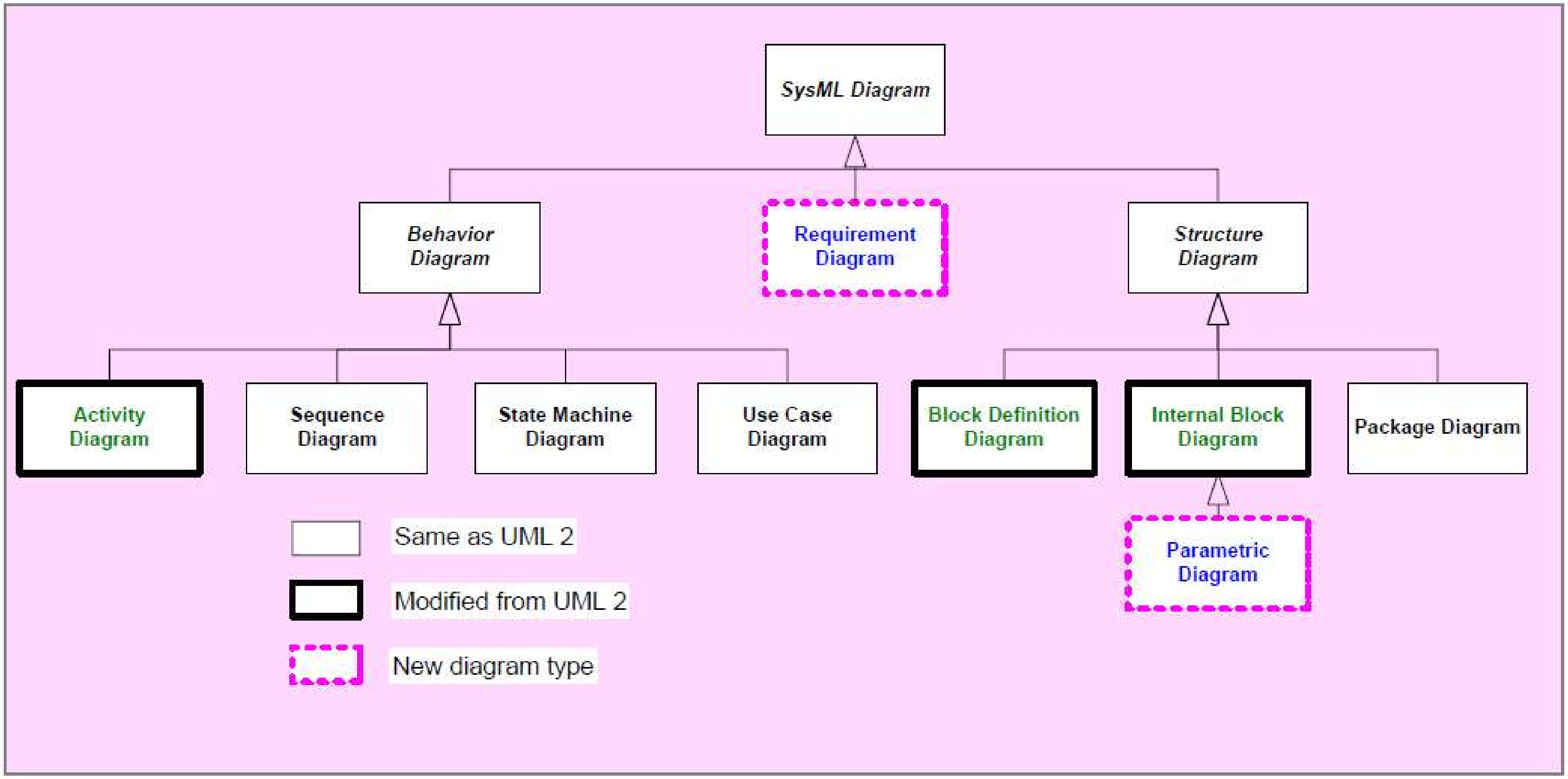
Systems Engineering

mariaiulianadascalu@gmail.com

SysML Diagram Taxonomy

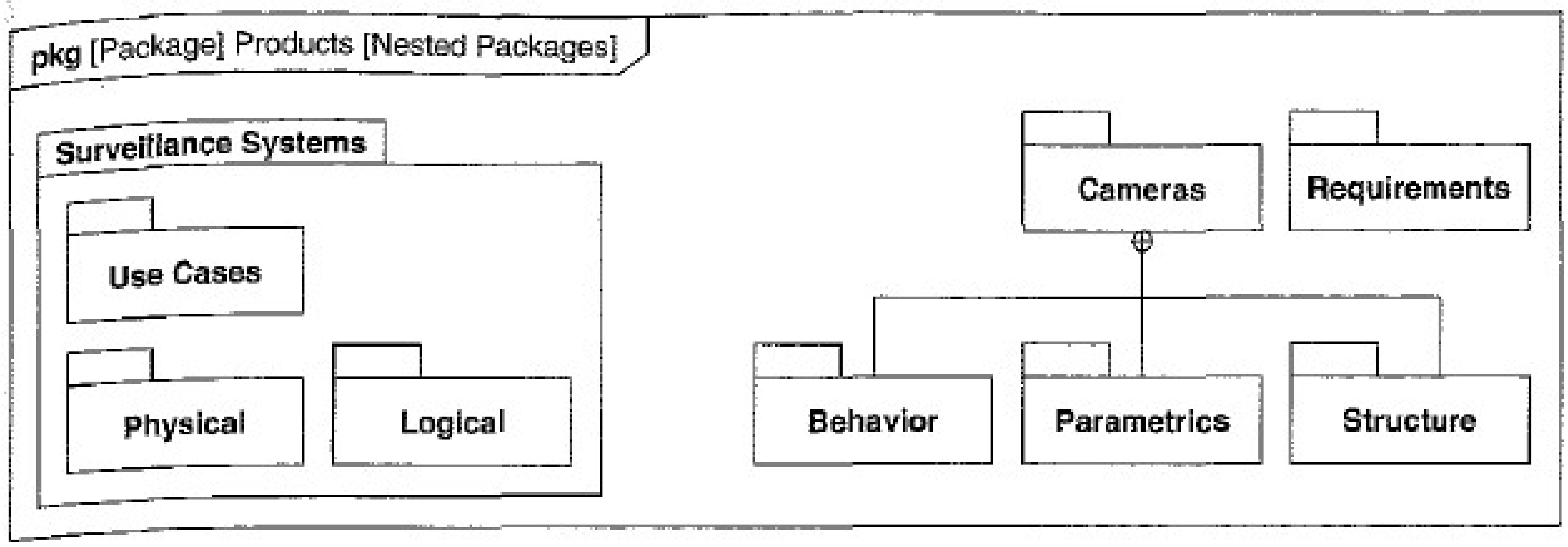


The Package Diagram

- In SysML, the fundamental unit of model organization is the package.
- Packages and their contents are shown on a package diagram.
- Packages are both containers and namespaces.
- A package may contain the following model elements: blocks, use cases, activities, packages.
- A package is a namespace for the packageable elements it contains.
- The complete diagram label for a package diagram is as follows:

  pkg [package type] package name [diagram name]
- Package types: model, package, model library, or view

Example


pkg [Package] Products [Nested Packages]
Surveillance Systems
Use Cases
Physical
Logical
Cameras
Requirements
Behavior
Parametrics
Structure

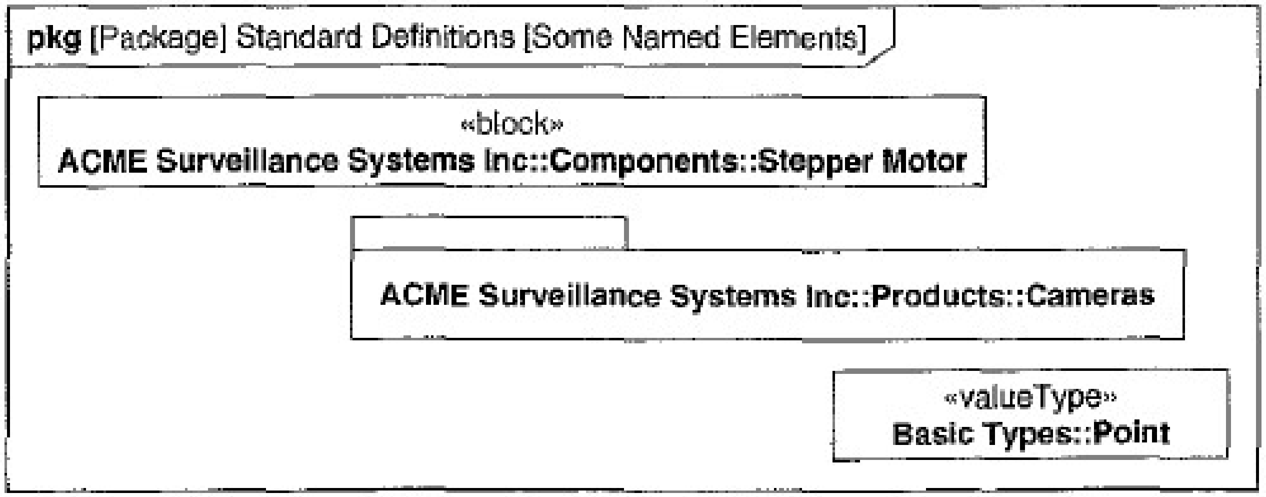
Package Types

- Packages are used to partition elements of the model into coherent units that can be subject to access control, model navigation, configuration management, and other considerations.
- A model in SysML is a top-level package in a nested package hierarchy and it is understood to represent a complete description of a system.

Principles of Organizing a Package Hierarchy

- By system hierarchy (e.g., system level, subsystem level, component level)
- By process life cycle where each model sub-package represents a stage in the process (e.g., requirements analysis, system design)
- By teams that worked on the model (e.g., Requirements Team, Integrated Product Team)
- By the type of model elements contained in it (e.g., requirements, behaviour, structure)
- By model elements that are likely to change together
- By model elements organized to support reuse (e.g., model libraries)
- By other logical or cohesive groupings of model elements based on defined model-partitioning criteria
- A combination of the above principles

Qualified Names within a Containment Hierarchy


pkg [Package] Standard Definitions [Some Named Elements]
«block»
ACME Surveillance Systems Inc::Components::Stepper Motor
ACME Surveillance Systems Inc::Products::Cameras
«valueType»
Basic Types::Point

- A::B::X => X is an element model contained in package B, which is contained in package A

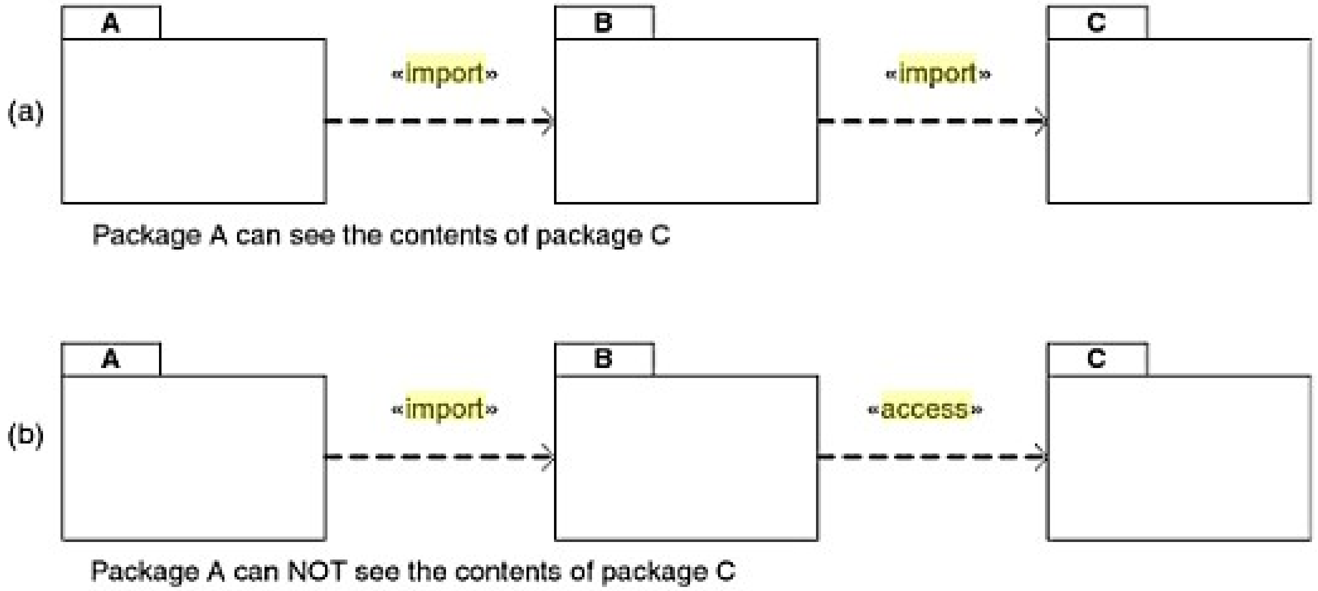
Importing Model Elements to Packages

- A package import applies to an entire package, and all the model elements of the Source package are imported into the target namespace.
- An element import applies to a single model element, and may be used when it is unnecessary and possibly confusing to import all the elements of a package.
- A name clash occurs when two or more model elements in the target namespace would have the same names as the result of imports.
- An element import has an alias field that can be used to provide an alternate name for a model element to prevent a name clash in the target namespace.

Visibility within Packages

- The named elements recognized within a namespace, whether through direct containment or as a result of being imported, are called members.
- Members have a visibility, either public or private, within their namespace.
- The visibility of a member determines whether it can be imported into another namespace.

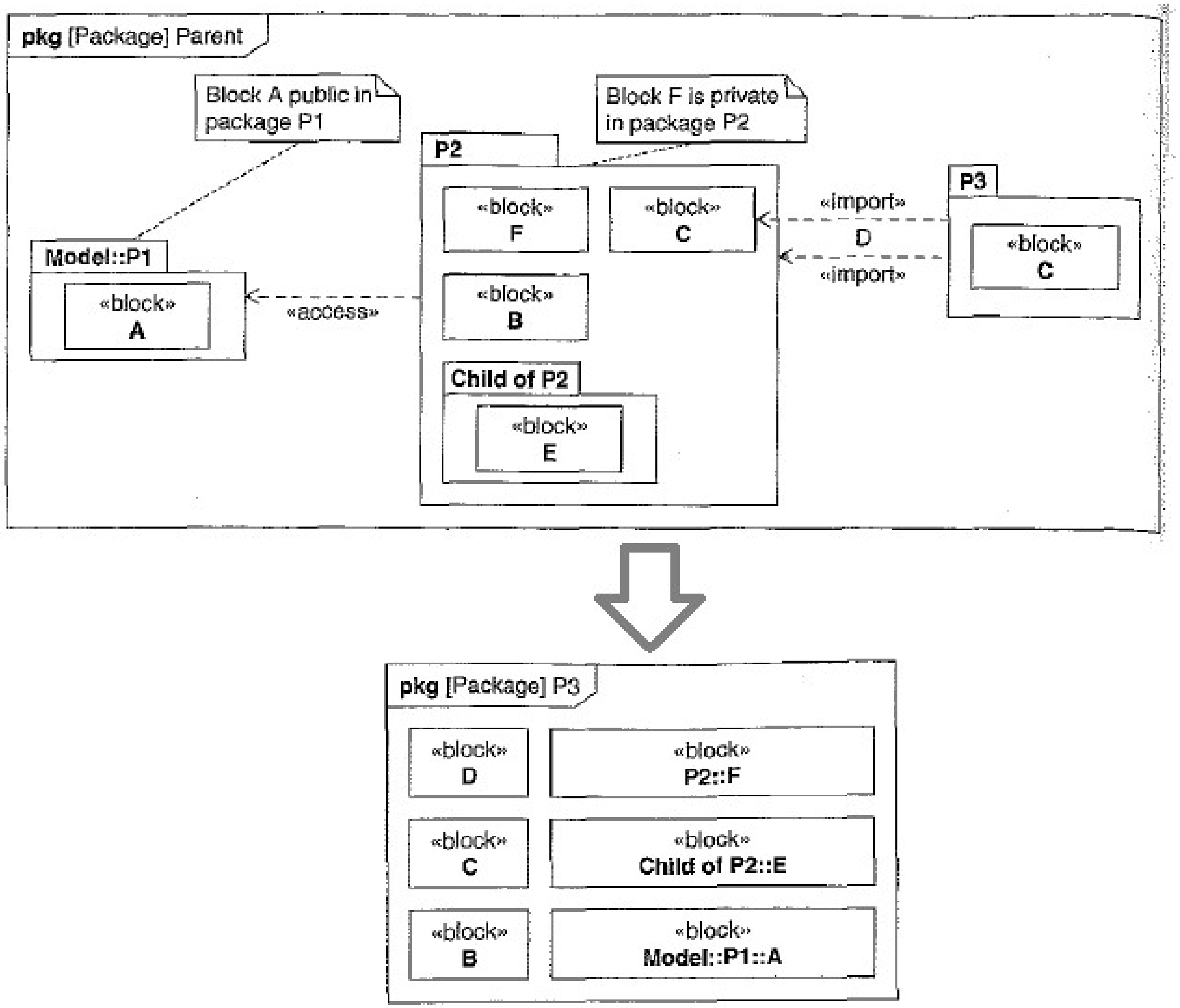
≪import≫ and ≪access≫ differ in the *visibility* of the information that is imported. ≪import≫ indicates that a *public* import is being performed. ≪access≫ indicates that a *private* import is being performed. What does this mean? Consider the two examples in Figure 4.30.


(a)
A
«import»
B
«import»
C
Package A can see the contents of package C
(b)
A
«import»
B
«access»
C
Package A can NOT see the contents of package C

Importing packages using ≪import≫ and ≪access≫

In example a) package B imports the contents of package C using the public ≪import≫ dependency. Package A then imports the contents of package B using the ≪import≫ dependency. Since A has imported B and B has *publicly* imported C, package A can also see the contents on package C.

In example b) package B imports the contents of package C using the private ≪access≫ dependency. Package A then imports the contents of package B using the ≪import≫ dependency. Since A has imported B and B has *privately* imported C, package A *cannot* see the contents on package C.


pkg [Package] Parent
Block A public in package P1
Block F is private in package P2
P2
«block» F
«block» C
«block» B
Child of P2
«block» E
Model::P1
«block» A
«access»
«import»
D
«import»
P3
«block» C
pkg [Package] P3
«block» D
«block» P2::F
«block» C
«block» Child of P2::E
«block» B
«block» Model::P1::A

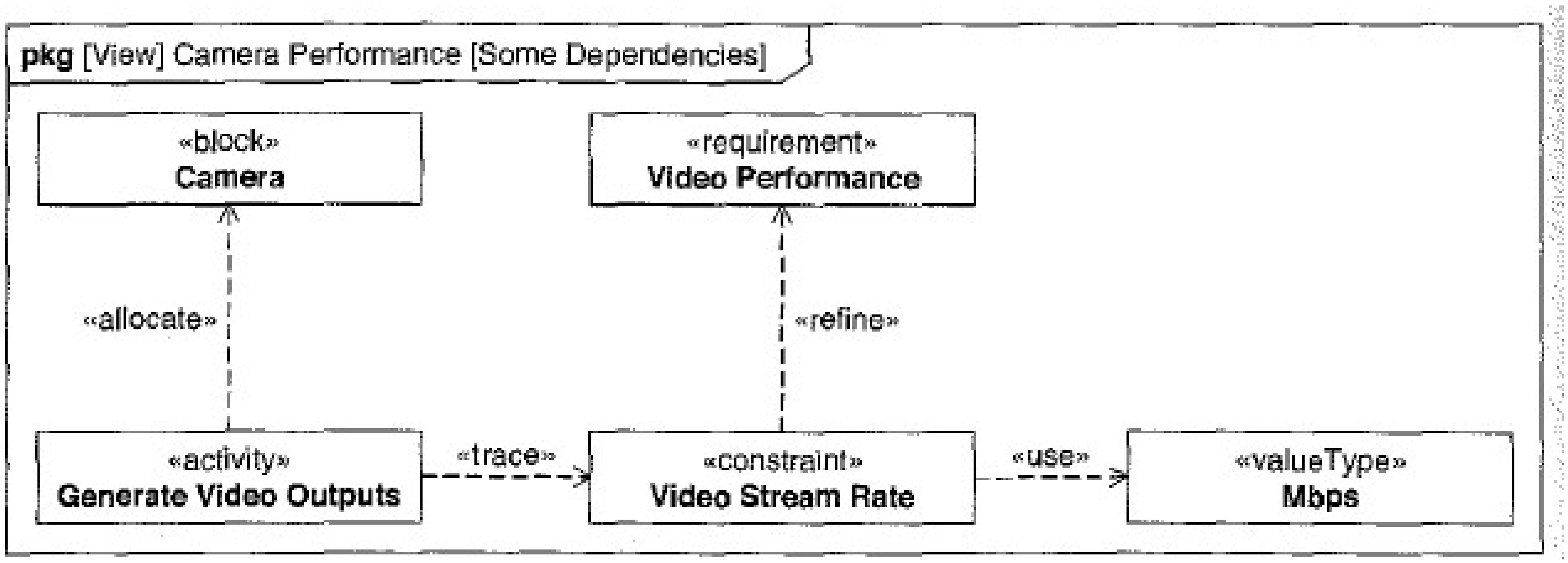
Showing Dependencies between Packageable Elements

- A dependency relationship can be applied between packageable elements to indicate that a change in the element on one end of the dependency may result in a change in the element on the other end of the dependency (client-supplier elements).

Types of Dependencies

- **Use**-indicates that the client uses the supplier as part of its definition
- **Refine**-indicates that the client represents an increase in detail compared to the specification of the supplier
- **Allocate**-indicates that the client realizes the specification expressed in the description of the supplier
- **Trace**-indicates that there is a linkage between the client and supplier without imposing the more significant semantic constraints of a more precise relationship

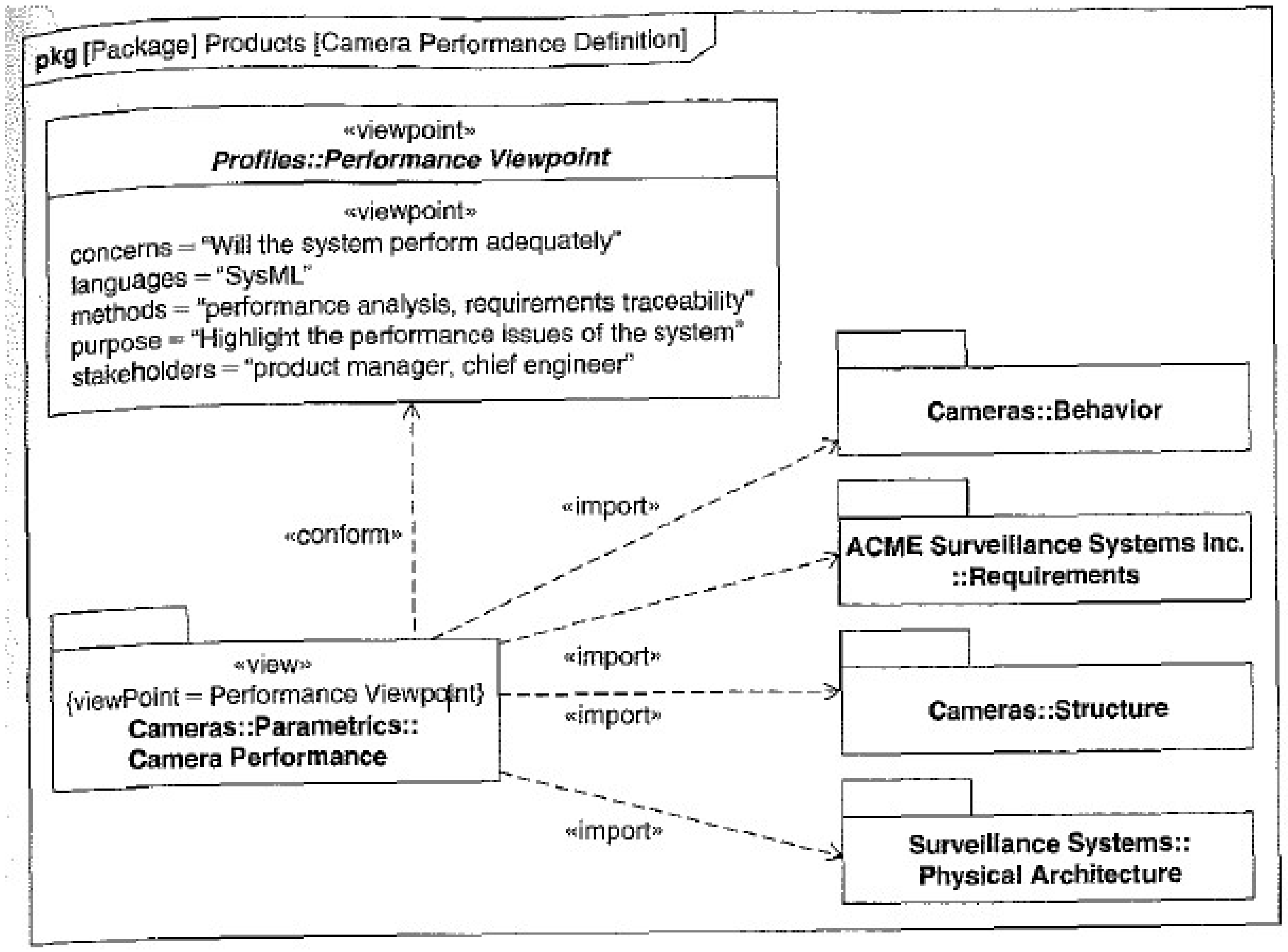
Example

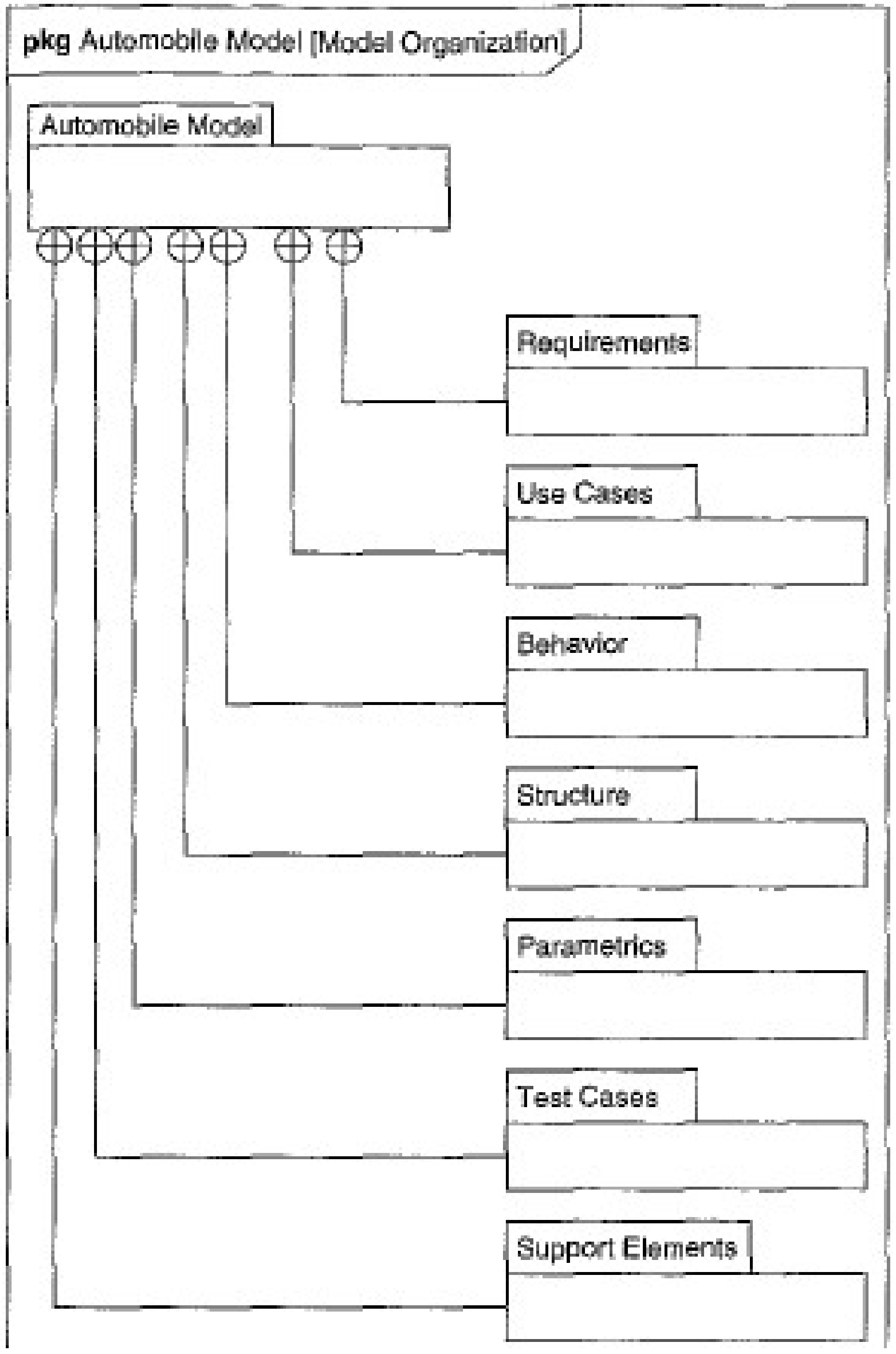

pkg [View] Camera Performance [Some Dependencies]
«block»
Camera
«requirement»
Video Performance
«allocate»
«refine»
«activity»
Generate Video Outputs
«trace»
«constraint»
Video Stream Rate
«use»
«valueType»
Mbps

Views and Viewpoints

- A view is a type of package that conforms to a viewpoint.
- A viewpoint describes a perspective of interest to a set of stakeholders that is used to specify a view of a model. A viewpoint includes a set of properties that identify:
  - The purpose or reason for taking this perspective
  - The stakeholders who have an interest in this perspective
  - The concerns that the stakeholders wish to address
  - The languages used to present the view
  - The methods *used to establish the view*

Example

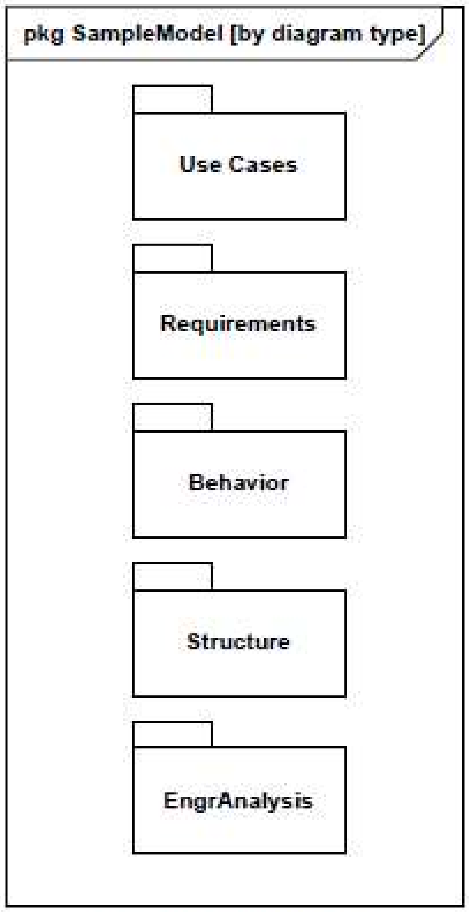

pkg [Package] Products [Camera Performance Definition]
«viewpoint»
Profiles::Performance Viewpoint
«viewpoint»
concerns = "Will the system perform adequately"
languages = "SysML"
methods = "performance analysis, requirements traceability"
purpose = "Highlight the performance issues of the system"
stakeholders = "product manager, chief engineer"
«conform»
«view»
{viewPoint = Performance Viewpoint}
Cameras::Parametrics::
Camera Performance
«import»
«import»
«import»
«import»
Cameras::Behavior
ACME Surveillance Systems Inc.
::Requirements
Cameras::Structure
Surveillance Systems::
Physical Architecture



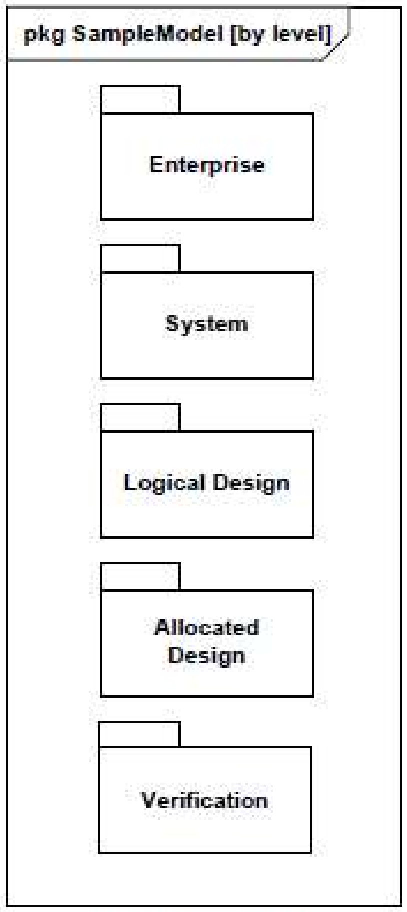
Exercise

- For a model that you are trying to build, discuss the type of model organization that is appropriate for it.

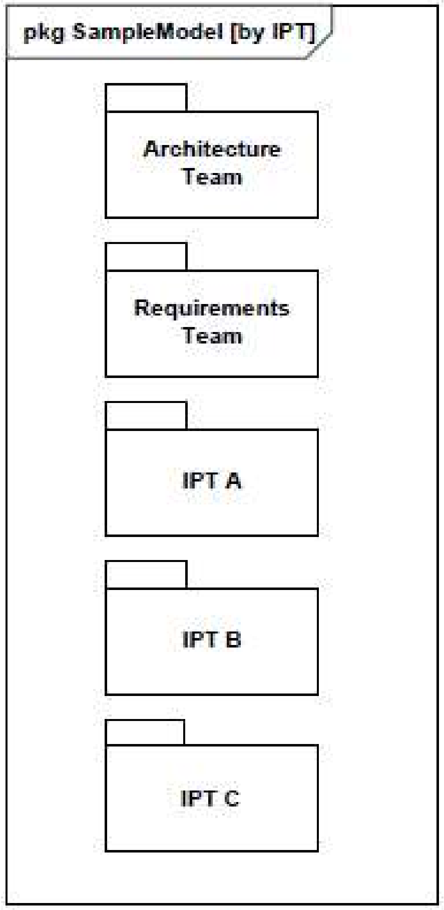
Example



By Diagram Type



By Hierarchy



By IPT

Blocks

- Definition:
  - The block is the modular unit of structure in SysML that is used to define a type of system, system component, or item that flows through the system, as well as conceptual entities or logical abstractions.
- Modeled with:
  - Block definition diagram
  - Internal block diagram

Block Definition Diagram (BDD)

- Defines block characteristics in terms of their features
  - Structural features
    - Properties
      - Parts
      - Values
      - References
    - Ports
      - Flow ports
      - Standard ports
  - Behavioral features
    - Operations
    - Receptions
  - Hierarchical relations related features
- Header:

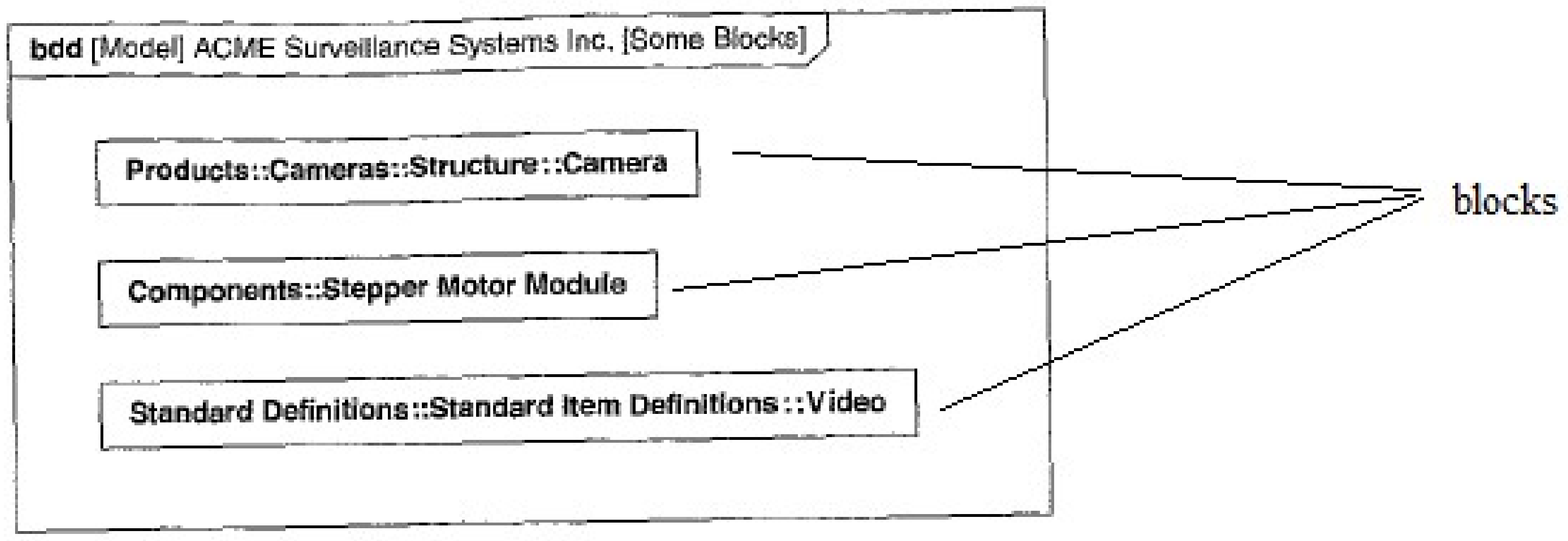
bdd [model element type] model element name [diagram name]

Internal Block Diagram (IBD)

- Describes the internal structure of a block in terms of how its parts are interconnected
- Header:

  ibd [Block] block name [diagram name]

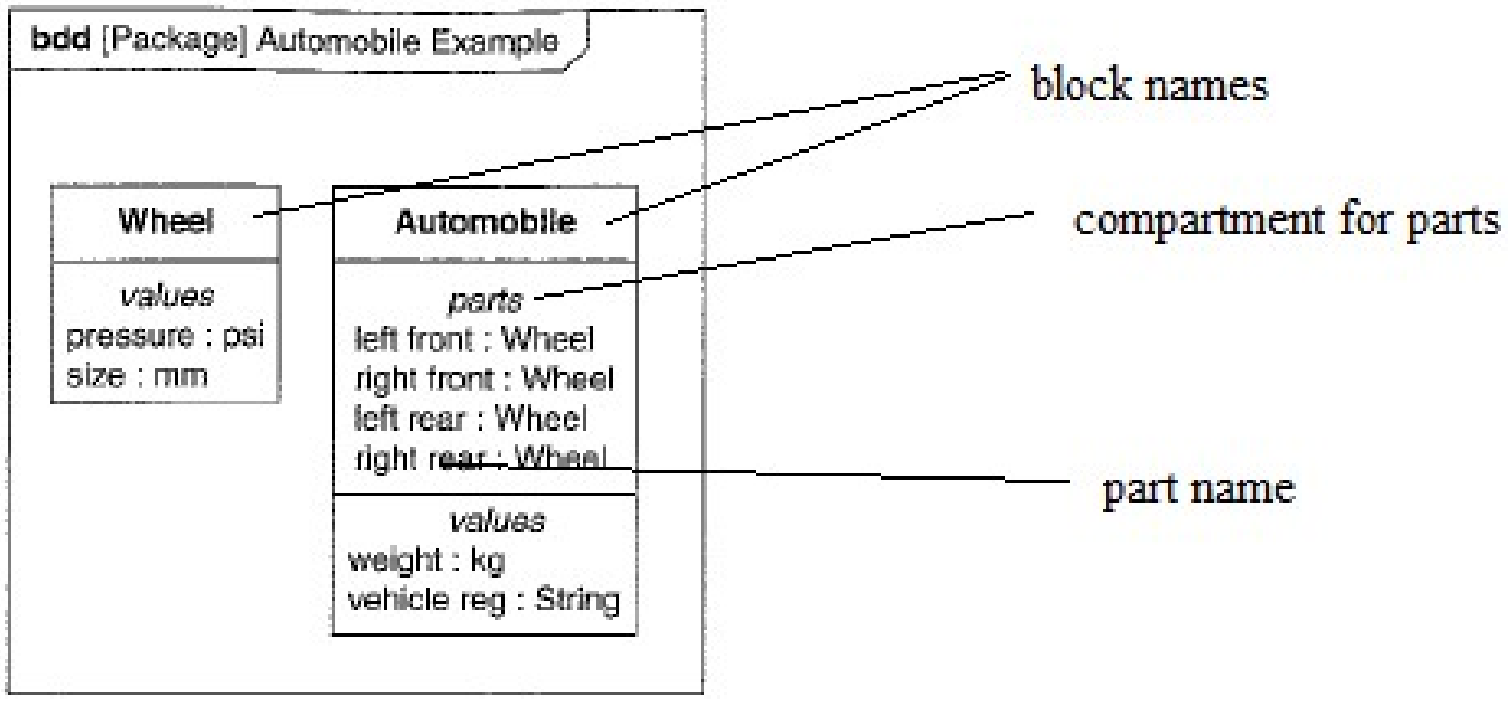
Example


bdd [Model] ACME Surveillance Systems Inc. [Some Blocks]
Products::Cameras::Structure::Camera
Components::Stepper Motor Module
Standard Definitions::Standard Item Definitions::Video
blocks

Part Properties

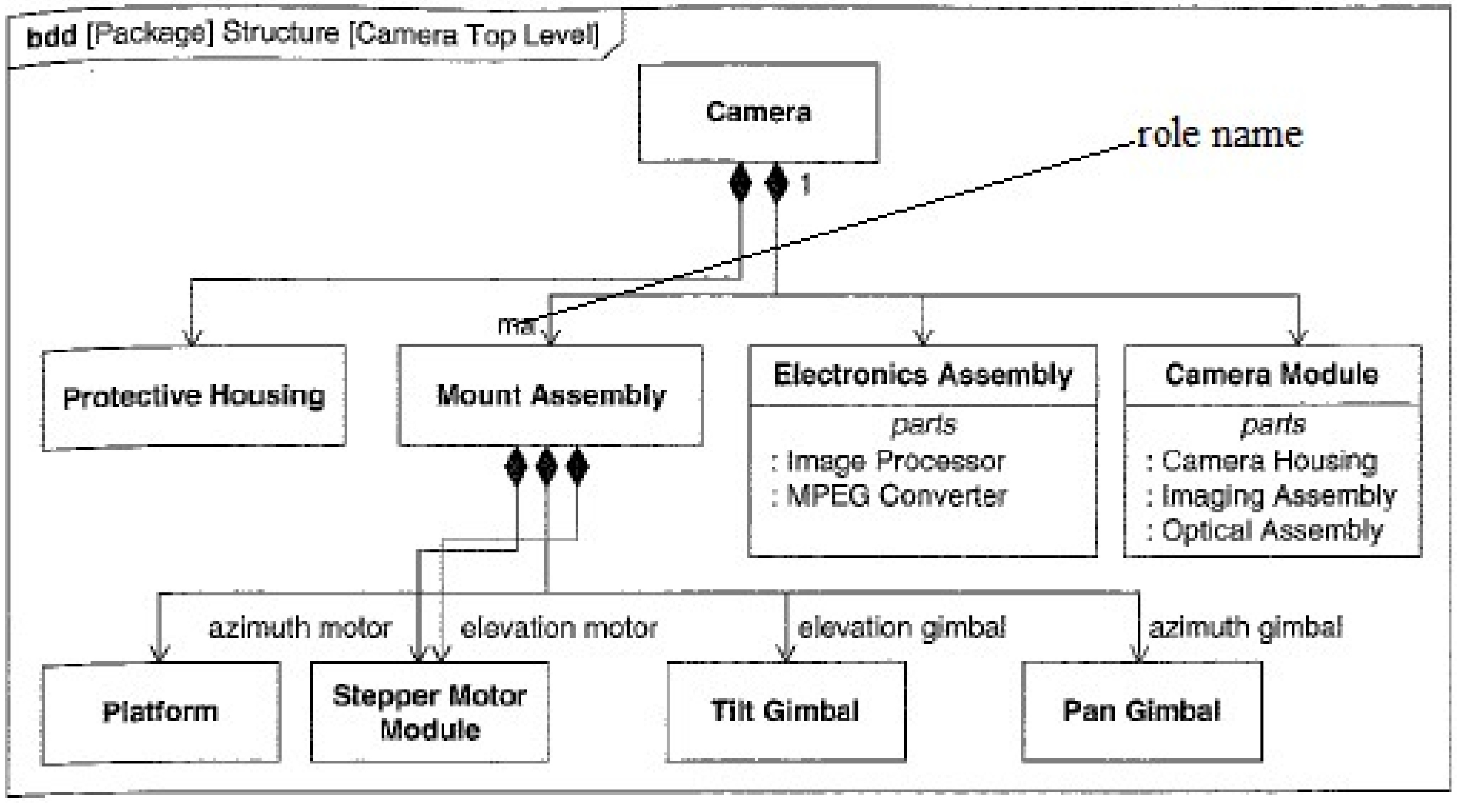
- Describe composition relationships between blocks (whole-part relationships)
- Are always **typed** by a block
- An instance of a whole may include multiple instances of a part property

Example in BDD



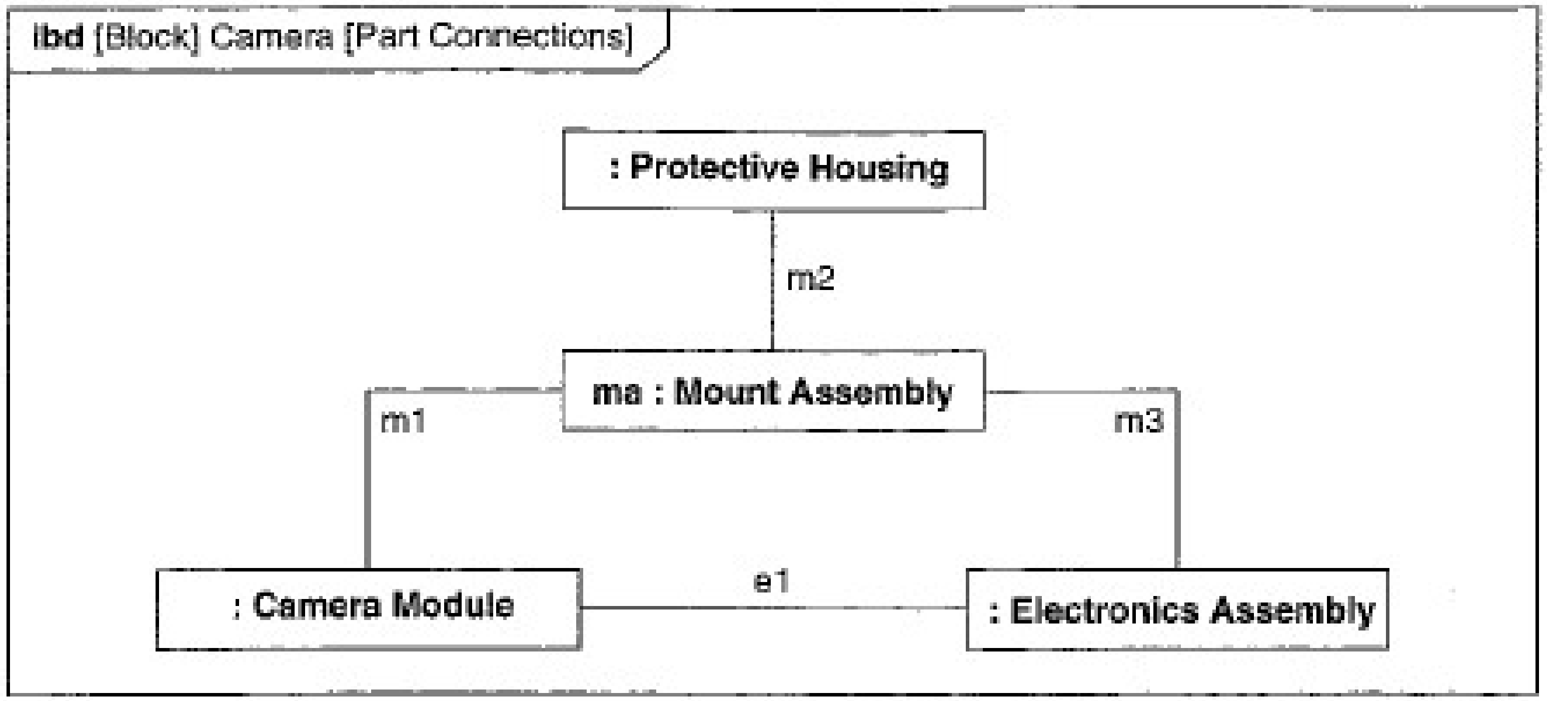
Composite Associations

- While a part property does state the number of instances that can be included in an instance of its whole, it does not state whether instances of the part must always exist as part of an instance of some whole.



Connectors

- Signature: connector name: association name



References Properties

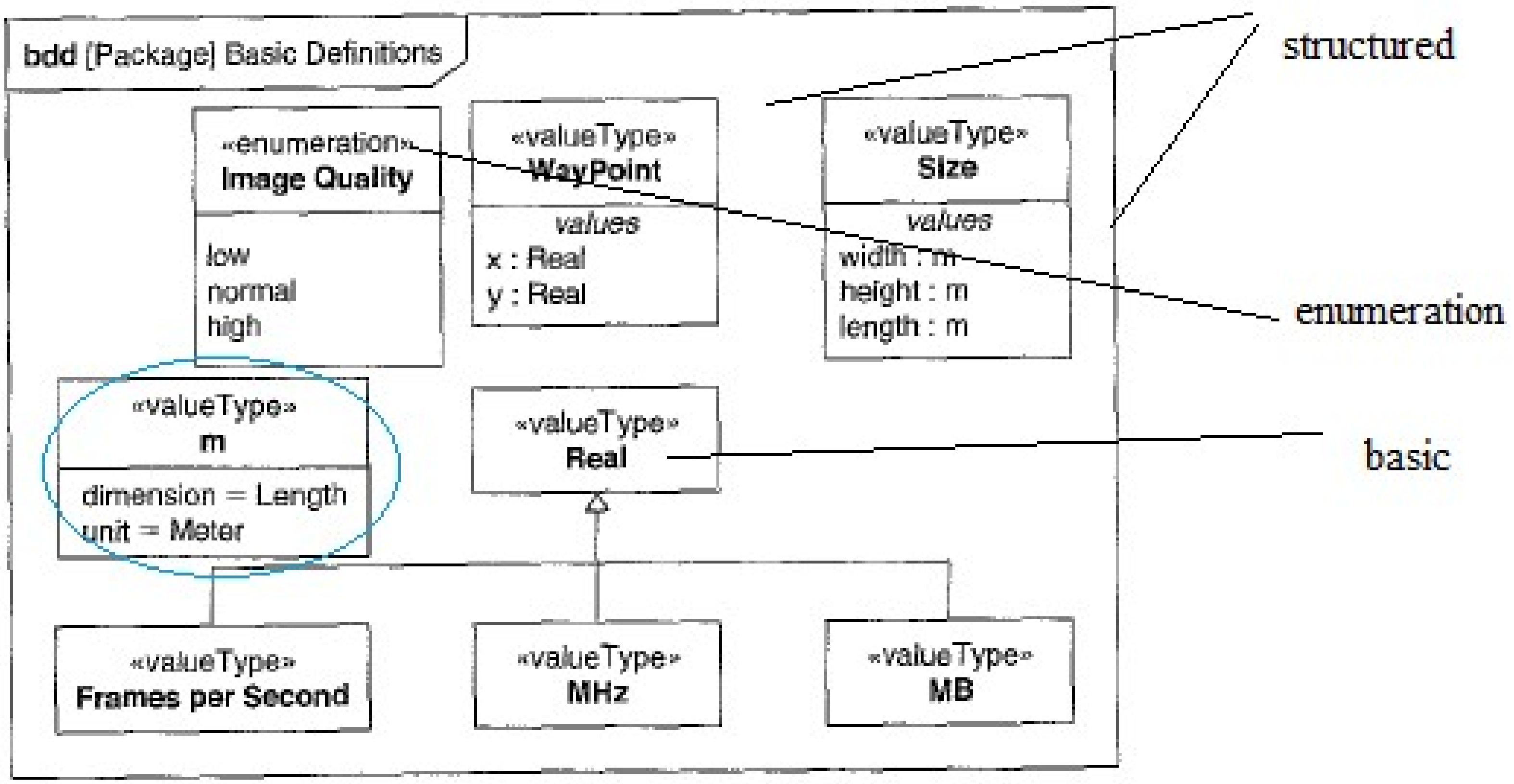
- Used to get additional views besides the primary system whole-part hierarchy

- Used to model stored value

Value Properties

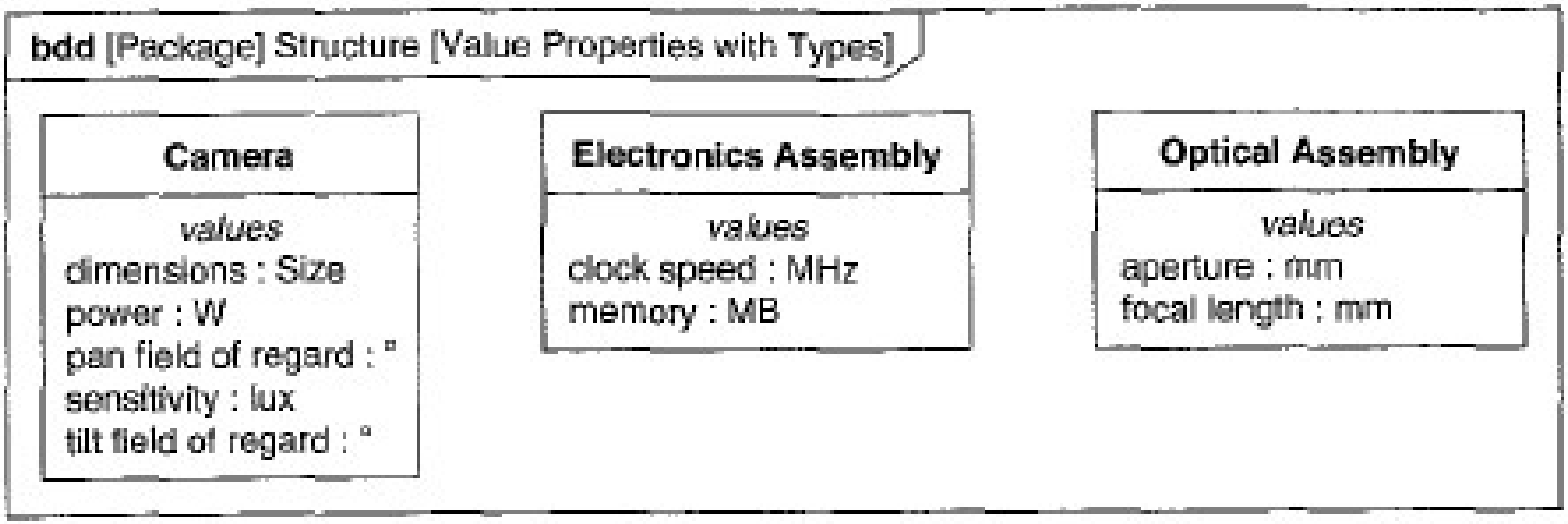
- Based on a value type that:
  - specifies the range of valid values
  - is characterized by
    - Unit (optional)
    - Dimension
- SysML provides fundamental types, such as Integer, String, Boolean, Real, Complex, enumeration.

Defining Value Types



Example

- Signature: property name: type name

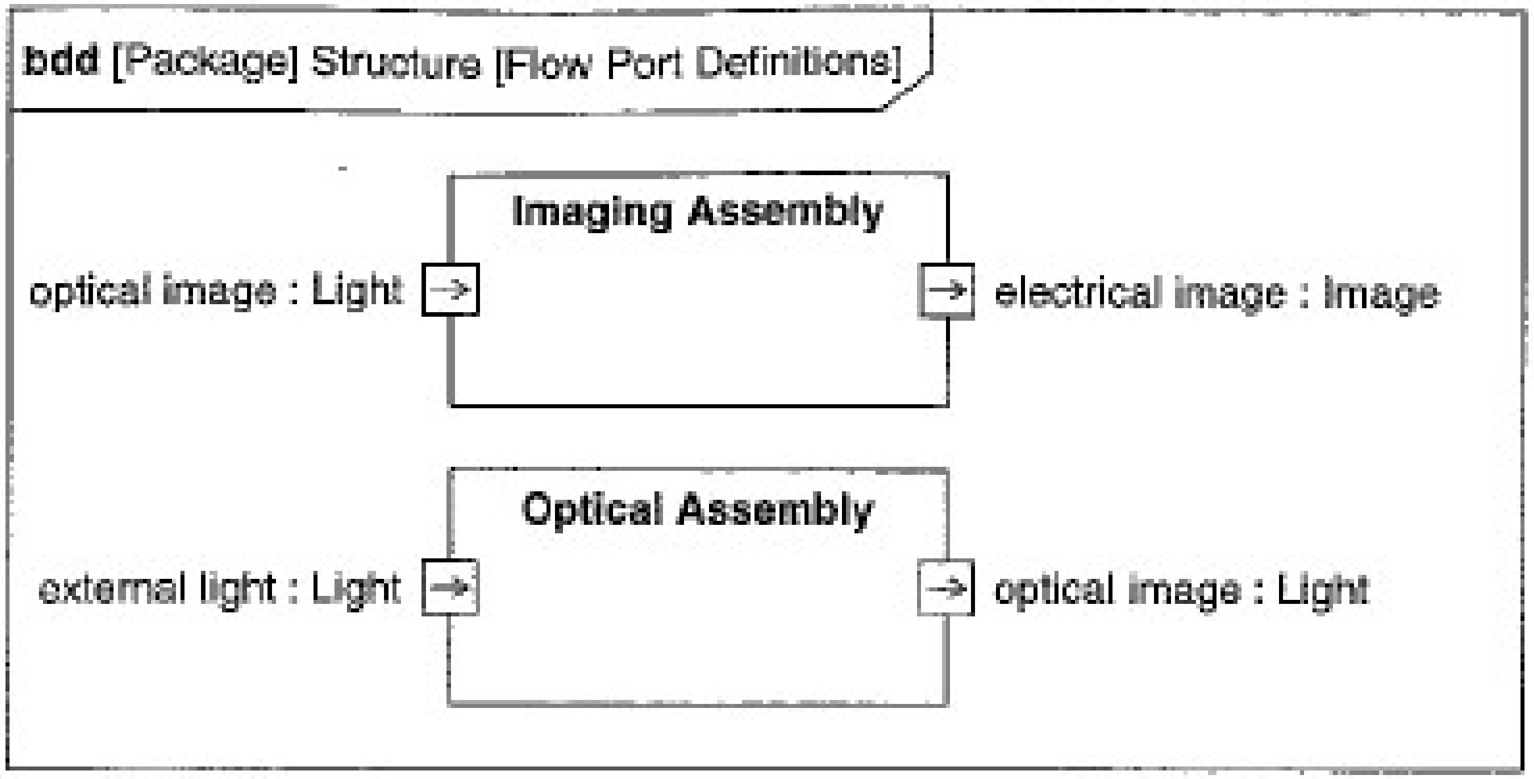


Ports

- Represent interaction points on the boundary of a block and on the boundary of any part typed by that block
- Enable the behavior of blocks/parts to be accessed.
- Are of 2 types:
  - Flow ports: what can flow in/out of the block at the interaction point (e.g. flow of information in electronic systems; the information is an item)
  - Standard ports: services required/provided by the block at the interaction point (e.g. interfaces between software components)

Example

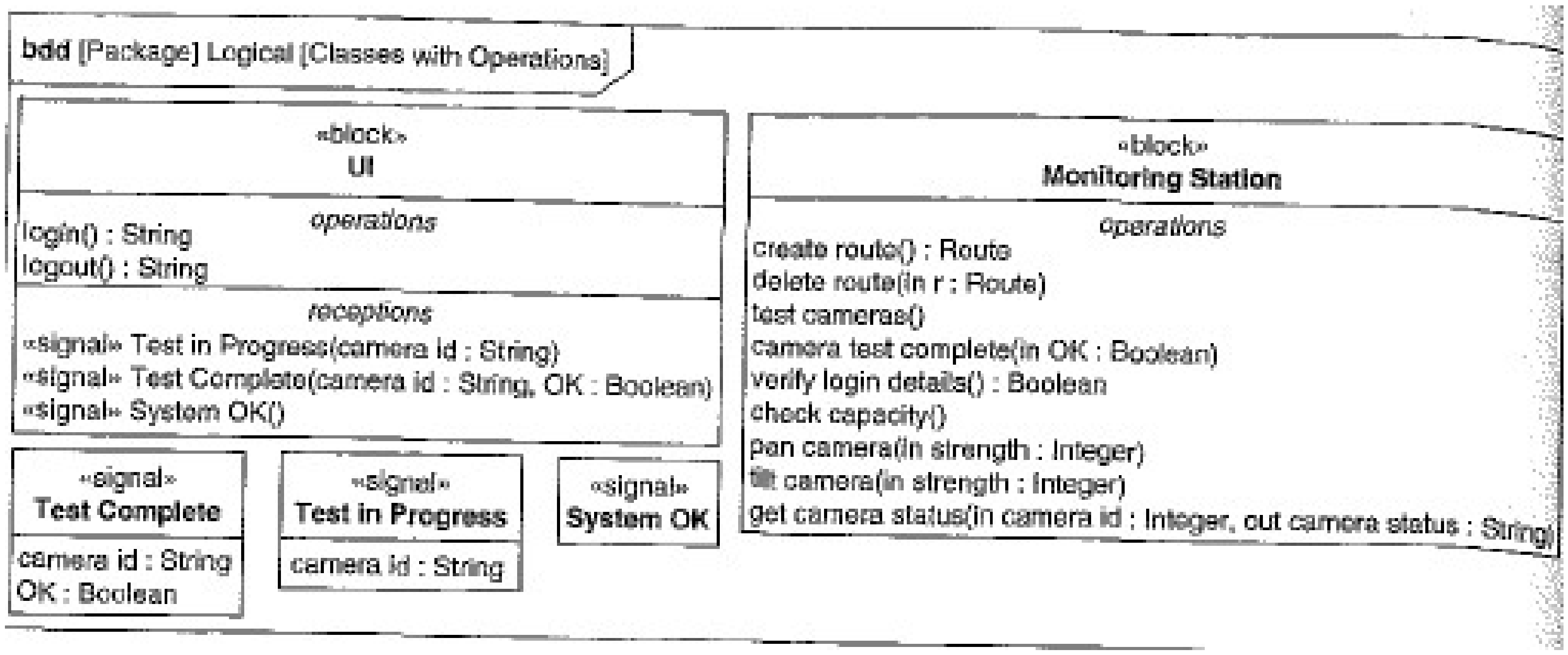
- Signatures:
  - port name: item name [multiplicity]
  - direction port name: item name [multiplicity]



Behavioral Features

- Describes requests a block can respond to
- Two types:
  - Receptions: represent asynchronous requests; each reception is associated with a signal that defines a message with a set of attributes that represent the content of the message
  - Operations: represents synchronous requests and define a set of parameters that describe the arguments passed with the request/passed out once the request is handled

Example


bdd [Package] Logical [Classes with Operations]
«block»
UI
operations
login() : String
logout() : String
receptions
«signal» Test in Progress(camera id : String)
«signal» Test Complete(camera id : String, OK : Boolean)
«signal» System OK()
«block»
Monitoring Station
operations
create route() : Route
delete route(in r : Route)
test cameras()
camera test complete(in OK : Boolean)
verify login details() : Boolean
check capacity()
pan camera(in strength : Integer)
tilt camera(in strength : Integer)
get camera status(in camera id : Integer, out camera status : String)
«signal»
Test Complete
camera id : String
OK : Boolean
«signal»
Test in Progress
camera id : String
«signal»
System OK

Hierarchical Relations related Features

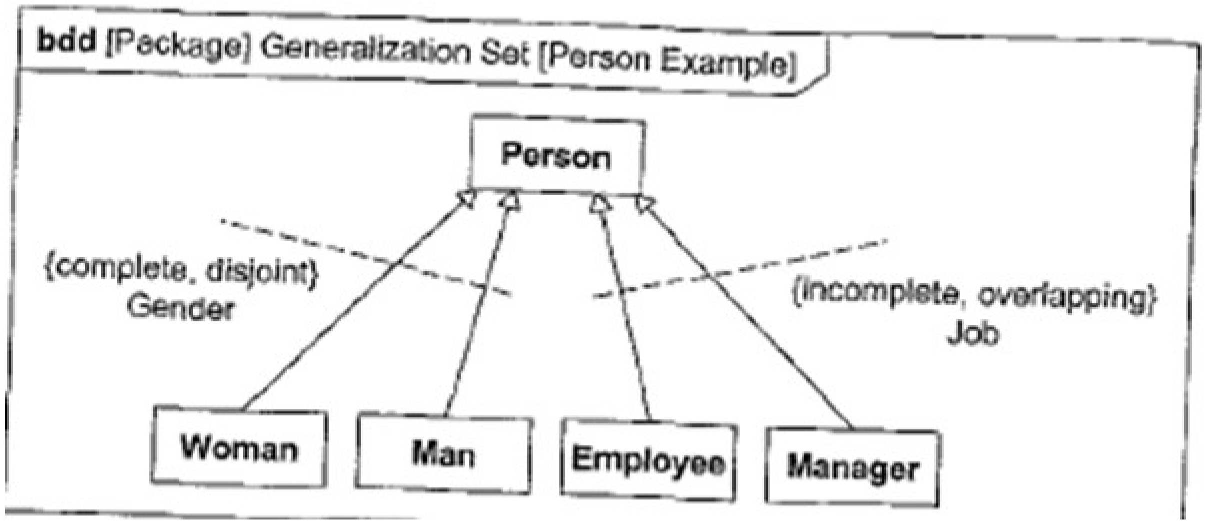
- All the elements in a BDD are classifiers: they can be classified/ organized in a hierarchy => generalization relationship, which facilitates reuse

- Sometimes a subclass may include features from multiple superclasses (multiple generalization/inheritance)

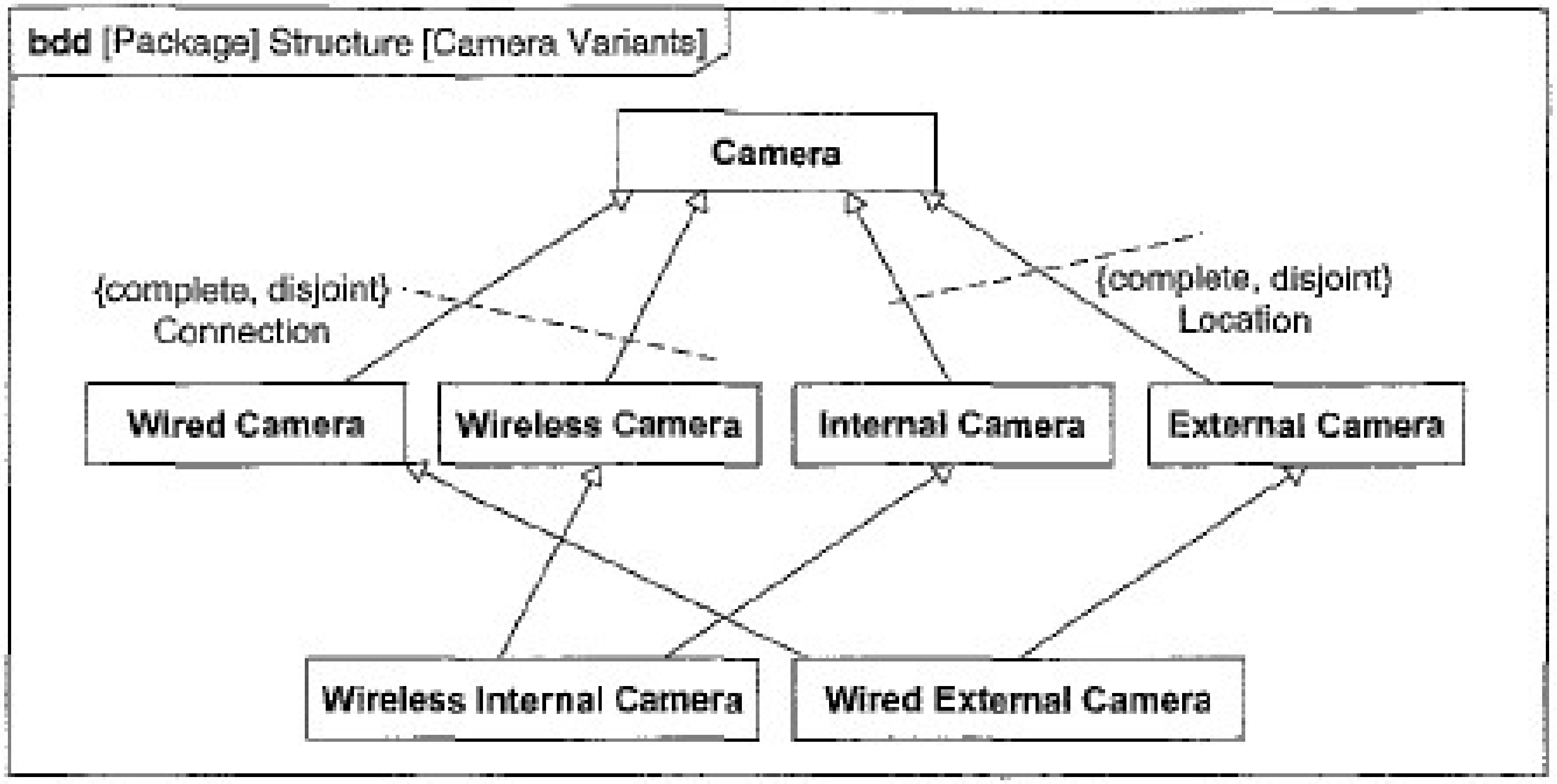
- The subclasses of a given class may also be organized into groupings based on how they can be used for further classification => **generalization sets**

Generalization Sets

- have 2 properties:
  - Coverage: with 2 values
    - complete
    - incomplete
  - Overlap
    - disjoint
    - overlapping


bdd [Package] Generalization Set [Person Example]
Person
{complete, disjoint}
Gender
{incomplete, overlapping}
Job
Woman
Man
Employee
Manager

Example

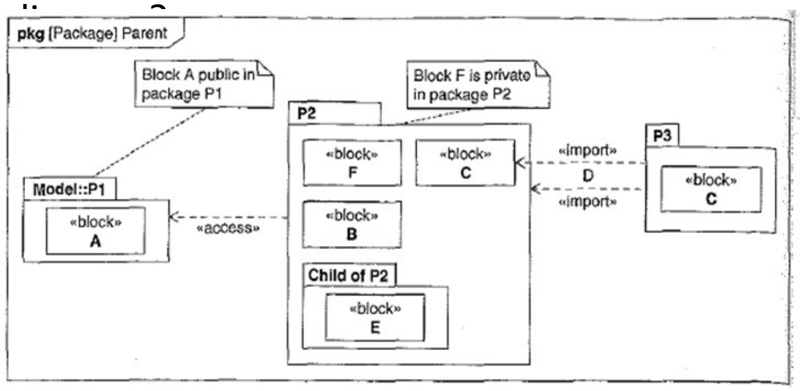

bdd [Package] Structure [Camera Variants]
Camera
{complete, disjoint}
Connection
{complete, disjoint}
Location
Wired Camera
Wireless Camera
Internal Camera
External Camera
Wireless Internal Camera
Wired External Camera

Exam Qs

- What diagrams are not UML diagrams:
  - Requirement diagram (check)
  - Package diagram
  - Parametric diagram (check)
  - Use case diagram
- A package in SysML may contain the following model elements:
  - blocks, use cases, activities
  - blocks, use cases, activities, packages (check)
  - blocks, activities
  - blocks, activities, packages

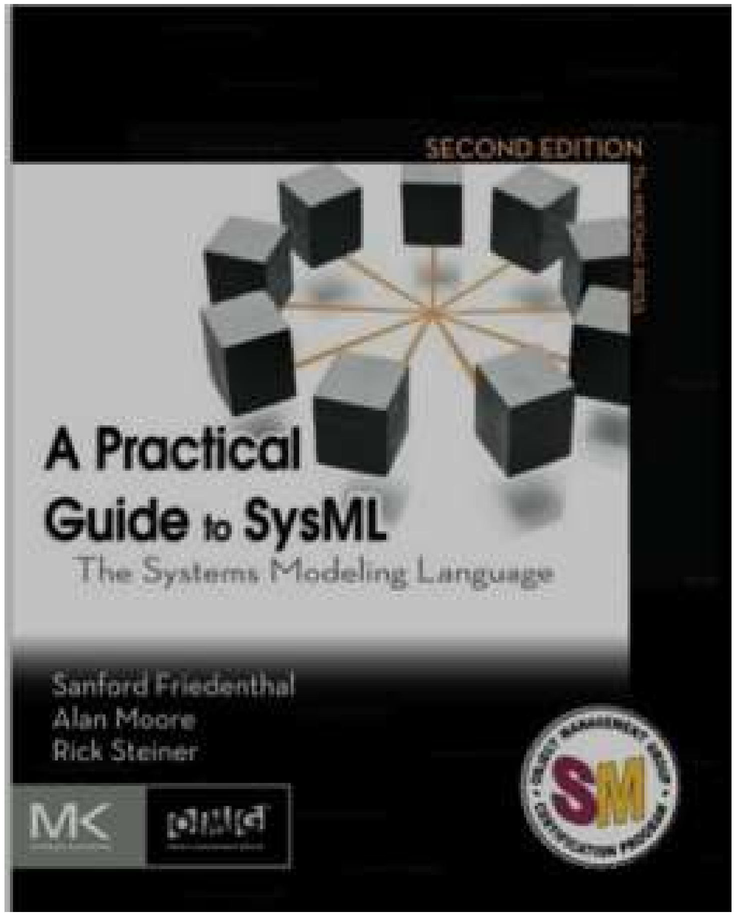
- Given the following diagram, what blocks will contain package P3?
  - B, C, D, F, Child of P2::E, Model::P1::A
  - B, C, D, P2::F, Child of P2::E, Model::P1::A (check)
  - B, C, P2::F, Child of P2::E, Model::P1::A
  - B, C, F, Child of P2::E, Model::P1::A

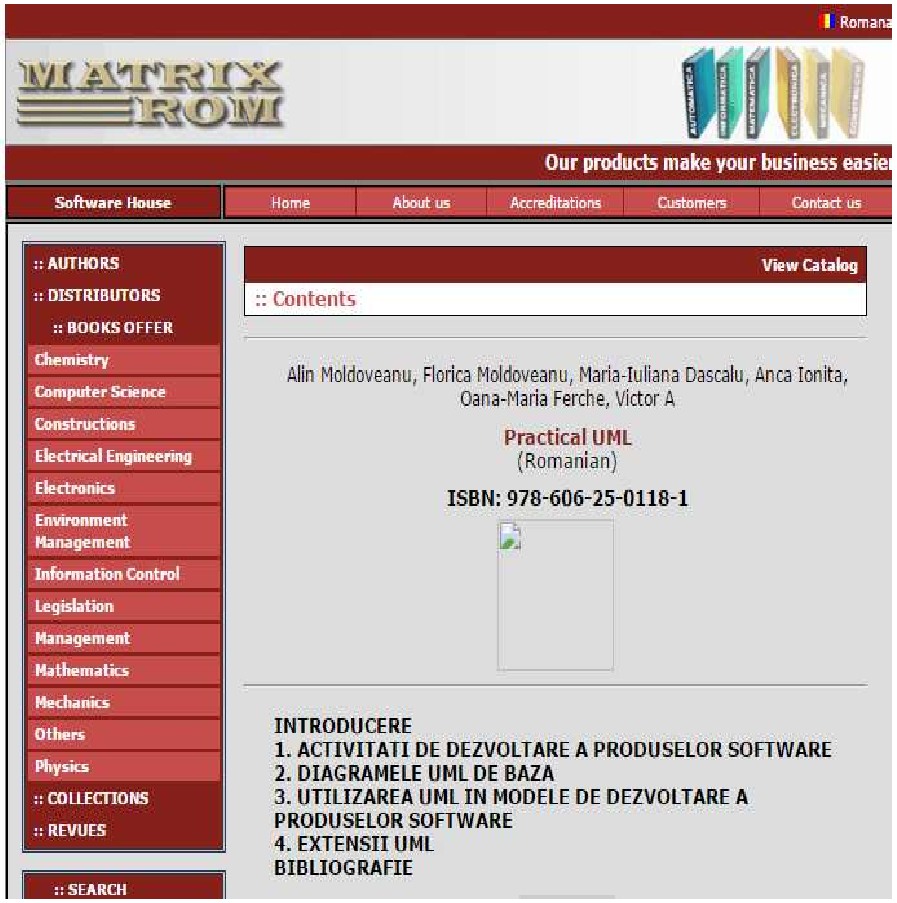
- What does the keyword “access” means in package
  - Private import of elements (check)
  - Protected import of elements
  - Public import of elements
  - This keyword doesn’t exist



- What is a view in SysML?
  - A package (check)
  - A model
  - A stereotype
  - Does not exist
- What is the modular unit of structure in SysML?
  - The class
  - The block (check)
  - The model
  - The package
- What are the behaviour features of the blocks?
  - Operations (check)
  - References
  - Receptions (check)
  - Flow ports
- What does the dashed line means in a block definition diagram?
  - A generalization set (check)
  - An inheritance
  - An association
  - Nothing

References





Alin Moldoveanu, Florica Moldoveanu, Maria-Iuliana Dascalu, Anca Ionita, Oana-Maria Ferche, Victor A

Practical UML
(Romanian)

ISBN: 978-606-25-0118-1

INTRODUCERE
1. ACTIVITATI DE DEZVOLTARE A PRODUSELOR SOFTWARE
2. DIAGRAMELE UML DE BAZA
3. UTILIZAREA UML IN MODELE DE DEZVOLTARE A PRODUSELOR SOFTWARE
4. EXTENSII UML
BIBLIOGRAFIE