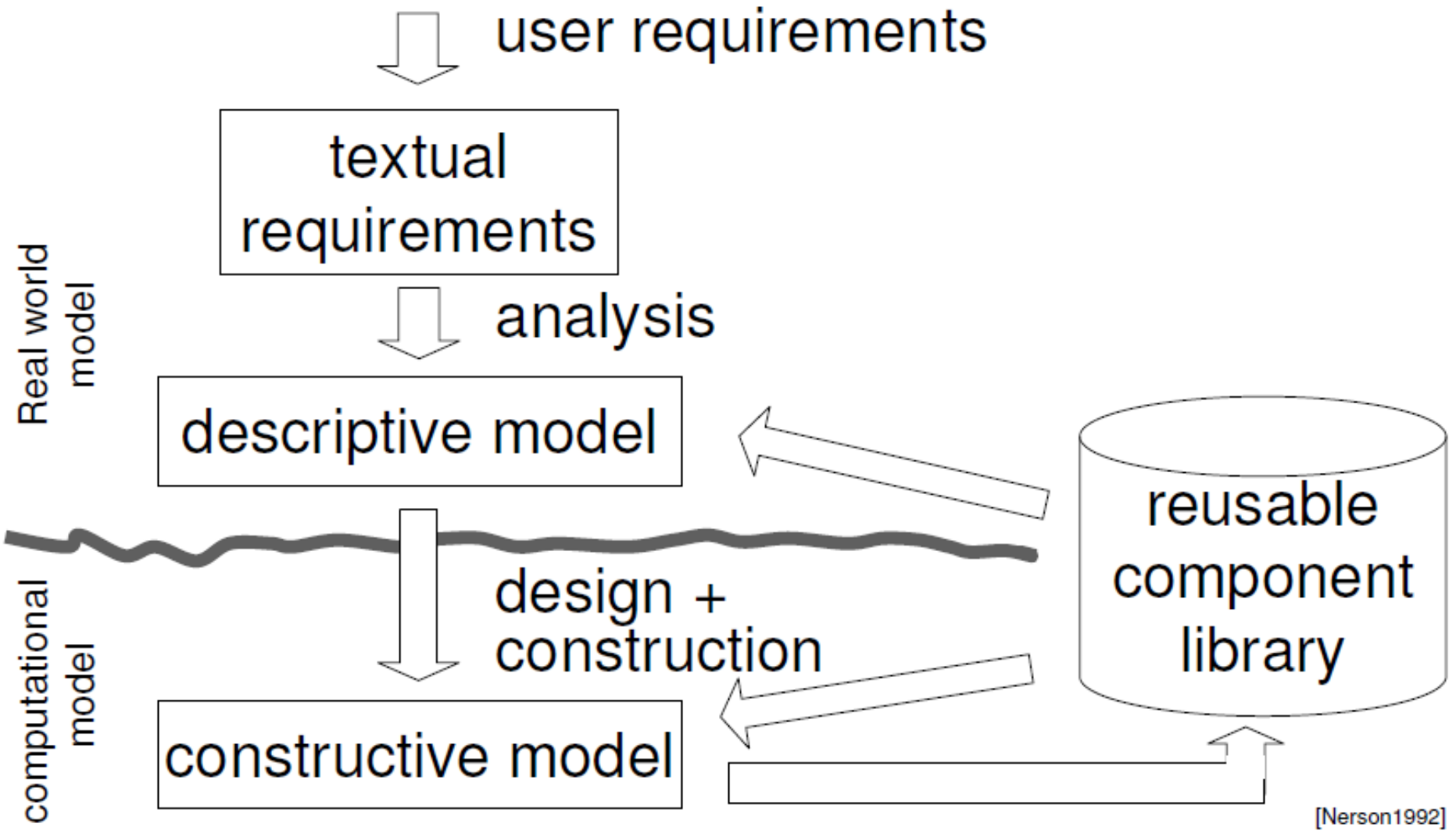
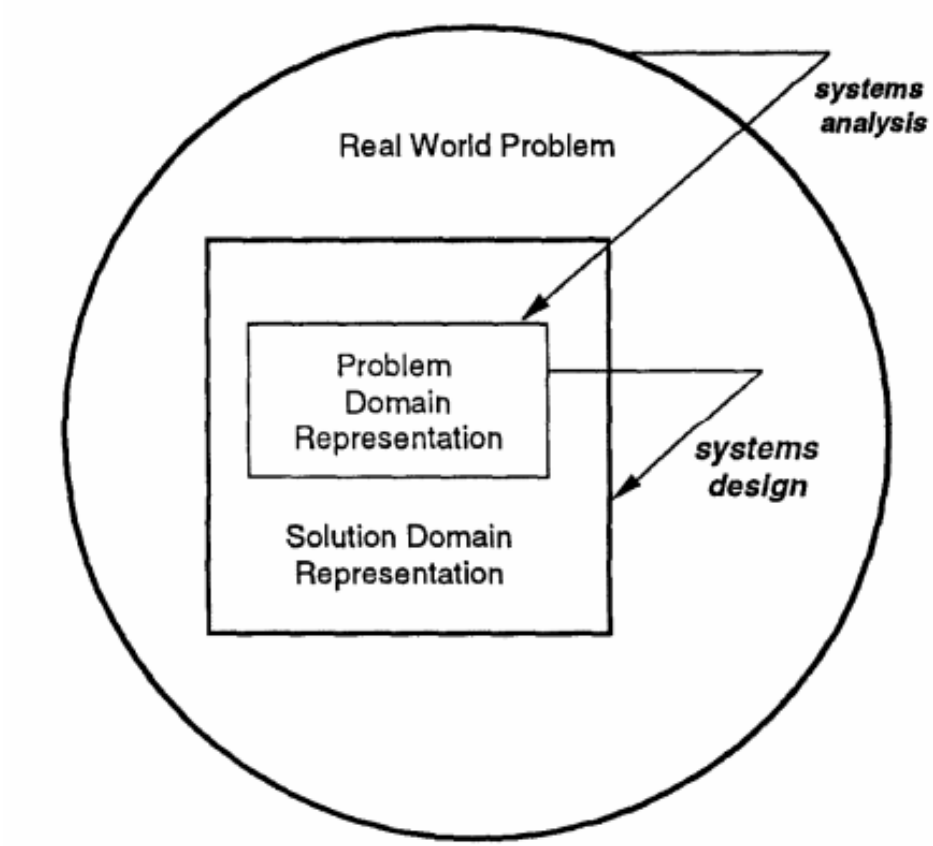
Stefan Henkler E-Mail: stefan.henkler@hshl.de


user requirements
textual requirements
analysis
descriptive model
Real world model
reusable component library
design + construction
computational model
constructive model
[Nerson1992]



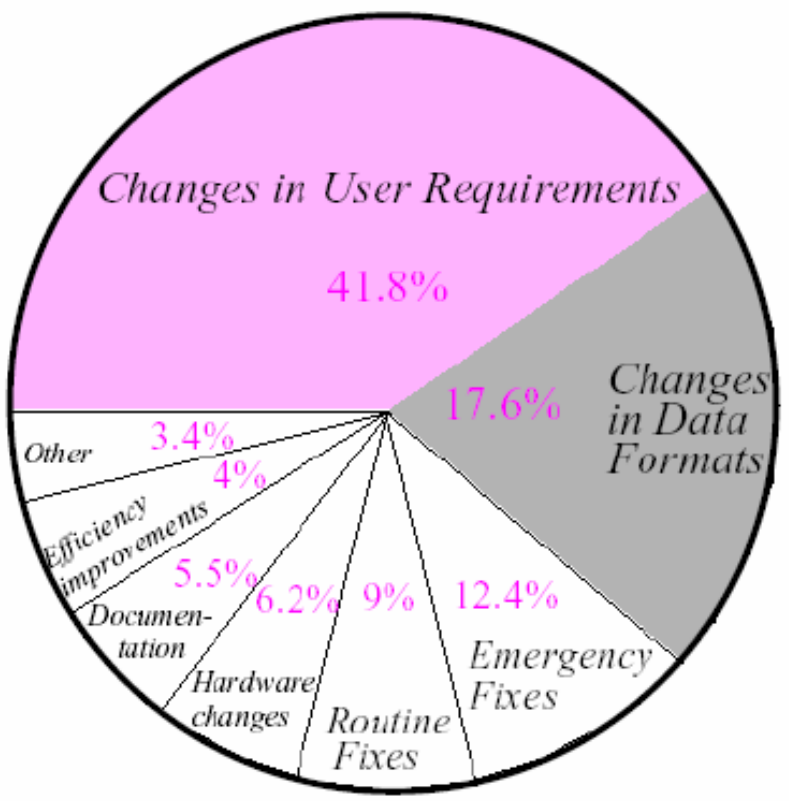
[Monarchi&Puhr1992]

- **Analysis** models the problem in the **problem domain** by identifying and specifying a relevant subset of the real world according to system requirements.
- **Design** models the solution in the **solution domain**, which often includes most elements of the problem plus additional solution specific elements.

- **Design** [IEEE-Std-610.12-1990][Taylor1959]:
  - (1) The process of defining the **architecture**, **components**, **interfaces**, and other characteristics of a system or component.
  - (2) The result of the process in (1). The process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization.
- **Design description** [IEEE-Std-610.12-1990]:
  - A document that describes the design of a system or component. Typical contents include system or component **architecture**, **control logic**, **data structures**, **input/output-formats**, **interface descriptions**, and **algorithms.**

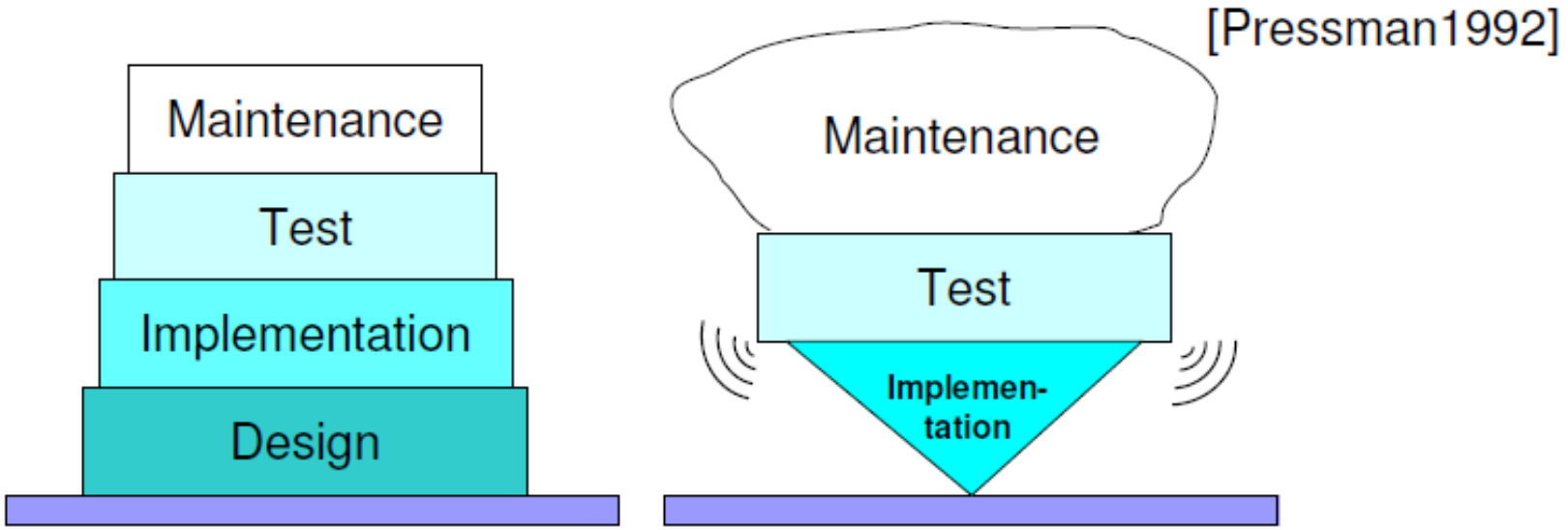
HOCHSCHULE HAMM-LIPPSTADT

- **First Law of Software Evolution**
  - A program that is used and that has an implementation of its specification reflects some reality, undergoes continual change or becomes progressively less useful.
- **First Law of System Engineering**
  - No matter where you are in the system life cycle, the system will change and the desire to change it will persist throughout the life cycle.



Breakdown of maintenance costs.
([Meyer1997] source [Lientz1980])

Software will be changed during maintenance



- Importance of software design: **quality**
  - Translate accurately requirements into product
  - Still stable systems for small changes
  - Otherwise difficult to test

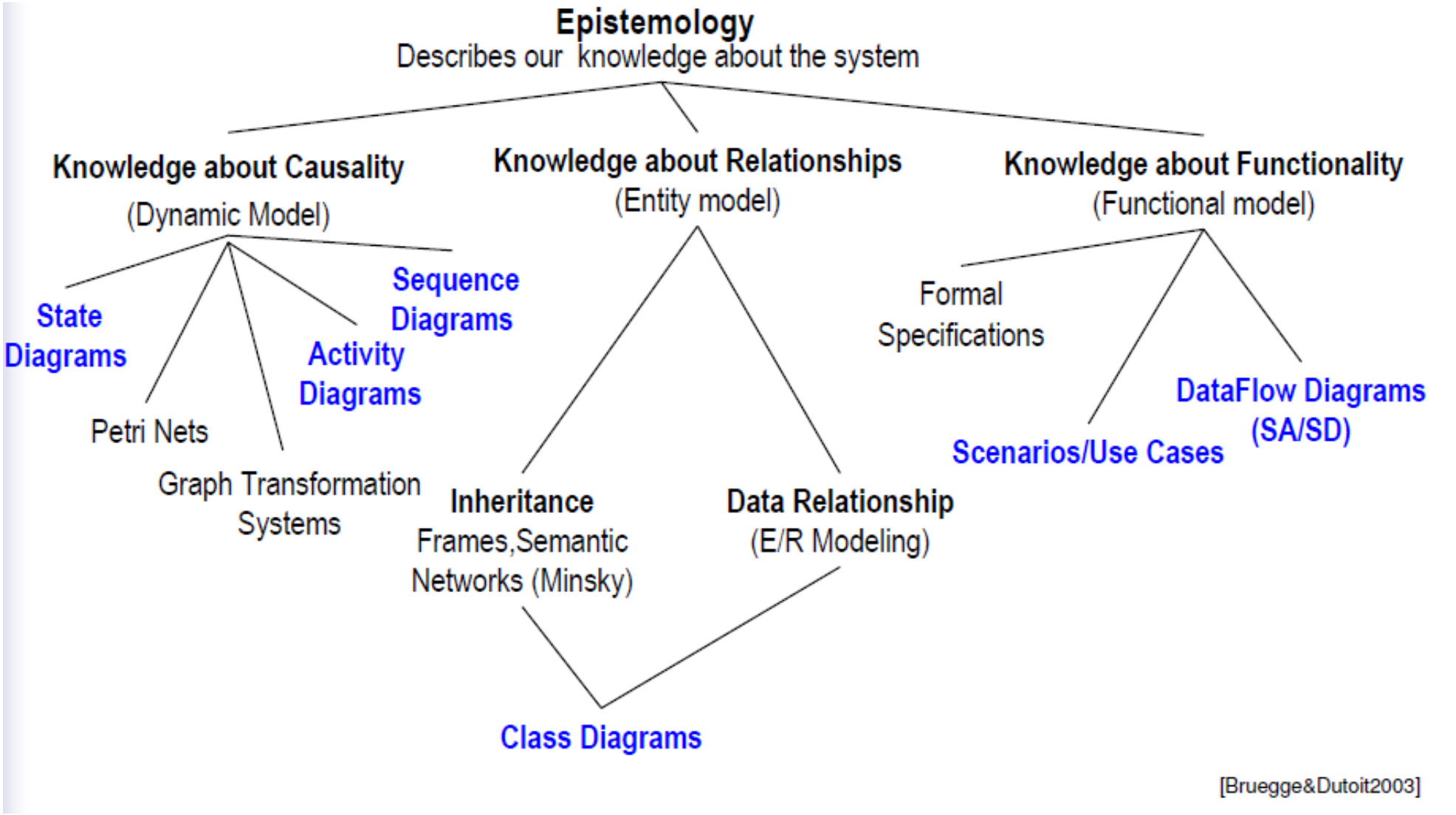
HOCHSCHULE HAMM-LIPPSTADT

1. Introduction
2. Methods
3. Analysis
4. Design
5. Advanced Design Concepts
6. Discussion & Summary
7. Bibliography

HOCHSCHULE HAMM-LIPPSTADT

Historic Trends:

- Structured programming
  - Dijkstra 1968: Goto statement considered harmful
  - Keywords: top-down, functional decomposition, stepwise refinement, divide-and-conquer, …
  - SA/SD: late 1970s
- Object-oriented programming
  - Simular 67, smalltalk, C++, Object C, Eiffel, Ada, Lisp
  - Liskov, Guttag, Shaw 1970s: Abstract data type
  - Parnas 1972: Information hiding
  - Keywords: objects, classes, reusable components
  - OOA/OOD: late 1980s

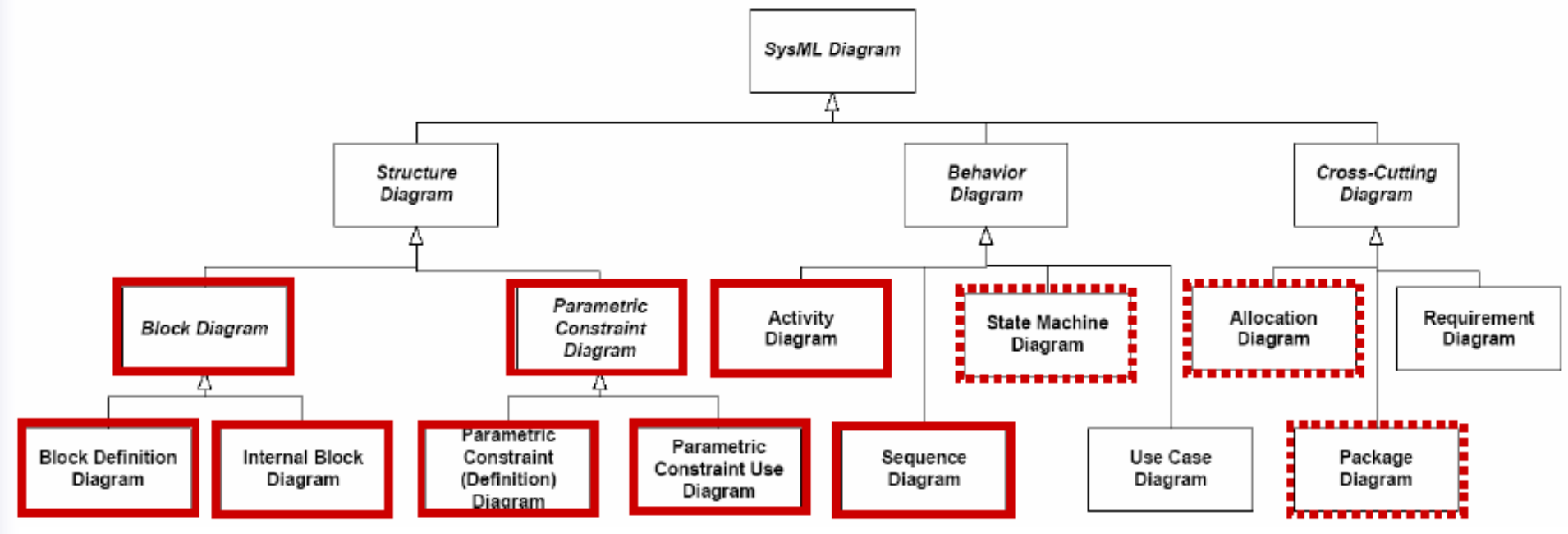


[Bruegge&Dutoit2003]

HOCHSCHULE HAMM-LIPPSTADT

1. Introduction
2. Methods
3. Analysis
4. Design
5. Advanced Design Concepts
6. Discussion & Summary
7. Bibliography

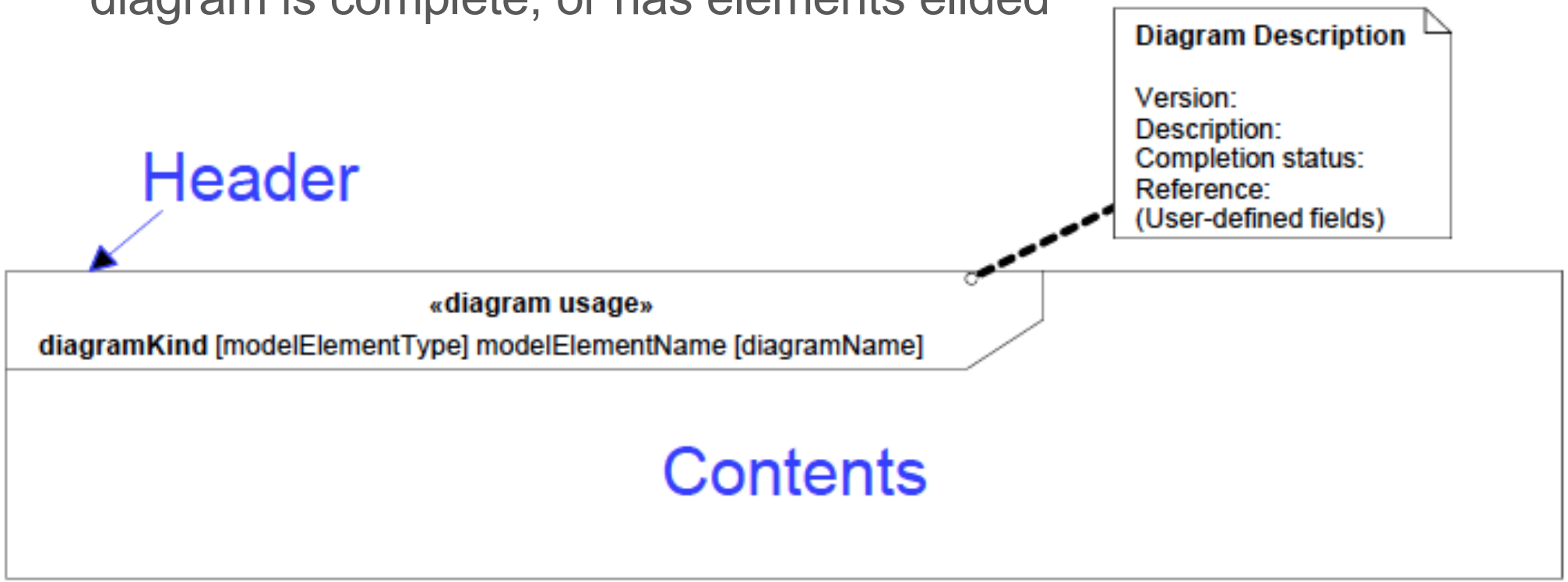
- Analysis should **model** aspects of the **real world** which are **relevant for the application** (objectives, requirements, application domain knowledge, requirements on the environment and requirements on the computer system).
- The model therefore should only describe the **required** or **existing structure** and **behavior of the application**.
- The **analysis model** is the **base for communication** between analysts, experts in the application domain and end users of the system.
- Main stake holders: end user, customer, analyst



- (1) Block Diagrams
- (2) Parametric Constraint Diagram
- (3) Activity Diagrams
- (4) Sequence Diagrams

SysML Diagram Frames

- Each SysML diagram represents a model element [SysMLTutorial09]
- Each SysML Diagram must have a Diagram Frame
- Diagram context is indicated in the header:Diagram kind (act, bdd, ibd, sd, etc.)
  - Model element type (package, block, activity, etc.)
  - Model element name
  - User defined diagram name or view name
- A separate diagram description block is used to indicate if the diagram is complete, or has elements elided

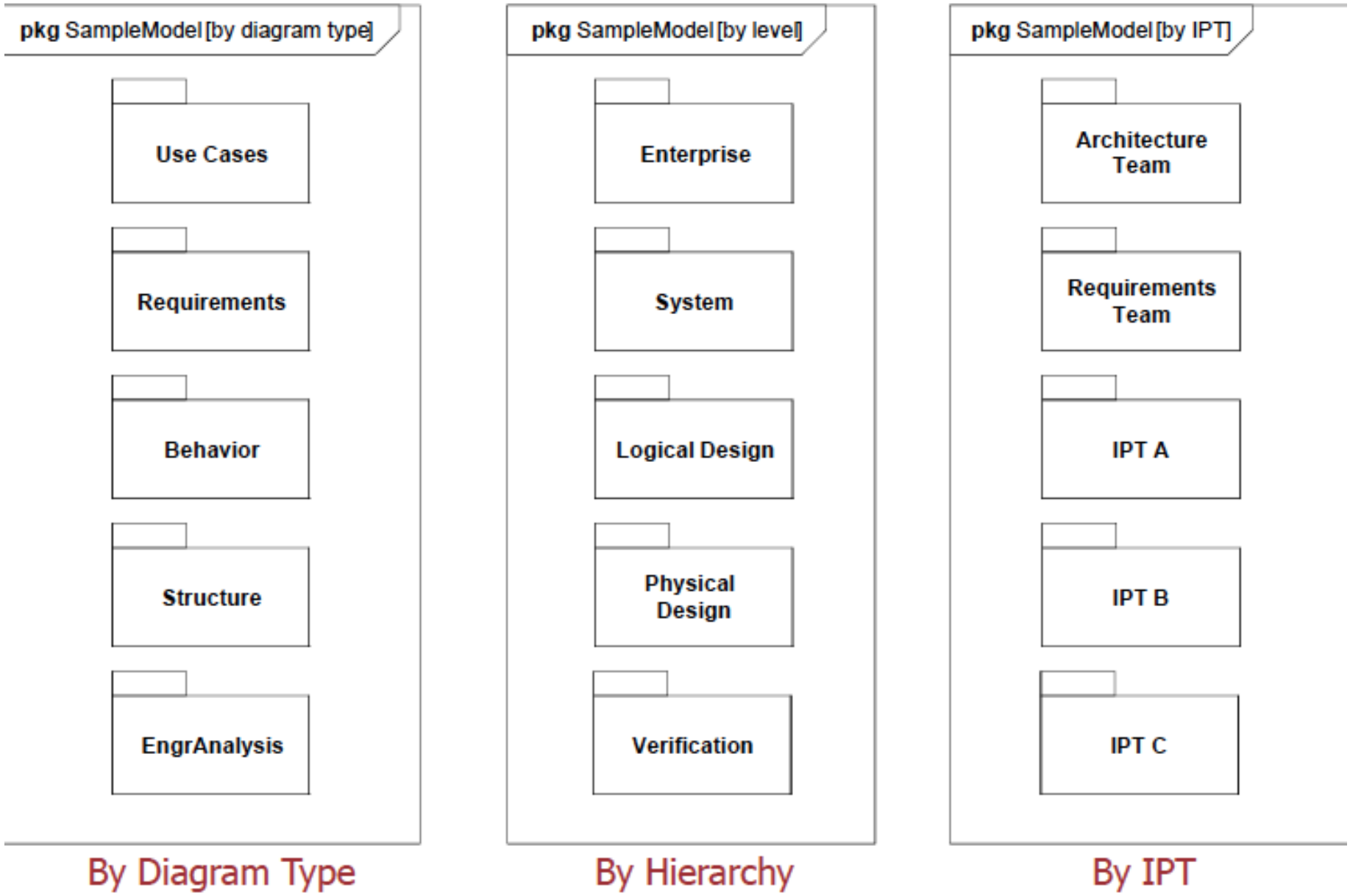


HOCHSCHULE HAMM-LIPPSTADT

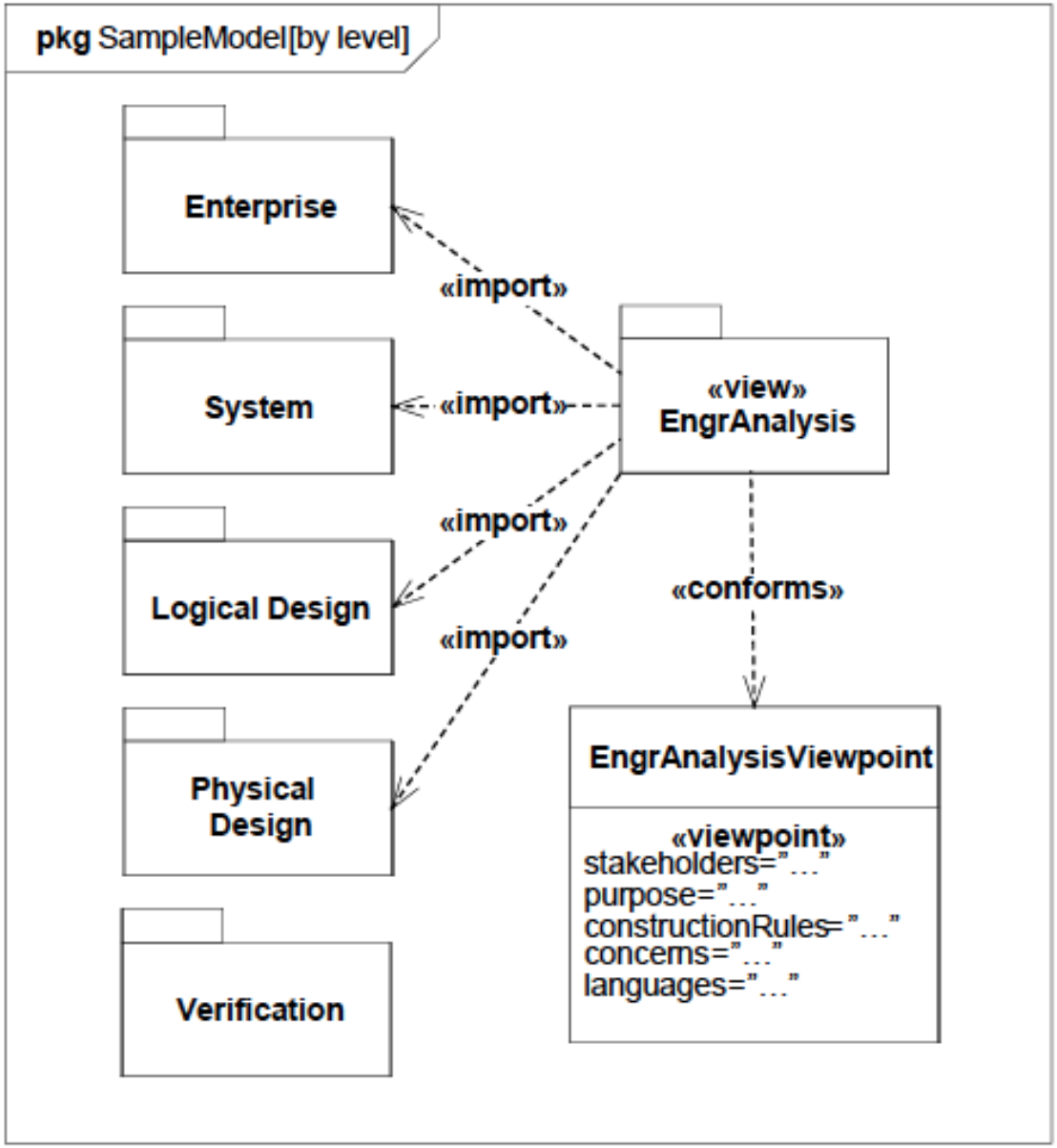
- Package diagram is used to organize the model [SysMLTutorial09]
  - Groups model elements into a name space
  - Often represented in tool browser
  - Supports model configuration management (check-in/out)
- Model can be organized in multiple ways
  - By System hierarchy (e.g., enterprise, system, component)
  - By diagram kind (e.g., requirements, use cases, behavior)
  - Use viewpoints to augment model organization
- Import relationship reduces need for fully qualified name (package1::class1)

Before we start...

Package Diagram Example



Package Diagram - Views



- Viewpoint represents the stakeholder perspective
- View conforms to a particular viewpoint
  - Imports model elements from multiple packages
- View and Viewpoint consistent with IEEE 1471 definitions

What can/should be Analyzed?

- (1) Block Diagrams
  - Given structures and interfaces
- (2) Parametric Constraint Diagram
  - Dependencies/Constraints between given elements
- (3) Activity Diagrams
  - Scenarios describing required activities
- (4) Sequence Diagrams
  - Required/Likely interaction scenarios
- (5) State Machine Diagram
  - Complete state-dependent reactive behavior of given elements

(1) Block Diagram

HOCHSCHULE HAMM-LIPPSTADT

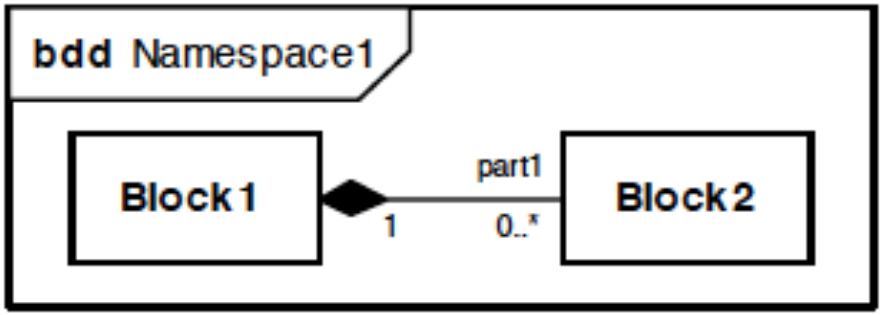
- A **Block** is a modular unit of system that **encapsulates** its contents, which include attributes, operations and constraints.
  - Blocks can be connected to other Blocks to form **composite structures**, and can be decomposed into **parts** to expose internal structures.
- Each **part** is specified by a block with its own properties, ports, and internal structure
  - a uniform set of elements is used to represent multiple levels of a system hierarchy. (**blocks are used for definitions, and parts are used for applications**)
- **Application:**
  - The SysML **block** model can be used throughout **all phases of system specification and design**, and can be applied to many different kinds of systems.
  - These **systems may be logical or physical, and may include software, hardware or human organizations**.
- A block is an unified concept for describing the structure of:
  - System, Hardware, Software, Data, Procedure, Facility, Person, …

Block Definition Diagram

- Used to represent block definitions
- follows the graphical conventions of a **UML class diagram.**
- Showing blocks, their properties and their relationships.

Block

- A SysML Block defines a **collection** of **features** to describe a system or other element of interest.
- SysML blocks are **based** on UML **classes** - extended by UML composite structures
- **Properties** are the **basic structural characteristics** of blocks.
  - Properties may be of several types: Value properties, part properties, reference properties



«block»
{encapsulated}
Block1

constraints
{ x > y }

operations
operation1(p1: Type1): Type2

parts
property1: Block2

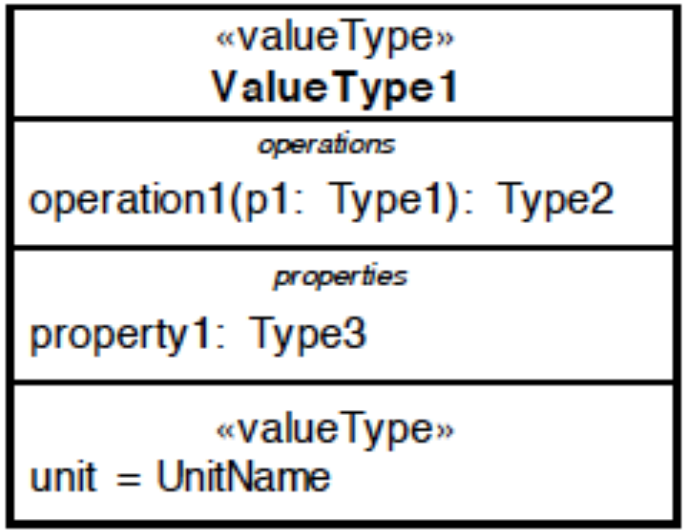
references
property2: Block3 [0..*] {ordered}

values
property3: Integer = 99 {readOnly}
property4: Real = 10.0

properties
property5: Type1

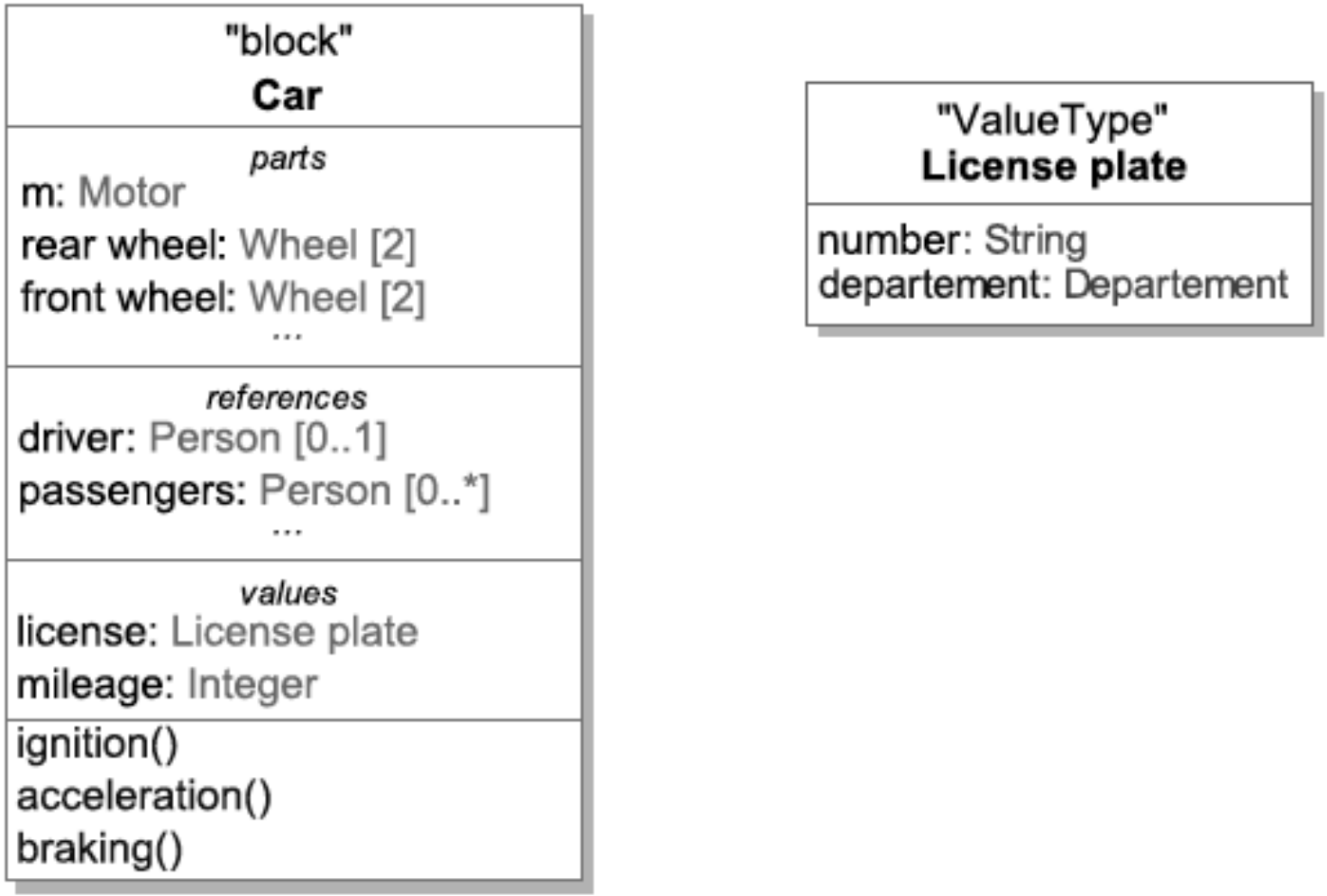
20 Block Property

- **Value** properties describe quantifiable characteristics in terms of value types (**range of values**, dimensions and optional units)
  - To define the types for value properties, SysML offers value types.
  - A SysML ValueType defines values that may be used within a model.
  - SysML value types are based on UML data types
- **Part** properties describe the decomposition hierarchy of the block in terms of other blocks.
  - A Block **is used** in the **context** of the enclosing block (composite) (e.g. left-front:wheel)
- **Reference** properties describe relations / association or simple aggregation with other blocks
  - Part **is not owned** by the enclosing block (no composition) (e.g. components are aggregated into logical subsystem)

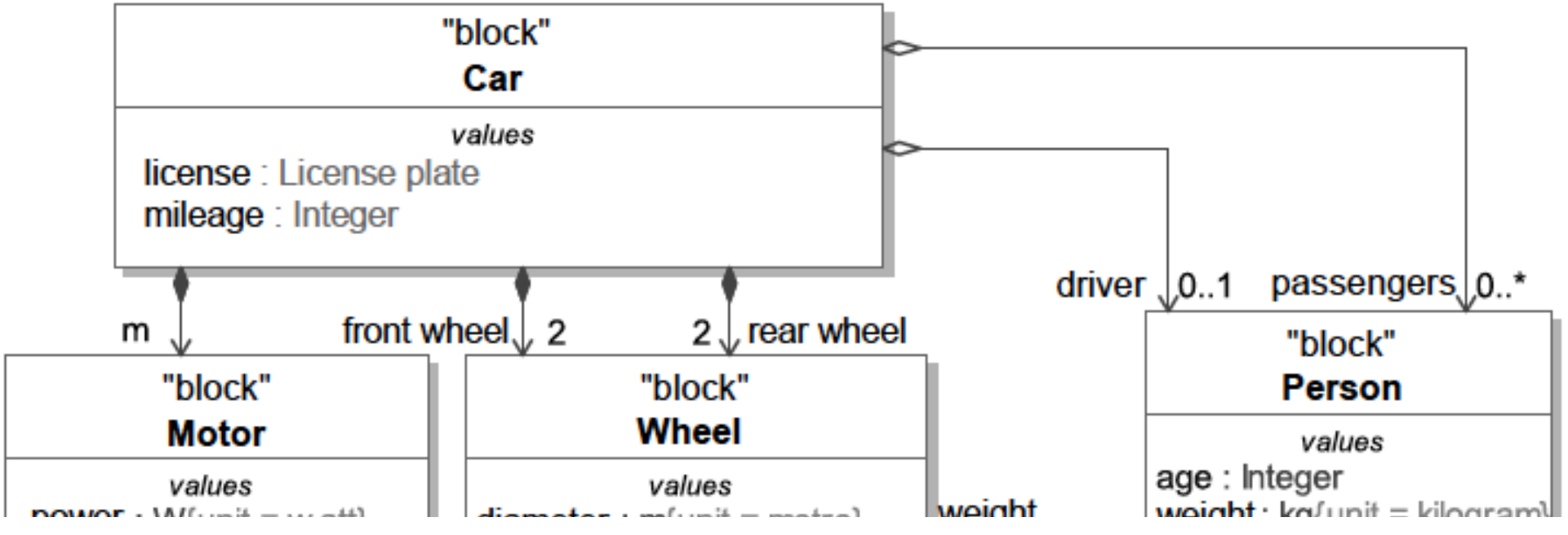


Example

- the value type “License Plate” makes it possible to create a type that can be reused in the block “Car”



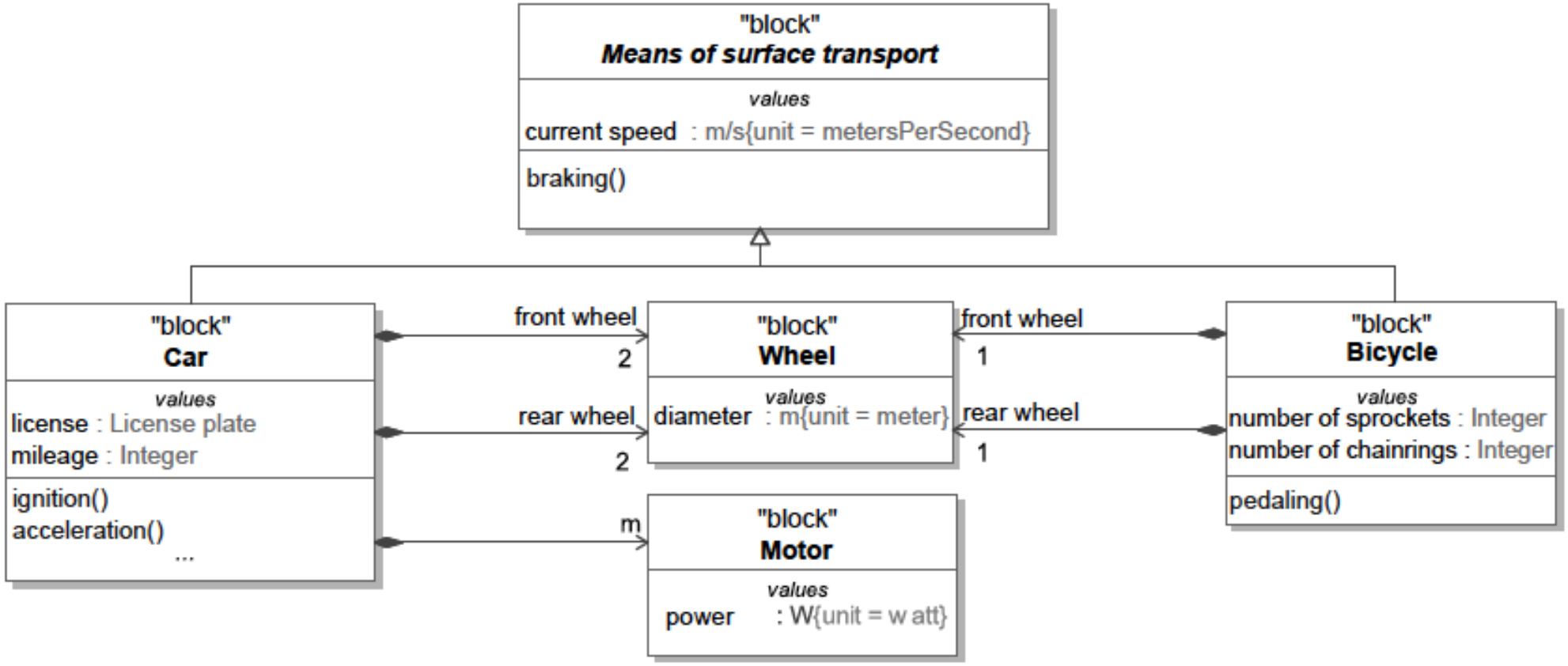
- Two containment types: Aggregation, Composition (as in UML)
- Static, enduring relation between two blocks
- Multiplicity should appear at each of its two ends
  - Specifies, as an interval, the number of instances that can participate in a relation with an instance of the other block in the context of this association
- An unidirectional association has an arrow pointing toward the block that is being referred to



▸23 Relations between Blocks

Generalization

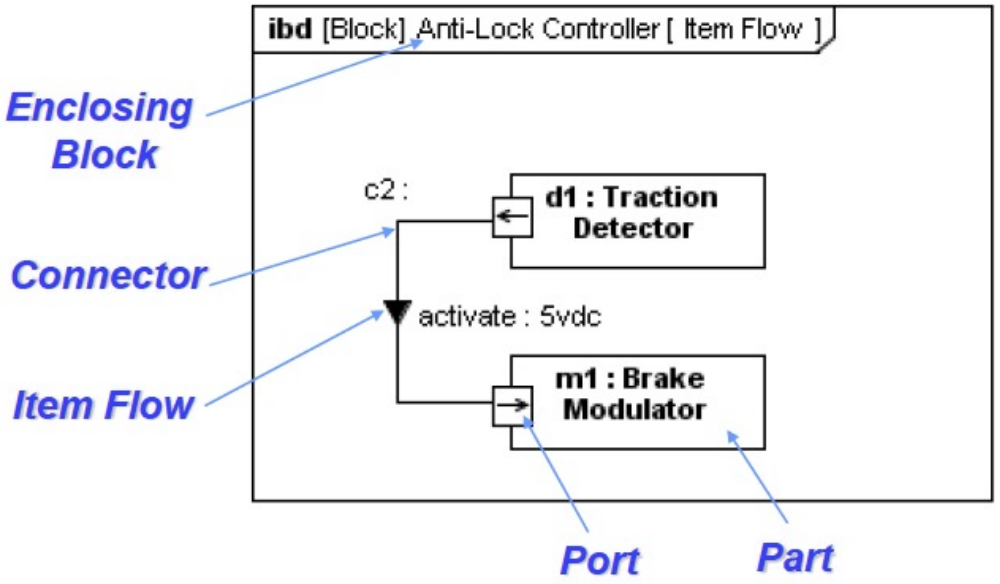
- Blocks can be organized in a classification hierarchy
- Intended to factorize properties that are common to several blocks (values, parts, etc) in a generalized block
- Specialized blocks “inherit” the properties of the generalized block and may have specific additional properties



24 Internal Block Diagram

Specify Interconnection of Parts

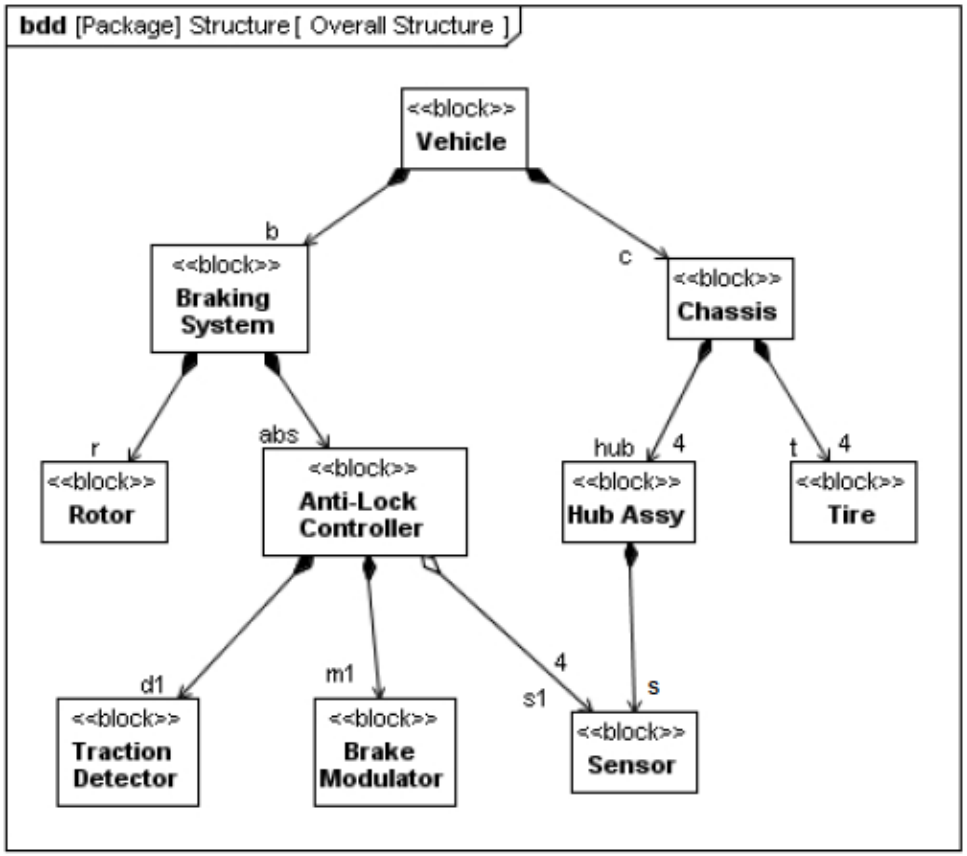
- The internal block diagram (ibd) describes the internal structure of a block in terms of parts, ports and connectors.
- Used to show the internal structure of a block and follows the graphical conventions of a **UML composite structure diagram** showing internal structure (**parts**, ports and connectors) of the subject block.
- note that we can represent several levels of decomposition in a single ibd.

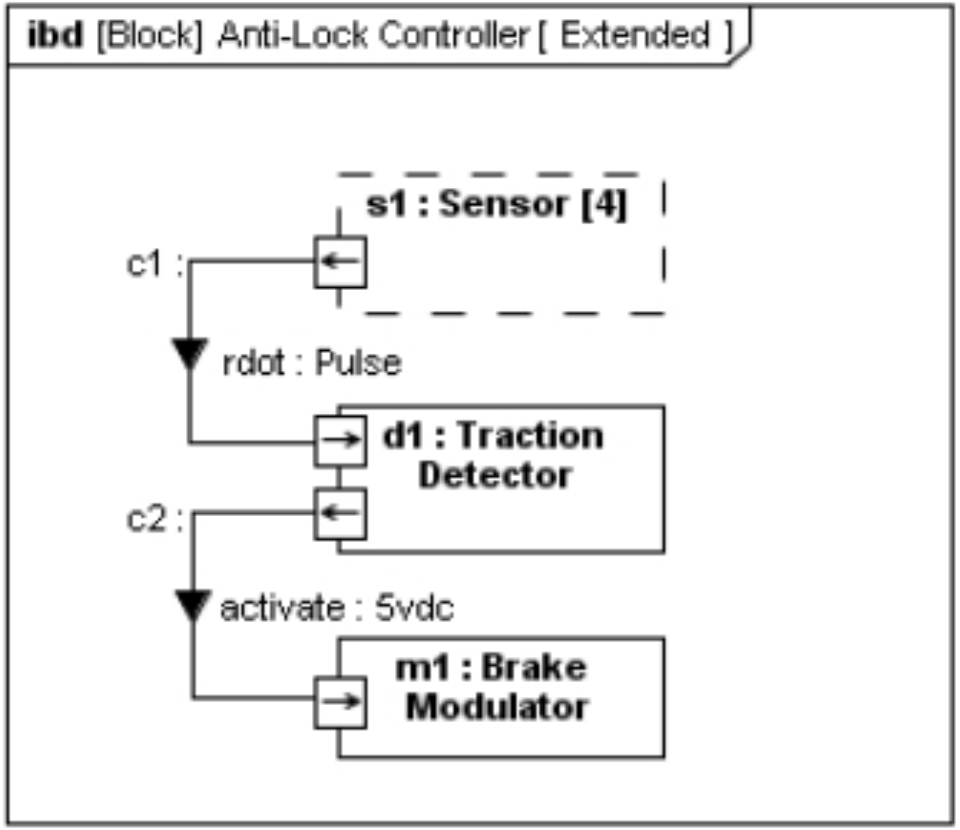


[SysMLTutorial09]

▸25 Internal Block Diagram

- A **composition** relation in a bdd can be represented with an ibd
- Each end of the composition relationship that exists in the bdd is **presented** as a **block** (known as a part) in the framework of the ibd
  - Part name is of the form: part_name: block_name [multiplicity]
- The multiplicity (1, by default) can also be represented in the upper right corner of the rectangle
- **Associations** and **aggregations** that are „outside“ the encompassing block are represented in a similar way to compositions, except that the line surrounding the block is **dashed**





- Based on UML Class from UML Composite Structure
  - Supports unique features (e.g., flow ports, value properties)

- **Block definition diagram** describes the relationship among blocks (e.g., composition, association, specialization)
- **Internal block diagram** describes the internal structure of a block in terms of its properties and connectors
- **Behavior** can be allocated to blocks

[SysMLTutorial09]

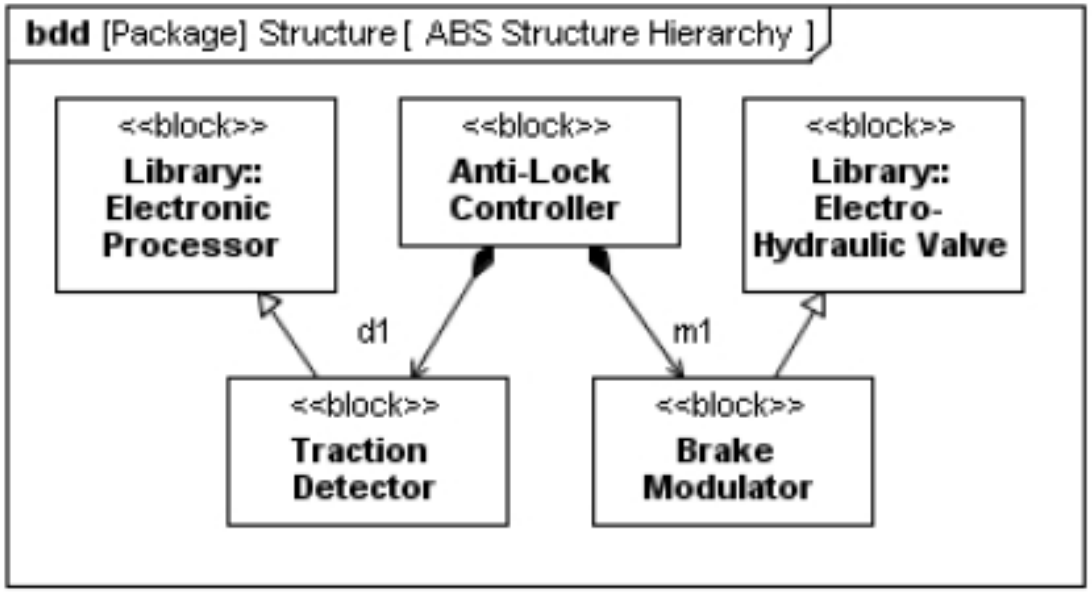
HOCHSCHULE HAMM-LIPPSTADT

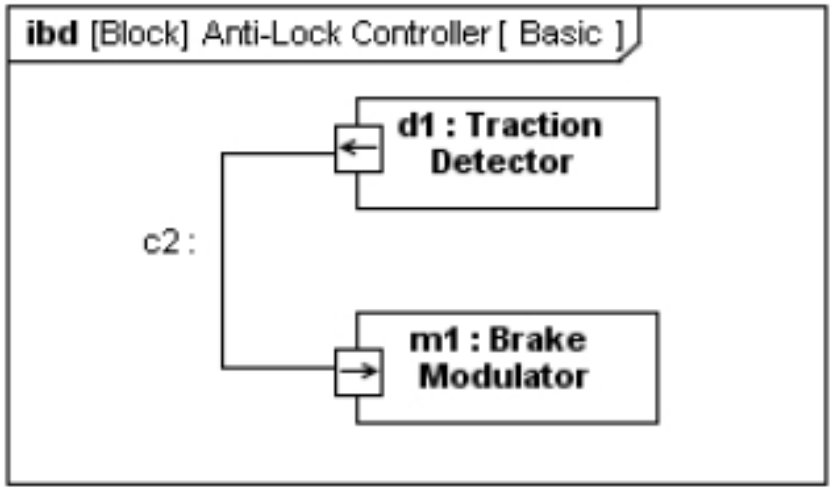
Definition

- Block is a definition/type
- Captures properties, etc.
- Reused in multiple contexts

Usage

- Part is the usage of a block in the context of a composing block
- Also known as a role

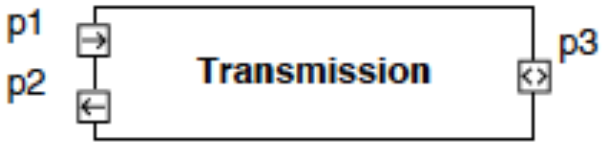




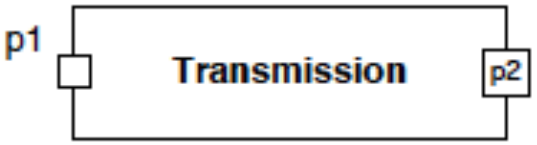
[SysMLTutorial09]

- The main motivation for specifying ports and flows is to enable design of modular, **reusable blocks** with **clearly defined** ways of **connecting** and **interacting** with their context of use.
- extends UML ports to support
  - nested ports
  - extends blocks to support flow properties
- Ports can be typed by blocks that support operations, receptions, and properties as in UML
- Two kinds of ports with respect of owning
  - One which exposes features of the owning block or its internal parts (**proxy ports**)
  - another that supports its own features (**full ports**)

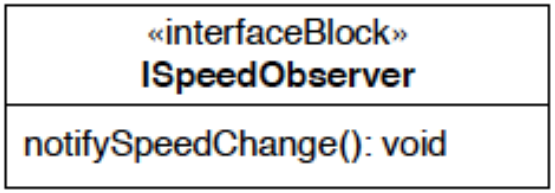
- Ports are points at which external entities can **connect to** and **interact** with a block in different or more limited ways than connecting directly to the block itself
- They are properties with a type that specifies features available to the external entities via connectors to the ports
  - including flow properties and association ends, as well as operations and receptions
- Port



Ports with Flow Properties

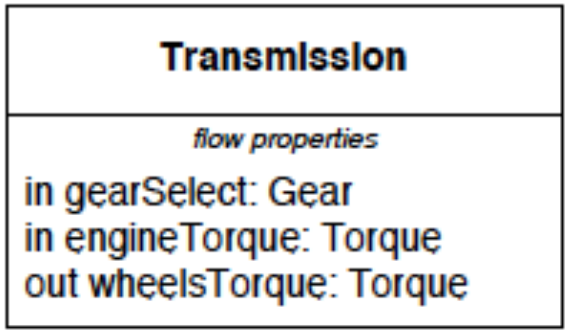


- Nested Port
  - ports can type other ports
    - Via nested ports
    - Defined by interface blocks

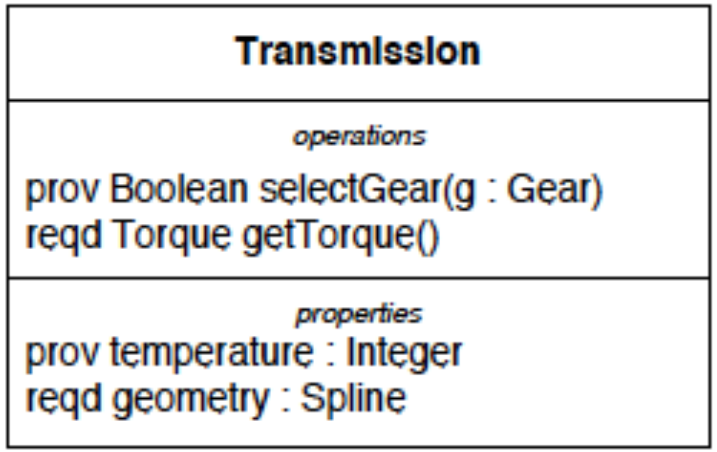


Flow Property, Required and Provided Features

- Extends blocks to support flow properties and provided and required features
- Flow properties specify the kinds of items that might flow between a block and its environment
  - Data
  - Material
  - Energy
- The kind of items that flow is specified by typing flow properties
  - E.g., a block specifying a car's automatic transmission could have a flow property for Torque as an input, and another flow property for Torque as an output.
- **Required** and **provided** features are operations, receptions, and non-flow properties that a block supports for other blocks to use, or requires other blocks to support for its own use, or both
  - See component ports

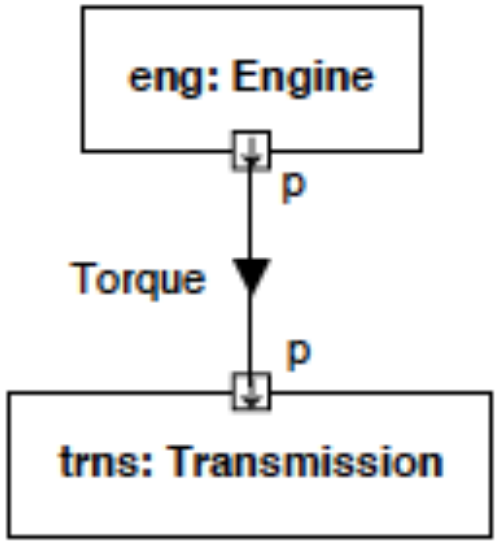


Ports with Flow Properties



Item flow

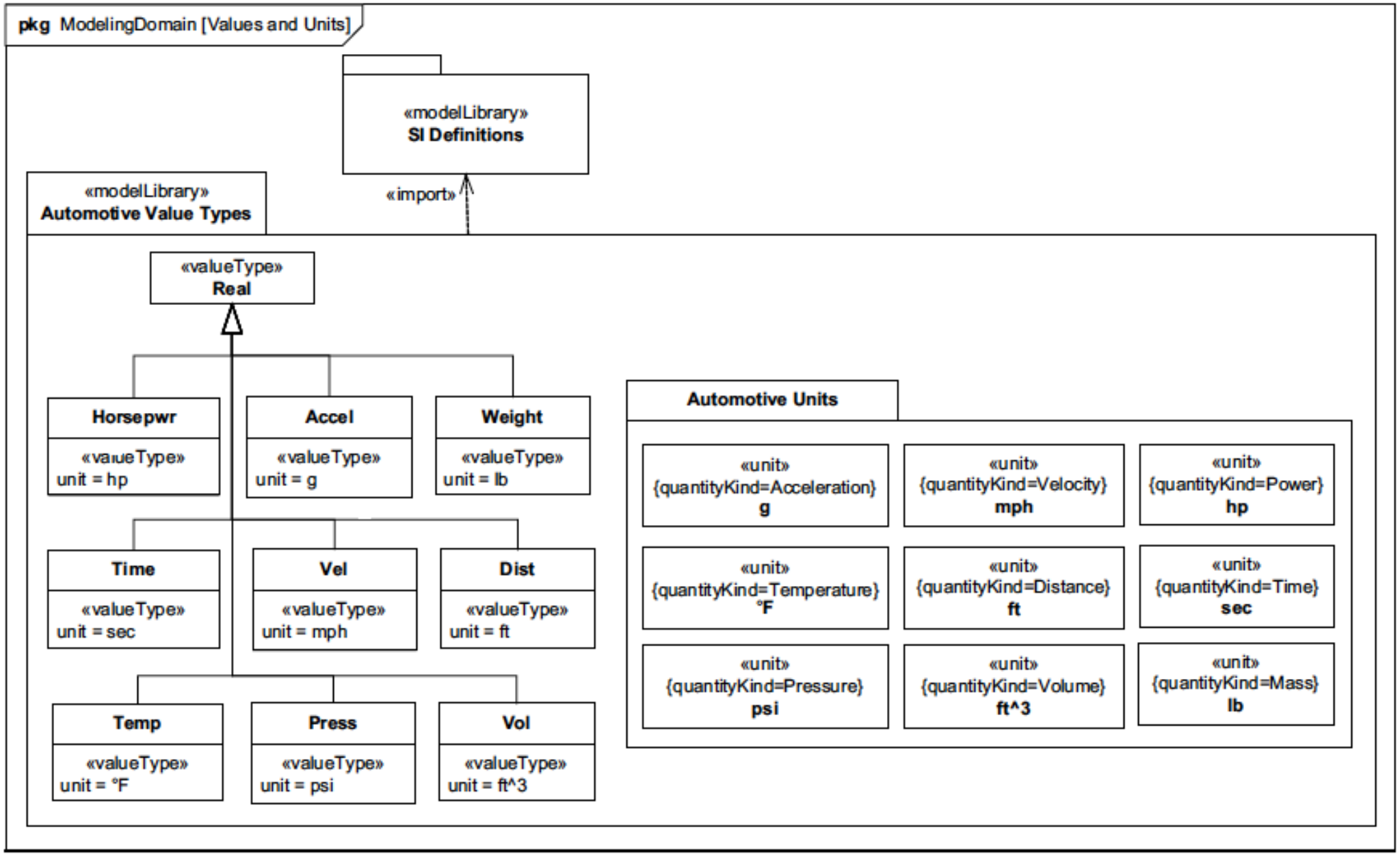
- Item flows specify the things that flow between blocks and/or parts and across associations or connectors
- Whereas flow properties specify what “can” flow in or out of a block, item flows specify what “does” flow between blocks and/or parts in a particular usage context



Item Flow

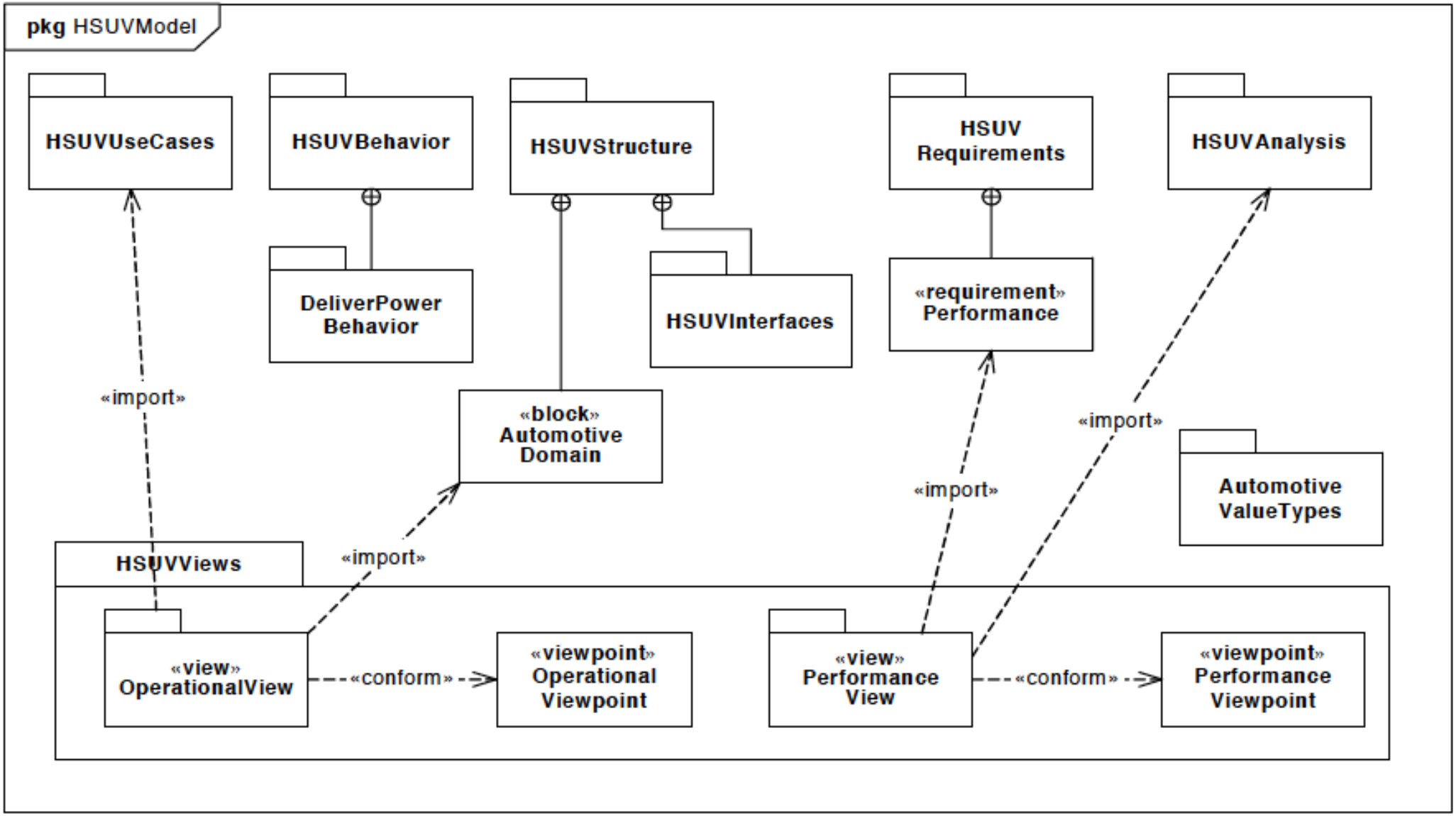
▸32 Automotive use case “Sample Problem”

specification of units and valueTypes employed in the sample problem



Sample Problem

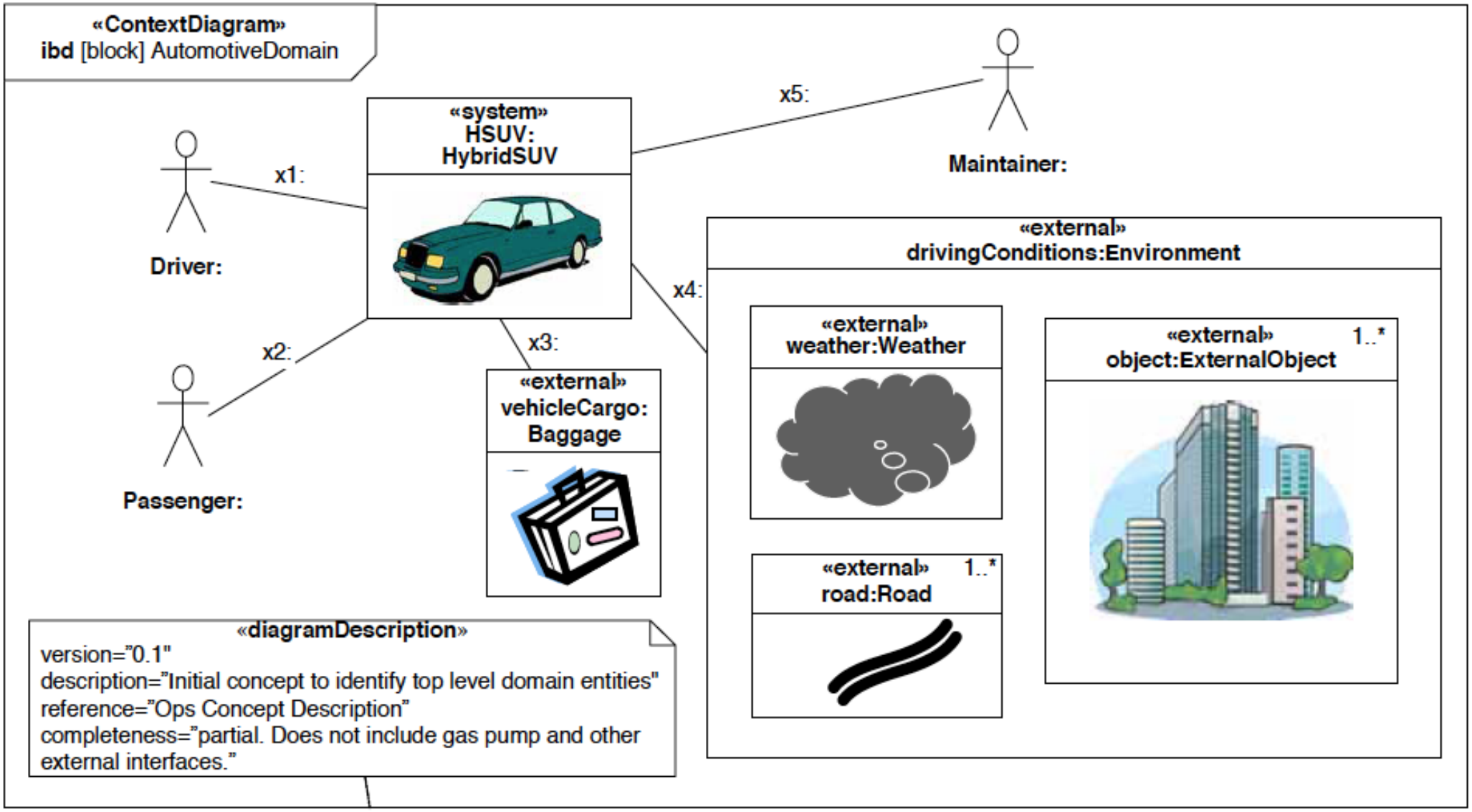
Package Diagram



Sample Problem

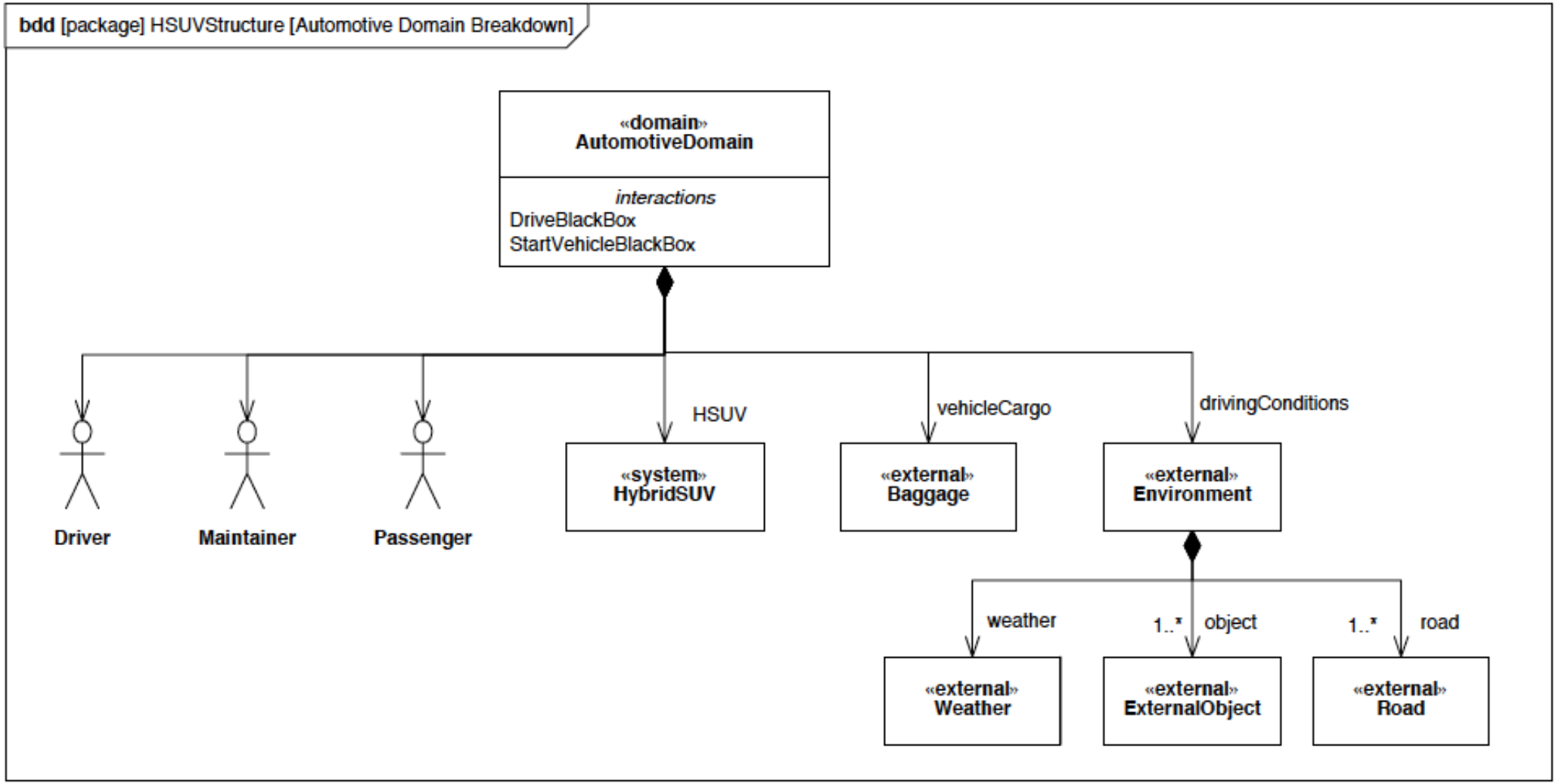
Setting the context

- ... See requirements lecture



Sample Problem

Block Definition Diagram – Automotive Domain



Homework

- Consider your Microcontroller project from last semester
  - Specify the blocks (bdd and ibd) on analysis level
    - Use paper and pen
  - In addition, if you have the possiblity, use a tool (e.g. draw.io)
- Readings
  - Tim Weilkiens, “Systems Engineering with SysML/UML” (see: https://learning.oreilly.com/library/view/systems-engineering-with/9780123742742/)
    - 4.5. Block diagrams (recap)
    - 4.6. Parametric block diagrams