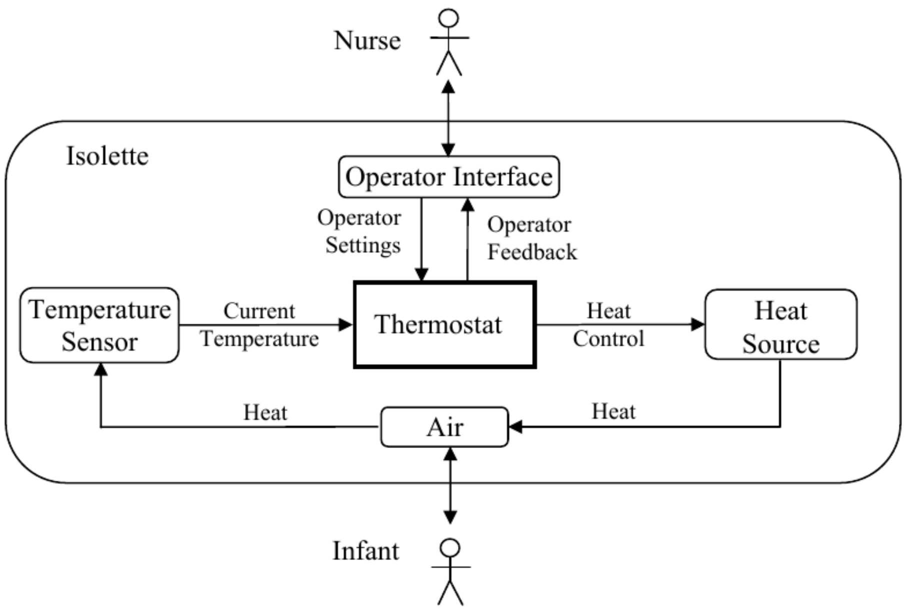
Progressive Isolette AADL Models[1]

Version No.	Description
01	Empty Isolette system type with two implementations
02	Minimally defined Thermostat subsystem added as Isolette subcomponent
03	Minimally defined physical devices added as Isolette subcomponents
04	System & device interfaces defined, but left untyped
05	System & device interfaces typed with modeled data entities
06	Intra-system connections made among subcomponents
07	Use of enumerated & range composite data entities
08	Abstract Air component in role as controlled block of Isolette control loop
09	Software for Thermostat regulating & monitoring control functionality
10	Operator Interface with temperature settings, display & alarm devices
11	Software for regulating & monitoring interface, mode, & failure functionality



Isolette Context Diagram

[1] Models have been added to the 890-isolette Git repository.

The AADL Models

e_single_sensor_In
te_dual_sensor_Ins

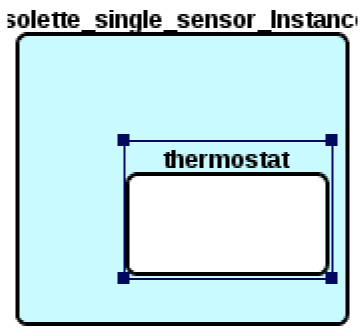
Version 01: Empty Isolette system type with two implementations

Isolette.aadl (single sensor)

```
system isolette
end isolette;

system implementation isolette.single_sensor
end isolette.single_sensor;
```

Isolette.aadl (dual sensor)

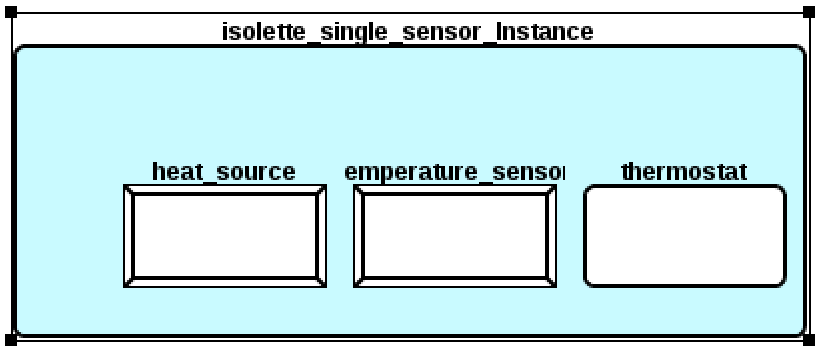
```
system isolette
end isolette;

system implementation isolette.dual_sensor
end isolette.dual_sensor;
```



Version 02: Minimally defined Thermostat subsystem added as Isolette subcomponent

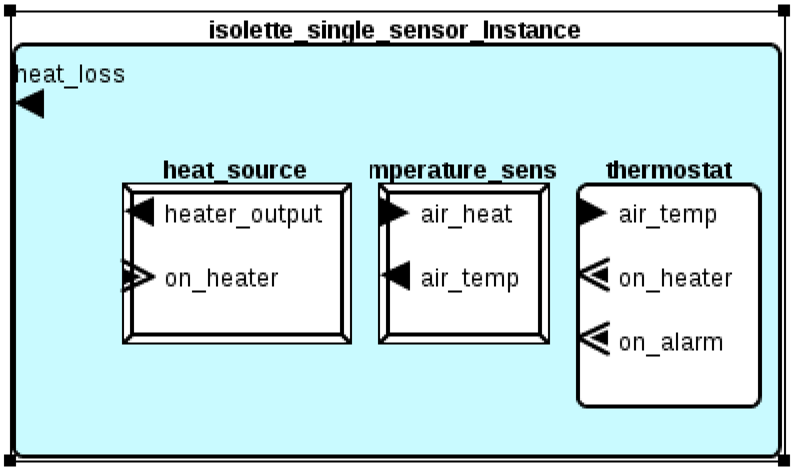
Isolette.aadl

```
system isolette
end isolette;

system implementation isolette.single_sensor
  subcomponents
    thermostat: system thermostat_single_sensor.impl;
end isolette.single_sensor;


system thermostat_single_sensor
end thermostat_single_sensor;

system implementation thermostat_single_sensor.impl
end thermostat_single_sensor.impl;
```



Version 03: Minimally defined physical devices added as Isolette subcomponents

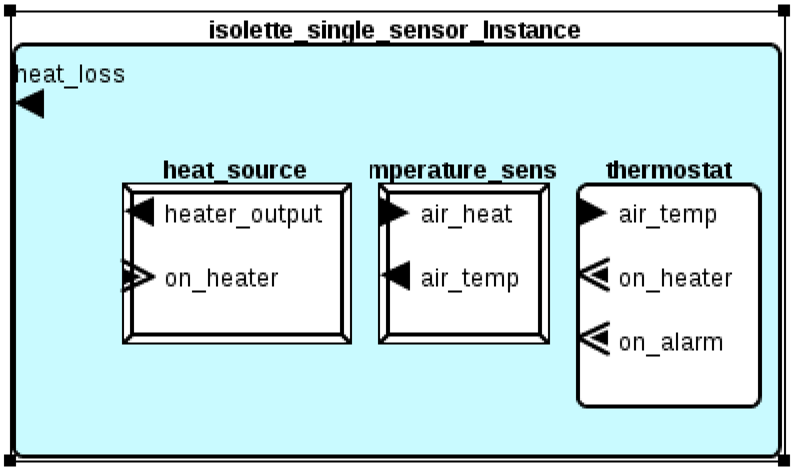
Isolette.aadl	Devices.aadl
`system isolette`	`device heat_source`
`end isolette;`	`end heat_source;`
`system implementation isolette.single_sensor`	`device implementation heat_source.impl`
`  subcomponents`	`end heat_source.impl;`
`    thermostat: system thermostat_single_sensor.impl;`	
`    heat_source: device Devices::heat_source.impl;`	
`    temp_sensor: device Devices::temp_sensor.impl;`	`device temp_sensor`
`end isolette.single_sensor;`	`end temp_sensor;`
	`device implementation temp_sensor.impl`
	`end temp_sensor.impl;`



Version 04: System & device interfaces defined, but left untyped

Isolette.aadl	Devices.aadl

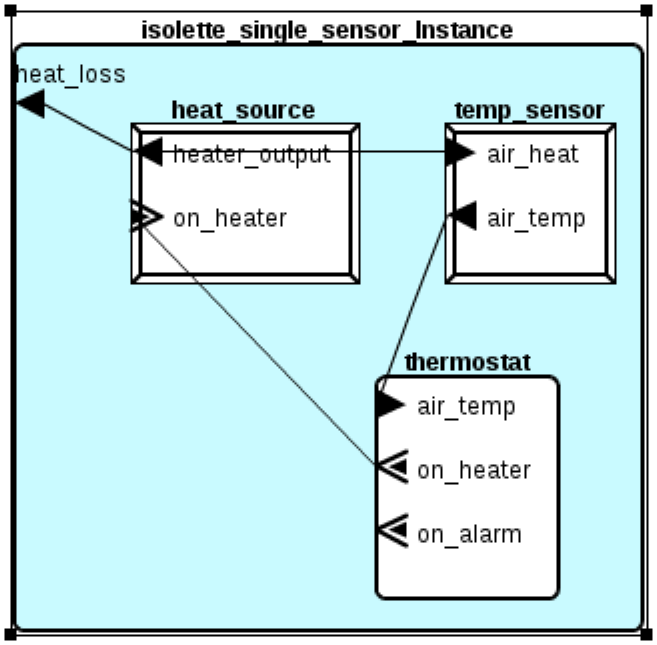
```
system isolette
  features
    heat_loss: out data port;
end isolette;


system thermostat_single_sensor
  features
    air_temp: in data port;
    on_heater: out event data port;
    on_alarm: out event data port;
end thermostat_single_sensor;
```
	```
	device heat_source
	  features
	    on_heater: in event data port;
	    heater_output: out data port;
	end heat_source;


	device temperature_sensor
	  features
	    air_heat: in data port;
	    air_temp: out data port;
	end temperature_sensor;
	```



Version 05: System & device interfaces typed with modeled data entities

Devices.aadl	Iso_Types.aadl

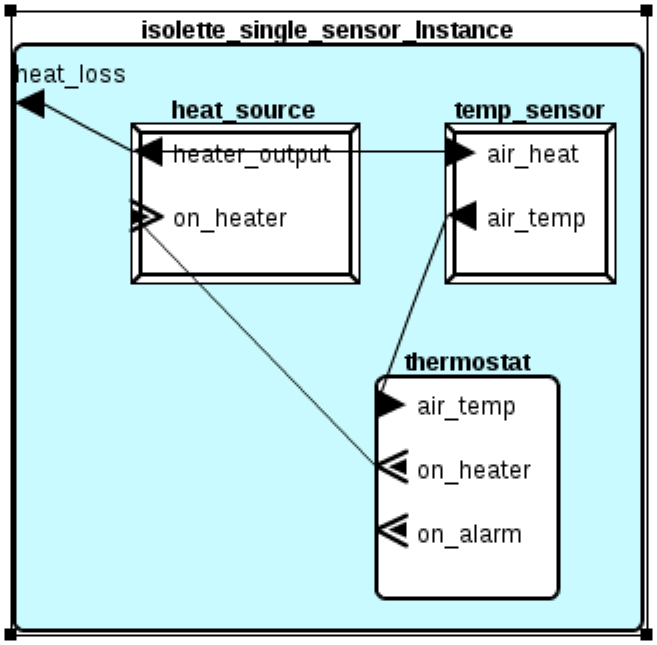
```
device heat_source
  features
    on_heater: in event data port Base_Types::
                                      Boolean;
    heater_output: out data port Iso_Types::
                                         heat;
end heat_source;


device temperature_sensor
  features
    air_heat: in data port Iso_Types::heat;
    air_temp: out data port Iso_Types::
                              temperature;
end temperature_sensor;
```
	```
	data heat
	  properties
	    Data_Model::Data_Representation => Float;
	    Data_Model::Measurement_Unit => "KJoule";
	end heat;


	data temperature
	  properties
	    Data_Model::Data_Representation => Float;
	    Data_Model::Measurement_Unit =>
	                                "Fahrenheit";
	end temperature;
	```



Version 06: Intra-system connections made among subcomponents

Isolette.aadl

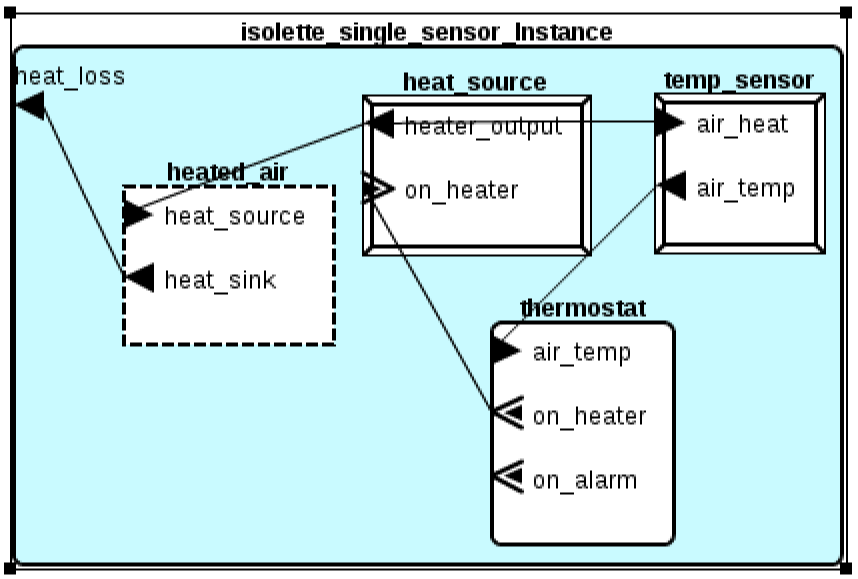
```
system implementation isolette.single_sensor
  subcomponents
    thermostat: system thermostat_single_sensor.impl;
    heat_source: device Devices::heat_source.impl;
    temp_sensor: device Devices::temperature_sensor.impl;

  connections
    t1: port temp_sensor.air_temp -> thermostat.air_temp;
    t2: port thermostat.on_heater -> heat_source.on_heater;

    h1: port heat_source.heater_output -> temp_sensor.air_heat;
    h2: port heat_source.heater_output -> heat_loss;  -- No place to go!
end isolette.single_sensor;
```



Version 07: Use of enumerated & range composite data entities

Iso_Types.aadl

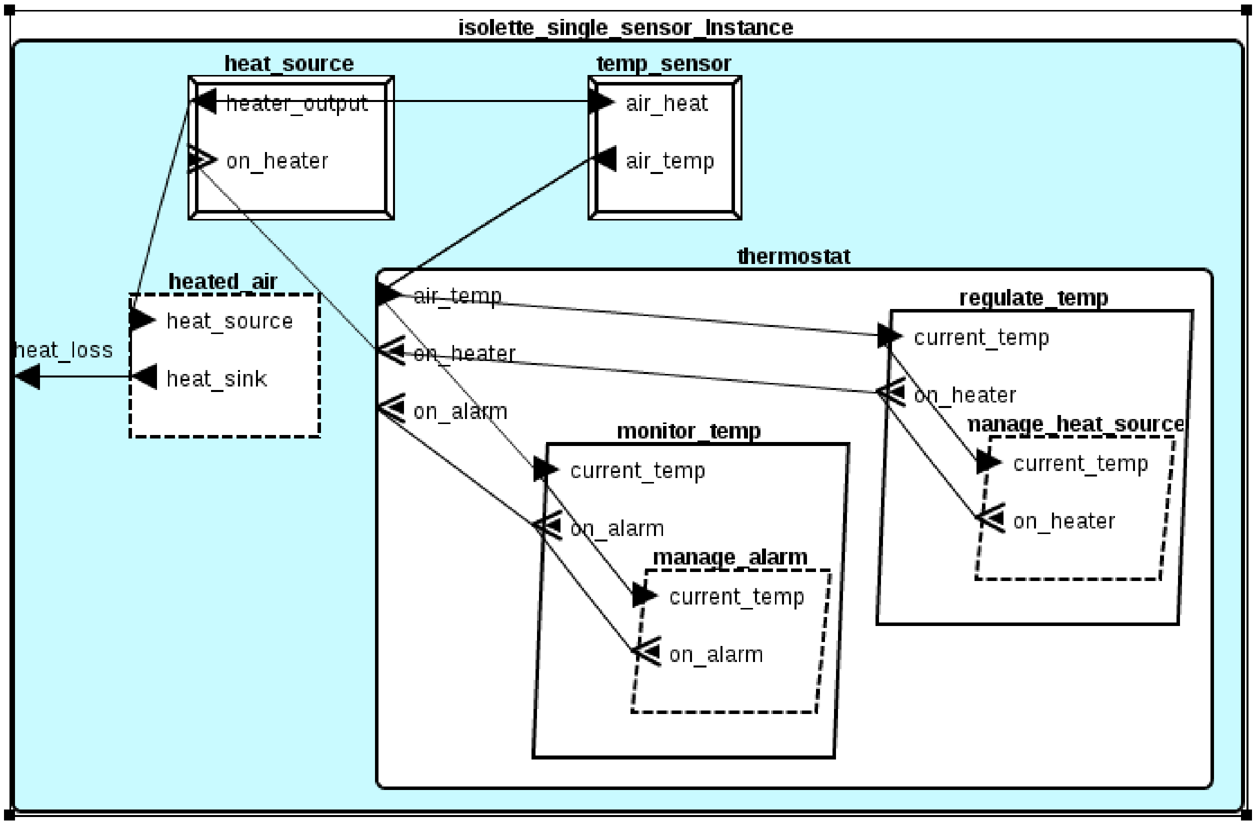
```
data sensed_temperature
  properties
  Data_Model::Data_Representation => Struct;
  Data_Model::Element_Names => ("temp","valid");
  Data_Model::Base_Type => (classifier (sensed_temperature_range), classifier (valid_flag));
end sensed_temperature;


data valid_flag
  properties
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => ("Invalid","Valid");
end valid_flag;


data sensed_temperature_range
  properties
    Data_Model::Real_Range => 68.0 .. 105.0;
    Data_Model::Measurement_Unit => "Fahrenheit";
end sensed_temperature_range;
```



Version 08: Abstract Air component in role as controlled block of Isolette control loop

Isolette.aadl	Isolette.aadl

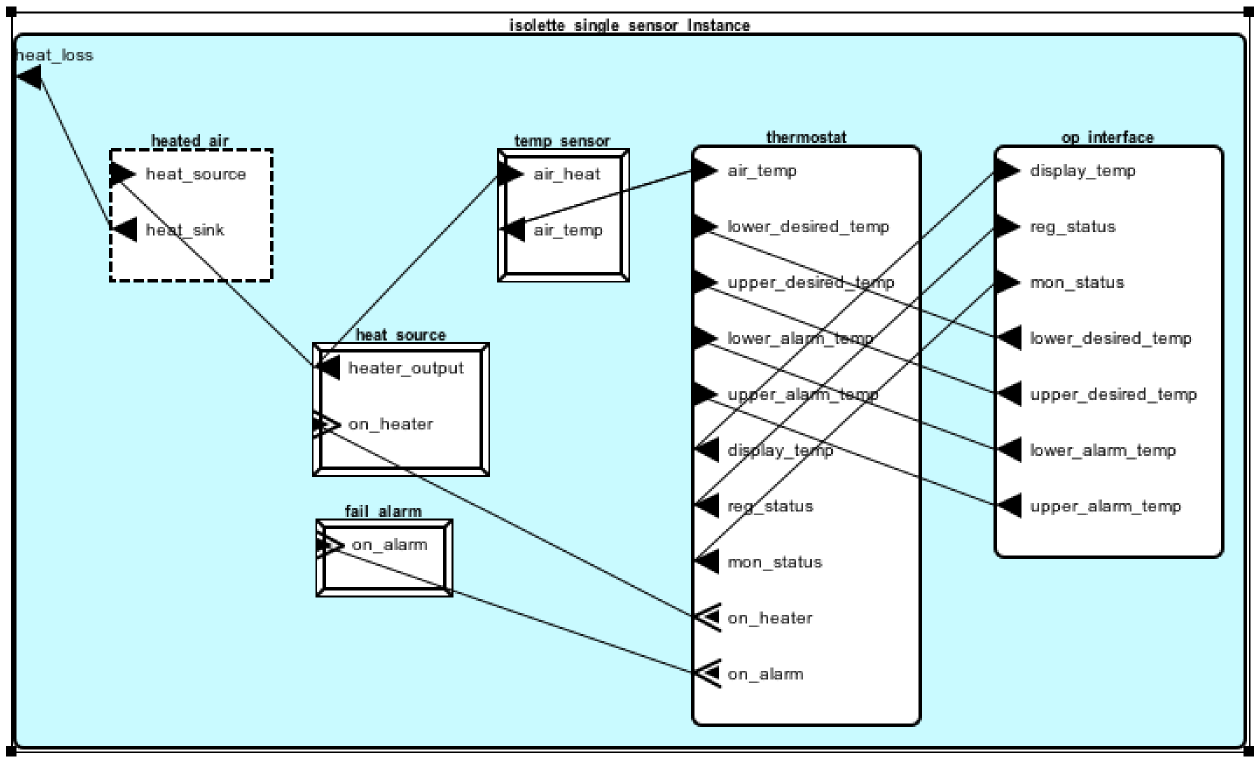
`system implementation isolette.single_sensor`
`  subcomponents`
`    ...`
`    heated_air: abstract air_to_heat.impl;`
 	
`  connections`
`    ...`
`    h1: port heat_source.heater_output ->`
`          temp_sensor.air_heat;`
`    h2: port heat_source.heater_output ->`
`          heated_air.heat_source;`
`    h3: port heated_air.heat_sink ->`
`          heat_loss;`
`...`
	`abstract air_to_heat`
	`  features`
	`    heat_source: in data port Iso_Types::heat;`
	`    heat_sink: out data port Iso_Types::heat;`
	`end air_to_heat;`
	
	`abstract implementation air_to_heat.impl`
	`  -- Model continuous behavior w/hybrid annex.`
	
	`  connections`
	`    h1: port heat_source -> heat_sink;`
	`end air_to_heat.impl;`



Version 09: Software for Thermostat regulating & monitoring control functionality

Regulate.aadl

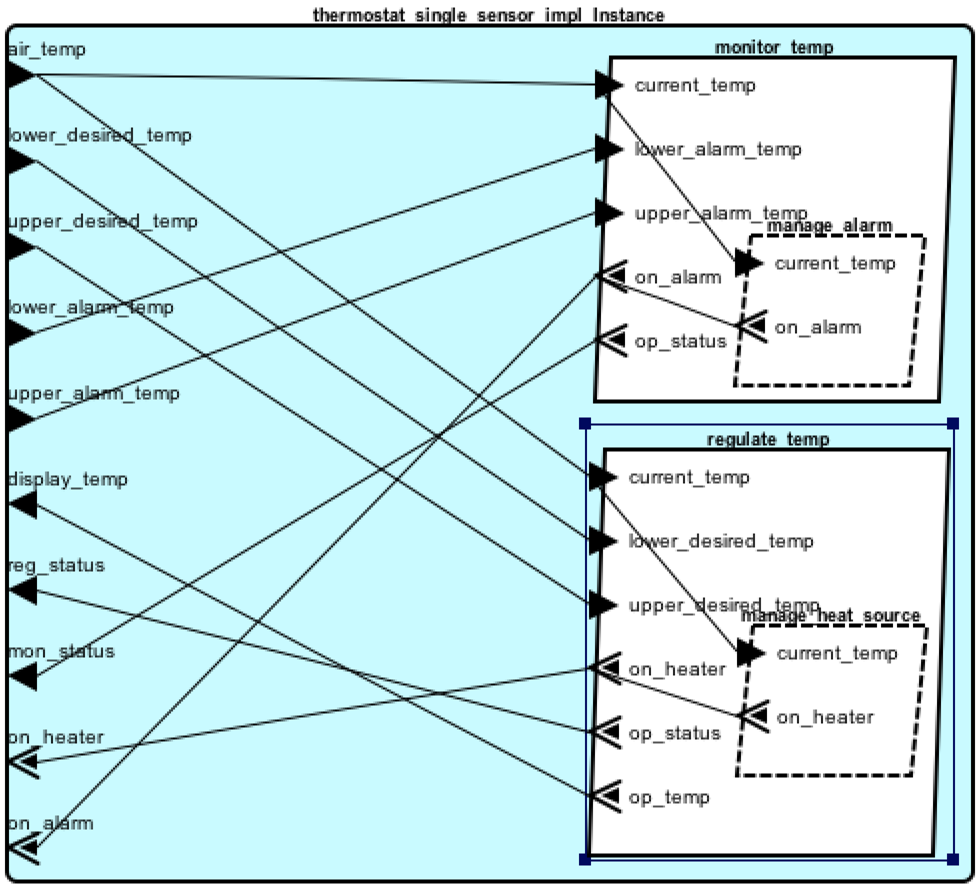
```
process regulate_temperature
  features
    current_temp: in data port Iso_Types::sensed_temperature;
    on_heater: out event data port Iso_Types::on_off;
end regulate_temperature;

process implementation regulate_temperature.impl
  subcomponents
    manage_heat_source: thread manage_heat_source.impl;

  connections
    c1: port current_temp -> manage_heat_source.current_temp;
    c2: port manage_heat_source.on_heater -> on_heater;
end regulate_temperature.impl;


thread manage_heat_source
  features
    current_temp: in data port Iso_Types::sensed_temperature;
    on_heater: out event data port Iso_Types::on_off;
end manage_heat_source;

thread implementation manage_heat_source.impl
end manage_heat_source.impl;
```



Version 10: Operator Interface with temperature settings, display & alarm devices

Isolette.aadl

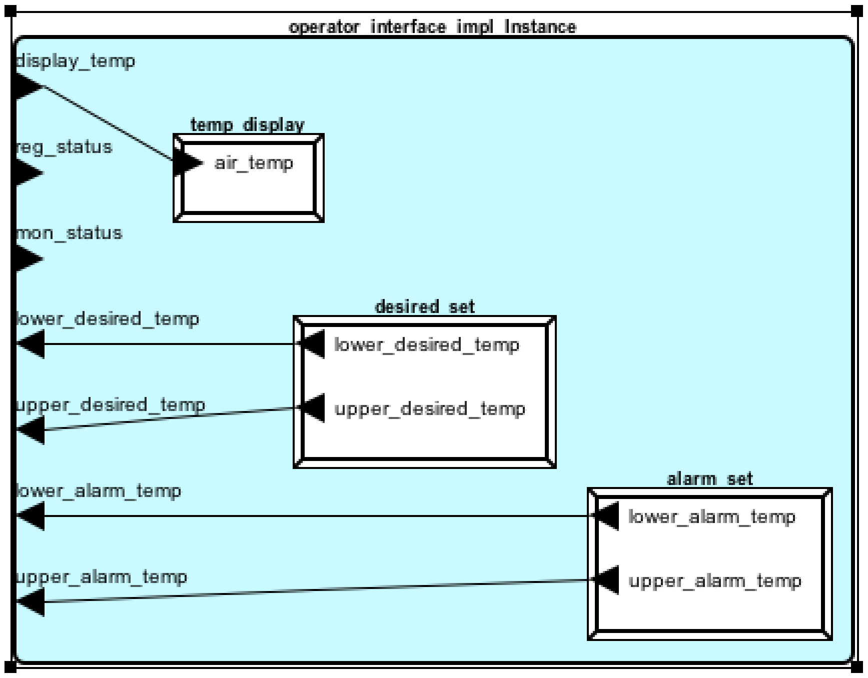
```
system implementation isolette.single_sensor
  subcomponents
    thermostat: system thermostat_single_sensor.impl;
    heat_source: device Devices::heat_source.impl;
    temp_sensor: device Devices::temperature_sensor.impl;
    heated_air: abstract air_to_heat.impl;
    fail_alarm: device Devices::failure_alarm.impl;
    op_interface: system operator_interface.impl;

  connections
    t1: port temp_sensor.air_temp -> thermostat.air_temp;
    t2: port thermostat.on_heater -> heat_source.on_heater;
    t3: port thermostat.on_alarm -> fail_alarm.on_alarm;

    h1: port heat_source.heater_output -> temp_sensor.air_heat;
    h2: port heat_source.heater_output -> heated_air.heat_source;
    h3: port heated_air.heat_sink -> heat_loss;

    o1: port op_interface.lower_desired_temp -> thermostat.lower_desired_temp;
    o2: port op_interface.upper_desired_temp -> thermostat.upper_desired_temp;
    o3: port op_interface.lower_alarm_temp -> thermostat.lower_alarm_temp;
    o4: port op_interface.upper_alarm_temp -> thermostat.upper_alarm_temp;

    o5: port thermostat.display_temp -> op_interface.display_temp;
    o6: port thermostat.reg_status -> op_interface.reg_status;
    o7: port thermostat.mon_status -> op_interface.mon_status;
end isolette.single_sensor;
```



Version 10: Thermostat details

Isolette.aadl

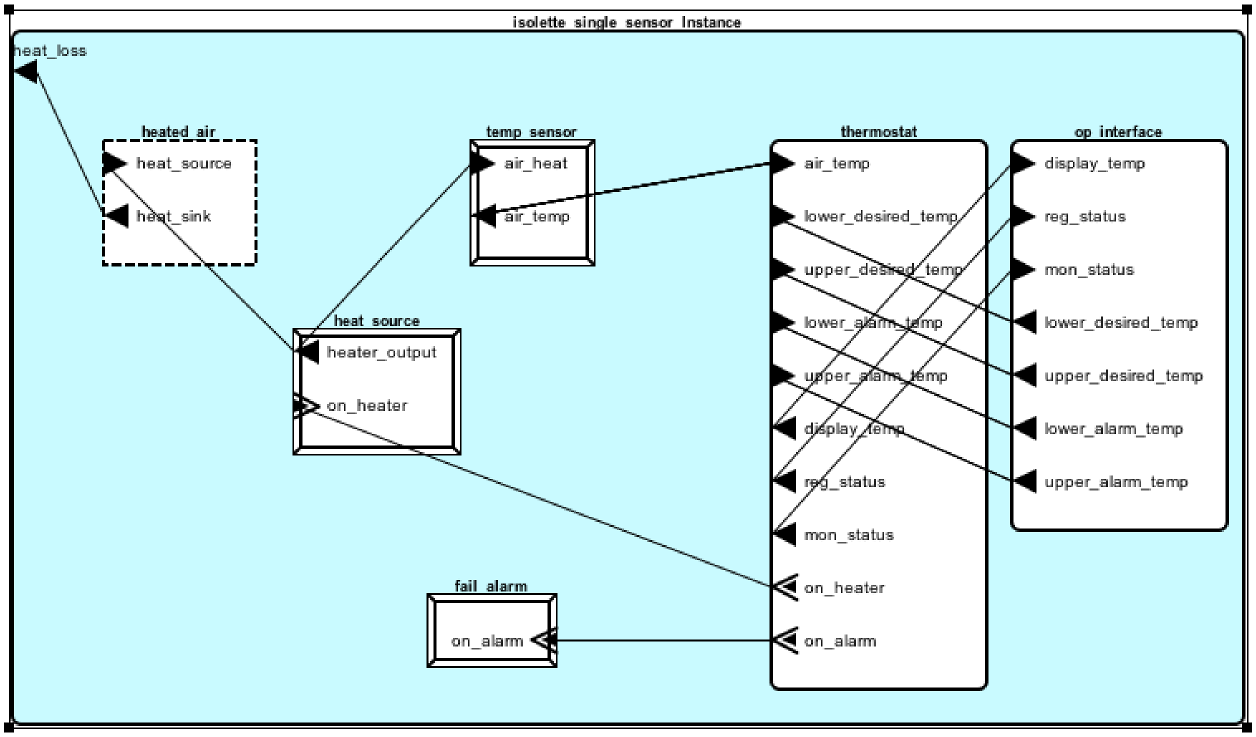
```
system implementation thermostat_single_sensor.impl
  subcomponents
    regulate_temp: process Regulate::regulate_temperature.impl;
    monitor_temp: process Monitor::monitor_temperature.impl;

  connections
    r1: port air_temp -> regulate_temp.current_temp;
    r2: port regulate_temp.on_heater -> on_heater;

    m1: port air_temp -> monitor_temp.current_temp;
    m2: port monitor_temp.on_alarm -> on_alarm;

    d1: port lower_desired_temp -> regulate_temp.lower_desired_temp;
    d2: port upper_desired_temp -> regulate_temp.upper_desired_temp;
    a1: port lower_alarm_temp -> monitor_temp.lower_alarm_temp;
    a2: port upper_alarm_temp -> monitor_temp.upper_alarm_temp;

    c1: port regulate_temp.op_status -> reg_status;
    c2: port monitor_temp.op_status -> mon_status;
    c3: port regulate_temp.op_temp -> display_temp;
end thermostat_single_sensor.impl;
```



Version 10: Operator Interface devices

MapIsolette.aadl (implementation)

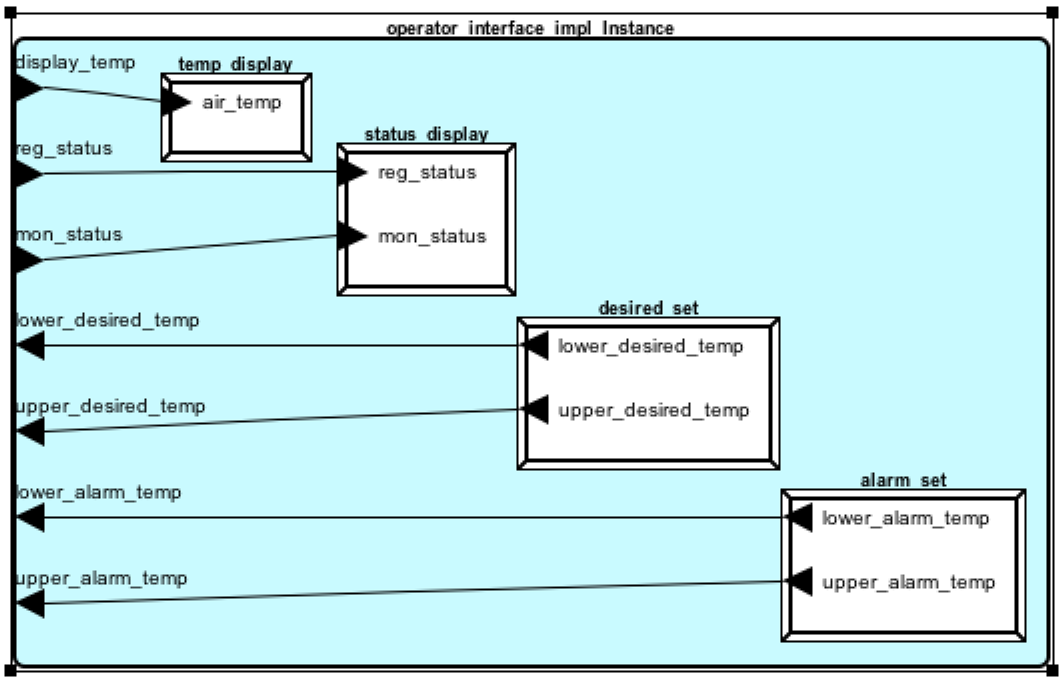
```
system operator_interface
  features
    display_temp: in data port Iso_Types::sensed_temperature;
    reg_status: in data port Iso_Types::op_status;
    mon_status: in data port Iso_Types::op_status;

    lower_desired_temp: out data port Iso_Types::lower_desired_temperature;
    upper_desired_temp: out data port Iso_Types::upper_desired_temperature;
    lower_alarm_temp: out data port Iso_Types::lower_alarm_temperature;
    upper_alarm_temp: out data port Iso_Types::upper_alarm_temperature;
end operator_interface;

system implementation operator_interface.impl
  subcomponents
    temp_display: device Devices::temperature_display.impl;
    status_display: device Devices::status_display.impl;
    desired_set: device Devices::desired_setter;
    alarm_set: device Devices::alarm_setter;

  connections
    t1: port display_temp -> temp_display.air_temp;
    s1: port reg_status -> status_display.reg_status;
    s2: port mon_status -> status_display.mon_status;
    d1: port desired_set.lower_desired_temp -> lower_desired_temp;
    d2: port desired_set.upper_desired_temp -> upper_desired_temp;
    a1: port alarm_set.lower_alarm_temp -> lower_alarm_temp;
    a2: port alarm_set.upper_alarm_temp -> upper_alarm_temp;
end operator_interface.impl;
```



Version 11: Software for regulating & monitoring interface, mode, & failure functionality

Isolette.aadl

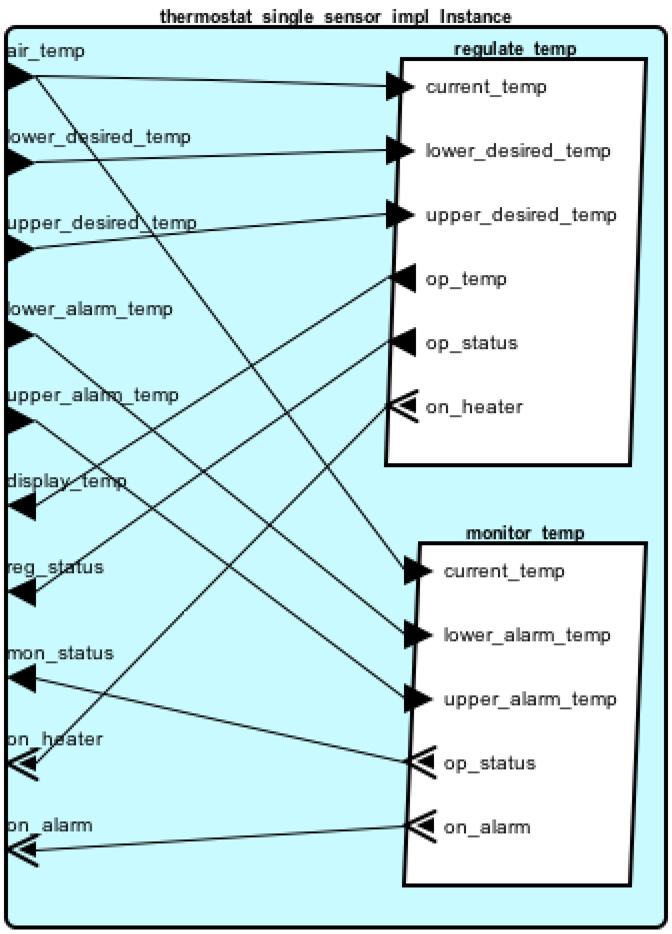
```
system implementation isolette.single_sensor
  subcomponents
    thermostat: system thermostat_single_sensor.impl;
    heat_source: device Devices::heat_source.impl;
    temp_sensor: device Devices::temperature_sensor.impl;
    heated_air: abstract air_to_heat.impl;
    fail_alarm: device Devices::failure_alarm.impl;
    op_interface: system operator_interface.impl;

  connections
    t1: port temp_sensor.air_temp -> thermostat.air_temp;
    t2: port thermostat.on_heater -> heat_source.on_heater;
    t3: port thermostat.on_alarm -> fail_alarm.on_alarm;

    h1: port heat_source.heater_output -> temp_sensor.air_heat;
    h2: port heat_source.heater_output -> heated_air.heat_source;
    h3: port heated_air.heat_sink -> heat_loss;

    o1: port op_interface.lower_desired_temp -> thermostat.lower_desired_temp;
    o2: port op_interface.upper_desired_temp -> thermostat.upper_desired_temp;
    o3: port op_interface.lower_alarm_temp -> thermostat.lower_alarm_temp;
    o4: port op_interface.upper_alarm_temp -> thermostat.upper_alarm_temp;

    o5: port thermostat.display_temp -> op_interface.display_temp;
    o6: port thermostat.reg_status -> op_interface.reg_status;
    o7: port thermostat.mon_status -> op_interface.mon_status;
end isolette.single_sensor;
```



Version 11: Software for regulating & monitoring interface, mode, & failure functionality

Isolette.aadl

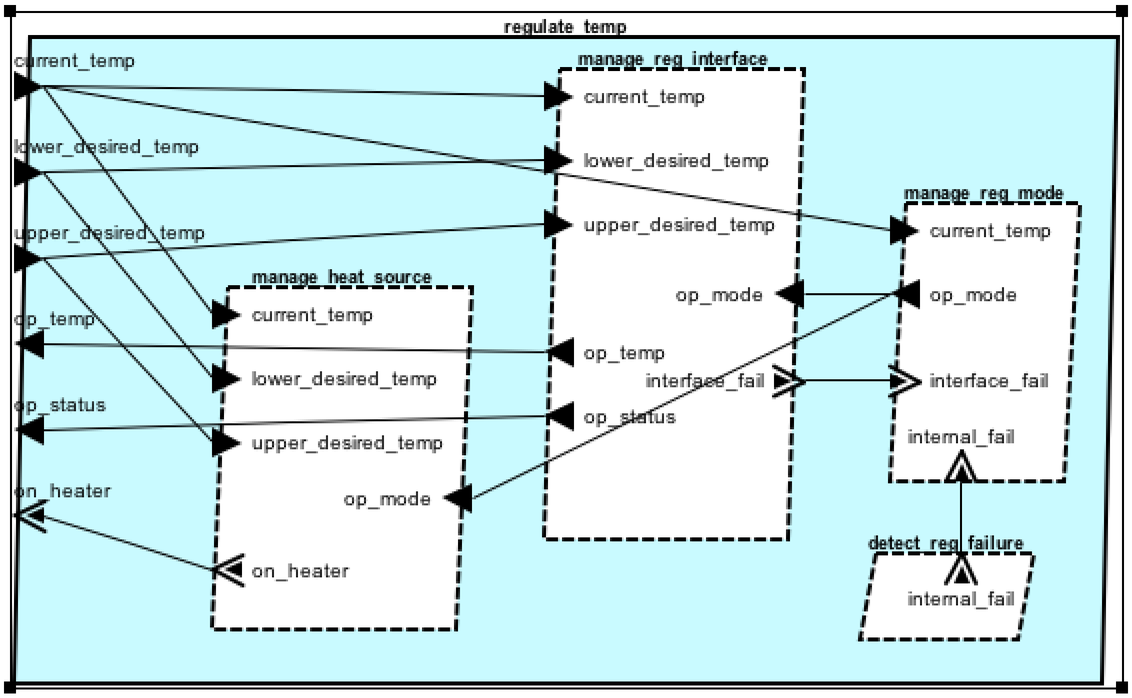
```
system operator_interface
  features
    display_temp: in data port Iso_Types::sensed_temperature;
    reg_status: in data port Iso_Types::op_status;
    mon_status: in data port Iso_Types::op_status;

    lower_desired_temp: out data port Iso_Types::lower_desired_temperature;
    upper_desired_temp: out data port Iso_Types::upper_desired_temperature;
    lower_alarm_temp: out data port Iso_Types::lower_alarm_temperature;
    upper_alarm_temp: out data port Iso_Types::upper_alarm_temperature;
end operator_interface;

system implementation operator_interface.impl
  subcomponents
    temp_display: device Devices::temperature_display.impl;
    status_display: device Devices::status_display.impl;
    desired_set: device Devices::desired_setter;
    alarm_set: device Devices::alarm_setter;

  connections
    t1: port display_temp -> temp_display.air_temp;
    s1: port reg_status -> status_display.reg_status;
    s2: port mon_status -> status_display.mon_status;
    d1: port desired_set.lower_desired_temp -> lower_desired_temp;
    d2: port desired_set.upper_desired_temp -> upper_desired_temp;
    a1: port alarm_set.lower_alarm_temp -> lower_alarm_temp;
    a2: port alarm_set.upper_alarm_temp -> upper_alarm_temp;
end operator_interface.impl;
```



Version 11: Software for regulating & monitoring interface, mode, & failure functionality

Isolette.aadl

```
system implementation thermostat_single_sensor.impl
  subcomponents
    regulate_temp: process Regulate::regulate_temperature.impl;
    monitor_temp: process Monitor::monitor_temperature.impl;

  connections
    r1: port air_temp -> regulate_temp.current_temp;
    r2: port regulate_temp.on_heater -> on_heater;

    m1: port air_temp -> monitor_temp.current_temp;
    m2: port monitor_temp.on_alarm -> on_alarm;

    d1: port lower_desired_temp -> regulate_temp.lower_desired_temp;
    d2: port upper_desired_temp -> regulate_temp.upper_desired_temp;
    a1: port lower_alarm_temp -> monitor_temp.lower_alarm_temp;
    a2: port upper_alarm_temp -> monitor_temp.upper_alarm_temp;

    c1: port regulate_temp.op_status -> reg_status;
    c2: port monitor_temp.op_status -> mon_status;
    c3: port regulate_temp.op_temp -> display_temp;
end thermostat_single_sensor.impl;
```



Version 11: Software for regulating interface, mode, & failure functionality

Isolette.aadl

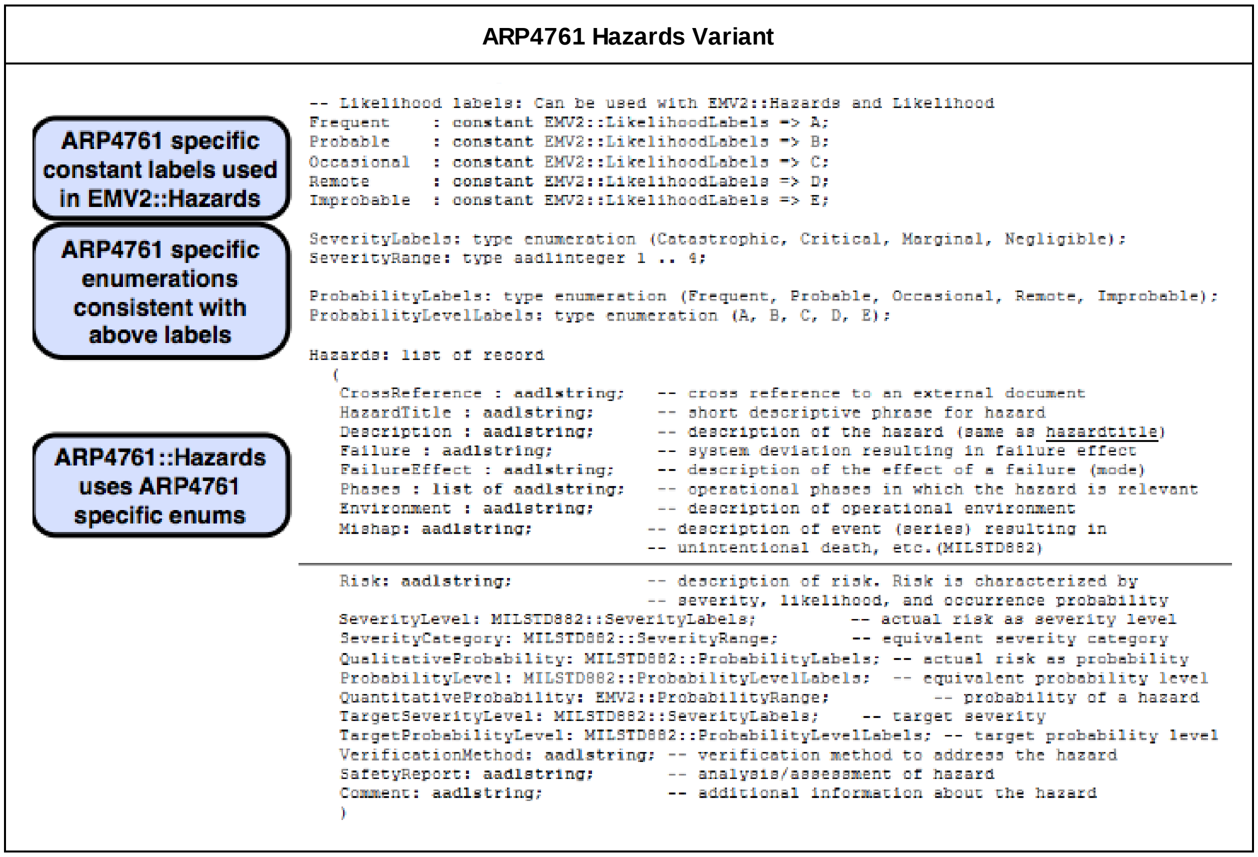
Isolette Modeling Versions, Branches & Analyses

#	System Features	AADL Concepts	prop	flow	EMv2	HyA	Late	FMEA	FTA	FIA	FHA	RBD	Com	UnH	MDCF
01	skeletal Isolette type; single & dual impls.	system type & implementations													
02	skeletal Thermostat	system subcomponents													
03	skeletal physical devices	actuator (device) subcomponents													
04	untyped interfaces	port declarations													
05	types & typed interfaces	port definitions & data modeling													
06	intra-system connections	system & device port connections		x											
07	enumerated & composite types	data modeling		x											
08	Air component controlled block	abstract type & implementation		x	x	x									
09	Thermostat control functions	processes & threads	x	x	x	x									
10	Operator Interface & physical devices	modeling tradeoffs	x	x	x	x									
11	Thermostat interface, mode, & failure function	thread interactions	x	x	x	x									
12	Full REMH	shared data	x	x	x	x									

EMv2 Property Set

EMv2::Hazards

```
Hazards: list of record
(
  CrossReference : aadlstring;   -- cross reference to an external document
  HazardTitle : aadlstring;      -- short descriptive phrase for hazard
  Description : aadlstring;      -- description of the hazard (same as hazardtitle)
  Failure : aadlstring;          -- system deviation resulting in failure effect
  FailureEffect : aadlstring;    -- description of the effect of a failure (mode)
  Phases : list of aadlstring;   -- operational phases in which the hazard is relevant
  Environment : aadlstring;      -- description of operational environment
  Mishap : aadlstring;           -- description of event (series) resulting in
                                 -- unintentional death, etc.(MILSTD882)
  FailureCondition : aadlstring; -- description of event (series) resulting in
                                 -- unintentional death, etc.(ARP4761)
  Risk : aadlstring;             -- description of risk. Risk is characterized by
                                 -- severity, likelihood, and occurrence probability
  Severity : EMV2::SeverityRange ;              -- actual risk as severity
  Likelihood : EMV2::LikelihoodLabels;          -- actual risk as likelihood/probability
  Probability: EMV2::ProbabilityRange;          -- probability of a hazard
  TargetSeverity : EMV2::SeverityRange;         -- acceptable risk as severity
  TargetLikelihood : EMV2::LikelihoodLabels;    -- acceptable risk as likelihood/prob
  DevelopmentAssuranceLevel : EMV2::DALLabels; -- level of rigor in development
                                                -- assurance (ARP4761)
  VerificationMethod : aadlstring; -- verification method to address the hazard
  SafetyReport : aadlstring;       -- analysis/assessment of hazard
  Comment : aadlstring;
)
     applies to ({emv2}**error type, {emv2}**type set, {emv2}**error behavior state,
                {emv2}**error propagation, {emv2}**error event, {emv2}**error flow);
```



ARP4761 Hazards Variant

ARP4761 specific constant labels used in EMV2::Hazards

ARP4761 specific enumerations consistent with above labels

ARP4761::Hazards uses ARP4761 specific enums

```
-- Likelihood labels: Can be used with EMV2::Hazards and Likelihood
Frequent    : constant EMV2::LikelihoodLabels => A;
Probable    : constant EMV2::LikelihoodLabels => B;
Occasional  : constant EMV2::LikelihoodLabels => C;
Remote      : constant EMV2::LikelihoodLabels => D;
Improbable  : constant EMV2::LikelihoodLabels => E;

SeverityLabels: type enumeration (Catastrophic, Critical, Marginal, Negligible);
SeverityRange: type aadlinteger 1 .. 4;

ProbabilityLabels: type enumeration (Frequent, Probable, Occasional, Remote, Improbable);
ProbabilityLevelLabels: type enumeration (A, B, C, D, E);

Hazards: list of record
  (
   CrossReference : aadlstring;   -- cross reference to an external document
   HazardTitle : aadlstring;      -- short descriptive phrase for hazard
   Description : aadlstring;      -- description of the hazard (same as hazardtitle)
   Failure : aadlstring;          -- system deviation resulting in failure effect
   FailureEffect : aadlstring;    -- description of the effect of a failure (mode)
   Phases : list of aadlstring;   -- operational phases in which the hazard is relevant
   Environment : aadlstring;      -- description of operational environment
   Mishap: aadlstring;           -- description of event (series) resulting in
                                 -- unintentional death, etc.(MILSTD882)
--------------------------------------------------------------------------------------
   Risk: aadlstring;             -- description of risk. Risk is characterized by
                                 -- severity, likelihood, and occurrence probability
   SeverityLevel: MILSTD882::SeverityLabels;          -- actual risk as severity level
   SeverityCategory: MILSTD882::SeverityRange;        -- equivalent severity category
   QualitativeProbability: MILSTD882::ProbabilityLabels; -- actual risk as probability
   ProbabilityLevel: MILSTD882::ProbabilityLevelLabels;  -- equivalent probability level
   QuantitativeProbability: EMV2::ProbabilityRange;          -- probability of a hazard
   TargetSeverityLevel: MILSTD882::SeverityLabels;    -- target severity
   TargetProbabilityLevel: MILSTD882::ProbabilityLevelLabels; -- target probability level
   VerificationMethod: aadlstring; -- verification method to address the hazard
   SafetyReport: aadlstring;       -- analysis/assessment of hazard
   Comment: aadlstring;            -- additional information about the hazard
   )
```

Create an STPA variant?

Devices.aadl

```
device heat_source
  features
    heater_output: out data port Iso_Types::heat;

  on_heater: in event data port Iso_Types::on_off;
  annex EMV2
  {**
    use types ErrorLibrary;
    use behavior System_Errors::FailStop;

    error propagations
      on_heater: in propagation {System_Errors::HeatControlError};
    end propagations;

    properties
      EMV2::OccurrenceDistribution =>
          System_Properties::HeatSourceFailure applies to fail;

      EMV2::hazards => (
        [ crossreference => "REMH A.3.2";
          failure => "heat source breaks";
          phases => ("all");
          environment => "infant intensive care";
          description => "mechanical disconnection of heat source";
          comment => "always fails open (off)";
        ]
      ) applies to fail;

      EMV2::Severity => ARP4761::Hazardous applies to fail;

      EMV2::Likelihood => ARP4761::ExtremelyRemote applies to fail;
  **};
end heat_source;
```

System_Errors.aadl

```
annex EMv2
{**
  error types
    HeatControlError: type;
    AlarmError: type;
    FalseAlarm: type extends AlarmError;
    MissedAlarm: type extends AlarmError;
  end types;

  -- FSM for components that source an out of range error and fail completely ("failed").
  error behavior FailStop
    use types ErrorLibrary;

    events
      fail: error event {OutOfRange};
--    fail: error event when "OutOfRange";  [scb] Are these the same?

    states
      working: initial state;  -- Initial/default state of component.
      failed: state;  -- State of component after out of range value error.

    transitions
      working -[ fail ]-> failed;
  end behavior;

  -- FSM for components that source an out of range error and fail completely ("failed"),
  --   or that source an undetectable error and fail intermittently ("flakey").
  error behavior FailSubtle
    use types ErrorLibrary;

    events
      hardfail: error event  -- {OutOfRange};  [scb] Can be untyped, what is the meaning?
      subtlefail: error event {UndetectableValueError, ItemOmission};  [scb] AND, OR, other?

    states
      working: initial state;  -- Initial/default state of component.
      failed: state;  -- State of component after out of range value error.
      flakey: state;  -- State of component after undetectable value error.

    transitions
      working -[ hardfail ]-> failed;
      working -[ subtlefail ]-> flakey;
  end behavior;

  -- FSM to propogate, and possibly, transform errors routed through a composite component.
  error behavior CompositeFailure
    use types System_Errors;

    states
      Operational: initial state;
      ReportedFailure: state {DetectedFault};
      MissedFailure: state {MissedAlarm};
      FalseAlarm: state {FalseAlarm};  --[scb] What does typing a state do?
  end behavior;
**};
```