PMR 5237

Modelagem e Design de Sistemas Discretos em Redes de Petri

Aula 12: System Design e Redes de Petri

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.

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

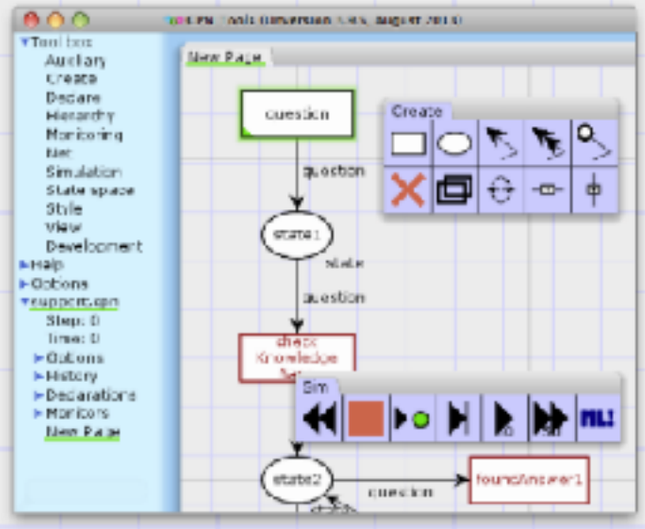
Like Page

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

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

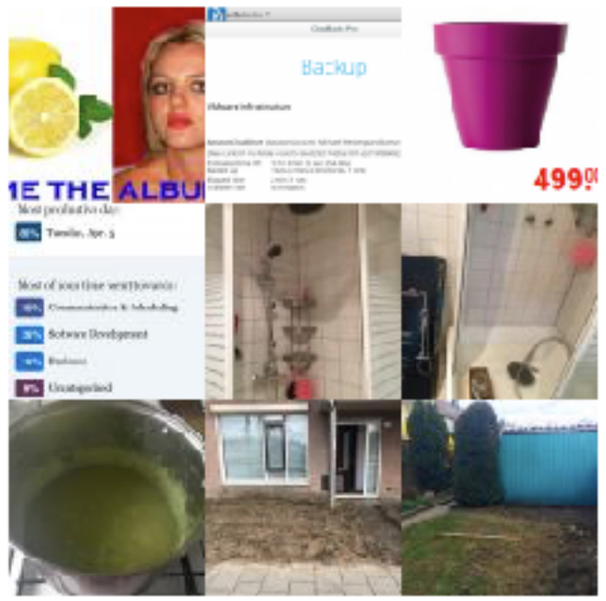
This post has 858 words. Reading it will take approximately 4 minutes.

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)

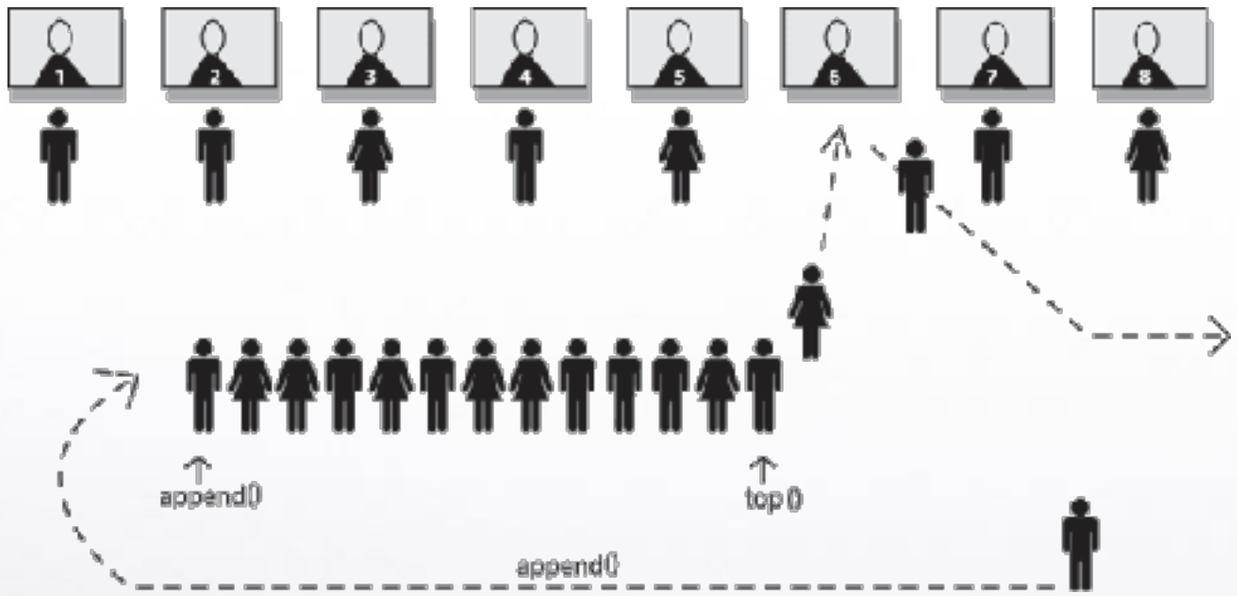
What is a model?

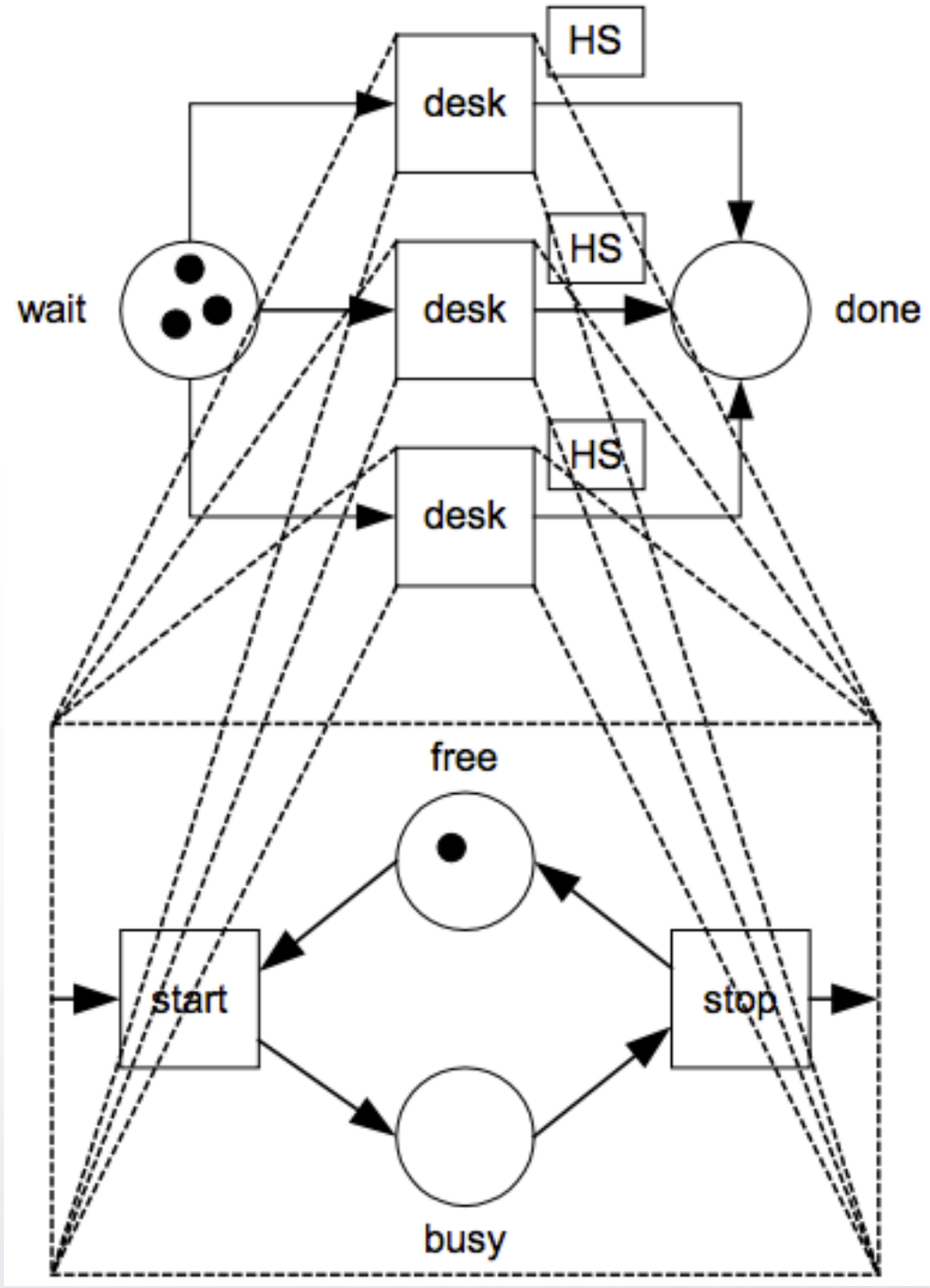
model and modelling
in painting, the *use of light and shade to simulate volume in the representation of solids*. In sculpture the terms denote a technique involving the use of a pliable material such as clay or wax. As opposed to carving, modelling permits addition as well as subtraction of material 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.

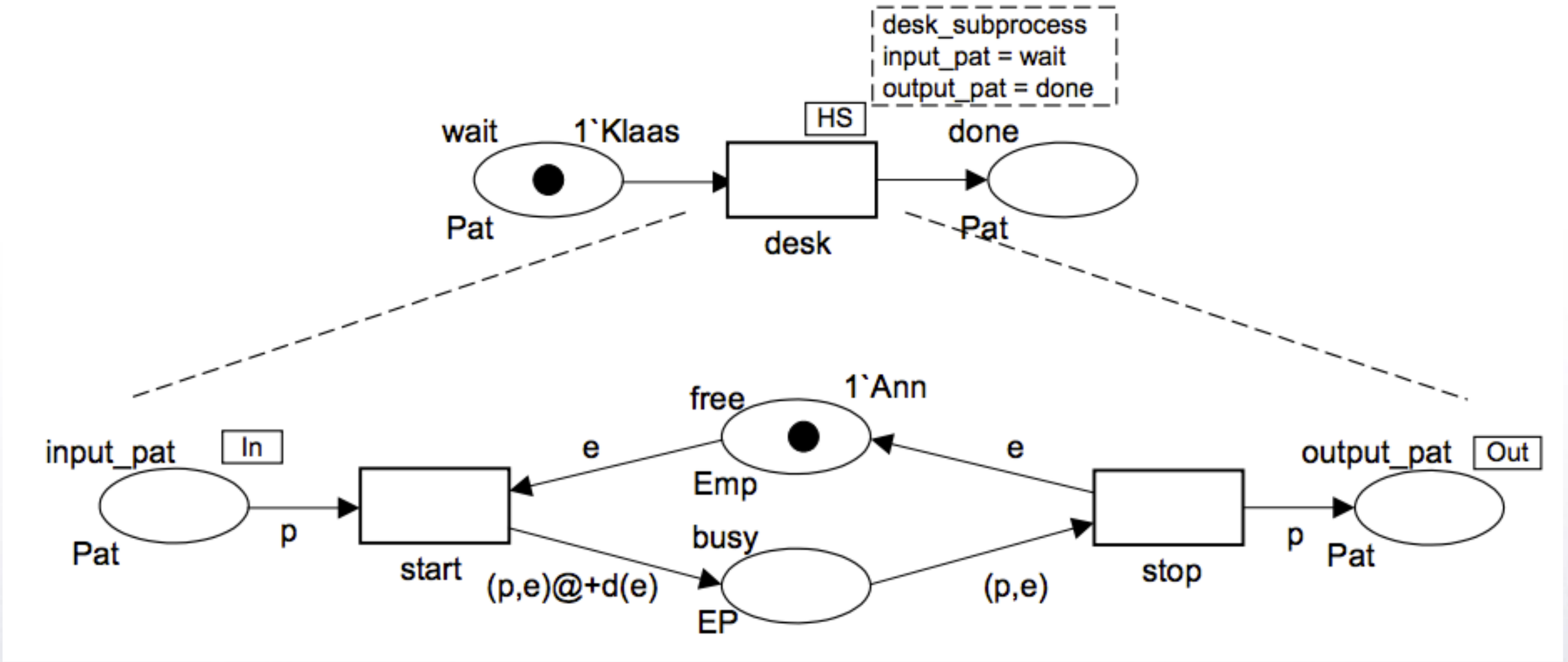
The Columbia Encyclopaedia, Sixth Edition. 2001.

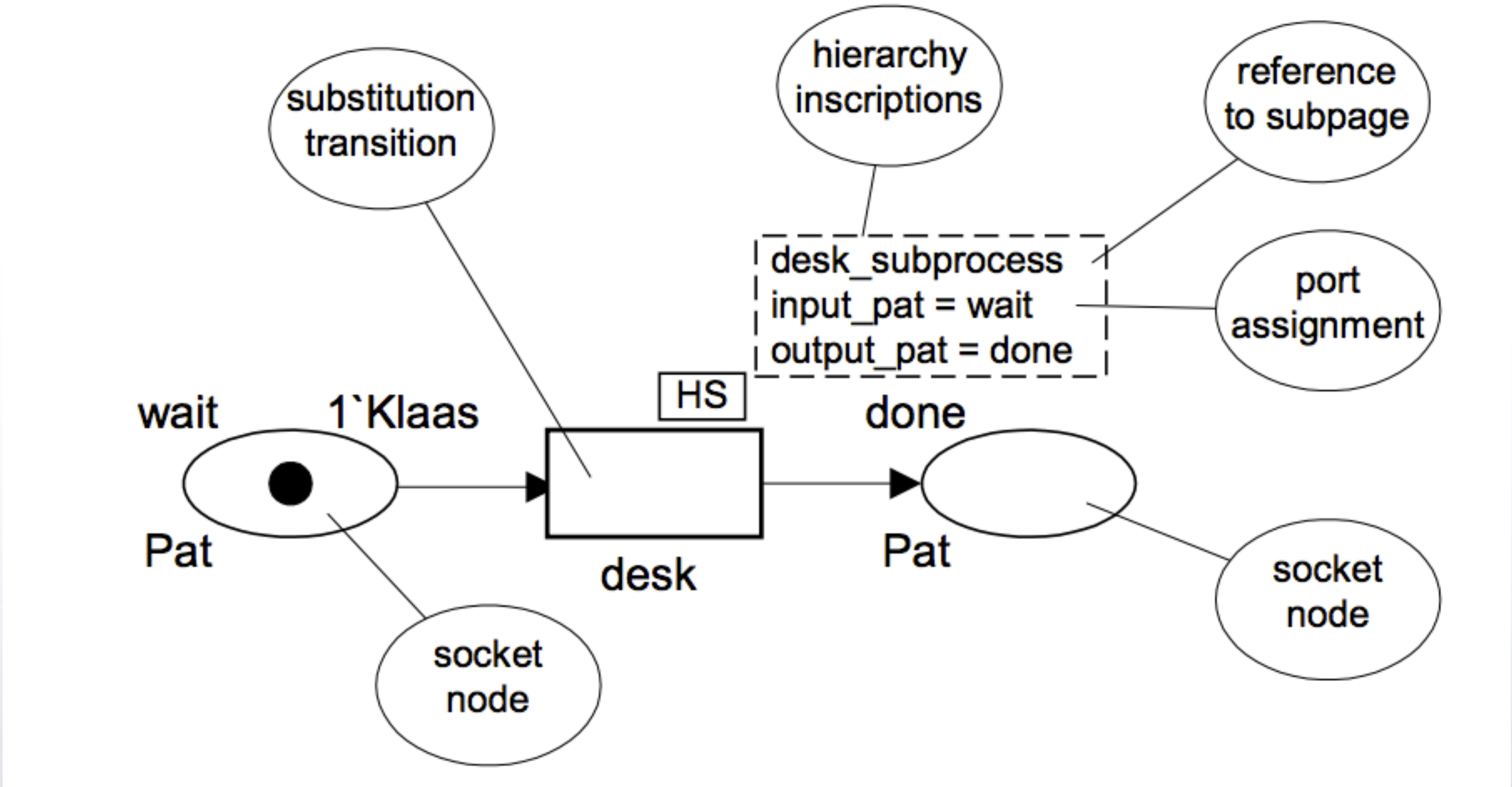
Abstract representation, scale model of future design

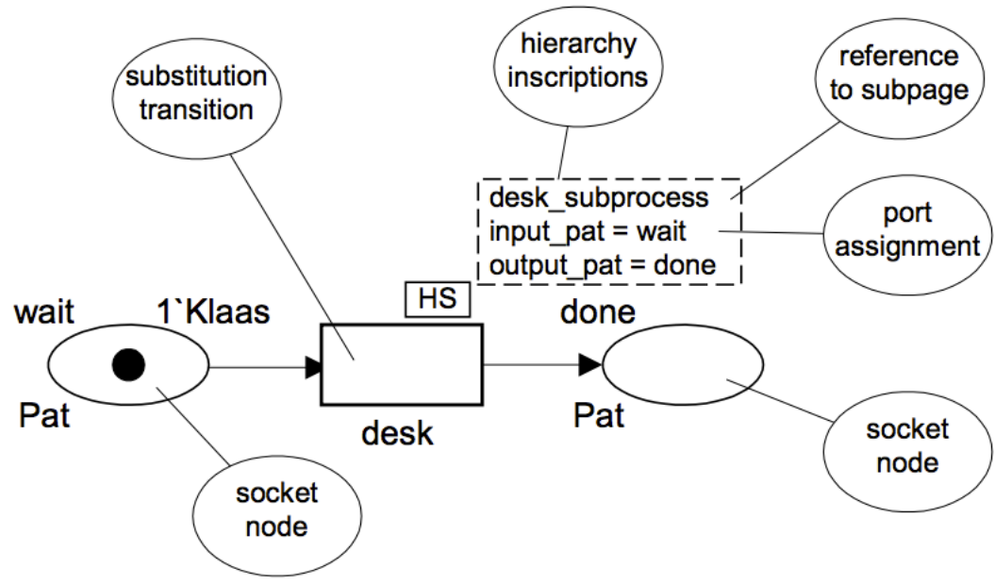
August 27, 2002 Søren Christensen, MOCA'02 7

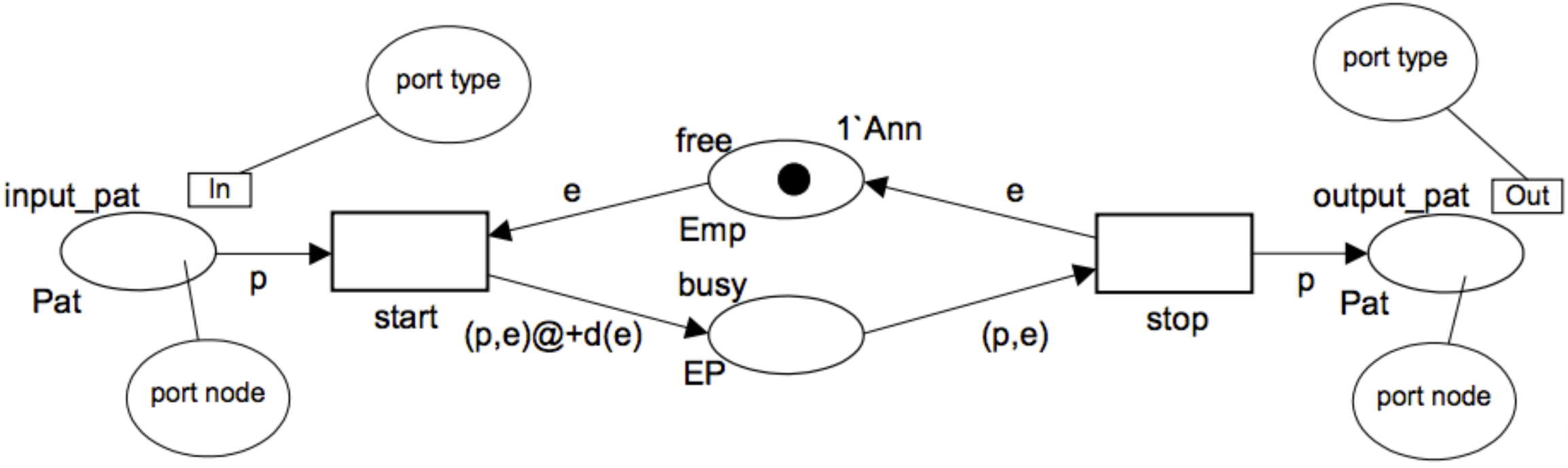

1
2
3
4
5
6
7
8
append()
top()
append()


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

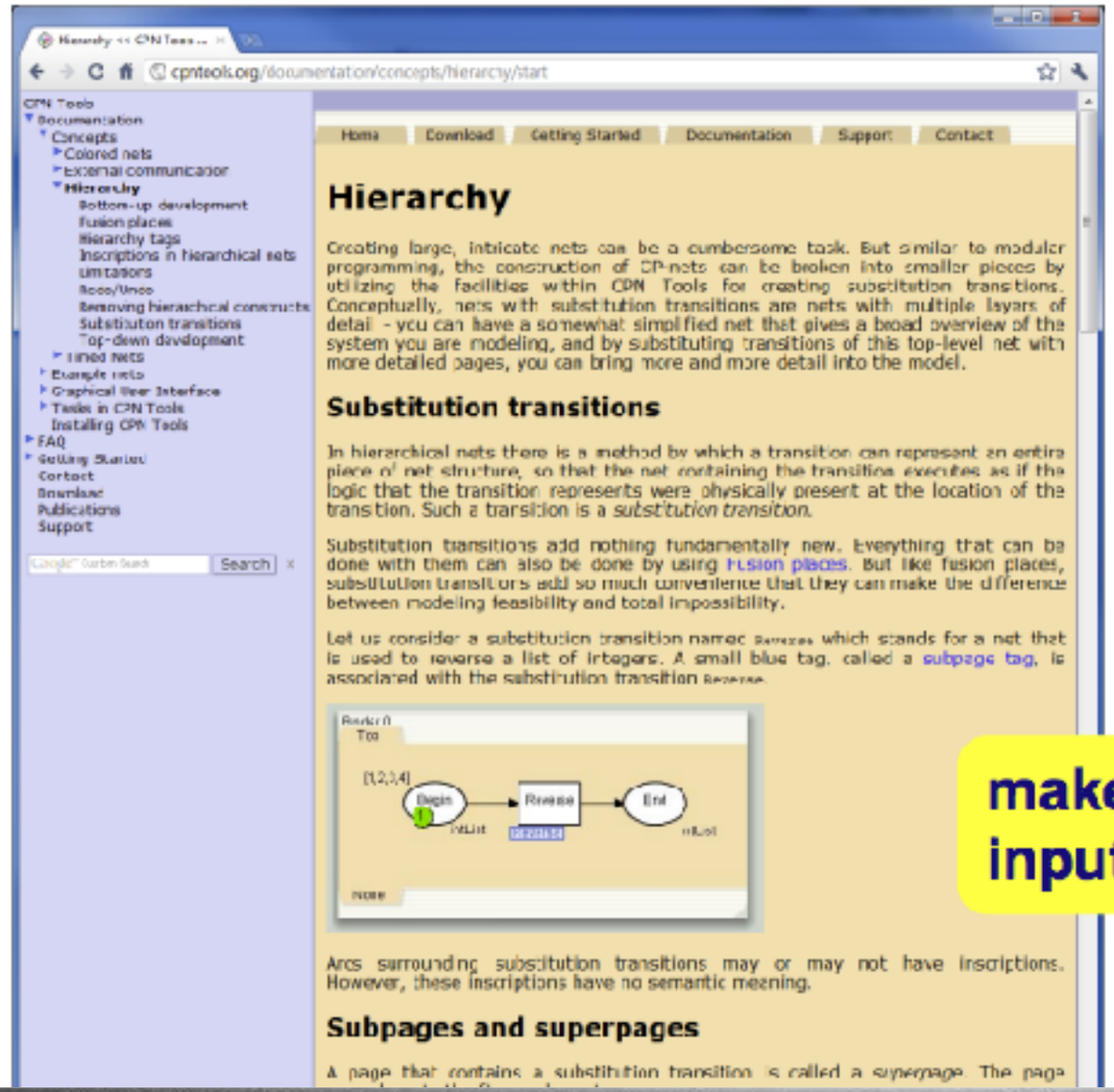

desk_subprocess
input_pat = wait
output_pat = done
HS
wait
1`Klaas
done
Pat
desk
Pat
1`Ann
free
input_pat
In
e
e
output_pat
Out
Emp
p
busy
p
Pat
Pat
start
(p,e)@+d(e)
(p,e)
stop
EP

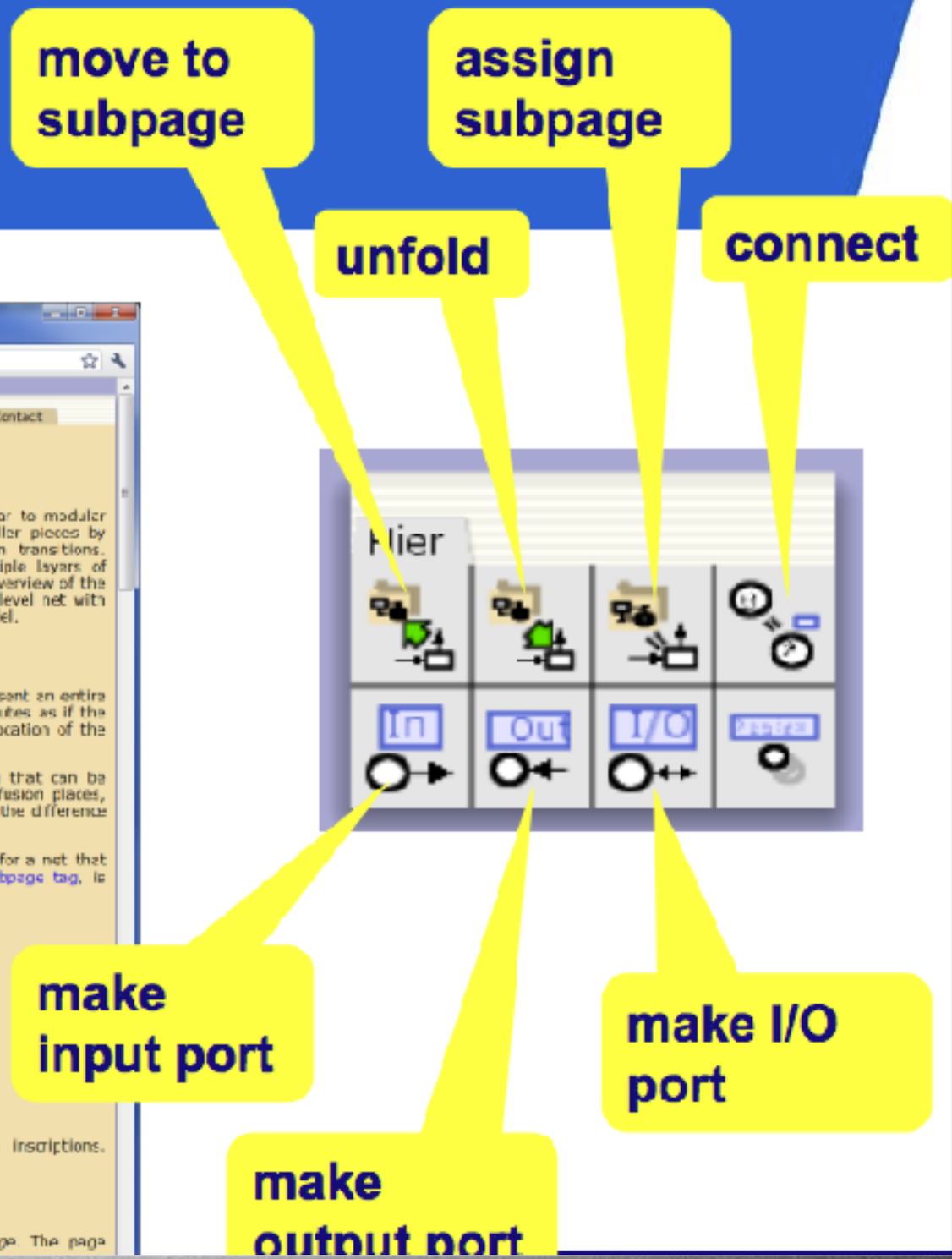

substitution transition
hierarchy inscriptions
reference to subpage
desk_subprocess
input_pat = wait
output_pat = done
port assignment
HS
wait
1`Klaas
done
Pat
desk
Pat
socket node
socket node


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
p
busy
p
Pat
start
(p,e)@+d(e)
(p,e)
stop
Pat
port node
EP
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

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 [ ].

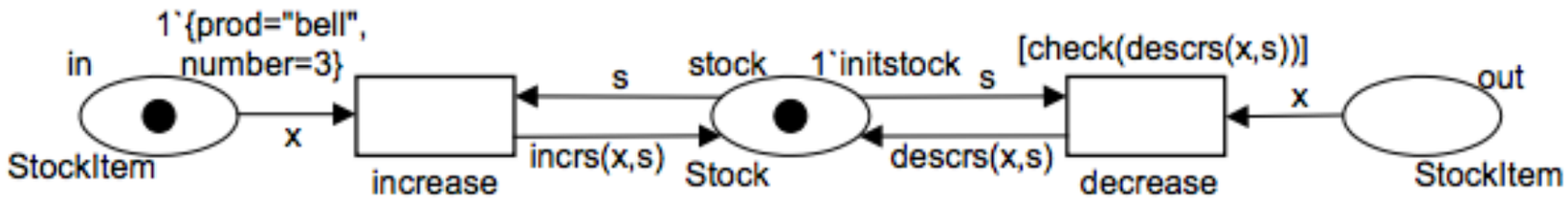
Uma lista não vazia é composta por dois elementos: head, que é o primeiro elemento da lista e tail que é uma lista (necessáriamente menor que a lista original), e se representa como [head | tail]. No exemplo dado anteriormente,

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

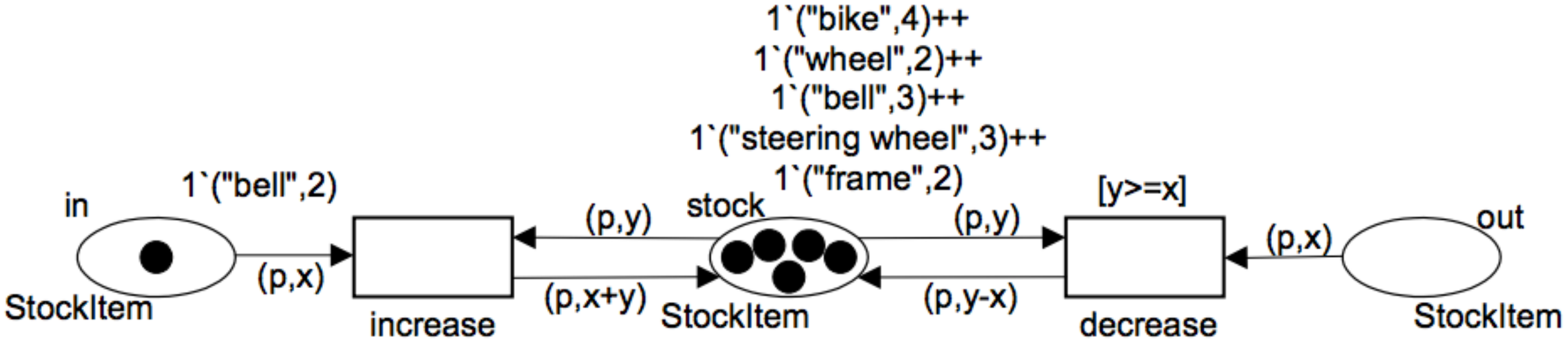
[1, 2, 4, 6, 7, 10] → [1 | [2, 4, 6, 7, 10]]

```
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

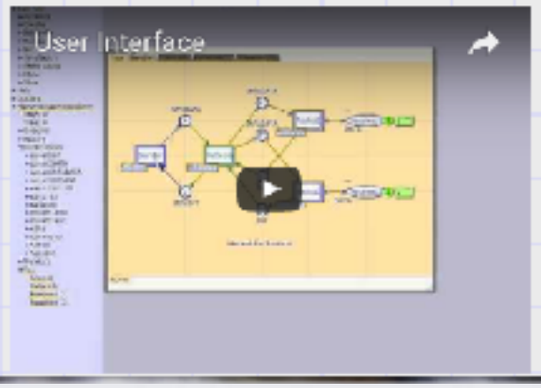
In all of the following and on all of these pages, we assume that you are familiar with coloured Petri nets and have at least some idea of constructing modules using the formalism. You can learn more about the modeling and the formalism in one or both of these books:

The User Interface of CPN Tools

First steps

Graphical User Interface

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

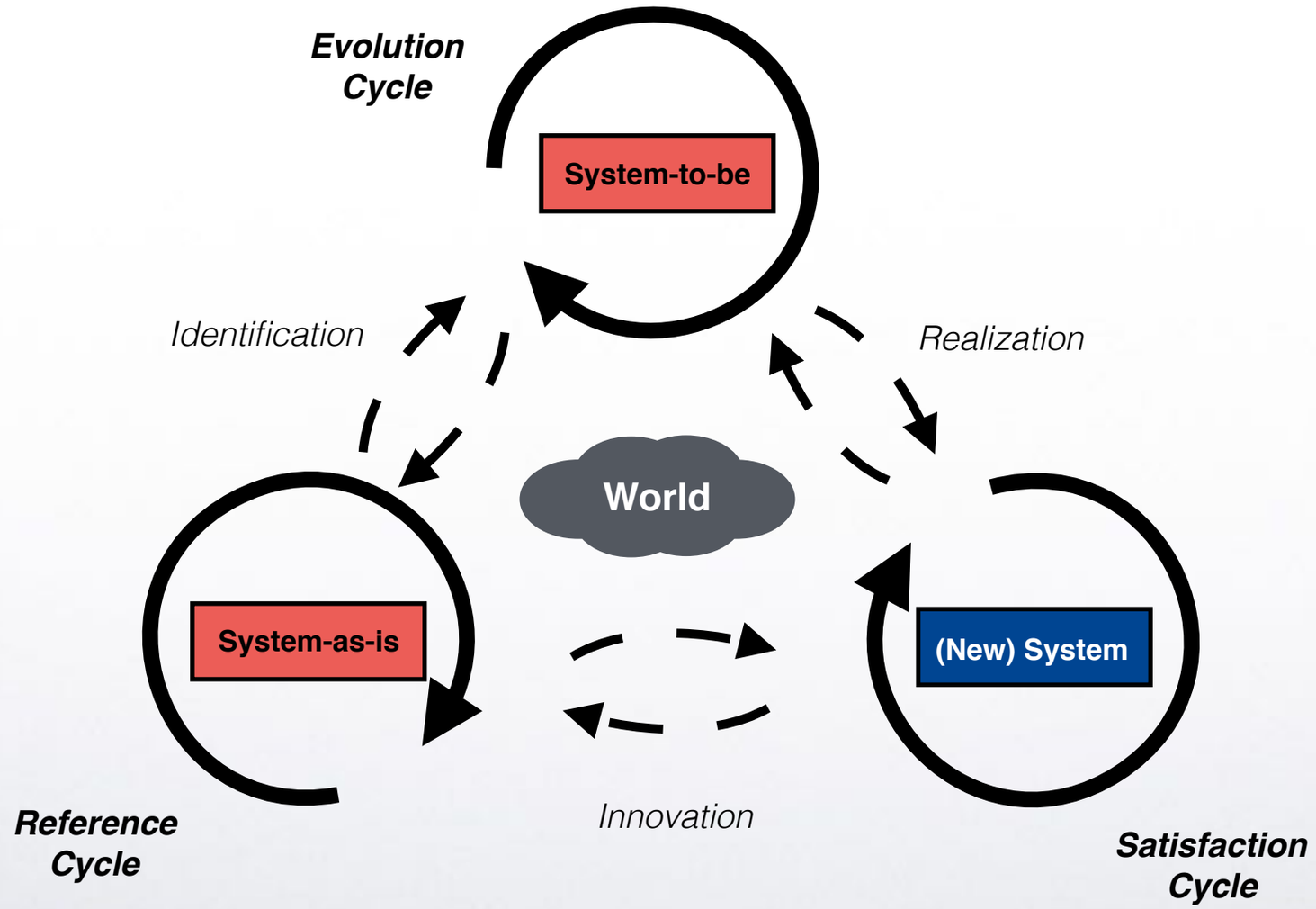


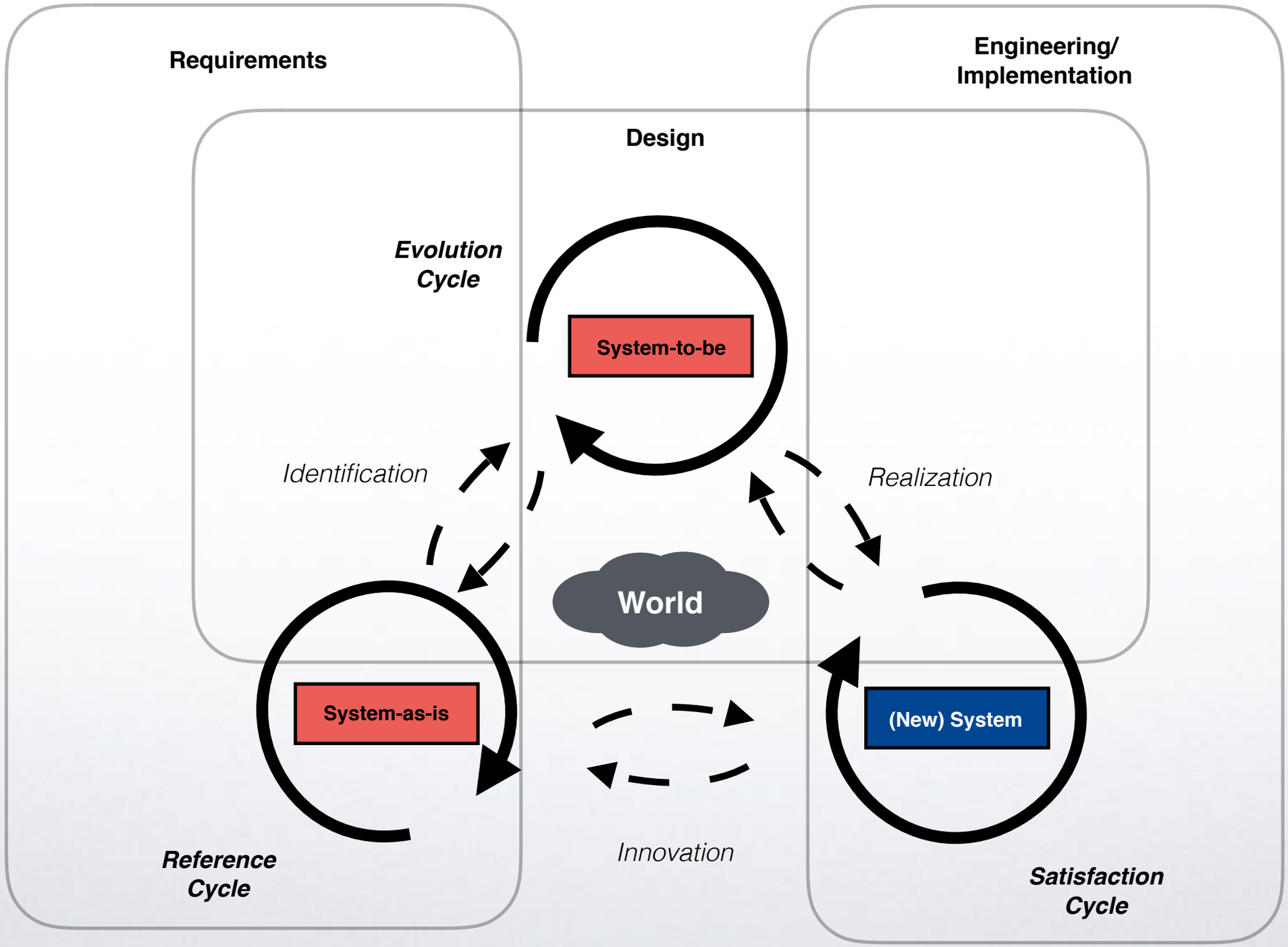
Buscando um processo de projeto

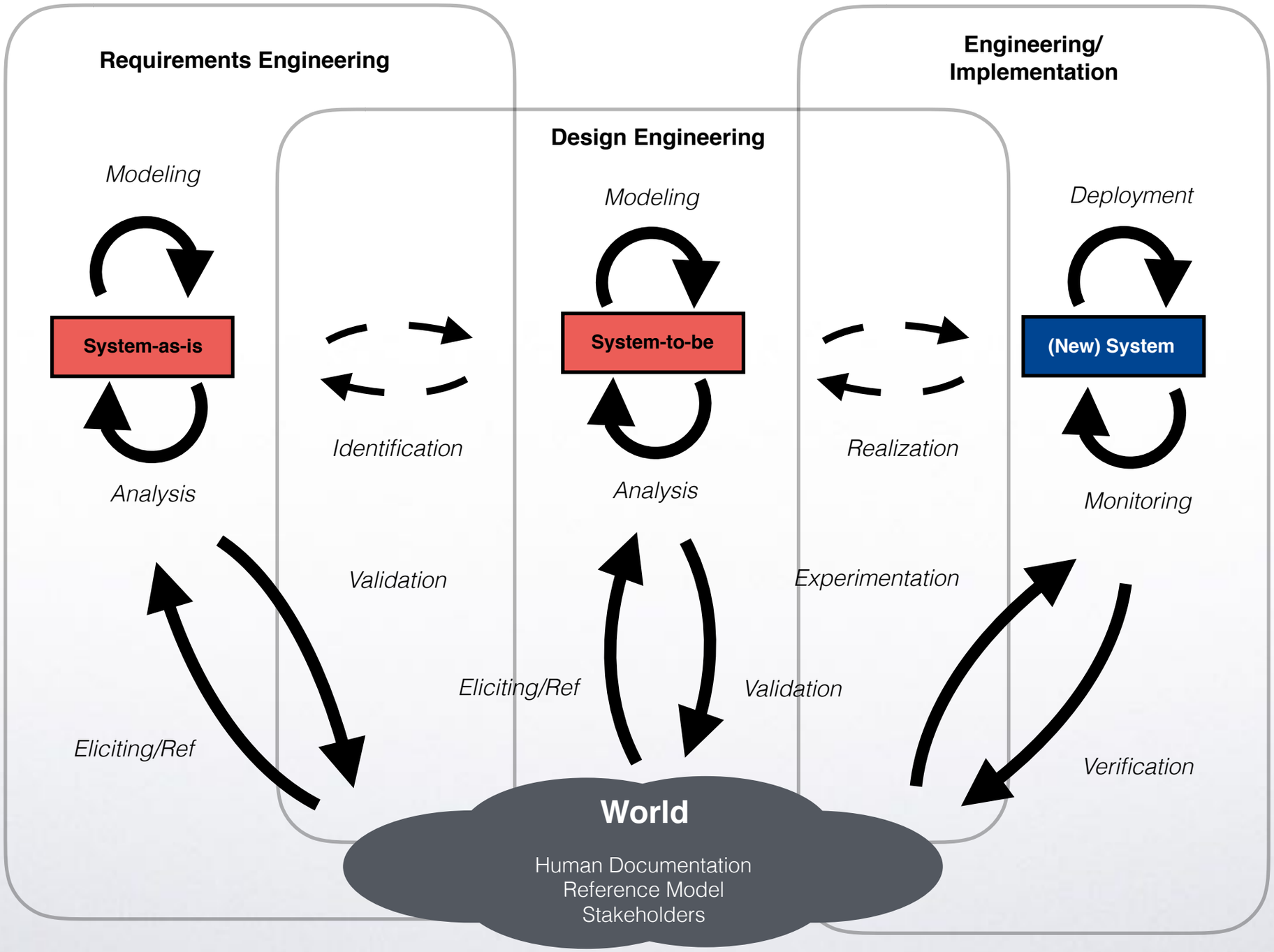
Nos casos em que intuitivamente temos um sistema que é plenamente representado por uma rede clássica, e, mais do que isso, onde este modelo é facilmente e completamente interpretado modelo, é fácil de entender que somente uma demanda por múltiplos casos de simetria ou dobramento nos levaria a apelar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

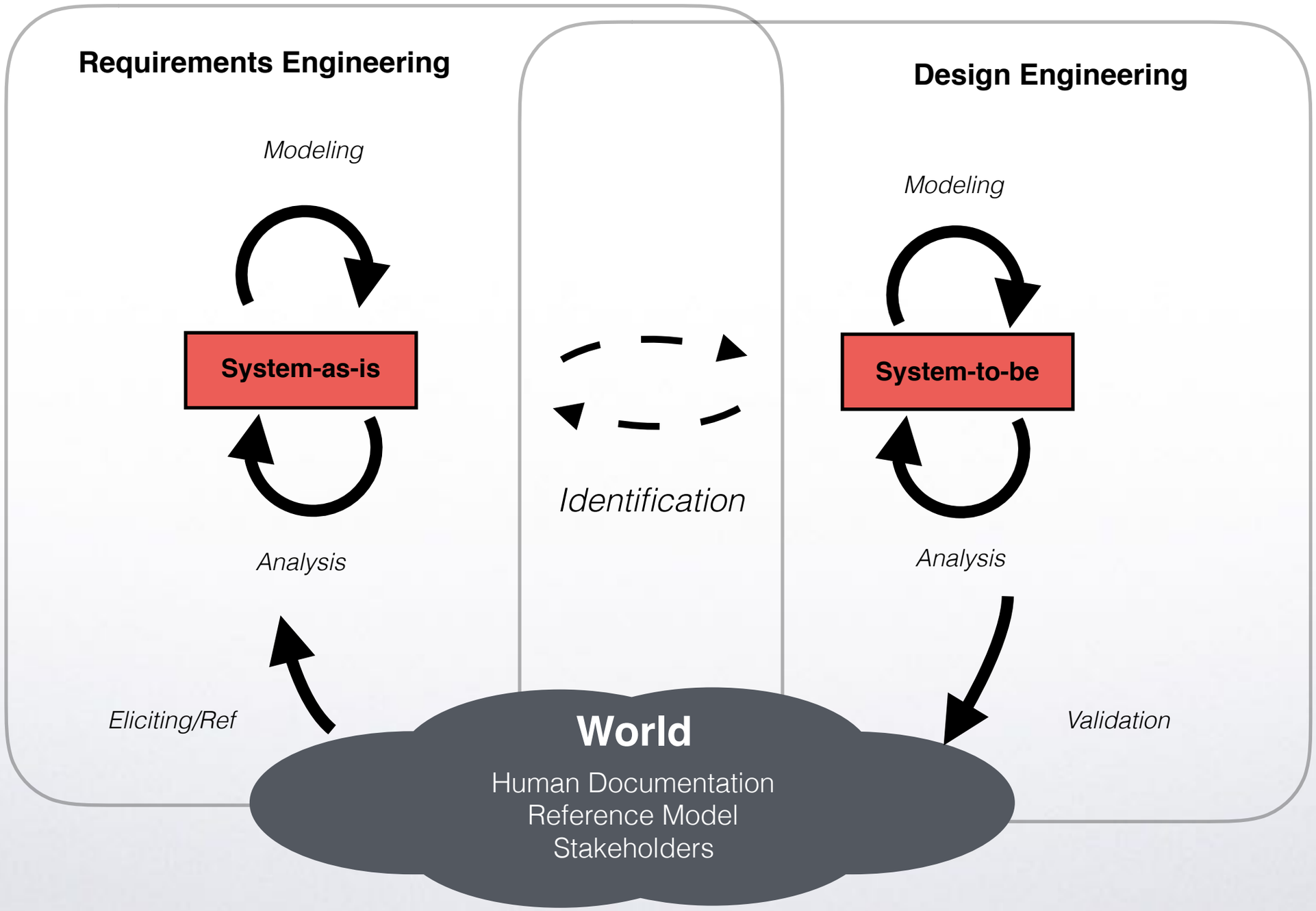
No exercício que acabamos de ver poderíamos introduzir vários tipos de peça no processo de fabricação, cuja "receita" seria dada por diferentes combinações das operações utilizadas operando nas diferentes máquinas. Neste caso seria bastante atraente a distinção de marcas por tipos. Mas será esta a sequencia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

Systems Design and Petri Nets

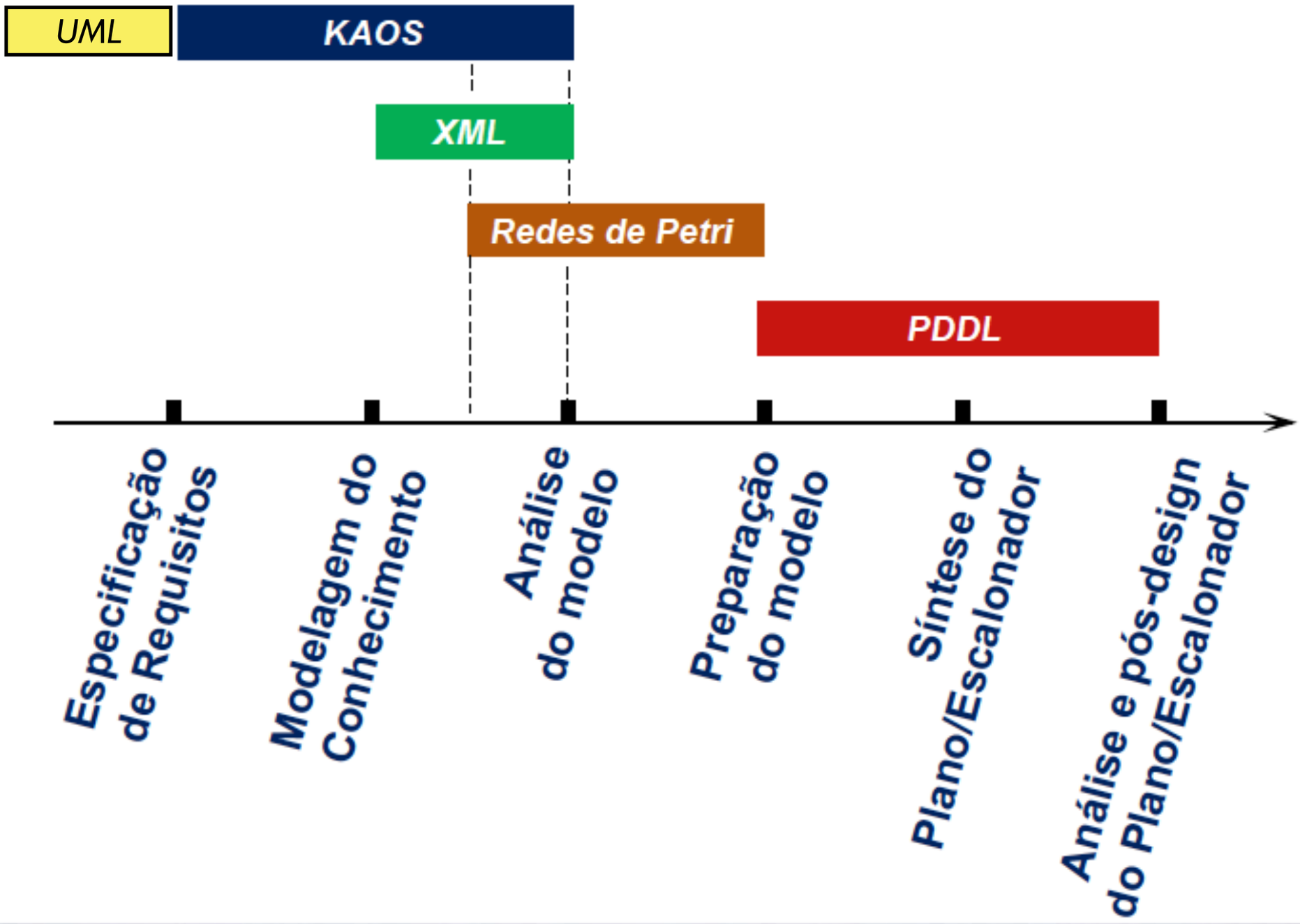



Requirements
Engineering/
Implementation
Design
Evolution
Cycle
System-to-be
Identification
Realization
World
System-as-is
(New) System
Innovation
Reference
Cycle
Satisfaction
Cycle


Requirements Engineering
Design Engineering
Engineering/
Implementation
Modeling
System-as-is
Analysis
Identification
Modeling
System-to-be
Analysis
Realization
Deployment
(New) System
Monitoring
Validation
Eliciting/Ref
Eliciting/Ref
Validation
Experimentation
Verification
World
Human Documentation
Reference Model
Stakeholders


Requirements Engineering
Modeling
System-as-is
Analysis
Identification
Design Engineering
Modeling
System-to-be
Analysis
Eliciting/Ref
Validation
World
Human Documentation
Reference Model
Stakeholders

E se não existir um system-as-is?


UML
KAOS
XML
Redes de Petri
PDDL
Especificação de Requisitos
Modelagem do Conhecimento
Análise do modelo
Preparação do modelo
Síntese do Plano/Escalonador
Análise e pós-design do Plano/Escalonador

No.	Attribute	Values
1	paradigm	state machine, algebra, process algebra, trace
2	formality	informal, semi-formal, formal
3	graphical representation	yes, no
4	object-oriented	yes, no
5	concurrency	yes, no
6	executability	yes, no
7	usage of variables	yes, no
8	non-determinism	yes, no
9	logic	yes, no
10	provability	yes, no
11	model checking	yes, no
12	event inhibition	yes, no

method name	paradigm	formality	graphical representation	object-oriented
Action Systems	state transition	formal	no	no
B	state transition	formal	no	no
CASL	algebra	formal	no	yes
Cleanroom & JSD	traces & process algebra	formal	yes	no
COQ	state transition	formal	no	no
Estelle	state transition	formal	no	no
LOTOS	process algebra	formal	no	yes
OMT & B	state transition	formal	yes	yes
Petri Nets	state transition	formal	yes	no
Petri Nets with Objects	state transition	formal	yes	yes
SART	state transition	informal & semi-formal	yes	no
SAZ	state transition	semi-formal & formal	yes	no
SCCS	process algebra	formal	no	no
SDL	state transition	formal	yes	yes
UML	state transition	informal & semi-formal	yes	yes
VHDL	state transition	formal	no	no
Z	state transition	formal	no	no

method name	concurrency	executability	usage of variables	non-determinism
Action Systems	no	yes	yes	yes
B	no	yes	yes	yes
CASL	no	yes	yes	no
Cleanroom & JSD	no	yes	yes	yes
COQ	no	yes	yes	yes
Estelle	yes	yes	yes	no
LOTOS	yes	yes	yes	yes
OMT & B	no	yes	yes	yes
Petri Nets	yes	yes	no	yes
Petri Nets with Objects	yes	yes	yes	yes
SART	yes	no	no	yes
SAZ	no	yes	yes	yes
SCCS	yes	yes	yes	yes
SDL	yes	yes	no	yes
UML	yes	no	no	no
VHDL	yes	yes	yes	no
Z	no	yes	yes	yes

Petri Nets pode se propagar pelo processo de design de sistemas desde a fase de requisitos até a fase de especificação da solução, passando pelos processos de validação e verificação

Fim

do curso