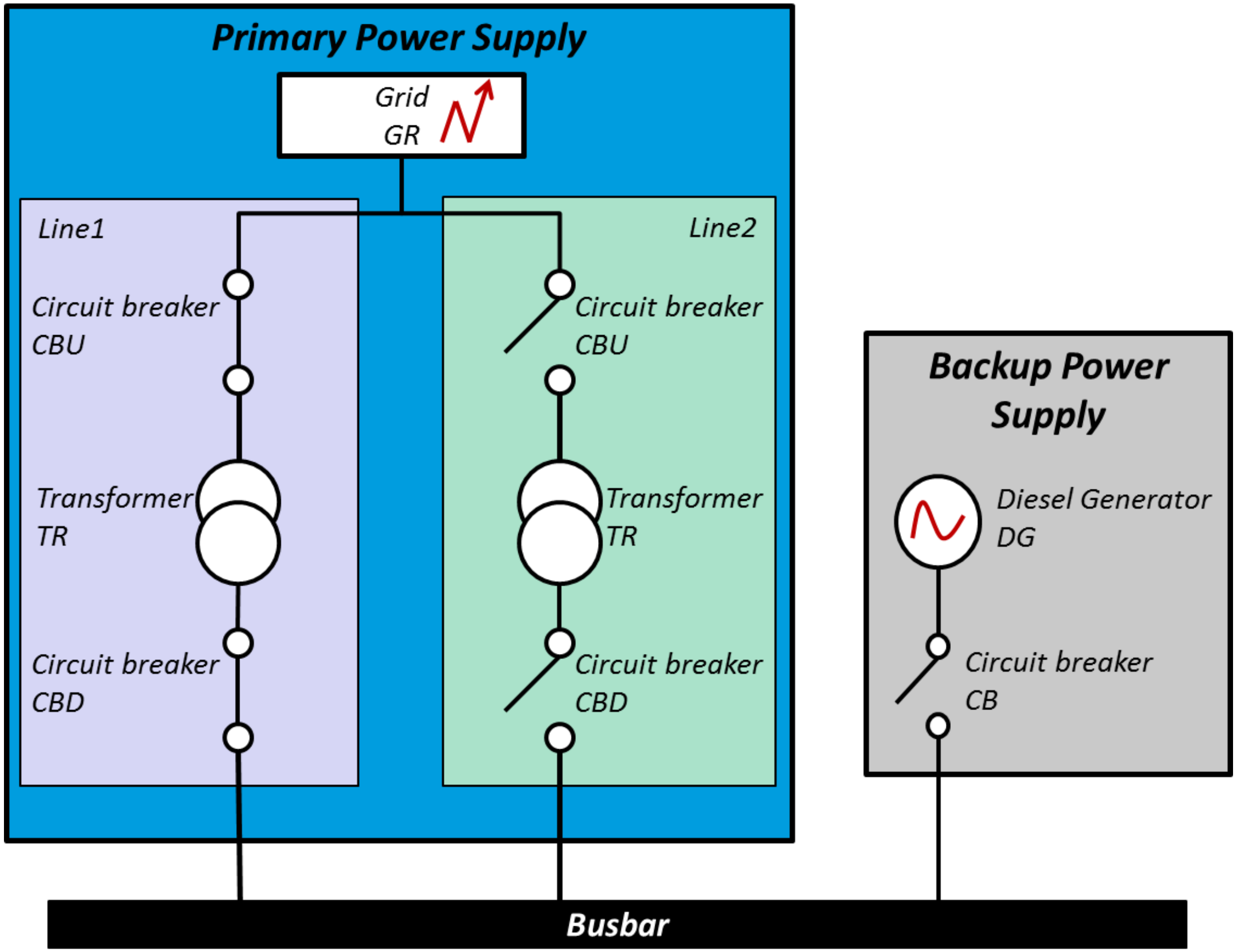
The Power Supply System



WHAT	Provide electrical power to the busbar
HOW	From the grid or a diesel generator by means of three redundant lines

The Power Supply System

Part 1 – Model the system

A. Warm up

The Power Supply System

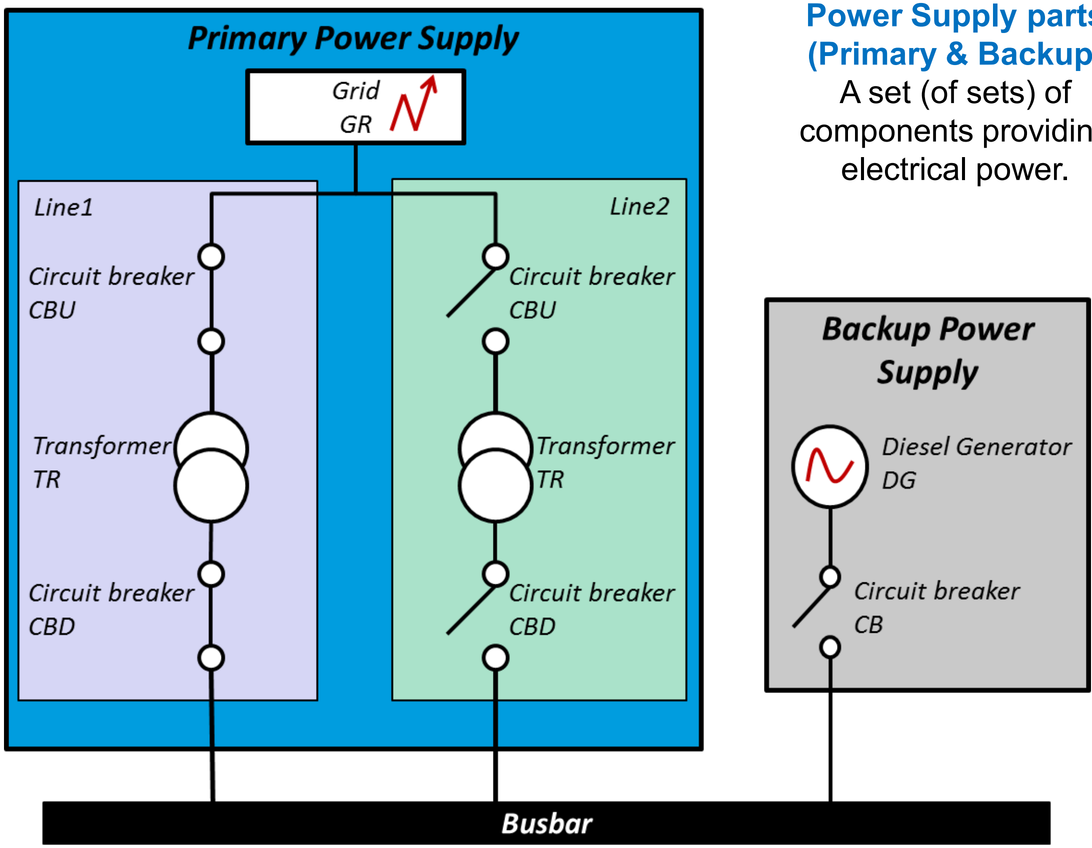
Different kinds of components/parts

Grid
A component providing power

Lines (`Line1` & `Line2`)
A set of components providing electrical power.

Transformer
To ‘transform’ the electrical power.

Busbar
The component to be powered



Power Supply parts (Primary & Backup)
A set (of sets) of components providing electrical power.

Diesel Generator
A component providing power

Circuit Breaker
To regulate (open or close) the power flow at input and output of lines.

Generic : Circuit Breaker, Transformer, (Diesel Generator, Grid, Busbar);
Ad-Hoc : (Diesel Generator, Grid, Busbar), Line1, Line2,
Primary Power Supply, Backup Power Supply, the system.

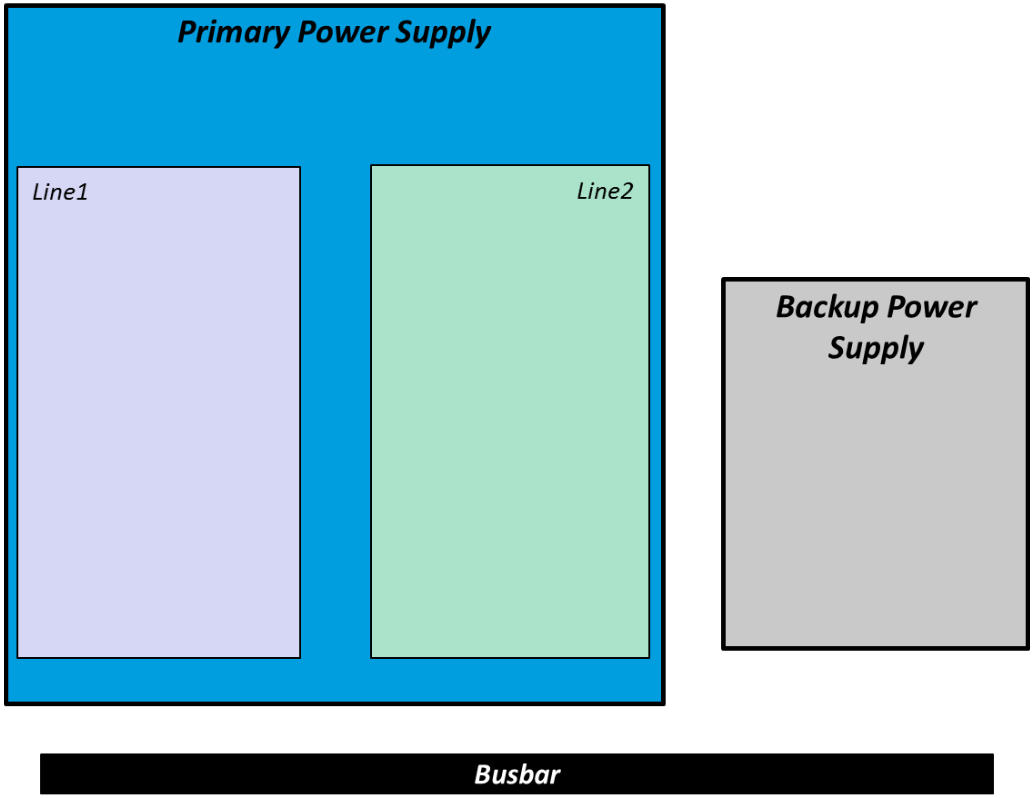
The Power Supply System

block

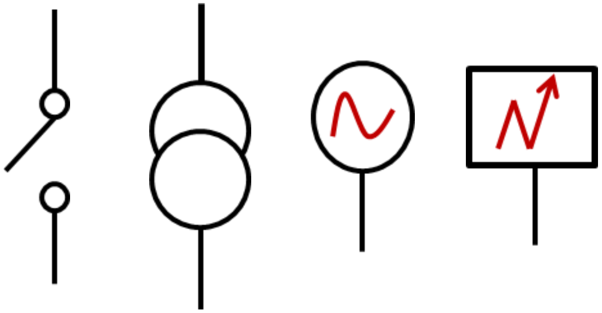
- Structural construct that represents a prototype, i.e. a component having a unique occurrence in the model;
- Since the model of the whole system is unique, it is always represented by a block.

class

- Structural construct that defines a generic component;
- Used in the model via instantiations;
- An instance of a class is called an "**object**".

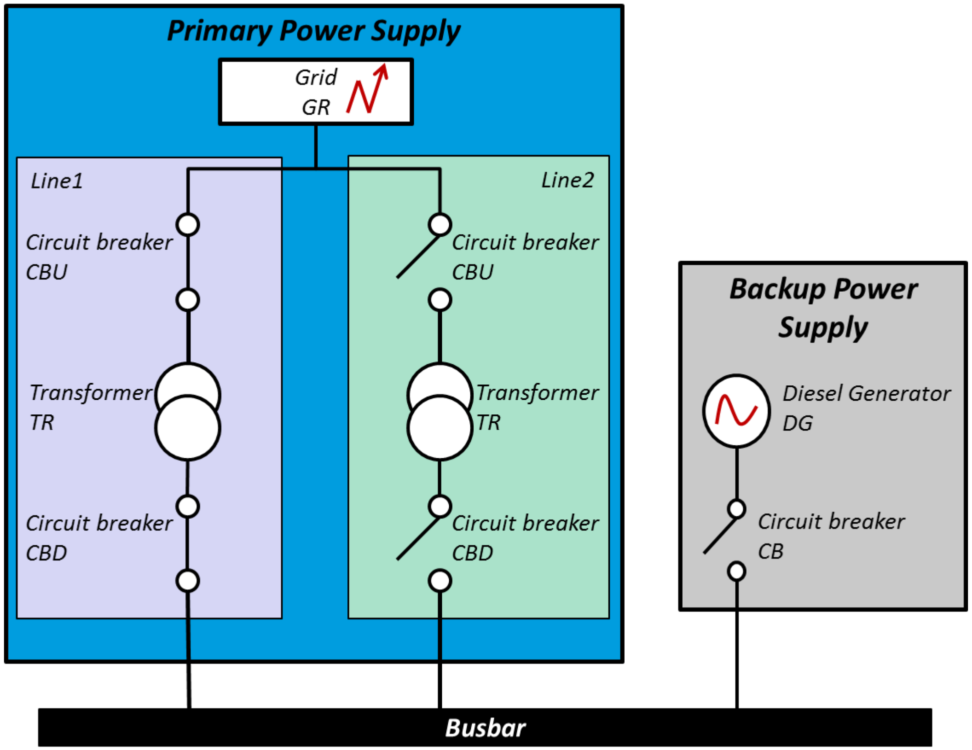


classes: Circuit Breaker, Transformer, Diesel Generator, Grid.

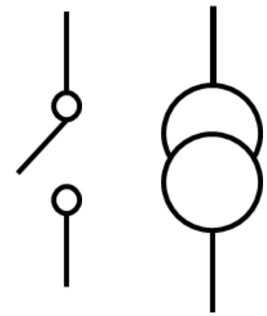


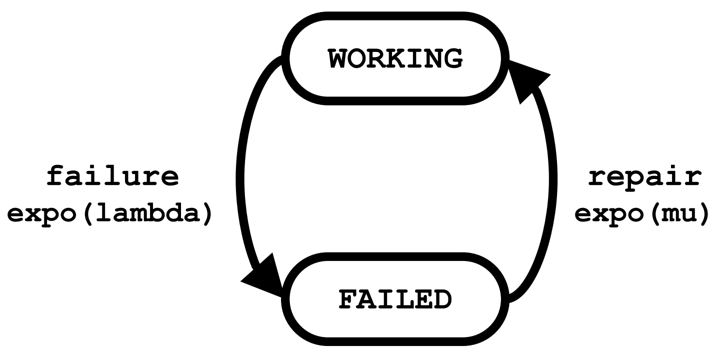
blocks: Busbar, Line1, Line2, Primary Power Supply, Backup Power Supply, (Power Supply System).

The Power Supply System



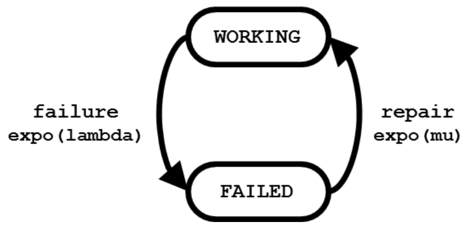
Transformer and Circuit Breaker components are repairable





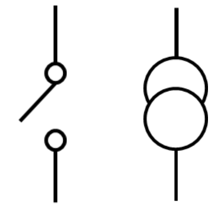
=> the same internal behavior

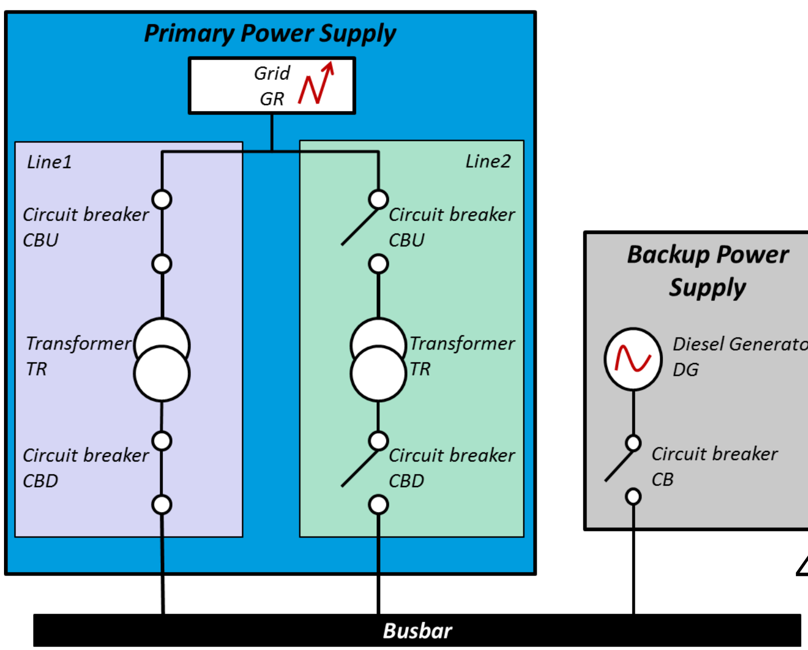
The Power Supply System



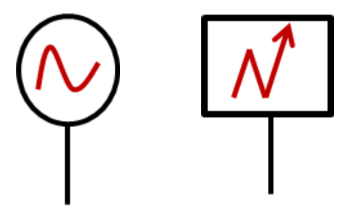
1. Define the internal repairable behavior into a generic element ‘class’;

2. Define Transformer and Circuit Breaker components by including this internal repairable behavior into specific elements ‘class’.





3. Define other generic components into specific element ‘class’.



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

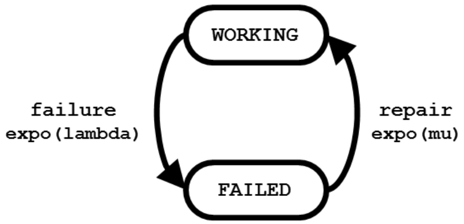
The Power Supply System

Part 1 – Model the system

A. Warm up

B. Model the components

The Power Supply System



1. Define the internal repairable behavior into a generic element ‘class’;
2. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
3. Define other generic components into a specific element ‘class’;
4. Build the model of the Power Supply System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

The Power Supply System

class

- Structural construct that defines a generic component;
- Used in the model via instantiations, i.e. cloning of a generic component;
- An instance of a class is called an "**object**".

```
class Coo

    Boolean vsState (init = true);
    Real vfOutput (reset = 0.0);
    parameter Real pLambda = 0.00001;
    event evItOccurs (delay = exponential(pLambda));

    transition
        evItOccurs: vsState -> vsState := false;
    assertion
        if vsState then vfOutput := 10.0;

end
```

The Power Supply System

class

- Structural construct that defines a generic component;
- Used in the model via instantiations, i.e. cloning of a generic component;
- An instance of a class is called an “**object**”.

Declaration of the class

Declaration of internal elements

```
class Coo
    Boolean vsState (init = true);
    Real vfOutput (reset = 0.0);
    parameter Real pLambda = 0.00001;
    event evItOccurs (delay = exponential(pLambda));
    transition
        evItOccurs: vsState -> vsState := false;
    assertion
        if vsState then vfOutput := 10.0;
end
```

Define the behavior

The Power Supply System

class

- Structural construct that defines a generic component;
- Used in the model via instantiations, i.e. cloning of a generic component;
- An instance of a class is called an "**object**".

Declaration
of internal elements

variables
state and flow

parameters

events

Etc.

```
class Coo
    Boolean vsState (init = true);
    Real vfOutput (reset = 0.0);
    parameter Real pLambda = 0.00001;
    event evItOccurs (delay = exponential(pLambda));
    transition
        evItOccurs: vsState -> vsState := false;
    assertion
        if vsState then vfOutput := 10.0;
end
```

The Power Supply System

class

- Structural construct that defines a generic component;
- Used in the model via instantiations, i.e. cloning of a generic component;
- An instance of a class is called an "**object**".

Internal
to the component

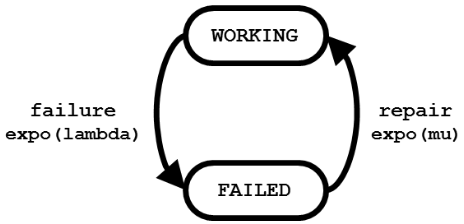
External
Relations with
other components

```
class Coo

    Boolean vsState (init = true);
    Real vfOutput (reset = 0.0);
    parameter Real pLambda = 0.00001;
    event evItOccurs (delay = exponential(pLambda));
    transition
        evItOccurs: vsState -> vsState := false;
    assertion
        if vsState then vfOutput := 10.0;
end
```

Define the behavior

The Power Supply System



1. Define the internal repairable behavior into a generic element 'class';
2. Define actuator components by including this internal repairable behavior into specific elements 'class';
3. Define other generic components into a specific element 'class';
4. Build the model of the Power Supply 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 = 0.00001;
    parameter Real pMu = 0.001;
    event evFailure (delay = exponential(pLambda));
    event evRepair (delay = exponential(pMu));
    transition
        evFailure: vsWorking -> vsWorking := false;
        evRepair: not vsWorking -> vsWorking := true;
end
```

The Power Supply System

1. Define the internal repairable behavior into a generic element ‘class’;
2. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
3. Define other generic components into a specific element ‘class’;
4. Build the model of the Power Supply System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

The Power Supply System

```
class Transformer
    extends RepairableComponent (pLambda = 0.0001, pMu = 0.001);
    Boolean vfInflow, vfOutflow (reset = false);
    assertion
        vfOutflow := if vsWorking then vfInflow else false;
end
```

```
class CircuitBreaker
    extends RepairableComponent;
    Boolean vfUpflow, vfDownflow (reset = false);
    assertion
        if vsWorking then vfDownflow :=: vfUpflow;
end
```

The Power Supply System

```
class Transformer
    extends RepairableComponent (pLambda = 0.0001, pMu = 0.001);
    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 CircuitBreaker
    extends RepairableComponent;
    Boolean vfUpflow, vfDownflow (reset = false);
    assertion
        if vsWorking then vfDownflow :=: vfUpflow;
end
```

The Power Supply System

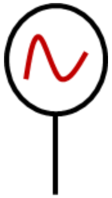
```
class Transformer
    extends RepairableComponent (pLambda = 0.0001, pMu = 0.001);
    Boolean vfInflow, vfOutflow (reset = false);
    assertion
        vfOutflow := if vsWorking then vfInflow else false;
end
```

Acausal connection ':=:'
between two flow variables

```
class CircuitBreaker
    extends RepairableComponent;
    Boolean vfUpflow, vfDownflow (reset = false);
    assertion
        if vsWorking then vfDownflow :=: vfUpflow;
end
```

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

The Power Supply System



1. Define the internal repairable behavior into a generic element ‘class’;
2. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
3. Define other generic components into specific elements ‘class’;
4. Build the model of the Power Supply System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

The Power Supply System

Grid

The component providing electrical power.

```
class Grid
    Boolean vfOutflow (reset = false);
    assertion
        vfOutflow := true;
end
```

The Power Supply System

Diesel Generator

This component can be empty (in sense of fuel).

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

The Power Supply System

Diesel Generator
This component can be empty (in sense of fuel).

```
class DieselGenerator
    Boolean vsIsEmpty (init = false);
    Boolean vfOutflow (reset = false);
    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)

The Power Supply System

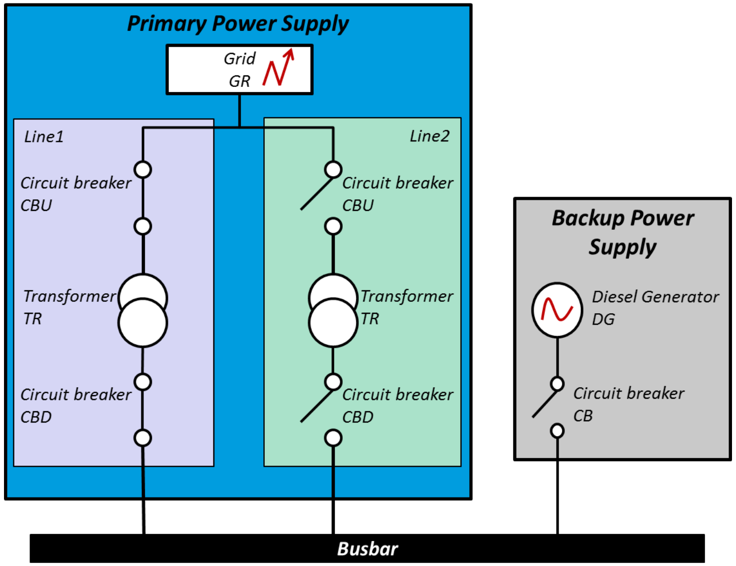
Part 1 – Model the system

A. Warm up

B. Model the components

C. Model the sub-parts

The Power Supply System



1. Define the internal repairable behavior into a generic element ‘class’;
2. Define actuator components by including this internal repairable behavior into specific elements ‘class’;
3. Define other generic components into specific elements ‘class’;
4. Build the model of the Power Supply System by instantiating these classes, creating ad-hoc parts (into specific elements “block”) and linking them.

classes: Circuit Breaker, Transformer, Diesel Generator, Grid.

blocks: Busbar, Line1, Line2, Primary Power Supply, Backup Power Supply, (Power Supply System).

The Power Supply System

block

- Structural construct that represents a prototype, i.e. a component having a unique occurrence in the model;
- Since the model of the whole system is unique, it is always represented by a block..

Declaration of internal elements

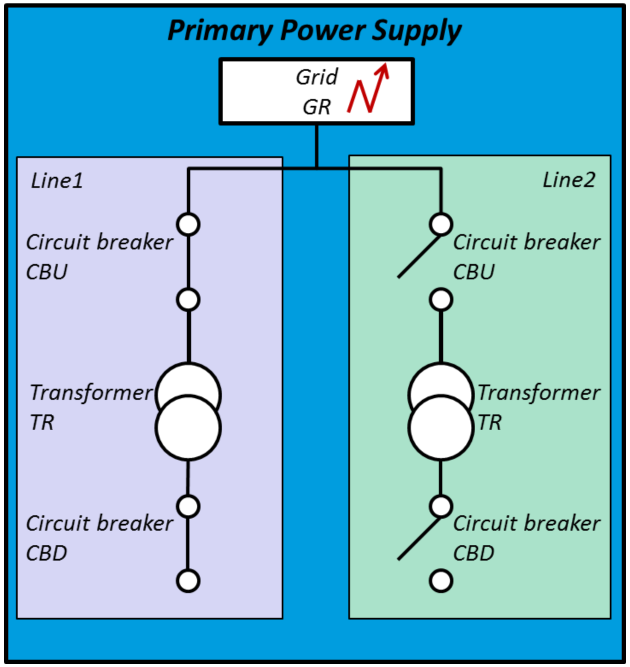
Declaration of the block

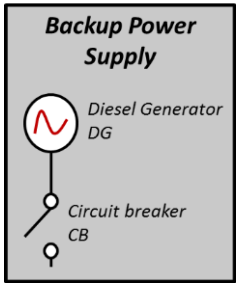
```
block Boo
    Boolean vsState (init = true);
    Real vfOutput (reset = 0.0);
    parameter Real pLambda = 0.00001;
    event evItOccurs (delay = exponential(pLambda));
    transition
        evItOccurs: vsState -> vsState := false;
    assertion
        if vsState then vfOutput := 10.0;
end
```

Define the behavior

The Power Supply System

Primary & Backup Power Supplies
A set of (sets of) components providing
electrical power to the Grid.






```
block PrimaryPowerSupply
    Grid G;

    block Line1
        CircuitBreaker CBU, CBD;
        Transformer TR;
        assertion
            TR.vfInflow := CBU.vfDownflow;
            CBD.vfUpflow := TR.vfOutflow;
    end

    clones Line1 as Line2;

    assertion
        Line1.CBU.vfInflow := G.vfOutflow;
        Line2.CBU.vfInflow := G.vfOutflow;
end
```

```
block BackUpPowerSupply
    DieselGenerator DG;
    CircuitBreaker CB;
    assertion
        CB.vfUpflow := DG.vfOutflow;
end
```

The Power Supply System

Busbar
A set of (sets of) components providing
electrical power to the Grid.


```
block Busbar
    Boolean vfInput (reset = false);
end
```

The Power Supply System

Part 1 – Model the system

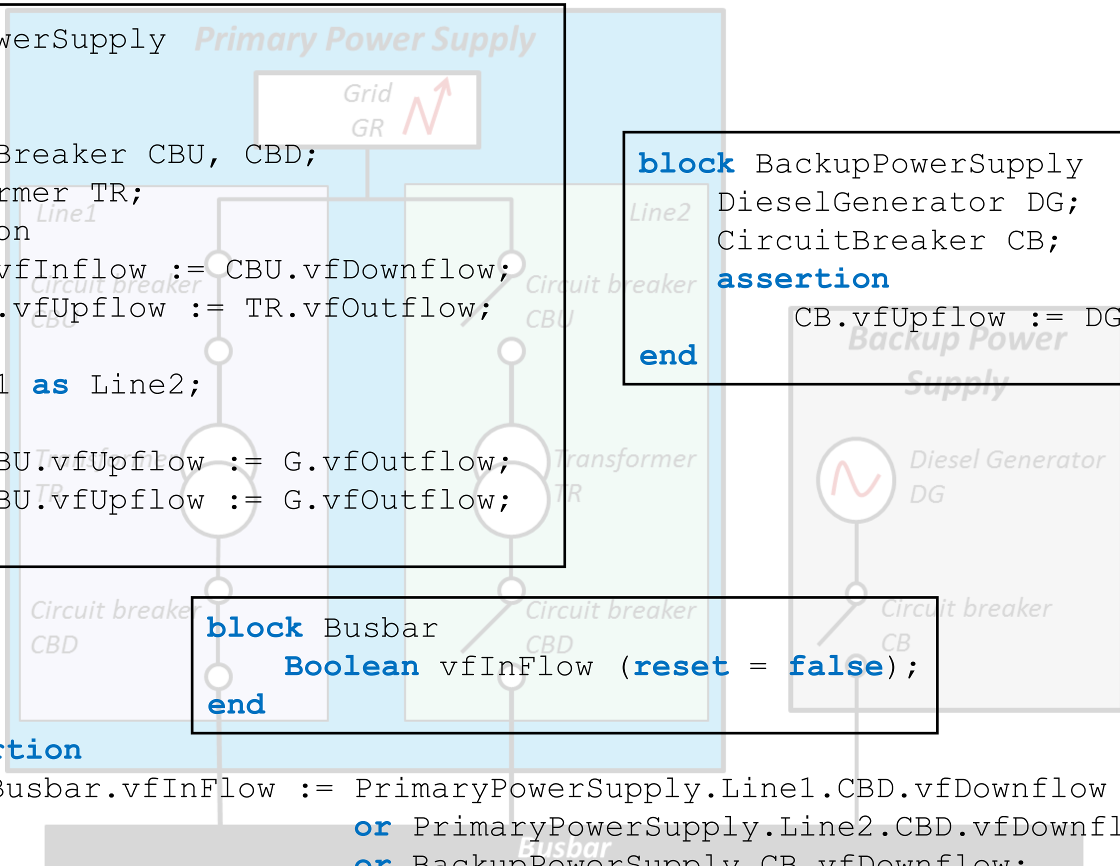
A. Warm up

B. Model the components

C. Model the sub-parts

D. Model the system

The Power Supply System



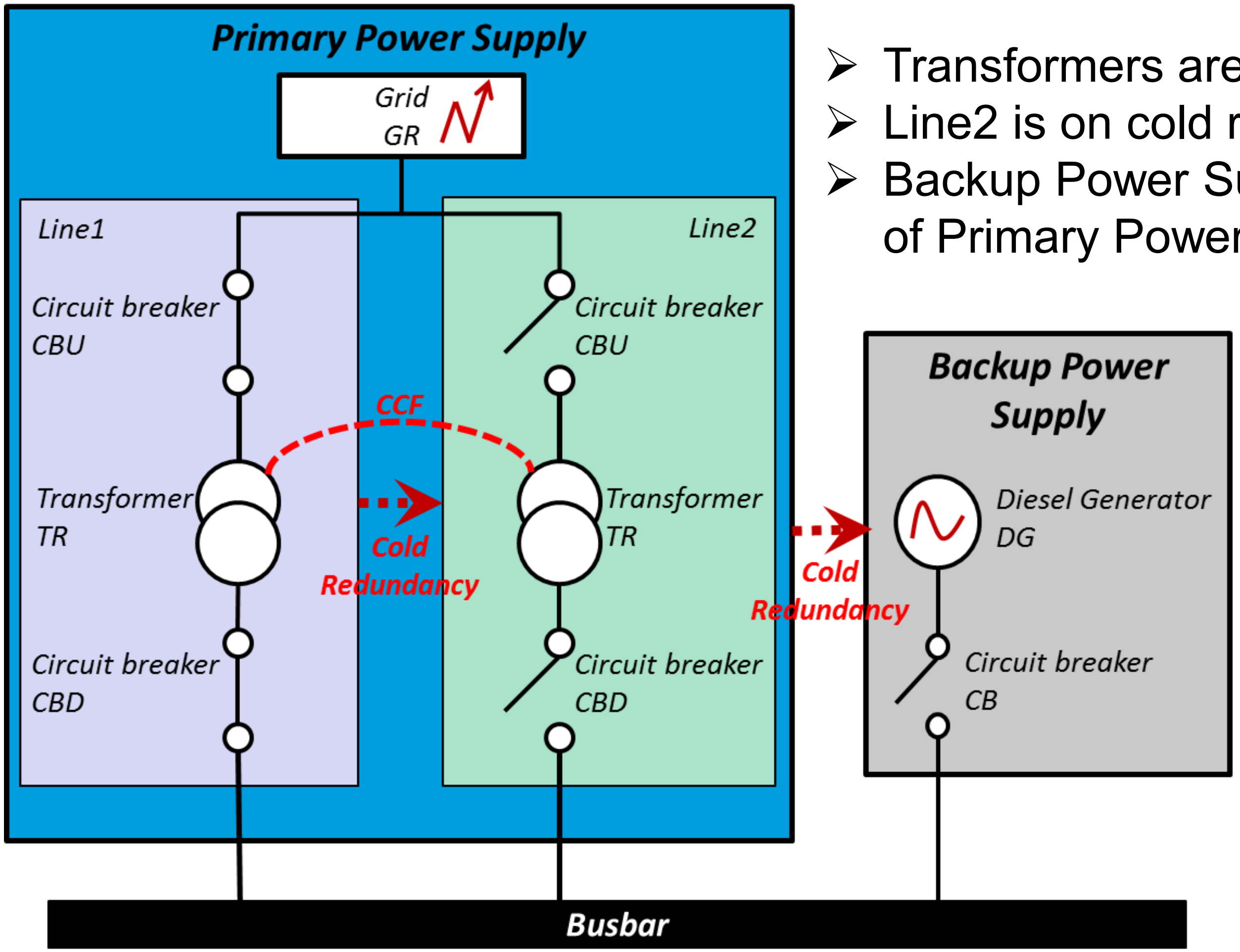

```
block PowerSupplySystem
  block PrimaryPowerSupply
    Grid G;
    block Line1
      CircuitBreaker CBU, CBD;
      Transformer TR;
      assertion
        TR.vfInflow := CBU.vfDownflow;
        CBD.vfUpflow := TR.vfOutflow;
    end
    clones Line1 as Line2;
    assertion
      Line1.CBU.vfUpflow := G.vfOutflow;
      Line2.CBU.vfUpflow := G.vfOutflow;
  end
  block BackupPowerSupply
    DieselGenerator DG;
    CircuitBreaker CB;
    assertion
      CB.vfUpflow := DG.vfOutflow;
  end
  block Busbar
    Boolean vfInFlow (reset = false);
  end
  assertion
    Busbar.vfInFlow := PrimaryPowerSupply.Line1.CBD.vfDownflow
                       or PrimaryPowerSupply.Line2.CBD.vfDownflow
                       or BackupPowerSupply.CB.vfDownflow;
end
```

The Power Supply System

Part 3 – Improve dynamically the system

The Power Supply System

The system contains dynamic features



- Transformers are on common cause failure
- Line2 is on cold redundancy of Line1
- Backup Power Supply is on cold redundancy of Primary Power Supply

2 steps

1. Design the common cause failure on transformers
2. Design cold redundancies

The Power Supply System

The system contains dynamic features

1. Design the common cause failure on transformers

```
block PrimaryPowerSupply
     Grid G;
    block Line1
        CircuitBreaker CBU, CBD;
        Transformer TR;
        assertion
            TR.vfInflow := CBU.vfDownflow;
            CBD.vfUpflow := TR.vfOutflow;
    end
    clones Line1 as Line2;

    parameter Real pLambdaCCF = 1.0e-5;
    event evCCF (delay = exponential(pLambdaCCF));
    transition
        evCCF: ?Line1.TR.evFailure & ?Line2.TR.evFailure;

    assertion
        Line1.CBU.vfInflow := G.vfOutflow;
        Line2.CBU.vfInflow := G.vfOutflow;
end
```

The Power Supply System

The system contains dynamic features

1. Design cold redundancies
   i. a spare component

```
domain SpareComponentState {WORKING, FAILED, STANDBY}

class SpareComponent
    SpareComponentState vs (init = WORKING);
    Boolean vfDemanded (reset = false);
    parameter Real pLambda = 0.0001;
    parameter Real pFailToStart = 0.00001;
    event evFailure (delay = exponential(pLambda));
    event evFailureOnDemand (delay = Dirac(0.0), expectation = pFailToStart);
    event evStart (delay = Dirac(0.0), expectation = 1 - pFailToStart);
    transition
        evFailure: vs == WORKING -> vs := FAILED;
        evFailureOnDemand: vs == STANDBY and vfDemanded -> vs := FAILED;
        evStart: vs == STANDBY and vfDemanded -> vs := WORKING;
end
```

The Power Supply System

The system contains dynamic features

1. Design cold redundancies
   i. a spare component
   ii. a spare transformer, etc.

```
class SpareTransformer
    extends SpareComponent;
    Boolean vfInflow, vfOutflow (reset = false);
    assertion
        vfOutflow := (vs == WORKING) and vfInflow;
end
```

The Power Supply System

The system contains dynamic features

1. Design cold redundancies
   i. a spare component
   ii. a spare transformer, etc.
   iii. Link a main component and a spare one

```
block System
    ...
    SpareTransformer main;
    SpareTransformer spare (vs.init = STANDBY);
    ...
    assertion
        spare.vfDemanded := not main.vfOutflow;
        ...
    ...
end
```