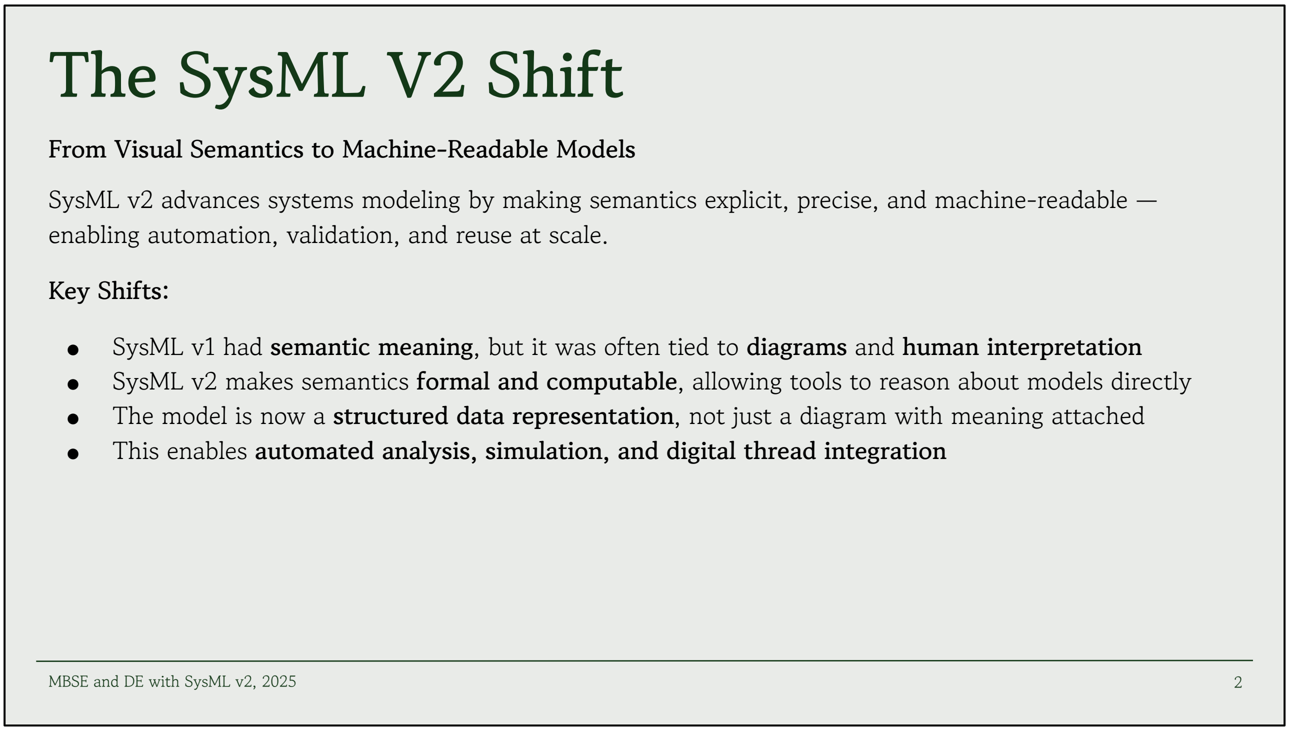
Introduction to SysML V2

Week 03



The SysML V2 Shift

From Visual Semantics to Machine-Readable Models

SysML v2 advances systems modeling by making semantics explicit, precise, and machine-readable — enabling automation, validation, and reuse at scale.

Key Shifts:

- SysML v1 had **semantic meaning**, but it was often tied to **diagrams** and **human interpretation**
- SysML v2 makes semantics **formal and computable**, allowing tools to reason about models directly
- The model is now a **structured data representation**, not just a diagram with meaning attached
- This enables **automated analysis, simulation, and digital thread integration**

Key Differences in Philosophy vs SysML v1: From Visual Semantics to Machine-Readable Models
In SysML v1, we were already working with semantic meaning — blocks represented system components, behaviors showed functional flows, and relationships carried intent. But here's the catch: that semantic information was primarily captured in **diagrams**, and interpreting it correctly often depended on **human understanding** or informal team conventions. Tools could read the diagrams, but **not fully reason about the meaning behind them**.
SysML v2 changes the game. It transforms modeling into a **data-centric, semantically precise language**. Every element in the model — whether it's a part, an action, or a requirement — exists in a **structured, machine-readable form**. This means your models are no longer just visual guides; they're **computable system descriptions** that tools can analyze, simulate, validate, and transform programmatically.
This shift enables automation, advanced verification, and integration with the broader **Digital Thread**. In short, SysML v2 moves systems modeling from *"draw it and explain it"* to *"model it once, reuse it everywhere."*

SysML V2 - Modeling as a First-Class Citizen

With SysML v2, the model itself becomes the authoritative source, equally accessible through diagrams and text, ensuring clarity and consistency.
Textual and graphical modeling are fully integrated

- **No redundancy**—diagrams and text reflect the same model
- Enables **rigorous analysis, traceability, and reuse**
- Designed for **machine processing** and **system automation**

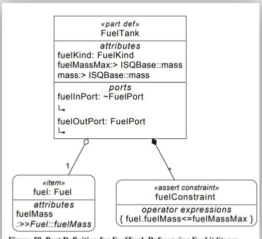
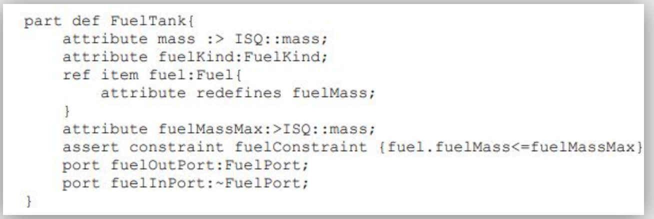


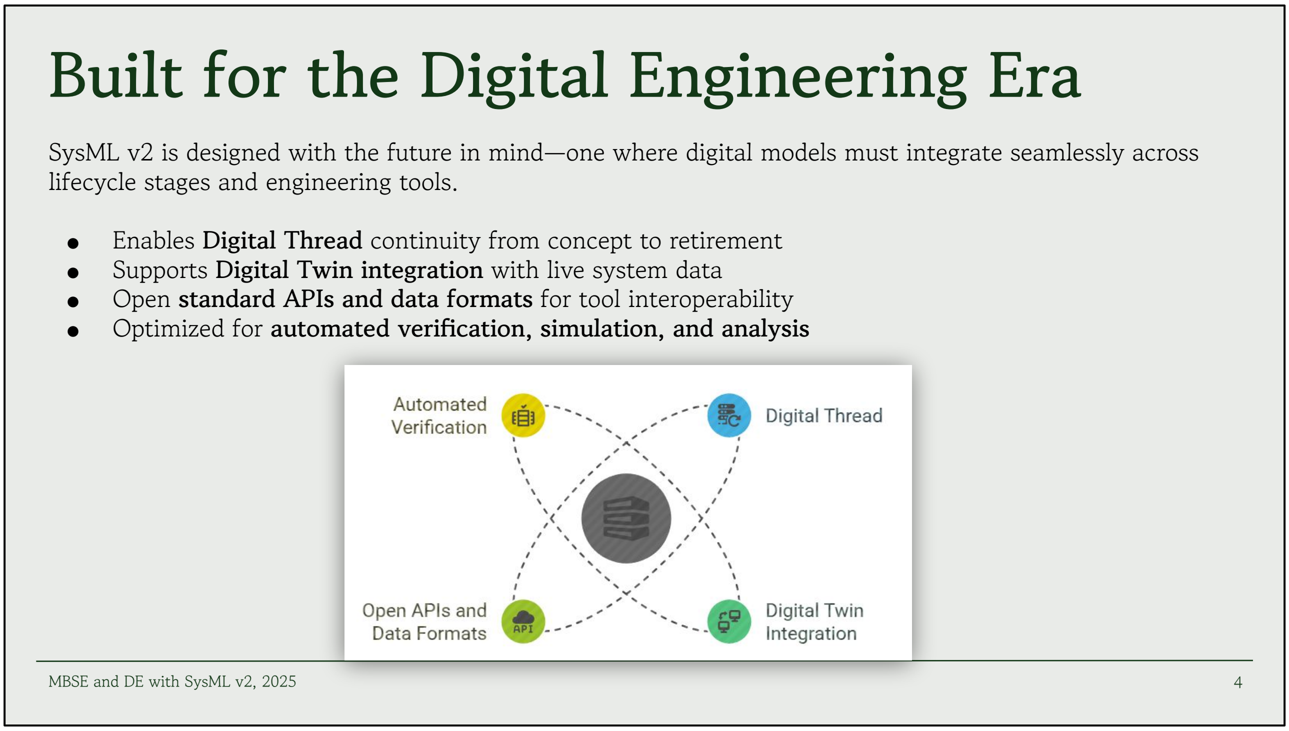
Figure 58. Part Definition for FuelTank Referencing Fuel it Stores



```
part def FuelTank{
    attribute mass :> ISQ::mass;
    attribute fuelKind:FuelKind;
    ref item fuel:Fuel{
        attribute redefines fuelMass;
    }
    attribute fuelMassMax:>ISQ::mass;
    assert constraint fuelConstraint {fuel.fuelMass<=fuelMassMax}
    port fuelOutPort:FuelPort;
    port fuelInPort:~FuelPort;
}
```

Image from OMG Systems Modeling Language™ (SysML®) Version 2.0 Beta 3 (Release 2025-02), Page 638.

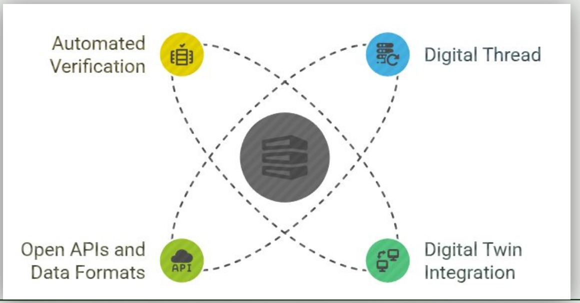
Key Differences in Philosophy vs SysML v1: SysML v2 — Modeling as a First-Class Citizen.
This slide captures one of the most important shifts in mindset: in SysML v1, the **modeling process revolved around diagrams**. We used them to communicate structure, behavior, and relationships — but those diagrams often lacked **formal structure underneath**. They helped humans understand the system, but they didn't give tools enough to **analyze or manipulate the model with precision**.
SysML v2 promotes the model itself as a **first-class citizen** — not the diagram. Every element in the model exists as a **structured semantic object**.
Diagrams in SysML v2 are no longer the source of truth — they're just **views of the underlying data model**.
This shift has massive implications. It enables **machine reasoning**, **automated transformation**, **cross-tool consistency**, and **reusability**.
In short, you're not just drawing — you're building a model that can drive **simulation, traceability, verification, and digital continuity**. This is what makes SysML v2 a foundational language for modern, data-driven engineering practices.



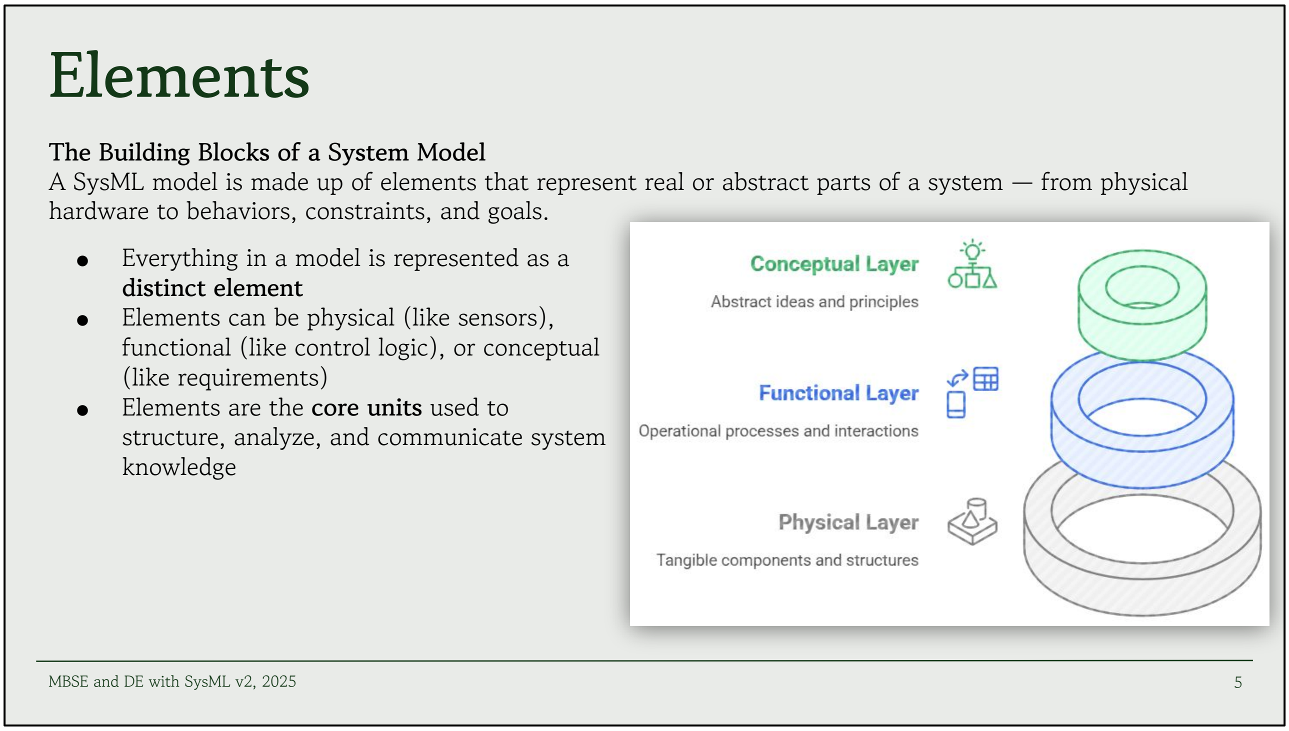
Built for the Digital Engineering Era

SysML v2 is designed with the future in mind—one where digital models must integrate seamlessly across lifecycle stages and engineering tools.

- Enables **Digital Thread** continuity from concept to retirement
- Supports **Digital Twin integration** with live system data
- Open **standard APIs and data formats** for tool interoperability
- Optimized for **automated verification, simulation, and analysis**



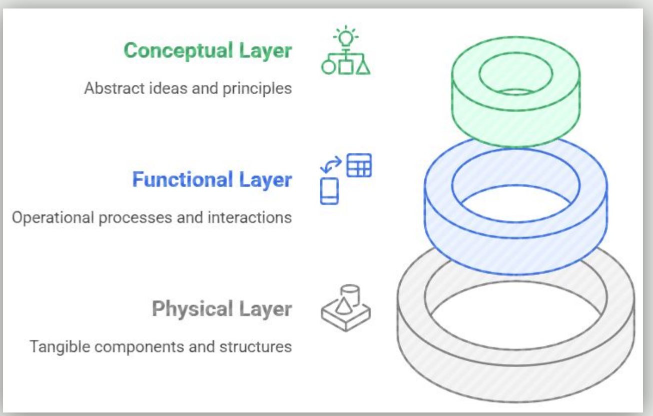
Key Differences in Philosophy vs SysML v1: Built for the Digital Engineering Era
SysML v2 isn't just a language update — it's a foundational shift that aligns systems modeling with the needs of the **digital engineering era**.
In today's systems, data can no longer live in isolated documents or static diagrams. We need a modeling approach that connects everything — from early requirements to late-stage operations — in one **integrated, traceable, and machine-readable ecosystem**.
SysML v2 enables this by supporting the **Digital Thread** — a concept where consistent model data flows across the entire system lifecycle: from concept, to development, to production, to operation, and even into retirement.
It also aligns with the idea of the **Digital Twin** — where live data from physical systems can inform and update the model in real time.
What makes this possible is SysML v2's **modular and open architecture**. Models built in SysML v2 can interface with tools for requirements management, simulation, formal analysis, AI reasoning, and even certification workflows. This isn't just a language for engineers — it's a **language for systems decision-making across disciplines and lifecycles**.



Elements

The Building Blocks of a System Model
A SysML model is made up of elements that represent real or abstract parts of a system — from physical hardware to behaviors, constraints, and goals.

- Everything in a model is represented as a **distinct element**
- Elements can be physical (like sensors), functional (like control logic), or conceptual (like requirements)
- Elements are the **core units** used to structure, analyze, and communicate system knowledge

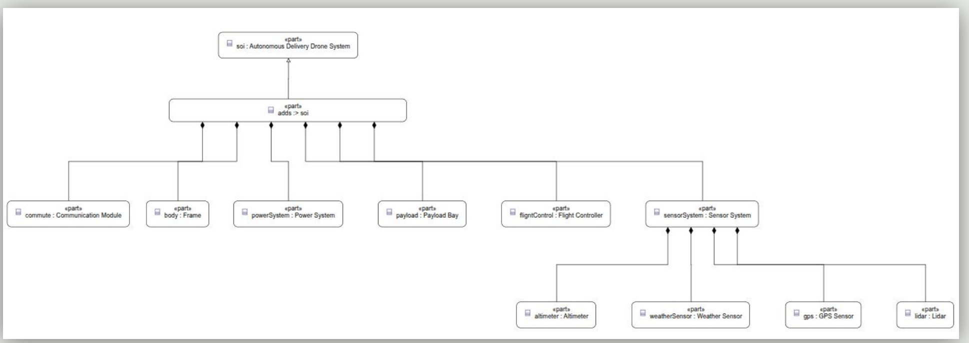

Conceptual Layer
Abstract ideas and principles
Functional Layer
Operational processes and interactions
Physical Layer
Tangible components and structures

Core Modeling Concepts: Elements — The Building Blocks of a System Model
Everything in SysML v2 starts with **elements**. Whether you're defining a physical object like a *sensor*, a logical function like *navigation*, or a design constraint like *must operate below 80°C*, you're modeling it as an element.
Think of elements as the **basic vocabulary of the modeling language**. They're the atomic units that we use to build up everything — structure, behavior, requirements, analysis, even test cases. When you create a model, you're really defining and organizing these elements into a meaningful system representation.This concept is foundational to SysML v2. It ensures that every part of your system — tangible or abstract — can be described in a consistent, formalized way. So no matter which domain you're working in, you're always operating on **structured elements** that can be reused, traced, and validated across the model.

Structure

Modeling What the System *Is*
Structural modeling allows us to define the system's architecture — its parts, interfaces, and how they fit and work together.

- Models the **components** of a system and their **interconnections**
- Essential for understanding **physical layout, interfaces, and system boundaries**
- Forms the **foundation** for analysis, traceability, and simulation

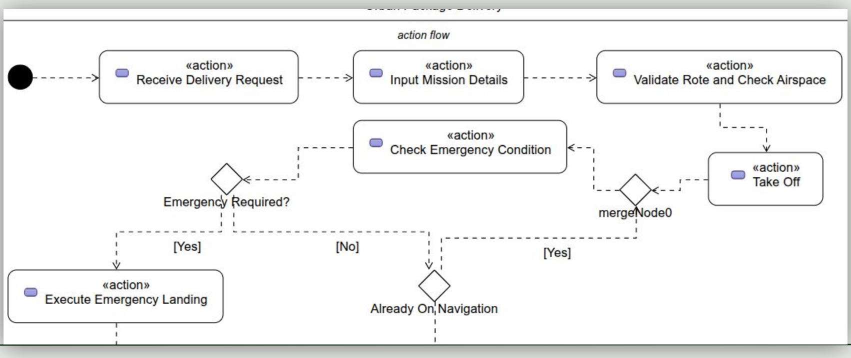

«part» soi : Autonomous Delivery Drone System
«part» adds :> soi
«part» commute : Communication Module
«part» body : Frame
«part» powerSystem : Power System
«part» payload : Payload Bay
«part» flightControl : Flight Controller
«part» sensorSystem : Sensor System
«part» altimeter : Altimeter
«part» weatherSensor : Weather Sensor
«part» gps : GPS Sensor
«part» lidar : Lidar

Core Modeling Concepts: Structure — Modeling What the System *Is*
Structure is the first thing most engineers think about when modeling a system. It's how we answer the question: **"What is the system made of?"** In SysML v2, structural modeling gives us the ability to **decompose complexity** — to break a system down into logical parts, physical modules, interfaces, and environments.
But this isn't just about drawing boxes and lines. It's about clearly defining **boundaries**, **responsibilities**, and **relationships** between components — so that everyone, from design to test, is working with the same architectural understanding.
Good structural modeling supports **reuse** — so you can apply the same modules across multiple systems. It improves **testability** by making dependencies and interfaces explicit. And it gives teams across disciplines a **shared language** for collaboration.
In SysML v2, structure becomes more than a sketch — it becomes a **formal blueprint** of how your system is physically and logically composed.

Behavior

Modeling What the System *Does*
While structure defines what a system *is*, behavior modeling describes how it acts, reacts, and evolves over time.

- Captures **operations**, **responses to events**, and **workflows**
- Supports the modeling of **state changes**, **scenarios**, and **missions**
- Crucial for **simulation, control logic, and performance analysis**

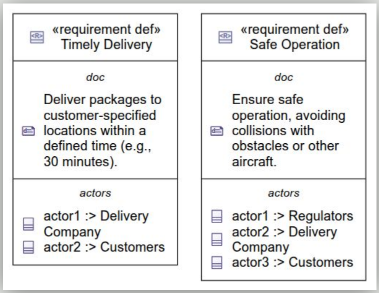

action flow
«action» Receive Delivery Request
«action» Input Mission Details
«action» Validate Rote and Check Airspace
«action» Check Emergency Condition
«action» Take Off
Emergency Required?
mergeNode0
[Yes]
[No]
[Yes]
«action» Execute Emergency Landing
Already On Navigation

Core Modeling Concepts: Behavior — Modeling What the System *Does*
While structure tells us *what a system is*, behavior modeling answers the question: **"What does the system do over time?"** This is where we capture the **dynamic aspects** — how the system reacts, responds, and flows through its operations.
For example, what happens when a **sensor detects an obstacle**? What's the system's **response pattern**? How does it **transition from standby to active mode**, or handle failure recovery? These kinds of questions are modeled using **behavior elements**.
Behavior modeling in SysML v2 supports everything from **high-level scenarios** — like mission sequences — down to **low-level action logic**, like control loops or signal flow. And the best part is, these behaviors are modeled in a way that's **machine-readable**, so they can be **simulated, validated, and reused**.
This is especially critical in **autonomous systems**, **real-time control**, or **safety-critical applications**, where understanding and verifying the system's behavior isn't just helpful — it's mission-essential.

Requirements

Modeling *Why* the System Exists
Requirements define the purpose and constraints of a system — they represent the "why" behind what we build.

- Represent **goals**, **needs**, and **conditions to satisfy**
- Provide a **bridge between stakeholders and engineers**
- Enable **traceability** from concept through implementation and testing



«requirement def»
Timely Delivery

doc

Deliver packages to customer-specified locations within a defined time (e.g., 30 minutes).

actors

actor1 :> Delivery Company
actor2 :> Customers

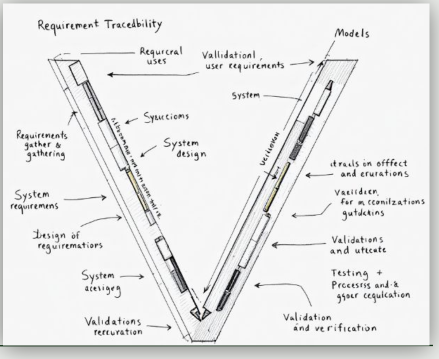
«requirement def»
Safe Operation

doc

Ensure safe operation, avoiding collisions with obstacles or other aircraft.

actors

actor1 :> Regulators
actor2 :> Delivery Company
actor3 :> Customers

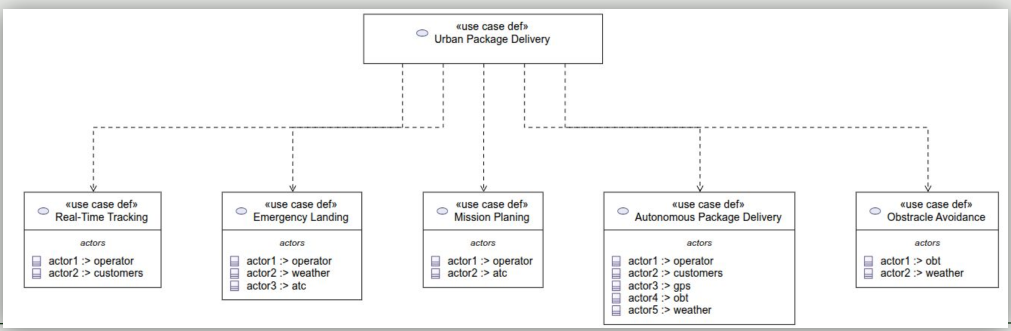

Requirement Tracedbility
Models
Requrcral uses
Vallidationl, user requirements
System
Syeccioms
System design
Requirements gather & gathering
Vericovon
itrails in offfect and erurations
System requiremens
Vaeirdeen, for m cconilzations gutdekins
Design of reguirematiors
Validations and utecate
System acesigeg
Testing + Processs and a gsoer cegulcation
Validations reecuration
Validation and verification

Core Modeling Concepts: Requirements — Modeling Why the System Exists
Requirements define the **mission and purpose** of the system. They capture what stakeholders expect — what the system must do, **how well** it must perform, and **under what conditions** it must operate.
In traditional projects, requirements often live in static documents, isolated from the design. But in SysML v2 — and in MBSE as a whole — we model requirements directly **within the system model**. This allows us to create **explicit links** between requirements and the design elements that fulfill them, and the tests that verify them.
This traceability ensures that nothing falls through the cracks. We gain **clarity**, **accountability**, and most importantly — the ability to check if what we're building actually meets the original goals. Modeling requirements this way also supports **impact analysis**: if a requirement changes, we can instantly see which parts of the system are affected.
This shift is essential for systems that are complex, regulated, or safety-critical — where **compliance isn't optional**, and **alignment must be provable** across the entire lifecycle.

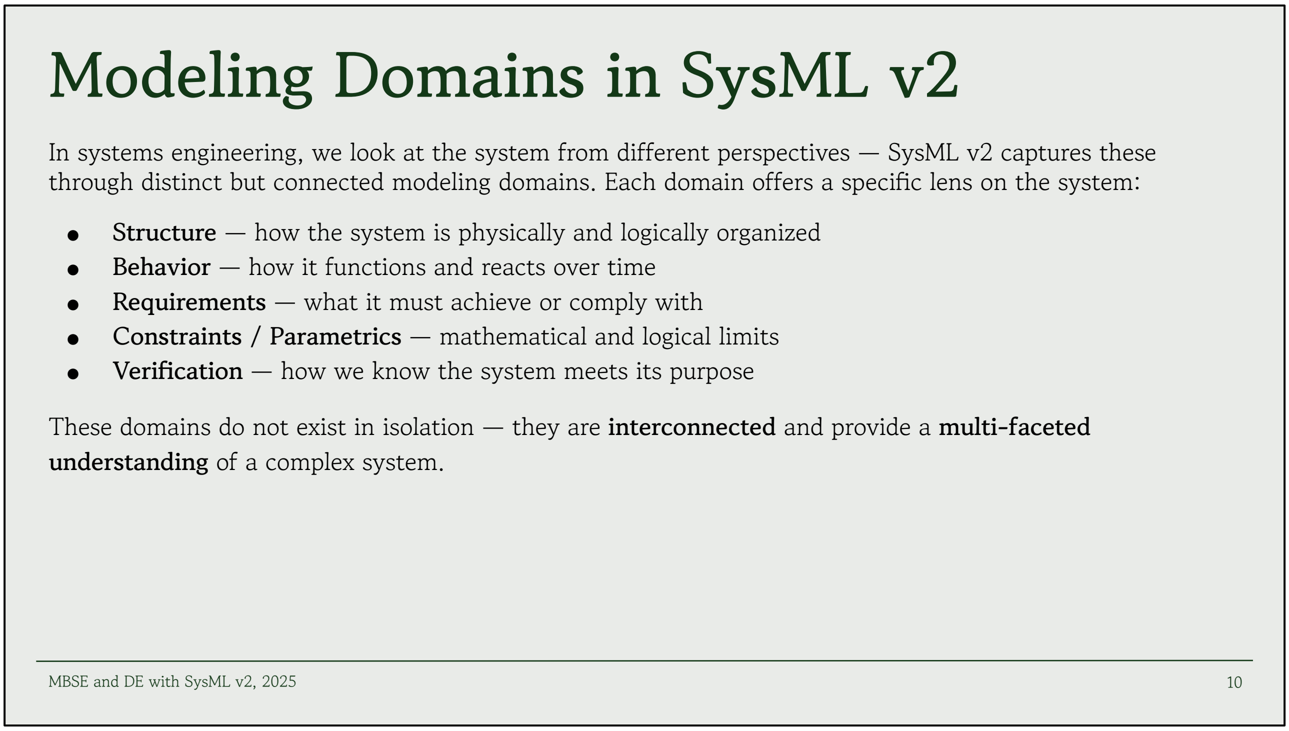
Relationships

Making the Model Cohesive
Relationships connect all parts of the model, enabling consistency, structure, and meaning across system elements.

- Link structure to behavior, behavior to requirements, and more
- Create **traceable, navigable paths** between design decisions
- Support **reuse**, **refinement**, and **impact analysis**



Core Modeling Concepts: Relationships — Making the Model Cohesive
A model is not just a collection of elements — it becomes powerful when those elements are **connected through relationships**. Relationships are what turn your system into a **cohesive, meaningful architecture**.
They allow you to trace how a **requirement leads to a function**, how that function is **realized by a component**, and how that component **interacts with others**. This web of relationships reflects the real-world dependencies in your system.
In SysML v2, these connections are **explicit and formal**. That means they're not just visual — they're part of the model's structure. You can **query them**, **visualize them**, and **automate checks** on them. This is what enables powerful features like **impact analysis**: if a requirement changes, you can instantly see what parts of the system are affected.
These relationships also support **reuse**. Instead of rebuilding structures from scratch, you can define patterns of interaction once and reuse them across systems. In large, evolving projects, this relational backbone is what allows a model to stay **organized, traceable, and resilient to change**.



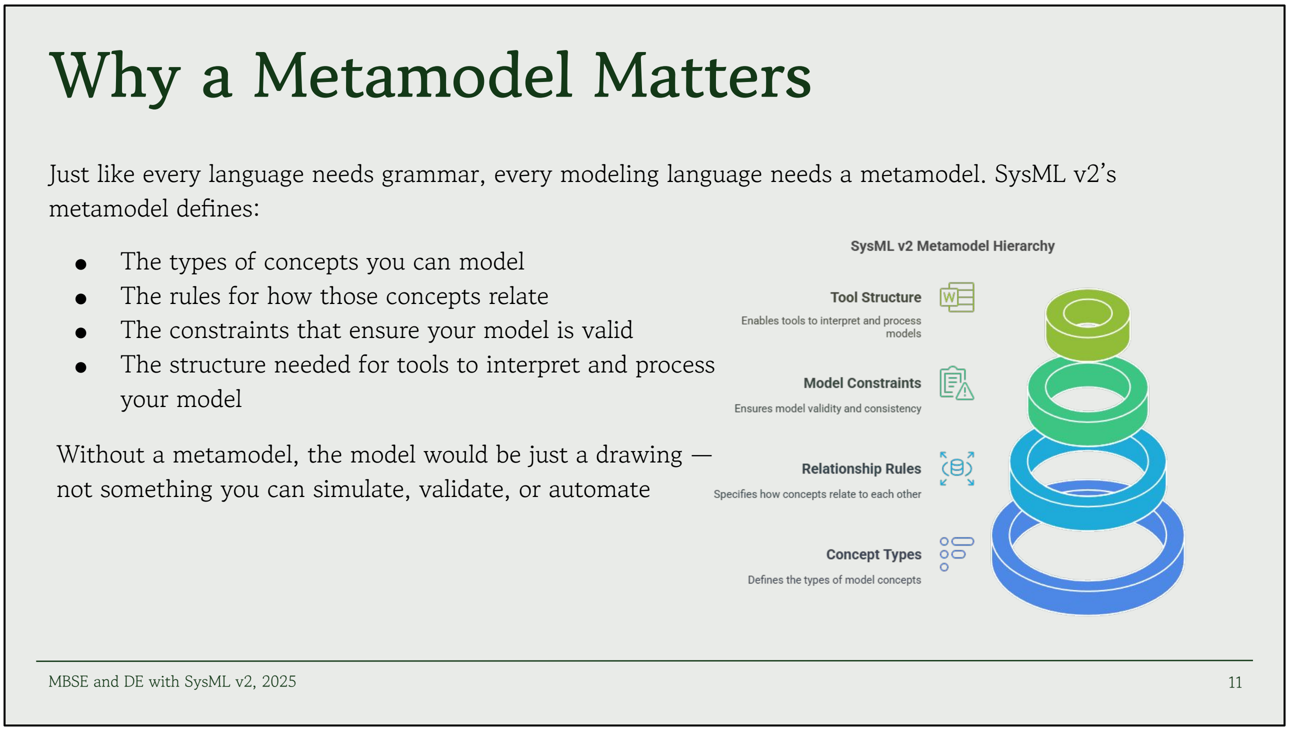
Modeling Domains in SysML v2

In systems engineering, we look at the system from different perspectives — SysML v2 captures these through distinct but connected modeling domains. Each domain offers a specific lens on the system:

- **Structure** — how the system is physically and logically organized
- **Behavior** — how it functions and reacts over time
- **Requirements** — what it must achieve or comply with
- **Constraints / Parametrics** — mathematical and logical limits
- **Verification** — how we know the system meets its purpose

These domains do not exist in isolation — they are **interconnected** and provide a **multi-faceted understanding** of a complex system.

Modeling Domains and Metamodel Foundation: Understanding Modeling Domains in SysML v2
Complex systems can't be modeled from a single point of view. That’s why SysML v2 organizes modeling into **domains** — each one representing a focused perspective of the system. You’ve already seen a few of these: **structure**, **behavior**, **requirements**, and even **parametrics**.
Each domain plays a unique role. Structure tells us **how the system is composed**. Behavior shows **how it functions over time**. Requirements ensure the system **meets stakeholder intent**. And there are more — including analysis, verification, and interfaces.
This separation allows **domain experts to work in parallel**. A control engineer can focus on behavior, while a system architect works on structure — without stepping on each other’s toes.
But the true power comes when we **integrate** these domains. That’s where a model becomes more than a set of views — it becomes a **coherent, cross-disciplinary system representation**.
SysML v2 makes this integration **explicit and formalized**, ensuring everything connects correctly and meaningfully — so the model stays valid, scalable, and adaptable across the lifecycle.

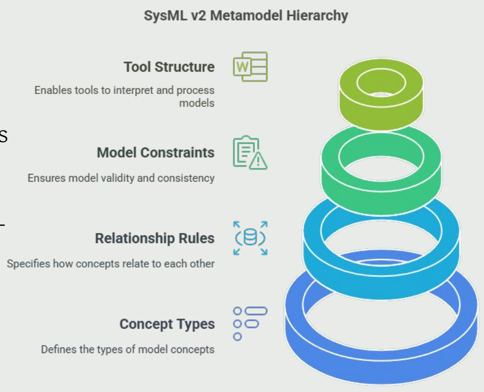


Why a Metamodel Matters

Just like every language needs grammar, every modeling language needs a metamodel. SysML v2's metamodel defines:

- The types of concepts you can model
- The rules for how those concepts relate
- The constraints that ensure your model is valid
- The structure needed for tools to interpret and process your model

Without a metamodel, the model would be just a drawing — not something you can simulate, validate, or automate


SysML v2 Metamodel Hierarchy
Tool Structure
Enables tools to interpret and process models
Model Constraints
Ensures model validity and consistency
Relationship Rules
Specifies how concepts relate to each other
Concept Types
Defines the types of model concepts

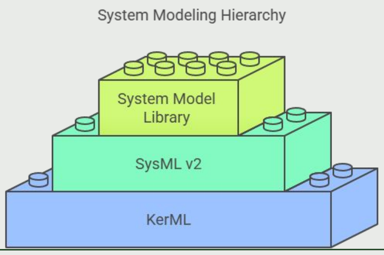
Modeling Domains and Metamodel Foundation: Why a Metamodel Matters
The **metamodel** is the part of SysML v2 you don't usually see — but it's absolutely essential. Think of it as the **grammar** of the modeling language. It defines what kinds of elements can exist — like parts, requirements, or actions — and how they're **allowed to connect**.
When you model something, like linking a behavior to a requirement or defining how components interact, it's the metamodel that ensures that those connections are **semantically correct** and **structurally valid**.
Why does that matter? Because without a metamodel, the model is just visual — a diagram with meaning that only humans understand. But with a metamodel in place, tools can **analyze** your model, **transform it**, check it for **completeness or errors**, and even **generate downstream artifacts** like test cases or code stubs.
This is one of the key reasons SysML v2 is such a leap forward from v1. In v1, the rules were implied — sometimes vague, sometimes inconsistent. SysML v2 formalizes these rules in the metamodel, making your model not just more robust, but also **machine-understandable**.

A Layered Foundation

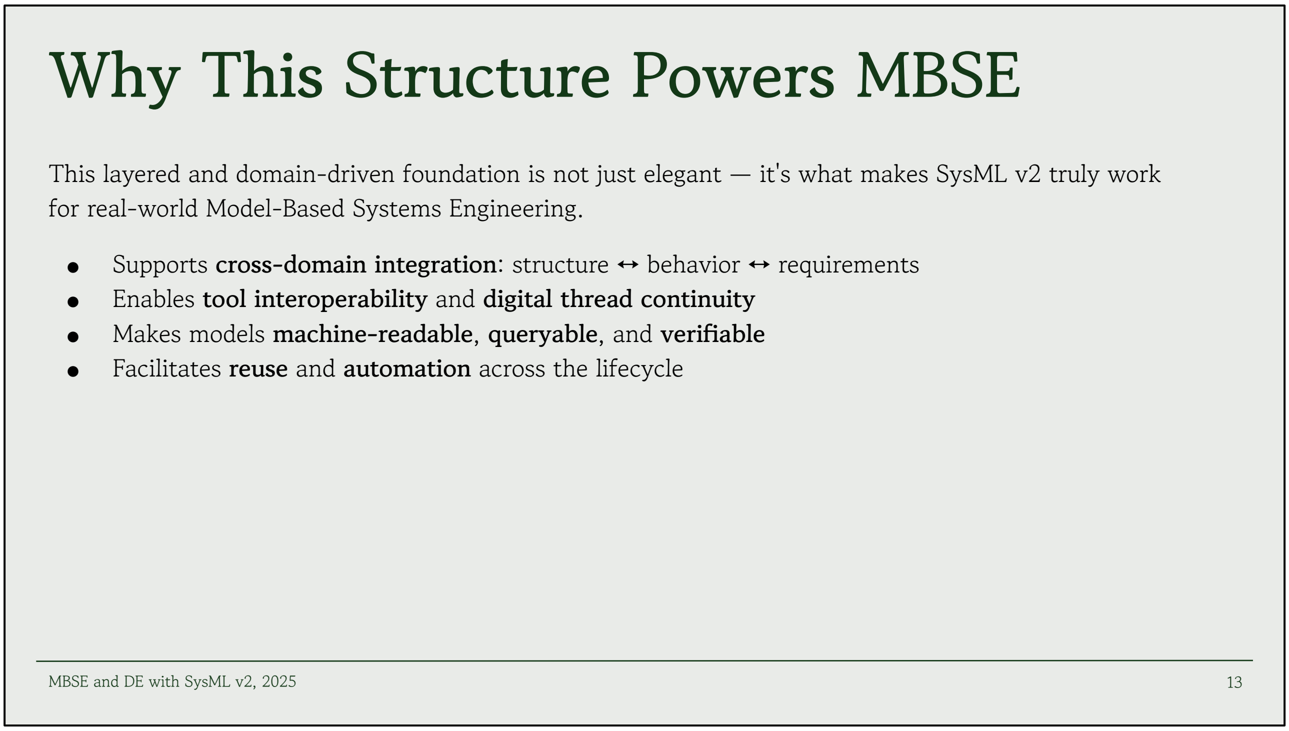
SysML v2 stands on top of a **layered architecture**:

- **KerML**: the abstract modeling language — provides concepts like types, elements, and relationships
- **SysML v2**: specializes KerML for systems engineering
- **System Model Library**: a curated set of reusable engineering model elements like SI units, physical flows, and behavioral patterns

Together, these layers provide a **modular and extensible foundation** for modeling across engineering disciplines.



Modeling Domains and Metamodel Foundation: A Layered Foundation — KerML, SysML v2, and System Model Library
SysML v2 doesn’t exist in a vacuum — it’s built on a **layered foundation** that gives the language both depth and flexibility.
At the bottom layer, we have **KerML** — the Kernel Modeling Language. This is like the **DNA of modeling**. It defines abstract concepts like *types*, *relationships*, *elements*, and *hierarchies* — concepts that are common to all modeling domains. KerML gives SysML v2 a consistent modeling backbone.
SysML v2 then **extends KerML** by adding system-specific capabilities — such as behavior modeling, requirements, and allocation. This layer is what makes SysML v2 suitable for **systems engineering use cases**, including modeling physical architecture, mission flows, and verification logic.
And on top of that, we have the **System Model Library** — a curated set of **reusable modeling elements** like SI units, standard parts, time constructs, flow definitions, and more. These are ready-to-use building blocks that save you time and promote consistency across teams.
This layered architecture enables **scalability**, **semantic integrity**, and **reuse**. It’s also what allows SysML v2 tools to stay extensible and upgradable as modeling practices evolve.

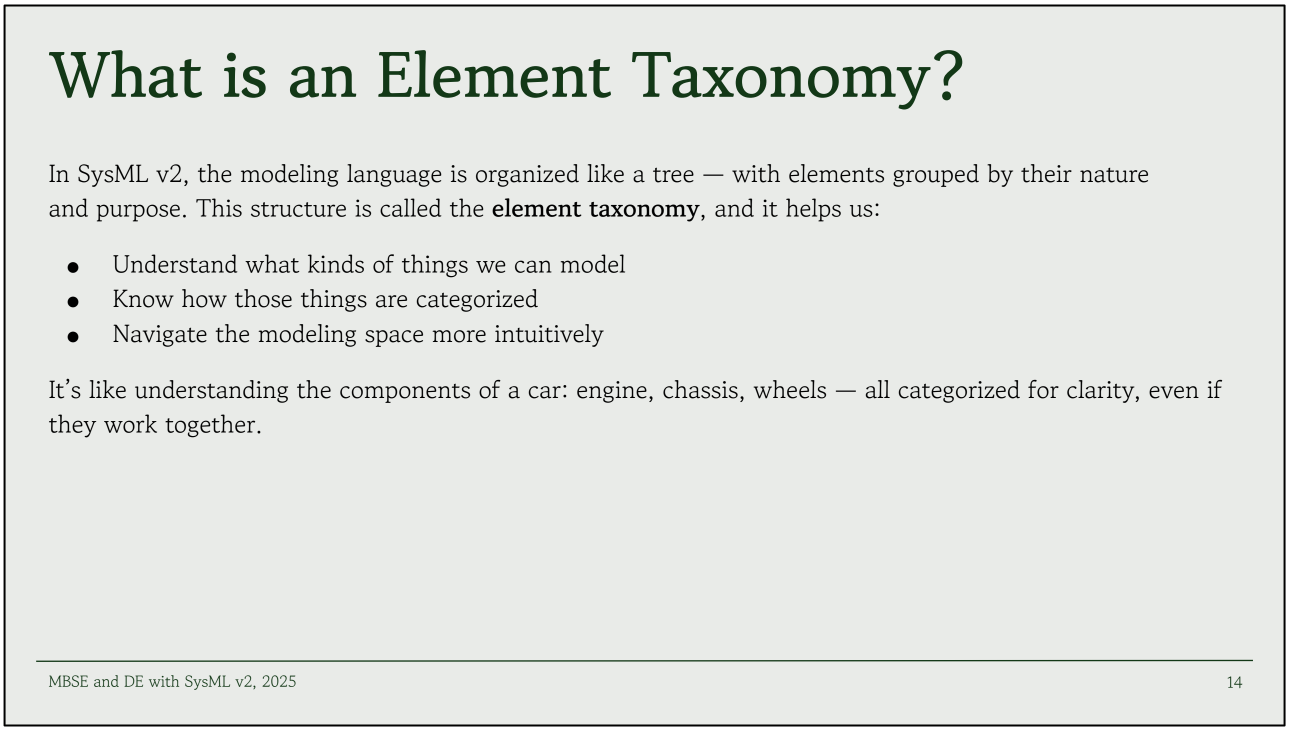


Why This Structure Powers MBSE

This layered and domain-driven foundation is not just elegant — it's what makes SysML v2 truly work for real-world Model-Based Systems Engineering.

- Supports **cross-domain integration**: structure ↔ behavior ↔ requirements
- Enables **tool interoperability** and **digital thread continuity**
- Makes models **machine-readable**, **queryable**, and **verifiable**
- Facilitates **reuse** and **automation** across the lifecycle

Modeling Domains and Metamodel Foundation: Why This Structure Powers MBSE
Model-Based Systems Engineering is not just about creating diagrams — it's about building **data-rich digital representations** of systems that live and evolve across the entire lifecycle. And this is exactly what SysML v2 is built to support.
Thanks to its structured foundation — the combination of **domains**, a formal **metamodel**, and **reusable libraries** — SysML v2 enables models to become **central decision-making assets**. These models are no longer just documentation — they become the system's **authoritative source of truth**.
Because the language is **semantically precise**, tools can perform powerful operations **directly on the model**: from **simulation and trade-space analysis**, to **verification planning**, to even **certification mapping**. You can run queries, trace dependencies, and evaluate system performance — all from a single, integrated model.
This is the essence of MBSE in the digital age: building models that don't just describe the system, but actively **drive engineering, validation, and decision-making**.



What is an Element Taxonomy?

In SysML v2, the modeling language is organized like a tree — with elements grouped by their nature and purpose. This structure is called the **element taxonomy**, and it helps us:

- Understand what kinds of things we can model
- Know how those things are categorized
- Navigate the modeling space more intuitively

It's like understanding the components of a car: engine, chassis, wheels — all categorized for clarity, even if they work together.

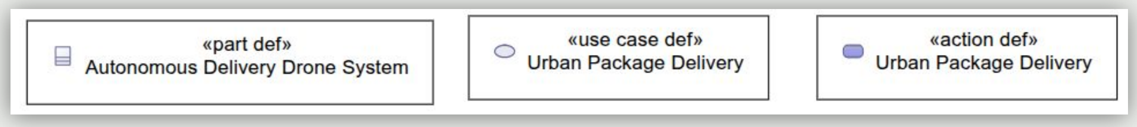
SysML v2 Element Taxonomy Overview: What Is an Element Taxonomy?
Think of the **element taxonomy** as your **map of the SysML v2 modeling world**. Just like any language has a vocabulary organized into categories — nouns, verbs, adjectives — SysML v2 has its own set of **element types**, and this taxonomy tells you how they're grouped and related.
Every model you create — whether you're modeling structure, behavior, requirements, or constraints — is built from these elements. The taxonomy helps you understand **what types of modeling elements exist**, and more importantly, **which ones are appropriate for a given modeling task**.
This structure isn't just academic. It supports **model checking**, **consistency enforcement**, and **reuse** across systems and teams. It also helps tools know how to interpret and validate the model.
And don't worry — you're not expected to memorize this taxonomy today. Right now, just focus on getting a sense of **how it organizes your modeling options** and why that's useful when building and navigating real system models.

Top-Level Categories of Elements

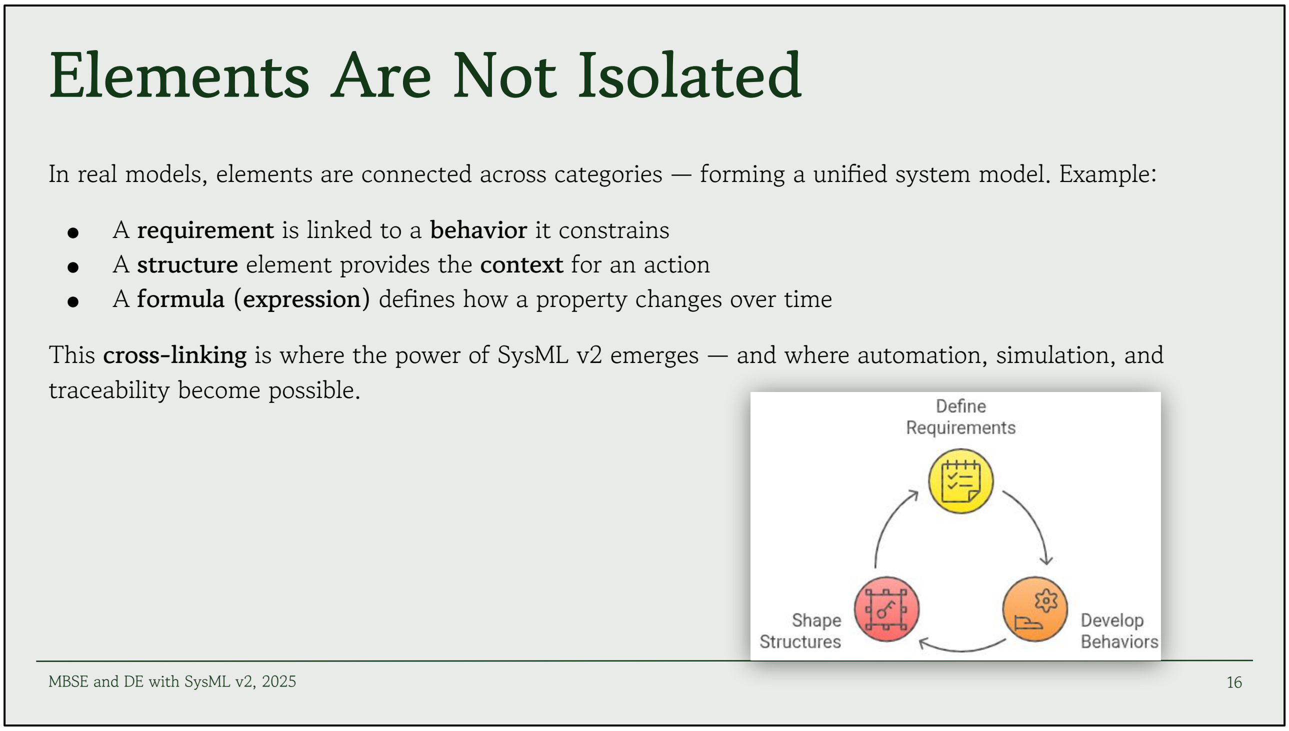
SysML v2 elements fall into several major categories:

- **Structure Elements** — Represent physical or logical system parts
- **Behavior Elements** — Define actions, interactions, and state changes
- **Requirement Elements** — Capture goals, constraints, and verifications
- **Expression Elements** — Model values, conditions, and formulas
- **Connections and Relationships** — Tie everything together semantically

These categories **reflect the modeling domains** we discussed earlier — but here, they're implemented as reusable modeling types.



SysML v2 Element Taxonomy Overview: Top-Level Categories of Elements
SysML v2 organizes its modeling elements into **clear, top-level categories** — and understanding these categories is key to choosing the right kind of element for the job.
Let's say you're modeling a drone. When you define the drone's components — like the motor, propeller, and controller — you're using **structure elements**. When you describe how it takes off, adjusts flight path, or responds to wind — you're working with **behavior elements**. If you capture things like "must maintain altitude within ±2 meters," that's part of the **requirement domain**.
Each of these categories represents a **different modeling perspective**, and the taxonomy helps ensure that those perspectives are formally structured and well-connected.
By understanding these categories, you start to develop **clarity in your modeling approach**. You'll know what kind of element to use, what relationships are valid, and how to keep your model **clean, scalable, and verifiable**.

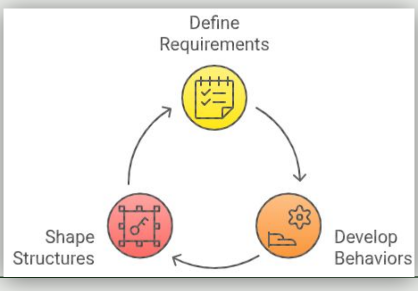


Elements Are Not Isolated

In real models, elements are connected across categories — forming a unified system model. Example:

- A **requirement** is linked to a **behavior** it constrains
- A **structure** element provides the **context** for an action
- A **formula (expression)** defines how a property changes over time

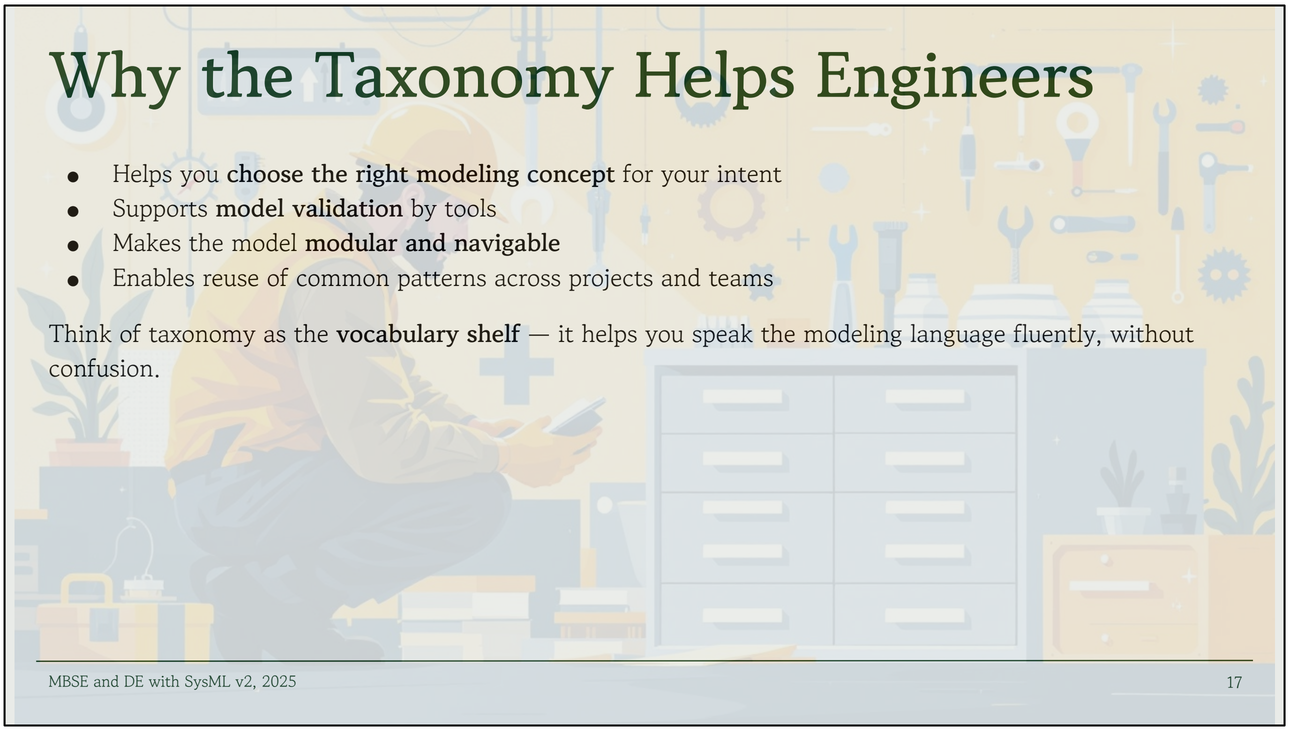
This **cross-linking** is where the power of SysML v2 emerges — and where automation, simulation, and traceability become possible.



SysML v2 Element Taxonomy Overview: Elements Are Not Isolated — They Work Together
One of the most powerful aspects of SysML v2 is that **everything lives in one unified model**. Structure, behavior, requirements, constraints — they're not separate diagrams in different files anymore. They're all modeled as **elements**, and more importantly, they can be **explicitly connected**.
This means you can trace a complete line of reasoning, all within the model. For example:

- A **requirement** says the system must stabilize within 3 seconds
- That requirement is satisfied by a **behavior** — a stabilization control loop
- That behavior is implemented by a **component** — maybe a sensor and actuator
- And that component depends on a **parameter** like system mass or delay

These relationships are **not just drawn as arrows** — they're **formally captured and queryable**. That's what makes SysML v2 such a leap forward for systems engineering. It supports **traceability, impact analysis, and real model intelligence**, all from one consistent source of truth.



Why the Taxonomy Helps Engineers

- Helps you **choose the right modeling concept** for your intent
- Supports **model validation** by tools
- Makes the model **modular and navigable**
- Enables reuse of common patterns across projects and teams

Think of taxonomy as the **vocabulary shelf** — it helps you speak the modeling language fluently, without confusion.

SysML v2 Element Taxonomy Overview: Why the Taxonomy Helps Engineers
Once you get familiar with the SysML v2 taxonomy, modeling becomes a lot more **intentional** — and much faster. You no longer hesitate about how to represent something. Instead, you can say confidently: "I need to define a constraint — I'll use a **requirement element** and connect it to the relevant **behavior**."
This clarity helps you **think like a modeler**, not just a diagrammer. You start making better decisions because you're choosing the **right type of element** for each concept, and that makes your model more **consistent**, **traceable**, and **tool-ready**.
In real projects — especially large or collaborative ones — this structure is what makes SysML v2 **scalable**. Everyone works from the same mental framework. Everyone knows the difference between a definition and a usage, or when to use a behavior versus a state.
So the taxonomy isn't just academic — it's a **practical guide** that helps engineers speak a common modeling language and build models that are not only correct, but also usable, shareable, and maintainable.

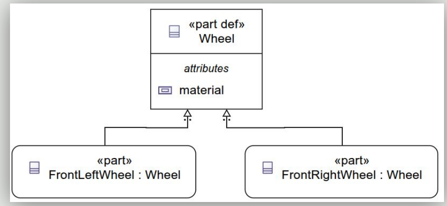
Definition and Usage

SysML v2 separates **“what something is”** from **“how it’s used”** — this principle enables reuse, clarity, and scalability.

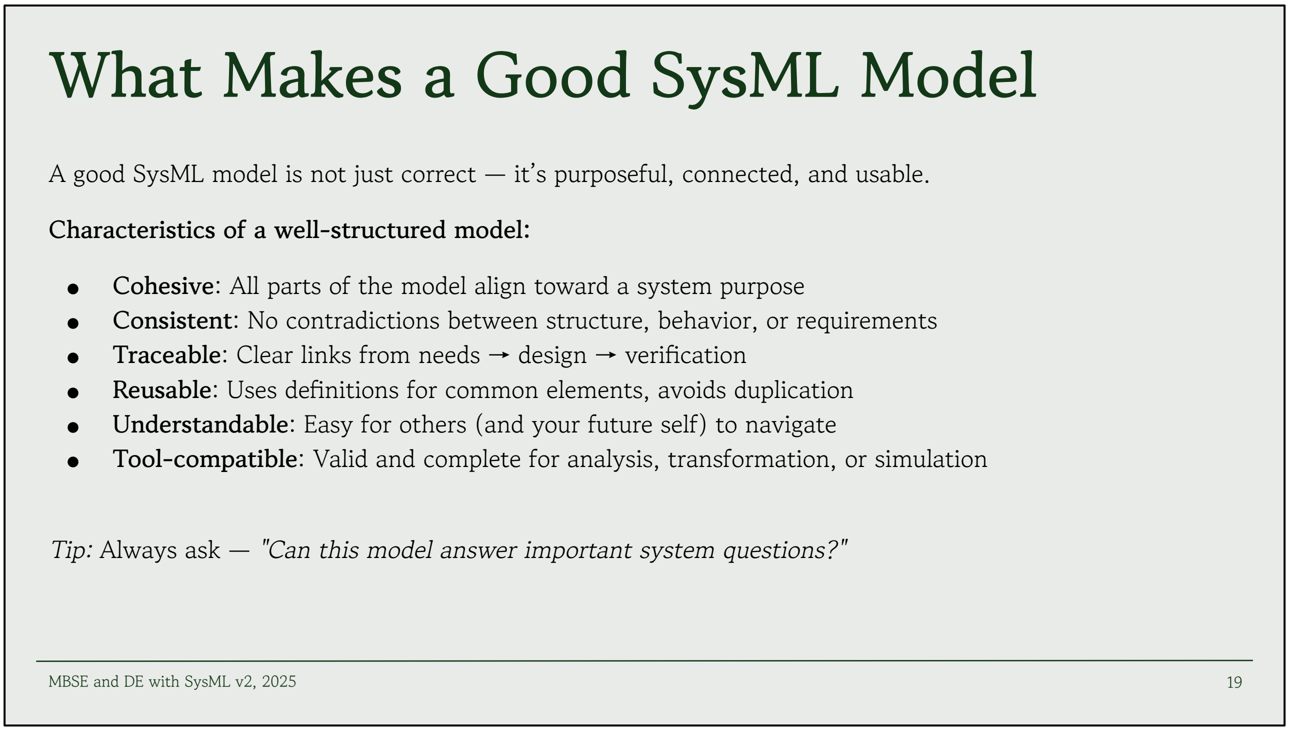
- A **definition** describes a general concept, like a *type of part*, *behavior*, or *requirement*
- A **usage** represents a specific application of that definition in a given context
- Every usage must be linked to a definition — either explicitly or implicitly
- This applies across the model: parts, actions, attributes, states, requirements, and more

Example: A *“Wheel”* can be defined once. Then in different vehicle models, you can reuse it as:

- FrontLeftWheel (usage of Wheel)
- RearRightWheel (another usage of Wheel)



SysML v2 Element Taxonomy Overview: Definition and Usage — The Backbone of Reuse
This is one of the most powerful and elegant principles in SysML v2 — the **separation between definitions and usages**. It’s a modeling pattern that supports **modularity, reuse, and consistency** across large, complex systems.
Here’s the idea: a **definition** is a generic concept — like a part definition for a “Wheel.” It captures what a wheel is in general. A **usage** is how that definition is applied in a specific context — like *FrontLeftWheel* or *RearRightWheel* in a particular vehicle model.
The advantage? You can change the definition once — say, to update its properties or constraints — and **all usages inherit that change automatically**. This means your model stays consistent, even as it grows.
And it’s not just for physical parts. The same concept applies to **behaviors**, **requirements**, **interfaces**, even **analysis artifacts**. You define something once, and then **use it many times** — in different packages, layers, or projects.
This pattern makes your model **scalable, traceable, and clean**. You avoid duplication. You gain control. And you build a model that behaves more like a real engineering asset — reusable, evolvable, and reliable over time.



What Makes a Good SysML Model

A good SysML model is not just correct — it's purposeful, connected, and usable.

Characteristics of a well-structured model:

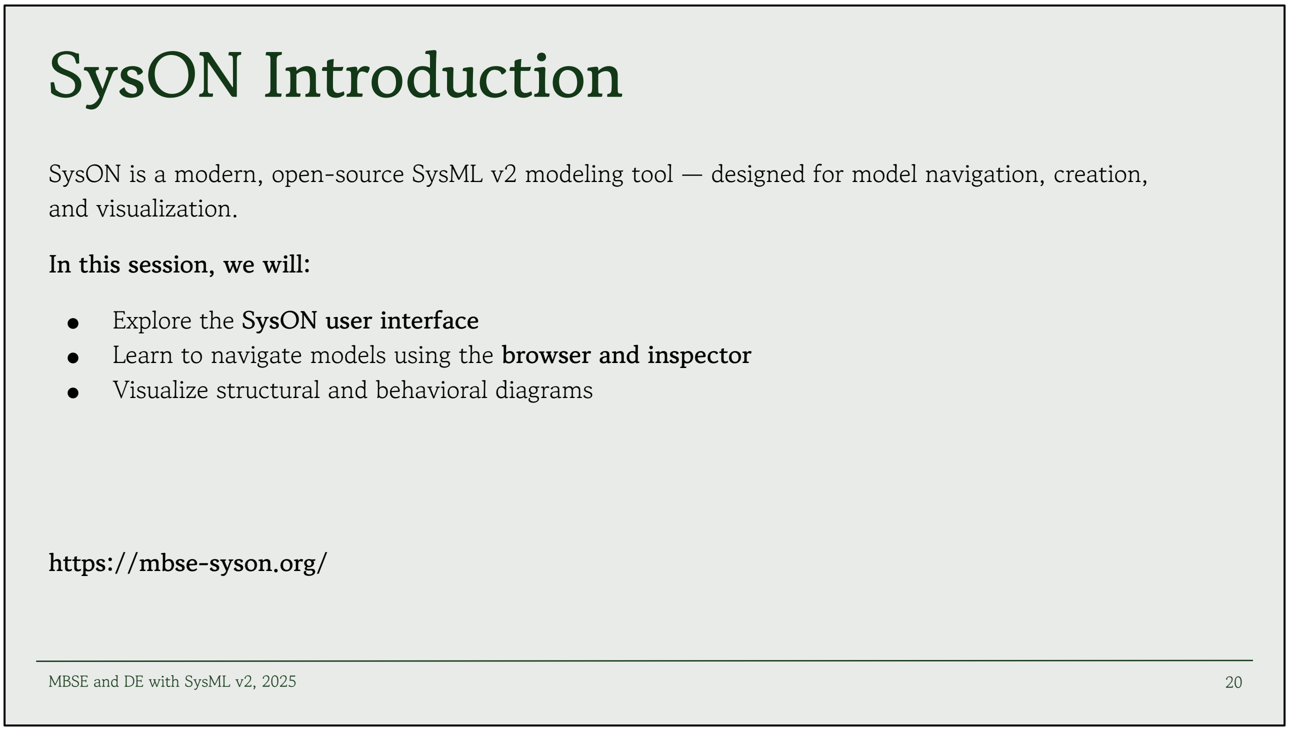
- **Cohesive**: All parts of the model align toward a system purpose
- **Consistent**: No contradictions between structure, behavior, or requirements
- **Traceable**: Clear links from needs → design → verification
- **Reusable**: Uses definitions for common elements, avoids duplication
- **Understandable**: Easy for others (and your future self) to navigate
- **Tool-compatible**: Valid and complete for analysis, transformation, or simulation

Tip: Always ask — *"Can this model answer important system questions?"*

What Makes a Good SysML Model?
SysML is more than just a modeling language — it's a way to **structure engineering thought**.
So how do we know if we're building a *good* model?
A good SysML model doesn't just look clean — it helps you **answer real engineering questions**.
For example:

- *Does this part satisfy its requirement?*
- *What happens if I change this interface?*
- *Can I reuse this behavior pattern in a different variant?*

The model should tell a **story** — a story that begins with stakeholder needs and flows all the way down to implementation logic, test cases, and operational scenarios. And that story should be **clear**, **consistent**, and **traceable**.
Reusability is another hallmark of a strong model. If you find yourself duplicating elements or rewriting behaviors, that's a sign something could be refactored into a reusable definition. Good models avoid redundancy and embrace structure — they're easier to maintain, easier to scale, and easier for teams to collaborate on.
And finally, remember this: the best models are not just technically correct — they are **useful**. Useful to your future self. Useful to your teammates. Useful to your tools. If your model helps others understand, build, test, and decide — then you're doing it right.



SysON Introduction

SysON is a modern, open-source SysML v2 modeling tool — designed for model navigation, creation, and visualization.

In this session, we will:

- Explore the **SysON user interface**
- Learn to navigate models using the **browser and inspector**
- Visualize structural and behavioral diagrams

https://mbse-syson.org/

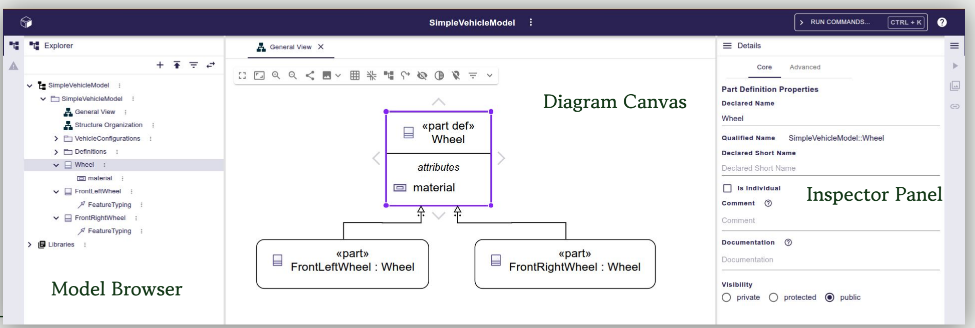
Tooling Environment Setup and Basic Navigation (SysON): Welcome to SysON — Your Modeling Playground
Welcome to **SysON** — your first hands-on gateway into SysML v2 modeling.
SysON is a **lightweight, open-source tool** designed specifically to support the SysML v2 standard. You've already gone through the installation in Week 2, so today isn't about setting it up — today is about **getting comfortable with it**.
We're going to walk through the interface — the **Model Browser**, **Inspector**, and **Representation Views**. You'll see how to create elements, inspect relationships, and begin building models from a system-level perspective.
Think of SysON as your **modeling playground**. It's where you'll apply everything you've learned conceptually — structure, behavior, requirements — and bring it to life. By the end of today, you won't just know what SysML v2 *is* — you'll have used it.
So let's jump in and start exploring your modeling environment.

Workspace Orientation

SysON’s UI is structured around three core areas: the Browser, the Inspector, and Diagrams.

Key Components:

- **Model Browser**: Explore your system model tree
- **Inspector Panel**: See details of selected elements (type, relations, notes)
- **Diagram Canvas**: View and navigate model structure graphically



Part Definition Properties
Declared Name: Wheel
Qualified Name: SimpleVehicleModel::Wheel
Declared Short Name: Declared Short Name
Is Individual
Comment: Comment
Documentation: Documentation
Visibility: private protected [✓ public]

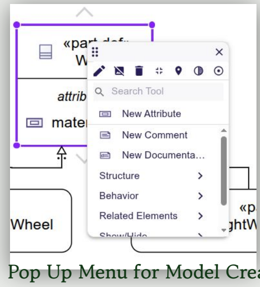
Tooling Environment Setup and Basic Navigation (SysON): Workspace Orientation — Panels and Views
Let’s take a moment to get oriented inside the SysON workspace.
When you first open a model, you’ll see the **Model Browser** on the left. This is your system’s **hierarchical structure** — think of it like a file explorer for your model. You can expand packages, dive into parts, requirements, and behaviors, and see how your system is organized.
When you select any element in the browser, its details appear in the **Inspector Panel** on the right. This is where you can view and edit the element’s properties — its name, type, relationships, and more. You don’t need to write any syntax; SysON uses structured forms to guide your modeling.
At the center of the screen is the **Diagram Canvas** — this is where you’ll create and view **visual representations** of your model. You’ll be able to place elements, connect them, and explore relationships like containment or interface links.
There’s no textual coding here — modeling in SysON is driven by **structured interaction**. This makes it a great environment for learning the language while staying focused on the model’s intent and logic.

Creating and Inspecting Elements

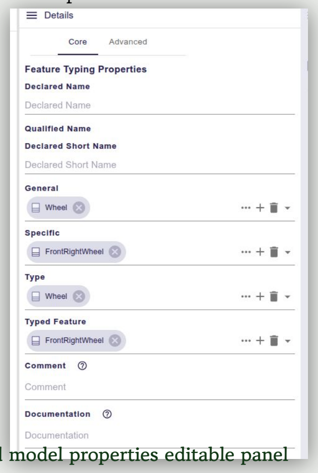
SysON lets you create and explore model elements using guided forms and relationships.

When you select a model element:

- You can **inspect its type and hierarchy**
- You'll see connected elements like **definitions**, **usages**, and **containments**
- You can **add related elements** via GUI tools or command palette



Pop Up Menu for Model Creation



Feature Typing Properties

Declared Name: Declared Name

Qualified Name

Declared Short Name: Declared Short Name

General: Wheel

Specific: FrontRightWheel

Type: Wheel

Typed Feature: FrontRightWheel

Comment: Comment

Documentation: Documentation

Selected model properties editable panel

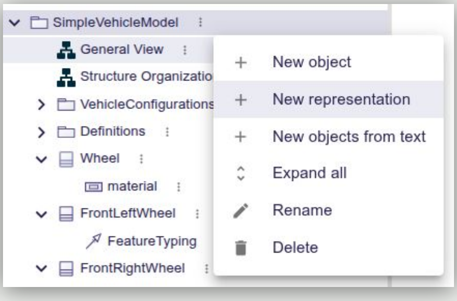
Tooling Environment Setup and Basic Navigation (SysON): Creating and Inspecting Elements
One of the key advantages of SysON is that you don't write any code — you build your model **interactively** using structured menus and guided inputs.

- When you **click on an element** in the Model Browser, the **Inspector Panel** shows you everything about it. You can see:
- What it **contains** — for example, what parts are inside a system
- What it **specializes** — showing if it's refining or extending another definition
- What it **satisfies** — connecting it to the requirements it's meant to fulfill

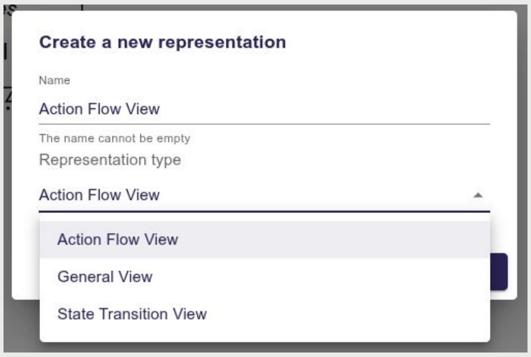
Right from this interface, you can also **add child elements** — like parts within a subsystem, requirements for a component, or behaviors under a controller. You can create those elements and define their relationships without leaving the screen.
This visual, form-based approach is especially helpful when you're learning SysML v2. It reduces errors, maintains consistency, and helps you stay focused on the **modeling intent** — not the syntax.

Creating Representation Views

In SysON, you can create a new diagram (representation) from the Project Explorer. This opens a blank canvas where you can manually build your model's visual structure.



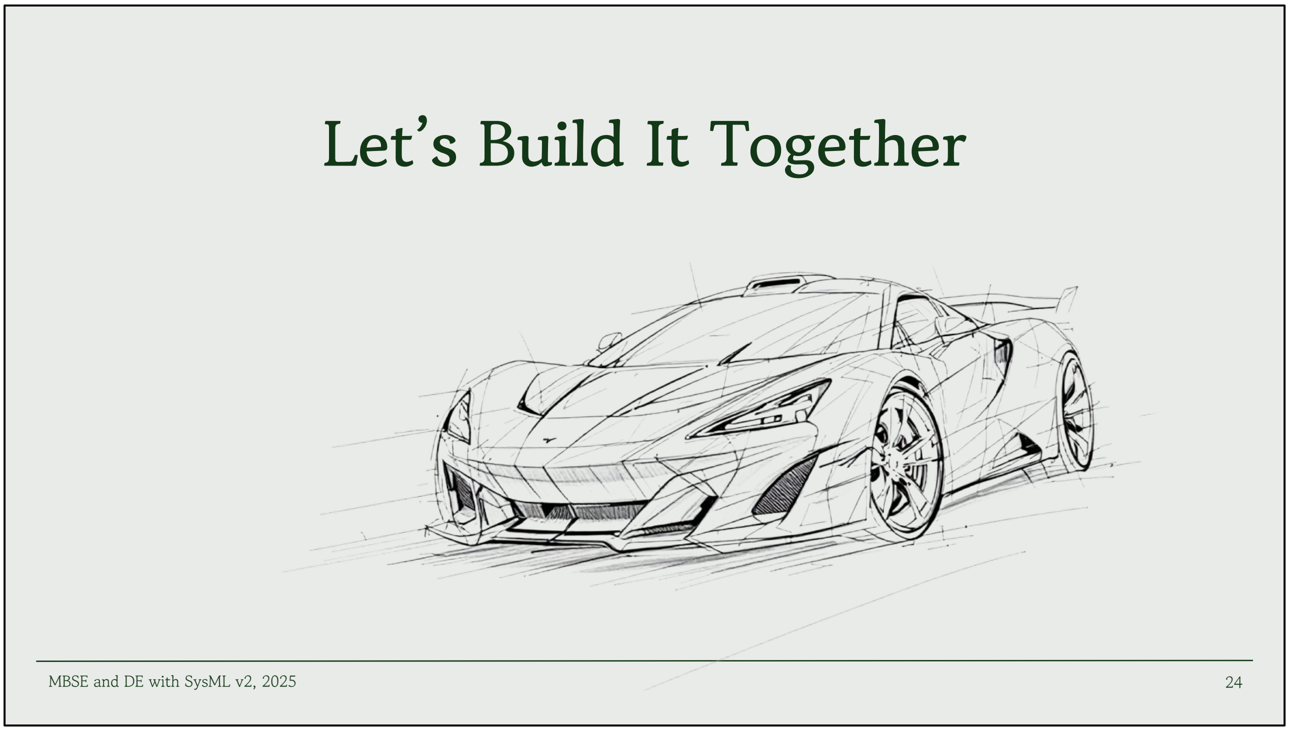
Pop-up menu to create the representation view



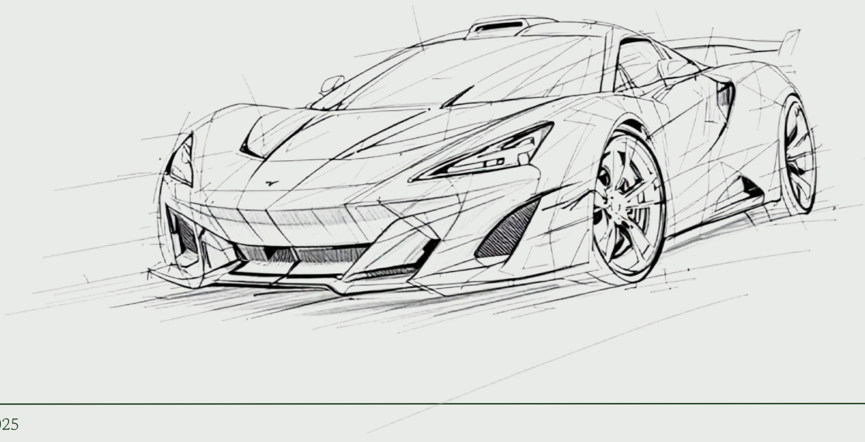
Create representation options dialog

Tooling Environment Setup and Basic Navigation (SysON): Creating Representation Views from the Model Browser
In SysON, creating a new representation gives you a blank canvas to visually construct your model. Unlike some tools that auto-generate diagrams, SysON requires you to manually add elements. This approach provides flexibility, allowing you to tailor the diagram to your specific needs. You can add elements using the palette or by dragging them from the Project Explorer. This hands-on method reinforces your understanding of the model's structure and relationships.
Steps to create a representation:

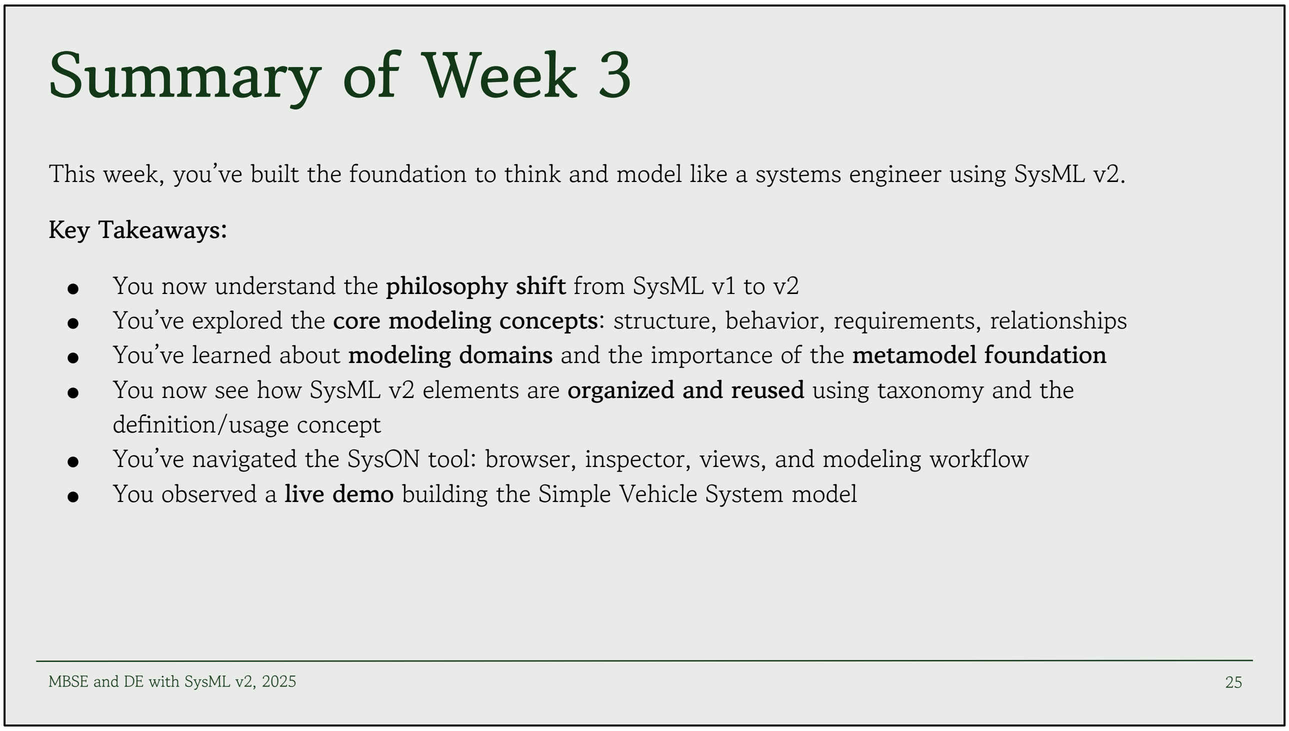
1. In the Project Explorer, click edit menu on a model element (e.g., a package or part).
2. Select "**New representation**".
3. Choose the type of diagram (e.g., General View, Interconnection View).
4. Click "**Create**".
5. A blank diagram opens. Use the palette to add elements or drag them from the Project Explorer.



Let’s Build It Together



Now I’ll walk you through creating and exploring a simple SysML v2 model — live in SysON — using the Simple Vehicle System.



Summary of Week 3

This week, you've built the foundation to think and model like a systems engineer using SysML v2.

Key Takeaways:

- You now understand the **philosophy shift** from SysML v1 to v2
- You've explored the **core modeling concepts**: structure, behavior, requirements, relationships
- You've learned about **modeling domains** and the importance of the **metamodel foundation**
- You now see how SysML v2 elements are **organized and reused** using taxonomy and the definition/usage concept
- You've navigated the SysON tool: browser, inspector, views, and modeling workflow
- You observed a **live demo** building the Simple Vehicle System model

Week 3 Summary — You've Just Entered the World of SysML v2

QUESTION!