Compiling DEMOS 2000 into Petri Nets

Chris Tofts
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2001-274
October 30th , 2001*

E-mail: chris_tofts@hp.com

simulation, petri nets, compilation

System models are often studied through the use of computer simulation. This approach can be very attractive as it provides a simple dynamic view of the system under study, which can be used for debugging, and performance data can be gathered by repeated experiment. However, simulation has the limitations that all experiments do, the cost of each data point, potential to avoid particular cases and difficulty in establishing precisely what has been measured. A dual approach of using a simulation style to present and debug programs followed by a formal mathematical analysis would have the benefits of both approaches. We demonstrate how a large fragment of a process oriented simulation language DEMOS can be translated into Petri Nets. This translation has benefits in that it permits the simular access to the analysis tools for Petri Nets (we demonstrate automated translation both to the Sharpe and SPNP formats) and provides a succinct formal account of DEMOS. The current compiler is abstracted from the underlying Petri Net representation syntax and consequently can be rapidly retargeted at an arbitrary Petri Net representation. Given the ratio between the Petri Net representation and the DEMOS shown herein it may well be the case that the later is a significantly better approach to introducing Petri Net analysis.

* Internal Accession Date Only Approved for External Publication

Compiling DEMOS 2000 into Petri Nets

C. Tofts
HP Research Laboratories Bristol,
Filton Road, Stoke Gifford,
Bristol, BS34 8QZ,
chris_tofts@hp.com

October 25, 2001

Abstract

System models are often studied through the use of computer simulation. This approach can be very attractive as it provides a simple dynamic view of the system under study, which can be used for debugging, and performance data can be gathered by repeated experiment. However, simulation has the limitations that all experiments do, the cost of each data point, potential to avoid particular cases and difficulty in establishing precisely what has been measured. A dual approach of using a simulation style to present and debug programs followed by a formal mathematical analysis would have the benefits of both approaches. We demonstrate how a large fragment of a process oriented simulation language DEMOS can be translated into Petri Nets. This translation has benefits in that it permits the simular access to the analysis tools for Petri Nets (we demonstrate automated translation both to the Sharpe and SPNP formats) and provides a succinct formal account of DEMOS. The current compiler is abstracted from the underlying Petri Net representation syntax and consequently can be rapidly retargeted at an arbitrary Petri Net representation. Given the ratio between the Petri Net representation and the DEMOS shown herein it may well be the case that the later is a significantly better approach to introducing Petri Net analysis.

1 introduction

Many complex problems are studied by building simulation models that are intended to replicate selected aspects of their behaviour [12, 13, 14, 15, 17, 18, 21, 22, 23, 24, 28, 32, 33]. Once we have a model, the first step is to validate it, that is show it exhibits certain expected behaviours. The standard practice of running the simulation "enough" times cannot guarantee that all possible model paths will be covered, so harmful behaviours such as deadlock may be missed by this approach. Equally data can only be gathered in an experimentaly fashion with all of the exposure to statistical risk, which it is essentially impossible to quantify in a discrete setting with the potential for deadlock and livelock (infinite cost points) without a complete systems analysis. In other work, I have demonstrated[5, 30, 9, 10] with colleagues how some of these questions may be addressed by exploiting a process algebra[19, 20] as the formal underpining. This approach is very advantageous in two settings

1. proving correctness properties[5, 9, 10];
2. proving relationships between operational and denotational accounts[8] of the languages semantics, an essential check of our understanding.

However, whilst the process algbera approach does admit some elegant solutions in the case or formal performance analysis in situations based on counting (such as queues) it may be better to exploit an underlying formalisms that contains counters as native. So we examine the difficultly in formalling expressing a discrete event simulation language (in this instance DEMOS, although it can be regarded as prototypical[4] for all languages/descriptions in this class) as Petri Nets [25, 26, 27]

1.1 DEMOS

DEMOS [2, 6, 7] is a process oriented discrete event simulation language defined as an extension to the simula [3] language. The current version [11] has little changed from the original other than in how entities slave themselves to others. DEMOS has essentially 3 components.

1. entities, defined as instances of classes;
2. resources;
3. bins.

These elements all have definitional forms:

1. **entity(name,className,offfset)** and **class name=body**
2. **res(foo,amt);**
3. **bin(foo,amt);**.

Entities are the active elements of the system, resources can be thought of a semaphores and bins are points of asynchrony. Entities interact with other elements of the system in the following manner:

1. **getR(resN,amt)** and **putR(resN,amt)** repectively claim and free the amt of resource resN, an entity cannot return amounts of resource it does not own;
2. **getB(binN,amt)** and **putB(binN,amt)** repectively claim and free the amt of bin binN;
3. sync(name) slaves the current entity on name;
4. **getS(syn,amt)** and **putS(syn,amt)** repectively claim and free the amt of sync syn, an entity cannot return amounts of sync it does not own.
5. **Entity(name,className,offset)** spawns a new instance of className called name at time offset into the future;
6. **Hold(ti)** advances the simulation clock ti into the future.

Collectively getR(resN,amt), getB(binN,amt) and getS(syn,amt) are refered to as acquisitions, we permit the following compound operations on them:

1. **req[acq1,...,acqN]** requires that all of the requests can be granted simutaneously before the entity can proceed.
2. **try [req1] then Bd1 etry [req2] then Bd2 ... etry [reqN] then BdN;** will try each of the requests in turn until one can be satisfied, otherwise it blocks until one can be satisfied. Often used with ... etry [] then hold(t) as a non-blocking test.

Finally we allow loops

1. **repeat body** executes body forever;
2. **do n body** excutes body n times, usually used in setup
3. **while [req] body** as long as the requirement can be met execute the body. Note a requirement can be a boolean.

A demos program consists of an entity (conventionally referred to as main) that sets up class and other definitions and then invokes some entities to form the running system.

1.2 Petri Nets

Petri nets are one of the original methods used to describe concurrent systems [25]. There are many fine textbooks detailing how they are understood and extended in order to describe phenomena such as time and probability [1, 27]. What follows is a very brief description of the kind of nets we shall be using throughout the sequel.

Basically Petri nets have 4 components:

1. places;
2. events;
3. arcs;
4. tokens.

Essentially places are counters that hold tokens; events are actions that an observer may record (which may include the passage of time); the arcs establish which events are the consequences of tokens being in places. The pattern of tokens at places is called a *marking* Furthermore the nets behaviour is established by a *firing rule*, this determines which events can take place given any particular marking. Standardly the rule requires that there are tokens at all of the source arcs of an event, and that after the event has taken place a token is is added at all of the destination places.

In Figure 1, we show some simple nets illustrating standard program constructions, in this instance sequence, parallel execution and non-deterministic execution. by moving the tokens (solid blobs) in accordance with the firing rules we see the following. In the sequential we perform the event a followed by the event b. In the parallel we can perform either event a1 or event a2, in any order but can do both. In the non-determinism case we can perform either b1 or b2 but not both.

As an extension we may permit our events to represent durations, in the following simple example this is in the form of rates. The net in Figure 2 represents a system where the are a number of items to be sold. When a customer arrives (a token at place customer) the event selling can be performed (as long as there is more than 1 token at waiting) when the cusomer token is at sold it then waits for a time given by rate(arrive) until it appears to make another purchase. One at a time unsold items are polished at rate(polish) to keep them clean and then returned to the waiting area.

Further refinements include relative probabilities on competing transitions, the use of multiplicity arcs that require that some number (n say) of tokens are present before a firing can occur; and the introduction of inhibitor arcs, the presence of tokens preventing events from occuring.

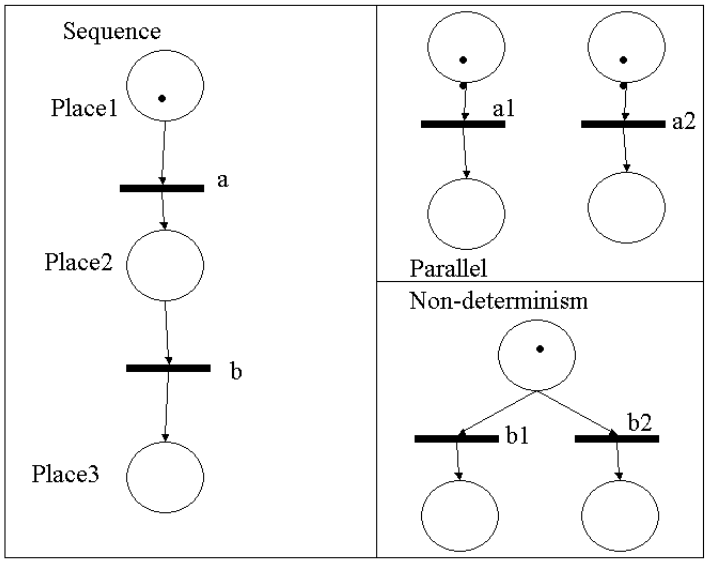

Figure 1: Three simple Petri nets.

2 Preliminary Example

Consider the classic[1] first Demos program given below. In this example Boats arrive at a dock, in order to dock they must have a jetty to dock at and two tugs to help them dock; they must retain a jetty whilst docked. The boats then unload and undock. To undock they require the assitance of a tug. When they have finished undocking then they relese control of their jetty. An obvious question of such a system how long the arrival interval between boats must be on average for the dock to cope?

```
Cons arrival=negexp(10.0);
Cons Leaving=2.0;
Cons dock=2.0;
Cons unload=normal(14,3);
Cons tug=3;
Cons jetty=2;
Cons sim_dur=28.0*24.0;

class boat=
{ hold(arrival);
  Entity(boat,boat,0);
   getR(jetties,1);
     getR(tugs,2);
          hold(dock);
     putR(tugs,2);

        hold(unload);
```

[1]If you are a Demos programmer!

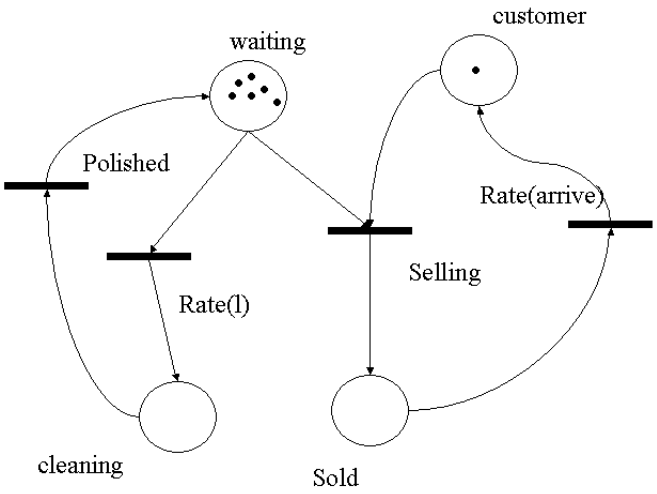

Figure 2: Simple timed Petri net.

```
      getR(tugs,1);
           hold(2.0);
      putR(tugs,1);
    putR(jetties,1);
} (***boat***)

Res(tugs,    tug);
Res(jetties, jetty);
Entity(boat,boat,0.0);
hold(sim_dur);
close;
```

This program can is equivalent to the net in Figure 3. In this net the named transitions have the duration assocaied with the name. The numbered ars require that many, rather than 1, tokens to fire. Only three places are named for clarity.

3 The full translation

This is presented diagramatically as a recursive descent on the syntax of DEMOS. It is assumed that the reader will understand a block stands for the remainder of a Net that has yet to be translated. This presentation is prefered to a more formal synactic presentation[2] for clarity. The diagramatic presentation of nets is generally more readable than the syntactic.

One important point is the role of the token that represents the local sequence controller [2]. A class is translated as a collection of places and transitions with an inital place where the tokens representing the number of live entity instances of that class are placed. Throughout the execution of the class body this token repesents the point to which execution of this class has proceeded.

[2]although that is what is used within the compiler.

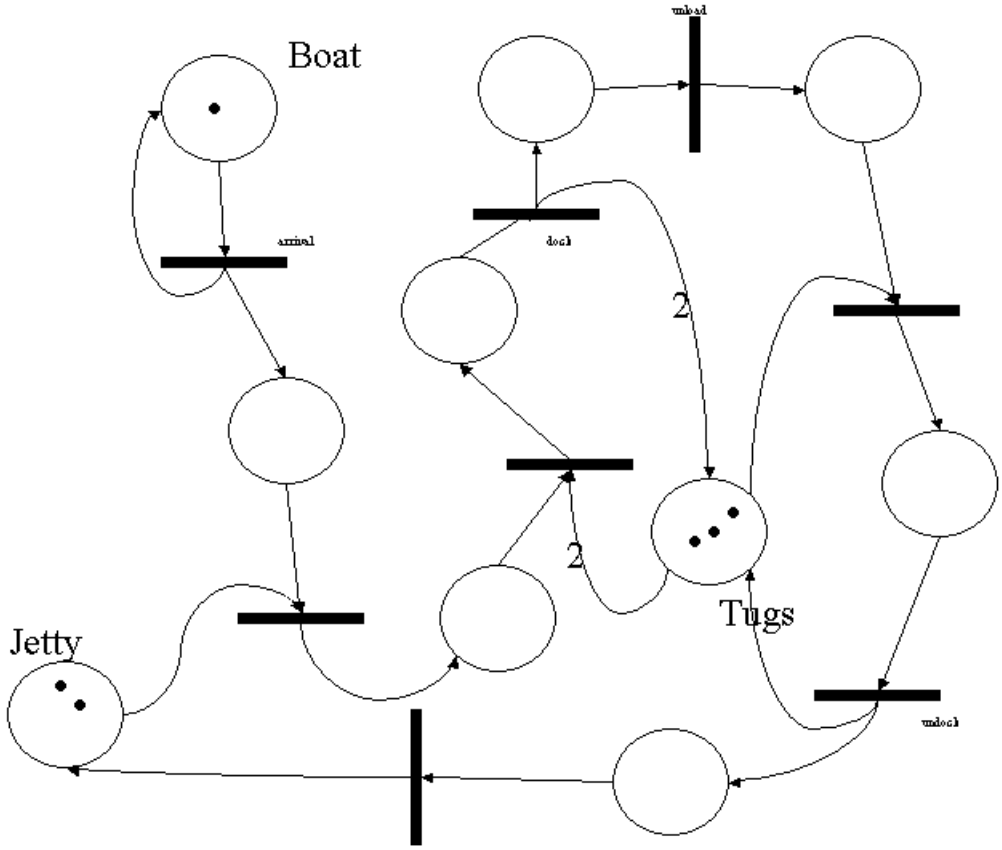

Figure 3: Boat system as Petri Net

3.1 Class definition

We translate the code fragment:

```
class fred={Body};
```

as the net given in figure 4.

We continue to translate the body in the manner that follows. This simple translation gives us a place to put tokens that indicate that an instance (entity) of the class should be started. These tokens are then used to express the sequentiality within the entity, and can be considered to be the local sequence controllers of DEMOS entities. These tokens are passed out of the entity. Notice that at this point we cannot add the arcs denoting when the entity is started those are added when that fragment of the demos code is encountered. This is the case for all of the definitional forms.

3.2 Resource definition

We translate the code fragment:

```
Res(fred,n);
```

as the net given in figure 5.

In this case the place acts simply as a counter of 'free' tokens. Arcs from the place will indicate **gets** on this resource and to the place with be the **puts**.

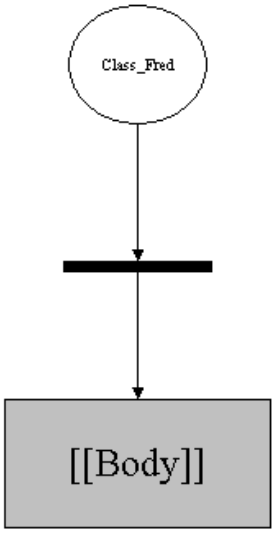

Figure 4: The translation of a class definition.

3.3 Bin definition

We translate the code fragment:

```
Bin(fred,n);
```

as the net given in figure 6.

As for resources the place acts simply as a counter of 'free' tokens. Arcs from the place will indicate **gets** on this bin and to the place with be the **puts**.

3.4 Entity creation

We translate the code fragment:

```
entity(foo,name,0);Body
```

as the net given in figure 7.

Here we essentially duplicate the sequentiality token, keeping one copy locally, our local sequence controller and placing one in the class place as the local sequence controller for the spawned entity. Whilst we can always normalise DEMOS programs to have zero delay on entity creation, we could equally introduce a delay transition on the path of the new sequence controller token.

3.5 Synchronisation point

We translate the code fragment:

```
Sync(name);Body
```

as the net given in figure 8.

Here we create two places. The first allows the master entity to claim the local sequence controller, and hence control of the local computation. The second allows it to be returned and consequently the local execution can then resume at the correct point.

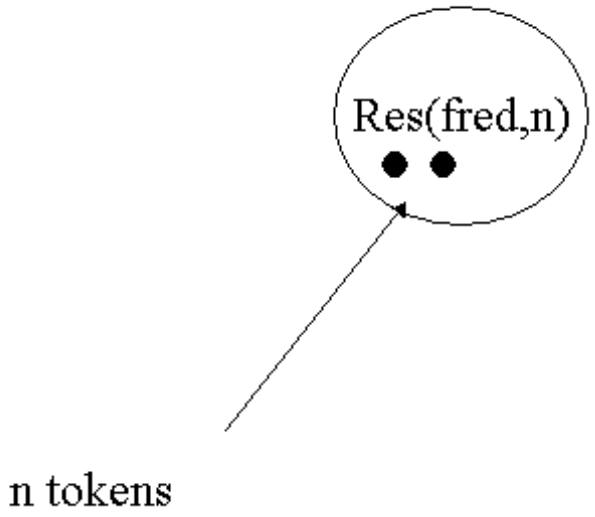

Figure 5: The translation of a resource definition.

3.6 Simple acquisition

We translate the code fragments:

```
getR(fred,amt);Body
getB(fred,amt);Body
getS(fred,amt);Body
```

as the net given in figure 9.

We use the event to distinguish the type of acquisition, useful for debugging/measurement. In this instance the presence of the local sequence control token and the required number of tokens on the source place allow the event to occur and the entity to proceed.

3.7 Returns

We translate the code fragments:

```
putR(fred,amt);Body
putB(fred,amt);Body
putS(fred,amt);Body
```

as the net given in figure 10.

We have two outgoing arcs from the transition. One is the token representing the continuation of the execution of the current entity. The other is the return of the tokens to the place where either the **bin** or **resource** contents are held, our the continuation place of the synchronisation.

3.8 Requires- multiway synchronisations

We translate the code fragments:

```
req[acq1,acq2,...,acqN];
```

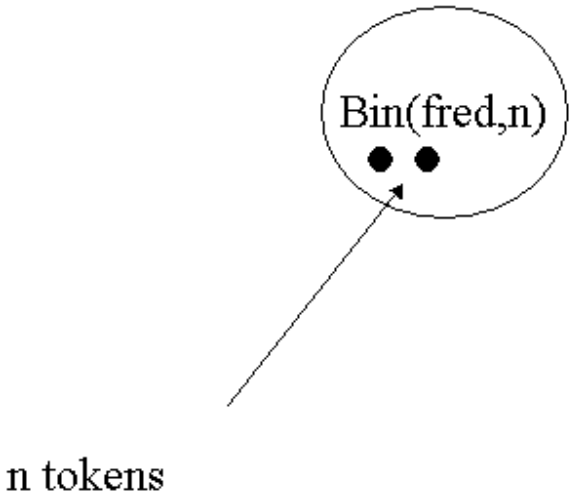

Figure 6: The translation of a bin definition.

where the `acqi` are drawn from `getR(n,a), getB(n,a), getS(n,a)` as the following net given in figure 11.

In this case the event can only fire when all of the requirement places **simultaneously** hold enough tokens for each of the arcs to fire. Afterwards we pass on the sequence token to the next state of the current entity.

3.9 Try choices (non-determinism)

We translate the code fragments:

```
try [req1] then {Bd1}
etry [req2] then {Bd2}
...
etry [reqN] then {BdN};
Body
```

where the `reqi` are requirements as above; as the net given in figure 12.

For any event to fire all of the places for its requirements must be met. Having fired it then passes the sequence token into the appropriate subnet. Hence, only one of the subnets is chosen. If more than one requirement could be met then the choice is resolved non-deterministically but **only** one instance of a subnet will acquire the execution token. If the Net framework admits priority than we can prioritise the events in presentation order to resolve the non-determinism. Fianlly, when (if) the subnet finishes execution it will pass the sequence token on to the continuation of the class.

3.10 Repeat

We translate the code fragment:

```
repeat {Body}
```

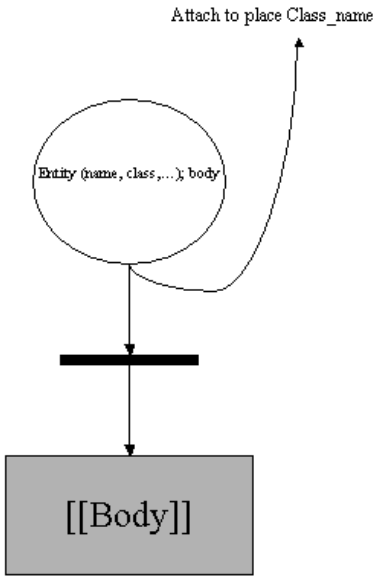

Figure 7: The translation of an entity creation.

as the net given in Figure 13:

This forces the local sequence token to return to this point afte the body has executed, thus restarting the body.

3.11 Do

We translate the code fragment:

```
Do n {body}
```

as the net given in figure 14.

In this instance we split the flow into a place with the sequence tokens and one with n tokens. Each time the body completes it puts a token back in the loop sequence controller. When the body has executed n times we can then firec the completion event as there is a token in the loop controller and n in the completion counter.

3.12 hold

We translate the code fragment:

```
hold(time);Body
```

as the net given in Figure 15, assuming that suitable timed transitions exist, for instance negative exponential times, fixed times, max/min,...:

We simply introduce a timed event that has the duration associated with the hold.

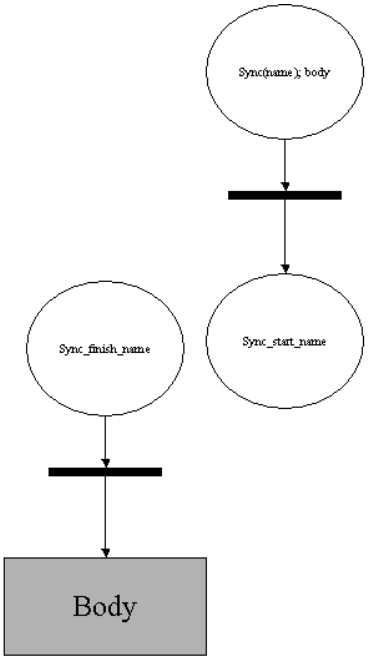

Figure 8: The translation of a synchronisation.

3.13 Main

To ensure that the system initialises correctly we translate main as any other class body. *However*, we place a *single* token in the first place to initiate the system.

3.14 Variables

Some extensions to Petri nets permit the addition of variables, in the presence of these extensions it would be straightforward to add translations for variables and boolean conditions in the class bodies and requirements respectively.

4 Automation

There are many tools for the analysis of Petri Nets [1], and there are also many different syntactic representations (Sharpe, SPNP, PML,...). To cope with this problem the implementation of the conversion from DEMOS to Petri nets works in two stages:

1. convert the system to an internal representation of a net, abstract notion of places and transitions in sml;

2. convert the internal representation into the appropriate syntax for the tool.

As nets have only 5 basic components this later transformation is a relatively straightforward task for any particular tool, as long as it does not rely on graphical input! The major limitation is the precise level of expressiveness of the nets that can be studied, and the forms of analysis the tool will permit. For a good overview see **www.daimi.au.dk/PetriNets/tools/**.

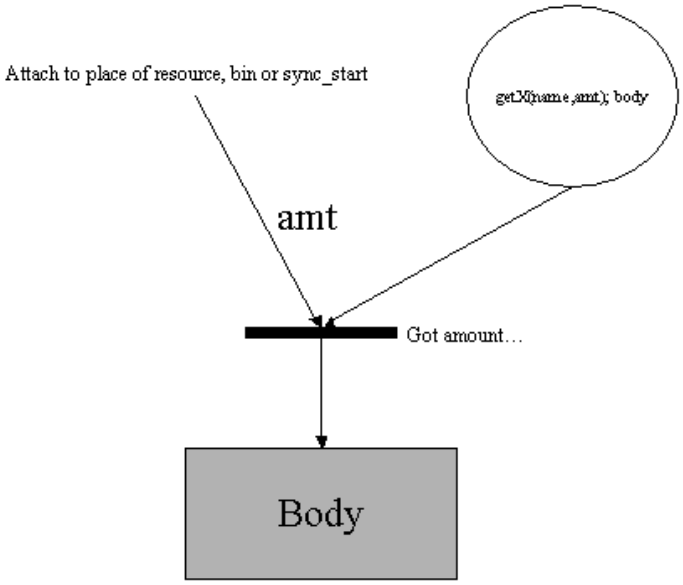

Figure 9: The translation of a acquisition.

The compiler from DEMOS to Petri nets is available from chris_tofts@hp.com as part of the demos simulation tool.

4.1 For the Sharpe tool

As an example this is the input syntax for the Sharpe tool [29] generated by the boat example (slightly altered as there are no fixed times, just negative exponential distributions) earlier.

```
gspn main
main_0 0
main 1
main_1 0
jetties 2
tugs 3
boat_11 0
boat_10 0
boat_9 0
boat_8 0
boat_7 0
boat_6 0
boat_5 0
boat_4 0
boat_3 0
boat_2 0
boat_1 0
```

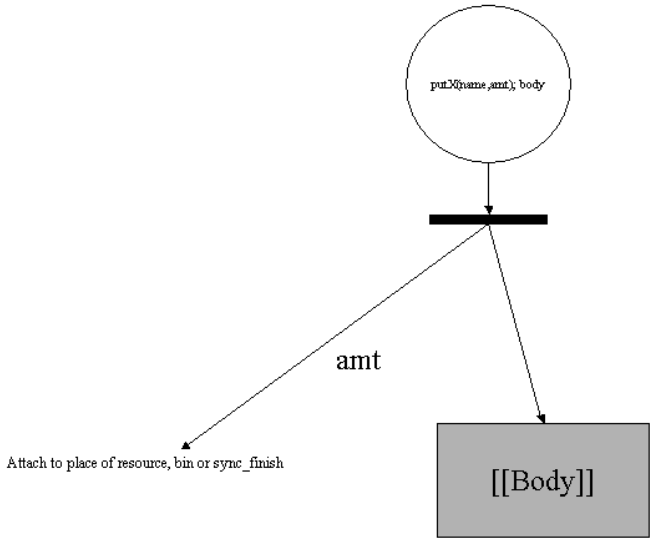

Figure 10: The translation of a return.

```
boat_0 0
end
boathl_9 ind undock
boathl_7 ind unload
boathl_5 ind dock
boathl_1 ind arrival
end
mainS ind 1
mainsp_1 ind 1
boatpR_11 ind 1
boatpR_10 ind 1
boatgR_8 ind 1
boatpR_6 ind 1
boatgR_4 ind 1
boatgR_3 ind 1
boatsp_2 ind 1
end
main mainS 1
main_0 mainsp_1 1
boat_10 boatpR_11 1
boat_9 boatpR_10 1
boat_8 boathl_9 1
boat_7 boatgR_8 1
tugs boatgR_8 1
boat_6 boathl_7 1
```

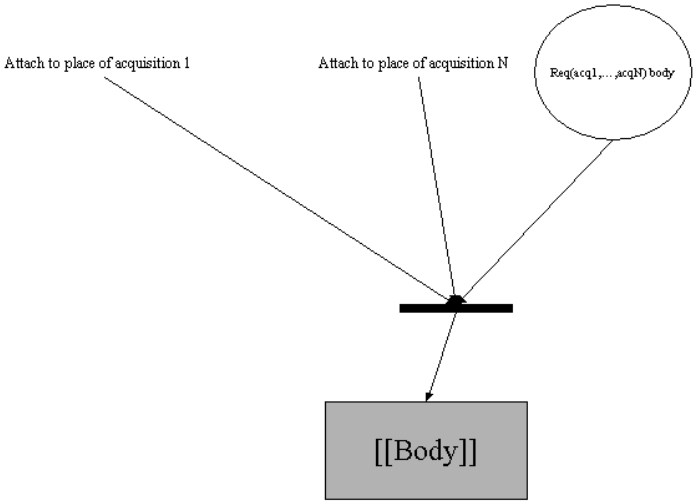

Figure 11: The translation of a require.

```
boat_5 boatpR_6 1
boat_4 boathl_5 1
boat_3 boatgR_4 1
tugs boatgR_4 2
boat_2 boatgR_3 1
jetties boatgR_3 1
boat_1 boatsp_2 1
boat_0 boathl_1 1
end
mainS main_0 1
mainsp_1 main_1 1
mainsp_1 boat_0 1
boatpR_11 boat_11 1
boatpR_11 jetties 1
boatpR_10 boat_10 1
boatpR_10 tugs 1
boathl_9 boat_9 1
boatgR_8 boat_8 1
boathl_7 boat_7 1
boatpR_6 boat_6 1
boatpR_6 tugs 2
boathl_5 boat_5 1
boatgR_4 boat_4 1
boatgR_3 boat_3 1
boatsp_2 boat_2 1
```

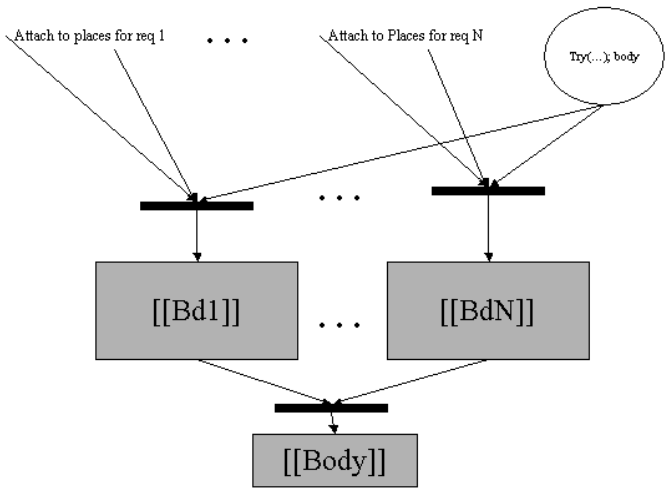

Figure 12: The translation of a try.

```
boatsp_2 boat_0 1
boathl_1 boat_1 1
end
end
```

4.2 As SPNP

An alternative presentation for nets that is popular is as extensions to C [31]

```
net () {
 place("main_0");
 place("main");
   init("main",1);
 place("main_1");
 place("jetties");
   init("jetties",2);
 place("tugs");
   init("tugs",3);
 place("boat_11");
 place("boat_10");
 place("boat_9");
 place("boat_8");
 place("boat_7");
 place("boat_6");
 place("boat_5");
```

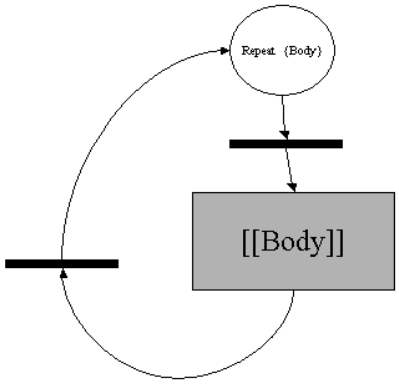

Figure 13: The translation of a repeat.

```
place("boat_4");
place("boat_3");
place("boat_2");
place("boat_1");
place("boat_0");
 trans("mainS");
 trans("mainsp_1");
 trans("boatpR_11");
 trans("boatpR_10");
 trans("boathl_9");
   rateval("boathl_9",undock);
 trans("boatgR_8");
 trans("boathl_7");
   rateval("boathl_7",unload);
 trans("boatpR_6");
 trans("boathl_5");
   rateval("boathl_5",dock);
 trans("boatgR_4");
 trans("boatgR_3");
 trans("boatsp_2");
 trans("boathl_1");
   rateval("boathl_1",arrival);
   iarc("mainS","main");
   iarc("mainsp_1","main_0");
   iarc("boatpR_11","boat_10");
```

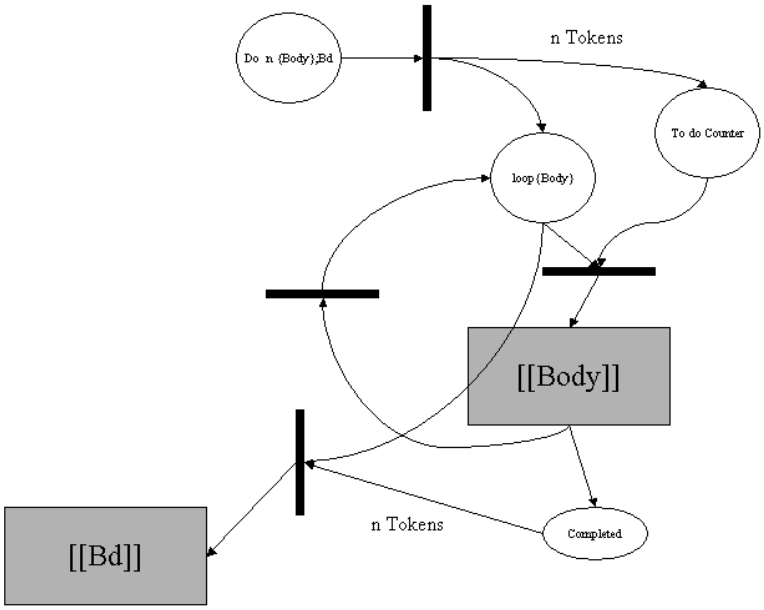

Figure 14: The translation of a do.

```
iarc("boatpR_10","boat_9");
iarc("boathl_9","boat_8");
iarc("boatgR_8","boat_7");
iarc("boatgR_8","tugs");
iarc("boathl_7","boat_6");
iarc("boatpR_6","boat_5");
iarc("boathl_5","boat_4");
iarc("boatgR_4","boat_3");
miarc("boatgR_4","tugs",2);
iarc("boatgR_3","boat_2");
iarc("boatgR_3","jetties");
iarc("boatsp_2","boat_1");
iarc("boathl_1","boat_0");
oarc("mainS","main_0");
oarc("mainsp_1","main_1");
oarc("mainsp_1","boat_0");
oarc("boatpR_11","boat_11");
oarc("boatpR_11","jetties");
oarc("boatpR_10","boat_10");
oarc("boatpR_10","tugs");
oarc("boathl_9","boat_9");
oarc("boatgR_8","boat_8");
oarc("boathl_7","boat_7");
oarc("boatpR_6","boat_6");
moarc("boatpR_6","tugs",2);
```

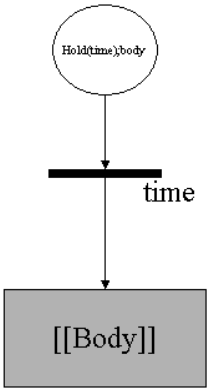

Figure 15: The translation of a hold.

```
    oarc("boathl_5","boat_5");
    oarc("boatgR_4","boat_4");
    oarc("boatgR_3","boat_3");
    oarc("boatsp_2","boat_2");
    oarc("boatsp_2","boat_0");
    oarc("boathl_1","boat_1");
}
```

5 Conclusions

The mapping between DEMOS and Petri nets seems to be a 'good' one. If we think of the DEMOS classes in terms of their states then we get one place for each state. The multiplicity of entities in that state is represented by the number of tokens at the place. Furthermore the subtlety of control[2] of allowing the scheduler to be simply another DEMOS process also carries over into the Petri net models with little additional cost. Indeed this seems to be the only extraneous cost incurred and could be removed with some simple optimisation (if required).

Given the current uptake of formal approaches, the simplicity of the DEMOS approach may well provide a better presentation language than the nets. Equally the cost of moving the DEMOS compiler from one net representation to another (provided they are not purely graphical) is low amounting to approximately 40 lines of SML code. DEMOS may also provide a better central notation for the various (and varied) Petri net analysis tools, the classical machine independence through compilation approach.

For the simulation side many of the current Petri net analysis tools [16] can numerically analyse very large systems. This allows for validation of simulation results and for the gathering of results

that are not exposed to the problems of experiments.

References

[1] G. Bolch, S. Greiner, H. de Meer and K. S. Trivedi, Queueing Networks and Markov Chains, John Wiley and Sons, New York, 1998.

[2] G. Birtwistle. *DEMOS — a system for discrete event modelling on Simula.* Macmillen, London, 1979.

[3] G. Birtwistle, O-J. Dahl, B. Myhrhaug, and K. Nygaard. *Simula begin.* Studentlitteratur, Lund, Sweden, 1973.

[4] G. Birtwistle, P.A.Luker, G.Lomow, and B.Unger. Process style packages for discrete event modelling: Experience from the transaction, activity, and event approaches. *Transactions of The Society for Computer Simulation*, 2(1):25–56, 1985.

[5] G. Birtwistle, R. Pooley, and C. Tofts. Characterising the Structure of Simulation Models in CCS. *Transactions of the Society for Computer Simulation*, 10(3):205–236, 1993.

[6] G. Birtwistle and C. Tofts. Operational Semantics of Process-Oriented Simulation Languages. Part 1: πDemos. *Transactions of the Society for Computer Simulation*, 10(4):299–333, 1993.

[7] G. Birtwistle and C. Tofts. Operational Semantics of Process-Oriented Simulation Languages. Part 2: μDemos. *Transactions of the Society for Computer Simulation*, 11(4):303–336, 1994.

[8] G. Birtwistle and C. Tofts. Relating Operational and Denotational Descriptions of πDemos. *Simulation Practice and Theory*, 5(1):1–33, 1997.

[9] G. Birtwistle and C. Tofts. Getting Demos Models Right - Part I: Practice. to appear *Transactions of the Society for Computer Simulation*, 2001.

[10] G. Birtwistle and C. Tofts. Getting Demos Models Right - Part II: ... and Theory. to appear *Transactions of the Society for Computer Simulation*, 2001.

[11] G. Birtwistle and C. Tofts. Operational Semantics of DEMOS 2000. Technical Report Number HPL-2001-263, Hewlett Packard Research Laboratories, Bristol, http://www.hpl.hp.com/techreports/2001/ 2001.

[12] P. Bratley, B. Fox, and L. Schrage. *A Guide to Simulation.* Springer Verlag, New York, 1987.

[13] A. S. Carrie. *Simulation of Manufacturing Systems.* J. Wiley and Sons, 1988.

[14] J. B. Evans. *Structures of Discrete Event Simulation.* Ellis Horwood, London, 1988.

[15] G. S. Fishman. *Principles of Discrete Event Simulation.* Wiley Interscience, New York, 1978.

[16] B. Haverkort, A. Bell and H. Bohnenkamp On the Efficient Sequential and Distributed Generation of Very Large Markov Chains from Stochastic Petri Nets Proceedings of the The 8th International Workshop on Petri Nets and Performance Models, 1998

[17] W. Kreutzer. *System simulation — programming styles and languages.* Addison Wesley, 1986.

[18] A. M. Law and J. S. Kelton. *Simulation Modelling and Analysis*. McGraw Hill, New York, 1982.

[19] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[20] R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.

[21] R. E. Nance. A History of Discrete Event Simulation Programming Languages. In T. J. Bergin and R. G. Gibons, editors, *History of Programming Languages*, pages 369–427. ACM Press and Addison-Wesley Publishing Company, 1996.

[22] G. T. Poole and J. Z. Szymankiewicz. *Using Simulation to Solve Problems*. McGraw Hill, London, 1977.

[23] A. A. B. Pritzker. *Introduction to Simulation and SLAMII*. Halstead Book Press, New York, 1984a.

[24] A. A. B. Pritzker and P. J. Kiviat. *Simulation with GASP II*. Prentice-Hall, New York, 1969.

[25] W. Reisig. A Primer in Petri Net Design, Springer Compass International, 1992.

[26] W. Reisig. Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets, Springer Verlag, 1998.

[27] W. Reisig & G. Rozenburg (Eds), Lectures on Petri Nets Basic Models : Advances in Petri Nets, Springer Verlag Lecture Notes in Computer Science, 1491, 1998.

[28] T. J. Schriber. *Simulation using GPSS/H*. John Wiley and Sons, New York, 1991.

[29] R. Sahner , K. S. Trivedi & A. Puliafito. Performance and Reliability Analysis of Computer Systems : An Example-Based Approach Using the Sharpe Software Package, Kluwer Academic Publishers, 1995

[30] C. Tofts and G. Birtwistle. A denotational semantics for a process-based simulation language. *ACM Transactions on Modelling and Simulation*, 8(3):281–305, 1998.

[31] K. Trivedi, Online manual for SPNP **http://www.ee.duke.edu/c̃hirel/IRISA/SPNP/spnpExampl** 1999.

[32] B. Ziegler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, New York, 1976.

[33] B. Ziegler. *Theory of Modelling and Simulation*. J. Wiley and Sons, New York, 1976.

A DEMOS to SPNP

In the context of small demos programs we show the compilation of the various constructs into SPNP for completeness.

DEMOS SPNP

```
class pr=
{repeat
 {putB(foo,2);
  hold(negexp(0.1));
 }
}

class c=
{repeat
 {getB(foo,1);
  hold(negexp(0.2));
 }
}

bin(foo,1);
entity(pr,pr,0);
entity(c,c,0);
close;
```

```
net () {
 place("main_0");
 place("main");
   init("main",1);
 place("main_2");
 place("main_1");
 place("foo");
   init("foo",1);
 place("c_2");
 place("c_1");
 place("c_0");
 place("pr_2");
 place("pr_1");
 place("pr_0");
  trans("mainS");
  trans("mainsp_2");
  trans("mainsp_1");
  trans("crp_3");
  trans("chl_2");
    rateval("chl_2",0.2);
  trans("cgB_1");
  trans("prrp_3");
  trans("prhl_2");
    rateval("prhl_2",0.1);
  trans("prpB_1");
    iarc("mainS","main");
    iarc("mainsp_2","main_1");
    iarc("mainsp_1","main_0");
    iarc("crp_3","c_2");
    iarc("chl_2","c_1");
    iarc("cgB_1","c_0");
    iarc("cgB_1","foo");
    iarc("prrp_3","pr_2");
    iarc("prhl_2","pr_1");
    iarc("prpB_1","pr_0");
    oarc("mainS","main_0");
    oarc("mainsp_2","main_2");
    oarc("mainsp_2","c_0");
    oarc("mainsp_1","main_1");
    oarc("mainsp_1","pr_0");
    oarc("crp_3","c_0");
    oarc("chl_2","c_2");
    oarc("cgB_1","c_1");
    oarc("prrp_3","pr_0");
    oarc("prhl_2","pr_2");
    oarc("prpB_1","pr_1");
    moarc("prpB_1","foo",2);
}
```

```
res(sem,1);

class cs=
{repeat
 {getR(sem,1);
  hold(negexp(0.2));          (*cs*)
  putR(sem,1);
  hold(negexp(0.4));          (*non cs*)
 }
}

entity(cs,cs,0);
entity(cs,cs,0);
entity(cs,cs,0);
close;
```

```
net () {
 place("main_0");
 place("main");
   init("main",1);
 place("main_3");
 place("main_2");
 place("main_1");
 place("cs_4");
 place("cs_3");
 place("cs_2");
 place("cs_1");
 place("cs_0");
 place("sem");
   init("sem",1);
 trans("mainS");
 trans("mainsp_3");
 trans("mainsp_2");
 trans("mainsp_1");
 trans("csrp_5");
 trans("cshl_4");
   rateval("cshl_4",0.4);
 trans("cspR_3");
 trans("cshl_2");
   rateval("cshl_2",0.2);
 trans("csgR_1");
   iarc("mainS","main");
   iarc("mainsp_3","main_2");
   iarc("mainsp_2","main_1");
   iarc("mainsp_1","main_0");
   iarc("csrp_5","cs_4");
   iarc("cshl_4","cs_3");
   iarc("cspR_3","cs_2");
   iarc("cshl_2","cs_1");
   iarc("csgR_1","cs_0");
   iarc("csgR_1","sem");
   oarc("mainS","main_0");
   oarc("mainsp_3","main_3");
   oarc("mainsp_3","cs_0");
   oarc("mainsp_2","main_2");
   oarc("mainsp_2","cs_0");
   oarc("mainsp_1","main_1");
   oarc("mainsp_1","cs_0");
   oarc("csrp_5","cs_0");
   oarc("cshl_4","cs_4");
   oarc("cspR_3","cs_3");
   oarc("cspR_3","sem");
   oarc("cshl_2","cs_2");
   oarc("csgR_1","cs_1");
}
```

```
res(sem,1);
bin(foo,0);

class csP=
{repeat
 {getR(sem,1);
  hold(negexp(0.2));          (*cs*)
  putR(sem,1);putB(foo,1);
  hold(negexp(0.4));          (*non cs*)
 }
}

class csc=
{repeat
 {req[getR(sem,1),getB(foo,1)];
  hold(negexp(0.2));          (*cs*)
  putR(sem,1);
  hold(negexp(0.4));          (*non cs*)
 }
}

entity(csp,csP,0);
entity(cs,csP,0);
entity(csc,csc,0);
close;
```

```
net () {
 place("main_0");
 place("main");       init("main",1);
 place("main_3");place("main_2");
 place("main_1");place("csc_4");
 place("csc_3");place("csc_2");
 place("csc_1");place("csc_0");
 place("csP_5");place("csP_4");
 place("csP_3");place("csP_2");
 place("csP_1");place("csP_0");
 place("foo");place("sem");  init("sem",1);
  trans("mainS");
  trans("mainsp_3");
  trans("mainsp_2");
  trans("mainsp_1");
  trans("cscrp_5");
  trans("cschl_4");rateval("cschl_4",0.4);
  trans("cscpR_3");
  trans("cschl_2");rateval("cschl_2",0.2);
  trans("cscreq_1");
  trans("csPrp_6");
  trans("csPhl_5");rateval("csPhl_5",0.4);
  trans("csPpB_4");
  trans("csPpR_3");
  trans("csPhl_2");rateval("csPhl_2",0.2);
  trans("csPgR_1");
    iarc("mainS","main");iarc("mainsp_3","main_2");
    iarc("mainsp_2","main_1");iarc("mainsp_1","main_0");
    iarc("cscrp_5","csc_4");iarc("cschl_4","csc_3");
    iarc("cscpR_3","csc_2");iarc("cschl_2","csc_1");
    iarc("cscreq_1","csc_0");iarc("cscreq_1","foo");
    iarc("cscreq_1","sem");iarc("csPrp_6","csP_5");
    iarc("csPhl_5","csP_4");iarc("csPpB_4","csP_3");
    iarc("csPpR_3","csP_2");iarc("csPhl_2","csP_1");
    iarc("csPgR_1","csP_0");iarc("csPgR_1","sem");
    oarc("mainS","main_0");oarc("mainsp_3","main_3");
    oarc("mainsp_3","csc_0");oarc("mainsp_2","main_2");
    oarc("mainsp_2","csP_0");oarc("mainsp_1","main_1");
    oarc("mainsp_1","csP_0");oarc("cscrp_5","csc_0");
    oarc("cschl_4","csc_4");oarc("cscpR_3","csc_3");
    oarc("cscpR_3","sem");oarc("cschl_2","csc_2");
    oarc("cscreq_1","csc_1");oarc("csPrp_6","csP_0");
    oarc("csPhl_5","csP_5");oarc("csPpB_4","csP_4");
    oarc("csPpB_4","foo");oarc("csPpR_3","csP_3");
    oarc("csPpR_3","sem");oarc("csPhl_2","csP_2");
    oarc("csPgR_1","csP_1");
}
```

```
res(sem,1);
bin(foo,0);

class csP=
{repeat
 {getR(sem,1);
  hold(negexp(0.2));          (*cs*)
  putR(sem,1);putB(foo,1);
  hold(negexp(0.4));          (*non cs*)
 }
}

class csc=
{repeat
 {try[getR(sem,1),getB(foo,1)] then
   {hold(negexp(0.2));         (*cs*)
     putR(sem,1);
   }
  etry[getB(foo,1)] then {hold(negexp(2));}
     hold(negexp(0.4));         (*non cs*)
 }
}

entity(csp,csP,0);
entity(cs,csP,0);
entity(csc,csc,0);
close;
```

```
net () {
 place("main_0");place("main");  init("main",1);
 place("main_3");place("main_2");place("main_1");
 place("csc_2");place("csc_tBR1_2");place("csc_tBR1_1");
 place("csc_tBR0_3");place("csc_tBR0_2");place("csc_tBR0_1");
 place("csc_1");place("csc_0");place("csP_5");
 place("csP_4");place("csP_3");place("csP_2");
 place("csP_1");place("csP_0");
 place("foo");
 place("sem");  init("sem",1);
  trans("mainS");trans("mainsp_3");
  trans("mainsp_2");trans("mainsp_1");
  trans("cscrp_3");
  trans("cschl_2"); rateval("cschl_2",0.4);
  trans("csc_tBR1hl_2"); rateval("csc_tBR1hl_2",2);
  trans("csc_EBR1_1");trans("csc_tBR0pR_3");
  trans("csc_tBR0hl_2"); rateval("csc_tBR0hl_2",0.2);
  trans("csc_EBR0_1");trans("cscCF_1");
  trans("csPrp_6");
  trans("csPhl_5"); rateval("csPhl_5",0.4);
  trans("csPpB_4");trans("csPpR_3");
  trans("csPhl_2"); rateval("csPhl_2",0.2);
  trans("csPgR_1");
    iarc("mainS","main");iarc("mainsp_3","main_2");
    iarc("mainsp_2","main_1");iarc("mainsp_1","main_0");
    iarc("cscrp_3","csc_2");iarc("cschl_2","csc_1");
    iarc("cscCF_1","csc_tBR1_2");iarc("csc_tBR1hl_2","csc_tBR
    iarc("csc_EBR1_1","csc_0");iarc("csc_EBR1_1","foo");
    iarc("cscCF_1","csc_tBR0_3");iarc("csc_tBR0pR_3","csc_tBR
    iarc("csc_tBR0hl_2","csc_tBR0_1");iarc("csc_EBR0_1","csc_
    iarc("csc_EBR0_1","foo");iarc("csc_EBR0_1","sem");
    iarc("csPrp_6","csP_5");iarc("csPhl_5","csP_4");
    iarc("csPpB_4","csP_3");iarc("csPpR_3","csP_2");
    iarc("csPhl_2","csP_1");iarc("csPgR_1","csP_0");
    iarc("csPgR_1","sem");
    oarc("mainS","main_0");oarc("mainsp_3","main_3");
    oarc("mainsp_3","csc_0");oarc("mainsp_2","main_2");
    oarc("mainsp_2","csP_0");oarc("mainsp_1","main_1");
    oarc("mainsp_1","csP_0");oarc("cscrp_3","csc_0");
    oarc("cschl_2","csc_2");oarc("csc_tBR1hl_2","csc_tBR1_2")
    oarc("csc_EBR1_1","csc_tBR1_1");oarc("csc_tBR0pR_3","csc_
    oarc("csc_tBR0pR_3","sem");oarc("csc_tBR0hl_2","csc_tBR0_
    oarc("csc_EBR0_1","csc_tBR0_1");oarc("csPrp_6","csP_0");
    oarc("csPhl_5","csP_5");oarc("csPpB_4","csP_4");
    oarc("csPpB_4","foo");oarc("csPpR_3","csP_3");
    oarc("csPpR_3","sem");oarc("csPhl_2","csP_2");
    oarc("csPgR_1","csP_1");
}
```

```
class simple=
{repeat
 {sync(foo);
  hold(negexp(0.3));
 }
}

class handle=
{repeat
 {getS(foo,1);
  hold(negexp(0.3));
  putS(foo,1);
  hold(negexp(0.2));
 }
}

entity(s,simple,0);
entity(h,handle,0);
close;
```

```
net () {
 place("main_0");
 place("main");
   init("main",1);
 place("main_2");
 place("main_1");
 place("handle_4");
 place("handle_3");
 place("handle_2");
 place("handle_1");
 place("handle_0");
 place("simple_2");
 place("simple_1");
 place("simple_0");
  trans("mainS");
  trans("mainsp_2");
  trans("mainsp_1");
  trans("handlerp_5");
  trans("handlehl_4");
    rateval("handlehl_4",0.2);
  trans("handlepS_3");
  trans("handlehl_2");
    rateval("handlehl_2",0.3);
  trans("handlegS_1");
  trans("simplerp_3");
  trans("simplehl_2");
    rateval("simplehl_2",0.3);
    iarc("mainS","main");
    iarc("mainsp_2","main_1");
    iarc("mainsp_1","main_0");
    iarc("handlerp_5","handle_4");
    iarc("handlehl_4","handle_3");
    iarc("handlepS_3","handle_2");
    iarc("handlehl_2","handle_1");
    iarc("handlegS_1","handle_0");
    iarc("handlegS_1","simple_0");
    iarc("simplerp_3","simple_2");
    iarc("simplehl_2","simple_1");
    oarc("mainS","main_0");
    oarc("mainsp_2","main_2");
    oarc("mainsp_2","handle_0");
    oarc("mainsp_1","main_1");
    oarc("mainsp_1","simple_0");
    oarc("handlerp_5","handle_0");
    oarc("handlehl_4","handle_4");
    oarc("handlepS_3","handle_3");
    oarc("handlepS_3","simple_1");
    oarc("handlehl_2","handle_2");
    oarc("handlegS_1","handle_1");
    oarc("simplerp_3","simple_0");
    oarc("simplehl_2","simple_2");
}
```