Systematica® Metamodel

Metamodel Version 8.0

04/07/2022

Systematica®

Do more with less

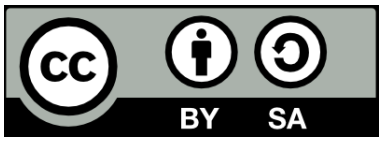

TABLE OF CONTENTS

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

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.
- Modeled 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.
- Modeled Relationship View: The formal view that depicts the abstract classes and relationships with respect to modeled 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 Modeled 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.
- 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.

- 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.

- 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.

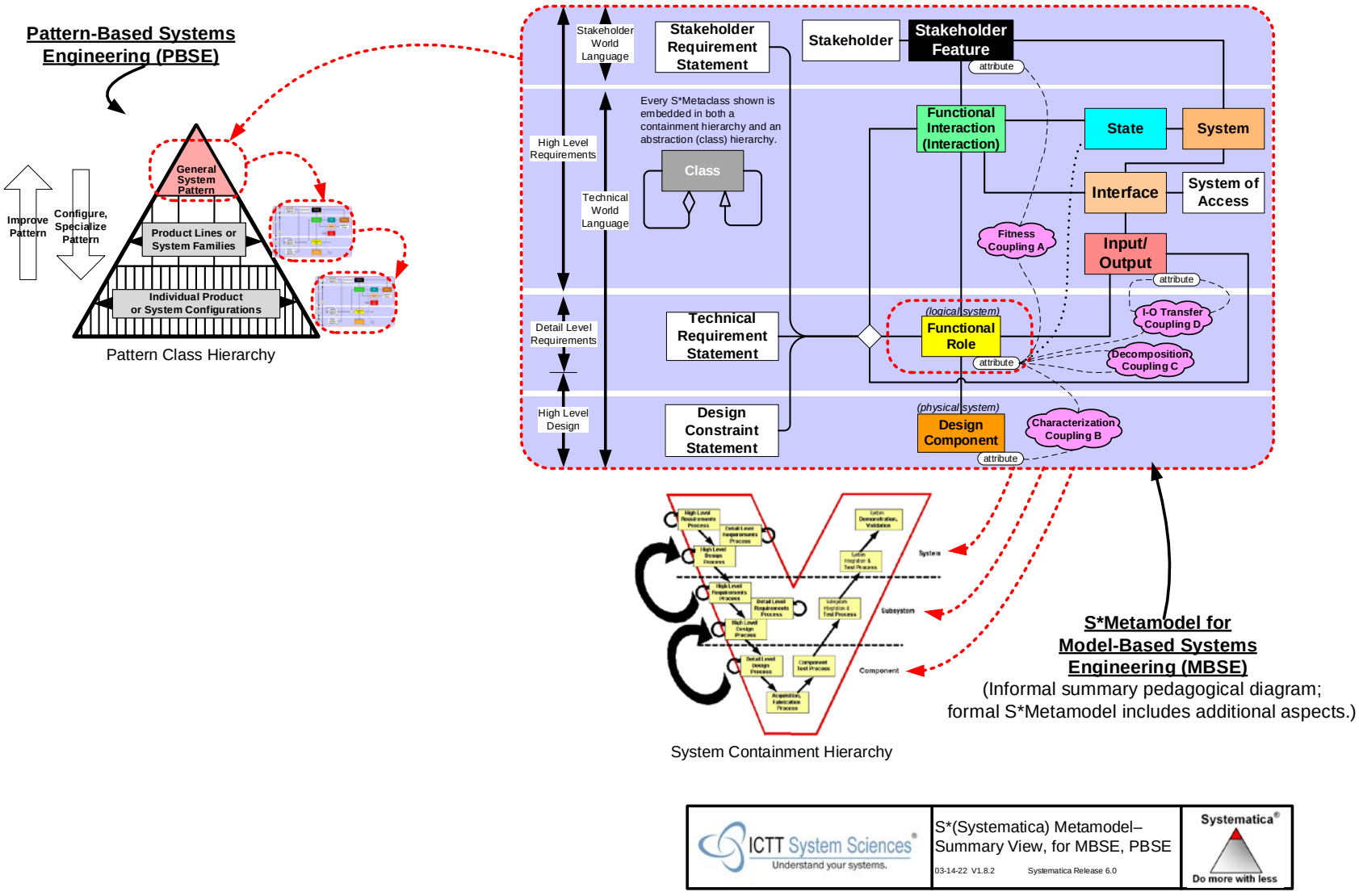

Figure 1: Summary Metamodel

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.

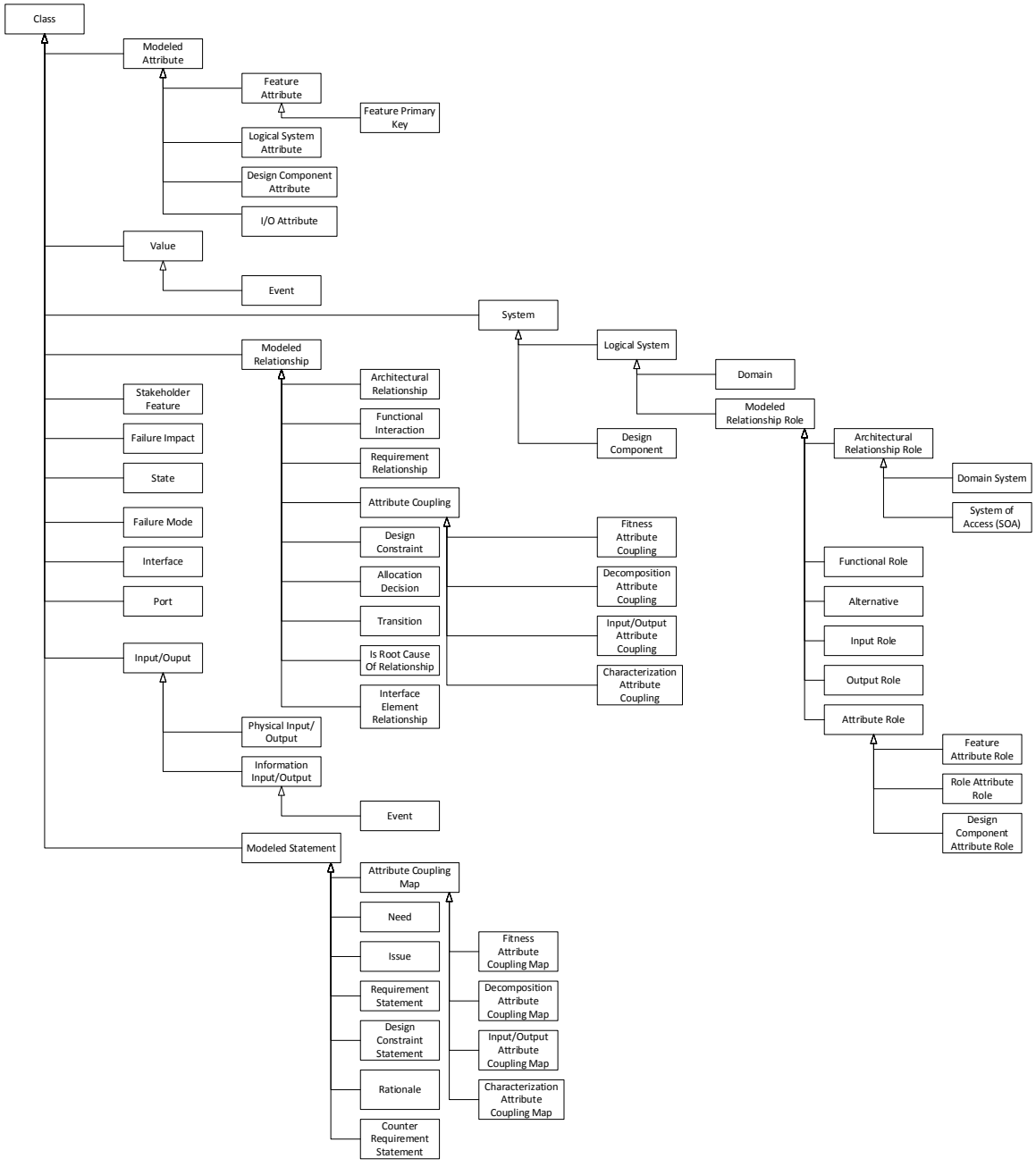

Figure 2: Class Hierarchy View

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.

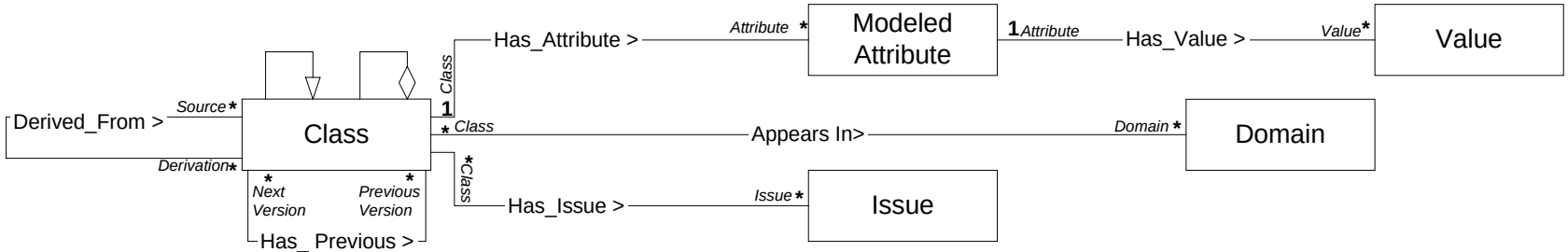

Figure 3: General Class View

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.

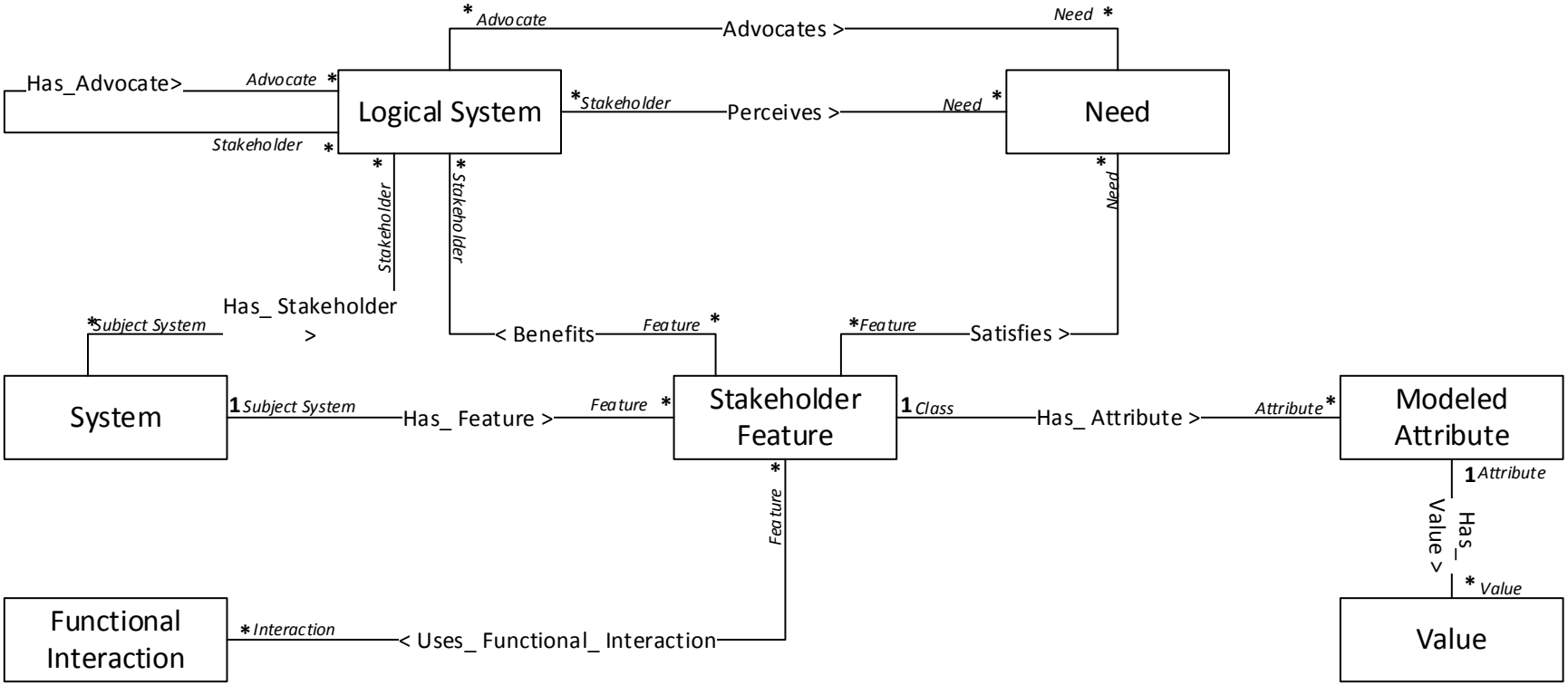

Figure 4: Feature Framework View

2.5 Modeled Relationship Views View

The metamodel defines abstract concepts such as a Modeled Relationship, its Modeled 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 Figure 5. This view does not impact the metamodel, but it does help relate each of the specialized relationship views to each other.

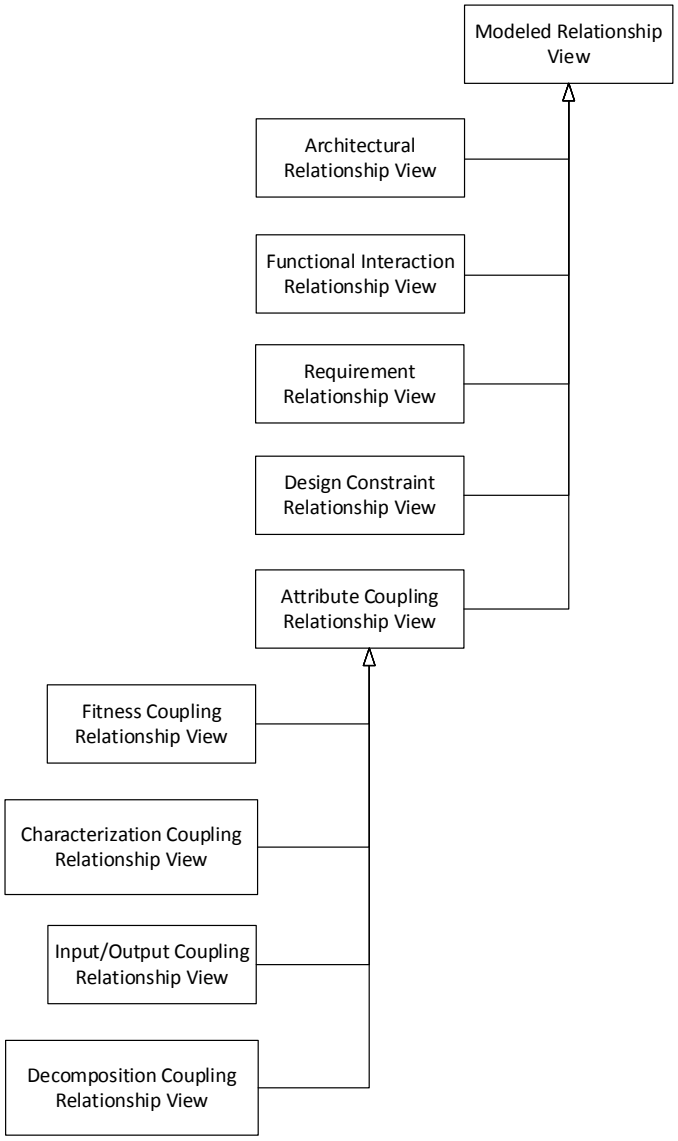

Figure 5: Modeled Relationship Views View

2.6 Modeled Relationship View

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

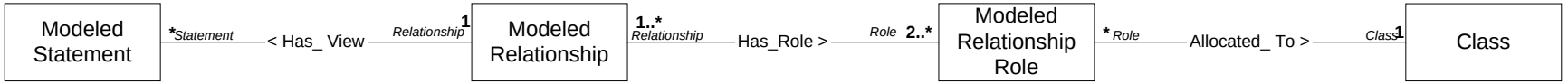

Figure 6: Modeled Relationship View

2.7 Architectural Relationship View

Figure 7 specializes the Modeled 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.

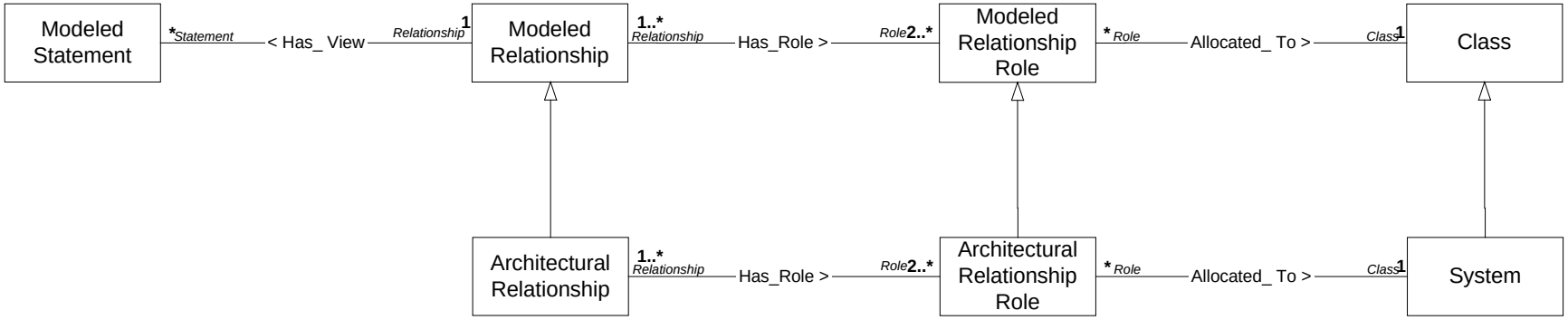

Figure 7: Architectural Relationship View

2.8 Functional Interaction View

The Functional Interaction View defines the Functional Interaction and its related classes as subclasses of the Modeled Relationship View classes. The Functional Interaction differs from the other relationships because it indirectly allocates its roles to classes through Allocation Decisions. Because Allocation Decision is itself a Modeled Relationship, the pattern holds.

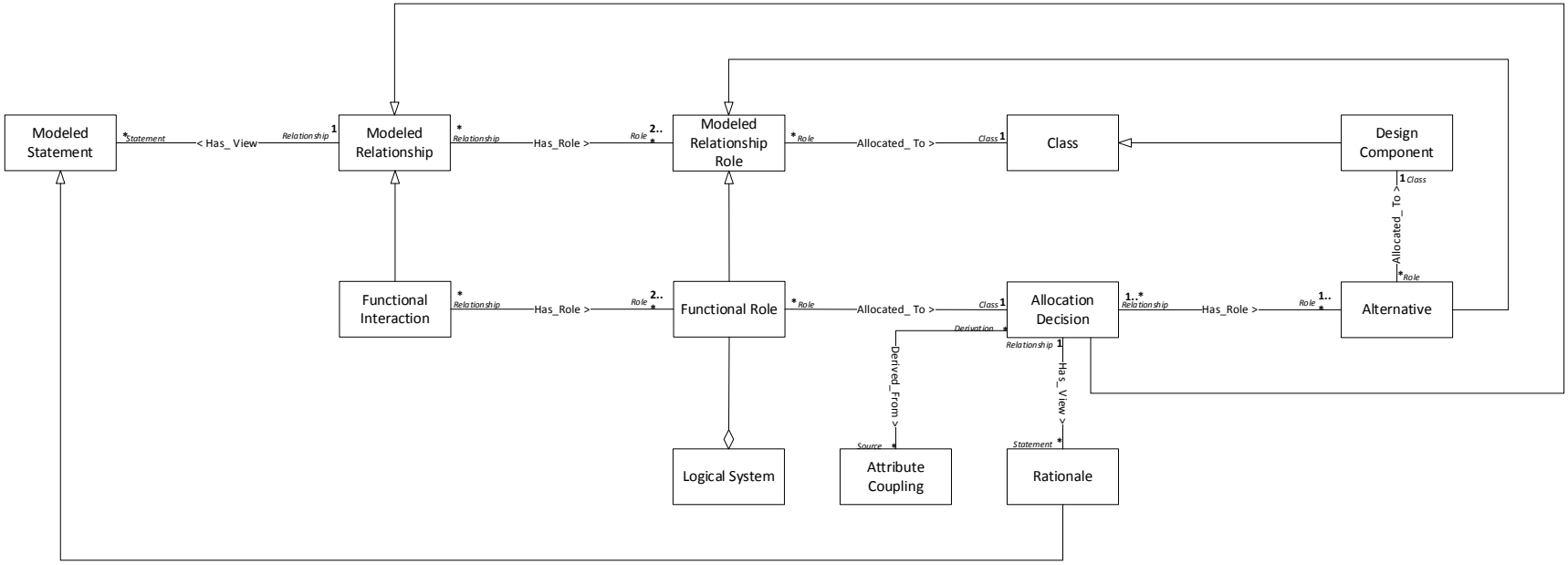

Figure 8: Functional Interaction View

2.9 Requirement Relationship View

Figure 9 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.

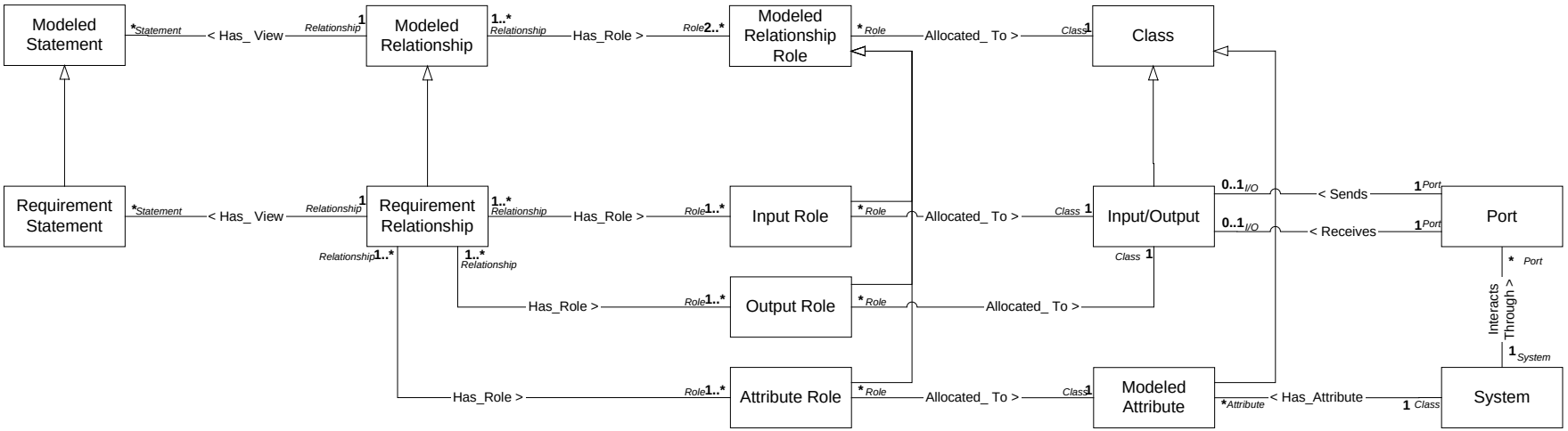

Figure 9: Requirement Relationship View

2.10 Design Constraint View

The Design Constraint View in Figure 10 defines the Design Constraint and Design Constraint Statements as a specialization of the Modeled Relationship pattern that modifies a System's Design Component Subsystem.

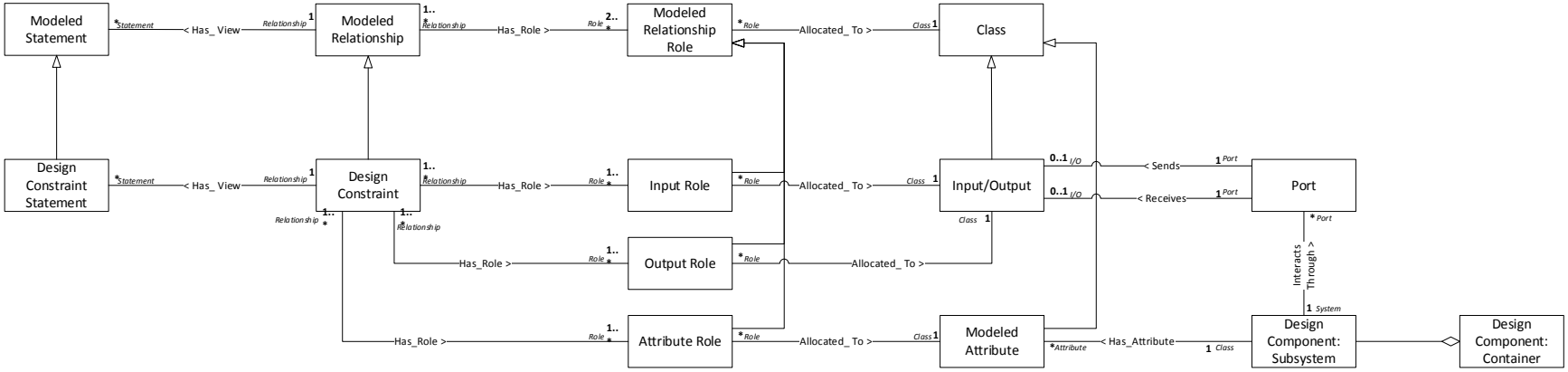

Figure 10: Design Constraint View

2.11 Attribute Coupling View

The Modeled Relationship View is specialized into a pattern that relates attributes in Figure 11. Attributes are coupled together with Attribute Coupling Maps as prose, mathematical equations, etc. to describe those relationships.

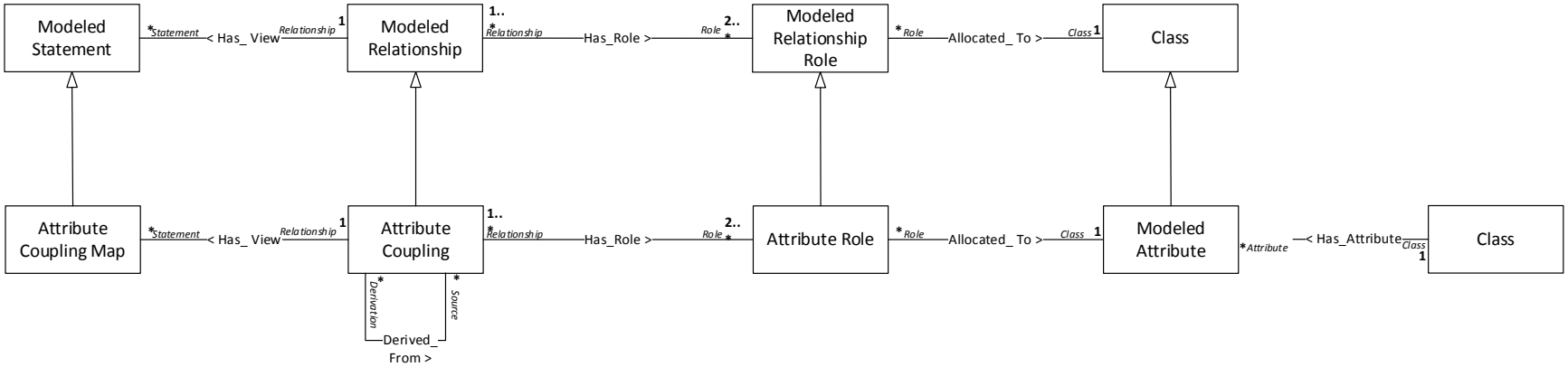

Figure 11: Attribute Coupling View

2.12 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.

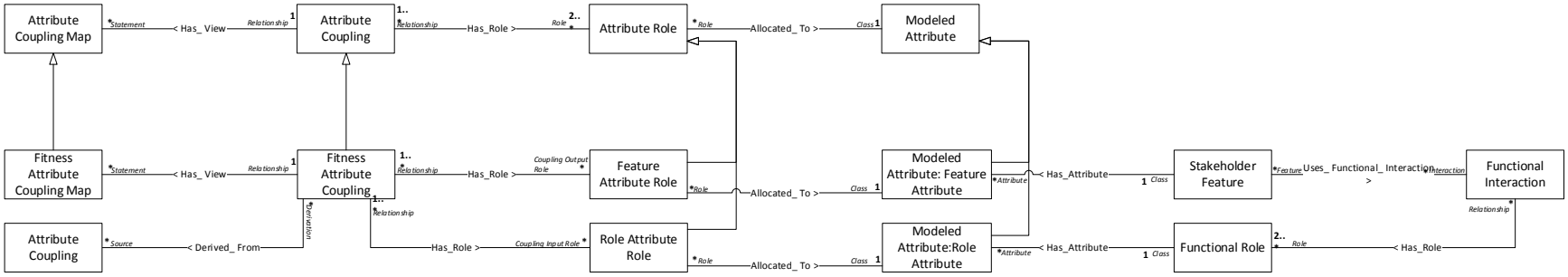

Figure 12: Fitness Coupling View

2.13 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.

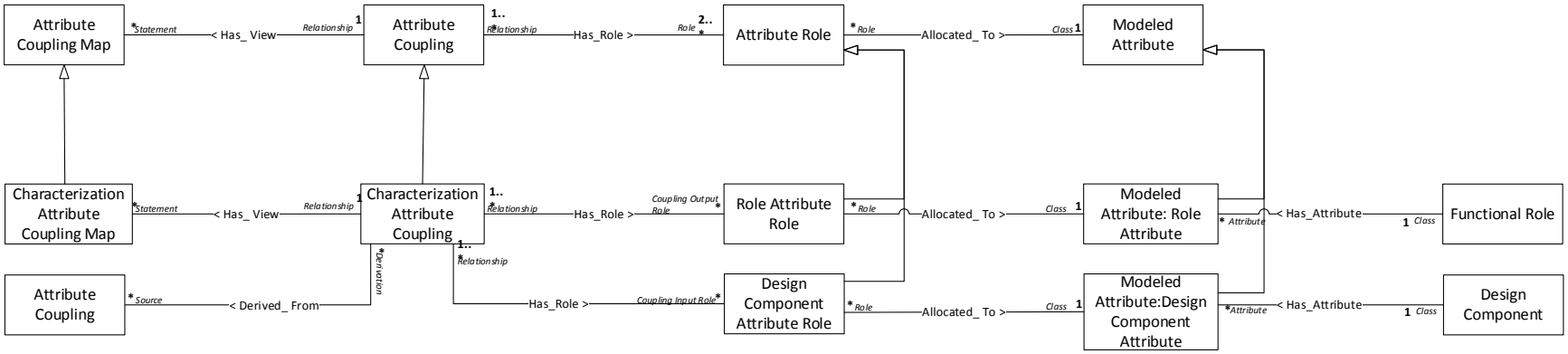

Figure 13: Characterization Coupling View

2.14 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.

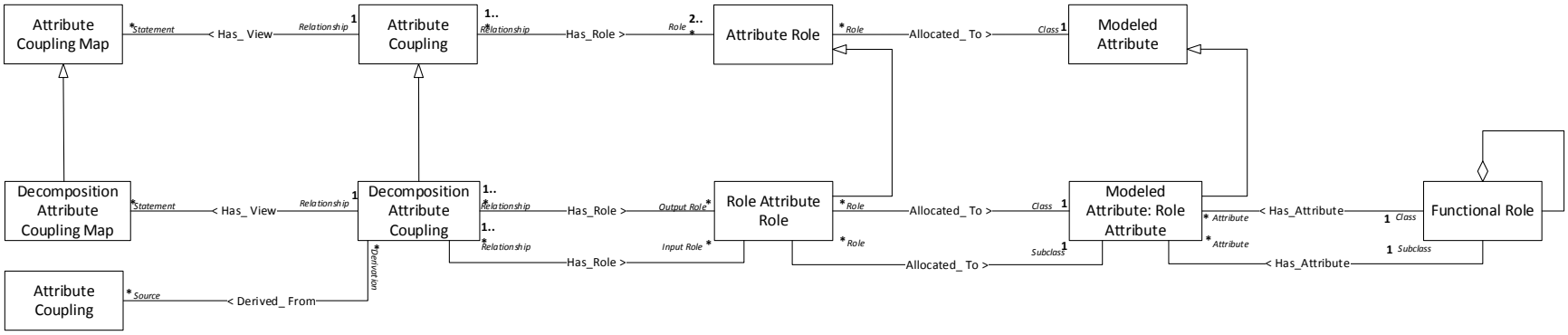

Figure 14: Decomposition Coupling View

2.15 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.

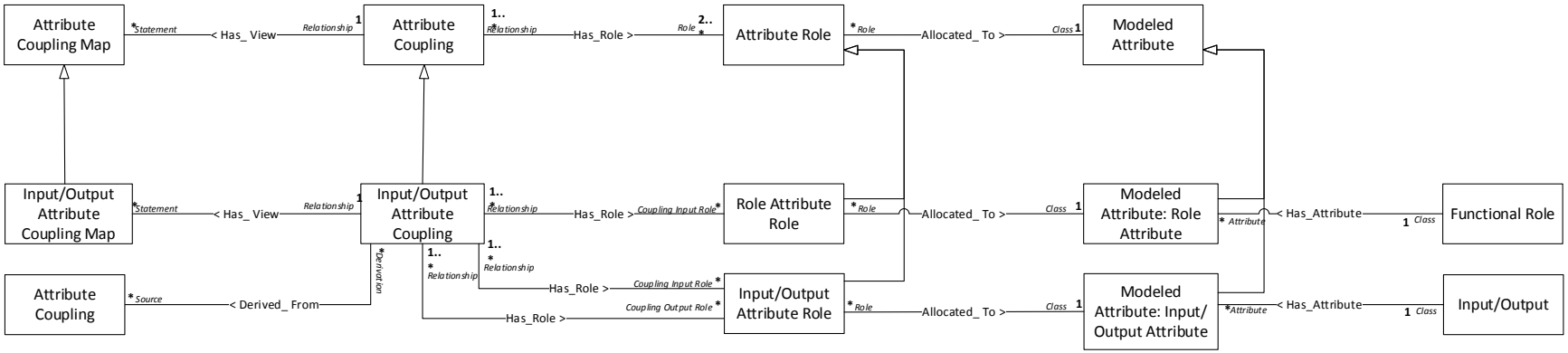

Figure 15: Input/Output Coupling View

2.16 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.

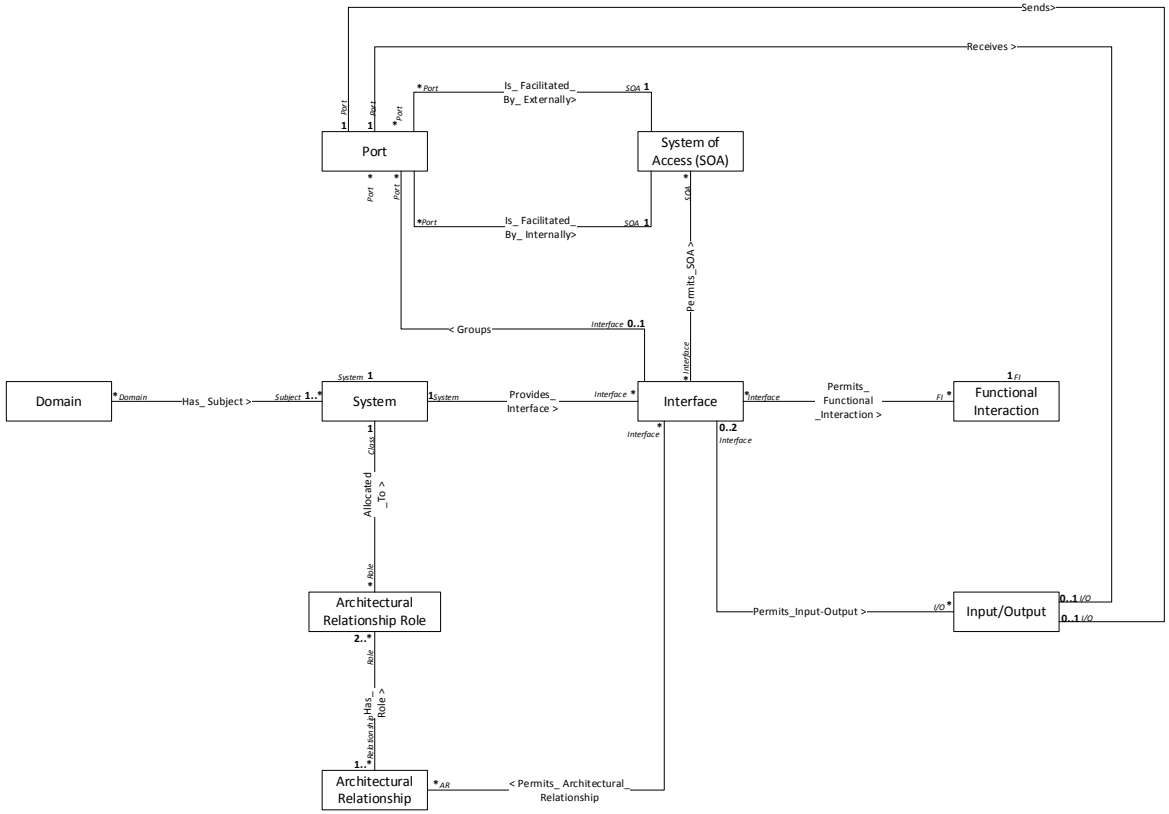

Figure 16: Domain Analysis View

2.17 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.

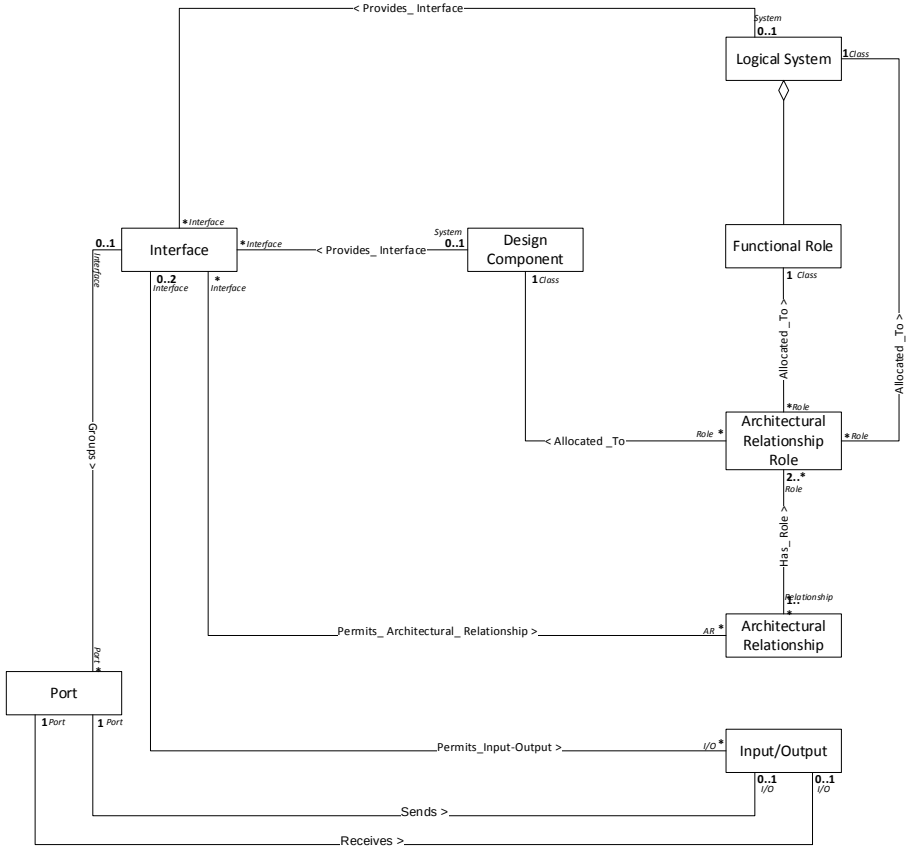

Figure 17: Logical Architecture View

2.18 State Analysis View

Figure 18 depicts the classes and relationships modeled to define a system's dynamic behavior using classes such as States, Events, and Functional Interactions.

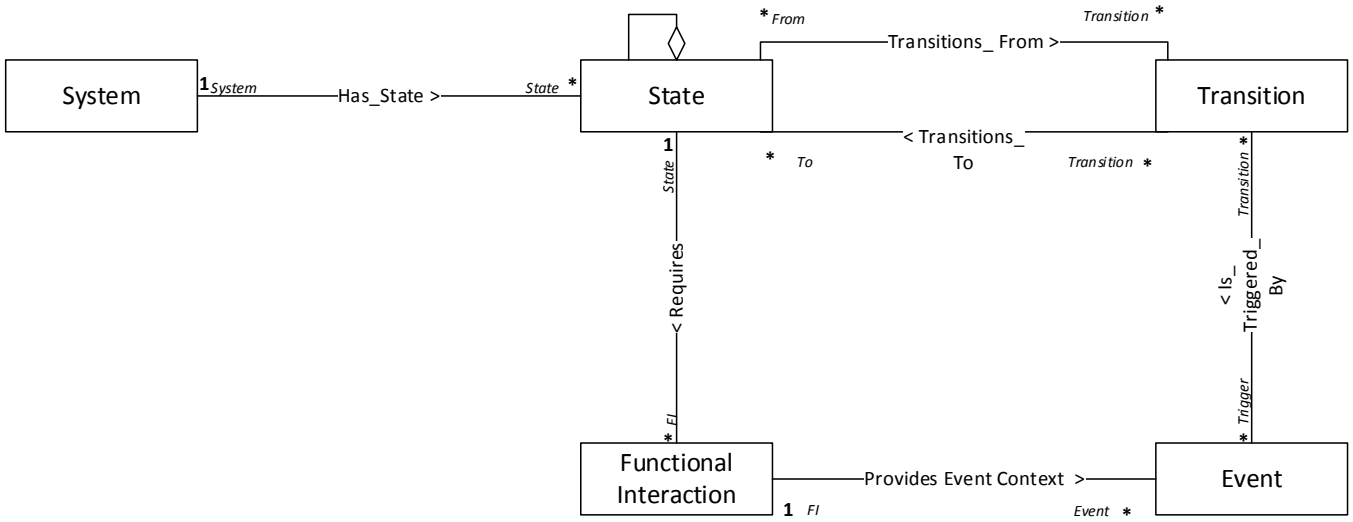

Figure 18: State Analysis View

2.19 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.

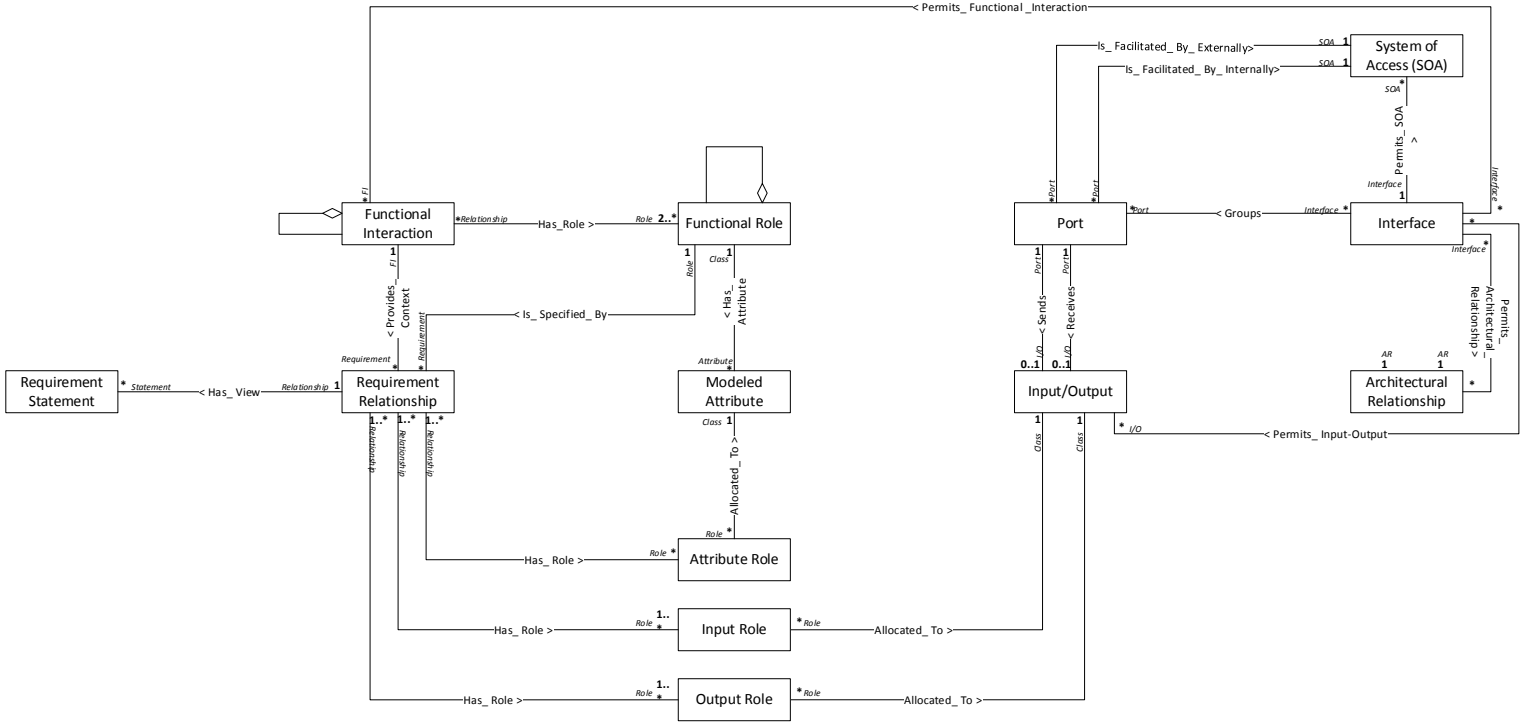

Figure 19: Detail Requirements View

2.20 High Level Design View

The High-Level Design View, pictured in Figure 20, 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.

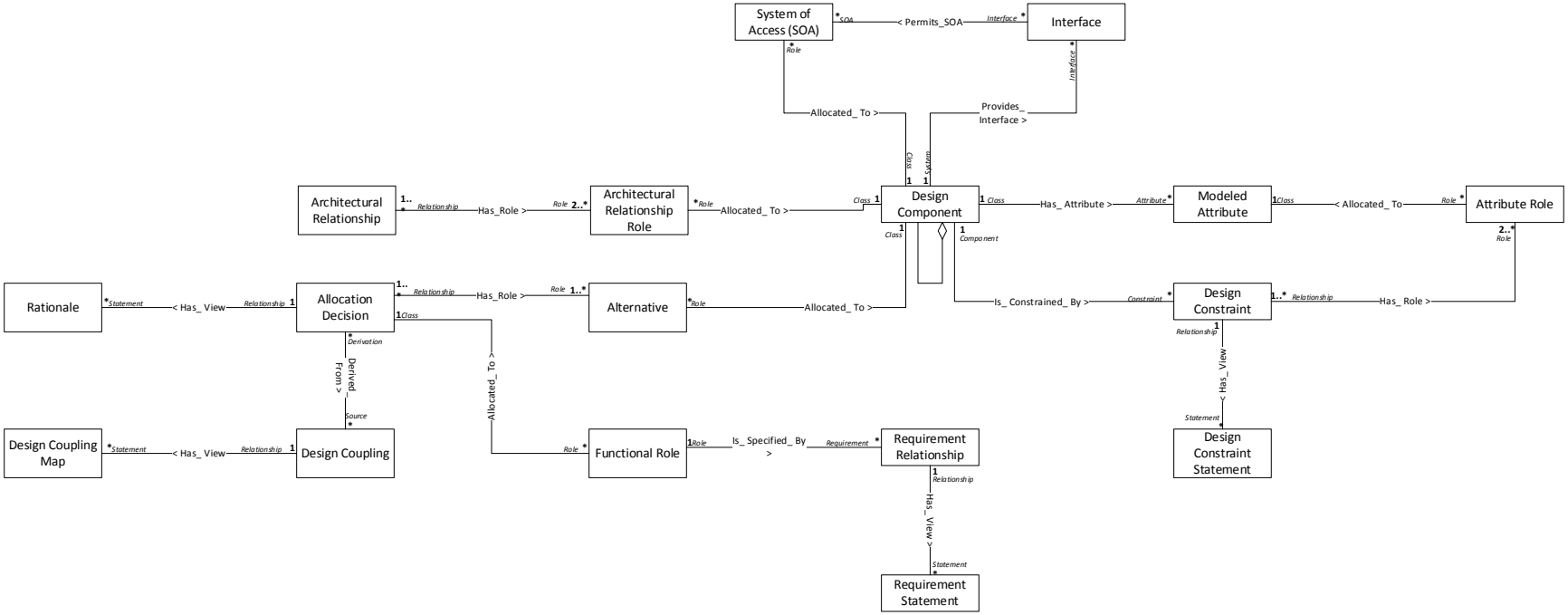

Figure 20: High Level Design View

2.21 Interface Context View

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

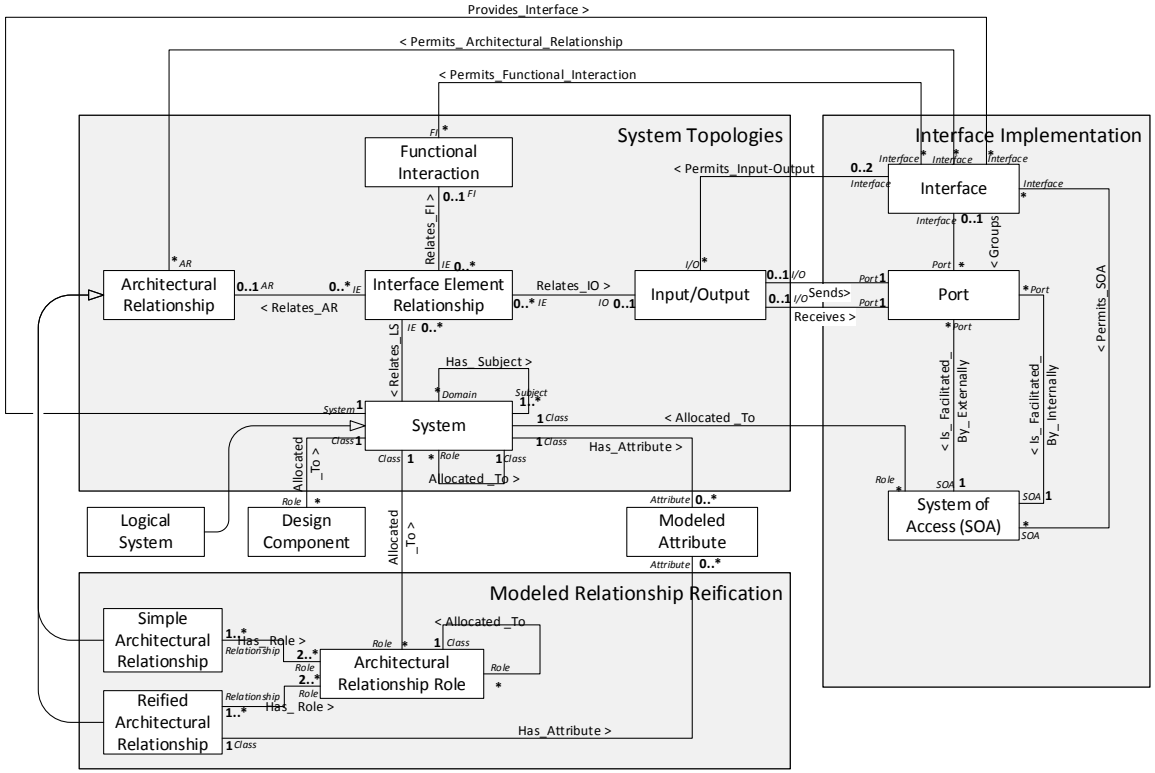

Figure 21: Interface Context View

2.22 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 22, 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. (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.

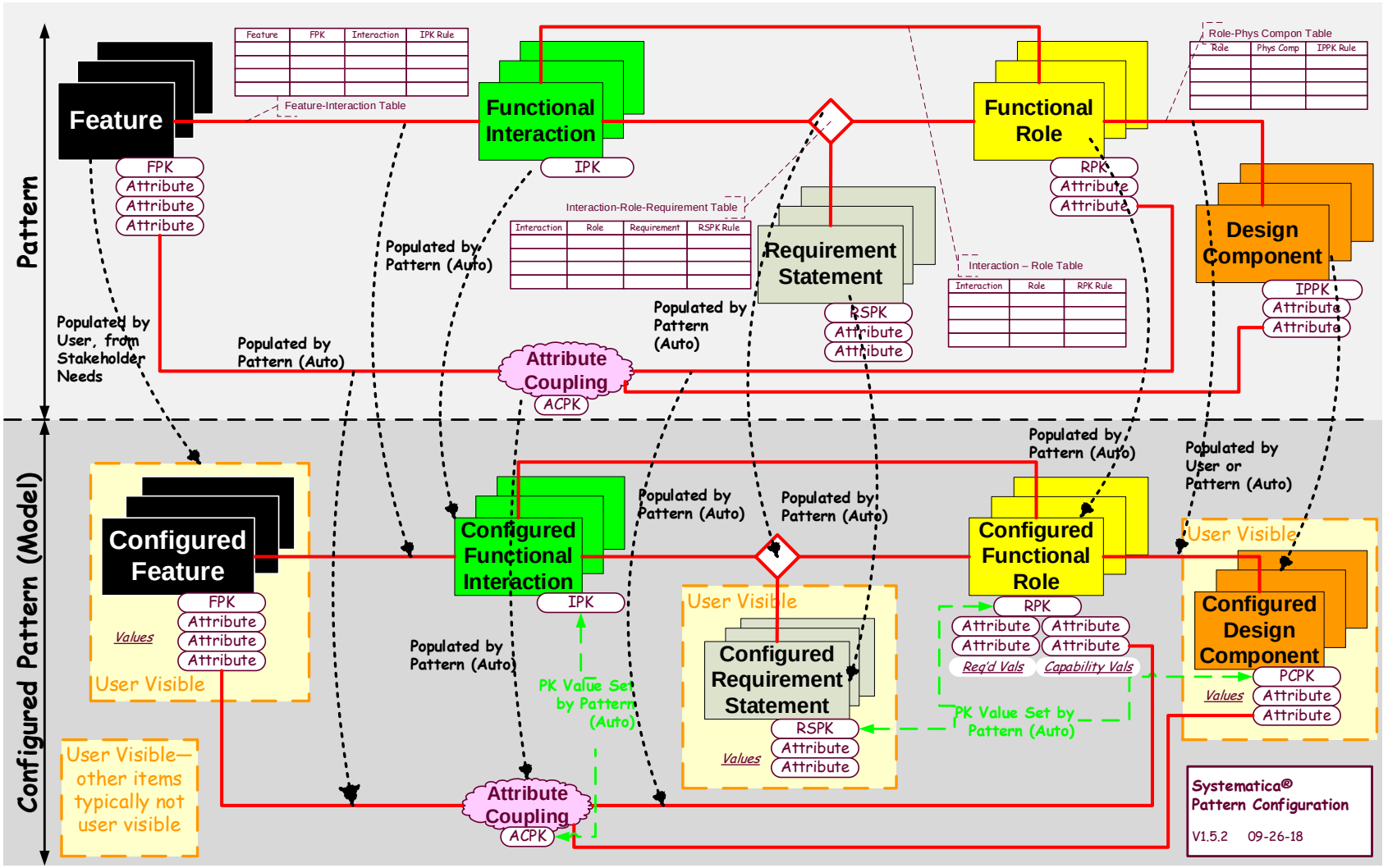

Figure 22: Summary Pattern Configuration View

	POPULATED METACLASSES ("THEN")																						
TRIGGERING METACLASSES ("IF")	Feature	Interaction	Role	Design Component	Requirement Statement	State	Interface	Architectural Relationship	Input/Output	Port	System of Access	Counter Requirement Statement	Failure Mode	Feature Impact	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						■																
Interface							■															
Architectural Relationship								■														
Input/Output									■									■				
Port										■												
System of Access											■											
Counter Requirement Statement												■										
Failure Mode													■									
Feature Impact														■								
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
Interaction	Feature	Uses Functional Interaction
Role	Interaction	Has Role
Design Component	Role	Allocated To
Requirement Statement	Interaction + Role	Requirement Relationship
State	Interaction	Requires
Interface	Interaction + Role	Interface Element Relationship
Architectural Relationship	Interaction + Role	Interface Element Relationship
Input/Output	Interaction + Role	Interface Element Relationship
Port	Interaction + Role	Interface Element Relationship
System of Access	Interaction + Role	Interface Element Relationship
Fitness Attribute Coupling	Feature Attribute	Feature Attribute Role
Decomposition Attribute Coupling	Role Attribute	Role Attribute Role
Characterization Attribute Coupling	Role Attribute	Role Attribute Role
IO Attribute Coupling	IO Attribute	IO Attribute Role
Counter Requirement Statement	Requirement Statement	Replaces
Failure Mode	Design Component	Abnormal State Of
Feature Impact	Feature	Impacts Feature
Feature Attribute	Feature	N/A
Role Attribute	Role	N/A
Input/Output Attribute	Input/Output	N/A
Design Component Attribute	Design Component	N/A

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

2.23 Risk Analysis View

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

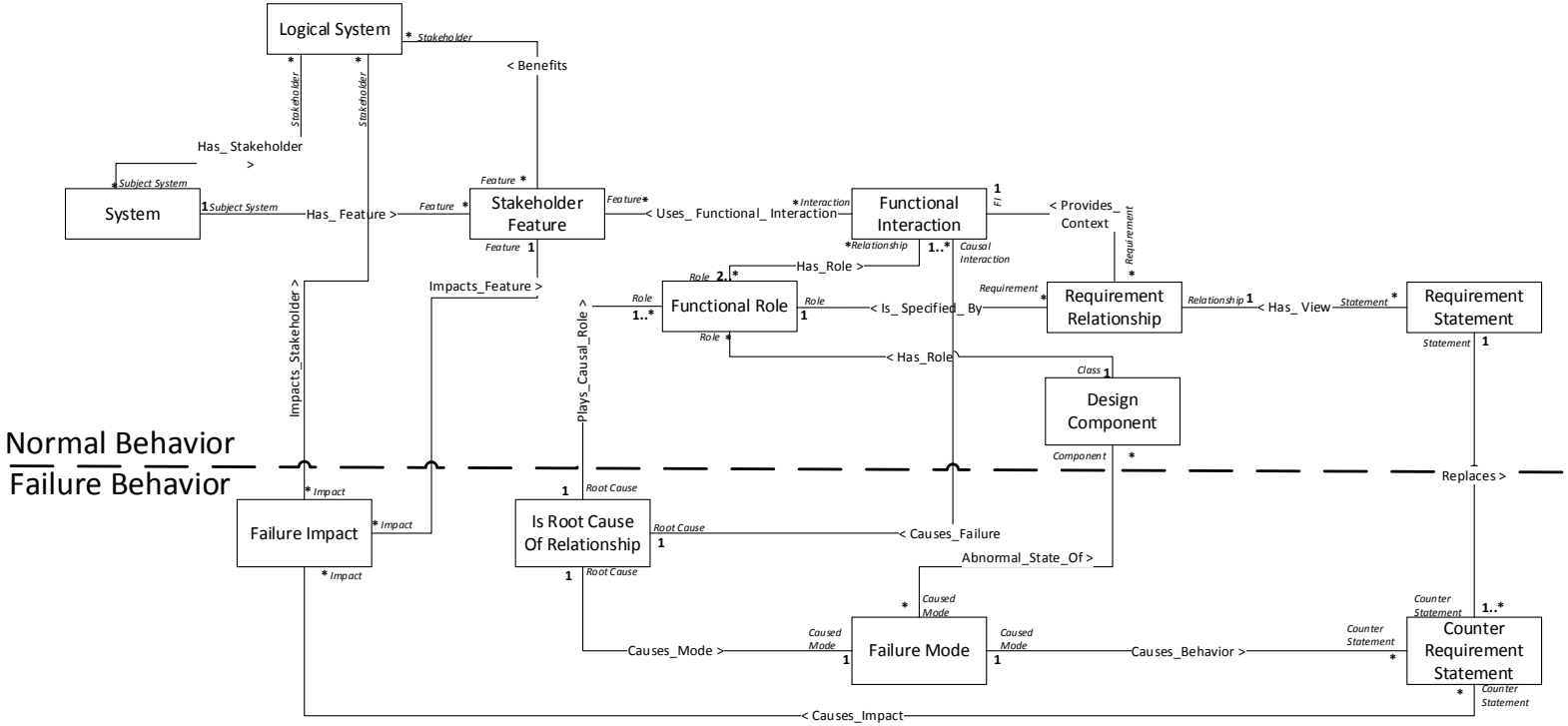

Figure 23: Risk Analysis View

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.

3.1.1 Allocation Decision

An Allocation Decision is the relationship between a Functional Role and one or more Design Components to which the Functional Role's performance may be allocated. It is the point at which an allocation analysis and decision occurs. Allocation Decisions may reference Rationales and score Alternatives, which are the roles the Design Components play in an Allocation Decision.

3.1.1.1 Aliases

- None

3.1.1.2 Relationships

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

3.1.1.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.1.4 Specific Attributes

- None

3.1.1.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 17: Logical Architecture View
- Figure 20: High Level Design View

3.1.2 Alternative

An Alternative is the role a Design Component plays in an Allocation Decision.

3.1.2.1 Aliases

- None

3.1.2.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.2.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.2.4 Specific Attributes

- Allocated
- Rank
- Score

3.1.2.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 17: Logical Architecture View
- Figure 20: High Level Design View

3.1.3 Architectural Relationship

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

3.1.3.1 Aliases

- None

3.1.3.2 Relationships

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

3.1.3.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.3.4 Specific Attributes

- None

3.1.3.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 7: Architectural Relationship View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View
- Figure 21: Interface Context View

3.1.4 Architectural Relationship Role

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

3.1.4.1 Aliases

- None

3.1.4.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.4.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.4.4 Specific Attributes

- None

3.1.4.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 7: Architectural Relationship View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 20: High Level Design View
- Figure 21: Interface Context View

3.1.5 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.

3.1.5.1 Aliases

- Parametric Coupling

3.1.5.2 Relationships

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

3.1.5.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.5.4 Specific Attributes

- None

3.1.5.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View

3.1.6 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.

3.1.6.1 Aliases

- Parametric Coupling

3.1.6.2 Relationships

- Has View
- Is a Type of

3.1.6.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.6.4 Specific Attributes

- Reference

3.1.6.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View

3.1.7 Attribute Role

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

3.1.7.1 Aliases

- None

3.1.7.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.7.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.7.4 Specific Attributes

- None

3.1.7.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.1.8 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.

3.1.8.1 Aliases

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

3.1.8.2 Relationships

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

3.1.8.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.8.4 Specific Attributes

- None

3.1.8.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 13: Characterization Coupling View
- Figure 20: High Level Design View

3.1.9 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.

3.1.9.1 Aliases

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

- Parametric Coupling

3.1.9.2 Relationships

- Has View
- Is a Type of

3.1.9.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.9.4 Specific Attributes

- Reference

3.1.9.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 13: Characterization Coupling View
- Figure 20: High Level Design View

3.1.10 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.

3.1.10.1 Aliases

- Entity

3.1.10.2 Relationships

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

3.1.10.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.10.4 Specific Attributes

- None

3.1.10.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 6: Modeled Relationship View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View

3.1.11 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.

3.1.11.1 Aliases

- None

3.1.11.2 Relationships

- Causes Behavior
- Causes Impact
- Replaces

3.1.11.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.11.4 Specific Attributes

- None

3.1.11.5 Metamodel View References

- Figure 23: Risk Analysis View

3.1.12 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.

3.1.12.1 Aliases

- Role-Role Coupling
- Parametric Coupling

3.1.12.2 Relationships

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

3.1.12.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.12.4 Specific Attributes

- Reference

3.1.12.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 14: Decomposition Coupling View

3.1.13 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.

3.1.13.1 Aliases

- Role-Role Coupling Map

3.1.13.2 Relationships

- Has View
- Is a Type of

3.1.13.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.13.4 Specific Attributes

- Reference

3.1.13.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 14: Decomposition Coupling View

3.1.14 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.

3.1.14.1 Aliases

- Physical System

3.1.14.2 Relationships

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

3.1.14.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.14.4 Specific Attributes

- None

3.1.14.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 10: Design Constraint View
- Figure 13: Characterization Coupling View
- Figure 17: Logical Architecture View
- Figure 20: High Level Design View

3.1.15 Design Component Attribute

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

3.1.15.1 Aliases

- Physical System Attribute

3.1.15.2 Relationships

- Has Value

3.1.15.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.15.4 Specific Attributes

- None

3.1.15.5 Metamodel View References

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

3.1.16 Design Component Attribute Role

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

3.1.16.1 Aliases

- None

3.1.16.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.16.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.16.4 Specific Attributes

- None

3.1.16.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 13: Characterization Coupling View

3.1.17 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.

3.1.17.1 Aliases

- None

3.1.17.2 Relationships

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

3.1.17.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.17.4 Specific Attributes

- None

3.1.17.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 10: Design Constraint View
- Figure 20: High Level Design View

3.1.18 Design Constraint Statement

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

3.1.18.1 Aliases

- None

3.1.18.2 Relationships

- Has View
- Is a Type of

3.1.18.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.18.4 Specific Attributes

- None

3.1.18.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 10: Design Constraint View
- Figure 20: High Level Design View

3.1.19 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.

3.1.19.1 Aliases

- System Context
- Context of Use
- System Environment

3.1.19.2 Relationships

- Appears In
- Has Subject
- Is a Type of

3.1.19.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.19.4 Specific Attributes

- None

3.1.19.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View

3.1.20 Domain System

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

3.1.20.1 Aliases

- None

3.1.20.2 Relationships

- Is a Type of

3.1.20.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.20.4 Specific Attributes

- None

3.1.20.5 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.21 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.

3.1.21.1 Aliases

- None

3.1.21.2 Relationships

- Is a Type of
- Is Triggered By

3.1.21.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.21.4 Specific Attributes

- None

3.1.21.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 18: State Analysis View

3.1.22 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.

3.1.22.1 Aliases

- None

3.1.22.2 Relationships

- Causes Impact
- Impacts Feature
- Impacts Stakeholder

3.1.22.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.22.4 Specific Attributes

- None

3.1.22.5 Metamodel View References

- Figure 23: Risk Analysis View

3.1.23 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.

3.1.23.1 Aliases

- None

3.1.23.2 Relationships

- Abnormal State Of
- Causes Behavior
- Causes Mode

3.1.23.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.23.4 Specific Attributes

- None

3.1.23.5 Metamodel View References

- Figure 23: Risk Analysis View

3.1.24 Feature Attribute

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

3.1.24.1 Aliases

- None

3.1.24.2 Relationships

- Has Value

3.1.24.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.24.4 Specific Attributes

- None

3.1.24.5 Metamodel View References

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

3.1.25 Feature Attribute Role

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

3.1.25.1 Aliases

- None

3.1.25.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.25.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.25.4 Specific Attributes

- Fitness Attribute Coupling PK Value Rule

- Fitness Attribute Coupling Population Rule

3.1.25.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Fitness Coupling View

3.1.26 Feature Primary Key Attribute

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

3.1.26.1 Aliases

- None

3.1.26.2 Relationships

- Has Value

3.1.26.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.26.4 Specific Attributes

- None

3.1.26.5 Metamodel View References

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

3.1.27 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.

3.1.27.1 Aliases

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

3.1.27.2 Relationships

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

3.1.27.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.27.4 Specific Attributes

- None

3.1.27.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Fitness Coupling View

3.1.28 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.

3.1.28.1 Aliases

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

3.1.28.2 Relationships

- Has View
- Is a Type of

3.1.28.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.28.4 Specific Attributes

- Reference

3.1.28.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Fitness Coupling View

3.1.29 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.

3.1.29.1 Aliases

- Function (Deprecated)
- Interaction

3.1.29.2 Relationships

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

3.1.29.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.29.4 Specific Attributes

- None

3.1.29.5 Metamodel View References

- Figure 1: Summary Metamodel

- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 8: Functional Interaction View
- Figure 12: Fitness Coupling View
- Figure 16: Domain Analysis View
- Figure 18: State Analysis View
- Figure 19: Detail Requirements View
- Figure 23: Risk Analysis View

3.1.30 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.

3.1.30.1 Aliases

- Function
- Logical System
- Role

3.1.30.2 Relationships

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

3.1.30.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.30.4 Specific Attributes

- None

3.1.30.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View
- Figure 23: Risk Analysis View

3.1.31 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.

3.1.31.1 Aliases

- Information View (Deprecated)

3.1.31.2 Relationships

- Is a Type of

3.1.31.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.31.4 Specific Attributes

- None

3.1.31.5 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.32 Input/Output

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

3.1.32.1 Aliases

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

3.1.32.2 Relationships

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

3.1.32.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.32.4 Specific Attributes

- None

3.1.32.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View

3.1.33 I/O Attribute

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

3.1.33.1 Aliases

- None

3.1.33.2 Relationships

- Has Value

3.1.33.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.33.4 Specific Attributes

- None

3.1.33.5 Metamodel View References

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

3.1.34 I/O Attribute Role

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

3.1.34.1 Aliases

- None

3.1.34.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.34.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.34.4 Specific Attributes

- I/O Attribute Coupling PK Value Rule
- I/O Attribute Coupling Population Rule

3.1.34.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Input/Output Coupling View

3.1.35 I/O Attribute Coupling

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

3.1.35.1 Aliases

- Parametric Coupling

3.1.35.2 Relationships

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

3.1.35.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.35.4 Specific Attributes

- None

3.1.35.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Input/Output Coupling View

3.1.36 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.

3.1.36.1 Aliases

- Parametric Coupling

3.1.36.2 Relationships

- Has View

- Is a Type of

3.1.36.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.36.4 Specific Attributes

- Reference

3.1.36.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 15: Input/Output Coupling View

3.1.37 Input Role

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

3.1.37.1 Aliases

- None

3.1.37.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.37.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.37.4 Specific Attributes

- None

3.1.37.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 19: Detail Requirements View

3.1.38 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.

3.1.38.1 Aliases

- None

3.1.38.2 Relationships

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

3.1.38.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.38.4 Specific Attributes

- None

3.1.38.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.1.39 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.

3.1.39.1 Aliases

- None

3.1.39.2 Relationships

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

3.1.39.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.39.4 Specific Attributes

- IPK Conditional Configuration Rule
- RPK Conditional Configuration Rule
- ARPK Population Configuration Rule
- IOPK Population Configuration Rule
- IFCPK Population Configuration Rule
- PPK Population Configuration Rule
- SOAPK Population Configuration Rule
- IPK Conditional Configuration Rule
- IPK Conditional Configuration Rule

3.1.39.5 Metamodel View References

- Figure 21: Interface Context View

3.1.40 Is Root Cause Of Relationship

An Is Root Cause Of Relationship is a reified 3-way relationship linking Functional Roles, Interactions, and Failure Modes in a Risk Analysis model.

3.1.40.1 Aliases

- None

3.1.40.2 Relationships

- Causes Failure
- Causes Mode
- Plays Causal Role

3.1.40.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.40.4 Specific Attributes

- None

3.1.40.5 Metamodel View References

- Figure 23: Risk Analysis View

3.1.41 Issue

An Issue is statement related to the properties of a Class that may indicate a need to change its model.

3.1.41.1 Aliases

- Action Item
- Open Issue

3.1.41.2 Relationships

- Has Issue
- Is a Type of

3.1.41.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.41.4 Specific Attributes

- None

3.1.41.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View

3.1.42 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.

3.1.42.1 Aliases

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

3.1.42.2 Relationships

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

3.1.42.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.42.4 Specific Attributes

- None

3.1.42.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 8: Functional Interaction View
- Figure 17: Logical Architecture View

3.1.43 Logical System Attribute

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

3.1.43.1 Aliases

- None

3.1.43.2 Relationships

- Has Value

3.1.43.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.43.4 Specific Attributes

- None

3.1.43.5 Metamodel View References

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

3.1.44 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.

3.1.44.1 Aliases

- Attribute

- Property
- Parameter
- Variable

3.1.44.2 Relationships

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

3.1.44.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.44.4 Specific Attributes

- None

3.1.44.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 4: Feature Framework View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.1.45 Modeled Relationship

A Modeled 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.

3.1.45.1 Aliases

This class has been reified from actual relationships to allow for clearer modeling.

3.1.45.2 Relationships

- Has Role
- Has View
- Is a Type of

3.1.45.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.45.4 Specific Attributes

- None

3.1.45.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 6: Modeled Relationship View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View

3.1.46 Modeled Relationship Role

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

3.1.46.1 Aliases

- None

3.1.46.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.46.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.46.4 Specific Attributes

- None

3.1.46.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 6: Modeled Relationship View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View

3.1.47 Modeled Statement

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

3.1.47.1 Aliases

- Statement

3.1.47.2 Relationships

- Has View
- Is a Type of

3.1.47.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.47.4 Specific Attributes

- None

3.1.47.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 6: Modeled Relationship View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View

3.1.48 Need

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

3.1.48.1 Aliases

- Informal Need
- Stakeholder Need

3.1.48.2 Relationships

- Advocates
- Is a Type of
- Perceives
- Satisfies

3.1.48.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.48.4 Specific Attributes

- Date Submitted
- Due Date

- Originator
- Priority
- Reference
- Request Type
- Source

3.1.48.5 Metamodel View References

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

3.1.49 Output Role

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

3.1.49.1 Aliases

- None

3.1.49.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.49.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.49.4 Specific Attributes

- None

3.1.49.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 19: Detail Requirements View

3.1.50 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.

3.1.50.1 Aliases

- Physical View (Deprecated)

3.1.50.2 Relationships

- Is a Type of

3.1.50.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.50.4 Specific Attributes

- None

3.1.50.5 Metamodel View References

- Figure 2: Class Hierarchy View

3.1.51 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.

3.1.51.1 Aliases

- None

3.1.51.2 Relationships

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

3.1.51.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.51.4 Specific Attributes

- Port Type

3.1.51.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View

3.1.52 Rationale

A Rationale is a statement, prose discussion, or some other explanation of the choice of an Alternative in an Allocation Decision.

3.1.52.1 Aliases

- None

3.1.52.2 Relationships

- Has View
- Is a Type of

3.1.52.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.52.4 Specific Attributes

- Reference

3.1.52.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 8: Functional Interaction View
- Figure 20: High Level Design View

3.1.53 Requirement Relationship

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

3.1.53.1 Aliases

- Requirement Transfer Function (RTF)

3.1.53.2 Relationships

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

3.1.53.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.53.4 Specific Attributes

- Requirement Population Rule--Interaction
- Requirement Population Rule--Role
- Requirement Statement PK Value Rule

3.1.53.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View
- Figure 23: Risk Analysis View

3.1.54 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.

3.1.54.1 Aliases

- “Shall” Statement

3.1.54.2 Relationships

- Has View
- Is a Type of

3.1.54.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.54.4 Specific Attributes

- Reference

3.1.54.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 9: Requirement Relationship View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View
- Figure 23: Risk Analysis View

3.1.55 Role Attribute Role

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

3.1.55.1 Aliases

- None

3.1.55.2 Relationships

- Allocated To
- Has Role
- Is a Type of

3.1.55.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.55.4 Specific Attributes

- Characterization Attribute Coupling PK Value Rule
- Characterization Attribute Coupling Population Rule
- Decomposition Attribute Coupling PK Value Rule
- Decomposition Attribute Coupling Population Rule

3.1.55.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View

3.1.56 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.

3.1.56.1 Aliases

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

3.1.56.2 Relationships

- Has State

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

3.1.56.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.56.4 Specific Attributes

- None

3.1.56.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 18: State Analysis View

3.1.57 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.

3.1.57.1 Aliases

- Service
- Feature
- Capability

3.1.57.2 Relationships

- Benefits
- Has Attribute

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

3.1.57.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.57.4 Specific Attributes

- None

3.1.57.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 12: Fitness Coupling View
- Figure 23: Risk Analysis View

3.1.58 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.

3.1.58.1 Aliases

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

3.1.58.2 Relationships

- Allocated To
- Has Attribute

- Has Feature
- Has Stakeholder
- Has State
- Has Subject
- Is a Type of
- Provides Interface

3.1.58.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.58.4 Specific Attributes

- None

3.1.58.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 4: Feature Framework View
- Figure 7: Architectural Relationship View
- Figure 9: Requirement Relationship View
- Figure 16: Domain Analysis View
- Figure 18: State Analysis View

3.1.59 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).

3.1.59.1 Aliases

- Medium
- Network
- Transport Mechanism
- SOA
- SOAC

- System of Access Component

3.1.59.2 Relationships

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

3.1.59.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.59.4 Specific Attributes

- None

3.1.59.5 Metamodel View References

- Figure 1: Summary Metamodel
- Figure 2: Class Hierarchy View
- Figure 16: Domain Analysis View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.1.60 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.

3.1.60.1 Aliases

- None

3.1.60.2 Relationships

- Is a Type of
- Is Triggered By

- Transitions From
- Transitions To

3.1.60.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.60.4 Specific Attributes

- None

3.1.60.5 Metamodel View References

- Figure 2: Class Hierarchy View
- Figure 18: State Analysis View

3.1.61 Value

A Value is the allowed range or specific value of a Class's Attribute. A Value describes a subset of a Class, and is part of the means of configuring a pattern.

3.1.61.1 Aliases

- None

3.1.61.2 Relationships

- Has Value
- Is a Type of

3.1.61.3 Common Attributes

- See Section 3.3.1 for Common Attributes

3.1.61.4 Specific Attributes

- None

3.1.61.5 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.

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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.1.3 Specific Attributes

- Failure Mode PK Value Rule
- Failure Mode Population Rule

3.2.1.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.2 Advocates

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

3.2.2.1 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.

3.2.2.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.2.3 Specific Attributes

- None

3.2.2.4 Meta-Model View References

- Figure 4: Feature Framework View

3.2.3 Allocated To

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

3.2.3.1 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 Modeled Relationship Role, Architectural Relationship Role, System of Access (SOA), Functional Role, Alternative, Input Role, Output Role, Attribute Role, Feature Attribute Role, Role Attribute Role, and Design Component Attribute Role. This role's cardinality is Many.

3.2.3.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.3.2 Specific Attributes

- Design Component Population Rule

- Design Component PK Value Rule

3.2.3.3 Meta-Model View References

- Figure 6: Modeled Relationship View

- Figure 7: Architectural Relationship View

- Figure 8: Functional Interaction View

- Figure 9: Requirement Relationship View

- Figure 10: Design Constraint View

- Figure 11: Attribute Coupling View

- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.2.4 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.

3.2.4.1 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.

3.2.4.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.4.3 Specific Attributes

- None

3.2.4.4 Meta-Model View References

- Figure 3: General Class View

3.2.5 Benefits

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

3.2.5.1 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.

3.2.5.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.5.3 Specific Attributes

- None

3.2.5.4 Meta-Model View Reference

- Figure 4: Feature Framework View

3.2.6 Causes Behavior

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

3.2.6.1 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.

3.2.6.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.6.3 Specific Attributes

- None

3.2.6.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.7 Causes Failure

The Causes Failure relationship links Functional Interactions to the Is Root Cause Of Relationship.

3.2.7.1 Roles

- Root Cause: This is the role played by the Is Root Cause Of Relationship. This role's cardinality is One.

- Causal Interaction: This is the role played by the Functional Interaction. This role's cardinality is One to Many.

3.2.7.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.7.3 Specific Attributes

- None

3.2.7.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.8 Causes Impact

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

3.2.8.1 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.

3.2.8.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.8.3 Specific Attributes

- None

3.2.8.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.9 Causes Mode

The Causes Mode relationship links Failure Modes with the Is Root Cause Of Relationship.

3.2.9.1 Roles

- Caused Mode: This is the role played by the Failure Mode. This role's cardinality is One.
- Root Cause: This is the role played by the Is Root Cause Of Relationship. This role's cardinality is One.

3.2.9.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.9.3 Specific Attributes

- None

3.2.9.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.10 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.

3.2.10.1 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.

3.2.10.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.10.3 Specific Attributes

- None

3.2.10.4 Meta-Model View References

- Figure 3: General Class View
- Figure 8: Functional Interaction View
- Figure 10: Design Constraint View
- Figure 17: Logical Architecture View
- Figure 18: State Analysis View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.2.11 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.

3.2.11.1 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.

3.2.11.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.11.3 Specific Attributes

- None

3.2.11.4 Meta-Model View References

- Figure 3: General Class View
- Figure 8: Functional Interaction View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 20: High Level Design View

3.2.12 Groups

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

3.2.12.1 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.

3.2.12.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.12.3 Specific Attributes

- None

3.2.12.4 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View

3.2.13 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.

3.2.13.1 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.

3.2.13.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.13.3 Specific Attributes

- None

3.2.13.4 Meta-Model View Reference

- Figure 4: Feature Framework View

3.2.14 Has Attribute

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

3.2.14.1 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.

3.2.14.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.14.3 Specific Attributes

- None

3.2.14.4 Meta-Model View References

- Figure 3: General Class View
- Figure 4: Feature Framework View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.2.15 Has Feature

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

3.2.15.1 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.

3.2.15.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.15.3 Specific Attributes

- None

3.2.15.4 Meta-Model View Reference

- Figure 4: Feature Framework View

3.2.16 Has Issue

The Has Issue relationship links an Issue to any class.

3.2.16.1 Roles

- Class: The class that has an Issue. This role is played by all Classes. This role's cardinality is Many.
- Issue: The Issue that relates to classes. This role is played by Issue. This role's cardinality is Many.

3.2.16.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.16.3 Specific Attributes

- None

3.2.16.4 Meta-Model View References

- Figure 3: General Class View

3.2.17 Has Previous

The Has Previous relationship links a Class to its previous version.

3.2.17.1 Roles

- Next Version: The version of a Class that has the most recent version. This role is played by all Classes. Its cardinality is Many.
- Previous Version: The version of a Class that has the next to recent version. This role is played by all Classes. Its cardinality is Many.

3.2.17.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.17.3 Specific Attributes

- None

3.2.17.4 Meta-Model View References

- Figure 3: General Class 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 Modeled 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 Modeled Relationship Role, Architectural Relationship Role, Functional Role, Alternative, 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.18.3 Specific Attributes

- Role Population Rule

- Role PK Value Rule

3.2.18.4 Meta-Model View References

- Figure 6: Modeled Relationship View

- Figure 7: Architectural Relationship View

- Figure 8: Functional Interaction View

- Figure 9: Requirement Relationship View

- Figure 10: Design Constraint View

- Figure 11: Attribute Coupling View

- Figure 12: Fitness Coupling View

- Figure 13: Characterization Coupling View

- Figure 14: Decomposition Coupling View

- Figure 15: Input/Output Coupling View

- Figure 16: Domain Analysis View

- Figure 17: Logical Architecture View

- Figure 19: Detail Requirements View

- Figure 20: 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.19.3 Specific Attributes

- None

3.2.19.4 Meta-Model 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.20.3 Specific Attributes

- None

3.2.20.4 Meta-Model View References

- Figure 18: 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.21.3 Specific Attributes

- None

3.2.21.4 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View

3.2.22 Has Value

The Has Value relationship links a Modeled Attribute to a defined Value it may have.

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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.22.3 Specific Attributes

- None

3.2.22.4 Meta-Model View References

- Figure 3: General Class View

- Figure 4: Feature Framework View

3.2.23 Has View

The Has View relationship links a Modeled 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 Modeled 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.

3.2.23.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.23.2 Specific Attributes

- None

3.2.23.3 Meta-Model View References

- Figure 6: Modeled Relationship View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: Input/Output Coupling View
- Figure 20: 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.24.3 Specific Attributes

- Failure Impact PK Value Rule
- Failure Impact Population Rule

3.2.24.4 Meta-Model View References

- Figure 23: 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 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.25.3 Specific Attributes

- None

3.2.25.4 Meta-Model View References

- Figure 23: 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.

3.2.26.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.26.2 Specific Attributes

- None

3.2.26.3 Meta-Model View References

- Figure 2: Class Hierarchy View
- Figure 3: General Class View
- Figure 5: Modeled Relationship Views View
- Figure 7: Architectural Relationship View
- Figure 8: Functional Interaction View
- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 11: Attribute Coupling View
- Figure 12: Fitness Coupling View
- Figure 13: Characterization Coupling View
- Figure 14: Decomposition Coupling View
- Figure 15: 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.

3.2.27.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.27.2 Specific Attributes

- None

3.2.27.3 Meta-Model View References

- Figure 20: 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.

3.2.28.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.28.2 Specific Attributes

- None

3.2.28.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 19: 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.

3.2.29.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.29.2 Specific Attributes

- None

3.2.29.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 19: 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.

3.2.30.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.30.2 Specific Attributes

- None

3.2.30.3 Meta-Model View References

- Figure 19: Detail Requirements View
- Figure 20: 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.

3.2.31.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.31.2 Specific Attributes

- None

3.2.31.3 Meta-Model View References

- Figure 18: State Analysis View

3.2.32 Perceives

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

3.2.32.1 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.

3.2.32.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.32.2 Specific Attributes

- None

3.2.32.3 Meta-Model View Reference

- Figure 4: Feature Framework View

3.2.33 Permits Architectural Relationship

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

3.2.33.1 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.

3.2.33.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.33.2 Specific Attributes

- None

3.2.33.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View

3.2.34 Permits Functional Interaction

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

3.2.34.1 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.

3.2.34.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.34.2 Specific Attributes

- None

3.2.34.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 19: Detail Requirements View

3.2.35 Permits Input/Output

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

3.2.35.1 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.

3.2.35.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.35.2 Specific Attributes

- None

3.2.35.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 19: Detail Requirements View

3.2.36 Permits SOA

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

3.2.36.1 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.

3.2.36.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.36.2 Specific Attributes

- None

3.2.36.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 19: Detail Requirements View
- Figure 20: High Level Design View

3.2.37 Plays Causal Role

The Plays Causal Role relationship links Functional Roles and Is Root Cause Of Relationship.

3.2.37.1 Roles

- Root Cause: This is the role played by the Is Root Cause Of Relationship. This role's cardinality is One.
- Role: This is the role played by the Functional Role. This role's cardinality is One to Many.

3.2.37.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.37.3 Specific Attributes

- None

3.2.37.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.38 Provides Context

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

3.2.38.1 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.

3.2.38.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.38.2 Specific Attributes

- None

3.2.38.3 Meta-Model View References

- Figure 19: Detail Requirements View
- Figure 23: Risk Analysis View

3.2.39 Provides Interface

The Provides relationship links an Interface to a System.

3.2.39.1 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.

3.2.39.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.39.2 Specific Attributes

- None

3.2.39.3 Meta-Model View References

- Figure 16: Domain Analysis View
- Figure 17: Logical Architecture View
- Figure 20: High Level Design View

3.2.40 Receives

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

3.2.40.1 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.

3.2.40.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.40.2 Specific Attributes

- None

3.2.40.3 Meta-Model View Reference

- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 16: Domain Analysis View
- Figure 19: Detail Requirements View

3.2.41 Relates AR

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

3.2.41.1 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.

3.2.41.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.41.2 Specific Attributes

- None

3.2.41.3 Meta-Model View Reference

- Figure 16: Domain Analysis View
- Figure 21: Interface Context View

3.2.42 Relates FI

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

3.2.42.1 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.

3.2.42.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.42.2 Specific Attributes

- None

3.2.42.3 Meta-Model View Reference

- Figure 16: Domain Analysis View

- Figure 21: Interface Context View

3.2.43 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.

3.2.43.1 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.

3.2.43.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.43.2 Specific Attributes

- None

3.2.43.3 Meta-Model View Reference

- Figure 16: Domain Analysis View
- Figure 21: Interface Context View

3.2.44 Relates LS

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

3.2.44.1 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.

3.2.44.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.44.2 Specific Attributes

- None

3.2.44.3 Meta-Model View Reference

- Figure 16: Domain Analysis View
- Figure 21: Interface Context View

3.2.45 Replaces

The Replaces relationship links Requirement Statements and Counter Requirement Statements.

3.2.45.1 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.

3.2.45.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.45.3 Specific Attributes

- Counter Requirement PK Value Rule
- Counter Requirement Population Rule

3.2.45.4 Meta-Model View References

- Figure 23: Risk Analysis View

3.2.46 Requires

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

3.2.46.1 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.

3.2.46.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.46.2 Specific Attributes

- None

3.2.46.3 Meta-Model View References

- Figure 18: State Analysis View

3.2.47 Satisfies

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

3.2.47.1 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.

3.2.47.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.47.2 Specific Attributes

- None

3.2.47.3 Meta-Model View Reference

- Figure 4: Feature Framework View

3.2.48 Sends

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

3.2.48.1 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.

3.2.48.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.48.2 Specific Attributes

- None

3.2.48.3 Meta-Model View Reference

- Figure 9: Requirement Relationship View
- Figure 10: Design Constraint View
- Figure 16: Domain Analysis View
- Figure 19: Detail Requirements View

3.2.49 Transitions From

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

3.2.49.1 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.

3.2.49.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.49.2 Specific Attributes

- None

3.2.49.3 Meta-Model View References

- Figure 18: State Analysis View

3.2.50 Transitions To

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

3.2.50.1 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.

3.2.50.1 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.50.2 Specific Attributes

- None

3.2.50.3 Meta-Model View References

- Figure 18: State Analysis View

3.2.51 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.

3.2.51.1 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.

3.2.51.2 Common Attributes

- See Section 3.3.1 for Common Attributes

3.2.51.3 Specific Attributes

- Interaction PK Value Rule
- Interaction Population Rule

3.2.51.4 Meta-Model View Reference

- Figure 4: Feature Framework View
- Figure 8: Functional Interaction View
- Figure 12: Fitness Coupling View
- Figure 23: 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

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 Allocated

The Allocated attribute indicates whether or not an Alternative in an Allocation Decision has been chosen.

3.3.2.2 Architectural Relationship PK Value Rule

The PK value configuration rule for the Reified Architectural Relationship used to set its Architectural Relationship Primary Key (ARPK) value, based on the following options:

- "IPK"—Set ARPK value to IPK value.
- Any other entry—Set ARPK value to (empty).

The PK value configuration rule for the Simple Architectural Relationship used to set its Architectural Relationship Primary Key (ARPK) value, based on the following options:

- "IPK"—Set ARPK value to IPK value.
- ""—Set ARPK value to blank.
- "R1PK"—Set ARPK value to Subject System RPK value.
- "R2PK"—Set ARPK value to Object System RPK value.
- "R12PK"—Set ARPK value to Subject System RPK value-Object System RPK Value.
- "IR12PK"—Set ARPK value to IPK Value-Subject System RPK value-Object System RPK Value.

3.3.2.3 Architectural Relationship Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Architectural Relationship population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.4 Architectural Relationship Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Architectural Relationship population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.5 Architectural Relationship Role PK Value Rule

The PK value configuration rule for the Architectural Relationship Role used to set the Architectural Relationship Role Primary Key (ARRPK), based on the following options:

- "IPK"—Set ARRPK value to IPK value.
- Any other entry—Set ARRPK value to (empty).

3.3.2.6 Architectural Relationship Role Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Architectural Relationship Role population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.7 Architectural Relationship Role Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Architectural Relationship Role population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.8 Characterization Attribute Coupling PK Value Rule

The PK value configuration rule for the Characterization Attribute Coupling used to set its Characterization Attribute Coupling Primary Key (CACPK) value, based on the following options:

- "RAPK"—Set CACPK value to RAPK value.
- "/string/"—Set CACPK value to string.
- "RAPK+/string/"—Set CACPK value to RAPK value plus string.

3.3.2.9 Characterization Attribute Coupling Population Rule

The conditional configuration rule for the Role Attribute Primary Key (RAPK) value used to trigger Characterization Attribute Coupling population, according to the following options:

- RAPK Value—The populating Role Attribute must have this PK value to trigger population.
- "*ANY*"—Any populating Role Attribute PK value will trigger population.

- (empty)—Populating Role Attribute must have (empty) PK value to trigger population.

3.3.2.10 Counter Requirement PK Value Rule

The PK value configuration rule for the Counter Requirement used to set its Counter Requirement Primary Key (CRPK) value, based on the following options:

- "RSPK"—Set CRPK value to RSPK value.
- "/string/"—Set CRPK value to string.
- "RSPK+/string/"—Set CRPK value to RSPK value plus string.

3.3.2.11 Counter Requirement Population Rule

The conditional configuration rule for the Requirement Statement Primary Key (RSPK) value used to trigger Counter Requirement population, according to the following options:

- RSPK Value—The populating Requirement Statement must have this PK value to trigger population.
- "*ANY*"—Any populating Requirement Statement PK value will trigger population.
- (empty)—Populating Requirement Statement must have (empty) PK value to trigger population.

3.3.2.12 Decomposition Attribute Coupling PK Value Rule

The PK value configuration rule for the Decomposition Attribute Coupling used to set its Decomposition Attribute Coupling Primary Key (DACPK) value, based on the following options:

- "RAPK"—Set DACPK value to RAPK value.
- "/string/"—Set DACPK value to string.
- "RAPK+/string/"—Set DACPK value to RAPK value plus string.

3.3.2.13 Decomposition Attribute Coupling Population Rule

The conditional configuration rule for the Role Attribute Primary Key (RAPK) value used to trigger Decomposition Attribute Coupling population, according to the following options:

- RAPK Value—The populating Role Attribute must have this PK value to trigger population.
- "*ANY*"—Any populating Role Attribute PK value will trigger population.
- (empty)—Populating Role Attribute must have (empty) PK value to trigger population.

3.3.2.14 Design Component PK Value Rule

The PK value configuration rule for the Design Component used to set its Design Component Primary Key (DCPK) value, based on the following options:

- "RPK"—Set DCPK value to RPK value.
- "/string/"—Set DCPK value to string.
- "RPK+/string/"—Set DCPK value to RPK value plus string.

3.3.2.15 Design Component Population Rule

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Design Component population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.16 Date Submitted

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

3.3.2.17 Due Date

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

3.3.2.18 Failure Impact PK Value Rule

The PK value configuration rule for the Failure Impact used to set its Failure Impact Primary Key (FIPK) value, based on the following options:

- "FPK"—Set FIPK value to FPK value.
- "/string/"—Set FIPK value to string.
- "FPK+/string/"—Set FIPK value to FPK value plus string.

3.3.2.19 Failure Impact Population Rule

The conditional configuration rule for the Feature Primary Key (FPK) value used to trigger Failure Impact population, according to the following options:

- FPK Value—The populating Feature must have this PK value to trigger population.
- "*ANY*"—Any populating Feature PK value will trigger population.
- (empty)—Populating Feature must have (empty) PK value to trigger population.

3.3.2.20 Failure Mode PK Value Rule

The PK value configuration rule for the Failure Mode used to set its Failure Mode Primary Key (FMPK) value, based on the following options:

- "DCPK"—Set FMPK value to DCPK value.
- "/string/"—Set FMPK value to string.
- "DCPK+/string/"—Set FMPK value to DCPK value plus string.

3.3.2.21 Failure Mode Population Rule

The conditional configuration rule for the Design Component Primary Key (DCPK) value used to trigger Failure Mode population, according to the following options:

- DCPK Value—The populating Design Component must have this PK value to trigger population.
- "*ANY*"—Any populating Design Component PK value will trigger population.
- (empty)—Populating Design Component must have (empty) PK value to trigger population.

3.3.2.22 Fitness Attribute Coupling PK Value Rule

The PK value configuration rule for the Fitness Attribute Coupling used to set its Fitness Attribute Coupling Primary Key (FACPK) value, based on the following options:

- "FAPK"—Set FACPK value to FAPK value.
- "/string/"—Set FACPK value to string.
- "FAPK+/string/"—Set FACPK value to FAPK value plus string.

3.3.2.23 Fitness Attribute Coupling Population Rule

The conditional configuration rule for the Feature Attribute Primary Key (FAPK) value used to trigger Fitness Attribute Coupling population, according to the following options:

- FAPK Value—The populating Feature Attribute must have this PK value to trigger population.
- "*ANY*"—Any populating Feature Attribute PK value will trigger population.
- (empty)—Populating Feature Attribute must have (empty) PK value to trigger population.

3.3.2.24 Interaction PK Value Rule

The PK value configuration rule for the Interaction used to set its Interaction Primary Key (IPK) value, based on the following options:

- “FPK”—Set IPK value to FPK value.
- “/string/”—Set IPK value to string.
- “FPK+/string/”—Set IPK value to FPK value plus string.
- “*ANY*”—Used to populate additional Feature-Interaction relationships for already (using the other PK value rules) populated Interactions.

3.3.2.25 Interaction Population Rule

The conditional configuration rule for the Feature Primary Key (FPK) value used to trigger Interaction population according to the following options:

- FPK Value—The populating Feature must have this PK value to trigger population.
- “*ANY*”—Any populating Feature PK value will trigger population.
- (empty)—Populating Feature must have (empty) PK value to trigger population.

3.3.2.26 Interface PK Value Rule

The PK value configuration rule for the Interface used to set the Interface Primary Key (IFCPK) value, based on the following options:

- “IPK”—Set IFCPK value to IPK (Interaction Primary Key) value.
- Any other entry—Set IFCPK value to (empty).

3.3.2.27 Interface Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Interface population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- “*ANY*”—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.28 Interface Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Interface population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- “*ANY*”—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.29 I/O Attribute Coupling PK Value Rule

The PK value configuration rule for the I/O Attribute Coupling used to set its I/O Attribute Coupling Primary Key (IOACPK) value, based on the following options:

- "IOAPK"—Set IOACPK value to IOAPK value.
- "/string/"—Set IOACPK value to string.
- "IOAPK+/string/"—Set IOACPK value to IOAPK value plus string.

3.3.2.30 I/O Attribute Coupling Population Rule

The conditional configuration rule for the I/O Attribute Primary Key (IOAPK) value used to trigger I/O Attribute Coupling population, according to the following options:

- IOAPK Value—The populating I/O Attribute must have this PK value to trigger population.
- "*ANY*"—Any populating I/O Attribute PK value will trigger population.
- (empty)—Populating I/O Attribute must have (empty) PK value to trigger population.

3.3.2.31 Input/Output PK Value Rule

The PK value configuration rule for the Input/Output used to set its Input/Output Primary Key (IOPK) value, based on the following options:

- "IPK"—Set IOPK value to IPK value.
- Any other entry—Set IOPK value to (empty).

3.3.2.32 Input/Output Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Input/Output population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.33 Input/Output Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Input/Output population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.34 Originator

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

3.3.2.35 Port PK Value Rule

The PK value configuration rule for the Port used to set its Port Primary Key (PPK) value, based on the following options:

- PPK Value set to IFCPK Value.

3.3.2.36 Port Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Input/Output population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.37 Port Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Input/Output population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.38 Port Type

A Port Type describes whether a Port is an Input Port, Output Port, or Both.

3.3.2.39 Priority

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

3.3.2.40 Rank

Rank is the relative preference of Alternatives in an Allocation Decision.

3.3.2.41 Reference

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

3.3.2.42 Request Type

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

3.3.2.43 Requirement Statement PK Value Rule

The PK value configuration rule for the Requirement used to set the Requirement Statement Primary Key (RSPK) value, based on the following options:

- "IPK"—Set RSPK value to IPK (Interaction Primary Key) value.
- "/string/"—Set RSPK value to string.
- "IPK+/string/"—Set RSPK value to IPK value plus string.
- "RPK"—Set RSPK value to RPK

3.3.2.44 Requirement Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Requirement Statement population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.45 Requirement Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Requirement Statement population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

3.3.2.46 Role PK Value Rule

The PK value configuration rule for the Role used to set its Role Primary Key (RPK) value, based on the following options:

- "IPK"—Set RPK value to IPK (Interaction Primary Key) value.
- "/string/"—Set RPK value to string.
- "IPK+/string/"—Set RPK value to IPK value plus string.

3.3.2.47 Role Population Rule

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Role population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.48 Score

Score is the result of an evaluation of an Alternative in an Allocation Decision.

3.3.2.49 Source

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

3.3.2.50 System of Access PK Value Rule

The PK value configuration rule for the System of Access (SOA) used to set its System of Access Primary Key (SOAPK) value, based on the following options:

- "IPK"—Set SOAPK value to IPK value.
- Any other entry—Set SOAPK value to (empty).

3.3.2.51 System of Access Population Rule--Interaction

The conditional configuration rule for the Interaction Primary Key (IPK) value used to trigger Input/Output population, according to the following options:

- IPK Value—The populating Interaction must have this PK value to trigger population.
- "*ANY*"—Any populating Interaction PK value will trigger population.
- (empty)—Populating Interaction must have (empty) PK value to trigger population.

3.3.2.52 System of Access Population Rule--Role

The conditional configuration rule for the Role Primary Key (RPK) value used to trigger Input/Output population, according to the following options:

- RPK Value—The populating Role must have this PK value to trigger population.
- "*ANY*"—Any populating Role PK value will trigger population.
- (empty)—Populating Role must have (empty) PK value to trigger population.

