# ENOLA

# Digitizing a Standard WORKSHOP

Date: March 14, 2024

Presented by: Aleczander Jackson
Parker Trombley

# AGENDA

**Lightweight MBSE Training**

- Introduction to MBSE
- Introduction to SysML
- Requirements
- Structures
- Behavior
- Parametrics

**Digitizing a Standard Workshop**

- Extracting Requirements
- Extracting Structures
- Extracting Behavior
- Extracting Metrics
- Simulation

## INTRODUCTION TO MBSE

# WHAT IS MBSE?

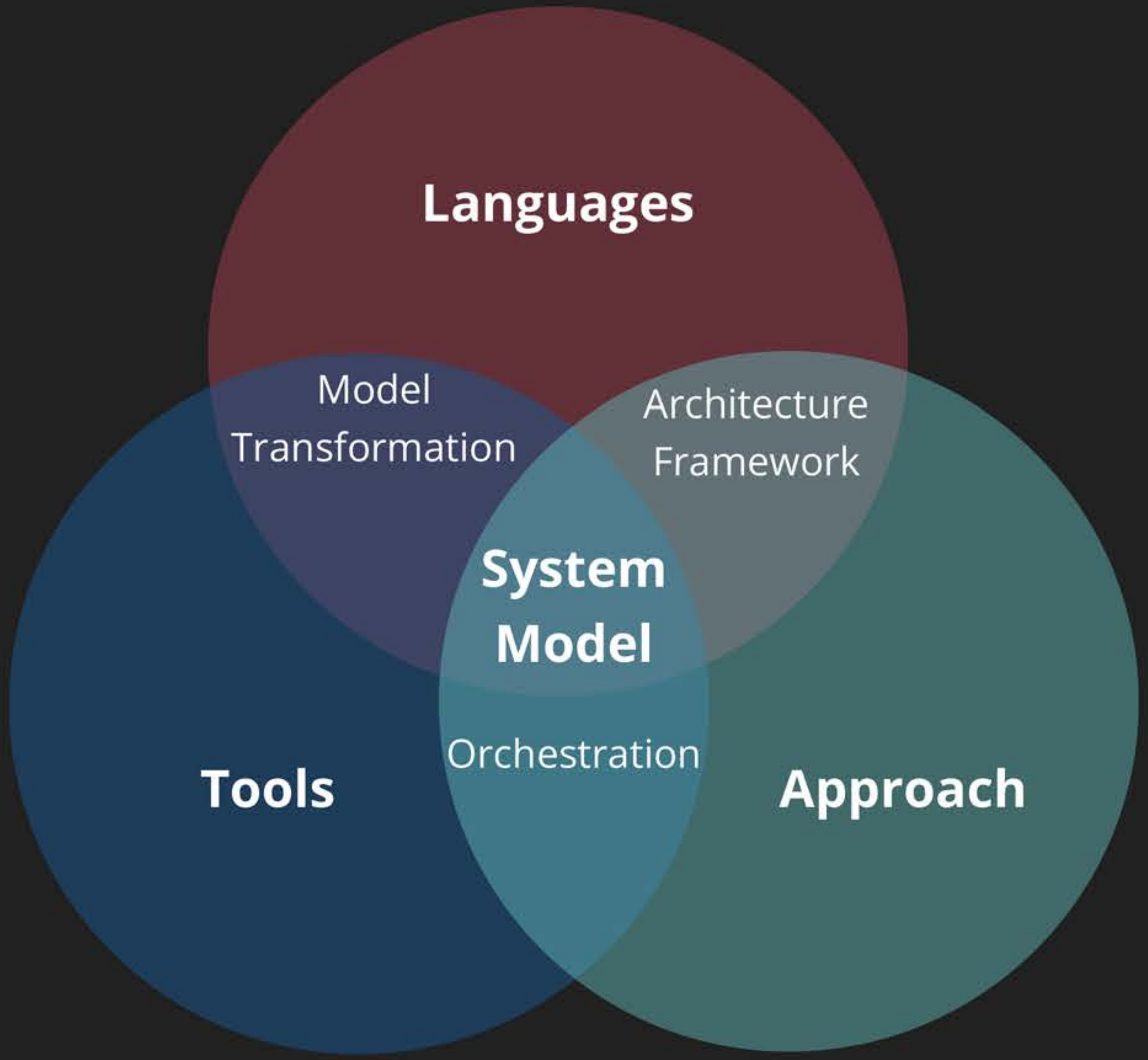
MODEL BASED SYSTEMS ENGINEERING (MBSE) IS THE USE OF A MODEL COMPOSED OF DATA AND DIAGRAMS TO PRODUCE SYSTEMS ENGINEERING ARTIFACTS.

A MODEL IS AN ABSTRACT REPRESENTATION OF A THING, AND IN THE WORLD OF MBSE IS SYSNONYMOUS WITH A DATABASE.

A DIAGRAM IS A VIEW INTO THE MODEL, A PERSPECTIVE OF ONE ASPECT OF A GIVEN SYSTEM.

DESIGN / ANALYSIS — APPLICATIONS — FOUNDATIONS

# Pillars of MBSE



## The pillars of MBSE are:

### Tool

Within MBSE, one must use a tool to house the model.

### Language

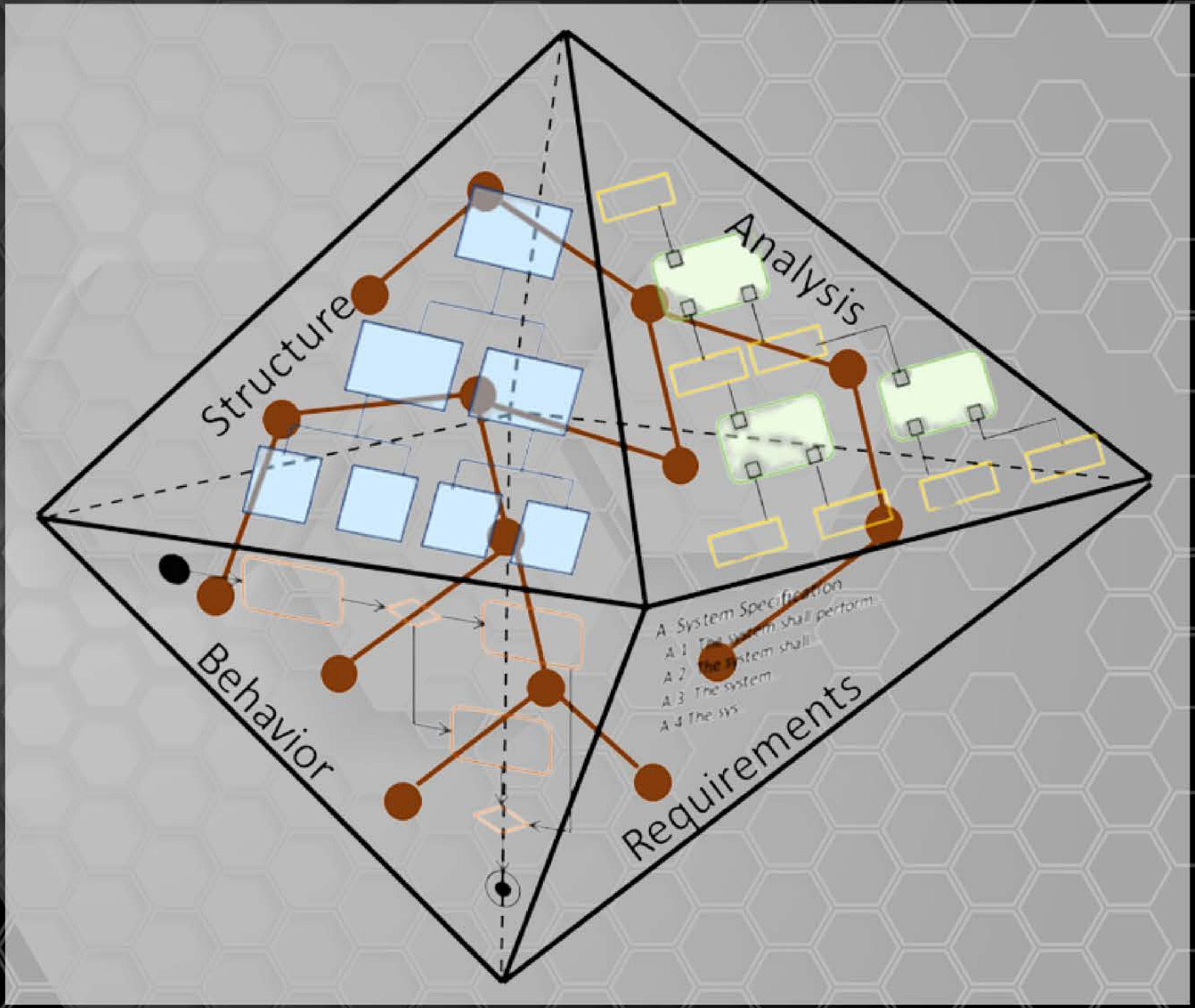
One must also select a language to use to specify the system's architecture.

### Approach

To provide consistency, one must select and standardize an approach that best meets the needs of the organization.

## Views within MBSE

- Different languages provide different diagrams.
- Different tools support different languages.
- Some tools support certain methods over others.
- The only topic generally consistent in the definition of the system architecture.
- System architecture is generally divided into four main pillars:
    - Structure
    - Behavior
    - Requirements
    - Analysis



## Key DEFINITIONS

### Language

Languages provide the elements, relationships, and diagrams necessary to fully capture system specifications.

- Examples: SysML, UML, and UPDM

### Methodology

Methodologies are processes that detail how a language should be applied.

- Examples: OOSEM, MagicGrid, and MoM

### Framework

Frameworks provide structures that dictate where artifacts should be placed within a model repository.

- Examples: MagicGrid, DoDAF, and UAF

## Key DEFINITIONS cont.

### Architecture

An architecture defines the specifications for a system. It defines the structural and behavioral aspects of the system.

### Models

Models are abstractions of a system focused on a specific aspect of that system. System architectures are composed of one or more models.

- Examples: Data Models, Structural Models, Behavioral Models, Conceptual/Logical/Physical Models

### Project

Projects are the digital repositories containing models for an architecture.

# Advantages of MBSE

## General
- Requirements Traceability
- Model Reuse
- Design Validation

## Simulation
- Design Verification
- Behavioral Simulation
- Parametric Evaluation

## Model Analytics
- Impact Analysis
- Trade-Study Analysis
- Gap Analysis
- Model Metrics

## Configuration Management
- Team Collaboration
- Version History
- Branching and Merging

## Integration with Other SE Tools
- DOORs, MATLAB, Jira, erwin, etc.

## Automation
- Large-Scale Custom Data Import
- Diagram Generation
- Model Refactoring

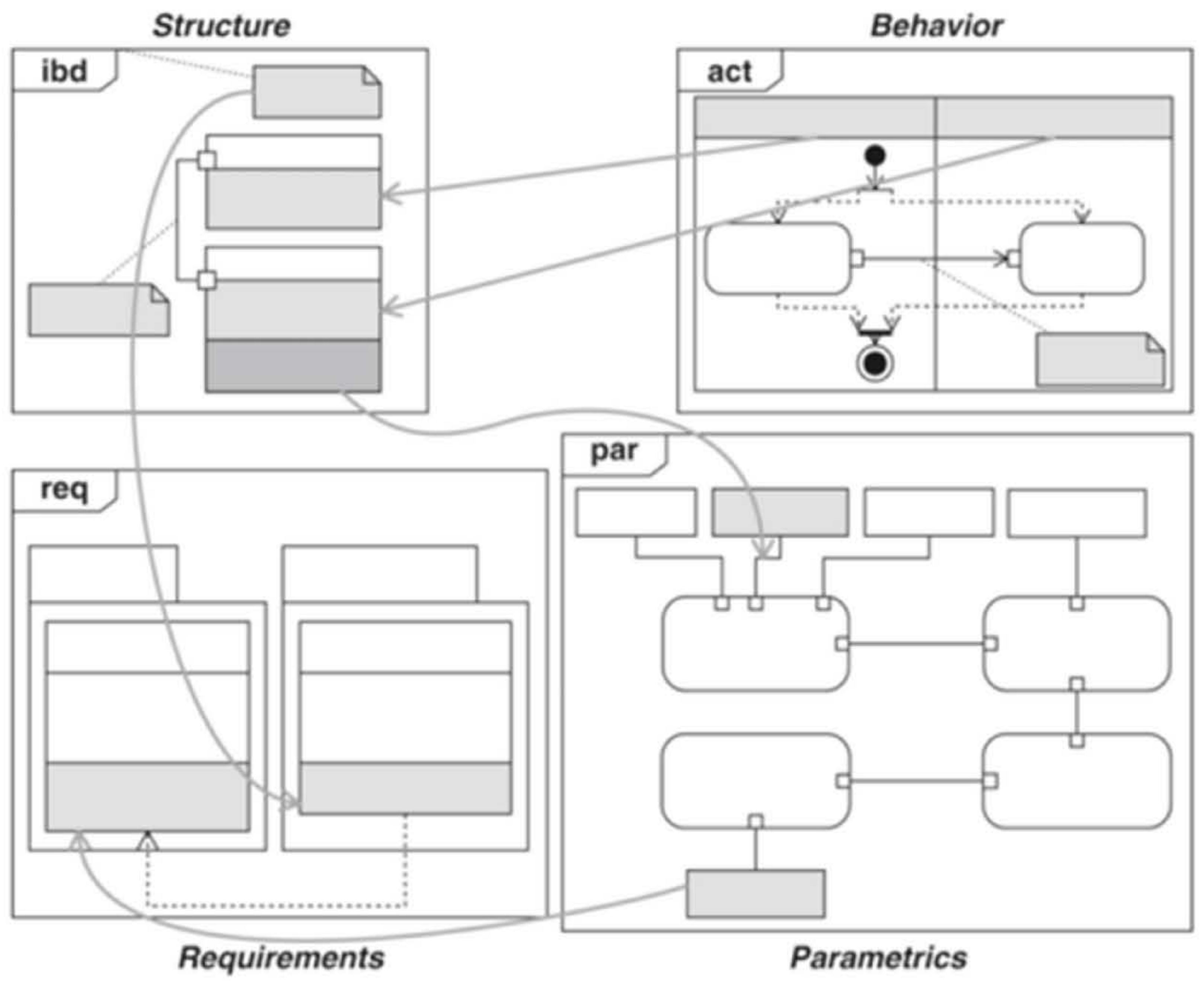
## Report Generation
- CDRL Generation
- HTML Publishing

## INTRODUCTION TO SYSML

## SYSML OVERVIEW

- The Systems Modeling Language (SysML) is a standardized language for applying MBSE.
- SysML is maintained by the Object Management Group (OMG) and was originally released in September of 2007.
- SysML is a profile built on top of the Unified Modeling Language (UML).
- Current version: 1.6

# FOUR PILLARS OF SYSML

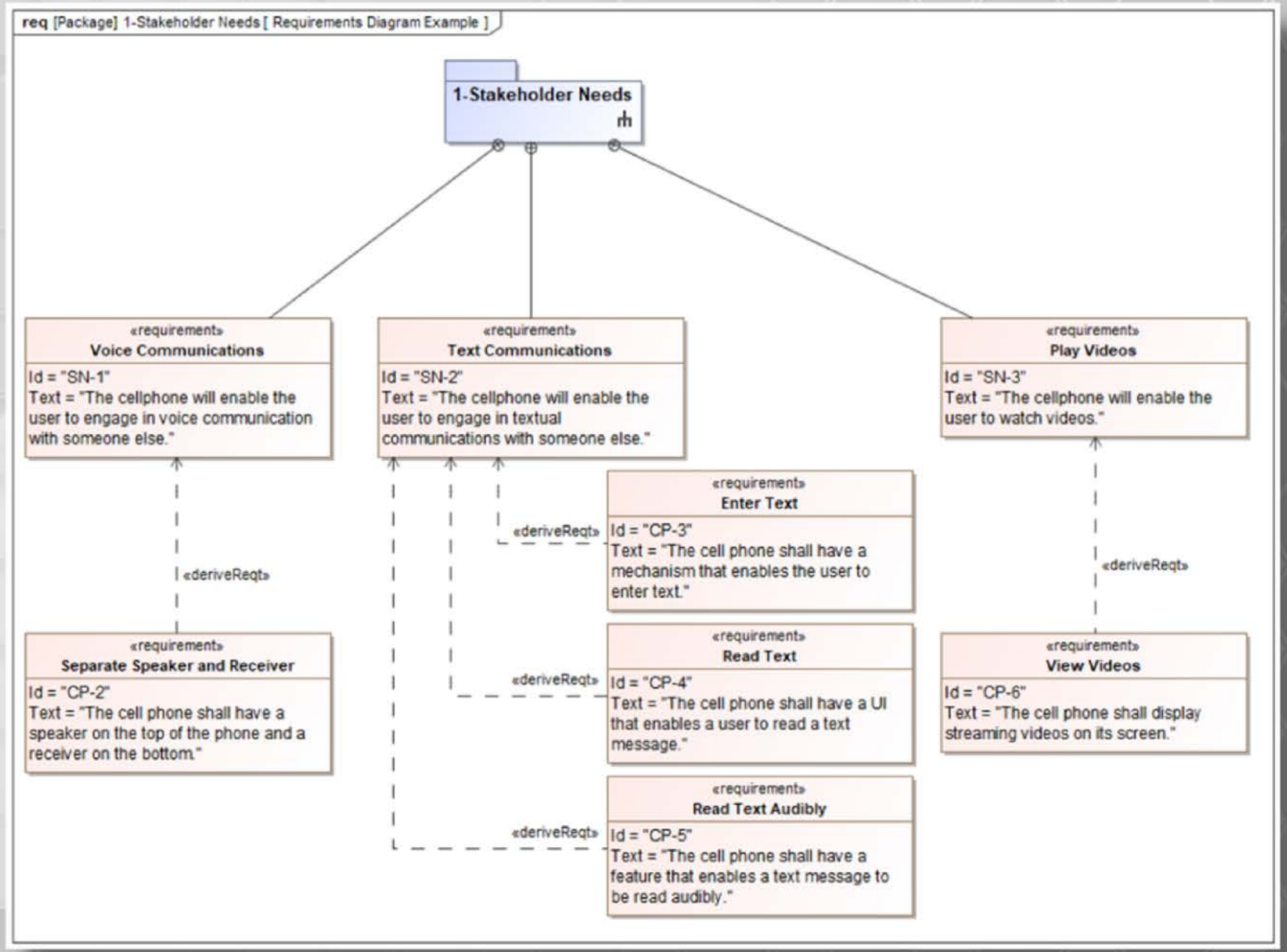


Four Pillars of SysML from 'A Practical Guide to SysML, 3rd Edition' (Figure 2.1) Copyright (c) 2015, 2012, 2009 Elsevier Inc. All rights reserved.

## REQUIREMENTS

# Diagram Example & PURPOSE

The Requirements Diagram is used to visualize the requirements of the architecture, their relationships to each other, and their relationships to other architectural elements.



«requirement»
**Voice Communications**
Id = "SN-1"
Text = "The cellphone will enable the user to engage in voice communication with someone else."

«requirement»
**Text Communications**
Id = "SN-2"
Text = "The cellphone will enable the user to engage in textual communications with someone else."

«requirement»
**Play Videos**
Id = "SN-3"
Text = "The cellphone will enable the user to watch videos."

«requirement»
**Separate Speaker and Receiver**
Id = "CP-2"
Text = "The cell phone shall have a speaker on the top of the phone and a receiver on the bottom."

«requirement»
**Enter Text**
Id = "CP-3"
Text = "The cell phone shall have a mechanism that enables the user to enter text."

«requirement»
**Read Text**
Id = "CP-4"
Text = "The cell phone shall have a UI that enables a user to read a text message."

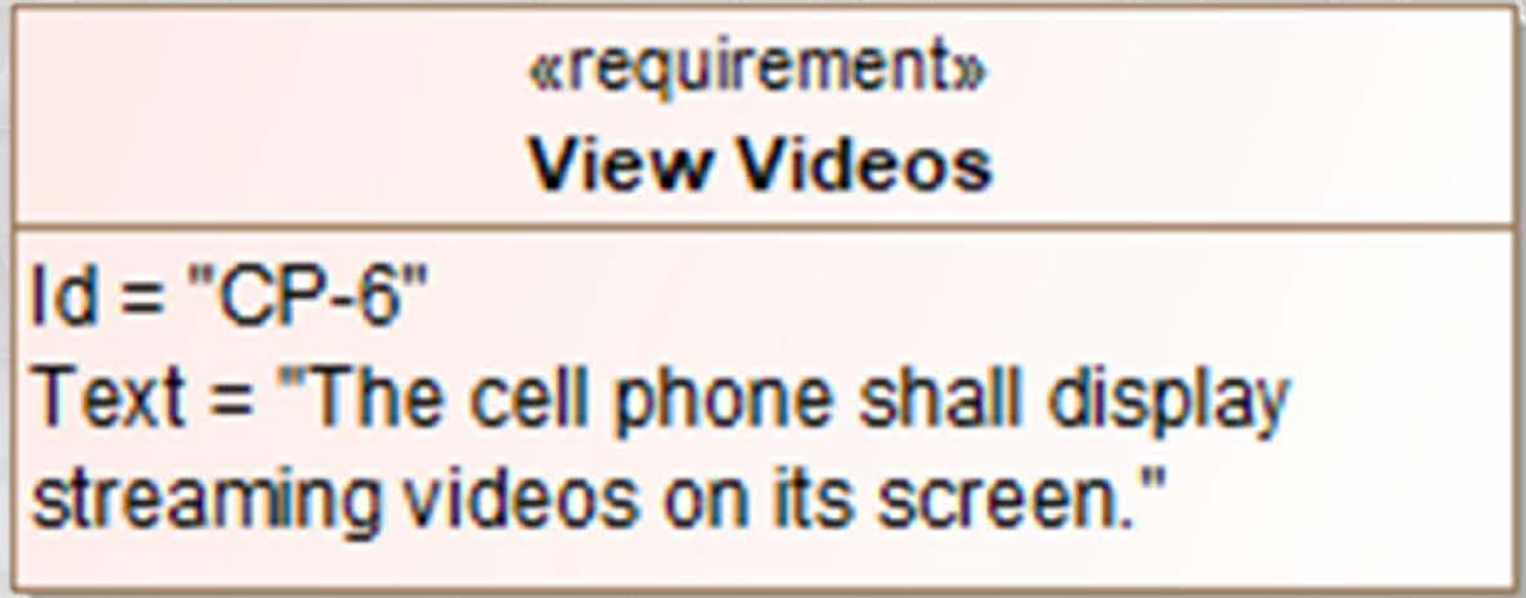
«requirement»
**Read Text Audibly**
Id = "CP-5"
Text = "The cell phone shall have a feature that enables a text message to be read audibly."

«requirement»
**View Videos**
Id = "CP-6"
Text = "The cell phone shall display streaming videos on its screen."

## REQUIREMENTS

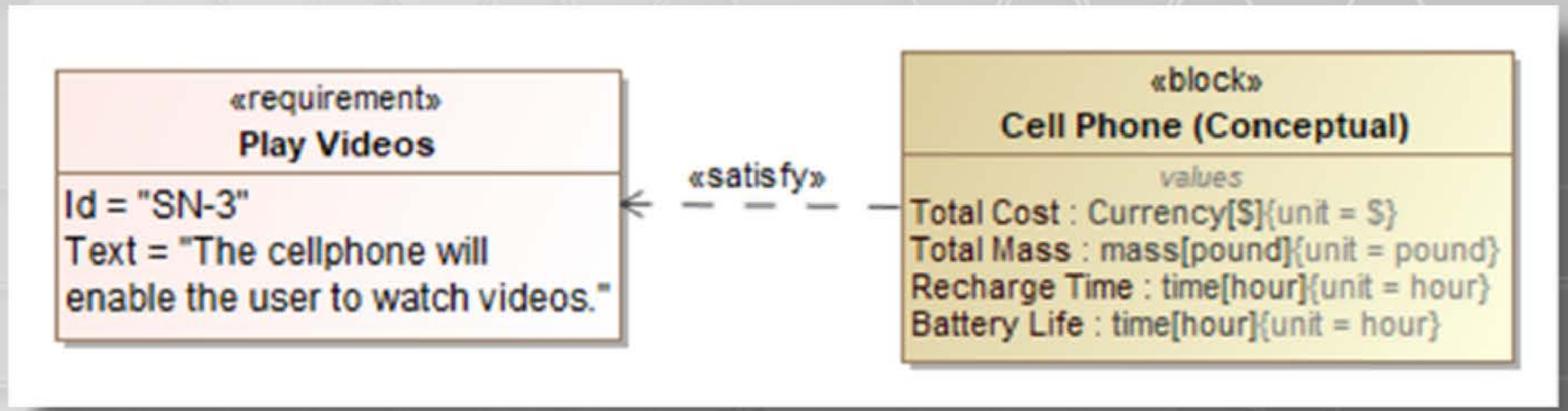
- Requirement elements represent the statements that must be met in order for a system to deliver the required functionality within certain performance metrics.

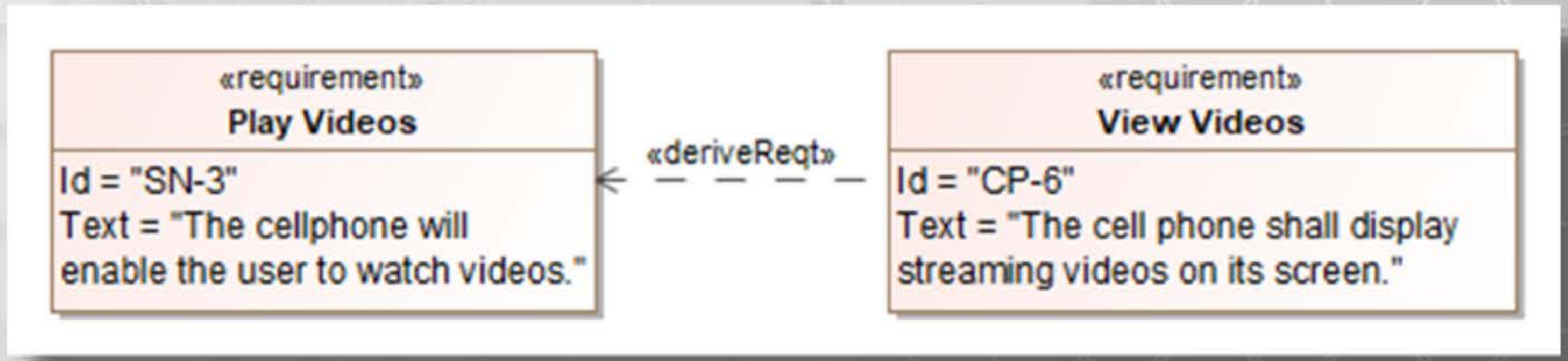
- Requirement elements in SysML have three main properties:
  - Name: a searchable/filterable title that is displayed in tables, matrices, and the Containment Tree
  - Id: the unique identifier for the requirement
  - Text: the textual specification or “shall statement” of the requirement


«requirement»
View Videos
Id = "CP-6"
Text = "The cell phone shall display streaming videos on its screen."

## Requirement RELATIONSHIPS

- Satisfy
  - Asserts that a model element meets the needs of a requirement.
- Derive
  - Specifies that one requirement is an extension or derivation of another.


«requirement»
Play Videos
Id = "SN-3"
Text = "The cellphone will enable the user to watch videos."
«satisfy»
«block»
Cell Phone (Conceptual)
values
Total Cost : Currency[$]{unit = $}
Total Mass : mass[pound]{unit = pound}
Recharge Time : time[hour]{unit = hour}
Battery Life : time[hour]{unit = hour}

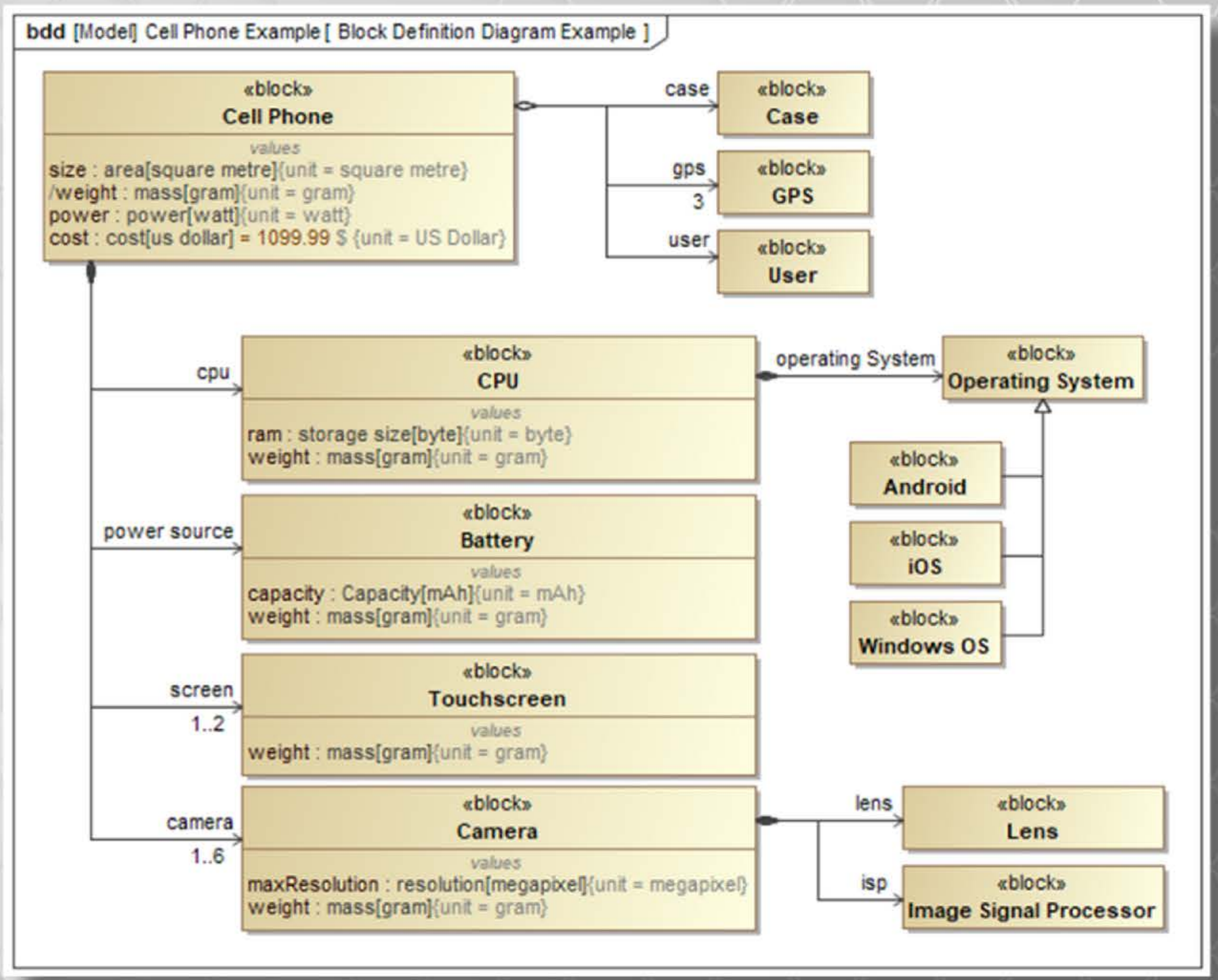

«requirement»
Play Videos
Id = "SN-3"
Text = "The cellphone will enable the user to watch videos."
«deriveReqt»
«requirement»
View Videos
Id = "CP-6"
Text = "The cell phone shall display streaming videos on its screen."

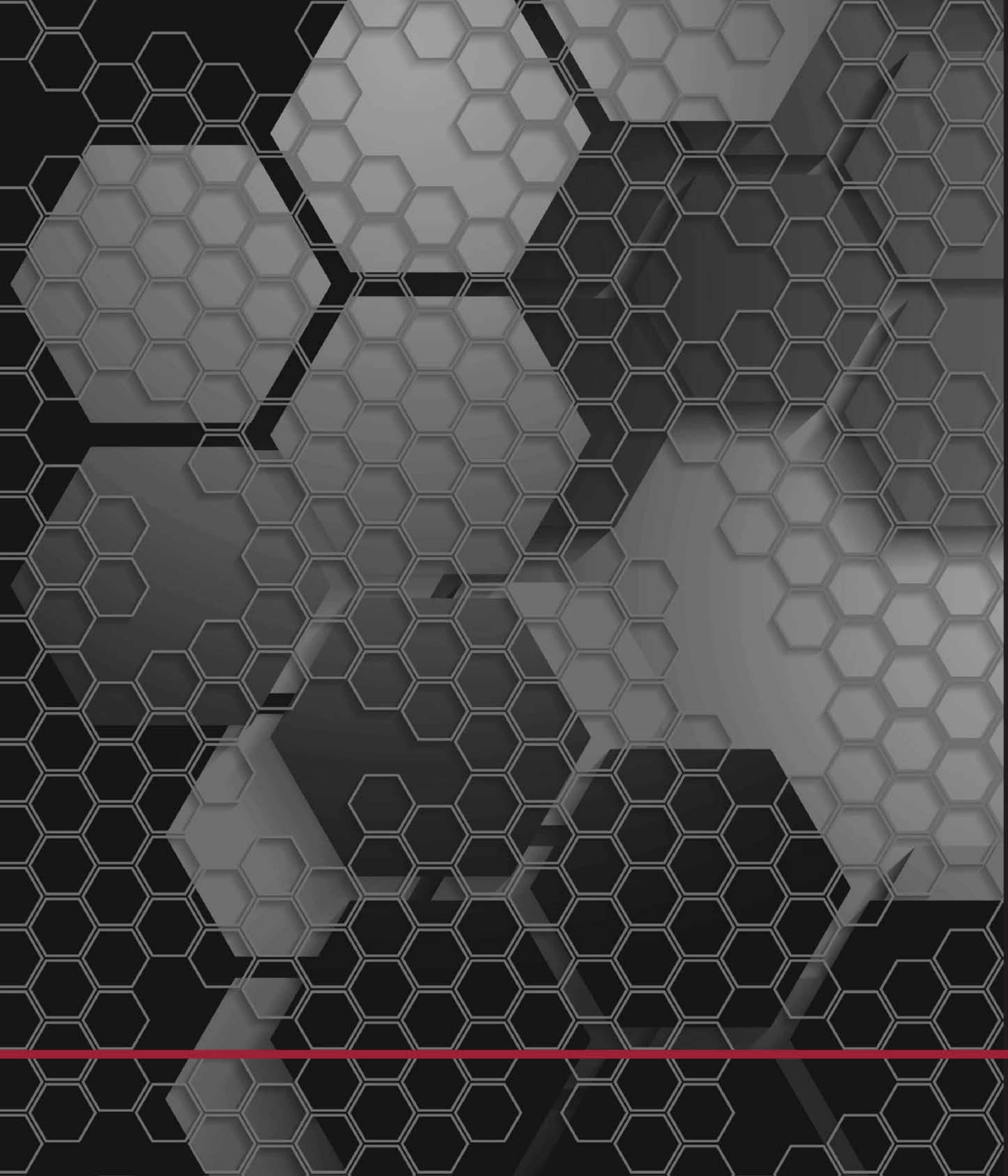
## REQUIREMENTS
## DEMO

## STRUCTURES

## Diagram Example & PURPOSE

- Block Definition Diagrams (BDDs) shows the breakdown of the system structure into its components.
- BDDs are also used to build taxonomies of elements and their subtypes
- Despite the name, BDDs are used to show taxonomies and decompositions of other SysML elements in addition to Blocks.


bdd [Model] Cell Phone Example [ Block Definition Diagram Example ]
«block» Cell Phone
values
size : area[square metre]{unit = square metre}
/weight : mass[gram]{unit = gram}
power : power[watt]{unit = watt}
cost : cost[us dollar] = 1099.99 $ {unit = US Dollar}
case
«block» Case
gps
3
«block» GPS
user
«block» User
cpu
«block» CPU
values
ram : storage size[byte]{unit = byte}
weight : mass[gram]{unit = gram}
operating System
«block» Operating System
«block» Android
«block» iOS
«block» Windows OS
power source
«block» Battery
values
capacity : Capacity[mAh]{unit = mAh}
weight : mass[gram]{unit = gram}
screen
1..2
«block» Touchscreen
values
weight : mass[gram]{unit = gram}
camera
1..6
«block» Camera
values
maxResolution : resolution[megapixel]{unit = megapixel}
weight : mass[gram]{unit = gram}
lens
«block» Lens
isp
«block» Image Signal Processor

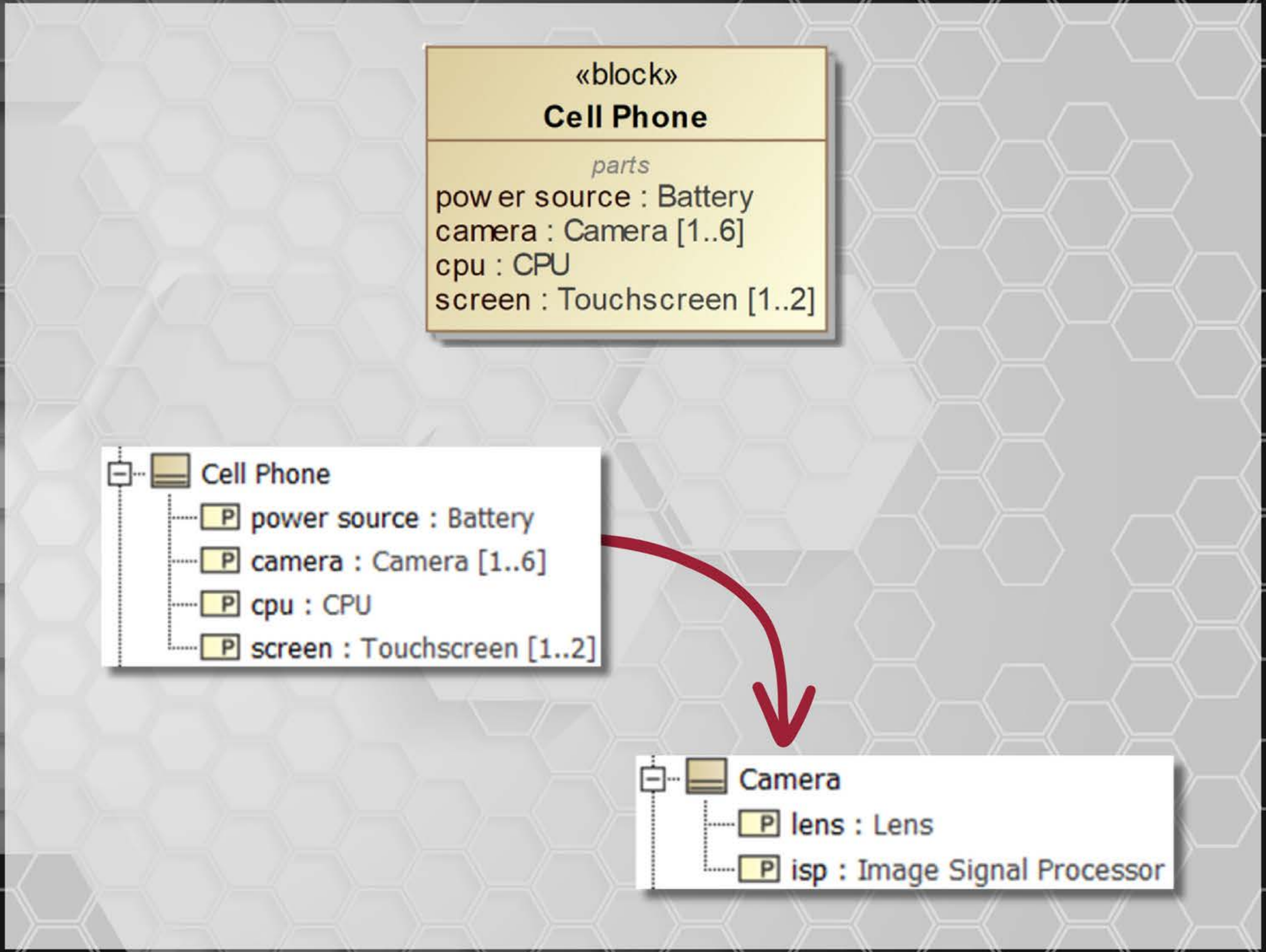


## DEFINITION & EXAMPLES

- Blocks represent the structural “things” of the architecture.
- Blocks are used to depict many different things, real or abstract:
  - Systems, subsystems, components, aggregates of systems, persons, organizations, facilities, etc.
- All aspects of real-world things can be captured in the specification of Blocks.
- Blocks are templates for objects that can be reused in other contexts.

## Part PROPERTIES

- Part properties represent the <u>internal components</u> of a block.
- When built, the part properties would be included in and delivered with the system.
  - E.g.: an engine in a car, a motherboard in a computer, a drum in a washing machine, etc.
- Part properties are always typed by blocks.
- Only the first layer of components will be listed within the system context.
  - Lower layers of components for each part property are contained within the element typing it.


«block»
**Cell Phone**
*parts*
power source : Battery
camera : Camera [1..6]
cpu : CPU
screen : Touchscreen [1..2]

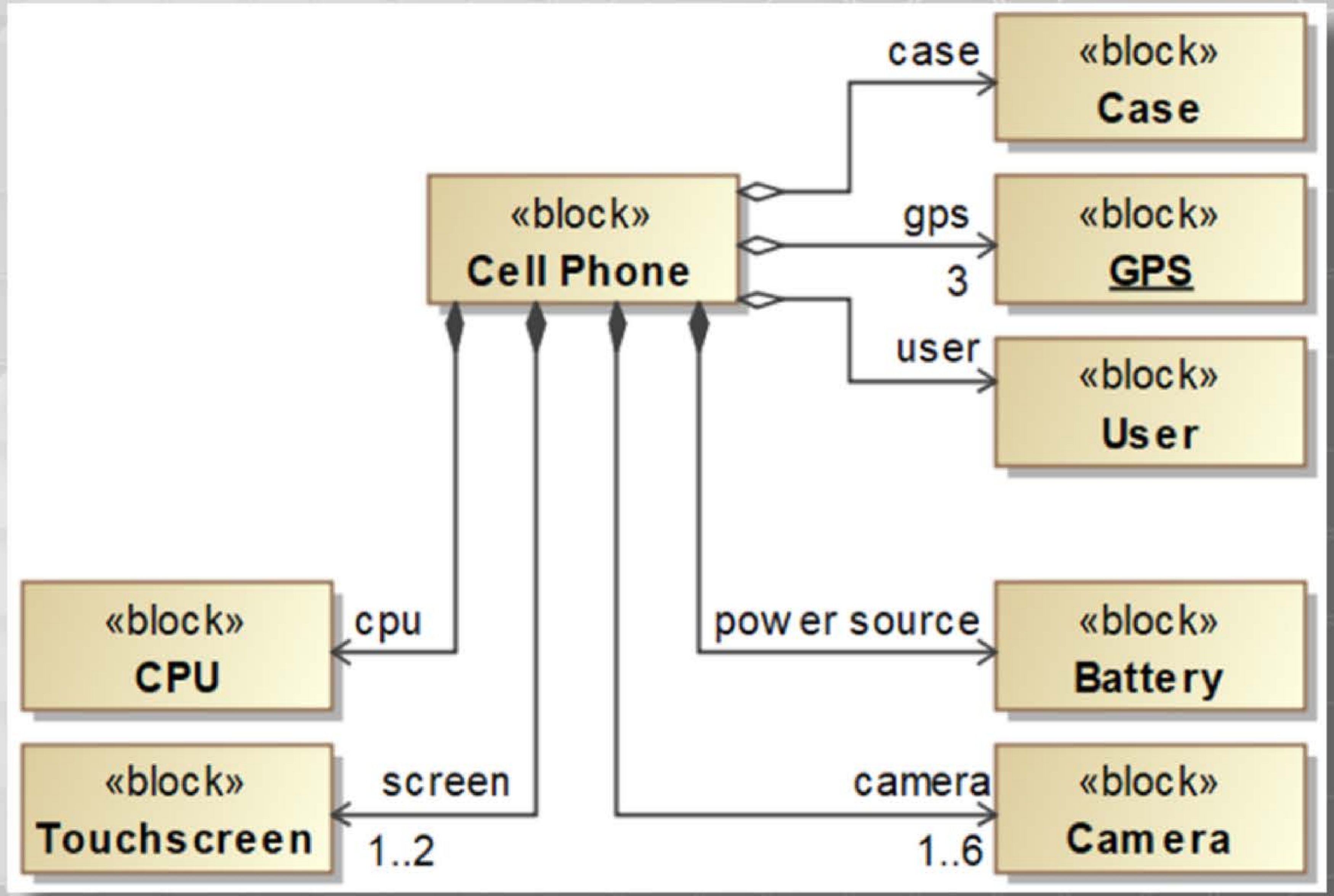
Cell Phone
- power source : Battery
- camera : Camera [1..6]
- cpu : CPU
- screen : Touchscreen [1..2]

Camera
- lens : Lens
- isp : Image Signal Processor

## Associations in BDDs

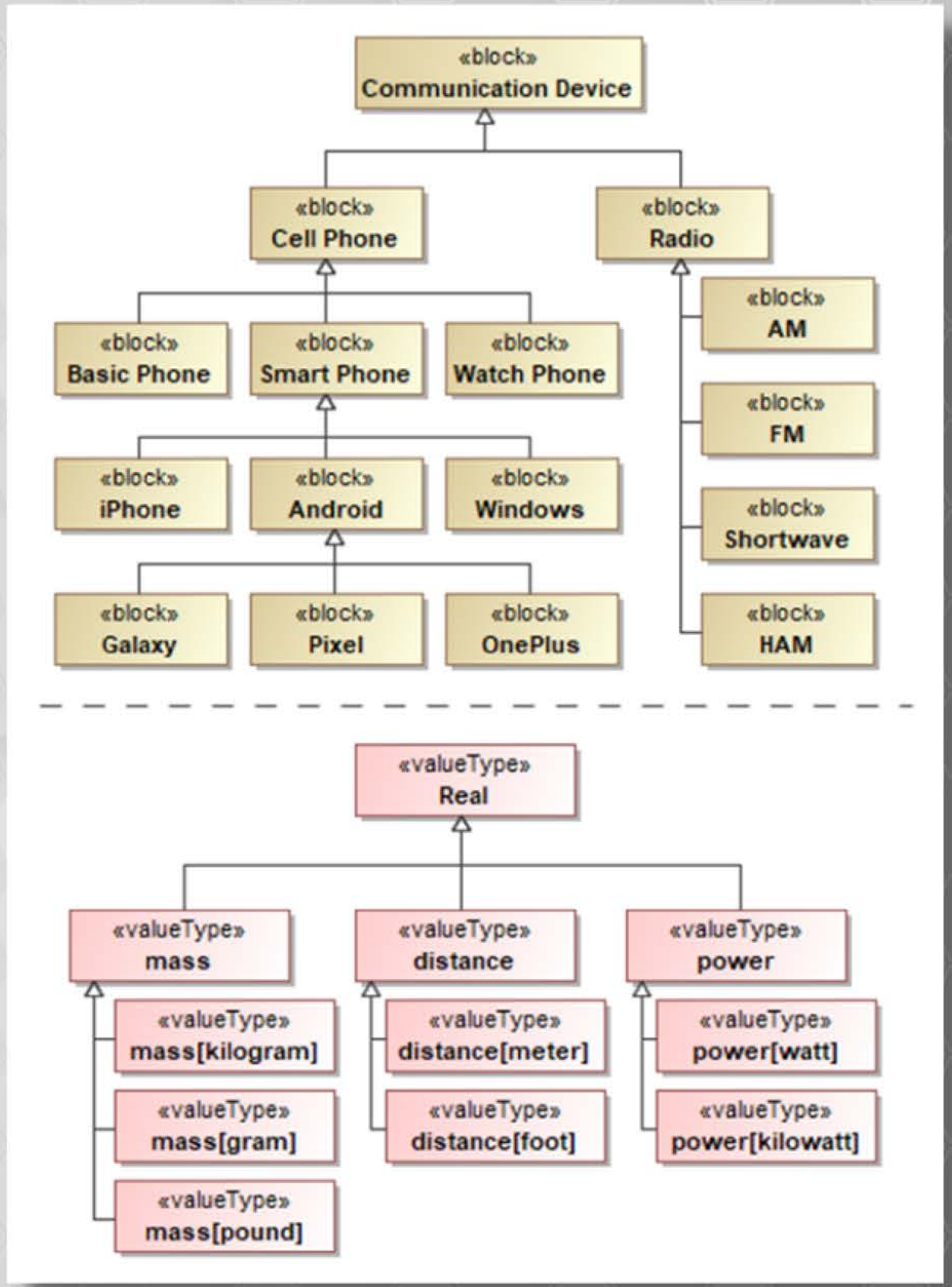
- Associations are most often going to be created as Directed Compositions and Directed Aggregations in BDDs.
  - This is because the diagram is typically used to decompose system structures.

- Directed Compositions will create Part Properties underneath the Composite Block.



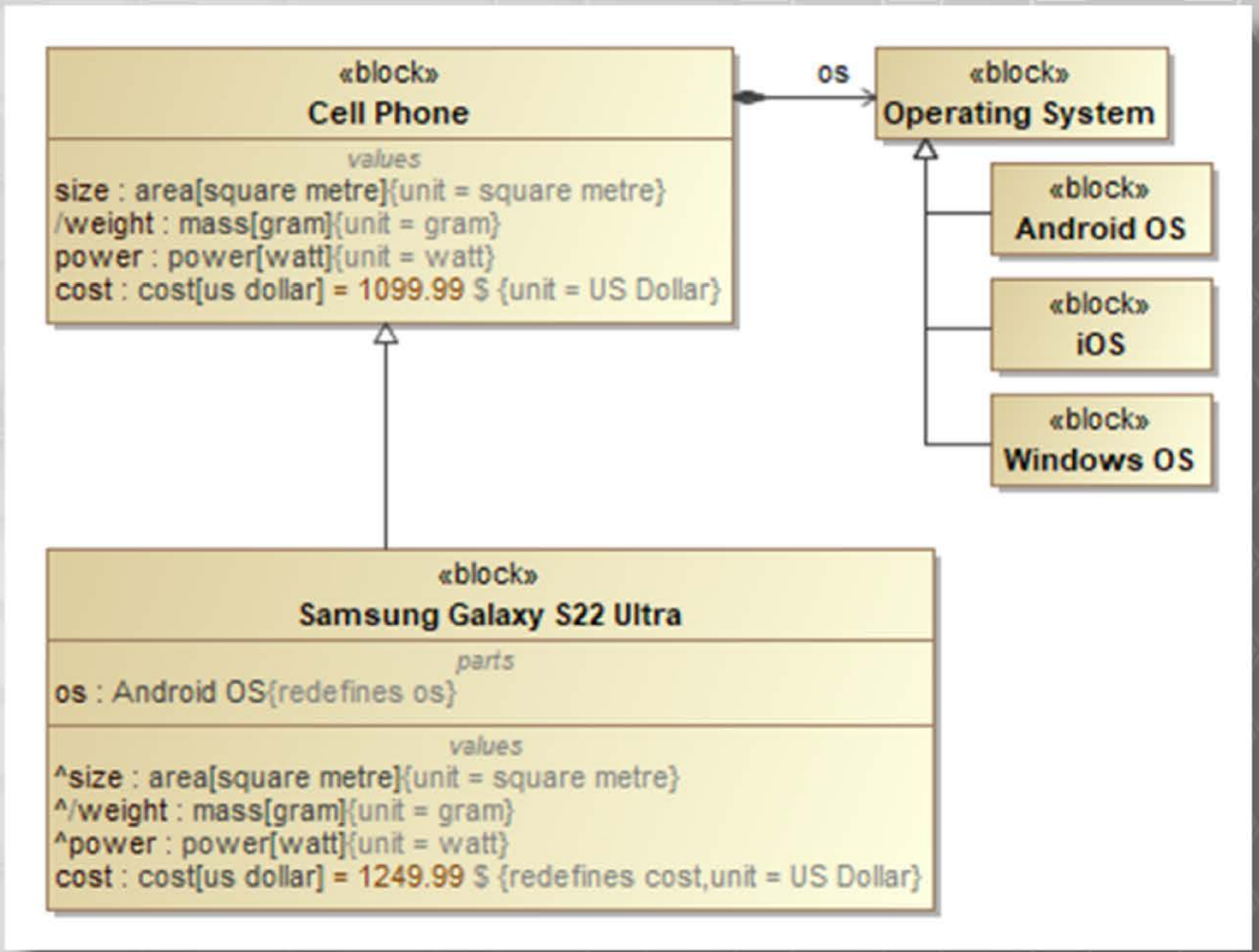
# Building TAXONOMIES

- Generalization relationships are used to build taxonomies of elements.
- Generalizations point from the specialized object, the <u>subtype</u>, to the generalized object, the <u>supertype</u>.
- Generalizations can only be created between two elements of the same SysML type.



# Inheritance & REDEFINING

- Specialized elements inherit the properties, operations, and relationships of their generalized parents.
- If shown on a diagram, inherited properties and operations will have a “^” symbol in front of them.
  - These are hidden by default in MagicDraw.
- Inherited properties are just pointers to the original versions within the generalized element.
  - If edited in the specialized element, both will be updated.
- Once inherited, specialized elements can redefine properties and operations to specify further detail / implementation.

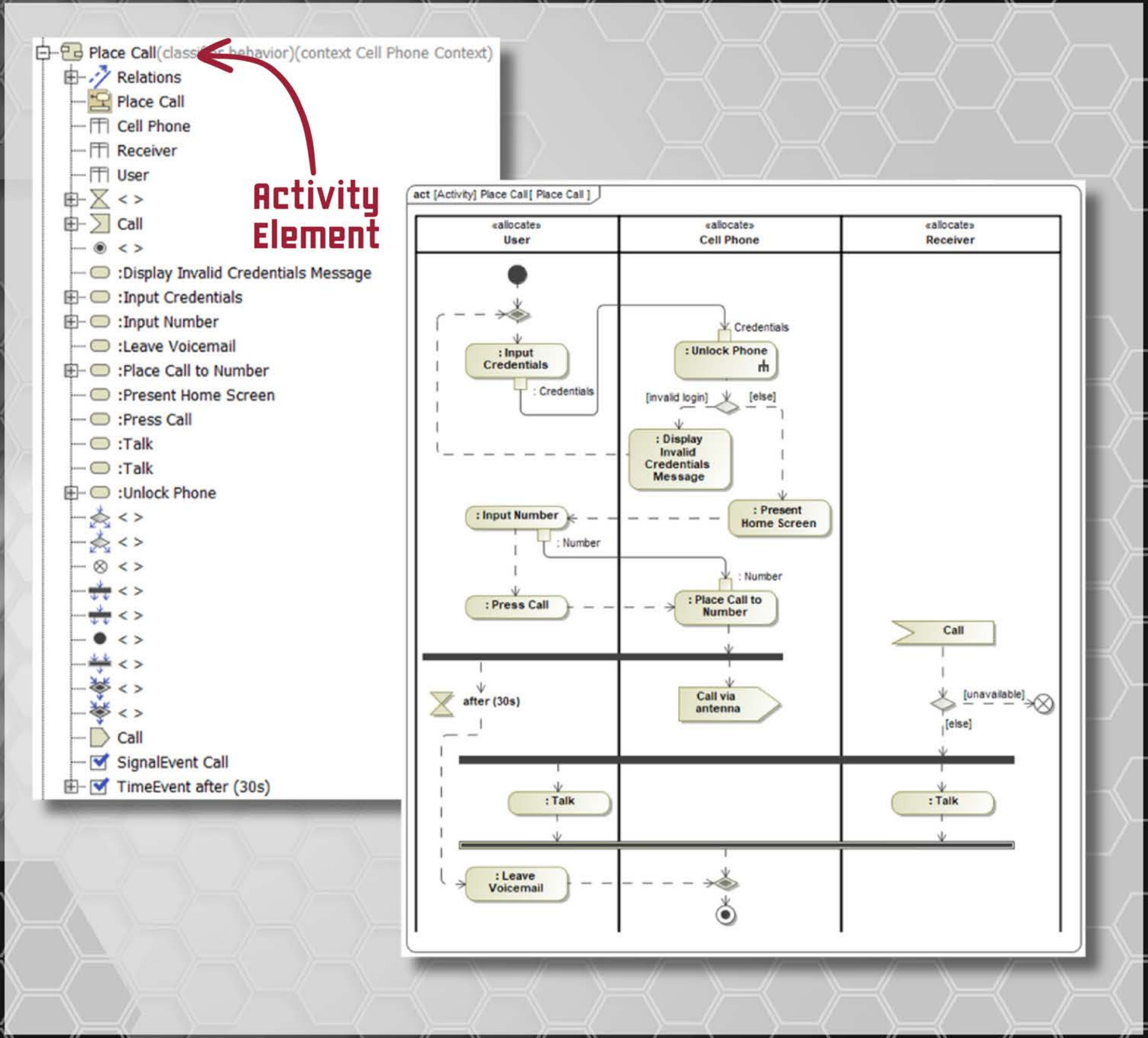

«block»
Cell Phone
values
size : area[square metre]{unit = square metre}
/weight : mass[gram]{unit = gram}
power : power[watt]{unit = watt}
cost : cost[us dollar] = 1099.99 $ {unit = US Dollar}
os
«block»
Operating System
«block»
Android OS
«block»
iOS
«block»
Windows OS
«block»
Samsung Galaxy S22 Ultra
parts
os : Android OS{redefines os}
values
^size : area[square metre]{unit = square metre}
^/weight : mass[gram]{unit = gram}
^power : power[watt]{unit = watt}
cost : cost[us dollar] = 1249.99 $ {redefines cost,unit = US Dollar}

## DEMO

## BEHAVIOR

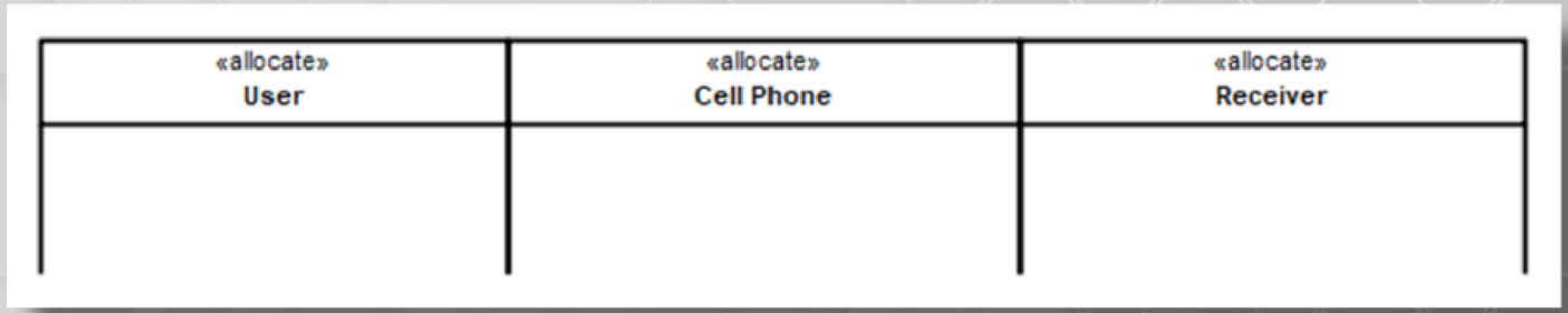
## Diagram Example & PURPOSE

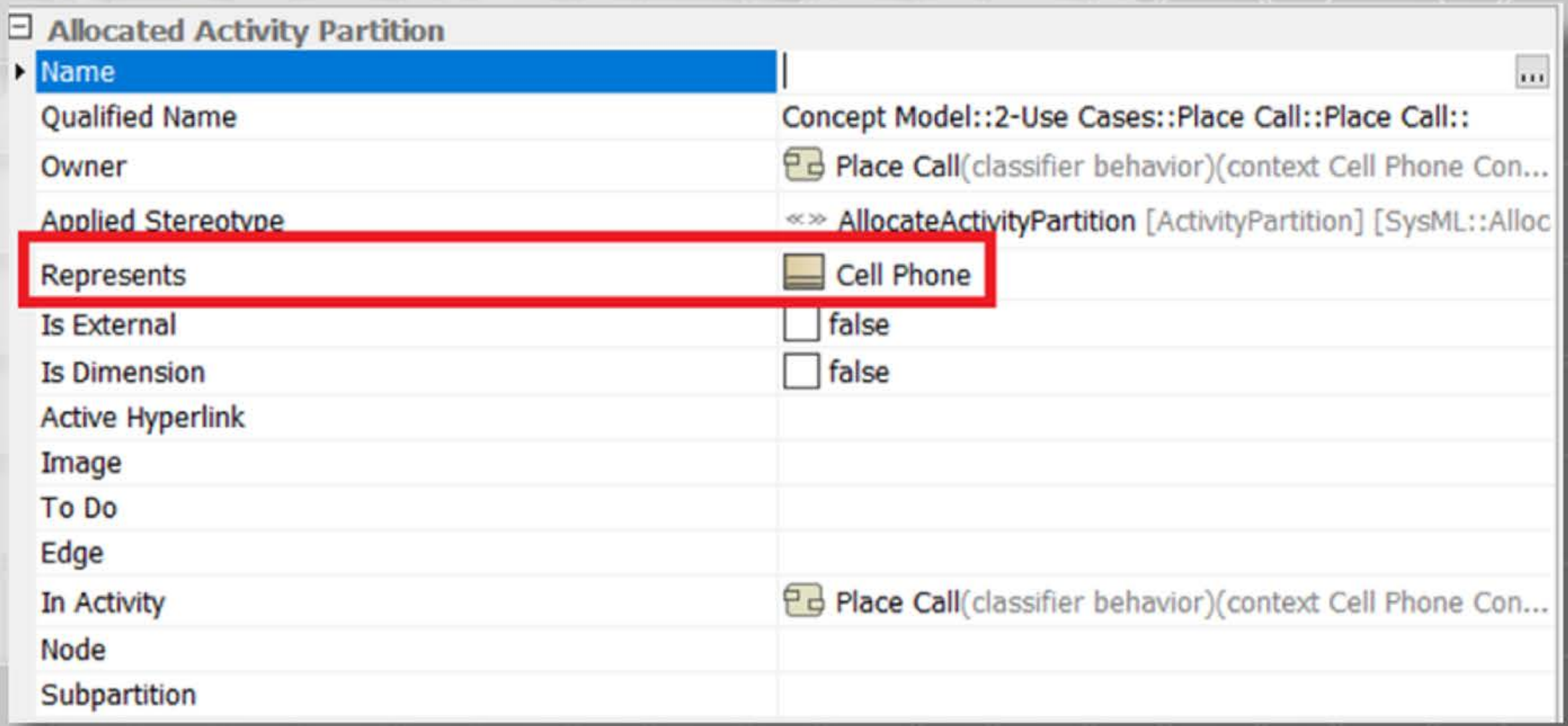
- System functionality flow is captured in Activity Diagrams.
- Activity Diagrams are created underneath Activity elements which they describe the behavior for.
- Activities are reusable elements that can define full system functions (placing calls), coordinate larger sets of functions (operating a phone), or represent complex steps in higher activities (ordering takeout).



# Activity PARTITIONS

- Activity Partitions, more commonly knowns as “swimlanes”, specify who or what is performing each of the actions (steps) in the Activity.
- Swimlanes are not named; they should be set to represent a Block, Actor, Part Property, or Reference Property.
- Actions placed into the swimlanes will be allocated to the element that swimlane represents.



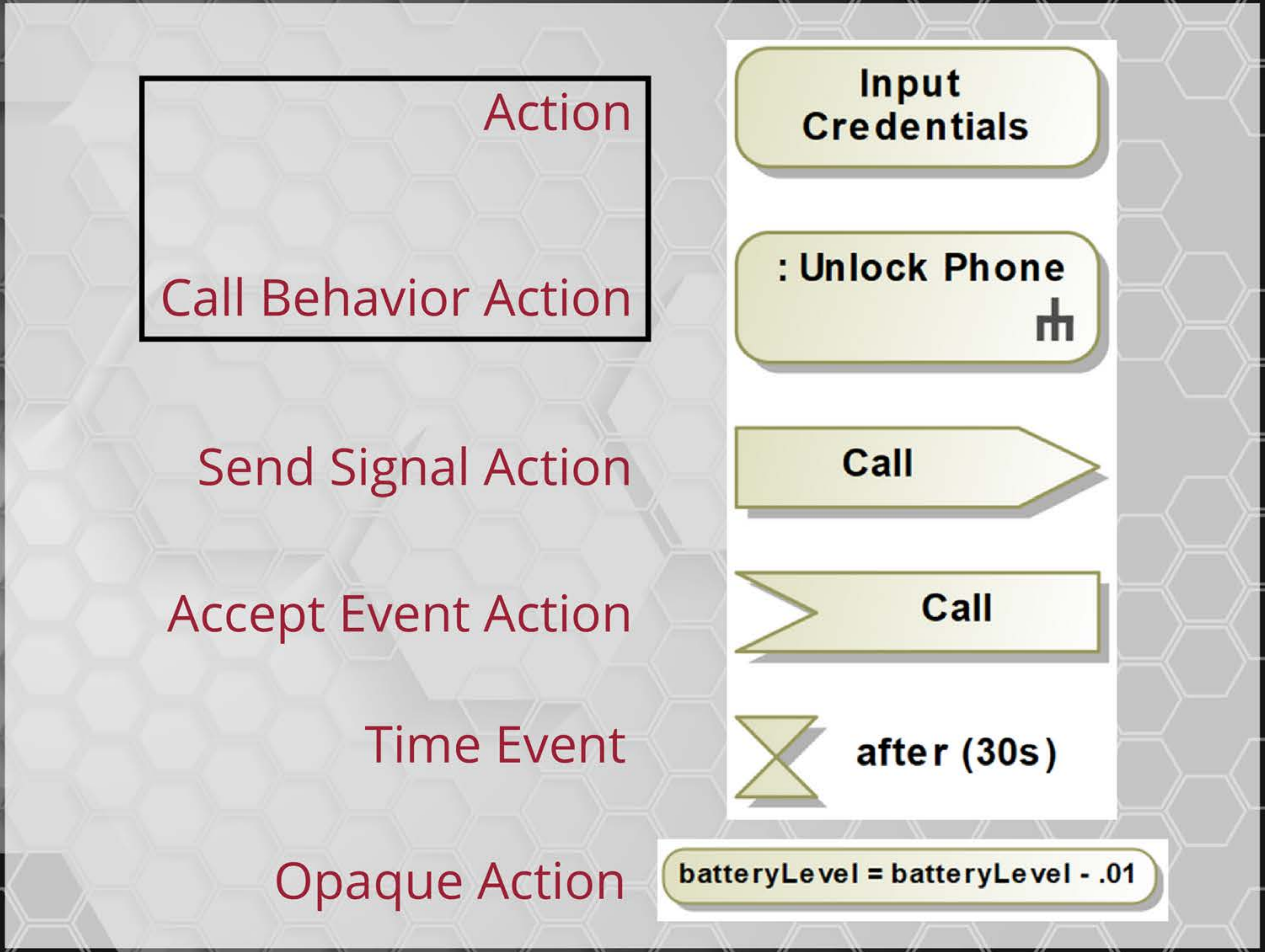


**Allocated Activity Partition**

| | |
|---|---|
| Name | |
| Qualified Name | Concept Model::2-Use Cases::Place Call::Place Call:: |
| Owner | Place Call(classifier behavior)(context Cell Phone Con... |
| Applied Stereotype | «» AllocateActivityPartition [ActivityPartition] [SysML::Alloc |
| Represents | Cell Phone |
| Is External | ☐ false |
| Is Dimension | ☐ false |
| Active Hyperlink | |
| Image | |
| To Do | |
| Edge | |
| In Activity | Place Call(classifier behavior)(context Cell Phone Con... |
| Node | |
| Subpartition | |

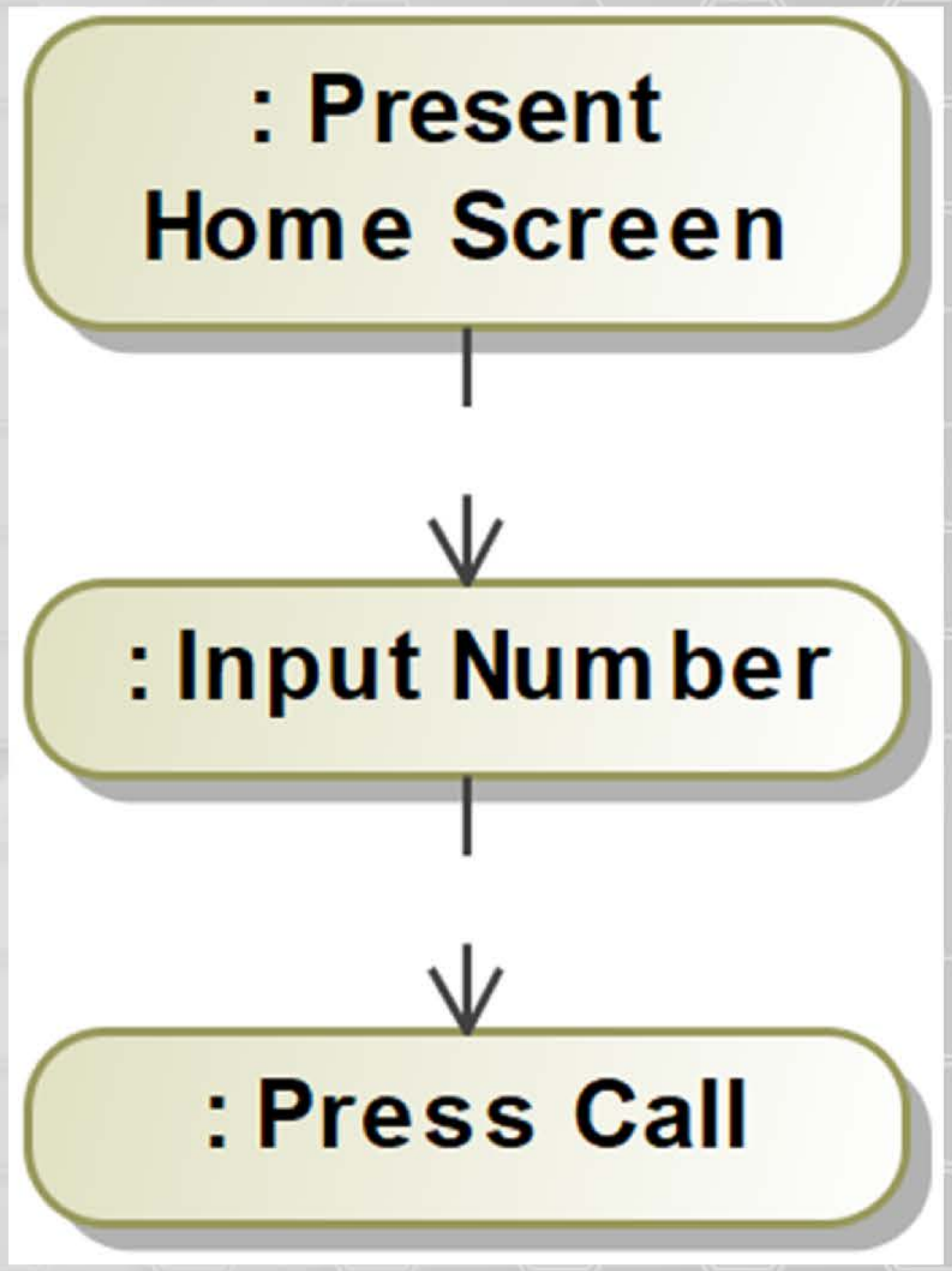
# Description of ACTIONS

- Actions represent the individual steps in an Activity.
- There are many different types of actions that can be created within an Activity Diagram.
- All actions have a name property that should describe the purpose of that step and it should be written in a verb-phrase.
  - E.g.: Answer Phone, Start Car, Pour Coffee, etc.
- Actions often represent full Activities when decomposed and can even express code which will run during simulation.



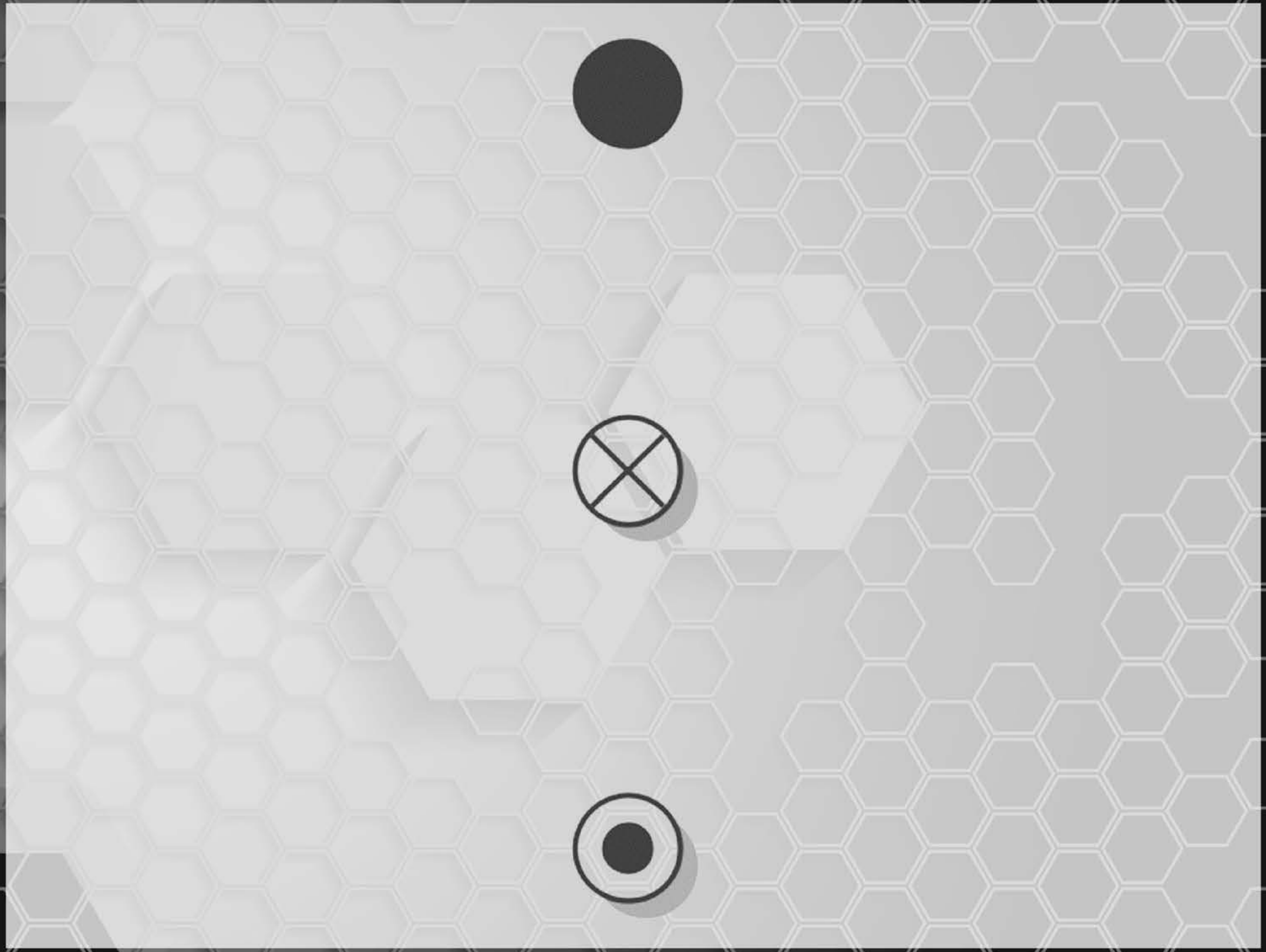
## Control FLOW

- When an action completes, the activity will flow through all of its outgoing flows into other actions and nodes.
- Control flow represents the flow from one action to the next.
    - It allows the activity to define the sequence of the actions within it.
- Actions should only ever have one incoming control flow and one outgoing control flow.



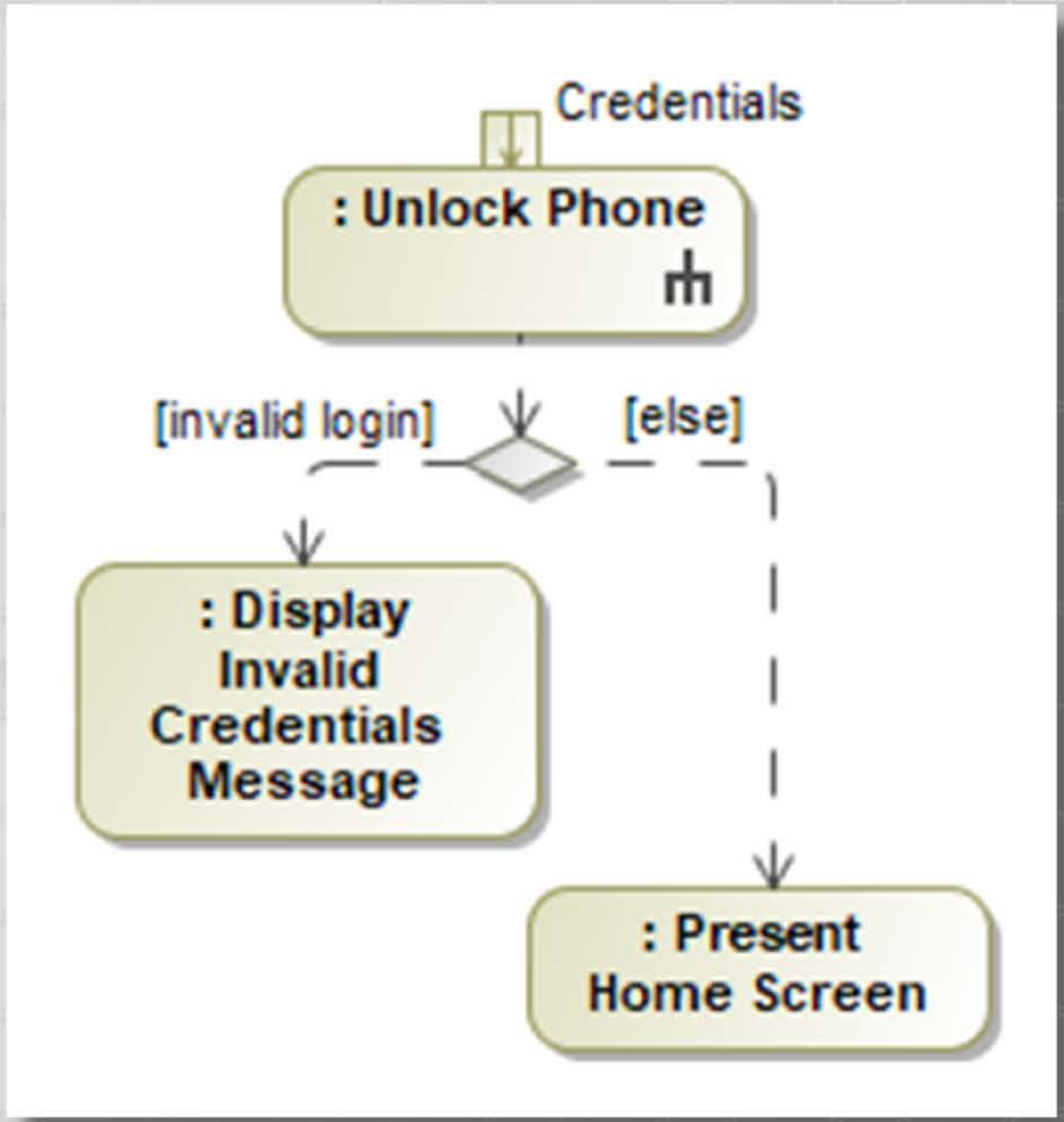
## Initial / Flow Final / Activity Final Nodes

- Initial nodes specify the formal start of the activity.
  - They are not necessary for simulation but it is certainly a best practice to include them on all activity diagrams.
- Flow final nodes specify the end to a specific path of execution in the activity.
  - Reaching a flow final node will stop the execution for that path but will not end the entire activity.
- Activity final nodes specify the end of the entire activity.
  - Reaching the activity final node will end the entire execution of that activity, even if other paths are still running.



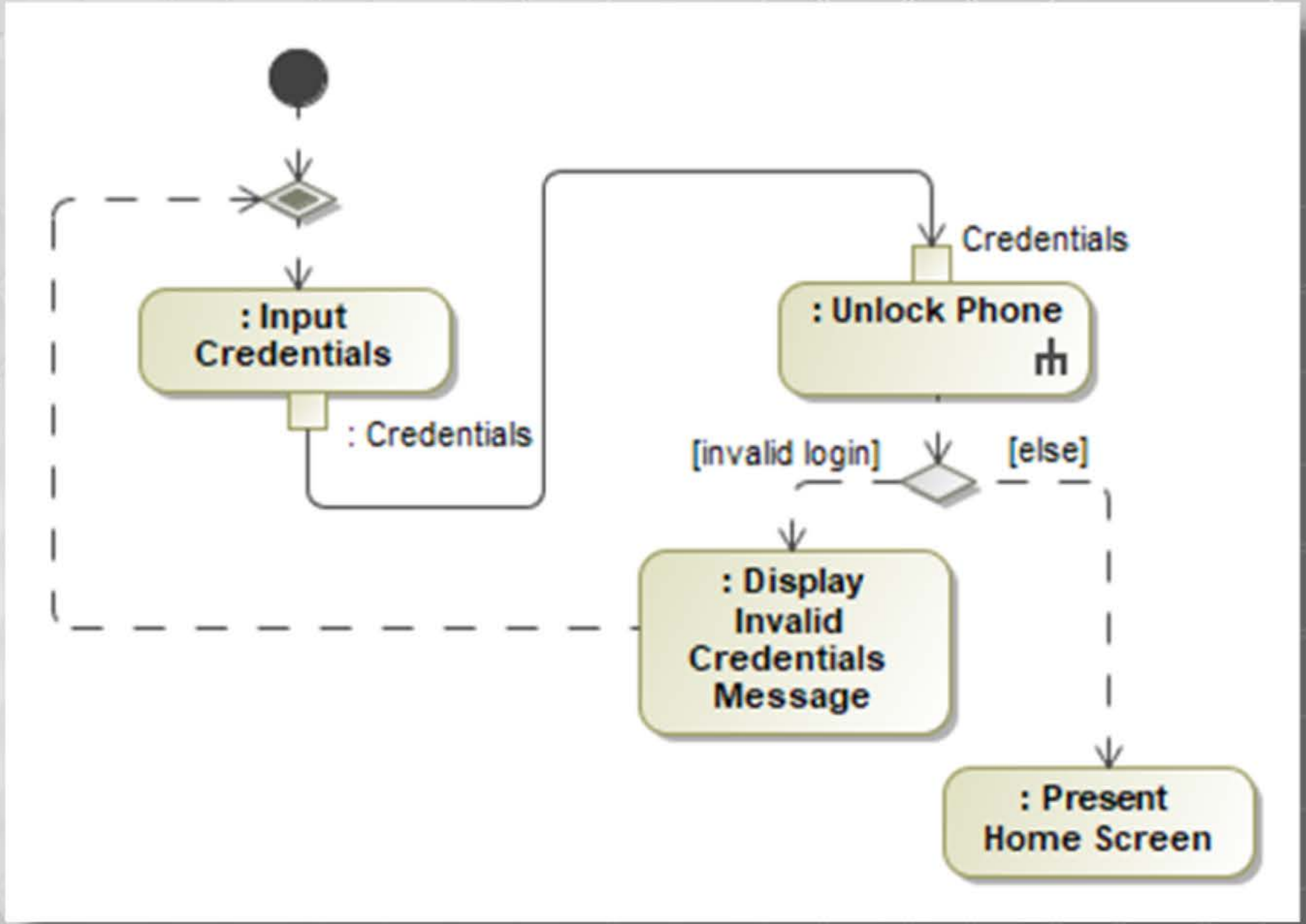
## Decision NODES

- Decision nodes allow the activity to take one of many alternative paths.
  - Decision nodes have a single incoming flow and many outgoing flows.
- Each outgoing flow of the decision node should have a <u>guard</u> specified for it.
- Guards are Boolean (true/false) expressions that are written with square brackets around them.
  - Guards can be written in English or with code that should evaluate to true or false.
- Exactly one path should have a guard that is true during simulation so that it is clear which path to take.
- If all paths are false, the path with the "else" keyword as a guard will be taken instead.



# Merge NODES

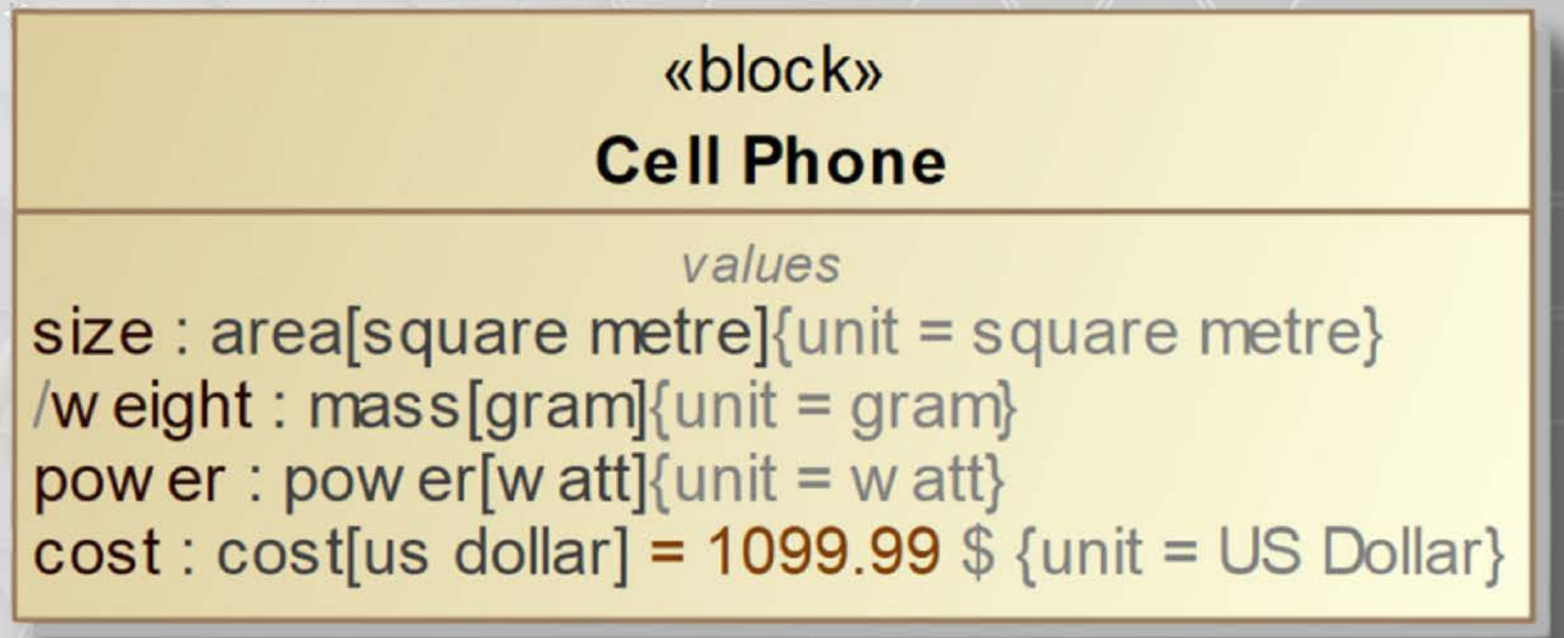
- Merge nodes allow for the convergence of multiple paths into a single path.
  - Merge nodes have many incoming flows but only one outgoing flow.
- Base on the simulation rules, actions with multiple incoming flows will wait for ALL incoming flows before executing.
- With a merge node, an action can funnel all of its flows into one and then execute if ANY of those flows are received.
- Merge nodes are essential for building loops since the initialization of the loop and the iterations of the loop both need to be accounted for.



## PARAMETRICS

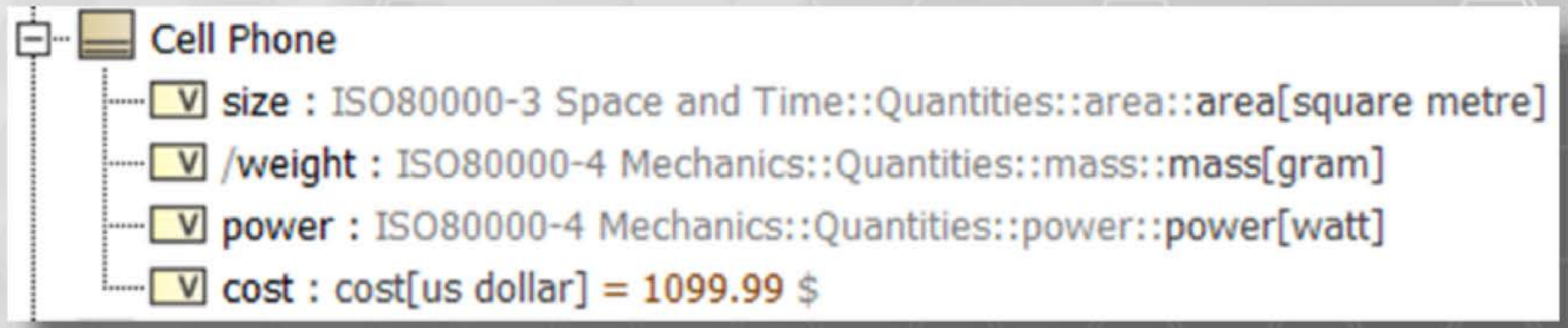
## Value PROPERTIES

- Value properties represent the quantitative characteristics of a block.
  - E.g.: the speed of a projectile, the remaining fuel of a rocket, the resolution of a screen, the paint color of a water gun, etc.
- Value properties can have Default Values which denote the typical value of that property for the block.
- Value properties that are going to be calculated, rather than defined, are marked with a "/" character in from of the name and are called derived value properties.
- Value properties are always typed by Value Types.



«block»
**Cell Phone**

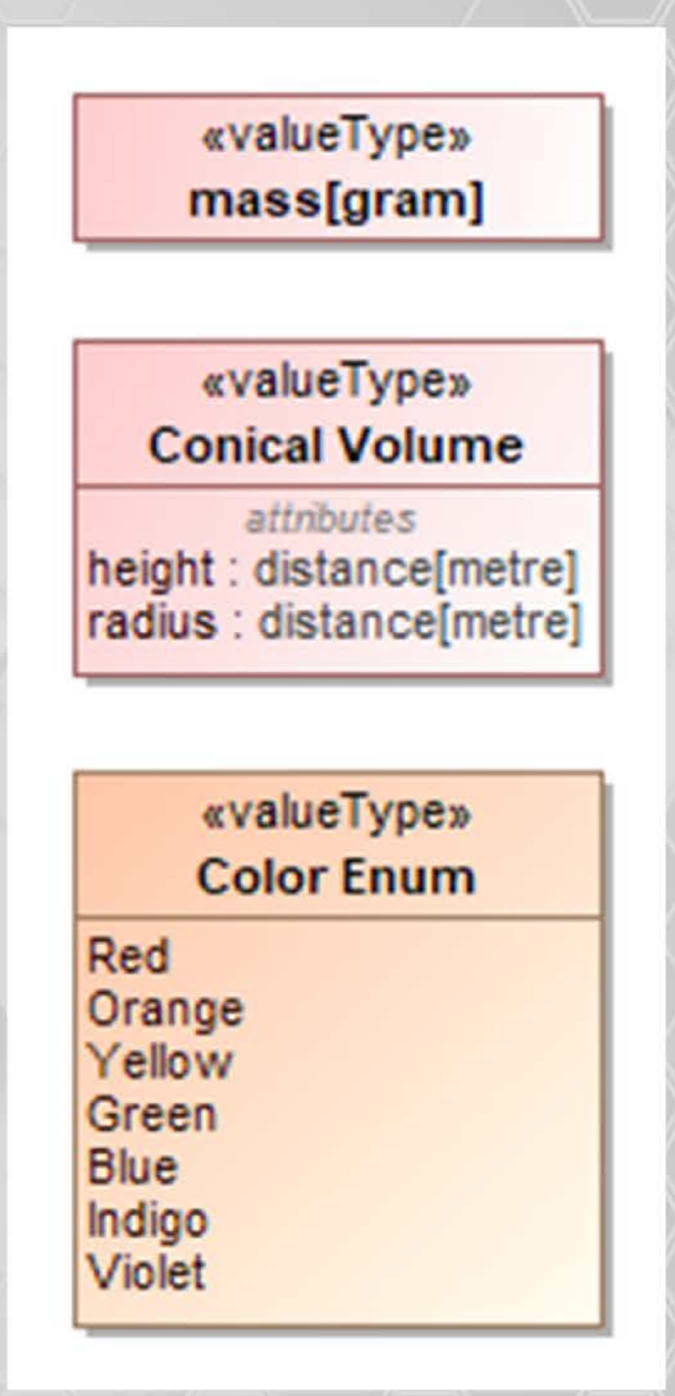
*values*
size : area[square metre]{unit = square metre}
/w eight : mass[gram]{unit = gram}
pow er : pow er[w att]{unit = w att}
cost : cost[us dollar] = 1099.99 $ {unit = US Dollar}



Cell Phone
- size : ISO80000-3 Space and Time::Quantities::area::area[square metre]
- /weight : ISO80000-4 Mechanics::Quantities::mass::mass[gram]
- power : ISO80000-4 Mechanics::Quantities::power::power[watt]
- cost : cost[us dollar] = 1099.99 $

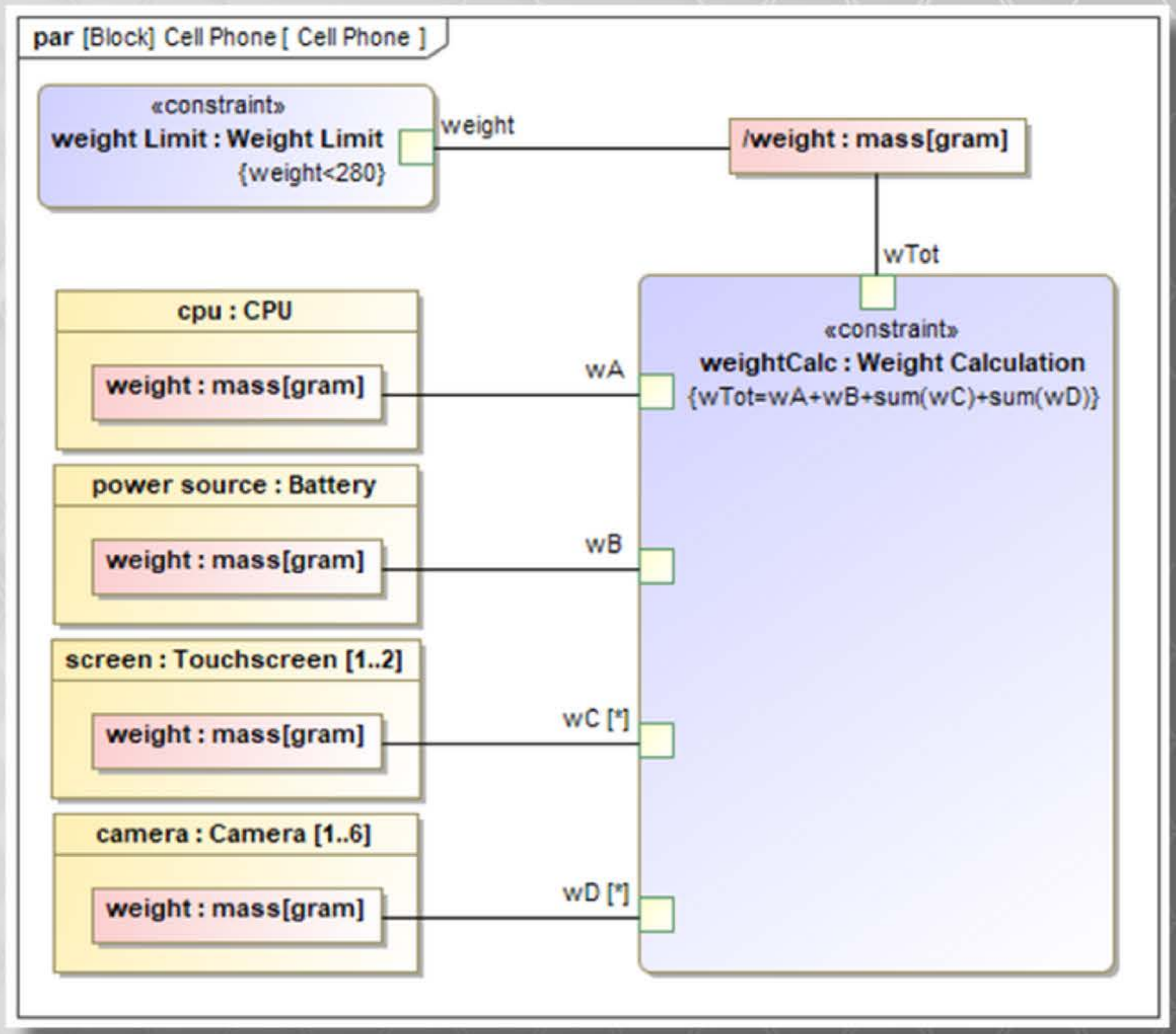
# Value TYPES

- Value Types are reusable elements used to specify the type of data that a value property can hold.
- Value Types come in three different categories:
  - **Simple**
    - Represents data types with scalar values
    - E.g.: speed, weight, cost, etc.
  - **Complex**
    - Represents data types composed of other data types
    - E.g.: area (length and width), volume (length, width, and height), etc.
  - **Enumeration**
    - Represents data types with a set of legal values to choose from
    - E.g.: color, flavor, classification, etc.
    - The options of an enumeration are captured in Enumeration Literals.



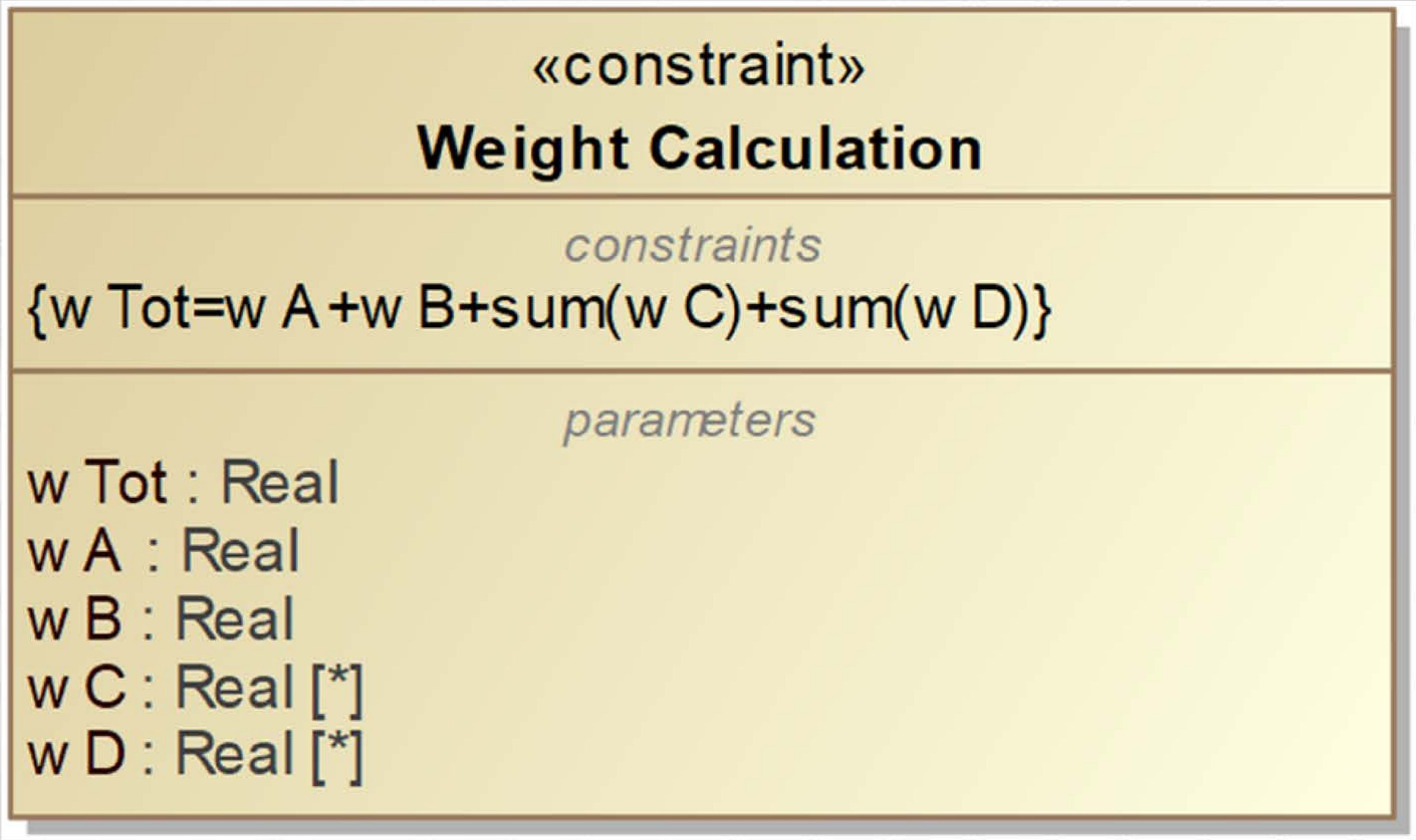
## Diagram Example & PURPOSE

- The Parametric Diagram is used to connect the value properties of blocks to the parameters of constraint expressions.
- The Parametric Diagram is not very useful for visualizing the system's specification, rather it is used primarily for simulation.
- These diagrams allow for automated performance requirements verification and enable trade study anlaysis.
- Parametric Diagrams are subtypes of Internal Block Diagrams and are contained within the block they specify.



## Constraining Values of BLOCKS

- Some value properties of blocks are derived from other values or change during behavioral simulation.
- Constraint Blocks are specialized blocks used for defining constraints that can solve for values of blocks or verify requirements.
- Constraint Blocks can connect to external solvers to “co-simulate”.
  - MagicDraw will send the input values to the external tool and get back the output values.
- Constraint Blocks have a constraint expression and a list of constraint parameters.

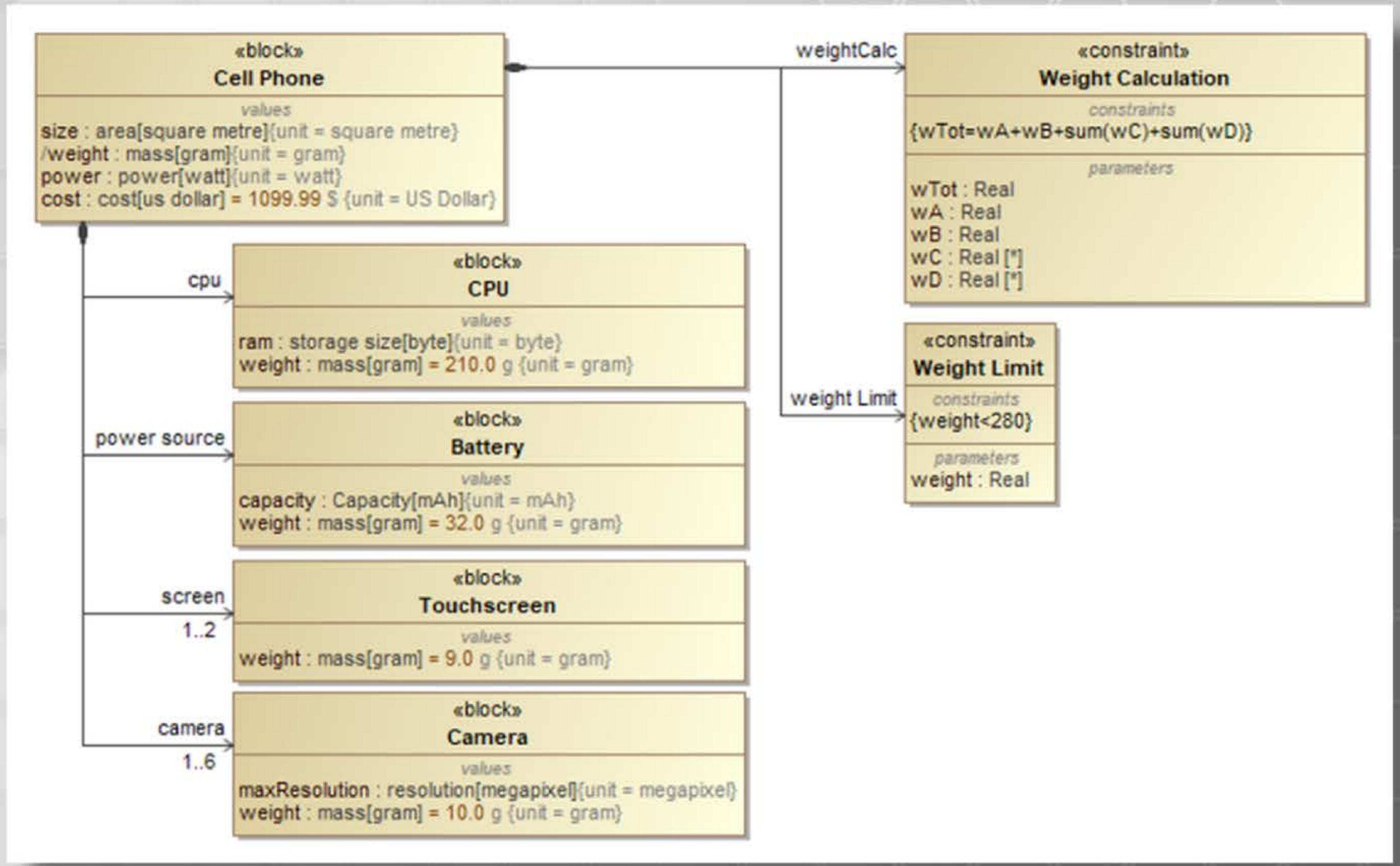


{w Tot=w A+w B+sum(w C)+sum(w D)}

w Tot : Real
w A : Real
w B : Real
w C : Real [*]
w D : Real [*]

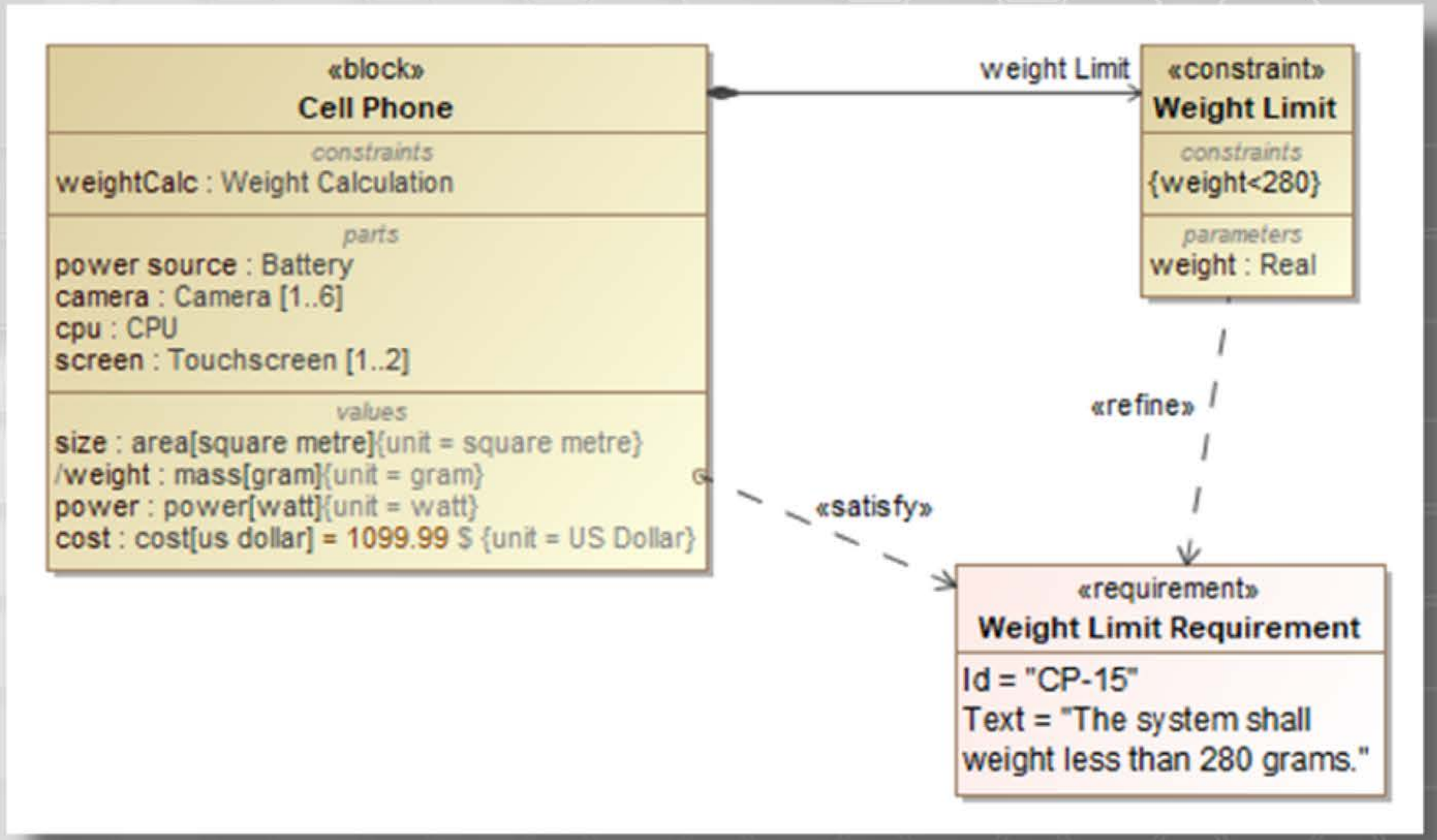
## Relations to BLOCKS

- Constraint Blocks are used to type constraint properties of blocks.
- Constraint properties directly tied to blocks will always be calculated for that block.
- Composition relationship can be used to relate constraint blocks to blocks.


«block»
Cell Phone
values
size : area[square metre]{unit = square metre}
/weight : mass[gram]{unit = gram}
power : power[watt]{unit = watt}
cost : cost[us dollar] = 1099.99 $ {unit = US Dollar}
weightCalc
«constraint»
Weight Calculation
constraints
{wTot=wA+wB+sum(wC)+sum(wD)}
parameters
wTot : Real
wA : Real
wB : Real
wC : Real [*]
wD : Real [*]
cpu
«block»
CPU
values
ram : storage size[byte]{unit = byte}
weight : mass[gram] = 210.0 g {unit = gram}
weight Limit
«constraint»
Weight Limit
constraints
{weight<280}
parameters
weight : Real
power source
«block»
Battery
values
capacity : Capacity[mAh]{unit = mAh}
weight : mass[gram] = 32.0 g {unit = gram}
screen
1..2
«block»
Touchscreen
values
weight : mass[gram] = 9.0 g {unit = gram}
camera
1..6
«block»
Camera
values
maxResolution : resolution[megapixel]{unit = megapixel}
weight : mass[gram] = 10.0 g {unit = gram}

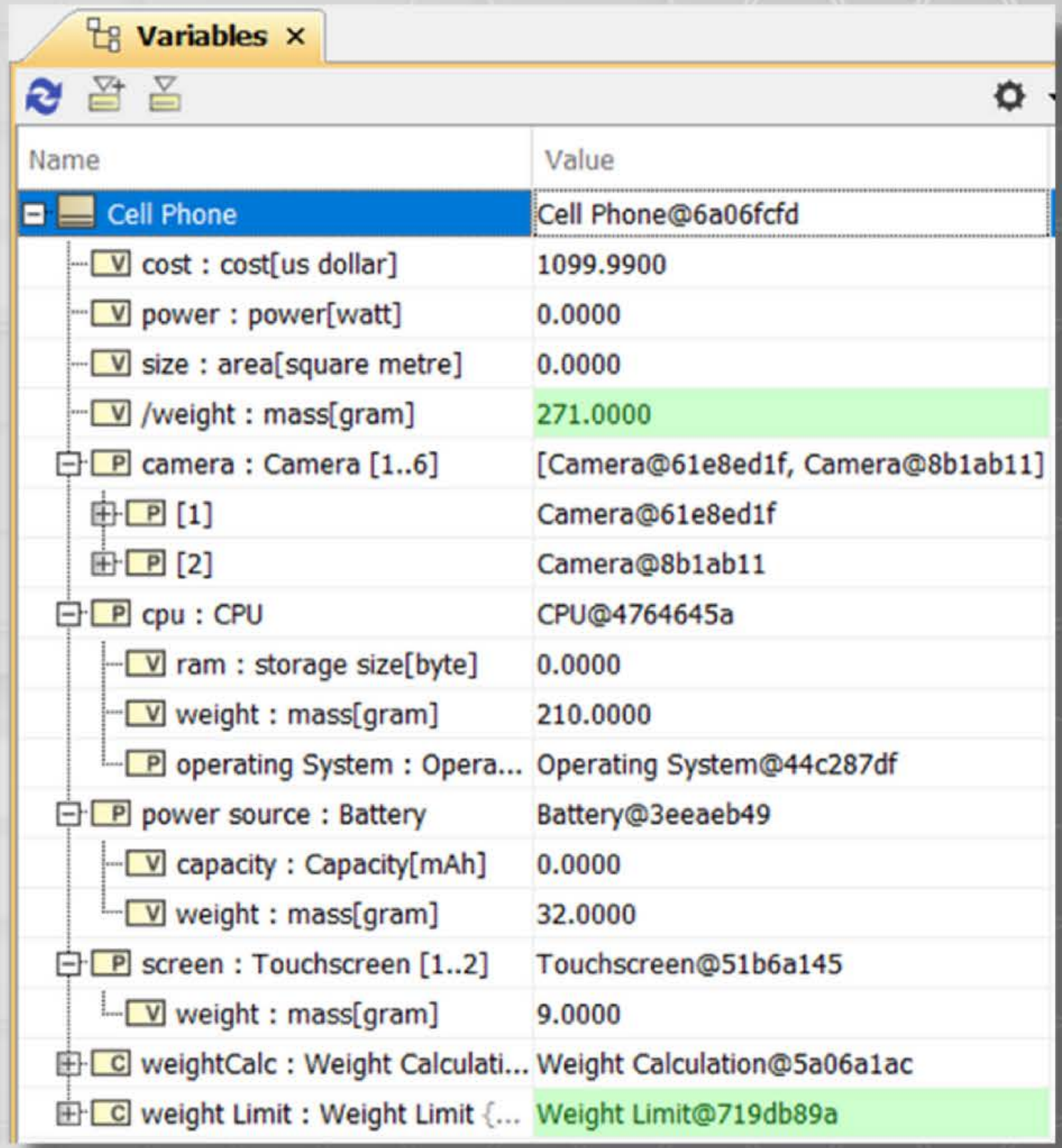
# Requirements TRACEABLILITY

- Parametric Diagrams are most often used for requirements verification.
- Requirements should be <u>refined by</u> Constraint Blocks since they represent the mathematically-formal expression for the textual requirement.
- Requirements should be <u>satisfied by</u> the value properties of the block since they represent the quantitative characteristics of the block.


«block»
Cell Phone
constraints
weightCalc : Weight Calculation
parts
power source : Battery
camera : Camera [1..6]
cpu : CPU
screen : Touchscreen [1..2]
values
size : area[square metre]{unit = square metre}
/weight : mass[gram]{unit = gram}
power : power[watt]{unit = watt}
cost : cost[us dollar] = 1099.99 $ {unit = US Dollar}
weight Limit
«constraint»
Weight Limit
constraints
{weight<280}
parameters
weight : Real
«refine»
«satisfy»
«requirement»
Weight Limit Requirement
Id = "CP-15"
Text = "The system shall weight less than 280 grams."

# Requirements VERIFICATION

- When simulating in MagicDraw, use the Variables tab to track the current values assigned to each property.
- Values can be manually changed by setting them in this window.
- Calculation should be performed automatically, and requirements verification should also be automated.
- Value properties highlighted in green have met their designated requirements while value properties in red have failed them.



| Name | Value |
|---|---|
| Cell Phone | Cell Phone@6a06fcfd |
| cost : cost[us dollar] | 1099.9900 |
| power : power[watt] | 0.0000 |
| size : area[square metre] | 0.0000 |
| /weight : mass[gram] | 271.0000 |
| camera : Camera [1..6] | [Camera@61e8ed1f, Camera@8b1ab11] |
| [1] | Camera@61e8ed1f |
| [2] | Camera@8b1ab11 |
| cpu : CPU | CPU@4764645a |
| ram : storage size[byte] | 0.0000 |
| weight : mass[gram] | 210.0000 |
| operating System : Opera... | Operating System@44c287df |
| power source : Battery | Battery@3eeaeb49 |
| capacity : Capacity[mAh] | 0.0000 |
| weight : mass[gram] | 32.0000 |
| screen : Touchscreen [1..2] | Touchscreen@51b6a145 |
| weight : mass[gram] | 9.0000 |
| weightCalc : Weight Calculati... | Weight Calculation@5a06a1ac |
| weight Limit : Weight Limit {... | Weight Limit@719db89a |

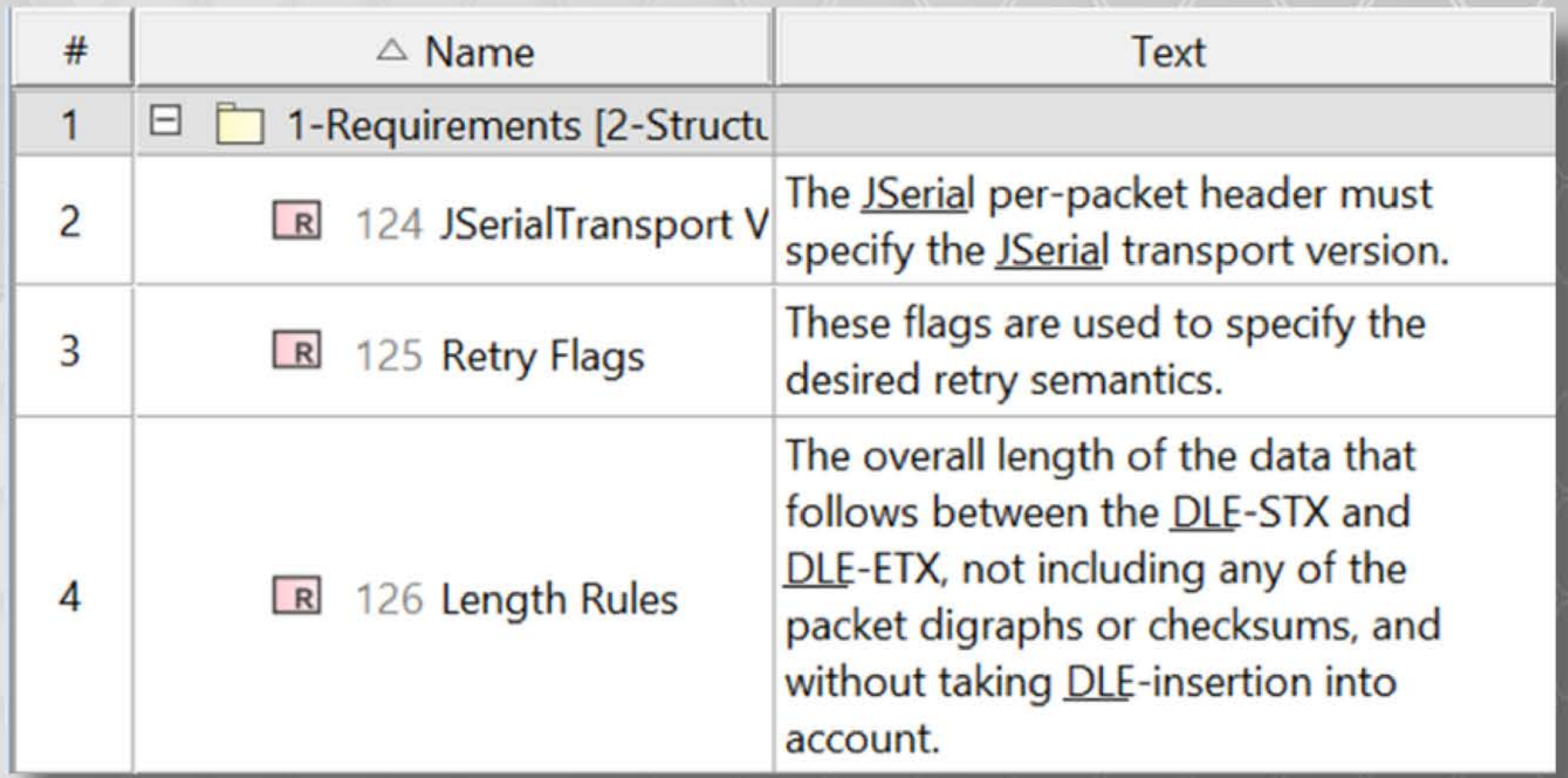
## DEMO

## STANDARDS DIGITIZATION

## DIGITAL STANDARDS WITH MBSE

# 1-REQUIREMENTS

- The first pass of digitization of a standard into MBSE is formalizing the requirement text into requirements.
- With text alone, one has to often scroll through the document repeatedly to verify implementations against the standard.
- MBSE enables automated verification of the requirements and is leveraged to directly trace the customer-defined system architecture to the standards they must comply with.

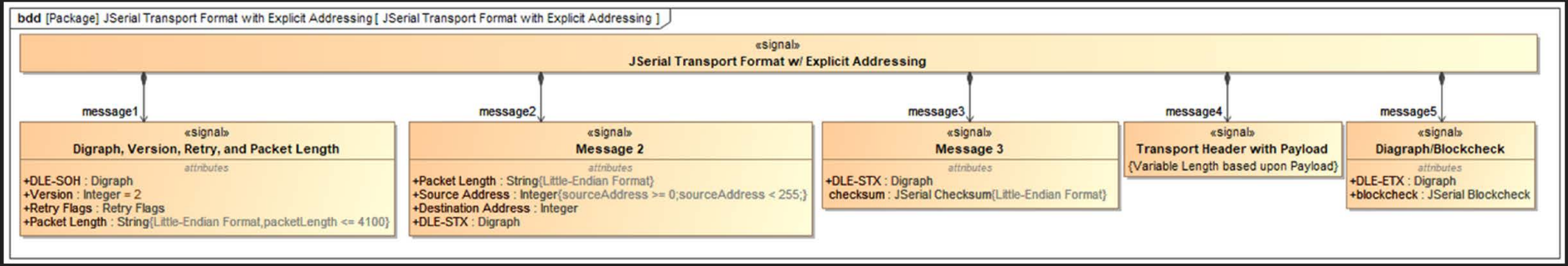


| # | △ Name | Text |
|---|---|---|
| 1 | ⊟ 1-Requirements [2-Structu | |
| 2 | R 124 JSerialTransport V | The JSerial per-packet header must specify the JSerial transport version. |
| 3 | R 125 Retry Flags | These flags are used to specify the desired retry semantics. |
| 4 | R 126 Length Rules | The overall length of the data that follows between the DLE-STX and DLE-ETX, not including any of the packet digraphs or checksums, and without taking DLE-insertion into account. |

## DEMO

## 2-STRUCTURE

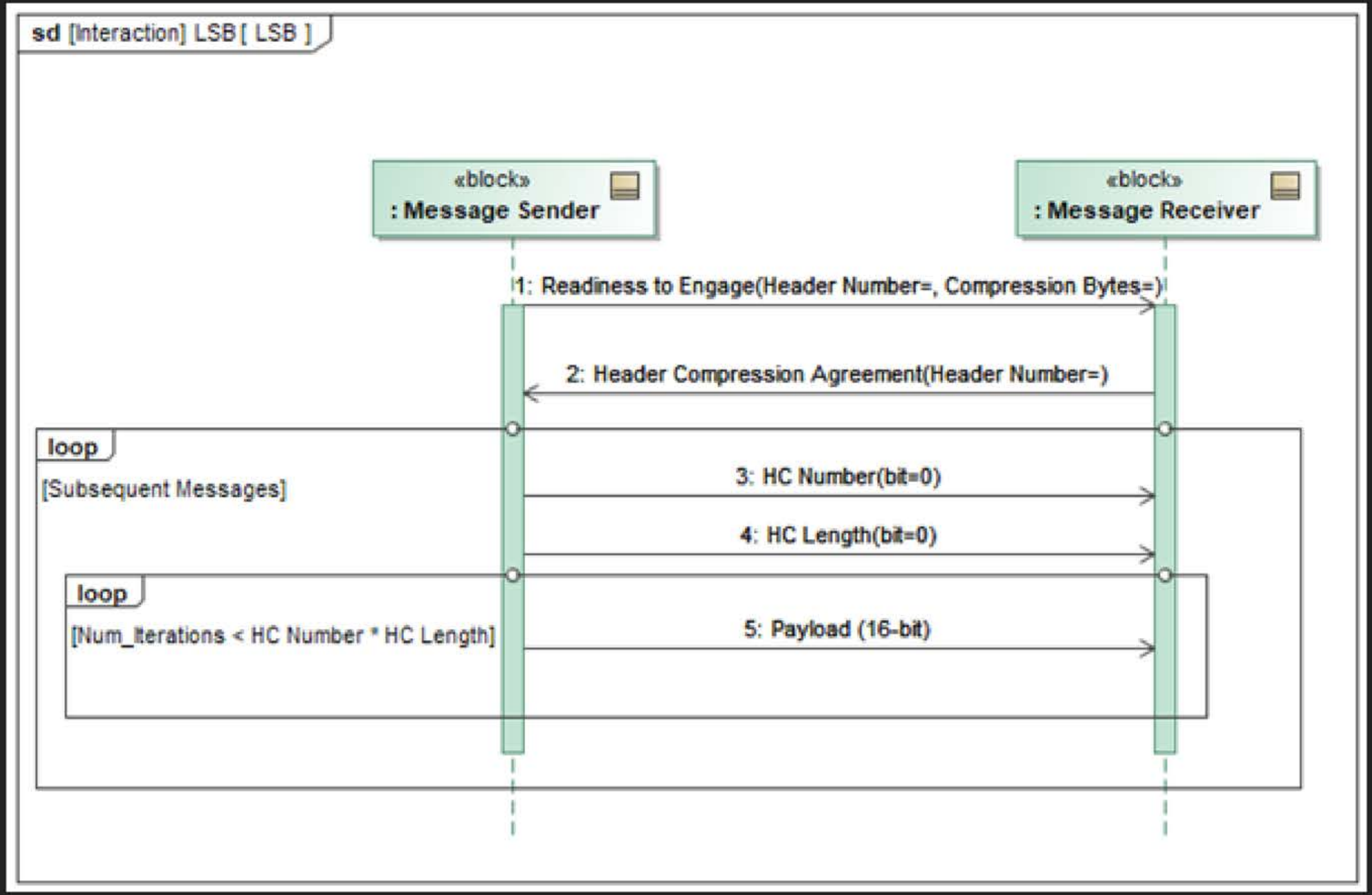
- After formalizing the text from the standard into requirements, the second pass is to develop the attributes of a “compliant” system for reuse by the users of the standard.
- The first form of this is the formalization of the structural side of the system architecture.



bdd [Package] JSerial Transport Format with Explicit Addressing [ JSerial Transport Format with Explicit Addressing ]
«signal»
JSerial Transport Format w/ Explicit Addressing
message1
message2
message3
message4
message5
«signal»
Digraph, Version, Retry, and Packet Length
attributes
+DLE-SOH : Digraph
+Version : Integer = 2
+Retry Flags : Retry Flags
+Packet Length : String{Little-Endian Format,packetLength <= 4100}
«signal»
Message 2
attributes
+Packet Length : String{Little-Endian Format}
+Source Address : Integer{sourceAddress >= 0;sourceAddress < 255;}
+Destination Address : Integer
+DLE-STX : Digraph
«signal»
Message 3
attributes
+DLE-STX : Digraph
checksum : JSerial Checksum{Little-Endian Format}
«signal»
Transport Header with Payload
{Variable Length based upon Payload}
«signal»
Diagraph/Blockcheck
attributes
+DLE-ETX : Digraph
+blockcheck : JSerial Blockcheck

## DEMO

## 3-BEHAVIOR

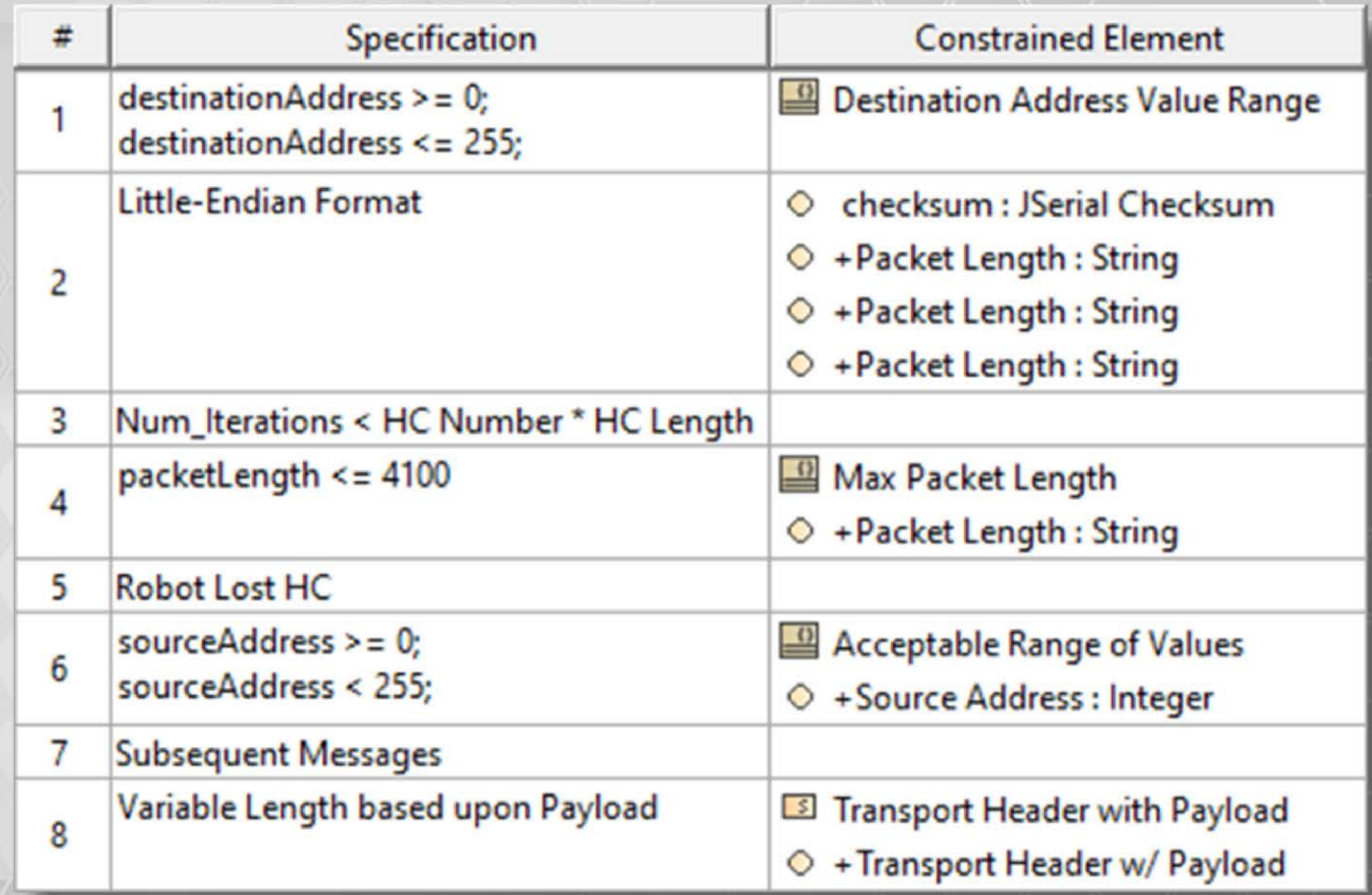

sd [Interaction] LSB [ LSB ]
«block» : Message Sender
«block» : Message Receiver
1: Readiness to Engage(Header Number=, Compression Bytes=)
2: Header Compression Agreement(Header Number=)
loop
[Subsequent Messages]
3: HC Number(bit=0)
4: HC Length(bit=0)
loop
[Num_Iterations < HC Number * HC Length]
5: Payload (16-bit)

- In addition to the structures being extracted from the requirements, the behavior, and example scenarios are able to as well.
- This enables the Systems Engineer to better understand the expected functionality of their system in order to be compliant with said standard.

## DEMO

# 4-PARAMETRIC

- Within MBSE, the pillar of parametrics represents the mathematical model defined to verify that a proposed architecture will meet it’s imposed requirements.
- We use constraints to enable us to do this. For this step, we will parse out reusable constraints from each performance-based or physical requirement which will then be imposed on the structures which must ensure compliance.



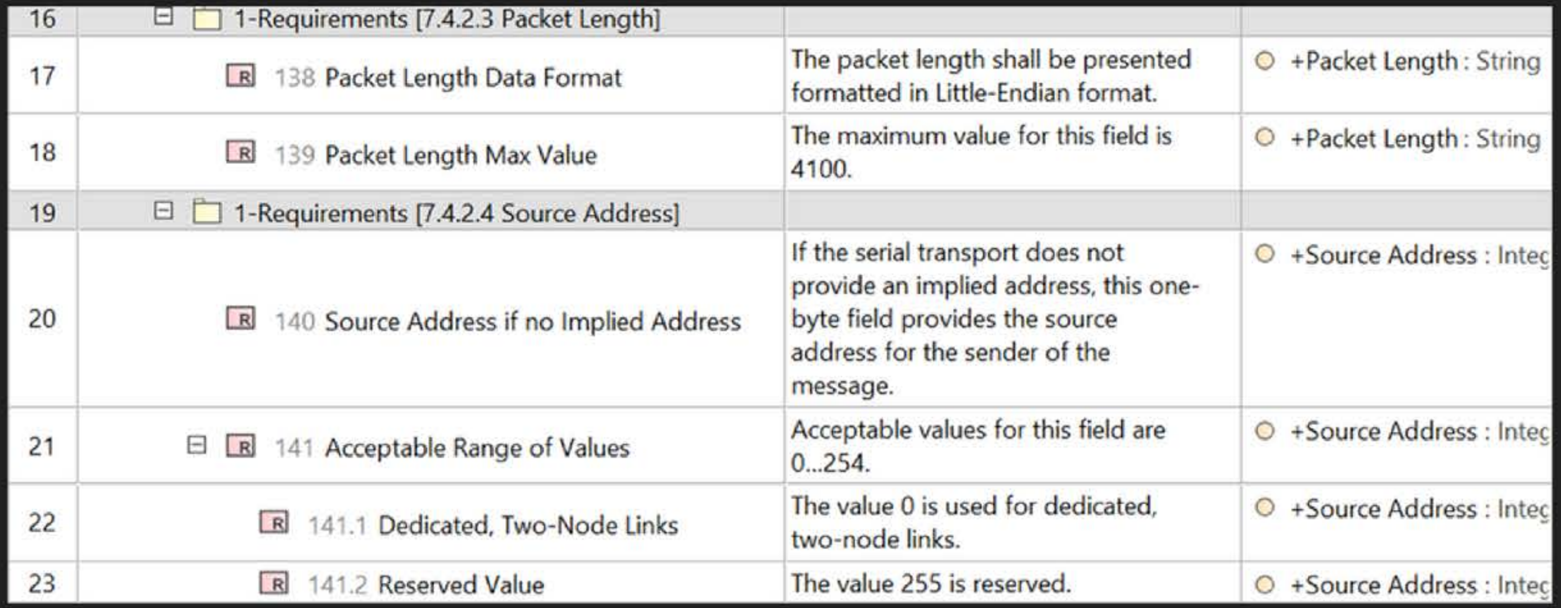
| # | Specification | Constrained Element |
|---|---|---|
| 1 | destinationAddress >= 0;<br>destinationAddress <= 255; | Destination Address Value Range |
| 2 | Little-Endian Format | ◇ checksum : JSerial Checksum<br>◇ +Packet Length : String<br>◇ +Packet Length : String<br>◇ +Packet Length : String |
| 3 | Num_Iterations < HC Number * HC Length | |
| 4 | packetLength <= 4100 | Max Packet Length<br>◇ +Packet Length : String |
| 5 | Robot Lost HC | |
| 6 | sourceAddress >= 0;<br>sourceAddress < 255; | Acceptable Range of Values<br>◇ +Source Address : Integer |
| 7 | Subsequent Messages | |
| 8 | Variable Length based upon Payload | Transport Header with Payload<br>◇ +Transport Header w/ Payload |

## DEMO

## DIGITAL STANDARDS WITH MBSE BENEFITS

# TRACEABILITY

MOST OF THE VALUE SURROUNDING MBSE IS REGARDING IMPROVED TRACEABILITY.



| | | | |
|---|---|---|---|
| 16 | ⊟ 1-Requirements [7.4.2.3 Packet Length] | | |
| 17 | 138 Packet Length Data Format | The packet length shall be presented formatted in Little-Endian format. | ○ +Packet Length : String |
| 18 | 139 Packet Length Max Value | The maximum value for this field is 4100. | ○ +Packet Length : String |
| 19 | ⊟ 1-Requirements [7.4.2.4 Source Address] | | |
| 20 | 140 Source Address if no Implied Address | If the serial transport does not provide an implied address, this one-byte field provides the source address for the sender of the message. | ○ +Source Address : Integ |
| 21 | ⊟ 141 Acceptable Range of Values | Acceptable values for this field are 0...254. | ○ +Source Address : Integ |
| 22 | 141.1 Dedicated, Two-Node Links | The value 0 is used for dedicated, two-node links. | ○ +Source Address : Integ |
| 23 | 141.2 Reserved Value | The value 255 is reserved. | ○ +Source Address : Integ |

WITH SUFFICIENT TRACEABILITY, ONE CAN QUERY THE MODEL TO AUTOMATE IMPACT ASSESSMENTS AND RUN WHAT-IF SCENARIOS, AS WELL AS DETERMINE IN A CROSS-CUTTING FASHION ALL REQUIREMENTS THAT THEY SYSTEM ARCHITECTURE IS EXPECTED TO MEET.

## REUSE

Biggest ROI for digitization of standards in MBSE.

The AS5669A model, the example for this deck, took roughly 24 man-hours to parse out a 55-page standard manually. Likely if your organization is using MBSE, your staff are doing the same thing per standard, often per project.

Which is why we advocate for library projects, per standard, to reuse the information contained within to prevent unnecessary duplication of effort.

ENOLA

## CONTACT US

www.enola.com

sales@enola.com

+1 877 281 7341

linkedin.com/company/enolatech