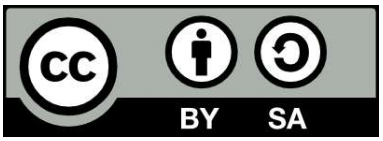
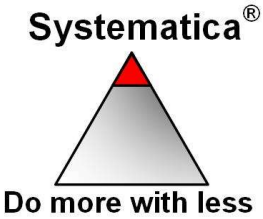
Systematica® Metamodel

Metamodel Version 8.0

2/6/2024





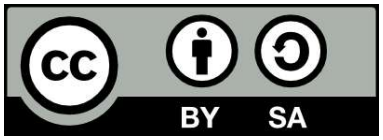


TABLE OF CONTENTS

1 Introduction

1.1 Document Purpose

This document describes the information model of the Systematica® systems engineering methodology at a conceptual level. Its intent is to provide the summarized and detailed views of Systematica information semantic structures and define the entities and relationships shown in those or related views. The intended audience of this document is a system engineering methodologist concerned with defining the underlying information structure supporting a methodology for an organization, or for a reference model for exchanges between organizations.

1.2 Document Scope

This document is at a conceptual level. No preferences to specific data model designs or software tool paradigms are intended, as this document should be read as a guidance and standard for any Systematica methodology implementations from pencil and paper to advanced object-oriented systems. This document also does not describe the methodology processes that develop, use, or maintain the information modeled herein; please refer to the references below for Systematica process descriptions and guidance. Instead, this document solely concentrates on explaining the information and concepts any Systematica user will need, independent of the form that that information takes.

1.3 Document Overview

- Section 1 describes the document's purpose, scope, structure, and history.
- Section 2 unveils the Metamodel by progressing from the summary view to the several detailed views of Systematica.
- Section 3 describes the classes, relationships, and attributes of the metaclasses shown in the Section 2 models.

1.4 Document References

1) "What Is the Smallest Model of a System?", in *Proc. of INCOSE 2011 International Symposium*, Denver, CO.

2) "Systematica Methodology: High Level Information & Process Models".

3) " MBSE Methodology Summary: Pattern-Based Systems Engineering (PBSE), Based On S*MBSE Models

4) "Introduction to Pattern-Based Systems Engineering (PBSE): Leveraging MBSE Techniques" in *Proc. of INCOSE 2013 Great Lakes Regional Conference on Systems Engineering*.

1.5 Document History

Date	Version	Changes
1/22/03	6.0.1	Initial Content
1/31/03	6.0.2	Edits to Views, Definitions, and Relationships
2/02/03	6.0.3	Metaclass Attributes added
2/12/03	6.0.4	Clarified text and collapsed Logical and Physical System synonyms.
7/14/05	7.0.1	Initial upgrade to Systematica 3.
12/01/07	7.0.1A	Update legends
05/29/09	7.1	Added Configurability Content
08/29/18	7.1.2	Corrected Spelling, Order errors
10/26/18	7.1.3	Corrected logos, registration marks, and branding.
11/19/18	7.1.4	Updated summary diagram to show coupling clouds, corrected meta relationship pasting errors.
03/04/19	7.1.5	Updated summary diagram to show new coupling clouds.
3/29/19	7.1.6	Corrected header formats and table of contents
1/13/22	8.0.1	Initial Upgrade to Systematica 6. Changed “Physical System” to “Design Component”
1/17/22	8.0.2	Added Interface Element Relationship Class
1/19/22	8.0.3	Separated Attributes into Common and Specific
1/21/22	8.0.4	Deprecated Emerges, Exemplifies, Is Linked By, Interacts Through, and Is Used During Relationships
1/24/22	8.0.5	Added Relates AR, Relates FI, Relates IO, and Relates LS Relationships, Updated Class Hierarchy View
1/25/22	8.0.6	Details Requirements View, Domain Analysis View
1/26/22	8.0.7	Bookmark link updates
1/27/22	8.0.8	Bookmark/Reference Updates
1/28/22	8.0.9	Cross Reference Updates for Common Attributes, Added Interface Context View, Updated High Level Design View, Design Coupling Relationship View, Design Constraint Relationship View, Functional

		Interaction View, Heading Format Consistency, Specific Attributes, Domain Analysis View, Logical Architecture View, Risk Analysis View
1/31/22	8.0.10	View descriptions, Configuration Details
2/2/22	8.0.11	Specific Attributes for Population and Configuration Rules
2/7/22	8.0.12	Deprecated EI Related content
2/9/22	8.0.13	Configuration Rules Table
2/11/22	8.0.14	CCBY-SA License addition with associated language, Metaclass/metarelationship definition updates
2/14/22	8.0.15	Attribute Coupling Metaclass additions, Risk Analysis Metaclasses
2/15/22	8.0.16	Modeled Relationship Views View
2/17/22	8.0.17	Configuration Table, Configuration Matrix
2/21/22	8.0.18	Document Formatting
2/22/22	8.0.19	Population Rules
2/23/22	8.0.20	Risk Analysis View Updates
2/24/22	8.0.21	Failure Impact, Counter Requirement Statement
2/25/22	8.0.22	Failure Mode, Is Root Cause Of Relationship
2/28/22	8.0.23	Risk Analysis View Updates
3/2/2022	8.0.24	Impacts Stakeholder, Impacts Feature, Causes Impact, Replaces
3/3/2022	8.0.25	Plays Causal Role, Causes Failure, Causes Mode, Causes Impact, Causes Behavior, Abnormal State Of
3/4/2022	8.0.26	Failure Analysis Configuration Rules
3/7/2022	8.0.27	Additional Coupling Views
3/8/2022	8.0.28	Additional Metamodel View References, Attribute Coupling Population Rules, I/O Attribute Role Metaclass
3/11/2022	8.0.29	Minor adjustments to terminology or explanations throughout.
3/12/2022	8.0.30	Adjustments to Views for improved readability
3/14/2022	8.0.31	Adjustments to diagram layout orientation

3/15/2022	8.0.32	Class Hierarchy View Update, Feature Attribute, Feature Primary Key, Logical System Attribute, Design Component Attribute, I/O Attribute Metaclass additions
3/15/2022	8.0.33	Removal of DRAFT Watermark for Beta Version Release
4/7/2022	8.0.34	State Analysis, Attribute Coupling, Fitness Coupling, Characterization Coupling, Input/Output Coupling Views Updates
11/27/2023	8.0.35	Risk Analysis Diagram
11/29/2023	8.0.36	Feature Framework View, High Level Design View, Detail Requirements View
12/1-12/4/2023	8.0.37	Deprecated Allocation Decision, Alternative, Issue, Rationale, Has Previous, Has Issue. Formalized use of “Stakeholder Requirement” instead of “Need”, and “Requirement Transfer Function” instead of “Requirement Relationship” Edited Allocated To details.
12/5-12/8/2023	8.0.38	Replaced “Modeled Relationship” with “Reified Relationship”
12/11-12/13/2023	8.0.39	Additional Columns for Table 2 Configuration Rules
12/14-12/21/2023	8.0.40	View order reorganization, Transition Relationship View addition
12/26/2023	8.0.41	Table 2 column entries
12/27/2023	8.0.42	Diagram references
12/28/2023	8.0.43	“Addresses” metarelationship, “Allowed Value” metaclass
1/2/2024	8.0.44	“Can Have Value” metarelationship
1/3/2024	8.0.45	“Provides Event Context” metarelationship
1/16/2024	8.0.46	Configuration Rules
1/17/2024	8.0.47	Configuration Rules
1/18/2024	8.0.48	Metarelationship attribute references
1/19/2024	8.0.49	Common Attributes, Specific Attributes and Configuration Rule Sets
1/22/2024	8.0.50	Common Attributes, Specific Attributes and Configuration Rule Sets
1/23/2024	8.0.51	Common Attributes, Specific Attributes and Configuration Rule Sets
1/24-2/6/2024	8.0.52-56	Table 2, Table 1, Feature Attribute Value Configuration Rule Set, Formatting, FPK Attribute Configuration Details

2 Metamodel Views

This section uncovers the Systematica Metamodel (S* Metamodel) by first reviewing an informal summary model and then by exploring a series of more detailed and formal views. The summary model is intended for training and reference situations which require a less formal description that still includes the main concepts of the Systematica Methodology. The detailed views describe the Metamodel in a formal manner. Each detail view depicts the metamodel in sometimes overlapping areas that roughly relate to Systematica process steps or artifacts. Finally, this section provides detailed views and information on how pattern classes and relationships are populated during a pattern configuration process. For explicit mappings to Systematica process or artifact views, please consult the relevant references listed in Section 1.4.

These Meta-Model views are explained in the following order:

- Summary Metamodel: The summary metamodel for informal reference and training.
- Class Hierarchy View: The formal view that depicts the class hierarchy of all metaclasses.
- General Class View: The formal view that depicts the relationships allowed for every metaclass.
- Feature Framework View: The formal view that depicts the relationships describing information concerning Stakeholders, Needs, Features, and Feature Attributes.
- Domain Analysis View: The formal view that depicts the classes and relationships relevant to model the systems in a domain, their interfaces, and the relationships and Input/Outputs between them.
- Logical Architecture View: The formal view that depicts the classes and relationships relevant to modeling a system's logical architecture.
- State Analysis View: The formal view that depicts the classes and relationships relevant to modeling a system's dynamic state behavior.
- Detail Requirements View: The formal view that depicts the classes and relationships relevant to modeling a system's detail level requirements (DLR) on a Functional Interaction basis.
- High Level Design View: The formal view that depicts the classes and relationships relevant to modeling a system's high-level design (HLD), including its physical architecture, Functional Role allocations, and design rationale.
- Interface Context View: The formal view that depicts the classes and relationships relevant to modeling a system's interfaces and related classes and relationships.
- Reified Relationship Views View: An informal view relating the following relationship views to each other. This view does not have an impact on the Metamodel and only explains how the next nine views relate.

- Reified Relationship View: The formal view that depicts the abstract classes and relationships with respect to reified relationships and statements.

- Architectural Relationship View: The formal view that depicts the classes and relationships relevant to Architectural Relationship modeling.

- Functional Interaction View: The formal view that defines the classes and relationships relevant to Functional Interactions to be specialization of those for Reified Relationships.

- Requirement Relationship View: The formal view that defines the classes and relationships relevant to Requirement Statements.

- Design Constraint View: The formal view that defines the classes and relationships relevant to Design Constraints.

- Transition Relationship View: The formal view that defines Transitions as relationships, used in state machine models.

- Attribute Coupling View: The formal view that defines the abstract classes and relationships relevant to coupling attributes.

- Fitness Coupling View: The formal view that depicts the classes and relationships used to couple Feature Attributes to Functional Role Attributes.

- Characterization Coupling View: The formal view that depicts the classes and relationships used to couple Functional Role Attributes to Design Component Attributes.

- Decomposition Coupling View: The formal view that depicts the classes and relationships used to couple Functional Role Attributes.

- Input/Output Coupling View: The formal view that depicts the classes and relationships used to couple Functional Role Attributes to Input/Output Attributes.

- Summary Pattern Configuration View: The summary view that depicts how the classes and relationships of a pattern are populated during the pattern configuration process.

- Risk Analysis View: The summary view that depicts the classes of risk analysis and how they are related to other classes.

Definitions and view references for the classes and relationships in the following views can be found in Section 3.

2.1 Summary Metamodel

The Summary Metamodel is an informal view of the S* Metamodel that covers the classes and relationships most relevant to the concepts of the Systematica Methodology. The Summary Metamodel is shown in Figure 1.

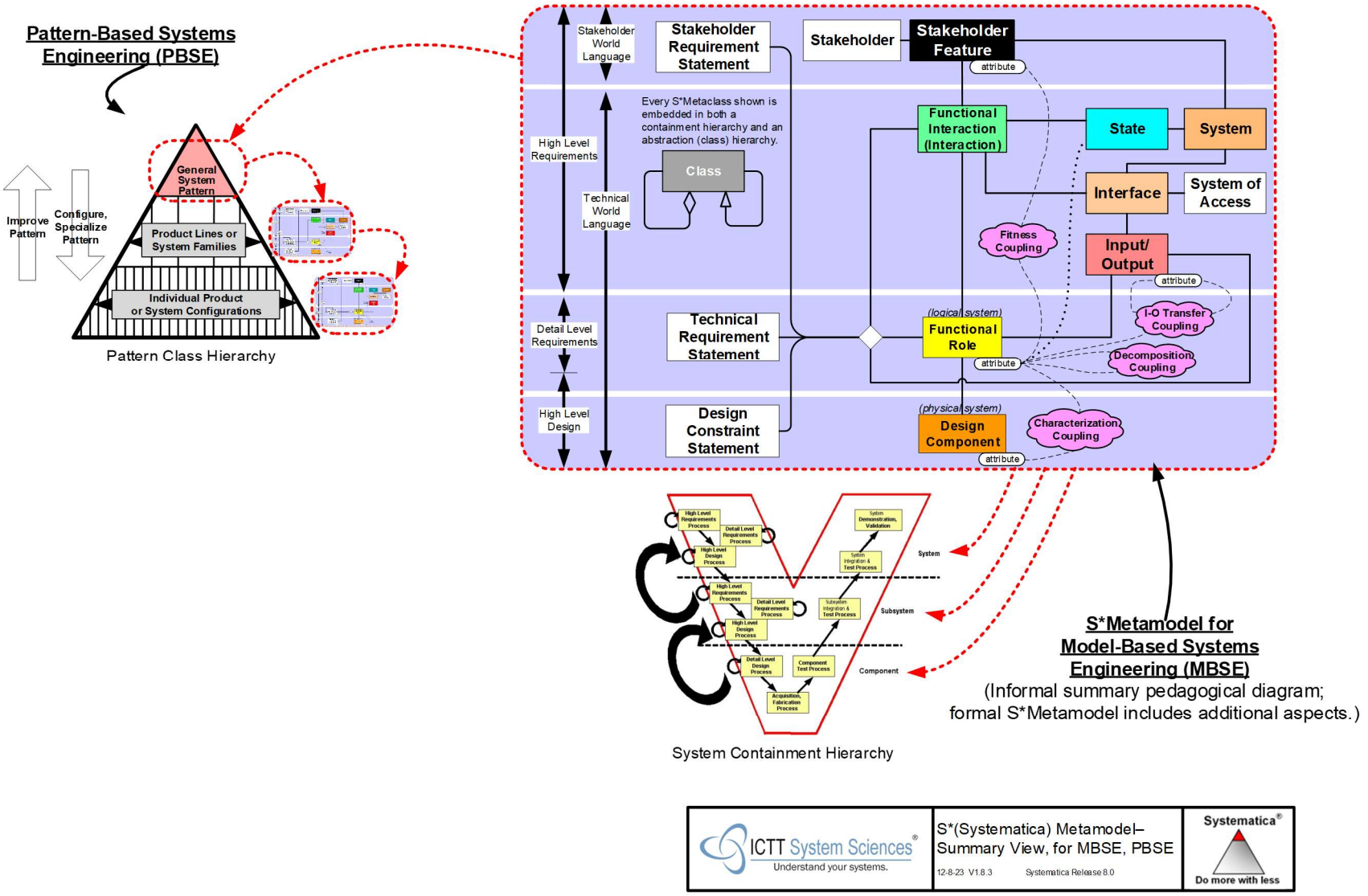


Figure 1: Summary Metamodel

The following subsections uncover the Systematica Summary Metamodel by considering a series of views of models and their related descriptions. These views get more complex as the Systematica scope of coverage increases:

- S*MBSE: Model Based Systems Engineering (MBSE), a systems engineering methodology for a single complex system.
- S*PBSE: Pattern-Based Systems Engineering (PBSE), a systems engineering methodology for a family or product line of systems.

2.1.1 Model-Based Systems Engineering (MBSE)

The Summary Metamodel view in Figure 1 shows a class web in the upper right enclosed. This web shows the classes most relevant to the methodology. The Systematica Methodology revolves around the modeling of a system. Each System has a set of Features, States, and Interfaces. Functional Interactions support the defined Features and States of a System. During these Functional Interactions, Functional Roles, which are Logical Systems, interact by transferring Input/Outputs through a System's Interface. A System's Interface model expresses the relationships between Input/Outputs, Functional Roles, and which System of Access facilitates the interactions, for interface control documentation. Requirement Statements are written with respect to a Functional Role in a context of a specific Functional Interaction. These Functional Roles are then allocated to a Design Component.

Systematica MBSE Methodology incorporates containment relationships for every class, so that each level of the System Containment Hierarchy, which is often symbolized by the Systems Engineering "Vee", can be modeled using the same metamodel.

2.1.2 Pattern-Based Systems Engineering (PBSE)

The Pattern-Based Systems Engineering (PBSE) model adds a whole model generalization and specialization capability, allowing models to be configured and specialized into separate yet related MBSE models for specific applications. An MBSE model can use the PBSE extension to define the common requirements and designs of an entire product line, system family, or even sets of product lines or system families. The pyramid in Figure 1 describes how the Systematica Metamodel can be applied at each abstraction level in the Pattern Class Hierarchy.

2.1.2.1 Domain Specific Systems Engineering

Knowledge about specific domains can be used to create generalized patterns that can be further specialized for particular systems. This can include domains such as Aerospace, Defense, Transportation, Medical Devices, Manufacturing, and Intelligence-Based Systems. Additionally, these patterns can be inherited into other patterns. For example, inheriting the Embedded Intelligence Pattern into a Manufacturing Pattern creates a configurable reusable core intelligence model for manufacturing that is also the basis for representing intelligence in other domains.

2.2 Class Hierarchy View

The first detailed, formal view of the S* Metamodel is the Class Hierarchy View in Figure 2. This view relates each of the classes in the metamodel in a class hierarchy, or generalization manner. The UML generalization line ending represents the "Is_A_Type_Of" Systematica relationship.

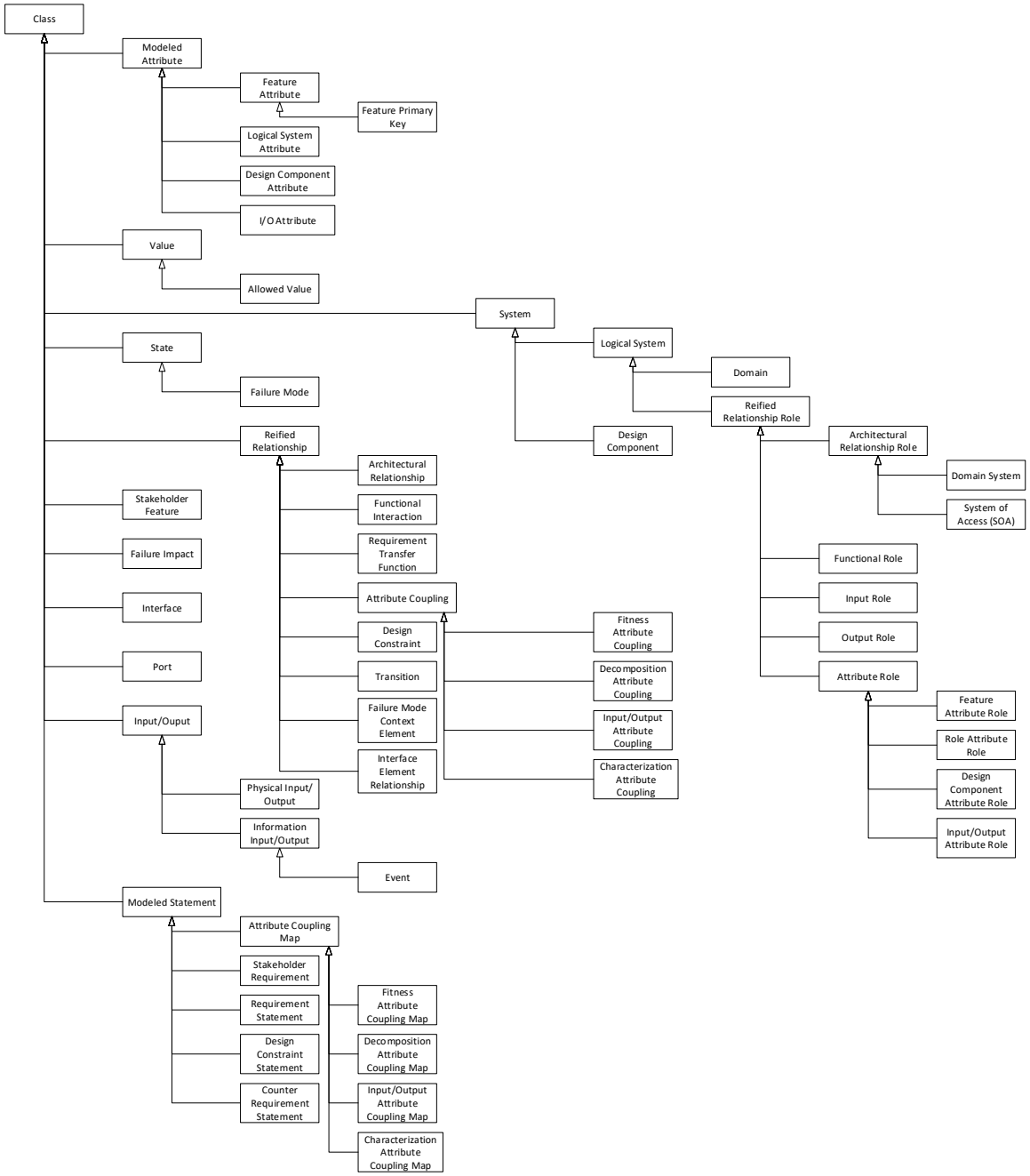


Figure 2: Class Hierarchy View

2.3 General Class View

The General Class View depicts the metamodel relationships that are relevant to all classes. As in all other views, the UML generalization line ending represents Systematica's "Is_A_Type_Of" and the UML aggregation line ending represents Systematica's "Contains" relationship. However, Systematica's "Contains" relationship is closer to UML's "Composition" concept in that a class can only have one container.

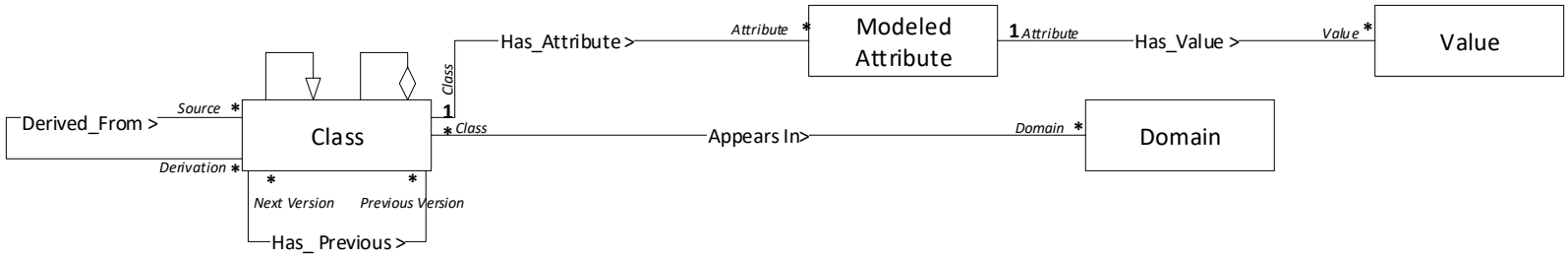


Figure 3: General Class View

2.4 Feature Framework View

Figure 4 depicts the Feature Framework View of the metamodel. This view details the classes and relationships that model the Needs Analyses and a System's Features. This view defines the framework that guides and support value-based requirements and design approaches.

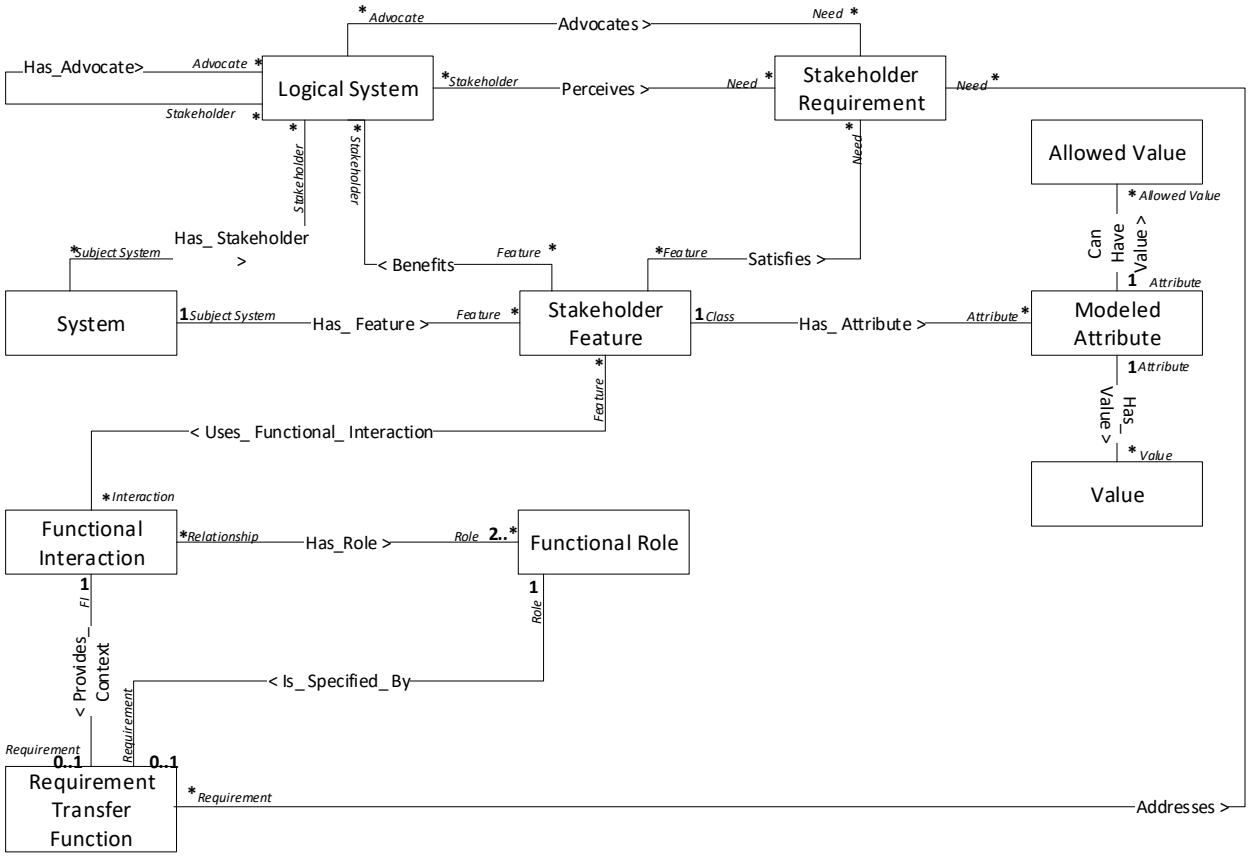


Figure 4: Feature Framework View

2.5 Domain Analysis View

The Domain Analysis View defines the classes and relationships required to model the environment of a system in a particular domain. This view corresponds to the Domain Diagram artifact but also includes other relationships and classes that would follow such a diagram to complete the system environment analysis.

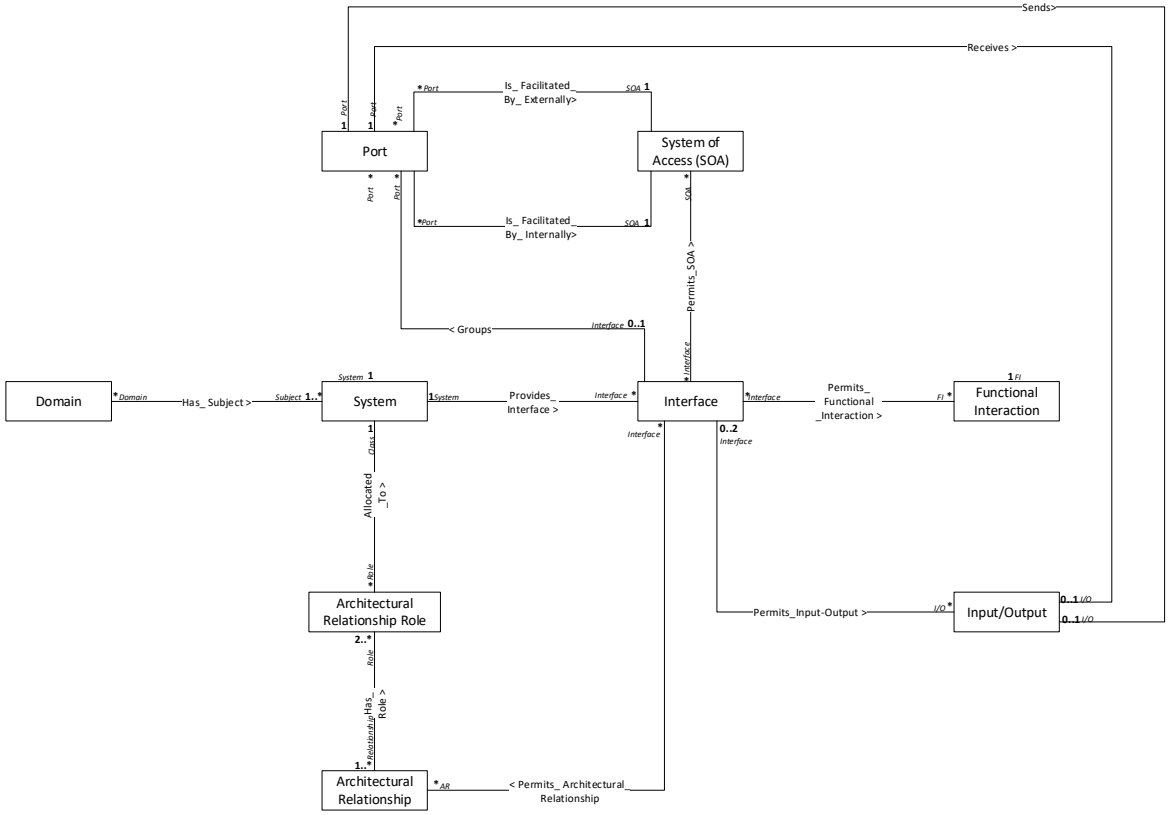


Figure 5: Domain Analysis View

2.6 Logical Architecture View

The Logical Architecture View details the part of the metamodel that decomposes a subject system in the Domain Analysis View into Logical Subsystems and their interactions that describe its externally viewable behavior.

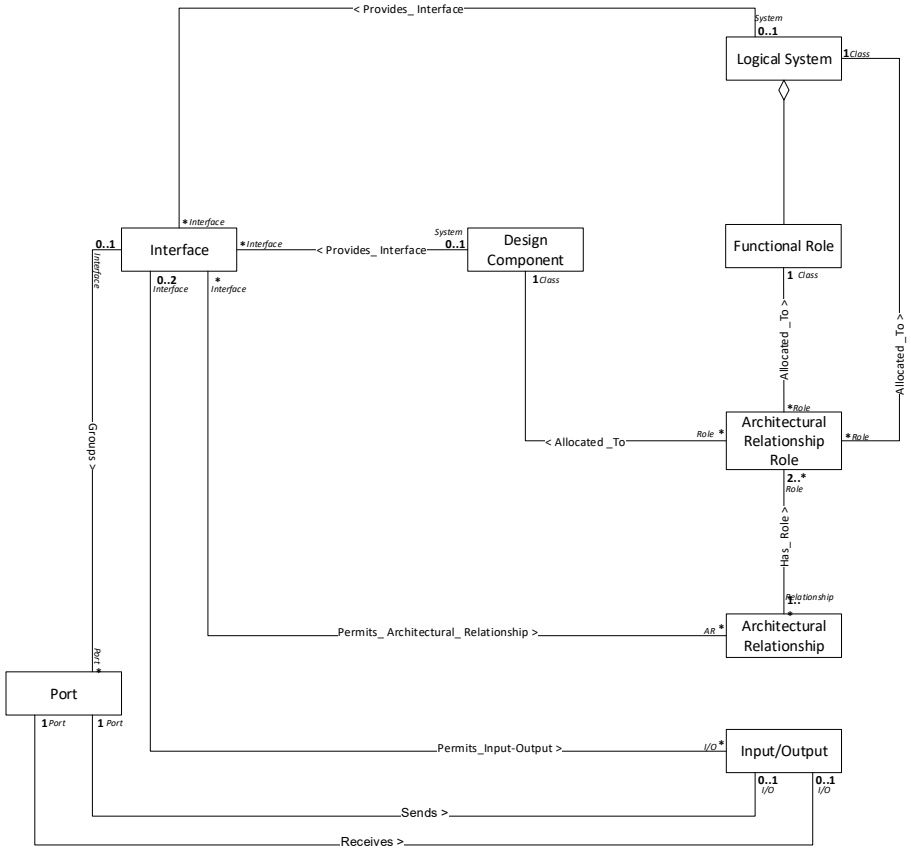


Figure 6: Logical Architecture View

2.7 State Analysis View

This figure depicts the classes and relationships modeled to define a system's dynamic behavior using classes such as States, Events, and Functional Interactions.

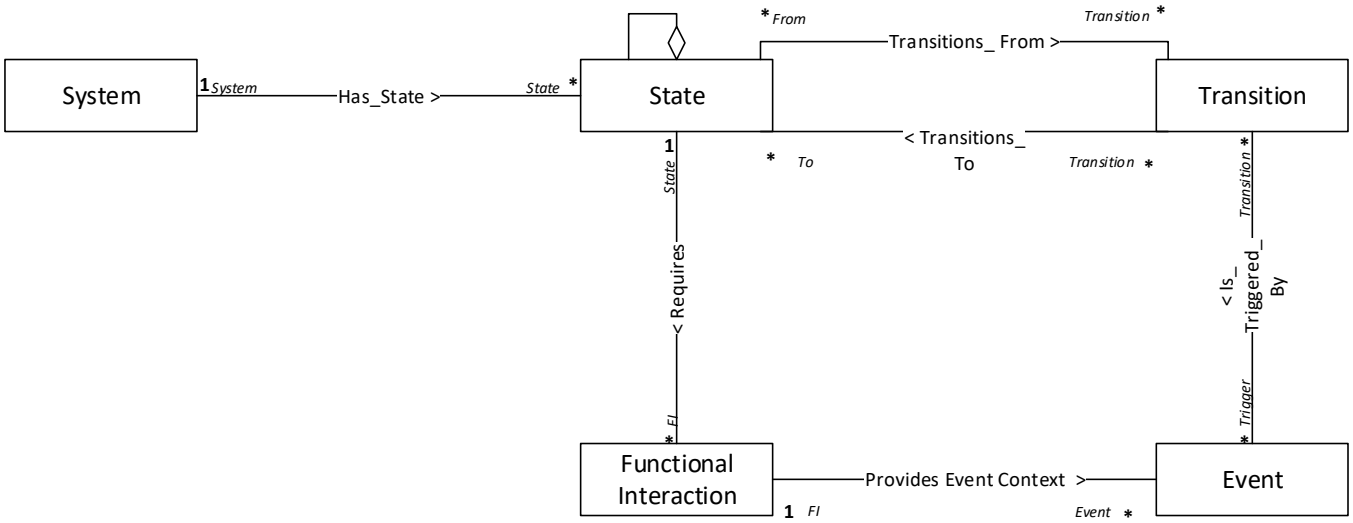


Figure 7: State Analysis View

2.8 Detail Requirements View

The Detail Requirements View defines the classes and relationships that model the detailed interactions and requirements that are summarized in the previous high-level views. Instead of being comprehensive across an entire system's scope, there should be a set of models using this view that each center on a single Functional Interaction and dive into the technical depth necessary for requirements analysis and allocation. The system's overall scope should be the union of all the scopes of the individual detail models.

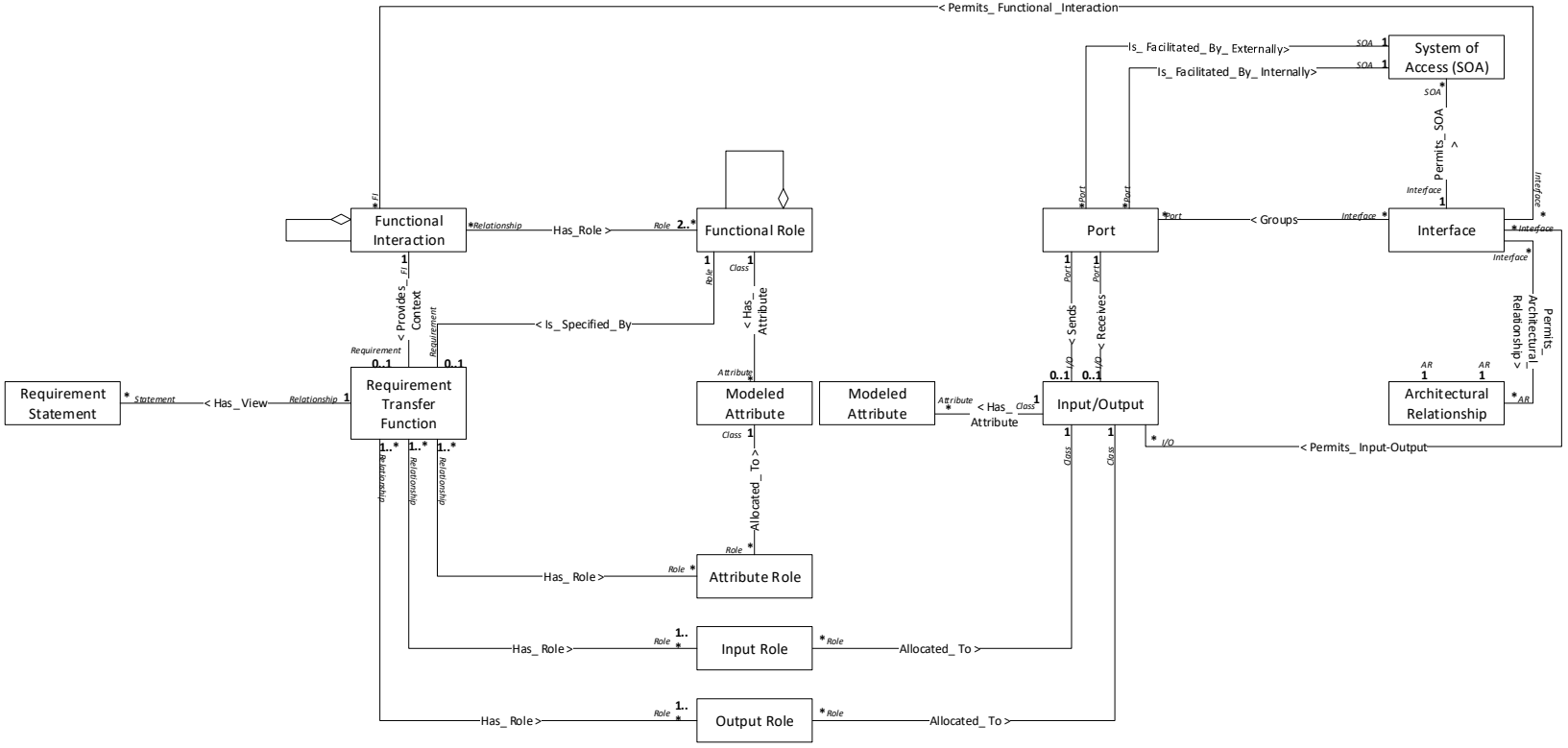


Figure 8: Detail Requirements View

2.9 High Level Design View

The High-Level Design View details the part of the metamodel that models a system's physical architecture, its Functional Role allocations, and Design Constraints. This view also shows that Requirement Statements relate to a Design Component through an allocated Functional Role. This provides for the capability to alter the design without changing the requirements or most of the models using the previous metamodel views.

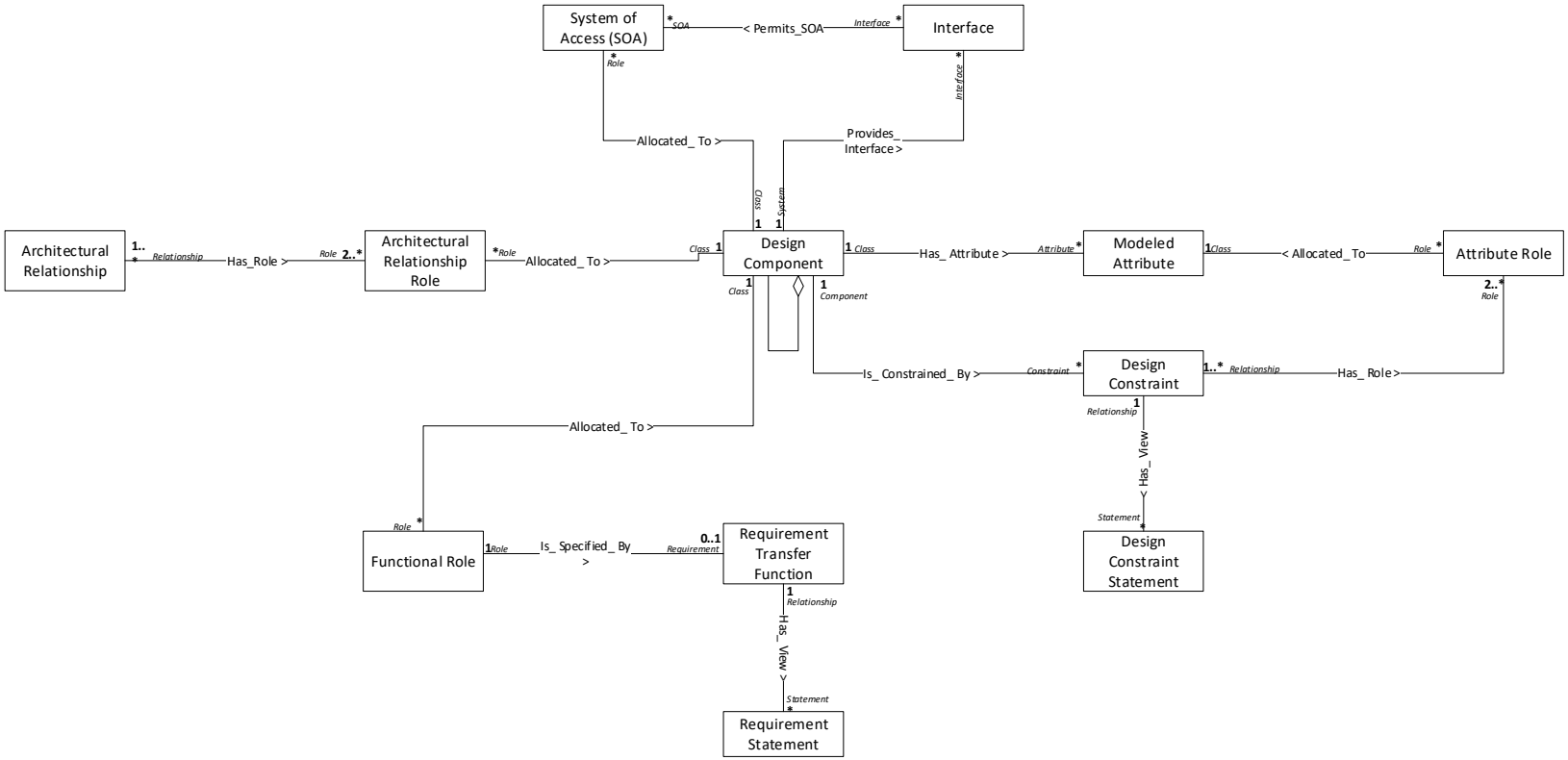


Figure 9: High Level Design View

2.10 Interface Context View

The Interface Context View depicts the classes and relationships relevant to modeling a system's interfaces and related classes and relationships.

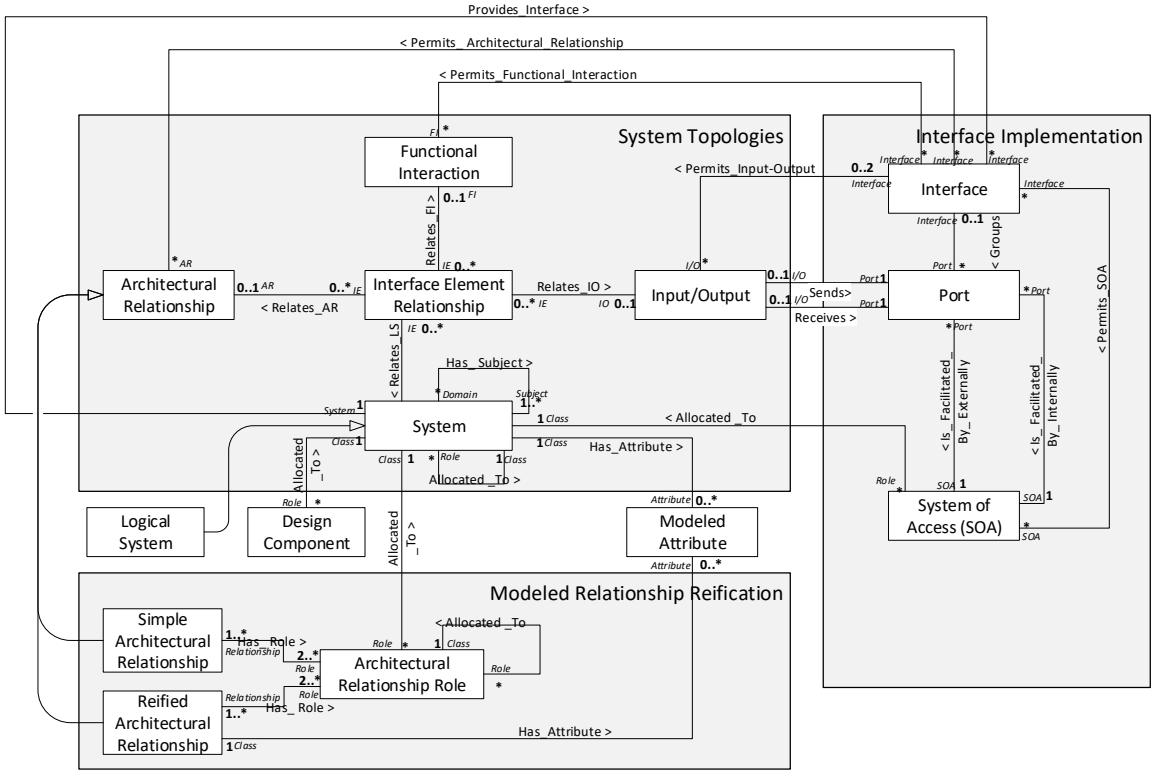


Figure 10: Interface Context View

2.11 Reified Relationship Views View

The metamodel defines abstract concepts such as a Reified Relationship, its Reified Relationship Roles, and Modeled Statements. These concepts are specialized to define Architectural Relationships, Functional Interactions, Requirements, Design Constraints, and Attribute Couplings. The following sections provide views defining each of these and are related in a class hierarchy manner in this figure. This view and the views shown do not impact the metamodel, but they do help relate each of the specialized relationship views to the Reified Relationship View.

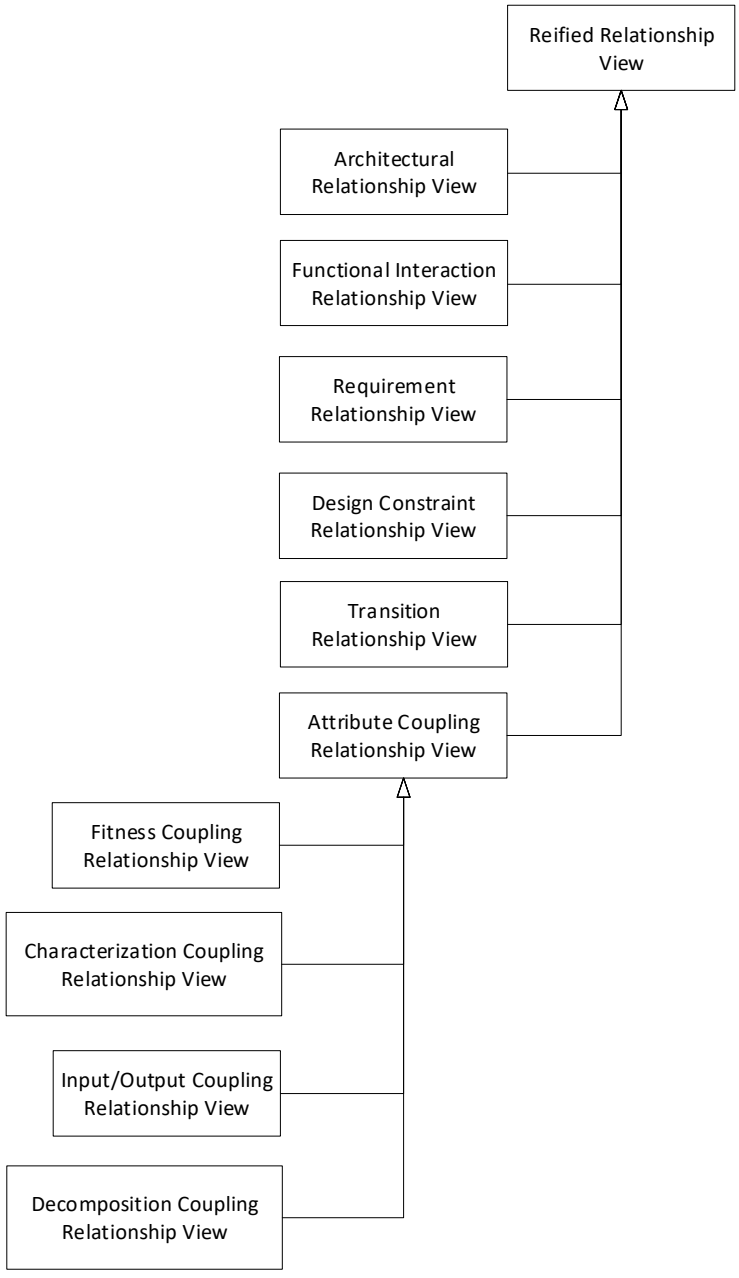


Figure 11: Reified Relationship Views View

2.12 Reified Relationship View

The Reified Relationship View defines the abstract concepts of Reified Relationships and Modeled Statements. This abstract portion of the model is specialized into other classes to create the views in the following sections.

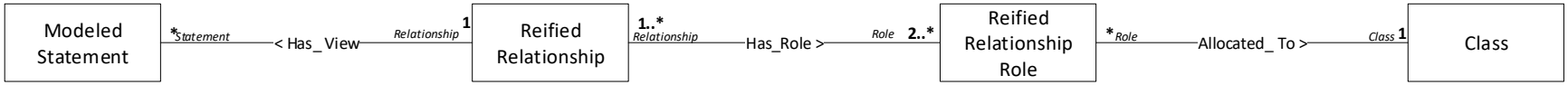


Figure 12: Reified Relationship View

2.13 Architectural Relationship View

This figure specializes the Reified Relationship View into classes that are used to model Architectural Relationships between modeled Systems. This view enables the High-Level Requirements (HLR) to be comprehensive yet much less detailed by summarizing specific Input/Outputs into Architectural Relationships.

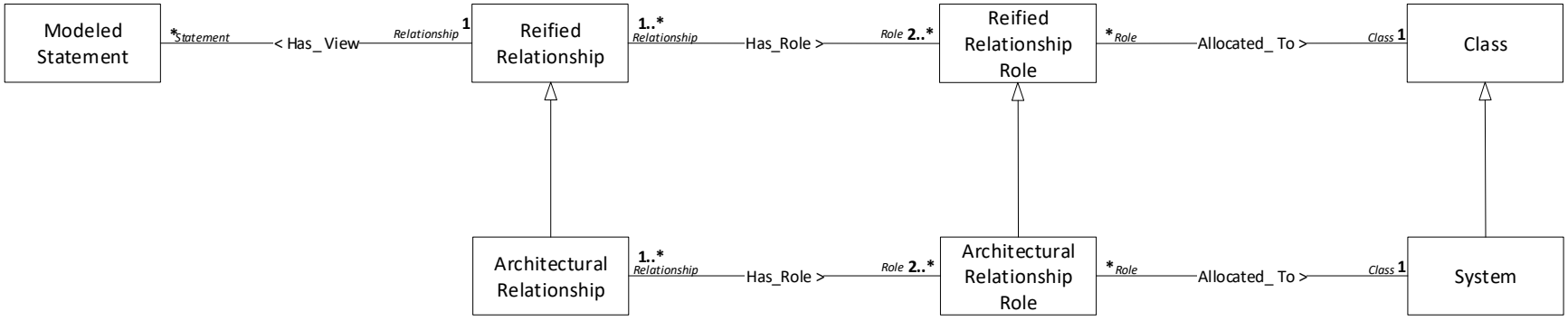


Figure 13: Architectural Relationship View

2.14 Functional Interaction View

The Functional Interaction View defines the Functional Interaction and its related classes as subclasses of the Reified Relationship View classes.

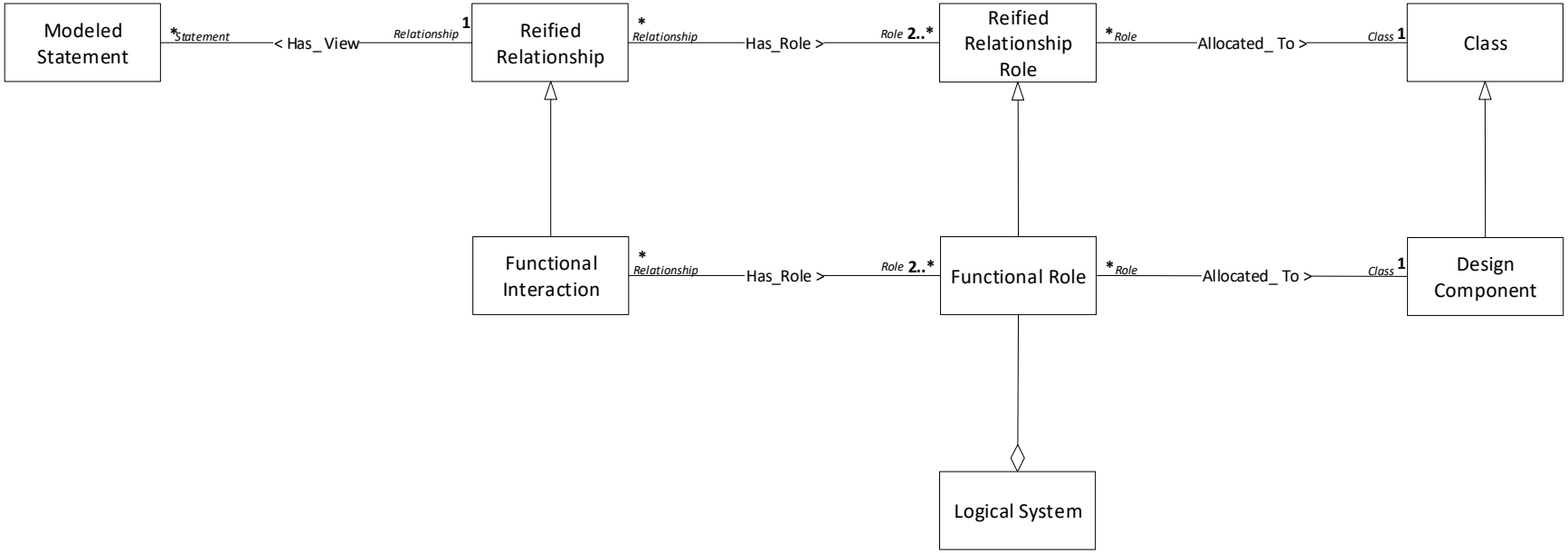


Figure 14: Functional Interaction View

2.15 Requirement Relationship View

This figure displays the Requirement Relationship View of the metamodel. A requirement is considered a relationship between a system's inputs and outputs and is modified by that system's attributes. A Requirement Statement, often a "shall" prose statement, describes the requirement relationship. Modeling requirements using a transfer function pattern directly links prose statements to the models and ensures testability of such statements.

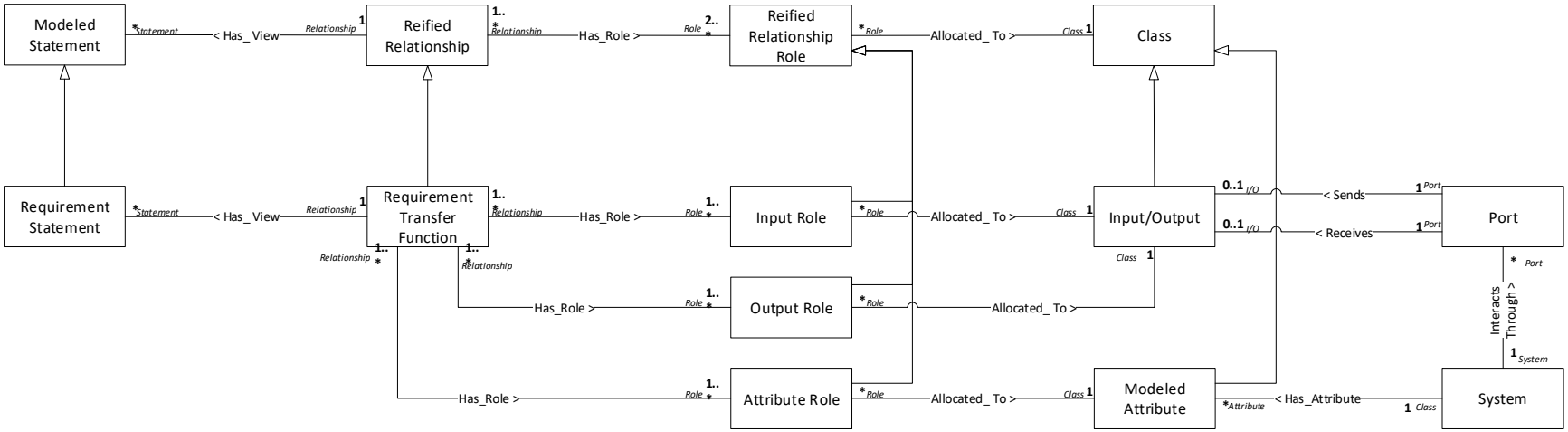


Figure 15: Requirement Relationship View

2.16 Design Constraint View

The Design Constraint View in this figure defines the Design Constraint and Design Constraint Statements as a specialization of the Reified Relationship pattern that modifies a System's Design Component Subsystem.

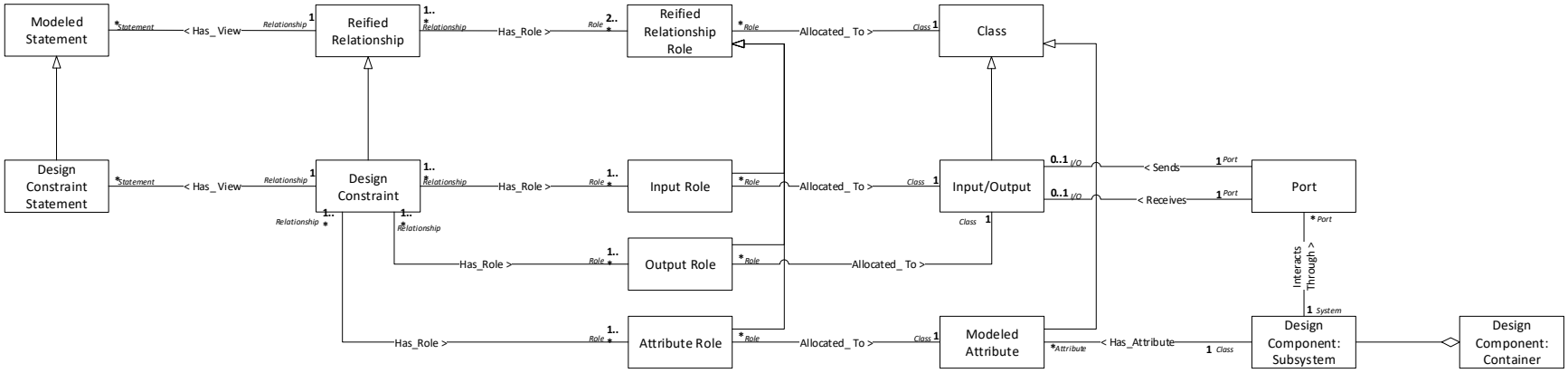


Figure 16: Design Constraint View

2.17 Transition Relationship View

The Transition Relationship View defines transitions as specializations of the reified relationship.

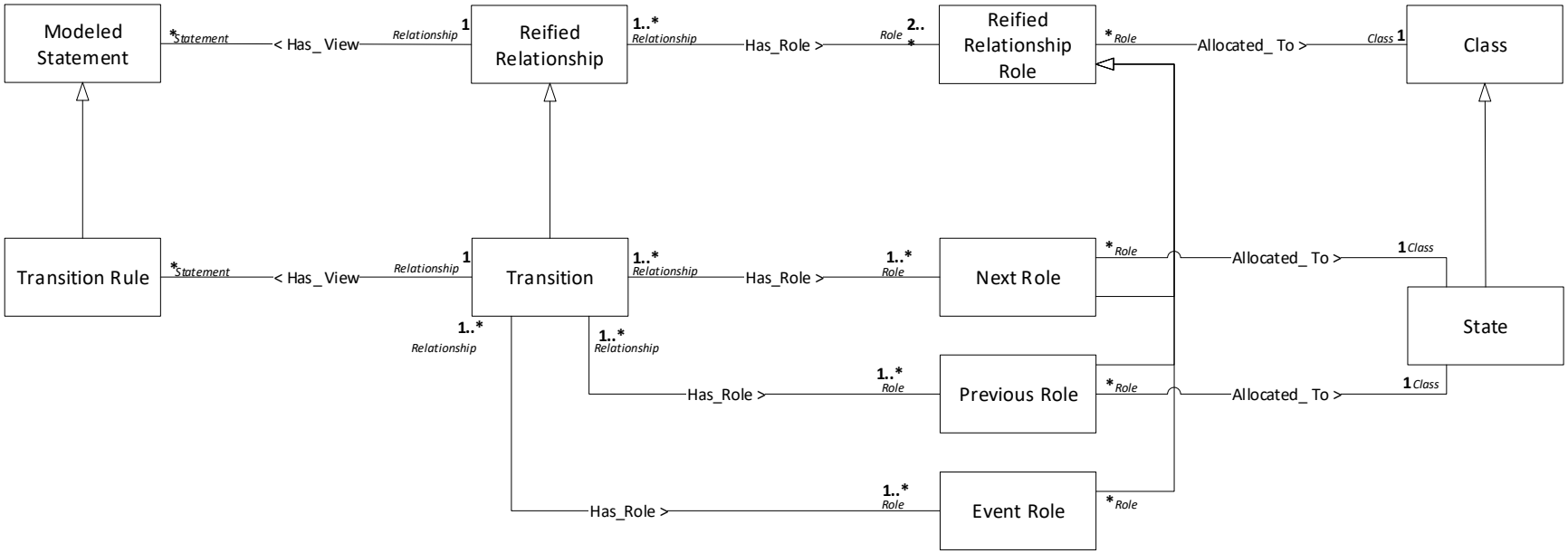


Figure 17: Transition Relationship View

2.18 Attribute Coupling View

The Reified Relationship View is specialized into a pattern that relates attributes in this figure. Attributes are coupled together with Attribute Coupling Maps as prose, mathematical equations, etc. to describe those relationships.

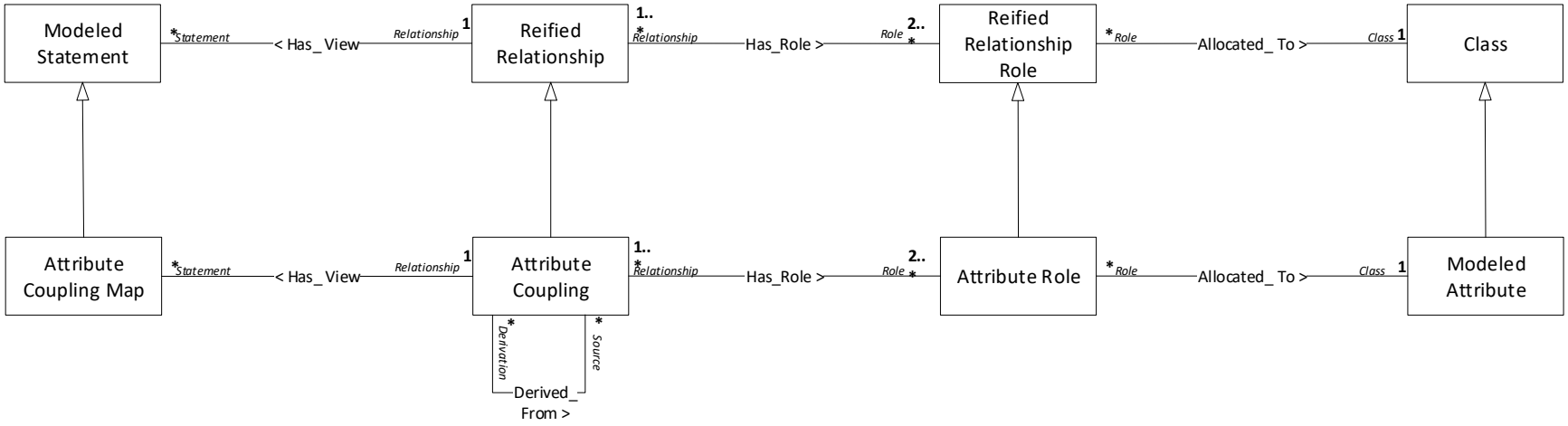


Figure 18: Attribute Coupling View

2.19 Fitness Coupling View

The Fitness Coupling View defines the Metamodel classes and relationships that link Feature Attributes (requirements in the Stakeholders' language) to Functional Role Attributes (requirements in the engineer's language). This view of the model is also often used to couple between Feature Attributes themselves.

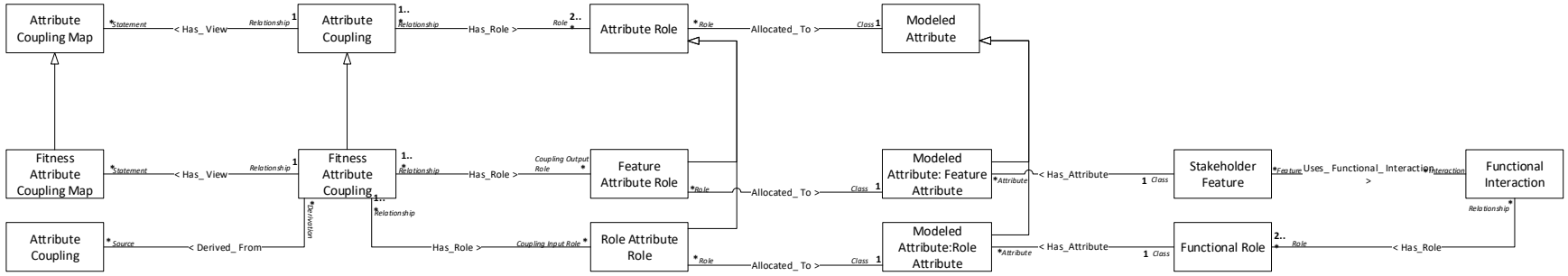


Figure 19: Fitness Coupling View

2.20 Characterization Coupling View

The Characterization Coupling View defines the Metamodel classes and relationships that link Functional Role Attributes to Design Component Attributes. This view of the model is also often used to couple between Design Component Attributes themselves.

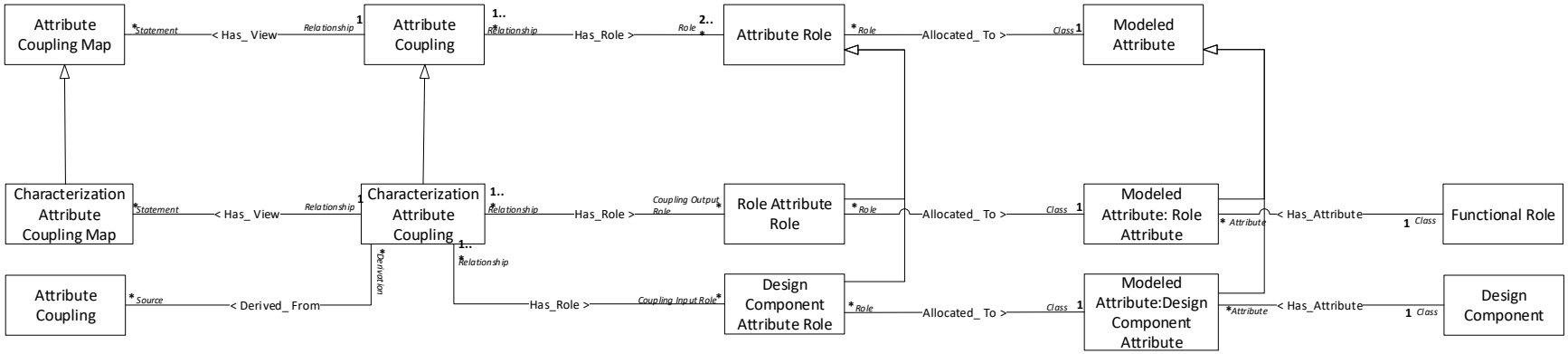


Figure 20: Characterization Coupling View

2.21 Decomposition Coupling View

The Decomposition Coupling View defines the Metamodel classes and relationships that link Black Box Functional Role Attributes to White Box Functional Role Attributes.

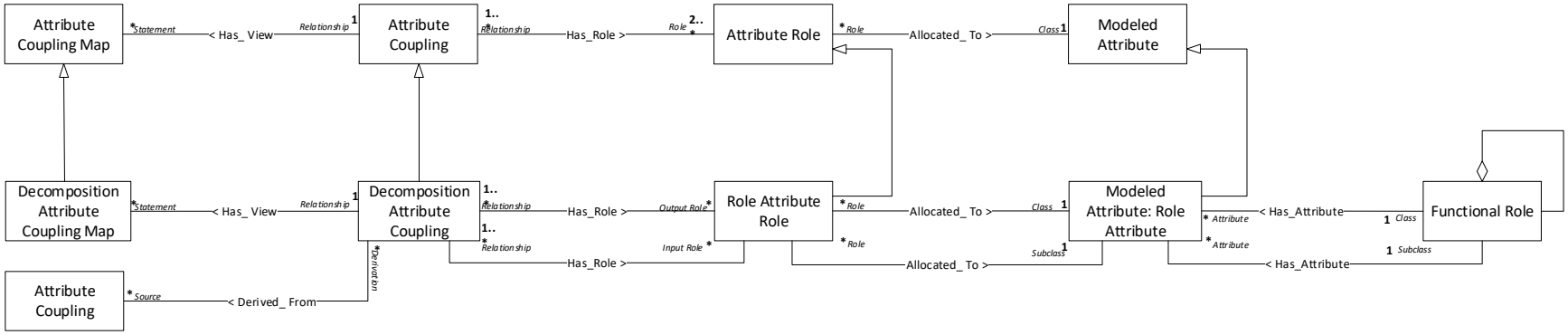


Figure 21: Decomposition Coupling View

2.22 Input/Output Coupling View

The Input/Output Coupling View defines the Metamodel classes and relationships that link Functional Role Attributes to Input/Output Attributes. This view of the model is also often used to couple between Input/Output Attributes themselves.

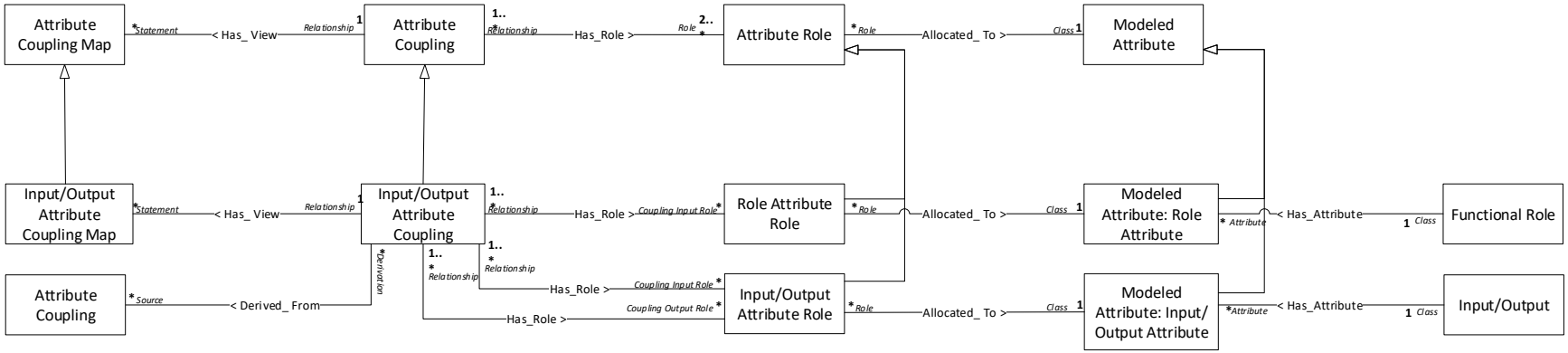


Figure 22: Input/Output Coupling View

2.23 Summary Pattern Configuration View

The S*Metamodel includes information supporting not only expression of a model of a single system, but also a model of a more general class of systems that are similar but not identical to each other. This includes the ability to re-use that model to represent different configured instances based on a common but configurable representation. Such a common, configurable model is called an S*Pattern, and can be used to rapidly create differently configured but similar S*Models. This Pattern-Based Systems Engineering (PBSE) situation is briefly summarized in Figure 1, further detailed by Figure 23, Table 1, and Table 2, and described by the PBSE references of Section 1.4.

The specialization of a general S*Pattern to represent a specific S*Model may be further constrained to an efficient form of specialization referred to as “configuration” in PBSE. In this case, the specialization process is limited to (1) the populating of classes and relationships (including their attributes) found in the general S*Pattern into a specialized S*Model, and (2) the setting of values in the S*Model for attributes populated from the S*Pattern. This configuration process means that the names and definitions of classes, relationships, and attributes from the S*Pattern survive into the S*Model, and the web of model relationships is determined by the S*Pattern, as are the model attributes.

This model population process can include creation of multiple instances of single entities found in the S*Pattern, thereby unfolding and specializing a compressed S*Pattern. When that occurs, more than one entity could have the same name, and such entities are differentiated from each other by an entity attribute called a “primary key” (PK) attribute. This in effect extends the name of the specialized entity to maintain uniqueness. These concepts of configuration population and PK values includes both classes and relationships, and allows a complex web of related model classes to unfold in a configured model. The pattern of those connective relationships is further governed by the configuration rules that establish the values of PK attributes, effectively identifying different entity instances and the connectivity between them.

The configuration rules which govern this unfolding of a compressed S*Pattern into a configured S*Model are inherent to and part of the S*Pattern--they are a part of the basic S*Metarelationships that connect the S*Metaclasses. Table 2 indicates which of the Metarelationships contains those configuration rules. The details of those configuration rules appear in Section 3.2, Metaclass Relationships, and Section 3.3.2, Specific Attributes and Configuration Rule Sets. (For a few reified relationships, configuration rules appear in Section 3.1, Metaclasses.) Configuration rules built into an S*Pattern allows the use of automated tooling to support (1) semi-automatic generation of S*Models that conform to the S*Pattern configuration rules, and (2) automated checking of other S*Models not sourced in that way, checking for their conformance to an S*Pattern.

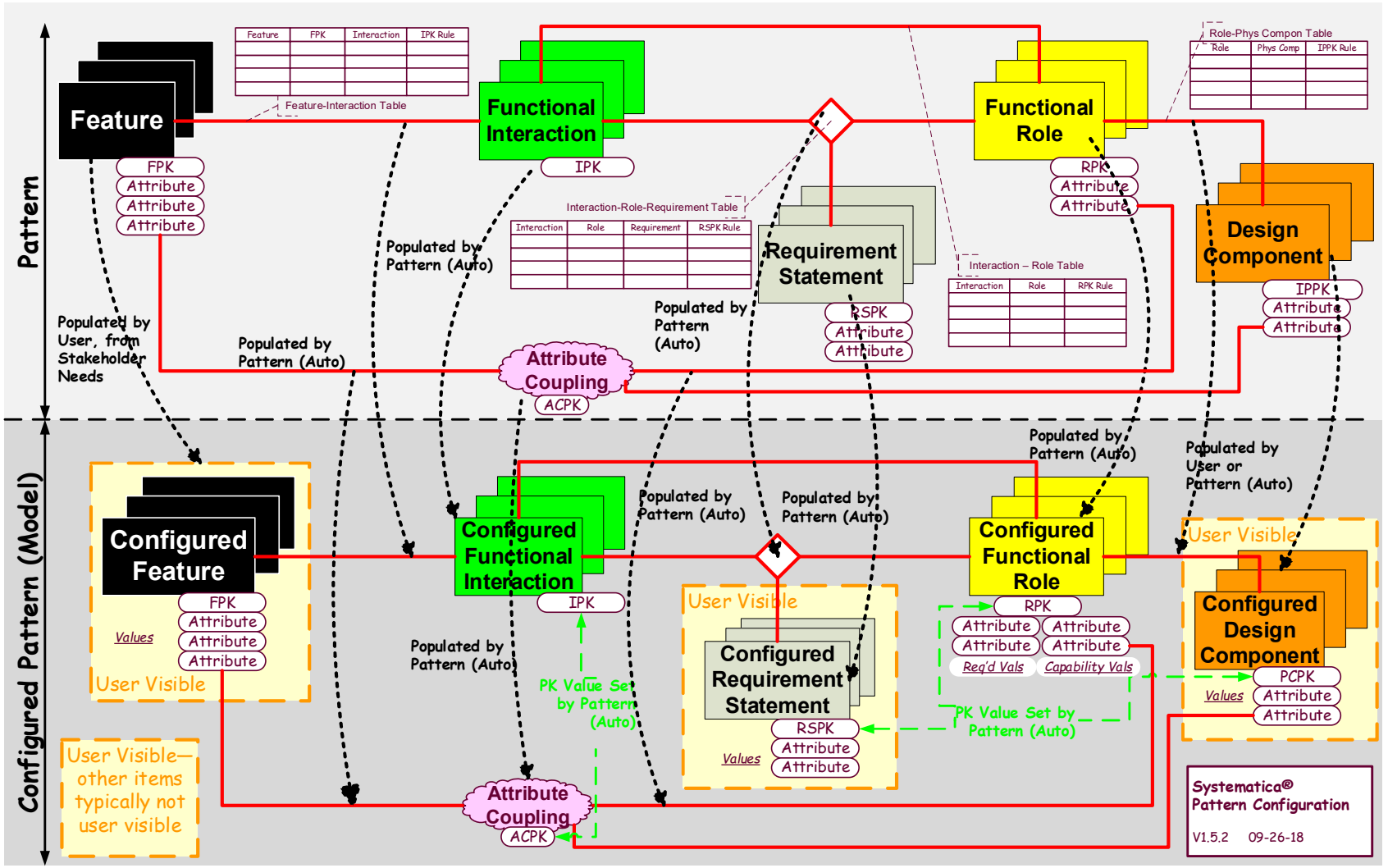


Figure 23: Summary Pattern Configuration View

	POPULATED METACLASSES ("THEN")																								
TRIGGERING METACLASSES ("IF")	Feature	Interaction	Role	Design Component	Requirement Statement	State	Event	Transition	Interface	Architectural Relationship	Input/Output	Port	System of Access	Failure Impact	Counter Requirement Statement	Failure Mode	Feature Attribute	Role Attribute	Design Component Attribute	Input/Output Attribute	Fitness Attribute Coupling	Decomposition Attribute Coupling	Characterization Attribute Coupling	IO Attribute Coupling
Stakeholder Input	■																								
Feature	■	■												■			■								
Interaction		■	■		■	■	■		■	■	■	■	■												
Role			■	■	■	■			■	■	■	■	■					■							
Design Component				■												■			■						
Requirement Statement					■										■										
State						■	■																		
Event							■	■																	
Transition								■																	
Interface									■																
Architectural Relationship										■															
Input/Output											■									■					
Port												■													
System of Access													■												
Failure Impact														■	■										
Counter Requirement Statement															■	■									
Failure Mode																■									
Feature Attribute																	■				■				
Role Attribute																		■				■	■		
Design Component Attribute																			■						
Input/Output Attribute																				■					■
Fitness Attribute Coupling																					■				
Decomposition Attribute Coupling																						■			
Characterization Attribute Coupling																							■		
IO Attribute Coupling																									■

Table 1: How Pattern Configuration Propagates—Driving Classes

Populated Metaclass	Metaclass(es) Driving Population	Metarelationship or Metaclass Carrying Configuration Rules	Figure Number Reference (Section 2)	Related Configuration Rules Set (See Section 3.3.3)
Interaction	Feature	Uses Functional Interaction	2	Features-Interactions
Role	Interaction	Has Role	2	Interactions-Roles
Design Component	Role	Allocated To	9	Roles-Design Components
Requirement Statement	Interaction + Role	Requirement Transfer Function	8	Interactions-Roles-Requirements
State	Interaction	Requires	7	Interactions-States
Interface	Interaction + Role	Interface Element	10	Interface Context
Architectural Relationship	Interaction + Role	Interface Element	10	Interface Context
Input/Output	Interaction + Role	Interface Element	10	Interface Context
Port	Interaction + Role	Interface Element	10	Interface Context
System of Access	Interaction + Role	Interface Element	10	Interface Context
Fitness Attribute Coupling	Feature Attribute	Feature Attribute Role	19	Attribute Coupling
Decomposition Attribute	Role Attribute	Role Attribute Role	21	Attribute Coupling
Characterization Attribute	Role Attribute	Role Attribute Role	20	Attribute Coupling
IO Attribute Coupling	IO Attribute	IO Attribute Role	22	Attribute Coupling
Counter Requirement	Requirement Statement	Replaces	24	Risk Analysis
Failure Mode	Design Component	Abnormal State Of	24	Risk Analysis
Feature Impact	Feature	Impacts Feature	24	Risk Analysis
Feature Attribute	Feature	Can Have Value	4	Pattern Feature Attribute Values
Role Attribute	Role	N/A	8	N/A
Input/Output Attribute	Input/Output	N/A	8	N/A
Design Component Attribute	Design Component	N/A	9	N/A
Event	Event Context Interaction	Provides Event Context	7	States-Transitions-Events
Transition	From State + To State + Event	Transition	7	States-Transitions-Events

Table 2: How Pattern Configuration Propagates—Location of Pattern Configuration Rules

2.24 Risk Analysis View

The Risk Analysis View depicts the classes of risk analysis and how they are related to other classes.

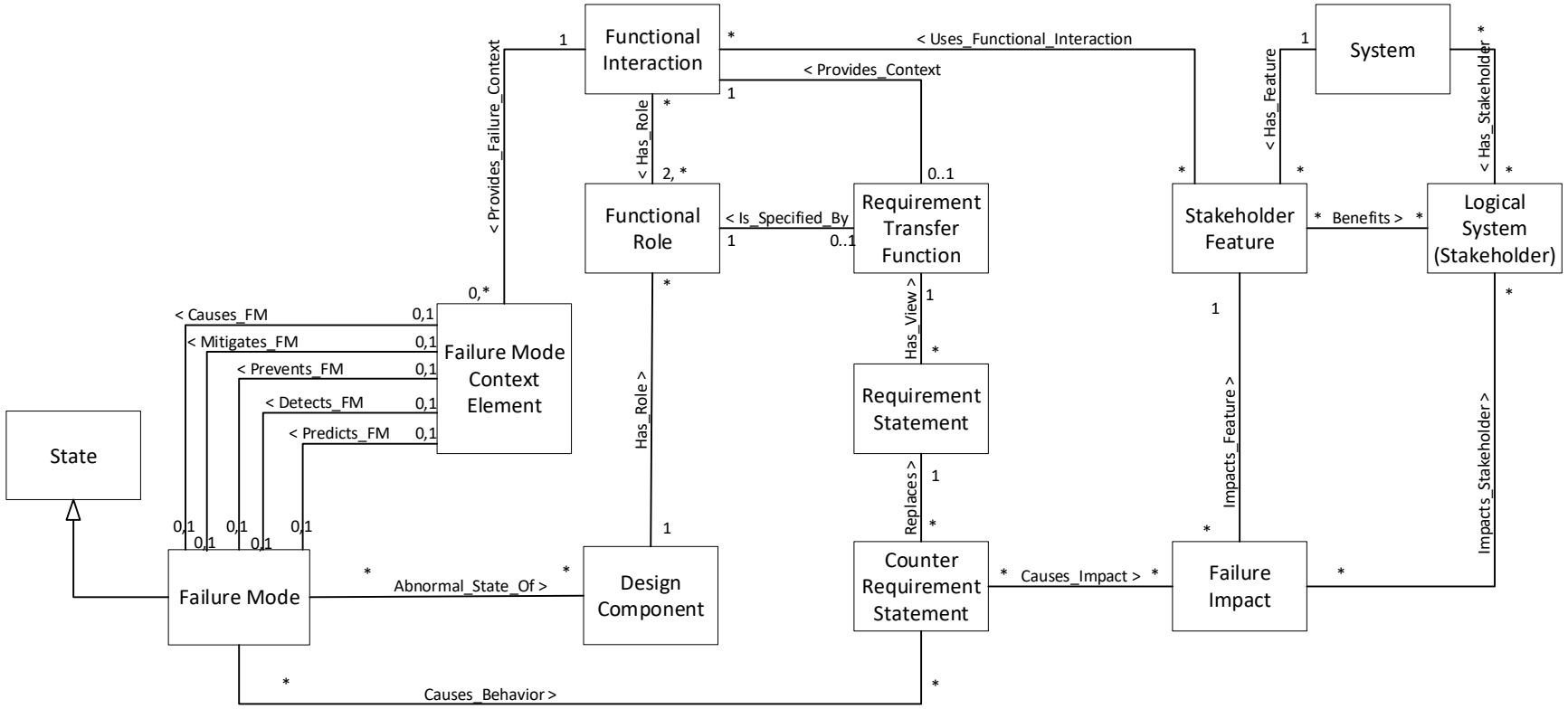


Figure 24: Risk Analysis View

3 Metamodel Definitions

This section defines the metaclasses, relationships, and attributes of the metaclasses shown in the views of the previous section.

3.1 Metaclasses

A metaclass models a particular system engineering concept. Classes (typically with noun names) are related to each other to form complete models of requirements or design using metaclass relationships (see next Section 3.2 Metaclass Relationships). They also have Class Attributes (parameters) to further tune the modeled concept of a class on an individual basis in a specific model. Certain relationships appear as "reified" relationships in this Section 3.1 Classes, instead of as Metaclasses in Section 3.2. This is typically to accommodate needs for extra or variable numbers of relationship roles. Such a reified relationship appears as a class connecting other classes that it relates, thereby serving as a relationship. See Section 3.3.1 for Common Attributes. See Section 3.3.2 for Specific Attributes including Configuration Rule Sets.

3.1.1 Allowed Value

An Allowed Value is an allowed (valid) value of a Feature's Attribute.

3.1.1.1 Relationships

- Can Have Value

3.1.1.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View

3.1.2 Architectural Relationship

An Architectural Relationship is a reified relationship that summarizes the architectural significance of a set of interactions between systems.

3.1.2.1 Relationships

- Has Role
- Is a Type of
- Permits Architectural Relationship

3.1.2.2 Metamodel View References

- Figure 2: Class Hierarchy View

- Figure 13: Architectural Relationship View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View
- Figure 10: Interface Context View

3.1.3 Architectural Relationship Role

An Architectural Relationship Role is a role defined within an Architectural Relationship that is played by a System.

3.1.3.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.3.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 13: Architectural Relationship View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 9: High Level Design View
- Figure 10: Interface Context View

3.1.4 Attribute Coupling

An Attribute Coupling is a reified relationship between two or more Attributes and one or more Attribute Coupling Maps that defines or constrains the value relationship between the Attributes.

3.1.4.1 Aliases

- Parametric Coupling

3.1.4.2 Relationships

- Derived From

- Has Role
- Has View
- Is a Type of

3.1.4.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 14: Functional Interaction View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View

3.1.5 Attribute Coupling Map

An Attribute Coupling Map is a statement in prose, mathematical equation, or other form that describes the value relationship between two or more Attributes.

3.1.5.1 Aliases

- Parametric Coupling

3.1.5.2 Relationships

- Has View
- Is a Type of

3.1.5.3 Specific Attributes

- Reference

3.1.5.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View

- Figure 22: Input/Output Coupling View

3.1.6 Attribute Role

An Attribute Role is a Reified Relationship Role in a Reified Relationship that specifically references an Attribute.

3.1.6.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.6.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.1.7 Characterization Attribute Coupling

A Characterization Attribute Coupling is a reified relationship between Attributes of a Functional Role and Attributes of a Design Component allocated that role. One or more Characterization Attribute Coupling Maps can define or constrain the value relationships between the Attributes.

3.1.7.1 Aliases

- B Matrix Coupling
- Role-Design Component Coupling
- Design Coupling

- Parametric Coupling

3.1.7.2 Relationships

- Derived From
- Has Role
- Has View
- Is a Type of

3.1.7.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 20: Characterization Coupling View
- Figure 9: High Level Design View

3.1.8 Characterization Attribute Coupling Map

A Characterization Attribute Coupling Map is a statement in prose, mathematical equation, or other form that describes the value relationship between Attributes of Functional Roles and Design Components.

3.1.8.1 Aliases

- B Matrix Coupling Map
- Role-Design Component Coupling Map
- Design Coupling Map
- Parametric Coupling

3.1.8.2 Relationships

- Has View
- Is a Type of

3.1.8.3 Specific Attributes

- Reference

3.1.8.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 20: Characterization Coupling View

- Figure 9: High Level Design View

3.1.9 Class

Class is the most abstract metaclass; it is the root of the class hierarchy tree of all the metaclasses as seen in Figure 2. A class is a set of things that are considered “similar” to each other by virtue of their membership in that class.

3.1.9.1 Aliases

- Entity

3.1.9.2 Relationships

- Appears In
- Allocated To
- Contains
- Derived From
- Has Attribute
- Has Previous
- Has Issue
- Is a Type of

3.1.9.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 12: Reified Relationship View
- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View

3.1.10 Counter Requirement Statement

A Counter Requirement Statement is the counter to a requirement statement. In effect, it replaces the “Shall” of the requirement statement with “Shall not” which describes the

negative or anomalous behavior occurring during failure to meet Requirements. Note that a given Requirement may have more than one Counter Requirement, as it may be violated in more than one way.

3.1.10.1 Relationships

- Causes Behavior
- Causes Impact
- Replaces

3.1.10.2 Metamodel View References

- Figure 24: Risk Analysis View

3.1.11 Decomposition Attribute Coupling

A Decomposition Attribute Coupling is a reified relationship between Attributes of Functional Roles at different levels of decomposition. One or more Decomposition Attribute Coupling Maps can define or constrain the value relationships between the Attributes.

3.1.11.1 Aliases

- Role-Role Coupling
- Parametric Coupling

3.1.11.2 Relationships

- Derived From
- Has Role
- Has View
- Is a Type of

3.1.11.3 Specific Attributes

- Reference

3.1.11.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 21: Decomposition Coupling View

3.1.12 Decomposition Attribute Coupling Map

A Decomposition Attribute Coupling Map is a statement in prose, mathematical equation, or other form that describes the value relationship between Attributes of Functional Roles.

3.1.12.1 Aliases

- Role-Role Coupling Map

3.1.12.2 Relationships

- Has View
- Is a Type of

3.1.12.3 Specific Attributes

- Reference

3.1.12.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 21: Decomposition Coupling View

3.1.13 Design Component

A Design Component is a System defined based upon its identity or composition, but not its behavior. Design Components may be given proper names, such as names of commercial products, materials, chemical elements or compounds, part numbers, corporate systems, people, organizations, buildings, etc. Design Components fulfill the Functional Roles (Logical Systems) allocated to them through an Allocation Decision.

3.1.13.1 Aliases

- Physical System

3.1.13.2 Relationships

- Allocated To
- Contains
- Has Attribute
- Has Subject
- Is a Type of
- Is Constrained By
- Provides Interface

3.1.13.3 Metamodel View References

- Figure 1: Summary Metamodel

- Figure 2: Class Hierarchy View
- Figure 14: Functional Interaction View
- Figure 16: Design Constraint View
- Figure 20: Characterization Coupling View
- Figure 6: Logical Architecture View
- Figure 9: High Level Design View

3.1.14 Design Component Attribute

A Design Component Attribute is a Modeled Attribute of a Design Component.

3.1.14.1 Aliases

- Physical System Attribute

3.1.14.2 Relationships

- Has Value

3.1.14.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View

3.1.15 Design Component Attribute Role

A Design Component Attribute Role is a Reified Relationship Role in a Reified Relationship that specifically references an Attribute of a Design Component.

3.1.15.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.15.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 20: Characterization Coupling View

3.1.16 Design Constraint

Design Constraint is a relationship that limits a subsystem's or components' attribute values or behavior with respect to its inputs and outputs and states. A Design Constraint is described by a Design Constraint Statement.

3.1.16.1 Relationships

- Has Role
- Has View
- Is a Type of
- Is Constrained By

3.1.16.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 16: Design Constraint View
- Figure 9: High Level Design View

3.1.17 Design Constraint Statement

A Design Constraint Statement is a description in prose, mathematical, or other form that expresses a Design Constraint.

3.1.17.1 Relationships

- Has View
- Is a Type of

3.1.17.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 16: Design Constraint View
- Figure 9: High Level Design View

3.1.18 Domain

A Domain is an environmental system. The components and relationships of this system establish an overall environment (domain) for a subject system. A domain establishes the domain knowledge relevant to a subject system. A system domain may be as large as the subject system's entire life cycle environment, or a smaller domain, such as the operational, production, sustainment, distribution, or other specialized domain of a subject system.

3.1.18.1 Aliases

- System Context
- Context of Use
- System Environment

3.1.18.2 Relationships

- Appears In
- Has Subject
- Is a Type of

3.1.18.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View

3.1.19 Domain System

A Domain System is a subsystem in a Domain whose interactions impact the characteristics of that Domain.

3.1.19.1 Relationships

- Is a Type of

3.1.19.2 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.20 Event

An Event is a subclass of an Information Input/Output or Value that describes an occurrence that triggers a transition from one modeled state to another. Such information is not always an engineered signal, and in some cases may be a condition or state, including an attribute value condition. Nevertheless, as such it is still information, whether instrumented or not.

3.1.20.1 Relationships

- Is a Type of
- Is Triggered By

3.1.20.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 7: State Analysis View

3.1.21 Failure Impact

A Failure Impact is the result of a failure that has impact on a Stakeholder through the inability of the system to perform to fully deliver the associated Feature capability. If a candidate failure cannot be traced to an impact on a Feature/Stakeholder, then it is apparently not a failure worth modeling a Failure Impact.

3.1.21.1 Relationships

- Causes Impact
- Impacts Feature
- Impacts Stakeholder

3.1.21.2 Metamodel View References

- Figure 24: Risk Analysis View

3.1.22 Failure Mode

A Failure Mode is an abnormal state of a design component that can be triggered by one or more causes and will result in abnormal behavior in the performance of some allocated role, such that a requirement is violated.

3.1.22.1 Relationships

- Abnormal State Of
- Causes Behavior
- Causes Mode

3.1.22.2 Metamodel View References

- Figure 24: Risk Analysis View

3.1.23 Feature Attribute

A Feature Attribute is a Modeled Attribute of a Stakeholder Feature.

3.1.23.1 Relationships

- Has Value

3.1.23.2 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View

3.1.24 Feature Attribute Role

A Feature Attribute Role is a Reified Relationship Role in a Requirements Relationship that specifically references an Attribute of a Stakeholder Feature.

3.1.24.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.24.2 Configuration Attributes

- Attribute Coupling Population Rule

3.1.24.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 19: Fitness Coupling View

3.1.25 Feature Primary Key Attribute

A Feature Primary Key Attribute is a Modeled Attribute of a Stakeholder Feature that is a type of Feature Attribute.

3.1.25.1 Relationships

- Has Value

3.1.25.2 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View

3.1.26 Fitness Attribute Coupling

A Fitness Attribute Coupling is a reified relationship between Attributes of Stakeholder Features and Attributes of Functional Roles. One or more Fitness Attribute Coupling Maps can define or constrain the value relationships between the Attributes.

3.1.26.1 Aliases

- A Matrix Coupling
- Feature-Role Coupling
- Requirements Coupling
- Parametric Coupling

3.1.26.2 Relationships

- Derived From
- Has Role
- Has View
- Is a Type of

3.1.26.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 19: Fitness Coupling View

3.1.27 Fitness Attribute Coupling Map

A Fitness Attribute Coupling Map is a statement in prose, mathematical equation, or other form that describes the value relationship between Attributes of Features and Functional Roles.

3.1.27.1 Aliases

- A Matrix Coupling Map
- Feature-Role Coupling Map
- Requirements Coupling Map
- Parametric Coupling

3.1.27.2 Relationships

- Has View
- Is a Type of

3.1.27.3 Specific Attributes

- Reference

3.1.27.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 19: Fitness Coupling View

3.1.28 Functional Interaction

A Functional Interaction is an interaction of two or more Systems, Subsystems, or System Components. Interaction means the exchange of Input-Outputs (typically force, energy, material flow or information) whereby one system affects the State (see State) of another system. Interactions are the phenomena-grounded basis of the theoretical foundations of the physical sciences and engineering disciplines. All behavior occurs in the context of interactions. The behavior of each interacting component is determined by its state, and that state can in turn be changed by the interactions.

3.1.28.1 Aliases

- Function (Deprecated)
- Interaction

3.1.28.2 Relationships

- Has Role
- Is a Type of
- Permits Functional Interaction
- Provides Context
- Requires
- Uses Functional Interaction

3.1.28.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 14: Functional Interaction View
- Figure 19: Fitness Coupling View
- Figure 5: Domain Analysis View
- Figure 7: State Analysis View
- Figure 8: Detail Requirements View

- Figure 24: Risk Analysis View

3.1.29 Functional Role

A Functional Role is the behavior displayed by one of the interacting entities during a Functional Interaction. Because it is entirely described as behavior, a Functional Role is a Logical System. A Functional Role may eventually be allocated to a Design Component to perform that behavior, but the Functional Role is viewed as meaningful whether or not so allocated.

3.1.29.1 Aliases

- Function
- Logical System
- Role

3.1.29.2 Relationships

- Allocated To
- Contains
- Has Attribute
- Has Role
- Is a Type of
- Is Specified By

3.1.29.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 14: Functional Interaction View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View
- Figure 24: Risk Analysis View

3.1.30 Information Input/Output

An Information Input/Output is a subclass of Input/Output that represents symbolic or other information exchanged between interacting systems. Such information is always "about" something.

3.1.30.1 Aliases

- Information View (Deprecated)

3.1.30.2 Relationships

- Is a Type of

3.1.30.3 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.31 Input/Output

An Input/Output is that which is exchanged between interacting systems. Most Input-Outputs of interest are forces, energy, materials, or information.

3.1.31.1 Aliases

- I/O
- Input
- Output
- View (Deprecated)

3.1.31.2 Relationships

- Allocated To
- Is a Type of
- Permits Input/Output
- Receives
- Sends

3.1.31.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View

- Figure 16: Design Constraint View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View

3.1.32 I/O Attribute

An I/O Attribute is a Modeled Attribute of an Input/Output.

3.1.32.1 Relationships

- Has Value

3.1.32.2 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View

3.1.33 I/O Attribute Role

An I/O Attribute Role is a Reified Relationship Role in an I/O Coupling that specifically references an Attribute of an Input/Output.

3.1.33.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.33.2 Configuration Attributes

- Attribute Coupling Population Rule

3.1.33.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 22: Input/Output Coupling View

3.1.34 I/O Attribute Coupling

An I/O Attribute Coupling is a reified relationship between values of Functional Role Attributes and Input/Output Attributes.

3.1.34.1 Aliases

- Parametric Coupling

3.1.34.2 Relationships

- Derived From
- Has Role
- Has View
- Is a Type of

3.1.34.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 22: Input/Output Coupling View

3.1.35 I/O Attribute Coupling Map

An I/O Coupling Map is a statement in prose, mathematical equation, or other form that describes the value relationship between Attributes of Input/Outputs and Functional Roles.

3.1.35.1 Aliases

- Parametric Coupling

3.1.35.2 Relationships

- Has View
- Is a Type of

3.1.35.3 Specific Attributes

- Reference

3.1.35.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 22: Input/Output Coupling View

3.1.36 Input Role

An Input Role is a Reified Relationship Role in a Reified Relationship that specifically references an Input/Output that is being transformed into another Input/Output or state change.

3.1.36.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.36.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 8: Detail Requirements View

3.1.37 Interface

An Interface is an association of a System (which owns, provides, displays, or exposes the Interface), one or more Input/Outputs (which flow through the Interface), one or more Functional Interactions (which describe behavior at the Interface), and a System of Access (SOA), which is the medium enabling or mediating the interaction between systems or transporting their exchanged Input-Outputs.

3.1.37.1 Relationships

- Groups
- Is a Type of
- Permits Architectural Relationship
- Permits Functional Interaction
- Permits Input/Output
- Permits SOA
- Provides Interface

3.1.37.2 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View

- Figure 9: High Level Design View

3.1.38 Interface Element Relationship

The Interface Element Relationship is a reified 4-way relationship among Architectural Relationships, Functional Interactions, Input/Outputs, and Logical Systems. The “elemental atoms” from which an Interface is built up are formed by a collection of Interface Element Relationships and their connections to Ports and Systems of Access.

3.1.38.1 Relationships

- Relates AR
- Relates FI
- Relates IO
- Relates LS

3.1.38.2 Configuration Attributes

- Interface Primary Key Value Rule
- IO Primary Key Value Rule
- Port Primary Key Value Rule
- SOA Primary Key Value Rule
- AR Primary Key Value Rule
- AR Role Primary Key Value Rule
- IO Direction
- SOA Internal/External
- Port Type
- AR Internal/External
- AR Complexity

3.1.38.3 Metamodel View References

- Figure 10: Interface Context View

3.1.39 Failure Mode Context Element

A Failure Mode Context Element is a reified relationship linking Functional Roles, Interactions, and Failure Modes in a Risk Analysis model.

3.1.39.1 Relationships

- Causes Failure
- Causes Mode
- Plays Causal Role

3.1.39.2 Metamodel View References

- Figure 24: Risk Analysis View

3.1.40 Logical System

A Logical System is a system defined solely by its (required or actual) functionality or behavior as "seen" by external systems interacting with it, and not based upon how it achieves that functionality internally or its identity or composition. Logical systems are typically named and defined in a behavioral sense without reference to their physical composition, unless (in some cases) this is a part of the external behavior description. Accordingly, all Functional Roles are Logical Systems.

3.1.40.1 Aliases

- Function
- Functional Role
- Logical Architecture Component (LAC)

3.1.40.2 Relationships

- Advocates
- Allocated To
- Benefits
- Contains
- Has Advocate
- Has Stakeholder
- Has Subject
- Is a Type of
- Perceives
- Provides Interface

3.1.40.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 14: Functional Interaction View
- Figure 6: Logical Architecture View

3.1.41 Logical System Attribute

A Logical System Attribute is a Modeled Attribute of a Logical System.

3.1.41.1 Relationships

- Has Value

3.1.41.2 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View

3.1.42 Modeled Attribute

A Modeled Attribute is a modeled property or characteristic of any of the metaclasses, which might take on different attribute values to further describe (parameterize) the various instances of that class and how they may vary. An attribute may belong to any metaclass, including another Attribute.

3.1.42.1 Aliases

- Attribute
- Property
- Parameter
- Variable

3.1.42.2 Relationships

- Allocated To
- Has Attribute
- Has Value
- Is a Type of

3.1.42.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 4: Feature Framework View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.1.43 Reified Relationship

A Reified Relationship is a statement about several classes that may be true or false. If true, the classes are said to be in that relationship with each other. This class has been reified from actual relationships to allow for clearer modeling.

3.1.43.1 Relationships

- Has Role
- Has View
- Is a Type of

3.1.43.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Reified Relationship View
- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View

- Figure 18: Attribute Coupling View

3.1.44 Reified Relationship Role

A Reified Relationship Role is the part a class plays when being referred to in a Reified Relationship.

3.1.44.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.44.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Reified Relationship View
- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View

3.1.45 Modeled Statement

A Modeled Statement is a prose statement, mathematical equation, or other description of another class, typically a Reified Relationship.

3.1.45.1 Aliases

- Statement

3.1.45.2 Relationships

- Has View
- Is a Type of

3.1.45.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Reified Relationship View

- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View

3.1.46 Output Role

An Output Role is a Reified Relationship Role in a Reified Relationship that specifically references an Input/Output that is being transformed from another Input/Output.

3.1.46.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.46.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 8: Detail Requirements View

3.1.47 Physical Input/Output

A Physical Input/Output is a subclass of Input/Output that represents a physical quantity like energy or mass exchanged between interacting Systems.

3.1.47.1 Aliases

- Physical View (Deprecated)

3.1.47.2 Relationships

- Is a Type of

3.1.47.3 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.48 Port

A Port is the coincidence of an Input/Output and System border. A Port is associated with a received or sent Input/Output, an internal or external System of Access (SOA), internal or external Architectural Relationships, and one or more Functional Interactions.

3.1.48.1 Relationships

- Groups
- Is a Type of
- Is Facilitated By Externally
- Is Facilitated By Internally
- Receives
- Sends

3.1.48.2 Specific Attributes

- Port Type

3.1.48.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View

3.1.49 Requirement Statement

A behavioral description, in prose, mathematical, or other form, relating a System's Inputs, Outputs, and Attributes, against which a System will be verified.

3.1.49.1 Aliases

- "Shall" Statement

3.1.49.2 Relationships

- Has View
- Is a Type of

3.1.49.3 Specific Attributes

- Reference

3.1.49.4 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 15: Requirement Relationship View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View
- Figure 24: Risk Analysis View

3.1.50 Requirement Transfer Function

A Requirement Transfer Function is a reified relationship that limits a System's attribute values or behavior with respect to its inputs and outputs. A Requirement Transfer Function is described by a Requirement Statement.

3.1.50.1 Aliases

- Requirement Relationship

3.1.50.2 Relationships

- Has Role
- Has View
- Is a Type of
- Is Specified By
- Provides Context

3.1.50.3 Configuration Attributes

- Requirement Population Rule-Interaction
- Requirement Population Rule-Role
- Requirement Primary Key Value Rule

3.1.50.4 Metamodel View References

- Figure 2: Class Hierarchy View

- Figure 15: Requirement Relationship View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View
- Figure 24: Risk Analysis View

3.1.51 Role Attribute Role

A Role Attribute Role is a Reified Relationship Role in a Fitness, Decomposition, Characterization, or I/O Coupling that specifically references an Attribute of a Functional Role.

3.1.51.1 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.51.2 Configuration Attributes

- Attribute Coupling Population Rule

3.1.51.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View

3.1.52 State

A State is the value of a state variable describing some changing or changeable condition, characteristic, or parameter of a system. Some state variables can take on a continuum of values, and others are constrained to a finite list of possible values. In the latter case, a finite state model enumerates those States. In the finite state case, each State persists for a period of time. In all cases, the state of a System determines future behavior in which Functional Interactions are to be performed, entered, and exited based upon events. The finite States of an environmental System of a subject system are use cases for the subject system. During a use case, the subject system is required or expected to perform certain functions, interacting with the environmental system.

3.1.52.1 Aliases

- Mode
- Situation
- State Variable Value
- Use Case (often includes required Functional Interactions)

3.1.52.2 Relationships

- Has State
- Is a Type of
- Requires
- Transitions From
- Transitions To

3.1.52.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 7: State Analysis View

3.1.53 Stakeholder Feature

A Stakeholder Feature is a collection of Functional Interactions having stakeholder value implications. Features are used to summarize product functionality in value sets or service sets to a customer or other stakeholder. Economics, quality, performance, risk, or other measures of effectiveness are often associated with Features. The total Feature set of a system of interest establishes the “trade space” in which various issues are traded off against or compared to each other, as to the relative stakeholder appeal, score, or likelihood of selection. In addition to the value-laden concepts, the same Features also represent risk—all risk is risk to Features (see Feature Impact). For system families, product line engineering (PLE), and configurable platforms or patterns, Features are the primary point at which stakeholder configuration choices are expressed, thereafter driving all other points of variation within a system model.

3.1.53.1 Aliases

- Service
- Feature
- Capability

3.1.53.2 Relationships

- Benefits
- Has Attribute
- Has Feature
- Is a Type of
- Satisfies
- Uses Functional Interaction

3.1.53.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 19: Fitness Coupling View
- Figure 24: Risk Analysis View

3.1.54 Stakeholder Requirement

A Stakeholder Requirement is a statement (either in formal or informal language) that implies formal requirements or design constraints upon a system. Once analyzed, a validated Stakeholder Requirement becomes an originating source for other, more formal metaclasses (e.g., Stakeholder Features) describing that system.

3.1.54.1 Aliases

- Need
- Informal Need
- Stakeholder Need

3.1.54.2 Relationships

- Advocates
- Is a Type of
- Perceives
- Satisfies

3.1.54.3 Specific Attributes

- Date Submitted
- Due Date
- Originator
- Priority
- Reference
- Request Type
- Source

3.1.54.4 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View

3.1.55 System

A System is a collection of interacting components. By “interact” we mean the components exchange input-outputs (typically energy, force, material, or information) that change the state of the components. The components transform inputs into outputs, depending upon the state of the components. A component can itself be a System, called a sub-system.

3.1.55.1 Aliases

- Actor
- Component
- Subject System
- Subsystem
- System of Interest

3.1.55.2 Relationships

- Allocated To
- Has Attribute
- Has Feature
- Has Stakeholder
- Has State

- Has Subject
- Is a Type of
- Provides Interface

3.1.55.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 13: Architectural Relationship View
- Figure 15: Requirement Relationship View
- Figure 5: Domain Analysis View
- Figure 7: State Analysis View

3.1.56 System of Access (SOA)

A System of Access (SOA) is the intermediary system through which two or more other systems are able to interact (exchange input-outputs to impact each other's states).

3.1.56.1 Aliases

- Medium
- Network
- Transport Mechanism
- SOA
- SOAC
- System of Access Component

3.1.56.2 Relationships

- Allocated To
- Has Role
- Is a Type of
- Is Facilitated By Externally
- Is Facilitated By Internally

- Permits SOA

3.1.56.3 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.1.57 Transition

A Transition is the instantaneous switch (change of state) from one State to another State that has been caused, or triggered, by some Event. A transition that is not deemed to be instantaneous can be modeled using a transitional state having persistent life for some period, which is entered instantaneously and exited instantaneously.

3.1.57.1 Relationships

- Is a Type of
- Is Triggered By
- Transitions From
- Transitions To

3.1.57.2 Configuration Attributes

- From State PK Matching Rule
- To State PK Matching Rule
- Transition PK Value Rule
- Transition Type

3.1.57.3 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 7: State Analysis View
- Figure 17: Transition Relationship View

3.1.58 Value

A Value is the value of a Class's Attribute.

3.1.58.1 Relationships

- Has Value
- Is a Type of

3.1.58.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 4: Feature Framework View

3.2 Metaclass Relationships

Metaclass relationships (typically with verb form names) semantically link metaclasses together to create statements about system required behavior, design, or other holistic or linking aspects of interest to system engineering or modeling. Each such relationship has roles that describe a certain concept which the related classes must fill in order to complete the semantic statement. A few such relationships in the S*Metamodel have been reified and therefore appear as classes in Section 3.1 instead of this section. See Section 3.3.1 for Common Attributes. See Section 3.3.2 for Specific Attributes including Configuration Rule Sets.

3.2.1 Abnormal State Of

The Abnormal State Of relationship links Design Components with Failure Modes.

3.2.1.1 Roles

- Component: The role played by a Design Component when it enters an abnormal state. This role's cardinality is Many.
- Caused Mode: The role played by a Failure Mode. This role's cardinality is Many.

3.2.1.2 Configuration Attributes

- Failure Mode Population Rule
- Failure Mode Primary Key Value Rule

3.2.1.3 Metamodel View References

- Figure 24: Risk Analysis View

3.2.2 Addresses

The Addresses relationship links a Stakeholder Requirement to the Requirement Transfer Function supporting it.

3.2.2.1 Roles

- Requirement: The Requirement represents the set of technical requirement statements supporting and used in the Validation of the Stakeholder Requirement(s). This role is played by Requirement Transfer Function. This role's cardinality is Many.

- Need: The statement elicited from and validated against by an Advocate. This role is played by a Stakeholder Requirement. This role's cardinality is Many.

3.2.2.2 Metamodel View References

- Figure 4: Feature Framework View

3.2.3 Advocates

The Advocates relationship links a Need to the Advocate it would be elicited from or validating it against delivered System performance.

3.2.3.1 Roles

- Advocate: The Logical System represents a Stakeholder during the elicitation of Needs and in the Validation of the Requirements and the System. This role is played by Logical System. This role's cardinality is Many.

- Need: The statement elicited from and validated against by an Advocate. This role is played by a Need. This role's cardinality is Many.

3.2.3.2 Metamodel View References

- Figure 4: Feature Framework View

3.2.4 Allocated To

The Allocated To relationship assigns a Class to a Reified Relationship Role in a Reified Relationship.

3.2.4.1 Roles

- Class: The class that plays the role in the relationship. This role is played by Class, System, Allocation Decision, Design Component, Input/Output, Modeled Attribute, Functional Role, and Logical System. Its cardinality is 1.

- Role: The role is the part in a relationship that is played by a Class it is allocated to. This role is played by Reified Relationship Role, Architectural Relationship Role, System of Access (SOA), Functional Role, Input Role, Output Role, Attribute Role, Feature Attribute Role, Role Attribute Role, and Design Component Attribute Role. This role's cardinality is Many.

3.2.4.2 Configuration Attributes

- Design Component Population Rule

- Design Component Primary Key Value Rule

3.2.4.3 Metamodel View References

- Figure 12: Reified Relationship View
- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 7: Detail Requirements View
- Figure 8: High Level Design View

3.2.5 Appears In

The Appears In relationship groups any type of Class into a Domain. These groupings are often organized by enterprise organizations, technologies, or products.

3.2.5.1 Roles

- Class: The Class that is organized into a domain category. This role is played by all Classes. Its cardinality is Many.
- Domain: The category that organizes classes into a group. This role is played by Domain. This role's cardinality is Many.

3.2.5.2 Metamodel View References

- Figure 3: General Class View

3.2.6 Benefits

The Benefits relationship relates a Feature to the stakeholders it benefits.

3.2.6.1 Roles

- Feature: The marketable value or valuable service that attempts to benefit a Stakeholder. This role is played by a Feature (Service). This role's cardinality is Many.

- Stakeholder: The Logical System that a Person or Organization plays that is most directly impacted by the change or benefit a Need request upon a System. This role is played by Logical System. This role's cardinality is Many.

3.2.6.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.7 Can Have Value

The Can Have Value relationship relates a Modeled Attribute of a Feature to an allowed (valid) value that the Modeled Attribute is permitted to take on.

3.2.7.1 Roles

- Allowed Value: The value that a Feature Attribute may (is allowed to) take on. This role is played by Allowed Value This role's cardinality is Many.

- Attribute: The Feature Attribute that can take on a value. This role's cardinality is One.

3.2.7.2 Specific Attributes

- None

3.2.7.3 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.8 Causes Behavior

The Causes Behavior relationship links Counter Requirement Statements with Failure Modes.

3.2.8.1 Roles

- Counter Statement: This is the role played by the Counter Requirement Statement. This role's cardinality is Many.

- Caused Mode: This is the role played by the Failure Mode. This role's cardinality is One.

3.2.8.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.9 Causes Failure Mode

The Causes Failure Mode relationship links Failure Modes to the Failure Mode Context Element.

3.2.9.1 Roles

- Direct Cause: This is the role played by the Failure Mode Context Element, on behalf of a Functional Interaction. This role's cardinality is 0 to 1.
- Caused Mode: This is the role played by the Failure Mode. This role's cardinality is 0 to 1.

3.2.9.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.10 Causes Impact

The Causes Impact relationship links Failure Impacts to the Counter Requirement Statements.

3.2.10.1 Roles

- Counter Statement: This is the role played by the Counter Requirement Statement. This role's cardinality is Many.
- Impact: This is the role played by the Failure Impact. This role's cardinality is Many.

3.2.10.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.11 Contains

The Contains relationship is a generic compilation or whole-part relationships between classes of the same metaclass. This relationship is represented by a diamond head towards the larger or containing class. This relationship is most similar to a UML™ composition relationship.

3.2.11.1 Roles

- Container Class: The larger class that includes the contained class. This role is played by all Classes. This role's cardinality is 1.
- Contained Class: The smaller class that aggregates with other small classes to form the larger Container Class. This role is played by all Classes. This role's cardinality is Many.

3.2.11.2 Metamodel View References

- Figure 3: General Class View

- Figure 14: Functional Interaction View
- Figure 16: Design Constraint View
- Figure 6: Logical Architecture View
- Figure 7: State Analysis View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.2.12 Derived From

The Derived From relationship links a class's purpose or origin to one or more classes. This relationship is often used for validation purposes, to trace the origin or disposition of information.

3.2.12.1 Roles

- Source: The statement or class impacting upon the destination. This role is played by all Classes. This role's cardinality is Many.
- Destination: The derived class that is impacted by or validated from the Source Class. This role is played by all Classes. This role's cardinality is Many.

3.2.12.2 Metamodel View References

- Figure 3: General Class View
- Figure 14: Functional Interaction View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 9: High Level Design View

3.2.13 Detects Failure Mode

The Detects Failure Mode relationship links Failure Modes with the Failure Mode Context Element.

3.2.13.1 Roles

- Detected Mode: This is the role played by the Failure Mode. This role's cardinality is 0 to 1.
- Detector: This is the role played by the Failure Mode Context Element. This role's cardinality is 0 to 1.

3.2.13.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.14 Groups

The Groups relationship links an Interface to the set of Ports it is used to group or manage.

3.2.14.1 Roles

- Interface: The Interface that groups the Port. This role is played by an Interface. This role's cardinality is 0 to 1.
- Port: The Port that is grouped by an Interface. This role is played by a Port. This role's cardinality is Many.

3.2.14.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View

3.2.15 Has Advocate

The Has Advocate relationship links a Logical System playing the Stakeholder role in a Has Stakeholder relationship to another Logical System that would represent that Stakeholder in evaluating a System's deliverable with respect to a Need.

3.2.15.1 Roles

- Advocate: The Logical System represents a Stakeholder during the elicitation of Needs and in the Validation of the Requirements and the System. This role is played by Logical System. This role's cardinality is Many.
- Stakeholder: The Logical System that a Person or Organization plays that is most directly impacted by the change or benefit a Need request upon a System. This role is played by Logical System. This role's cardinality is Many.

3.2.15.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.16 Has Attribute

The Has Attribute relationship links a Modeled Attribute to any Class that has that Attribute.

3.2.16.1 Roles

- Attribute: The attribute that models a property of a Class. This role is played by Modeled Attribute. This role's cardinality is Many.
- Class: The class that has a property modeled by the Attribute. This role is played by all Classes. This role's cardinality is 1.

3.2.16.2 Metamodel View References

- Figure 3: General Class View
- Figure 4: Feature Framework View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.2.17 Has Feature

The Has Feature relationship links a subject's system to a Stakeholder Feature.

3.2.17.1 Roles

- Feature: The feature that provides value for the stakeholders of a system. This role is played by a Feature (Service). This role's cardinality is Many.
- Subject System: The system that offers certain Features. This role is played by a System. Its cardinality is 1.

3.2.17.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.18 Has Role

The Has Role relationship connects a relationship to the roles described for that relationship.

3.2.18.1 Roles

- Relationship: The relationship between two or more classes. This role is played by Reified Relationship, Architectural Relationship, Manages, Functional Interaction, Allocation Decision, Requirement Relationship, Design Constraint, Attribute Coupling, Requirements Coupling, and Design Coupling. This role's cardinality is 1 to Many.

- Role: A role is a part within a relationship that is played by a Class. This role is played by Reified Relationship Role, Architectural Relationship Role, Functional Role, Input Role, Output Role, Attribute Role, Feature Attribute Role, Role Attribute Role, and Design Component Attribute Role. Its cardinality is 1 or 2 to Many.

3.2.18.2 Configuration Attributes

- Role Population Rule

- Role Primary Key Value Rule

3.2.18.3 Metamodel View References

- Figure 12: Reified Relationship View

- Figure 13: Architectural Relationship View

- Figure 14: Functional Interaction View

- Figure 15: Requirement Relationship View

- Figure 16: Design Constraint View

- Figure 18: Attribute Coupling View

- Figure 19: Fitness Coupling View

- Figure 20: Characterization Coupling View

- Figure 21: Decomposition Coupling View

- Figure 22: Input/Output Coupling View

- Figure 5: Domain Analysis View

- Figure 6: Logical Architecture View

- Figure 8: Detail Requirements View

- Figure 9: High Level Design View

3.2.19 Has Stakeholder

The Has Stakeholder relationship links a stakeholder to a Domain's subject system.

3.2.19.1 Roles

- Stakeholder: The Logical System that a Person or Organization plays that is most directly impacted by the change or benefit a Need request upon a System. This role is played by Logical System. This role's cardinality is Many.

- Subject System: The System that is being specified or is the focus of attention in a Domain. This role is played by System. This role's cardinality is Many.

3.2.19.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.20 Has State

The Has State relationship requires that a situation in which a System participates (through external interaction) is modeled as a State for that System.

3.2.20.1 Roles

- System: A System that participates during the State. This role is played by a System. This role's cardinality is 1.

- State: The situation in which a System participates. This role is played by a State. This role's cardinality is Many.

3.2.20.2 Metamodel View References

- Figure 7: State Analysis View

3.2.21 Has Subject

The Has Subject relationship links a Domain to a System that is the focus of attention and is being specified.

3.2.21.1 Roles

- Domain: The Domain with the Subject as its focus point. This role is played by a Domain. This role's cardinality is Many.

- Subject: The System that is the focus point and subject of a Domain. This role is played by a System, Logical System, and Design Component. This role's cardinality is 1 to Many.

3.2.21.2 Metamodel View References

- Figure 5: Domain Analysis View

- Figure 6: Logical Architecture View

3.2.22 Has Value

The Has Value relationship links a Modeled Attribute to a Value it has.

3.2.22.1 Roles

- Attribute: The Modeled Attribute that has one or more Values. This role is played by Modeled Attribute. This role's cardinality is 1.

- Value: The Value of a Modeled Attribute. This role is played by Value. This role's cardinality is Many.

3.2.22.2 Specific Attributes

- None

3.2.22.3 Metamodel View References

- Figure 3: General Class View

- Figure 4: Feature Framework View

3.2.23 Has View

The Has View relationship links a Reified Relationship to the various Modeled Statements that describe it and how its role relates to each other.

3.2.23.1 Roles

- Relationship: The relationship between two or more classes. This role is played by Reified Relationship, Allocation Decision, Requirement Relationship, Design Constraint, Attribute Coupling, Requirements Coupling, and Design Coupling. This role's cardinality is 1.

- Statement: The statement describing how the relationship's roles relate. This role is played by Modeled Statement, Rationale, Requirement Statement, Design Constraint Statement, Attribute Coupling Map, Requirements Coupling Map, and Design Coupling Map. This role's cardinality is Many.

3.2.23.2 Metamodel View References

- Figure 12: Reified Relationship View

- Figure 13: Architectural Relationship View

- Figure 14: Functional Interaction View

- Figure 15: Requirement Relationship View

- Figure 16: Design Constraint View

- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View
- Figure 9: High Level Design View

3.2.24 Impacts Feature

The Impacts Feature relationship links Stakeholder Features and Failure Impacts.

3.2.24.1 Roles

- Feature: This is the role played by the Stakeholder Feature. This role's cardinality is One.
- Impact: This is the role played by the Failure Impact. This role's cardinality is Many.

3.2.24.2 Configuration Attributes

- Failure Impact Population Rule
- Failure Impact Primary Key Value Rule

3.2.24.3 Metamodel View References

- Figure 24: Risk Analysis View

3.2.25 Impacts Stakeholder

The Impacts Stakeholder relationship links Stakeholders with Failure Impacts.

3.2.25.1 Roles

- Impact: This is the role played by the Failure Impact. This role's cardinality is Many.
- Stakeholder: The Logical System that a Person or Organization plays that is most directly impacted by the Failure. This role is played by Logical System. This role's cardinality is Many.

3.2.25.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.26 Is a Type of

The Is a Type of relationship is a generic taxonomy, generalization, or abstraction relationship between two classes. This relationship is represented in UML™ by an arrow from the more special class (subclass) towards the more general class (superclass).

3.2.26.1 Roles

- Superclass: The class that generalizes the Subclass. This role is played by all Classes. This role's cardinality is Many.
- Subclass: The class that is generalized by the Superclass. This role is played by all Classes. This role's cardinality is Many.

3.2.26.2 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 11: Reified Relationship Views View
- Figure 13: Architectural Relationship View
- Figure 14: Functional Interaction View
- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 18: Attribute Coupling View
- Figure 19: Fitness Coupling View
- Figure 20: Characterization Coupling View
- Figure 21: Decomposition Coupling View
- Figure 22: Input/Output Coupling View

3.2.27 Is Constrained By

The Is Constrained By relationship describes which Design Component is the subject of a Design Constraint.

3.2.27.1 Roles

- Component: The Design Component that is the subject of the Design Constraint. This role is played by a Design Component. This role's cardinality is 1.
- Constraint: The Design Constraint that restricts aspects of a Design Component. This role is played by a Design Constraint. This role's cardinality is Many.

3.2.27.2 Metamodel View References

- Figure 9: High Level Design View

3.2.28 Is Facilitated By Externally

The Is Facilitated By Externally relationship links a Port to the System of Access that it uses outside of the System boundary.

3.2.28.1 Roles

- Port: The Port that uses the System of Access outside of the System boundary. This role is played by a Port. This role's cardinality is Many.
- SOA: The System of Access that links to a Port outside of the System boundary. This role is played by a System of Access (SOA). This role's cardinality is 1.

3.2.28.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View

3.2.29 Is Facilitated By Internally

The Is Facilitated By Internally relationship links a Port to the System of Access that it uses inside of the System boundary.

3.2.29.1 Roles

- Port: The Port that uses the System of Access inside of the System boundary. This role is played by a Port. This role's cardinality is Many.
- SOA: The System of Access that links to a Port inside of the System boundary. This role is played by a System of Access (SOA). This role's cardinality is 1.

3.2.29.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View

3.2.30 Is Specified By

The Is Specified By relationship describes which Functional Role is the subject of a Requirement Relationship.

3.2.30.1 Roles

- Requirement: A Requirement Relationship specifying a Functional Role. This role is played by a Requirement Relationship. This role's cardinality is Many.

- Role: The Functional Role being specified by the Requirement Relationship. This role is played by a Functional Role. This role's cardinality is 1.

3.2.30.2 Metamodel View References

- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.2.31 Is Triggered By

The Is Triggered By relationship describes which Event causes one State to end and another to begin.

3.2.31.1 Roles

- Transition: A path triggered by the Event. This role is played by a Transition. This role's cardinality is Many.
- Trigger: The Event that triggers the Transition from State to another. This role is played by an Event. This role's cardinality is Many.

3.2.31.2 Metamodel View References

- Figure 7: State Analysis View

3.2.32 Mitigates Failure Mode

The Mitigates Failure Mode relationship links Failure Modes with the Failure Mode Context Element.

3.2.32.1 Roles

- Mitigated Mode: This is the role played by the Failure Mode. This role's cardinality is 0 to 1.
- Mitigator: This is the role played by the Failure Mode Context Element, on behalf of an Interaction. This role's cardinality is 0 to 1.

3.2.32.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.33 Perceives

The Perceives relationship is between a Stakeholder and the Need they perceive for the subject system.

3.2.33.1 Roles

- Need: The statement elicited from and validated against by an Advocate. This role is played by a Need. This role's cardinality is Many.

- Stakeholder: The Logical System that a Person or Organization plays that is most directly impacted by the change or benefit a Need request upon a System. This role is played by Logical System. This role's cardinality is Many.

3.2.33.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.34 Permits Architectural Relationship

The Permits Architectural Relationship relationship links an Interface to the allowed Architectural Relationships with which its Ports can be linked.

3.2.34.1 Roles

- AR: The Architectural Relationship allowed by the Interface. This role is played by an Architectural Relationship. This role's cardinality is Many.

- Interface: The Interface that allows the Functional Interaction. This role is played by an Interface. This role's cardinality is Many.

3.2.34.2 Metamodel View References

- Figure 5: Domain Analysis View

- Figure 6: Logical Architecture View

- Figure 8: Detail Requirements View

3.2.35 Permits Functional Interaction

The Permits Functional Interact relationship links an Interface to the allowed Functional Interactions for which its Ports can be used.

3.2.35.1 Roles

- FI: The Functional Interaction allowed by the Interface. This role is played by a Functional Interaction. This role's cardinality is Many.

- Interface: The Interface that allows the Functional Interaction. This role is played by an Interface. This role's cardinality is Many.

3.2.35.2 Metamodel View References

- Figure 5: Domain Analysis View

- Figure 8: Detail Requirements View

3.2.36 Permits Input/Output

The Permits Input/Output relationship links an Interface to the allowed Input/Outputs to which its Ports can link.

3.2.36.1 Roles

- Interface: The Interface that allows the Input/Output. This role is played by an Interface. This role's cardinality is 0 to 2.
- I/O: The Input/Output that is allowed through an Interface. This role is played by an Input/Output. Its cardinality is Many.

3.2.36.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 8: Detail Requirements View

3.2.37 Permits SOA

The Permits SOA relationship links an Interface to the allowed Systems of Access (SOAs) to which its Ports can link.

3.2.37.1 Roles

- Interface: The Interface that allows the System of Access. This role is played by an Interface. This role's cardinality is Many.
- SOA: The System of Access that is permitted. This role is played by a System of Access (SOA). This role's cardinality is Many.

3.2.37.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View
- Figure 9: High Level Design View

3.2.38 Predicts Failure Mode

The Predicts Failure Mode relationship links Failure Modes with the Failure Mode Context Element.

3.2.38.1 Roles

- Predicted Mode: This is the role played by the Failure Mode. This role's cardinality is 0 to 1.
- Predictor: This is the role played by the Failure Mode Context Element on behalf of an Interaction. This role's cardinality is 0 to 1.

3.2.38.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.39 Prevents Failure Mode

The Prevents Failure Mode relationship links Failure Modes with the Failure Mode Context Element.

3.2.39.1 Roles

- Prevented Mode: This is the role played by the Failure Mode. This role's cardinality is 0 to 1.

- Preventor: This is the role played by the Failure Mode Context Element, on behalf of an Interaction. This role's cardinality is 0 to 1.

3.2.39.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.40 Provides Context

The Provides Context relationship defines for which Functional Interaction a Requirement Relationship is valid.

3.2.40.1 Roles

- FI: The Functional Interaction for which the Requirement Relationship is valid. This role is played by a Functional Interaction. This role's cardinality is 1.

- Requirement: A Requirement Relationship specified during a Functional Interaction. This role is played by a Requirement Relationship. This role's cardinality is Many.

3.2.40.2 Metamodel View References

- Figure 8: Detail Requirements View

- Figure 24: Risk Analysis View

3.2.41 Provides Event Context

The Provides Event Context relationship defines the Functional Interaction context for an Event.

3.2.41.1 Roles

- FI: The Functional Interaction for which the Event is valid. This role is played by a Functional Interaction. This role's cardinality is 1.

- Event: An event during a Functional Interaction. This role is played by an Event. This role's cardinality is Many.

3.2.41.2 Specific Attributes

- From State PK Matching Rule
- To State PK Matching Rule
- Event PK Value Rule

3.2.41.3 Metamodel View References

- Figure 7: State Analysis View

3.2.42 Provides Failure Context

The Provides Failure Context relationship links Functional Interactions with the Failure Mode Context Element.

3.2.42.1 Roles

- Contextual Interaction: This is the role played by the Functional Interaction. This role's cardinality is One.
- Context: This is the role played by the Failure Mode Context Element on behalf of a Failure Mode. This role's cardinality is One.

3.2.42.2 Metamodel View References

- Figure 24: Risk Analysis View

3.2.43 Provides Interface

The Provides relationship links an Interface to a System.

3.2.43.1 Roles

- Interface: The Interface that is provided by the System. This role is played by an Interface. This role's cardinality is Many.
- System: The System that has the Interface. This role is played by a System, Logical System, and Design Component. Its cardinality is 1.

3.2.43.2 Metamodel View References

- Figure 5: Domain Analysis View
- Figure 6: Logical Architecture View
- Figure 9: High Level Design View

3.2.44 Receives

The Receives relationship links an internal Input/Output to an output Port or an external Input/Output to an input Port.

3.2.44.1 Roles

- I/O: The Input/Output that is being received at the Port. This role is played by an Input/Output. This role's cardinality is 0 to 1.
- Port: The Port that is receiving the Input/Output. This role is played by a Port. This role's cardinality is 1.

3.2.44.2 Metamodel View Reference

- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View

3.2.45 Relates AR

The Relates AR relationship links an Architectural Relationship to an Interface Element Relationship as part of a model of an interface context.

3.2.45.1 Roles

- Architectural Relationship: The Architectural Relationship that is related to the Interface Element Relationship. This role is played by an Architectural Relationship. This role's cardinality is 0 to 1.
- Interface Element Relationship: The Interface Element Relationship that is related to the Architectural Relationship. This role is played by an Interface Element Relationship. This role's cardinality is 0 to Many.

3.2.45.2 Metamodel View Reference

- Figure 5: Domain Analysis View
- Figure 10: Interface Context View

3.2.46 Relates FI

The Relates FI relationship links a Functional Interaction to the Interface Element Relationship of an interface context in which that interaction participates.

3.2.46.1 Roles

- Functional Interaction: The Functional Interaction that is related to the Interface Element Relationship. This role is played by a Functional Interaction. This role's cardinality is 0 to 1.

- Interface Element Relationship: The Interface Element Relationship that is related to the Functional Interaction. This role is played by an Interface Element Relationship. This role's cardinality is 0 to Many.

3.2.46.2 Metamodel View Reference

- Figure 5: Domain Analysis View

- Figure 10: Interface Context View

3.2.47 Relates IO

The Relates IO relationship links an Input/Output to the Interface Element Relationship of an interface context in which that Input/Output participates.

3.2.47.1 Roles

- I/O: The Input/Output that is related to the Interface Element Relationship. This role is played by an Input/Output. This role's cardinality is 0 to 1.

- Interface Element Relationship: The Interface Element Relationship that is related to the Input/Output. This role is played by an Interface Element Relationship. This role's cardinality is 0 to Many.

3.2.47.2 Metamodel View Reference

- Figure 5: Domain Analysis View

- Figure 10: Interface Context View

3.2.48 Relates LS

The Relates LS relationship links a Logical System to the Interface Element Relationship of an interface context in which that system participates.

3.2.48.1 Roles

- Logical System: The Logical System that is related to the Interface Element Relationship. This role is played by a Logical System. This role's cardinality is 0 to 1.

- Interface Element Relationship: The Interface Element Relationship that is related to the Logical System. This role is played by an Interface Element Relationship. This role's cardinality is 0 to Many.

3.2.48.2 Metamodel View Reference

- Figure 5: Domain Analysis View
- Figure 10: Interface Context View

3.2.49 Replaces

The Replaces relationship links Requirement Statements and Counter Requirement Statements.

3.2.49.1 Roles

- Statement: This is the role played by the Requirement Statement. This role's cardinality is One.
- Counter Statement: This is the role played by the Counter Requirement Statement. This role's cardinality is One to Many.

3.2.49.2 Specific Attributes

- Counter Requirement Population Rule
- Counter Requirement Primary Key Value Rule

3.2.49.3 Metamodel View References

- Figure 24: Risk Analysis View

3.2.50 Requires

The Requires relationship asserts that a Functional Interaction is required or expected during a certain State.

3.2.50.1 Roles

- FI: A required Functional Interaction between Systems. This role is played by a Functional Interaction. Its cardinality is Many.
- State: The situation that requires a Functional Interaction. This role is played by a State. This role's cardinality is 1.

3.2.50.2 Specific Attributes

- State Population Rule-Interaction
- State Population Rule-Role
- State Primary Key Value Rule

3.2.50.3 Metamodel View References

- Figure 7: State Analysis View

3.2.51 Satisfies

The Satisfies relationship links a Need to the Features of a System that attempt to satisfy it.

3.2.51.1 Roles

- Feature: The marketable value or valuable service that attempts to satisfy a set of Needs. This role is played by a Feature (Service). This role's cardinality is Many.
- Need: The statement describing what a Stakeholder desires of a System's Features. This role is played by a Need. This role's cardinality is Many.

3.2.51.2 Metamodel View Reference

- Figure 4: Feature Framework View

3.2.52 Sends

The Sends relationship links an external Input/Output to an output Port or an internal Input/Output to an input Port.

3.2.52.1 Roles

- I/O: The Input/Output that is being sent from the Port. This role is played by an Input/Output. This role's cardinality is 0 to 1.
- Port: The Port that is sending the Input/Output. This role is played by a Port. This role's cardinality is 1.

3.2.52.2 Metamodel View Reference

- Figure 15: Requirement Relationship View
- Figure 16: Design Constraint View
- Figure 5: Domain Analysis View
- Figure 8: Detail Requirements View

3.2.53 Transitions From

The Transitions From relationship links a Transition to the State it is leaving.

3.2.53.1 Roles

- From: The State that ends during the transition. This role is played by a State. This role's cardinality is 1.

- Transition: A path leaving the From State. This role is played by a Transition. This role's cardinality is Many.

3.2.53.2 Metamodel View References

- Figure 7: State Analysis View

3.2.54 Transitions To

The Transitions To relationship links a Transition to the State it is entering.

3.2.54.1 Roles

- To: The State that begins during the transition. This role is played by a State. This role's cardinality is 1.

- Transition: A path entering the To State. This role is played by a Transition. This role's cardinality is Many.

3.2.54.2 Metamodel View References

- Figure 7: State Analysis View

3.2.55 Uses Functional Interaction

The Uses Functional Interaction relationship asserts that a certain Functional Interaction is required to deliver at least part of a Stakeholder Feature's value.

3.2.55.1 Roles

- Feature: The Stakeholder Feature whose value is supported by the Functional Interaction. This role is played by a Stakeholder Feature. This role's cardinality is Many.

- Interaction: The Functional Interaction that supports the Stakeholder Feature's value. This role is played by a Functional Interaction. Its cardinality is Many.

3.2.55.2 Configuration Attributes

- Interaction Population Rule

- Interaction Primary Key Value Rule

3.2.55.3 Metamodel View Reference

- Figure 4: Feature Framework View

- Figure 14: Functional Interaction View

- Figure 19: Fitness Coupling View

- Figure 24: Risk Analysis View

3.3 Metaclass and Metarelationship Attributes

Metaclass attributes are properties of a metaclass. These properties (along with the metaclass relationships above) allow a metaclass to parameterize its concepts.

3.3.1 Common Attributes

Common Attributes are the subset of metaclass and metarelationship properties that apply to all Metaclasses and Metarelationships (In what follows below, read “class” as meaning “class or relationship”.)

3.3.1.1 Author

An Author of a class is the person who last made changes to that class.

3.3.1.2 Change Date

The Change Date of a class the time and date in which the latest changes were made to that class.

3.3.1.3 Change Description

The Change Description of a class is an explanation of the changes made to the previous version of that class.

3.3.1.4 Class Level

The Class Level of a class is the depth of the class hierarchy in which that class is defined. This attribute indicates how abstract or specific a class with reference to the other classes defined. The smaller the level number, the more abstract a class is. The definitions and meanings of the class levels vary and are specific to an enterprise.

3.3.1.5 Definition

The Definition of a class is a short summary of the concept that class models.

3.3.1.6 ID

The ID of a class is a unique identifier of that class.

3.3.1.7 Major Version

The Major Version of a class signifies the number of substantial changes of that class. A class with version X.Y.Z has a Major Version of X.

3.3.1.8 Minor Version

The Minor Version of a class signifies the number of significant yet less than substantial changes of that class. A class with version X.Y.Z has a Minor Version of Y.

3.3.1.9 Name

The Name of a class is a short label or title by which that class is identified and summarizes that class's concepts.

3.3.1.10 Organization Owner

An Organization Owner of a class is the organization that is responsible for maintaining and managing a class's attribute values and relationships.

3.3.1.11 Owner

An Owner of a class is the person responsible for managing a class's attribute values and relationships.

3.3.1.12 Status

The Status of a class is the systems engineering procedural state in which the class is at. The status values, definitions, and meanings vary and are specific to an enterprise and even class.

3.3.1.13 Update Version

The Update Version of a class signifies the number of insignificant changes or bug fixes of that class. A class with version X.Y.Z has an Update Version of Z

3.3.2 Specific Attributes including Configuration Rule Sets

Specific Attributes occur only for certain classes or relationships, and are listed individually when then apply in Sections 3.1 and 3.2. Some of these Specific Attributes are used as S*Pattern Configuration Rules for creation or checking of S*Models against an S*Pattern. In particular, they may be used to describe configuration rules about Primary Key (PK) attributes. That subject is further discussed in Section 2.22.

3.3.2.1 Date Submitted

The Date Submitted of a class is the date in which a Need was first recognized and recorded.

3.3.2.2 Due Date

The Due Date of a Need is the date by which that Need must be fulfilled.

3.3.2.3 Originator

An Originator of a Need is the person or organization that first raised the Need upon a System.

3.3.2.4 Priority

A Priority of a Need describes the relative importance of fulfilling a Need of a System.

3.3.2.5 Reference

A Reference is a listing to find more information concerning a Modeled Statement.

3.3.2.6 Request Type

A Request Type of a Need is an enterprise specific categorization of a Need.

3.3.2.7 Source

A Source is the document in which a Need was originally stated or documented.

3.3.2.8 Configuration Rules Set: Pattern Feature Attribute Values

During population of a configured model from a pattern, the pattern user selects Features from the pattern, for population in the configured model. Advisory Configuration Rules from the pattern provide additional user guidance in that selection, such as indication of mandatory versus optional Features. For populating pattern Features that have Feature Primary Key (FPK) Attributes, the user also selects the values of those FPK Attributes to be populated. Those values are selected from a pattern set of valid or allowed FPK Attribute values.

This occurs under the control of Configuration Rules indicating:

1. The names of pattern-based Features which the pattern user can select Features to populate.

2. User advisory Configuration Rules for those Features, which indicate mandatory, optional, or other kinds of constraints on the user's selection of Features.

3. For each such Feature that has an FPK Attribute, the name of that attribute and an enumerated list of possible (valid, allowed) values from which the user can select values to be populated for those attributes.

Examples appear in the lower rows:

Features Table

Feature Name	Advisory Configuration Rule
<Feature Name>	<Feature Advisory Configuration Rule>
<Feature Name>	<Feature Advisory Configuration Rule>
<Feature Name>	<Feature Advisory Configuration Rule>
Safety Feature	Mandatory
Passenger Comfort Feature	Optional

FPK Attribute Values Table

Feature Name	FPK Attribute Name	Allowed Value
<Feature Name>	<Feature Primary Key Attribute Name>	<value1>
<Feature Name>	<Feature Primary Key Attribute Name>	<value 2>
<Feature Name>	<Feature Primary Key Attribute Name>	<value 3>
Passenger Comfort Feature	Comfort Aspect	Leg Room
Passenger Comfort Feature	Comfort Aspect	Head Room
Passenger Comfort Feature	Comfort Aspect	Seat Back Fit
Safety Feature	Safety Risk Type	Ejection
Safety Feature	Safety Risk Type	Dash Injury
Safety Feature	Safety Risk Type	Traction Loss

Configuration Rule Attributes

Entry in Advisory Configuration Rule Column	Interpretation
Optional	Means the pattern user selection of the Feature for population is at the option of the user.
Mandatory	Means the pattern user is advised to always populate this Feature.
<any other text>	Any other advice to the pattern user as to population of non-population of the Feature.
(empty)	Means no advice to pattern user as to configuration of the Feature.

Entry in Allowed Value Column	Interpretation
<any value>	Means a possible value of the FPK Attribute, which the pattern configuring user can select.

3.3.2.9 Configuration Rules Set: Pattern Features-Interactions

During auto-population of a configured model from a pattern, the user-selected set of Features populated leads to auto-population of the supporting Interactions. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required Feature type that must already have been populated.

2. Responses to what Feature Primary Key (FPK) values have been chosen by the pattern configuring user.

When the above conditions are satisfied, then for the Interaction to be populated, the configuration rule also indicates:

3. The name of the Interaction to be populated.

4. The Interaction Primary Key (IPK) Rule as to what value the Interaction primary key should have.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF FEATURE ALREADY POPULATED (FPK must also be satisfied)		**THEN POPULATE INTERACTION** (with an IPK Value)	
Feature	**Interaction Population Rule**	**Interaction**	**Interaction PK Value Rule**
<Feature Name>		<Interaction Name>	*ANY*
<Feature Name>	<value>	<Interaction Name>	FPK + /<literal string>/
Reliability & Availability Feature		Travel Over Terrain	/Reliability & Availability/
Passenger Comfort Feature	*ANY*	Ride In Vehicle	FPK

Configuration Rule Attributes

Entry in Interaction Population Rule Column	**Interpretation**
ANY	*ANY* means that any FPK value for a populated instance of the Feature type listed in the rule may cause population of an instance of the specified Interaction.
<value>	Any other value means a populated instance of the Feature type in the rule must have FPK value equal to that listed, for it to be able to populate an instance of the specified Interaction type.
(empty)	Populated instance of Feature type must have (empty) FPK value, for it to be able to populate an instance of the specified Interaction type.
Entry in Interaction PK Value Rule Column	**Interpretation**

FPK	Means set the IPK of the populated Interaction to equal the value of the FPK of the already populated Feature associated with the Interaction by this rule.
/<literal string>/	Means set the IPK of the populated Interaction to be <literal string>
FPK + /<literal string>/	Means set the IPK of the populated Interaction to equal the value of the FPK of the Feature associated with the Interaction by this rule plus the <literal string>
ANY	Used to populate additional Feature-Interaction relationships for already (using the other PK value rules) populated Interactions, without populating new Interactions.
(empty)	Means set the IPK of the populated Interaction to blank.

3.3.2.10 Configuration Rules Set: Pattern Interactions-Roles

During auto-population of a configured model from a pattern, the already populated Interaction leads to auto-population of the Roles which participate in that Interaction. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required Interaction type that must already have been populated.

2. What Interaction Primary Key (IPK) value has been chosen.

When the above conditions are satisfied, then for the Role to be populated, the configuration rule also indicates:

3. The name of the Role to be populated.

4. The Role Primary Key (RPK) Rule as to what value the Role primary key should have.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF INTERACTION ALREADY POPULATED (IPK must also be satisfied)		**THEN POPULATE ROLE** (with an RPK Value)	
Interaction	**Role Population Rule**	**Role**	**Role PK Value Rule**
<Interaction Name>	<IPK Value>	<Role Name>	<RPK Rule>
Configure Vehicle	*ANY*	Vehicle	IPK

Configuration Rule Attributes

Entry in Role Population Rule Column	**Interpretation**
ANY	*ANY* means that any IPK value for a populated instance of the Interaction type listed in the rule may cause population of an instance of the specified Role.

<value>	Any other value means a populated instance of the Interaction type in the rule must have IPK value equal to that listed, for it to be able to populate an instance of the specified Role type.
(empty)	Populated instance of Interaction type must have (empty) IPK value, for it to be able to populate an instance of the specified Role type.
Entry in Role PK Value Rule Column	**Interpretation**
IPK	Means set the RPK of the populated Role to equal the value of the IPK of the Interaction associated with the Role by this rule.
/<literal string>/	Means set the RPK of the populated Role to be <literal string>
IPK + /<literal string>/	Means set the RPK of the populated Role to equal the value of the IPK of the Interaction associated with the Role by this rule plus the <literal string>

3.3.2.11 Configuration Rules Set: Roles-Design Components

During auto-population of a configured model from a pattern, the already populated Role leads to auto-population of the Design Component to which the Role is allocated. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required Role type that must already have been populated.

2. What Role Primary Key (RPK) value has been chosen.

When the above conditions are satisfied, then for the Design Component to be populated, the configuration rule also indicates:

3. The name of the Design Component to be populated.

4. The Design Component Primary Key (DCPK) Rule as to what value the Design Component primary key should have.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF ROLE ALREADY POPULATED (RPK must also be satisfied)		**THEN POPULATE DESIGN COMPONENT** (with an DCPK Value)	
Role	**Design Component Population Rule**	**Design Component**	**Design Component PK Value Rule**
<Role Name>	<RPK Value>	<Design Component Name>	<DCPK Rule>
Vehicle	*ANY*	Vehicle	RPK

Configuration Rule Attributes

Entry in Design Component Population Rule Column	Interpretation
ANY	*ANY* means that any RPK value for a populated instance of the Role type listed in the rule may cause population of an instance of the specified Design Component.
<value>	Any other value means a populated instance of the Role type in the rule must have RPK value equal to that listed, for it to be able to populate an instance of the specified Design Component type.
(empty)	Populated instance of Role type must have (empty) RPK value, for it to be able to populate an instance of the specified Design Component type.
Entry in Design Component PK Value Rule Column	**Interpretation**
RPK	Means set the DCPK of the populated Design Component to equal the value of the RPK of the Role associated with the Design Component by this rule.
R1PK	Means set the DCPK of the populated Design Component to equal the first part of the value of the compound RPK of the Role associated with the Design Component by this rule.
R2PK	Means set the DCPK of the populated Design Component to equal the latter part of the value of the compound RPK of the Role associated with the Design Component by this rule.
/<literal string>/	Means set the DCPK of the populated Design Component to be <literal string>
RPK + /<literal string>/	Means set the DCPK of the populated Design Component to equal the value of the RPK of the Role associated with the Design Component by this rule plus the <literal string>

3.3.2.12 Configuration Rules Set: Interactions-Roles-Requirements

During auto-population of a configured model from a pattern, the combination of an already populated Interaction-Role instance pair, consisting of a certain Interaction and Role leads to auto-population of that Requirement. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required Interaction type that must already have been populated.
2. What Interaction Primary Key (IPK) value the existing Interaction instance must have.
3. The name of the required Role type that must already have been populated.
4. What Role Primary Key (RPK) value the existing Role instance must have.

When the above conditions are satisfied, then for the Requirement to be populated, the configuration rule also indicates:

5. The Requirement to be populated.

6. The Requirement Statement Primary Key (RSPK) Rule as to what value the Requirement primary key should have.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF ROLE-INTERACTION PAIR ALREADY POPULATED (both IPK and RPK must also be satisfied)				**THEN POPULATE REQUIREMENT** (with a Requirement Type and an RSPK Value)	
Interaction	**Requirement Population Rule-Interaction**	**Role**	**Requirement Population Rule-Role**	**Requirement**	**Requirement Statement PK Value Rule**
<Interaction Name>	<IPK Rule>	<Role Name>	<RPK Rule>	<Req Name>	<RSPK Rule>
Convert Electrical Power	*ANY*	Electrically Powered Device	*ANY*	REQ 003	IPK
Consumer Electrical Power		Local Power Distribution System		REQ 004	

Configuration Rule Attributes

Entry in Requirement Population Rule-Interaction Column	**Interpretation**
ANY	*ANY* means that any IPK value for a populated instance of the Interaction type listed in the rule may cause population of an instance of the specified Requirement.
<value>	Any other value means a populated instance of the Interaction type in the rule must have IPK value equal to that listed, for it to be able to populate an instance of the specified Requirement type.
(empty)	Populated instance of Interaction type must have (empty) IPK value, for it to be able to populate an instance of the specified Requirement type.
Entry in Requirement Population Rule-Role Column	**Interpretation**

ANY	*ANY* means that any RPK value for a populated instance of the Role type listed in the rule may cause population of an instance of the specified Requirement.
<value>	Any other value means a populated instance of the Role type in the rule must have RPK value equal to that listed, for it to be able to populate an instance of the specified Requirement type.
(empty)	Populated instance of Role type must have (empty) RPK value, for it to be able to populate an instance of the specified Requirement type.

Requirement Statement PK Value Rule	**Interpretation**
IPK	Means set the RSPK of the populated Requirement to equal the value of the IPK of the Interaction associated with the Requirement by this rule.
RPK	Means set the RSPK of the populated Requirement to equal the value of the RPK of the Role associated with the Requirement by this rule.
/<literal string>/	Means set the RSPK of the populated Requirement to be <literal string>
IPK + /<literal string>/	Means set the RSPK of the populated Requirement to equal the value of the IPK of the Interaction associated with the Requirement by this rule plus the <literal string>
RPK + /<literal string>/	Means set the RSPK of the populated Requirement to equal the value of the RPK of the Role associated with the Requirement by this rule plus the <literal string>
(empty)	If no RSPK rule, then populate the Requirement with no RSPK value.

3.3.2.13 Configuration Rules Set: Pattern Interactions-States

During auto-population of a configured model from a pattern, the combination of an already populated Interaction-Role instance pair, consisting of a certain Interaction (which can occur during a certain State) and Role (which can have a certain State) leads to auto-population of that State. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required Interaction type that must already have been populated.
2. What Interaction Primary Key (IPK) value the existing Interaction instance must have.
3. The name of the required Role type that must already have been populated.
4. What Role Primary Key (RPK) value the existing Role instance must have.

When the above conditions are satisfied, then for the State to be populated, the configuration rule also indicates:

5. The name of the State to be populated.
6. The State Type of the State to be populated.
7. The State Primary Key (SPK) Rule as to what value the State primary key should have.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF ROLE-INTERACTION PAIR ALREADY POPULATED (both IPK and RPK must also be satisfied)				**THEN POPULATE STATE** (with a State Type and an SPK Value)		
Interaction	**State Population Rule-Interaction**	**Role**	**State Population Rule-Role**	**State**	**State Type**	**State PK Value Rule**
<Interaction Name>	*<IPK Rule>*	*<Role Name>*	*<RPK Rule>*	*<State Name>*	*<State Type Rule>*	*<SPK Rule>*
<Interaction Name>	*<IPK Rule>*	*<Role Name>*	*<RPK Rule>*	*<State Name>*	*<State Type Rule>*	*<SPK Rule>*
Operate Starter	*ANY*	Vehicle	*ANY*	Starting	Simple	

Configuration Rule Attributes

Entry in State Population Rule-Interaction Column	**Interpretation**
ANY	*ANY* means that any IPK value for a populated instance of the Interaction type listed in the rule may cause population of an instance of the specified State. (This includes no IPK value.)
<value>	Any other value means a populated instance of the Interaction type in the rule must have IPK value equal to that listed, for it to be able to populate an instance of the specified State type. (This includes the case of no IPK value.)
Entry in State Population Rule-Role Column	**Interpretation**
ANY	*ANY* means that any RPK value for a populated instance of the Role type listed in the rule may populate an instance of the specified State. (This includes no RPK value.)
<value>	Any other value means that a populated instance of the Role type listed in the rule must have RPK value equal to that listed, in order for it to be able to populate an instance of the specified State type. (This includes no IPK value.)
Entry in State PK Value Rule Column	**Interpretation**
IPK	Means set the SPK of the populated State to equal the value of the IPK of the Interaction associated with the State by this rule.
I2PK	Means set the SPK of the populated State to equal the second half of the IPK of the Interaction associated with the State by this rule. (Assumes a divided two-part IPK of the form XXXX-YYYY, delimited by (-)).

/<literal string>/	Means set the SPK of the populated State to be <literal string>
(blank)	If no SPK Rule, then populate the State with no SPK value.

Entry in State Type Column	Interpretation
Simple	This is a “normal” state, for which the associated Interaction will be related to the state to indicate it occurs during that state.
Empty	This state is populated by an instance of the associated interaction, but no Interactions are associated with this state as occurring during that state.
Initial	This state is the first of a sequence of states, so may have associated state transitions “from” it, but not “to” it.
Final	This state is the last of a sequence of states, so may have associated state transitions “to” it, but not “from” it.

3.3.2.14 Configuration Rules Set: Pattern States-Transitions-Events

During auto-population of a configured model from a pattern, the combination of an already populated pair of States may cause auto-population of a certain State Transition between those states. This occurs conditionally based on Configuration Rules indicating:

1. The name of the required “from State” that must already have been populated.

2. The value of the State Primary Key (SPK) of the “from State” that must have already been populated.

3. The name of the required “to State” that must already have been populated.

4. The value of the State Primary Key (SPK) of the “to State” that must have already been populated.

5. Event Context Interaction: Whether the State Transition is auto-populated depends on not just the existence of the above “from” and “to” states, but also depends on the population of an Event that would trigger such a transition. Whether such an Event will be auto-populated depends on whether a particular Interaction has already been populated that provides the Event Context for that Event to occur.

When the above conditions are satisfied, then for the State Transition to be populated, the configuration rule also indicates:

6. The name of the Event to be populated.

7. The Event Primary Key (EPK) to be set for the Event that is to be populated

8. The State Transition Type of the transition to be populated.

9. The Transition PK Value that the populated Transition is to have.

10. For Join Transition types, a disambiguating name for the each of the multiple incoming transitions flows.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

IF STATES AND EVENT CONTEXT ALREADY POPULATED (both State PKs & Event Context Interaction must be satisfied)					**THEN POPULATE EVENT AND TRANSITION** (with PK values and Transition Type)				
From State	**From State PK Matching Rule**	**To State**	**To State PK Matching Rule**	**Event Context Interaction**	**Event**	**Event PK Value Rule**	**Transition Type**	**Transition PK Value Rule**	**Transition Name for Joins**
<State Name>	*<FSPK Rule>*	*<State Name>*	*<TSPK Rule>*	*<Interaction Name>*	*<Event Name>*	*<EPK Rule>*	*<Transition Type>*	*<TPK Rule>*	*<Transition Name>*
<State Name>	*<FSPK Rule>*	*<State Name>*	*<TSPK Rule>*	*<Interaction Name>*	*<Event Name>*	*<EPK Rule>*	*<Transition Type>*	*<TPK Rule>*	*<Transition Name>*
Starting	*ANY*	Idling	*ANY*	Perform Engine Operation	Engine Started	IPK	Simple	IPK	

Configuration Rule Attributes

Entry in From/To State PK Matching Rule Columns	**Interpretation**
IPK	Means the PK of the configured state must equal the PK of the Interaction providing Event context in order for a transition to be populated between the From State and the To State.
EPK	Means the PK of the configured state must equal the PK of the Event in order for a transition to be populated between the From State and the To State.
=	Means the PKs of the configured From State and To State must equal each other in order for a transition to be populated between the From State and the To State.
ANY	Means the PK of the configured state may be any value for a transition to be populated between the From State and the To State.
(no value/blank)	Means the PK of the configured state must be no value/blank in order for a transition to be populated between the From State and the To State.
Entry in Event PK Value Rule Column	**Interpretation**
IPK	Means the value set for the Event PK will be the value of the PK of the Interaction providing the Event Context.
<any other value>	The Event PK will be set to blank.
Entry in Transition Type Column	**Interpretation**
Simple	Means populate a simple State Transition from a single State to a single State.

Fork	Means populate a Fork Transition between states, from a single State to multiple States. Multiple rows of the Configuration Rules Table are used to indicate what “to States” are connected, all for the same Event.
Join	Means populate a Join Transition between states, from multiple States to a single State. Multiple rows of the Configuration Rules Table are used to indicate what “from States” are connected, potentially involving different Events.
Entry in Transition PK Value Rule Column	**Interpretation**
No entry needed—inferred as shown.	If the transition is not a Join, the transition unique instance is identified as the concatenation of the populated From State (including its PK value) and Event (including its PK value). In the case of a Join transition, the transition unique instance is identified by the configuration attribute Transition Name for Join.
Entry In Transition Name for Joins Column	**Interpretation**
<Join Transition Name>	For a transition that is a Join, there will be multiple configuration rule rows, with different “from States” and potentially different Events, so this Transition Name indicates which rows participate in the same Join. (Transitions that are Simple use only one row. Transitions that are Forks are multi-row but are linked by the fact they have a common Event.) Transitions that are Simple or Forks are auto-named.

3.3.2.15 Configuration Rules Set: Interface Context

During auto-population of a configured model from a pattern, the combination of an already populated Interaction-Role pair may cause auto-population of a certain related “Interface Context” classes. This occurs conditionally, based on Configuration Rules spread across four Interface Context Tables (ICT1, ICT2, ICT4, ICT5), as follows:

Configuration Rules that are rows of ICT1 and ICT2 act together for populating Input-Outputs, Interfaces, Ports, and/or Systems of Access, as follows:

1. The name of a System (Role) that must already have been populated appears in both ICT1 and ICT2.

2. The name of an Input-Output to be populated must appear in both ICT1 and ICT2 for that same System.

When the above conditions are satisfied, the same row of ICT1 specifies the Input-Output, Interface, Port, and/or System of Access to be populated, as follows:

3. The name of an Interface that the System (Role) can have.

4. The Primary Key (PK) of that Interface.

5. The name of an Input-Output (IO) for the System (Role).

6. The Primary Key (PK) of that IO.

7. When the IO flows In (as an Input) or Out (as an Output) or InOut (as an Input-Output).

8. The name of a Port marking the intersection of the System boundary with an IO.

9. The Primary Key (PK) of that Port.

10. The name of a System of Access (SOA) providing interaction access to the System.

11. The Primary Key (PK) of that SoA.

12. Whether the SoA is External to or Internal to (built into) the System.

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

ICT 1	IF SYSTEM PRESENT	THEN POPULATE INTERFACE, IO, PORT, and SOA (with PK values and parameters shown)									
	System Name	**Interface Name**	**Interface PK Value Rule**	**Input-Output Name**	**IO PK Value Rule**	**IO Direction**	**Port Name**	**Port PK Value Rule**	**SOA Name**	**SOA PK Value Rule**	**SOA Internal or External**
	<System Name>	*<Interface Name>*	*<Interface PK Rule>*	*<IO Name>*	*<IO PK Rule>*	*<IO Direction>*	*<Port Name>*	*<Port PK Rule>*	*<SoA Name>*	*<SoA PK Rule>*	*<Int or Ext>*
	<System Name>	*<Interface Name>*	*<Interface PK Rule>*	*<IO Name>*	*<IO PK Rule>*	*<IO Direction>*	*<Port Name>*	*<Port PK Rule>*	*<SoA Name>*	*<SoA PK Rule>*	*<Int or Ext>*
	Vehicle	Starter Interface	IPK	Start Request	IPK	In	Start Port	IPK	Button 5		External

ICT 2	IF INTERACTION-SYSTEM PRESENT		THEN POPULATE INPUT-OUTPUT, ARCHITECTURAL RELATIONSHIP	
	Interaction Name	**System Name**	**IO Name**	**Architectural Relationship**
	<Interaction Name>	*<Role Name>*	*<IO Name>*	*<AR Name>*
	<Interaction Name>	*<Role Name>*	*<IO Name>*	*<AR Name>*
	Operate Starter	Vehicle	Start Request	Operates

Configuration Rule Attributes

Entry in Interface PK Rule Column	Interpretation
IPK	Means the PK of the configured Interface will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured Interface will be set to blank.

Entry in IO PK Rule Column	Interpretation
IPK	Means the PK of the configured IO will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured IO will be set to blank.
Entry in IO Direction Column	**Interpretation**
In	Means the direction of the System Port will be set to In.
Out	Means the direction of the System Port will be set to Out.
InOut	Means the direction of the System Port will be set to InOut.
Entry in Port PK Rule Column	**Interpretation**
IPK	Means the PK of the configured Port will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured Port will be set to blank.
Entry In SoA PK Rule Column	**Interpretation**
IPK	Means the PK of the configured SoA will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured Soa will be set to blank.
Entry in SoA Internal or External Column	**Interpretation**
Internal	Means the SOA is understood to be part of (internal to) the system.
External	Means the SoA is understood to be outside (external to) the system.

Configuration Rules that are rows of ICT2, ICT4, and ICT5 act together for populating Architectural Relationships, and their Architectural Relationship Roles, as follows:

1. The name of an (already populated) System that is a candidate to be connected by an Architectural Relationship (also named), by virtue of its (already populated) participation in an Interaction (also named), all appear together in a configuration rule row of ICT 2.

2. The same named System and Architectural Relationship also appear together in a configuration rule row of either ICT 5 (for simple 2-way / binary relationships) or ICT 4 (for reified or n-way relationships, n>2).

When the above conditions are satisfied, details of filling the relationship roles are specified by ICT4 (for reified or n-way relationships, n>2), or ICT5 (for simple 2-way/binary relationships), as follows:

3. The name of Architectural Relationship (AR) to be populated.

4. The Primary Key (PK) of that AR.

5. For Reified ARs (ICT4), the name of the AR's Roles.

6. For Reified ARs (ICT4), the value of the PKs of the AR's Roles.

7. For Simple ARs (ICT5), the names of the Systems filling the two AR Roles.

8. For Reified ARs (ICT4), the name of the System filling each AR Role.

9. Whether the AR is (EXT?INT?)

A single configuration rule of this Configuration Rules Set is viewed as an individual row of the following Configuration Rules Table, whose vertical row order is insignificant. The upper portion of table shows the general form of the rules and the lower portion shows some examples. The Configuration Rule Attributes provide additional description of the allowable entries for the applicable columns.

Configuration Rules Table

ICT 4							
	System Name	**Architectural Relationship Name**	**AR PK Value Rule**	**AR Role Name**	**AR Role PK Value Rule**	**AR Internal or External**	**AR Complexity**
	<System Name>	*<AR Name>*	*<AR PK>*	*<AR Role Name>*	*<AR PK>*	*<Int or Ext>*	*<AR Complexity>*
	<System Name>	*<AR Name>*	*<AR PK>*	*<AR Role Name>*	*<AR PK>*	*<Int or Ext>*	*<AR Complexity>*
	Vehicle	Operates	IPK	Driver	RPK		Reified

ICT 5	**THEN POPULATE ARCH REL**		**← IF BOTH SYSTEMS POPULATED →**		**POPULATED AR DETAILS**	
	Architectural Relationship Name	**AR PK Value Rule**	**Connected System, Subject End of AR**	**Connected System, Object End of AR**	**Internal or External AR**	**AR Complexity**
	<AR Name>	*<AR PK>*	*<Subject System>*	*<Object System>*	*<Int or Ext>*	*<AR Complexity>*
	<AR Name>	*<AR PK>*	*<Subject System>*	*<Object System>*	*<Int or Ext>*	*<AR Complexity>*
	Isolates		Atmosphere	Passenger		Simple

Configuration Rule Attributes

Entry in AR PK Rule Column	**Interpretation**
IPK	Means the PK of the configured AR will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured AR will be set to blank.
Entry in AR Role PK Rule Column	**Interpretation**

IPK	Means the PK of the configured AR Role will be set equal to the PK of the Interaction in ICT2.
Blank, other values	Means the PK of the configured AR Role will be set to blank.
Entry AR Internal or External Column	**Interpretation**
Internal	Means the AR is understood to be part of (internal to) the system.
External	Means the AR is understood to be outside (external to) the system.
Entry in AR Complexity Column	**Interpretation**
Simple	Means the Architectural Relationship should be treated as a simple binary (2-Way) relationship.
Reified	Means the Architectural Relationship should be reified, with roles instantiated. Particularly suitable for n-Way relationships, n>2.

3.3.2.16 Configuration Rules Set: Pattern Attribute Couplings

During auto-population of a configured model from a pattern, the existence of an already populated attribute of certain classes (Features, Roles, Design Components, Input-Outputs, which "own" the attribute) may cause auto-population of a related Attribute Coupling. This occurs conditionally, based on the pattern's Configuration Rules:

1. The name of an Attribute shown in the rule must already have been populated.

2. And it must be an Attribute of the Owner shown in the rule, and must already have been populated.

3. And that Owner must be of the S*Metaclass type shown as Stereotype in the rule.

4. And the PK of that Owner (which is also effectively the Attribute PK) must satisfy the population rule shown.

When the above conditions are satisfied, the same configuration rule row specifies:

5. The name of the Coupling that will be populated, and linked to the Attribute listed.

6. The Direction of the Coupling, differentiating inputs to the coupling versus outputs.

Each configuration rule of this Configuration Rules Set is viewed as one row of the following Configuration Rules table, whose vertical row order is insignificant.

Note that a single Attribute Coupling will always couple two or more Attributes, but each row/rule of the table shown covers only one Attribute. Therefore, a single Attribute Coupling can be expected to appear in two or more rows of the Attribute Coupling rules. Only one of them should be for an Out direction, because one Attribute Coupling drives a single (output) Attribute, from one or more (input) Attributes.

Note that the configuration rule does not explicitly specify the PK value to be set for the Coupling to be populated, because an Attribute Coupling's PK value is always implicitly the

same as the PK value of the Attribute it drives, which is implicitly the same as the PK value of the class owning the driven Attribute.

The last rows in the table below provides an example:

Configuration Rules Table

IF ATTRIBUTE AND ITS OWNER ALREADY POPULATED (and meeting PK rule shown)				THEN POPULATE COUPLING	
Attribute Name	Owner	Stereotype	Attribute Coupling Population Rule	Coupling Name	Coupling Direction
<Attribute Name>	<Class Name>	<Metaclass>		<Coupling Name>	<Coupling Direction>
Part Number	Power Converter Assembly	Physical System		Product Characterization Coupling	in
Max Power Drain	International Power Converter	Logical System		Max Power Coupling	in
Max Drain on Mains	Power Mains Compatibility	Feature		Max Power Coupling	out

Configuration Rule Attributes

Entry in Stereotype Column	**Interpretation**
Feature	Means the listed Attribute is an Attribute of a Feature.
Role	Means the listed Attribute is an Attribute of a Role.
Physical System	Means the listed Attribute is an Attribute of a Design Component.
Input-Output	Means the listed Attribute is an Attribute of an Input-Output.
Entry in Attribute Coupling Population Rule Column	**Interpretation**
CPK=APK	Means that the listed Attribute will not be coupled to the listed Coupling unless the PK of that Attribute (which is always the same as the PK of that Attribute's Owner) is equal to the PK of Attribute Coupling (which is always the same as the PK of the coupling driven Attribute's Owner).
CPK<APK	Means that the listed Attribute will not be coupled to the listed Coupling unless the PK of that Attribute (which is always the same as the PK of that Attribute's Owner) contains (as a substring) the PK of Attribute Coupling (which is always the same as the PK of the coupling driven Attribute's Owner).
APK<CPK	Means that the listed Attribute will not be coupled to the listed Coupling unless the PK of that Attribute (which is always the same as the PK of that Attribute's Owner) is a substring of the PK of Attribute Coupling (which is always the same as the PK of the coupling driven Attribute's Owner).
ANY or blank	Means that the listed Attribute will be coupled to the listed Coupling, without consideration of their PK values.
Entry Coupling Direction Column	**Interpretation**

in	Means the Attribute listed in this rule is an input to the Coupling listed in this rule. That is, the Coupling is driven, partly or completely, by that Attribute. A Coupling may be driven by more than one Attribute.
out	Means the Attribute listed in this rule is the output of the Coupling listed in this rule. That is, the Coupling drives that Attribute. Only one Attribute can be driven by a Coupling.

3.3.2.17 Configuration Rules Set: Pattern Risk Analysis

During auto-population of a configured model from a pattern, the combination of an already populated Features, Interactions, Requirements, and Design Components may cause auto-population of a certain related "Risk Analysis" classes and relationships. This occurs conditionally, based on Configuration Rules spread over six inter-linked Risk Analysis rule sets, listed below in tabular form:

Rows of tables act together, populating Failure Modes, Counter Requirements, Failure Impacts, related parameters as to Severity, Probability, Causality, Prevention, Detection, Mitigation, and Prognostics:

1. The name of a Stakeholder Feature that must already have been populated

2. The name of an Interaction that must already have been populated

3. The identity of a Requirement Statement that must already have been populated

4. The name of a Design Component that must already have been populated

When the above conditions are satisfied, the same configuration rules specify the Failure Modes, Counter Requirements, and Failure Impacts, to be populated, as follows:

5. The name of a Failure Mode to populate.

6. The Primary Key (PK) of that Failure Mode.

7. The name of a Counter Requirement to populate.

8. The Primary Key (PK) of that Counter Requirement.

9. The name of a Failure Impact to populate.

10. The Primary Key (PK) of that Failure Impact.

11. Failure Mode Context to populate, potentially linkages to other already populated Interactions involved in the failure mode's cause, detection, prevention, mitigation, and/or prognostics.

Each configuration rule of this Configuration Rules Set is viewed as one row of the following Configuration Rules tables, whose vertical row order is insignificant. Note that the combination of the following tables is what ultimately determines population. For example, a particular populated design component may cause population of a particular related failure mode, but if there is no populated feature whose failure impacts would be invoked by that failure mode, then that failure mode is not populated. The last row in each table below provides an example:

Configuration Rules Table

FM 1	IF PRESENT		THEN POTENTIALLY POPULATE		
	Design Component	Failure Mode Population Rule	Failure Mode	Failure Mode PK Value Rule	Probability
	<DC Name>	*<DCPK Rule>*	*<Failure Mode>*	*<FMPK Rule>*	
	<DC Name>	*<DCPK Rule>*	*<Failure Mode>*	*<FMPK Rule>*	
	Power Converter Assembly	*ANY*	Regulator Failure	CRPK	0.0002
	Power Converter Assembly	*ANY*	Internal Electrical Short to Case	DCPK	0.0003

FM 2	FM-CR RULES	
	Failure Mode	Counter Requirement Name
	<Failure Mode>	*<CR Name>*
	<Failure Mode>	*<CR Name>*
	Regulator Failure	CREQ 002
	Internal Electrical Short to Case	CREQ 001

FM 3	IF PRESENT		THEN POPULATE		
	Requirement Name	Counter Requirement Population Rule	Counter Requirement Name	Counter Requirement PK Value Rule	Counter Requirement Statement
	<Req Name>	<RSPK Value Rule>	*<CR Name>*	<CR PK Value Rule>	*<CR Statement>*
	<Req Name>	<RSPK Value Rule>	*<CR Name>*	<CR PK Value Rule>	*<CR Statement>*
	REQ 005	*ANY*	CREQ 001	RSPK	The system presents a shock hazard to users when operated according to its instructions.
	REQ 006	*ANY*	CREQ 002	RSPK	The system generates Output Power to attached Electrically Powered Devices which exceeds the [Output Voltage-Power Profile].

FM 4	IF PRESENT		THEN POPULATE		
	Feature	Failure Impact Population Rule	Failure Impact	Failure Impact PK Value Rule	Severity
	<Feature>	*<Feature PK Val Rule>*	*<Failure Impact>*	*<Fail Imp PKV Rule>*	
	<Feature>	*<Feature PK Val Rule>*	*<Failure Impact>*	*<Fail Imp PKV Rule>*	
	Powered Devices Compatibility	*ANY*	Damage to Powered Device	FPK	Serious
	Safety	*ANY*	Electrical Shock		Severe
	Reliability and Durability	*ANY*	Loss of Converted Power Output	FPK	Serious

FM 5

FI-CR RULES	
Failure Impact	**Counter Requirement Name**
<Failure Impact>	*<CR Name>*
<Failure Impact>	*<CR Name>*
Electrical Shock	CREQ 001
Damage to Powered Device	CREQ 002

FM 6

IF POPULATED			IF STATES AND EVENT CONTEXT ALREADY POPULATED (both State PKs & Event Context Interaction must be satisfied)					
Failure Mode	**Interaction**	**Failure Mode Context Element Population Rule**	**FM Context Element Name**	**Causes**	**Mitigates**	**Prevents**	**Detects**	**Predicts**
<Failure Mode>	*<Interaction>*	*<IPK Rule>*	*<FM Context Element Name>*	*<0/1>*	<0/1>	<0/1>	<0/1>	<0/1>
<Failure Mode>	*<Interaction>*	*<IPK Rule>*	*<FM Context Element Name>*	*<0/1>*	<0/1>	<0/1>	<0/1>	<0/1>
Regulator Failure	Convert Electrical Power	FMPK	IC Overheat	1				
Internal Electrical Short to Case	Assemble Product	*ANY*	Insulation Damage	1				

Configuration Rule Attributes

Entry in FM 1 Failure Mode Population Rule Column	**Interpretation**
<PK value>	Means the PK of the already populated design component must equal <PK value> in order for failure model to be populated.
ANY	Means the failure mode will be populated irrespective of the PK value of the design component.
Entry in FM 1 Failure Mode PK Value Rule Column	**Interpretation**
DCPK	Means the PK of the failure mode will be set equal to the PK of the Design Component.
CRPK	Means the PK of the failure mode will be set equal to the PK of the Counter Requirement.
Blank	Means the PK of the configured IO will be set to blank.
Entry in FM 1 Probability Column	**Interpretation**
Numeric value from 0-1	Means the populated failure mode probability will be set to the value shown.
Entry in FM 3 Counter Requirement Population Rule Column	**Interpretation**

<PK value>	Means the PK of the already populated requirement statement must equal <PK value> in order for the counter requirement statement to be populated.
ANY	Means the counter requirement statement will be populated irrespective of the PK value of the requirement statement.
Entry in FM 3 Counter Requirement Primary Key Value Rule Column	**Interpretation**
RSPK	Means the PK of the configured counter requirement will be set equal to the PK of the requirement statement.
Blank	Means the PK of the configured counter requirement will be set to blank.
Entry in FM 4 Failure Impact Population Rule Column	**Interpretation**
<PK value>	Means the PK of the already populated feature must equal <PK value> in order for the failure impact to be populated.
ANY	Means the failure impact will be populated irrespective of the PK value of the feature.
Entry in FM 4 Failure Impact Primary Key Value Rule Column	**Interpretation**
FPK	Means the PK of the configured failure impact will be set equal to the PK of the feature.
Blank	Means the PK of the configured failure impact will be set to blank.
Entry in FM 6 Failure Mode Context Element Population Rule Column	**Interpretation**
<PK value>	Means the PK of the already populated interaction (providing an aspect of FM context) must equal <PK value> in order for the failure mode context link from the failure mode to the interaction to be populated.
ANY	Means the failure mode context link from the failure mode to the interaction will be populated irrespective of the PK value of the interaction.
Entry in FM 6 Causes Column	**Interpretation**
1	Means populate linkage from failure mode to interaction shown, indicating the interaction helps to cause the failure mode.
Blank, 0, or otherwise	Means no such linkage is populated by this rule.
Entry in FM 6 Mitigates Column	**Interpretation**
1	Means populate linkage from failure mode to interaction shown, indicating the interaction helps to mitigate occurrence of the failure mode.
Blank, 0, or otherwise	Means no such linkage is populated by this rule.
Entry in FM 6 Prevents Column	**Interpretation**

1	Means populate linkage from failure mode to interaction shown, indicating the interaction helps to prevent occurrence of the failure mode.
Blank, 0, or otherwise	Means no such linkage is populated by this rule.
Entry in FM 6 Detects Column	**Interpretation**
1	Means populate linkage from failure mode to interaction shown, indicating the interaction helps to detect occurrence of the failure mode.
Blank, 0, or otherwise	Means no such linkage is populated by this rule.
Entry in FM 6 Predicts Column	**Interpretation**
1	Means populate linkage from failure mode to interaction shown, indicating the interaction helps to predict occurrence of the failure mode (as in prognostics).
Blank, 0, or otherwise	Means no such linkage is populated by this rule.