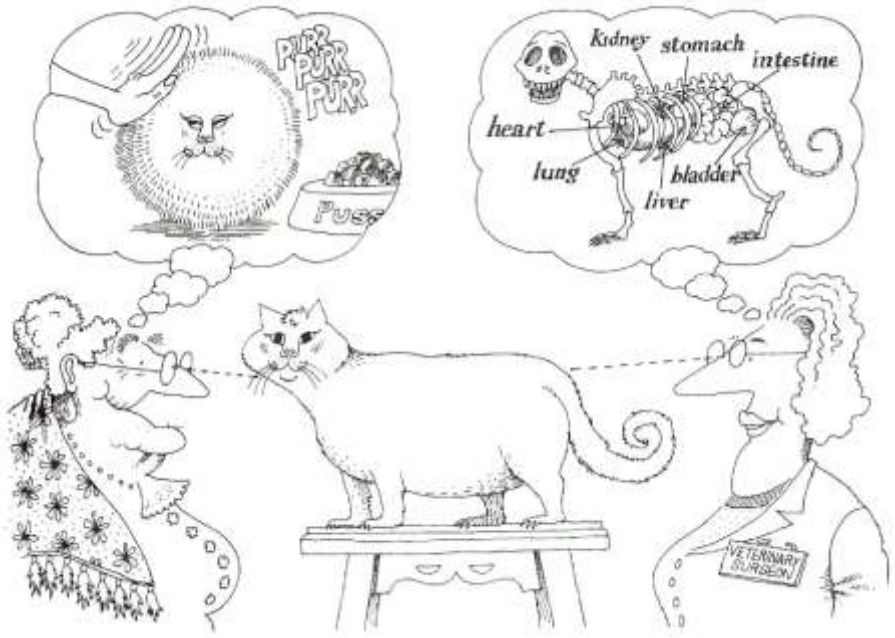
Object-Oriented Systems Engineering



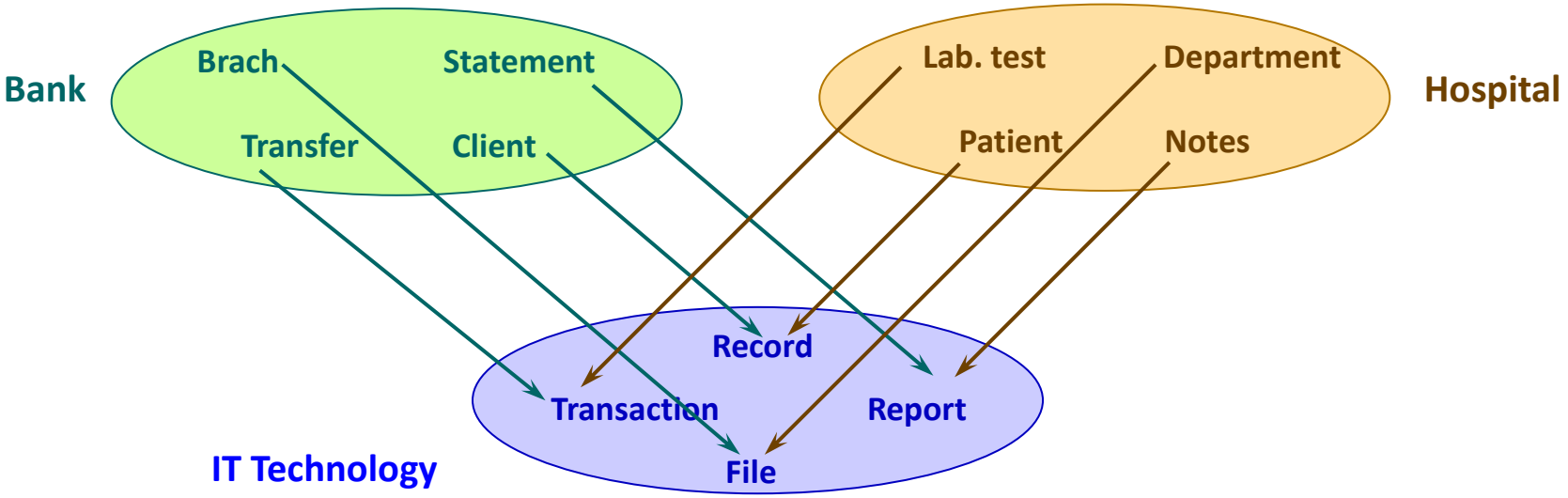
Picture Source: Booch, G., *Object-Oriented Analysis and Design*, Benjamin/Cummings, 1994

האיגוד הישראלי להנדסת מערכות
INCOSE IL
Israeli Society For Systems Engineering

amir@amirtomer.com

Motivation (Tomer, 1994)

- Engineering, unlike art, is usually not “creating from scratch”
  - Engineers map entities (objects) from the (client’s) problem domain onto entities (objects) from their technological domain
  - Different problem domains map onto the same technology domain



Object Mapping

- Mapping is based upon comparison between
  - The **required** properties (of the problem-domain object), and
  - The **existing** properties (of the technology-domain objects)

Operator's Work-Station

Required properties:
- is portable
- internet avail.: perm.
- weight: light

Inherited properties:
- **size: small**

may be implemented by

✗ 

Existing properties:
- ✓ is portable
- ✗ internet avail.: WiFi depend.
- ✗ weight: medium+
- size: medium

✗ 

Existing properties:
- ✓ is portable
- ✓ internet avail.: permanent
- ✗ weight: medium
- size: medium

✓ 

Existing properties:
- ✓ is portable
- ✓ internet avail.: permanent
- ✓ weight: light
- size: small

The Purposes of this Webinar

- To acquaint systems engineers with the Object-Oriented (OO) Thinking and its application to their profession
- To see a few examples of OO practices used in SE
- To improve the cooperation between Systems Engineers and Software Engineers who use OO design and programming

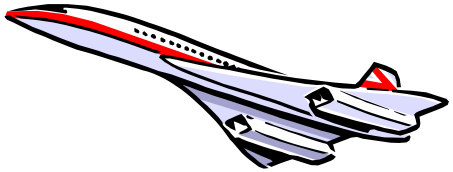
Prof. Amir Tomer – short bio

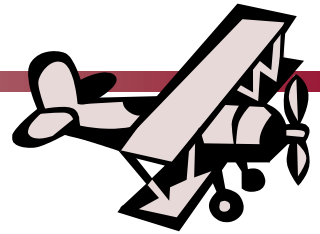
- Education
  - B.Sc. & M.Sc. in Computer Science – Technion, Israel
  - Ph.D. in Computing – Imperial College, London
- Industrial experience
  - 1982-2009: Rafael, Advanced Defense System
    - 1999-2006: Director of Software and Systems Engineering Processes
- Academic experience
  - 2009-now: Head of Software Engineering Department, Kinneret College on the Sea of Galilee, Jordan Valley, Israel
  - 1994-now: Senior lecturer, Technion, Haifa, Israel
- Research interests
  - Software-intensive Systems Engineering
  - Model-Based systems/Software Engineering
  - Software reuse

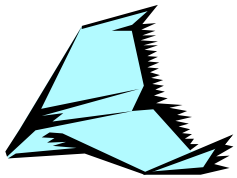
Object-Oriented (OO) Thinking

- Organizing knowledge about a world as a set of discrete entities related to each other
  - "A world" may be
    - A field of knowledge
    - An organization
    - A system-of-systems, a system, a subsystem etc.
    - ...
  - Very natural and intuitive – dates back many years
  - Adopted by the programming world as a programming paradigm
    - from "think like a computer" (C-like languages)
    - to "think like a developer" (Java-like languages)

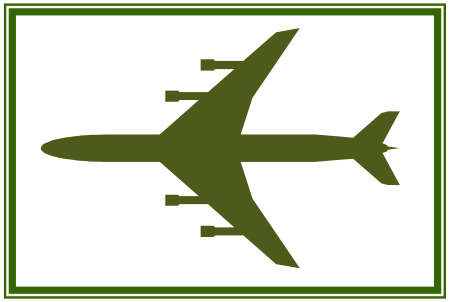
The OO view on the world





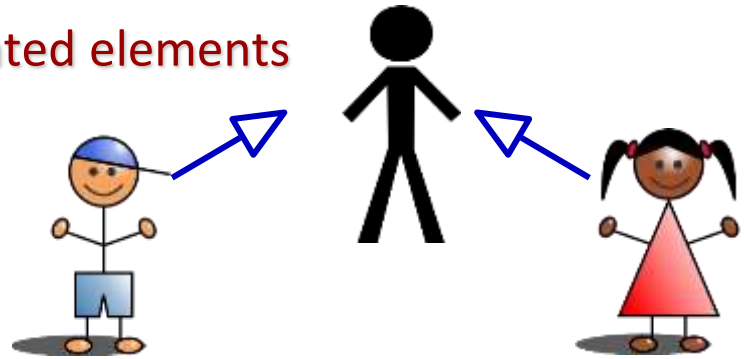


- Object: An element of the world
  - Discrete
  - Defined borders and identity
  - Has **properties** (attributes, data) and **capabilities** (functions)
  - Belongs to a class
  - **Objects exist in real!**
- Class: An abstract description of a group of objects
  - The objects belonging to the same class share
    - Properties
    - Functions
    - Relations
    - Behavior
  - **Classes are abstractions**
  - **Objects are instantiations from classes**



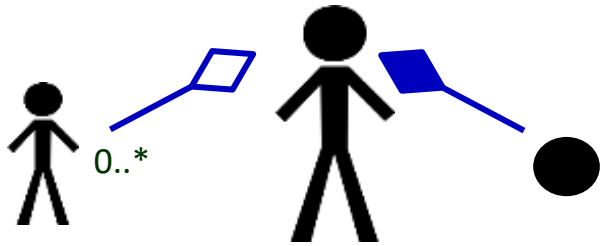
Basic Relations and Semantic Networks

- A Semantic Network is a set of inter-related elements
  - Relations are of 3 kinds* :



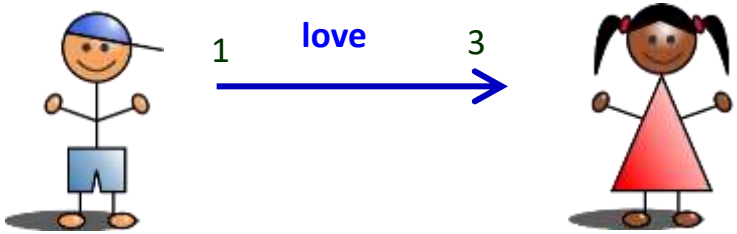
Classification / Generalization

- **"A is a B"** or **"A is a kind of B"**
  - a **BOY** is (a kind of) a **PERSON**
  - a **GIRL** is (a kind of) a **PERSON**



Aggregation / Composition

- **"A has a B"** or **"B is a part of A"**
  - a **PERSON** has a (0 or more) **CHILDREN**
  - a **HEAD** is a part of a **PERSON**



Association

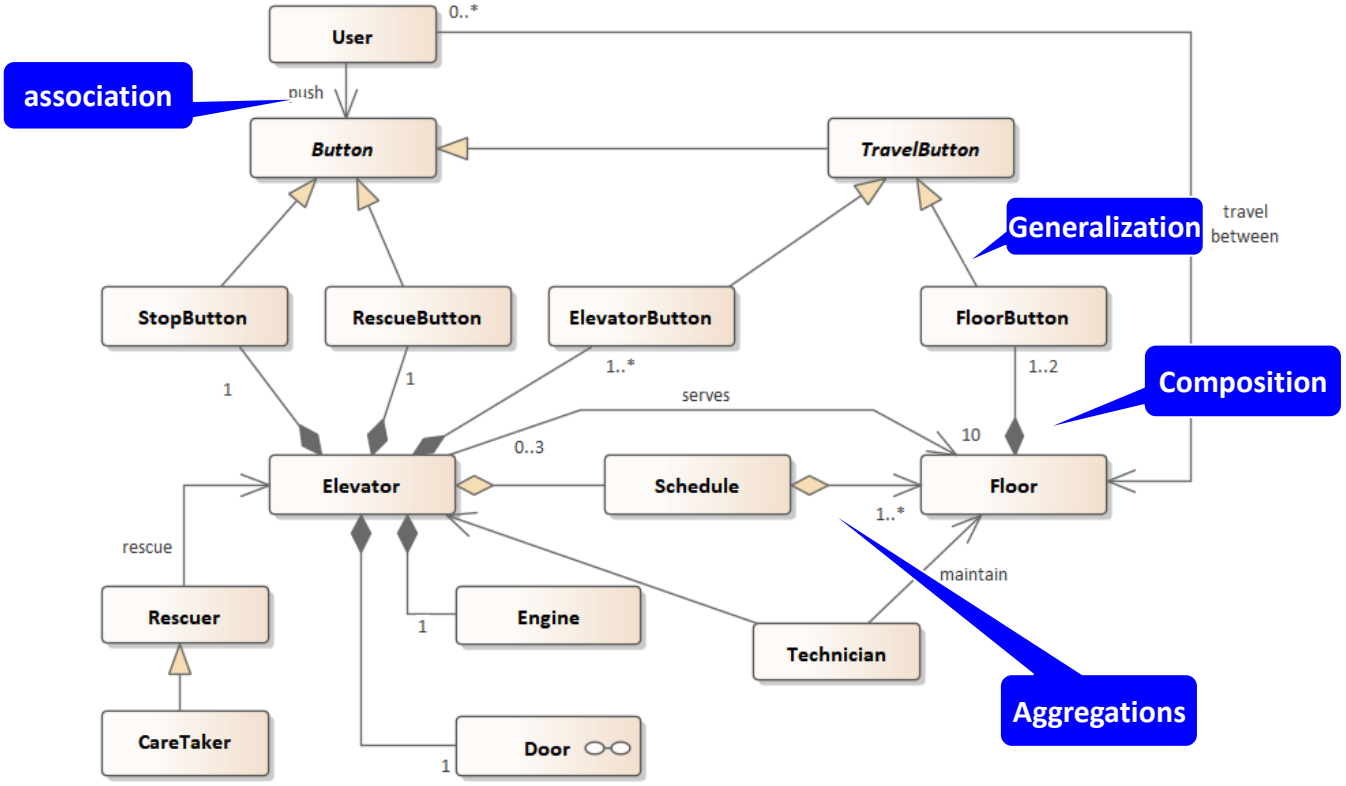
- B <relates to> A
  - a BOY loves 3 GIRLs

** (UML/SysML annotations)*

Use-Case #1: PDOM (Problem-Domain Object Model)

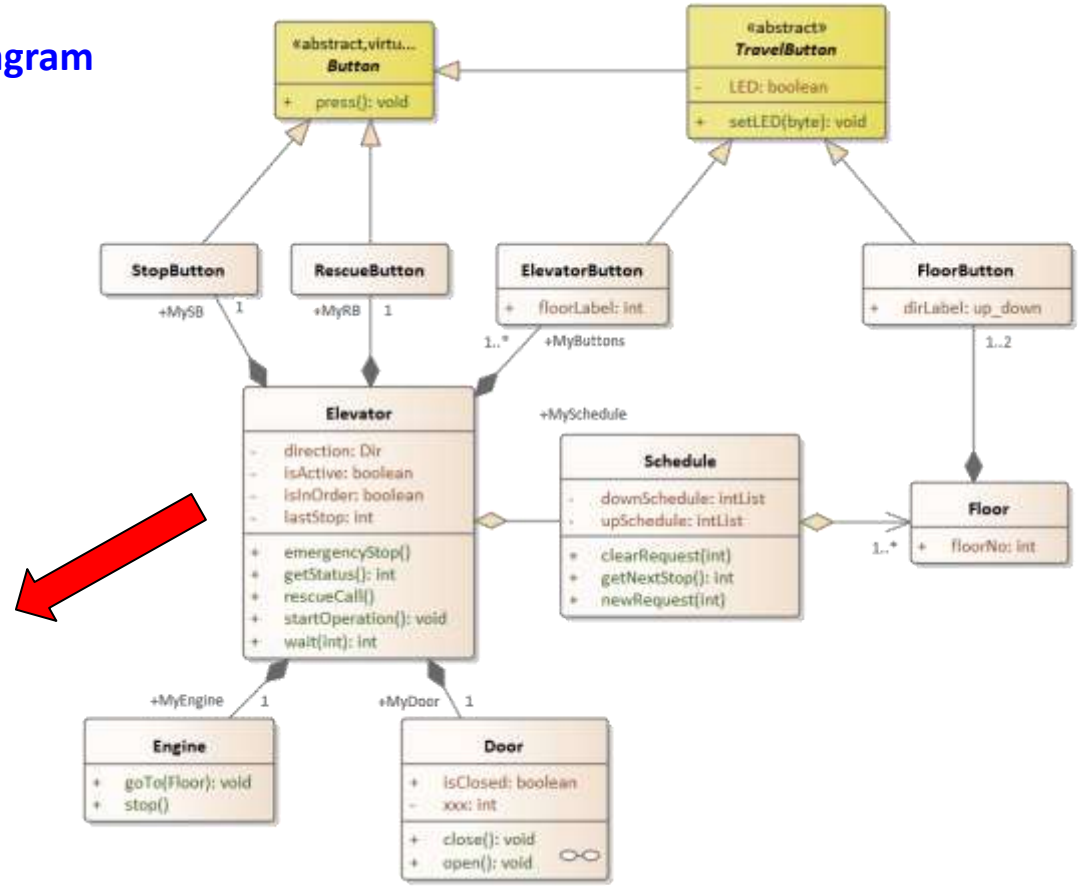
- PDOM is a representation of the (customer's) knowledge about the system
  - Sharing common knowledge among all stakeholders
  - No contradictions, ambiguities and unknowns

Example: elevator System



PDOM as the Basis for Software Design / Code Structure

Class Diagram



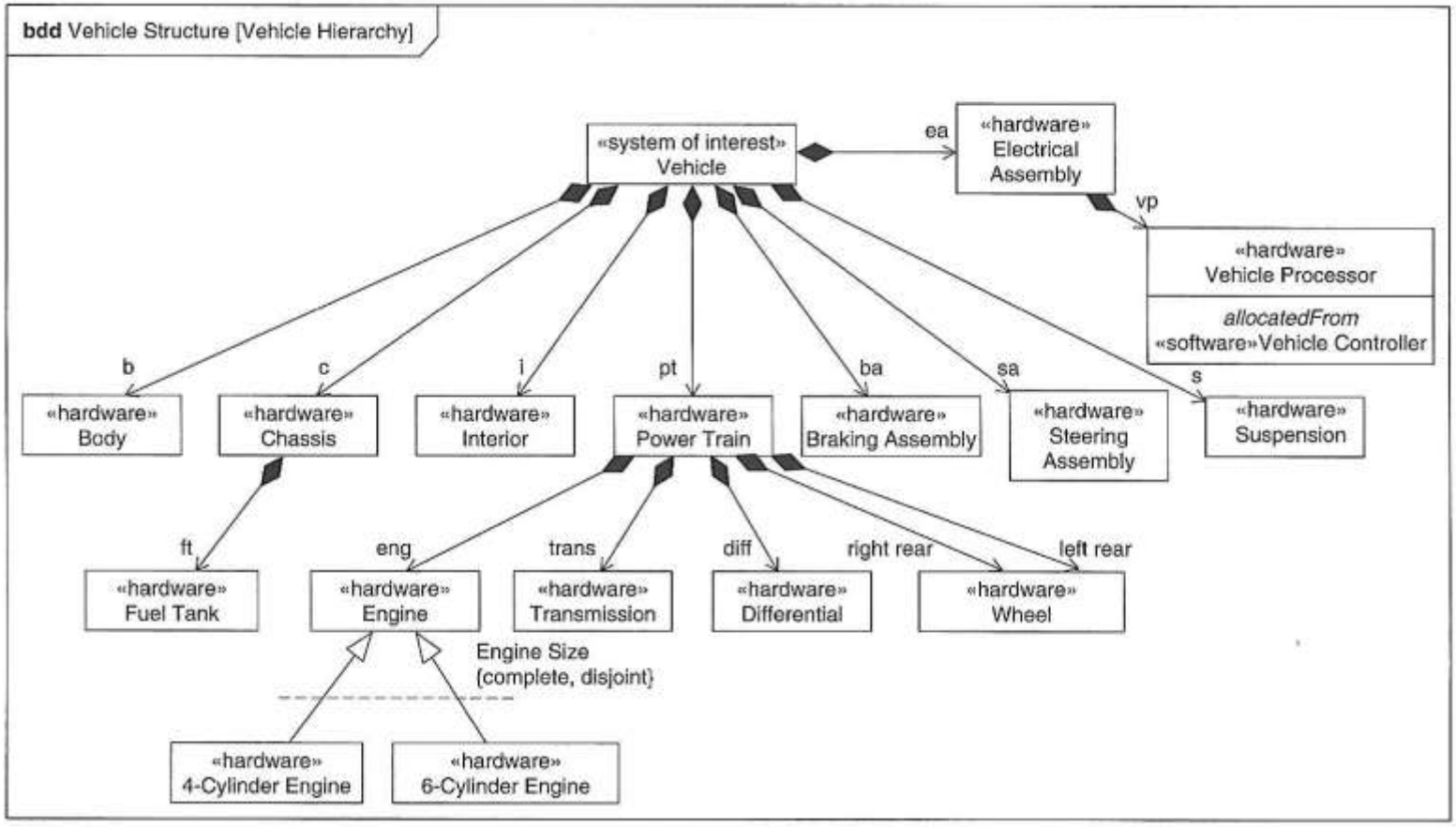
Static Code (Java)

```
public class Elevator {

        public Elevator(){ }
        public void finalize()
                throws Throwable { }
        public emergencyStop(){}
        public int getStatus(){
                return 0;
                }
        public rescueCall(){}
        public void startOperation(){ }
        public int wait(int Sec){}
```

Use-Case #2: Block Definition

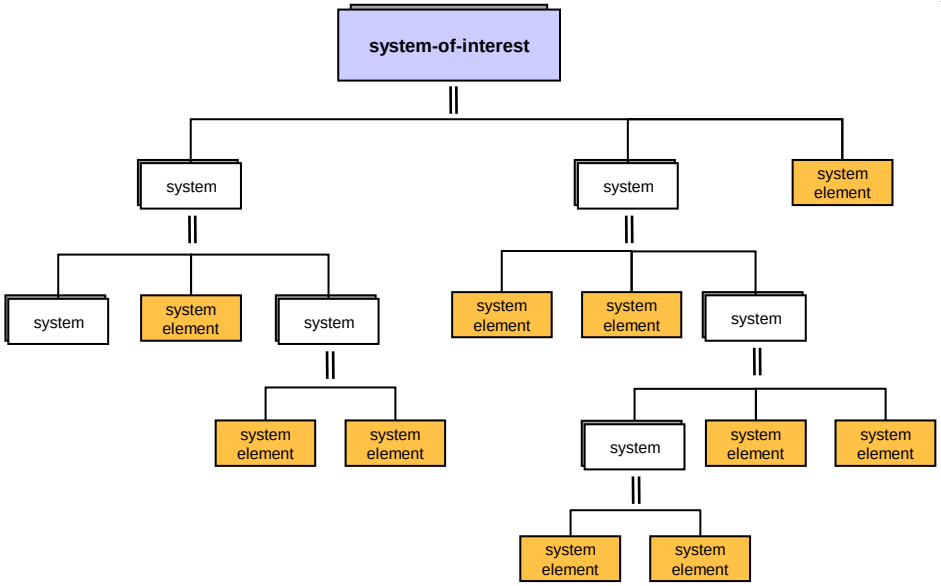
- SysML's BDD reflects the knowledge about a "system of interest"
  - Each element in the diagram may be elaborated with further specification

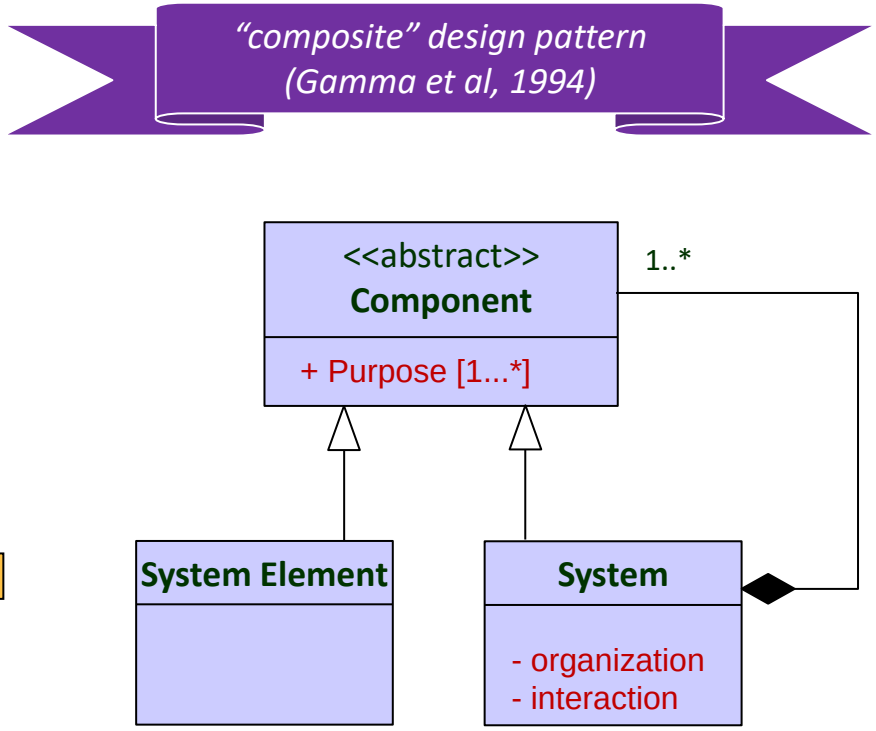


Picture Source: A Practical Guide to SysML (see resources slide)

Use Case #3: System Breakdown

- System – Definition*
  - combination of interacting elements organized to achieve one or more stated purposes

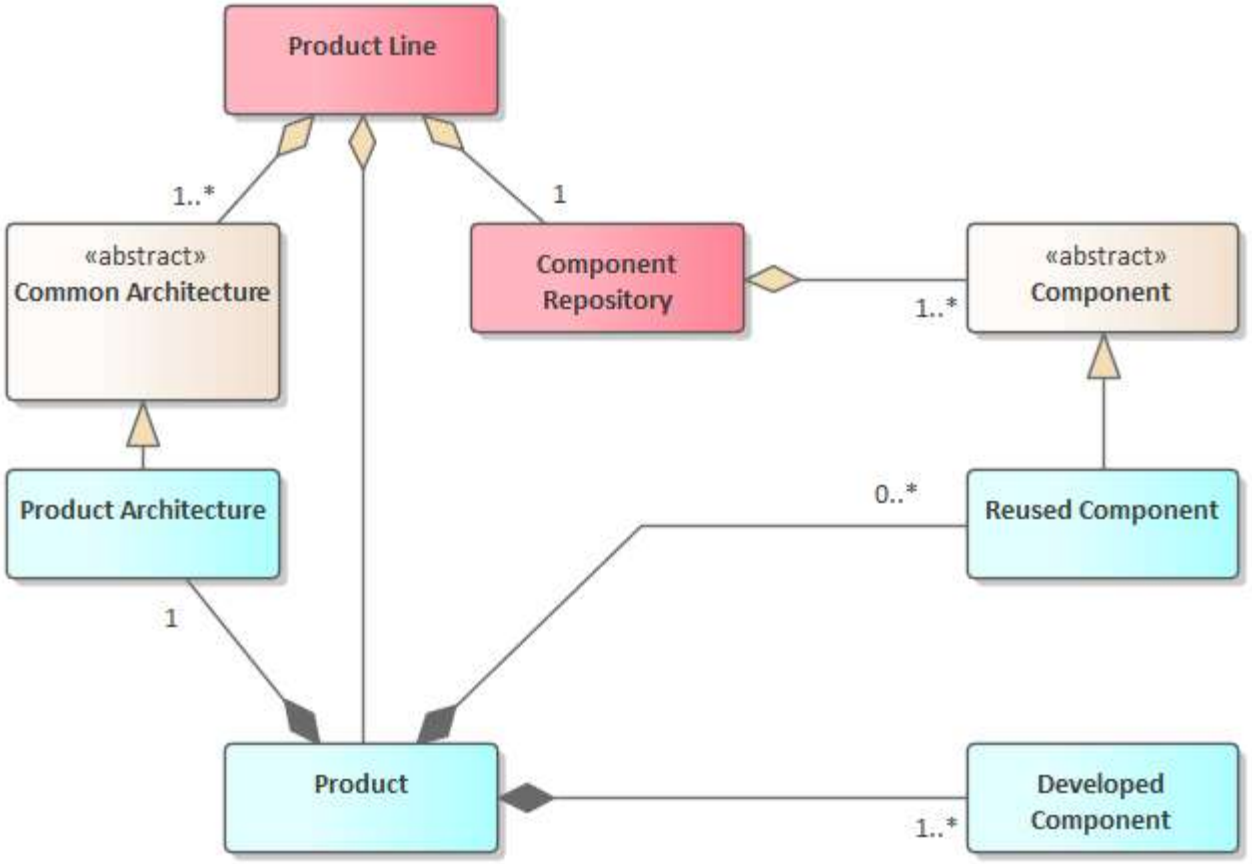




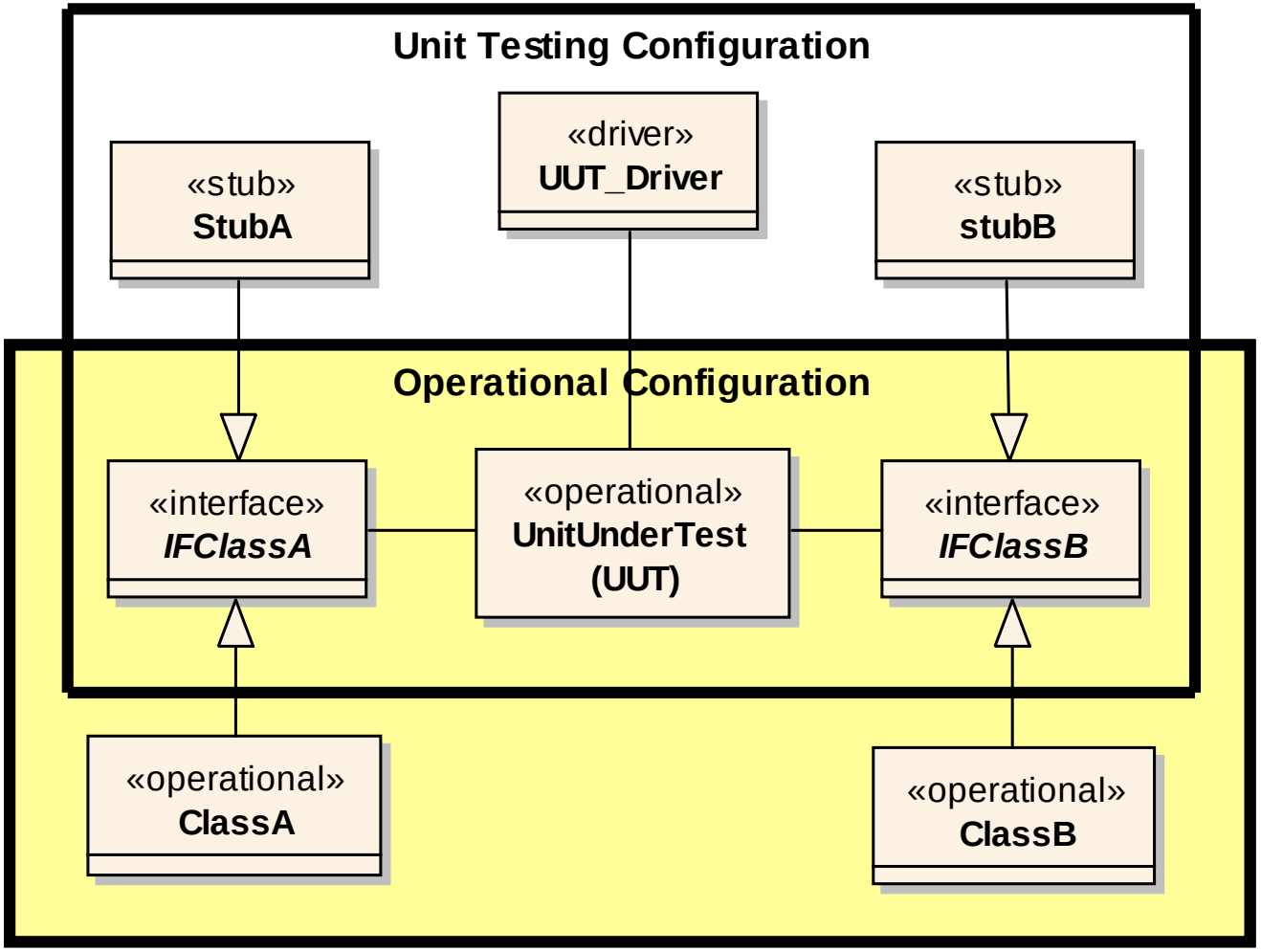
* ISO/IEC/IEEE 15288:2015 *Systems and software engineering -- System life cycle processes*

Use Case #4: Abstract Structures

- Example: Product-line Organization with OO Approach (abstraction)



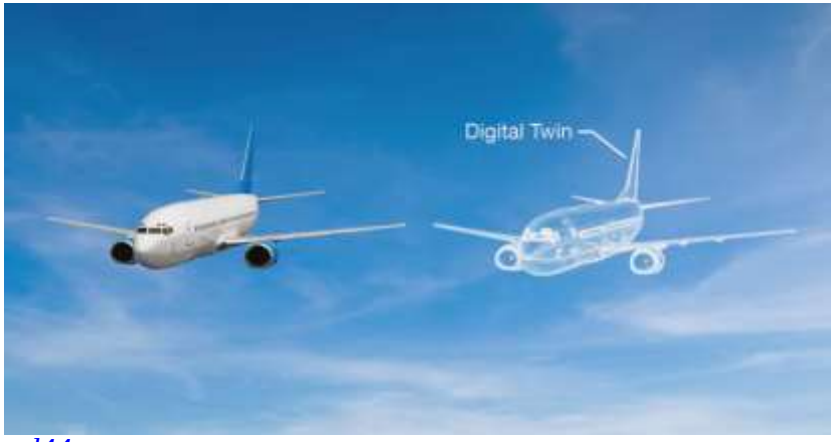
Use Case #5: Aligning Configurations



- See also (Tomer, 2012)

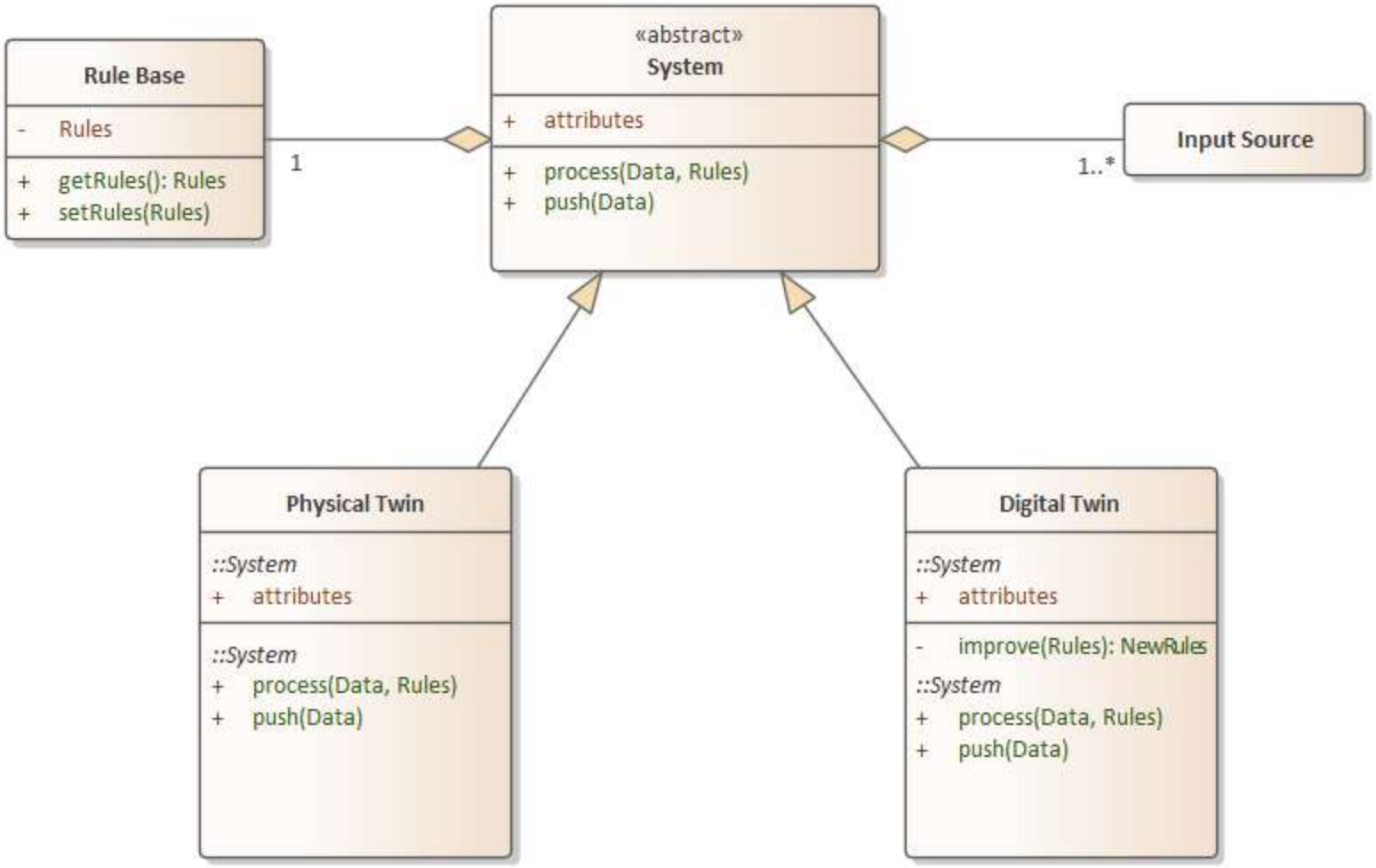
Use Case #6: Digital Twins

- Many systems have Digital-Twins
  - A digital-twin is a simulated clone of the system
- Digital twin's principles
  - Both twins share the same (abstract) architecture
  - Both twins respond to same inputs
  - Both twins share the same set of operational rules
  - Both twins apply the same process-types, based on the shared rule-base
    - Physical twin – physically
    - Digital twin – simulatively
  - The digital twin has a machine-learning mechanism which enables it to improve the rule-base



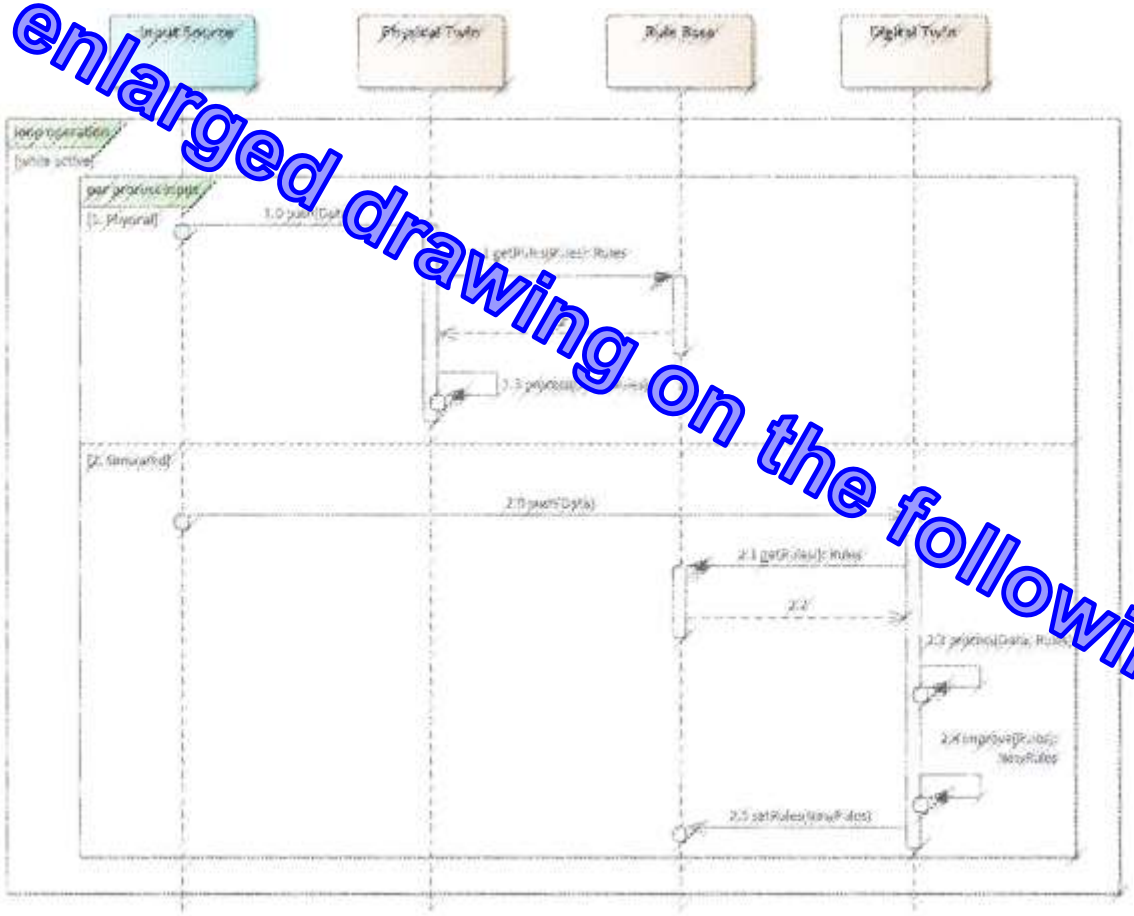
Picture source: https://medium.com/future-today/what-is-a-digital-twin-e38f3b6acd44

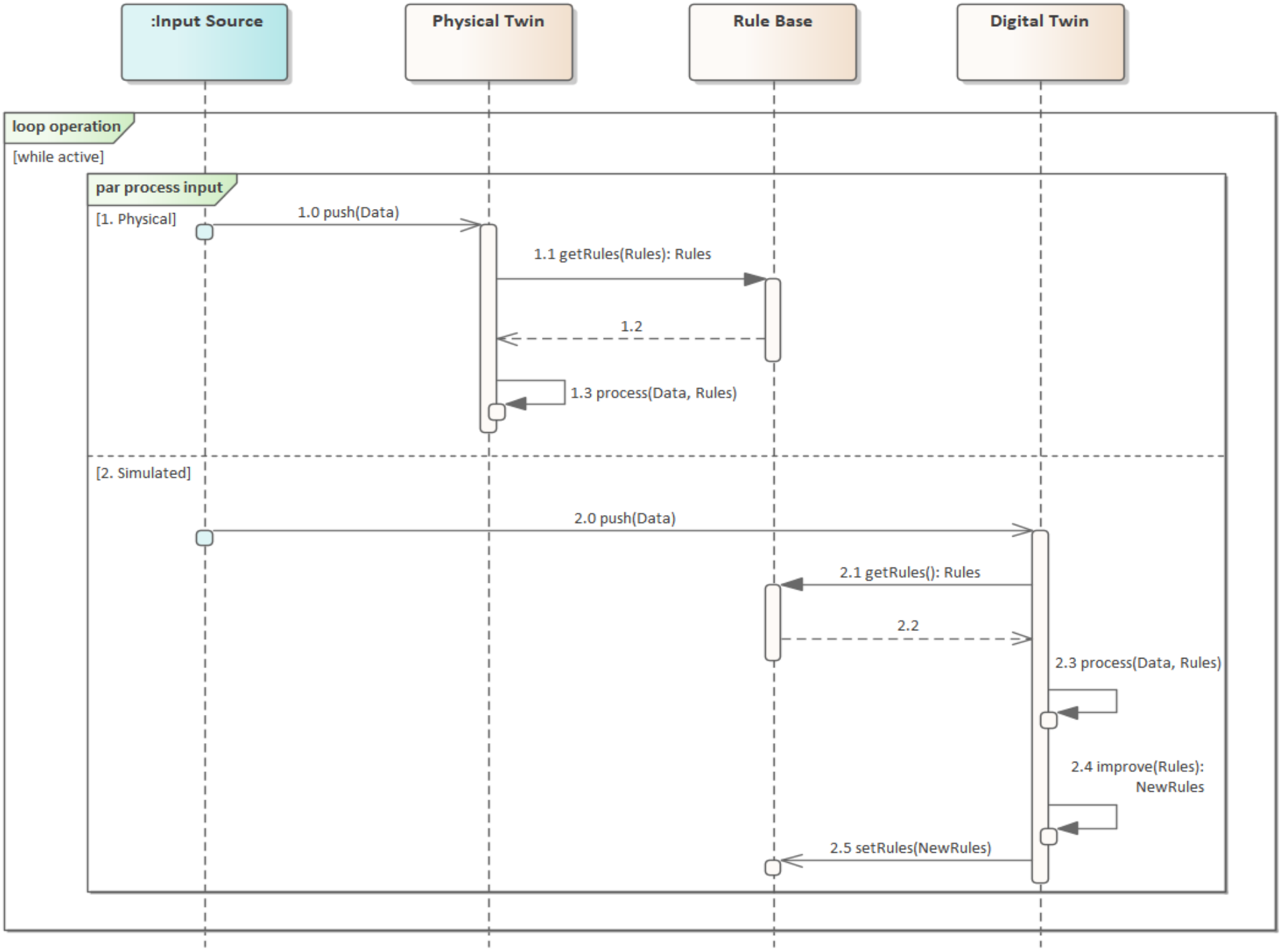
Digital Twin “pattern” - Structure



Digital Twin “pattern” - Behavior

see enlarged drawing on the following slide



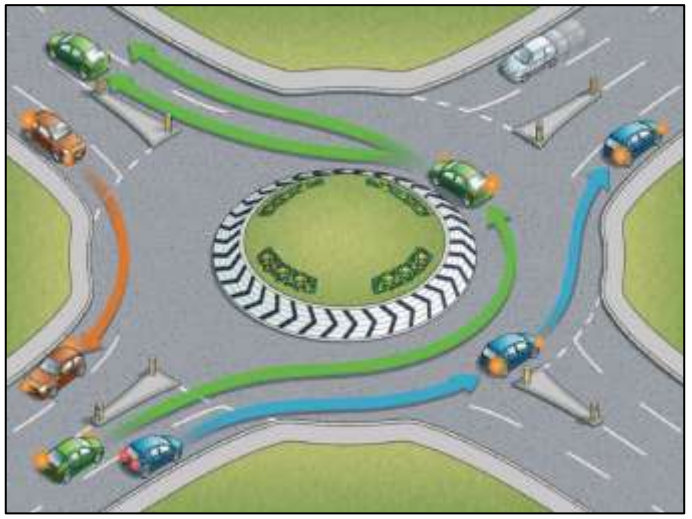

:Input Source
Physical Twin
Rule Base
Digital Twin
loop operation
[while active]
par process input
[1. Physical]
1.0 push(Data)
1.1 getRules(Rules): Rules
1.2
1.3 process(Data, Rules)
[2. Simulated]
2.0 push(Data)
2.1 getRules(): Rules
2.2
2.3 process(Data, Rules)
2.4 improve(Rules): NewRules
2.5 setRules(NewRules)

The Encapsulation Principle

- Each object is an autonomous entity
  - Self responsibility for its attributes and functions
  - No direct access to private data
  - Interaction with other objects via defined interfaces
    - In software: API = Application Program Interface
- The autonomous behavior is derived from the class definition
- Advantages
  - A class (i.e. any of its derived objects) may change without affecting other objects
    - enhance modularity and loose-coupling
  - Reusability

Encapsulation: What is the Difference between the two Junctions?

- Fixed junction management
  - The “junction management” algorithm is in possession of the traffic-light only
  - The traffic-light has no information about the cars



- Adaptive junction management
  - The “junction management” algorithm is in possession of every car
  - Every car manages itself in interaction with other cars
  - **Better efficiency**

OO Methodologies for Systems Engineering

- OOSEM = OO Systems Engineering Methodology (INCOSE, 2015, Ch. 9.4)
  - Integrates OO concepts with MBSE and other traditional SE methods
- SysML = Systems Modeling Language (Friedenthal et al, 2009)
  - A modeling language for systems engineering
  - Developed from UML
  - Demonstrates OOSEM
- OPM = Object Process Methodology (Dori, 2002)
  - A conceptual modeling language and methodology for capturing knowledge and designing systems
  - Based on an ontology of stateful objects and processes that transform them
  - Specified as ISO/PAS 19450

References

- Friedenthal, S., Moore, A., Steiner, R., *A Practical Guide to SysML,* Elsevier, 2009
- Gamma, E. et al, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.
- Tomer, A. & Hogger, C. J., A logic approach to knowledge-based engineering, *WIT Transactions on Information and Communication Technologies*, Vol. 6, WITpress, 1994

  https://www.witpress.com/elibrary/wit-transactions-on-information-and-communication-technologies/6/10952

- Tomer, A., Software Modeling from Life-Cycle Perspective, *Proceedings of the IEEE CS International Conference on Software, Science, Technology, and Engineering (SwSTE 2012), Hertzliyah, Israel, June 12-13*, 2012.

  DOI: 10.1109/SWSTE.2012.17

- INCOSE Systems Engineering Handbook, Wiley, 2015.

What else?

- Other tutorials by this author
  - Webinars (~1 hour each)
    - Making Use Cases great again
    - The Functionality behind the Non-functional Requirements
    - Relay Race: The Shared Challenge of Systems- and Software-Engineers
  - Workshops
    - Software requirements and use cases (1 day)
    - Software engineering for systems engineers (2 days)
    - Software-intensive systems architecture (3 days)
- For details go to https://incoseil.org

Thank you for listening