A Programming Model for Heterogeneous CPS from the Physical Point of View

Martin Richter Theresa Werner Matthias Werner

Chemnitz University of Technology (TUC)
martin.richter@informatik.tu-chemnitz.de

TECHNISCHE UNIVERSITÄT
CHEMNITZ

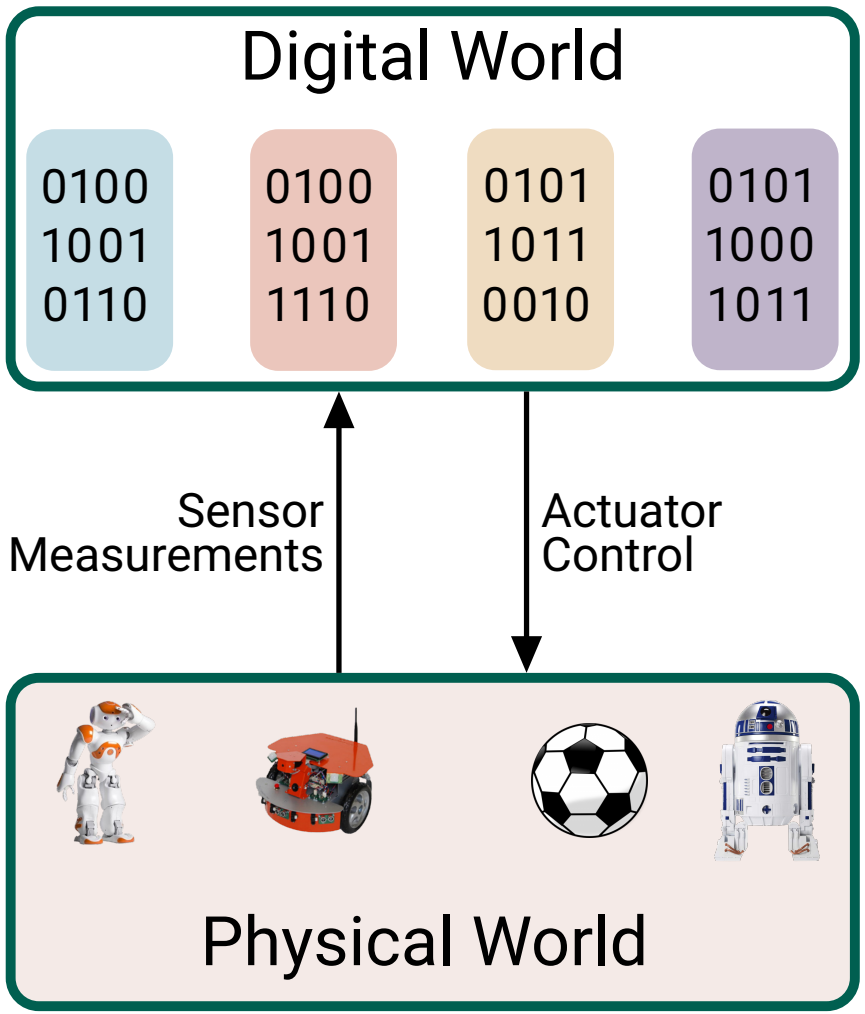
Short Resume

- Since March 2020 research assistant at the Operating Systems Group of the TUC
- 2016–2020 tutor for research and teaching
- 2013–2020 student of Applied Informatics and Automotive Software Engineering

- Research focused on programming models for CPS

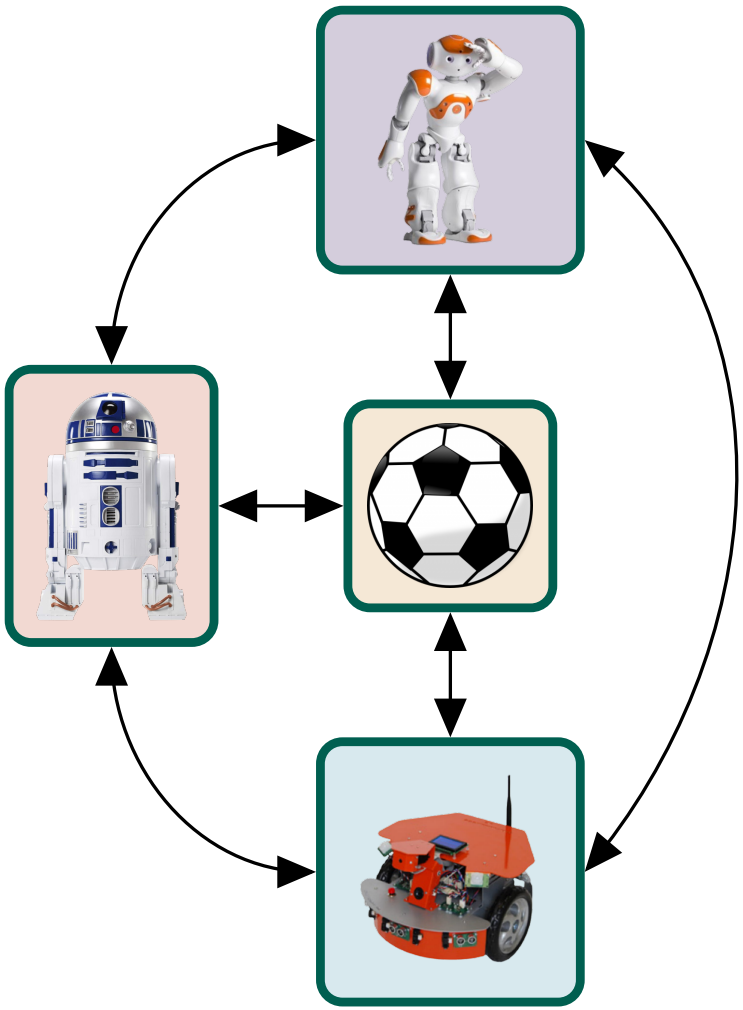
CPS

- Connect digital and physical world
- Multitude of heterogeneous sensors and actuators
- Observing through sensors
- Influencing through actuators



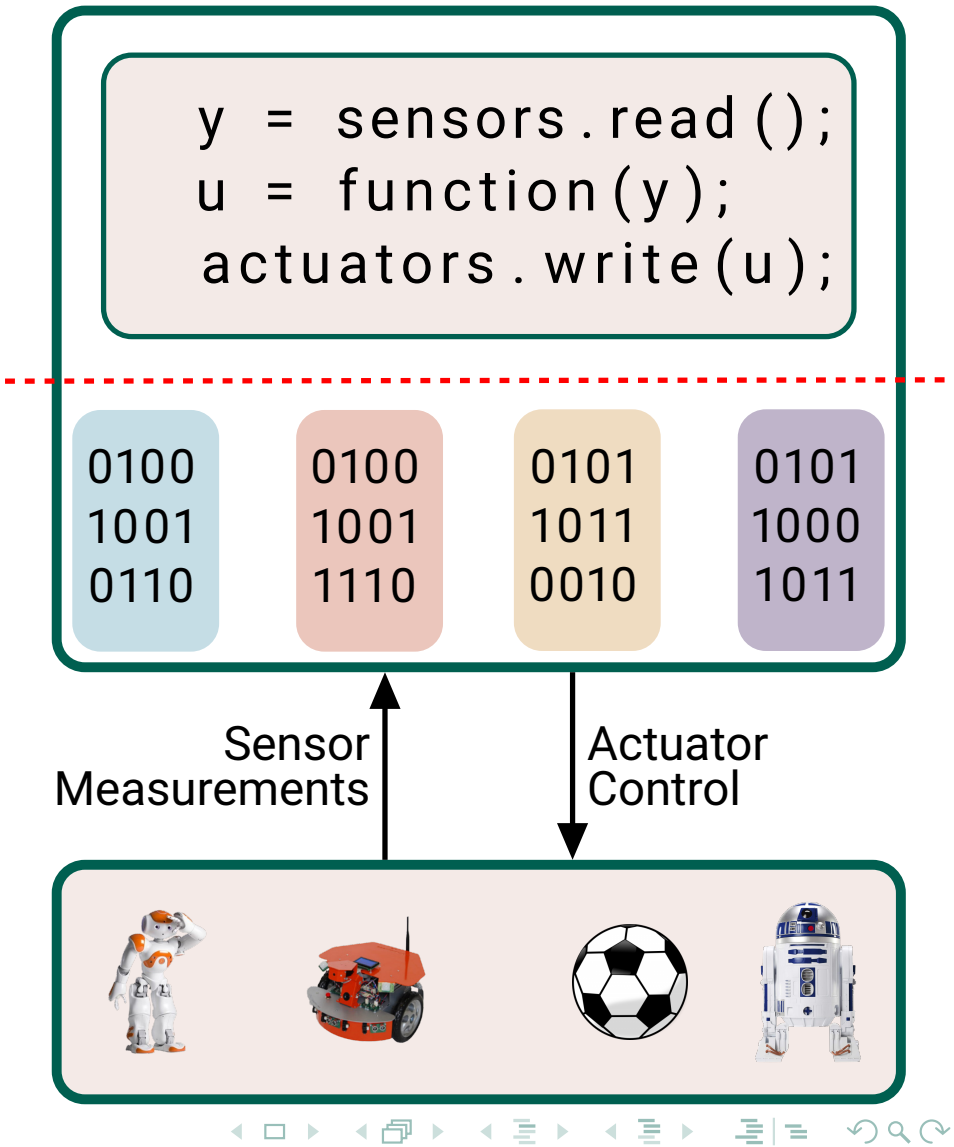
Running Example — Robot Soccer

- Heterogeneous players
- **Physical objects:**
  players, ball, goal
- **Sensors:**
  observing ball and players
- **Actuators:**
  influencing ball and players
- Distribution, Coordination, ...



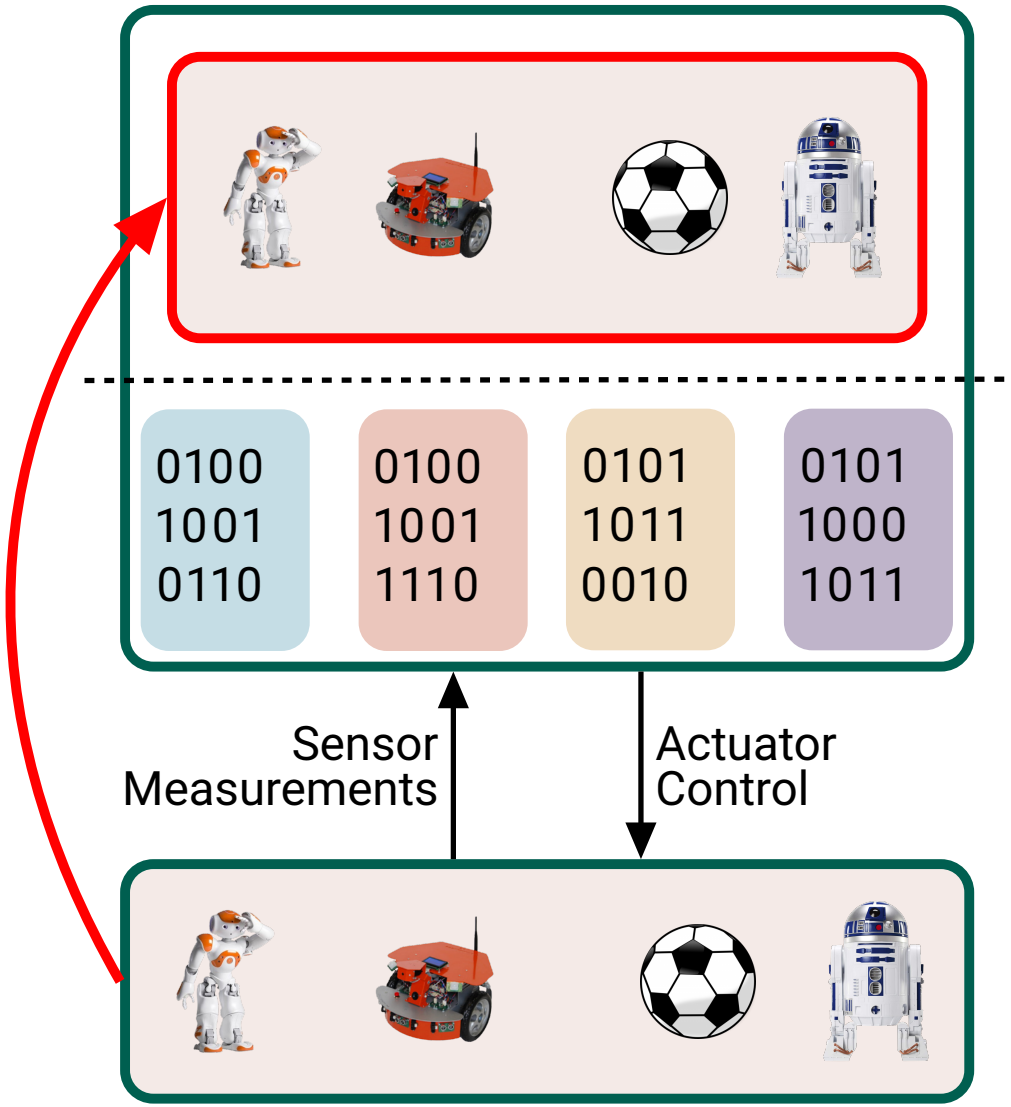
Challenges

- Distribution and heterogeneity
- Currently: digital point of view
- Impact on physical environment often not clear
- Disconnection between programmer's view and environment
- Programming is error-prone and complex
- We need **abstractions**!



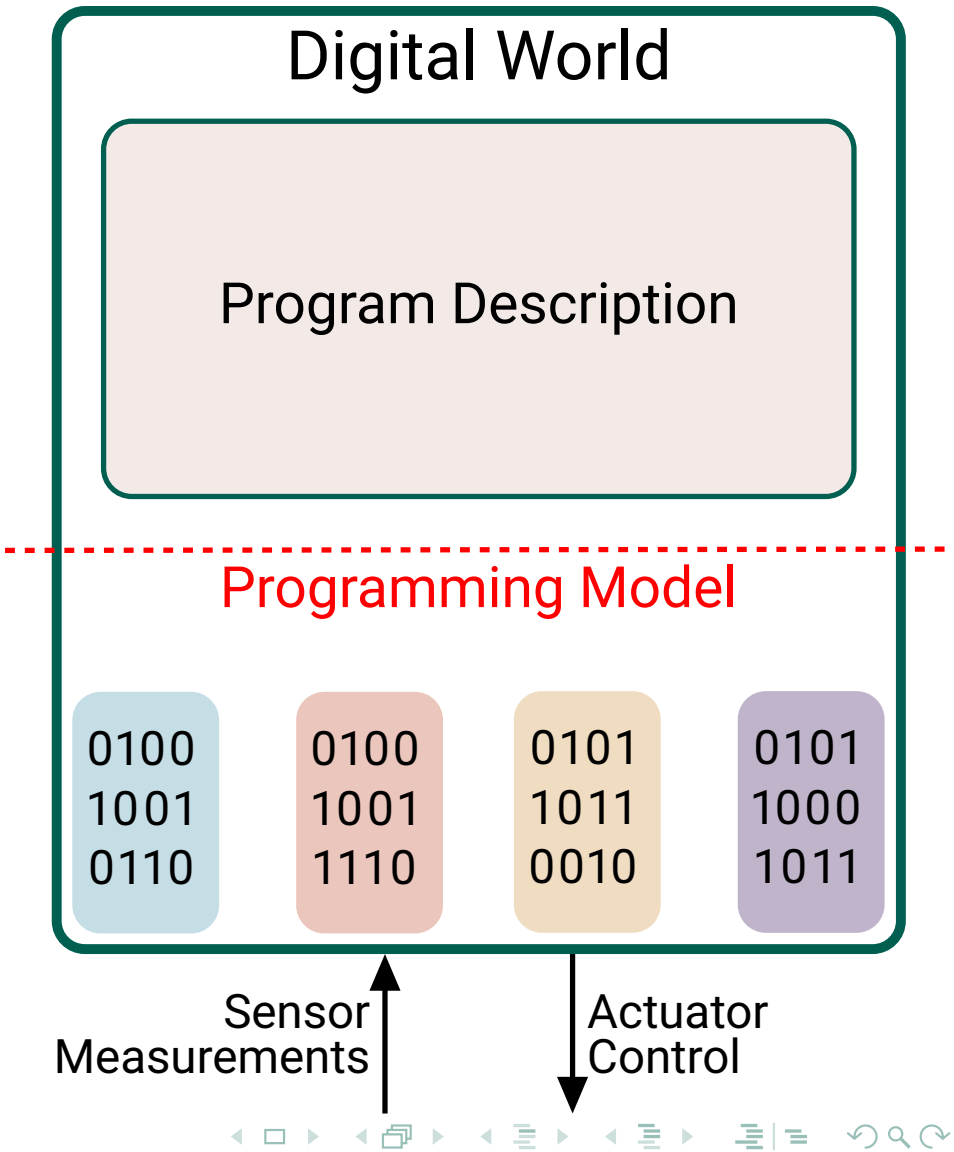
Abstractions

- Physical point of view
  - clear physical semantics
- Program physical objects
  - Which properties?
  - Influences on them
  - Desired change
- Sensor and actuator virtualization
  - Which devices achieve above is not relevant
- Systemic view



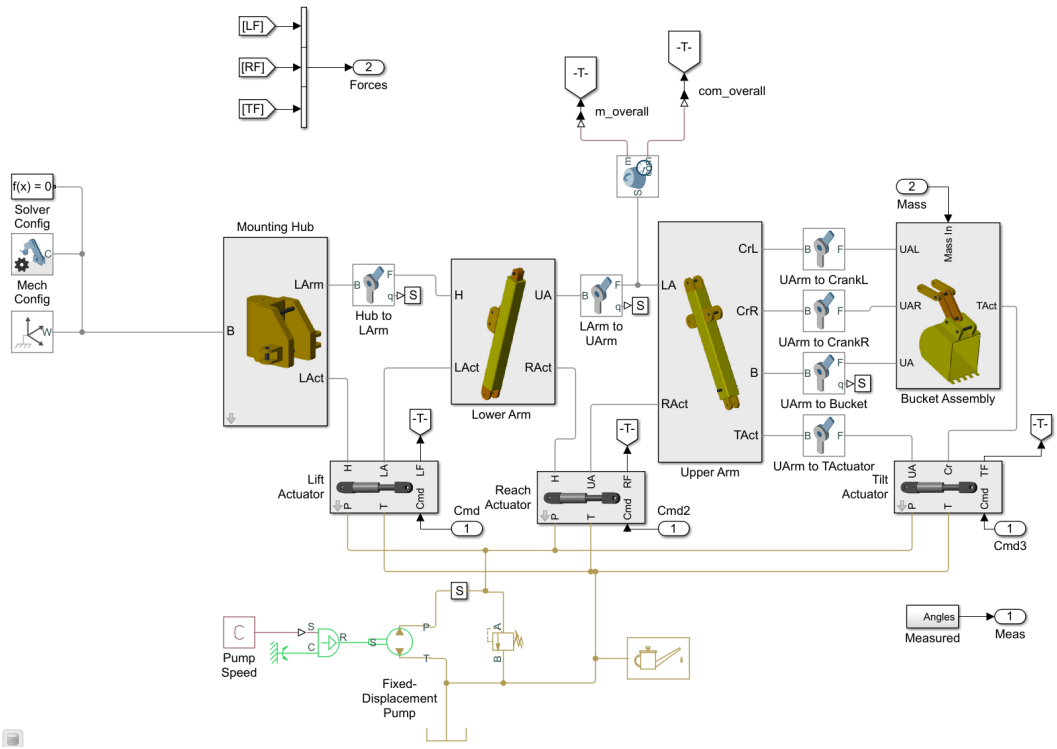
Programming Models

- Describe programmer's view on the system
- Allow to hide system features through transparency
- Allow to access them through awareness
- Describe translation of program description into actions



State of the Art — Physical Modeling

- Approaches for taking physical point of view
- Physical modeling languages (Modelica, Simulink, etc.)
- Description of physical processes
- Code generation for utilizing sensors/actuators
- No virtualization or distribution transparency

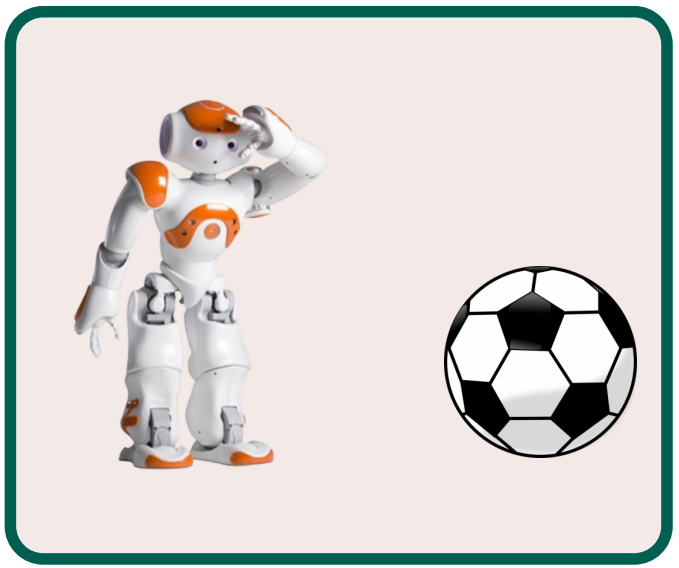


[1]

[1] https://www.mathworks.com/matlabcentral/fileexchange/39520-backhoe-model-in-simscape

Our Approach — The Programmer's View

- Developer programs physical objects
- Attributes for properties
  - What has to be measured?
- Methods for behaviour
  - How may it be influenced?
- Virtualization
  - Measurements matter, not sensors
  - Influences matter, not actuators
- Inheritance for similar properties and behaviour



```
Ball extends MovingObject {
    ...
}
```

Physical Objects — Attributes

- Describe physical properties (shape, velocity, mass, . . .)
- Constitute state of the object
- Measured by sensors
- Influenced by actuators

```
constructor() {
  this.mass = Mass(0.3[kg]);
  this.position = Position(?[m],?[m],?[m]);
  ...
}
```

Physical Objects — Methods

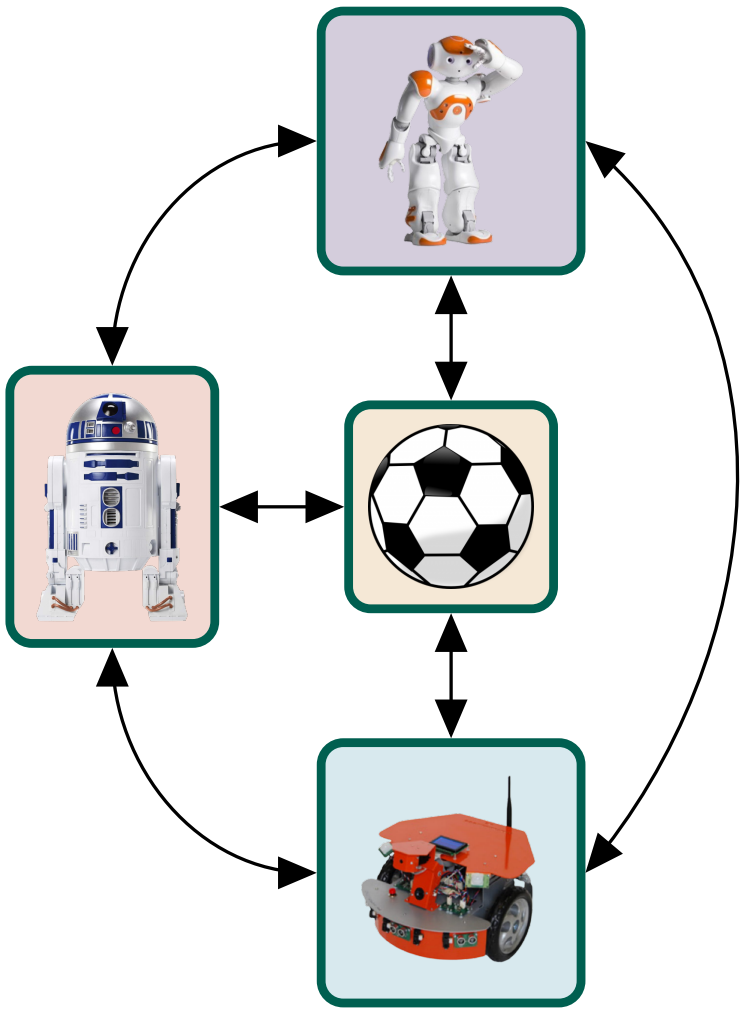
- Change of physical properties based on physical actions or internal dynamics $\rightarrow$ change of state
- Actuator inputs as parameters
- Possibly from different actuators
- Requirements on actuators to provide inputs

```
Require(Act(v) == Act(m) && Act(v).pos == this.pos)
method get_kicked(Velocity v, Mass m) {
   this.v = ...;
}
```

The Programmer's View — Physical System

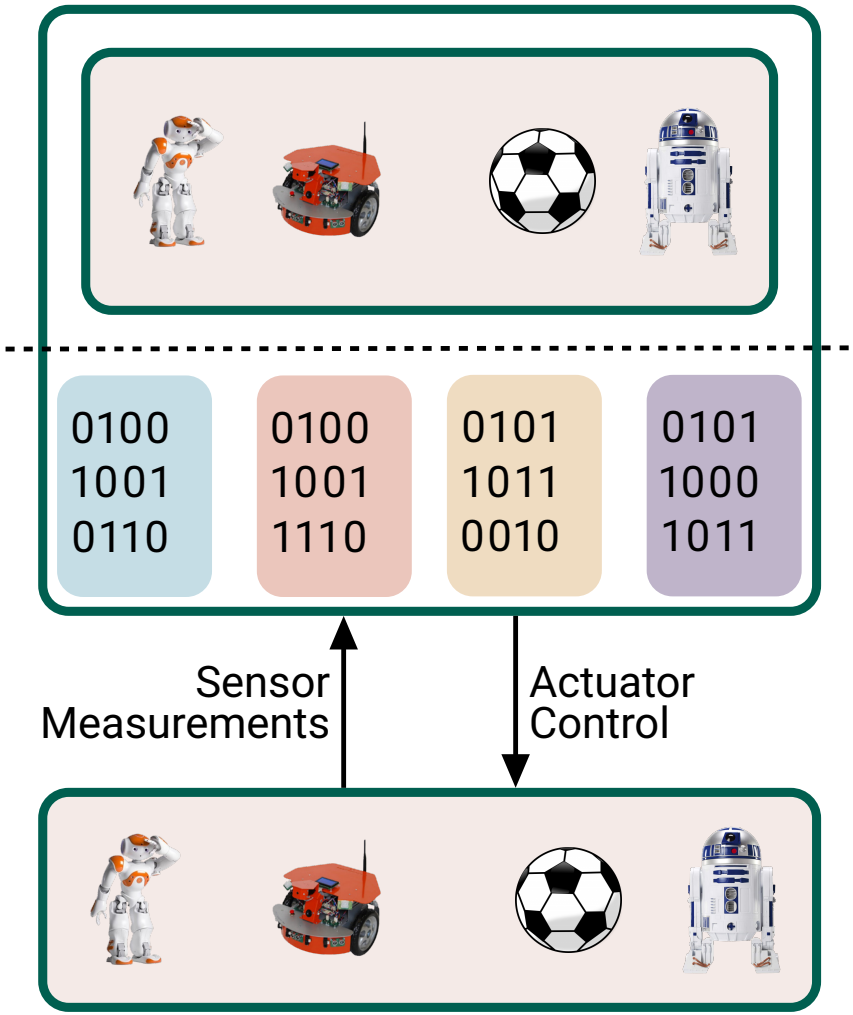
- So far: physical object state and behaviour description
- Bundle of properties and behaviours of objects
- Constraints for describing target state of physical system

```
target defense() {
    abs(player, opponent) <= 1.0[m];
    abs(player, ball) <= 0.5[m];
    ...
}
```

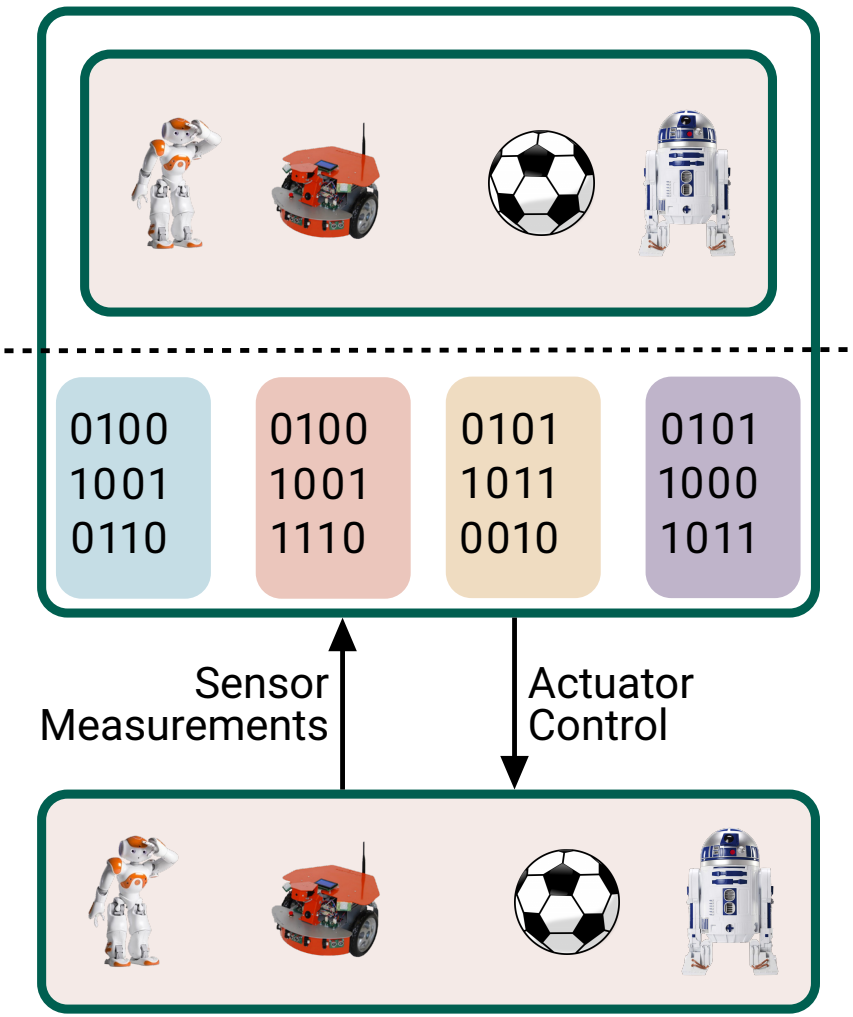


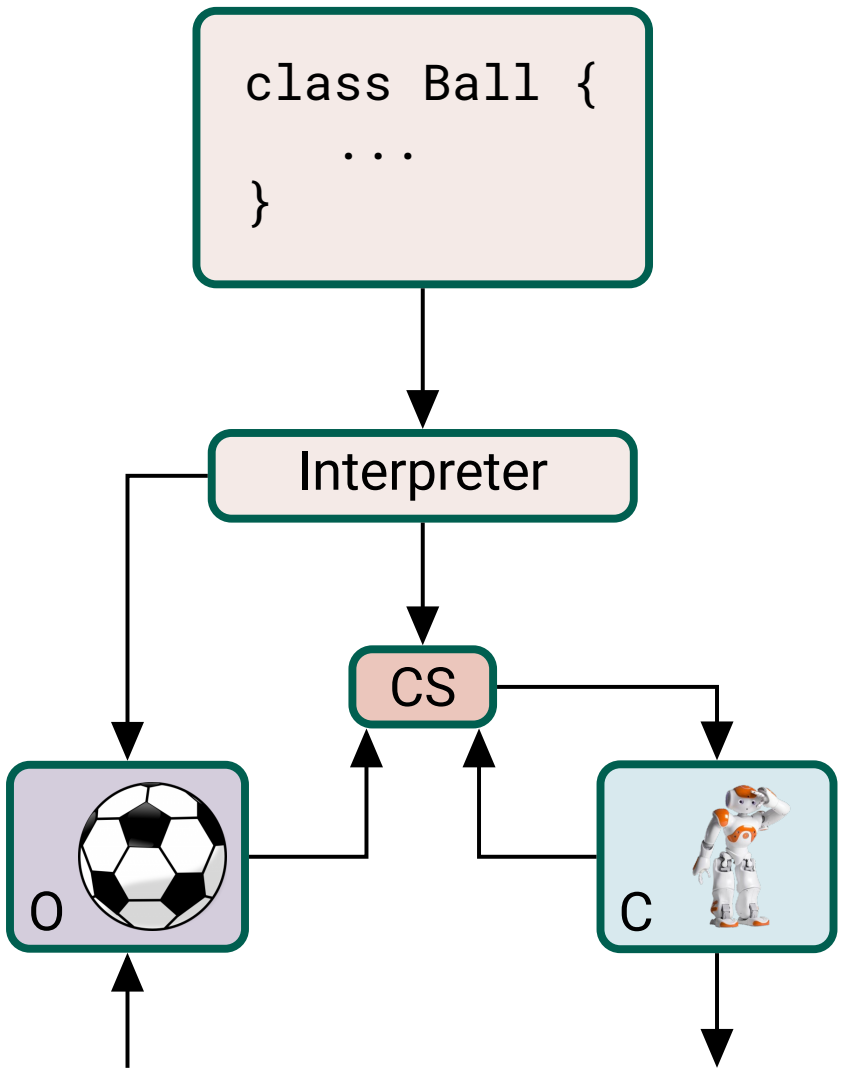
The Programmer's View — Recap

- Physical system of interest is group of physical objects
- OOP for describing physical objects
- **Attributes:** measured by sensors, constitute state
- **Methods:** change of state, based on possible acuator inputs and internal dynamics
- **Constraints** for describing target state space of system



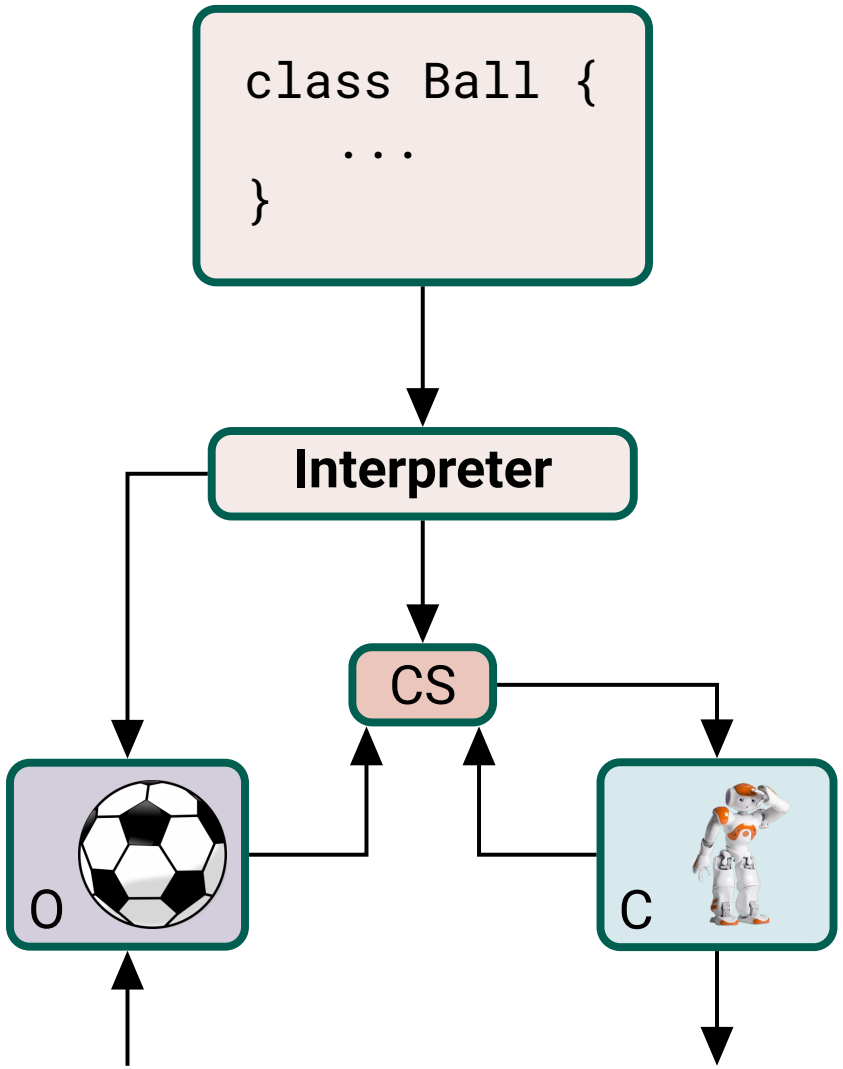
Runtime Environment





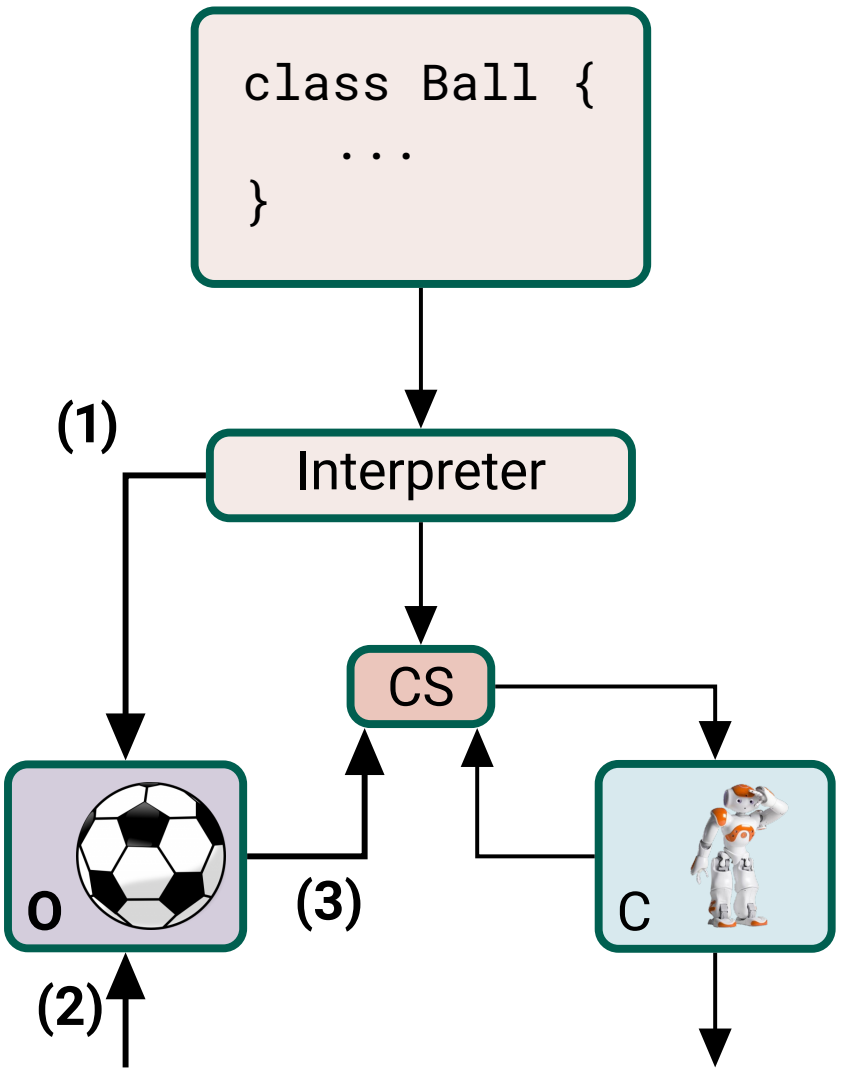
RTE – Interpreter

- Interpreter creates property specifications from attributes
  - What has to be measured?
- Creates state space equations from methods
  - How can it be influenced?
  - How does system evolve?



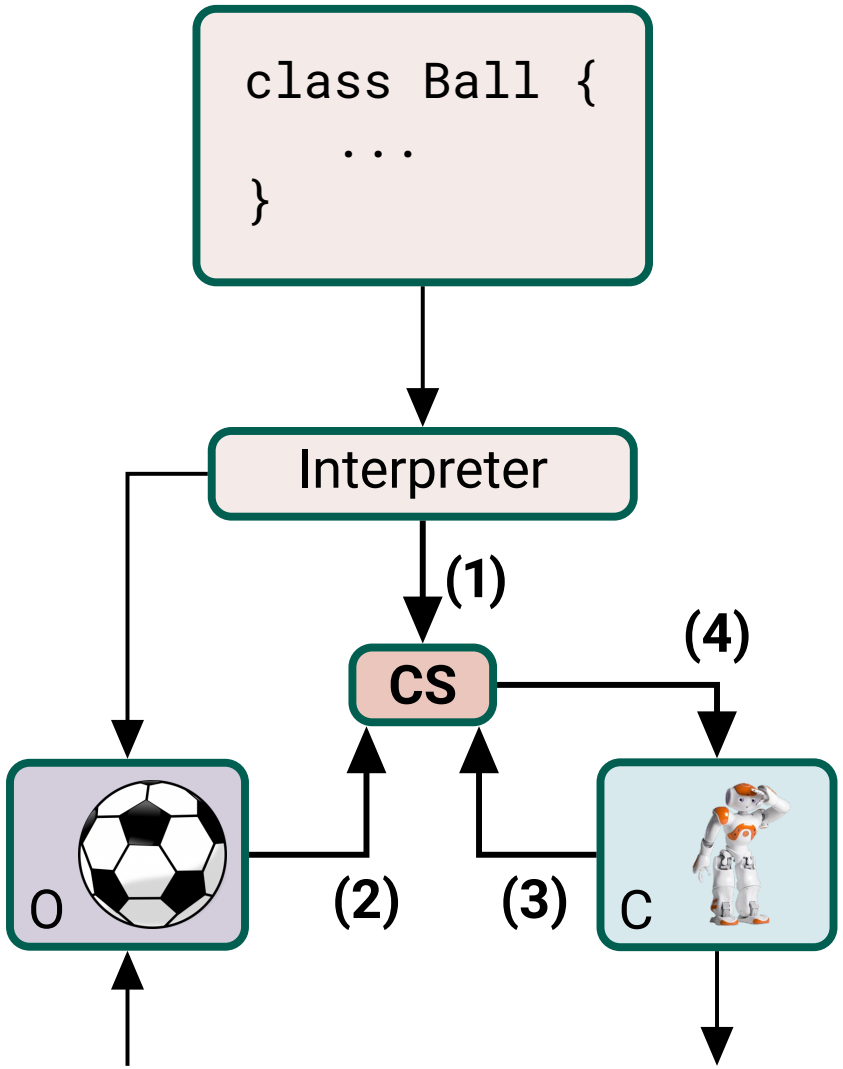
RTE – Observer

- Distributed observer (O) module observes physical objects of interest
- **Input:**
  - Property descriptions (1)
  - Sensor measurements (2)
- **Output:**
  - Class instances of observed objects (3)



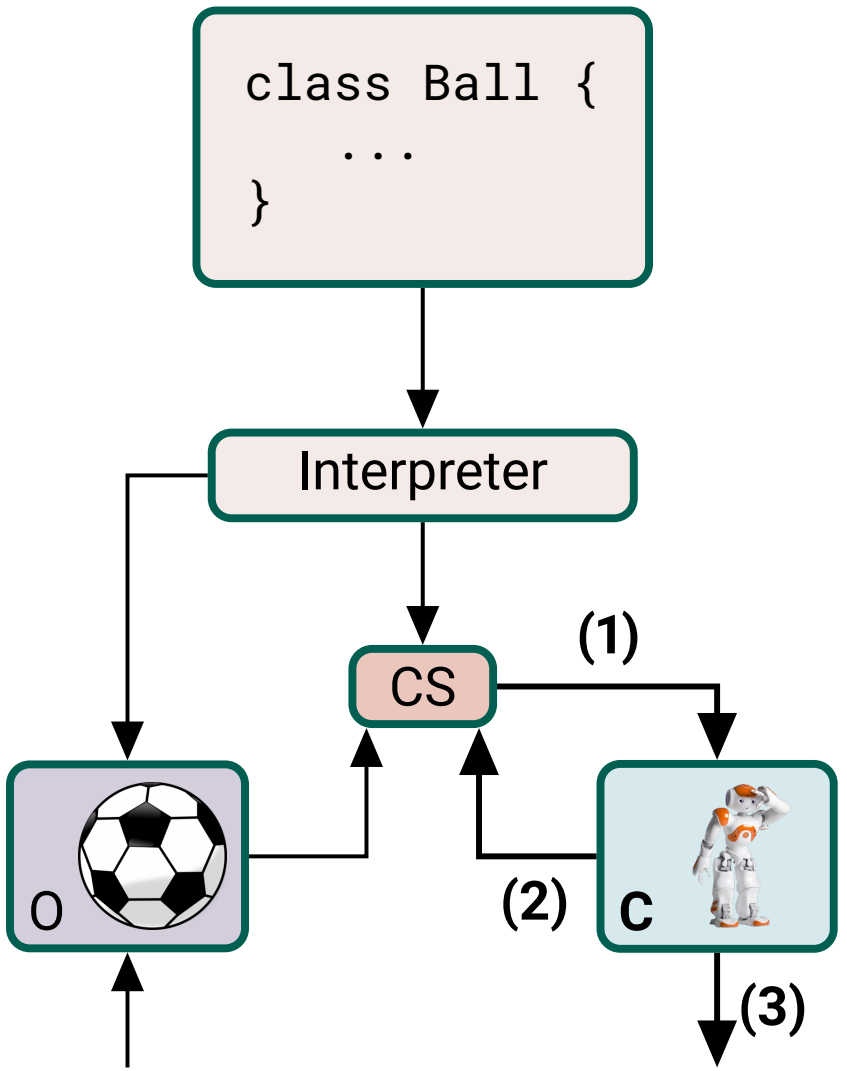
RTE – Constraint Solver

- Constraint solver (CS) solves state equations (1) for inputs to reach target state
- **Input:**
  - Measured object properties (2)
  - Available actuator influences (3)
  - Target state constraints and state equations (1)
- **Output:**
  - Required actuator influences (4)



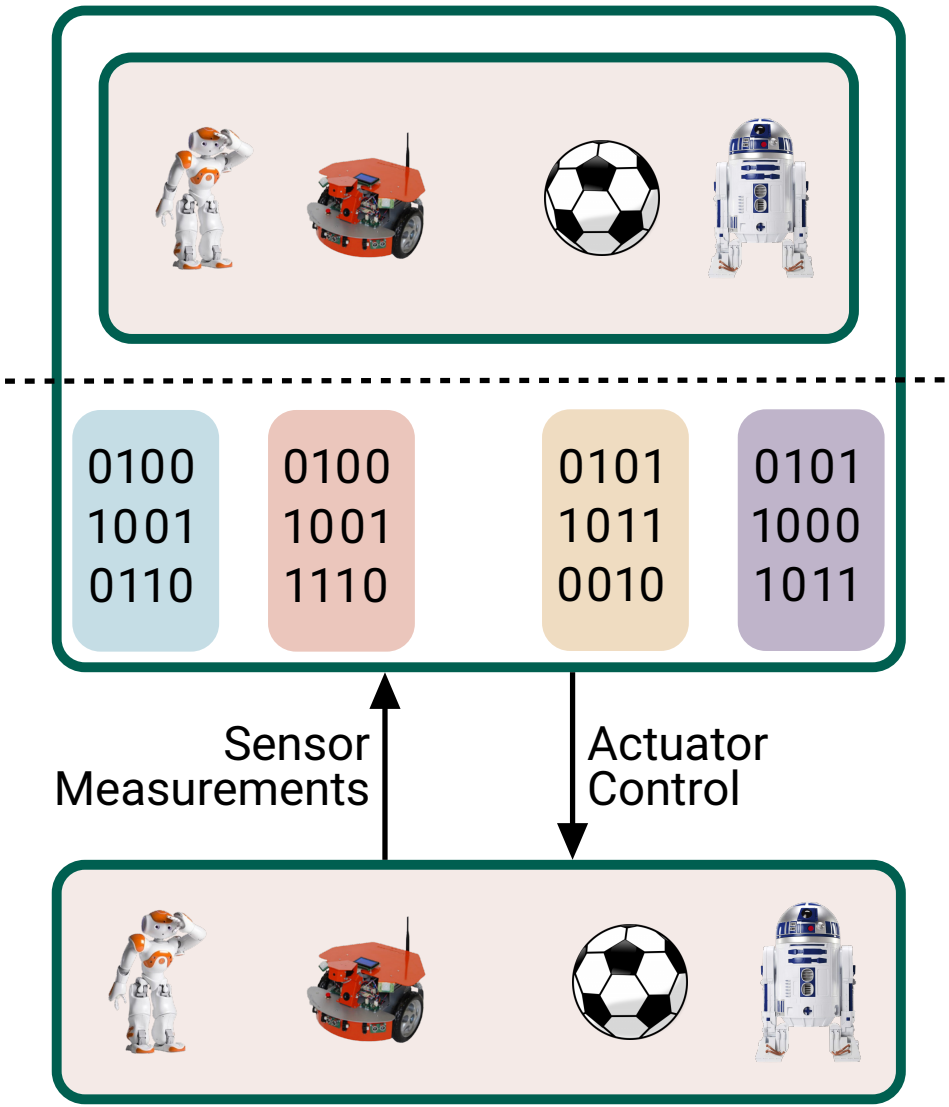
RTE – Controller

- Distributed controller (C) module for coordinating actuators
- **Input:**
  - Required actuator influences (1)
- **Output:**
  - Available actuator influences (2)
  - Actuator control (3)



Conclusion

- Provided conceptual programming model
- Allows description of desired behaviour of physical objects
- Physical impact on environment made explicit
- Sensor and actuator virtualization
- RTE for supporting programmer's view



Next Steps – Observer

- Sensor model for generic specification of sensors
- Allows mapping of available sensors to object properties
- Sensor fusion based on physical system description
- Data distribution and aggregation

Thank you!

State Equation

Method m, Class c, State $\vec{x}$, Actuator Input $\vec{u}$, System Σ:

$$\vec{x}'_{c,m}(t) = f_m(\vec{x}_c, \vec{u}_m, t) \xrightarrow{\text{linear}} \vec{x}'_{c,m}(t) = A_m \vec{x}_c + B_m \vec{u}_m(t) \tag{1}$$

$$\vec{x}'_c(t) = \sum_{m \in M} \vec{x}'_{c,m}(t) = \sum_{m \in M} (A_m \vec{x}_c + B_m \vec{u}_m(t)) \tag{2}$$

$$\vec{x}_\Sigma(t) = \begin{bmatrix} \vec{x}_{c^0} & \vec{x}_{c^1} & \dots & \vec{x}_{c^n} \end{bmatrix}^T (t) \tag{3}$$