Grafisk presentasjon av GPenSim-simulering

Karsten Hofsmo

Våren 2009

Sammendrag

GPenSim er et verktøy for modellering og simulering av diskret hendelsesystemer (DES). GPenSim er integrert i Matlab-plattformen, og har dermed tilgang til innebygde Matlab-funksjoner som plot etc. I GPenSim blir Petri net-grafen definert i Petri net-definisjonsfiler. Resultatet av en simulering blir vist i tekst. Oppgaven gikk ut på å utvikle et verktøy som skulle presentere både Petri net-grafen og simuleringsresultatet grafisk. En grafisk presentasjon viser tydeligere sammenhengen mellom graf og simuleringsresultat.

I GPenSim kan systemer deles opp i moduler. Med denne fremgangsmåten kan modellen av komplekse systemer bli mer oversiktlig. Ved å bruke dette i den grafiske presentasjonen ble også tegningen av grafen mer oversiktlig. Det kan være vanskelig å få oversikt over resultatet fra en simulering dersom det bare presenteres i tekst. I denne oppgaven ser en at en grafisk presentasjon kombinert med tekst, vil gjøre analyser av resultatet enklere.

Implementasjonen av verktøyet bak den grafiske presentasjonen er i Java. Java2D er brukt for å lage grafikken. Java2D tilbyr mulighet for utskrift av skjermbildet til skriver. Petri net-grafen til et system kan dermed skrives ut og brukes i dokumentasjon.

Innholdsfortegnelse

Kapittel 1

Introduksjon

GPenSim er et verktøy for modellering og simulering av diskret hendelsesystemer. GPenSim har ingen metode for å presentere Petri net-grafen til et system grafisk. GPenSim har heller ingen metode for å presentere resultatet fra simulering av et system grafisk. Dess større system som modelleres og simuleres, dess vanskeligere er det å få oversikt over både graf, og resultat fra simuleringen, når tekst er eneste alternativ presentasjonsmiddel.

I oppgaven utvikles et verktøy som skal presentere Petri-net grafen og simuleringsresultatet grafisk. Petri net-grafen blir i GPenSim definert i Petri net-definisjonsfiler. Etter innlesing av definisjonsfilene blir en tegning av grafen generert på skjermen. Grafen skal være oversiktlig dersom det er mulig. Modellen til komplekse systemer kan bli uoversiktlig og vanskelig å holde orden på. I GPenSim kan systemer deles opp i moduler. I rapporten ser en om fremgangsmåten med moduler, også kan være med på å gjøre tegningen av grafen mer oversiktlig.

I andre kapittel introduseres Petri net. En presenteres for en skisse av en Petri net-graf, og ser hvilke element som inngår i grafen.

I kapittel 3 omtales GPenSim. En ser hvordan et system modelleres og simuleres i GPenSim. En presenteres for resultatet av en simulering, og ser hva en grafisk presentasjon kan tilføre.

Kapittel 4 beskriver GPenSimGUI-en. Først introduseres en metode for å overføre simuleringsresultatet fra Matlab til GPenSimGUI-en. Videre introduseres kjerneelementene som inngår i programmet og koblingene mellom de. Elementene skal plasseres på skjermen. En beskriver derfor en metode for å dele opp skjermen i områder. Ved utplassering tildeles områdene moduler. Etter å ha sett hvor elementene kan bli plassert, beskrives en algoritme for å plassere dem på skjermen. En ser også hvordan moduler kan organiseres. Til slutt introduseres en utskriftsfunksjon.

I kapittel 5 brukes et eksempel for å beskrive hvordan GPenSimGUI-en virker. En sammenligner en presentasjon gitt i tekst med en grafisk presentasjon.

En konklusjon blir til slutt gitt i kapittel 6.

Kapittel 2

Petri net

Petri net består av to deler. En statisk del og en dynamisk del. Den statiske delen inneholder en Petri net-graf. Den dynamiske delen inneholder en start tilstand og en transition-mekanisme som flytter tokens rundt i nettet, og dermed skifter tilstand. Dette kapittelet er inspirert av [1,2].

2.1 Statisk del

En Petri net-graf består av 3 typer element. Places, transitions og arcs. Arcs er koblinger mellom place og transition. Arcs kan ikke gå fra place til place eller fra transition til transition. Arcs har en vekt som bestemmer hvor mange tokens som flyttes når en hendelse skjer.

Hendelser er representert av transitions. For at en hendelse skal kunne skje må nødvendige betingelser være oppfylt. Places er assosiert med disse betingelsene. Videre er inngang-places assosiert med de nødvendige betingelsene for at en handling skal kunne skje. Mens utgang-places er assosiert med nye betingelser som har oppstått på grunn av hendelsen som tok sted. En transition *t*s inngangs-places er places som har en arc fra seg til *t*. *T*s utgangs-places er places som har en arc koblet fra *t* til seg. Dette viser av figuren under. Places er tegnet som sirkler. Transitions er tegnet som fylte rektangel og arcs som piler. En hendelse eller transitions som kan skje sies å være enabled. Å si at en transition skyter er det samme som å si at en hendelse skjer. Disse begrepene er kjent i Petri net-terminologien og brukes i denne rapporten.

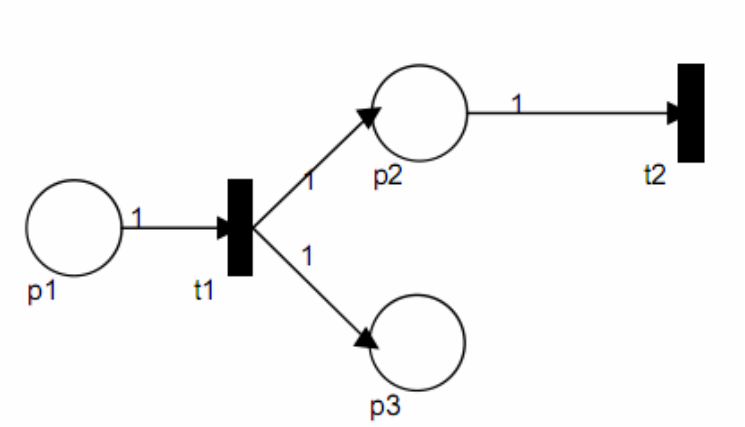


Fig. 2.1: Enkel Petri net-graf.

I figur 2.1 har t1 1 inngang-place og 2 utgang-places. En trenger noe som kan fortelle om de nødvendige betingelsene er oppfylt eller ikke. Dette noe er tokens. Tokens blir flyttet

rundt i grafen når transitions skyter. Tokens representeres med svarte fylte sirkler i places. Petri nettets tilstand beskrives av antall tokens som de ulike places har.

En transition er i følge definisjonen enabled når:

X(pi) >= w(pi,tj) for alle pi E I(tj).

Altså må antall tokens i en inngang-place til tj være større enn eller lik vekten på arc-en som kobler inngang-place-en til tj. Dette må være sant for alle inngang-places til tj.
I figur 2.1 er ingen transitions enabled. I figur 2.2 er t1 enabled.

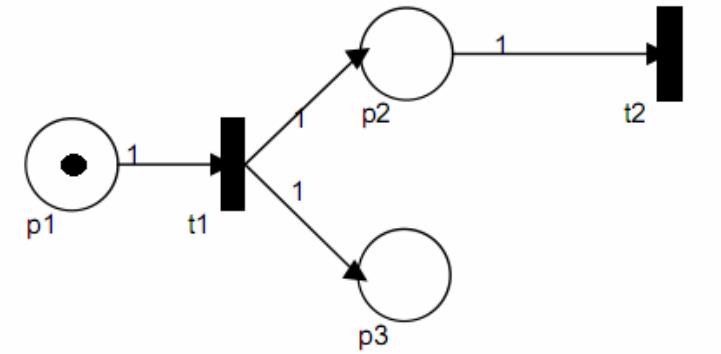


Fig. 2.2: Enkel Petri net-graf som inneholder en enabled transition.

2.2 Dynamisk del

Når transitions skyter flyttes tokens og dermed skifter Petri nettets tilstand. Antall tokens som fjernes fra inngang-places er lik vekten på arc-en som kobler de sammen. På samme måte blir antall tokens i utgang-places bestemt av vekten på arc-en mellom transition og utgang-place. I figur 2.2 har t1 en inngang-place p1. Arc-en mellom dem har en vekt lik 1. Siden p1 inneholder en token er t1 enabled. Resultatet av at t1 skyter viser i figur 2.3. t1 er ikke lenger enabled, men nå er betingelsene for transition t2 oppfylt, og t2 er dermed enabled.

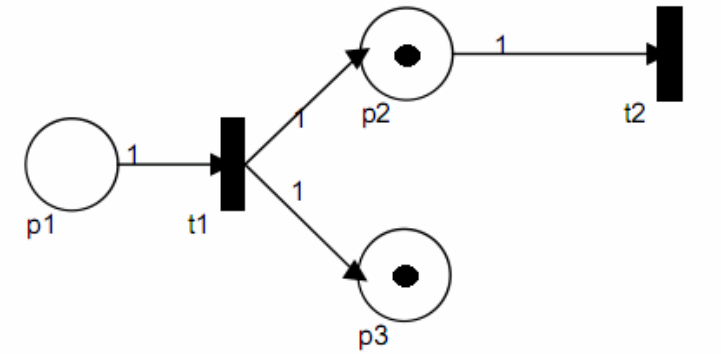


Fig. 2.3: Enkel Petri net-graf etter transition t1 har skjedd.

Kapittel 3

GPenSim

Det er utviklet flere Petri net-verktøy. Fordelen med GPenSim er at det er enkelt å bruke, har en enkel syntaks og er integrert i Matlab-plattformen. Matlab har et velkjent programmeringsspråk og gir GPenSim støtte for integrerte funksjoner og verktøy som fuzzy logic, kontrollsystemer, statistikk etc. Dette avsnittet er i stor grad hentet fra [3].

For å simulere et system i GPenSim trengs et minimum av to filer. Informasjon om den statiske delen (Petri net-grafen) lagres i en Petri net-definisjonsfil. Mens informasjon om den dynamiske delen lagres i en hovedsimulasjonsfil (Main Simulation File). Dersom systemet er delt opp i moduler, lagres informasjon om hver modul i egne definisjonsfiler. Transitions skyter når nødvendige betingelser beskrevet av tokens er oppfylt. Tilleggsbetingelser kan defineres og lagres i transition-definisjonsfiler.

Transition-definisjonsfiler blir kalt under simulering. De er ikke nødvendige i den grafiske presentasjonen av simuleringsresultatet. Transition-definisjonsfiler leses derfor ikke inn i GPenSimGUI-en og er heller ikke nærmere diskutert i rapporten. Mer informasjon om transition-definisjonsfiler finnes i[3].

I kapittel 3.1 modelleres et enkelt system med GPenSim. Resultatet fra simulering av systemet blir presentert i tekst. Resultatet blir også lagret i en variabel. 3 av feltene i variabelen er nødvendige for å kunne gi en grafisk presentasjon av simuleringsresultatet i GPenSimGUI-en.

3.1 Modellering av et enkelt system i GPenSim

En Petri net-graf består av places, transitions og arcs. Disse elementene blir i GPenSim definert i Petri net-definisjonsfiler. For et lite system kan en definisjonsfil være nok. Dersom systemet er stort, kan moduler være med på å gjøre modellen mer oversiktlig. Da har hver modul en egen Petri net-definisjonsfil. Modulene blir koblet sammen i hovedsimulasjonsfilen.

Først opprettes Petri net-definisjonsfilen(e). Deretter legges dynamisk informasjon i en hovedsimulasjonsfil. Eventuelle transition-definisjonsfiler opprettes også. Petri net-grafen til systemet i figur 3.1 defineres i Petri net-definisjonsfilen under. Systemet er kjent fra GPenSim-dokumentasjonen.

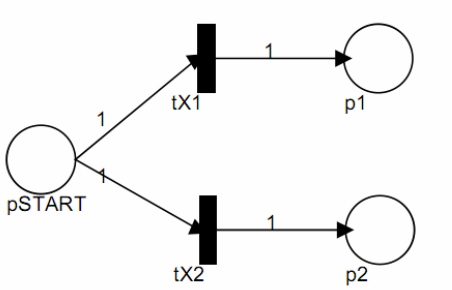


Fig. 3.1: En enkel Petri net-graf.

```
function [PN_name, set_of_places, set_of_trans, set_of_arcs]...
                        = loadbalance_def(global_info)
% file: loadbalance_def.m:

PN_name='Web Server Load Balancer';

set_of_places={'pSTART', 'p1', 'p2'};
set_of_trans={'tX1','tX2'};

set_of_arcs={'pSTART','tX1',1, 'tX1','p1',1,...
             'pSTART','tX2',1, 'tX2','p2',1};
```

3.2 Simulering av et enkelt system i GPenSim

I dette systemet skifter transition tX1 og tX2 på å skyte. Dette er spesifisert i transition-definisjonsfiler. I hovedsimulasjonsfilen er det definert at pSTART skal starte med 10 tokens. Simuleringen av systemet gir følgende resultat:

State:0 Time: 0
Initial State:
pSTART p1 p2
10 0 0
At time: 0 enabled transtions are:
tX1 tX2
At time: 0 firing transtions are:
tX1

State: 1 Time: 10
Fired Transition: tX1
Current State:
pSTART p1 p2
9 1 0
At time: 10 enabled transtions are:
tX1 tX2
At time: 10 firing transtions are:
tX2

State: 2 Time: 25
Fired Transition: tX2
Current State:
pSTART p1 p2
8 1 1
At time: 25 enabled transtions are:
tX1 tX2
At time: 25 firing transtions are:
tX1
…
…
…

Når simuleringen er ferdig lagres resultatet i en variabel i Matlab. Siden GPenSimGUI-en ikke utfører selve simuleringen, må programmet lese inn deler av informasjonen som er lagret i denne variabelen. Navn på variabelen defineres i hovedsimulasjonsfilen. Sim-variabelen fra systemet over viser under. Som en kan se inneholder variabelen mye informasjon. Ikke all informasjon er nødvendig for gi en grafisk presentasjon av simuleringen.

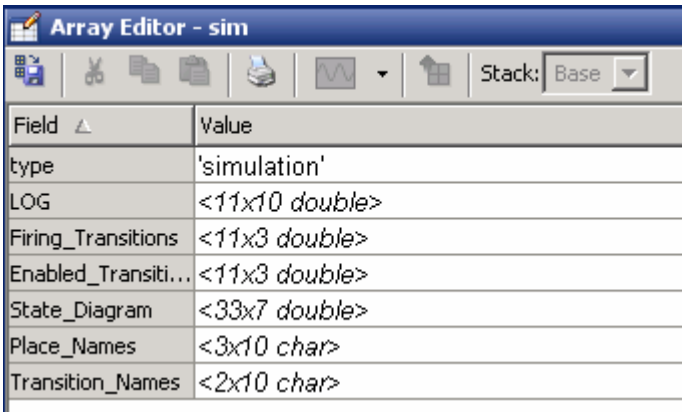
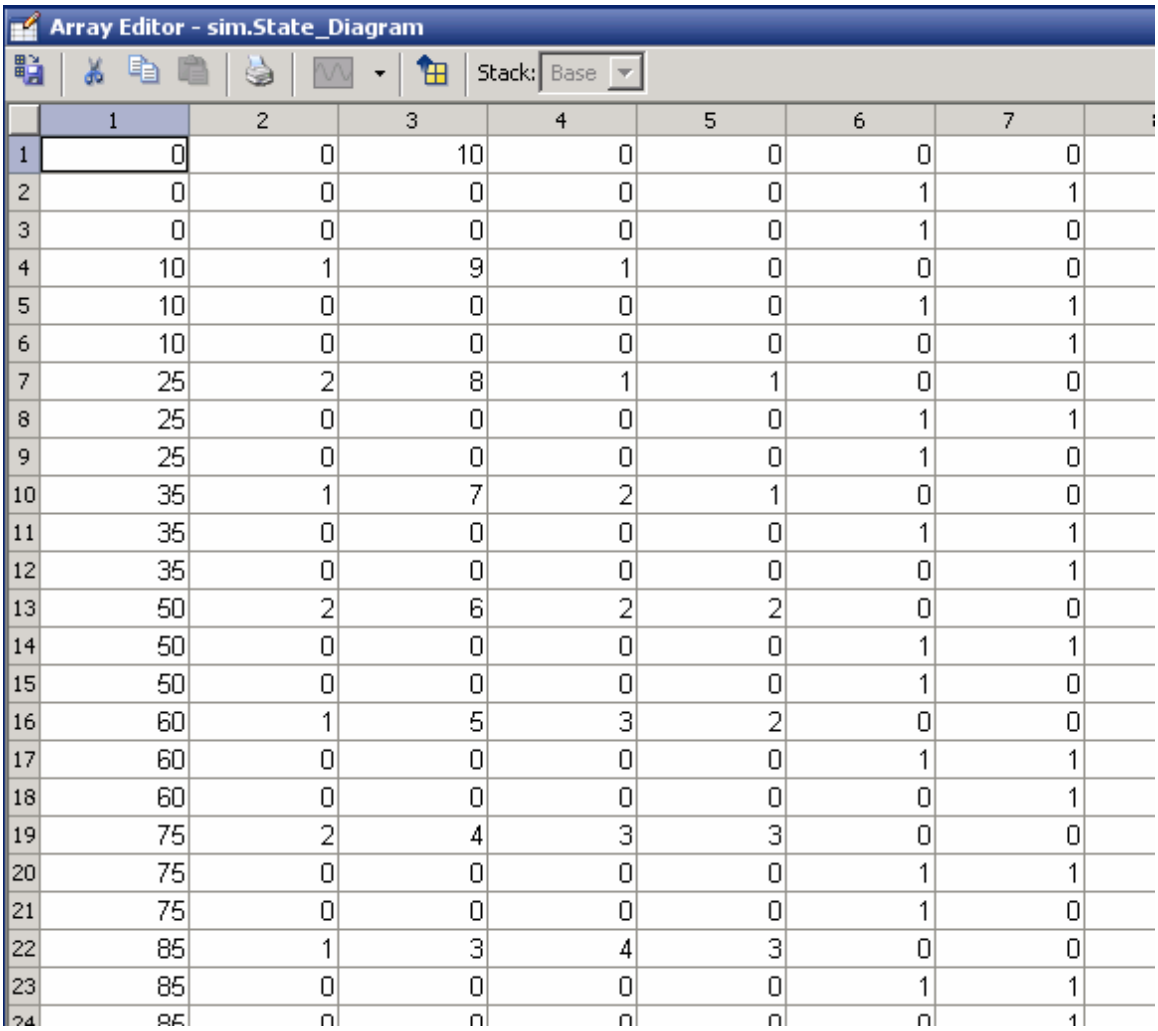


Fig. 3.2: Sim-variabelen fra simulering av enkelt system.

Variabelen har flere felt. Simuleringsresultatet er lagret i LOG-matrisen. Denne matrisen inneholder også informasjon som ikke er nødvendig for å lage en grafisk presentasjon av simuleringsresultatet. Nyere versjon av GPenSim har inkludert en matrise som er kalt State_Diagram. Denne matrisen inneholder nøyaktig den informasjonen som trengs for en grafisk presentasjon. Matrisen er strukturert og oversiktlig. Firing_Transitions viser hvilke transitions som skjøt og når. Enabled_Transitions viser som navnet sier når de ulike transitions var enabled. GPenSim-manualen beskriver informasjonen som er lagret i LOG-matrisen.

3.2.1 State_Diagram av en simulering

GPenSimGUI-en bruker 3 av matrisene fra simuleringsvariabelen. Programmet leser inn State_Diagram, Place_Names og Transition_Names. Simulering av systemet over gav et State_Diagram som i figur 3.3.



	1	2	3	4	5	6	7
1	0	0	10	0	0	0	0
2	0	0	0	0	0	1	1
3	0	0	0	0	0	1	0
4	10	1	9	1	0	0	0
5	10	0	0	0	0	1	1
6	10	0	0	0	0	0	1
7	25	2	8	1	1	0	0
8	25	0	0	0	0	1	1
9	25	0	0	0	0	1	0
10	35	1	7	2	1	0	0
11	35	0	0	0	0	1	1
12	35	0	0	0	0	0	1
13	50	2	6	2	2	0	0
14	50	0	0	0	0	1	1
15	50	0	0	0	0	1	0
16	60	1	5	3	2	0	0
17	60	0	0	0	0	1	1
18	60	0	0	0	0	0	1
19	75	2	4	3	3	0	0
20	75	0	0	0	0	1	1
21	75	0	0	0	0	1	0
22	85	1	3	4	3	0	0
23	85	0	0	0	0	1	1
24	85	0	0	0	0	0	1

Fig. 3.3: State_Diagram fra et enkelt system.

Første kolonne i matrisen viser tiden. Andre kolonne viser hvilken transition som gav tilstandsendringen. De neste kolonnene refererer til places. En kolonne per place. Verdiene i disse kolonnene forteller hvor mange tokens hver place har i denne tilstanden. De resterende kolonnene tilhører transitions.

Som en kan se ut fra matrisen har tre og tre rekker samme tid. Disse 3 rekkene hører til samme tilstand. Den første av disse rekkene viser hvor mange tokens hver place har i tilstanden. Verdien i andre kolonne, i rekken, viser som sagt hvilken transition som førte til den nye tilstanden. Kolonnene for transitions er ikke brukt og er alltid 0. I de to neste rekkene er det bare kolonnene for transitions som brukes. Den andre rekken beskriver

hvilke transitions som er enabled. Den tredje rekken beskriver hvilke transitions som skyter. Rekke nummer 1, 2 og 3 er starttilstanden. Her kan en lese hvor mange tokens hver place starter med.

Dersom en ser på rekke 7, 8 og 9 i figur 3.3 vet en følgende: I tid 25 har transition nr 2 skutt. Dette har ført til at places nr 1, 2 og 3 har henholdsvis 8, 1 og 1 tokens hver. I rekke 8 ser en at både transition nummer 1 og transition nummer 2 er enabled. I rekke 9 ser en videre at transition nummer 1 skyter. I rekke nr 10 er transition nummer 1 ferdig i tid 35, og førte til en ny tilstand med flyttede tokens. Place nummer 1 har mistet 1 token, mens place nummer 2 har blitt tilført 1 token.

For at GPenSimGUI-en skal vite hvilke places som har hvor mange tokens, og hvilke transitions som er enabled og skyter, trenger den også å vite sammenhengen mellom plasseringen i matrisen, og navn på de ulike elementene. Place_Names er en liste over navn til places i den rekkefølgen som de står oppgitt i matrisen.Transition_Names er på samme måte en liste over navn til transitions. I figur 3.4 står pSTART oppført først. En vet da at pSTART refererer til kolonne 3 i figur 3.3. Den første opplistede transition er tX1 og kolonne 6 tilhører dermed tX1. Altså var det tX1 som førte til den nye tilstanden i rekke nr 10.

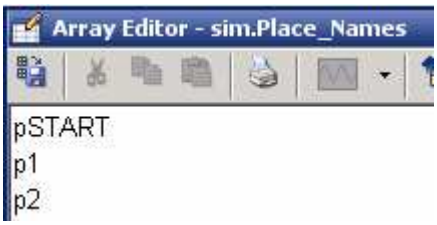


Fig. 3.4: Place_Names fra eksemplet.Fig. 3.5: Transition_Names fra eksemplet.

3.2.2 Lange navn blir forkortet

Det er ett punkt som er viktig å merke seg. I simuleringsresultatet blir navn til places og transitions forkortet dersom de overstiger 10 tegn (bokstaver, tall, ...). Forkortelsen går igjen i Place_Names- og Transition_Names-feltene. Dersom en ikke er oppmerksom på dette kan den grafiske presentasjonen gi en presentasjon av simuleringen som ikke er i samsvar med simuleringsresultatet. I figur 3.4 og 3.5 var ingen navn lengre enn 10 tegn og systemet ville dermed blitt presentert riktig grafisk.

Når en ny tilstand blir lest inn, sjekker programmet enten hvor mange tokens hver place har, eller listen over transitions for enabled eller firing transitions. For å skifte status på en transition eller oppdatere antall tokens i en place, søkes navnet på elementet opp i modulene. Det er viktig å oppdatere riktig element. Alle navn må være ulike for at en kan være helt sikker på at riktig element oppdateres.

I programmet tar en hensyn til at lengden på navn som overstiger 10 tegn vil bli forkortet til de 10 første tegnene. Dersom statusen på en transition med navn lengre enn 10 tegn skal oppdateres, vil programmet søke gjennom alle moduler etter en transition. Søket ville bare sjekket de 10 første tegnene på alle navn. Altså vil alt fungere som det skal, også med navn lengre enn 10 tegn, så lenge de 10 første tegnene i alle navn er ulike. Problemet oppstår når flere places eller transitions starter med 10 like tegn.

Eksempelvis kan et system ha 3 transitions. Disse transitions kalles tCrashAndRecoveryServer1, tCrashAndRecoveryServer2 og tCrashAndRecoveryServer3. I GPenSim vil navnene bli forkortet til tCrashAndR, tCrashAndR og tCrashAndR. Når GPenSimGUI-en skulle oppdatere tCrashAndRecoveryServer3, ville de to andre transitions også gå igjennom som den riktige transition-en å oppdatere. Det ville ikke vært mulig å skille de fra hverandre.

Det er derfor veldig viktig at de 10 første tegnene i alle navn er ulike. Et alternativ til navn i eksemplet over kunne vært: tServer1CrashAndRecovery, tServer2CrashAndRecovery og tServer3CrashAndRecovery. I GPenSim ville de tre transitions blitt forkortet til tServer1Cr, tServer2Cr og tServer3Cr. De tre forkortelsene er ulike. Den grafiske presentasjonen av simuleringen ville dermed vært i samsvar med simuleringsresultatet.

3.2.3 Simuleringsresultatet presenteres i tekst

Resultatet av simuleringen presenteres i form av tekst. Resultatet beskriver antall tokens i hver place, for alle tilstander. Videre er enabled transitions listet opp. Det samme er transitions som skyter. Tiden fra forrige tilstand til den nye tilstanden er tiden en transition brukte på å bli ferdig.

Dersom en tenker seg et komplekst system som er delt opp i flere moduler, kan systemet ha flere titalls places og transitions. Resultatet i GPenSim vil liste opp alle places og beskrive hvor mange tokens hver place inneholder. Ut fra en delmengde av tilstandene, ser en at antall tokens i en bestemt place vokser og vokser. En vil kanskje vite hvor denne place-en er i systemet. I hvilken modul er den definert? For å finne svaret på dette spørsmålet kan en bli nødt til å lete etter place-en i de ulike Petri net-definisjonsfilene. Dette er tungvindt. Grafisk vil en enklere se hvor place-en er. Grafisk ser en også koblingene (arcs-ene) mellom elementene.

Det er likeledes lettere å se hvor en bestemt transitions er i systemet. Grafisk ser en også hvilke places en transition er koblet til. Hvilke places måtte ha hvor mange tokens for at en bestemt transition var enabled? Hvilke inngang-places har en bestemt transition? Svar på disse spørsmålene og flere finner en i den grafiske presentasjonen.

Kapittel 4

GPenSimGUI

4.1 Innlesing av informasjon fra GPenSim

4.1.1 Innlesing av Petri net-graf

For å kunne tegne Petri net-grafen på skjerm, trenger GPenSimGUI-en informasjonen som er lagret i Petri net-definisjonsfilene. Definisjonsfilene er Matlab-filer og leses inn i programmet som vanlige tekstfiler. Dersom systemet er delt opp i moduler, leser GPenSimGUI-en én definisjonsfil per modul. Petri net-definisjonsfilen til et enkelt system er tatt med under. Systemet består av en klient som sender forespørsler til ulike tjenere. Klientforespørslene blir lagt i meldingsbuffere hos tjenerne.

```
1   function [PN_name, set_of_places, set_of_trans, set_of_arcs]...
2                       = sendReqs_def(global_info)
3
4   PN_name='Sending requests to servers';
5
6   set_of_places={'pClientRequests', 'pServer1MessageBuffer',
7         'pServer2MessageBuffer', 'pServer3MessageBuffer'};
8
9   set_of_trans={'tSendToS1','tSendToS2', 'tSendToS3'};
10
11  set_of_arcs={'pClientRequests','tSendToS1',1, ...
12        'pClientRequests','tSendToS2',1, ...
13        'pClientRequests','tSendToS3',1, ...
14        'tSendToS1', 'pServer1MessageBuffer', 1, ...
15        'tSendToS2', 'pServer2MessageBuffer', 1, ...
16        'tSendToS3', 'pServer3MessageBuffer',1};
```

GPenSimGUI-en leser definisjonsfilen linje for linje. Programmet tar hensyn til spesielle tegn. De to øverste tegnene i tabellen ignoreres.

%	Alle tegn bak % ignoreres. Dette er en kommentar.
\t	Er skjult i definisjonsfilen, men viser til innrykk i teksten.
[,]	Listen er tom.
{, }	Element som skal legges til i en liste er lagret innenfor klammeparanteser.
’	Navn på element er omringet av apostrof.
...	Element på neste linje skal legges til listen.
;	Listen har lagt til alle element.

GPenSimGUI-en oppretter datastrukturer for set_of_places, set_of_trans og set_of_arcs. Navnet på modulene lagres i strenger. Når strengen ”set_of_places” leses fra definisjonsfilen opprettes en liste for alle places. En place blir opprettet med navn og posisjon. Navnet leses fra definisjonsfilen. Posisjonen blir satt til punktet (0, 0). Posisjonen oppdateres når grafen blir plassert på skjermen. Transitions opprettes på samme måte.

Tidlig i utviklingsfasen ble filnavn for alle definisjonsfiler lest fra samme tekstfelt. Det førte til at arcs måtte bli opprettet etter alle places og transitions var lest inn og opprettet. Grunnen for dette ligger i definisjonsfilen som kobler sammen alle modulene. Denne definisjonsfilen inneholder bare arcs. For å opprette en arc må arc-ens place og transiton være opprettet på forhånd. Denne filen kunne da ikke leses inn før places og transitions som trengtes var lest inn og opprettet. Siden filene blir lest i den rekkefølgen de står oppgitt i tekstfeltet for filnavn, ville en få feilmeldinger dersom filen som kobler modulene ikke var oppført sist. Arcs blir derfor opprettet etter at alle modulene er lest inn.

I stedet blir det opprettet temporære lister med strenger for places, transitions og vekt. Arcs opprettes ved å gå igjennom listen og søke opp places og transitions i modulenes lister for places og transitions.

4.1.2 Innlesing av simuleringsresultat

Exportere simuleringsresultat til tekstfil

State_Diagram-, Place_Names- og Trans_Names-matrisene må overføres fra Matlab til GPenSimGUI-en. En måte å gjøre dette på er å eksportere matrisene til tekstfiler. Tekstfilene kan så leses av programmet. For at GPenSimGUI-en skal gjenkjenne resultatet for hver simulering, må fremgangsmåten for overføring være den samme hver gang. For å overføre resultatet fra Matlab til GPenSimGUI-en er det derfor laget en enkel prosedyre som må følges. Matlab-matrisene eksporteres til samme tekstfil. Ovenfor innholdet av hver matrise lagres en linje med tekst for gjenkjenning av matrisene. Eksempelvis vil tekststrengen ”Places” markere at informasjon om places blir lest herfra. Koden under oppretter en tekstfil som kan leses av programmet. Denne tekstfilen vil bli korrekt tolket.

OBS: For å vise en grafisk presentasjon av en simulering i GPenSimGUI-en, er det helt nødvendig at denne ekporteringsprosessen følges.

Filnavnet på tekstfilen kan selvfølgelig endres. Tekstfilen vil bli lagret i den katalogen som står som aktuell eller nåværende i Matlab. Dersom en har kalt simuleringsvariabelen noe annet enn sim må denne endres tilsvarende. Kodeutdraget på neste side kan enten skrives av linje for linje i Matlab, eller enklere lagres i en Matlab-fil, og kjøres etter simulering. Dersom en vil eksportere flere simuleringsresultat, er det viktig å skifte navn

på tekstfilen eller skifte nåværende katalog i Matlab. Hvis ikke vil den eksisterende tekstfilen bli overskrevet.

```
1   file_1 = fopen('simres.txt','wt')
2   fprintf(file_1,'Places\n')
3   fclose(file_1)
4   dlmwrite('simres.txt', sim.Place_Names, '-append', 'delimiter',
5   '')
6   file_1 = fopen('simres.txt','a')
7   fprintf(file_1,'Transitions\n')
8   fclose(file_1)
9   dlmwrite('simres.txt', sim.Transition_Names, '-append',
10  'delimiter', '')
11  file_1 = fopen('simres.txt','a')
12  fprintf(file_1,'State Diagram\n')
13  fclose(file_1)
14  dlmwrite('simres.txt', sim.State_Diagram, '-append',
15  'delimiter', ' ')
```

I kodeutdraget over skriver dlmwrite Place_Names, Transition_Names og State_Diagram-feltene fra variabelen sim til en tekstfil kalt simres.txt[4]. fprintf legger til tekststrengene ”Places”, ”Transitions” og ”State Diagram”. Kodeutdraget kan lagres i en fil, eksempelvis lagreSimres.m. Når filen blir kjørt opprettes simres.txt.

I GPenSim-kapittelet ble det vist resultatet fra simuleringen av et enkelt eksempel. Petri net-grafen til systemet i eksemplet er repetert i figur 4.1. Etter simulering av systemet i GPenSim, kjøres lagreSimres.m som inneholder kodeutdraget over.

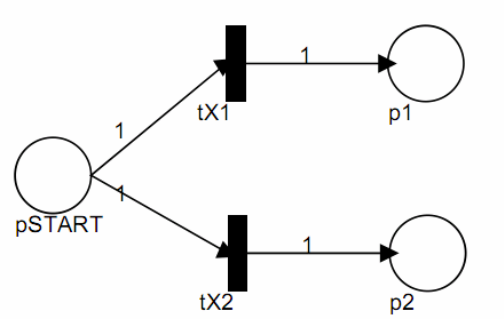


Fig. 4.1: Petri net-graf til et enkelt eksempel.

Ved å kjøre denne koden i Matlab blir en simres.txt fil opprettet med informasjonen som viser under. Systemet har 3 places, pSTART, p1 og p2 og to transitions tX1 og tX2. Innformasjon om simuleringen er lagt under strengen ”State Diagram”.

```
Places
pSTART
  p1
  p2
Transitions
```

```
      tX1
      tX2
 State Diagram
0 0 10 0 0 0 0
 0 0 0 0 0 1 1
 0 0 0 0 0 1 0
10 1 9 1 0 0 0
10 0 0 0 0 1 1
10 0 0 0 0 0 1
25 2 8 1 1 0 0
25 0 0 0 0 1 1
25 0 0 0 0 1 0
35 1 7 2 1 0 0
35 0 0 0 0 1 1
35 0 0 0 0 0 1
50 2 6 2 2 0 0
50 0 0 0 0 1 1
50 0 0 0 0 1 0
60 1 5 3 2 0 0
60 0 0 0 0 1 1
60 0 0 0 0 0 1
75 2 4 3 3 0 0
75 0 0 0 0 1 1
75 0 0 0 0 1 0
85 1 3 4 3 0 0
85 0 0 0 0 1 1
85 0 0 0 0 0 1
100 2 2 4 4 0 0
100 0 0 0 0 1 1
100 0 0 0 0 1 0
110 1 1 5 4 0 0
110 0 0 0 0 1 1
110 0 0 0 0 0 1
125 2 0 5 5 0 0
125 0 0 0 0 0 0
125 0 0 0 0 0 0
```

Lese simuleringsresultat i GPenSimGUI

Tekstfilen som beskriver simuleringsresultatet leses av GPenSimGUI-en. Som ved innlesing av Petri net-definisjonsfiler leses også denne linje for linje. Navn på places og transitions lagres i to forskjellige lister. Simuleringsresultatet lagres i et State Diagram. Dette har et felt for tid og et felt for hvilken transition som ble ferdig. Det har også en liste med antall tokens i hver place og en liste som beskriver hvilke transitions som er enabled og hvilke som skyter.

Simuleringsresultatet over leses inn i GPenSimGUI-en. Figur 4.2, 4.3, 4.4 og 4.5 viser til linje 1, 2, 3 og 4 av simuleringsresultatet over.

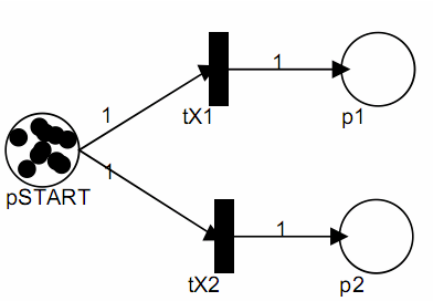


Fig. 4.2: Starttilstanden.

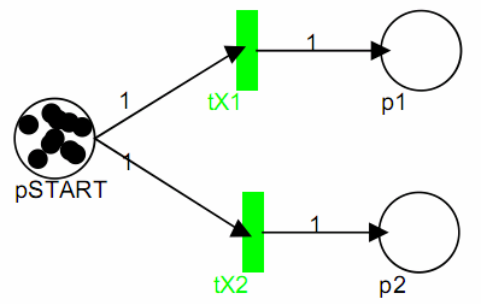


Fig.4.3: To enabled transitions.

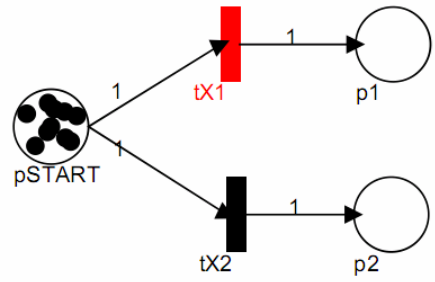


Fig. 4.4: En firing transition.

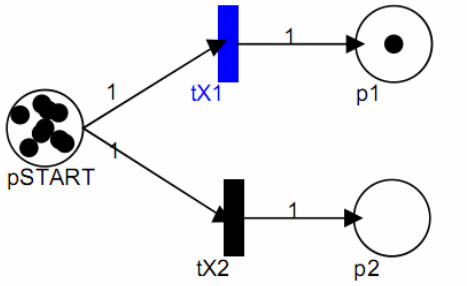


Fig. 4.5: En fired transition.

4.2 Kjerneklasser i GPenSimGUI

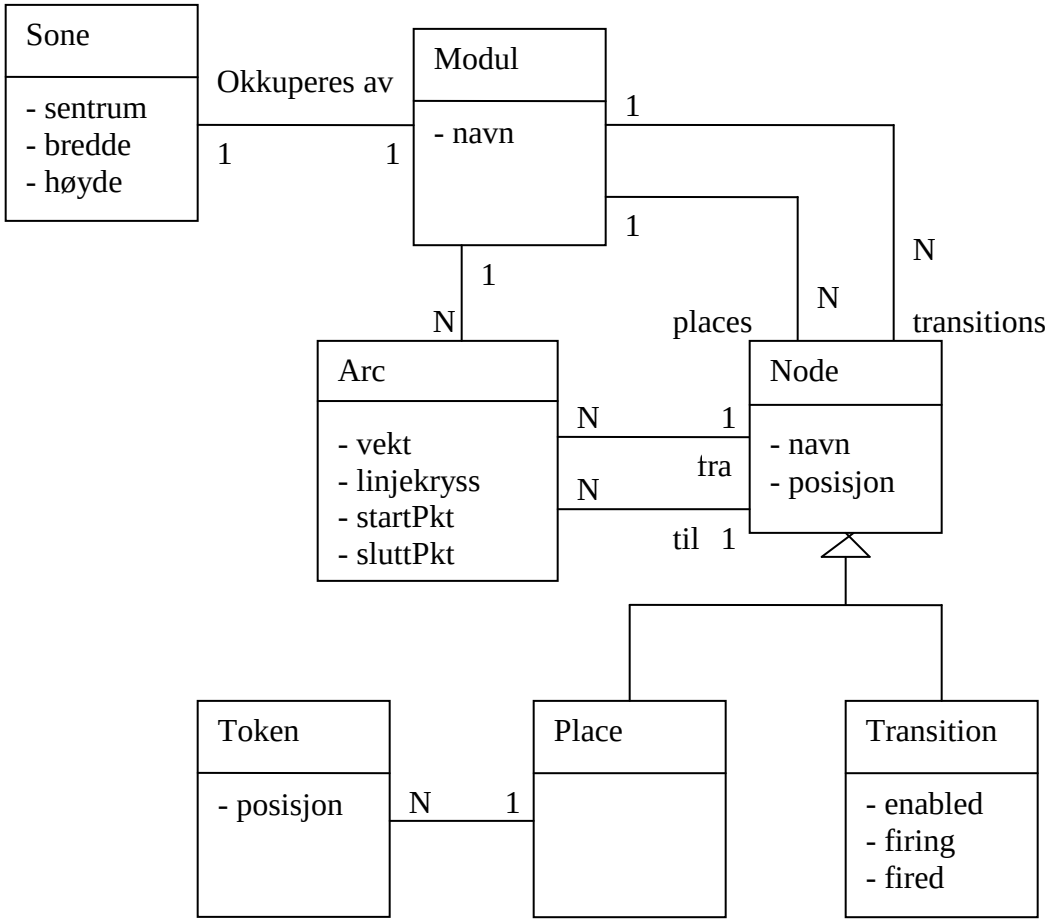


Fig. 4.6: Kjerneklasser i GPenSimGUI-en og koblingene mellom dem.

GPenSimGUI-en er implemenert i Java[5]. I GPenSimGUI-en består en Petri net-graf av en eller flere moduler. Dersom systemet ikke er modellert ved hjelp av moduler, vil hele Petri net-grafen være én modul. En modul har en liste med places, en liste med transitions og en liste med arcs. Hver modul har også et navn. En modul har ikke en posisjon knyttet til seg, men okkuperer i stedet en sone. En sone er et området hvor modulens places og transitions kan plasseres. Sonene har et sentrum, og en bredde og høyde som avgjør hvor stort område av skjermen hver enkelt sone dekker. Dess større sone, dess mer plass for modulens element. Det vil igjen gi større avstander mellom elementene.

En place har et navn og en posisjon. Den har i tillegg en liste med tokens. Listen kan være tom. Transitions har som places et navn og en posisjon på skjermen. Transitions har også egenskaper som sier om den er enabled, om den skyter eller om den er ferdig. Places kan være knyttet til andre transitions via en arc. Hver place og transition kan ha flere arcs knyttet til seg, men en arc knytter alltid bare to element sammen.

En arc har en vekt. Arcs har et startelement og et destinasjonselement. Linjekryss brukes av plasseringsalgoritmen. Etter utplassering av arc-ens element sjekkes antall linjekryss. Dersom arc-en krysser andre element vil arc-ens element få nye posisjoner. Det alternativet som gir færrest linjekryss velges. En arc har også en start og slutt posisjon. Arc-en plasseres i forhold til elementenes posisjon. Startposisjonen er i enden av elementet arc-en går fra. Sluttposisjonen er i enden av elementet arc-en går til.

Tokens blir lagt til og fjernet fra places. Tokens som legges til vil få en posisjon som er innenfor place-en den tilhører. En erfarte at de siste to tillagte tokens ofte ble lagt på nesten samme sted. Når det var få tokens i en place, kunne det være vanskelig å se når en ny ble lagt til. Derfor opprettes en ny token med en liten angitt avstand fra forrige tillagte token.

4.3 Grafikk

For å konstruere en Petri net-graf grafisk, trenger en funksjoner som kan tegne firkanter, sirkler, linjer og trekanter. Places tegnes som sirkler. Transitions tegnes som firkanter. Og arcs tegnes som en kombinasjon av linjer og trekanter. Java2D inneholder funksjoner som kan tegne alle disse geometriske formene. I Java2D kan en tegne elementer på spesifikke posisjoner, forskyve elementene, og forstørre eller forminske dem alt ettersom. Java2D gir også mulighet for å skrive en tegning ut på skriver. Grafikken er derfor laget i Java2D[6,7].

GPenSimGUI-en skal i første rekke være et nyttig program. Et alternativ til Java2D er Java3D. Design var ikke prioritert. Java2D har tilstrekkelig med funksjoner og er også enklere å bruke. Java2D ble derfor valgt framfor Java3D.

4.3.1 Metoder for å tegne geometriske former

Tegne place og tokens

Metoden som tegner en place med tokens vises på neste side.

```
1   tegnPlaceMedTokens(Place p, double flytteX, double flytteY, double skaler){
2      AffineTransform origTransform = getG2().getTransform();
3      getG2().translate(flytteX, flytteY);
4      getG2().scale(skaler, skaler);
5
6      //Place
7      getG2().setColor(Color.BLACK);
8      Ellipse2D e = new Ellipse2D.Double(p.getRadius()*-1,p.getRadius()*-1,
9         p.getRadius()*2,p.getRadius()*2);
10     AffineTransform tr = new AffineTransform();
11     tr.translate(p.getPosisjon().getX(), p.getPosisjon().getY());
12     Shape shape = tr.createTransformedShape(e);
13     getG2().draw(shape);
14
15     //Tokens
16     Iterator itr = p.getTokens().iterator();
17     while(itr.hasNext()){
18        Token tmp = (Token)itr.next();
19        tr = new AffineTransform();
20        e = new Ellipse2D.Double(tmp.getRadius()*-1, tmp.getRadius()*-1,
21           tmp.getRadius()*2,tmp.getRadius()*2);
22        tr.translate(tmp.getPosisjon().getX(), tmp.getPosisjon().getY());
23        shape = tr.createTransformedShape(e);
24        getG2().fill(shape);
25     }
26
27     //Place name
28     getG2().translate(p.getPosisjon().getX()-20, p.getPosisjon().getY()+30);
29     getG2().drawString(p.getName(), 0, 0);
30
31     getG2().setTransform(origTransform);
32     getG2().setColor(Color.WHITE);
33  }
```

En place blir tegnet som en sirkel. Det samme gjør tokens. Dersom brukeren har flyttet skjermbildet, må en ta hensyn til det. Tegningene blir derfor i første omgang flyttet til venstre hjørnet av skjermbildet. Tegningene blir deretter flyttet videre til elementenes posisjon. Skaleringen er med hensyn til hvor mye brukeren har forstørret eller forminsket tegningen som viser på skjermbildet. Places og tokens blir tegnet med svart farge. En place blir tegnet som en hul sirkel. Dersom place-en inneholder tokens blir disse også tegnet på skjermen. Tokens blir tegnet som fylte svarte sirkler. Tokens har posisjon innenfor sirkelen som beskriver place-en. Navnet på place-en blir til slutt påført under tegningen, i svart farge. Figur 4.7 viser resultatet av å tegne en place med 3 tokens.

Fig. 4.7: Tegning av en place med 3 tokens.

Tegne transition

Metoden som tegner en transition vises under.

```
1  tegnTransition(Transition t, double flytteX, double flytteY, double skaler){
2     AffineTransform origTransform = getG2().getTransform();
3     getG2().translate(flytteX, flytteY);
4     getG2().scale(skaler, skaler);
5
6     //Transition
7     if(t.isFiring())
8        getG2().setColor(Color.RED);
9     else if(t.isEnabled())
10       getG2().setColor(Color.GREEN);
11    else if(t.isFired())
12       getG2().setColor(Color.BLUE);
13    else
14       getG2().setColor(Color.BLACK);
15
16    Rectangle2D r = new Rectangle2D.Double((t.getBredde()/2)*-1,
17       (t.getHoyde()/2)*-1,t.getBredde(),t.getHoyde());
18    AffineTransform tr = new AffineTransform();
19    tr.translate(t.getPosisjon().getX(), t.getPosisjon().getY());
20    Shape shape = tr.createTransformedShape(r);
21    getG2().fill(shape);
22
23    //Transition name
24    getG2().translate(t.getPosisjon().getX()-20, t.getPosisjon().getY()+30);
25    getG2().drawString(t.getName(), 0, 0);
26
27    getG2().setTransform(origTransform);
28    getG2().setColor(Color.WHITE);
29 }
```

Places og tokens blir tegnet som svarte sirkler. Når Petri net-grafen opprettes blir også transitions tegnet med svart farge. Under simuleringen skifter transitions status. Dette må tydelig komme frem i GPenSimGUI-en og er symbolisert med forskjellige farger. En enabled transition tegnes med grønn farge. En transition som skyter tegnes med rød farge. Og når en transition er ferdig skifter den til blå farge. Når en skal tegne en transition må en også ta hensyn til hvor på skjermen en befinner seg. En flytter derfor først til flytteX og flytteY. Tegningen blir videre flyttet til transition-ens posisjon. Deretter skaleres tegningen hvis brukeren har zoomet ut eller inn. Transitions blir tegnet som fylte rektangler. Som i metoden som tegner en place med tokens, blir navnet til transition-en skrevet under fra venstre. Figur 4.8 viser tegning av en transition som er kalt Transition-1.

Fig. 4.8: Tegning av en transition.

Tegne arc

Metoden som tegner en arc vises under.

```
1  tegnArc(Arc a, double flytteX, double flytteY, double skaler){
2     AffineTransform origTransform = getG2().getTransform();
3     getG2().translate(flytteX, flytteY);
4     getG2().scale(skaler, skaler);
5
6     //Pilen
7     getG2().setColor(Color.BLACK);
8     l = new Line2D.Double(a.getStartPos().getX(), a.getStartPos().getY(),
9        a.getSluttPos().getX(), a.getSluttPos().getY());
10    tr.translate(a.getSluttPos().getX(), a.getSluttPos().getY());
11    getG2().draw(l);
12
13    //Pilspissen
14    AffineTransform tr = new AffineTransform();
15    int[] xPkt = {-5, 5, -5};
16    int[] yPkt = {-5, 0, 5};
17    Polygon trekant = new Polygon(xPkt, yPkt, 3);
18    double rotasjonsVinkel; //Rotasjon til pilspissen
19    if(a.getFra() instanceof Place){
20       rotasjonsVinkel = finnVinkel(a.getPlace().getPosisjon(),
21          a.getTransition().getPosisjon());
22    else
23       rotasjonsVinkel = finnVinkel(a.getTransition().getPosisjon(),
24          a.getPlace().getPosisjon());
25    tr.rotate(rotasjonsVinkel);
26    Shape shape = tr.createTransformedShape(trekant);
27    getG2().fill(shape);
28
29    //Arc-ens vekt
30    double x;
31    double y;
32    if(a.getFra() instanceof Place){
33       x = (a.getPlace().getPosisjon().getX()-
34          a.getTransition().getPosisjon().getX())/3;
35       y = (a.getPlace().getPosisjon().getY()-
36          a.getTransition().getPosisjon().getY())/3;
37    }
38    else{
39       x = (2*(a.getPlace().getPosisjon().getX()-
40          a.getTransition().getPosisjon().getX()))/3;
41       y = (2*(a.getPlace().getPosisjon().getY()-
42          a.getTransition().getPosisjon().getY()))/3;
43    }
44    getG2().translate(a.getPlace().getPosisjon().getX()-x,
45       a.getPlace().getPosisjon().getY()-y);
46    getG2().drawString(String.valueOf(a.getVekt()), 0, 0);
47
48    getG2().setTransform(origTransform);
49    getG2().setColor(Color.WHITE);
50 }
```

Metoden flytter og skalerer som de forrige metodene. En arc har svart farge. Den består av en linje, og en trekant som representerer pilspissen i den ene enden. Pilspissen tegnes som en polygon av 3 punkt. Ved tegning av en arc er det nødvendig å vite hvilken av place og transition som er på venstre og høyre side. Det er også nødvendig å vite hvilken vei arc-en går. Om den er fra place til transition eller motsatt.

En arc tegnes fra enden av det ene elementet til enden av det andre. Figuren under viser en arc tegnet fra en place til en transition. Som en kan se er x1 og x2 ikke like lange. En trenger derfor å vite om det er place eller transition som er mest til venstre.

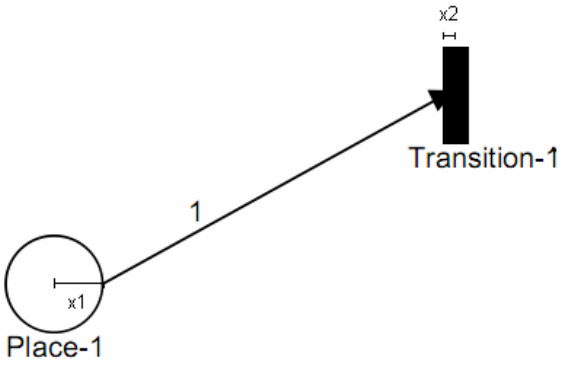


Fig. 4.9: Tegning av en arc, en place og en transition.

Det er nødvendig å vite hvilken vei arc-en går når en skal tegne pilspissen. Pilspissen må være i riktig ende. For at pilspissen skal peke samme retning som pilen går, må en også regne ut vinkelen mellom arc-en og x-aksen. I programmet er det laget en metode som finner denne vinkelen. Figuren under til venstre viser hvordan metoden finner rotasjonsvinkelen til arc-en i figuren over. Først regner den ut vinkelen mellom arc-en og x-aksen som i figuren kalles theta. Siden skjermkoordinatsystemet har positiv y-retning nedover, og Java roterer tegningen som figuren til høyre under viser, må metoden returnere vinkel A. Det er denne vinkelen som brukes for rotasjonen til pilspissen. I dette tilfellet er vinkel A = 2 * pi – theta.

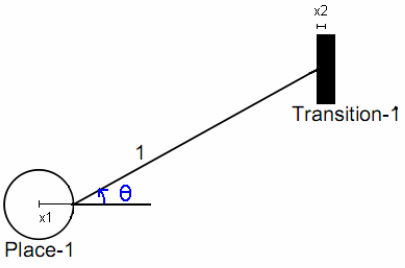


Fig. 4.10: Finner rotasjonsvinkelen.

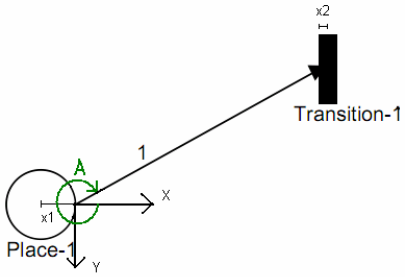


Fig. 4.11: Roterer pilspissen.

Vekten på en arc skrives over arc-en, og nærmest det elementet arc-en kommer fra. Dersom det gikk en arc fra Place-1 til Transition-1, og en arc fra Transition-1 til Place-1, vil vekten tilhøre den arc som har elementet arcen går fra nærmest verdien til vekten.

Tegne sone

Metoden under tegner en sone.

```
1   tegnSone(Sone s, double flytteX, double flytteY, double skaler){
2      AffineTransform origTransform = getG2().getTransform();
3      getG2().translate(flytteX, flytteY);
4      getG2().scale(skaler, skaler);
5
6      //Sone
7      getG2().setColor(Color.BLACK);
8      Rectangle2D r = new Rectangle2D.Double((s.getBredde()/2)*-
9         1,(s.getHoyde()/2)*-1,s.getBredde(),s.getHoyde());
10     AffineTransform tr = new AffineTransform();
11     tr.translate(s.getSentrum().getX(), s.getSentrum().getY());
12     Shape shape = tr.createTransformedShape(r);
13     getG2().draw(shape);
14
15     getG2().setTransform(origTransform);
16     getG2().setColor(Color.WHITE);
17  }
```

En sone tegnes som et rektangel. En kunne tidlig anta at eksperimentering med sonens størrelse ville bli nødvendig da systemer med få elementer trenger liten plass, mens store systemer krever mer plass. Det kan også være ønskelig å variere sonens form for å få en bedre fremvisning av grafen. En sone har derfor egenskapene bredde og høyde. En sone tegnes om dens sentrum. Figur 4.12 viser tegning av en sone med noen element.

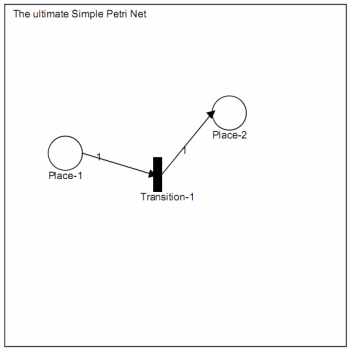


Fig. 4.12: Tegning av en sone med to places, en transition og to arcs.

4.4 Sonekart for å plassere moduler

Skjermbildet blir delt opp i flere områder. En modul plasserer elementene sine i et av områdene. Et slikt område kalles en sone. Det er alltid like mange soner vertikalt og horisontalt. Hver sone har en størrelse og en form. Begge egenskapene kan endres av brukeren. Det vil ikke være mulig å plassere store moduler i små soner. Det vil ikke være nok plass. Små soner med mellomstore moduler vil gi et høyt antall arcs som krysser andre element. Så hvorfor ikke alltid bruke store soner? Dersom en har moduler med få element, eksempelvis. to moduler med 2 places og 1 trasition i hver, vil avstanden

mellom modulenes element bli veldig stor. Brukeren har derfor muligheten til å endre sonestørrelsen.

Hvor mange soner som genereres er avhengig av hvor mange moduler som skal plasseres. Å opprette mange soner som ikke blir brukt vil bare gi soner som er unødvendige å tegne. Det opprettes noen flere soner enn det er moduler. Det er for å gi brukeren større frihet til å velge hvordan grafen skal se ut. Figuren under viser et eksempel. En klient sender ut en forespørsel. Etter at forespørselen er behandlet, i opp til flere moduler, får klienten et svar tilbake. Sonekart A har nok soner til å beskrive Petri net-grafen, men sonekart B gir brukeren flere plasseringsmuligheter. I dette eksemplet ville det kanskje vært mer naturlig å plassere modulene som i sonekart B. Dersom flere moduler er koblet til mer enn én annen modul, har sonekart B flere plasseringsmuligheter, som igjen kan føre til færre linjekryss.

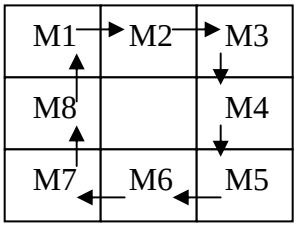


Fig. 4.13: Sonekart A.

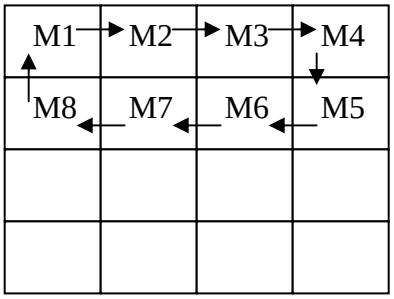


Fig.4.14: Sonekart B.

En sone har en bredde, en høyde og flere punkt. En modul okkuperer en sone, og kan plassere places og transitions i disse punktene.

En sone var i utgangspunktet laget som en sirkel, og hadde et gitt antall punkt tilfeldig plassert innenfor sirkelen. En bedre løsning ble å lage en sone som et rektangel som vist i figurene over. I stedet for å plassere punktene tilfeldig rundt innenfor rektangelet blir de systematisk plassert som figur 4.15 viser. Avstanden mellom hvert punkt er stor nok til at to element som plasseres i to punkt ved siden av hverandre, ikke vil skjære hverandre.

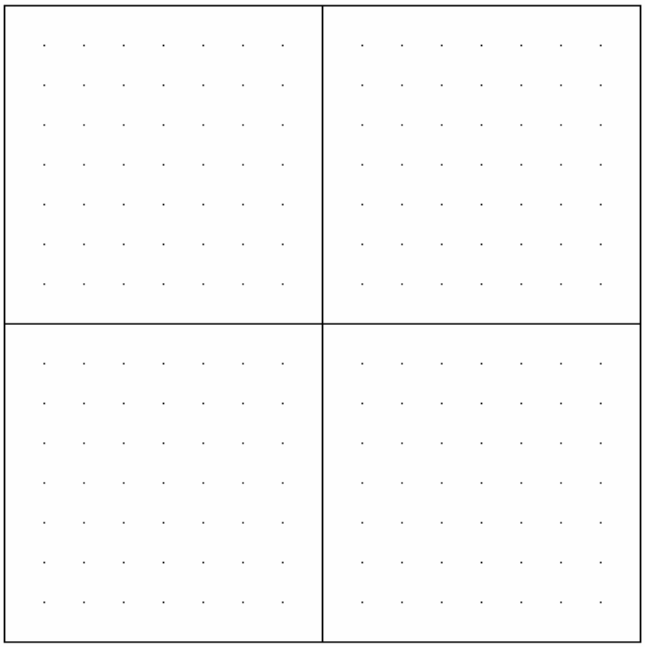


Fig. 4.15: Et 2x2 sonekart.

Figur 4.16 viser en okkupert sone. Modulen som okkuperer sonen har places og transitions plassert i sonens punkt.

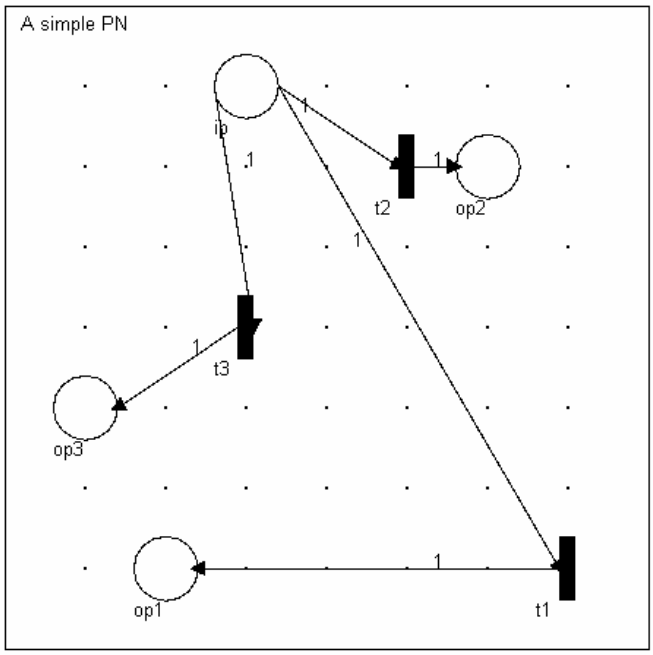


Fig. 4.16: Viser en modul plassert i en sone.

4.5 Plassere element på skjermen

Når en skal modellere små systemer, slik som systemet der to transitions bytter på å skyte, definerer en Petri net-grafen i én Petri net-definisjonsfil. Dess større systemer som skal modelleres, dess større område på skjermen kreves, for å kunne plassere alle places og transitions. For å plassere alle places og transitions til større systemer, kan en øke sonens størrelse. Et stort system plasseres i en stor sone. Det kan høres enkelt ut.

Det er allikevel noen problemer med denne fremgangsmåten. Når systemene blir store og komplekse, vil fremgangsmåten gi uoversiktlige Petri net-grafer. Places som en vil at skal tegnes et stykke fra hverandre, kan ende opp ved siden av hverandre. En har ikke oversikt over hvilke places og transitions som hører til hvilke deler av systemet. Utplasseringsalgoritmen vil få problemer med å holde antall linjekryss lavt. Ved å øke antall alternative utplasseringer øker tiden algoritmen bruker, og dette vil ikke løse det første problemet.

En kunne enkelt eliminert flere linjekryss ved å flytte places og transitions manuelt etter GPenSimGUI-en hadde plassert dem. Det som er mer oppsiktsvekkende er at systemet blir veldig uoversiktlig. En place som tilhører klientforespørsler og er definert på klientsiden, kan eksempelvis bli plassert blant places som tilhører meldingsbuffere på tjenersiden. Ved å dele systemer opp i moduler kan tegningen av grafen bli langt mer oversiktlig. Områder er avmerket med navn på modulene. Eksempelvis klient, Internett, tjener osv.

Når systemet er modellert med moduler, kan en la plasseringsalgoritmen fokusere på å holde antall linjekryss lavt innenfor hver modul. Dersom komplekse systemer modelleres, kan en velge å vise én og én modul, og ignorere de fleste linjekryss som måtte være på oversiktsbildet av grafen.

Når alle element er plassert på skjermen er det ingen restriksjoner til hvor elementene kan flyttes. Brukeren trenger ikke være redd for å flytte et element fra en sone litt inn i en sone som okkuperes av en annen modul. Det er mer brukt som en retningslinje. Å flytte et element fra en sone, godt inn i en sone som beskriver en annen modul, vil ikke gi mening og bare gjøre grafen mer rotete.

Under kan en se to figurer. Begge figurene viser Petri net-grafen til samme system. I figur 4.17 er systemet modellert i en Petri net-definisjonsfil. I figur 4.18 er systemet modellert med moduler. Når en viser et system som er modellert med moduler kan en se på en og en modul, og dermed ignorere linjekryss på tvers av moduler. Disse figurene har som hensikt å vise at grafen til systemet som er modellert med moduler er mer oversiktlig enn grafen til systemet som er modellert uten. Det skal eksempelvis gå ut fra grafen om en place er på klientsiden eller den er en del av den taktiske modulen. Figurene fokuserer ikke på linjekryss. Noen places og transitions er derfor flyttet slik at fokuset ikke skyves over på linjekryss i dette tilfellet.

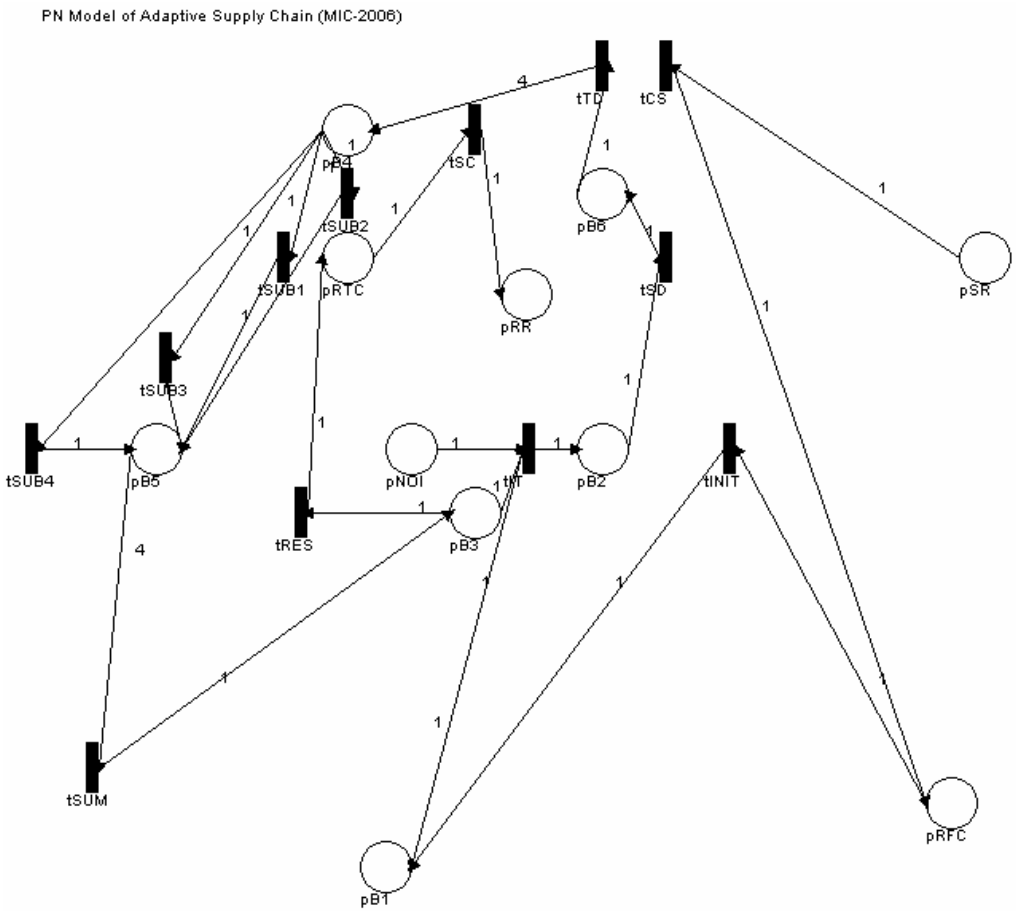


Fig.4.17: Petri net-grafen til et system definert i én Petri net-definisjonsfil.

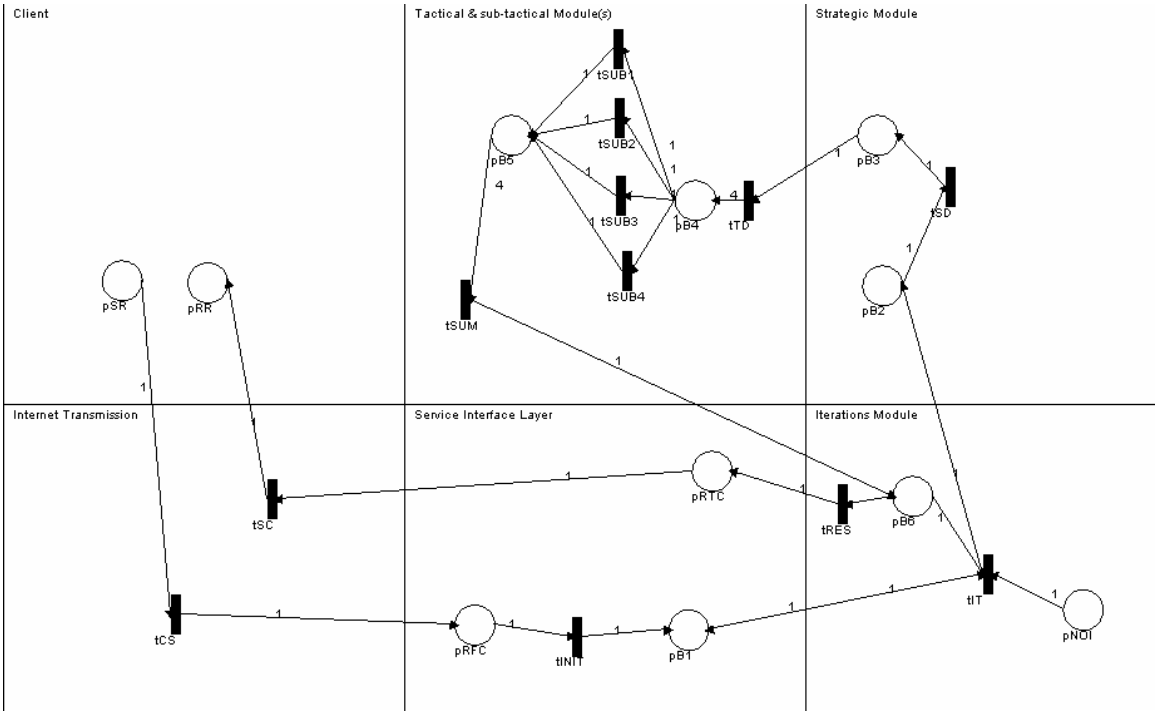


Fig.4.18 Petri net-grafen til et system definert i flere Petri net-definisjonsfiler.

I neste delkapittel ser en på plassering av places og transitions. Plasseringen kan være av en Petri net-definisjonsfil, eller flere (dersom modellen er delt opp i moduler). Delkapittelet som følger beskriver plassering av eventuelle moduler. Begge kapittelene fokuserer på å gjøre grafen mest mulig oversiktlig presentert på skjerm.

4.5.1 Plassere places og transitions

En av oppgavens utfordringer var å plassere places og transitions slik at færrest mulig arcs krysset hverandre. Dess større systemene ble, dess vanskeligere var det å holde antall linjekryss lavt.

Plasseringsalgoritmen kjøres for alle modulene. Algoritmen går igjennom arc for arc. For hver arc sjekkes place og transition. Dersom place eller transition ikke er plassert blir de tilgitt en posisjon. Posisjonen er gitt av et tilfeldig valgt punkt fra modulens sone. Anta at ingen places eller transitions er plassert. Når den første arc-en velges, sjekkes place og transition. Ingen av elementene er plassert. De får da valgt en tilfeldig posisjon. Siden ingen andre arcs er plassert vil den nye arc-en ikke krysse andre element. Alt er i orden og algoritmen fortsetter med neste arc. Den første arc-ens place og transition er nå plassert og posisjonene deres fjernes fra sonen. De er ikke lenger ledig for andre element.

Ved neste iterasjon vil en place og en transition være plassert. Disse har en arc mellom seg. Den nye arc-ens place eller transition skal helst ikke plasseres slik at det skjer et linjekryss. Verken at den nye arc-en krysser den første arc-en, eller at den krysser noen av de plasserte elementene.

Når en sjekker etter linjekryss må en ta hensyn til flere ting. To arcs som har en felles place eller transitions må kunne krysse hverandre. Arcs med felles element vil krysse hverandre i endepunktet. Dette forklares enklere med en figur.

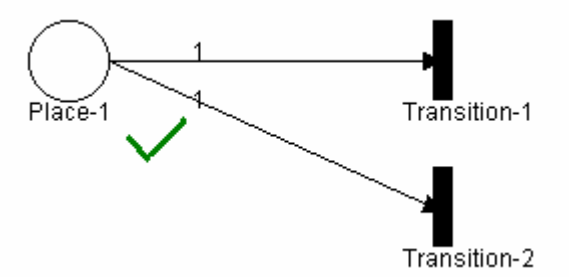


Fig. 4.19: Viser et lovlig linjekryss.

Det skal forekomme minst mulig linjekryss mellom to arcs som ikke har et felles element. Dette går ut fra figur 4.20.

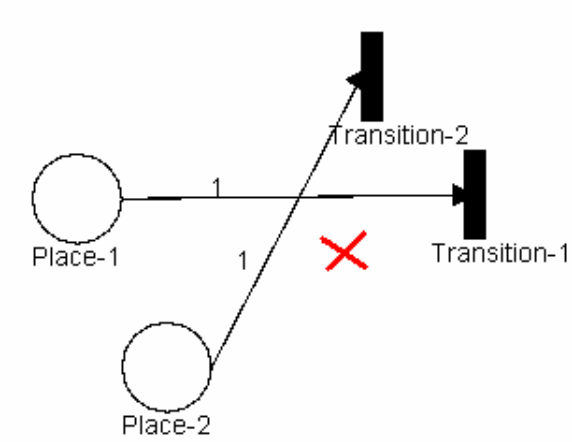


Fig. 4.20: Viser et ulovlig linjekryss.

En arc bør heller ikke krysse en allerede plassert place eller transition. Dette gjelder ikke for arc-ens egne element. I fig. 4.21 er den nye arc-ens place allerede plassert. Arc-en kan krysse denne place-en. Arc-en kan ikke krysse den plasserte transition-en som tilhører en annen arc.

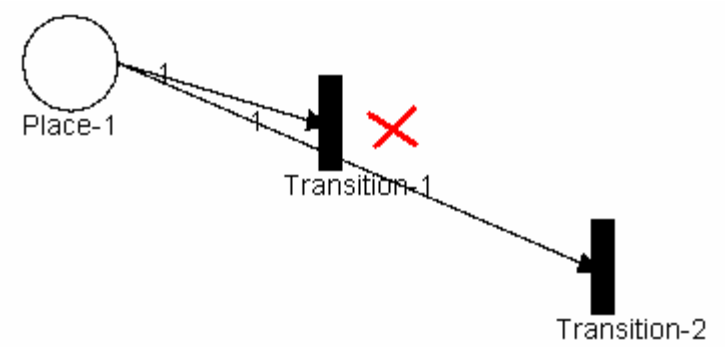


Fig. 4.21: Viser et lovlig og to ulovlige linjekryss.

Arc-ens place eller transition skal helst ikke plasseres slik at de blir krysset av plasserte arcs.

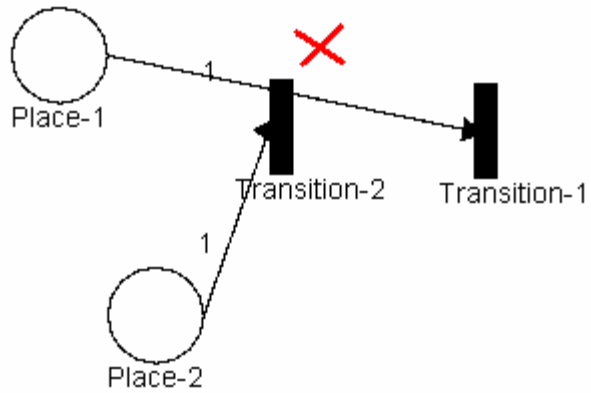


Fig. 4.22: Viser to ulovlige linjekryss.

Etter algoritmen har gått igjennom alle arcs, må den sjekke om modulen inneholder places eller transitions som ikke er koblet med en arc i samme modul. Dersom modulen inneholder en slik place eller transition må de også plasseres. De kan plasseres på et ubrukt punkt, men skal helst ikke la en tidligere arc krysse igjennom seg.

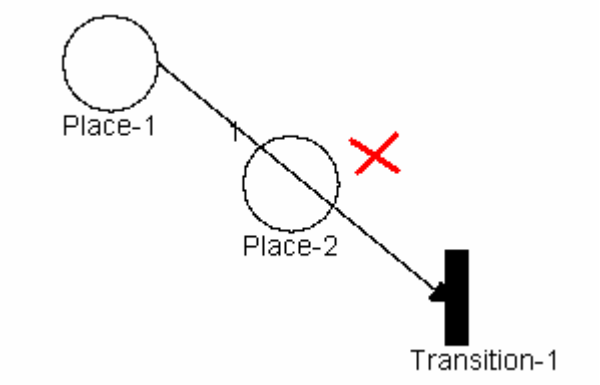


Fig. 4.23: Viser to ulovlige linjekryss.

Algoritme for å sjekke om to linjer krysser hverandre

Å sjekke om to linjesegment krysser hverandre er en viktig del av plasseringsalgoritmen. Det finnes flere algoritmer for å gjøre dette. Dette avsnittet er i stor grad hentet fra [8]. En enkel måte å finne ut om to linjesegment krysser hverandre på er som følger. Først må en finne likninger for de to linjesegmentene. Likningen for en rett linje er gitt på formen Ax + By = C. A og B finnes ut fra endepunktene til en linje.

$A = y_2 - y_1$
$B = x_1 - x_2$
$C = A*x_1 + B*y_1$

En har da to likninger:

$A_1x + B_1y = C_1$
$A_2x + B_2y = C_2$

En har to likninger med to ukjente. Linjene krysser i (x, y). x og y kan finnes på flere måter. En fremgangsmåte for å finne y er som følger:

Likning 1 gir:
$x = (C_1 - B_1y) / A_1$

Dette settes inn i likning 2 for å finne y:
$A_2 * ((C_1 - B_1y) / A_1) + B_2y = C_2$
- ⇨ $(A_2C_1 / A_1) - (A_2B_1y / A_1) + B_2y = C_2$
- ⇨ $y * (B_2 - (A_2B_1 / A_1) = C_2 - A_2C_1 / A_1$

- ⇨ $y = (C_2 – (A_2C_1 / A_1)) / (B_2 – (A_2B_1 / A_1)$
- ⇨ $y = (C_2 – (A_2C_1 / A_1)) / (A_1B_2 – A_2B_1 / A_1)$
- ⇨ $y = (A_1C_2 – A_2C_1) / (A_1B_2 – A_2B_1)$

x kan finnes på samme måte. x blir da:

$x = (B_2C_1 – B_1C_2) / (A_1B_2 – A_2B_1)$

Linjene krysser hverandre i (x,y). Siden en trenger å vite om to linjesegment krysser hverandre, og ikke to linjer, må en sjekke om (x,y) er innenfor begge linjesegmentene. Om så er tilfellet, vil linjesegmentene krysse hverandre.

Algoritme for linjekryss [8]:

```
1  double det = A1*B2 – A2*B1
2  if(det == 0) {
3         // Parallelle linjer
4  else{
5         double x = (B2*C1 – B1*C2)/det
6         double y = (A2*C2 – A2*C1)/det
7  }
```

Algoritme for å plassere places og transitions

I Java[5] finnes allerede definerte metoder for å sjekke om en linje krysser en annen linje eller et rektangel. Metodene kalles intersectsLine og intersects. For å sjekke om en linje krysser en place (som er representert av en sirkel), lages et rektangel over place-en med bredde og høyde lik radius ganger 2. En sjekker så om linjen krysser rektangelet.

Algoritmen vil alltid klare å plassere ut en ny arc uten at det forekommer linjekryss, dersom et av elementene enda ikke er plassert, og det er tilstrekkelig med plass. Når en arc skal opprettes der begge elementene allerede er plassert blir det fort linjer som krysser hverandre. For å minimere antall linjekryss lages en liste med alternative modulplasseringer. Dersom algoritmen ikke klarer å plassere ut en moduls element, uten at det forekommer linjekryss, opprettes alternativer. Algoritmen kjøres igjennom flere ganger. Alle alternative modulplasseringer har et felt som beskriver antall linjekryss ved denne modulplasseringen. Når alle alternative plasseringer er opprettet, sorteres de etter antall linjekryss. Den modulplasseringen som gav færrest linjekryss velges. Places og transitions fra modulen får posisjonene som var valgt i denne plasseringen.

En arc har også et felt som forteller hvor mange element arc-en krysser. Det kan være andre allerede plasserte arcs, plasserte places og plasserte transitions. Når elementene til en arc skal plasseres blir arc-ens place og transition tildelt en ledig posisjon innenfor sonen, så lenge elementet ikke allerede er plassert. Det opprettes deretter en linje mellom place og transition. Linjen er fra den ene enden på place-en til den ene enden av transition-en, eller motsatt.

Det er også laget en liste for alternative plasseringer for hver arc. Place og transition til arc-en plasseres ut. Deretter sjekker en hvor mange linjer arc-en krysser og lagrer antallet.

Dersom det er linjekryss, prøver en på ny. Hvis algoritmen ikke klarer å opprette den bestemte arc uten at det forekommer linjekryss, velges til slutt den arc som gav færrest linjekryss. Deler av plasseringsalgoritmen viser under.

```
For alle arcs
        Hvis place ikke er plassert
                Gi place tilfeldig posisjon i sone
        Hvis transition ikke er plassert
                Gi transition tilfeldig posisjon i sone
        Gi arc start og slutt posisjon fra enden av place til enden av transition eller motsatt

        Opprett linje L1 fra arc.startPosisjon til arc.sluttPosisjon
        For alle plasserte arcs
                Opprett linje L2 fra plassertArc.startPosisjon til plassertArc.sluttposisjon
                Hvis L1 krysser L2
                        Linjekryss
        For alle plasserte places
                Opprett rektangel R1 rundt plassertPlace
                Hvis L1 krysser R1
                        Linjekryss
        For alle plasserte transitions
                Opprett rektangel R2 over plassertTransition
                Hvis L1 krysser R2
                        Linjekryss
For alle uplasserte places
        Gi place tilfeldig posisjon i sone
        Opprett rektangel R1 rundt uplassertPlace
        For alle plasserte arcs
                Opprett linje L1 fra arc.startPosisjon til arc.sluttPosisjon
                Hvis L1 krysser R1
                        Linjekryss
For alle uplasserte transitions
        Gi transition tilfeldig posisjon i sone
        Opprett rektangel R2 rundt uplassertTransition
        For alle plasserte arcs
                Opprett linje L1 fra arc.startPosisjon til arc.sluttPosisjon
                Hvis L1 krysser R2
                        Linjekryss
```

Flytte place eller transition

Med et komplekst system vil ikke plasseringsalgoritmen klare å plassere alle elementene slik at linjekryss ikke forekommer. Det er allikevel forsatt mulig å fjerne flere linjekryss. Brukeren kan selv flytte places og transitions. Når en place eller en transition flyttes, følger arcs som er koblet til elementet etter. En kan flytte element for å unngå linjekryss,

men en kan også ta i bruk flyttefunksjonen for å få en mer ønskelig struktur på Petri net-grafen. Eksempelvis kan brukeren ville at grafen skal ha en flyt fra venstre mot høyre og tilbake igjen. Hvert element kan da enkelt flyttes med musepekeren. Dette er lettvint og går raskt.

For å flytte et element holdes musepekeren over det elementet som skal flyttes. Deretter trykker en og holder nede første musetast, og drar elementet til en ny posisjon. Når elementet er ved en ønskelig plassering slippes musetasten. Posisjonen oppdateres etter hvert som elementet flyttes. Det er da enklere å se når en har flyttet elementet tilstrekkelig for å fjerne et linjekryss, enn om posisjonen ikke oppdateres før brukeren slipper musetasten. Figuren under viser en del av en Petri net-graf. I figuren er et linjekryss som kunne vært unngått. Et element flyttes og grafen blir mer oversiktlig. Dette linjekrysset har oppstått ved å flytte p2. Det ville ikke oppstått ved utplassering av denne grafen.

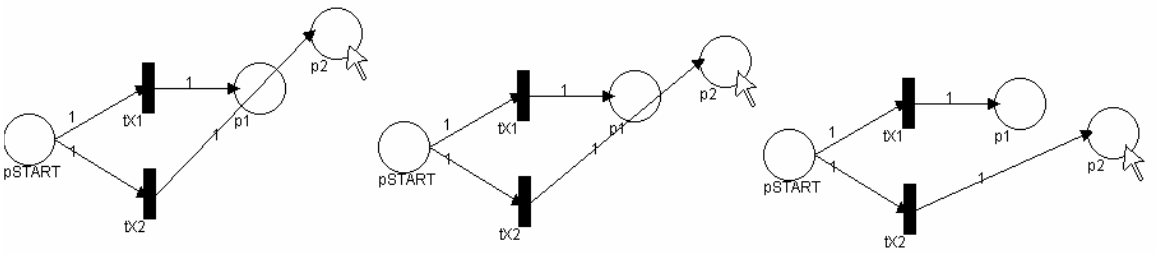


Fig.4.24: Flytting av en place.

4.5.2 Plassere modulene i soner

Modulene blir plassert i soner. Hver modul plasseres i en egen sone. De modulene som har flest koblinger seg i mellom plasseres først. Dersom brukeren ikke er fornøyd med det endelige oppsettet, kan det endres ved å flytte moduler over i nye soner.

GPenSimGUI-en går igjennom intermodular for å finne hvilken rekkefølge modulene skal plasseres etter. En etter en arc leses fra intermodular. Ved å sjekke arc-ens place og transition, finner en hvilke to moduler arc-en kobler sammen. Det opprettes en liste over koblingene. Listen er sortert på antall arcs. De to modulene som har flest arcs seg i mellom ligger først i listen.

Programmet starter i begynnelsen av listen. Den første sammenkoblingen tas ut. En av modulene fra koblingen hentes og plasseres i en sone. Deretter blir den andre modulen plassert i nærmeste ledige sone. Listen itereres og en og en kobling hentes ut. Dersom ingen av modulene er plassert, plasseres en av de, og den andre i nærmeste ledige sone, som for første modulpar. Dersom en av modulene er plassert, plasseres den andre nærmest mulig modulen som allerede er plassert.

Siden utplasseringen av moduler ikke tar hensyn til linjekryss, er den overnevnte teknikken brukt. Det kan være med på å gjøre at arcs slipper å krysse i gjennom flere

soner enn nødvendig. I figur 4.25 vises 3 moduler. Modulene til venstre har flest arcs seg i mellom og er derfor plassert nærmest hverandre. I figur 4.26 har to av modulene byttet plass. Det er gjort ved å manuelt flytte modulene. Nå må arcs unødvendig krysse i gjennom flere soner.

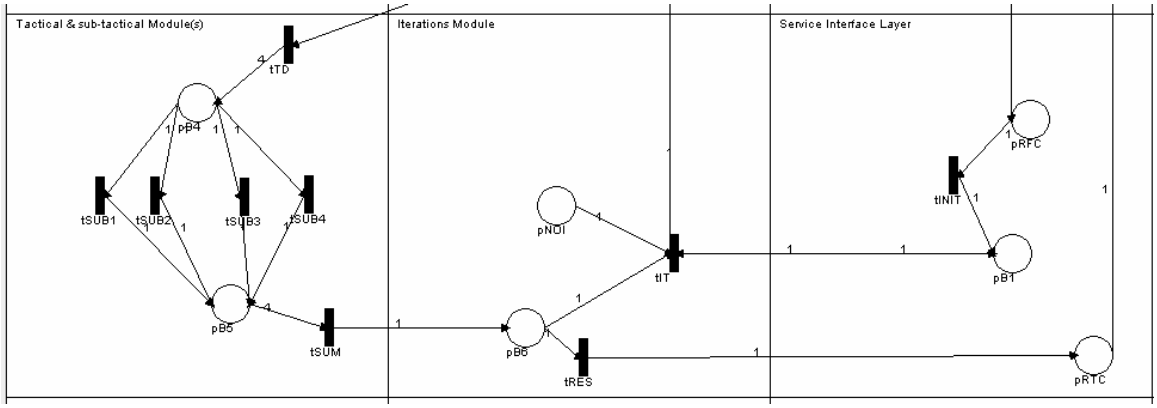


Fig. 4.25: Viser 3 moduler plassert ved siden av hverandre.

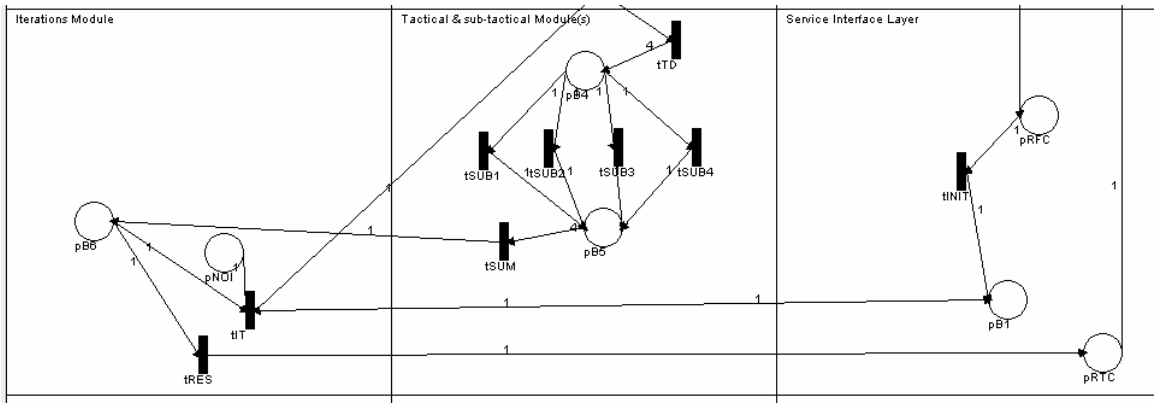


Fig. 4.26: Viser 3 moduler ved siden av hverandre etter flytting.

Dersom Petri net-grafen blir mer oversiktlig ved at to moduler bytter plass er det mulig å gjøre det. Hvordan en flytter en modul over til en annen sone er forklart i neste delkapittel.

Flytte modul

Moduler plasseres på sonekartet. De modulene som har flest koblinger seg i mellom plasseres først. Dersom brukeren ikke liker oppsettet kan han/hun endre på det ved å flytte på modulene. Ved å flytte en modul kan Petri net-grafen bli mer oversiktlig.

For å skille flyttefunksjonene fra hverandre er det opprettet en avkryssingsboks. Denne kalles flytte modul. For å flytte en modul må boksen være avkrysset. Det vil da ikke lenger være mulig å flytte places eller transitions før krysset fra boksen er fjernet. En modul flyttes ved å klikke første musetast innenfor sonen den tilhører, dra musepekeren over til en ny sone og slippe tasten. En modul kan flyttes til en ledig sone eller til en sone

som allerede er okkupert av en annen modul. Dersom modulen flyttes til en okkupert sone, bytter modulene plass.

Når en modul skifter sone, flyttes alle elementene som er innenfor sonen. Elementene blir plassert i den nye sonen, som i forrige sone. Det er enklere å flytte en modul når sonekartet er avmerket på skjermen. Figurene over viste flytting av to moduler.

4.6 Grafisk presentasjon av simulering i GPenSimGUI

I høyre hjørne i GPenSimGUI-en er det to knapper. Knappene er merket med ikoner. Den ene knappen har en tegning av en pil som peker bakover, mens den andre har en tegning av en pil som peker fremover. Når brukeren trykker på en av knappene leses en linje fra simuleringsresultatet. Forrige linje leses dersom brukeren trykker på pil bakover, og neste linje dersom brukeren trykker på pil fremover.

Når simuleringsresultatet er lest inn, hentes første tilstand. Knappene aktiveres og brukeren kan studere simuleringsresultatet ved å klikke frem og tilbake.

I tillegg til den grafiske presentasjonen av simuleringsresultatet, gir GPenSimGUI-en litt informasjon om tilstandene i tekst. Til høyre for knappene som styrer visningen av simuleringsresultatet, finner en tiden til den nåværende tilstand. Når antall tokens vokser blir det vanskeligere å se nøyaktig hvor mange tokens hver place inneholder. Denne informasjonen er derfor også gitt i tekst. Siden en liste av alle places med tilhørende tokens kan ta stor plass, er den opprettet i et eget vindu. Det er for å gi størst mulig plass til Petri net-grafen i hovedvinduet.

I figur 4.27 er knappene som brukes for å hente forrige og neste linje fra simuleringsresultatet vist. Denne figuren fremhever også hvor tiden til en tilstand kan avleses. Figur 4.28 er et utklipp fra vinduet som gir tilleggsinformasjon om tilstanden. Antall tokens avleses bak navnet til en place.

Dersom systemet er delt opp i moduler er alle modulene som inneholder places listet opp. Modulens navn er uthevet med store bokstaver. Places fra en modul er listet opp bak modulen de hører til. Med denne ordningen er det lettere å finne tilbake til en place i Petri net-grafen.

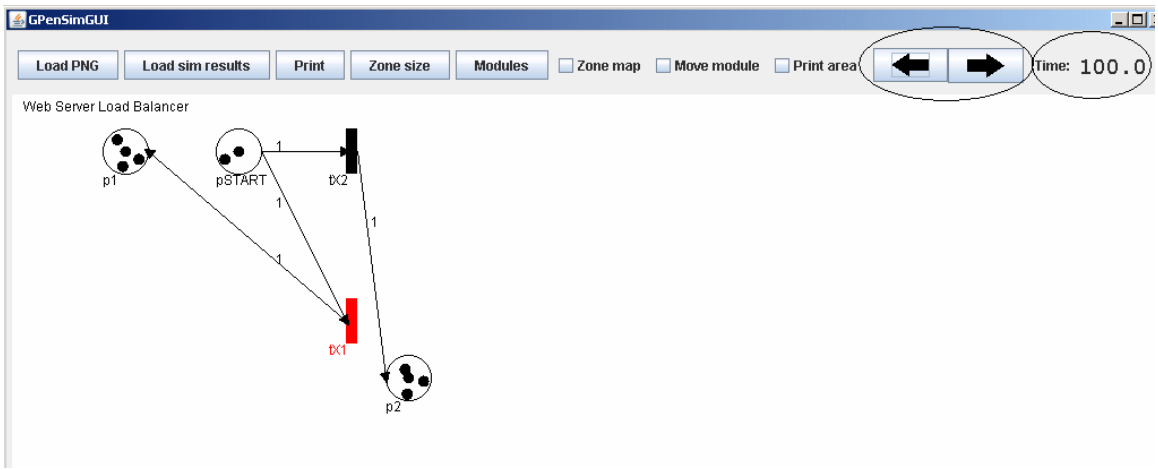


Fig. 4.27: Utklipp fra GPenSimGUI-en. Figuren viser knapper for å hente forrige og neste linje fra simuleringsresultatet, og tiden for tilstanden.

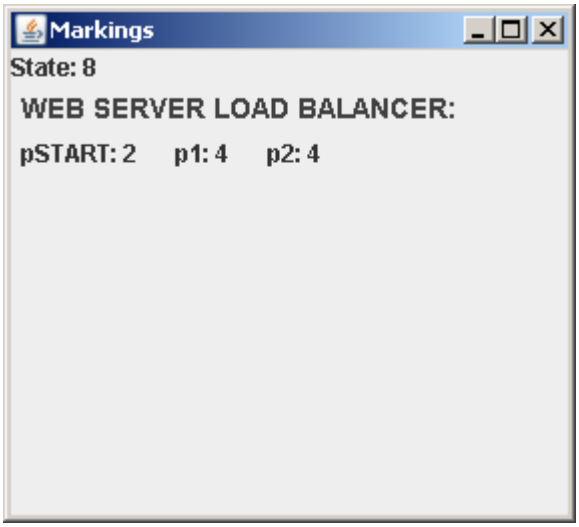


Fig. 4.28: Utklipp av vinduet som beskriver informasjon om en tilstand.

Hver tredje klikk på nesteknappen fører til visning av en ny tilstand. Første gang en klikker nesteknappen går det ut fra grafen hvilke transitions som er enabled. Andre gang vises transitions som skyter med rød farge. Og tredje gang vises hvilken transition som er ferdig med blå farge. Når en trykker en tredje gang oppdateres også antall tokens i places. Om en forsetter å klikke på nesteknappen vil samme prosess gjentas for neste tilstand.

Dersom frem og tilbake knappene skulle føre til at en ny tilstand ble lest, ville den grafiske presentasjonen blitt dårligere. Tilstander ville hatt informasjon om enabled transitions, transitions som skyter og transitions som er ferdig. En transition kunne da vært representert av 2 av alternativene. En må enten unnlate å vise dette eller merke transitions som skyter med en kombinasjon av rød og grønn farge. På samme måte ville en ved å skifte tilstand i figur 4.27 få transitions som er ferdig og enabled. De kunne eksempelvis blitt markert med blå og grønn farge. Da er et bedre alternativ å vise hvilke transition som ble ferdig først, så hvilke som nå er enabled, og til slutt de som skyter. Denne fremgangsmåten er valgt i rapporten.

4.7 Utskrift av Petri net-graf til skriver

Petri net-grafen skisseres gjerne på papir. Hvis en senere ønsker å vise til grafen, må en finne frem kladden. En kan eventuelt tegne grafen på ny i et program som tillater det. Dvs. et program som kan tegne places, transitions og arcs. Med en liten tilleggsfunksjon i GPenSimGUI-en slipper en å tegne grafen på ny. Petri net-grafen blir tegnet på skjermen og kan også enkelt skrives ut til skriver eller pdf-fil. En enkel Petri net-graf er skrevet ut til pdf-fil. Utklipp av utskriften viser i figur 4.30.

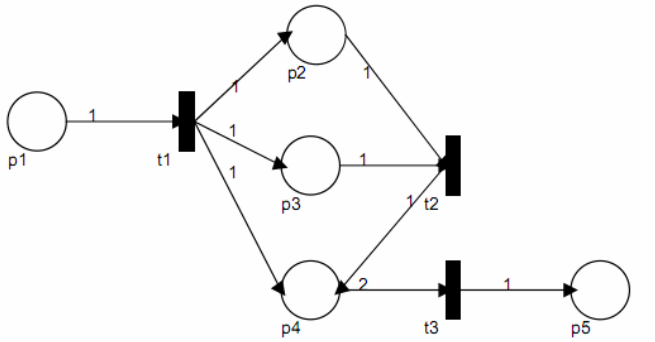


Fig. 4.30: Utskrift av en enkel Petri net-graf.

4.7.1 Feil skalering på utskriften

For å flytte tegneelementene til riktig sted og gi de riktig størrelse, ble transformasjonen under brukt.

```
1  AffineTransform at = new AffineTransform();
2  at.translate(flytteX, flytteY);
3  at.scale(skaler, skaler);
4  getG2().setTransform(at);
```

Dette gav en riktig transformasjon av tegneelementene på skjermen. Men det dukket opp et problem først når en skulle skrive tegningen ut på skriver. Det viste seg at utskriften fra programmet gav en tegning som var nedskalert flere hakk. Resultatet er vist i figuren under. På skjermen vises tegningen som i figur 4.30, men på utskriften vises tegningen som i figur 4.31.

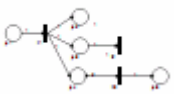


Fig. 4.31: Nedskalert tegning.

En løste problemet ved å ta vare på transformasjonen, flytte, og forstørre eller forminske tegneelementene, for så å resette transformasjonen.

```
1  AffineTransform origTransform = getG2().getTransform();
2  getG2().translate(flytteX, flytteY);
3  getG2().scale(skaler, skaler);
4  ..
5  ..
6  getG2().setTransform(origTransform);
```

Nå er tegningen som blir skrevet ut, nesten helt lik den som viser på skjermen. Den lille forskjellen som forsatt er tilstedet skyldes i følge [9] dobbel buffering. En kan også skrive grafen ut under simuleringen.

Kapittel 5

Eksempel

I dette kapittelet uforskes GPenSimGUI-en med et eksempel. Eksemplet er hentet fra GPenSim-manualen[3]. Systemet i eksemplet er modellert med moduler. Det gjør at en får se alle sidene av GPenSimGUI-en. Systemet er allikevel ikke så komplekst at det trengs flere sider med figurer for å vise Petri net-grafen. Systemet er modellert i 6 forskjellige moduler, og grafen kan dermed bli presentert strukturert og oversiktlig.

Systemer kan bli så komplekse at selv flytting av element ikke vil medføre at grafen får et oversiktlig utseende. Grafen til slike system kan bli problematisk å tegne oversiktlig også på papir og i en Petri net-editor. I GPenSimGUI-en er det opprettet en metode for å vise deler av grafen av gangen. Selv om det ikke er nødvendig for systemet som presenteres i dette kapittelet, så blir fremgangsmåten får å gjøre det vist og forklart.

Eksemplet brukes som en bruksanvisning til GPenSimGUI-en. Leseren får se deler av definisjonsfilene som brukes for å bygge opp Petri net-grafen i GPenSim. Videre blir en presentert for deler av simuleringsresultatet fra simuleringen av systemet med GPenSim. En Matlab-fil brukes for å eksportere resultatet fra Matlab til en tekstfil. En figur viser hvordan definisjonsfilene leses inn i GPenSimGUI-en. En får se hvordan moduler kan flyttes for å skaffe en mer oversiktlig graf. En kan også flytte places og transitions for å gjøre grafen mer oversiktlig. En får se hvordan en enkelt kan skrive grafen eller deler av grafen til pdf-fil.

Tekstfilen som inneholder simuleringsresultatet blir lest inn i GPenSimGUI-en. Figurer viser deler av simuleringen grafisk.

Ved å bruke piltastene kan en flytte skjermbildet. Dersom grafen er større enn skjermbildet kan en forskyve skjermbildet og dermed se resten av grafen. En kan bruke zoomefunksjonen for å forstørre tegningen, komme nærmere på grafen og se flere detaljer. På denne måten kan en fokusere på deler av grafen. En kan også zoome ut og få et oversiktsbilde av grafen.

Siden modulene i systemet inneholder få places og transitions vil en først endre størrelsen på sonene. Grafen blir da mer kompakt. Det vil gi kortere avstand mellom modulene, og mer av grafen blir vist på skjermbildet av gangen.

Systemet i eksemplet viser positive sider med å gi en grafisk presentasjon av graf og simuleringsresultat. Det vil spesielt være vanskelig å få oversikt over simuleringsresultatet til store og komplekse systemer dersom det bare presenteres i tekst. En grafisk presentasjon sammen med en presentasjon i tekst gjør analyser av resultatet enklere.

5.1 Fleksibel forsyningskjede

Eksemplet som er tatt med i rapporten er en gjerne allerede blitt kjent med i GPenSim-manualen. Det er fordelen med å velge akkurat dette eksemplet. I GPenSim-manualen blir systemet modellert. I denne rapporten blir grafen og deler av simuleringsresultatet til systemet presentert grafisk.

Målet med systemet er å finne en fremgangsmåte for utvikling av fleksible forsyningskjeder. Dette avsnittet er i stor grad hentet fra [10]. For detaljer henvises leseren til [10]. Systemet er delt opp i 6 moduler. Klient, internett, initialisering, iterasjon, strategisk modul og taktisk modul. Under initialiseringen blir alle parametere tillagt startverdier fra denne eller tidligere samarbeid. I den strategiske modulen blir strategiske valg tatt som lokalisasjon av detaljist og grossist. Resultatet blir sendt videre til en taktisk modul, der taktiske valg gjøres. Det kan være planlegging av optimale transportruter mellom detaljist og grossist etc. Iterasjon gjør at parametrene som kommer fra den taktiske moduler blir sendt inn i den strategiske modulen igjen. Dersom det er få endringer i iterasjonsresultatene kan forsyningskjeden opprettes.

Under følger definisjonsfilene til to av modulene, strategisk og taktisk modul. Definisjonsfilene til de andre modulene finnes i GPenSim-manualen.

Strategisk modul

```
1   function [PN_name, set_of_places, set_of_trans, set_of_arcs,
2   ...
3           sources] = strategy_def(global_info)
4   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5   %% File: strategy_def.m: Definition of the Strategic Module
6   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7   PN_name = 'Strategic Module';
8   set_of_places = {'pB2', 'pB3'};
9   set_of_trans = {'tSD'};
10  set_of_arcs = {'pB2','tSD',1, 'tSD','pB3',1};
```

Taktisk modul

```
function [PN_name, set_of_places, set_of_trans, set_of_arcs,
...
        sources] = tactic_def(global_info)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% File: tactic_def.m: Definition of the Tactical & subtactical
modules
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PN_name = 'Tactical & sub-tactical Module(s)';
set_of_places = {'pB4', 'pB5'};
set_of_trans = {'tTD','tSUB1','tSUB2','tSUB3','tSUB4','tSUM'};
set_of_arcs = {'tTD','pB4',4, ...
    'pB4','tSUB1',1, 'pB4','tSUB2',1, 'pB4','tSUB3',1,
'pB4','tSUB4',1,...
    'tSUB1','pB5',1, 'tSUB2','pB5',1, 'tSUB3','pB5',1,
'tSUB4','pB5',1, ...
    'pB5','tSUM',4};
```

5.1.1 Tegne Petri net-grafen på skjerm

Definisjonsfiler for alle 6 moduler leses inn i GPenSimGUI-en. For å lese definisjonsfilene i GPenSimGUI-en trykker en på Load PNG-knappen. Vinduet som vist i figur 5.1 vil da dukke opp på skjermen.

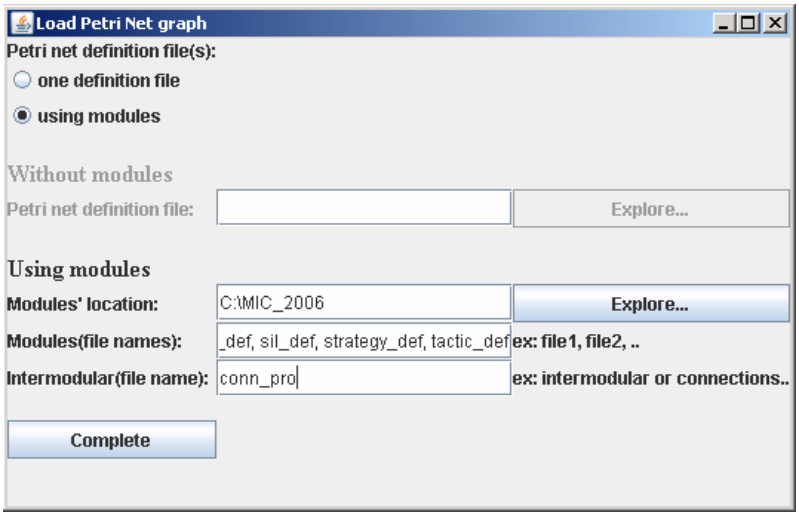


Fig. 5.1: Vinduet der Petri net-definisjonsfilene blir lest inn.

En velger om systemet er modellert med eller uten moduler. Dersom systemet ikke er delt opp i moduler, velges en Petri net-definisjonsfil. I det omtalte eksemplet er systemet modellert med moduler. For å slippe å lese inn en og en fil, leses i stedet mappen som inneholder definisjonsfilene. I et eget felt listes så filnavnene opp, separert med komma. Filnavnet til intermodular leses i et eget felt. Petri net-grafen tegnes på skjermen når brukeren trykker Complete.

Modulene blir plassert i store soner. For å få et oversiktsbilde av grafen kan en bruke zoomefunksjonen. En zoomer ut med punktum og zoomer inn med komma. En kan også zoome med mushjulet dersom en er eier av en mus med hjul på. Linjekryssene kommer fra arcs som kobler sammen moduler. I dette systemet kan en enkelt fjerne alle linjekryss ved å flytte på noen element. Før en flytter på grafen vil en endre sonekartet. Siden eksemplet inneholder få places og transitions vil store soner gi store avstander mellom elementene. For å få en mer sammenpakket graf kan en endre på sonestørrelsene. Det gjør en ved å trykke på zone size-knappen. Når en trykker på knappen vil et nytt vindu dukke opp på skjermen. Figur 5.2 viser et utklipp av vinduet.

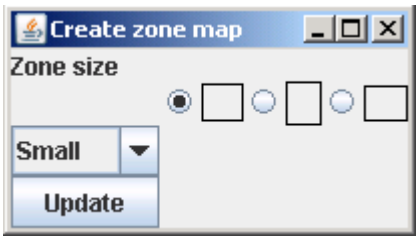


Fig. 5.2: Vindu for å skifte på størrelsen til sonene.

I dette vinduet kan en velge form og størrelse på sonene. Når en har valgt en ønsket størrelse og form på sonene trykker en på update-knappen. Grafen vil da bli tegnet på ny. I eksemplet velges liten sonestørrelse. Resultatet av endringer er vist i figur 5.3. Grafen er mer kompakt.

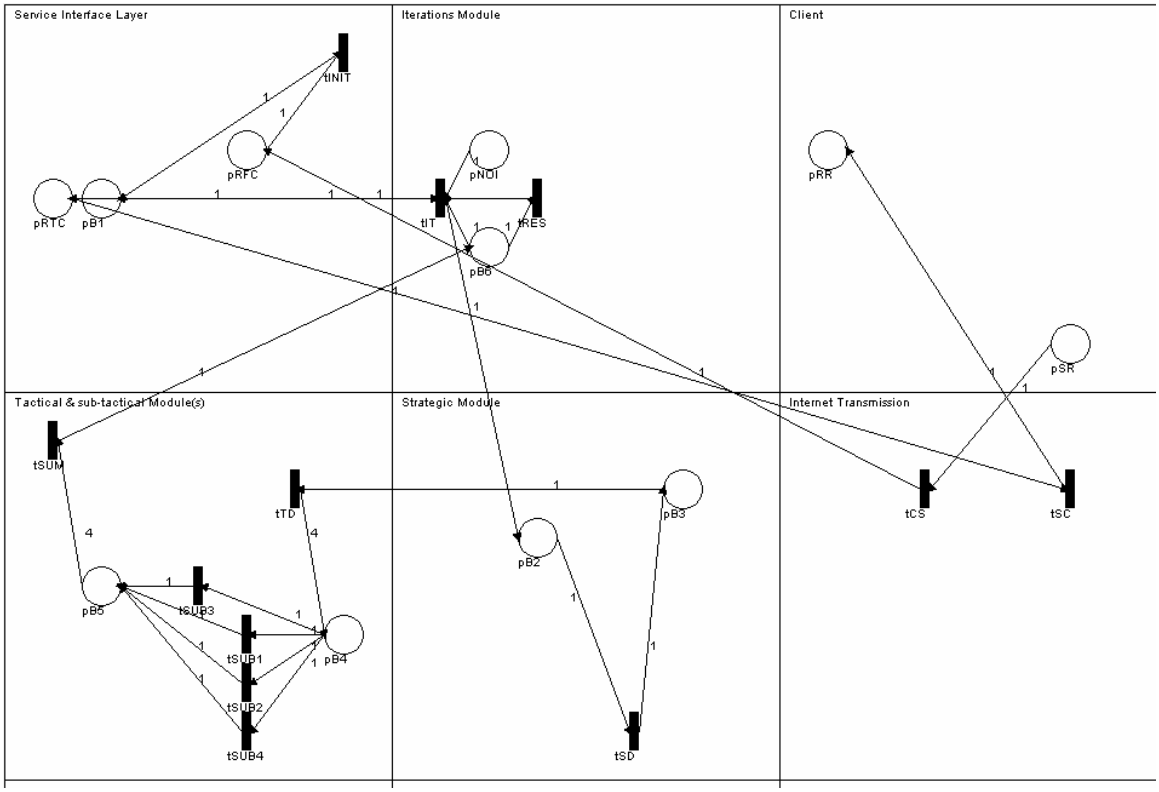


Fig. 5.3: Grafisk presentasjon av Petri net-graf.

Som en ser ut av figur 5.3 er det flere linjekryss. Linjekryssene er på tvers av moduler. Det er ingen linjekryss innad i modulene. I modulmenyen kan en velge å se på en og en modul. Utklipp av 4 av modulene viser under. Det er ingen linjekryss i de to siste modulene. Disse modulene inneholder ingen arcs.

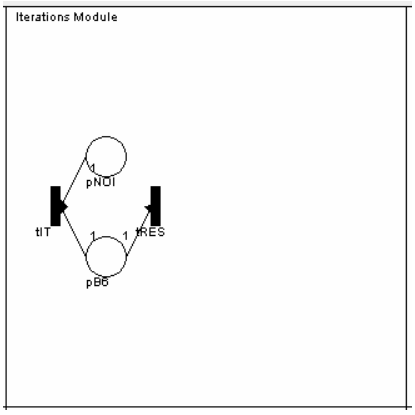


Fig. 5.4: Iterations Module.

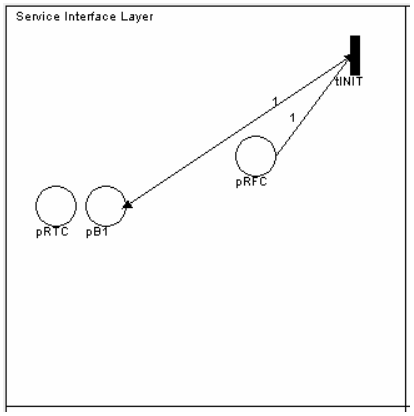


Fig. 5.5: Service Interface Layer.

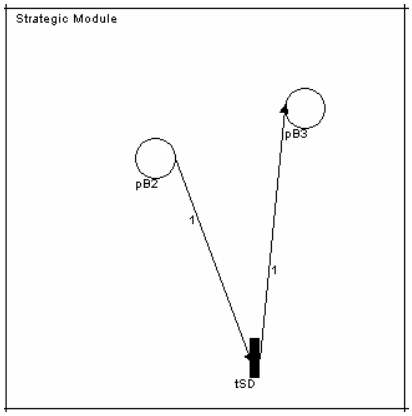


Fig. 5.6: Strategic Module

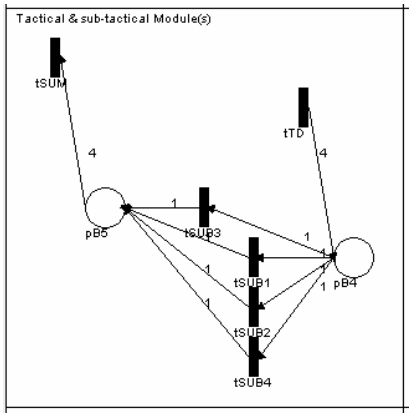


Fig. 5.7: Tactical & sub-tactical Module(s).

Siden grafen i dette eksemplet er relativt liten, kan en enkelt flytte places og transitions og/eller moduler, og i dette tilfellet fjerne alle linjekryss. Eventuelt kan en ignorere linjekryss og bruke menyen for å se på en og en modul. Da vil ikke arcs mellom moduler bli tegnet på skjermen. Ved å fjerne visning av sonene blir tegningen enda mer oversiktlig og fin som figur 5.8 viser.

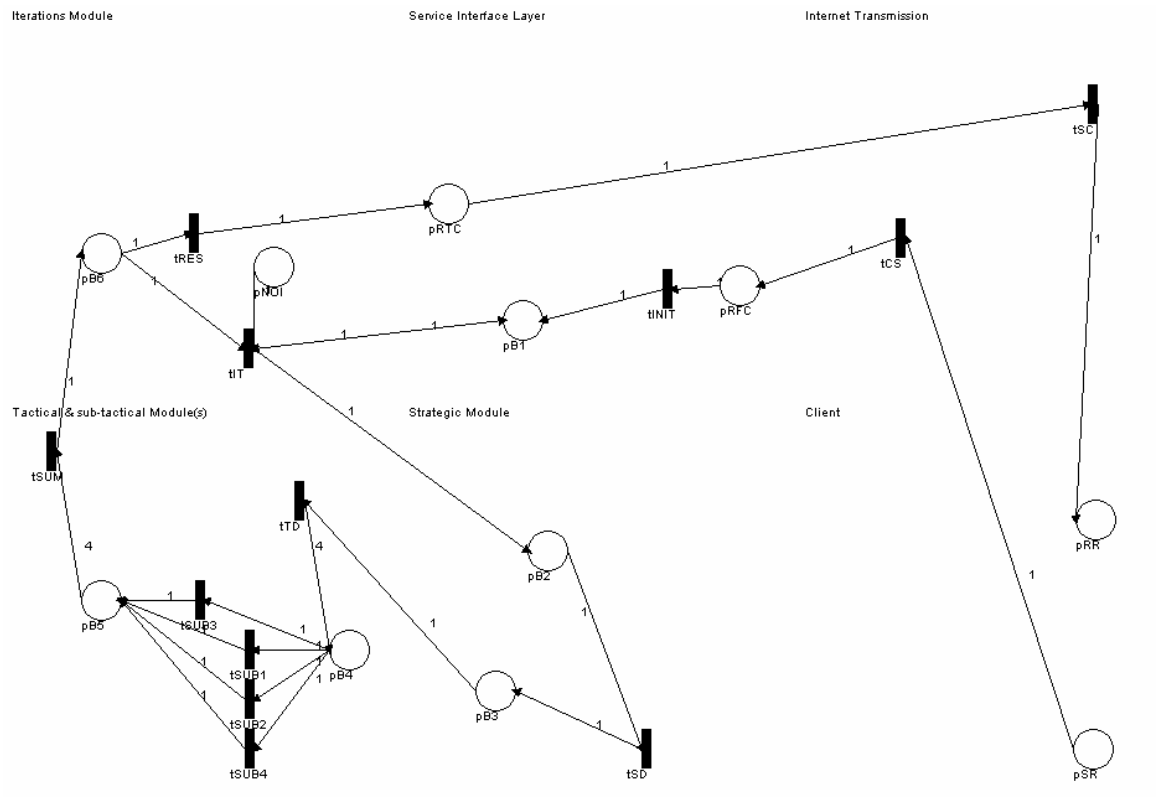


Fig. 5.8: Grafisk presentasjon av ordnet Petri net-graf.

5.1.2 Eksportere og presentere simuleringsresultat grafisk

I hovedsimuleringsfilen lagres informasjon om starttilstanden og firing times. Place-ene pSR, pNOI og pB3 starter med tokens. Hovedsimulasjonsfilen for systemet er vist i GPenSim-manualen[2]. Når hovedsimulasjonsfilen kjøres blir gpensim-funksjonen kalt med to parametere. Parametrene er Petri net-grafen og dynamisk informasjon. Resultatet av simuleringen lagres i sim-variabelen. For å eksportere resultatet til tekstfil kjøres filen under.

```
1   file_1 = fopen('simres.txt','wt')
2   fprintf(file_1,'Places\n')
3   fclose(file_1)
4   dlmwrite('simres.txt', sim.Place_Names, '-append', 'delimiter',
5   '')
6   file_1 = fopen('simres.txt','a')
7   fprintf(file_1,'Transitions\n')
8   fclose(file_1)
9   dlmwrite('simres.txt', sim.Transition_Names, '-append',
10  'delimiter', '')
11  file_1 = fopen('simres.txt','a')
12  fprintf(file_1,'State Diagram\n')
13  fclose(file_1)
14  dlmwrite('simres.txt', sim.State_Diagram, '-append',
15  'delimiter', ' ')
```

Dette gir en tekstfil kalt simres.txt som inneholder navn på places og transitions, og simuleringsresultatet. Tekstfilen leses inn i GPenSimGUI-en med Load sim results-knappen. Ved å trykke på knappen dukker en boks opp hvor simres.txt velges. Når en så trykker ok vil starttilstanden bli lest inn. Et nytt vindu dukker opp på skjermen. Dette vinduet inneholder informasjon om tilstandene. Det forteller hvor mange tokens hver place inneholder til hver tid.

Simuleringen spilles av ettersom en trykker på forrige- og nesteknappene. I figur 5.9 vises en grafisk presentasjon av en tilstand fra simuleringen. Informasjon om tilstanden finnes også i tekst. Denne er vist i et eget vindu som i figur 5.10.

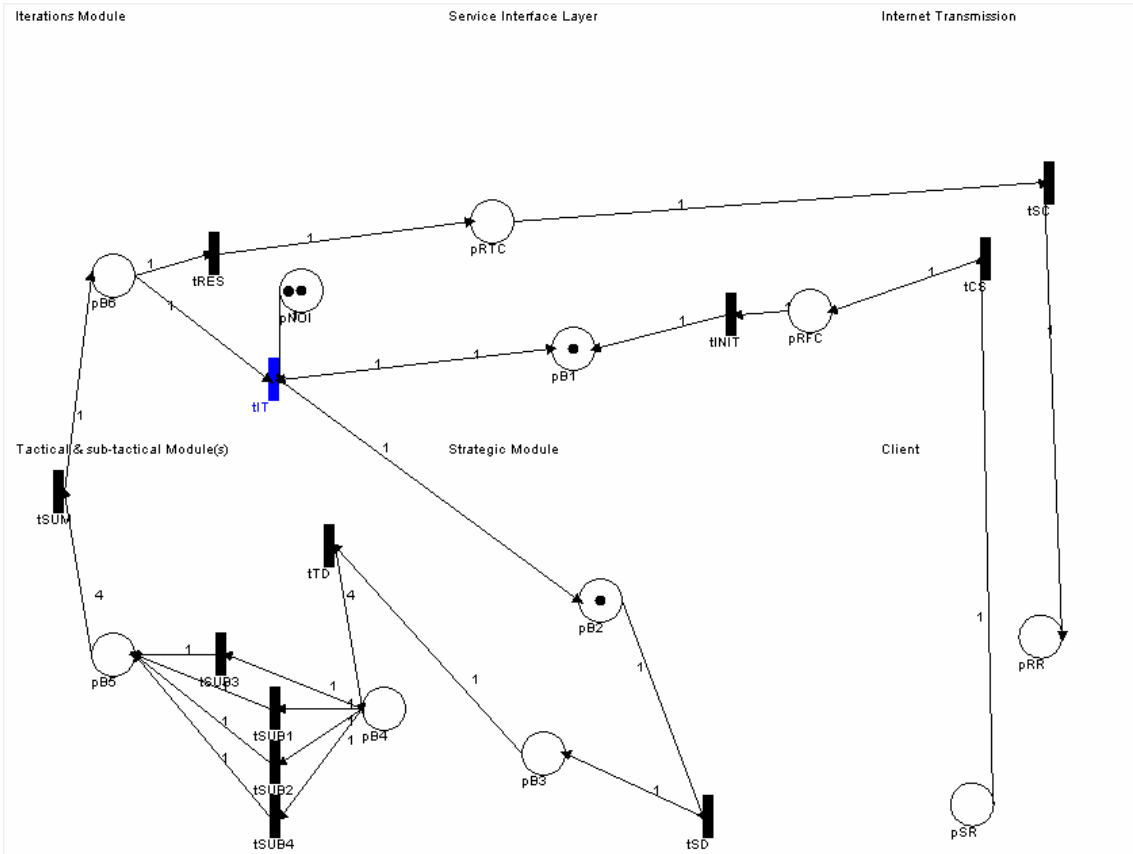


Fig. 5.9: Grafisk presentasjon av Petri net-graf.

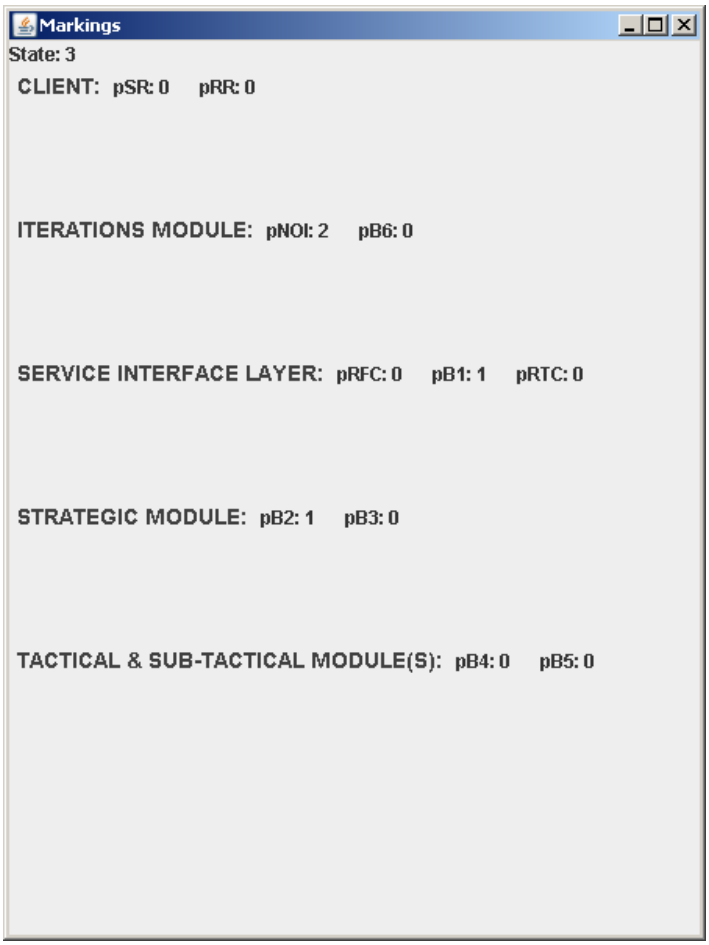


Fig. 5.10: Informasjon om en tilstand presentert i GPenSimGUI.

Informasjonen som er gitt under er resultatet slik det blir presentert i GPenSim. Dersom en sammenligner med figur 5.10 ser en at det er enklere å få oversikt over hvor de ulike places er definert i modellen i GPenSimGUI-en.

State: 3 Time: 5365.3375
Fired Transition: tIT
Current State:

pSR	pRR	pRFC	pRTC	pNOI	pB1	pB2	pB3	pB4	pB5
0	0	0	0	2	1	1	0	0	0

pB6
0

Dersom en vil skrive grafen ut på skriver eller til pdf-fil, er det enklest å først vise et rektangel rundt det som vil komme med på utskriften. Rektangelet eller boksen vises på skjermen ved å trykke på Print area-knappen. Skriver velges ved å trykke på print-knappen. Dersom en ikke får skrevet hele grafen ut på en side, kan en bruke piltastene til å flytte skjermbildet og dermed flytte utskriftsboksen. En kan så skrive ut resten av grafen på samme måte.

Kapittel 6

Konklusjon

Verktøyet som er utviklet i oppgaven fokuserer på GPenSims mangel på grafikk. Med en grafisk presentasjon blir sammenhengen mellom graf og simuleringsresultat klarere. På samme tid blir resultatet fra en simulering enklere å følge steg for steg. GPenSim lar systemer bli modellert i moduler. Denne fremgangsmåten viste seg å gjøre tegningen av grafen betraktelig mer oversiktlig.

Grafikken ble utviklet med Java2D som inneholder funksjoner for å tegne de nødvendige geometriske formene. Kapittel 2 introduserte Petri net, mens kapittel 3 omhandlet GpenSim. Kapittel 4 var det mest omfattende kapittelet. Kapittelet beskrev en metode for å plassere moduler og dermed grafen. Videre gav kapittelet en detaljert beskrivelse av metoden for å plassere places og transitions med fokus på å holde antall linjekryss lavt. Det ble innført en mal for å eksportere simuleringsresultat fra Matlab til en tekstfil. Videre ble en utskriftsfunksjon som Java2D tilbyr beskrevet. Med en utskriftsfunksjon kunne en skrive grafen ut og bruke utskriften i dokumentasjon av system. Kapittel 5 forklarte virkemåten til GPenSimGUI-en med et eksempel.

Grafen ble generert fra Petri net-definisjonsfiler. Som videreutvikling kunne GPenSimGUI-en blitt oppdatert med tilleggsfunksjonalitet. En funksjonalitet kunne vært å la brukeren tegne grafen i GPenSimGUI-en. Definisjonsfilene kunne så blitt generert ut fra tegningen.

Bibliografi

[1] Christos G. Cassandras and Stéphane Lafortune. Introduction to Discrete Event Systems. Petri Nets på side 225-234. ISBN 0-7923-8609-4.

[2] Reggie Davidrajuh. Lecture notes. Defining a Petri net.

[3] Reggie Davidrajuh. GPenSim user manual, version 3.1, 2009 - http://davidrajuh.net/gpensim/GPenSIM-User-Manual-v32.pdf

[4] The MathWorks, MATLAB. Exporting Text Data - http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/matlab_prog/f5-15544.html&http://www.google.no/search?hl=no&q=matlab+export+to+txt&btnG=Google-s%C3%B8k&meta=&aq=f&oq=

[5] Sun Microsystems, Java SE platform - http://java.sun.com/javase/.

[6] Sun Microsystems, Java 2D tutorial - http://java.sun.com/docs/books/tutorial/2d/index.html.

[7] Hong Zhang and Y. Daniel Ljiang. Computer Graphics using Java 2D and 3D. Pearson Presentice Hall 2006. ISBN 0-13-0351180-0

[8]: TopCoder. Ibackstrom. Geometry Concepts: Line Intersection and its Applications - http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=geometry2

[9] Quick Java Swing Tutorial for AWT Programmers, Printing Swing Components in Java 1.2 - http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-Printing.html.

[10] Davidrajuh, R. (2007) 'A service-oriented approach for developing daptive distribution chain', *Int. J. Services and Standards*, Vold. 3, No. 1, pp.64-78.