

# How to read\* SysML Models v 3.2

\*(and understand and critique)

---

**Dr. Bruce Powel Douglass, Ph. D.**

**Senior Principal Agile Systems Engineer**

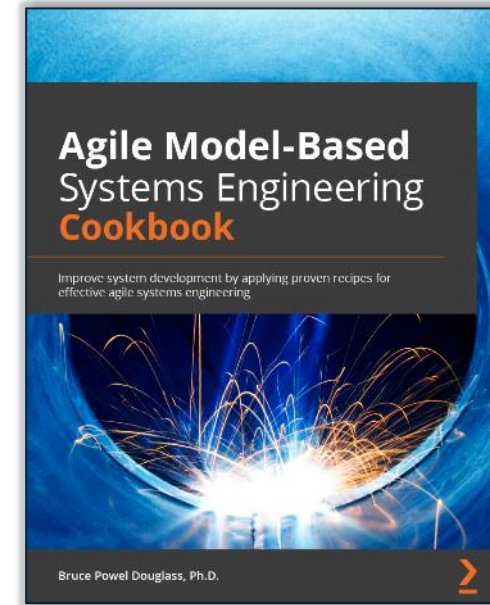
**L260 Systems Engineering Tech Center**

**The MITRE Corporation**

**[bdouglass@mitre.org](mailto:bdouglass@mitre.org)**

This document is publicly released by The MITRE Corporation following its public release standards. The MITRE Corporation retains sole copyright.

# About the Course Author



## Bruce Douglass, Ph.D.

- Senior Principal Agile Systems Engineer
  - Systems Engineering Tech Center
  - The MITRE Corporation
  - Can be reached at [bruce.douglass@outlook.com](mailto:bruce.douglass@outlook.com)
- Contributor to UML standard
- Contributor to SysML standard
- Developer of UML Dependability Profile
- Former Cochair RTAD Task Force for the OMG

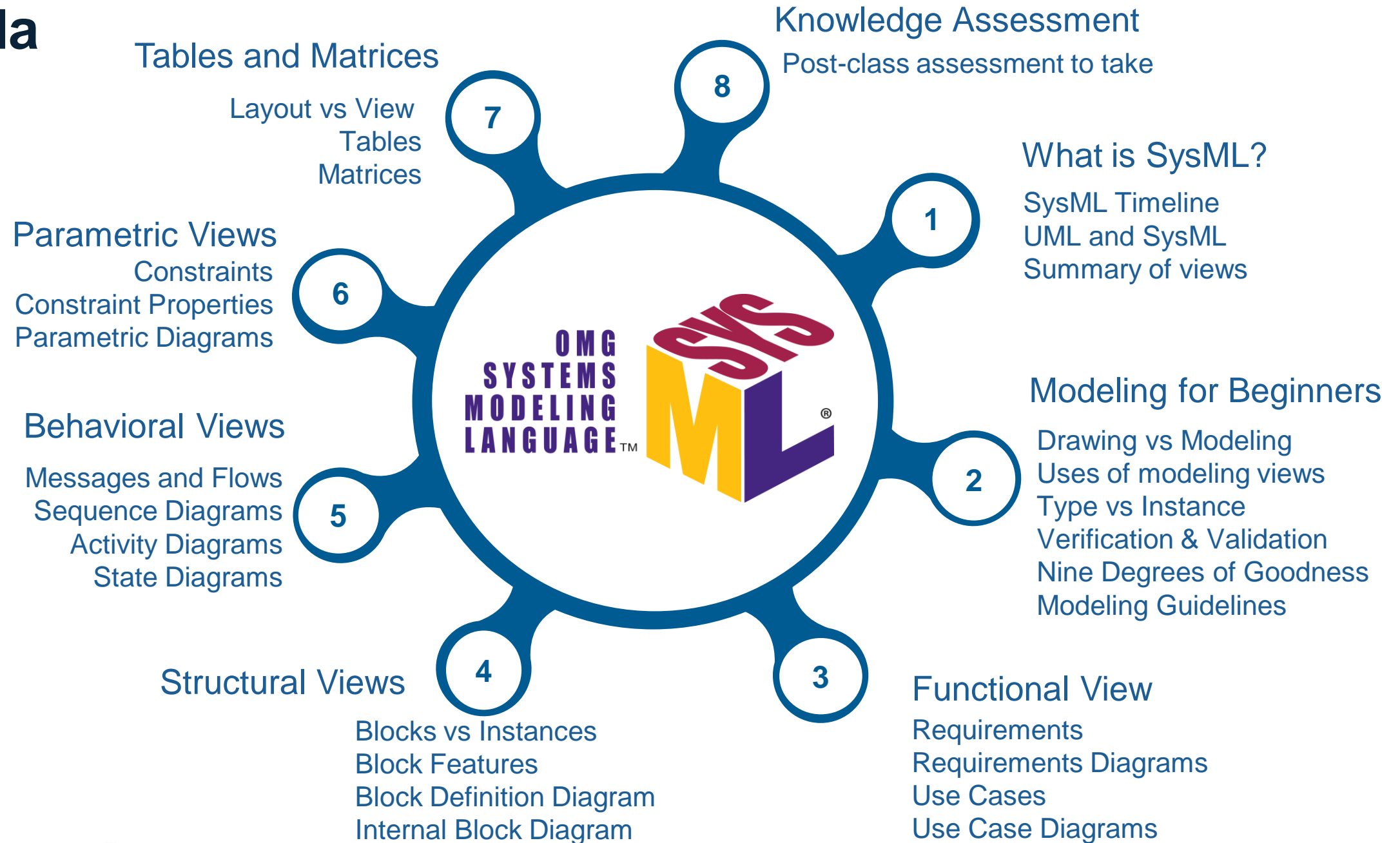
**For class materials visit**

**<https://de.mitre.org/training-assets/>**

**For more information on modeling visit**

**[www.bruce-douglass.com](http://www.bruce-douglass.com)**

# Agenda



# Course Learning Objectives

After completing this training, participants will be able to:



**Review a model viewpoint**



**Critique basic diagrams and elements in SysML:**

- **Requirements diagrams**
- **Use case diagrams**
- **Block definition diagrams**
- **Internal block diagrams**
- **Sequence diagrams**
- **Activity diagrams**
- **State diagrams**
- **Parametric diagrams**

Slides with this icon are “Key Takeaway” Slides.

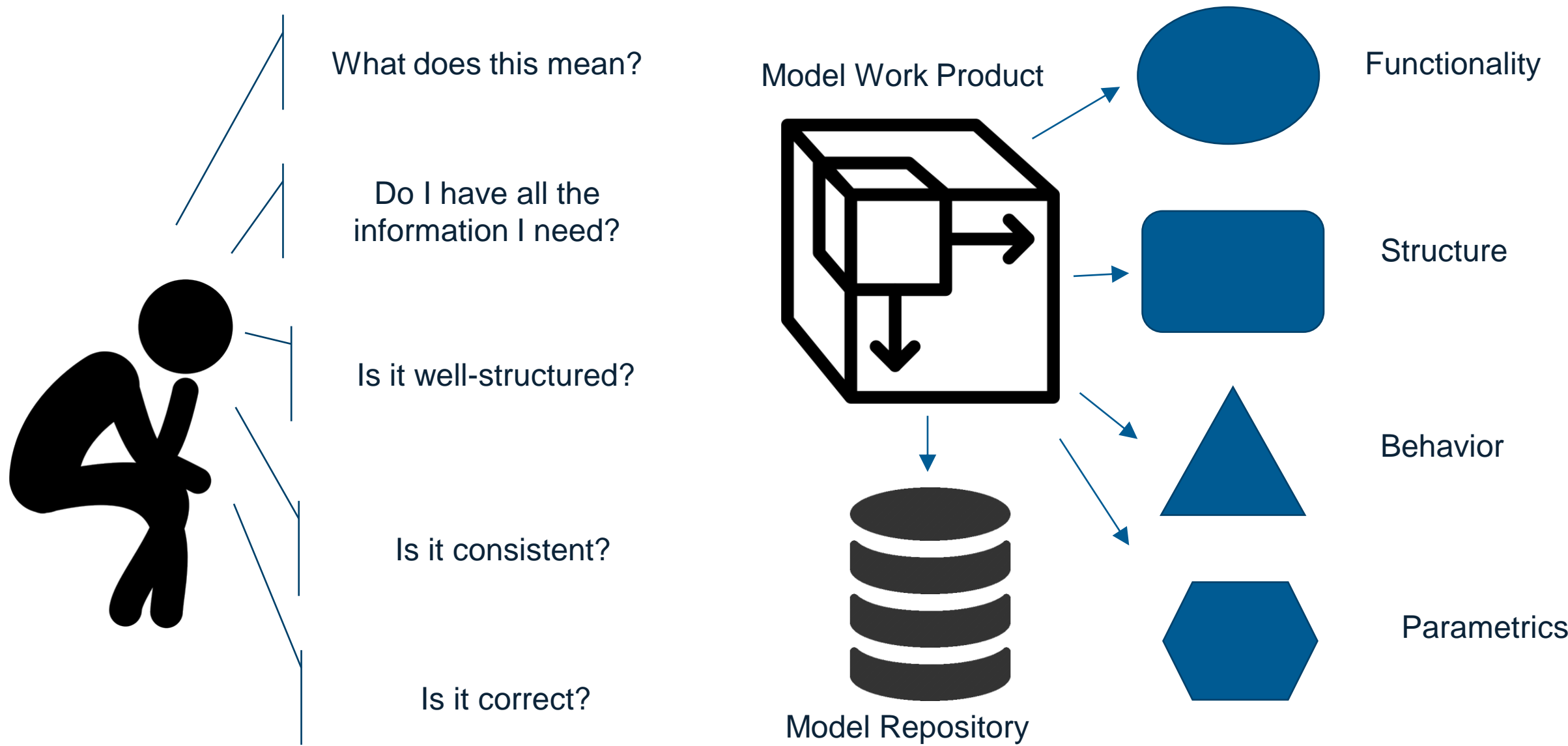


Slides with this icon are Checklist Slides.



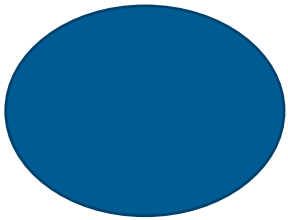


# Scenario: Acquisition Officer Receives a Design Model



# SysML Views

## SysML Pillars



Functionality



Structure



Behavior



Parametrics

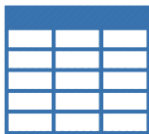
## SysML Views



Requirements Diagram



Use Case Diagram



Requirements Table



Requirements Trace Matrix

## SysML Model Elements

Requirement  
Use Case  
Association  
Generalization  
«include»  
«extend»  
«deriveReq»  
«trace»  
«satisfy»  
«verify»  
«allocate»

# SysML Views

## SysML Pillars



Functionality



Structure



Behavior



Parametrics

## SysML Views



Package Diagram



Block Definition Diagram



Internal Block Diagram



Allocation Matrix

## SysML Model Elements

Package

Block

Value property

Flow property

Operation

Port

Part

Association (reference prop)

Aggregation

Composition (part property)

Generalization

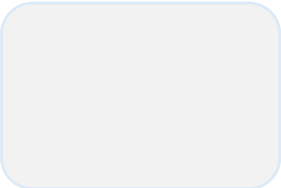
Dependency

# SysML Views

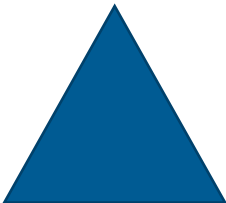
## SysML Pillars



Functionality



Structure



Behavior



Parametrics

## SysML Views



Sequence Diagram



Activity Diagram



State Diagram



Allocation Matrix

## SysML Model Elements

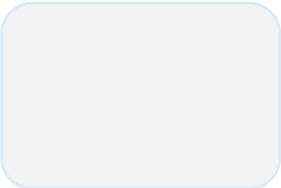
- Message
- Signal
- Call
- Activity
- Action
- Call Behavior Action
- Call Operation Action
- State
- Event
- Transition
- Control Flow
- Object Flow

# SysML Views

## SysML Pillars



Functionality



Structure



Behavior



Parametrics

## SysML Views



Block Diagram



Parametric Diagram

## SysML Model Elements

- Constraint Block
- Constraint Property
- Constraint
- Constraint Parameter
- Binding Connector



# What is SysML?

SysML is derived from UML

SysML Timeline

UML vs SysML



# What is SysML?

- A graphical modeling language in response to the UML for Systems Engineering RFP developed by the Object Management Group (OMG), International Council on Systems Engineering (INCOSE), and AP233
  - a UML Profile that is both a subset and extension to UML 2
- Designed specifically for the Systems Engineering domain with extensions for requirements and analysis
- Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- SysML is the most common way to represent systems engineering information in a rigorous, structured way by storing the information in **models**. We discuss models in more detail shortly.
- The pervasive application of models for systems engineering is known as Model-Based Systems Engineering (MBSE)

Important! **At a basic level of use, UML and SysML are the same language**, with only minor naming differences between them.

- More advanced features of SysML will highlight the differences between them.

Like UML, SysML is a *language* and is process-agnostic.



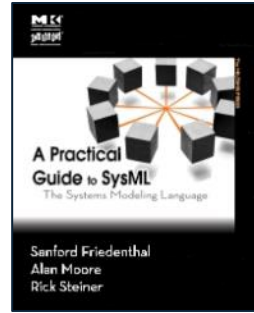
# SysML History



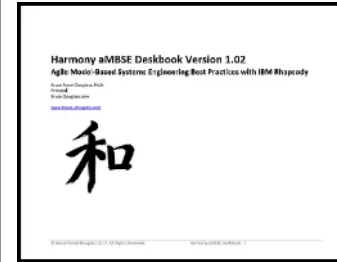
UML 1.1 Adopted by the Object Management Group (OMG)

Initial release of SysML for adoption

Friedenthal et. al. release **A Practical Guide to SysML**

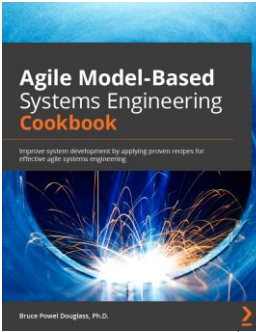


Bruce Douglass releases **Harmony Agile Model-Based Systems Engineering** (Harmony aMBSE) process



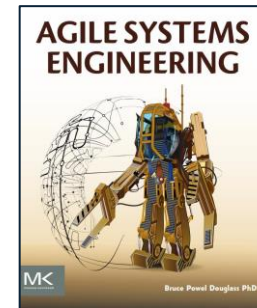
Work begun on SysML 2.0

Bruce Douglass publishes **Agile Model-Based Systems Engineering Cookbook**



SysML 1.6 released

Bruce Douglass publishes **Agile Systems Engineering** book



Click here to learn about the latest release of SysML  
<https://www.omg.org/spec/SysML/About-SysML/>

# UML and SysML – The Preeminent Modeling Languages

## SysML Pillar

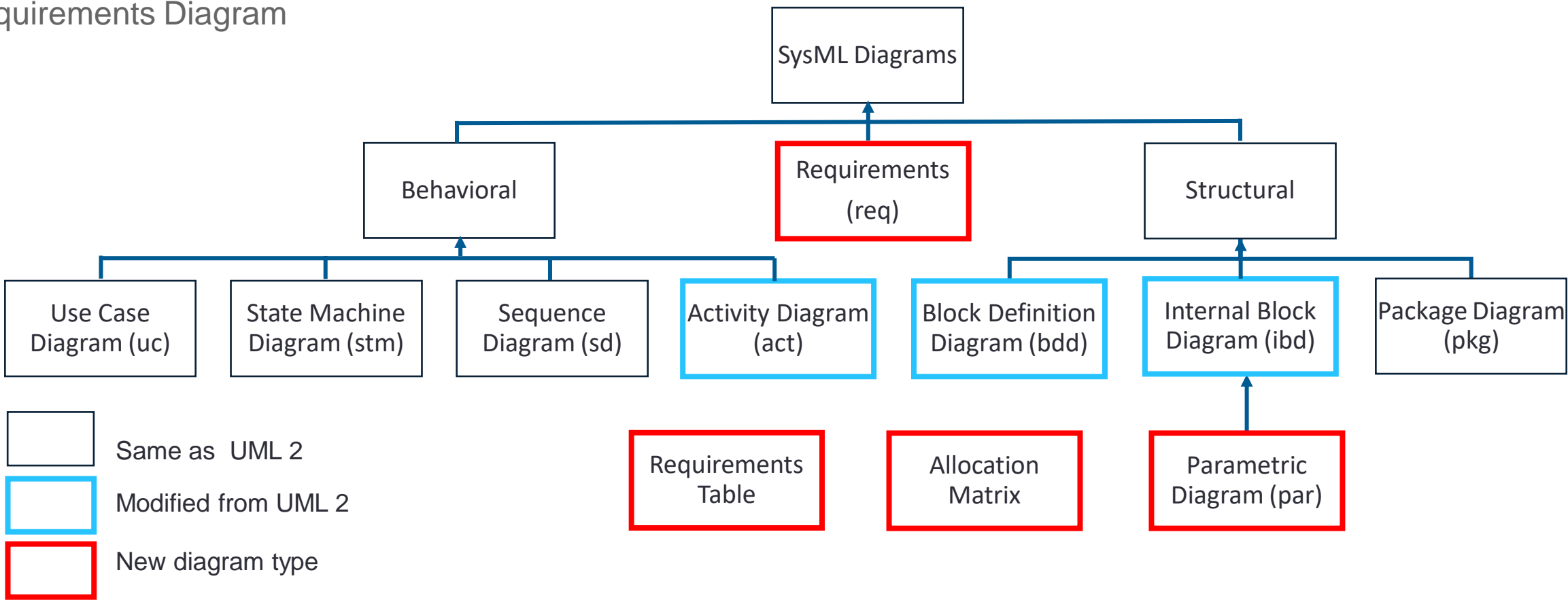
UML (software)	UML4SysML	SysML (systems)
<b>Structural</b> Class Diagram Structure Diagram Deployment Diagram Class Object (Instance) Attribute Operation	Package Profile Stereotype Interface Package Diagram Signal Operation Port	Block Part Value Property Units SI Units Model Library Block Definition Diagram Internal Block Diagram Proxy port Full port Interface Block
<b>Functional</b>	Use case Use Case Diagram	Requirement Requirement Diagram Requirement Table Allocation Matrix
<b>Behavioral</b> Communication Diagram Timing Diagram Interaction Overview	State Diagram Activity Diagram Sequence Diagram { stream } «rate»	«continuous» «discrete» «control» «probability»
<b>Parametric</b>	Constraint	Constraint Block Constraint Parameter Parametric Diagram Parametric Constraint

At the UML 101/SysML 101 level, they are *the same*, except some elements are renamed

# Nine SysML Views

The nine SysML diagrams are categorized as follows:

- Behavioral Diagrams - dynamic change of system **behavior** over time
- Structural Diagrams - static system **structure** diagrams
- Requirements Diagram





# Characteristics of Predefined SysML Views

View	Type	UML2 Analog	Lifecycle usage	Essential	Dynamic simulation	Computational	Supports code gen	Formal
Requirements Diagram (req)	Static Functionality	n/a	Requirements Specification; Functional Analysis					
Use Case Diagram (uc)	Static Functionality	Use case diagram	Requirements Specification; Functional Analysis	☑				
Activity Diagram (act)	Dynamic Behavior	Activity diagram – minor changes	All	☑	☑		☑	☑
Sequence Diagram (sd)	Interaction Behavior	Sequence Diagram	All	☑	☑			☑
State Diagram (stm)	Dynamic Behavior	State Diagram	All	☑	☑		☑	☑
Block Definition Diagram (bdd)	Static Structure	Class Diagram (moderate change)	Architecture; Design	☑			☑	☑
Internal Block Diagram (ibd)	Static Structure	Structure Diagram (moderate change)	Architecture; Design	☑			☑	☑
Parametric Diagram (par)	Static Functionality	n/a	All			☑		☑
Package Diagram (pkg)	Static Structure	Package diagram	All					
Requirements Table	Static Table	n/a	Requirements Specification; Functional Analysis					
Allocation Matrix	Static Matrix	n/a	All					

# Modeling For Beginners

Drawing vs Modeling

What's a model?

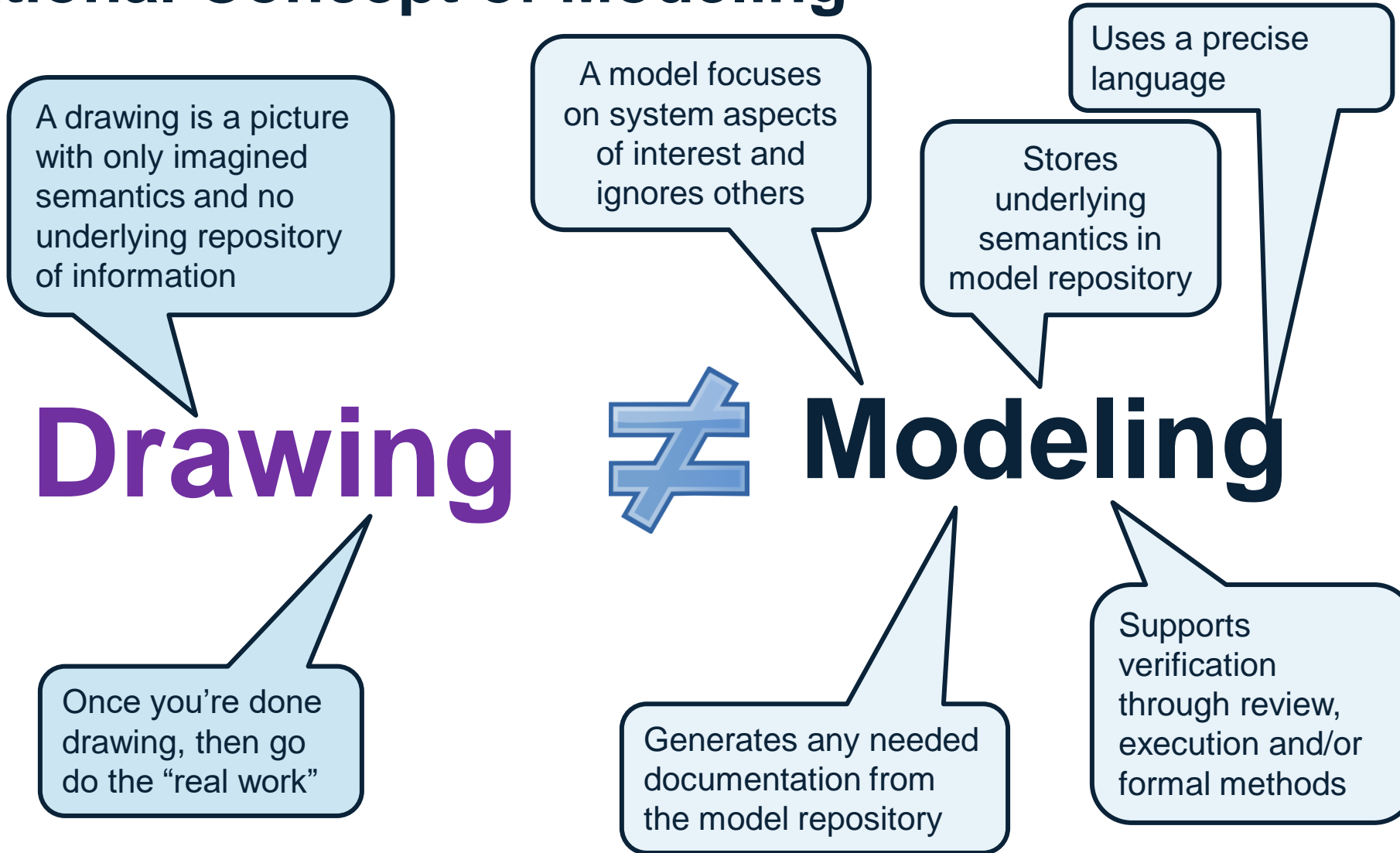
Models & Views

Nine degrees of Goodness

Checklist



# Foundational Concept of Modeling



**Note: it IS possible to use a modeling tool solely for drawing and not modeling, but it's not a good idea!**



# So What IS a Model exactly?

**Modeling** is the development of a set of system data of relevant systems and their properties

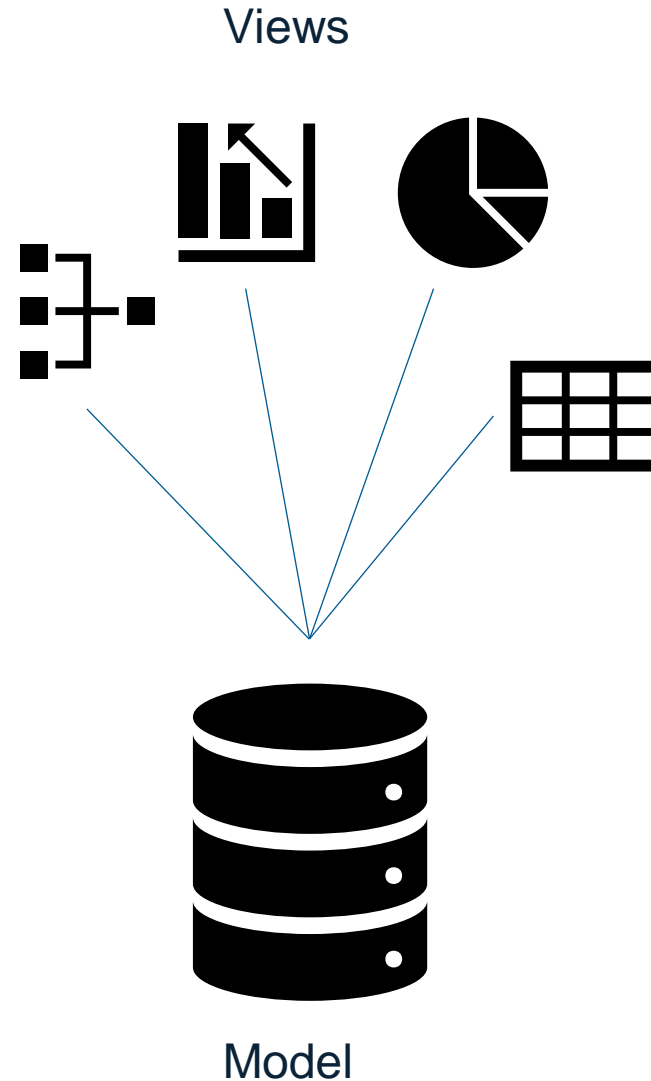
**Models** have views (e.g. diagrams)

**Diagrams** show subsets of eng. data

**Diagrams** have singular purpose

**Diagrams** answer questions

**Diagrams** support specific reasoning



**Models** have scope



**Models** have purpose

**Models** have precision

**Models** have accuracy

**Models** have fidelity

**Models** are falsifiable

**Models** are verifiable

**Models** *are*  
*interconnected data!*

# Uses of Diagrams and Tabular Views

- Data Entry
  - Drawing diagrams or entering data into tables/matrices is a way of entering information into the model
  - When you create an element on the diagram, the model either
    - Refers to an existing element, and updates it based on your actions, or
    - Creates a new element in the model repository
- Model visualization
  - Creating a diagram or tables allows you to create a view of a subset of the model information
- Simulation / Execution Debugging & Execution Control
  - Some modeling tools provide special diagrams and tools to control execution, insert events, change values, set breakpoints, etc.



# Drawing or Model?

- It's a Drawing (and not a model) if **any** of the following are true:
  - There is no data whose current value is dynamically reflected in the view
  - It cannot be reused, specialized or extended in other contexts
  - Changes made here are not reflected anywhere else
  - Disconnected from any analytical tools
  - Cannot, in principle, be verified by any means other than “inspection”
  - It cannot be used in a query to answer a question about the data it represents
  - Elements on it cannot be linked backward to source elements nor forward to subsequent elements

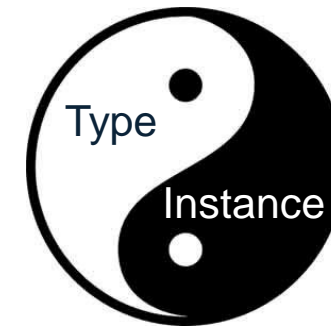
# Model Elements

- May be
  - Structures
  - Functionalities
  - Behaviors
  - Relations
  - Properties / features
- Best practice
  - All model elements should have useful, meaningful names
  - All model elements should be specified with enough rigor and precision to meet their purpose
  - All important model elements should have meaningful descriptions that address the *why* of the model element more than the *how*. In general,
    - *Purpose*
    - *Explanation*
    - *Preconditions*
    - *Post-conditions*
    - *Assumptions (invariants)*

# Theory Time

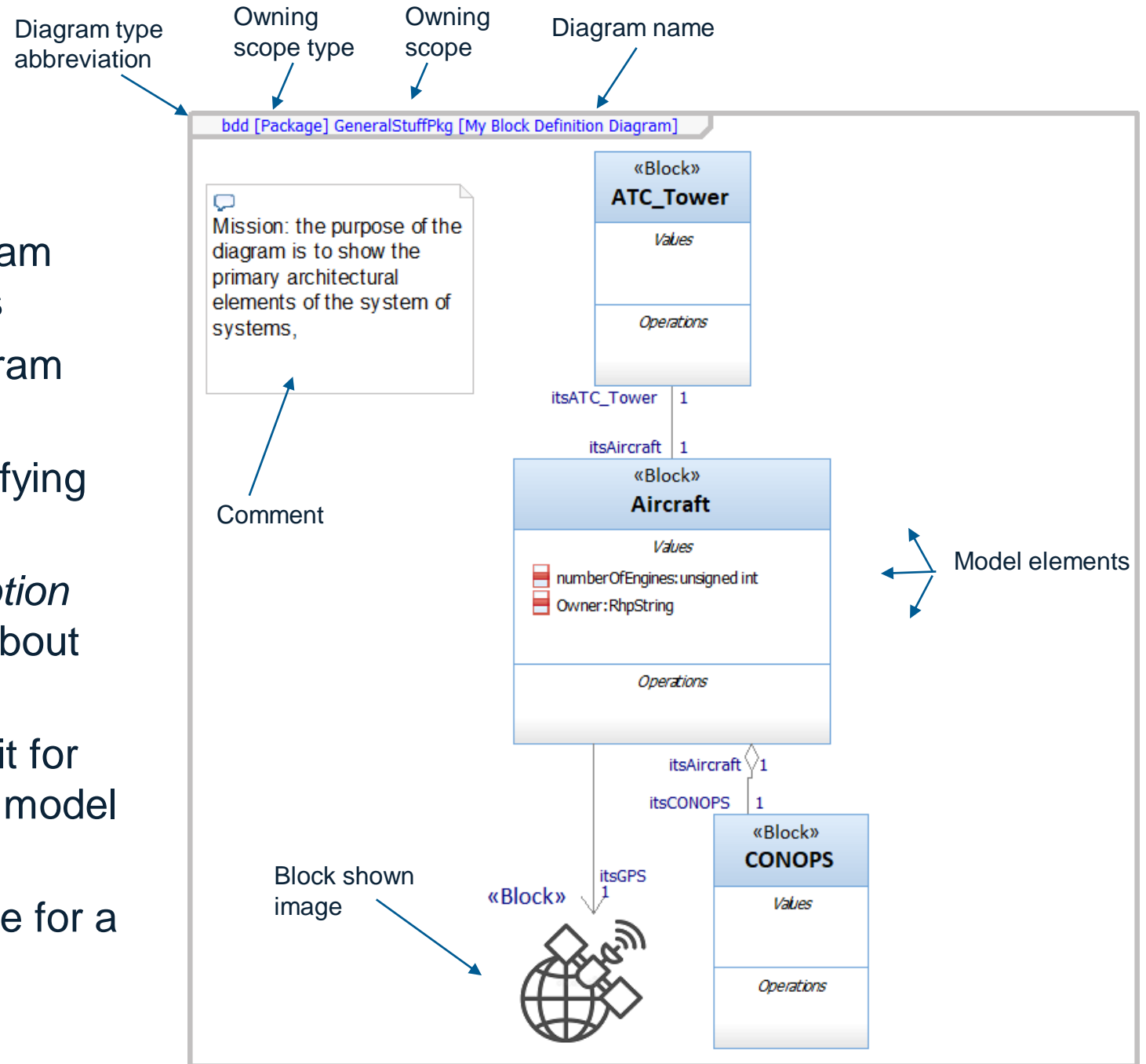


- Throughout SysML you will find the **type-instance dichotomy**:
  - **Types** exist at analysis/design time and specify the properties of all their instances
    - Blocks are a kind of type
  - **Instances** exist at run-time and their structure and behavior is defined by their types. However, the instances all have their own copy of their data and state.
    - Instances (also known as parts) are instances of a block
- Example
  - *Block – Instance (part)*
  - *Association – connector*
  - *Table layout – table view*
- Similarly there is a **specification-view dichotomy**.
  - Model element specifies its properties
  - A diagrammatic view visualizes some set of those properties

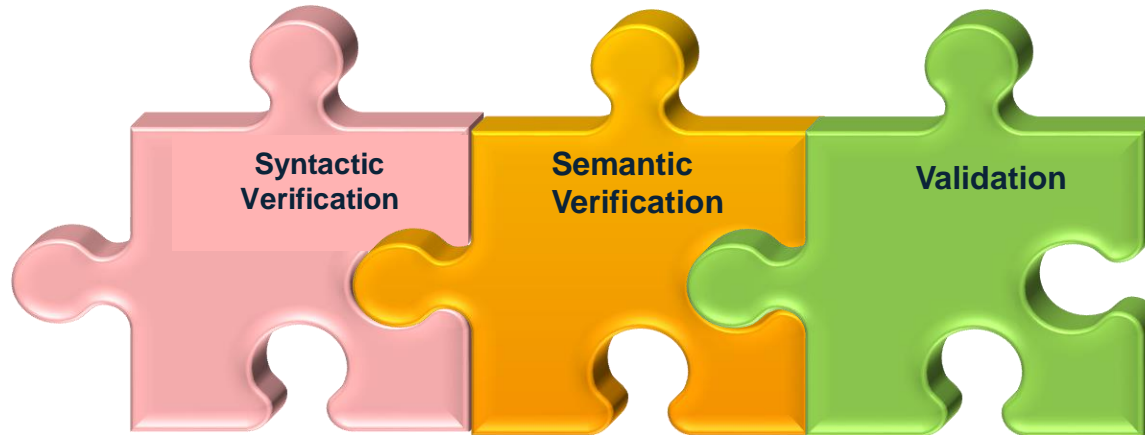


# Before we get started in SysML

- All diagrams have an *optional* diagram frame in addition to model elements
- You can add comments to any diagram
  - We recommend virtually every diagram have a comment identifying its scope and purpose
- Every model element has a *Description* property that can hold information about the element
- *Packages* are the organizational unit for models – (almost) all diagrams and model elements reside in some package
- You can substitute a graphical image for a model element if desired



# What do we mean by “verification & validation” of work products?



## Semantic: Is the content correct?

- *Compliance in meaning*  
Performed by engineering personnel  
Three basic techniques
- **Semantic review** (subject matter expert & peer) – most common, weakest means
- **Testing** – requires executability of work products, impossible to fully verify
- **Formal methods** – strongest but hard to do and subject to invariant violation

## Syntactic: Is it well-formed?

- “Compliance in form”  
Performed by quality assurance personnel
- **Audits** – work tasks are performed as per plan and guidelines
- **Syntactic review** – work products conform to standard for organization, structure and format

## Valid: Does it solve the right problem?

- Validation = “meets the stakeholder need”  
Performed by customer + engineering  
Some common techniques
- **Review** – (subject matter expert & customer) – most common, weakest
- **Simulation** – show simulated input → outputs
- **Sandbox** – exploratory usage in constrained environment
- **Flight test** – demonstration of system capabilities
- **Deployment** – early usage of system of partial capability

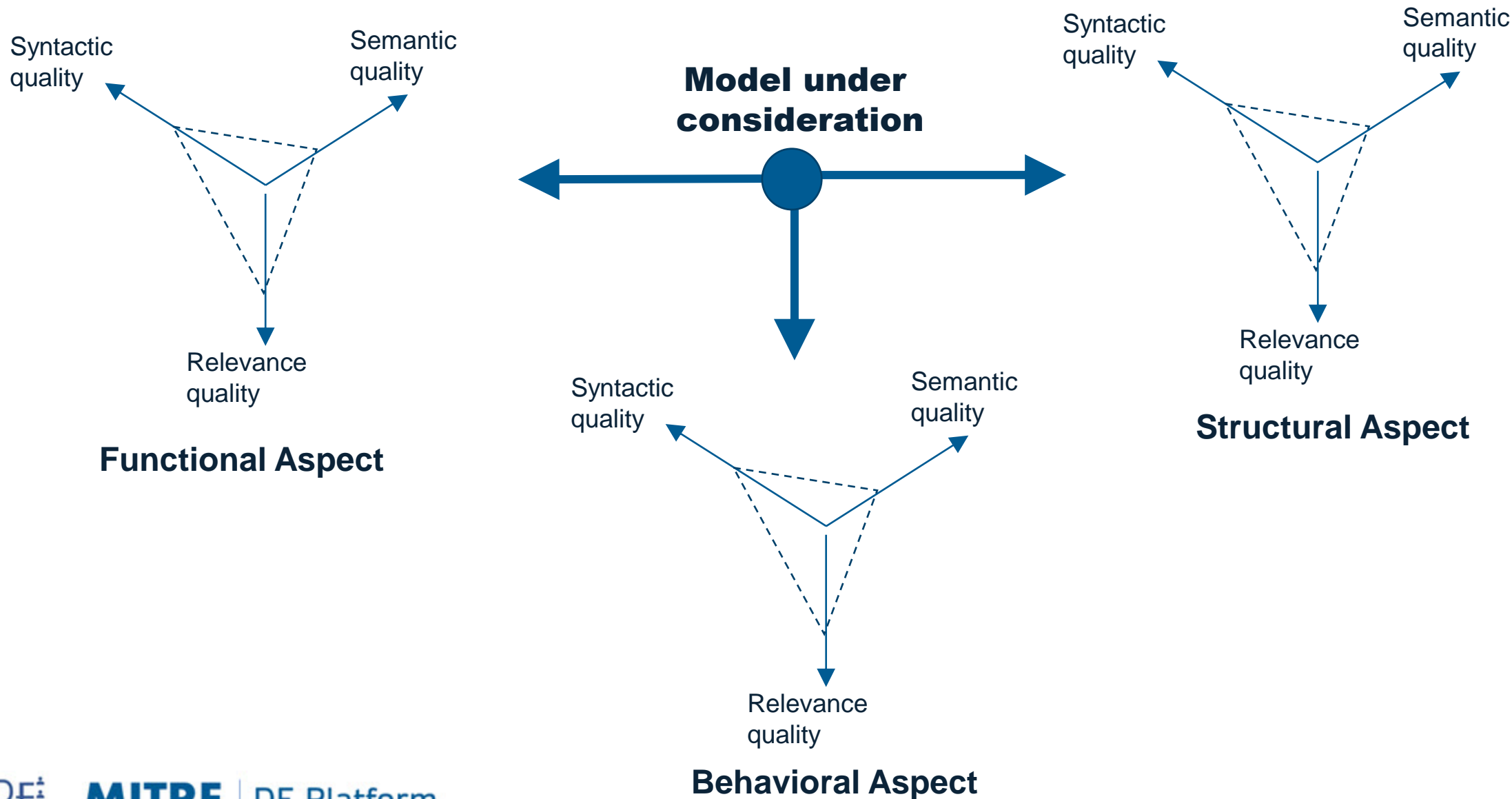


# Evaluating Models – Nine Degrees of Goodness



- Model views cannot generally be considered in isolation but must be critiqued in combination with other related information
  - In SysML a model subset of focus is called a *viewpoint*
- To evaluate a model, include:
- Consider 3 areas of focus
  - Functionality
  - Structure
  - Behavior
- Consider three aspects of quality for each
  - Syntactic correctness (well-formedness)
  - Semantic correctness (accuracy)
  - Validity (relevancy to the problem)
- Example:
  - To adequately review a use case analysis, several views would be required:
    - Functionality
      - Requirements diagram and tables
      - Use case diagrams
    - Structure
      - System context diagram
    - Behavior
      - Multiple scenarios (sequence diagrams)
      - State Diagram
      - Activity diagram
  - Evaluate **each** in terms of
    - Well-formedness
    - Accuracy
    - Validity (relevance)

# Nine Degrees of Goodness for Evaluating Models



# Modeling Guidelines

- Every modeling project should have a modeling guidelines standard that is used to specify project-specific rules for model development. This should include
  - Naming conventions
    - Common: Camel Case: **ThisIsABlock**
    - Common: Underscores: **This\_Is\_A\_Block**
    - Types are generally named beginning with uppercase: **RadarTrack**
    - Type properties and instances generally named beginning with lowercase **enemyTrack:RadarTrack**, or **RadarTrack::range** (**range** is a value property of **RadarTrack**)
  - Model Organization
  - Model Federation
  - Model Scope and Purpose
  - Modeling language and subsets permitted
  - Action language
  - Diagrammatic rules and conventions



For example, see modeling guidelines at <https://www.bruce-douglass.com/papers>



# General Model Checklist

- ☐ Does the model have a *Model Overview Diagram* that describes the model purpose, organization, and content?
- ☐ Is the purpose, scope, level of precision of the model clearly identified?
- ☐ Is the model organization reasonable for its purpose?
- ☐ Is there a clearly identified modeling guidelines standard applied?
- ☐ Does the model adhere to the modeling guidelines?
  - ☐ Does it follow the naming convention?
  - ☐ Does it follow the organizational schema?
  - ☐ Does it contain the recommended model elements?
  - ☐ Does it contain the recommend modeling views?
- ☐ Is the model compliant with the relevant external standards?
- ☐ Do all important model elements have a useful and meaningful description?
- ☐ Do all important views have a stated mission / purpose / objective?

# Basic Functional Modeling in SysML

Requirements

Requirements Diagrams

Requirements Tables

Use Cases / Actors

Use Case Diagrams

Checklists



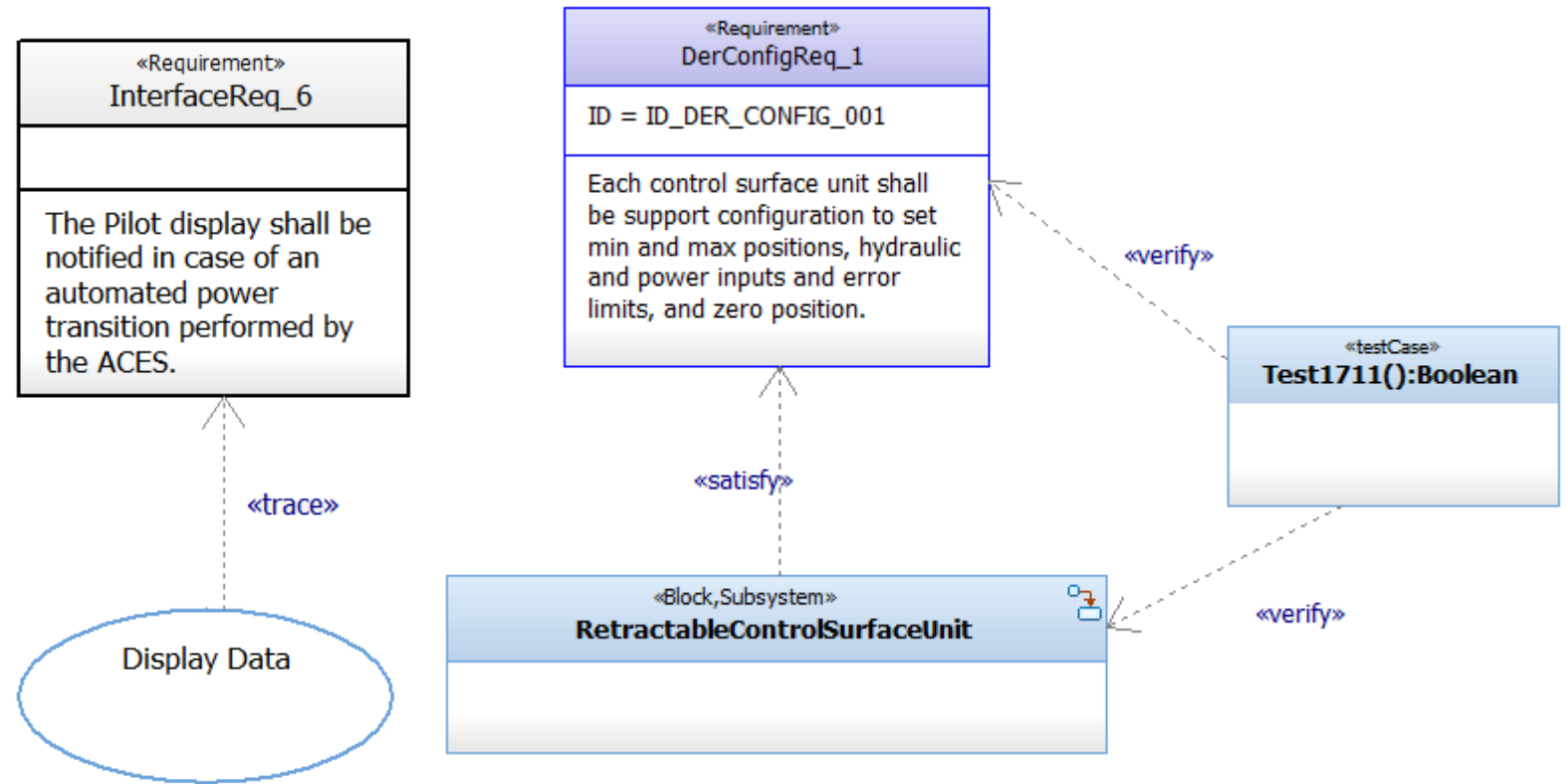
# Requirements

- Requirement is an element defined in SysML
  - A requirement specifies a capability or condition that must (or should) be satisfied. A requirement may specify a function that a system must perform or a performance condition a system must achieve.
  - A requirement is defined as a stereotype of UML Class subject to a set of constraint
    - Not allowed to have operations, attributes, generalization
  - Has properties
    - ID
    - Text (Specification)
    - Description
- Requirements may appear on ANY SysML diagram
- Requirements may be linked to requirements tools in one of two ways
  - Copy & Sync
  - Reference “remote resources” (on Jazz Platform)

Name	«Requirement» DerConfigReq_1
ID	ID = ID_DER_CONFIG_REQ_0001
Text	Each control surface unit shall be configurable to set min and max positions, hydraulic and power input error limits, and zero position.

# Requirements Relations

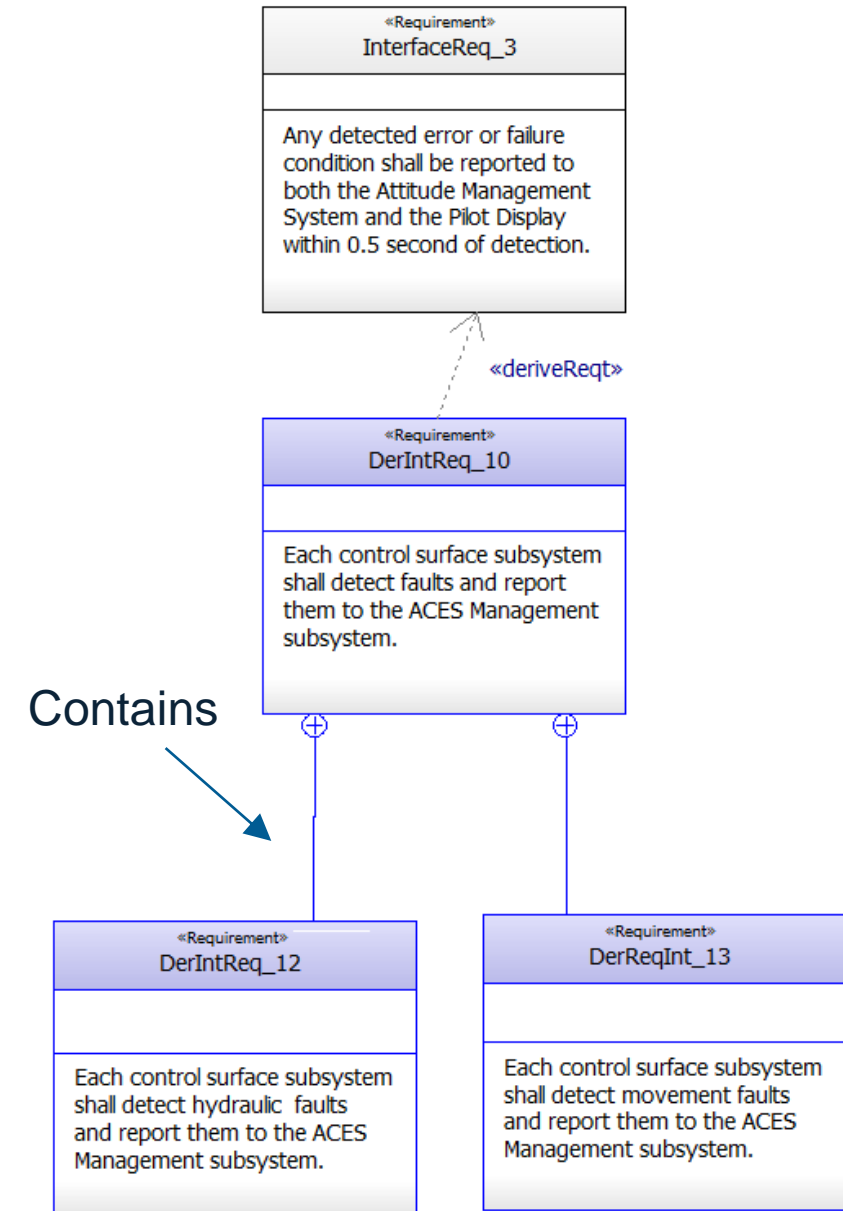
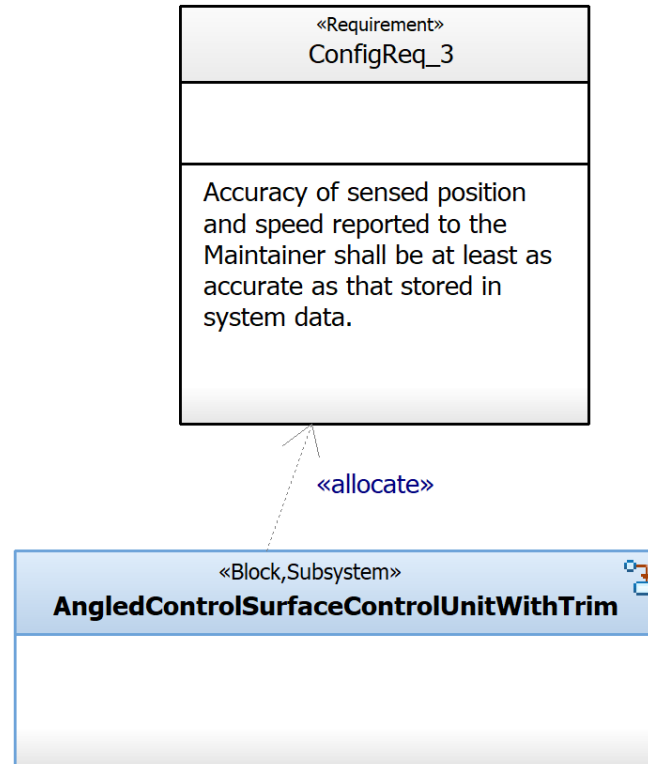
- «trace»  
A navigable relation from one element to another
- «satisfy»  
A relation from an element that meets a need to the corresponding requirement
- «verify»  
A relation from a test case to the element it verifies



Note: Directed relations have an *owner* end that “know about” the relation and can navigate to the *target* end. Use cases “know about” the *requirements* to which they trace, but the requirements don’t know how they are mapped to use cases. This is as it should be!

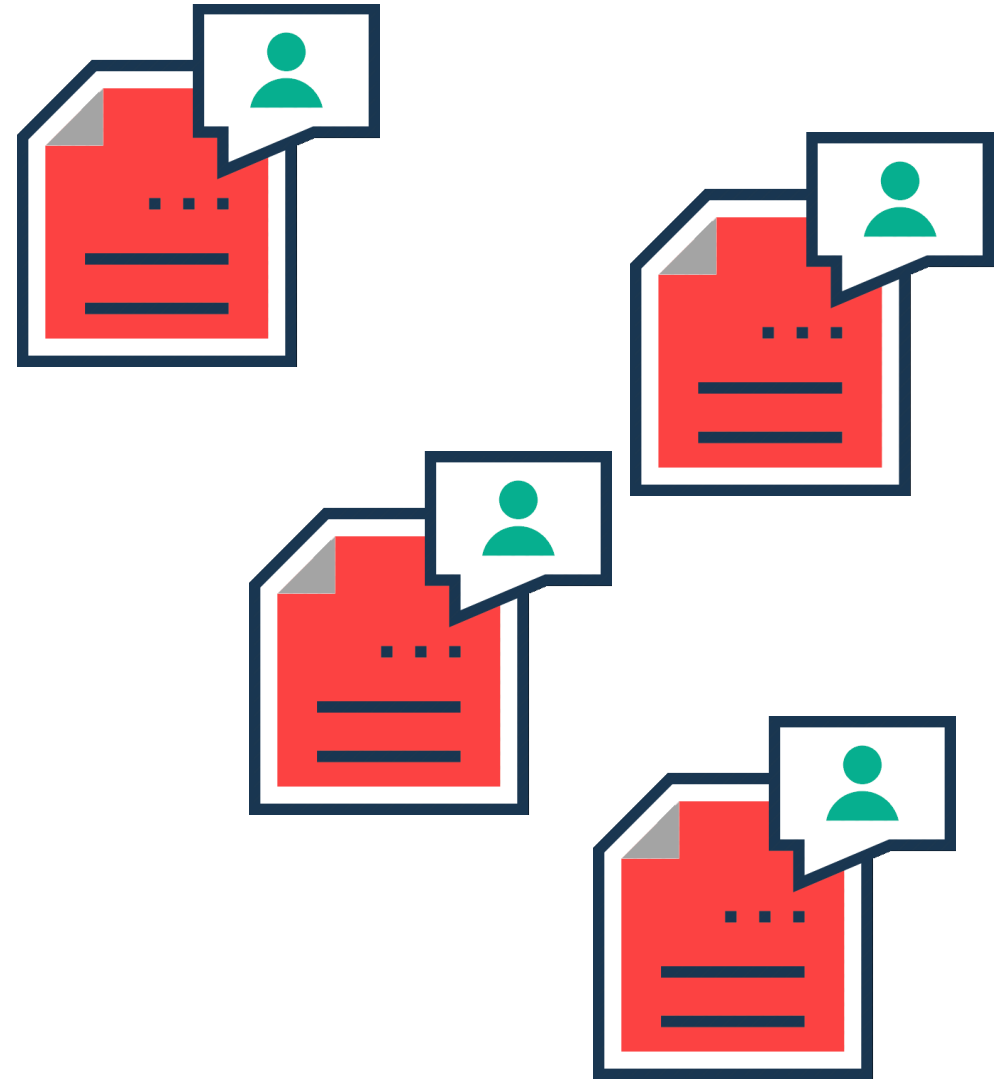
# Requirements Relations

- «allocate»  
A mapping of one element to another, such as the mapping the realization of a requirement to a design element
- Contains  
A design-time (not run-time) ownership relation
- «deriveReq»  
Indicates that one requirement is derived from another

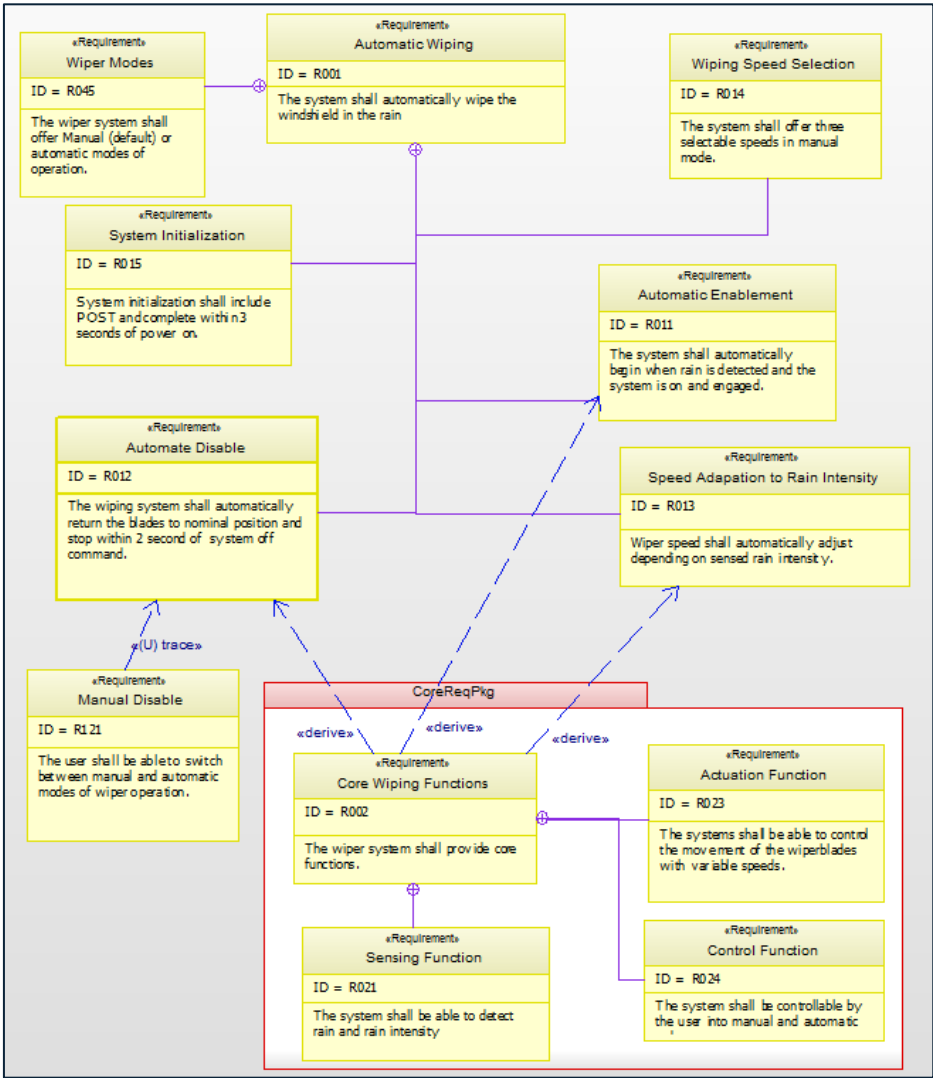




# Requirements Diagram



# Requirements Views



Requirements Diagram

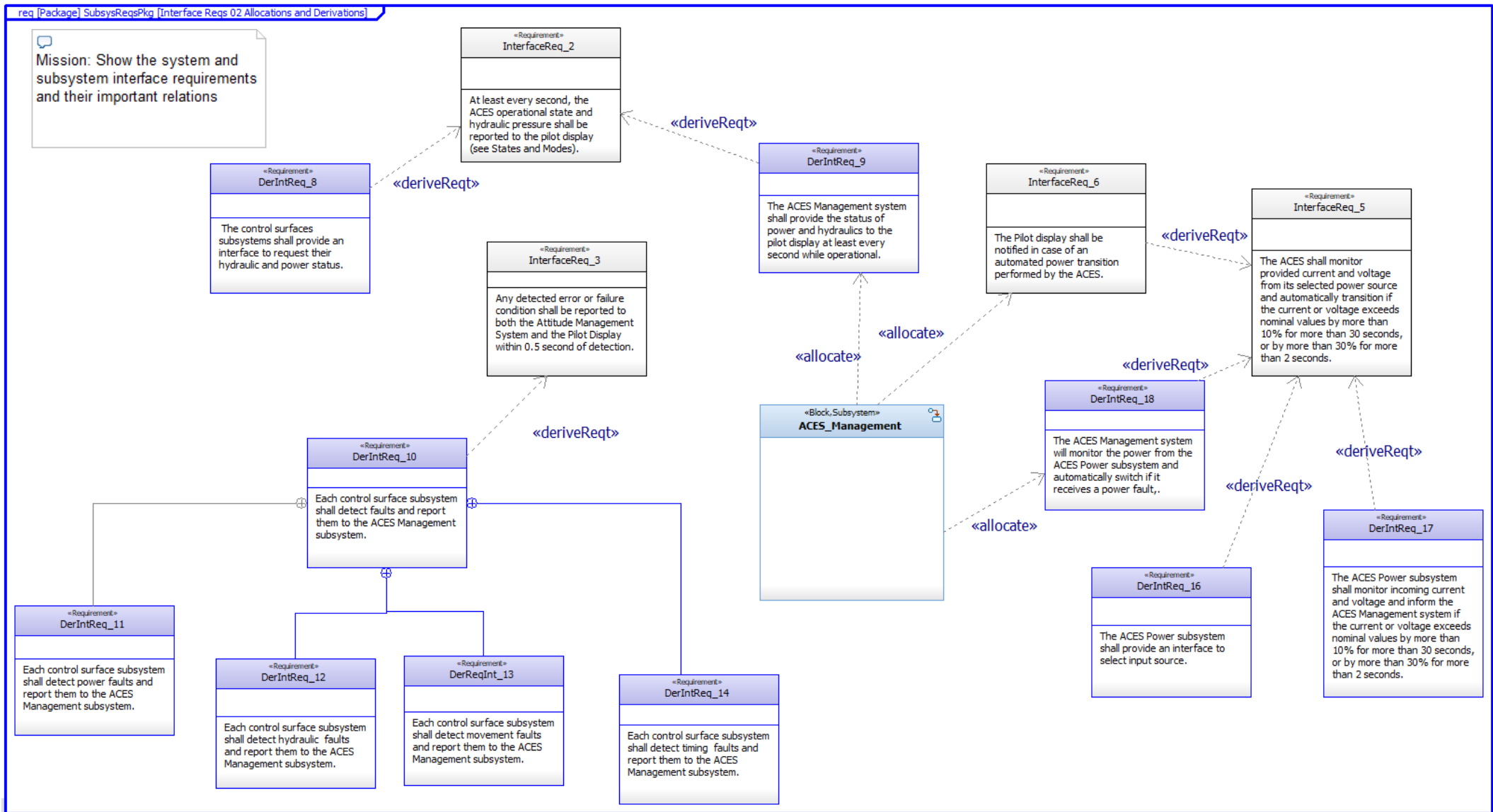
Name	Specification
requirement_16	The proximity detection shall classify three levels of promity alerts at the identified object level: informational (low), warning (moderate), and dangerous (high).
requirement_17	The promity detect capability will identify objects within the proximity field and provide range, position, and size.
requirement_18	The proximity detection capability will be automatically enabled when the system has passed power on self test and enters operational mode or under explicit user
requirement_19	The proximity detection capability shall be disabled only when then system is powered down or by explicit user command.
requirement_20	The proximity detection capability shall support range, object size threshold, and sensitivity (noise rejection) all combinations of CLUTTERED, SPARSE, and CLE speeds of SLOW_MOVING and FAST_MOVING.
requirement_21	The system shall notify the Pilot of current proximity detectiononn configuration values.
requirement_22	Proximity detect range shall be configurable in the range of 1 meter to 20 meters in 0.5 meter increments.
requirement_23	Object size detection threshold shall be setttable in apparent diameters from 20 cm to 100 cm, setttable in 5 cm increments.
requirement_24	Proximity detection sensitivity shall determine the level of noise rejection and shall be setttable in the range of LOW, MEDIUM, and HIGH.
requirement_26	When the proximity detection capability is operational, faults that prevent proximity detection shall be identified and the user shall be alerted within 1.0 seconds of
requirement_27	When active, alert tones shall sound and alert messages shall flash every every 3 seconds.
requirement_50...	The system shall employ a "deadman switch" approach to ensure that movement only occurs with active Pilot control.
requirement_50...	The system shall provide a surface level Emergency Shutdown function that is always available to the Pilot.
requirement_50...	Default proximity field size shall vary according with system velocity but may be overridden with explicit Pilot command.
requirement_62	To prevent spoofing, the GPS system shall identify a spoofing attempt if the GPS-computed position changes more than 100m within 1 second
requirement_61	To prevent spoofing, the GPS shall use military encryption for the military version of the system.
requirement_60	If more than 4 satellites are available, the redundancy shall be used to refine GPS time and position. Required GPS position accuracy shall be +/- 1 m.
requirement_59	The system GPS receiver shall operate in 3D GPS mode, and require at least 4 satellites of minimal signal strength to determine 3D position.
requirement_58	The accuracy of the GPS receiver clock shall be accurate within 1.0ns.
requirement_57	The syste, GPS radio frequency shall center around 1575.42 MHz (but be able to receive signals between 1560 and 1610 MHz) (L1 signal) and 1227.60 MHz but signals between 1150and 1400 Mhz (L2 signal) using military band of 10.23 million CDMA chips/second.

Requirements Table

	requirement_3	requirement_4	requirement_5	requirement_6	requirement_7	requirement_8
requirement_24						
requirement_25						
requirement_26						
requirement_28		requirement_4				
requirement_29	requirement_3					
requirement_30					requirement_7	
requirement_31					requirement_7	
requirement_32	requirement_3					
requirement_33	requirement_3					
requirement_34			requirement_5			
requirement_35						requirement_8
requirement_36				requirement_6		
requirement_37	requirement_3					
requirement_40						

Trace between stakeholder and system requirements

# Requirements Diagram



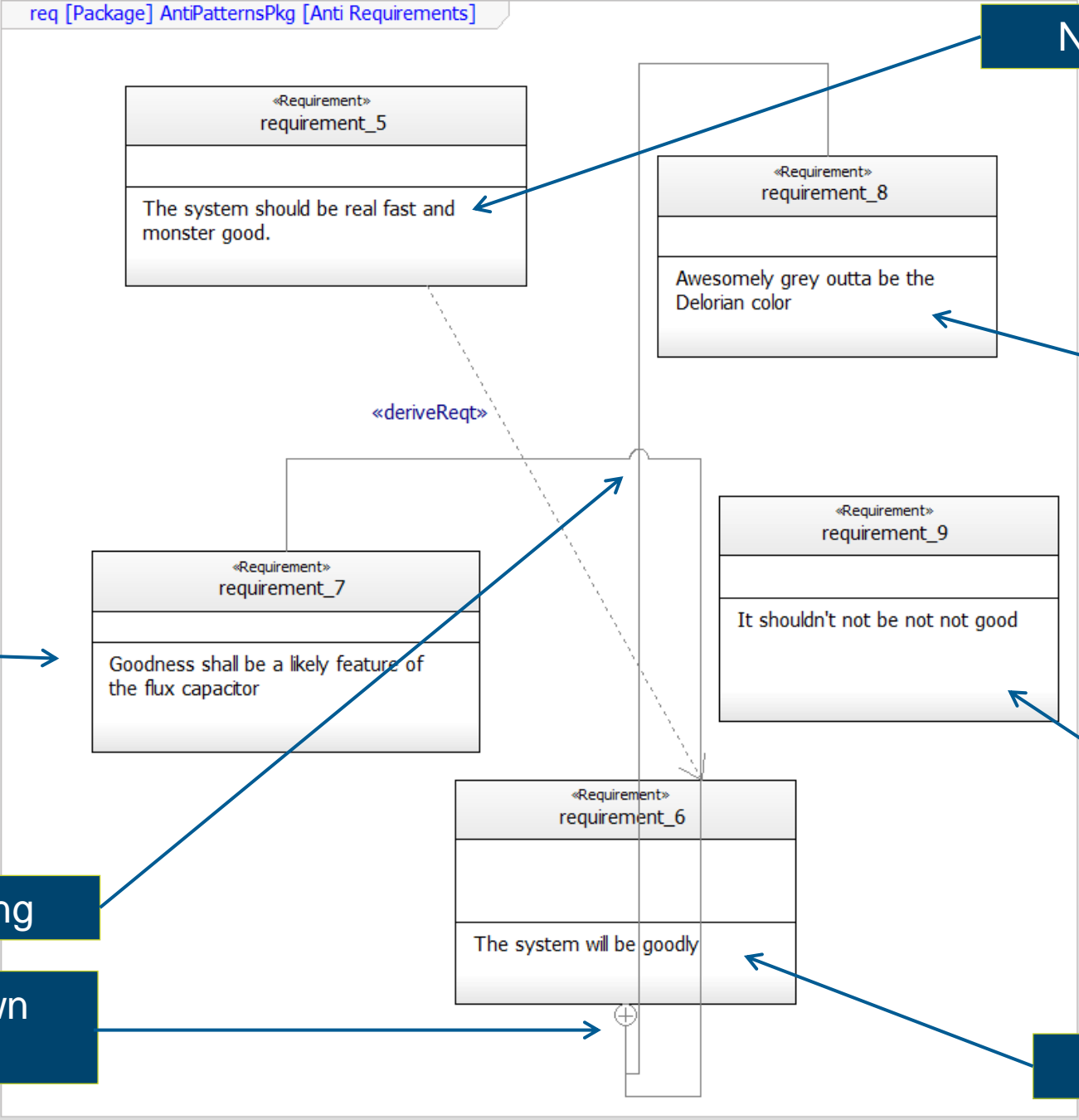
# Antipatterns

No mission diagram stated

Passive voice

Line crossing

Upside down relation



Not testable

Not in canonical form

Unclear

Not testable

# Requirements Table

Found 186 elements

ID	Name	Specification
ACES_SS_requirement_32		Any subsystem running software shall - both at start up and upon command - run an integrity check of the installed software object code verified by a method at least as robust as 32-bit CRC check
ACES_SS_requirement_33		Any subsystem running software that contains configuration data shall - both at start up and upon command - run an integrity check of the installed configuration verified by a method at least as robust as 32-bit CRC check as well as reasonable
ACES_SS_requirement_34		All subsystems other than the ACES_Management subsystem shall report error status and BIT results upon query or upon completion of tests.
ACSCUNT_requirement_10		The accuracy of movement of the control surface shall be +/- 0.5 degrees angle of +/- 0.5 cm distance.
ACSCUNT_requirement_11		Each control surface shall measure achieved control position with an accuracy of +/- 0.05 degrees or +/- 0.05 cm
ACSCUNT_requirement_12		If achieved position of any control surface unit is out of specification or takes longer than 3.0s, the control surface unit shall inform ACES_Management of the error
ACSCUNT_requirement_13		Each control surface shall accept a command for its position and will respond with both current commanded position and current measured position.
ACSCUNT_requirement_16		The ACES_Management subsystem shall check that each commanded movement takes place within 3.0seconds.
ACSCUNT_requirement_17		The ACES_Management subsystem shall check that each angular movement of less than 10 degrees is performed in less than 1.0 seconds.
ACSCUNT_requirement_18		Each control surface subsystem shall report movement completion to the ACES_Management subsystem with acquired measured position and time required for the movement.
ACSCUNT_requirement_19		The ACES_Management subsystem shall listen for life ticks from each surface control subsystem interface, expecting them to arrive at least every 0.5s.
ACSCUNT_requirement_20		If the ACES_Management subsystem does not receive a life tick within 0.5s of the initiating life tick, it shall report an error to both the Pilot Display and Attitude Management systems.
ACSCUNT_requirement_21		Each control surface input shall issue a life tick message to the ACES_Management subsystem at least every 0.5s.
ACSCUNT_requirement_24		Each control surface unit instance shall have a unique identifier which shall be used to in messages to the ACES_Management subsystem.
ACSCUNT_requirement_25		Each control surface unit shall have, as persistent configuration data, low and high movement limits, required measurement accuracy, and movement time limits.
ACSCUNT_requirement_26		Each surface control unit instance shall report an error to the ACES_Management subsystem if the result of a commanded movement is out of specification either in accuracy or timing.
ACSCUNT_requirement_3		All control surfaces shall accept commands from the ACES_Management subsystem to set rotational position.
ACSCUNT_requirement_7		Each control surface shall accept a command to move it to the desired position and shall begin movement based on that command within 0.1 seconds.
AM_requirement_1		The ACES_Management system shall command each control surface position either as a response to a received command or during built in test.
AM_requirement_27		The ACES_Management subsystem shall issue an error message to the Attitude Management system if both incoming and outgoing hydraulic pressures are not within +/- 1000 kPa of the default pressure of 35000 kPa and if this is not true
AM_requirement_28		The ACES_Management subsystem shall check hydraulic pressure at least once every 2.0 seconds.
AM_requirement_29		The ACES_Management subsystem shall issue an error message to the Attitude Management subsystem if incoming or internal power for fluctuations of more than 5% in voltage.
AM_requirement_30		The ACES_Management subsystem shall issue an error to the Attitude Control System within 0.5s if it detects a sudden power loss.
AM_requirement_35		The ACES_Management subsystem shall request a built in test run by every subsystem that contains software.
AM_requirement_4		The ACES_Management subsystem shall range check each movement command for each control surface movement to ensure that the set position is in range.
AM_requirement_6		If a movement position is out of range for the a specified control surface, the ACES_Management subsystem shall reject all positions specified within the incoming command and respond with a message indicating its rejection.
AM_requirement_9		The setting precision of the ACES_Management subsystem for control surface position shall be +/- 0.1 degree of angle or +/- 0.1 cm distance
ConfigReq_0		While in Maintenance Mode, the Maintainer shall be able to independently configure each control surface by setting actuator movement acceleration rates and ranges within the limits of the physical devices.
ConfigReq_1		While in Maintenance Mode, the Maintainer shall be able to independently command the full range of movement for each control surface and receive achieved position and time-to-complete each commanded movement.
ConfigReq_2		The Maintainer shall be able to download the set of configuration values to the maintenance device.

# Requirement Diagram Modeling: Key Takeaways



- **Requirement** is a SysML element that contains a textual specification
- Good requirements are understandable, unambiguous, precise, correct, complete, consistent, achievable, and testable
- Requirements may be added to any SysML diagram
- Requirements connect to other requirements with a **deriveReq** or **Contain** relation
- Requirements connect to other model elements with a variety of relations
  - **Trace**
  - **Satisfy**
  - **Allocate**
  - **Verify**
- The relations from elements other than requirements are almost universally **from** the other model element **to** the requirement





# Requirement Checklist

- Is each requirement
  - ☐ Stated in its canonical form?
  - ☐ A single statement of functionality or quality of services of a function?
  - ☐ Testable?
  - ☐ Unambiguous?
  - ☐ Correct?
  - ☐ Adequately specific for the need?
  - ☐ Specifies *need* (what) and not *design* (how)?
  - ☐ Compliant with the relevant standards?
  - ☐ Follows requirements naming standard?
  - ☐ Follows requirements ID standard followed?
- Is each functional requirement fully qualified in terms of
  - ☐ Performance
  - ☐ Precision
  - ☐ Accuracy
  - ☐ Latency
  - ☐ Availability
  - ☐ Safety
  - ☐ Reliability
- Is the set of requirements
  - ☐ Complete?
  - ☐ Consistent?
  - ☐ Covering important exception, performance, and edge cases?

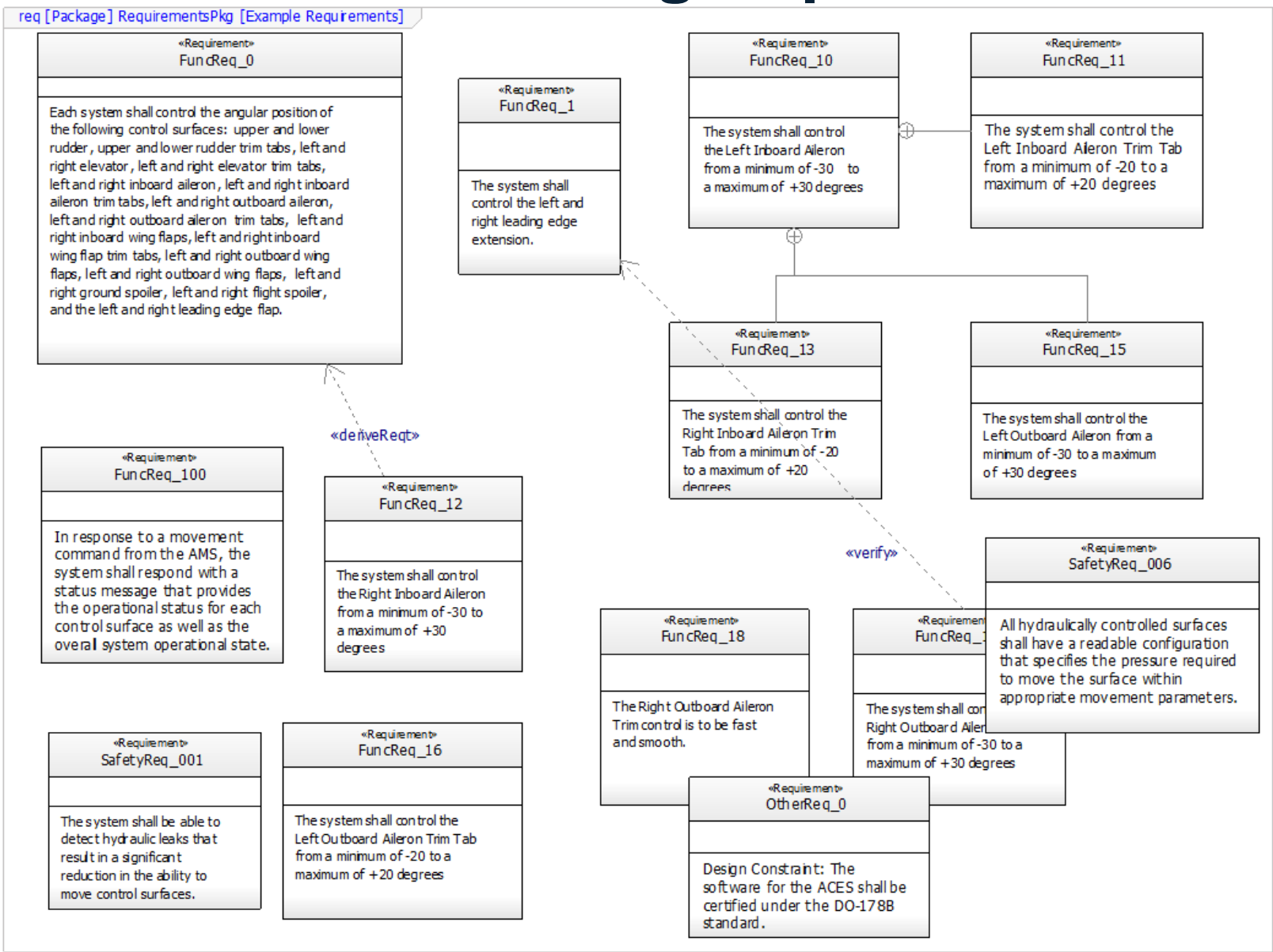


# Requirement Diagram Checklist

- ☐ Are the requirements predominately more abstract and broader at the top and left with more detailed requirements lower or more to the right?
- ☐ Is the purpose and scope of the diagram stated on the diagram?
- ☐ Are the relevant properties for all represented requirements visible?
- ☐ Are all relevant model elements (including but not limited to requirements) to achieve the mission of the diagram shown?
- ☐ Are all relevant relations shown?
- ☐ Does the diagram hide model elements and properties not relevant to the diagram mission?
- ☐ Is line crossing avoided?
- ☐ Do all the requirements individually meet the requirements check list?
- ☐ Is the diagram appropriately named?
- ☐ Is the diagram appropriately scoped?
- ☐ Is the diagram appropriately located within the model?



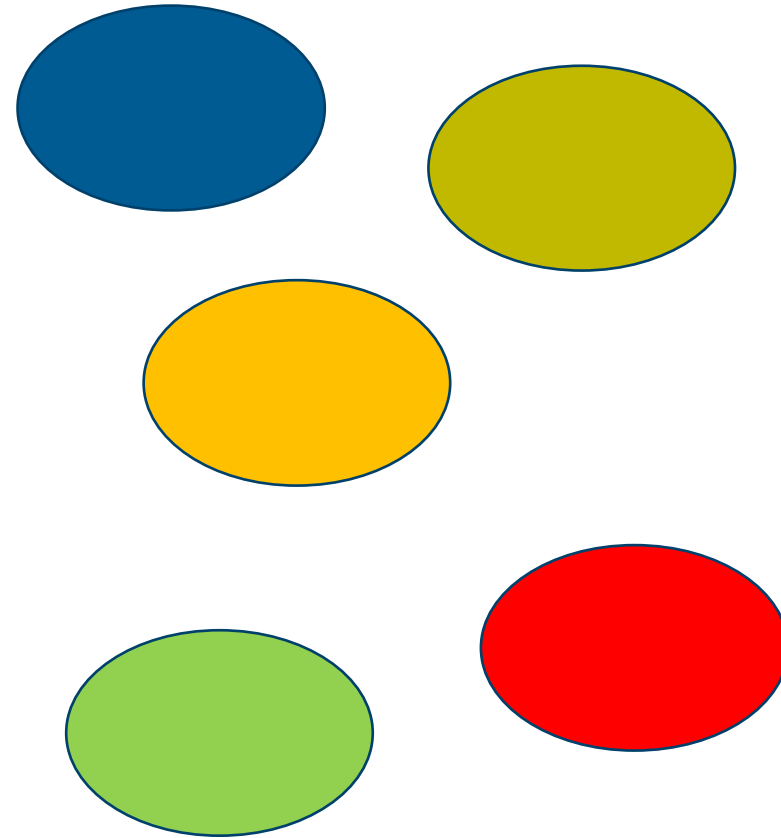
# Exercise: Evaluate the following requirement set & diagram



[Click to see answer](#)

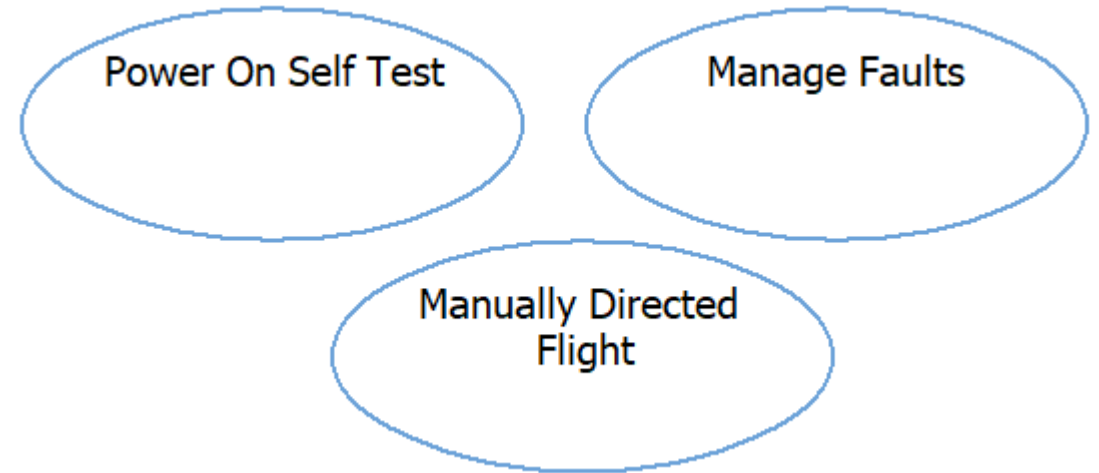


# Use Case Diagram



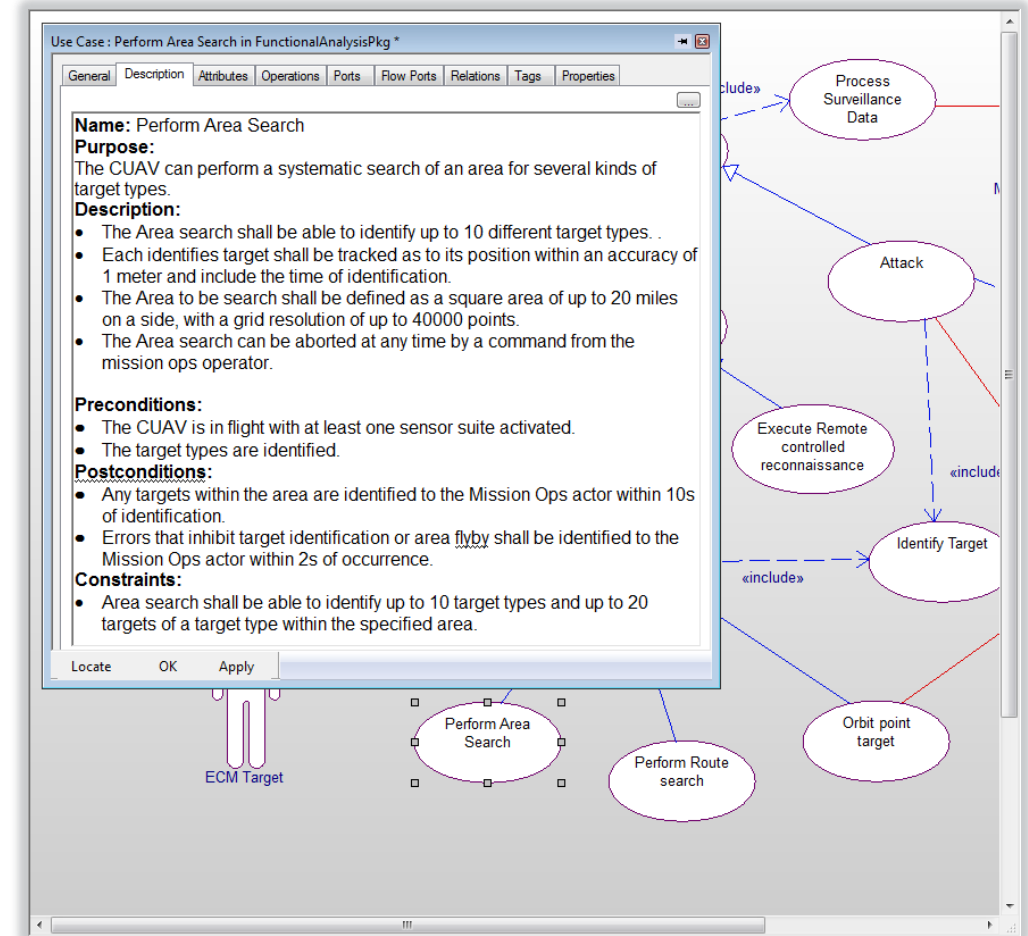
# Use Cases

- A **Use Case** specifies *a set of actions performed by its subjects*, which yields an observable result that is of value for one or more actors or other stakeholders of each subject.
- A use case
  - Is a set of actions that collectively define an operational capability of a system from an external, stakeholder perspective
  - Is set of interactions between a system and a set of actors around a singular use of a system
  - Traces to a set of requirements
  - Returns a result visible to one or more actors
  - Does not reveal or imply internal design or implementation
  - Is independent of other use cases



# More on Use Cases

- A use case by itself does not define behavior; it may be *detailed* with behavioral views such as sequence, activity, and/or state diagrams
- All use cases should have a description. Recommended form:
  - Name
  - Purpose
  - Explanation
  - Preconditions
  - Post-conditions
  - Constraints/Invariants/Assumptions
- Remember all model elements should have a useful and meaningful description



# Use Case Relations

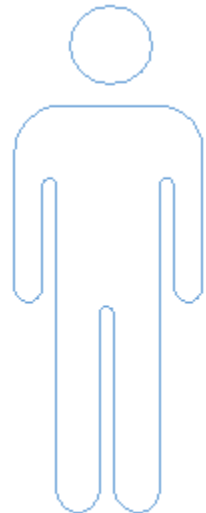
- Association
  - Shows the actors with which use cases collaborate
  - Note: NEVER show associations between use cases
- Generalization
  - Arrow points to the more general use case. Indicates the more specialized use case includes ALL in the more general plus some specializations and extensions
- Include
  - A kind of “whole-part” relation with the arrow pointing to the included (part) use case
  - Enables reuse of a common capability in other larger use cases
- Extend
  - Used when a smaller use case just provides additions to a larger use case

# Actor

- An **Actor** is an element outside the system of concern with which the system has meaningful interactions
- An actor
  - Is a Block (more on blocks later)
  - May be a human user, a connected computing device, or an element sensed, manipulated, or in some way interacted with by the system



GPS\_Satellite



GPS\_Satellite



Charging\_System



Bird\_Watcher

# Use Case Diagram

System boundary (optional)

Actor

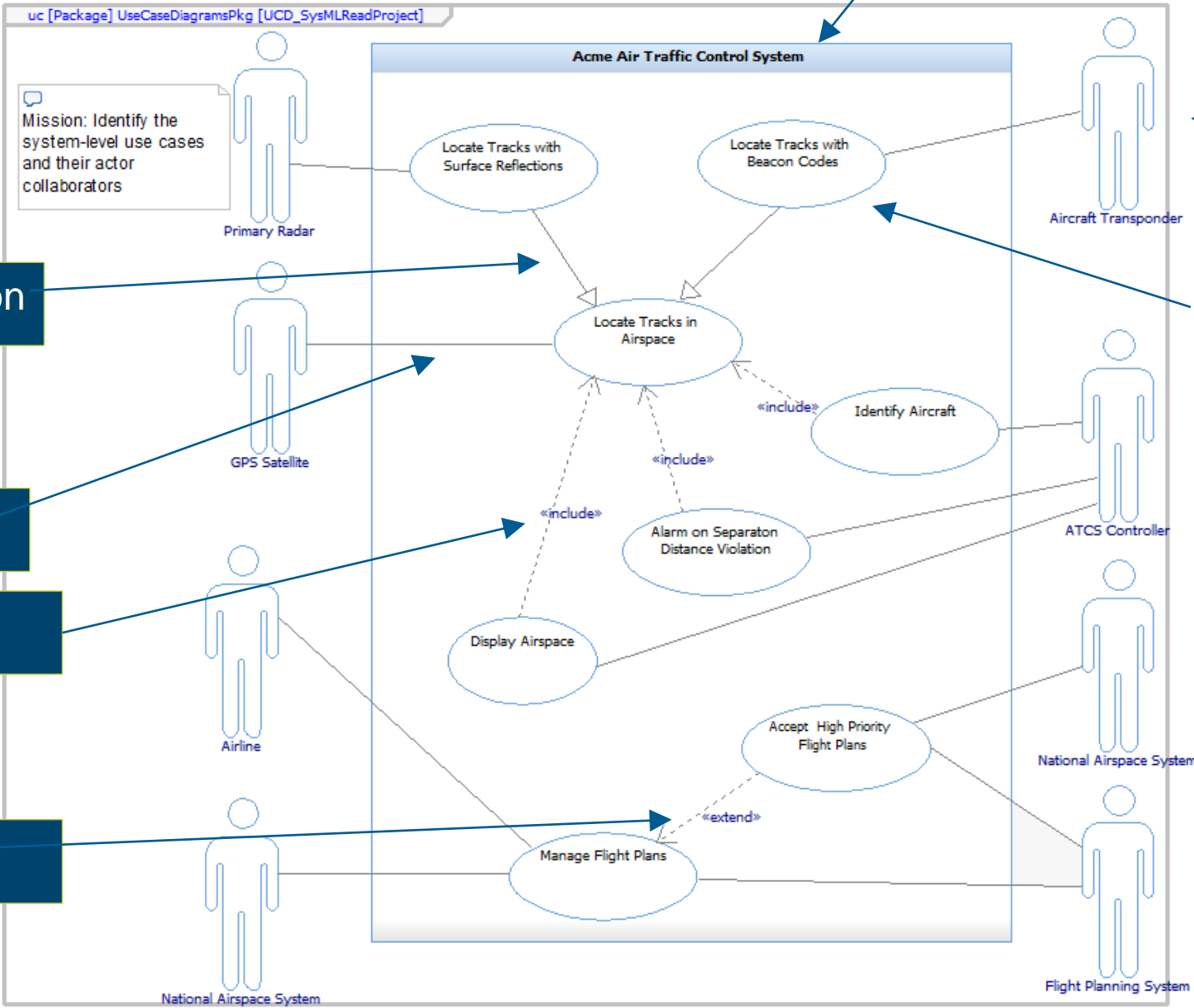
Use case

Generalization

Association

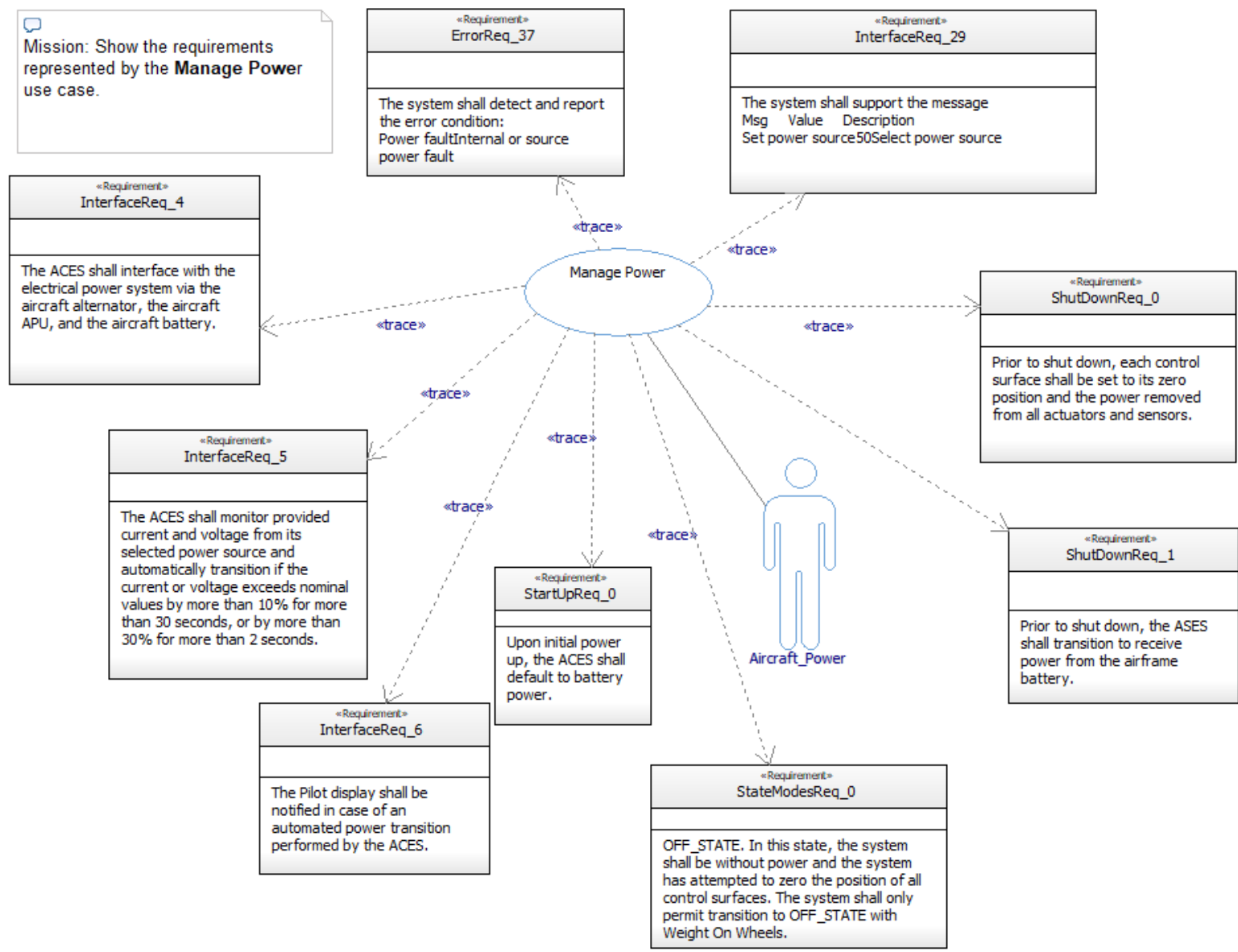
Include

Extend



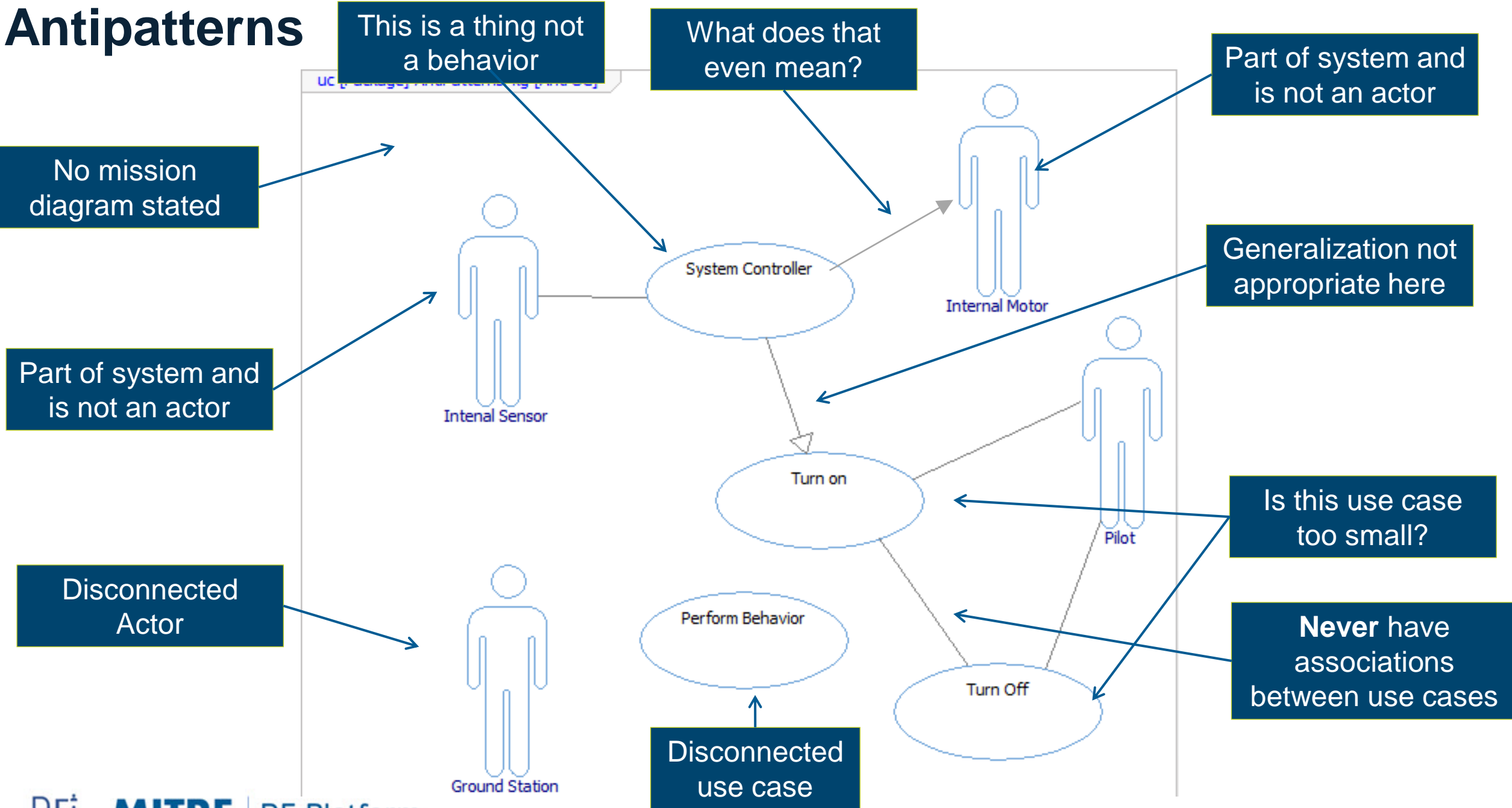
A very common problem with use case diagrams is that people add many relations among them “for completeness” but these additions do not help achieve the use case modeling objectives

# Another use case diagram with a different mission





# Antipatterns



# Use Case Modeling: Key Takeaways



- **Use Case** represents
  - A collection of requirements around a common system usage
  - A set of scenarios around a common usage
  - A set of system functions used together
- **Actors** are objects outside the scope of the system of concern with which the system has important interactions
  - Use cases related to actors via **associations**
- Use cases can relate to other use cases in several ways
  - **Generalization**
  - **Include**
  - **Extend**
- Use cases should never have associations to other use cases



# Use Case Checklist

- Each use case

- ☐ Follows the naming convention
- ☐ The name is a verb or verb phrase
- ☐ Has a meaningful description, including
  - ☐ Purpose
  - ☐ Pre-conditions
  - ☐ Post-conditions
- ☐ Associates to at least one actor
- ☐ Identifies externally-visible behavior (only)
- ☐ Traces to a coherent set of requirements
- ☐ Traced requirements include
  - ☐ Error / exception / rainy-day cases
  - ☐ Performance
  - ☐ Precision / accuracy / fidelity
  - ☐ Safety
  - ☐ Reliability
  - ☐ Security
- ☐ Is independent (in terms of the requirements) from all other use cases

- ☐ Is detailed with
  - ☐ More than one sequence diagram
  - ☐ An activity and/or state diagram
  - ☐ Does this detail avoid internal design?
  - ☐ Does this detailed model execute/simulate?
  - ☐ Is this detail linked to specific requirements?

- Things to avoid. The use case

- ☐ Does not have associations to other use cases
- ☐ Avoids indicating designs (often indicated by having too many relations among use cases)
- ☐ Is not too small - can be reasonably expected to have multiple requirements and multiple scenarios
- ☐ Is not too large – can be reasonably expected to have fewer than 100 requirements or 50 independent scenarios
- ☐ Is too dependent on other use cases

- For the set of use cases

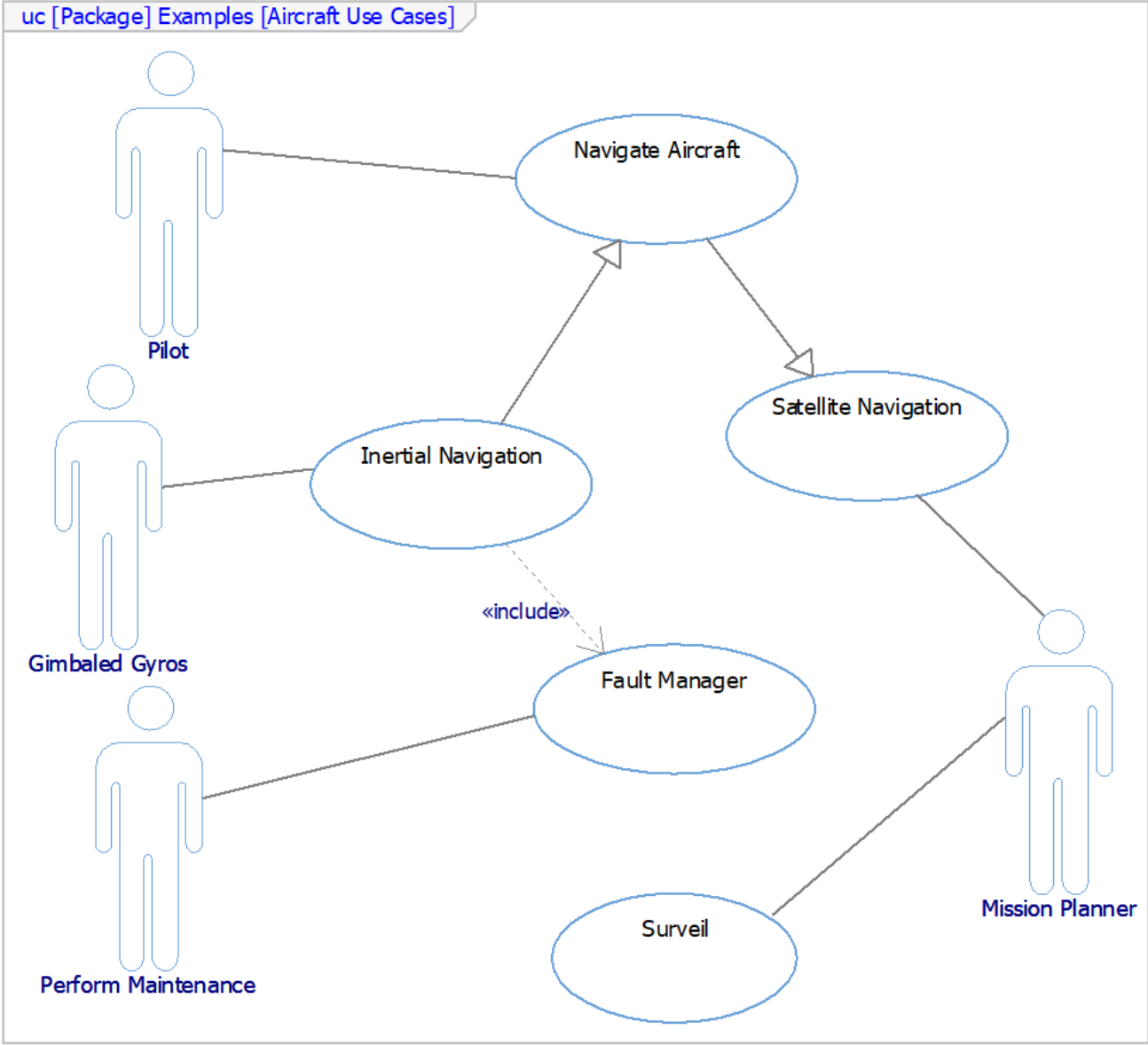
- ☐ All functional and quality of service
- ☐ All relations make sense
- ☐ Not overly burdened with “extra” relations that don’t add value



# Use Case Diagram Checklist

- ☐ Is the purpose and scope of the diagram stated on the diagram?
- ☐ Are the relevant properties for all represented requirements visible?
- ☐ Are all relevant model elements (including but not limited to use cases, actors, and requirements) to achieve the mission of the diagram shown?
- ☐ Are all relevant relations shown?
- ☐ Does the diagram hide model elements and properties not relevant to the diagram mission?
- ☐ Is line crossing avoided?
- ☐ If requirements are shown, do they all properly relate to the use case?
- ☐ Is the diagram appropriately named?
- ☐ Is the diagram appropriately scoped?
- ☐ Is the diagram appropriately located within the model?

# Exercise: Evaluate the following Use Case Diagrams



# Basic Structural Modeling in SysML

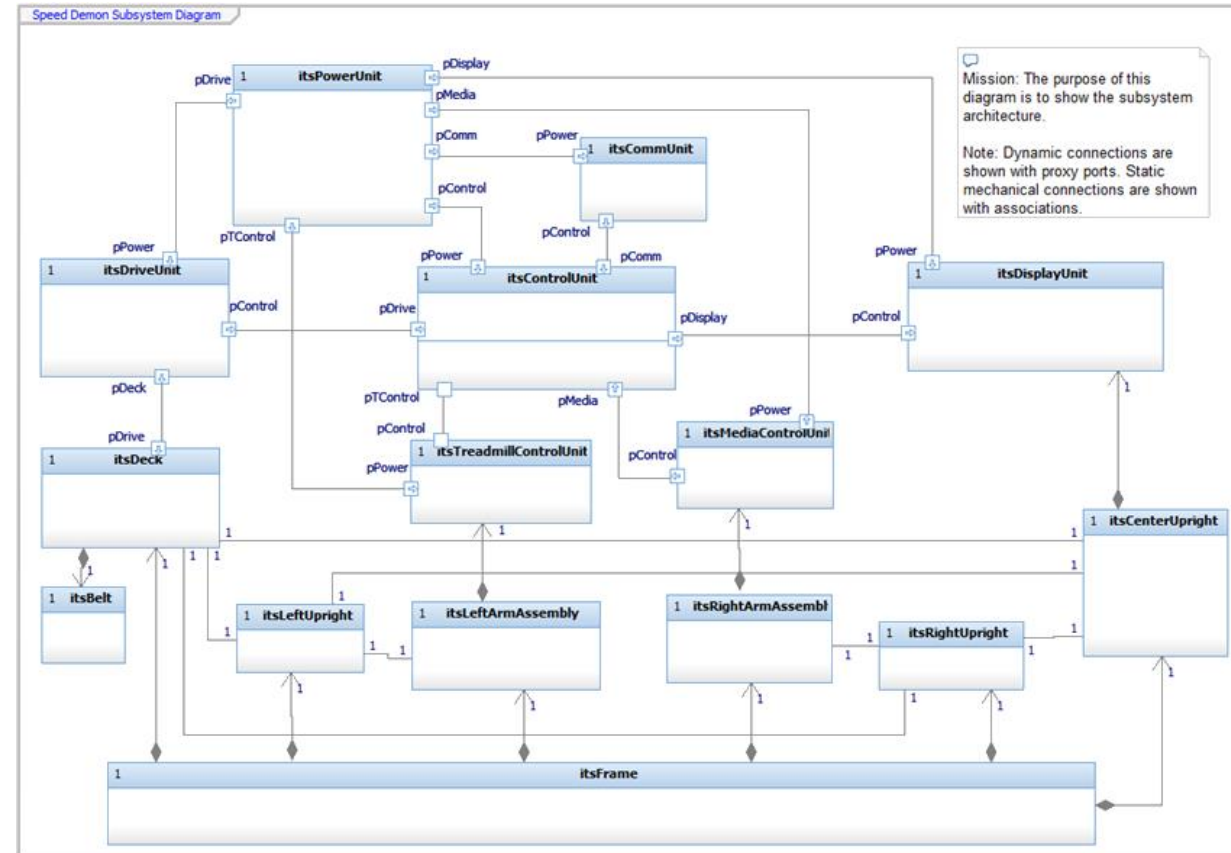
Blocks / Instances

Port / Interfaces

Block Definition Diagrams

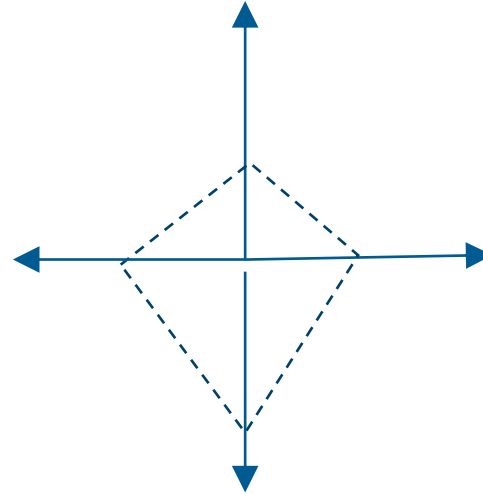
Internal Block Diagrams

Checklists



# SysML Structural Features

- **Language / Tool Extensions**
  - Profiles
  - Stereotypes
- **Design Structures**
  - Block, Actors, Events
  - Value Types, Dimensions, Units
  - Instances
  - Which have properties ...
    - Operations
    - Event receptions
    - Value Properties
    - Ports
- **Relations**
  - Association
    - Aggregation
    - Composition
  - Generalization
  - Dependency
  - Multiplicity
  - Connector
- **Model Structures**
  - Models
  - Packages
  - Diagrams
  - Tables
  - Matrices

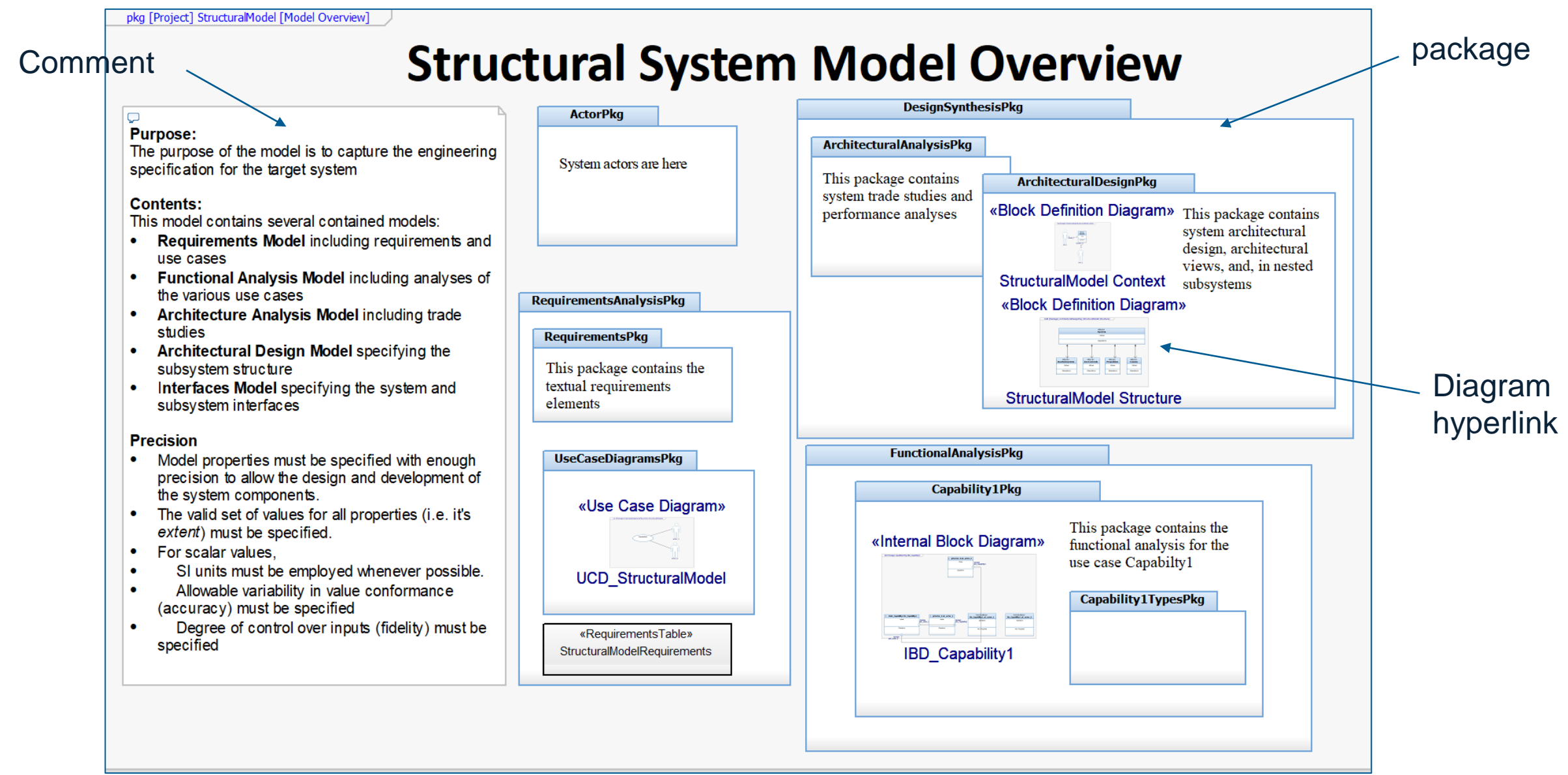


# Package diagram

- A *package* is a model element that contains other model elements
- Packages serve as the basic organizational unit for models
  - Models are divided up into different packages
  - Packages can contain any model element, including diagrams, tables, and matrices
- A *package diagram* shows a set of packages, possibly with some of their contained elements and relations
- It is considered good practice to have a package diagram “at the top” of the model to serve as an overview / table of contents for the model to assist in model comprehension and navigation
  - Such a diagram is often referred to as a *model overview diagram*
  - It is common for such a diagram to have hyperlinks to important diagrams and tables in the model

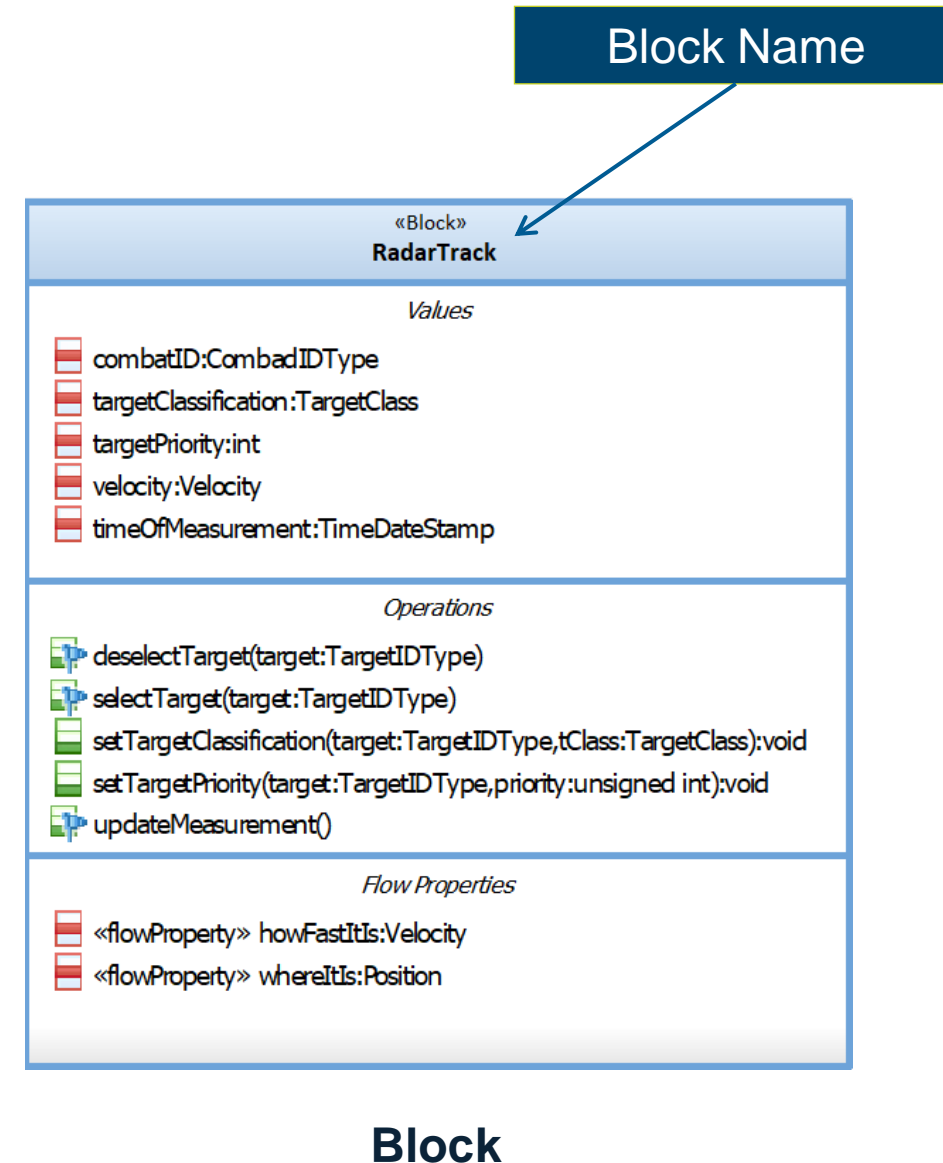


# Model Overview Diagram Example



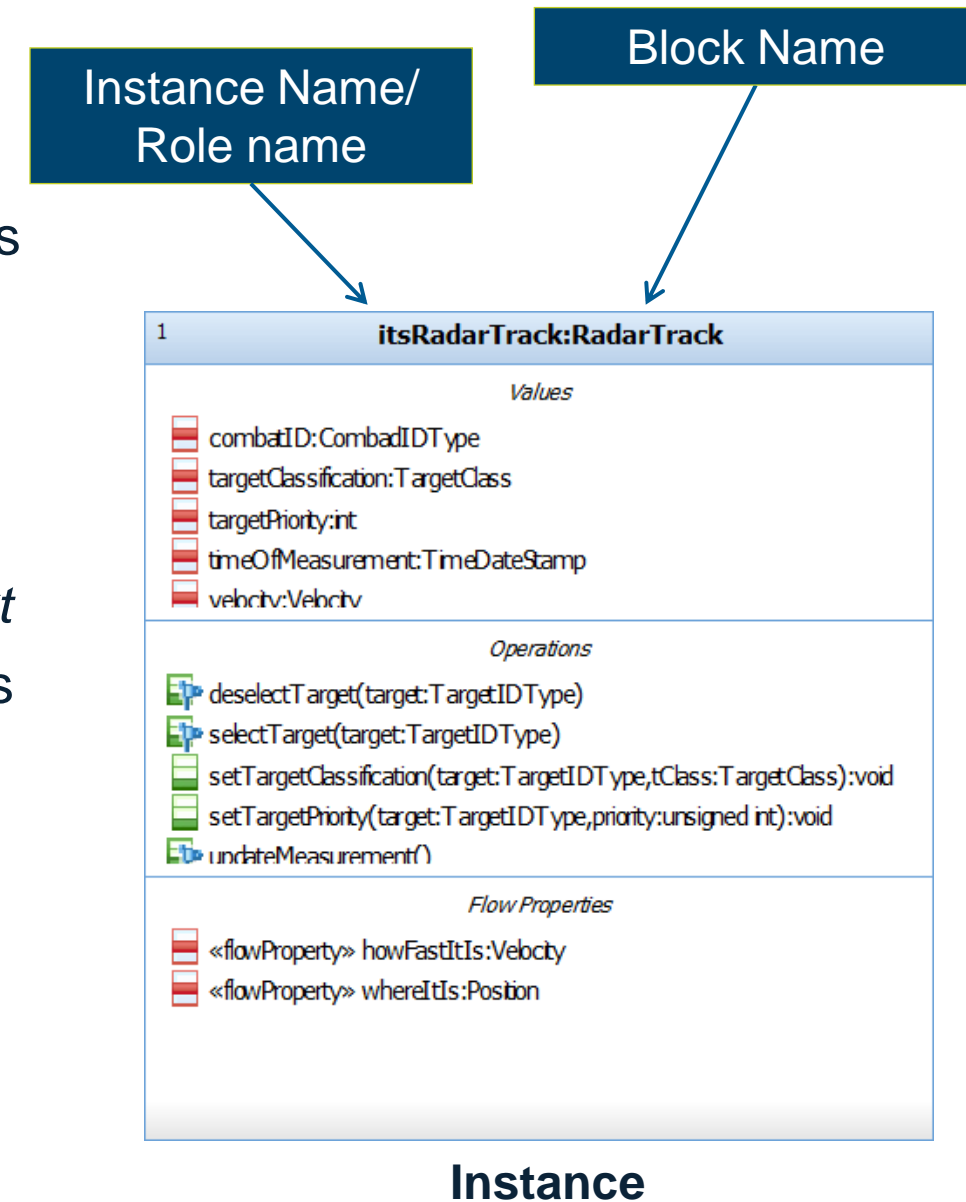
# Key structural Concepts: Block

- A *block* represents the “type” of entity
  - Consists of
    - information (value and flow properties) and
    - services that act on that information (operations & event receptions)
    - other features (e.g. ports)
  - As types, blocks exist in the specification / design activities (“design-time”)
  - Hint: Generally, blocks should always have a singular name
    - Collections are indicated with *multiplicity*
- Note: you can choose what features of a block or element you want to expose on a diagram



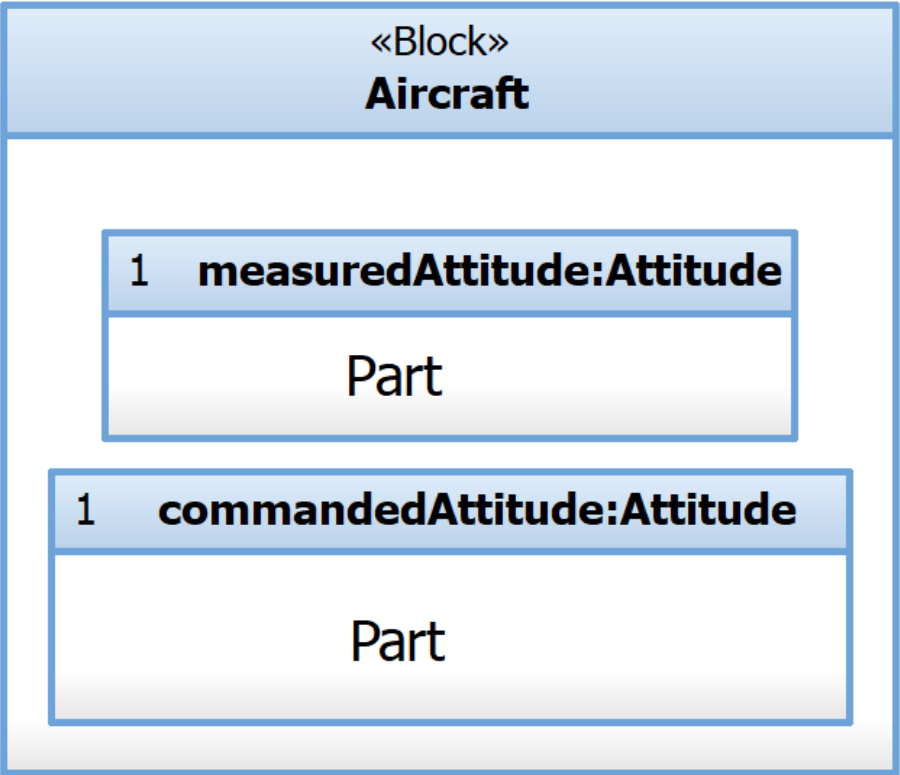
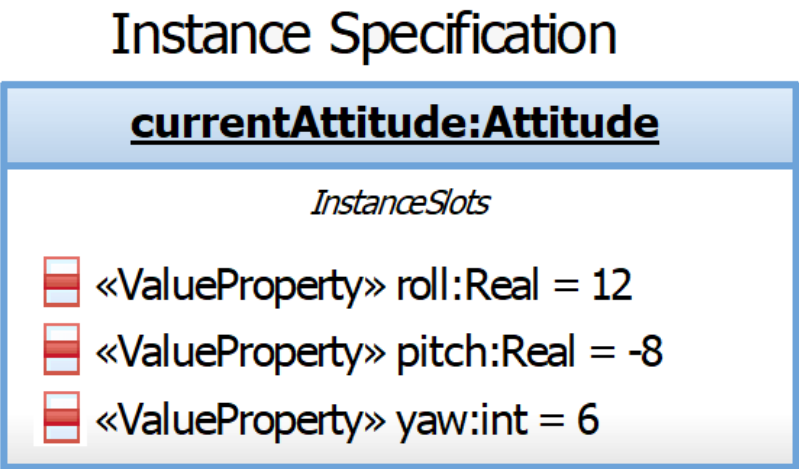
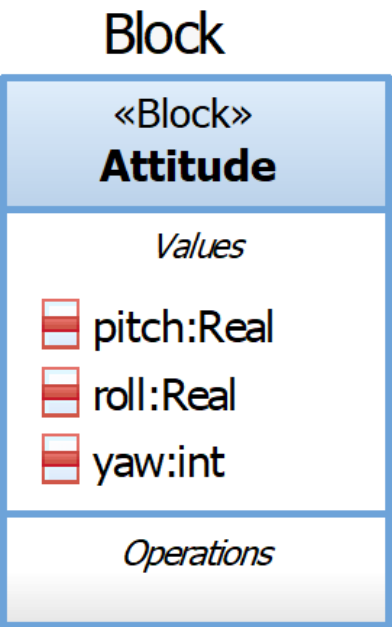
# Key Structural Concept: Instance and Part

- An **instance** is a run-time or real-world exemplar of its block specification; that is, the block *types* the instances
  - All instances of the same type have all the same properties but each of those properties may have a different value from other instances
- Instances are named <instance name>:<type name> as in
  - **upperRudder:Rudder**
- A **part** is a role an instance of a type plays in a specific context
- If blocks are related via **associations** then instances and parts are related via **connectors**
- Blocks can associate to other blocks, but not ports on blocks
  - Only ports on instances or parts can be connected
- An internal block diagram with instances should always be thought of as a snapshot of a running or operational system



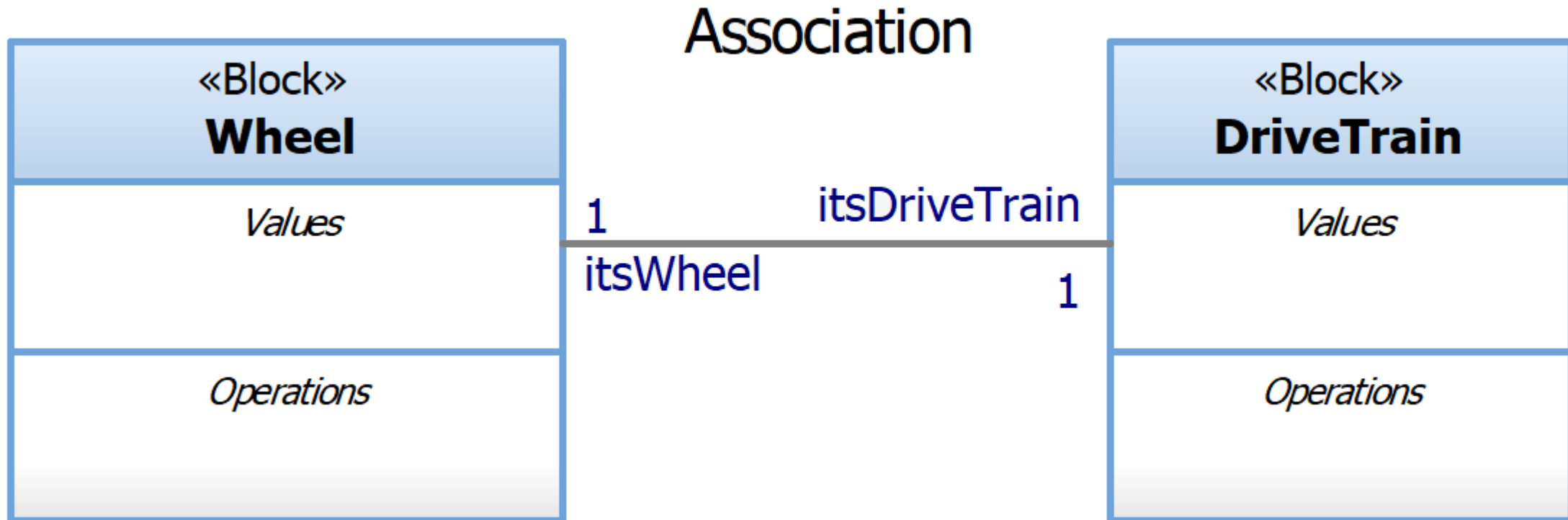
# Block vs Part vs Instance

- A **block** is a compound type
- An **instance** is an exemplar of a block with particular values in *instance value slots*
- A **part** is a role that an instance plays in a particular context



# Block Relations: Association

- Association
  - A relation between blocks that indicates that, at run-time, one instance can navigate to, and invoke services of, another
  - Show as a solid line with an optional open arrowhead, if unidirectional



# Block Relations: Aggregation and Composition

- Association

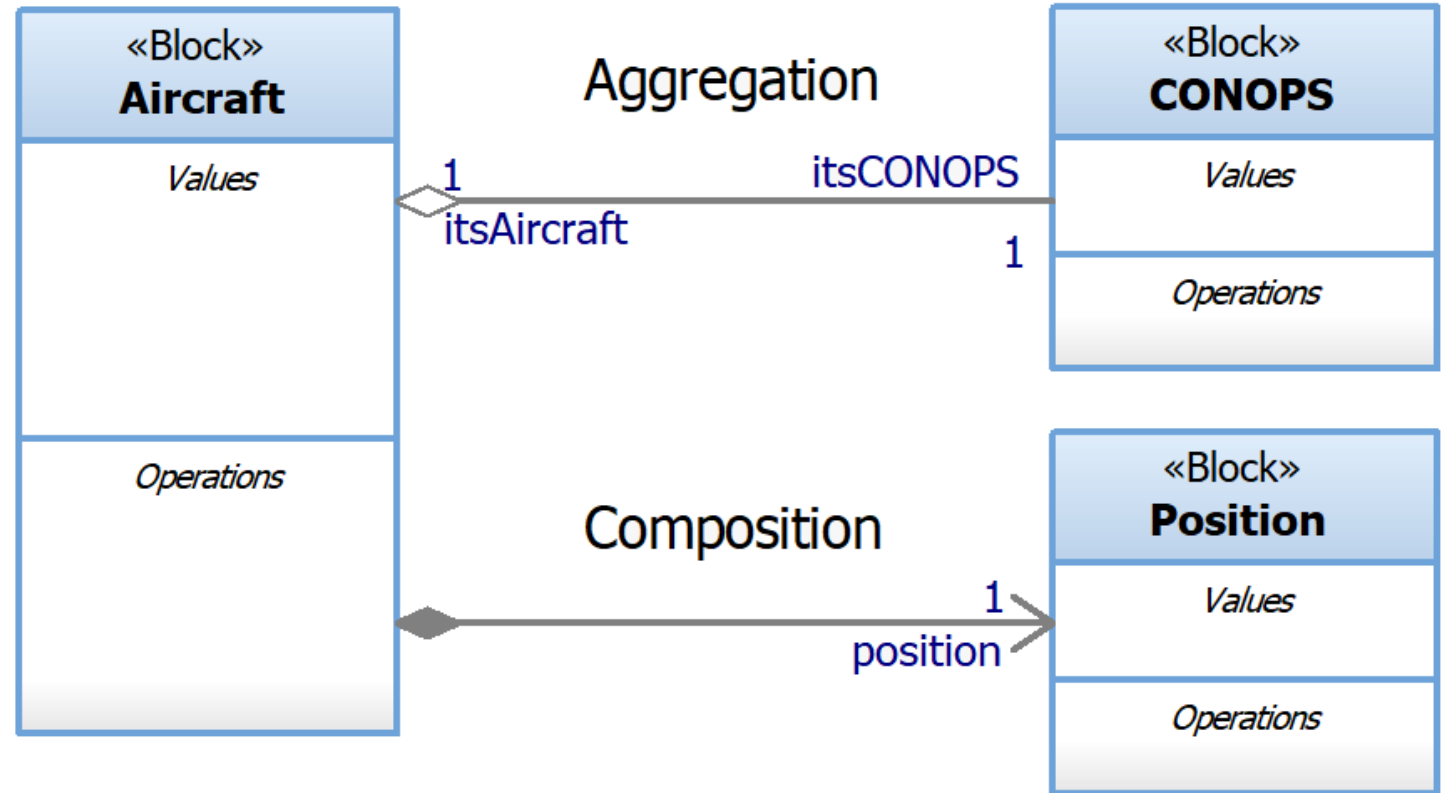
- Specialized forms

- Aggregation

- Weak form of whole-part (part instance can exist independently from the “whole”)
    - Shown as a solid line with an empty diamond at the owner end

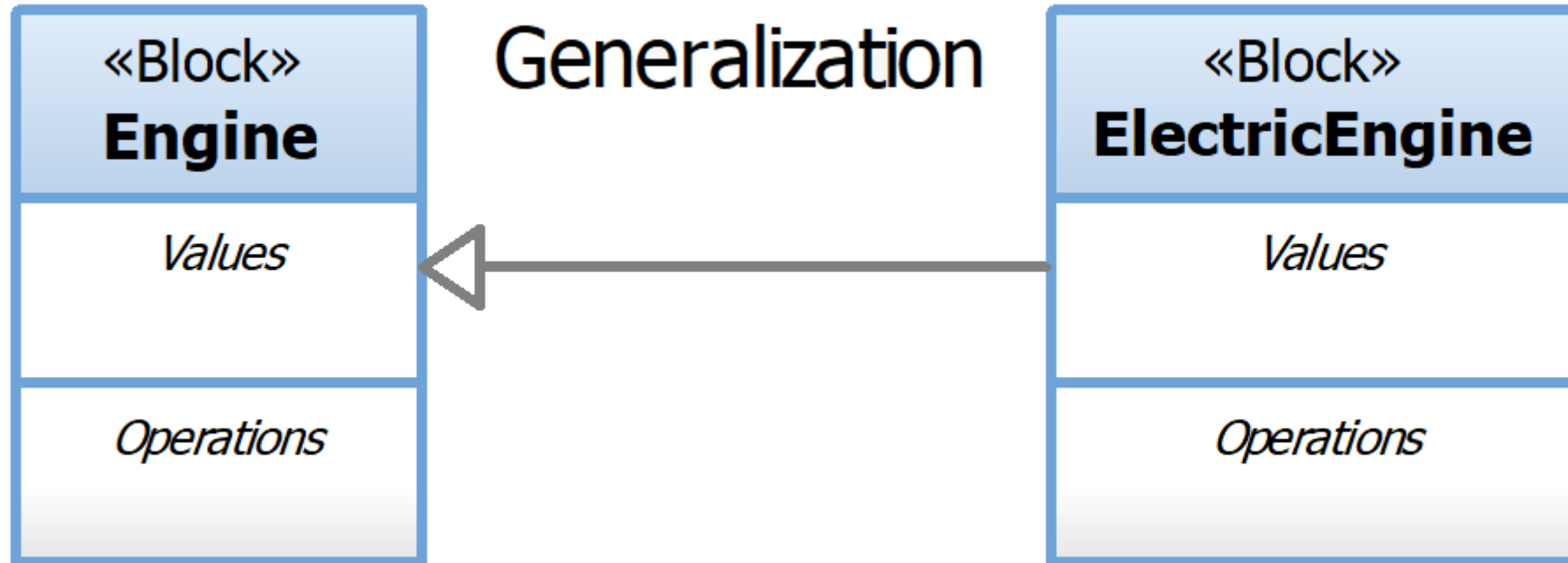
- Composition

- Strong form of whole-part (means that the part instance cannot exist independently from “whole”)
    - Shown as a solid line with a filled-in diamond at the owner end



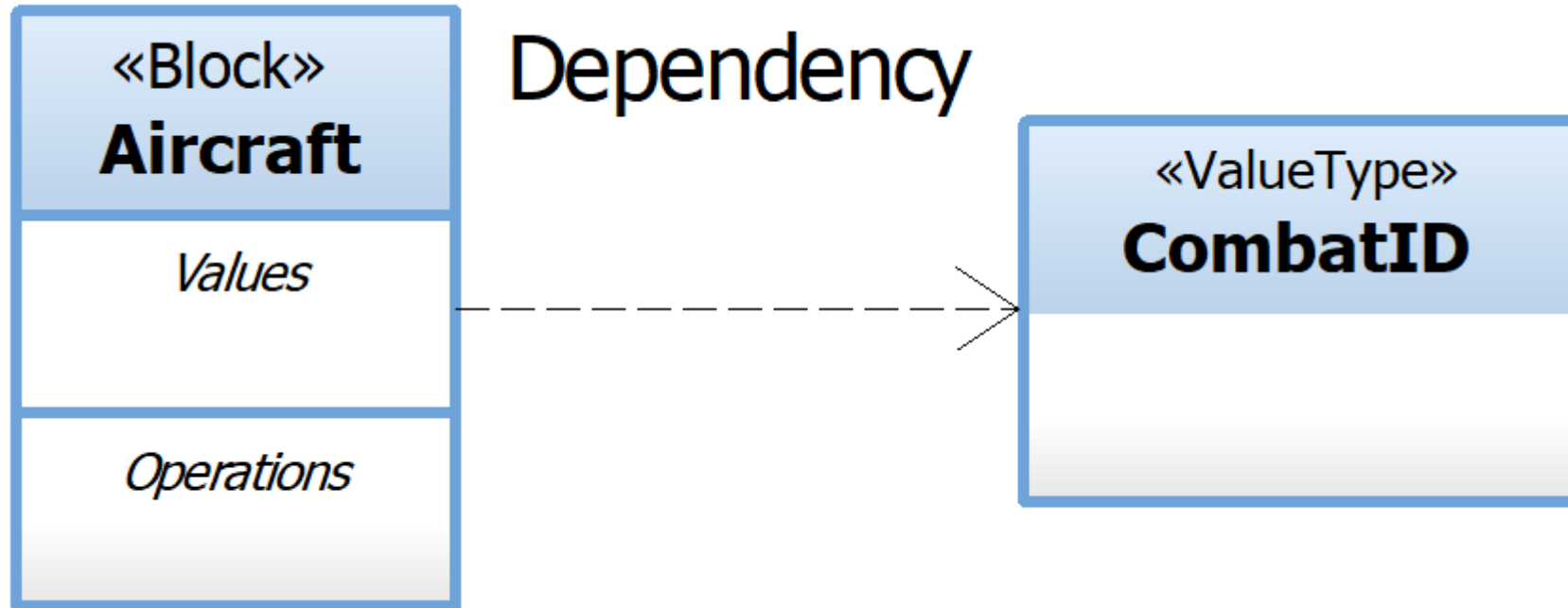
# Block Relations: Generalization

- Generalization
  - One “is-a-specialized-kind of” another
  - One block inherits the structure and behavior of another
  - Shown as a solid line with a closed arrowhead pointing to the more general element



# Block Relations: Dependency

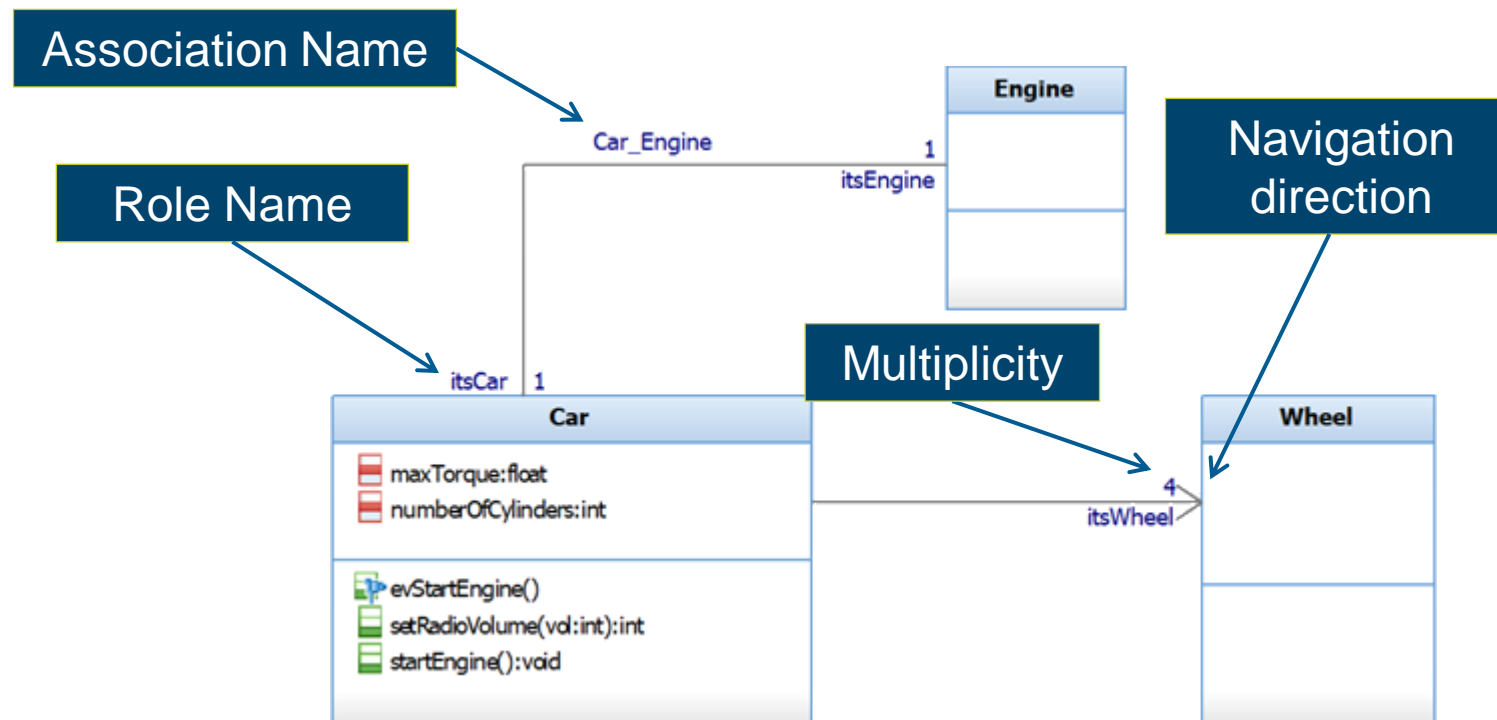
- Dependency
  - Some other (weak) kind of relation
  - Shown as a dashed line with an open arrowhead





# Association

- The *association* relationship allows instances of classes to collaborate by “sending messages” which may be
  - Synchronous invocation (operation call or synchronous event)
  - Asynchronous invocation (asynchronous event)
- Contains (optional) direction, multiplicity, association end role names
- Association names can be added but they add little value and are little used



# Putting Relations Together

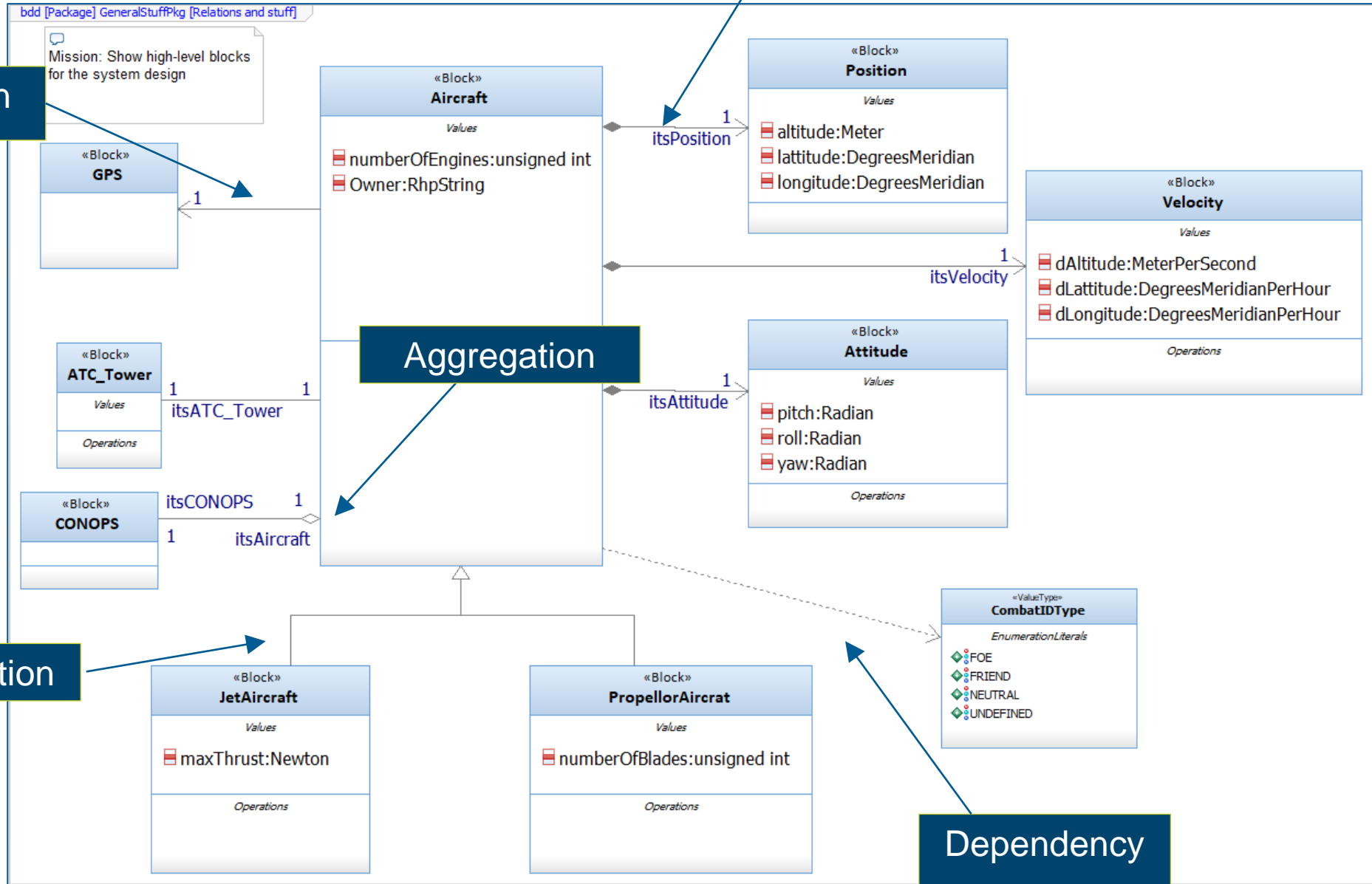
Composition

Association

Aggregation

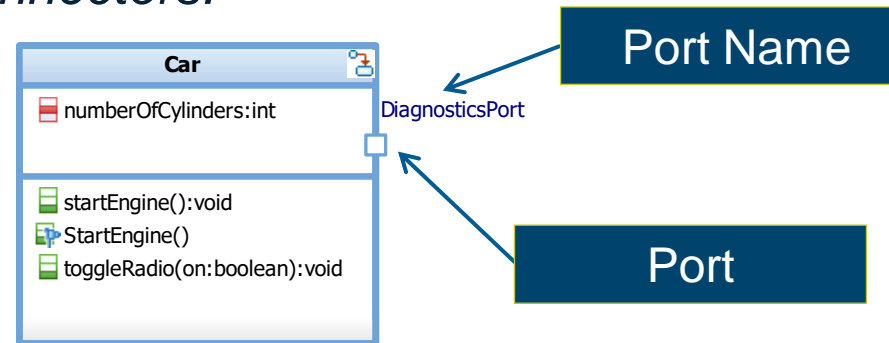
Generalization

Dependency

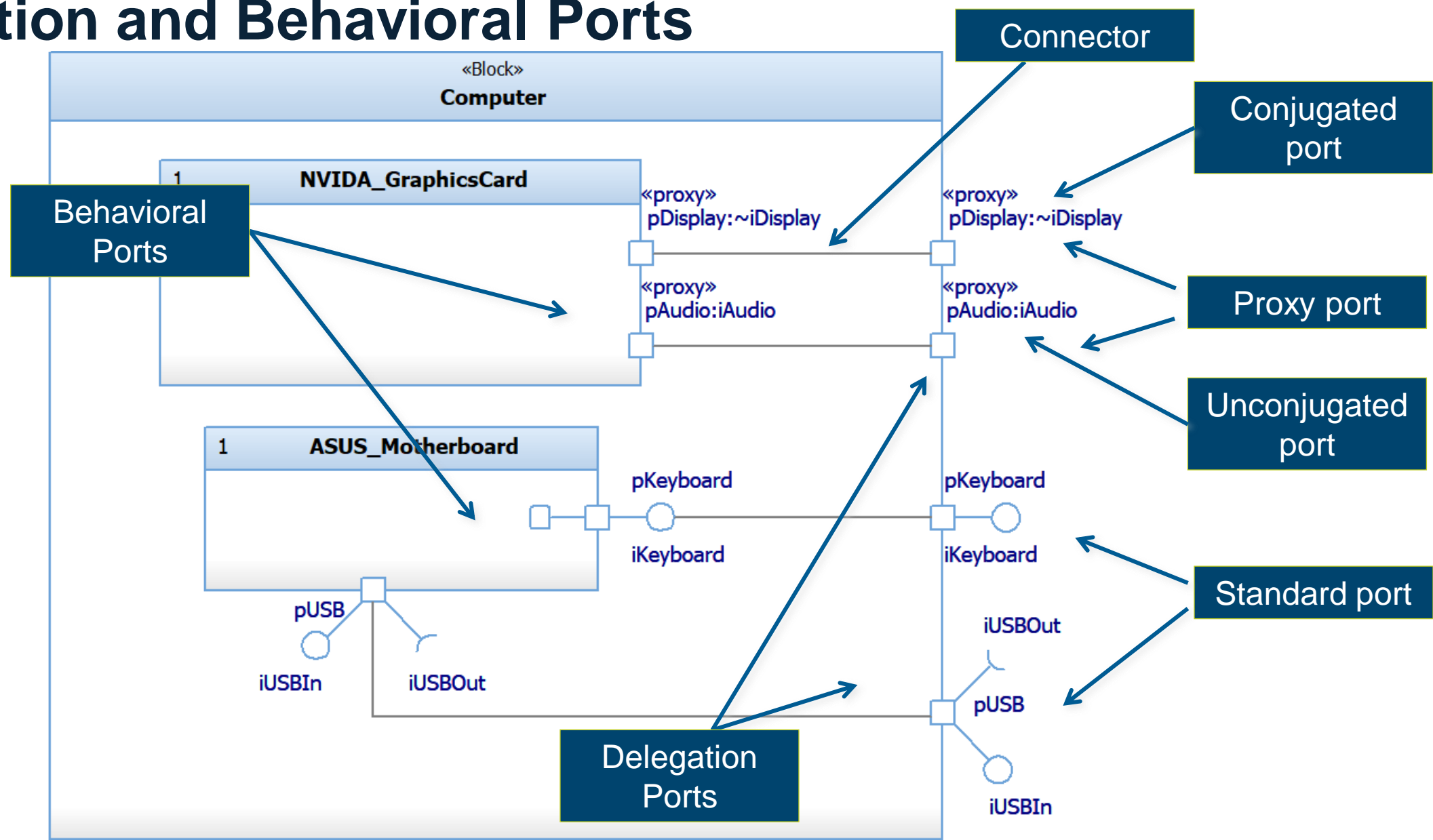


# Ports

- Ports are named connection points between objects
- **Ports are an alternative means to connect blocks to using associations. *Ports are optional.***
- SysML has multiple types of ports. The most common are
  - Standard ports (from UML) – typed by *Interfaces*
  - Proxy ports – typed by *Interface Blocks*
- Ports are *typed by interfaces* which define the messages that can be sent between the port pairs
- In MBE, ports are used to
  - Receive events from other parts of the system
  - Pass messages to lower-level parts
- Note: Ports on blocks *cannot be associated* to ports on other blocks; only ports on instances can be linked together. Ports are linked with *connectors*.



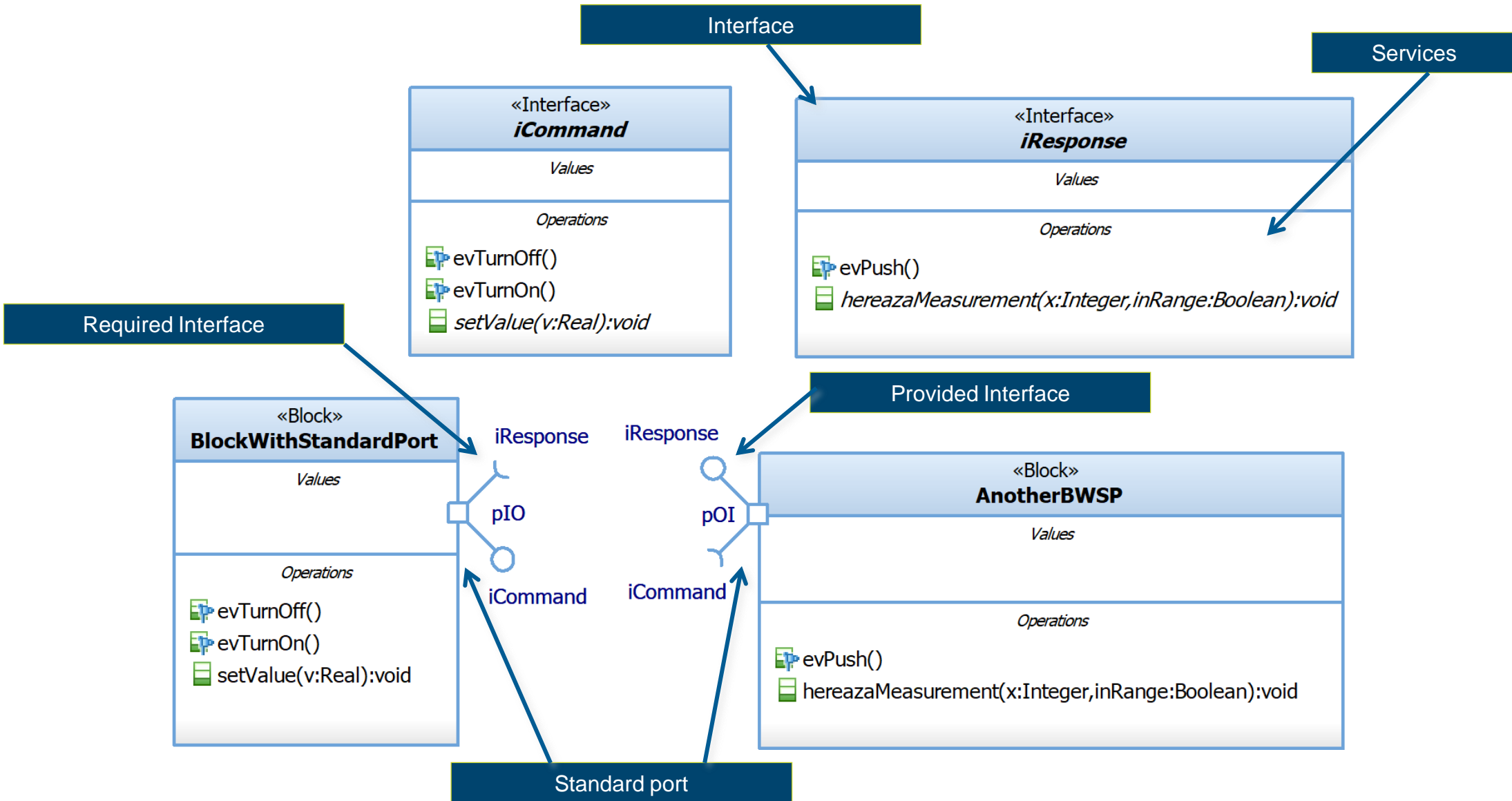
# Delegation and Behavioral Ports



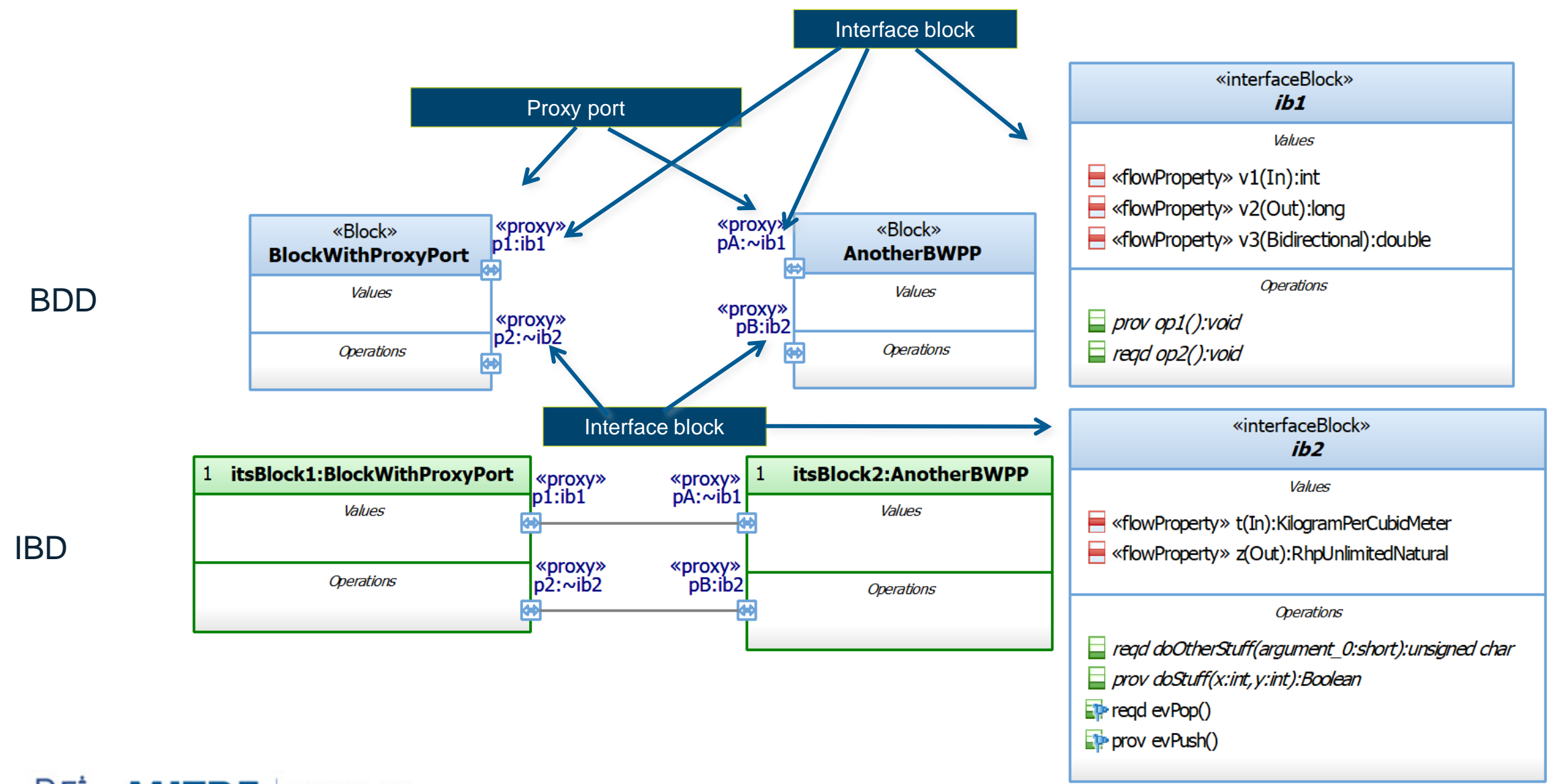
# SysML Has several different kinds of ports

- UML (Standard) ports are typed by interfaces, which can specify operations and event receptions but not value or flow properties, nor have implementation
  - Offered interfaces
  - Provided interfaces
- Flow ports are typed by the value types (such as Joule, Liter, or Real)
- Proxy ports are typed by Interface Blocks
  - One side must be *normal* (p:ib) and the other *conjugated* (p:~ib)
  - Tilde (~) indicates a *conjugated* interface
- Full ports are internal parts of a block, exposed to the block's clients

# SysML Has several different kinds of ports: Standard Ports

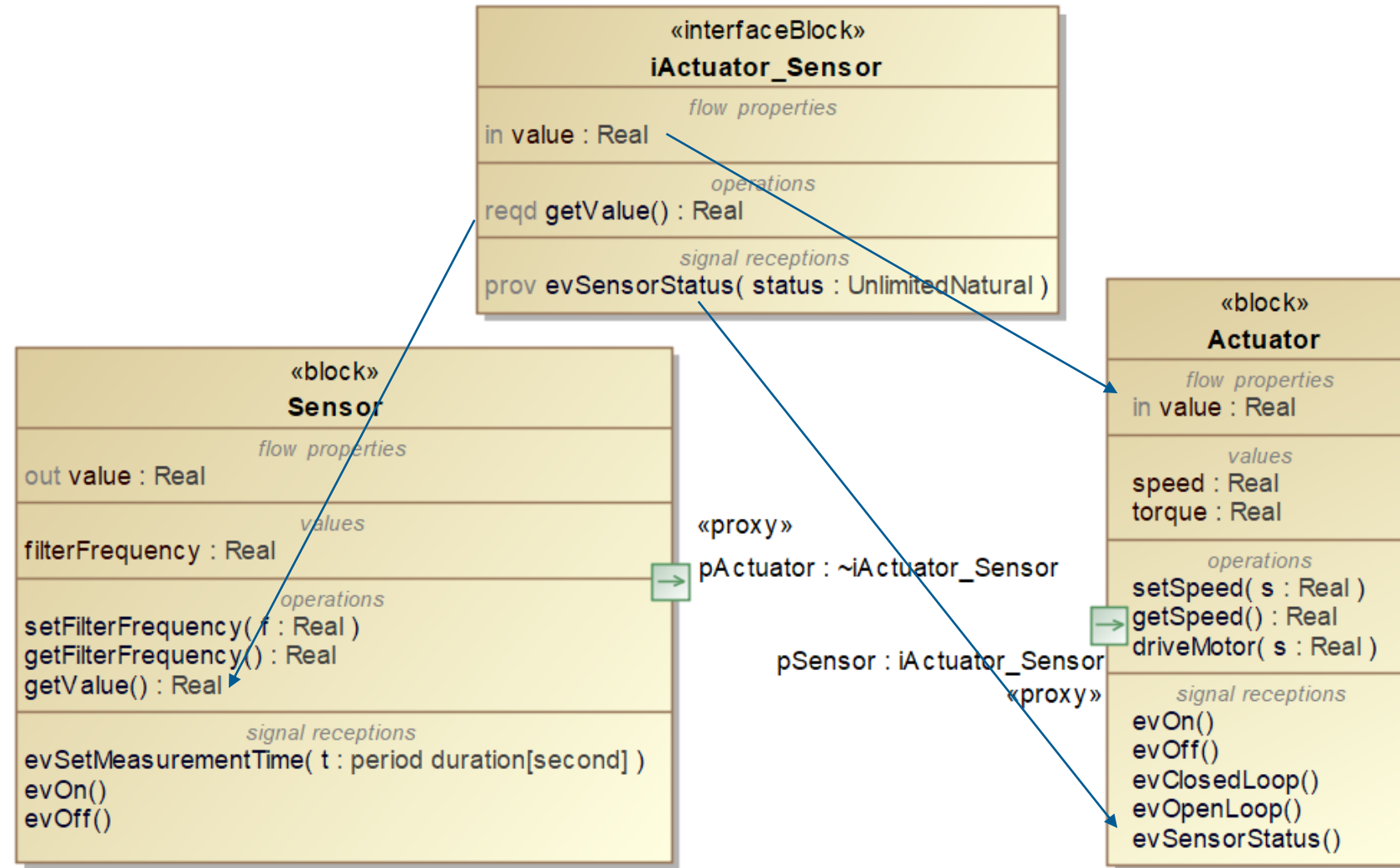


# SysML Has several different kinds of ports: Proxy Ports



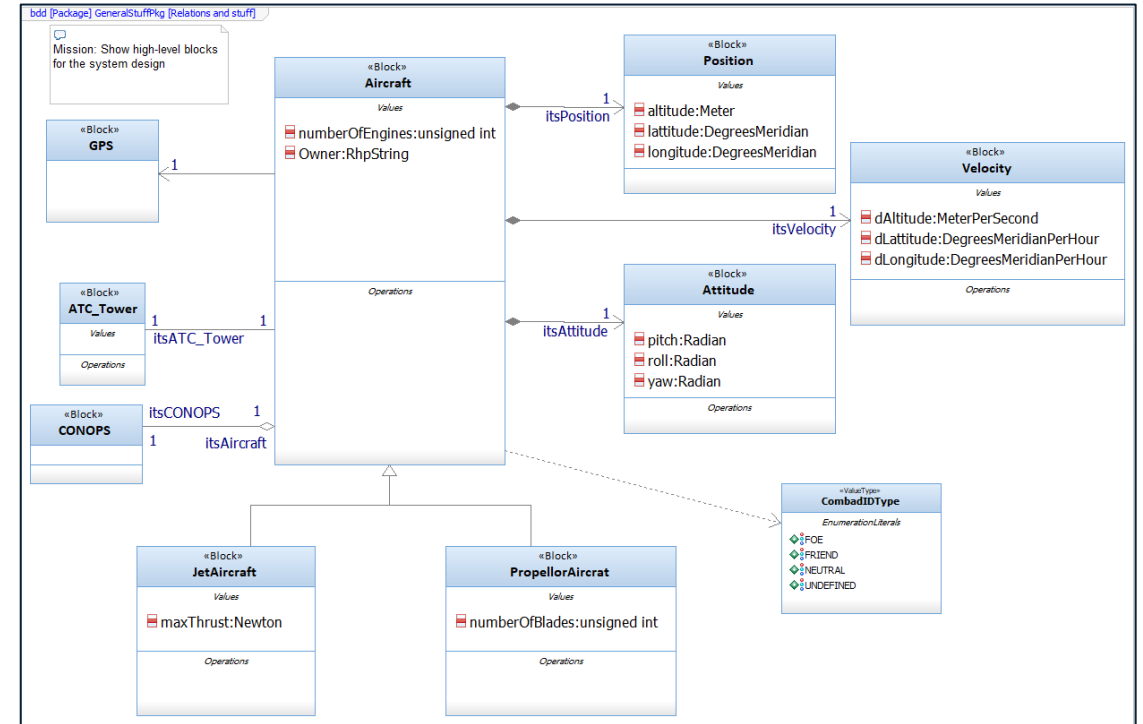
## Always read the direction from the unconjugated side

- Always read the “direction” of the elements in an interface block from the perspective of the *unconjugated* side (without ~)
- **Prov** means that the *unconjugated side provides* the implementation of the requested service
- **Reqd** means the *conjugated side provides* the implement of the requested service
- **In** means the flow property flows *in to the unconjugated side and out from the conjugated side*
- **Out** means the flow property flows *out from the unconjugated side and in to the conjugated side*



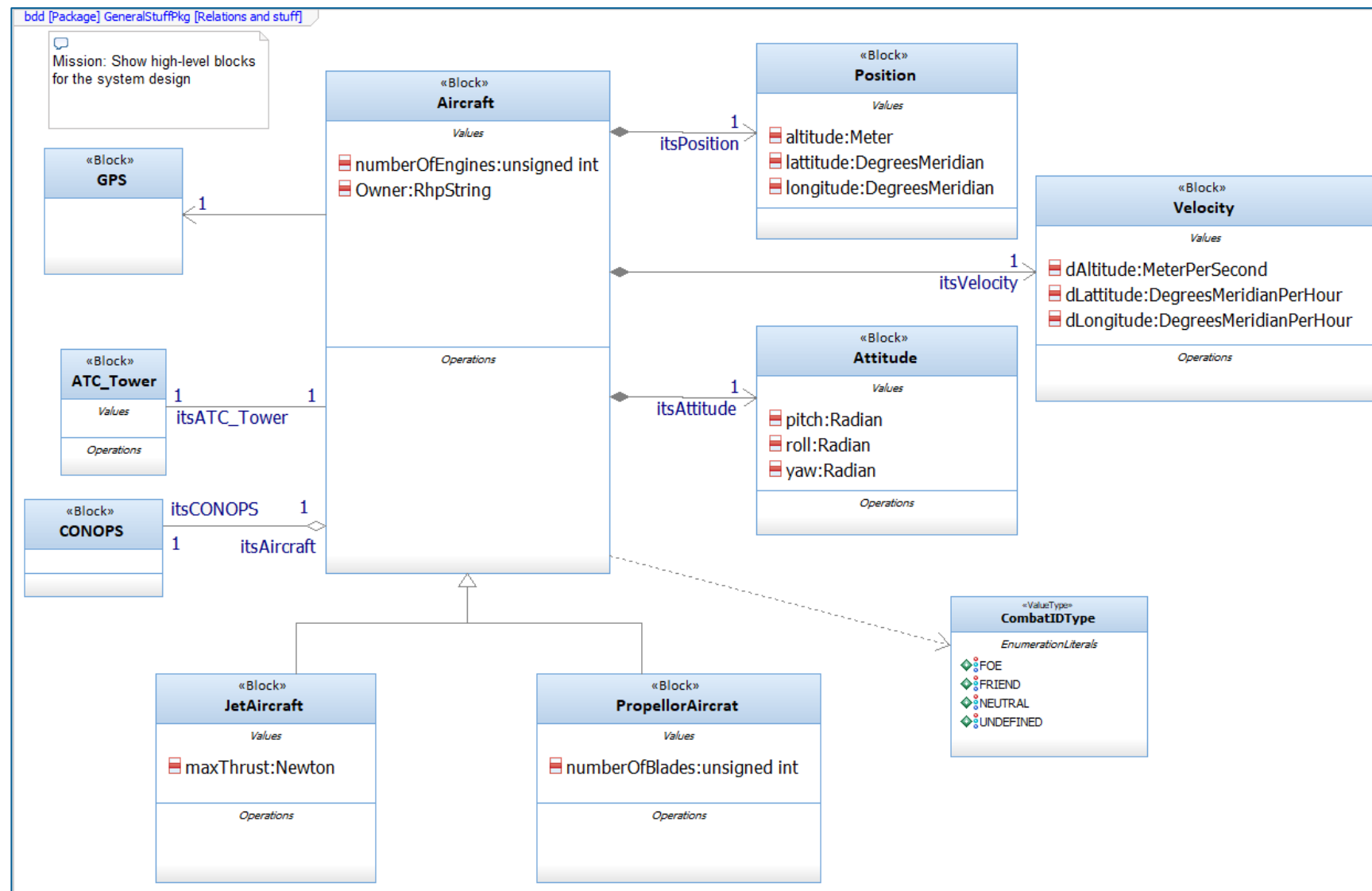


# Block Definition Diagrams (BDD)



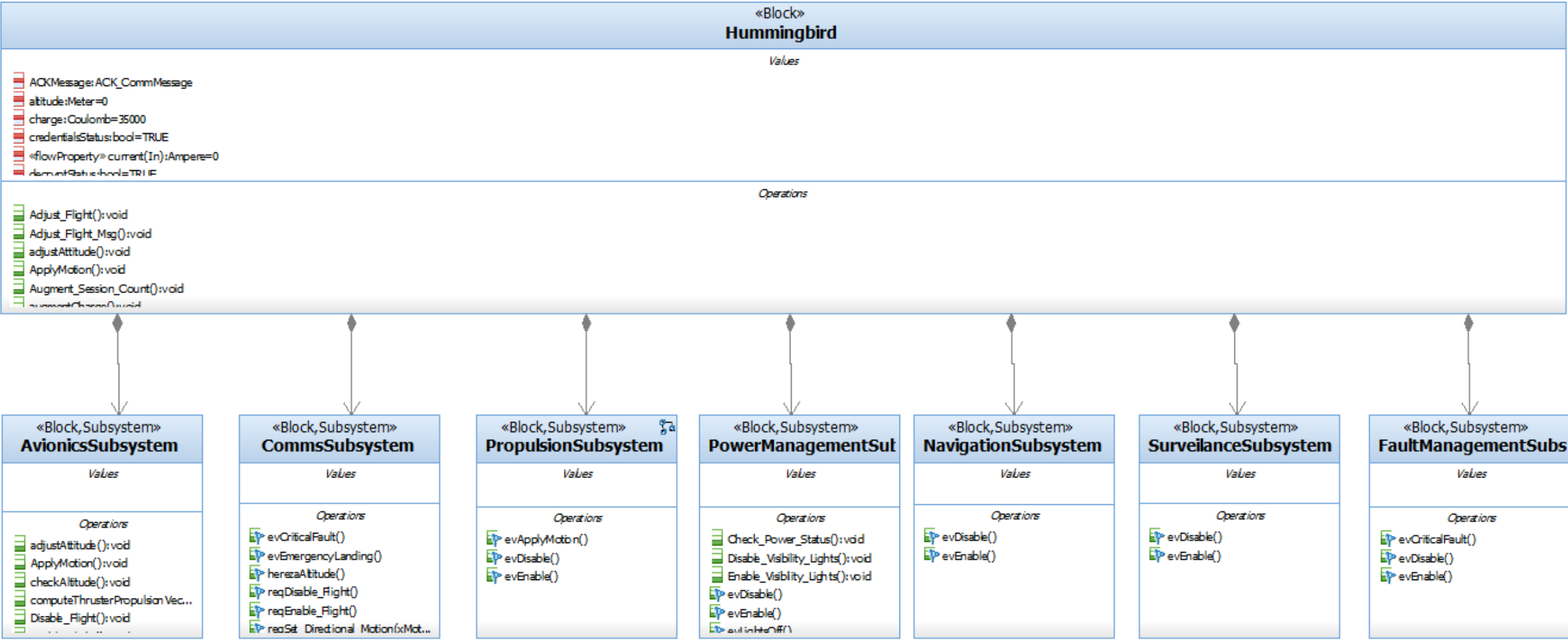
# Structural Diagrams

- Block Definition Diagram
  - Shows blocks and types and their relations
  - May include instances if desired



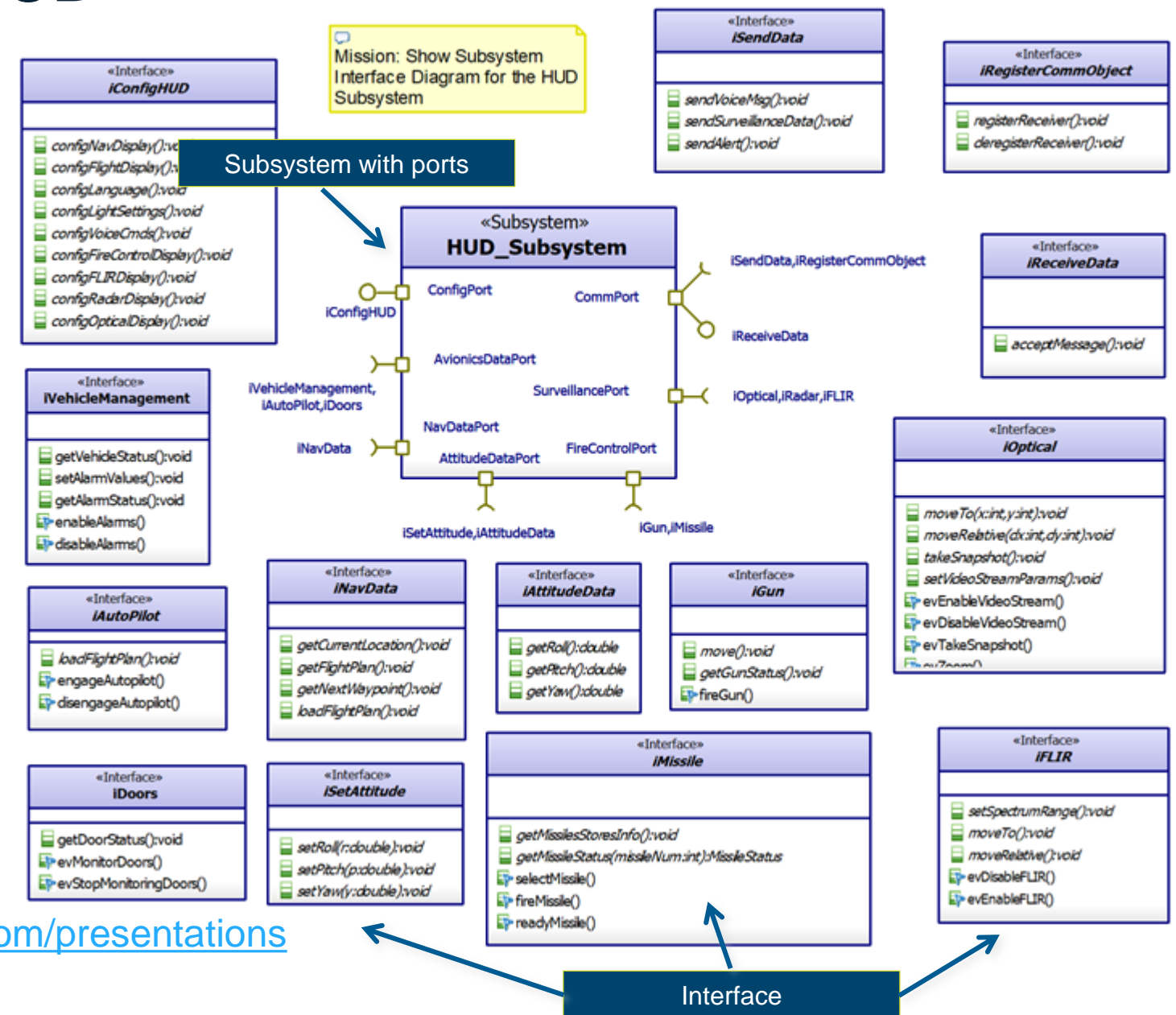
# Another Block Definition Diagram

bdd [Package] ArchitecturalDesignPkg [HummingbirdSE Structure]



# Block Diagrams and ICD

- As an alternative to textual **Interface Control Documents**, BDDs and IBDs can be used to show interface details
- This is an “Interface diagram” →



For more detail see  
“Model-Based ICDs” at  
<https://www.bruce-douglass.com/presentations>



# Antipatterns

Mission statement doesn't describe the diagram contents

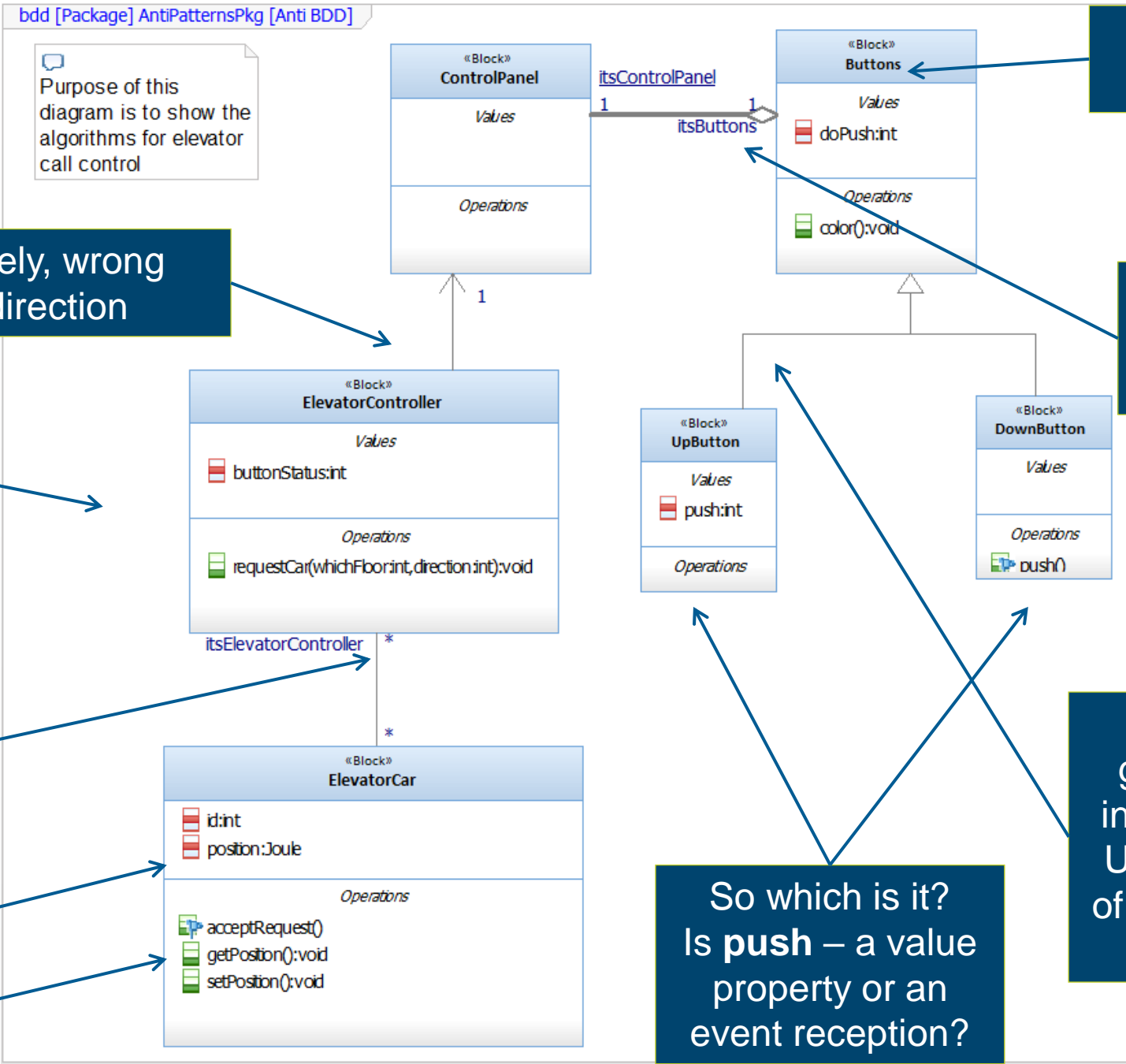
Likely, wrong direction

Why does elevator controller know 'button status'?

There are unlikely to be multiple elevator controllers

Inappropriate type

Missing arguments



Name isn't singular

No, button doesn't aggregate **Control Panel**

This confuses generalization with instantiation / usage: UpButton is a *usage* of button not a *type* of button

So which is it? Is **push** – a value property or an event reception?



# Block Modeling: Key Takeaways

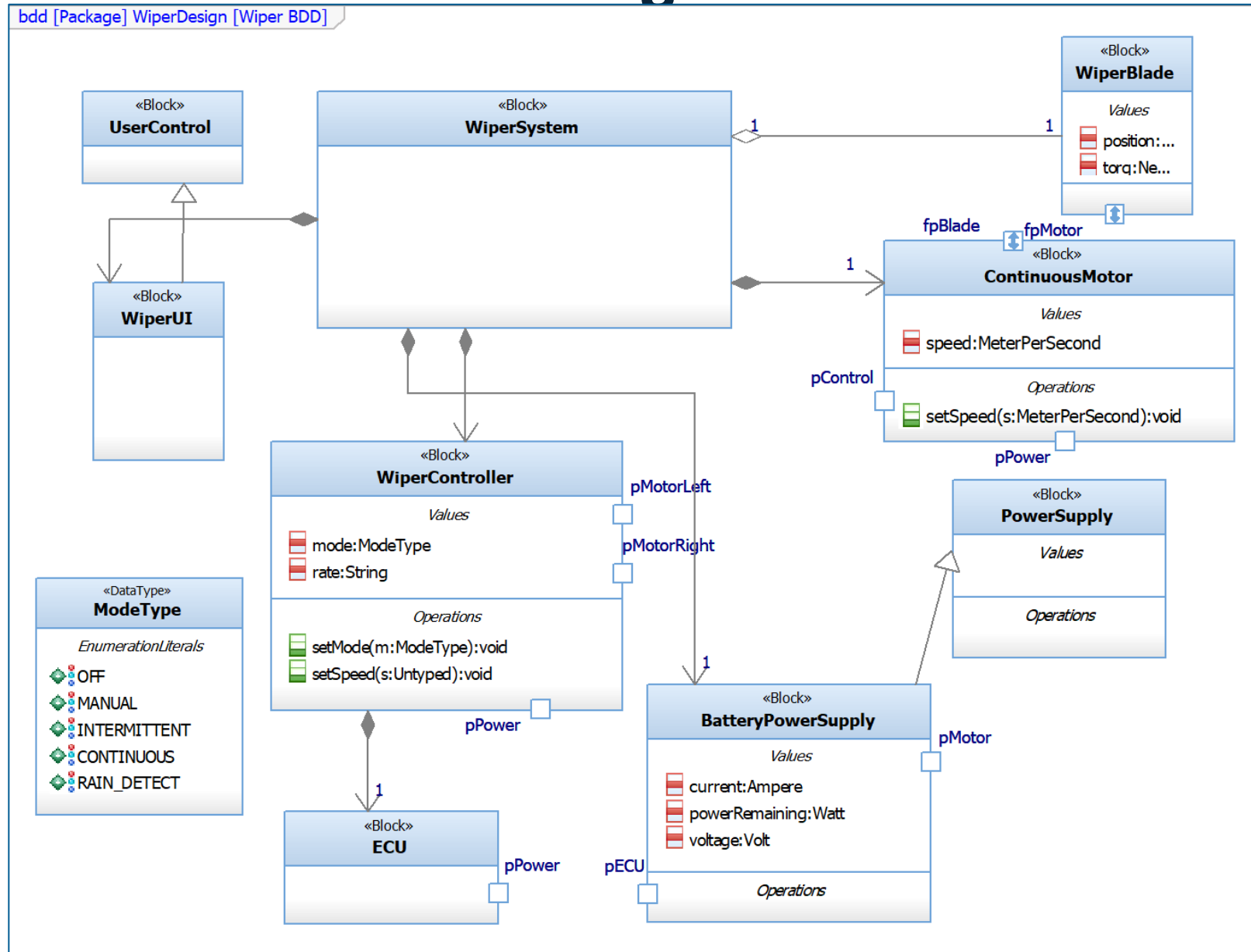
- A **Block** is the foundational structural element in SysML
  - Blocks exist at "analysis/design time"
  - Block serves as a type for an instance
- Blocks contain important features, such as
  - **Value Properties** – values that it owns
  - **Flow Properties** – values that it owns that flow across the block boundary w/o the use of services
  - **Operations** – behaviors that it performs
  - **Signal Receptions** – signals received that invoke behavior
  - **Classifier Behavior** (such as State machines) (optional) – state behavior that it performs
  - **Ports** (optional) – named connection points typed by
    - **Interfaces** (standard ports)
    - **Interface Blocks** (proxy ports)
- Blocks relate to other blocks in a variety of ways
  - **Association** – collaborates with
  - **Aggregation** – (weakly) owns
  - **Composition** – (strongly) owns, creates, and destroys
  - **Generalization** – is-a-kind-of
  - **Dependency**



# Block Definition Diagram Checklist

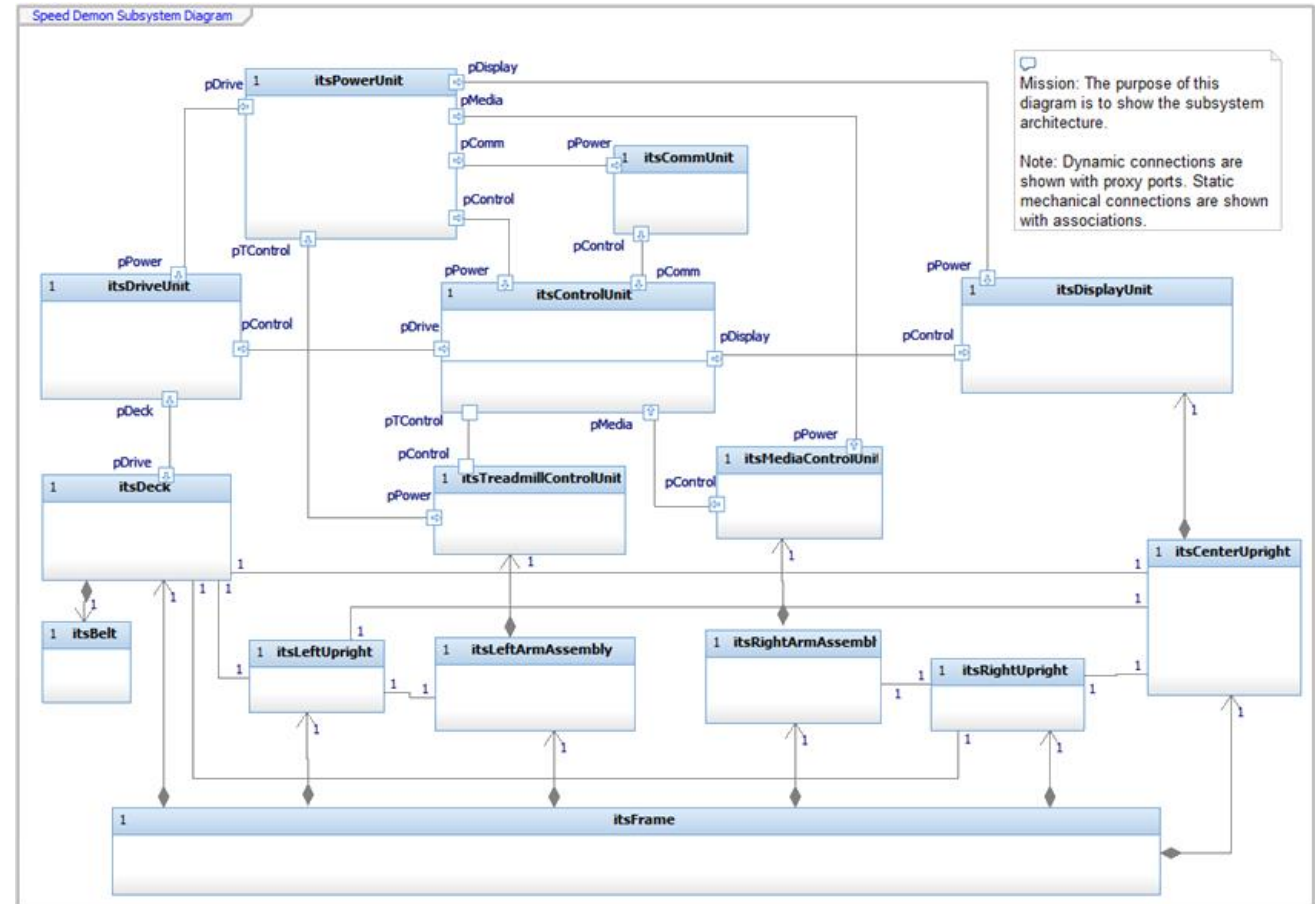
- Does each block
  - ☐ Have a singular noun or noun-phrase name?
  - ☐ Do all views of the block actually relate to the same block definition?
  - ☐ Show relevant properties, such as value properties, operations, receptions, and ports?
  - ☐ Appropriately type all value and flow properties?
  - ☐ Appropriately type all ports with elaborated interfaces (standard ports) or interface blocks (proxy ports)?
  - ☐ Show all operation and reception parameters with their appropriate types?
  - ☐ Avoid the use of generic primitive types, such as *int*, when enumerated or elaborated types are more appropriate (e.g. **CombatID** using a **CombatIDType** enumeration type rather than *int*)?
  - ☐ Use relations appropriately?
    - ☐ Association
    - ☐ Aggregation
    - ☐ Composition
- ☐ Does the diagram
  - ☐ Identify its purpose and scope?
  - ☐ Contain only elements relevant to its purpose and scope?
  - ☐ Contain all elements relevant to its purpose and scope?
  - ☐ Minimize line crossing?

## Exercise: Evaluate the following Block Definition Diagram



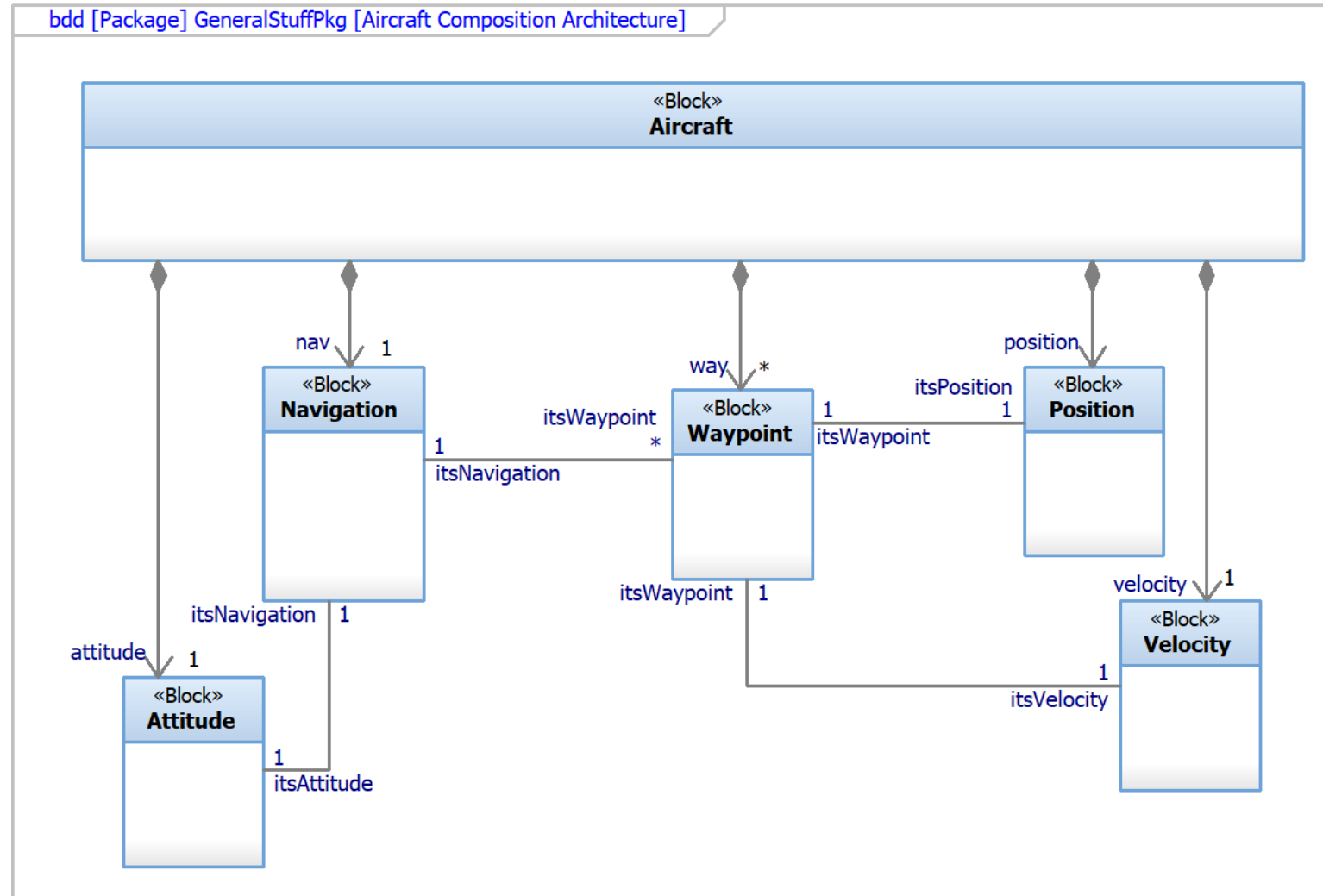


# Internal Block Diagrams



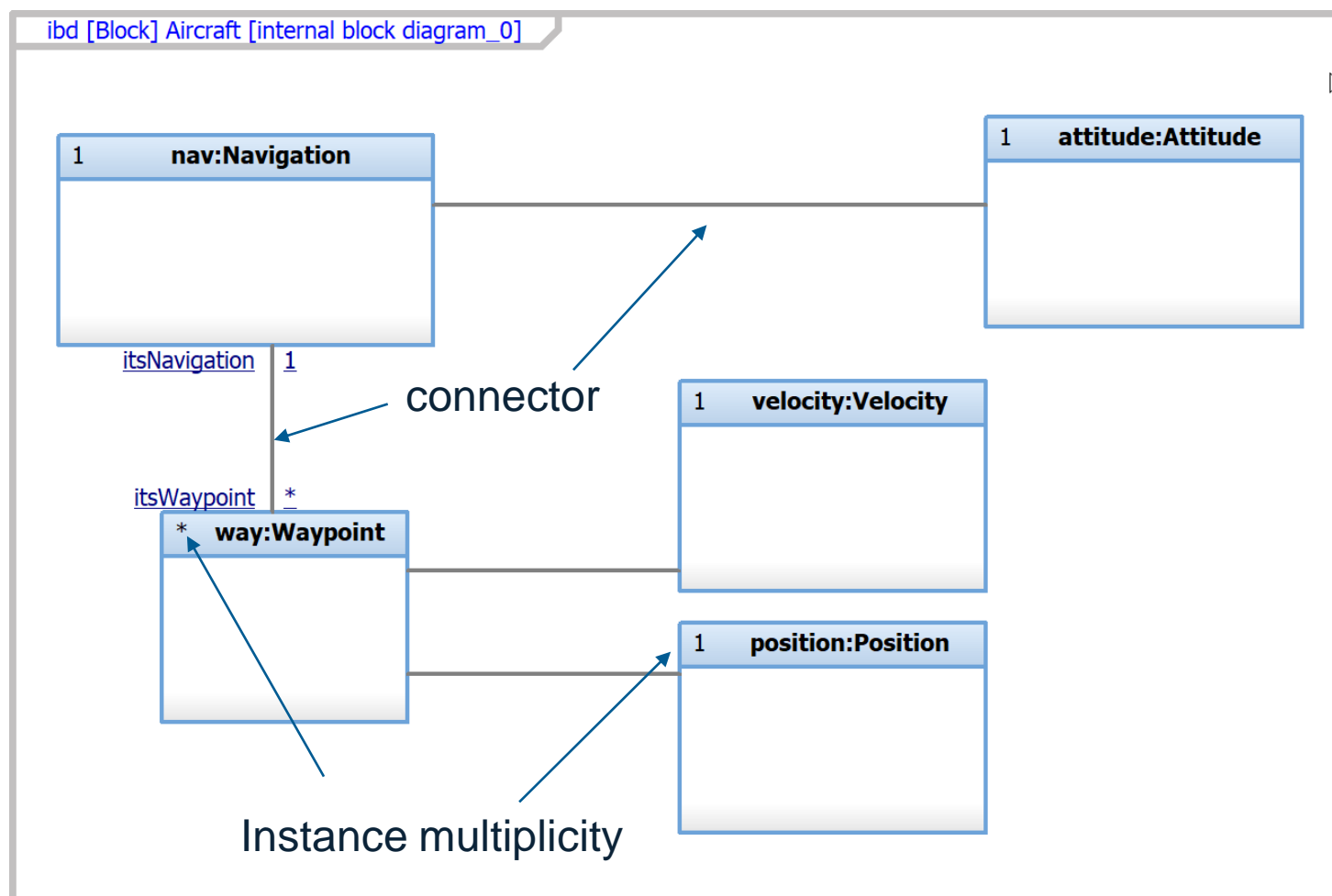
# Internal Block Diagrams for Associations

- IBDs display the parts of a block and their relations.
- Consider the BDD at the right



# Internal Block Diagrams for Associations

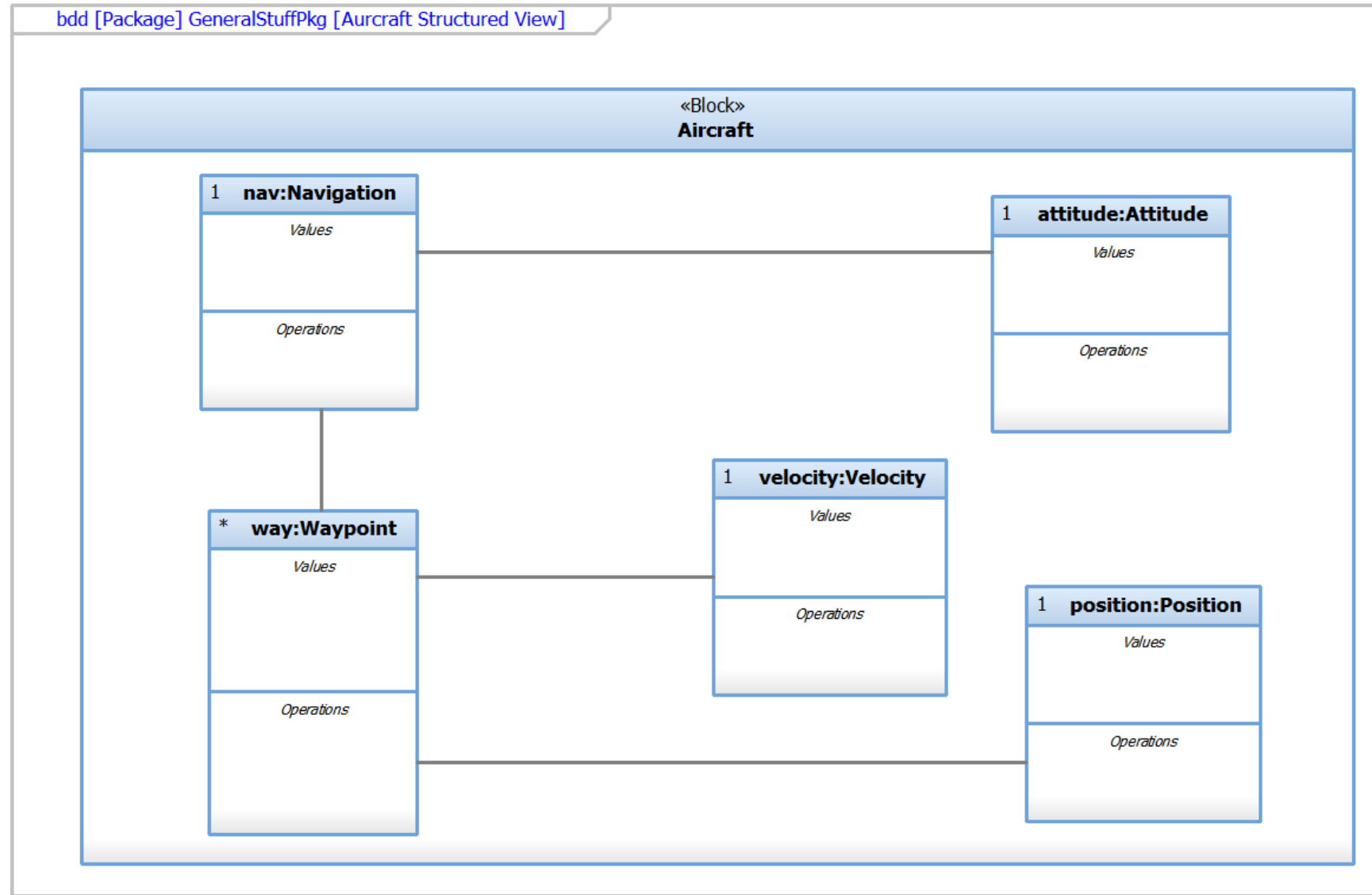
- Shows instances and their connections
- Shows the internal decomposition of blocks into contained parts
- Some tools indicate multiplicity inside square brackets
  - For example, nav:Navigation[1]
- Remember that while ports cannot be connected on block definition diagrams, they can be connected on internal block diagrams
- Note the names of the parts match the role end names of the composition relations, and the instance multiplicity matches the role end multiplicity



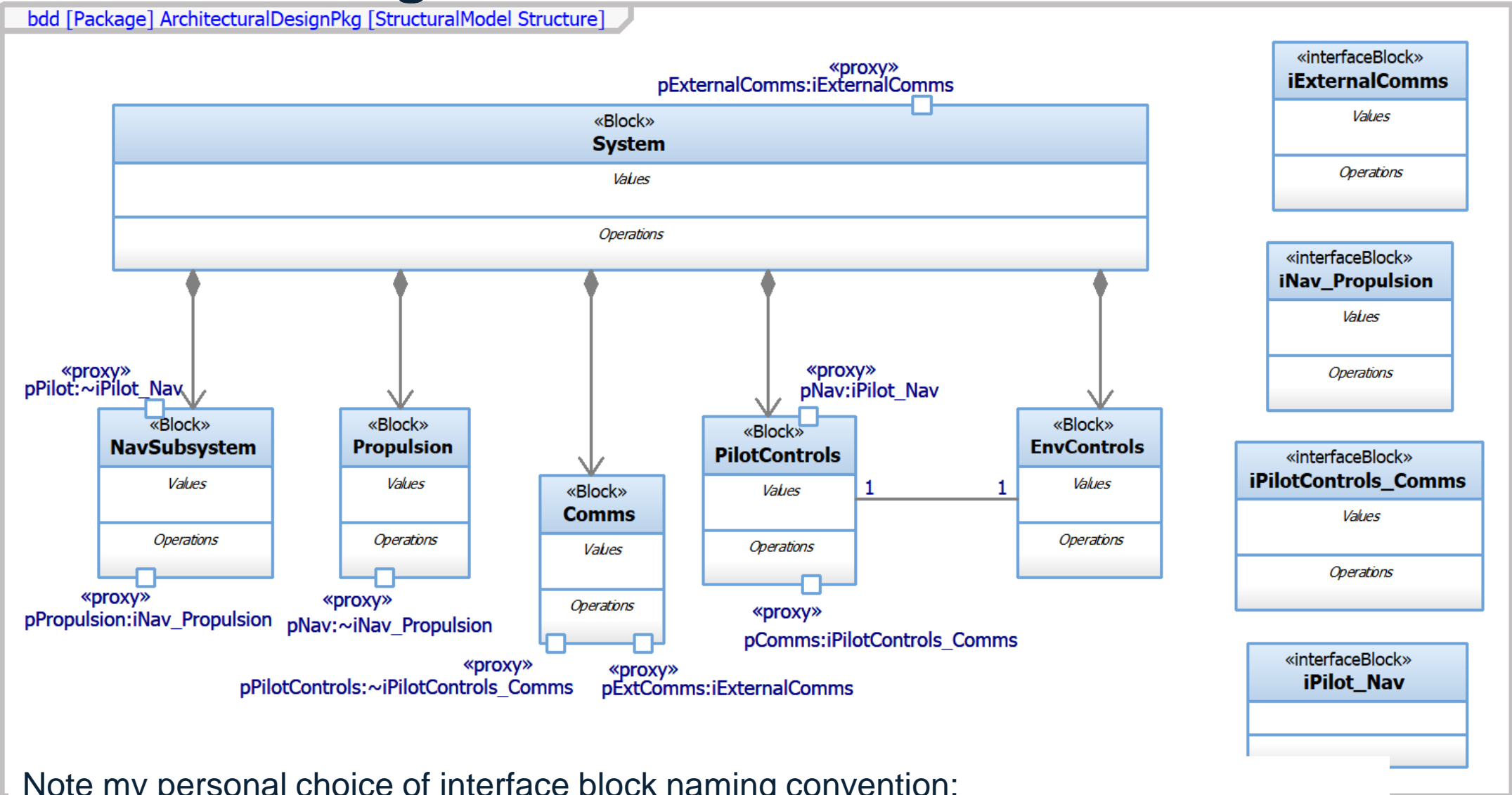
Note that you can optionally show role end names and multiplicity on the connectors (from the associations) if desired.

# Side Note: Structured View on Block Definition Diagrams

- Note that this is a block definition diagram but the **aircraft** block is shown in *structured view* rather than the more common *compartment view*



# Internal Block Diagrams with Ports

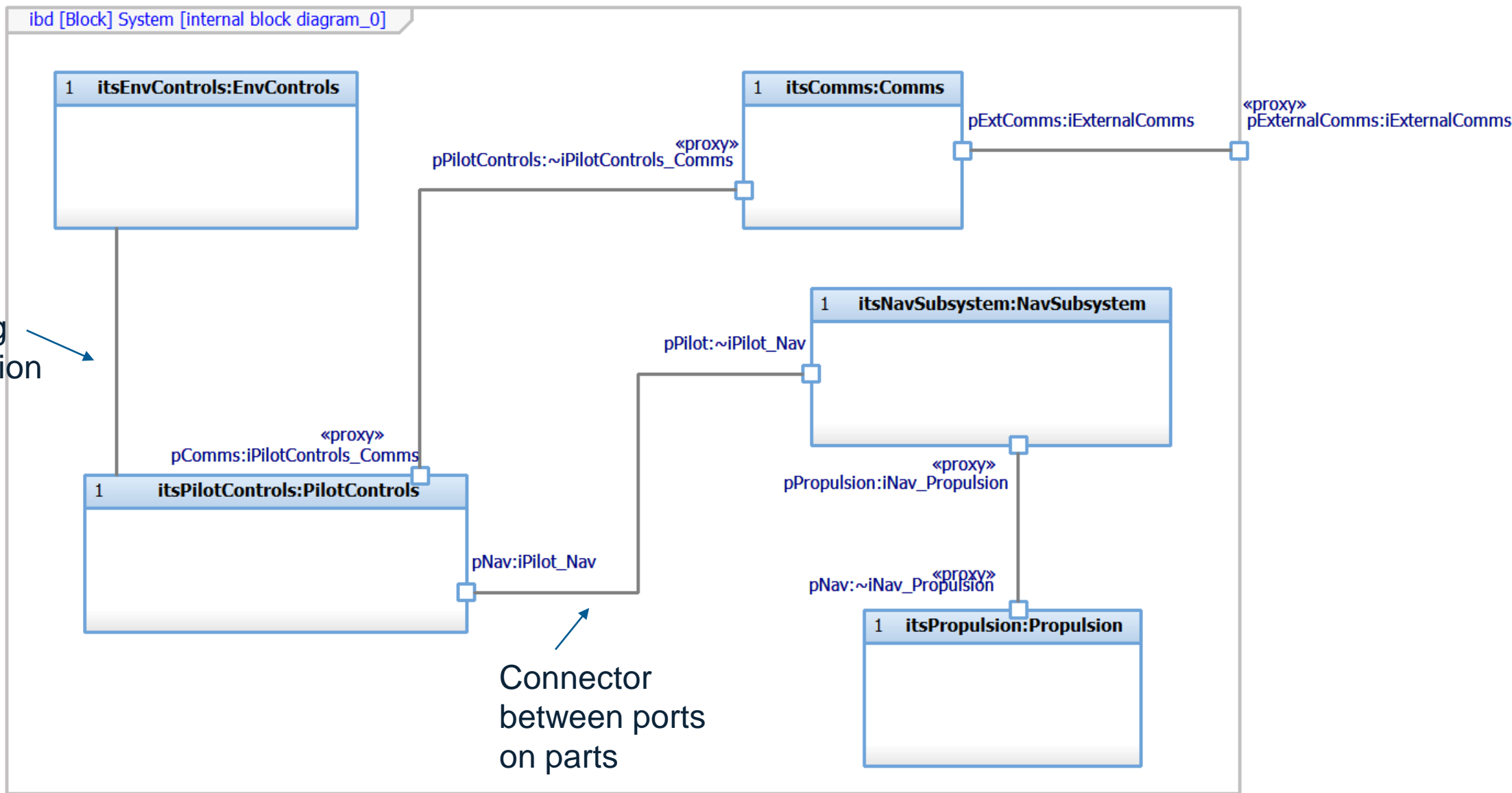


Note my personal choice of interface block naming convention: “i” + <unconjugated side> + “\_” + <conjugated side>. Ports are named: “p” + <target block>



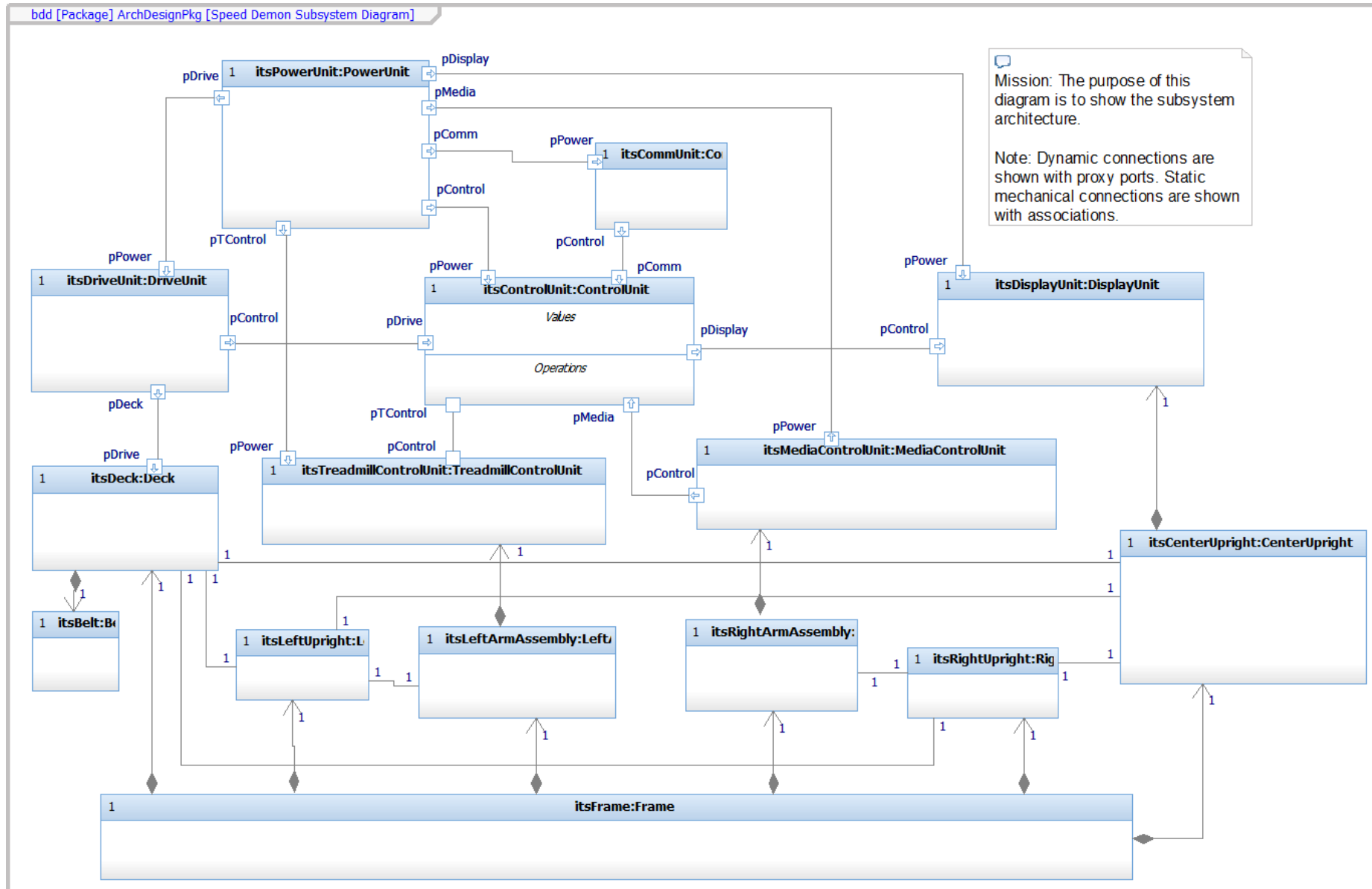
# Internal Block Diagrams with Ports

Connector  
instantiating  
an association

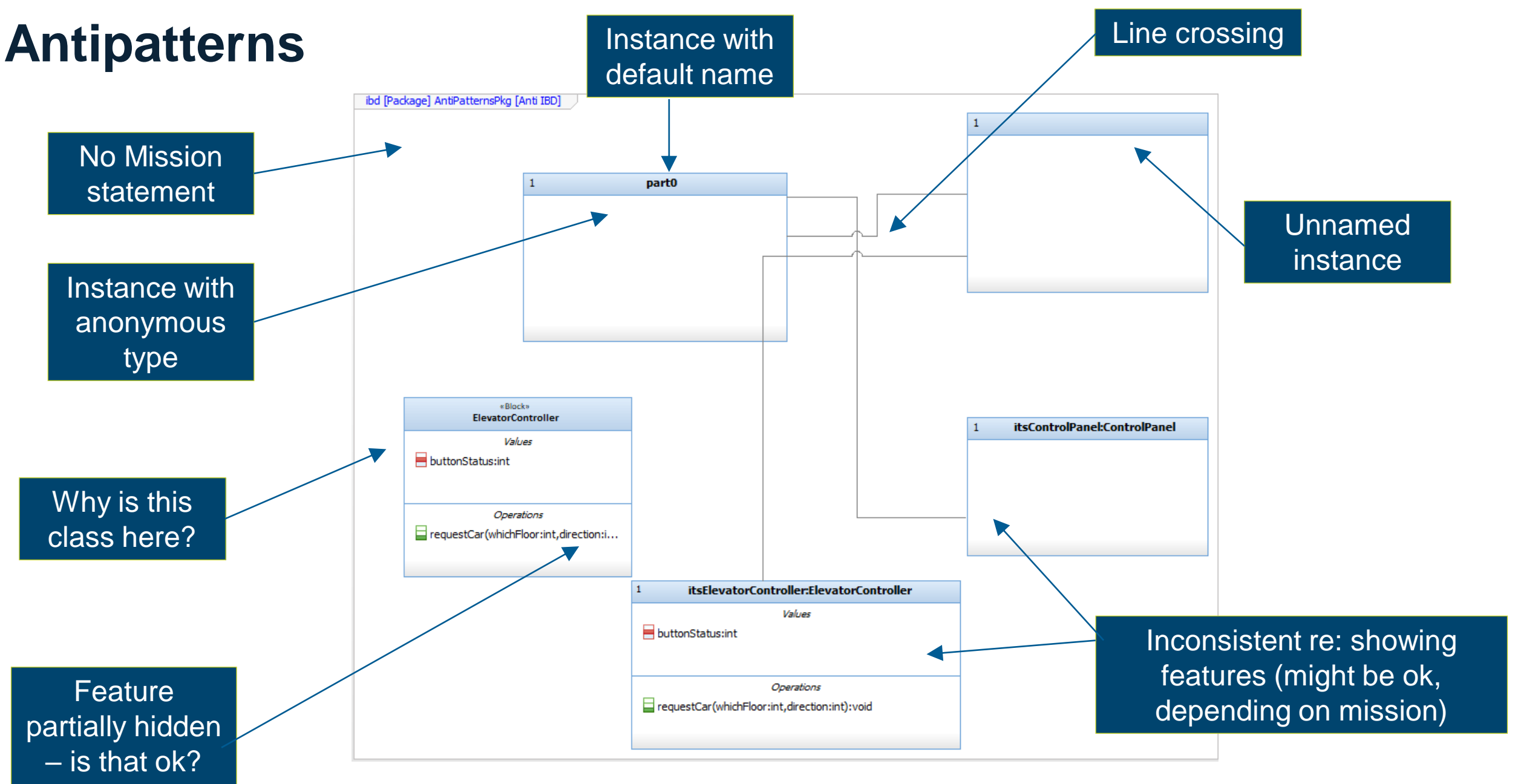


Connector  
between ports  
on parts

# An Internal Block Diagram for The Speed Demon™ Treadmill



# Antipatterns



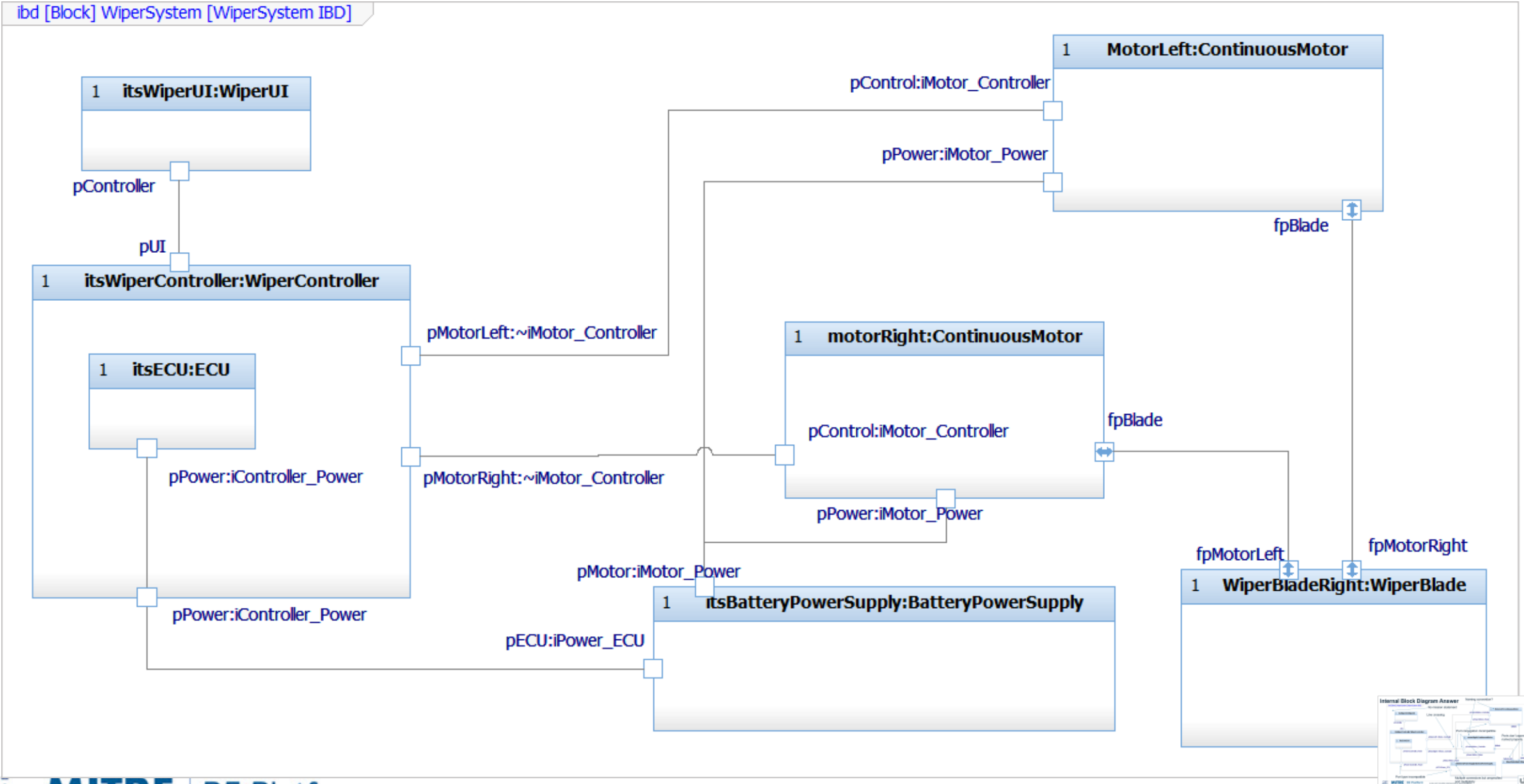




# Internal Block Diagram Checklist

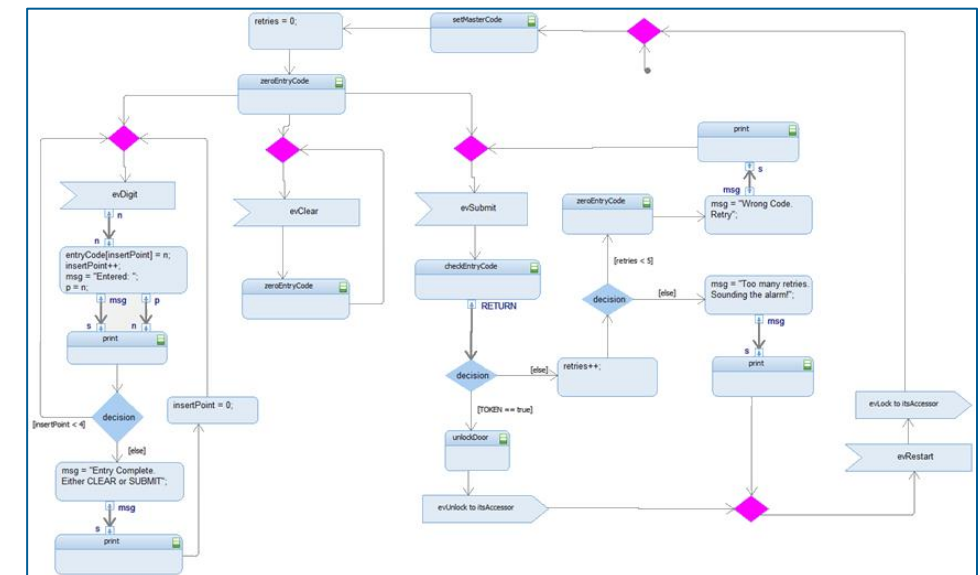
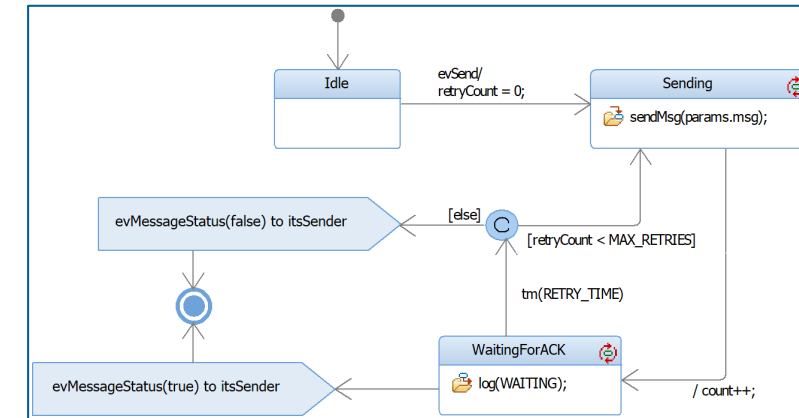
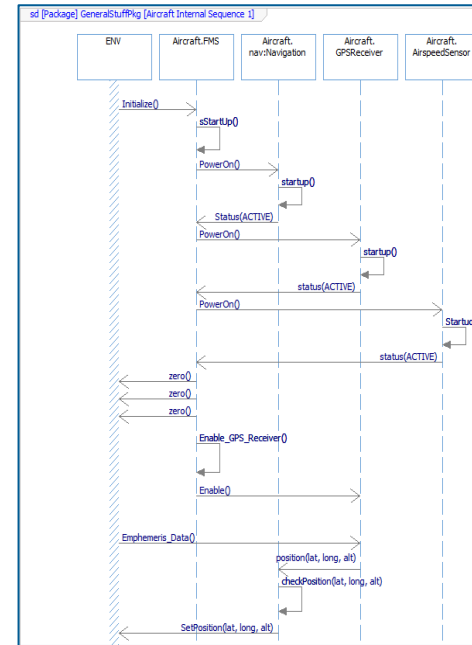
- Does each block instance
  - ☐ Have a singular noun or noun-phrase name?
  - ☐ Show its type (block)?
  - ☐ Do all instances of the block actually reference the same block definition?
  - ☐ Show relevant properties, such as value properties, operations, receptions, and ports?
  - ☐ Show appropriate connectors to other instances?
  - ☐ Use relations appropriately?
    - ☐ Association
    - ☐ Aggregation
    - ☐ Composition
- ☐ Does the diagram
  - ☐ Identify its purpose and scope?
  - ☐ Contain only elements relevant to its purpose and scope?
  - ☐ Contain all elements relevant to its purpose and scope?
  - ☐ Minimize line crossing?

# Exercise: Evaluate the following Internal Block Diagram

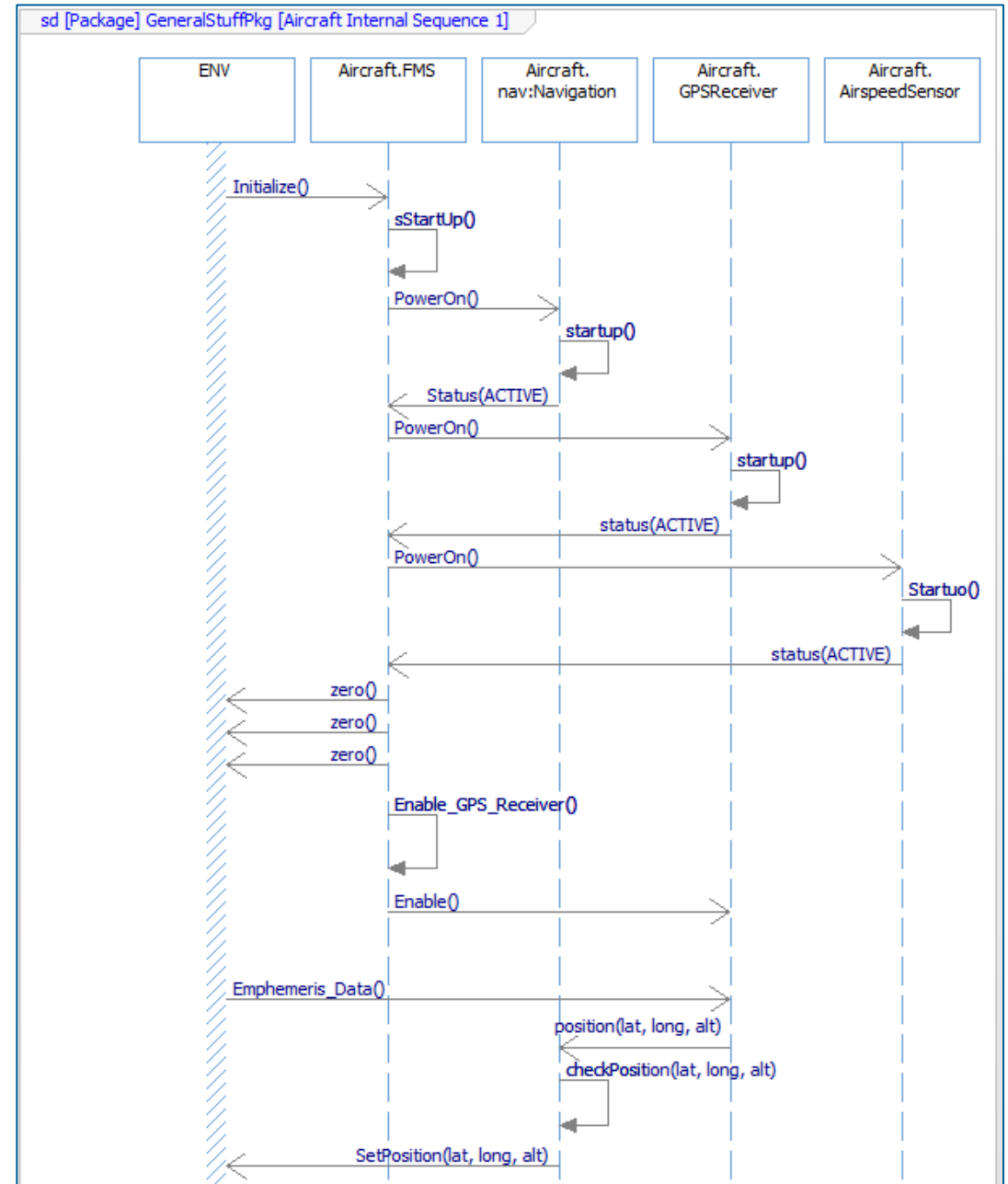


# Basic Behavioral Modeling in SysML

Sequence Diagrams  
Activity Diagrams  
State Diagrams  
Checklists

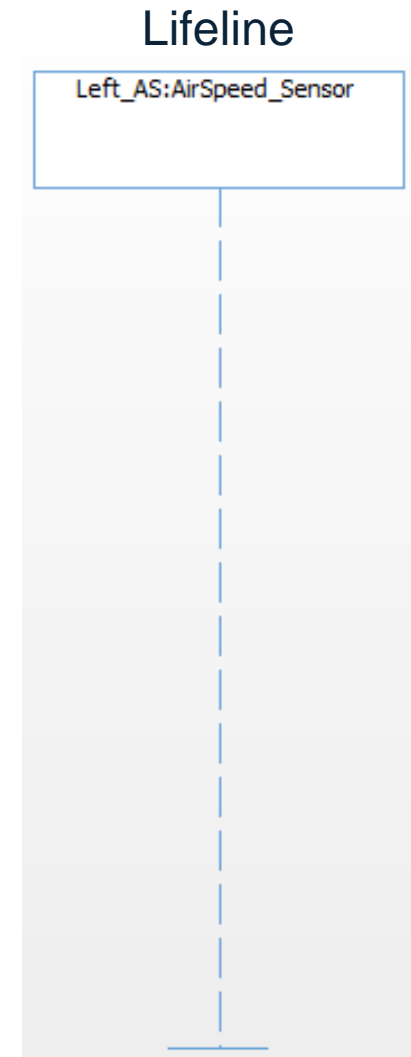


# Sequence Diagrams



# Sequence Diagrams

- Sequence diagrams show the behavior of a group of instance roles over time in specific circumstances.
- Messages are shown as arrowed lines from the sender to the receiver
- Time flows down the page (mostly)
- Useful for
  - Capturing typical or exceptional interactions for requirements
  - Understanding collaborative behavior
  - Demonstrating that collaboration realizes use case properly
  - Testing collaborative behavior with model-based testing (see UML Testing Profile)



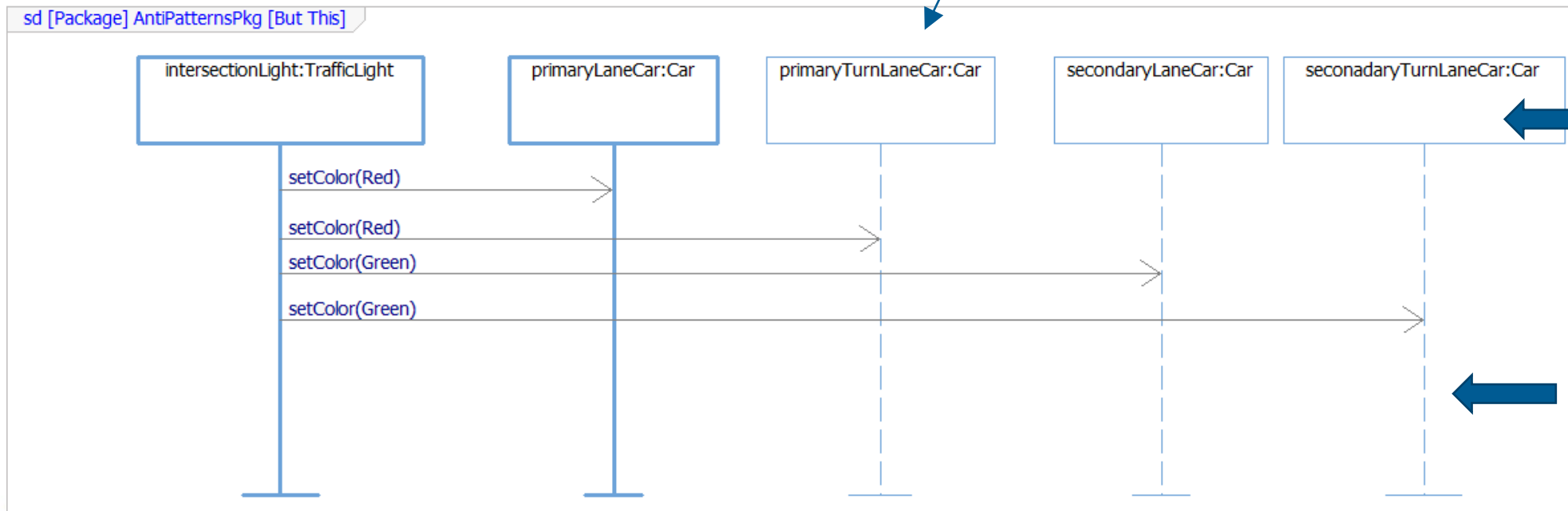
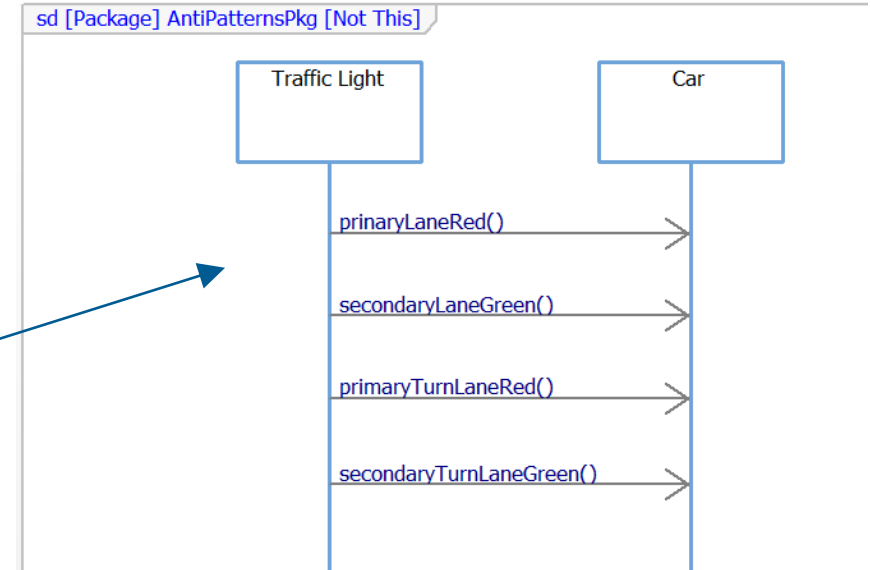
# Sequence Diagram elements

- Lifelines are roles that instances of blocks play in the context of the scenario, including
  - Architectural blocks – e.g. system, subsystem
  - Design elements – e.g. sensor, actuator, controller
  - Use cases – a stand-in for the system in the context of the use case scenario
  - Actors – objects outside the scope of the system that contribute to the interaction
  - ENV – the “system boundary” lifeline that stands for “everything else in the universe other than those lifelines explicitly shown”
- Messages may be
  - Asynchronous event receptions
  - Synchronous – most commonly “function calls” to the target lifeline services
  - Messages to self –
    - One function owned by an instance calls another function provided by the same instance
    - An event sent by the instance state machine to itself
    - Timeouts – typically show up on state machines as a timeout event

# More on sequence diagrams

- A lifeline is a representation of a **role** – *the usage of an instance of a type in a context*
  - Lifelines may be instance roles of actors, use cases, systems, subsystems, etc
  - Lifelines can be the source or target of messages

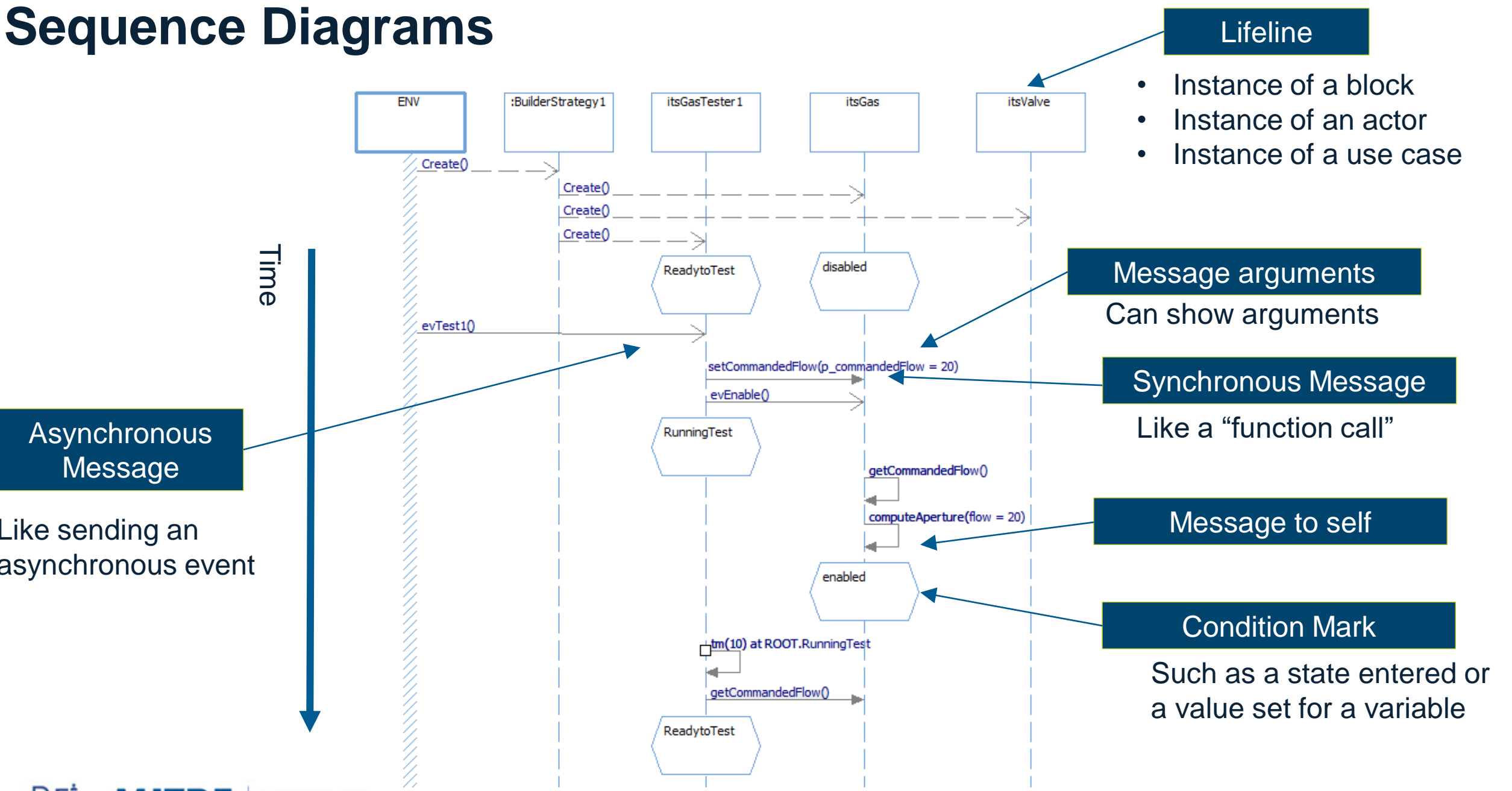
Which is better?



Lifelines are typed

Different roles are separated

# Sequence Diagrams



**Lifeline**

- Instance of a block
- Instance of an actor
- Instance of a use case

**Message arguments**

Can show arguments

**Synchronous Message**

Like a “function call”

**Message to self**

**Condition Mark**

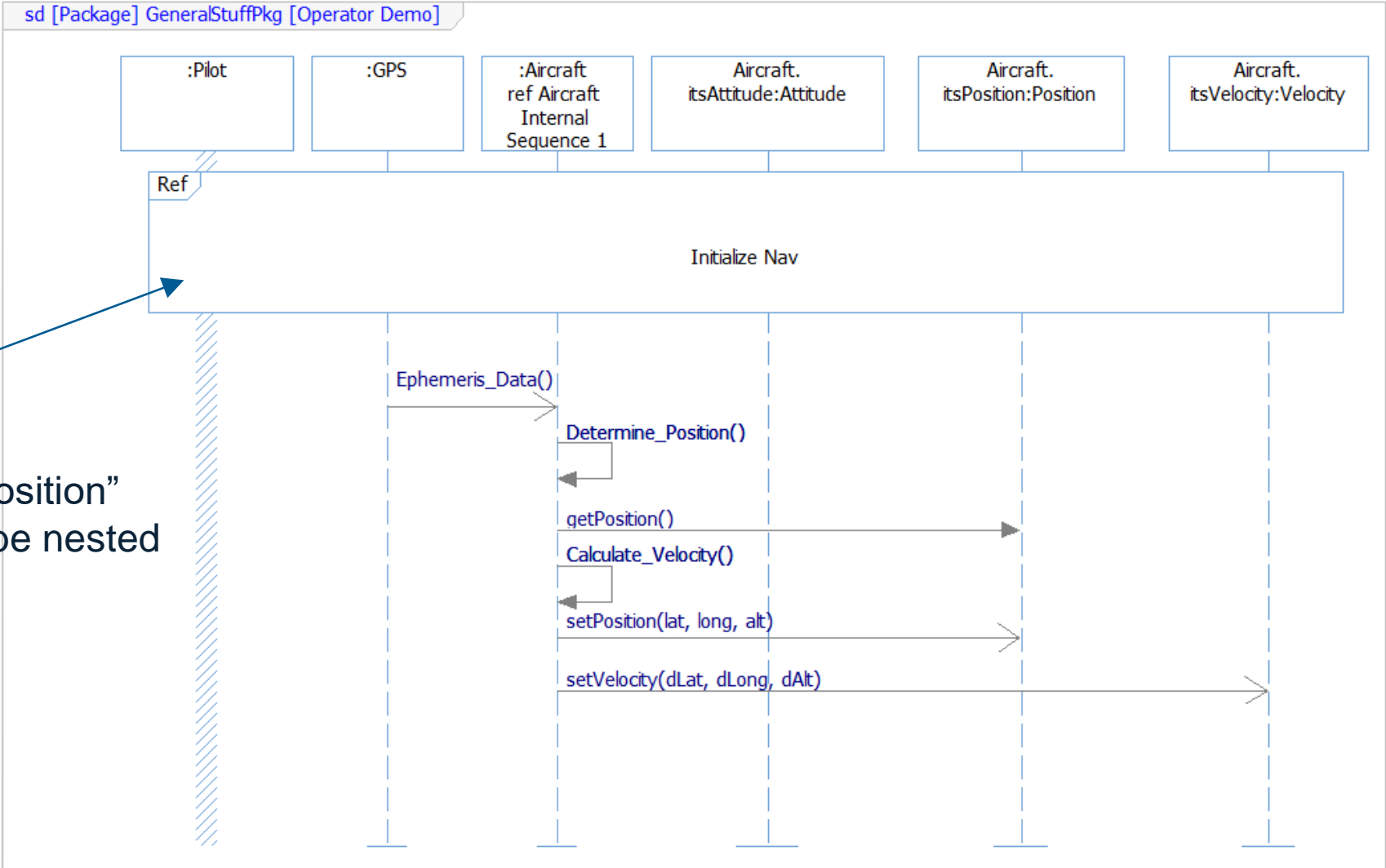
Such as a state entered or a value set for a variable

**Asynchronous Message**

Like sending an asynchronous event



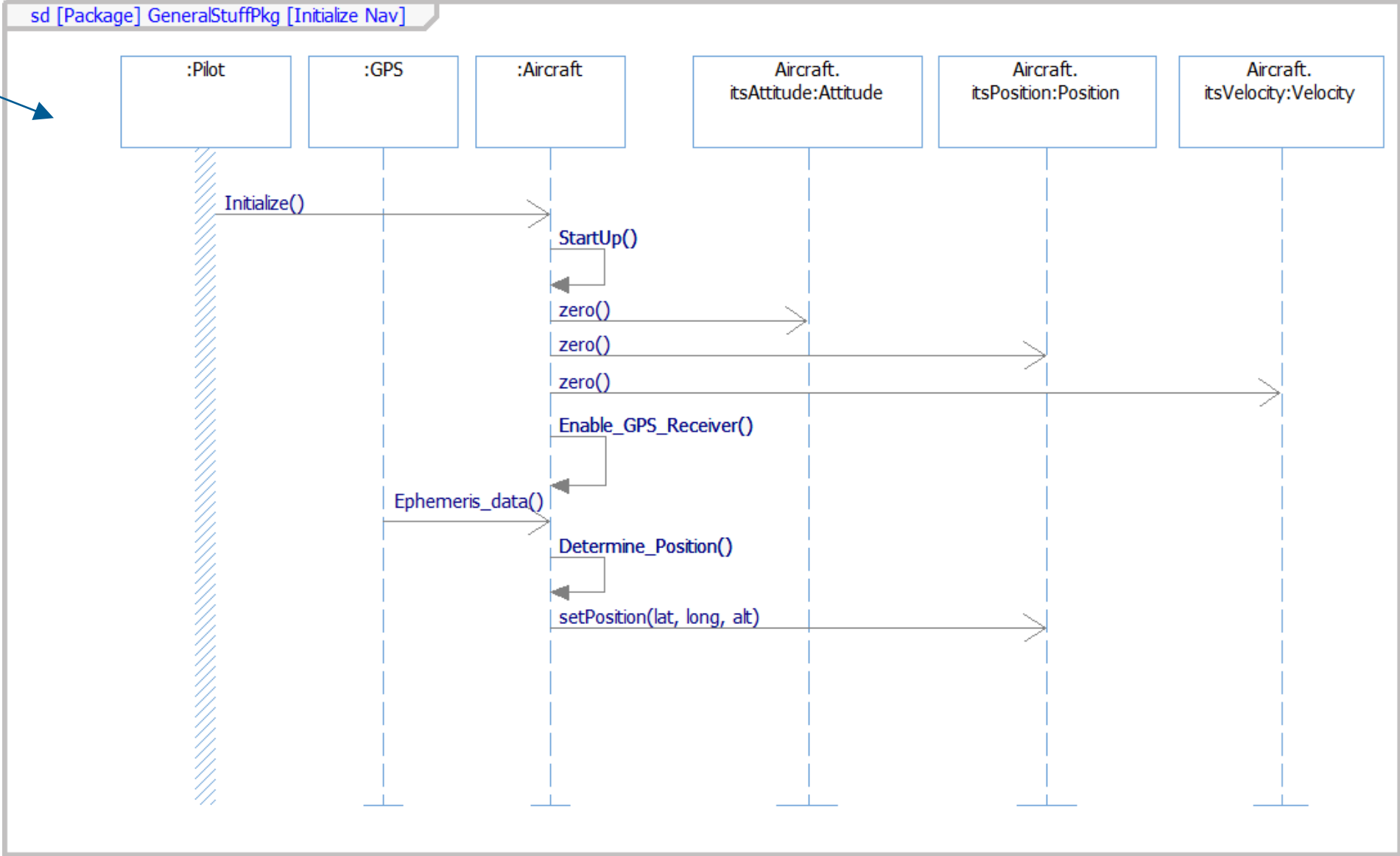
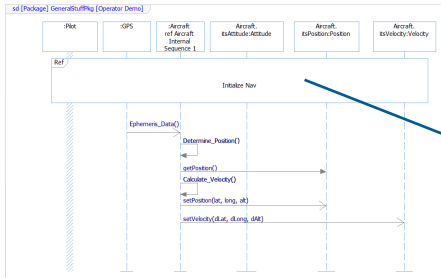
# Decomposing Sequence diagrams 1



Referenced  
Sequence  
diagram

Known as “horizontal decomposition”  
this allows sub-sequences to be nested

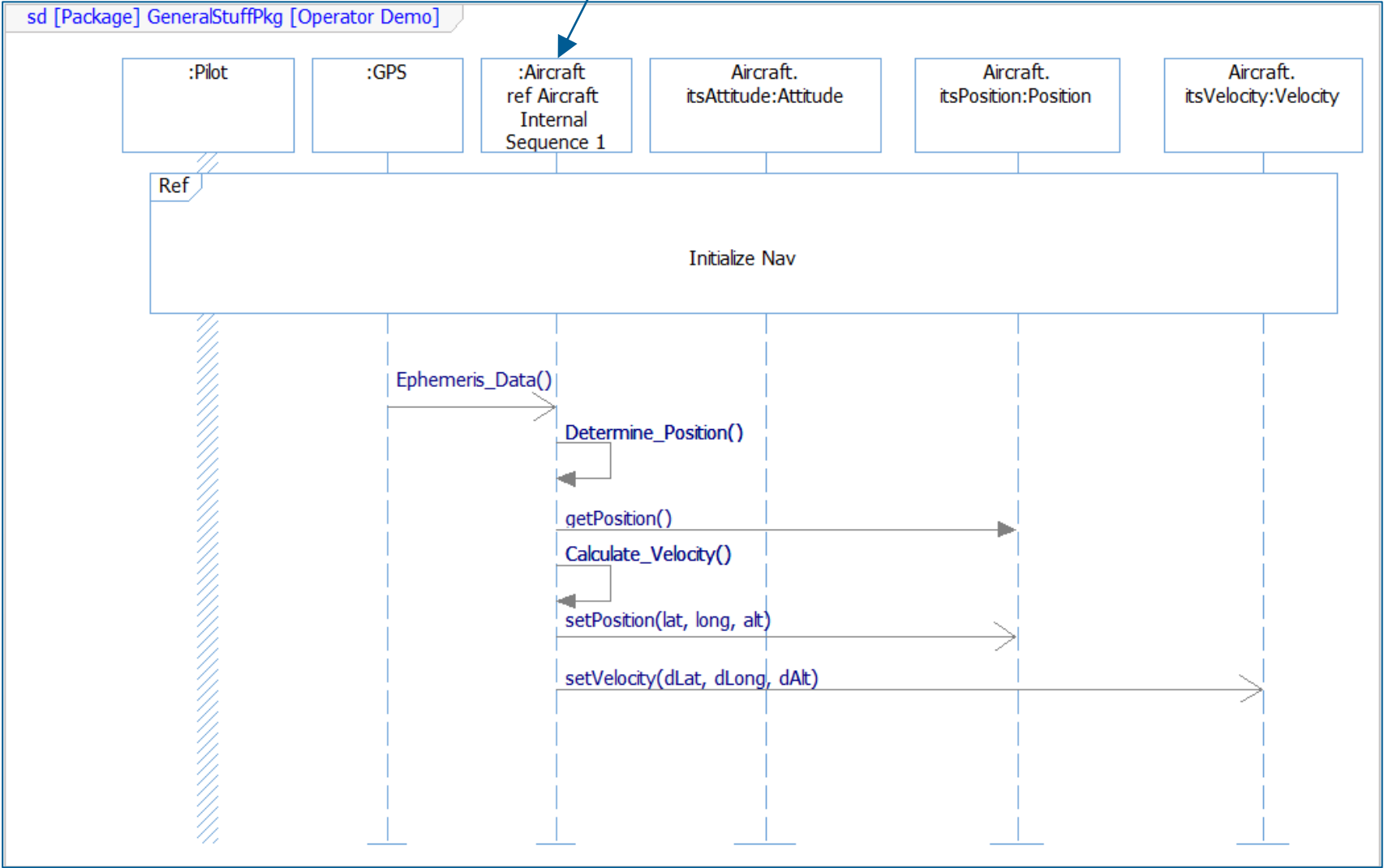
# Referenced Sequence Diagram



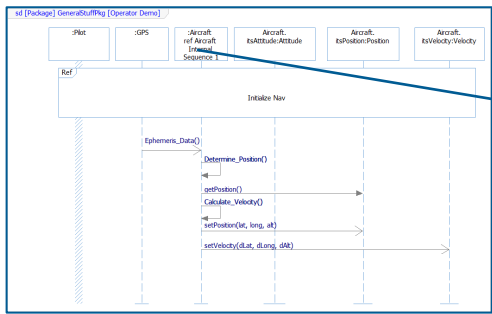
# Decomposing Sequence diagram 2

Decomposed Lifeline

- Known as “vertical decomposition” this lifelines to be decomposed



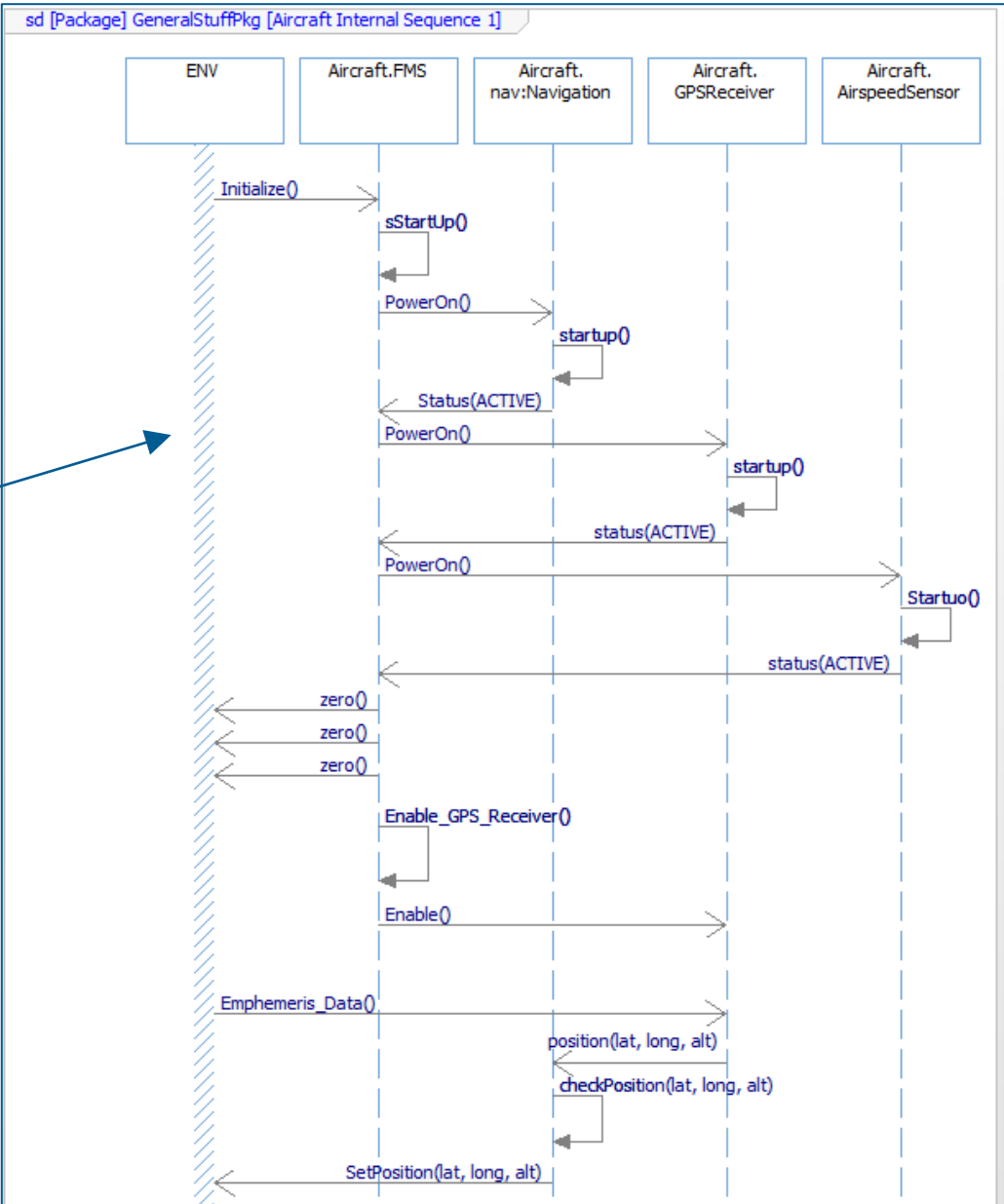
# Decomposed Lifeline



Environment  
lifeline

Shows messages in and out at  
the higher level.

These should match the higher  
level sequence diagram for  
messages to and from the Aircraft  
lifeline in the higher-level SD

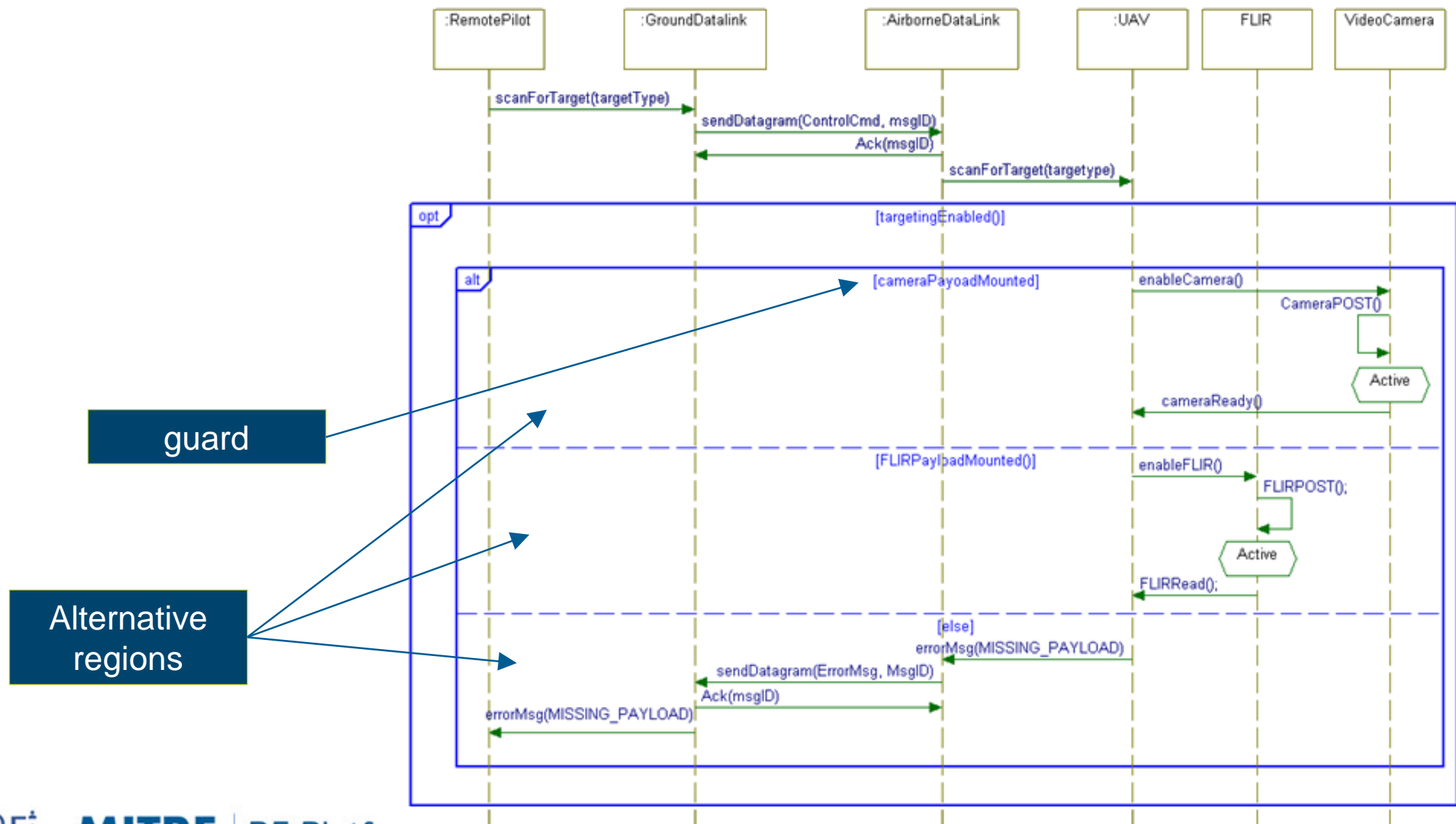


# Interaction Fragment Operators

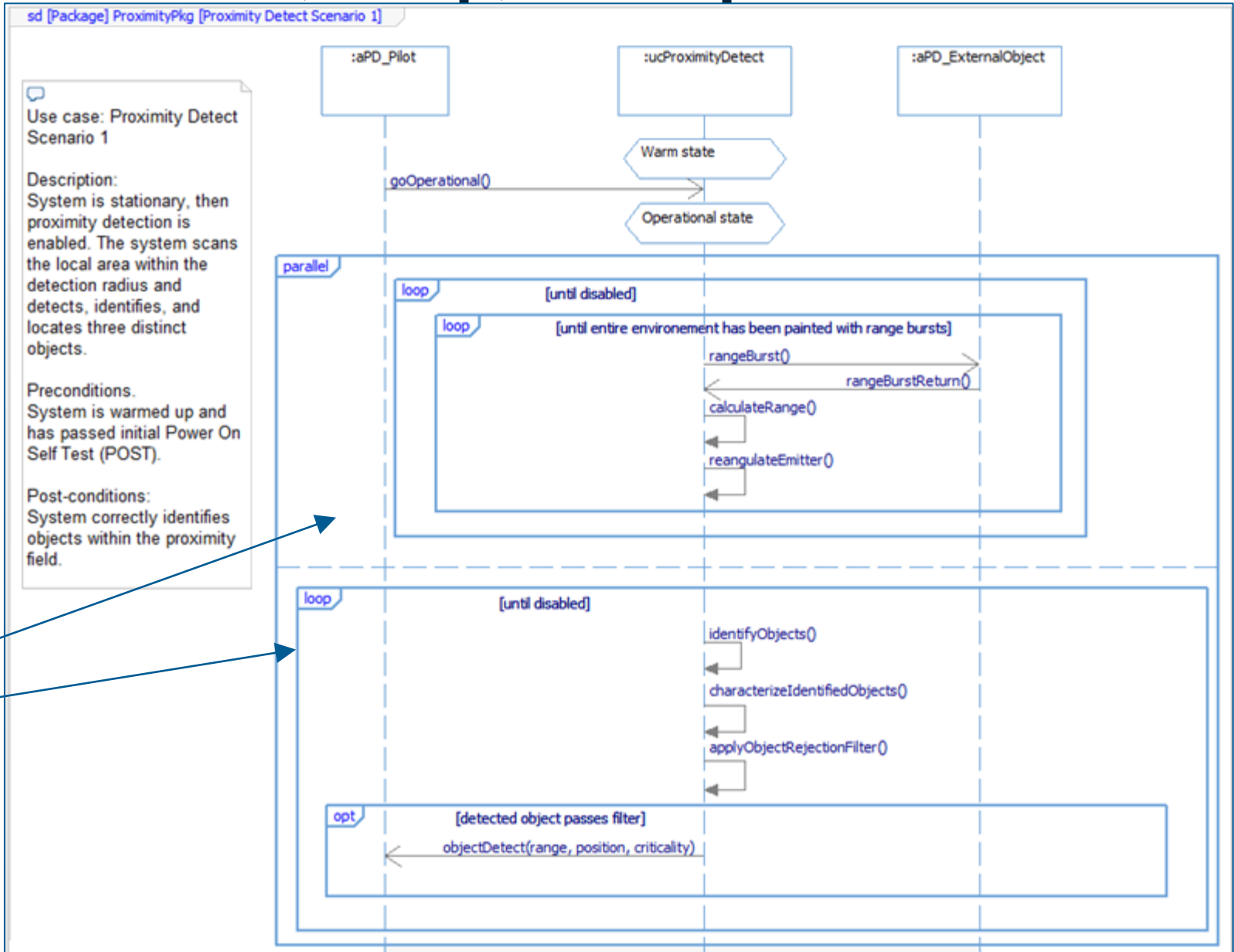
Most commonly used

Category	Operator	Description	Syntax
<b>Naming</b> ★	sd	Name the interaction diagram	sd <name>
	ref	Reference an interaction diagram	ref <name>
<b>Control Flow</b> ★	loop	Repeat interaction fragment	loop <range> <loop condition> loop (0, *) [until stop pressed]
	opt	Optionally execute interaction	Opt <condition> opt [x<10]
	alt	Selection	alt [case 1] ... [case n] ... [else] Alt [x<10]
	par, para, parallel	Parallel execution	par [guard1] ... [guard n]
<b>Ordering</b>	seq	Partial or “weak” ordering (default)	seq
	strict	Strict ordering	strict
	critical	Critical region	critical
<b>Causality</b>	assert	The interaction fragment represents an assertion	assert
	neg	The interaction represents a negative assertion (“can’t happen”)	neg
	ignore/ consider	Ignore or consider the interaction fragment to be within the causal stream	ignore    consider (message name list) ignore (m1, m2) consider (m3, m4, m5)

# Interaction Operator : Opt and Alt



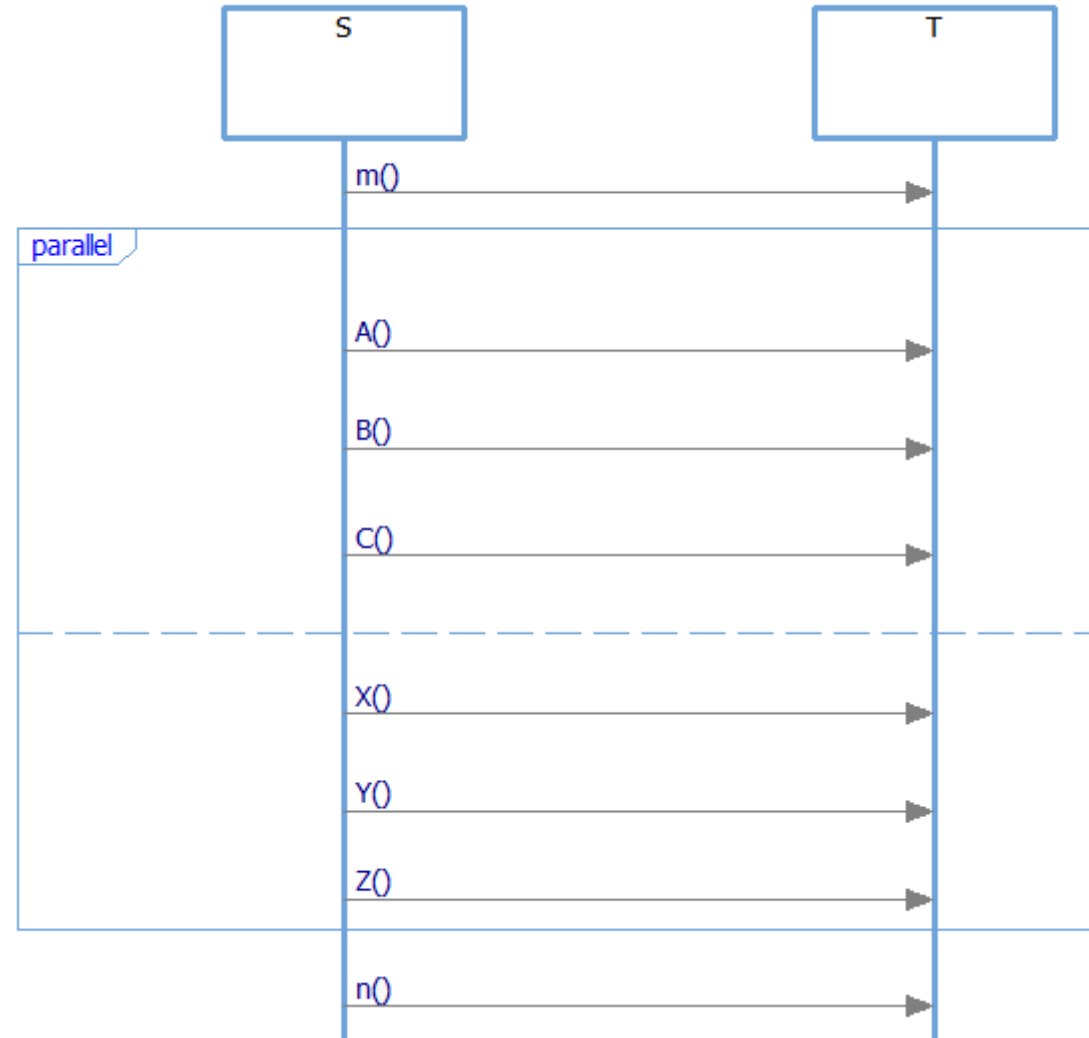
# Interaction Operator : Parallel, Loop, and Opt



Parallel regions

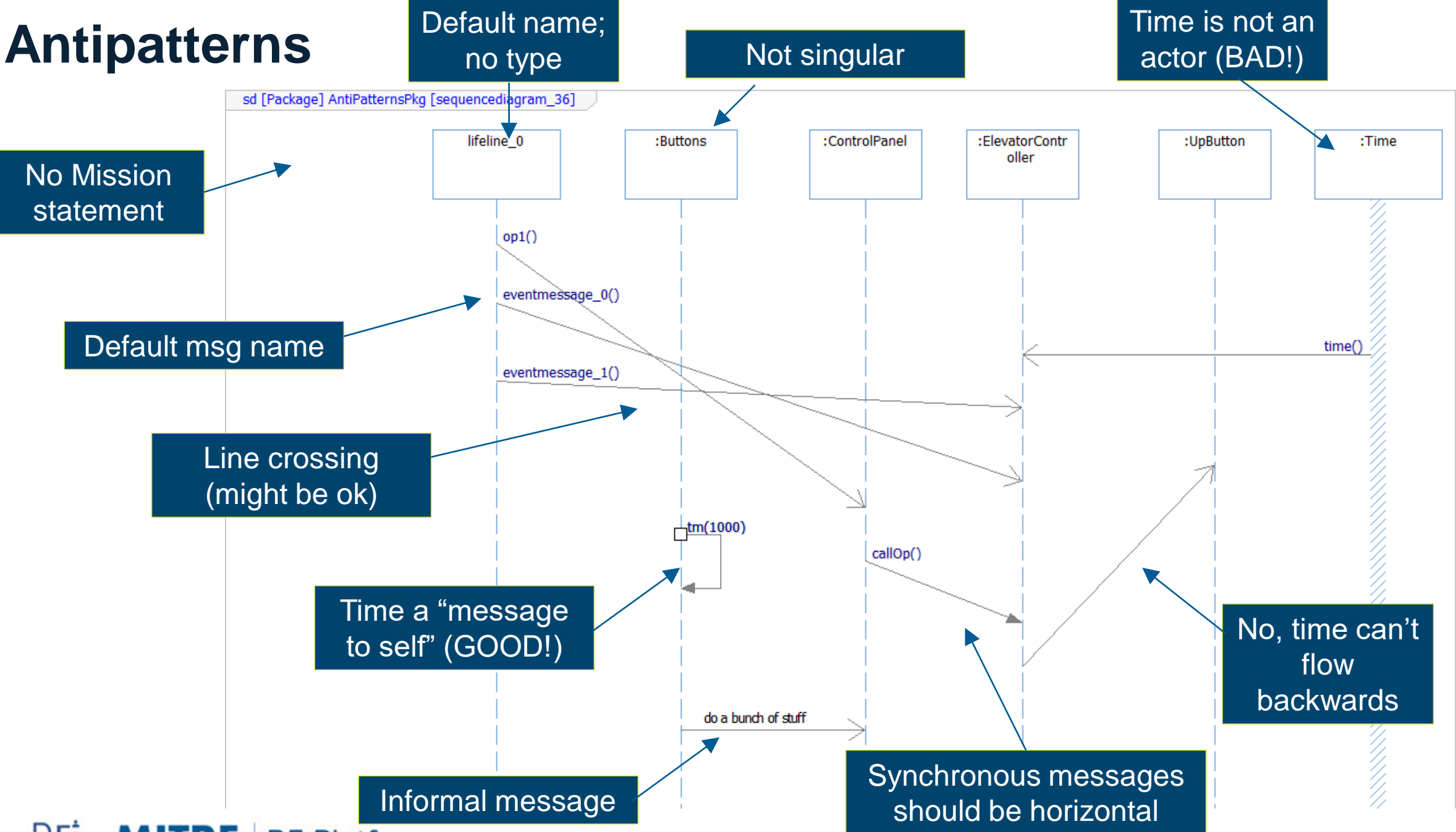
# What does parallel mean?

- A then B then C
- X then Y then Z
- But happens first?
  - A or X?
  - B or Z?
  - Z or A?
- Because A and B are in one parallel region, but X and Z are in another, we don't know.
- If we cared, then parallel regions was the wrong design choice!





# Antipatterns

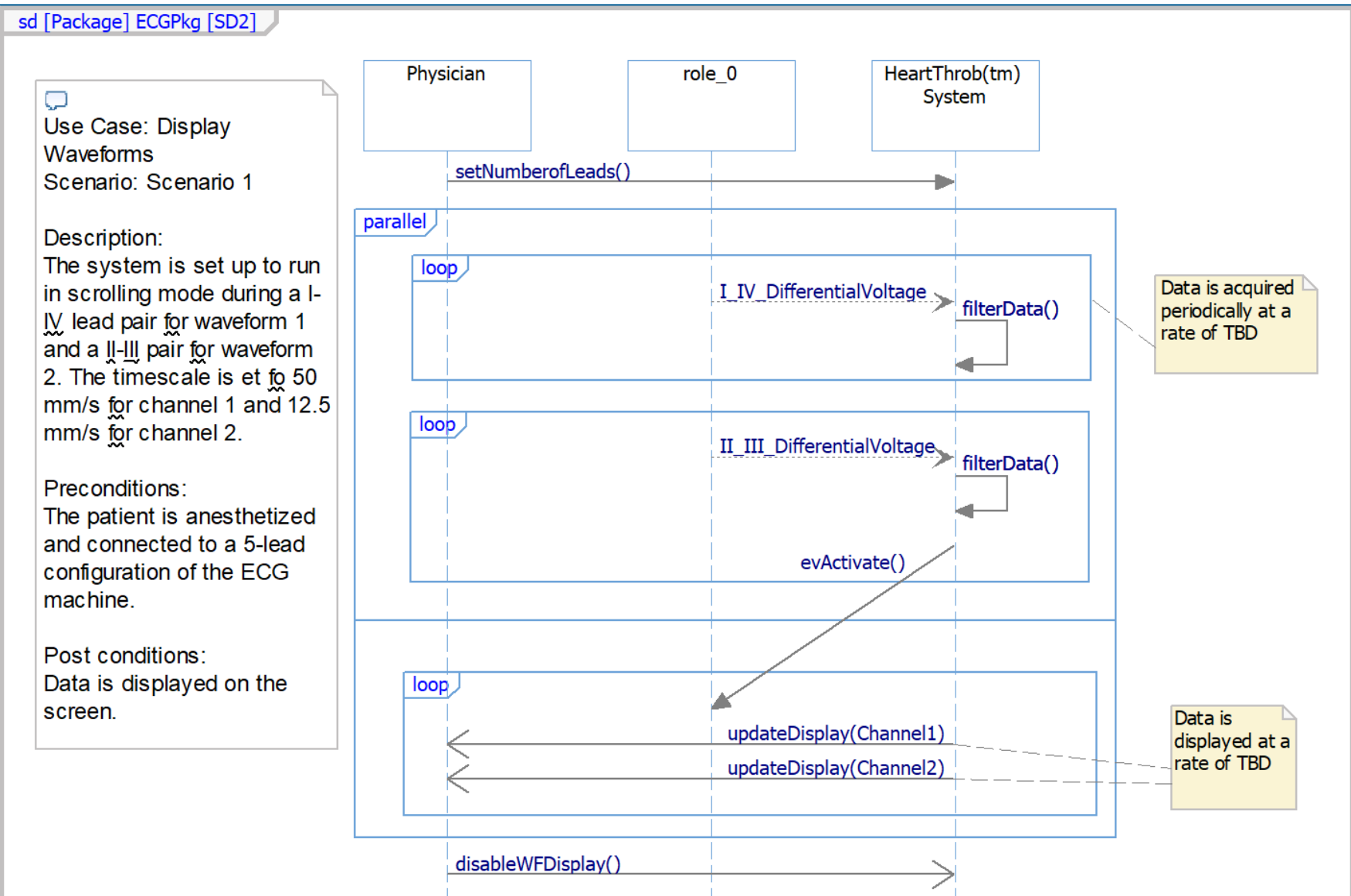




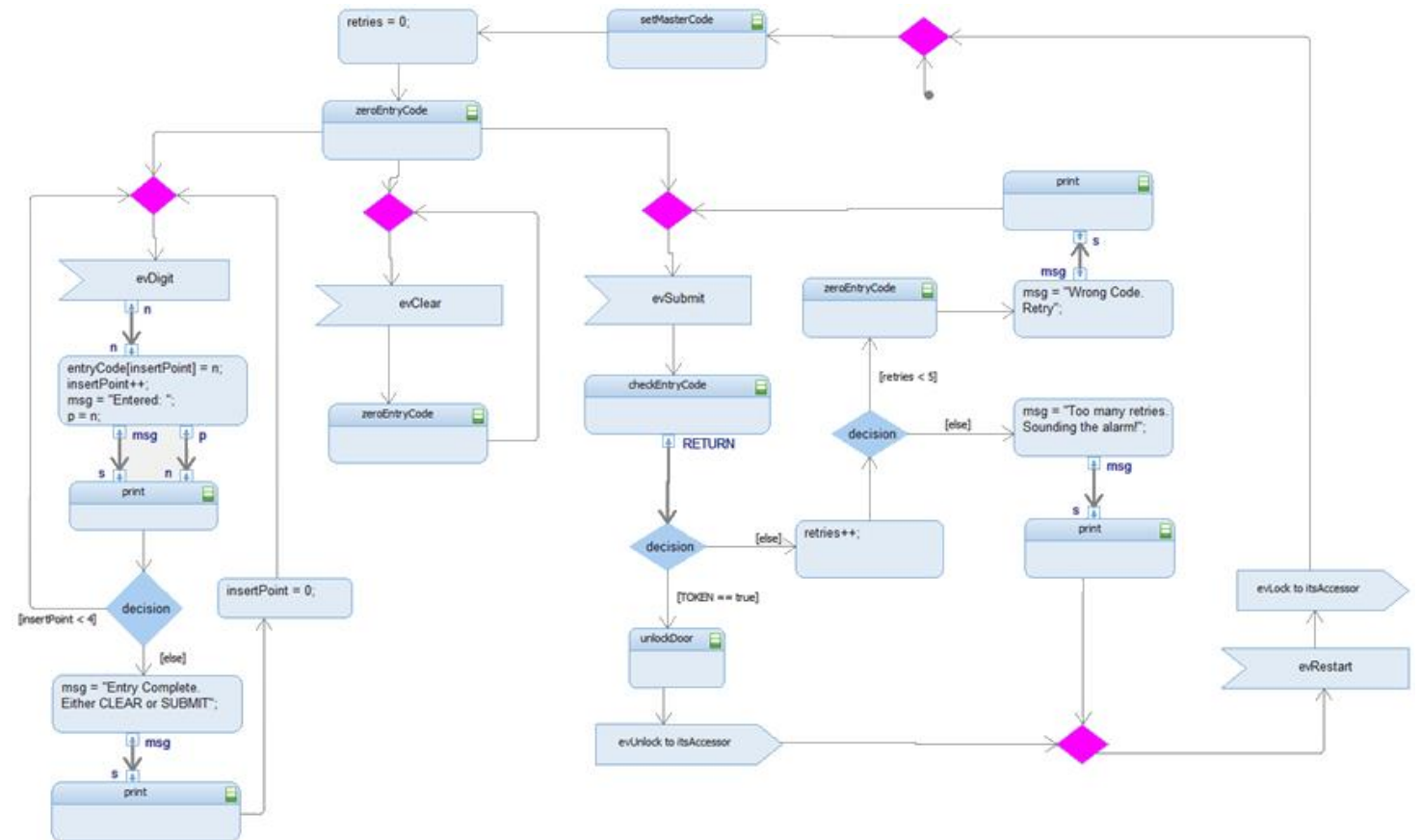
# Sequence Diagram Checklist

- ☐ Are the lifelines typed by blocks in the model?
- ☐ Is the sequence correct?
- ☐ Are the use of message types (synchronous vs asynchronous) appropriate?
- ☐ Do the messages include parameters and arguments, if appropriate?
- ☐ Is the purpose and scope of the diagram stated?
- ☐ Is the purpose and scope of the diagram adhered to?
- ☐ Does the sequence represent the appropriate level of abstraction?
- ☐ Does it have comments to aid in understanding the flow where needed?
- ☐ Are there too many levels of nesting of interaction operators on the diagram (recommended to limit to 3 levels on a single diagram)?
- ☐ Is message line crossing minimized?
- ☐ Are any interaction operators well used?

# Exercise: Evaluate the following Sequence Diagram



# Activity Diagrams

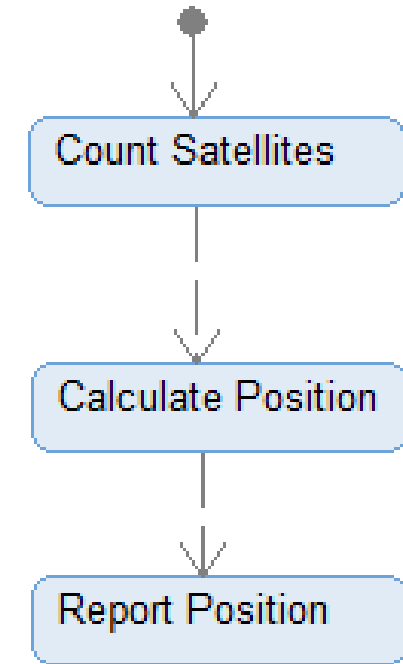


# Why Activity Modeling?

- Many behaviors are “flow of control” style in which an action is performed after the previous action ends, with possible decision and forking points
- Such modeling can
  - Define algorithmic behavior
  - Show how collaborations of structural elements interact in the course of executing a larger scale behavior
  - Allocate actions to different structural elements
  - Specify fully constructive behavior
  - (design) Can be used to generate algorithmic code
- Activities are an alternative to state-based behavioral modeling but can be combined with it

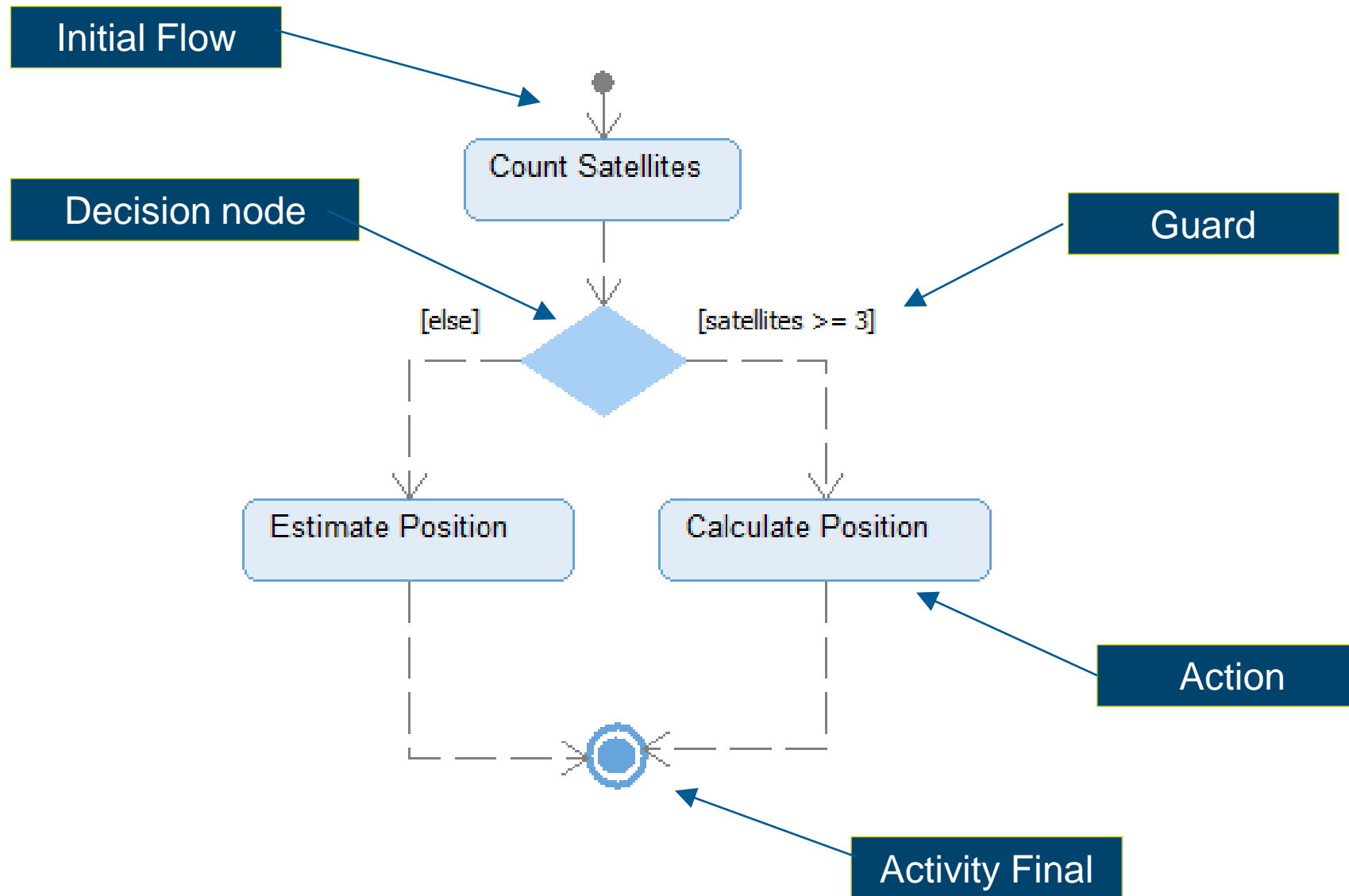
# Introduction

- Activity Modeling is all about flow of
  - Control (most common)
  - Data
  - Energy
  - Mass
  - Materiel
- Shows the sequencing of actions using control flows
- Execution proceeds via *token flow semantics*
  - *An action can begin where there is a token on all inputs*
  - *Once an action completes, it places a token on all outputs*

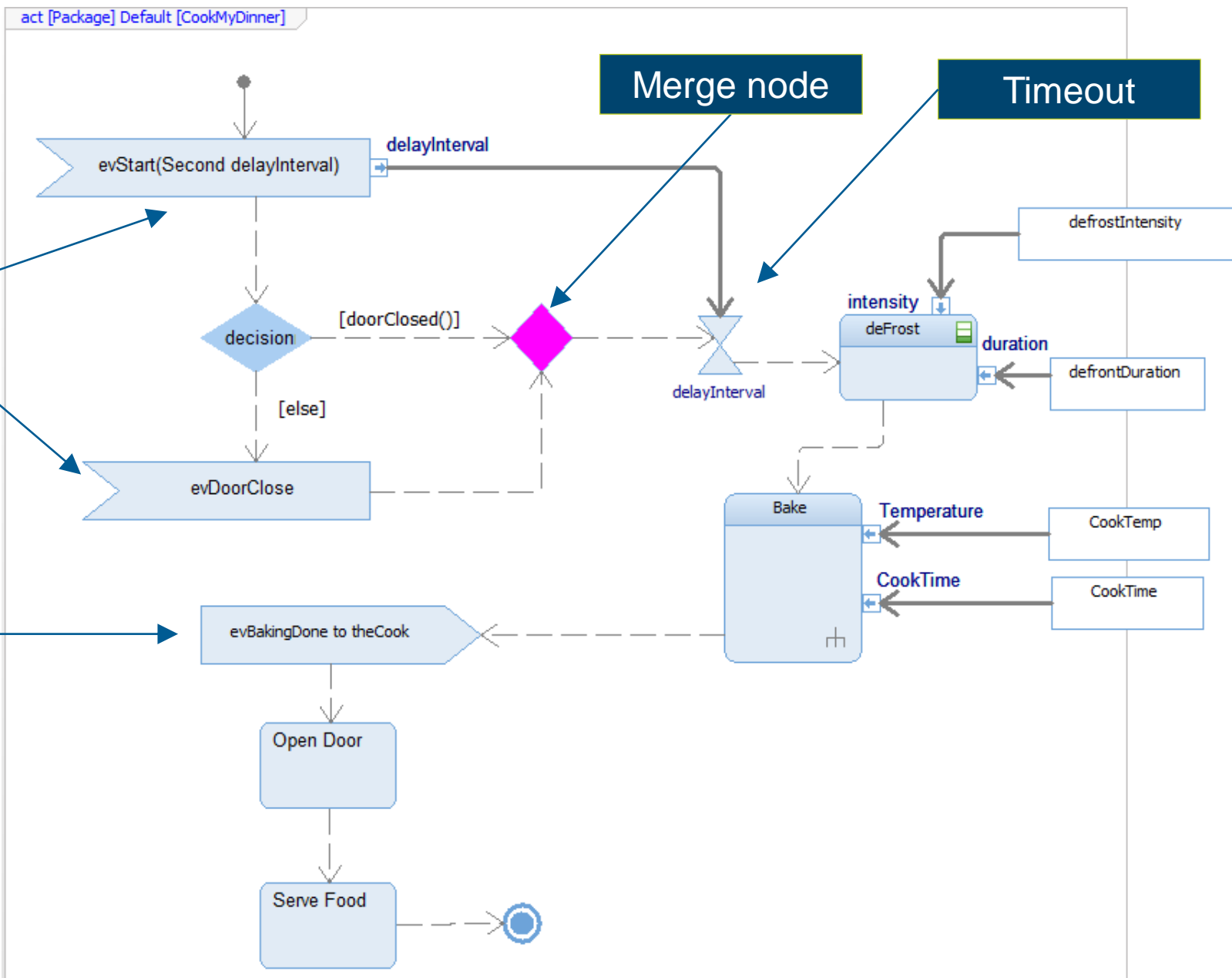


Activities are defined by sequences of actions (behaviors). This is importantly different from state machines which show the sequence of states (conditions) but execute behaviors along the way.

# Basic syntax – actions and flows

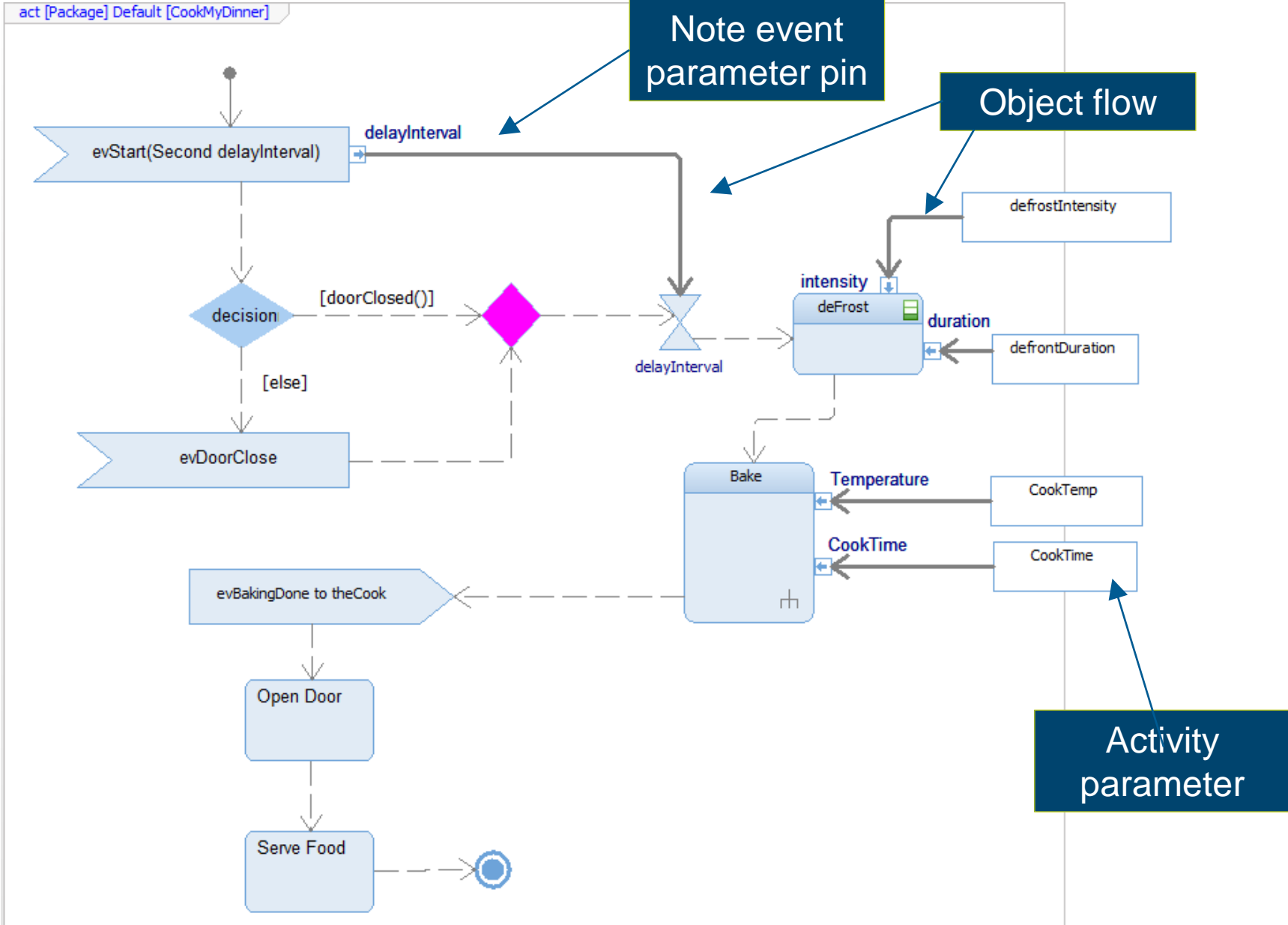


# Events



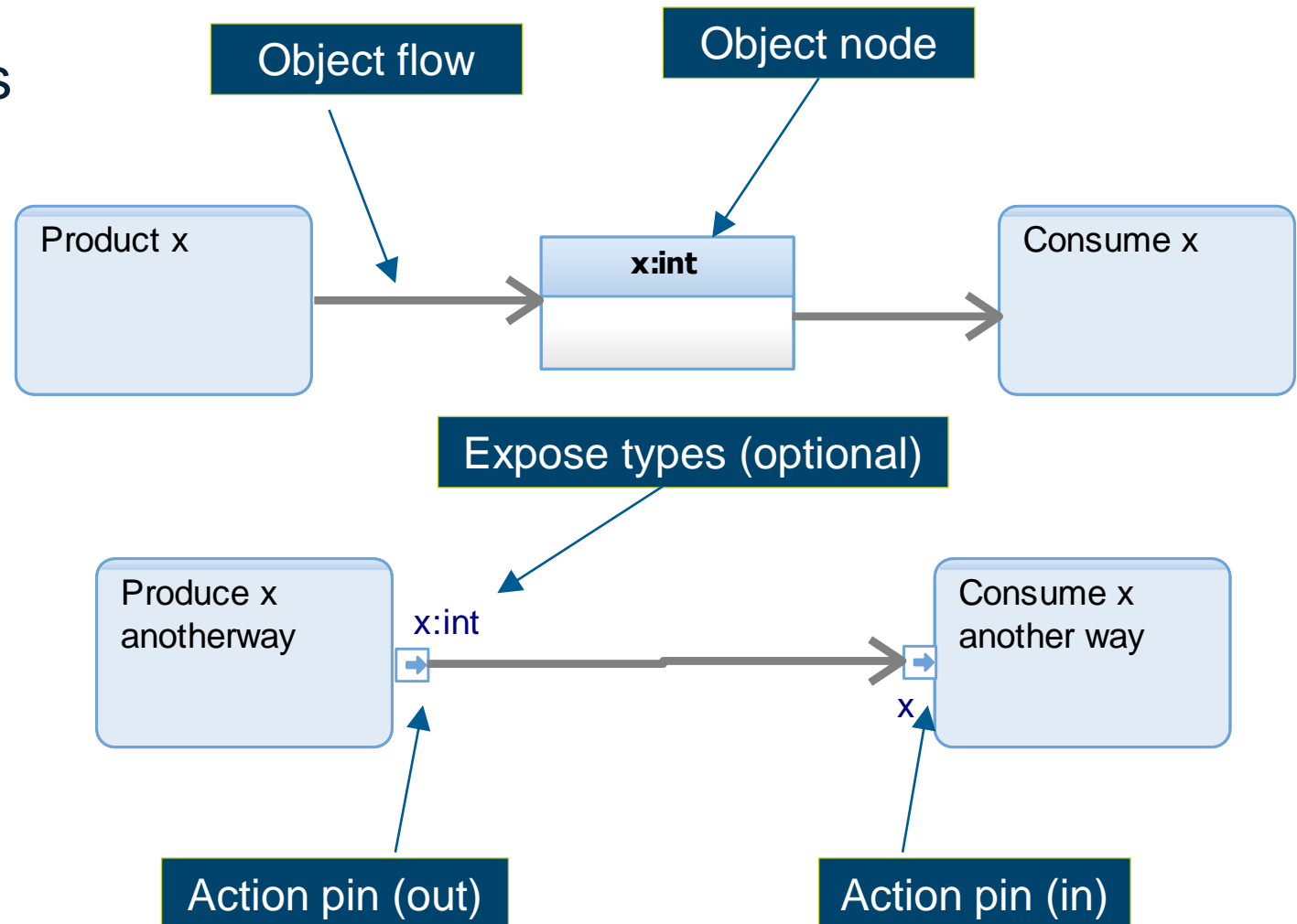


# Activity Parameters and Action Pins

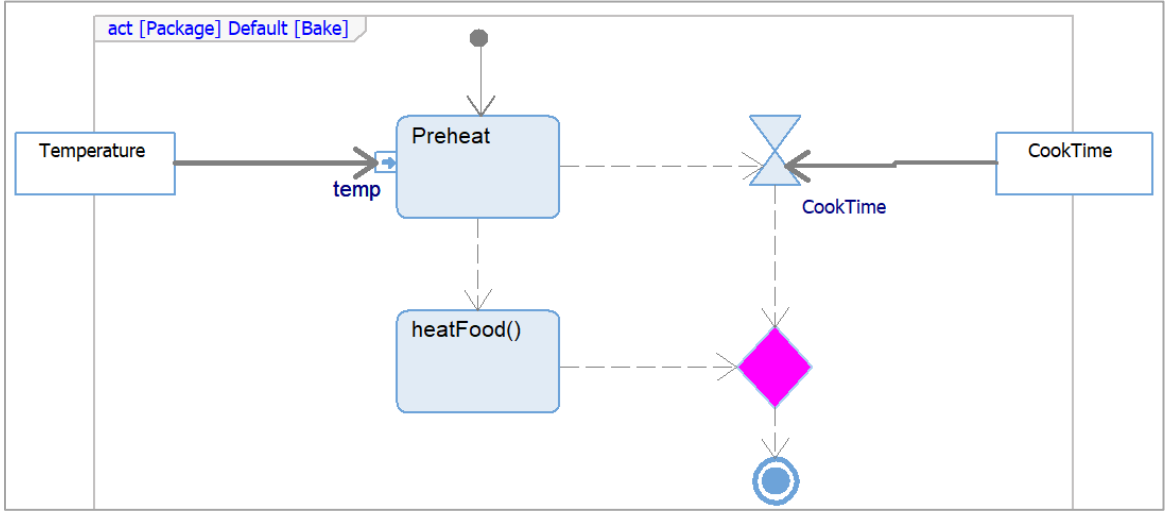
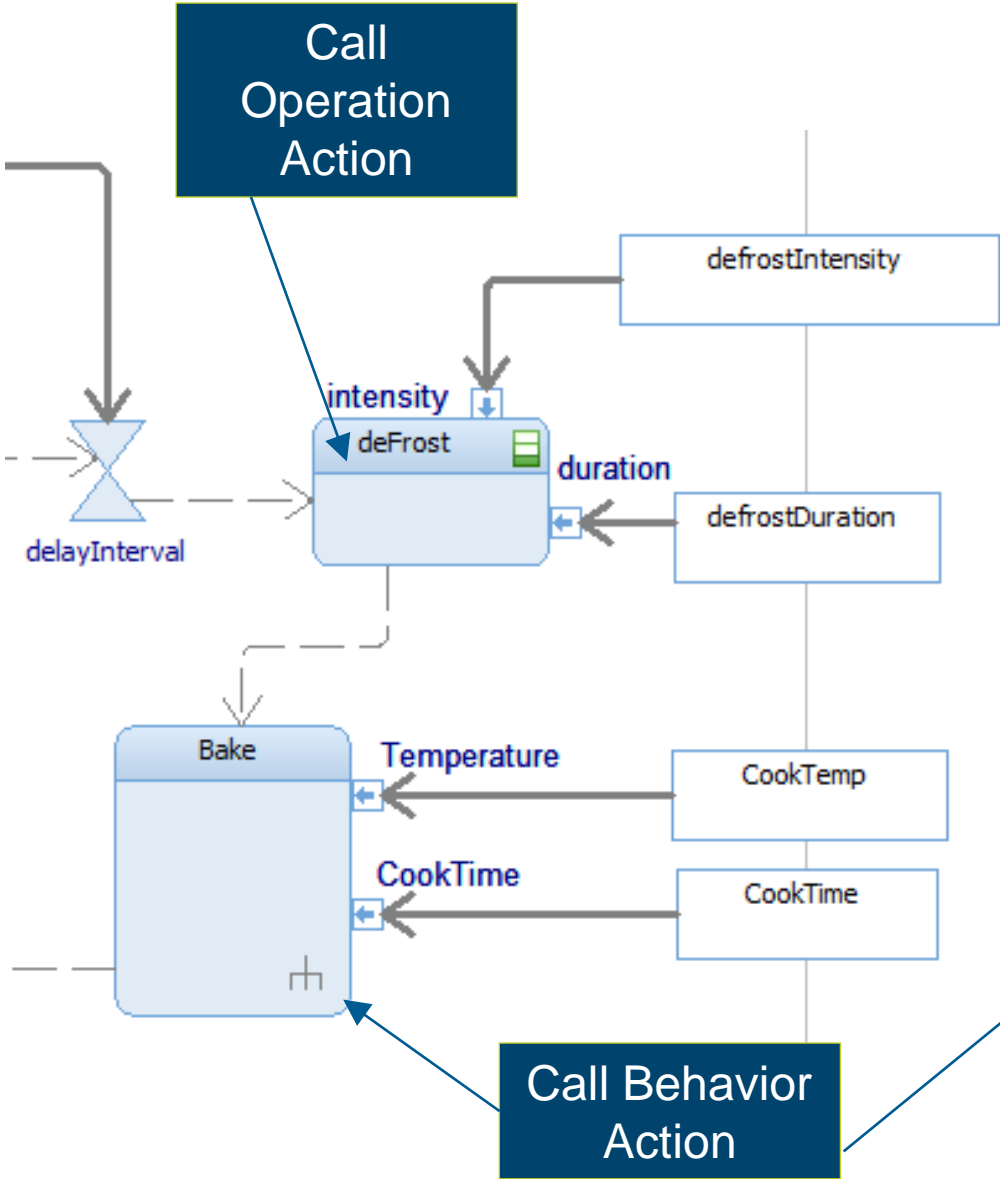


# Object flow

- Object token flow always implies a control token flow
- Object tokens are inherently queued because tokens shouldn't be lost
- These two different notations (object node vs action pins) mean exactly the same thing
- Pins should be type-compatible



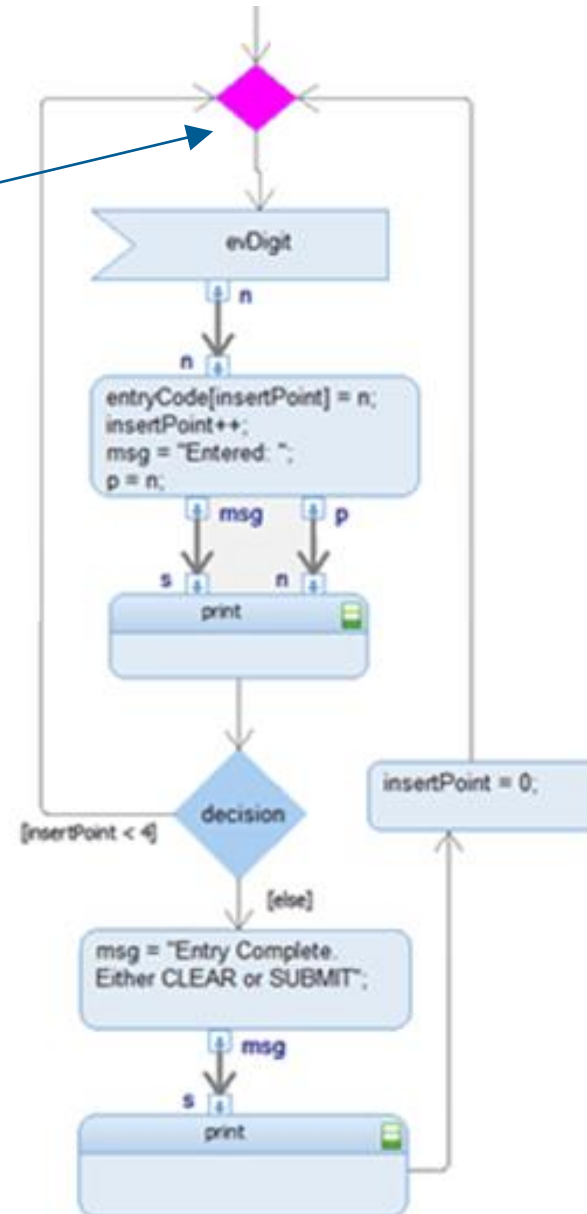
# Activity Parameters and Action Pins



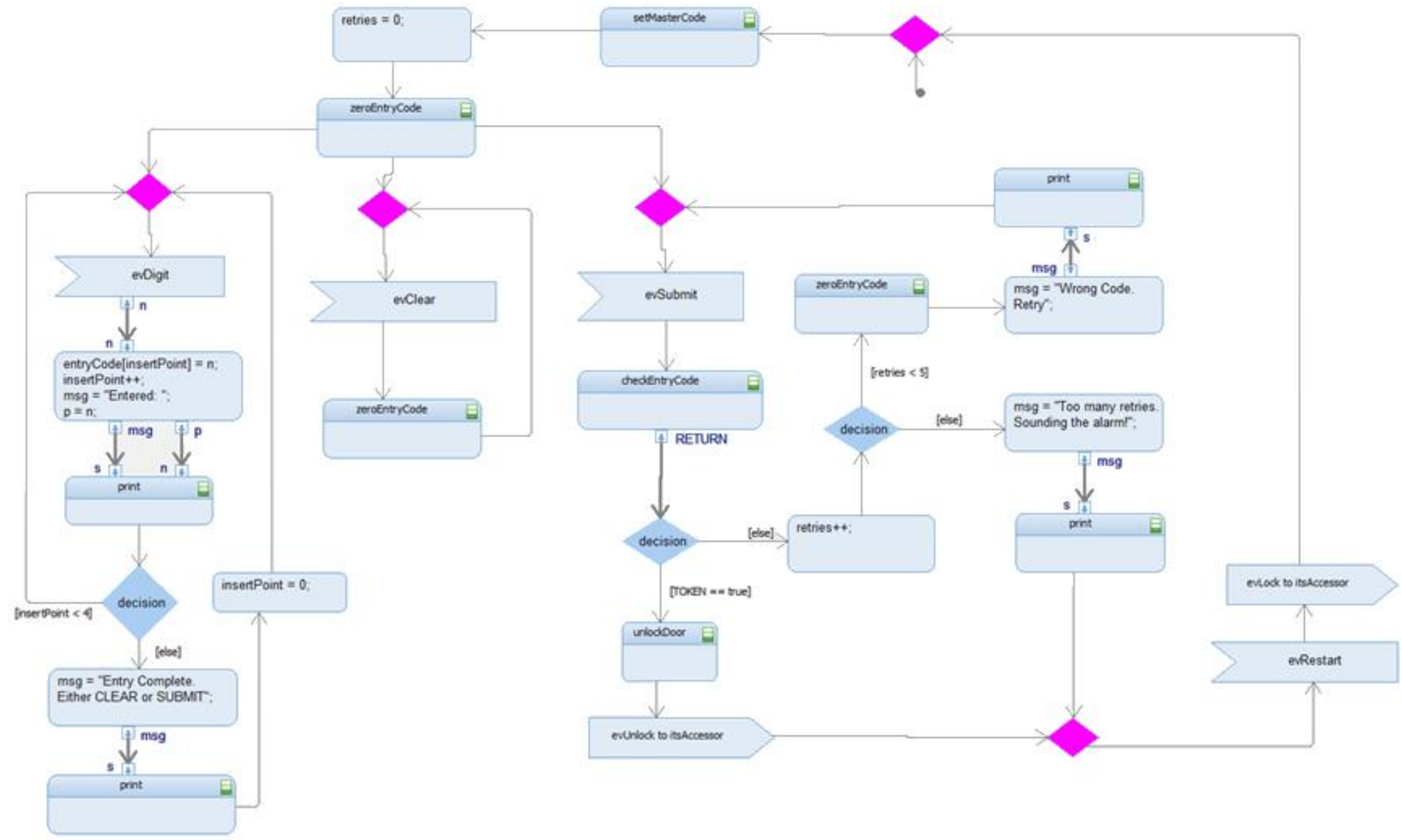
# Merge Node

Merge node

Merge is needed because control flows have ANDing semantics  
So this is a way of saying “continue here if ANY of these flows occurs”

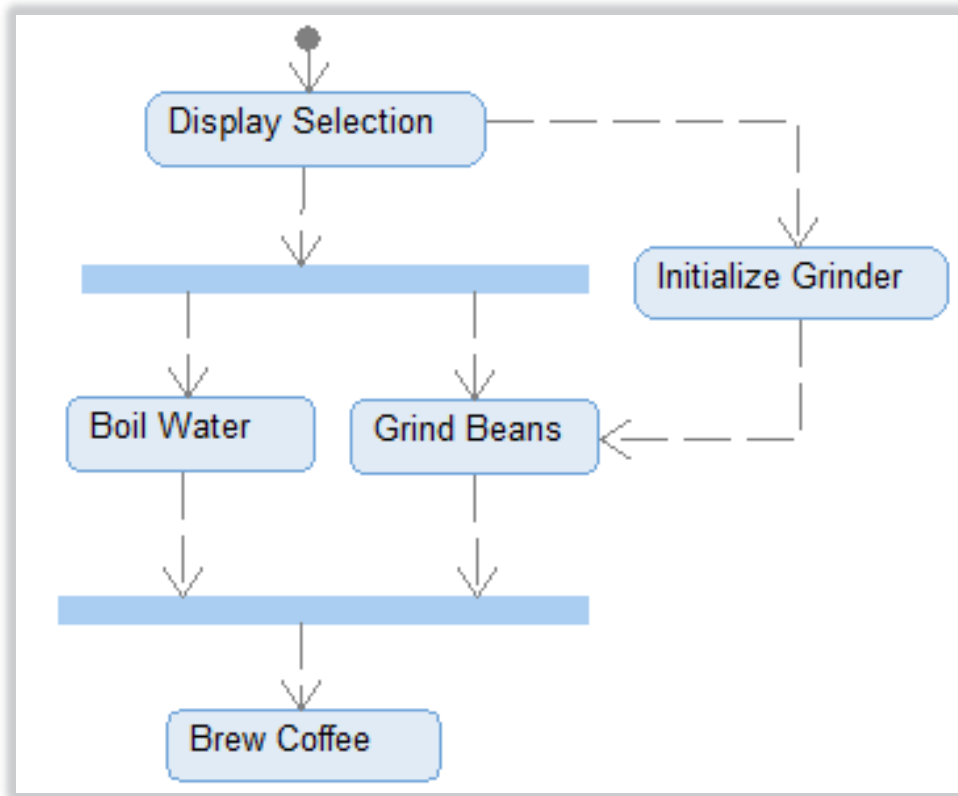


# Another Activity Diagram: Use Key Pad to Unlock Door

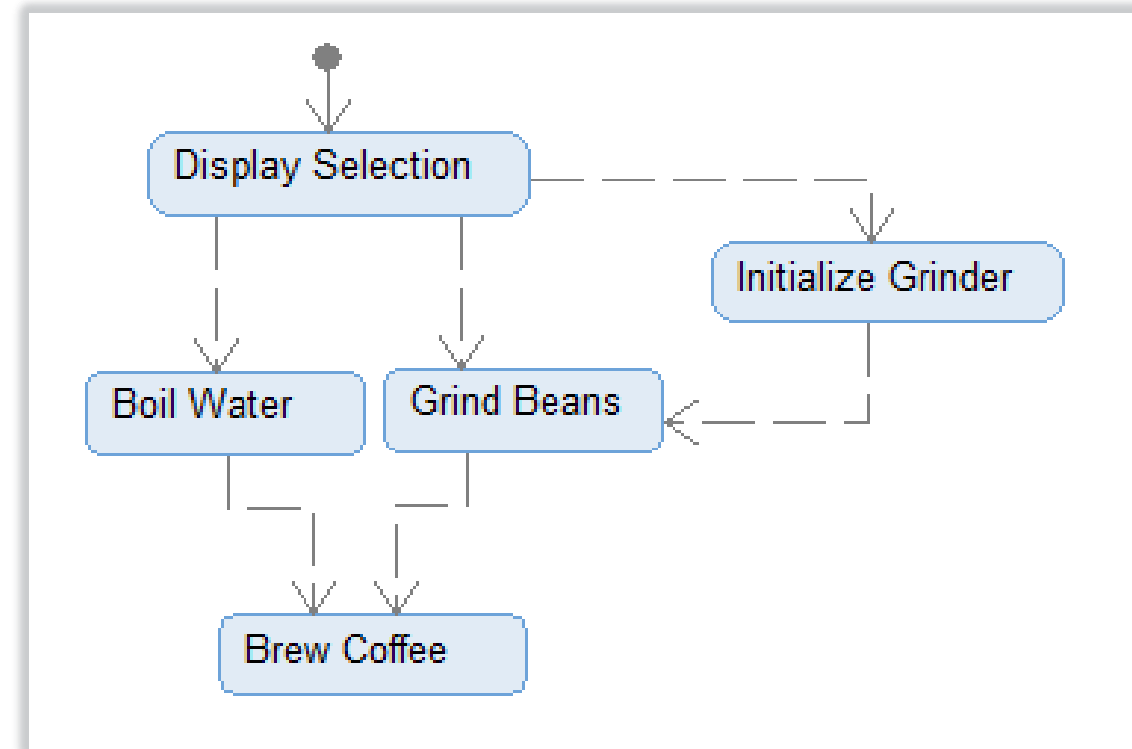


# Fork and Join

- Explicitly shows parallelism
- Not needed in most cases because of token execution semantics

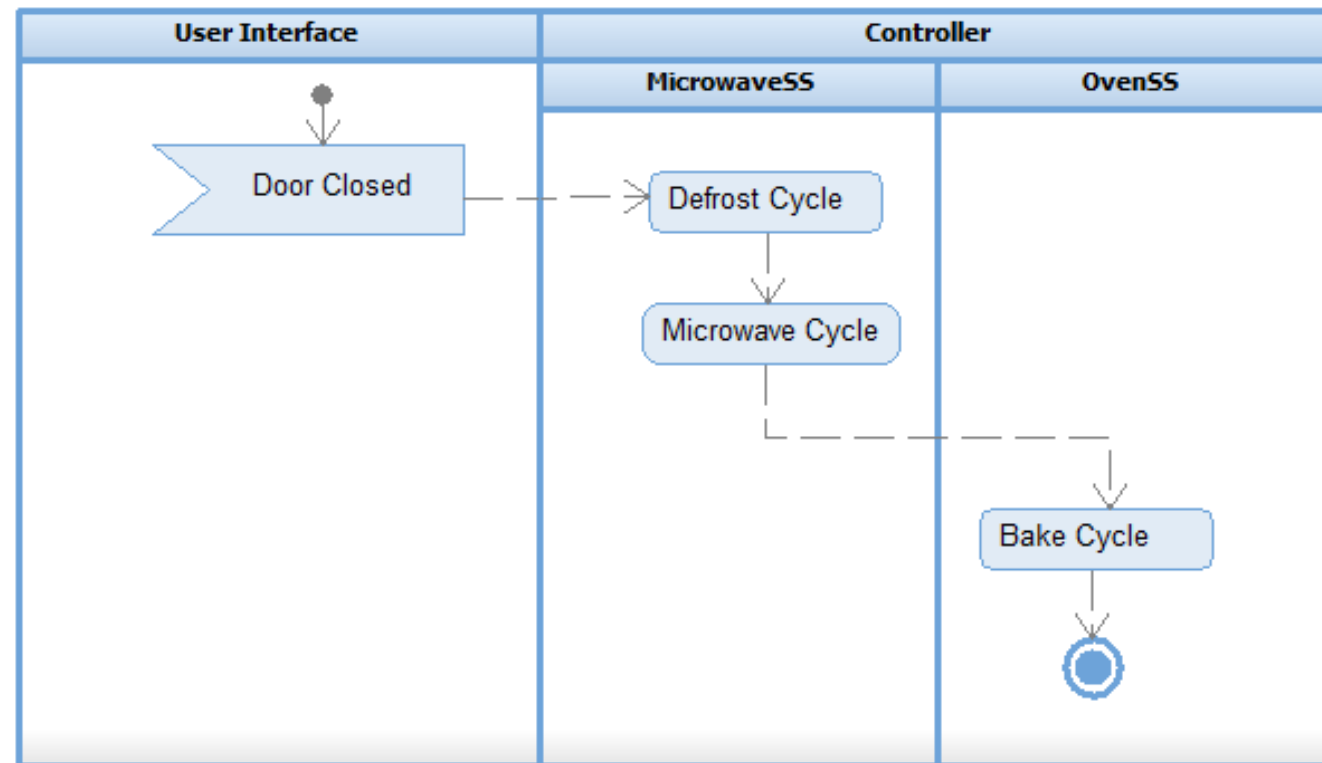


=

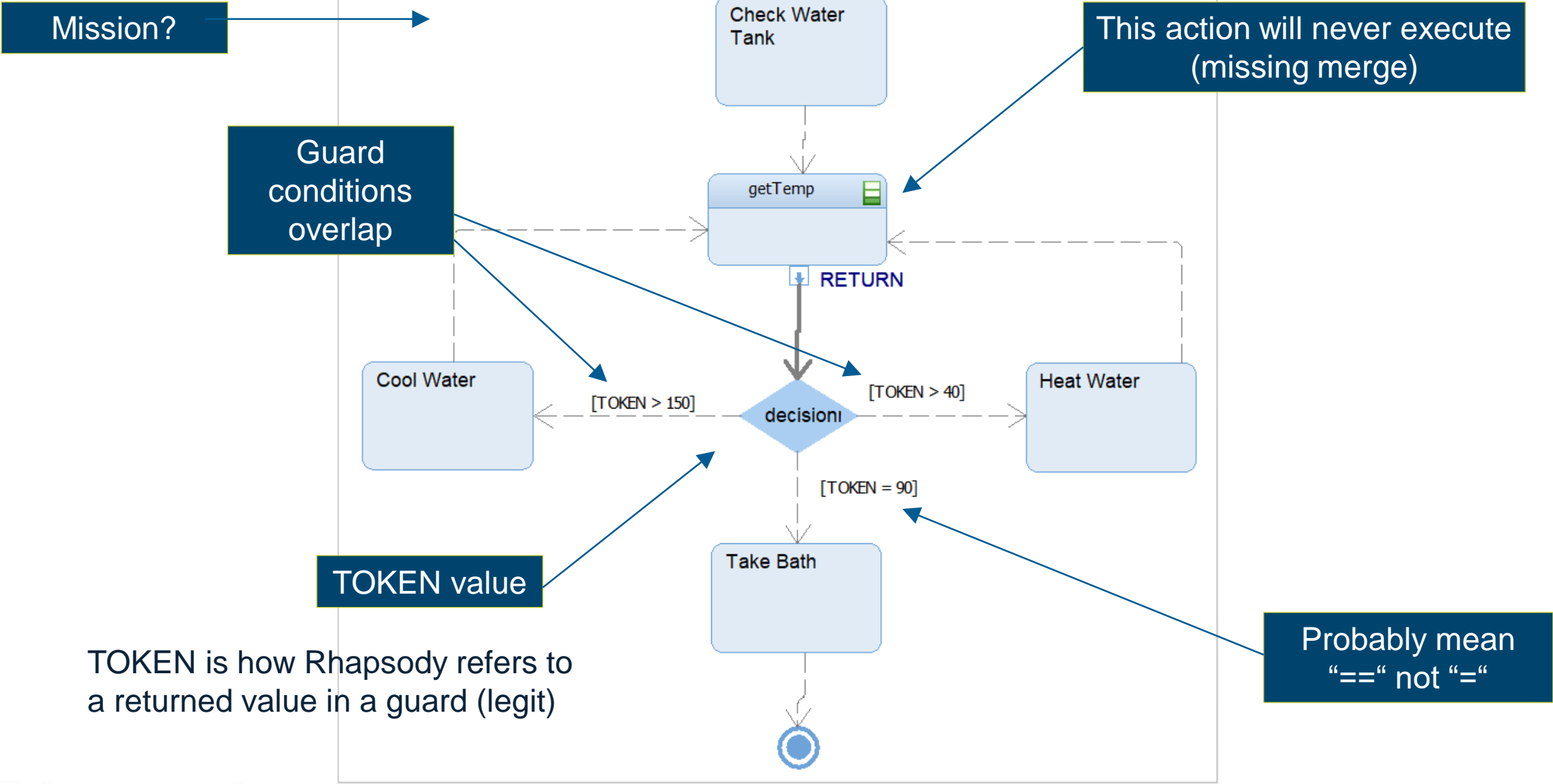


# Activity partitions (“Swimlanes”)

- Activities may be partitioned into swim lanes to show which block is responsible for executing the actions in the activity.



# Antipatterns



TOKEN is how Rhapsody refers to a returned value in a guard (legit)



# Activity Modeling: Key Takeaways



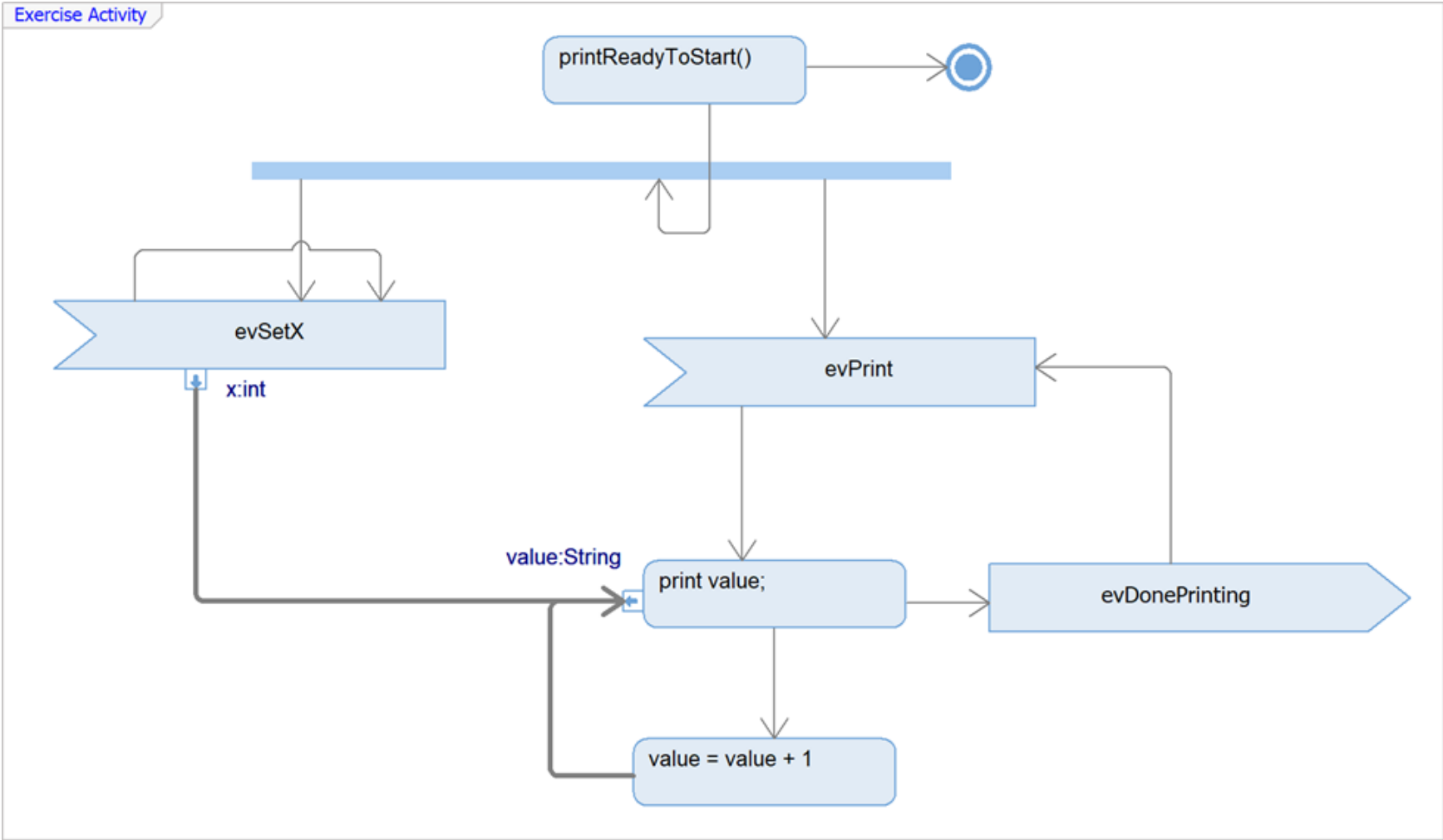
- **Activities** can represent the behavior of
  - A single block
  - A single function or operations
  - A collection of blocks
- **Actions** are run-to-completion behaviors which are connected by control flows within an activity
- **Data** is represented either with pins on actions (or parameters on activity diagrams) or by object nodes
- **Special actions** include
  - **Event Reception** actions
  - **Event Send** actions
  - **Timeout** actions
  - **Call Operation** actions (call a named function)
  - **Call Behavior** actions (invoke another activity diagram)



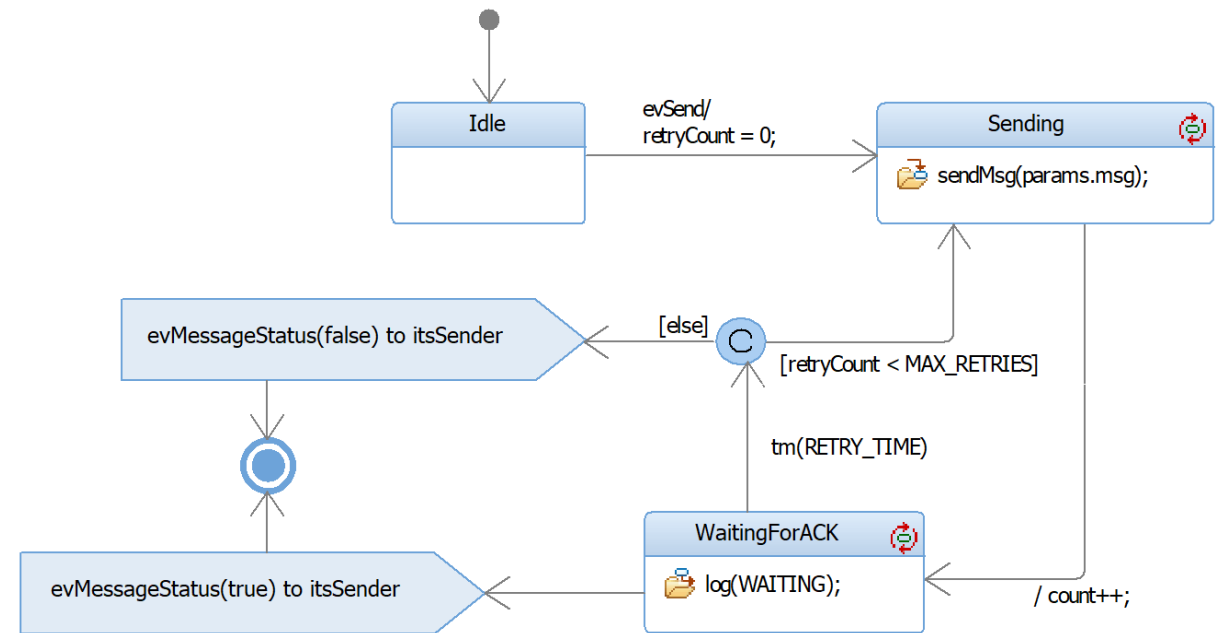
# Activity Diagram Checklist

- ☐ Is there an initial (default) control flow?
- ☐ Are all actions connected by a control or object flow?
- ☐ Is there a terminal connector?
- ☐ Is line crossing minimized?
- ☐ Is the purpose of the diagram properly specified?
- ☐ Are parallel flows independent in terms of order of actions?
- ☐ Is the action language used appropriate for the intent?
  - ☐ Natural language to capture conceptual flow
  - ☐ Action language for specific and/or executable flow
- ☐ Does the activity model execute?
- ☐ Are swim lanes used appropriately?
  - ☐ Do they imply allocation of functionality
- ☐ Are race conditions avoided?
- ☐ Does the activity diagram execute?
- ☐ Is it clear how the correctness of the activity diagram is verified?

# Exercise: Evaluate the following Activity Diagram



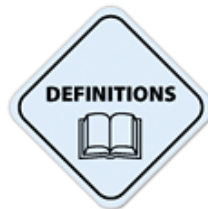
# State Diagrams



# State Machines



A **state** is a *distinguishable, disjoint, orthogonal condition of existence of an object that persists for a significant period of time*



A **transition** is a response to an event of interest moving the object from a state to a state.

# State Machines



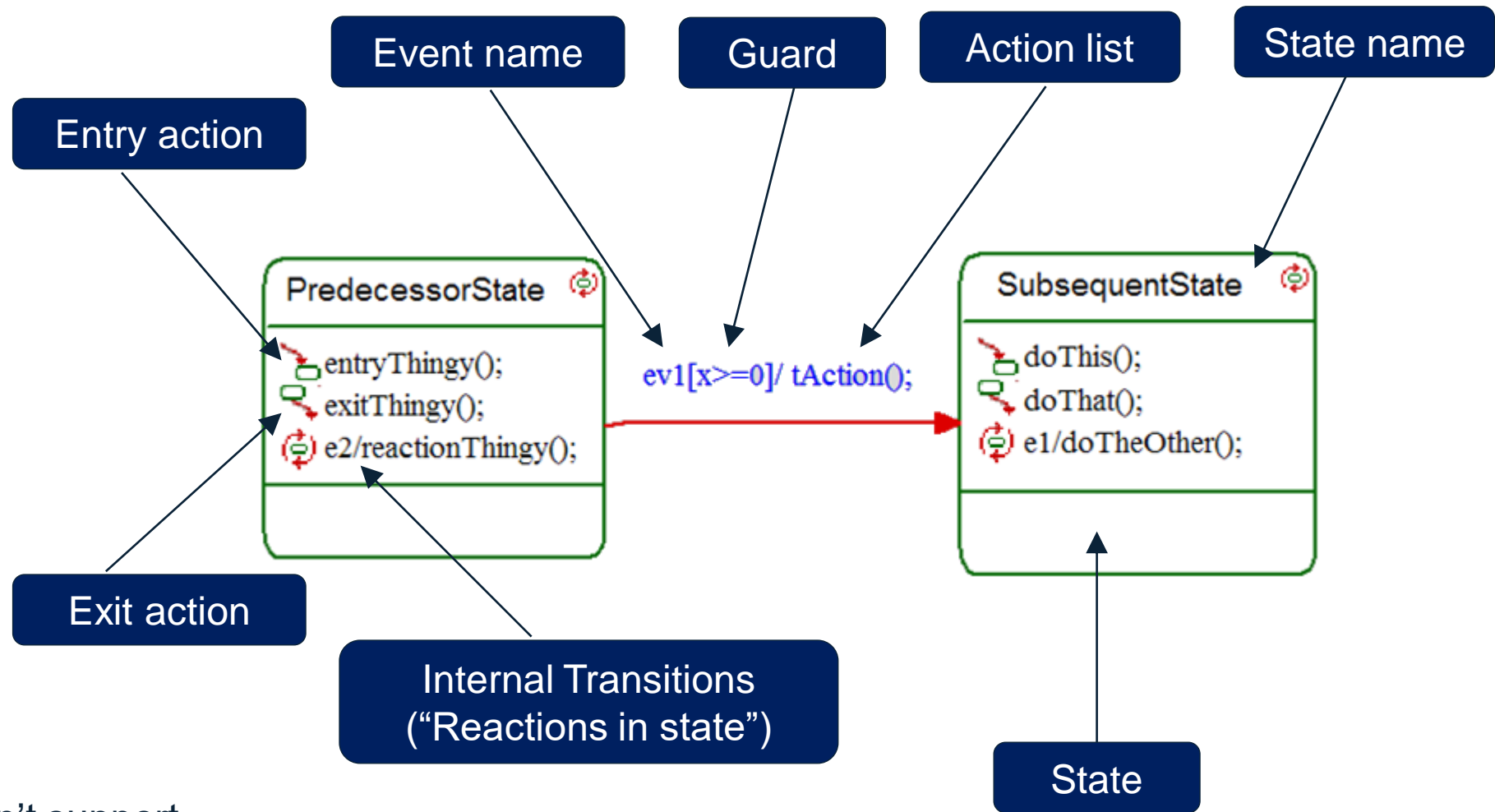
An **action** is a run-to-completion behavior. The object will not accept or process any new events until the actions associated with the current event are complete.



Order of action execution

- (1) exit actions of current state
- (2) transition actions
- (3) entry actions of next state

# Basic State Machine Syntax (Rhapsody)



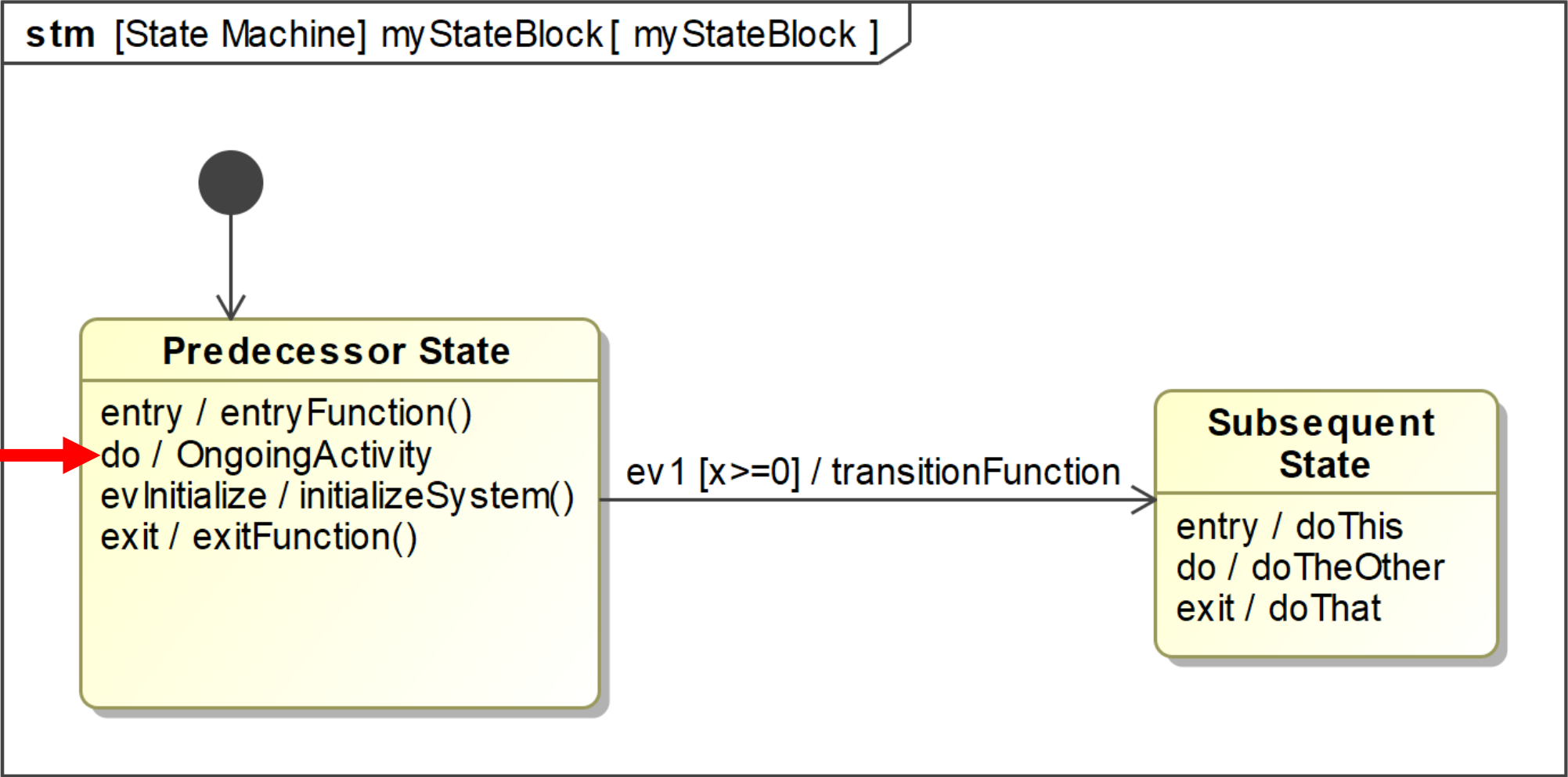
Rhapsody doesn't support "do activity"

# Basic State Machine Syntax (Cameo MagicDraw)

**Do activities** are behaviors that

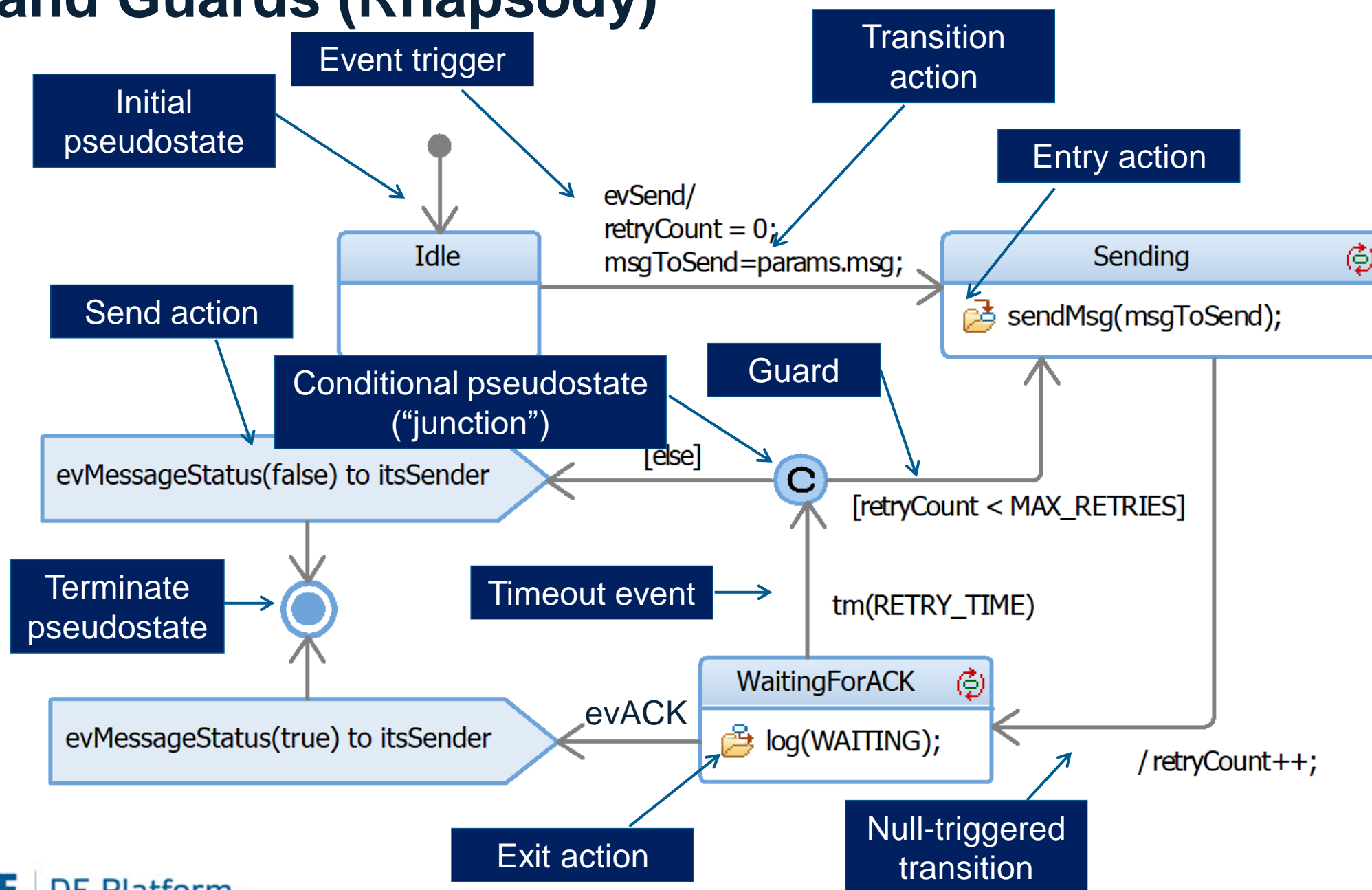
- 1. Start after the state is entered
- 2. Are interruptible by incoming events to the part (interruptible between the actions on the activity)

Note all the other behaviors are *non-interruptible* by incoming events.

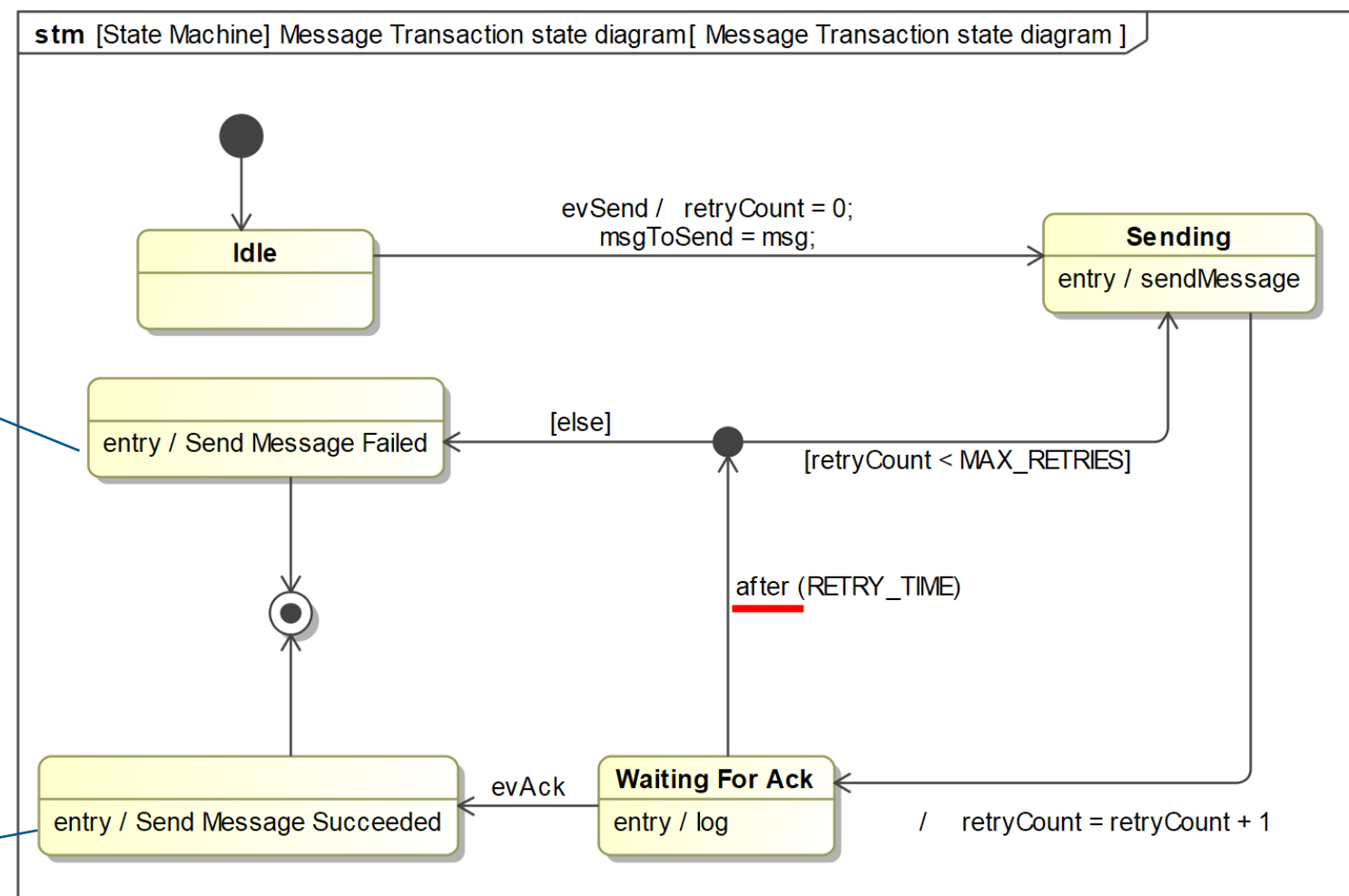
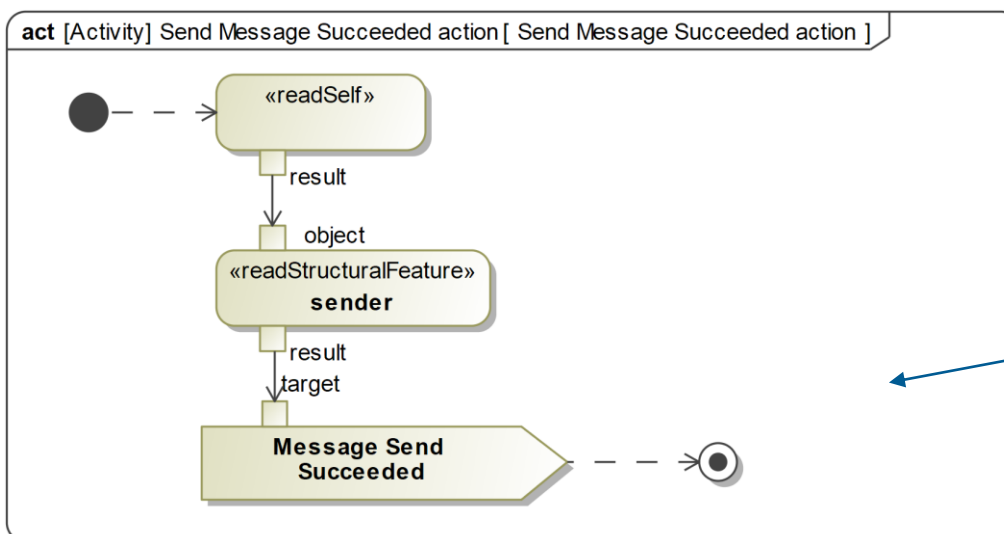
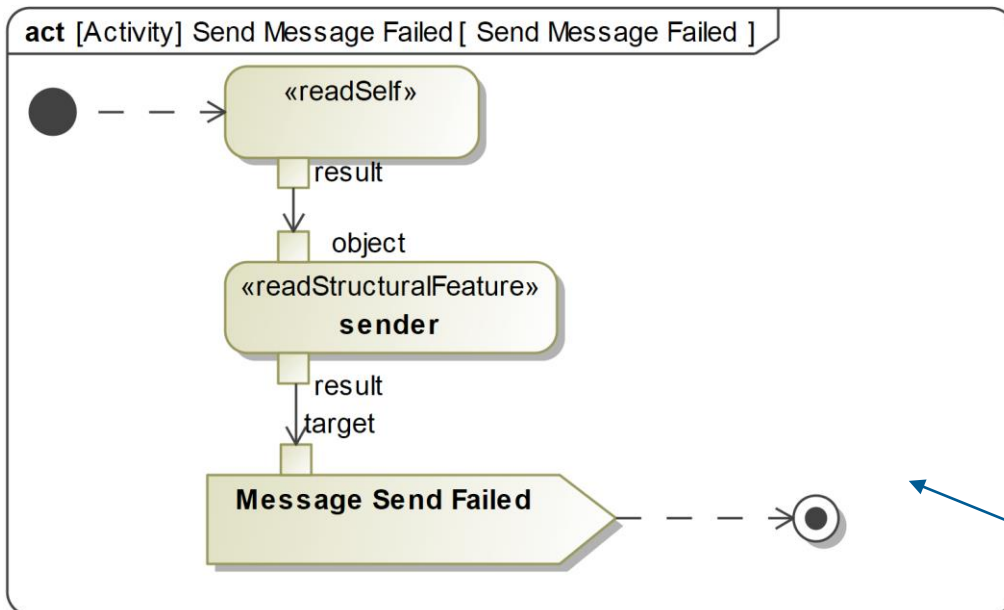




# Triggers and Guards (Rhapsody)

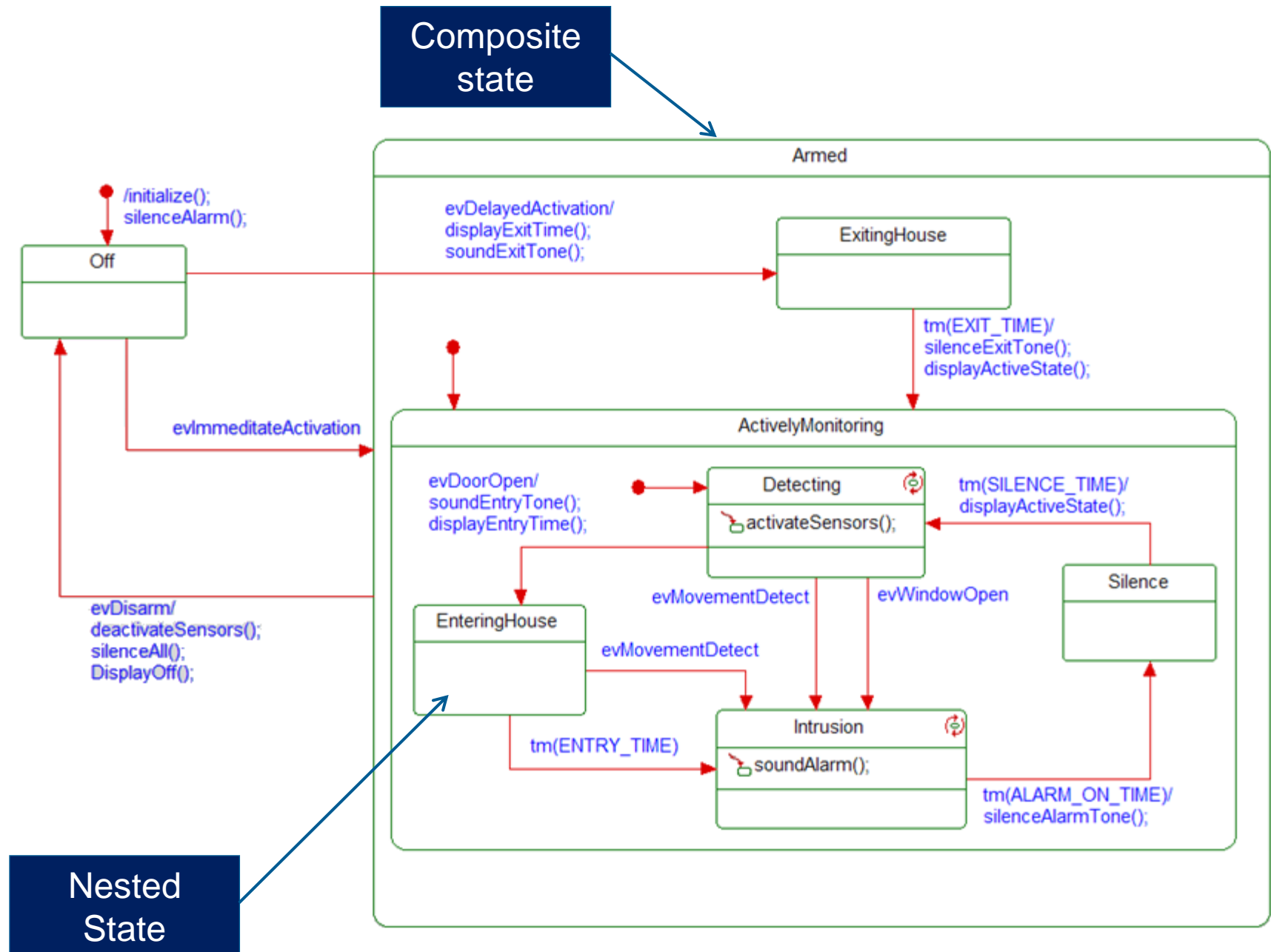


# Triggers and Guards (Cameo MagicDraw)

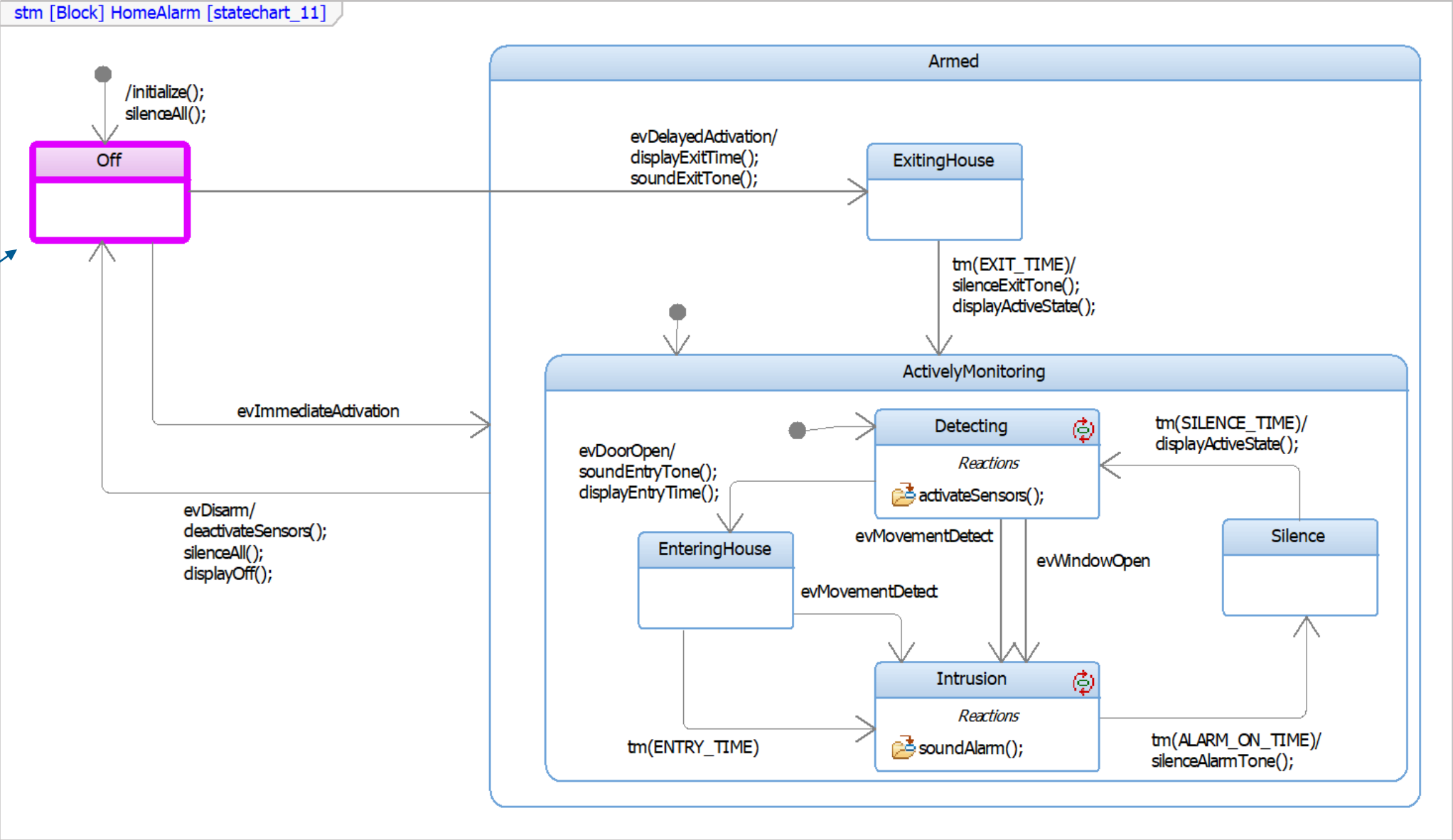


# Nested States

- States may be nested inside composite states
  - Transitions to the composite state go to the default nested states (unless a specific nested state is specified)
  - Transitions leaving the composite state apply to all nested states
- Nested states should be used only when it simplifies the overall model

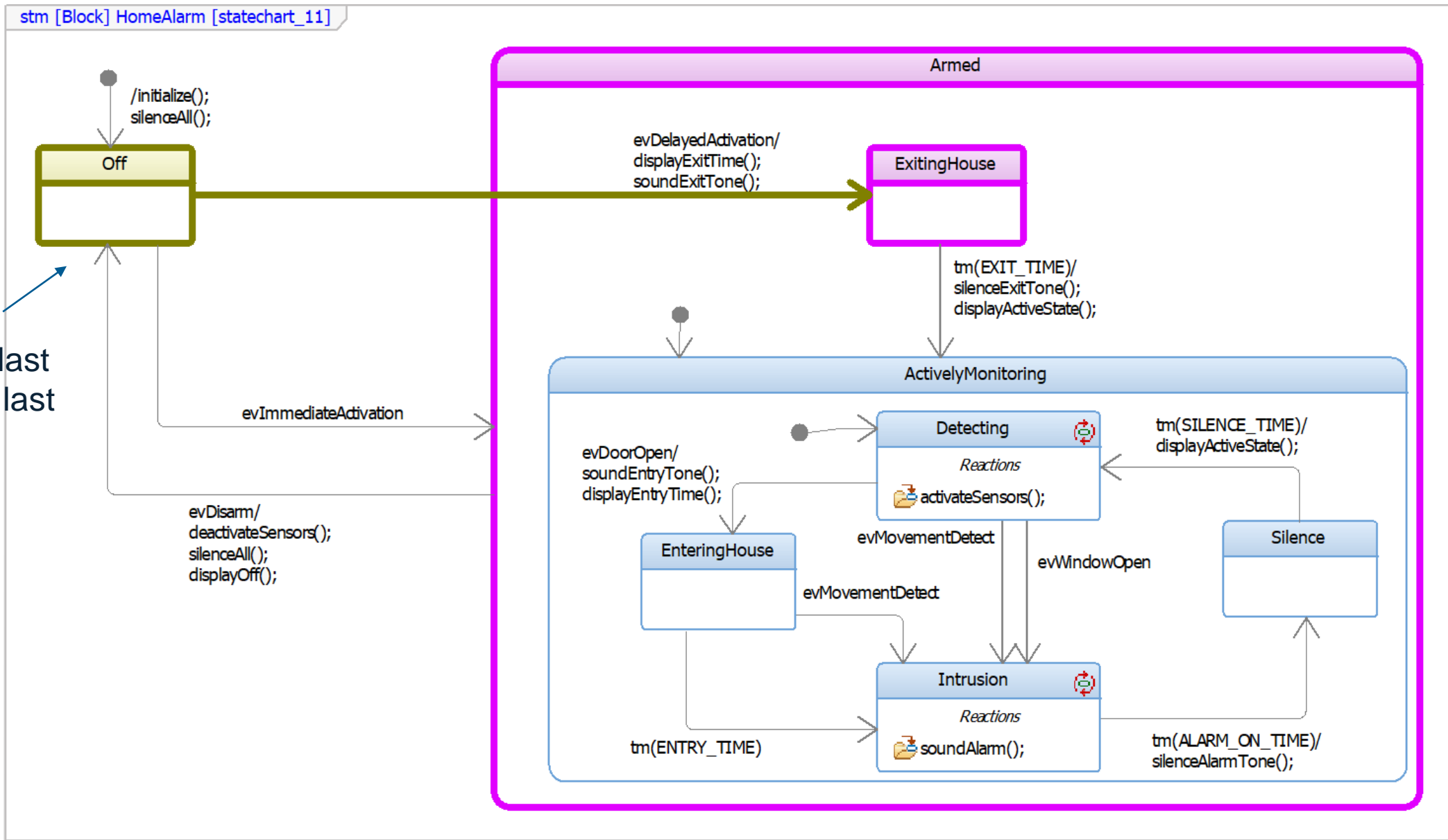


# Let's see how it works: Scenario 1 Step 1



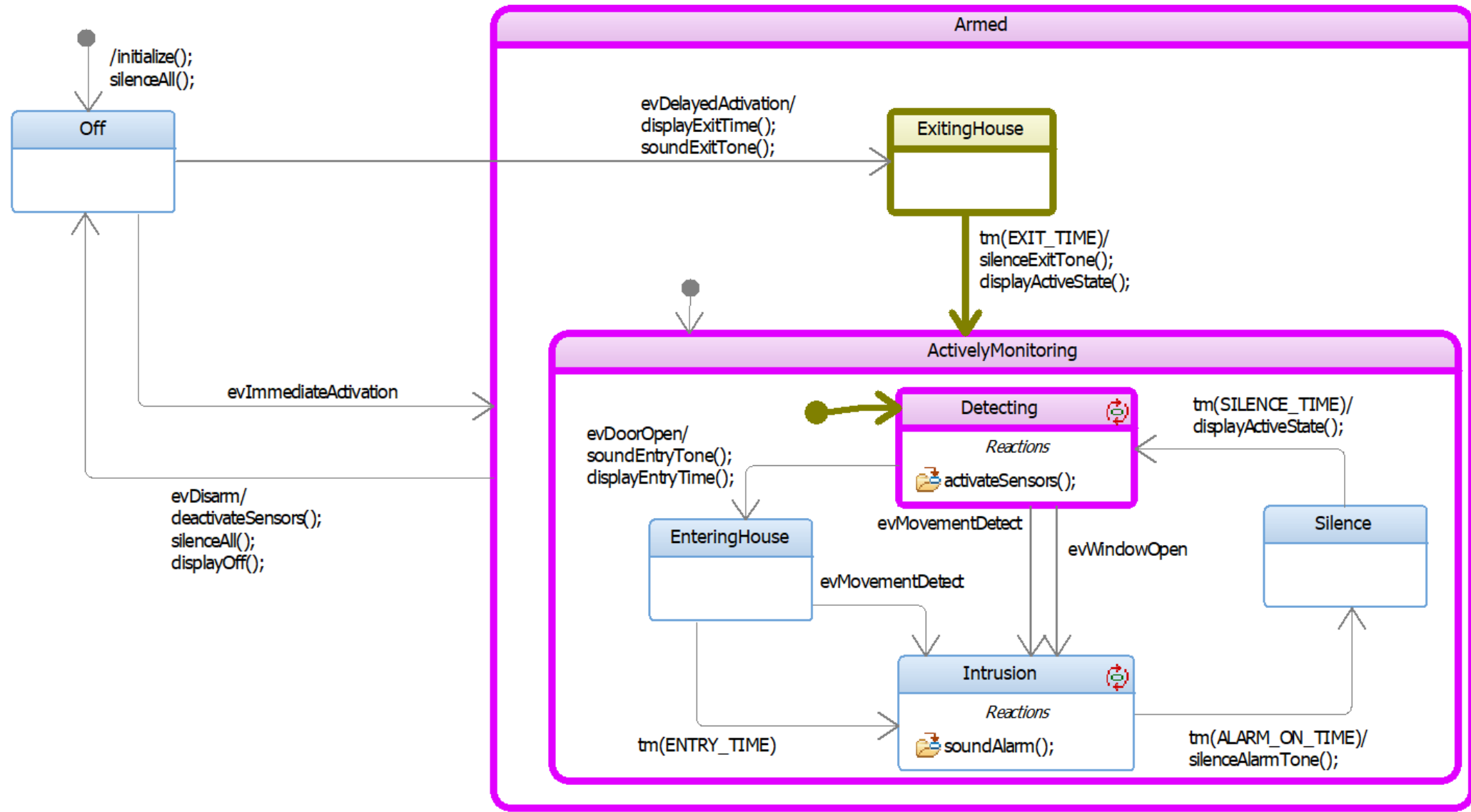
# Scenario 1 Step 2

Green indicates last state and last transition taken

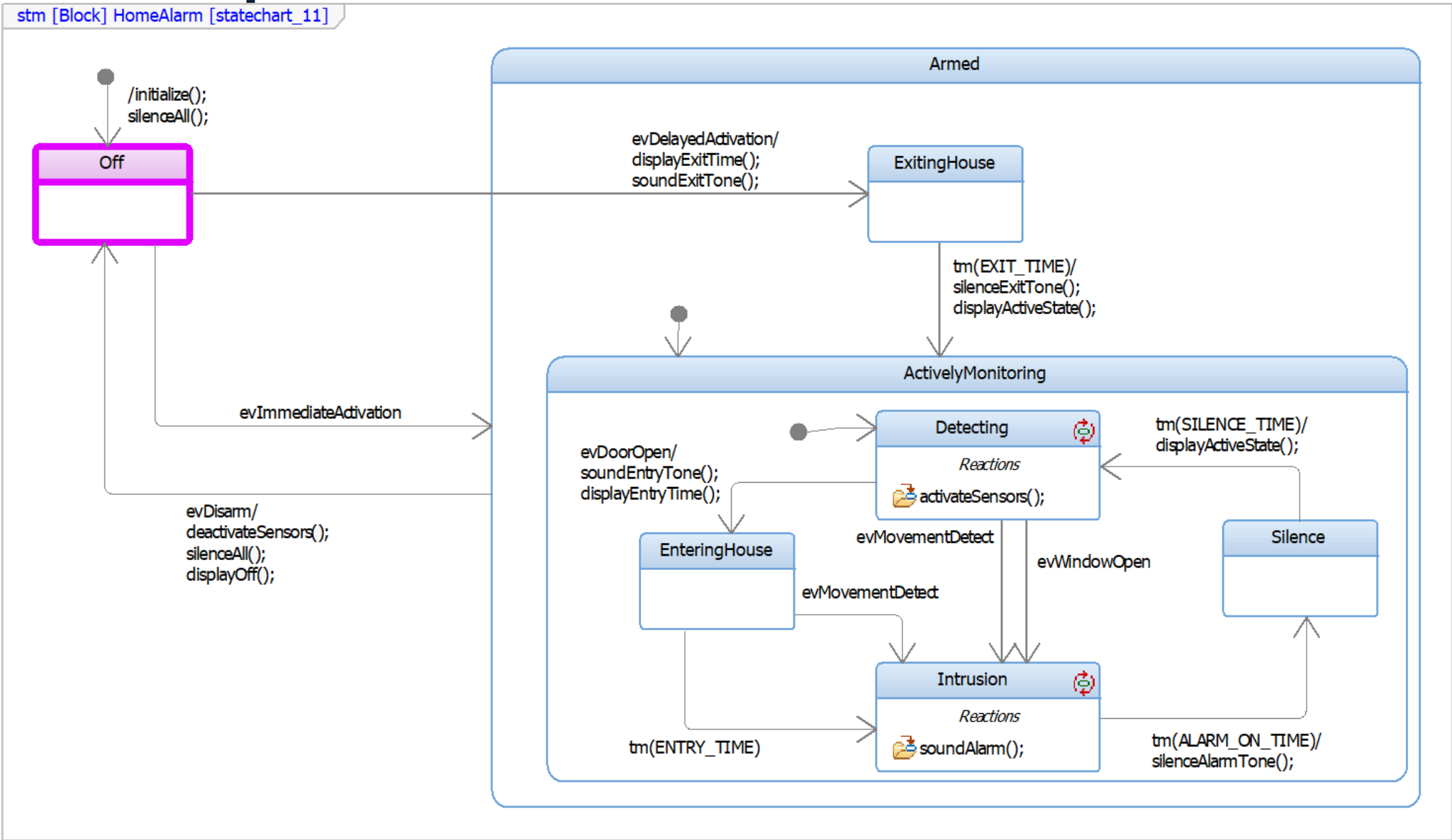


# Scenario 1 Step 3

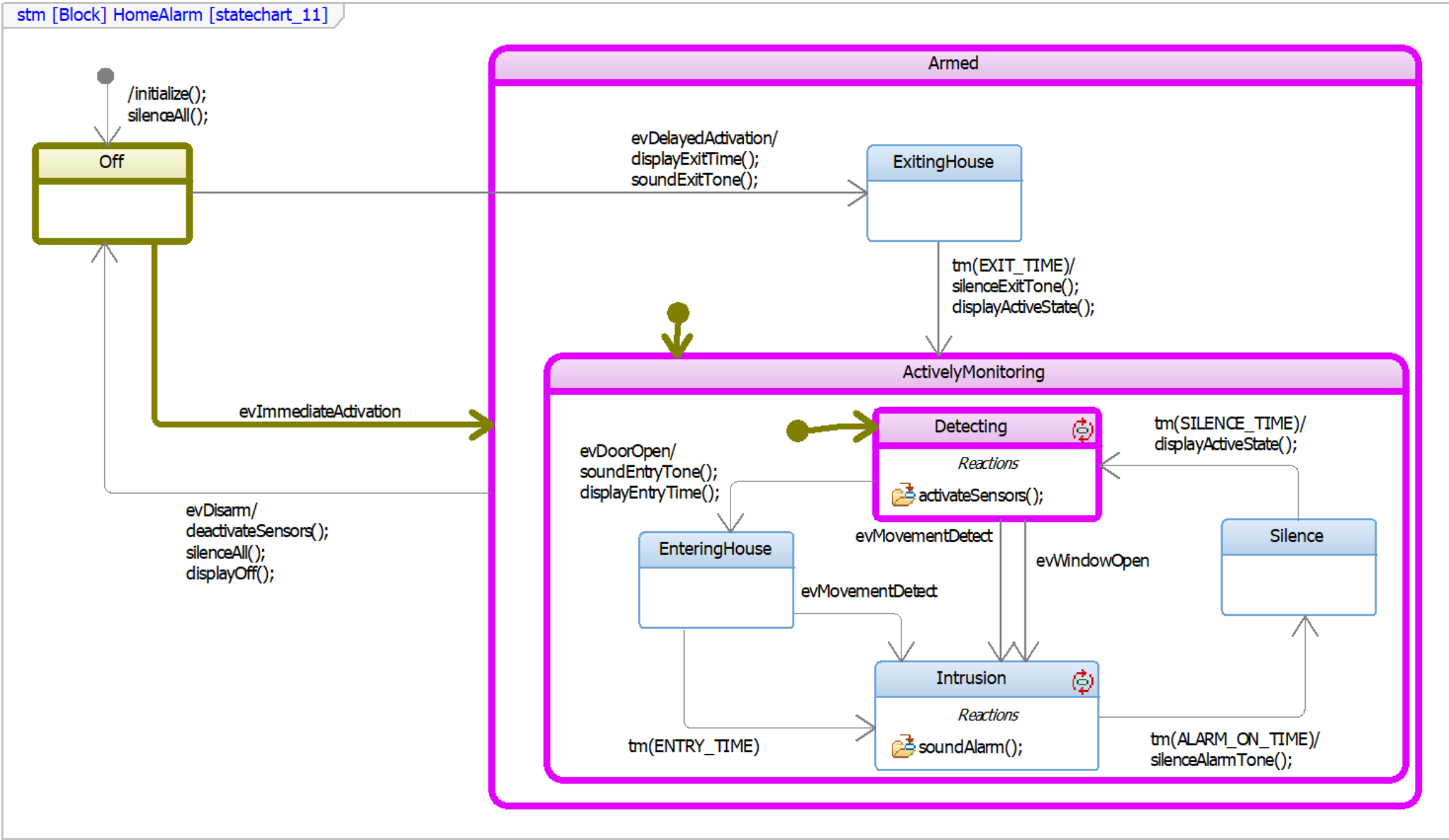
stm [Block] HomeAlarm [statechart\_11]



# Scenario 2 Step 1

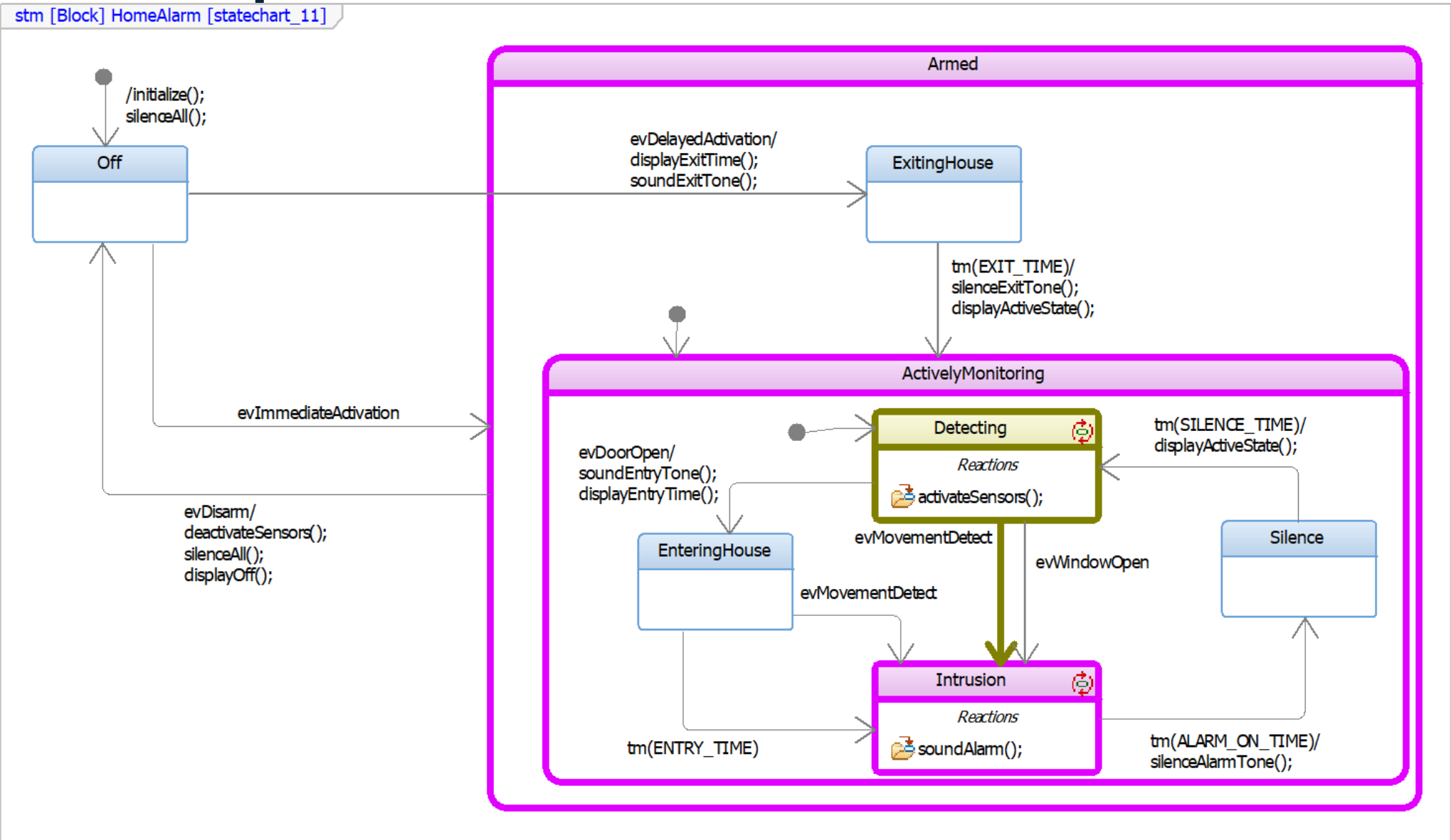


# Scenario 2 Step 2

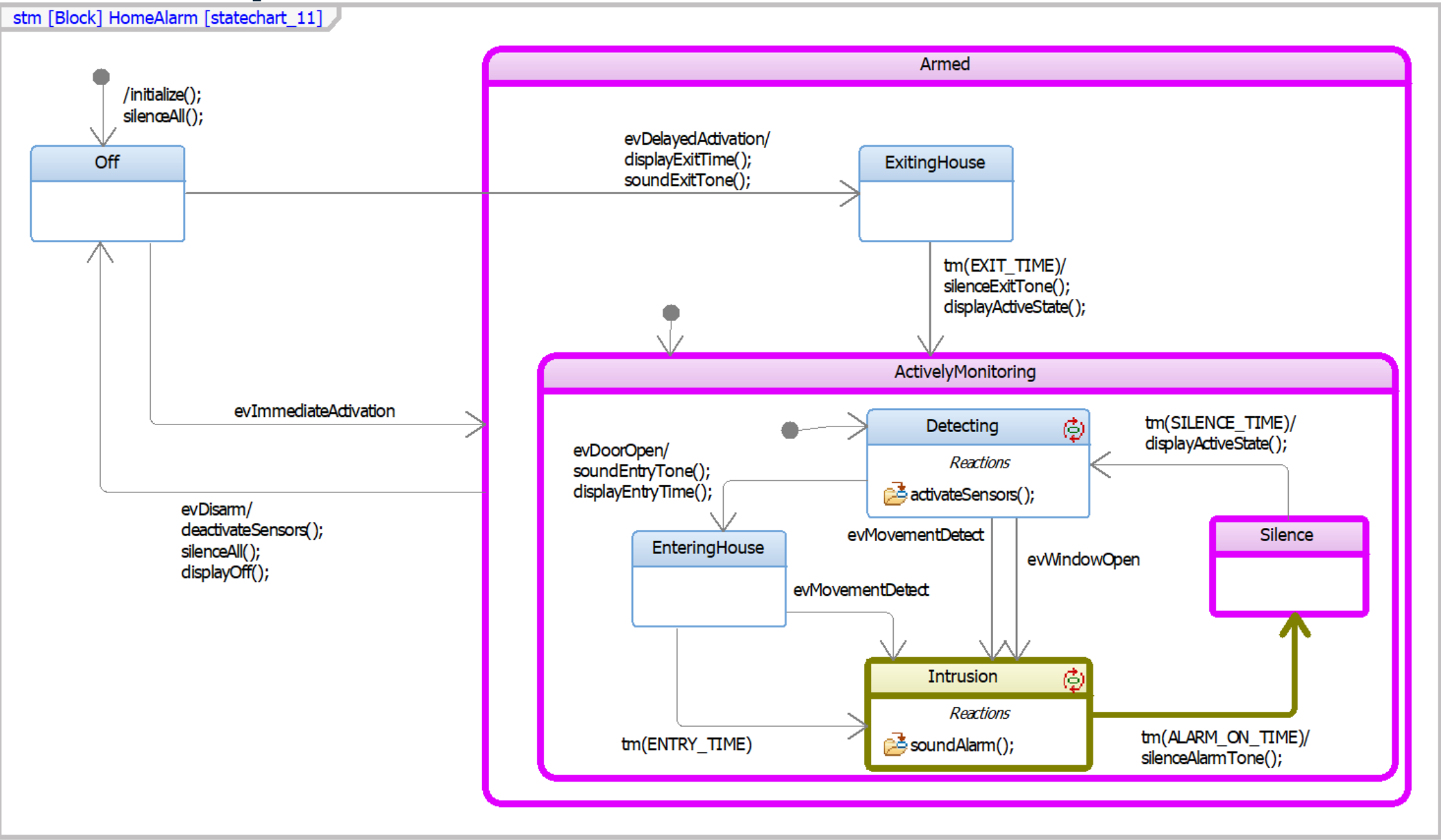




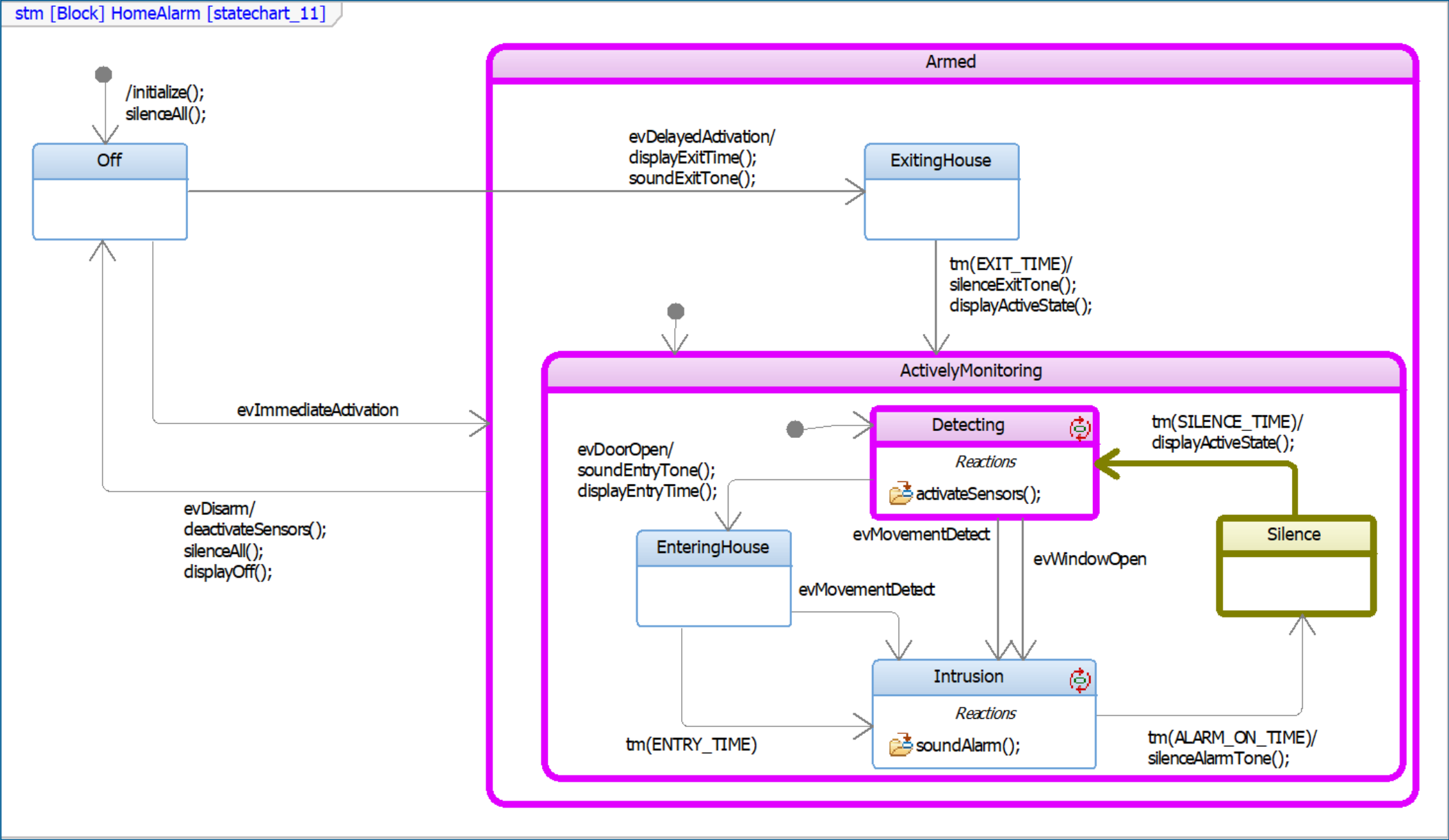
# Scenario 2 step 3



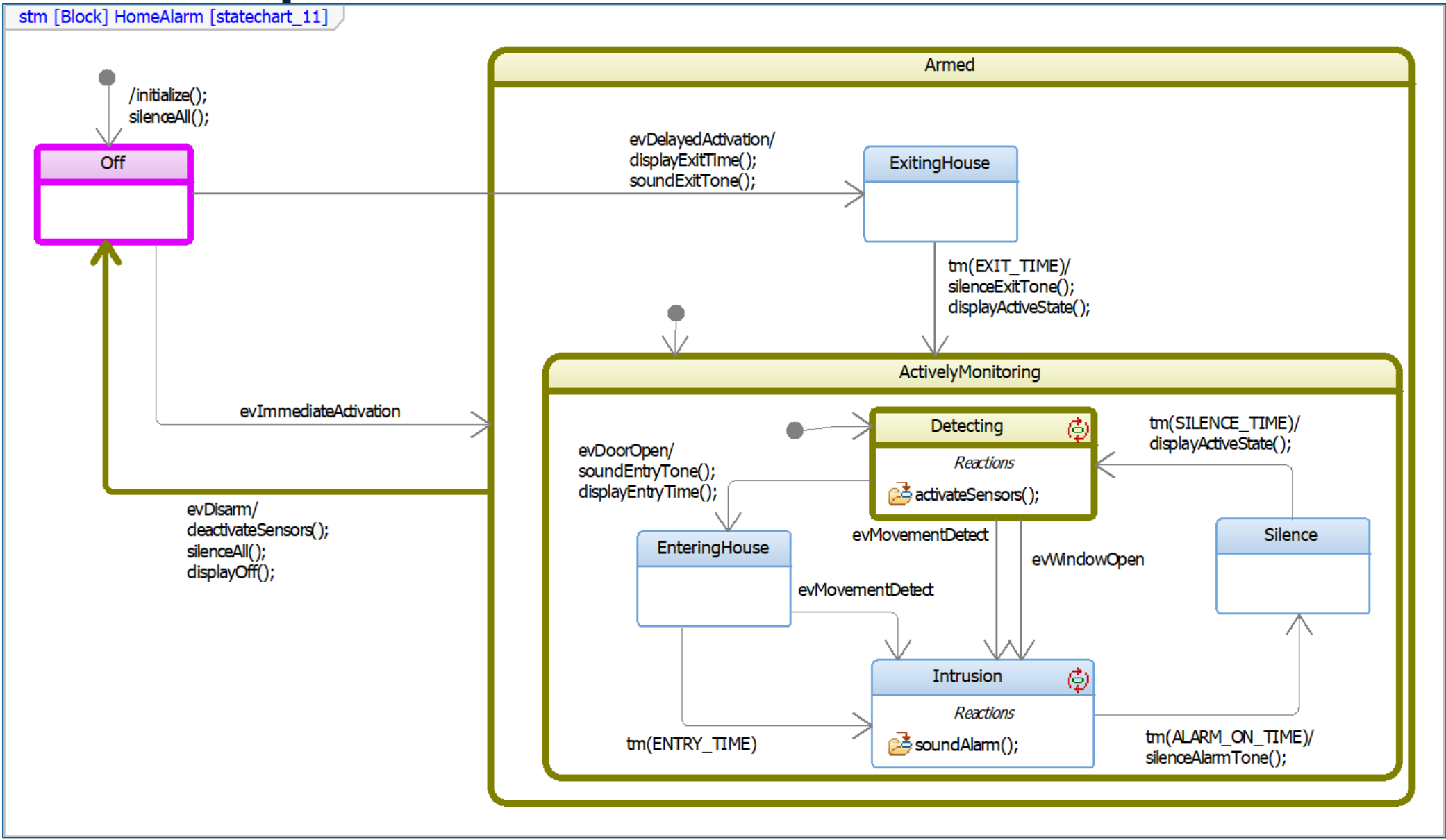
# Scenario 2 Step 4



# Scenario 2 Step 5

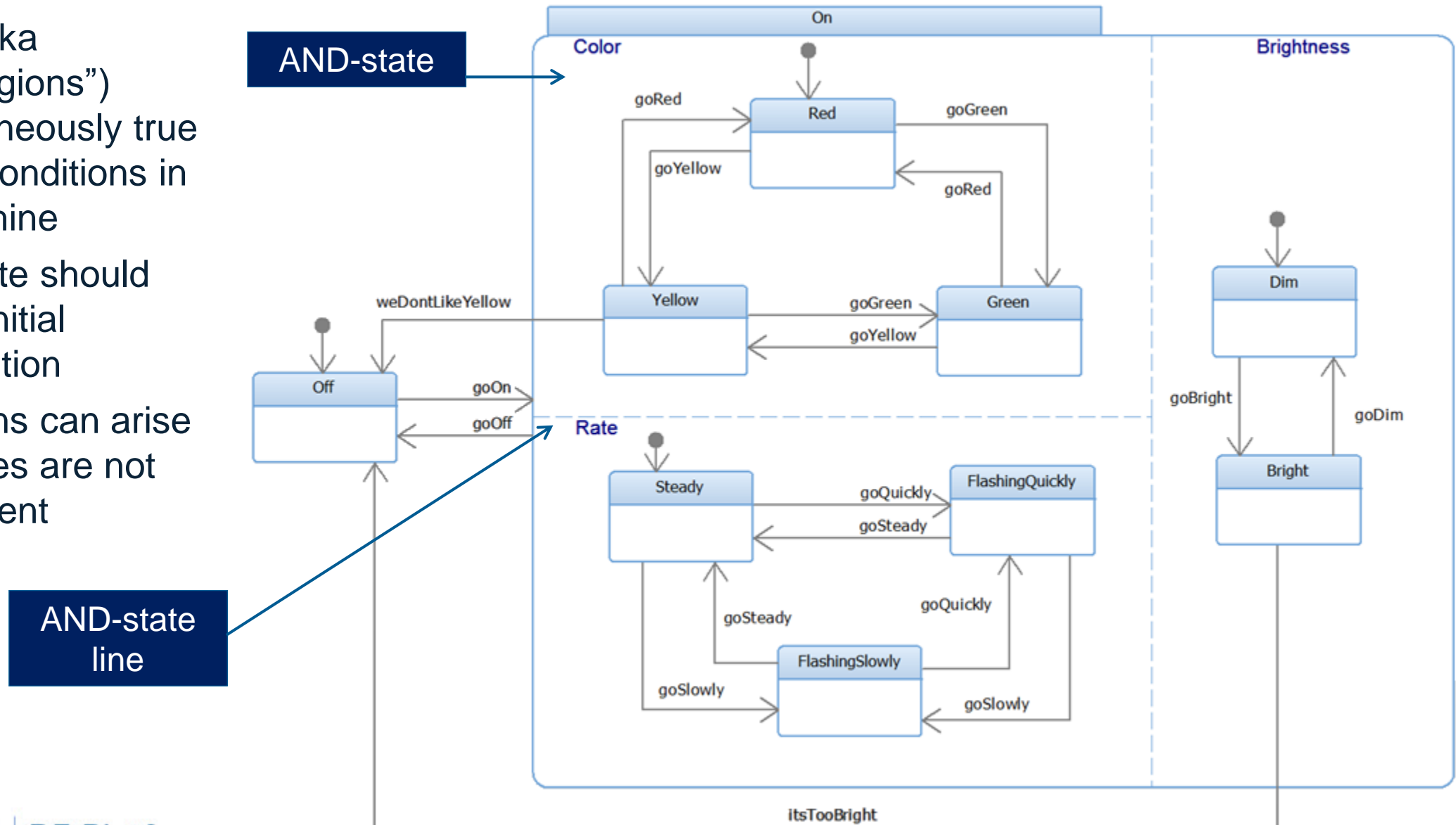


# Scenario 5 Step 6



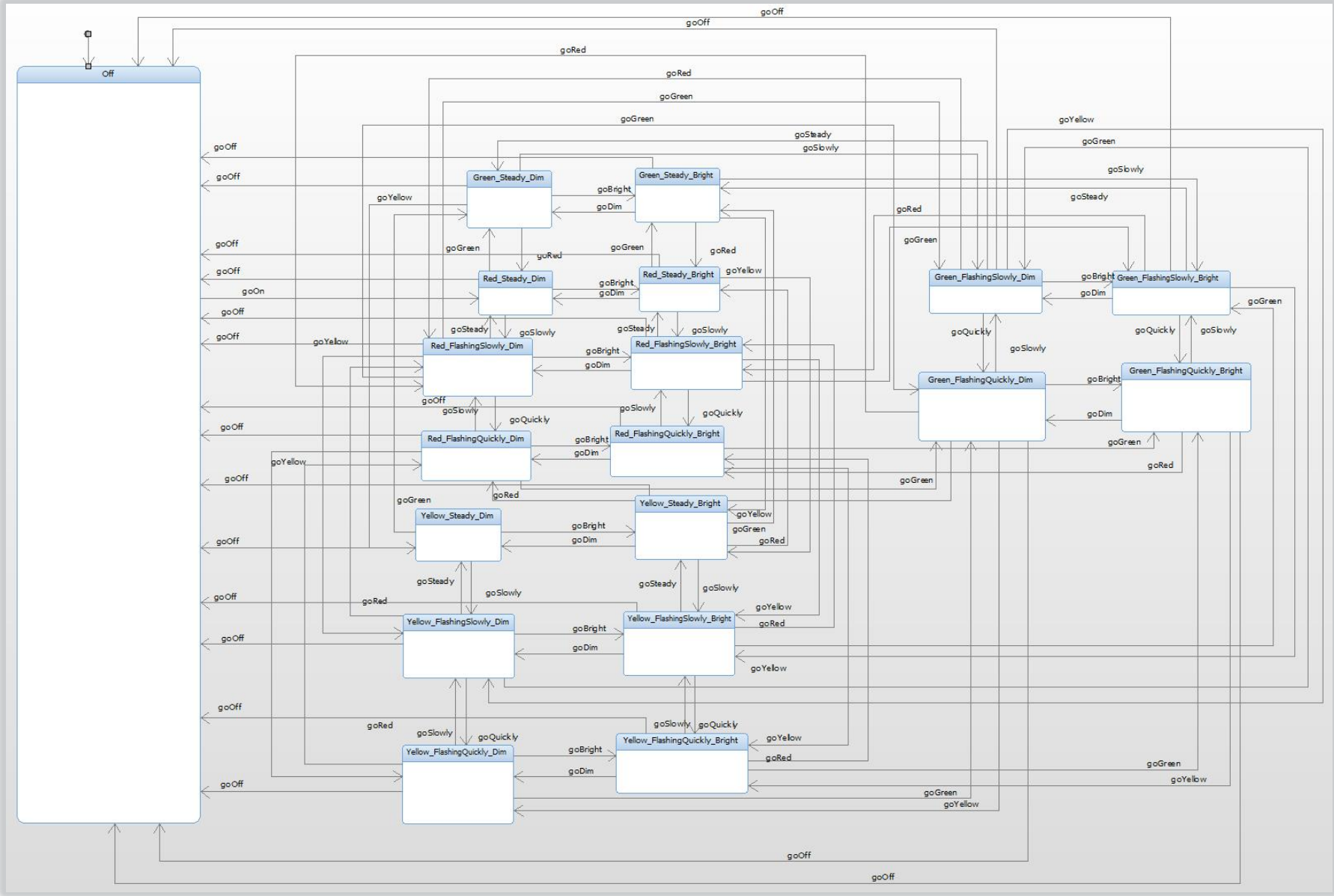
# AND States for simultaneously true conditions

- AND-states (aka “orthogonal regions”) model simultaneously true independent conditions in the state machine
- Each AND-state should have its own initial (default) transition
- Race conditions can arise if the and-states are not truly independent

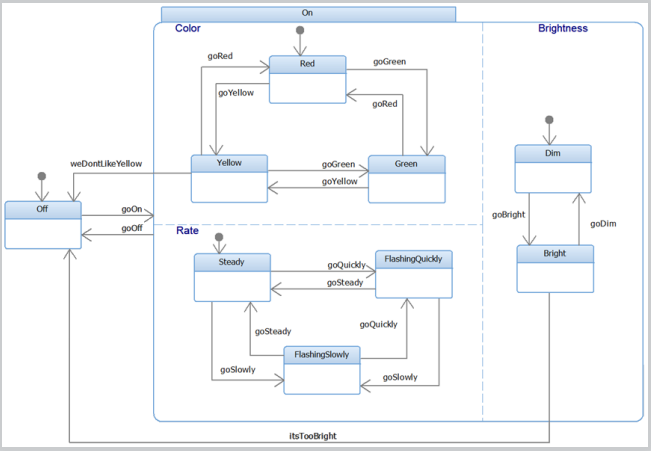


# AND-States can simplify models

Without And-States

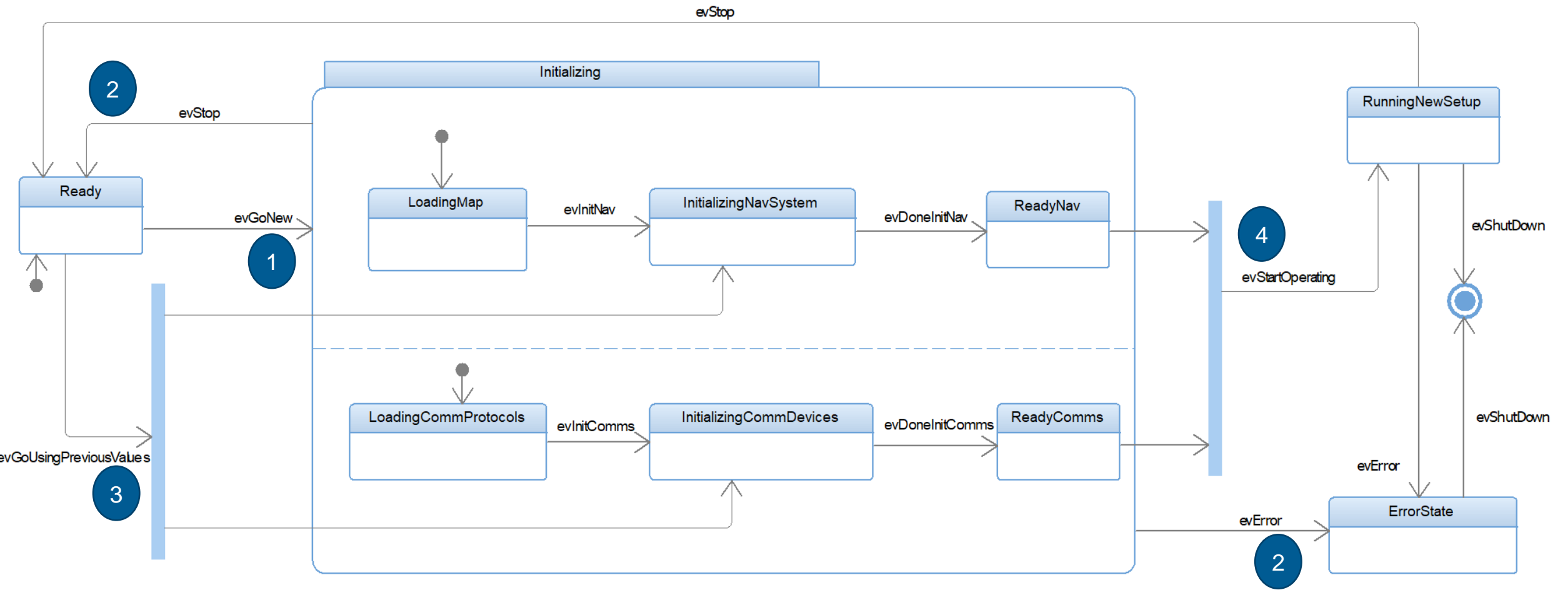


With And-States

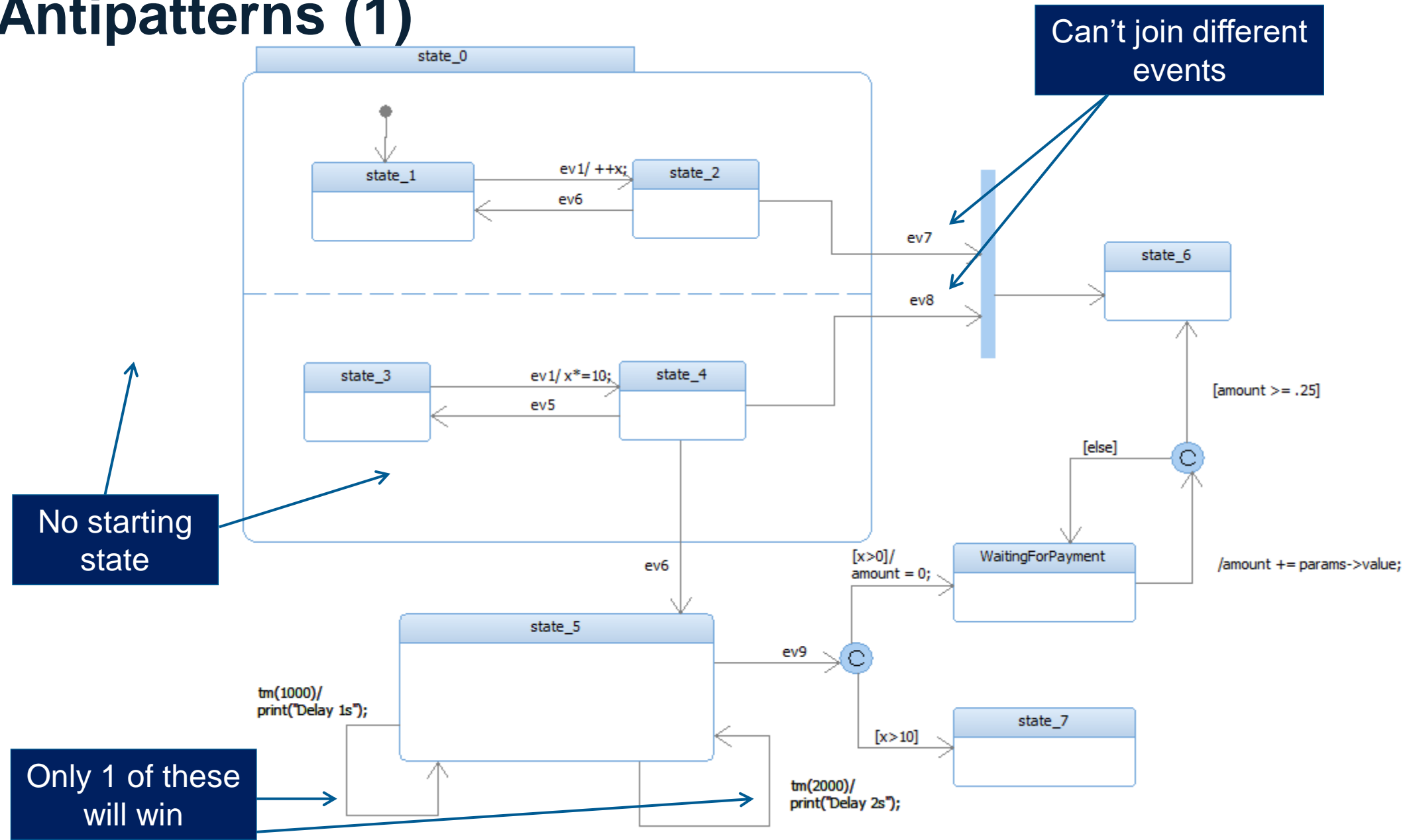


# Forks and Joins in State Machines

stm [Block] ANDStateBlock [statechart\_0]



# Antipatterns (1)



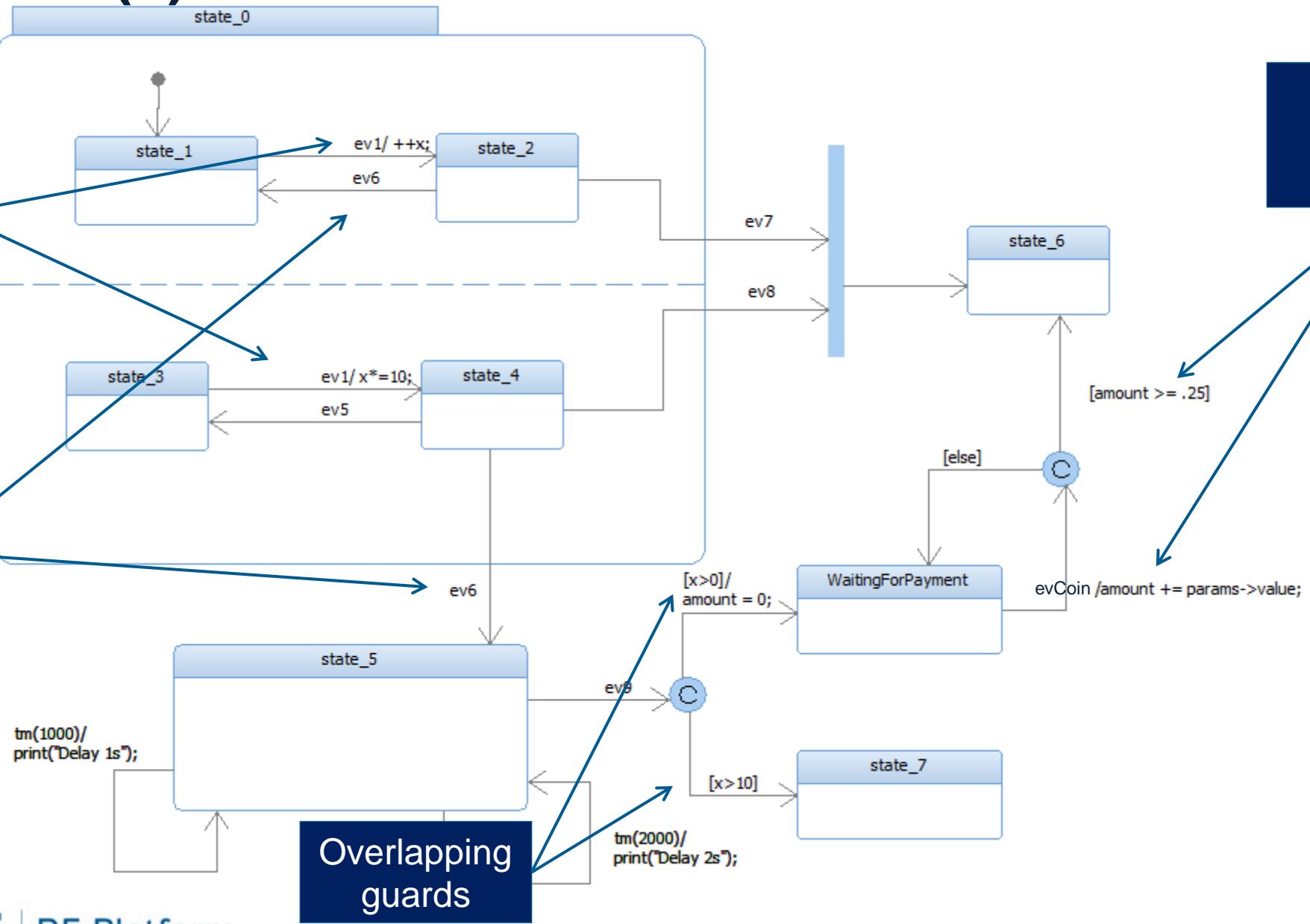


# Antipatterns (2)

Race Conditions

Conflicting Transitions

Probably ordering problem



# State Modeling: Key Takeaways



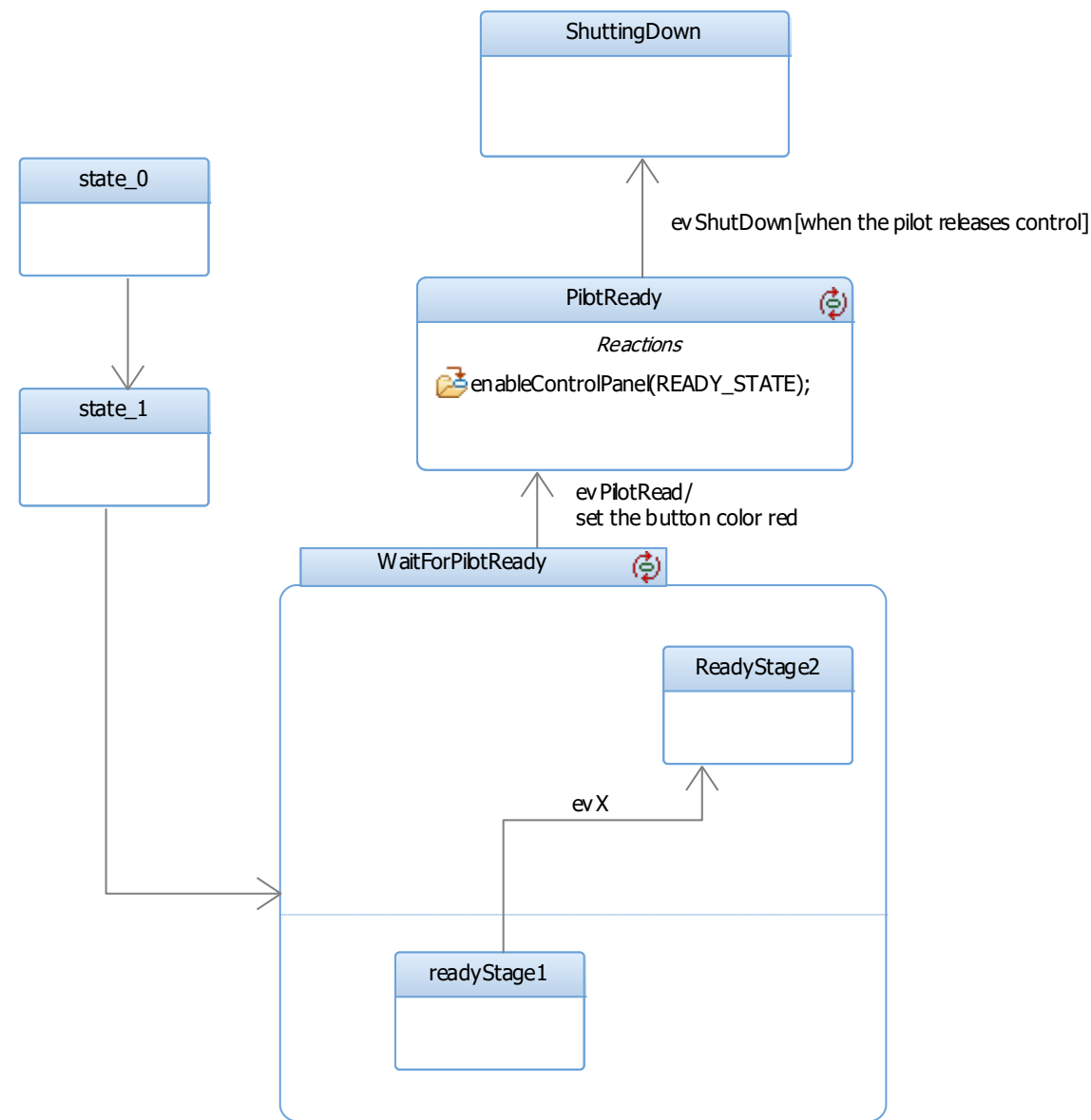
- **State machines** always represent the behavior of a single classifier (e.g. block)
- **States** are conditions of the instance
- **Transitions** represent the change of condition when the instance receives an event
- **Actions** are the behaviors executed when the instance changes state and may be
  - Entry actions
  - Exit Actions
  - Transition actions
  - Send actions are a kind of action that send events to the same or different instances
- Timeouts are a kind of event based on time (e.g. `tm(1000)` or `after(99)`)



# State Diagram Checklist

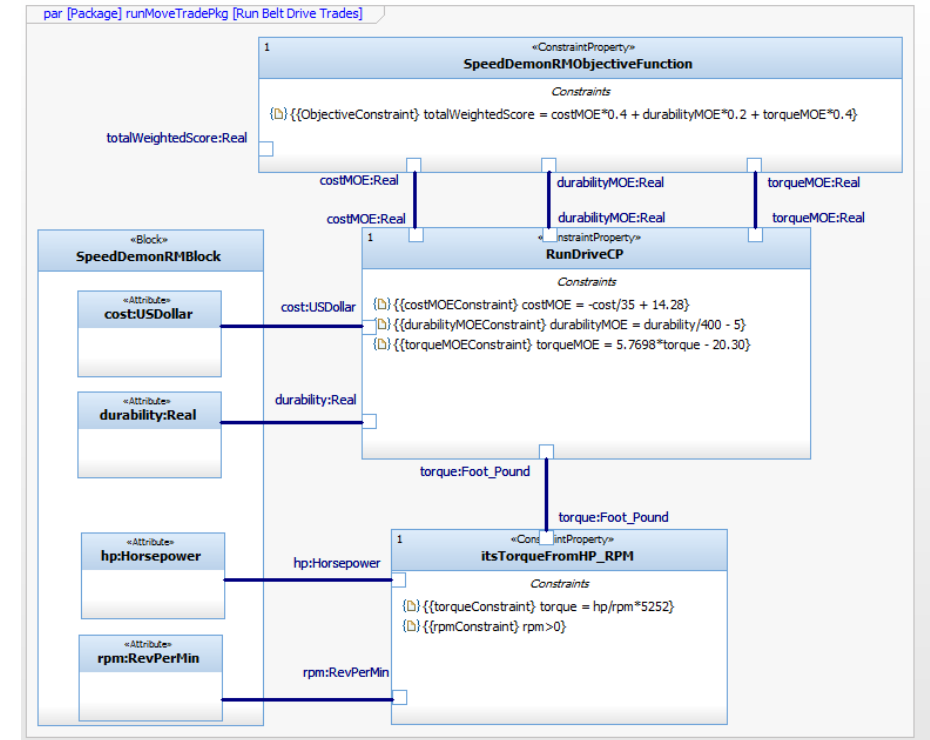
- ☐ Is there an initial (default) transition flow?
- ☐ Do the states (generally) wait for an event of interest to execute behaviors?
- ☐ Are actions specified in the appropriate action language?
  - ☐ Natural language for conceptual model
  - ☐ Action language for specific / executable models
- ☐ Is the data carried by events properly specified?
- ☐ Does the use of nested states simplify the state model?
  - ☐ Does each nested state provide its own initial (default) transition
- ☐ Are AND-states used sparingly?
  - ☐ Are states in different models independent in order of execution
  - ☐ Does each AND-state have its own initial (default) connector
  - ☐ Use of AND-states simplifies the state model
- ☐ Does the state machine execute?
- ☐ Is it clear how the correctness of the state machine is verified?
- ☐ (DON'T) Is the state machine really just an activity flow using state syntax?

# Exercise: Evaluate the following State Diagram



# Basic Parametric Modeling in SysML

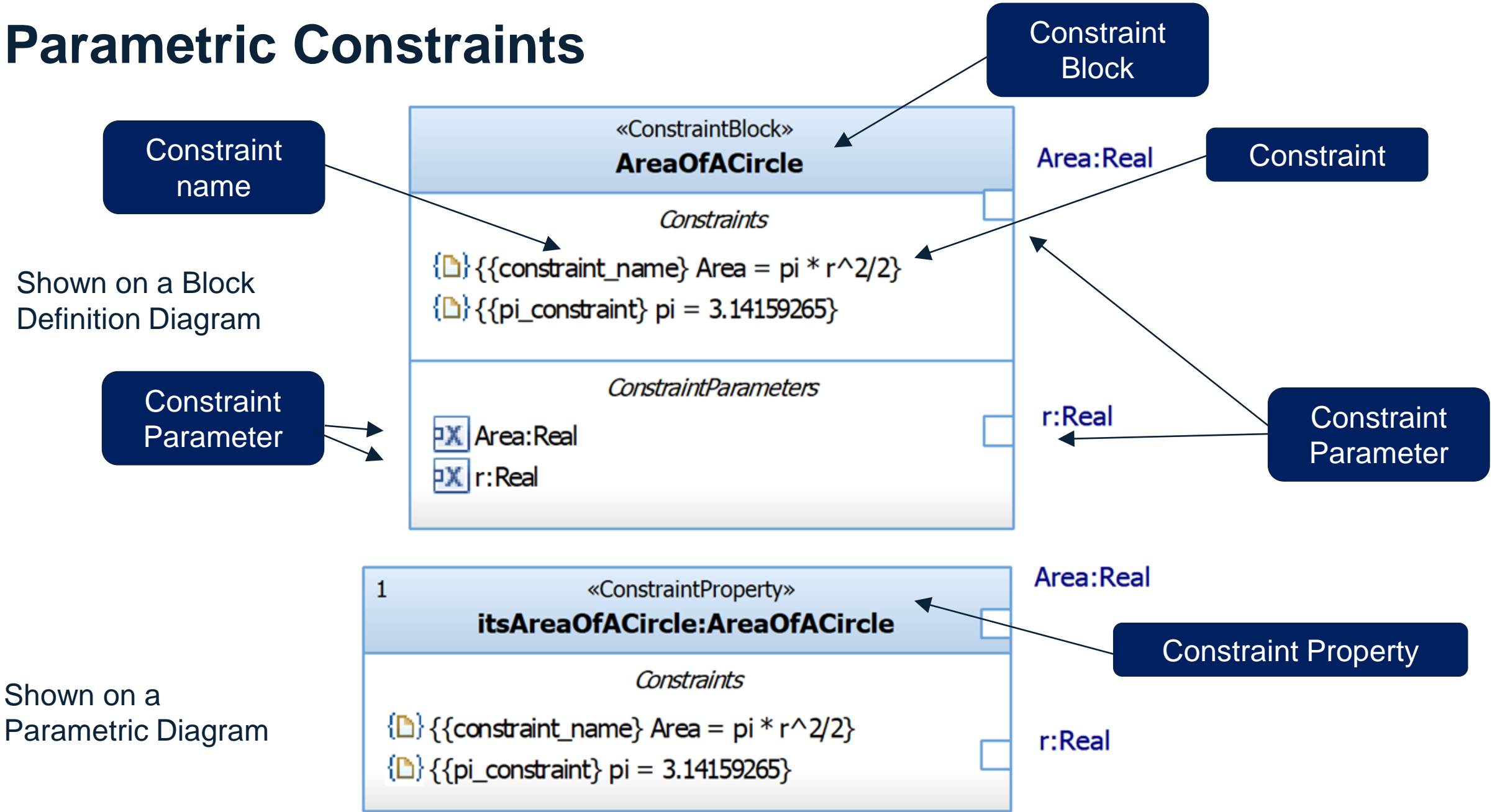
Parametric Constraints  
Instance Specifications  
Parametric Diagrams  
Checklists



# Parametric Constraint Modeling Core Concepts

- Constraint
  - Specifies a limitation on a value or property which is either mathematical or logical.
- Constraint Block
  - A SysML Block that contains of one or more constraints and one or more constraint parameters
- Constraint Parameter
  - Binds an input or output value to a value reference in a constraint
- Constraint Property
  - An instance (usage) of a constraint block
- Binding Connector
  - A binding connector links a value property to a constraint parameter

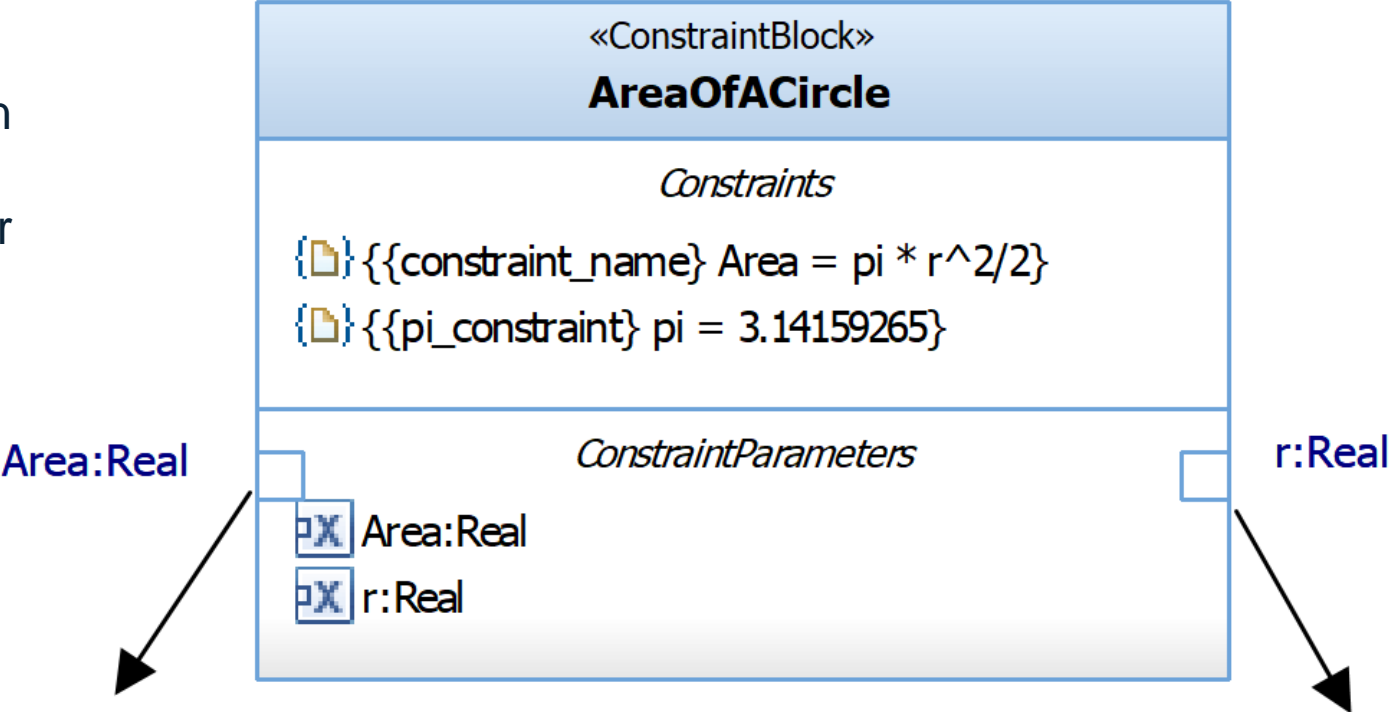
# Parametric Constraints



# Specifying Parametric Constraints



These are defined in property definition dialogs, available for all model elements.



Constraint Parameter: Area in AreaOfACircle

General Description Relations Tags Properties

Name: Area Label...

Stereotype: [dropdown]

Type: Real in SysML::StandardValueTypes [dropdown]

☐ Conjugated

Multiplicity: 1 [dropdown]

Direction: ☐ Input ☒ Output ☐ Bi-Directional

Advanced

Locate OK Apply

Constraint Parameter: r in AreaOfACircle

General Description Relations Tags Properties

Name: r Label...

Stereotype: [dropdown]

Type: Real in SysML::StandardValueTypes [dropdown]

☐ Conjugated

Multiplicity: 1 [dropdown]

Direction: ☒ Input ☐ Output ☐ Bi-Directional

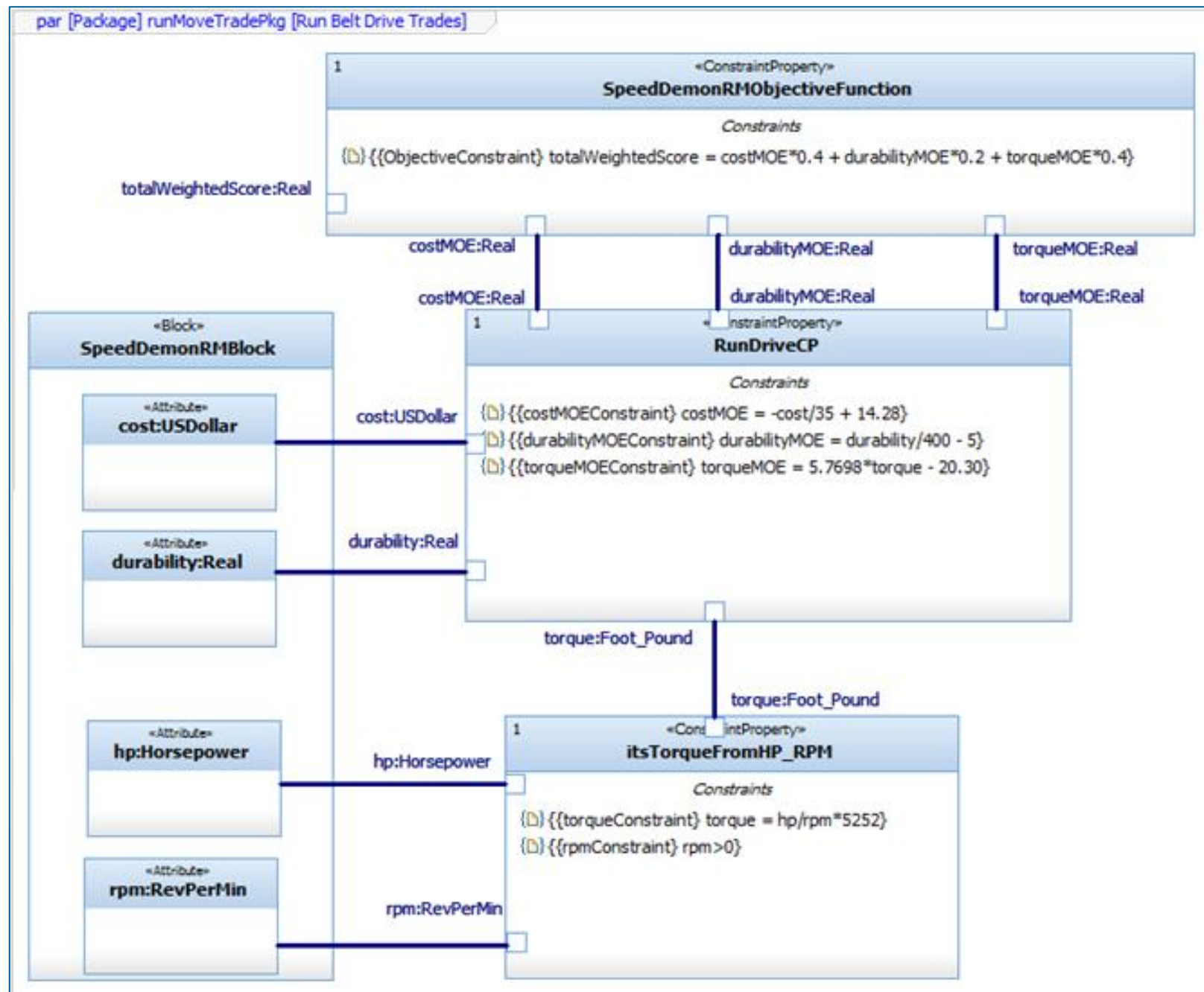
Advanced

Locate OK Apply



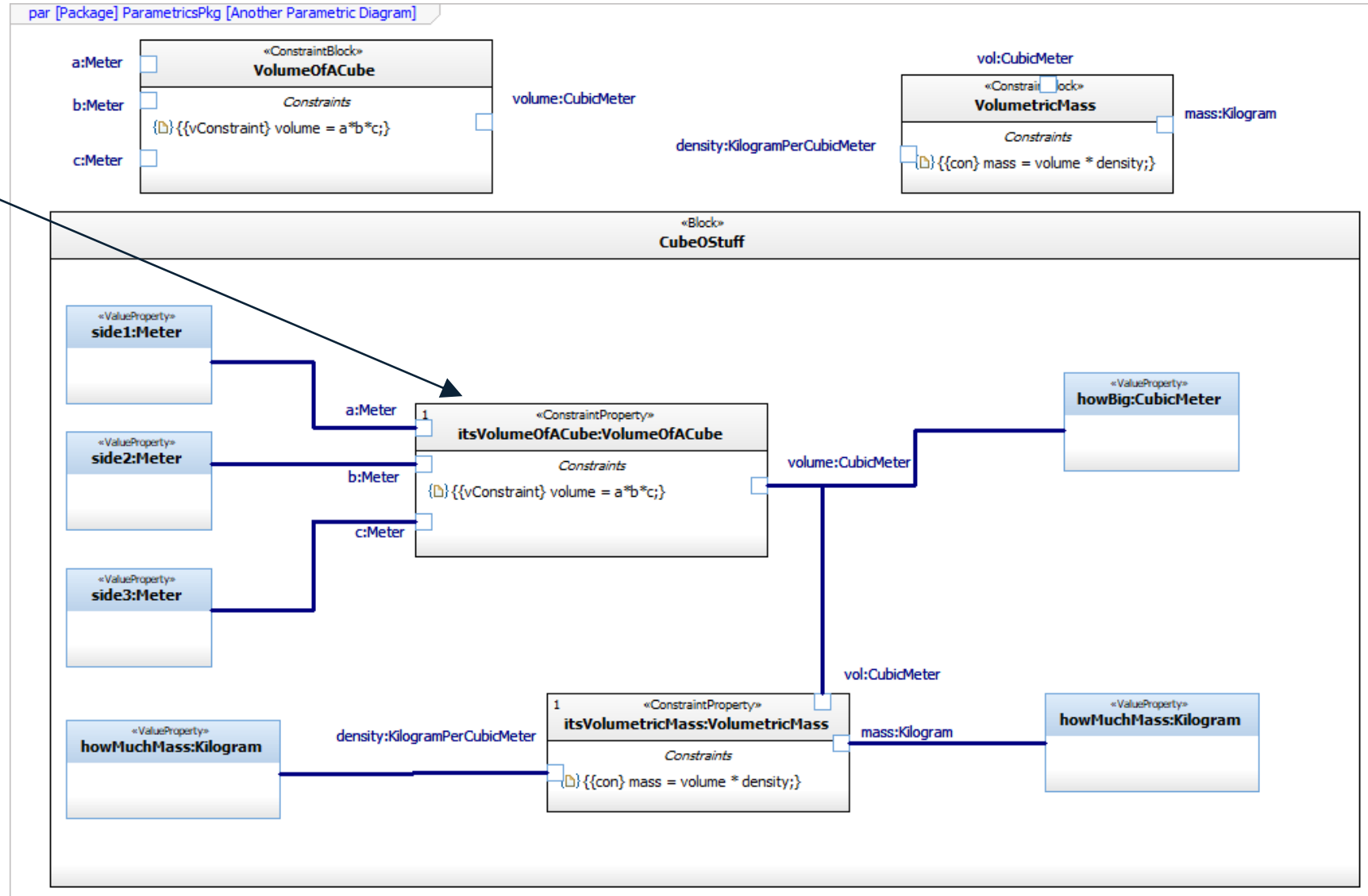
# Parametric Diagrams

- Link together constraint properties, which are defined by constraint blocks, with binding connectors to--
  - produce some computational or logical outcome, or
  - to express a computational or logical relation
- Used to--
  - Perform mathematical analysis
  - Aid in evaluation of measures of effectiveness for trade studies
  - Provide a series of computations



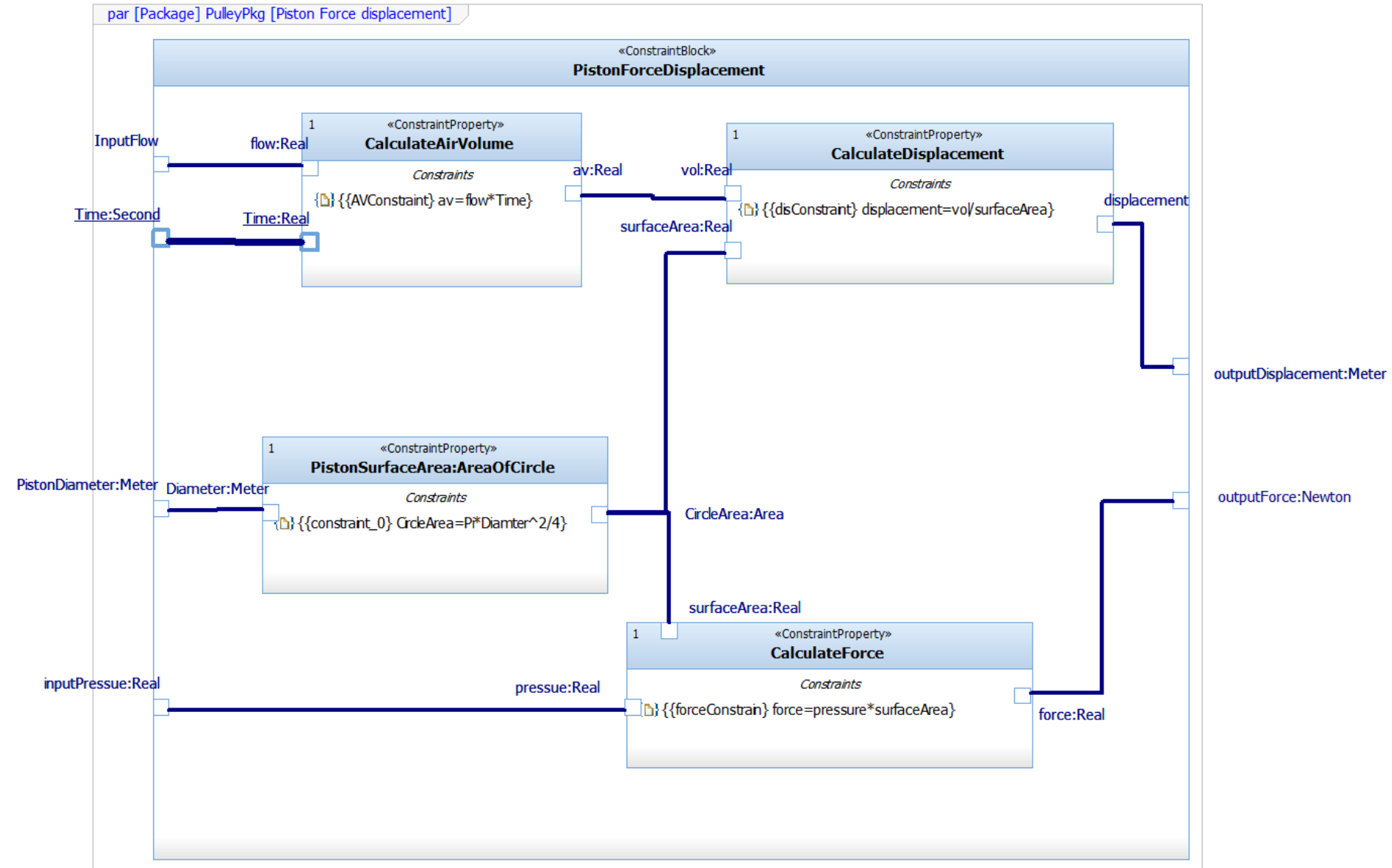
# Parametric Diagrams (simple example)

Blocks can contain  
Constraint Properties



Fix input to density  
calc

# Parametric Diagrams (more complex example)



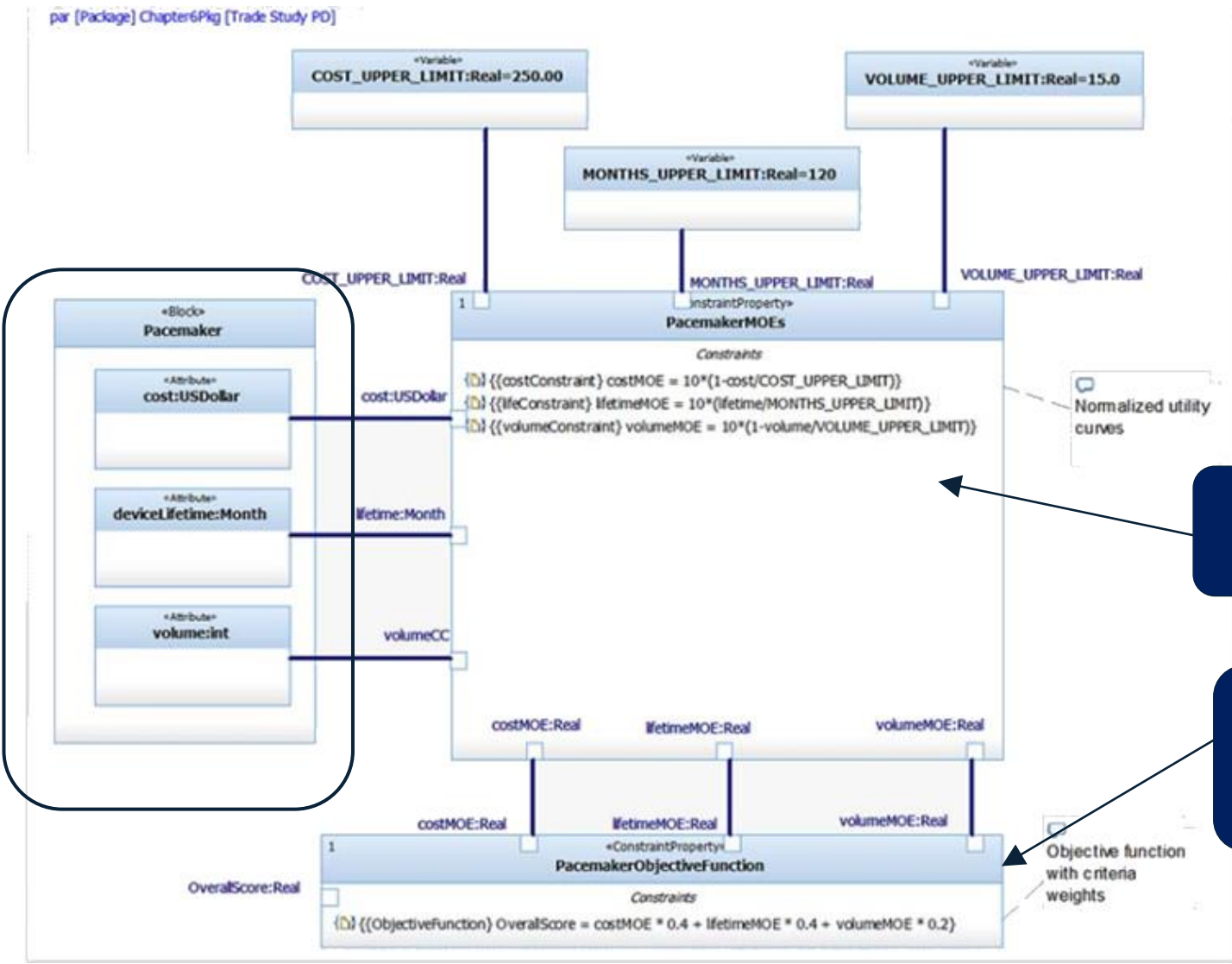
# Parametric Diagrams (trade study example)

Alternative case values

This Pacemaker block has attributes

- cost
- deviceLifetime
- volume

This block is special-purpose, created for this analysis

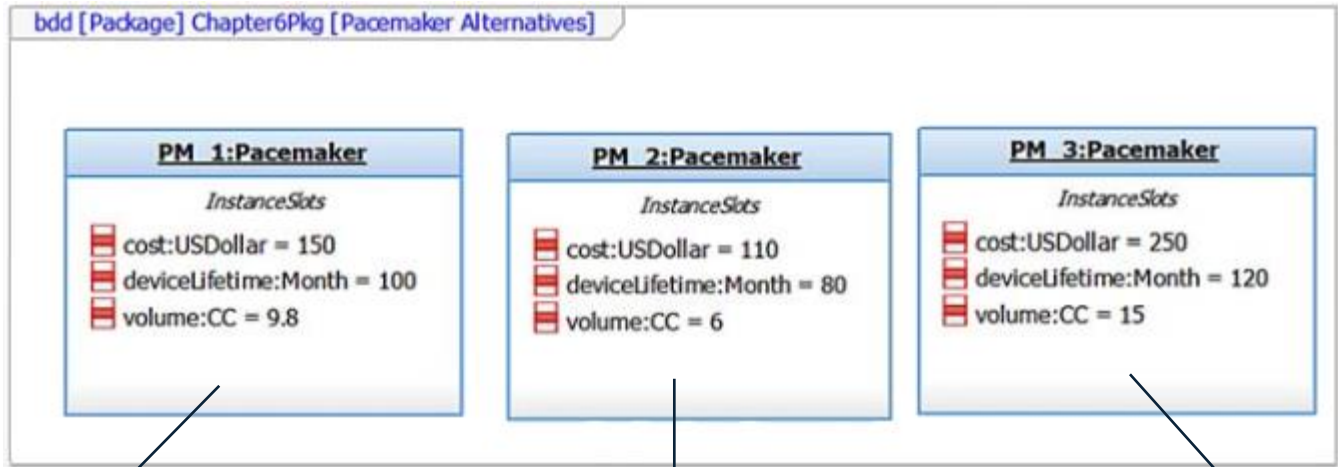


MOE Utility curves

Optimization objective function

# Parametric Diagrams (trade study example)

Instance specifications of  
**Pacemaker** Block



volumeConstraint	Constraint	volumeMOE ...	volumeMOE ...
PacemakerObjectiveFunc	PacemakerObjectiv...		
costMOE	Real		0
lifetimeMOE	Real		10
volumeMOE	Real		0
OverallScore	Real		4
ObjectiveFunction	Constraint	OverallScore ...	OverallScore ...

volumeConstraint	Constraint	volumeMOE ...	volumeMOE ...
PacemakerObjectiveFunc	PacemakerObjectiv...		
costMOE	Real		4
lifetimeMOE	Real		8.333333333...
volumeMOE	Real		3.466666666...
OverallScore	Real		5.626666666...
ObjectiveFunction	Constraint	OverallScore ...	OverallScore ...

volumeConstraint	Constraint	volumeMOE ...	volumeMOE ...
PacemakerObjectiveFunc	PacemakerObjectiv...		
costMOE	Real		5.600000000...
lifetimeMOE	Real		6.666666666...
volumeMOE	Real		6
OverallScore	Real		6.106666666...
ObjectiveFunction	Constraint	OverallScore ...	OverallScore ...

See Trade Studies with Rhapsody and SysML at <https://www.bruce-douglass.com/presentations> for more detail



# Antipatterns

Parameter is being treated as an input, but is defined as an output

Constraint Parameter : in1 in SomeMath

General

Description

Relations

Tags

Properties

Name:

in1

Label...

Stereotype:

Type:

OMString

☐ Conjugated

Multiplicity:

1

Direction:

☐ Input

☒ Output

☐ Bi-Directional

Advanced

Redefines:

...

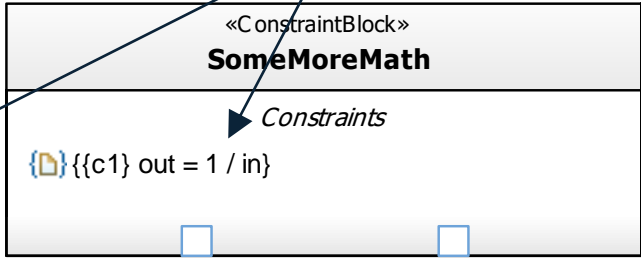
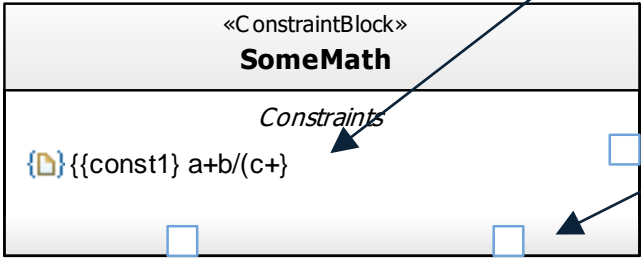
Locate

OK

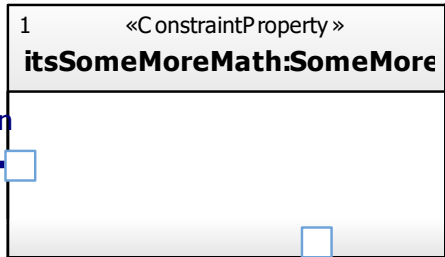
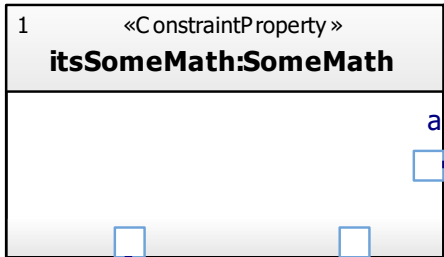
Apply

Invalid expression as constraint

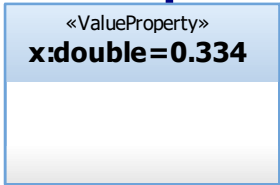
Computation not relevant to inputs



Incompatible types



Incompatible types



Parameter not connected

# Parametric Constraint Key Takeaways



- Parametric diagrams express mathematical or logical constraint relations among sets of values
- These constraint relations include
  - Constraints: statement of the mathematical or logical relation
  - Constraint blocks: combine constraints and constraint parameters
  - Constraint parameters: define input and output variables
  - Constraint properties: usages of a constraint block in a specific context
  - Binding connectors: connect values and constraint parameters



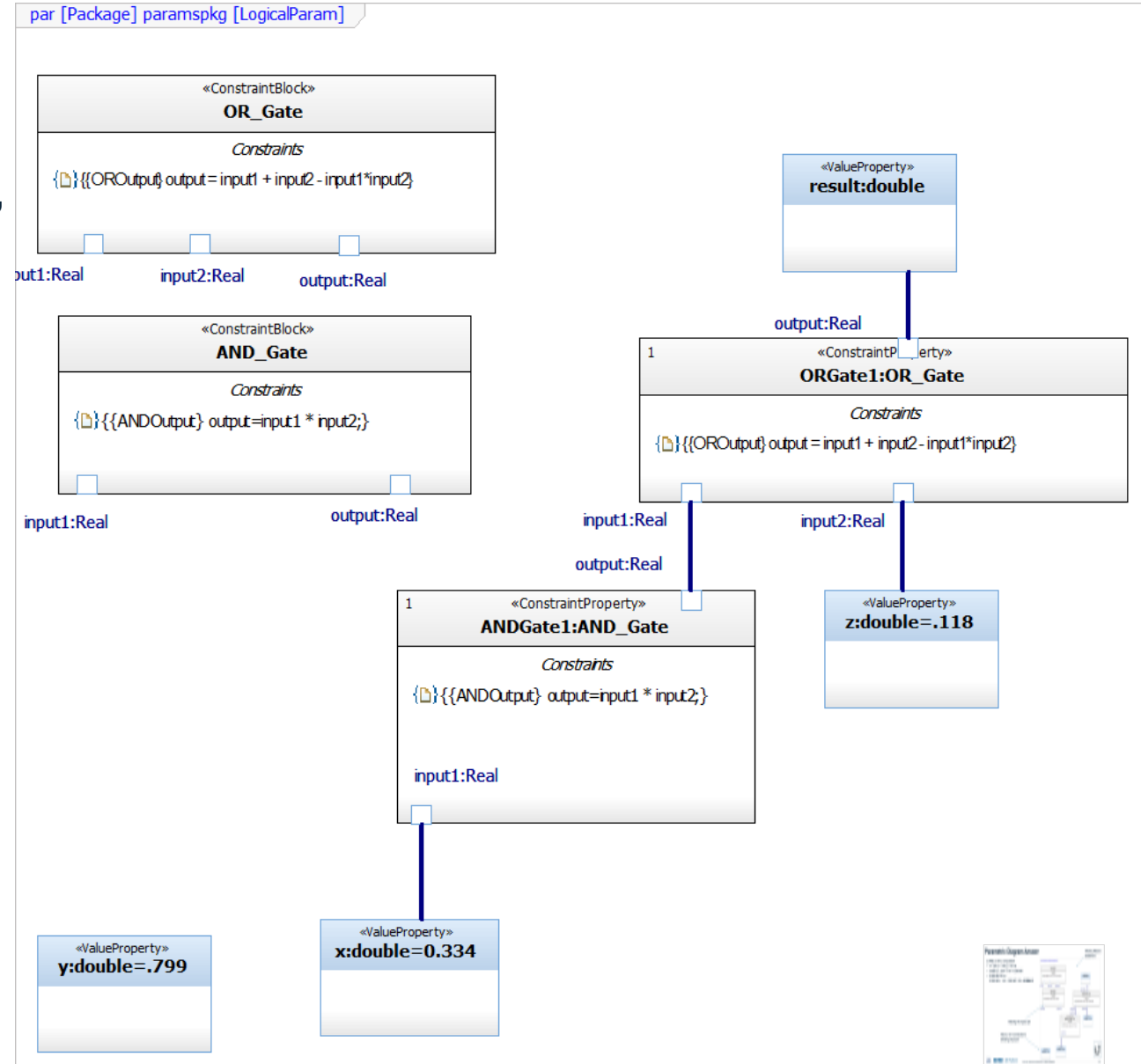
# Parametric Diagram Checklist

- ☐ Is the constraint mathematically or logically valid?
- ☐ Are the connected input and outputs compatible in terms of type and units?
- ☐ Are the inputs and outputs appropriately named?
- ☐ Do the outputs of an element match up to the inputs of the constraint property to which it is bound
- ☐ Are the constraint blocks and constraint properties appropriately named
- ☐ Is the purpose of the parametric diagram clearly stated



## Exercise: Evaluate the following Parametric Diagram

- Identify any problems with the parametric diagram
- If  $x$  (.334),  $y$  (.779), and  $z$  (0.118) are all probabilities, what is  $(x \text{ AND } y) \text{ OR } z$ , given that
  - $\text{AND} = \text{input1} * \text{input2}$   
and
  - $\text{OR} = \text{input1} + \text{input2} - \text{input1} * \text{input2}$ ?
- Does this parametric diagram properly compute that?



# Basic Tables & Matrices in SysML

Traceable Dependencies

Trace Matrices

Checklists



# Table vs Matrix

- A **table** is a view of a selected set of model elements in a defined context which exposes data or properties of those elements
  - Example: **Requirements Table** (predefined in SysML) shows all requirements in a specific set of packages and their name, id, and requirements text
- A **matrix** is a view of a common set of relations between sets of model elements in a specified context
  - Example: **Use Case – Requirements Trace Matrix** shows the requirements that are traced to or from a selected set of use cases
- Core concepts
  - **Layout vs View**: Layout defines the structure and contents, the view adds the data from a specific context (typically a set of packages)
  - **Metadata** – data owned by a model element that may be exposed in a table
  - **Relation** – a navigable relation between pairs of model elements

# Tables are Useful for Summarizing Information

Tables are useful for showing the information, properties, relations, and connections of model elements in the model.

- Information held in the properties of model elements is often referred to as **metadata**























Tables are commonly used to show requirements and requirements metadata.

In addition, tables can:

- Show design element metadata
- Show relations among model elements and the properties of elements to which the model elements are related
- Summarize information held in multiple diagrams
- Be easily imported and exported to spreadsheets
  - Import allows information in spreadsheets to be brought into models
  - Export allows information to be exported to spreadsheets

# Example: Requirements Table











































- Tables report on elements of a common type within a specified scope
- Columns show the values different properties of the elements within the table

Criteria			
Scope (optional): Customer Requirements Pkg [v] Filter: ▼			
#	Name	Id	Text
1	 60 Req802	60	In most situations, the sensors positions are controlled by the CMPCS but may be automatically controlled for automated search missions.
2	 77 Req912	77	Once located, the missile homes in on the reflected laser light. This operational mode requires the <u>CUAV</u> to maintain the laser designator on target until the missile reaches its target, so a single target can be fired upon at a time.
3	 14 Req101	14	Each control station shall consist of two manned substations - one for controlling the <u>CUAV</u> and one for monitoring and controlling payloads.
4	 30 Req303	30	The <u>CUAV</u> shall be able to fly reliably in inclement and low visibility weather;
5	 17 Req104	17	Stable flight mechanics shall be managed by the aircraft itself but this can be disabled for remotely controlled flight.
6	 49 Req704	49	A single CMPCS can control up to 4 CUAVs in flight with one station per <u>CUAV</u> .
7	 16 Req103	16	Control of the aircraft shall consist of transferring navigational commands which may be simple (set altitude, speed, direction), operational (fly to coordinate set, orbit point, execute search pattern, etc), planned (upload multi-segment flight plan) or remote controlled with a joystick interface.
8	 38 Req502	38	If the <u>CUAV</u> experiences an unexpected loss of communications with its CMPCS in excess of 60 minutes, it can abort the mission and perform an automated return and landing. This is meant to decrease incidence of <u>UAV</u> loss due to <u>ECM</u> and communications failures.
9	 82 Req1002	82	The control datalink is an encrypted secure low-bandwidth datalink that supports vehicle and <u>payload</u> commands and status information.
10	 73 Req908	73	The missiles attach to the <u>CUAV</u> on firing rails mounted on pylons, allowing two missiles to be mounted on each wing.
11	 79 Req914	79	Once the <u>CUAV</u> has located the target and the CMPCS has given the command to release, the missile maintains radar lock on the target. This allows the <u>CUAV</u> to fire and mask, or to seek other targets while the missile flies to its target.
12	 55 Req710	55	The <u>CUAV</u> shall support up to 4 separate targets to be fired upon in rapid succession.
13	 33 Req311	33	In addition to the ATR, the <u>payload</u> operator can add targets visually identified from reconnaissance data or gather from other sources.
14	 43 Req600	43	The <u>UAV</u> shall be able to take off and land using autonomous or remote piloting. Beyond flight modes, <u>CUAV</u> shall be designed for highly flexible mission parameters. Normal mission modes include: - Preplanned reconnaissance - Remote controlled reconnaissance - Area search - Route search - Orbit point target - Attack
15	 44 Req601	44	A mission can consist of any number of sequential sub-missions each operating in a different mission mode, depending on the current <u>payload</u> .
16	 71 Req905	71	The missile shall be able to defeat reactive armor and successfully deploy in adverse and hazy weather.
17	 7 Req007	7	Control commands shall be encrypted while telemetry data can be compressed but unprotected.
18	 4 Req004	4	While controllable from the ground station CMPCS, it is also capable of flying complex flight plans with specific operational goals of systematic area search, ground route (road-based) search, and orbit surveillance of point targets.
19	 64 Req807	64	The daylight variable aperture or the infrared electro-optical sensor may be operated simultaneously with the synthetic aperture radar.
20	 84 Req1004	84	The system shall be able to detect and correct all single and dual bit errors and detect multiple-bit errors with a reliability of 0.9994.
21	 86 Req1007	86	The high speed data link is used for transmission from the <u>CUAV</u> only and is meant to include real-time telemetry, reconnaissance and surveillance data.
22	 42 Req506	42	In a high <u>ECM</u> environment, data rate shall be 320x200 at 15 fps for optical and <u>FLIR</u> imaging. <u>SAR</u> data shall be supported as single frame images (640x400) transmitted at a rate not exceeding 1 fps.
			When flying in environments where fewer than 3 GPS satellites will be visible, the <u>CUAV</u> shall have an on-board altimeter as well.



You can add more columns of metadata as desired.

# Example: Requirements Table with Derivation Metadata

#	Id	△ Name	Text	Derived	Derived From
1	1	 1 Stakeholder Req 1	The patient shall receive enough oxygen to sustain life for patients ranging from neonates to full adults in size and mass."	 2 System Requirement 1	
2	2	 2 System Requirement 1	The system shall deliver oxygen from 50 ml/minute to 1,500 ml/minute settable in 1 ml increments	 3 Gas Supply Requirement 1  4 Gas Supply Requirement 2  5 Gas Supply Requirement 3  6 Gas Mixer Requirement 1  7 Gas Mixer Requirement 2	 1 Stakeholder Req 1
3	3	 3 Gas Supply Requirement 1	The Gas Supply Subsystem shall be provide flows of from 0 to 1500 ml/min of O2		 2 System Requirement 1
4	4	 4 Gas Supply Requirement 2	The Gas Supply subsystem shall provide flows of from 0 to 7500 ml/min of Air		 2 System Requirement 1
5	5	 5 Gas Supply Requirement 3	The Gas Supply subsystem shall provide flows of from 0 to 6000 ml/min of N2		 2 System Requirement 1
6	6	 6 Gas Mixer Requirement 1	The Gas Mixer subsystem shall mix gases to deliver flows between 50 and 7500 ml/min of mixed gas.	 11 GM ME Requirement 1  12 GE ME Requirement 2  10 GM EE Requirement 1  8 GM SW Requirement 1  9 GM SW Requirement 2	 2 System Requirement 1
7	7	 7 Gas Mixer Requirement 2	The Gas Mixer subsystem shall only deliver gas flows of at least 21% O2 concentration.		 2 System Requirement 1
8	8	 8 GM SW Requirement 1	The software shall raise an alert if a commanded flow is less than 21% O2.		 6 Gas Mixer Requirement 1
9	9	 9 GM SW Requirement 2	The software shall raise an alert if the total commanded O2 flow is outside the range of 50-7500 ml/n		 6 Gas Mixer Requirement 1
10	10	 10 GM EE Requirement 1	The electronics shall control the gas supply valves to ensure gas flow is within 1% of commanded value		 6 Gas Mixer Requirement 1
11	11	 11 GM ME Requirement 1	Each mechanical valve on the gas mixer manifold shall support openings that deliver a range of 0 (completely closed) to 10,000 ml/min at 10 KPa pressure.		 6 Gas Mixer Requirement 1
12	12	 12 GE ME Requirement 2	The mechanical gas manifold shall support input from 4 gas supplies simultaneously.		 6 Gas Mixer Requirement 1
13	13	  13 RA Req 1	The system shall move the robot arm in compliance with user command		
14	13.1	 13.1 RA Req 2	The system shall move the robot are in compliance with user command in the range of – 30 degrees and + 45 degrees.		
15	13.2	 13.2 RA Req 3	The system shall move the robot arm in compliance with the user command with an accuracy ± 0.1 degrees (accuracy is precision of the output)		
16	13.3	 13.3 RQ Req 4	The user shall be able to specify the robot arm position to within 0.05 degrees (fidelity is the precision of the input)		
17	13.4	 13.4 RA Req 5	The system shall move the robot arm to the specified position within 300ms		
18	13.5	 13.5 RA Req 6	The system shall reject movement commands that are outside of the allowable range and raise a Caution Alert.		
19	13.6	 13.6 RA Req 7	The system shall raise a Warning Alert if the required accuracy or timing of the robot arm movement is not compliant upon completion of movement.		



# Data Schema Table

Criteria												
Element Type: Value Property		...	Scope (optional): Logical Data Schema Pkg		{0}	...	Filter:					
#	Owner	Name	Type	Applied Stereotype	extent	low value	high value	maximum latency in seconds	prohibited values	scale of accuracy	scale of fidelity	scale of precision
1	Computed Performance Data	acceleration	acceleration[kilometre p...	ValueProperty [Property] «> tempered [Element]		0	65000			0	0	1
2	App Account Data	account name	String	ValueProperty [Property]								
3	App Account Data	appID	String	ValueProperty [Property]								
4	RideSession	average power	power[watt]	ValueProperty [Property] «> tempered [Element]		0	2000			1	1	1
5	Computed Performance Data	average power	power[watt]	ValueProperty [Property] «> tempered [Element]		0	2000			1	1	1
6	RideSession	average speed	speed[kilometre per hour]	ValueProperty [Property] «> tempered [Element]		0	300			1	1	1
7	Computed Performance Data	averaging interval	time[second]	ValueProperty [Property] «> tempered [Element]		0	10			0	0	0
8	Computed Performance Data	burned calories	Kilocalorie	ValueProperty [Property] «> tempered [Element]		0	1000000			0	0	0
9	Computed Performance Data	cadence	RPM	ValueProperty [Property] «> tempered [Element]		0	300			1	1	1
10	Computed Performance Data	distance	distance[kilometre]	ValueProperty [Property] «> tempered [Element]		0	5000			2	2	2
11	RideSession	elapsed time	period duration[second]	ValueProperty [Property] «> tempered [Element]		0	2000000			0	0	0
12	RideSession	ending time	Date Time	ValueProperty [Property]								
13	Bike Settings	front gear	UnlimitedNatural	ValueProperty [Property] «> tempered [Element]		1	12					
14	Rider Personal Data	ftp	Watts per Kilogram	ValueProperty [Property] «> tempered [Element]		0	6			2	2	2
15	RideSession	ID	UnlimitedNatural	ValueProperty [Property]								
16	Measured Performance Data	incline	Degrees of Arc	ValueProperty [Property] «> tempered [Element]		-20	20			1	1	1
17	Bike Settings	incline	Degrees of Arc	ValueProperty [Property] «> tempered [Element]		-20	20			1	1	1
18	Rider Login Data	number of accounts	Integer	ValueProperty [Property]								
19	Ride History	number of sessions	UnlimitedNatural	ValueProperty [Property]								
20	Bike Settings	operational mode	Resistance Mode	ValueProperty [Property]								
21	App Account Data	password	String	ValueProperty [Property]								
22	Measured Performance Data	pedal position	plane angle[radian]	ValueProperty [Property] «> tempered [Element]		0	6.2832			3	3	3
23	Measured Performance Data	power	power[watt]	ValueProperty [Property] «> tempered [Element]		0	2000			0	0	0
24	Bike Settings	rear gear	UnlimitedNatural	ValueProperty [Property] «> tempered [Element]		1	12					
25	Rider Personal Data	rider name	String	ValueProperty [Property]								
26	Computed Performance Data	speed	speed[kilometre per hour]	ValueProperty [Property] «> tempered [Element]		0	300			1	1	1
27	RideSession	starting time	Date Time	ValueProperty [Property]								
28	RideSession	total calories	Kilocalorie	ValueProperty [Property] «> tempered [Element]		0	100000			0	0	0
29	RideSession	total distance	length[kilometre]	ValueProperty [Property] «> tempered [Element]		0	5000			2	2	2
30	Computed Performance Data	wats per kg	Watts per Kilogram	ValueProperty [Property] «> tempered [Element]		0	0			3	3	3
31	Rider Personal Data	weight	mass[kilogram]	ValueProperty [Property] «> tempered [Element]		0	500			1	1	1
32	Rider Performance Data	when measured	Date Time	ValueProperty [Property]								

Tables are useful beyond requirements.

This table shows design elements related to the data schema (model of information), including detailed metadata such as extent (range of permissible values), latency, accuracy, fidelity, and precision.



# Table Checklist

- ☐ Is the right layout selected?
- ☐ Is the purpose of the table understood and described in the model?
- ☐ Is the depicted set of metadata appropriate to meet the objective of the view?
- ☐ Is the context (set of packages) appropriate to meet the objective of the view?
- ☐ Is the table readable?
- ☐ Is the data in the table actually correct?
- ☐ Does the data in the table conform to your modeling standard?



# Matrices Are Useful

Matrices are more limited than tables. They are arranged in a **model element x model element** structure where the table cells indicate the presence or absence or relations.

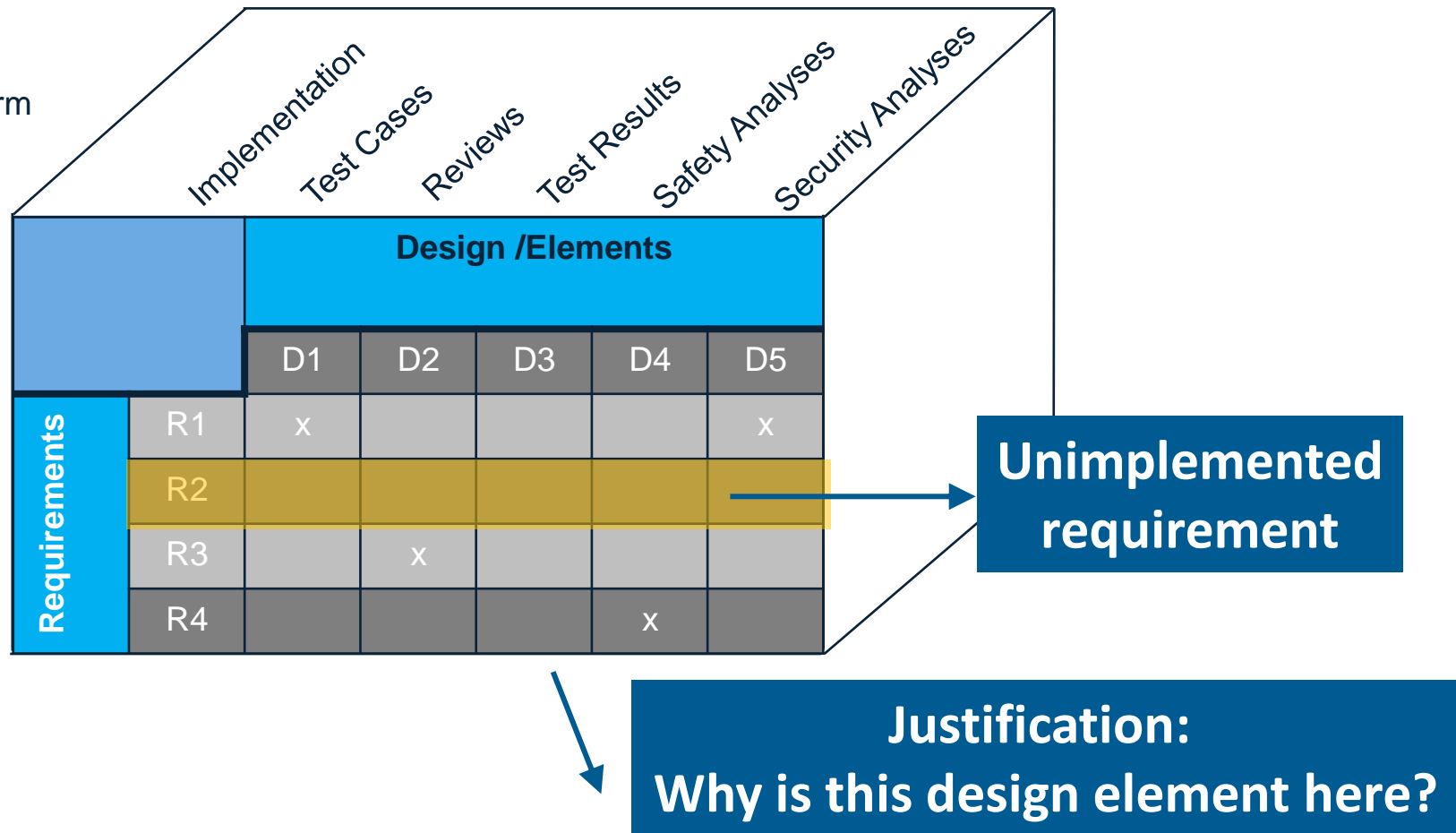
- Relations: Matrices provide an easily consumable view of the relations between sets of elements
  - Empty rows and columns can be particularly informative. (“Should there really be no relation there?”)
- Common Uses: Common uses of matrices in models are to:
  - Trace relations between sets of requirements. For example, stakeholder versus system requirements
  - Trace relations between use cases or user stories and requirements
  - Show relations between requirements and design
  - Show relations between requirements and test cases
  - Show allocations of requirements and features to design elements

# Connect Work Products with Traceability Links

## Benefits

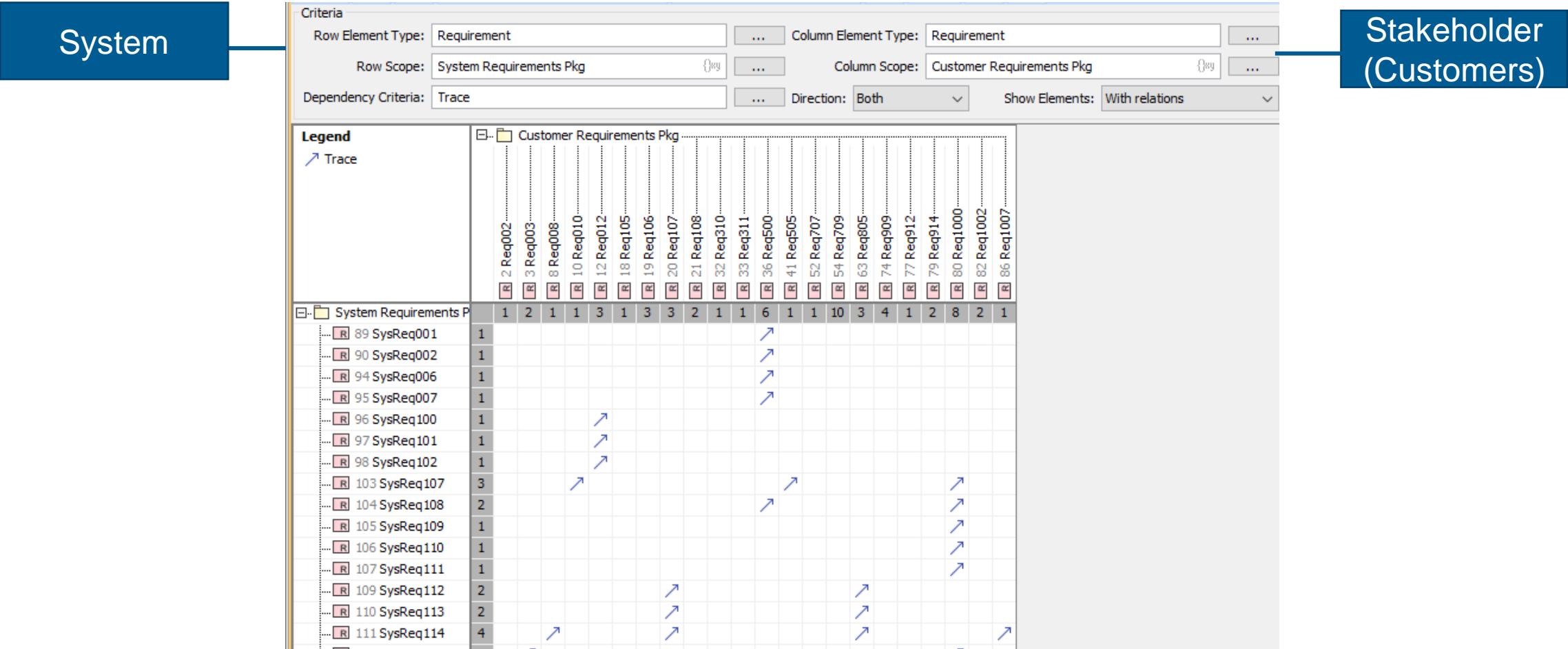
- **Requirements:** Helps ensure requirements are met and verified
- **Risk:** Lowers project risk
- **Audits:** Creates an audit trail
- **Consistency:**
  - Ensures consistency among work products
  - Helps manage consistency over the long term

Different kinds of model elements to potentially include in trace matrices



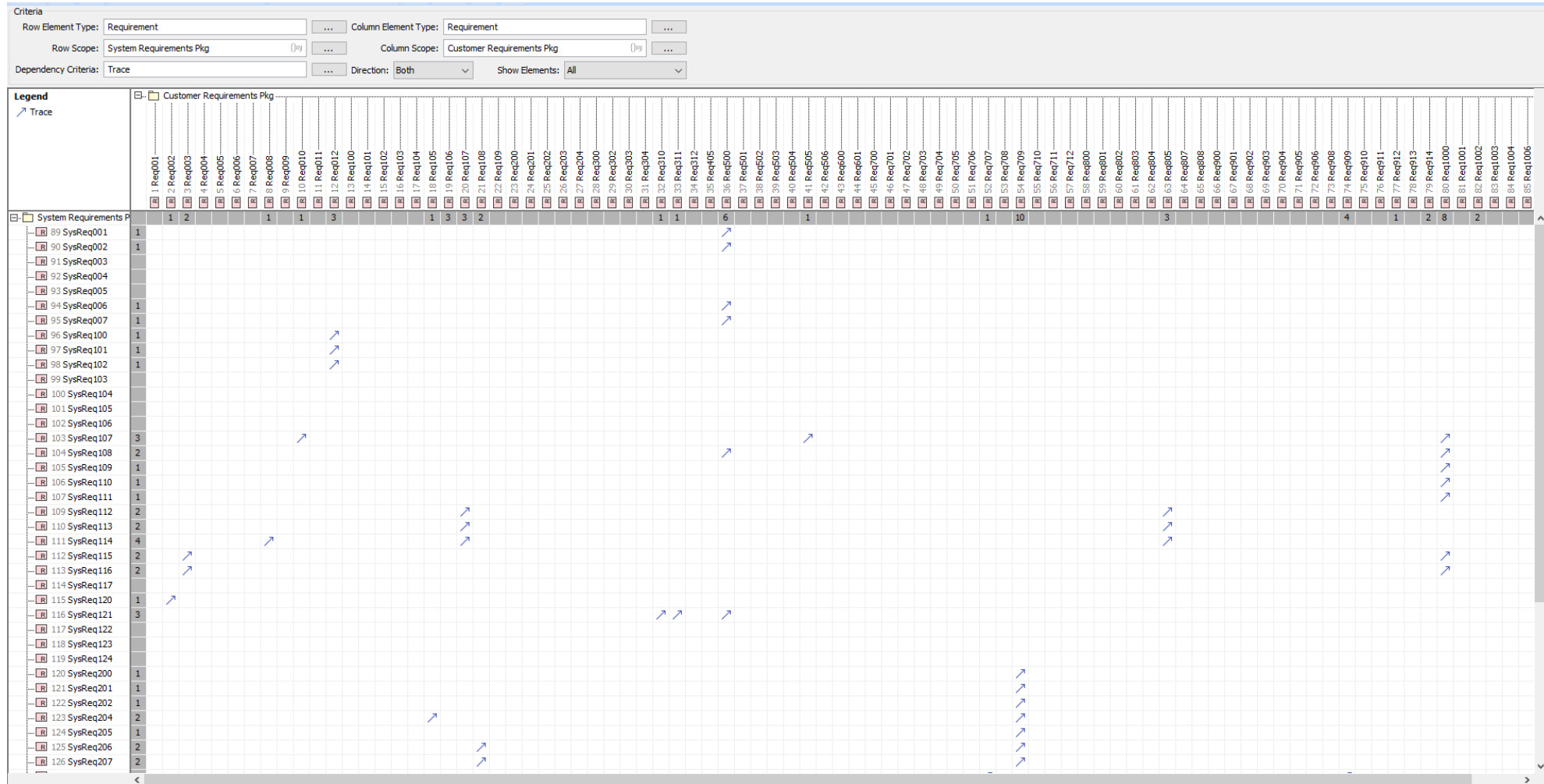
# Requirements Trace Matrix

- Matrices show specified kinds relations between sets of elements of different types from different scopes (packages)



# Trace relations between Stakeholder and System Requirements

- Empty columns indicate the absence of system requirements for a stated stakeholder need



# Example: Use Case – Requirements Trace Matrix

AirSurfaceControlSystem... X

From: UseCase Scope: AirSurfaceControlSystem

To: Requirement	Shut Down	Manage Power	Update Status	Configure System	Manage Data	Start Up	Control Air Surfaces
ErrorReq_29							ErrorReq_29
ErrorReq_30							ErrorReq_30
ErrorReq_31							ErrorReq_31
ErrorReq_32							ErrorReq_32
ErrorReq_33							ErrorReq_33
ErrorReq_34							ErrorReq_34
ErrorReq_35							ErrorReq_35
ErrorReq_36							ErrorReq_36
ErrorReq_37							
ConfigReq_0							
ConfigReq_1							
ConfigReq_2							
ConfigReq_3							
OtherReq_0							
OtherReq_1							
StartUpReq_0						StartUpReq_0	
StartUpReq_1						StartUpReq_1	
StartUpReq_2						StartUpReq_2	
StartUpReq_3						StartUpReq_3	
StartUpReq_4						StartUpReq_4	
StartUpReq_5						StartUpReq_5	
StartUpReq_6						StartUpReq_6	
SafetyReq_390197							SafetyReq_390197
SafetyReq_390198							SafetyReq_390198
SafetyReq_390199							SafetyReq_390199
SafetyReq_390200							SafetyReq_390200
SafetyReq_390201							SafetyReq_390201
SafetyReq_390202							SafetyReq_390202
SafetyReq_390203							SafetyReq_390203
SafetyReq_390204							SafetyReq_390204
SafetyReq_390205							SafetyReq_390205

Scope: RequirementsPkg

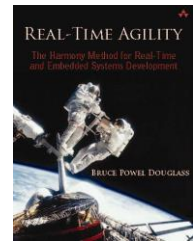
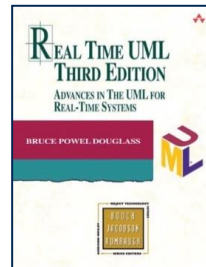
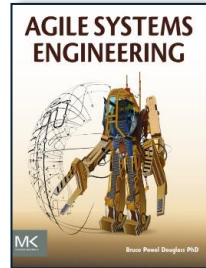
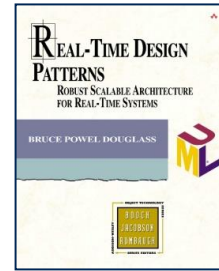
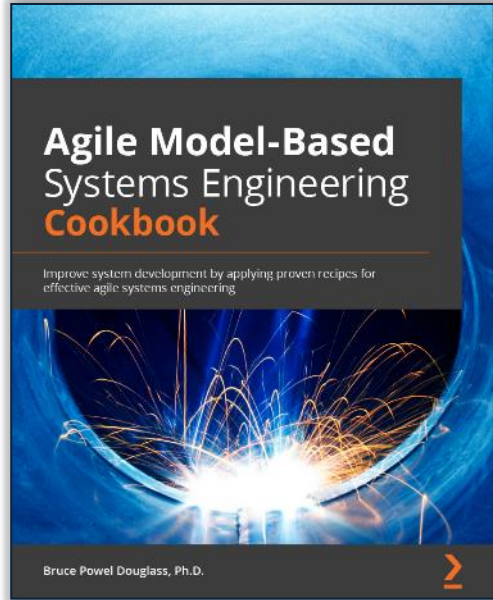


# Matrix Checklist

- ☐ Is the right layout selected?
- ☐ Is the purpose of the matrix understood?
- ☐ Is the set of source elements appropriate to meet the objective of the view?
- ☐ Is the set of target elements appropriate to meet the objective of the view?
- ☐ Is the set of relations shown among those elements appropriate?
- ☐ Is the context (set of source and target packages) appropriate to meet the objective of the view?
- ☐ Is the matrix readable?
- ☐ Is the data in the matrix actually correct?
  - ☐ Are there missing relations?
  - ☐ Are there relations that shouldn't be there?
- ☐ Do the relations in the table conform to your modeling standard?



# Thank You



**Bruce Douglass, Ph.D.**

[bdouglass@mitre.org](mailto:bdouglass@mitre.org)

@IronmanBruce

<https://www.linkedin.com/in/bruce-douglass-phd/>



**MITRE** | DE Platform

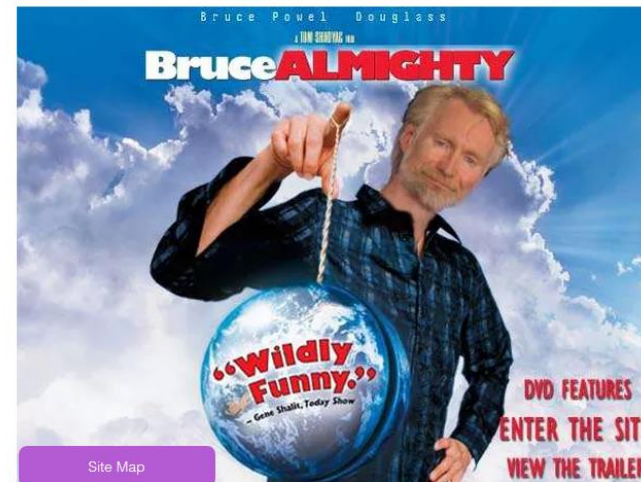
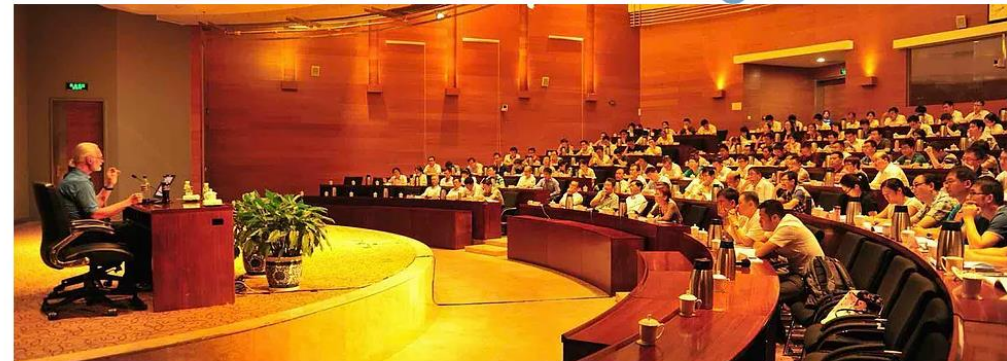
© 2020, 2021 THE MITRE CORPORATION. ALL RIGHTS RESERVED.

Bruce Powel Douglass, Ph.D. [Log In](#)

[Content](#) [Services](#) [Public Interest](#) [Blog](#) [What's New](#) [Forum](#) [About](#) [Comments](#) [Site Map](#) [Geekosphere](#) [Members](#)

## Real-Time Agile Systems and Software Development

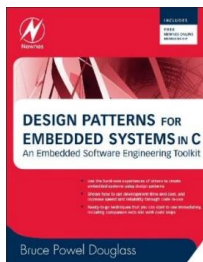
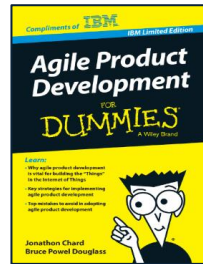
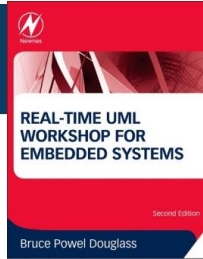
Welcome to [www.bruce-douglass.com](http://www.bruce-douglass.com)



You've found yourself on [www.bruce-douglass.com](http://www.bruce-douglass.com), my web site on all things real-time and embedded.

On this site you will find papers, presentations, models, forums for questions / discussions, and links (lots of links) to areas of interest, such as

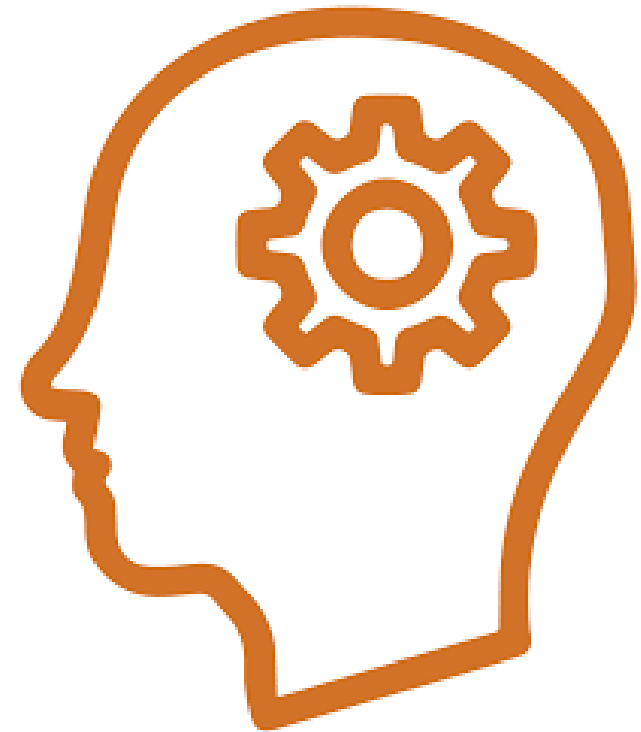
- Developing Embedded Software
- Model-Driven Development for Real-Time Systems
- Model-Based Systems Engineering
- Safety Analysis and Design
- Agile Methods for Embedded Software
- Agile Methods for Systems Engineering
- The Harmony agile Model-Based Systems Engineering process
- The Harmony agile Embedded Software Development process
- Models and profiles I've developed and authored
- List and links to many of my books.



# Knowledge Assessment

## ON YOUR OWN TIME

Review diagrams representing a system and respond to questions about its content and meaning





# Review the following diagrams and views

- These diagrams represent the visualization of content of a systems engineering model of a microwave oven.
- Review the diagrams and the elements they visualize
- Answer the following questions and critique diagrams where indicated
- Recommendation
  - Refer to the SysML checklists when answering questions in the Knowledge Assessment

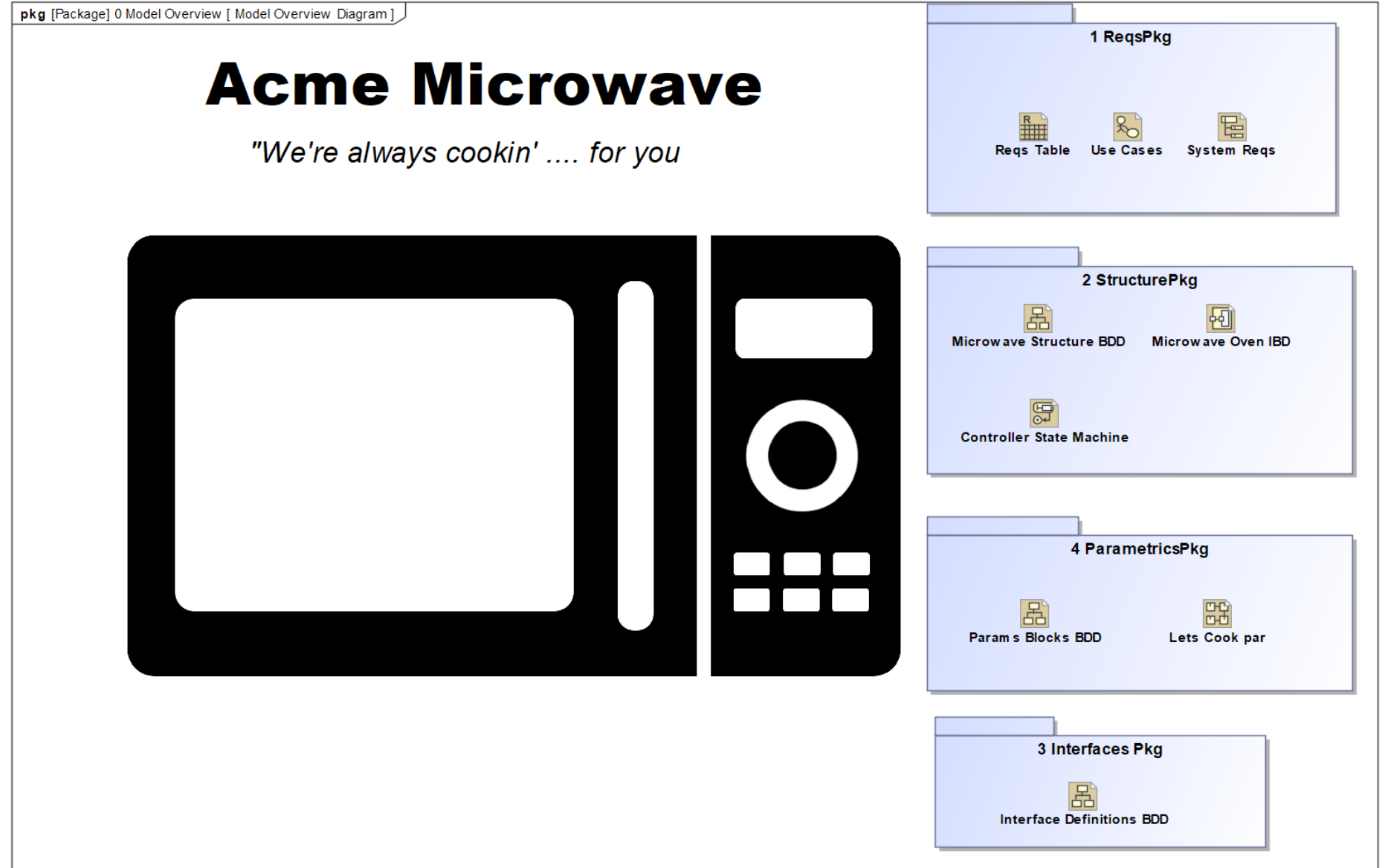
# Diagram types (List to be used for all 15 diagram type questions)

- Requirements diagram
- Use case diagram
- Package diagram
- Block definition diagram
- Internal block diagram
- Sequence diagram
- Activity diagram
- State diagram
- Parametric diagram
- Requirements table
- User-defined table
- Allocation matrix
- Trace matrix
- User-defined matrix
- Other diagram or view

# Knowledge Assessment 1



All diagrams in this Knowledge Assessment come from the same model.



# Questions KA 1

1. What kind of diagram is this? Select from the list\*.
2. What is the purpose or usage of this diagram? Select the best answer.
  - a) Shows the outcome of system verification activities
  - b) Links behavior to structural design elements
  - c) Lists behavioral elements of the model
  - d) Provides an overview of the model
3. Are there any issues with this diagram? Select all that apply.
  - a) Wrong kind of diagram used
  - b) Missing purpose statement
  - c) Missing scope statement
  - d) Using images in inappropriate on diagram
  - e) Diagrams shouldn't be shown inside of packages

\* See list of diagram types [here](#)

# Knowledge Assessment 2

req [Package] 1 ReqsPkg [ System Reqs ]

«requirement»

**Req 1**

Id = "1"  
Text = "The cook shall be able to set the power level from 1 to 10, where 1 = 150W delivery and 10 = 1500W delivery."

«requirement»

**Req 2**

Id = "2"  
Text = "The cook shall be able to set cook time in minutes from 1 to 10 with a single key press."

«requirement»

**Req 3**

Id = "3"  
Text = "The cook shall be able to set cook time in mm:ss where ss is seconds in the range of 0-59 and mm is minutes in the range of 0 to 99."

«requirement»

Id = "7"  
Text = "If the door is opened during cooking, the microwave emission shall halt and the cooking time shall pause. "

«requirement»

**Req 4**

Id = "4"  
Text = "The system shall alert when the cook timer has elapsed."

«requirement»

**Req 5**

Id = "5"  
Text = "The system shall only emit microwaves if the door is closed."

«requirement»

**Req 6**

Id = "6"  
Text = "Cooking may only begin if the door is closed."

«requirement»

Id = "8"  
Text = "The Cook shall be able to resume paused cooking with a single keystroke but only if the door is closed. "

«requirement»

**Req 10**

Id = "9"  
Text = "The cooking shall start quickly after the cook button is pressed."

«requirement»

**Req 11**

Id = "10"  
Text = "The microwave oven will be stylish and attractive."

«requirement»

**Req 12**

Id = "11"  
Text = "The system won't be too heavy to carry."

# Questions KA 2

1. What kind of diagram is this? Select from the list.
2. What is the purpose or usage of this diagram? Select the best answer
  - a) Show design realization
  - b) Show requirements, their properties and relations
  - c) Link use cases and requirements
  - d) Connect constraint properties to value properties
3. Are there any issues with this diagram? Select all that apply.
  - a) Missing mission statement comment
  - b) Missing relations to design
  - c) Two requirements are unnamed
  - d) Req 11 and Req 12 are poor requirements (inexact, ambiguous and untestable)
  - e) The use of the *shall* keyword is wrong

# Knowledge Assessment 3

Criteria		
Scope (optional): 1 ReqsPkg <span>{ xy}</span> <span>...</span> Filter: <span>Y</span>		
#	△ Name	Text
1	<span>R</span> 1 Req 1	The cook shall be able to set the power level from 1 to 10, where 1 = 150W delivery and 10 = 1500W delivery.
2	<span>R</span> 2 Req 2	The cook shall be able to set cook time in minutes from 1 to 10 with a single key press.
3	<span>R</span> 3 Req 3	The cook shall be able to set cook time in mm:ss where ss is seconds in the range of 0-59 and mm is minutes in the range of 0 to 99.
4	<span>R</span> 4 Req 4	The system shall alert when the cook timer has elapsed.
5	<span>R</span> 5 Req 5	The system shall only emit microwaves if the door is closed.
6	<span>R</span> 6 Req 6	Cooking may only begin if the door is closed.
7	<span>R</span> 7	If the door is opened during cooking, the microwave emission shall halt and the cooking time shall pause.
8	<span>R</span> 8	The Cook shall be able to resume paused cooking with a single keystroke but only if the door is closed.
9	<span>R</span> 9 Req 10	The cooking shall start quickly after the cook button is pressed.
10	<span>R</span> 10 Req 11	The microwave oven will be stylish and attractive.
11	<span>R</span> 11 Req 12	The system won't be too heavy to carry.

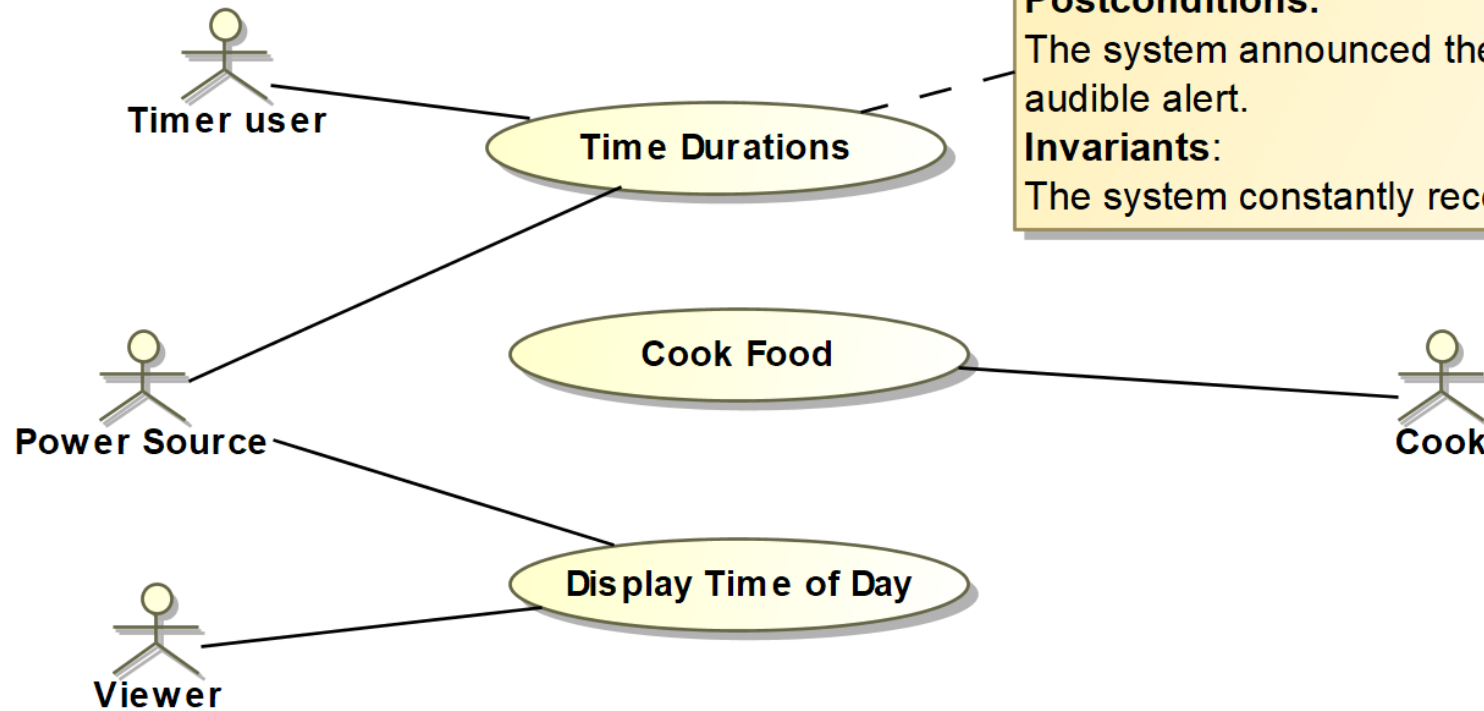
# Questions KA 3

1. What kind of diagram is this? Select from the list.
2. What is the purpose or usage of this view? Select the best answer.
  - a) Show design realization
  - b) Show requirements and their properties
  - c) Trace between use cases and requirements
  - d) Connect constraint properties to value properties
3. Are there any issues with this diagram? Select all that apply.
  - a) Missing relations to design
  - b) Two requirements are unnamed
  - c) Req 11 and Req 12 are poor requirements (inexact, ambiguous and untestable)
  - d) The use of the *shall* keyword is wrong
  - e) The requirement ids are missing



# Knowledge Assessment 4

uc [Package] 1 ReqsPkg [ Use Cases ]



**Name:** Time Durations

**Purpose:**

All the user to set a count-down timer for cooking and other activities.

**Precondition:**

The system is on.

**Postconditions:**

The system announced the elapse of the count-down time via audible alert.

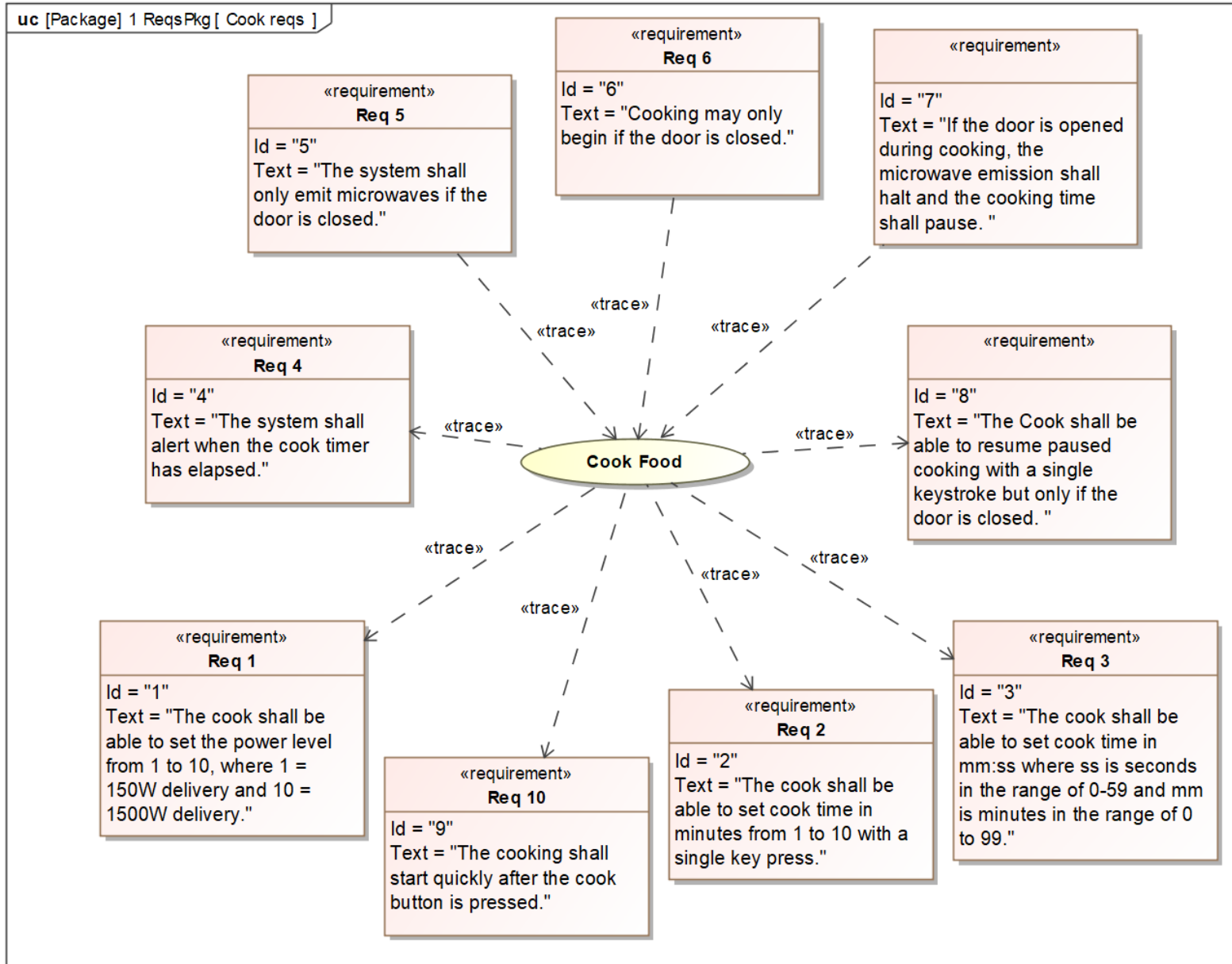
**Invariants:**

The system constantly receives wall power.

# Questions KA 4

1. What kind of diagram is this? Select from the list.
2. What is the purpose or usage of this diagram? Select the best answer.
  - a) Show use cases, actors, and their relations
  - b) Show design realization
  - c) Show the requirements related to a particular use case
  - d) Show trace relations between use cases and requirements
  - e) Show system parametrics
  - f) Show algorithmic behavior
3. Are there any issues with this diagram? Select all that apply.
  - a) Wrong relation is used between use cases and actors
  - b) Non-noun actor names
  - c) Non-verb use case names
  - d) No diagram mission statement
  - e) One use case description exposed on the diagram; do the others have descriptions?
  - f) Missing relation: Surely the power source contributes to the cooking of food as well?



























# Knowledge Assessment 5



# Questions KA 5

1. What kind of diagram is this? Select from the list.
2. What is the purpose or usage of this diagram? Select the best answer.
  - a) Show use cases, actors, and their relations
  - b) Show design realization
  - c) Show the requirements related to a particular use case
  - d) Show trace relations between use cases and requirements
  - e) Show system parametrics
  - f) Show algorithmic behavior
3. Are there any issues with this diagram? Select all that apply.
  - a) No diagram mission statement
  - b) Wrong relation between requirements and use case used
  - c) Traces from Req 5, Req 6 and the unnamed required (id 7) trace relations are in the wrong direction
  - d) Traces to Req 1, Req 2, Req 3, Req 4, Req 10 and the unnamed requirement (id 8) are in the wrong direction
  - e) Missing links to design elements
  - f) Non-verb use case name

# Knowledge Assessment 6

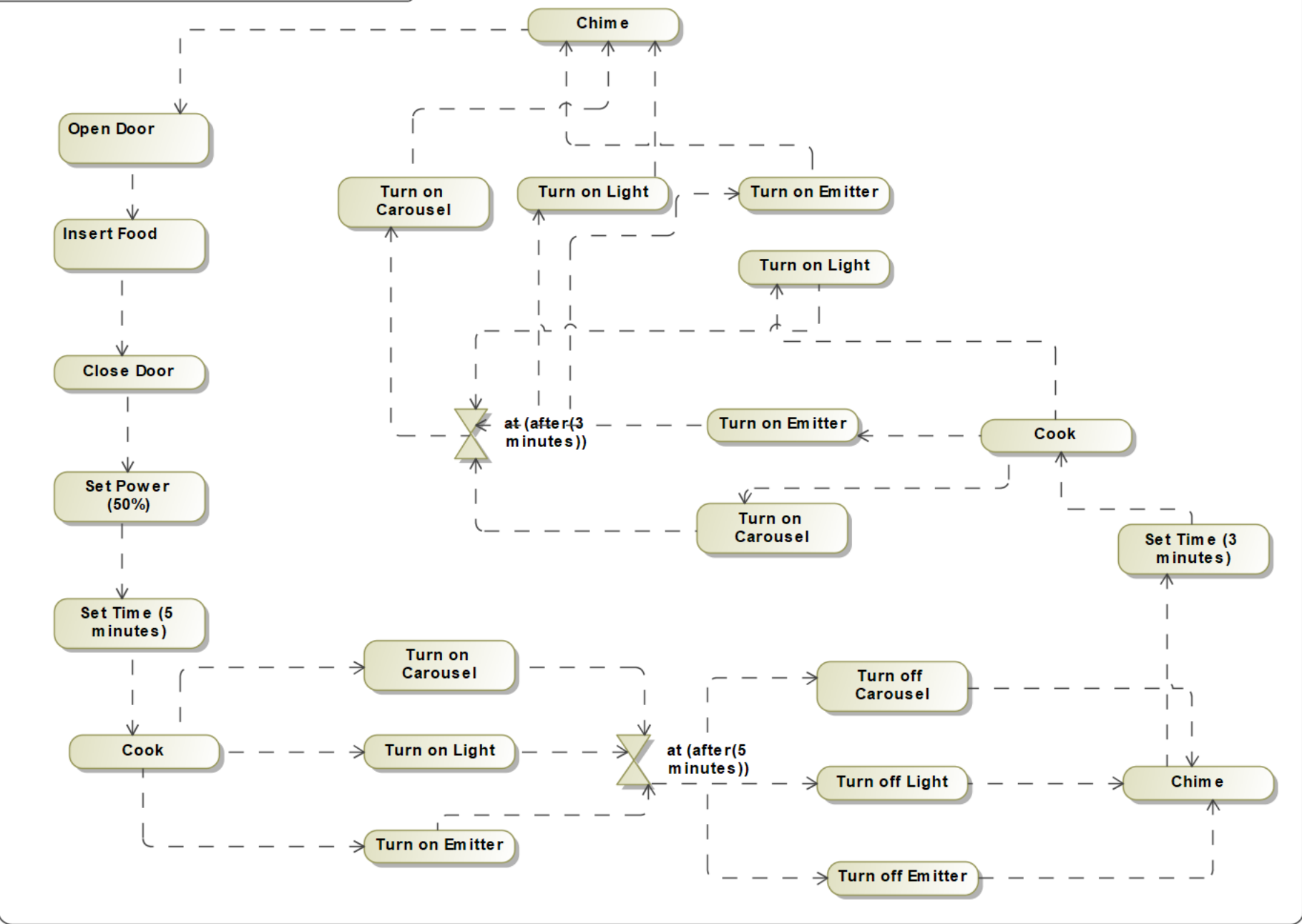
<b>Legend</b>  Trace		 1 ReqsPkg											
			1 Req 1	2 Req 2	3 Req 3	4 Req 4	5 Req 5	6 Req 6	7	8	9 Req 10	10 Req 11	11 Req 12
													
 1 ReqsPkg			1	1	1	1	1	1	1	1			
 Cook Food		9											
 Display Time of Day													
 Time Durations													

# Questions KA 6

1. What kind of diagram is this? Select from the list.
2. What is the purpose or usage of this view? Select the best answer.
  - a) Show use cases, actors, and their relations
  - b) Show design realization
  - c) Show the requirements related to a particular use case
  - d) Show trace relations between use cases and requirements
  - e) Show system parametrics
  - f) Show algorithmic behavior
3. Are there any issues with this diagram? Select all that apply.
  - a) You can't create trace relations between use cases and requirements
  - b) Some requirements are unnamed
  - c) There are no requirements for two of the use cases
  - d) Some traces are in the wrong direction
  - e) Requirements text and other metadata are not shown
  - f) Relations to actors are not shown

# Knowledge Assessment 7

act [Activity] Cook Mean Activity [ Cook Mean Activity ]

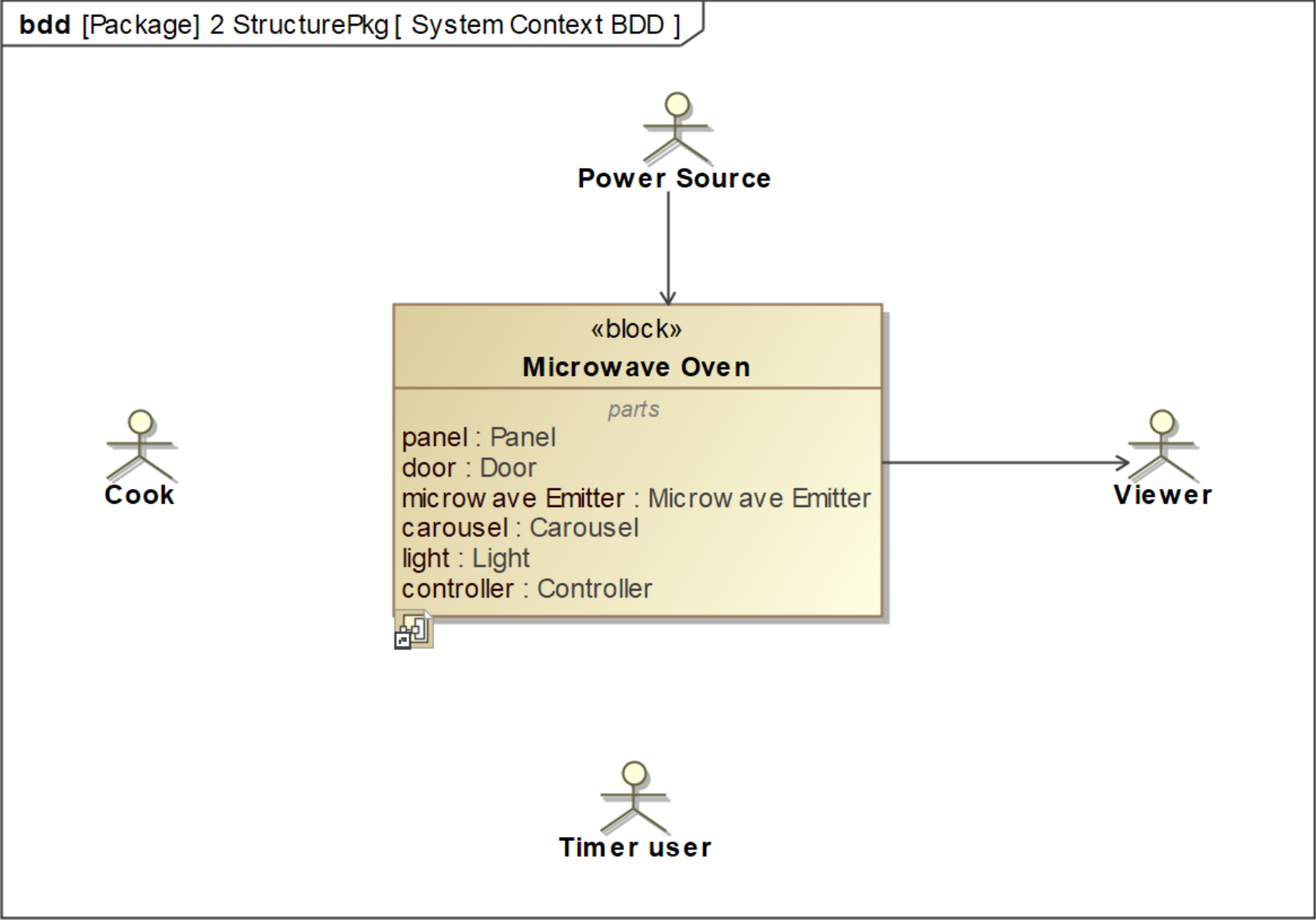


# Questions KA 7

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show microwave capabilities
  - b) Illustrate microwave parametrics
  - c) Show the behavioral flow for cooking food
  - d) Show microwave oven state behavior for cooking food
  - e) Describe structural design model types
  - f) Show system structural context
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Wrong kind of arrow used
  - c) Unnecessary line crossing
  - d) Missing initial flow
  - e) Open Door activity can never start
  - f) Multiple flows exiting an action are not allowed



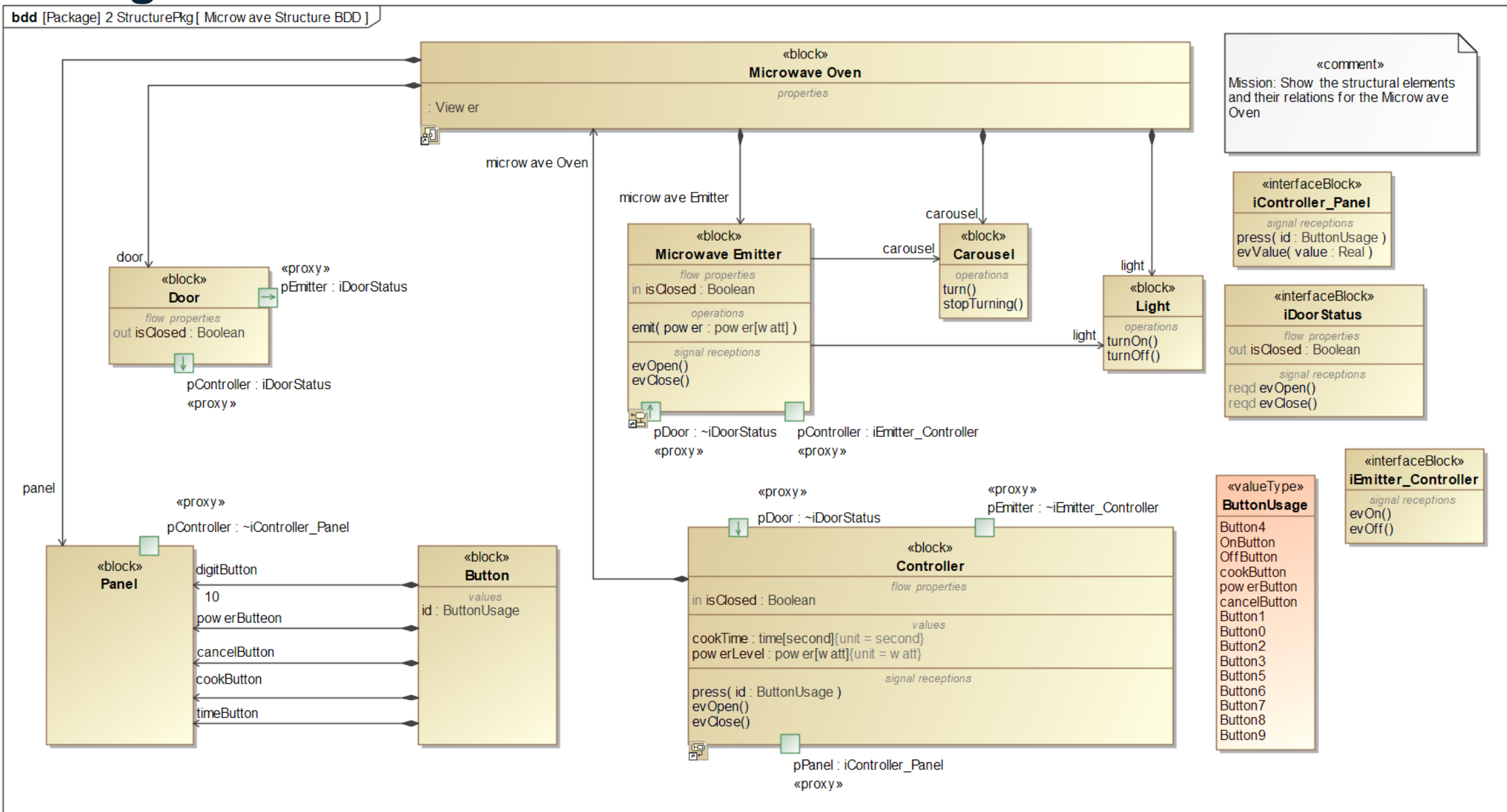
# Knowledge Assessment 8



# Questions KA 8

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show microwave capabilities
  - b) Show the constraint blocks and elements related to the parametric analysis
  - c) Show the behavioral flow for cooking food
  - d) Show microwave oven state behavior
  - e) Describe structural design model types and their relations
  - f) Show system in its operational context
3. Are there any issues with this diagram? Select all that apply.
  - a) No missions statement comment
  - b) Block shouldn't show parts
  - c) Wrong relation used to actors
  - d) Poor actor choice
  - e) Not all actors are connected

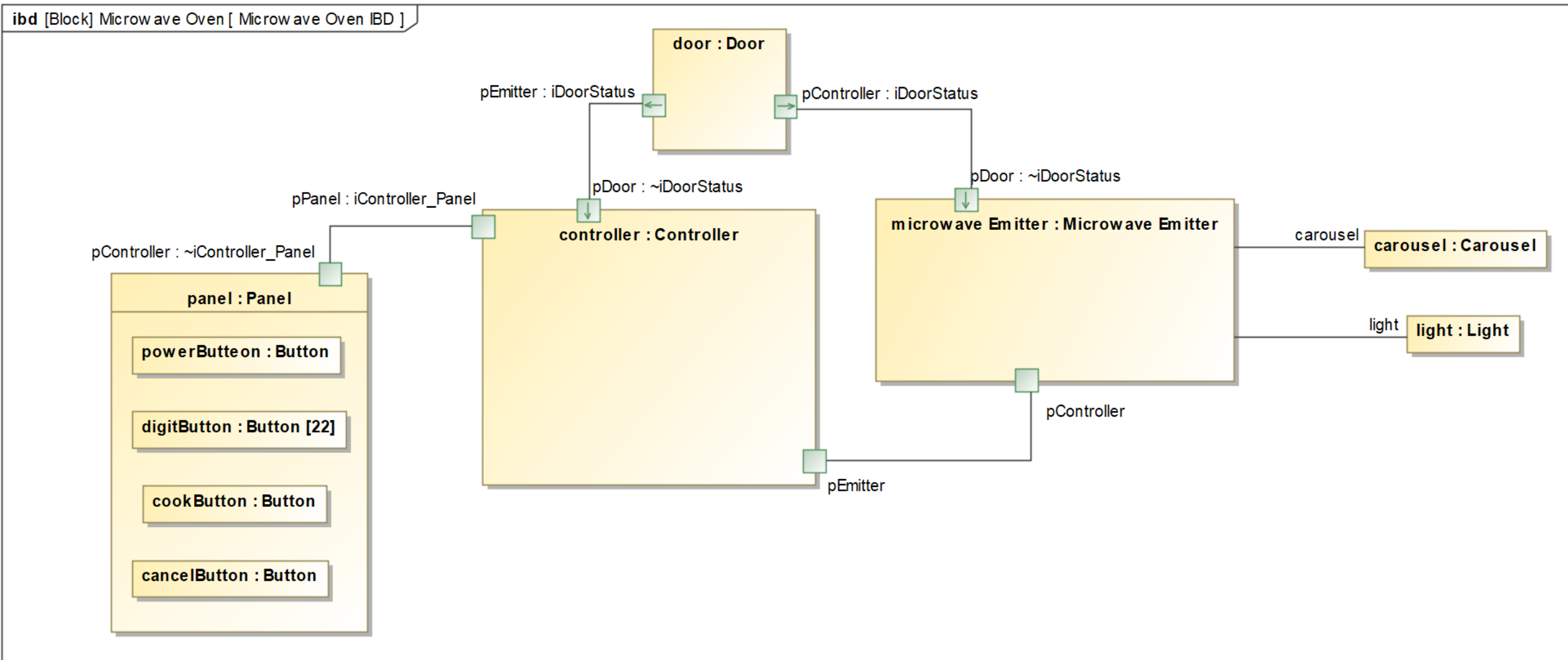
# Knowledge Assessment 9



# Questions KA 9

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show microwave capabilities
  - b) Show the constraint blocks and elements related to the parametric analysis
  - c) Show the behavioral flow for cooking food
  - d) Show microwave oven state behavior
  - e) Describe structural design model types and their relations
  - f) Show system in its operational context
3. Are there any issues with this diagram? Select all that apply.
  - a) No missions statement comment
  - b) Blocks shouldn't show values
  - c) Wrong block relation types used
  - d) Composition relation between Controller and Microwave oven is backwards
  - e) Composition relations between Button and Panel are all backwards
  - f) Interface blocks shouldn't be shown
  - g) Ports are not connected

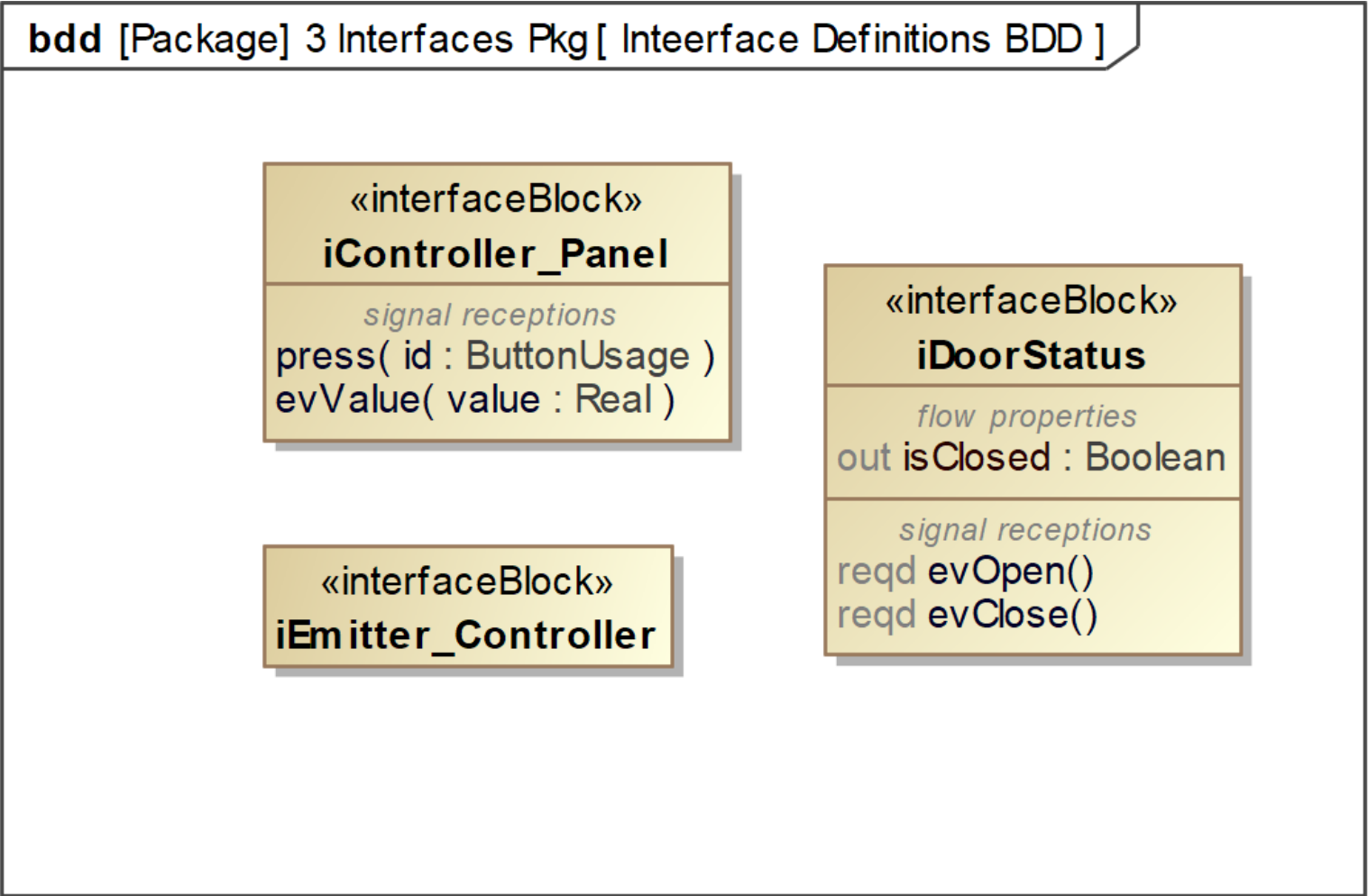
# Knowledge Assessment 10



# Questions KA 10

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use?
  - a) Show microwave capabilities
  - b) Show the constraint blocks and elements related to the parametric analysis
  - c) Show the behavioral flow for cooking food
  - d) Show how the parts of the microwave oven connect
  - e) Describe structural design model types and their relations
  - f) Show system in its operational context
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Part values not shown
  - c) Ports pEmitter and pController are untyped
  - d) ~ character not allowed in port type names
  - e) Multiplicity on digitButtons (22) doesn't match the previous BDD (10)
  - f) Carousel and light are connected without using ports

# Knowledge Assessment 11

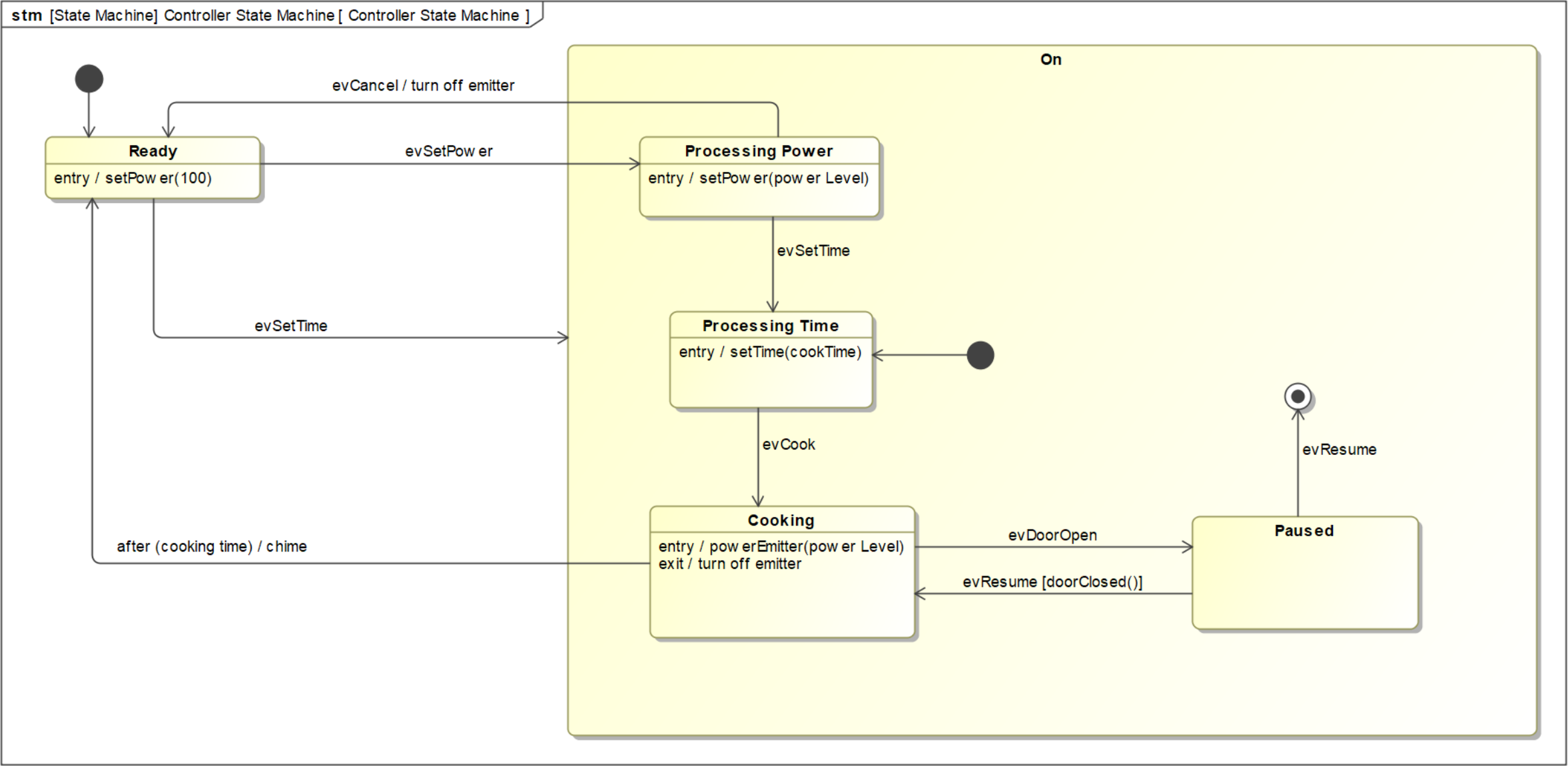


# Questions KA 11

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show system interfaces
  - b) Show microwave capabilities
  - c) Show the constraint blocks and elements related to the parametric analysis
  - d) Show internal structure of a block
  - e) Describe structural design model types and their relations
  - f) Show system in its operational context
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Relations among blocks not shown
  - c) Flow properties cannot be shown on this diagram type
  - d) iEmitter\_Controller has no signals, operations, or flows.
  - e) Directionality of iController\_Panel signal receptions not specified



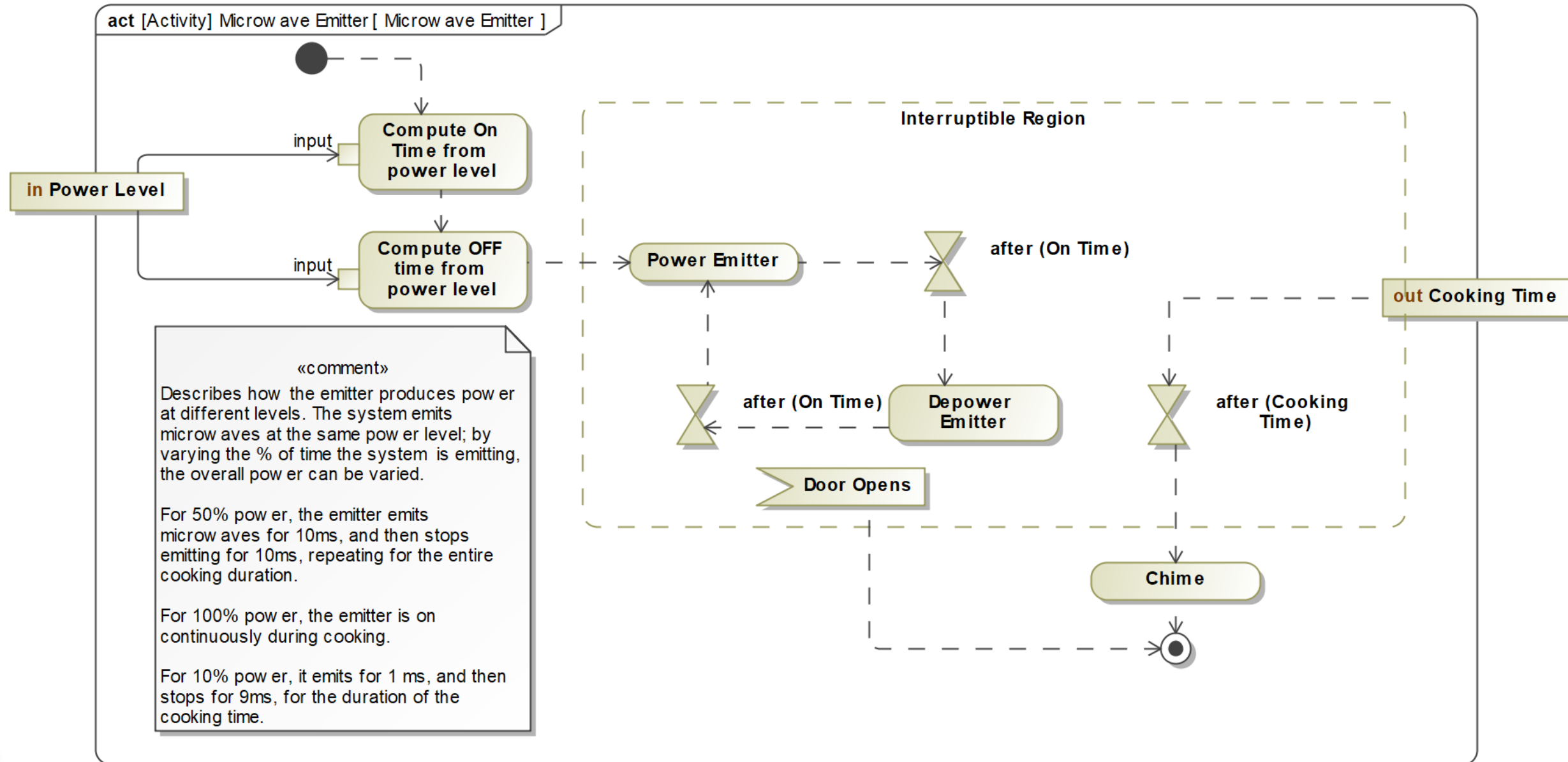
# Knowledge Assessment 12



# Questions KA 12

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show controller block activities
  - b) Show controller block structure
  - c) Show controller block state behavior
  - d) Show controller block parametrics
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Missing initial state
  - c) After being Paused, there are two evResume transitions; which one would be taken?
  - d) evCook doesn't have a guard to ensure the door is closed
  - e) Transitioning via evResume to Final State contradicts requirements
  - f) You can only cancel from the Processing Power state not from Processing Time, Cooking or Paused
  - g) Missing final state
  - h) States cannot contain states

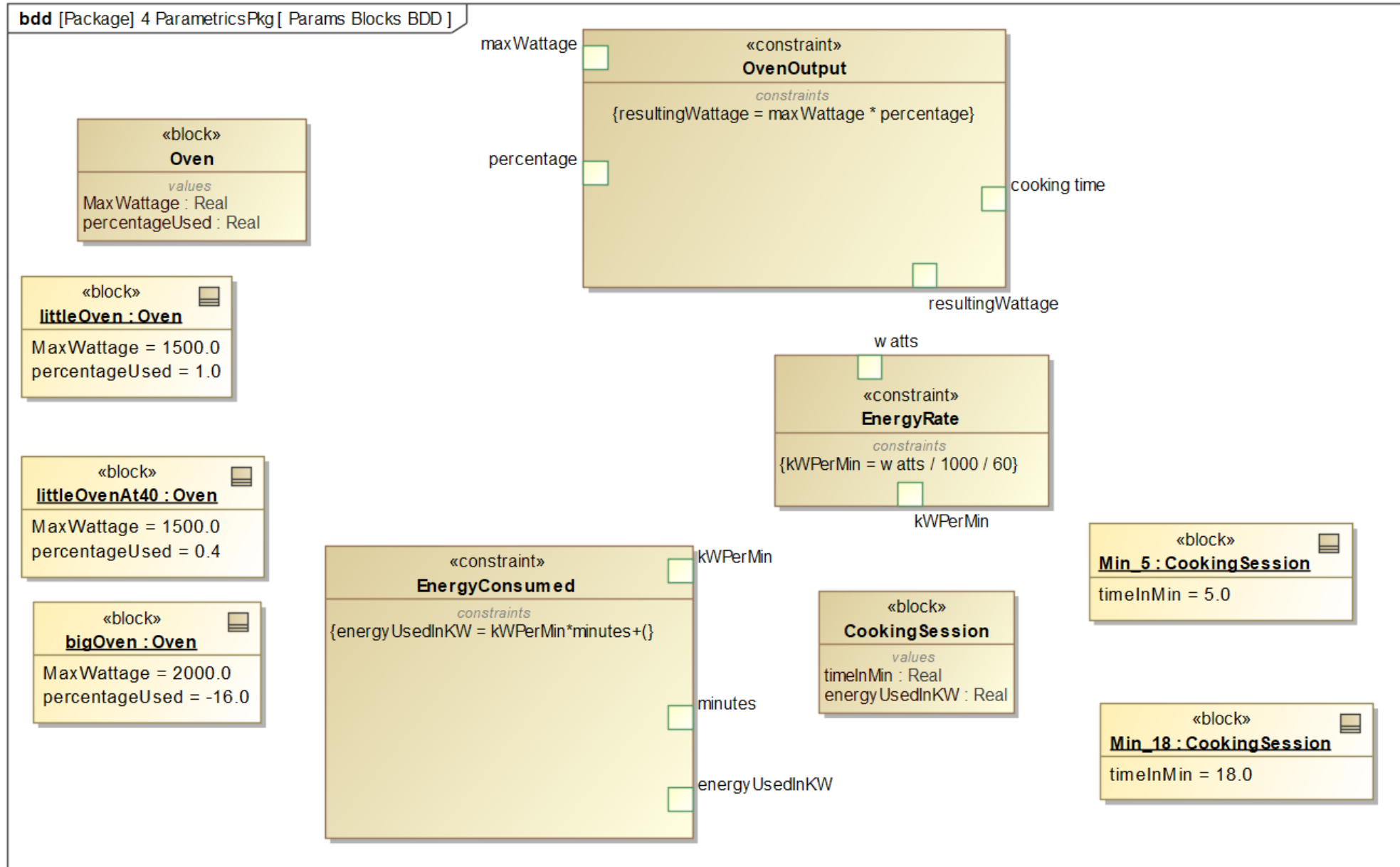
# Knowledge Assessment 13



# Questions KA 13

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show the activity behavior of emitting microwaves with varying power levels
  - b) Show the state behavior of emitting microwaves with varying power levels
  - c) Show the operations of the microwave emitter
  - d) Show the value properties of the microwave emitter
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Missing initial node
  - c) Missing activity final node
  - d) Activity parameters are not allowed on the diagram frame
  - e) Power Emitter action can never happen because there are two mutually exclusive input control flows
  - f) Cooking Time activity parameter is shown as an output parameter but it is actually an input
  - g) Wrong flow type used between actions
  - h) When the door open, the activity terminates, but it might leave the emitter ON if it was executing that action

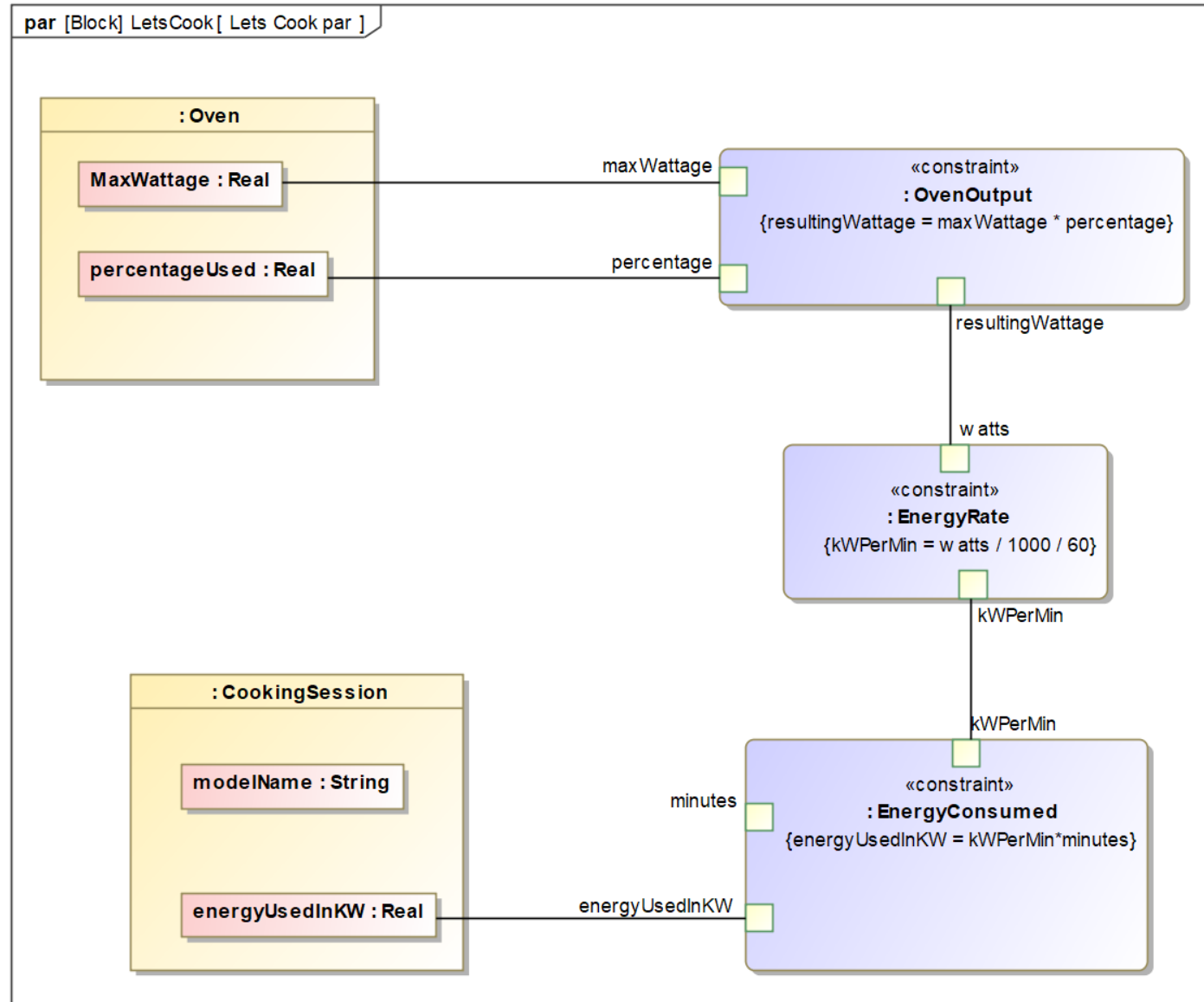
# Knowledge Assessment 14



# Questions KA 14

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show system interfaces
  - b) Show microwave capabilities
  - c) Show the constraint blocks and elements related to the parametric analysis
  - d) Show internal structure of a block
  - e) Describe structural design model types
  - f) Show constraint relations related to power use
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement
  - b) Cooking time parameter for OvenOutput constraint block isn't used
  - c) Constraint for EnergyConsumed constraint block ill-formed
  - d) Constraint properties are not connected
  - e) percentageUsed slot for bigOven instance specification has the value -16.0. What does that even mean?

# Knowledge Assessment 15



# Questions KA 15

1. What kind of diagram is this? Select from the list.
2. What is its purpose of use? Select the best answer.
  - a) Show system interfaces
  - b) Show microwave capabilities
  - c) Show the constraint blocks and elements related to the parametric analysis
  - d) Show internal structure of a block
  - e) Describe structural design model types
  - f) Show constraint relations related to power use
3. Are there any issues with this diagram? Select all that apply.
  - a) No mission statement comment
  - b) Energy Consumed constraint property has a minutes constraint parameter that isn't connected
  - c) Cooking Session part doesn't have a relevant value property to provide minutes of cooking (although it does name a model name as a string)
  - d) Wrong kind of relation used between constraint blocks
  - e) Proxy ports should be used instead of constraint parameters
  - f) Constraint blocks should be used rather than constraint properties



# Knowledge Assessment Answers

## 1. KA 1

1. Package Diagram
2. D
3. B, C

## 2. KA 2

1. Requirements Diagram
2. B
3. A, C, D

## 3. KA 3

1. Requirements Table
2. B
3. B, C

## 1. KA 4

1. Use Case Diagram
2. A
3. D, E, F

## 2. KA 5

1. Use Case Diagram
2. C
3. A, C

## 3. KA 6

1. Trace Matrix
2. D
3. B, C, D

# Knowledge Assessment Answers

## 1. KA 7

1. Activity Diagram
2. C
3. A, C, D, E

## 2. KA 8

1. Block Definition Diagram
2. F
3. A, E

## 3. KA 9

1. Block Definition Diagram
2. E
3. D, E

## 1. KA 10

1. Internal Block Diagram
2. D
3. A, C, E

## 2. KA 11

1. Block Definition Diagram
2. A
3. A, D, E

## 3. KA 12

1. State Matrix
2. C
3. A, C, D, E, F

# Knowledge Assessment Answers

## 1. KA 13

1. Activity Diagram
2. A
3. E, F, H

## 2. KA 14

1. Block Definition Diagram
2. C
3. A, B, C, E



## 3. KA 15

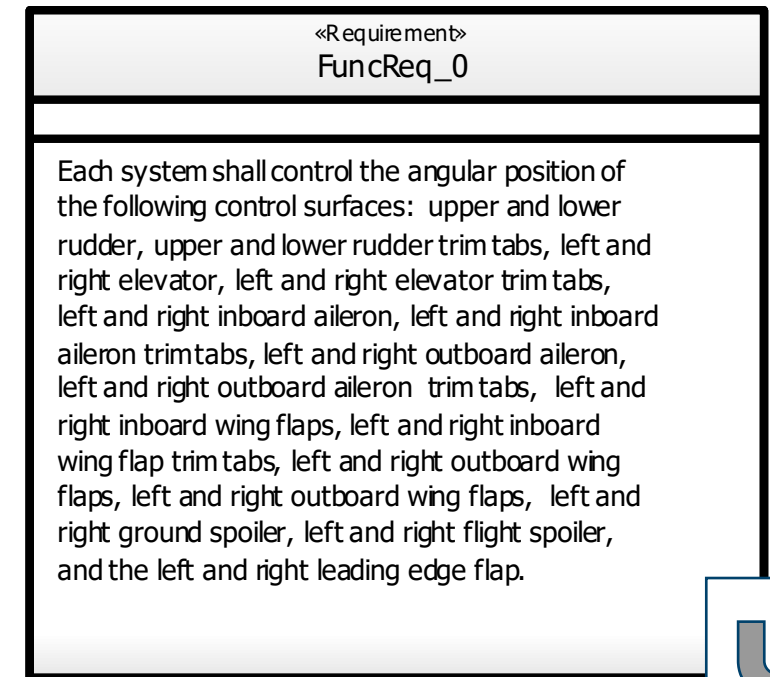
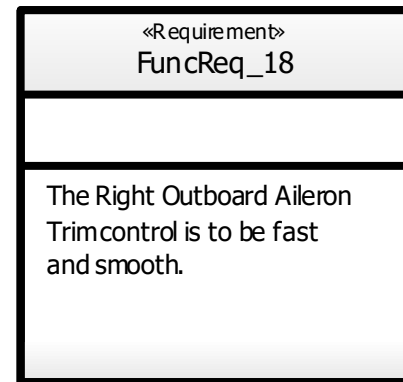
1. Parametric Diagram
2. F
3. A, B, C



## Answers

# Requirements Diagram Answer

- The Contain relations don't seem to make sense
- Missing mission statement for diagram
- Some requirements are on top of others (harder to read)
- The use of <<verify>> is wrong
- Mixed mission, since requirements exist about functionality, hydraulic, and compliance with DO-178 standard
- This requirement is not in canonical form and isn't clear or testable: 
- This one isn't a single testable statement 

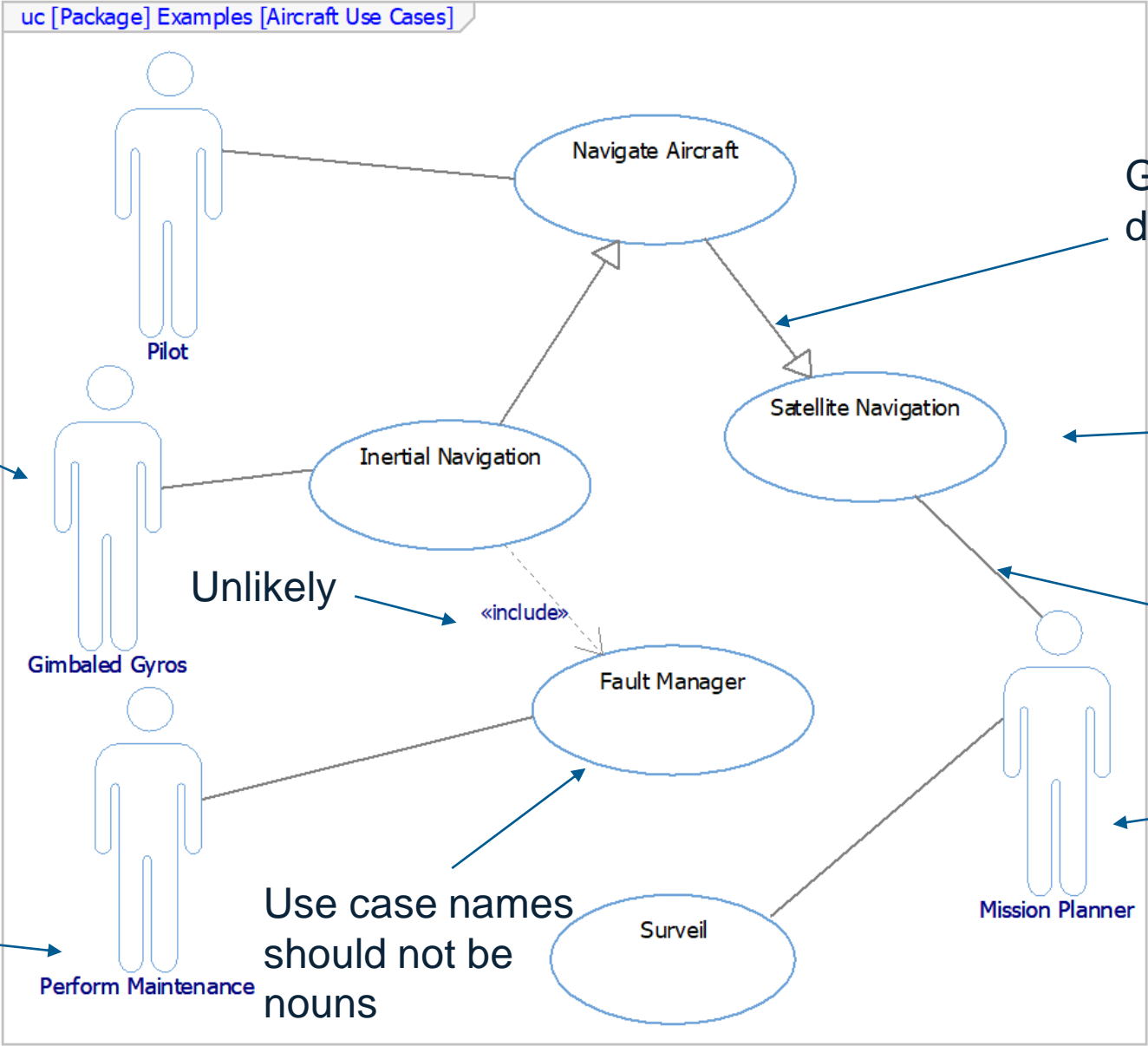


# Use Case Diagram Answer

No diagram mission statement

Likely part of the System and not an actor

Actors should have noun named, not verbs



Generalization direction incorrect

Missing actor. "Satellite" or "GPS Satellite" maybe?

Unlikely

Poor actor choice. "Missions Ops" actor maybe?

Use case names should not be nouns



Type not fully exposed.  
Is that ok?

Did you mean  
/Composition?



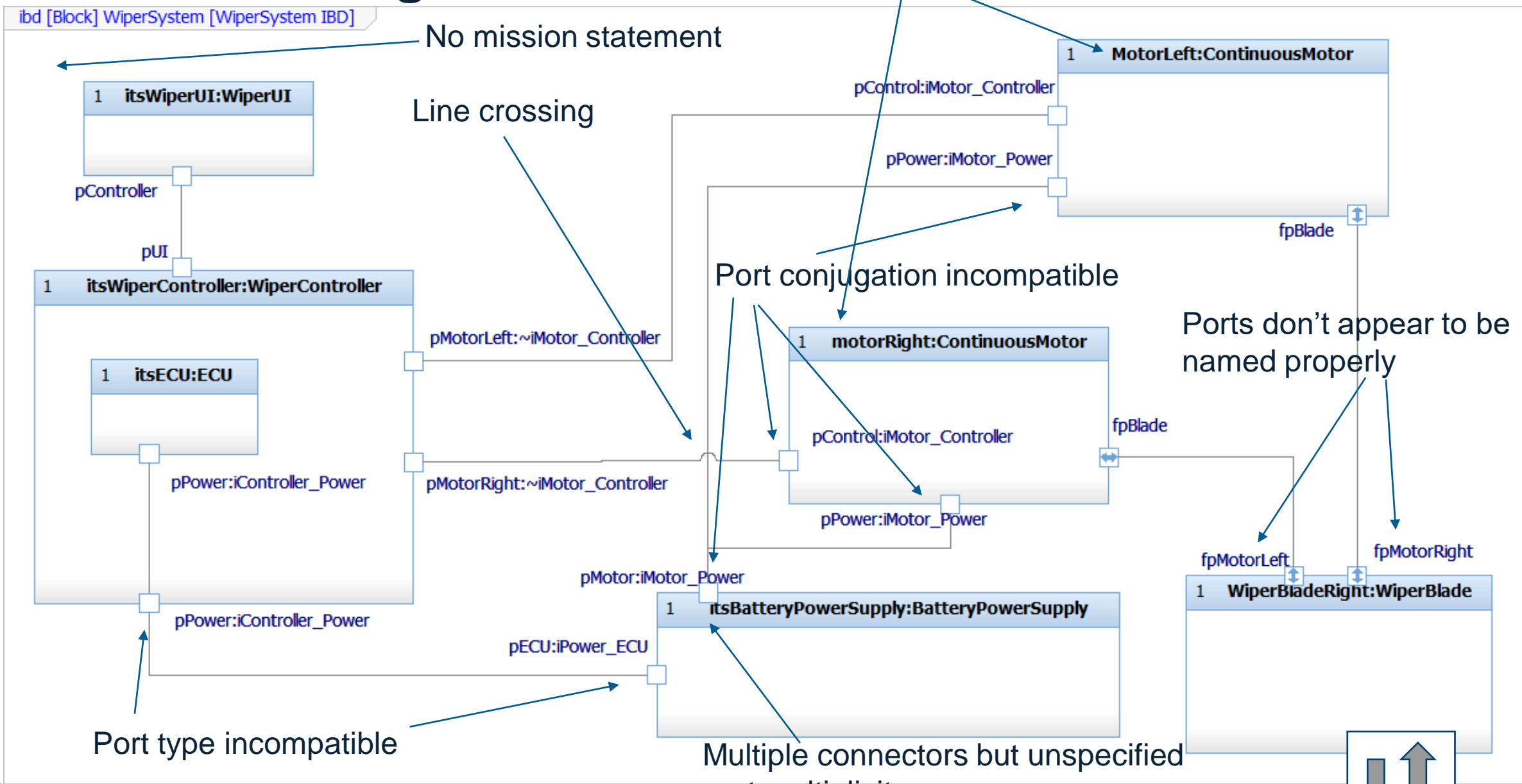
## Line crossing

Features not shown.  
Is that ok?



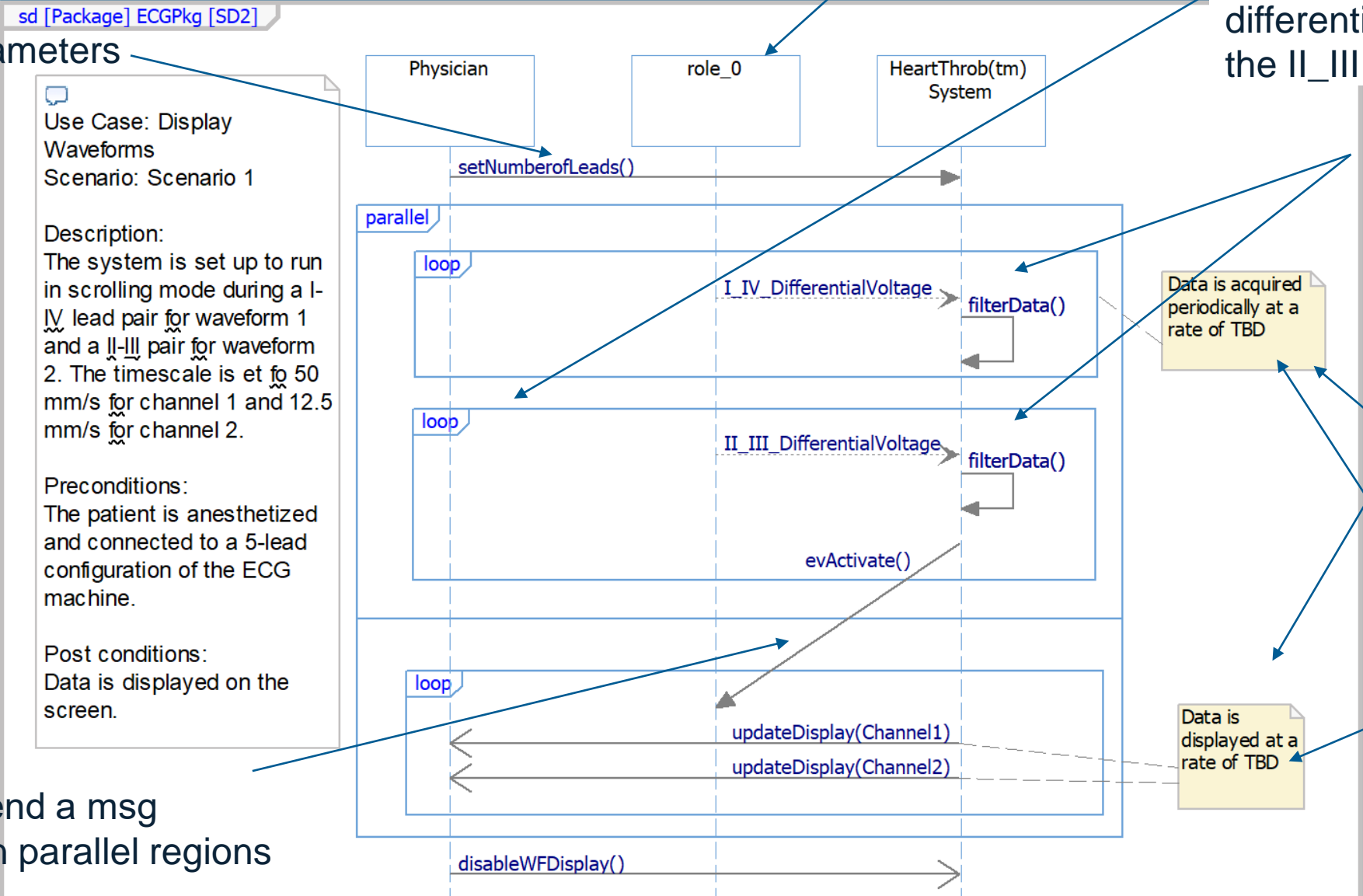
# Internal Block Diagram Answer

Naming convention?





# Sequence Diagram Answer



Missing parameters

Use Case: Display Waveforms  
Scenario: Scenario 1

Description:  
The system is set up to run in scrolling mode during a I-IV lead pair for waveform 1 and a II-III pair for waveform 2. The timescale is set to 50 mm/s for channel 1 and 12.5 mm/s for channel 2.

Preconditions:  
The patient is anesthetized and connected to a 5-lead configuration of the ECG machine.

Post conditions:  
Data is displayed on the screen.

Unnamed role

Seems unlikely that we get all the I\_IV voltage differentials and THEN get all the II\_III differentials.

When does the loop Terminate (missing guard)?

Data is acquired periodically at a rate of TBD

Timeouts would be a better way to specify timing than comments

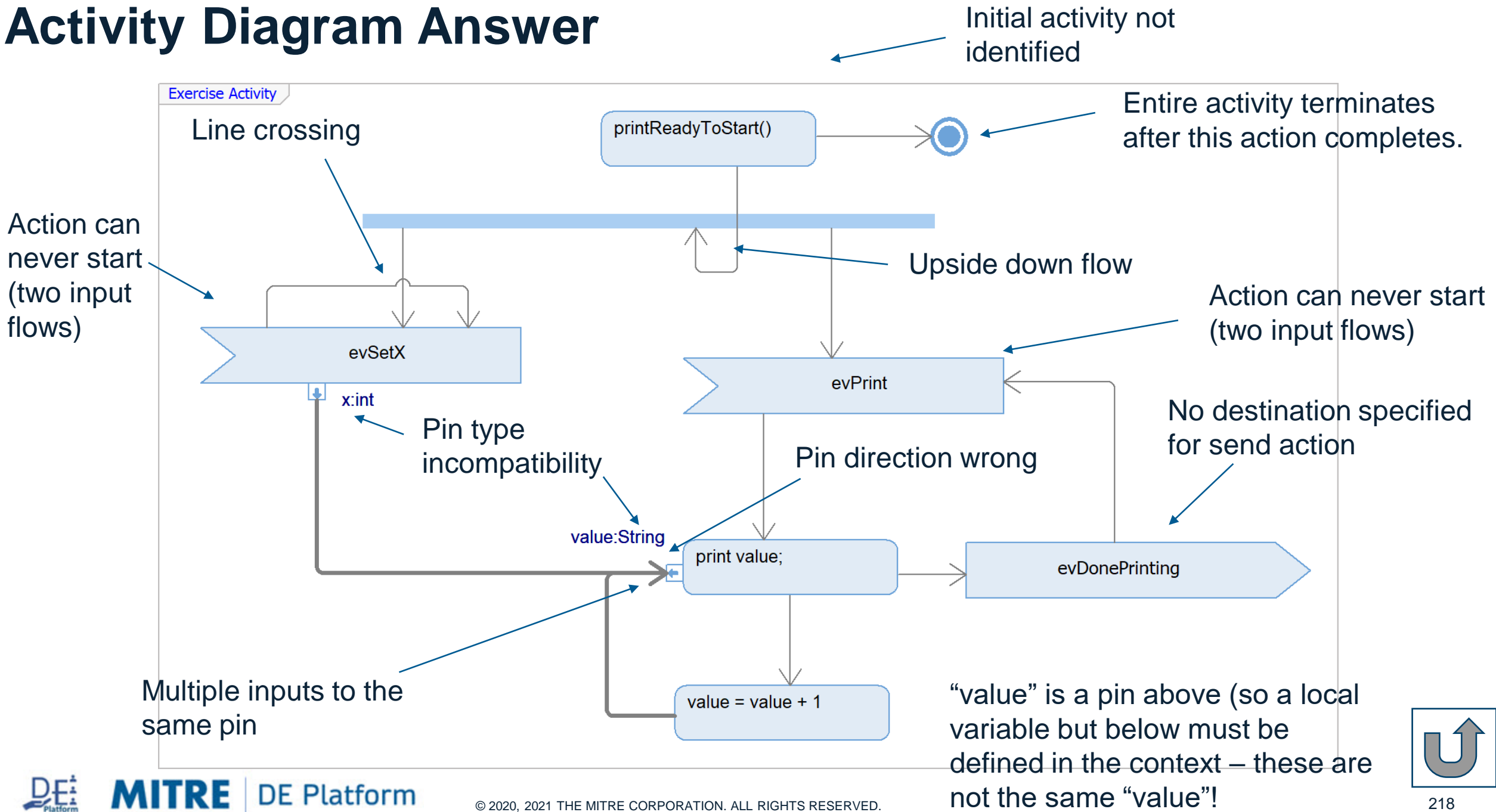
TBD Data should probably be filled in

Data is displayed at a rate of TBD

Can't send a msg between parallel regions

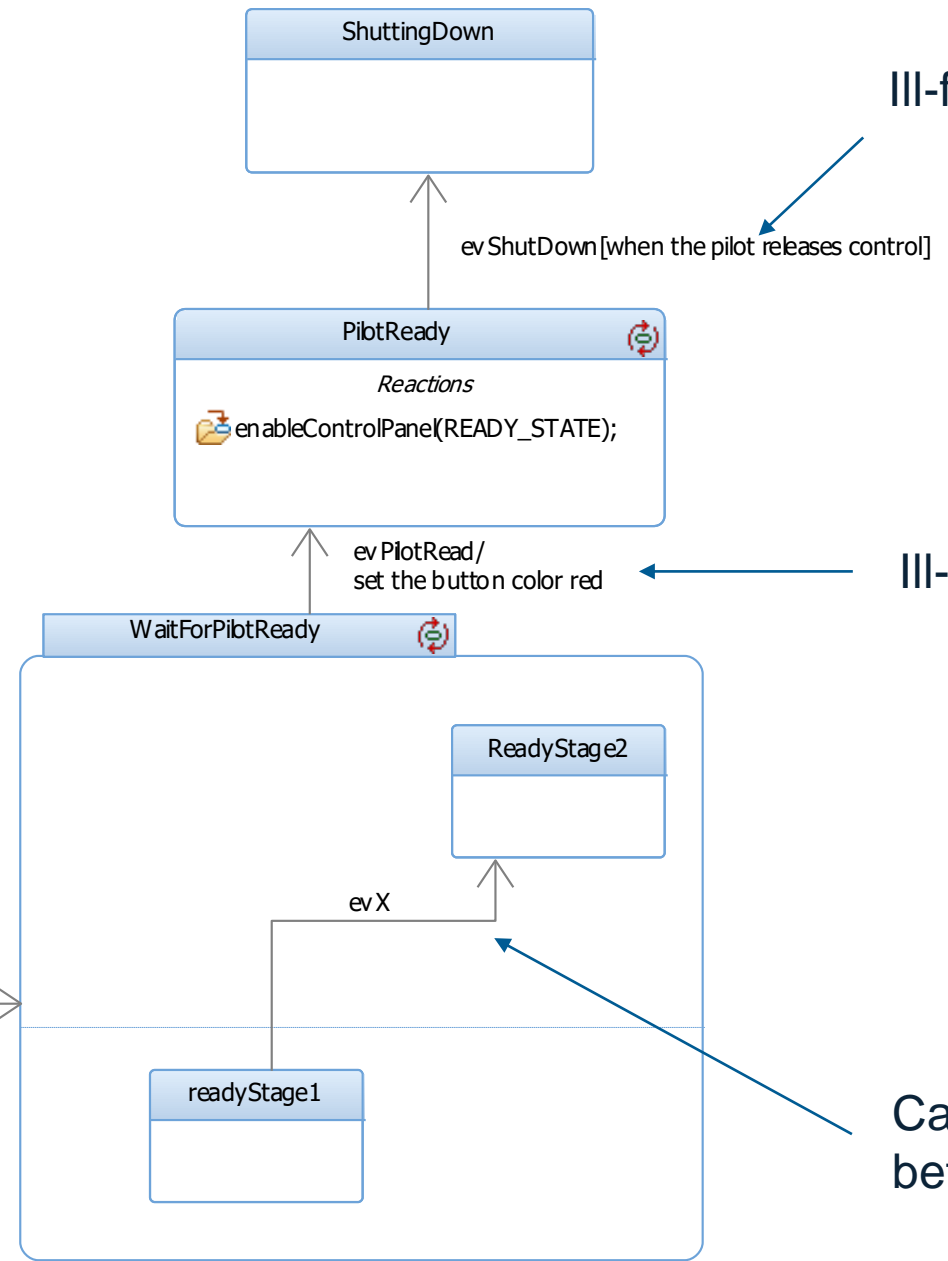
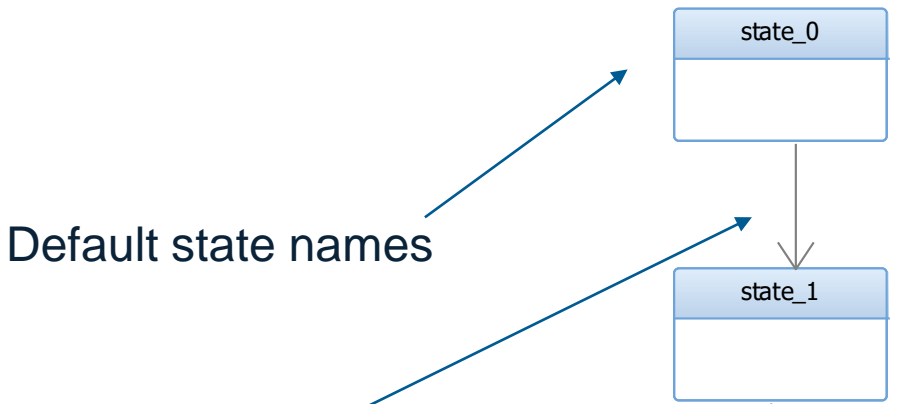


# Activity Diagram Answer



# State Diagram Answer

What's the starting state? → ?



III-formed guard statement

III-formed action statement

Cannot transition between AND-states



# Parametric Diagram Answer

(x AND y) OR z computation

- x= .334, y =.779, y = 0.118
- x AND y = .334 \*.779 = 0.260186
- 0.260186 OR z =  
0.260186 + .118 - .260186\*.118 = **0.039412**

Missing Mission Statement

Missing the input2 pin

Value y isn't connected to (missing) input pin

