

# Petri nets modeling and distributed embedded controller design

**Luis Gomes**

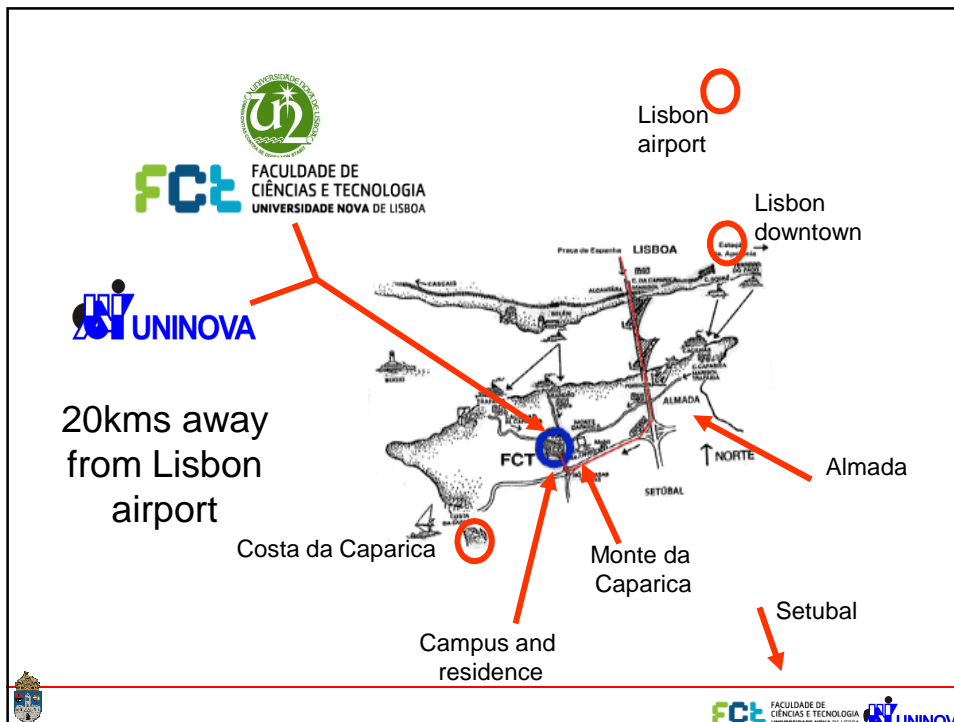
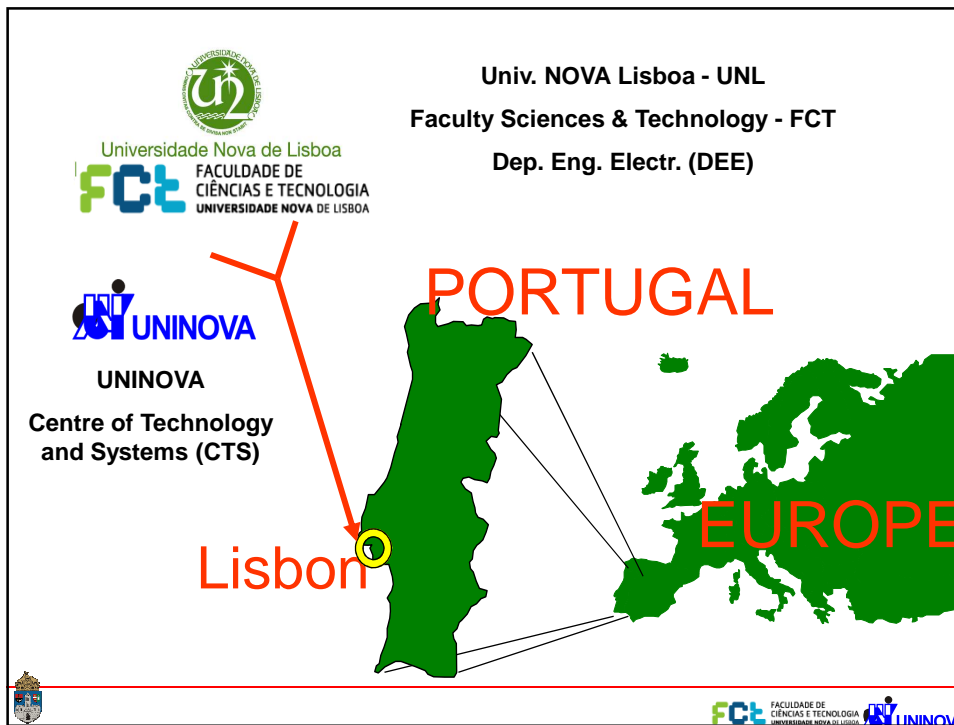
Univ. Nova de Lisboa – Fac. Sciences & Technology &  
UNINOVA - CTS, Portugal



[lugo@ieee.org](mailto:lugo@ieee.org)

**September 2, 2014**  
**Óbuda University**  
**Budapest, Hungary**







## Universidade Nova de Lisboa

- The Universidade Nova de Lisboa or NOVA is a Portuguese public university established in 1973, in Lisbon. In Portuguese, the name of the University means "New University of Lisbon" and it reflects the fact of being the youngest of the public universities of Lisbon.
- The University is commonly referred to, and is now officially branded, as NOVA (Portuguese for "new").



## Universidade Nova de Lisboa

- Schools:
  - Faculty of Science and Technology
  - Faculty of Social and Human Sciences
  - Nova School of Business and Economics
  - Faculty of Medical Sciences
  - Faculty of Law
  - Statistics and information management
  - Chemical and biological technology
  - Hygiene and tropical medicine
  - Public health



## Univ. Nova Lisboa in rankings

- QS World University Rankings 2013
  - Portuguese Universities
    - 343= University of Porto
    - 353= Universidade Nova de Lisboa
      - Top 50 under 50 @ 2013 (46)
    - 358= University of Coimbra
    - 551-600 Univ. Catolica Portuguesa, Lisboa
    - 551-600 University of Lisbon



## Univ. Nova Lisboa in rankings

- QS World University Rankings by Faculty 2013 - Engineering and Technology – Portuguese Universities

- 152= University of Porto
- 272= Universidade Nova de Lisboa
- 295= University of Coimbra
- 309= Universidade Técnica de Lisboa



### facts & numbers

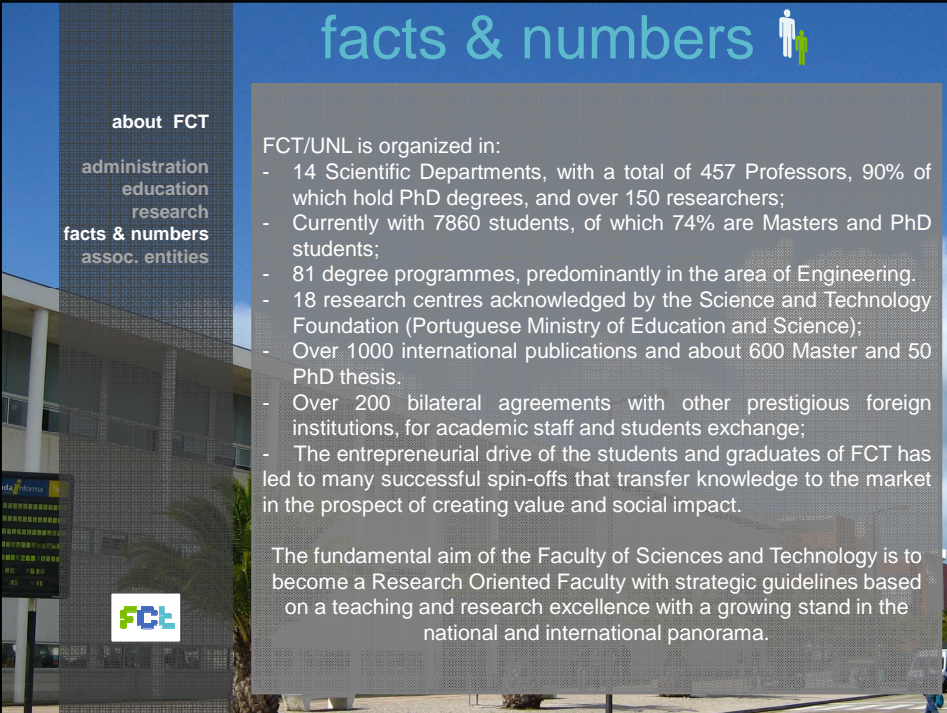
about FCT

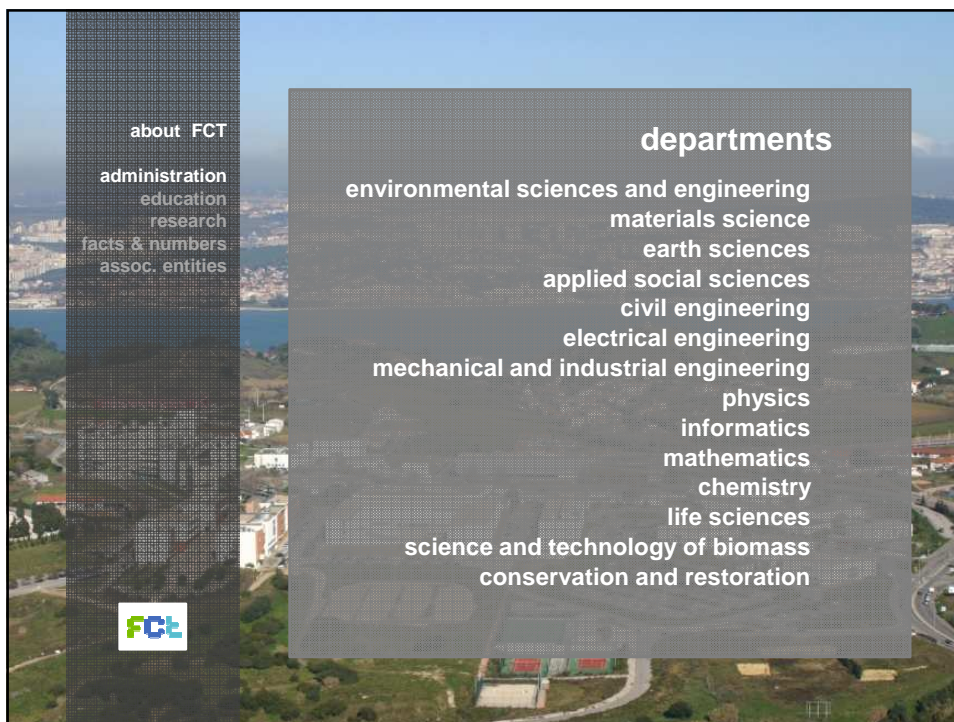
- administration
- education
- research
- facts & numbers**
- assoc. entities

FCT/UNL is organized in:

- 14 Scientific Departments, with a total of 457 Professors, 90% of which hold PhD degrees, and over 150 researchers;
- Currently with 7860 students, of which 74% are Masters and PhD students;
- 81 degree programmes, predominantly in the area of Engineering.
- 18 research centres acknowledged by the Science and Technology Foundation (Portuguese Ministry of Education and Science);
- Over 1000 international publications and about 600 Master and 50 PhD thesis.
- Over 200 bilateral agreements with other prestigious foreign institutions, for academic staff and students exchange;
- The entrepreneurial drive of the students and graduates of FCT has led to many successful spin-offs that transfer knowledge to the market in the prospect of creating value and social impact.

The fundamental aim of the Faculty of Sciences and Technology is to become a Research Oriented Faculty with strategic guidelines based on a teaching and research excellence with a growing stand in the national and international panorama.





# Petri nets modeling and distributed embedded controller design

**Luis Gomes**

Univ. Nova de Lisboa – Fac. Sciences & Technology &  
UNINOVA - CTS, Portugal



[lugo@ieee.org](mailto:lugo@ieee.org)

**September 2, 2014**  
**Óbuda University**  
**Budapest, Hungary**



Acknowledgments to contributions from  
João Paulo Barros, Anikó Costa, Filipe Moutinho, and Fernando Pereira,  
members of the  
Group on Reconfigurable and Embedded System - GRES  
from Univ. Nova de Lisboa / UNINOVA – CTS (Centre of Technology and Systems),  
Portugal.  
<http://gres.uninova.pt/>



## 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



- Exercise 1:
  - Think about the car that we were able to drive twenty years ago.
  - Compare with the car that we are able to drive today.





- **Exercise 1:**
  - Think about the car that we were able to drive twenty years ago.
  - Compare with the car that we are able to drive today.
- **Exercise 2:**
  - Think about the phone that we were able to use twenty years ago.
  - Compare with the phone that we are able to use today (from land lines phones to cell phones).



These examples are from our daily life.

**What about distributed  
embedded controller design?**



## 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  $\sim 1 \frac{1}{2}$  years since 1959

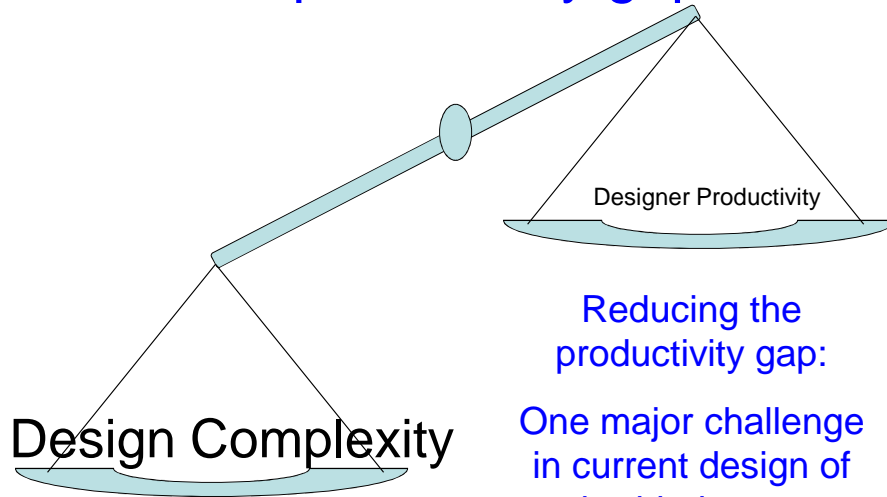


## Design complexity versus designer productivity

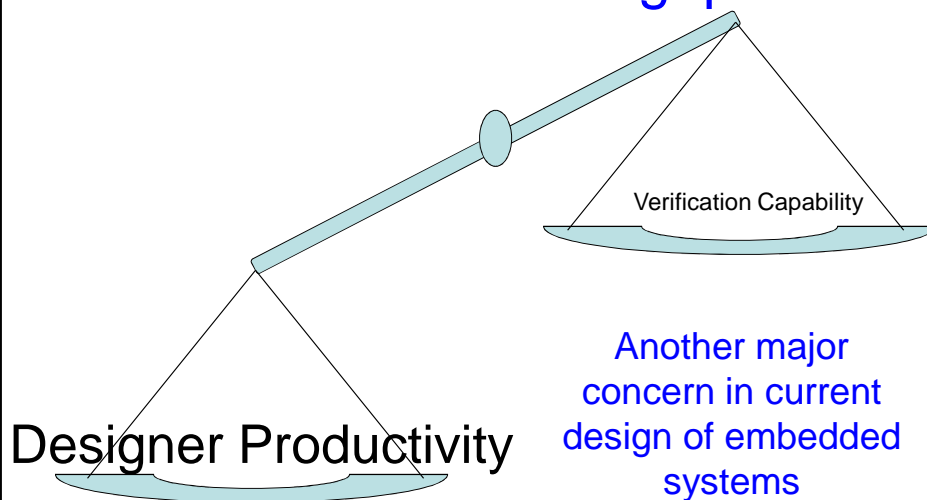
- Top-down versus bottom-up approaches
  - To handle complexity, it is normally assumed that analysis needs to be hierarchical and top-down.
  - However, reuse of modules is fundamental.
- Pragmatic approach (mostly followed):
  - Primarily follow top-down approach (system-level)
  - Complemented with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



## The productivity gap



## The verification gap



## Needs to improve the design...

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



## 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)

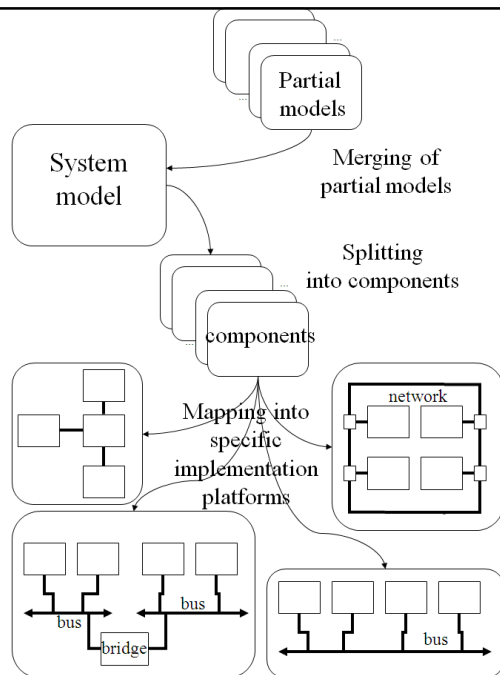


# Moving to model-based development

- 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



## 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 most common formalisms, it is worth to mention state diagrams, hierarchical and concurrent state diagrams, statecharts, and Petri nets.
- The selected formalism for this presentation is Petri nets. Why?
  - Rigorous computational model
  - Precise execution semantics
  - Graphical representation
  - Formal representation



# Petri nets

- Can be seen as a generalization of state diagrams
- **Bipartite graph**
  - Places (associated with the static part of the model)
  - Transitions (... dynamic part of the model)
  - Arcs (from places to transit. or from trans. to places)
- **Locality** on evaluation of model evolution.
- **Concurrency** on model evolution.



# Outline

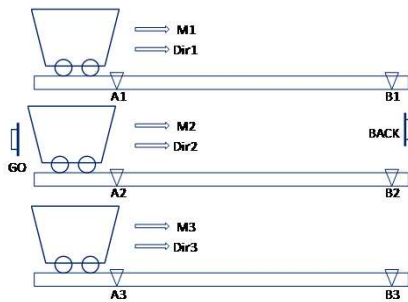
- Motivation
  - How to handle design complexity
  - Some issues and challenges
- Petri-nets for controller modeling
- Distributed Embedded Controllers Development Flow
  - Operations on nets
  - Distributed execution
  - Tools
- Sum-up







## 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.

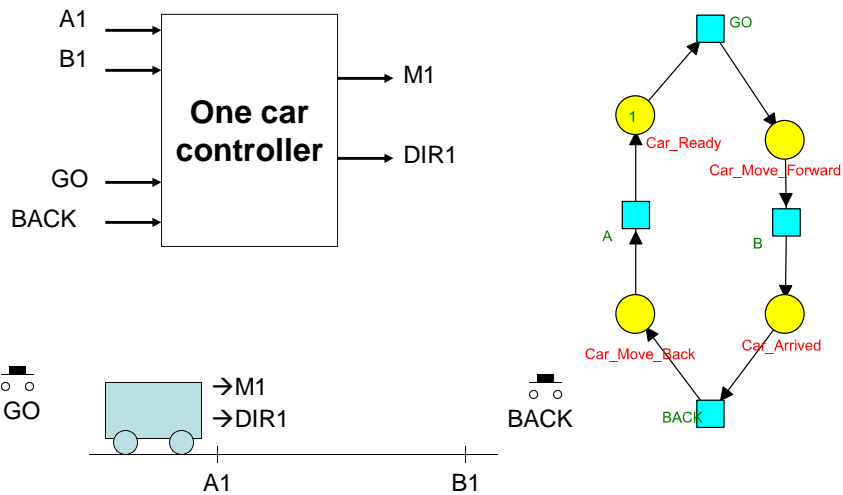


## Application example

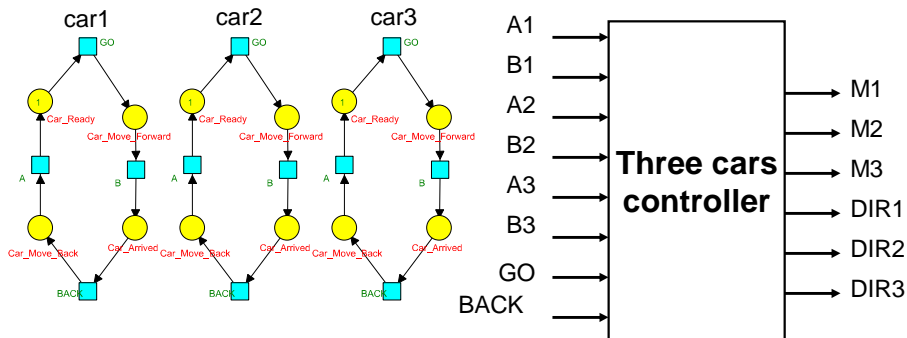
- Goal 1: to obtain a controller for one car
- Goal 2: to obtain a controller for the whole system composed by three cars
- Goal 3: to obtain a distributed controller composed by 3 controllers, one per car



## Goal 1: to obtain a controller for one car



## 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

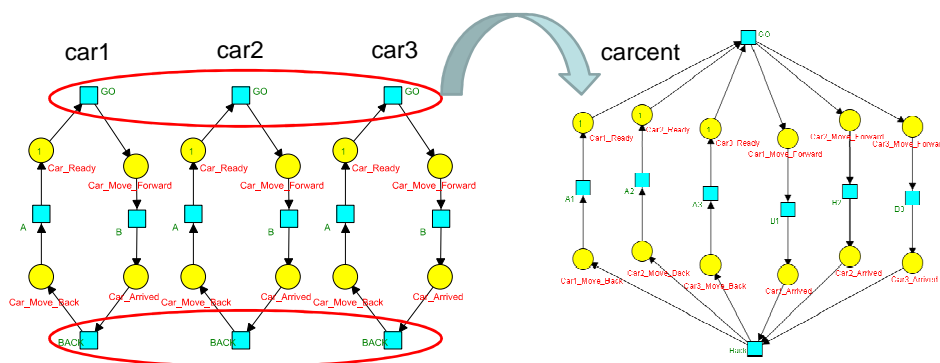


## Composability of net models

- 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



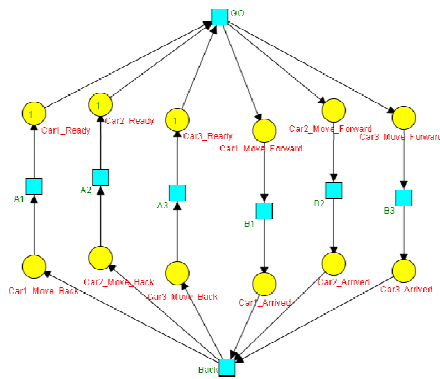
## The net addition operation



$\text{carcent} = \text{car1} + \text{car2} + \text{car3}$   
 (car1.go/car2.go/car3.go  $\rightarrow$  go,  
 car1.back/car2.back/car3.back  $\rightarrow$  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

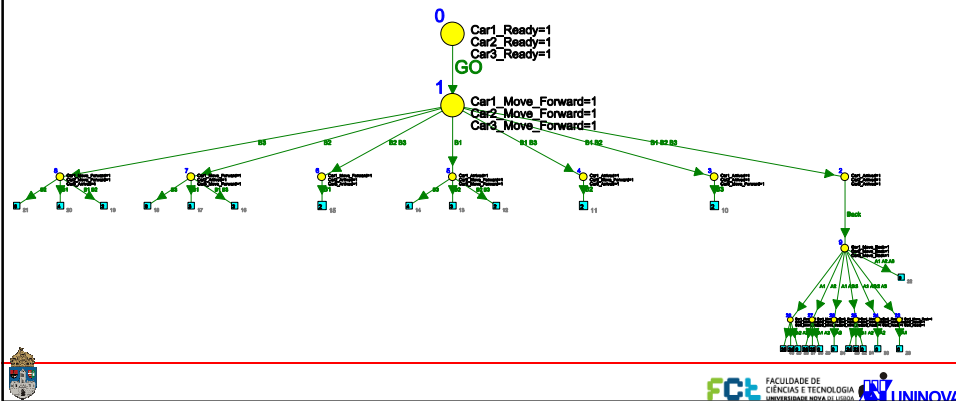
16 states detected; no conflict; no deadlocks  
 All places having as minimum marking 0 tokens  
 and as maximum marking 1 token

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]



## 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



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

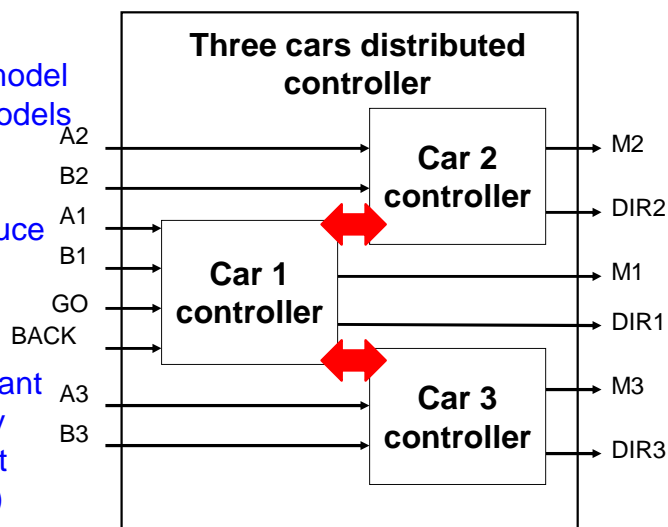
### Approach:

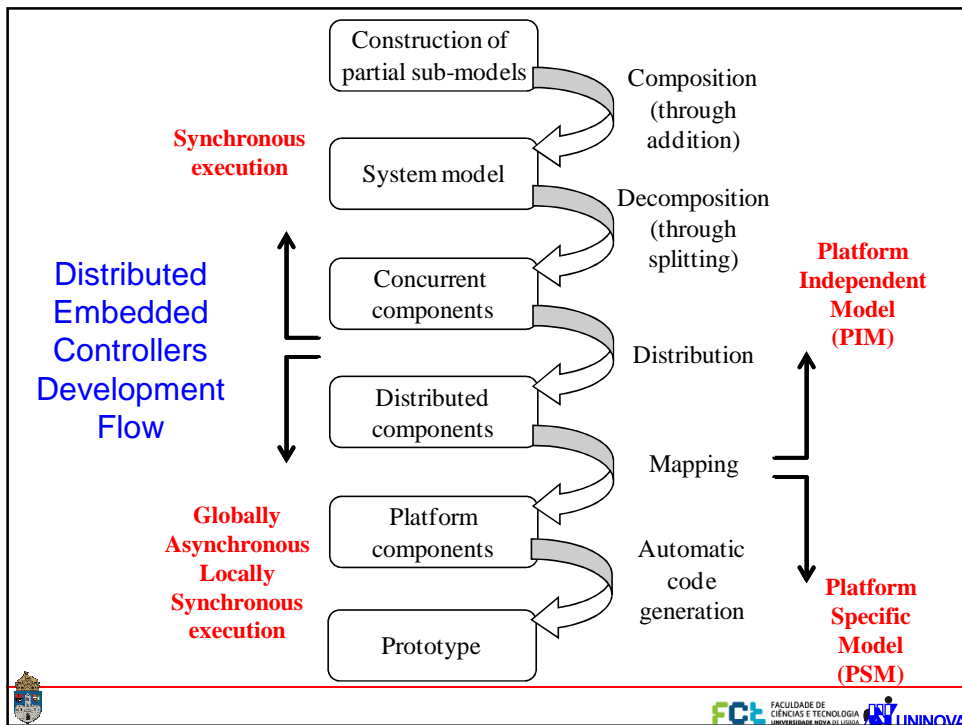
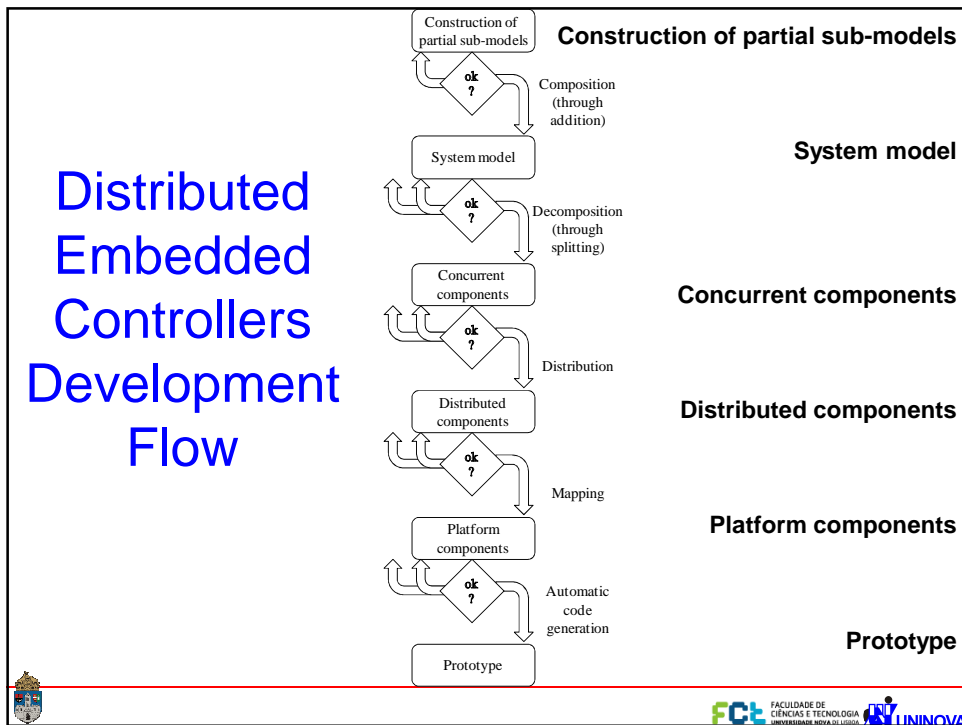
Decompose the model into concurrent models

### Problem:

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

**Constraint:** we want to assure property preservation (or at least predicability)





## 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)



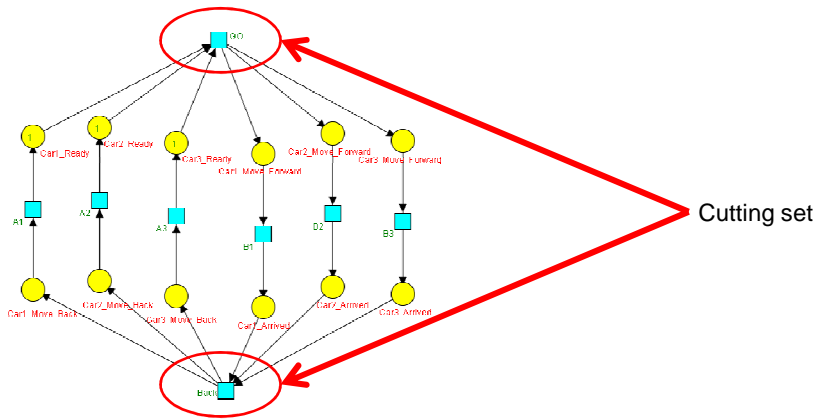
## The net splitting operation

- 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

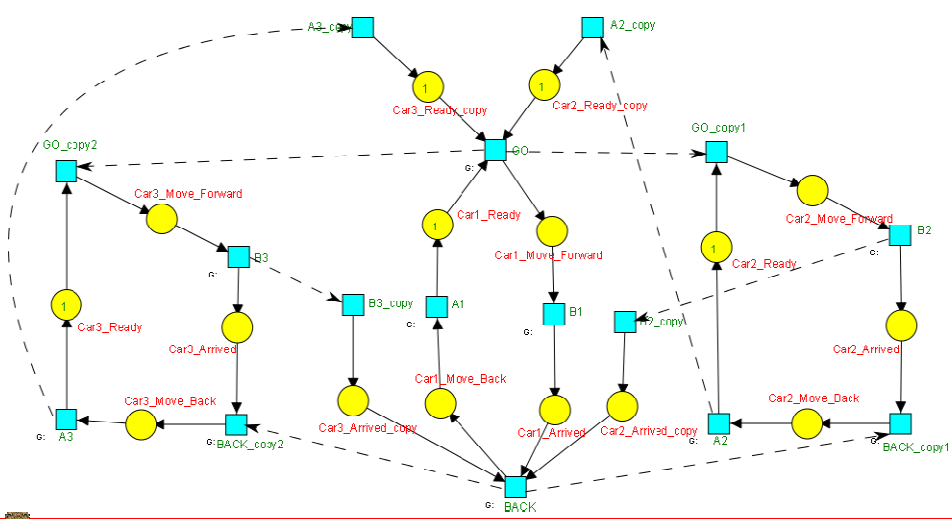




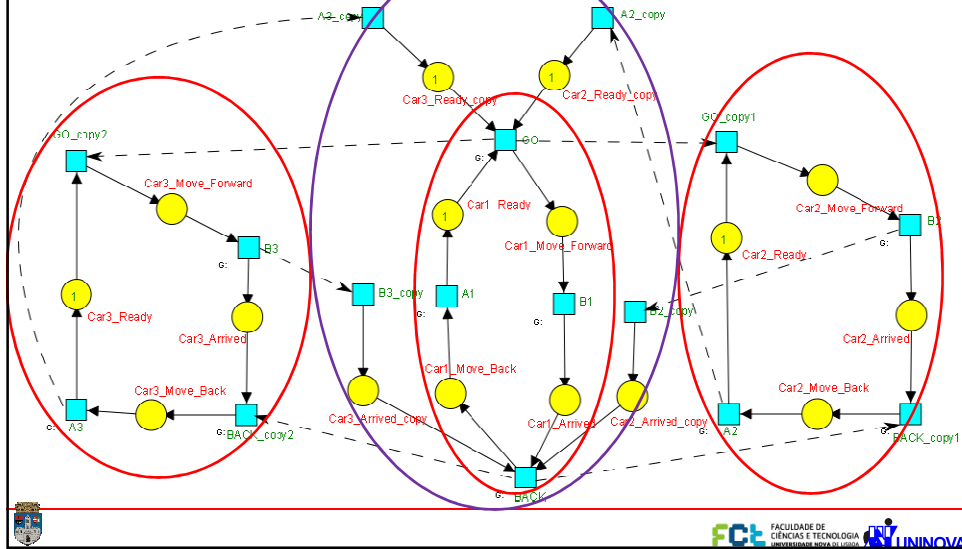
# Splitting our model



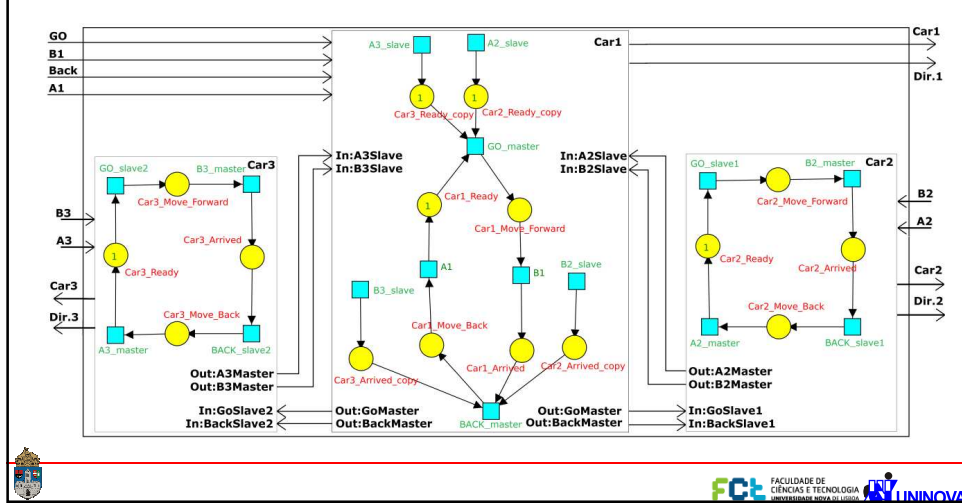
# Concurrent sub-models



# Concurrent sub-models (synchronous execution)



# 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.

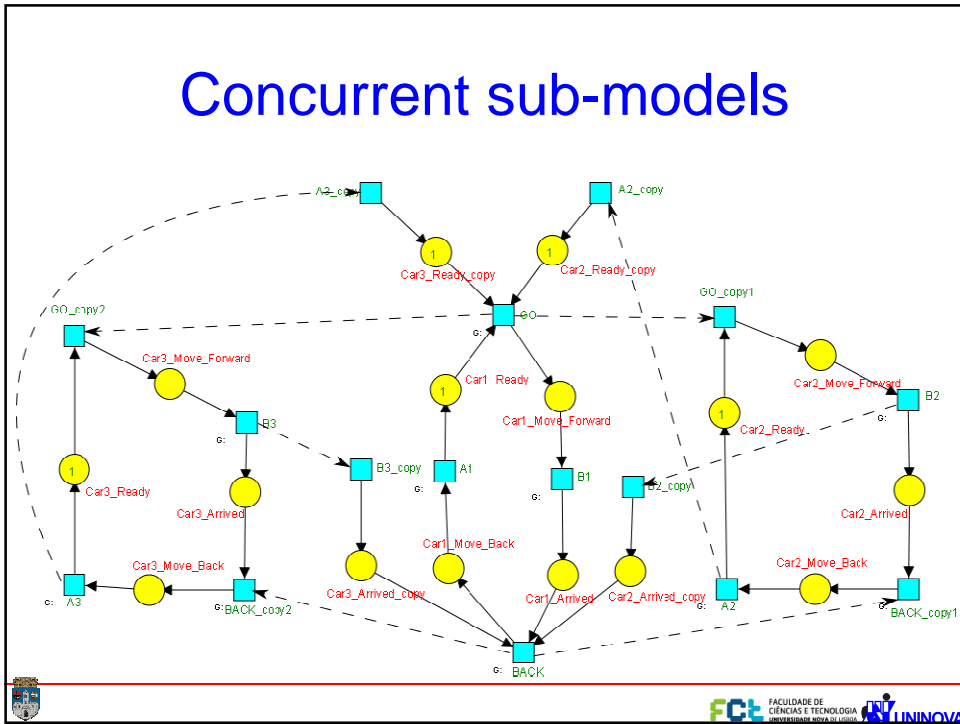


## Facing distributed implementations - II

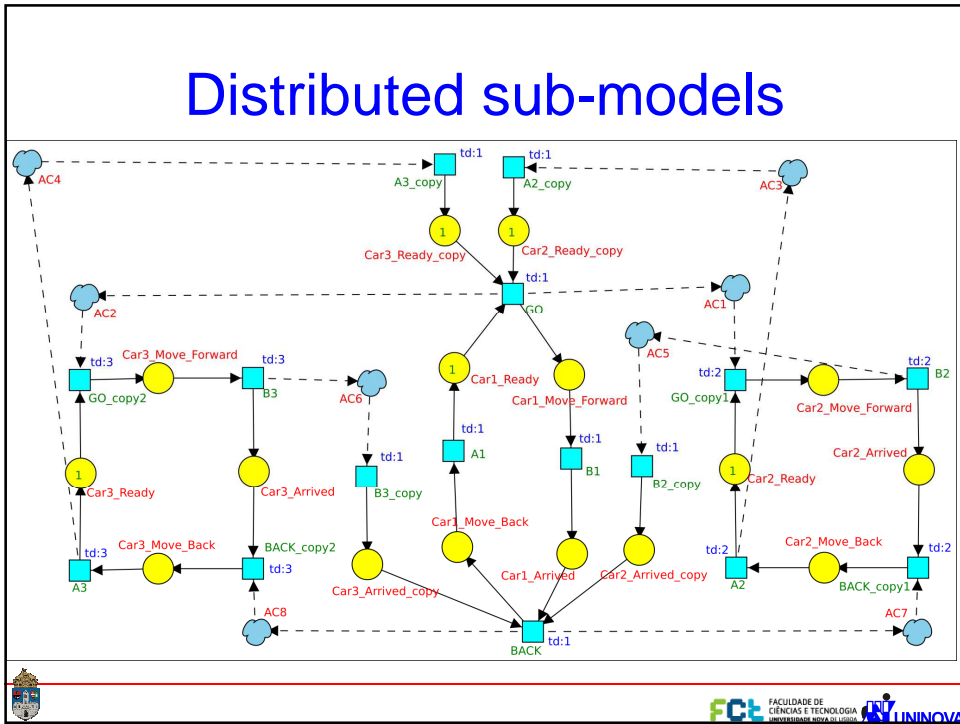
- Approach:
  - Definition of time domains (each time domain has its own tick / execution step)
  - Each component will be associated with one different time domain
  - Communication channels have a place semantics (holding non-instantaneous pending communication)
  - Each directed synchronous channel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



# Concurrent sub-models

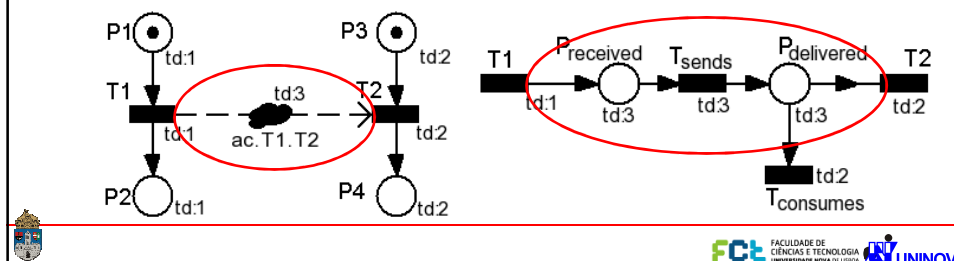


# Distributed sub-models



## 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



## 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).

## New tools are coming

- Petri nets already have a set of supporting tools mostly covering specification and verification.
- However, 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/>



## IOPT-Tools cloud-based framework

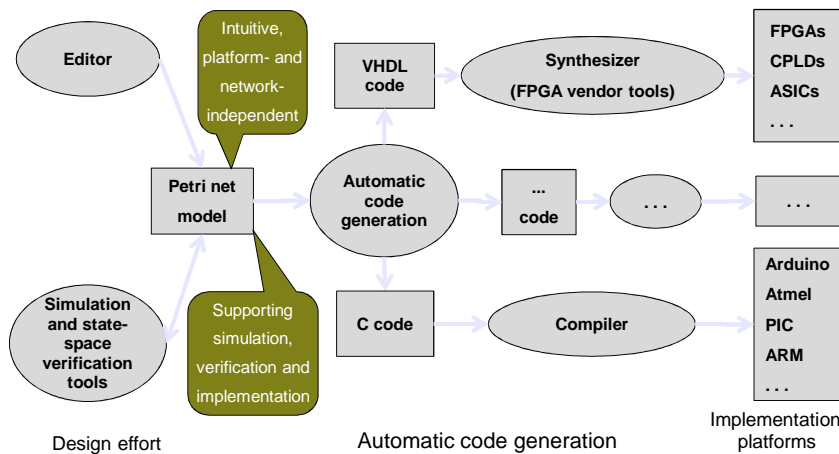
Tools offered under a cloud-based user interface

- Web User Interface (<http://gres.uninova.pt>)
- AJAX Based IOPT Petri Net Editor
- Simulation / Debugger
- State Space Generation Tool
- Model-checking using a Query System
- Automatic controller C code generator
- Automatic controller VHDL hardware synthesis



# Development flow:

Model-based design of embedded controllers using IOPT Petri-nets



# IOPT-Tools

IOPT Tools

User Login:

Username:

Password:

NOTES: Example models available under username "**models**" and password "**models**".

Anonymous users can login into the system with username "**guest**" and password "**guest**", to create new IOPT models and test all system functionalities.

However, these models will be openly accessible to all users and may be modified by other users or deleted at any time. Therefore, creating a personal user account [free] is highly recommended.

[IOPT-Tools](#) have been developed by several members of the R&D Group on Reconfigurable and Embedded Systems ( [GRES](#) )

At current development phase, some changes and improvements will occur in the near future. Comments or requests can be directed to [gres@uninova.pt](mailto:gres@uninova.pt)

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

Copyright (C) 2013 GRES Research Group  
[gres.uninova.pt](http://gres.uninova.pt)

# IOPT-Tools – Overview

The screenshot shows the IOPT Tools web interface. The browser address bar displays `gres.uninova.pt/IOPT-Tools/index.php`. The page title is "IOPT Tools (fjp)" with the subtitle "Net file: uart\_recv.pnml". The main area contains a Petri net diagram with various places (circles) and transitions (squares). Places include `S: ReadRatePeriod`, `S: RxD`, `S: DataByte`, `S: RxDv`, `S: RxDError`, `S: ReadValue`, `S: ReadRateCtrl`, `S: Ctrl`, `E: Ctrl - Ctrl`, `P: WaitStart`, `T: StartBitEdge`, `P: WaitHalfStart`, `T: StartBitConf`, `T: StartBitFail`, `P: ResetHwCtrl`, `T: FirstBit`, `P: StoreLastValue`, `T: RxDvEnd`, `P: StoreNewValue`, `T: NextBit`, `P: LastBit`, `T: BandDelay`, `T: BandDelay`, and `P: ShiftBits`. Below the diagram are several buttons: "Edit Model", "Simulator", "Generate State Space", "Generate C Code", "Synthesize VHDL Code", "Query Editor", and "Query Results". At the bottom, there are "Model actions" including "Download Model File", "Export Snoopy/C", "Export Snoopy/VHDL", "Decompose GALs", "HIPPO", and "Model List". A "Start new model:" field and an "Upload model file:" button are also visible.

# IOPT-Tools – Model Editor

The screenshot shows the IOPT Tools Model Editor web interface. The browser address bar displays `gres.uninova.pt/opt-tools/pnml_editor.php`. The page title is "IOPT Tools - PNML Editor". The main area contains the same Petri net diagram as the overview. On the left, there is a toolbar with various icons for editing the model. On the right, there is a "Place Properties:" panel with fields for "ID", "Name", "Initial", "Marking", "Bound", "Time", "Domain", "Comment", "Output", "Action 1", and "When". The "Name" field is filled with "PWaitStart". At the bottom, there are "Select place 2", "x: 750", "y: 310", "Grid", and "Snap" options.



**Arc Properties:**

ID: 3025

Type: Normal

Inscription: 1

Save Cancel

**Signal Properties:**

Name/ID: GotTicket

Mode: input

Type: Boolean

Value: 0

Min: 0

Max: 1

Save Cancel Change ID

Used by input-event Identified;

**Event Properties:**

ID: Identified

Mode: input

Autonomous:

Edge: Up

Level: 0

Signal: GotTicket

Save Cancel Change ID

Used by transitions tr-3007.got\_ticket;

**Transition Properties**

ID: 3028

Name: leave\_empty

Priority: 1

Guard: leave = 1

Input Events: Arriveln

Output Events:

Action 1: =

Time-Domain:

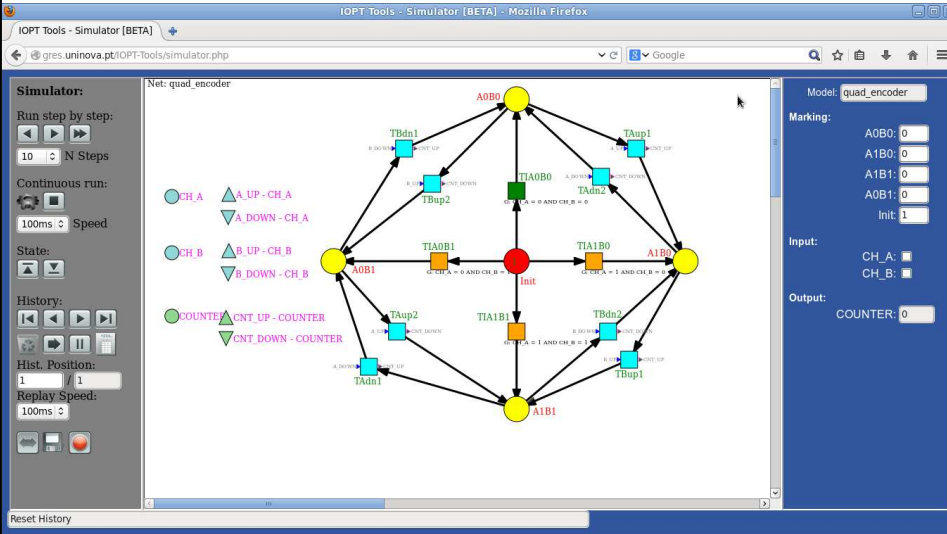
Comment:




Save Cancel





## IOPT-Tools – Simulator/Debugger



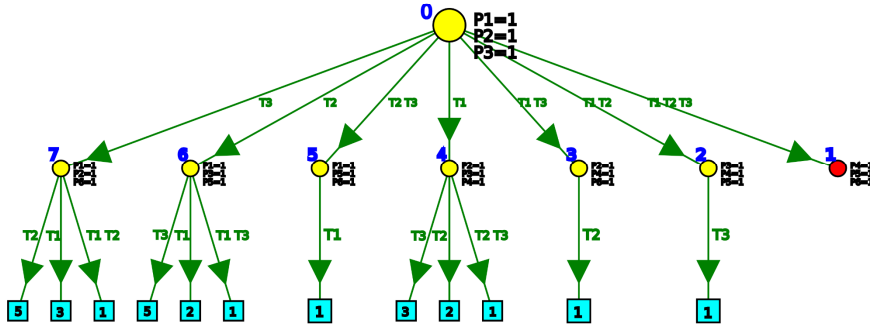
# IOPT-Tools – Spate-space generator

## Net Model2

8 Nodes, 12 Loops, 1 Deadlocks, 0 Conflicts, Max. Depth = 3

Min Bound = [P1=0 P2=0 P3=0 P4=0 P5=0 P6=0]

Max Bound = [P1=1 P2=1 P3=1 P4=1 P5=1 P6=1]



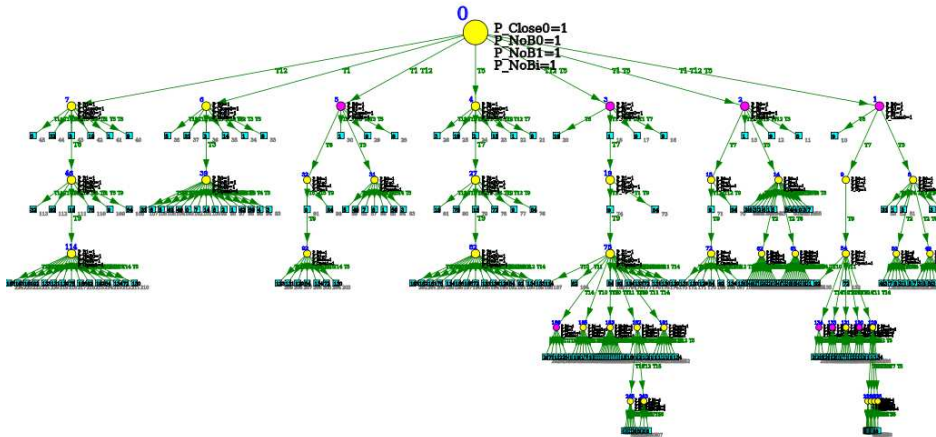
# IOPT-Tools – Spate-space generator

## Net Elevator01

44 (from 44) Nodes, 273 Loops, 0 Deadlocks, 8 Conflicts, Max. Depth = 7, 0 Invalid

Min Bound = [P\_B0=0 P\_B1=0 P\_B=0 P\_Close0=0 P\_Close1=0 P\_Down=0 P\_NoB0=0 P\_NoB1=0 P\_NoB=0 P\_Open0=0 P\_Open1=0 P\_Up=0]

Max Bound = [P\_B0=1 P\_B1=1 P\_B=1 P\_Close0=1 P\_Close1=1 P\_Down=1 P\_NoB0=1 P\_NoB1=1 P\_NoB=1 P\_Open0=1 P\_Open1=1 P\_Up=1]



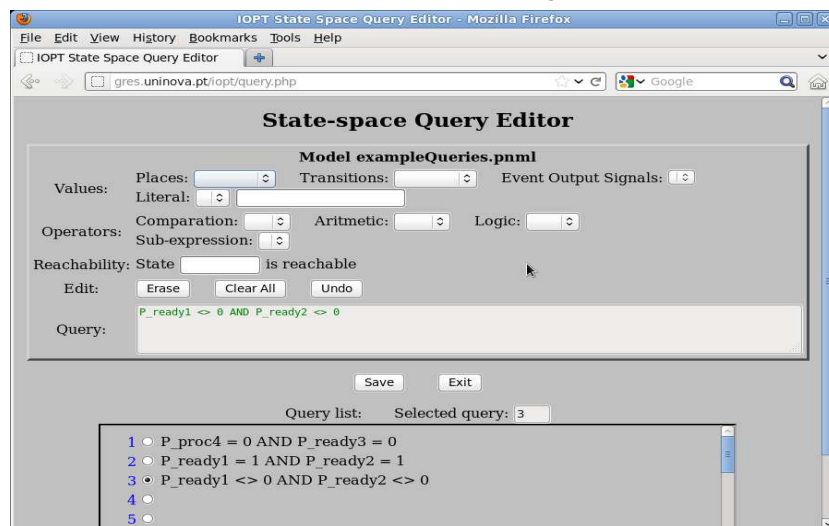
# Model Checking

**Very complex state-space graphs (millions of states) cannot be visually inspected:**

- **Query Editor**  
Graphical user interface to define queries and automatically check properties on the resulting state-space graph
- **Query results filters**  
Inspect and filter query results
- **Compilation strategy**  
Queries are compiled into C code and inserted into the state-space generator program



## IOPT-Tools – Query Editor



## IOPT-Tools – Query results

Found 12 matching query results for model exampleQueries

**Summary:**

<input checked="" type="checkbox"/> Query 1 ( P_proc4 = 0 AND P_ready3 = 0 )	6 matching states
<input type="checkbox"/> Query 2 ( P_ready1 = 1 AND P_ready2 = 1 )	3 matching states
<input type="checkbox"/> Query 3 ( P_ready1 <> 0 AND P_ready2 <> 0 )	3 matching states

Sort by Query    Sort by State     

State/Link ID	Query Nr.
29	1
36	1
37	1
38	1
39	1
40	1



## 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



## 50 Years after Prof. Petri's seminal work (1962)

- Professor Carl Adam Petri passed away on July 2, 2010, with the age of 83 years old; he was born on 12 July 1926 in Leipzig.
- As stated in his message to the Academy of Transdisciplinary Learning and Advanced Studies, in the occasion of accepting the "Academy Gold Medal of Honor" recognizing his lifetime achievements, in 2007, Professor Carl Adam Petri referred that "Yet, I do not consider my work as finished".
- Luckily for us, so many of us are very grateful to him for laying the foundation for such flourishing research area and are willing to continue developing and applying Petri nets in so many fields of science and engineering.

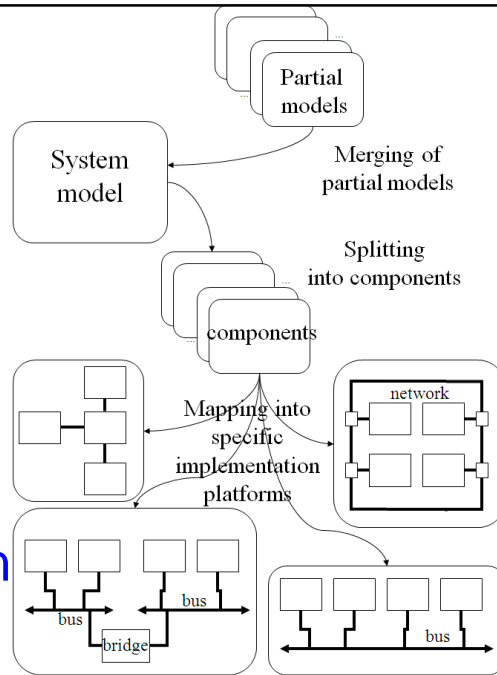


## 50 Years after Prof. Petri's seminal work (1962)

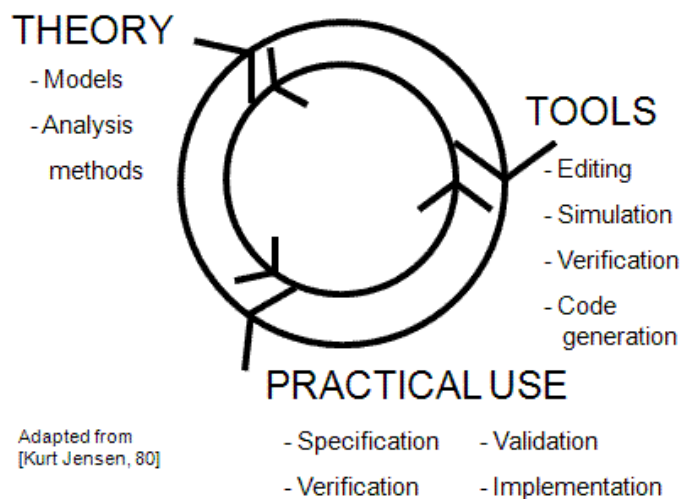
- Several parallel paths were traveled in different domains...
- ... nowadays Petri nets can be placed in the center of development processes fully supporting the development of complex systems though design automation tools.



Petri nets at the center of the distributed embedded controller development: Composition, decomposition, and composition again

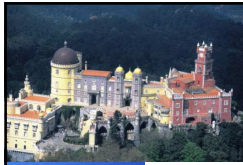


Underlying approach



Adapted from [Kurt Jensen, 80]

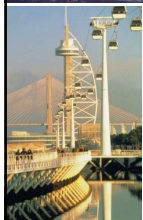




## Further information



- Feel free to contact me:
  - Luis Gomes,
    - [lugo@fct.unl.pt](mailto:lugo@fct.unl.pt)
    - <http://www.uninova.pt/~lugo>
- to use IOPT-Tools
  - <http://gres.uninova.pt>
- and to let us know your comments



FCT FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

## Petri nets modeling and distributed embedded controller design

**Luis Gomes**

Univ. Nova de Lisboa – Fac. Sciences & Technology & UNINOVA - CTS, Portugal



[lugo@ieee.org](mailto:lugo@ieee.org)

Se  
Ó  
University  
Budapest, Hungary

**Thank you**



FCT FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA UNINOVA