PMR 5237

Modelagem e Design de Sistemas Discretos em Redes de Petri

Aula 11: Modelagem e análise de Sistemas

Prof. José Reinaldo Silva
reinaldo@poli.usp.br

Andamento do curso

Meta -> capítulo 9 do livro texto

Leitura da semana

Capítulos 5 e 6 (revisão)

Capítulos 7 e 8 (modelagem)

Milestone 5 -> feedback

Milestone 6 (primeira versão do artigo completo)

Teremos mais duas semanas de aula

Modelagem: dilema do começo

O começo de qualquer projeto, ou da modelagem de um sistema (novo ou revisitado) é sempre marcado pelo dilema cuja metáfora mais conhecida é o do dilema se vem primeiro o ovo ou a galinha.

Buscando um processo de projeto

Nos casos em que intuitivamente temos um sistema que é plenamente representado por uma rede clássica, e, mais do que isso, onde este modelo é facilmente e completamente interpretado modelo, é fácil de entender que somente uma demanda por múltiplos casos de simetria ou dobramento nos levaria a apelar para um sistema de alto nível.

No exercício que acabamos de ver poderíamos introduzir vários tipos de peça no processo de fabricação, cuja "receita" seria dada por diferentes combinações das operações utilizadas operando nas diferentes máquinas. Neste caso seria bastante atraente a distinção de marcas por tipos. Mas será esta a sequencia adequada em todos os casos?

Requisitos: o início de um grande problema

Certamente o início de todo projeto bem sucedido é a *eliciação* de um conjunto de requisitos que descreve com precisão as funcionalidades do sistema que deve ser modelado e implementado. Portanto para se chegar a um processo de projeto que termine na modelagem do sistema em Redes de Petri é preciso ter em conta de que este projeto deve começar com uma boa representação de requisitos. A representação mais usada e difundida para isso é com certeza a UML.

Análise de Requisitos, Síntese de redes, Building blocks

Uma hipótese bastante tentadoras seria ter um processo de projeto que pudesse ser reduzido a uma sequencia de transformações de transferência semântica entre linguagens, começando pela UML. Uma rede de Petri derivada de um ou mais diagramas UML poderia servir de base para um processo de **análise destes requisitos** e mais tarde com as devidas mudanças inseridas resultar no modelo do sistema.

Este processo poderia perfeitamente ser combinado com o método conhecido como ***building blocks*** onde várias partes da rede poderiam ser sintetizadas como descrito no parágrafo acima até se ter, por composição o sistema completo.

Partindo dos Casos de Uso

Casos de uso podem ser representados segundo uma forma diagramática como proposto em (Silva e Santos, 2004)

Symbol	Description
●	Start of a process (Use Case)
⦿	End of a process (Use Case)
→	Start of an event of basic flow of process
↳	Start of an event of alternative flow of process
◇	Start of a conditional event
$\rightsquigarrow^n$	Jump of current iteration to iteration ^{n}
||	Sequence of concurrent events

Silva, J.R. e Santos, E.A.; Applying Petri Nets to Requirements Validation, ABCM Symposium Series in Mechatronics, vol 1, pp. 508-517.

Um exemplo: o caixa eletrônico

Caso de uso textual

1. Initiate Withdraw – Customer inserts bank's card in the card reader on the ATM machine
2. Verify Bank Card – The ATM reads the account code from the magnetic strip on the bank card and checks if it is an acceptable bank card
3. Enter PIN – The ATM asks for the customer's PIN code (4 digits)
4. Verify PIN – The account code and PIN are verified to determine if the account is valid and if the PIN entered is the correct PIN for the account. For this flow, the account is a valid account and the PIN is the correct PIN associated with this account
5. Select Withdraw – The ATM displays the different alternatives available at this ATM. In this flow, the bank customer always selects "Cash Withdraw"
6. Enter Amount – The ATM asks for the amount to withdraw. For this flow the customer selects a pre-set amount ($10, $20, $50, or $100)
7. Authorization – The ATM initiates the verification process with the Banking System by sending the Card ID, PIN, Amount, and Account information as a transaction. For this flow, the Banking System is online and replies with the authorization to complete the cash withdrawal successfully and updates the account balance accordingly
8. Dispense – The Money is dispensed
9. Receipt – The receipt is printed and dispensed. The ATM also updates the internal log accordingly
10. Return Card – The Bank Card is returned

1 ● Initiate Withdraw – Customer inserts bank's card in the card reader on the ATM machine; (

1.1 ◇ Is the card valid? Verify Bank Card The ATM reads the account code from the magnetic strip on the bank's card and checks if it is an acceptable card;

1.1.1 ↳Send message of invalid card – If the card isn't an acceptable card, send an appropriate message. ⇝1.9

1.2 → Enter PIN – The ATM asks for the customer's PIN code (4 digits);
1.3 ◇ Is PIN Correct? – Verify PIN – The account code and PIN are verified to determine if the account is valid and if the PIN entered is the correct PIN for the account;

1.3.1 ◇ Is the account valid? – Verify account code – The account code is verified;

1.3.1.1 ↳Send message of invalid account – The Banking system returns a code indicating the account could not be found or is not an account which allows withdrawals. ⇝1.9

1.3.2 ◇ Is the final try? – Checks the number of tries – The customer has three tries to enter the correct PIN;

1.3.2.1 ↳Send message of incorrect PIN – The ATM send an appropriate message. ⇝1.2

1.3.3 ↳Retain card – on the final try the card is retained, ATM returns to Ready State, and this use case terminates. ⇝2

1.4 ◇ Have money the ATM? – Select Withdraw – The ATM displays the different alternatives available at this ATM. In this flow, the bank customer always selects "Cash Withdraw";

1.4.1 ↳Send message ATM out of Money – If the ATM is out of money, the "Cash Withdraw" option will not be available. ⇝1.4

1.5 ◇ Sufficient funds and does not exceed the daily limit? – Enter amount – The ATM asks for the amount to withdraw. For this flow the customer selects a pre-set amount ($10, $20, $50, or $100);

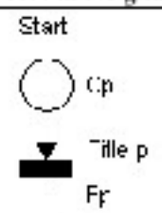
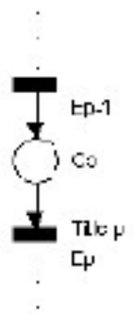
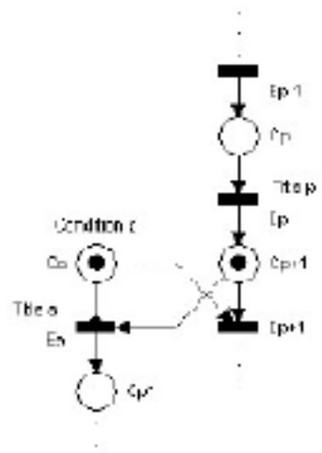
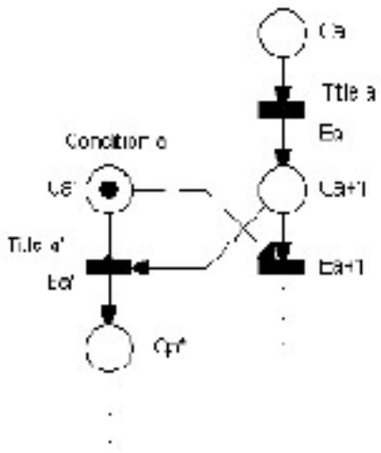
1.5.1 ◇ Sufficient funds? – Check sufficient funds – Check if the funds are sufficient;

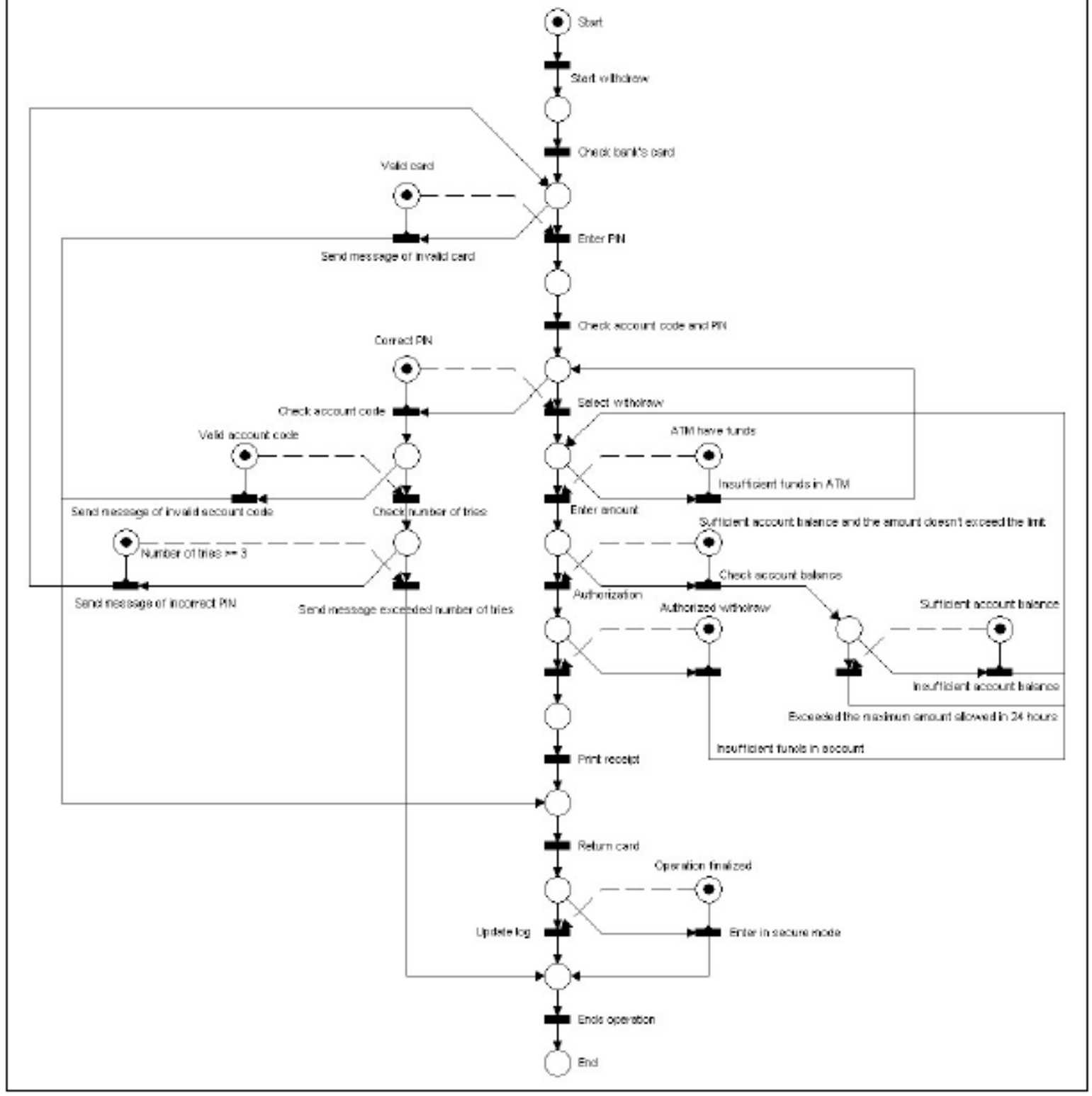
1.5.1.1 ↳Send message insufficient funds in ATM – The ATM contain insufficient funds to dispense the requested amount. ⇝1.5

```
<BF>          ::= <initial> { <eventB> } <final> { <AF> }
<initial>     ::= <label> "●" <title> " " <description> ";" "("
<label>       ::= <number> { "." <number> }
<number>      ::= <sequential number>
<title>       ::= <string>
<description> ::= <string>
<string>      ::  <any_caractor> { <any_caractor> }
<eventB>      ::= [ <eventBB> | <eventBA> ]
<eventB>      ::= <labelB> [ ( "→" <title> " " <description> ";" ) |

                             ( "◇" <condition> "-" <title> "-" <description> ";" <eventA>
                             ) ] { <eventB> } |

                  "||" "(" <eventB> { <eventB> } ")" "(" <eventB> { <eventB> } ")" "||" {
                  <eventB> }
<labelB>      ::= <label> "." <number>
<condition>   ::= <string>
<eventA>      ::= <labelA> [ ( "↳" <title> " " <description> <branch> ) |

                             ( "◇" <condition> "-" <title> "-" <description> ";" <eventA>
                             ) ] { <eventA> }

<labelA>      ::= <labelB> "." <number>
<branch>      ::= "↷" [ <label> | <labelB> ]
<final>       ::= ")" <label> "◉" <title> "-" <description> ";"
<AF>          ::= <eventUA>
<eventUA>     ::= <label> "!" <title> "-" <description> <branch> { <eventUA> }
```

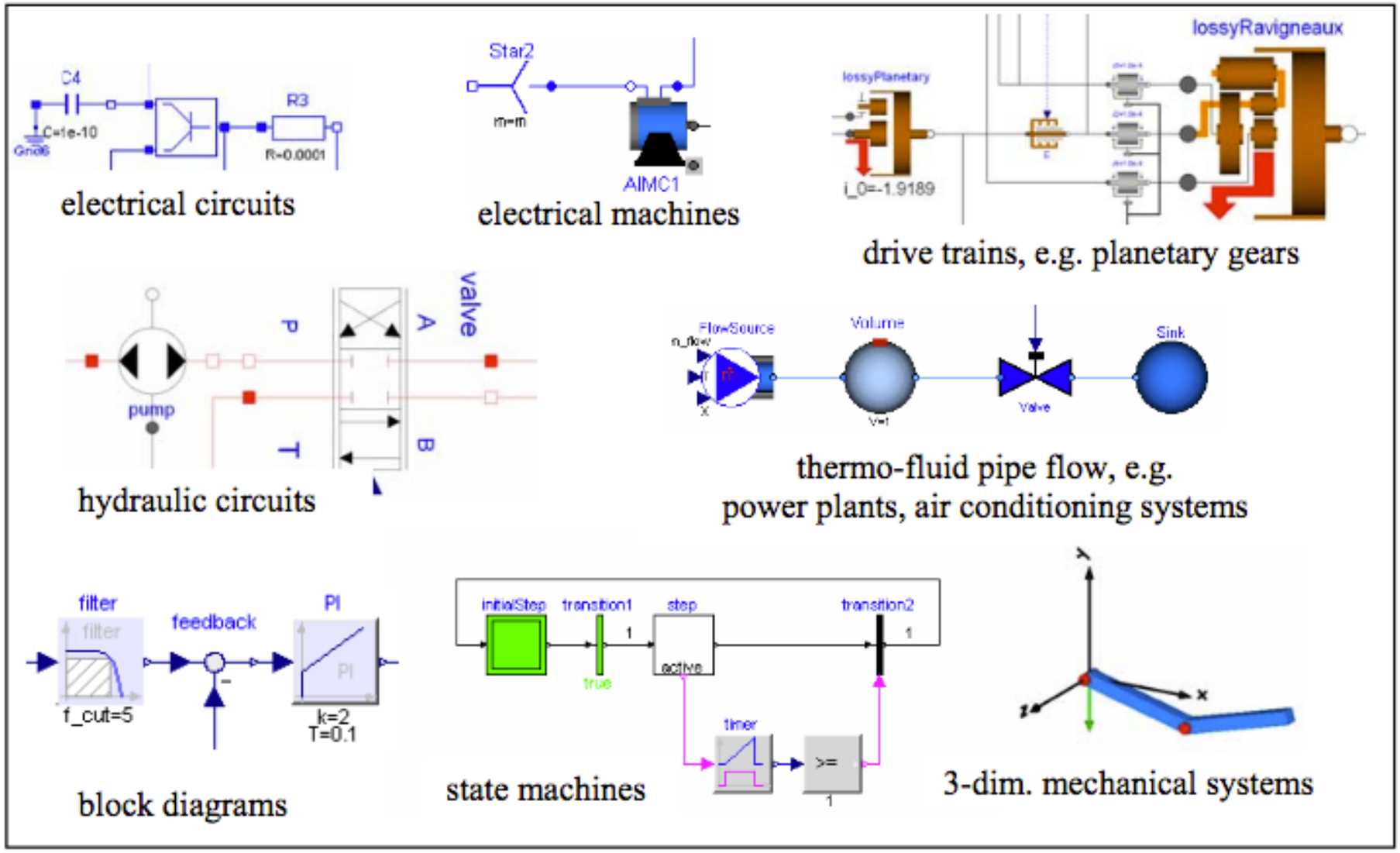
Where:

- eventB → Event of Basic flow
- labelB → Event id of Basic flow
- eventA → Alternative Event of Basic flow
- labelA → Alternative Event id of Basic flow
- eventUA → Unconditional Event of Alternative Event
- labelUA → Unconditional Event id of Alternative Event

Event	BNF notation	Petri net segment
Initial	p ● Title - Description;	
Basic	p → Title - Description;	
Basic conditional and alternative normal[1]	p◇Condition-Title-Descript.; a → Title-Description ~p*	
Alternative conditional and normal	a◇Condition-Title-Descript.; a' → Title-Description ~p*	


Start
Start withdraw
Check bank's card
Valid card
Send message of invalid card
Enter PIN
Check account code and PIN
Correct PIN
Check account code
Select withdraw
Valid account code
ATM have funds
Send message of invalid account code
Check number of tries
Insufficient funds in ATM
Enter amount
Number of tries == 3
Sufficient account balance and the amount doesn't exceed the limit
Check account balance
Send message of incorrect PIN
Send message exceeded number of tries
Authorization
Authorized withdraw
Sufficient account balance
Insufficient account balance
Exceeded the maximum amount allowed in 24 hours
Insufficient funds in account
Print receipt
Return card
Operation finalized
Update log
Enter in secure mode
Ends operation
End

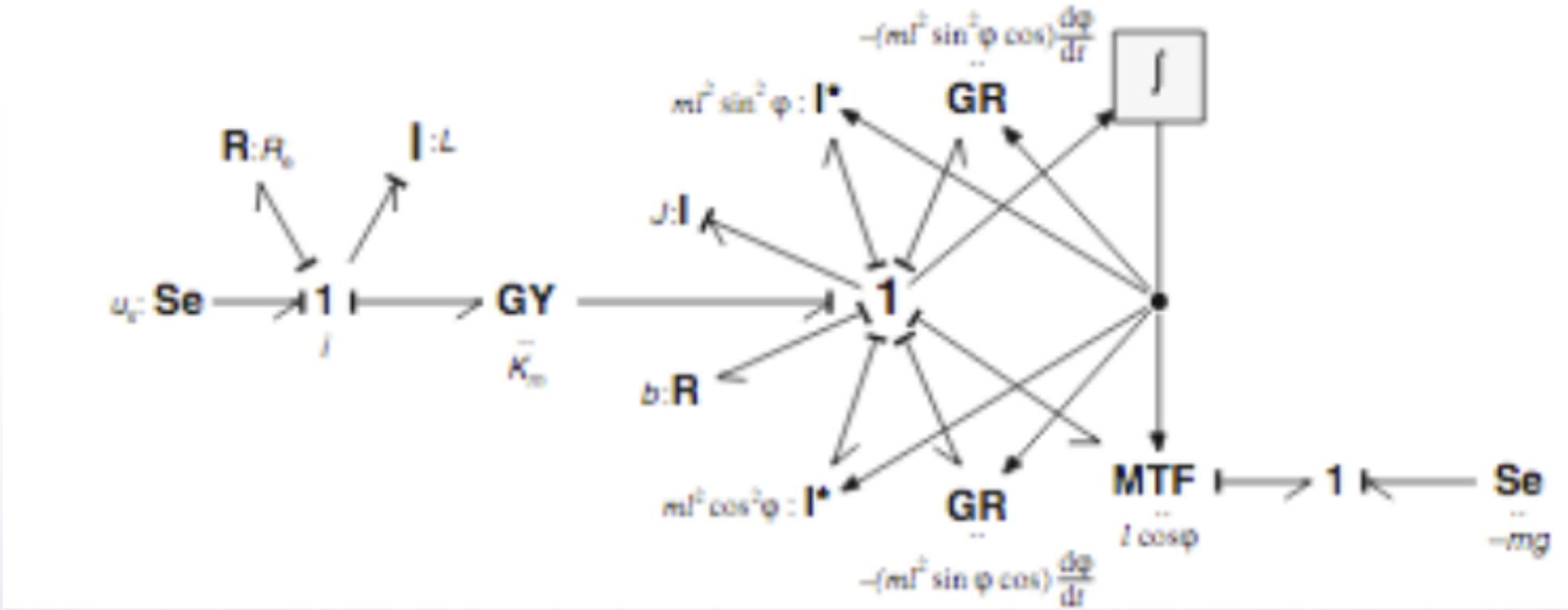
Estrutura do Modelica



Especificação em Modelica

```
parameter Real a[:]"Vector of coefficients";
final parameter Integer nx = size(a,1) - 1;
Real sigma[nx];
Real A[nx,nx] "System matrix";
equation
  A = [zeros(nx-1,1), identity(nx-1);
       -a[1:na-1]/a[na]                ];
  sigma = Modelica.Math.Matrices.singularValues(A);
```

Bond Graph


R:R_0
I:L
u_c: Se
1
i
GY
$\bar{K}_m$
J:I
b:R
$ml^2\sin^2\varphi$: I*
GR
$-(ml^2\sin^2\varphi\cos)\frac{d\varphi}{dt}$
∫
1
MTF
$l\cos\varphi$
1
Se
$-mg$
$ml^2\cos^2\varphi$: I*
GR
$-(ml^2\sin\varphi\cos)\frac{d\varphi}{dt}$

Modelagem esquemática

X

Modelagem conceitual/causal

Exploring CPN Tools

Teoria:
modelagem
req. formais
análise

Ferramentas:
editor
simulador
verificador

Aplicação:
especificação
validação
verificação
implementação

Theory

- The Coloured Petri Nets formalism
  - Introduction of hierarchies
- Modelling primitives
  - Channels, inhibitor arcs, test arcs, ...
  - Monitoring framework
- Analysis methods
  - Symmetry methods
  - Sweep-line method

August 27, 2002 Søren Christensen, MOCA'02

Ferramentas para Redes de Alto nível

1990-2002

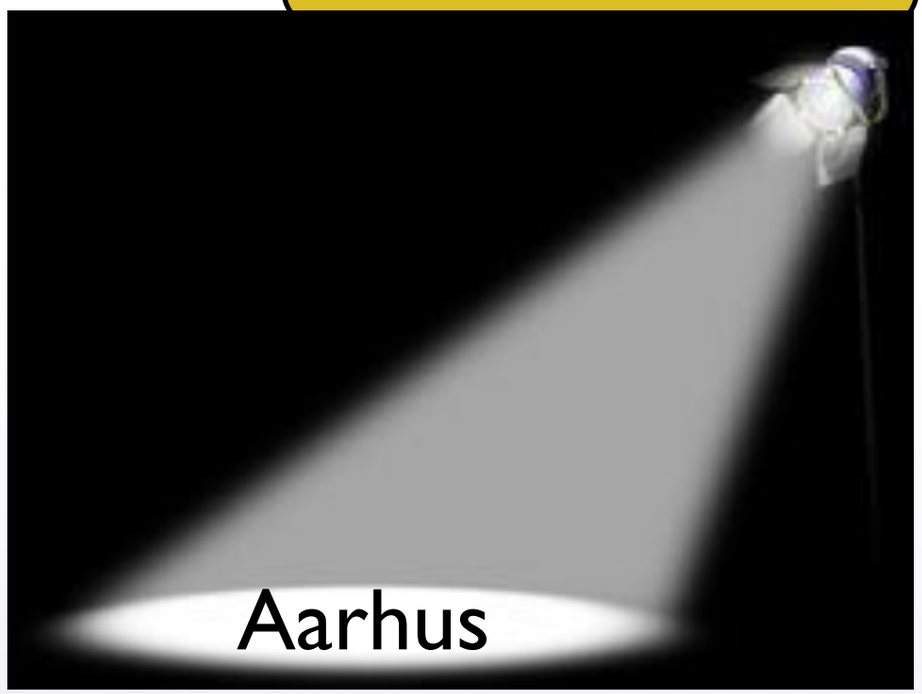
Design CPN

2003-…

CPN Tools

1990-2002

Design CPN



Aarhus Univ., Denmark

2003-…

CPN Tools

Eindhoven Univ., Netherlands

cpntools.org

CPN Tools
Access/CPN
- Books
- Documentation
- Download

FAQ
- Getting Started

Grade/CPN
- Knowledge Base
- Licensing
- Support

Contact
Publications

Google™ Cus Search

G+1 158

CPN T... 1.1K likes

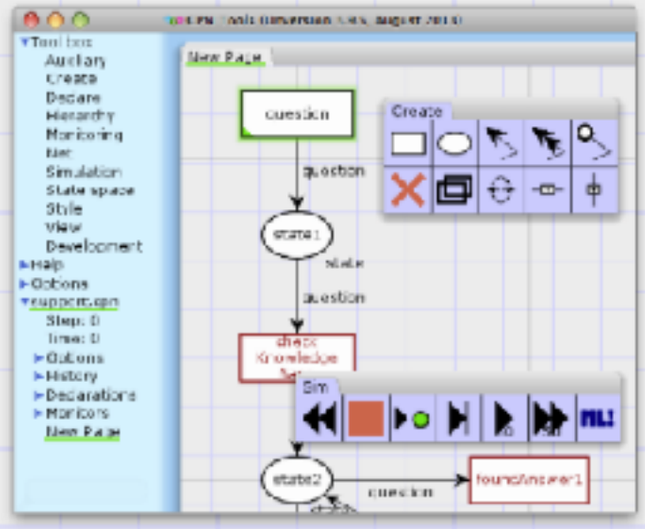
Like Page

Be the first of your friends to like this

Home | Download | Getting Started | Documentation | Support | Contact

CPN Tools is a tool for *editing*, *simulating*, and *analyzing* Colored Petri nets.

The tool features incremental syntax checking and code generation, which take place while a net is being constructed. A fast simulator efficiently handles untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information, such as boundedness properties and liveness properties.



New Features in Version 4.0

- Declarative constraints
- 3rd part extensions
- Simplified use of non-colored nets
- Support for export to PNML
- Support for real and time colorsets
- Improved support for time (time intervals and state-space reduction)
- Simplified state-space analysis
- Fresh new look

CPN Tools is originally developed by the CPN Group at Aarhus University from 2000 to 2010. The main architects behind the tool are Kurt Jensen, Søren Christensen, Lars M. Kristensen, and Michael Westergaard. From the autumn of 2010, CPN Tools is transferred to the AIS group, Eindhoven University of Technology, The Netherlands.

Latest Version of CPN Tools

The latest released version is version **4.0.1** from February 2015. Get it from the Download page. For a full list of new features for each version of CPN Tools, Access/CPN, and Grade/CPN refer to the Whats New?-list.

Michael's blog on CPN Tools

- Is Google's Go-bot a Breakthrough for Artificial Intelligence? (2016/03/25 22:12)
- Improved Paradigm for Analysis, Verification

Tradeoff

- **More information in tokens**
  - color sets, functions, etc.
  - behavior may be hidden in “code”
  - extreme case: all behavior folded into one place and one transition
- **More information in network**
  - possibly spaghetti networks to encode simple things
  - behavior may be incomprehensible
  - cannot be parameterized
  - extreme case: (infinite) classical Petri net

Uma outra extensão importante para as redes de Petri é o tempo. O tempo é uma grandeza positiva que corre inexoravelmente do passado para o futuro, isto é, desde um instante quando se começou a contar o tempo para um tempo futuro.

Assim, o tempo seria normalmente uma grandeza contínua (teoricamente). Entretanto o tempo de simulação que costumamos contar é discreto e marca tão somente a sucessão de estados. É possível portanto ter um *tempo discreto*, contado por um relógio independente dos eventos analisados.

Chandler Ramchandani, em sua dissertação de mestrado no MIT em Setembro de 1973 propôs a primeira temporização que se tem noticia das Redes de Petri.

ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS

by

Chander Ramchandani

B.Tech., Indian Institute of Technology, New Delhi
(1968)

S.M., Massachusetts Institute of Technology
(1970)

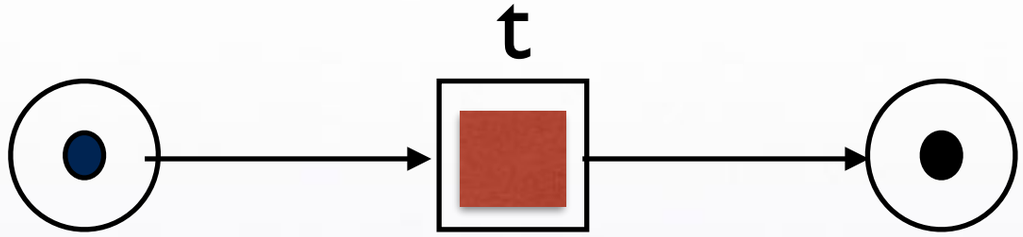
SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September, 1973

Signature of Author _____
Department of Electrical Engineering, July 3, 1973

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Graduate Students

No caso de uma Timed Petri Net estilo Ramchandani temos que uma transição habilitada (a proposta era basicamente para transições) que não fosse instantânea teria as marcas das pré-condições recolhidas à transição e esta ficaria habilitada e contando o tempo de simulação até que pudesse ser disparada.



Mais recentemente estas redes são associadas a um tempo (real) especificado e são simplesmente chamadas de Deterministic Timed Petri Nets

Chapter 16

Timed Petri Nets

José Reinaldo Silva and Pedro M. G. del Foyo

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/50117

1. Introduction

In the early 60's a young researcher in Darmstad: looked for a good representation for communicating systems processes that were mathematically sound and had, at the same time, a visual intuitive flavor. This event marked the beginning of a schematic approach that become very important to the modeling of distributed systems in several and distinct areas of knowledge, from Engineering to biologic systems. Carl Adam Petri presented in 1962 his PHD which included the first definition of what is called today a Petri Net. Since its creation Petri Nets evolved "from a sound representation of discrete dynamic systems into a general schemata, capable to represent knowledge about processes and (discrete and distributed) systems according to their internal relations and not to their work domain. Among other advantages, that feature opens the possibility to reuse some experiences acquired in the design of known and well tested systems while treating new challenges.

In the conventional approach, the key issue for modeling is the partial ordering among constituent events and the properties that arise from the arrangement of state and transitions once some basic interpretation rules are preserved. Such representation can respond from several systems of practical use where the foundation for analysis is based in reachability and other property analysis. However, there are some cases where such approach is not enough to represent processes completely for instance, when the assumption that all transitions can fire instantaneously is no longer a good approximation. In such cases a time delay can be associates to firing transitions. This is absolutely equivalent (in a broader sense) to say that firing pre-conditions must hold for a time delay before the firing is completed. The first approach is called T-time Petri Net and the second P-time Petri Nets.

Thus, what we have in conclusion is that even in a hypothesis that we should consider only firing pre-conditions[1] [31][19] a time delay is associated with a transition location and consequently to its firing. Several applications in manufacturing, business, workflow and

[1] In many text books and review articles the enabling condition is presented using only firing pre-conditions as a requirement. This can be justified since the use of this weak firing condition is sufficient if a complete net, that is, that includes its dual part, is used.

Definition 6. [Timed Petri Net] A timed Petri net is a six-tuple

$$N = (P, T, A, w, M_0, f)$$

where

(P, T, A, w, M_0) is a marked Petri net

$f : T \to \mathbb{R}^+$ is a firing time function that assigns a positive real number to each transition on the net

Therefore, the firing rule has to be modified in order to consider time elapses in the transition firing. If an enabled transition $t_j \in enb(M)$ then it will fire after $f(t_j)$ times units since it became enabled. The system state is not only determined by the net marking but also by a timer attached to every enabled transition in the net.

Silva, J. R. and del Foyo, P.M.G.

Michael Westergaard

Intergalactic Michaelpedia of the World

Musings about Britney, computer science, politics, tinkering, and whatever regales me...

Tutorials | Recipes | Search...

By Michael | May 24, 2015 — 0 Comments

Basic Design Principle of CPNaaS

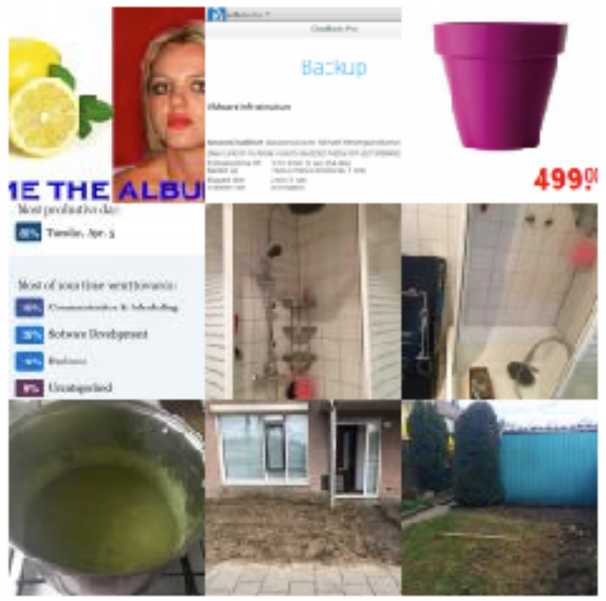
This post has 858 words. Reading it will take approximately 4 minutes.

Rating: 0.0/5 (0 votes cast)

CPNaaS is my new under-development API for colored Petri nets-as-a-service, i.e., a web-API for tools.

At the core of the API is a RESTful web-service. This just means that each resource has a unique URL, which is manipulated as if it were a web-page. Web-pages are normally just obtained for reading (GET), but may also be sent information to generate extra information (POST). Tools for editing web-pages can additionally upload new pages (PUT) or delete out-dated ones (DELETE).

PICTURES



ON MY MIND...

This post has 65 words. Reading it will take less than one minute. — 15 May 2016

Rating: 0.0/5 (0 votes cast)

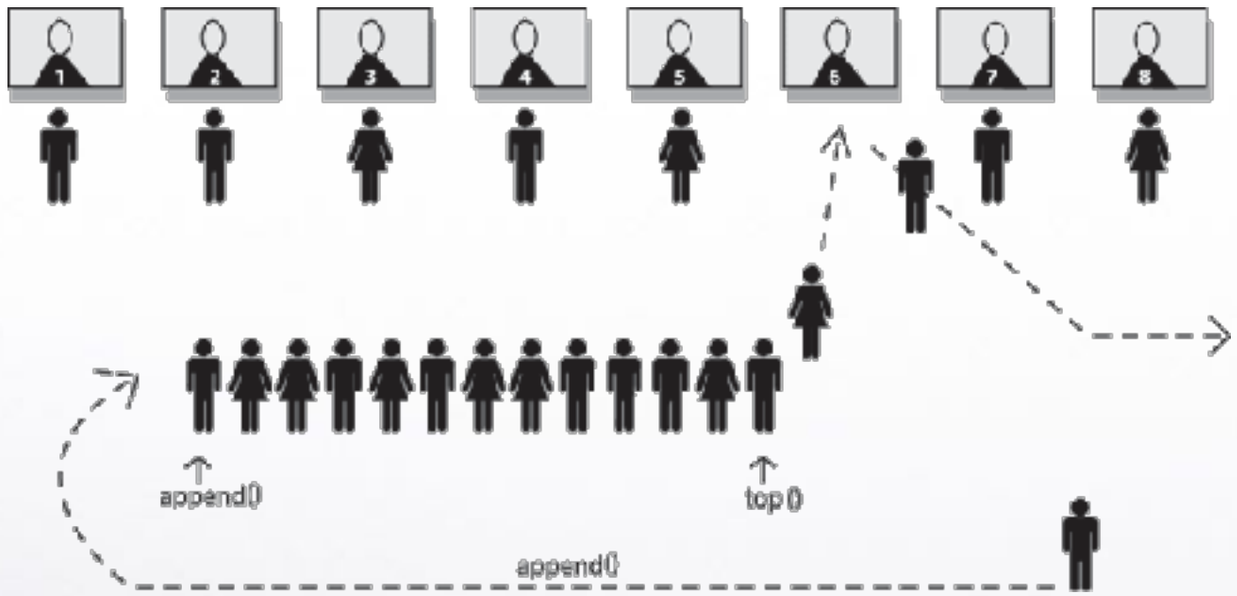
What is a model?

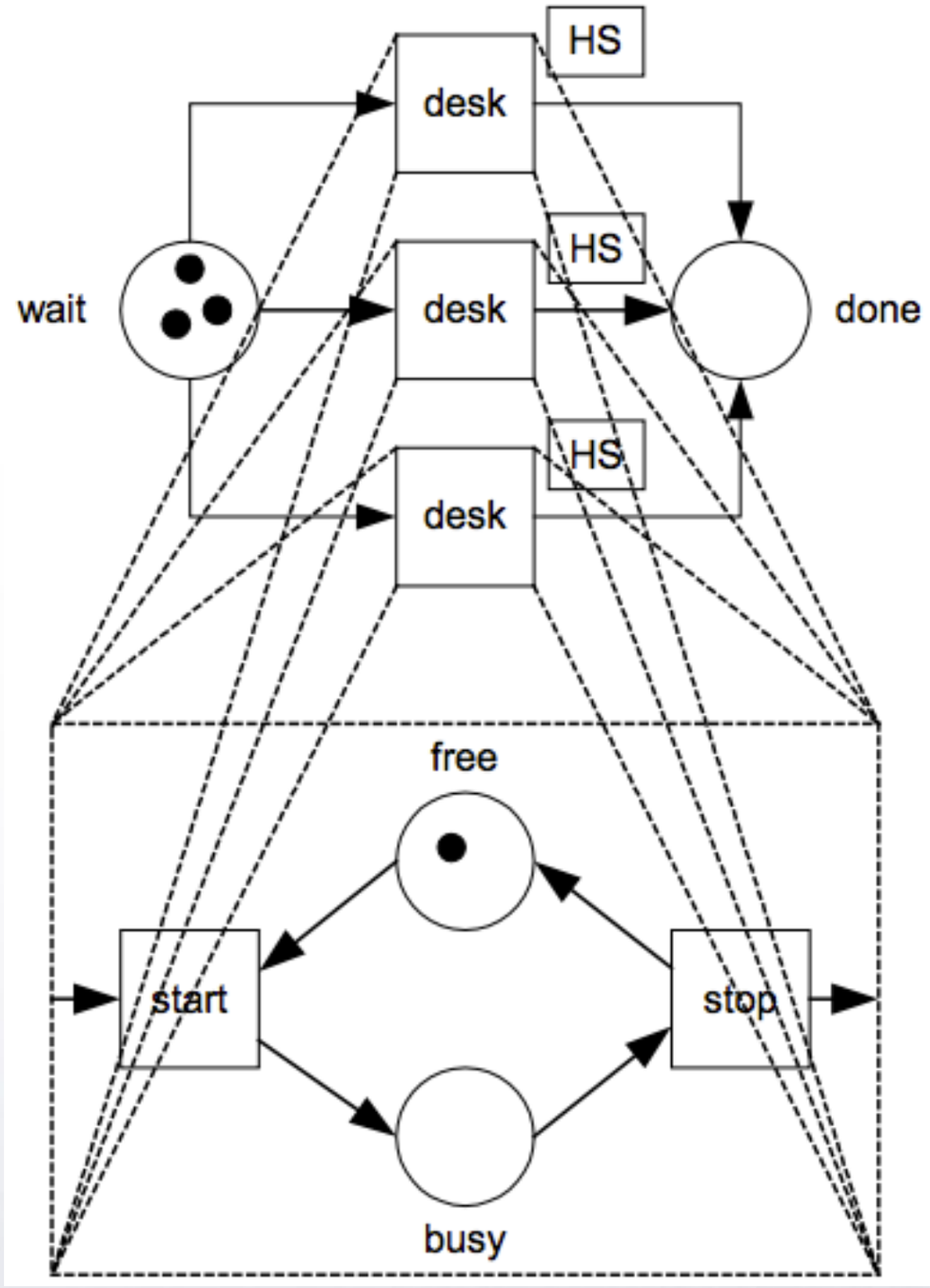
model and modelling
in painting, the *use of light and shade to simulate volume in the representation of solids*. In sculpture the terms denote a technique involving the use of a pliable material such as clay or wax. As opposed to carving, modelling permits addition as well as subtraction of material and lends itself to freer handling and change of intention. The technique is exemplified also by those works in cast metal and plaster that are made from the mold of a clay original. The mold is made by the process of cire perdue. The noun model is used to describe such an original and also *any three-dimensional scale model for a larger or more elaborate project in architecture, landscaping, or industry*. It also denotes a person or object used as an aid to representation in painting.

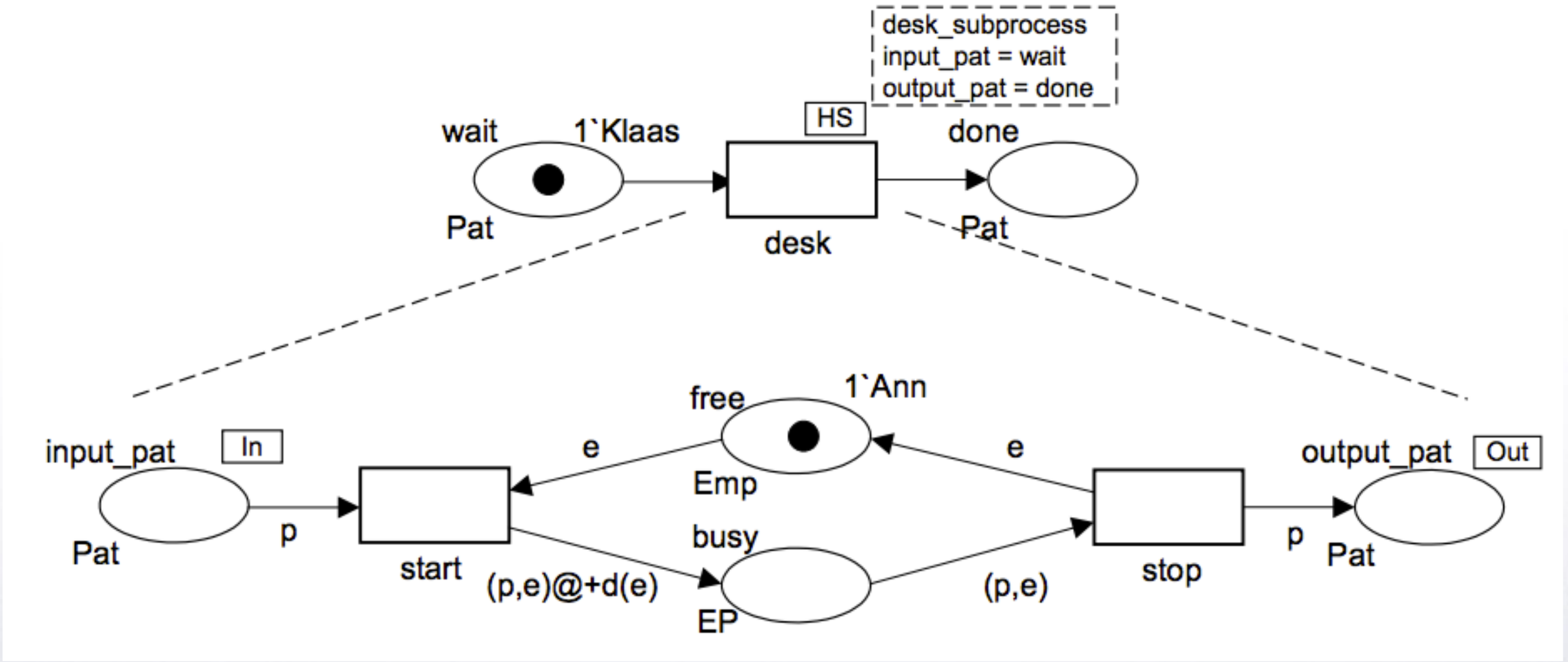
The Columbia Encyclopaedia, Sixth Edition. 2001.

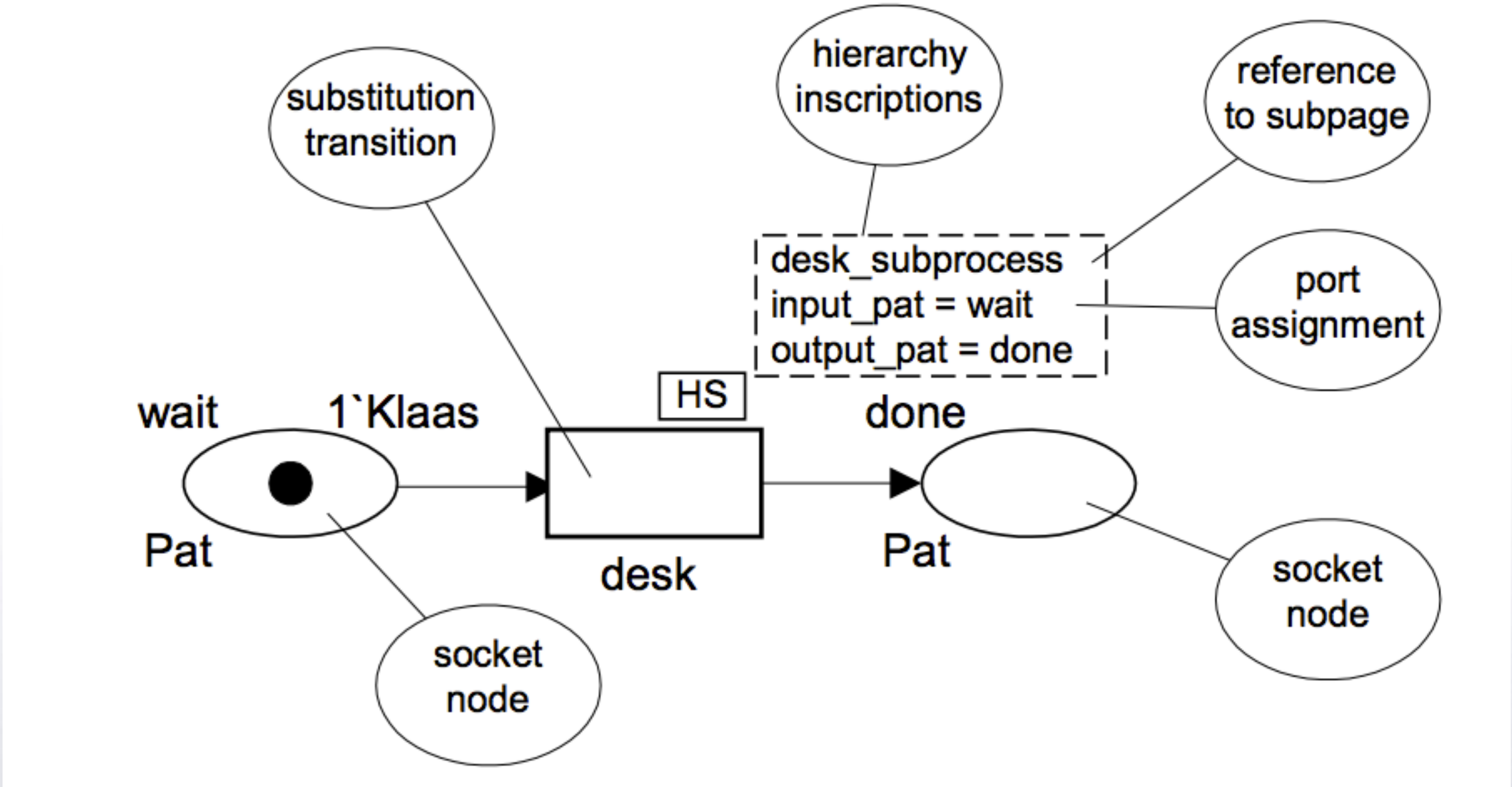
Abstract representation, scale model of future design

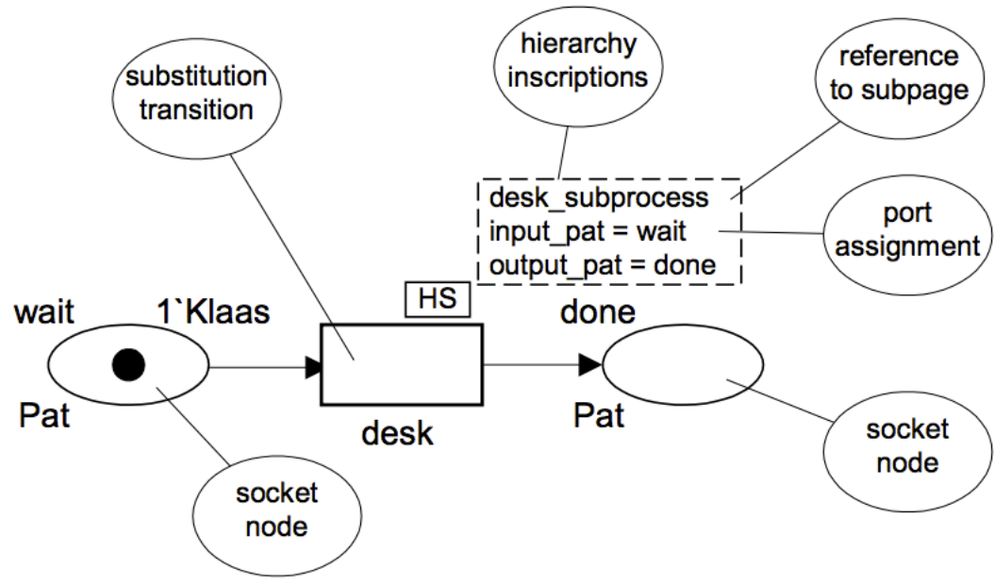
August 27, 2002 Søren Christensen, MOCA'02 7

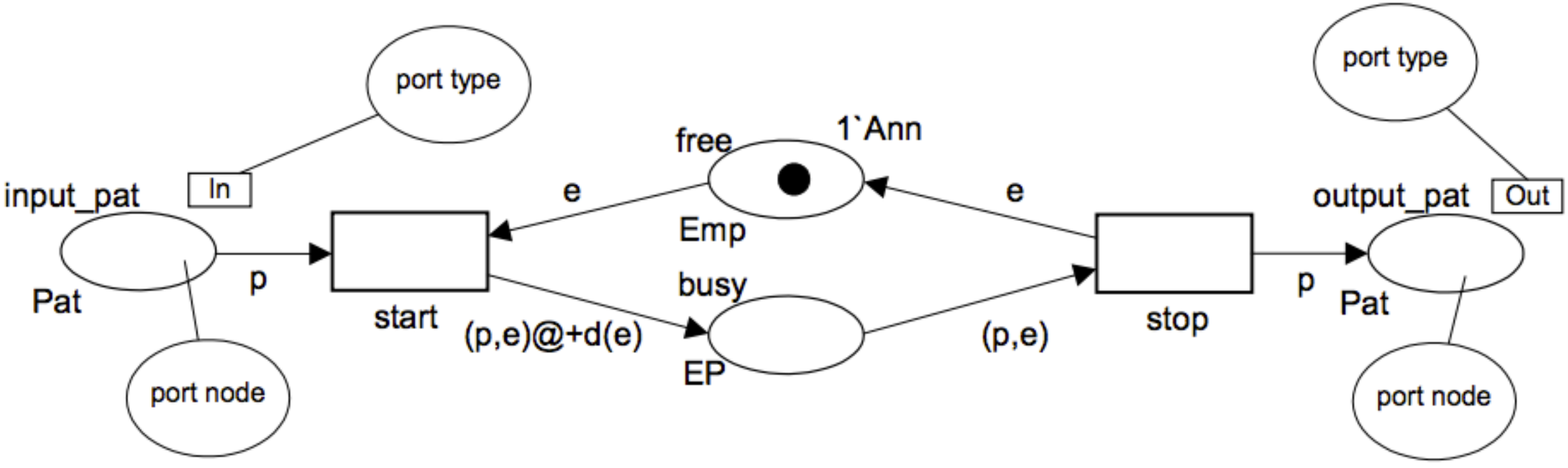

append()
top()
append()


HS
desk
HS
wait
desk
done
HS
desk
free
start
stop
busy

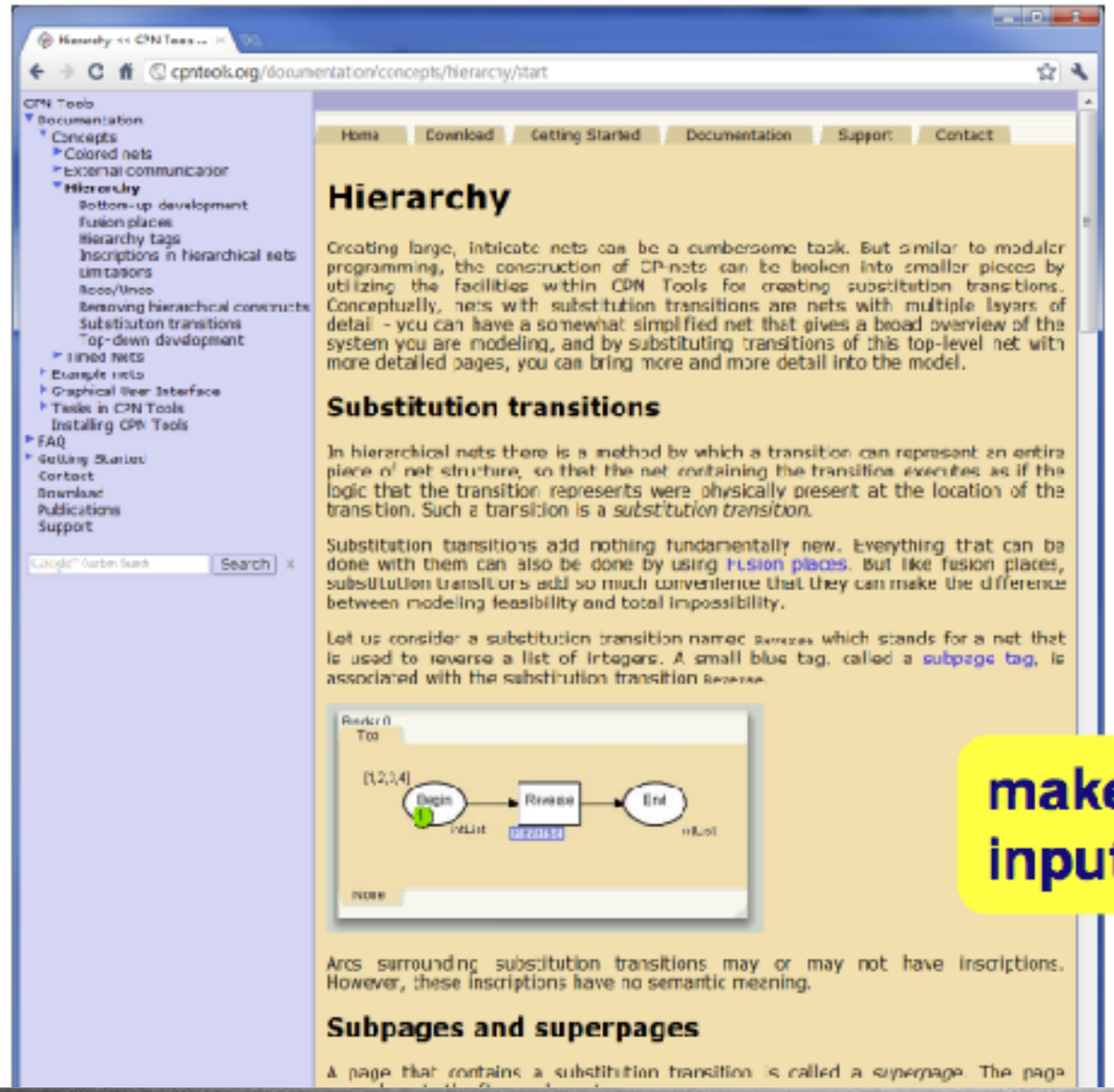

desk_subprocess
input_pat = wait
output_pat = done
HS
wait
1`Klaas
done
Pat
desk
Pat
1`Ann
free
input_pat
In
e
e
output_pat
Out
Emp
p
busy
p
Pat
Pat
start
(p,e)@+d(e)
(p,e)
stop
EP

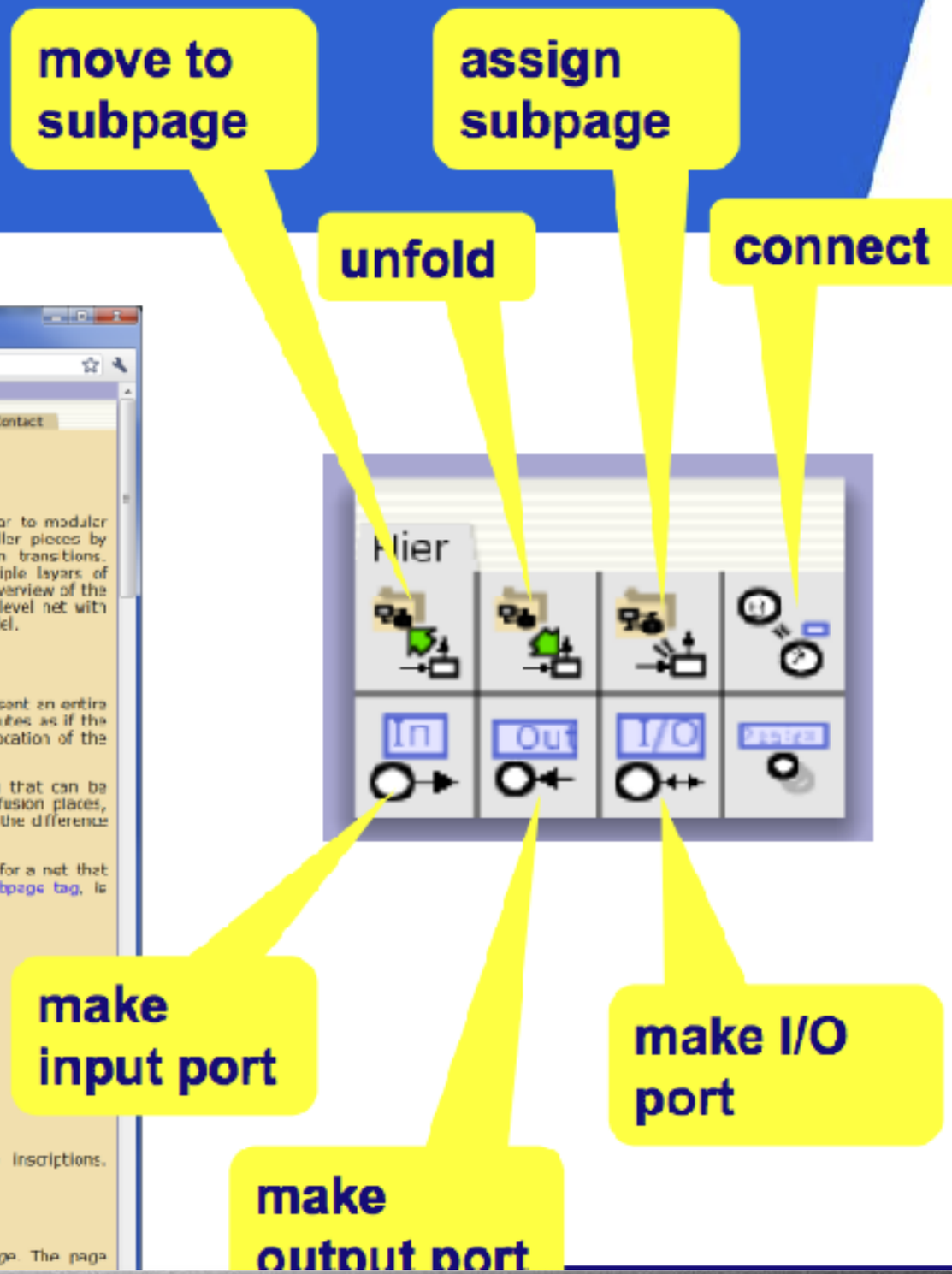

substitution transition
hierarchy inscriptions
reference to subpage
desk_subprocess
input_pat = wait
output_pat = done
port assignment
HS
wait
1`Klaas
done
Pat
desk
Pat
socket node
socket node

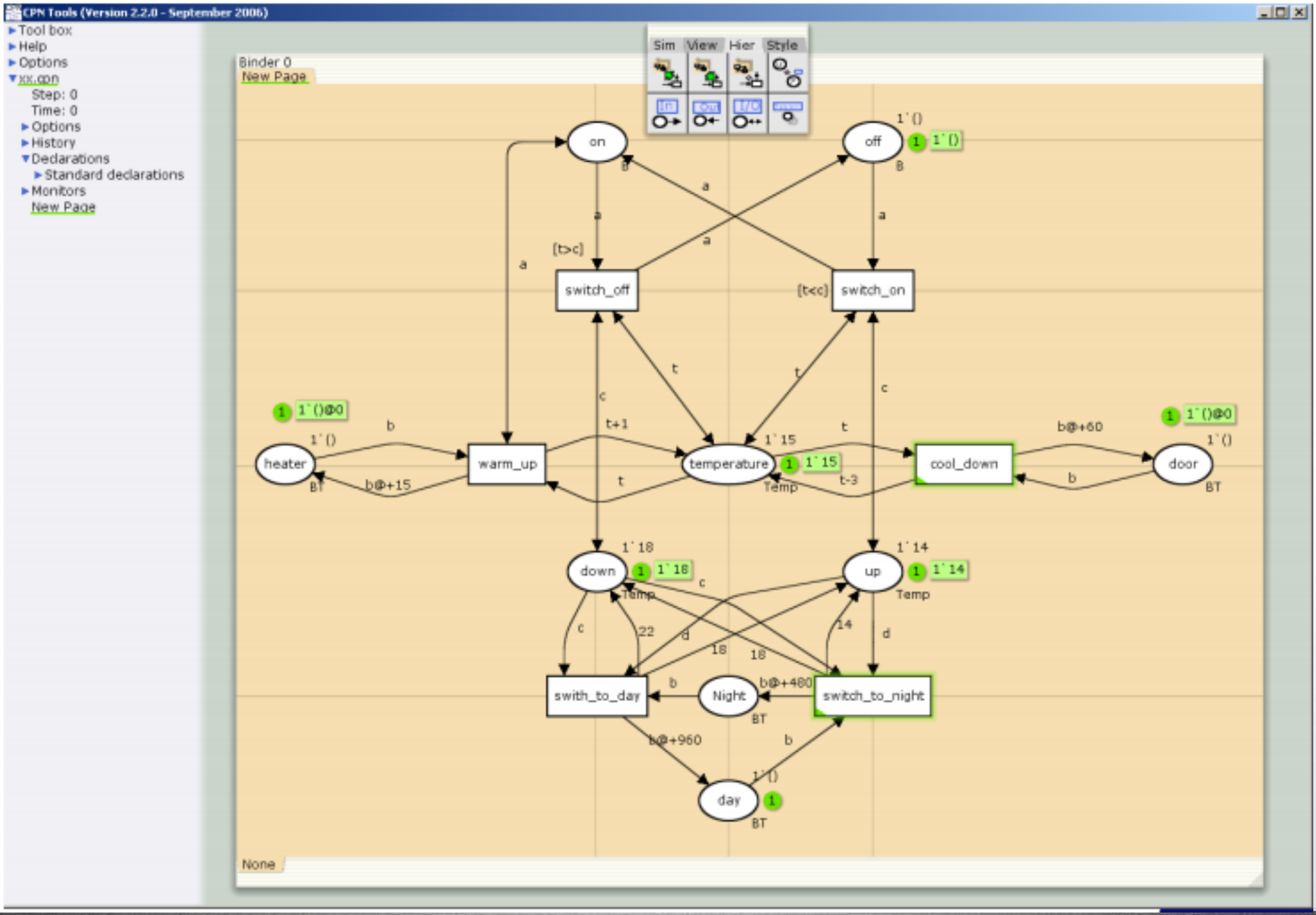

substitution transition
hierarchy inscriptions
reference to subpage
desk_subprocess
input_pat = wait
output_pat = done
port assignment
HS
wait
1`Klaas
done
Pat
desk
Pat
socket node
socket node

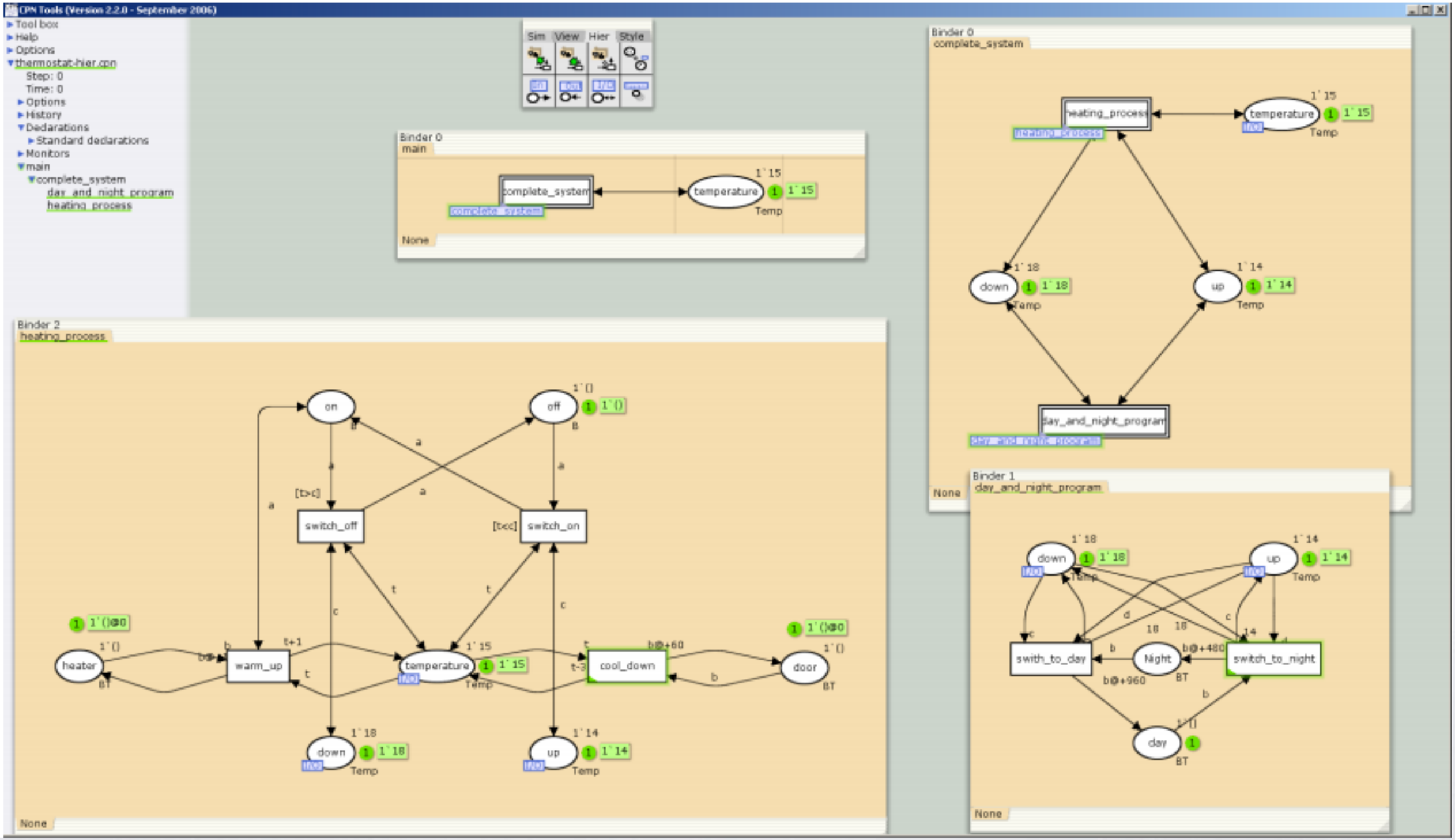

port type
free
1`Ann
input_pat
In
e
e
output_pat
Out
Emp
Pat
p
busy
p
Pat
start
(p,e)@+d(e)
(p,e)
stop
EP
port node
port node

CPN Tools


cpntools.org/documentation/concepts/hierarchy/start
CPN Tools
Documentation
Concepts
Colored nets
External communicator
Hierarchy
Bottom-up development
Fusion places
Hierarchy tags
Inscriptions in hierarchical nets
Limitations
Redo/Undo
Removing hierarchical constructs
Substituton transitions
Top-down development
Timed Nets
Example nets
Graphical User Interface
Tasks in CPN Tools
Installing CPN Tools
FAQ
Getting Started
Contact
Download
Publications
Support
Search
Home
Download
Getting Started
Documentation
Support
Contact
Hierarchy
Creating large, intricate nets can be a cumbersome task. But similar to modular programming, the construction of CP-nets can be broken into smaller pieces by utilizing the facilities within CPN Tools for creating substitution transitions. Conceptually, nets with substitution transitions are nets with multiple layers of detail - you can have a somewhat simplified net that gives a broad overview of the system you are modeling, and by substituting transitions of this top-level net with more detailed pages, you can bring more and more detail into the model.
Substitution transitions
In hierarchical nets there is a method by which a transition can represent an entire piece of net structure, so that the net containing the transition executes as if the logic that the transition represents were physically present at the location of the transition. Such a transition is a substitution transition.
Substitution transitions add nothing fundamentally new. Everything that can be done with them can also be done by using fusion places. But like fusion places, substitution transitions add so much convenience that they can make the difference between modeling feasibility and total impossibility.
Let us consider a substitution transition named Reverse which stands for a net that is used to reverse a list of integers. A small blue tag, called a subpage tag, is associated with the substitution transition Reverse.
Reverse'0
Top
[1,2,3,4]
Begin
Reverse
End
n'List
n'List
Arcs surrounding substitution transitions may or may not have inscriptions. However, these inscriptions have no semantic meaning.
Subpages and superpages
A page that contains a substitution transition is called a superpage. The page

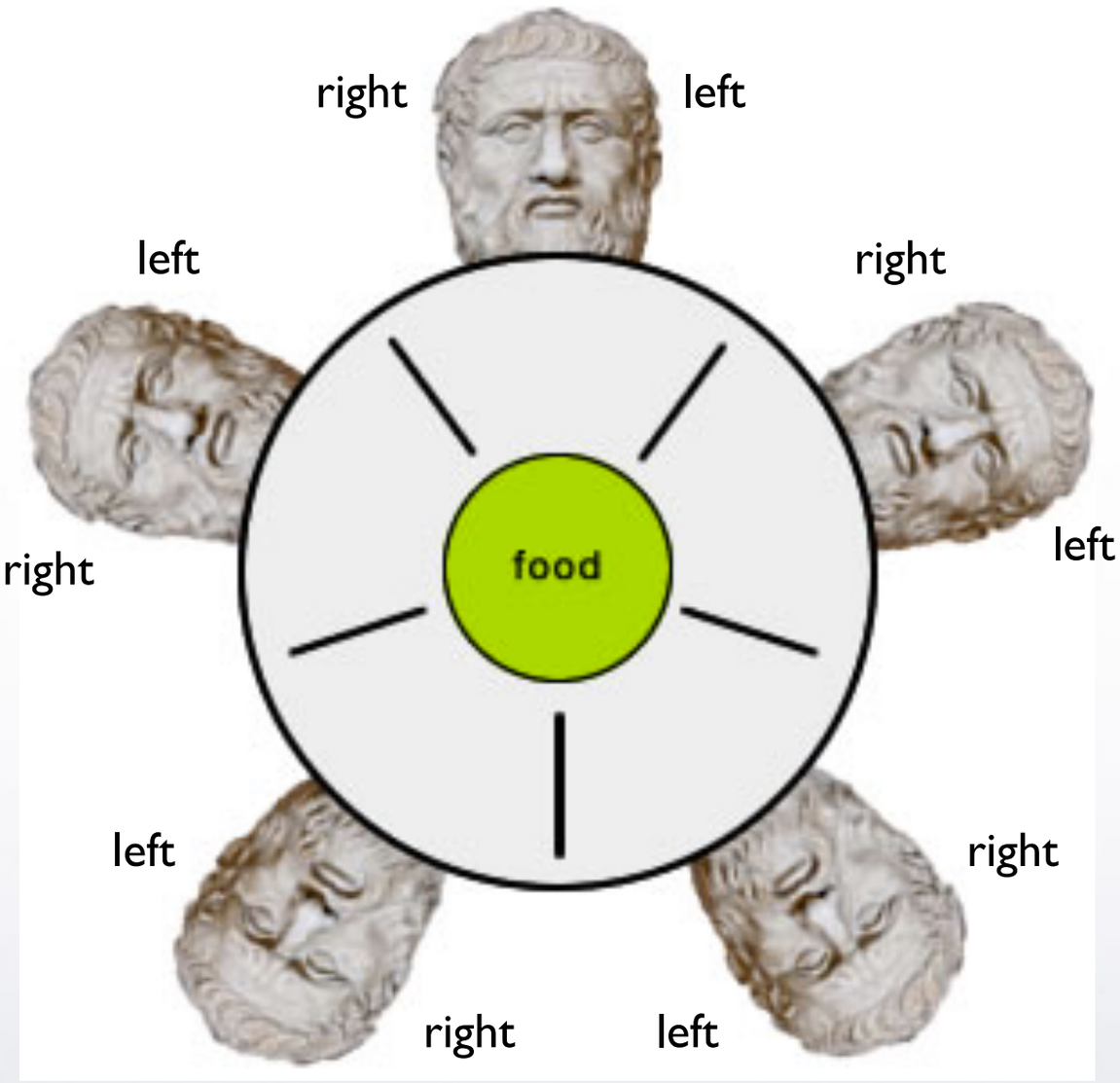

move to subpage
assign subpage
unfold
connect
Hier
In
Out
I/O
make input port
make output port
make I/O port


CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
xx.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
Monitors
New Page
Sim
View
Hier
Style
Binder 0
New Page
on
off
1`()
switch_off
switch_on
[t>c]
[t<c]
heater
warm_up
temperature
cool_down
door
1`()@0
1`15
b@+15
b@+60
down
up
1`18
1`14
Temp
switch_to_day
Night
switch_to_night
b@+480
b@+960
day
None

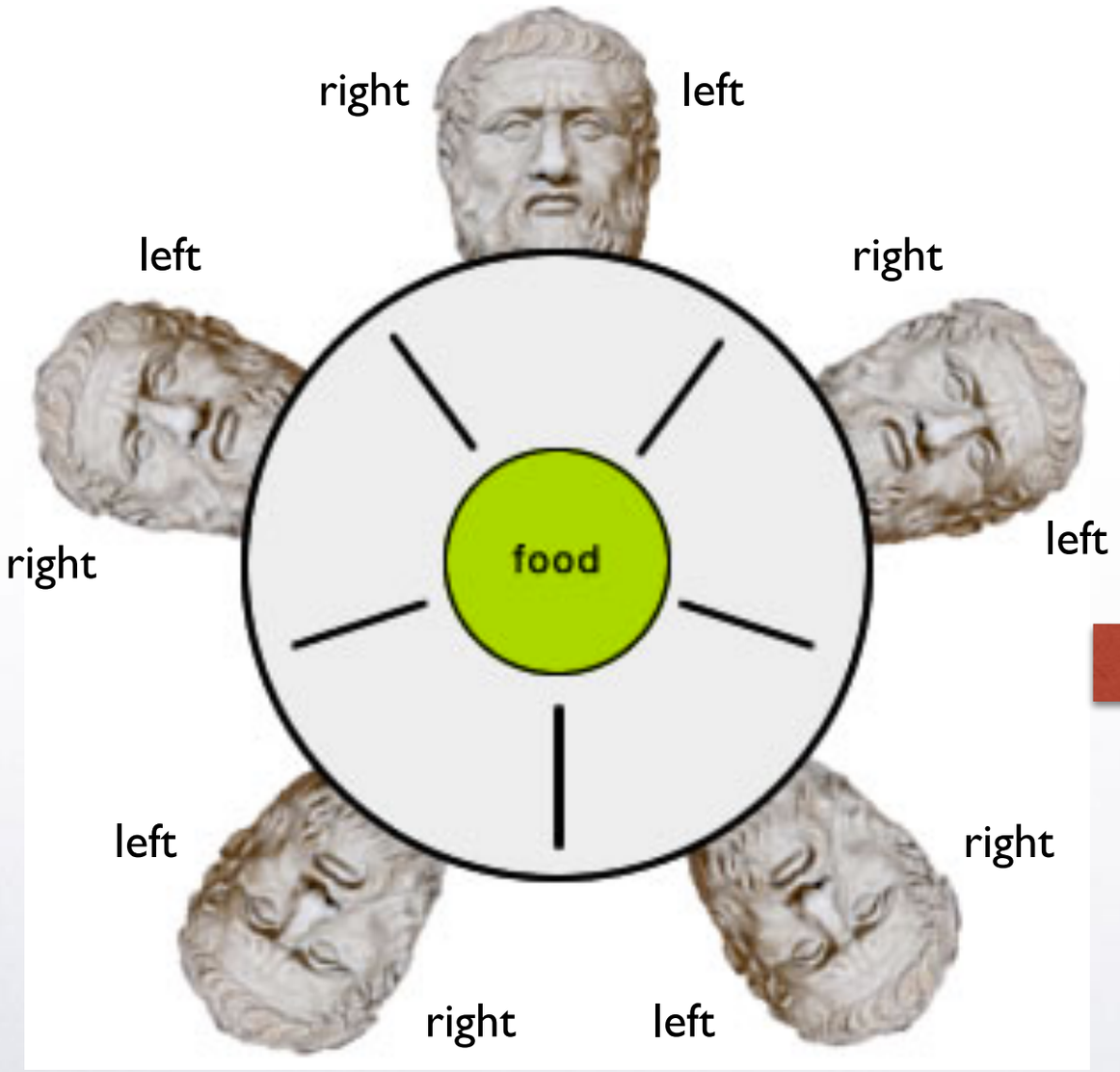

CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
thermostat-hier.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
Monitors
main
complete_system
day_and_night_program
heating_process
Sim View Hier Style
Binder 0
main
complete_system
temperature
1`15
Temp
None
Binder 0
complete_system
heating_process
day_and_night_program
down
up
Temp
Binder 1
day_and_night_program
swith_to_day
Night
switch_to_night
day
Binder 2
heating_process
on
off
switch_off
switch_on
heater
warm_up
cool_down
door
[t>c]
[t<c]
None

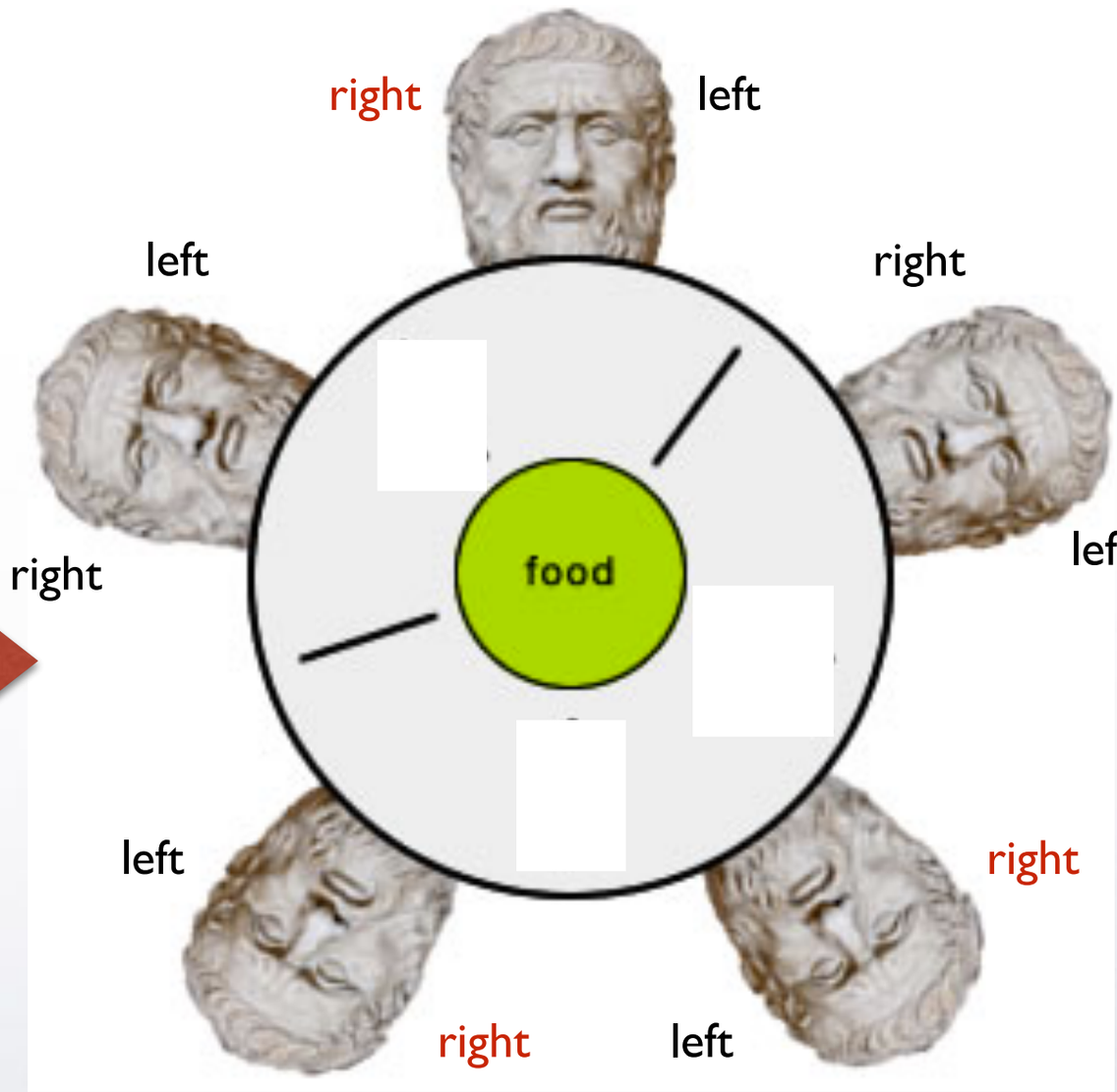
- Make a hierarchical CPN model of five Chinese philosophers alternating between states thinking and eating. To eat two chopsticks are needed. In total there are five chopsticks. The philosophers are sitting in a circle, and need to complete for chopsticks with their direct neighbors (left and right). Assume that both chopsticks need to be taken at the same time.

  Model this using a hierarchical CPN model. Make sure to model the behavior of a philosopher only once and just use the color set BlackToken of type unit.
- Change the model such that philosophers can take one chopstick at a time but avoid deadlocks and a fixed ordering of philosophers.
- Flatten the hierarchical CPN model.

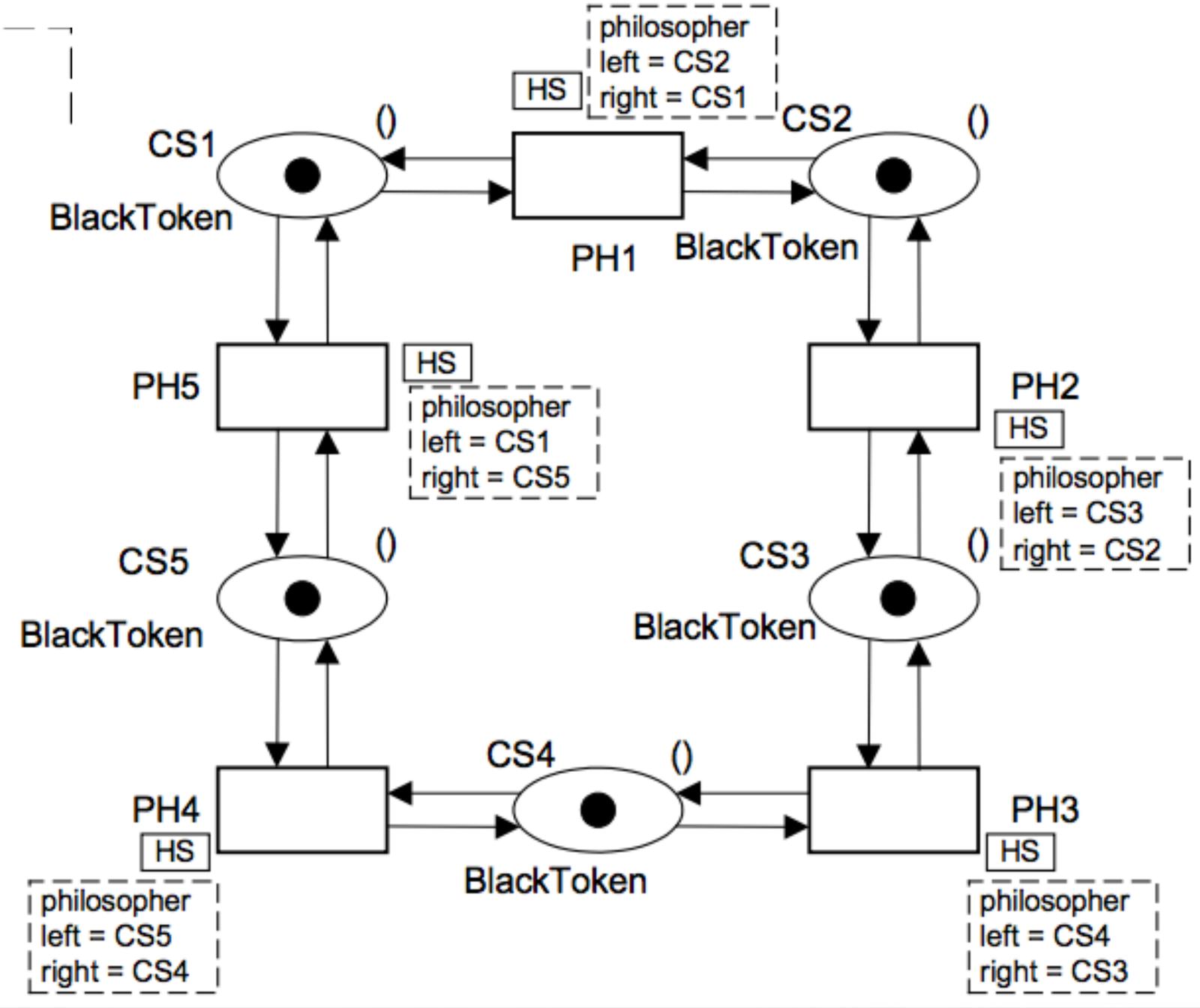

right
left
left
right
right
left
food
left
right
right
left

deadlock


right
left
left
right
food
right
left
left
right
right
left


right
left
left
right
food
right
left
right
right
left

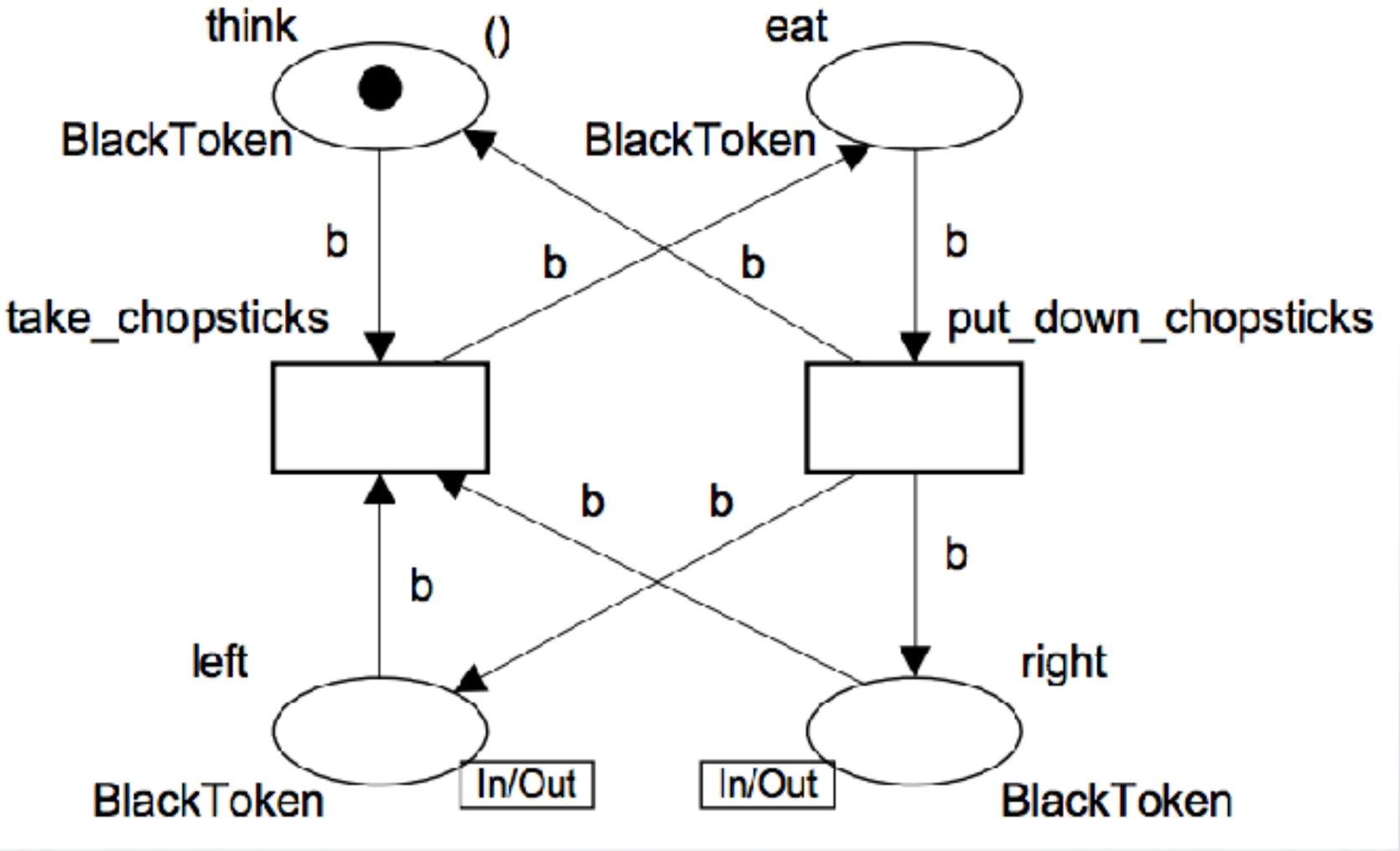
```
color BlackToken = unit;
var b:BackToken
```

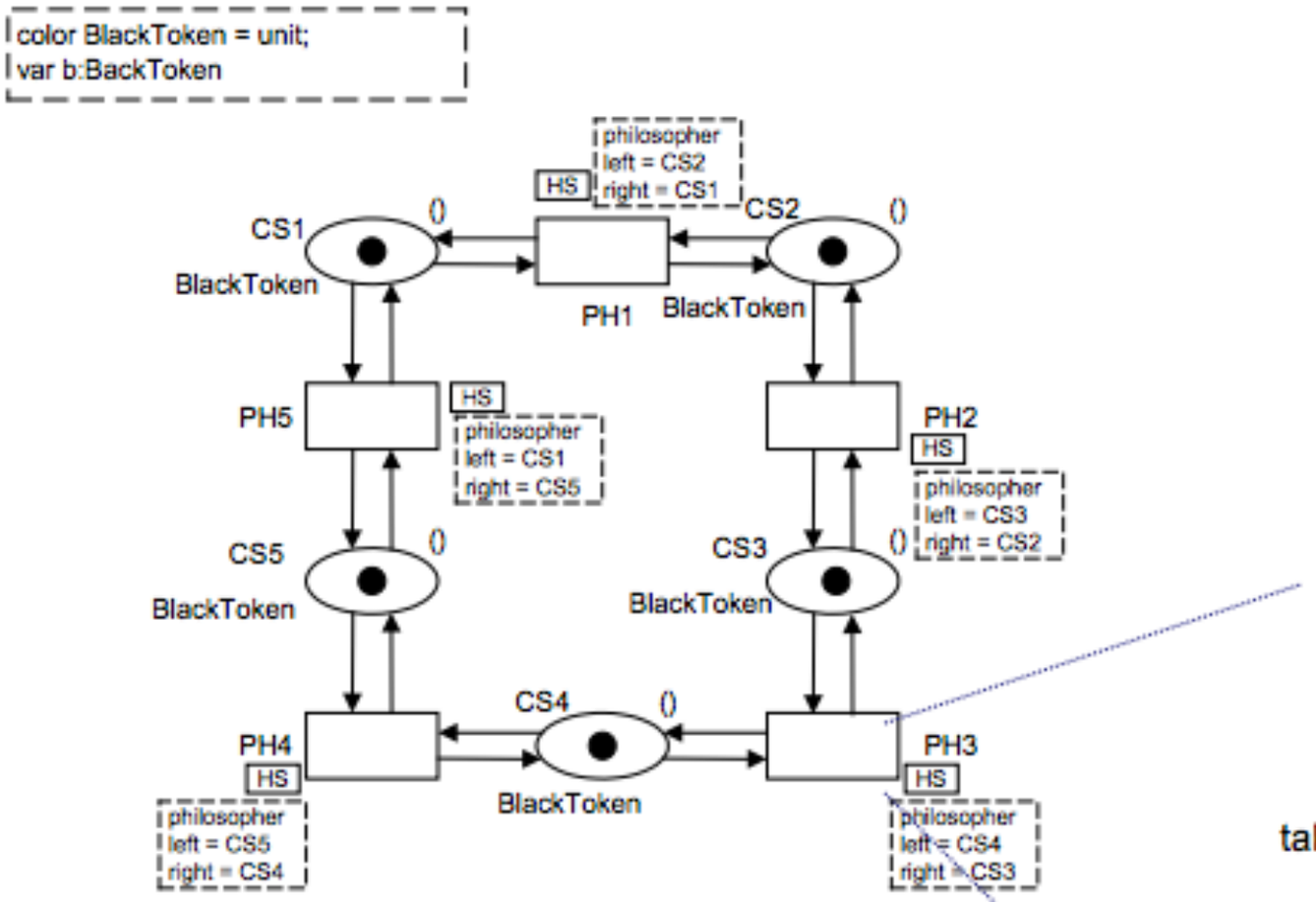


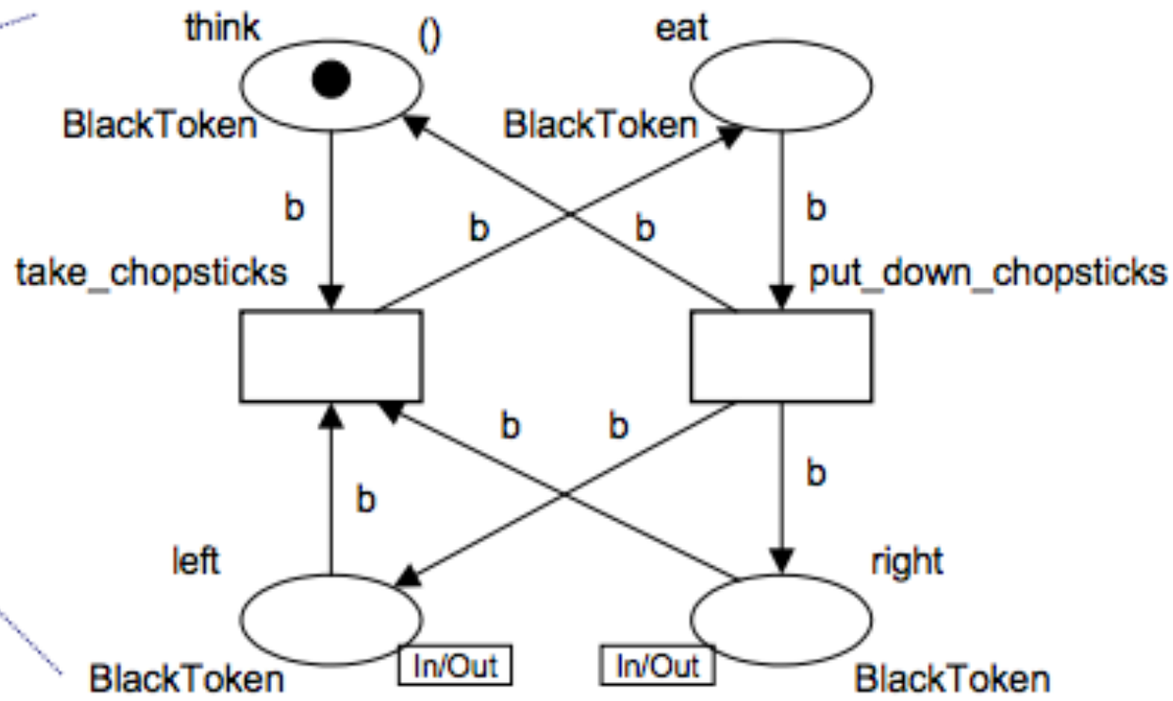
Esta modelagem tem dois aspectos interessantes:

a) não tem muito sentido pegar os sticks e devolver logo em seguida, exceto se retiver os sticks por algum tempo;
b) o estado “comendo" não aparece explicitamente, o que indica que deve ocorrer para cada filósofo em um nível hierárquico mais baixo, dentro das transições.

Portanto cada filósofo usa os chopsticks para passar para o estado comendo, e, expandindo cada transição associada a ele temos:




color BlackToken = unit;
var b:BackToken
philosopher
left = CS2
right = CS1
HS
CS1
()
BlackToken
PH1
CS2
()
BlackToken
PH5
HS
philosopher
left = CS1
right = CS5
PH2
HS
philosopher
left = CS3
right = CS2
CS5
()
BlackToken
CS3
()
BlackToken
CS4
()
BlackToken
PH4
HS
philosopher
left = CS5
right = CS4
PH3
HS
philosopher
left = CS4
right = CS3


think
()
BlackToken
eat
BlackToken
b
take_chopsticks
put_down_chopsticks
left
right
In/Out
In/Out
BlackToken
BlackToken

CPN Tools (Version 3.0.2, January 2011)

- Tool box
- Help
- Options
- DiningPhilosophersClassic.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - colset UNIT = unit;
  - Monitors
  - main
    - philosopher (5)
    - philosopher (1)
    - philosopher (2)
    - philosopher (3)
    - philosopher (4)

Binder 0

main | philosopher(1) | philosopher(2) | philosopher(3) | philosopher(4) | philosopher(5)

Hier

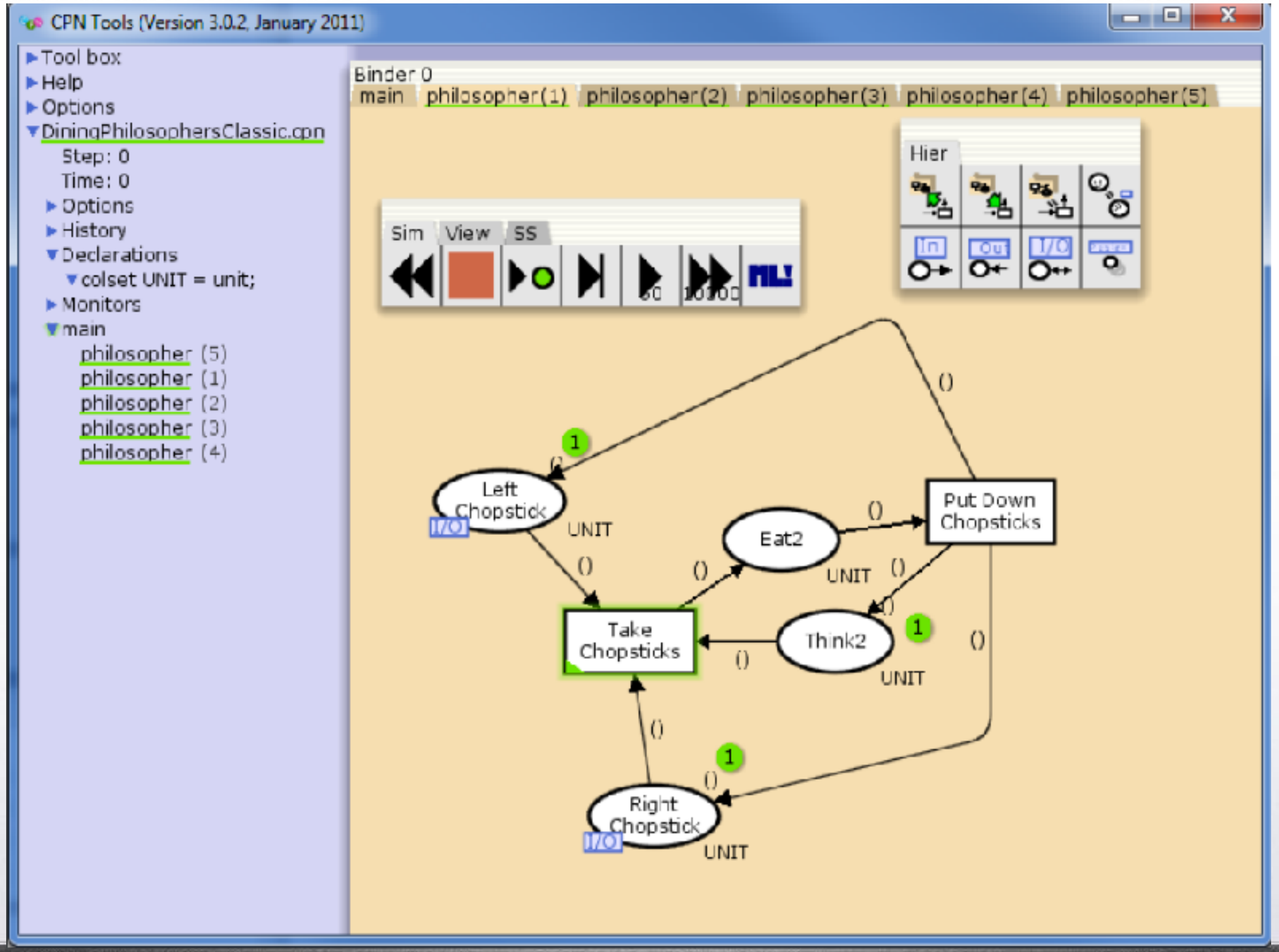
In | Out | I/O

Sim | View | SS

philosopher5 (philosopher)
philosopher4 (philosopher)
philospher1 (philosopher)
philosopher3 (philosopher)
philosopher2 (philosopher)

Free Chopstick5 UNIT
Free Chopstick1 UNIT
Free Chopstick4 UNIT
Free Chopstick2 UNIT
Free Chopstick3 UNIT

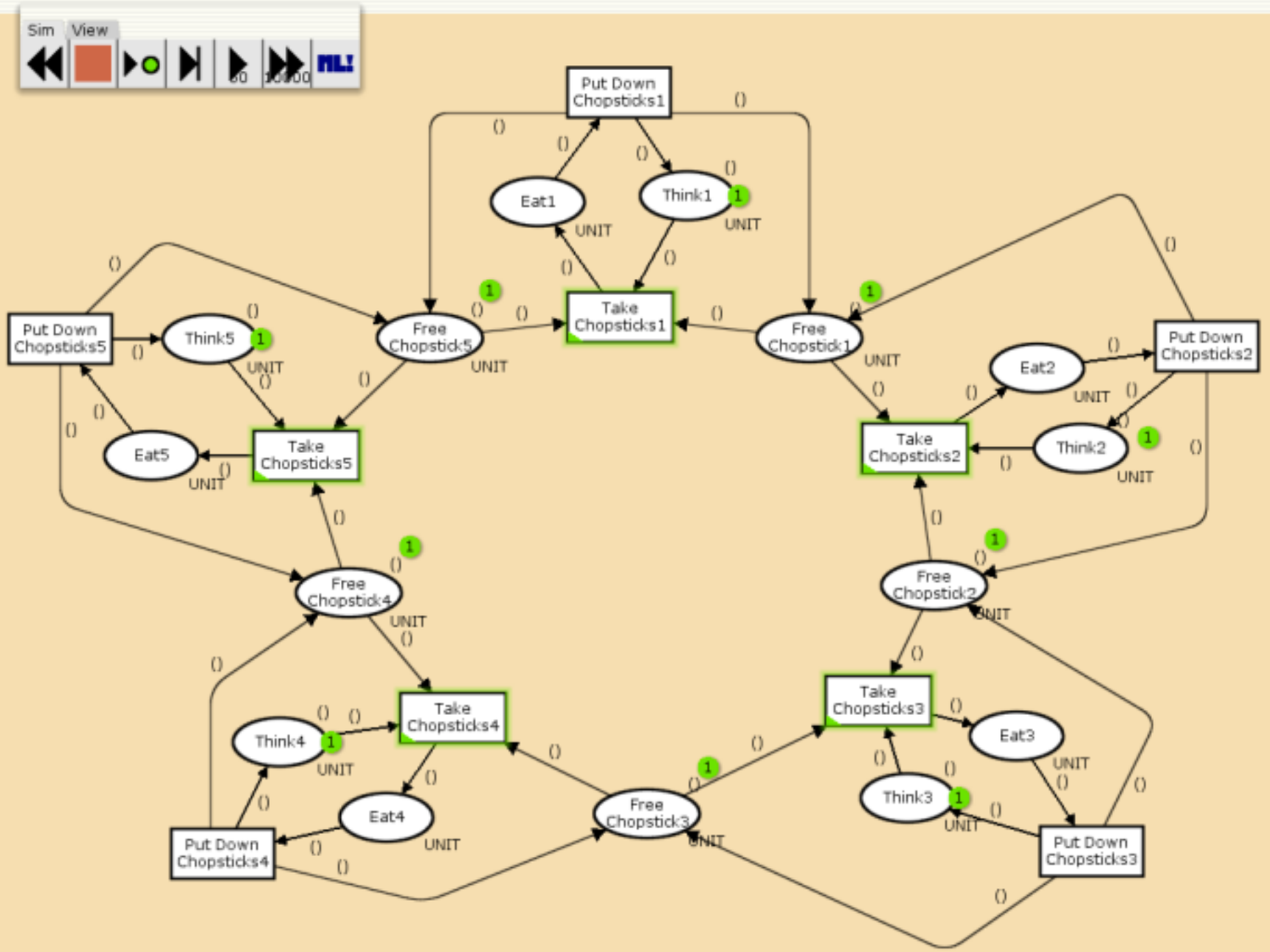
1 ()


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersClassic.cpn
Step: 0
Time: 0
Options
History
Declarations
colset UNIT = unit;
Monitors
main
philosopher (5)
philosopher (1)
philosopher (2)
philosopher (3)
philosopher (4)
Binder 0
main
philosopher(1)
philosopher(2)
philosopher(3)
philosopher(4)
philosopher(5)
Sim
View
SS
Hier
In
Out
I/O
Fusion
Left Chopstick
I/O
UNIT
Eat2
UNIT
Put Down Chopsticks
Take Chopsticks
Think2
UNIT
Right Chopstick
I/O
UNIT
()
1

CPN Tools (Version 3.0.2, January 2011)

- Tool box
- Help
- Options
- DiningPhilosophersClassic.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - colset UNIT = unit;
  - Monitors
  - Page

Binder 0
Page



Inserindo funções

Uma maneira de incrementar as inscrições é incluir elementos de “ordem" superior à que temos até agora (variáveis, expressões, elementos lógicos básicos, if-then-else, etc.).

Uma função é associada a um identificador e uma assinatura que especifica o tipo dos argumentos e o tipo do valor de retorno.

Introduziremos também o conceito de **lista**.

Uma **lista** é uma estrutura homogênea (todos os componentes são do mesmo tipo), composta de uma sucessão de elementos.

Exemplo: [a, b, c, d, e, f, g]
[1, 2, 4, 6, 7, 10]

Uma lista também pode ser definida de forma recursiva, como sendo composta de dois elementos básicos. Para isso definiremos em primeiro lugar a lista vazia [ ].

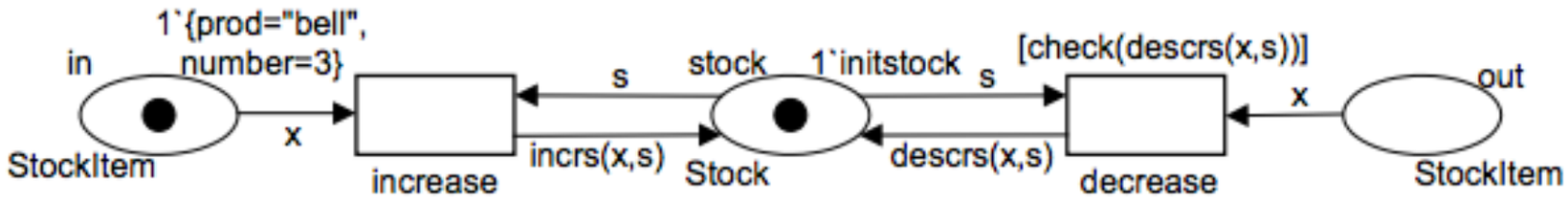
Uma lista não vazia é composta por dois elementos: head, que é o primeiro elemento da lista e tail que é uma lista (necessáriamente menor que a lista original), e se representa como [head | tail].
No exemplo dado anteriormente,

[a, b, c, d, e, f, g] → [a | [b, c, d, e, f, g]]

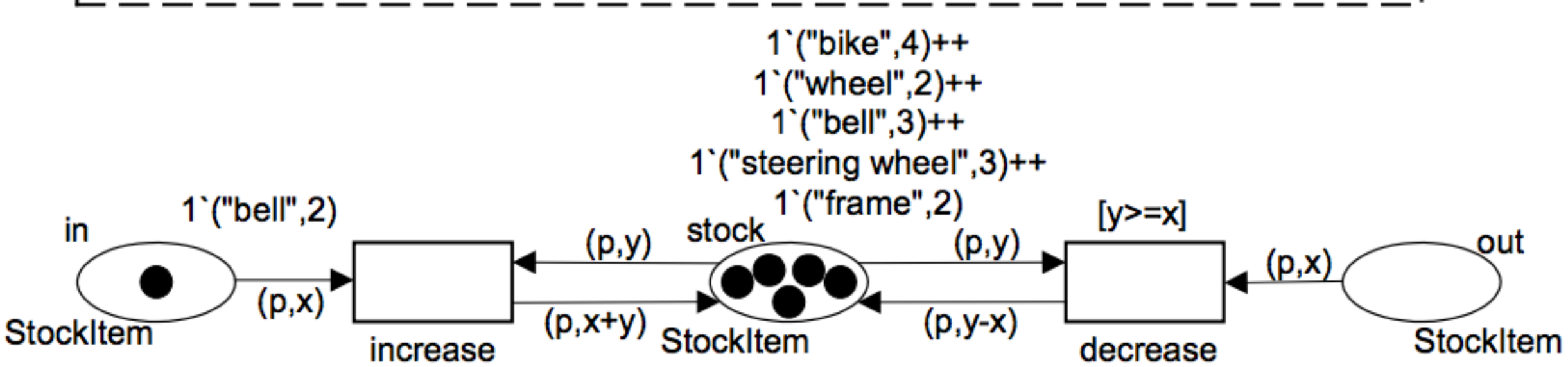
[1, 2, 4, 6, 7, 10] → [1 | [2, 4, 6, 7, 10]]

```
fun totalstock(s:Stock) =
  if s=[ ]
    then 0
    else (#number(hd(s)))+totalstock(tl(s));
```

```
color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
var x:StockItem;
var s:Stock;
fun incrs(x:StockItem,s:Stock) = if s=[] then [x] else (if (#prod(hd(s)))=(#prod(x))
   then {prod=(#prod(hd(s))),number=((#number(hd(s)))+(#number(x)))}::tl(s)
   else hd(s):: incrs(x,tl(s)));
fun decrs(x:StockItem,s:Stock)= incrs({prod=(#prod(x)),number=(~(#number(x)))},s);
fun check(s:Stock)= if s=[] then true else if (#number(hd(s)))<0 then false
  else check(tl(s));
val initstock = [{prod="bike", number=4},{prod="wheel", number=2},
  {prod="bell", number=3}, {prod="steering wheel", number=3},
  {prod="frame", number=2}];
```



```
color Product = string;
color Number = int;
color StockItem = product Product*Number;
var p:Product;
var x:Number;
var y:Number;
```



Note the simplicity/elegance of the arc inscriptions.

Recursion (4)

Function has two arguments

```
fun enoughstock(s:Stock,n:Number) =
  if s = [ ]
  then [ ]
  else if (#number(hd(s)))>= n then hd(s)::enoughstock(tl(s),n)
       else enoughstock(tl(s),n);
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…

n=400

Prod:Product	Number:number
"orange"	504
"pear"	423
…	…

CPN Tools
Access/CPN
Books
Documentation
Download
FAQ
Getting Started
First steps
Grade/CPN
Knowledge Base
Licensing
Support
Contact
Publications

Search

G+1 159

CPN To...
1.1K likes
Like Page

Be the first of your friends to like this

Home | Download | **Getting Started** | Documentation | Support | Contact

Getting Started

Modeling With Coloured Petri Nets

In all of the following and on all of these pages, we assume that you are familiar with coloured Petri nets and have at least some idea of constructing modules using the formalism. You can learn more about the modeling and the formalism in one or both of these books:

The User Interface of CPN Tools

First steps

Graphical User Interface

- **Index**
- **Marking menus**
- **Other tools**
- **Palette tools**

Fim