Frédéric Boulanger
frederic.boulanger@supelec.fr

Cécile Hardebolle
cecile.hardebolle@supelec.fr

Execution of models with heterogeneous semantics

Tutorial on Critical Systems Simulation

2012, December 14

Outline

Heterogeneity in systems

Physical world

Sensors

Software

Networks

Control

Electronics

Aerodynamics

Actuators

Mechanics

Zoom on the power window

Model Driven Engineering

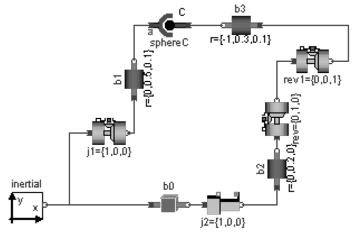
- Model Driven Engineering (MDE) approach
  ➥ intensive use of models during the whole engineering process

- Models of the system under design are used for:
  - Simulating the behavior of the system ☛ *Does it look OK?*
  - Exploring all the possible execution paths ☛ *Is it always OK?*
  - Testing the system ☛ *Is it OK in a particular case?*

+ Model of the <u>environment</u> of the system under design

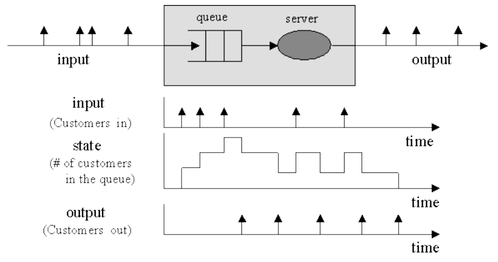
+ Models of the <u>design constraints</u> (cost, time, space, energy, safety, security…)

Modeling the power window



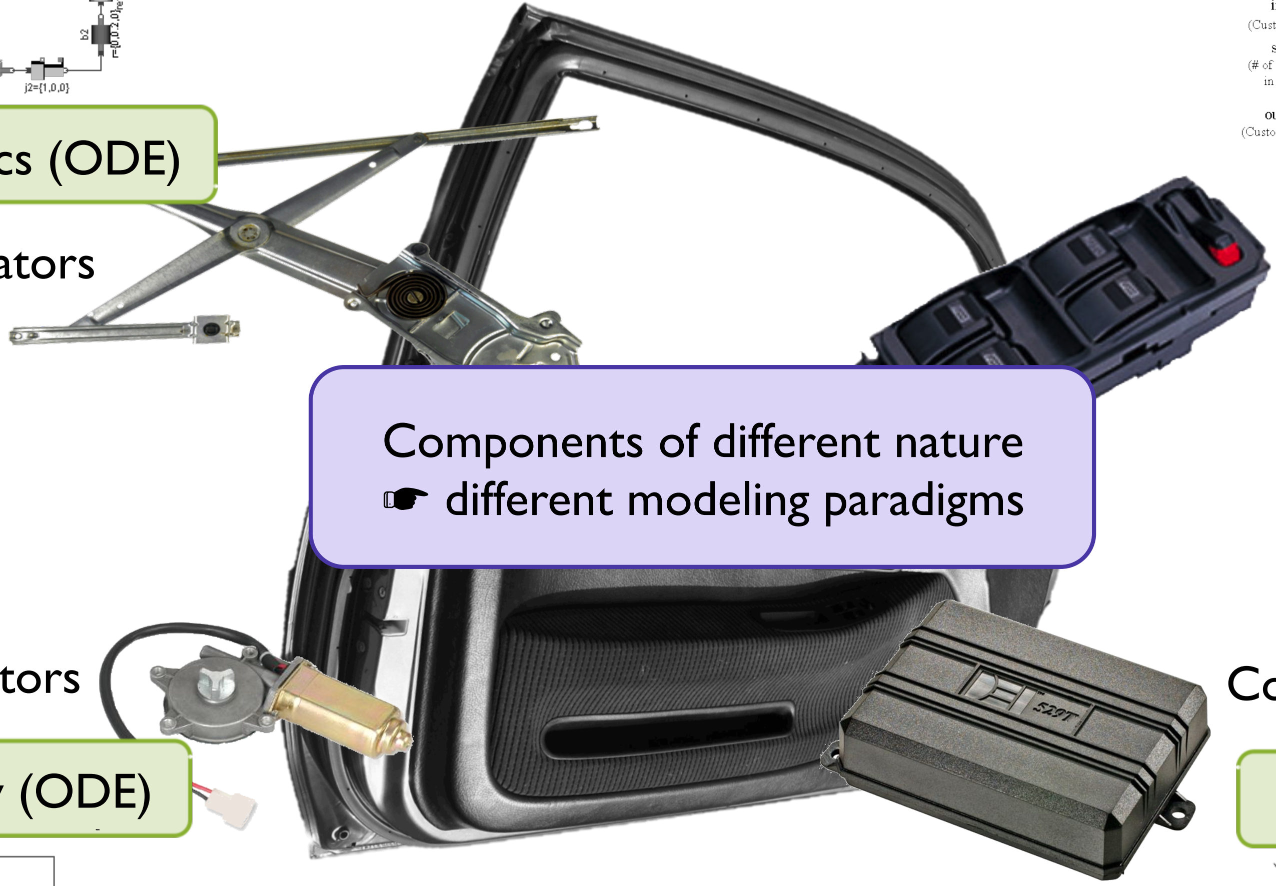
Mechanics (ODE)

Regulators



Discrete events

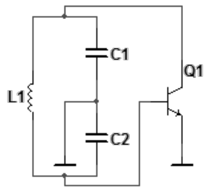
Sensors + bus



Components of different nature
☛ different modeling paradigms

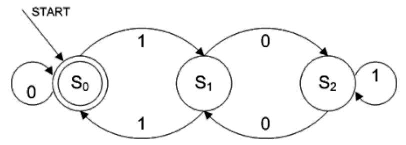
Actuators

Electricity (ODE)



Control

State machine



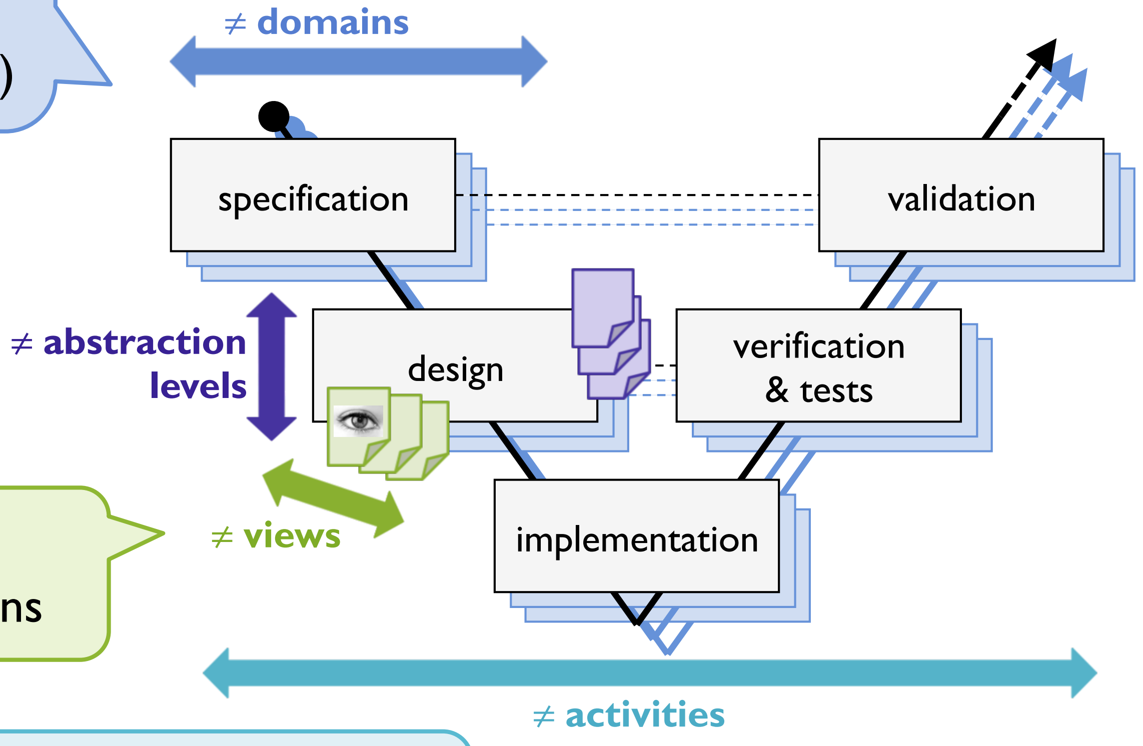
Heterogeneity in models

Components of different nature (signal processing, electronics, control…)

Different levels of abstraction

Functional and extra-functional concerns

Different activities during the process



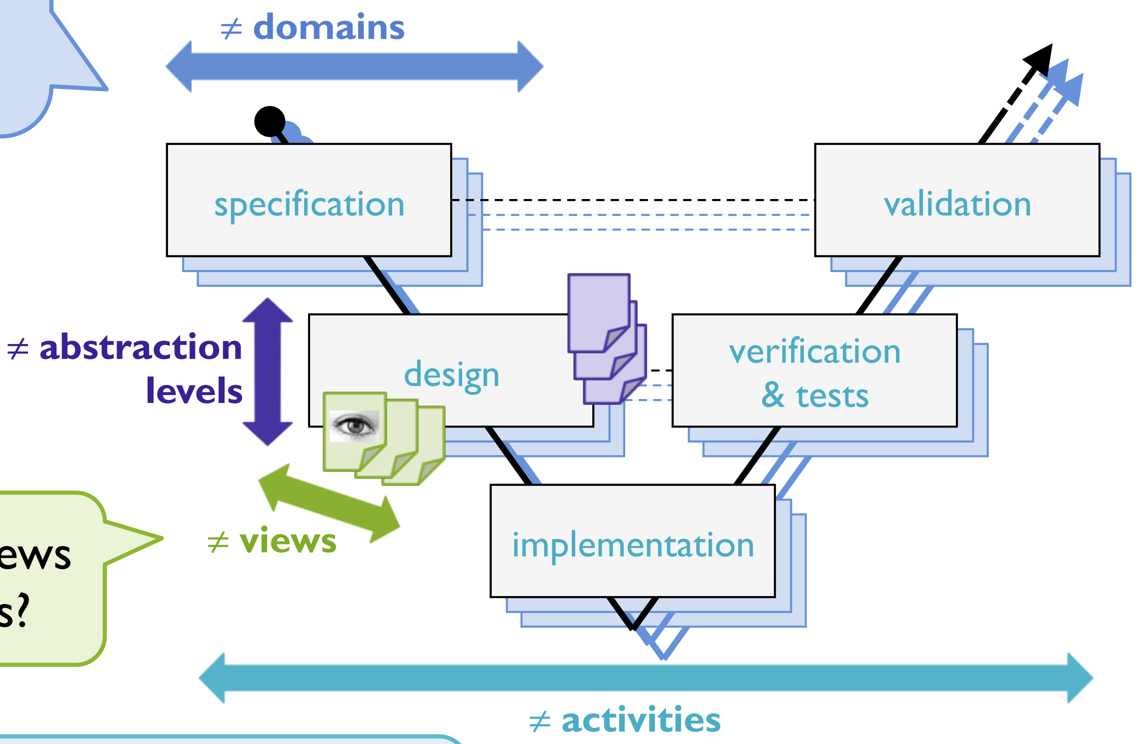
Issues with heterogeneity

How to compose models described with different modeling paradigms?

How to check the conformance of an implementation w.r.t a specification?

How to synchronize the views during the design process?

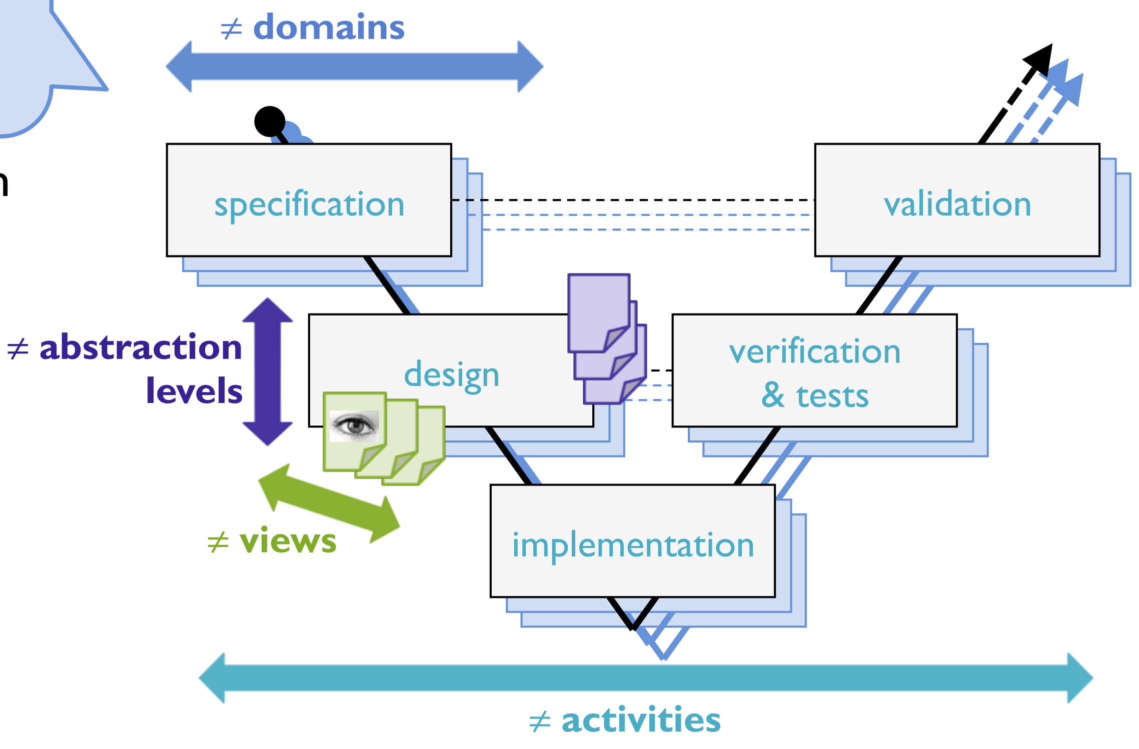
How to check the consistency of models used for different activities?



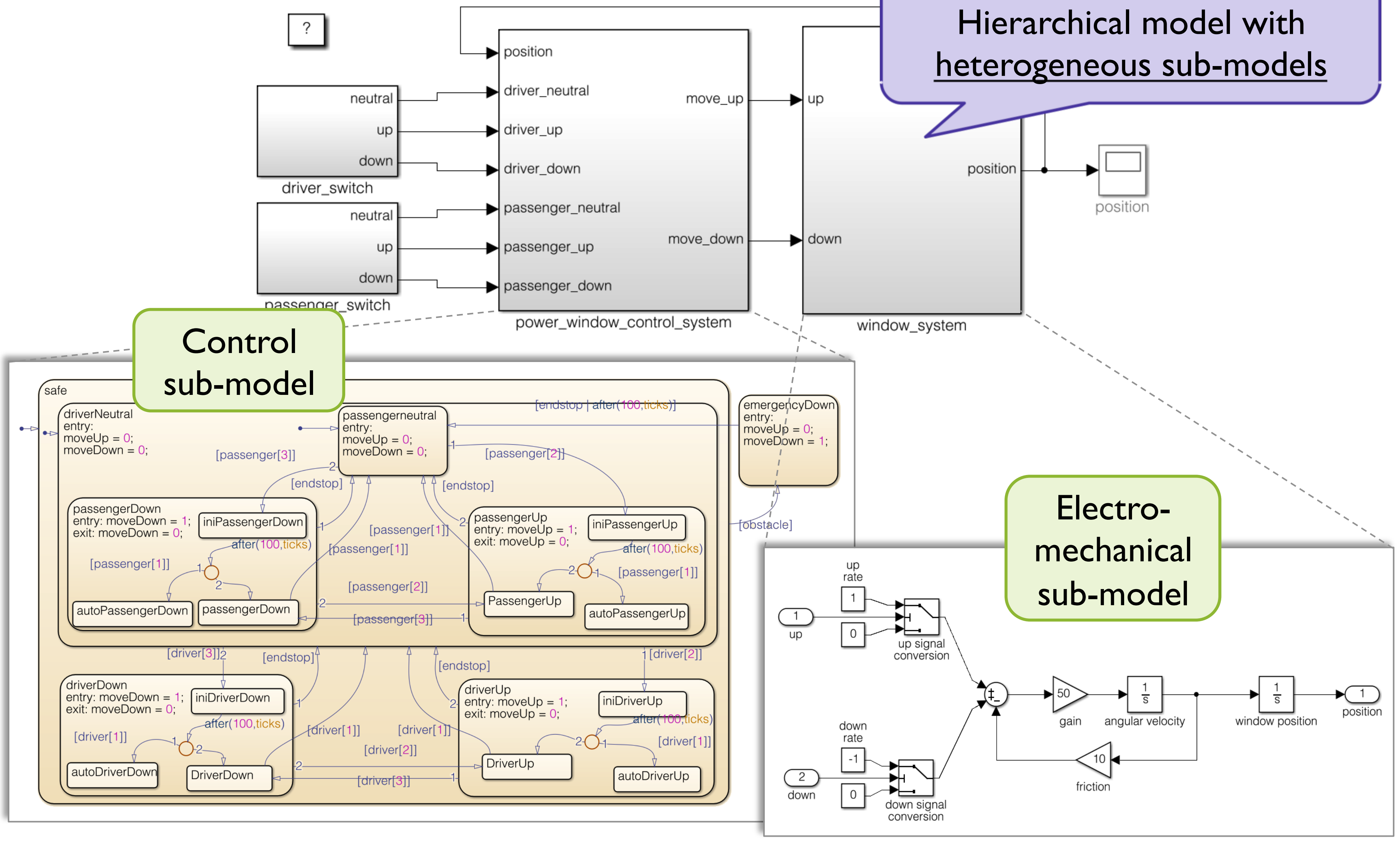
This presentation is about…

How to compose models described with different modeling paradigms?

+ Focus on simulation



Modeling the power window in Simulink/Stateflow



Hierarchical model with <u>heterogeneous sub-models</u>

Control sub-model

Electro-mechanical sub-model

Execution of heterogeneous models?

In order to be able to perform analysis (execution, verification, test) on a model obtained by composition of heterogeneous sub-models:

1. The sub-models must have a well defined meaning
2. The composition mechanism must be well defined

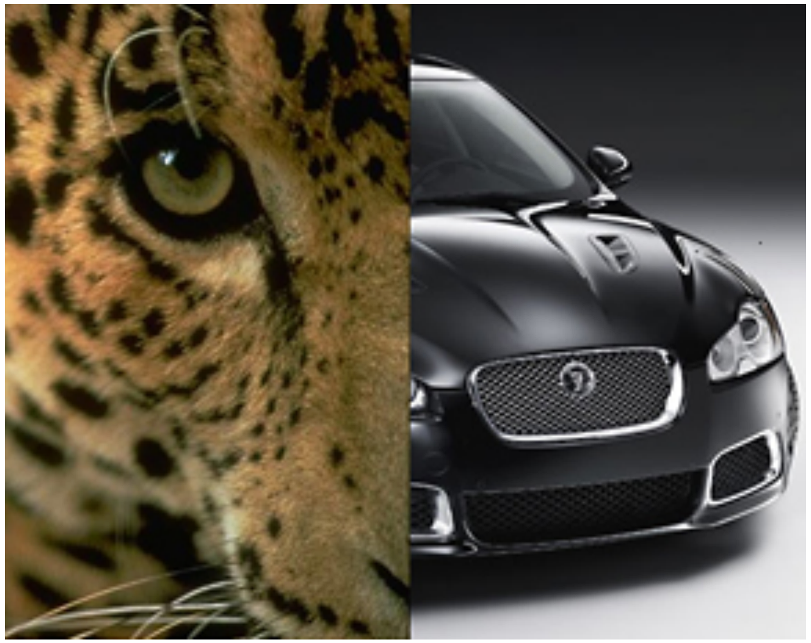
Notion of Semantics

☛ Necessary so that the global model can also have a well defined meaning!

Outline

What is semantics?

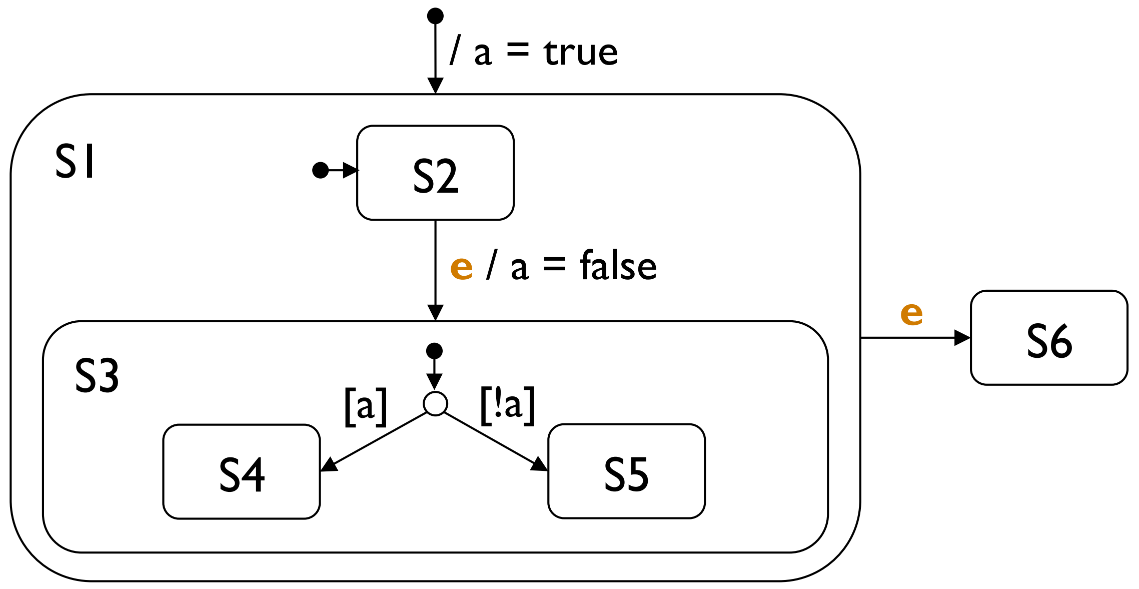
- What is the meaning of jaguar?



The problem with semantics…

Taken from:
"UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal"
Michelle Crane, Juergen Dingel

- What is the behavior described by this Statechart diagram when the event e occurs?



- Event e may lead to:
  - S4 with UML: outer transition to S1 has priority and sets a to true
  - S5 with Rhapsody: transition from S2 to S3 has priority and sets a to false
  - S6 with Stateflow: outer transition preempts state S1

Explicit definition of semantics

- All three meanings for the diagram are correct…
  …The problem is that the semantics is implicitly defined by the tool !

  - What if:
    - The designer of a system thinks according to UML semantics
    - The code generator interprets the model according to Rhapsody's semantics
    - The verification is made according to Stateflow's semantics

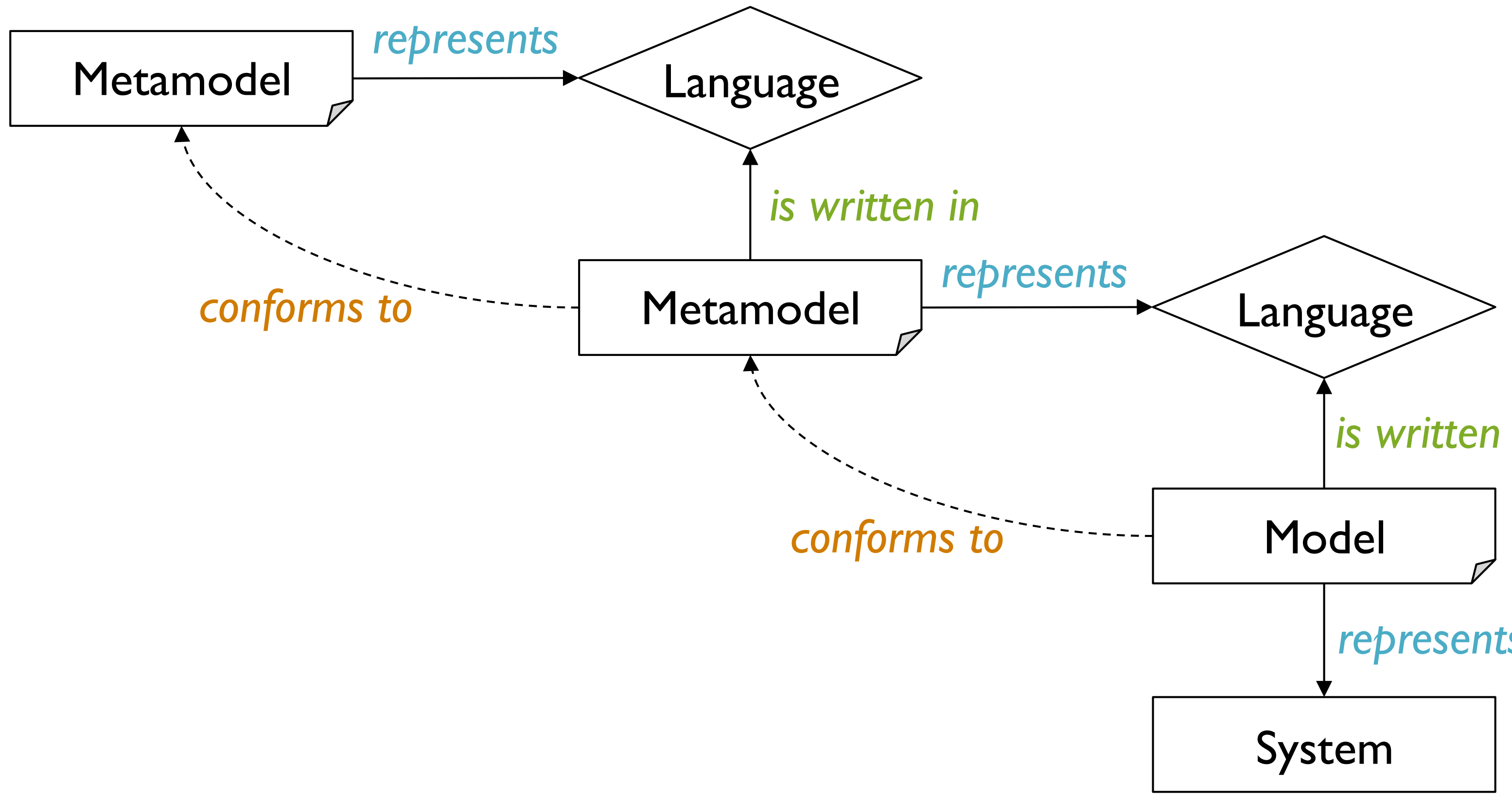
  $\Big\}$?

☛ The semantics of a model should be:

- Explicit, so that there is no doubt about how to interpret it
- Well defined, so that the properties of the model can be verified

- Formal semantics = semantics defined in such a way that a model can be processed automatically in a consistent way by programs

Model, metamodel and modeling language



Defining the semantics of a language

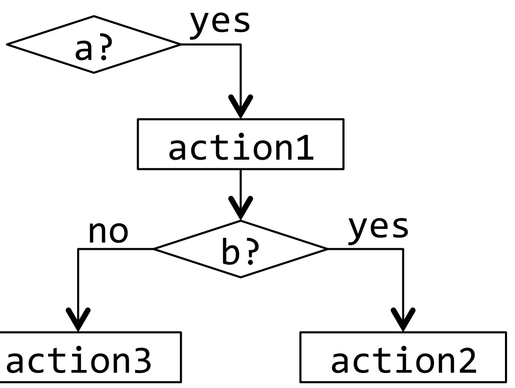
- The formal semantics of a language is based on its syntax
  - Abstract syntax = concepts and relations (metamodel)
  - Concrete syntaxes = text or graphics that obey a grammar

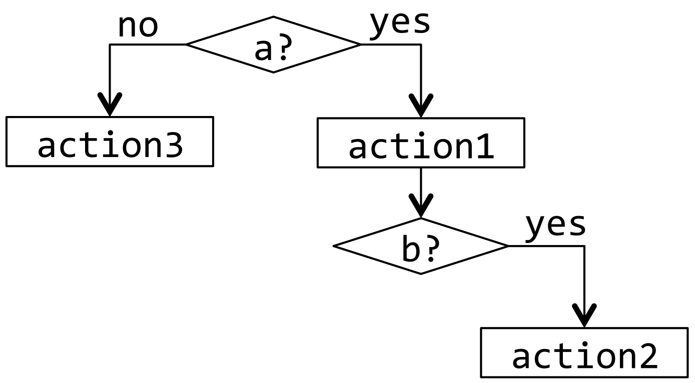
"Things must be *well written* to be *well understood*"

```
if (a) then
do action1
if (b) then
do action2
else
do action3
```

Does the `else` correspond to:

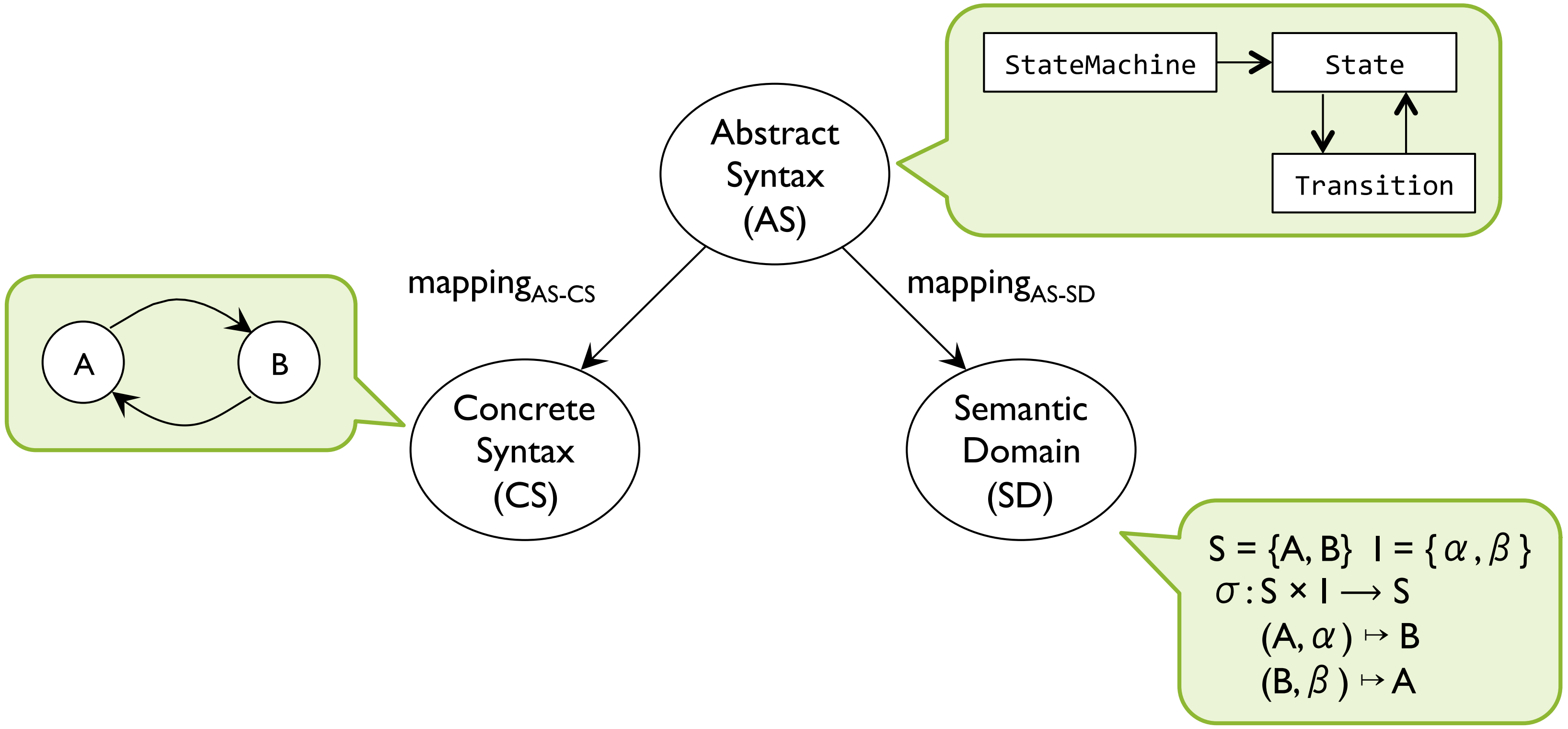
- not a?
- a and not b?





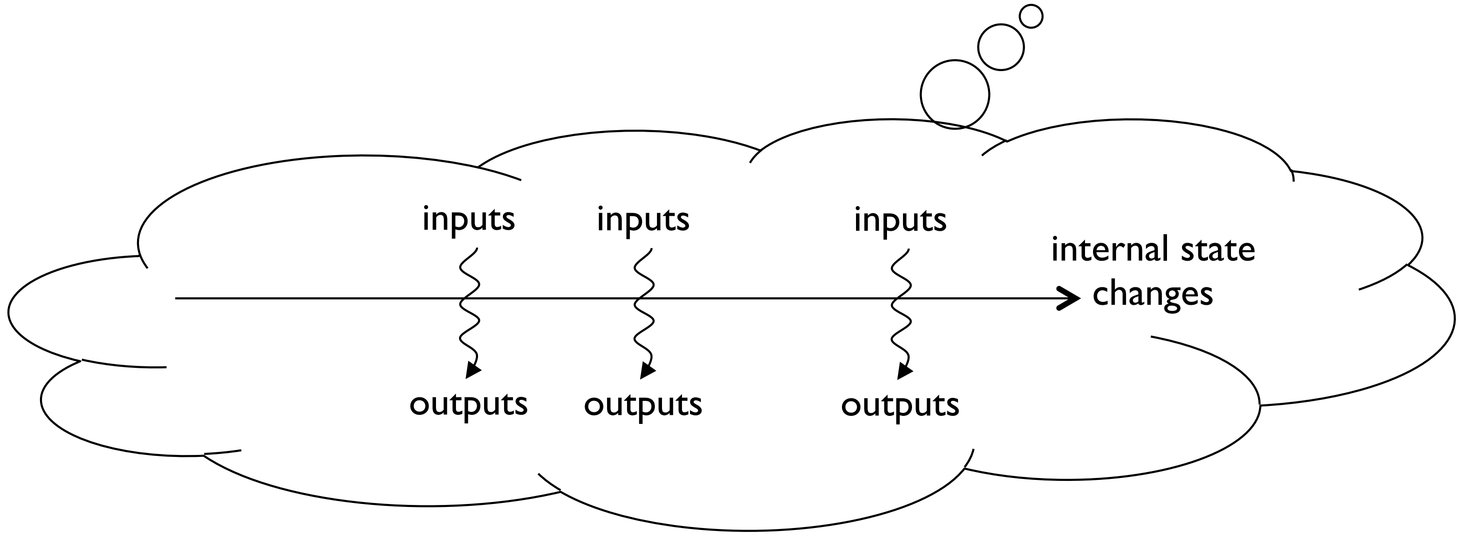
Defining the semantics of a language

- How to define the semantics?
  1. Choose a semantic domain (other language or mathematics)
  2. Define a mapping of the syntactic elements to items in the semantic domain



Execution semantics

How to describe the execution of a model?



☛ Semantic domain = abstract execution machine

Abstract execution machine = state + primitive operations

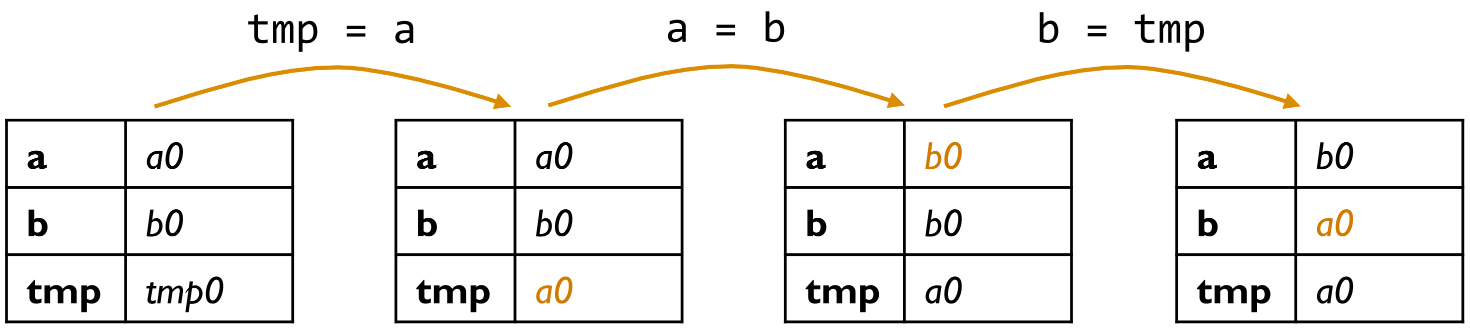
➥ The execution of the model is described in terms of changes in the state of the machine

Different flavors of semantics

- Operational semantics describes the execution of a model as a series of state changes of the execution machine
  - Example: how to swap two integers a and b?

Execution machine = state + primitive operations

```
tmp = a;
a = b;
b = tmp
```



a	*a0*
b	*b0*
tmp	*tmp0*

a	*a0*
b	*b0*
tmp	*a0*

a	*b0*
b	*b0*
tmp	*a0*

a	*b0*
b	*a0*
tmp	*a0*

- Operational semantics describes the complete sequence of states
  ➡ *May be too much detailed…*
  - Example: for the swap behavior, we don't care which variable is overwritten first!

Different flavors of semantics

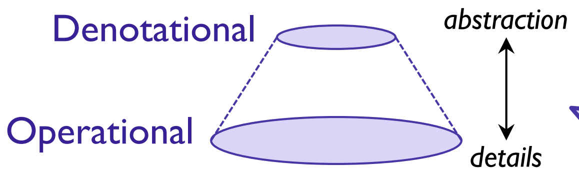
- Denotational semantics describes the path from initial to final state
  - Example: how to swap two integers a and b?

```
swap: initial state ⟼ new state
```

swap(a,b)

a	*a0*
b	*b0*
tmp	*tmp0*

a	*b0*
b	*a0*
tmp	*a0*



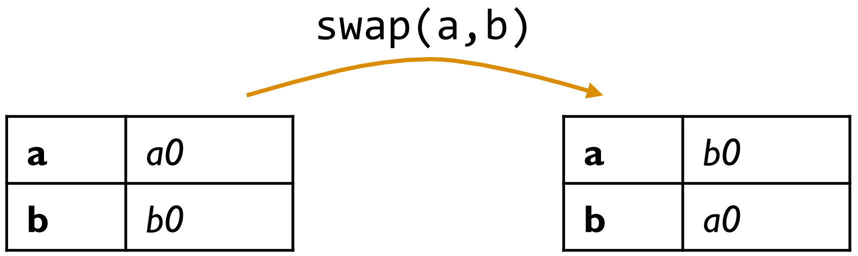
2 models
with equivalent *operational* semantics
have equivalent *denotational* semantics

- Denotational semantics describes the change of the complete state
  ➡ *May be too much detailed…*
  - Example: for the swap behavior, we don't care about the value of tmp at the end

Different flavors of semantics

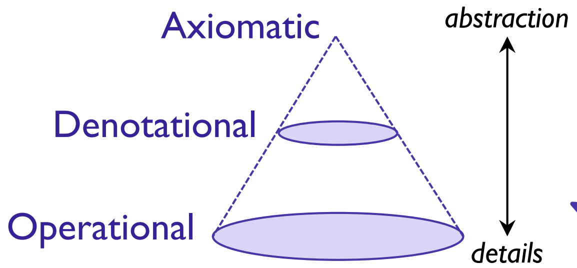
- Axiomatic semantics describes the change of the properties of the state
  - Example: how to swap two integers a and b?

```
{ a = a0 ∧ b = b0 } swap(a,b) { a = b0 ∧ b = a0 }
```



a	*a0*
b	*b0*

a	*b0*
b	*a0*



2 models
with equivalent *denotational* semantics
have equivalent *axiomatic* semantics

Different semantics, different uses

Formal semantics allows for unambiguous interpretation of models
☛ Execution, verification, computation of properties (timing, power…)

- Operational semantics describes the details of the execution
  - OK for simulation and code generation
  - Example: describe the execution steps for swapping a and b
- Denotational semantics describes the results of the execution
  - OK for verifying the correctness of the results
  - Example: obtain the values of `tmp`, a and b from the initial values of a and b
- Axiomatic semantics describes properties of the execution state
  - OK for verifying invariants, safety properties
  - Example: assert that the values of a and b have been swapped

Execution of heterogeneous models?

In order to be able to perform analysis (execution, verification, test) on a model obtained by composition of heterogeneous sub-models:

1. The sub-models must have a well defined meaning
2. The composition mechanism must be well defined

☛ Necessary so that the global model can also have a well defined meaning!

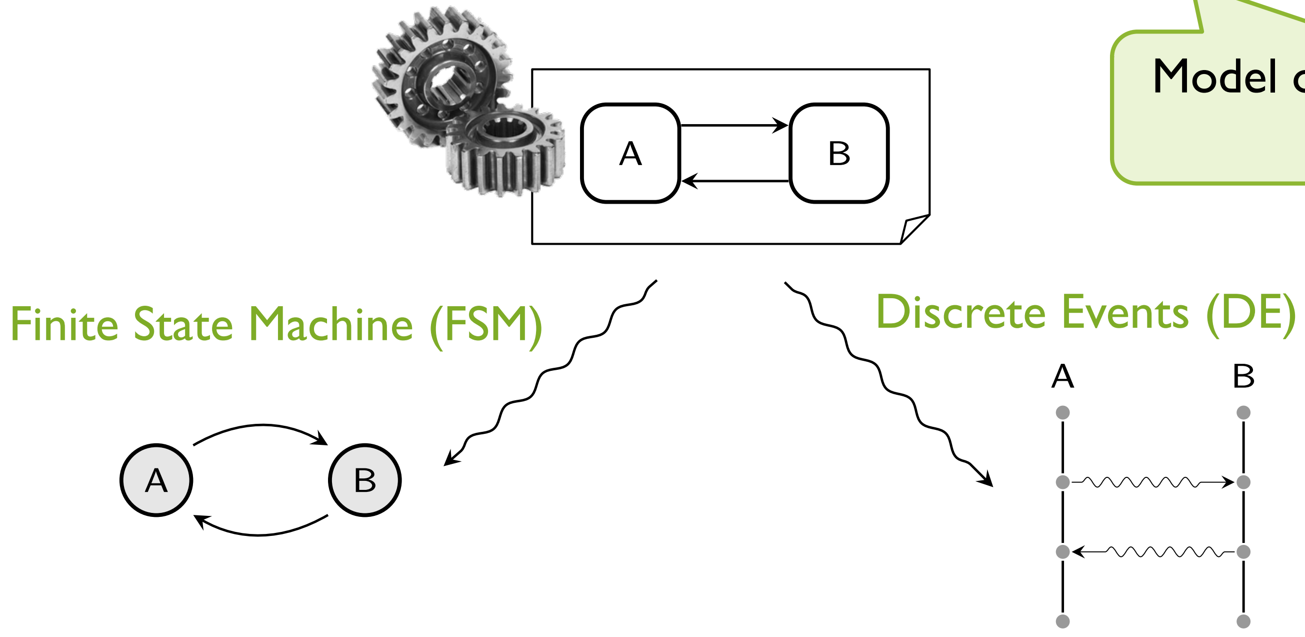
Outline

Model composition and execution

- Executing a heterogeneous model = co-executing several (sub-)models which have different semantics using a single tool…

- How can a single tool support several execution semantics?
  A. Tailored integration of several (sub-)tools
  B. Generic execution engine + generic abstract syntax + “pluggable” semantics

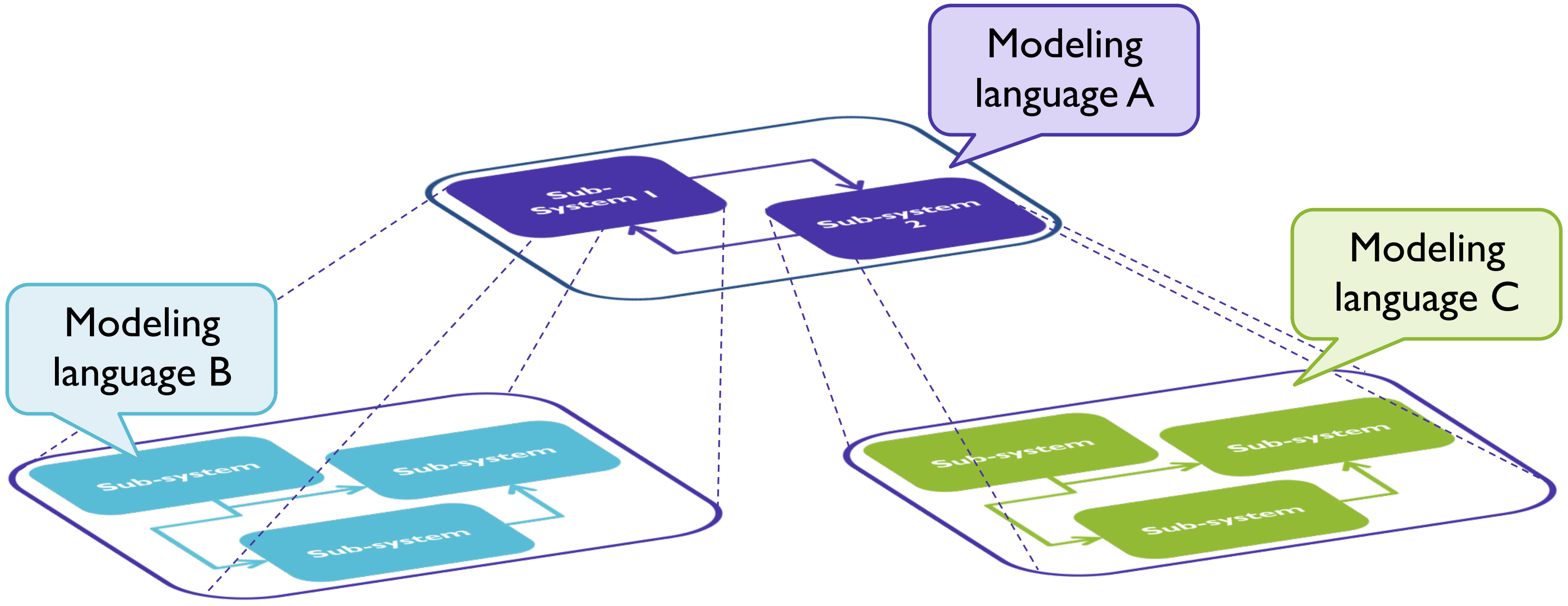
Model of Computation (MoC)



Concrete syntax can be specialized…

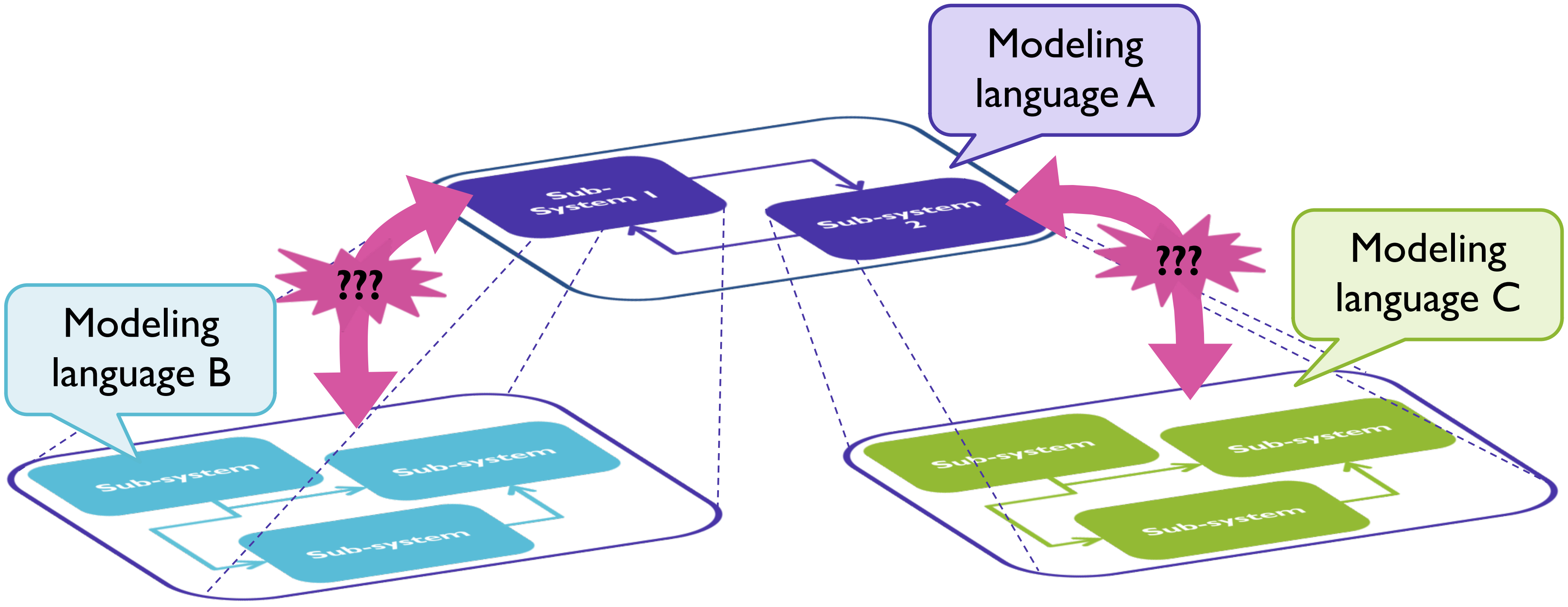
Hierarchy

- Hierarchy is used to reduce the complexity of models (black-box approach)
- Hierarchy + heterogeneity
  - ☛ rules inside a block ≠ rules outside the block

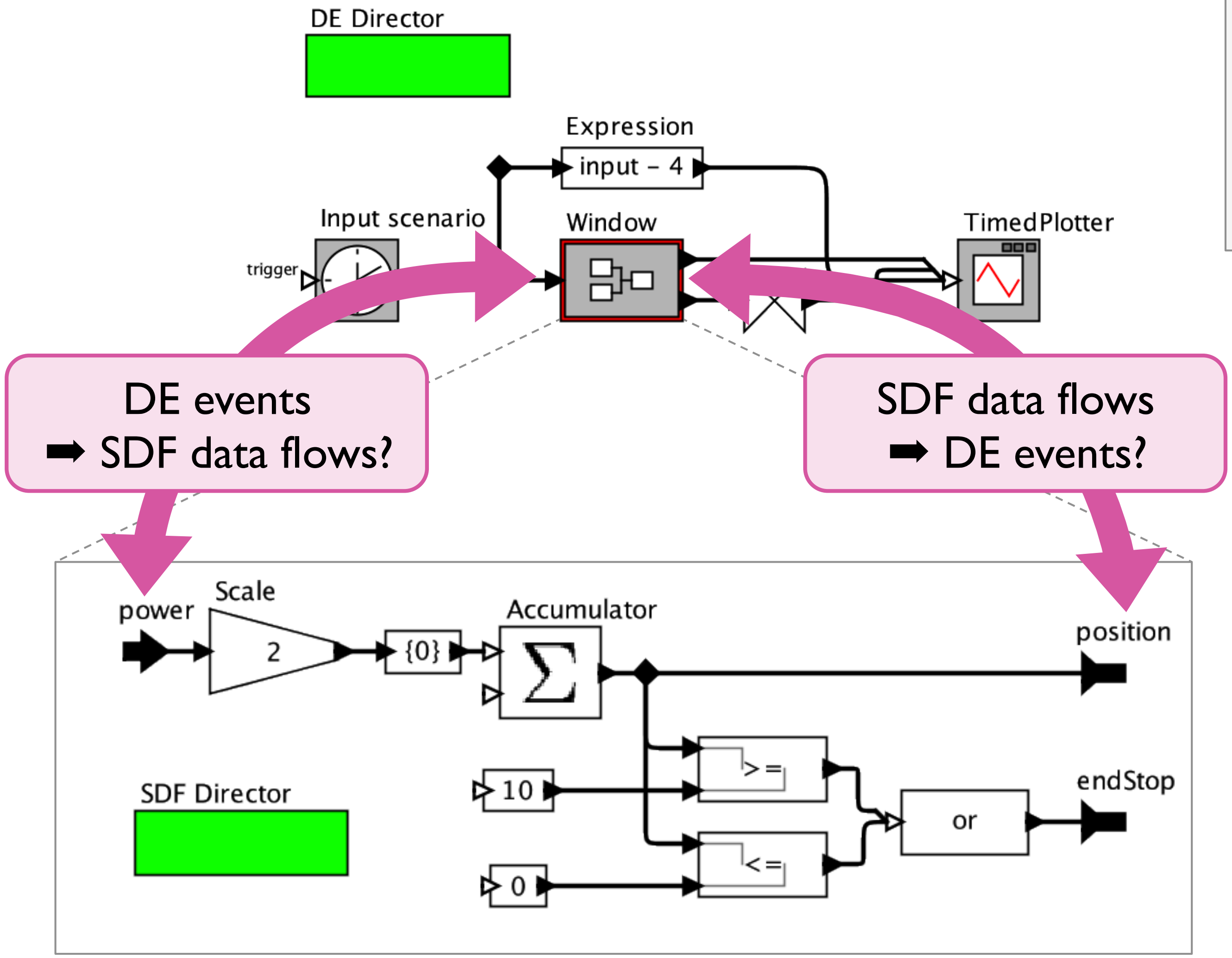


The problem with composition…

- What happens at the boundary between heterogeneous models?
  - Data flows versus events? Events versus functions of continuous time?
  - When should a model be updated?
  - Relations between discrete time, continuous time, series of samples?

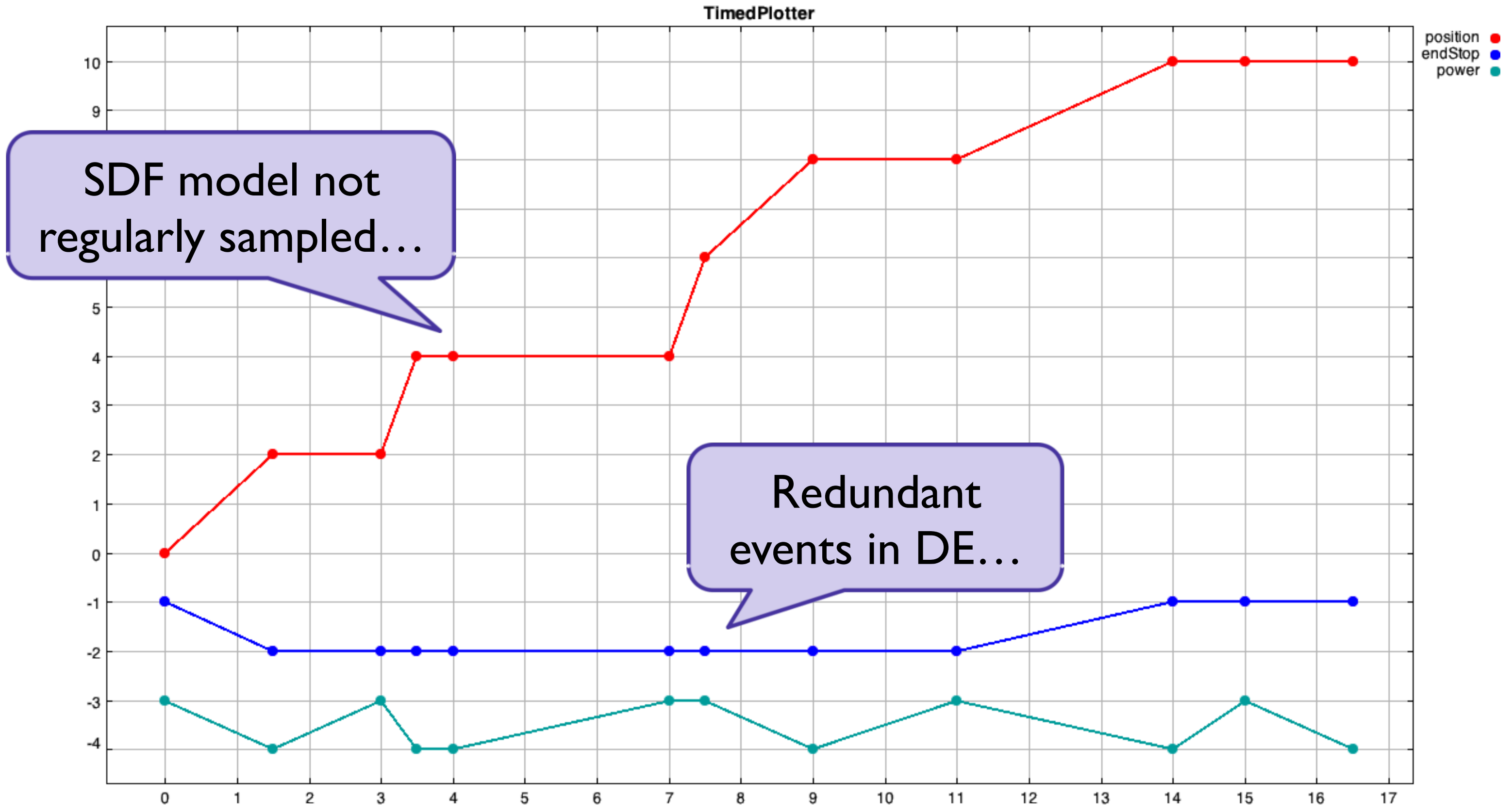


The power window example (simplified) in Ptolemy II



Simplification:
model of the window in "open loop"
+ stimulation with an input scenario

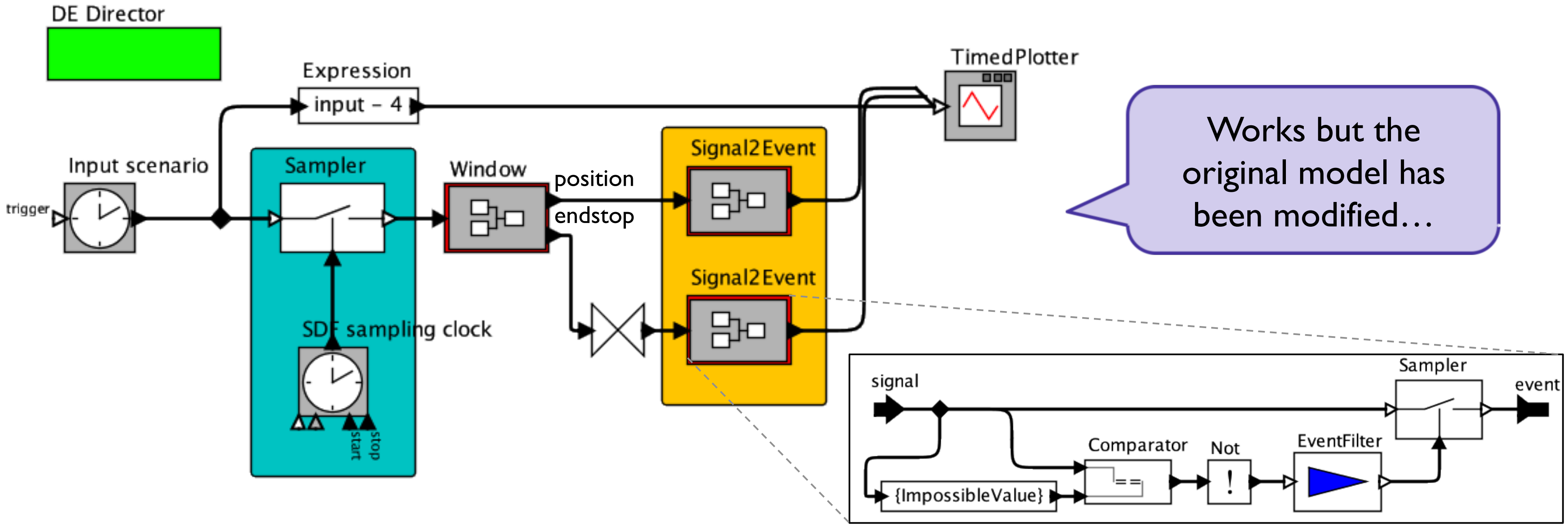
The power window example (simplified) in Ptolemy II



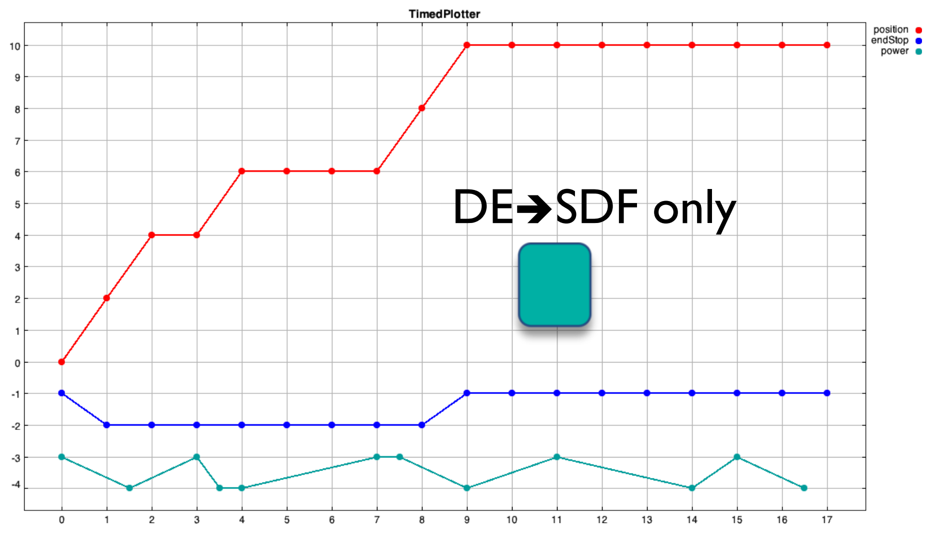
- Default adaptation:
  - The SDF model reacts only when events are processed in DE
  - DE events are produced in the DE model each time the SDF model reacts

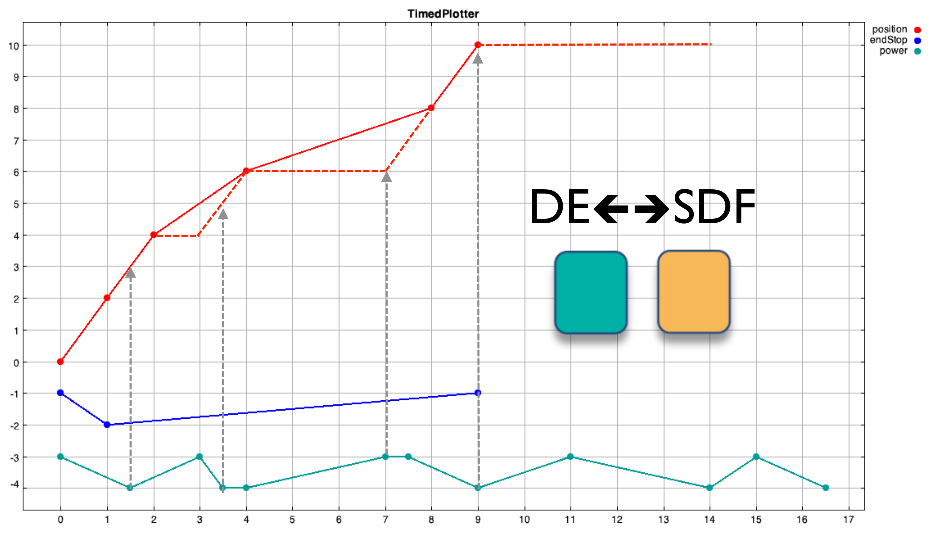
- Changing the adaptation means modifying one of the two models!

Adapted model in Ptolemy II



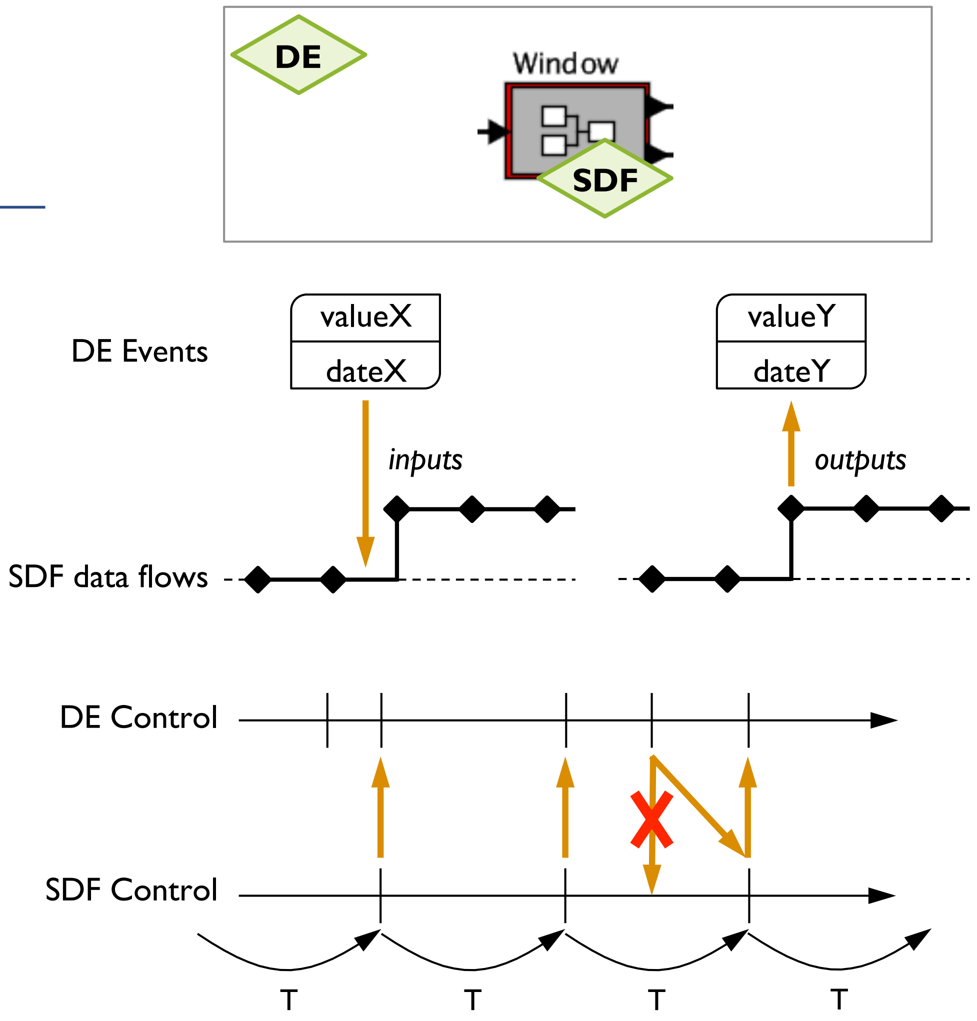
Works but the original model has been modified…





What is adaptation?

- Adaptation has three aspects:
  - Adaptation of data
    - Forms
    - Values
  - Adaptation of control flow
    - "Moments" at which "things" happen
  - Adaptation of time notions
    - Time scales
    - Time forms (seconds, revolutions, centimeters…)



- Explicit adaptation is as much important as explicit semantics for models!
- Adaptation should be separated from the models themselves to preserve modularity and reusability

Outline

Semantics and verification

- Well defined semantics ⇒ well defined behavior and properties
- Formal semantics ⇒ behavior can be analyzed automatically

- Verification is used to check for:
  - Unreachable states (dead code)
  - Properties that should always hold (security)
  - States that should always be reachable (liveness)
  - Forbidden operations (divide by zero, square root of negative number)
  - Value overflow

- Three flavors of verification:
  - Model-checking: complete, automatic, but combinatory explosion
  - Proof: complete, partially automated
  - Test: incomplete

Workflow

① Exploratory "informal" design

- Create a model
- Execute the model (simulate the behavior of the system)
- Iterate until the model seems to behave properly

② Formal design

- Formalize properties from the specification
- Check the properties
  - Properties OK → done
  - Property does not hold → understand why (counter example) and fix it

③ Implementation verification

- Generate code from the model
- Perform static analysis on the code to check that the properties hold
- Generate test scenarios and evaluate their coverage
- Test the real system using the test scenarios

Semantics and verification

Verification requires:

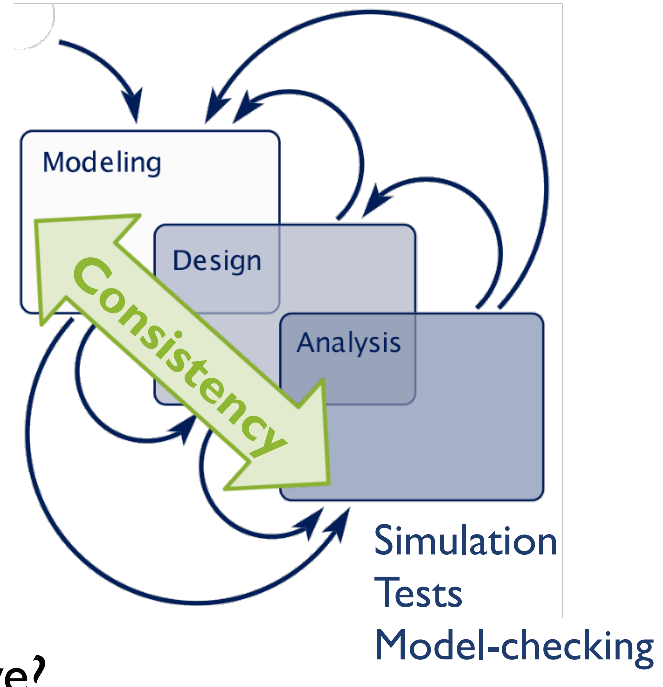
A. Precise semantics for each model
B. Precise semantics for the interactions between models

A. Tools for the verification of homogeneous models
- SCADE (Esterel Technologies): model-checking of synchronous reactive models
- Simulink Design Verifier (The MathWorks): proofs on Matlab/Simulink models
- Polyspace (The MathWorks): static analysis of C/C++ or Ada code
- Frama C (CEA, INRIA): static analysis of C code
- Krakatoa (Univ. Paris-Sud): static analysis of Java code
- … and many other theorem provers

B. Verification of heterogeneous models
- Some academic experimental tools for hybrid automata
- The future: combine proofs on homogeneous systems in a meta-logic (Isabelle)

Some issues with verification…

- Is the proof you made on the model of the system really valid on the system?
  - ➥ What You Prove Is What You Execute (WYPIWYE)



- Did you really prove what you wanted to prove?
  - ➥ What You Prove Is What You Mean (WYPIWYM)

$$\Box((up \land \neg obstacle) \Rightarrow \Diamond\, power = 1) \land \Box(down \Rightarrow \Diamond\, power = -1)$$

"When the user puts the switch in the up position the window closes unless there is an obstacle, and when the user puts the switch in the down position the window opens." (*liveness*)

Outline

Conclusion

- Complex systems
  - Are made of heterogeneous parts
  - Must be designed with their environment in mind
  - May be critical and require verifications ☛ analysis of models

- Heterogeneity in systems ☛ heterogeneity in models
  - 4 sources : domain, abstraction level, extra-functional views, activities

- Analysis on composition of heterogeneous models requires:
  - Explicit and well defined semantics for each model
  - Explicit and well defined composition mechanism (= semantic adaptation)

- Several techniques exist for the simulation of heterogeneous models
- For verification, heterogeneity is not well supported
  + issues with the consistency of the different models

Heterogeneous semantics is still a challenge,
but being aware of the issue helps avoiding traps and pitfalls!

Bibliography

Pieter J. Mosterman and Gautam Biswas
A Hybrid Modeling and Simulation Methodology for Dynamic Physical Systems
SIMULATION: Transactions of The Society for Modeling and Simulation International, January, 2002

Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.
Taming heterogeneity – the Ptolemy approach
Proceedings of the IEEE vol. 9, #1, 2003

Cécile Hardebolle, Frédéric Boulanger
Multi-Formalism Modeling and Model Execution
International Journal of Computers and their Applications, vol. 31, #3, July 2009

Frédéric Boulanger, Cécile Hardebolle, Christophe Jacquet, Dominique Marcadet
Semantic Adaptation for Models of Computations
Proc. of the International Conference on Application of Concurrency to System Design, 2011

Benoît Combemale, Cécile Hardebolle, Christophe Jacquet, Frédéric Boulanger, Benoît Baudry
Bridging the Chasm between Executable Metamodeling and Models of Computation
Proceedings of the 5th International Conference on Software Language Engineering, 2012

Marc Aiguier, Frédéric Boulanger, Bilal Kanso
A formal abstract framework for modeling and testing complex software systems
Theoretical Computer Science, vol. 455, October 2012