ARCML

Session 06

Prepared by Prof. Dr. Christopher Shneider Cerqueira

Review

Last Lectures Review

1. Systems Engineering Basics
   - Stk – Lifecycle – CONOPs – Functions – Architecture – V&V
   - Classical Representations
2. Path to SE Digitalization
   - Meta(meta)models
   - Language
   - Methodologies

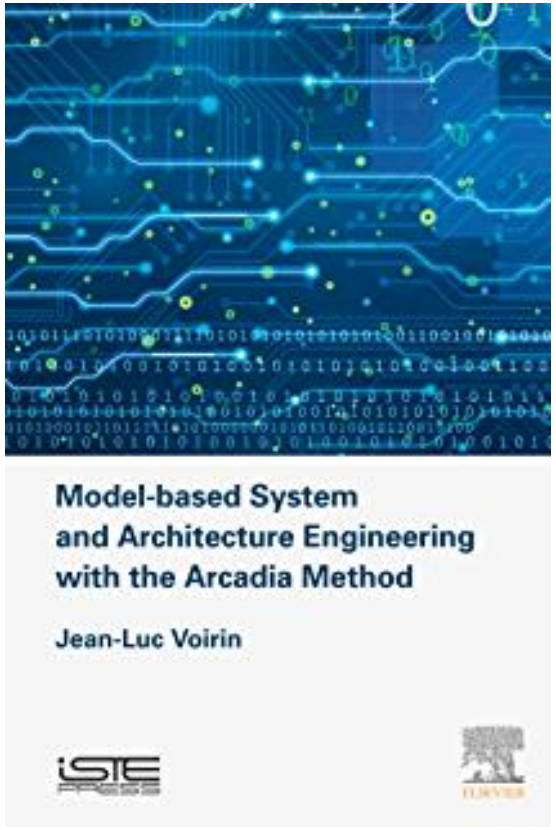
- OPM
  - Simple Structure <-> Behavior diagram
  - Two things / ~10 relations
  - Can map **almost** everything - great flexibility comes with great complexity
  - Ideal to model during “conversations”

Motivation

- Well… OPM is nice and has the capability to be the “the-facto” language…
- The other language competitor is the SysML.
  - SysML is not a simple language – has A LOT of details.
  - It is not really usable straight out of the box

SEMANA		TEORIA	INDIVIDUAL	PESO	GRUPO	PESO
1	1	Estrutura e Filosofia do Curso	AI-01 - Resumo Cap 1 - HB INCOSE	10%		
05-Aug	1	O que é Engenharia de Sistemas? INCOSE				
	1	Elementos da Eng Sis.				
	1	Introdução aos diagrams clássicos.				
2		* (Viagem ao EUA)	AI-02 - Leitura/Resumo paper sobre representações clássicas.	10%		
12-Aug						
3		* (Viagem ao EUA)	AI-03 - Exercício sobre arquitetura e escrita de requisitos.	10%		
19-Aug						
4	1	Metodologias de MBSE e uso de modelos.	AI-04 - Resumo Artigo de Metodologias	10%		
26-Aug	1	Revisão de UML-SysML.				
	1	OPM				
	1	Arcadia				
5	1	OPM	AI-05 - Lista de exercícios	10%		
02-Sep	1					
	1					
	1					
6	1	Blocos e Classes	AI-06 - Lista de Exercícios	20%		
09-Sep	1					
	1	Máquina de Estados				
	1					
7	1	Casos de Uso	AI-07 - Lista de Exercícios	20%		
16-Sep	1					
	1	Sequência				
	1					
8	1	Integração dos pontos de vistas em um	AI-08 - Resumo sobre Ciclo de Vida de Modelos	10%	AI-08 - Descrição e Contorno do Problema.	100%
23-Sep	1	Associação dos artefatos de SE com modelos				
	1	Análise Operacional				
	1					
				100%		**100%**
SEM						
30-Sep						

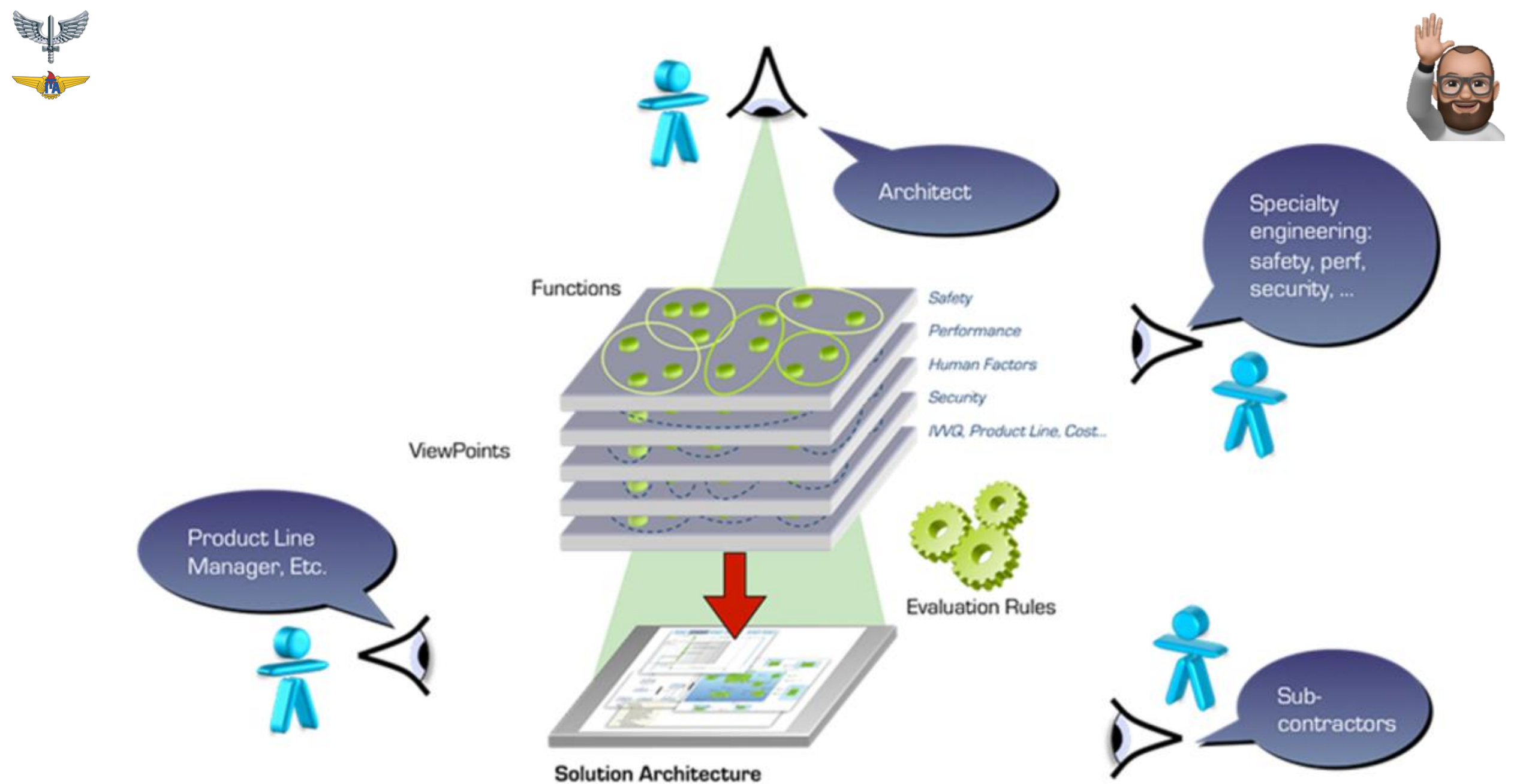
XP Z67-140 - ARCADIA



[Norme XP Z67-140 (afnor.org)]

Founding principles

- all of the engineering stakeholders share the **same methodology**, the **same information**, **the same description of the need** and the product in the form of a shared model;
- each **specialized type of engineering** (for example security, performance, cost and mass) **is formalized as a "viewpoint"** in relation to the requirements from which the proposed architecture is then verified;
- the rules for the **anticipated verification of the architecture** are established in order to verify the architecture as soon as possible;
- **co-engineering between the different levels** of engineering is supported by the joint elaboration of models, and the models of the different levels and specialties are deducted/validated/linked one to the other.


Architect
Functions
Safety
Performance
Human Factors
Security
IVVQ, Product Line, Cost...
ViewPoints
Specialty engineering: safety, perf, security, ...
Product Line Manager, Etc.
Evaluation Rules
Sub-contractors
Solution Architecture

Traditionally the SysML has 9 diagrams

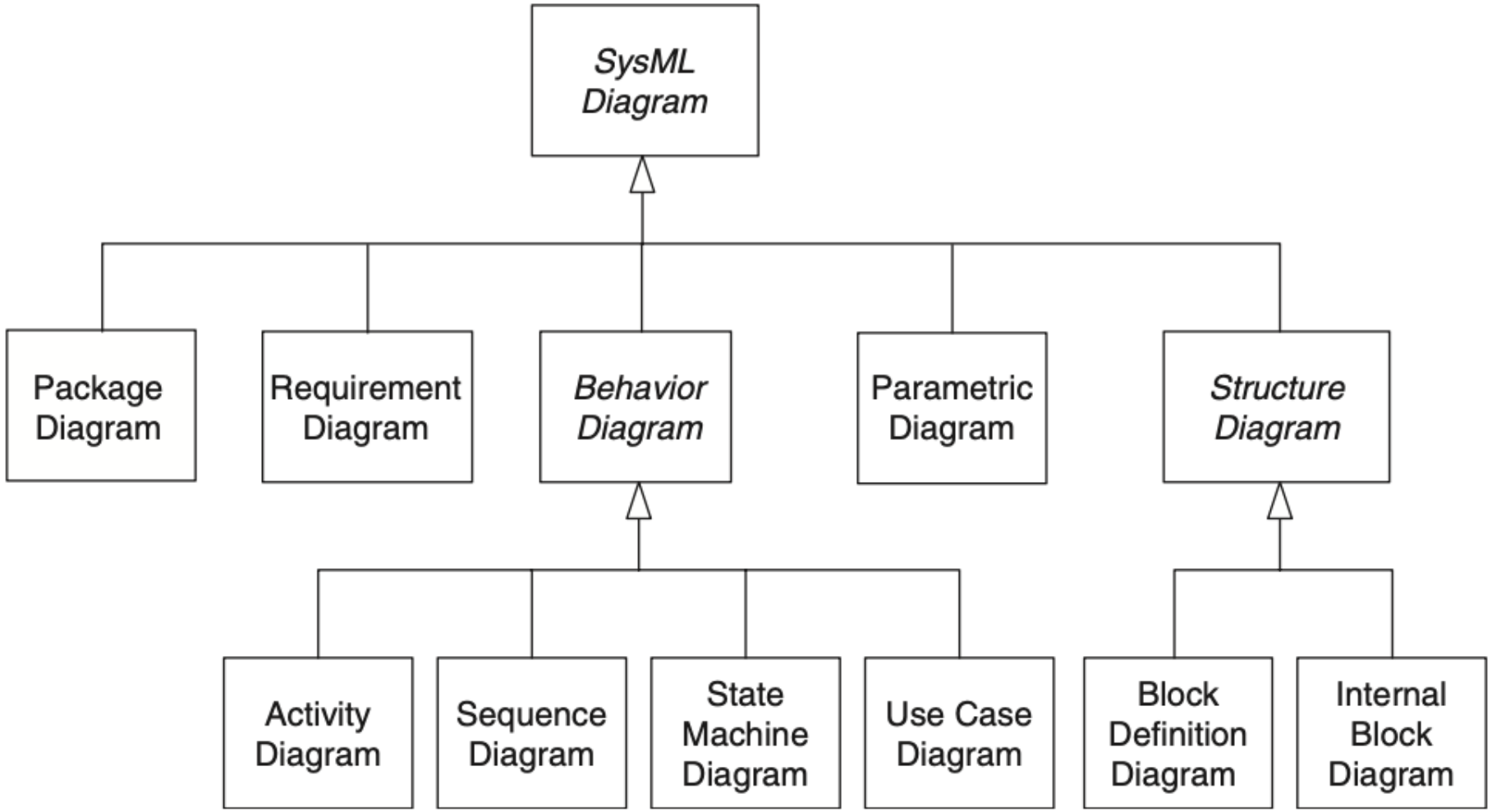


FIGURE 3.1

SysML diagram taxonomy.

To simplify... ARCADIA reduced to a few

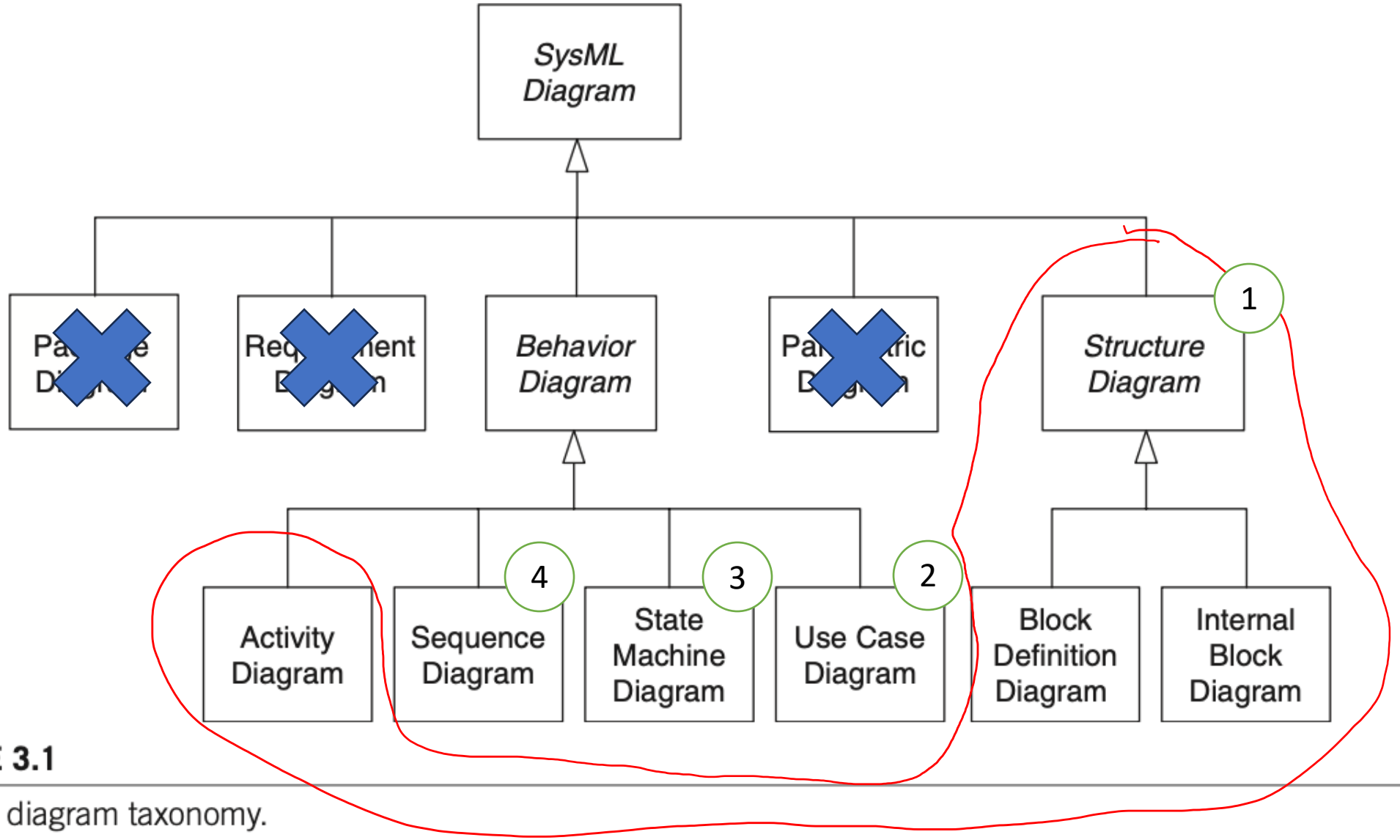
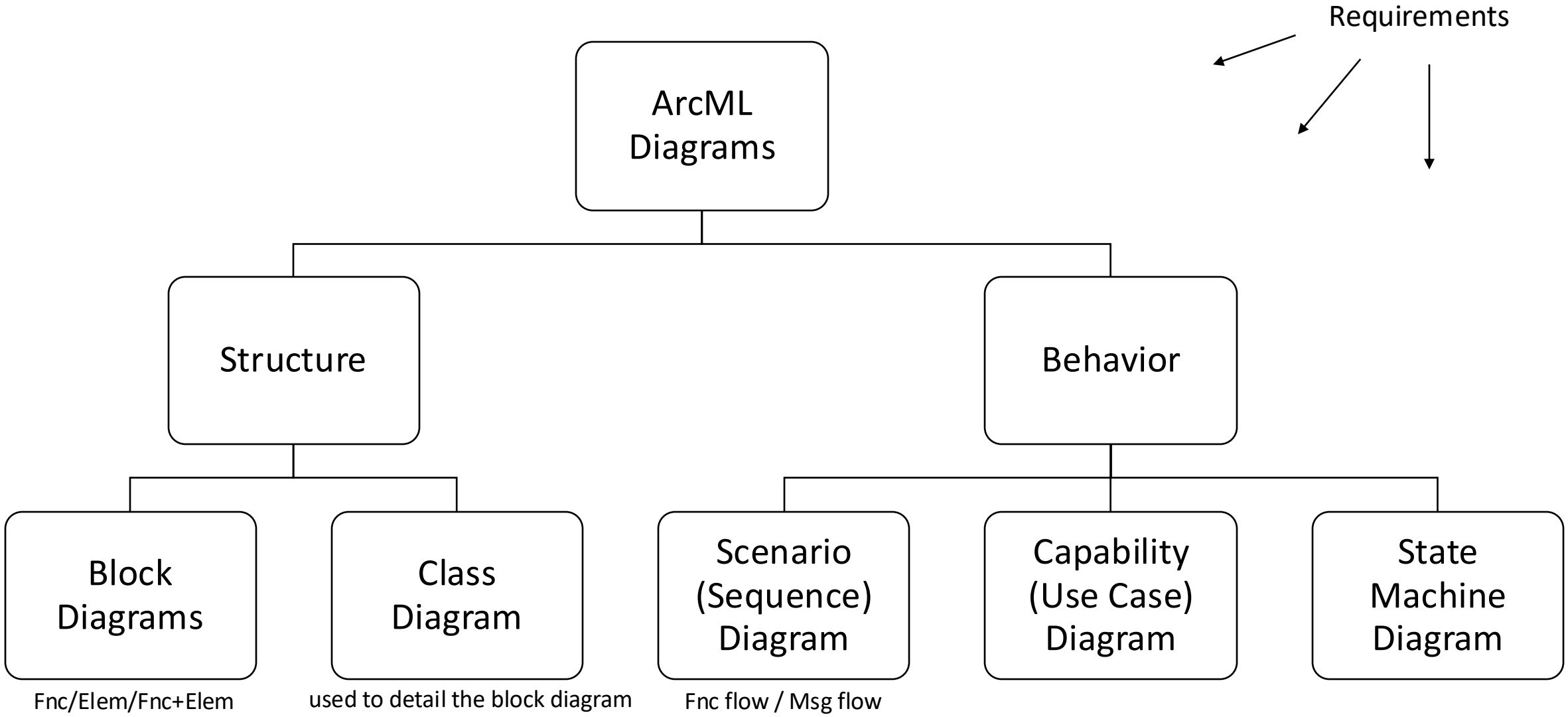


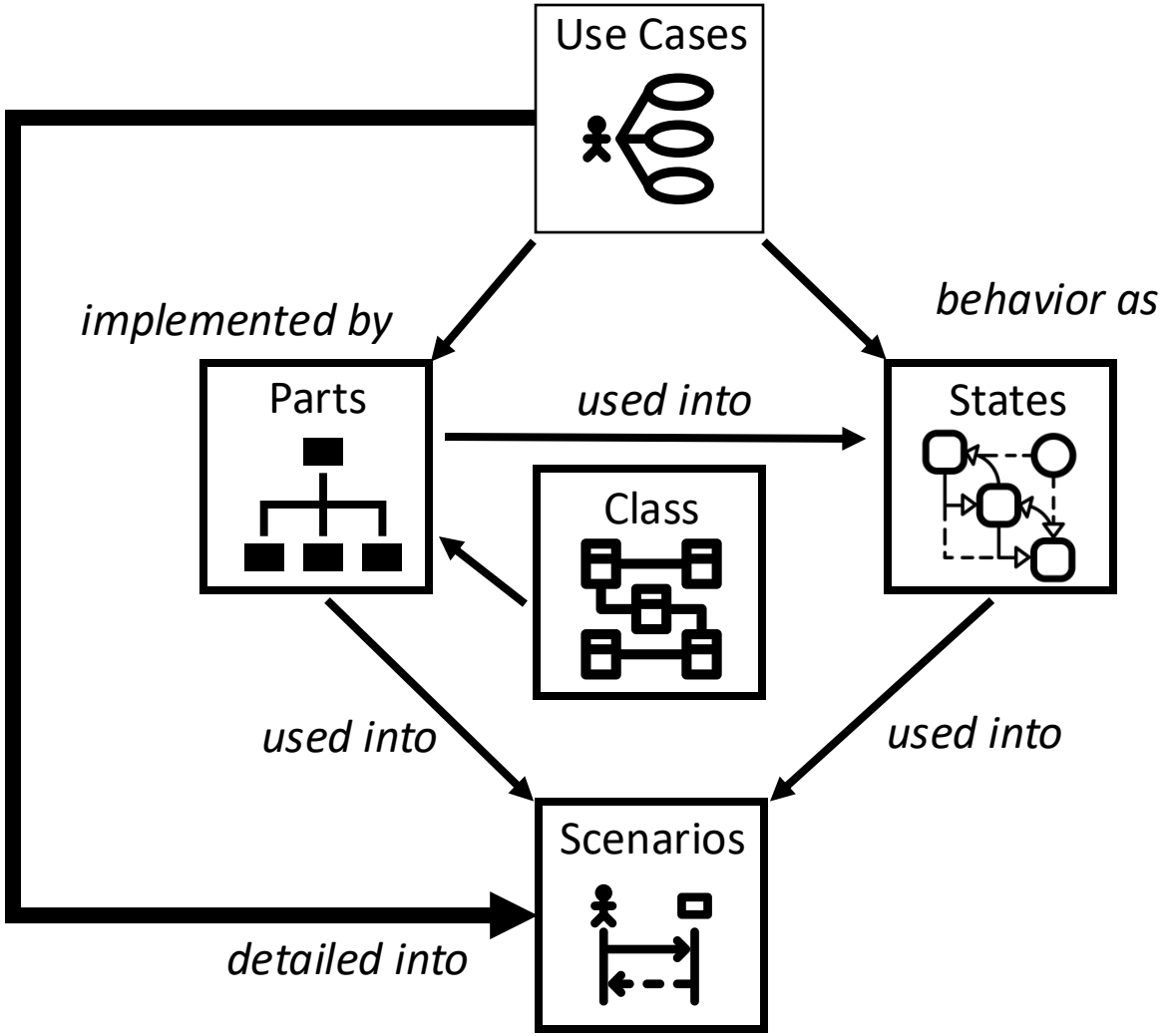
FIGURE 3.1

SysML diagram taxonomy.

New viewpoint taxonomy



Giving another view

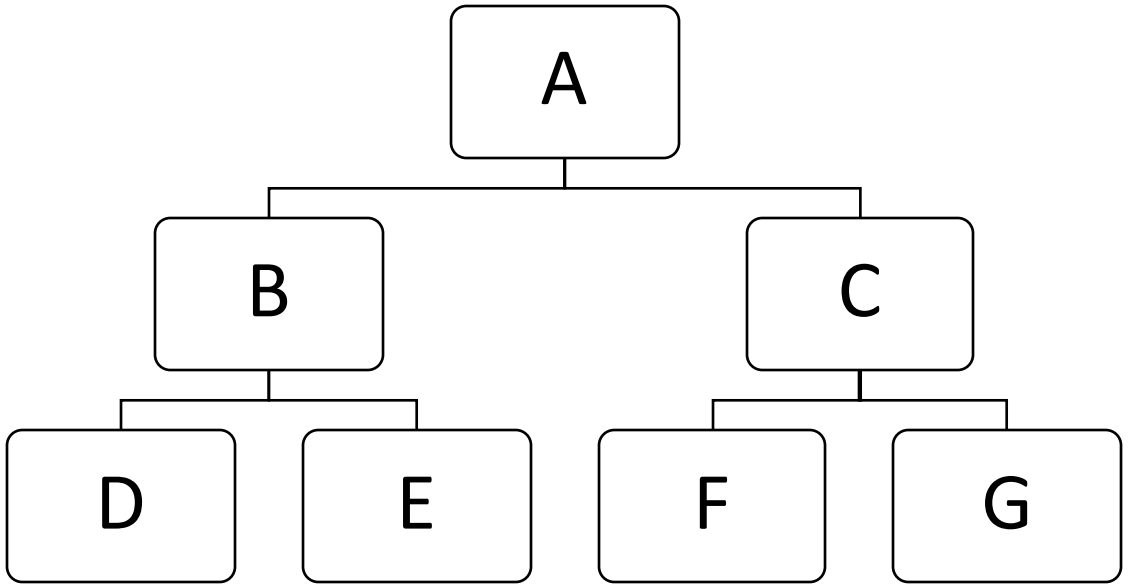


Common System Engineering Viewpoints

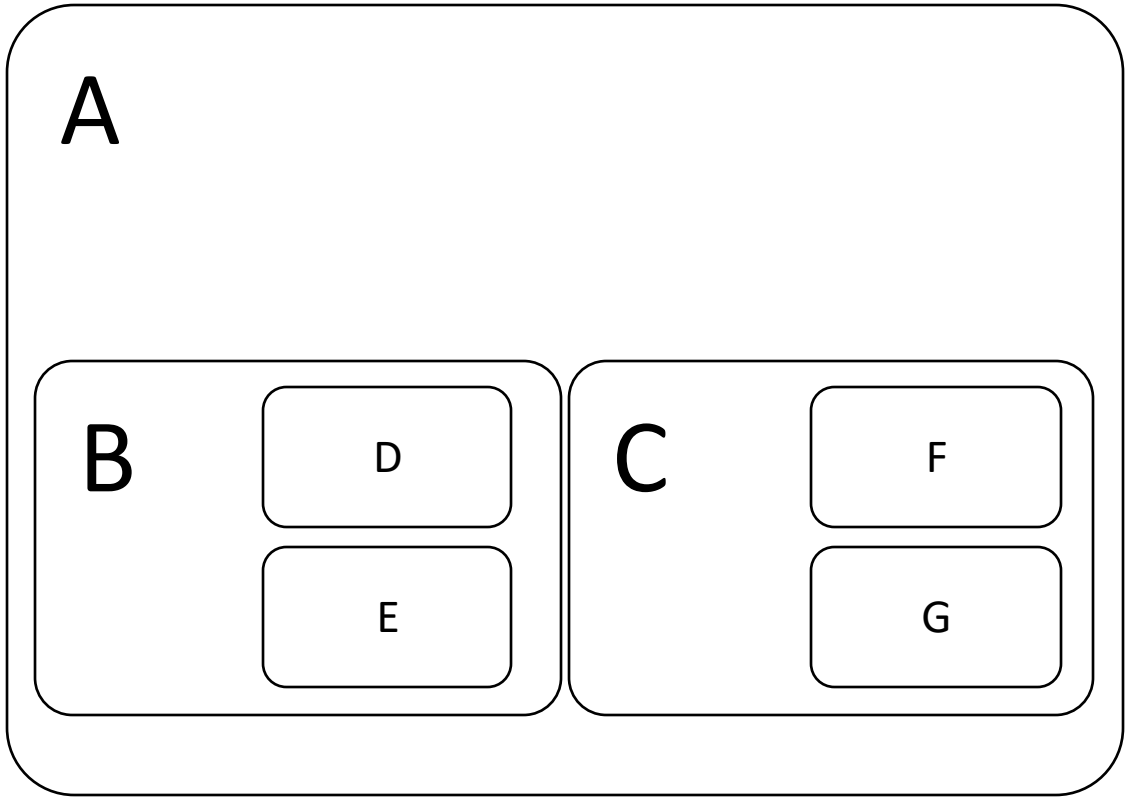
This time we will cover hierarchy, flow - instantiation

Hierarchy (it does not show interconnections)

- Tree shaped view



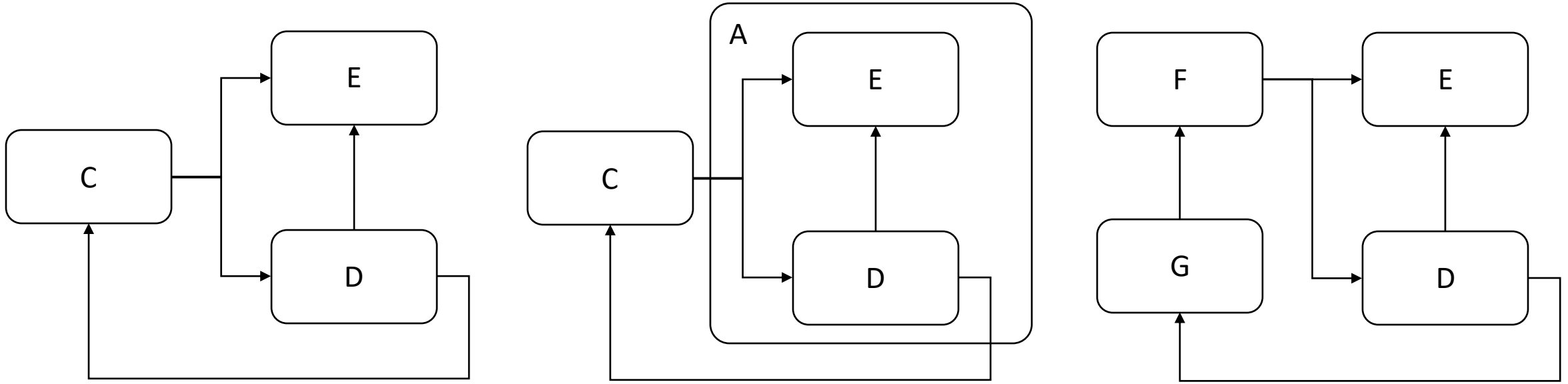
- Levels



Can be used to hierarchy of parts or functions

Flow - Instantiation (it does not focus on hierarchy)

- Interface-flows / ownerships

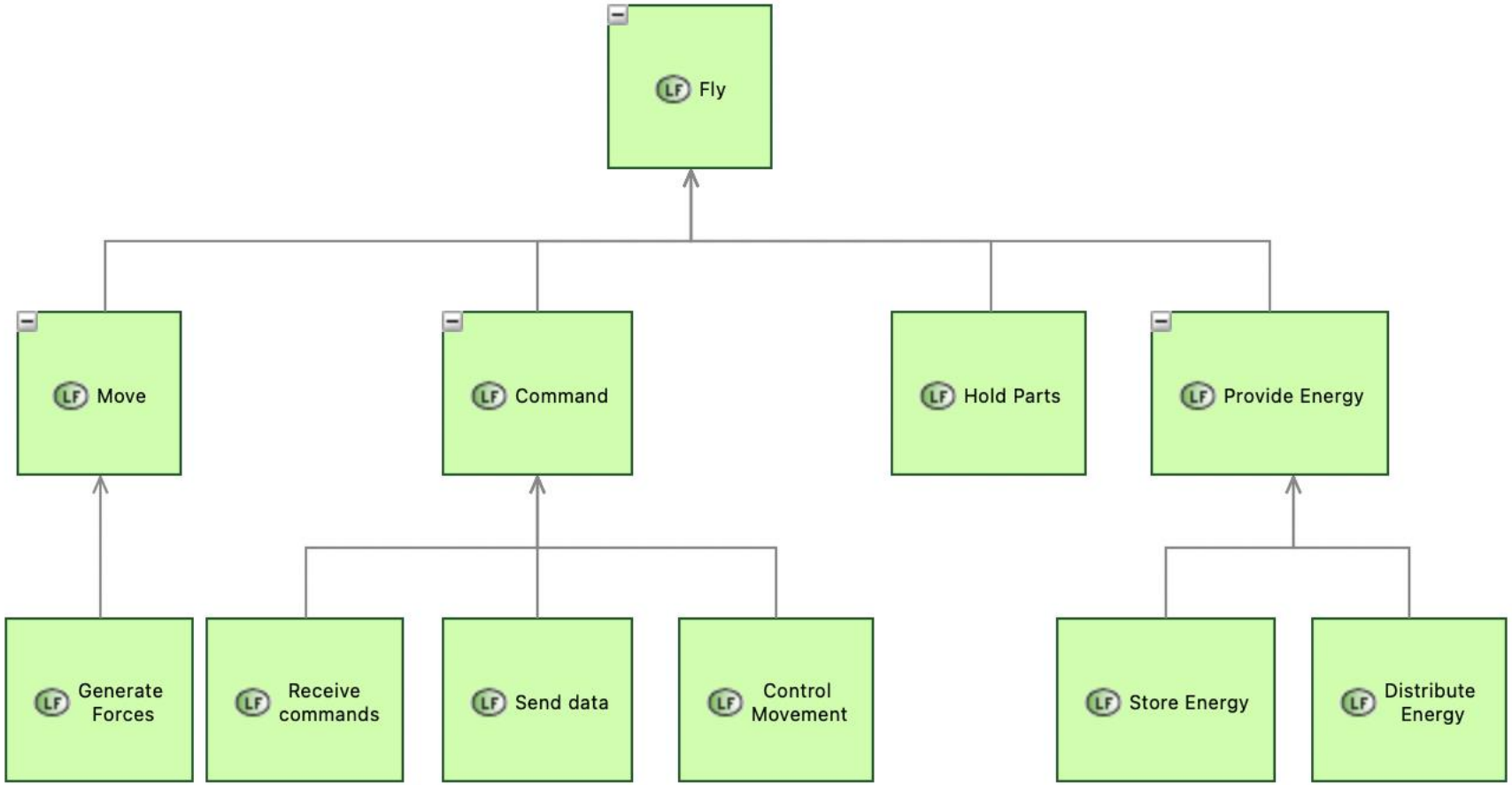


Can be used to show the interconection/flow of parts or functions

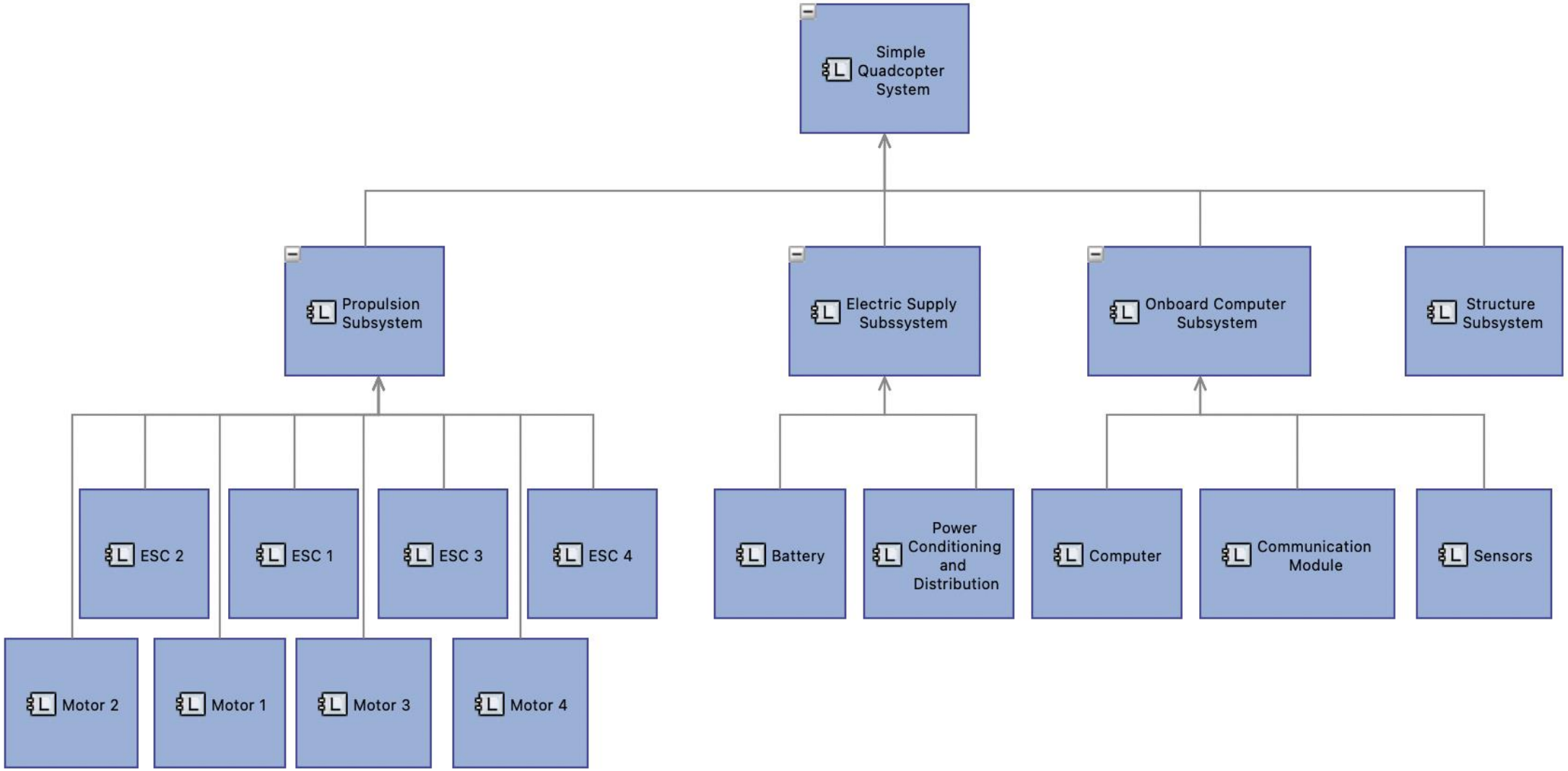
Example

Prepared on the Logical Architecture Layer

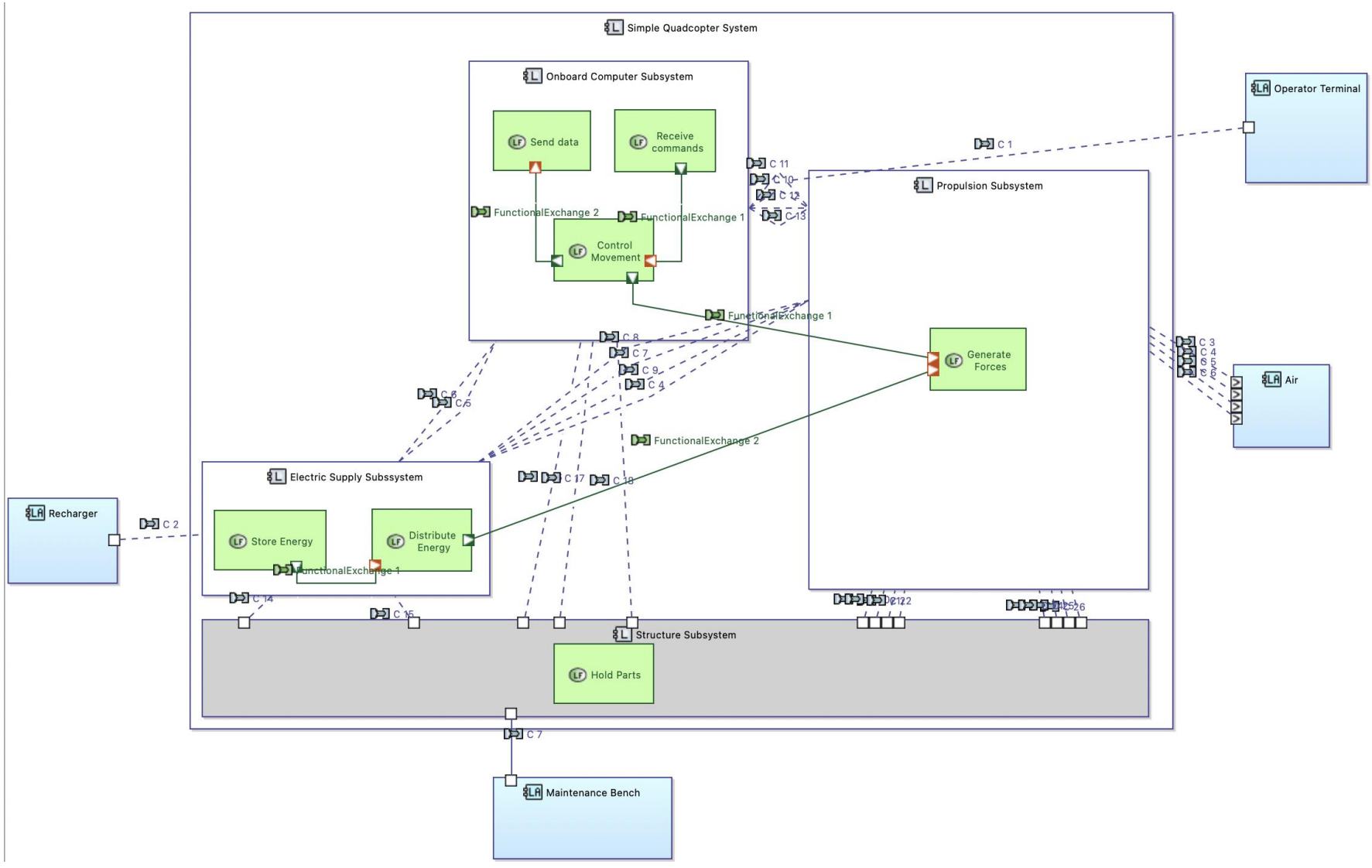
Function Hierarchy Example



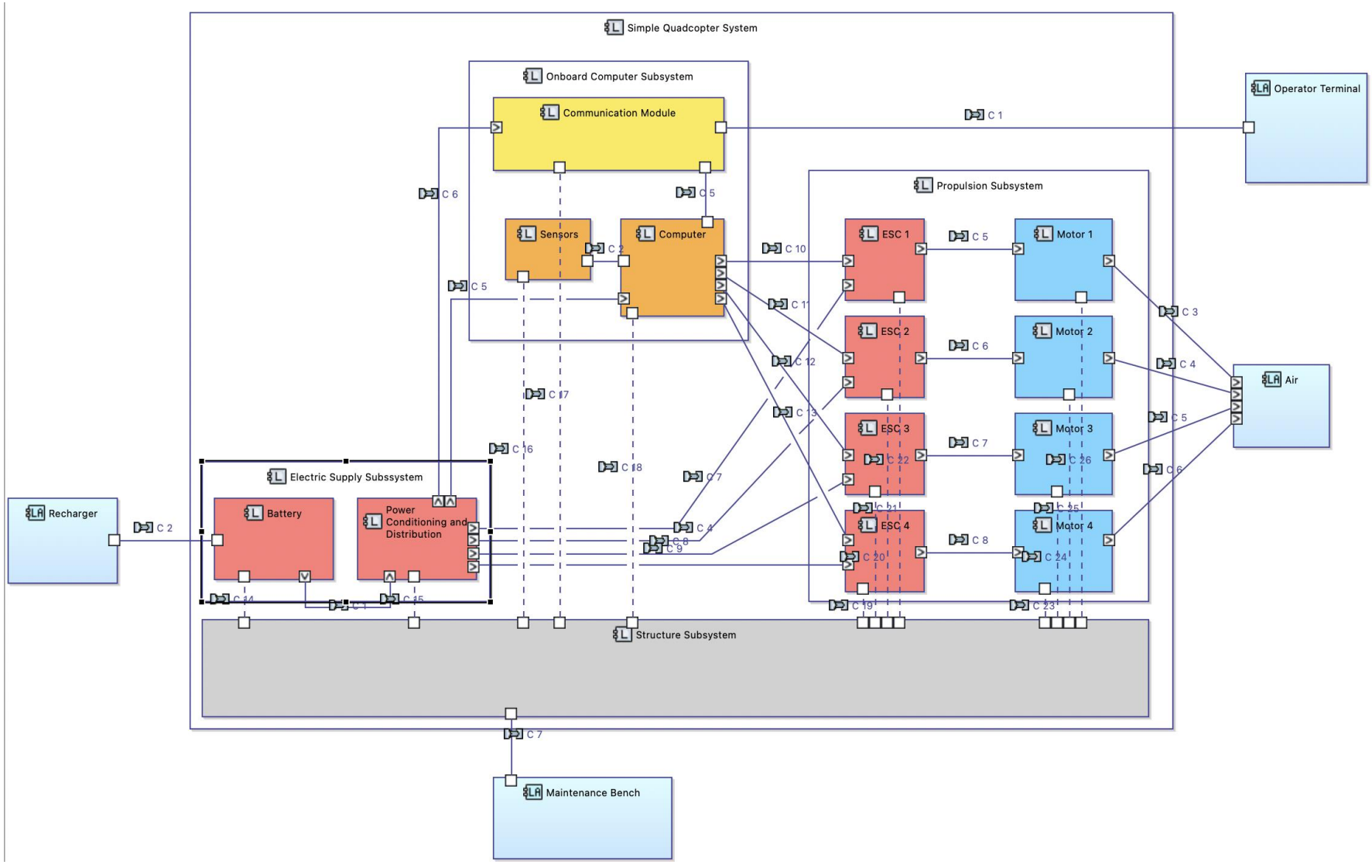
Part Hierarchy Example



Instantiated Functions Example

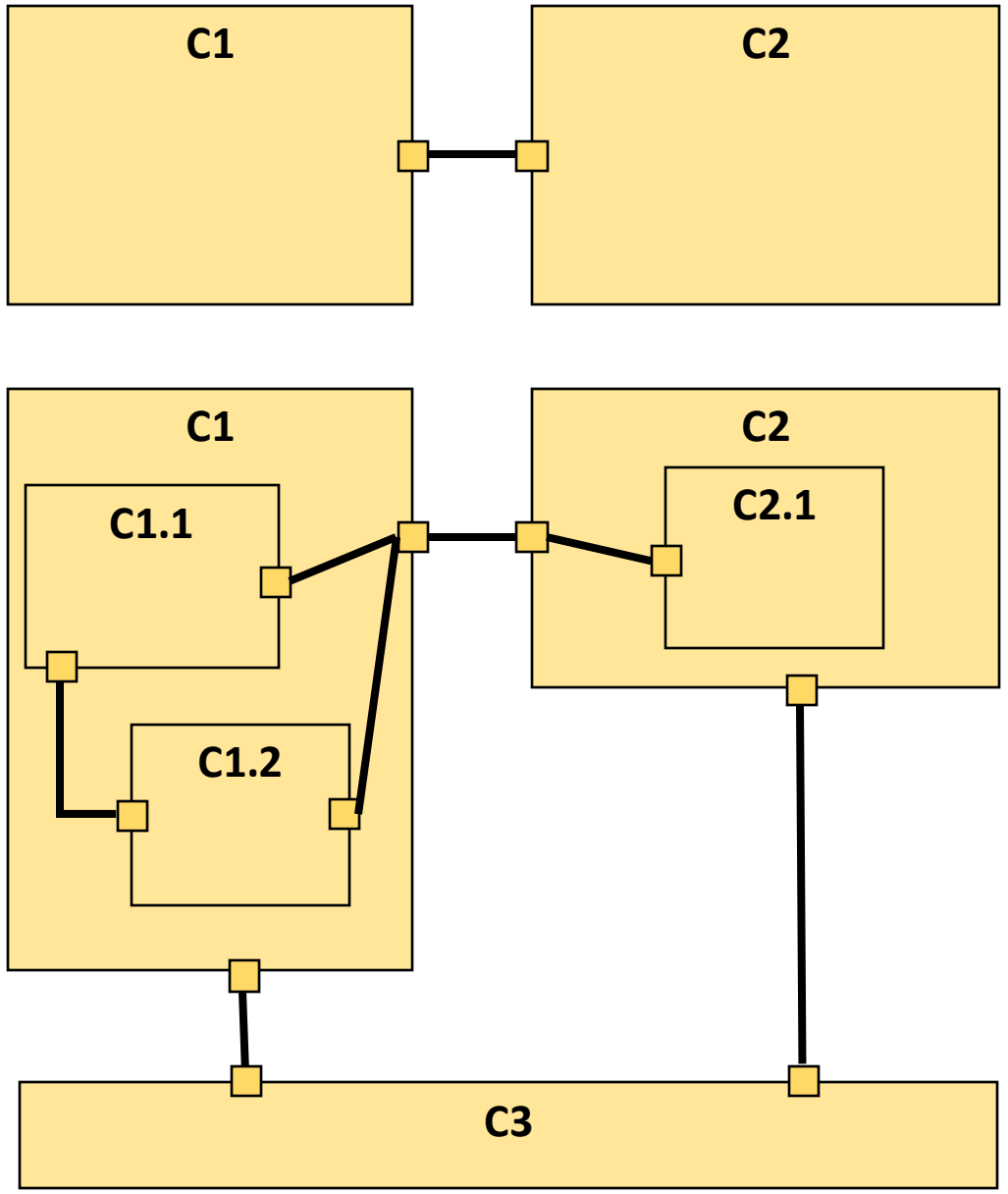


Mapped Flow Example



Definitions

- The **system** is an ordered set of elements that work as a whole, responding to the demand and needs of the customer and the user, and subject to engineering supported by Arcadia.
- An **actor** is an entity that is external to the system (human or otherwise), interacting with it, especially through its interfaces.
- A **component** is a constituent part of the system, contributing to its behavior and/or properties, along with other components and actors external to the system.
  - A component can be **broken down into subcomponents**.
  - To generalize, a component can also be allocated to an actor, to define their interactions and connections with the system or other actors.

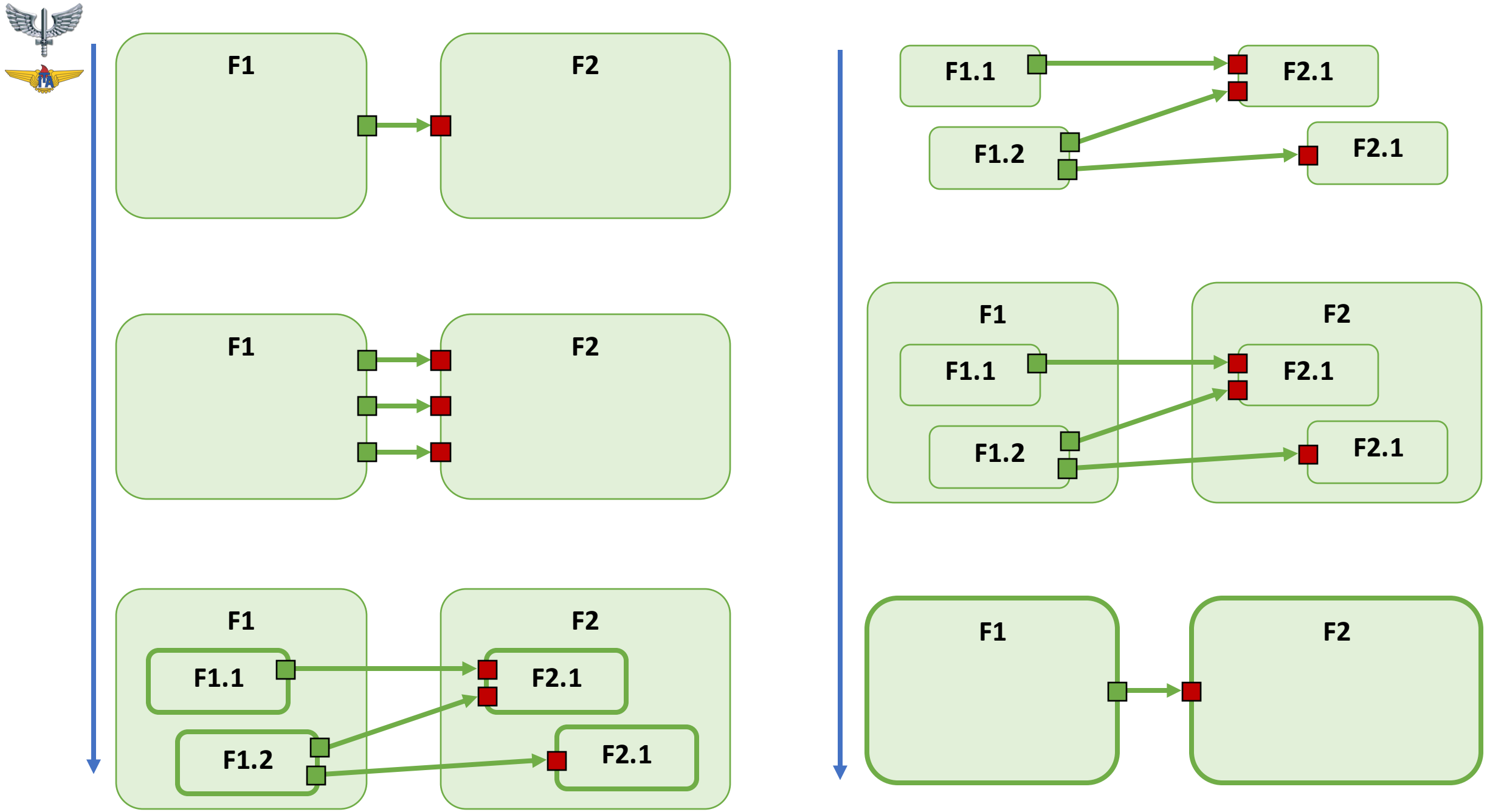

C1
C2
C1
C2
C1.1
C2.1
C1.2
C3

Functional Analysis

- **Functional analysis** is a classic technique widely used by systems engineers.
- **Arcadia and Capella** provide methodological guidance and engineering aids to support this technique that **was left out of SysML**.
- The mapping of Capella functions **is the most natural in terms of semantics**.
- **Functions are verbs** that specify the expected actions of the component to which they are allocated.

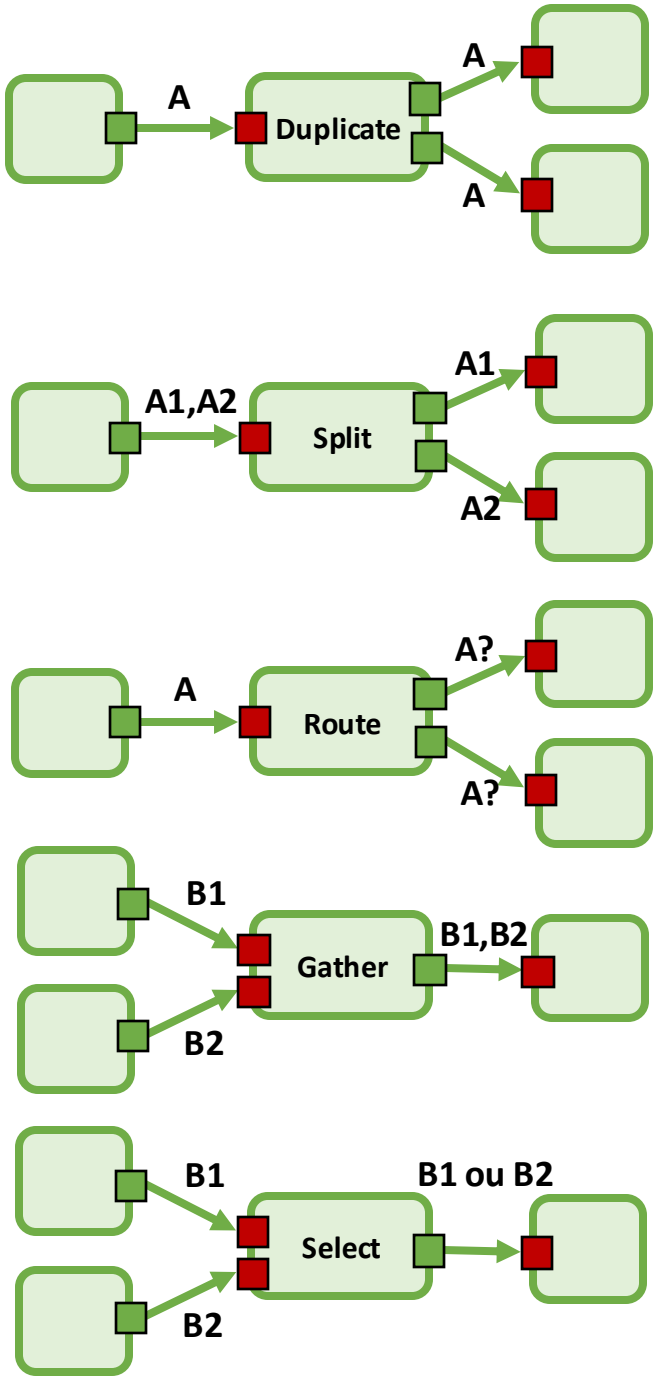
Functions for ARCML

- A function is an action, an operation, or a service, performed by the system or one of its components, or also by an actor interacting with the system.
  - Executing a function usually produces **exchange items** expected by other functions, and to do so requires other items provided by other functions.
  - Multiple functions can be grouped into a parent function (they are then called subfunctions, or child functions, of this function). Symmetrically, a function can be refinedinto multiple functions.
  - This grouping is not a strong relationship of structural decomposition; The grouping of functions forms only a synthetic representation of them, essentially for documentary purposes.
- Generally, in a finished model, **only the leaf functions** (without subfunctions) refer to and carry the expected functional description.


F1
F2
F1
F2
F1
F2
F1.1
F1.2
F2.1
F2.1
F1
F2
F1.1
F1.2
F2.1
F2.1
F1
F2
F1.1
F1.2
F2.1
F2.1
F1
F2

Flow Controls

- Flow control functions are intermediaries between the source(s) and the recipient(s), responsible for controlling the conditions of interaction:
  - To specify a **concurrent broadcast** of a source exchange to multiple recipients, we define a Duplicate function that transmits the same exchange items to all recipients;
  - to specify the simultaneous broadcast of some of the swap items for **each recipient selectively**, a Split function that routes each part to a separate recipient;
  - to specify the **selection of one among several potential recipients, a Route** function that transmits (most often subject to conditions) to each destination only some of the received exchange items;
  - to specify the **combination of items from multiple trades** from different sources, a Gather function can be a single trade item combining those received from different sources;
  - to specify the **selection of one source among many**, a Select function that directs only the elements coming from the selected source (most often subject to conditions)

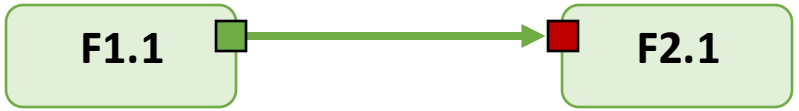


Exchanges

ITENS DE TROCAS (EXCHANGE ITENS)

- Um item de troca é um conjunto ordenado de referências a elementos roteados juntos, durante uma interação ou troca entre funções, componentes e atores.
- Os itens são roteados simultaneamente, nas mesmas condições, com as mesmas propriedades não funcionais. Esses itens são chamados de dados e são caracterizados pela **classe** à qual pertencem.
- Um item de troca é definido por:
  - um nome;
  - A lista de elementos do item de troca; cada elemento é definido no item de troca por um nome, e a classe à qual ele pertence, e se a troca é bidirecional, a direção de transmissão (por convenção, "in" na direção da troca por padrão, "out" na direção oposta, ou "in/out");
  - a descrição das condições de comunicação, se necessário, por exemplo, serviço, mensagem, evento, fluxo de dados, dados compartilhados, fluxo de material, quantidade física, etc.

TROCAS FUNCIONAIS



- **Pelo menos um item de troca** deve ser alocado para cada porta funcional em uma função para caracterizar o conteúdo que a função pode produzir ou que ela precisa.
  - Este item de troca **pode ser compartilhado por várias portas**.

- Se uma porta transporta vários itens de troca, então precisamos especificar, em cada uma das trocas funcionais conectadas a ela, o(s) item(ns) realmente roteado(s), que deve ser coerente com os das portas conectadas pela troca. Além disso, por conveniência, é possível começar alocando um item para uma troca, antes de propagá-lo para as portas conectadas a ele.

- **Recomenda-se definir apenas um único item em cada troca funcional**.

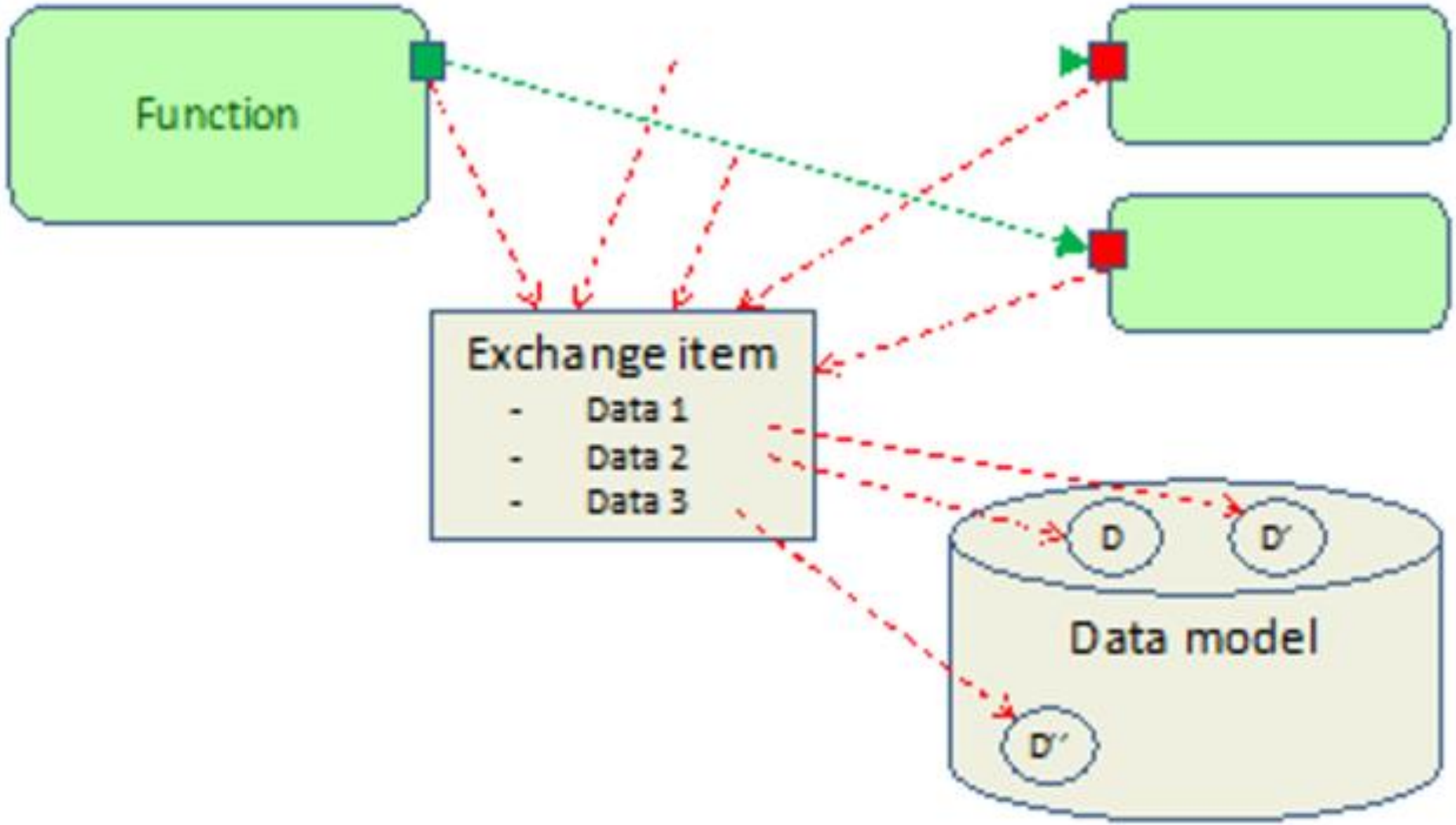
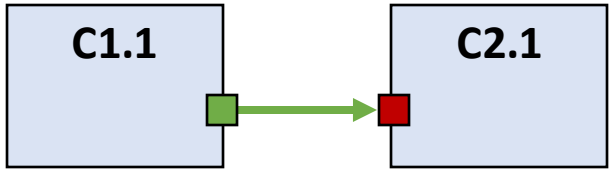


Figure 21.2. *Allocation of exchange items to functional ports and exchanges*

Exchanges between components



- The **content of an exchange** between components is defined by the exchange items carried over by the functional exchanges it implements.

- **An interface is a set of exchange items** that allows two components (and the system and the actors) to communicate with each other, according to a communication "contract" shared between them.

- **Multiple interfaces can be grouped into a single interface.**

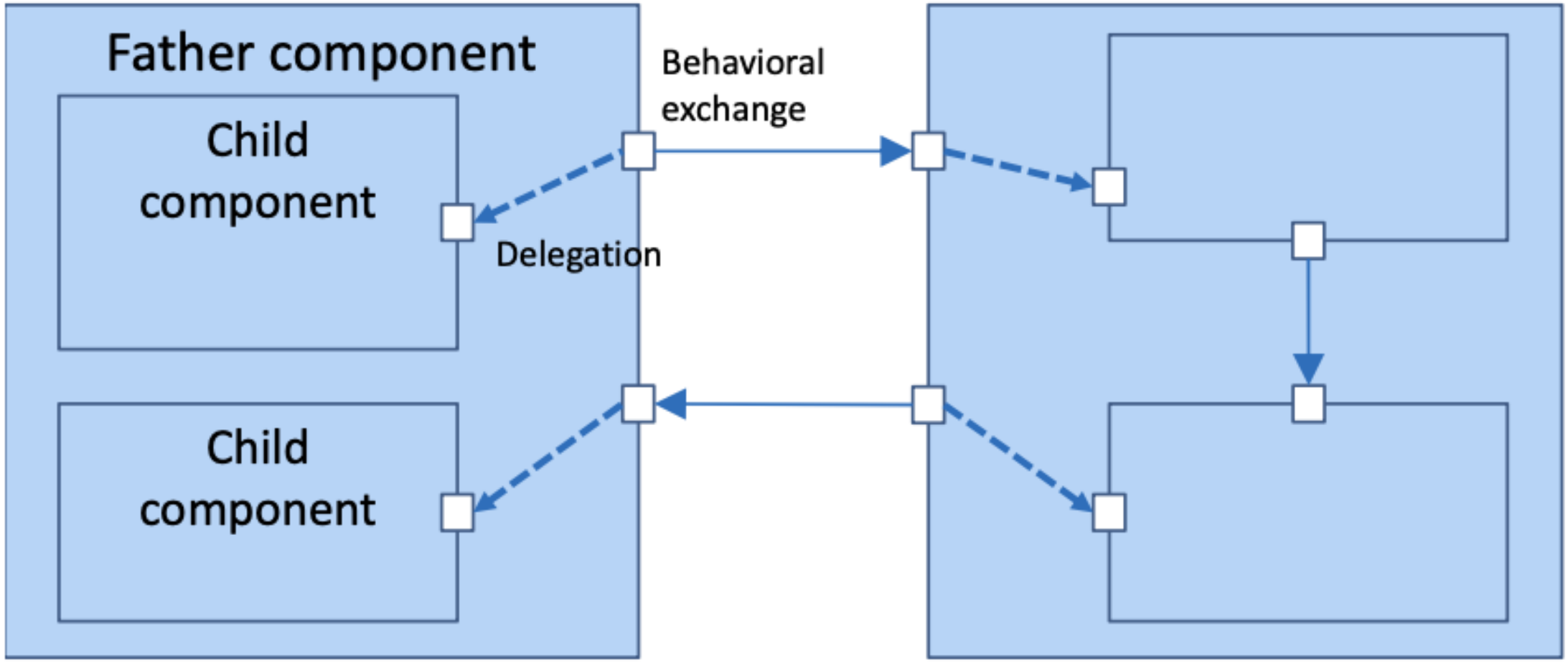


Figure 19.2. *Behavioral components, ports, exchanges and delegation*

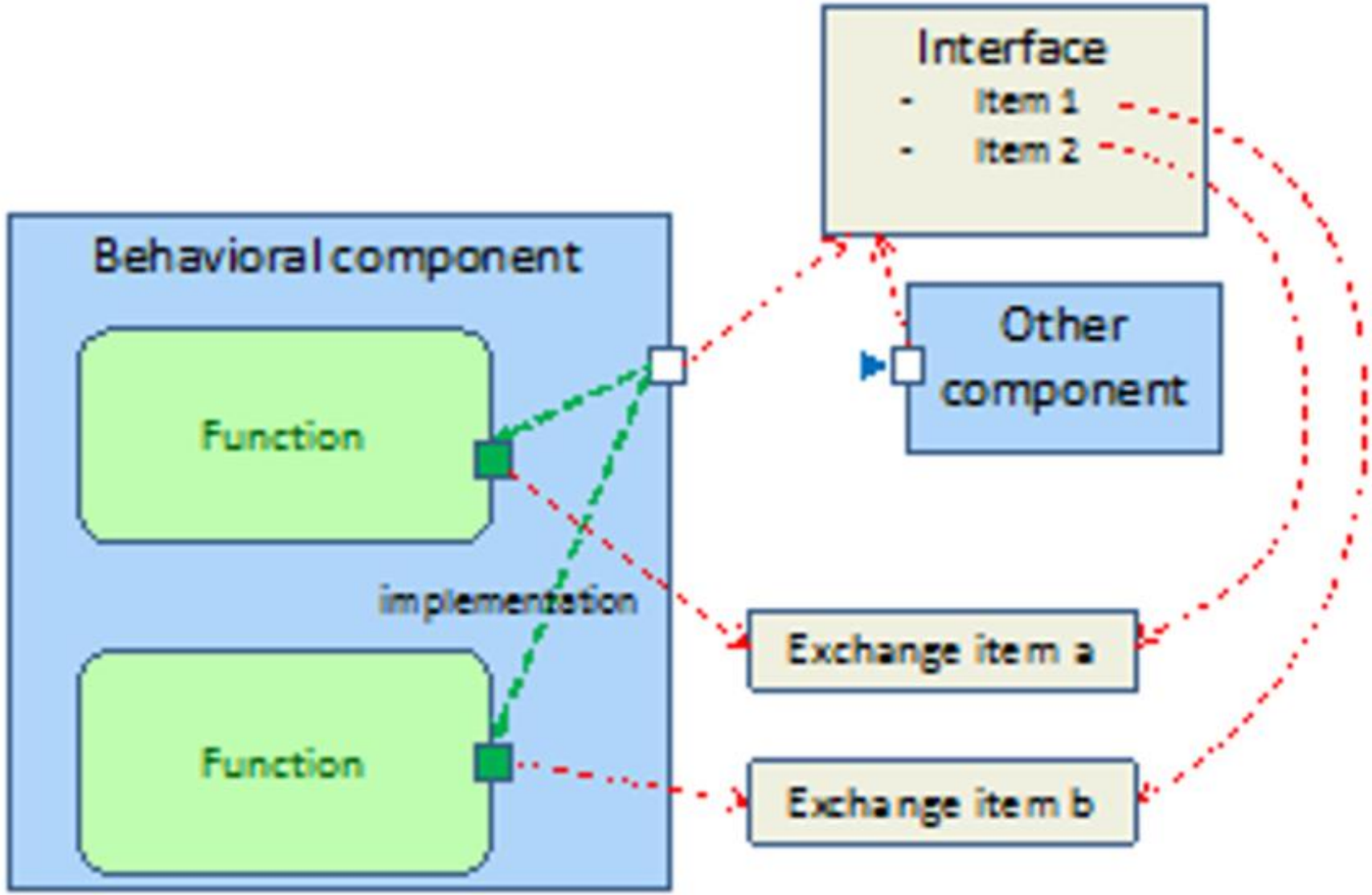
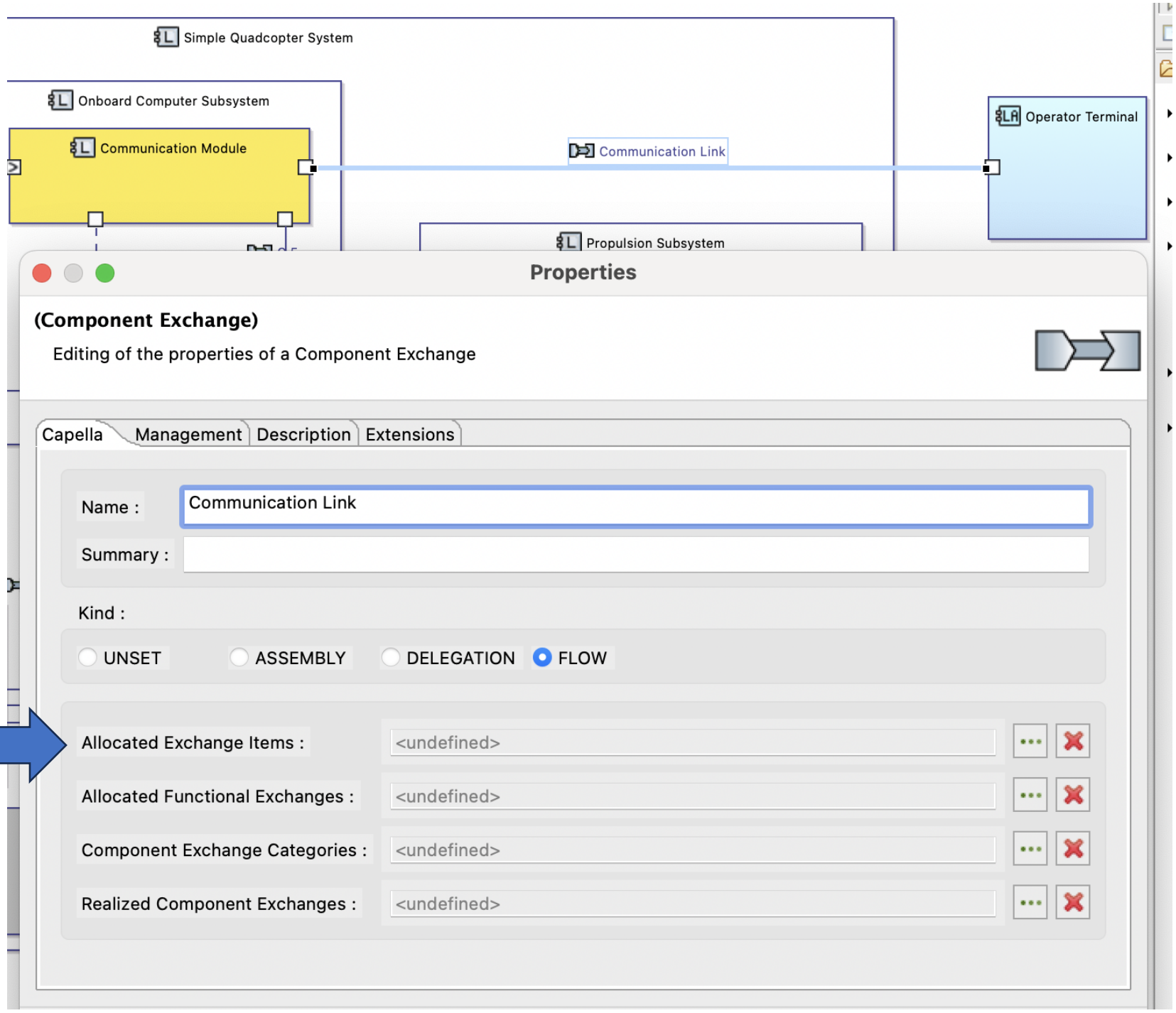


Figure 21.3. *Links between exchange elements involved in the functional and structural description*

Class Diagrams

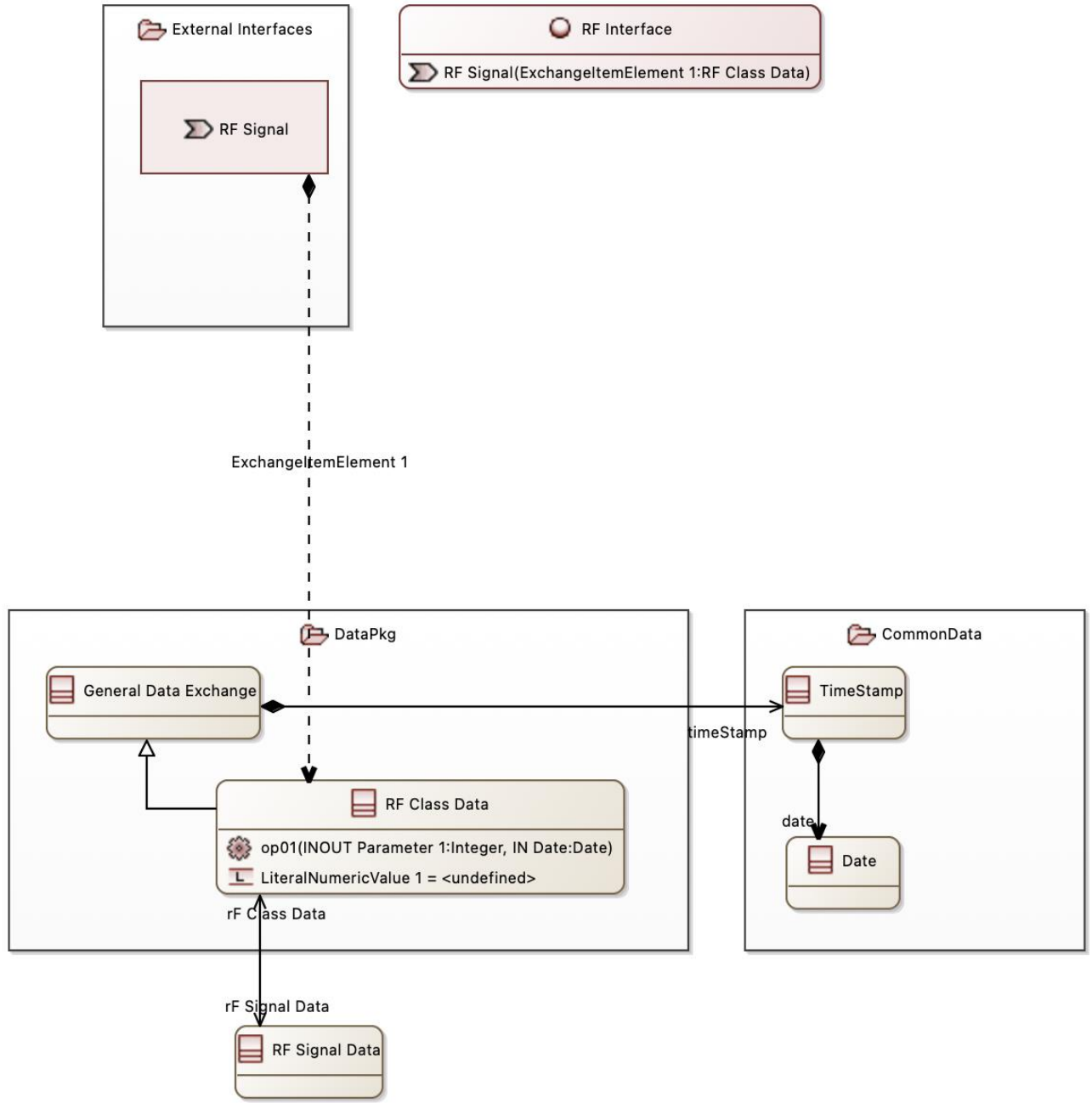
Used to improve description of the exchanged items

We can improve the overall description of the exchanged items



Class Diagrams

- The way that Arcadia defined the Exchange item description was using the Class Diagram.
- The Class Diagram is a way to create complex data structures, that in this case, is used to define the Exchange Items.
- This is the only use... up until now.

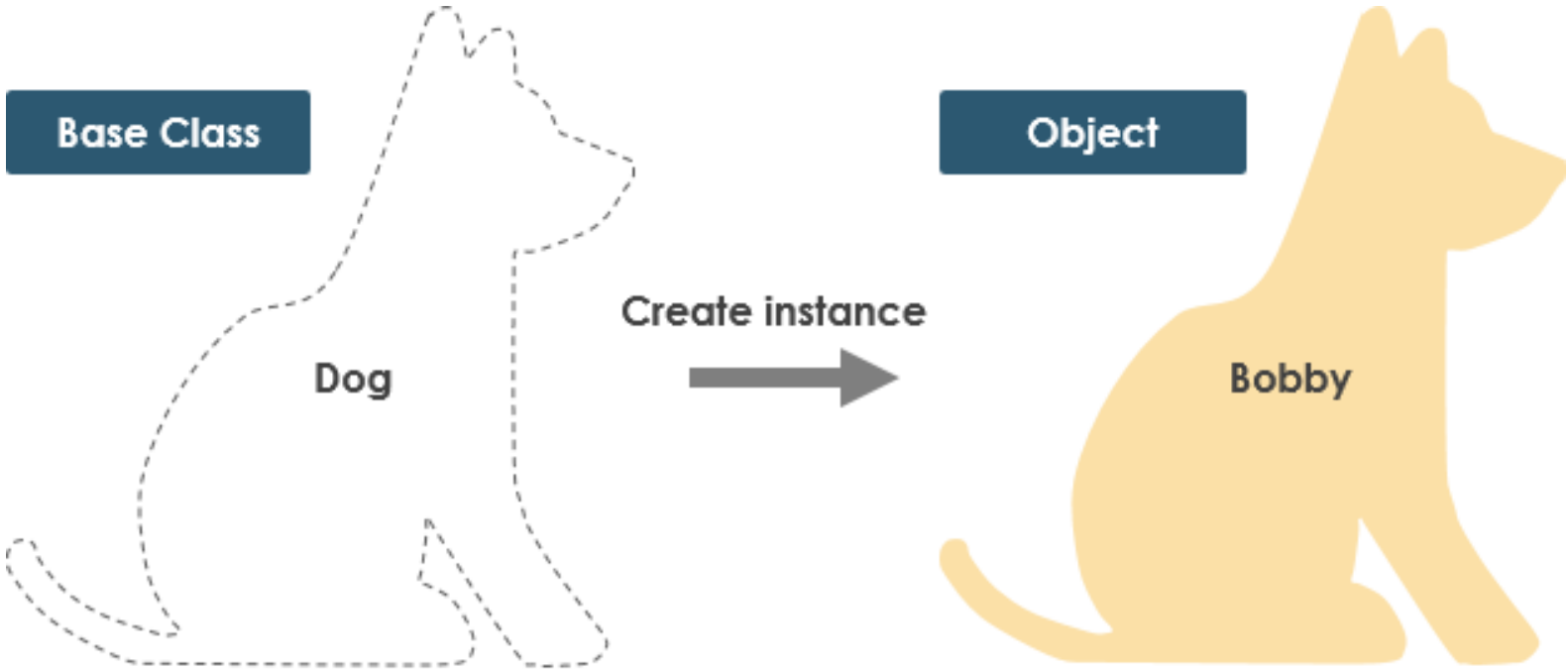

External Interfaces
RF Signal
RF Interface
RF Signal(ExchangeItemElement 1:RF Class Data)
ExchangeItemElement 1
DataPkg
General Data Exchange
RF Class Data
op01(INOUT Parameter 1:Integer, IN Date:Date)
LiteralNumericValue 1 = <undefined>
rF Class Data
rF Signal Data
RF Signal Data
timeStamp
CommonData
TimeStamp
date
Date

Fast UML Class Diagram catch-up

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

- The UML Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:
  - classes,
  - their attributes,
  - operations (or methods),
  - and the relationships among objects.

A Class is a blueprint for an object

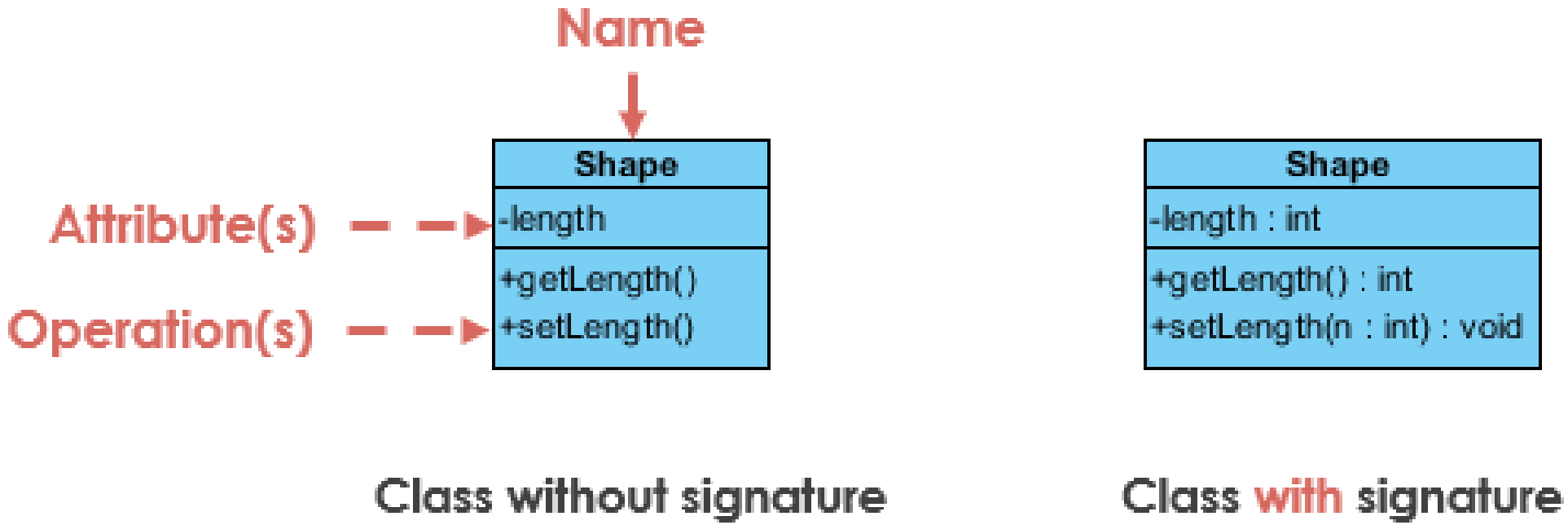


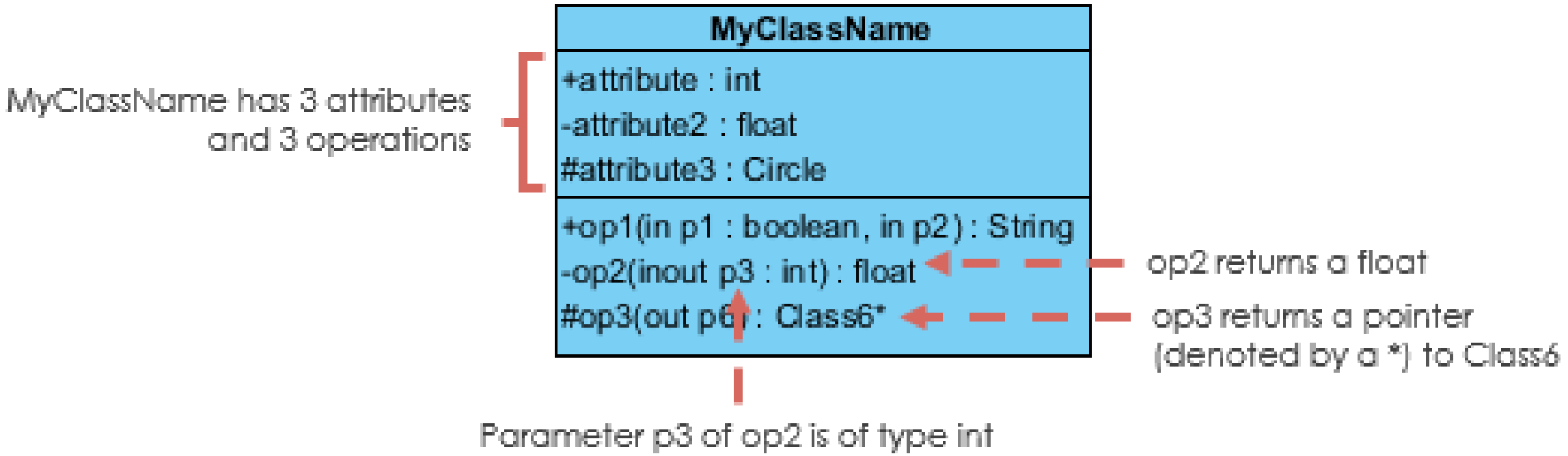
Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

UML Class Notation

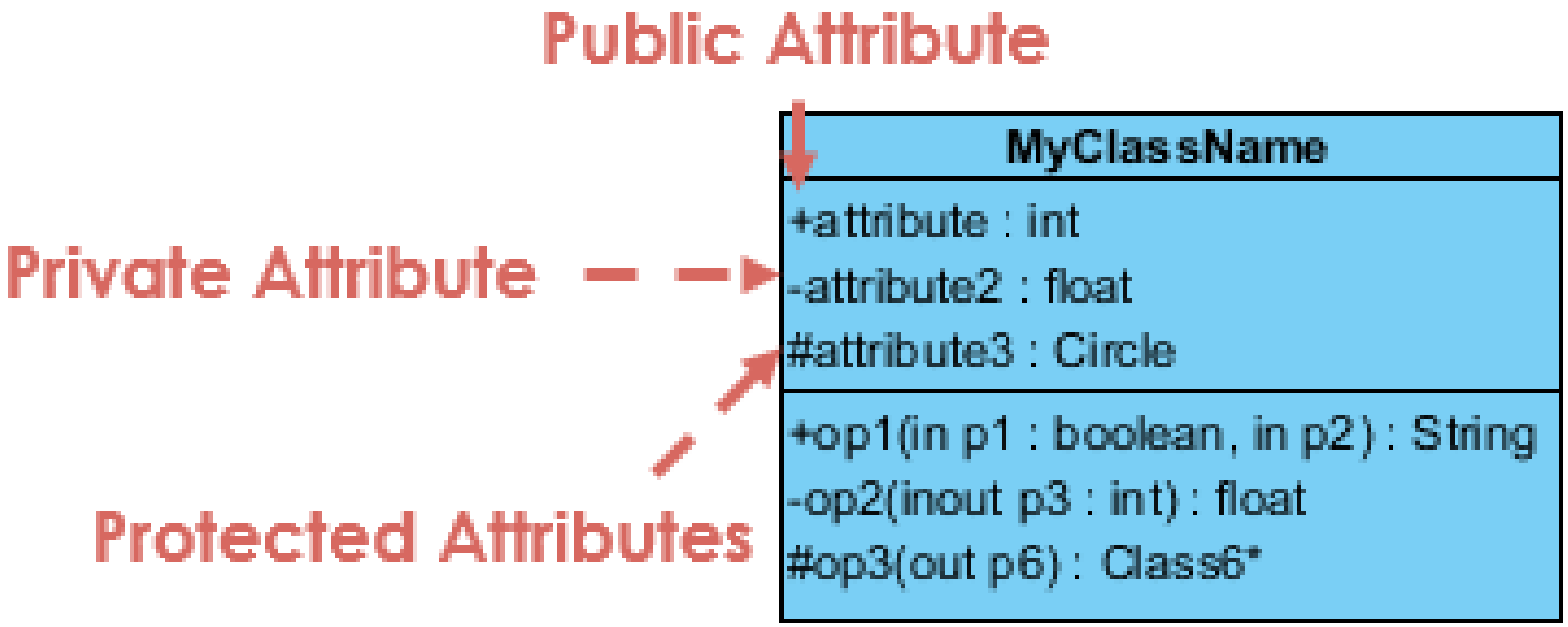
- A class represent a concept which encapsulates state (attributes) and behavior (operations). Each attribute has a type. Each operation has a signature. The class name is the only mandatory information.




MyClassName
+attribute : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : boolean, in p2) : String
-op2(inout p3 : int) : float
#op3(out p6 : Class6*
MyClassName has 3 attributes and 3 operations
op2 returns a float
op3 returns a pointer (denoted by a *) to Class6
Parameter p3 of op2 is of type int

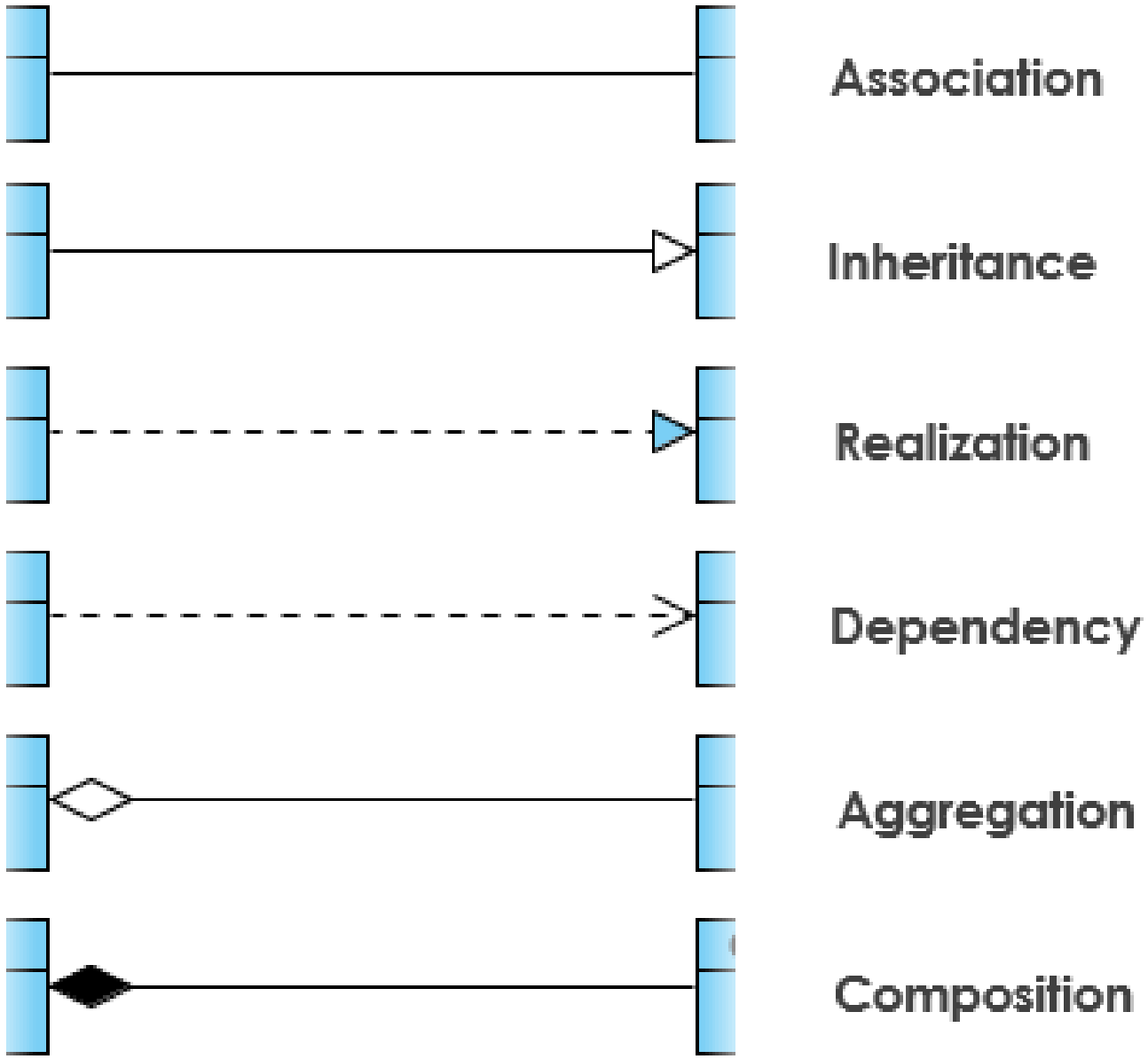
Class Visibility

- The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.
- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations


Public Attribute
MyClassName
+attribute : int
-attribute2 : float
#attribute3 : Circle
Private Attribute
+op1(in p1 : boolean, in p2) : String
-op2(inout p3 : int) : float
#op3(out p6) : Class6*
Protected Attributes

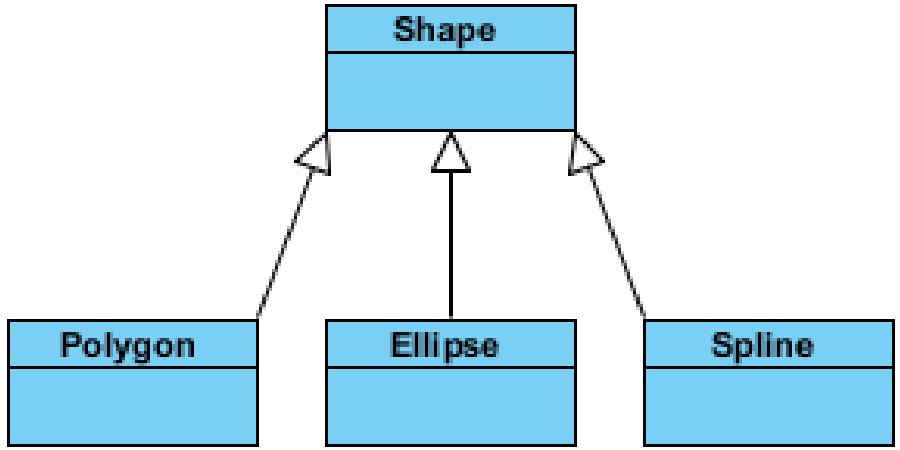
Relationships

- A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:

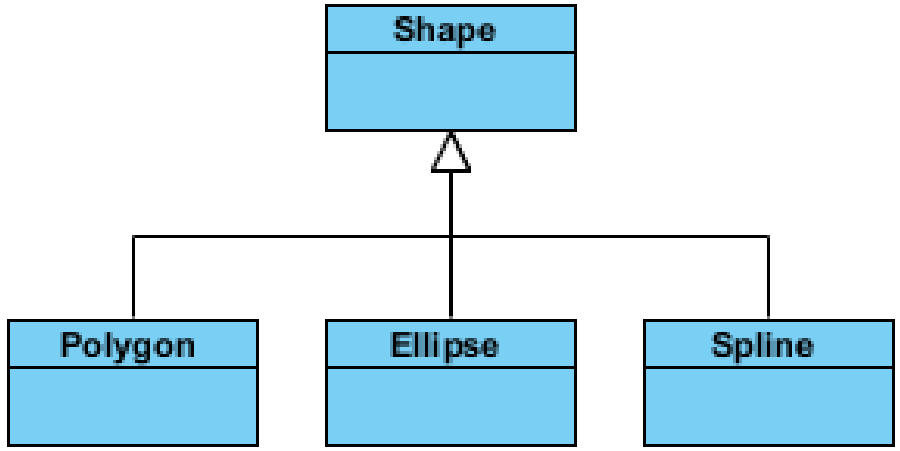


Inheritance (or Generalization):

- A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.



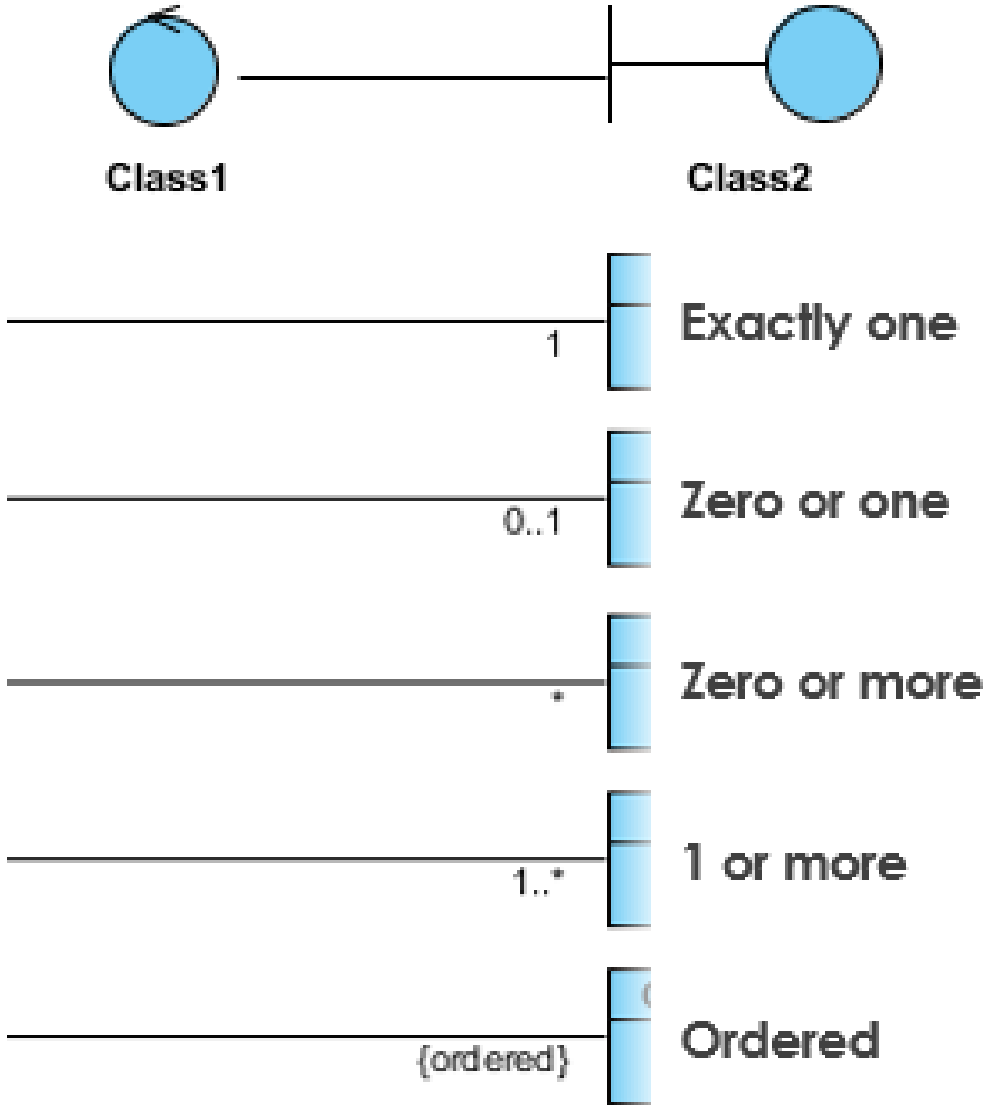
Style 1: Separate target



Style 2: Shared target

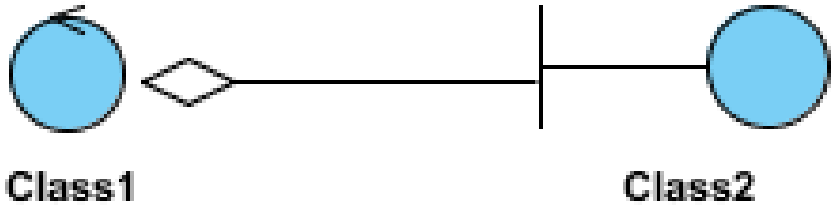
Association

- Associations are relationships between classes. Associations are typically named using a verb or verb phrase which reflects the real world problem domain.



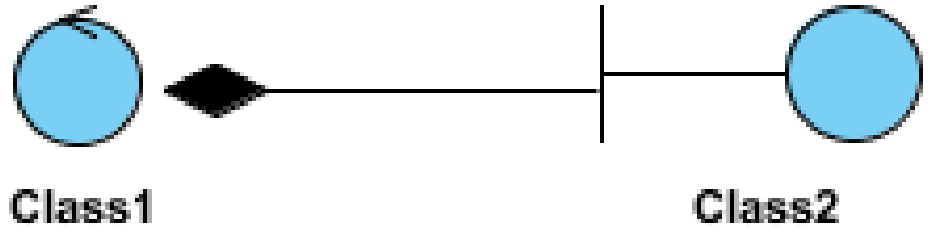
Aggregation

- A special type of association.
  - It represents a "part of" relationship.
  - Class2 is part of Class1.
  - Many instances (denoted by the *) of Class2 can be associated with Class1.
  - Objects of Class1 and Class2 have separate lifetimes.

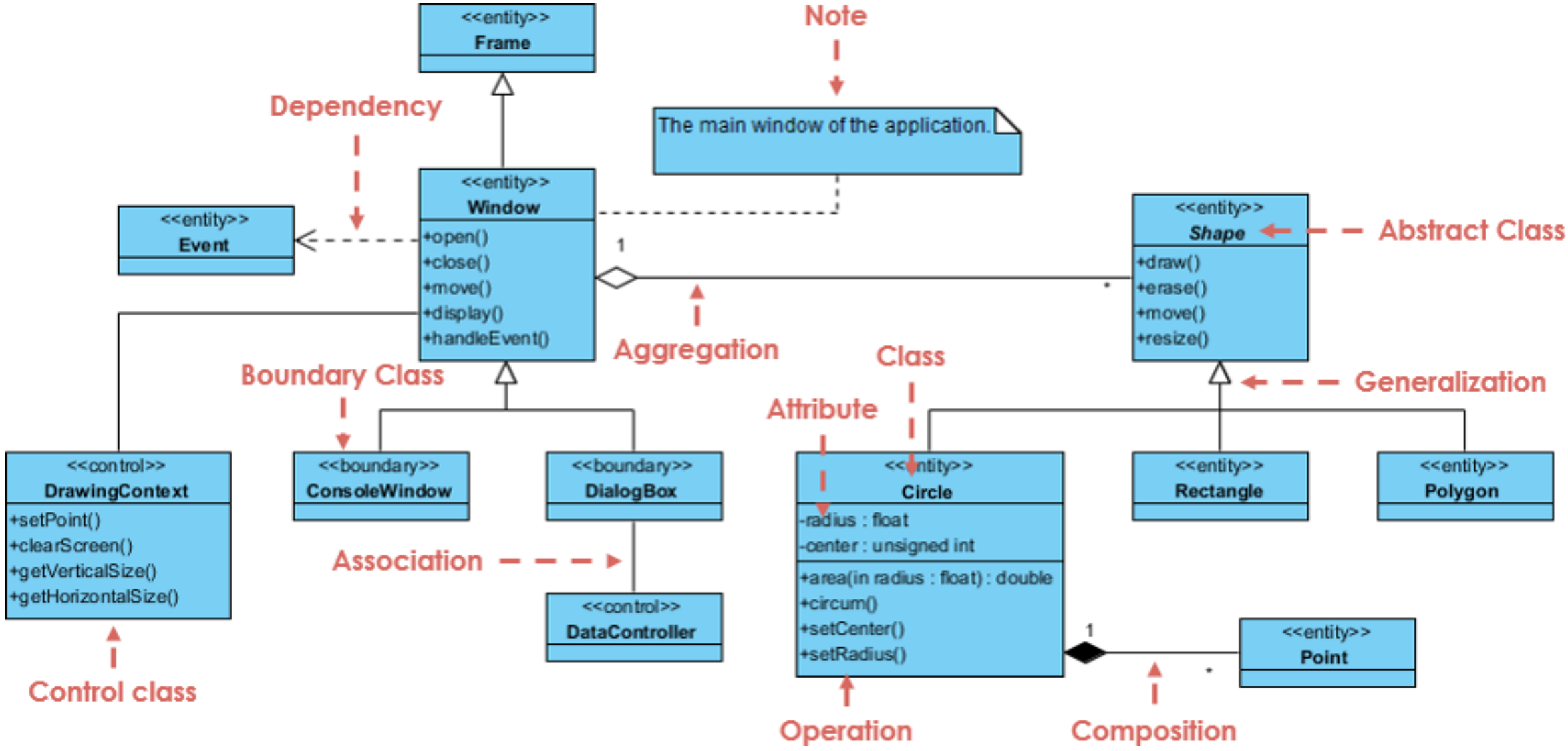


Composition

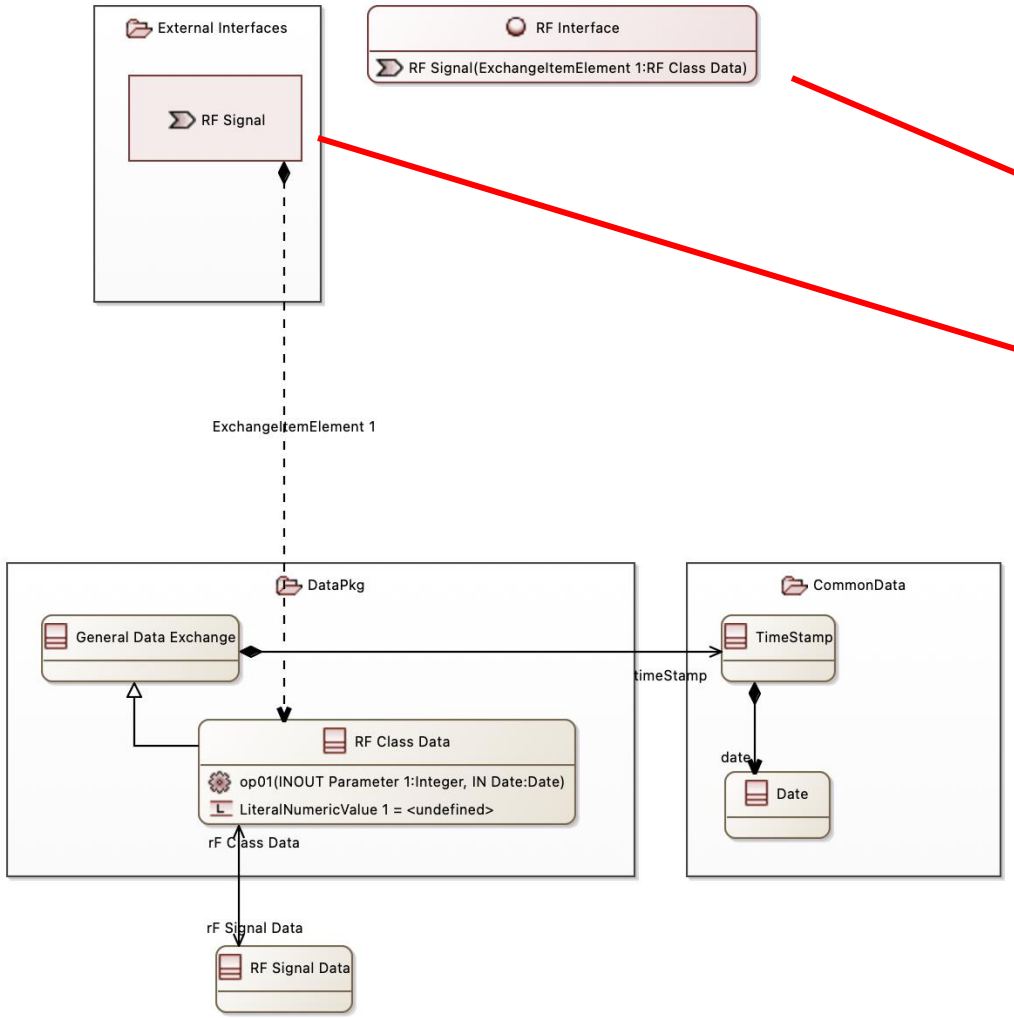
- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

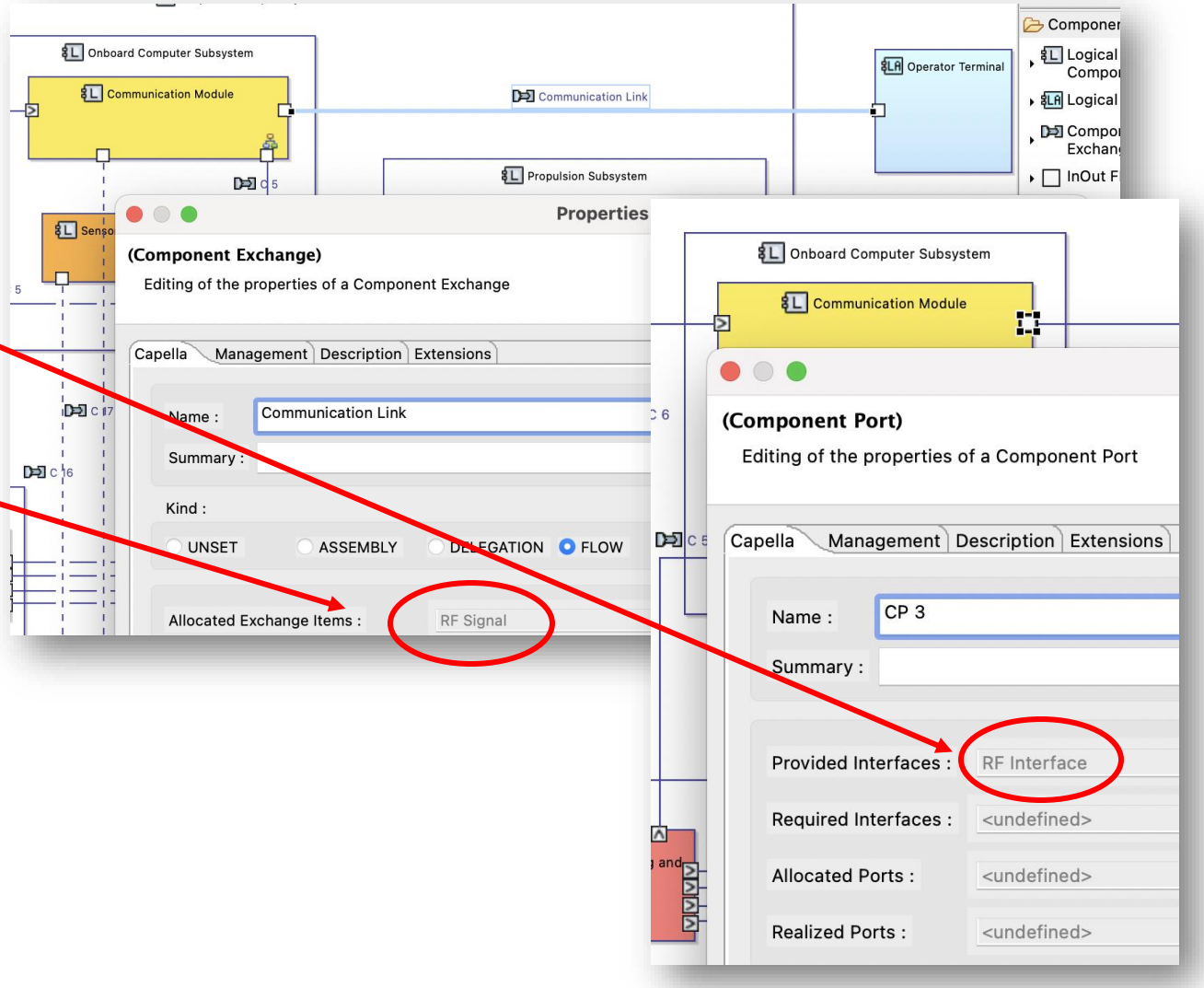


Class Diagram Example: GUI



And into our case:





Final Considerations

Final Consideration

- The block diagrams are used to represent functional and component (forms) architectures
- They are the simplest format to do systems engineering that I'm aware. If you like, research how the SysML implements those elements.
  - With a little effort you can create DSMs / eFFBDs / Activity Diagrams / so on..
- The exchanges are the key on the Arcadia, so define it well...