Petri net models within distributed embedded controller design

Luis Gomes

Univ. Nova de Lisboa – Faculty of Sciences & Technology &

UNINOVA - CTS, Portugal

lugo@fct.unl.pt

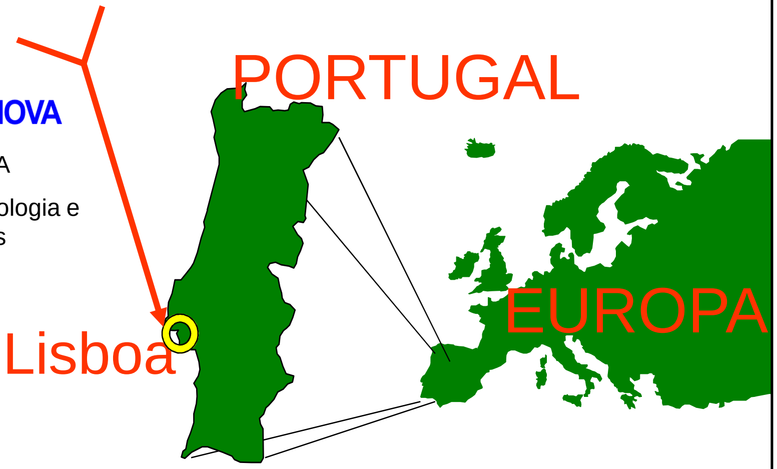
Universidade Nova de Lisboa

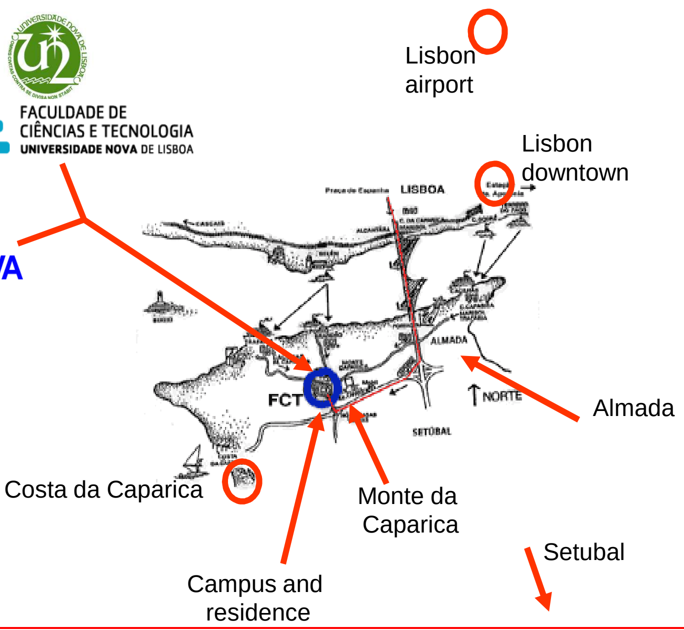
Faculdade de Ciências e Tecnologia

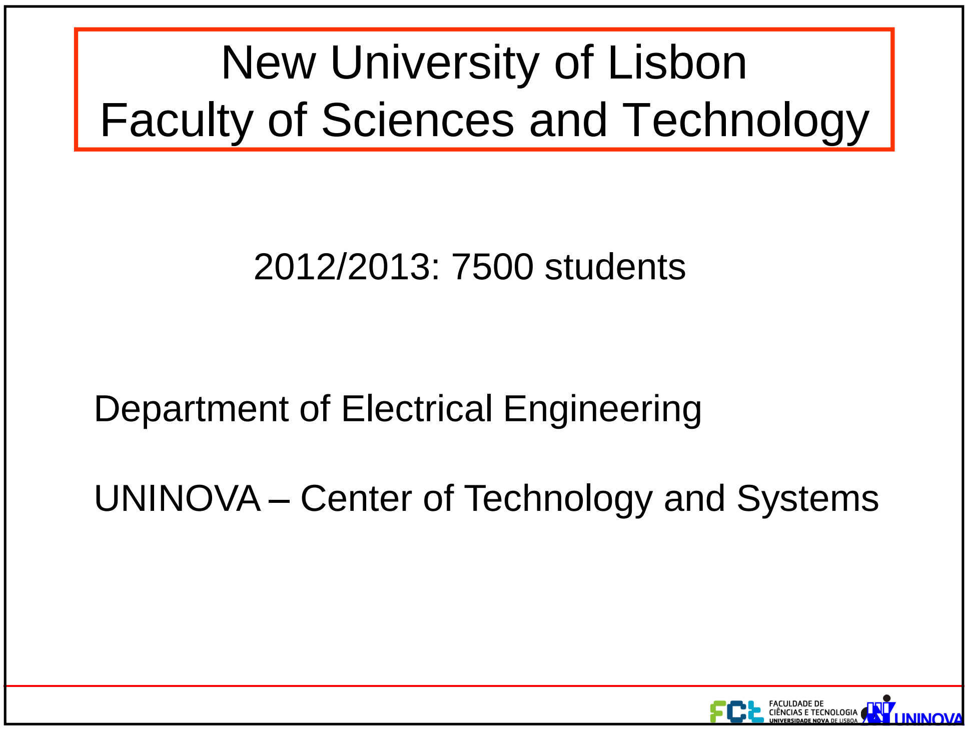
Departamento de Engenharia Electrotécnica

UNINOVA

Centro de Tecnologia e Sistemas







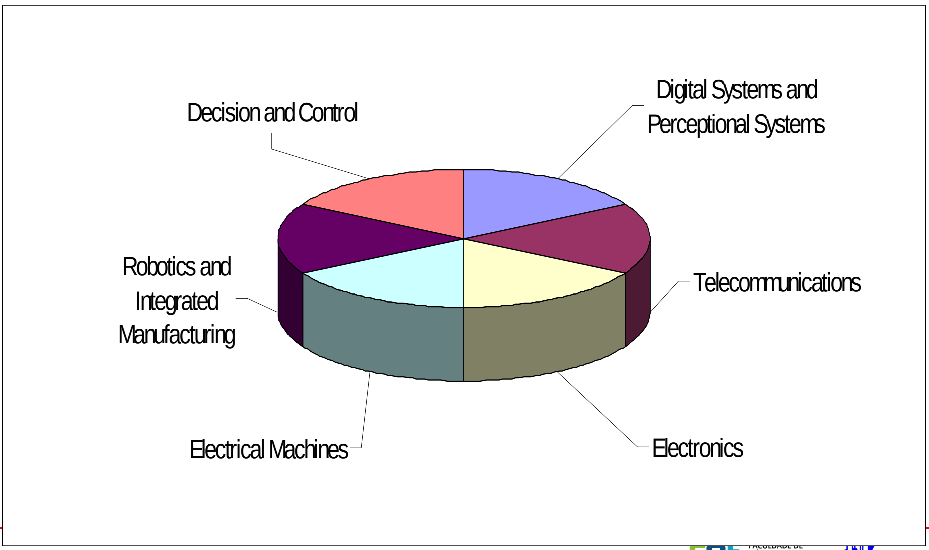
New University of Lisbon Faculty of Sciences and Technology

2012/2013: 7500 students

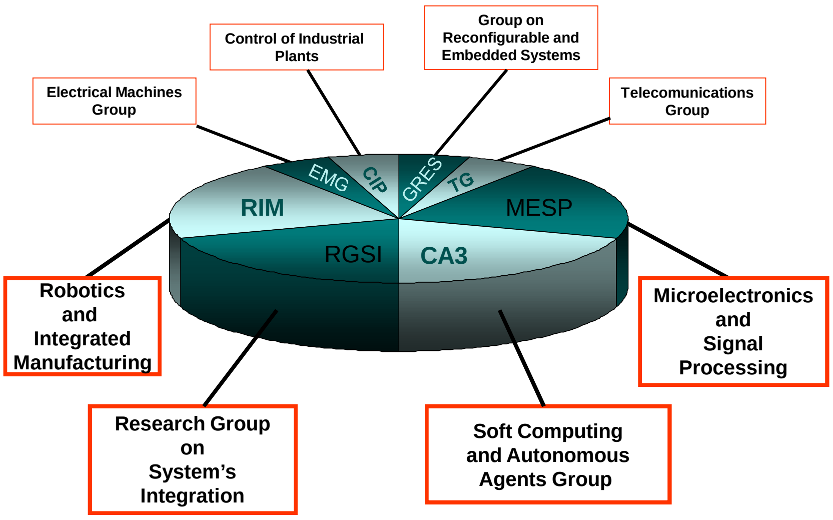
Department of Electrical Engineering

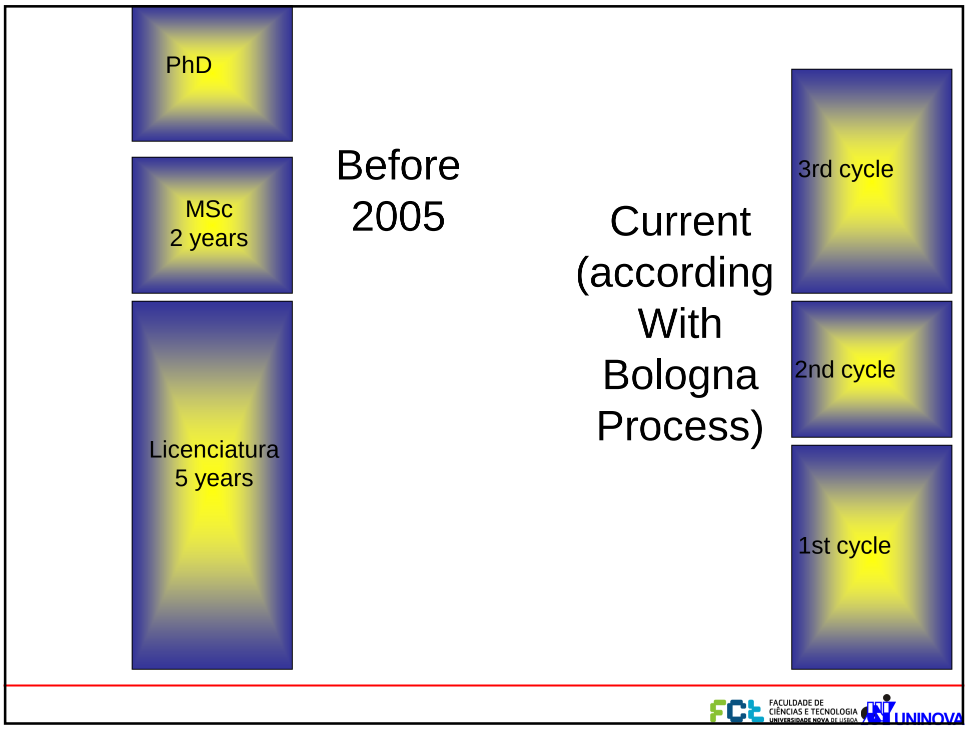
UNINOVA – Center of Technology and Systems

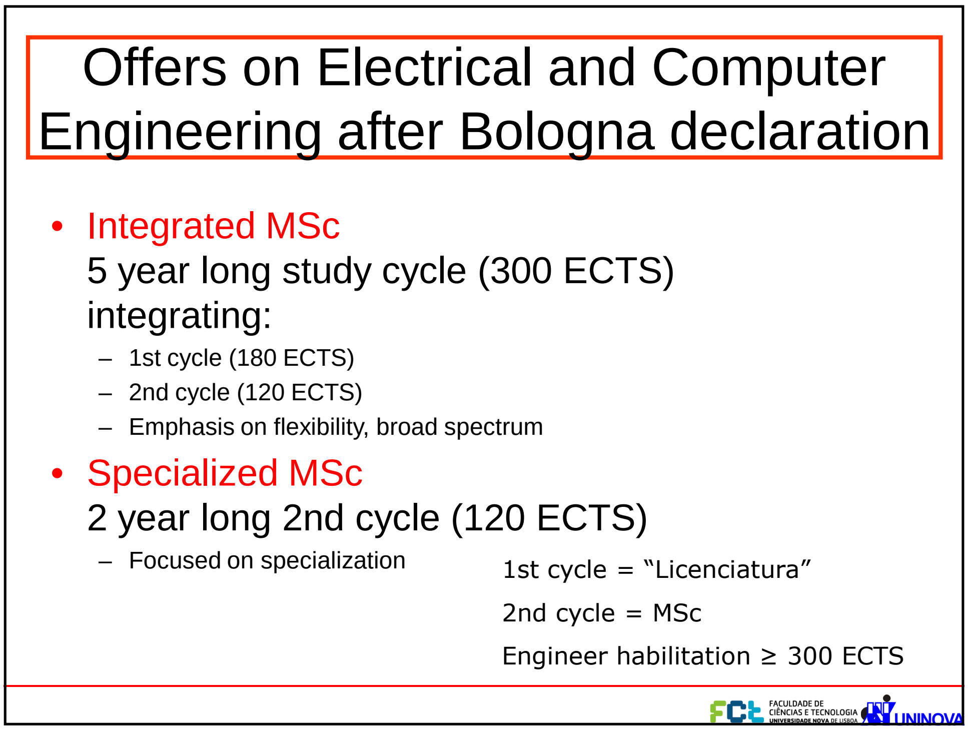
Department Structure



Research Groups


Control of Industrial Plants
Group on Reconfigurable and Embedded Systems
Electrical Machines Group
Telecomunications Group
EMG
CIP
GRES
TG
RIM
MESP
RGSI
CA3
Robotics and Integrated Manufacturing
Microelectronics and Signal Processing
Research Group on System's Integration
Soft Computing and Autonomous Agents Group





Offers on Electrical and Computer Engineering after Bologna declaration

- Integrated MSc
  5 year long study cycle (300 ECTS) integrating:
  - 1st cycle (180 ECTS)
  - 2nd cycle (120 ECTS)
  - Emphasis on flexibility, broad spectrum
- Specialized MSc
  2 year long 2nd cycle (120 ECTS)
  - Focused on specialization

1st cycle = "Licenciatura"

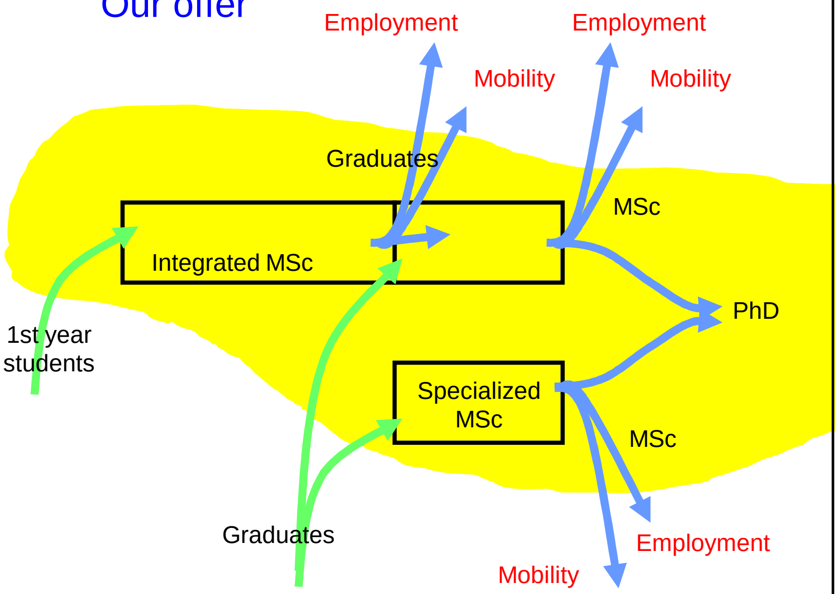
2nd cycle = MSc

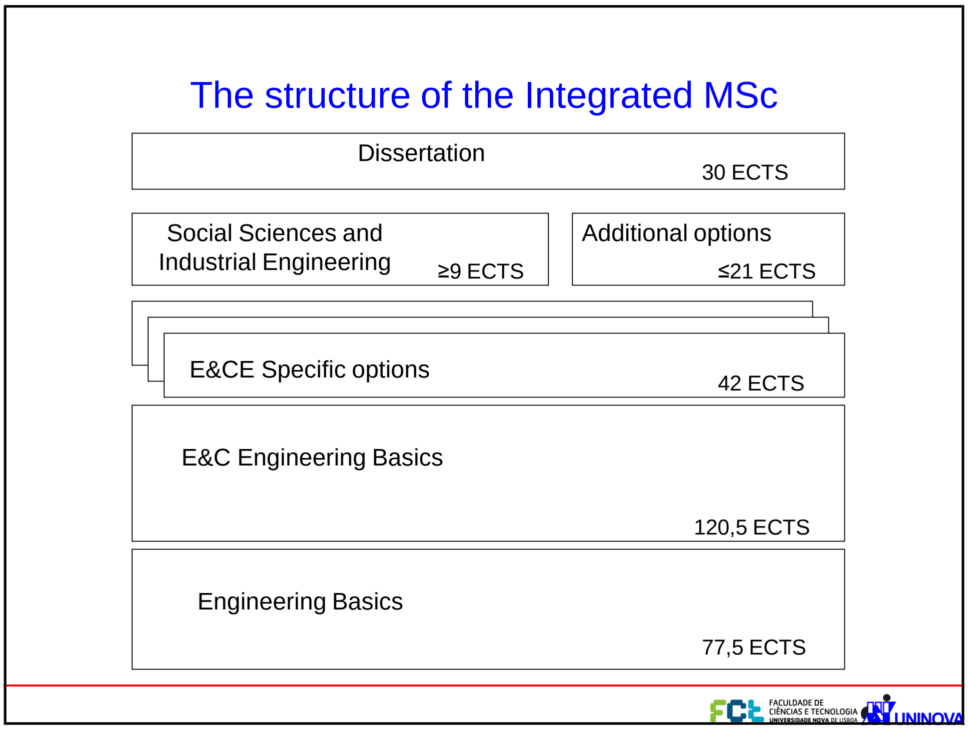
Engineer habilitation ≥ 300 ECTS

Offers on Electrical and Computer Engineering after Bologna declaration

- Integrated MSc on Electrical and Computer Engineering
- MSc on Electrical, Systems and Computer Engineering
  - Five current specializations:
    - Electric Energy and Automation
    - Microelectronics and Digital Systems
    - Corporate Cooperative Networks
    - Robotics and Sensorial Systems
    - Telecomunications

Our offer





The structure of the Integrated MSc

Dissertation 30 ECTS

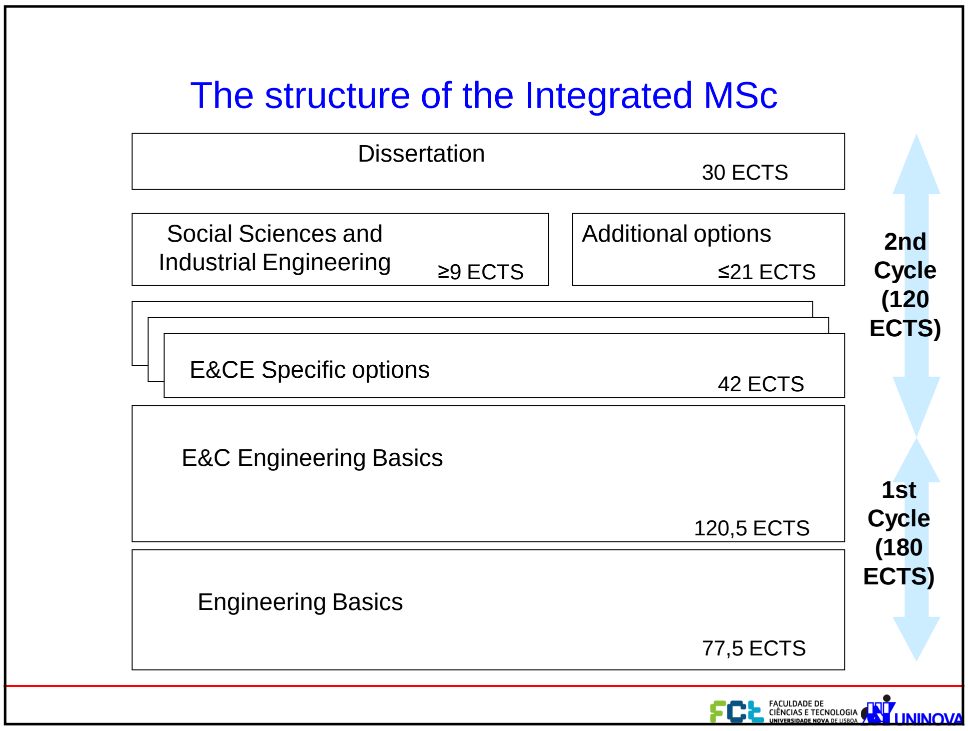
Social Sciences and Industrial Engineering ≥9 ECTS

Additional options ≤21 ECTS

E&CE Specific options 42 ECTS

E&C Engineering Basics 120,5 ECTS

Engineering Basics 77,5 ECTS



The structure of the Integrated MSc

Dissertation 30 ECTS

Social Sciences and Industrial Engineering ≥9 ECTS

Additional options ≤21 ECTS

E&CE Specific options 42 ECTS

E&C Engineering Basics 120,5 ECTS

Engineering Basics 77,5 ECTS

Petri net models within distributed embedded controller design

Luis Gomes

Univ. Nova de Lisboa – Faculty of Sciences & Technology &

UNINOVA-CTS, Portugal

lugo@fct.unl.pt

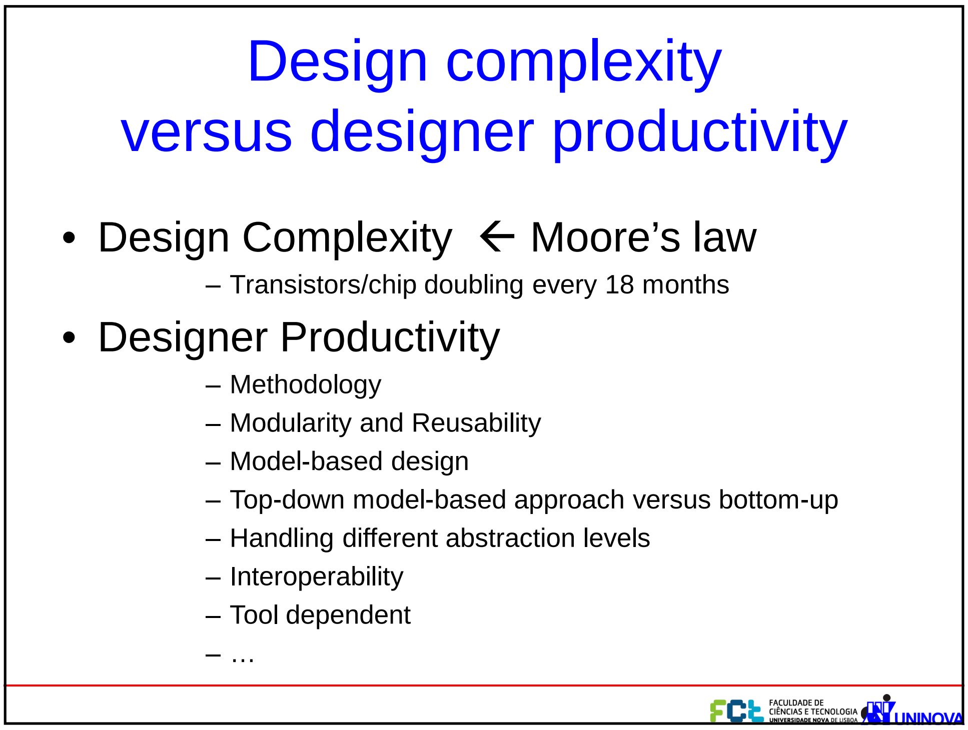
Acknowledgments to contributions from
João Paulo Barros, Anikó Costa, Filipe Moutinho, and Fernando Pereira,
members of the
Group on Reconfigurable and Embedded System
from Univ. Nova de Lisboa / UNINOVA, Portugal.

Outline

- Motivation
  - How to handle design complexity
  - Some issues and challenges
- Petri-nets for controller modeling
- Distributed Embedded Controllers Development Flow
  - Operations on nets
  - Distributed execution
  - Tools
- Sum-up

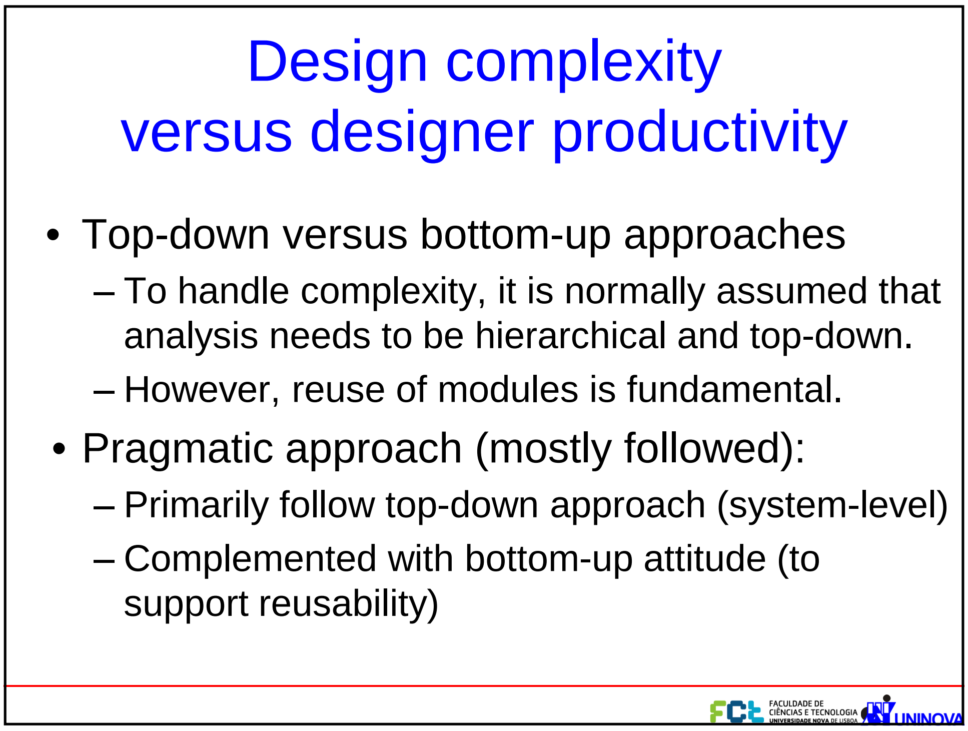
How to handle design complexity?

- For how long Moore's law will stand?
  … forever?
  - Gordon Moore, "Cramming more components onto integrated circuits", Electronics Magazine 19 April 1965:
- Sustained increase in the transistors/chip doubling every ~1 ½ years since 1959



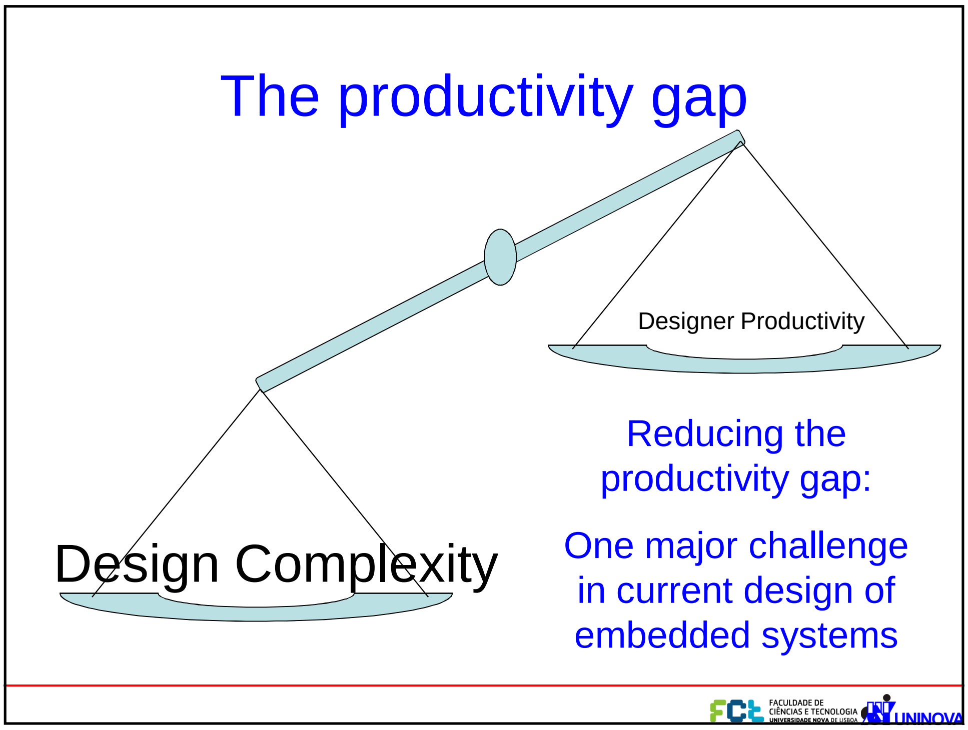
Design complexity versus designer productivity

- Design Complexity ← Moore's law
  - Transistors/chip doubling every 18 months
- Designer Productivity
  - Methodology
  - Modularity and Reusability
  - Model-based design
  - Top-down model-based approach versus bottom-up
  - Handling different abstraction levels
  - Interoperability
  - Tool dependent
  - …

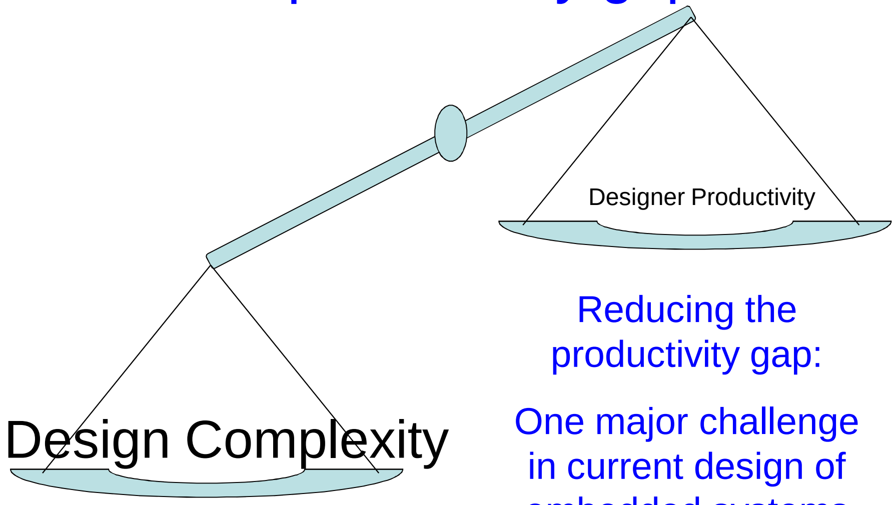


Design complexity versus designer productivity

- Top-down versus bottom-up approaches
  - To handle complexity, it is normally assumed that analysis needs to be hierarchical and top-down.
  - However, reuse of modules is fundamental.
- Pragmatic approach (mostly followed):
  - Primarily follow top-down approach (system-level)
  - Complemented with bottom-up attitude (to support reusability)

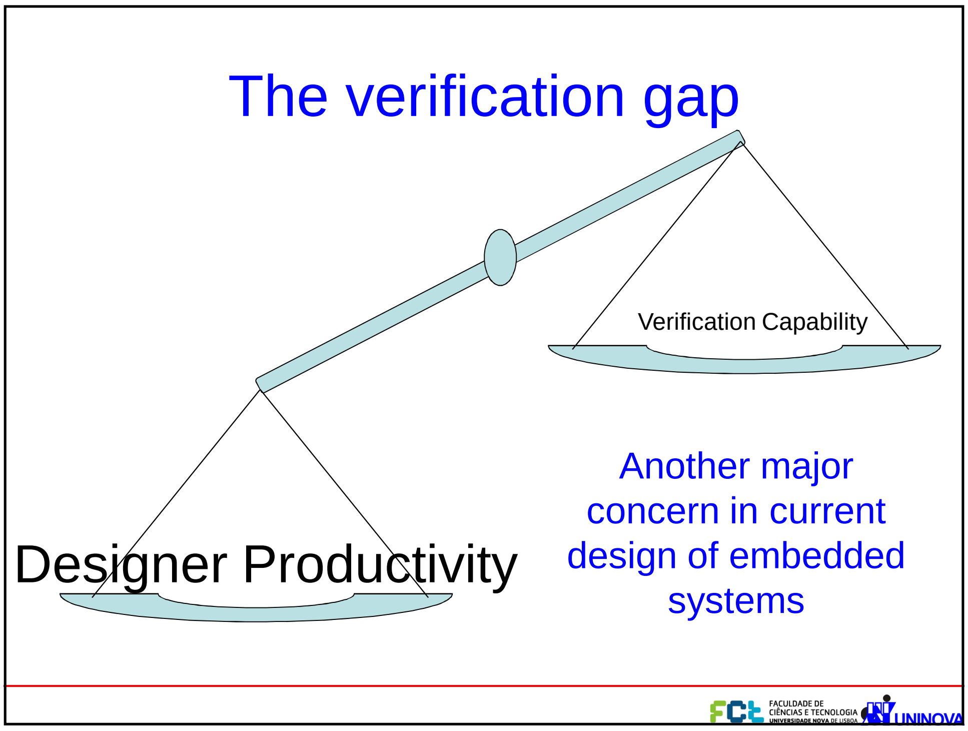


The productivity gap

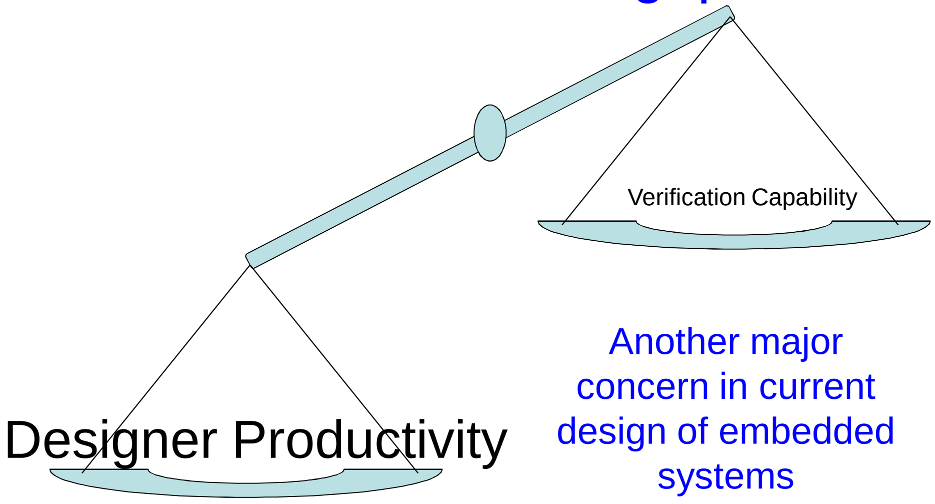


Reducing the productivity gap:

One major challenge in current design of embedded systems



The verification gap



Another major concern in current design of embedded systems

Needs to improve the design…

- If one looks into ways for:
  - improving performance,
  - reducing power consumption,
  - reducing costs,
  - reducing time-to-market,
  - reducing…
  - Improving…
- Concurrent and distributed computing and control is one major option to support improvement on several aspects.

Academic example on a concurrency class

```
var N: Integer := 0;

process P1;                  process P2;
var I: Integer;              var I: Integer;
begin                        begin
   for I := 1 to 10 do          for I := 1 to 10 do
        N := N + 1                   N := N + 1
end;                         end;
```

“perfect” interleaving →

Proc.	Instr.	Reg. P1	Reg. P2	N
P1	Load N	0	0	0
P2	Load N	0	0	0
P1	Increm	0	0	0
P2	Increm	1	0	0
P1	Store N	1	1	0
P2	Store N	1	1	1
		1	1	1

From: **The Concorde Doesn’t Fly Anymore; Moti Ben-Ari**
Keynote Talk, SIGCSE 2005, St. Louis, MO

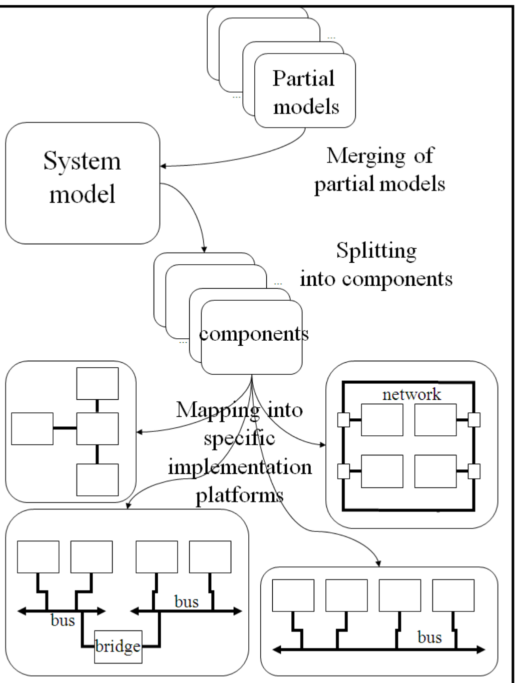
Open issues and challenges

- How to reduce the productivity gap?
- How to reduce the verification gap?
- How to support reliable distributed execution?
- Contribution to the answers:
  - Relying more and more on Model-based Development
  - Increasing usage of design automation tools (including specification, simulation/validation, verification, code generation, and test)

Moving to model-based development

- Models are used not only for describing specifications of the system at earlier phases of development, but also intended to be used along the whole development process, including automatic code generation (verification and implementation).
- Start with platform independent specification, “easily” supporting porting/implementation into specific platforms.
- For that end, an operational model having a precise execution semantics needs to be selected, allowing usage of the model at the different stages of the development process.

Model-based development : from partial models to deployment into implementation platforms


Partial models
System model
Merging of partial models
Splitting into components
components
Mapping into specific implementation platforms
network
bus
bus
bridge
bus

Selection of model formalism

- Several modeling formalisms already proved their adequacy fully supporting this model-based development flow strategy
- Considering controller design, it is common to give preference to state-based modeling formalisms due to its expressiveness capabilities.
- Also selecting an operational formalism will support the whole development cycle, including automatic code generation.

Selection of model formalism

- Among those eligible formalisms, it is worth to mention state diagrams, hierarchical and concurrent state diagrams, statecharts, and Petri nets.
- It is not a surprise that the selected formalism for this presentation is Petri nets:
  - Rigorous computational model
  - Precise execution semantics
  - Graphical representation
  - Formal representation

Outline

- Motivation
  - How to handle design complexity
  - Some issues and challenges
- Petri-nets for controller modeling
- Distributed Embedded Controllers Development Flow
  - Operations on nets
  - Distributed execution
  - Tools
- Sum-up

Petri nets for controller modeling

- Starting with autonomous classes of Petri nets…
- Extremely important to have the possibility to add dependencies to the environment under control, namely input and output signals and events.
- In those cases, those Petri nets classes become non-autonomous.
- Several classes of non-autonomous Petri nets have been referred in the literature (some having strong links with automation systems) ((Silva 1985) (David & Alla 1992) (Venkatesh, Zhou & Caudill 1994) (Hanisch & Lüder 2000) (Frey 2000) (Frey & Wagner 2006)).

The Input-Output Place-Transition Petri net class (IOPT nets)

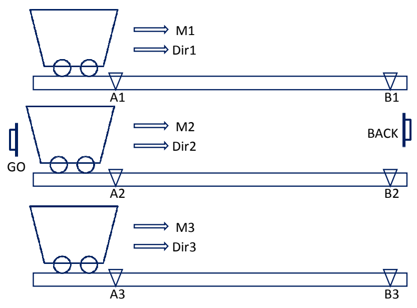
- **Extended from the Place/Transition net class (and benefiting from interpreted and synchronous net classes)**

- **Non-autonomous dependencies:**
- Input and output signals Input and output events
- Transition firing conditioned by input events and guard functions referencing input signals
- Transition firing can generate output events
- Output signals can be associated with places

- **Allows Deterministic execution:** Maximal step execution semantics

- Includes Transition priorities and Test arcs

Application example (from automation area)



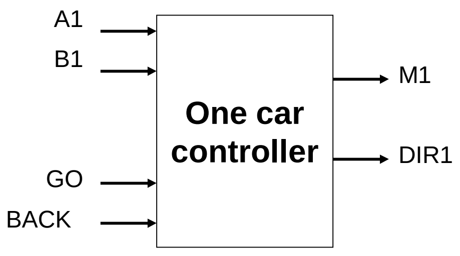
From: M. Silva, Las Redes de Petri: en la Automática y la Informática. Madrid: Editorial AC, 1985

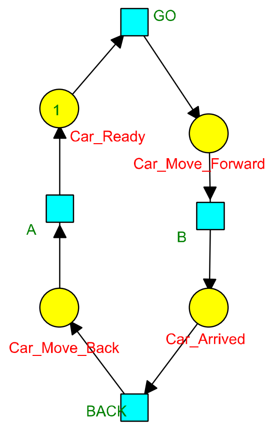
- Controller for a transportation system composed by three cars
- Cars move asynchronously but start synchronizely at both ends.

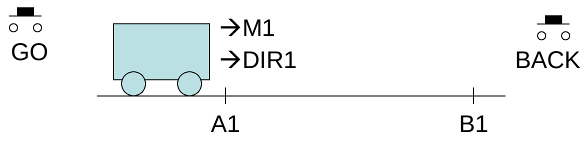
Application example

- Goal 1: to obtain a controller for one car
- Goal 2: to obtain a controller for the whole system composed by three cars
- Goal 3: to obtain a distributed controller composed by 3 controllers, one per car

Goal 1: to obtain a controller for one car



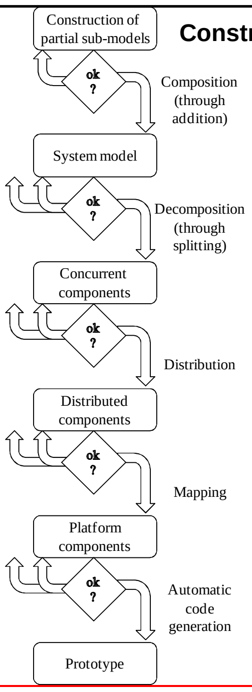




Outline

- Motivation
  - How to handle design complexity
  - Some issues and challenges
- Petri-nets for controller modeling
- Distributed Embedded Controllers Development Flow
  - Operations on nets
  - Distributed execution
  - Tools
- Sum-up

Distributed Embedded Controllers Development Flow


Construction of partial sub-models
ok ?
Composition (through addition)
System model
ok ?
Decomposition (through splitting)
Concurrent components
ok ?
Distribution
Distributed components
ok ?
Mapping
Platform components
ok ?
Automatic code generation
Prototype

Construction of partial sub-models

System model

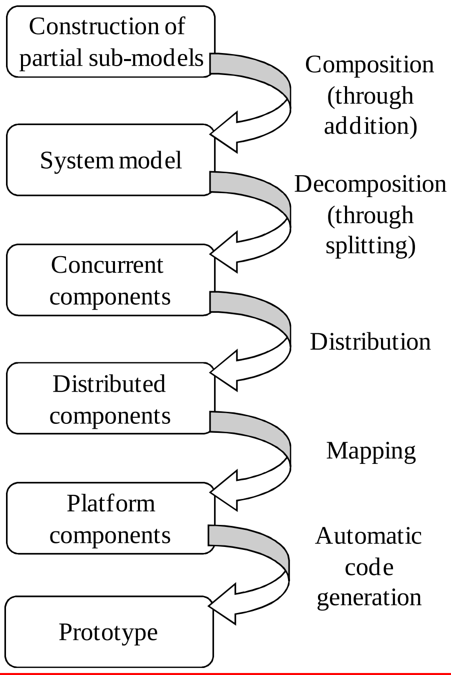
Concurrent components

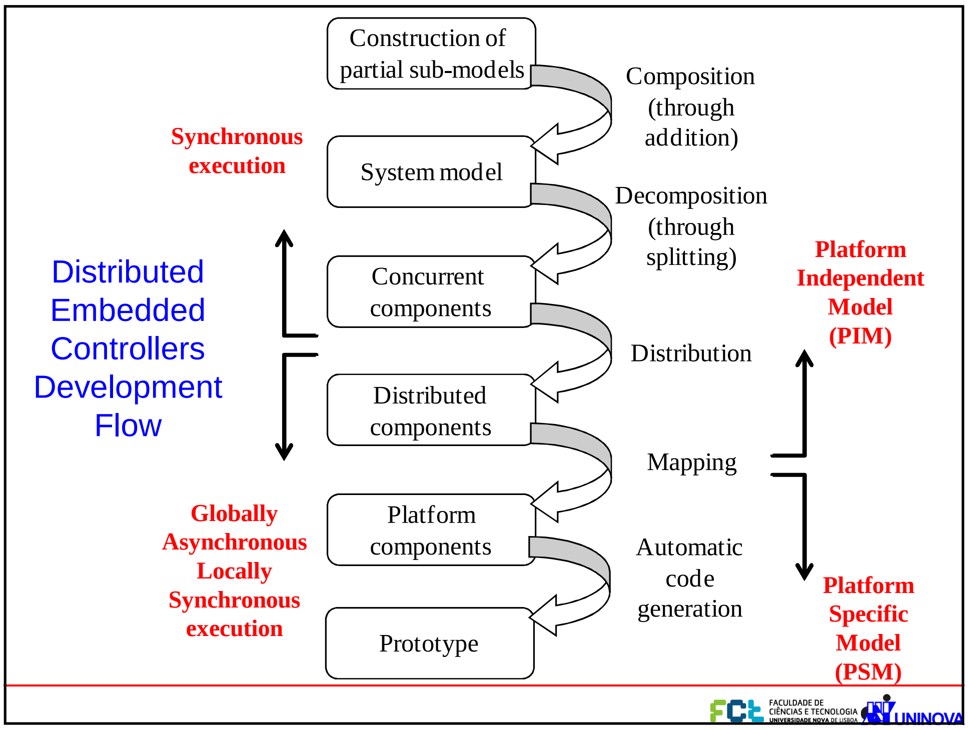
Distributed components

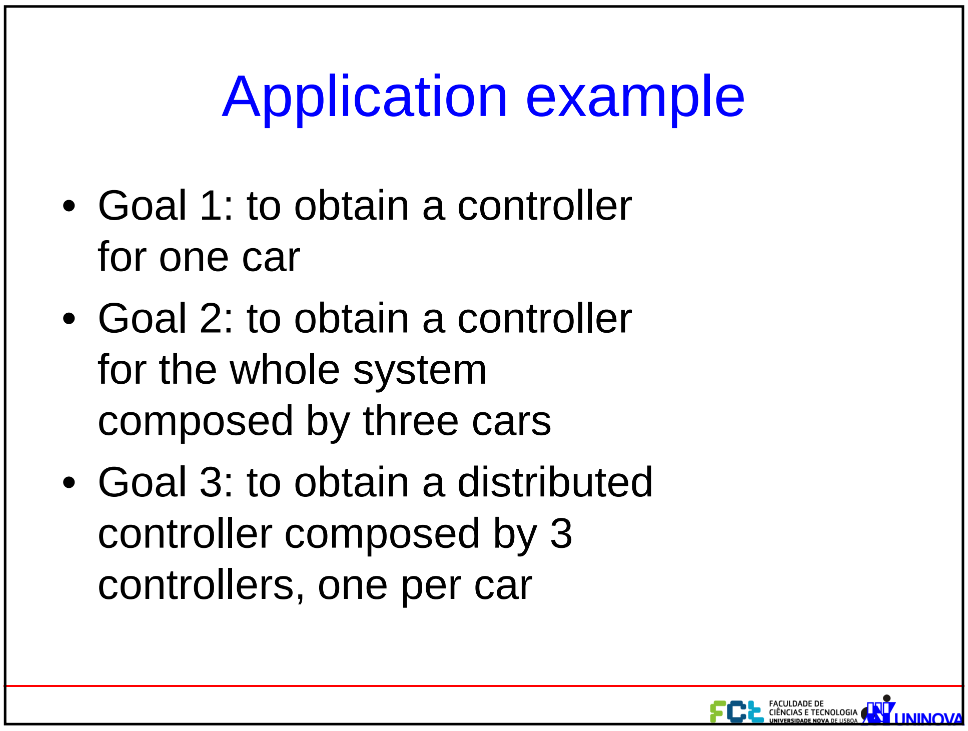
Platform components

Prototype

Distributed Embedded Controllers Development Flow


Construction of partial sub-models
Composition (through addition)
System model
Decomposition (through splitting)
Concurrent components
Distribution
Distributed components
Mapping
Platform components
Automatic code generation
Prototype

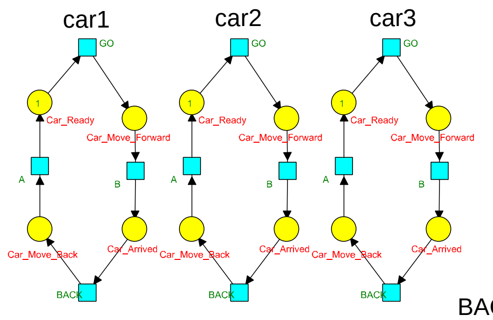

Construction of partial sub-models
Composition (through addition)
System model
Decomposition (through splitting)
Concurrent components
Distribution
Distributed components
Mapping
Platform components
Automatic code generation
Prototype
Synchronous execution
Distributed Embedded Controllers Development Flow
Globally Asynchronous Locally Synchronous execution
Platform Independent Model (PIM)
Platform Specific Model (PSM)

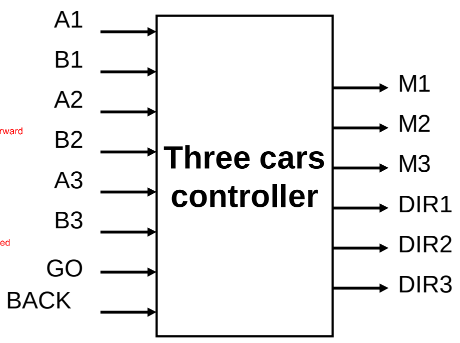


Application example

- Goal 1: to obtain a controller for one car
- Goal 2: to obtain a controller for the whole system composed by three cars
- Goal 3: to obtain a distributed controller composed by 3 controllers, one per car

Goal 2: to obtain a controller for the whole system composed by three cars



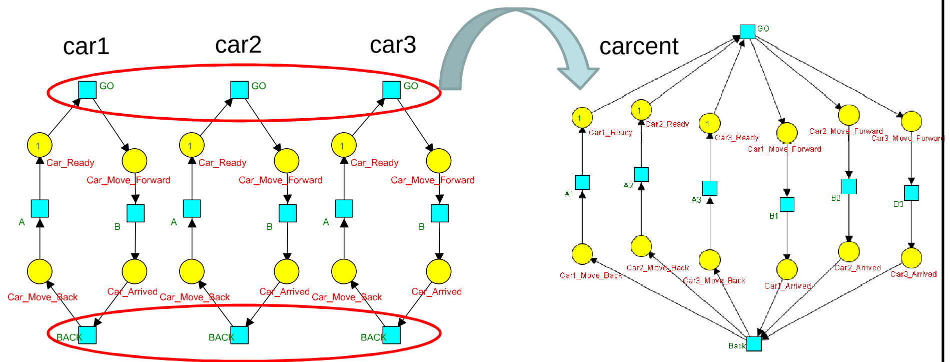


Approach: Replication of individual models (supporting reusability)
Problem: synchronizaton at both ends is not satisfied
Solution: we need to adequately compose the models

Composability of net models

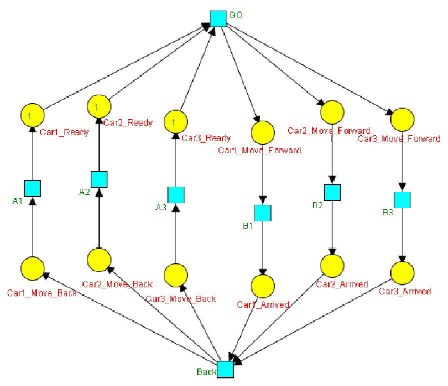
- Several solutions have been proposed
- Fusion of places (asynchronous composition)
- Fusion of transitions (synchronous composition)
- Fusion of places and transitions
- Major three steps
  - Identification of the models to compose
  - Definition of the interfaces of the models and nodes to be merged
  - Merging models

The net addition operation



carcent = car1 + car2 + car3
(car1.go/car2.go/car3.go → go,
car1.back/car2.back/car3.back → back)

What about property verification?



- Having non-autonomous Petri nets, we need to face state-space based verification techniques.

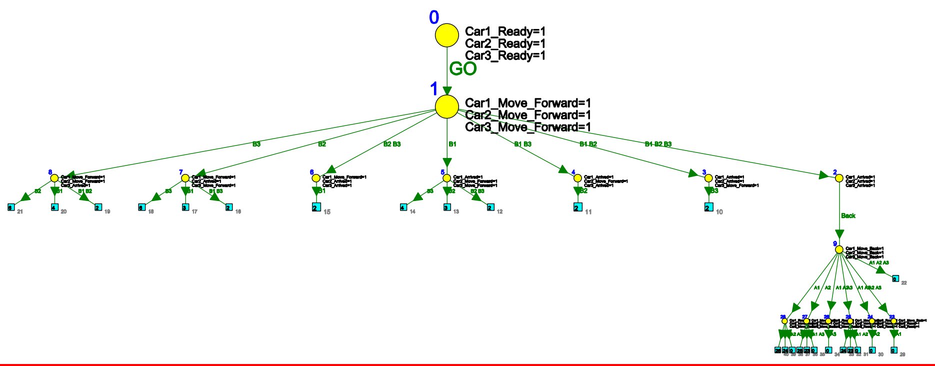
State space verification

- Plenty of tools available for verification of autonomous low-level nets.
- The number of tools shrinks if maximal step is considered as execution semantics.
- And shrinks again if non-autonomous dependencies are considered.
- Anyway, for automation systems, several tools are available.

Properties for our controller model

16 states detected; no conflict; no deadlocks
All places having as minimum marking 0 tokens
and as maximum marking 1 token

Net N_3Car_Chapter
16 (from 16) Nodes, 25 Loops, 0 Deadlocks, 0 Conflicts, Max. Depth = 6, 0 Invalid
Min Bound = [Car1_Arrived=0 Car1_Move_Back=0 Car1_Move_Forward=0 Car1_Ready=0 Car2_Arrived=0 Car2_Move_Back=0 Car2_Move_Forward=0 Car2_Ready=0 Car3_Arrived=0 Car3_Move_Back=0 Car3_Move_Forward=0 Car3_Ready=0]
Max Bound = [Car1_Arrived=1 Car1_Move_Back=1 Car1_Move_Forward=1 Car1_Ready=1 Car2_Arrived=1 Car2_Move_Back=1 Car2_Move_Forward=1 Car2_Ready=1 Car3_Arrived=1 Car3_Move_Back=1 Car3_Move_Forward=1 Car3_Ready=1]

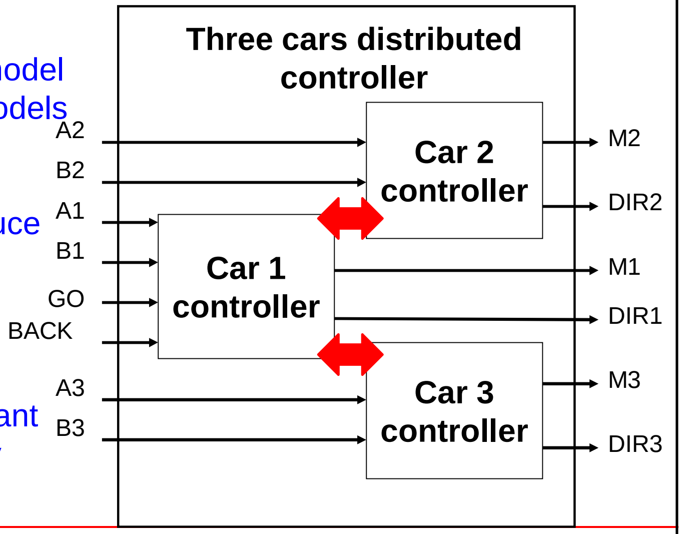


Goal 3: to obtain a distributed controller composed by 3 controllers, one per car

Approach:
Decompose the model into concurrent models

Problem:
we need to introduce **communication** between sub-models

Constraint: we want to assure property preservation



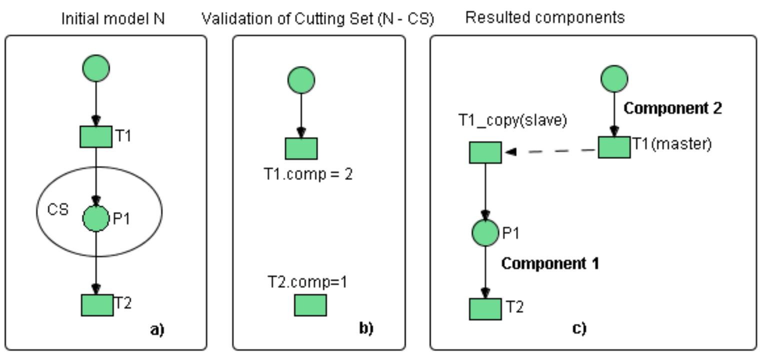
The net splitting operation

- Decomposition into a set of concurrent models, which (whenever executed according with synchronous paradigm) will preserve properties.
- Usage of directed synchronous channels to communicate among components synchronizing transition synchrony sets.
  - One master transition (responsible for the firing of the synchrony set);
  - One or more slave transitions
- Concurrent models are amenable to support distributed execution of the Petri net model (in a later stage)

The net splitting operation

- Identifying the nodes (the cutting set) where the model should be broken.
- The nodes defined as cutting set have to be validated.
- Once defined a valid cutting set, the result sub-models can be obtained applying three rules, depending on the cutting node:
  - Rule#1, cutting node is a place
  - Rule #2, cutting node is a transition with incoming arcs only from one component
  - Rule #3, cutting node is a transition with incoming arcs from more than one component

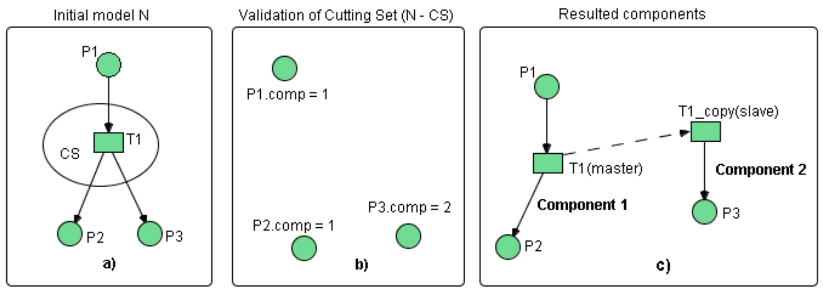
Rule1 – Splitting by place



Initial model
(only referring the locality associated with the cutting node)

Component models

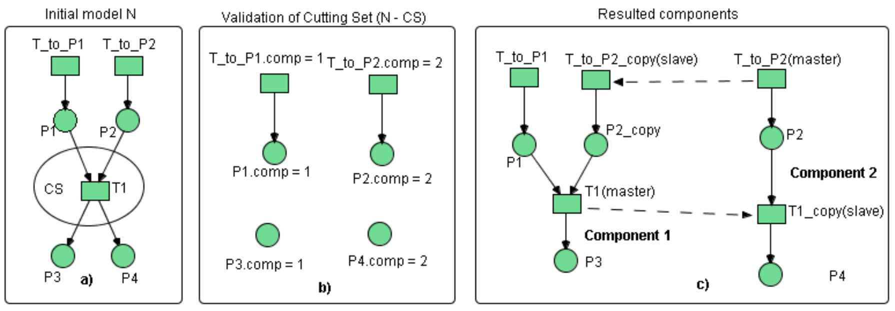
Rule2 – Splitting by transition with incoming arcs only from one component



Initial model
(only referring the locality
associated with the cutting node)

Component models

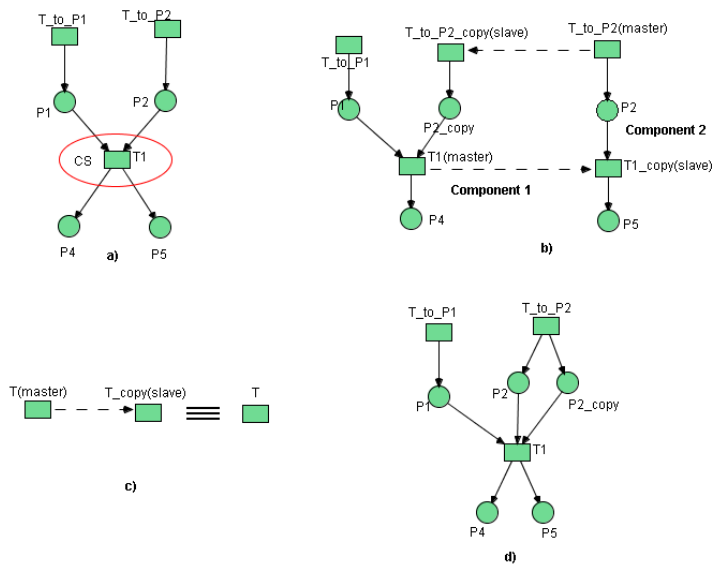
Rule3 – Splitting by transition with incoming arcs from more than one component



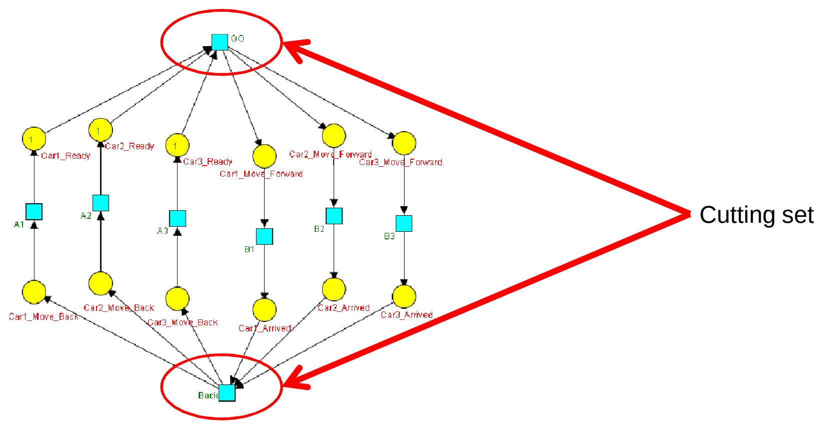
Initial model
(only referring the locality
associated with the cutting node)

Component models

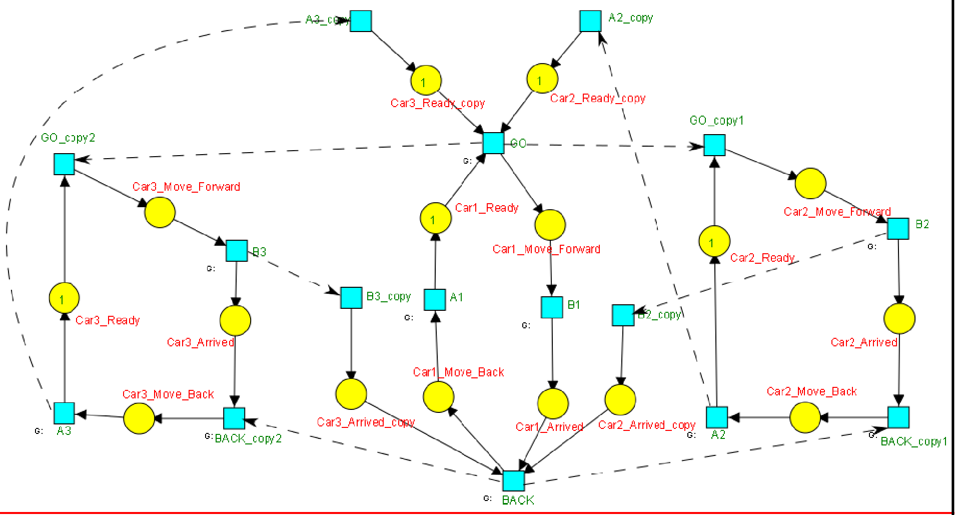
Properties are preserved!



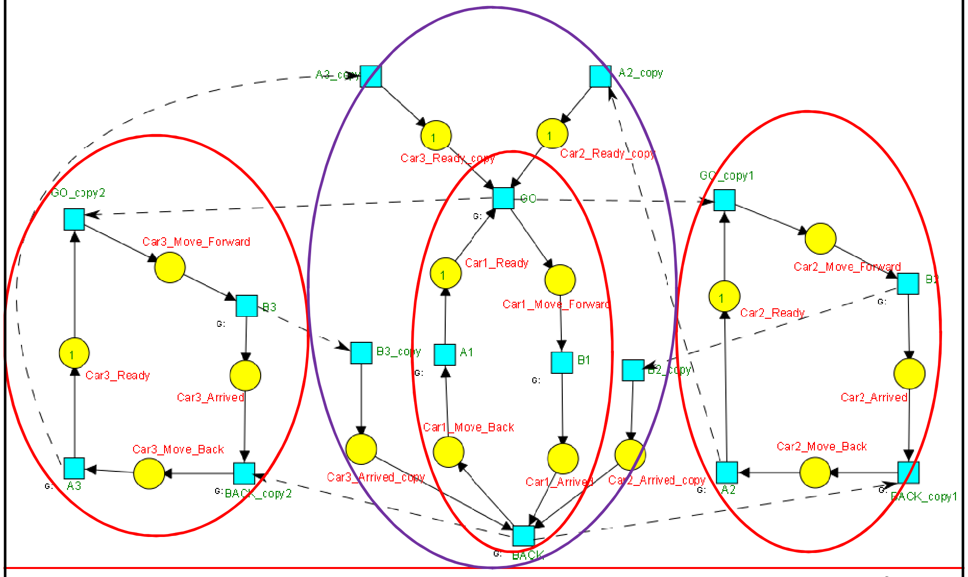
Splitting our model



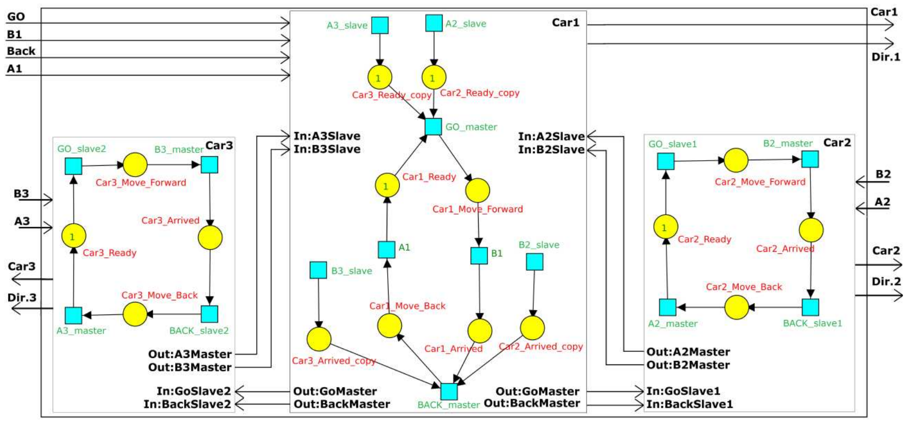
Concurrent sub-models



Concurrent sub-models



Concurrent sub-models implementation view



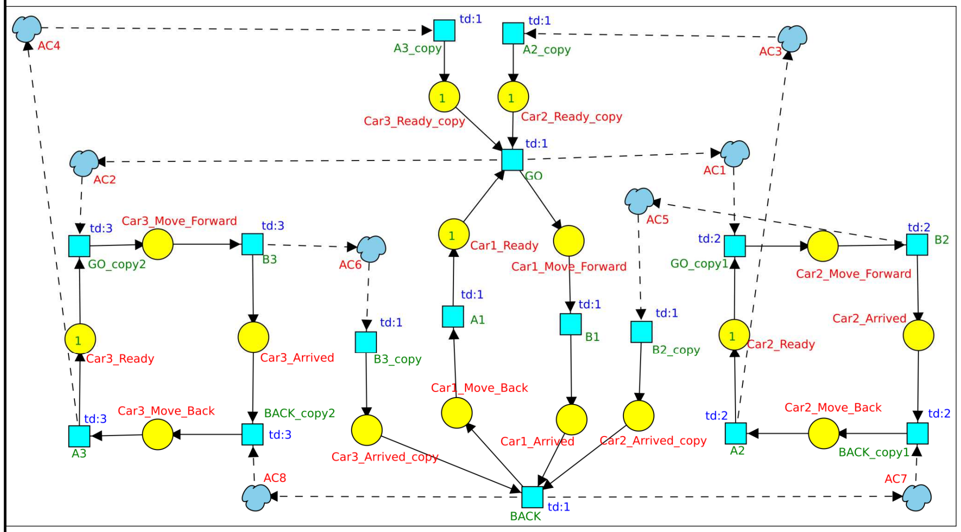
Facing distributed implementations - I

- When global execution of the model is not viable anymore, and the system needs to be seen as a collection of parallel components.
- We need to move away from the synchronous paradigm (where one global tick / execution step is considered) and need to face globally asynchronous locally synchronous (GALS) execution semantics.
- Maximal step execution semantics needs to be kept in each component.

Facing distributed implementations - II

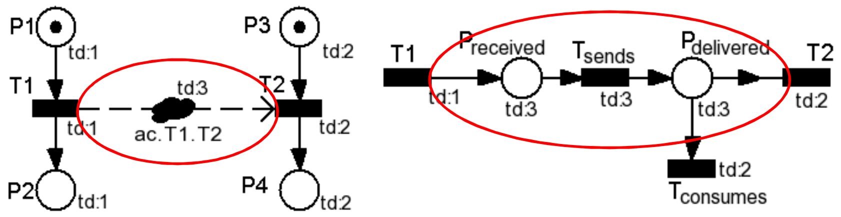
- Approach:
  - Definition of time domains (each time domain has its own tick / execution step)
  - Each component will be associated with one different time domain
  - Communication channels have a place semantics (holding non-instantaneous pending communication)
  - Each directed synchronous channel will be replaced by a directed asynchronous channel, where master transition, slave transitions, and the channel itself are associated with different time domains.

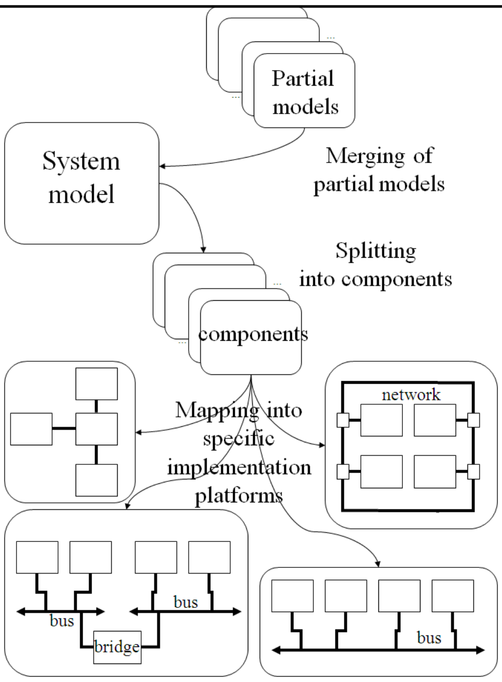
Distributed sub-models



Coming back to property verification

- Property verification still possible based on state space construction.
- Behavioral model for the asynchronous channels needs to be used, complemented by interleaving execution between all time domains (each of them having a maximal execution step), assuring GALS evolution





Partial models
System model
Merging of partial models
Splitting into components
components
Mapping into specific implementation platforms
network
bus
bridge
bus
bus

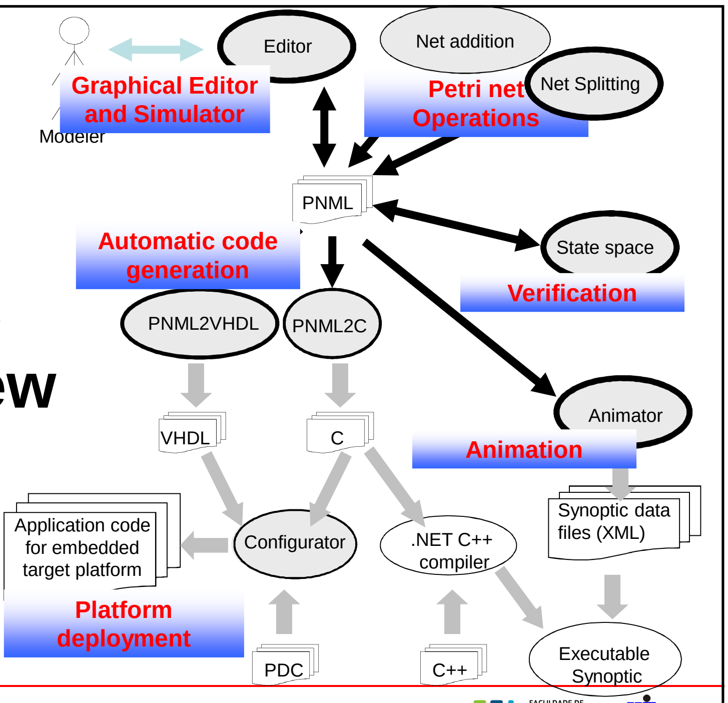
Configuring communication layers

- Using the presented Petri net-based distributed embedded controllers development flow is possible to check a-priori impact of using different types of communication support between components.
- The maximum number of messages that each Asynchronous-Channel may need to buffer can be determined through analysis of associated state space (determining maximum bound of associated places).

Tools

- Petri nets already have a set of supporting tools mostly covering specification and verification.
- Petri nets need additional tools, mostly covering automatic code generation, to be fully integrated in engineering development flows.
- A contribution (for IOPT nets) is available at http://gres.uninova.pt/

Tools overview



IOPT-Tools: Previous tool-chain

- Model edition (*Snoopy IOPT Petri Net editor*)
- User interface/synoptic/animation editor
- Automatic controller C code generator
- Automatic controller VHDL synthesis
- Automatic GUI/synoptic C code generator
- Automatic GUI/synoptic VHDL synthesis
- Software animation and simulation tool
- Configurator tool

No IOPT state-space generators and model-checking tools

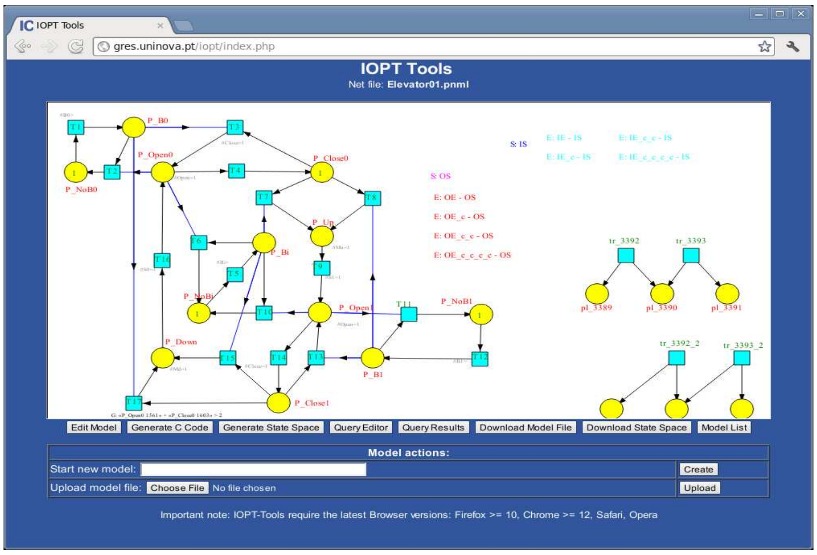
IOPT-Tools: The present

Web User Interface (http://gres.uninova.pt)
AJAX Based IOPT Petri Net Graphical Editor
Relax-NG Syntax Validation Grammar
Automatic controller C code generator
State Space Generation Tool
Model-checking using a Query System

IOPT-Tools: The future

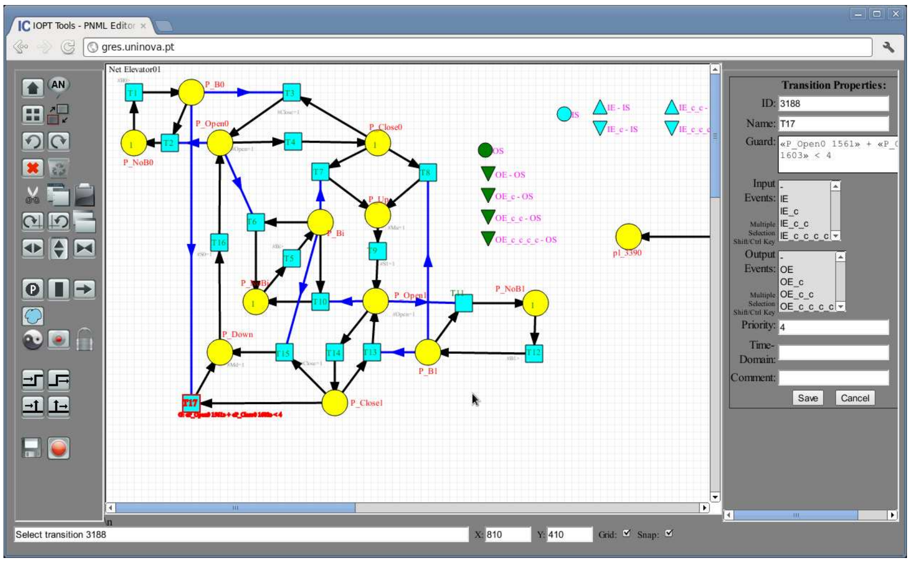
Porting all other tools
Full support for distributed execution of models (globally asynchronous locally synchronous systems)
… and more

IOPT-Tools – Web User Interface



Important note: IOPT-Tools require the latest Browser versions: Firefox >= 10, Chrome >= 12, Safari, Opera

Web based IOPT Model Editor



Transition Properties:

ID: 3188

Name: T17

Guard: «P_Open0 1561» + «P_(1603» < 4

Input Events: - IE IE_c IE_c_c IE_c_c_c_c

Multiple Selection Shift/Ctrl Key

Output Events: - OE OE_c OE_c_c OE_c_c_c_c

Multiple Selection Shift/Ctrl Key

Priority: 4

Time-Domain:

Comment:

Select transition 3188

X: 810 Y: 410 Grid: ☑ Snap: ☑

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA UNINOVA

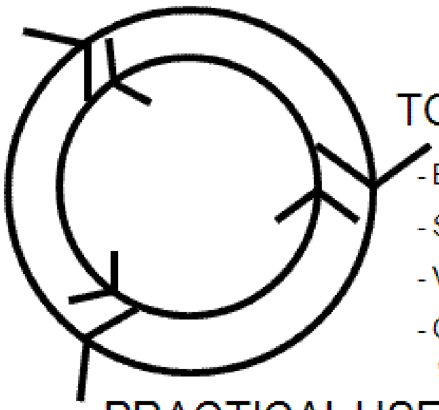
Outline

- Motivation
  - How to handle design complexity
  - Some issues and challenges
- Petri-nets for controller modeling
- Distributed Embedded Controllers Development Flow
  - Operations on nets
  - Distributed execution
  - Tools
- Sum-up

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA UNINOVA

THEORY

- Models
- Analysis methods



TOOLS

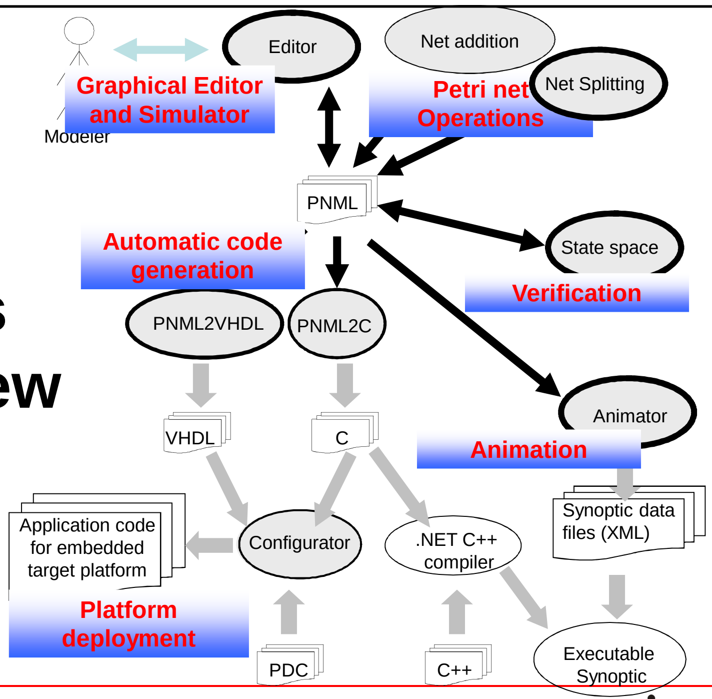
- Editing
- Simulation
- Verification
- Code generation

PRACTICAL USE

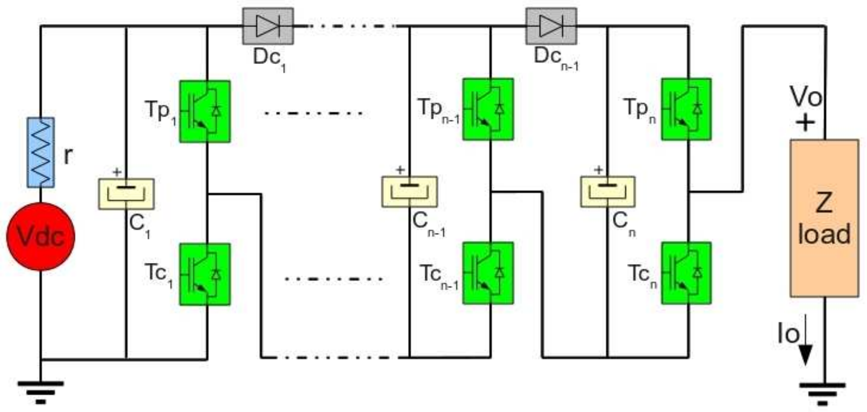
- Specification
- Verification
- Validation
- Implementation

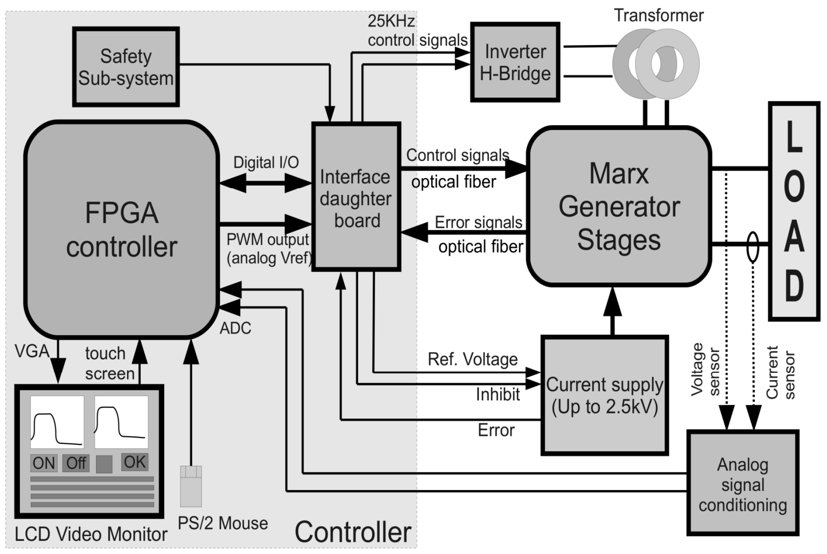
Adapted from
[Kurt Jensen, 80]

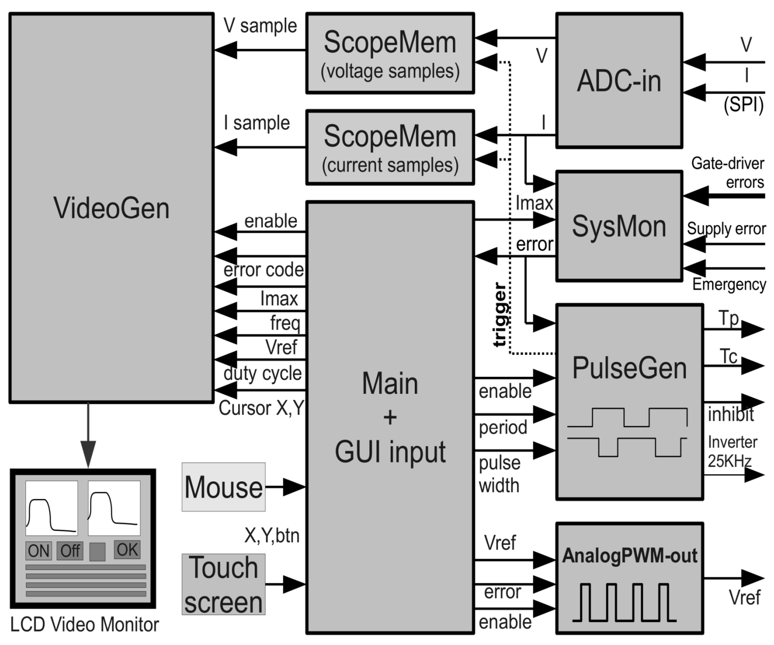
Tools overview



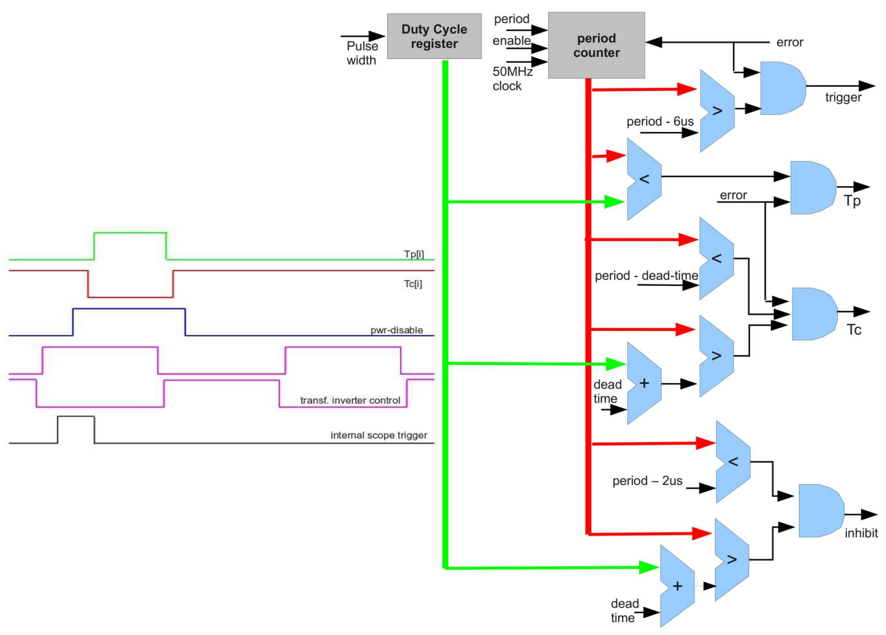
THE MARX PULSE GENERATOR TOPOLOGY







VideoGen
V sample
I sample
ScopeMem
(voltage samples)
ScopeMem
(current samples)
ADC-in
V
I
(SPI)
V
I
Gate-driver
errors
Supply error
Emergency
SysMon
Imax
error
trigger
enable
error code
Imax
freq
Vref
duty cycle
Cursor X,Y
Main
+
GUI input
PulseGen
Tp
Tc
inhibit
Inverter
25KHz
enable
period
pulse
width
Mouse
X,Y,btn
Touch
screen
ON
Off
OK
LCD Video Monitor
Vref
error
enable
AnalogPWM-out
Vref



Pulse
width
Duty Cycle
register
period
enable
50MHz
clock
period
counter
error
trigger
period - 6us
error
Tp
period - dead-time
Tc
dead
time
period – 2us
inhibit
dead
time
Tp[i]
Tc[i]
pwr-disable
transf. inverter control
internal scope trigger

Open issues and challenges

- How to reduce the productivity gap?
- How to reduce the verification gap?
- How to support reliable distributed execution?
- Contribution to the answers:
  - Relying more and more on Model-based Development
  - Increasing usage of design automation tools (including specification, simulation/validation, verification, code generation, and test)

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA UNINOVA

Thank you

Petri net models
within distributed embedded controller design
Luis Gomes
lugo@fct.unl.pt
Univ. Nova de Lisboa &
UNINOVA, Portugal

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA UNINOVA