Hardware-in-the-loop Simulation Framework

Marco Aurelio Antonio Sanvido

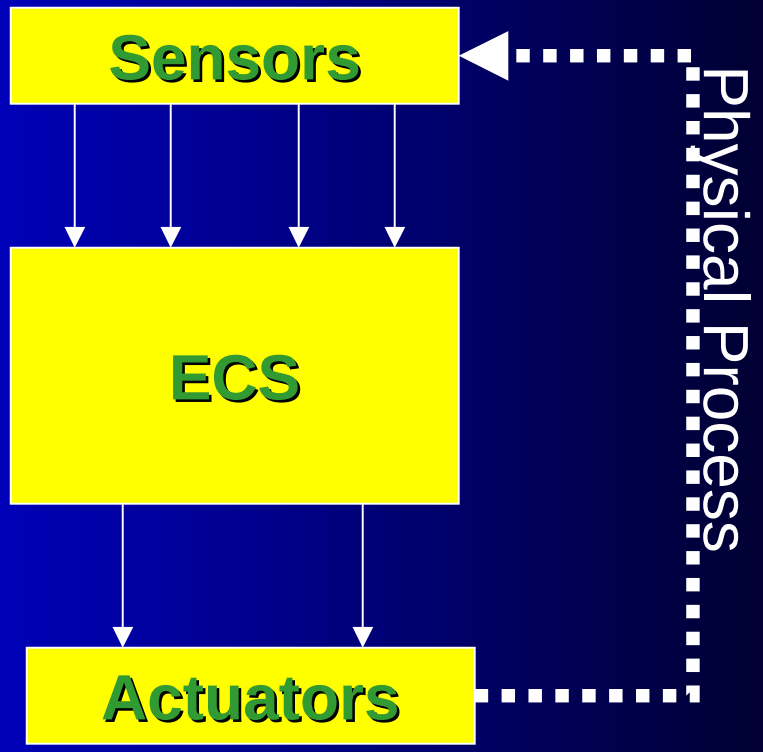
Outline

- Motivation: ECS Testing with HIL
  - ECS Properties
  - HIL Simulation for ECS
- The HIL Simulation Framework
  - Concepts & Structure
- Numerical Simulation
- Hardware Interface
- Real-time Scheduler
- Fault Specification Language (Fausel)
  - Testing with Fausel
  - A small Example
- Conclusion & Demo

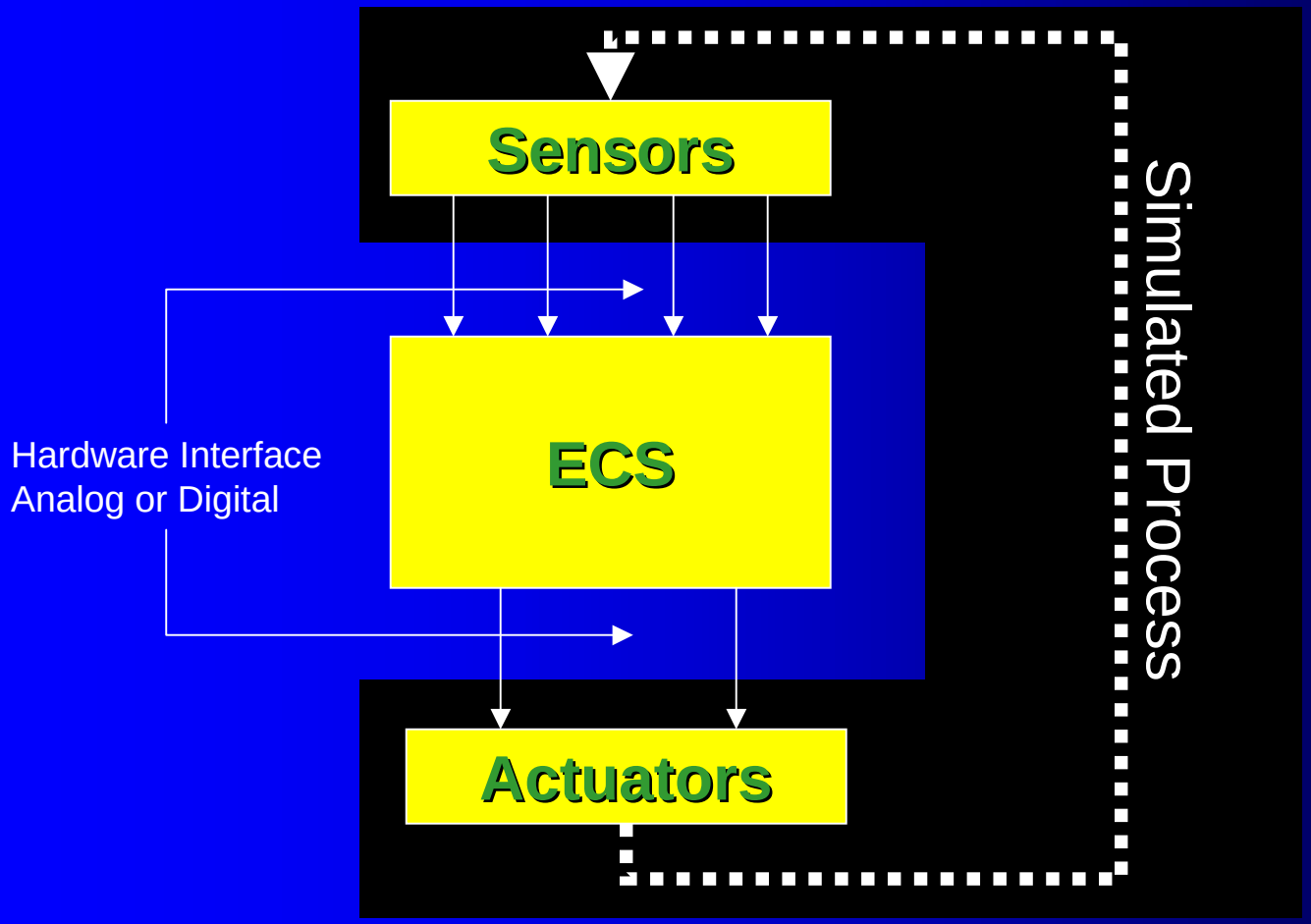
Embedded Control System

Unique Properties of ECS Software:

- Timeliness
- Concurrency
- Liveness
- Reactivity
- Safety



Hardware-in-the-loop Simulation

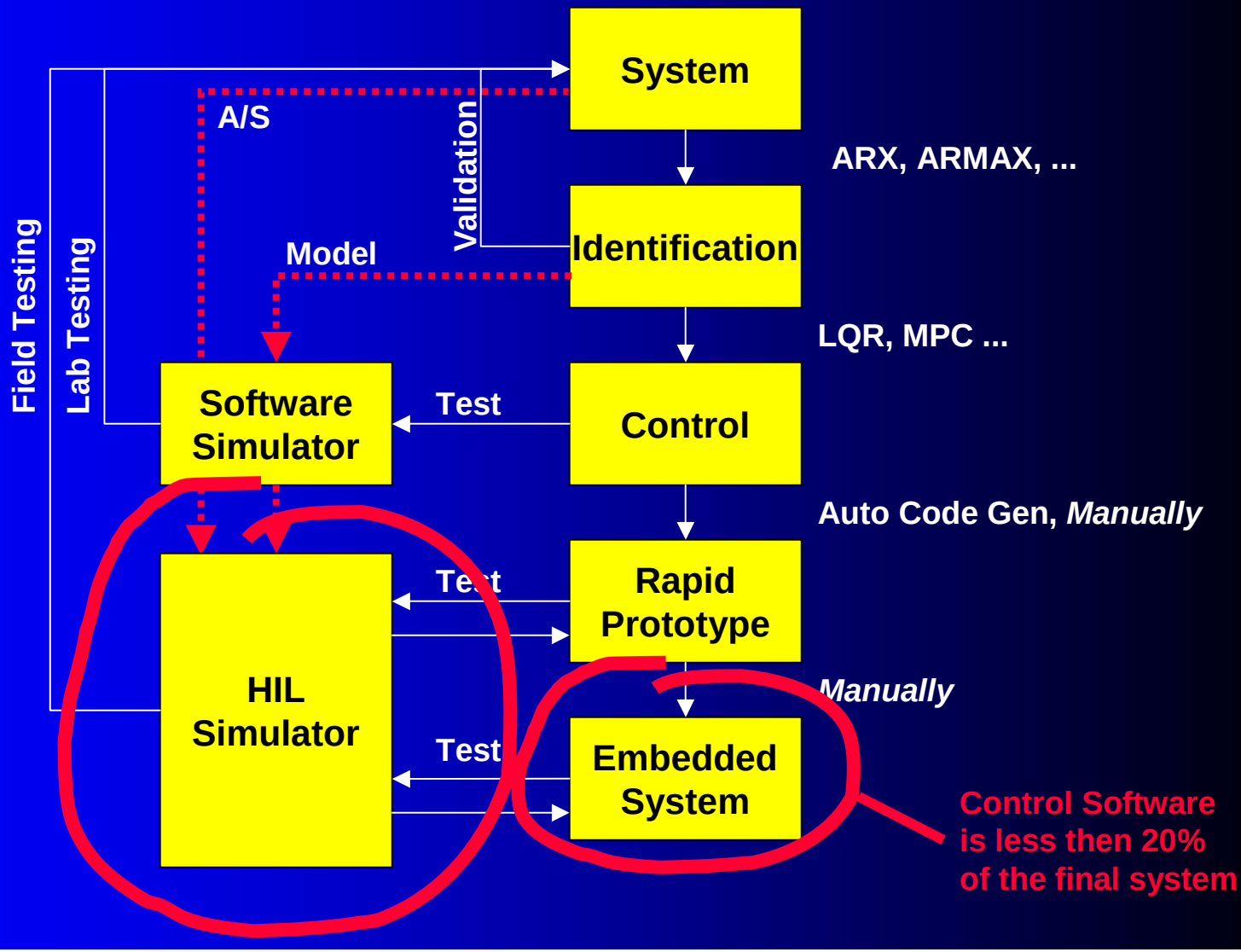


1. The simulated process can be operated with the *real* control hardware

2. The simulated process replaces either fully or partially the controlled process consisting of actuators, physical process and sensors.

(Isermann et al. ‘99)

ECS Development



Software for ECS

E. Lee, *Embedded Software*, vol. 56, Advances in Computer, Academic Press, London 2002

State-of-the Art :

- Software for ECS is implemented just as software on small computers (except for RTOS)
- Engineers who write ECS are rarely computer scientists

Software is:
Unreliable
Huge, i.e. over-dimensioned
Expensive

Software is:
problem oriented,
not solution oriented

Hardware-in-the-loop Simulation

Why do we need HIL Simulation?

- Allows testing dangerous situations
- Ensure a *correct* implementation, reducing the gap between design and implementation
- Allows repeating the same test again and again deterministically

HIL Simulation Framework

Why do we need a HIL Simulation Framework?

- Often the HILS are built from scratch
- Often the HILS are more expensive than the real process
- Implementation with standard simulation software, which is not developed for HIL:
    - I/O interfacing can be done only with supported cards
    - No support for automatic testing

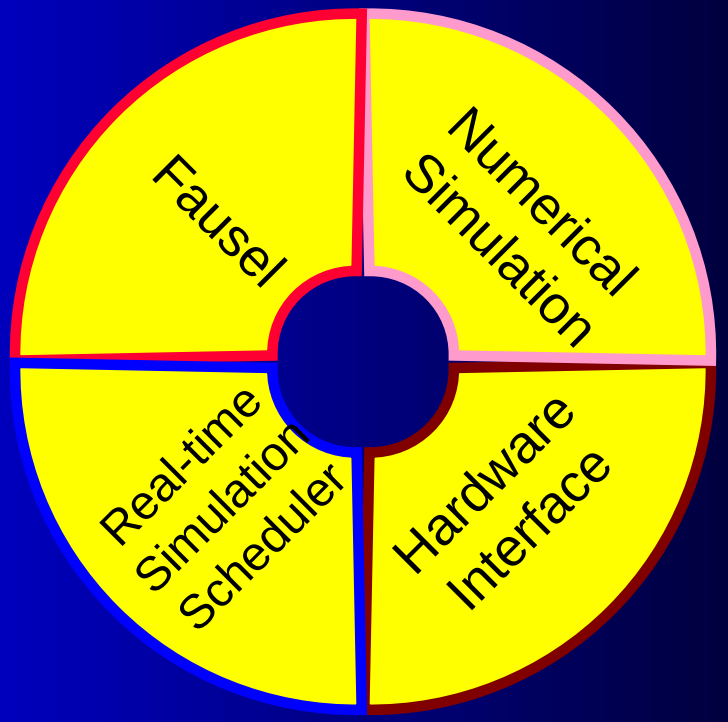
HIL Simulation Framework

Which are the advantages of a Framework:

- Simplify the implementation of dedicated HILS
  - Reuse of previously implemented components
  - Simplify the generation of sensor signals
  - Simplify the acquisition of actuator signals
- Automatic fault simulation
- Automatic ECS response *verification*
- Real-time-simulation (on Native Oberon)

The HIL Framework

- Simple implementation of the mathematical formulas of the simulated process.
- Generalized signal acquisition and generation.
- Real-time simulation scheduler.
- Generic fault generation/verification engine (Fausel).

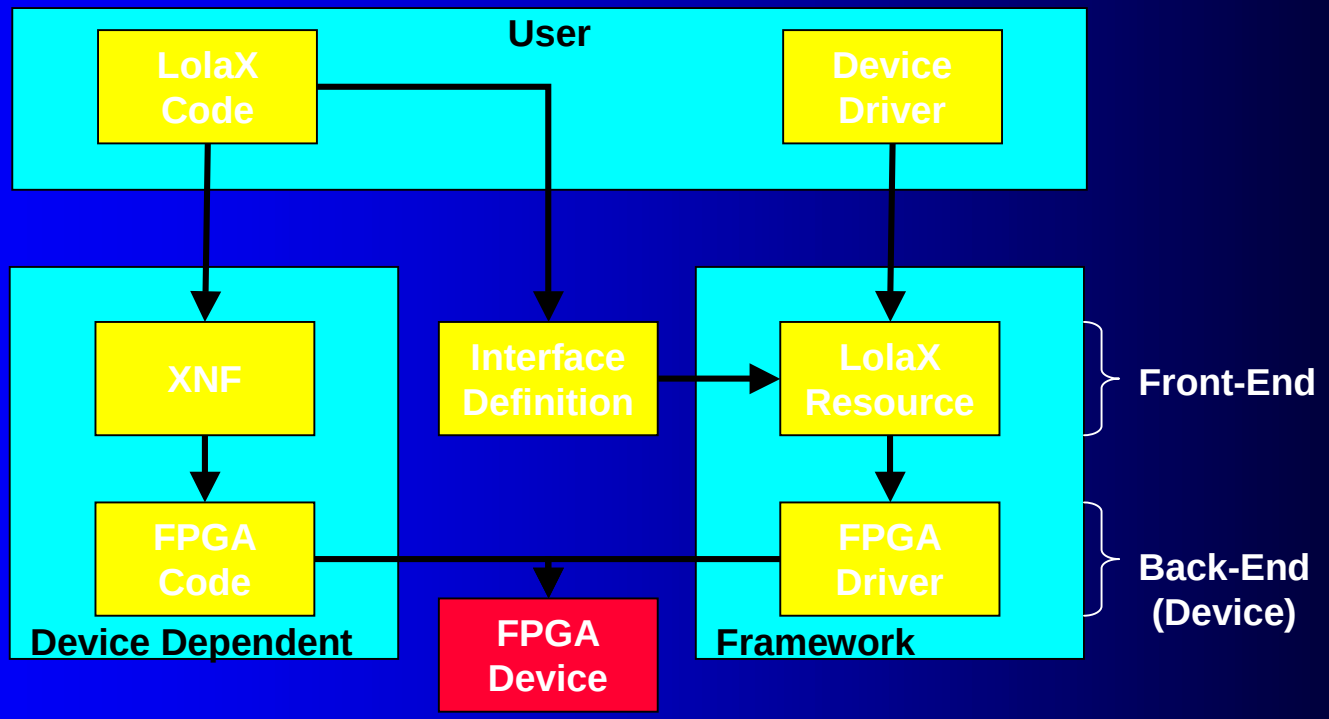


Numerical Simulation

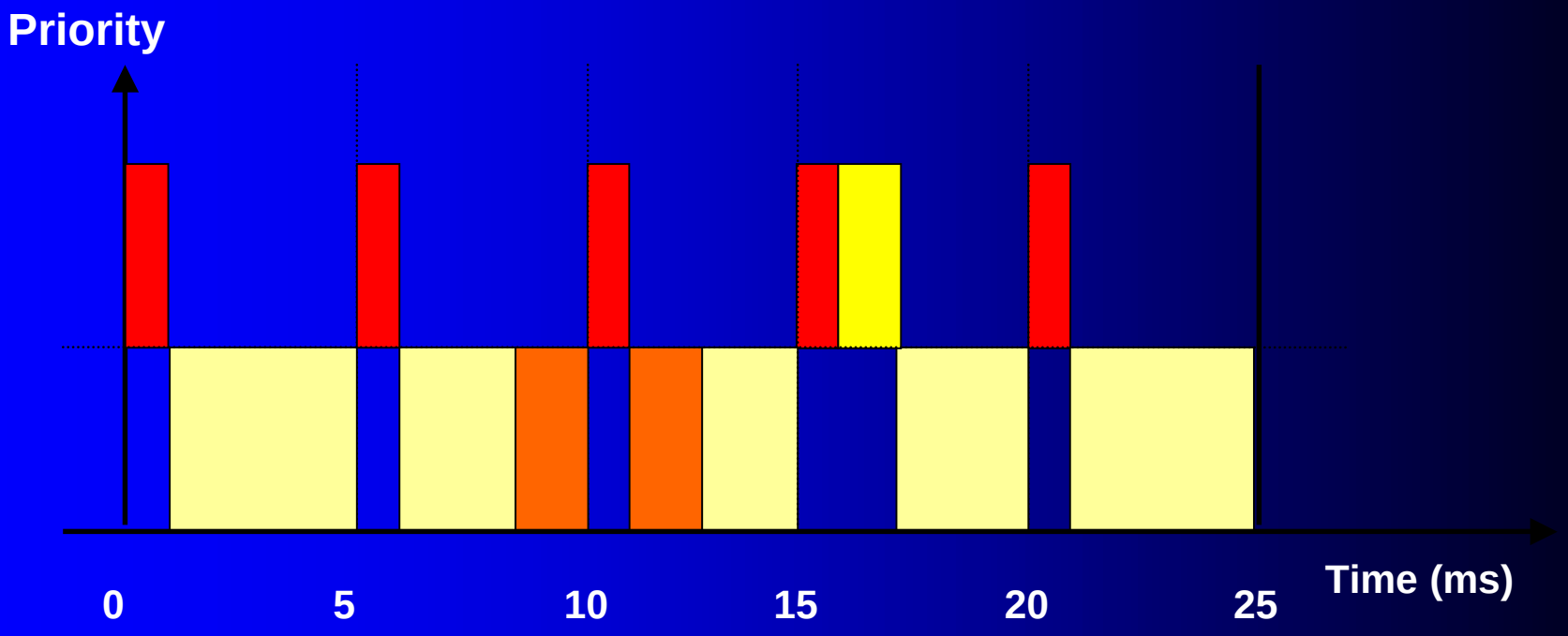
- Simple to implement new models
- Efficient (i.e. real-time constraints)
- Extensible ODE solvers
- Easy to connect dynamically sensors and actuators

Hardware Interface, i.e. LolaX

- Lola Compiler extended with interface definition
- Simple to write device drivers for customized signal generation/acquisition



Real-time Simulation Scheduler



200Hz

Simulate Model (Synchronous I/O)

50Hz

Low Priority A/S

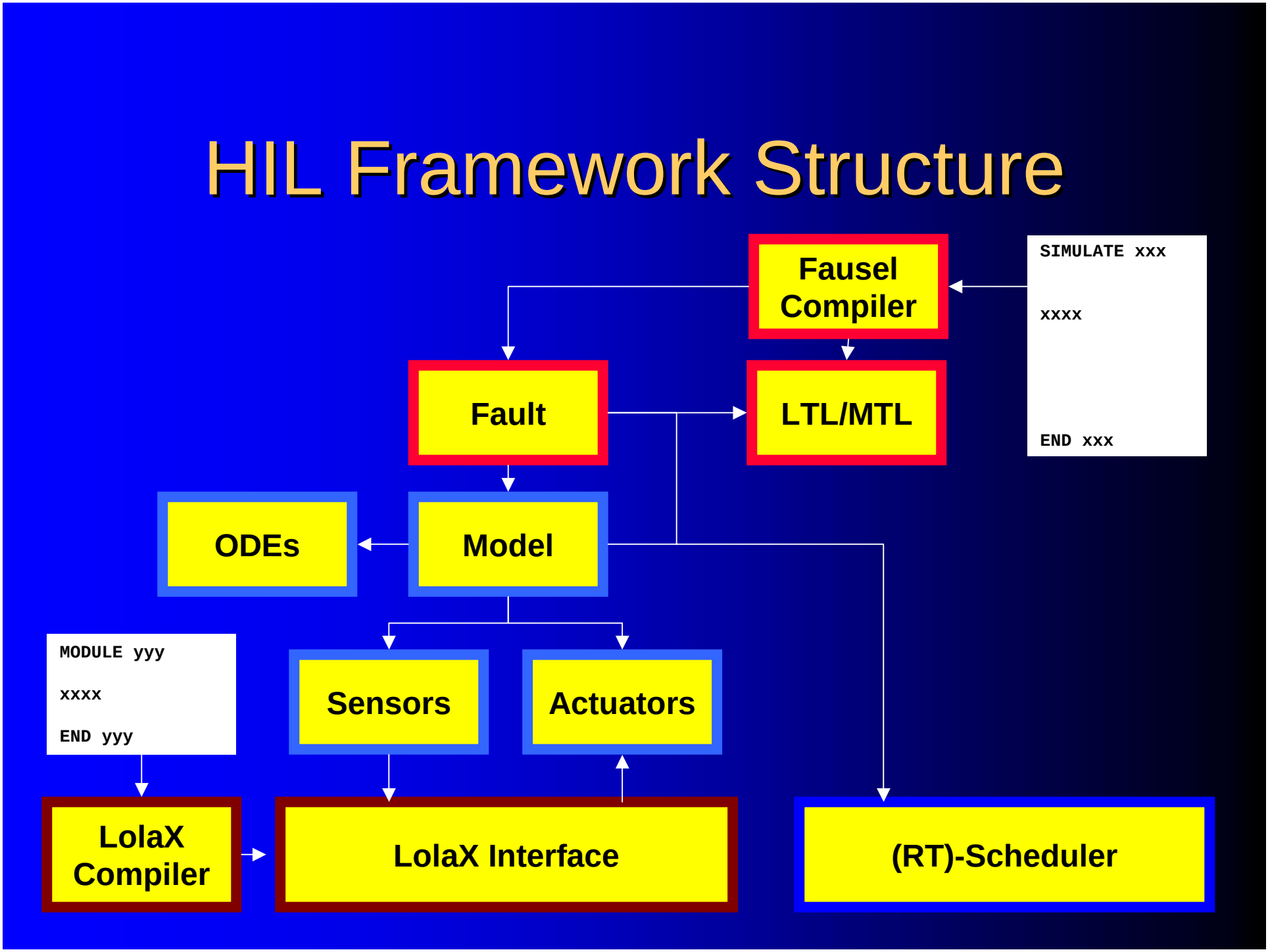
Background

Normal Oberon Activity + Oberon Tasks

Fausel Checker

Based on System7 and on HelyOS

HIL Framework Structure



Fausel (FAUlt SpEcification Language)

The HIL generates fault sequences:

- Deterministic, non-deterministic, configurable
- Logically interconnected,
  e.g. Fault ***B*** can start only if Fault ***A*** has happened once

The HIL tests the ECS response specification:

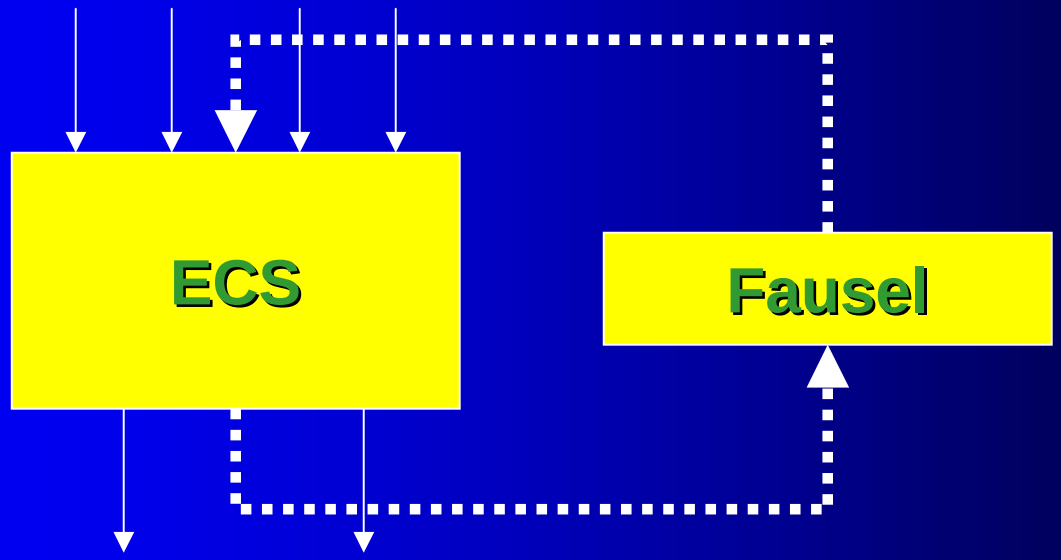
- Tests for temporal behavior (using LTL)
- Tested during simulation time

➡ regression tests, documentation
on the real system are possible

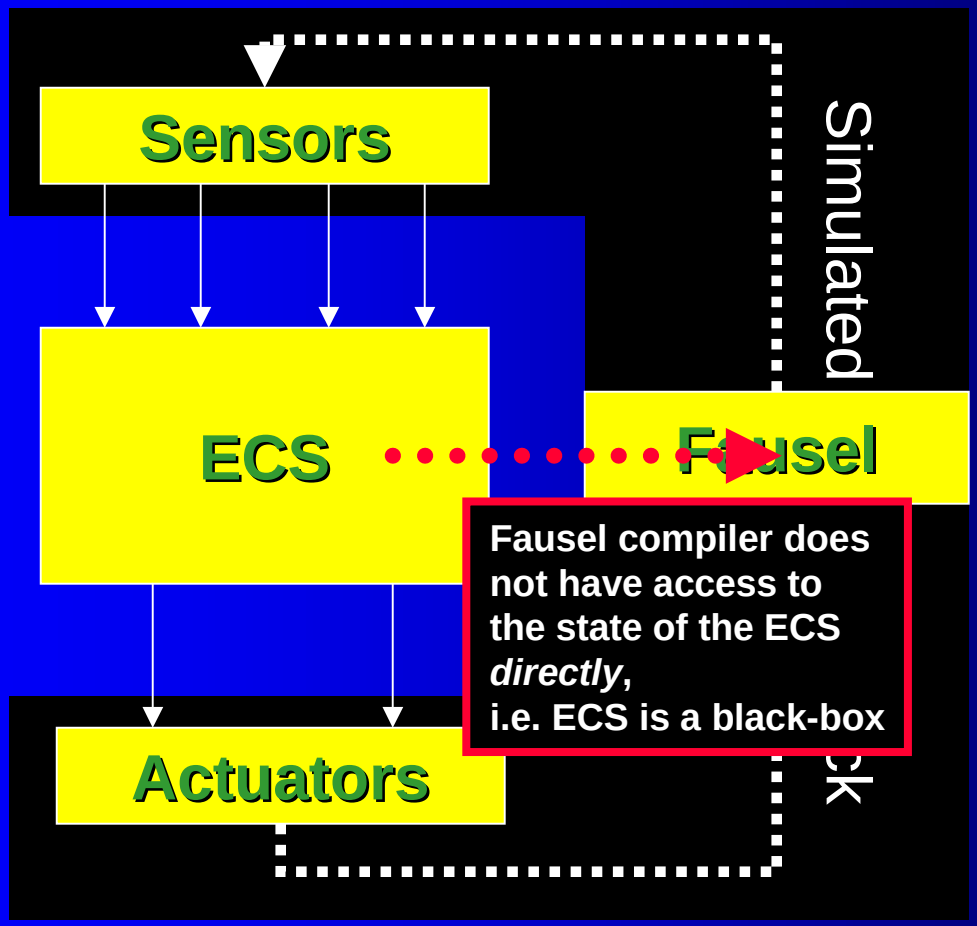
ECS Testing with Fausel

Fausel goals:

- ECS correctness ≠ checking the final answer
- ECS correctness = stream of partial answers



ECS Testing Integrated into the HIL




```
SIMULATE Helicopter;

  FAULT Wind0 IS Wind(w:=10.0);
  START (pitch > 0.1);
  STOP firsttime + 10.0;
  PERIOD 0.5;
  SPEC
    F[5,10](yaw = startyaw);
  END Wind0;

END Helicopter.
```

Test Classes

What are we able to test?

1. We are only able to test expected faults!
2. We are only able to generate errors on the Model, Sensor and Actuators
3. We are NOT able to check ECS states (only possible via online-debug or extrapolation)
4. The *Temporal Logic* is *reasonably* expressive, but not complete. E.g. "*p* must be *TRUE* X times"
5. Test sequences are finite, but a specification could be infinite, e.g. G ***p*** => use constrained temporal logic (MTL)

Conclusions and Outlook

Framework is easy to apply if you have:

1. OOP knowledge (Oberon)
2. Hardware-design knowledge (Full HIL)
3. First design is difficult

Framework has been applied to:

1. Barrage Simulation
2. OLGA Helicopter Main Rotor Controller
3. OLGA Helicopter Hovering Controller

Framework Extensions:

1. Java implementation is available (Demo)
2. Matlab interface (Demo ??)