UNIVERSITY OF Southampton

Integrating Formal Verification and Simulation of Hybrid Systems

Vitaly Savicks, Michael Butler, John Colley

ESS Open Day 2014

ADVANCE

Hybrid Systems

- Mixture of interacting discrete-event and continuous-time components
- Different development tools for domain-specific components
- High complexity of components, particularly in safety-critical domain
- Concept of time is essential

We ♥ Formal Methods

- Great for discrete-event system modelling!

- How to verify timing constraints?

- Can we abstract the continuous dynamics of the environment?

- How to validate that our abstraction behaves correctly in a real environment?

Simulation

- Detailed mathematical models of the physical processes

- Industry-level tools with certified component libraries used to model mechanics, hydraulics, electrical circuits, power systems, etc.

- Simulation-based analysis and optimisation is a standard technique

Rodin + Simulation!

- ProB Model Checker does that already (animation), but only for the Rodin components.

- Why not use a standard?

- Functional Mock-up Interface for Co-Simulation — supported by multiple industry-level tools, cross-platform and open-source

- ProB 2.0 interface to FMI CS via Java FMI library

RMS

- ProB animator for the Event-B components and FMI interface for the FMI units (FMUs)

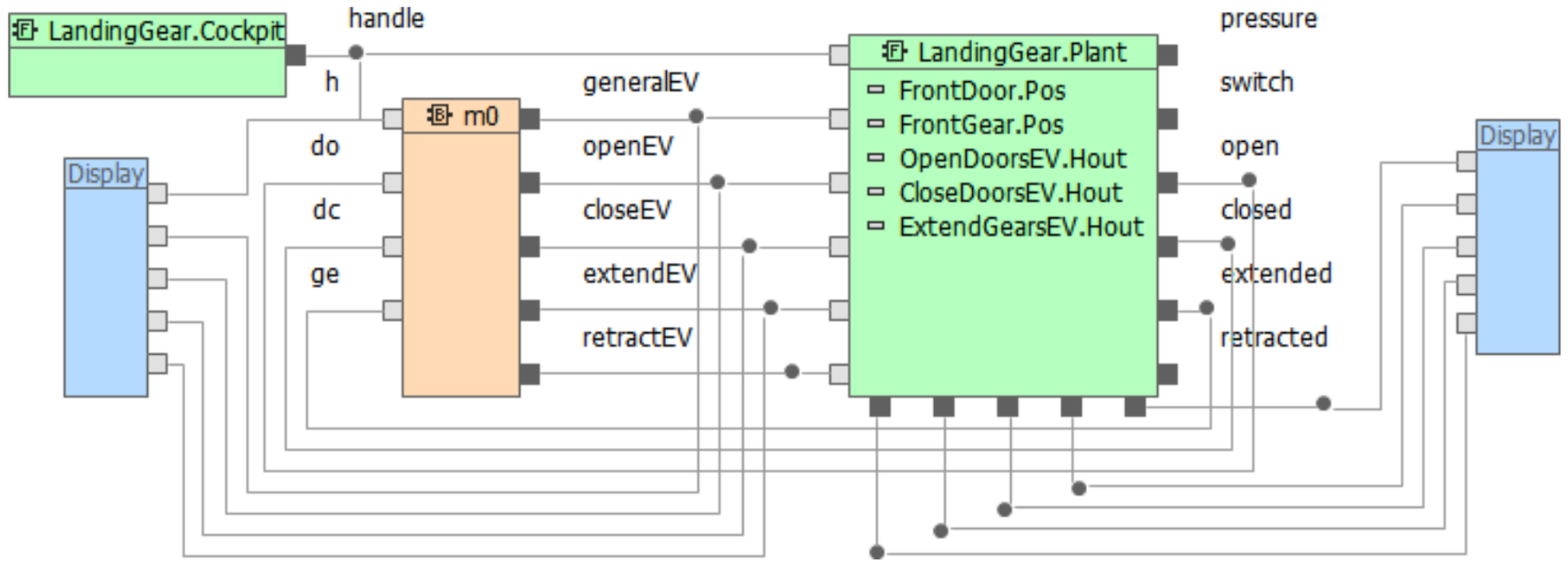
- Generic master algorithm, based on the two-list algorithm (“A Guide to VHDL”, P. Langstraat)

- Flexible formal modelling of Event-B components for co-simulation

- Graphical component composition and simulation environment

R M S

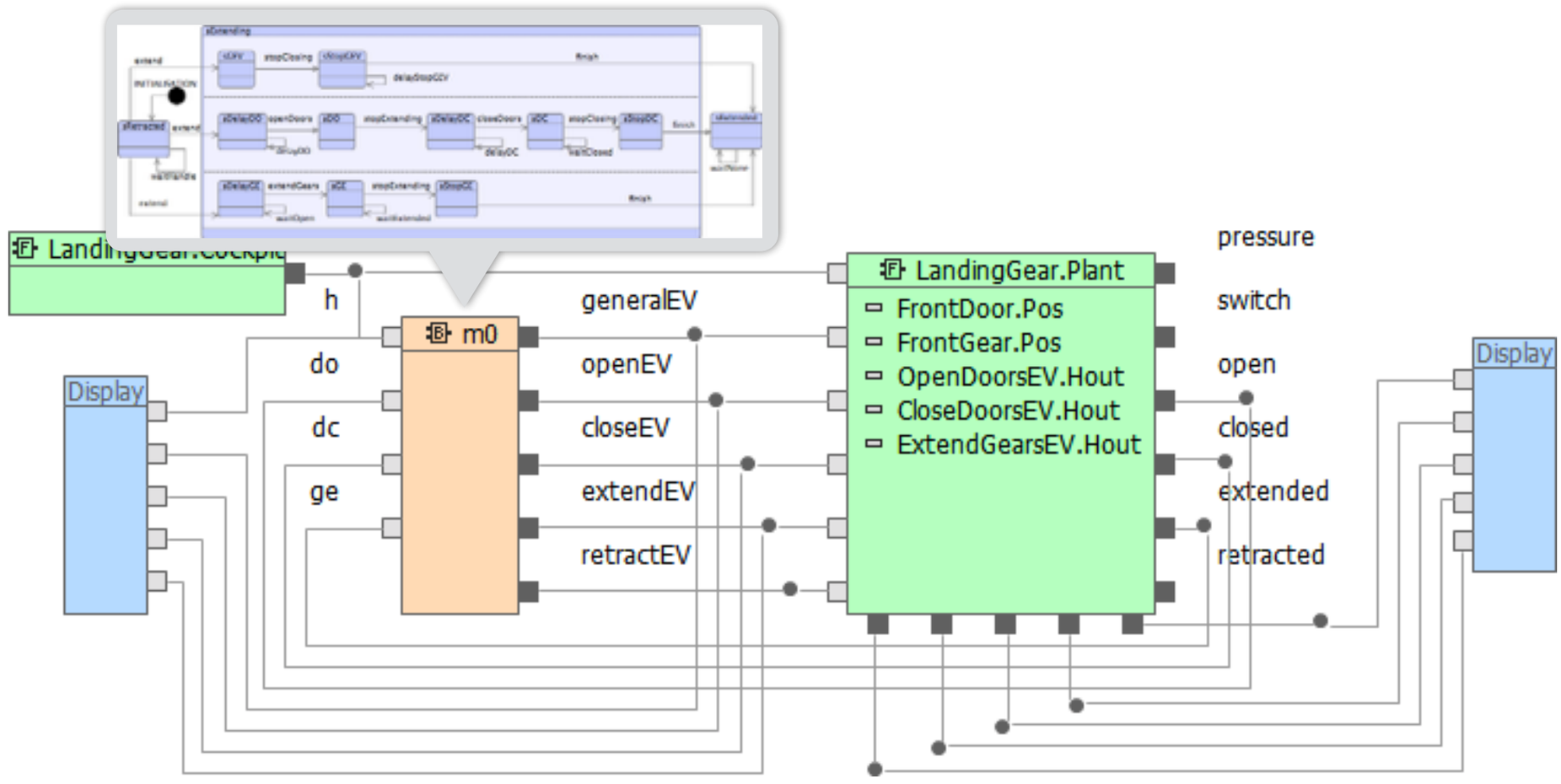
University of Southampton



ADVANCE

University of Southampton

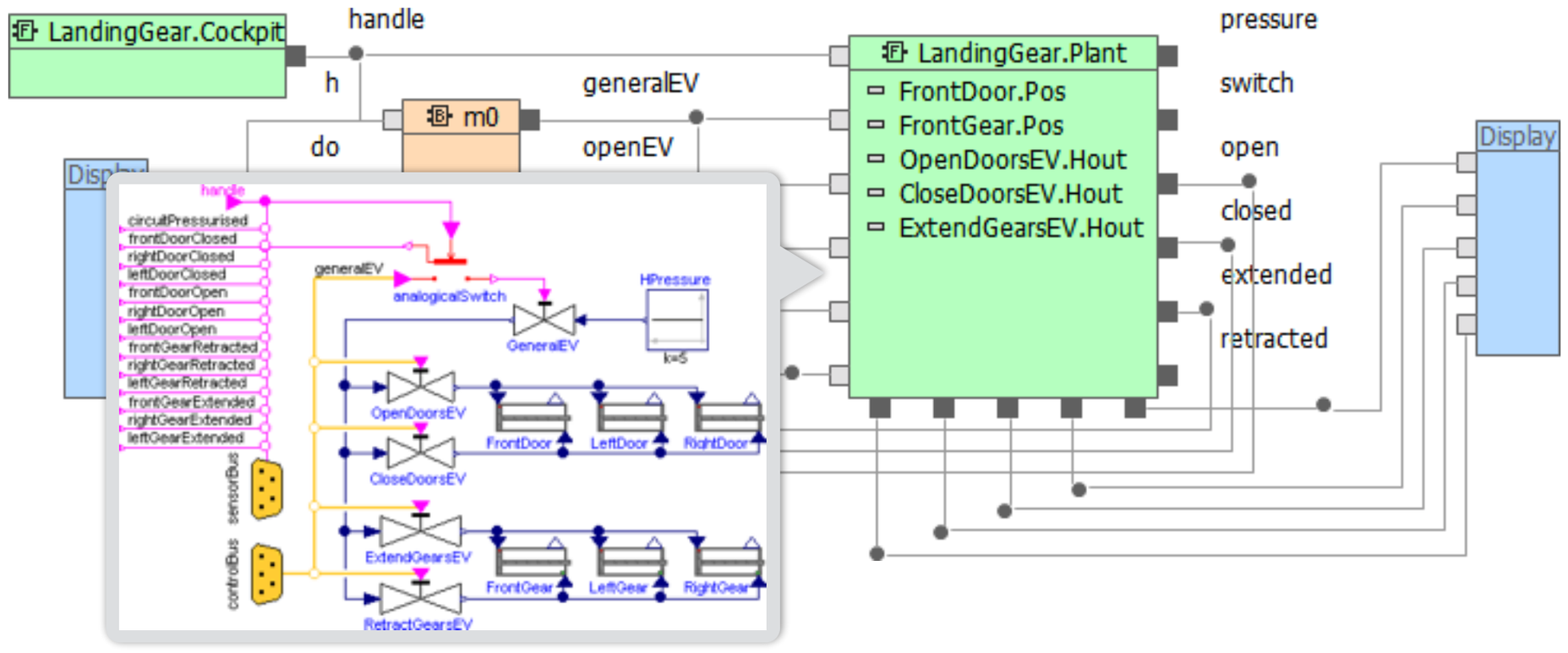
R M S



ADVANCE

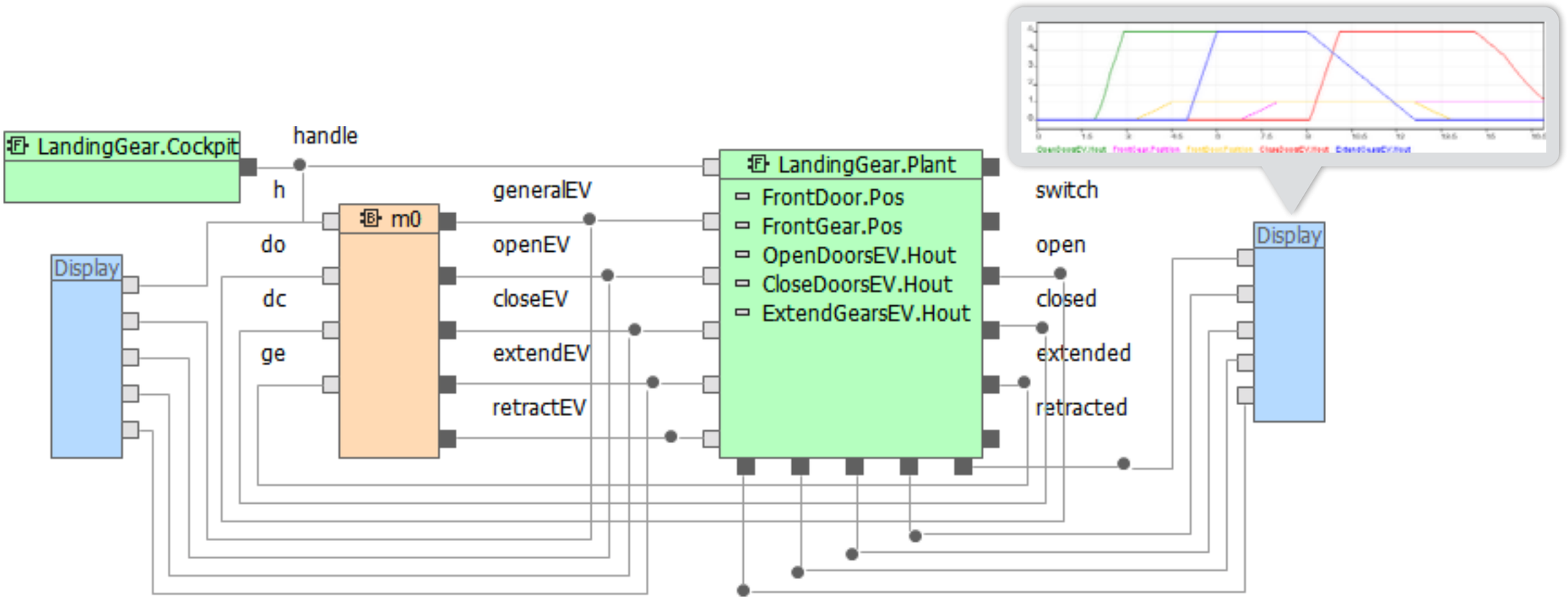
R M S

UNIVERSITY OF Southampton



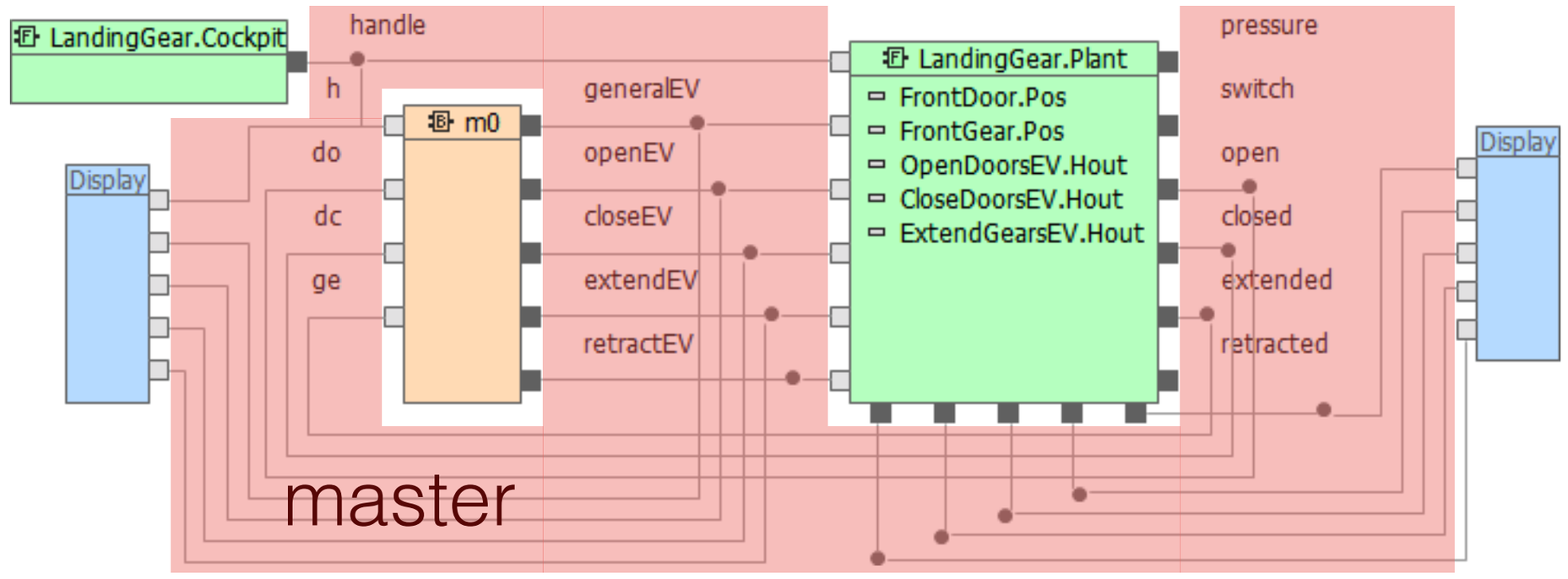
ADVANCE

R M S

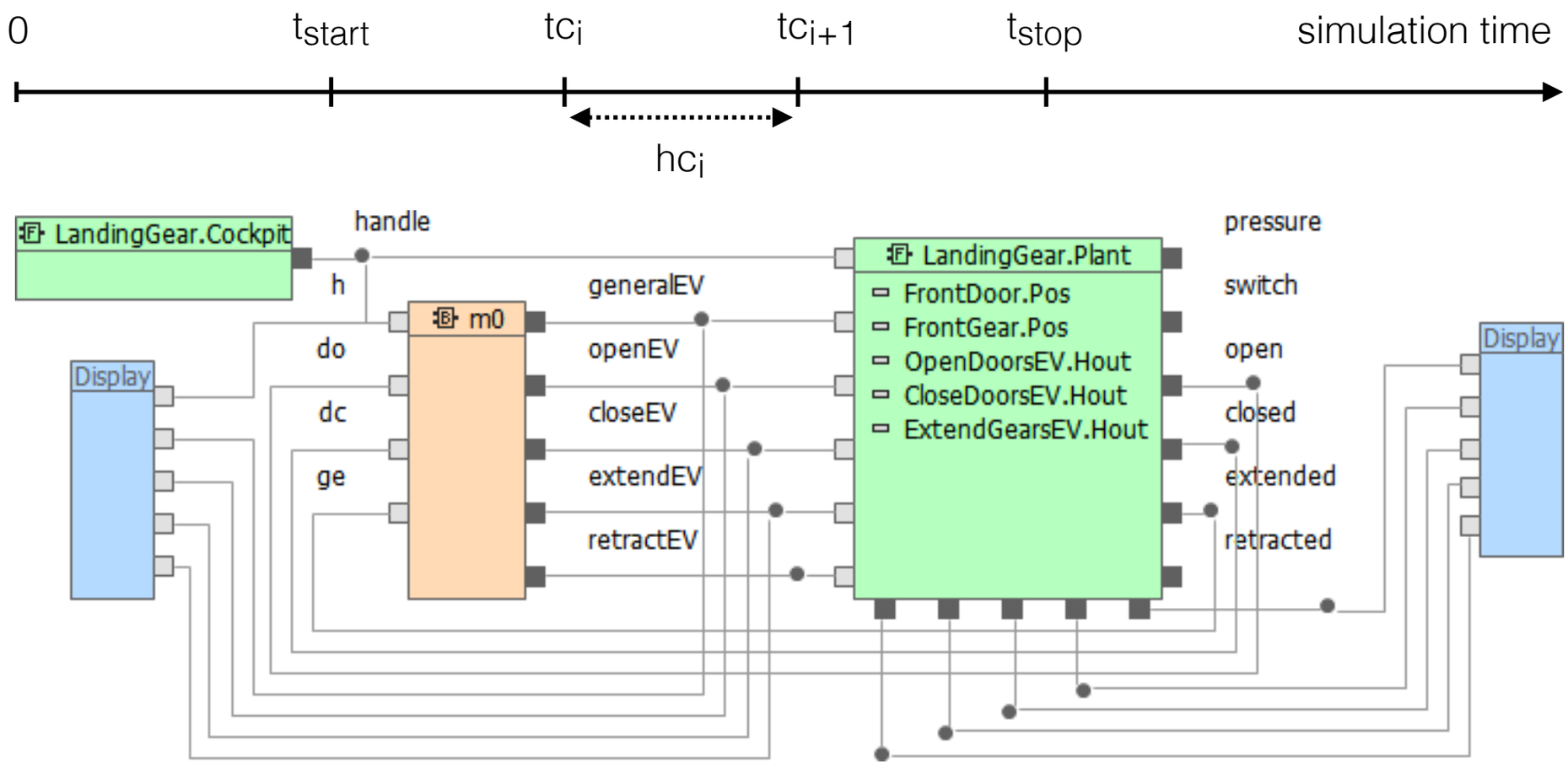

LandingGear.Cockpit
handle
h
do
dc
ge
m0
generalEV
openEV
closeEV
extendEV
retractEV
Display
LandingGear.Plant
FrontDoor.Pos
FrontGear.Pos
OpenDoorsEV.Hout
CloseDoorsEV.Hout
ExtendGearsEV.Hout
switch
open
closed
extended
retracted
Display

R M S

Master orchestrates the simulation of components and performs the data exchange

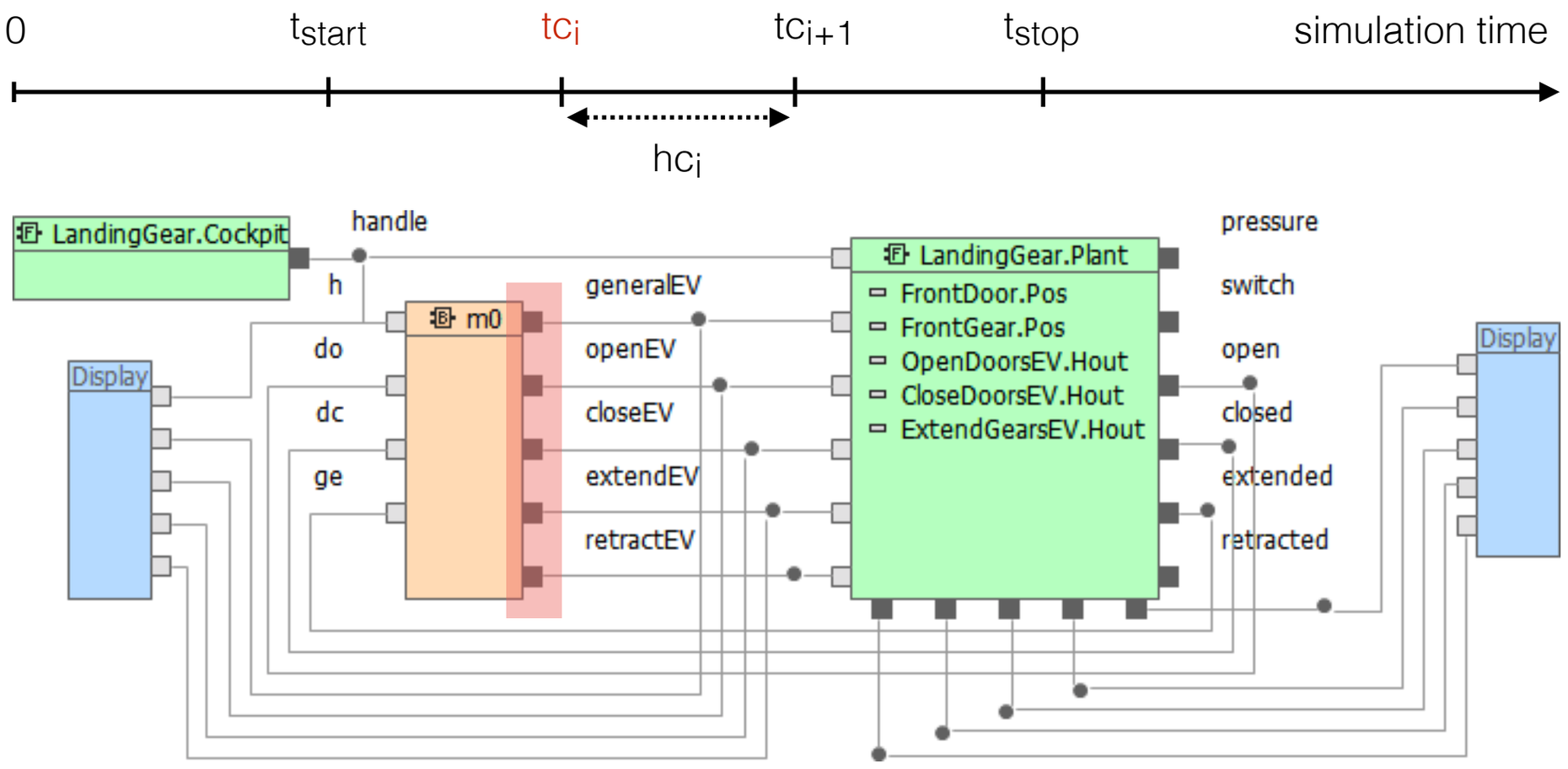


R M S



Simulation is broken down into communication steps $tc_i \rightarrow tc_{i+1}$ of the size $hc_i = tc_{i+1} - tc_i$

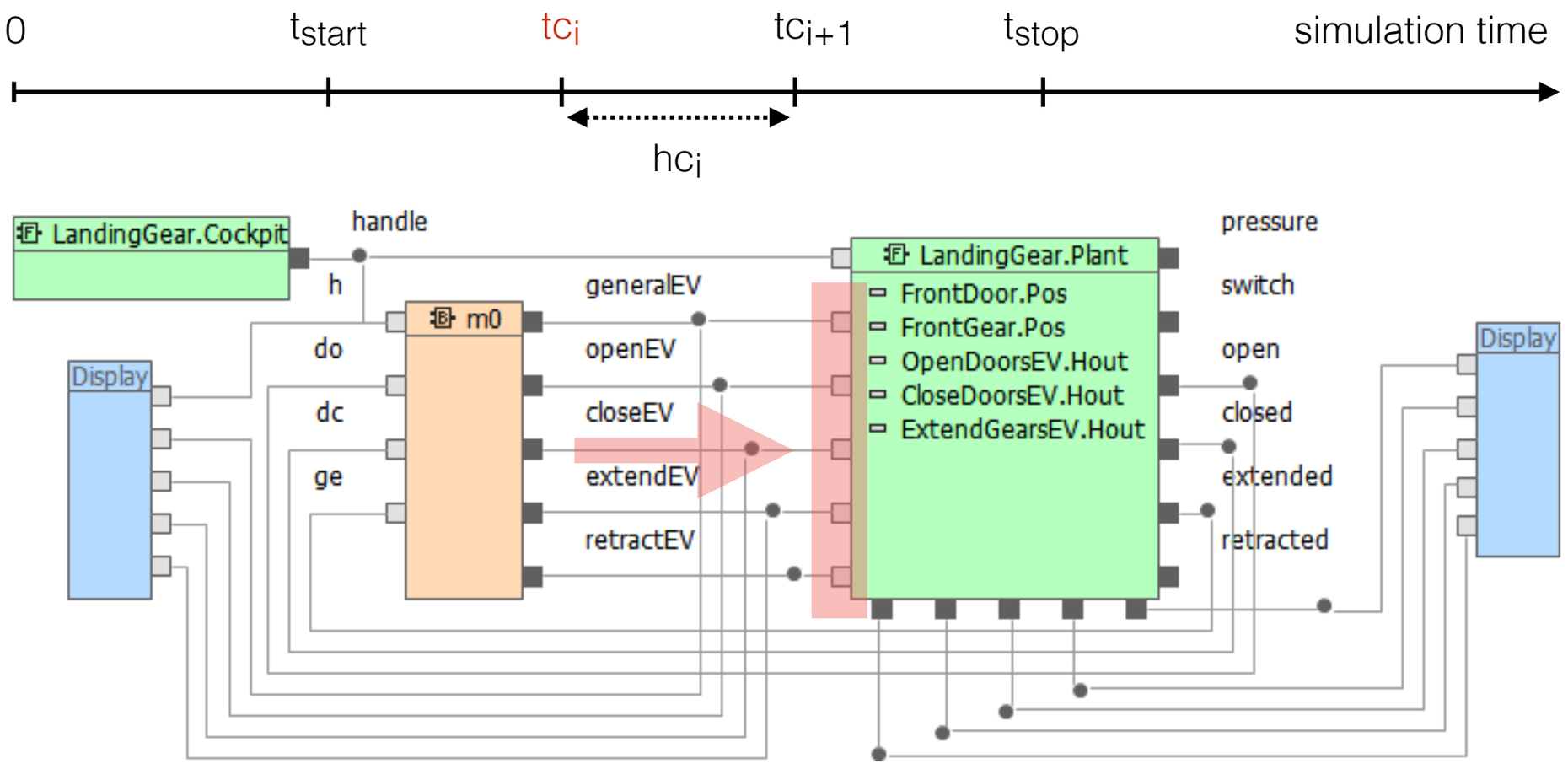
R M S



trace.value(p_n)

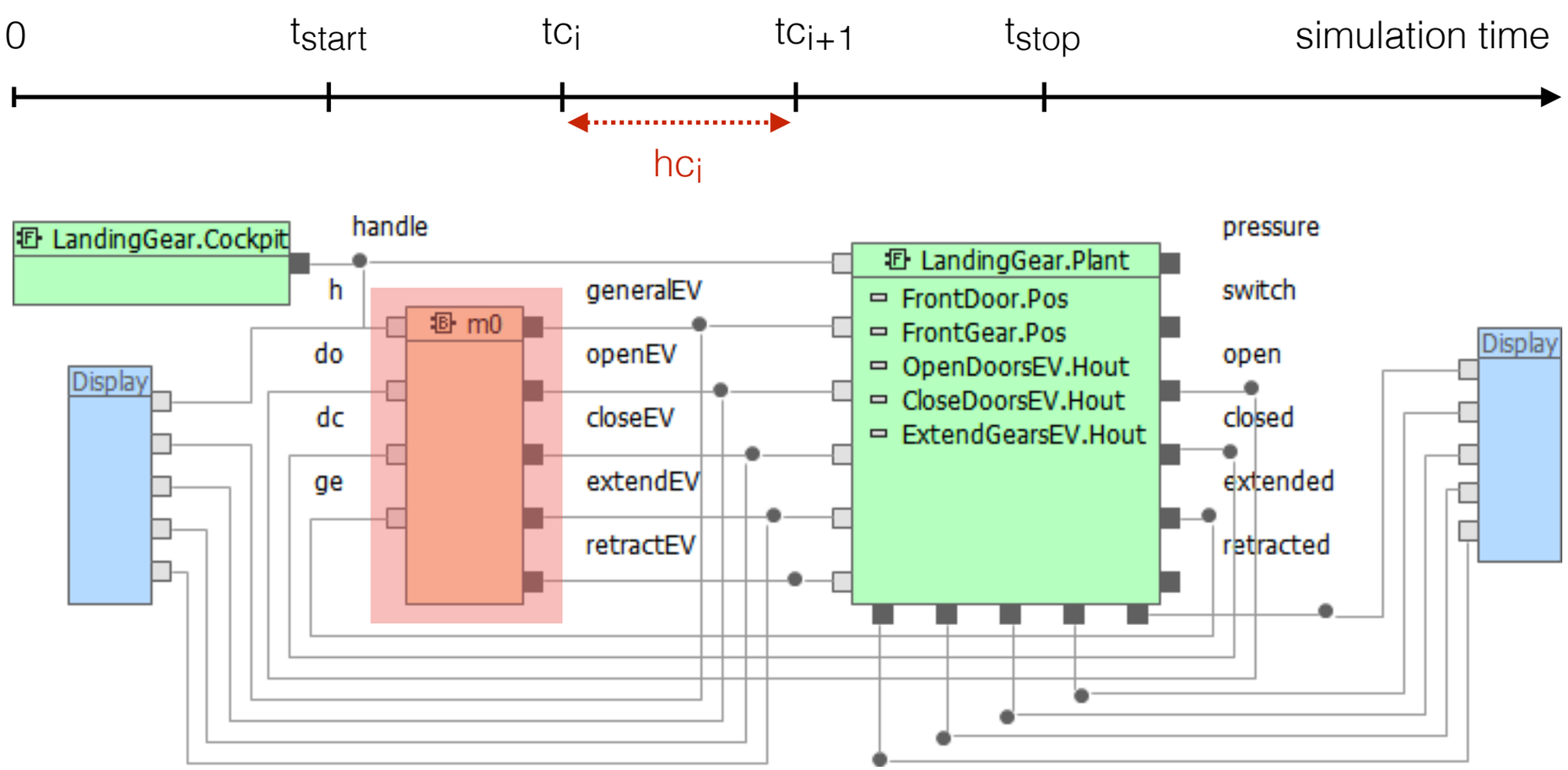
ADVANCE

RMS



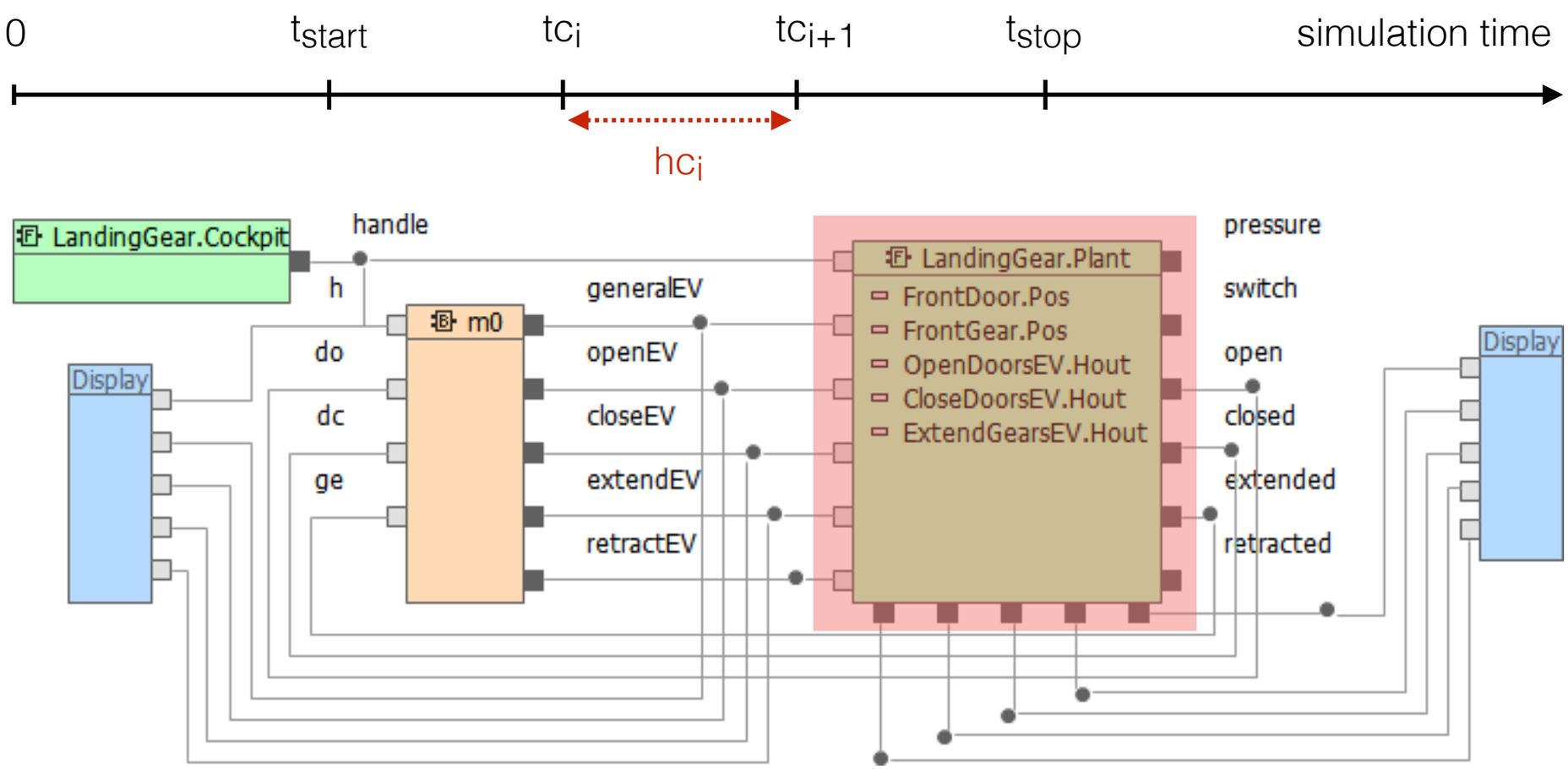
$fmu.set(p_n)$

R M S



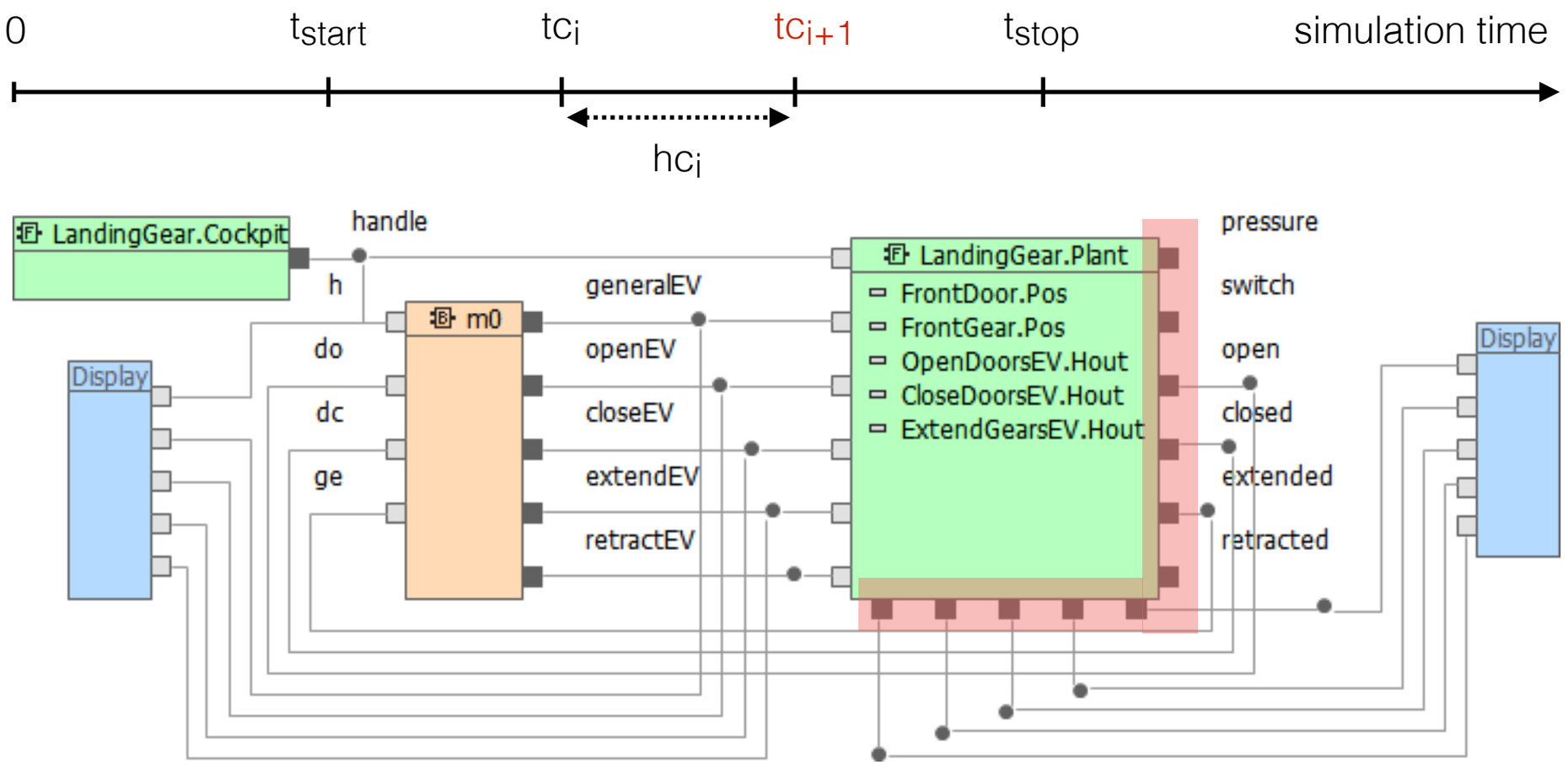
do trace.anyEvent() until enabled(waitEvent)

R M S



$$fmu.doStep(tc_i, hc_i)$$

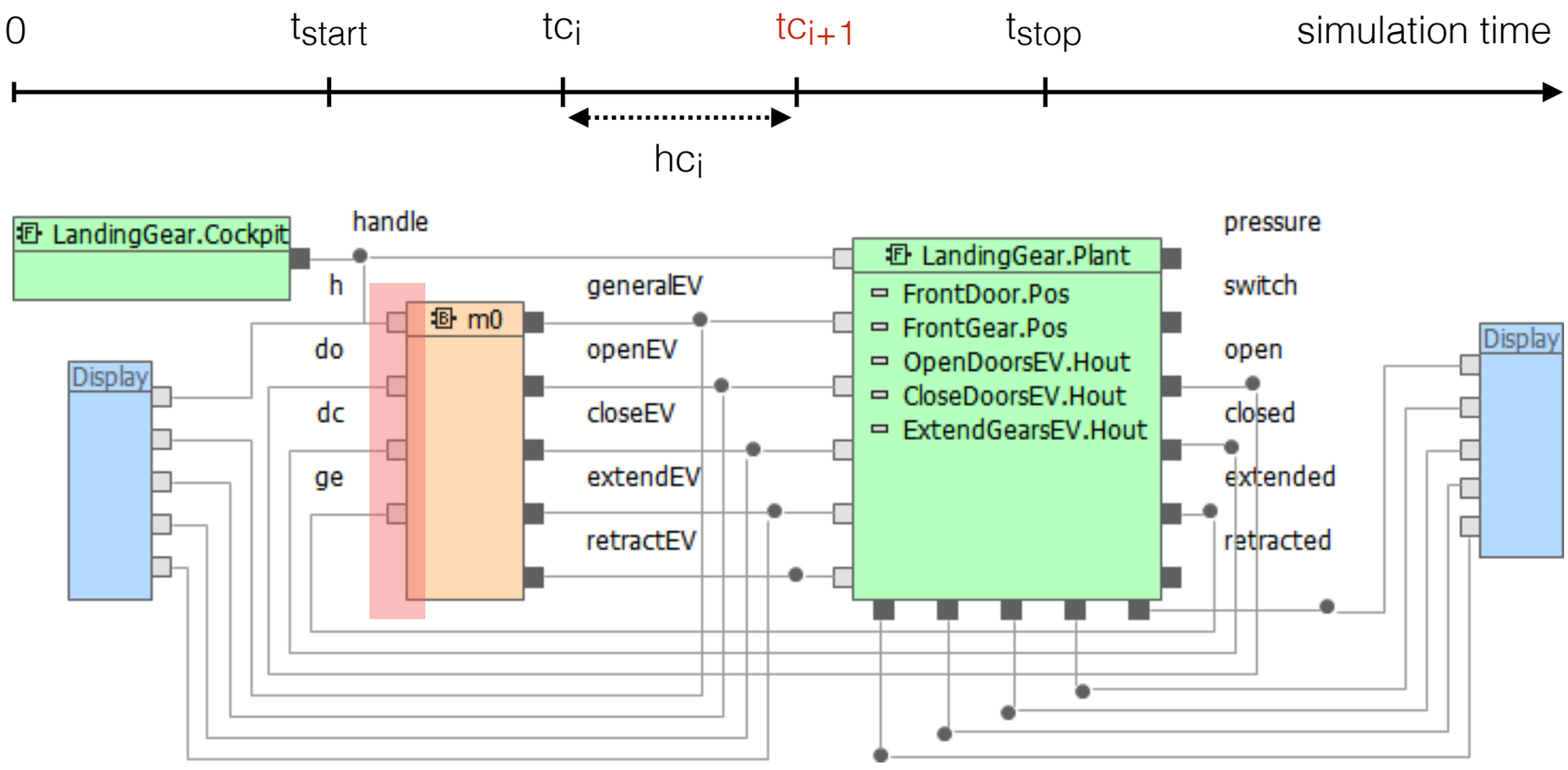
R M S



$fmu.get(p_n)$

ADVANCE

R M S

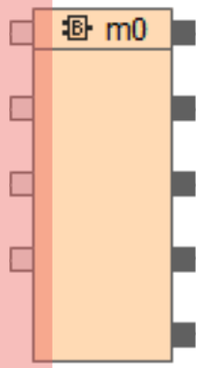


trace.readInput($p_0,\ldots,p_n$)

R M S

UNIVERSITY OF Southampton

event readInput
any h, ge, do, dc
where
  h ∈ BOOL
  ge ∈ BOOL
  do ∈ BOOL
  dc ∈ BOOL
  state = io
then …
end



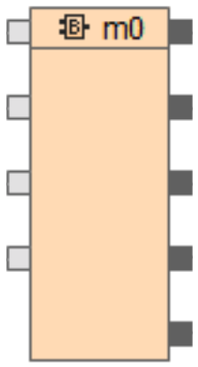
ADVANCE

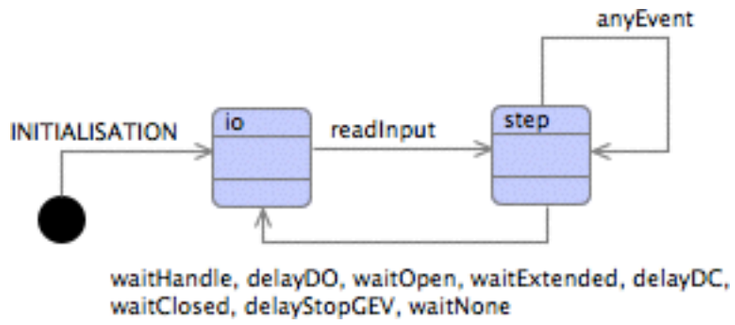
R M S

UNIVERSITY OF Southampton

Control flow

event readInput
any h, ge, do, dc
where
  h ∈ BOOL
  ge ∈ BOOL
  do ∈ BOOL
  dc ∈ BOOL
  state = io
then …
end



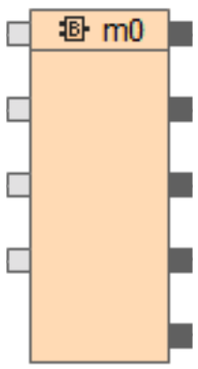


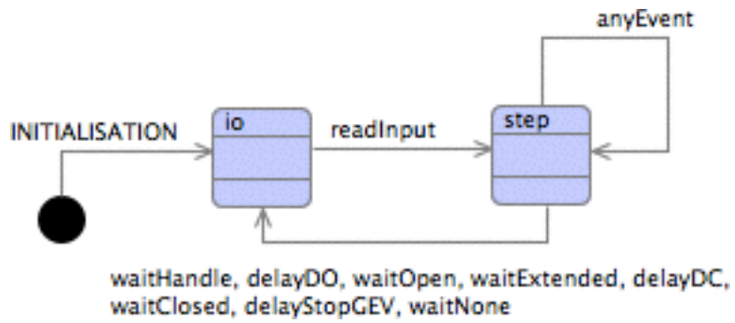
ADVANCE

RMS

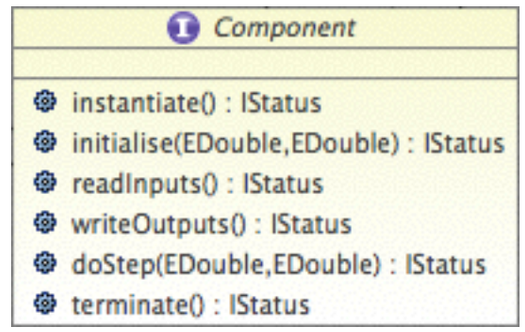
UNIVERSITY OF Southampton

event readInput
any h, ge, do, dc
where
  h ∈ BOOL
  ge ∈ BOOL
  do ∈ BOOL
  dc ∈ BOOL
  state = io
then …
end





API



ADVANCE

Summary

- Generic simulation master
- Support of the FMI CS v1.0 standard components
- Flexible mapping of Event-B machines to co-simulation components, including the refinement of *read* and *wait* events
- Either timed or non-timed Event-B components
- Graphical simulation environment in Rodin, with signal plotting, simulation trace recoding/playback, deadlock and invariant checking

ToDo

- Simulation performance improvements
- Support of the FMI CS v2.0 standard (FMI Library for RC2 is already available)
- Smarter master algorithm, i.e. variable step size and error control, use of the FMI flags *canHandleVariableCommunicationStepSize* and *canRejectSteps*, handling of events and algebraic loops (FMI 2.0)
- Other simulation semantics (MA), features?

Thank you!