On the Model Continuity in Control Systems Design Using DEVS, UML, and IEC 61499

Radek Kočí and Vladimír Janoušek

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 00 Brno, Czech Republic
koci@fit.vut.cz

https://www.fit.vut.cz/person/koci

ICSEA 2021, Barcelona, Spain, October 3-7, 2021

Presenter - dr. Radek Kočí

- assistant professor at Faculty of Information technology, Brno University of Technology
- teaching
  - software engineering
  - programming
  - artificial intelligence
  - operating systems
- research interest includes
  - modeling and simulation
  - formal models in software engineering
  - genealogy databases
- projects
  - cooperation on modeling and simulation of technological processes, co-author of two software tools
  - System for creating community genealogical databases
  - Virtual power plants – predicting the availability of cogeneration units
  - he was/is a team member of several Czech Science Foundation projects and EU projects, he led the CSF project

Selected topics and activities

- Simulation Driven Development
  - formal models in the system design and requirements engineering
  - validation through simulation / scenarios
  - model continuity
  - supporting tools
- Agent Systems
  - improving the reasoning of agents described by the Agent-Speak language
  - competition of multiagent systems – 2nd place in the first round
- Genealogy databases and tools
  - systems for transcription of parish records
  - creation of genealogy models (family trees, . . .)
  - TACR (Technology Agency of the Czech Republic) project: Possibilities of creation of community genealogical databases with semantic information and uncertainty

Motivation

- control systems: we suppose tree-level structure: sensors/actuators, distributed controllers, and a Supervisory Control And Data Acquisition (SCADA) system
- the standard IEC 61499 for industrial processes and control systems was established in 2005—it defines a generic model for distributed control systems based on function blocks
- nevertheless, a uniform procedure including a problem analysis or requirements specification is not defined
- we aim to describe the whole development process, from the conceptual models to their implementation following the Model-Driven Development (MDD) and Model-continuity principles
- the aim is to have **a unified approach independently of the target implementation environment**

MDD

- the development process is a series of constant refinement and transformation of models; ideally, more specific models are generated and, in the last step, the code for a particular platform
- UML is a standard language for modeling various aspects of software systems, both in academia and in industrial development

Model-Continuity

- the simulation model can evolve during the development process from a pure simulation until its final deployment in the target environment without re-implementation
- the UML has limited ability to simulate and investigate models in real conditions

$\Rightarrow$ we need a language/model/tool allowing **formal description and simulation as well as a run-time execution**

DEVS-based Modeling

- the Discrete Event System Specification (DEVS) represents a combination of formal modeling and simulation
- DEVS is well-defined, intuitive, understandable, and universal

⇒ **the DEVS simulation engine can become the run-time execution environment**

DEVS Simulation Engine

- PowerDEVS is the DEVS-based real-time simulation engine; it can be used in the role of runtime execution environment
- we also consider the possibility of using other environments for implementation, e.g., Node-RED flows, or IEC 61499 applications, e.g. the open-source development environment 4diac with the runtime environment FORTE

UML and the IEC 61499

- some works propose generating IEC 61499 from UML, or System Modeling Language (SysML), typically from a class diagram
- other works deal with the behavior of atomic components and propose a transformation of activity diagrams to the IEC 61499 execution control charts (ECC)

$\Rightarrow$ we propose a step from UML to DEVS

$\Rightarrow$ we focus on the structure of components and sub-components; we assume the availability of a library of well-defined atomic components in the target environment

Conceptual and Simulation Models

- there are approaches that attempt to transform conceptual models, such as those described by SysML, into simulation models

$\Rightarrow$ we do not consider the simulation model a goal but a potential option in the development

The Work Objective

- a uniform modeling methodology for heterogeneous implementation environments, e.g., 4diac/FORTE, PowerDEVS, or Node-RED
- the focus on the hierarchical organization of components and the transformation between similar environments
- we assume the library of well-defined atomic (domain=specific) components in the target environment

The Proposed Solution

- conceptual modelling using UML use case diagrams and class diagrams
- creation of Platform Independent Models (PIM)
  - using UML class and component diagrams
  - using DEVS with the same effect
- direct seamless transition from PIM to Platform Specific Models (PSM) using DEVS

Case Study

- central heating with zone valves
- each zone contains a temperature sensor, valve actuator, and Human-Machine Interface (HMI)
- the zone controller compares sensor data and setpoint and decides whether to open or close the radiator valve
- the central controller sets the boiler On/Off according to the state of zone valves; it also contains interface to central HMI/SCADA

Case Study – HMI Example

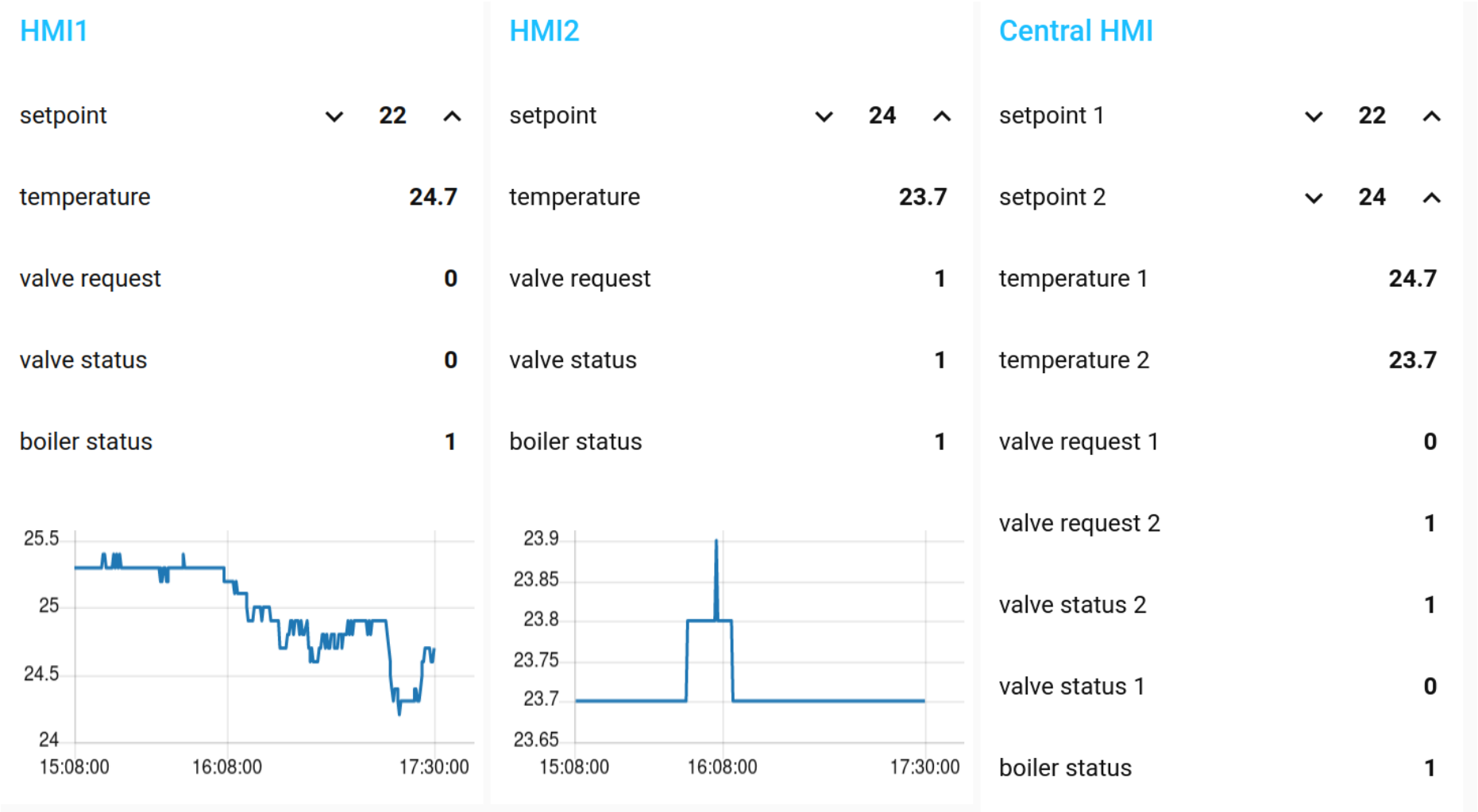


Figure: Zone HMIs and a central HMI example.

Conceptual Models

- Use Case Model – to define the system's requirements
- Domain Model – the system's conceptual elements (classes) that are needed to solve individual use cases.

Platform Independent Models (PIM)

- Component Model – a specialization of a structured class; it can be understood as an entity encapsulating a more complex structure of classes (and thus other components) and interchangeable with another component that meets the required interface

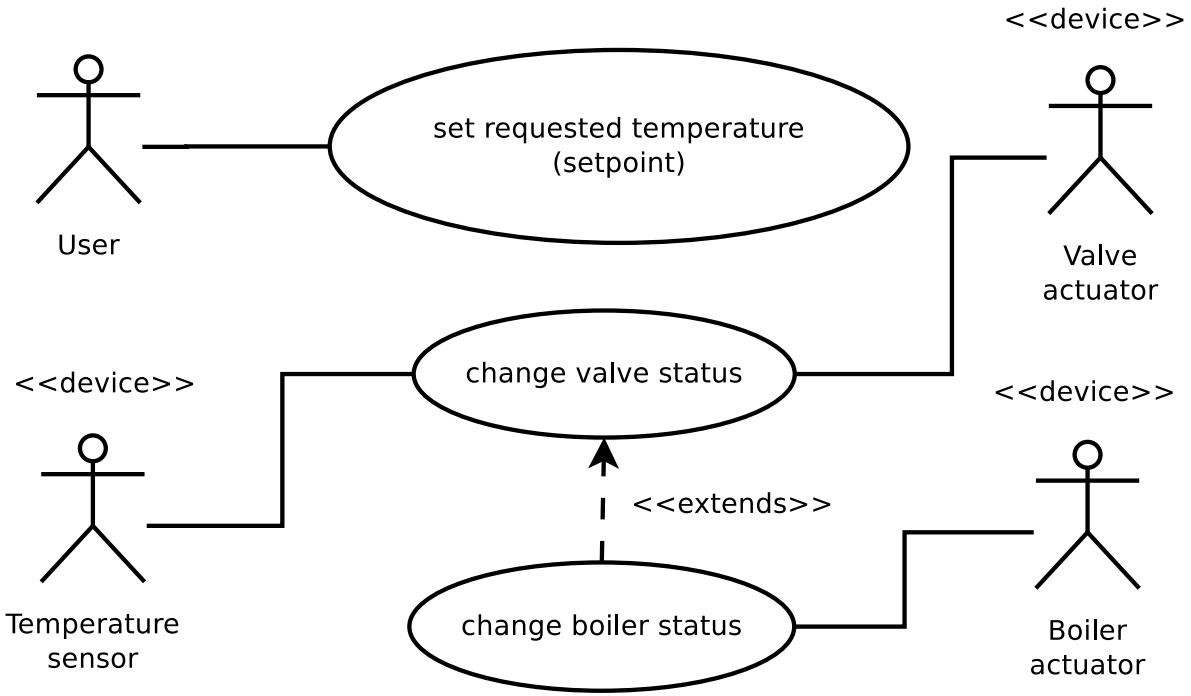


Figure: Use case model of the Case study.

- ZoneController – sets of conceptual classes covering the zone controller structure and behavior
- CentralController – sets of conceptual classes covering the central controller structure and behavior

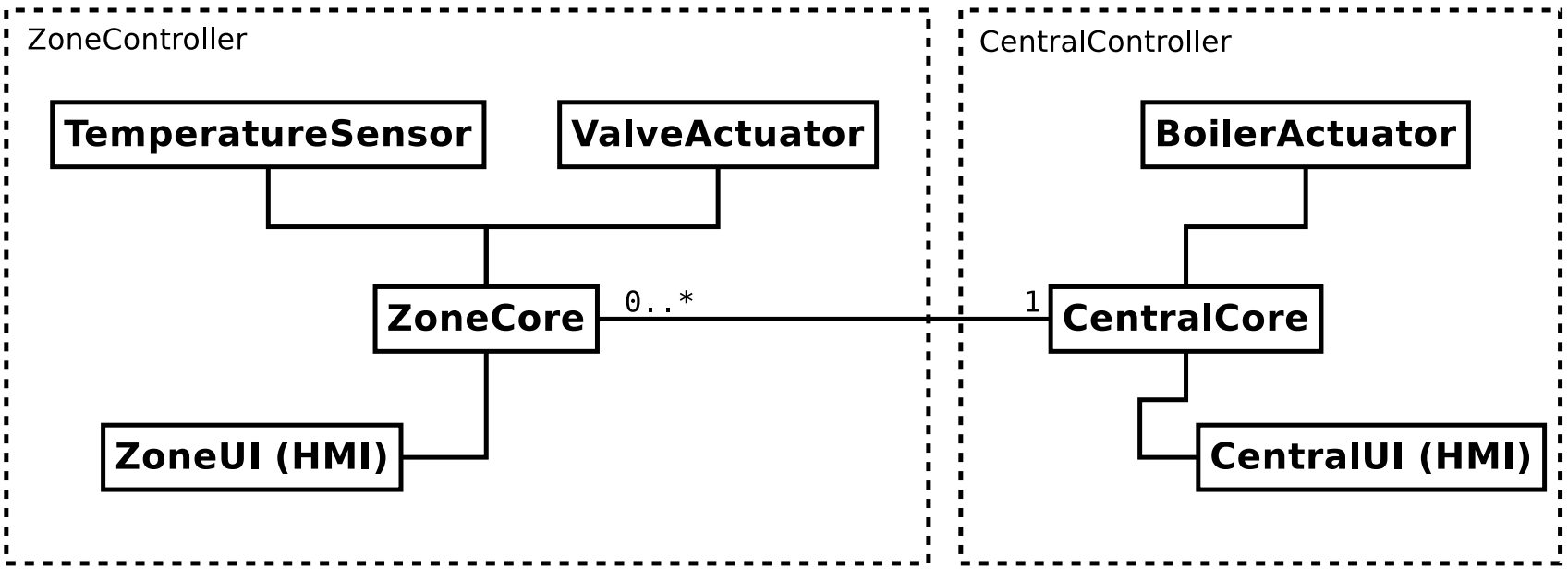


Figure: Basic domain model of the Case study.

Component model – ZoneController

- provided interface ZoneInterface, through which the component receives external events
- required interface CentralInterface from the connected components

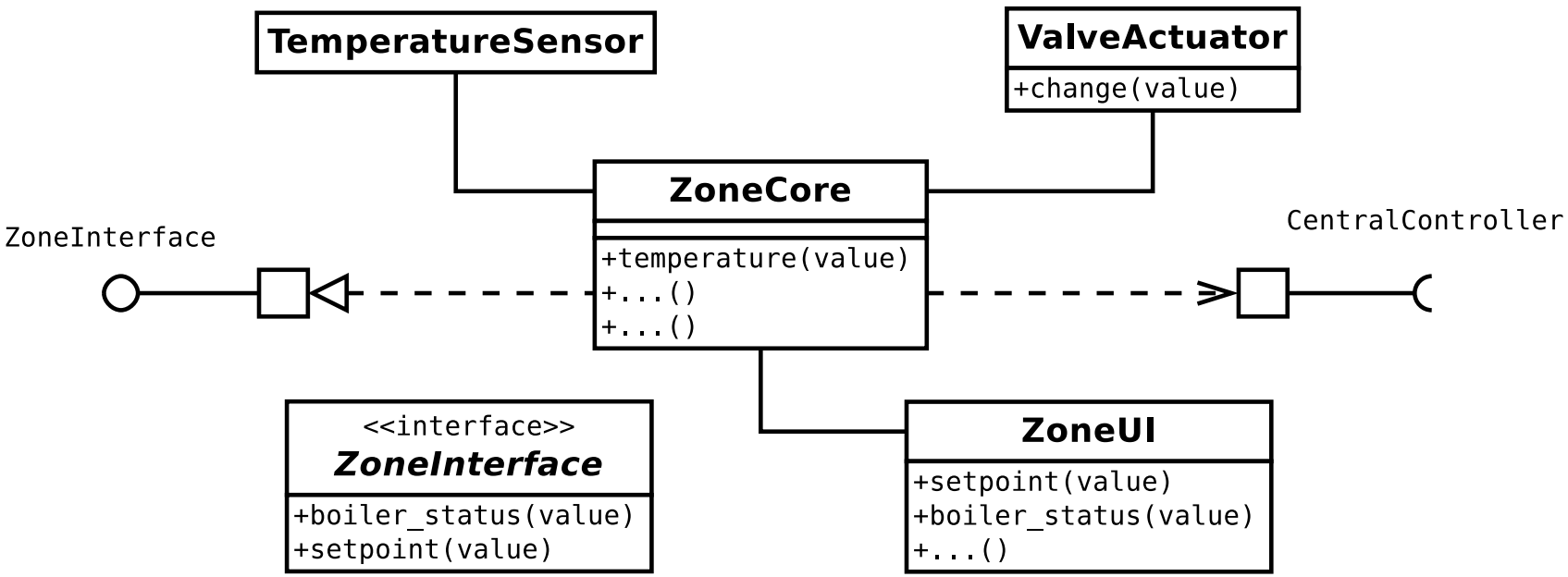


Figure: Component model of the ZoneController conceptual class.

- provided interface CentralInterface, through which the component receives external events
- required interface ZoneInterface from the connected components

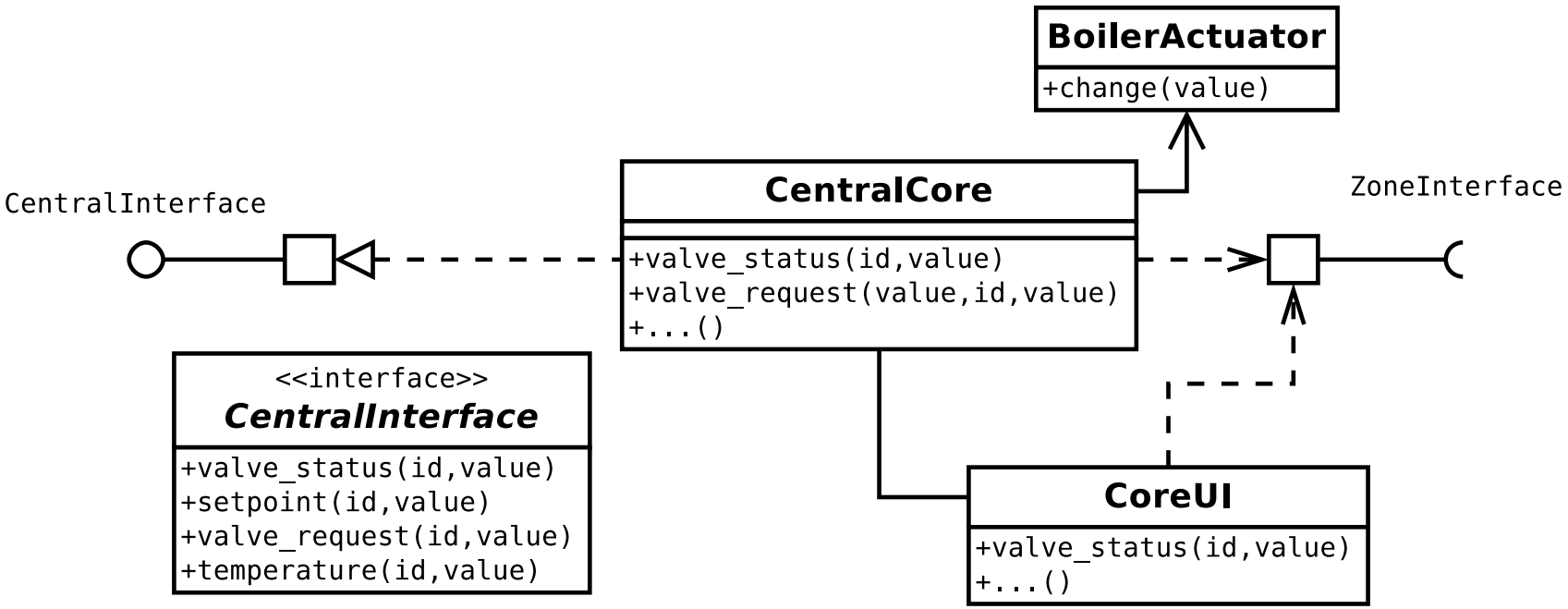


Figure: Component model of the CentralController conceptual class.

- it is better, especially for clarity, that each event is associated with an individual port of the component
- we can derive specific interfaces from the original interface in the component diagram
- each such interface contains only one method corresponding to the event

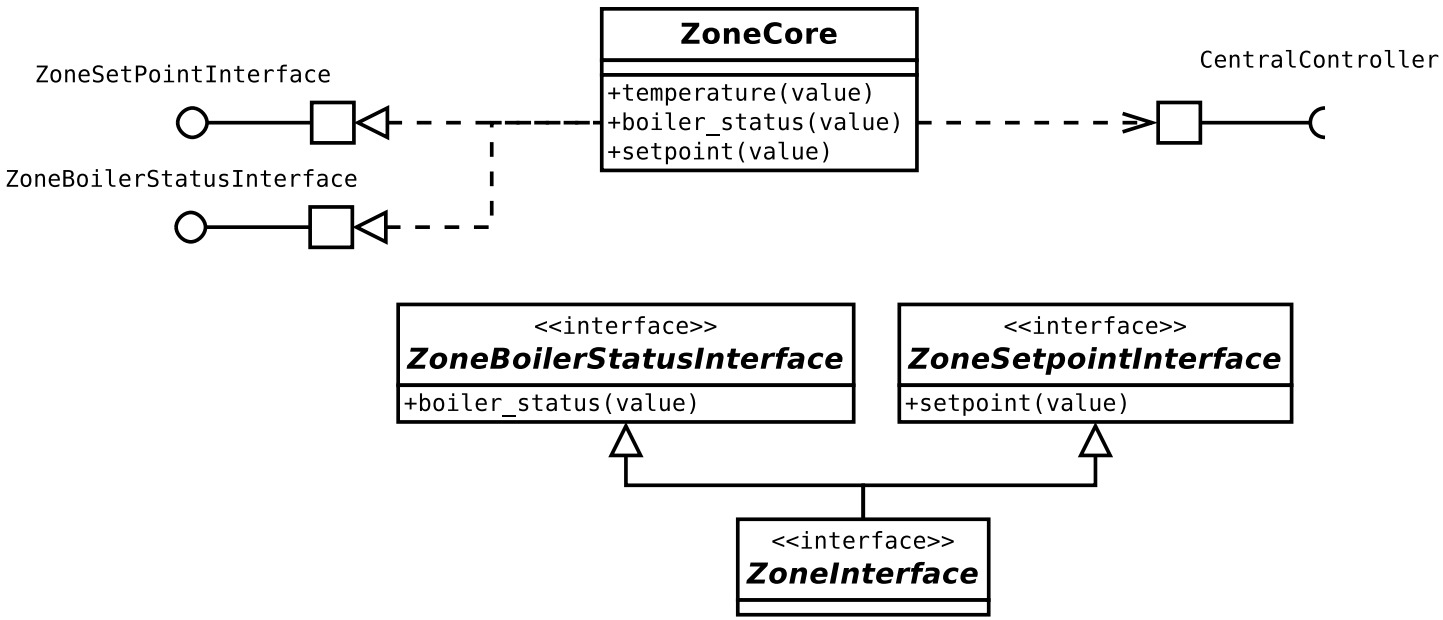


Figure: Part of the ZoneController component model having individual ports for each event.

- a composite component consisting of other interconnected components
- a simple system consisting of one zone controller
- simple, so-called atomic components, which are no longer further divided into internal parts, can be described by some of the behavior models, or another suitable formalism can be attached to them

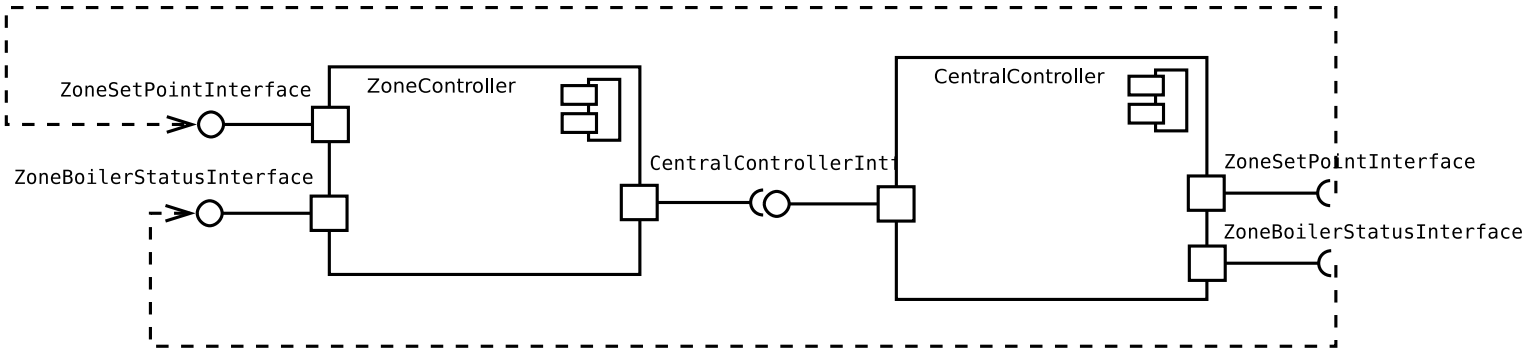


Figure: System component model.

Platform Independent Models (PIM)

- *Component Model* – *a specialization of a structured class; it can be understood as an entity encapsulating a more complex structure of classes (and thus other components) and interchangeable with another component that meets the required interface*
- DEVS Model – similar effect can be achieved with the DEVS formalism

Platform Specific Models (PSM)

- DEVS Model – it allows model continuity from PIM; it can be directly simulated with a suitable tool and considered an actual control system model and its implementation

Model-Continuity

- we can create the same component diagram using the DEVS formalism
- DEVS can represent both the model and the implementation of a specific system
- we can systematically derive implementation for a specific platform, like PowerDEVS, Node-RED, or 4diac/FORTE.
- the DEVS PIM and PSM models merge – they differ mainly in the implementation of atomic components.

PowerDEVS

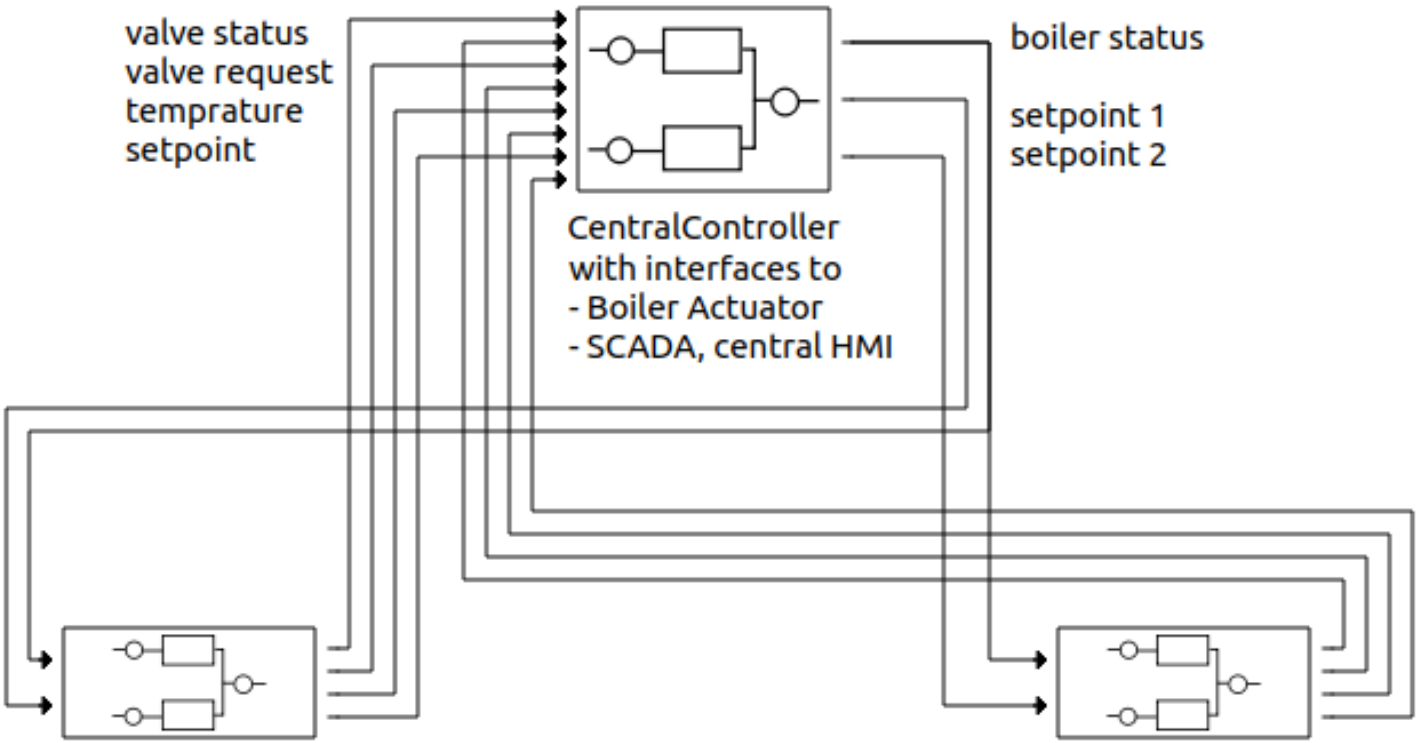
- PowerDEVS model can be directly used for implementation and deployment
- simulator works in real-time mode, with hadrware and network interface components connected

PowerDEVS model (1/3)

Central Heating with Zone Valves. Each zone has its own controller with sensor, actuator and HMI. Datapoints can be accessed by local HMI as well as by central HMI/SCADA.

Zone HMI data poits:
- valve status
- valve request
- temperature
- setpoint

SCADA and central HMI data points:
- heat request
- boiler status
- plus all zone HMI points



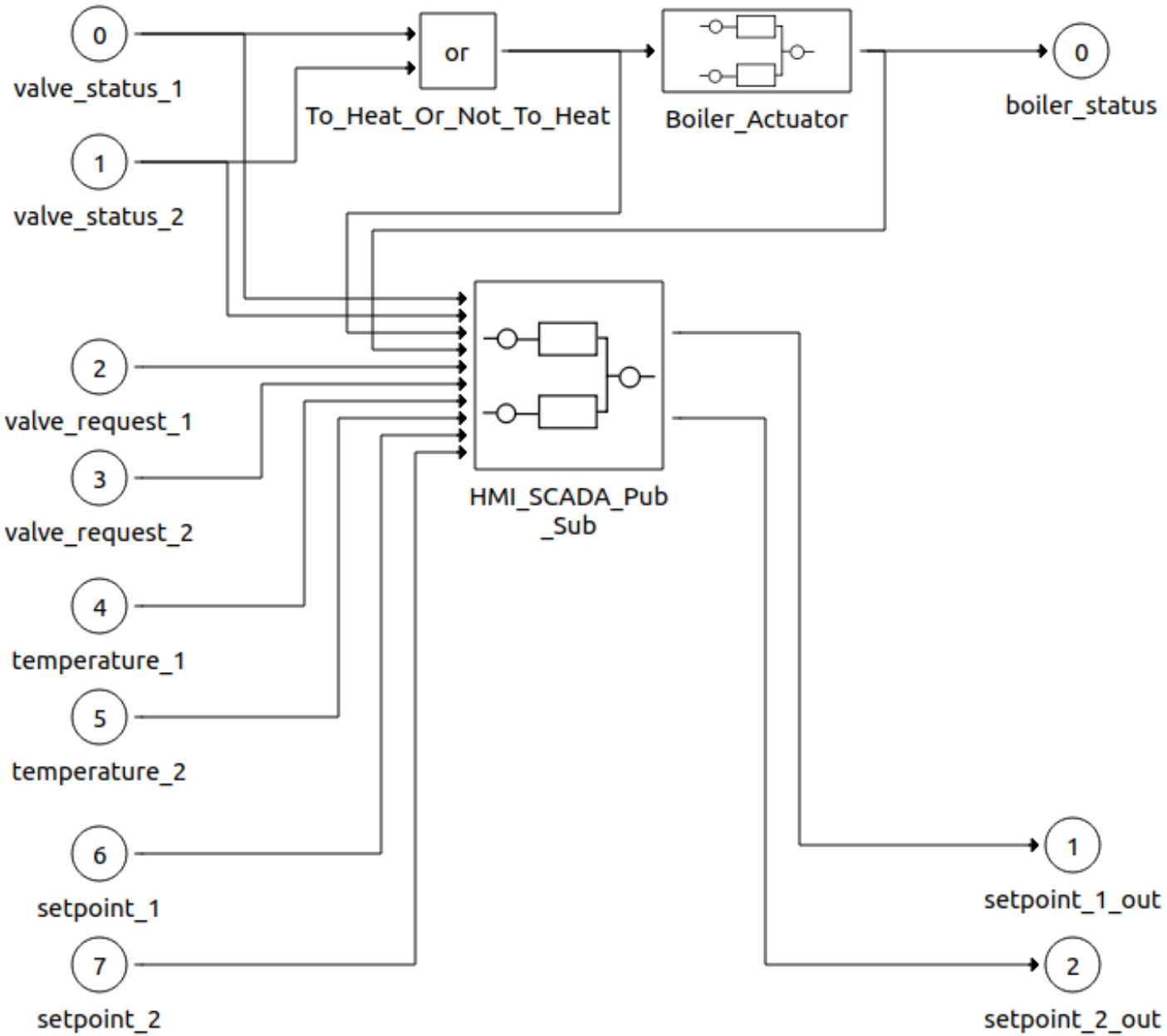


Figure: PowerDEVS model of central controller.

Zone Controller decides whether to open or close radiator valve.
It contains interfaces to
- Temperatute Sensor
- Valve Actuator
- Local HMI

Outputs and inputs are supposed to be connected to central controller.

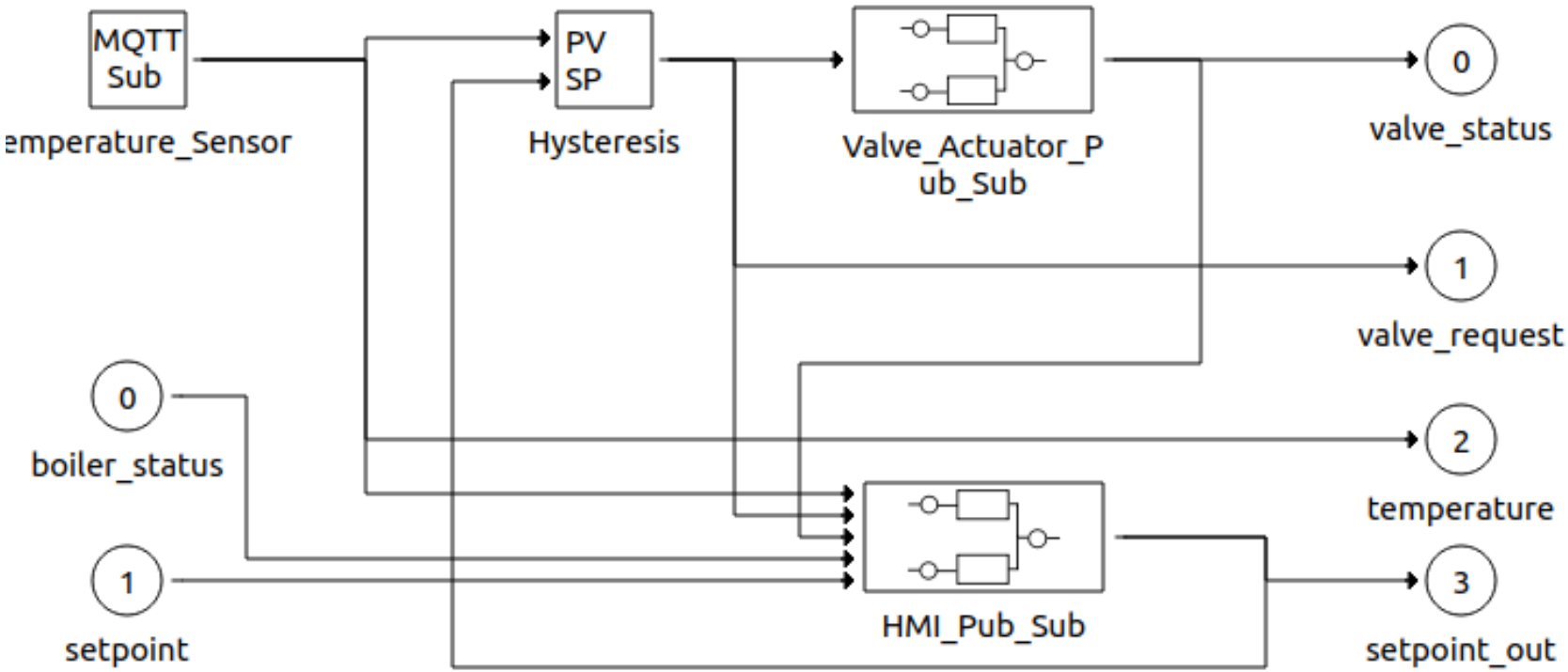


Figure: PowerDEVS model of zone controller.

Why?

- coordination level of IoT system can be modeled and implemented using Node-RED (soft real-time is sufficient)

How?

- flows can be hierarchically organized (like in DEVS)
- input ports are modeled by adding topic to messages (using "change" nodes in the flow)

Node-RED model (1/3)

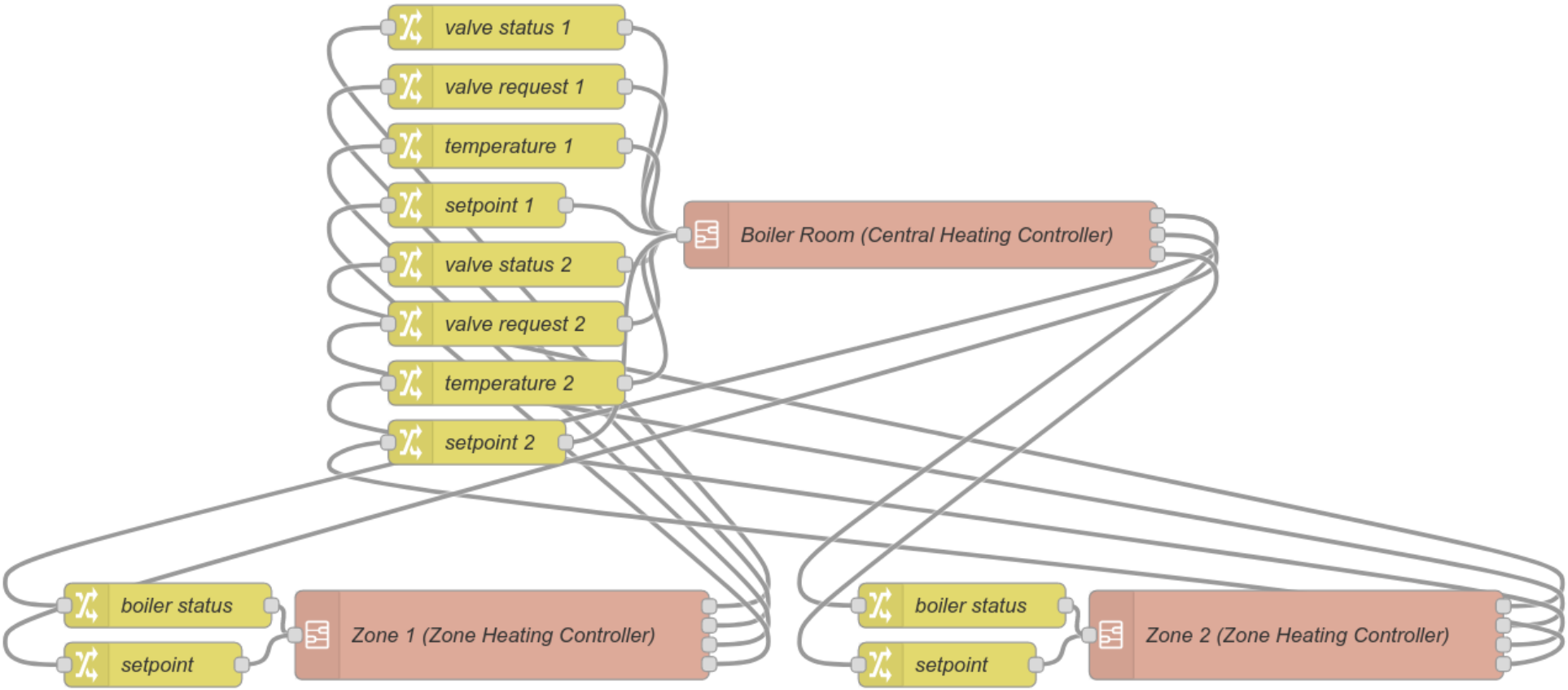


Figure: Node-RED model of the system.

Node-RED model (2/3)

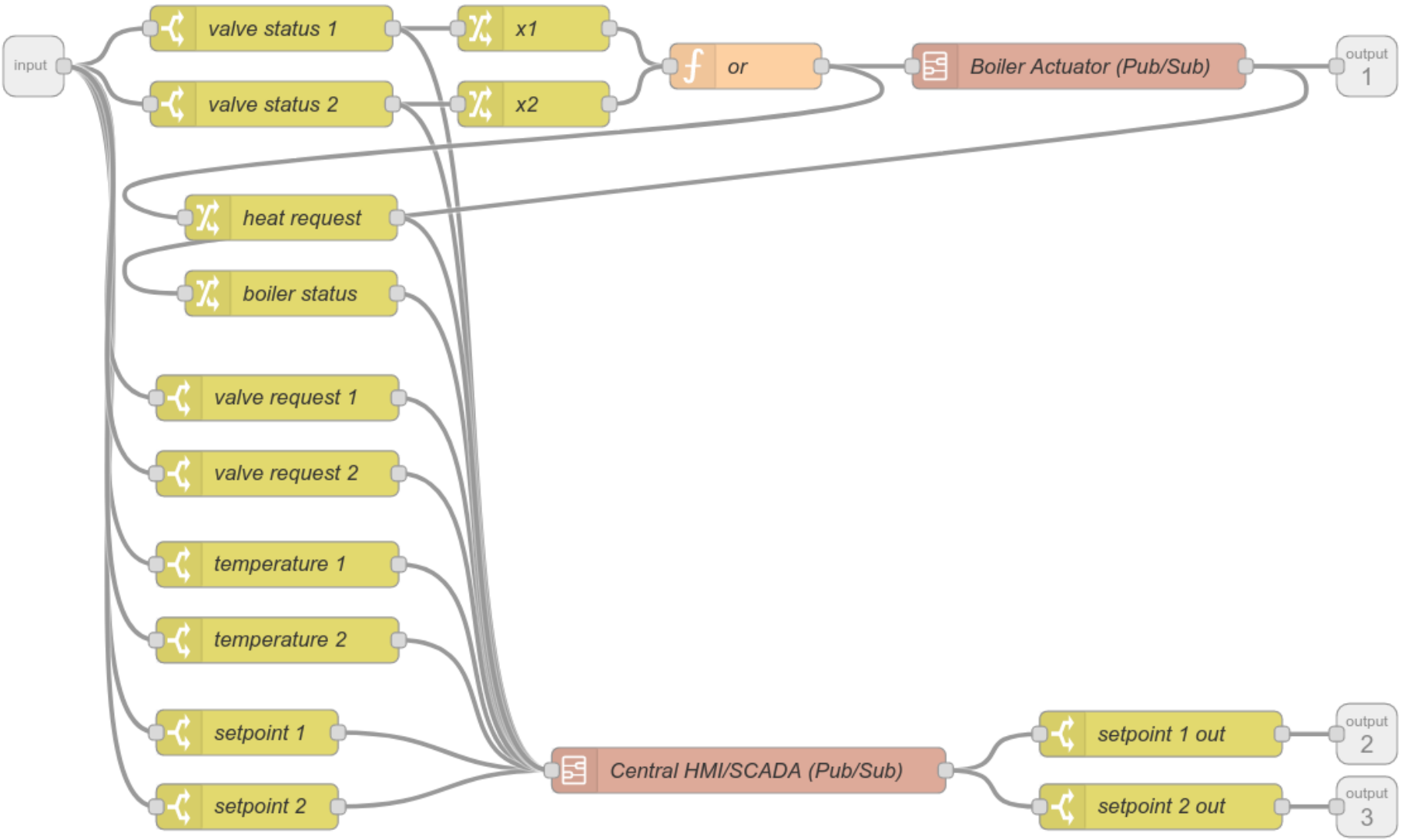


Figure: Node-RED model of central controller.

Node-RED model (3/3)

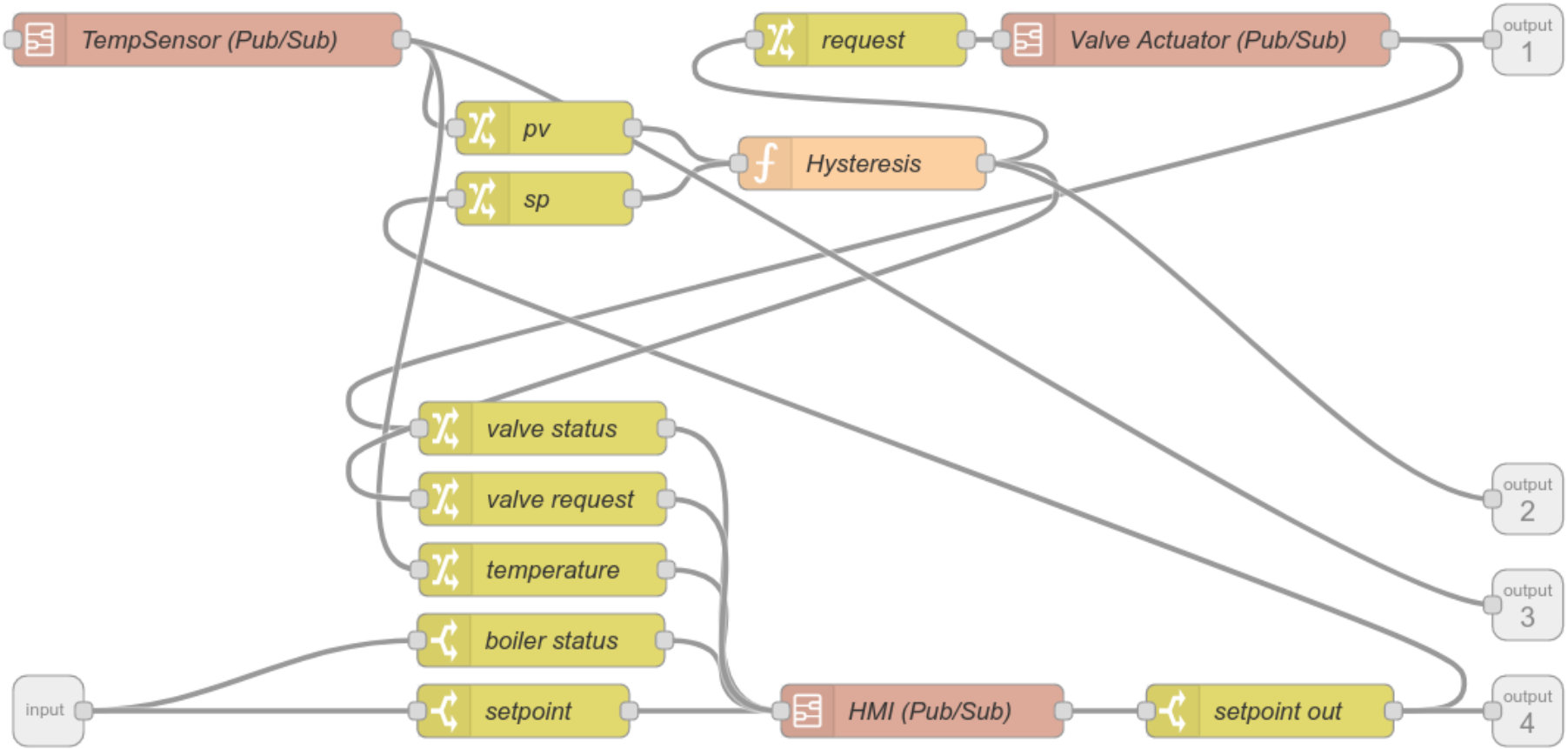


Figure: Node-RED model of zone controller.

Why?

- industrial standard
- real-time features
- native support for distributed deployment

How?

- deal with data ports and event ports (see next slides)
- adapt the connection of the output ports of components to the input ports of other components (in some cases, we have to include split and merge components)

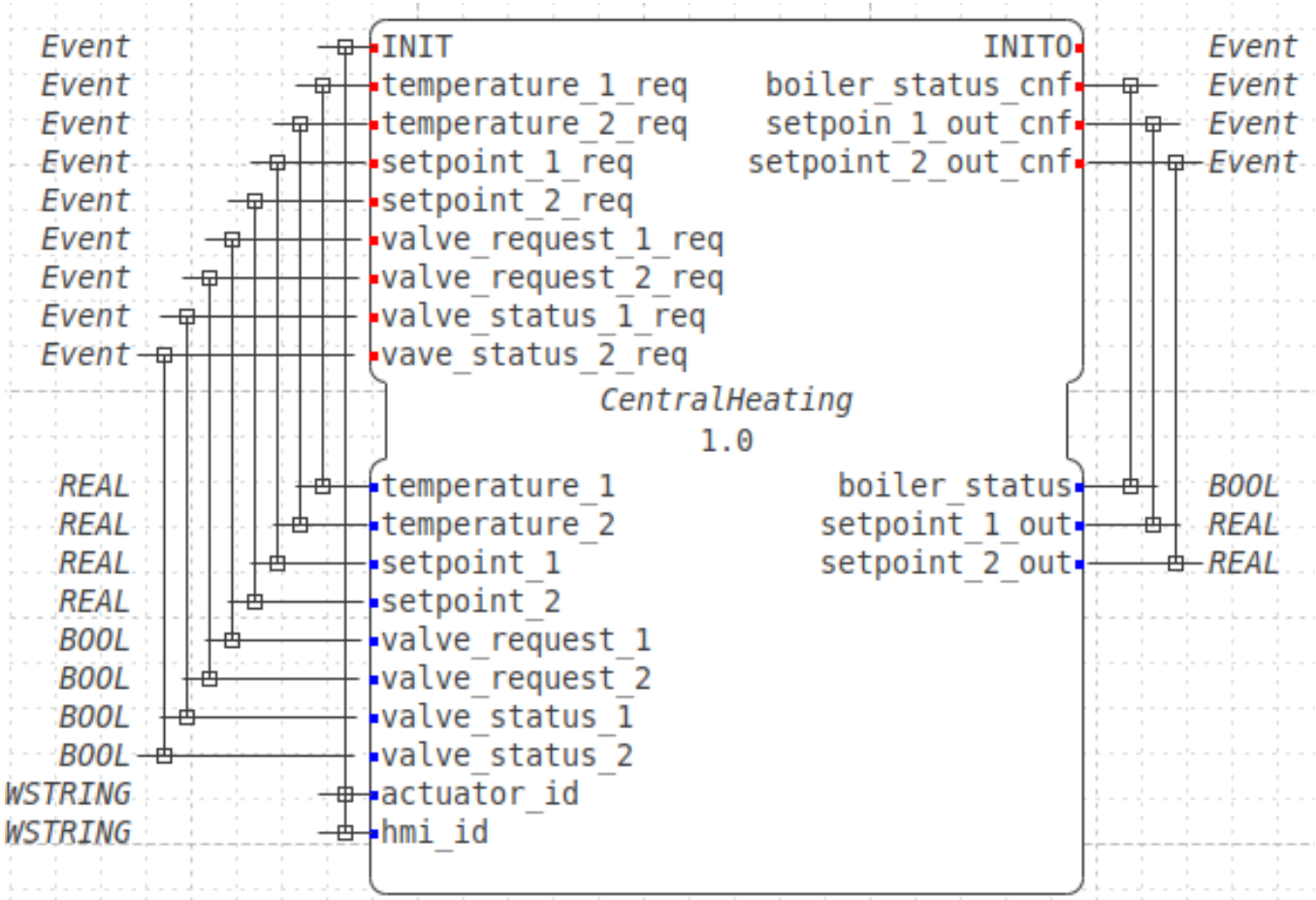


Figure: Interface of Central controller. To be DEVS-compatible, each data input and output has its associated event.

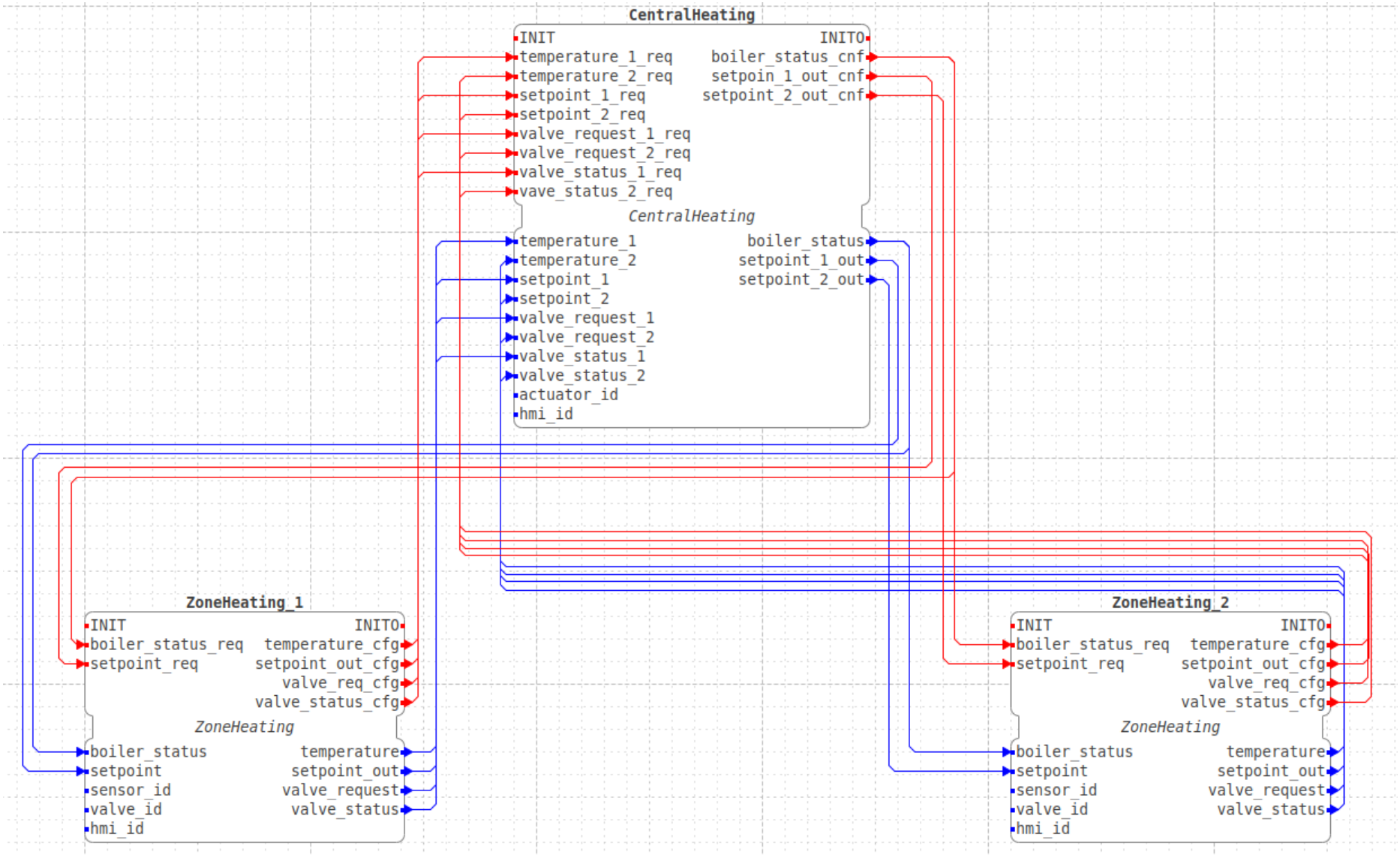


Figure: IEC 61499 model of the system.

Distributed application

- the application can be distributed - let's suppose two hosts
- we need to specify mapping components to hosts and add communication and initialization components (service interface blocks) - see next slides

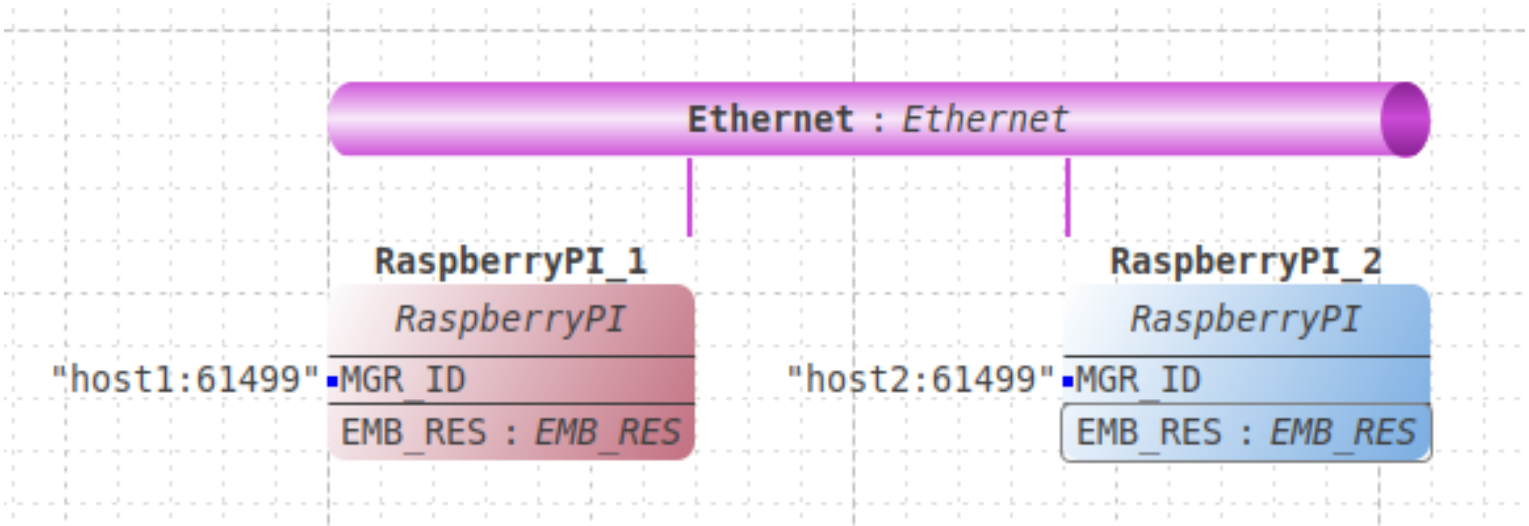


Figure: IEC 61499 distributed system model. One RPi hosts Central controller and Zone 1 controller, The second one hosts Zone 2 controller.

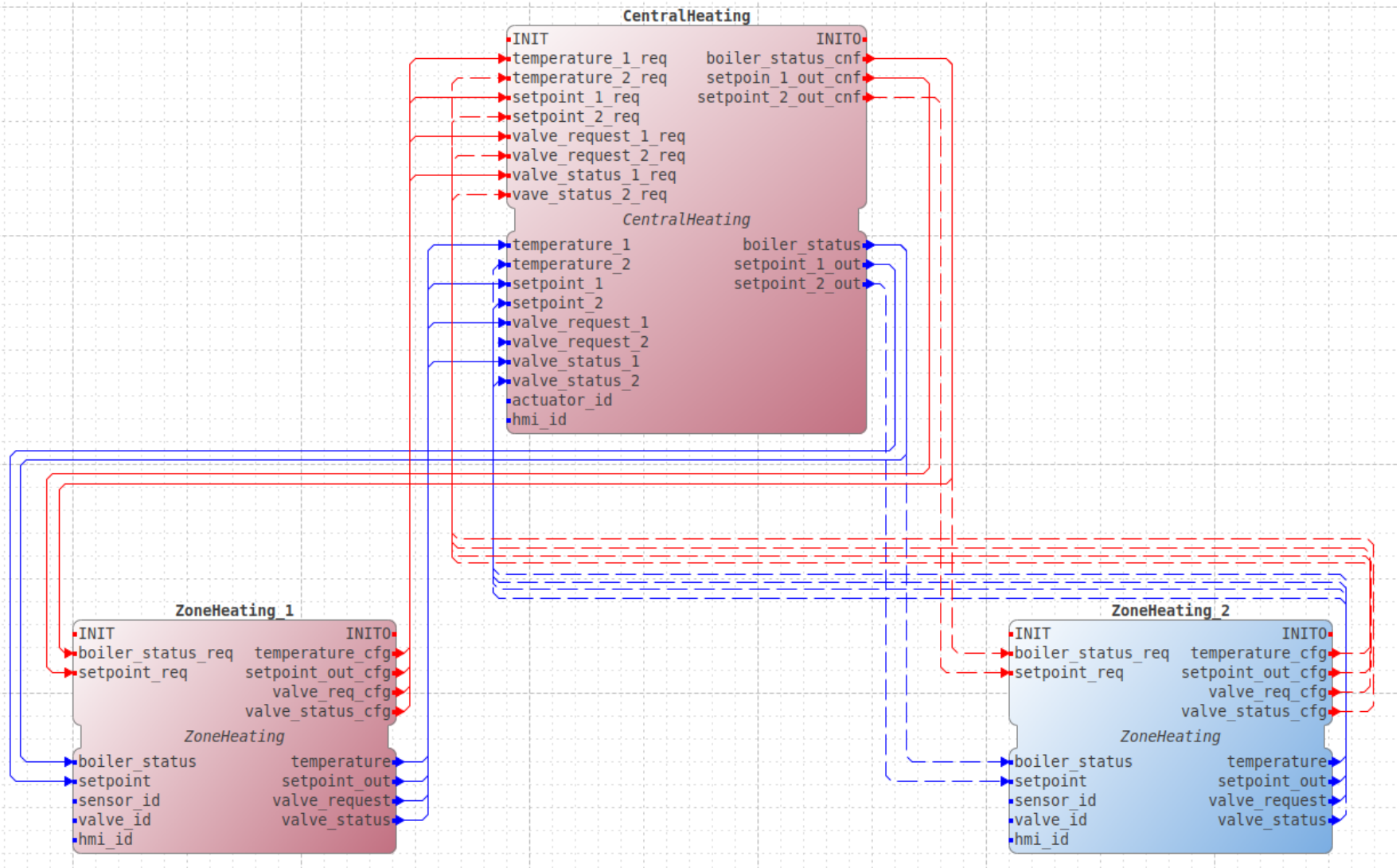


Figure: IEC 61499 model of the system - distributed version. Different component colors mean that they are mapped to different nodes of the distributed system. Dotted lines model network connections.

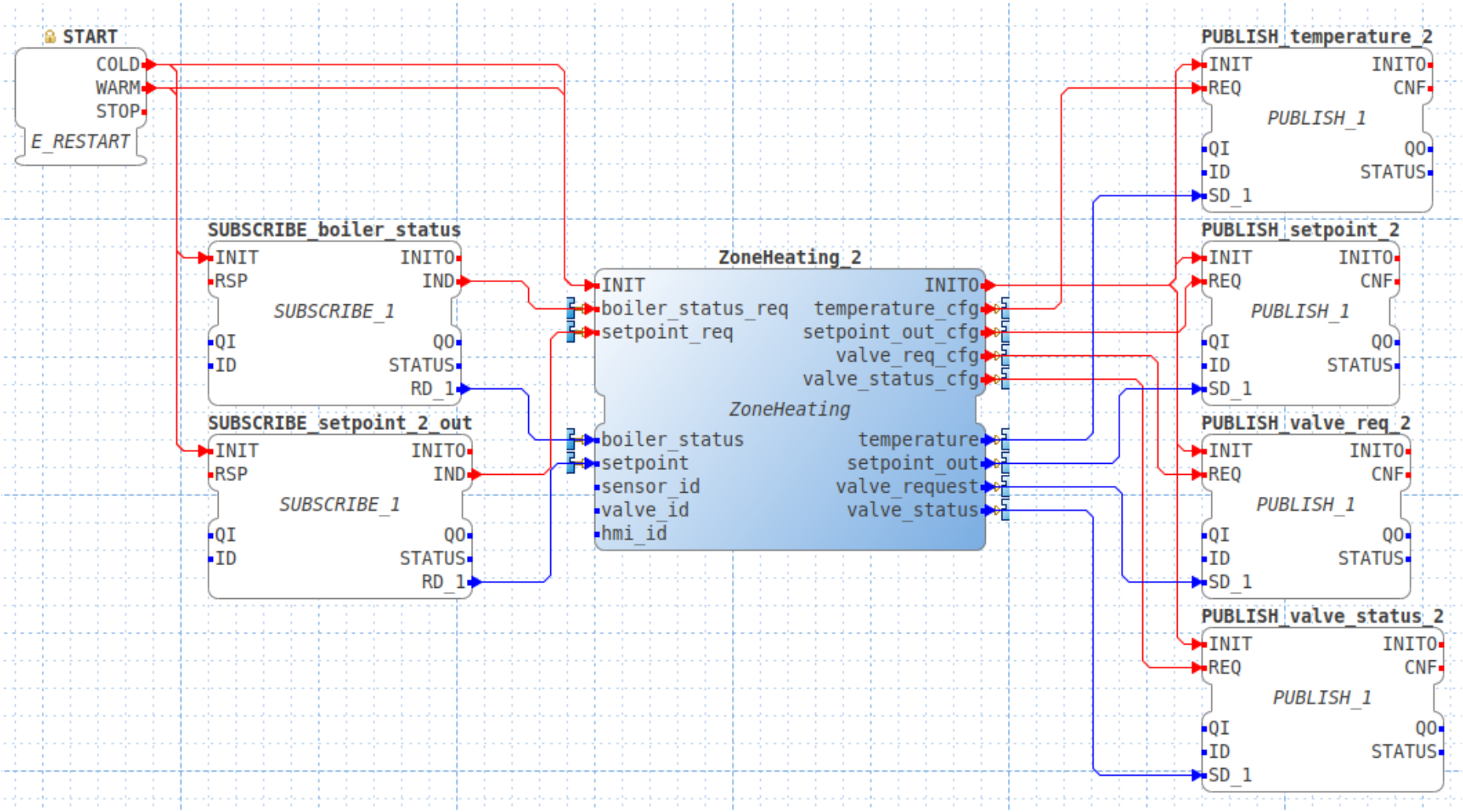


Figure: Zone 2 controller with and interface to central controller is deployed on Raspberry Pi 2.

We have presented

- the transition from a conceptual model through a platform-independent model to platform-specific models in several environments usable for control applications
- the initial modeling tool was the UML language, which is followed by the DEVS formalism
- the the transition from DEVS to three implementation environments
- the approach supposes an existence of library of well-defined atomic components (domain-specific, e.g., smart-home)

Contribution

- the continuity of the DEVS model in all considered implementation environments
- the transformation of the DEVS model into the target environment is based on relatively simple rules, and the original structured DEVS model is retained
- a uniform modeling methodology for heterogeneous implementation environments, e.g., 4diac/FORTE, PowerDEVS, or Node-RED

Future work

- combination with existing techniques for transforming atomic components specified by state charts, Petri nets, etc.
- validation in more complex environments, e.g., with more demanding real-time requirements, or in combination with, e.g., Robot operating system (ROS)

Thank You For Your Attention !