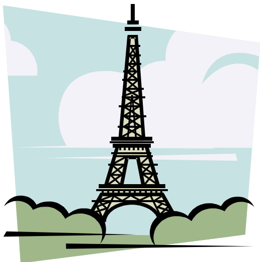
Mathematisch-Geographische Fakultät
Fach Informatik

From Human Knowledge to Process Models

Jörg Desel
Katholische Universität Eichstätt-Ingolstadt

Based on an invited paper for UNISCOM 2008, April 2008, Klagenfurt

Paris, 15/02/08



Outline

The general setting

Modularity

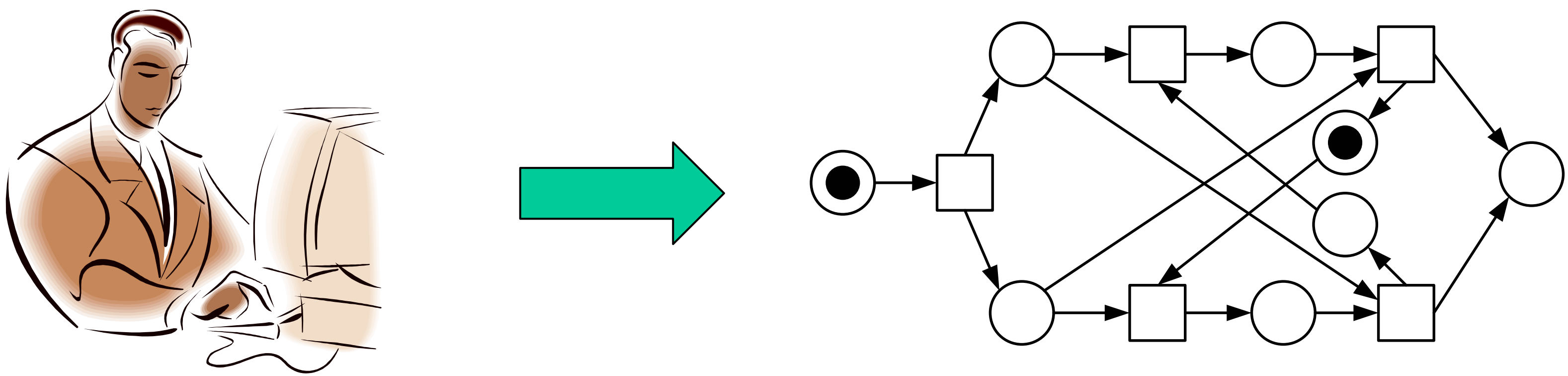
Synthesis

The complete picture

VIPtool

Audi project

Creating process models



Is the process description correct (valid) w.r.t. realitiy or intended reality?

Is the process description correct w.r.t. specified properties?

Are these properties correct (valid)?

Validation and Verification of a system

Validation: Did we build the right system?

Does the system fulfill the purpose for which is was intended?
Which aspects are missing? What is wrong?

Verification: Did we build the system right?

Automated or manual creation of a proof
showing that the system matches its specification.
Which specification is not satisfied? Counterexample?

Evaluation: Is the system useful?

Will it be accepted by the intended user?
Aspects that cannot be formulated in terms of formal specificatio

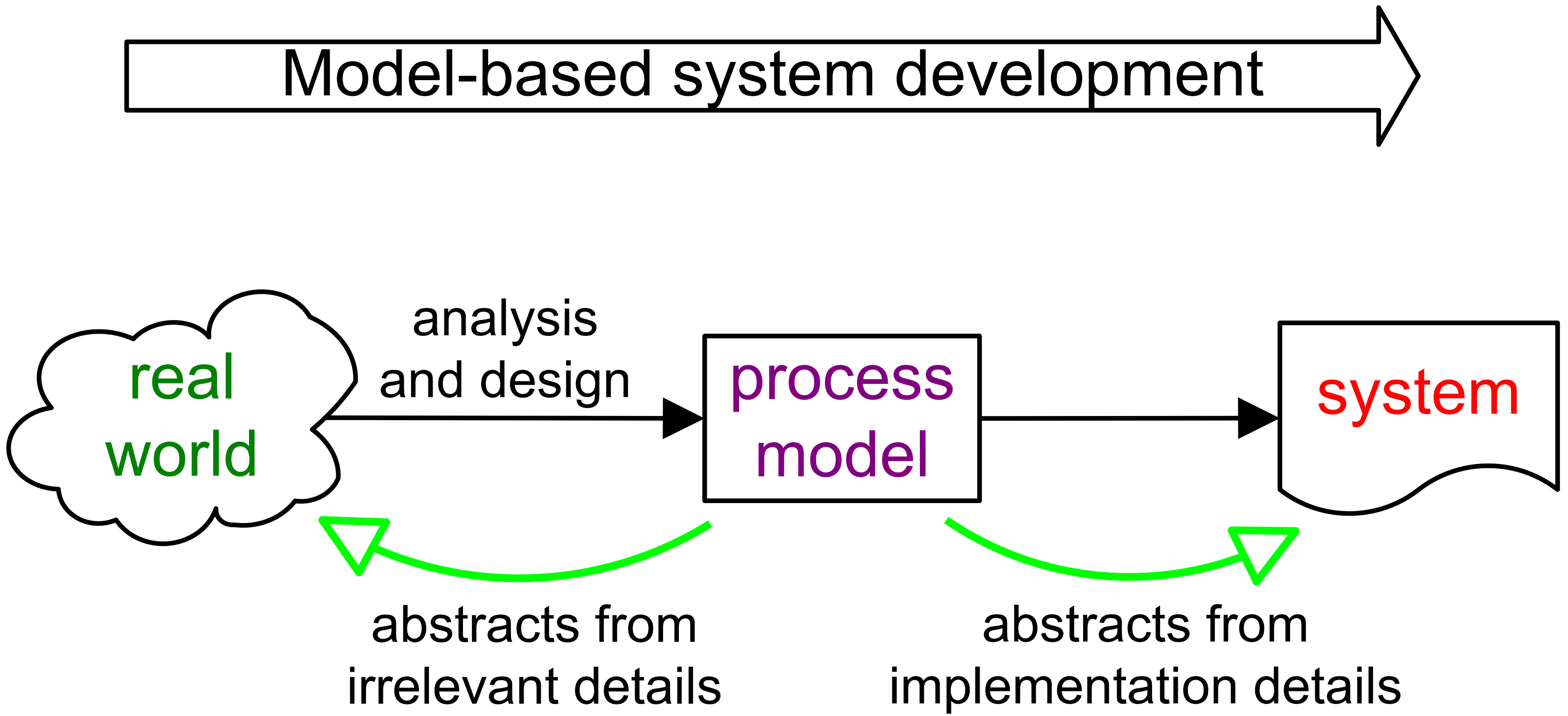
Validation of a process model?

Validation: Did we build the right process model?

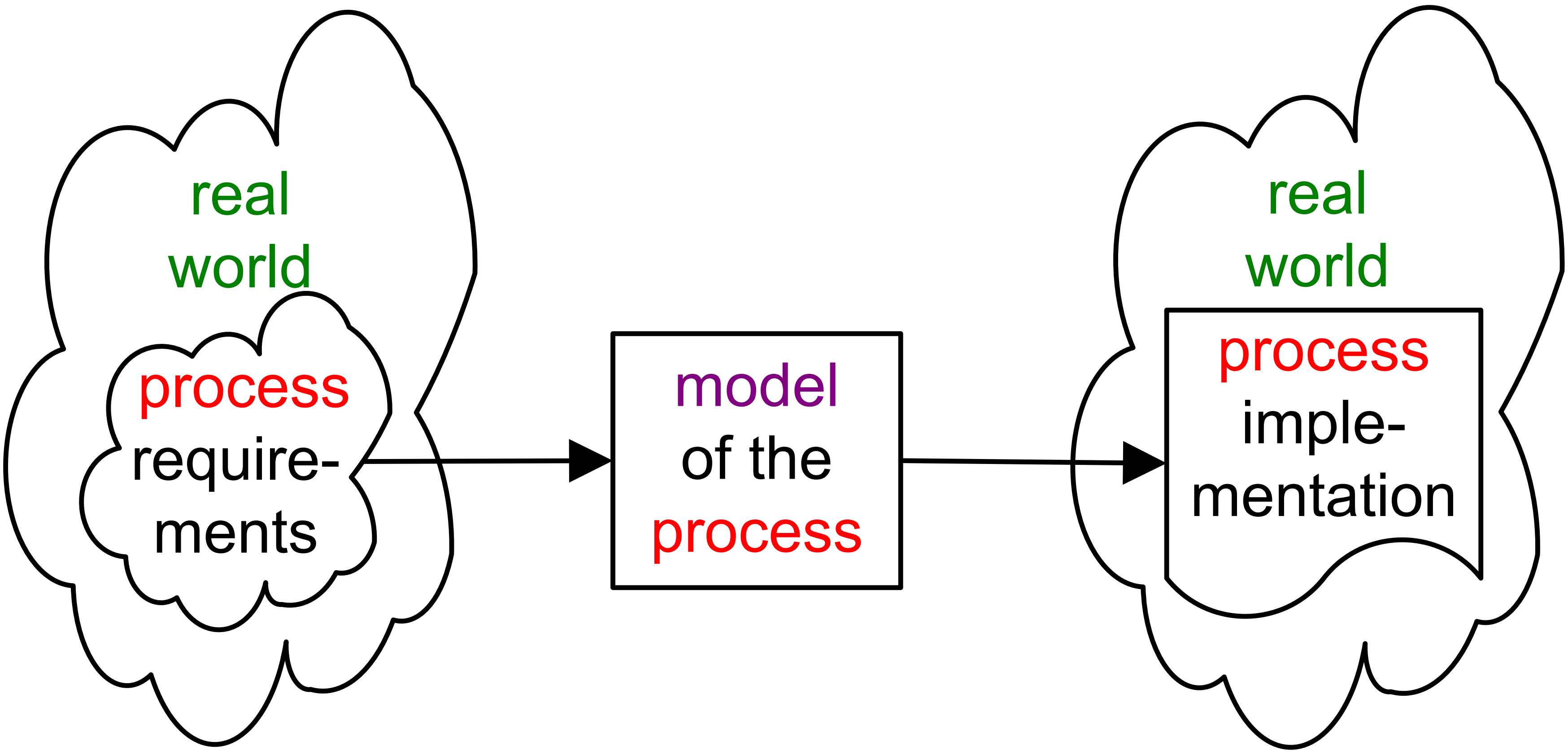
Does the process model fulfill the purpose for which is was intended
Which aspects are missing? What is wrong?

What is the purpose of a process model in system development?

Model-based System Development

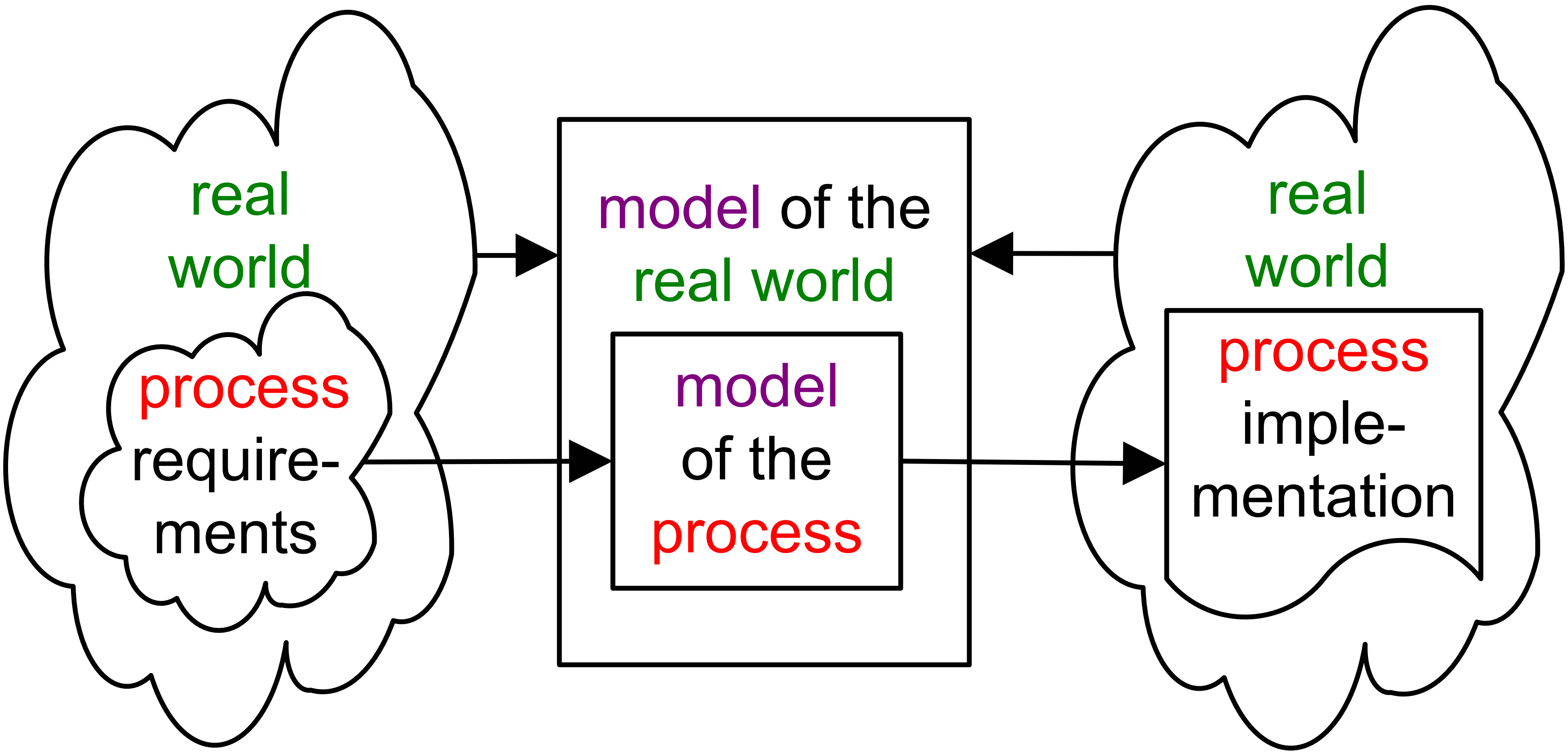


A Process in the Real World

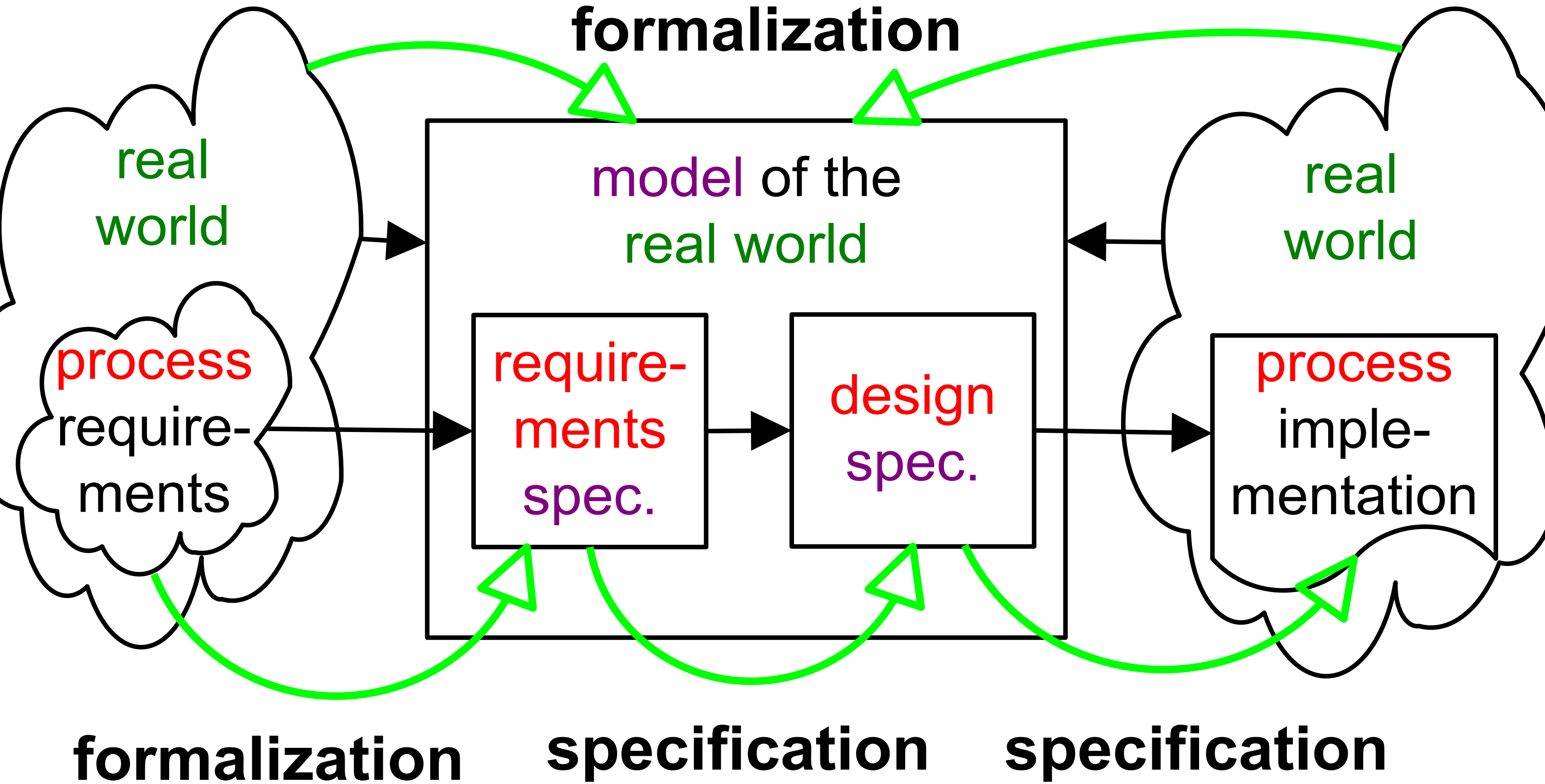


real world:
environment / assumptions on the environment /

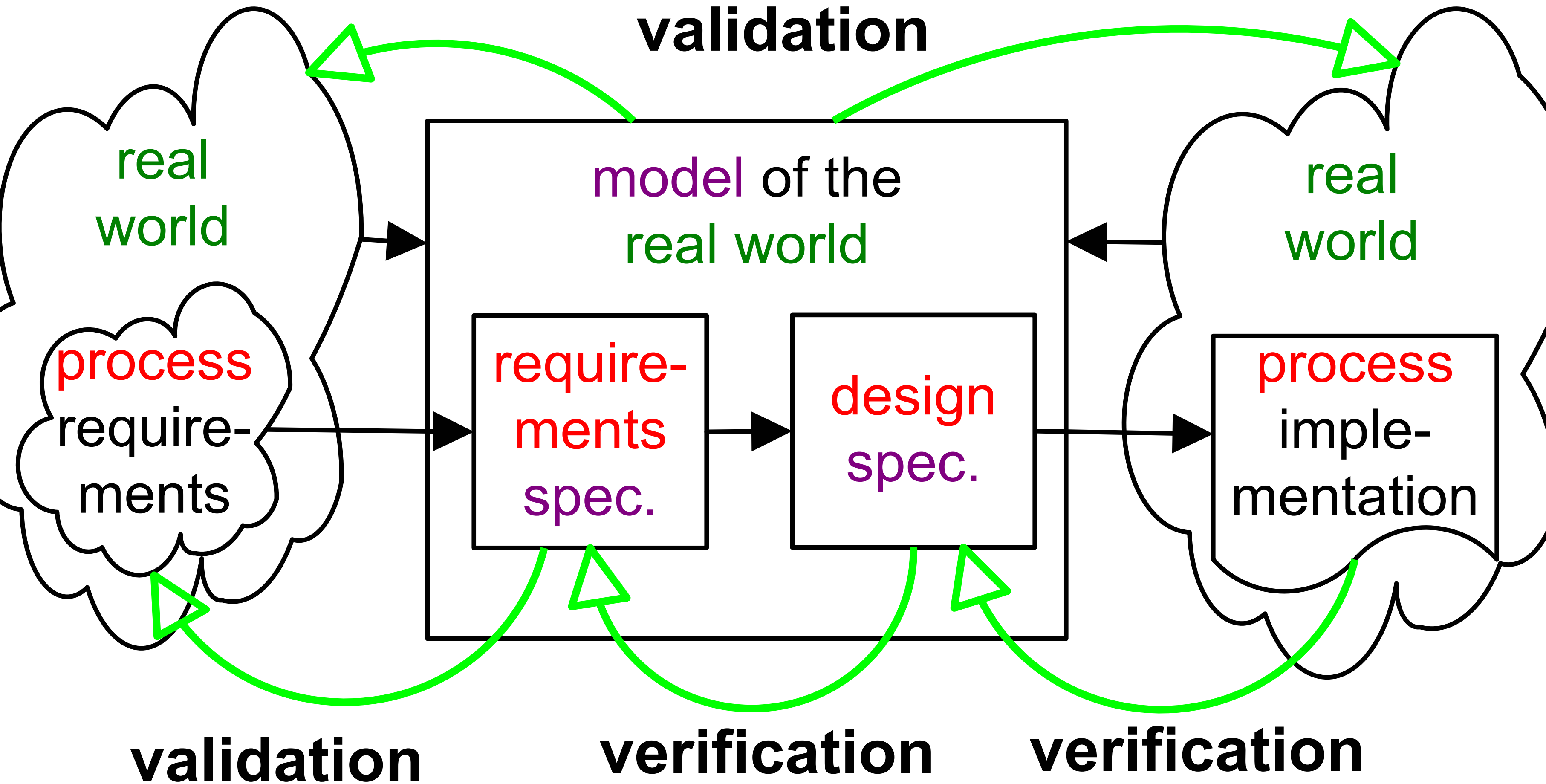
A Process Model in the Real World´s Model



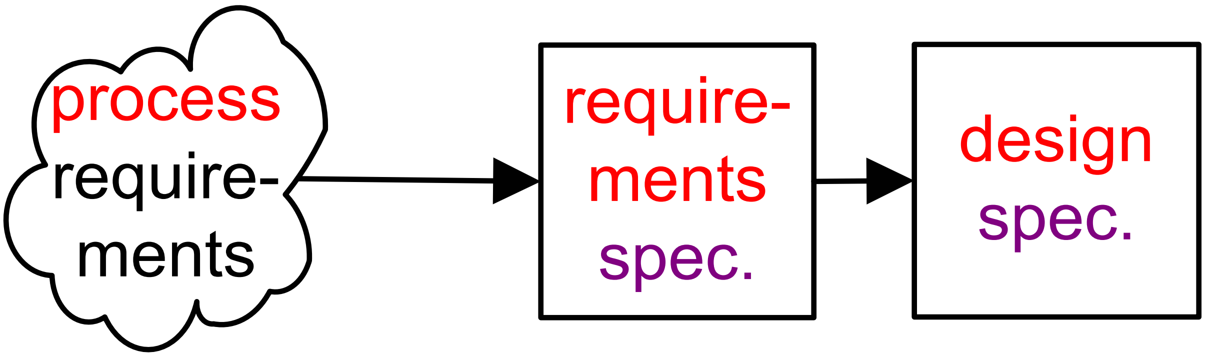
Splitting the Process Model

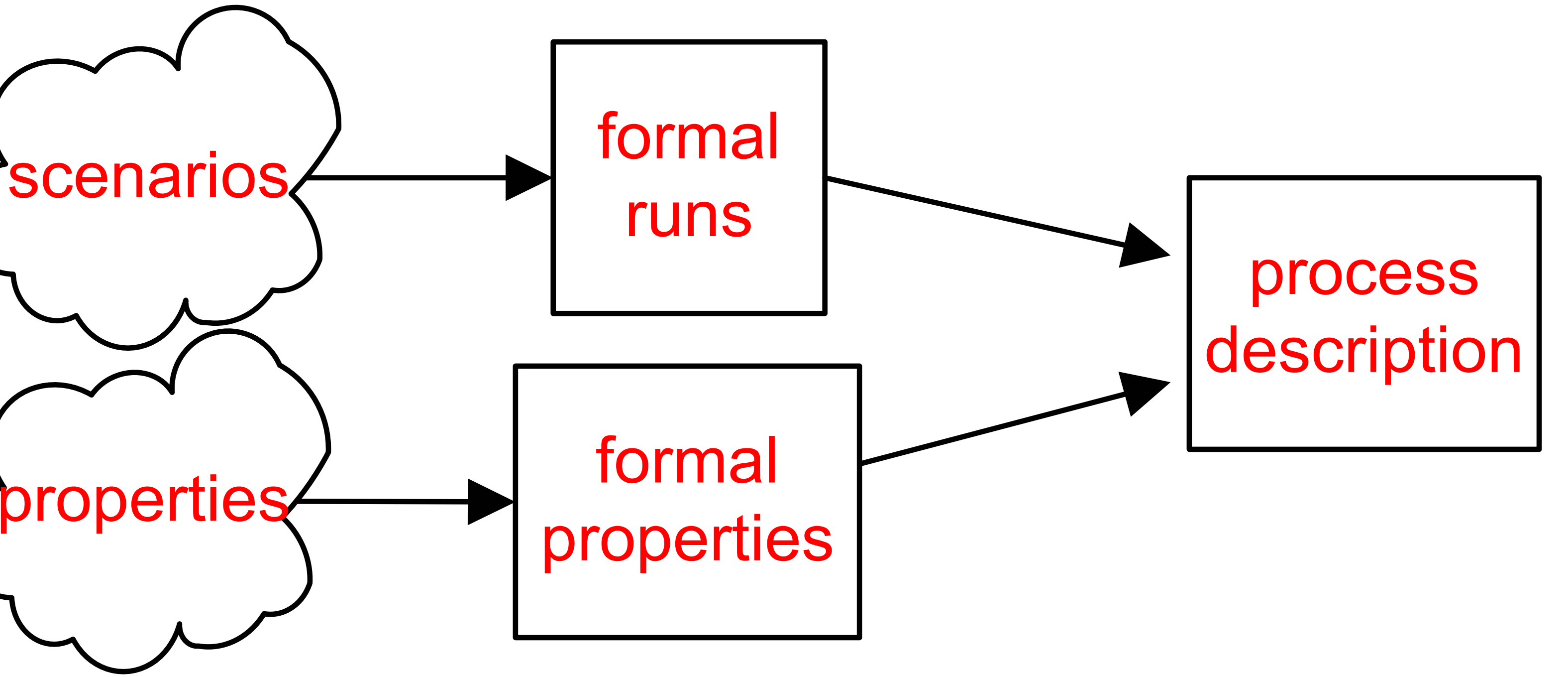


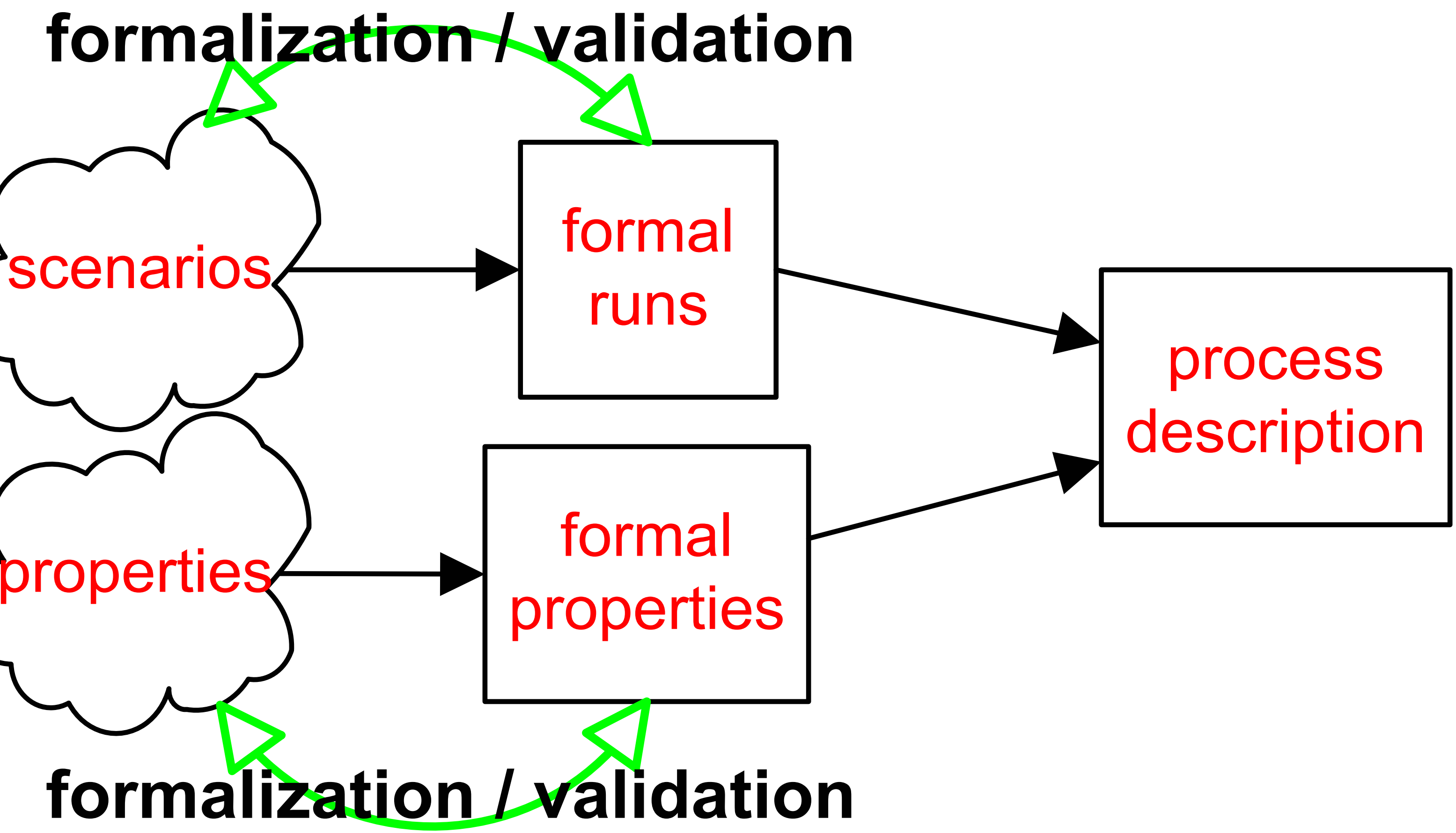
The Reverse of Formalization?

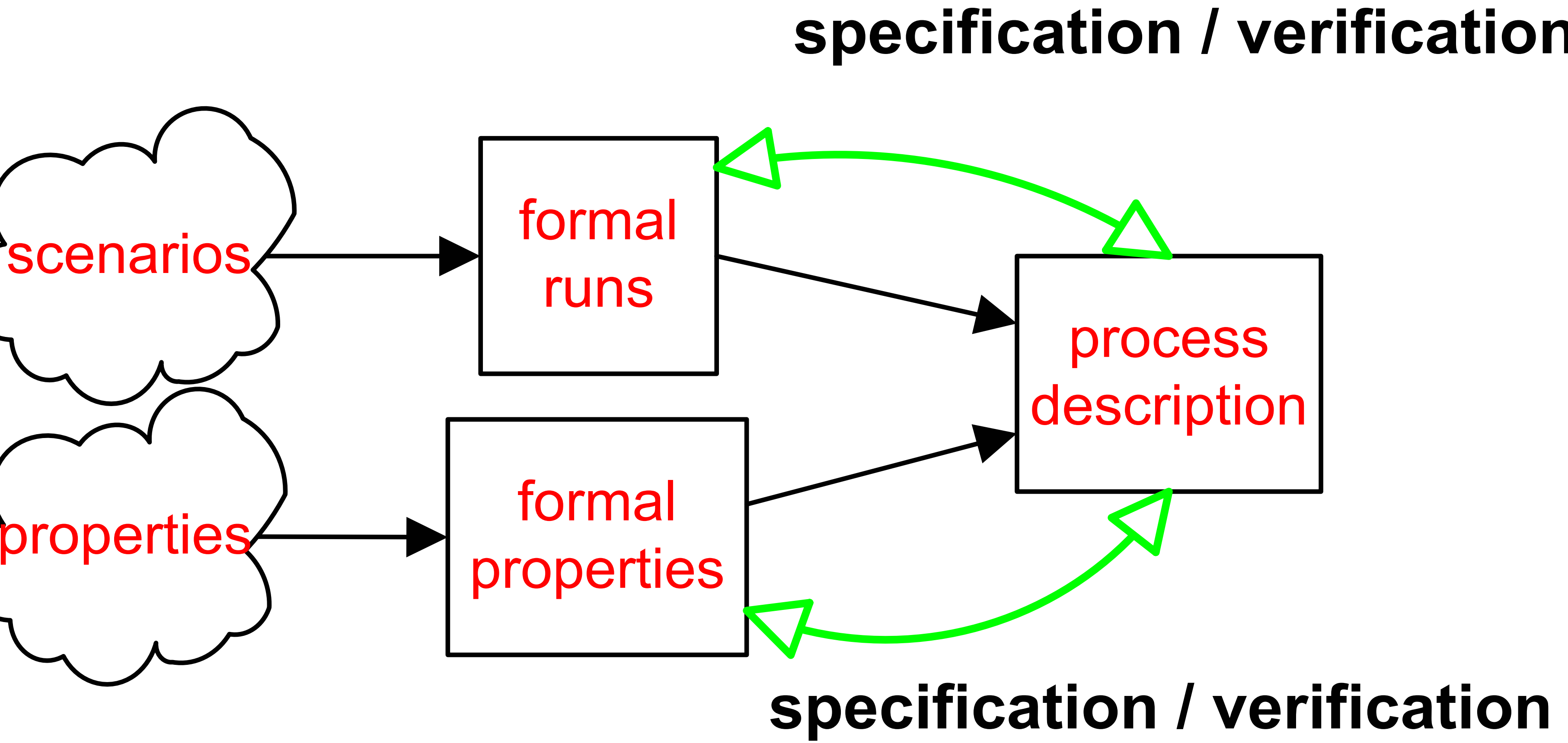


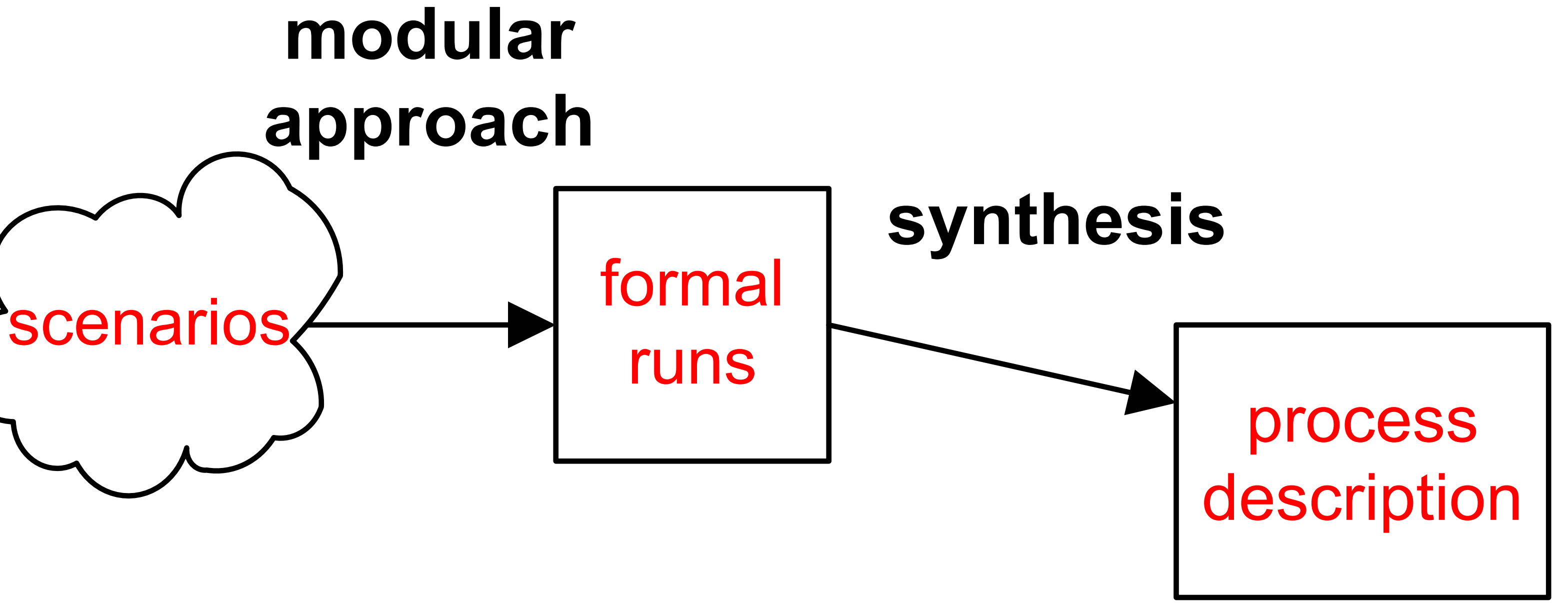
The Reverse of Formalization?




scenarios
properties
formal runs
formal properties
process description


formalization / validation
scenarios
formal runs
process description
properties
formal properties
formalization / validation


specification / verification
scenarios
formal runs
process description
properties
formal properties
specification / verification


modular
approach
scenarios
formal
runs
synthesis
process
description

Explanations

Scenarios: single runs
no alternatives, no IF-THEN-ELSE
instance level

Formal runs: labeled partial orders of events
why partially ordered?
- more natural for processes
- vertical composition (this talk)
- horizontal composition (Paris)

Process descriptions: Petri nets
processes and process modules

Synthesis: work done in Eichstätt

Modularity

x and y occur concurrently

x consists of a followed by b

y consists of c followed by d

sequential setting

possible runs: xy and yx

x = ab

y = cd

resulting runs: abcd, cdab

concurrent setting

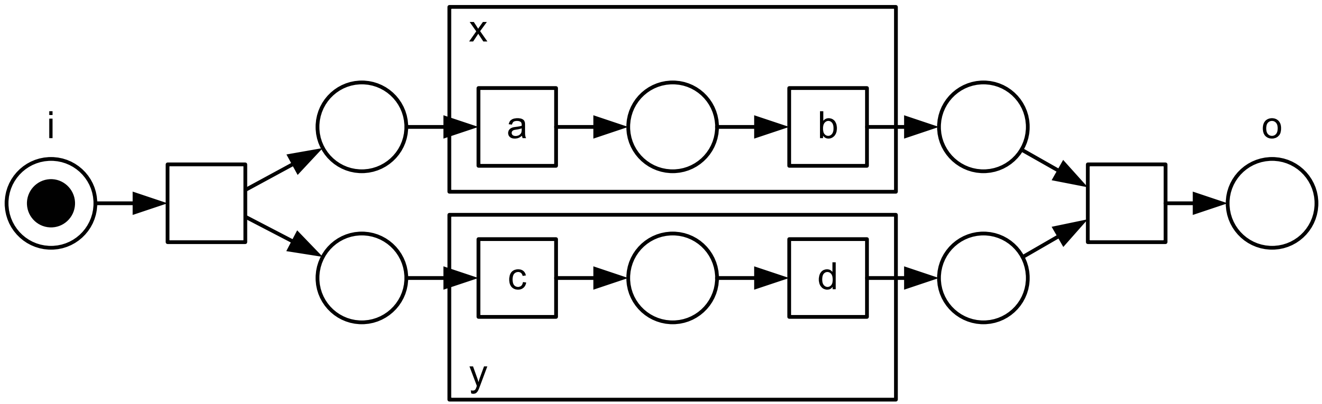
The only possible run: x || y

x = ab

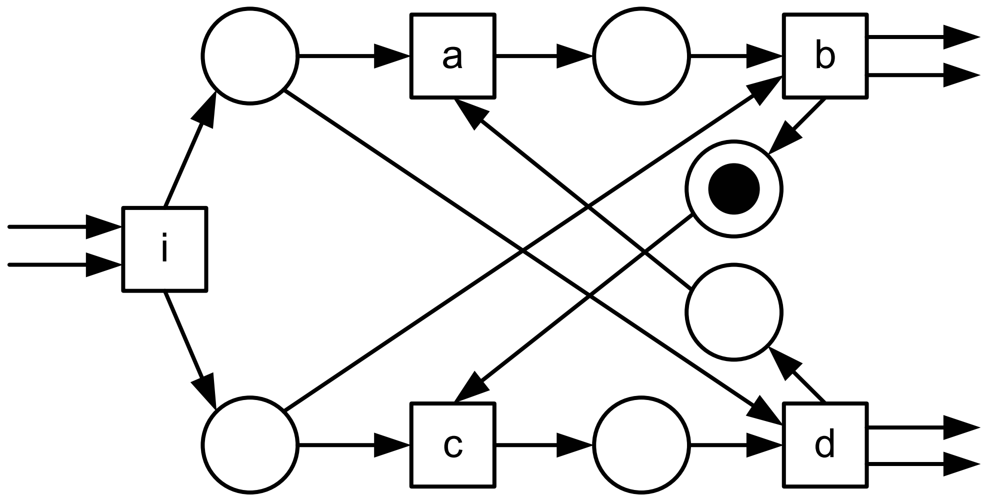
y = cd

resulting run: ab || cd

as a Petri net



Definition of a process

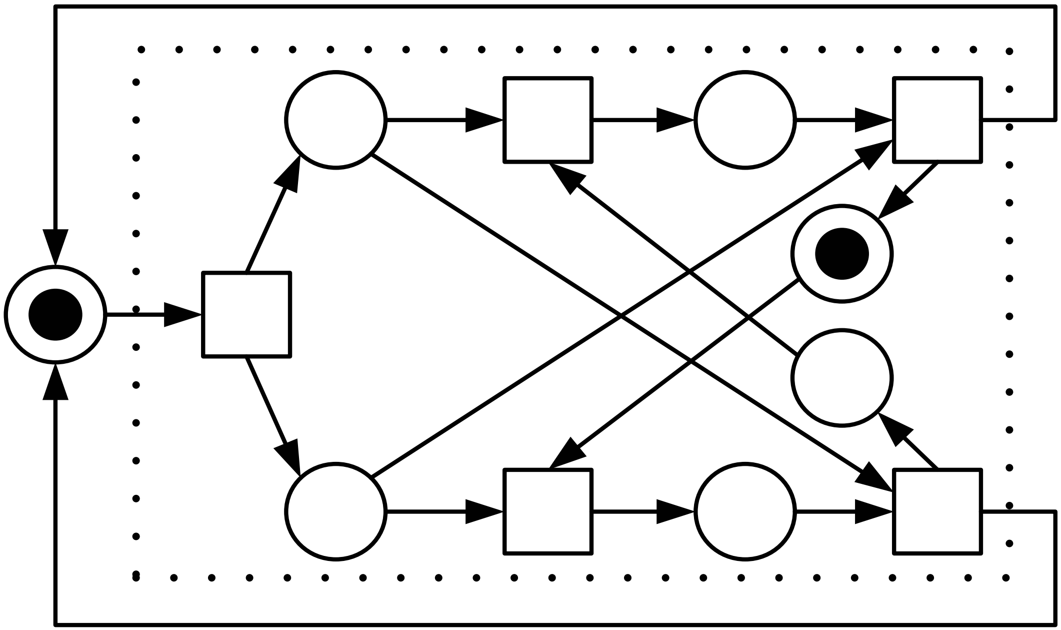


A connected Petri net with two sets of transitions

T_i: input transitions (in the example { i })

T_o: output transitions (in the example { b, d })

safe processes



A process is safe if this net is 1-bounded

$\rightarrow$ transitions of T_i and T_o occur alternatingly

$\rightarrow$ no autoconcurrency

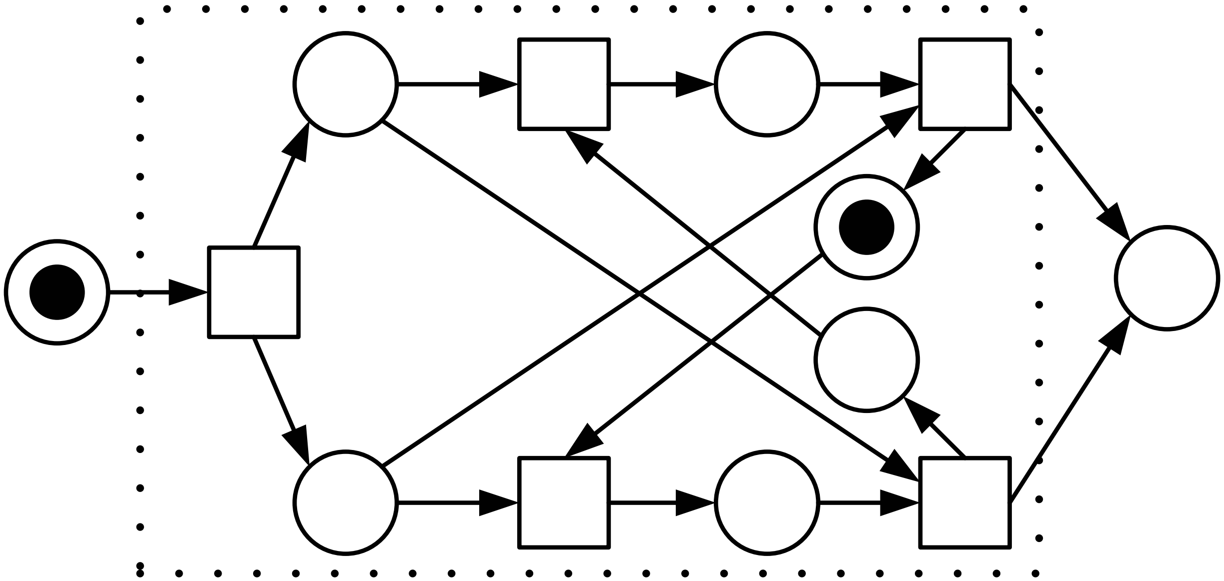
Difference to van der Aalst‘s workflow nets

Workflow nets start with input place and end with output place
(and sometimes have a feedback transition)

Workflow nets start with empty initial marking
(only input place marked) and hence no memory

Sound workflow nets are safe but also live
(with feedback transition)

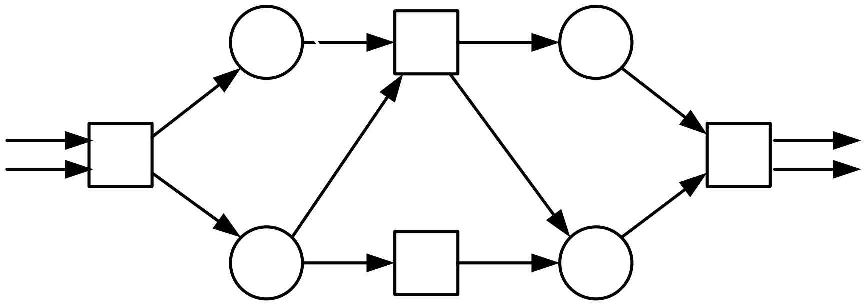
A process viewed as a main process



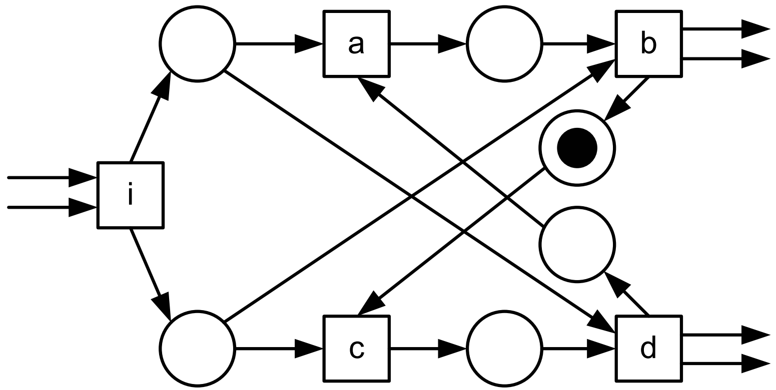
now similar to a workflow net

Refinement

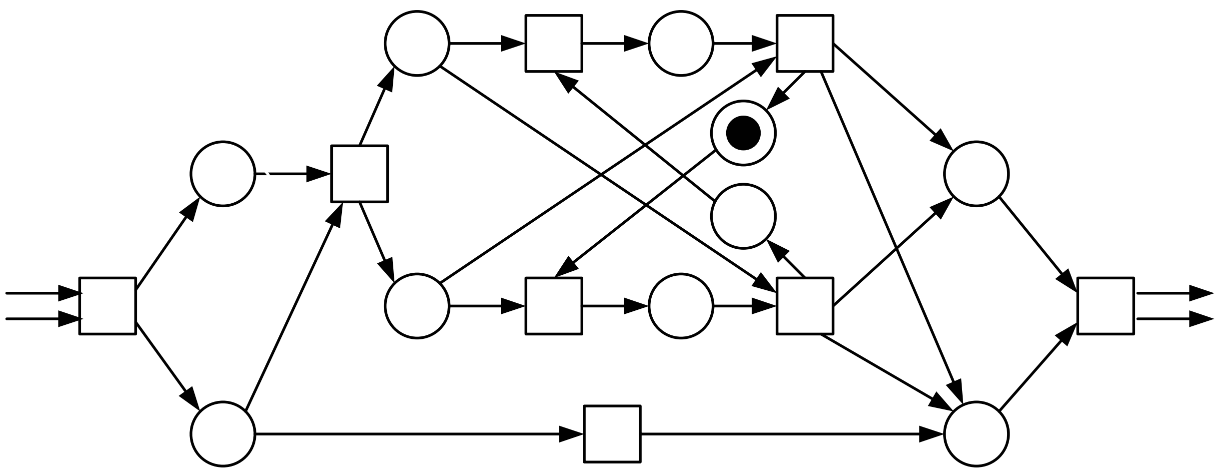
Refinement of t in



by



yields



Observation

We do not distinguish isomorphic processes

Proposition:

The order of refinement does not matter

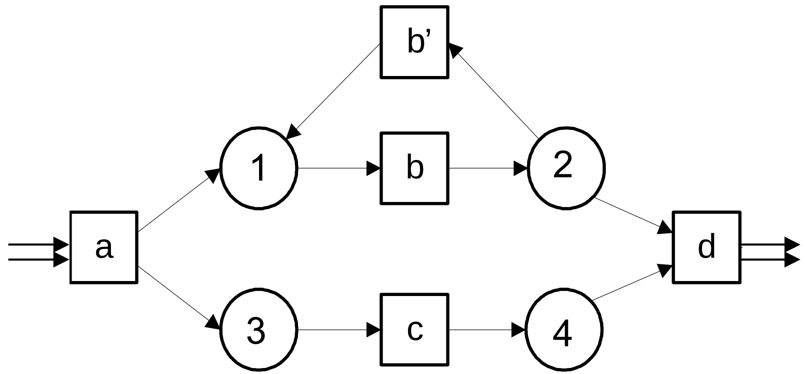
Hence we can do all refinements in one step

But the refined process can have new refinements

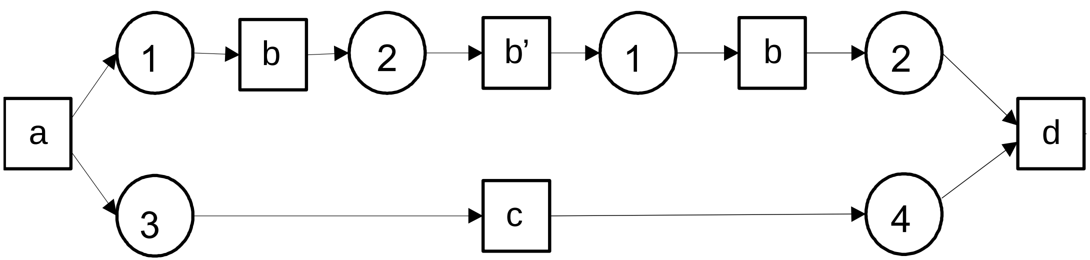
Hence there is a refinement hierarchy

Partially ordered runs

A process



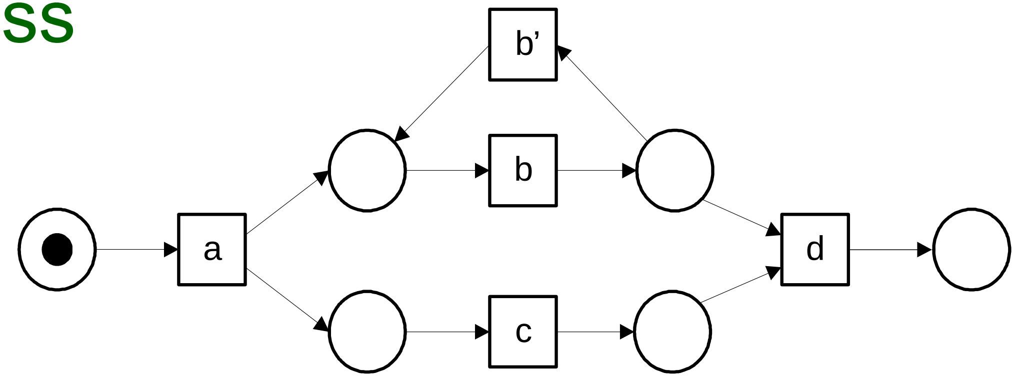
A partially ordered run of the process



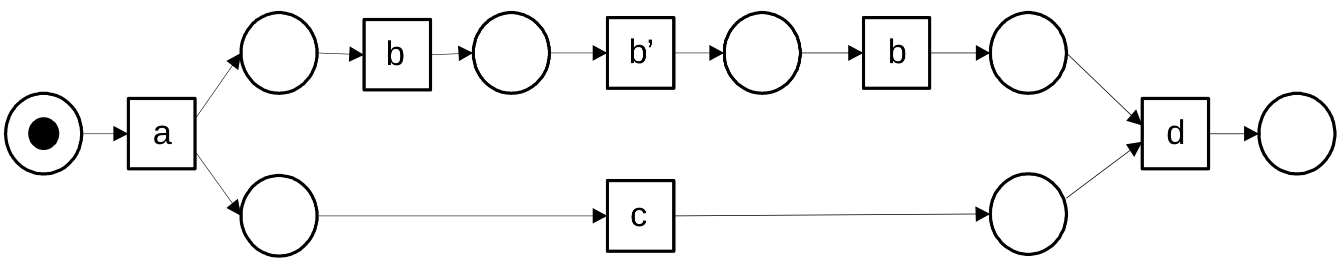
No branching places, no circles,
vicinity of transitions is respected

More convenient view

A process



A partially ordered run of the process



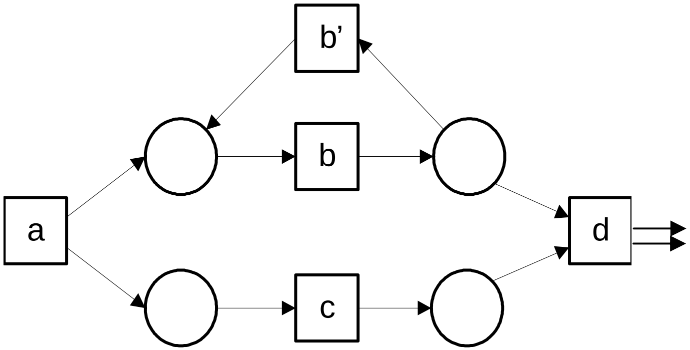
Every occ. seq. of the run is an occ. seq. of the process

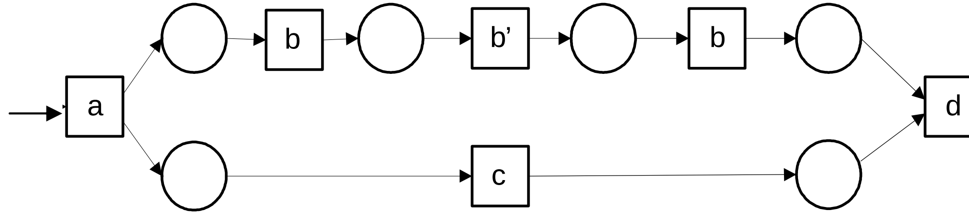
For each occ. seq. of the process there is an according run

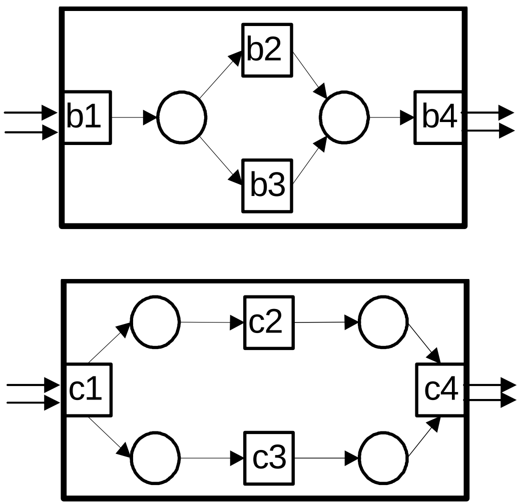
Main argument

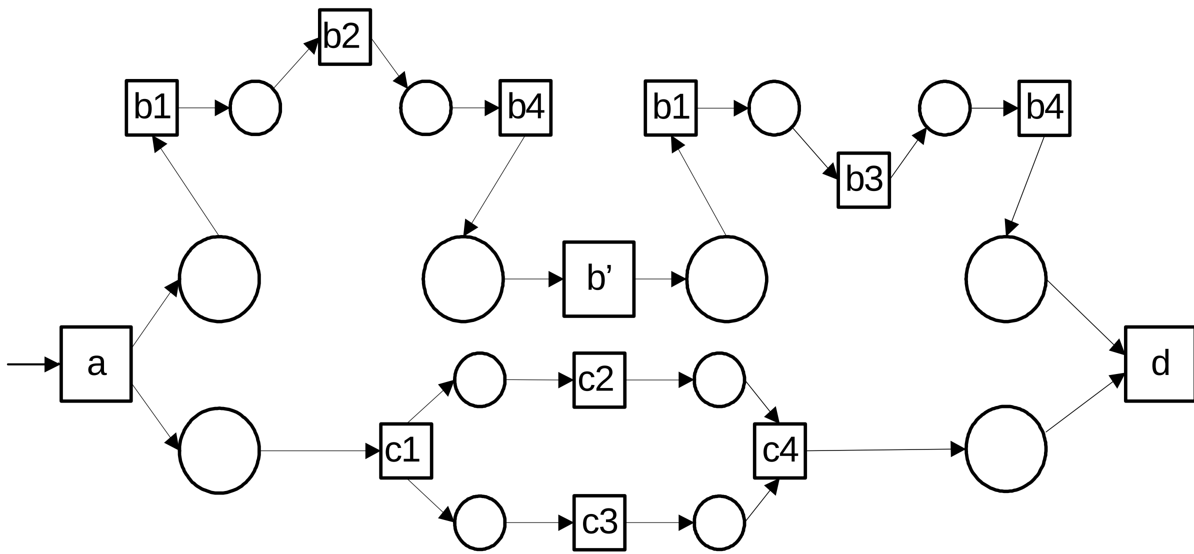
For partially ordered runs, this diagram commutes

$$\begin{tikzcd}
\text{process with transitions } T \arrow[r, "\text{behavior}"] \arrow[d, "\text{efine transitions of } T \text{ o subprocesses with transitions } T'"'] & \text{set of concurrent runs over } T \arrow[d, "\text{refine occurrences of } T \text{ to runs of subprocesses with occurrences of } T'"] \\
\text{process with transitions } T' \arrow[r, "\text{behavior}"] & \text{set of concurrent runs over } T'
\end{tikzcd}$$


b’
b
a
d
c


b
b’
b
a
d
c


b2
b1
b4
b3
c2
c1
c4
c3


b2
b1
b4
b1
b4
b3
b’
a
d
c2
c1
c4
c3

synthesis / process mining

Generate a Petri net from a description of its behavior
(state space, language, partial orders, …)

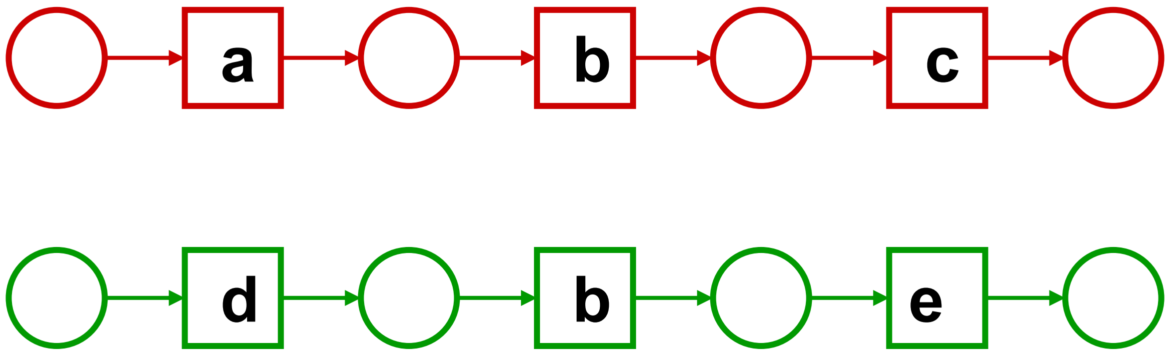
… such that the behavior of the generated net is
- precisely the initial behavior
- little more than the initial behavior

… such that generated Petri net is small / easy to understand

Theory for state spaces developed by Ehrenfeucht / Rozenber
Main player in synthesis theory: Phillippe Darondeau

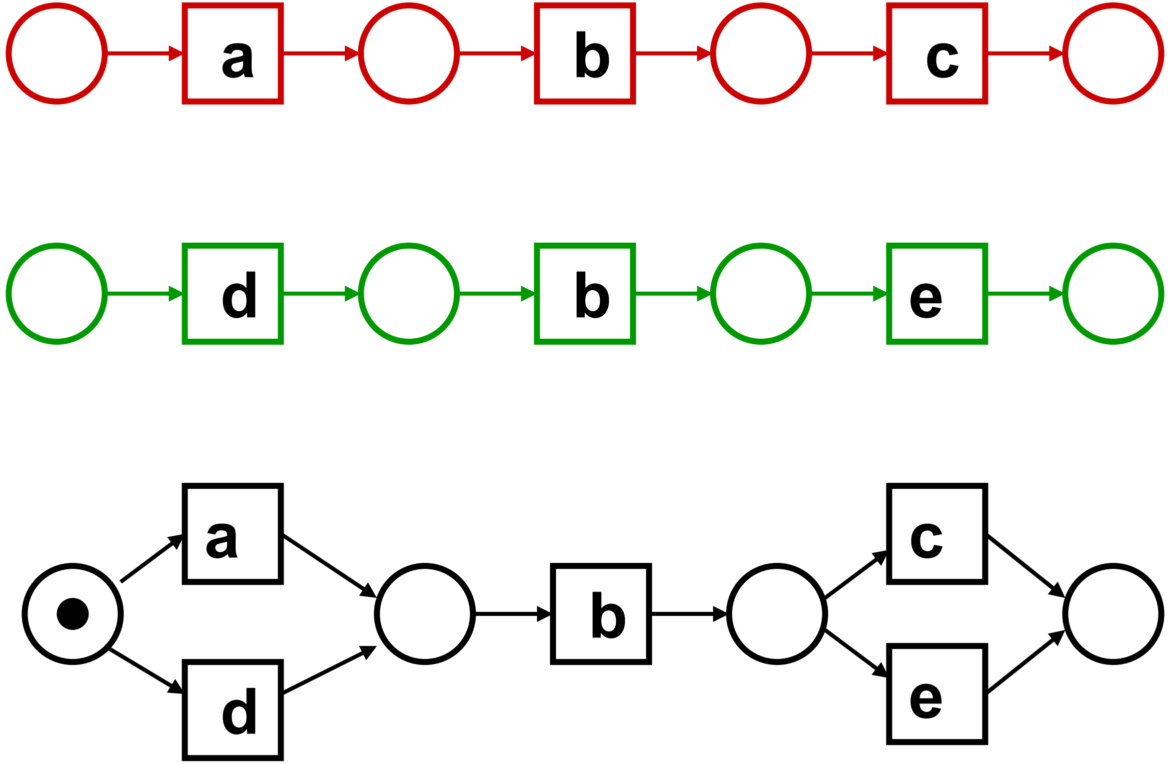
Synthesis from partial languages:
Lorenz / Mauser / Bergenthum / Desel

Synthesis from runs



Synthesis from runs

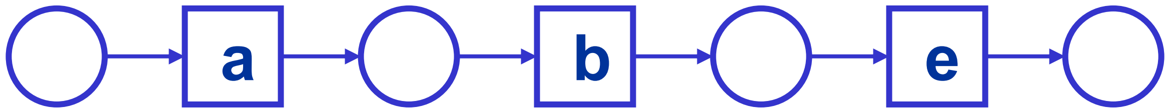
Result obtained by folding

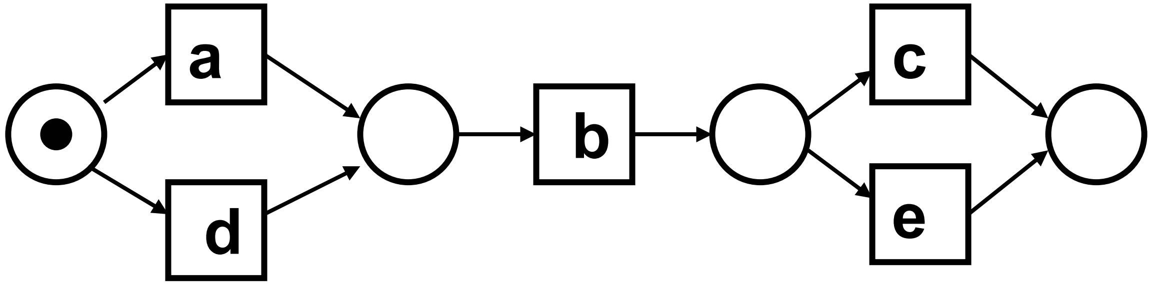


Synthesis from runs

Result obtained by folding

Problem: this process has additional runs, e.g.

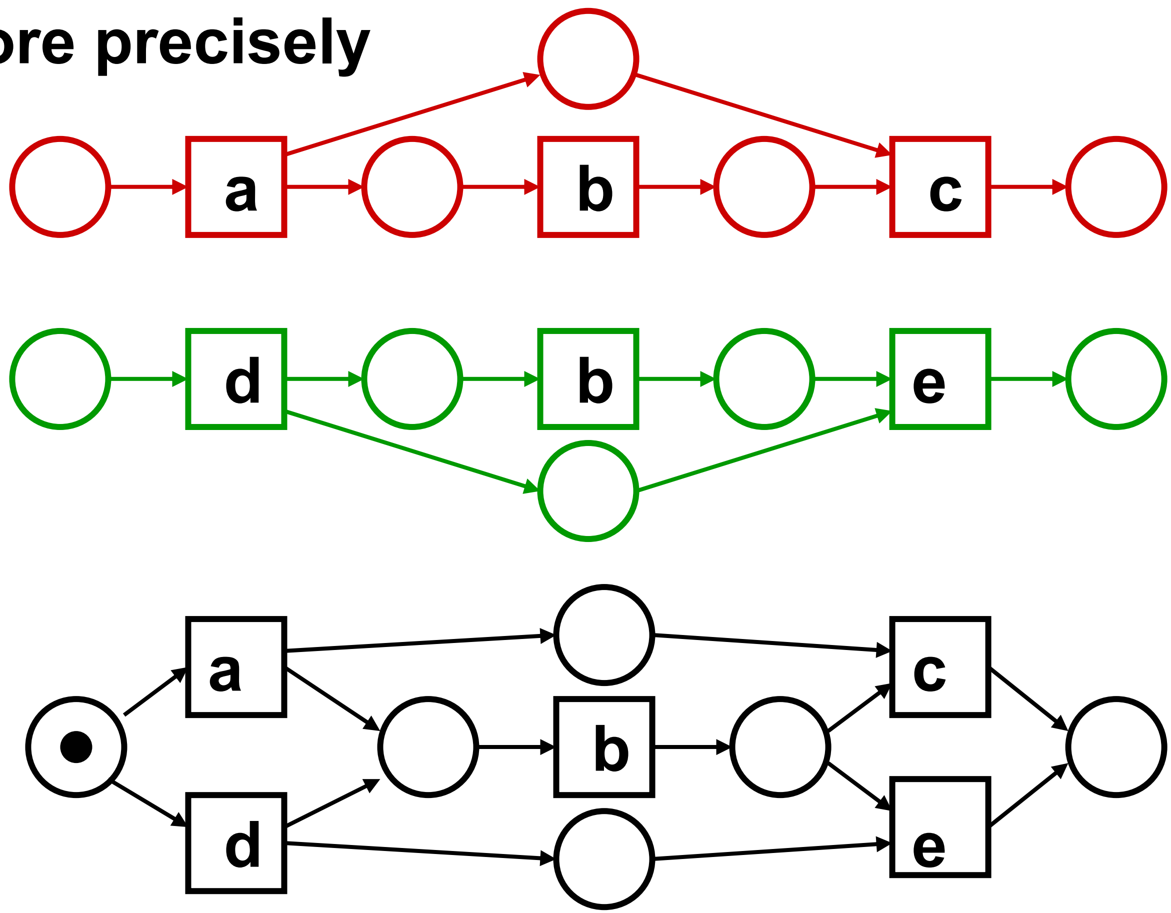




Synthesis from runs

Result obtained by folding

This is either intended or runs have to be specified more precisely



Process construction using synthesis

1. Identify start conditions, start actions and end actions of the process to be defined

2. Let relevant people define runs of the process on an abstract level

3. Agree on the abstract actions that occur in these runs

4. Synthesize a process from the runs

5. Validate this process by construction of runs

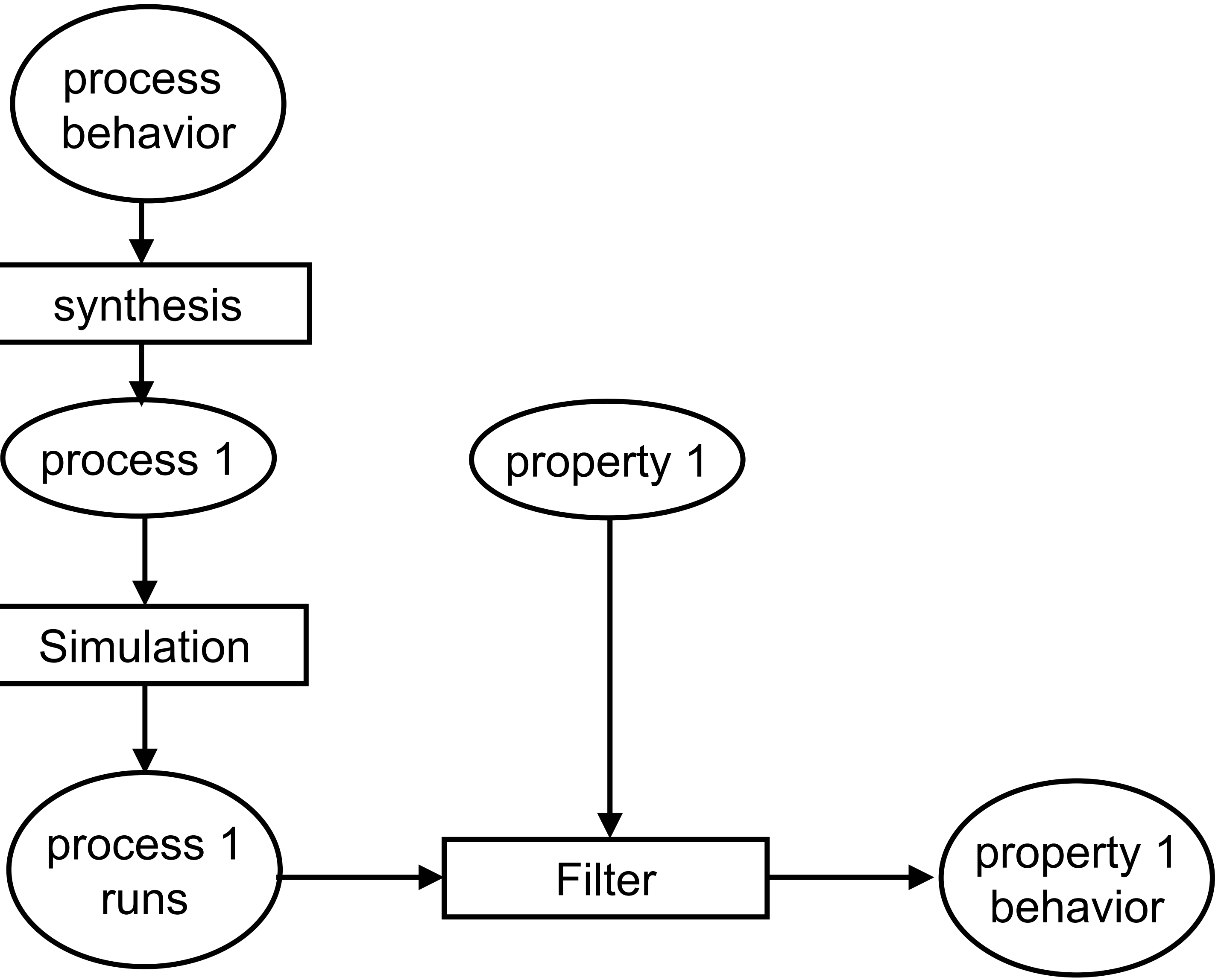
6. If actions that have to be refined then

   Find experts that can provide information (runs) for these actions and continue with 3

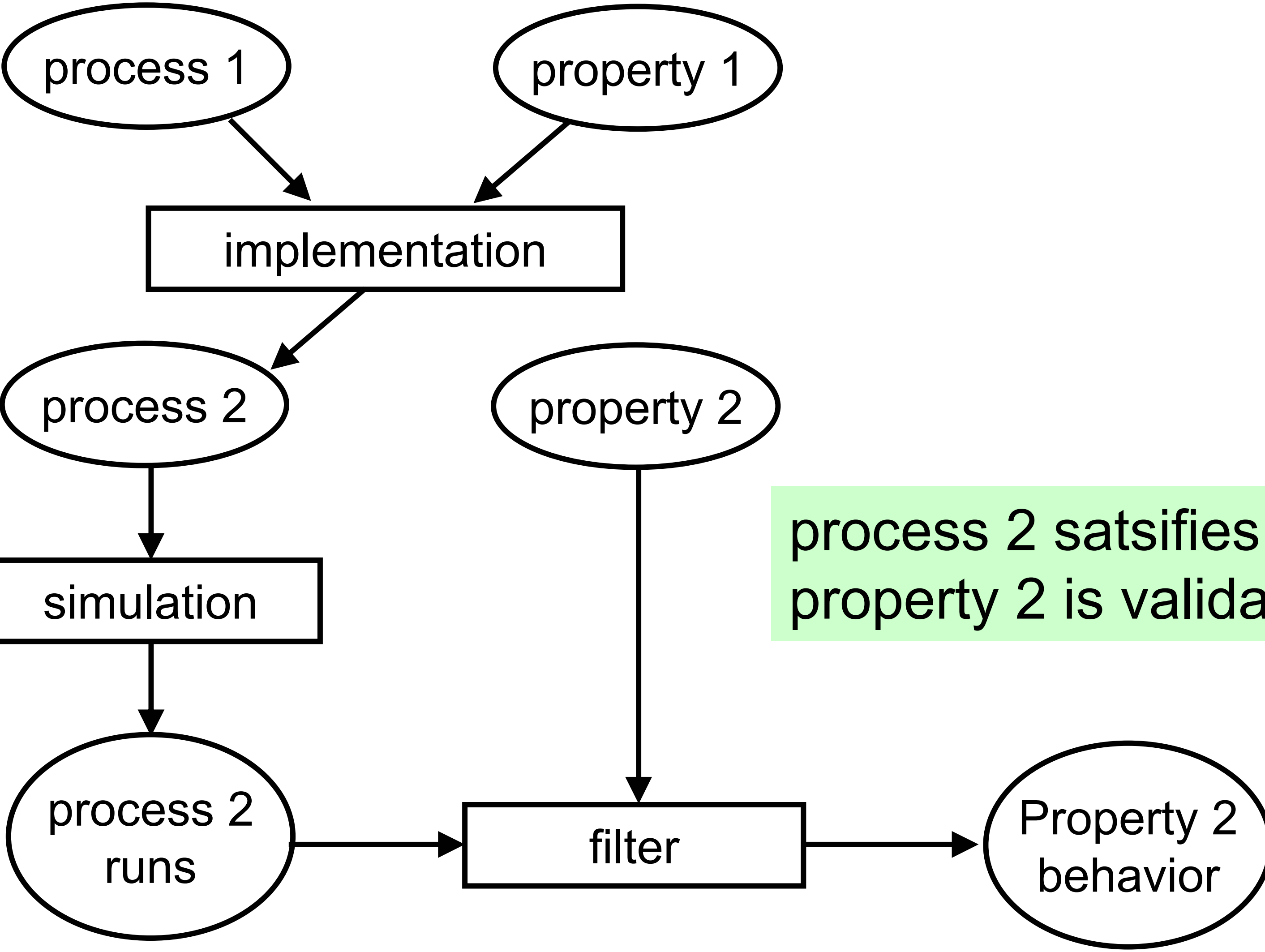
7. Otherwise construct the flat process by repeated refinement of all actions for further analysis

Steps 2-4 described in detail in:
Bergenthum, Desel, Mauser: Synthesis of Petri Nets for Business Process Design, Modellierung 2008, Berlin, 12.-14.March

stepwise validation of processes and properties

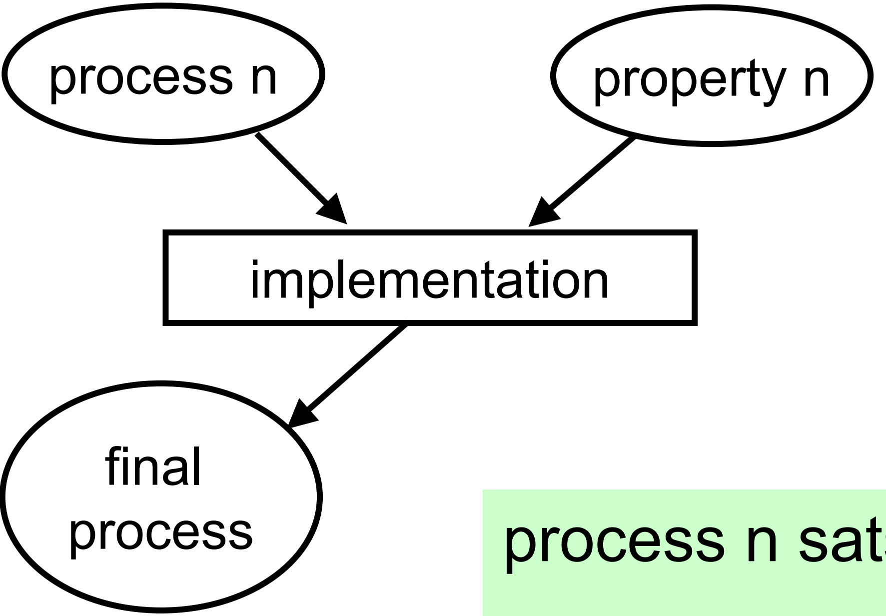


stepwise validation of processes and properties



process 2 satsifies property 1
property 2 is validated.

stepwise validation of processes and properties



process n satsifies properties 1, … , n-1

The final process satisfies all properties

This approach does only work as long as the properties restrict behavior.

VIPtool

Constructs and visualizes partially ordered runs

Allows to define properties graphically (fact transitions etc)

Checks properties on runs

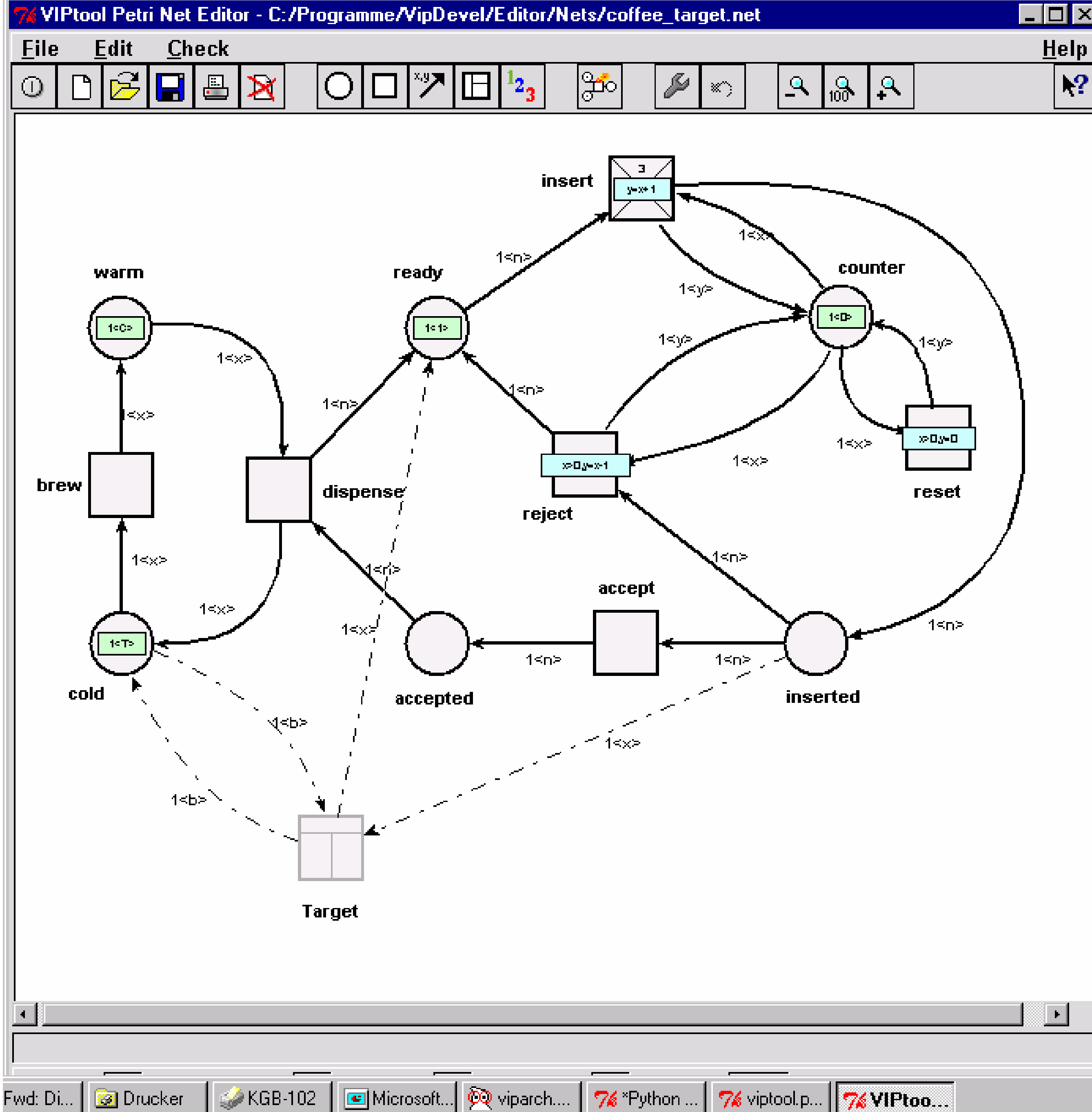
Synthesizes nets from partially ordered runs
(various algorithms)

Will support the entire procedure described before.

VipTool

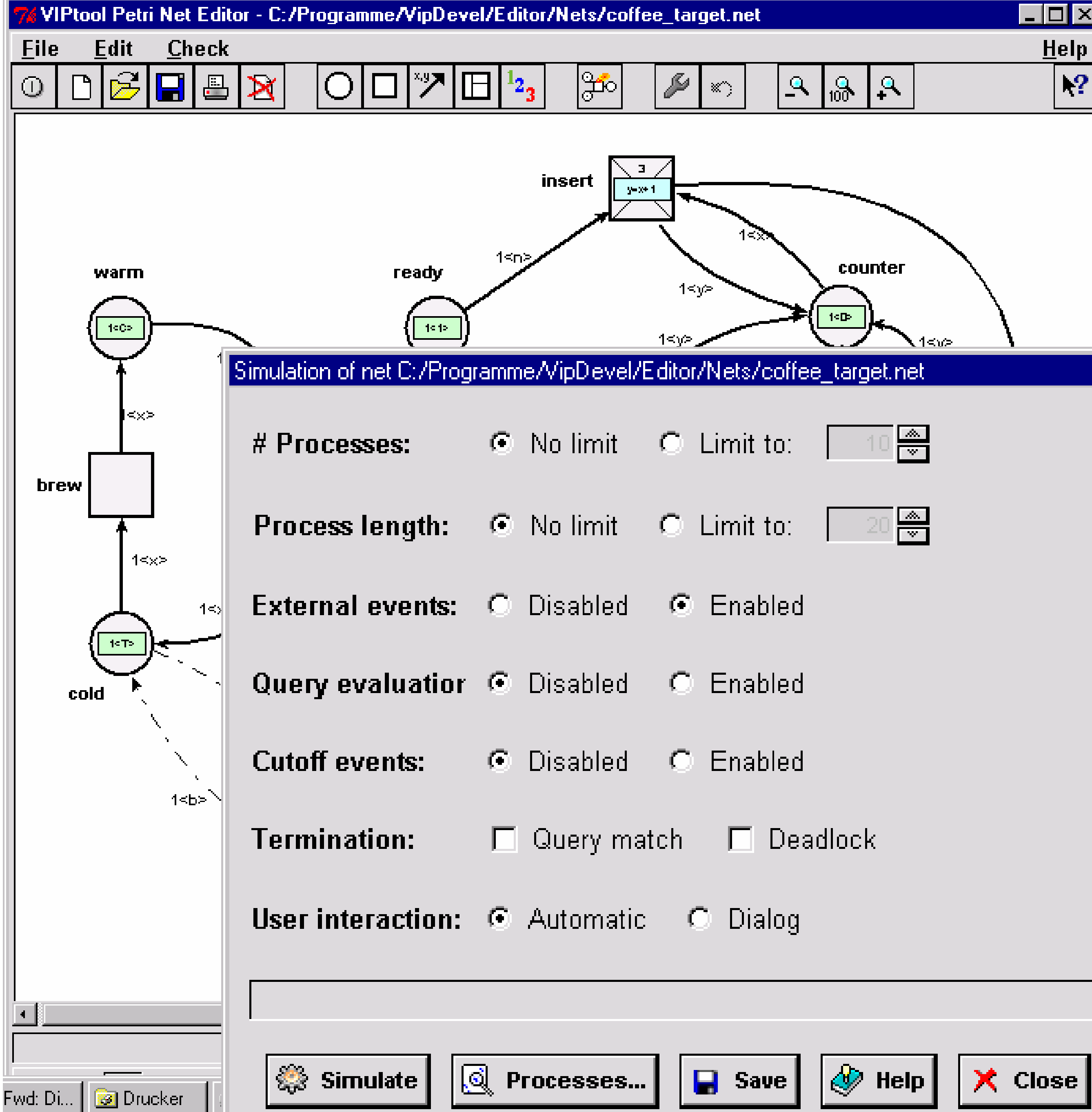
Simulation =

Generation of runs



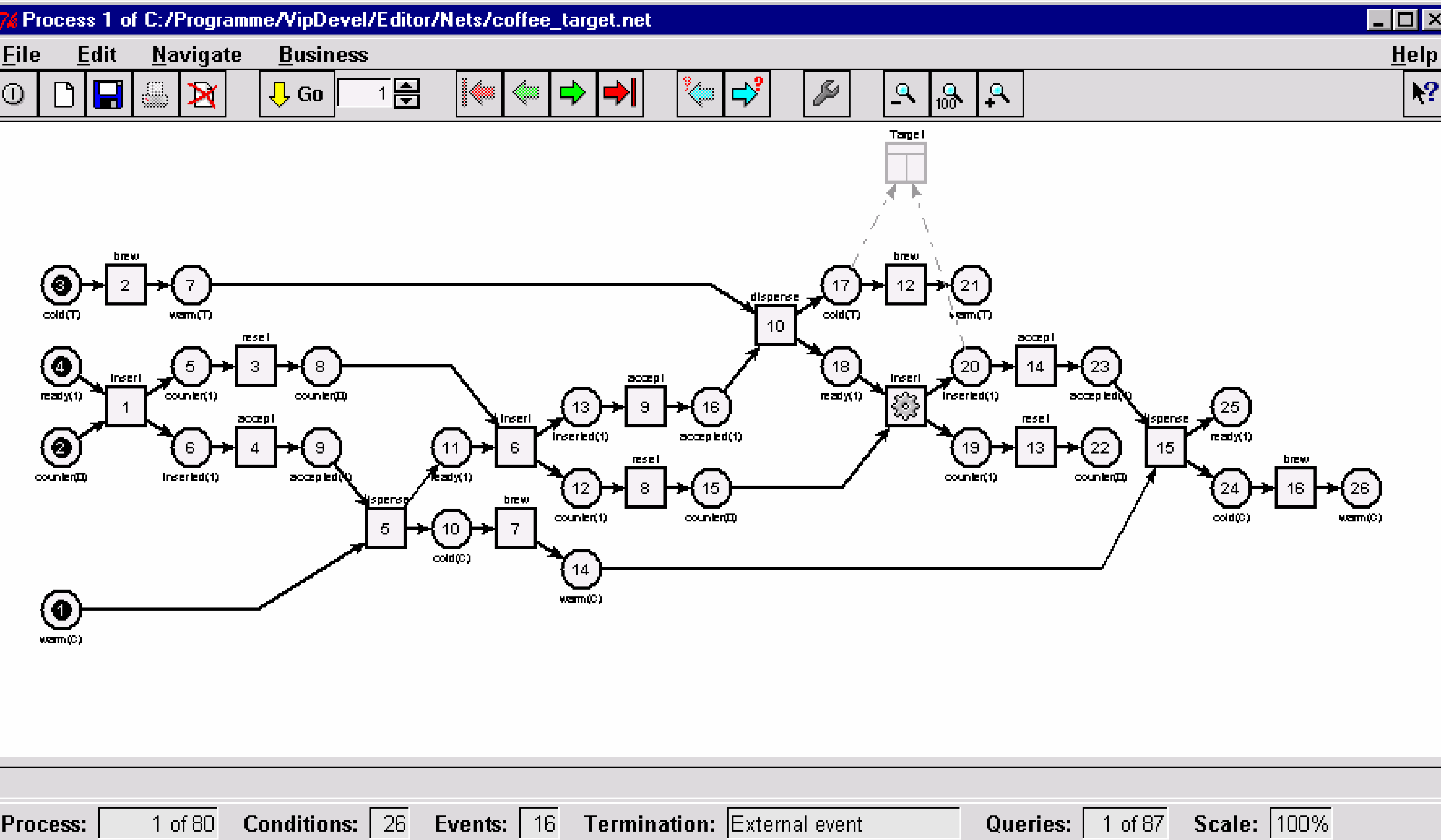
VipTool

control of parameters



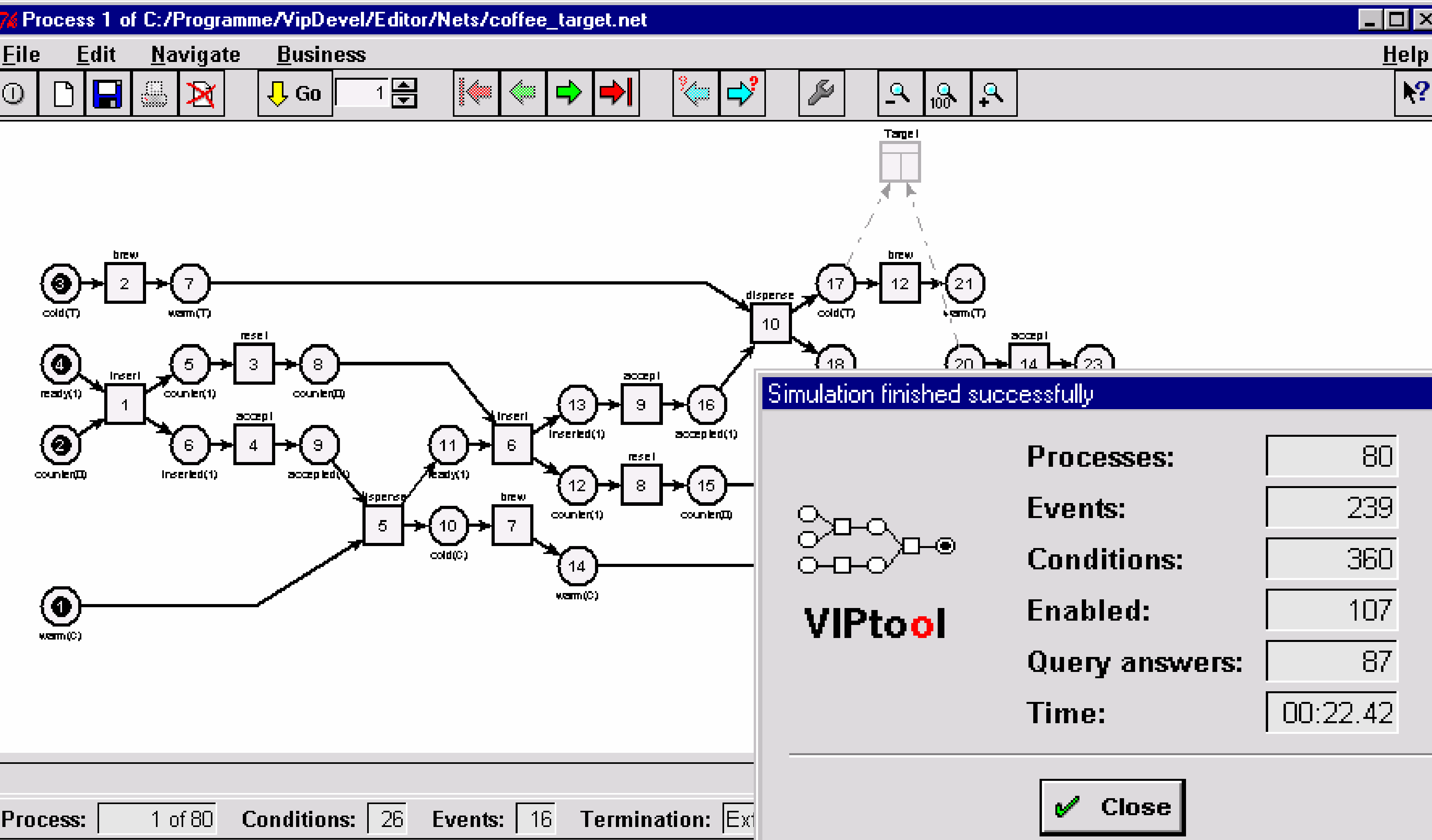
VipTool

A generated and visualized run



VipTool

Analysis of runs



The AUDI project

Audi improves algorithms for refuel identification and updates of the fuel / remaining milige indication

The intended algorithms are given in an informal way, in form of natural-language scenarios.

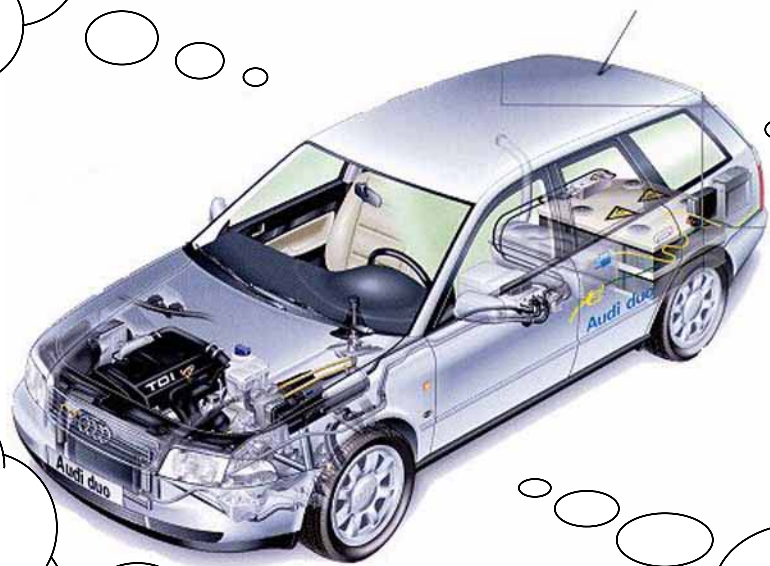
1. Formalize / validate the scenarios and requirements
2. Synthesize a model from these formalized specifications

How much fuel is in the tank?

Did someone refuel?

How much?

How correct are signals from petrol gauge?



Did refueling really happen, or is the car on an aslope road so that it seems that petrol has increased?

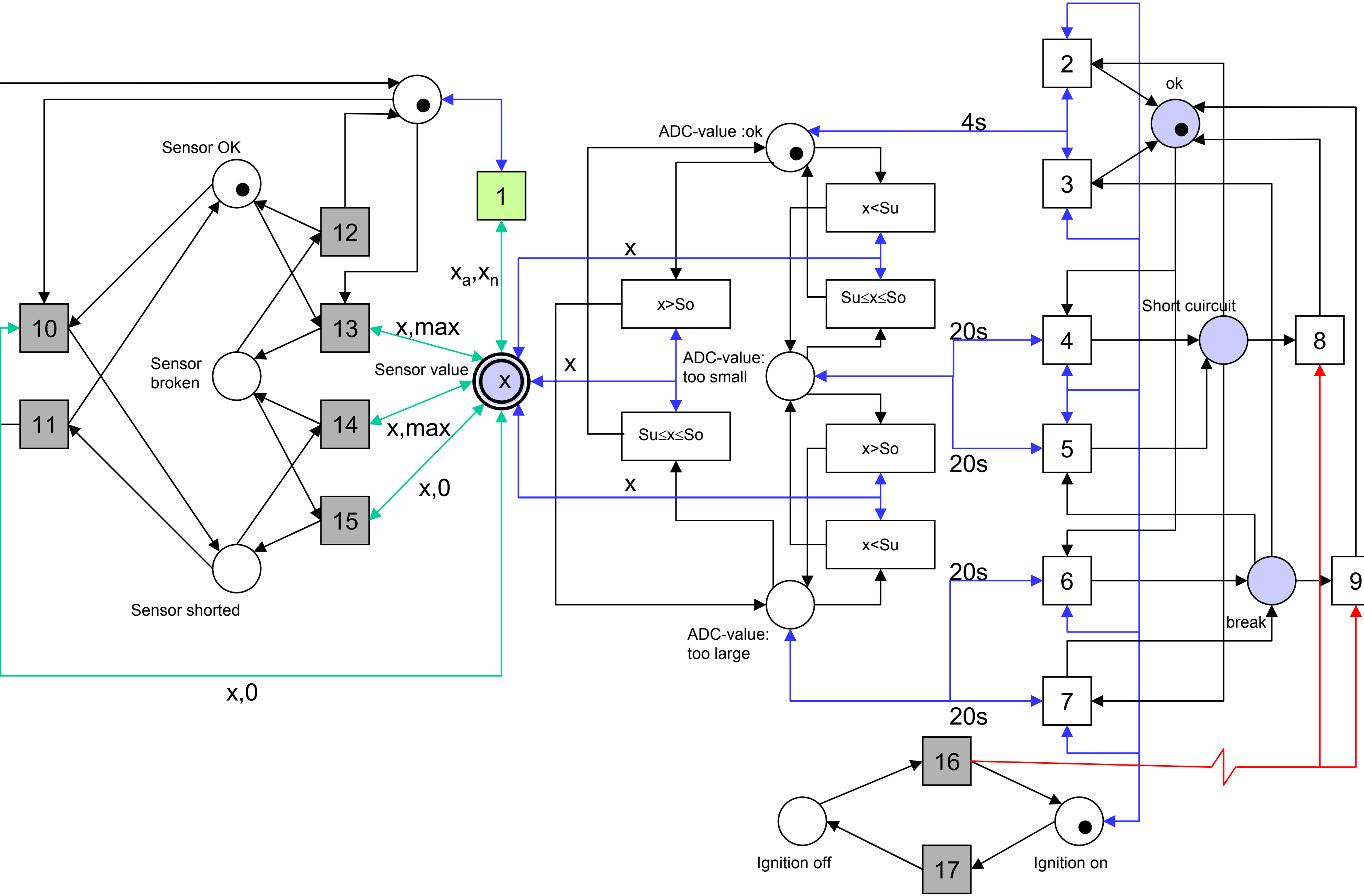
Ignition is still on. Was petrol used (e.g. by auxilliary heating)?

Requirements

A complete refuel identification requires two gaging rounds. At the end of the second round the assumed petrol level is updated, provided refuel was identified.

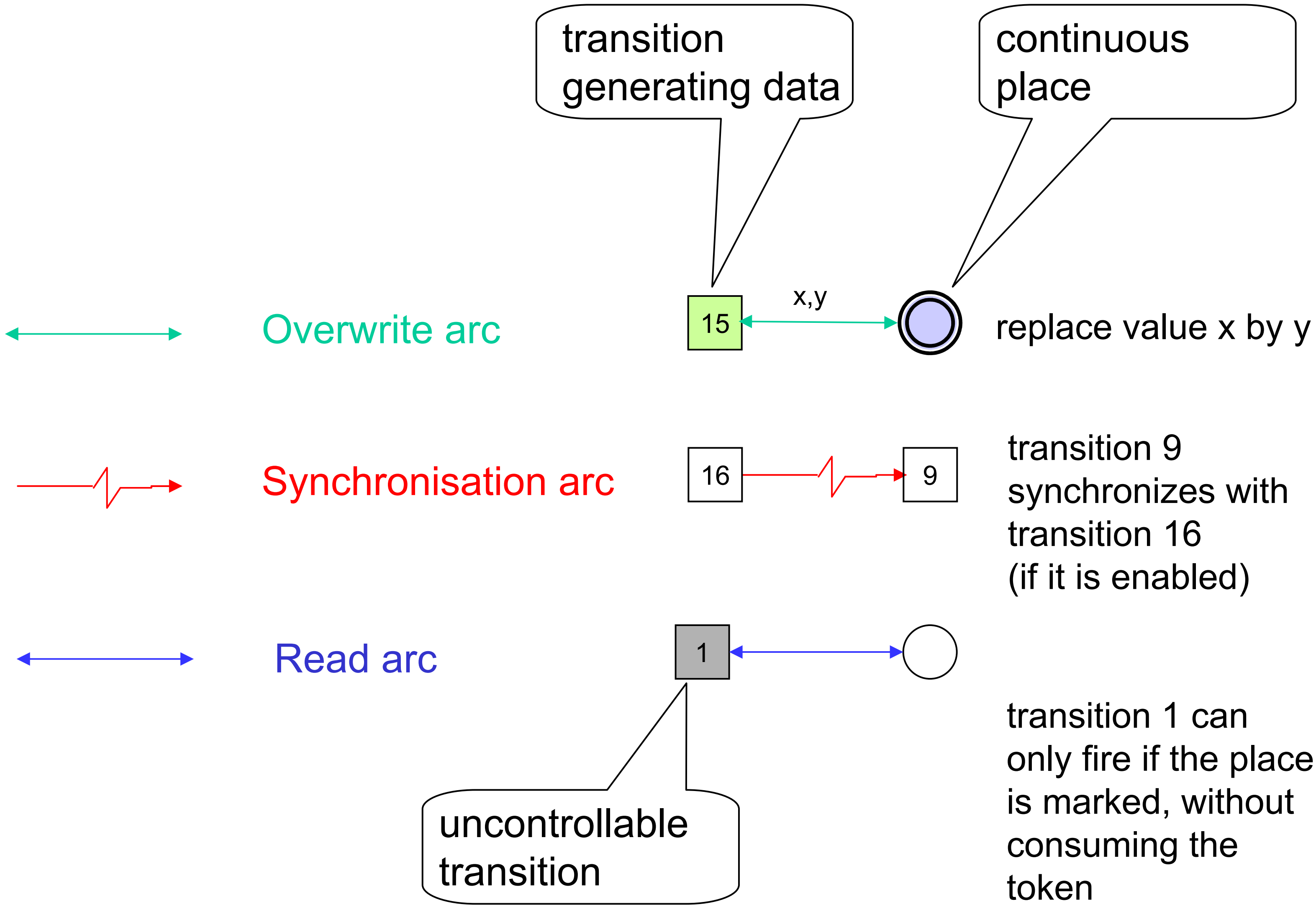
For refuel identification in state “ignition off, gaging round 1 starts 6 seconds after ignition was turned off, and it takes 4 seconds. Gaging round 2 happens when ignition is turned on again (0.5 seconds)

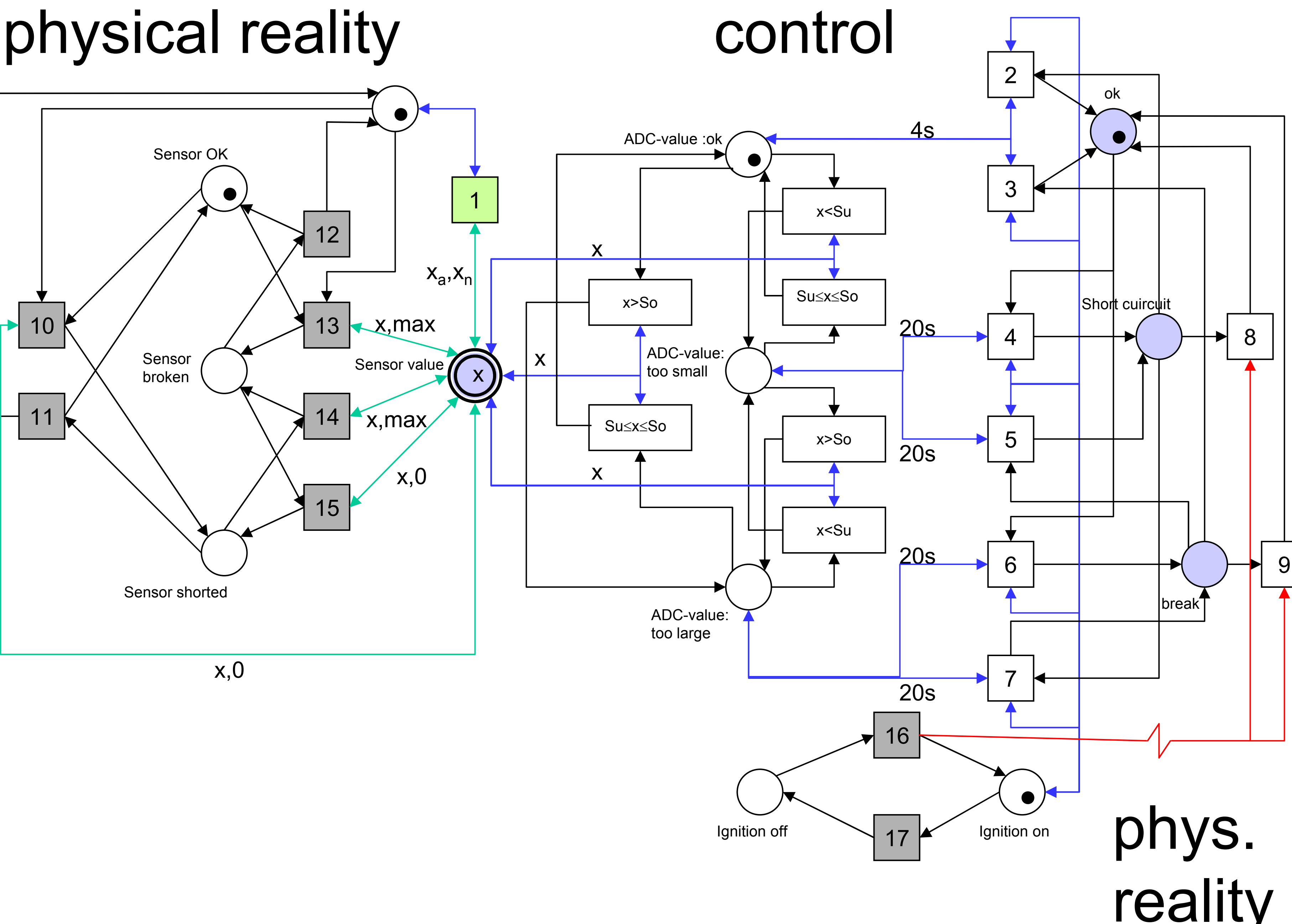
If there is not sufficient time to perform gaging round 1 while ignition is off, no refuel identification will be performed. The result of gaging round 1 survives immediate on- and off-turning of the ignition which does not suffice for gaging round 2.

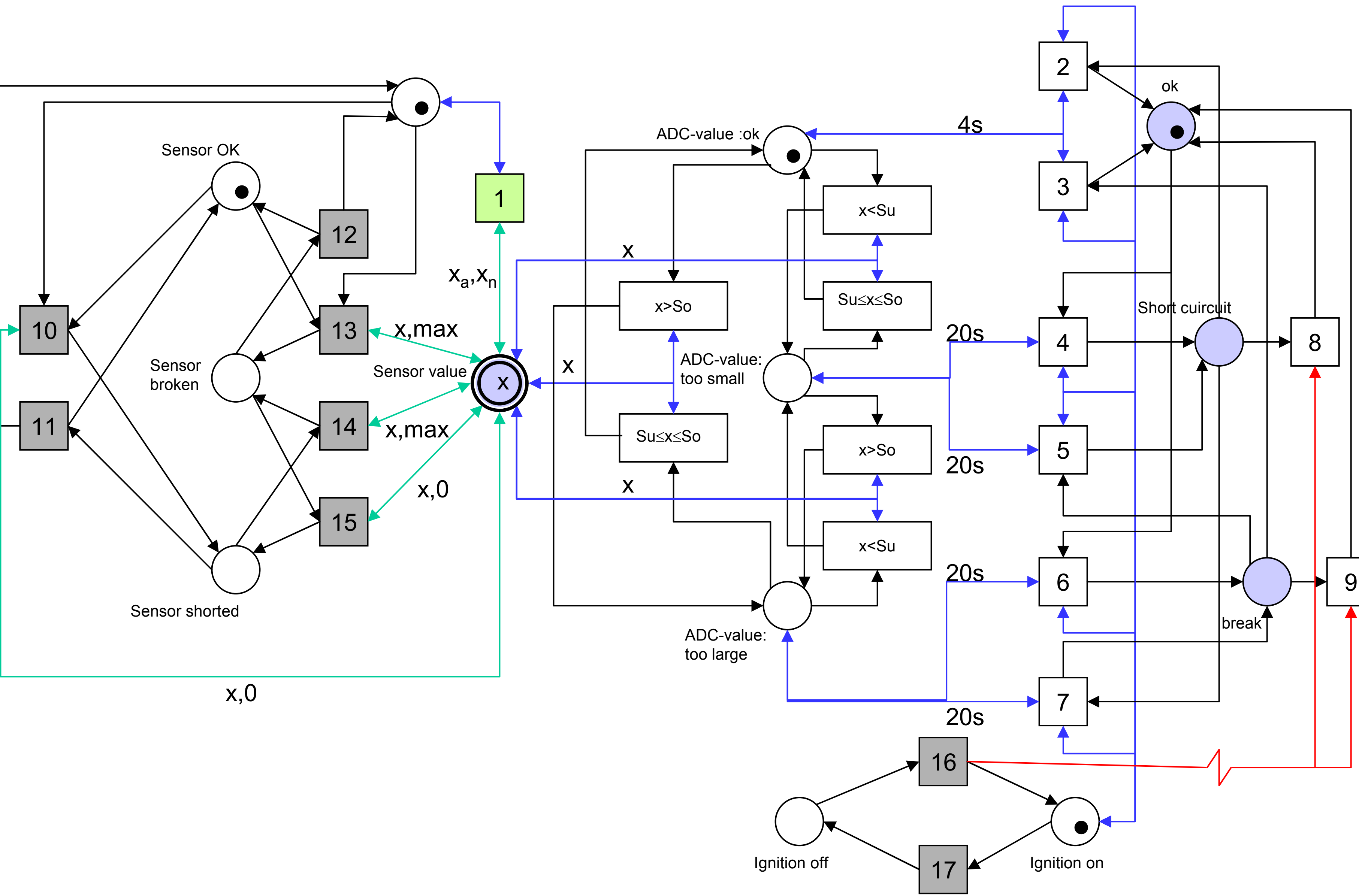


Sensor OK
Sensor broken
Sensor shorted
Sensor value
x
1
x_a, x_n
x,max
x,max
x,0
x,0
10
11
12
13
14
15
ADC-value :ok
ADC-value: too small
ADC-value: too large
x<Su
Su≤x≤So
x>So
Su≤x≤So
x>So
x<Su
x
x
x
4s
20s
20s
20s
20s
2
3
4
5
6
7
8
9
ok
Short cuircuit
break
16
17
Ignition off
Ignition on

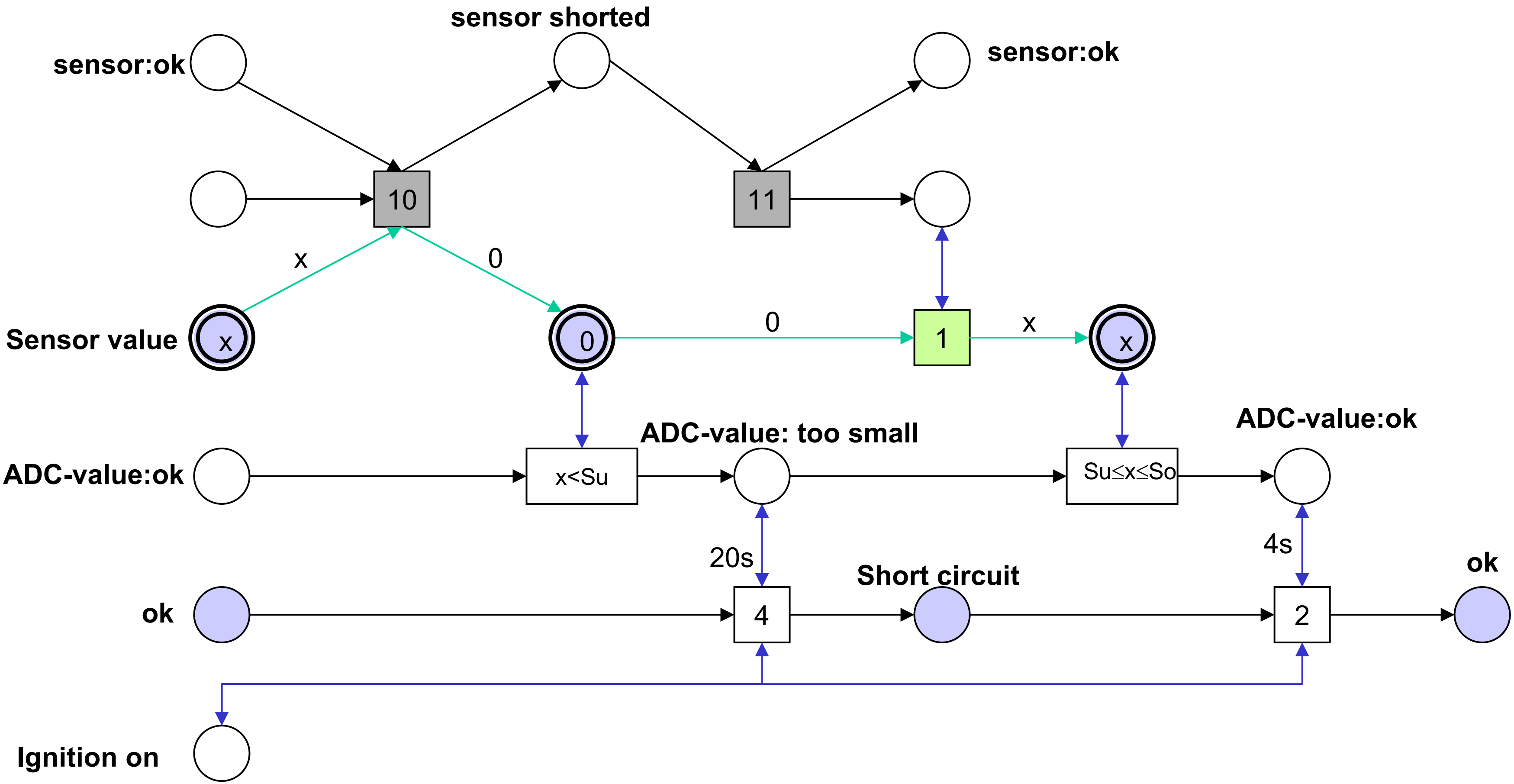
extensions

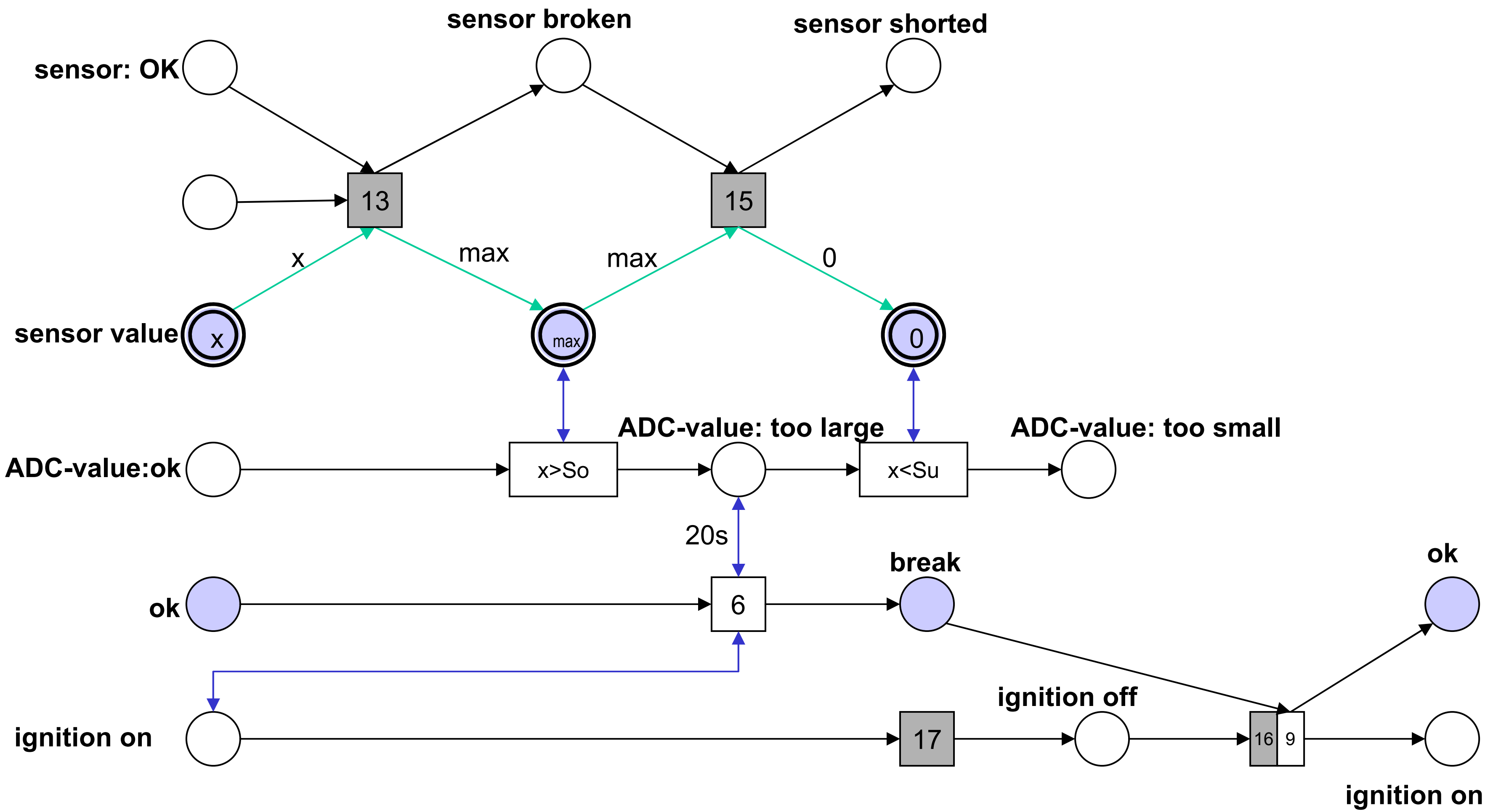

transition generating data
continuous place
x,y
15
Overwrite arc
replace value x by y
Synchronisation arc
16
9
transition 9 synchronizes with transition 16 (if it is enabled)
Read arc
1
uncontrollable transition
transition 1 can only fire if the place is marked, without consuming the token


physical reality
control
phys. reality
ok
4s
ADC-value :ok
Sensor OK
1
2
3
12
x
x<Su
x_a,x_n
Su≤x≤So
10
13
x,max
x>So
Short cuircuit
20s
4
8
Sensor broken
Sensor value
x
x
ADC-value: too small
11
14
x,max
Su≤x≤So
x>So
5
20s
x,0
15
x
x<Su
20s
6
9
Sensor shorted
ADC-value: too large
break
x,0
7
20s
16
17
Ignition off
Ignition on

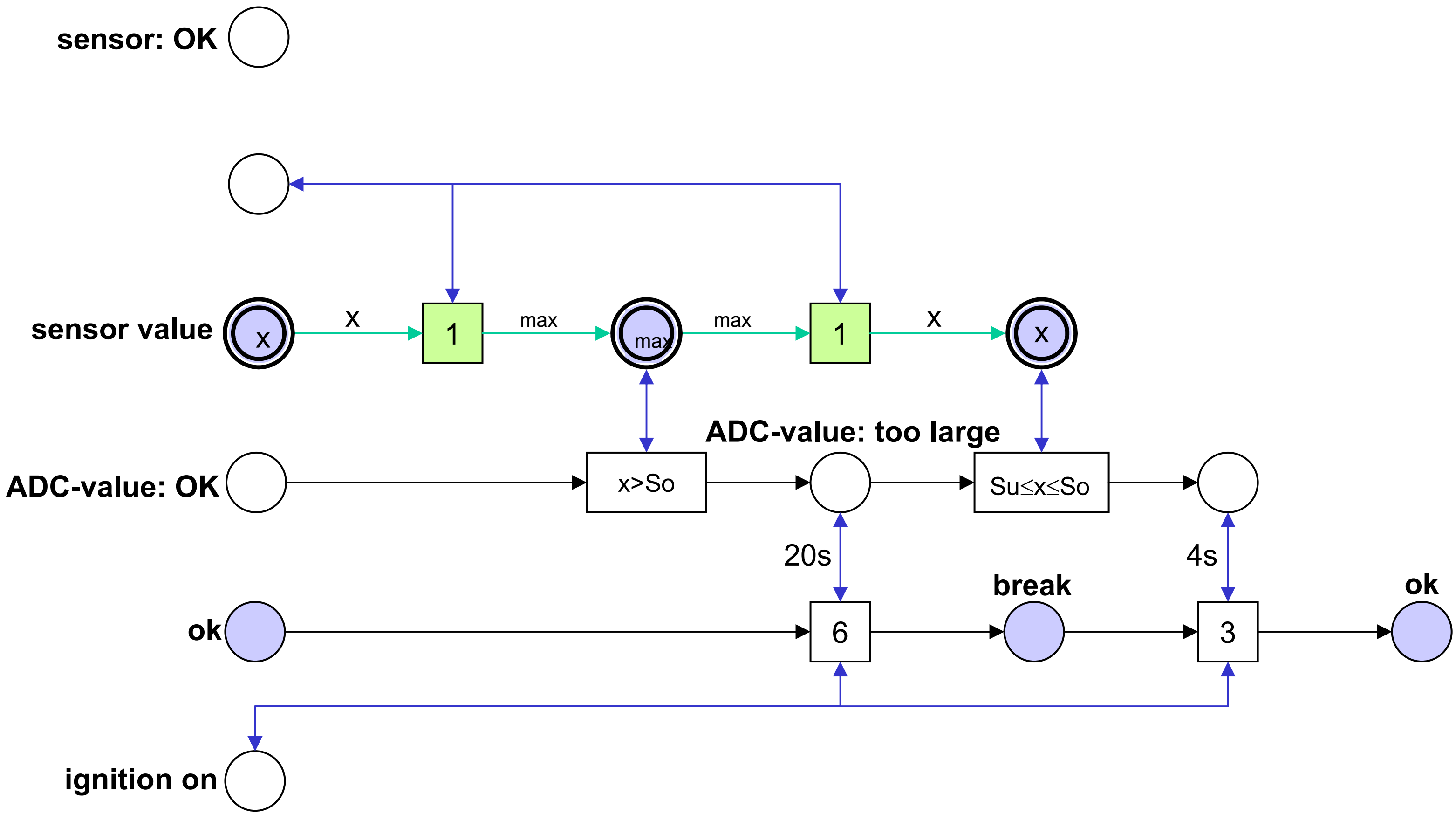


Sensor OK
Sensor broken
Sensor shorted
Sensor value
x
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
x_a, x_n
x,max
x,max
x,0
x,0
x
x
x
ADC-value :ok
ADC-value: too small
ADC-value: too large
x<Su
Su≤x≤So
x>So
x>So
Su≤x≤So
x<Su
4s
20s
20s
20s
20s
ok
Short cuircuit
break
Ignition off
Ignition on

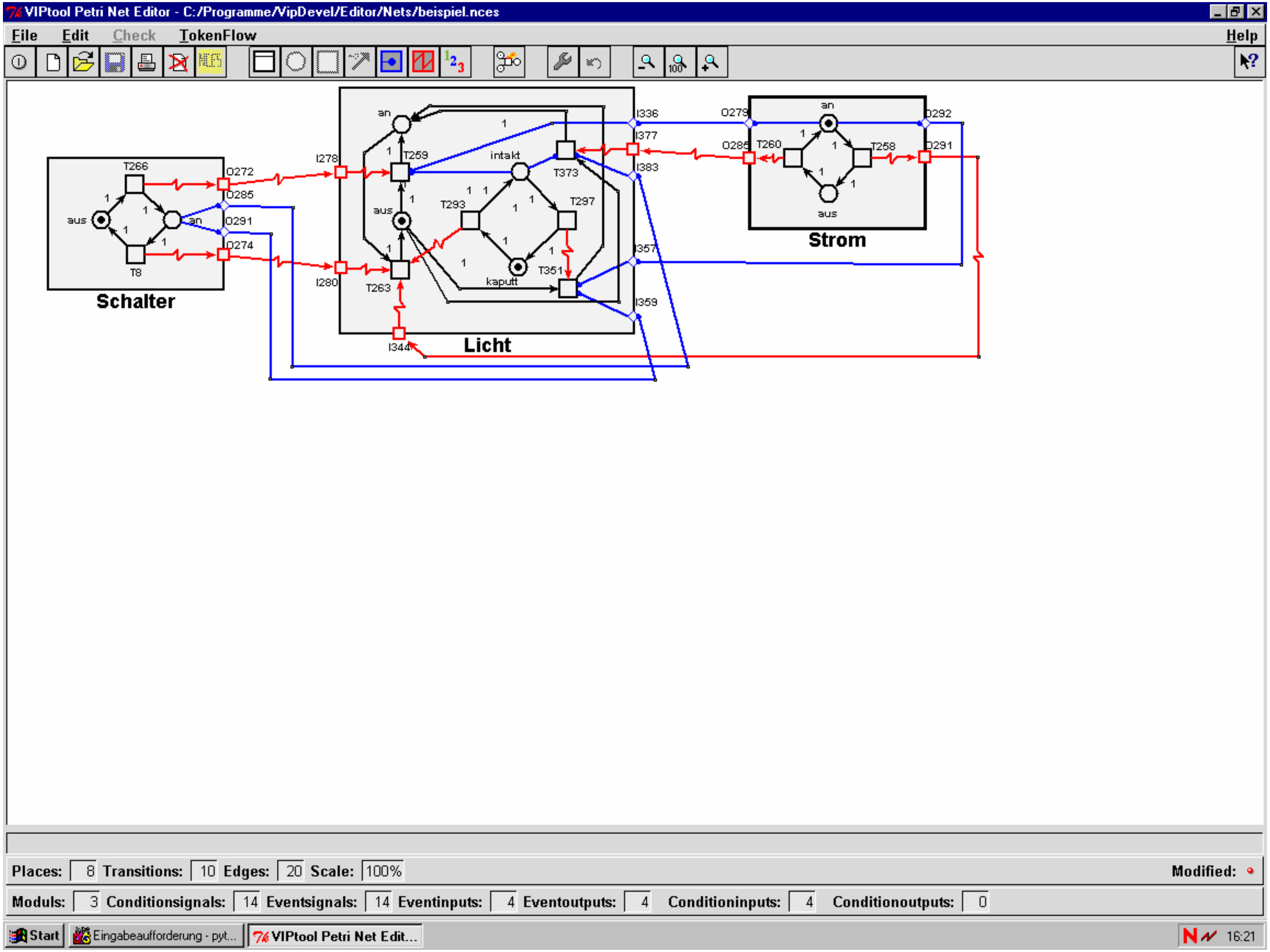

sensor shorted
sensor:ok
sensor:ok
10
11
x
0
Sensor value
x
0
0
1
x
x
ADC-value: too small
ADC-value:ok
ADC-value:ok
x<Su
Su≤x≤So
20s
4s
ok
4
Short circuit
2
ok
Ignition on



sensor broken
sensor shorted
sensor: OK
13
15
x
max
max
0
sensor value
x
max
0
ADC-value: too large
ADC-value: too small
ADC-value:ok
x>So
x<Su
20s
break
ok
ok
6
ignition off
ignition on
17
16
9
ignition on



sensor: OK
sensor value
x
1
max
max
max
1
x
x
ADC-value: too large
ADC-value: OK
x>So
Su≤x≤So
20s
4s
break
ok
ok
6
3
ignition on


VIPtool Petri Net Editor - C:/Programme/VipDevel/Editor/Nets/beispiel.nces
File
Edit
Check
TokenFlow
Help
Schalter
T266
T8
aus
an
O272
O285
O291
O274
Licht
an
aus
intakt
kaputt
T259
T263
T293
T297
T351
T373
I278
I280
I344
I336
I377
I383
I357
I359
Strom
an
aus
T260
T258
O279
O285
O292
O291
Places: 8
Transitions: 10
Edges: 20
Scale: 100%
Modified:
Moduls: 3
Conditionsignals: 14
Eventsignals: 14
Eventinputs: 4
Eventoutputs: 4
Conditioninputs: 4
Conditionoutputs: 0
Start
Eingabeaufforderung - pyt...
VIPtool Petri Net Edit...
16:21