Accélérateur de la transformation numérique

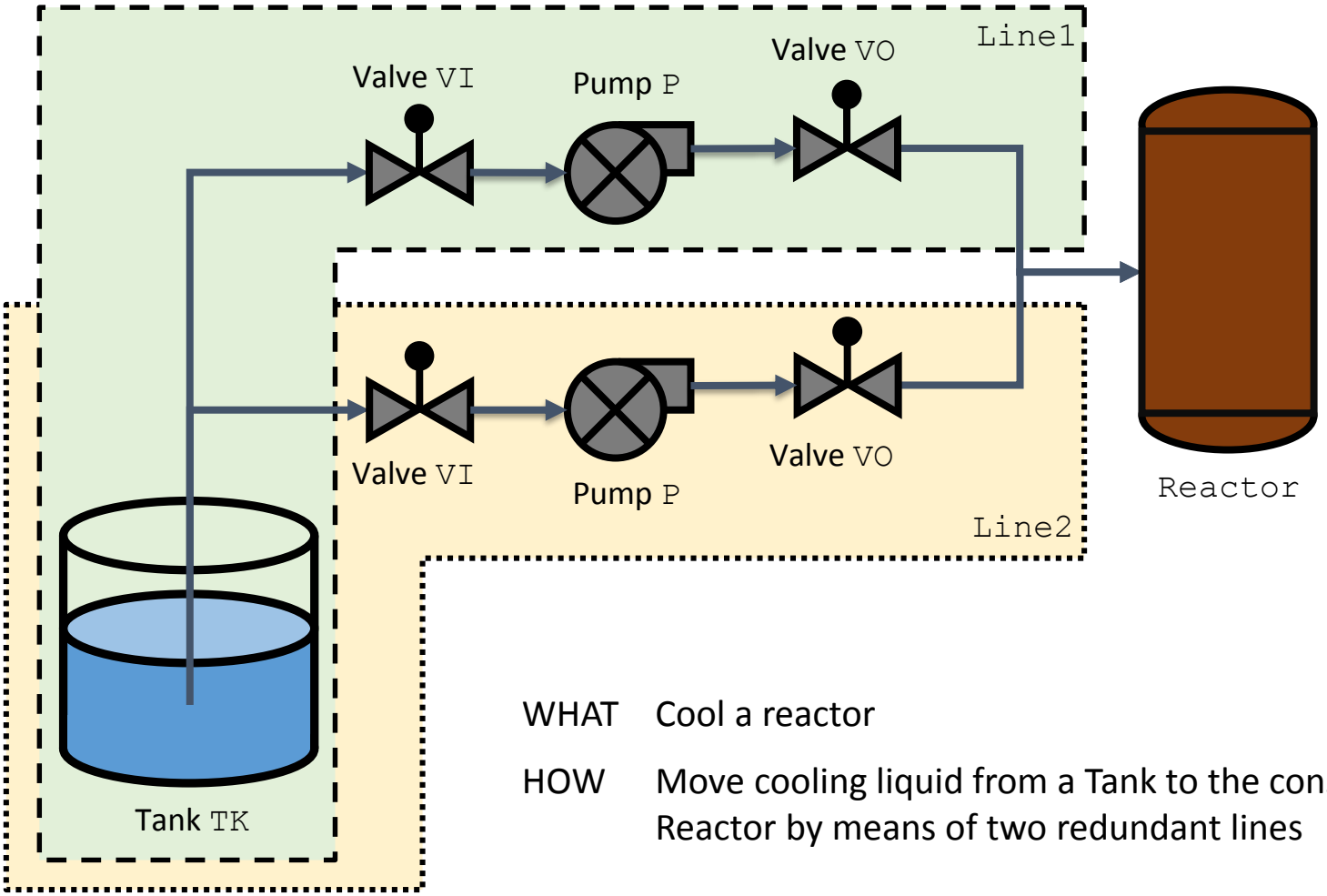
OpenAltaRica - Example

A (Simple) Cooling System

www.irt-systemx.fr

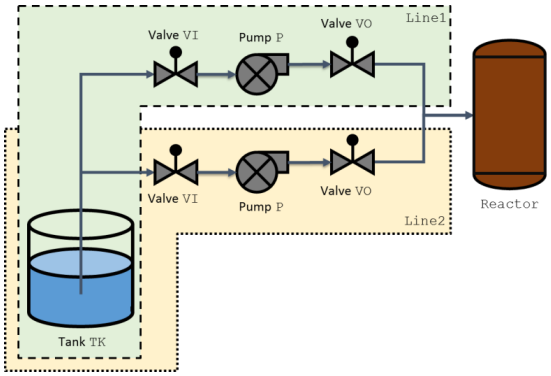
OpenAltaRica - Example

A (Simple) Cooling System

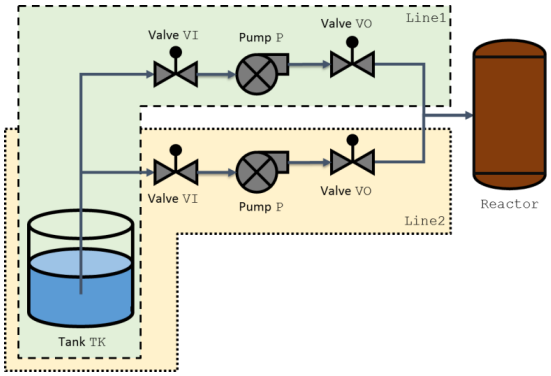


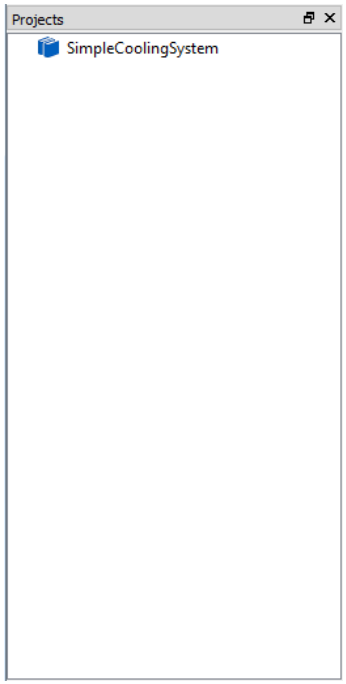
WHAT Cool a reactor

HOW Move cooling liquid from a Tank to the considered Reactor by means of two redundant lines



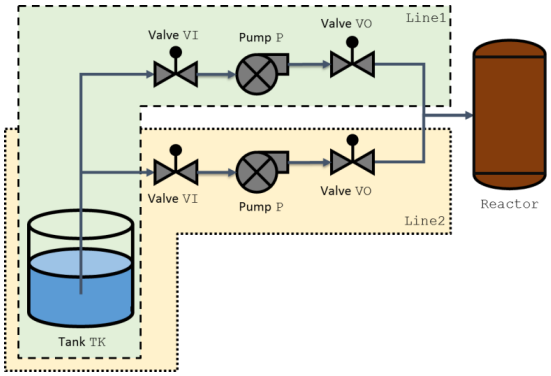
A. To design your model





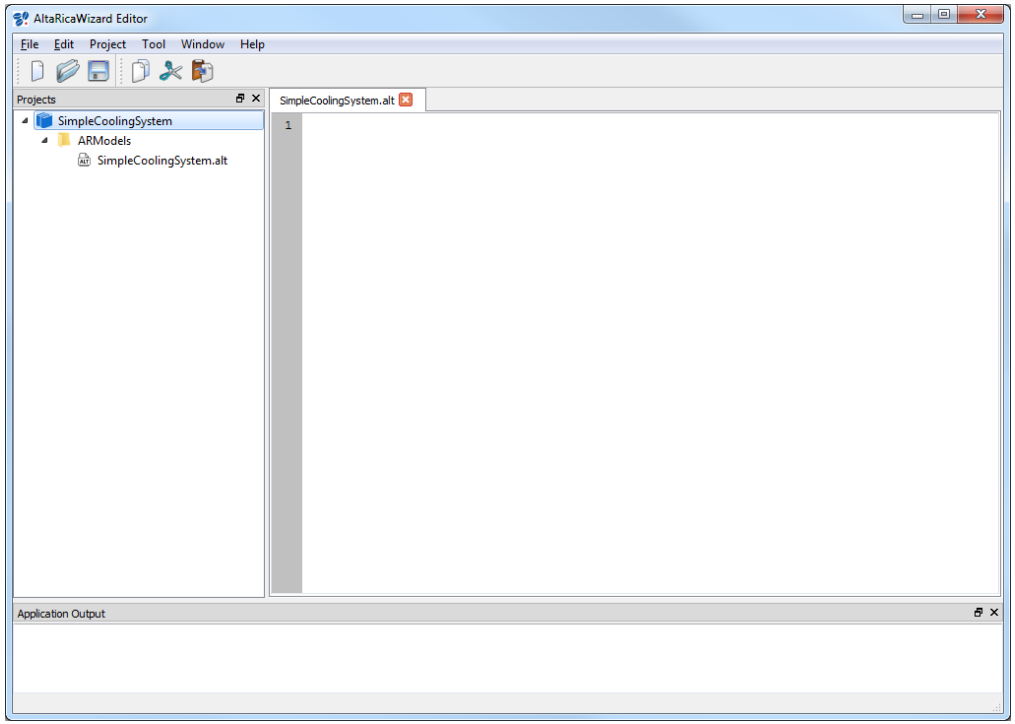
A. To design your model

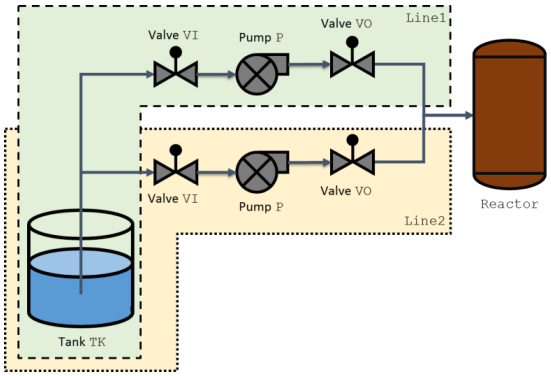
1. Create a new project into the AltaRicaWizard:
   a. ‘File’->‘New File or Project’->’New Project’;
   b. ‘SimpleCoolingSystem’ -> ‘OK’.



A. To design your model

2. Create a new AltaRica 3.0 model:
   a. Right-click on the project ‘SimpleCoolingSystem’;
   b. ‘Add new file to “SimpleCoolingSystem”’;
   c. You can create a new folder (e.g. ‘ARModels’);
   d. ‘SimpleCoolingSystem.alt’ -> ‘Save’.



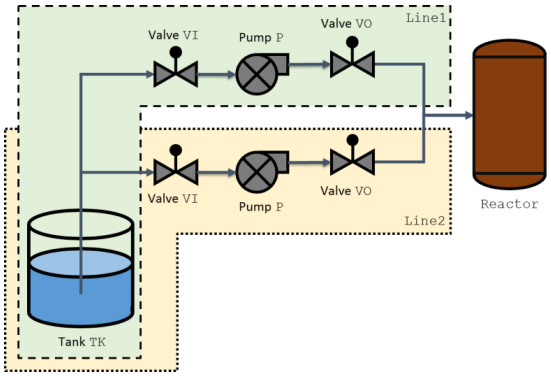


A. To design your model

3. Design your AltaRica 3.0 model.
   a. Different kinds of components/parts
      Generic: Valve, Pump, (Tank, Reactor)
      => use classes.
      Ad-Hoc: (Tank, Reactor), Line1, Line2, the system
      => use blocks.
   b. All actuator components are repairable
      => the same internal behavior.
   c. Process
      i. Define the internal repairable behavior into a generic element ‘class’;
      ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
      iii. Define other generic components into specific element ‘class’;
      iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

OpenAltaRica - Example

A (Simple) Cooling System

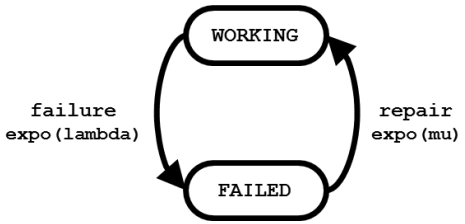


i. Define the internal repairable behavior into a generic element ‘class’;

ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;

iii. Define other generic components into a specific element ‘class’;

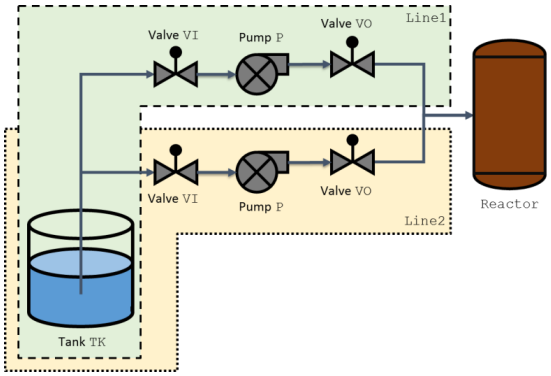
iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.




```
class RepairableComponent
    Boolean vsWorking (init = true);
    parameter Real pLambda = 1.0e-5;
    parameter Real pMu = 1.0e-2;
    event evFailure (delay = exponential(pLambda));
    event evRepair (delay = exponential(pMu));
    transition
        evFailure: vsWorking -> vsWorking := false;
        evRepair: not vsWorking -> vsWorking := true;
end
```

OpenAltaRica - Example

A (Simple) Cooling System



i. Define the internal repairable behavior into a generic element ‘class’;

ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;

iii. Define other generic components into a specific element ‘class’;

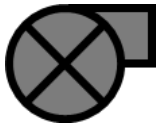
iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

```
class Pump
    extends RepairableComponent;
    Boolean vfInFlow, vfOutFlow (reset = false);
    assertion
        vfOutFlow := if vsWorking then vfInFlow else false;
end
```

```
class Valve
    extends RepairableComponent (pLambda = 1.0e-4);
    Boolean vfLeftFlow, vfRightFlow (reset = false);
    assertion
        if vsWorking then vfLeftFlow :=: vfRightFlow;
end
```

OpenAltaRica - Example

A (Simple) Cooling System



```
class Pump
    extends RepairableComponent;
    Boolean vfInFlow, vfOutFlow (reset = false);
    assertion
        vfOutFlow := if vsWorking then vfInFlow else false;
end
```

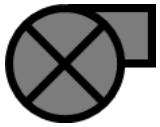
Inheritance of the class 'RepairableComponent'

- A keyword ('extends');
- The name of an inherited class ('RepairableComponent');
- (optional) redefinition of values.

```
class Valve
    extends RepairableComponent (pLambda = 1.0e-4);
    Boolean vfLeftFlow, vfRightFlow (reset = false);
    assertion
        if vsWorking then vfLeftFlow :=: vfRightFlow;
end
```

OpenAltaRica - Example

A (Simple) Cooling System



```
class Pump
    extends RepairableComponent;
    Boolean vfInFlow, vfOutFlow (reset = false);
    assertion
        vfOutFlow := if vsWorking then vfInFlow else false;
end
```

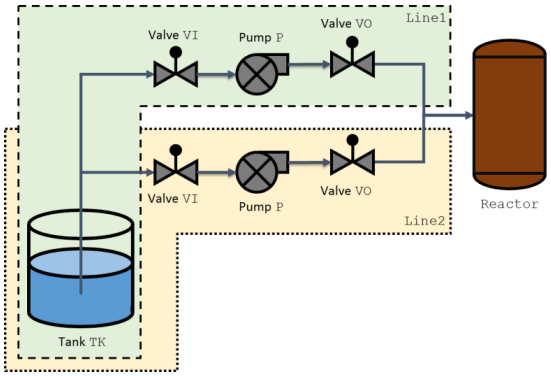
Acausal connection ':=:' between two flow variables

```
class Valve
    extends RepairableComponent (pLambda = 1.0e-4);
    Boolean vfLeftFlow, vfRightFlow (reset = false);
    assertion
        if vsWorking then vfLeftFlow :=: vfRightFlow;
end
```

REM: for only data-flow models (components, parts, etc.) do not use the acausal connection

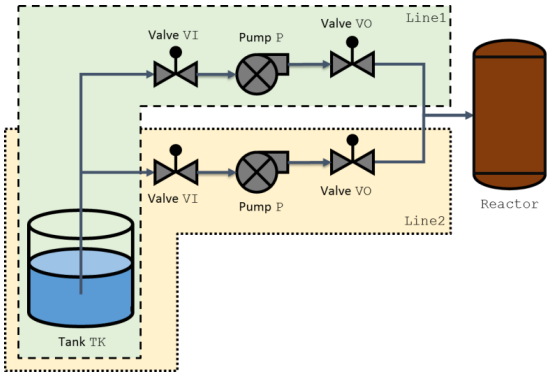
OpenAltaRica - Example

A (Simple) Cooling System



i. Define the internal repairable behavior into a generic element ‘class’;
ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
iii. Define other generic components into specific elements ‘class’;
iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

```
class Tank
    Boolean vsIsEmpty (init = false);
    Boolean vfOutFlow (reset = true);
    event evGetEmpty;
    transition
        evGetEmpty: not vsIsEmpty -> vsIsEmpty := true;
    assertion
        vfOutFlow := not vsIsEmpty;
end
```



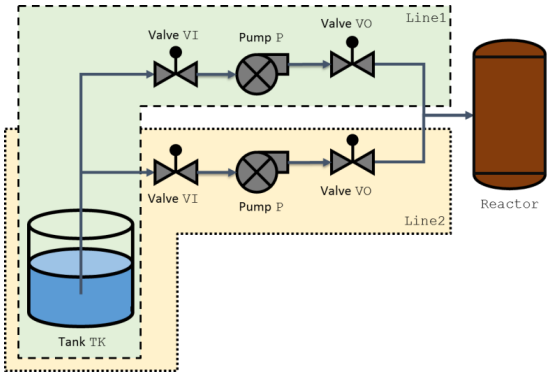
i. Define the internal repairable behavior into a generic element ‘class’;
ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
iii. Define other generic components into specific elements ‘class’;
iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

```
class Tank
    Boolean vsIsEmpty (init = false);
    Boolean vfOutFlow (reset = true);
    event evGetEmpty;
    transition
        evGetEmpty: not vsIsEmpty -> vsIsEmpty := true;
    assertion
        vfOutFlow := not vsIsEmpty;
end
```

No delay associated to this event.
If no defined at instantiation, set to Constant(1.0).

OpenAltaRica - Example

A (Simple) Cooling System



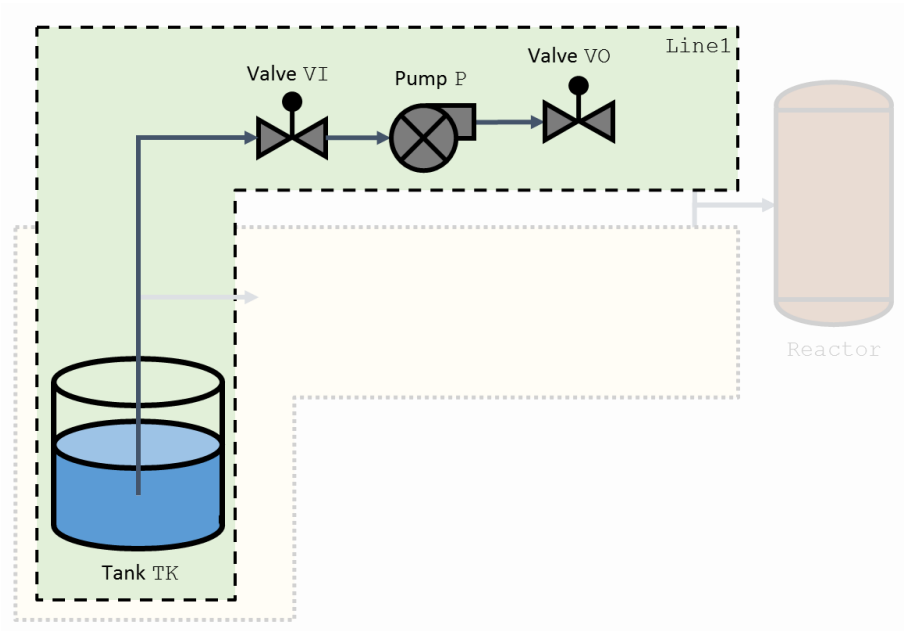
i. Define the internal repairable behavior into a generic element ‘class’;
ii. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
iii. Define other generic components into specific elements ‘class’;
iv. Build the model of the Cooling System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

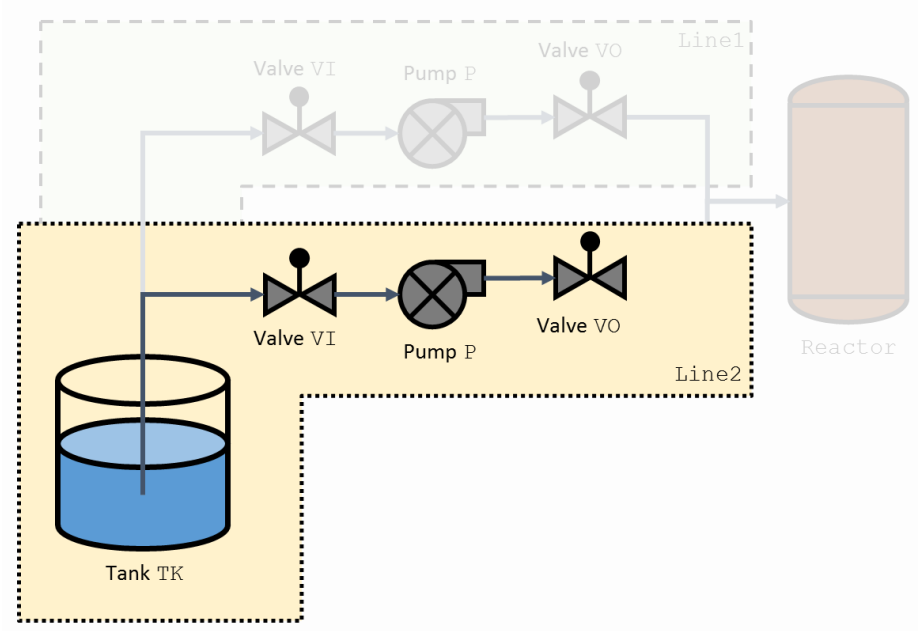
classes: Valve, Pump, Tank.

blocks: Reactor, Line1, Line2, the Cooling System.

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.



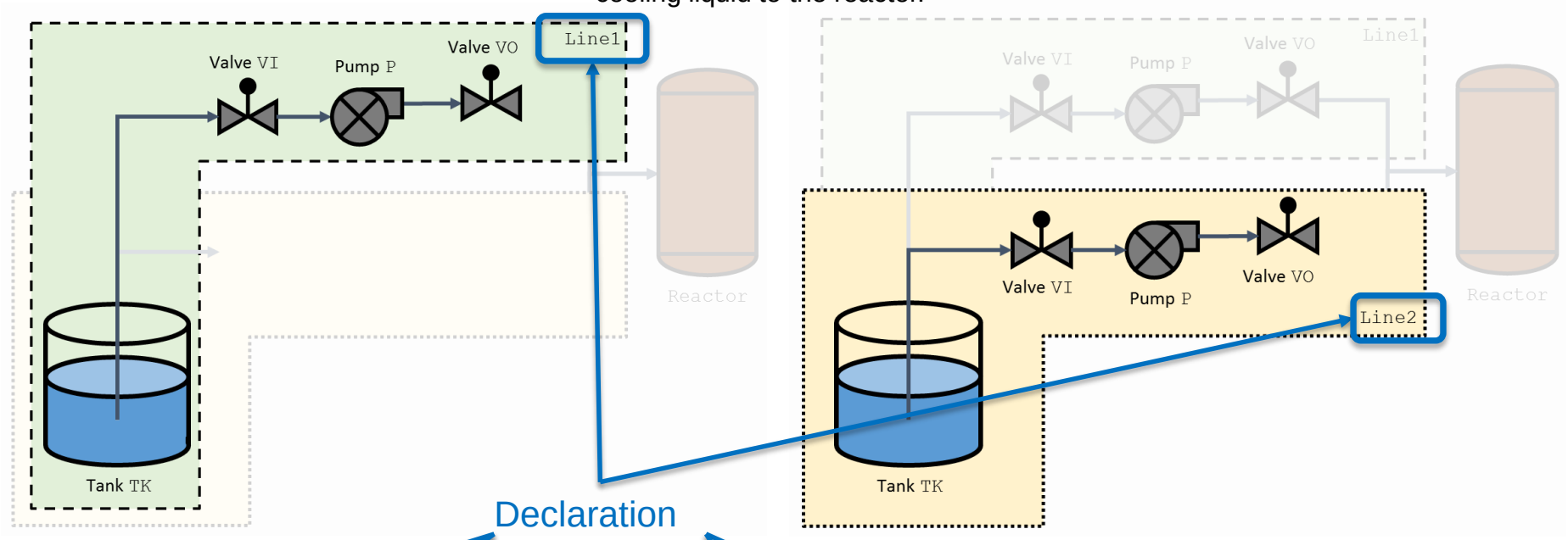



```
block Line1
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.



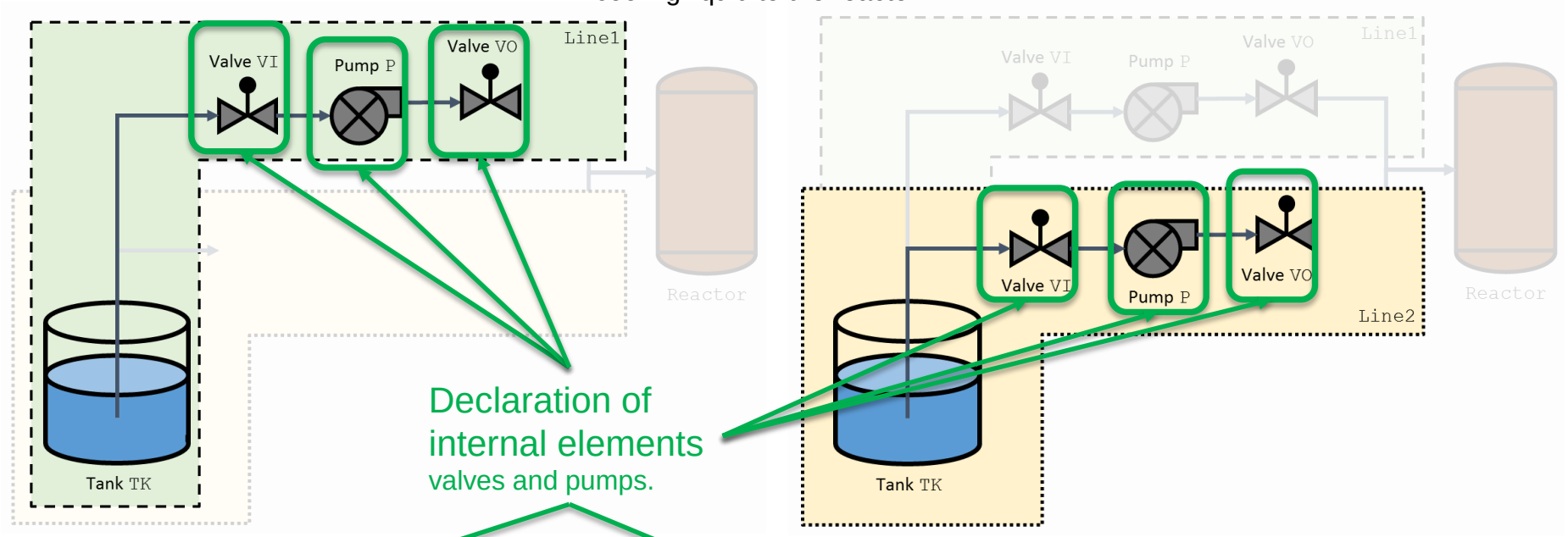
Declaration of the blocks

```
block Line1
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.

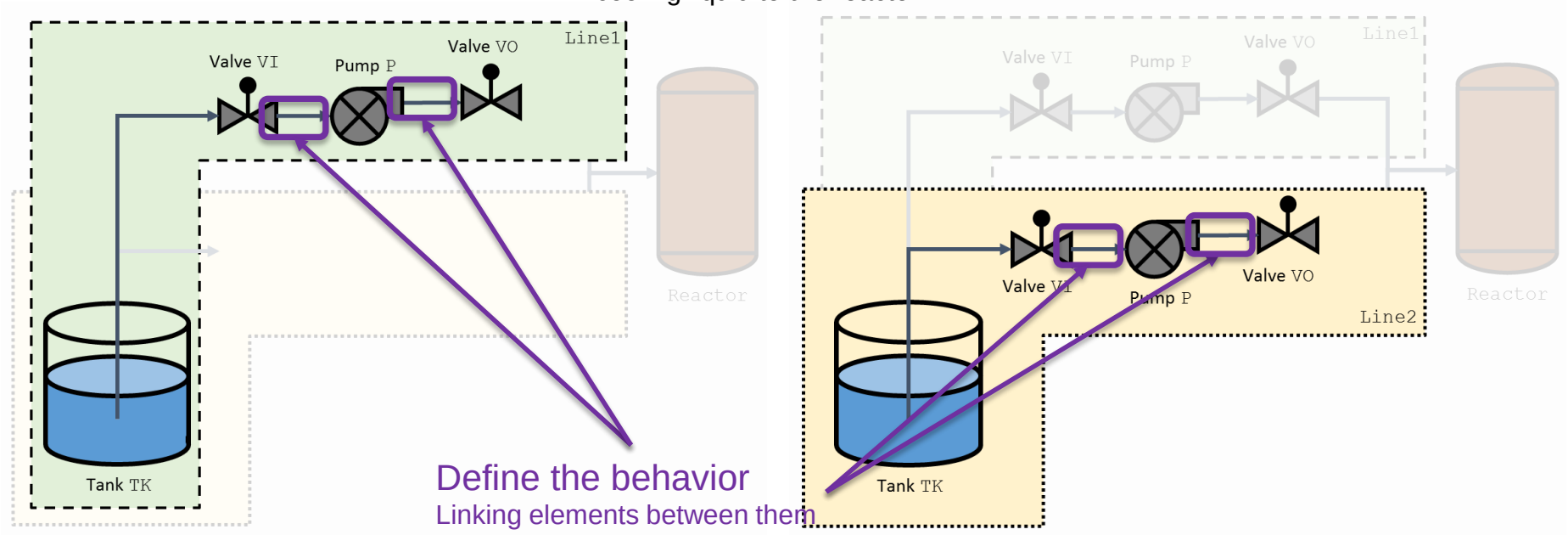



```
block Line1
    Valve VI, VO;
    Pump P;
  assertion
      P.vfInFlow := VI.vfRightFlow;
      VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    Valve VI, VO;
    Pump P;
  assertion
      P.vfInFlow := VI.vfRightFlow;
      VO.vfLeftFlow := P.vfOutFlow;
end
```

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.



Define the behavior

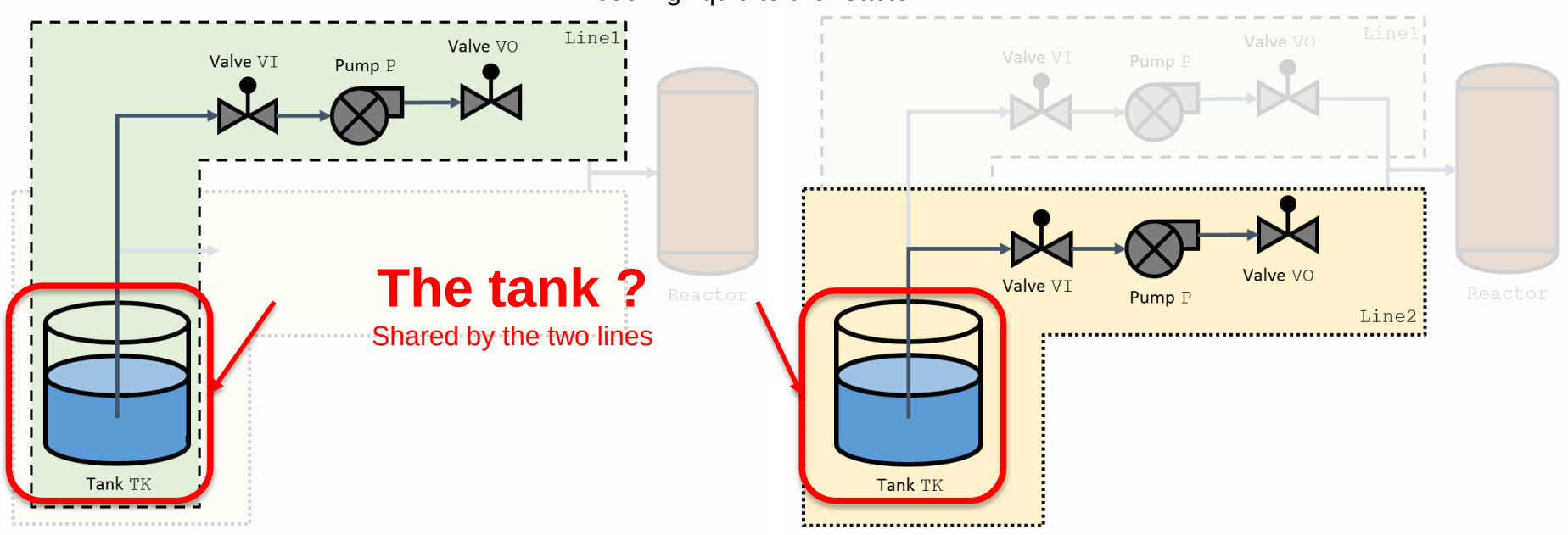
Linking elements between them

```
block Line1
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.

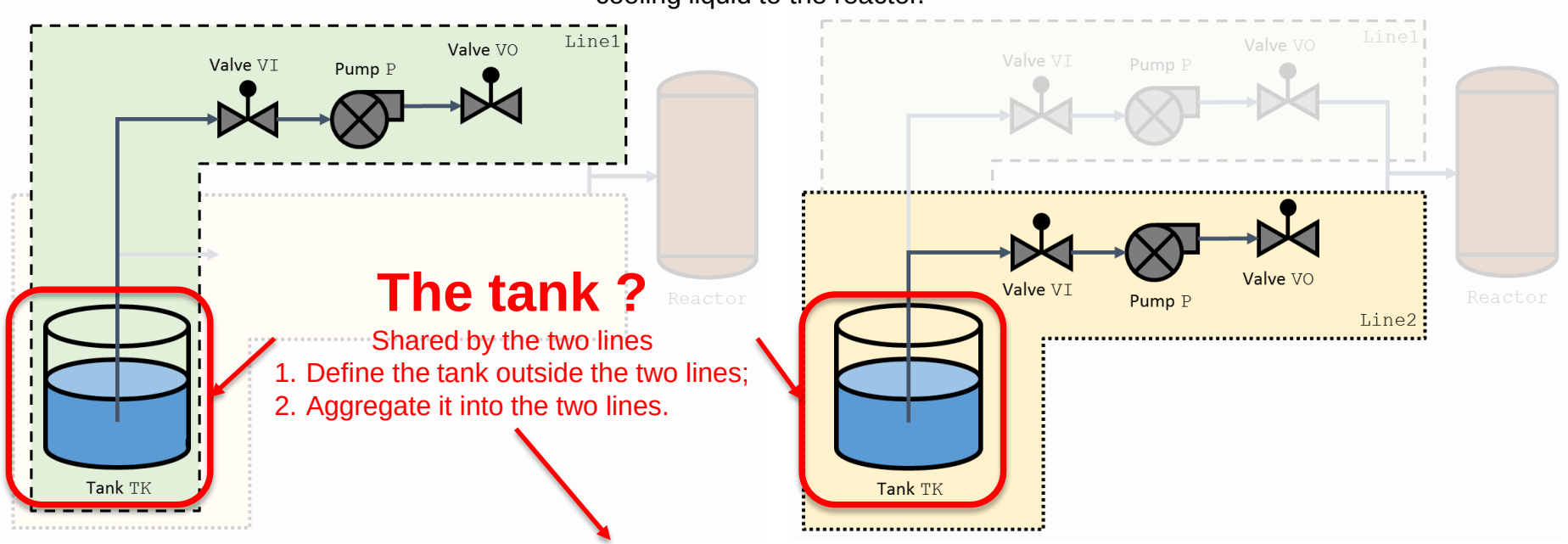



```
block Line1
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    Valve VI, VO;
    Pump P;
    assertion
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

Lines (`Line1` & `Line2`)

A set of components providing cooling liquid to the reactor.



The tank ?

Shared by the two lines

1. Define the tank outside the two lines;
2. Aggregate it into the two lines.

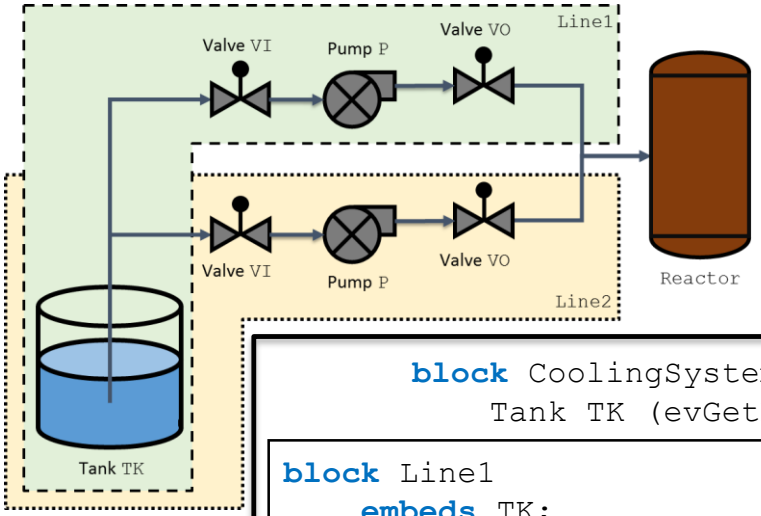
```
Tank TK (evGetEmpty.delay = Dirac(0.0));
```

```
block Line1
    embeds TK;
    Valve VI, VO;
    Pump P;
    assertion
        VI.vfLeftFlow := TK.vfOutFlow;
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Line2
    embeds TK;
    Valve VI, VO;
    Pump P;
    assertion
        VI.vfLeftFlow := TK.vfOutFlow;
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

OpenAltaRica - Example

A (Simple) Cooling System




```
block CoolingSystem
    Tank TK (evGetEmpty.delay = Dirac(0.0));
```

```
block Line1
    embeds TK;
    Valve VI, VO;
    Pump P;
    assertion
        VI.vfLeftFlow := TK.vfOutFlow;
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

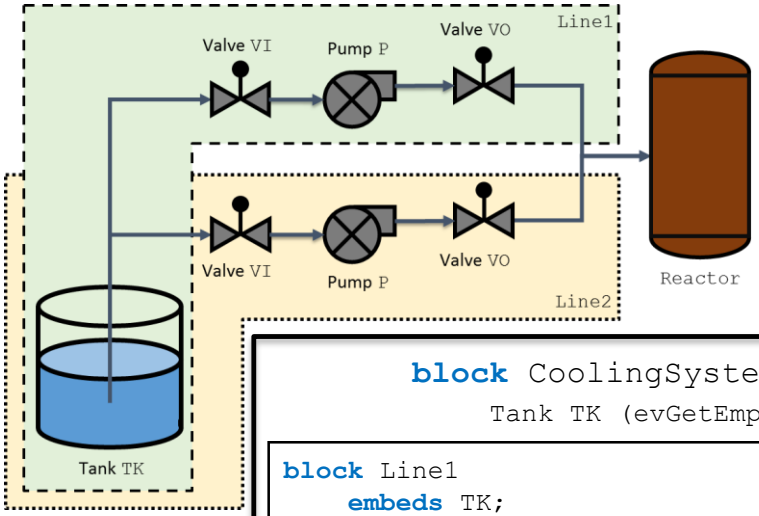
```
block Line2
    embeds TK;
    Valve VI, VO;
    Pump P;
    assertion
        VI.vfLeftFlow := TK.vfOutFlow;
        P.vfInFlow := VI.vfRightFlow;
        VO.vfLeftFlow := P.vfOutFlow;
end
```

```
block Reactor
    Boolean vfInFlow (reset = false);
end
```

```
    assertion
        Reactor.vfInFlow := Line1.VO.vfRightFlow
                         or Line2.VO.vfRightFlow;
end
```

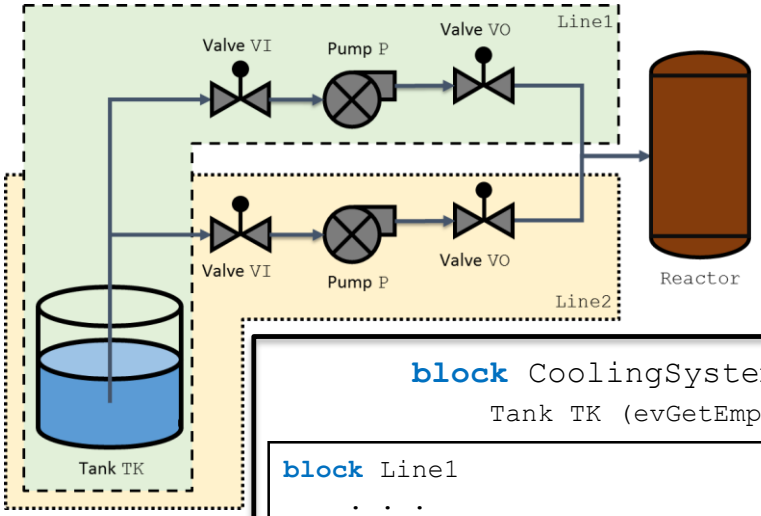
OpenAltaRica - Example

A (Simple) Cooling System



Add an observer to the input of the tank.

```
block CoolingSystem
    Tank TK (evGetEmpty.delay = Dirac(0.0));

    block Line1
        embeds TK;
        Valve VI, VO;
        Pump P;
        assertion
            VI.vfLeftFlow := TK.vfOutFlow;
            P.vfInFlow := VI.vfRightFlow;
            VO.vfLeftFlow := P.vfOutFlow;
    end

    block Line2
        embeds TK;
        Valve VI, VO;
        Pump P;
        assertion
            VI.vfLeftFlow := TK.vfOutFlow;
            P.vfInFlow := VI.vfRightFlow;
            VO.vfLeftFlow := P.vfOutFlow;
    end

    block Reactor
        Boolean vfInFlow (reset = false);
    end

    observer Boolean oCooledReactor = Reactor.vfInFlow;

    assertion
        Reactor.vfInFlow := Line1.VO.vfRightFlow
                         or Line2.VO.vfRightFlow;
end
```



Define a common-cause failure on the two pumps.

```
block CoolingSystem
    Tank TK (evGetEmpty.delay = Dirac(0.0));

block Line1
    . . .
end

block Line2
    . . .
end

block Reactor
    Boolean vfInFlow (reset = false);
end

observer Boolean oCooledReactor = Reactor.vfInFlow;

parameter Real pPumpsCCF = 1.0e-6;
event evPumpsCCF (delay = exponential(pPumpsCCF));

transition
    evPumpsCCF: !Line1.P.evFailure & !Line2.P.evFailure;

assertion
    Reactor.vfInFlow := Line1.VO.vfRightFlow
                     or Line2.VO.vfRightFlow;
end
```

CONTACTS

The OpenAltaRica project

www.openaltarica.fr

contact.oar@irt-systemx.fr

www.irt-systemx.fr