Bridging the gap between the system and software architectures using ISE&PPOOA MBSE methodology

Webinar at INCOSE Chesapeake chapter
21 September 2022

José L. Fernández
jose.fernandez@incose.net

Content

1. What is a MBSE methodology?
2. Overview of main MBSE concepts as used by ISE&PPOOA
3. The ISE&PPOOA conceptual model/ontology
4. ISE&PPOOA dimensions and views
5. ISE&PPOOA process
6. PPOOA software architecture framework and software architecting process
7. The domain model. The bridge between the system and the software architectures
8. To conclude

Speaker bio

Jose L. Fernandez has a PhD in Computer Science, and an Engineering Degree in Aeronautical Engineering, both by the Universidad Politécnica de Madrid (Spain).

He has over 30 years of experience in industry, involved in projects dealing with software development and maintenance of large systems, specifically real-time systems for air traffic control, power plants, avionics and cellular phone applications.

Author the MBSE methodology ISE&PPOOA, books, journal papers and conferences papers.

He was associate professor at the Universidad Politécnica de Madrid (UPM) until 2018.

INCOSE member, IEEE senior member and PMI member.

1.What is a MBSE methodology?

Process(es)+method(s)+tool(s)

What is a MBSE methodology?

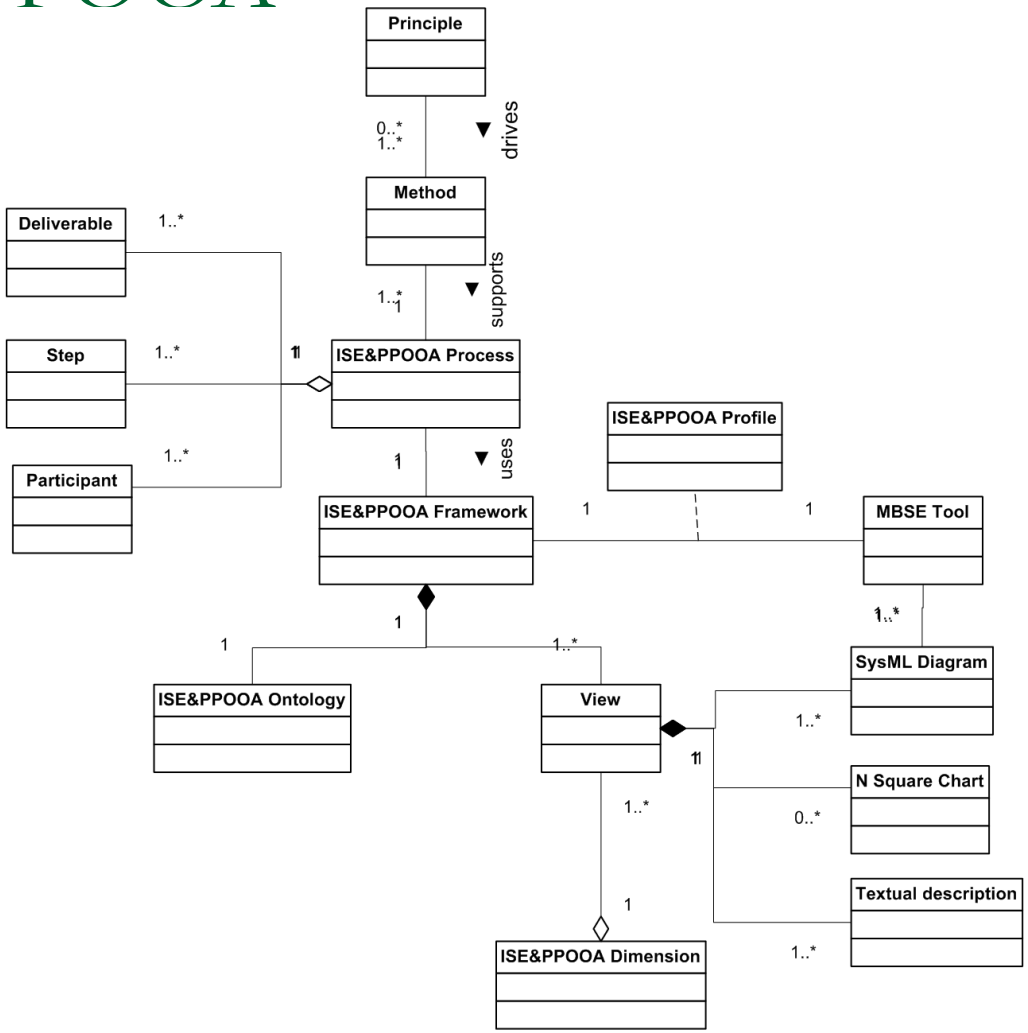
(Estefan, 2008) characterizes a **MBSE methodology** as the collection of related **processes, methods, and tools** used to support the discipline of systems engineering in a “model-based” or “model driven" context

- **Process**: logical sequence of steps performed to achieve a particular objective **(What to be done)**
- **Method:** is a *technique, practice or procedure* for performing an step task **(How to be done)**
- **Tool**: is an instrument that, when applied to a particular method, can enhance the efficiency of the task; provided it is applied properly and by somebody with proper skills

2.Overview of main MBSE concepts as used by ISE&PPOOA

The context of the ISE&PPOOA process and framework

Overview of main MBSE concepts as used by ISE&PPOOA



Main MBSE concepts as used by ISE&PPOOA

- The **ISE&PPOOA process** uses the **ISE&PPOOA framework**
- The **ISE&PPOOA framework** contains the ISE&PPOOA **ontology** (described later) and the **set of views** to represent the **system** and provided as **deliverables** (described later)
- **Engineering principles** support methods or practices used to support ISE&PPOOA process
- The **ISE&PPOOA main process** is described by its **steps, deliverables and participants**
- Some **MBSE tools** supporting **SysML diagrams** may be tailored to support ISE&PPOOA using a **profile (optional)**

Systems engineering principles

- As stated by the **SEBoK**, **SE principles** are a specialized and contextualized instantiation of systems principles that address the approach to the realization, use, and retirement of systems
- System principles **guide the definition and application of SE processes**
- A **principle** transcends a particular lifecycle model or phase, transcends system types, transcends a system context, informs a world view on Systems Engineering, is not a “how to” statement, is supported by literature or widely accepted by the community; i.e. has proven successful in practice across multiple organizations and multiple system types, and is focused, concise, and clearly worded (SEBoK)

Principles driving ISE&PPOOA

- **Breadth first development**. The ISE&PPOOA process recommends a breadth first approach in which scope is covered comprehensively before fidelity is addressed

- **Form follows function**. This principle states that the shape of a building or object should primarily relate to its intended function or purpose

- **Modularity or modular design** is a design principle that subdivides a system into smaller parts called modules which can be independently created, modified, replaced, or exchanged with other modules or between different systems. Modularity is based on **functional cohesion and minimum coupling**

Scope and fidelity

- **Scope** is defined as the area within the boundaries of the problem space. (Topper and Horner, 2013)
- **Fidelity** is defined as the level of detail incorporated into the modeling or analysis activities associated with the problem(Topper and Horner, 2013)

Tip: Scope in ISE&PPOOA

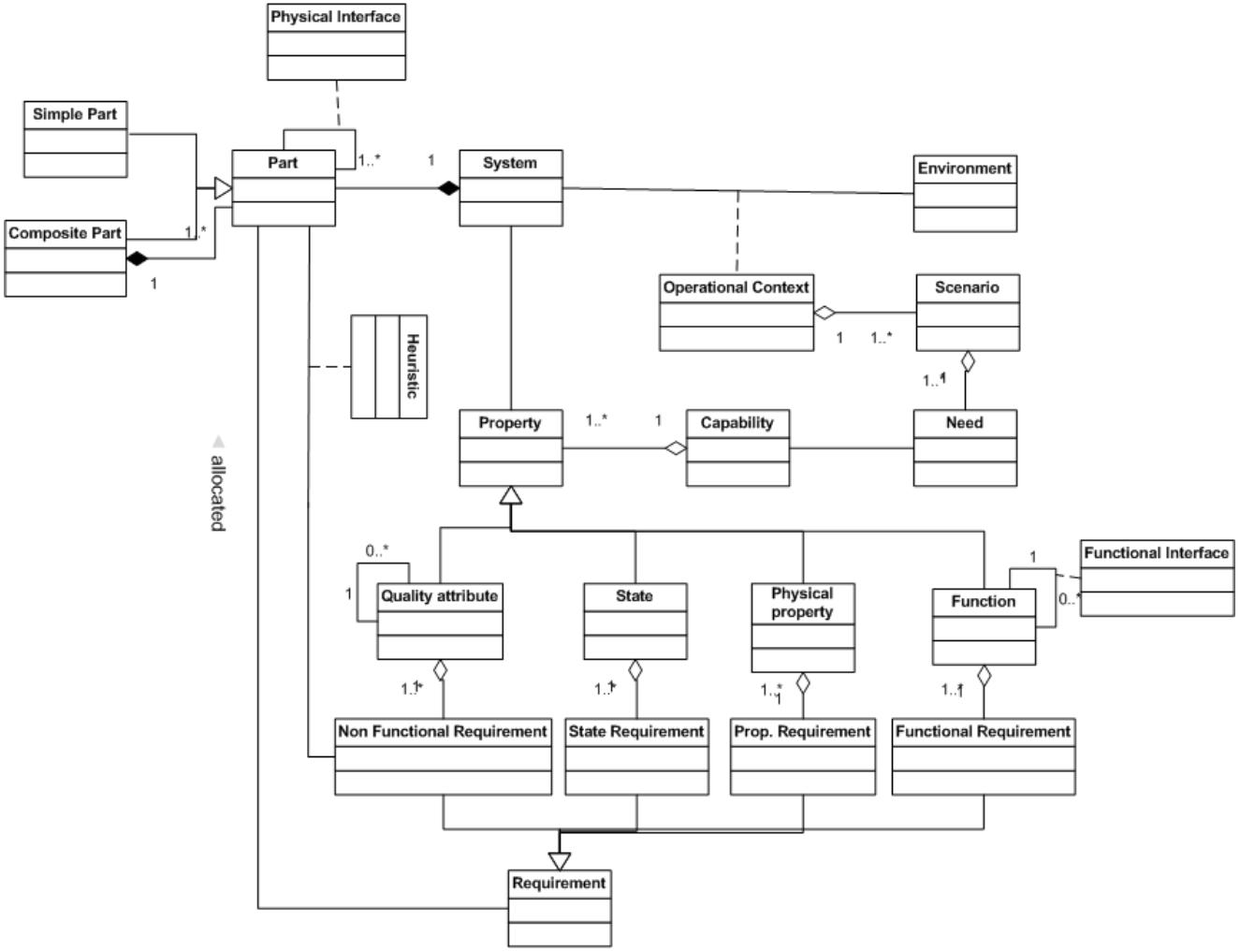
We recommend to model:

- All system inputs and outputs including matter, energy and data (**context diagram**)
- Use cases or interactions between the system and external entities (**use cases diagram**)
- All functions (**functional hierarchy Block Definition Diagrams**, functional interfaces (**N Square chart**), functional flows (**activity diagrams**)
- All quality attributes (**Block Definition Diagrams**)
- All subsystems and their interfaces (matter, energy and data) (**Internal Block Diagrams**)
- System requirements (functional+nonfunctional) (**Requirements diagram or requirements tables**)

3.The ISE&PPOOA conceptual model/ontology

Understanding the concepts to be used in the MBSE process

ISE&PPOOA conceptual model



Brief description of the ISE&PPOOA conceptual model

- A **system has parts** that may be either simple or composite parts.
- A **system interacts with the environment**. These interactions are described by an operational context that models the interactions as a set of scenarios.
- Based on the **operational context and scenarios**, the engineer translates the set of specific needs into a set of **system capabilities** that should be solution-independent.
- Each capability is a container of **system properties** that may be either system **quality attributes, physical properties, states, or functions.**
- In contrast to **functional requirements that are allocated** to system parts, nonfunctional requirements implementation is essentially different. **Nonfunctional requirements may be met by the application of design heuristics**. For this reason a specific association is depicted between the nonfunctional requirement concept and the part concept as shown in previous slide

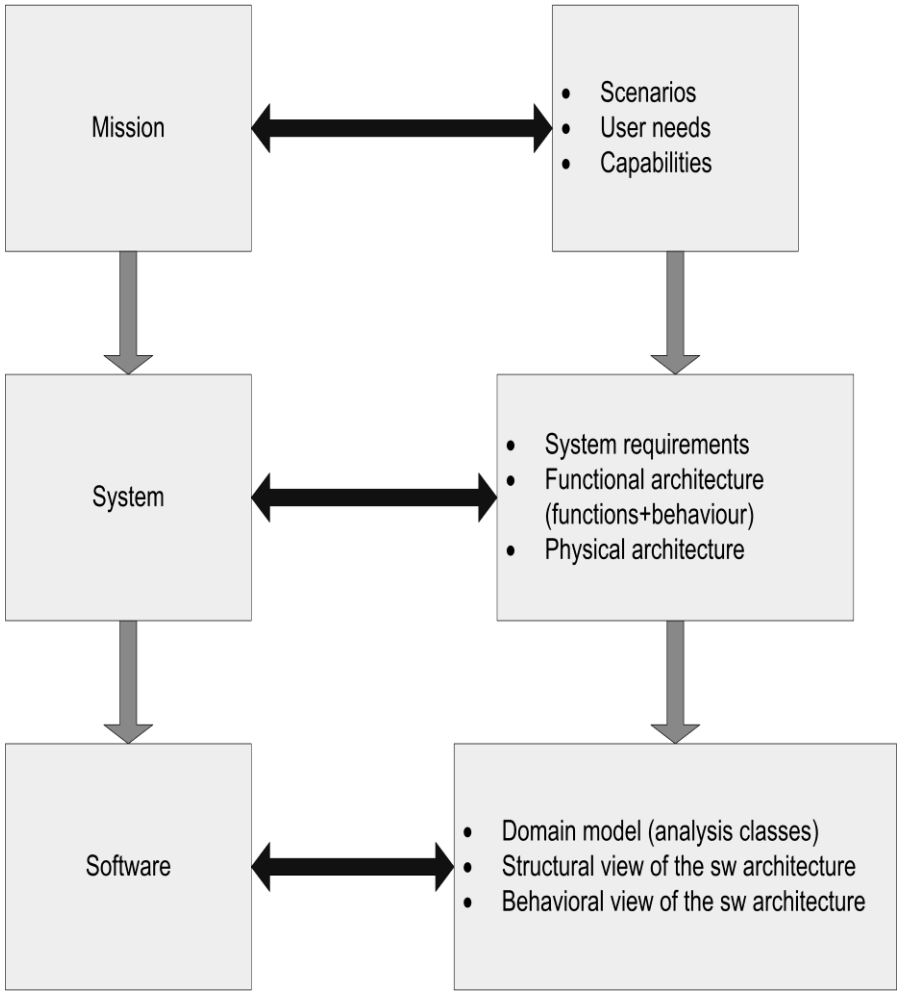
4. ISE&PPOOA dimensions and views

How the model scope is organized

ISE&PPOOA dimensions

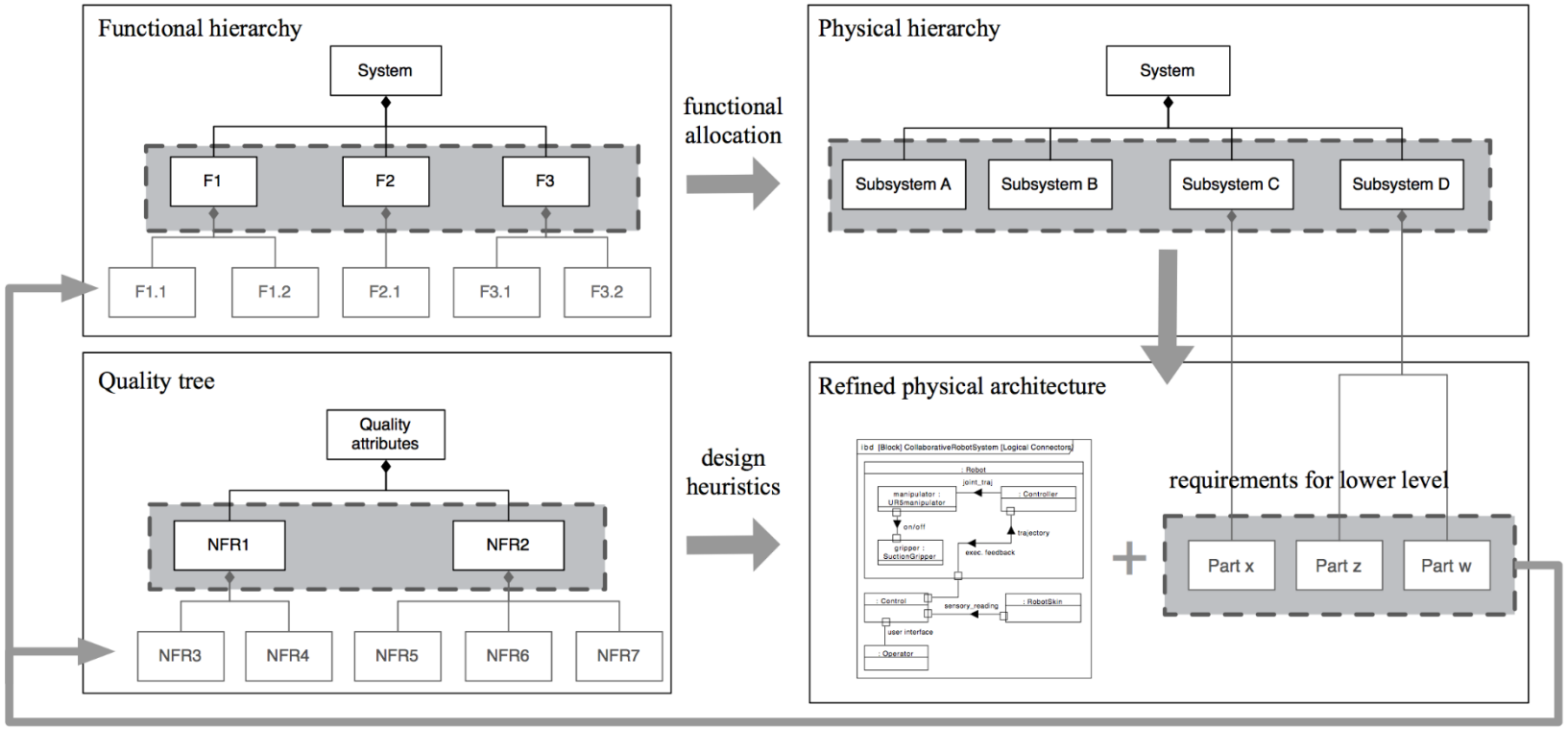
- The ISE&PPOOA views of the system can be envisioned as the assembly of the **three dimensions** used for software-intensive systems design
- Each **dimension has associated project deliverables**, mainly SysML diagrams but complemented with textual and tabular representations, for example the N square chart

The three dimensions of ISE&PPOOA

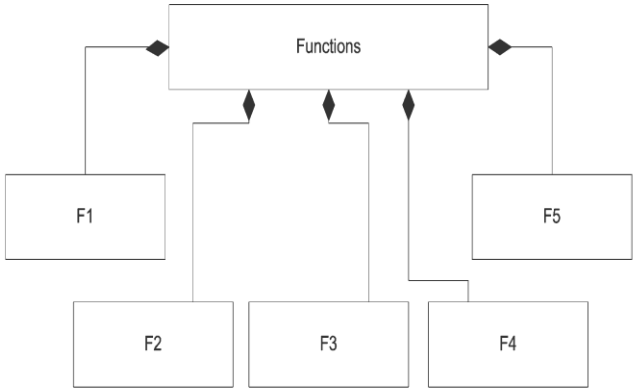


Tip: The importance of hierarchies

Hierarchies are important to meet the breadth first approach in which scope is covered comprehensively before fidelity is addressed



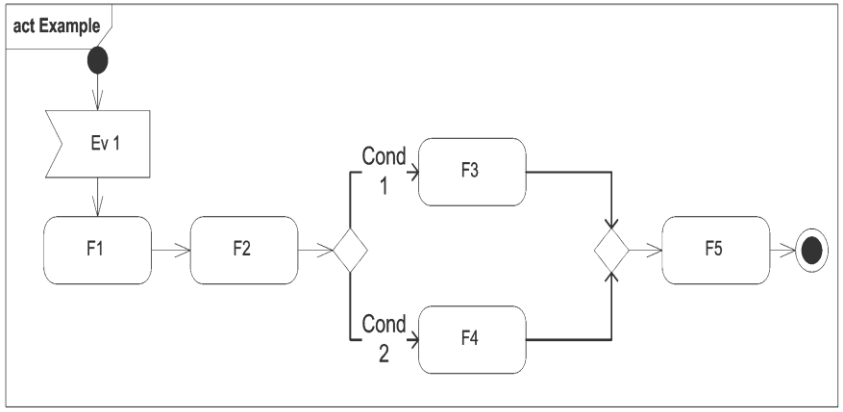
Functional architecture of the system



Functional hierarchy

item1				item 6	
F1	item 2				
	F2	item 3			
		F3	item 4		
			F4	item 5	
				F5	item 7

N square chart

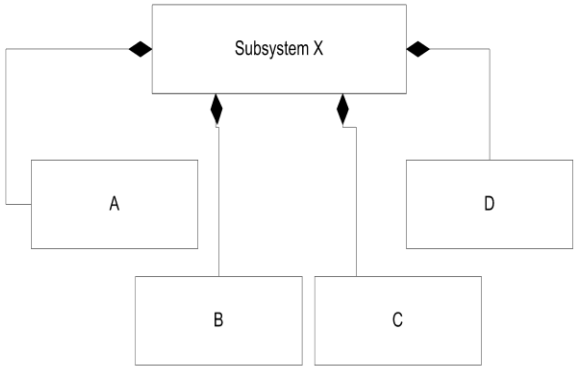


Functional flow

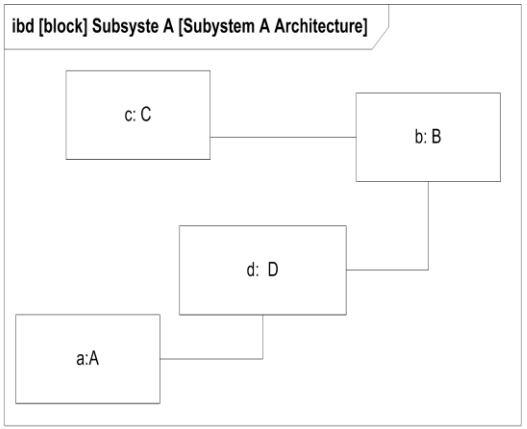
Functions description
(text and tabular format)

Functions description

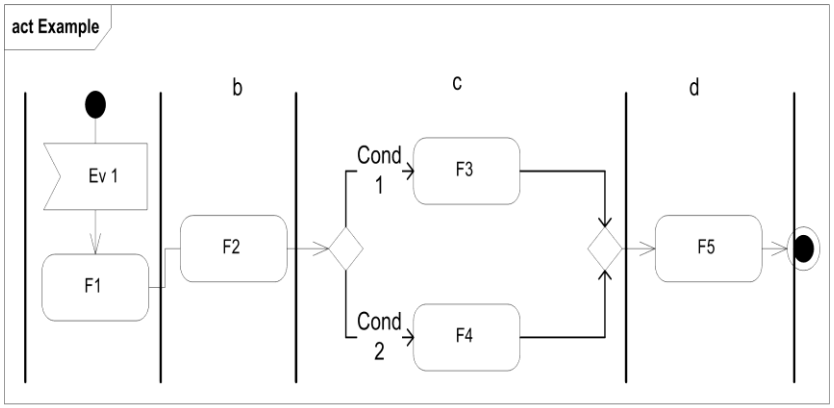
Physical architecture of the system



Physical hierarchy



IBD



Allocated Functional Flow

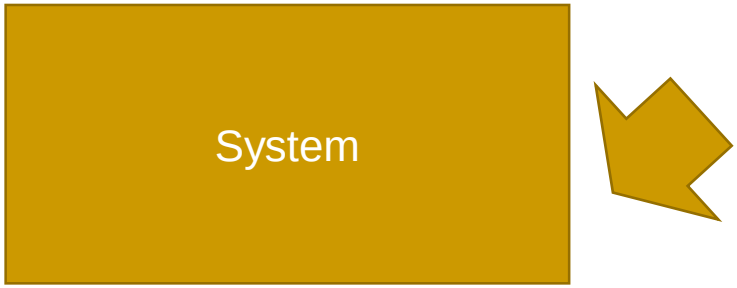
Parts Descriptions
(text and tabular format)

5. ISE&PPOOA process

Which are the methodological process steps, deliverables and participants

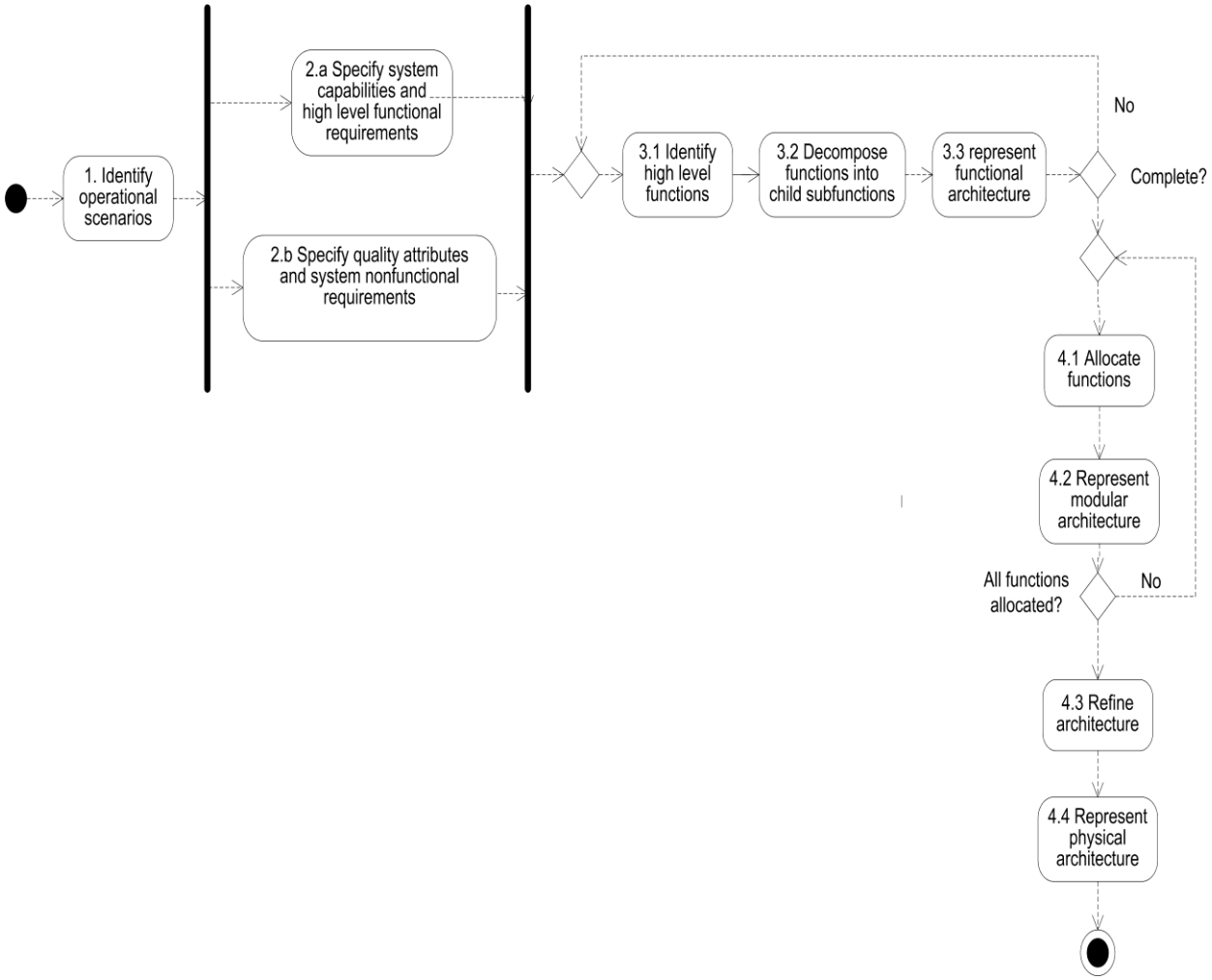
The out-in approach is the foundation of the ISE&PPOOA process

1. How the system operates **in a context**
2. Identify and specify the **operational needs** of the diverse actors
3. Identify and model the system **external interfaces**
4. Identify and model the system **functions**
5. Specify ***system requirements***
6. Identify and model **subsystems** and their interfaces



Out-In Approach

ISE&PPOOA process



Step 1. Identify operational scenarios

- **Goal***:* Identify the operational context of the system and describe its operational scenarios for the different modes of operation

- **Deliverable***:* The system intended behaviors are described by the operational scenarios, where additionally to the preconditions, postconditions, and steps of each scenario, the needs are identified. These needs are the inputs for the later identification of the system capabilities and quality attributes in the following steps of the subprocess

- **Participants***:* Operational concept experts, requirements engineers, future system users, and other project stakeholders

Step 2a. Specify system capabilities and high-level functional requirements

- **Goal***:* Transform scenarios and needs into a set of system capabilities and high-level system requirements

- **Deliverable***:* The deliverable is the representation of capabilities with a hierarchical decomposition using the block definition diagram of SysML. System functional requirements specified in natural language but based on the hierarchical decomposition are obtained. System measures of effectiveness (MoEs) can be identified in this step

- **Participants***:* Domain experts supported by systems engineers, customers, and other project stakeholders

Step 2b. Specify quality attributes and system nonfunctional requirements

- **Goal***:* Transform scenarios and needs into a set of quality attributes for example reliability, availability, security, and others including the associated nonfunctional requirements

- **Deliverable***:* In decomposing a nonfunctional requirement, the systems engineer can chose to decompose its type (security, reliability, etc.) based on a selected quality framework, or its topic, considering if it applies to the whole system or one of its parts. It is possible and should be taken into account that other nonfunctional requirements may be affected either positively or negatively at the same time

- **Participants***:* Systems engineers with the collaboration of customers and experts in some quality domains for example security or safety

Step 3.1. Identify high-level functions

- **Goal***:* Find the top-level functions of the system. Top-level functions are those functions used to organize the system functionality. They may be identified by previous knowledge of systems with similar capabilities or analyzing the main inputs and outputs of the system to be developed
- **Deliverable***:* The output is the top level of the functional hierarchy using a SysML block definition diagram
- **Participants***:* Systems engineers with the collaboration of customers and users

Step 3.2. Decompose functions into child subfunctions

- **Goal***:* Build the functional hierarchy identifying the subfunctions of each high-level function of the system and continuing until the granularity of the subfunctions or children functions is appropriate for the allocation step (step 4.1). Therefore, step3.2 is part of an iterative process of building the functional hierarchy
- **Deliverable***:* The deliverable is the functional hierarchy using a SysML block definition diagram
- **Participants***:* Systems engineers with the collaboration of customers and users

Step 3.3. Represent functional architecture

- **Goal***:* Represent the functional architecture identifying the functional hierarchy, functional interfaces and functional flows or behavior
- **Deliverable***:* The deliverable is the functional architecture representing the functional hierarchy using a SysML block definition diagram. N square chart is used for functional interfaces description where the main functional interfaces are represented. Activity diagrams are used for representing the main system functional flows of behavior. A textual description of each system function is provided as well
- **Participants***:* Systems engineers with the collaboration of customers and users

Step 4.1. Allocate functions

- **Goal***:* Identify the building elements of the solution that implement each of the functions represented in step 3.3
- **Deliverable***:* The building elements or physical components of the solution are identified. The heuristics of modularity described in Chapter 6 are applied here, choosing the solution elements so that they are as independent as possible; that is, solution elements with low external complexity (low coupling) and high internal complexity (high functional cohesion).
- **Participants***:* Systems architects

Step 4.2. Represent modular architecture

- **Goal***:* The selection of the solution is based mainly on the clustering of functions to obtain a modular architecture

- **Deliverable***:* The deliverable is the first version of the physical architecture. The modular architecture is represented by the system decomposition into subsystems and parts using a SysML block definition diagram. This diagram is complemented with SysML internal block diagrams representing the system physical blocks with either logical or physical connectors for each subsystem identified and activity and state diagrams for behavioral description as needed. A textual description of the system blocks is provided as well

- **Participants***:* Systems architects

Step 4.3. Refine the architecture

- **Goal***:* The modular architecture of the previous step is refined applying trade-off and considering the implementation of nonfunctional or quality attributes requirements. Design heuristics are used for taking into account the system nonfunctional requirements

- **Deliverable***:* Each used heuristic may be documented. The project/company heuristics collection is an asset that will be updated with the experience of the projects closed

Trade studies may be performed to select the preferred physical architecture that optimizes the measures of effectiveness that may be defined in step 2a

- **Participants***:* Systems architects with the collaboration of customers and experts in some quality domains; for example reliability, maintainability, security or safety.

Tip: Refining the system architecture is performed iteratively

Select a nonfunctional requirement to implement, **select the heuristic to implement this nonfunctional requirement**, **solve the conflicts** with other nonfunctional requirements and implement the heuristic **modifying the existing system architecture**

6. PPOOA (Pipelines of Processes in Object Oriented Architectures)

PPOOA a software architecture framework and software architecting process

PPOOA

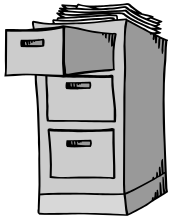
- **PPOOA** is an **architectural framework** (“software architecting process”+”building elements”) for **real-time software intensive systems**.
- The main building elements of PPOOA framework are software **components** and **coordination mechanisms**.
- The main software components are the “domain class” and the “process” implementing an independent thread of control.
- Coordination mechanisms are the building elements supporting synchronization and communication.
- A PPOOA “process component” may be implemented using an Ada task, java thread or by the light processes supported by the real-time operating system used.

Software components and coordination mechanisms provided by the PPOOA

Components

Controller :
Manages external events

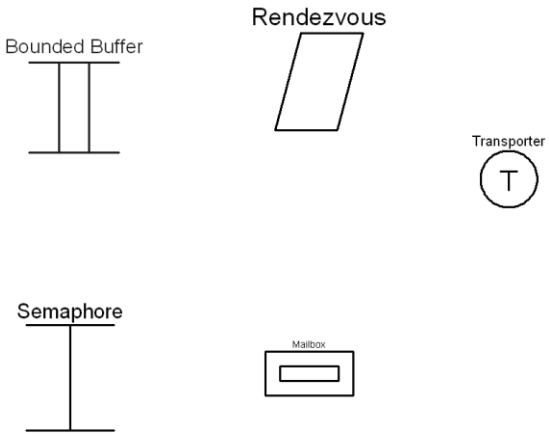
Domain component/ Algorithmic component:
Performs operations



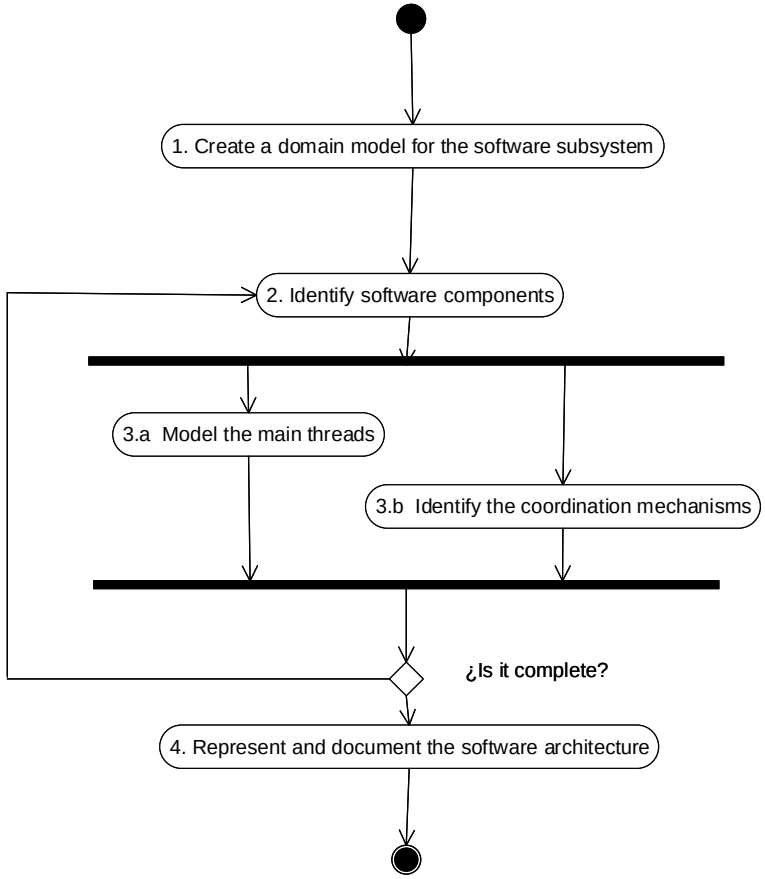
Structure:
Maintains relations between objects

Process:
Coordinates work to others

Coordination Mechanisms:
Synchronization+ communication



PPOOA software architecting process



Step 1. Create the domain model

- **Goal**: The domain model is the bridge between the system and the software dimensions
- **Deliverable**: A domain model is described using more formalism than textual descriptions, for example UML class diagrams. The domain model is the result of a **domain analysis**

  Classes in the domain model represent concepts or terms derived from use cases and the functionalities to be implemented by software. Classes are abstractions of physical and non-physical entities for example **things**, **events, roles, descriptions**.The domain model represents the **relations** between classes mainly generalization, specialization, “is part of”, “is member of” and associations
- **Participants**: Systems engineers and software engineers

Step 2. Identify software components

- **Goal***:* Create the initial set of components of the software subsystem. A software component is a computation entity that performs an assigned responsibility, provides interfaces to other components, and may require some interfaces from other components

- **Deliverable**: Set of selected software components. PPOOA provides criteria and guidelines for the selection of the software components most appropriate to implement a class in the domain model

- **Participants**: Software engineers

Step 3. Model the main threads of execution and identify coordination mechanisms

- **Goal***:* Identify the execution thread and how they coordinate

- **Deliverable**: Execution threads and coordination mechanisms. The main execution threads are determined by the periodic processes and the software responses to events. PPOOA provides criteria and guidelines for that and for the selection of coordination mechanisms

- **Participants**: Software engineers

Step 4. Represent and document the software architecture

- **Goal***:* Describe the structural and behavioral views of the software subsystem architecture
- **Deliverable**: The static structure of a system can reveal what it contains and how its elements are related, but it does not explain how these elements cooperate to provide the software subsystem functionality. PPOOA uses an UML class diagram but adapted to its component's types and coordination mechanisms
  To represent software behavior, PPOOA proposes to model it using UML/SysML activity diagrams as used in the systems engineering subprocess of ISE&PPOOA. Each event response is represented by a causal flow of activities (CFA) modeled as an activity diagram. A CFA is a cause-effect chain of activities that cuts across different building elements of the software architecture
- **Participants***:* Software engineers and systems engineers

7. The domain model

The bridge between the system and the software architectures

Why software is different and how we deal with software architecture?

- A mere physical system (traditional SE) and a software system are different
  - Software components can be created and destroyed
  - **Software does not meet Physics laws** of mass, energy and momentum conservation
  - **Software components are abstractions of physical and non-physical entities** for example events, roles, descriptions…
- For complex and software intensive systems we need "**best practices+methodology"** that bridge the system and software semantic gap. Here we have proposed: **ISE+ *domain driven modeling* + PPOOA**

Domain model

- A domain model is described using more formalism than textual descriptions, for example UML class diagrams.
- The domain model is the result of a **domain analysis (Evans, 2003)**
  - **Classes** in the domain model represent **concepts or terms** derived from use cases and the functionalities to be implemented by software.
  - So, classes are abstractions of physical and non-physical entities for example **things**, **events, roles, descriptions**…
  - The domain model represents the **relations** between classes mainly generalization, specialization, “is part of”, “is member of” and associations.
- A domain model is an essential input when the subsystem is shaped in software architecture, design and implementation

Entities, Value objects and Aggregates in the domain model (Evans, 2004)(Vernon, 2016)

- An **entity** is a domain model object defined primarily by its identity
- **Value objects** are domain model objects without identity that describe things
- An **aggregate** is a cluster of associated objects treated as a unit when a data changes
- The **root class of each aggregate** owns all the other classes clustered inside it
- Each aggregate is designed to be a **change consistency boundary** so protect business invariants inside aggregate. Aggregates should **be as small as possible**

CRC cards (Beck and Cunningham, 1989)

- **CRC cards** are index cards, one for each domain model class, upon which the responsibilities of the class are briefly documented and a list of classes collaborated with to achieve those responsibilities

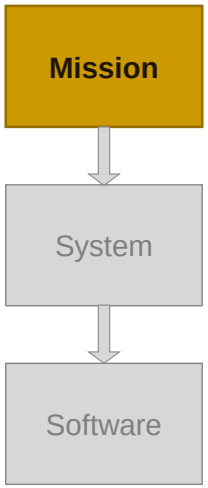
Class name Software class identity
Class responsibilities • What the class knows? • What the class does?
Class collaboration • Other classes the class is collaborating to achieve its responsibilities

Example

Collaborative robot developed by Carlos Hernandez team at the Delft TU

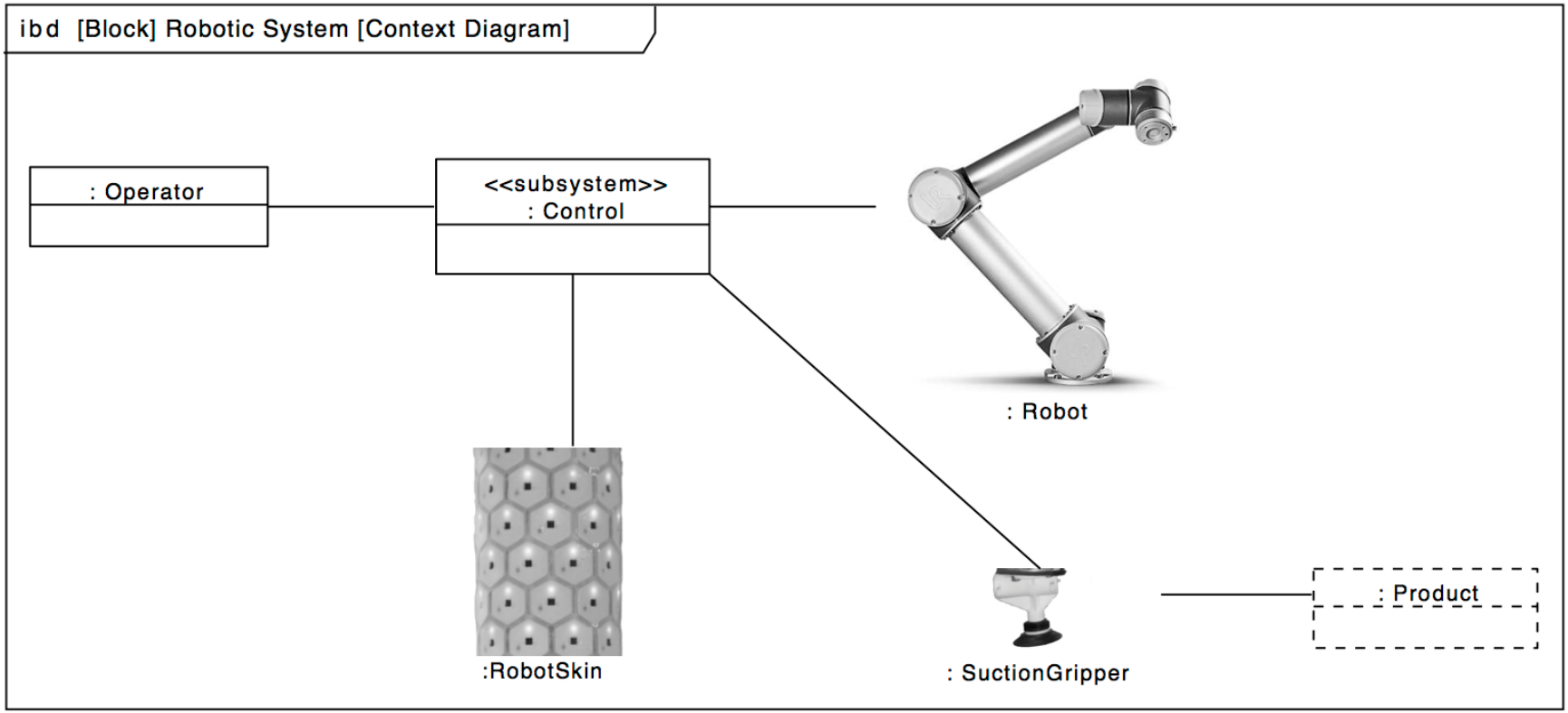
Example: Collaborative Robot

Collaborative robot- **Mission dimension**



Context, scenarios, needs and capabilities

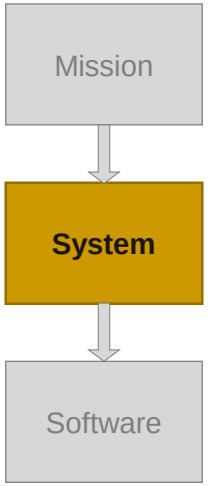
Context diagram of the collaborative robot



Operational Scenarios

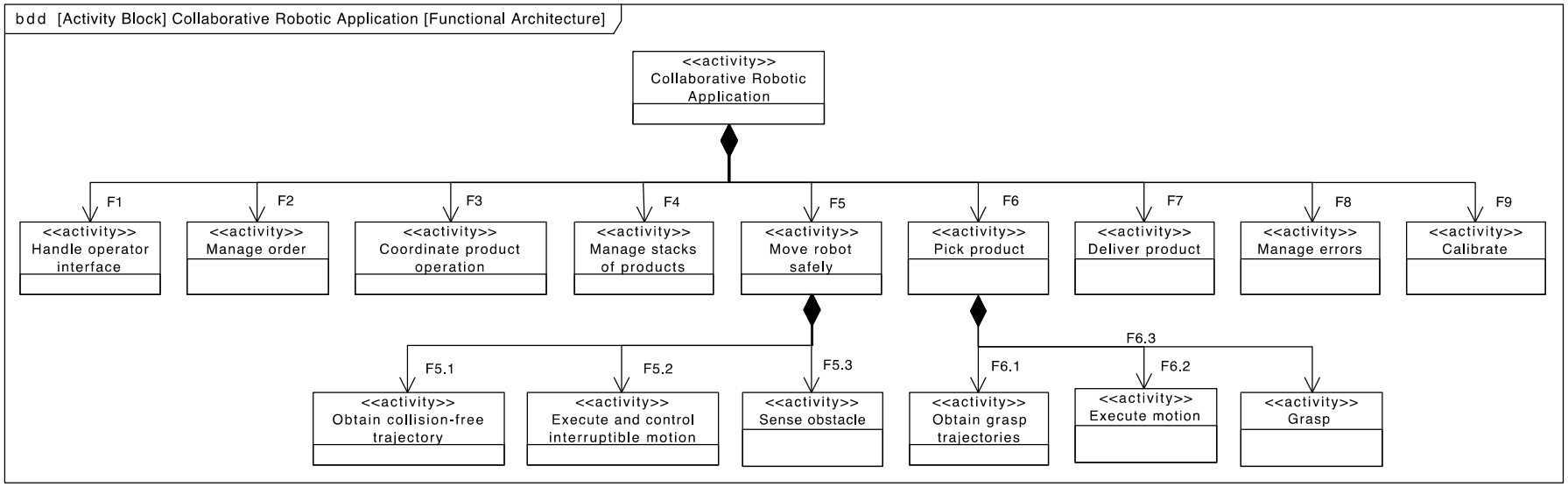
Deployment	S1 System configuration and calibration	The operator installs the robot arm and the product containers in the frames, and moves the robot arm to the reference pose for each container, so all end-effector poses are calibrated off-line for the robot to be able to reach all the products.
	S2 Configure new product	The operator adds a stack of a new product type, possibly of different dimensions (within a limited variation range), or replaces an existing type with the new one, and informs the system through the operator interface. The system is capable of processing order that include the new product type.
Operation	S3 System start	The operator powers up all subsystems. The robot arm calibrates its joints.
	S4 Manage order	The system receives an order request consisting of several products of one or more product types, and delivers the order in the bin by autonomously handling the products.
	S5 Pick product	The robot moves the gripper to the container of the next product in the order, grasp an available product, and retreats from the container holding it.
	S6 Deliver product	The robot delivers the product it is holding with the gripper in the delivery bin.
	S7 Refill product	When one of the product stacks is empty, the robot notifies the operator and moves to a configuration to allow the operator to replace the container with one filled with products.
	S8 Shut down	The operator shutdowns the system through the user interface. As a result, the status of the containers is stored in memory and the robot arm moves to the stand-by position
	S9 Emergency stop	Upon an anomalous behavior, the emergency stop of the robot arm is activated, stopping any current motion, and this is notified to the operator.

Collaborative robot- **System dimension**



Functional architecture, (requirements) and physical architecture

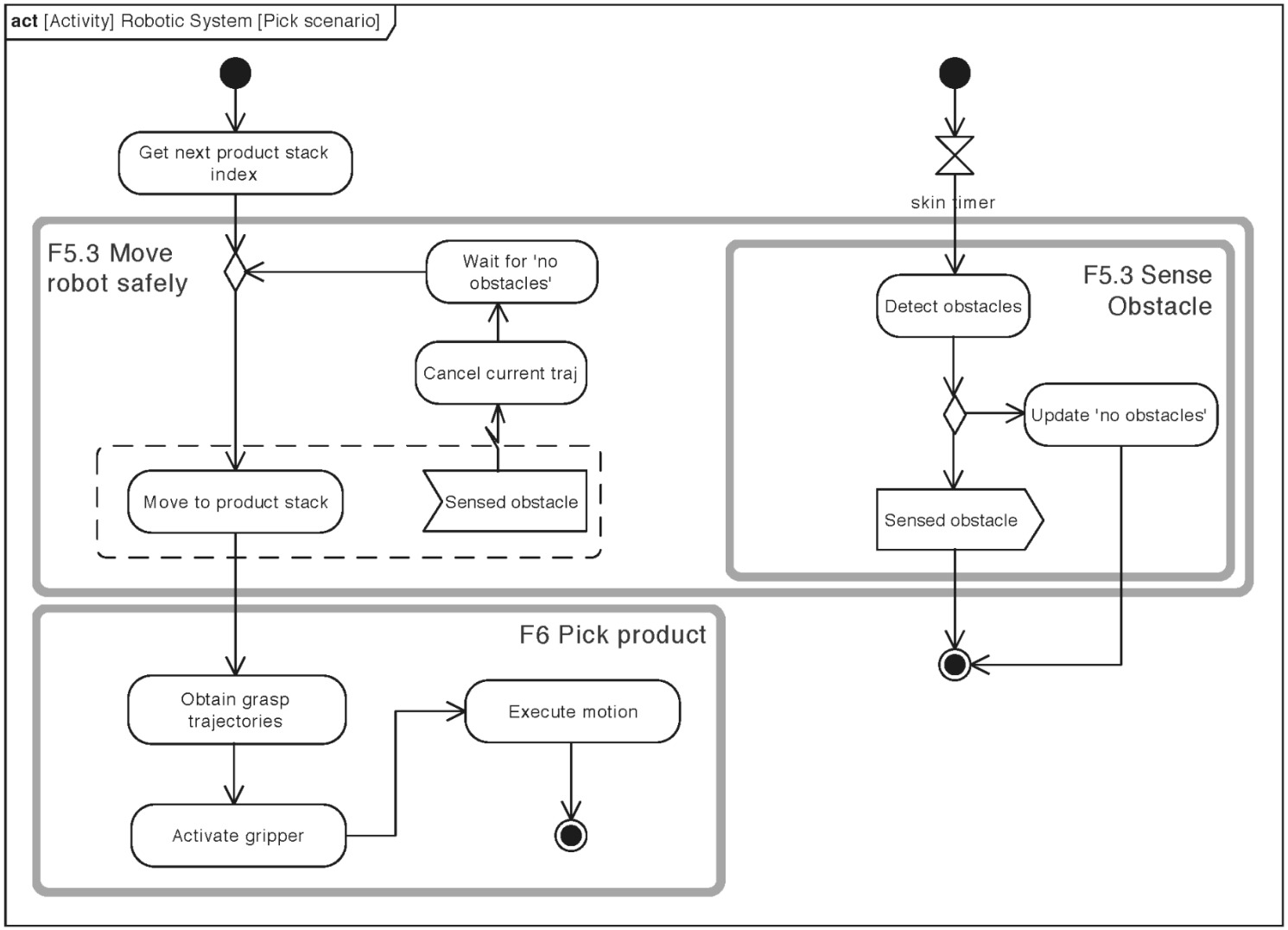
Collaborative robot functional hierarchy



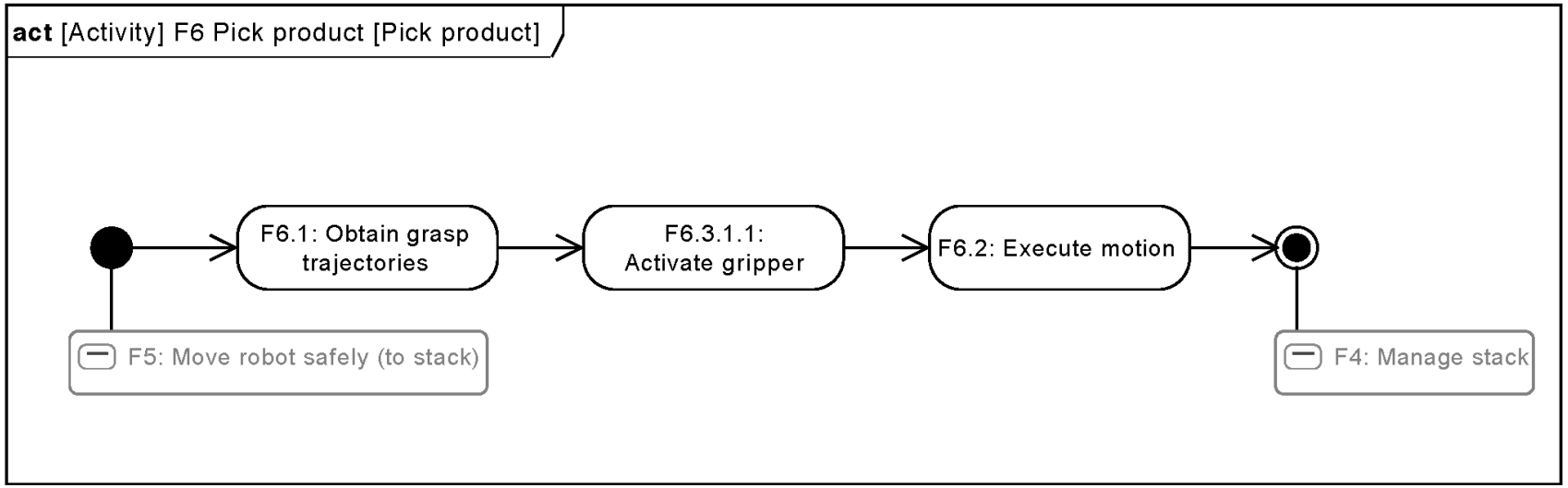
N^2 Chart of the Functional Interfaces for the scenario "Pick Product"

• product	• move to stack • cell 3D model	• request pick	• gripper ON		*INPUTS* *OUTPUTS*
F4 Manage stacks of products		• product index			
	F5 Move robot safely	• gripper over product stack			
		Obtain grasp trajectories (F6.1)		• trajectory	
			Activate gripper (F6.3.1.1)	• gripper suction	
				Execute motion (F6.2)	• product picked

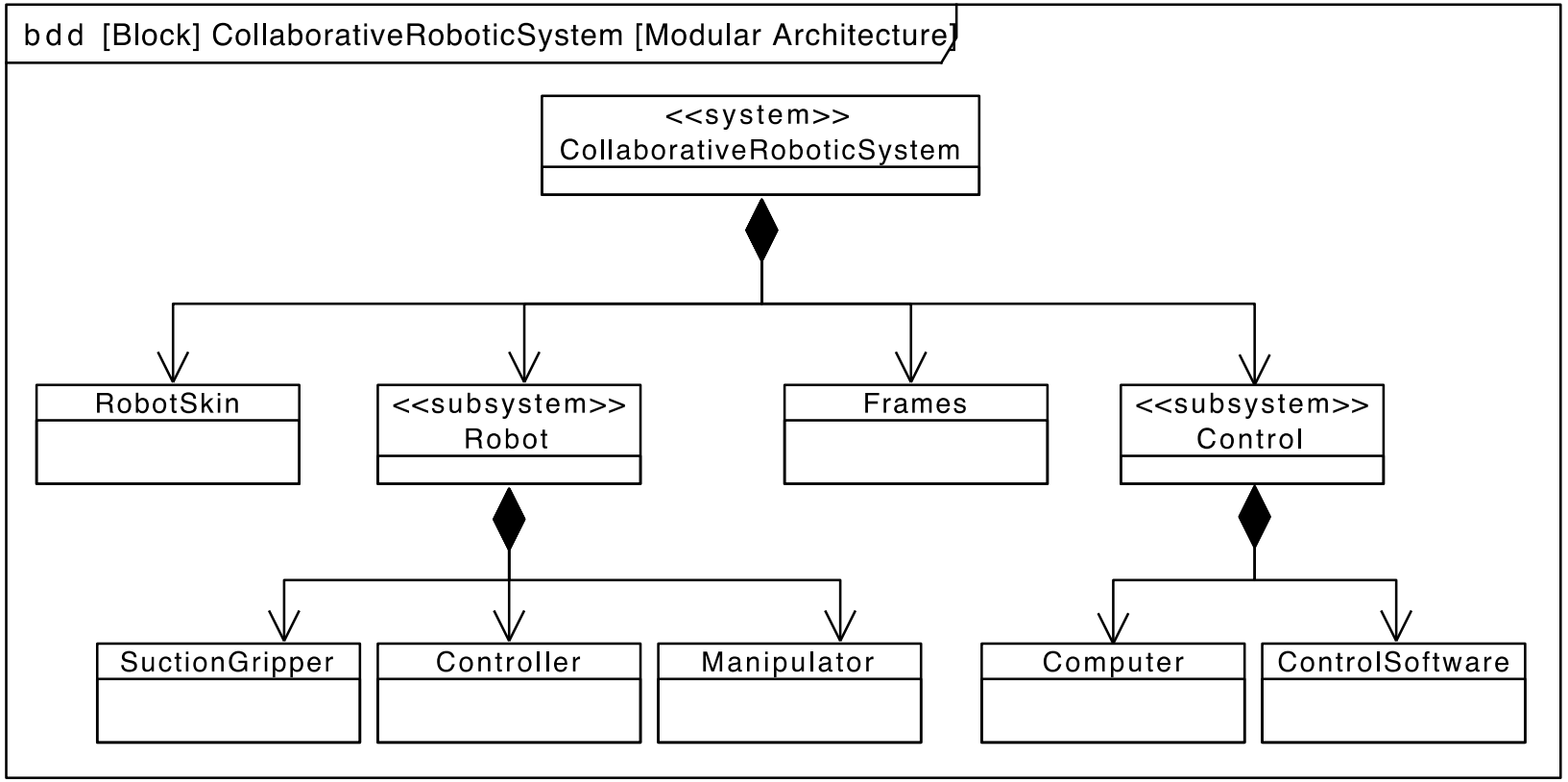
Collaborative robot main functional flows



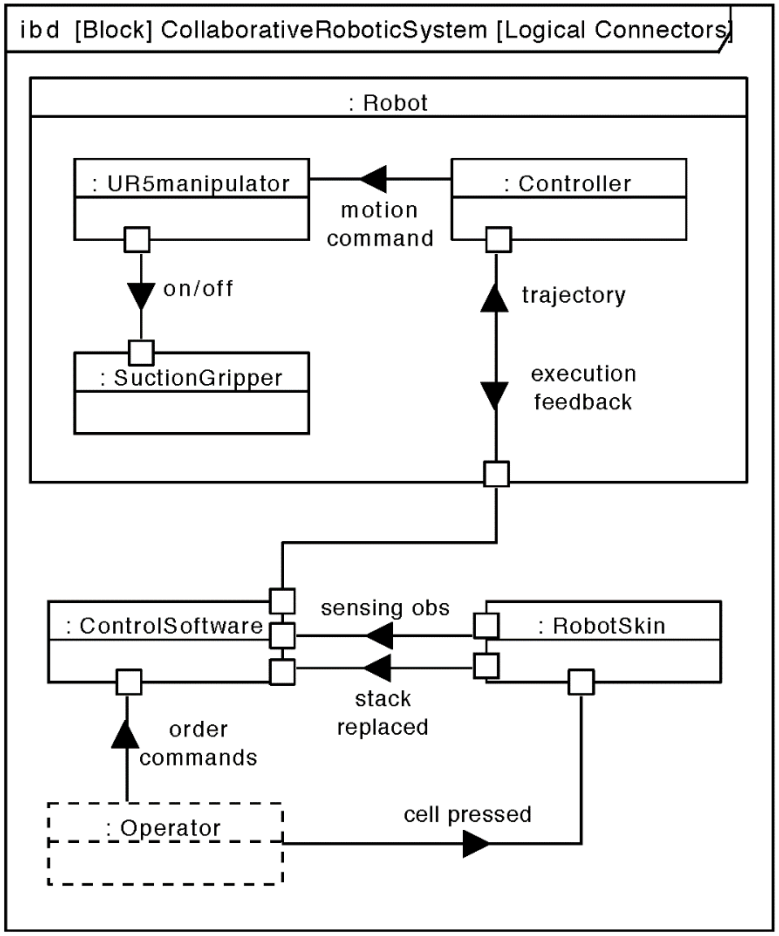
F6 pick product functional flow



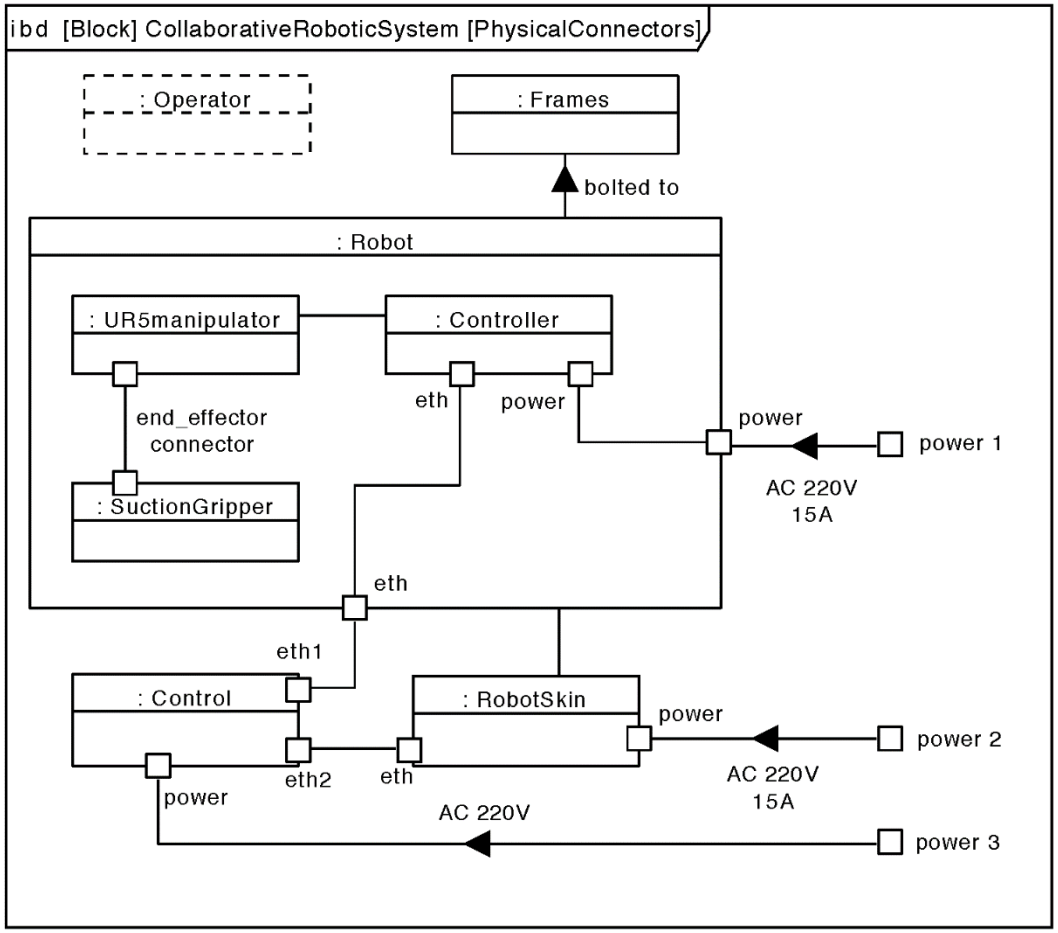
Physical architecture- SysML BDD



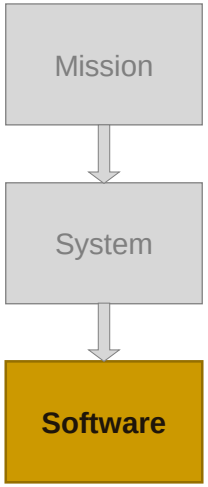
Physical architecture IBD with logical connections



Physical architecture IBD with physical connections

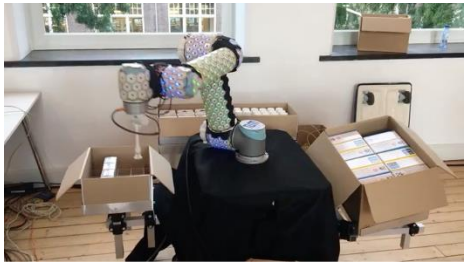


Collaborative robot- **Software dimension**

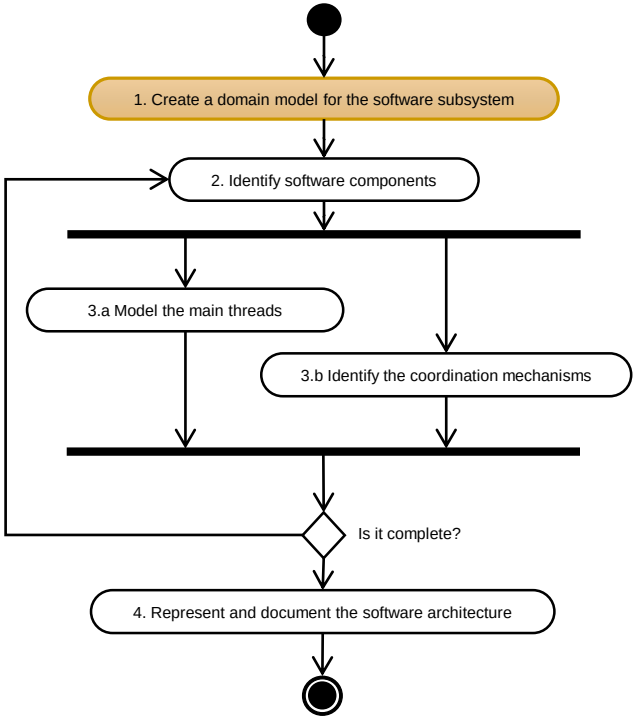


Domain model, software architecture structural view, software architecture behavioral view

Domain model



1. Identify the **functions to be** allocated to **software**
2. the **nouns** in the functions description are candidates to be **domain model classes** and the verbs are candidates to be represented as **associations between domain model classes**

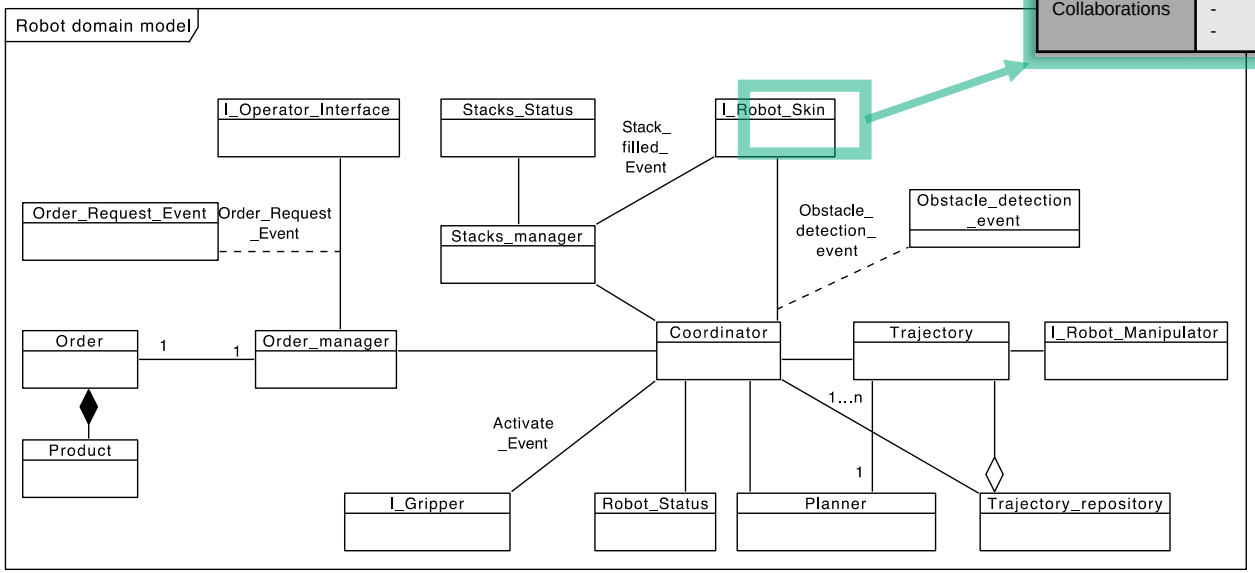


	RobotSkin	Gripper	Robot	Control	Frames
F1 Handle **Operator Interface**	X			X	
F2 Manage **Order**				X	
F3 Coordinate **product** operation				X	
F4 Manage **stacks** of products				X	
F5.1 Obtain collision-free **trajectory**			X	X	
F5.3.1 Sense distance to robot arm	X				
F5.2 Execute and control interruptible **trajectory**				X	
F5.3.2 Detect **obstacle**				X	
F6.1 Obtain grasp **trajectories**				X	
F6.2 Execute uninterruptible **trajectory**			X	X	
F6.3.1 Control **grasp**		X	X	X	
F7 Deliver **product**				X	
F8 Manage **errors**				X	X

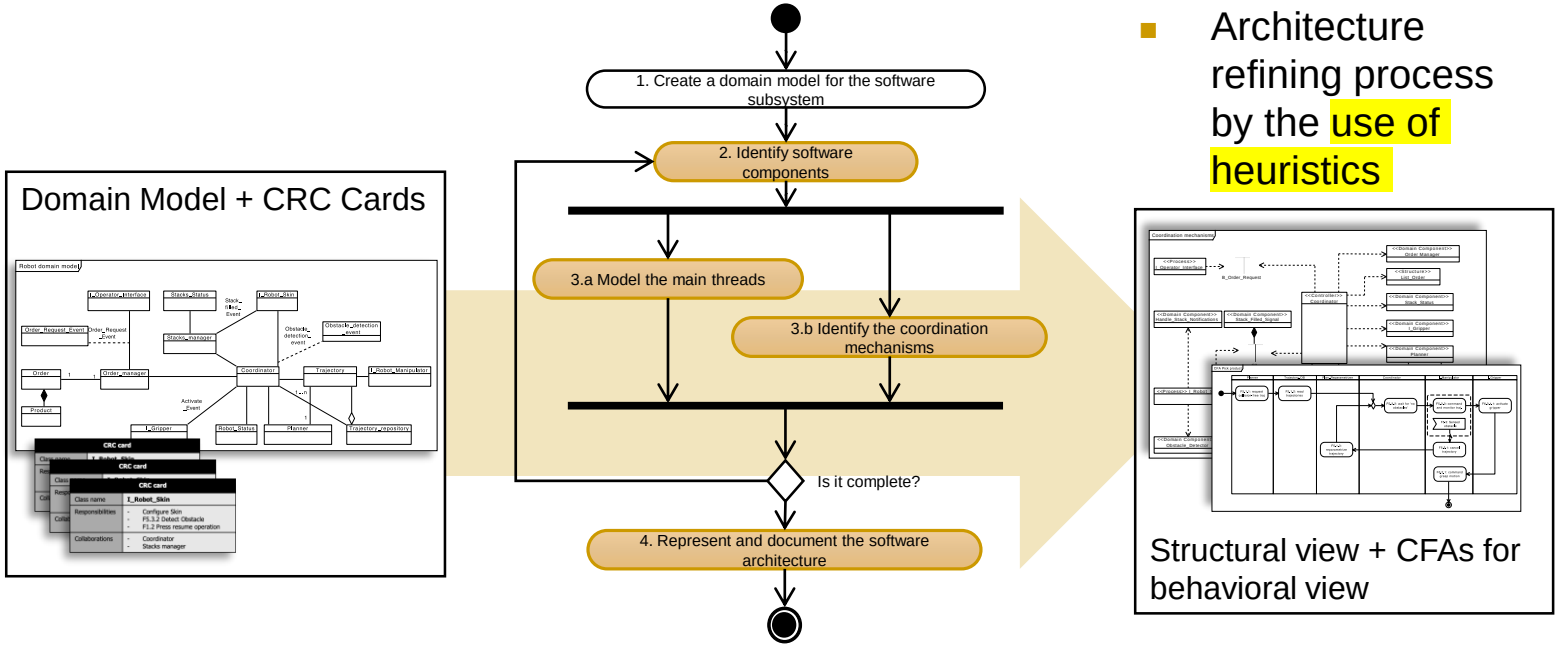
Domain model

- **Nouns** are **classes** in the domain model.
- Verbs are relations between the classes.
- **CRC cards** are used to define the responsibilities for each class of the domain model.

CRC card	
Class name	**I_Robot_Skin**
Responsibilities	- Configure Skin - F5.3.2 Detect Obstacle - F1.2 Press resume operation
Collaborations	- Coordinator - Stacks manager



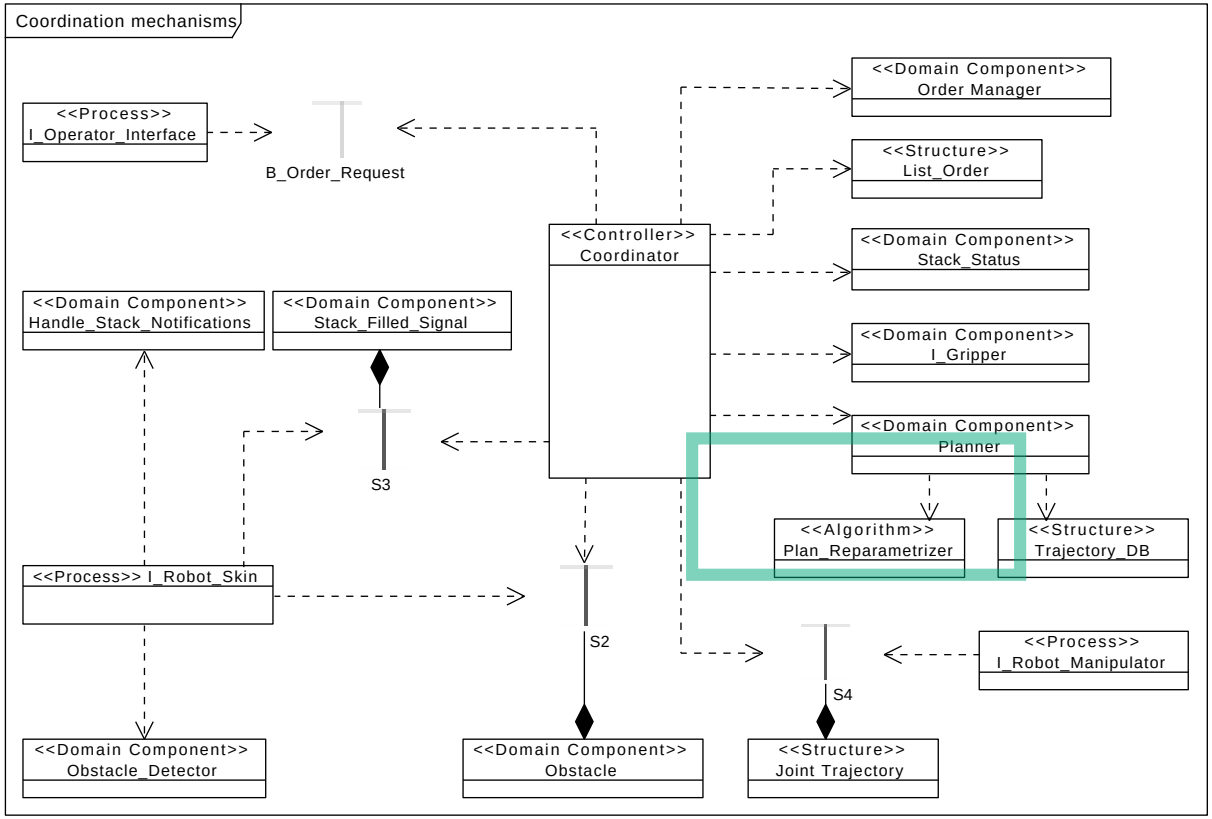
Software architecture is obtained from the domain model and using PPOOA building elements (UML-PPOOA profile)



- Architecture refining process by the use of heuristics

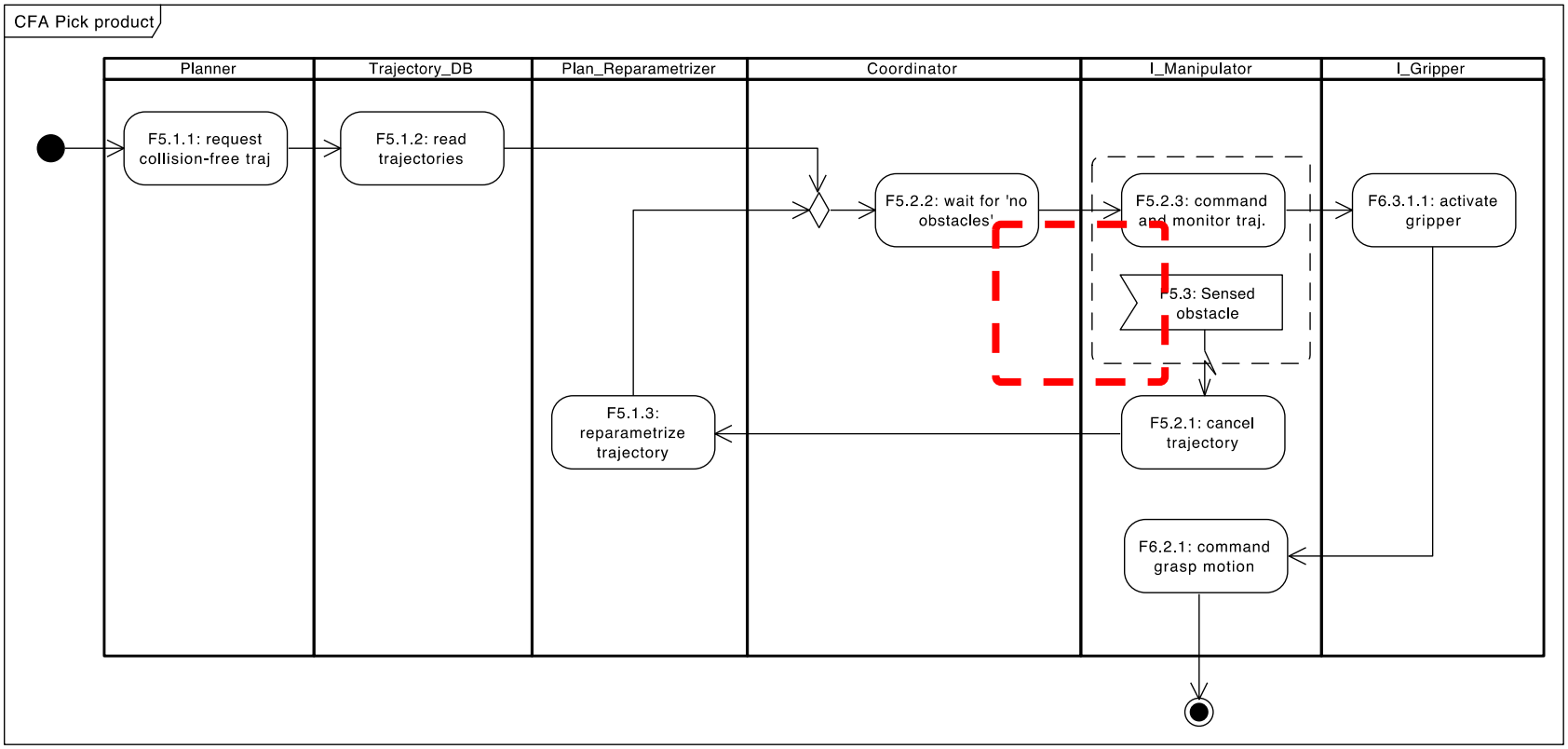
Software architecture structural view using UML extended with PPOOA building elements

- **Heuristic - Safety:** Avoid Non-deterministic Behavior
  "Solution based on a database with trajectories obtained offline is used to generate the motions to retrieve products."



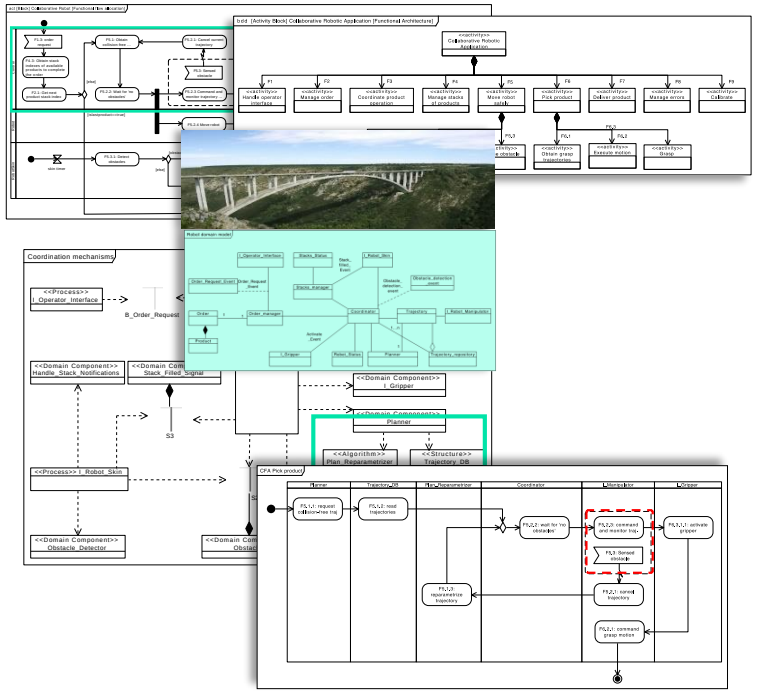
Software architecture behavioral View- UML activity diagrams with swimlanes

- **Heuristic - Safety:** Avoid Non-deterministic Behavior
  "Interruptible regions only during motion execution in the shared workspace"



To conclude

- ISE&PPOOA provides a method to formalize and **use system required functionality** to obtain the system and software **physical architectures** refined later **using heuristics to implement NFRs**
- **Domain model** allows to **bridge** the **system** and the **software** architectures
- Domain modeling is performed **based on the functional architecture**



ISE&PPOOA

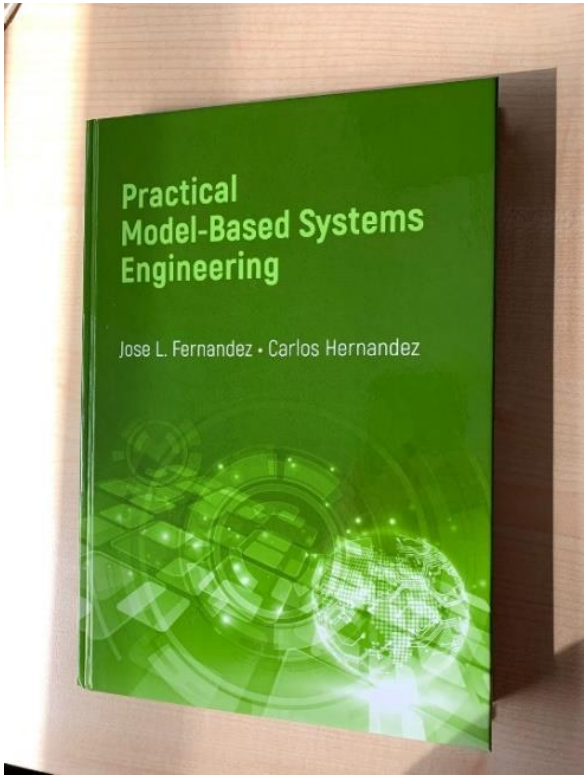
Methodology book with examples

Authors: José L. Fernández and Carlos Hernández

Artech House, Norwood, MA. July 31,2019

ISBN-13:978-1-63081-579-0

OMG wiki with links to resources: https://www.omgwiki.org/MBSE/doku.php?id=mbse:ppooa



Other references

- Estefan, J. A., Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev. B, INCOSE Technical Publication, Document No. INCOSE-TD-2007-003-01, San Diego, CA: June 10, 2008
- Topper, J.S. and C. Horner. Model-Based Systems Engineering in Support of Complex Systems Development. Johns Hopkins APL Technical Digest, Volume 32, Number 1, 2013
- Fernandez, J.L. and Martinez J.A. Applying Heuristics to Model the System Physical Architecture, PPI SyEN issue 103, August 2021
- D´Souza, D.F. and A.C. Wills. Objects, Components and Frameworks with UML. The Catalysis Approach. Addison Wesley, Reading MA 1998
- Fernandez J.L. and A. Monzon, Extending UML for Real-Time Component Based Architectures. 14th International Conference Software & Systems Engineering and their Applications, Paris, France, December 2001
- Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley 2004
- Vernon , V. Domain-Driven Design Distilled. Addison Wesley 2016.
- Beck, K. and A. Cunningham. A Laboratory for Teaching Object-Oriented Thinking. Proceedings of the OOPSLA'89, pp.1-6

Questions