Behavioral Modeling Part 1 Use Case, Scenarios and Interactions

Week 06

Why We Model System Behavior

Systems don't just exist — they act.
Modeling system behavior shows *what the system must do* in response to users, environments, and internal conditions.

- Structure tells us *what the system is made of*
- Behavior shows *what it does, when, and in what sequence*
- Behavior modeling focuses on:
  - **Goals** the system must fulfill
  - **Responses** to external and internal triggers
  - **Interactions** with users or other systems

Without modeling behavior, we can't verify mission success, timing, or logic flow

Behavior Modeling Foundations
In systems engineering, we often begin by modeling structure — the parts of a system and how they're organized.
But structure alone doesn't tell us how the system *responds*, *operates*, or *fulfills its mission*.
That's the role of **behavior modeling**.
When we model behavior, we describe how the system acts in time — triggered by actors, sensors, or events.
This includes sequences like: *the drone receives a command, takes off, delivers a package, and returns*.
These are not just random steps — they form the **operational logic** that connects the system's capabilities to real-world outcomes.
Behavior modeling helps us:

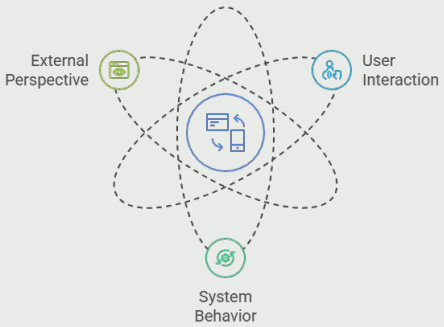
- Define **how the system reacts** to inputs
- Capture **mission flows** and operation steps
- Support **validation** and **simulation** later in the lifecycle

Without it, we're left guessing whether our system will actually do what stakeholders need.

What Are Use Cases — Behavior from the Outside

A Use Case models what the system must do for an external actor.
It captures *goal-driven interactions* that start outside the system and lead to meaningful system behavior.

- Use Cases describe **what** the system does — not **how** it does it
- Each Use Case represents a **goal** initiated by a user, system, or environment
- Helps define the **system boundary**: what's *inside* vs *outside*
- Forms the **starting point** for scenario and behavior modeling



Behavior Modeling Foundations

A **Use Case** is one of the most fundamental concepts in behavior modeling. It describes **how the system is used** — not its internal operations, but its external purpose.
Use Cases focus on the **goal** an external actor wants to achieve through the system. They show *why* the system exists from the user's point of view.
For example, in our drone system, one Use Case might be: **"Deliver Package"**.
The actor is the **Operator** — they initiate the process by issuing a command. The **Drone System** is the subject — it performs the behavior to fulfill the goal.
Use Cases help us answer:

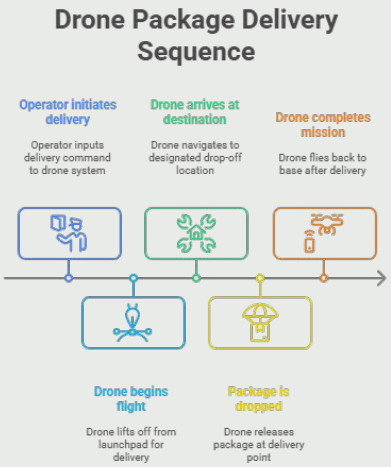
- Who uses the system?
- What do they want to accomplish?
- What must the system do in response?

This keeps us focused on **value**, **interaction**, and **intended outcome** — before we ever dive into how the system performs the task.
This abstraction also helps us communicate clearly with stakeholders, who may not understand technical architecture — but will understand usage and goals.

What Are Scenarios — Sequences of Interaction

Scenarios show how a Use Case unfolds over time.
They describe the *step-by-step interactions* between actors and the system — modeling what happens, when, and in what order.

- A **Use Case** defines *what* the system must do
- A **Scenario** shows *how* that interaction plays out
- Modeled as a **sequence of actions, messages, and decisions**
- Helps clarify:
  - What starts the interaction?
  - What events occur in order?
  - How the system responds and completes the goal



Behavior Modeling Foundations

While **Use Cases** tell us what the system is expected to do, they don't describe how it actually happens.

That's where **Scenarios** come in. A Scenario is a **narrative sequence** — a step-by-step story of how the actor and the system interact to fulfill a goal.

It answers questions like:

- What happens first?
- What happens next?
- What decisions are made along the way?

For example, in the **"Deliver Package"** Use Case, the Scenario might start with:

1. The operator sends a command
2. The drone takes off
3. It navigates to the destination
4. It confirms arrival
5. It releases the package
6. And finally, it returns to base

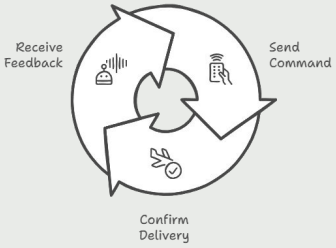
Each of these steps can be modeled with events, actions, and interactions — all traceable and verifiable.

Scenarios help bridge the gap between **goal** and **execution**, and they become the basis for simulations, validations, and timing analysis later in the project.

Interaction Modeling in MBSE

Interaction modeling helps us understand how the system communicates with the outside world.
It focuses on capturing meaningful exchanges between actors and the system — not internal computations or logic.

- In MBSE, we treat the system as a **participant in a conversation**
- Each **interaction** starts with an event, flows through actions, and ends with a result
- This interaction defines what the system must respond to — and how it fulfills stakeholder intent
- Modeling these interactions allows us to trace system behavior back to **who started it, why it matters, and what outcome is expected**



Behavior Modeling Foundations

When we model interactions in MBSE, we're modeling **communication**.
We're not yet concerned with internal algorithms or control loops — we're focused on **what passes between the system and its environment**, and **how it reacts**.
Interaction modeling treats the system as if it's **engaged in a meaningful dialogue** with external actors. These actors might be users, regulators, sensors, or other systems.
This is where:

- **events** are triggered (e.g., an operator sends a command)
- **responses** are generated (e.g., the drone takes off)
- and **causality** unfolds step by step.

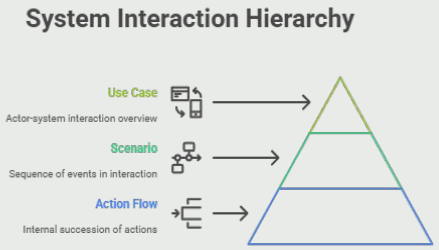
By capturing this flow, we can verify whether the system's behavior actually fulfills the use case — and whether it does so in the right sequence.
Interaction modeling is especially useful early in the lifecycle, when we need to **understand system usage** before committing to implementation details.
This interaction-first mindset keeps the system grounded in **real-world goals**, not just internal design.

Behavior Types in SysML v2

SysML v2 offers multiple ways to describe behavior — each with a different focus.
This week, we focus on Use Cases, Scenarios, and Action Flows to model *what happens, when, and between whom.*

- **Use Case** — Captures external goals and expected outcomes
- **Scenario** — Describes step-by-step interactions between participants
- **Action Flow** — Models internal response logic and control flow

Other behavior models (State Machines, Activities)
will be covered in Week 9



Behavior Modeling Foundations
SysML v2 offers several modeling techniques to represent **system behavior** — each designed for a specific modeling purpose.
This week, we focus on three types:

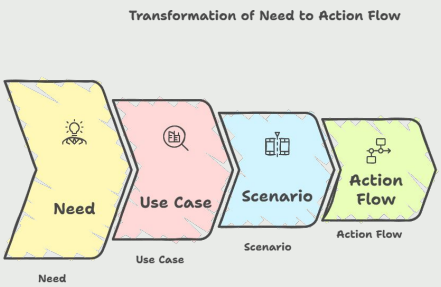
1. **Use Cases**, which capture *what the system must do* from an external actor's perspective.
2. **Scenarios**, which show *how* that Use Case unfolds over time — step by step.
3. **Action Flows**, which describe *how the system responds internally*, using control logic like sequences and decisions.

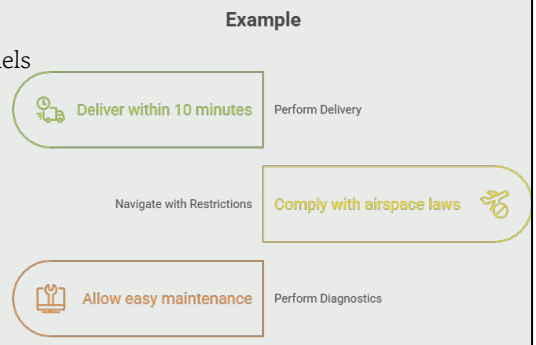
You'll learn how to go from a **stakeholder need** to a **Use Case**, from a Use Case to a **Scenario**, and from a Scenario to a **Behavior Flow**.
Later in the course — particularly in Week 9 — we'll explore **State Machines** and **Activities** to model more complex, reactive, and parallel behaviors.
But for now, we're laying the foundation: behavior from the outside-in, structured and traceable.

How to Derive Use Cases from Stakeholder Needs

Every stakeholder need implies something the system must do.
We transform those needs into Use Cases by asking: *"What interaction fulfills this expectation?"*

- *A* **need** describes *what the stakeholder wants*
- A **Use Case** defines *what the system must do* in response
- Use Cases make behavior traceable to stakeholder intent
- Use this reasoning to build consistent, purpose-driven models





Capturing Use Cases from Stakeholder Needs
When a stakeholder says, *"The drone should deliver packages within 10 minutes,"* that isn't just a general hope. It's a **Need** — and it demands **behavior** from the system.
So we ask: **What does the system need to do to fulfill that?** The answer becomes our **Use Case** — in this case, **"Perform Delivery."**
This is how we derive Use Cases:

- We start with a **Need**
- We identify the **actor** involved (e.g., Customer, Operator)
- We define the **goal** the system must fulfill
- And we capture it as a Use Case

This is a direct, traceable path from concept to model — and it ensures that our behavior is always grounded in stakeholder expectations.
Later, we'll extend each Use Case into a full **Scenario** and finally into **action-based logic**.

Traceable Use Cases: Actor, Subject, Objective, and Require

A complete Use Case connects four key elements: actor, subject, objective, and requirement.
This structure captures the *behavioral boundary* and ensures traceability back to Stakeholder Needs.

- The **actor** initiates the interaction — someone or something external
- The **subject** is the system or part responsible for fulfilling the goal
- The **objective** describes the intended outcome from the user's perspective
- The **require** relationship links the Use Case back to a stakeholder **need** or **requirement**

```
usecase def DeliverPackage {
  actor operator: Operator
  subject drone: DroneSystem
  objective {
    doc /* The drone delivers the package to the destination safely */
    require fastDelivery: FastDelivery
  }
}
```


Capturing Use Cases from Stakeholder Needs
In SysML v2, a complete Use Case contains four elements that make it **clear, usable, and traceable**.

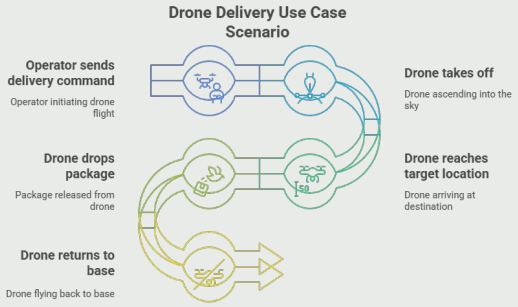
- The **actor** starts the interaction — it's someone or something external to the system.
- The **subject** is the responsible part of the system — the one that performs the behavior.
- The **objective** describes the intended outcome — why the Use Case matters.
- And inside the objective, we use the **require** keyword to reference a requirement that this Use Case fulfills.

For example, let's say we have a Need from Week 4 called **FastDelivery**, modeled as requirement def FastDelivery.When we model the Use Case **DeliverPackage**, we write inside its objective.
This creates a **formal trace** from the stakeholder's need — all the way to a system behavior that aims to satisfy it.
This pattern ensures that everything we model has a **purpose**, and that we can always answer:
"Why does this behavior exist?"

From Use Case to Scenario

Once we define what the system must do (Use Case), the next step is to describe how it unfolds over time. A Scenario turns a goal into a *sequence* of actions and interactions — revealing behavior, responsibility, and flow.

- Use Case = **What** the system must achieve
- Scenario = **How** the behavior progresses step-by-step
- Scenarios break behavior into:
  - Ordered actions
  - Interaction points
  - Optional or branching conditions
- Scenarios are the **blueprint for modeling interaction logic**



Capturing Use Cases from Stakeholder Needs

Once you've defined a Use Case, the next question is — *how does it happen?*

That's the role of the **Scenario**. It takes a single Use Case and expresses it as a **structured series of interactions** over time.

Scenarios are:

- Linear or branching
- Centered on time and causality
- Built around **events**, **messages**, or **actions**

For example, in our Drone System:

1. The operator sends a delivery command
2. The drone takes off
3. It navigates to the delivery location
4. It drops the package
5. Then returns home

This sequence is what we will soon model as an **Action Flow**, using action def, action, and succession.

But before we go there, we must explore the idea of **events and interactions**, and how they form the backbone of behavior in SysML v2 — even if we model them differently in SysON.

What Is an Event in a Scenario?

Events are the building blocks of scenarios.
Each event marks a specific point in time when something happens — triggered by an actor or system element.

- An **event** represents a single moment — something starts, changes, or completes
- Events are **not internal logic** — they are *observable outcomes* in the system's interaction with the world
- They can:
  - Be triggered by actors (e.g., "button pressed")
  - Be internal signals or environmental triggers (e.g., "battery low")
  - Be sent or received as messages in the system

Event: sendCommand — the operator initiates the delivery
Event: packageDropped — drone confirms action is complete

Scenario Modeling Concepts and Limitations
Now that we've seen how a Use Case expands into a Scenario, let's break that scenario into its smallest parts — **events**.
An **event** is a single point in time where something important happens.
It's not a full activity or state — just a moment: "Operator sends command," "Drone takes off," or "Battery warning appears."
Events can:

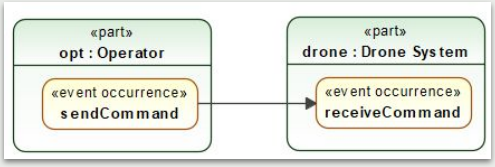
- Come from outside the system (like an operator action)
- Be system responses (like a confirmation signal)
- Or be environmental triggers (like a wind speed alert)

Each of these becomes a **step** in our behavioral flow.
They give us precision — and let us sequence what happens, and when.
And most importantly, **they give us traceability** — each event is part of fulfilling a Use Case, which satisfies a stakeholder need.
In SysML v2, we represent these formally as event occurrence, which we'll explore next — along with how they connect.

What Is an Interaction?

An interaction connects two events across participants.
It represents a *cause-and-effect relationship* — one event leads to another.

- An **interaction** models communication, coordination, or signal exchange
- It links an **event from one participant** to an **event in another**
- Interactions define **system boundaries** — who communicates with whom
- They are essential for capturing:
  - Actor ↔ system communication
  - System ↔ system collaboration
  - Conditional or triggered behaviors



Scenario Modeling Concepts and Limitations
Now that we understand what an **event** is, let's see how events become meaningful — by being **connected**.
That connection is called an **interaction**.
An interaction is what happens **between participants** — one actor or part triggers something in another.
It's not a standalone behavior — it's a **relationship** between two events:

- A source event: something that initiates
- A target event: something that responds

For example, in our Drone System:

- The Operator has an event: sendCommand
- The Drone has a corresponding event: receiveCommand
- The interaction is the message or signal that connects them

This is how we capture **cause and effect**, and how we define **system usage** — through interaction sequences that make behavior observable, testable, and traceable.
In SysML v2, we'll model this using message relationships — but even before syntax, the concept of interaction is core to any scenario.

Time and Causality

Scenarios unfold in time — and the order of events matters.

Modeling *causality* helps us describe not just what happens, but *when* and *why*.

- **Causality** defines the logic of "this happens → then that happens"
- Events are connected not just by interaction, but by **temporal order**
- Sequences allow us to model:
  - Operational flows
  - Safety-critical responses
  - Conditional logic paths
- Precise event order improves:
  - Test planning
  - Simulation
  - Requirements coverage

first sendCommand **then** takeoff **then** dropPackage **then** return

Scenario Modeling Concepts and Limitations

So far, we've introduced events and interactions — but those alone don't give us a full understanding of **how behavior flows**.

To build real scenarios, we need to model **time and causality**.

That means saying not only *what* happens — but *in what order*, and *why*.

For example:

- First, the operator sends a command
- Then, the drone takes off
- Then it navigates to the destination
- Then it drops the package
- Then it returns

This flow isn't just narrative — it's **essential for engineering**:

- If the drone drops the package before reaching the target — we have a problem
- If it takes off without a command — even worse

By modeling **event order explicitly**, we can: Validate timing assumptions, Improve safety and correctness, and Simulate flows and detect edge cases

In SysML v2, we model this sequence using keywords like first, then, and even if for conditional flows — which we'll explore next.

SysML v2 Event Syntax Overview

SysML v2 provides precise syntax to model events, interactions, and causal flow.
While these constructs exist in the language, current tools like SysON may not fully support them yet.

- event occurrence: declares a time-specific event in a scenario
- message: connects events between participants (interaction)
- flow: groups event sequences to describe causality
- first, then: define the order of event execution
- if event occurrence: models conditional branching in scenarios

Note: SysON currently does **not support** sequence or scenario diagrams using these elements.

```
part def Operator;
part def 'Drone System';
item def Command;
part opt : Operator {
    event occurrence sendCommand {
        out item item1;
    }
}
part drone : 'Drone System' {
    event occurrence receiveCommand {
        in item item1;
    }
}
message of  item1 : Command from opt.sendCommand to drone.receiveCommand;
connection connect opt.sendCommand to drone.receiveCommand;
flow from opt.sendCommand.item1 to drone.receiveCommand.item1;
```


Scenario Modeling Concepts and Limitations

Now that you understand events, interactions, and time-based flow — here's how SysML v2 lets us model them in practice.
The key syntax elements are:

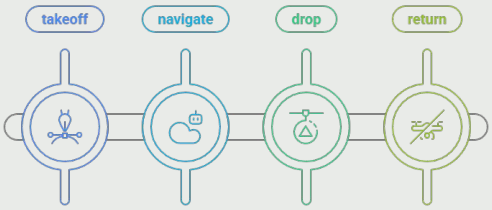
- event occurrence: this is the actual event, tied to a participant (either system or actor)
- message: connects two events across participants to model interaction
- flow or first ... then ...: models causal sequence
- if event occurrence: lets us branch the scenario based on conditions

In theory, you could model full, rich scenarios using just these elements — and the SysML spec supports that completely.
However — **SysON does not yet support modeling these constructs graphically**. You can read and write them in textual .sysml files, but there's no visual sequence or scenario diagram feature yet.
That's why — in this course — we will use **Action Flows** instead to model scenario behavior in practice.
They're supported in SysON, and still let us model **sequences**, **decisions**, and **responses** in a structured way. And that's what we'll explore next.

Why Use Action Flow for Scenario Modeling

In SysON, we model scenarios using Action Flow — not sequence diagrams.
Action Flow captures *what the system does* in a step-by-step manner, using nodes, flows, and decisions.

- Sequence diagrams and event-based flows (event occurrence, message) are not yet supported
- Action Flow provides an **equivalent modeling path**:
  - Captures system response to external triggers
  - Models step-by-step logic
  - Supports branching and parallel flows



- We'll use:
  - action def to define reusable behavior
  - action to use those behaviors in a specific flow
  - succession and decision node to show flow order and logic

Modeling Scenarios with Action Flow in SysON
So far, we've talked about events, interactions, and causal flow — all of which can be modeled using standard SysML v2 syntax.
But here's the reality: Tools like **SysON** do **not yet support** sequence diagrams or event occurrence modeling.
That's why we shift to something **equally expressive** — **Action Flow**.
Action Flow is a way of modeling **system behavior as a network of steps**:

- Each step is called an action
- Actions are connected using succession to indicate what happens next
- We can also add logic, like decision node, to model branches

And most importantly — **SysON supports this natively**, with a visual diagramming tool.
So for the course project, we'll model each Scenario using:

- action def to define the behavior
- action to use it
- succession and decision node to define order and branching

This is not a compromise — this is a **practical, standards-compliant** way to model behavior that's compatible with real-world MBSE workflows.

Action Flow Syntax in SysML v2 (Beginner Form)

We model behavior flow using simple sequences: first … then.
This syntax is clear, readable, and supported in tools like SysON — perfect for scenario modeling today.
Elements Used:

- action def — declares the flow
- action — defines steps inside the flow
- first … then — sequences actions over time

```
action def FlowName {
  action step1: ActionType1;
  action step2: ActionType2;
  first step1 then step2;
}
```


Modeling Scenarios with Action Flow in SysON
To keep modeling simple and focused, we'll use the first … then syntax for Action Flows. This is ideal for defining linear behavior — like a drone taking off, flying, and dropping a package.
Let's walk through the syntax:

- We define the flow using action def
- Each step is declared as an action and given a name
- Then we describe the sequence using first … then

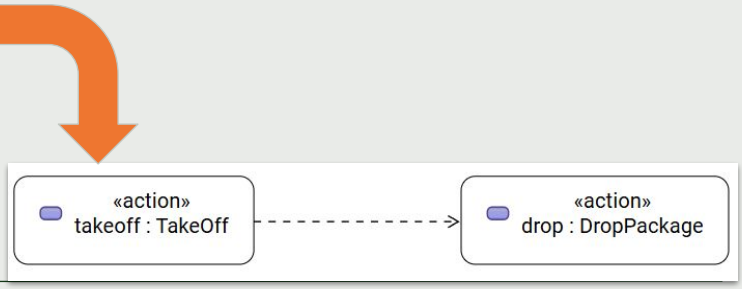
This is easier to read and model than more abstract control structures — and it's directly supported in SysON.
We'll revisit more complex flow syntax like succession flow, parallel, and branching later in **Week 9**. For now, focus on using first … then to describe your scenarios clearly and traceably.

Key Elements of Action Flow

Action Flow uses connected nodes to represent behavior over time.
It defines *what happens*, *in what order*, and *under what conditions* — using simple, modular elements.

- action def — defines reusable behavior (e.g., Takeoff, DropPackage)
- action — usage of the behavior in a specific flow
- succession — defines the order: *this follows that*
- decision node — adds branching logic (if/else style)

```
action def Takeoff;
action def DropPackage;
action def DeliveryFlow {
    action takeoff : Takeoff;
    action dropPackage : DropPackage;
    first takeoff then dropPackage;
}
```



Modeling Scenarios with Action Flow in SysON
Let's break down how **Action Flow** works in SysML v2 — especially in **SysON**, where you'll be modeling this behavior graphically.
The main building blocks are:

- action def: This is where you **define a named behavior** — like Takeoff, DeliverPackage, or EmergencyLand. Think of it like a function in programming.
- action: This is where you **use** that behavior inside another flow.
- succession: This defines the order — like "do this, then do that."
- decision node: This lets you **branch** the flow depending on conditions — for example, checking if battery level is safe.

With just these four elements, you can model:

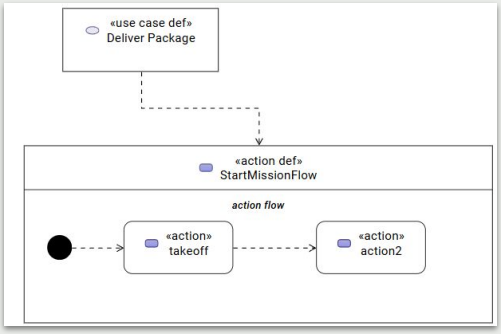
- Linear flows
- Parallel flows
- Conditional logic
- System reactions to external input

You can think of Action Flow as a lightweight but powerful alternative to State Machines or Sequence Diagrams — perfect for describing what your system *does* in response to interaction scenarios.
Next, we'll see a complete example using these elements in the context of a delivery mission.

Example Breakdown – Drone Takeoff Scenario

Let's model a simple scenario: the drone begins a mission after receiving a command.
We'll break it into actions and define the flow using succession.

- **Use Case**: Deliver Package
- **Scenario**: Operator initiates mission → Drone takes off → Drone confirms flight
- We express this as:
  - action def StartMissionFlow
  - Inside it: action takeoff, action confirm
  - Connect them using succession



Modeling Scenarios with Action Flow in SysON
Let's apply what we've learned to a simple but meaningful scenario — the start of a drone mission.
Imagine the operator has just issued a command to begin delivery. The drone should:

1. Receive that command (modeled outside this flow)
2. **Take off**
3. **Confirm altitude stability** before proceeding further

We model this in SysML v2 using an action def called **StartMissionFlow**.
Inside this flow:

- We define two actions: takeoff and confirm
- And we link them using a succession — saying: *first takeoff, then confirm*

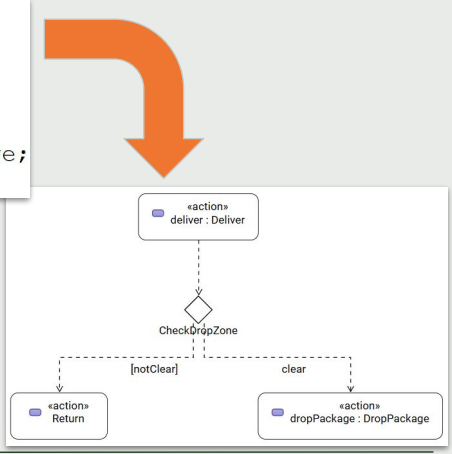
This simple pattern gives us a foundation for modeling every other behavior — not just for flights, but for navigation, delivery, diagnostics, or even emergency landing.
On the next slide, we'll review the syntax elements in detail so you can begin building your own Action Flows in SysON.

Modeling Decision Logic with ‘Decide’

The decide keyword lets us introduce decision logic inside a behavior flow.

It represent the decision node and works naturally with first … then syntax.

```
action deliver : Deliver;
action dropPackage : DropPackage;
action Return;
decide CheckDropZone;
first deliver then CheckDropZone;
succession '[clear]' first CheckDropZone then dropPackage;
succession '[notClear]' first CheckDropZone then Return;
```



Modeling Scenarios with Action Flow in SysON

In SysML v2, when you want to include **decision-making** in your action flow — like choosing between two actions based on a condition — you use the decide keyword.

This is a cleaner, more modern alternative to declaring a decision node explicitly.

Here’s how it works:

- You define an action that performs a check — like CheckDropZone
- Then you use decide to say: based on that check, do one of two things
  - If the condition is [clear], then drop the package
  - If [notClear], then hold position and wait

This approach integrates perfectly with first … then, and keeps your models clean, readable, and executable in tools like SysON.

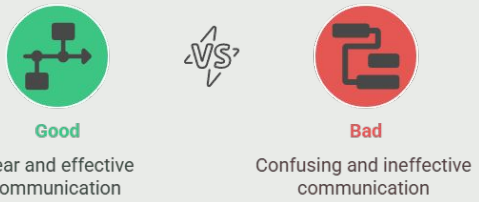
So for any logic that depends on system conditions — battery level, safety status, weather, GPS lock — use decide to capture that logic elegantly.

Graphical Layout Tips for Action Flow

A clear diagram tells a clear story.
Structure your Action Flow diagrams to reflect time, responsibility, and logic — using space, labels, and flow direction.

Best Practices for Visual Clarity:

- Use **left-to-right** or **top-to-bottom** direction for readability
- Group related actions visually — don't scatter them
- Label each action meaningfully (avoid step1/step2)
- Keep arrows **straight** and **non-overlapping** whenever possible
- Use white space — give each node breathing room



Modeling Scenarios with Action Flow in SysON
In SysML — and in systems thinking in general — **how you lay out a diagram matters**.
A well-structured Action Flow isn't just about correct syntax. It's about making the behavior **understandable at a glance**.
Here are some layout tips:

- Use **left-to-right flow** (or top-to-bottom) — it matches how we read and think about time
- Give each action a **meaningful name** — like CheckBattery, DropPackage, or ReturnToBase
- Keep **arrows clean** — avoid loops, intersections, or tangled lines
- Group related actions close together — this shows logic and modularity
- **Leave white space** — it's not wasted; it's clarity

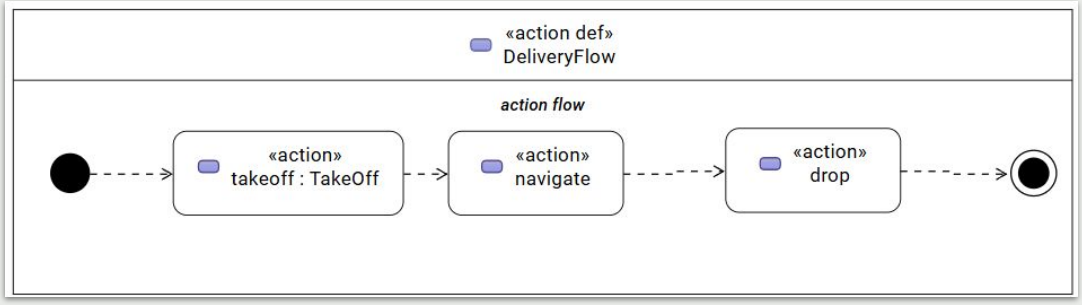
Encourage yourself to model as if someone else will need to read their diagram — because that's often true in real-world MBSE projects.
Clear diagrams = trustworthy models.

Mapping Scenario Steps to Action Nodes

Each scenario step becomes an action node.

Start with a simple narrative, then turn each step into a named action — and sequence them using first … then.

"The operator sends a command. The drone takes off. It navigates to the target. Then it drops the package."



Modeling Scenarios with Action Flow in SysON

This is where **writing meets modeling**.

Let's say you've captured a scenario from your stakeholder:

> "The drone starts the mission, flies to the destination, and drops the package."

How do you turn that into a SysML model?

Step-by-step:

1. Break the narrative into **individual steps** (verbs are your guide!)
2. For each step, define an action def — this lets you reuse it in other flows
3. In your main flow, create action usages of those steps
4. Connect them in order using first … then

This lets you:

- Preserve the **intent** of the original narrative
- Create **traceable, testable behavior**
- Keep your model clean and modular

In your Drone System project, this is exactly how you'll transform Use Case scenarios into **working behavior flows**.

What You Can't Model in SysON (Yet)

SysON is evolving — not every SysML v2 feature is supported yet.
Focus on what's available now: Action Flow. We'll explore advanced behavior modeling in later weeks.

Currently Supported and Recommended:

- action def and action
- first ... then (succession) behavior sequences
- decision node (basic conditional logic)
- Modeling with visual Action Flow diagrams

Not Yet Supported in SysON:

- Sequence diagrams and event occurrence
- message between participants
- flow, occurrence def, and causal interaction chains
- Conditional if event occurrence branches
- Full state-based or parallel behavior flows

Modeling Scenarios with Action Flow in SysON
Before we move into full modeling practice, it's important to **set clear expectations** about what you can and can't do in SysON *today*.
SysML v2 is a powerful language — but like all tools, **SysON implements it gradually**.
Here's what's **not yet supported**:

- You can't create sequence diagrams or model event occurrence and message
- You won't find support for full scenario structures like occurrence def or flow
- Advanced constructs like parallel actions or if event occurrence conditions aren't there yet

But — and this is key — you can still model **everything important** about your system's behavior using **Action Flows**:

- Define behavior using action def
- Chain steps using first ... then
- Branch logic with decision node if needed

What you're learning is not a workaround — it's **real MBSE** using the subset of SysML v2 that is ready and usable now. And as SysON evolves, your models will be future-proof and extendable.

Example – Start Vehicle Scenario

Let's walk through a full Action Flow model for a simple system interaction.
This example shows a driver starting a vehicle and the system responding in steps.

"The driver presses the start button. The vehicle checks the battery. If okay, it starts the engine and initializes the display. If not, it sends an alert."

Example and Practice
Here's a complete example that brings everything together — using a **Start Vehicle** scenario.
We begin with a simple narrative:

> The driver presses a button. The system checks the battery. If it's okay, it starts the engine and initializes the display. If not, it sends an alert.

We model this using:

- action def: to define the overall flow
- action: to include each behavior step
- decision node: to control the logic
- first ... then: to express the sequence

The key benefit of modeling this way is:

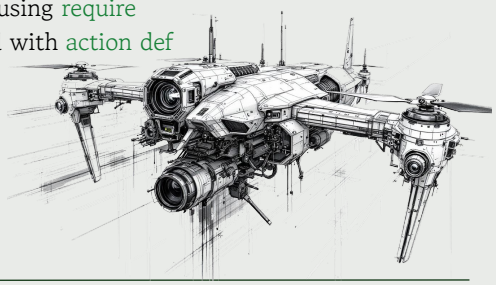
- It's **simple to read**
- Easy to validate
- Works perfectly in **SysON**

This pattern — a clean, readable behavior flow — is what your drone system will need as well. Whether it's mission execution, delivery, or emergency handling, you'll apply the same flow structure.

Continue Your Project

Use Case and Scenario Modeling
It's time to expand your project model. This week, you will build on your stakeholder needs and system context by defining Use Cases and modeling scenarios using Action Flow.

- Revisit your Stakeholder Needs from Week 4
- Confirm your System Context from Week 5
- This week:
  - Define at least one use case def with actor, subject, and objective
  - Link it to the appropriate stakeholder requirement def using require
  - Break the Use Case into a sequence of actions modeled with action def
  - Use succession to structure the flow
- Use SysON to implement and validate your model visually



This week, you will continue developing your **Autonomous Drone System** model.
Pick up where you left off:

- Start from the **stakeholder needs** you defined in Week 4
- Build on the **System Context** and boundaries from Week 5

Now you'll move into the **Behavior Layer**:

- Define one or more **Use Cases** that your system must perform
- For each, clearly declare the **actor**, **subject**, and **objective**
- Use the require keyword inside the objective to link it back to your stakeholder Need

Then take the next step:

- Convert that Use Case into a Scenario
- Use **Action Flow** modeling to represent what happens, in what order, and how the system responds

Use **SysON** to model it visually. The goal is not to be perfect — but to build **traceable, executable logic** grounded in stakeholder expectations.

Summary of Week 6

This week you learned how to model system behavior from the outside in.
You now have the tools to turn stakeholder intent into structured, traceable behavior flows.

- Use Cases describe what the system must do for an **external actor**
- Each Use Case should have a **subject**, **actor**, **objective**, and require link to a **stakeholder requirement**
- Scenarios break Use Cases into step-by-step **interactions**
- We model behavior using **Action Flows**:
  - action def, action
  - first ... then
  - decide for conditional logic
- **SysON** supports Action Flow modeling — focus your project work there

This week, you took your first step into modeling system behavior.
You started with Stakeholder Needs, and learned how to transform them into:

- **Use Cases** — structured, purpose-driven system interactions
- **Scenarios** — ordered steps that unfold over time
- **Action Flows** — visual, model-based logic that defines how the system behaves

You now know how to:

- Write a use case def with a clear objective
- Connect that Use Case to a requirement using the require keyword
- Build a step-by-step behavior model using first ... then and decide

This modeling flow is powerful — and it's directly traceable back to what stakeholders want from the system.
In your project this week, focus on implementing one or more **real Use Cases** in your model.
Capture the flow clearly, make it visual, and stay grounded in the needs you already defined.
This sets you up for Weeks 7 and 8 — where we'll go deeper into requirements and interfaces.

QUESTION!