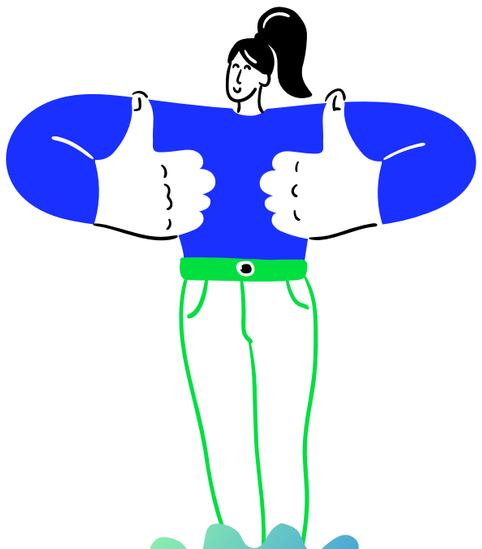
oose. x DASSAULT SYSTEMES

SysML v2 Quick Sheet



We are your trusted partner in professional development

Definition and Usage

Definition

«occurrence def»
MBSE Project

Usages

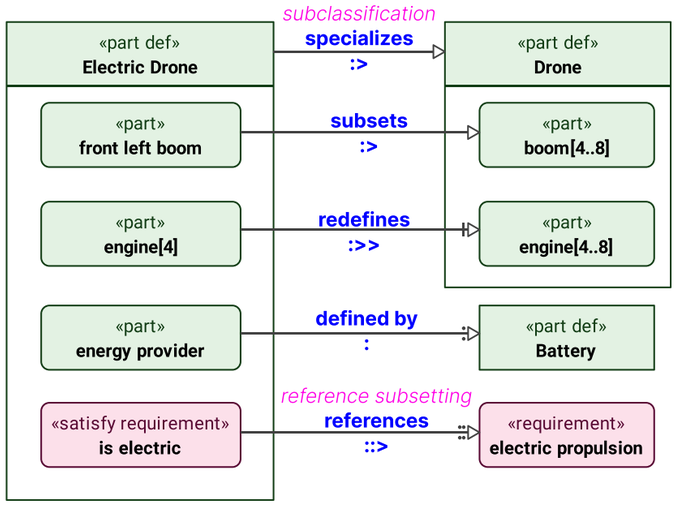
«occurrence»
thirty meter telescope : MBSE Project

«occurrence»
ITER Fusion Reactor : MBSE Project

«occurrence»
TVP Coffee Machine : MBSE Project

```
occurrence def 'MBSE Project';
occurrence 'thirty meter telescope' : 'MBSE Project';
occurrence 'TVP Coffee Machine' : 'MBSE Project';
occurrence 'ITER Fusion Reactor' : 'MBSE Project';
```

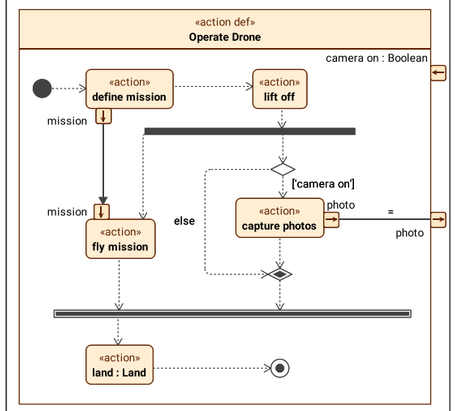
Specializations



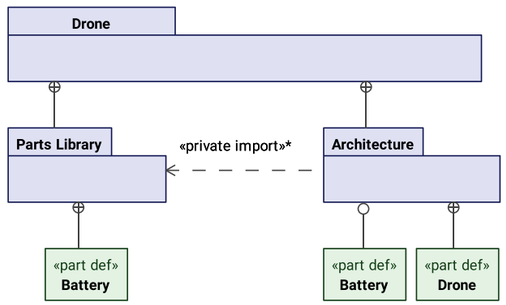

```
part def Drone {
        part engine [4..8];
        part boom [4..8];
}
part def Battery;
requirement 'electric propulsion';
part def 'Electric Drone' :> Drone {
        part 'front left boom' :> boom;
        part 'engine' :>> engine [4];
        part 'energy provider' : Battery;
        satisfy requirement 'is electric' ::> 'electric propulsion';
}
```

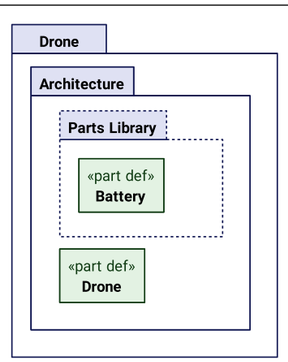
Actions

```
action def Land;
action def 'Operate Drone' {
    action 'define mission' { out item mission; }
    first start;
    then 'define mission';
    action 'lift off';
    decide decide1;
    fork fork1;
    join join1;
    action 'fly mission' { in item mission; }
    action 'capture photos' { out item photo; }
    action land : Land;
    merge merge1;
    first fork1 then 'fly mission';
    first fork1 then decide1;
    first 'fly mission' then join1;
    first 'capture photos' then merge1;
    first land then done;
    out item photo;
    first 'lift off' then fork1;
    flow flow1 from 'define mission'.mission to 'fly
mission'.mission;
    first decide1;
    if 'camera on' then 'capture photos';
    else merge1;
    first join1 then land;
    first merge1 then join1;
    in attribute 'camera on' : ScalarValues::Boolean;
    first 'define mission' then 'lift off';
    binding bind 'capture photos'.photo = photo;
}
```



Packages

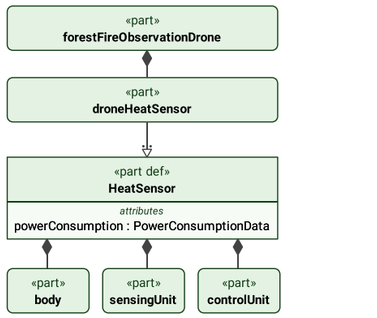


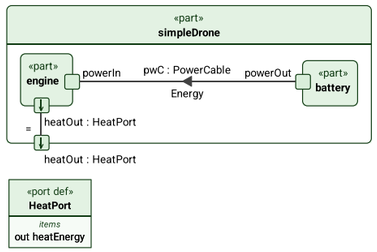


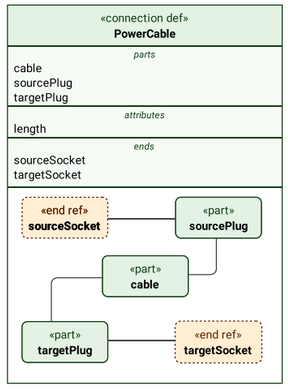

```
package Drone {
    package Architecture {
        private import 'Parts Library'::*;
        part def Drone {
            part battery : Battery;
        }
    }
    package 'Parts Library' {
        part def Battery;
    }
}
```

Structure

```
part forestFireObservationDrone {
    part droneHeatSensor : HeatSensor;
}
part def HeatSensor {
    part controlUnit;
    part body;
    part sensingUnit;
    attribute powerConsumption :
        PowerConsumptionData;
}
attribute def PowerConsumptionData;
port def HeatPort {
    out item heatEnergy;
}
part simpleDrone {
    part battery {
        port powerOut;
    }
    part engine {
        port heatOut : HeatPort;
        port powerIn;
    }
    connection pwC : PowerCable connect
     battery.powerOut to engine.powerIn {
        flow of Energy {
            end ::> sourceSocket;
            end ::> targetSocket;
        }
    }
    port heatOut : HeatPort;
    binding bind heatOut = engine.heatOut;
}
item def Energy;
connection def PowerCable {
    end sourceSocket;
    end targetSocket;
    attribute length;
    part cable;
    part sourcePlug;
    part targetPlug;
    connection connect sourceSocket to
       sourcePlug;
    connection connect sourcePlug to cable;
    connection connect targetPlug to cable;
    connection connect targetSocket to
       targetPlug;
}
```

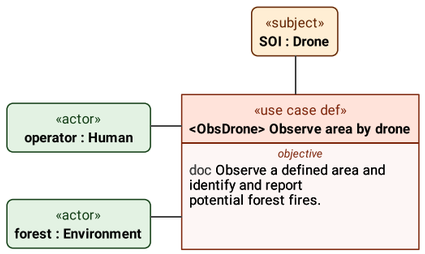






Use Cases

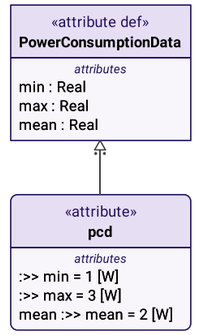
```
part def Human;
part def Environment;
part def Drone;
use case def <ObsDrone> 'Observe area by
       drone' {
    subject SOI : Drone;
    actor operator : Human;
    actor forest : Environment;
    objective {
        doc
        /* Observe a defined area and
         * identify and report
         * potential forest fires.
         */
    }
}
```



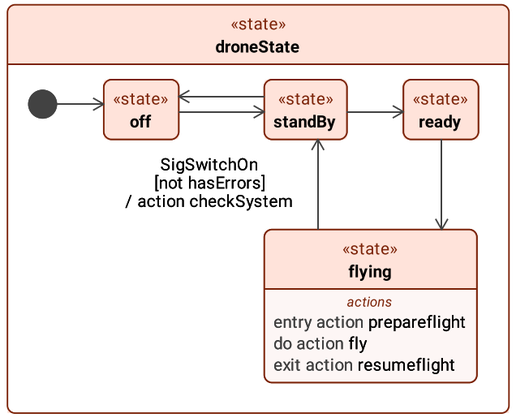
Attributes and Units

```
private import ScalarValues::*;
private import SI::*;

attribute def PowerConsumptionData {
    attribute min : Real;
    attribute max : Real;
    attribute mean : Real;
}
attribute pcd : PowerConsumptionData {
    attribute :>> min = 1.0 [W];
    attribute :>> max = 3.0 [W];
    attribute mean :>> mean = 2.0 [W];
}
```

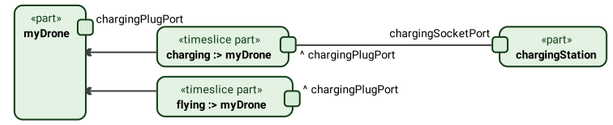


States



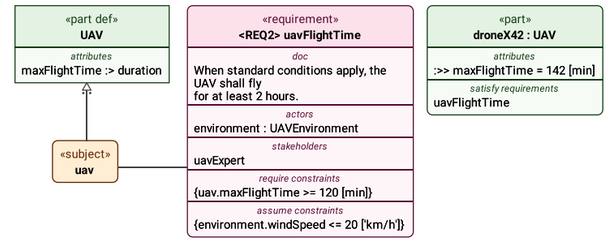

```
state droneState {
    in attribute hasErrors : ScalarValues::Boolean;
    transition t1 first off accept SigSwitchOn if not hasErrors do
      checkSystem then standBy;
    transition t2 first standBy then ready;
    transition t3 first ready then flying;
    state off;
    state standBy;
    state ready;
    state flying {
        entry action prepareflight;
        do action fly;
        exit action resumeflight;
    }
    transition first start then off;
    transition first flying then standBy;
    transition first standBy then off;
}
item def SigSwitchOn;
action checkSystem;
```

Occurrences in Space & Time




```
part myDrone {
    port chargingPlugPort;
    timeslice part flying :> myDrone;
    timeslice part charging :> myDrone;
}
part chargingStation {
    port chargingSocketPort;
}
interface connect myDrone.charging.chargingPlugPort to
chargingStation.chargingSocketPort;
```

Requirements




```
private import SI::**;
part def UAV {
    attribute maxFlightTime :> duration;
}
part def UAVEnvironment {
    attribute windSpeed;
}
requirement <REQ2> uavFlightTime {
    doc
    /* When standard conditions apply, the UAV shall fly
     * for at least 2 hours.
     */
    subject uav : UAV;
    actor environment : UAVEnvironment;
    stakeholder uavExpert;
    assume constraint {
        environment.windSpeed <= 20 ['km/h']
    }
    require constraint {
        uav.maxFlightTime >= 120 [min]
    }
}
part droneX42 : UAV {
    attribute :>> maxFlightTime = 142 [min];
    satisfy uavFlightTime;
}
```

What's next?

- Deepen your knowledge with our SysML v2 trainings
- Explore SysML Club, our online learning platform and community
- Share your feedback to help improve this QuickSheet

Hamza Bassam
Trainer & Consultant

Julia Stede
Sales & Organisation