



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

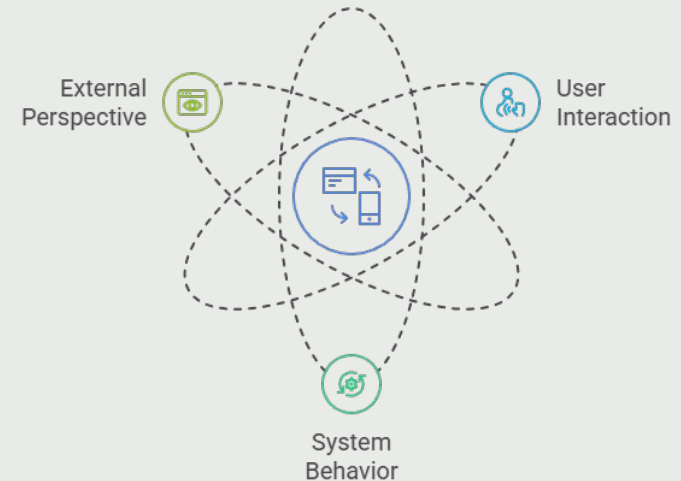
Without modeling behavior, we can't verify mission success, timing, or logic flow

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

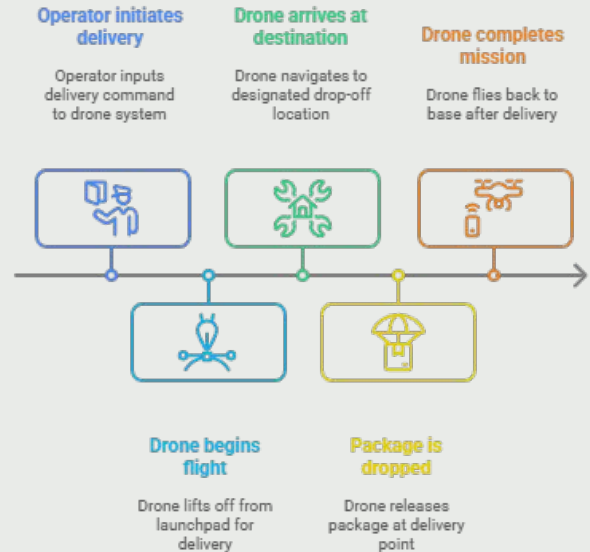


What Are Scenarios — Sequences of Interaction

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

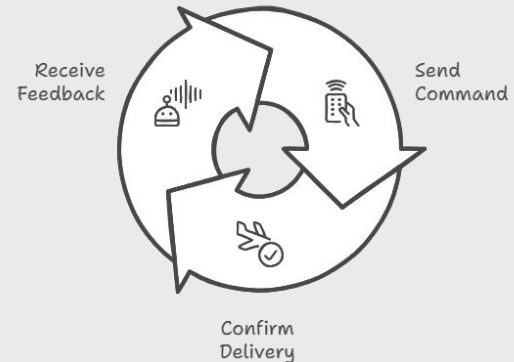
Drone Package Delivery Sequence



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 Types in SysML v2

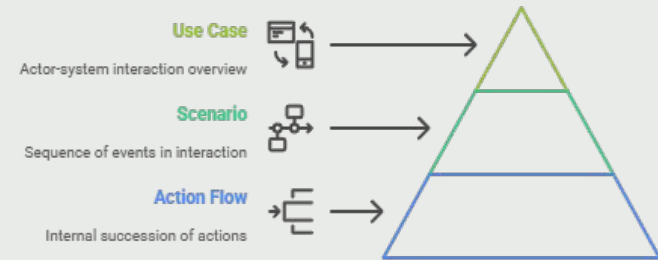
SysML v2 offers multiple ways to describe behavior — each with a different focus.

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

System Interaction Hierarchy



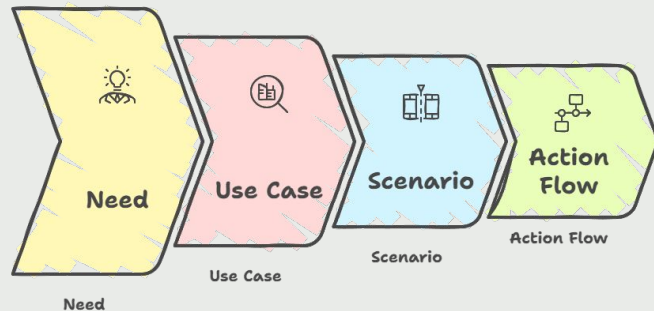
How to Derive Use Cases from Stakeholder Needs

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

Transformation of Need to Action Flow



Example



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.

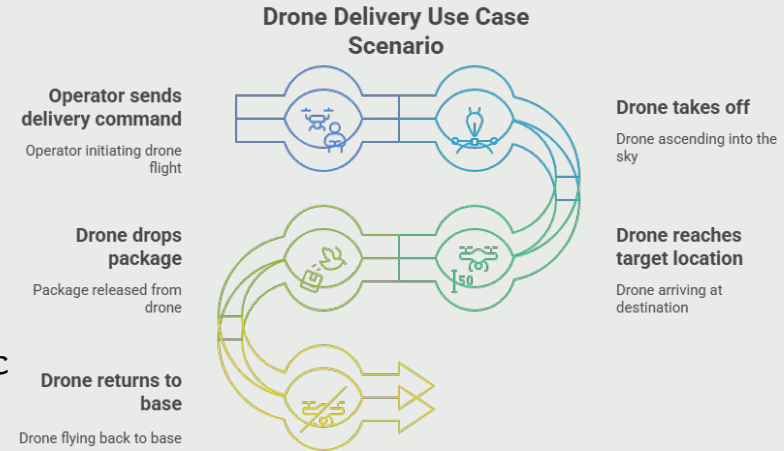
- The **actor** initiates the interaction — someone or something external
- The **subject** is the system or part responsible for fulfilling the goal
- The **objective** describes the intended outcome from the user's perspective
- The **require** relationship links the Use Case back to a stakeholder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


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**



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

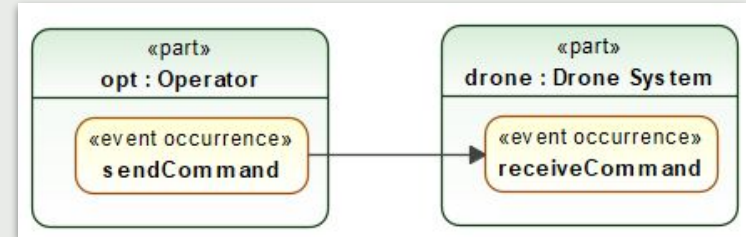
Event: `packageDropped` — drone confirms action is complete

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



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

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.

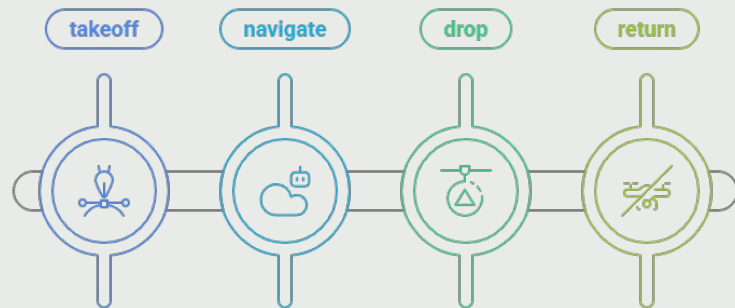
```
part def Operator;
part def 'Drone System';
item def Command;
part opt : Operator {
  event occurrence sendCommand {
    | out item item1;
  }
}
part drone : 'Drone System' {
  event occurrence receiveCommand {
    | in item item1;
  }
}
message of item1 : Command from opt.sendCommand to drone.receiveCommand;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

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



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;  
}
```

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;  
}
```

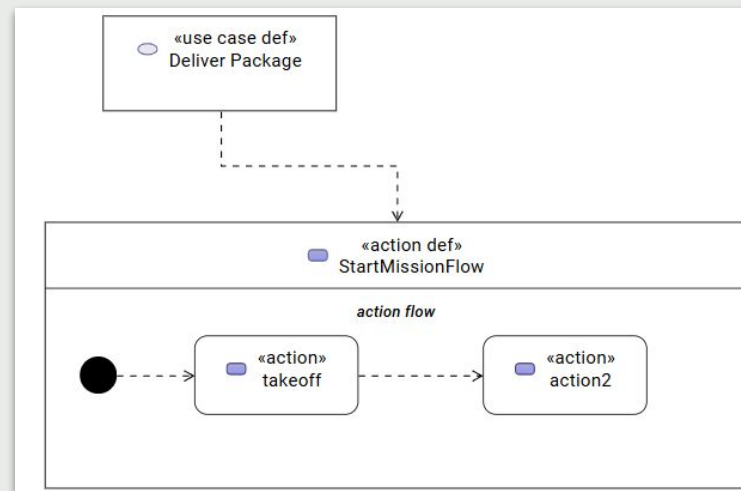


Example Breakdown – Drone Takeoff Scenario

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**

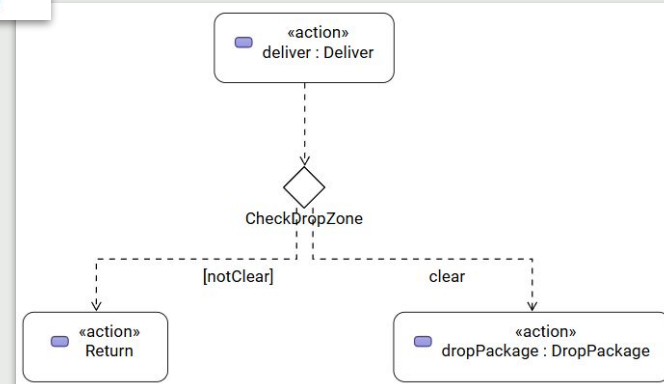


Modeling Decision Logic with 'Decide'

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

It represents 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;
```



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



Good

Clear and effective communication



Bad

Confusing and ineffective communication

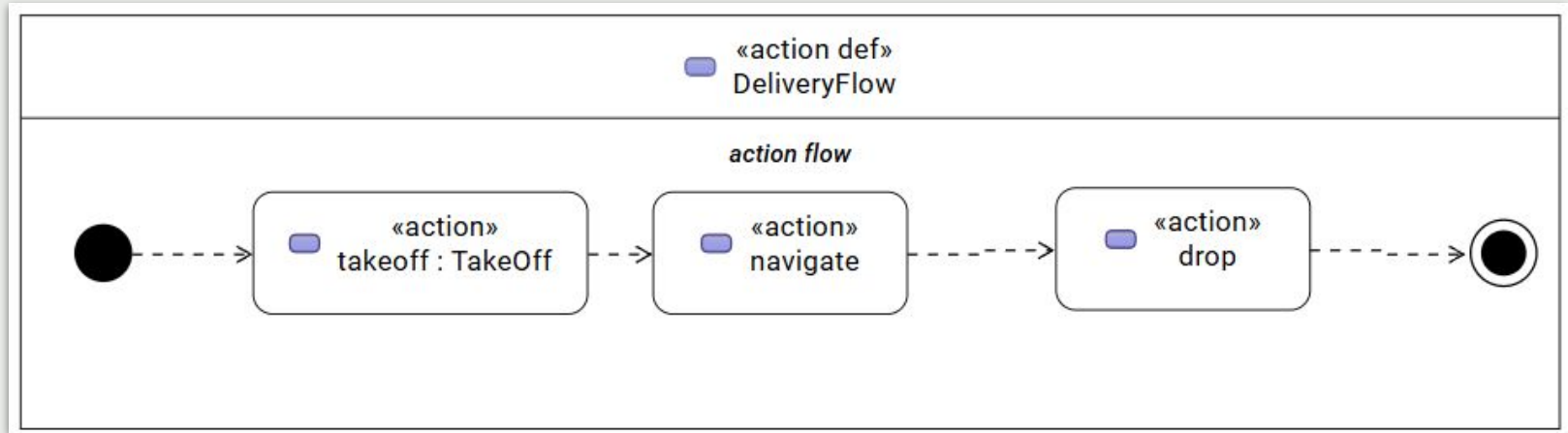
Mapping Scenario Steps to Action Nodes

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."



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



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.



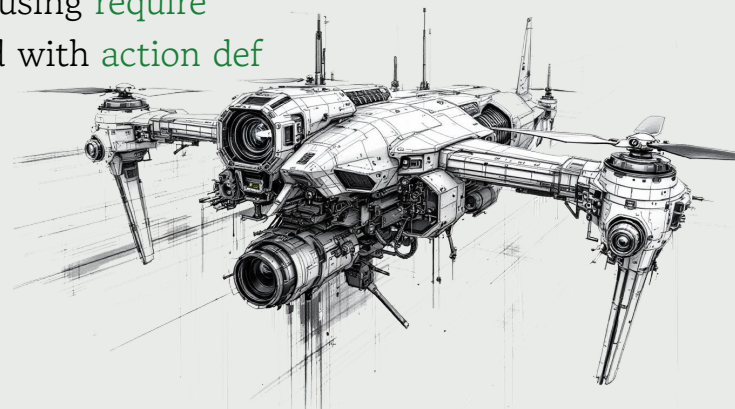
“The driver presses the start button. The vehicle checks the battery. If okay, it starts the engine and initializes the display. If not, it sends an alert.”

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



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

QUESTION!