Labs in Julia

Mattias Fält

The goal (Educational)

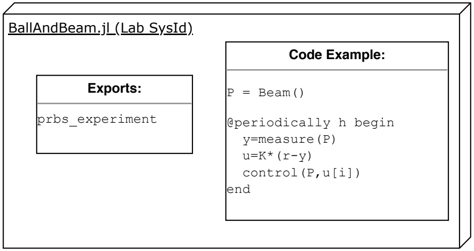
- Allow students to actually write a controller
- Simple and understandable interface for a student
- Not locked to the LabComputers! It should be possible to run the same controller and code:
  - In simulation at home
  - With the hardware at the lab

The goal (Developer perspective)

- Easy and understandable to implement code to run a lab
- Simple to define a new process
- Interface that works with different hardware interfaces:
  - Comedi.c
  - BeagleBone
  - Moberg
- Maintenence

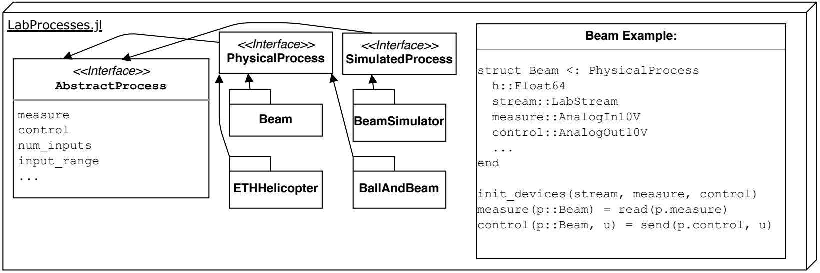
Interface for labs

- BallAndBeam (SysId lab)
  `gitlab.control.lth.se/processes/BallAndBeam.jl`
- LabProcesses (Implementation of processes)
  `gitlab.control.lth.se/processes/LabProcesses.jl`
- LabConnections (Communication protocol, branch julia1)
  `gitlab.control.lth.se/labdev/LabConnections.jl`
- Hardware Implementations



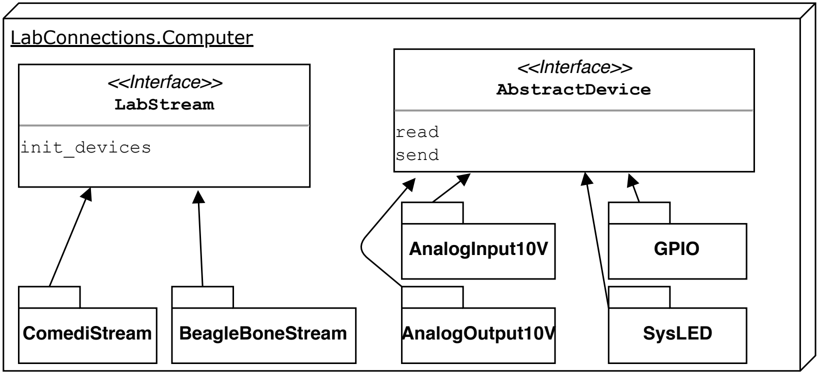
```
P = Beam()

@periodically h begin
  y=measure(P)
  u=K*(r-y)
  control(P,u[i])
end
```

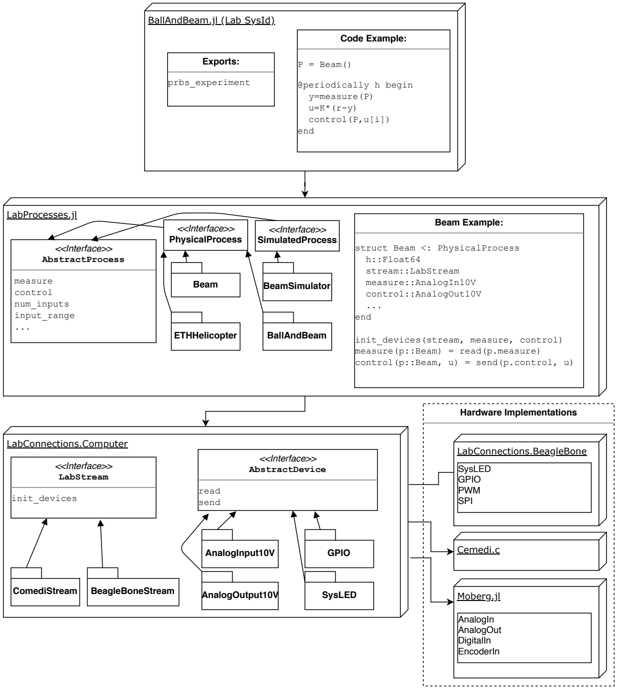


```
struct Beam <: PhysicalProcess
  h::Float64
  stream::LabStream
  measure::AnalogIn10V
  control::AnalogOut10V
  ...
end

init_devices(stream, measure, control)
measure(p::Beam) = read(p.measure)
control(p::Beam, u) = send(p.control, u)
```



LabConnections.Computer
<<Interface>>
LabStream
init_devices
<<Interface>>
AbstractDevice
read
send
ComediStream
BeagleBoneStream
AnalogInput10V
AnalogOutput10V
GPIO
SysLED



```
P = Beam()

@periodically h begin
  y=measure(P)
  u=K*(r-y)
  control(P,u[i])
end
```

```
struct Beam <: PhysicalProcess
  h::Float64
  stream::LabStream
  measure::AnalogIn10V
  control::AnalogOut10V
  ...
end

init_devices(stream, measure, control)
measure(p::Beam) = read(p.measure)
control(p::Beam, u) = send(p.control, u)
```

Future

What we have:

- Processes, what we have:
  - ETHHelicopter (julia 1?)
  - Beam/BallAndBeam/BeamSimulator
  - DoubleTank (in DoubleTankLab/DoubleTankSimulator, is it working? julia1? GUI exists)
- Ideas for more Processes:
  - Linear Servo with Pendulum (Already part of lab already in julia, needs hardware connection and MPC controller)
  - Furuta Pendulum (Needs everything)
  - Throttle (Needs everything)
  - Linear Servo (Martin H?)
- Other things:
  - Implement MobergStream
  - Your ideas?

Other

Note: Building julia yourself can speed up computations!

Searching packages: `pkg.julialang.org`

Notable Packages (Not mentioned so far in course):

Documentation: Documenter.jl

Saving Data: JLD.jl, HDF5.jl, Matlab: MAT.jl

Data: DataFrames.jl, Missings.jl

3D graphics: Makie.jl

Plots: Gadfly.jl, Winston.jl

Graphs: LightGraphs.jl, Graphs.jl

Maths: Interpolations.jl

Reactive Programming: Reactive.jl

Web Servers: Escher.jl, Genie.jl

Calling Python: PyCall.jl

Calling C functions: `docs.julialang.org/en/v1/base/c/`

Sockets: `docs.julialang.org/en/v1/stdlib/Sockets`