Simulation in the LAAS Architecture

Sylvain Joyeux, Alexandre Lampe
Rachid Alami, Simon Lacroix
firstname.lastname@laas.fr

LAAS/CNRS

18 April 2005

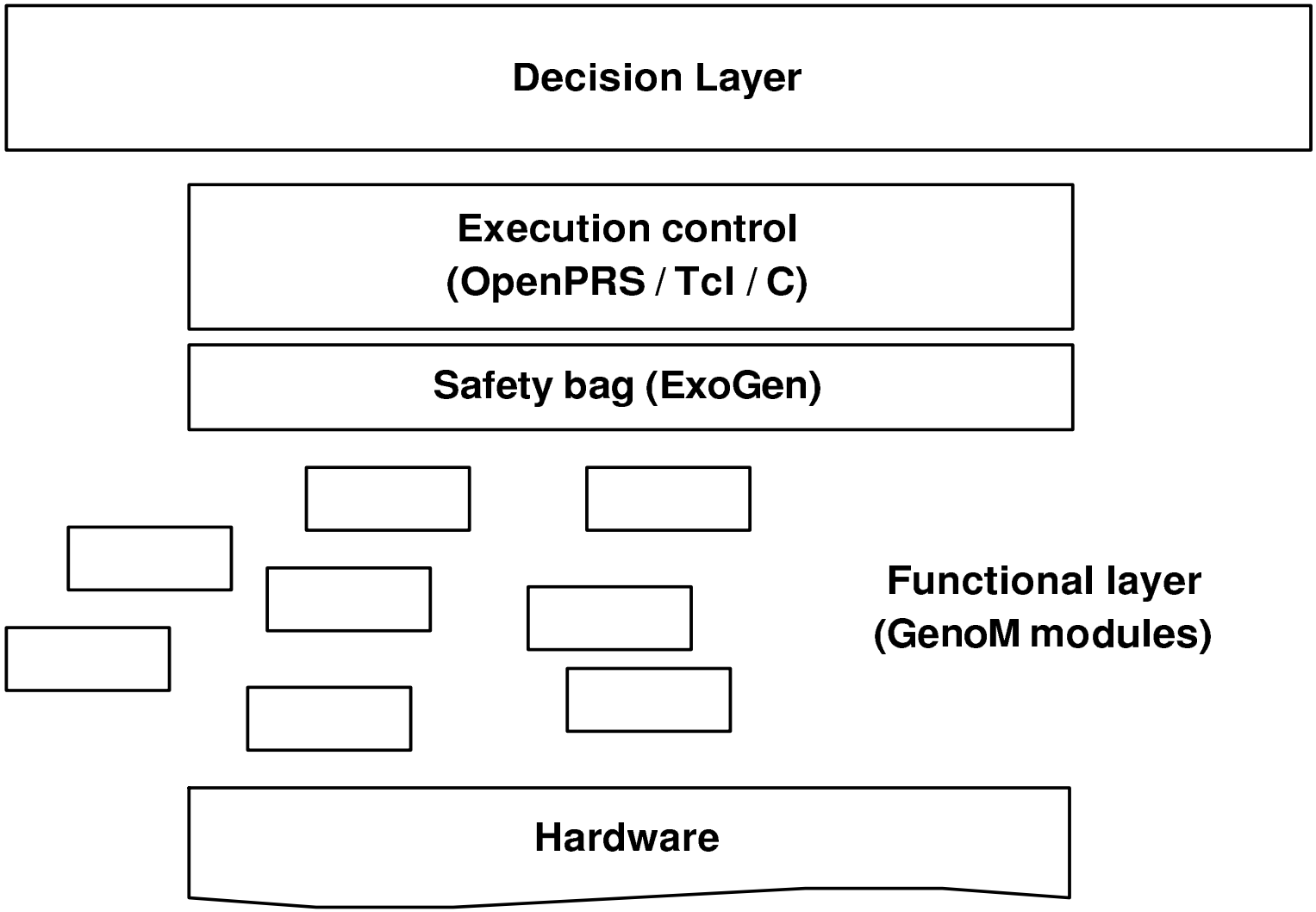
Simulation, what for ?

- extensive testing
  - access to situations unavailable in experiments
  - evaluating robustness
  - stress-test
- prototyping
  - high level layers (reasoning, multi-robot interaction)
  - testing algorithms not ready for use on real robots

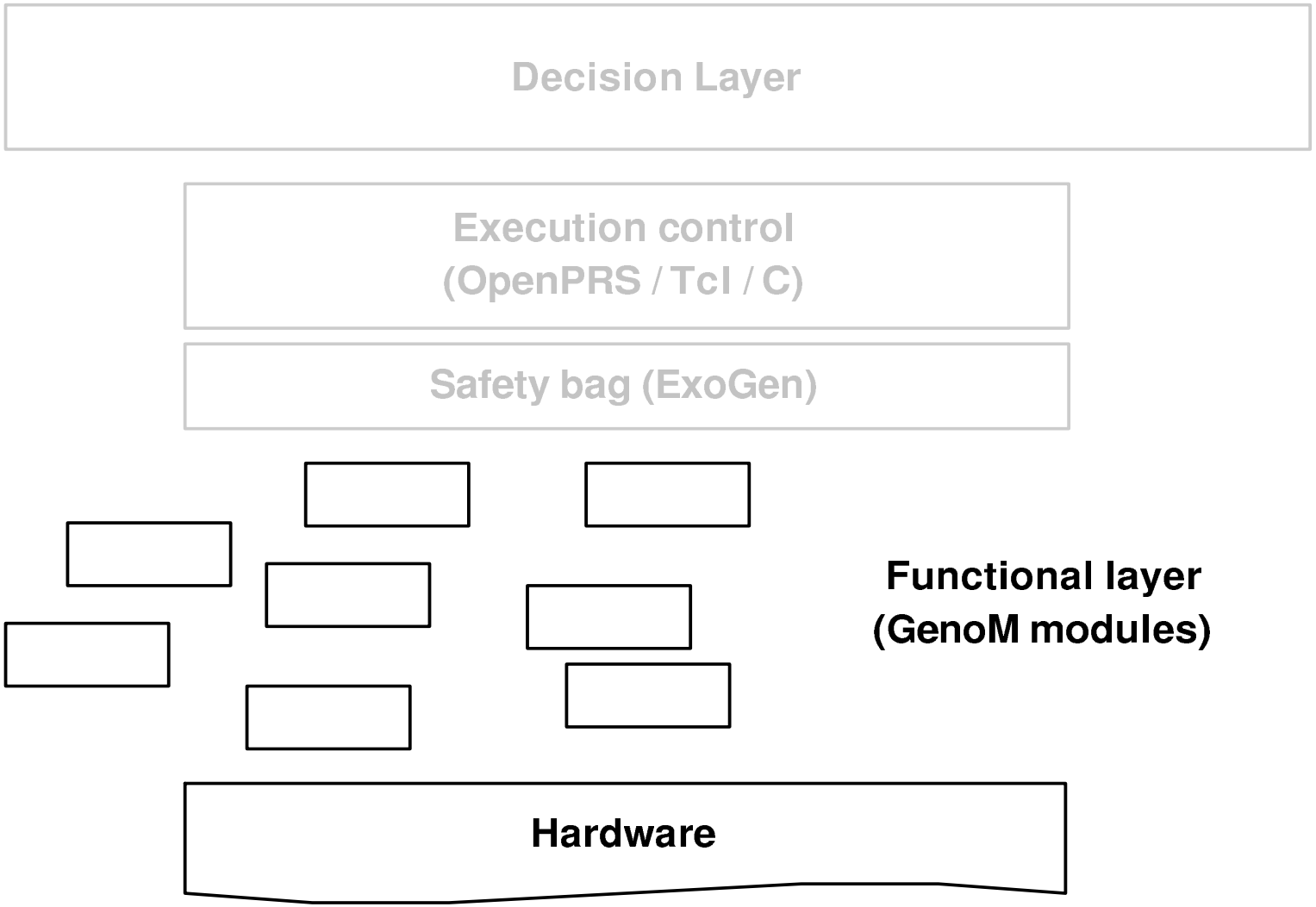
Design goals

- for prototyping as well as testing
- distributed multi-robot simulations
- integration of the software currently running on the robots
  - compatibility with the existing tools
  - simulating both real-time (RT) and non-RT software
  - no modification of the software needed

Current implementation of the LAAS Architecture



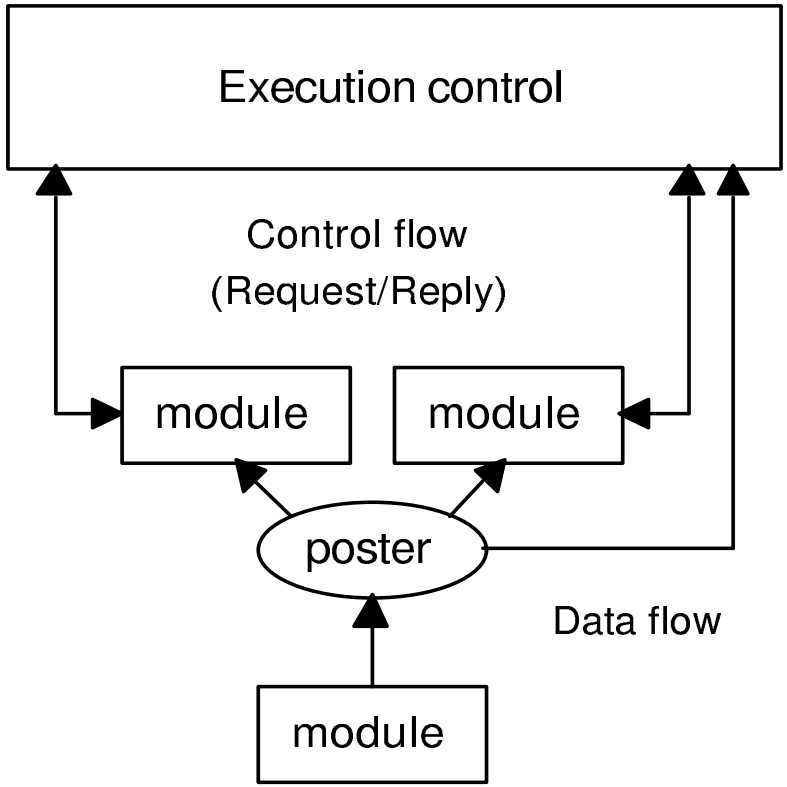
Components managed in simulation



Genom-based functional layers

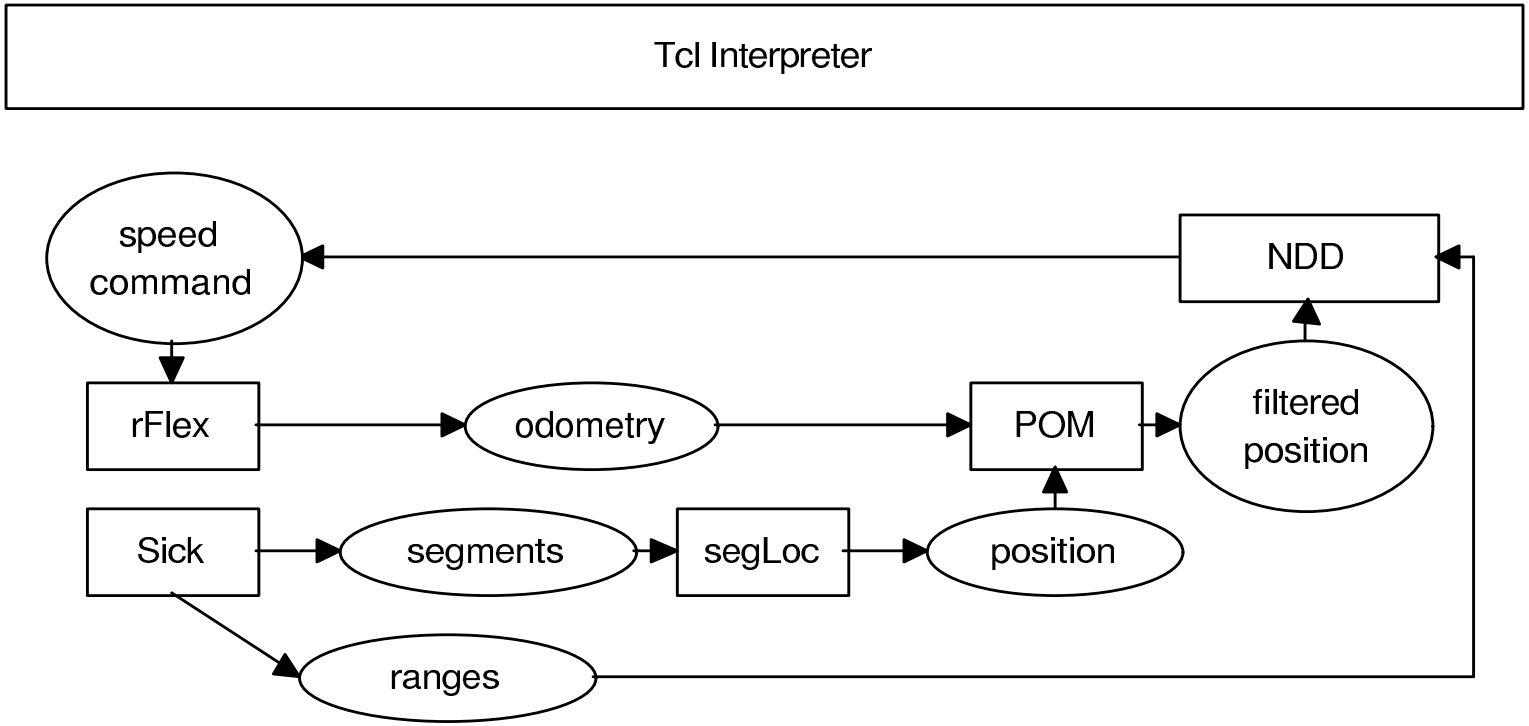
The GenoM layer

- modular
- asynchronous control of module activity
- inter-modules and module/control data flow
- real-time specification



Example

Reactive navigation



Functional layer behaviour

Execution environments differ

.

It is not required that

- the physics simulation is fast enough
- the host system is a RT OS
- the processing power is the same
- the communication use the same media (WiFi vs. eth)
- algorithms in develoment may have less than ideal performances

The layer behaviour can change when the execution environment changes

Functional layer behaviour

Because of these differences, we should provide

- a simulated real-time behaviour
- specification of execution duration
  - for real-time code
  - for prototyped algorithms
- simulated hardware latencies
- simulated communication latencies

We want the execution results to be the same as when the time specifications are met

Time control

A time control layer guarantees the functional layer behaviour

- relies on the discrete event nature of software code
- catches calls to low-level synchronization functions
- allows to specify:
  - arbitrary durations to sections of the client code
  - arbitrary latencies to hardware access and communication
- no module recompilation or modification needed

Access to the simulated world

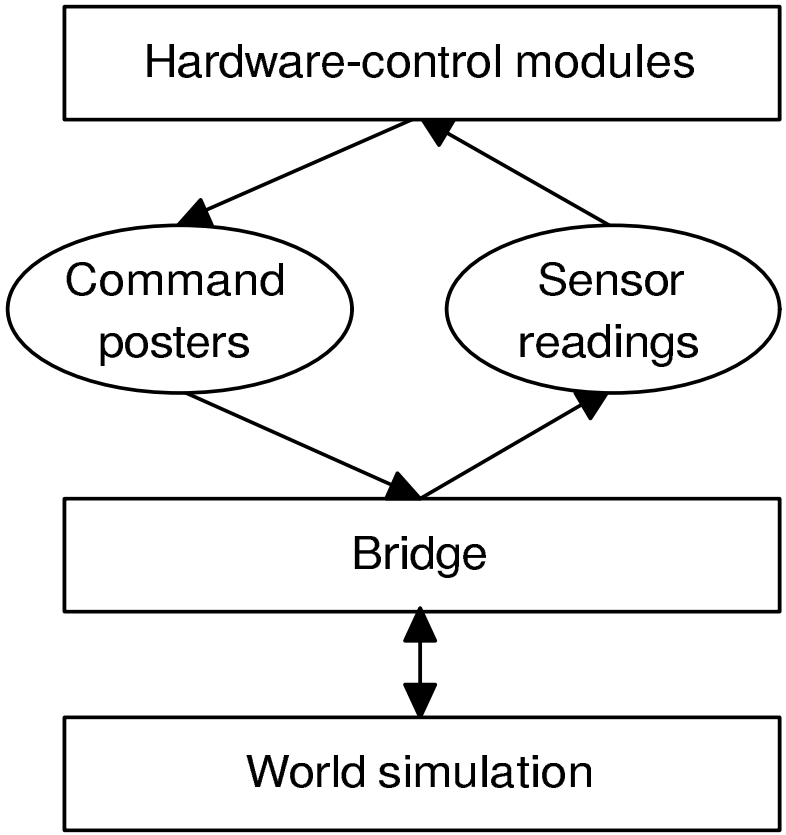
Hardware access

Generic communication between the modules and the world simulator

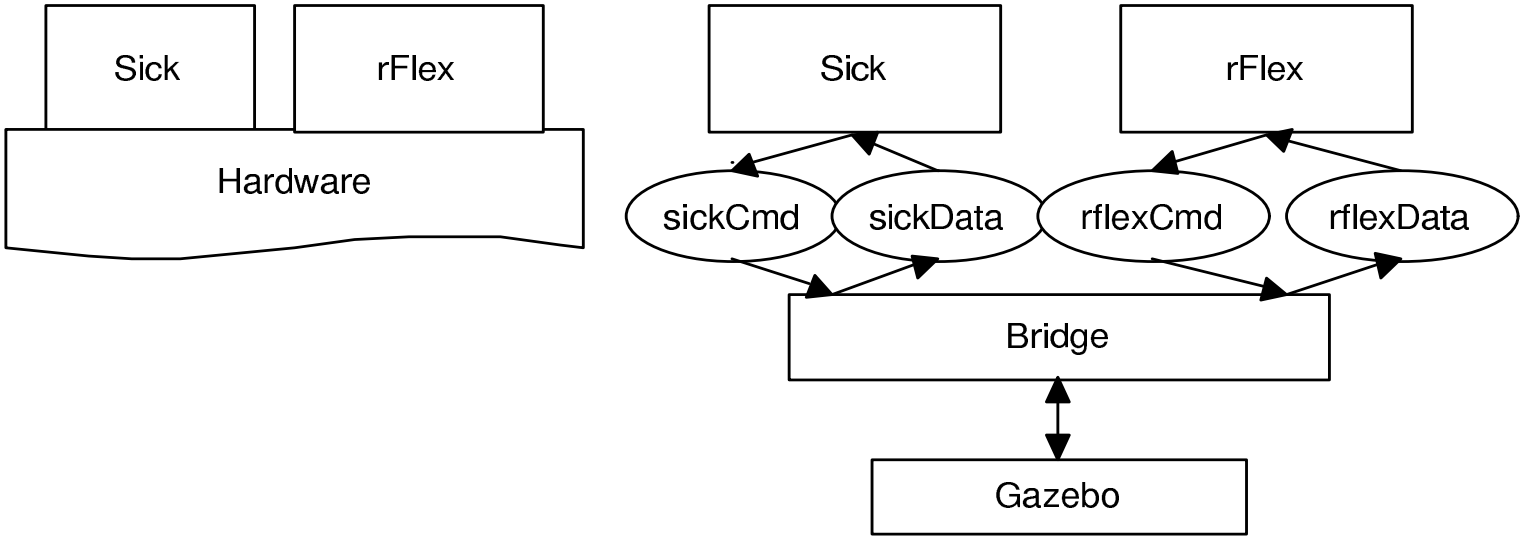
- adds hardware communication latencies
- adds sensor error models

World simulation

- currently using Gazebo
- fixed-step simulation
- provides sensor simulation



Integration: hardware access



Example simulation

Integration of upper layers

- execution control:
  - based on OpenPRS or Tcl
  - event-based procedural reasoning
  - time-related events allowed (deadlines)
- time control guarantees the sequence of events from the functional layer
- time-based events handling:
  - may be based on system timers
  - may be based on event loops

Currently

We only make the calls to time-reading functions return the simulation time

Distributed, multi-robot

Multi-robot interactions

- implicit communication: via the physical world
- no explicit communication (for now)

Implemented solution

- one Gazebo server for each simulation host
- at each Gazebo simulation step, synchronize all servers
- we use HLA for data distribution and time services

Future developments

- provide communication simulation in multi-robot setups
- heterogeneous simulations via HLA
- scripted events during simulations
- replaying logged values in order to test isolated modules