ACTIFSOURCE GMBH

Actifsource CIP

Language Workbench Challenge 2012

This paper shows the Actifsource CIP solution of the language workbench challenge 2012 assignment.
Keywords: Domain-specific language, Model-driven development

Contents

Assignment

The assignment of the language workbench challenge 2012 can be found at http://www.languageworkbenches.net/index.php?title=LWC_2012#The_assignment. The goal is to show the capabilities of the tool by designing a meta-model for piping and instrumentation chart, and to implement an example including control logic design.

Introduction: The Actifsource CIP tool

This article describes, how the problem of the assignment is solved using the CIP Edition of Actifsource.

Actifsource is a modeling tool and code generator that is integrated into the Eclipse IDE. For general information of the tool, visit our webpage www.actifsource.com. For information, how concrete tasks are solved, read the tutorials provided or look in our contribution to the Language Workbench Challenge 2011.

The CIP Edition is still under development and will be released this summer. It contains the CIP meta-model, several specific editors and templates to generate code for embedded systems. CIP stands for Communicating Interacting Processes – a methodology developed in the 90's at the ETH Zürich by Prof. Dr.Hugo Fierz for the design of complex embedded systems. The systems developed are characterized by their reliability, robustness and flexibility. And it has been shown, that over the whole software life-cycle, the expenses for development and maintenance can be reduced by 50% compared to conventional embedded software development.

Actifsource CIP generates C,C++, or Java code. In the past, those languages were not supported by most PLC tool chains – except B&R Automation Studio. Beckhoff TwinCAT – the tool recommended by the assignment – will support them in the new version 3 that will be released soon (in 2012).

We focused on the design of the control logic of embedded system, not on the stack that is necessary to get the system running on the PLC. The meta-model of the CIP methodology is fully accessible to the programmer, so the system can easily be extended with custom templates that generate the required binding code and mocks for simulation and visualization.

The CIP methodology

The general aim of the CIP methodology is to structure the embedded system the same way as the physical system and to connect the physical engines with their embedded counterparts by message channels.

According to the CIP methodology, embedded systems are designed as reactive components: They consist of a number of cooperating extended state machines called Processes. External events are communicated as event messages to the processes, and the Processes themselves communicate via messages to the extern.

Internally, processes can also communicate synchronously if they belong to the same process group called Cluster. For every Cluster, the incoming event messages are processed sequentially. There are different types of synchronous communication: Pulse is the equivalent to a message. The information about the current states of the other processes in the cluster by the definition of so called Gates, that evaluate to true for certain state combinations. And further state inspection is performed via Inquiries.
The transitions of the state machines can be exchanged completely by defining different Modes. The current mode is defined similarly to Gates, by mapping state combinations to a process mode.

1. DSL for Piping and Instrumentation

As explained before, we will focus on the functionality of the system and show how the problem is solved using the CIP methodology. CIP already defines its own DSL:
There are processes, grouped by clusters, and message channels, grouped by channel categories.

The model of the control behavior differs notably from the model of a piping and instrumentation drawing. And, besides some correlation of the elements, there is no obvious method to automatically transform a P&I drawing into a control program.

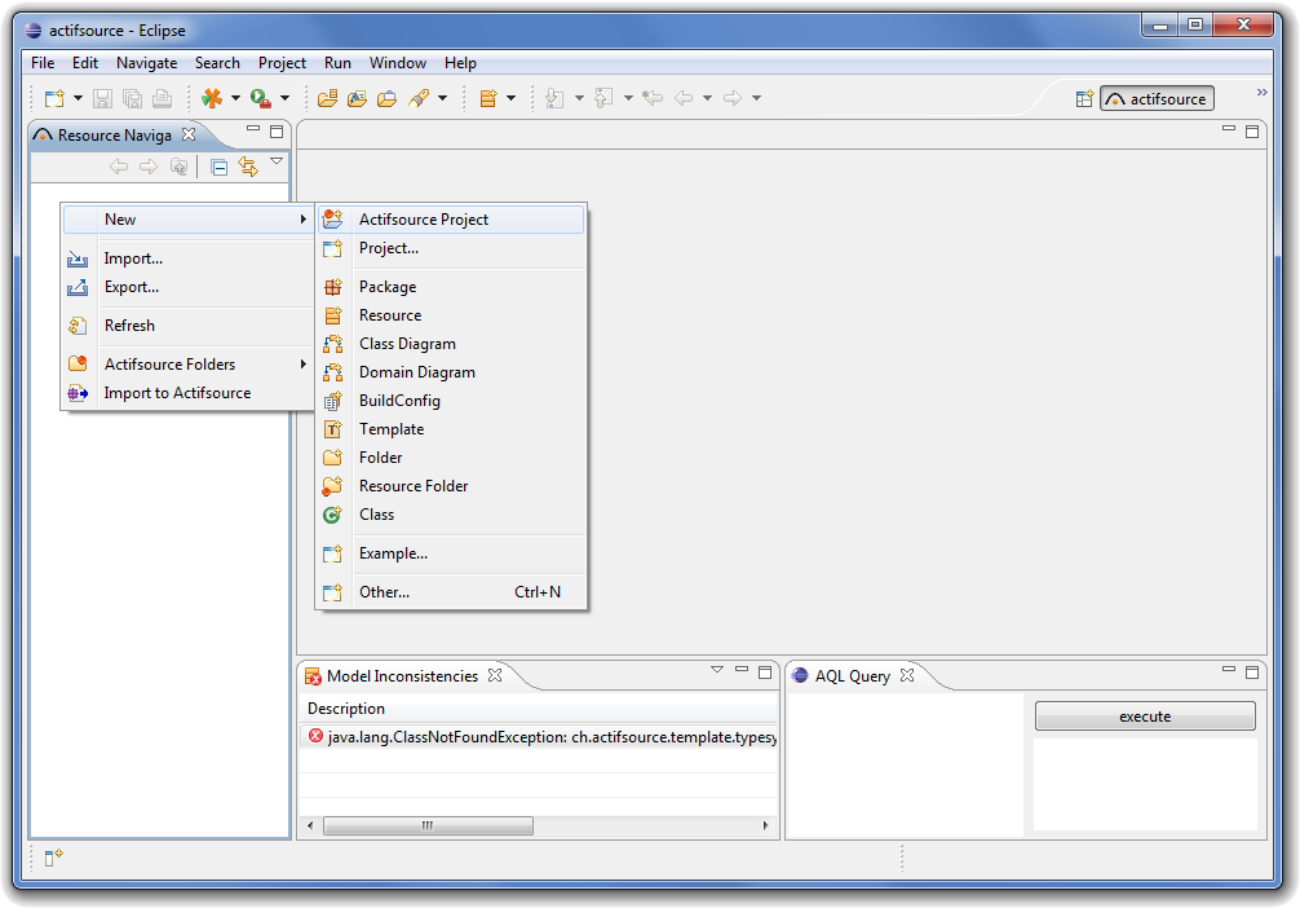
So, in order to get a system working properly, the order must be, to define and model the function of the system first, and then in a second step, implement that system using the P&I elements that meet the requirements.

We decided to focus on the control behavior of the system, not the actual topology. If you are interested in designing a DSL for P&I drawings, look in our contribution of last year's LWC or in the tutorials.

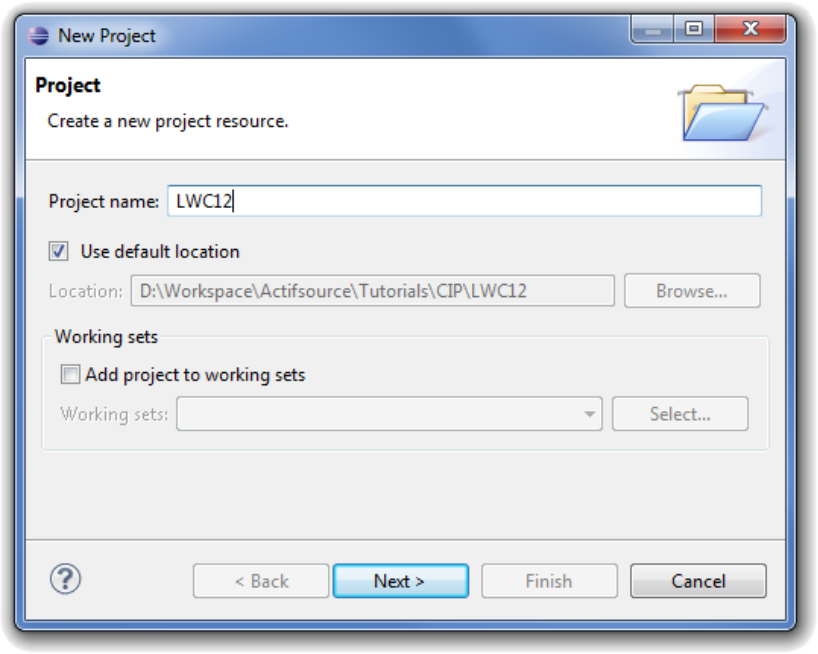
2. Instantiation of DSL elements

Actifsource CIP lets you instantiate the elements in a graphical way.

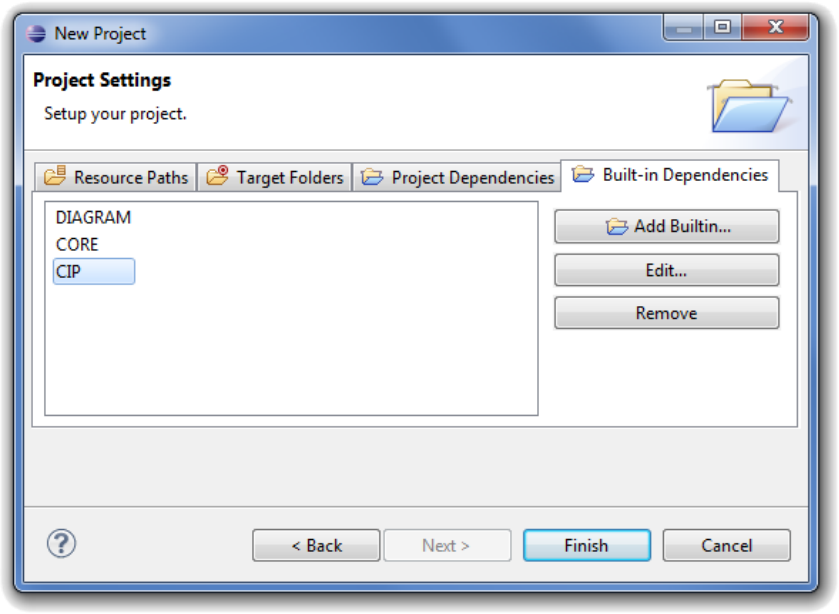
Let's start with a new actifsource Project:



Choose a name for the project, and optionally a specific project location.

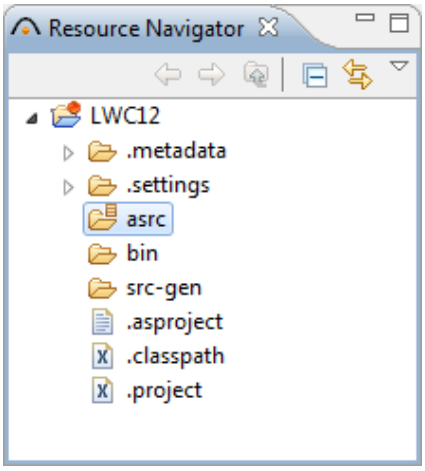


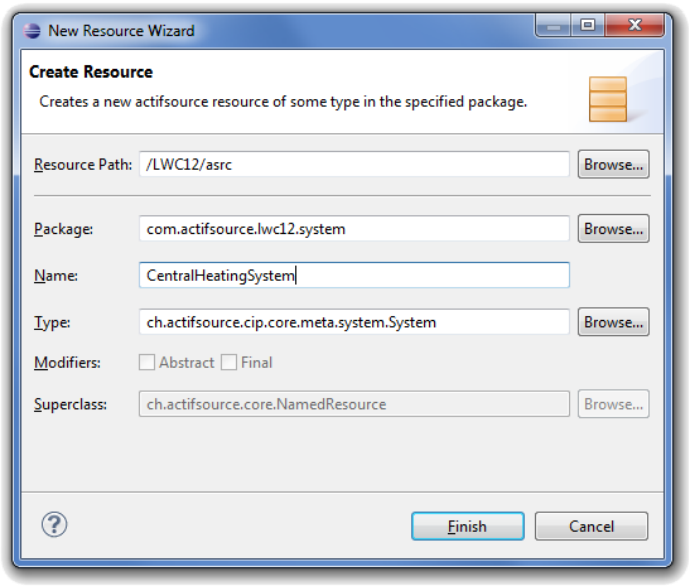
Make sure there is the built-in dependency to the CIP extension set.

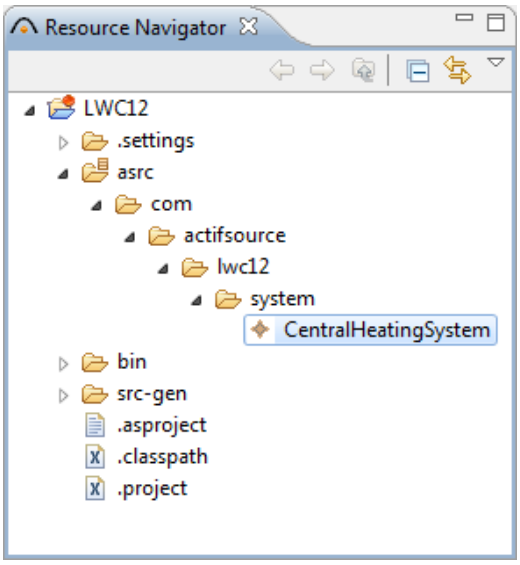


Press *Finish*.

Now, create a new CIP System in the *asrc* Directory.

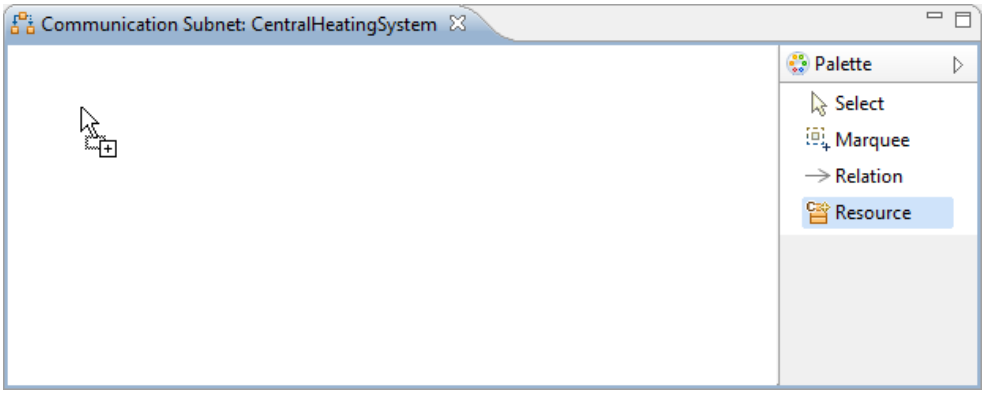




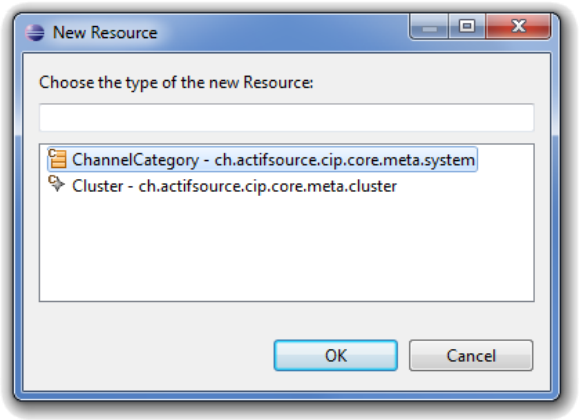


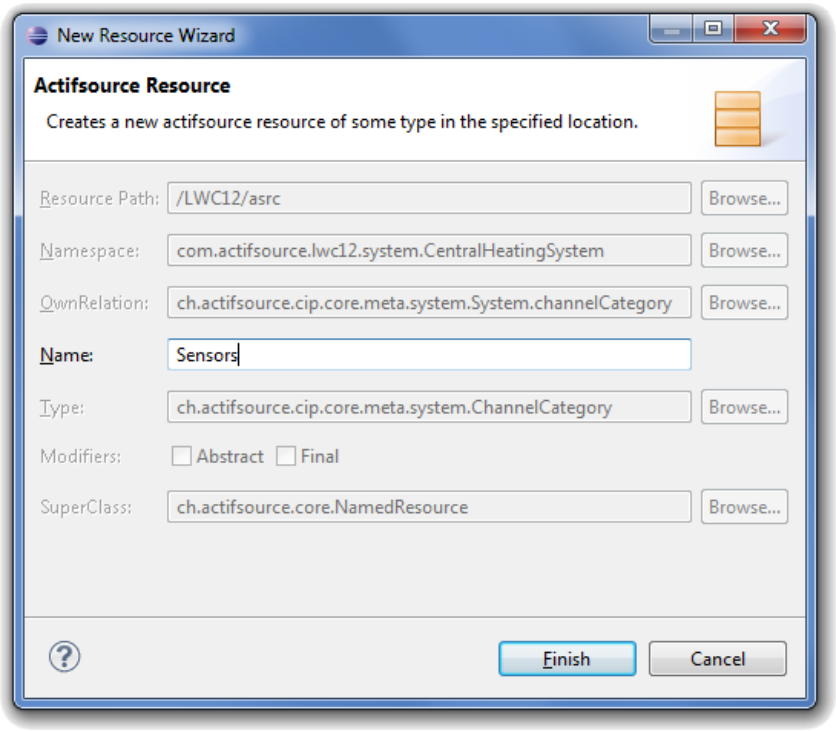
Open a new Communication Subnet in the Domain Diagram Editor: Processes and Channels are instantiated in the Domain Diagram Editor using the Resource Tool.

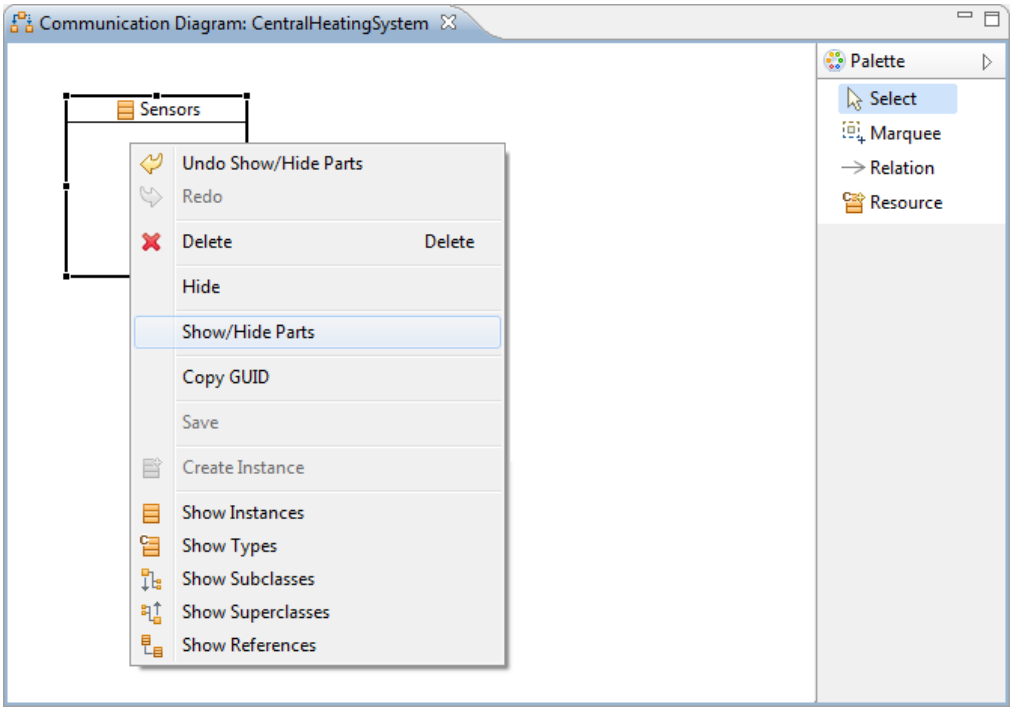
First, let us create a new ChannelCategory **Sensors** – a container, where we will create an InputChannel for every Sensor in the model.



Select the Resource Tool from the Palette and click into the diagram area. You are asked for the Resource type and name.





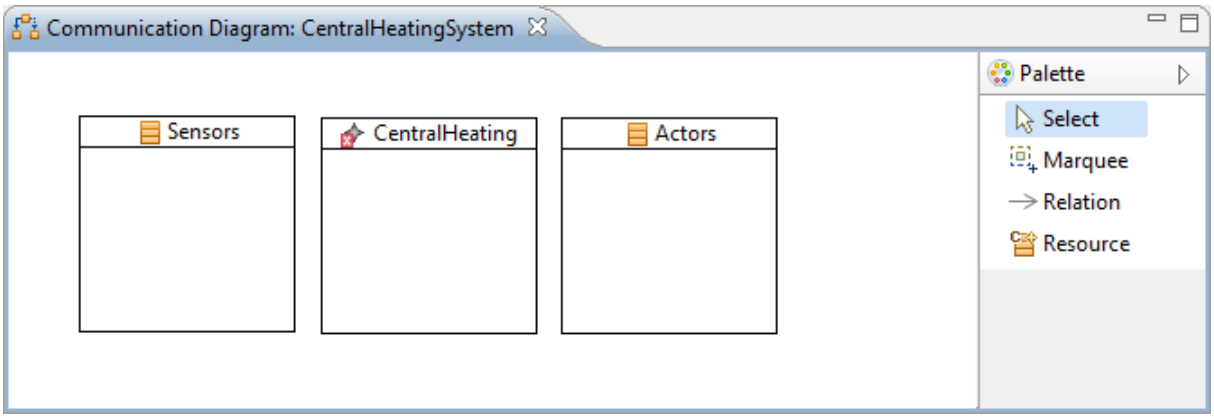


Second, we need a Cluster **CentralHeating** for the control logic.

This is done in the same way as the channel categories.

Third, the ChannelCategory **Actors** has to be created for the OutputChannels that deliver the commands to the extern.

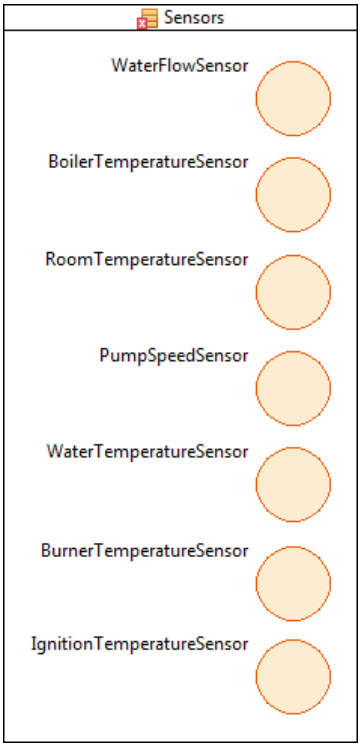
You will get the following system:



Input Channels for the Sensors

We represent the sensors of the P&I network as (stateless) InputChannels.

Select the Resource tool again and click into the **Sensors** ChannelCategory to create the representations.

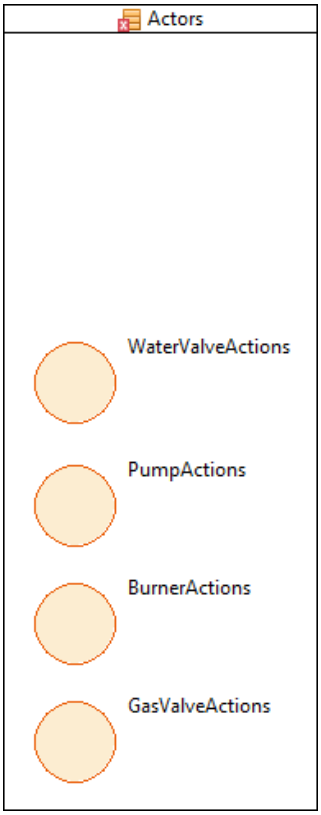


You are asked for names. Create a Channel for the WaterFlowSensor (F1), BoilerTemperatureSensor (T2), PumpSpeedSensor (S3), IgnitionTemperatureSensor (T4), BurnerTemperatureSensor (T5) WaterTemperatureSensor (T6) and there must be a RoomTemperatureSensor (T7).

Output Channels for the Actors

We can identify the following actors in the system: The WaterValve (V1), Pump (P1) Ignition, and the GasValve (V2).

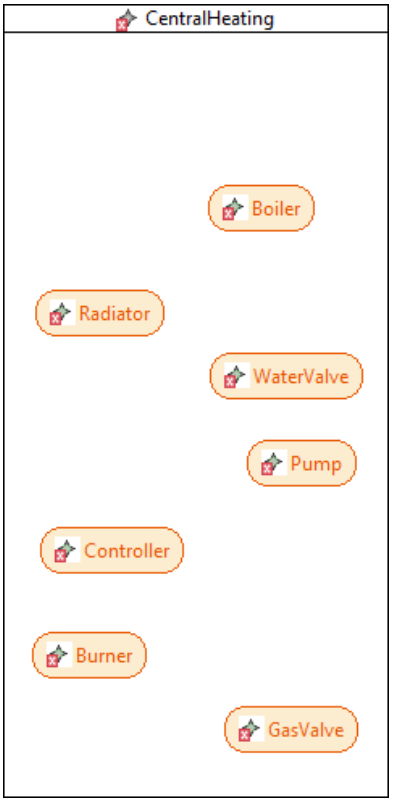
Create a Channel for every of those actors.



The CentralHeating Cluster

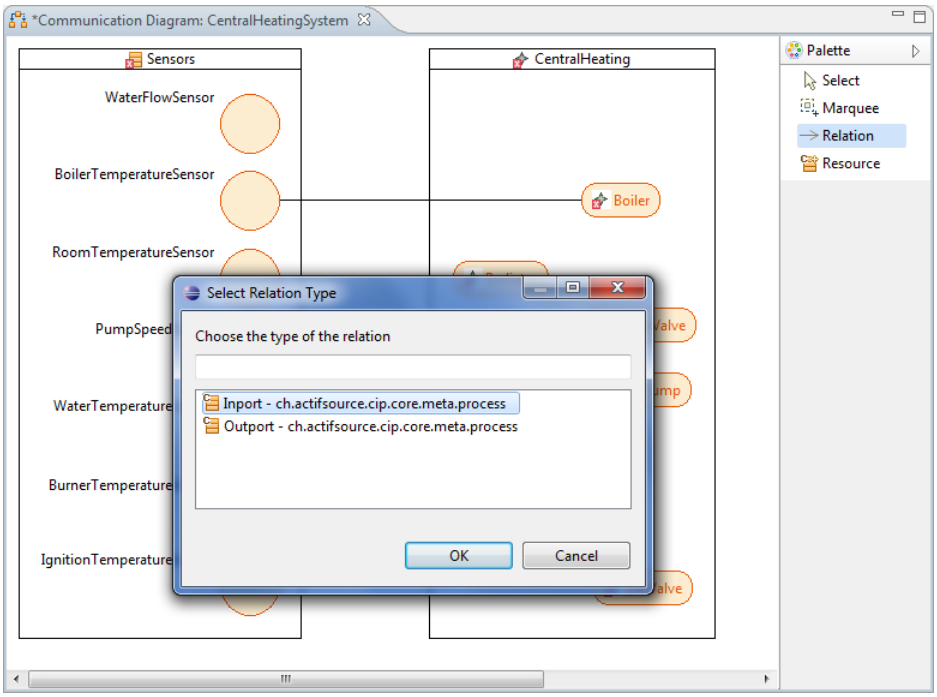
In our example, we identify the following Processes: Boiler and Radiator on the client side, Burner, Gas Valve, Pump and Water Valve on the server side. Additionally, we add a Contoller process that implements the behavior of the overall system, that cannot be distributed to a single process without customizing that process to this specific application.

We put all processes into the same cluster, meaning they communicate synchronously with each other.

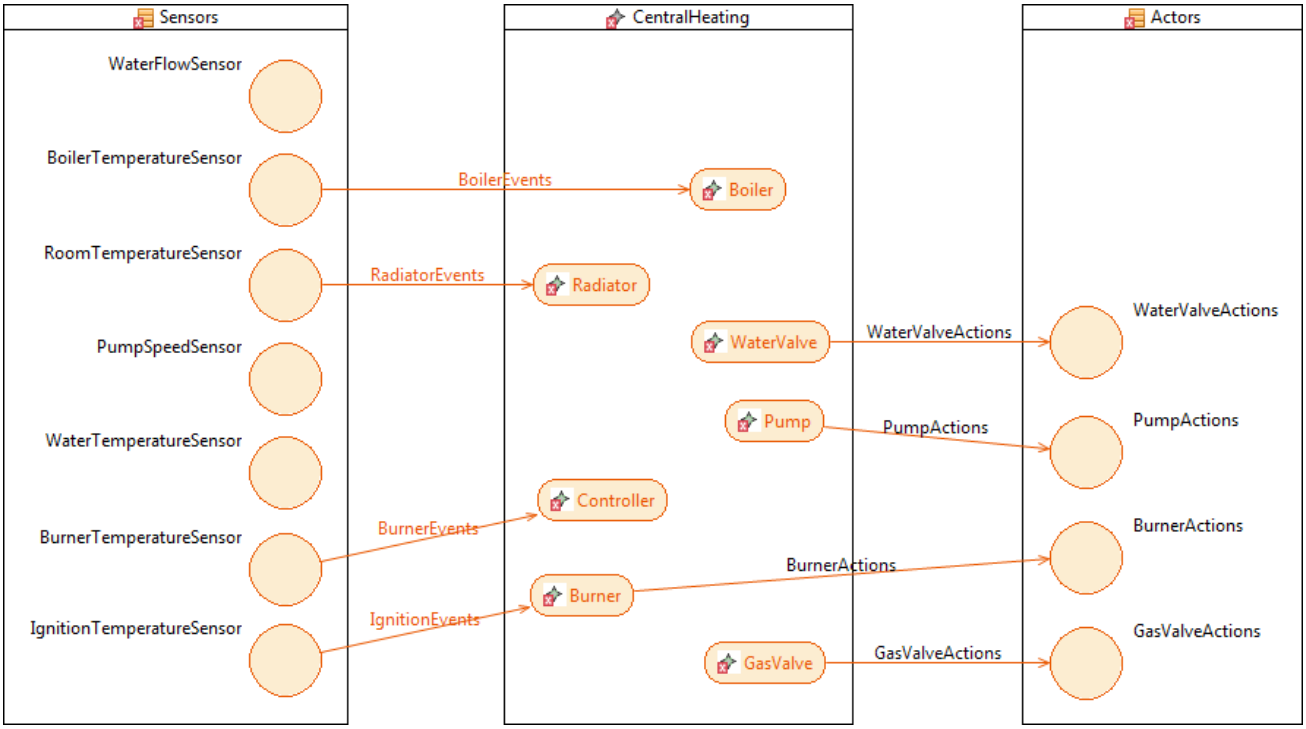


Communication

You can connect the Processes with the respective input or output channel by clicking on the Process and then on the Channel. You are asked if the created connection is an Inport or an Outport.



The resulting Communictation Diagram will look as follows:



3. Intuitive P&I network

The network designed as we did it reflects mainly its function and not its topology as the P&I network does. It is described in a graphical way by many diagrams, as the PulseCast Diagram, the Communication Diagram and the State Diagram.

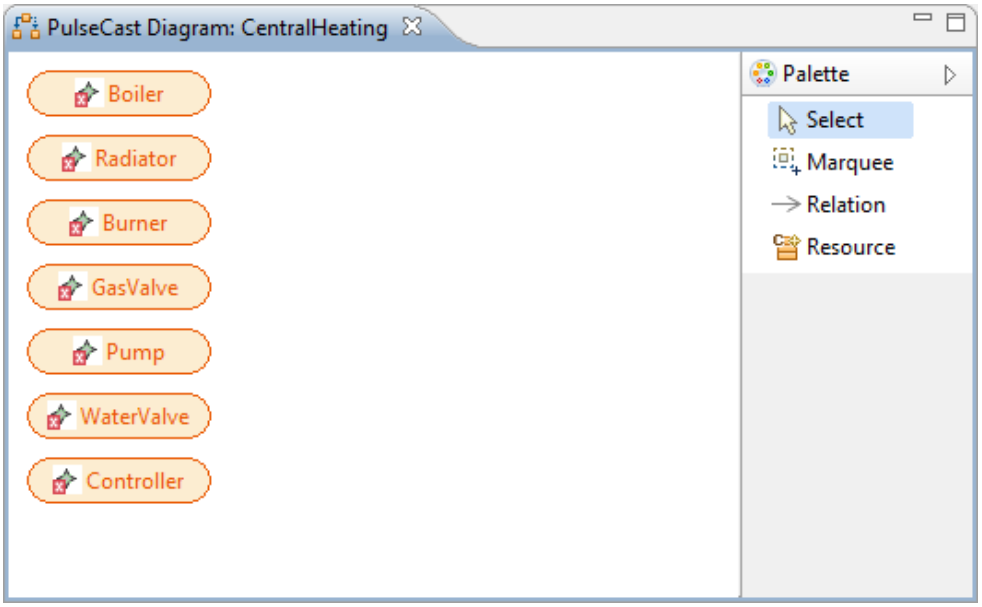
Model the interaction between the elements

If either Boiler or Room temperature is too low, heating is requested. This requires the water valve being open in the desired direction, the pump running and the burner heating.

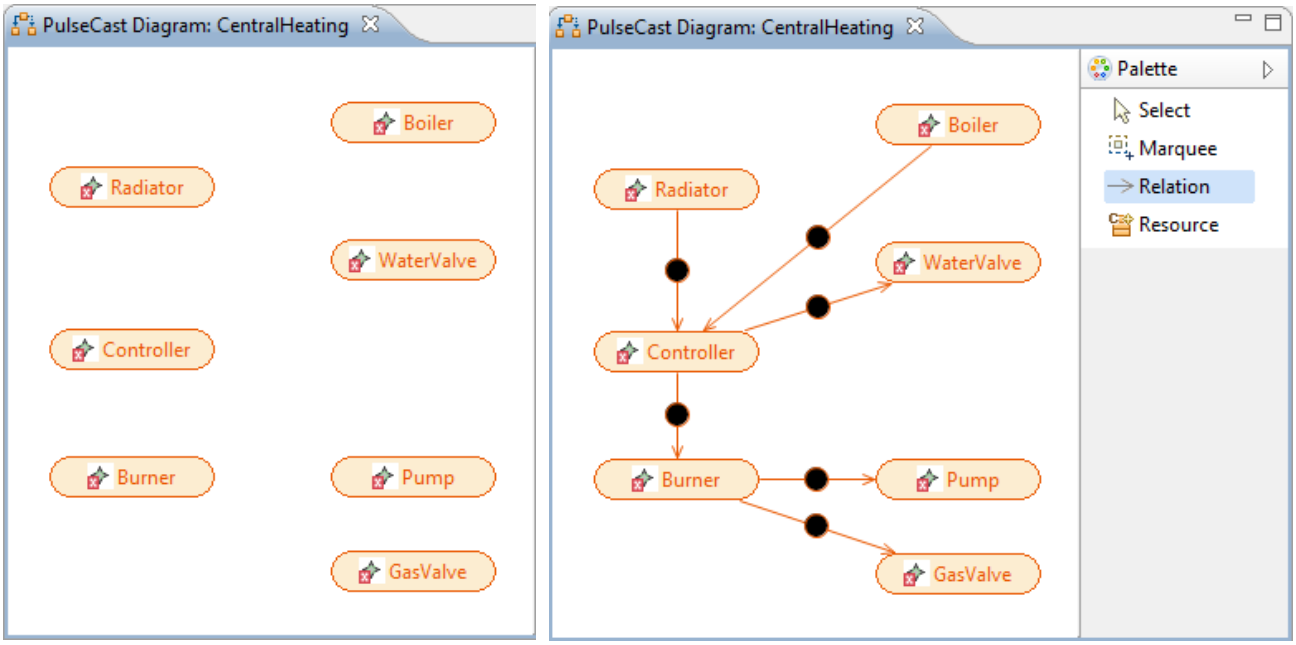
The heating strength has to be communicated from to the burner, so he can decrease burner heat if target temperature is in reach.

The communication inside the cluster

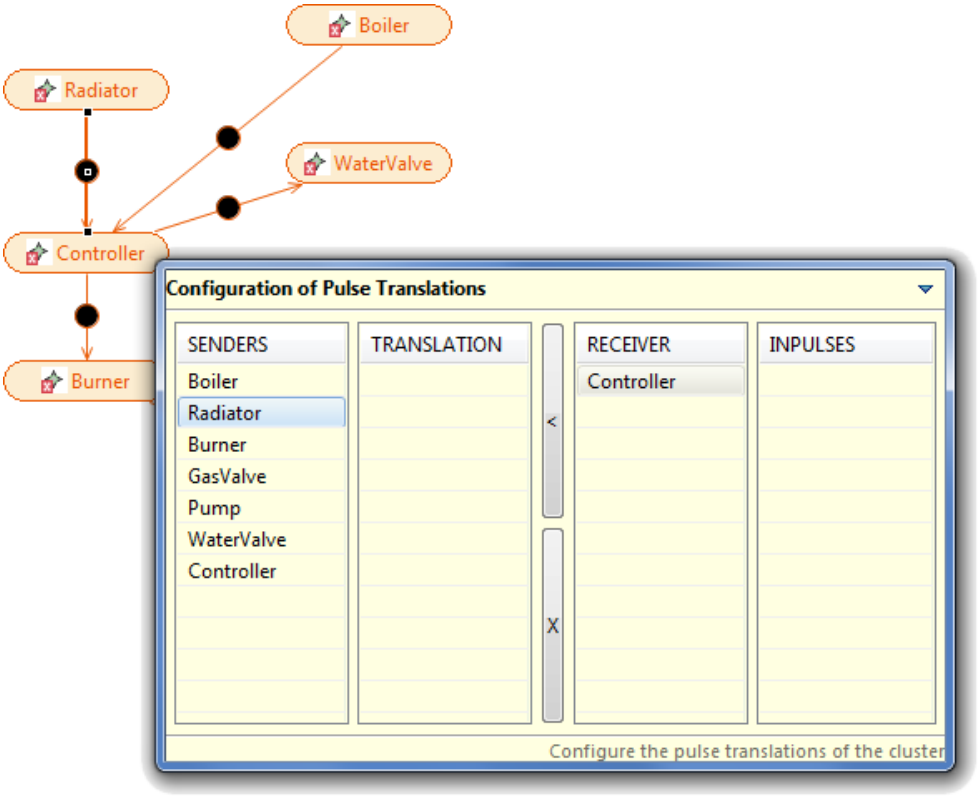
If you create a new PulseCast Diagram for the Cluster CentralHeating, all the we have created before initially appear in the diagram.



Order them using the select tool and, using the Relation tool, create the connections we need to get our Central Heating System working.

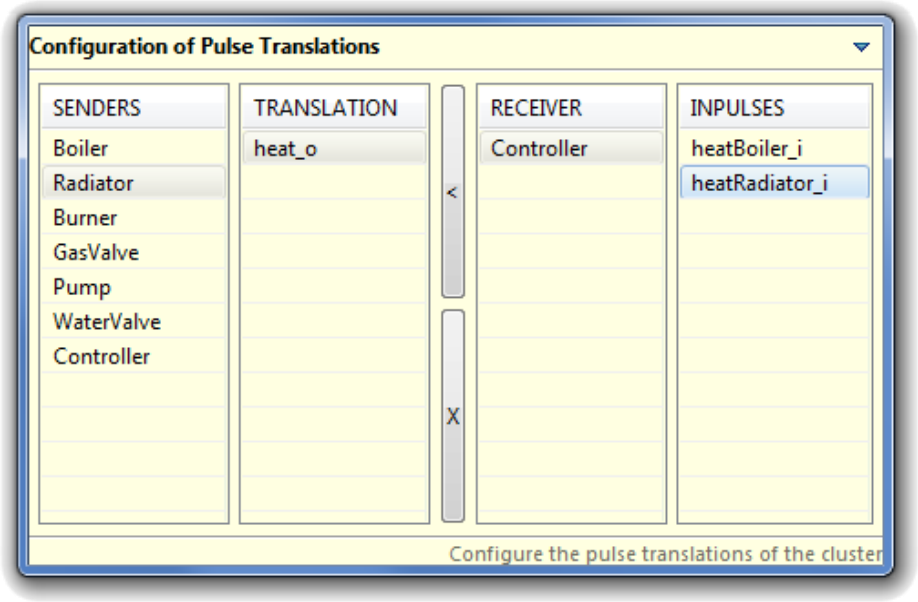


The connections are called Pulse Translations: A Pulse Translation connects an Outpulse of a Process to an Inpulse of another Process. You can open an Editor with a double-click on the bullet of a pulse cast.



Add the necessary Inpulses and Outpulses.

Define the Pulse Translation by assigning the Inpulse of the Receiver process to the respective Outpulse of the Sender process.



Custom display

For the actual physical implementation of the system with the described behavior, it would be possible to decorate the functional model by DSL elements for piping and instrumentation and even display them in the Domain Diagram editor with different shapes.

But, in contrary, it is not possible to create a P&I network and then infer a functional system out of the topology of the elements.

4. The control behavior

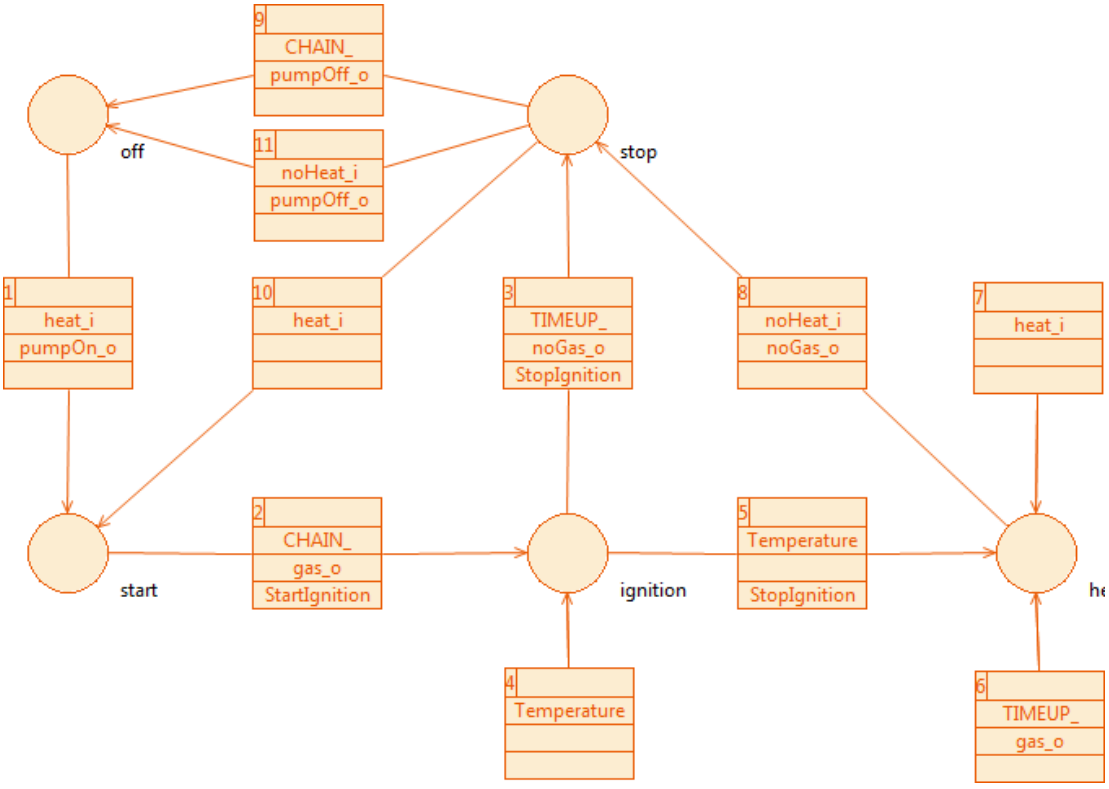
The system, in general, consists of two clients (Boiler and Radiator) and one server (Burner). The clients place their heat requests independently, and the burner heats up the medium (water), which then is transported to the clients. Either of the transport pipes can be interrupted by the valve.

The Burner process

Let's start with the state diagram of the Burner: The main states are **off** and **heating**. There are states on the way from off to on: **start** (having the pump running) and **ignition** and on the way back the **stop** state to swich the pump off at the end.

In each transition, at most one pulse can be sent. But it is possible to define chains of transitions sending a pulse in every transition.

For the normal case, the state machine of the Burner looks as shown below:

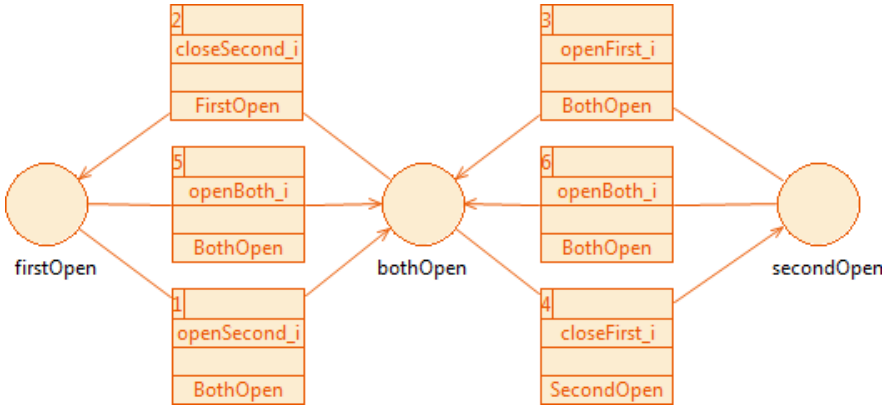


The Gas Valve process

The gas valve is an analog control valve. It has only a single state and a variable Flow denoting the current flow.

The Water Valve process

The water valve has three states: Flow to Boiler, Flow to Radiator and Flow to both.



The Pump process

The pump has the states on and off.

The Radiator process

The radiator has the states on and off.

We could add a config mode to set the variables.

The Boiler process

The boiler has the states on and off.

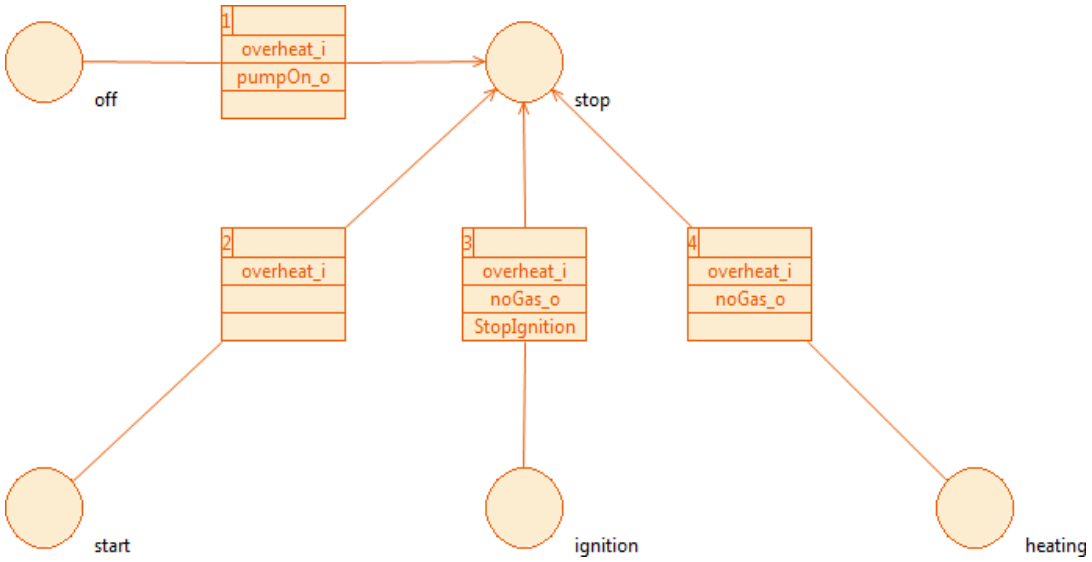
We could add a config mode to set the variables.

5. Constraint definitions

The CIP methodology does not contain a language to define constraints. However, there are several methods and mechanisms to implement additional checks for the system:

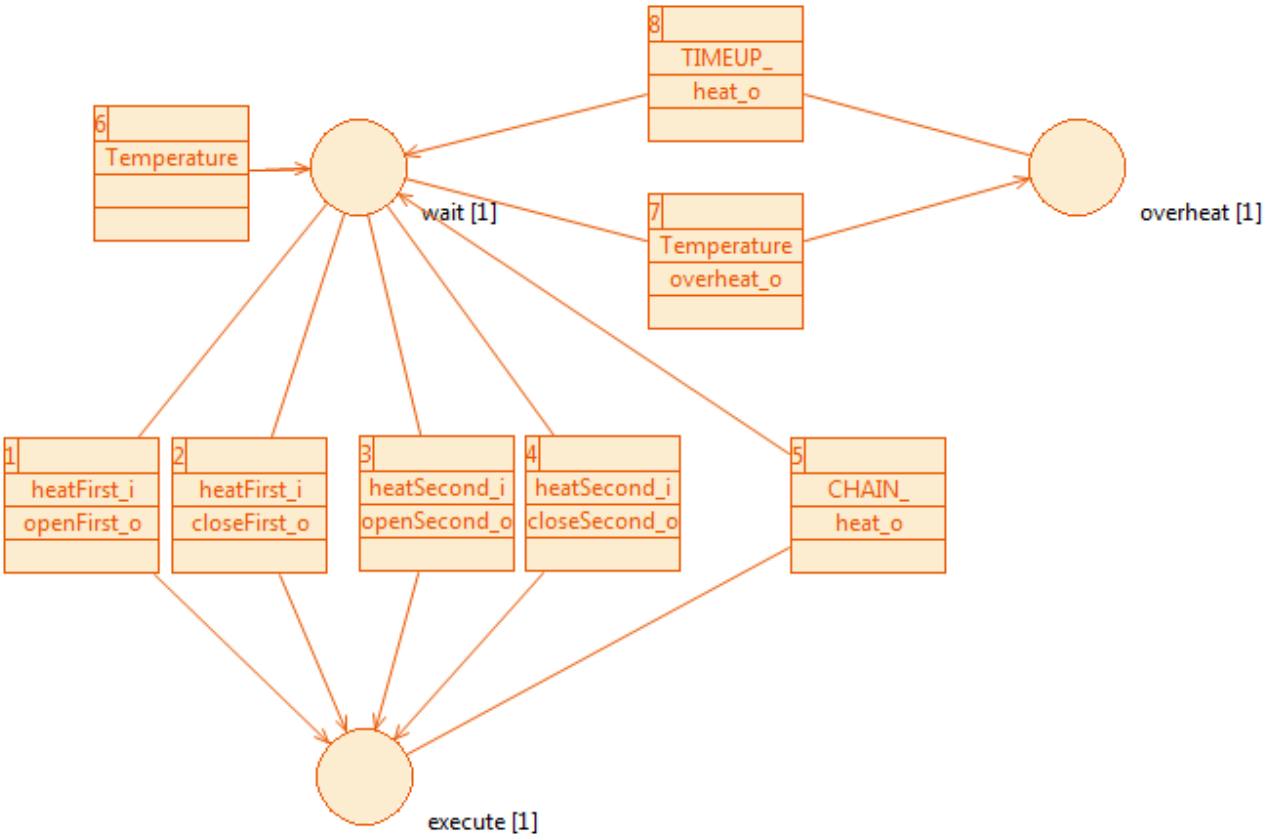
- So-called **Mode**s can be defined as combination of process states of other processes in the same cluster. This enables switching the mode for initialization, emergencies and recovery.
- **ErrorHandler**s are executed, whenever a asynchronous event occurs the process cannot handle in the current state.
- There are ways to query the state of another process (besides **Mode** control):
    - **Gate**s check for a certain set of state combinations in other states of the same cluster
    - **Inspection** allows to read a variable of an other process.

As an example, we will add an overheat mode to the burner, to prevent the water from overheating. The gas and ignition are switched off and the pump switched on from every state.



The overheat mode is controlled by the single state overheat in the Controller process. The mode changes the behavior of the process. But if there are actions to be done upon a mode switch, they must be modeled separately: Pulses sent in a transition, that causes a mode switch in the receiver process, is received already in the new mode.

In our simple example, on the state transition of the Controller process from **waiting** to **overheat**, it sends a pulse to the Burner process, which then makes sure the gas and ignition is off and the pump is on.



Requirements

Heat up the radiators if the actual room temperature is below the requested temperature.

Solution: The Radiator process compares the temperature from the room temperature sensor with the target temperature. If it is below, a heat request pulse is sent to the Controller process. If it changes to lie above the target temperature, a pulse is sent to stop heating.

Keep the hot water to configured set point (e.g. 95°C)

Solution: Same solution as for the radiator.

To increase efficiency, when heating up, gradually increase the burner heat to its max over a certain time period.

Solution: After ignition, the requested heating power is stored and the gas is flow is adapted periodically. The gas flow increase per period is restricted to a certain amount.

To increase efficiency, decrease the burner heat when the actual room temperature is in reach of the requested temperature.

The heating power the Boiler and Radiator process request from the Controller is calculated from the temperature difference. The gas flow the Burner requests is calculated from the requested heating power. So, the gas flow decreases when the requested heat tends to zero.

The pump must run, when the burner is on.

In the Burner process, in the transition from the **off** state a pulse is sent to the Pump process to switch on (whereas in the transitions to the **off** state the Pump is switched off).

The pump can be set to on (max speed) or off (no speed).

So the Pump process is modeled with only 2 states: **on** and **off**.

The mid-position valve can in three positions: all flow to boiler, all flow to radiators, or flow going to both boiler and radiators.

So the WaterValve process is modeled with 3 states, one for each position.

Be able to connect to a 'Smart Energy Management System' for:

- *Accepting set points of maximum temperature of water, burner, etc.*
- *Accepting specific user settings (user profile), like rate of heating up, holiday/non-holiday setting, etc.*
- *Visualization of the actual status of the central heating system (status of actuators, values of sensors)*

We did not implement a smart energy management system. It could be implemented as additional cluster containing a process that sends the target temperature settings to the processes in the CentralHeating cluster. The user profile database and holiday settings would also be modeled as processes in this cluster.

6. Stubs

The stub code is generated automatically.

7. Generate control code

The control code is generated by the tool. What is left to do is to connect the machine with the environment:

The singleton struct tIN_ImplementationUnit is used as interface to call when an external event like a temperature measurement is communicated.

```
struct tIN_ImplementationUnit

{
      void (*BoilerTemperatureSensor) (enum eMSG_BoilerTemperatureSensor , union
tDATA_BoilerTemperatureSensor * );
      void (*PumpSpeedSensor) (enum eMSG_PumpSpeedSensor , union
tDATA_PumpSpeedSensor * );
      void (*IgnitionTemperatureSensor) (enum eMSG_IgnitionTemperatureSensor ,
union tDATA_IgnitionTemperatureSensor * );
```

```
      void (*WaterTemperatureSensor) (enum eMSG_WaterTemperatureSensor , union
tDATA_WaterTemperatureSensor * );
      void (*RoomTemperature) (enum eMSG_RoomTemperature , union
tDATA_RoomTemperature * );
      void (*EINPUT_) (enum eCHAN_ImplementationUnit, enum eIN_ERR_, int);
};
```

The singleton struct tOUT_ImplementationUnit interface is invoked, when an action must be propagated to the extern.

```
/* Output Interface Type */

struct tOUT_ImplementationUnit
{
      void (*IgnitionActor) (enum eMSG_IgnitionActor  );
      void (*PumpActor) (enum eMSG_PumpActor  );
      void (*WaterValveActor) (enum eMSG_WaterValveActor , union
tDATA_WaterValveActor * );
      void (*GasValveActor) (enum eMSG_GasValveActor , union tDATA_GasValveActor
* );
};
```

The unit initialization function `int fINIT_ImplementationUnit(void)` must set the function pointers and implement functions initiating the proper actions.

Summary

The Actifsource tool in its CIP Edition let us design Embedded Systems graphically in straight-forward manner using the CIP methodology, and it produces runnable C/C++ code. The CIP meta-model is accessible to the user, so it can add custom templates that generate arbitrary code based on the same model.