CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

DEVS, TIMED AUTOMATA and FORMAL VERIFICATIONS

Norbert Giambiasi
norbert.giambiasi@univ.u-3mrs.fr

LSIS : Laboratoire des Sciences de l'Information et des Systèmes
UMR 6168

The aim of this presentation is not to give detailed results on the transformation of DEVS models into Timed Automata.

It has rather for object to open research ways which could introduce the DEVS formalism into real-time system design

DEVS Standardization Study group.

Minutes of the first meeting.
April 26th. 2001 (ASTC; Seattle, WA).

It is proposed to make diffusion in Academic environments, to impose the standard in the academia as a way of achieving diffusion.

DEVS Standardization group

Meetings in: European Simulation Symposium (October 2001, Marseille, France)

Prof. Kim

Decide which is the goal of the std. to be proposed: interoperation? Systems analysis?

Defining a goal is the first step to be carried out.

Or Modeling during a Design process.

DEVS Standardization group

Meeting in Wintersim (December 2001, Arlington, VA)

DEVS can be used as the core formalism, in order to enable proof of correctness.

Decide which is the goal of the std. to be proposed: interoperation? Systems analysis?

Defining a goal is the first step to be carried out.

This Presentation

Design approaches for real-time systems

DEVS can be used as the core formalism, in order to enable proof of correctness.

Links between DEVS and Timed Automata

It is proposed to make diffusion in Academic environments, to impose the standard in the academia as a way of achieving diffusion.

Links between DEVS and Timed Automata

DEVS or TIMED AUTOMATA

Behind this presentation and the transformation of DEVS models into timed automata, we try to answer to the questions:

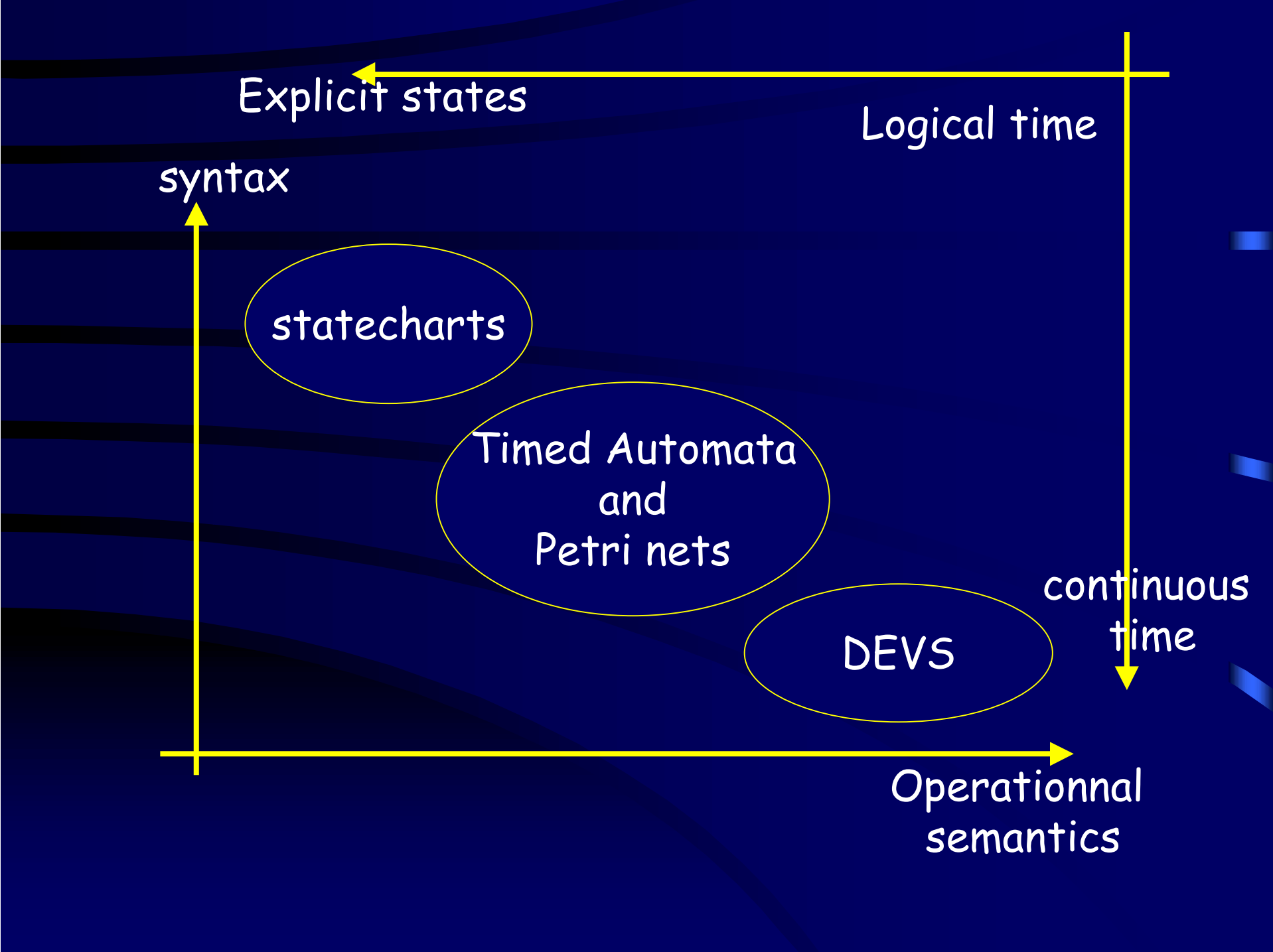
Why, in the field of software engineering, people use timed automata (Petri nets) rather than DEVS ?

How (and where) to introduce the use of DEVS formalism into software engineering approaches ?

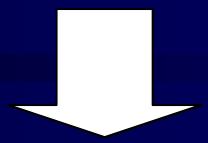
Why, in the field of software engineering and computer sciences, academic people use timed automata (or Petri nets or Statecharts) rather than DEVS ?

A first answer can be :

DEVS formalism is not framed within the dichotomy between syntax and semantics.
DEVS is not a syntactic formalism with a corresponding semantic model,
it is a symbolic specification of system semantics [O'Neill].


Explicit states
Logical time
syntax
statecharts
Timed Automata
and
Petri nets
DEVS
continuous
time
Operationnal
semantics

DEVS = clean operational semantics
and clean interpretation



- Only **one way** to execute the model,

- A clean interpretation of the model elements in the real-world

Are these 2 properties considered as needed for high-level specification formalisms by software engineering community ?

Why, in the field of software engineering and computer sciences, academic people use timed automata (or Petri nets or Statecharts) rather than DEVS ?

Some other answers can be :

Classical academic approaches are based on formal verification and not on simulation,

DEVS models are deterministic and this can be considered as an inconvenience for high level specifications of real-time software

DEVS well-adapted to low-level models → Analysis models of real systems

Timed Autmata well-adapted to high-level models → Software specifications

Why to establish a link between DEVS and Timed Automata

To allow
formal verification
methods to be used
in the
DEVS world

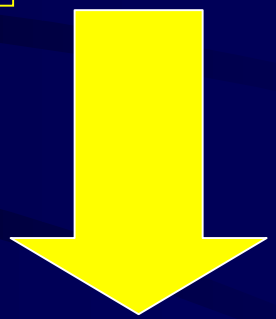
To allow
clean and successful
simulation methods
to be used
in the Timed Automata
world

Why to establish a link between DEVS and Timed Automata

Design Methodology

Timed automata can be seen as a high level specification formalism

DEVS model can be seen as a low level specification of the system



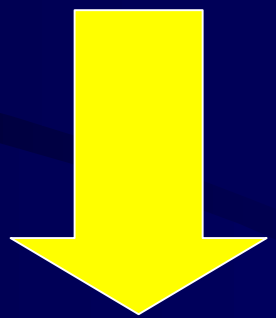
Prove that the DEVS model implements correctly the specification given by the Timed automaton

Why to establish a link between DEVS and Timed Automata

Design Methodology

Timed automata used as a high level specification for control systems

DEVS used to represent the systems to be controlled



Verify the behavior of the coupled model

TIMED AUTOMATON

T.A introduced in the 90 for high level
formal specifications
of real-time systems

A variety of formal methods have been developed to prove that a T.A. satisfies basic correctness properties
and timing properties
which guarantee the performance
of the real systems.

Model checking was one of the most successful verification technique.

An analysis is done to explore the reachable state space. Tools are typically unable to analyze models with a large number of states

The notions of **safety and liveness** properties have been introduced to express that:

-'something (bad) will not happen during an execution: safety property,

-eventually 'something (good) will happen during an execution: liveness property

Simulation Relationship

Some works in verification of real-time systems are based on the use of something called 'simulation'

A **simulation proof** involves establishing a correspondence between the states of two models M1 and M2, one of these models is regarded as an implementation and the other as a specification.
The correspondence between these two models is called a **simulation relationship.**
The existence of a simulation relationship is used to show that any behavior that can be exhibited by M1 can also be exhibited by M2
Typically, the model M1 contains more details than the specification M2.

TIMED AUTOMATA

Classical timed automata (TA) are, non-deterministic finite discrete state automata extended with a finite number of real-valued clocks.

A TA alternates between two modes of execution, letting time pass continuously, then taking a step changing its discrete state.

A classical timed automaton is seen as a generator (or an acceptor), it has only output (input) actions or events associated with state transitions.

Recall: TIMED AUTOMATA
basic definitions from the literature

The states of the automaton are called ***vertices (location)*** and the transitions (arcs, edges) are called ***switches.***

Switches (edges) are **instantaneous**.

Time can **elapse** in locations.

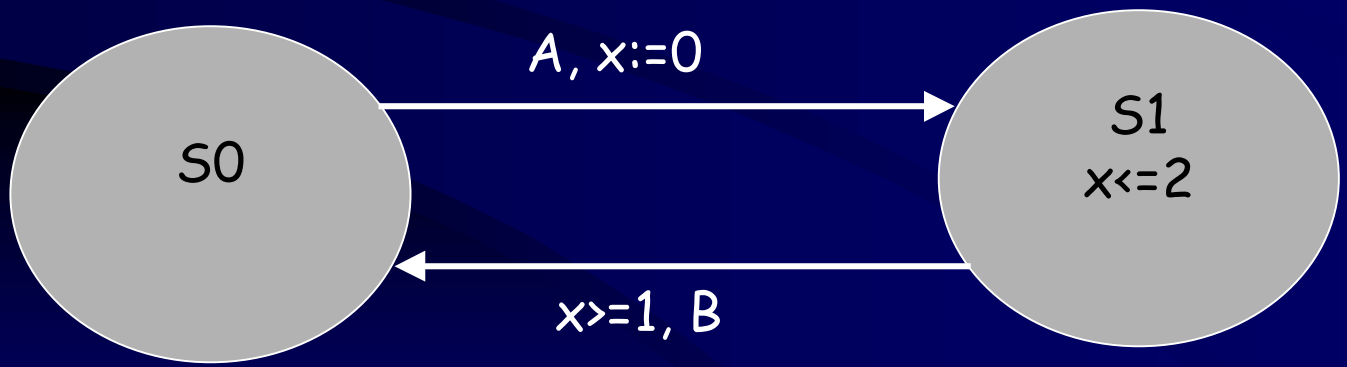
A clock can be reset to zero simultaneously with any switch.

The reading of a clock equals the time elapsed since the last time it was reset (time is global)

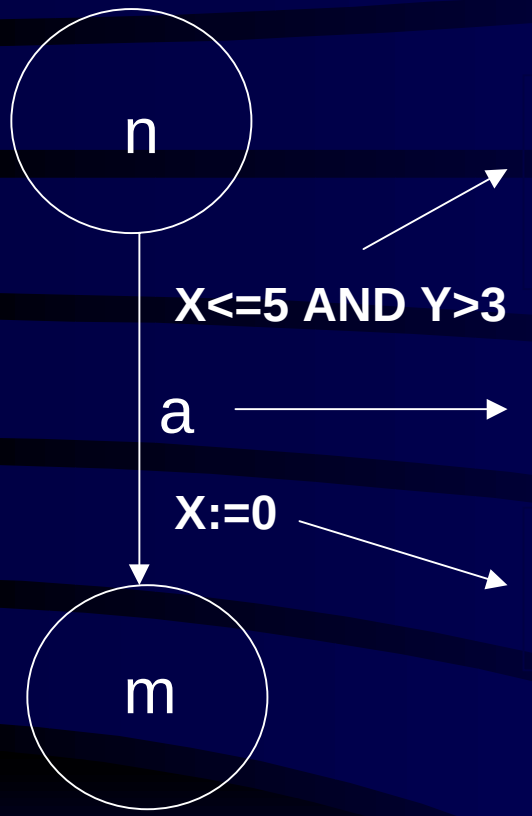
TIMED AUTOMATA

With each switch one may associate a clock constraint, and require that the switch may occur only if the current values of the clocks satisfy this constraint.

With each location we associate a clock constraint called its *invariant,*
and require that time can elapse in a location only as long as its invariant stays true.



Timed Automata



GUARD
Boolean combination of comparisons with integer bounds

Action
used for synchronization

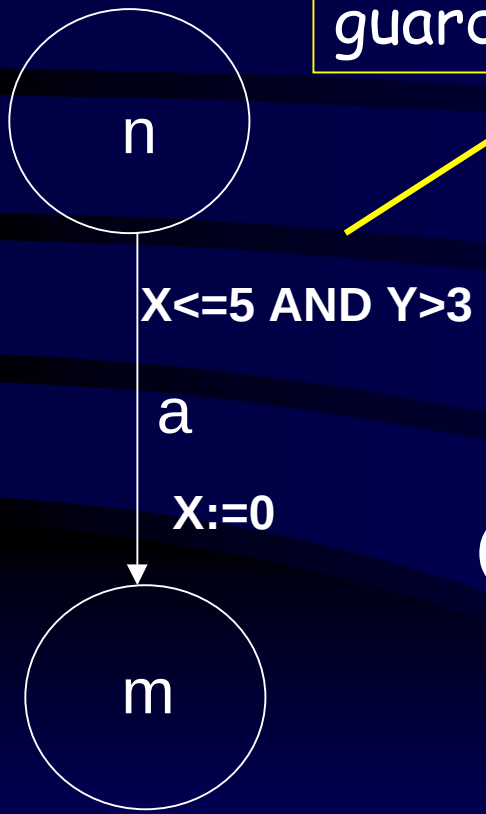
Reset
Action performed on clocks

A STATE of the Automaton is defined by:
(location, X, Y)
called total state in the DEVS formalism

Timed Automata

Transitions

guards indicate when an edge ***may*** be taken,



Transition due to a location switch (discrete):

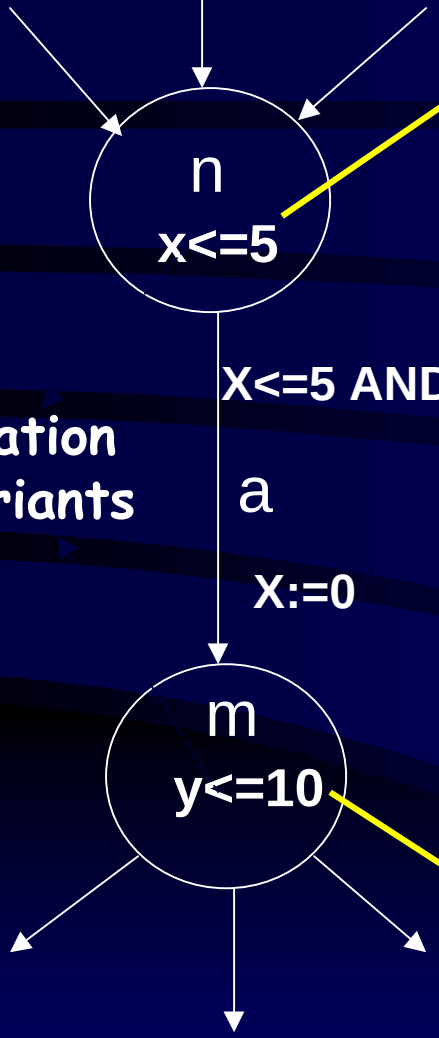
$$(n, x = 2.4, y = 3.31) \xrightarrow{a} (m, x = 0, y = 3.31)$$

Transition due to time elapse (continuous):

$$(n, x = 2.4, y = 3.31) \xrightarrow{e(1.1)} (n, x = 3.5, y = 4.41)$$

Timed Automata

Invariants



Location invariants

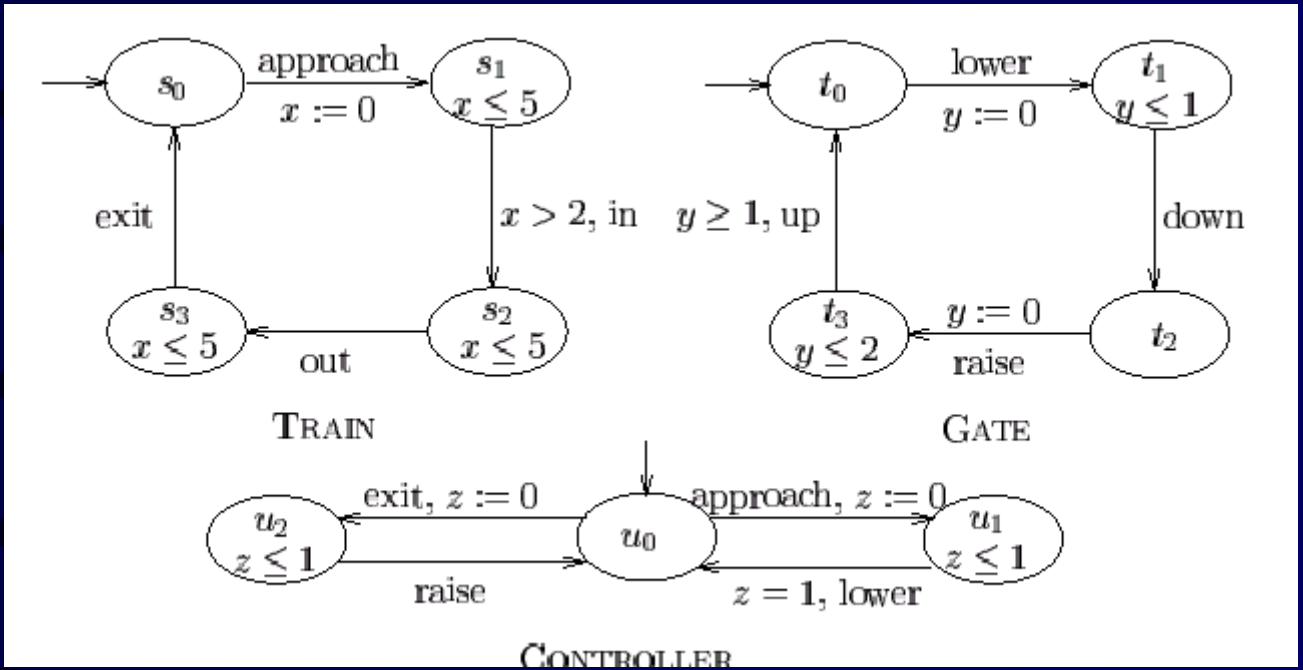
an invariant is a clock constraint that specifies the amount of time that **may be** spent in a location

(n, x = 2.4, y = 3.31) —e(3.2)→ ~~

(n, x = 2.4, y = 3.31) —e(1.1)→ (n, x = 3.5, y = 4.41)

(location) invariants are used **to** ***force*** an edge to be taken

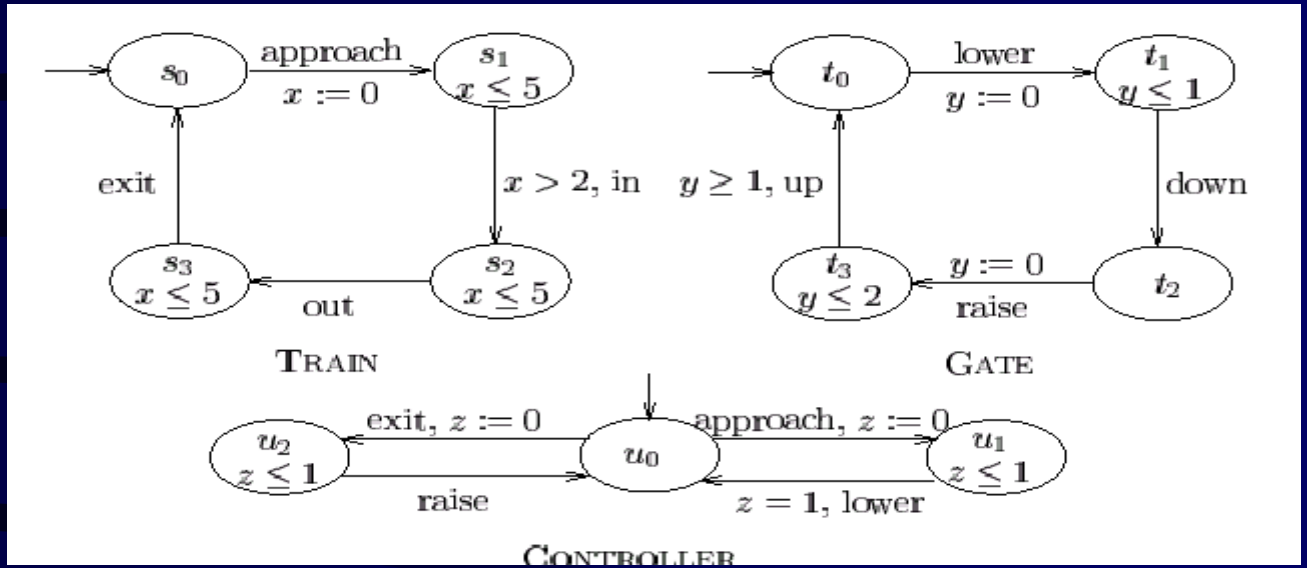
Example from R. Alur: Composition of T.A

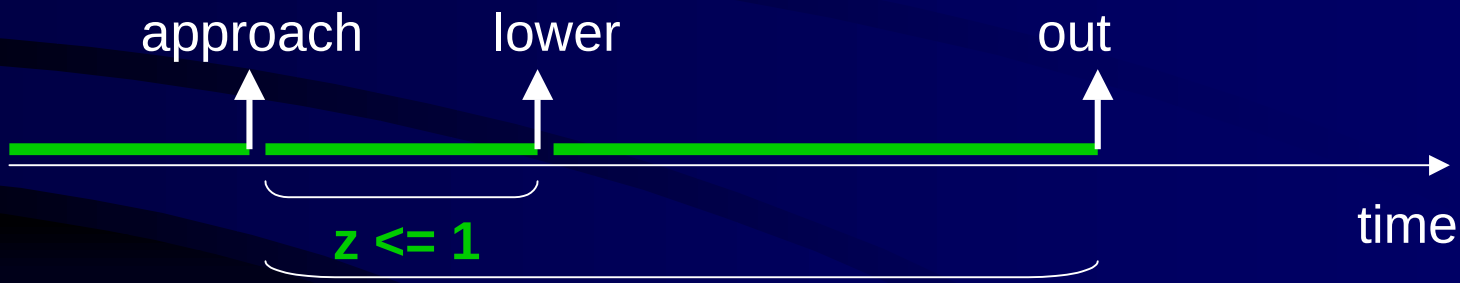


Interpretation : the train is far in the state s0, it sends ' approach ', the clock x is reset to 0 , it goes in the state s1, it is near the crossing. The invariant x<=5 defines the life-time of the state s1.
At least 2 t.u after ' approach ', the train sends 'in ' and goes in s2. It remains in s2 no more than 3 t.u, ...

The specification is not deterministic

The T.A is not deterministic

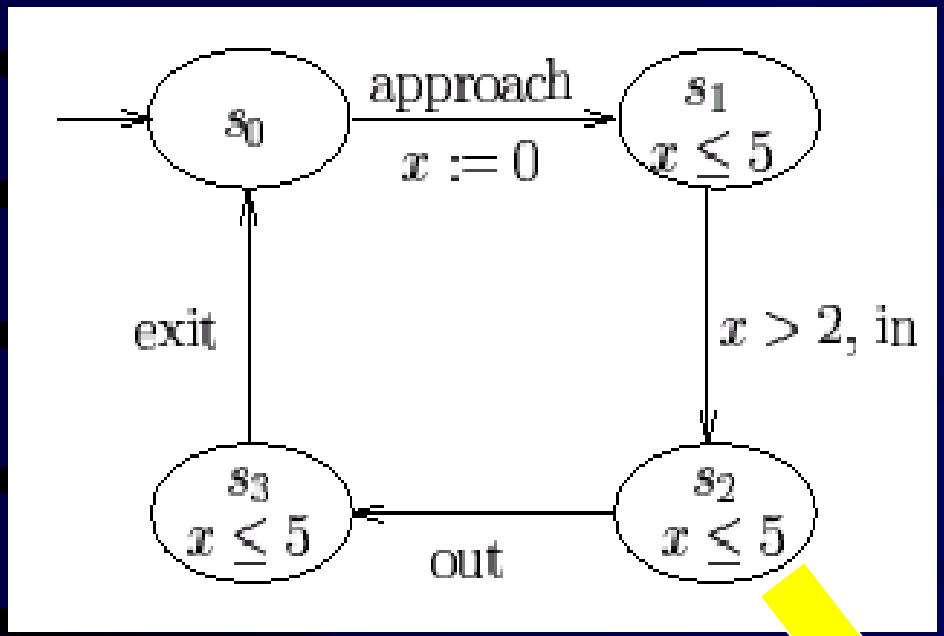




x > 2 ∧ x <= 5 → Temporal constraint between "approach" and "out"

"out" must occurs after "lower" because x and z are reset during the discrete transitions with "approach"

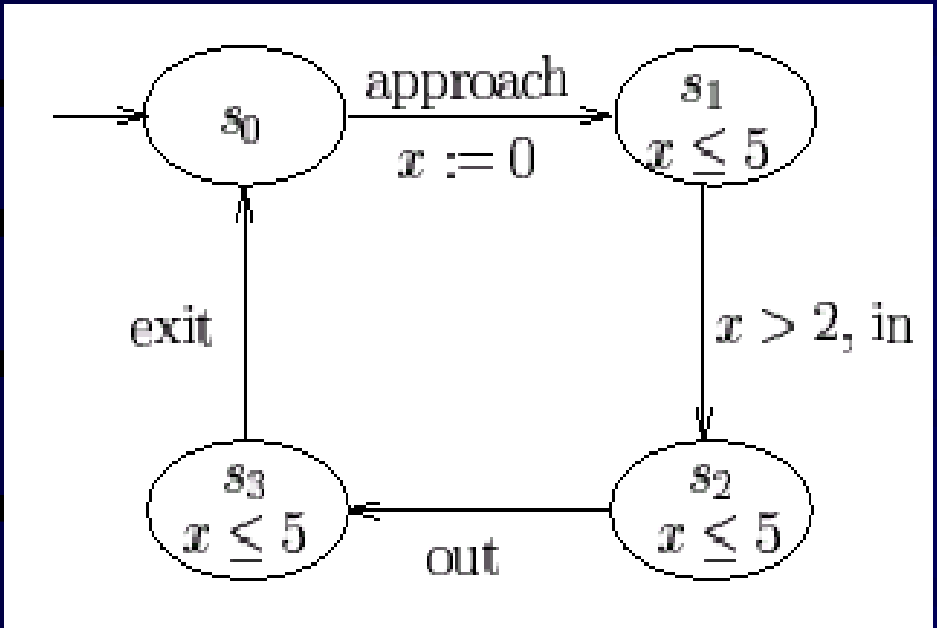
Remark:



X is not the life-time of the state s2

$\sigma = x$ - (elapsed time in s1)

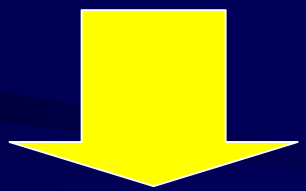
because x was reset to 0 during the discrete transition (s0,s1)



To obtain the same specification in DEVS, we must add a state variable which memorizes the elapsed time in s1 and which is used to define the life-time of s2

The timed constraint between the actions 'approach' and 'out' is expressed more easily with the T.A

Timed automata are oriented towards the definition of timed constraints (using clocks) between events



They are not **simulation models**
But they can be used as high level specifications of a simulation models

T.A and DEVS

CLOCKS

The clocks { c1,c2,…cj,….cn } of a timed automaton are state variables in the corresponding DEVS model

with :

$$cj = f (e, cj)$$

(e elasped time in the state)

INVARIANT

The invariant of a state sj of a T.A directly linked with the life time of the discrete state in the DEVS world

GUARDS

A guard allow to define the next discrete state taking into account the elapsed time in the present state

From DEVS to Timed Automata Syntax and Semantics

DEVS formalism is not framed within the dichotomy between syntax and semantics.
DEVS is not a syntactic formalism with a corresponding semantic model,
it is a symbolic specification of system semantics [O'Neill].

Untimed automaton and Syntactic Untimed DEVS Model

Definition:
A syntactic untimed DEVS model is a structure :

$$A = (Q; E; \Sigma; src; act; trg; q0),$$

Where :
- Q is a finite set of **explicit discrete states** (*limitation)*
- E a finite set of transitions,
– Σ a set of actions (events),
src : E → Q, associates a source to a transition
act : E → Σ associates an action (event) to a transition
trg : E → Q associate a target to a transition
q0 is the initial state.

M = < X, S, Y, δ int , δ ext , λ , D>.

We have the following relationships :

$S = Q$,

$\Sigma = X \cup Y$

$\exists$ **ei** $\in$ **E** if $\exists$ a pair (si, sj) such as

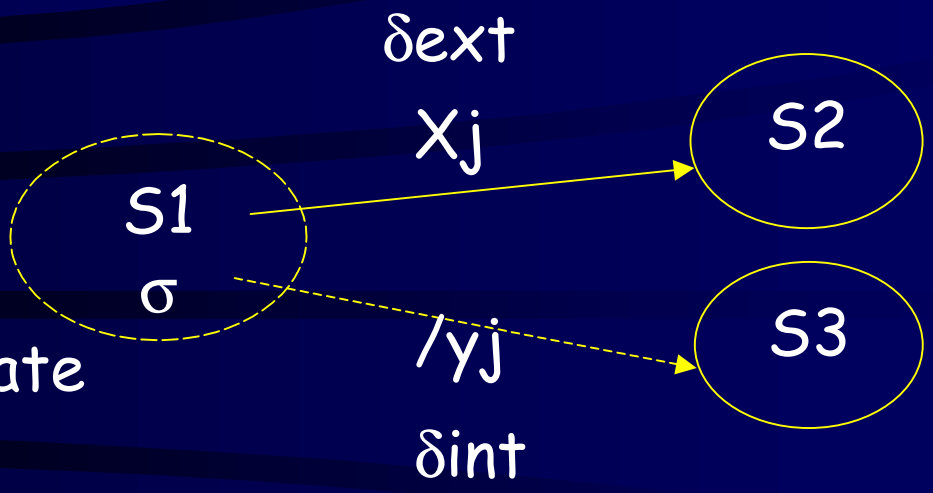
- $\exists$ (si, e) and $\exists$ xk such as δ ext(si, e, xk) = sj ,
- OR $\exists$ (si, e) such as δ int(si) = sj

Then : src (ei) = si,
trg(ei) = sj

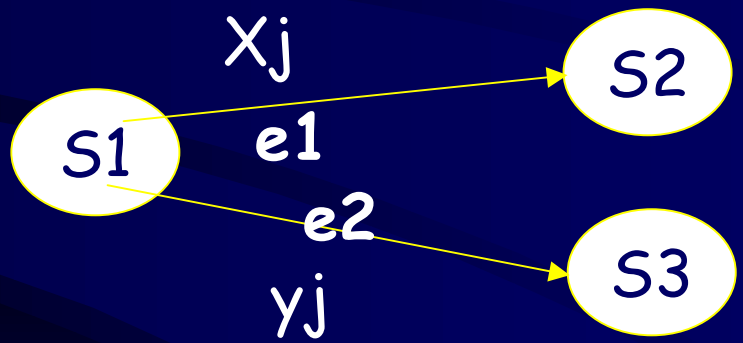
and

- act(ei) = xk if δ ext(si, e, xk) = sj
  this kind of edges corresponds to an ***external event***
- act(ei) = yk if δ ext(si) = sj with λ(si) = yk
  this kind of edges corresponds to an ***internal event***

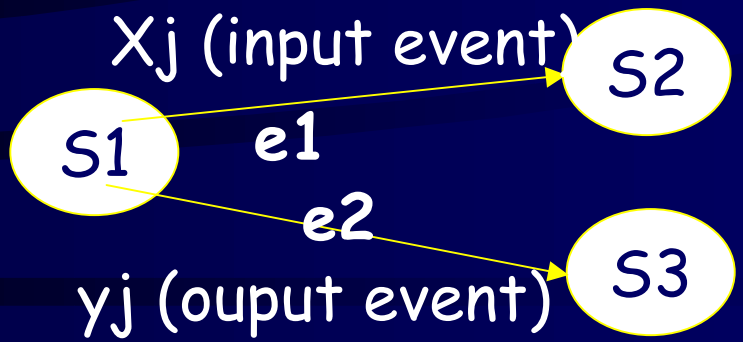
DEVS



Untimed Syntactic DEVS



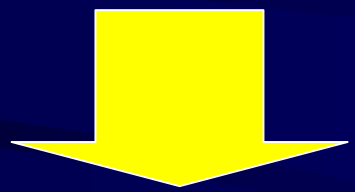
edge e1 -->	external transition,
edge e2 -->	internal transition.



In the DEVS world,
The edge e1 corresponds to an external transition,
e2 to an internal transition.

At the syntactic level,
nothing is said on how to do
the simulation of the model
Operational Semantics rules must be
added to allow simulation

Before introducing **Operational Semantics rules** ,
we define the timing annotation of
a syntactic DEVS in order
to represent time



Timing Annotation for syntactic DEVS

Timing Annotation for syntactic DEVS

Definition :

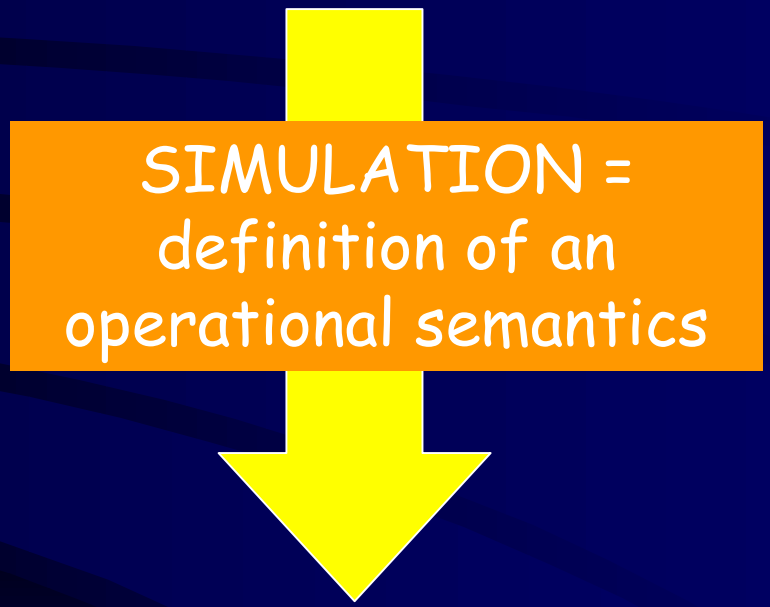
A timing annotation for an untimed DEVS is structure :

T = (C; Inv; G; A; v0),

where

- C is a clock.
- Inv : associates the invariant c <= D(si) for each si ∈ S,
- G : E → F(C) associates the guard c= D(si) for an internal transition, and the guard c<D(si) for an external transition.
- A : E → M(C) associates the assignment c := 0 to each transition.
- c := 0 is the initial valuation of the clock

A syntactic untimed DEVS with a timing annotation is an **syntactic** atomic DEVS model



Definition of an Operational Semantics for syntactic DEVS with Timing annotation

Operational Semantics of syntactic DEVS with Timing annotation

the operational semantics of a DEVS model can be defined in terms of a transition system (SQ, s0, →) where :

-SQ is a set of total states, i.e. (s; e), where:

$s \in S$ is a discrete state ,

$e \in$ **invar**(e) is a valuation of the clock c satisfying the invariant of s.

-s0 is the initial state (if it exists),

- → is the transition relation that defines how to evolve from one state to another.

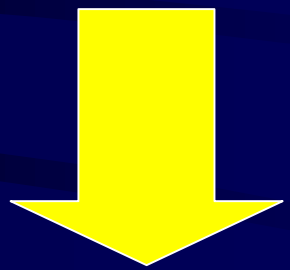
Operational Semantics of syntactic DEVS with Timing annotation

→ is the transition relation, two possible ways in which the model can proceed :

-Discrete transition by traversing an edge (discrete change),

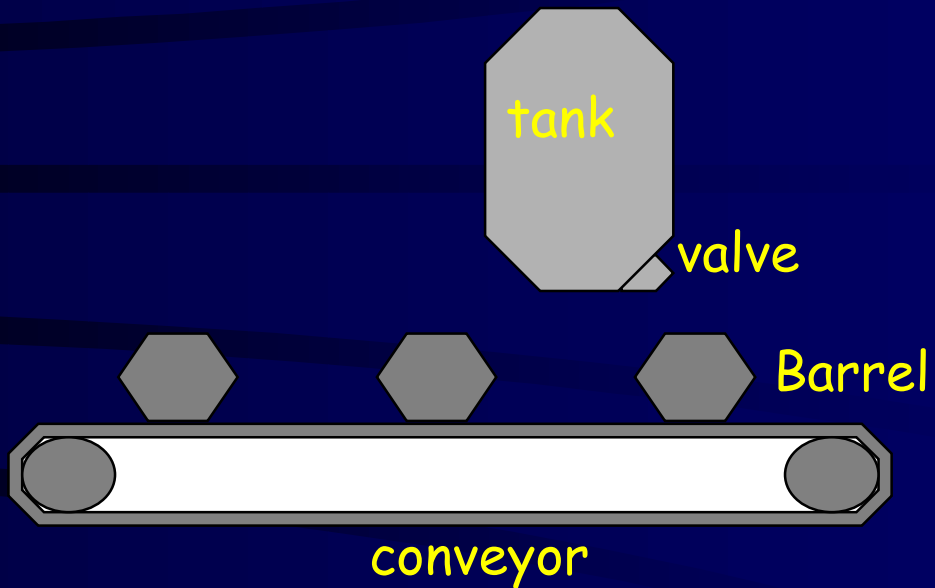
-Time transition by letting time progress while staying in the same discrete state sk as long as the clock c satisfies the invariant of sk.

A syntactic untimed DEVS with a timing annotation and its operational semantics is a classical atomic DEVS model



It is also a deterministic Timed Automaton

Example of transformation: DEVS to T.A - filling system



This filling system has two inputs:

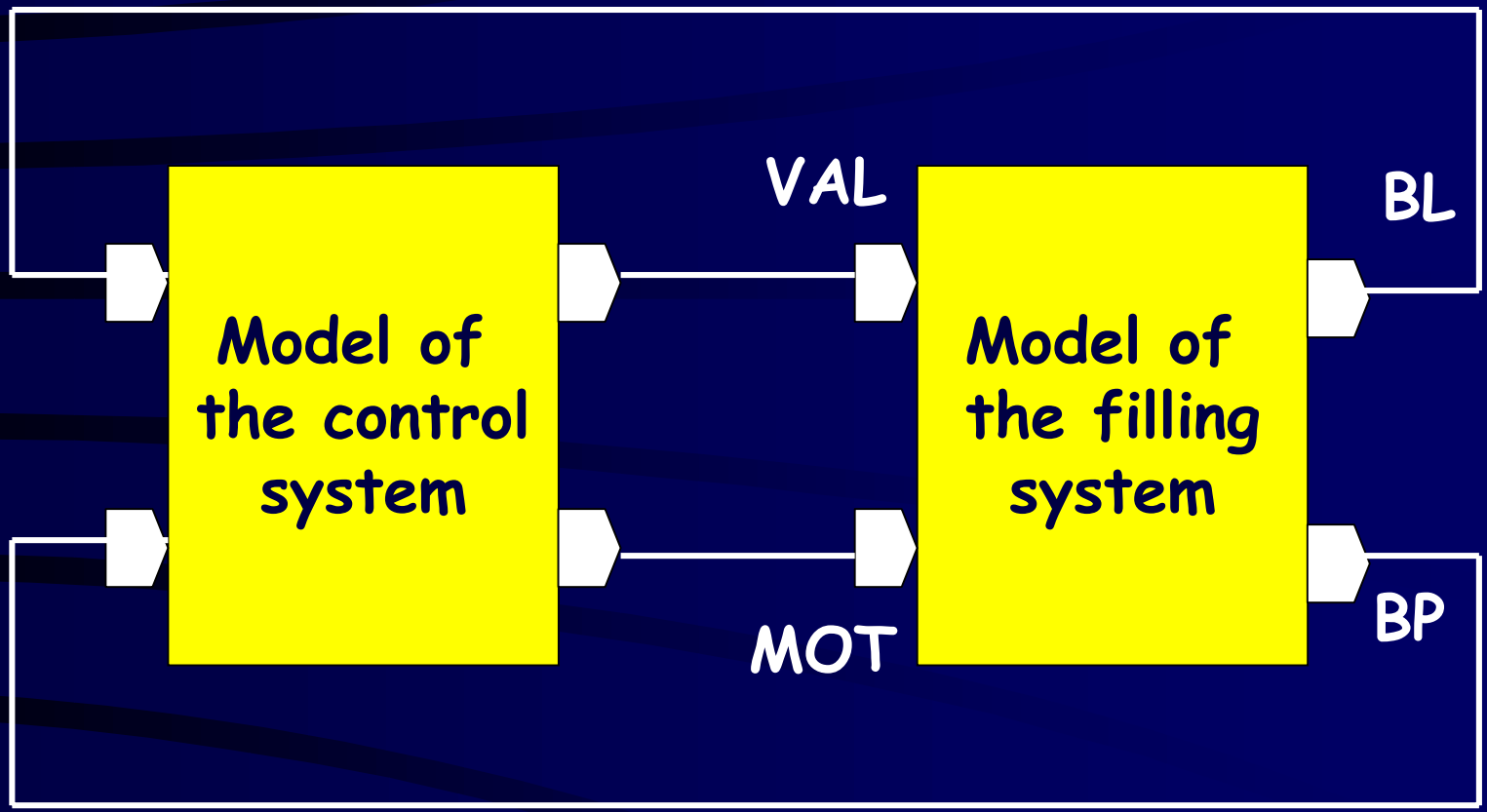
-control of the valve: val = {open, close}

-control of the conveyor: mot ={start, stop}

and two sensor outputs:

-barrel level Bl ={full},

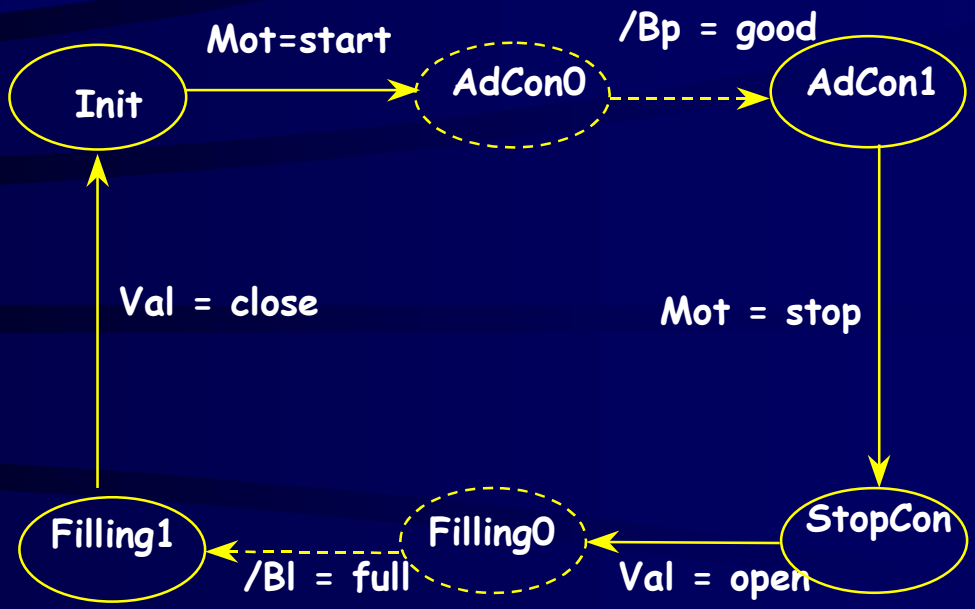
-barrel position Bp = {good}



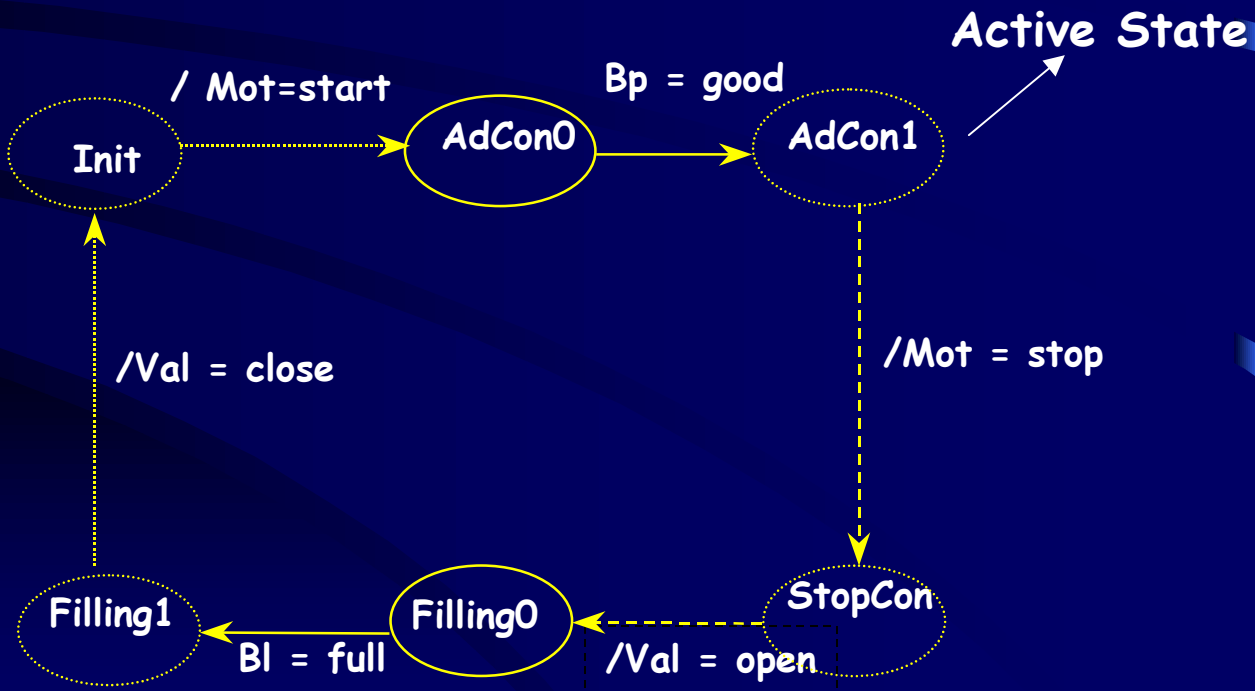
-control of the valve: VAL = {open, close}
-control of the conveyor: MOT ={start, stop}
-barrel level: BL ={full},
-barrel position: BP = {good}

DEVS Models

Filling System



Control System



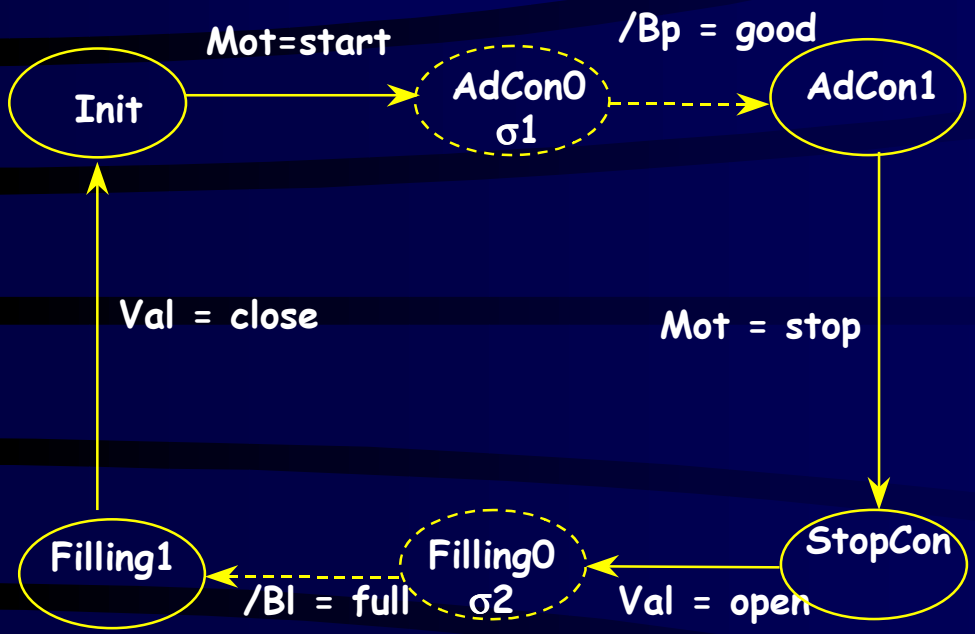
Filling System: From DEVS to T.A

Each active or passive state of the DEVS model corresponds a location (discrete state) in the T.A

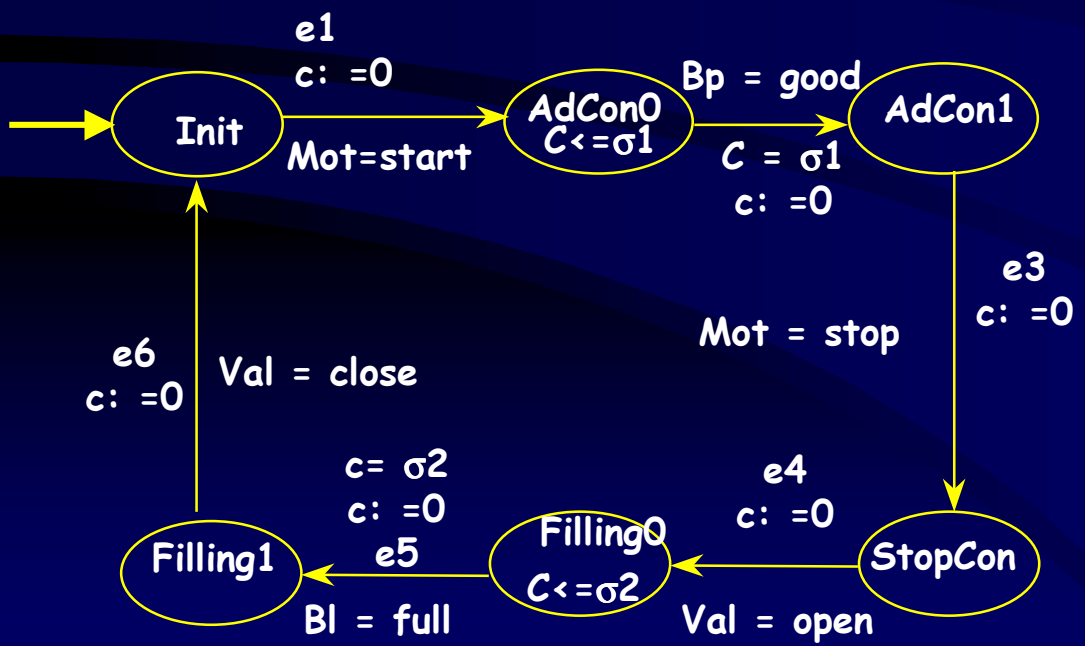
Each internal or external transition of the DEVS model corresponds to an edge in the T.A

Life-times of active states are used to define the invariants and the guards in the T.A

Filling System under control



DEVS model



Timed Automaton
(which is deterministic)

Untimed syntactic DEVS

+

Timing annotation

+

operational semantics

=

Atomic DEVS = Timed Automaton

Design Process of a Control System

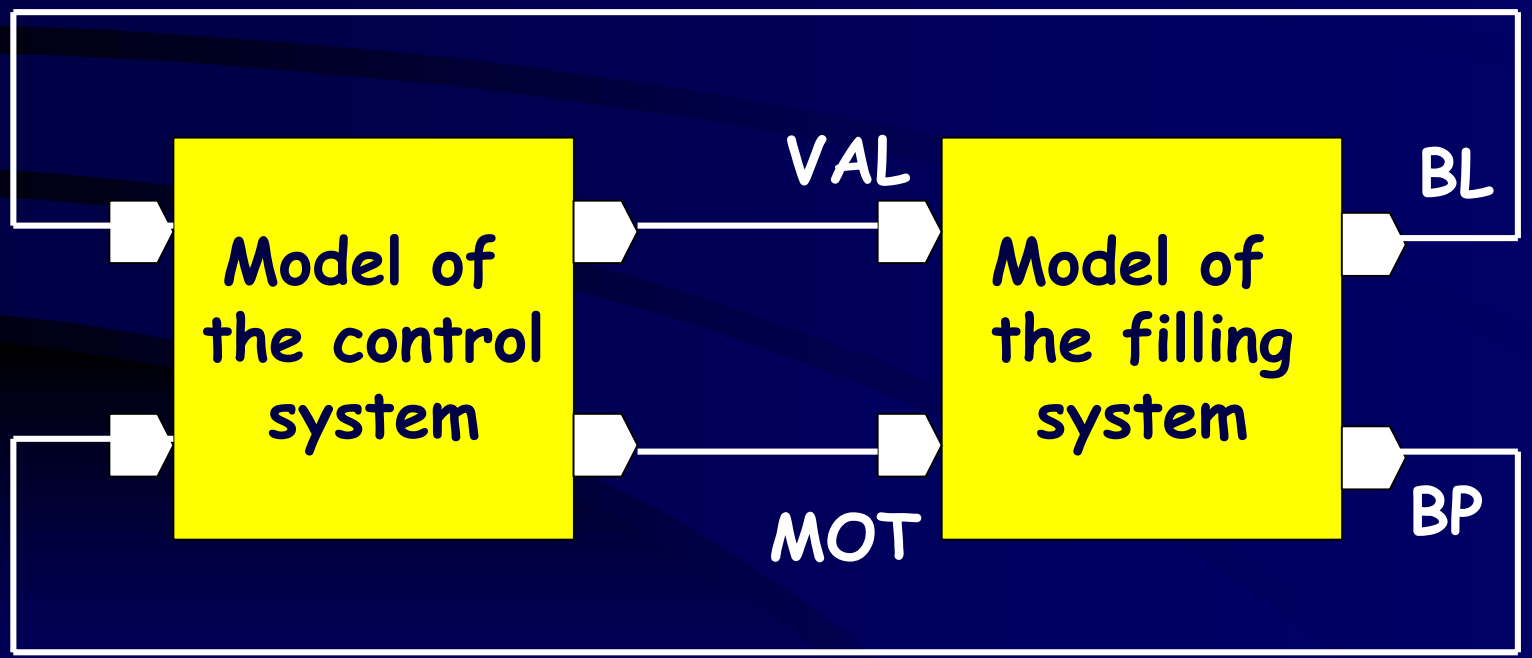
Three steps:

- specification,
- design of an implementation,
- verification that the implementation satisfies the specification

(with feedback loops)

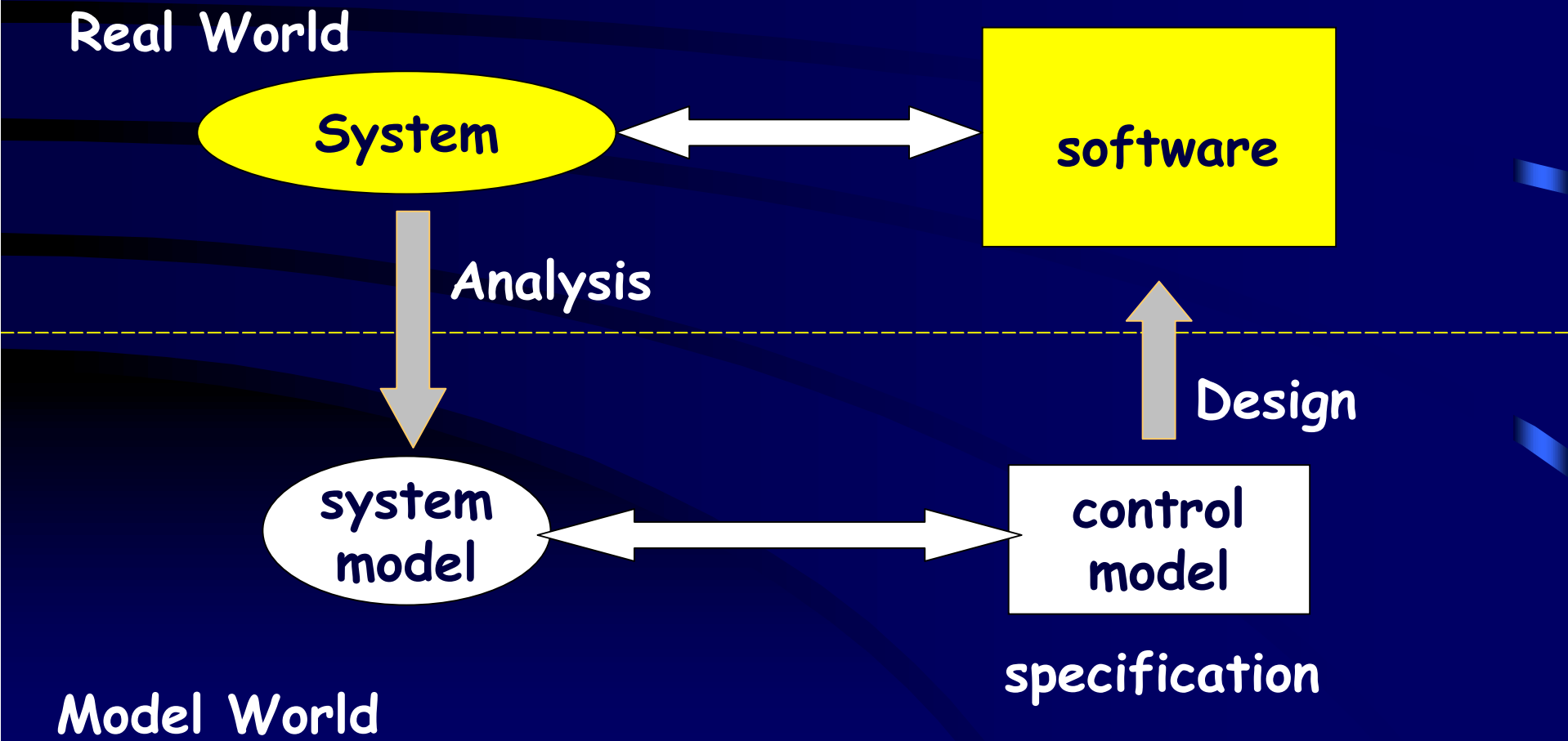
Design Process of a Control System
Model Verification

Aim : built the model of the control system and verify it and the coupled model

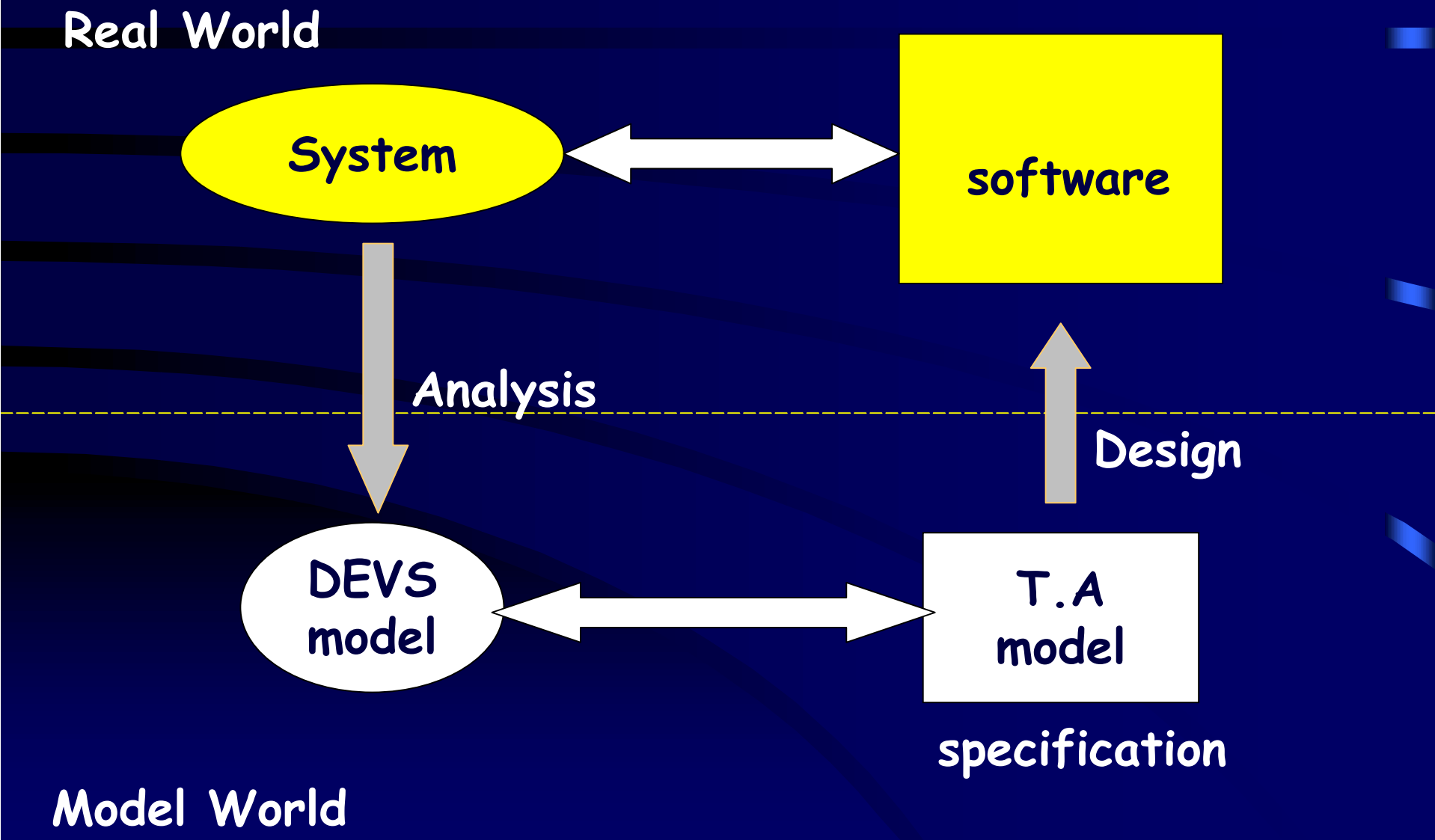


Design Process of a Control System

Specification + Verification



Models in a Design Process of a Control System



Design Process of a Control System
Model Verification

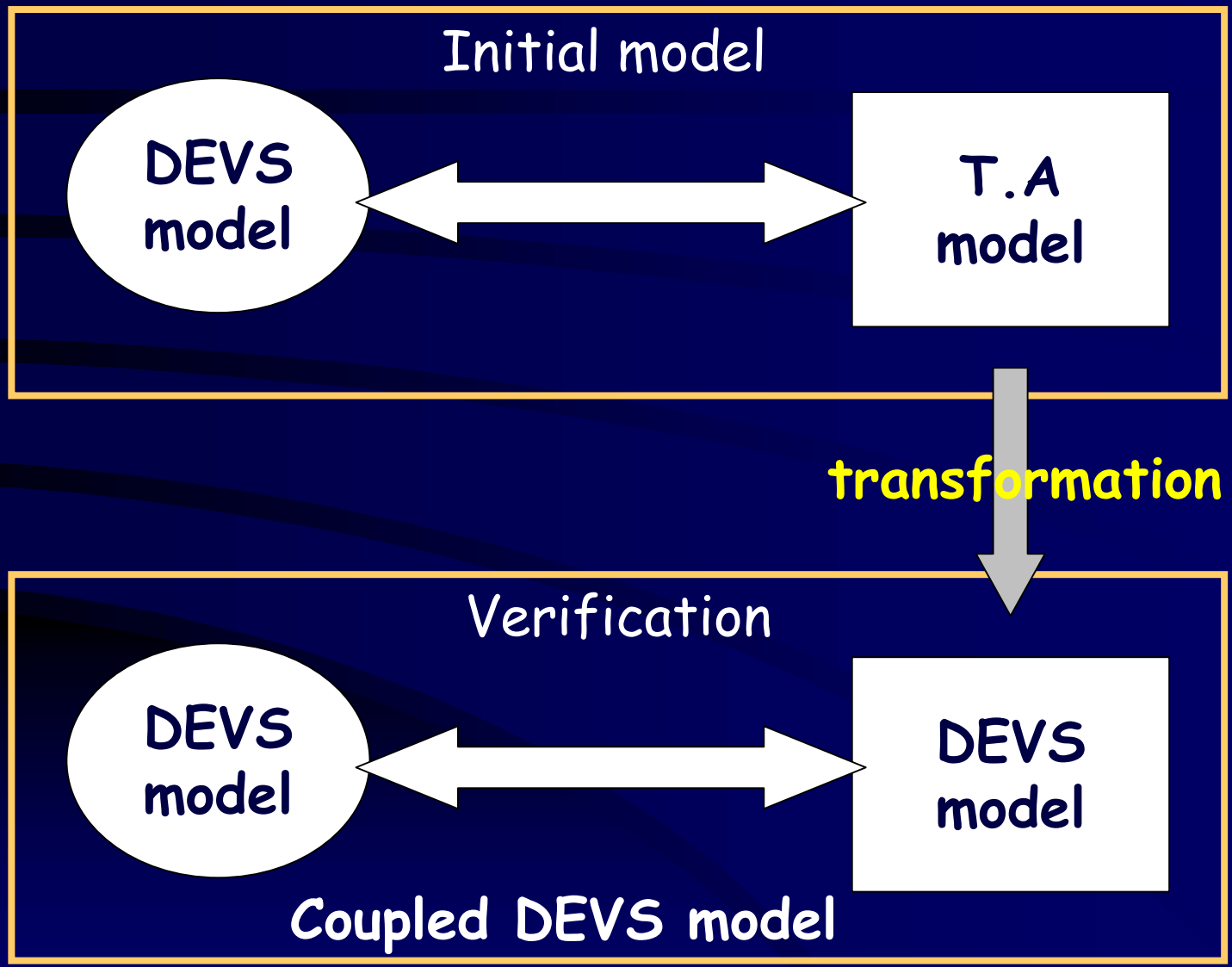
Proposition of 3 possible verification approaches

1 - in the T.A world
by model checking and/or proof of properties

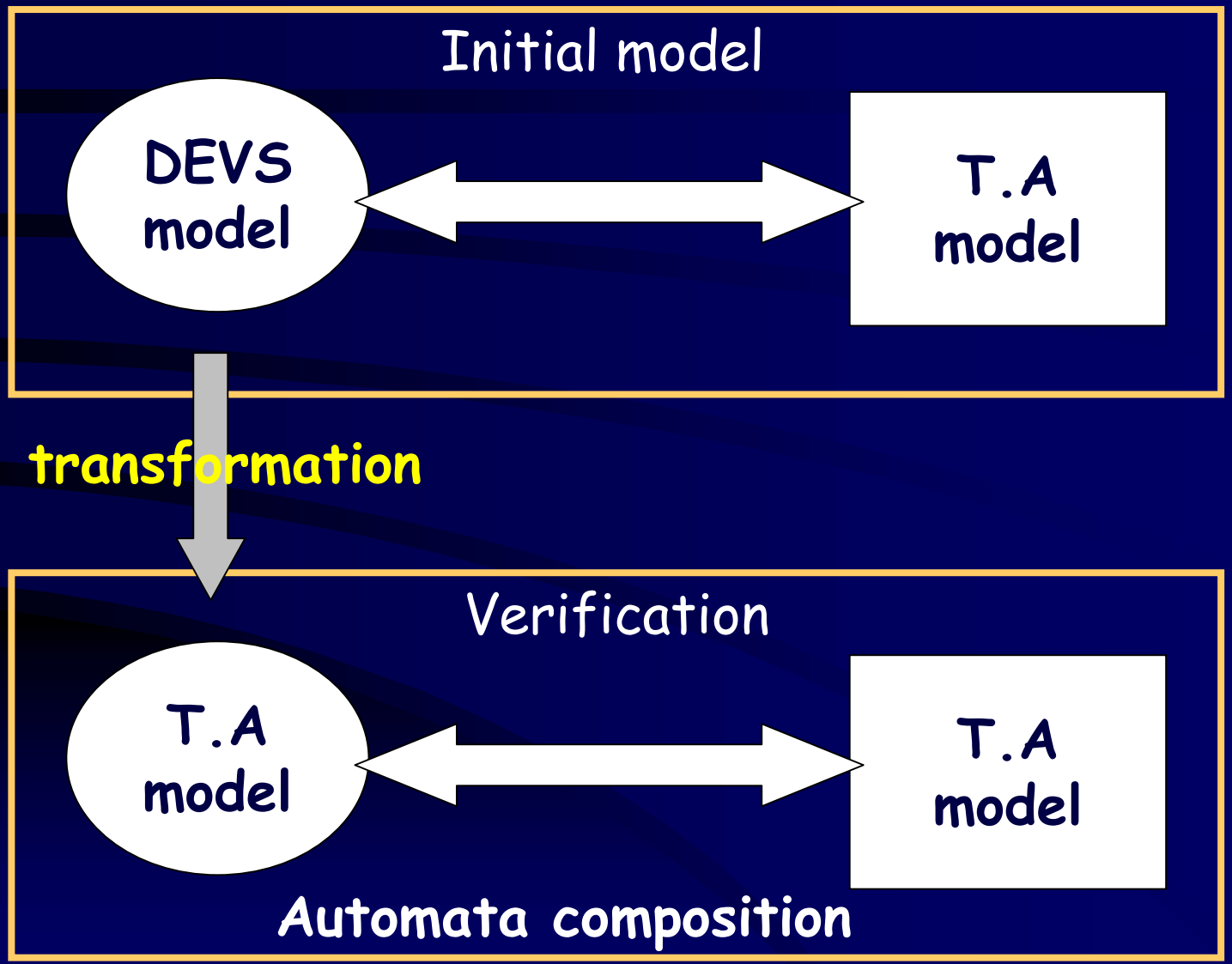
2 - in the DEVS world
based on simulation

3 - in the T.A world,
'simulation proof'

Verification based on Simulation



Formal Verification



T.A world

Formal Verification

DEVS model of the system to be control

Deterministic Timed Automaton

T.A control system

Composition of T.A

Formal Verification

model checking

proof of properties

DEVS world

Verification

T.A of the control system

Transformation

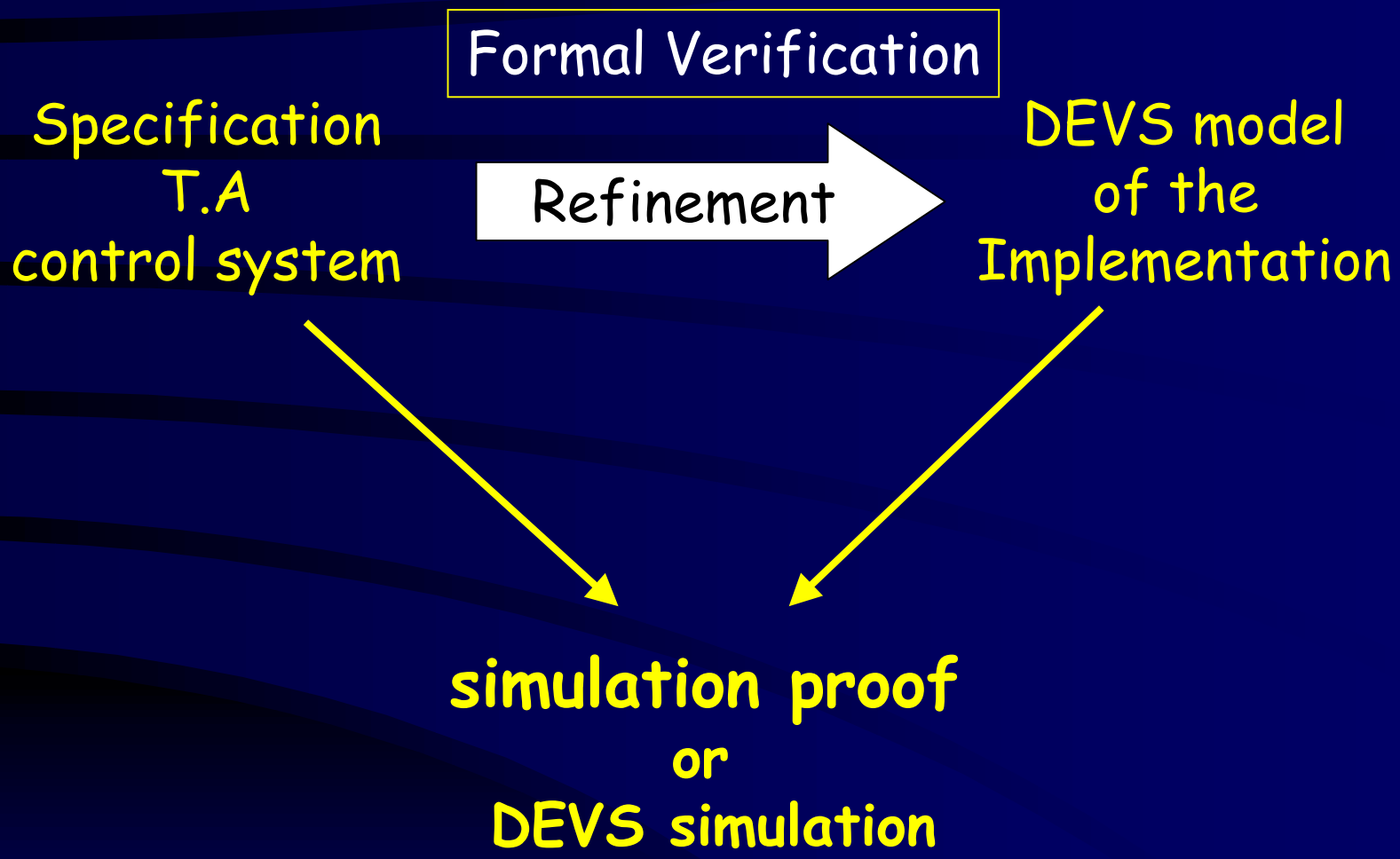
DEVS model

DEVS model of the system to be controlled

Coupled DEVS Model

Verification based on simulation

Other possibility for the design approach



Example of a design Methodology based on Simulation proof

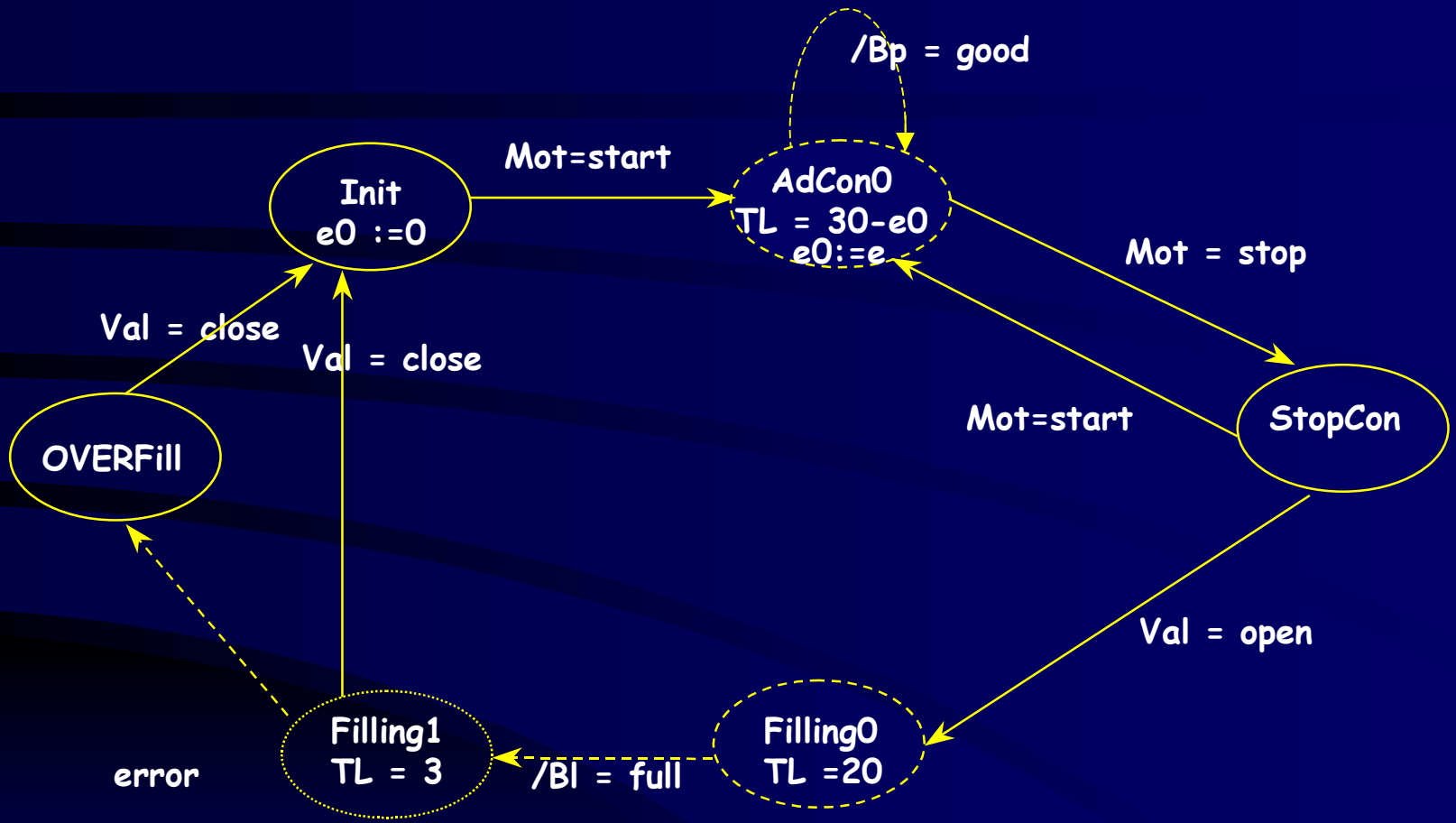
1 - built the min-max DEVS model of the system to be control,

2- built the T.A representing the H-L specification of the control,

3- built the DEVS model of the control implementation,

4- prove that any behavior that can be exhibited by the DEVS model can also be exhibited by the High-level specification.

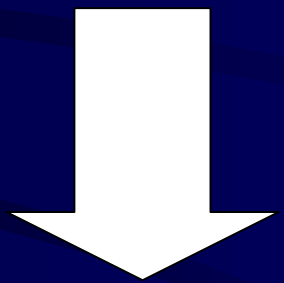
Complete Behavioral Model of the Filling System



In this model, we do not make the hypothesis that the control events occur at the good times.

In this model, with precise values for the life-times (time advance) of the states, we have done the hypothesis that, for example, the time interval between the events Mot= start and Bp = good has always the same value.

It is obvious that this hypothesis is not realistic



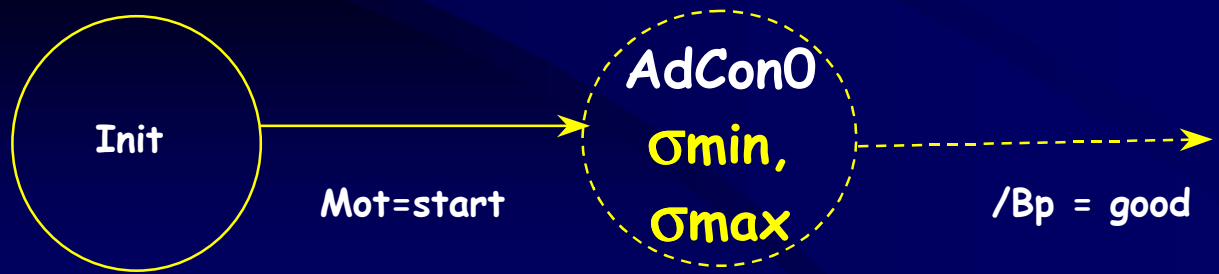
Min-Max DEVS [giam2000]

built the min-max DEVS model of the system to be control,

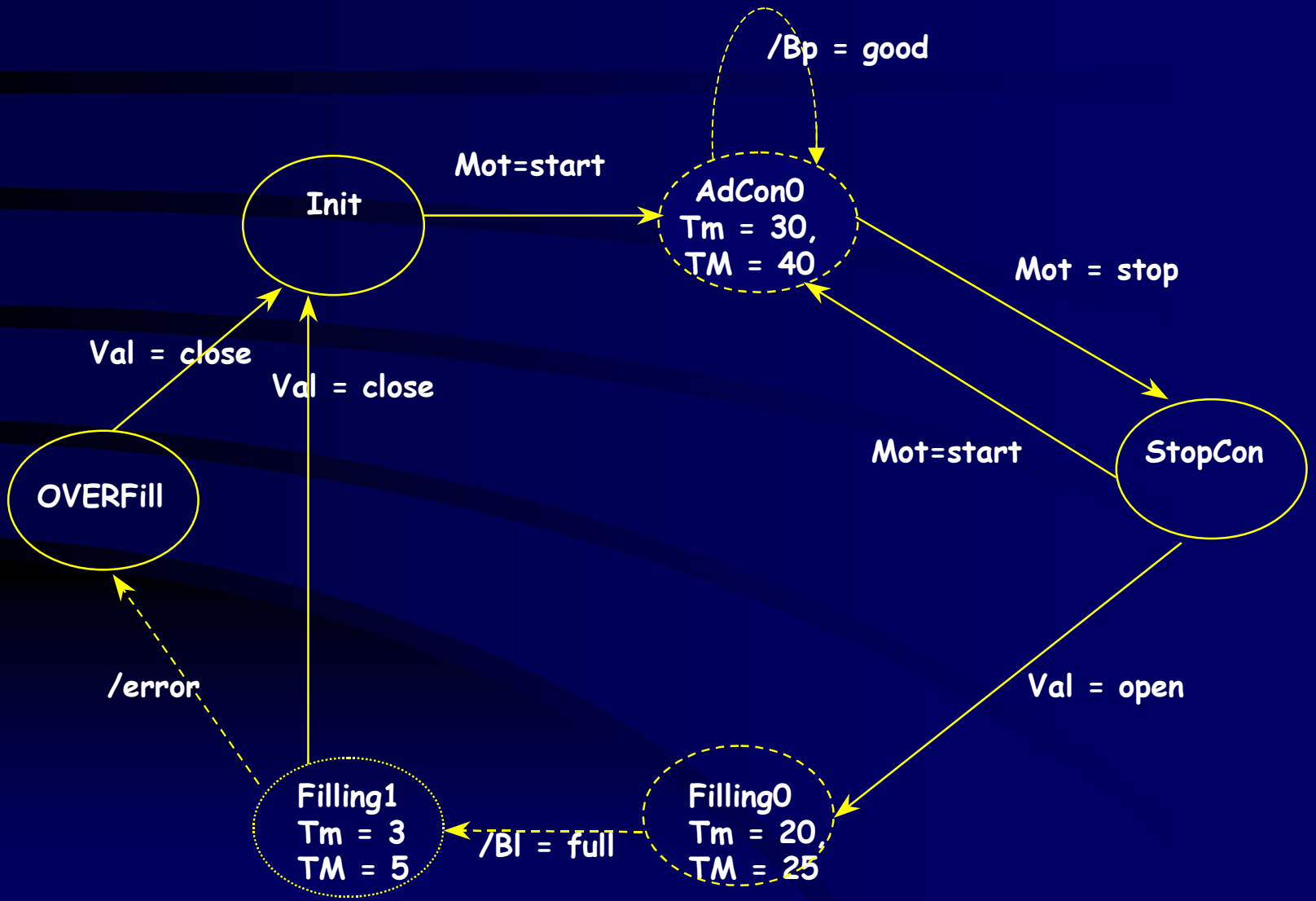
why

The life-time of transient states can not be known with accuracy

The life-time of AdCon0, σ1, depends on:
- the speed of the conveyor,
- the length between the barrels.



Behavioral Model of the Filling System with min-max life-times



Example of a Methodology based on Simulation proof

1 - built the min-max DEVS model of the system to be control,

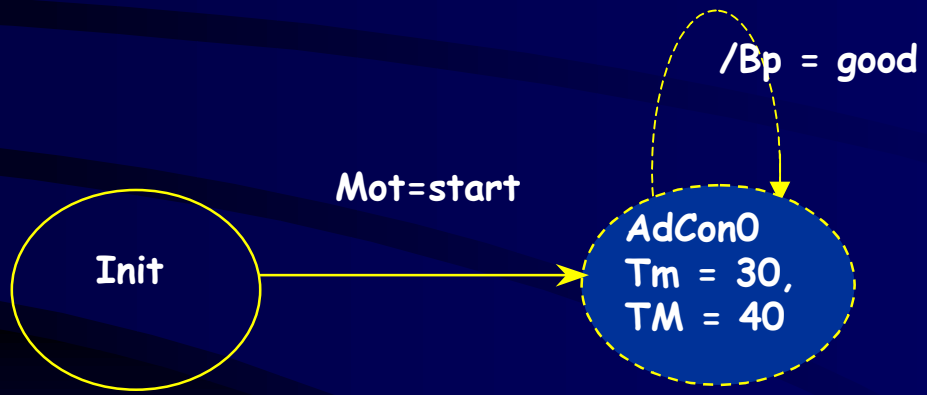
2- built the T.A representing the H-L specification of the control,

3- built the DEVS model of the control implementation,

4- prove that any behavior that can be exhibited by the DEVS model can also be exhibited by the High-level specification.

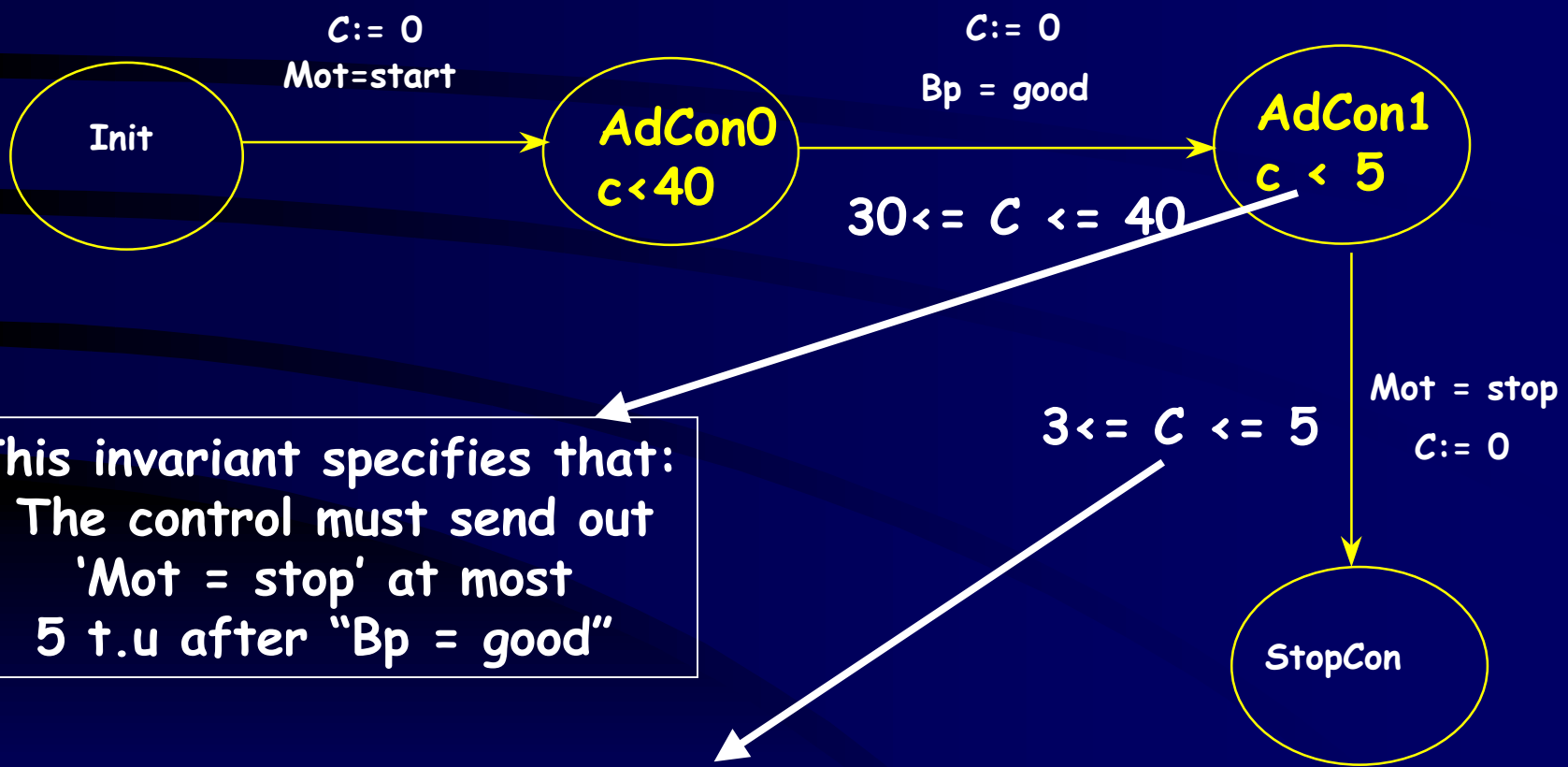
Using the min-max DEVS model we can built the T.A representing the H-L specification of the control system

Looking at the min-max model of the system and at the requirements for the control, we can built the specification of its control by a T.A expressing the timing constraints between events



Between (Mot=start) and (Bp=good), there is at least 30 t.u and at most 40 t.u

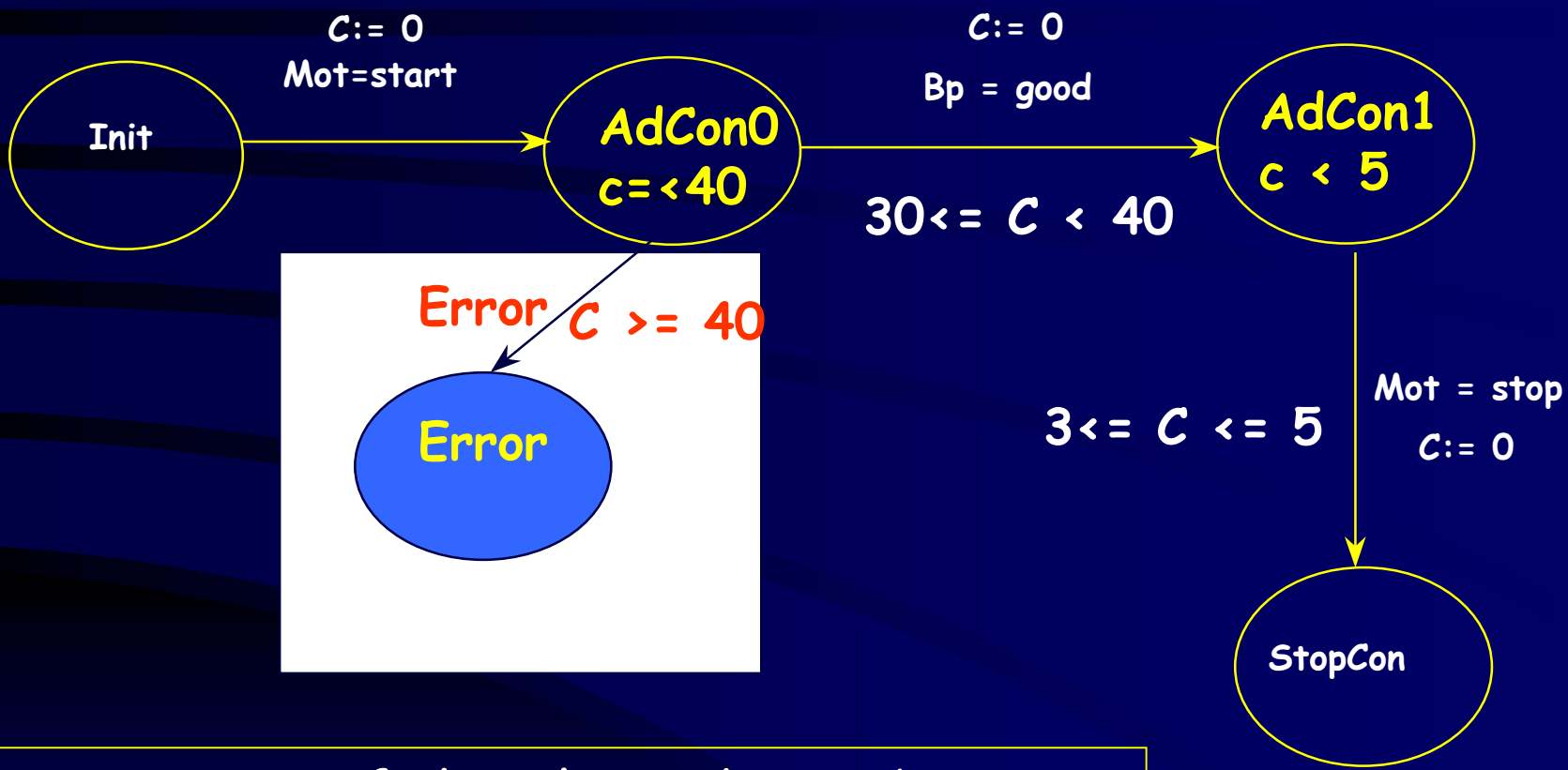
We obtain the following T.A for the high level specification of the control system



This invariant specifies that:
The control must send out
'Mot = stop' at most
5 t.u after "Bp = good"

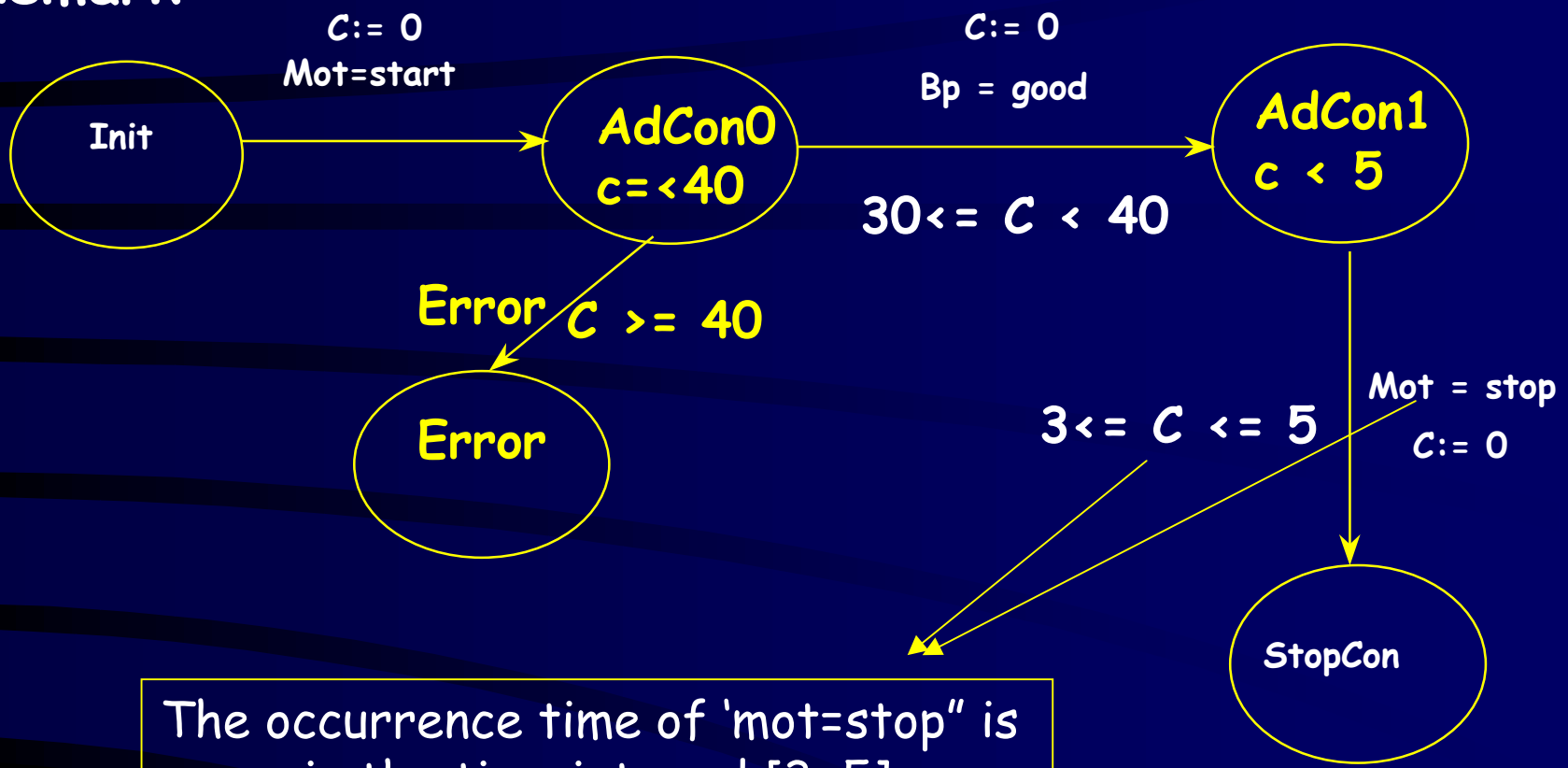
This guard specifies that: The control must not send out
'Mot = stop' before 3 t.u after "Bp = good"

Knowing the min-max behavior of the system
we can add control states to detect erroneous behaviors
of the controlled system



error message if the elapsed time between
"Mot = Start" and "Bp = good"
is greatest or equal than 40 t.u

Remark



The occurrence time of 'mot=stop" is in the time interval [3 ,5]

The model is not time-deterministic and can not be simulate, but it expresses time constraints for the control system

Example of a Methodology based on Simulation proof

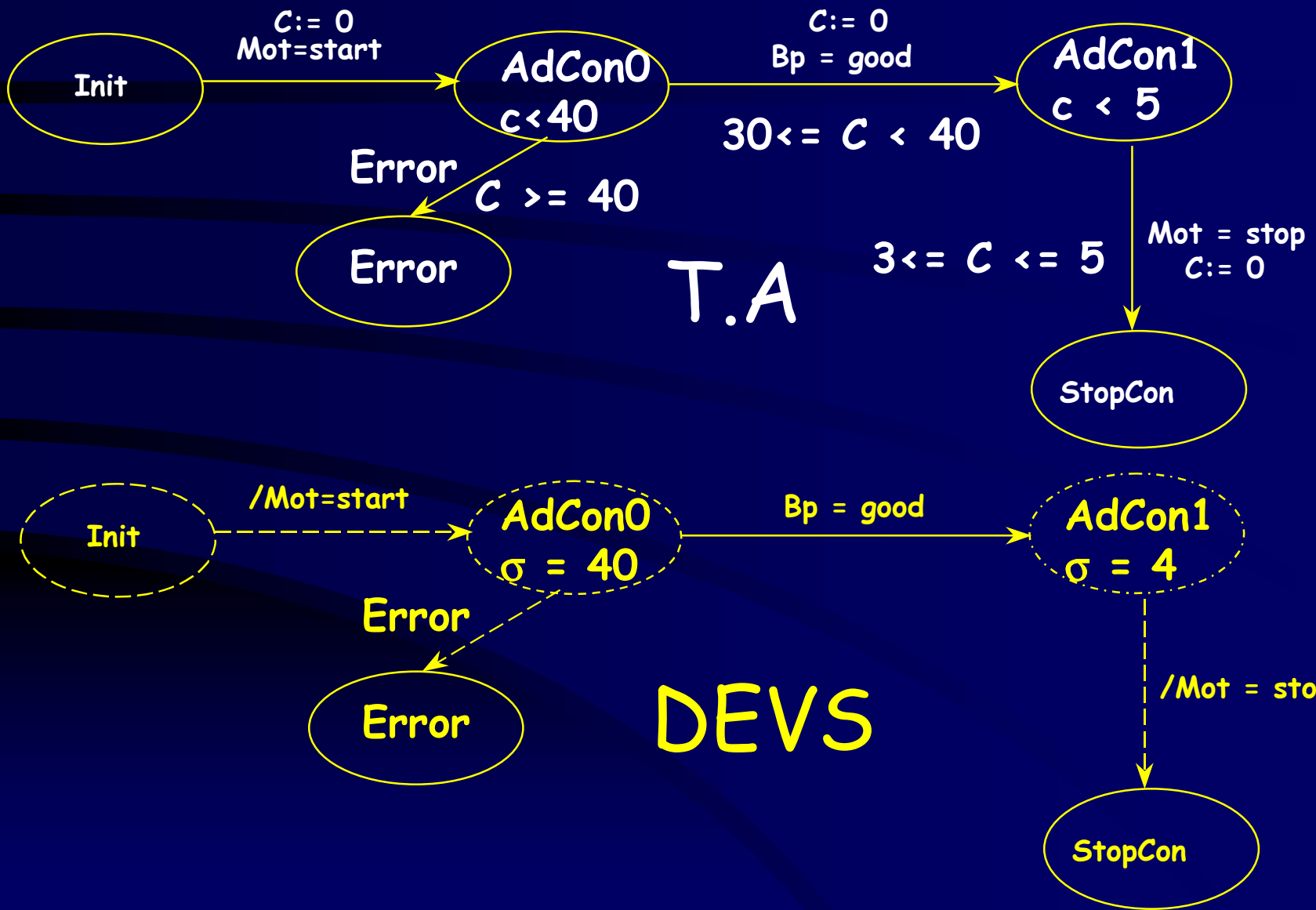
1 - built the min-max DEVS model of the system to be control,

2- built the T.A representing the H-L specification of the control,

3- built the DEVS model of the control implementation,

4- prove that any behavior that can be exhibited by the DEVS model can also be exhibited by the High-level specification.

From the high level specification, we can built the DEVS model of a possible implementation of the control system.



Example of a Methodology based on Simulation proof

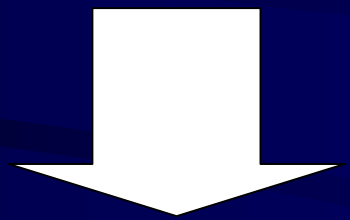
1 - built the min-max DEVS model of the system to be control,

2- built the T.A representing the H-L specification of the control,

3- built the DEVS model of the control implementation,

4- prove that any behavior that can be exhibited by the DEVS model can also be exhibited by the High-level specification.

Using a DEVS simulator, we can show that exists a simulation relationship between the high-level specification by the T.A and the DEVS model of the control part



To establish this simulation proof
we must simulate all the possible input sequences
of the DEVS model

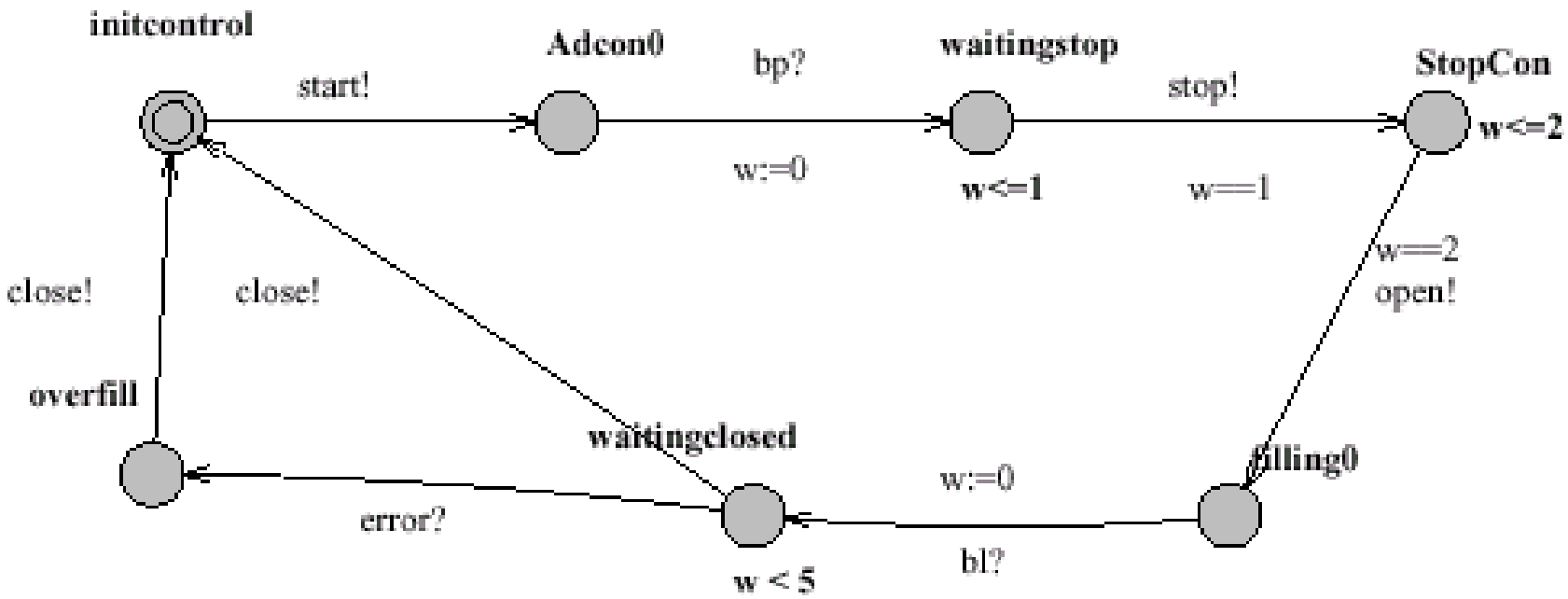
But, It is also possible to simulate only
typical scenarios

Other approach: Composition of T.A

The DEVS model of the filling system is transformed into a T.A. The control system is specified by a T.A

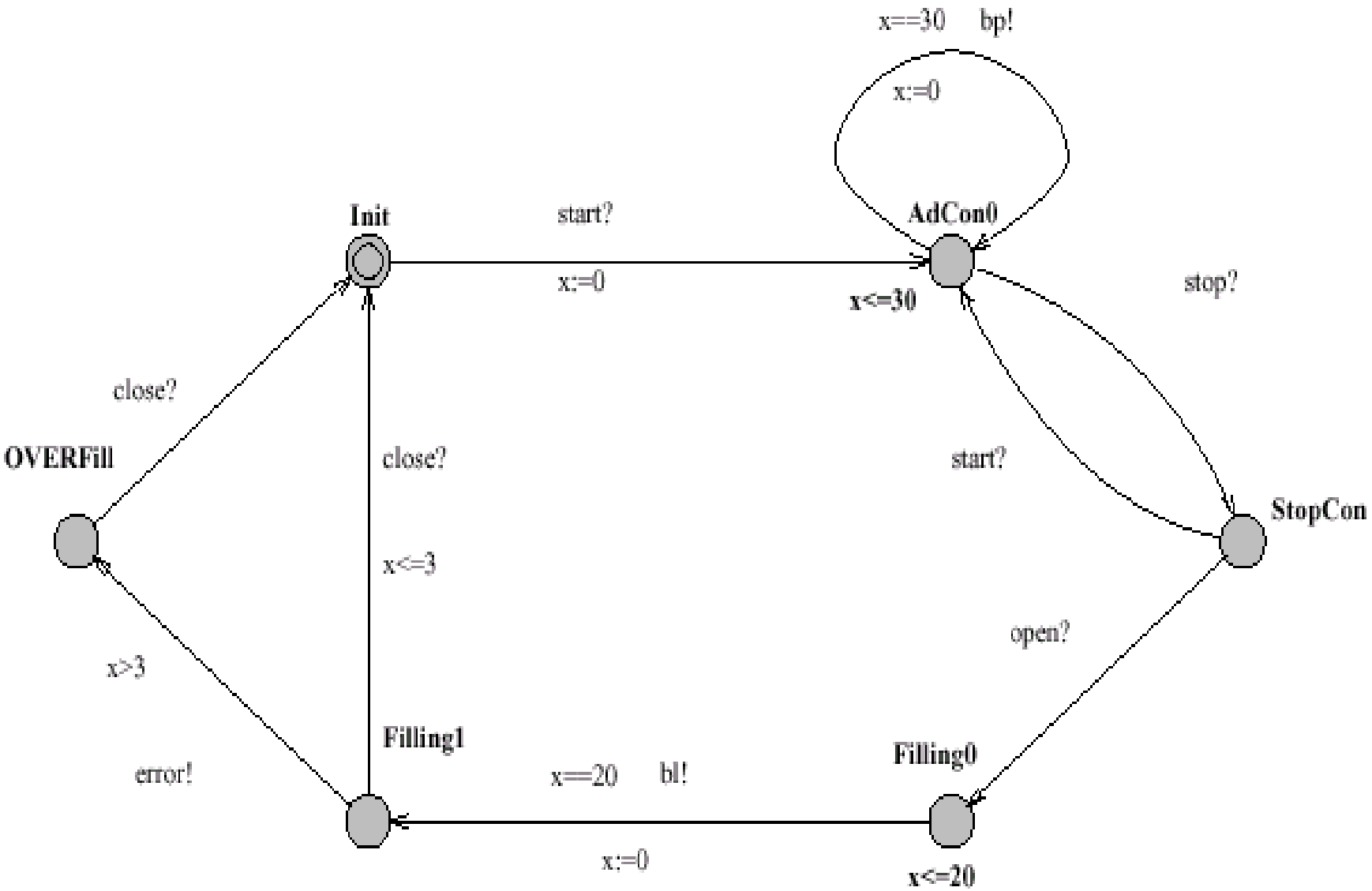
Using a tool as UPPAAL, we can verify some properties of the automaton built by the composition of the 2 T.A

Using UPPAAL for formal verification
1st approach Composition of T.A



T.A for the control of the filling system

T.A of the filling system obtained from the DEVS model



Conclusion and some ways to explore

a DEVS model is deterministic, a TA not necessarily,

a TA has a finite number of discrete states, a DEVS not necessarily

Transformation of an DEVS into a T.A implies to only consider DEVS with a finite number of discrete states

Transformation of a T.A into a DEVS implies to only consider deterministic T.A

Conclusion and some ways to explore

Min-max DEVS seems more close to T.A than classical DEVS

Formal Transformation method for min-max DEVS into T.A
and
Formal verification methods from T.A world to min-max DEVS

Conclusion and some ways to explore

Diffusion in Academic environments

Develop works on transformation of DEVS into TA, and T.A into DEVS

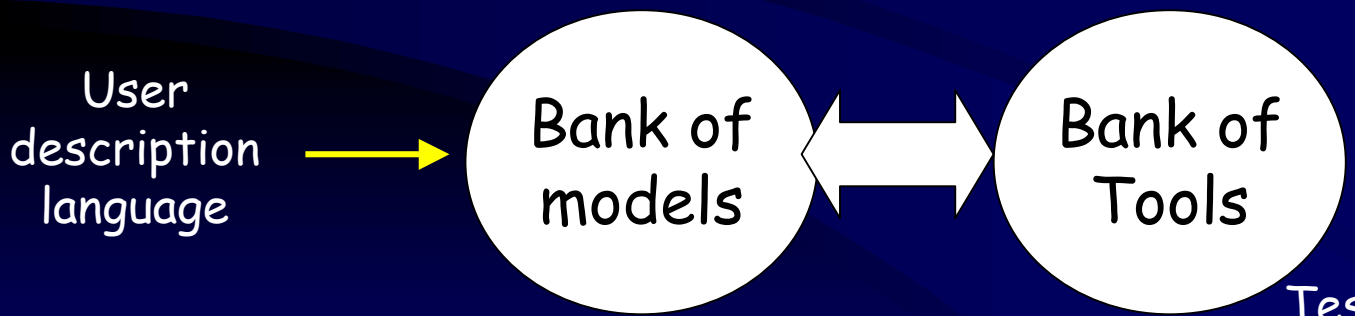
Propose design methodologies with formal verification based on DEVS simulation

Diffusion in Industrial environments

Do not forget the need of simple and domain-oriented languages to bring users in the DEVS world

Domain-oriented description languages for DEVS models

Description languages independent of problem-solving tools which can be used on the description



CONCLUSION OF THE CONCLUSION