PMR 5237

Modelagem e Design de Sistemas Discretos em Redes de Petri

Aula 11: 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.

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

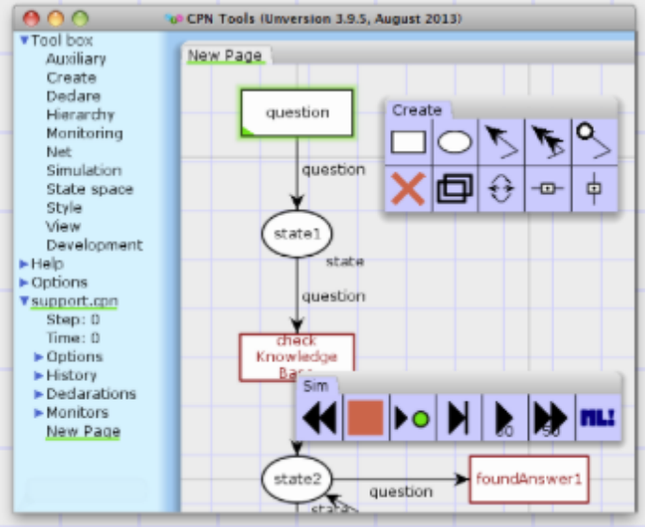
Like Page

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

What is a model?

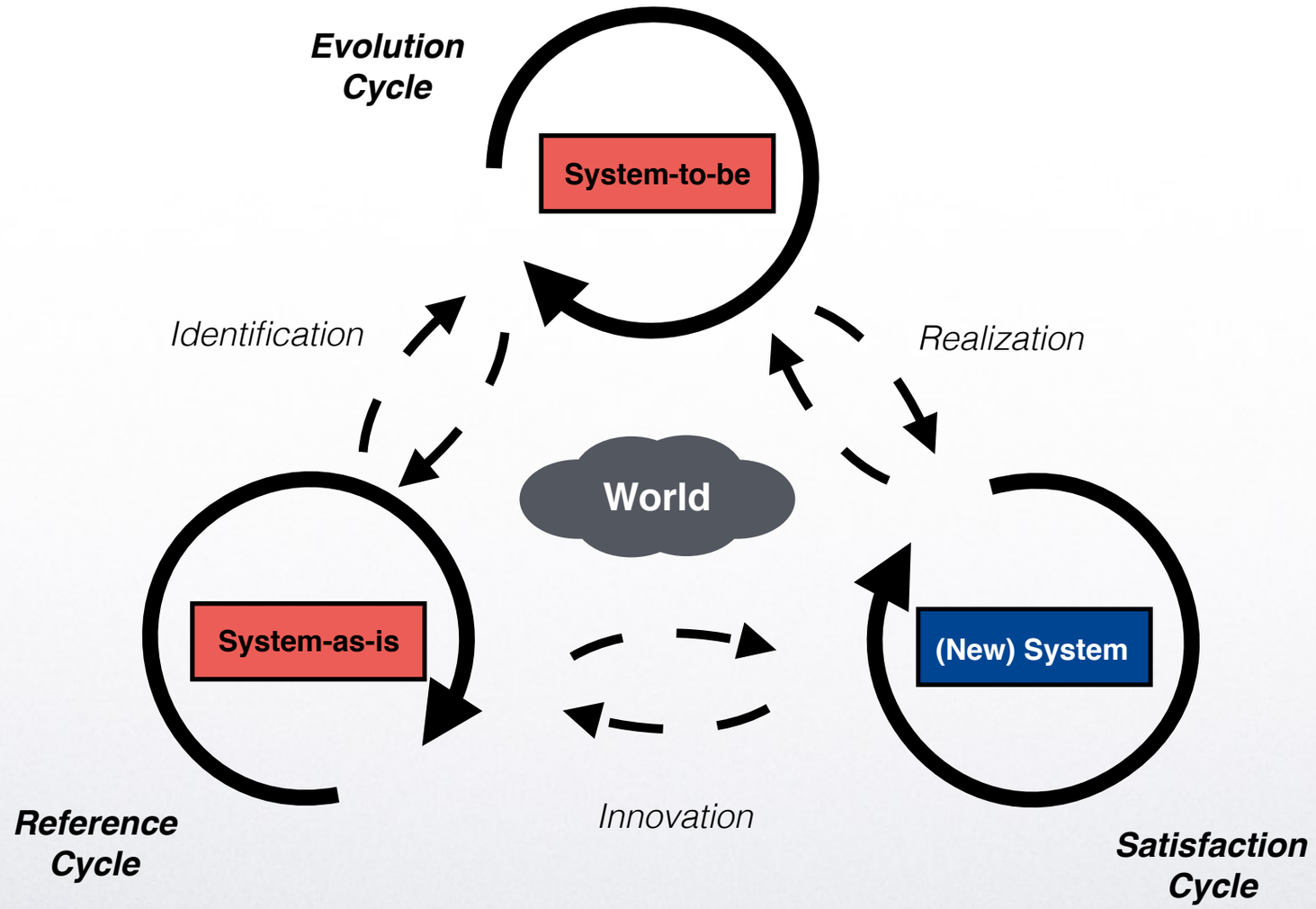
model and modelling
in painting, the *use of light and shade to simulate volume in the representation of solids*. In sculpture the terms denote a technique involving the use of a pliable material such as clay or wax. As opposed to carving, modelling permits addition as well as subtraction of material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

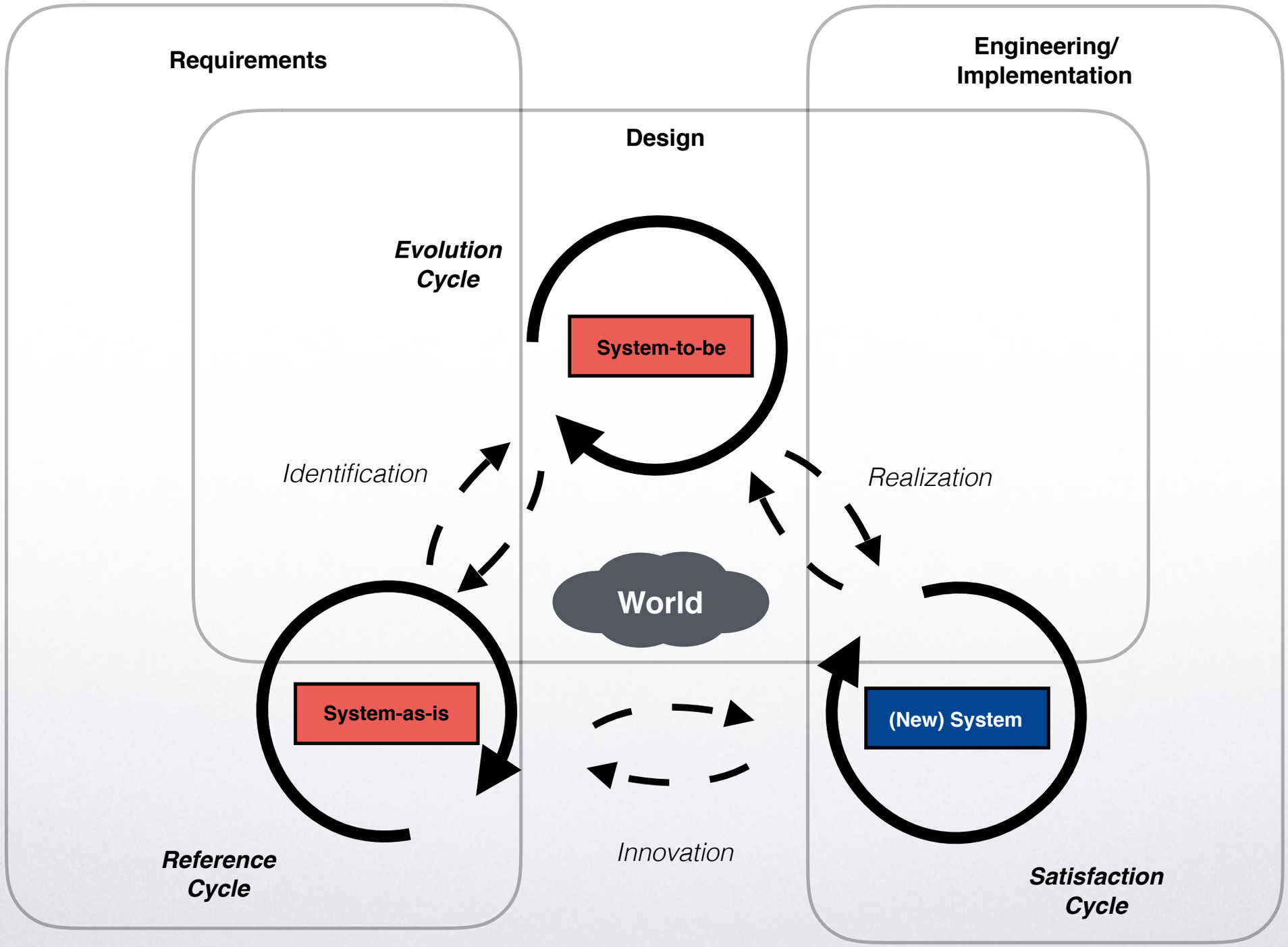
The Columbia Encyclopaedia, Sixth Edition. 2001.

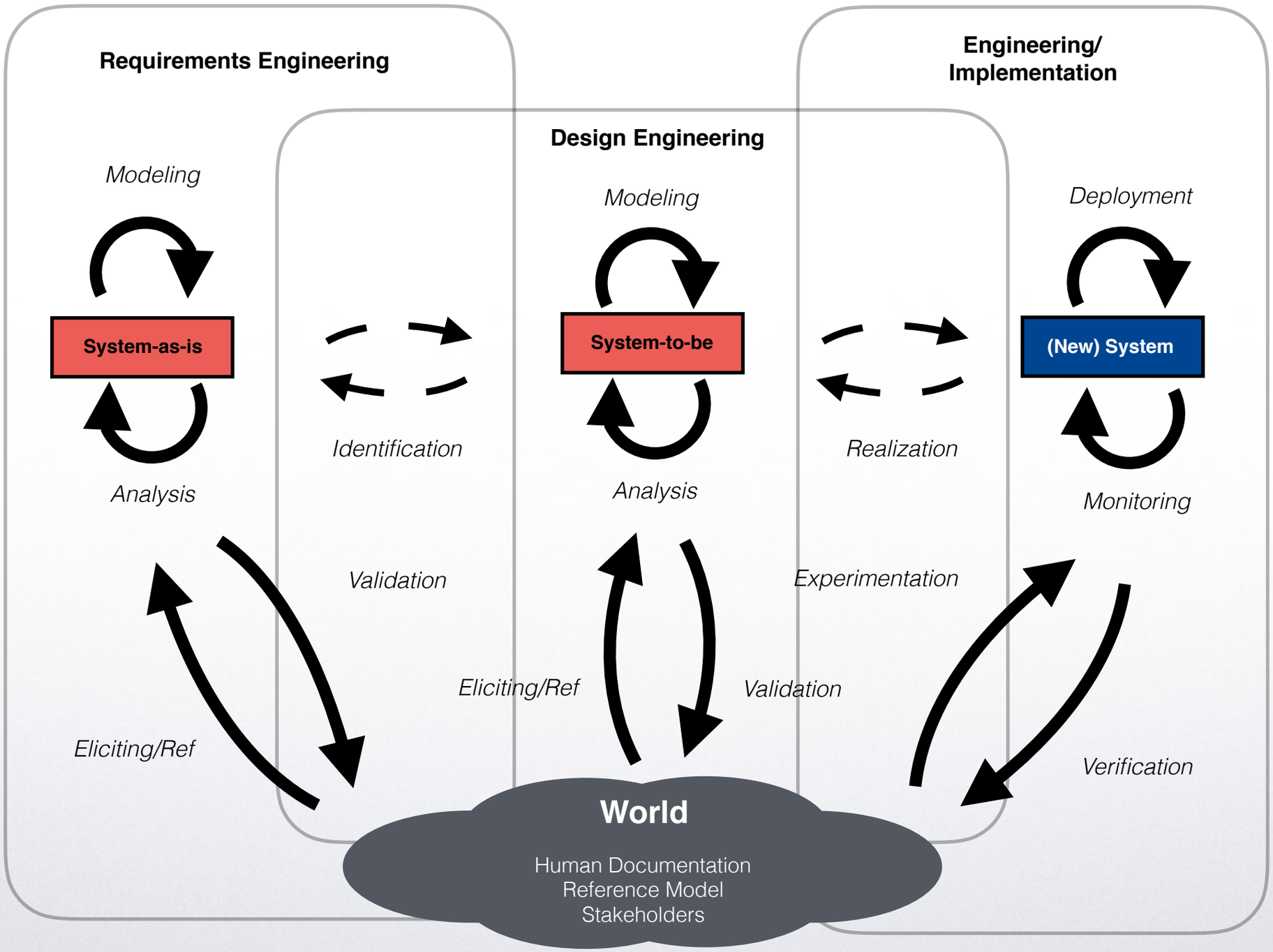
Abstract representation, scale model of future design

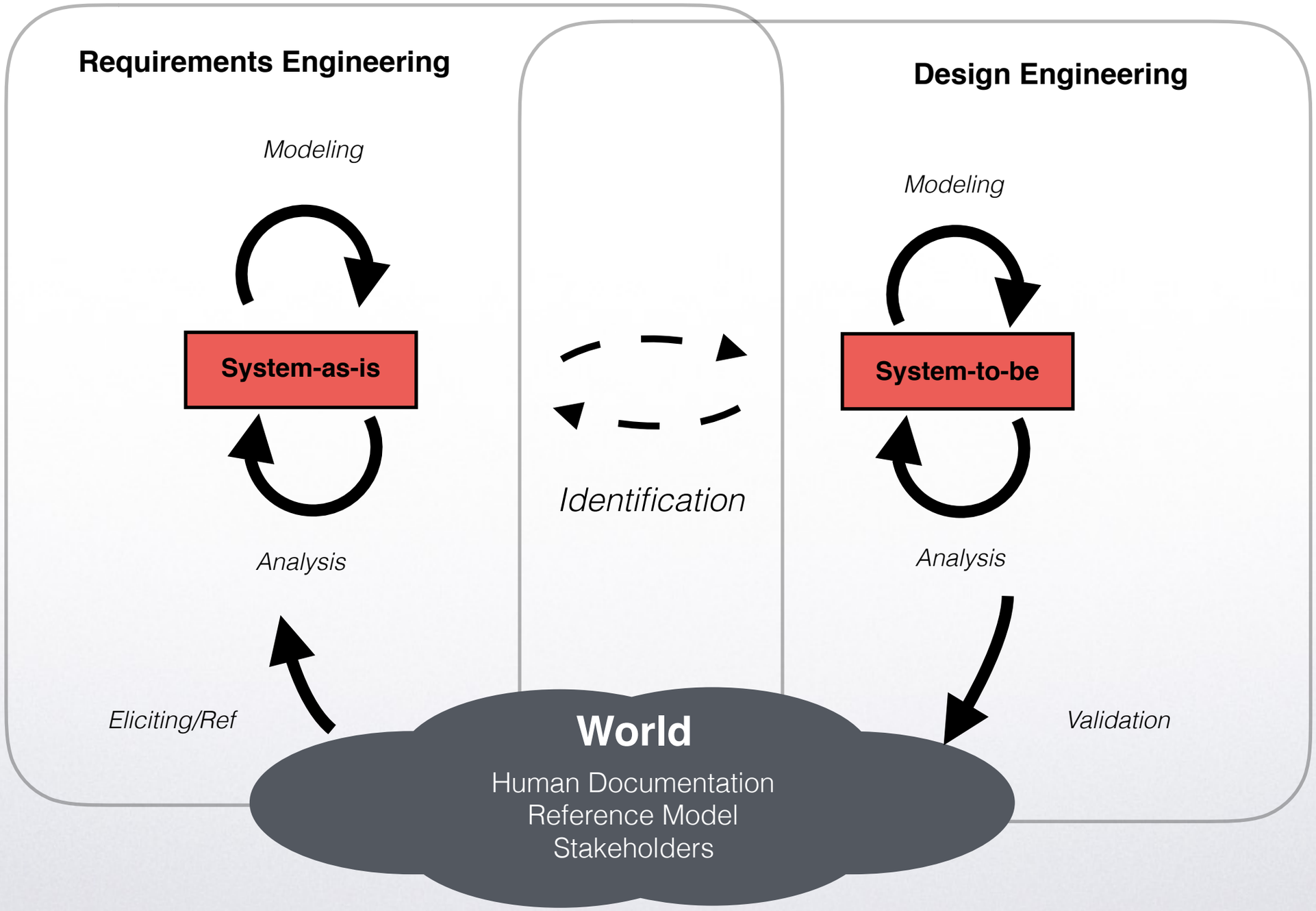
August 27, 2002 Søren Christensen, MOCA'02 7

Systems Design and Petri Nets

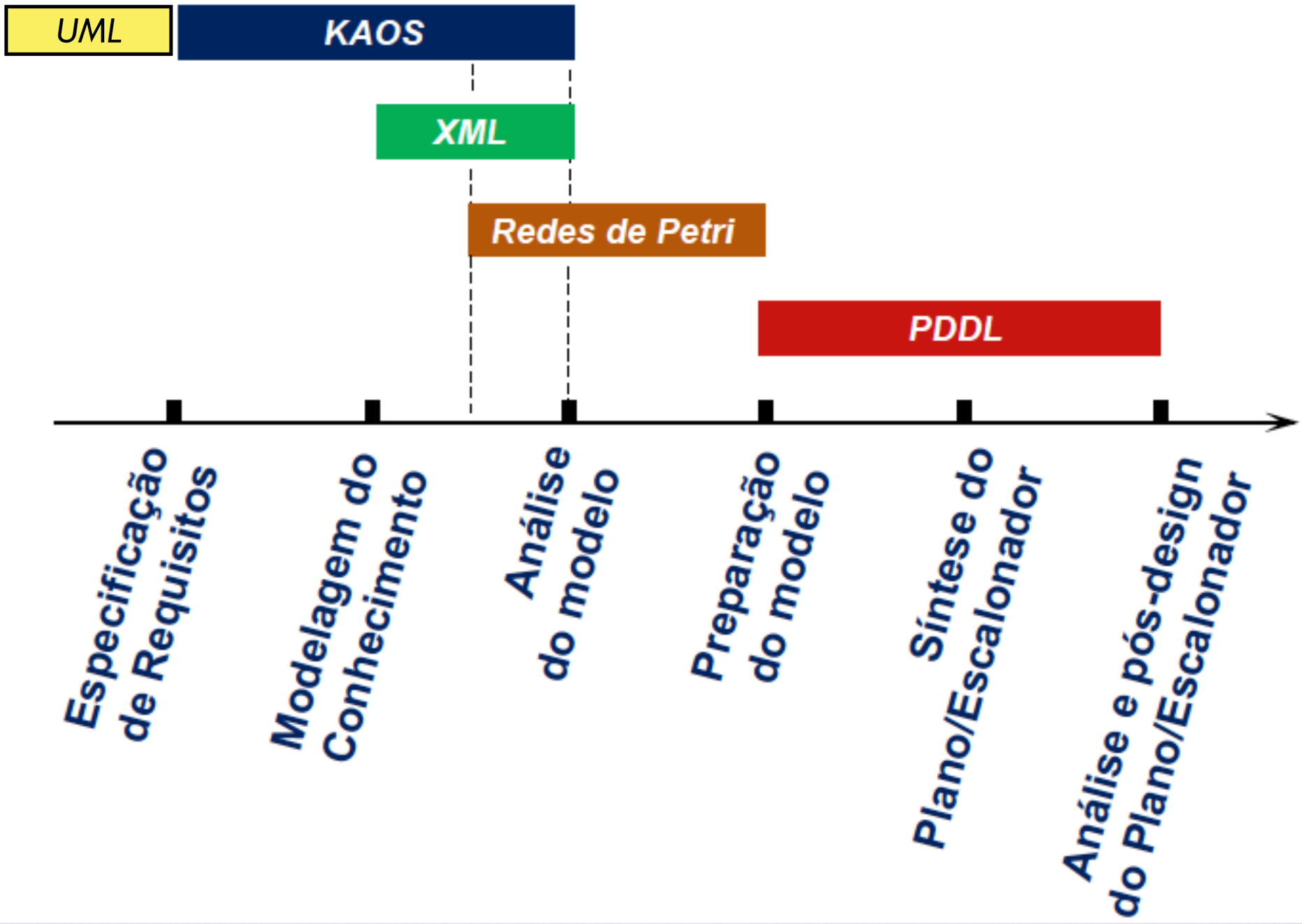



Requirements
Engineering/
Implementation
Design
Evolution
Cycle
System-to-be
Identification
Realization
World
System-as-is
(New) System
Innovation
Reference
Cycle
Satisfaction
Cycle


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


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

E se não existir um system-as-is?

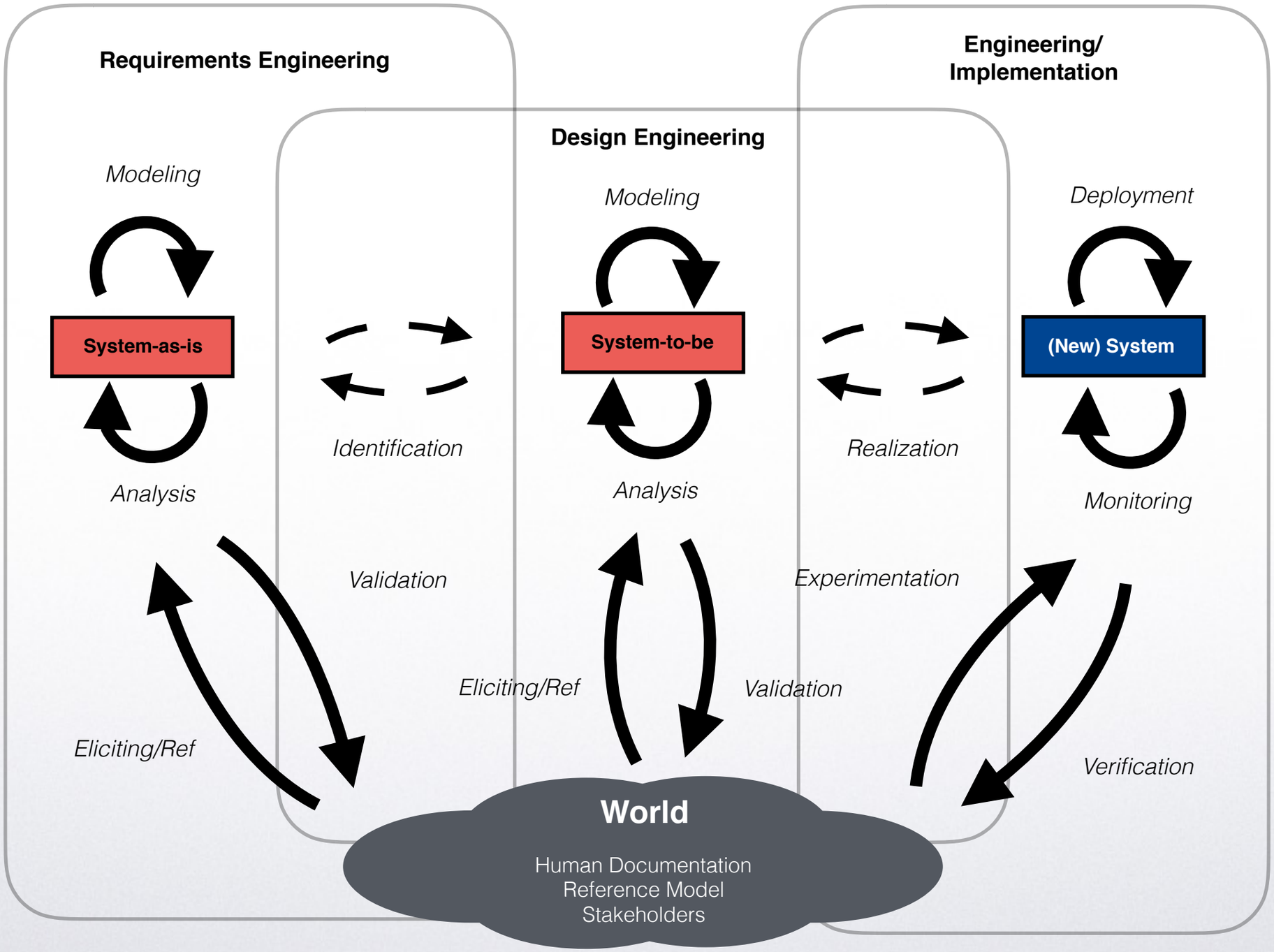

UML
KAOS
XML
Redes de Petri
PDDL
Especificação de Requisitos
Modelagem do Conhecimento
Análise do modelo
Preparação do modelo
Síntese do Plano/Escalonador
Análise e pós-design do Plano/Escalonador

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

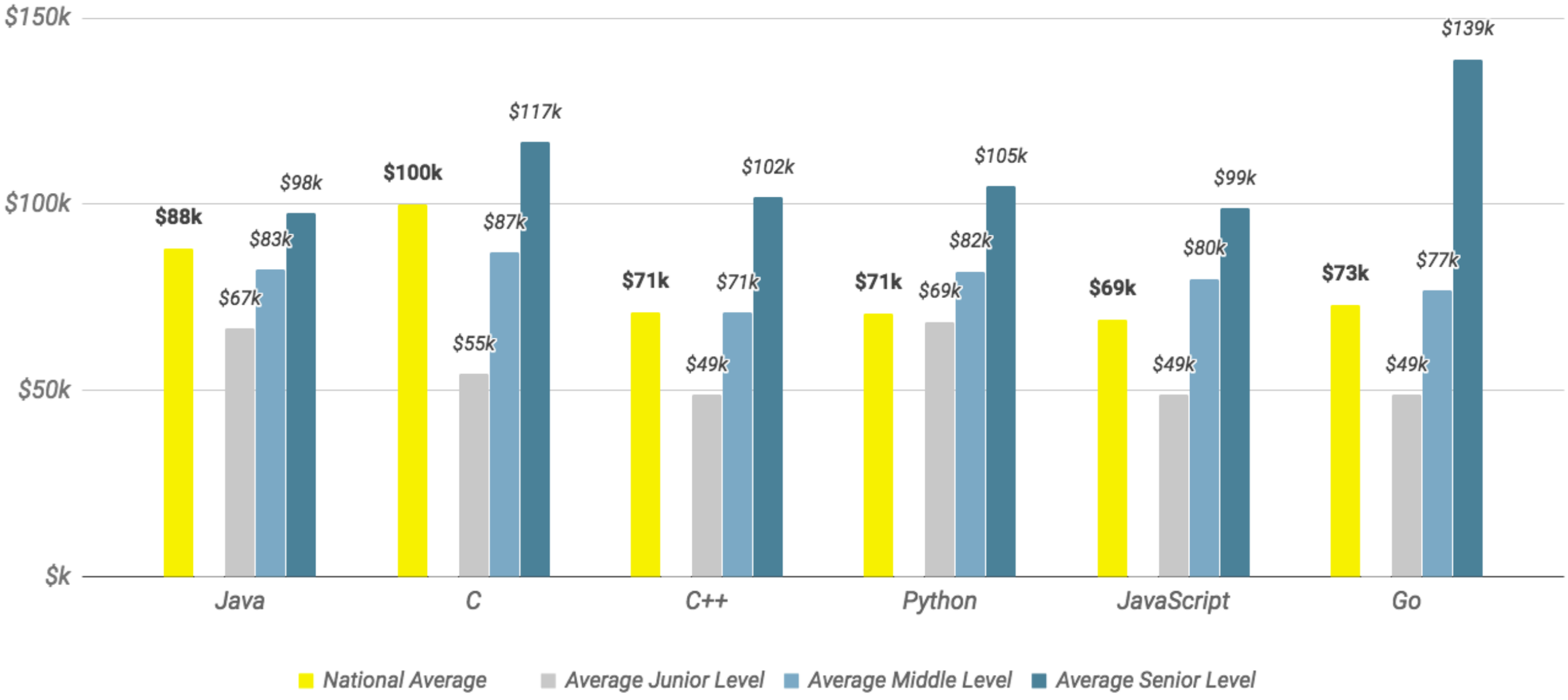

Requirements Engineering
Design Engineering
Engineering/
Implementation
Modeling
System-as-is
Analysis
Identification
Validation
Modeling
System-to-be
Analysis
Realization
Eliciting/Ref
Deployment
(New) System
Monitoring
Experimentation
Eliciting/Ref
Validation
Verification
World
Human Documentation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

Quando o tema é controle discreto, temos o problema de ter o instrumento de controle básico, o PLC programado em um nível muito baixo comparado com o nível da modelagem em redes de Petri.

PLCs modernos

PLCs com HMI acoplado: na direção da simbiose homem-máquina

IoT Developer Salaries in the US (by programming language and seniority)



The 5 Languages of IEC 61131-3

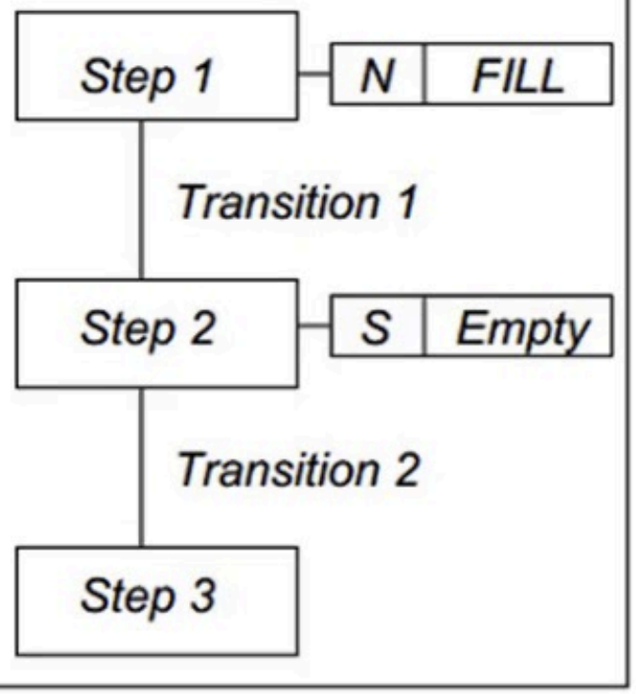
Instruction List

```
LD    A
ANDN  B
ST    C
```

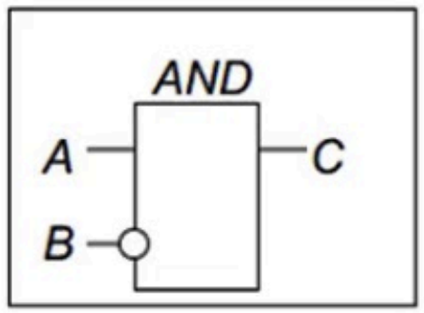
Structured Text

```
C:= A  AND  NOT B
```

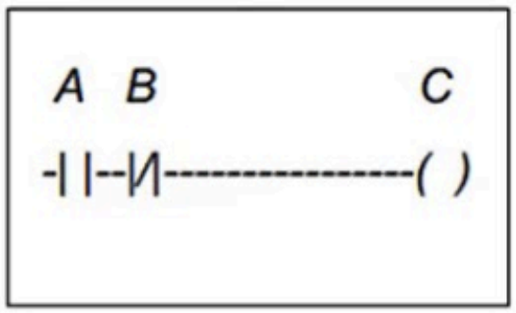
Sequential Function Chart



Function Block Diagram



Ladder Diagram



IEC 61131-3 standard | 1-Programming languages

The 5 Languages of IEC 61131-3

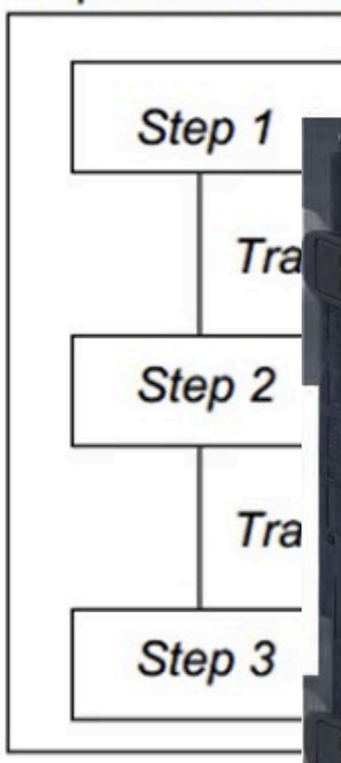
Instruction List

```
LD      A
ANDN    B
ST      C
```

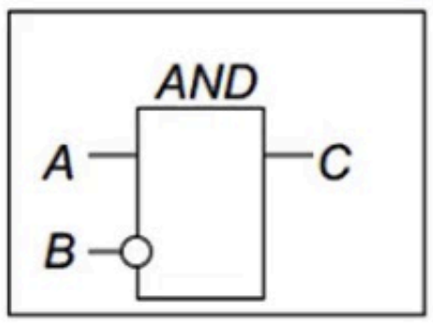
Structured Text

```
C:= A  AND  NOT B
```

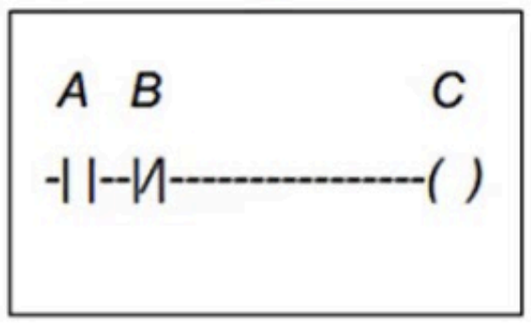
Sequential Function Chart



Function Block Diagram



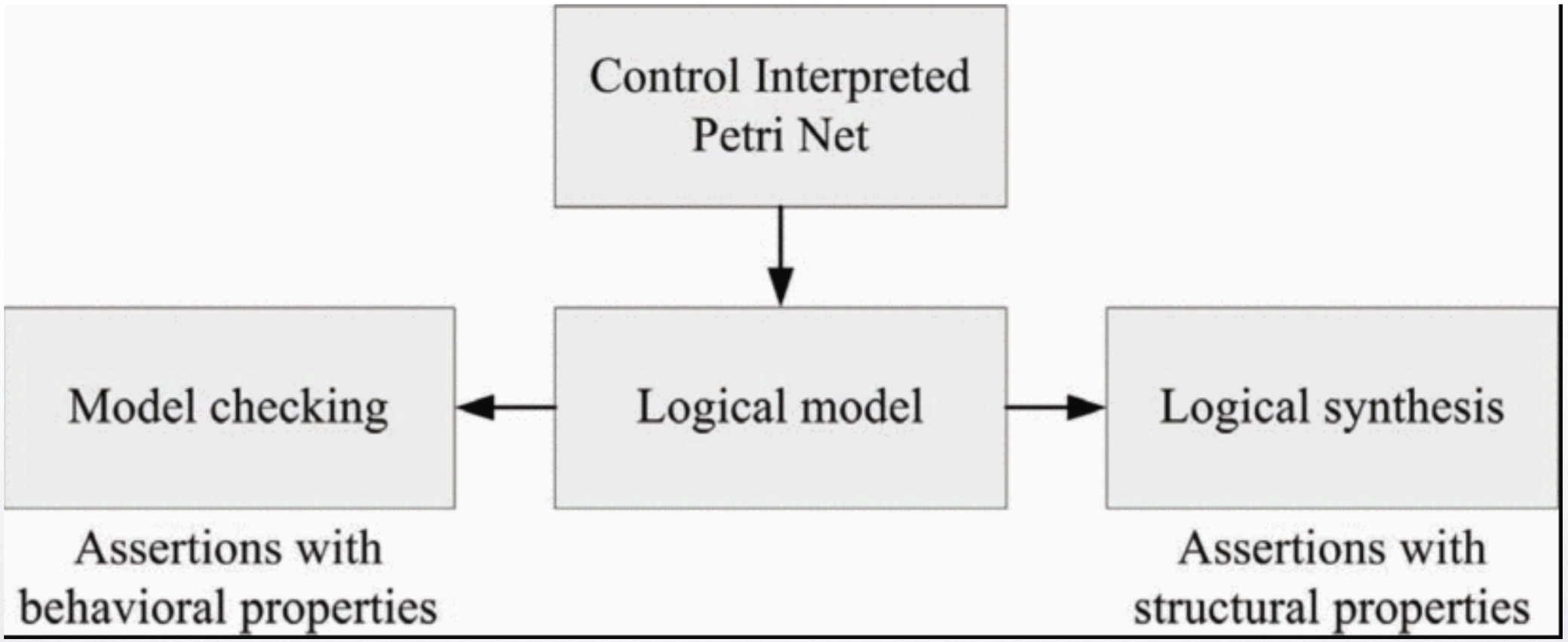
Ladder Diagram



Programming | PLC programming language

Mais do que "instrumentação", o problema mais geral é como modelar comportamento, ao invés de ação direta de controle.

Grobelna, I., Adamski, M., Model checking of interpreted controlled Petri Nets, Procc. of the 18th. Int. Conf. on Mixed Design of Integrated Circuits and Systems, Gliwice, Poland, 2011.



Formally, an Interpreted Petri Net can be defined [2] as a 6-tuple $PN_{IO} = (PN, X, Y, \rho, \lambda, \gamma)$, where:

1. PN is an alive and safe Petri net

2. X is a set of input states

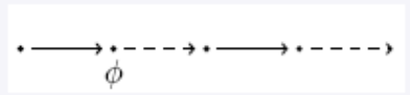
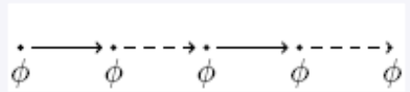
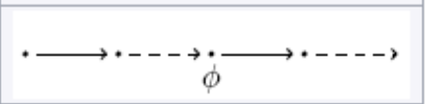
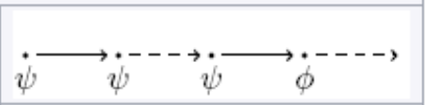
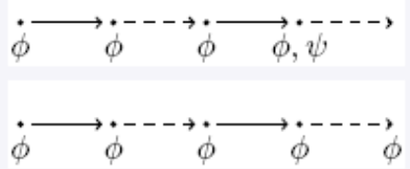
3. Y is a set of output states

4. $\rho : T \rightarrow 2^X$ is a function, that each transition assigns the subset of input states $X(T)$; 2^X states for the set of all possible subsets of X

5. $\lambda : M \rightarrow Y$ is a function of Moore outputs, that each marking M assigns the subset of output states $Y(M)$

6. $\gamma : (M \times X) \rightarrow Y$ is a function of Mealy outputs, that each marking M and input states X assigns the subset of output states Y

Linear-time Tree Logic (LTL)

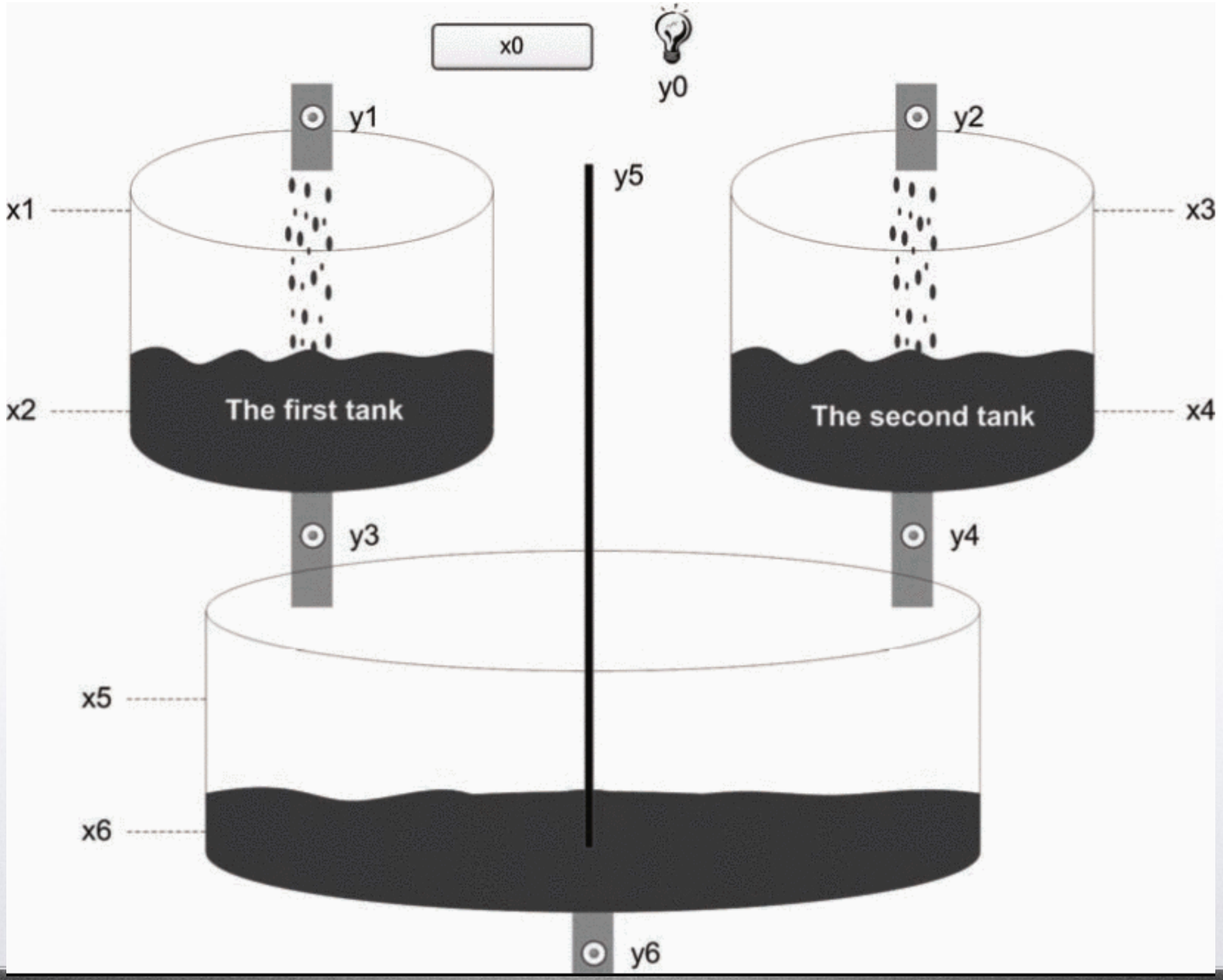
Textual	Simbólico†	Explicação	Diagrama
Unary operators:			
X ϕ	$\bigcirc\phi$	ne**X**t(Próximo): ϕ Tem de manter no próximo estado.	
G ϕ	$\Box\phi$	**G**lobally(Globalmente): ϕ Tem que segurar todo o caminho subseqüente.	
F ϕ	$\Diamond\phi$	**F**inally(Finalmente): ϕ Eventualmente tem que segurar (em algum lugar no caminho subseqüente).	
Binary operators:			
ψ **U** ϕ	$\psi\,\mathcal{U}\,\phi$	Until**(Até)**: ψ Tem de manter pelo menos até ϕ, Que se mantém na posição atual ou futura.	
ψ **R** ϕ	$\psi\,\mathcal{R}\,\phi$	**R**elease(Lançamento): ϕ Tem de ser verdade até e incluindo o ponto onde ψ Primeiro se torna verdadeira; E se ψNunca se torna verdade, ϕ Deve permanecer verdadeiro para sempre.	

Computation Tree Logic (CTL)

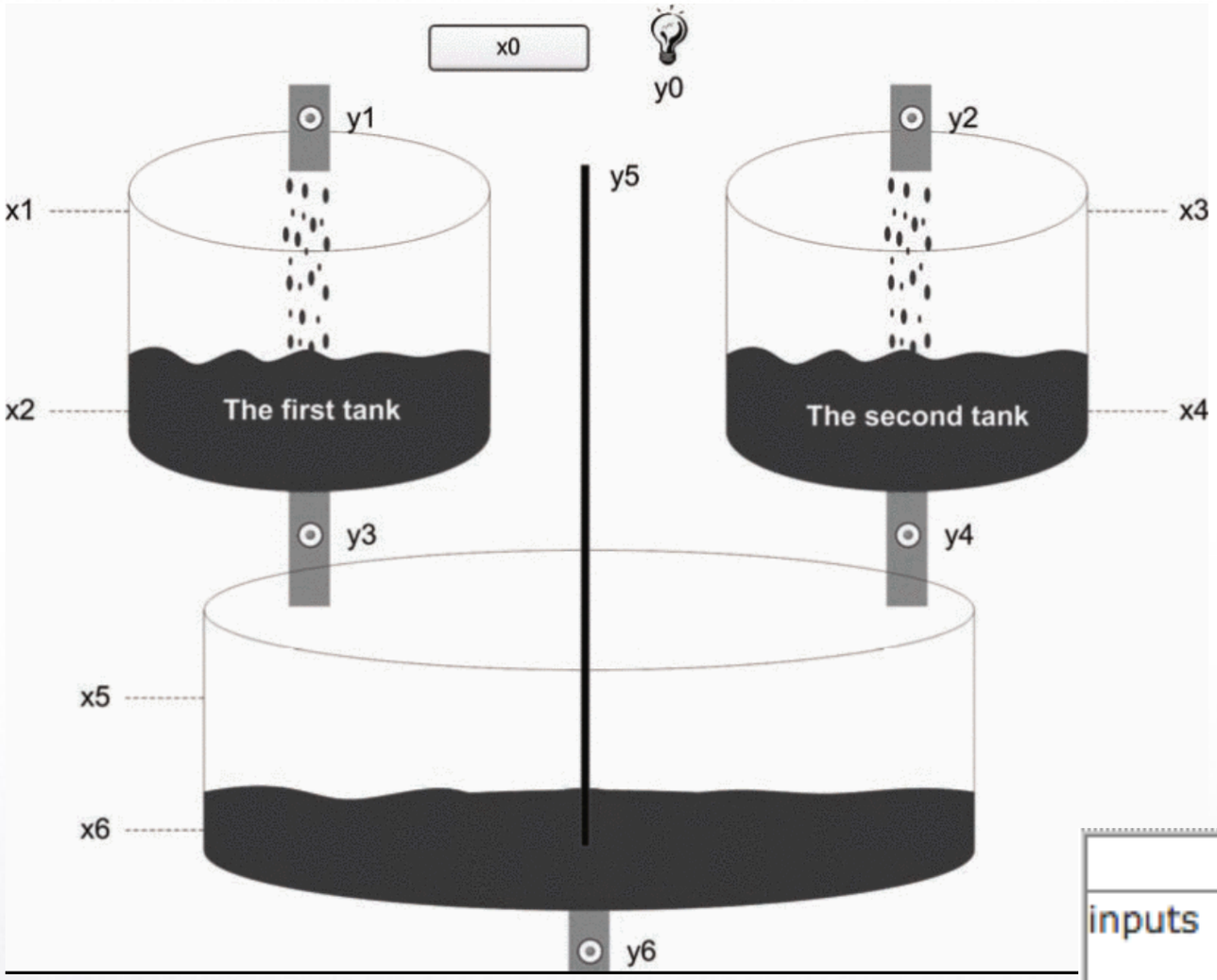
The temporal operators are the following:

- Quantifiers over paths
  - **A** ϕ – **A**ll: ϕ has to hold on all paths starting from the current state.
  - **E** ϕ – **E**xists: there exists at least one path starting from the current state where ϕ holds.
- Path-specific quantifiers
  - **X** ϕ – Ne**x**t: ϕ has to hold at the next state (this operator is sometimes noted **N** instead of **X**).
  - **G** ϕ – **G**lobally: ϕ has to hold on the entire subsequent path.
  - **F** ϕ – **F**inally: ϕ eventually has to hold (somewhere on the subsequent path).
  - ϕ **U** ψ – **U**ntil: ϕ has to hold *at least* until at some position ψ holds. This implies that ψ will be verified in the future.
  - ϕ **W** ψ – **W**eak until: ϕ has to hold until ψ holds. The difference with **U** is that there is no guarantee that ψ will ever be verified. The **W** operator is sometimes called "unless".

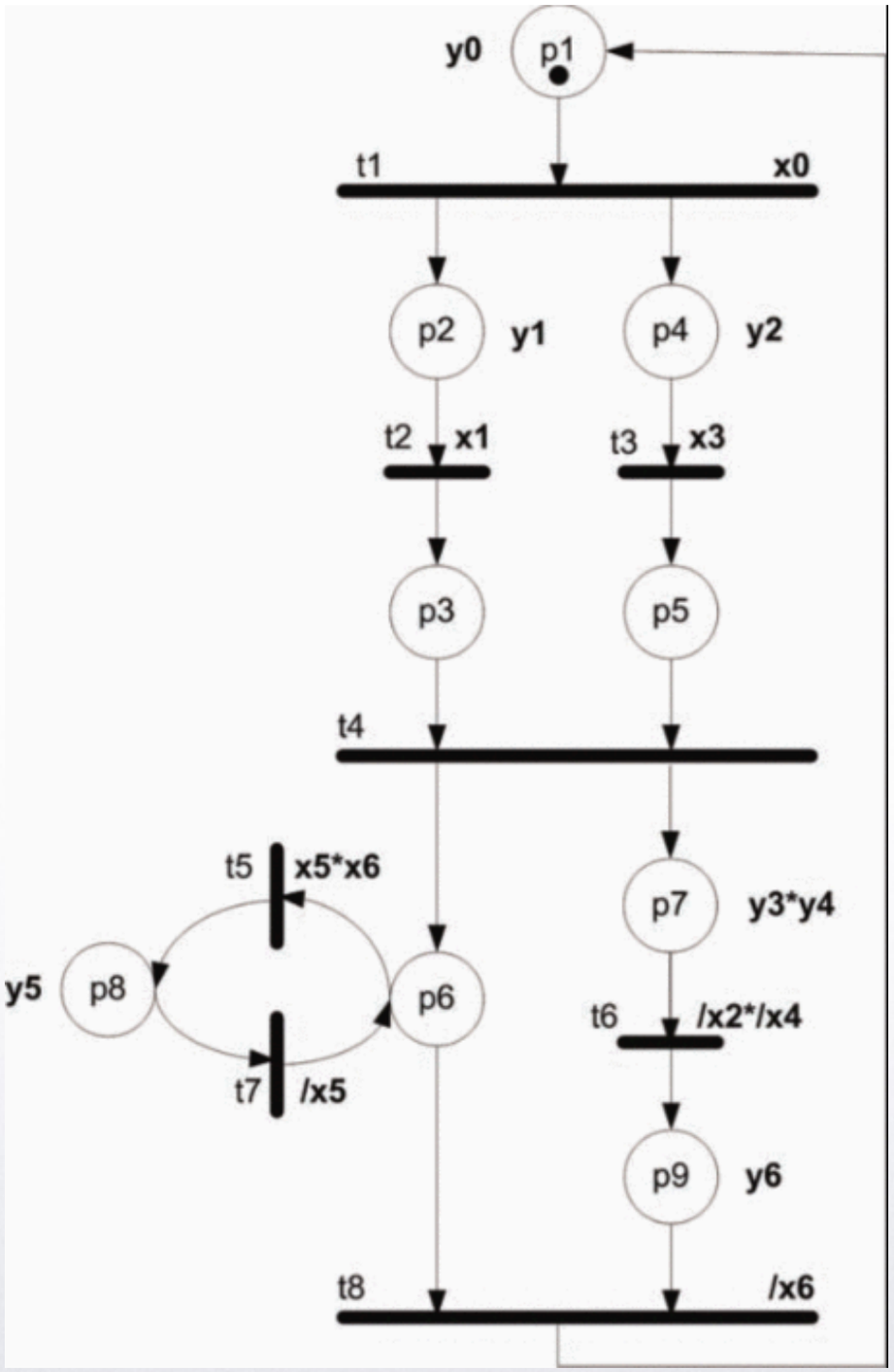
In CTL*, the temporal operators can be freely mixed. In CTL, the operator must always be grouped in two: one path operator followed by a state operator. See the examples below. CTL* is strictly more expressive than CTL.


x0
y0
y1
y2
y5
x1
x3
x2
x4
The first tank
The second tank
y3
y4
x5
x6
y6

	Signal	Meaning
inputs	x0	Signal to start the process
	x1, x2	Fluid level in the first tank
	x3, x4	Fluid level in the second tank
	x5, x6	Fluid level in the third (big) tank
outputs	y0	Ready signal
	y1, y2	Controll the engine of pumps for the first / second tank
	y3, y4	Controll the valves from the first / second tank
	y5	Controll the engine of agitator
	y6	Controll the valve from the third (big) tank



	Signal	Meaning
inputs	x0	Signal to start the process
	x1, x2	Fluid level in the first tank
	x3, x4	Fluid level in the second tank
	x5, x6	Fluid level in the third (big) tank
outputs	y0	Ready signal
	y1, y2	Controll the engine of pumps for the first / second tank
	y3, y4	Controll the valves from the first / second tank
	y5	Controll the engine of agitator
	y6	Controll the valve from the third (big) tank


y0
p1
t1
x0
p2
y1
p4
y2
t2
x1
t3
x3
p3
p5
t4
t5
x5*x6
p7
y3*y4
y5
p8
p6
t6
/x2*/x4
t7
/x5
p9
y6
t8
/x6

A verificação formal consiste em seguir a rede prevendo os estados sucessores e suas condições de habilitação de acordo com os valores das variáveis de input e output.

The syntax of the logic Timed Computation Tree Logic is defined as follows:

$$\varphi ::= a \mid g \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathbf{E}(\varphi_1 \mathbf{U}^J \varphi_2) \mid \mathbf{A}(\varphi_1 \mathbf{U}^J \varphi_2)$$

where:

- a is an atomic action;
- g is a clock constraint;
- E means ”for some path”
- A means ”for all paths”
- J is an interval whose bounds are natural numbers

Com a TCTL é possível trabalhar com sistemas de tempo real, prevendo e modelando o seu comportamento.

Fim

do curso