Parametric and Constraint Modeling

Week 10

Why Model Constraints in Systems Engineering?

Engineering systems are governed by mathematical relationships, performance limits, and physical laws.
Modeling these constraints explicitly enables:

- **Analysis** — evaluate system behavior under different conditions
- **Optimization** — explore trade-offs between design options
- **Verification** — ensure system satisfies performance-related requirements

Without constraints, models are descriptive only.
With constraints, models become predictive and analytical — supporting better design decisions.

How Constraints Connect to Requirements and Structure

Constraints are the bridge between system requirements and system structure.

They enable us to model **how performance requirements are satisfied by the system architecture**.

- Constraints relate **parameters** and **values** in the system structure.
- Constraints can be traced to **non-functional requirements** (e.g., endurance, efficiency, capacity).
- Constraints enable **automated checking** and **analysis** as part of the system model.

This makes parametric modeling an integral part of a traceable, verifiable MBSE approach.

What Is a Constraint?

A Constraint defines a rule or relationship that must hold true between parameters or values in the system.

In SysML v2, we model constraints using:

- constraint def — defines the mathematical form of the constraint
- constraint — applies the constraint to specific elements in the model

Constraints express engineering knowledge directly in the model.

They can be used to support analysis, trade studies, and verification against performance requirements.

Element	Graphical Notation	Textual Notation

Constraint Definition	«constraint def»
	ConstraintDef1
	
	«constraint def»
	ConstraintDef1
	compartment stack
	...
		`constraint def ConstraintDef1;`
		
		`constraint def ConstraintDef1 { /* members */ }`

Constraint	«constraint»
	constraint1 : ConstraintDef1
	
	«constraint»
	constraint1 : ConstraintDef1
	compartment stack
	...
		`constraint constraint1 : ConstraintDef1;`
		
		`constraint constraint1 : ConstraintDef1 { /* members */ }`

Example of Constraint in both Graphic Notation and Textual Notation from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 126

What Is a Calculation?

A Calculation defines how an attribute value is computed based on other attributes in the model.

In SysML v2, we use:

- calc def — defines a reusable calculation logic
- calc — applies the calculation to compute a result in the model

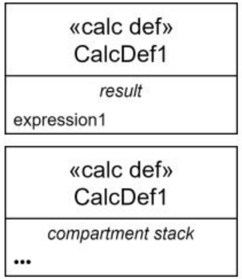
In a calc def:

- in attributes specify the inputs to the calculation
- return specifies the result of the calculation

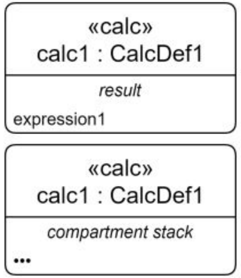
This provides a clear, traceable way to compute derived values within the system model.

What Is a Calculation?

Element	Graphical Notation	Textual Notation

Calc Definition		`calc def CalcDef1 {`
		`expression1`
		`}`
		
		`calc def CalcDef1 {`
		`/* members */`
		`}`

Element	Graphical Notation	Textual Notation

Calc		`calc calc1 : CalcDef1 {`
		`expression1`
		`}`
		
		`calc calc1 : CalcDef1 {`
		`/* members */`
		`}`

Example of Calculation in both Graphic Notation and Textual Notation from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 123

Attributes and Constraints

How They Work Together

In SysML v2, both Constraints and Calculations operate on attributes.

- **Attributes** represent system properties — defined in part definitions (e.g. mass, capacity, speed).
- **Calculations** (calc def) compute new attribute values from existing ones — using in and return.
- **Constraints** (constraint def) define mathematical relationships that must hold true between attributes.

By binding Calculations and Constraints to attributes in the system structure, we enable **traceable and executable analysis**.

Example Calculation and Constraint — Textual Notation

```
constraint def IsFull {
    in tank : FuelTank;
    tank.fuelLevel == tank.maxFuelLevel // Result expression
}
part def Vehicle {
    part fuelTank : FuelTank;
    constraint isFull : IsFull {
        in tank = fuelTank;
    }
}
```

```
calc def Dynamics {
    in initialState : DynamicState;
    in time : TimeValue;
    return : DynamicState;
}
calc def VehicleDynamics specializes Dynamics {
    // Each parameter redefines the corresponding parameter of Dynamics
    in initialState : VehicleState;
    in time : TimeValue;
    return : VehicleState;
}
```

Example from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 124 and page 128

What Is a Binding Connection?

A Binding Connection asserts that two attributes or features must have the same value.

In SysML v2:

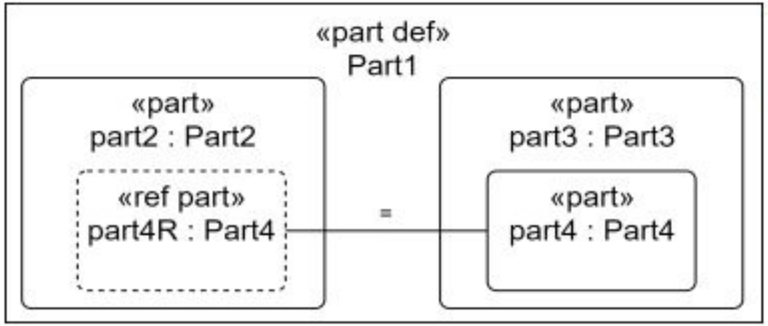
- A **Binding** is defined using bind — connecting two attributes.
- When one value changes, the other is updated automatically.

In Interconnection View:

- Bindings are typically shown between **Attribute Ports** on parts.
- These ports expose the part’s attributes, enabling clear visualization of bindings.

This supports propagation of values between Calculations, Constraints, and system attributes — enabling dynamic analysis in the model.

What Is a Binding Connection?

<table>
<tr><th>Element</th><th>Graphical Notation</th><th>Textual Notation</th></tr>
<tr><td>Binding Connection</td><td>
</td><td><pre>part def Part1 {
   part part2:part2 {
      ref part
part4R:Part4;
   }
   part part3:part3 {
      part part4:Part4;
   }
   bind part2.part4R =
        part3.part4;
}</pre></td></tr>
</table>

Example of Binging Connection from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 66

How Binding Propagates Values

A Binding ensures that two connected attributes remain equal — so changes automatically propagate through the model.

When one bound attribute is updated, the other reflects the change immediately.

This supports:

- **Dynamic analysis** — recalculations when design parameters change
- **Traceability** — visibility of how values impact system behavior
- **Model consistency** — avoiding mismatched or stale values

In Interconnection View, Binding Connections make these dependencies visible between Attribute Ports.

Interconnection View as a Parametric View

In SysML v2, Interconnection View provides a clear way to visualize how attributes are bound and how engineering constraints are applied to system parts.

- Attributes are exposed via **Attribute Ports** on parts.
- **Binding Connections** show how attributes are linked across parts.
- **Calculations** and **Constraints** can be applied and traced through these connections.

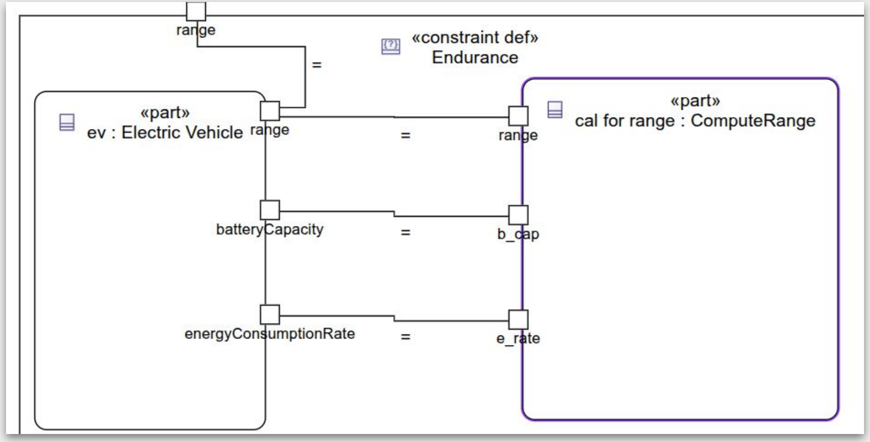
This enables a modern, specification-compliant approach to parametric modeling — fully integrated with the system structure.

Example: Binding Vehicle Attributes in Interconnection View

Scenario: Modeling how a Vehicle's **computed range** is linked to system attributes and constraints.

Elements involved:

- vehicle.batteryCapacity and vehicle.energyConsumptionRate — input attributes
- calc computeRange — calculates vehicle.range
- constraint enduranceConstraint — checks if vehicle.range meets required endurance
- **Binding Connections** link all these attributes



In Interconnection View:

- Attribute Ports expose these attributes on the Vehicle part and related elements
- Binding Connections show how values propagate across the model

Constraint Propagation: Supporting Analysis and Sensitivity

Constraint propagation ensures that changes in system attributes automatically update related calculations and constraints.

When an attribute bound in a Calculation or Constraint changes:

- The Calculation is re-evaluated
- The Constraint is re-checked
- Dependent values propagate through the model

This enables:

- **What-if analysis** — exploring design alternatives
- **Sensitivity analysis** — understanding which factors impact performance
- **Dynamic validation** — keeping the model consistent as designs evolve

Using Constraints to Support Trade Studies

Constraints enable systematic evaluation of design alternatives by embedding engineering logic into the model.

With constraints in place, we can:

- **Vary design inputs** — such as battery size, payload mass, or aerodynamic profile
- **Automatically update derived attributes** through Calculations and Bindings
- **Evaluate whether design variants satisfy performance Constraints**
- **Compare alternatives** based on objective criteria

This turns the model into a decision-support tool for engineering trade studies.

Example Scenarios

Battery Size vs. Range vs. Payload

Typical engineering questions that Constraints can help analyze:

- How does **battery capacity** impact **vehicle range**?
- How does increasing **payload mass** affect **energy consumption** and **range**?
- What is the optimal balance between **battery size**, **payload capacity**, and **total mass**?
- Can the system still meet **endurance requirements** with different payload configurations?

By modeling these relationships explicitly, we can:

- Perform **what-if scenarios**
- Guide **design optimization**
- Validate **requirement compliance** under varying conditions

Practice

Model Constraint and Bindings in Vehicle System

In this practice, we will model a simple engineering constraint for a Vehicle System.

Scenario: Electric Vehicle part with attributes:

- batteryCapacity :> ScalarValues::Integer
- energyConsumptionRate :> ScalarValues::Integer
- range :> ScalarValues::Integer

Model elements:

- calc def computeRange — computes range from battery capacity and energy consumption
- constraint def enduranceConstraint — checks that range >= requiredEndurance
- **Binding Connections** link attributes to Calculation and Constraint

Practice instructions:

- Define calc def and constraint def in textual model
- Apply cal and constraint to Vehicle part **(note, we will use part instead of cal in SysOn)**
- Use **Interconnection View** to create and visualize the **Binding Connections**

How to Apply Constraints to Your Drone System

In your Drone System project, you can apply the same modeling pattern to performance-related requirements.

Typical examples:

- **Flight endurance** as a function of battery capacity, payload mass, and power consumption
- **Payload impact** on energy consumption and center of gravity
- **Takeoff performance** under varying payload and battery configurations

Approach:

- Define **Calculations** for derived performance attributes
- Define **Constraints** to check compliance with functional or non-functional requirements
- Use **Bindings** to connect Calculations and Constraints to system attributes
- Visualize relationships in **Interconnection View**

Tracing Constraints to Non-Functional Requirements

Constraints provide a direct way to verify that the system satisfies non-functional requirements (NFRs).

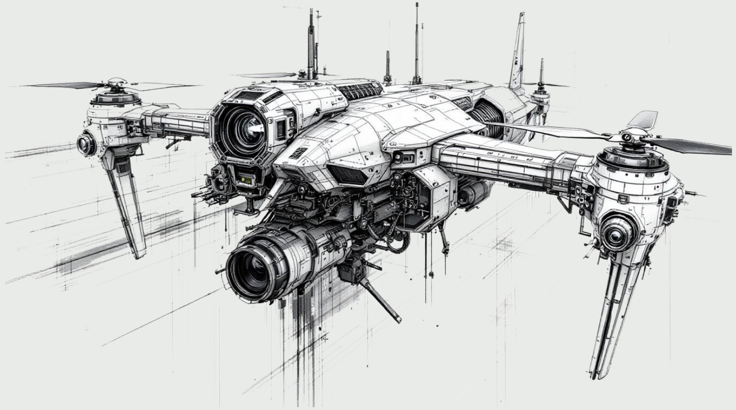
- Many NFRs can be expressed mathematically (e.g., endurance ≥ X minutes, weight ≤ Y kg).
- Constraints capture these relationships in the model.
- By binding system attributes to Constraints, compliance can be checked dynamically.
- Traceability links Constraints back to the originating NFRs.

This creates a complete chain: Requirement → Constraint → System Attributes → Verification.

Add Constraints to Your Drone Model

For your Drone System project, apply this week's modeling patterns to relevant performance requirements:

- Identify **at least one non-functional requirement** that can be expressed mathematically (e.g., endurance, payload capacity, weight limit).
- Define a corresponding **constraint def** and **constraint** in your model.
- If needed, define a **calc def** and **calc** to compute any derived attributes.
- Use **Binding Connections** to link system attributes, Calculations, and Constraints.
- Visualize the relationships in **Interconnection View**.



Summary of Week 10

This week, we learned how to model **engineering constraints** in SysML v2 to support analysis, trade studies, and performance verification:

- **Calculations** (calc def and calc) compute derived attributes from system attributes.
- **Constraints** (constraint def and constraint) express engineering rules that must be satisfied.
- **Binding Connections** ensure that attributes are linked and values propagate automatically.
- **Interconnection View** provides a clear visualization of how Calculations and Constraints connect to system structure.
- **Constraints trace to non-functional requirements**, enabling dynamic verification within the model.

QUESTION!