dodana še funkcionalnost programskega jezika *CPN ML*. Ker so barvne Petrijeve mreže predvsem grafični jezik s pridihom programiranja, je izdelava modelov predvsem odvisna od računalniškega orodja *CPN Tools*. Orodje *CPN Tools* nam tako omogoča izdelavo načrtovanega sistema v obliki modela, kjer lahko izdelani model simuliramo in preverjamo njegove lastnosti. Z izvajanjem simulacije lahko pri snovanju novih sistemov odkrijemo napake, po odpravi napak pa lahko preverimo kakšne so zmogljivosti našega načrtovanega sistema.

Da bi lahko preverili dejansko uporabnost orodja in njegovo zmogljivost, smo za zgled uporabili že dobro znani komunikacijski protokol drsečega okna. Na ta način lahko preverimo ali orodje zazna tipične napake pri izdelavi modela, ki so za protokol značilne in načine kako napako v modelu odkriti. Prav tako lahko preverimo zmogljivost simulatorja, ki nam lahko ponudi informacije o zmogljivosti našega izdelanega modela. Pridobljene rezultate pa lahko nato ustrezno primerjamo, saj za modelirani protokol že poznamo pričakovane rezultate.

Ključne besede: Petrijeve mreže, barvne Petrijeve mreže, modeliranje, CPN Tools.

University of Ljubljana
Faculty of Computer and Information Science

Darko Božić

Dynamical systems modelling with CPN Tools and coloured Petri nets

ABSTRACT

The purpose of this thesis is to show how we can create a model that represents a planned system using the software tool "CPN Tools" and the graphical language of coloured Petri nets. Coloured Petri nets are updated Petri nets, in the sense that the functionality of the programming language "CPN ML" has been added to them. Since the coloured Petri nets are mainly a graphical language with a touch of programming, creating models mainly depends on the software "CPN Tools". "CPN Tools" thus allows us to make a model of the planned system where we can simulate and check its properties. While designing the system, we can detect errors, correct them and check the capacities of our planned system by performing simulations.

In order to verify the actual usability of the tool and its capacity, we used a well known communication protocol sliding window as an example. That way we can check if the tool could detect typical errors in the model which were protocol-specific and eventually find a way to detect errors in the model. We can also check the performance of the simulator which gives us information about the performance of the model that we have developed. Acquired results can then be compared, as we already know the expected results for the modeled protocol.

Key words: Petri nets, coloured Petri nets, modelling, CPN Tools.

ZAHVALA

Zahvaljujem se mentorju izr. prof. dr. Mihi Mrazu za sprejeto mentorstvo in vse nasvete pri izdelavi diplomskega dela.
Hvala Sanji Savatić in Petri Prusnik za pomoč pri prevodu povzetka v angleški jezik.
Posebna zahvala pa gre družini, ter dolgoletnemu prijatelju Janu Kocjančiču za vso izkazano podporo v času mojega šolanja.

Vive y aprende.

— Darko Božić, Ljubljana, september 2012.

KAZALO

1 Uvod

1.1 Motivacija

Razvoj sistemov v današnjem času predstavlja zahtevno opravilo. Posebej težak je razvoj velikih sistemov, ki so ponavadi sestavljeni iz množice manjših sistemov, kjer vsak manjši sistem opravlja svojo točno določeno nalogo. Zato se lahko v takih velikih sistemih, opravila izvajajo istočasno in tudi ne-deterministično. Tak način izvajanja opravil pa za razvijalce predstavlja problem, kajti pri razvoju lahko razvijalec hitro spregleda in izpusti ključne elemente pri zasnovi sistema. Če se napake nato implementirajo, ne da bi se prej odpravile, sistem ne bo deloval pravilno. Odprava napak na sistemih, ki že delujejo, pa je lahko časovno in denarno potratna ali pa v najslabšem primeru odprava napake ni mogoča zaradi same zasnove sistema. Da bi se izognili težavam pri implementaciji sistema je potrebno razvijalcem ponuditi rešitve, kjer se lahko na nek način že v fazi razvoja sistem testira in odkrije možne napake.

Barvne Petrijeve mreže [3, 4] predstavljajo grafični jezik, ki razvijalcu omogočajo predstavitev sistema kot model, le tega pa se lahko testira in analizira. So razširitev navadnih Petrijevih mrež na ta način, da jim je dodana funkcionalnost programskega

jezika CPN ML, ki bazira na programskem jeziku Standard ML. Barvne Petrijeve mreže so zasnovane tako, da se osredotočajo na izdelavo različnih, ne samo določenih sistemov. Zato je njihova uporaba možna ne samo na računalniškem področju, ampak tudi na poslovnem področju, na področju izvajanja kemijskih in bioloških procesov, itn.

Modeli predstavljeni z barvnimi Petrijevimi mrežami, lahko prikazujejo stanje v sistemu. Stanj, ki jih lahko model zavzame, je mnogo. Vsako stanje pa je posledica nekega zaporedja akcij, ki so se izvedle v modelu. Izvajanje akcij lahko ustvarjalec nadzoruje s pomočjo simulacije, ki se lahko izvaja interaktivno (ustvarjalec ima nadzor nad simulacijo), ali pa samodejno (ustvarjalec določi kriterije za izvajanje simulacije). Ker so lahko modeli nemalokrat veliki, se lahko model predstavi hierarhično tako, da se ga razdeli na posamezne module. Podprta sta tako "top-down" kot "bottom-up" pristop izdelave modela. V določenih sistemih je lahko ključnega pomena tudi čas izvajanja dogodkov. Barvne Petrijeve mreže zato omogočajo tudi vpeljavo časa v model. Čas ponuja razvijalcu možnost, da preveri, ali se model oz. posamezni moduli izvedejo v pričakovanem času, ali pa se model izvaja prepočasi in s tem povzroča kakšne nepričakovane zastoje v simuliranem modelu. V primeru, da se pri izvajanju modela opazi nepravilno delovanje, nam barvne Petrijeve mreže omogočajo tudi analizo prostora stanj. Z analizo prostora stanj se v modelu izračunajo vsa možna stanja, ki jih lahko nek model zavzame. Stanja se nato predstavijo z usmerjenim grafom, kjer je v vozliščih grafa predstavljeno točno določeno stanje v modelu, puščice pa prikazujejo akcije, ki so model pripeljale v to stanje. Na ta način lahko razvijalec pokaže, da sistem deluje pravilno, oz. pokaže, da sistem ne more priti v neka nedovoljena stanja, ki bi lahko pomenila napako v sistemu.

Zaradi kompleksnosti barvnih Petrijevih mrež je praktična izdelava modelov in izvajanje simulacije popolnoma odvisna od računalniškega orodja imenovanega *CPN Tools*[1]. *CPN Tools* je računalniško orodje za urejanje, simuliranje, izvajanje analize prostora stanj in izvajanje performančne analize ustvarjenega modela z barvnimi Petrijevimi mrežami. Orodje je sestavljeno iz treh komponent tj. grafičnega vmesnika, simulatorja in orodja za interakcijo s simulatorjem imenovanega *Access/CPN Framework*[2].

[1] http://cpntools.org
[2] http://cpntools.org/accesscpn/start

1.2 Metodologije

V pričujočem delu so definirane Petrijeve in barvne Petrijeve mreže. Narejena je tudi podrobnejša analiza formata projekta, ki je ustvarjen in shranjen z orodjem *CPN Tools*. Prav tako je prikazan način uporabe orodja in primer izdelave modela protokola drsečega okna. Opravljeni so bili tudi različni poskusi izvajanja simulacije nad ustvarjenim modelom z namenom, da se preveri ali se ustvarjeni model obnaša po pričakovanjih.

1.3 Pregled diplomskega dela

V delu je prikazana uporabnost barvnih Petrijevih mrež in orodja *CPN Tools* na primeru izdelave modela, ki predstavlja komunikacijski protokol imenovan drseče okno. V drugem poglavju je najprej obravnavana teorija o Petrijevih mrežah, na katere se barvne Petrijeve mreže oslanjajo. Prikazani so osnovni gradniki Petrijeve mreže, kako se Petrijeva mreža izvaja, ter kako se Petrijevo mrežo predstavi z grafom. Vsebina tretjega poglavja je namenjena predstavitvi barvne Petrijeve mreže. S pomočjo preprostega primera so prikazani gradniki in izvajanje mreže. Prikazana je tudi modularna izdelava modela in vpeljava časa v model. Četrto poglavje nas popelje skozi orodje *CPN Tools*. V prvem delu poglavja je prikazano, kako se uporabljajo razna orodja za izdelavo modela, ter v kakšnem formatu se izdelani projekt shrani. V drugem delu pa je prikazana uporaba simulatorja, ter izvedba analize prostora stanj in performačne analize za ustvarjeni model. Peto poglavje obsega predstavitev in izdelavo modela protokola drseče okno. Najprej je predstavljen problem drsečega okna in kako protokol deluje. Nato sledi podrobnejši opis izdelave in delovanja modela. Na koncu poglavja pa so prikazani simulacijski rezultati izdelanega modela z barvnimi Petrijevimi mrežami.

2 Petrijeve mreže

Petrijeve mreže je uvedel Carl A. Petri leta 1969 in so orodje za modeliranje, analizo in načrtovanje diskretnih sistemov. Gre za matematično orodje, ki se uporablja za modeliranje komunikacijskih protokolov, programske opreme, strojne opreme, kemijskih reakcij, itd.

2.1 Definicija Petrijevih mrež

Petrijeva mreža je urejen četverček $C = (P,T,I,O)$, kjer je [1]

- $P = \{p_1, p_2, \ldots, p_n\}, n \geq 0$, končna množica pogojev (angl. *places*),
- $T = \{t_1, t_2, \ldots, t_m\}, m \geq 0$, končna množica akcij (angl. *transitions*),
- množici P in T sta si tuji, $P \cap T = \emptyset$,
- I je vhodna funkcija in določa množico pogojev, iz katerih vstopajo povezave v posamezne akcije,

- O je izhodna funkcija in določa množico pogojev, v katere vstopajo povezave iz posameznih akcij.

Če je število pogojev n in število akcij m, funkciji I in O lahko predstavimo kot matriki reda $m \times n$. V množicah $I(t_i)$ in $O(t_i)$ se lahko določen pogoj pojavi večkrat, zato imamo opravka s posplošenimi množicami. Usmerjene povezave povezujejo samo akcije s pogoji in obratno, medtem ko povezava med dvema akcijama, ali dvema pogojema, ni mogoča.

Število vhodnih pogojev p_i za akcijo t_j je število elementov p_i v posplošeni množici $I(t_j)$

$$\#(p_i, I(t_j)),\ p_i \in I(t_j). \tag{2.1}$$

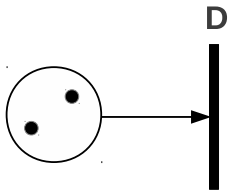
Število izhodnih pogojev p_i za akcijo t_j je število elementov p_i v posplošeni množici $O(t_j)$

$$\#(p_i, O(t_j)),\ p_i \in O(t_j). \tag{2.2}$$

2.2 Graf Petrijeve mreže

Petrijevo mrežo si lahko predstavljamo kot graf, ki po [1, 2] vsebuje

- **pogoje** (narisane s krogom), le ti pa lahko vsebujejo žetone (črne pike),
- **akcije** (narisane kot kvadrat ali črta), ki so označene s časom D in zadržijo izhod žetonov za določen čas D; če je $D = 0$, je akcija takojšnja (hipna), v nasprotnem primeru ($D > 0$) pa je akcija zakasnjena,
- **usmerjene povezave** (puščica), ki predstavljajo vhodne in izhodne funkcije,
- **žetone** (narisane s črno piko), ki označujejo Petrijevo mrežo.



Slika 2.1: Pogoj z dvema žetonoma, povezavo in akcijo.

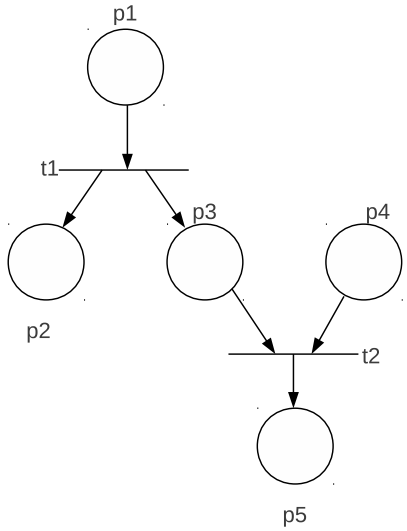
Def.: Petrijev graf G je po [1] bipartitni (dvodelni) usmerjeni multigraf $G = (V, A)$, kjer je $V = \{v_1, v_2, ..., v_s\}$, $V = P \cup T$ množica spojišč in $A = \{a_1, a_2, ..., a_r\}$ posplošena

množica usmerjenih povezav $a_i = (v_j, v_k) \in V$, kjer je $(v_j \in P)$ in $(v_k \in T)$ ali $(v_j \in T)$ in $(v_k \in P)$.

2.3 Primer Petrijeve mreže

Primer formalnega zapisa Petrijeve mreže sledi v nadaljevanju, njegova grafična predstavitev je prikazana na sliki 2.2:

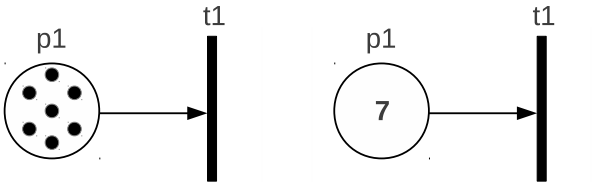
- $C = (P, T, I, O)$,
- $P = (p_1, p_2, p_3, p_4, p_5),\ T = (t_1, t_2)$
- $I(t_1) = p_1, \quad I(t_2) = p_3, p_4$
- $O(t_1) = p_2, p_3, \quad O(t_2) = p_5$



Slika 2.2: Primer Petrijeve mreže predstavljene s Petrijevim grafom.

2.4 Označevanje v Petrijevi mreži

Označevanje je dodeljevanje žetonov posameznim pogojem v mreži [1]. Število žetonov v posameznem pogoju nam pove, kolikokrat je pogoj izpolnjen. Tako se pri izvajanju



(a) Preveliko število žetonov. (b) Zamenjava žetonov s številom žetonov.

Slika 2.3: Problem velikega števila žetonov.

Petrijeve mreže število žetonov v posameznih pogojih lahko spreminja. Za označevanje v mreži skrbi funkcija o, ki pogojem P prireja pozitivna cela števila N

$$o : P \to N. \tag{2.3}$$

Označevanje lahko predstavimo s časovno odvisnim vektorjem

$$\mathbf{o}(t) = (o_1(t), o_2(t), \ldots, o_n(t)),\ n = |P|. \tag{2.4}$$

Označeno Petrijevo mrežo zapišemo kot $M = (P,T,I,O,\boldsymbol{o(t)})$. Označitev $\mathbf{o}(t)$ enačimo s stanjem Petrijeve mreže v času t.

V Petrijevem grafu so žetoni ponazorjeni s pikami. Če je število pik preveliko, jih lahko nadomestimo s številom, ki ponazarja število žetonov (glej sliko 2.3).

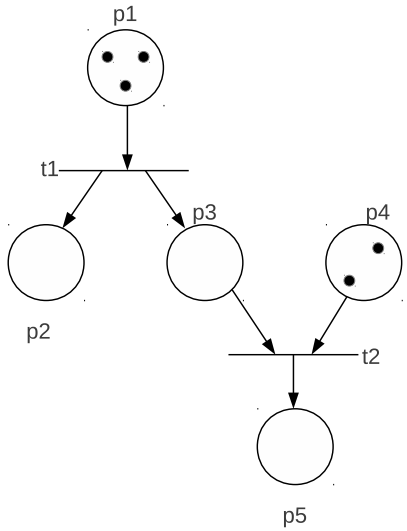
2.5 Izvajanje akcij v Petrijevi mreži

Izvajanje je pogojeno z označitvijo mreže $\mathbf{o}$(t), ki se s časom spreminja [1]. Izvajanje mreže se izvede z vžigom izbrane akcije t_j. Akcija $t_j \in T$ je izbrana, če lahko vsaka povezava, ki vstopa v akcijo zagotovi po en žeton iz vsakega pogoja $p_i \in P$

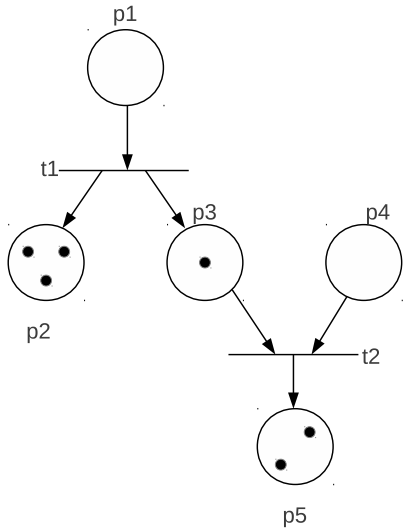
$$o(p_i) \geq \#(p_i, I(t_j)),\ o(t) \geq e(j) * I. \tag{2.5}$$

$e(j)$ predstavlja enotni vektor, ki ima vrednost 1 samo na mestu j. Vžig izbrane akcije t_j povzroči, da se iz vhodnih pogojev po en žeton odvzame ter se na izhodne pogoje po en žeton doda. Z vžigom izbrane akcije t_j dobimo novo označitev

$$o'(p_i) = o(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j)),\ o(t+1) = o(t) + e(j) * (O - I). \tag{2.6}$$



Slika 2.4: Začetna označitev v Petrijevi mreži.



Slika 2.5: Končna označitev Petrijeve mreže s slike 2.4.

V primeru s slike 2.4 je začetna označitev $\mathbf{o}(t_0) = (3, 0, 0, 2, 0)$. Tako označena mreža izpolnjuje pogoj za izvedbo akcije t_1, medtem ko pogoj za izvedbo akcije t_2 ni izpolnjen. Po izvedbi akcije t_1 dobimo novo označitev $\mathbf{o}(t_1) = (2, 1, 1, 2, 0)$. Taka označitev sedaj

omogoča, da se lahko izvaja še akcija t_2. Po končanem izvajanju mreže s slike 2.4, bi dobili končno označitev $\mathbf{o}(t_5) = (0, 3, 1, 0, 2)$ prikazano na sliki 2.5.

3 Barvne Petrijeve mreže

Načrtovanje velikih sistemov, v industriji ali poslovnih okoljih, je lahko zelo zahtevno. Ponavadi se v takih sistemih odvija veliko stvari hkrati, so tudi ne-deterministični (izvajajo se lahko na mnogo načinov), kar privede do težkega testiranja in odpravljanja napak v zasnovi sistema. Zato lahko z barvnimi Petrijevimi mrežami postavimo model, preverimo njegovo delovanje in odpravimo napake še preden sistem začnemo uporabljati v realnem okolju, ali preden ga začnemo graditi.

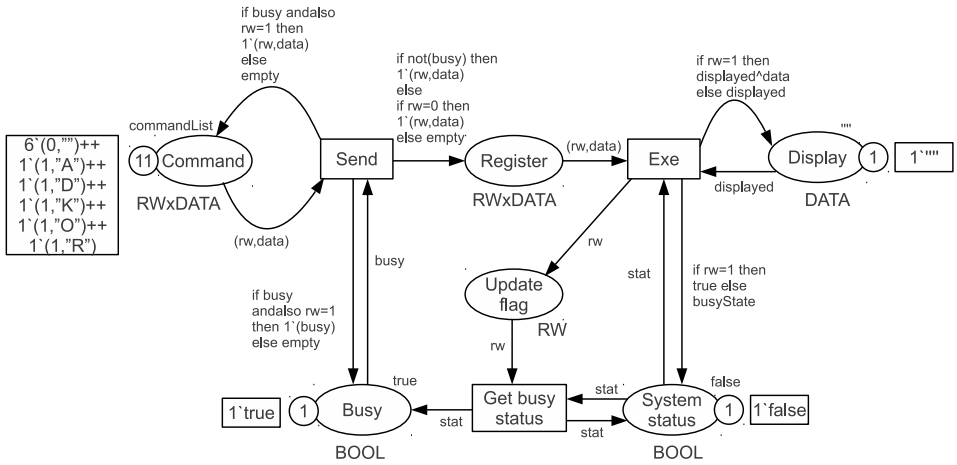
3.1 Uvod

Barvne Petrijeve mreže (v nadaljevanju CPN, angl. *coloured Petri nets*) so grafični jezik za modeliranje diskretnih sistemov in njihovo analizo. So kombinacija Petrijevih mrež in programskega jezika CPN ML, ki temelji na programskem jeziku Standard ML[1]. Petrijeve mreže zagotavljajo grafični zapis in osnovne primitive za modeliranje sočasnosti, komunikacije in sinhronizacije. S pomočjo programskega jezika Standard ML pridobimo še primitive za definicijo podatkovnih tipov in njihovo manipulacijo. CPN omogočajo

[1] http://www.smlnj.org

tudi modularno izdelavo in vpeljavo časa v model. Modularna izdelava omogoča izdelavo velikih in kompleksnih sistemov, časovni koncept pa ponazarja čas, ki je potreben za izvajanje posameznih dogodkov (akcij). Tipično se CPN uporabljajo na področjih komunikacijskih protokolov, avdio in video sistemov, operacijskih sistemov, izdelave strojne opreme, programske opreme in poslovnih procesov.

3.2 Struktura CPN



Slika 3.1: Primer barvne Petrijeve mreže.

CPN je vedno predstavljena v grafični obliki. Slika 3.1 predstavlja preprost primer pošiljanja znakov na LCD prikazovalnik. Na levi strani se nahaja pošiljatelj, na desni strani pa LCD prikazovalnik. Pošiljatelj lahko pošilja dva tipa ukazov. Prvi ukaz je ukaz za zapis znaka na prikazovalnik, z drugim ukazom pa se preverja stanje prikazovalnika. Prikazovalnik za zapis znaka potrebuje nekaj časa in v tem času ne more sprejeti novega znaka. Zato mora pošiljatelj z ukazom za preverjanje stanja preveriti, ali je prikazovalnik pripravljen na sprejem novega znaka. Pogoj *Command* predstavlja pošiljatelja in ukaze, ki jih želi pošiljatelj poslati prikazovalniku. Prejeti in izpisani znaki se zapisujejo na prikazovalnik, ki je predstavljen s pogojem *Display*. Logika prikazovalnika dekodira ukaze, ki jih pošilja pošiljatelj in se zapisujejo v register *Register*. *System status* odraža stanje prikazovalnika, tj. prikazovalnik je zaseden, ali pa pripravljen na nov sprejem znaka. Stanje prikazovalnika lahko uporabnik preveri preko signala *Busy*. Primer na

sliki 3.1 vsebuje 6 pogojev (elipsa ali krog), 3 akcije (pravokotnik), usmerjene puščice, ki povezujejo pogoje in akcije ter razne napise poleg pogojev, akcij in puščic. Napisi so zapisani v programskem jeziku CPN ML [3, 4].

3.2.1 Pogoji v CPN

Pogoji odražajo stanje v CPN [3, 4]. Vsak pogoj lahko vsebuje nič, enega ali več žetonov. Vsak žeton nosi neke vnaprej definirane podatkovne vrednosti, ki jim pravimo barva žetona. Število žetonov in barva žetona v posameznem pogoju skupaj predstavljajo stanje sistema, ki mu pravimo tudi označitev CPN. Imena pogojev praviloma zapisujemo znotraj elips ali krogov. Sam zapis v pogoju nima nekega formalnega pomena, ampak pomaga pri berljivosti mreže.

Poleg vsakega pogoja je napis, ki določa kakšne barve je lahko žeton v tem pogoju. Barva žetona v posameznem pogoju je določena z *barvnim nizom* (angl. *colour set*) in se zapisuje pod posameznim pogojem. Pogoj *Display* na primeru s slike 3.1 nosi barvni niz *DATA*. Barvni nizi se po CPN ML definirajo z besedo *colset*. Izvorna koda 3.1 prikazuje definicijo barvnih nizov v primeru s slike 3.1.

Izvorna koda 3.1: Definicija barvnih nizov v primeru s slike 3.1.

```
colset BOOL = bool;
colset DATA = string;
colset RW = int with 0..1;
colset RWxDATA = product RW*DATA;
```

Vsi žetoni v pogoju *Display* imajo barvo tipa *string*. Podobno je z žetoni v pogojema *Busy* in *System status*, ki imajo barvo tipa *bool*. Pogoja *Command* in *Register* imata barvni niz *RWxDATA*, ki je produkt tipov *RW* in *DATA*. Barvni niz predstavlja vse pare, kjer je prvi element tipa *int* in drugi element tipa *string*. Elemente zapisujemo znotraj oklepajev ter so med seboj ločeni z vejico. Barvni niz tipa *RW* je omejen samo na števili 0 in 1. Število 0 predstavlja ukaz za branje stanja naprave, medtem ko 1 predstavlja ukaz za zapis znaka na prikazovalnik. Ukazu za branje ali pisanje sledi še črka, katero želimo izpisati na zaslon. Par (1,"A") na sliki 3.1 pomeni, da želimo na zaslon izpisati črko *A*, medtem ko par (0,"") pomeni, da želimo preveriti stanje naprave.

Naslednji napis določa *začetno označitev* pogoja. Napis se nahaja nad posameznim pogojem. Pogoj *Display* ima začetno označitev zapisano desno zgoraj in vsebuje en

žeton z barvo "" (prazen tekst), pogoj *Busy* pa vsebuje en žeton z barvo *true*. Napis *commandList* je konstanta, ki pogoju *Command* določa začetno označitev z enajstimi žetoni. Definicija konstante, za pogoj *Command*, je prikazana v kodi 3.2.

Izvorna koda 3.2: Definicija konstante *commandList* v primeru s slike 3.1.

```
val commandList = 6‘(0,"")++1‘(1,"A")++1‘(1,"D")++
1‘(1,"K")++1‘(1,"O")++1‘(1,"R");
```

Operatorja ‘ in ++ omogočata gradnjo posplošene množice (glej poglavje 2) z barvo žetonov. Posplošeno množico v CPN mreži lahko zapišemo kot [5]:

$$x_1‘v_1 + +x_2‘v_2 + + \ldots + +x_n‘v_n, \tag{3.1}$$

kjer je v_1 vrednost žetona in x_1 število pojavitev žetona. Operator ++ za argumenta sprejme dve posplošeni množici in vrne njuno unijo. V primeru, da napis ni določen, potem pogoj v začetni označitvi ne vsebuje nobenega žetona.

Trenutna označitev je vidna poleg vsakega pogoja. Sestavljena je iz majhnega kroga, kjer je zapisano trenutno število žetonov in kvadrata, kjer je zapisana barva posameznih žetonov. Trenutna in začetna označitev sta na začetku enaki.

3.2.2 Akcije in povezave v CPN

Akcije predstavljajo dogodke, ki se lahko zgodijo v sistemu [3, 4]. Narisane so kot kvadrat, znotraj katerega zapišemo ime akcije. Ime samo, enako kot pri pogojih, nima nobenega formalnega pomena, ampak pomaga pri berljivosti narisanega sistema. Vsakič, ko se akcija sproži, se iz vhodnih pogojev odstranijo žetoni in se na vse izhodne pogoje dodajo žetoni, kar povzroči tudi spremembo označitve sistema. Vhodni pogoji so tisti pogoji, iz katerih povezave vodijo v opazovano akcijo. Izhodni pogoji pa so tisti pogoji, v katere vodijo povezave iz opazovane akcije. Točno število žetonov in barvo žetonov, ki se odstranjujejo ali dodajajo, določajo *izrazi ob povezavah*. Izrazi so napisani v CPN ML programskem jeziku in lahko vsebujejo spremenljivke, konstante, operatorje in funkcije. Izraz se lahko oceni šele takrat, ko imajo vse spremenljivke vezane ustrezne vrednosti. Spremenljivke se v CPN definira kot

Izvorna koda 3.3: Definicija spremenljivke.

```
var vName : vType;
```

kjer je *vName* ime spremenljivke in *vType* podatkovni tip spremenljivke. Spremenljivke za akcijo *Send* iz slike 3.1 so definirane na sledeč način:

Izvorna koda 3.4: Definicija spremenljivk v primeru s slike 3.1.

```
var rw:RW;
var data : DATA;
var busy : BOOL;
var busyState : BOOL;
```

Tako so za spremenljivko *rw* celoštevilske vrednosti tipa *int* med 0 in 1, *data* tekstovne vrednosti tipa *string* in *busy* logične vrednosti tipa *bool* (*true* ali f*alse*). Da bi se akcija lahko sprožila, mora biti ta najprej omogočena. Akcija je omogočena takrat, ko je zagotovljena ustrezna vezava. Vezava za akcijo se po [4] zapiše kot

$$\langle v_1 = d_1, v_2 = d_2, \ldots, v_n = d_n \rangle \, , i \in 1 \ldots n, \tag{3.2}$$

kjer je v_i spremenljivka in d_i ustrezna vrednost vezana na spremenljivko.

Poleg izrazov ob puščicah lahko akcijo opremimo še s stražarjem. *Stražar* je logični izraz, ki omogoča sprejemanje samo tistih vezav, za katere se logični izraz oceni kot resničen. Zapiše se ga nad akcijo med oglatimi oklepaji "[" in "]".

3.3 Izvajanje akcij v CPN

Izvajanje akcij v CPN je pogojeno z izrazi ob povezavah in barvo žetonov v vhodnih pogojih, ki skupaj določajo, ali je akcija omogočena [3, 4]. Akcija je omogočena takrat, ko lahko naredimo ustrezno vezavo, tj. vsaki spremenljivki v izrazih ob akciji pripišemo neko ustrezno vrednost. Ko se akcija z ustrezno vezavo sproži, se iz vhodnih pogojev preko vhodnih povezav odstranijo žetoni z ustrezno barvo in se v izhodne pogoje preko izhodnih povezav dodajo žetoni z ustrezno barvo.

Izvajanje nekega CPN modela se v splošnem opisuje s pojavljanjem zaporedij, ki predstavljajo dosežene označitve ter korake, ki se pojavijo. Korak je sestavljen iz hkratne pojavitve enega ali več veznih elementov. Vezni element je par, ki je sestavljen iz akcije in ustrezne vezave.

V primeru s slike 3.1 vidimo, da je v začetni označitvi M_0 omogočena samo akcija

Send. Akcija *Send* je omogočena z naslednjimi veznimi elementi:

$$S_{1a} = (Send,\ \langle rw = 1, data = "A", busy = true\rangle), \tag{3.3}$$

$$S_{1d} = (Send,\ \langle rw = 1, data = "D", busy = true\rangle), \tag{3.4}$$

$$S_{1k} = (Send,\ \langle rw = 1, data = "K", busy = true\rangle), \tag{3.5}$$

$$S_{1o} = (Send,\ \langle rw = 1, data = "O", busy = true\rangle), \tag{3.6}$$

$$S_{1r} = (Send,\ \langle rw = 1, data = "R", busy = true\rangle), \tag{3.7}$$

$$S_0 = (Send,\ \langle rw = 0, data = "", busy = true\rangle). \tag{3.8}$$

Vseh šest veznih elementov je med seboj v konfliktu. Vsi omogočajo proženje akcije *Send*, pojavi se pa lahko samo eden, ker pogoj *Busy* vsebuje samo en žeton z barvo *true*. Recimo, da se pojavi vezni element S_{1a}. Iz vhodnega pogoja *Send* se odstrani žeton z barvo (1,"A") in iz pogoja *Busy* žeton z barvo *true*. Ker ima žeton *busy* barvo *true* pomeni, da je naprava zasedena, zato se žeton z barvo (1,"A") vrne nazaj v pogoj *Command* in žeton z barvo *true* v pogoj *Busy*, kar narekujejo izrazi na izhodnih povezavah. V primeru, da želimo pisati na napravo, ko je le ta zasedena, se ukazi za pisanje ne bodo izvedli, dokler naprava ne bo prosta na sprejem novega ukaza za zapis znaka. Zato je potrebno osvežiti status naprave. V tem primeru je potrebno izvesti ukaz za branje (rw=0) zato, da osvežimo status naprave. Edini ustrezni vezni element, ki izvede osvežitev stanja naprave, je S_0. V tem primeru se iz pogoja *Busy* odstrani žeton z barvo *true*. Žeton z barvo (0,"") iz pogoja *Command* pa se prenese v pogoj *Register*, kar prinese novo označitev M_1.

V označitvi M_1 se lahko izvede samo akcija *Exe* z veznima elementoma

$$busy_- = (Exe,\ \langle rw = 0, data = "", displayed = "", stat = false, busyState = false\rangle) \tag{3.9}$$

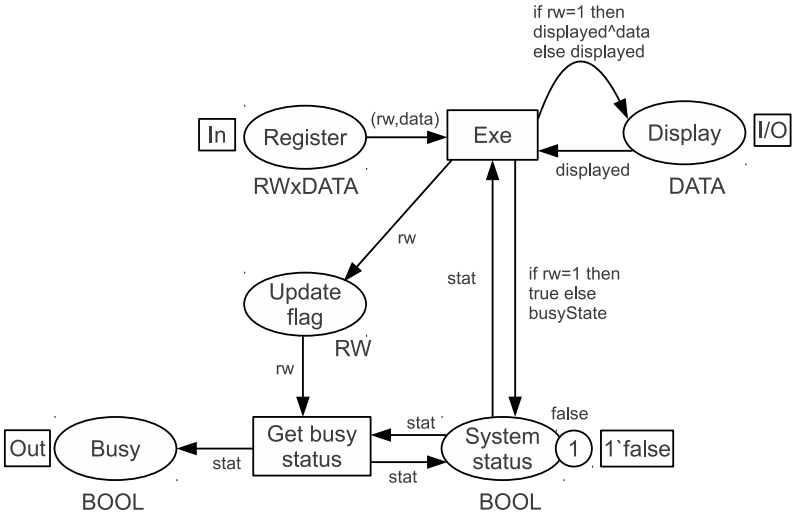
$$busy_+ = (Exe,\ \langle rw = 0, data = "", displayed = "", stat = false, busyState = true\rangle). \tag{3.10}$$

Spremenljivka *busyState* se nahaja samo na izhodni povezavi, kar pomeni, da se nanjo lahko veže poljubna vrednost iz barvnega niza *BOOL* [3, 4]. Vezna elementa $busy_-$ in $busy_+$ sta prav tako v konfliktu, ker sta oba omogočena, vendar imamo samo po en žeton v pogojema *Register* in *Display*. Recimo, da se pojavi vezni element $busy_-$. Ker želimo samo osvežiti stanje naprave, se v pogoj *Display* vrne žeton z barvo "" (prazen tekst),

pogoj *System status* dobi nov žeton z barvo *false* in pogoj *Update flag* dobi nov žeton z barvo *0*. Tako dobimo novo označitev M_2^-. V označitvi M_2^- se lahko sproži akcija *Get busy status* z veznim elementom $Gbs = (Get\ busy\ status,\ \langle rw = 0, stat = false \rangle)$, ki v pogoj *Busy* vrne žeton z osveženim stanjem naprave in prinese označitev M_3. Sedaj se lahko pravilno izvedejo tudi ukazi, ki predstavljajo zapis znaka na zaslon.

3.4 Modularna zgradba CPN modelov

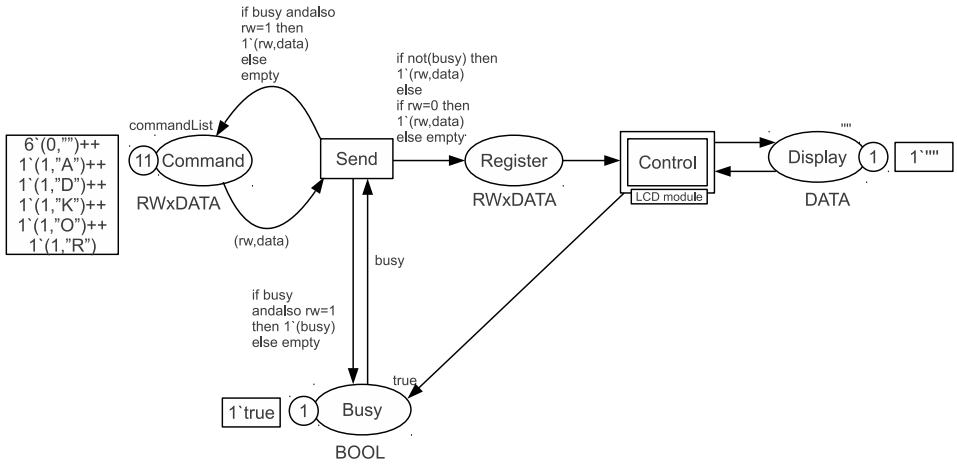
Modularna gradnja omogoča izdelavo kompleksnejših CPN modelov. Gradnja večjih in bolj kompleksnih modelov, kjer je tudi po sto ali več pogojev in akcij, lahko hitro pripelje do nepreglednosti modela. Zato model razbijemo na posamezne module. Tako je model z vsakim nivojem bolj podrobno opisan.



Slika 3.2: Krmilni modul.

Za ilustracijo vzemimo primer na sliki 3.1. Pošiljanje znakov na zaslon lahko razdelimo na pošiljatelja in krmilnik, ki poskrbi za ustrezen izpis znaka. Izdelava modula za pošiljatelja je nesmiselna, ker je že poenostavljen, medtem ko slika 3.2 prikazuje krmilni modul. Modul vsebuje dve akciji in pet pogojev. Pogoj *Register* je vhodni port, *Busy* izhodni port in *Display* vhodno-izhodni port. To pomeni, da pogoji *Register*, *Busy* in *Display* predstavljajo vmesnik, skozi katerega krmilni modul izmenjuje žetone z okoljem (ostalimi moduli). Krmilni modul bo sprejemal žetone preko vhodnega porta *Register* in

jih oddajal preko izhodnega porta *Busy*. Preko vhodno-izhodnega porta *Display* pa lahko modul sprejema in oddaja žetone. Kateri pogoji v CPN so porti lahko prepoznamo po pravokotni oznaki poleg pogoja, ki določa ali je pogoj vhodni, izhodni ali vhodno-izhodni port. Pogoja *Update flag* in *System status* sta notranja pogoja in sta pomembna samo za krmilni modul.



Slika 3.3: Končna oblika modularne zgradbe.

Slika 3.3 prikazuje končno obliko modularne zgradbe iz primera s slike 3.1. Na sliki 3.3 vidimo *nadomestno akcijo Control* (pravokotnik z dvojnimi črtami). Osnovna ideja v več nivojskem modelu je ta, da ustvarjamo asociacije med moduli in nadomestnimi akcijami [3, 4]. Ko je asociacija enkrat vzpostavljena, postane modul *pod-modul* za izbrano nadomestno akcijo. Kateri pod-modul pripada kateri nadomestni akciji vidimo s pomočjo pravokotne oznake. Sam pod-modul omogoča vpogled v bolj podrobno obnašanje nadomestne akcije. Primer s slike 3.3 vsebuje nadomestno akcijo *Control*, kateri pripada pod-modul z imenom *LCD module*, ki predstavlja krmilni modul.

Vhodno-izhodne pogoje nadomestne akcije imenujemo *vhodno-izhodni vtiči* (angl. *socket*) [3, 4]. V primeru s slike 3.3 je pogoj *Register* vhodni vtič za nadomestno akcijo *Control*, pogoj *Busy* izhodni vtič in pogoj *Display* vhodno-izhodni vtič. Vtiči predstavljajo vmesnik za nadomestne akcije in jih je potrebno povezati z ustreznimi porti. To naredimo s preslikavo portov, ki pove kateri port v pod-modulu se poveže s katerim vtičem nadomestne akcije. Vhodni porti se preslikajo v vhodne vtiče, izhodni porti v izhodne vtiče in vhodno-izhodni porti v vhodno-izhodne vtiče. Na primeru s slike 3.3

imajo vtiči enaka imena kot porti v pod-modulu na sliki 3.2, kar ni nujno potrebno. Ko se port preslika v določen vtič, pogoja predstavljata dva različna pogleda na isti pogoj. To pomeni, da imata pogoja vedno isto označitev in iste žetone z isto barvo [3, 4].

3.5 Uporaba časa v CPN

Za razliko od običajnih Petrijevih mrež lahko v CPN uporabimo informacijo o času. Z uporabo informacije o času lahko ocenjujemo kako dobro sistem izvaja svoje operacije. Ocenjujemo lahko tudi sisteme, ki delujejo v realnem času. Pri takih sistemih je lahko pravilno delovanje pogojeno s pravočasno izvedbo dogodkov in ali sistem konča operacije v nekem že vnaprej določenem časovnem roku.

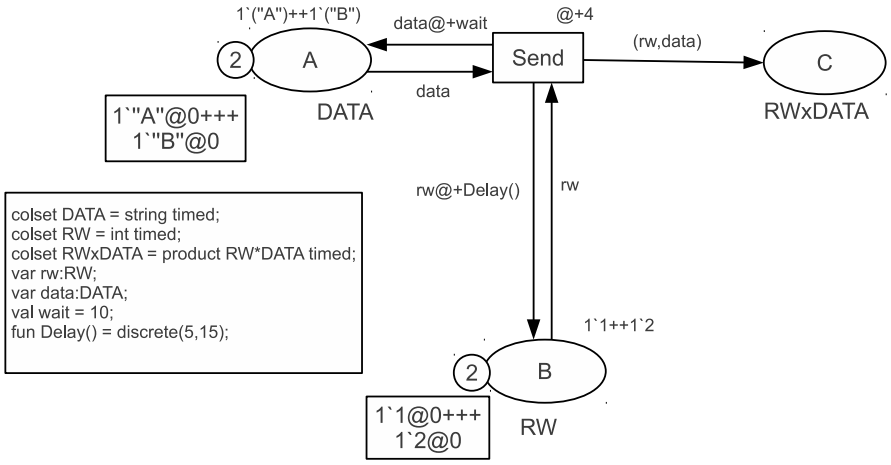
Časovne CPN so zelo podobne ne časovnim CPN. Glavna razlika med mrežama je, da v časovni CPN žeton poleg barve lahko nosi še drugo vrednost imenovano *časovni žig*. Tako v pogojih, kjer žetoni nosijo dodatno še časovni žig, dobimo *časovno posplošeno množico*. Časovna CPN ima globalno uro, ki predstavlja čas v modelu. Porazdelitev žetonov po pogojih skupaj z njihovim časovnim žigom in globalno uro imenujemo *časovna označitev* [3]. Časovni žig je lahko nenegativno celo ali realno število in nam pove, kdaj je žeton pripravljen za uporabo, tj. kdaj se lahko žeton odstrani iz pogoja, ko je akcija omogočena. Žetoni bodo nosili časovni žig, če bo barvni niz ob deklaraciji časovno označen. Barvni niz se časovno označi z CPN ML besedo *timed*.

Izvajanje časovnega CPN modela nadzoruje globalna ura. Model se zadržuje pri določenem času toliko časa, dokler imamo na voljo ustrezne vezne elemente, ki so barvno omogočeni (imamo ustrezne vhodne žetone) in pripravljeni za izvajanje (ustrezni žetoni imajo časovni žig manjši ali enak trenutni globalni uri). Šele takrat se akcija lahko sproži. Ko v modelu ni več ustreznih veznih elementov se globalna ura premakne na čas, kjer se lahko sproži naslednja akcija z ustreznimi veznimi elementi.

Slika 3.4 prikazuje preprost primer uporabe časovne CPN mreže. Barve žetonov se zapisujejo enako kot pri nečasovni CPN mreži, vendar imajo žetoni zapisano še informacijo o času (časovni žig). Pogoj A ima začetno označitev zapisano kot

$$1`"A"@0 + + + 1`"B"@0. \tag{3.11}$$

Časovni žig se zapisuje za simbolom @. V začetni označitvi M_0 nosijo vsi žetoni časovni žig 0. Operator +++ sprejme dve časovno posplošeni množici in vrne njuno unijo. Vrednost globalne ure v začetni označitvi je prav tako enaka 0. Recimo, da se pojavi



Slika 3.4: Preprost primer uporabe časovne CPN mreže. Barvni nizi so označeni z CPN ML besedo *timed*.

vezni element $(Send,\ \langle data = "A",\ rw = 1\rangle)$. Vidimo, da oba žetona obstajata in oba nosita časovni žig 0. To pomeni, da se akcija *Send* lahko sproži v času 0. Akcija *Send* odstrani žeton z barvo *data*="A" iz pogoja A in žeton z barvo rw=1 iz pogoja B in ustvari tri nove žetone, ki jih pošlje po izhodnih povezavah. Za te žetone je potrebno sedaj določiti nove časovne žige. Časovni žigi izhodnih žetonov se določijo na podlagi časovnih zamikov ob akcijah in posameznih izhodnih povezavah. Časovni zamiki ob akcijah se pripišejo vsem žetonom na izhodnih povezavah, medtem ko se časovni zamiki ob povezavah pripišejo samo žetonom, ki potujejo po tej povezavi [3]. Akcija *Send* na sliki 3.4 ima časovni zamik *@+4*, na izhodni povezavi proti pogoju A časovni zamik *@+wait* in proti pogoju B časovni zamik *@+Delay()*. Izraz na izhodni povezavi proti pogoju C nima nobenega dodatnega časovnega zamika. Skupen časovni zamik se sedaj izračuna po formuli

$$\textit{vrednost globalne ure } + \textit{čas. zamik ob akciji } + \textit{čas. zamik ob izhodni povezavi}. \tag{3.12}$$

Žeton proti pogoju C dobi nov časovni žig

$$0 + 4 + 0 = 4. \tag{3.13}$$

Prva vrednost 0 kaže kdaj se je akcija *Send* sprožila, druga vrednost 4 je časovni zamik ob akciji, tretja vrednost 0 pa je časovni zamik ob povezavi. Žeton, ki je poslan v pogoj

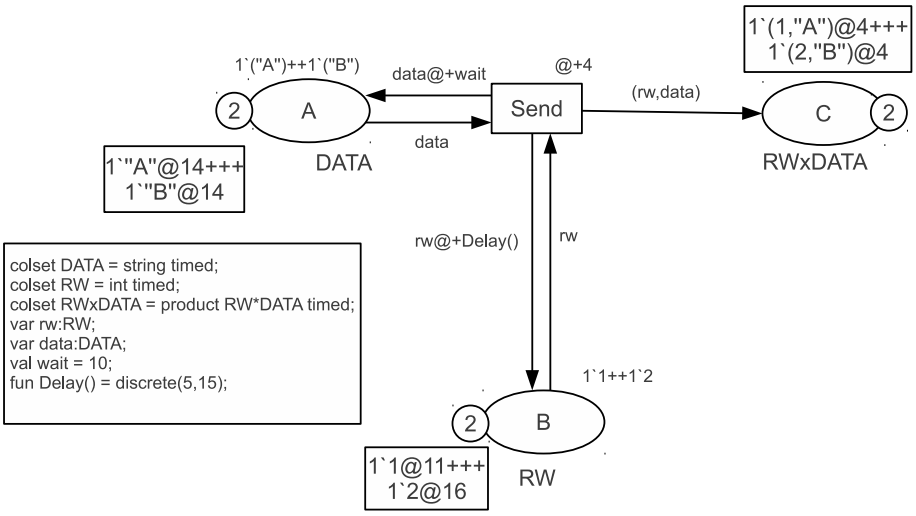
C, dobi barvo s časovnim zamikom 1‘(1,”A”)@4. Podobno se zgodi s preostalima dvema žetonoma. Žeton, ki je poslan v pogoj B, lahko dobi časovni žig

$$0 + 4 + 7 = 11. \tag{3.14}$$

Prva vrednost 0 kaže vrednost globalne ure ob sprožitvi akcije, druga vrednost 4 je časovni zamik ob akciji, tretja vrednost 7 pa je vrednost, ki nam jo vrne izraz *@+Delay()*. Funkcija *Delay()* vrača cele vrednosti na zaprtem intervalu med [5,15], kjer imajo vsa števila (na tem intervalu) enako verjetnost da bodo izbrana. Barva žetona, ki prispe v pogoj B, je enaka 1‘1@11. Žeton, ki se pošlje nazaj v pogoj A, dobi časovni žig

$$0 + 4 + 10 = 14. \tag{3.15}$$

Za prvi dve vrednosti velja enako, kot za prej opisana žetona. Tretjo vrednost 10 vrne izraz *@+wait*, kjer je *wait* konstanta z vrednostjo 10. V tem primeru žeton, ki prispe nazaj v pogoj A, dobi barvo 1‘”A”@14.



Slika 3.5: Preprost primer uporabe časovne CPN mreže. Označitev M_2.

Po prvi sprožitvi akcije *Send* dobimo novo označitev M_1. Na podoben način se akcija *Send* v označitvi M_1 ponovno sproži v času 0 z veznim elementom $(Send, \langle data = \text{”}B\text{”}, rw = 2\rangle)$. Dobimo označitev M_2, ki je prikazana na sliki 3.5. Sedaj se globalna ura premakne na čas @14. Vidimo, da je žeton v pogoju B z barvo 1 dosegljiv že v času @11, vendar noben od žetonov v pogoju A ni dosegljiv pred časom @14, zato se akcija *Send* pred časom @14 ne more sprožiti.

Akcije v časovnih CPN mrežah se prožijo takoj tj. so idealno hitre in ne zahtevajo nič časa. S pomočjo časovnih zamikov lahko modeliramo akcije, ki v sistemu nimajo neničelnega trajanja. To preprosto storimo tako, da se izhodnim žetonom dodajo časovni žigi ob proženju akcije, ki preprečujejo uporabo žetona, preden se akcija dejansko ne konča [3].

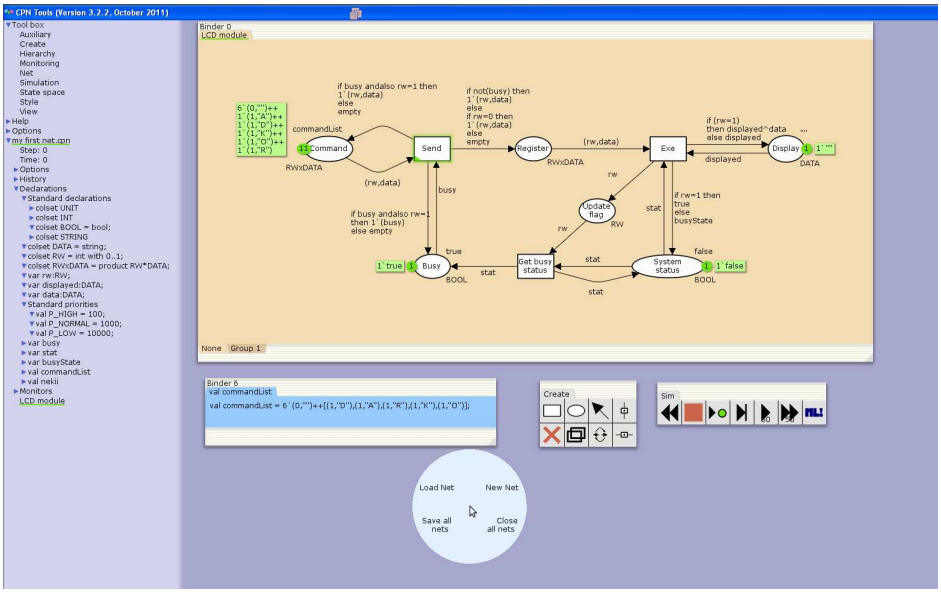
4 Orodje CPN Tools

CPN Tools je računalniško orodje za modeliranje in analizo CPN modelov. S pomočjo orodja lahko analiziramo obnašanje modela, preverimo njegove lastnosti in naredimo performančno analizo na osnovi simulacije. Orodje je brezplačno in izdano pod GNU GPL 2 licenco.

4.1 Uporabniški vmesnik

4.1.1 Pregled uporabniškega vmesnika

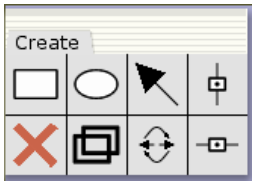
Slika 4.1 prikazuje uporabniški vmesnik orodja *CPN Tools*, ki je razdeljen na dva dela. Na levi strani se nahaja *indeks*. Orodja za izdelavo, simulacijo, dodajanje, oblikovanje, shranjevanje, itn. se nahajajo v rubriki *Tool box*. V indeksu se nahaja tudi predogled modela, ki prikazuje informacije o imenu, vsebovanih deklaracijah, modulih, zgodovini sprememb in nastavitvah. Dodatne rubrike je mogoče vključiti v predogled s pomočjo določenih orodij iz rubrike *Tool box*. S pomočjo trikotnika, ki se nahaja na levi strani določenega vnosa, lahko razkrijemo ali zakrijemo bolj podrobne informacije o vnosu. Preostali del, ki se nahaja na desni strani, se imenuje *delovna površina*. Delovna površina



Slika 4.1: Zajem zaslonske slike orodja CPN Tools.

vsebuje lahko več različnih strani, ki se imenujejo *Binder*. Poznamo dve vrsti strani. Ena vrsta vsebuje elemente CPN modela, druga vrsta pa vsebuje paleto z orodji s katerimi lahko uporabnik manipulira z modelom. Slika 4.1 prikazuje na delovni površini štiri strani. Na eni strani je narisan CPN model, na drugi je vsebovana deklaracija *commandList*. Preostali dve strani oz. paleti *Create* in *Sim* sta namenjeni manipulaciji z modelom. Poleg strani na delovni površini vidimo tudi *pomožni meni* okrogle oblike.

4.1.2 Izdelava CPN modelov



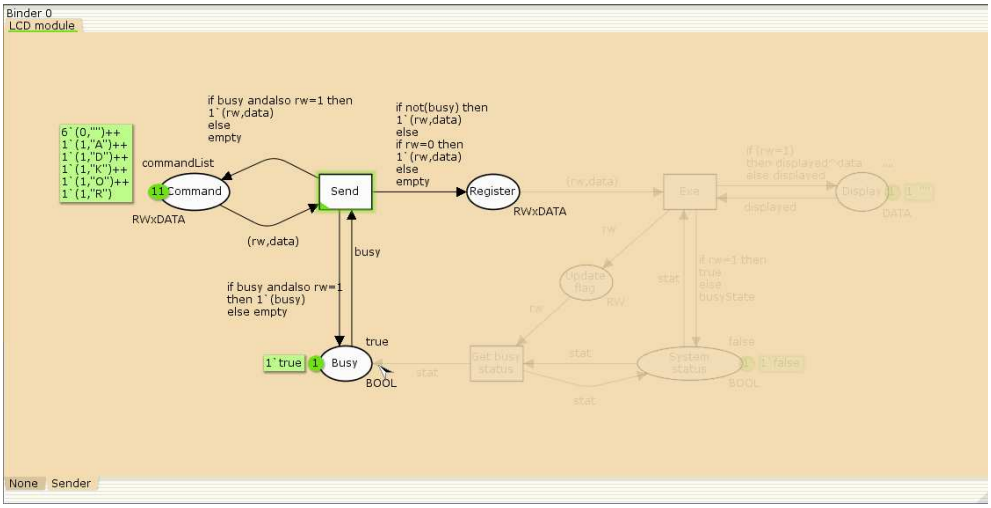
Slika 4.2: Paleta z orodji za izdelavo CPN modelov.

Orodje omogoča odpiranje že obstoječih ali dodajanje novih modelov s pomočjo palete *Net*. Ko je model ustvarjen, se v indeks naloži predogled modela. V predogledu so prikazane vse definirane deklaracije (barvni nizi, spremenljivke, funkcije in konstante).

Vsi napisi, ki se uporabljajo v modelu, morajo biti definirani v deklaracijah za izbrani model. Deklaracije dodajamo z orodjem *New Declaration* preko pomožnega menija. Slika 4.2 prikazuje paleto za izdelavo CPN modelov. Z orodji v paleti lahko

- dodajamo nove pogoje in akcije,
- dodajamo povezave,
- brišemo in kopiramo elemente,
- spremenimo smer povezave,
- dodajamo vertikalne in horizontalne smernice.

Ko izberemo ustrezni element, ga moramo opremiti z ustreznimi napisi. Vsak element ima različno število napisov, med njimi pa prehajamo s tipko *Tab*.



Slika 4.3: Združevanje elementov v posamezne skupine.

Če izdelujemo večje in bolj kompleksne modele, si lahko pomagamo z različnimi orodji. S paleto *Style* lahko modele opremimo z različnimi barvami, kar pripomore k lažji berljivosti modela. Novih elementov nam ni treba vedno znova ustvarjati, saj jih lahko z orodjem za kloniranje podvojimo. Podvojimo lahko skoraj vse elemente skupaj z napisi, barvo, obliko, itn. V primeru, da želimo popraviti določene lastnosti več elementov hkrati, jih lahko združimo v isto *skupino*. Skupine ustvarimo s pomočjo pomožnega menija in so vidne spodaj na strani kot posamezni zavihki. Slika 4.3 prikazuje nekaj

elementov skupaj, ki pripadajo isti skupini poimenovani *Sender*. Ostali elementi, ki niso v skupini, so osenčeni v ozadju.

4.1.3 Izdelava modularnih CPN modelov

Slika 4.4: Paleta z orodji za izdelavo CPN modulov.

S pomočjo palete *Hierarchy* lahko izdelujemo posamezne module, ki jih kasneje med seboj povežemo v celoto. Slika 4.4 prikazuje paleto z orodji s katerimi lahko

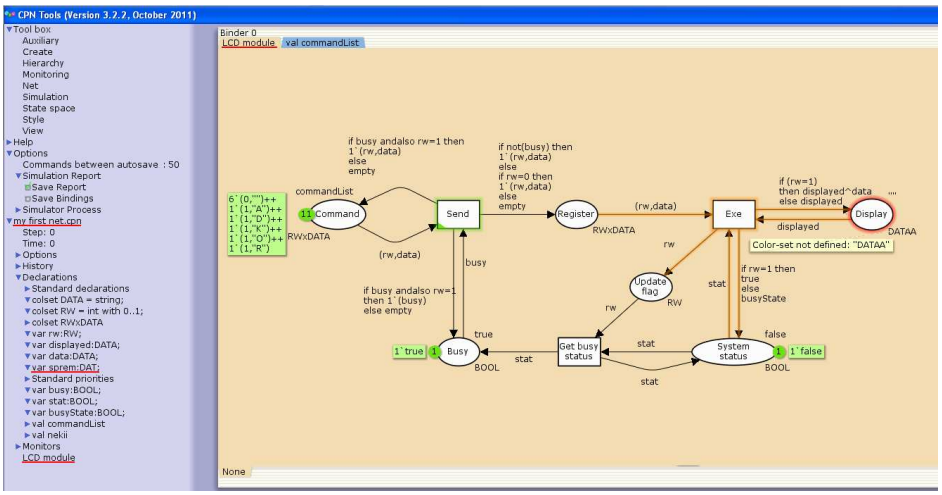
- prestavimo akcijo ali skupino v nov pod-modul,
- nadomestimo nadomestno akcijo z njenim pod-modulom,
- dodelimo pod-modul nadomestni akciji,
- povežemo port z vtičnikom,
- ustvarimo vhodni, izhodni in vhodno-izhodni port.

Orodja nam omogočajo tako *top-down* kot *bottom-up* pristop izdelave modela. Z orodjem za prestavljanje v nov pod-modul (top-down pristop) lahko premaknemo skupino elementov iz enega modula v nov pod-modul. V originalnem modulu se ustvari nadomestna akcija z ustreznimi povezavami. V pod-modulu se ustvarijo ustrezni porti, ki se preslikajo na ustrezne vtičnike. Z dodeljevanjem pod-modula (bottom-up pristop) se obstoječi modul dodeli ustrezni že obstoječi nadomestni akciji. Porti pod-modula se samodejno preslikajo na ustrezne vtičnike, kadar je to mogoče.

V indeksu je prikazana tudi informacija o hierarhiji modela. Trikotnik poleg imena modula nakazuje, da le ta vsebuje pod-module. Število zapisano v oklepajih poleg imena modula nakazuje, da imamo več primerkov tega modula v modelu. Če števila v oklepajih ni, pomeni da imamo samo en primerek tega modula v modelu [3].

4.1.4 Preverjanje sintakse in povratna informacija

Orodje sproti preverja nove vnose in sintakso modela. V primeru napake se nam ta tudi izpiše poleg elementa v obliki pogovornega okna. Elementi, ki vsebujejo napako, so obkroženi, ali pa podčrtani z rdečo barvo. Rdeča barva se odstrani šele, ko je napaka odpravljena. Primer na sliki 4.5 prikazuje napako pri pogoju *Display* in na deklaraciji spremenljivke *sprem*. Ker se preverjanje sintakse in generacija kode izvajata postopoma z urejanjem modela, lahko določene dele CPN modela izvedemo kljub temu, da model ni v celoti pravilen. Ob spremembi določenih delov modela se preverjanje sintakse in generacija kode izvaja samo na elementih, ki so odvisni od spremembe. Nekateri elementi se ne preverjajo, vse dokler ni zagotovljena minimalna sintaktična pravilnost [3].



Slika 4.5: Primer prikazovanja napak v CPN modelu. Pogoj *Display* uporablja barvni niz *DATAA*, ki v deklaracijah modela ni definiran. Spremenljivka *sprem* je definirana z barvnim nizom *DAT*, ki v deklaracijah modela ni definiran.

4.2 Format projekta

Orodje CPN Tools shranjuje ustvarjene modele v datoteko s končnico *.cpn*. Vsebina shranjene datoteke je struktura XML[1] (eXtensible Markup Language). Za pravilno strukturo dokumenta XML je definiran tudi eksterni DTD[2] (Document Type Definition), ki

[1] http://www.w3schools.com/xml/
[2] http://www.w3schools.com/dtd/default.asp

se nahaja na naslovu http://cpntools.org/DTD/6/cpn.dtd.

Izvorna koda 4.1: Glavni elementi ustvarjene datoteke z orodjem CPN Tools.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE workspaceElements
PUBLIC "-//CPN//DTD CPNXML 1.0//EN"
"http://cpntools.org/DTD/6/cpn.dtd">

<workspaceElements>
  <generator tool="CPN Tools"
             version="3.2.2"
             format="6"/>
  <cpnet>
    <globbox>...</globbox>
    <page id="ID6">...</page>
    <instances>...</instances>
    <options>...</options>
    <binders>...</binders>
    <monitorblock name="Monitors">...</monitorblock>
    <indexNode expanded="true">...</indexNode>
  </cpnet>
</workspaceElements>
```

Izvorna koda 4.1 prikazuje glavne elemente, ki jih vsebuje shranjena datoteka. Začne se s korenskim elementom <*workspaceElements*>. Sledi element <*generator ... /*>, v katerem je zapisana verzija orodja, s katerim je bil model ustvarjen in format zapisa, ki ga orodje trenutno uporablja. Element <*cpnet*> vsebuje vse gradnike, ki opisujejo elemente v indeksu in na delovni površini. Ti elementi so:

- <*globbox*>, ki vsebuje vse definirane deklaracije prikazane v indeksu orodja;
- <*page id="IDx"*>, ki predstavlja posamezen zavihek znotraj strani *Binder* in se lahko znotraj elementa <*cpnet*> pojavi večkrat; atribut *id* je enolično določen identifikator za vsak zavihek;
- <*instances*>, ki predstavlja instance na ustvarjene zavihke;

- <*options*>, ki vsebuje izbrane nastavitve v indeksu pod rubriko *Options* za posamezen model;
- <*binders*>, ki vsebuje nastavitve o posamezni strani in vsebuje informacijo, kateri zavihki pripadajo posamezni strani;
- <*monitorblock name="Monitors"*>, ki vsebuje informacije o uporabljenih nadzornikih ustvarjenih s pomočjo orodij v paleti *Monitoring*;
- <*indexNode expanded=x*>, ki vsebuje informacijo o tem, kateri vnosi v indeksu so bolj ali manj podrobno razkriti; vrednost atributa *expanded* je lahko *true* ali *false*;

Znotraj elementa <*globbox*> se nahajajo elementi, ki opisujejo definirane barvne nize, spremenljivke, funkcije in konstante v indeksu modela. Model na sliki 4.1 ima definiran barvni niz *DATA* in spremenljivko *data*. Barvni nizi se definirajo z elementom <*color*>, medtem ko se spremenljivke definira z elementom <*var*>. Oba primera definicije sta prikazana na primeru kode v 4.2.

Izvorna koda 4.2: Definicija barvnega niza *DATA* in spremenljivke *data* iz primera na sliki 4.1.

```
<color id="ID1412311014">
  <id>DATA</id>
  <string/>
  <layout>colset DATA = string;</layout>
</color>
<var id="ID1412312516">
  <type>
    <id>DATA</id>
  </type>
  <id>data</id>
  <layout>var data:DATA;</layout>
</var>
```

Poleg barvnih nizov in spremenljivk lahko definiramo še konstante in funkcije, ki se definirajo z elementom <*ml*>. Posamezne barvne nize, spremenljivke, konstante in funkcije lahko združujemo v bloke. Blok se definiran z elementom <*block*>, kar prikazuje primer izvorne kode 4.3.

Izvorna koda 4.3: Definicija bloka elementov.

```
<block id="ID1">
  <id>BLOCK NAME</id>
  <color id="IDx"> ... </color>
  ...
</block>
```

Element <*page id="IDx"*> vsebuje elemente oz. gradnike CPN modela na posameznem zavihku na strani *Binder*. Prvi element, ki sledi elementu <*page*>, je <*pageattr name= "Page Name"/*>, ki vsebuje informacijo o imenu zavihka. Nato sledijo gradniki CPN modela (pogoj, povezava, akcija). Pogoj predstavlja element <*place*>. Element vsebuje še pod-elemente, ki do potankosti opišejo ustvarjeni pogoj (uporabljene barve, napisi, barvni niz, začetna označitev, ...). Akcije predstavlja element <*trans*>, povezave pa so predstavljene z elementom <*arc*>. Podobno kot element <*place*>, imata tudi elementa za akcije in povezave še pod-elemente, ki ponudijo bolj podroben opis le teh.

Izvorna koda 4.4: Vključitev posameznih zavihkov na stran.

```
<instances>
  <instance id="ID2149" page="ID6"/>
</instances>
<binders>
  <cpnbinder id="ID2222" x="458"
             y="46" width="1426"
             height="929">
    <sheets>
      <cpnsheet id="ID2215"
                panx="-219.062500"
                pany="0.437500"
                zoom="1.000000"
                instance="ID2149">
        <zorder>
          <position value="0"/>
        </zorder>
      </cpnsheet>
```

```
      <textsheet id="ID1412377301"
                 panx="7.000000"
                 pany="20.000000"
                 zoom="1.000000"
                 decl="ID1412349117">
        <zorder/>
      </textsheet>
    </sheets>
  <zorder>
    <position value="1"/>
    <position value="0"/>
  </zorder>
  </cpnbinder>
</binders>
```

V elementu <*instances*> se ustvarijo instance za vsak element <*page*>, kar prikazuje koda 4.4. Element <*binders*> vsebuje informacijo o prikazanih zavihkih. Z atributi je določena velikost in pozicija strani *Binder*. Nato sledi element <*sheets*>, v katerem so opisani vsi vsebovani zavihki na strani. Zavihke se vključi z elementom <*cpnsheet*>, ki vsebuje pozicijo narisanega modela, faktor povečave in katera instanca elementa <*page*> je prikazana na tem zavihku. Kateri izmed zavihkov na strani je trenutno prikazan, je določeno z elementom <*zorder*>. Zavihek je lahko tudi vsebina neke deklaracije, kar opisuje element <*textsheet*>.

4.3 Simulacija

4.3.1 Izvajanje simulacije

Simulacija se v orodju izvaja na grafični način. Informacija o številu žetonov, ki se nahajajo v posameznem pogoju, je prikazana v majhnem krogu poleg pogoja. Poleg števila žetonov se prikazuje tudi barva žetonov, ki je zapisana v kvadratu poleg posameznega pogoja. Če pogoj ne vsebuje nobenega žetona, se ne prikaže nobena informacija.

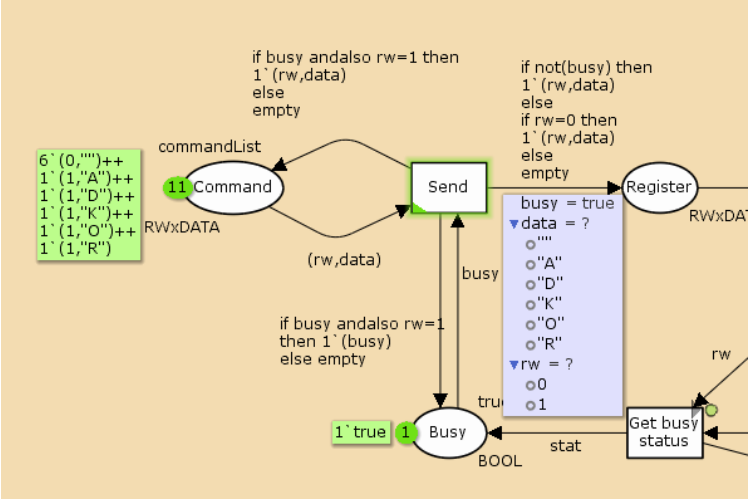
Orodja za izvajanje simulacije najdemo v paleti *Simulation*, ki je prikazana na sliki 4.6. Od leve proti desni imamo naslednja orodja:

Slika 4.6: Paleta z orodji za simulacijo.

- vrnitev na začetno označitev v mreži,
- ustavitev samodejno animirane simulacije,
- izvedba ene akcije z ročno izbranim veznim elementom,
- izvedba ene akcije z naključno izbranim veznim elementom,
- izvedba samodejno animirane simulacije; izvajanje mreže z naključno izbranimi veznimi elementi in prikazom trenutne označitve med vsakim korakom,
- izvedba hitre samodejne simulacije; izvajanje mreže z naključno izbranimi veznimi elementi brez prikaza trenutne označitve med vsakim korakom,
- ocenitev CPN ML izraza.

V interaktivnem načinu izvajanja simulacije simulator izračuna katere akcije so omogočene v trenutni označitvi. Omogočene akcije se obkrožijo z zeleno barvo. Odločitev, katera akcija se bo izvedla, je prepuščena uporabniku. Uporabnik lahko tudi sam določi kateri vezni element naj se uporabi za izvedbo akcije. Slika 4.7 prikazuje primer izbire veznega elementa za akcijo *Send*. Spremenljivka *busy* samodejno dobi vrednost *true*, ker pogoj *Busy* vsebuje samo en žeton. Po izbiri ustreznih vrednosti simulator ponovno izračuna katere akcije so omogočene in uporabnik lahko ponovno izbira med omogočenimi akcijami. Interaktivni način simulacije je počasen in je zato primeren za razhroščevanje (angl. *debugging*) modela.

V samodejno animirani simulaciji simulator sam izračuna množico omogočenih akcij v vsaki označitvi. Nato se simulator sam odloči, katere vezne elemente bo uporabil za izvedbo omogočenih akcij. Vsaka nova označitev se nato v modelu tudi osveži. Orodje s simbolom *predvajaj* (angl. *play*) omogoča tudi nastavitev števila korakov izvajanja simulacije. Po doseženem številu se simulacija samodejno ustavi, razen v primeru, ko se doseže označitev, kjer nimamo več nobene omogočene akcije ali pa simulacijo sami ustavimo z orodjem *Stop*.



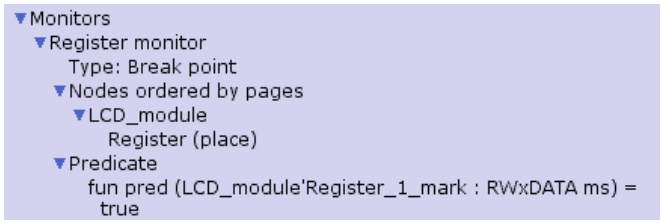
Slika 4.7: Ročni način izbire veznega elementa za akcijo *Send*.

Na modelu lahko izvajamo tudi hitro samodejno simulacijo, ki omogoča izvajanje nekaj tisoč korakov na sekundo [3]. Pred začetkom izvajanja simulacije uporabnik lahko določi koliko korakov se bo izvedlo, ali koliko časovnih enot v modelu naj se izvede. Ko sprožena simulacija doseže enega izmed postavljenih pogojev, se izvajanje ustavi in se izpiše dosežena označitev.

Simulacija nekega modela se lahko ustavi v končni označitvi. Končna označitev za primer na sliki 3.1 bi pomenila prenos vseh znakov (A, D, K, O, R) na prikazovalnik (pogoj *Display*). Vendar zaradi ne-determinizma v modelu ni nujno, da se bo končna označitev dosegla. Naprava lahko odpove in ne spremeni statusa za sprejem novega znaka. Zato lahko z večkratnim ponavljanjem simulacije samo testiramo ali je zasnovan model na videz pravilno postavljen. S simulacijo ne moremo zagotoviti, da se bodo za ustvarjen model izvedli vsi možni koraki. Če želimo preveriti lastnosti ustvarjenega modela moramo izvesti analizo prostora stanj, ki omogoča pokritje vseh možnih stanj v modelu. Simulator je tako zelo dobro orodje za testiranje in iskanje napak v modelu.

4.3.2 Uporaba nadzornikov

Orodja za izvajanje simulacije omogočajo, da se simulacija ustavi po določnem številu korakov ali po določenem pretečenem času. Simulacijo lahko ustavimo tudi z uporabo *nadzornikov* (angl. *monitor*), ki omogočajo ustavljanje v določenem stanju, ali ko se



Slika 4.8: Prikaz nadzornika v indeksu orodja CPN Tools.

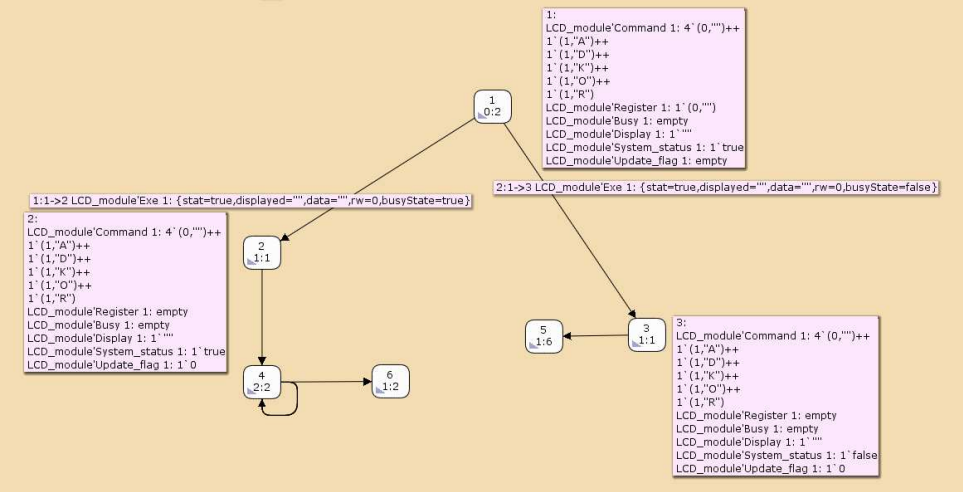
sproži določena akcija. V orodju CPN Tools imamo več različnih nadzornikov. Vsi nadzorniki vsebujejo *nadzorno funkcijo*, ki določa njegovo funkcionalnost. Pri nadzornikih, ki nadzorujejo proženje akcij, je nadzorna funkcija skrita pred uporabnikom.

Vsak na novo dodan nadzornik se ustvari iz predloge. Uporabnik nato predlogo popravi, da se doseže želena funkcionalnost. Nadzorniki opravljajo nadzor samo nad elementi ali skupino elementov, ki so povezani z nadzornikom. Slika 4.8 prikazuje ustvarjenega nadzornika *Register monitor* v indeksu orodja. Nadzornik lahko izvaja nadzor samo nad pogojem *Register*, ker je to edini pogoj, ki je povezan z nadzornikom. Nadzorna funkcija, ki jo opravlja, je zapisana v rubriki *Predicate*. Funkcija, ki je zapisana pregleduje, ali se je v pogoju *Register* pojavil nov žeton.

4.4 Analiza prostora stanj

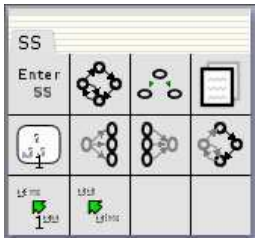
S simulacijo lahko izvedemo in pokažemo, da se izvajanje modela vedno uspešno zaključi glede na neka vnaprej določena stanja. Ne moremo pa z 100% gotovostjo trditi, da smo pokrili vse možne poti pri izvajanja simuliranega modela. Zato naredimo analizo prostora stanj, ki omogoča prikaz vseh možnih stanj, v katerih se lahko model nahaja. Z analizo se izračunajo vsa dosegljiva stanja (označitve) v sistemu in vse spremembe (pojavitev veznih elementov) CPN modela in se predstavijo z usmerjenim grafom v katerem vozlišča predstavljajo vse možne označitve, puščice pa predstavljajo pojavitev veznih elementov. Vezni elementi se na puščicah pojavijo samo, če pojavitev prikazanega veznega elementa vodi iz označitve M_1 v novo označitev M_2 [3, 4].

Slika 4.9 kaže del grafa analize prostora stanj. Vsako vozlišče v grafu je predstavljeno s tremi številkami. Število na vrhu v vozlišču je *številka vozlišča*. Števili ločeni z dvopičjem predstavljata *število predhodnih* in *število naslednjih* vozlišč. Zapis v pravokotniku poleg vozlišča predstavlja označitev posameznih pogojev v tem stanju. Napis



Slika 4.9: Delni prikaz prostora stanj iz primera na sliki 3.1.

na puščici predstavlja ustrezni vezni element, ki pripelje model v naslednje stanje.



Slika 4.10: Paleta z orodji za izdelavo prostora stanj.

Orodja za izdelavo prostora stanj najdemo v paleti *State Space* (slika 4.10). Od leve proti desni se nahajajo naslednja orodja:

- ustvari kodo za analizo prostora stanj,
- ustvari prostor stanj,
- ustvari SCC[3] graf,
- prikaži vozlišče v prostoru stanj,
- prikaži naslednike vozlišča,

[3] http://en.wikipedia.org/wiki/Strongly_connected_component

- prikaži predhodnike vozlišča,
- oceni CPN ML izraz, ki vrne seznam vozlišč ali povezav prostora stanj in prikaže ustrezna stanja/povezave iz prostora stanj,
- prikaži označitev v simulatorju, ki ustreza vozlišču prostora stanj,
- dodaj trenutno označitev v prostor stanj.

Prostor stanj nam orodje *CPN Tools* izračuna popolnoma samodejno. Izračunani graf se shranjuje v delovnem pomnilniku, kar pomeni, da se graf lahko izračuna samo v primeru, če imamo na voljo dovolj delovnega pomnilnika. Zato ima uporabnik pri izdelavi grafa opcije (uporaba pomožnega menija), s katerimi lahko nadzoruje izdelavo prostora stanj.

Ko je prostor stanj izračunan, lahko ustvarimo poročilo, ki nam prikaže nekaj osnovnih informacij o CPN modelu. Rubrike v poročilu so:

- *Statistics* - kako velik je prostor stanj. Primer na sliki 3.1 ima 3359 vozlišč, 5526 povezav in izračun prostora je trajal 2 sekundi. Izračunani so tudi podatki o SCC grafu, ki ima 3359 vozlišč, 4561 povezav in izračun grafa je trajal 1 sekundo. V primeru, da je število vozlišč v SCC grafu manjše od števila vozlišč v modelu nakazuje na to, da model lahko nikoli ne doseže končne označitve, ker v grafu obstajajo cikli [3].
- *Best integer bounds* - koliko žetonov je lahko, v katerikoli dosegljivi označitvi, hkrati prisotnih v posameznem pogoju.
- *Best upper multi-set bound* - največje število žetonov z isto barvo v opazovanem pogoju v katerikoli dosegljivi označitvi.
- *Best lower multi-set bound* - minimalno število žetonov z isto barvo v opazovanem pogoju v katerikoli dosegljivi označitvi.
- *Home properties* - prikazuje (številka vozlišča) domačo označitev M_{home}, ki se jo lahko doseže iz katerekoli dosegljive označitve.
- *Dead Markings* - označitve, kjer ni več veznih elementov.
- *Dead Transition Instances* - v dosegljivih označitvah ne obstajajo vezni elementi za proženje akcije.

- *Live Transition Instances* - akcija, ki se vedno pojavi v zaporedju dogodkov iz katerekoli dosegljive označitve.
- *Fairness Properties* – kako pogosto se posamezne akcije prožijo.

Poročilo tako vsebuje neke standardne informacije, ki se lahko ustvarijo za poljuben model. Če želi uporabnik pridobiti specifične informacije, ki veljajo samo za določen model, mu orodje omogoča izdelavo lastnih funkcij [6] napisanih v CPN ML programskem jeziku.

4.5 Performančna analiza

Izvaja se s pomočjo simulacije in uporabe podatkovnih nadzornikov (paleta *Monitors*). Skozi izvajanje daljših simulacij se s pomočjo podatkovnih nadzornikov zbira ustrezne podatke. Zbrani podatki se lahko zapišejo v dnevniške datoteke (angl. *log files*) za kasnejšo obdelavo, ali pa se jih shrani v obliki poročil, ki povzamejo glavne lastnosti modela (povprečje, standardna deviacija, intervali zaupanja, ...). Podatkovni nadzorniki poleg glavne funkcije vsebujejo še dve dodatni funkciji. *Inicializacijska funkcija* (angl. *initialisation function*) se lahko uporablja za zajem podatkov v začetni označitvi modela. *Stop funkcija* pa se lahko uporablja za zajem podatkov iz končne označitve. Inicializacijske in stop funkcije ni mogoče uporabiti za zajem podatkov iz veznih elementov. Podatkovni nadzorniki, ki so na voljo v orodju CPN Tools, so sledeči:

- *Data collector monitor* - nadzornik za zajem numeričnih podatkov iz veznih elementov in označitev, ki so dosegljive med simulacijo.
- *Count transition occurances monitor* - standardni nadzornik, ki šteje kolikokrat se akcija sproži med izvajanjem simulacije.
- *Marking size monitor* - standardni nadzornik, ki meri število žetonov v pogoju med izvajanjem simulacije in lahko izračuna povprečno ali pa maksimalno število žetonov.
- *Generic data collector monitor* – nadzornik katerega obnašanje določi uporabnik sam in lahko zajema katerekoli podatke iz modela.

5 Izdelava modela z orodjem CPN Tools

Današnji mediji, ki se uporabljajo za prenos podatkov med napravami, so nezanesljivi. Podatki se lahko med prenosom pokvarijo ali popolnoma izgubijo, kar za sprejemnika na drugi strani medija predstavlja problem. Da bi preprečili neljube dogodke je potrebno uvesti mehanizme, ki omogočajo, da se ob izgubi ali okvari podatkov napake zaznajo in nato tudi popravijo.

5.1 Protokol drsečega okna

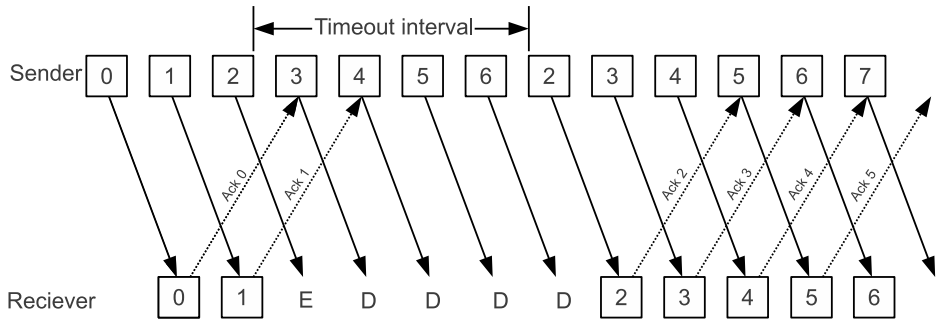
Drseče okno (angl. *sliding window protocol*) je po [7] protokol, ki omogoča zanesljiv in zaporeden prenos podatkov. Protokol je realiziran v treh različicah:

- ustavi in čakaj (angl. *stop and wait*),
- pojdi nazaj N (angl. *go back* N) in
- izbirno ponavljanje (angl. *selective repeat*).

Najbolj preprosta različica protokola je "ustavi in čakaj". Deluje tako da, pošiljatelj pošlje en paket sprejemniku in nato čaka na njegovo potrditev. Ko paket prispe do

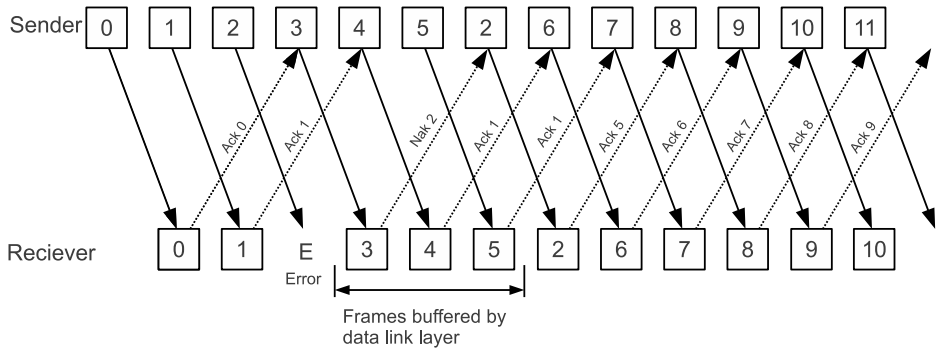
sprejemnika, ga ta sprejme v primeru, da je to naslednji paket, ki ga sprejemnik pričakuje in pošlje potrditev za prispeli paket. Če prispeli paket ni pričakovani paket, ali pa je paket pokvarjen, ga sprejemnik enostavno zavrže. Poslani potrditveni paket po nekem času lahko prispe do pošiljatelja ali pa se izgubi oz. pokvari. V primeru, da je potrditveni paket prispel nepoškodovan, pošiljatelj prevzame naslednji paket in ga pošlje sprejemniku. Če se potrditveni paket izgubi, oz. se izgubi poslani paket, pošiljatelj po izteku časovnika ponovno pošlje isti paket. Ponovni prenos paketa se prav tako opravi, če je prispeli paket pokvarjen. Da bi lahko pošiljatelj in sprejemnik razlikovala med posameznimi paketi, morajo biti paketi pri prenosu oštevilčeni z neko zaporedno številko. Za protokol "ustavi in čakaj" je zadosti, če se za prenos uporablja 1-bitno zaporedje (števili 0 in 1). Zato ima protokol ustavi in čakaj okno dolžine 1.

Težava protokola "ustavi in čakaj" je v hitrosti pošiljanja paketov. Če obhodni čas (angl. *round trip time*) enega paketa ni zanemarljiv, potem pošiljatelj večji del časa ne počne ničesar ampak samo čaka na potrditev. Zato pošiljatelju dovolimo, da lahko pošlje W paketov, preden se ustavi in čaka na njihovo potrditev. Sprejemnik na drugi strani, sprejema pakete in pošilja potrditev za posamezni sprejeti paket. V primeru, da prispeli paket ni pričakovani paket, se ga zavrže. Prav tako se zavržejo tudi vsi ostali paketi, ki prispejo za pokvarjenim ali izgubljenim paketom. Velikost okna je zato na sprejemni strani enaka 1. Po nekem času pošiljatelj ugotovi, da ni več potrditev za posamezne pakete, zato začne ponovno pošiljati vse pakete od zadnjega še ne potrjenega paketa. Tak protokol se imenuje "pojdi nazaj N" in je prikazan na sliki 5.1.



Slika 5.1: Prikaz delovanja protokola "pojdi nazaj N" [7].

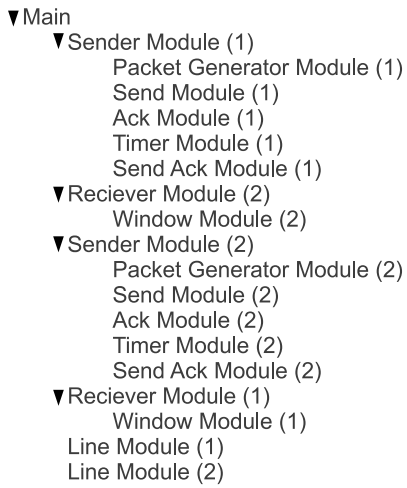
Protokol "pojdi nazaj N" deluje dobro, če so napake pri prenosu redke. V primeru, da imamo veliko napak pri prenosu, protokol porabi veliko pasovne širine za ponovno pošiljanje izgubljenih paketov. Da bi se izognili ponovnemu pošiljanju nepotrjenih paketov, sprejemnik namesto da zavrže pakete, ki pridejo za izgubljenim oz. pokvarjenim paketom, jih začasno shrani v predpomnilnik. Za izgubljeni paket pa se pošlje negativno potrditev (angl. *negative acknowledgment – NAK*). Ob sprejemu negativne potrditve, pošiljatelj ponovno pošlje paket, ki se je pri prenosu izgubil. Šele ko ima sprejemnik na voljo vse pakete, tudi izgubljene, jih lahko nato v pravilnem vrstnem redu posreduje višji plasti. Tak protokol se imenuje "izbirno ponavljanje" in je prikazan na sliki 5.2. Pri "izbirnem ponavljanju" je dolžina okna tako pri sprejemniku kot tudi pošiljatelju večja od 1. Dolžina okna v tem primeru predstavlja sekvenco paketov, ki jih lahko pošiljatelj pošlje brez potrditve oz. jih sprejemnik lahko sprejme, ne da bi jih zavračal v primeru napak pri prenosu.



Slika 5.2: Prikaz delovanja protokola "izbirno ponavljanje" [7].

5.2 Realizacija protokola izbirno ponavljanje

V nadaljevanju je prikazana izdelava protokola "izbirno ponavljanje", njegovo delovanje pa je podrobno opisano v [7]. Celoten protokol je razdeljen na različne module, hierarhija modela pa je prikazana na sliki 5.3. Na najvišjem nivoju imamo glavni modul z imenom *Main*. Modul predstavlja dve napravi (A in B), ki si želita pošiljati pakete. Vsaka naprava vsebuje modul za pošiljanje (*Sender Module*) in sprejemanje paketov (*Reciever Modul*). Modul za pošiljanje je naprej razdeljen še na več pod-modulov, kjer vsak pod-



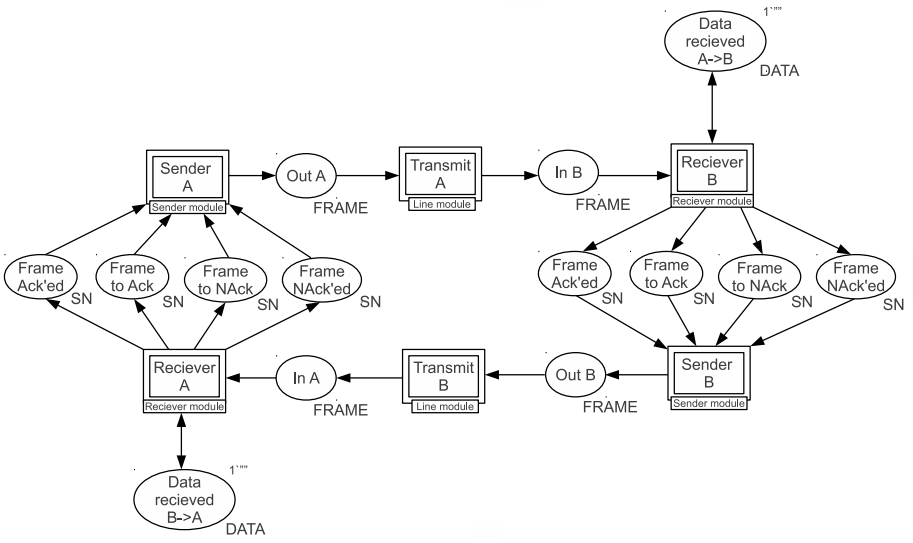
Slika 5.3: Hierarhija modela "izbirno ponavljanje" v orodju CPN Tools.

modul opravlja svojo nalogo. *Packet Generator* skrbi za izdelavo paketov, *Send* skrbi za pošiljanje okvirjev, *Ack* skrbi za potrjevanje paketov in pomikanje okna, *Timer* skrbi, da se izgubljeni paketi ponovno pošljejo in *Send Ack* skrbi, da se pošljejo potrditve za tiste pakete, ki so prispeli brez napak. Pod-modul *Window* pa skrbi za ustrezno pomikanje okna pri sprejemniku. Napravi nato lahko komunicirata preko povezave, ki je predstavljena z modulom *Line*.

5.2.1 Glavni modul

Slika 5.4 prikazuje glavni modul *Main* ter način, kako se dve napravi pravilno povežeta, da si lahko izmenjujeta pakete. Nadomestni akciji *Sender A* in *Reciever A* skupaj predstavljata napravo *A*. Za notranjo komunikacijo skrbijo pogoji *Frame Ack'ed*, *Frame to Ack*, *Frame to Nack* in *Frame Nack'ed*. Preko teh pogojev sprejemnik sporoča, katere pakete je potrebno potrditi, oz. so bili potrjeni in katere je potrebno ponovno poslati. Barva žetonov v teh pogojih je tipa *int*, barvni niz *SN* pa ponazarja zaporedno številko paketa v modelu.

Ko je nek paket pripravljen za pošiljanje, se le ta v okvirju, pojavi v pogoju *Out*. Pogoji *OutA*, *OutB*, *InA* in *InB* nosijo barvni niz *FRAME*, ki je produkt tipov *KIND*, *SN* in *DATA*. Deklaracija tipa *FRAME* je prikazana v kodi 5.1. Tip *KIND* predstavlja dve vrsti okvirjev tj. podatkovni okvir (barva *true*) in nadzorni okvir (barva *false*). Tip



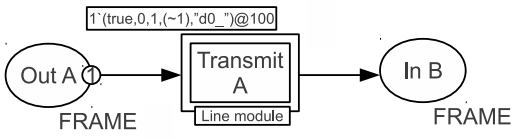
Slika 5.4: Povezava dveh naprav za prenos podatkov.

DATA pa predstavlja paket s podatki (barva tipa *string*), ki jih želimo poslati.

Izvorna koda 5.1: Definicija barvnega niza FRAME in pripadajočih tipov.

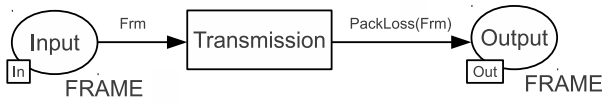
```
colset SN = int timed;
colset DATA = string;
colset KIND = bool;
colset FRAME = product KIND*SN*SN*DATA timed;
```

Barvni niz *FRAME* tako ponazarja okvirje, ki se prenašajo med napravama. Vsak okvir je organiziran kot peterček *(bool, int, int, int, string)*. Logični pomen okvirja bi tako bil (*podatkovni/nadzorni okvir*, *zaporedna št. paketa (Seq)*, *zadnji prejeti paket (Ack)*, *zadnji izgubljeni paket (NAck)*, *paket s podatki*). Barvni niz *FRAME* je tudi časovno označen. Čas na okvirjih predstavlja čas, ki je potreben, da okvir pripotuje na drugo stran medija. Primer okvirja je prikazan na sliki 5.5. Vidimo, da pogoj *OutA* vsebuje en žeton, ki predstavlja poslani okvir. Na prvem mestu imamo vrednost *true*, ki pove, da gre za podatkovni okvir. Na drugem mestu imamo vrednost, ki pove, da je naprava A poslala paket z zaporedno številko 0. Na tretjem mestu se nahaja vrednost, ki pove, da je zadnji pravilno prejeti paket imel zaporedno številko 1. Na četrtem mestu se nahaja vrednost



Slika 5.5: Primer žetona, ki predstavlja okvir v modelu.

(~ 1), ki pove, da ni nobenega izgubljenega paketa. Na petem mestu se nahaja dejanski paket s podatki "d0_". Časovni žig @100 pa pove, da bo okvir prispel do sprejemnika v času 100.



Slika 5.6: Pod-modul z imenom *Line module* nadomestne akcije *TransmitA*.

Nadomestna akcija *Transmit A* predstavlja nezanesljiv medij. Slika 5.6 prikazuje zgradbo pod-modula nadomestne akcije *Transmit A*. Pogoj *Input* je vhodni port, ki je povezan na vhodni vtič *OutA* nadomestne akcije *Transmit A*. Podobno velja za pogoj *Output*, ki je izhodni port povezan z izhodnim vtičem *InB* nadomestne akcije *Transmit A*. Ob proženju akcije *Transmission* se na izhodni povezavi izvede funkcija *PackLoss*, definirana v kodi 5.2.

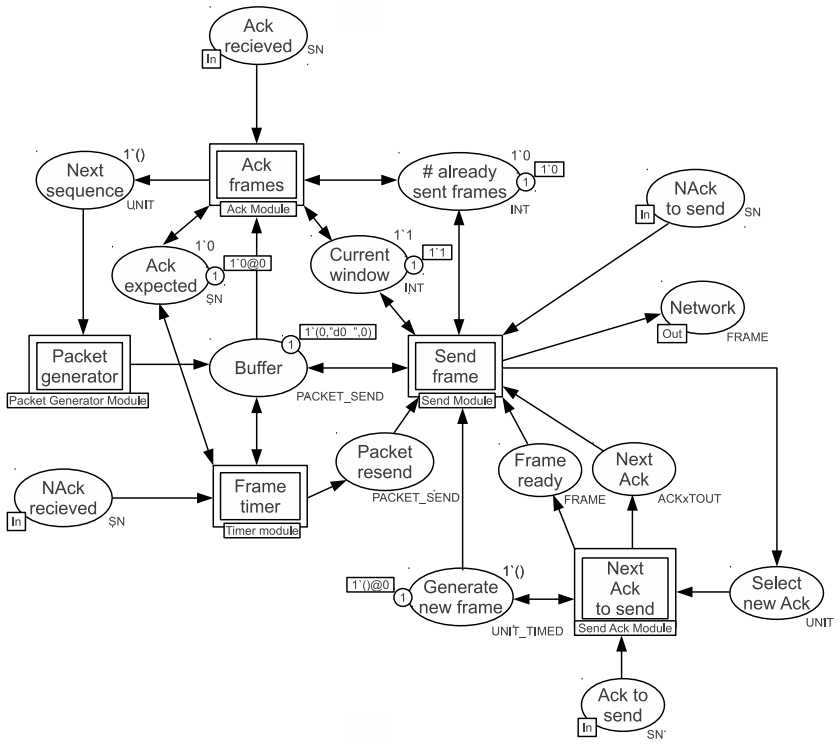
Izvorna koda 5.2: Definicija funkcije PackLoss.

```
val correct = 0.85;
fun PackLoss(frame)=
if uniform(0.0,1.0) < correct then
 1‘frame
else
 empty;
```

Funkcija *PackLoss* simulira izgube na prenosnem mediju. S konstanto *correct* (vrednosti med 0.0 in 1.0), pa določimo odstotek pravilno prenesenih okvirjev. Okvir iz slike 5.5, se bo tako v času @100 pojavil v pogoju *InB* z verjetnostjo 0.85.

5.2.2 Pošiljanje novih paketov

Vsakič, ko se v oknu sprosti mesto, oz. se kakšen paket potrdi, pošiljatelj začne s pripravo naslednjega paketa. Pogoj *Next sequence* na sliki 5.7 pove nadomestni akciji *Packet generator*, koliko novih paketov se lahko doda v predpomnilnik. Na začetku se lahko v pred-



Slika 5.7: Pod-modul *Sender Module* nadomestne akcije *Sender* na sliki 5.4. Stanje v CPN mreži po izvedbi nadomestne akcije *Packet generator*.

pomnilnik doda 1 paket, ker je trenutna velikost okna enaka 1 (pogoj *Current window*). Po proženju nadomestne akcije *Packet generator* na sliki 5.7 dobimo v predpomnilnik *Buffer* naslednji paket, ki ga je potrebno poslati. V pogoju se nahajajo paketi, ki jih je potrebno poslati, oz. so bili že poslani in čakajo na potrditev, da so bili uspešno prejeti s strani sprejemnika. Pogoj *Buffer* nosi barvni niz *PACKET_SEND*, definiran v kodi 5.3. *PACKET_SEND* predstavlja pakete z zaporedno številko in časovnikom. Največje

zaporedno število, ki je še dovoljeno, določi uporabnik s konstanto *MAX_SEQ*. Po [7], se vrednost konstante *MAX_SEQ* izračuna po formuli

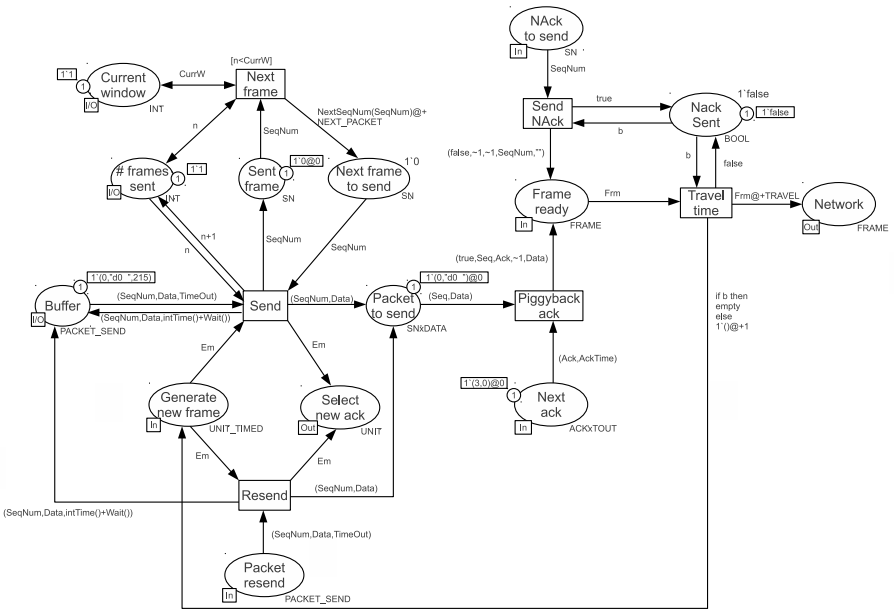
$$MAX\_SEQ = 2^n - 1, \tag{5.1}$$

kjer n predstavlja število bitov uporabljenih za oštevilčenje paketov. Vrednost časovnika določa, kdaj je potrebno paket ponovno poslati v primeru, da potrditev za poslani paket ne pride v naprej določenem času. Na začetku je vrednost časovnika enaka 0, kar pomeni, da paket še ni bil poslan.

Izvorna koda 5.3: Definicija barvnega niza PACKET_SEND.

```
colset PACKET_SEND = product SN*DATA*INT;
```

Ko je paket v predpomnilniku in imamo v oknu še kakšno prazno mesto, se sproži nadomestna akcija *Send frame*. Naloga nadomestne akcije je, da paket pretvori v okvir, ki je primeren za pošiljanje preko medija. Slika 5.8 prikazuje, kako se paket spremeni v okvir. Akcija *Send* se sproži, če se v predpomnilniku nahaja ustrezni paket, ki še ni bil poslan. Pogoj *Next frame to send* določa kateri paket se bo poslal v naslednjem trenutku. Po sprožitvi akcije *Send*, se paket z zaporedno številko *SeqNum* doda v pogoj *Packet to send*. V pogoju *Buffer* se osveži časovnik za ponovno pošiljanje paketa. Žeton iz pogoja *Generate new frame* se prenese v pogoj *Select new ack*, kar za trenutek onemogoči akciji *Send* in *Resend*. Število poslanih paketov (pogoj *# frames sent*) se poveča za 1. S tem se je onemogočilo nadaljnje pošiljanje novih paketov, ker je število poslanih paketov enako trenutni velikosti okna. S prenosom žetona v pogoj *Select new ack*, se lahko sproži nadomestna akcija *Next Ack to send*, prikazana na sliki 5.7. Nadomestna akcija priskrbi naslednjo zaporedno številko paketa, ki ga je potrebno potrditi. Če sprejemnik, na isti strani kot pošiljatelj že dalj časa ni sprejel nobenega novega paketa, se pošlje potrditev za zadnji še pravilno sprejeti paket. Na začetku je zaporedna številka zadnjega sprejetega paketa enaka *MAX_SEQ*. Zato nadomestna akcija *Next Ack to send* v pogoj *Next ack* doda žeton z vrednostjo 3, kar je prikazano na sliki 5.8. Sedaj se lahko sproži akcija *Piggyback ack*, ki trenutnemu paketu, ki se pošilja, doda še potrditev za zadnji prejeti paket. Naknadno se doda še informacija, da okvir, ki se pošilja, nosi podatke (prvi element v okvirju ima vrednost *true*). Četrti element v okvirju ima vrednost (~ 1), ker se negativne potrditve vedno pošiljajo v svojem okvirju, ki jih sestavlja akcija *Send NAck*. Sestavljen okvir se nato pojavi v pogoju *Frame ready*. Akcija *Travel time* doda okvirju

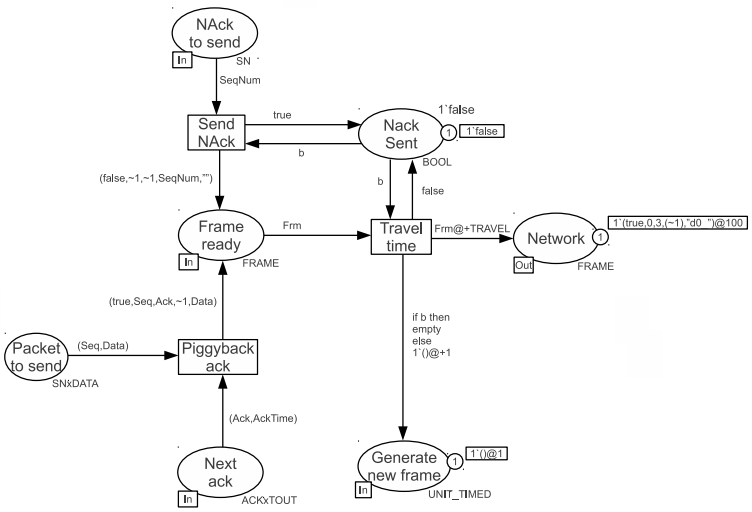


Slika 5.8: Pod-modul *Send Module* nadomestne akcije *Send frame* na sliki 5.7. Stanje v CPN mreži po izvedbi akcije *Send* in nadomestne akcije *Next Ack to send*, prikazane na sliki 5.7.

čas (vrednost konstante *TRAVEL*), ki določi koliko časa potuje paket do sprejemnika, kar je prikazano na sliki 5.9. Okvir se nato pojavi v pogoju *Network*, ki je povezan z vtičnikom *Out* v glavnem modulu. V pogoj *Generate new frame* se nato vrne nazaj žeton. S tem se omogoči pošiljanje novega oz. ponovno pošiljanje nepotrjenega paketa šele v času, ki je za +1 večji od trenutnega časa.

5.2.3 Sprejemanje paketov

Sprejemanje paketov je razdeljeno v dve fazi. Slika 5.10 prikazuje, kako prva faza v nadomestni akciji *Reciver* poteka. V pogoju *Recieved frame* se pojavljajo prejeti okvirji. Pogoj *Recieved frame* predstavlja vhodni port za vtič *In* nadomestne akcije *Reciever* iz slike 5.4. V pogoju *Recieved frame* se tako v času @100 lahko pojavi paket, ki smo ga poslali v prejšnjem razdelku 5.2.2. V primeru, da se paket izgubi, se pri pošiljatelju sproži časovnik in se ponovno pošlje isti paket. Ob pojavitvi okvirja v pogoju *Recieved*

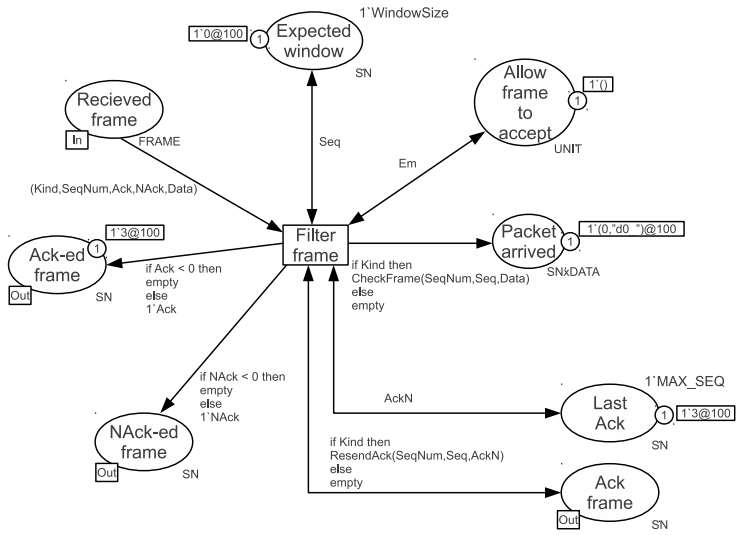


Slika 5.9: Delni prikaz pod-modula *Send Module* nadomestne akcije *Send frame* iz slike 5.7. Stanje v CPN mreži po izvedbi akcij *Piggyback ack* in *Travel time*.

frame se sproži akcija *Filter frame*. V pogoj *Ack-ed frame* se doda zaporedna številka zadnjega potrjenega paketa. V pogoj *NAck-ed frames* se doda zaporedna številka zadnjega izgubljenega paketa. Na primeru iz slike 5.9 je vrednost zadnjega nepotrjenega paketa enaka ~ 1, kar pomeni, da sprejemnik ni izgubil nobenega paketa, ki mu ga je pošiljatelj poslal. V primeru, da se paket izgubi, je vrednost na tem mestu večja od ~ 1 in se prenese v pogoj *NAck-ed frames*. V pogoj *Packet arrived* se prenese vsebina paketa in zaporedna številka pod pogojem, da je prejeti okvir podatkovni okvir in da se paket nahaja znotraj pričakovanega okna.

Izvorna koda 5.4: Definicija funkcije CheckFrame.

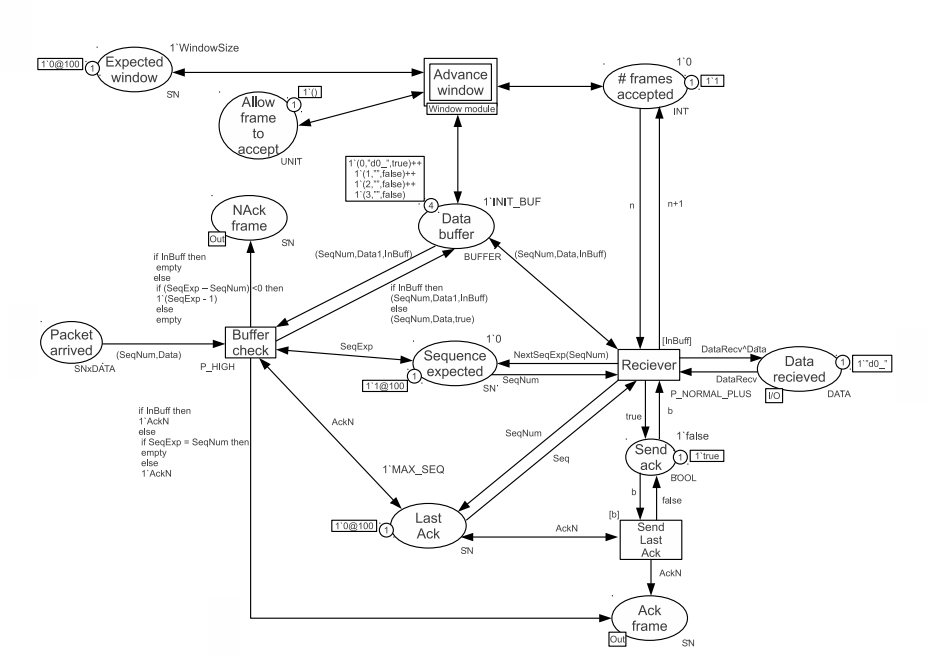
```
fun CheckFrame(sNum,MinW,fData)=
if (sNum >= MinW) andalso (sNum < (MinW+WindowSize)) then
1‘(sNum,fData)
else
empty;
```



Slika 5.10: Delni prikaz pod-modula *Reciever Module* nadomestne akcije *Reciever* na sliki 5.4. Stanje v CPN mreži po prihodu okvirja v pogoj *Recieved frame* in izvedbi akcije *Filter frame* v času @100.

Koda 5.4 prikazje definicijo funkcije *CheckFrame*, ki preveri ali se prejeti paket nahaja v pričakovanem oknu. Če je pogoj izpolnjen, se paket prenese v pogoj *Packet arrived*, kot je to prikazano na sliki 5.10. V primeru, da se sprejeti paket ne nahaja v mejah pričakovanega okna, pomeni da pošiljatelj ni sprejel potrditve za poslane pakete, ki jih je sprejemnik sprejel. Zato se pošlje potrditev za zadnji še pravilno sprejeti paket. Zaporedna številka zadnjega pravilno sprejetega paketa se nahaja v pogoju *Last Ack*, ki se doda v pogoj *Ack frame*.

Slika 5.11 prikazuje drugo fazo v kateri se preverja ustreznost sprejetega paketa in sprejem, ter pošiljanje paketov na višjo plast. Akcija *Buffer check* se sproži vsakič, ko je v pogoju *Packet arrived* prisoten paket. Če se sprejeti paket ne nahaja v predpomnilniku *Data buffer*, se ga doda, kot je prikazano na sliki 5.11. V primeru, da je paket že shranjen v predpomnilnik, sprejemnik doda v pogoj *Ack frame* zaporedno številko zadnjega sprejetega paketa, trenutnega pa enostavno zavrže. Če se kakšen izmed paketov izgubi, kot v sliki 5.2, se v pogoj *NAck frame* doda zaporedna številka izgubljenega paketa.



Slika 5.11: Delni prikaz pod-modula *Reciever Module* nadomestne akcije *Reciever* na sliki 5.4. Stanje v CPN mreži po sprožitvi akcij *Buffer check* in *Reciever* v času @100.

Paketi, ki pridejo za izgubljenim paketom se shranijo v predpomnilnik *Data buffer* in se ne zavržejo. Po shranjevanju paketa v predpomnilnik se sproži akcija *Reciever*. Pogoj *Sequence expected* vsebuje zaporedno številko pričakovanega paketa. Če se pričakovani paket nahaja v predpomnilniku *Data buffer*, se paket prenese v pogoj *Data recieved*, ki predstavlja naslednjo (višjo) plast v modelu (npr. mrežna plast v ISO/OSI). Vrednost žetona v pogoju *# frame accepted*, se poveča za 1, saj smo uspešno sprejeli pričakovani paket. V pogoj *Sequence expeted* se zapiše zaporedna številka naslednjega pričakovanega paketa. Pogoj *Last ack* pa dobi novo zaporedno številko, tj. zaporedno številko zadnjega sprejetega paketa. Slika 5.11 prikazuje stanje po sprožitvi akcije *Reciever*. Akcija *Send Last Ack* omogoča, da sprejemnik pošlje samo eno potrditev, za več zaporedoma pravilno sprejetih paketov. Zaporedna številka v pogoju *Last Ack* se doda v pogoj *Ack frame*, žeton v pogoju *Send ack* pa dobi barvo *false*. Nadomestna akcija *Advance window* omogoča pomikanje okna na sprejemni strani. Ko vrednost žetona v pogoju *# frame accepted* doseže velikost okna, ki jo definira uporabnik s konstanto *WindowSize*,

se okno pomakne za *WindowSize* mest naprej. Največja velikost okna se po [7] izračuna po formuli

$$WindowSize = (MAX\_SEQ + 1)/2. \tag{5.2}$$

Nova spodnja meja okna se nato zapiše v pogoj *Expected window*. Nadomestna akcija *Advance window* pobriše tudi predpomnilnik *Data buffer*.

5.2.4 Pošiljanje in sprejemanje potrditev

V pogoju *Ack frame* na sliki 5.11 se pojavljajo zahteve za potrjevanje pravilno sprejetih paketov. Pogoj *Ack frame* je povezan s pogojem *Ack to send*, ki je prikazan na sliki 5.7. Ob proženju nadomestne akcije *Next ack to send* na sliki 5.7 se zaporedna številka potrditve doda v seznam potrditev. Istočasno se za dodano potrditev sproži časovnik. V primeru, da ima pošiljatelj pripravljen nov paket za pošiljanje še preden se izteče časovnik za najstarejšo potrditev, se potrditev pošlje skupaj s paketom v istem okvirju, časovnik za to potrditev pa se ustavi. V nasprotnem primeru se potrditev pošlje kot ločen okvir, ki se po izteku časa določenega s konstanto *ACK_WAIT*, pojavi v pogoju *Frame ready*. Od tu naprej se okvir pošlje tako, kot vsi ostali okvirji, ki so pripravljeni za pošiljanje.

Po določenem času potrditveni okvir prispe do sprejemnika. Potrditveni okvir se dekodira v prvi fazi pri sprejemniku, kot je to zapisano v razdelku 5.2.3. Pogoj *Ack-ed frame* na sliki 5.10 je povezan s pogojem *Ack recieved* na sliki 5.7. Ob sprejemu potrditve za določen paket se sproži nadomestna akcija *Ack frames*, v kateri se najprej preveri veljavnost potrditve. Začetek okna določa pogoj *Ack expected*. Če prispela potrditev ni znotraj okna, se potrditev enostavno zavrže. V nasprotnem primeru se potrditev sprejme. Nato se izračuna zaporedna številka naslednjega pričakovanega paketa, ki se zapiše v pogoju *Ack expected*. Trenutna velikost okna, zapisana v pogoju *Current window*, se poveča za 1. Tako se okno na pošiljateljevi strani počasi veča do največje velikosti, ki je določena s konstanto *WindowSize*. Prav tako se iz predpomnilnika *Buffer* odstranijo pravkar potrjeni paketi. V pogoj *Next sequence* se doda toliko žetonov, kolikor prostih mest se je pojavilo v trenutnem oknu. Tako lahko nadomestna akcija *Packet generator* doda nove pakete v predpomnilnik *Buffer* in krog pošiljanja novih paketov se ponovi.

5.3 Izvajanje simulacije in rezultati

5.3.1 Nastavitev parametrov in možne napake

Izvorna koda 5.5: Nastavitev parametrov za izvajanje simulacije.

```
val correct = 1.0;
val MAX_SEQ = 1;
val WindowSize = 1;
val ACK_WAIT = 1;
val TRAVEL = 5;
val NEXT_PACKET = 0;
fun Wait() = (2*TRAVEL)+ACK_WAIT+1;
```

Pri izvajanju simulacije model dopušča uporabniku nekaj svobode pri izbiri določenih parametrov. Parametri, ki jih lahko uporabnik nastavi, so prikazani v kodi 5.5 in so:

- *Correct* – odstotek paketov (interval med 0.0 - 1.0), ki se med prenosom pravilno prenese in se jih ne izgubi; ker v modelu ni nekega mehanizma za zaznavo pokvarjenih paketov jih upoštevamo skupaj z izgubljenimi paketi; zato odstotek pravilno prenesenih paketov ustrezno zmanjšamo;
- *MAX_SEQ* – največje zaporedno število, ki se uporabi za oštevilčenje paketov pri prenosu in se izračuna po formuli 5.1;
- *WindowSize* – največja dovoljena velikost okna, ki se izračuna po formuli 5.2;
- *ACK_WAIT* – čas, ki mora preteči preden se potrditev pošlje v ločenem okvirju; če se potrditev pošlje skupaj s paketom se časovnik za poslani potrditveni paket ustavi;
- *TRAVEL* – čas, ki je potreben, da paket pripotuje do sprejemnika;
- *NEXT_PACKET* – čas, ki je potreben za pripravo novega paketa;
- Wait() - čas, ki preteče preden se paket ponovno pošlje;

Za parametre *ACK_WAIT*, *NEXT_PACKET* in *TRAVEL* se lahko uporabijo tudi rezultati poljubnih funkcij. Vendar je potrebno paziti, da se paketi med pošiljanjem ne začnejo prehitevati. Paketi, morajo prihajati v takem zaporedju, kot so bili ustvarjeni. Če pošiljatelj želi poslati pakete 0, 1 in 2 morajo tudi časi sprejema okvirja ustrezati zaporedju. Čas prihoda posameznega okvirja, ki ga dodeljuje akcija *Travel time* na sliki

5.8, mora ustrezati pogoju

$$t_0 < t_1 < t_2 < \ldots < t_n. \tag{5.3}$$

V primeru, da se omenjena omejitev ne upošteva, lahko pride do napak v prenosu, kot so izguba paketov, podvajanje paketov in tudi smrtni objem (angl. *deadlock*). Eden izmed možnih scenarijev, kjer pride do smrtnega objema, če se zanemari pogoj 5.3, je:

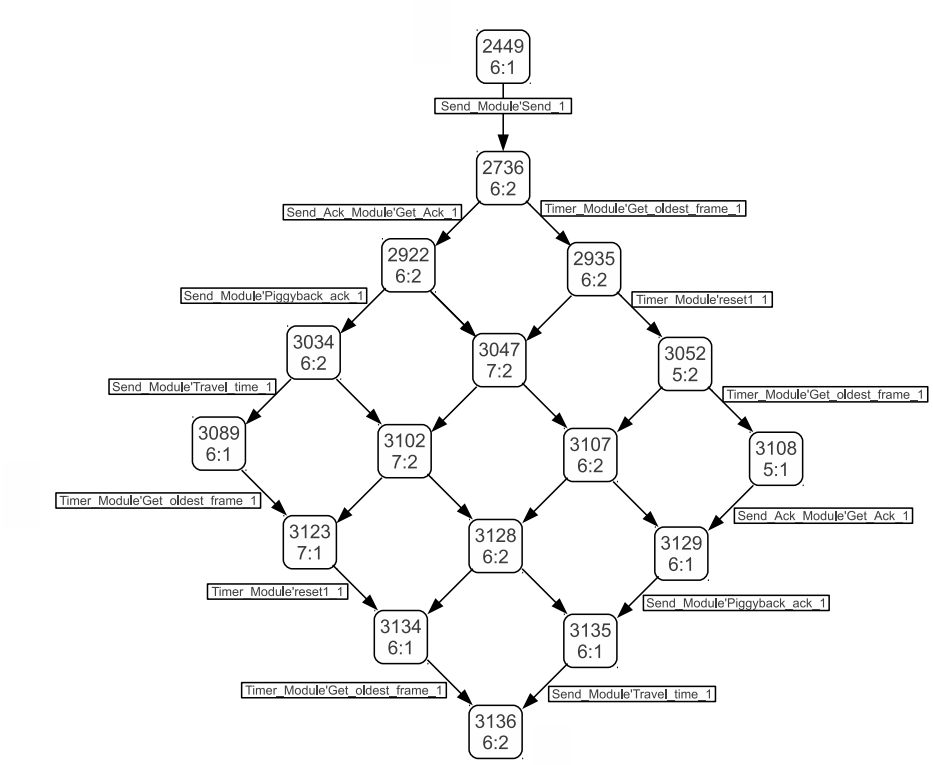
- $MAX\_SEQ = 3$, $WindowSize = 2$;
- sprejemnik B in pošiljatelj A imata okno nastavljeno na paketa 2 in 3;
- sprejemnik B ima v predpomnilniku paket 3 vendar čaka na paket 2;
- pošiljatelj B pošlje svoj paket *PackX*@100 (paket bo sprejet v času 100 na drugi strani povezave) sprejemniku A; skupaj s paketom se pošlje tudi potrditev za zadnji pravilno sprejeti paket tj. 1;
- sprejemnik B, takoj za tem, sprejme paket 2, zato pošlje potrditev za oba sprejeta paketa 2 in 3 hkrati, tj. *Ack3*@99 (zanemarimo pogoj 5.3); B pomakne svoje okno na paketa 0 in 1;
- v času @99 dobi sprejemnik A potrditev *Ack3*, zato pošiljatelj A pomakne svoje okno na paketa 0 in 1;
- pošiljatelj A pošlje paket 0@200;
- v času @100 sprejemnik A dobi paket *PackX* s potrditvijo paketa 1; potrditev paketa pomeni premik okna na zaporedje paketov 2 in 3 pri pošiljatelju A (napaka, A je preskočil paket 1);
- sedaj skuša pošiljatelj A poslati paketa 2 in 3 sprejemniku B, ne da bi poslal paketa 1;
- pošiljatelj A pošlje paket 2@201;
- sprejemnik B sprejeme paket 0@200, vendar zavrne paket 2@201 (ker ni v oknu 0 in 1);
- sedaj pošiljatelj A vedno znova pošilja paket 2, medtem ko sprejemnik B pričakuje sprejem paketa 1 (smrtni objem), ki ga je pošiljatelj preskočil zaradi prepoznega sprejema potrditve, ki je prišla s paketom *PackX*;

V zgoraj opisanem primeru smrtnega objema vidimo, da je težava v tem, da se okvirji med seboj prehitevajo. Okvir, ki se je poslal kasneje, se je obravnaval pred ostalimi okvirji in zato pride do napake v modelu. Zato smrtni objem z ustreznimi časovnimi pogoji ni mogoč.

5.3.2 Dosegljivost stanj

Ko nastavimo želene parametre, lahko začnemo z izvajanjem simulacije, ki jo izvajamo poljubno dolgo. Nastavimo lahko, koliko korakov naj se v simulaciji izvede, preden se ta ustavi, ali v katerem času naj se simulacija ustavi. Simulacijo lahko ustavimo tudi ob poljubnem dogodku z uporabo nadzornikov za ustavljanje simulacije (angl. *breakpoint monitor*). Po končanem izvajanju se simulacija ustavi v nekem stanju, ki je lahko za simulirani model pravilno oz. nepravilno. V primeru, da se model izvaja nepravilno, si zelo težko pomagamo samo z izvajanjem simulacije, ker ne vemo točno kdaj je do napake prišlo oz. pod kakšnimi pogoji. Zato si pomagamo z orodjem za prikaz prostora stanj. Njegova uporaba je prikazana v razdelku 4.4.

Slika 5.12 prikazuje primer delnega izrisa prostora stanj za primer iz slike 5.8. Prikazuje namreč, katere akcije se lahko sprožijo za tem, ko se paket pojavi v pogoju *Buffer* in preden se pretvori v okvir, ki se pojavi v pogoju *Network*. Na vrhu je prikazano vozlišče s številko 2449. Vozlišče ustreza stanju, ko se v pogoju Buffer pojavi paket (0,"d0_",0). Vidimo tudi, da ima vozlišče 2449, 6 predhodnikov in enega naslednika. S proženjem akcije *Send* v modulu *Send Module*, pridemo v vozlišče 2736, ki ustreza stanju v CPN modelu po sprožitvi omenjene akcije. V tem stanju se lahko zgodita dve stvari. Prva je pridobitev potrditve za zadnji še pravilno sprejeti paket, ali pa se začne inicializacija časovnika za paket (0,"d0_",0). Recimo, da se izvede akcija *Get Ack* v modulu *Send Ack*. Tako pridemo v vozlišče 2922. Vozlišče 2922 bi ustrezalo stanju, ki je prikazano na primeru slike 5.8. Podobno velja za vozlišča 3047, 3107 in 3129. Vozlišča prav tako prikazujejo stanje na sliki 5.8, vendar s to razliko, da so se prej izvedle določene akcije za inicializacijo časovnika, ki na primeru 5.8 niso vidne, ker se ne nahajajo v istem modulu. Na povezavah, kjer ni opisa velja, da se proži ista akcija kot en nivo višje. Na primer iz vozlišča 2922 v vozlišče 3047 pridemo s proženjem iste akcije, kot med vozliščema 2736 in 2935, tj. akcije *Get oldest frame* v modulu *Timer module*. Podobno velja tudi za ostale povezave, kjer manjka opis na povezavi. Iz vozlišča 2922 nato pridemo v vozlišče 3034 tako, da se paket spremeni v okvir s pomočjo akcije *Piggyback ack* v modulu *Send*



Slika 5.12: Dosegljivost stanj po prihodu paketa (0,"d0_",0) v pogoj *Buffer* na sliki 5.8.

Module, ki pripne potrditev *Ack* za zadnji sprejeti paket. Ostalo je samo še, da se ustvarjenemu okvirju določi čas v katerem bo dosegel sprejemnika. To se zgodi s proženjem akcije *Travel time* v modulu *Send Module*, ki nas pripelje v vozlišče 3089. Sedaj se lahko izvedejo samo še inicializacije časovnika za poslani paket (0,"d0_",0), kar nas pripelje v vozlišče 3136, ki predstavlja stanje v primeru na sliki 5.9.

Na primeru 5.12 vidimo, da smo z orodjem za izdelavo prostora stanj, dobili možne pogoje proženja od enega do drugega stanja v mreži. Tako z analizo prostora stanj vidimo pod katerimi pogoji je neko stanje dosegljivo. Še več, izrišemo si lahko čisto vse možne pogoje, ki pripeljejo v neko izbrano stanje. Če izbrano stanje predstavlja napako v modelu, jo lahko na ta način zelo hitro odpravimo.

5.3.3 Simulacijski rezultati

Koda 5.5 prikazuje vrednosti parametrov, ki so bili nastavljeni med osnovno simulacijo. Parameter *TRAVEL* določa čas potovanja okvirja od oddajnika do sprejemnika, ki zahteva 5 časovnih enot. *ACK_WAIT* določa, da oddajnik počaka 1 časovno enoto, preden pošlje potrditev za paket v ločenem okvirju. Funkcija *Wait()* določa, da oddajnik počaka 12 časovnih enot, preden skuša ponovno poslati paket. Če je paket potrjen prej kot v 12 časovnih enotah, se časovnik za paket ustavi. Vrednost parametra *NEXT_PACKET* je 0, kar pomeni, da se naslednji paket pošlje takoj, ko je na voljo. Vrednosti preostalih parametrov *Correct*, *MAX_SEQ* in *WindowSize* pa se določijo za vsako izvajanje simulacije posebej.

Najprej poglejmo, če se model obnaša po pričakovanjih. To najlažje preverimo tako, da velikost okna *WindowSize* in največjo zaporedno številko *MAX_SEQ*, ki se lahko dodeli paketu, nastavimo na vrednost 1. Tako nastavljeni parametri predstavljajo protokol "ustavi in čakaj". Pri protokolu "ustavi in čakaj" lahko hitro izračunamo, s kakšno hitrostjo se prenašajo paketi. Pošiljanje paketa po liniji, glede na prej določene parametre, traja 5 enot. Obhodni čas enega paketa (angl. *round trip time*) znaša najmanj 10 časovnih enot, če se potrditev pošlje v okvirju skupaj s paketom in največ 11 časovnih enot, če se potrditev za paket pošlje v ločenem okvirju. Da bi dosegli omenjene čase se morajo paketi prenašati z 100% zanesljivostjo, kar pomeni, da je vrednost parametra *correct* enaka 1.0.

Tabela 5.1: Simulacijski rezultati za protokol "ustavi in čakaj".

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
DuplicatePackets	0	0	0.000000	0	0
NAckedPackets	0	0	0.000000	0	0
PacketLoss	1999	1999	1.000000	1	1
PacketTravelTime	1000	5000	5.000000	5	5
PacketsResent	0	0	0.000000	0	0
TimeBetweenPackets	1000	10994	10.994000	5	11
TimeOutPackets	0	0	0.000000	0	0

Tabela 5.1 prikazuje rezultate simulacijskega poročila za protokol "ustavi in čakaj", ki nam ga po koncu simulacije ustvari orodje CPN Tools. Pogoj za ustavitev simulacije je 1000 prejetih paketov in velja za vse izvedene simulacije. V polju *Name* je prikazano ime monitorja, ostala polja pa prikazujejo statistiko, ki jo je monitor med izvajanjem simulacije pridobil. Monitor *DuplicatePackets* kaže, da sprejemnik ni dobil nobenega podvojenega paketa, *NackedPackets* pa kaže, da pošiljatelj ni dobil nobene zahteve po ponovnem prenosu paketa od sprejemnika. Monitor *PacketLoss* prikazuje, da se paketi v povprečju prenašajo z 100% zanesljivostjo (polje *Avrg*). *PacketTravelTime* prikazuje, da paket v povprečju potuje 5,0 časovnih enot, kar je tudi pričakovani rezultat. *PacketResent* (polje *Count*) kaže skupno število paketov, ki jih je oddajnik ponovno poslal. Monitor *TimeBetweenPackets* prikazuje, da je sprejemnik dobil nov paket v povprečju na 10,994 časovnih enot. Točna vrednost 11 se nikoli ne doseže, ker prvi poslani paket potuje 5 časovnih enot, kar je prikazano v polju *Min*, vsi ostali paketi pa nato pridejo na vsakih 11 časovnih enot. Monitor *TimeOutPackets* prikazuje, da ni bil noben paket ponovno poslan zaradi izteka časovnika.

Tabela 5.2: Simulacijski rezultati za protokol "ustavi in čakaj". Verjetnost uspešno prenesenih paketov je 85%.

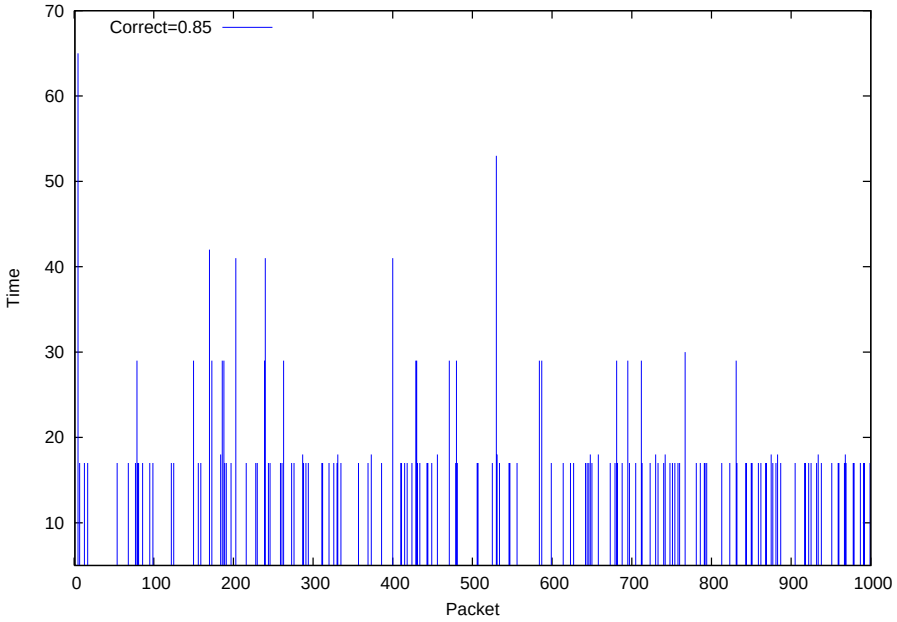
Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
DuplicatePackets	98	98	1.000000	1	1
NAckedPackets	0	0	0.000000	0	0
PacketLoss	2069	1756	0.848719	0	1
PacketTravelTime	1000	7284	7.284000	5	65
PacketsResent	306	152386	497.993464	4	998
TimeBetweenPackets	1000	14242	14.242000	5	71
TimeOutPackets	306	306	1.000000	1	1

Kako sedaj na prenos vpliva povečanje okna? Po [7] velikost okna vpliva samo na čas prihoda med posameznimi paketi pri sprejemniku, ki se zmanjšuje z večanjem okna, medtem ko se čas prenosa posameznega paketa ne more izboljšati. Slika 5.13 b prikazuje vpliv okna na izboljšanje medprihodnih časov posameznih paketov pri sprejem-

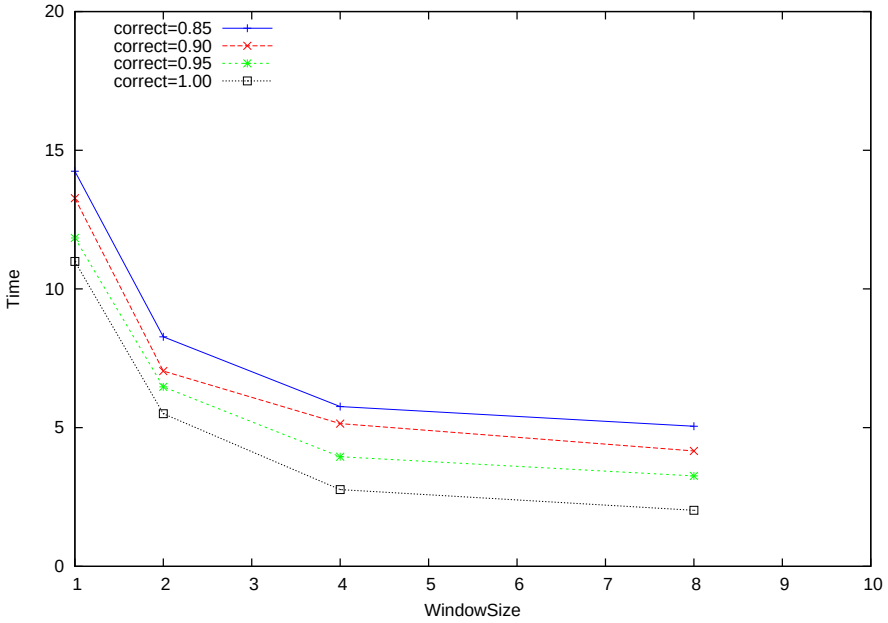
niku. Povečanje velikosti okna (za prenose brez izgub, *correct=1.0*) na 2 paketa izboljša čas prihoda med paketi iz 10,994 časovnih enot na 5,505 enot, kar predstavlja 50% pohitritev. Če se okno poveča na 4, se čas zmanjša na 2,766 enot, kar prinese še dodatno 50% pohitritev v primerjavi z velikostjo okna 2. Okno z velikostjo 8 izboljša čas na 2,02 enote, kar je tudi spodnja meja za simulirani model. Skupna pohitritev tako znaša 81,6%.

Do sedaj so se paketi prenašali pod idealnimi pogoji, kjer ni nobenih izgub ali okvar pri prenosu. Tabela 5.2 prikazuje statistiko, kjer je velikost okna enaka 1, vendar so izgube in okvare pri prenosih 15% (parameter *correct = 0.85*). Vidimo, da so se paketi pravilno prenesli v 84,8%. Čas potovanja paketa se je povečal na 7,284 časovnih enot, kar je za 45,7% slabše od idealnih pogojev. Najpočasnejši paket je prišel šele po 65 časovnih enotah od časa, ko je bil prvič poslan. Čas prihoda med posameznimi paketi se je povečal iz 11 na 14,242 časovnih enot. Zaradi izgub je bilo potrebno ponovno poslati 306 paketov. Od tega se je 306 paketov ponovno poslalo, ker potrditev ni prišla pred iztekom časovne omejitve. 98 paketov je bilo med prenosom podvojenih. Na sliki 5.13 a vidimo, kateri paketi so se izgubili in koliko časa so potrebovali, da so prispeli do sprejemnika.

Prenosi z izgubami nam tako poslabšujejo hitrost prenosa. Večja kot je izguba, slabši so časi, vendar lahko s povečanjem okna dosežemo boljše prenosne čase in omilimo posledice, kar je prikazano z grafom na sliki 5.13 b. Ne glede na količino izgubljenih paketov, nam okno v vsakem primeru izboljša in pohitri prenos paketov, oz. zmanjšuje čas med posameznimi prihodi paketov na strani sprejemnika.



(a) Izgubljeni paketi in čas prihoda izgubljenega paketa. Verjetnost uspešnega prenosa je enaka 85%.



(b) Vpliv velikosti izbranega okna na hitrost sprejemanja paketov, pri različnih izgubah.

Slika 5.13: Statistika protokola izbirno ponavljanje.

6 Zaključek

V diplomskem delu je prikazano, kako lahko s pomočjo orodja *CPN Tools* ustvarimo nek poljuben model (v našem primeru je to protokol drsečega okna) z osnovnimi gradniki barvnih Petrijevih mrež. Postavljeni model nato testiramo s pomočjo izvajanja simulacije in s tem ugotovimo ali se model obnaša po pričakovanjih. V primeru napačnega izvajanja postavljenega modela je potrebno poiskati napake, ki povzročajo težave v izvajanju. Napake se pojavljajo predvsem zaradi prevelikega števila stanj v katerih se lahko model nahaja, razvijalec pa zelo težko predvidi vsa mogoča izvajanja v modelu. Da bi odkrite napake odpravili, je potrebno simulacijo ustaviti v stanju, ki predstavlja napako v modelu. To storimo z uporabo nadzornikov. V stanju, ko se simulacija ustavi, naredimo nato analizo prostora stanj in izrišemo graf, ki nam pokaže katere akcije so vodile do napačnega stanja. Ko napako enkrat odkrijemo, model ustrezno popravimo. Po odpravi napak izvedemo še performančno analizo modela, da bi ugotovili ali se model izvaja zadosti hitro in ali izpolnjuje neke vnaprej določene zahteve.

Ustvarjen model za protokol drsečega okna se po izvedeni simulacij in performačni analizi obnaša po pričakovanjih. Vendar v modelu obstaja posplošitev, tj. način kako je

realiziran modul, ki predstavlja povezavo med dvema napravama. Na povezavi se paketi lahko samo izgubijo, medtem ko se po [7] paket tudi lahko pokvarijo. To vpliva predvsem na to, da bi se lahko sprejemnik prej odzval na sprejeti pokvarjeni paket z negativno potrditvijo kot pa, da sprejemnik ne sprejme nobenega paketa in se izguba paketa šele zazna pri oddajniku, ko poteče časovnik za poslani paket, ali pa po prihodu paketa, ki sledi izgubljenemu paketu. Rešitev bi bila, da se pri pošiljanju paketa v okvir zapiše še informacija, ki bi preprečila morebitne napake (CRC koda, angl. *cyclic redundancy check*), med prenosom paketa pa bi se lahko zgodilo tudi to, da se njegova vsebina spremeni. Tako spremenjen paket dobi sprejemnik, ki bi potreboval dodaten modul za preverjanje napak. Če se ugotovi, da se je napaka pri prenosu zgodila, bi sprejemnik poslal negativno potrditev za paket, sprejeti pokvarjeni paket pa bi se zavrgel.

Eden izmed glavnih ciljev poleg prikaza izdelave modela in uporabe orodja *CPN Tools* je tudi odgovoriti na vprašanje; "Ali je orodje primerno za uporabo in načrtovanje sistemov in ali je razvijalcu orodje v kakšno pomoč pri razvoju sistema?" Odgovor je da. Med izdelavo modela za protokol drsečega okna se je izkazalo, da je model med simulacijo v določenih trenutkih izvajal ravno take napake, kot jih je med razlago protokola v [7] navedel avtor. Tudi če način odprave napake ne bi bil naveden v [7], bi lahko s pomočjo analize prostora stanj hitro odkrili kje v modelu je bila napaka storjena in jo potem tudi ustrezno popravili.

LITERATURA

[1] James Lyle Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR, NJ, 1981

[2] Winfrid G. Schneeweiss, *Petri Nets for Reliability Modeling (in the Fields of Engineering Safety and Dependability)*, LiLoLe-Verlag GmbH, Hagen, 1999.

[3] Kurt Jensen, Lars Michael Kristensen, Lisa Wells, *Coloured Petri Nets and CPN Tools for modeling and validation of concurrent systems*, Springer-Verlag, marec 2007, str. 213-254

[4] Lars M. Kristensen, Søren Christensen, Kurt Jensen, *The practitioner's guide to coloured Petri nets*, Springer-Verlag, 1998, str. 98-132

[5] Wil van der Aalst, *CPN, A concrete language for high-level Petri nets*, Technische Universiteit Eindhoven. Dostopno na:
http://cpntools.org/_media/book/cpn.pdf

[6] Kurt Jensen, Søren Christensen, Lars M. Kristensen, *CPN Tools State Space Manual*, University of Aarhus. Dostopno na:
http://cpntools.org/_media/documentation/manual.pdf

[7] Andrew S. Tanenbaum, *Computer Networks, Fourth Edition*, Prentice Hall PTR, 2003, str. 149-170