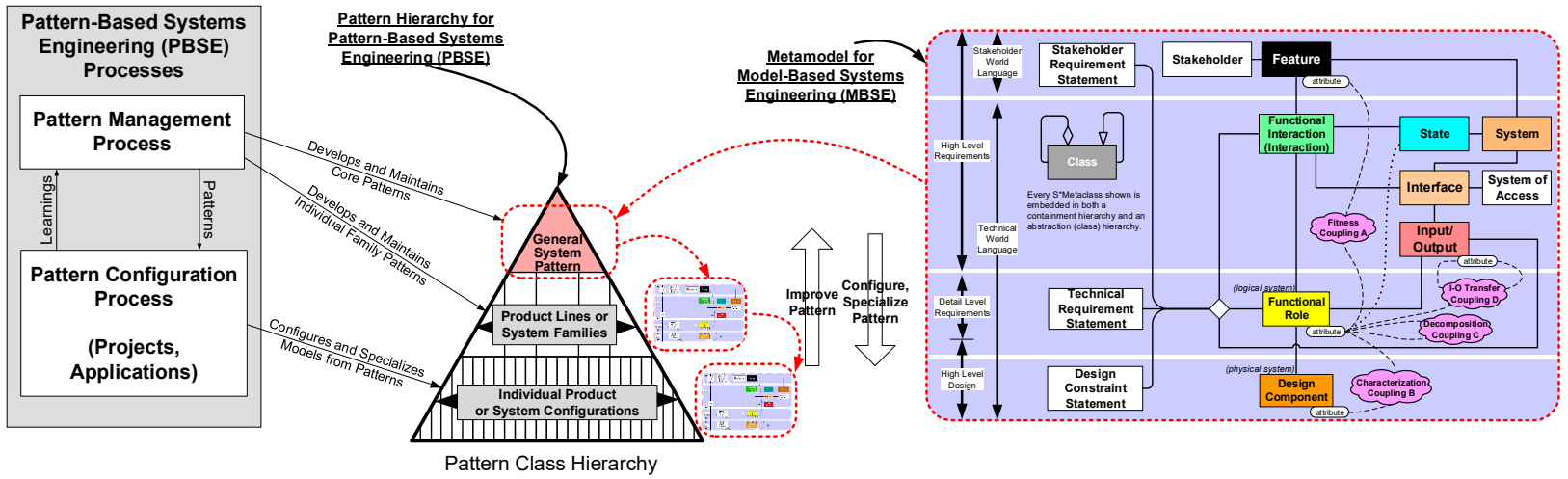
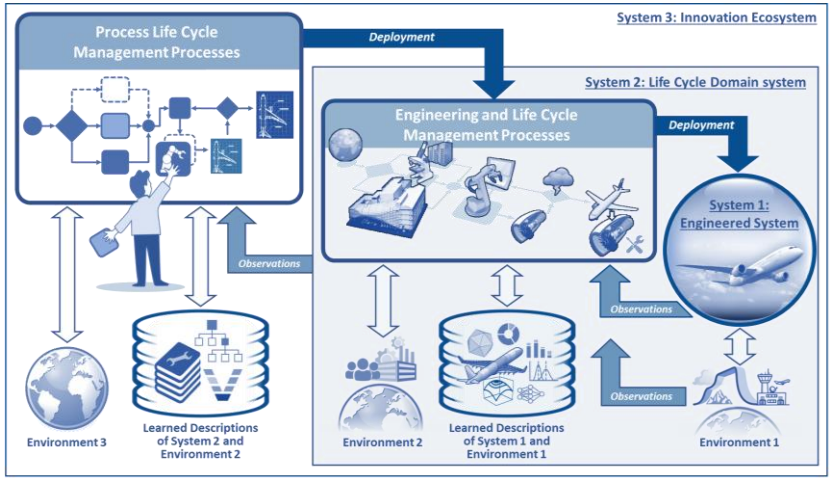
ICTT System Sciences®
Understand your systems.

<<Company/Org. Logo>>

Phase I Session 2

Introduction to S*Models, S*Metamodel, S*Patterns, Mappings to Languages & Tools, S*Configuration Wizard, ASELCM S*Pattern

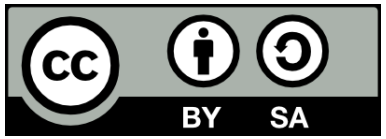




schindel@ictt.com
lewis@ictt.com

V1.2.17

Copyright 2025, System Sciences, LLC



Licensed under a Creative Commons
Attribution Share Alike-License CC BY SA International 4.0

License Link: https://creativecommons.org/licenses/by-sa/4.0/legalcode

Uses are permitted under this license without further permission from the copyright owner, provided each use (1) is clearly marked to attribute the underlying work to "S*Patterns Community", (2) provides a link to the CC BY SA license, (3) indicates if changes were made, (4) does not suggest the licensor endorses the user or use, (5) does not apply legal terms or technological measures that legally restrict others from doing anything the license permits, and (6) if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Permissions beyond the scope of this license are administered through contacting:

Corporate Officer
ICTT System Sciences
378 South Airport Street
Terre Haute, IN 47803
812-232-2208

Systematica is a registered trademark of System Sciences, LLC.

Contents

- Session 1 Take-aways, homework, questions, discussion
- Phase I Session 2 Class Content
- References

Recap of Session 1:
Take-aways impacting actual practices

1. What are **Systems**? Consistent use of System Domain Models;
2. What are **Interactions**? Consistent use of Interactions to describe behavior; no "naked" behavior;
3. **States**, state variables, finite state models: Consistent description of "When", temporal architecture;
4. **Input-Outputs**: Consistent representations of exchanged force, energy, material, information;
5. Stakeholder **Features**; Consistent representation of value across all stakeholders; trade space; Selection as the observable phenomenon describing value; all risk is risk to Features; use in FMEAs; use to drive PLE configuration variation;
6. **Finding all the Interactions** by managing consistency of Interactions across Features, States, and External Interfaces/Actors;
7. **Functional Roles versus Design Components**, what do we mean by 'allocate"? by "physical"?
8. **Attributes**: parameterizing Features, Roles, Design Components;
9. **S*Metamodel**, mapped to languages and third-party tools, as a consistent point of reference across variable environments.

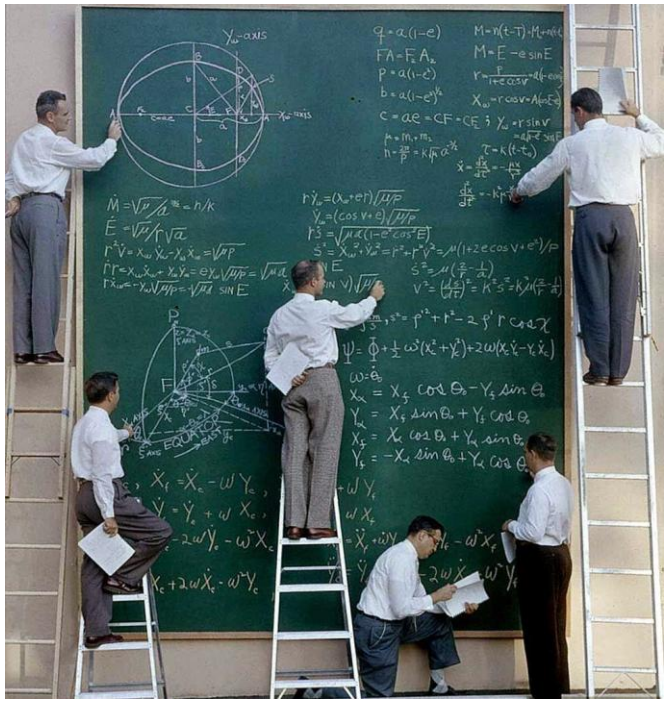


- Can a creative, talented team *align* its work?
- What should be *standard* about this, in your customer work?
- What should be *configurable* about this, in your customer work?



Session 1 Homework Discussion

Assignment was:



- Locate the pattern of the engineered system of interest to your organization or its customers, or example teaching pattern:
  - Since we will be studying models of the system of engineering itself as a modeled system in later units, for now the "engineered system" will be used for modelling.
  - For example, a defense platform or a commercial manufactured product family.

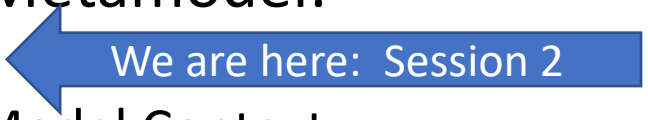
In a Local Pattern Package, *containing the existing Pattern*, augment the high-level S*Pattern for the above, to include:

- Domain System: Add external Domain Actors
- Add Stakeholders and Stakeholder Features
- Add Interactions (external Black Box interactions only) and the external Actors they include (as logical systems)
- Add States

Create External Interaction relationships with States, Features, and external Actors:

- Determine the completeness of your external Interactions set by their common coverage of States, Features, and External Actors
- Are all the Features "satisfied" by their Interactions?
- Are all the Actors "satisfied" by their Interactions?
- Are all the States (think about Use Cases) "satisfied" by their Interactions?

Phase I Class Sessions: Microstructure of Models and Patterns (6 Sessions, 3 hours each)

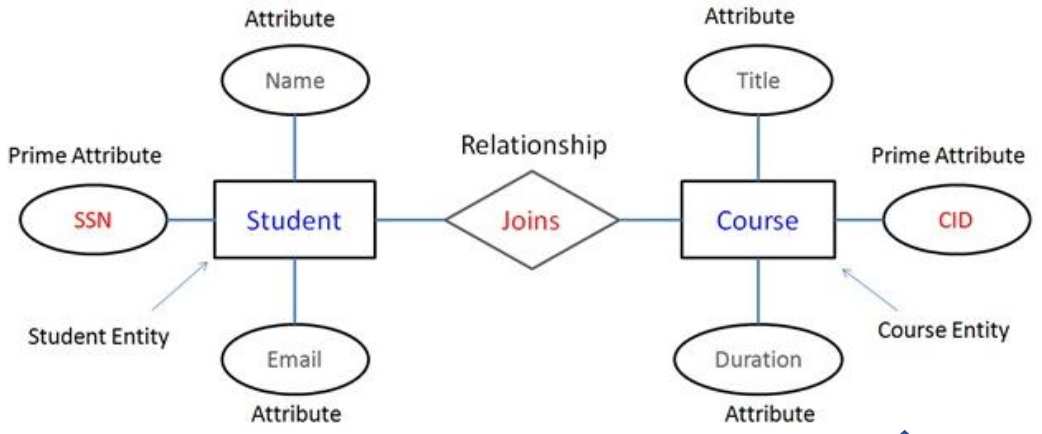
- S*Models, S*Metamodel:
  - Sessions 1-3 

  - In Example Model Context
- Formal mapping of S*Metamodel into Tooling
  - Session 4
- S*Patterns, Configuration: In Example Model Context
  - Session 5
- Introduction to 3rd Generation Pattern Configurator Technology
  - Session 6
- Homework Presentation and Feedback
  - Distributed over above sessions

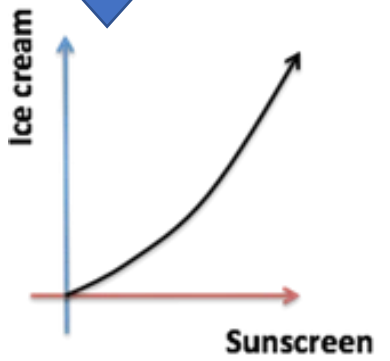
S*Models, S*Metamodel: Session 2 of 3

- Architectural Relationships
- Model-based Requirement Statements as transfer functions
- Interfaces, Systems of Access, Interface Context Model
- Attribute Couplings; fitness, input-output, decomposition, state, characterization, and heuristic couplings
- Homework assignment 2
- Discussion, Q&A

Architectural Relationships



- Key technical formalizations of relationships are variously seen in:
  - Entity-Relationship (E-R) modeling, Peter Chen, circa 1976, RDBMS data models
  - Set-theoretic relationships in mathematics, as in cross product sets [Set A X Set B]
  - Parametric relationships, including deterministic and statistical parametrics (more on those later, under Attribute Couplings)
  - Relationships in propositional logic



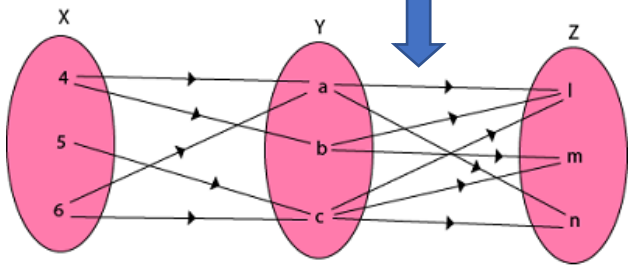
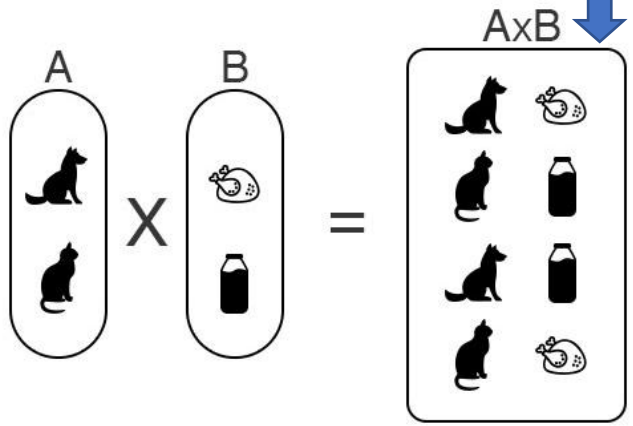


Fig : R_1 o R_2



"tuples"

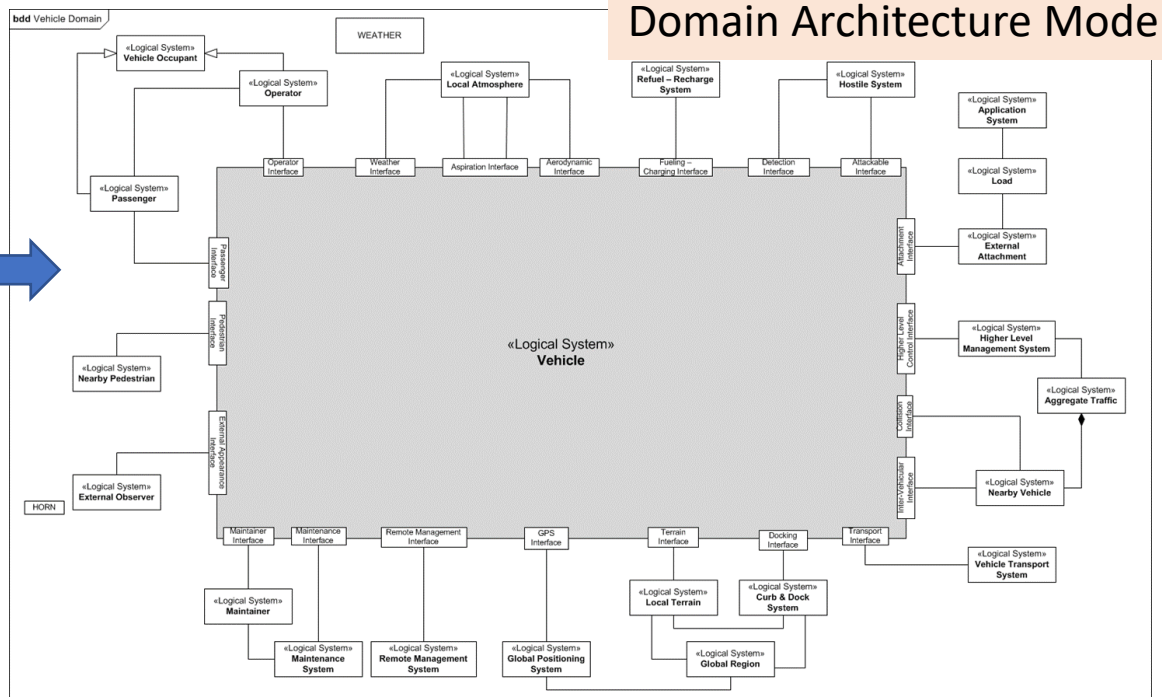
Architectural Relationships

We are interested in **Architectural Relationships** that appear (often as *connecting lines*, but not always that simply) within:

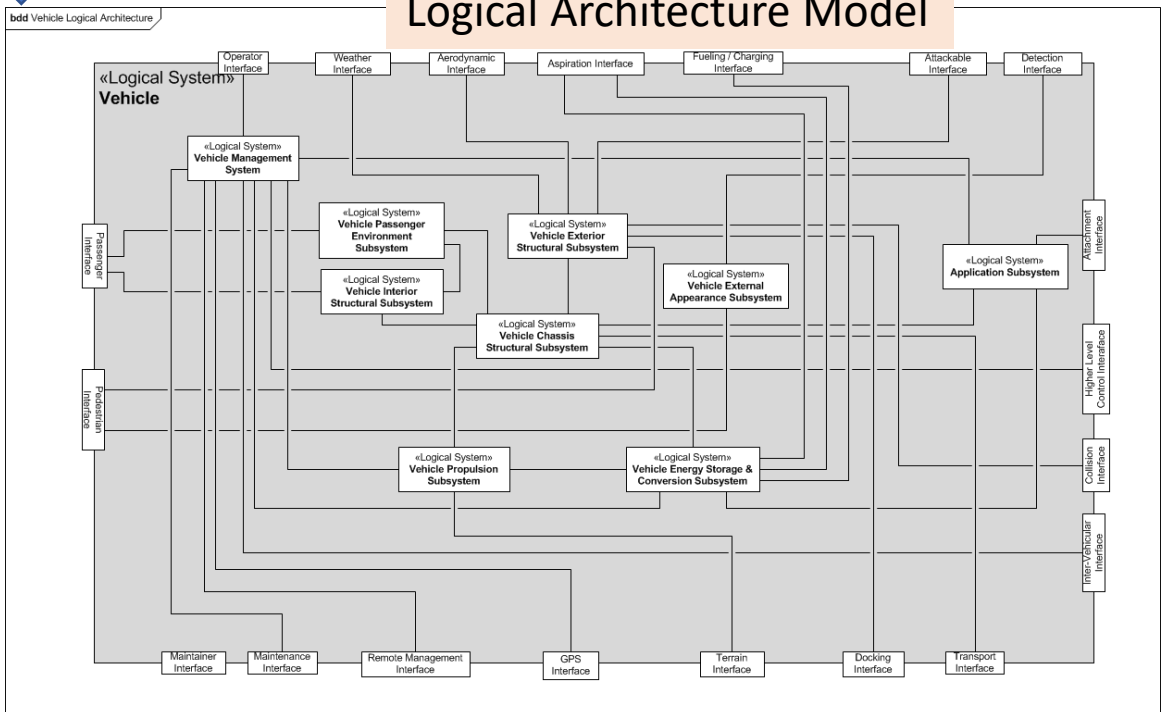
- **Domain Architecture Models**: How a System of Interest "*relates*" to its external environmental Actors; how the Domain System is *organized*;
- **Logical Architecture Models**: How the sets of decomposed external behavior components are *related* to each other as Logical (behavioral) Subsystems; how the logical architecture is *organized*;
- **Physical Architecture Models**: How the Design Components of a system are *related* to each other (spatially, by connections, or otherwise); how the physical architecture is *organized*;

Collectively, these describe the "*architecture*" (organization) of their respective subject matter.

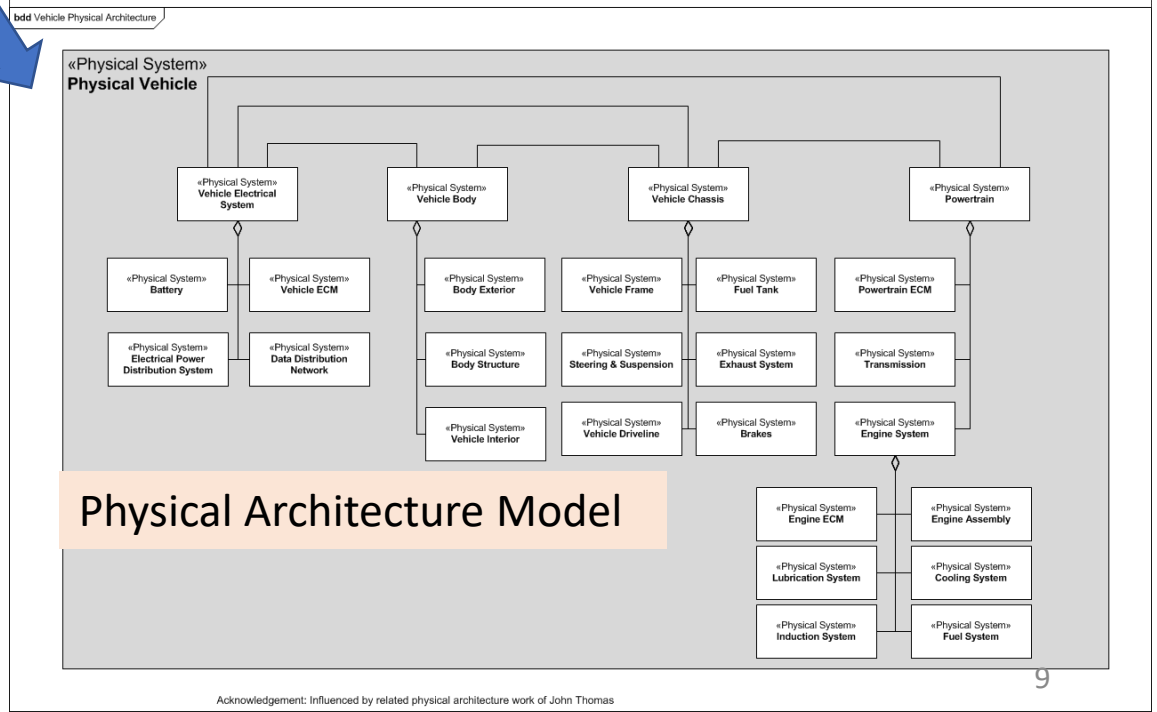
Domain Architecture Model



Logical Architecture Model

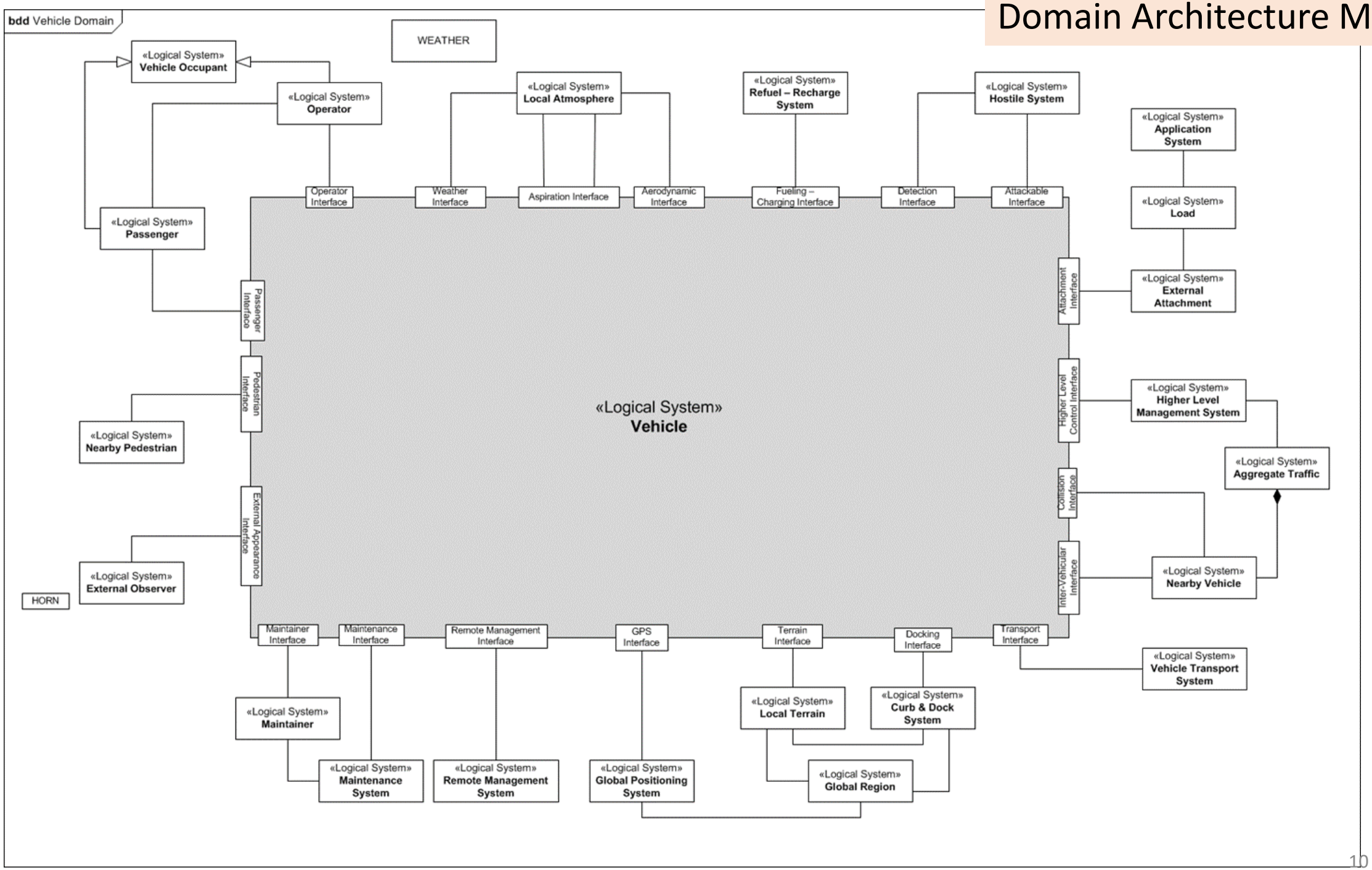


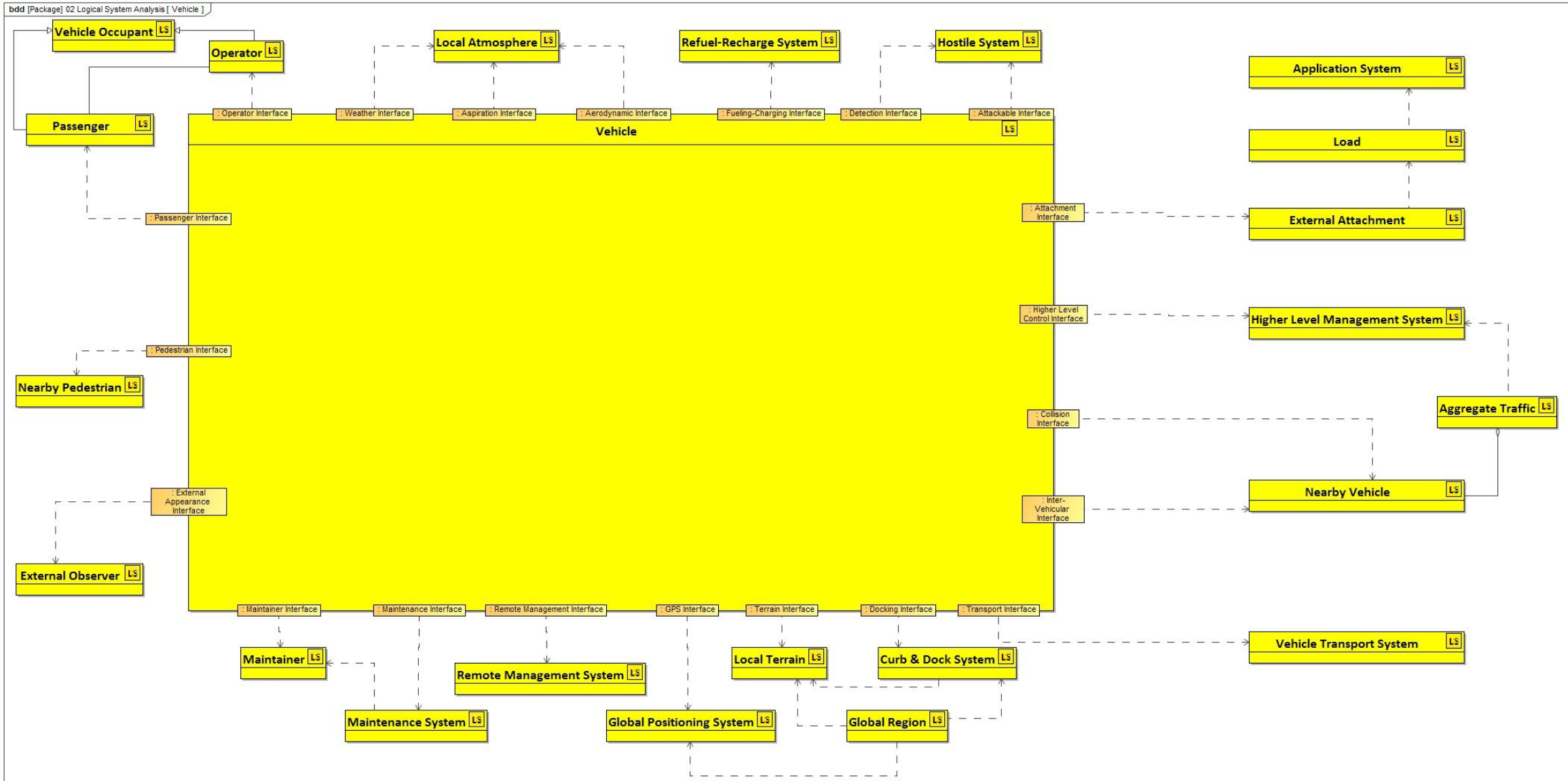
Physical Architecture Model



Acknowledgement: Influenced by related physical architecture work of John Thomas

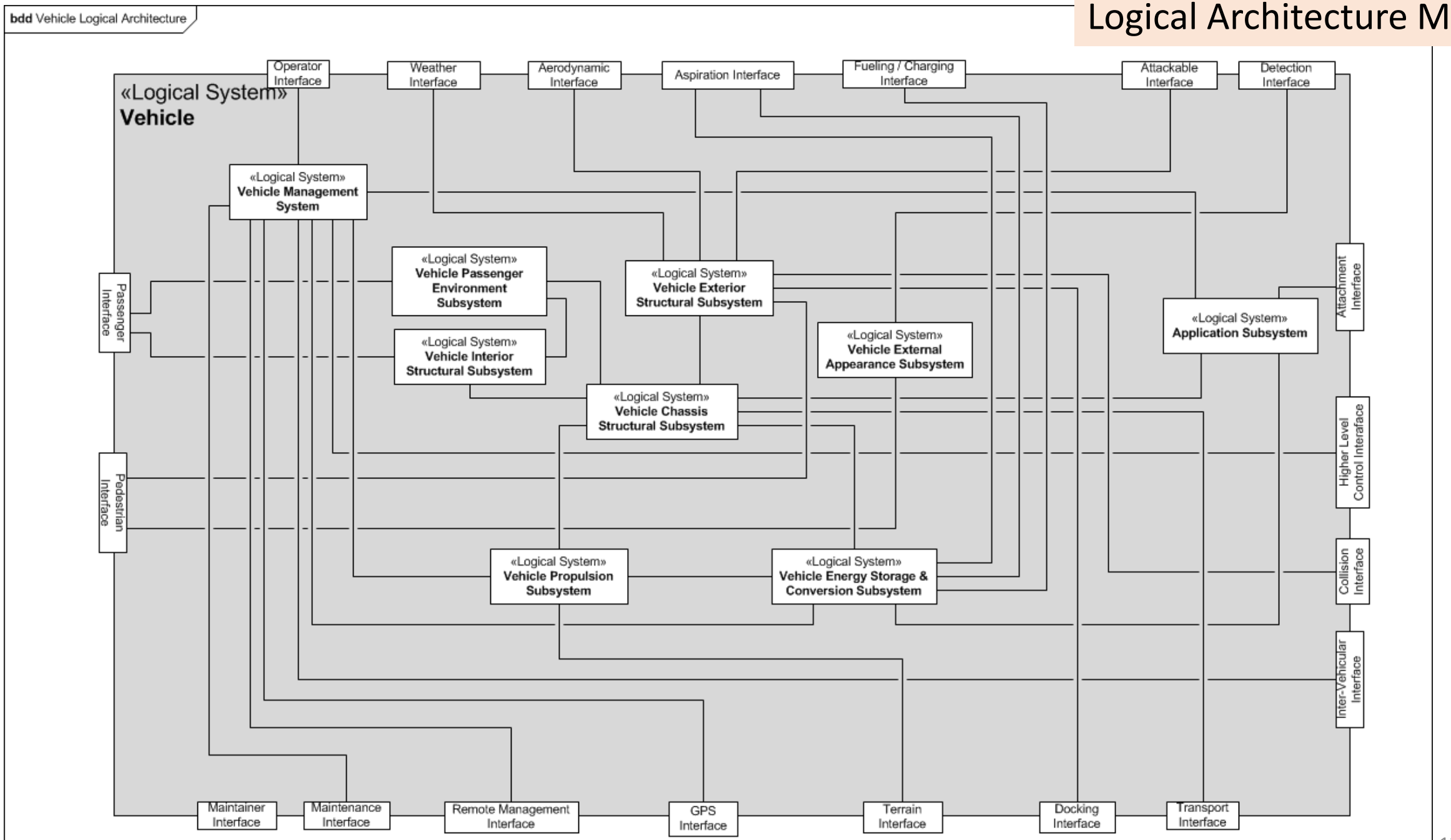
Domain Architecture Model

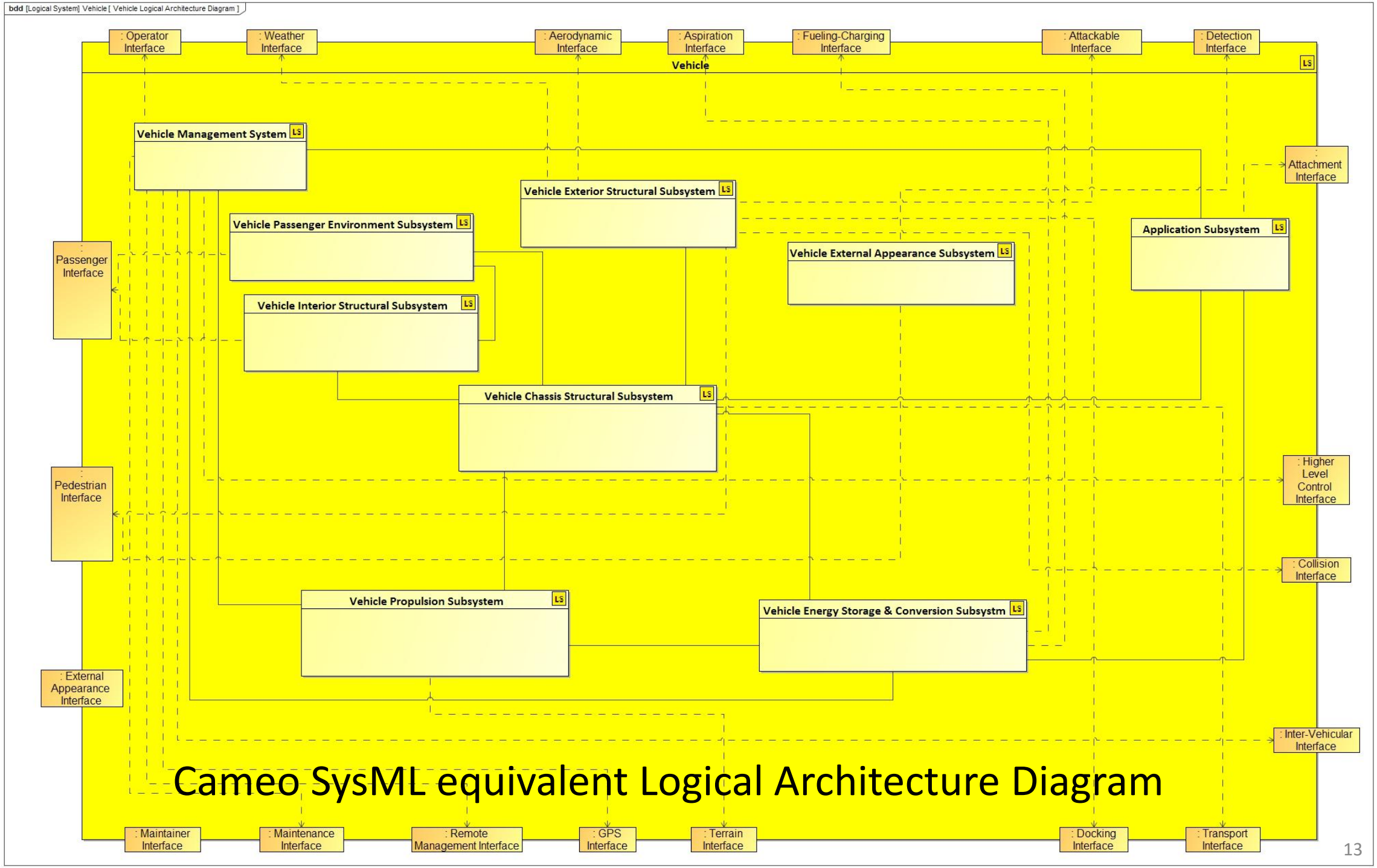




Cameo SysML equivalent Domain Architecture Diagram

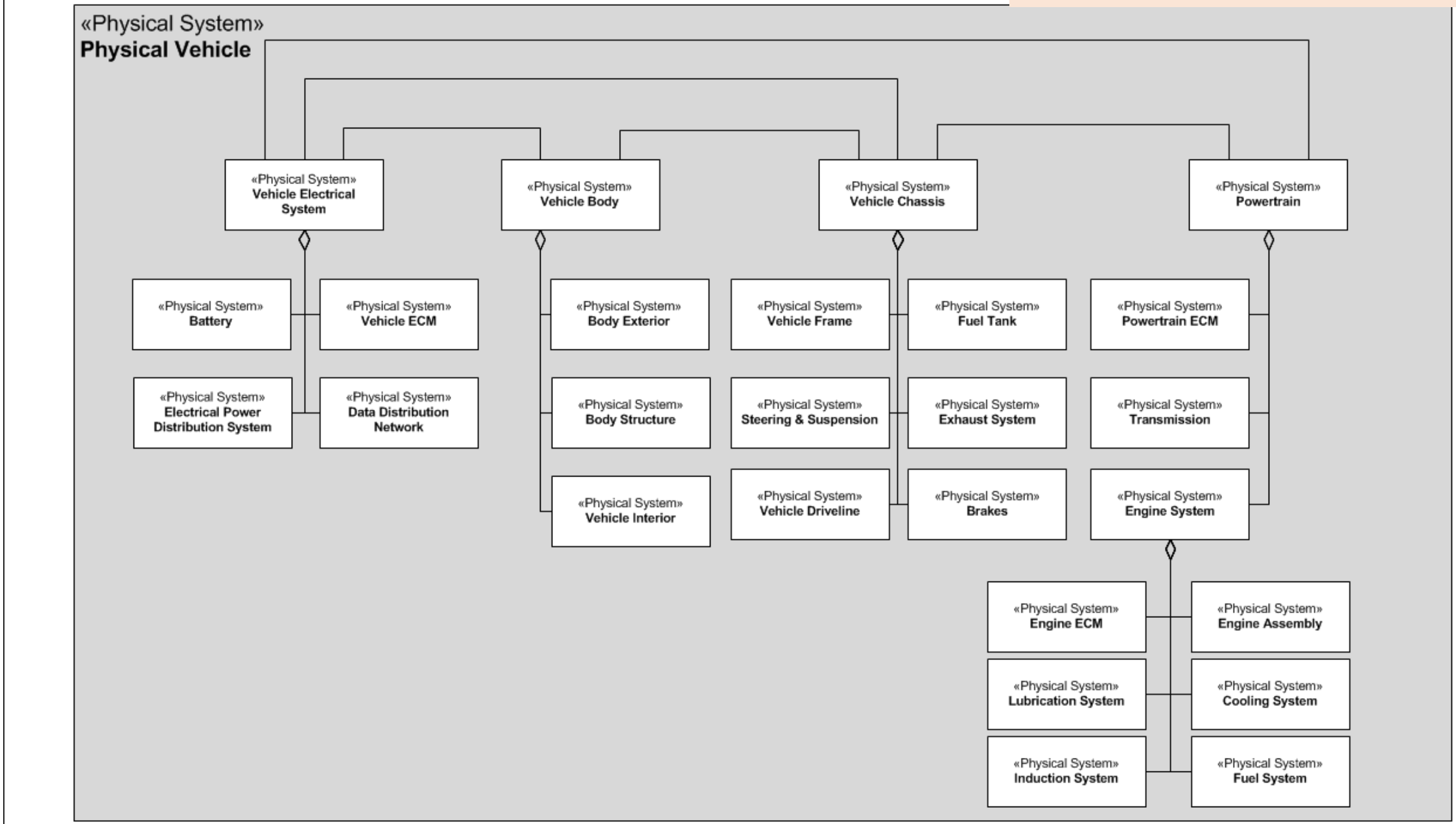
Logical Architecture Model





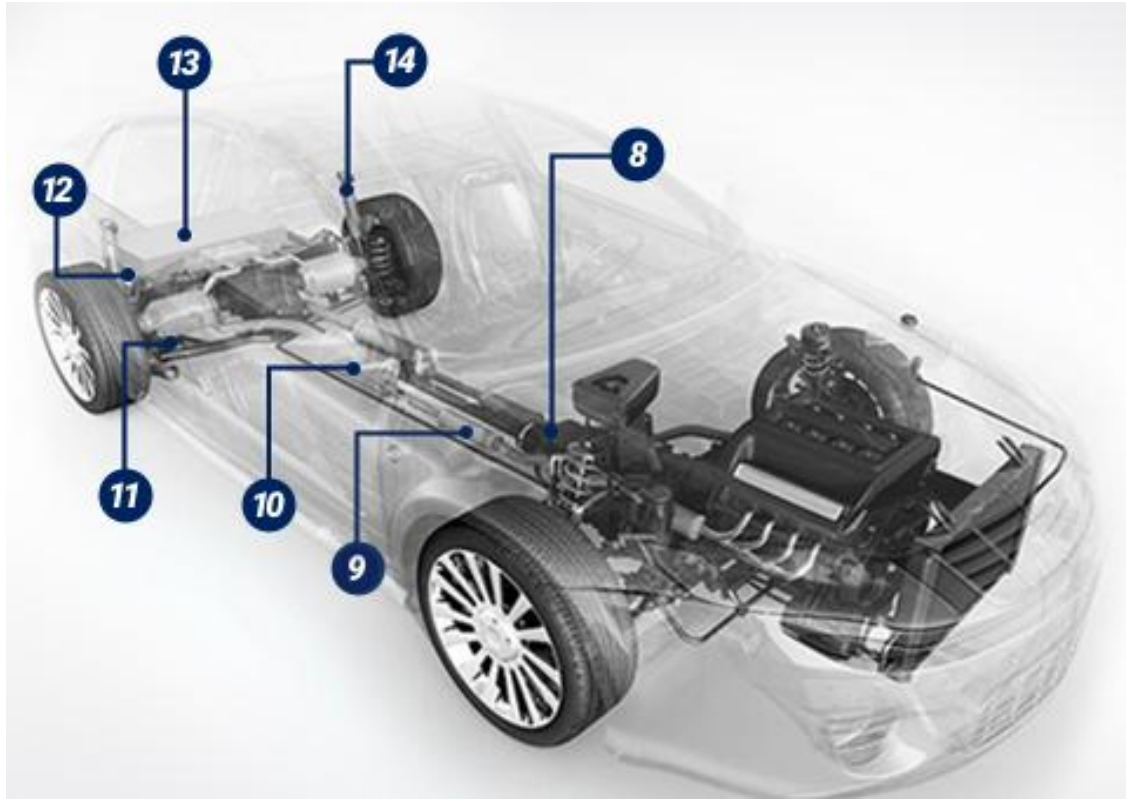
Cameo SysML equivalent Logical Architecture Diagram

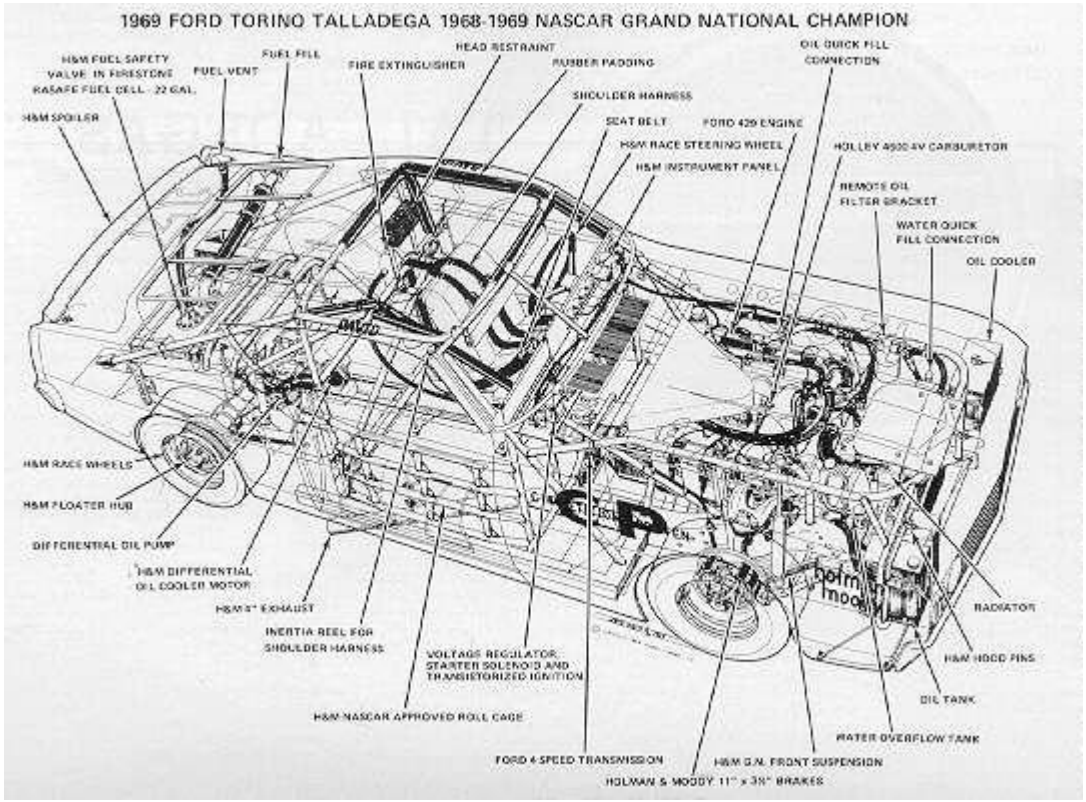
Physical Architecture Model



Acknowledgement: Influenced by related physical architecture work of John Thomas

Sometimes Physical Architecture Diagrams "look like the real thing"—even though the "things" in them are still entities in the model.





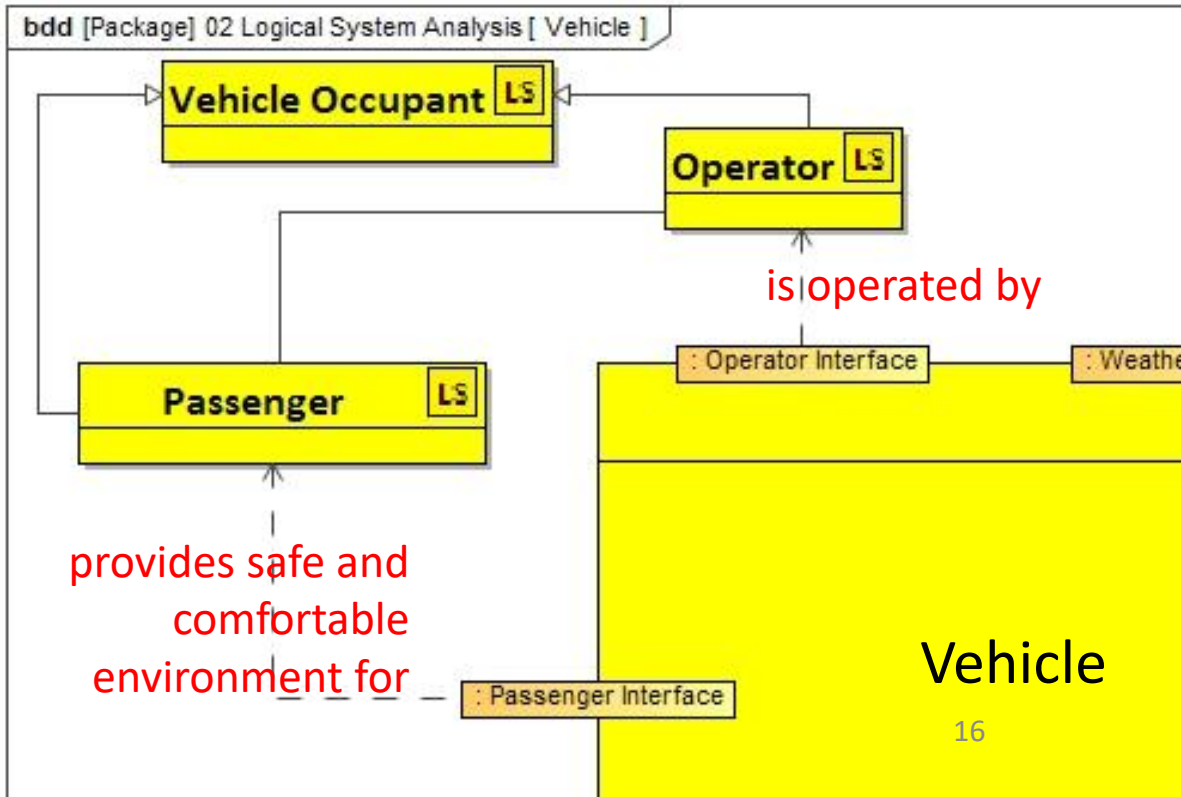
Naming Architectural Relationships

- Following conventions dating back to Peter Chen E-R models, it is common to name an architectural relationship with a verb form phrase, so that combining the name of the relationship with the names of the connected entities (usually nouns) creates a sentence summarizing the relationship:

"The **Vehicle** *is operated by* the **Operator**."

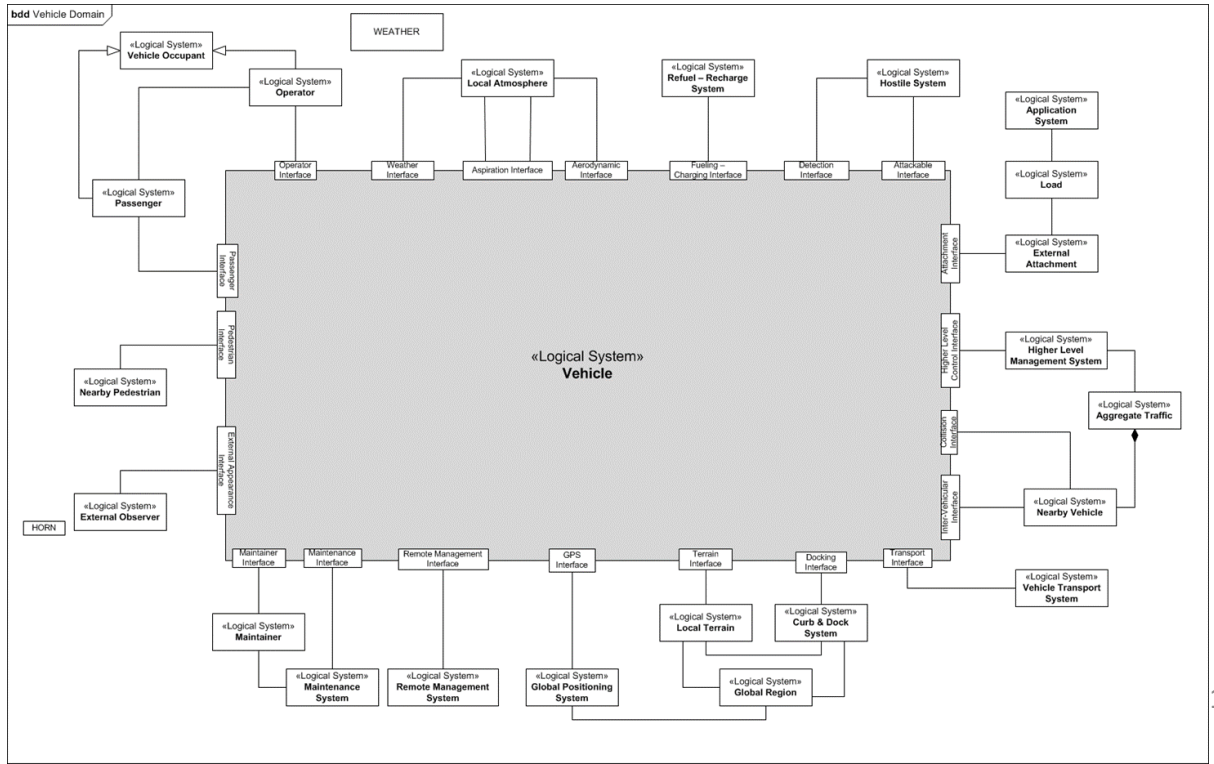
"The **Vehicle** *provides safe and comfortable environment for* the **Passenger**."

This can also provide a bridge later to the propositional logic of semantic technologies, for reasoning about the model in OWL DL or other semantic language—the instantiation of the relationship can be viewed as the "assertion" that the statement it makes about the entities is true.
(A vision of SysML V2.0 – OWL integrated environment.)



Architectural Relationships
Summarize Detailed Interactions, IOs & Requirements

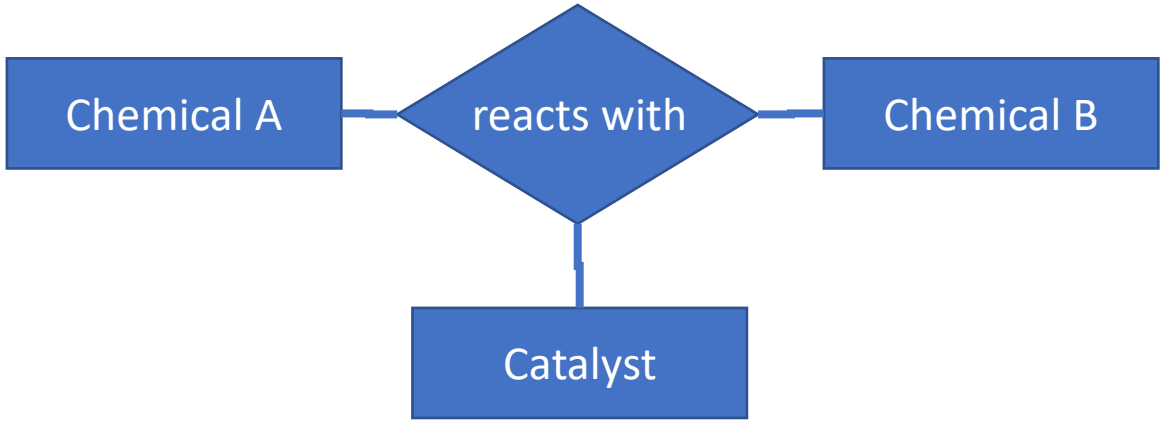
- Listing all the detailed external Interactions, Input-Outputs, and related Requirements would take up much more space and attention than allowed by one diagram—
  - But formally, those Relationships should be a traceable summary of the Interactions, IOs, and Requirements.



Not all Architectural Relationships are 2-way

- The Architectural Relationship examples we saw above were between <u>two</u> things:
  - That is why they are sometimes called “binary” relationships, or 2-way relationships.

- However, some relationships are <u>inherently 3-way</u>, or even more (n-way), and diagrammed with a connectable shape such as a diamond, instead of just a single line:

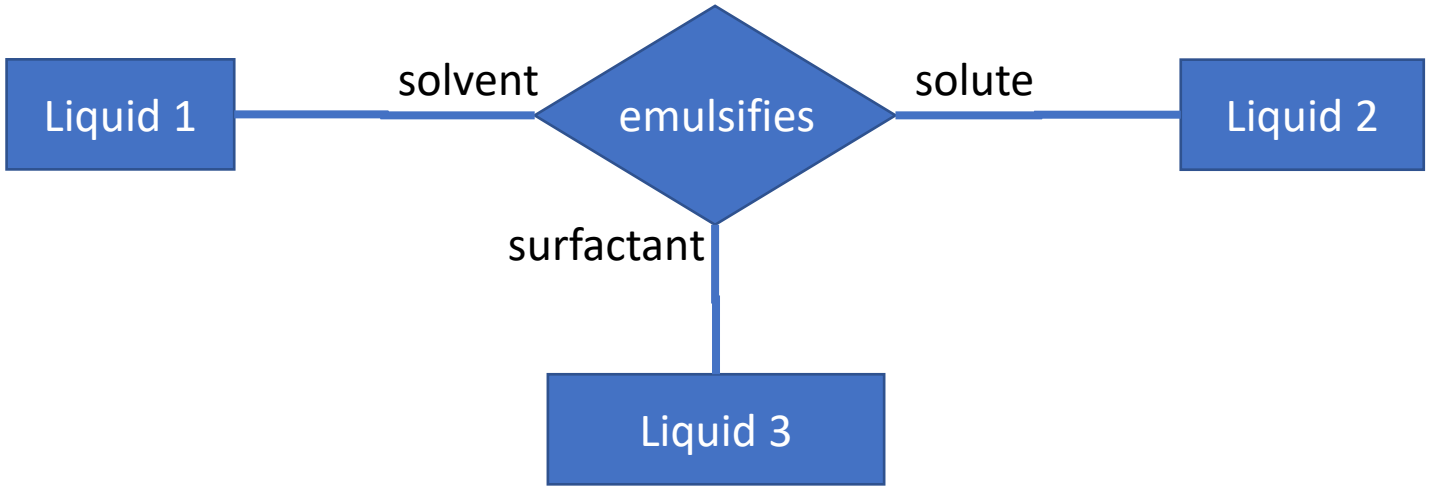


Examples:

- The relationship between two interacting chemicals and a catalyst that enables their reaction: “*reacts with*”
- The relationship between three connected structural trusses “*transmits structural forces between*”

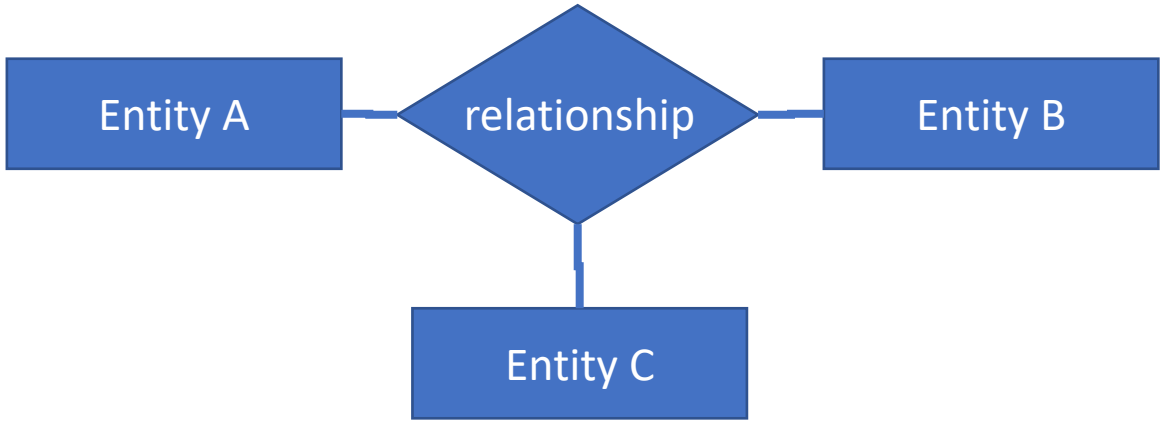
Not all Architectural Relationships are 2-way

- In those cases, it is a common practice to name the different “roles” of the relationship, so that the overall relationship is fully understood. For example:
  - solvent role
  - solute role
  - surfactant role
- Notice that this begins to suggest or imply an Interaction of the three, but is still less detailed and explicit than the interaction model it summarizes.



Reified relationships (not a way to cook beans!)

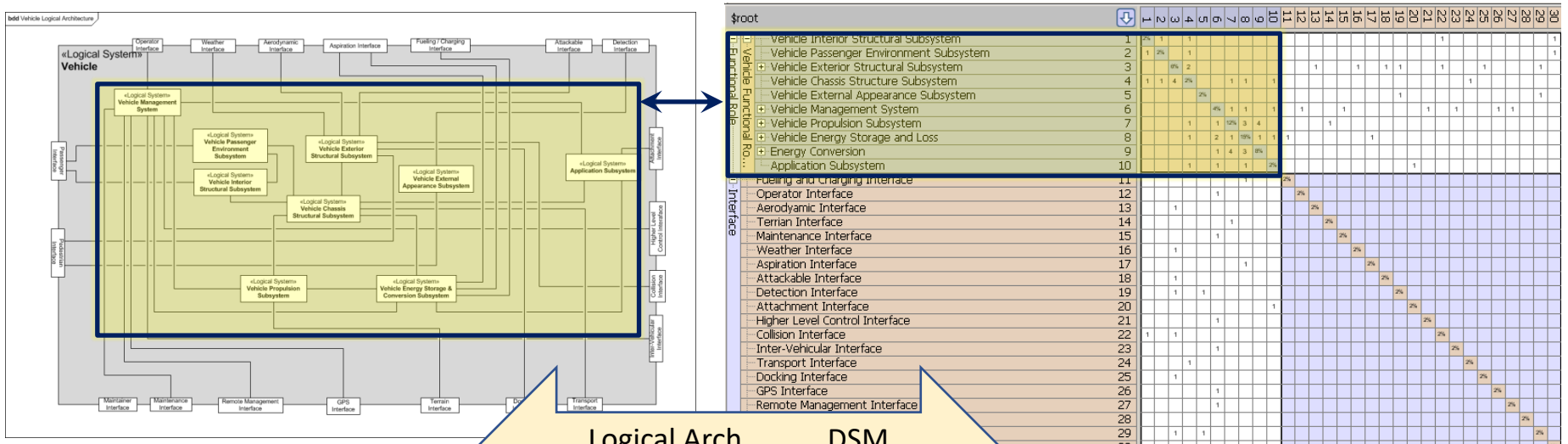
- Sometimes modelers will begin to refer to such an n-way relationship as “reified” (made real), or as a “class relationship”, in the sense that the relationship itself has become a modeled class or entity in its own right.
- Such reification can also be helpful (even for 2-way relationships!) if we want the relationship to have its own attributes (parameters) or to be allocatable.





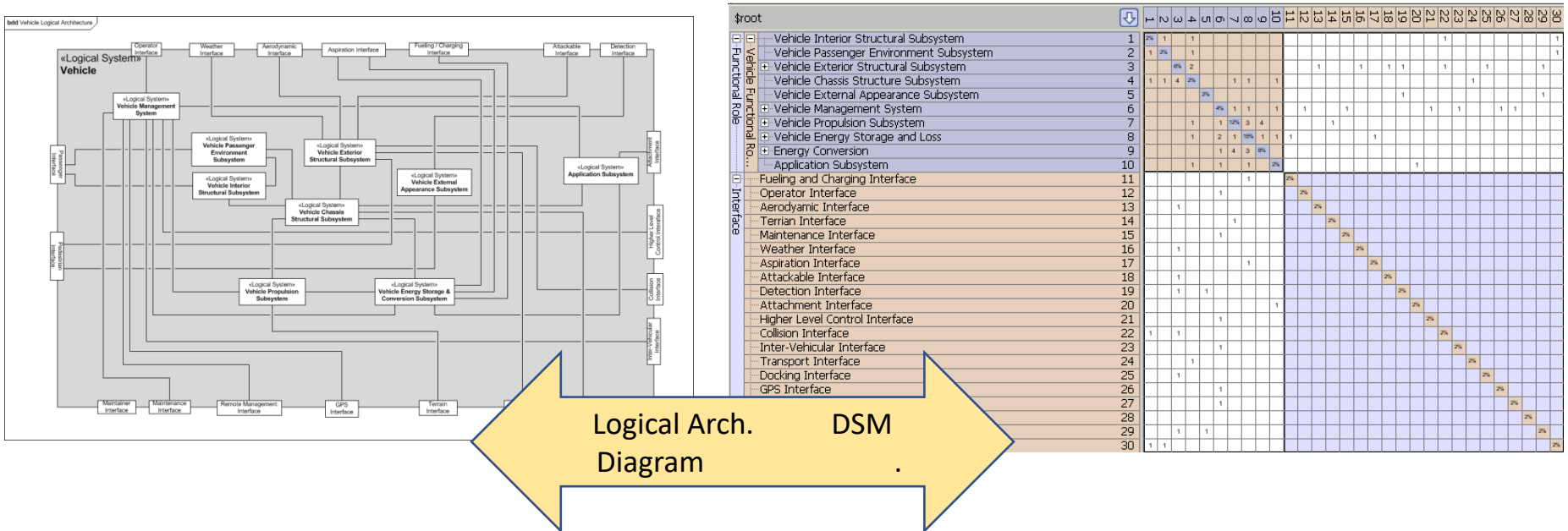
Logical Architecture Views

- In systems engineering, the structure shown in the above architectural diagrams can also be expressed in matrix form:
  - These matrices are known as: N^2 matrices, Adjacency Matrices and Design or Dependency Structure Matrices (DSMs)
  - N^2 when their column and row headings are identical, with the matrix cells showing “marks” indicating relationships between components.



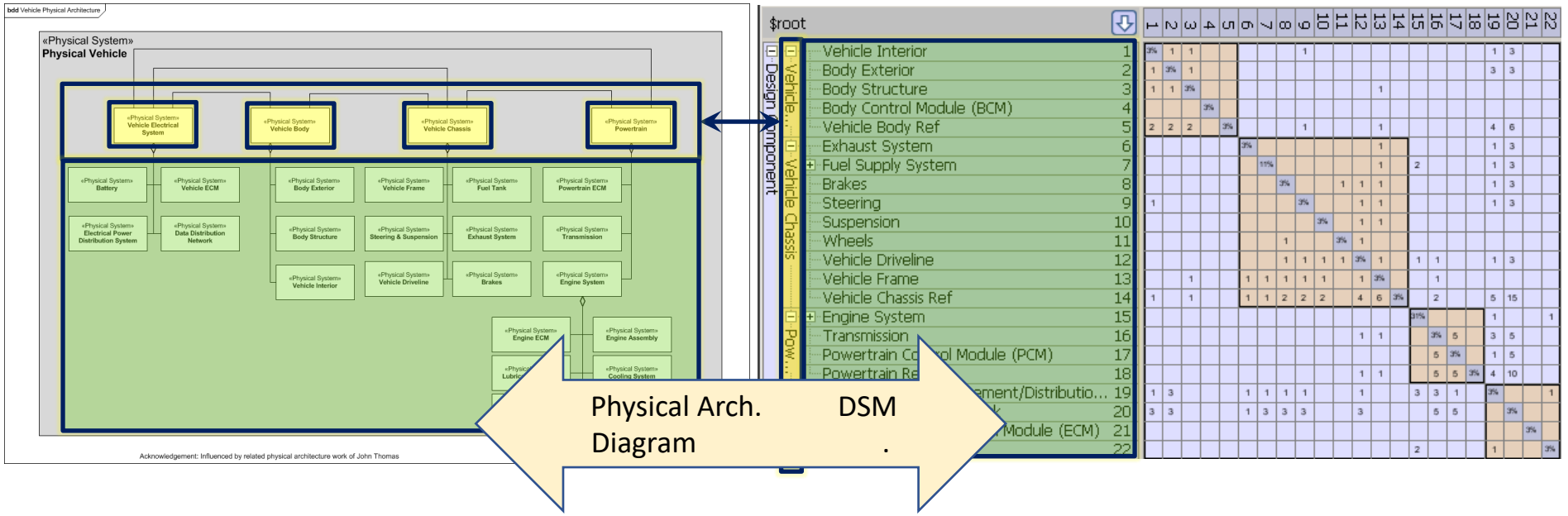
Logical Architecture Views

- In the case of Logical Architecture:
  - The blocks in the LA diagram become rows and columns of the DSM
  - The connection lines in the LA diagram become marks in the DSM
- Both views are visualizations of the same information:
  - However the functionality has been partitioned into interacting subsets – Vehicle Functional Roles and Interfaces in this case.



Physical Architecture Views

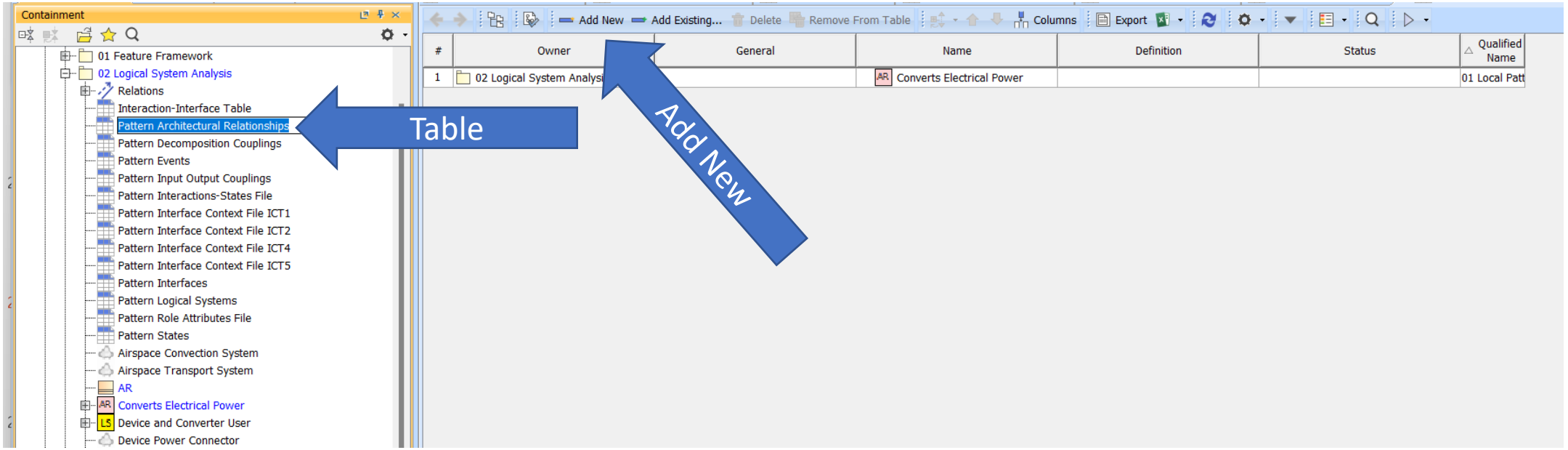
- In the case of Physical Architecture:
  - The blocks in the diagram become rows and columns of the DSM
  - The connection lines in the diagram become subsystems or components in the DSM shown in rows and columns
- Both views provide visualizations of hierarchy
  - How the physical system has been partitioned into physical sub-systems that are physically related (connected, contained, adjacent, etc.)
  - The DSM additionally shows the interactions of subsystems



Simple exercise: Populating architectural relationships

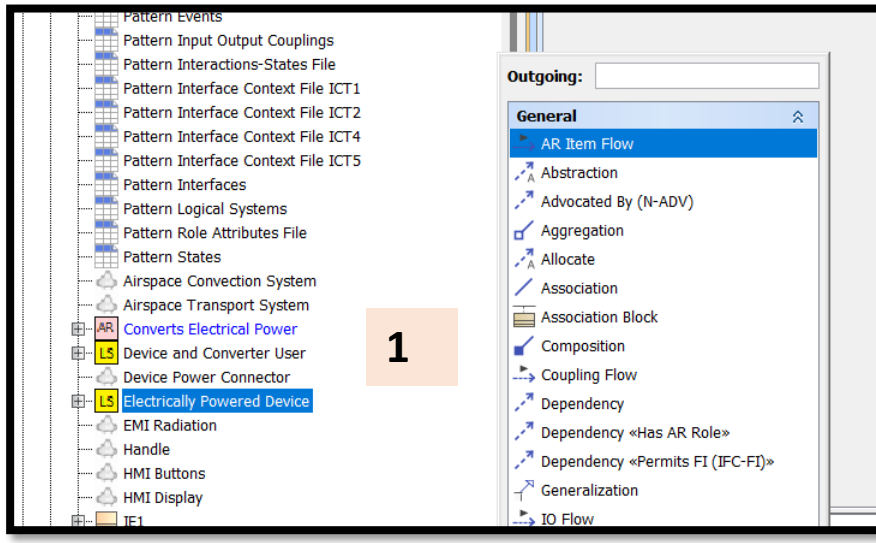
- Between two Logical Systems
- Between two Design Components

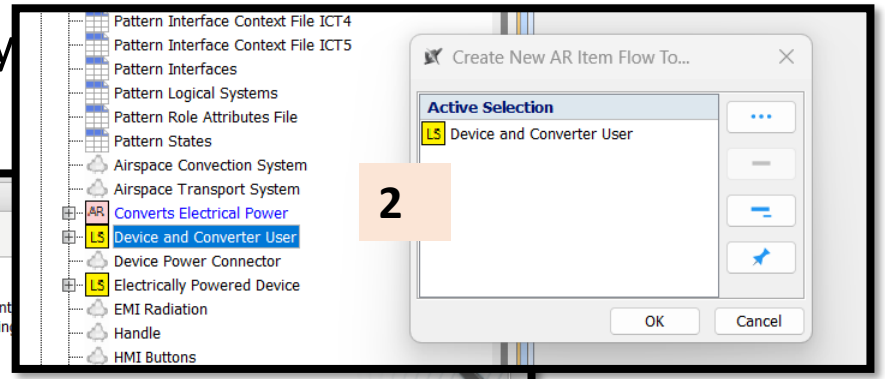
- Architectural Relationships are created in package 01 Local Pattern::02 Logical System Analysis.
- Select and Open the Pattern Architectural Relationships Table and then Select Add New.
- The newly added Architectural Relationship will show up in the table and the Containment Browser

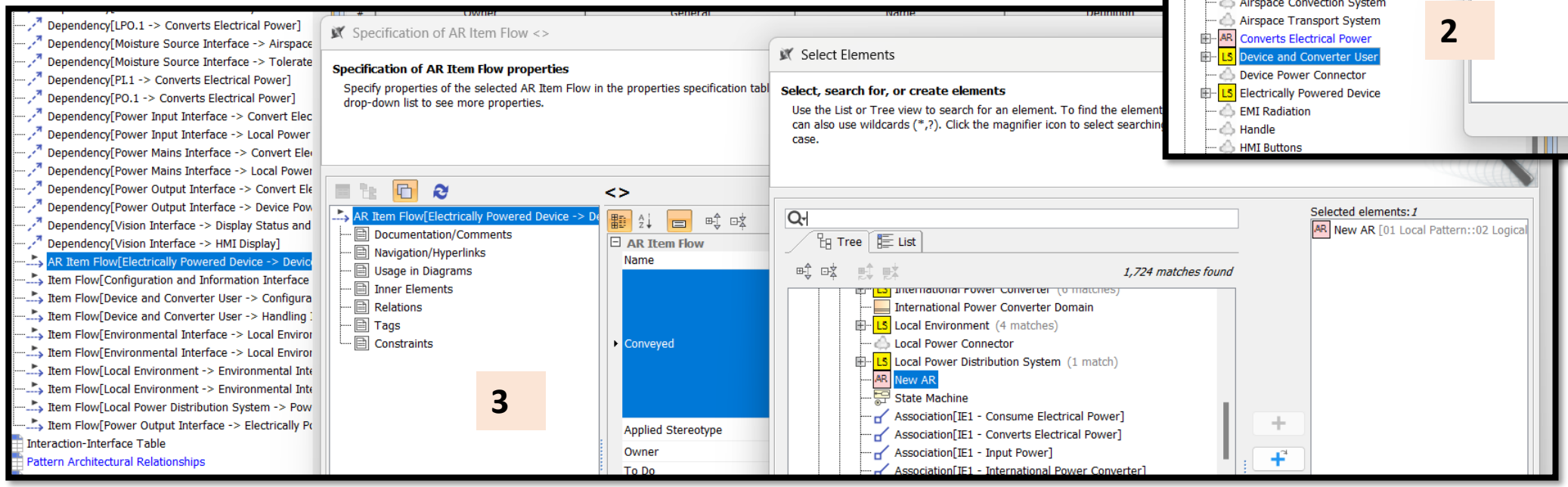


Architectural Relationships Modeling: continued

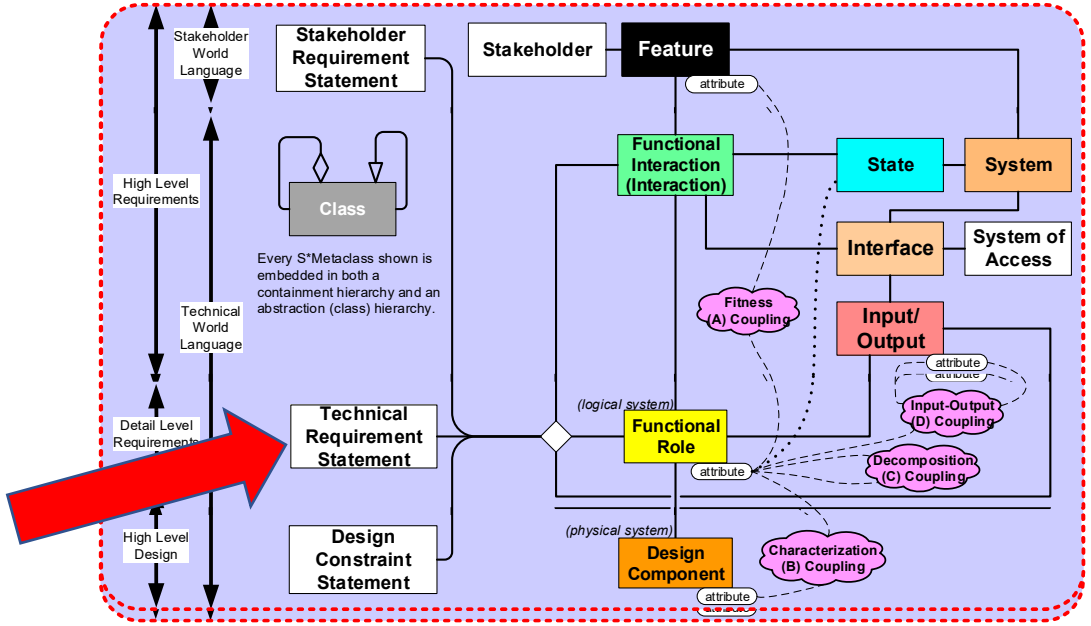
1. Select the Logical System or Physical System that conveys the Architectural Relationship > Create Relation > Select Outgoing or Incoming > Select AR Item Flow.
2. Select the other Logical System or Physical System involved in the Architectural Relationship.
3. Select the new AR Item Flow relation in the browser, open its specification, select the Conveyed property and choose the newly created Architectural Relationship.







Model-based Requirement Statements as transfer functions

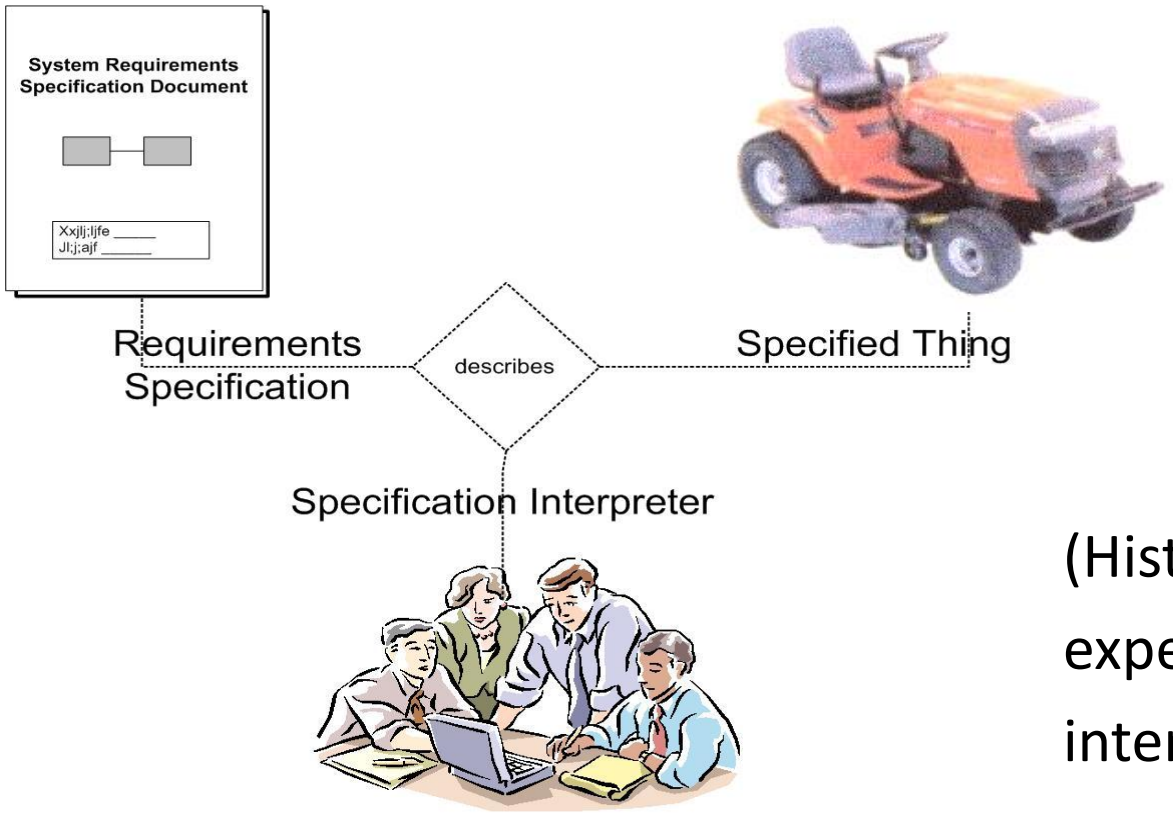


Traditional requirements statement writing discipline--

- Engineers are traditionally taught that requirements statements should be:
  - Unambiguous, Understandable, Atomic, Correct, Concise, Traceable, Design Independent, Verifiable, Unique, Complete, Consistent, Comparable, Modifiable, Attainable, . . .

Requirements are then . . .

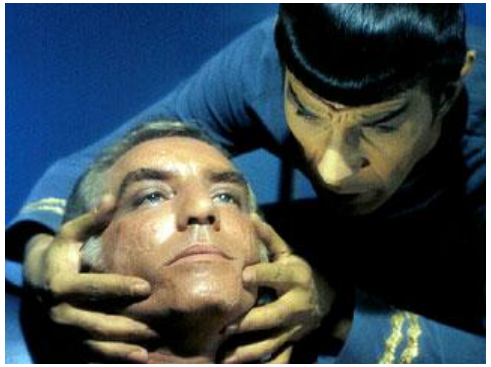
- Stored in databases, rendered in requirements documents, and interpreted by people



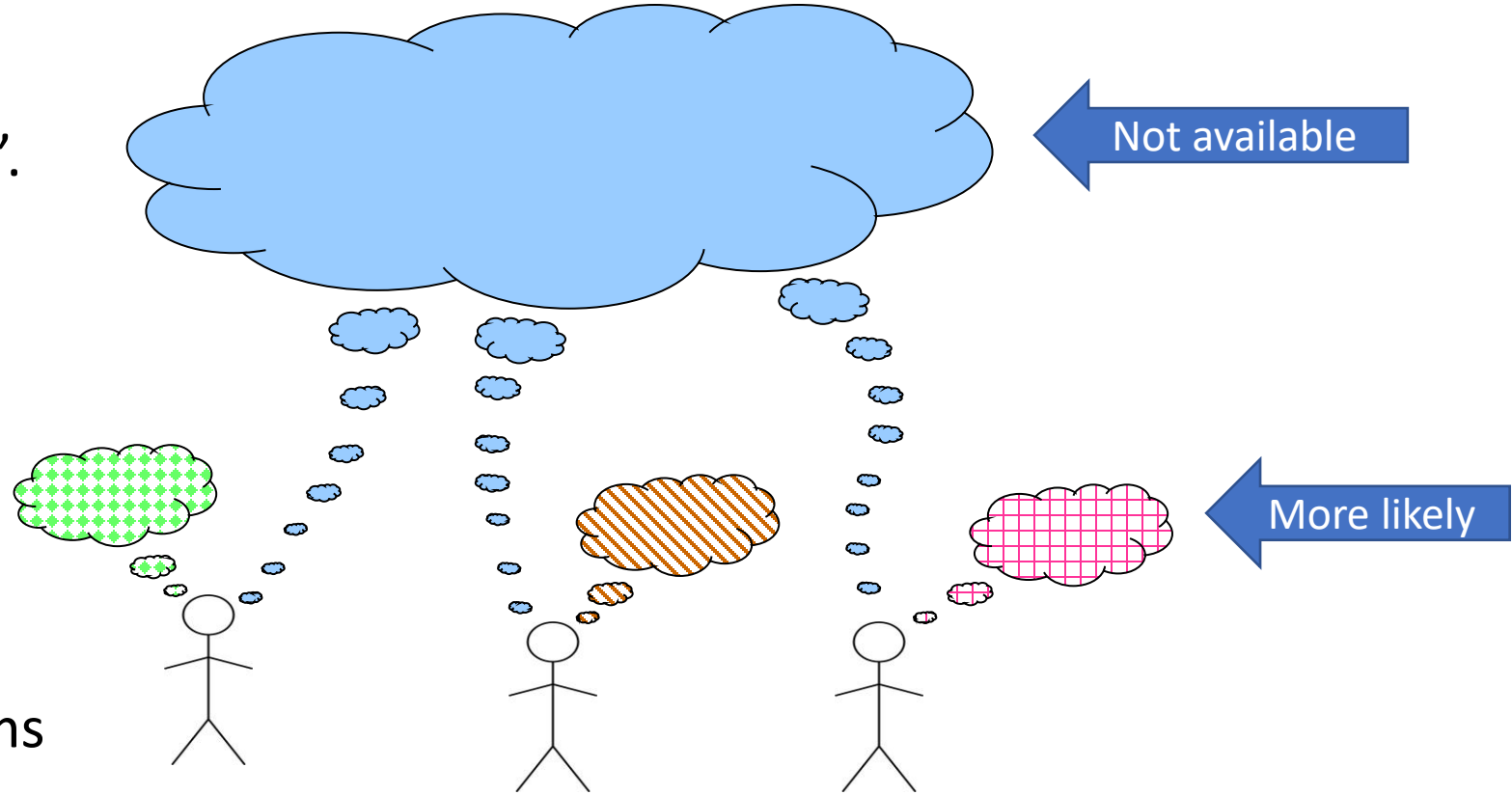
(Historical challenge: Three subject matter experts read a set of Requirements and interpret them three different ways.)

The challenge of complicated systems

- Success requires consistent team understanding of complicated systems—including Requirements.
- But, human biology does not support a "Vulcan Mind Meld".

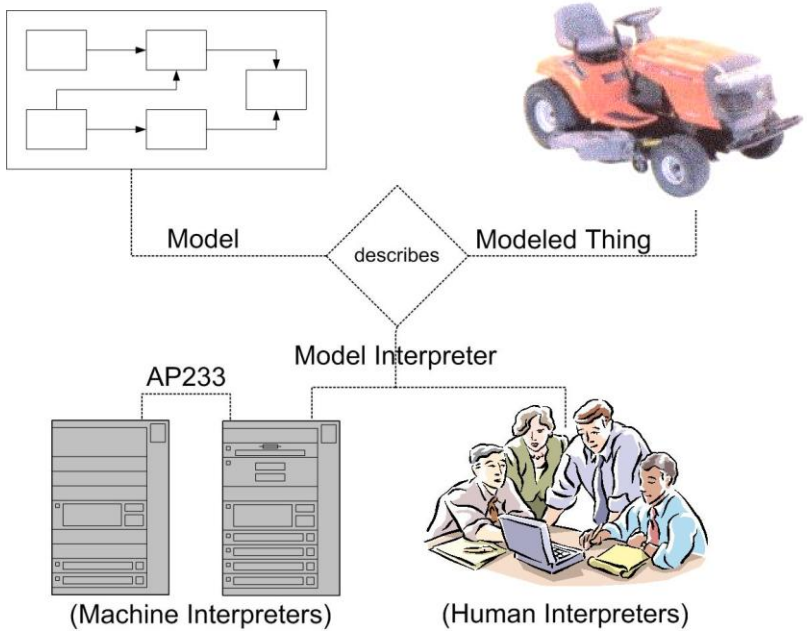


- So, how can we "integrate thinking" over teams in systems projects?



Models to the rescue?

- Models are data structures (often, but not always, graphically expressed) intended to explicitly represent facts (e.g., requirements, designs, etc.) about systems . . .
- That might otherwise be implicit in the knowledge of experts or others.
- Compared to traditional prose, models frequently:
  - Are more explicit, more compact, enhance visualization, understanding, communication, improve theoretical support for understanding, improve ability to analyze, simulate, or synthesize, increase sharing across people and tools, . . .

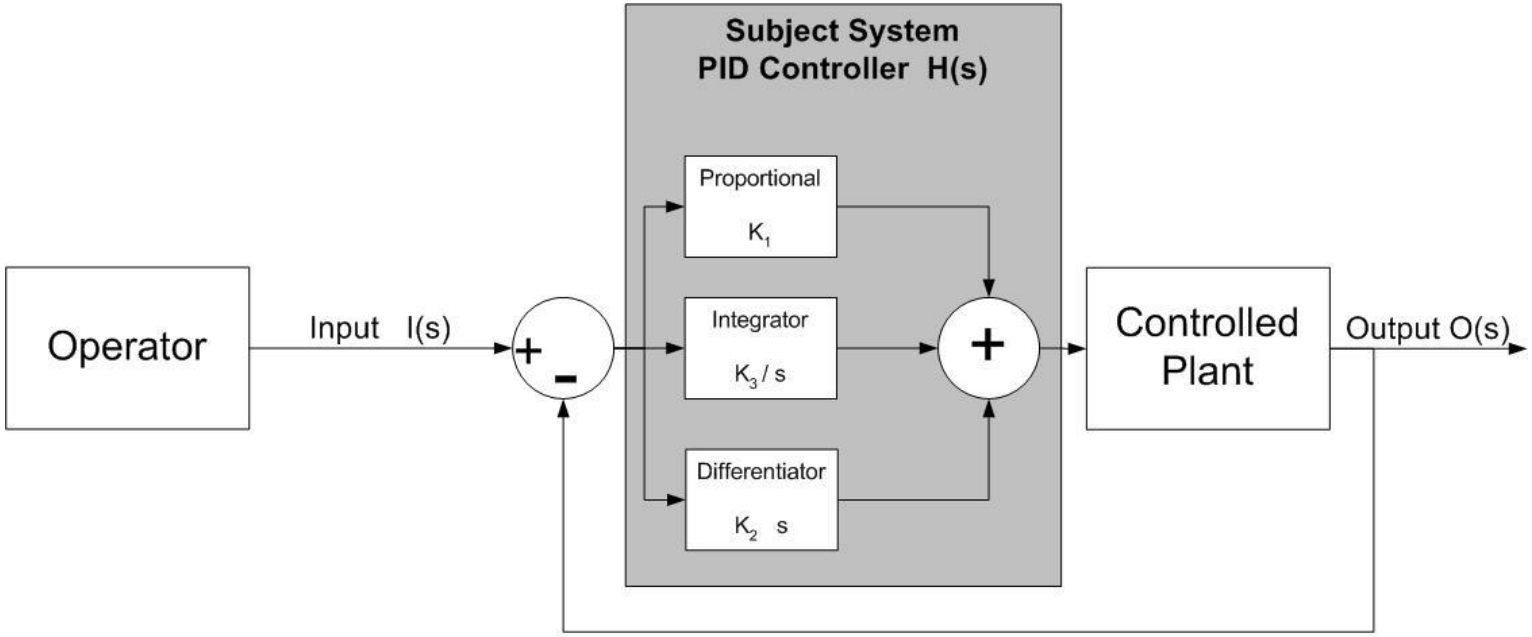


Embedding the prose in the models

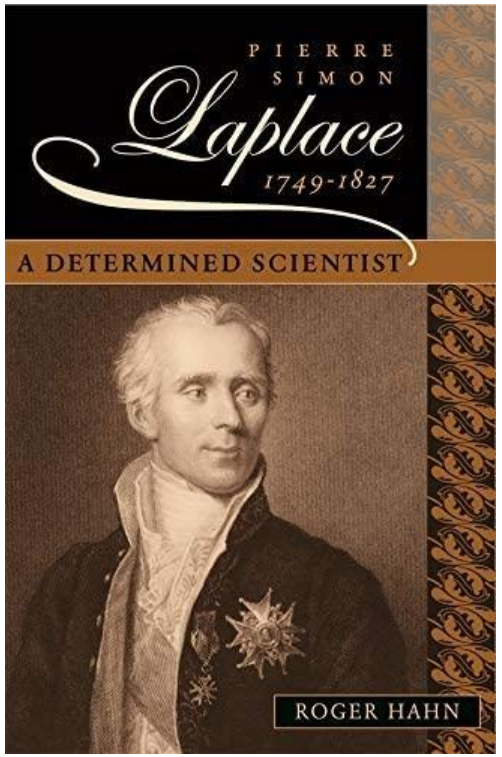
- Our experience is that models don't replace prose Requirement Statements:
  - They can be merged into a total model that includes both.
  - The Requirement Statements become part of the model.
  - Contemporary efforts to incorporate prose into models includes language support (SysML) and information models (AP233).
- Our interest here is something more than how to imbed prose statements into model data structures, or languages for doing so:
  - What does it "really mean" to bring these together?
  - Our inspiration comes from examining the special semantics of Requirement Statements as a special subset of all prose.
- Just treating Requirement Statements as prose "text" misses this point:
  - And there is a huge payoff to what we discover . . .

Transfer functions, as a source of inspiration

- Transfer functions completely characterize the externally viewed (black box) behavior of idealized linear systems:

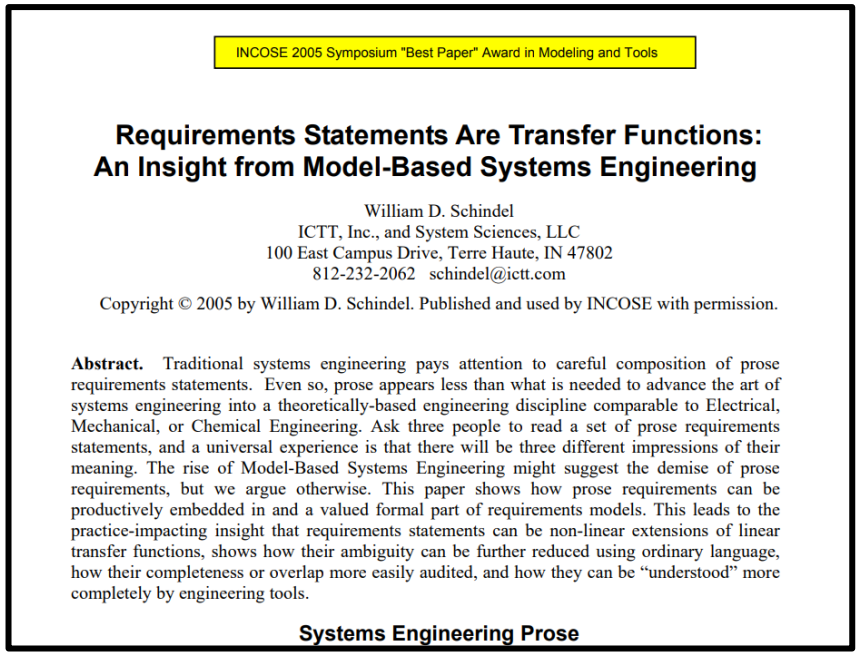


$$H(s) = (K_1 + (K_2 \ s) + (K_3 / s))$$



But, our systems aren't linear!
(And our stakeholders don't speak math!)





INCOSE 2005 Symposium "Best Paper" Award in Modeling and Tools

Requirements Statements Are Transfer Functions: An Insight from Model-Based Systems Engineering

William D. Schindel
ICTT, Inc., and System Sciences, LLC
100 East Campus Drive, Terre Haute, IN 47802
812-232-2062 schindel@ictt.com

Copyright © 2005 by William D. Schindel. Published and used by INCOSE with permission.

Abstract. Traditional systems engineering pays attention to careful composition of prose requirements statements. Even so, prose appears less than what is needed to advance the art of systems engineering into a theoretically-based engineering discipline comparable to Electrical, Mechanical, or Chemical Engineering. Ask three people to read a set of prose requirements statements, and a universal experience is that there will be three different impressions of their meaning. The rise of Model-Based Systems Engineering might suggest the demise of prose requirements, but we argue otherwise. This paper shows how prose requirements can be productively embedded in and a valued formal part of requirements models. This leads to the practice-impacting insight that requirements statements can be non-linear extensions of linear transfer functions, shows how their ambiguity can be further reduced using ordinary language, how their completeness or overlap more easily audited, and how they can be "understood" more completely by engineering tools.

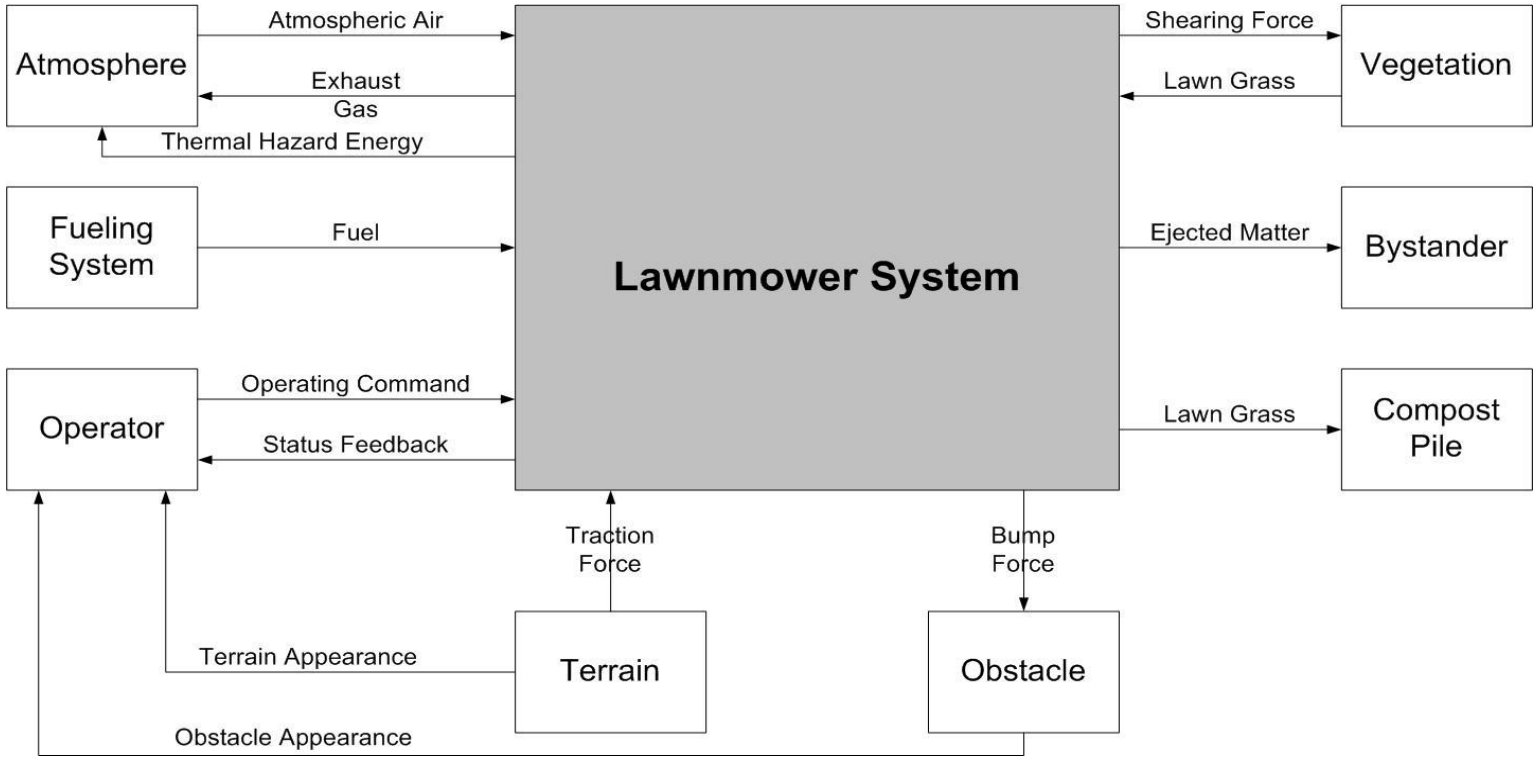
Systems Engineering Prose

https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:incose_2005.pdf

- Nevertheless, we can borrow a key idea from the "transfer function perspective":
  - Characterizing a system's behavior by stating the externally visible relationships between its inputs and outputs
  - (Think about this for systems you know—test the idea)
- Why is characterizing behavior in this way important?

What are functional requirements, really?

The technical (engineer's language) description of testable external black box functional requirements is entirely about its interactions (in a physics sense) with external systems:



What about stakeholder and "non-functional" requirements?

- Two issues suggest we don't traditionally think about requirements statements that way:
  - Stakeholder requirements are usually non-technical language, more suited to QFD coupling, etc.
  - The "-ilities" (reliability, manufacturability, maintainability, etc.), as well as capacity, tolerance, or other requirements categories may not seem to fit this approach.
- However:
  - We traditionally derive "technical requirements" from the stakeholder requirements, for purposes of design, test, etc.
  - And all the "-ilities" (and their cousins) turn out to be externally visible behavioral attributes of the functional (physical interactions) behavior.
  - (Think about it . . .)

Requirements from interactions?

- So, we can legitimately focus our attention on the externally visible interaction behavior, described as input-output relationships over time.
- This has major positive impacts:
  - To become a theory-grounded discipline (like EE, ME, ChE, CE), Systems Engineering needs a stronger theoretical base than principles of writing clear prose and business process rules: A physics of systems engineering
  - Interactions are at the theoretical heart of the 300 year success of science and most engineering disciplines
  - Our work as systems engineers can move farther from “literature writing” and closer to the scientific-mathematical edifice built by Newton, Hamilton, Maxwell, and others
  - And, it plays to the rise of modeling languages (SysML, etc.)

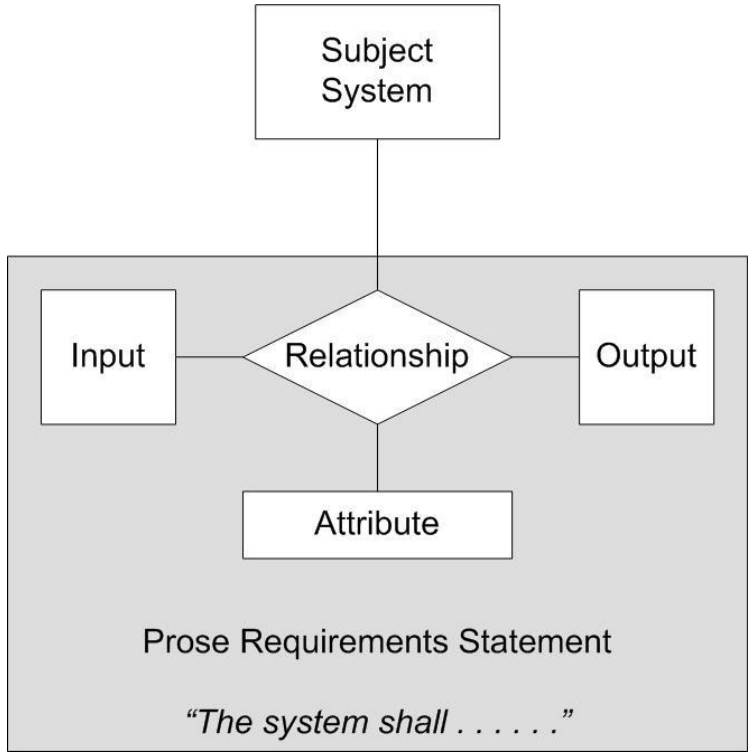
The return of prose to the model

- Requirements prose: for the purpose of expressing required system input-output relationships.
- This is not really as unconventional engineering as you might think—we already use many “equivalents” to mathematical equations to characterize I/O behavior:
  - VHDL for digital electronics
  - Propositional calculus for system logical behavior
  - Algebraic, differential, or integral equations
  - Fuzzy relationships (Zadeh)
  - Tables, graphs
  - And now, “prose requirements statements”

The return of prose to the model

- The first implication of this approach is the discovery that what every requirements statement should tell us is something about system Inputs, Outputs, how they are Related, and possible Attributes (parameterization) of that relationship:

  - Not every requirements statement needs to contain all of these.
  - But, every statement should contain some of them.
  - And the aggregation of these statements should form an "equation" characterizing the overall I/O relationship–or we are not done.
  - Note the relationships between IOs can be about relative flow quantities, time delays, statistics, losses, variations, etc.



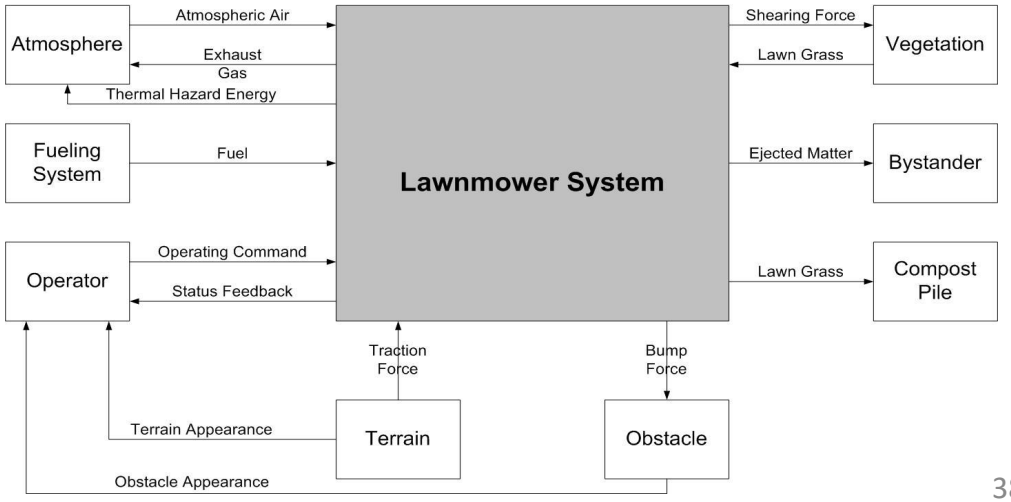
Example requirements statements in I/O form

1. "The Lawnmower System shall *operate* with [Hourly Mowing Capacity] of at least 1 level ground acre per hour, at [Max Elevation] up to 5,000 feet above sea level, and [Max Ambient Temperature] of up to 85 degrees F., at up to 50% [Max Relative Humidity], for [Foliage Cutting Capacity] of Acme American Standard one week Lawn Grass."
2. "The Lawnmower System shall *operate* using Fuel consisting of gasoline having a [Min Octane Rating] of not less than 92, combusted with Atmospheric Air."
3. "The Lawnmower System shall *operate* with [Fuel Economy] of at least 1 hour / gallon at [Min Elevation] of 0 feet ASL, at [Max Ambient Temperature] 85 degrees F., 50% [Max Relative Humidity], for Acme American Standard one week Lawn Grass."
4. "The Lawnmower System shall *operate* with [Elevation Derating] of 10% improvement in [Fuel] per 1,000 feet of elevation reduction, to a [Min Elevation] of 0 feet ASL.
5. "The Lawnmower System shall *operate* meeting the more demanding of state and federal standards for [Max Gaseous Pollution] and [Max Particulate Pollution] of Exhaust Gas.
6. "The Lawnmower System shall *operate* with [Operating MTBF] no less than 500 hours."
7. "The Lawnmower System shall *operate* so as to protect the Operator from Thermal Hazard Energy by maintaining all accessible metallic surfaces at a [Maximum Surface Temperature] of less than 180 degrees F."

To make the above clear:

- Inputs and Outputs are underlined.
- [Attributes] are in brackets.
- *Relationships* are italicized.

Just for this example—not required.

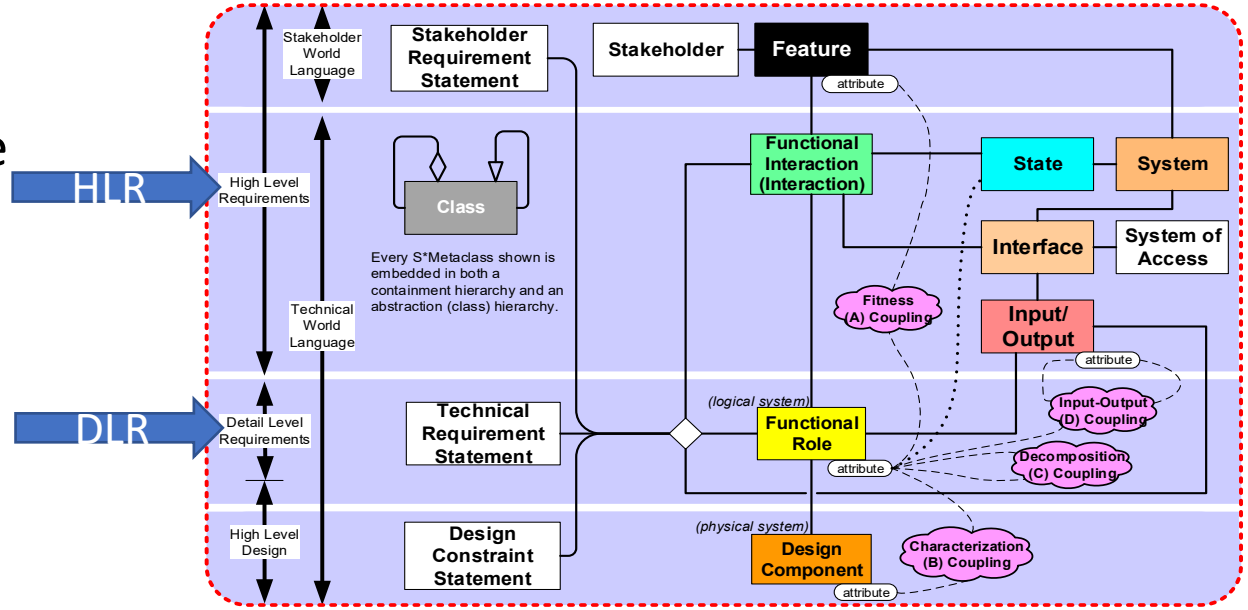


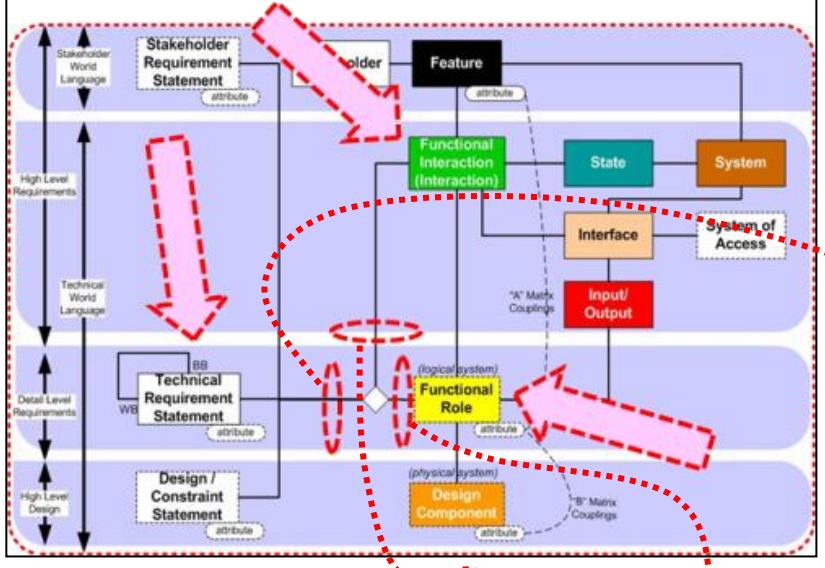
More implications

- Improved requirements statements: Ability to audit/inspect for completeness as well as clarity.
  - “Does this set of statements specify the required output values for the possible input values (and history)?”
- Implications for tools: Tools can better “understand” micro-structure of requirements.
- Implications for models: Improved explication compared to simply viewing requirements as embedded text.

Requirements Statements are "organized" by the other parts of the S*Metamodel

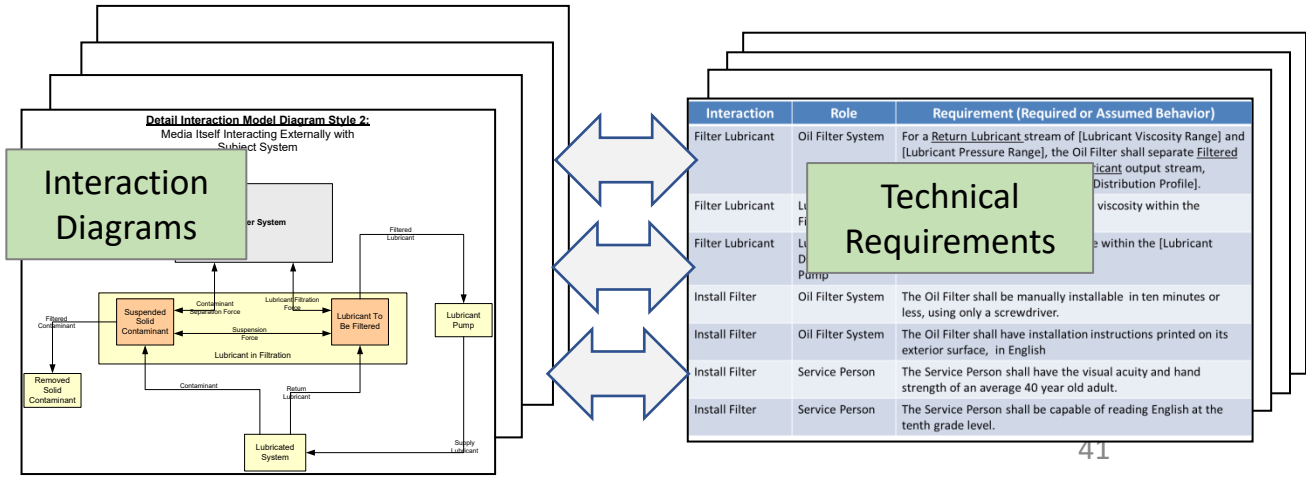
- Requirement Statements typically do not carry all the information found in State Models, Domain Models, etc.
  - And if they did, it is not clear that they would be very handy to understand in prose.
- So, a better way to think about this is that the High-Level Requirements (HLR) portion of an S*Model "organizes" the Technical Requirements of the Detail Level Requirements (DLR) portion of the S*Model.
- This is a reminder that we don't really fully understand Requirements if we don't have a domain and state model!





Interaction	Role	Requirement (Required or Assumed Behavior)
Filter Lubricant	Oil Filter System	For a Return Lubricant stream of [Lubricant Viscosity Range] and [Lubricant Pressure Range], the Oil Filter shall separate Filtered Contaminant particles from the Lubricant output stream, according to the [Filter Particle Size Distribution Profile].
Filter Lubricant	Lubricant in Filtration	The Lubricant in Filtration shall have viscosity within the [Lubricant Viscosity Range].
Filter Lubricant	Lubricant Distribution	The Pump shall maintain oil pressure within the [Lubricant Pressure Range]

- Notice that Interactions, Roles, and Requirements form a three-column table, in which each Requirement is allocated to a Functional Role, to occur during a Functional Interaction

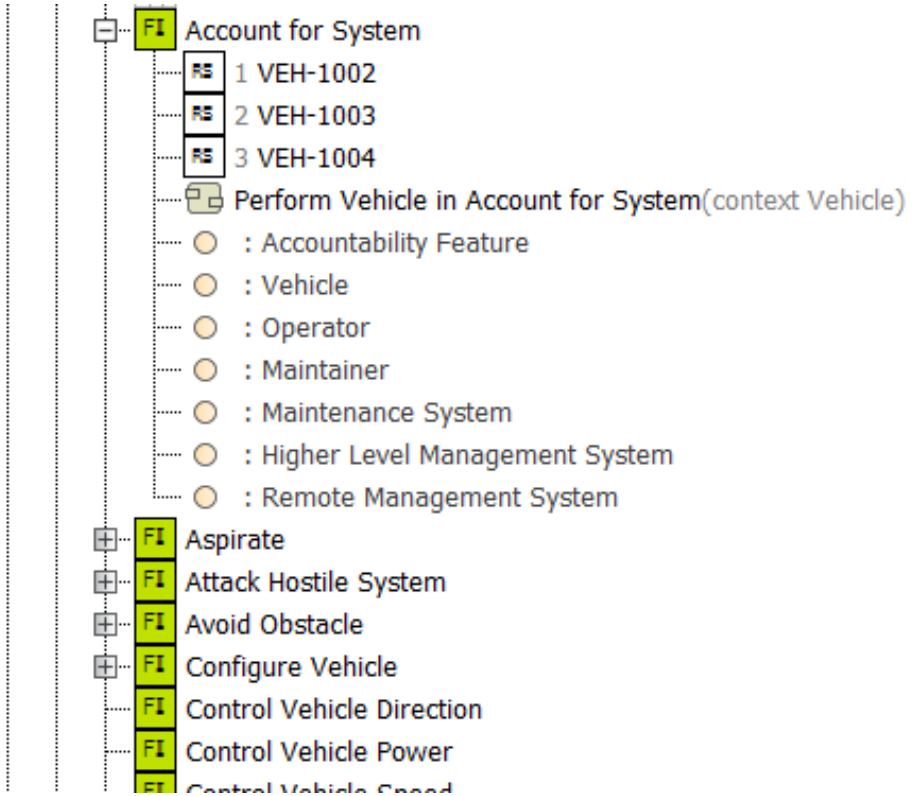


BREAK

Simple exercise: Populating Requirement Statements

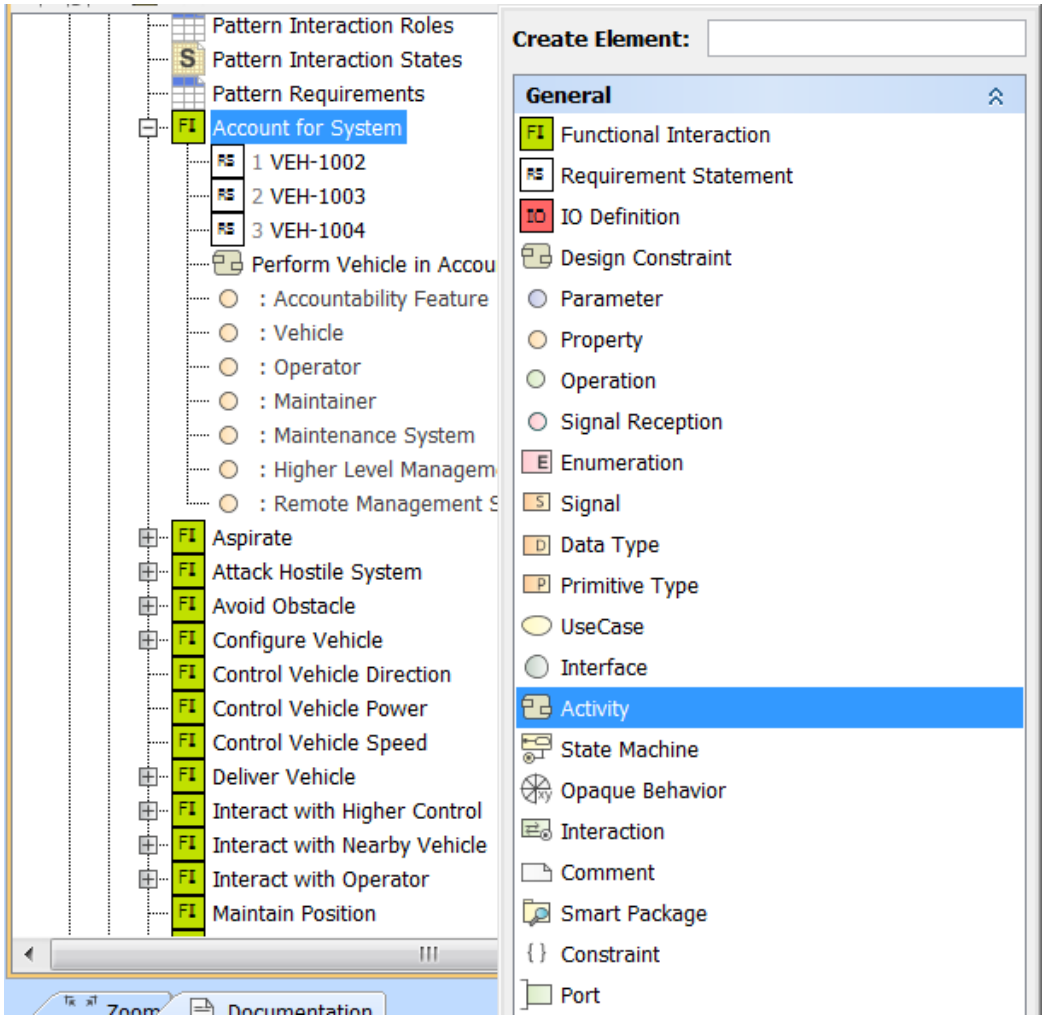
- For a modeled Functional Interaction, and two of its modeled Functional Roles, populate at least one Requirement Statement for each Role.
- Include populating the connection between the Interaction, Role, and Requirement, using the “Requirement Transfer Function” concept.

- Requirement Statements and the Requirement Transfer Functions they specify are created under their respective Functional Interaction.
- There should be at least one Requirement Transfer Function for each Interaction-Role pair.



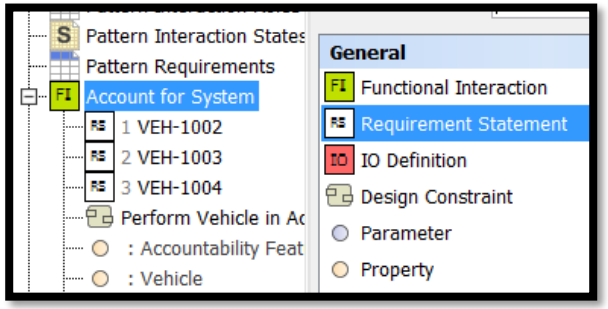
Detail Requirements Modeling: Creating Requirement Transfer Functions

- Use ~Create Element and choose Activity
- Add the stereotype “Requirement Transfer Function” to the new Activity
- Create an Allocate dependency from the new Requirement Transfer Function to the Logical Systemthat will perform it.



Detail Requirements: Creating Requirement Statements

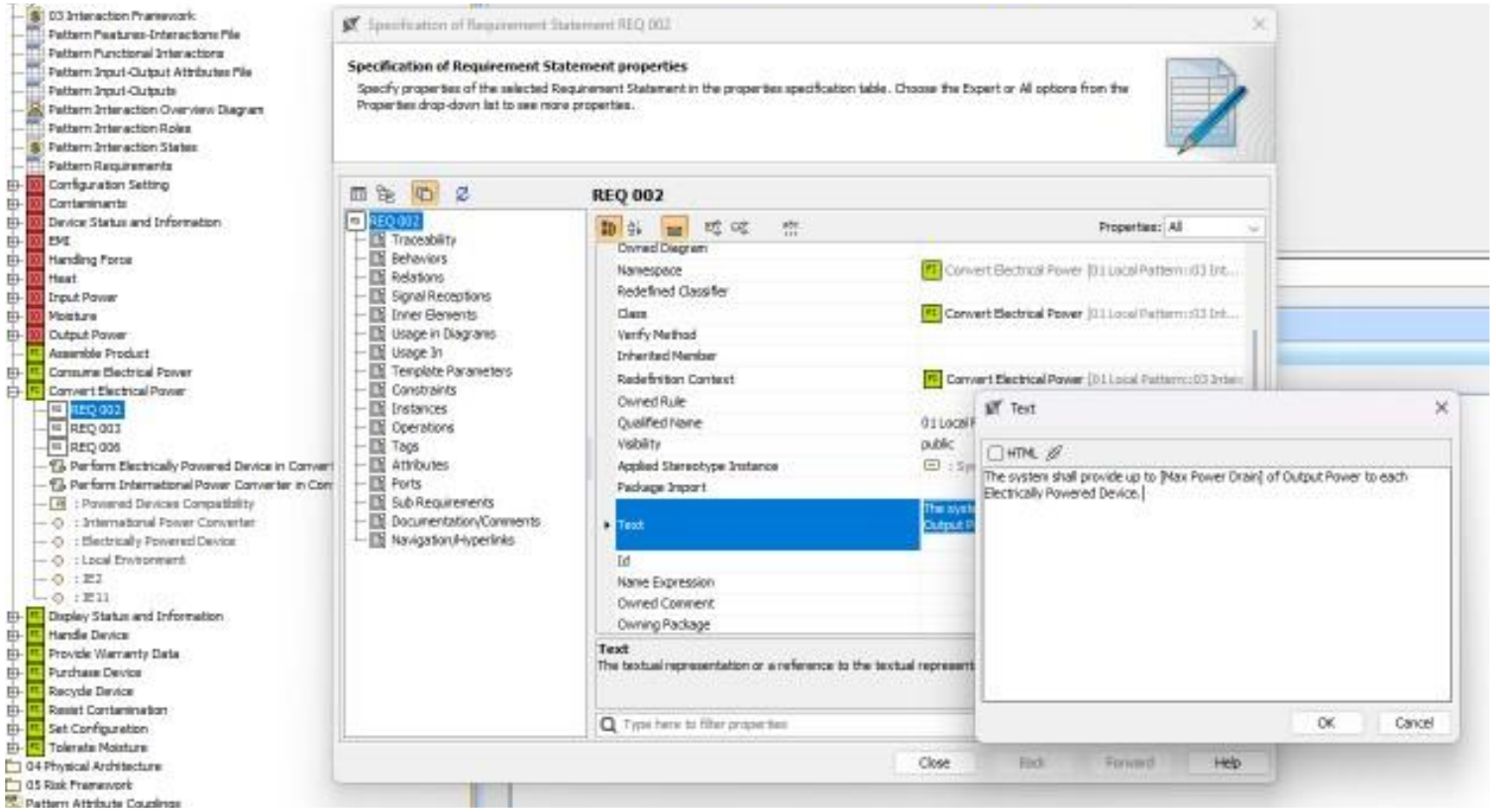
- Use ~Create Element and choose Requirement Statement
- Name the new Requirement Statement with its Requirement ID.
- Create a Satisfy dependency to the new Requirement Statement from the Requirement Transfer Function it specifies.
- At this point, the Requirement row will appear in the Pattern Requirements Table. The Requirement Statement can be entered in the Text column.
- Alternatively, open the Specification for the new Requirement Statement and enter the statement details in the Text property field (next slide).



#	Functional Interaction	IPK Value	Functional Role	RPK Value	Req ID	RSPK Rule	Text
1	Consume Electrical Power		International Power Converter		REQ 001		The system shall consume not more than [Max Drain on Mains] of Input Power from the Local Power Distribution System.
2	Convert Electrical Power	*ANY*	International Power Converter	*ANY*	REQ 002	IPK	The system shall provide up to [Max Power Drain] of Output Power to each Electrically Powered Device.
3	Convert Electrical Power	*ANY*	Electrically Powered Device	*ANY*	REQ 003	IPK	The system shall not consume more than [Max Power Drain] of Output Power.
4	Consume Electrical Power		Local Power Distribution System		REQ 004		The system shall provide up to [Max Drain on Mains] of Input Power.
5	Handle Device		International Power Converter		REQ 005		The system shall not present a shock hazard to users when operated according to its instructions.
6	Convert Electrical Power	*ANY*	International Power Converter	*ANY*	REQ 006	IPK	The system shall generate Output Power to attached Electrically Powered Devices which is compatible with the [Output Voltage-Power Profile] and [Output Frequency Profile].

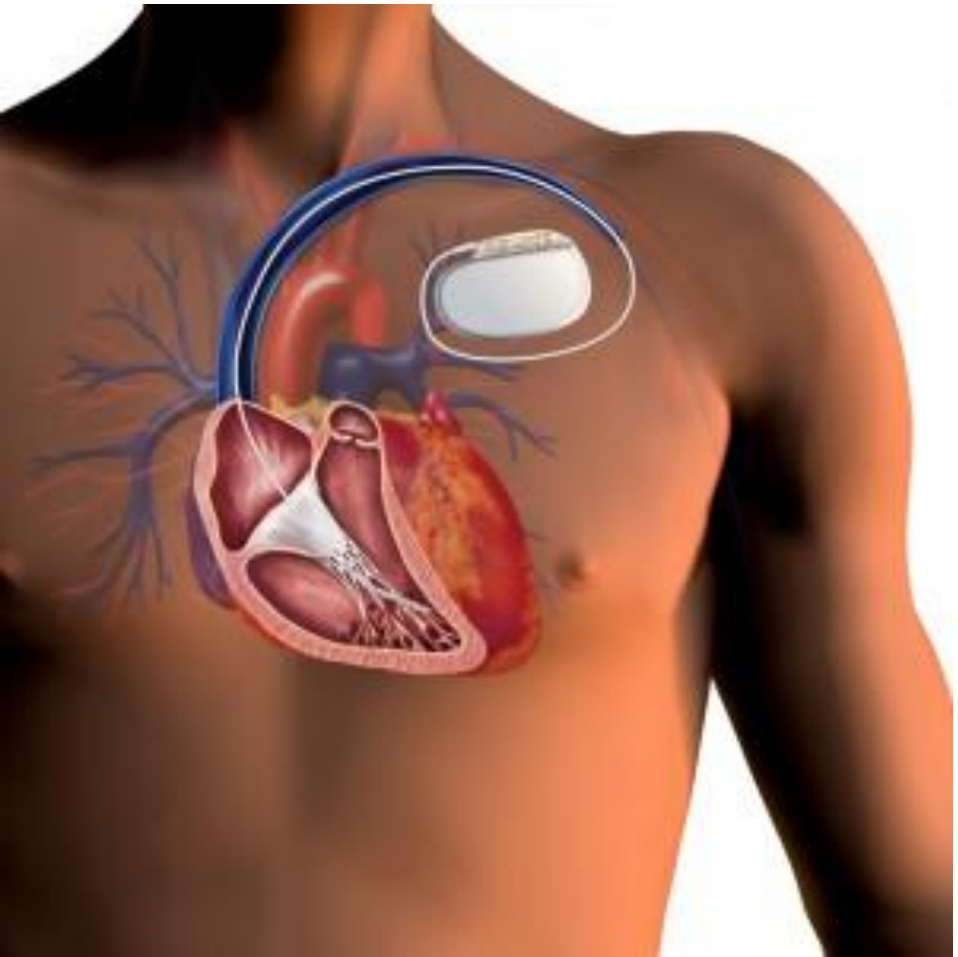
Detail Requirements:
Creating Requirement Statements

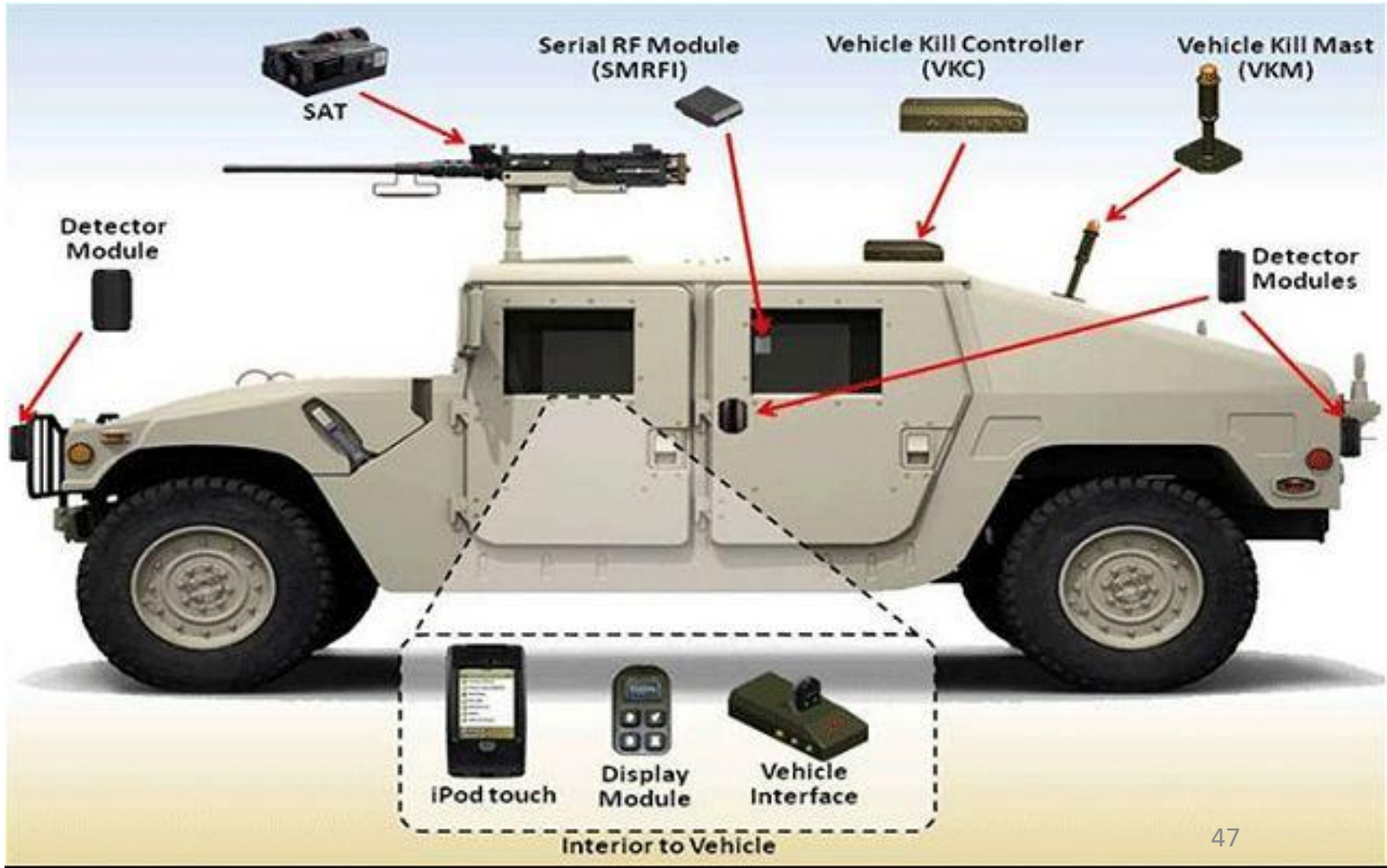
- Alternatively, open the Specification for the new Requirement Statement and enter the statement details in the Text property field.



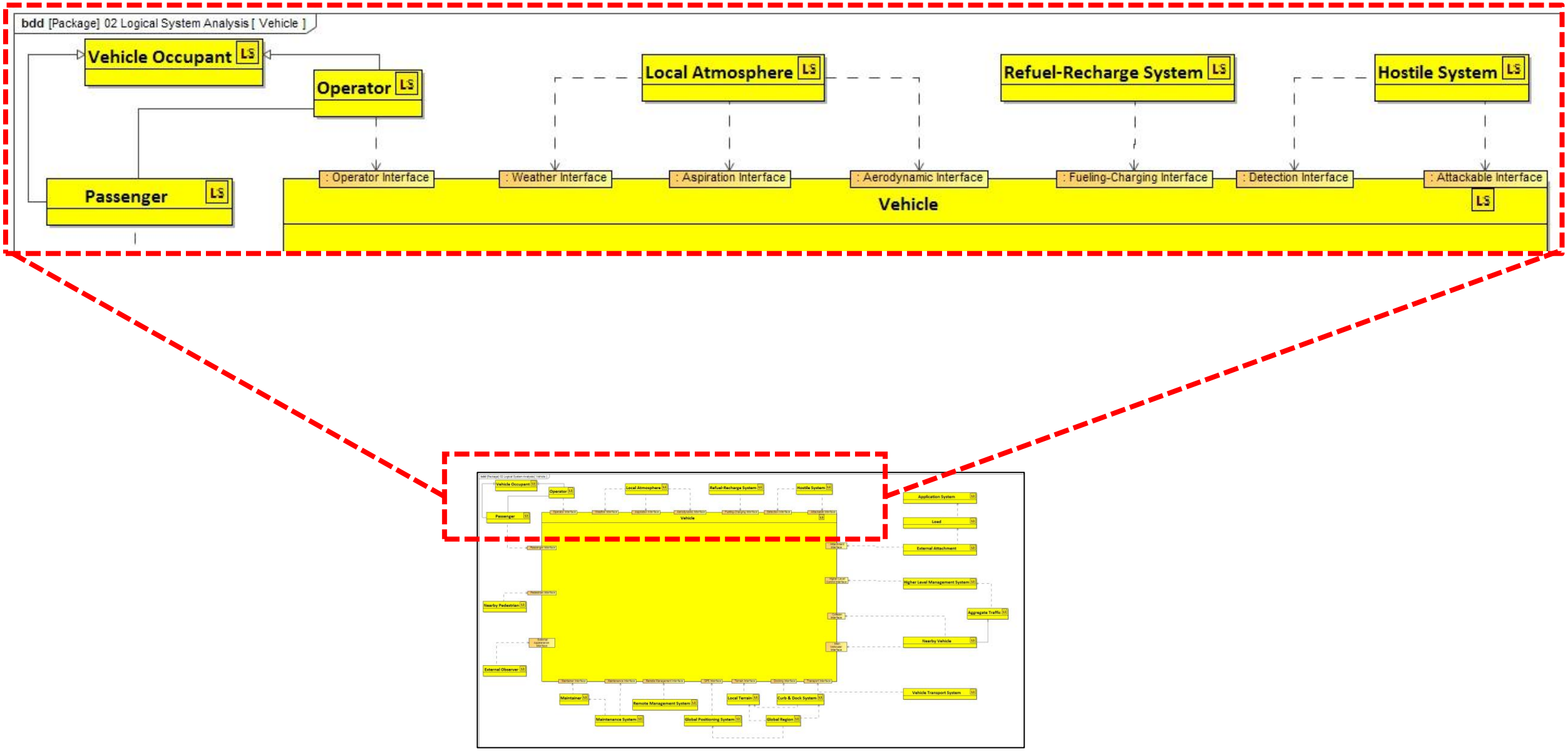
Interfaces, Systems of Access, Interface Context Model

Interfaces come in many types—what can describe them all?





Example: Vehicle External Interfaces, visible in Domain Model



Semantics of S*Interface

The Setting: Consider two interacting systems, exchanging at least one Input-Output (e.g., a Force, Energy Flow, Material Flow, or Information), during Interaction D:

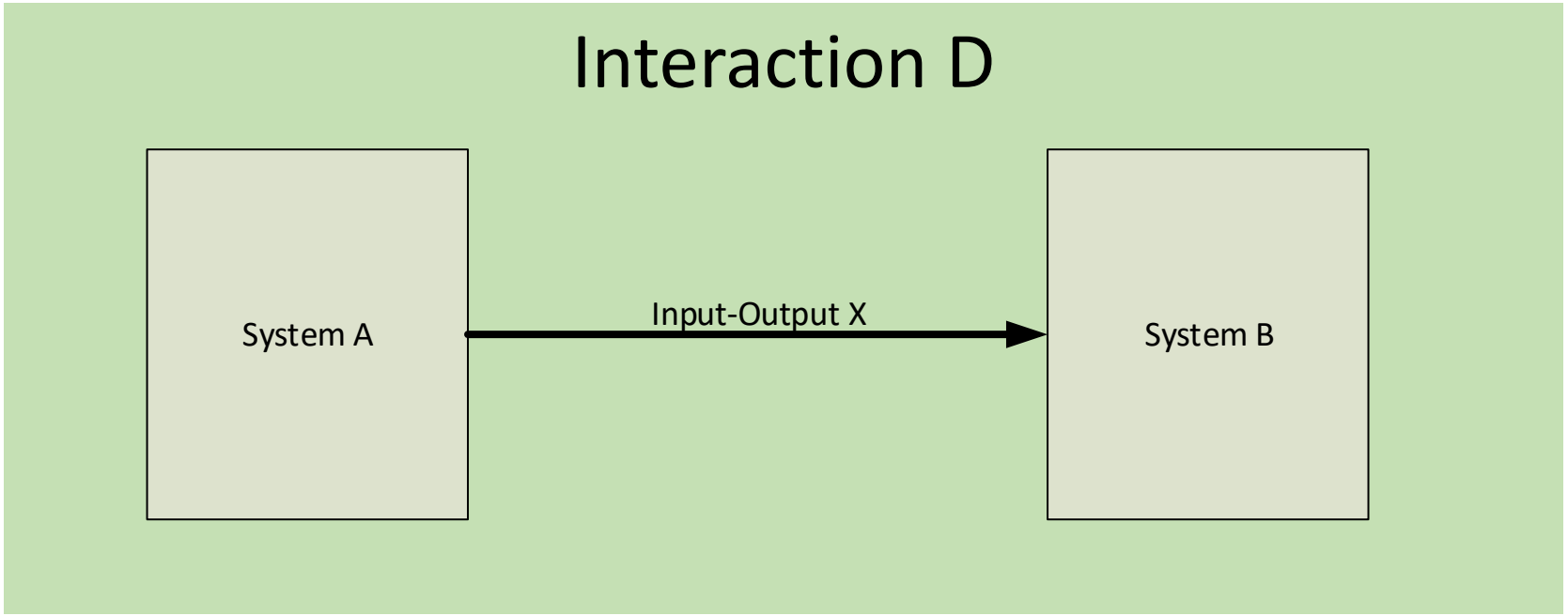


Figure 1: (Exact notation used not important to this discussion)

- In certain (important to identify) circumstances, we need to represent Interfaces involved in Interaction D.
- No matter what (graphical or other) modeling language or notation is used, the S*Metamodel tells us that an Interface is a reified association of:
  - A System, which “has” or “presents” the Interface;
  - A (set of) Input-Output(s), which “pass through” the Interface;
  - A (set of) Interaction(s), which describe “behavior at the Interface”;
  - A System of Access (SOA), providing the interaction “medium”:

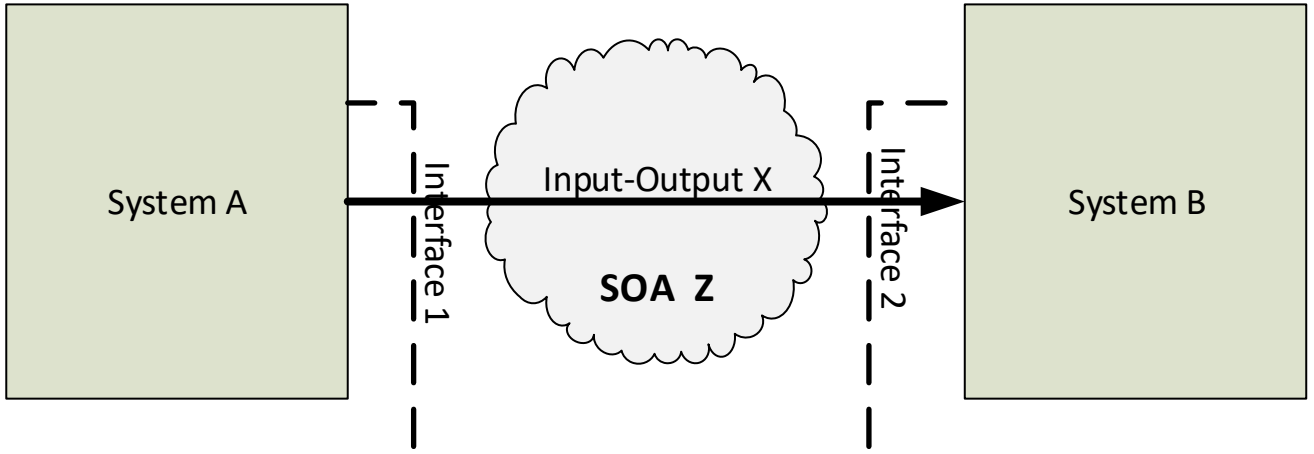


Figure 2: (Exact notation used not important to this discussion)

Example SOAs:
- Electrical Cable
- Ethernet LAN
- The Internet
- Hydraulic Plumbing
- Mechanical Coupling
- Animate Nervous System
- Airspace for Speaking
- Atmospheric Line of Sight
- Email
- Word of Mouth
- Sensors and Actuators

- However, there is a subtle inconsistency in the transition between Figure 1 and Figure 2 above:
  - Figure 1 and Figure 2 imply that the scope of “System A” must have changed between the two diagrams, . . .
  - Because, System A in Figure 2 can interact with an external-looking SOA Z, but . . . .
  - System A in Figure 1 implies that the scope of System A is such that it can interact directly with System B.

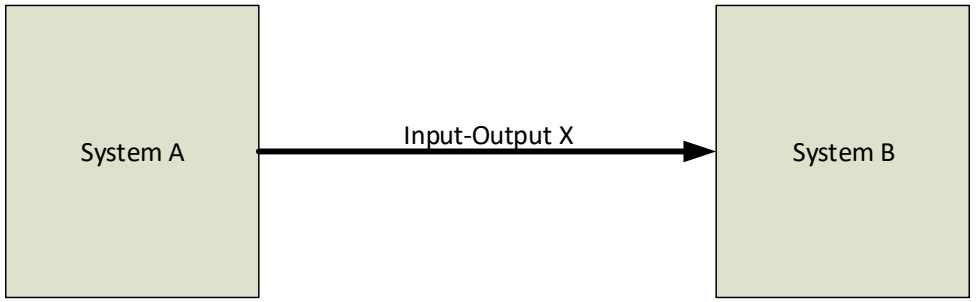


Figure 1

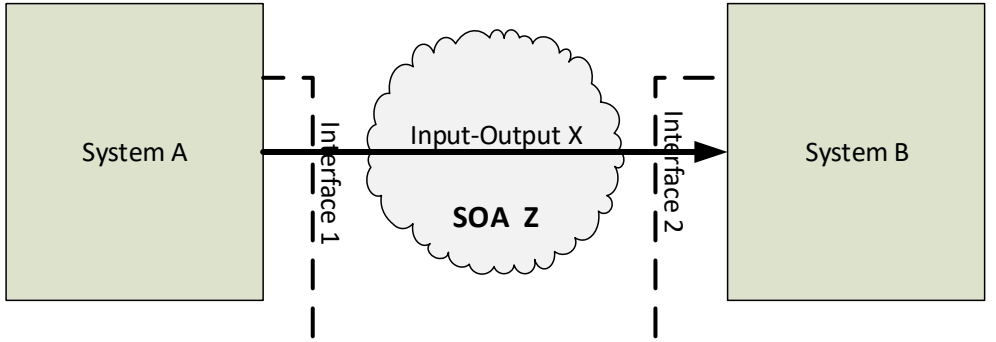


Figure 2

- The problem here is that even intended “neutral” notations can be specific enough to mislead us or create ambiguities.
- The real problem is that, independent of notation, the System of Access by definition has larger scope than Figure 2 implied:

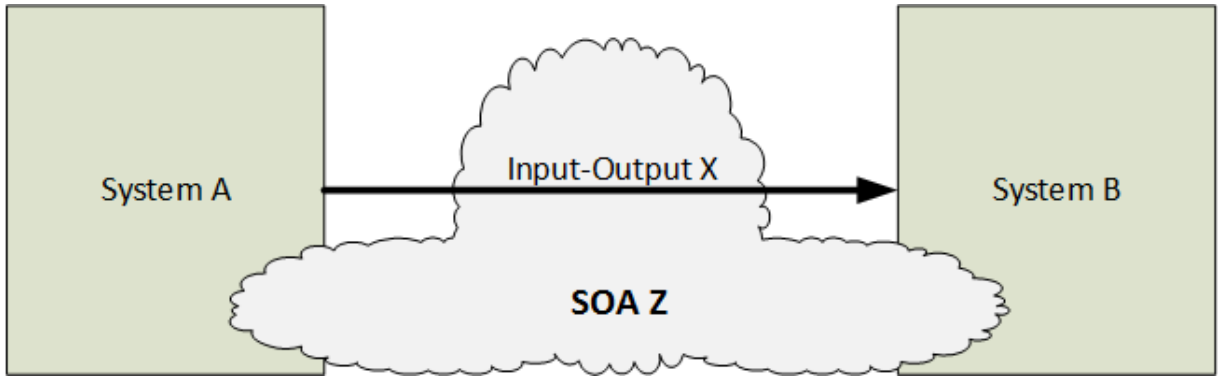


Figure 3: (Exact notation used not important to this discussion)

- Part of the scope of the System of Access for two interacting systems must necessarily be within the two interacting systems . . .

- So, to avoid conflicting or ambiguous definitions of the scope of System A, we have to recognize a slightly larger system, shown in Figure 4 as System A'
- The additional scope adds the SOA role shown here as SA SOA:

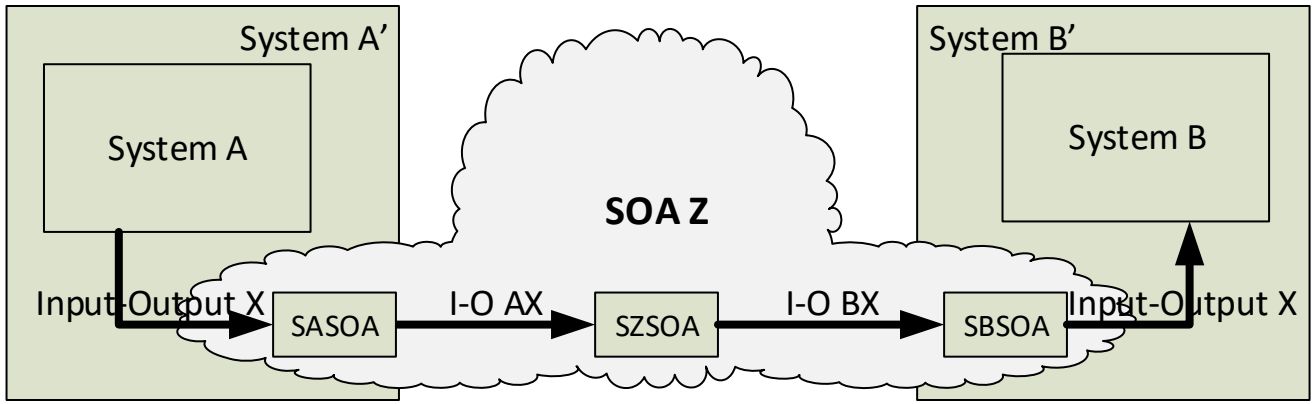
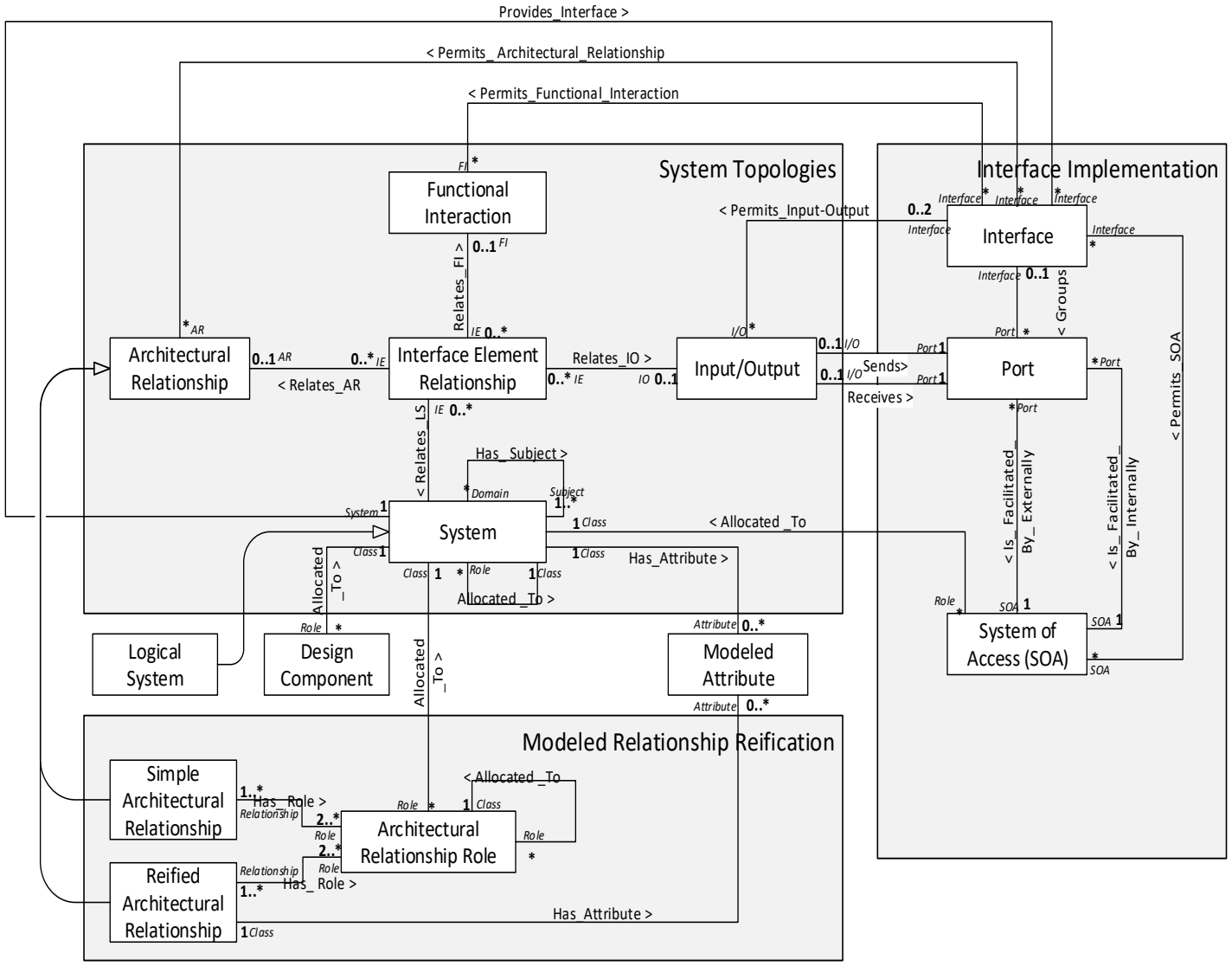


Figure 4: (Exact notation used not important to this discussion)

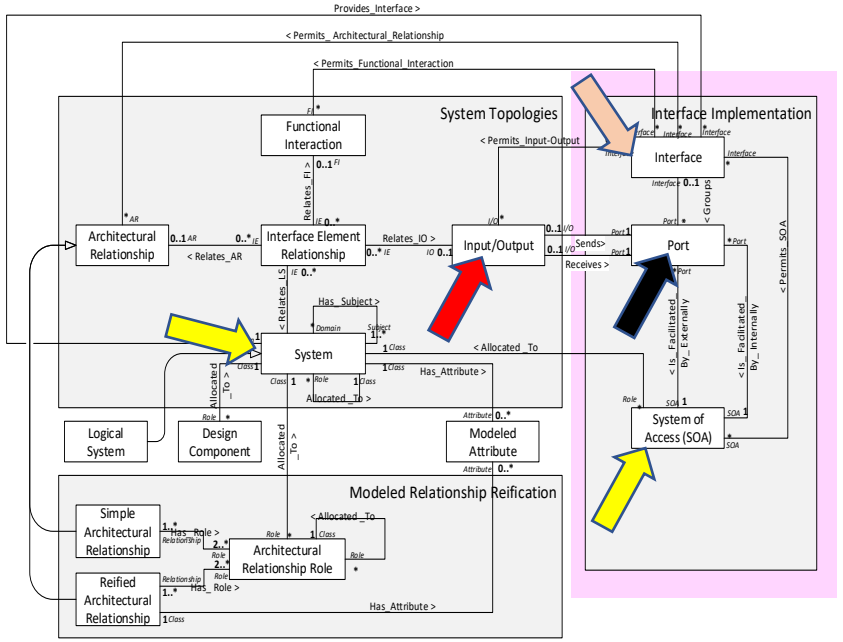
Example Internal SOA Elements:

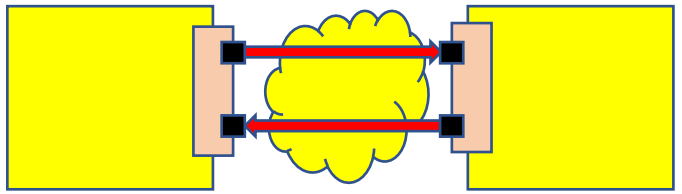
- Electrical Line Drivers
- Software Drivers
- APIs
- Hydraulic Coupling Connectors
- Mechanical Mounting Brackets
- Fixtures
- Electrical Connectors
- Voice box
- Animate Pain Nerve
- Animate Musculature
- Keyboard
- Mouse
- Display Screen
- Alarm Buzzer
- Speaker
- Microphone



The overall Interface Context Pattern is administered in the form of four Interface Context (ICT) pattern segment tables, related to the three Interface Context segments shown here, to minimize Pattern size and complexity.

Interface Context Table 1 (ICT1) (mostly about the Interface Implementation Segment)





The ICT1 table's own primary key (row generator element) is the Port Name column.

The five classes that are the backbone of the ICT1 segment

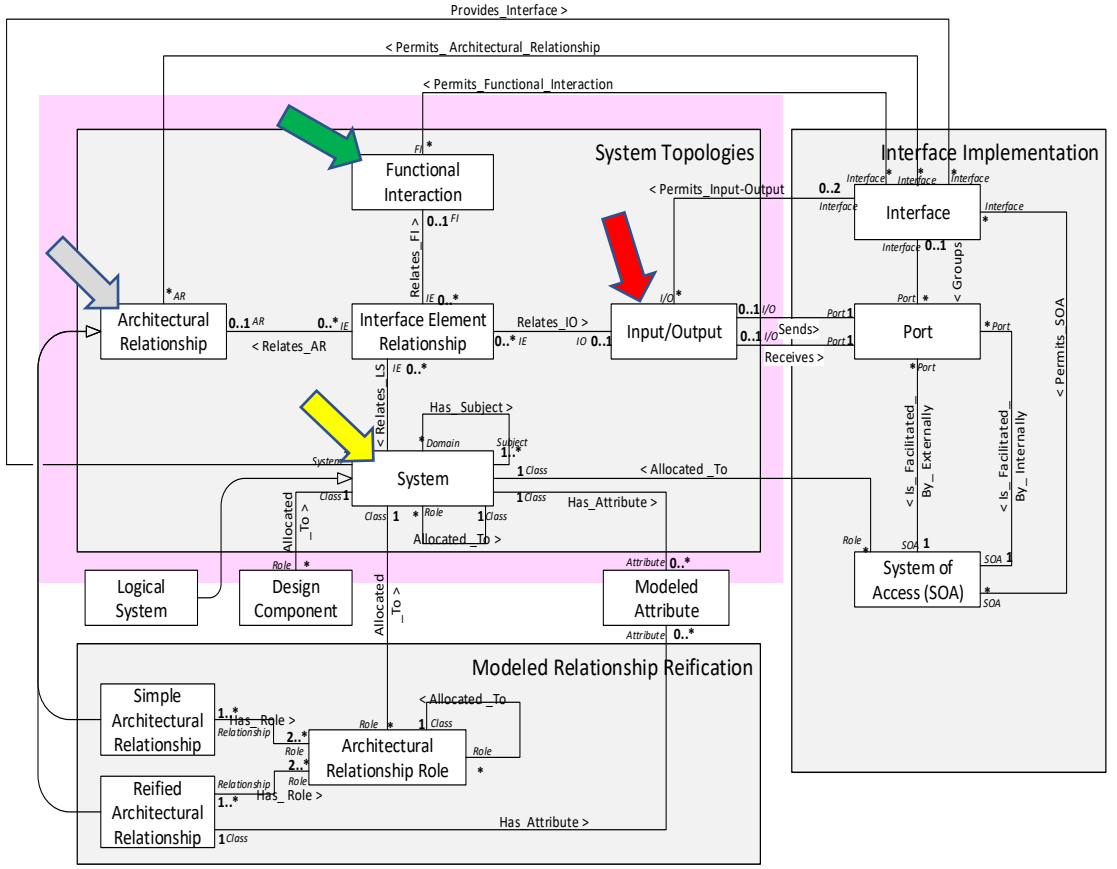
System Name	Interface Name	Interface PK Rule	Input-Output Name	IO PK Rule	IO Direction	Port Name	Port PK Rule	SOA Name	SOA PK Rule	SOA Internal or External
Vehicle	Aspiration Interface		Intake Air		In	VH.1		Intake Manifold		External
Vehicle	Aspiration Interface		Exhaust Air		Out	VH.2		Exhaust Pipe		External

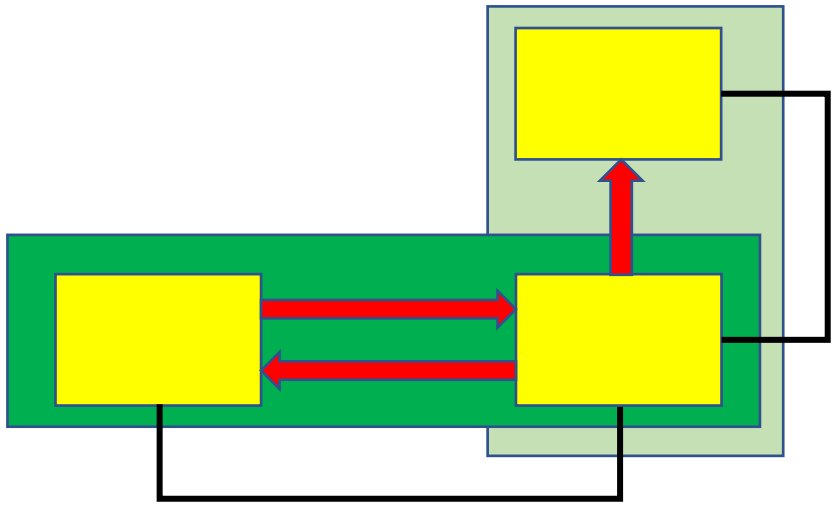
Interface Context Table 2 (ICT2) (mostly about the System Topologies Segment)

The four classes that are the backbone of the ICT2 segment

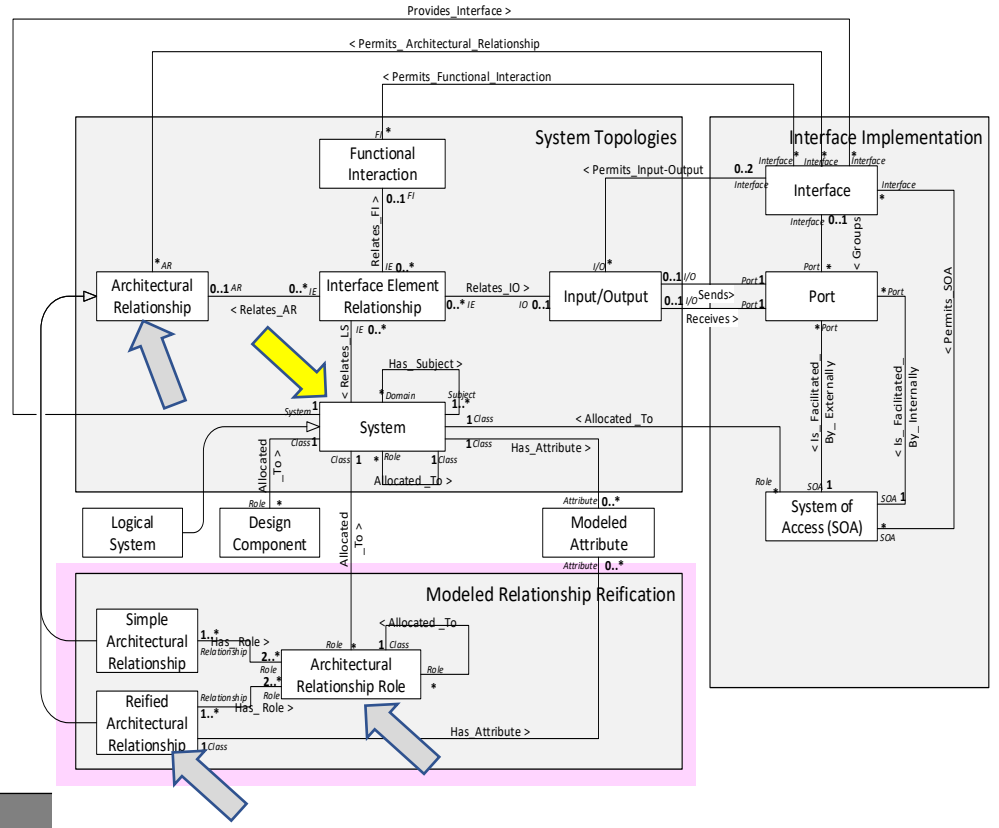
Interface Element	Interaction Name	System Name	IO Name	Architectural Relationship
IE1	Aspirate	Vehicle System	Intake Air	Provides Oxidant to
IE2	Aspirate	Vehicle System	Exhaust Air	Removes Exhaust from

The ICT2 table's own primary key (row generator element) is the Interface Element column, which is the unique identifier of each 4-tuple shown.



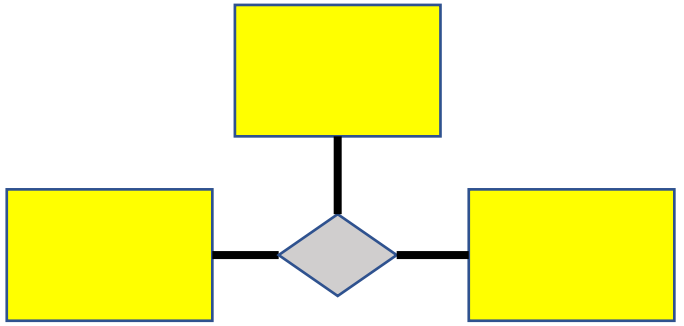


Interface Context Table 4 (ICT4) (mostly about the Modeled Relationships segment, especially Reified Relationships)



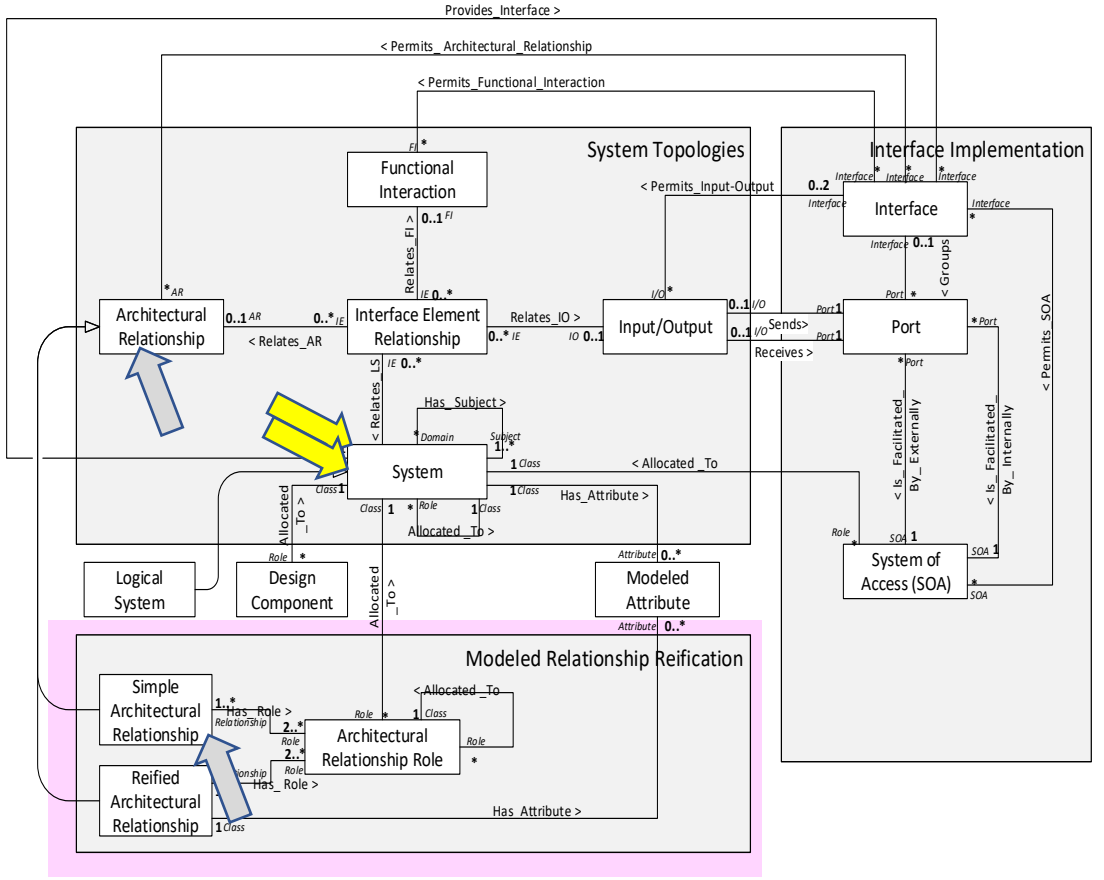
The three classes that are the backbone of the ICT4 segment

System Name	Architectural Relationship Name	AR PK Rule	AR Role Name	AR Role PK Rule	AR Internal or External	AR Complexity



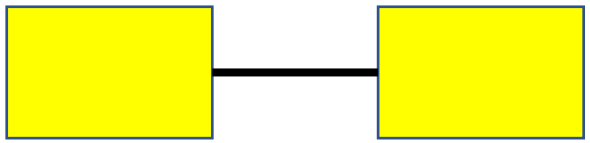
The ICT4 table's own primary key (row generator) is the Architectural Relationship-AR Role pair, for Architectural Relationships that are Reified.

Interface Context Table 5 (ICT5) (mostly about the Modeled Relationships segment, for Simple Relationships)



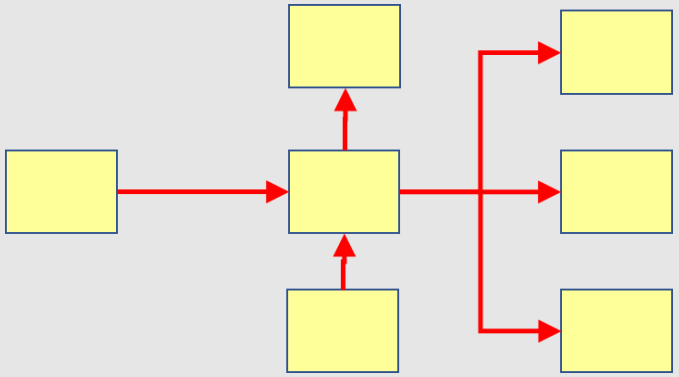
The three classes that are the backbone of the ICT5 segment

Architectural Relationship Name	AR PK Rule	Connected System, Subject End of AR	Connected System, Object End of AR	Internal or External AR	AR Complexity
Provides Oxidant to		Local Atmosphere	Vehicle System	External	Simple

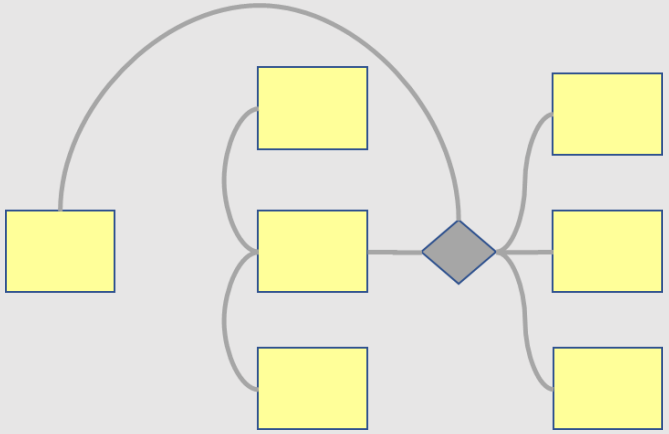


The ICT5 table's own primary key (row generator element) is the Architectural Relationship Name, for Architectural Relationships that are Simple.

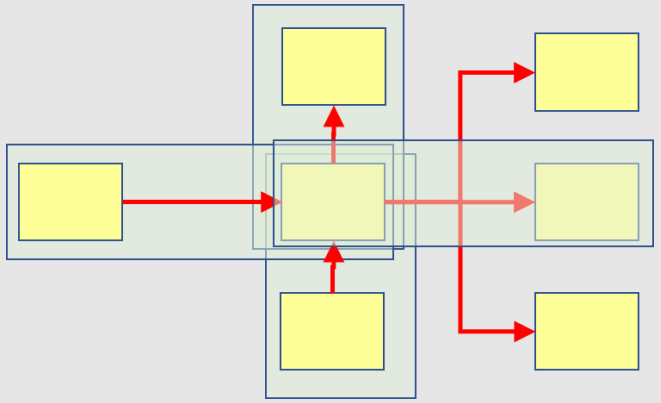
Behavioral Topologies (Behavioral Architectures)



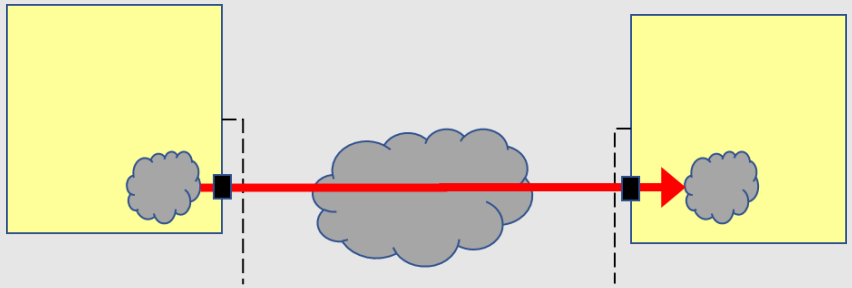
Input-Outputs Topology



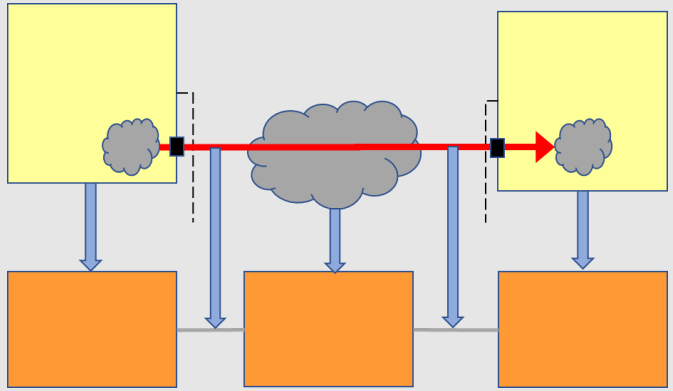
Architectural Relationships Topology (Logical Architecture)



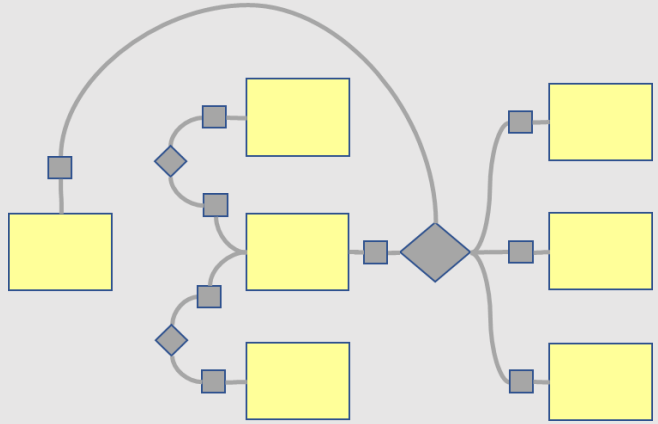
Interaction Topology



Interface Implementation



Allocation Topology, Physical Architecture



Reification of Relationships

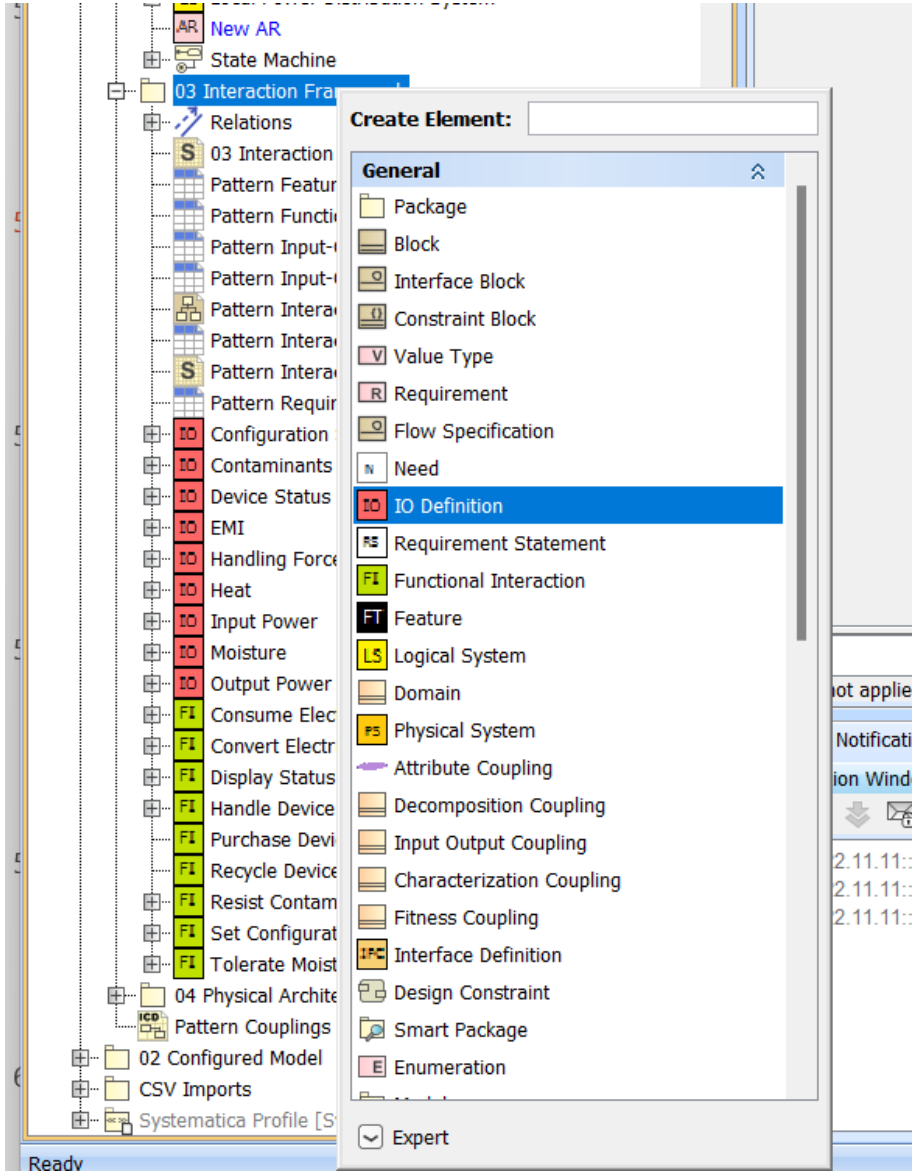
Exercise: Populating Interface Context Model

- For a modeled “black box” Functional Interaction of system of interest with an external Actor:
  - Populate a modeled Input-Output exchanged between them during the Interaction.
  - Populate a modeled external Interface for the system of interest, through which the Input-Output passes.
  - Populate a related System of Access (SOA).
  - Populate Ports for the two systems.

Detail directions . . .

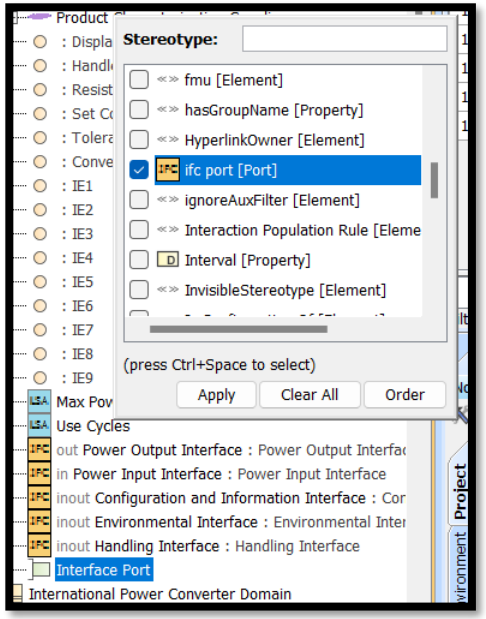
Exercise: Interface Context Modeling: Populating Input-Outputs

- Input-Outputs are created in package 01 Local Pattern::03 Interaction Framework.
- Use ~Create Element and choose IO Definition



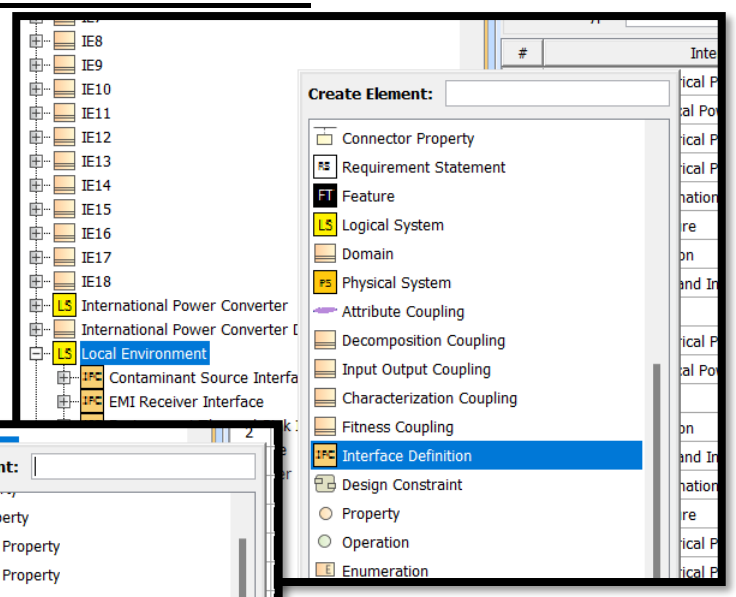
Exercise: Interface Context Modeling: Populating Interfaces

- Interface Definitions are created in package 01 Local Pattern::02 Logical System Analysis under the Logical System that owns them.
  1. Use ~Create Element and choose Interface Definition; give the new Interface a name.
  2. Next, use ~Create Element and choose Proxy Port; give it the Proxy Port the same name as the Interface Definition created in the above step.
  3. Select the Proxy Port in the browser, select the Stereotype option, then add ifc port stereotype.

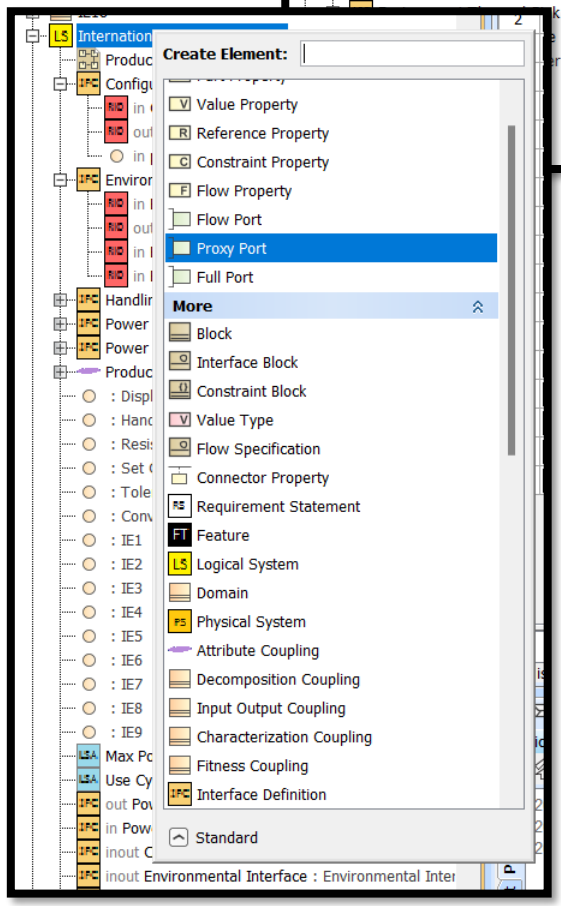


3

1

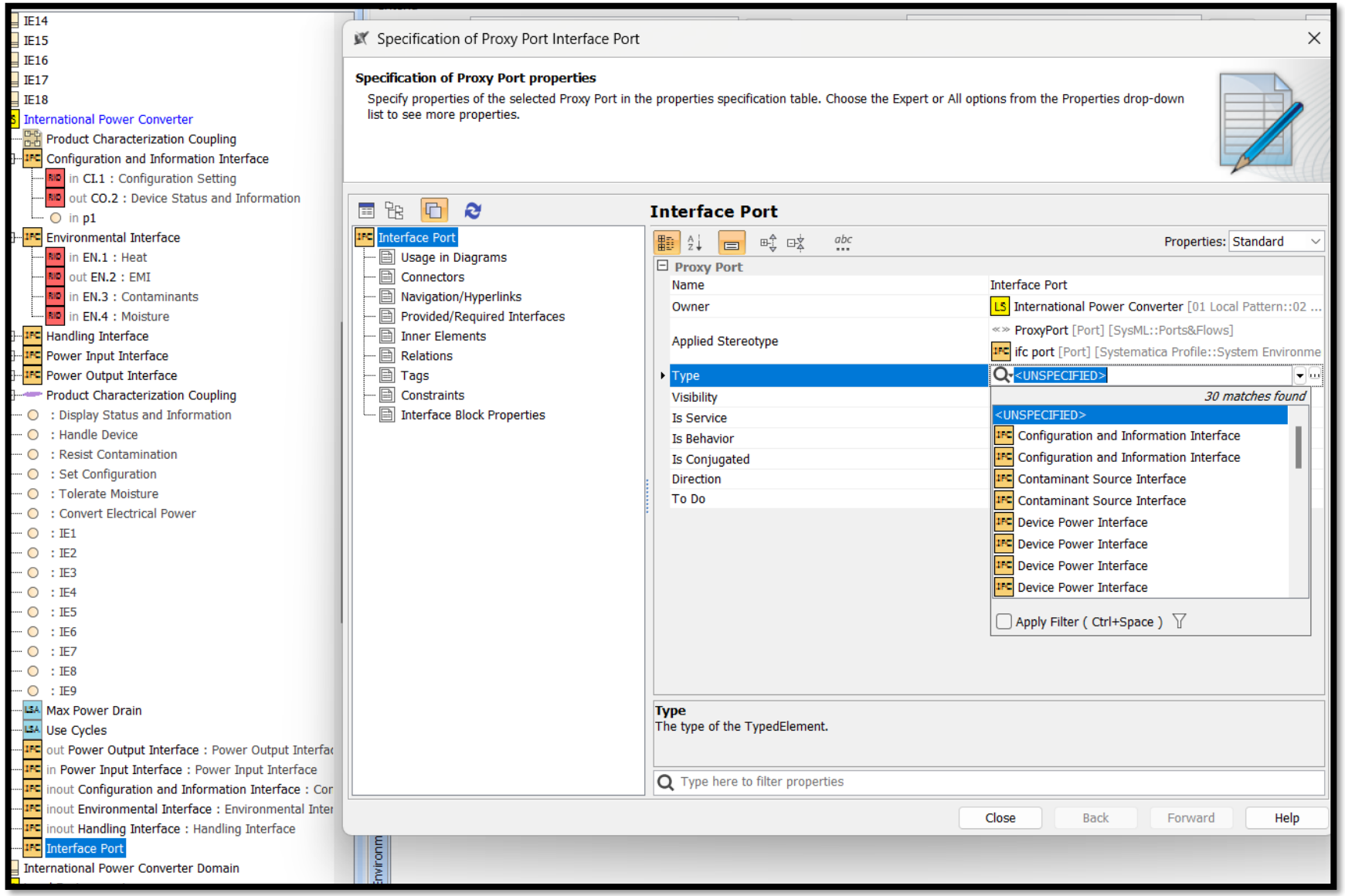


2



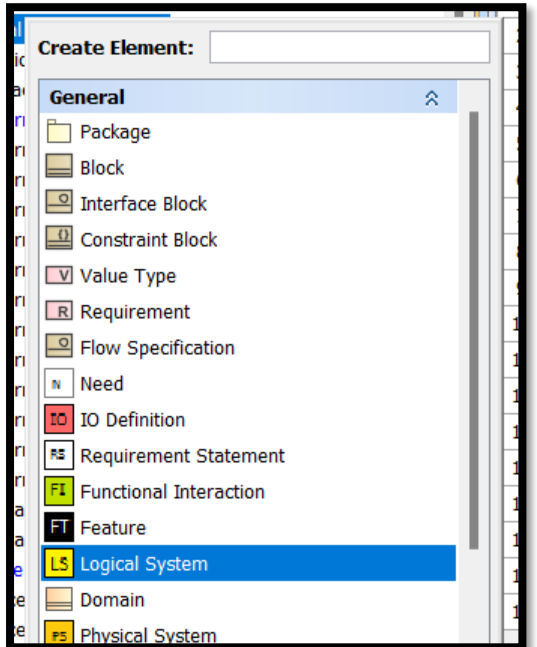
Exercise: Interface Context Modeling: Populating Interfaces, continued

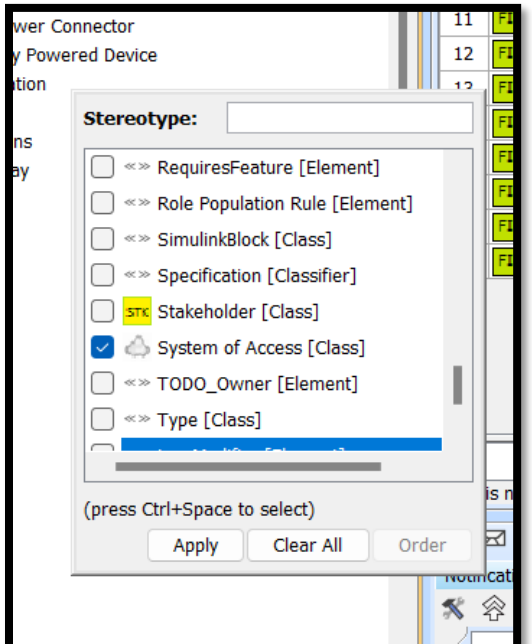
- Select the new ifc port in the browser, open its specification, select the Type property and choose the associated Interface Definition with the same name.



Exercise: Interface Context Modeling: Populating Systems of Access (SOAs)

- Systems of Access are created in package 01 Local Pattern::02 Logical System Analysis
- Use ~Create Element and choose Logical System (because System of Access is a type of Logical System).
- Right-click on the newly created Logical System, select Stereotype and change the Stereotype to System of Access and remove the Logical System Stereotype.



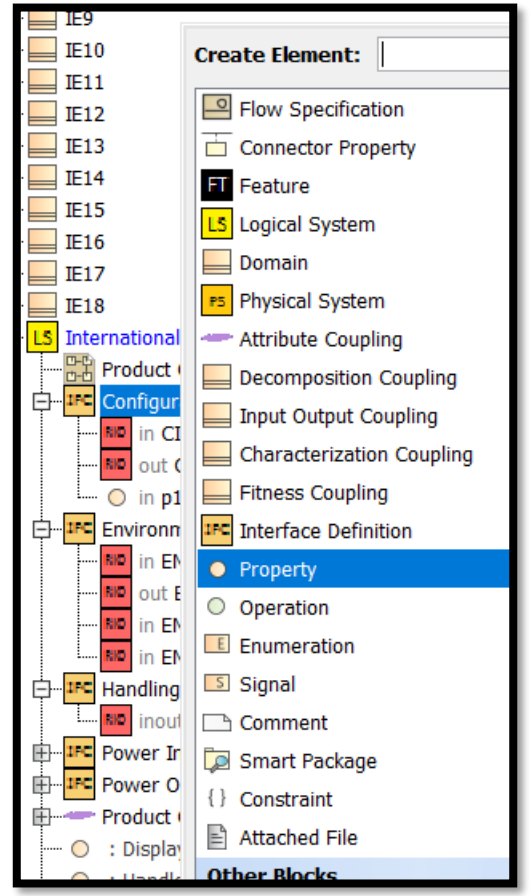


Exercise: Interface Context Modeling: Populating Ports

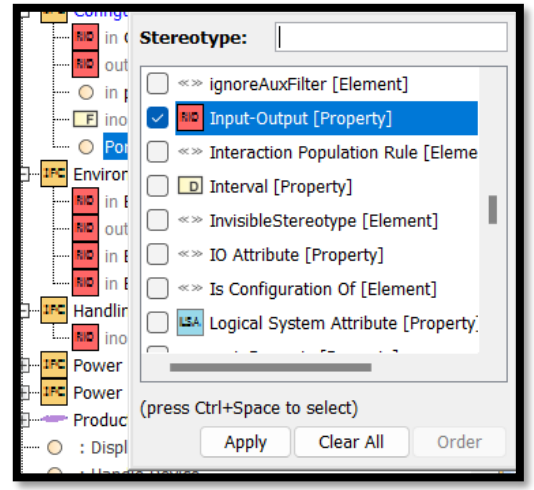
- Ports are created in package 01 Local Pattern::02 Logical System Analysis under the Interface Definition that owns the Port:

1. Use ~Create Element and choose Property.
2. Select and right-click the property in the containment browser, choose Stereotype, choose Input-Output [Property].
3. Select the Port/IO Property in the browser, open its specification, select the Type property and choose the associated IO Definition.

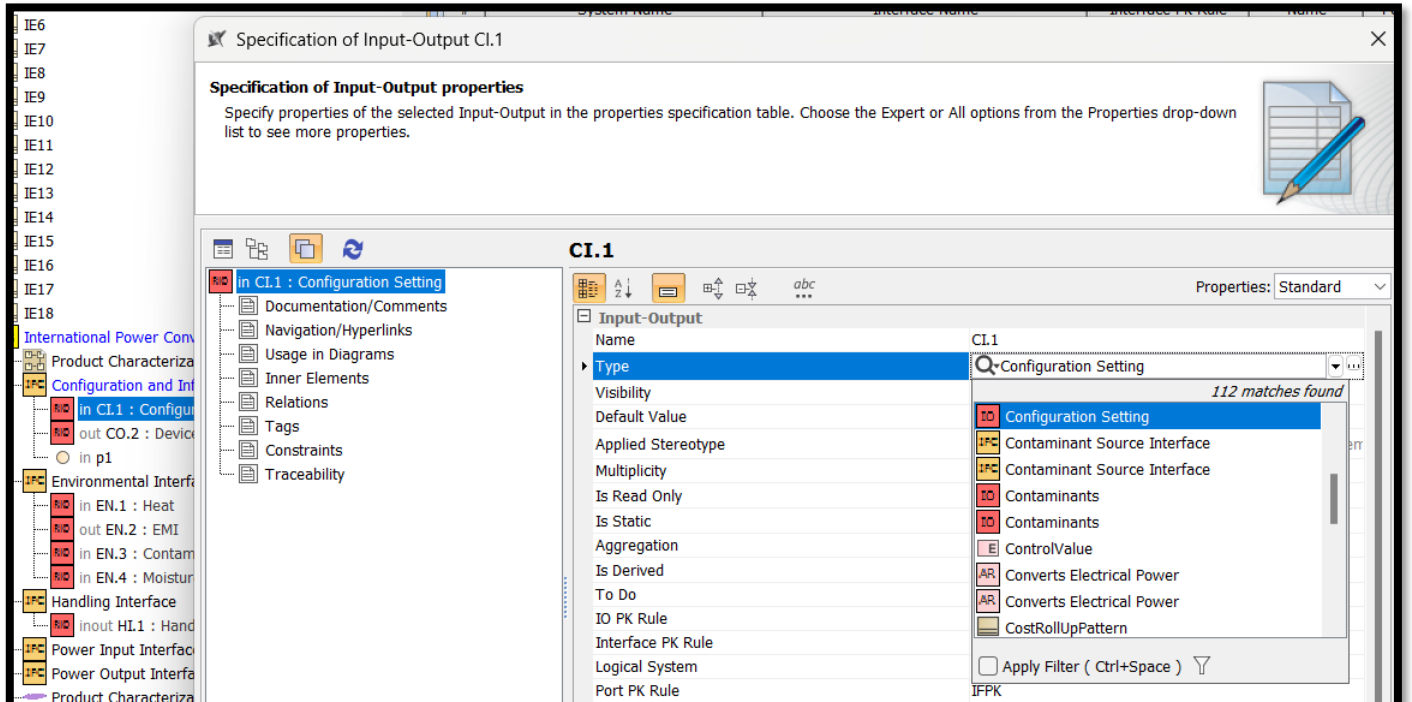
1



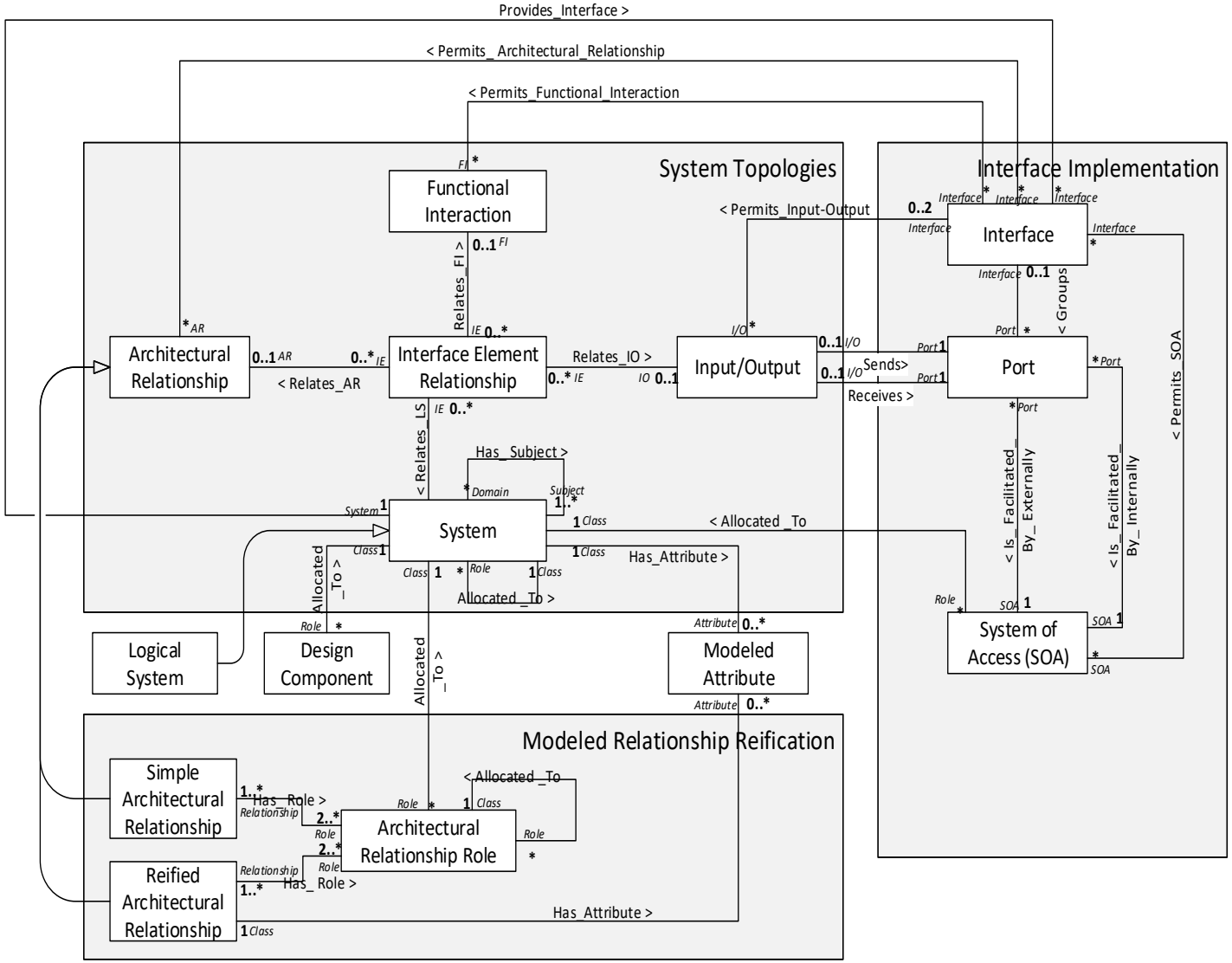
2



3

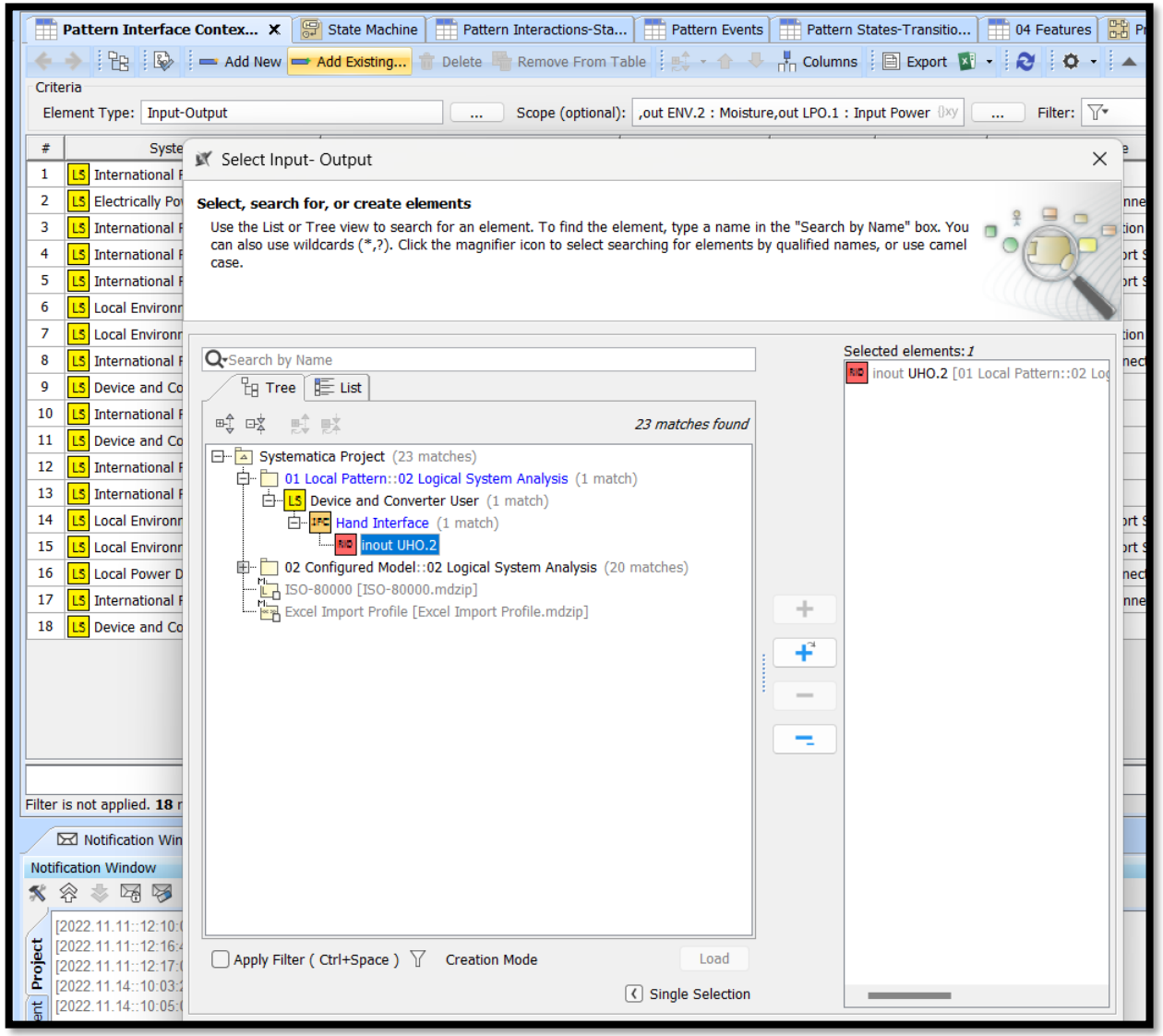


Exercise: Interface Context Modeling: Connecting the Elements You Populated



Exercise: Interface Context Modeling: Connecting the Elements: Table ICT1

- Pattern Table ICT1 shows relationships to Ports and also administers the related configuration rules.
- To add a Port to the table, select Add Existing and select the Port for its browser location.
- The System, Interface, and Input-Output column will have content automatically populated based on the steps on the previous slide.
- That is, the first three are <u>already</u> "connected" because your method of entry recognized ownership connections.

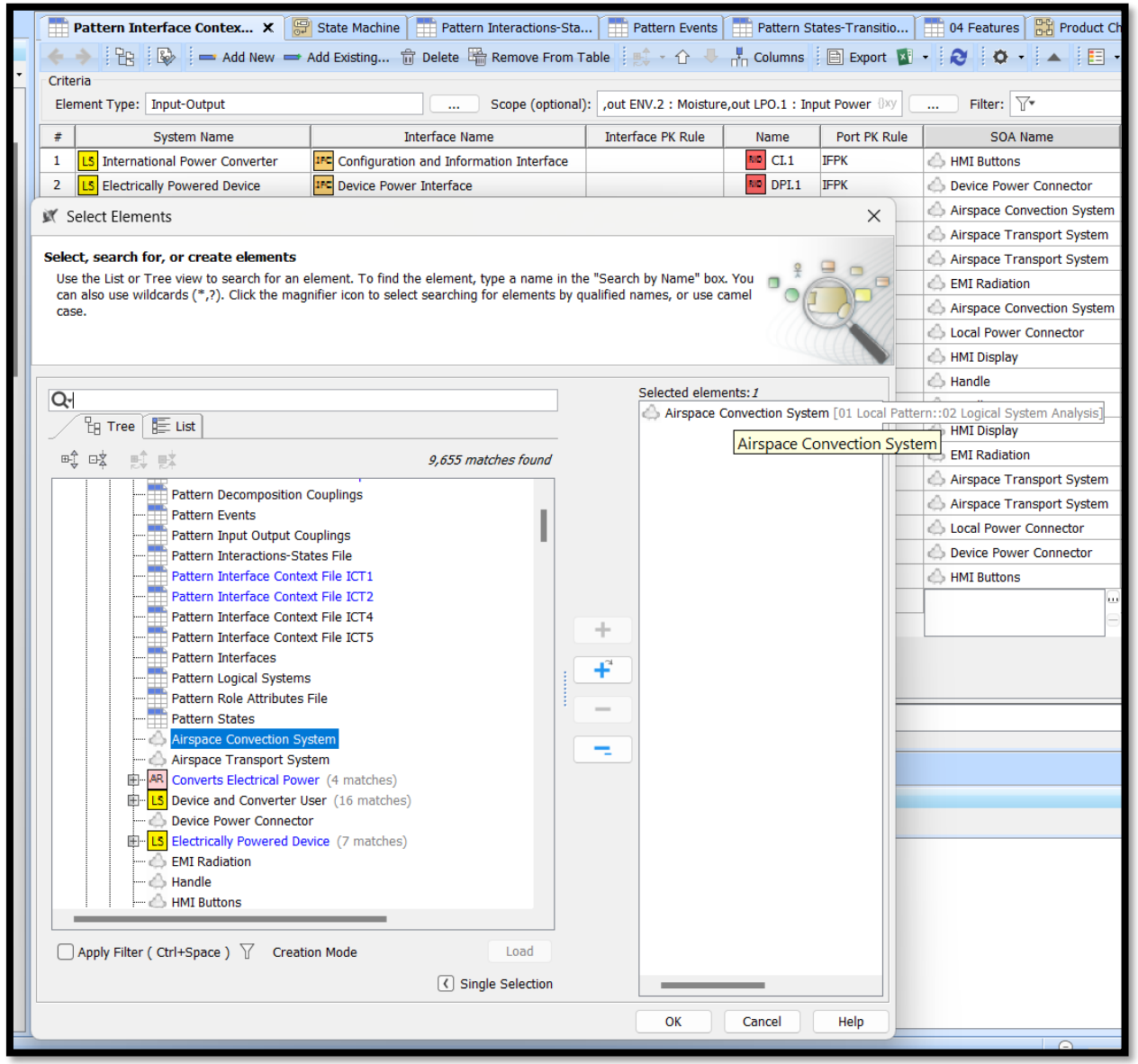


The five classes that are the backbone of the ICT1 segment

System Name	Interface Name	Interface PK Rule	Input-Output Name	IO PK Rule	IO Direction	Port Name	Port PK Rule	SOA Name	SOA PK Rule	SOA Internal or External
Vehicle	Aspiration Interface		Intake Air		In	VH.1		Intake Manifold		External
Vehicle	Aspiration Interface		Exhaust Air		Out	VH.2		Exhaust Pipe		External

Exercise: Interface Context Modeling: Connecting the Elements: Table ICT1

- To add the relationship to Systems of Access (SOAs), double-click the cell, then the ellipses to bring up the selection dialog window.
- Choose the related SOA from the containment browser.



The five classes that are the backbone of the ICT1 segment

System Name	Interface Name	Interface PK Rule	Input-Output Name	IO PK Rule	IO Direction	Port Name	Port PK Rule	SOA Name	SOA PK Rule	SOA Internal or External
Vehicle	Aspiration Interface		Intake Air		In	VH.1		Intake Manifold		External
Vehicle	Aspiration Interface		Exhaust Air		Out	VH.2		Exhaust Pipe		External

Exercise: Interface Context Modeling: Connecting the Elements: Table ICT1

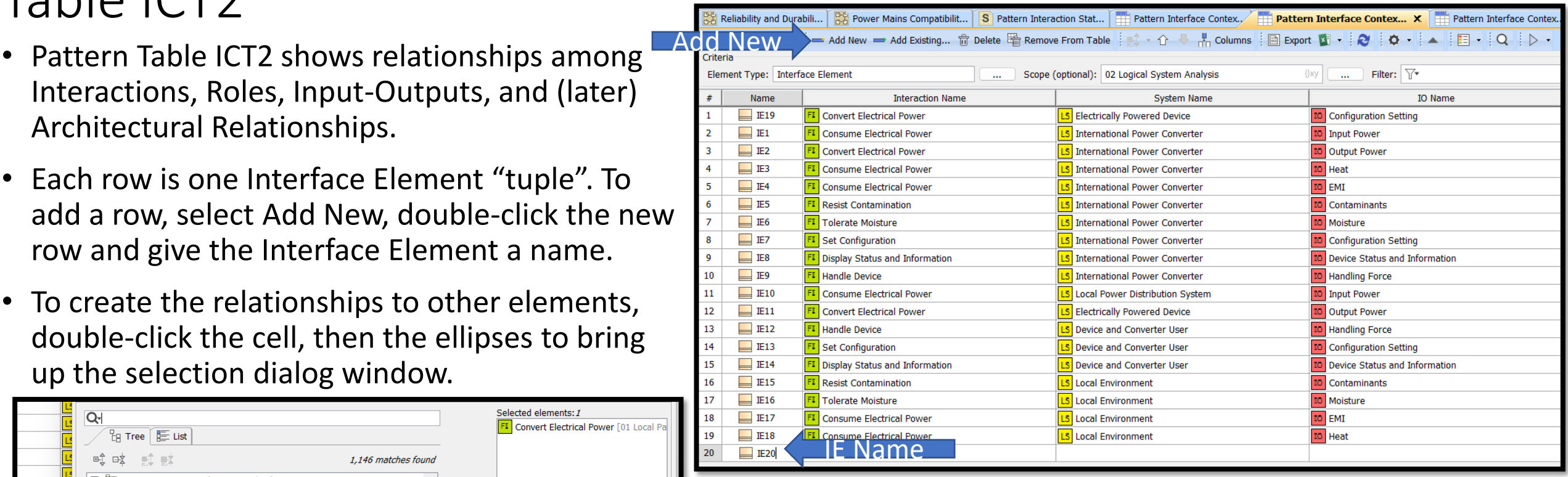
(Functional Role) (Interface) (Input-Output) (Port) (SOA)

#	System Name	Interface Name	Type	Name	SOA Name
1	International Power Converter	Configuration and Information Interface	Configuration Setting	CI.1	HMI Buttons
2	Electrically Powered Device	Device Power Interface	Output Power	DPI.1	Device Power Connector
3	International Power Converter	Environmental Interface	Heat	EN.1	Airspace Convection System
4	International Power Converter	Environmental Interface	Contaminants	EN.3	Airspace Transport System
5	International Power Converter	Environmental Interface	Moisture	EN.4	Airspace Transport System
6	Local Environment	EMI Receiver Interface	EMI	ENV.3	EMI Radiation
7	Local Environment	Environment Thermal Sink Interface	Heat	ENV.4	Airspace Convection System
8	International Power Converter	Power Input Interface	Input Power	PI.1	Local Power Connector
9	Device and Converter User	Vision Interface	Device Status and I...	UC1.1	HMI Display
10	International Power Converter	Handling Interface	Handling Force	HI.1	Handle
11	Device and Converter User	Hand Interface	Handling Force	UHO.1	Handle
12	International Power Converter	Configuration and Information Interface	Device Status and I...	CO.2	HMI Display
13	International Power Converter	Environmental Interface	EMI	EN.2	EMI Radiation
14	Local Environment	Contaminant Source Interface	Contaminants	ENV.1	Airspace Transport System
15	Local Environment	Moisture Source Interface	Moisture	ENV.2	Airspace Transport System
16	Local Power Distribution System	Power Mains Interface	Input Power	LPO.1	Local Power Connector
17	International Power Converter	Power Output Interface	Output Power	PO.1	Device Power Connector
18	Device and Converter User	Finger Interface	Configuration Setting	UCO.1	HMI Buttons
19	Device and Converter User	Hand Interface	Handling Force	UHO.2	Handle

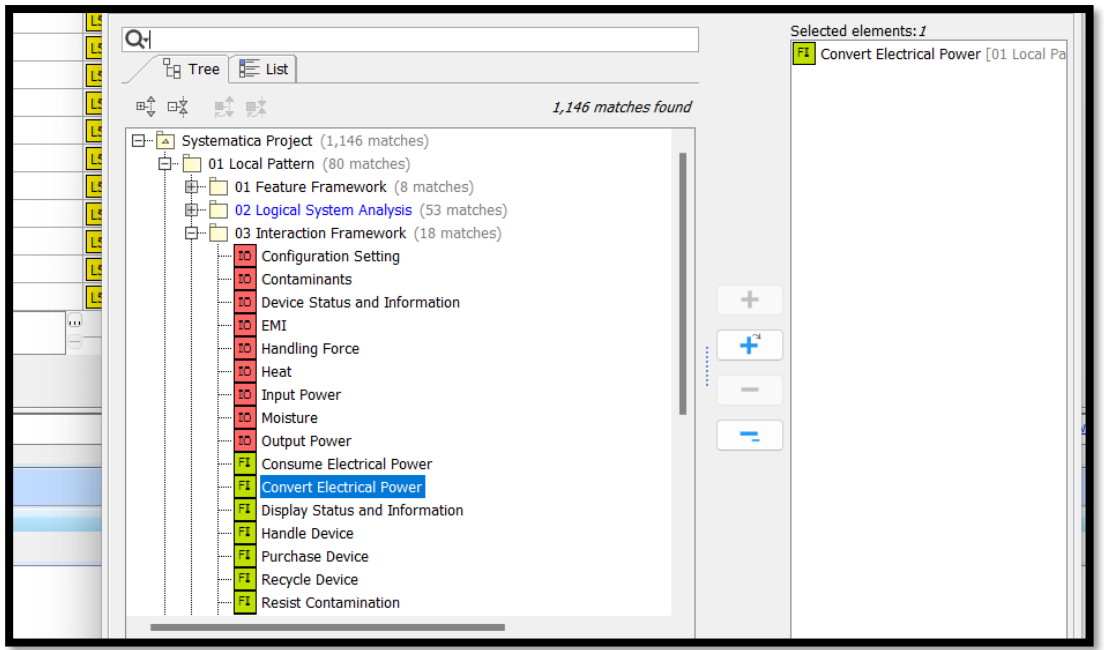
Example Pattern Interface Context File ICT1

Exercise: Interface Context Modeling: Connecting the Elements: Table ICT2

- Pattern Table ICT2 shows relationships among Interactions, Roles, Input-Outputs, and (later) Architectural Relationships.
- Each row is one Interface Element “tuple”. To add a row, select Add New, double-click the new row and give the Interface Element a name.
- To create the relationships to other elements, double-click the cell, then the ellipses to bring up the selection dialog window.

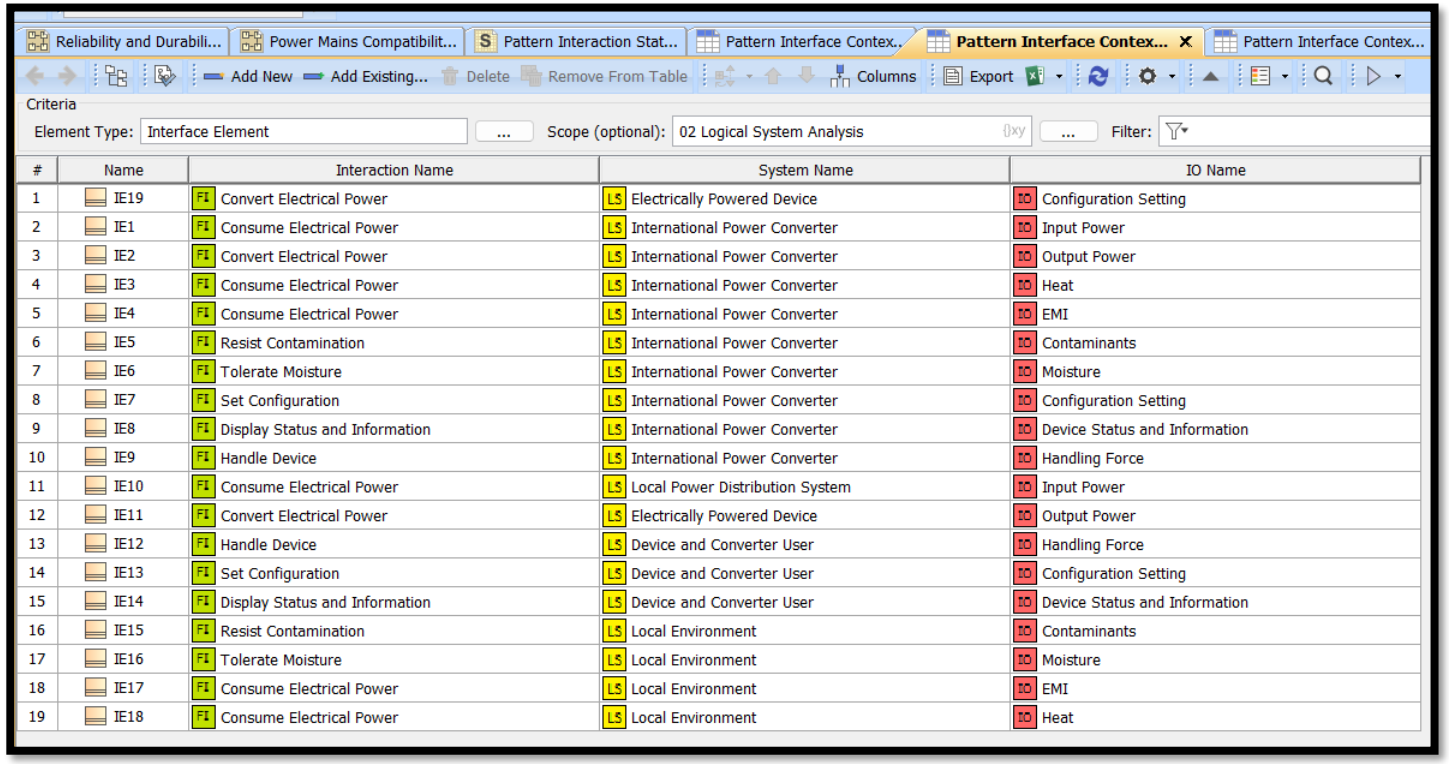


#	Name	Interaction Name	System Name	IO Name
1	IE19	Convert Electrical Power	Electrically Powered Device	Configuration Setting
2	IE1	Consume Electrical Power	International Power Converter	Input Power
3	IE2	Convert Electrical Power	International Power Converter	Output Power
4	IE3	Consume Electrical Power	International Power Converter	Heat
5	IE4	Consume Electrical Power	International Power Converter	EMI
6	IE5	Resist Contamination	International Power Converter	Contaminants
7	IE6	Tolerate Moisture	International Power Converter	Moisture
8	IE7	Set Configuration	International Power Converter	Configuration Setting
9	IE8	Display Status and Information	International Power Converter	Device Status and Information
10	IE9	Handle Device	International Power Converter	Handling Force
11	IE10	Consume Electrical Power	Local Power Distribution System	Input Power
12	IE11	Convert Electrical Power	Electrically Powered Device	Output Power
13	IE12	Handle Device	Device and Converter User	Handling Force
14	IE13	Set Configuration	Device and Converter User	Configuration Setting
15	IE14	Display Status and Information	Device and Converter User	Device Status and Information
16	IE15	Resist Contamination	Local Environment	Contaminants
17	IE16	Tolerate Moisture	Local Environment	Moisture
18	IE17	Consume Electrical Power	Local Environment	EMI
19	IE18	Consume Electrical Power	Local Environment	Heat
20	IE20			



- Choose the related element from the containment browser.

Exercise: Interface Context Modeling: Connecting the Elements: Table ICT2



Reliability and Durabili... | Power Mains Compatibilit... | Pattern Interaction Stat... | Pattern Interface Contex.. | **Pattern Interface Contex...** | Pattern Interface Contex...

Add New | Add Existing... | Delete | Remove From Table | Columns | Export

Criteria

Element Type: Interface Element | Scope (optional): 02 Logical System Analysis | Filter:

#	Name	Interaction Name	System Name	IO Name
1	IE19	Convert Electrical Power	Electrically Powered Device	Configuration Setting
2	IE1	Consume Electrical Power	International Power Converter	Input Power
3	IE2	Convert Electrical Power	International Power Converter	Output Power
4	IE3	Consume Electrical Power	International Power Converter	Heat
5	IE4	Consume Electrical Power	International Power Converter	EMI
6	IE5	Resist Contamination	International Power Converter	Contaminants
7	IE6	Tolerate Moisture	International Power Converter	Moisture
8	IE7	Set Configuration	International Power Converter	Configuration Setting
9	IE8	Display Status and Information	International Power Converter	Device Status and Information
10	IE9	Handle Device	International Power Converter	Handling Force
11	IE10	Consume Electrical Power	Local Power Distribution System	Input Power
12	IE11	Convert Electrical Power	Electrically Powered Device	Output Power
13	IE12	Handle Device	Device and Converter User	Handling Force
14	IE13	Set Configuration	Device and Converter User	Configuration Setting
15	IE14	Display Status and Information	Device and Converter User	Device Status and Information
16	IE15	Resist Contamination	Local Environment	Contaminants
17	IE16	Tolerate Moisture	Local Environment	Moisture
18	IE17	Consume Electrical Power	Local Environment	EMI
19	IE18	Consume Electrical Power	Local Environment	Heat

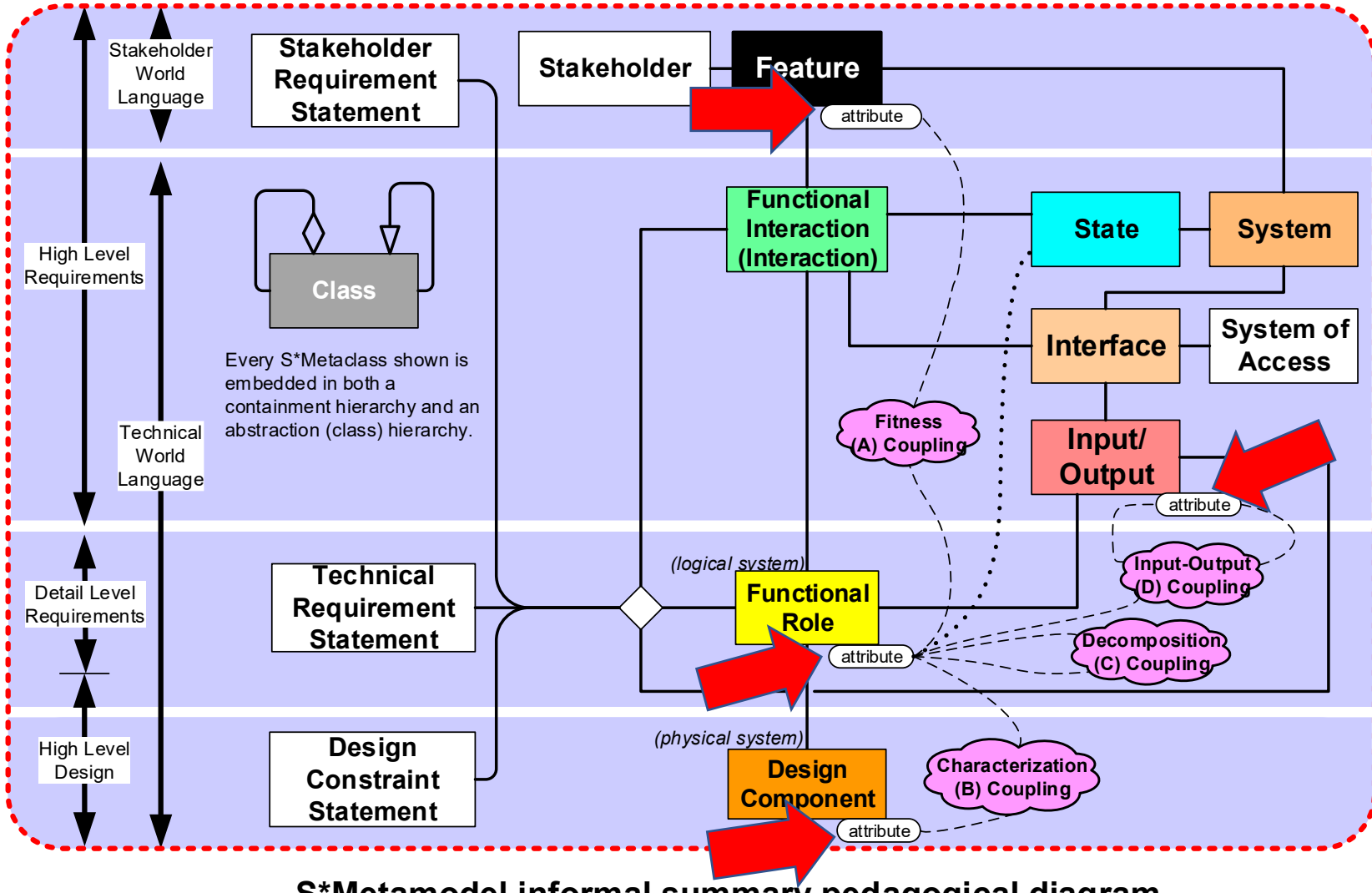
Example Pattern Interface Context File ICT2

BREAK

More about Attributes: Capturing quantitative science, fitness, and knowledge gaps using parametric attribute couplings

Examples of Each Attribute Type:

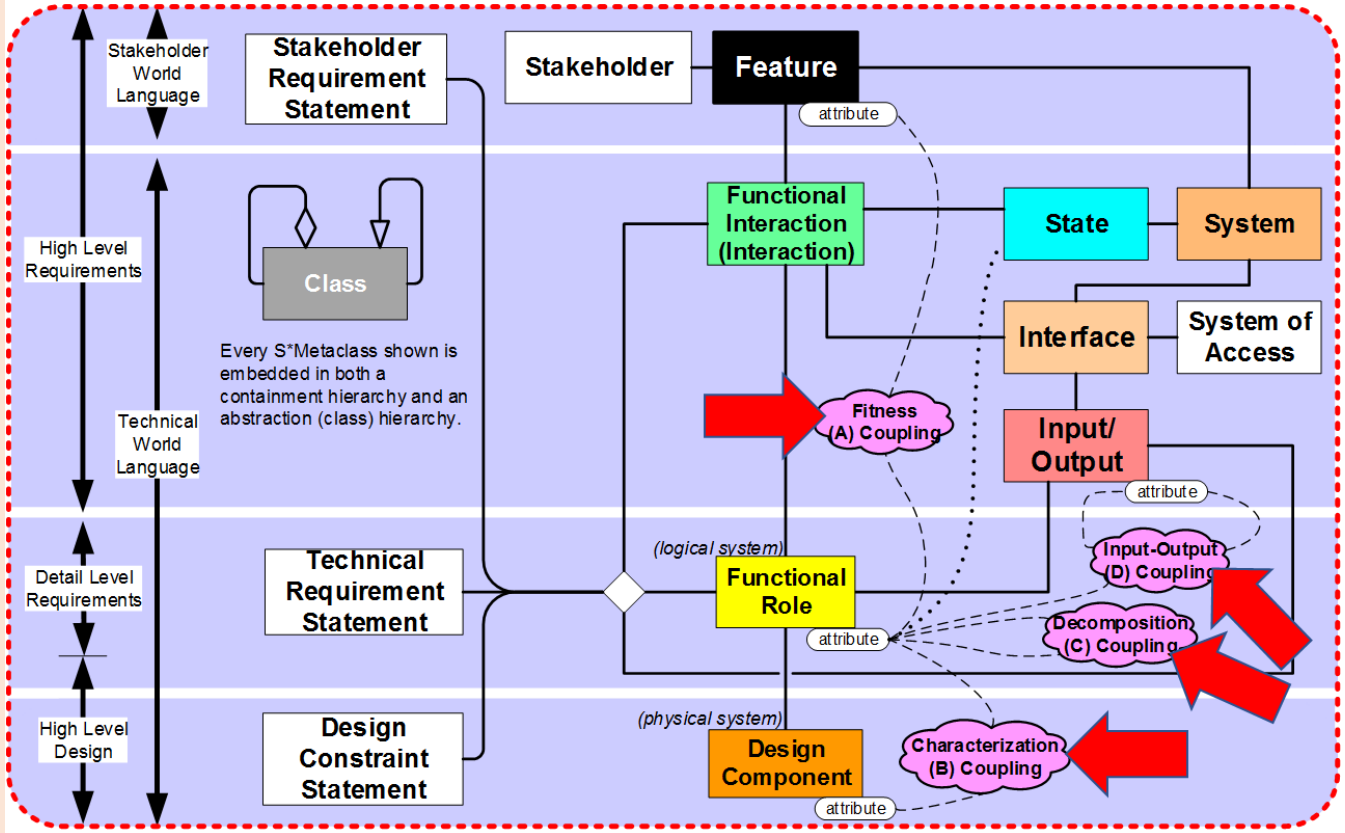
- **Stakeholder Feature Attributes**: Fitness of system; e.g., Aircraft Operational Ceiling; Maximum Speed at Altitude; Range
- **Functional Role Attributes:** Parameters of technical behavior, static or dynamic; e.g., Aircraft Fuel Consumption at Altitude Maximum Thrust, Aircraft Drag at Max Speed and Altitude.
- **Input/Output Attributes:** Value of an Input/Output parameter; e.g., Aircraft Instantaneous Trust.
- **Design Component Attributes**: Identity, as in material of composition, part number, chemical identity.



S*Metamodel informal summary pedagogical diagram
(formal S*Metamodel includes additional details.)

Examples of Each Attribute Coupling Type:

- **Fitness Couplings**: How is technical behavior valued by stakeholders? e.g., Ceiling-Thrust Coupling
- **Decomposition Couplings:** (AKA Emergence Couplings) How does component or subsystem performance impact system performance? e.g., Velocity-Thrust-Drag Coupling.
- **Characterization Couplings**: How does the identity of material, chemical composition, or part number predict behavior of same item? e.g., Engine Thrust as function of Engine Type.
- **Input-Output Couplings**: How does a role input impact a role output? e.g., Fuel Consumption Rate as a function of Velocity and Altitude.



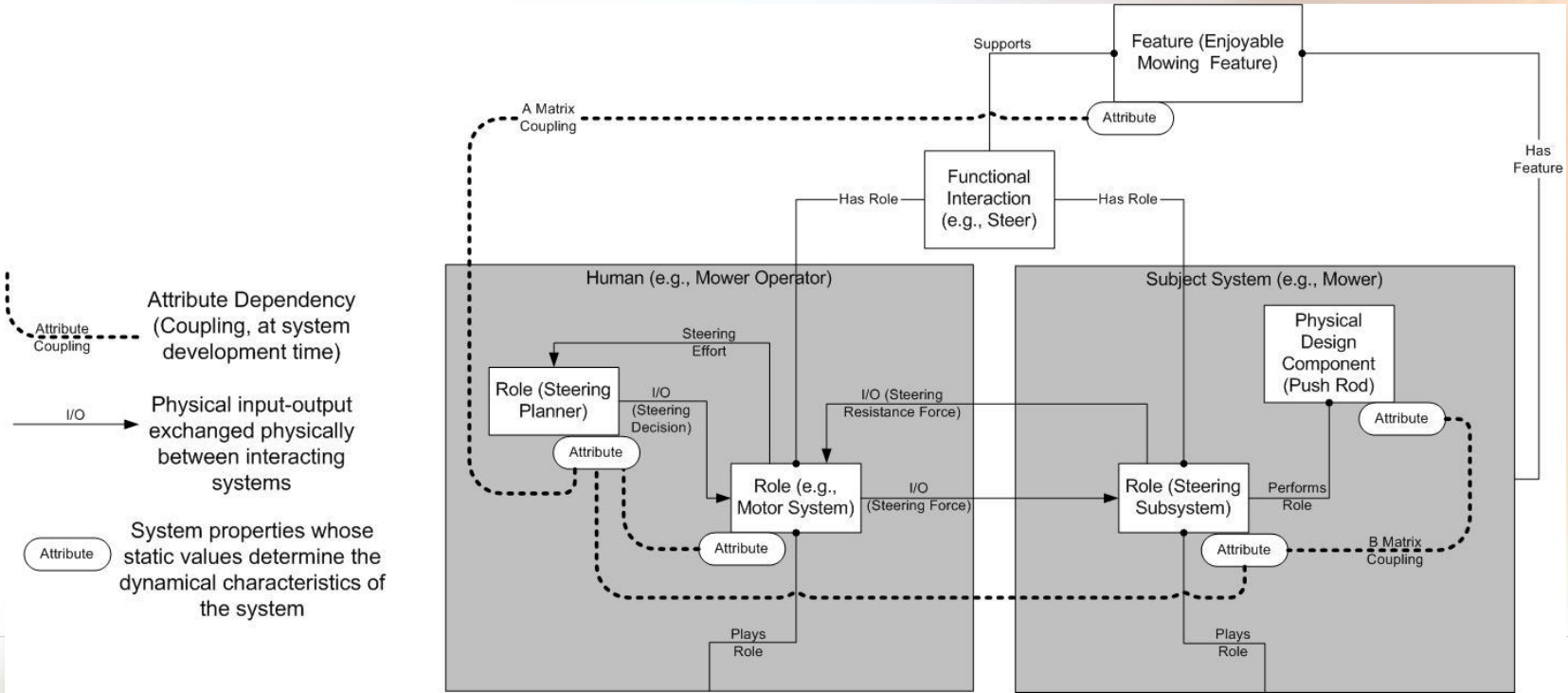
S*Metamodel informal summary pedagogical diagram
(formal S*Metamodel includes additional details.)

A simple example

Operator fatigue, sense of control, coupled to technical steering gain, mower speed:

- "Enjoyable Mowing Feature" attributes
  are coupled to attributes of . . .
- "Operator Steering Planner" role
- "Operator Motor System" role
- "Mower Steering Subsystem" role
  which are coupled to attributes of . . .
- "Mower Steering Push Rod" component

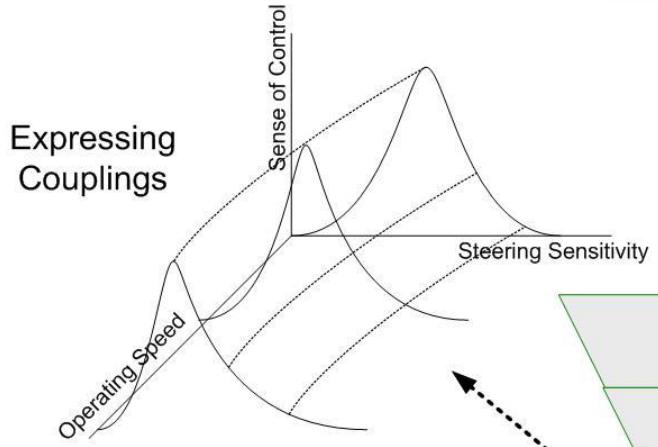




A simple example

- Expressing these couplings as tables, charts, graphs, or otherwise captures our best currently available knowledge of human behavior as well as mechanics.
- Creates integrated view contributed to & shared across a team of specialists in humans versus machine design.





Features

Feature Attributes

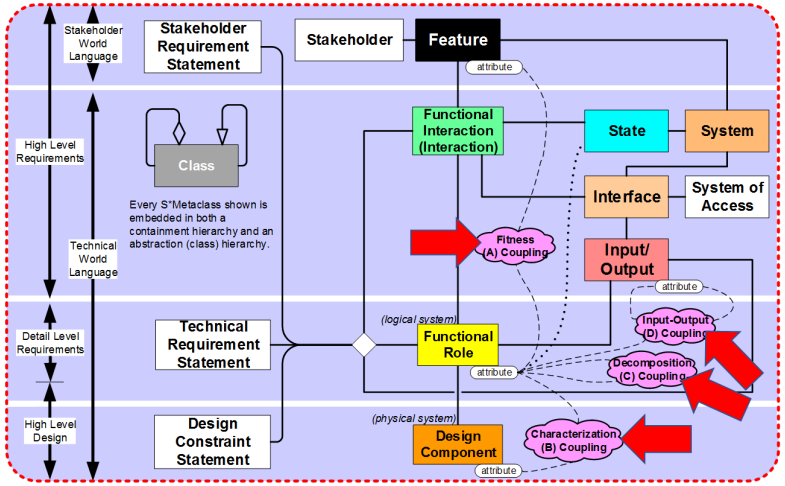
Functional Roles

Role Attributes

Attribute Couplings

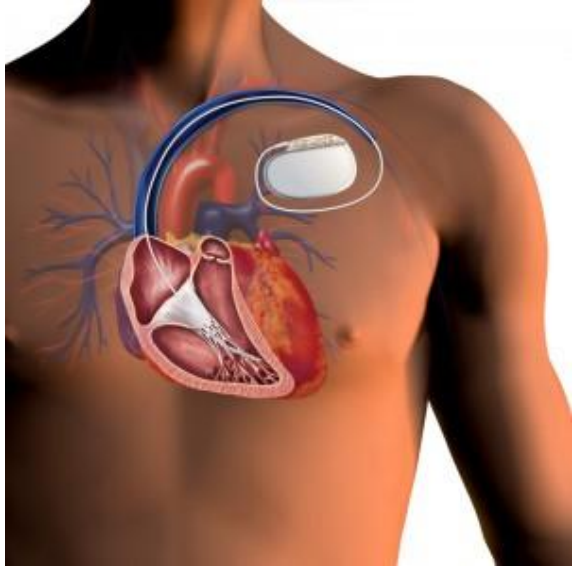
		Enjoyable Mowing Feature			Market Segment Fit Feature	
		Sense of Control	Operator Fatigue	Operator Comfort	Application Positioning	Purchase Price
Application Environment	Terrain Roughness			X	X	
Steering Subsystem	Steering Tactile	X				
	Steering Sensitivity	X	X		X	
	Cost					X
Seating System	Shock Absorbence		X	X	X	
	Seat Cushioning		X	X	X	
	Cost					X
Propulsion System	Max Acceleration	X	X		X	
	Max Mowing Speed	X	X		X	
	Braking	X			X	
	Cost					X

"A" Matrix: Expresses Feature-Role Attribute Couplings



S*Metamodel informal summary pedagogical diagram
(formal S*Metamodel includes additional details.)



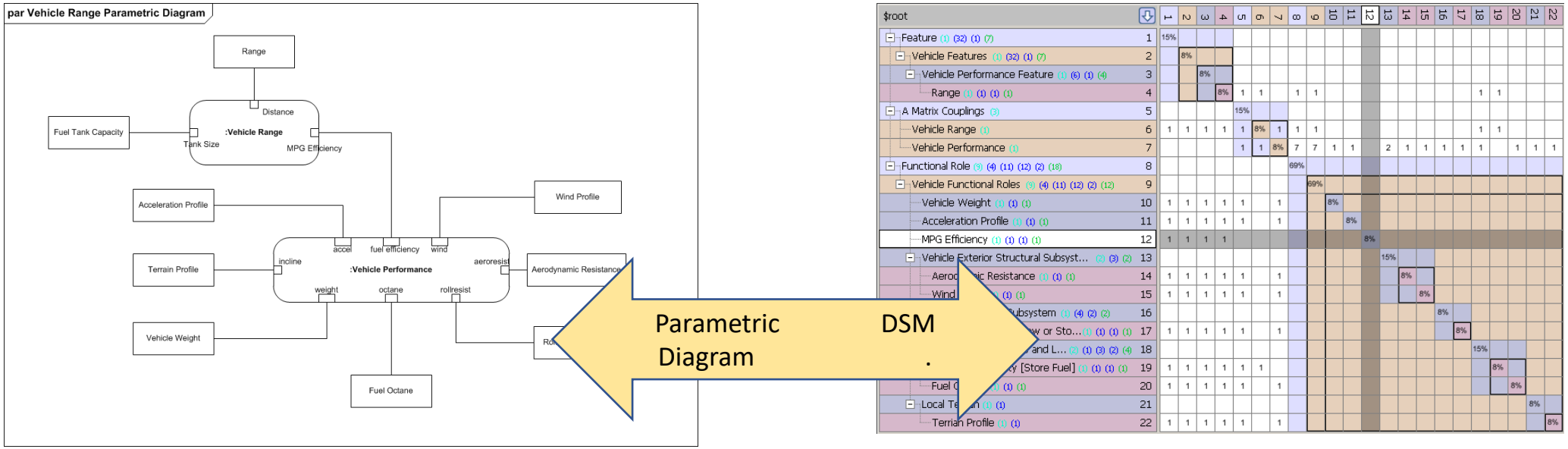


Hints for modeling Attribute Couplings

- Try to identify all the parameters (attributes) first, and after that, their couplings.
- Fitness (stakeholder value, risk, optimization, product line variation) couplings
- Characterization couplings (for materials science, aerodynamic, electronics, humans)
- Decomposition couplings (for emergent behaviors)
- Input-Output couplings (for transformations, conversions)
- You don’t need to know the quantitative basis for coupling to get value from its identification.
- Some couplings are part of Requirements Models, and some part of Design Models.
- For a product line family, consider a standard pattern of (configurable) attribute couplings.

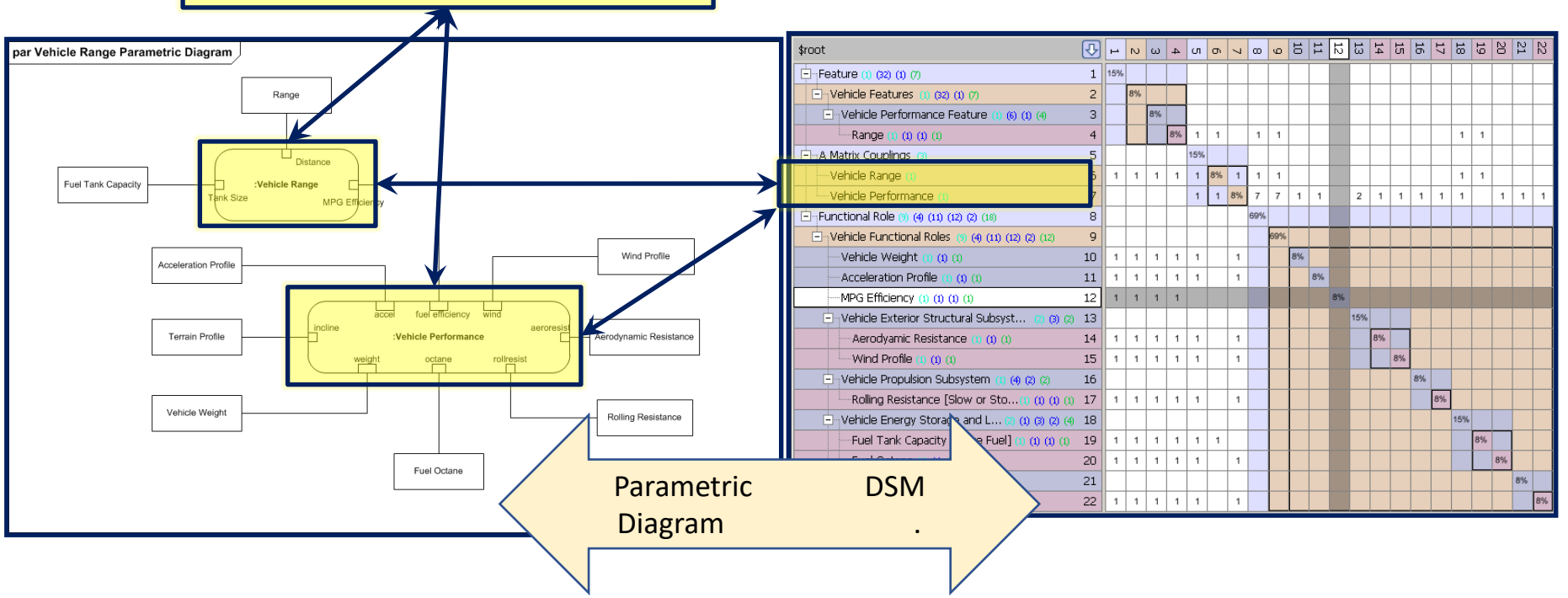
DSM View of Attribute Couplings (Parametric Couplings)

- In the case of Coupled Parameters (attributes):
  - Attributes become row and column headings in the DSM
  - This includes adding rows and columns to the Logical Architecture DSM, showing attributes of the Logical Subsystems
  - Connection lines in the drawing become marked cells in the DSM
- Both views convey the same information:
  - Which attributes are coupled (impact each others' values)

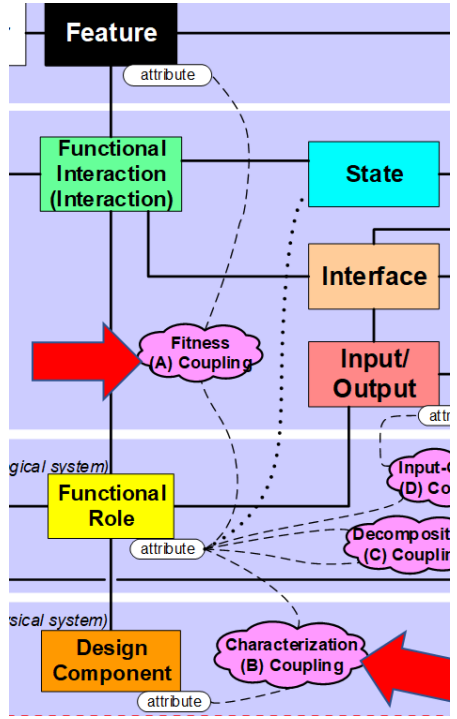


Domain Structure Matrix (DSM) View of Same

- Instead of just showing which attributes are coupled, the DSM (like the Parametric Diagram) can also symbolize the named Coupling that connects them:
  - This provides a reference to a (separately documented) quantitative coupling description.
- The names of the couplings can be introduced as row and column headings, separate from the rows and columns that list the attribute names:
  - This becomes a Multi-Domain Matrix (MDM):



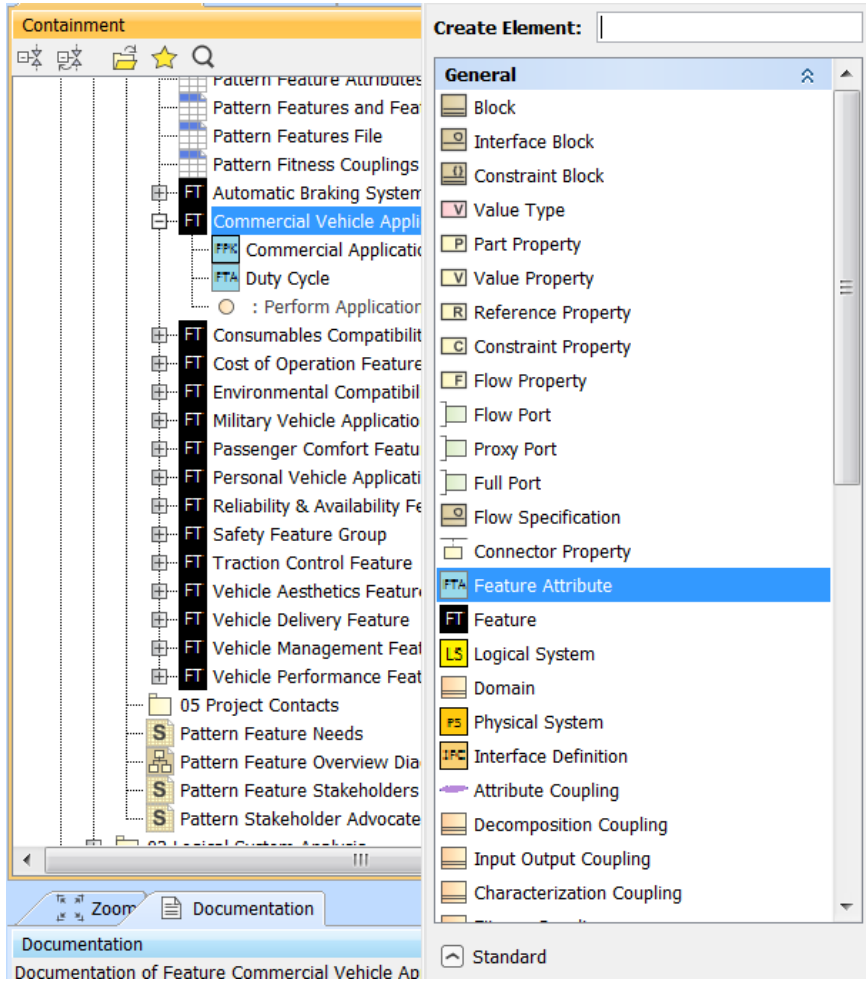
Simple exercise:
Populating Attributes and Attribute Couplings



- Add a modeled Feature Attribute to a Feature.
- Add a modeled Role Attribute to a Functional Role
- Add a modeled Design Component Attribute to a Design Component
- Populate a (Fitness) Attribute Coupling between your modeled Feature and Role attributes above.
- Populate a (Characterization) Attribute Coupling between your modeled Design Component Attribute and Role Attribute.

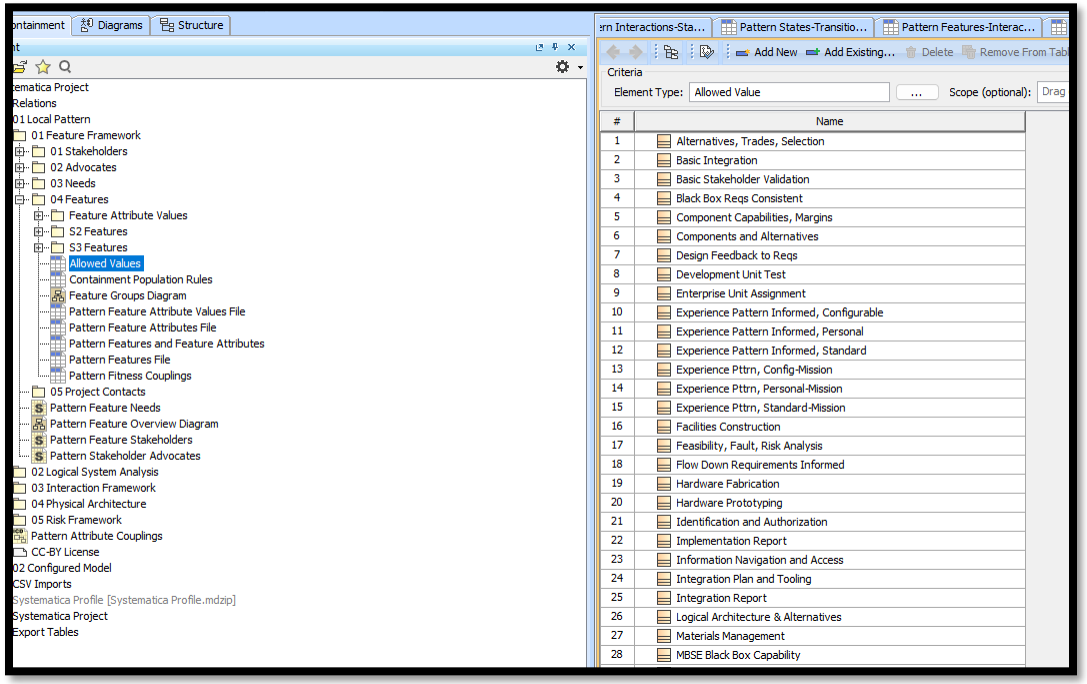
Feature Framework Modeling: Creating Feature Attributes

- Features may have Feature Attributes. A special type of Feature Attribute is called a Feature Primary Key, which is the Feature Attribute that is used to store the Feature's unique identifier.
- Select a Feature and use ~Create Element and choose either Feature Attribute or Feature Primary Key
- If adding a Feature Primary Key, enter the name of the Feature Primary Key into its Feature's Primary Key Name tag.



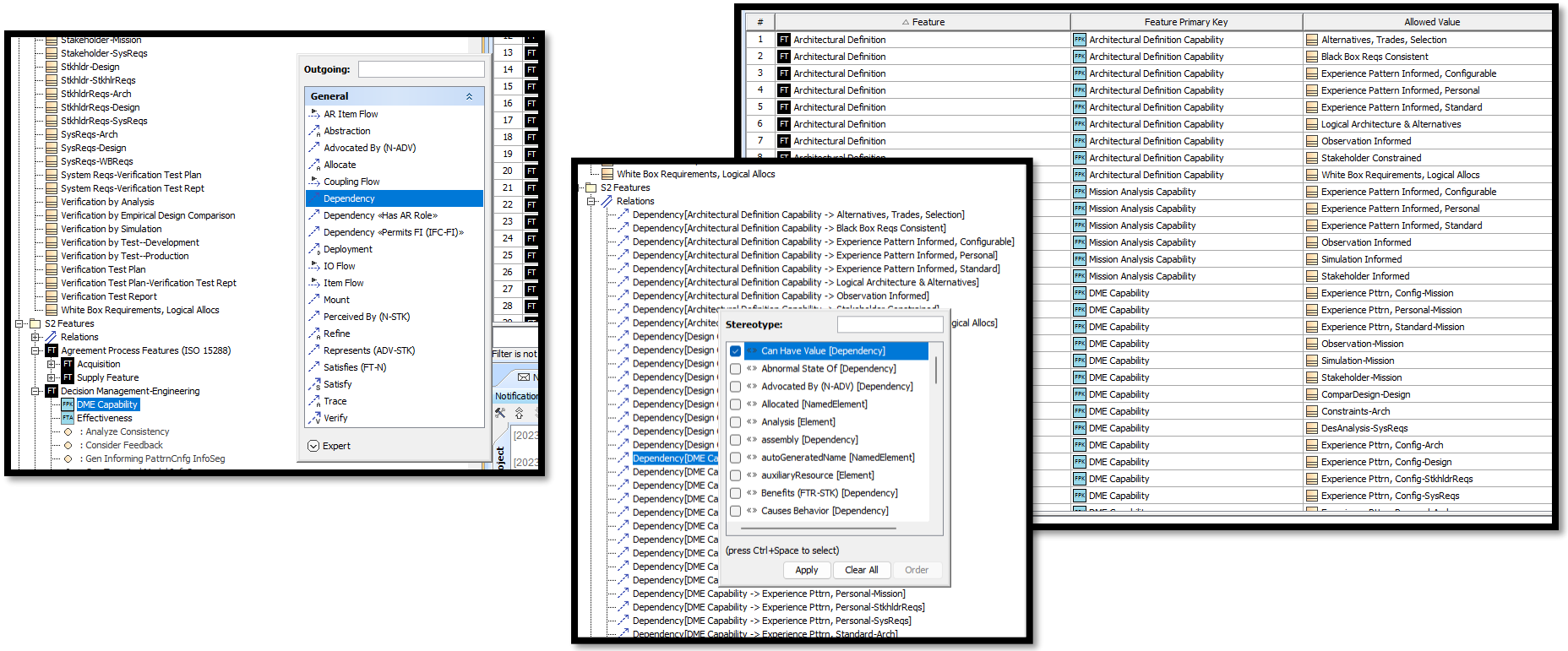
Feature Framework: Feature Attribute Allowed Values

- An alternative method for creating Feature Attributes and connecting them to Features for configuration includes the use of allowed values.
- Enter the complete list of Feature Attribute Values in the Allowed Values Table.



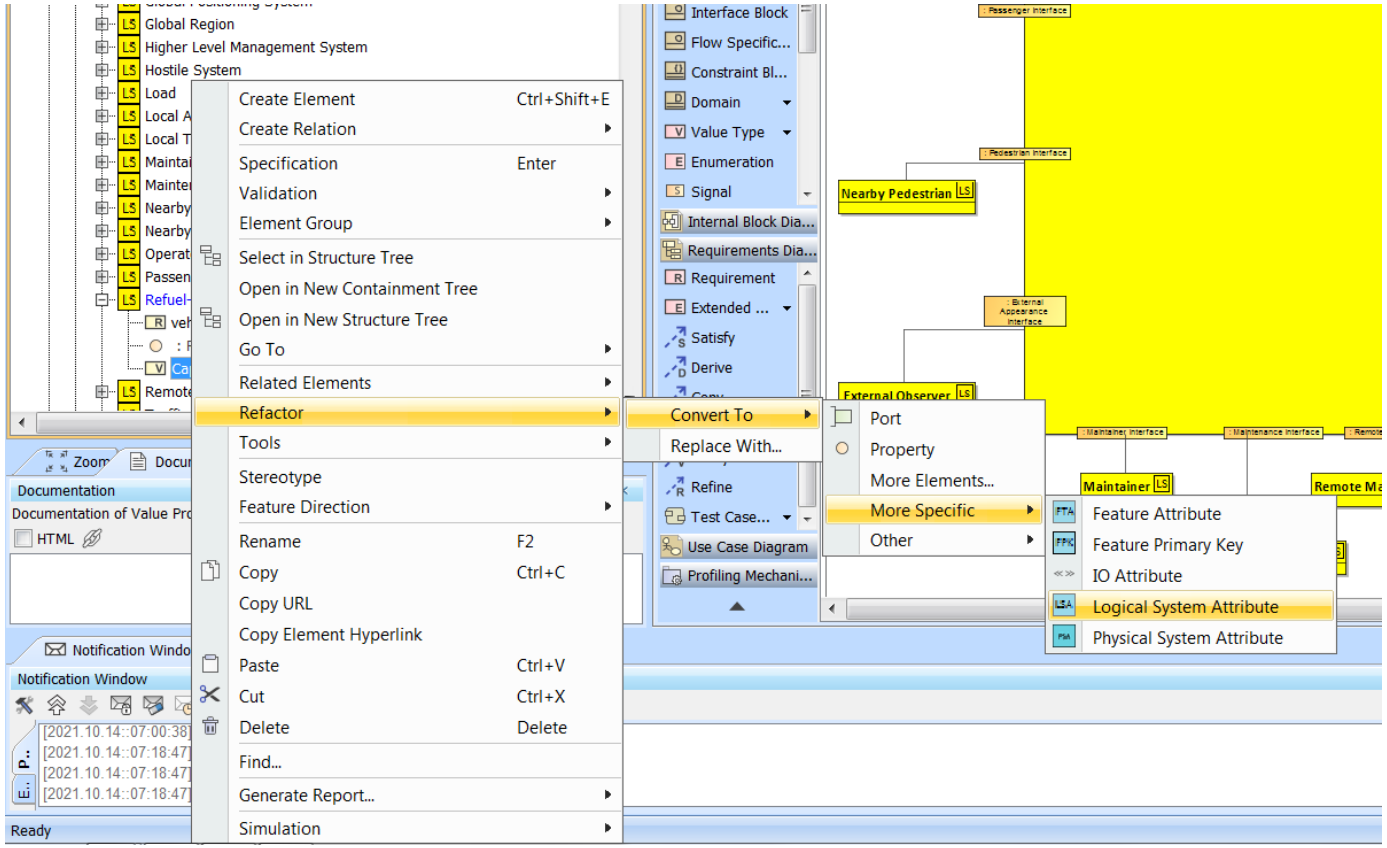
Feature Framework: Feature Attribute Allowed Values

- Create a Dependency relationship between the Feature Primary Key and Feature Attribute Value.
- Set the Stereotype of the relationship to Can Have Value.
- View the result in the Pattern Feature Attribute Values Table.



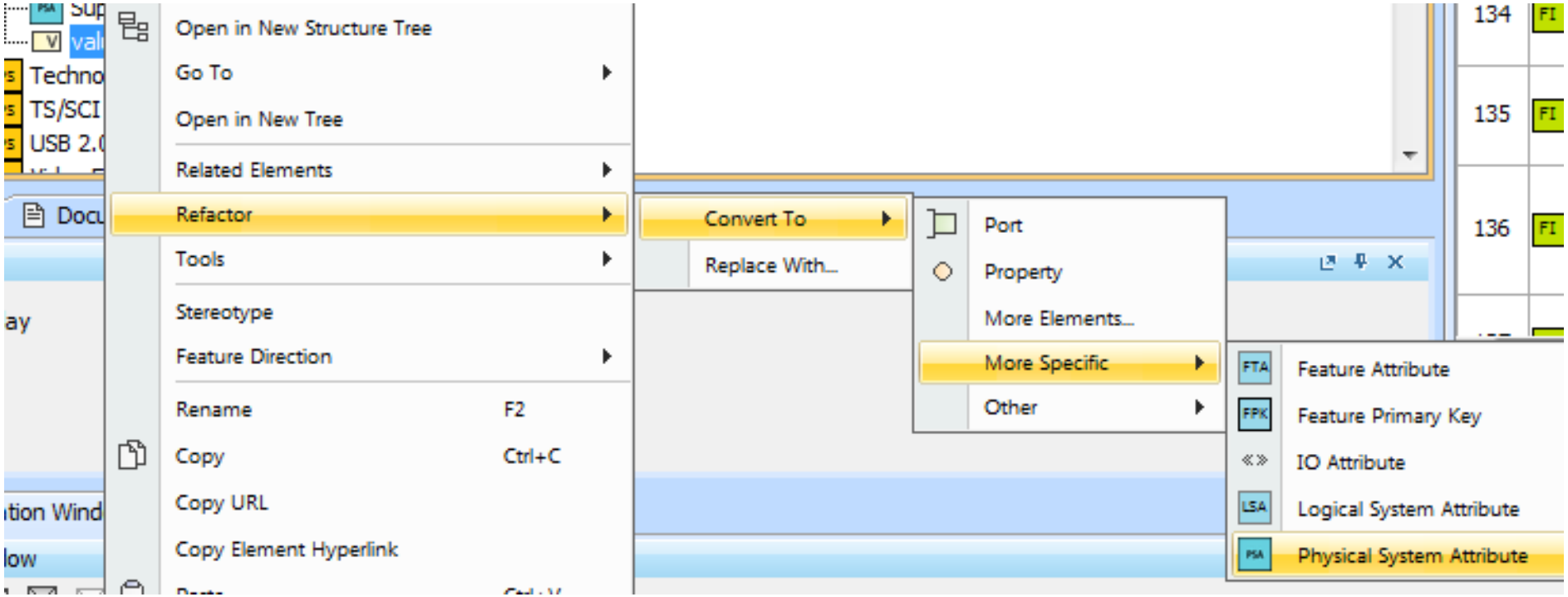
Functional Role (Logical System) Modeling: Creating Logical System Attributes

- Select a Logical System under the 01 Local Pattern::02 Logical System Analysis package
- Use ~Create Element and choose Value Property
- Right click and refactor the new Value Property as a Logical System Attribute



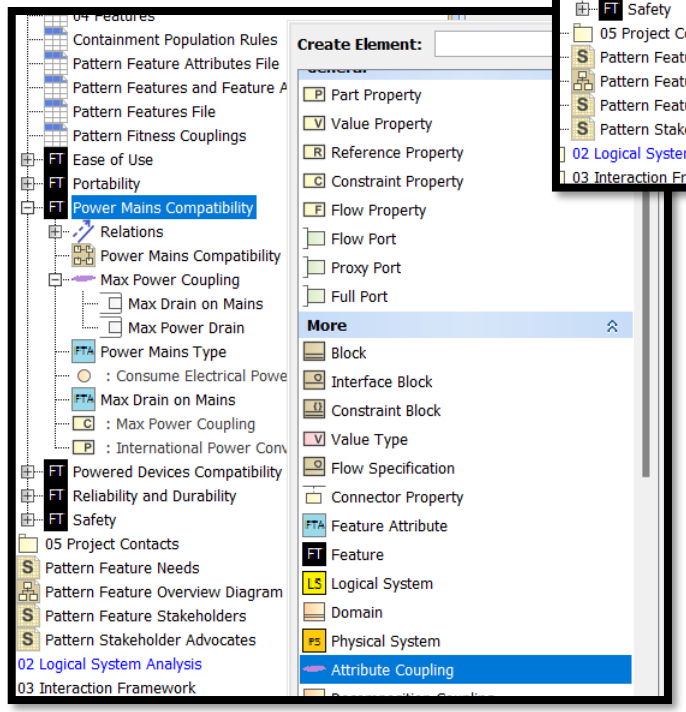
High Level Design Modeling: Creating Design Component Attributes

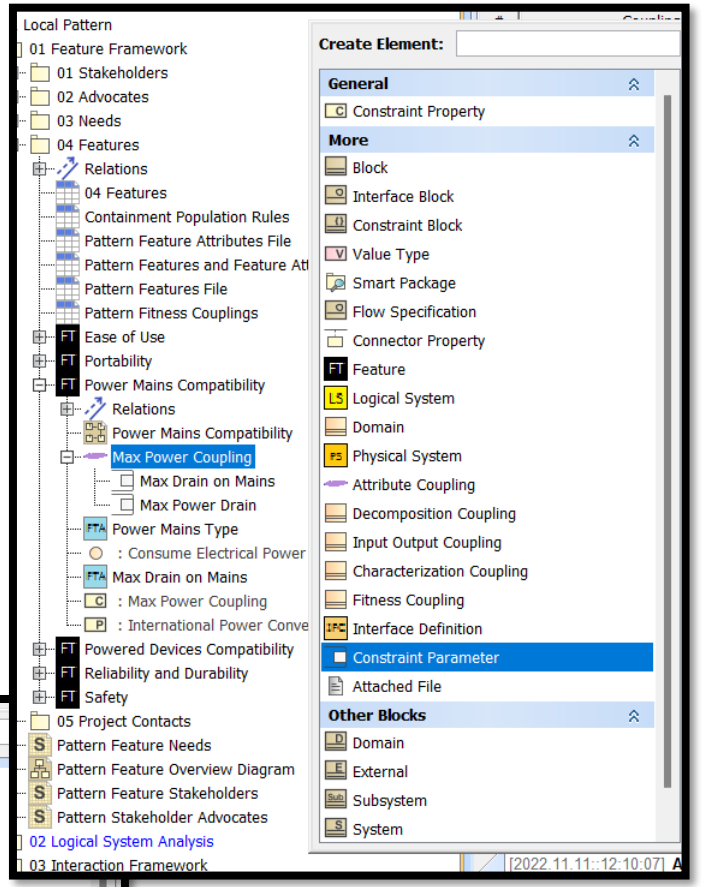
- Use ~Create Element and choose Value Property
- Right click and refactor the new Value Property as a Physical System Attribute



Attribute Coupling Modeling: Attribute Couplings

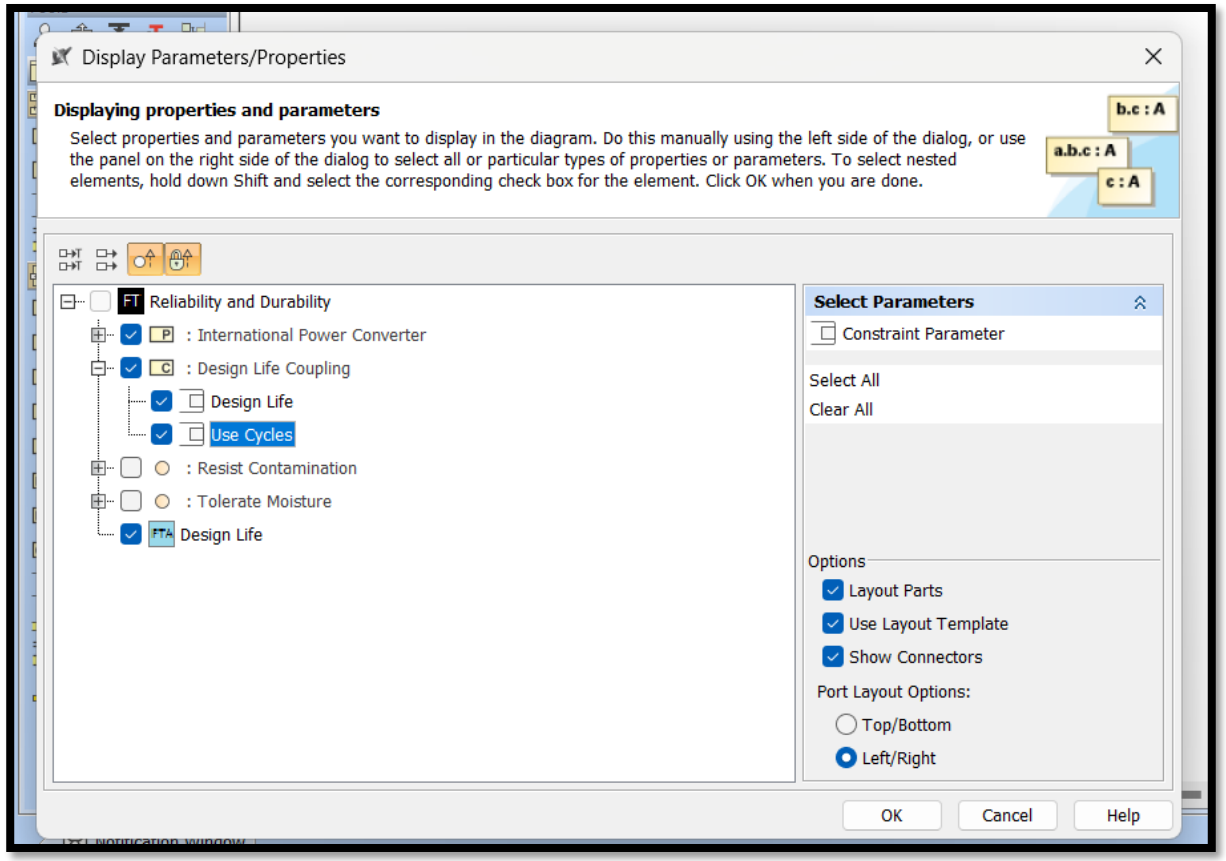
- Attribute Couplings are created under the element that owns them in various packages including Features, Roles, Design Components, and Input-Outputs:
  - A given Attribute Coupling involves multiple Attributes, so which one belongs to the Coupling's owner?
  - Each Coupling drives (in a causality sense) only one Attribute—look there for the Coupling's ownership.
- Use ~Create Element and choose Attribute Coupling.
- Right-click on the new Attribute Coupling, use ~Create Element and choose Constraint Parameter. Give it the same name as the attribute it will be connected to.
- Add the two or more constraint parameters for the coupling(s).

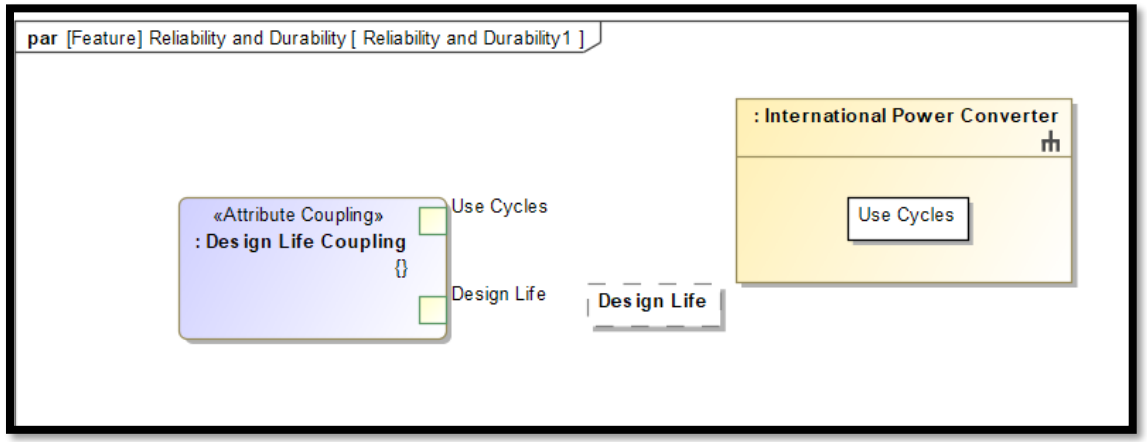


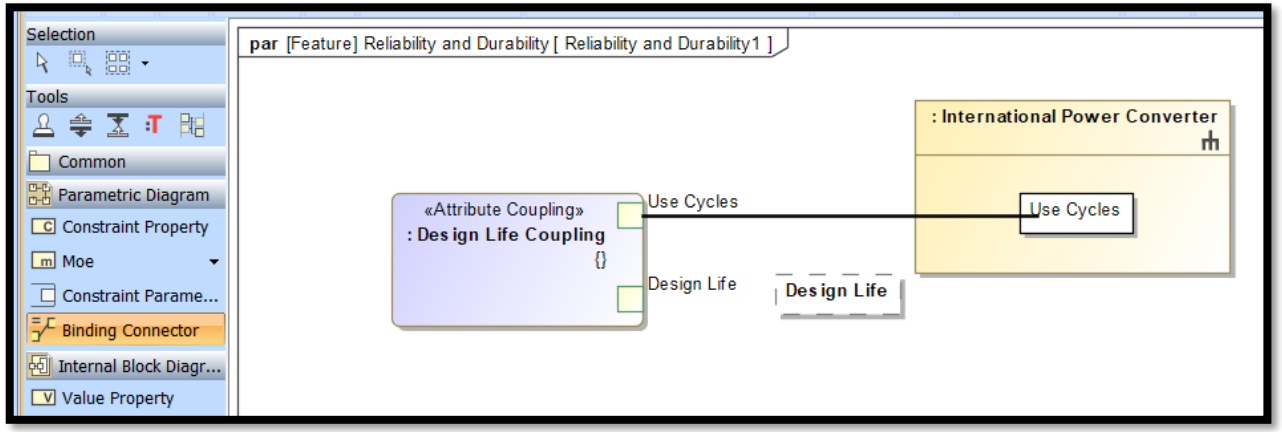


Attribute Coupling Modeling: Attribute Couplings

- Right-click on the element that owns the new Attribute Coupling, use ~Create Diagram and choose SysML Parametric Diagram.
- Choose the associated display parameter details as shown in the box to the right.
- View the resulting populated diagram as shown below.
- Create binding connectors from the constraint parameters to the associated attributes as shown in the lower right.







Homework assignment 2

Save results for use in the next class HW

Start with last week's homework system model results (System of Interest, External Actors, External Interactions, Stakeholder Features), and make the following additions:

- Add one or more external Interfaces, with Input-Outputs to the External Actors.
- Add some black box Requirements for the System of Interest, for the modeled Interactions.
- Add some actor Requirements (assumptions) for some external Actors, for the modeled Interactions.
- Decompose the system of interest to logical subsystems and add architectural relationships between them, creating a Logical Architecture model.
- Add at least one Design Component and allocate at least one logical subsystem to it.
- Add at least one attribute to:
  - A modeled Feature;
  - The System of Interest (black box);
  - A Subsystem;
  - A Design Component.
- Add at least one Attribute Coupling of each of the following
  - A Fitness Coupling
  - A Decomposition Coupling
  - A Characterization Coupling

Discussion, Q&A.

-
-
-
-
-
-



References

1. Requirement Statements are Transfer Functions: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:incose_2005.pdf
2. System Interactions: Making the Heart of Systems More Visible: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:system_interactions--making_the_heart_of_systems_more_visible_v1.2.2.pdf
3. Systematica Metamodel: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:systematica_8_metamodel_v8.0.61.pdf
4. What Is the Smallest Model of a System?" https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:what_is_the_smallest_model_of_a_system_v1.4.4.pdf
5. A Systems Engineering Insight from Ancient Navigators: Maps vs. Itineraries: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:maps_vs._itineraries_v1.3.1.pdf
6. Guide to Systematica Patterns in Cameo Systems Modeler: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:guide_to_systematica_patterns_in_cameo_systems_modeler_v1.4.6.pdf