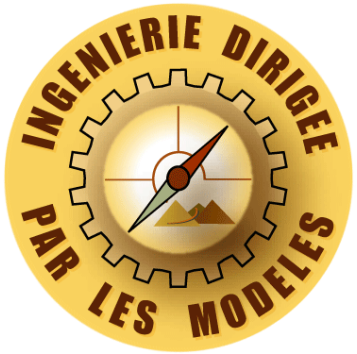
UMONS
Université de Mons

Faculté
des Sciences

Executable modeling and simulation of system software and processes

Tom Mens

tom.mens@umons.ac.be



Software Engineering Lab, Dept. Computer Science

complexys
UMONS RESEARCH INSTITUTE
FOR COMPLEX SYSTEMS

Software-controlled systems are *omnipresent*

Software-controlled systems are *difficult to develop*

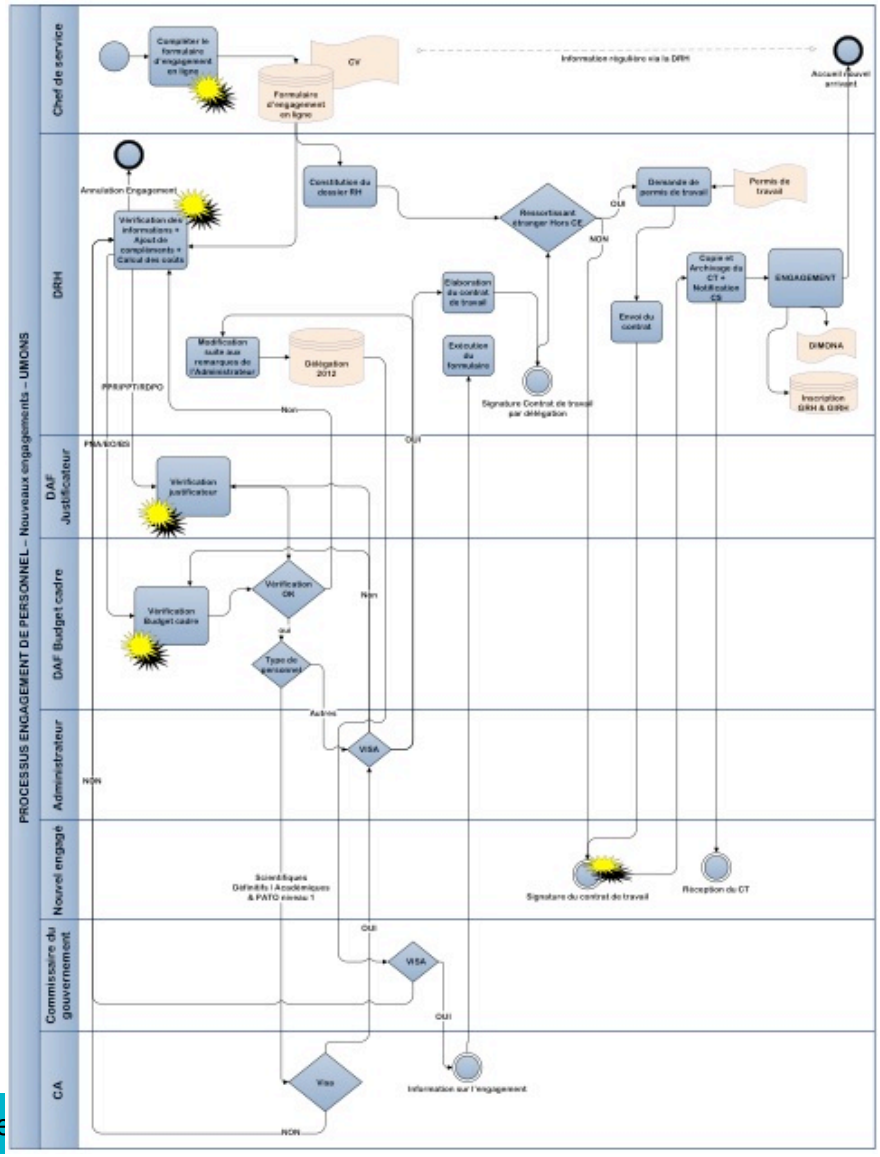
Control software can be very ***complex***

- Continuous interaction between software and hardware
- Continuous interaction with external world and users
- Must respect *functional* requirements
  - Vending machine does not dispense correct beverage
  - Cash machine returns more cash than requested
  - …
- Must respect *non-functional* requirements
  - Safety / Security / Performance / Energy constraints / Maintainability / Usability / …
    - Microwave/elevator should not function with open doors
    - Traffic lights should never be green simultaneously

The same is true for (business) processes

Example 1:

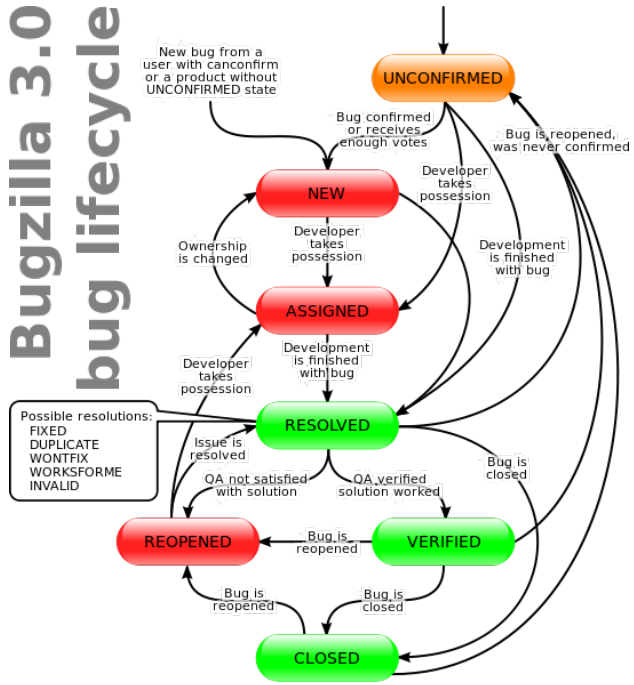
Modèle du processus d'engagement de personnel UMONS (version 1, 2013)

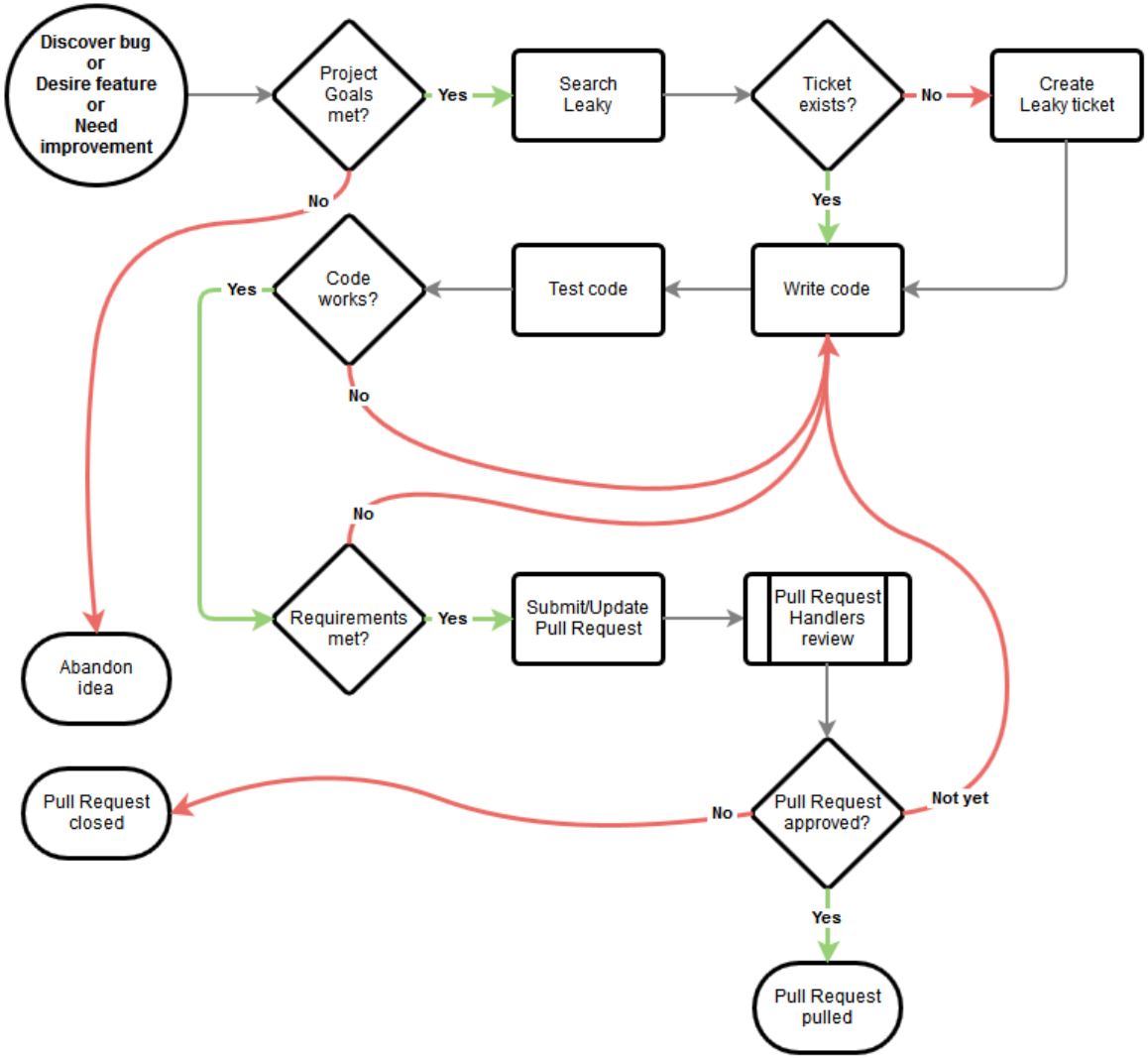


The same is true for (business) processes

Example 2:

Distributed software development and bug fixing processes





Model simulation and analysis can help to …

- better understand the problem
- reduce the "accidental complexity" of the solution
- detect errors early
  - Test or verify the solution before it is actually implemented
- explore the design space
  - compare alternative solutions through simulation

Modeling languages

- Allow to express the solution at a higher level of abstraction than traditional programming languages
- Provide support for
  - *Simulating* the desired behaviour or process
  - *Generating* software code automatically from the simulated model in order to
    - *Execute* and *integrate* the generated software with other software or hardware
    - *Support* or *control* the modeled process
  - *Verify*ing *correctness* of the modeled behaviour

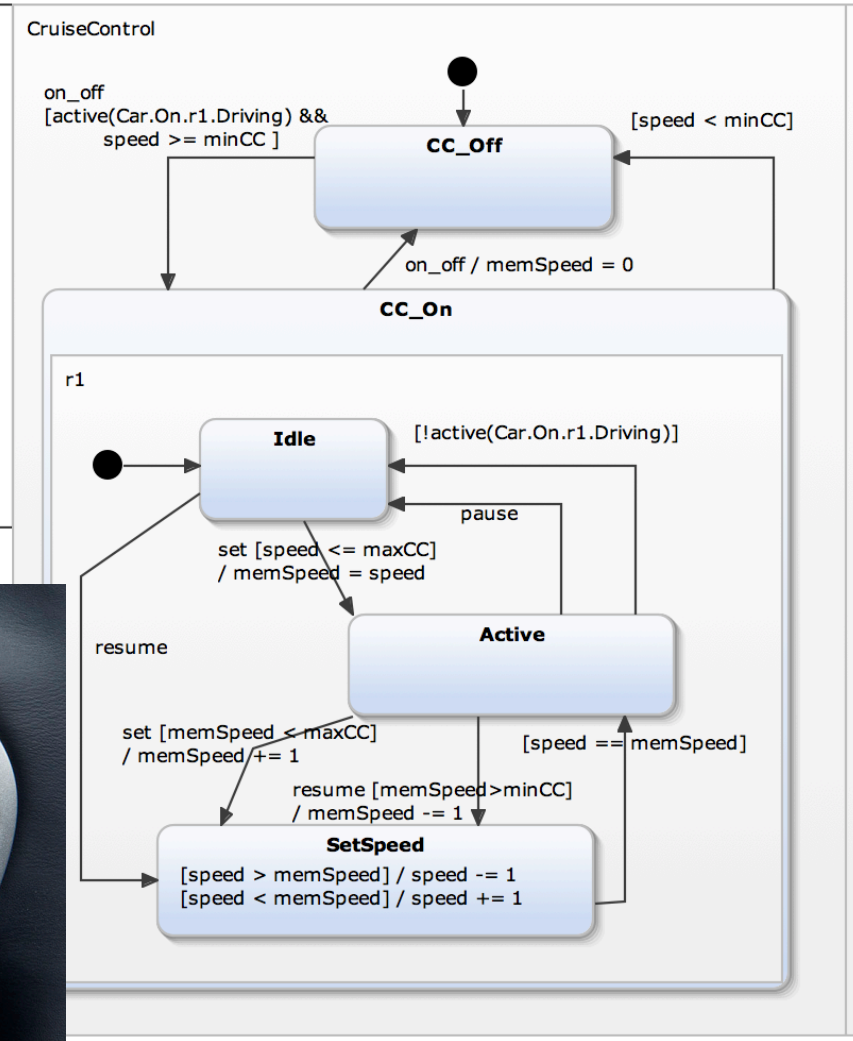
Modeling Languages
Example : Statecharts

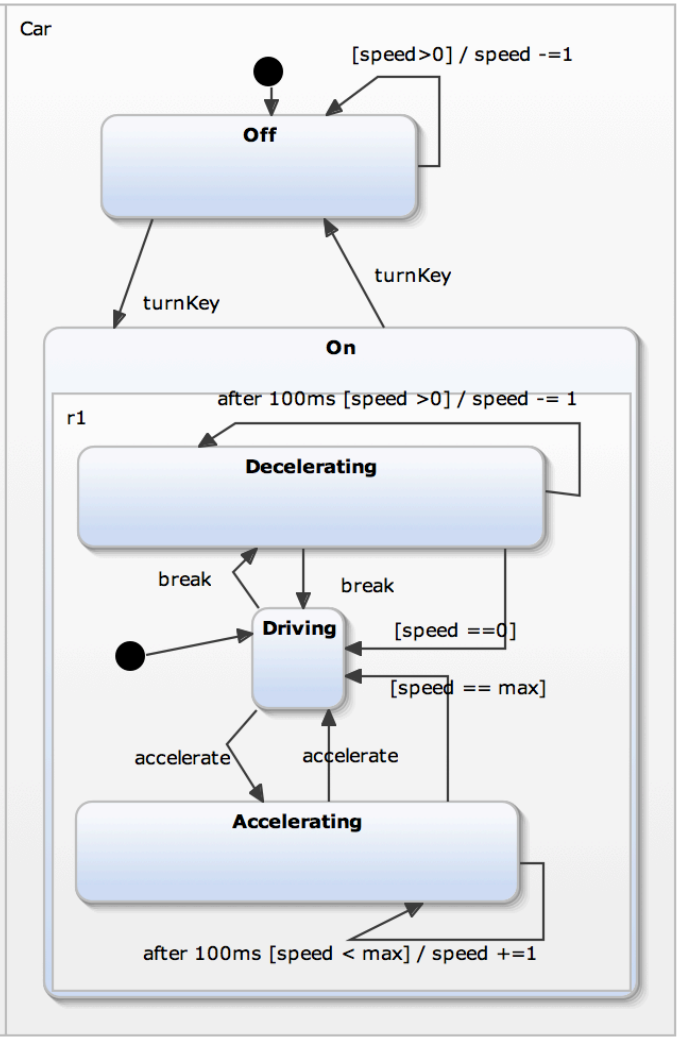
CruiseControl

interface:

var speed: integer = 0 // car speed
const max: integer = 180 // max car speed
// cruise control speed is either 0
// or a value between 30 and 120
var memSpeed: integer = 0
const minCC: integer = 30
const maxCC: integer = 120

//events for Car
in event turnKey
in event break
in event accelerate
//evens for CC
in event on_off
in event pause
in event set
in event resume

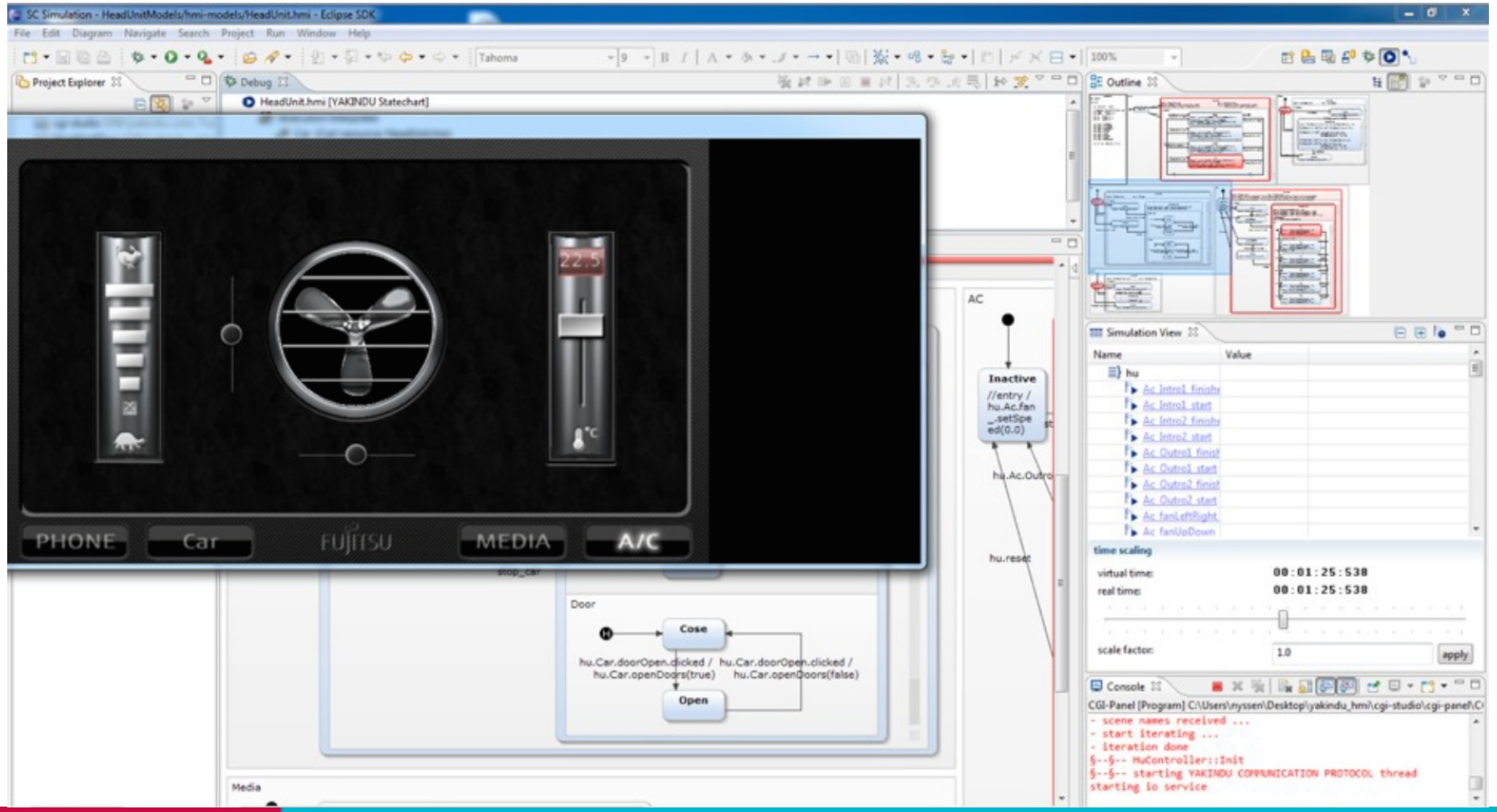




Modeling Tools
Example : Statecharts

Many tools for simulation and execution

Example: **Yakindu** (www.statecharts.org)

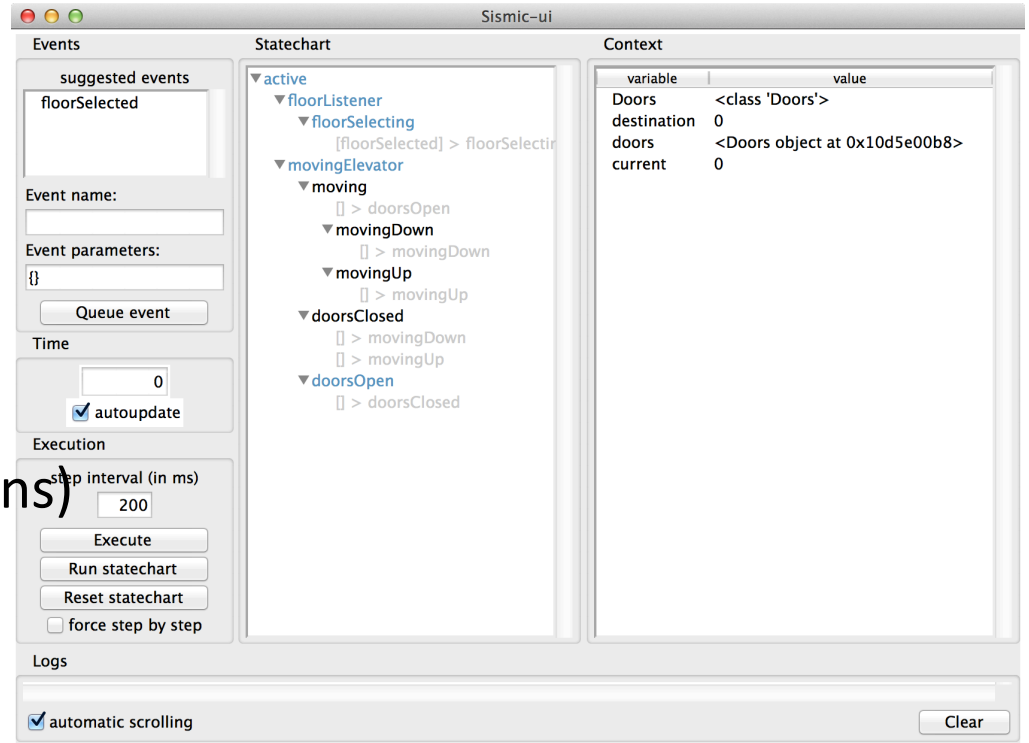


Modeling Languages
Example : Statecharts

SISMIC (created by A. Decan)

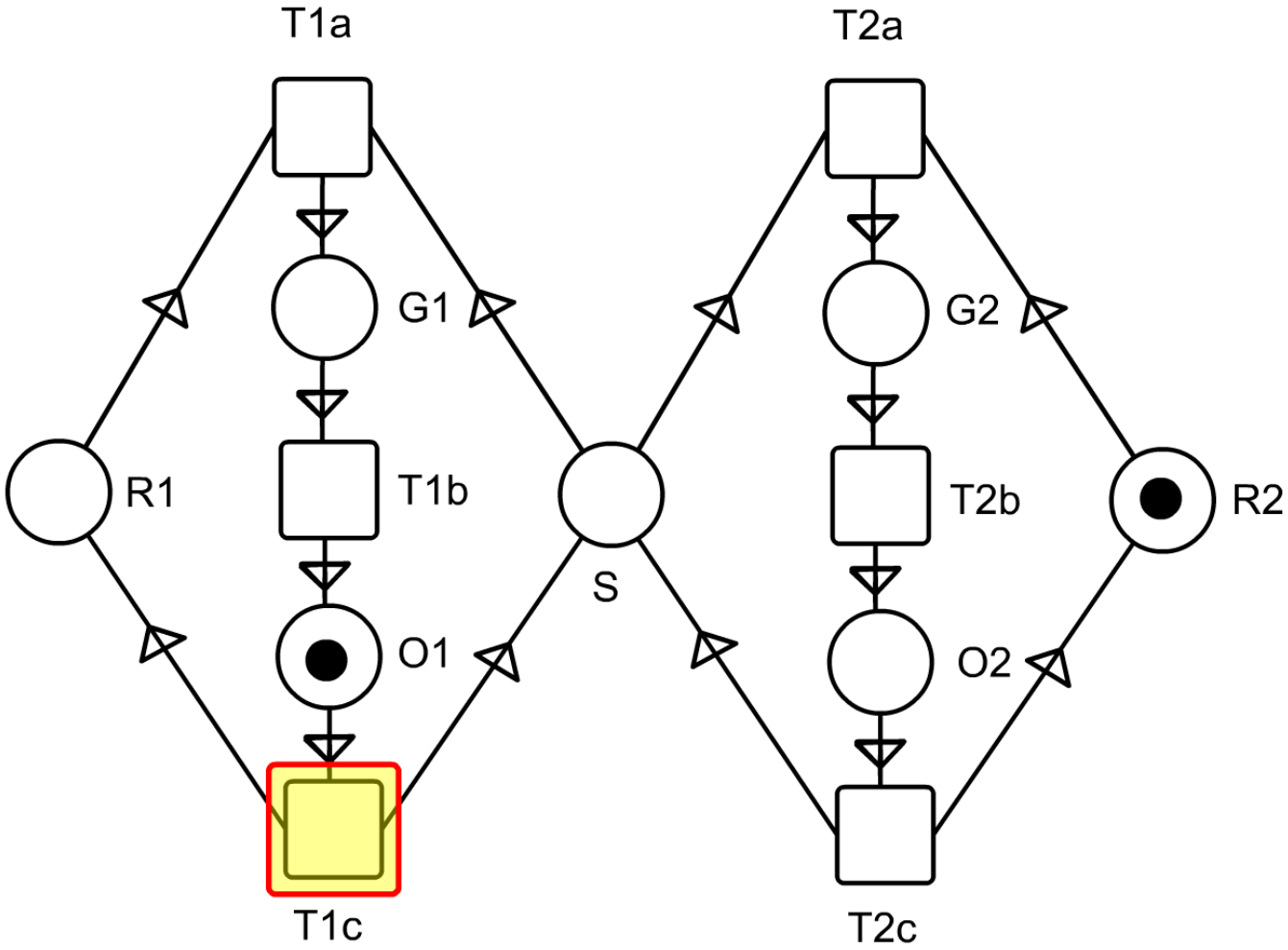
A tool under development at Software Engineering Lab (UMONS) to

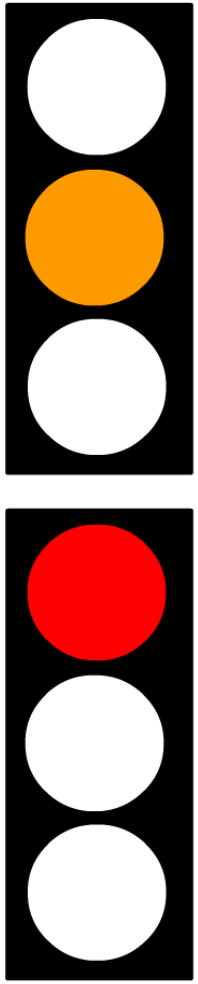
- Simulate statecharts
- Integrate with Python code
- Facilitate testing (based on "stories")
- Support multiple communicating statecharts
- Express statechart contracts (invariants, pre- and postconditions)
- And many more…



See https://github.com/AlexandreDecan/sismic

Modeling Languages
Example : Petri nets



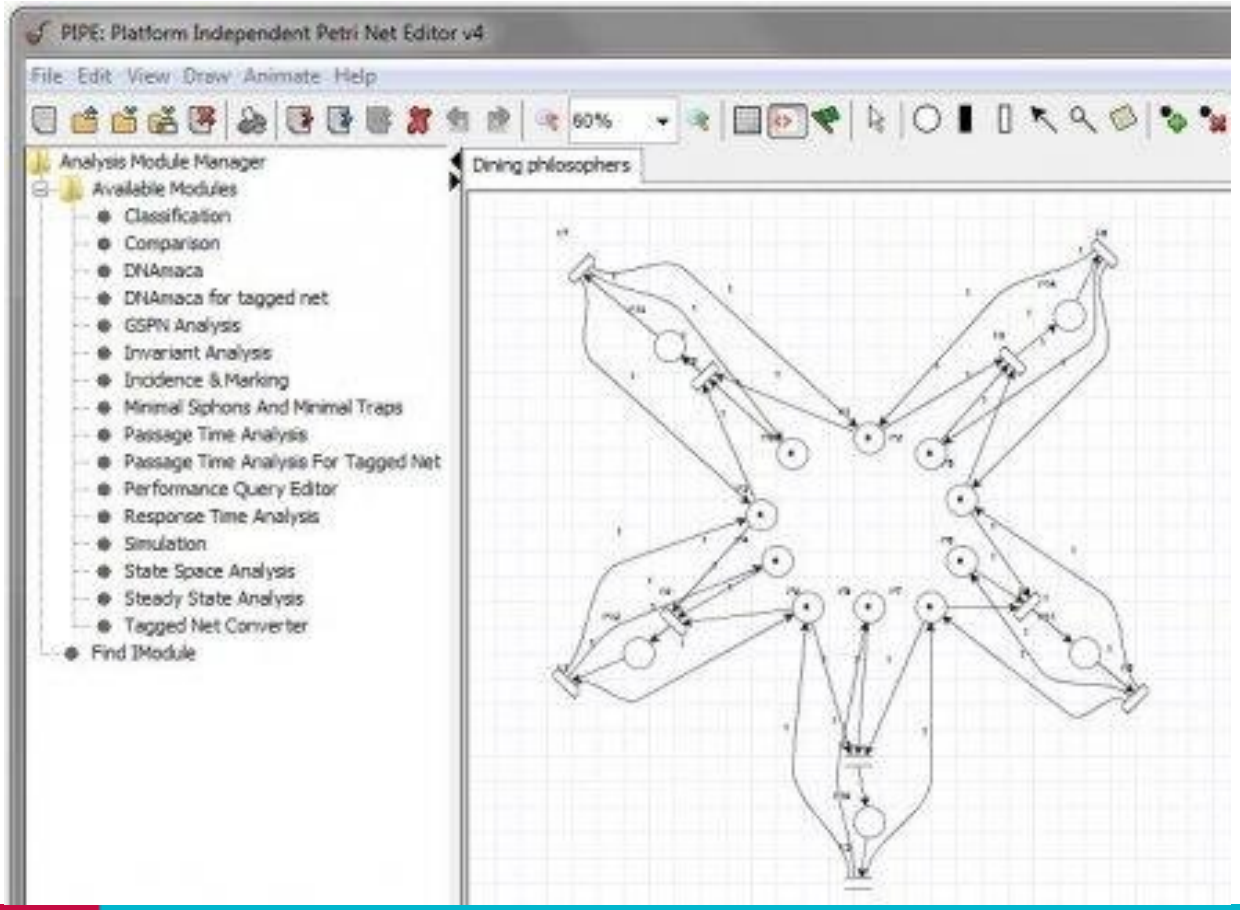


See “Petri Nets World” for more information
http://www.informatik.uni-hamburg.de/TGI/PetriNets

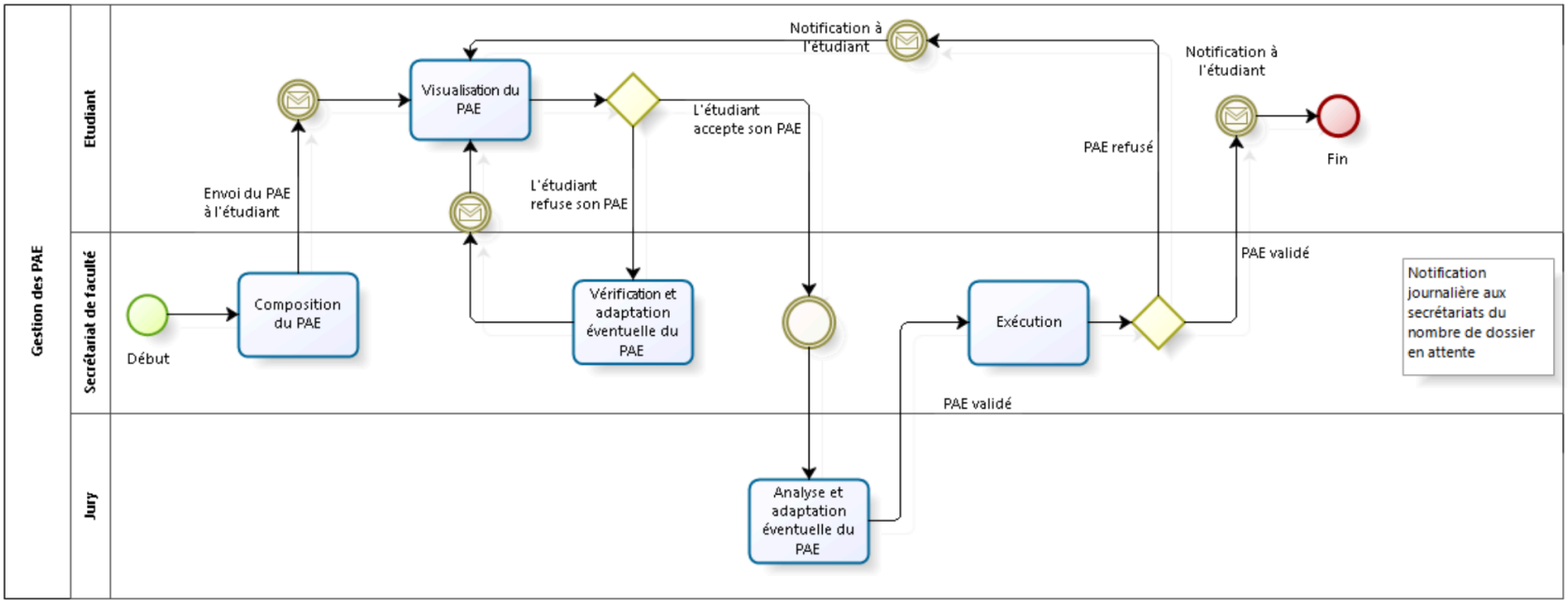
Modeling tools
Example: Petri nets

Many tools for simulation and formal analysis

Example: Pipe2 (pipe2.sourceforge.net)

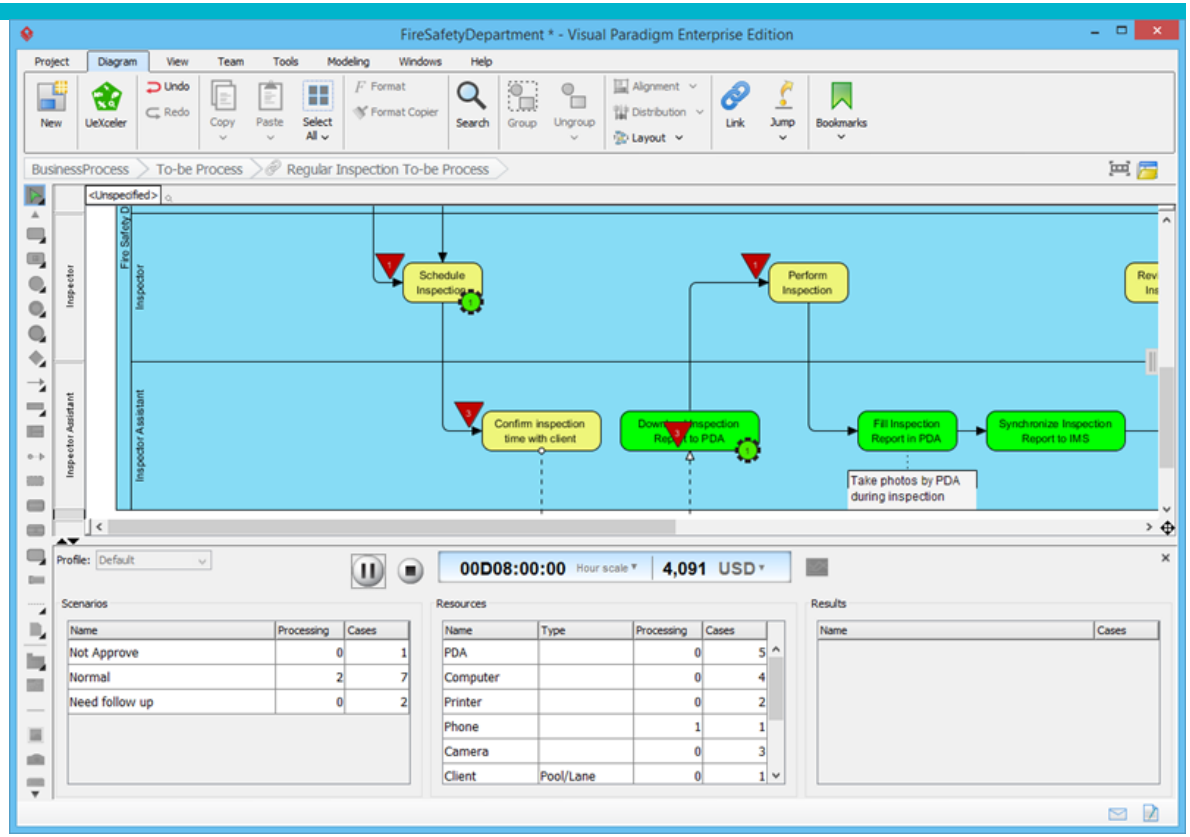


Modeling languages
Examples: BPMN



Modeling tools BPMN

Example:
Visual Paradigm



Simulate execution of business process to

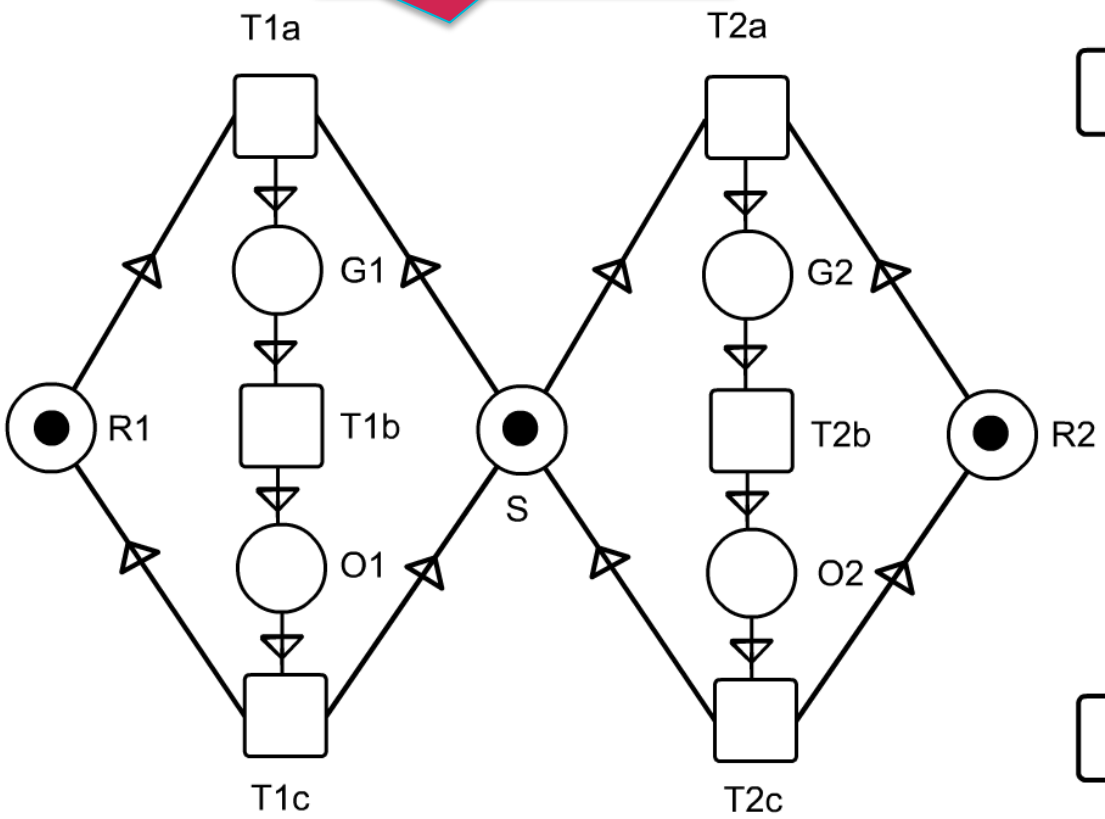
- Study resource consumption (e.g. human resources, devices) throughout the process
- Evaluate cost
- Identify bottlenecks
- Compare design alternatives

Model verification and model checking

	Examples
Verify if the model has all desirable **properties** (structural, behavioural, temporal, …)	• **Reachability** • **Safety**: something should never happen • **Liveness**: something must eventually happen • **Fairness**: every possible process should be executed infinitely often
Use most appropriate **formalism**	• linear temporal logic (LTL) • computation tree logic (CTL) • …
Use most appropriate **(model checking) tool**	• SPIN • Alloy • …

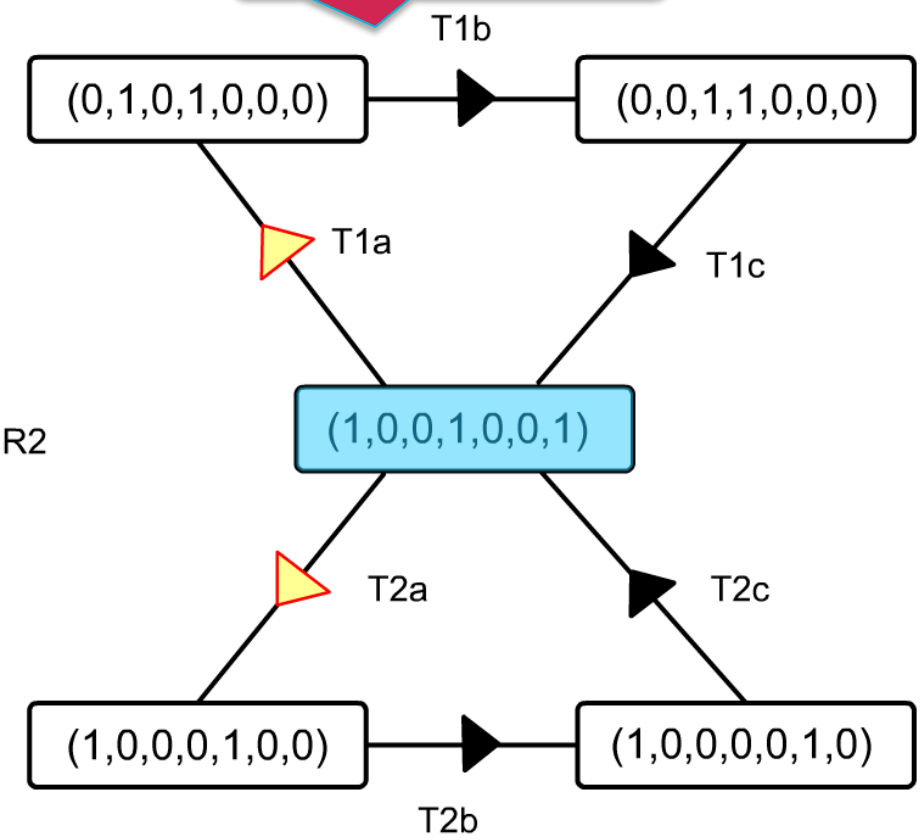
Model verification and model checking
Example : Reachability analysis

Petri net model



Vector = (R1,G1,O1,R2,G2,O2,S)

Reachability graph



See “Petri Nets World” for more information
http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst

Model verification and model checking
Example : Invariant analysis

R1 + G1 + O1

Place Invariant 1

R2 + G2 + O2

Place Invariant 2

R1 + R2 - S

Place Invariant 3

S + G1 + G2 + O1 + O2

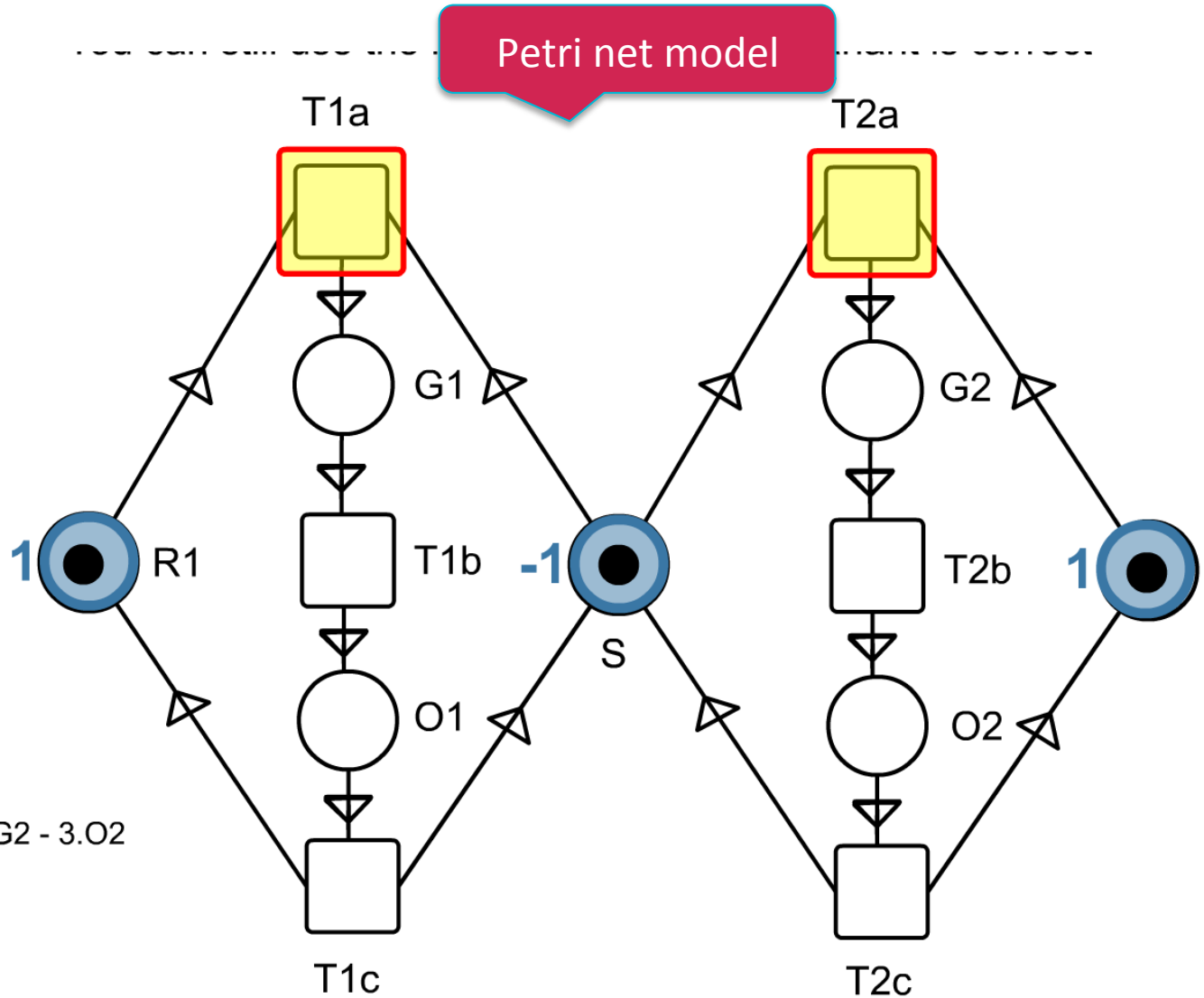
Place Invariant 4

R1 + G1 + O1 + R2 + G2 + O2

Place Invariant 5

2.R1 + 2.G1 + 2.O1 - 3.R2 - 3.G2 - 3.O2

Place Invariant 6



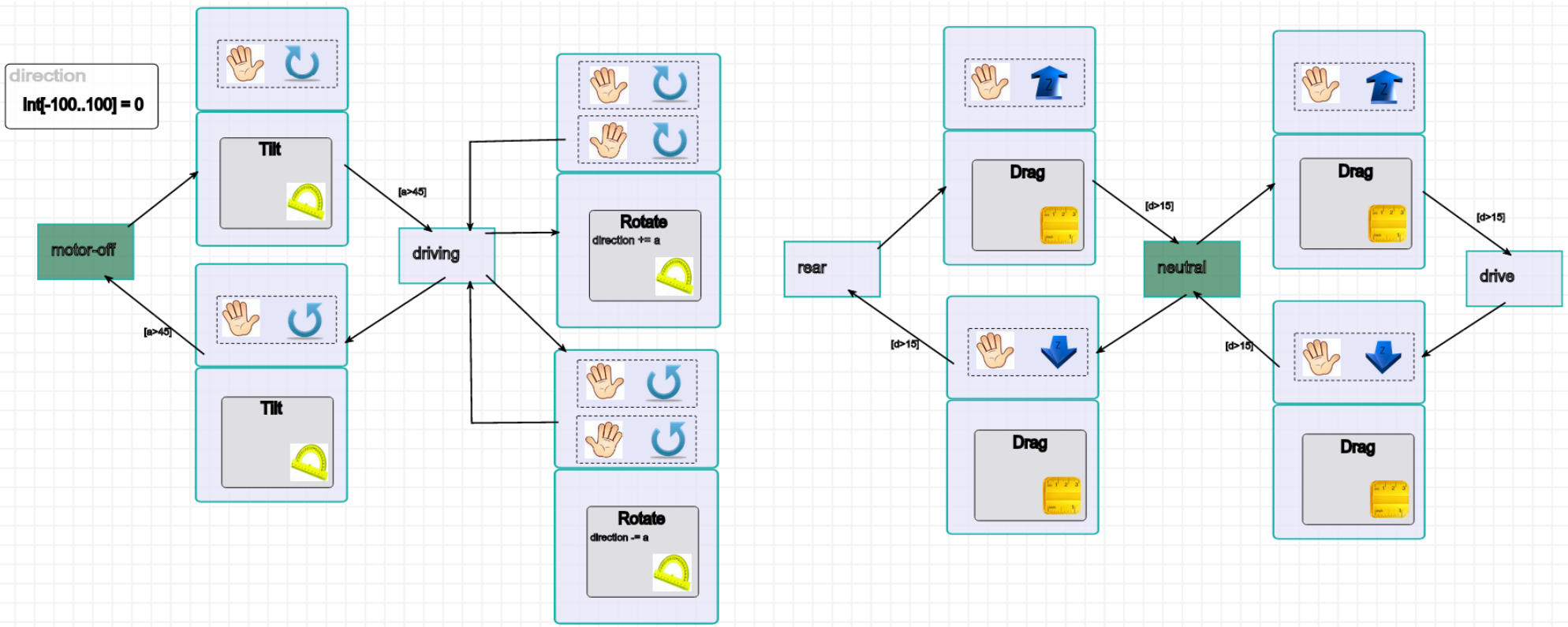
Challenges

Provide better tool support and formal support for

- Domain-specific modeling
  - Express models in a language close to the domain expert
  - Examples: Human-computer interaction modeling, robot control systems
- Model-based testing
  - Facilitate testing of models / generate automated tests from models
- Design space exploration
  - Evaluate the qualities of alternative models
- Model evolution
  - Facilitate changing the model while preserving its desirable properties

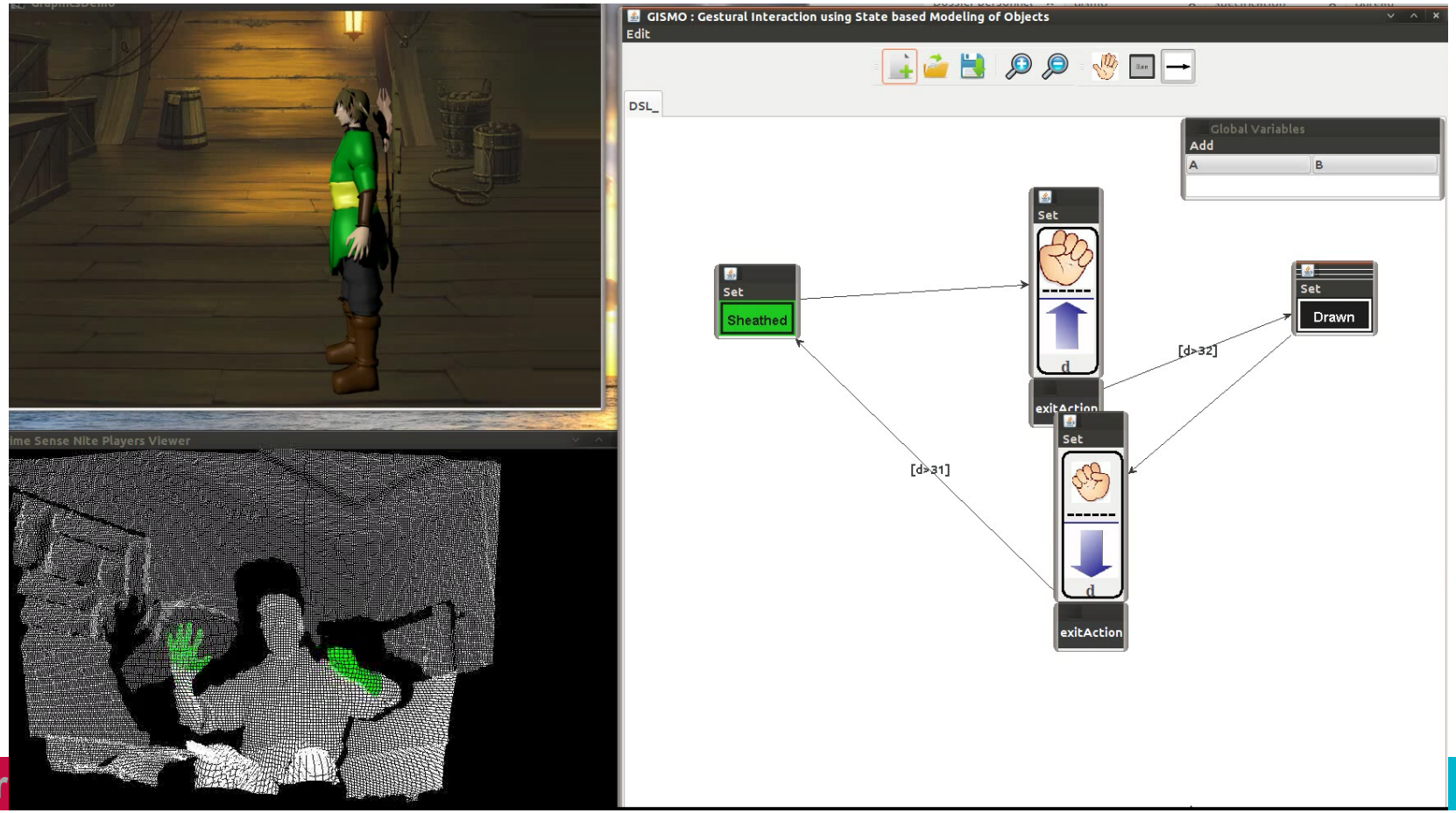
Challenges
Domain-specific software modeling

Example: A modeling language for gestural interaction (GISMO, PhD thesis Romuald Deshayes, UMONS, 2015)



Challenges
Domain-specific software modeling

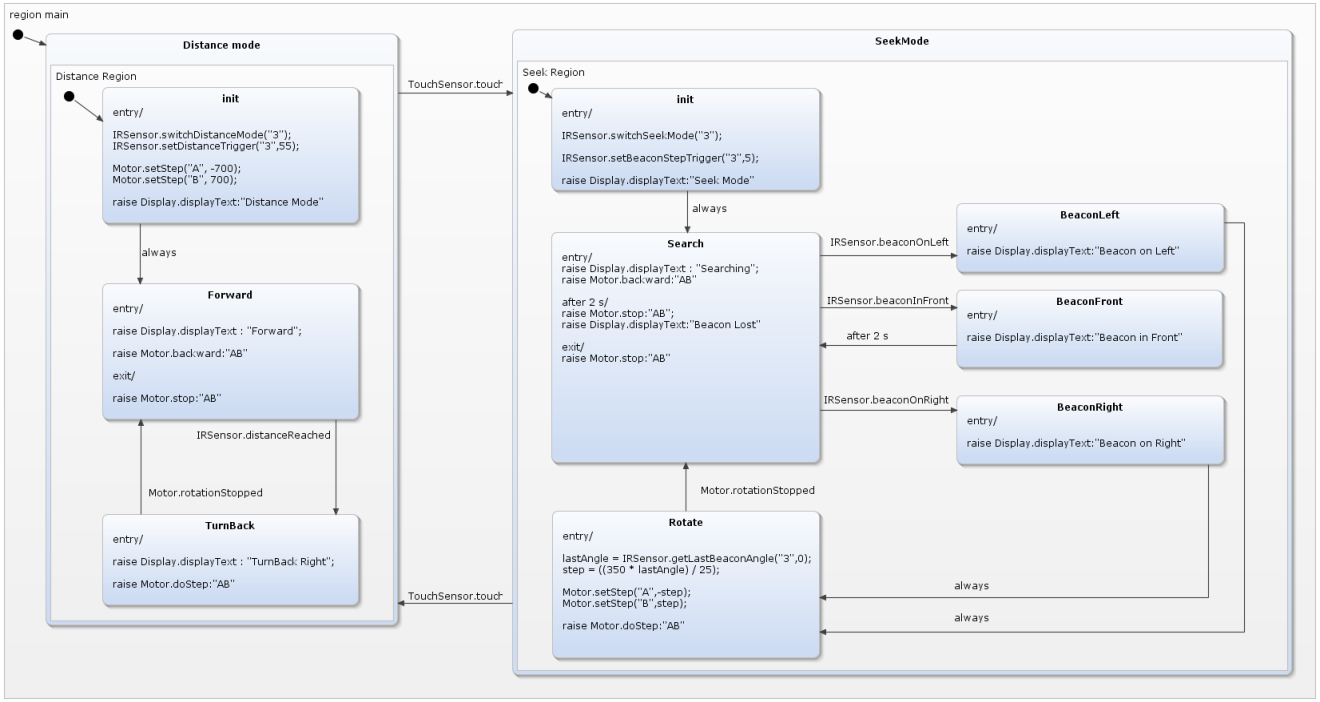
Example: A modeling language for gestural interaction (GISMO, PhD thesis Romuald Deshayes, UMONS, 2015)

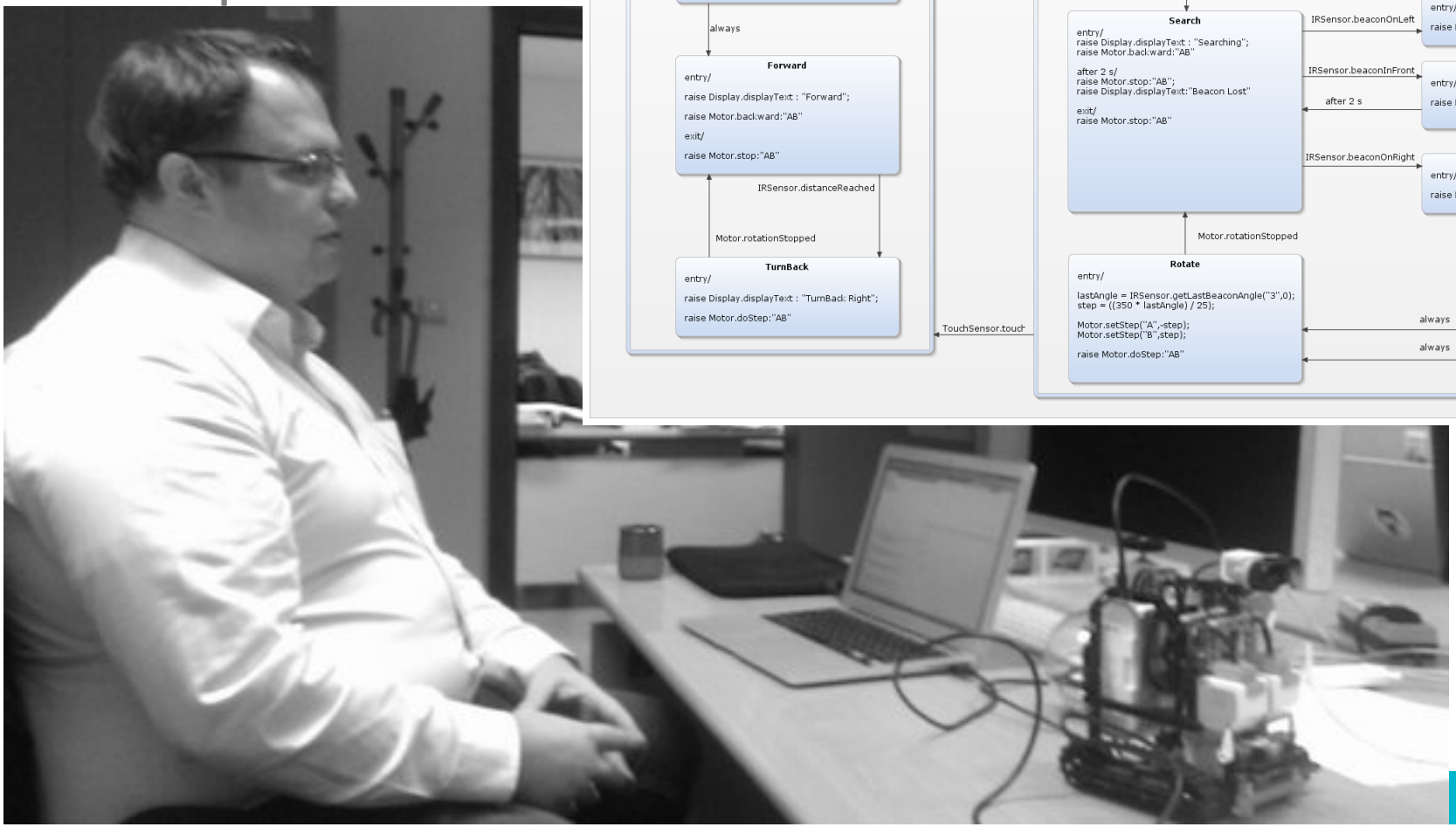


Challenges
Executable robot modeling

Mémoire

Christophe D'Hondt





Challenges Model-based testing

- Research in progress
  - Property-based testing
    - Generate simplest test cases that violate a desirably property / invariant
  - Design by contract
    - Express pre- and postconditions and invariants on the model
    - Raise exceptions during simulation/execution if contract violated
  - Test generation
    - Use model to generate tests for source code automatically
  - Mutation testing
    - To evaluate and improve existing test suite