

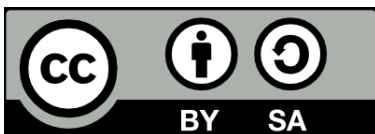
---

# S\*Metamodel

---

Metamodel Version 7.1

07/03/2019





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

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

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

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

Corporate Officer  
ICTT System Sciences  
378 South Airport Street  
Terre Haute, IN 47803  
[812-232-2208](tel:812-232-2208)

Systematica is a registered trademark of System Sciences, LLC.

## TABLE OF CONTENTS

1.1	DOCUMENT PURPOSE .....	7
1.2	DOCUMENT SCOPE .....	7
1.3	DOCUMENT OVERVIEW .....	7
1.4	DOCUMENT REFERENCES .....	7
1.5	DOCUMENT HISTORY .....	8
2.1	SUMMARY METAMODEL.....	10
2.1.1	MODEL-BASED SYSTEMS ENGINEERING (MBSE).....	12
2.1.2	PATTERN-BASED SYSTEMS ENGINEERING (PBSE) .....	12
2.1.3	INTELLIGENCE-BASED SYSTEMS ENGINEERING (IBSE) .....	12
2.2	CLASS HIERARCHY VIEW .....	13
2.3	GENERAL CLASS VIEW .....	14
2.4	FEATURE FRAMEWORK VIEW.....	15
2.5	MODELED RELATIONSHIP VIEWS VIEW .....	16
2.6	MODELED RELATIONSHIP VIEW .....	17
2.7	ARCHITECTURAL RELATIONSHIP VIEW .....	17
2.8	EMBEDDED INTELLIGENCE (EI) VIEW .....	18
2.9	FUNCTIONAL INTERACTION VIEW .....	19
2.10	REQUIREMENT RELATIONSHIP VIEW .....	20
2.11	DESIGN CONSTRAINT VIEW .....	21
2.12	ATTRIBUTE COUPLING VIEW.....	22
2.13	REQUIREMENTS COUPLING VIEW .....	23
2.14	DESIGN COUPLING VIEW .....	24
2.15	DOMAIN ANALYSIS VIEW.....	25
2.16	LOGICAL ARCHITECTURE VIEW .....	26
2.17	STATE ANALYSIS VIEW .....	27

- 2.18    DETAIL REQUIREMENTS VIEW ..... 27**
- 2.19    HIGH LEVEL DESIGN VIEW ..... 29**
- 2.20    SUMMARY PATTERN CONFIGURATION VIEW ..... 30**
- 3.1    CLASSES ..... 32**
  - 3.1.1    ALLOCATION DECISION..... 32
  - 3.1.2    ALTERNATIVE..... 33
  - 3.1.3    ARCHITECTURAL RELATIONSHIP ..... 34
  - 3.1.4    ARCHITECTURAL RELATIONSHIP ROLE ..... 36
  - 3.1.5    ATTRIBUTE COUPLING..... 37
  - 3.1.6    ATTRIBUTE COUPLING MAP..... 38
  - 3.1.7    ATTRIBUTE ROLE ..... 39
  - 3.1.8    CLASS..... 40
  - 3.1.9    DESIGN COMPONENT ATTRIBUTE ROLE ..... 41
  - 3.1.10    DESIGN CONSTRAINT ..... 42
  - 3.1.11    DESIGN CONSTRAINT STATEMENT ..... 44
  - 3.1.12    DESIGN COUPLING ..... 45
  - 3.1.13    DESIGN COUPLING MAP ..... 46
  - 3.1.14    DOMAIN ..... 47
  - 3.1.15    DOMAIN SYSTEM ..... 48
  - 3.1.16    EVENT ..... 49
  - 3.1.17    FEATURE (SERVICE) ..... 50
  - 3.1.18    FEATURE ATTRIBUTE ROLE..... 51
  - 3.1.19    FUNCTIONAL INTERACTION ..... 52
  - 3.1.20    FUNCTIONAL ROLE ..... 53
  - 3.1.21    INFORMATION INPUT/OUTPUT ..... 55
  - 3.1.22    INPUT/OUTPUT ..... 56
  - 3.1.23    INPUT ROLE..... 57
  - 3.1.24    INTERFACE ..... 58
  - 3.1.25    ISSUE..... 59
  - 3.1.26    LOGICAL SYSTEM..... 60
  - 3.1.27    MANAGED SYSTEM (MDS) ..... 62
  - 3.1.28    MANAGEMENT SYSTEM (MTS) ..... 63
  - 3.1.29    MANAGES..... 64
  - 3.1.30    MODELED ATTRIBUTE..... 65
  - 3.1.31    MODELED RELATIONSHIP ..... 66
  - 3.1.32    MODELED RELATIONSHIP ROLE ..... 68
  - 3.1.33    MODELED STATEMENT ..... 69
  - 3.1.34    NEED ..... 70
  - 3.1.35    OUTPUT ROLE ..... 71
  - 3.1.36    PHYSICAL INPUT/OUTPUT ..... 72
  - 3.1.37    PHYSICAL SYSTEM ..... 73
  - 3.1.38    PORT..... 75
  - 3.1.39    RATIONALE..... 76
  - 3.1.40    REQUIREMENTS COUPLING..... 77
  - 3.1.41    REQUIREMENT RELATIONSHIP ..... 78
  - 3.1.42    REQUIREMENTS COUPLING MAP..... 79
  - 3.1.43    REQUIREMENT STATEMENT ..... 80
  - 3.1.44    ROLE ATTRIBUTE ROLE..... 81
  - 3.1.45    STATE..... 82
  - 3.1.46    SYSTEM ..... 83
  - 3.1.47    SYSTEM OF ACCESS (SOA) ..... 85

3.1.48	SYSTEM OF USERS (SOU).....	86
3.1.49	TRANSITION.....	87
3.1.50	VALUE.....	88
<b>3.2</b>	<b>METACLASS RELATIONSHIPS .....</b>	<b>89</b>
3.2.1	ADVOCATES .....	89
3.2.2	ALLOCATED TO .....	90
3.2.3	APPEARS IN.....	92
3.2.4	BENEFITS .....	92
3.2.5	CONTAINS.....	92
3.2.6	DERIVED FROM.....	93
3.2.7	EMERGES FROM .....	93
3.2.8	EXEMPLIFIES.....	94
3.2.9	GROUPS .....	94
3.2.10	HAS ADVOCATE .....	95
3.2.11	HAS ATTRIBUTE .....	95
3.2.12	HAS FEATURE.....	96
3.2.13	HAS ISSUE.....	96
3.2.14	HAS PREVIOUS.....	96
3.2.15	HAS ROLE .....	97
3.2.16	HAS STAKEHOLDER.....	97
3.2.17	HAS STATE .....	98
3.2.18	HAS SUBJECT.....	98
3.2.19	HAS VALUE .....	98
3.2.20	HAS VIEW .....	99
3.2.21	INTERACTS THROUGH .....	100
3.2.22	IS A TYPE OF .....	100
3.2.23	IS CONSTRAINED BY.....	101
3.2.24	IS FACILITATED BY EXTERNALLY.....	101
3.2.25	IS FACILITATED BY INTERNALLY .....	101
3.2.26	IS LINKED BY EXTERNALLY.....	102
3.2.27	IS LINKED BY INTERNALLY .....	102
3.2.28	IS TRIGGERED BY.....	103
3.2.29	IS SPECIFIED BY .....	103
3.2.30	IS USED DURING .....	103
3.2.31	PERCEIVES.....	104
3.2.32	PERMITS ARCHITECTURAL RELATIONSHIP .....	104
3.2.33	PERMITS FUNCTIONAL INTERACTION .....	104
3.2.34	PERMITS INPUT/OUTPUT .....	105
3.2.35	PERMITS SOA .....	105
3.2.36	PROVIDES CONTEXT.....	106
3.2.37	PROVIDES INTERFACE.....	106
3.2.38	RECEIVES .....	106
3.2.39	REQUIRES .....	107
3.2.40	SATISFIES .....	107
3.2.41	SENDS .....	107
3.2.42	TRANSITIONS FROM.....	108
3.2.43	TRANSITIONS TO .....	108
3.2.44	USES FUNCTIONAL INTERACTION .....	109
<b>3.3</b>	<b>METACLASS ATTRIBUTES .....</b>	<b>109</b>
3.3.1	ALLOCATED .....	109
3.3.2	AUTHOR .....	109
3.3.3	CHANGE DATE .....	109
3.3.4	CHANGE DESCRIPTION .....	109

3.3.5 CLASS LEVEL ..... 109

3.3.6 DATE SUBMITTED..... 110

3.3.7 DEFINITION ..... 110

3.3.8 DUE DATE..... 110

3.3.9 ID..... 110

3.3.10 MAJOR VERSION..... 110

3.3.11 MINOR VERSION..... 110

3.3.12 NAME ..... 110

3.3.13 ORGANIZATION OWNER..... 110

3.3.14 ORIGINATOR ..... 110

3.3.15 OWNER..... 110

3.3.16 PORT TYPE ..... 111

3.3.17 PRIORITY..... 111

3.3.18 RANK ..... 111

3.3.19 REFERENCE..... 111

3.3.20 REQUEST TYPE..... 111

3.3.21 SCORE ..... 111

3.3.22 SOURCE ..... 111

3.3.23 STATUS..... 111

3.3.24 UPDATE VERSION..... 111

# 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 and describe the entities shown in these views. The intended audience of this document is a system engineering methodologist concerned with defining a methodology for an organization.

## 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) Systematica Process Views Model
- 2) Systematica Pattern-Based Systems Engineering Process
- 3) The Systems Engineering Process Workshop

## 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
7/3/19	7.1.6A	Creative Commons License nomenclature



# 2 Metamodel Views

This section uncovers the Systematica Metamodel (S\* Metamodel) by first reviewing a 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 a summary view 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:

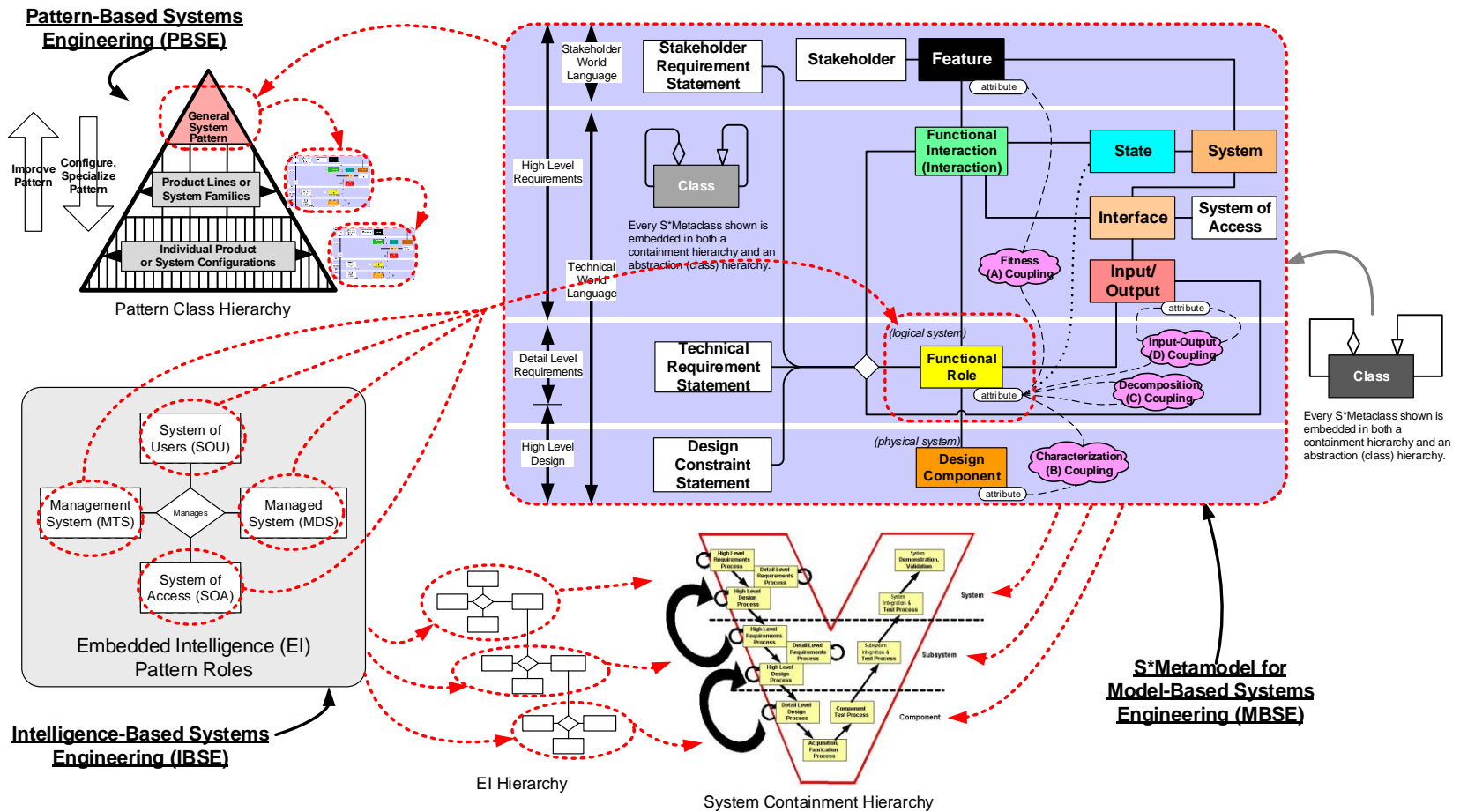
- Summary Metamodel: The summary metamodel for informal reference and training.
- Class Hierarchy View: The formal view that depicts the class hierarchy of all metaclasses.
- General Class View: The formal view that depicts the relationships allowed for every metaclass.
- Feature Framework View: The 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.
- Embedded Intelligence (EI) View: The formal view that depicts the specialization of the abstract Architectural Relationship View into the model upon which the Intelligence-Based Systems Engineering (IBSE) and the Embedded Intelligence (EI) pattern is based.
- 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.
- Requirements Coupling View: The formal view that depicts the classes and relationships used to couple Feature Attributes to Functional Role Attributes.
- Design Coupling View: The formal view that depicts the classes and relationships used to couple Functional Role Attributes to Design Component 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.
- Summary Pattern Configuration View: The summary view that depicts how the classes and relationships of a pattern are populated during the pattern configuration process.

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](#).





 <p>ICTT System Sciences® Understand your systems.</p>	<p>S* Metamodel—Summary View, for MBSE, PBSE, and IBSE</p> <p>03-04-19 V1.7.1</p>	 <p>Systematica® Do more with less</p>
---	---	---

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 level increases:

- Systematica Level 1: Model Based Systems Engineering (MBSE), a systems engineering methodology for a single complex system.
- Systematica Level 2: Pattern-Based Systems Engineering (PBSE), a systems engineering methodology for a family or product line of systems.
- Systematica Level 3: Intelligence-Based Systems Engineering (IBSE), a systems engineering methodology for intelligent 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 manages the relationships between an Input/Output, the Functional Role, and which System of Access facilitates the interaction 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 Physical System, often called 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 class, or generalization, relationship for each class, 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.

### **2.1.3 Intelligence-Based Systems Engineering (IBSE)**

The Intelligence-Based Systems Engineering (IBSE) model describes a specific pattern of Functional Roles that is appropriate and valuable for systems that center on embedded intelligence and management of systems. The Embedded Intelligence (EI) roles in the lower left of [Figure 1](#) describe the pattern suggested in IBSE. The EI pattern models four functional roles in any management interaction. A Management System (MTS) manages a Managed System (MDS) through a System of Access (SOA) to provide services consumed by a System of Users (SOU). This pattern leads to a decomposition strategy to the System Containment Hierarchy that leads to much improved understanding, analyses, and decisions on the very complicated issues dealing with embedded intelligence and management of systems.

## 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.

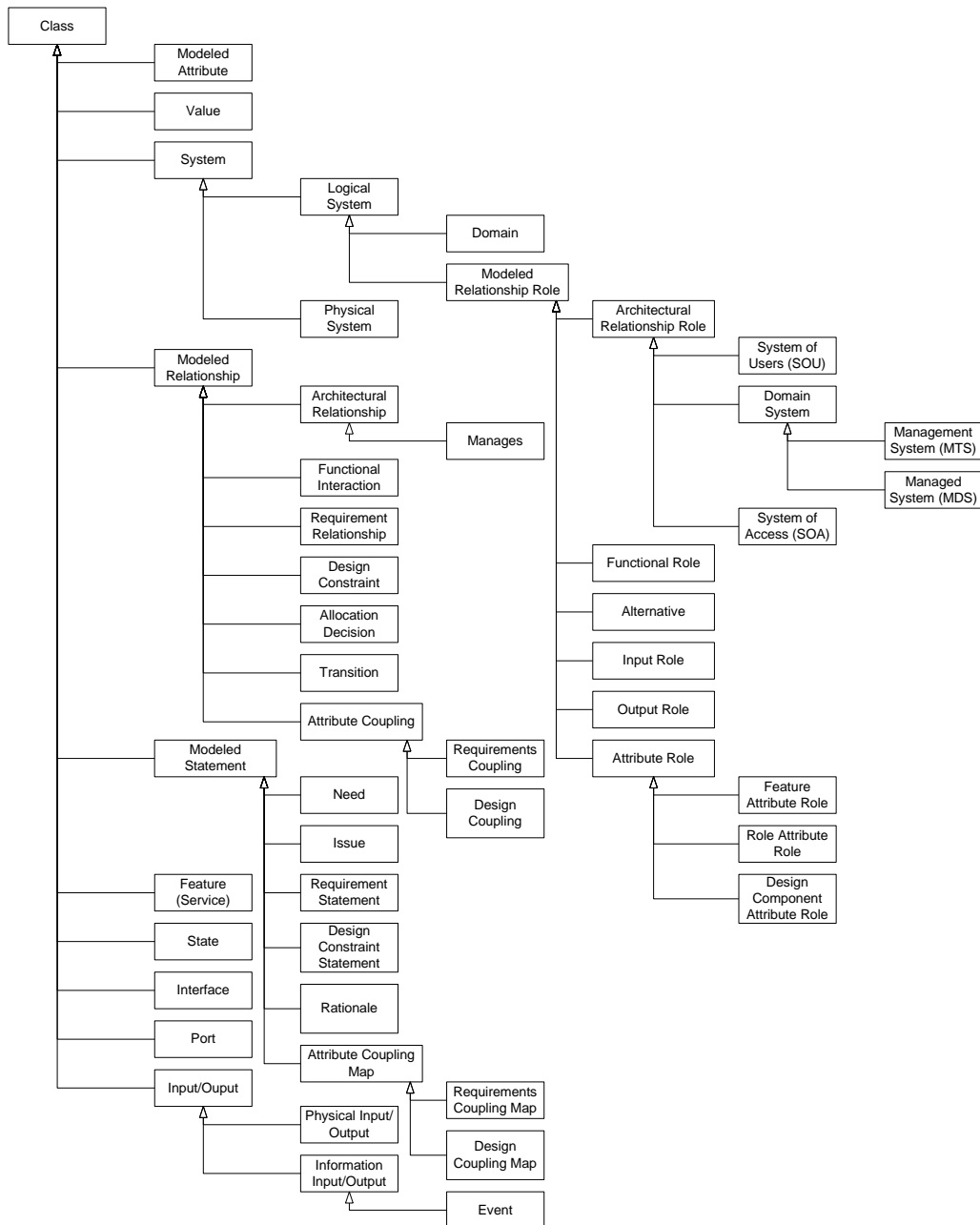


Figure 2: Class Hierarchy View

## 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.

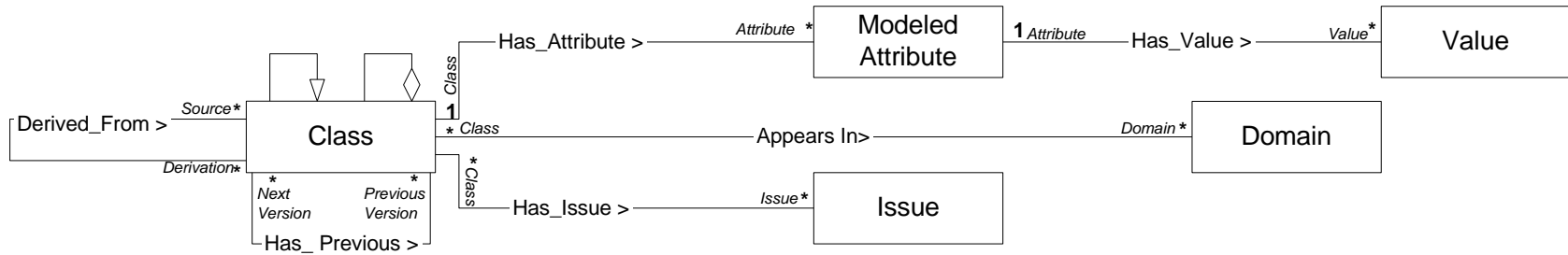


Figure 3: General Class View

## 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, or Services. This view defines the framework that guides and support value-based requirements and design approaches.

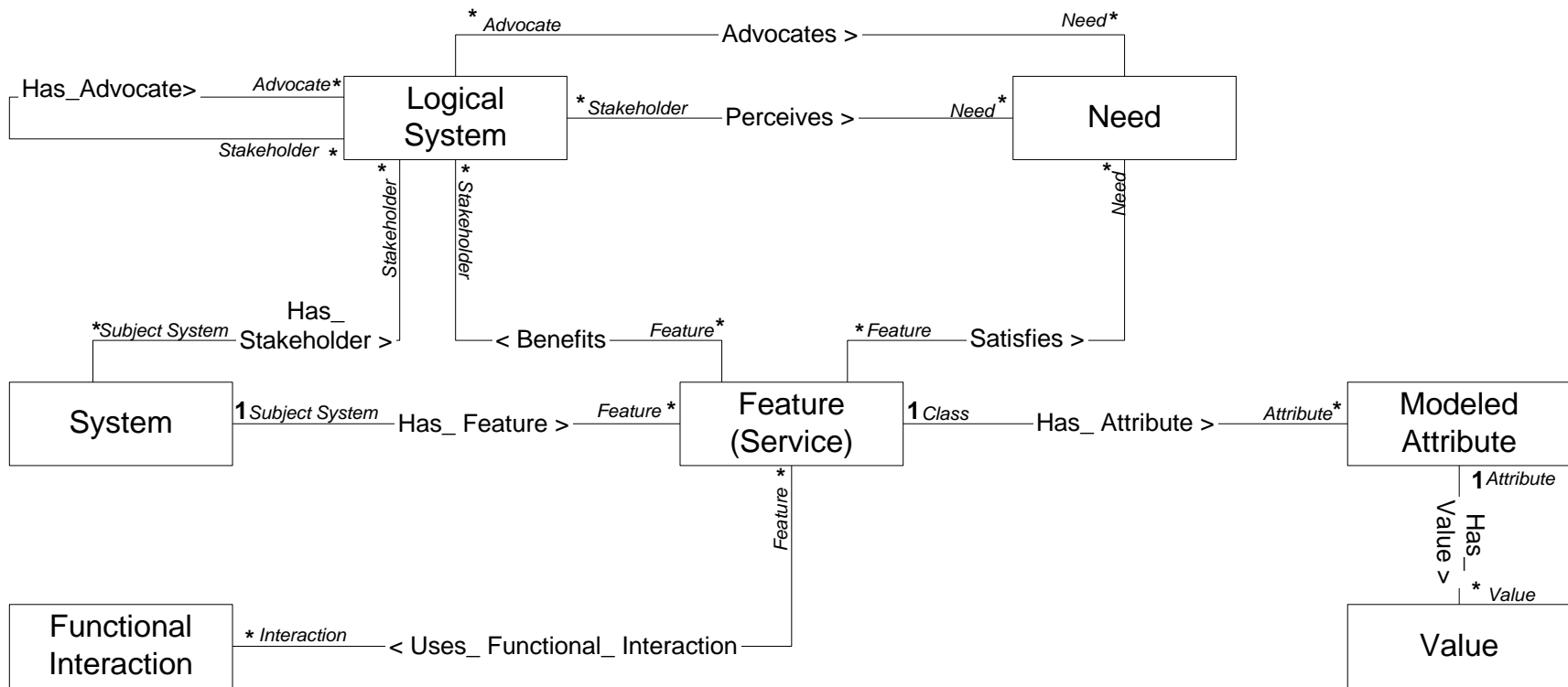


Figure 4: Feature Framework View

## 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.

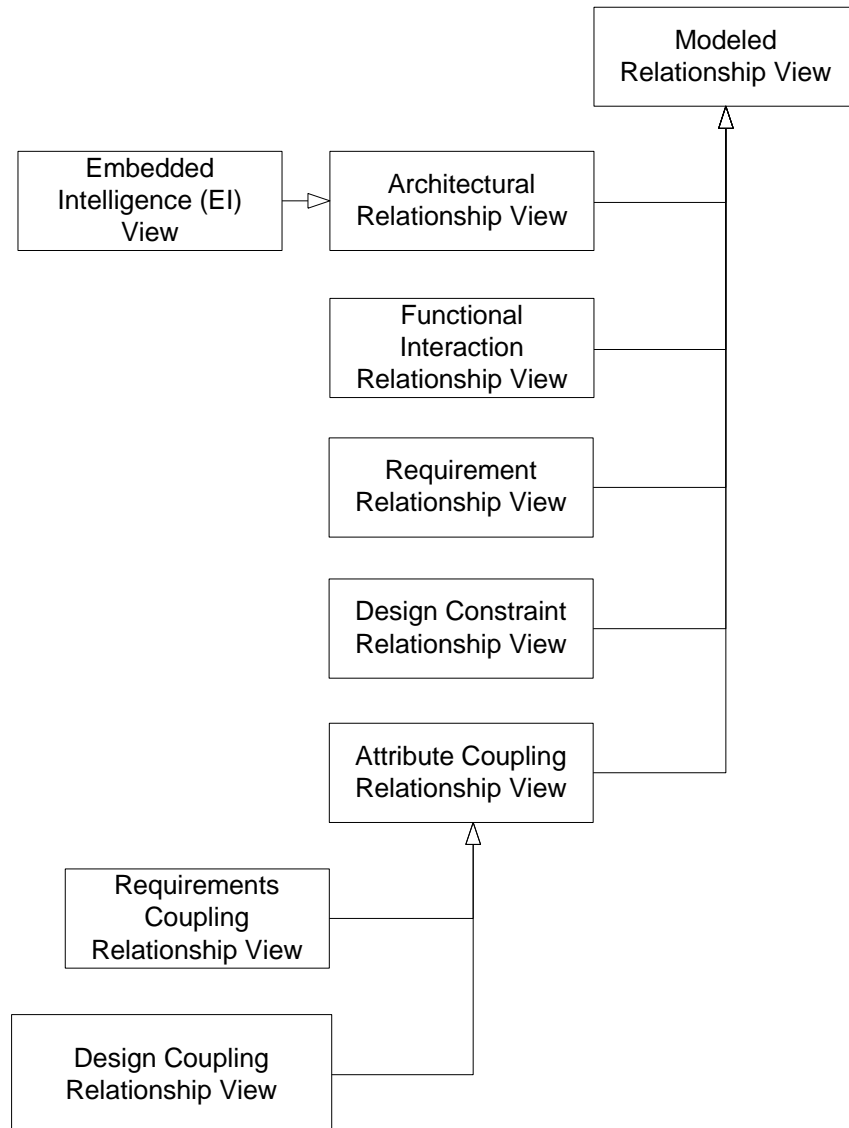


Figure 5: Modeled Relationship Views View



## 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.

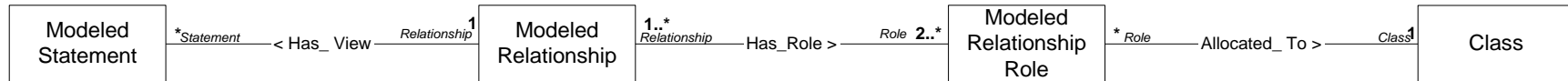


Figure 6: Modeled Relationship View

## 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.

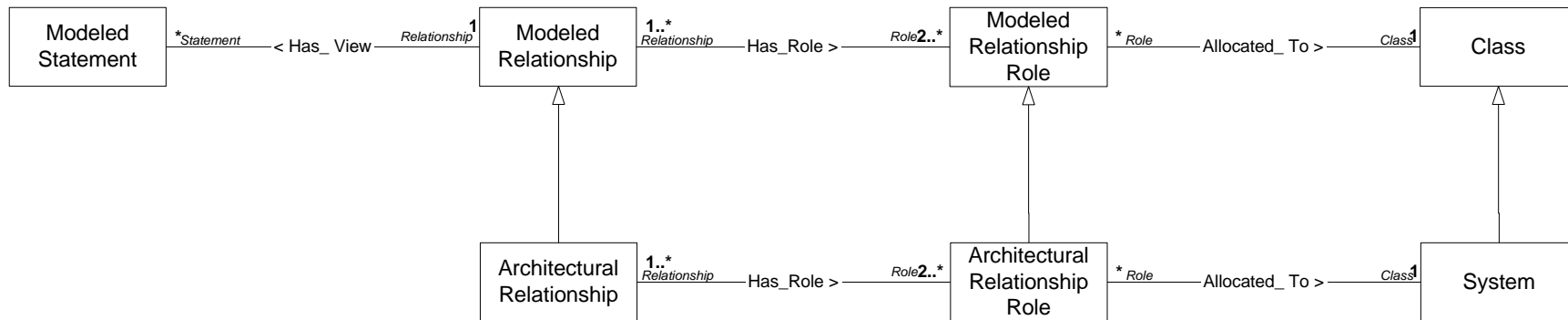


Figure 7: Architectural Relationship View

## 2.8 Embedded Intelligence (EI) View

The Embedded Intelligence (EI) View specializes the Architectural Relationship View into a model that is central to the Intelligence-Based Systems Engineering (IBSE) methodology. The “manages” relationship and its roles are the main Architectural Relationship pattern used in systems that feature system intelligence or management.

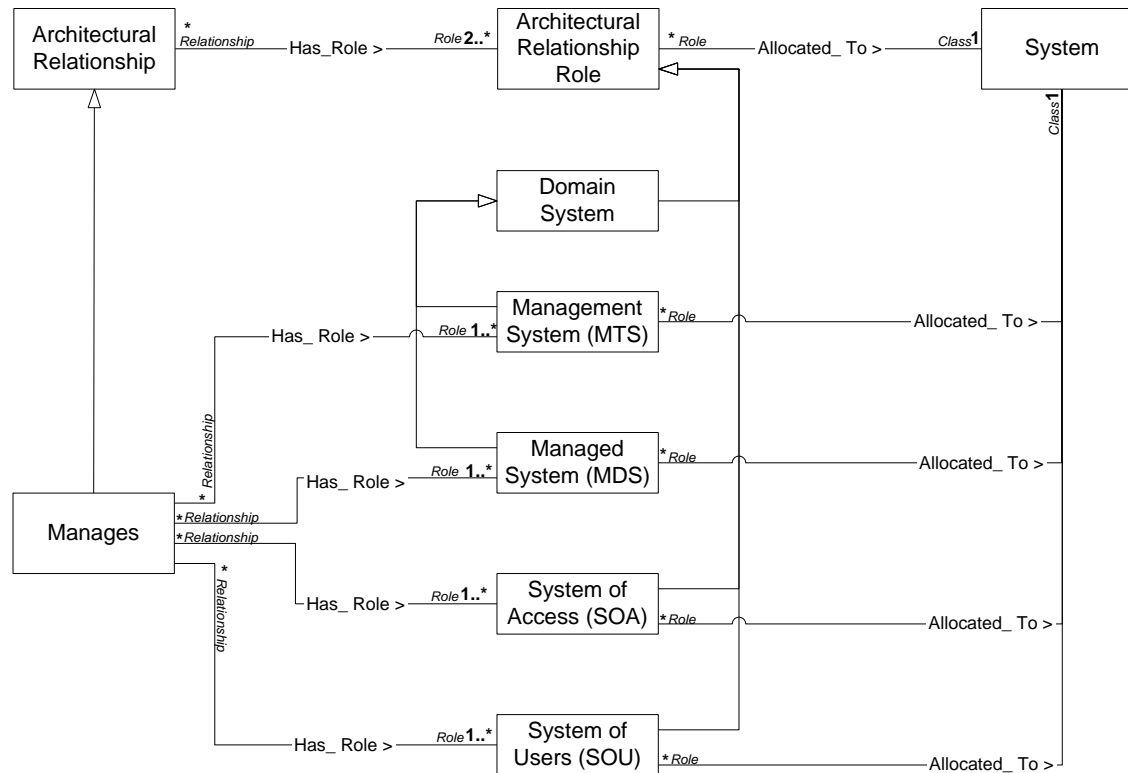


Figure 8: Embedded Intelligence (EI) View

## 2.9 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.

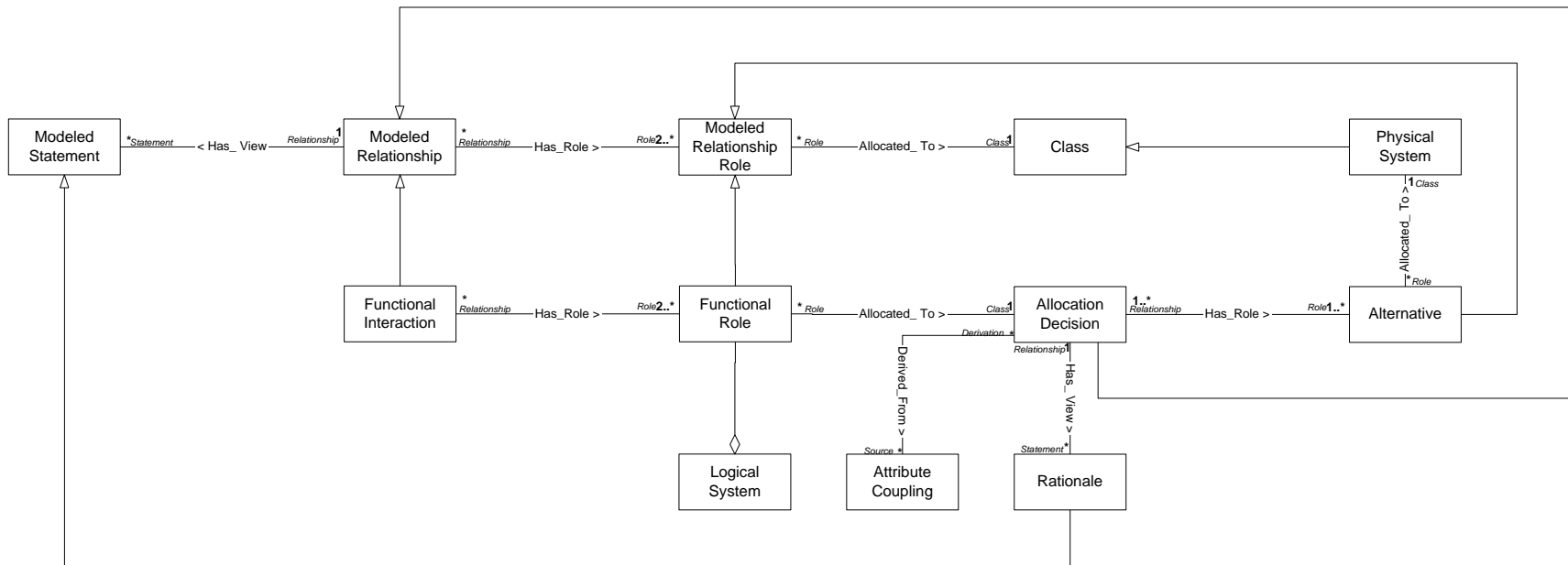


Figure 9: Functional Interaction View

## 2.10 Requirement Relationship View

Figure 10 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.

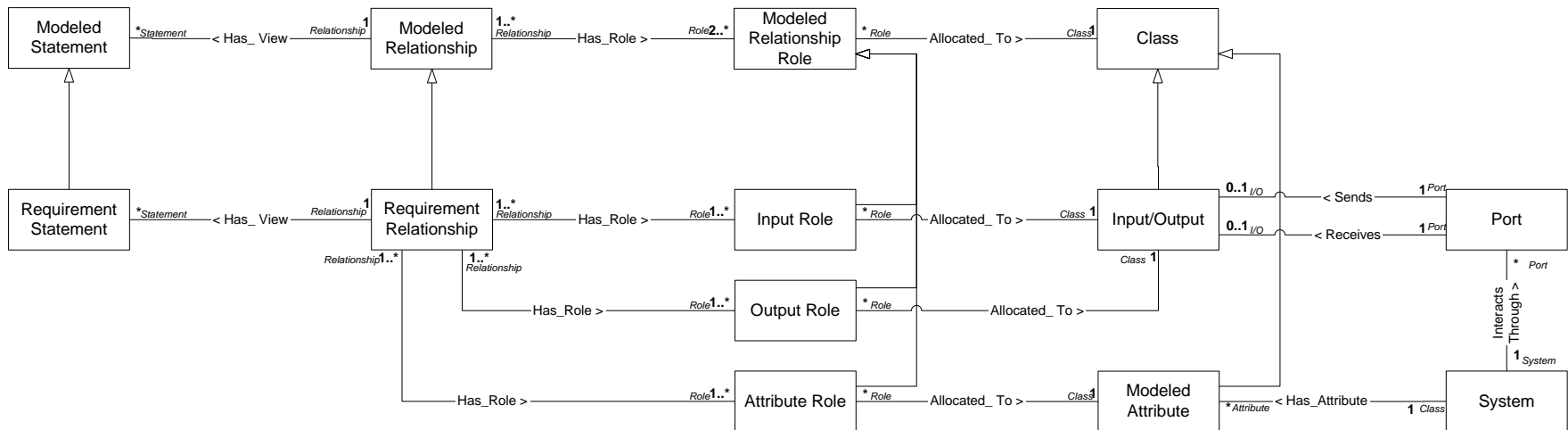


Figure 10: Requirement Relationship View

## 2.11 Design Constraint View

The Design Constraint View in Figure 11 defines the Design Constraint and Design Constraint Statements as a specialization of the Modeled Relationship pattern that modifies a System's Physical Subsystem.

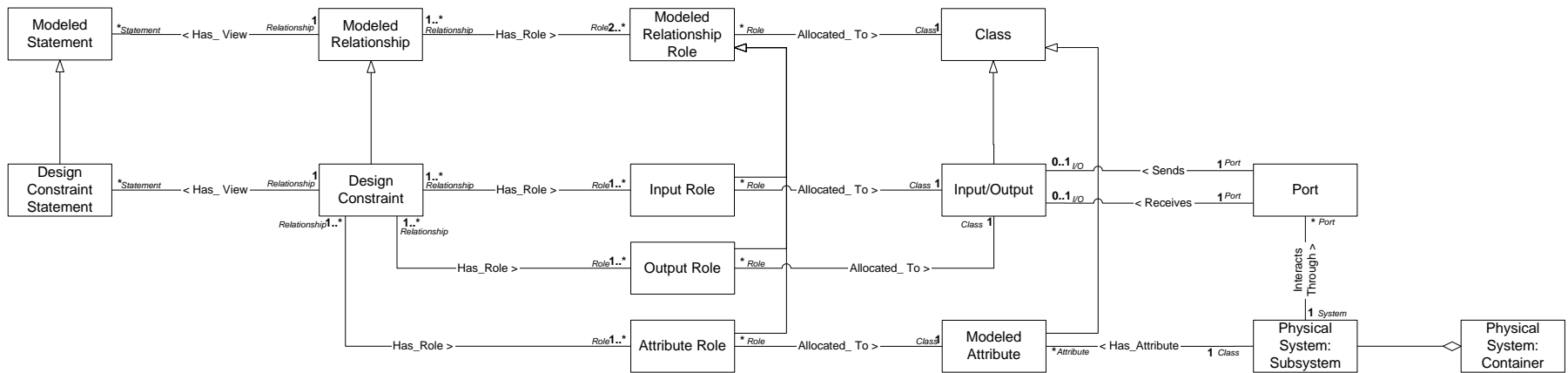


Figure 11: Design Constraint View

## 2.12 Attribute Coupling View

The Modeled Relationship View is specialized into a pattern that relates attributes in Figure 12. Attributes are coupled together with Attribute Coupling Maps as prose, mathematical equations, etc. to describe those relationships.

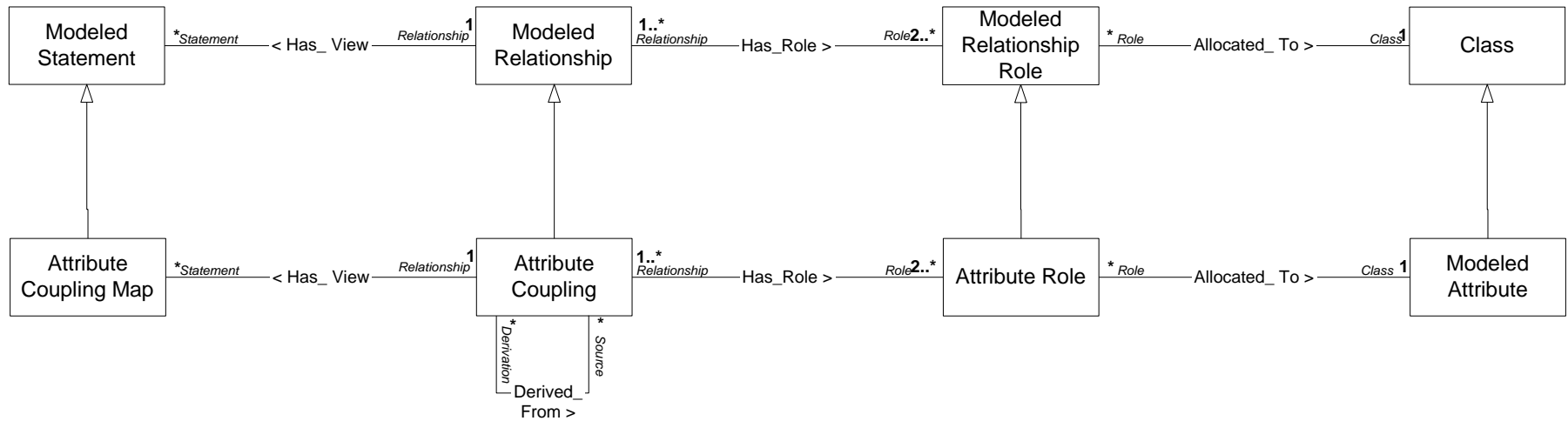


Figure 12: Attribute Coupling View

## 2.13 Requirements Coupling View

The Requirements 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 and also between Functional Role Attributes.

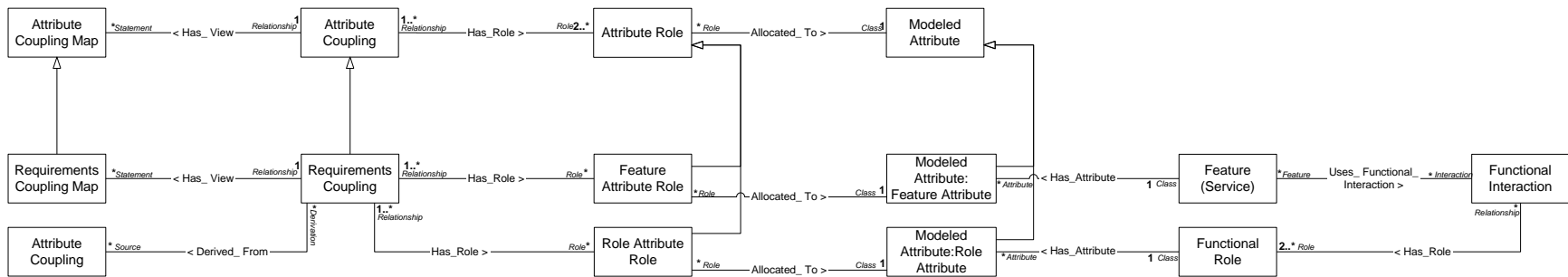


Figure 13: Requirements Coupling View

## 2.14 Design Coupling View

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

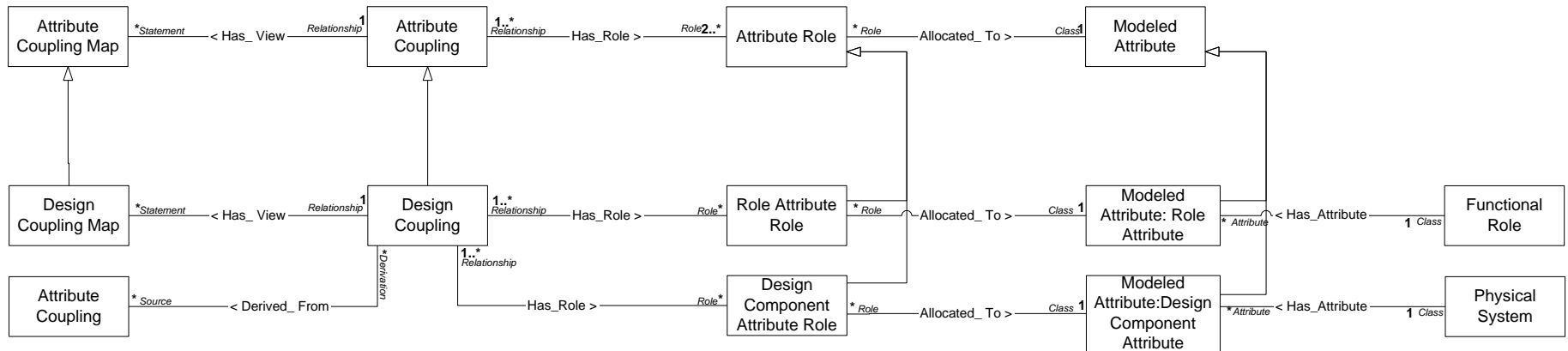


Figure 14: Design Coupling View



## 2.15 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.

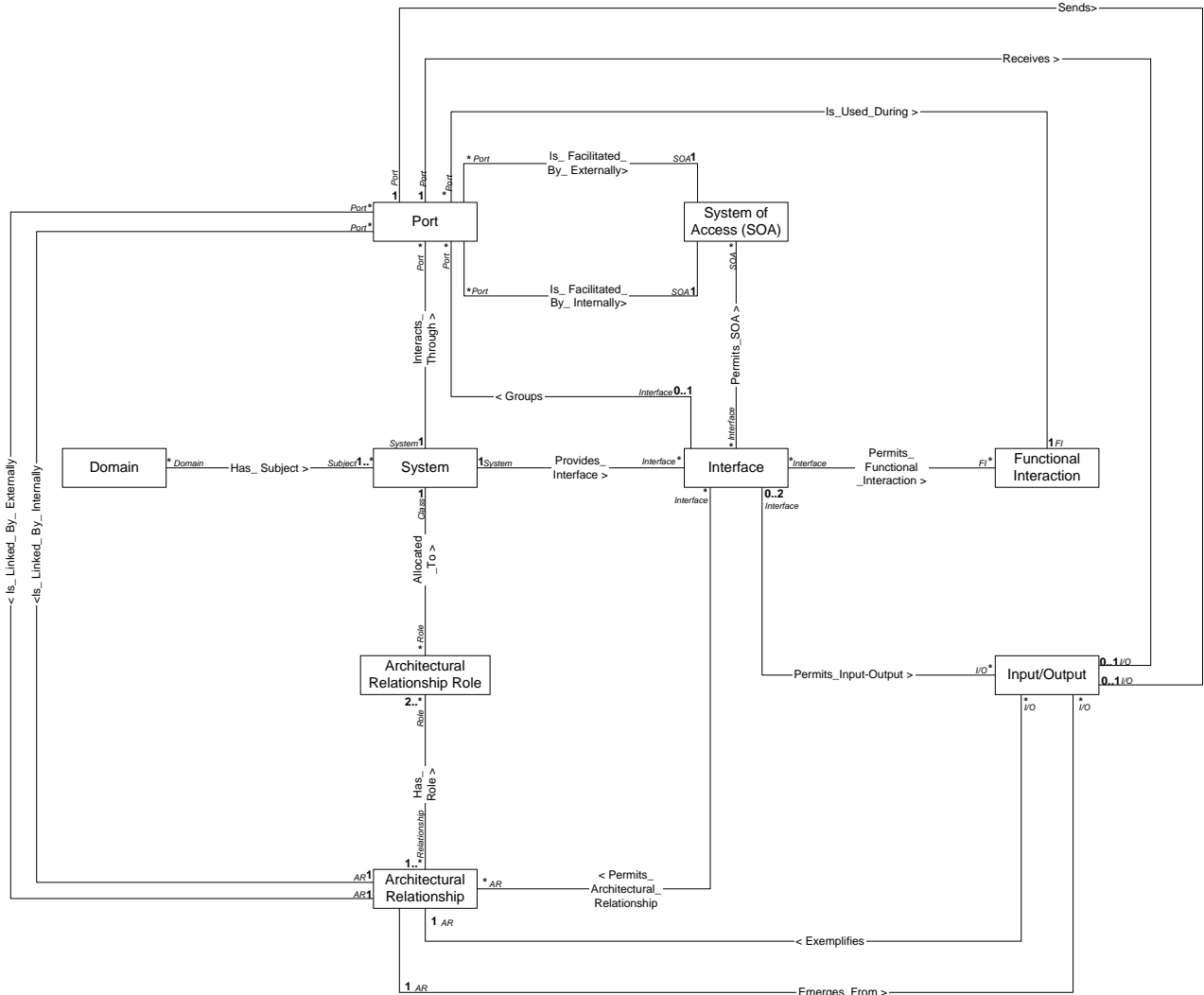


Figure 15: Domain Analysis View



## 2.17 State Analysis View

Figure 17 depicts the classes and relationships modeled to define a system's dynamic behavior using classes such as States, Events, and Functional Interactions.

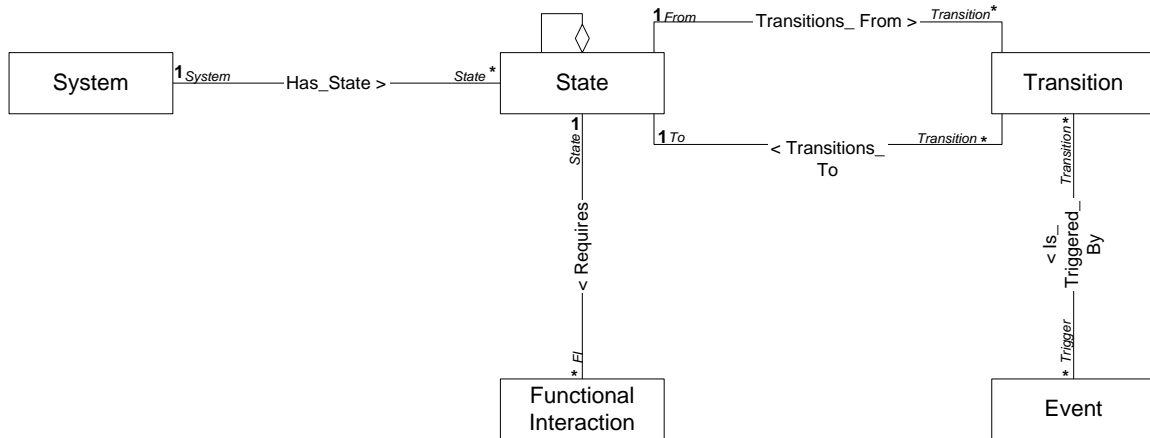


Figure 17: State Analysis View

## 2.18 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.

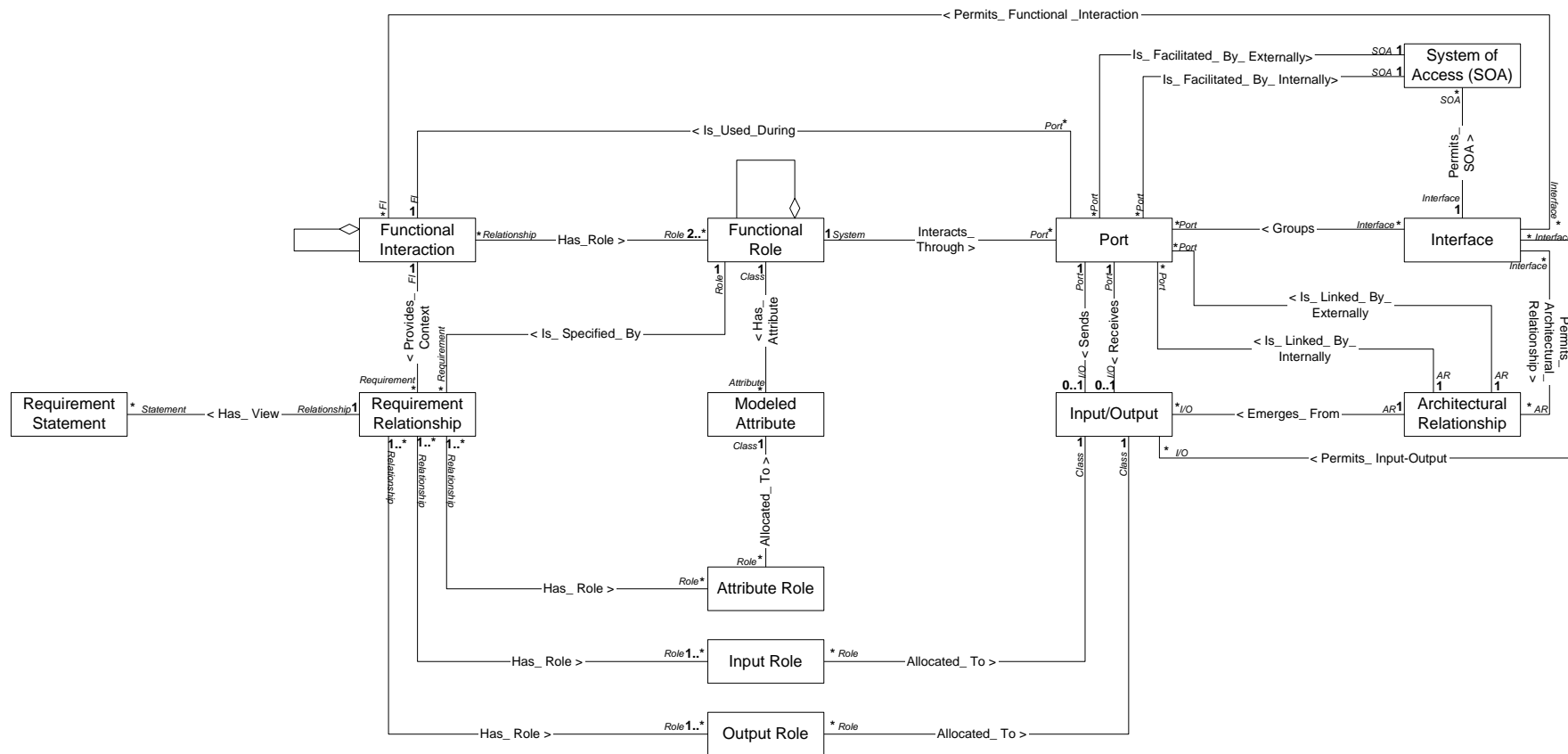


Figure 18: Detail Requirements View

## 2.19 High Level Design View

The High Level Design View, pictured in Figure 19, 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 Physical System 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.

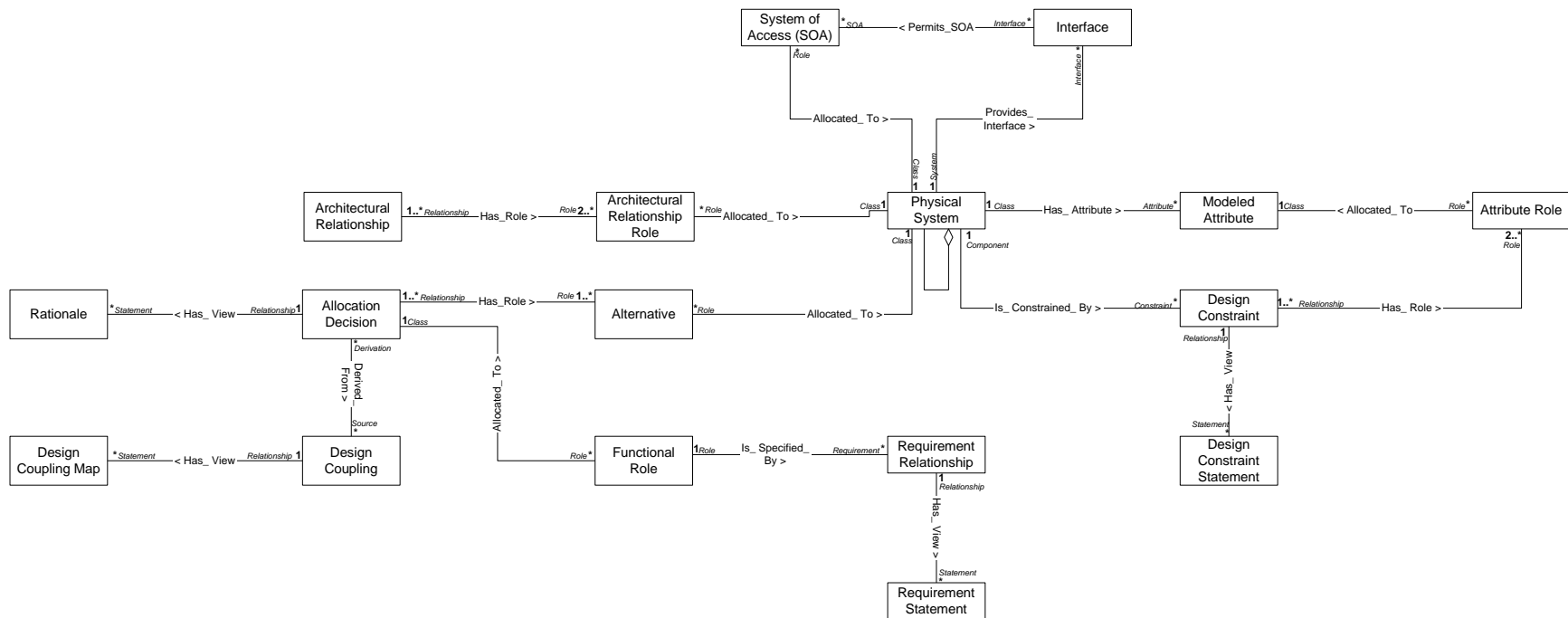


Figure 19: High Level Design View

## 2.20 Summary Pattern Configuration View

The Summary Pattern Configuration View presents how the classes and relationships of a pattern are populated during the pattern configuration process. Once the detailed models in a pattern are established, only three views are visible to the user during the pattern configuration process; the configuration rules embedded in the pattern are used to trace through the models behind the scenes to hide much of the data required during pattern management.

The user's Feature population choices and provided Feature Attribute values are used in conjunction with the configuration rules within the modeled `Uses_Functional_Interaction` relationships to determine how many copies of each Interaction to populate. The configuration rules describe for each Interaction copy what behavior should be populated as well as a set of larger system allocations for that Interaction's roles. Each populated Feature instance is identified uniquely by a Feature Primary Key (FPK). Each populated Interaction instance is identified uniquely by an Interaction Primary Key (IPK), which comprises of the Behavior Primary Key (BPK) and a larger system allocation, called a Role Primary Key (RPK), for each Role of that Interaction

The configuration rules embedded within the `Has_Role` relationships are used to automatically create copies of Roles for each copy of an Interaction and convert the relevant RPK of an Interaction's IPK into an RPK of a newly populated Role. The configuration rules embedded within the `Provides_Context` relationships are used to automatically populate sets of Requirement Statements based on an populated Interaction's BPK. Each Requirement Statement copy is identified with a Requirement Statement Primary Key (RSPK), which is a copy of the entire IPK. The user can then view and manage the populated Roles and Requirement Statements, including setting the values of the Attributes for each Requirement Statement.

The last visible pattern configuration process view allows the user to populate the desired Physical Components and provide values for their Attributes. The Primary Keys for the Physical Components (PCPK) are determined by a combination of user entry and configuration rules embedded in the `Allocated_To` relationships.

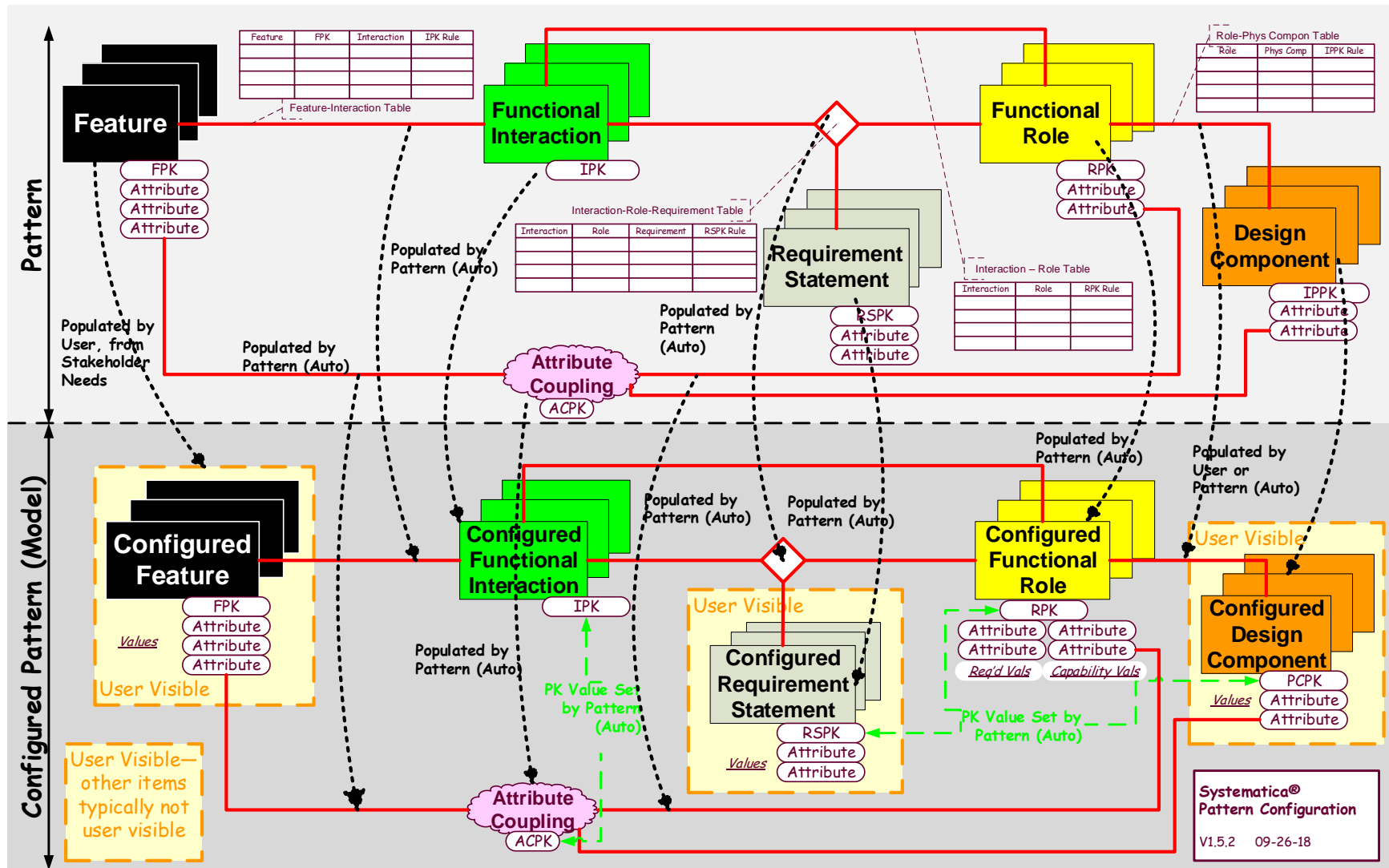


Figure 20: Summary Pattern Configuration View

# 3 Metamodel Definitions

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

## 3.1 Classes

A metaclass models a particular system engineering concept. Classes are related to each other to form complete models of requirements or design using metaclass relationships (see next section). They also have attributes to further tune the modeled concept on an individual basis.

### 3.1.1 Allocation Decision

An Allocation Decision is the relationship between a Functional Role and one or more Physical Systems that may play it. It is the point at which an allocation analysis and decision occurs. Allocation Decisions reference Rationales and score Alternatives, which are the roles the Physical Systems 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 Attributes

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)



- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.1.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 19: High Level Design View](#)

### **3.1.2 Alternative**

An Alternative is the role a Physical System 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 Attributes**

- [Allocated](#)
- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)

- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Rank](#)
- [Score](#)
- [Status](#)
- [Update Version](#)

#### **3.1.2.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 19: High Level Design View](#)

### **3.1.3 Architectural Relationship**

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

#### **3.1.3.1 Aliases**

None

#### **3.1.3.2 Relationships**

- [Emerges From](#)
- [Has Role](#)
- [Is a Type of](#)
- [Is Linked By Externally](#)
- [Is Linked By Internally](#)
- [Permits Architectural Relationship](#)

### **3.1.3.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [Emerges From](#)
- [Exemplifies](#)
- [Has Role](#)
- [ID](#)
- [Is Linked By Externally](#)
- [Is Linked By Internally](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Permits Architectural Relationship](#)
- [Status](#)
- [Update Version](#)

### **3.1.3.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

- [Figure 19: High Level Design 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 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.4.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 7: Architectural Relationship View](#)

- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 19: High Level Design View](#)

### **3.1.5 Attribute Coupling**

An Attribute Coupling is a relationship between two or more Attributes and one or more Attribute Coupling Maps that discuss the relationships between the Attributes.

#### **3.1.5.1 Aliases**

None

#### **3.1.5.2 Relationships**

- [Derived From](#)
- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)

#### **3.1.5.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)

- [Update Version](#)

### **3.1.5.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design 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 relationship between two or more Attributes.

### **3.1.6.1 Aliases**

None

### **3.1.6.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

### **3.1.6.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)

- [Reference](#)
- [Status](#)
- [Update Version](#)

#### **3.1.6.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design 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 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)

- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.7.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.8 Class**

Class is the most abstract class; 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.

#### **3.1.8.1 Aliases**

None

#### **3.1.8.2 Relationships**

- [Appears In](#)
- [Allocated To](#)
- [Contains](#)
- [Derived From](#)
- [Has Attribute](#)
- [Has Previous](#)
- [Has Issue](#)
- [Is a Type of](#)



### **3.1.8.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.8.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 3: General Class View](#)
- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)

### **3.1.9 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 Physical System.

### **3.1.9.1 Aliases**

None

### **3.1.9.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

### **3.1.9.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.9.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 14: Design Coupling View](#)

## **3.1.10 Design Constraint**

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

### **3.1.10.1 Aliases**

None

### **3.1.10.2 Relationships**

- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)
- [Is Constrained By](#)

### **3.1.10.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.10.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 19: High Level Design View](#)

### 3.1.11 Design Constraint Statement

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

#### 3.1.11.1 Aliases

None

#### 3.1.11.2 Relationships

- [Has View](#)
- [Is a Type of](#)

#### 3.1.11.3 Attributes

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

#### 3.1.11.4 Metamodel View References

- [Figure 2: Class Hierarchy View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 19: High Level Design View](#)

### **3.1.12 Design Coupling**

A Design Coupling is a relationship between Attributes of Functional Roles and Physical Systems that play them. One or more Design Coupling Maps can discuss the relationships between the Attributes.

#### **3.1.12.1 Aliases**

- B Matrix Coupling
- Role-Design Component Coupling

#### **3.1.12.2 Relationships**

- [Derived From](#)
- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)

#### **3.1.12.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.12.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)

- [Figure 14: Design Coupling View](#)
- [Figure 19: High Level Design View](#)

### **3.1.13 Design Coupling Map**

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

#### **3.1.13.1 Aliases**

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

#### **3.1.13.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

#### **3.1.13.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

#### **3.1.13.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 19: High Level Design View](#)

#### **3.1.14 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.

##### **3.1.14.1 Aliases**

None

##### **3.1.14.2 Relationships**

- [Appears In](#)
- [Has Subject](#)
- [Is a Type of](#)

##### **3.1.14.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.14.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 3: General Class View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)

### **3.1.15 Domain System**

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

#### **3.1.15.1 Aliases**

None

#### **3.1.15.2 Relationships**

- [Is a Type of](#)

#### **3.1.15.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)



#### **3.1.15.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)

### **3.1.16 Event**

An Event is a subclass of an Information Input/Output that describes an occurrence that triggers a transition from one modeled state to another.

#### **3.1.16.1 Aliases**

None

#### **3.1.16.2 Relationships**

- [Is a Type of](#)
- [Is Triggered By](#)

#### **3.1.16.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.16.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)

- [Figure 17: State Analysis View](#)

### **3.1.17 Feature (Service)**

A Feature is a collection of Functional Interactions having marketable value. Features are also known as a Service in some domains. Features are used to summarize product functionality in value sets or service sets to a customer. Prices are often associated with Features.

#### **3.1.17.1 Aliases**

- Service

#### **3.1.17.2 Relationships**

- [Benefits](#)
- [Has Attribute](#)
- [Has Feature](#)
- [Is a Type of](#)
- [Satisfies](#)
- [Uses Functional Interaction](#)

#### **3.1.17.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)

- [Update Version](#)

### **3.1.17.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 13: Requirements Coupling View](#)

### **3.1.18 Feature Attribute Role**

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

#### **3.1.18.1 Aliases**

None

#### **3.1.18.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

#### **3.1.18.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)

- [Status](#)
- [Update Version](#)

#### **3.1.18.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 13: Requirements Coupling View](#)

### **3.1.19 Functional Interaction**

A Functional Interaction is an interaction of two or more Systems. Interaction means that one system affects the state of another system. All functions are relationships between systems, expressing the externally visible behavioral outcome (requirement) of the interactions. A Function is also sometimes called a Collaboration.

#### **3.1.19.1 Aliases**

- Collaboration
- Contract
- Function (Deprecated)
- Interaction

#### **3.1.19.2 Relationships**

- [Has Role](#)
- [Is a Type of](#)
- [Is Used During](#)
- [Permits Functional Interaction](#)
- [Provides Context](#)
- [Requires](#)
- [Uses Functional Interaction](#)

#### **3.1.19.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)

- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.19.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 17: State Analysis View](#)
- [Figure 18: Detail Requirements View](#)

### **3.1.20 Functional Role**

A Functional Role is the behavioral description (and therefore Logical System) of a part played by (through an Allocation Decision) a System in a Functional Interaction's relationship.

#### **3.1.20.1 Aliases**

- Function
- Function Role
- Role

#### **3.1.20.2 Relationships**

- [Allocated To](#)

- [Contains](#)
- [Has Attribute](#)
- [Has Role](#)
- [Interacts Through](#)
- [Is a Type of](#)
- [Is Specified By](#)

### **3.1.20.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.20.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 16: Logical Architecture View](#)

- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.21 Information Input/Output**

An Information Input/Output is a subclass of Input/Output that represents symbolic information exchanged between interacting systems. Such information is always “about” another System, and has semantic meaning to a Management System (MTS).

#### **3.1.21.1 Aliases**

- Information View (Deprecated)

#### **3.1.21.2 Relationships**

- [Is a Type of](#)

#### **3.1.21.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.21.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)

### 3.1.22 Input/Output

An Input/Output is that which is exchanged between interacting systems.

#### 3.1.22.1 Aliases

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

#### 3.1.22.2 Relationships

- [Allocated To](#)
- [Emerges From](#)
- [Exemplifies](#)
- [Is a Type of](#)
- [Permits Input/Output](#)
- [Receives](#)
- [Sends](#)

#### 3.1.22.3 Attributes

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)



- [Status](#)
- [Update Version](#)

#### **3.1.22.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

### **3.1.23 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.

#### **3.1.23.1 Aliases**

None

#### **3.1.23.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

#### **3.1.23.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)

- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.23.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 18: Detail Requirements View](#)

## **3.1.24 Interface**

An Interface is an association of Input/Outputs, Functional Interactions, Systems of Access (SOAs), and Architectural Relationships through which a system interacts with other systems. Each interface is owned by that system.

### **3.1.24.1 Aliases**

None

### **3.1.24.2 Relationships**

- [Groups](#)
- [Is a Type of](#)
- [Permits Architectural Relationship](#)
- [Permits Functional Interaction](#)
- [Permits Input/Output](#)
- [Permits SOA](#)
- [Provides Interface](#)
- 

### **3.1.24.3 Attributes**

- [Author](#)

- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.24.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

#### **3.1.25 Issue**

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

##### **3.1.25.1 Aliases**

- Action Item
- Open Issue

##### **3.1.25.2 Relationships**

- [Has Issue](#)

- [Is a Type of](#)

### **3.1.25.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

### **3.1.25.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 3: General Class View](#)

## **3.1.26 Logical System**

A Logical System is a system identified based solely upon its required functionality or behavior as seen by external systems interacting with it., and not based upon how it achieves that functionality internally or its physical make-up. Logical systems are typically named and defined without reference to their physical composition, unless (in some cases) this is a part of the external behavior description. Logical Systems are exactly the same as Functional Roles (for some Functional Interaction which may not have been identified), and are candidates to be named as roles in Functional Interaction definitions.

### **3.1.26.1 Aliases**

- Function
- Logical Architecture Component (LAC)

### **3.1.26.2 Relationships**

- [Advocates](#)
- [Allocated To](#)
- [Benefits](#)
- [Contains](#)
- [Has Advocate](#)
- [Has Stakeholder](#)
- [Has Subject](#)
- [Interacts Through](#)
- [Is a Type of](#)
- [Perceives](#)
- [Provides Interface](#)

### **3.1.26.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.26.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 16: Logical Architecture View](#)

#### **3.1.27 Managed System (MDS)**

A Managed System (MDS) is a system which provides a valuable consumable service to a System of Users, and which is planned to be managed with a Management System. An MDSC is a component of an MDS. Examples: A semi-trailer-truck fleet, with individual trucks providing freight transportation services to customers of the trucking company; an electrical power generation system.

##### **3.1.27.1 Aliases**

- Managed System Component
- MDS
- MDSC

##### **3.1.27.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

##### **3.1.27.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)

- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.27.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)

## **3.1.28 Management System (MTS)**

A Management System (MTS) is a system used to manage the performance, fault, configuration, security, or accounting aspects of a Managed System. An MTSC is a component of a MTS. Examples: The Operations Center Systems of a trucking company, used to monitor and manage the fuel economy performance of the total trucking fleet; the Machine Controller of an Electrical Power Generator, used to monitor and control the operation of the Generator.

### **3.1.28.1 Aliases**

- Management System Component
- MTS
- MTSC

### **3.1.28.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)
- 

### **3.1.28.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)

- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.28.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)

### **3.1.29 Manages**

Manages is the central Architectural Relationship in the Embedded Intelligence (EI) pattern. It relates a Management System (MTS), Managed System (MDS), System of Access (SOA), and System of Users (SOU).

#### **3.1.29.1 Aliases**

- Interacts
- Manages For

#### **3.1.29.2 Relationships**

- [Has Role](#)
- [Is a Type of](#)

#### **3.1.29.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)



- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.29.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)

### **3.1.30 Modeled Attribute**

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

#### **3.1.30.1 Aliases**

- [Attribute](#)

#### **3.1.30.2 Relationships**

- [Allocated To](#)
- [Has Attribute](#)
- [Has Value](#)
- [Is a Type of](#)
- [Attributes](#)
- [Author](#)
- [Change Date](#)

- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.30.3 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 3: General Class View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.31 Modeled Relationship**

A Modeled Relationship has 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.

### **3.1.31.1 Aliases**

This class has been reified from actual relationships to allow for clearer modeling. Some examples of such relationships are dsr, mdsr, mtsr, soar, and sour.

### **3.1.31.2 Relationships**

- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)

### **3.1.31.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.31.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)

- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)

### **3.1.32 Modeled Relationship Role**

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

#### **3.1.32.1 Aliases**

None

#### **3.1.32.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

#### **3.1.32.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.32.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)

- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)

### **3.1.33 Modeled Statement**

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

#### **3.1.33.1 Aliases**

- [Statement](#)

#### **3.1.33.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

#### **3.1.33.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)

- [Status](#)
- [Update Version](#)

### **3.1.33.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)

### **3.1.34 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. Features) describing that system.

#### **3.1.34.1 Aliases**

- Informal Need

#### **3.1.34.2 Relationships**

- [Advocates](#)
- [Is a Type of](#)
- [Perceives](#)
- [Satisfies](#)

#### **3.1.34.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Date Submitted](#)

- [Definition](#)
- [Due Date](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Originator](#)
- [Owner](#)
- [Priority](#)
- [Reference](#)
- [Request Type](#)
- [Source](#)
- [Status](#)
- [Update Version](#)

#### **3.1.34.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 4: Feature Framework View](#)

### **3.1.35 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.

#### **3.1.35.1 Aliases**

None

#### **3.1.35.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

### **3.1.35.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.35.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 18: Detail Requirements View](#)

## **3.1.36 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.

### **3.1.36.1 Aliases**

- Physical View (Deprecated)

### **3.1.36.2 Relationships**

- [Is a Type of](#)



### **3.1.36.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.36.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)

## **3.1.37 Physical System**

A Physical System is System defined based upon its identity or physical compositions, but not its behavior. Physical systems may be given proper names, such as names of commercial products, corporate systems, people, organizations, buildings, etc. Physical Systems are Design Components that fulfill the Functional Roles (Logical Systems) allocated to them through an Allocation Decision.

### **3.1.37.1 Aliases**

- Design Component

### **3.1.37.2 Relationships**

- [Allocated To](#)
- [Contains](#)
- [Has Attribute](#)
- [Has Subject](#)

- [Interacts Through](#)
- [Is a Type of](#)
- [Is Constrained By](#)
- [Provides Interface](#)

### **3.1.37.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.37.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 19: High Level Design View](#)

### **3.1.38 Port**

A Port is the coincidence of an Input/Output and System border. A Port is a specific relationship between a received and sent Input/Output, internal and external Systems of Access (SOAs), internal and external Architectural Relationship, and a Functional Interaction.

#### **3.1.38.1 Aliases**

None

#### **3.1.38.2 Relationships**

- [Groups](#)
- [Interacts Through](#)
- [Is a Type of](#)
- [Is Facilitated By Externally](#)
- [Is Facilitated By Internally](#)
- [Is Linked By Externally](#)
- [Is Linked By Internally](#)
- [Is Used During](#)
- [Receives](#)
- [Sends](#)

#### **3.1.38.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)

- [Organization Owner](#)
- [Owner](#)
- [Port Type](#)
- [Status](#)
- [Update Version](#)

#### **3.1.38.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

#### **3.1.39 Rationale**

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

##### **3.1.39.1 Aliases**

None

##### **3.1.39.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

##### **3.1.39.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)

- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

### **3.1.39.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 19: High Level Design View](#)

## **3.1.40 Requirements Coupling**

A Requirements Coupling is a relationship between Attributes of Features and Attributes of Functional Roles. One or more Requirements Coupling Maps can discuss the relationships between the Attributes

### **3.1.40.1 Aliases**

- A Matrix Coupling
- Feature-Role Coupling

### **3.1.40.2 Relationships**

- [Derived From](#)
- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)

### **3.1.40.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)

- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.40.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 13: Requirements Coupling View](#)

### **3.1.41 Requirement Relationship**

A Requirement Relationship is a 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.

#### **3.1.41.1 Aliases**

None

#### **3.1.41.2 Relationships**

- [Has Role](#)
- [Has View](#)
- [Is a Type of](#)
- [Is Specified By](#)
- [Provides Context](#)

#### **3.1.41.3 Attributes**

- [Author](#)
- [Change Date](#)

- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.41.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.42 Requirements Coupling Map**

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

#### **3.1.42.1 Aliases**

- A Matrix Coupling Map
- Feature-Role Coupling Map

#### **3.1.42.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

#### **3.1.42.3 Attributes**

- [Author](#)

- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

#### **3.1.42.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 13: Requirements Coupling View](#)

### **3.1.43 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.

#### **3.1.43.1 Aliases**

- "Shall" Statement

#### **3.1.43.2 Relationships**

- [Has View](#)
- [Is a Type of](#)

#### **3.1.43.3 Attributes**

- [Author](#)
- [Change Date](#)



- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Reference](#)
- [Status](#)
- [Update Version](#)

#### **3.1.43.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.44 Role Attribute Role**

A Role Attribute Role is a Modeled Relationship Role in a Requirements or Design Coupling that specifically references an Attribute of a Functional Role.

#### **3.1.44.1 Aliases**

None

#### **3.1.44.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)

### **3.1.44.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.44.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)

## **3.1.45 State**

A State is a System condition, situation, or mode that has existence for a length of time. The State of a System determines future behavior in which Functional Interactions are to be performed, entered, and exited based upon events. The States of an environmental System of a subject system are use cases for the subject system. During a use case, the subject system is requested to perform certain functions, interacting with the environmental system.

### **3.1.45.1 Aliases**

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

### **3.1.45.2 Relationships**

- [Has State](#)
- [Is a Type of](#)
- [Requires](#)
- [Transitions From](#)
- [Transitions To](#)

### **3.1.45.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.45.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 17: State Analysis View](#)

## **3.1.46 System**

A system is a collection of interacting components. A component can itself be a System, called a sub-system. Information about the purpose or configuration of a system is encoded into the metaclasses associated with the System (e.g., Feature).

### **3.1.46.1 Aliases**

- Actor
- Component
- Subject System

### **3.1.46.2 Relationships**

- [Allocated To](#)
- [Has Attribute](#)
- [Has Feature](#)
- [Has Stakeholder](#)
- [Has State](#)
- [Has Subject](#)
- [Interacts Through](#)
- [Is a Type of](#)
- [Provides Interface](#)

### **3.1.46.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)

- [Update Version](#)

#### **3.1.46.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 17: State Analysis View](#)

#### **3.1.47 System of Access (SOA)**

A System of Access (SOA) is the system which allows other systems to interact (impact each other's state).

##### **3.1.47.1 Aliases**

- SOA
- SOAC
- System of Access Component

##### **3.1.47.2 Relationships**

- [Allocated To](#)
- [Has Role](#)
- [Is a Type of](#)
- [Is Facilitated By Externally](#)
- [Is Facilitated By Internally](#)
- [Permits SOA](#)

##### **3.1.47.3 Attributes**

- [Author](#)
- [Change Date](#)

- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

#### **3.1.47.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### **3.1.48 System of Users (SOU)**

A System of Users (SOU) is the system that consumes services from a Managed System or Management System.

#### **3.1.48.1 Aliases**

- SOU
- SOUC
- System of Users Component

#### **3.1.48.2 Relationships**

- [Allocated To](#)

- [Has Role](#)
- [Is a Type of](#)

### **3.1.48.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.48.4 Metamodel View References**

- [Figure 1: Summary Metamodel](#)
- [Figure 2: Class Hierarchy View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)

## **3.1.49 Transition**

A Transition is the instantaneous switch from one State to another State that has been caused, or triggered, by some Event.

### **3.1.49.1 Aliases**

None

### **3.1.49.2 Relationships**

- [Is a Type of](#)

- [Is Triggered By](#)
- [Transitions From](#)
- [Transitions To](#)

### **3.1.49.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.49.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 17: State Analysis View](#)

## **3.1.50 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 the main means of configuring a pattern.

### **3.1.50.1 Aliases**

None

### **3.1.50.2 Relationships**

- [Has Value](#)



- [Is a Type of](#)

### **3.1.50.3 Attributes**

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)
- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.1.50.4 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Figure 3: General Class View](#)
- [Figure 4: Feature Framework View](#)

## **3.2 Metaclass Relationships**

Metaclass relationships semantically link metaclasses together to create statements about system required behavior, design, or other aspects of interest to system engineering processes. Each relationship has roles that describe a certain concepts classes must fill in order to complete the semantic statement.

### **3.2.1 Advocates**

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

### 3.2.1.1 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.

### 3.2.1.2 Meta-Model View References

- [Figure 4: Feature Framework View](#)

## 3.2.2 Allocated To

The [Allocated To](#) relationship assigns a Class to a Modeled Relationship Role in a Molded Relationship.

### 3.2.2.1 Roles

- Class: The class that plays the role in the relationship. This role is played by [Class](#), [System](#), [Allocation Decision](#), [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.

### 3.2.2.2 Aliases

- Physical View (Deprecated)

### 3.2.2.3 Relationships

- [Is a Type of](#)

### 3.2.2.4 Attributes

- [Author](#)
- [Change Date](#)
- [Change Description](#)
- [Class Level](#)
- [Definition](#)
- [ID](#)
- [Major Version](#)
- [Minor Version](#)
- [Name](#)

- [Organization Owner](#)
- [Owner](#)
- [Status](#)
- [Update Version](#)

### **3.2.2.5 Metamodel View References**

- [Figure 2: Class Hierarchy View](#)
- [Physical System](#), [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](#), [Managed System \(MDS\)](#), [Management System \(MTS\)](#), [System of Access \(SOA\)](#), [System of Users \(SOU\)](#), [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.

### **3.2.2.6 Meta-Model View References**

- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### 3.2.3 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.

#### 3.2.3.1 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.

#### 3.2.3.2 Meta-Model View References

- [Figure 3: General Class View](#)

### 3.2.4 Benefits

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

#### 3.2.4.1 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.

#### 3.2.4.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)

### 3.2.5 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.

#### 3.2.5.1 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.

### 3.2.5.2 Meta-Model View References

- [Figure 3: General Class View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 17: State Analysis View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### 3.2.6 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.

#### 3.2.6.1 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.

#### 3.2.6.2 Meta-Model View References

- [Figure 3: General Class View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 19: High Level Design View](#)

### 3.2.7 Emerges From

The Emerges From relationship links an Architectural Relationship with its summarized Input/Outputs.

### 3.2.7.1 Roles

- AR: The Architectural Relationship resulting from Input/Outputs being transferred between Systems. This role is played by an Architectural Relationship. This role's cardinality is 1.
- I/O: The Input/Output that results in an Architectural Relationship being true. This role is played by an Input/Output. Its cardinality is Many.

### 3.2.7.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.8 Exemplifies

The Exemplifies relationship links an Architectural Relationship to its Input/Outputs that are used to refer to the full set of Input/Outputs summarized by the Architectural Relationship.

### 3.2.8.1 Roles

- AR: The Architectural Relationship referred to by the Input/Output. This role is played by an Architectural Relationship. This role's cardinality is 1.
- I/O: The Input/Output that refers to an Architectural Relationship. This role is played by an Input/Output. Its cardinality is Many.

### 3.2.8.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)

## 3.2.9 Groups

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

### 3.2.9.1 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.

### 3.2.9.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)

- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.10 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..

#### 3.2.10.1 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.

#### 3.2.10.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)

### 3.2.11 Has Attribute

The [Has Attribute](#) relationship links a Modeled Attribute to any Class that has that Attribute.

#### 3.2.11.1 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.

#### 3.2.11.2 Meta-Model View References

- [Figure 3: General Class View](#)
- [Figure 4: Feature Framework View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 18: Detail Requirements View](#)

- [Figure 19: High Level Design View](#)

### 3.2.12 Has Feature

The [Has Feature](#) relationship links a subjects system to a Feature.

#### 3.2.12.1 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.

#### 3.2.12.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)

### 3.2.13 Has Issue

The [Has Issue](#) relationship links an Issue to any class.

#### 3.2.13.1 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.

#### 3.2.13.2 Meta-Model View References

- [Figure 3: General Class View](#)

### 3.2.14 Has Previous

The [Has Previous](#) relationship links a Class to its previous version.

#### 3.2.14.1 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.

#### 3.2.14.2 Meta-Model View References

- [Figure 3: General Class View](#)



### 3.2.15 Has Role

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

#### 3.2.15.1 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, Managed System (MDS), Management System (MTS), System of Access (SOA), System of Users (SOU), 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.

#### 3.2.15.2 Meta-Model View References

- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 8: Embedded Intelligence \(EI\) View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### 3.2.16 Has Stakeholder

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

### 3.2.16.1 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.

### 3.2.16.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)

## 3.2.17 Has State

The [Has State](#) relationship requires that a situation in which a System participates is modeled as a State for that System.

### 3.2.17.1 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.

### 3.2.17.2 Meta-Model View References

- [Figure 17: State Analysis View](#)

## 3.2.18 Has Subject

The [Has Subject](#) relationship links a Domain to a System that is the focus of attention and is being specified.

### 3.2.18.1 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 Physical System. This role's cardinality is 1 to Many.

### 3.2.18.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)

## 3.2.19 Has Value

The [Has Value](#) relationship links a Modeled Attribute to a defined Value it may have.

### 3.2.19.1 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.

### 3.2.19.2 Meta-Model View References

- [Figure 3: General Class View](#)
- [Figure 4: Feature Framework View](#)

## 3.2.20 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.

### 3.2.20.1 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.

### 3.2.20.2 Meta-Model View References

- [Figure 6: Modeled Relationship View](#)
- [Figure 7: Architectural Relationship View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)
- [Figure 19: High Level Design View](#)

### 3.2.21 Interacts Through

The Interacts Through relationship links a System to one of its Ports.

#### 3.2.21.1 Roles

- Port: The Port through which a System interacts. This role is played by a Port. This role's cardinality is Many.
- System: The System that interacts through a Port. This role is played by a System, Physical System, Functional Role, and Logical System. This role's cardinality is 1.

#### 3.2.21.2 Meta-Model View Reference

- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.22 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).

#### 3.2.22.1 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.

#### 3.2.22.2 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: Embedded Intelligence \(EI\) View](#)
- [Figure 9: Functional Interaction View](#)

- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 12: Attribute Coupling View](#)
- [Figure 13: Requirements Coupling View](#)
- [Figure 14: Design Coupling View](#)

### 3.2.23 Is Constrained By

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

#### 3.2.23.1 Roles

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

#### 3.2.23.2 Meta-Model View References

- [Figure 19: High Level Design View](#)

### 3.2.24 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.

#### 3.2.24.1 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.

#### 3.2.24.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.25 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.

### 3.2.25.1 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.

### 3.2.25.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.26 Is Linked By Externally

The [Is Linked By Externally](#) relationship links a Port to the Architectural Relationship that it uses outside of the System boundary.

### 3.2.26.1 Roles

- AR: The Architectural Relationship outside of the System boundary that the Port uses. This role is played by an Architectural Relationship. Its cardinality is 1.
- 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.

### 3.2.26.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.27 Is Linked By Internally

The [Is Linked By Internally](#) relationship links a Port to the Architectural Relationship that it uses inside of the System boundary.

### 3.2.27.1 Roles

- AR: The Architectural Relationship inside of the System boundary that the Port uses. This role is played by an Architectural Relationship. Its cardinality is 1.
- 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.

### 3.2.27.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)

- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.28 Is Triggered By

The transitions relationship describes which Event causes one State to end and another to begin.

#### 3.2.28.1 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.

#### 3.2.28.2 Meta-Model View References

- [Figure 17: State Analysis View](#)

### 3.2.29 Is Specified By

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

#### 3.2.29.1 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.

#### 3.2.29.2 Meta-Model View References

- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### 3.2.30 Is Used During

The Is Used During relationship explains for which Functional Interaction a Port is used by a System.

#### 3.2.30.1 Roles

- FI: The Functional Interaction during which a Port is used. This role is played by a Functional Interaction. This role's cardinality is 1.
- Port: The Port used during the Functional Interaction. This role is played by a Port. This role's cardinality is Many.

### 3.2.30.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.31 Perceives

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).

#### 3.2.31.1 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.

#### 3.2.31.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)

### 3.2.32 Permits Architectural Relationship

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

#### 3.2.32.1 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.

#### 3.2.32.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

### 3.2.33 Permits Functional Interaction

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



### 3.2.33.1 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.

### 3.2.33.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.34 Permits Input/Output

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

### 3.2.34.1 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.

### 3.2.34.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.35 Permits SOA

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

### 3.2.35.1 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.

### 3.2.35.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)

- [Figure 18: Detail Requirements View](#)
- [Figure 19: High Level Design View](#)

### 3.2.36 Provides Context

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

#### 3.2.36.1 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.

#### 3.2.36.2 Meta-Model View References

- [Figure 18: Detail Requirements View](#)

### 3.2.37 Provides Interface

The Provides relationship links an Interface to a System.

#### 3.2.37.1 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 Physical System. Its cardinality is 1.

#### 3.2.37.2 Meta-Model View References

- [Figure 15: Domain Analysis View](#)
- [Figure 16: Logical Architecture View](#)
- [Figure 19: High Level Design View](#)

### 3.2.38 Receives

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

#### 3.2.38.1 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.

### **3.2.38.2 Meta-Model View Reference**

- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

## **3.2.39 Requires**

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

### **3.2.39.1 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.

### **3.2.39.2 Meta-Model View References**

- [Figure 17: State Analysis View](#)

## **3.2.40 Satisfies**

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

### **3.2.40.1 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.

### **3.2.40.2 Meta-Model View Reference**

- [Figure 4: Feature Framework View](#)

## **3.2.41 Sends**

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

### 3.2.41.1 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.

### 3.2.41.2 Meta-Model View Reference

- [Figure 10: Requirement Relationship View](#)
- [Figure 11: Design Constraint View](#)
- [Figure 15: Domain Analysis View](#)
- [Figure 18: Detail Requirements View](#)

## 3.2.42 Transitions From

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

### 3.2.42.1 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.

### 3.2.42.2 Meta-Model View References

- [Figure 17: State Analysis View](#)

## 3.2.43 Transitions To

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

### 3.2.43.1 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.

### 3.2.43.2 Meta-Model View References

- [Figure 17: State Analysis View](#)

### 3.2.44 Uses Functional Interaction

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

#### 3.2.44.1 Roles

- Feature: The Feature whose value is supported by the Functional Interaction. This role is played by a Feature (Service). This role's cardinality is Many.
- Interaction: The Functional Interaction that supports the Feature's value. This role is played by a Functional Interaction. Its cardinality is Many.

#### 3.2.44.2 Meta-Model View Reference

- [Figure 4: Feature Framework View](#)
- [Figure 9: Functional Interaction View](#)
- [Figure 13: Requirements Coupling View](#)

## 3.3 Metaclass Attributes

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

### 3.3.1 Allocated

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

### 3.3.2 Author

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

### 3.3.3 Change Date

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

### 3.3.4 Change Description

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

### 3.3.5 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.

### **3.3.6 Date Submitted**

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

### **3.3.7 Definition**

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

### **3.3.8 Due Date**

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

### **3.3.9 ID**

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

### **3.3.10 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.

### **3.3.11 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.

### **3.3.12 Name**

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

### **3.3.13 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.

### **3.3.14 Originator**

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

### **3.3.15 Owner**

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

### **3.3.16 Port Type**

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

### **3.3.17 Priority**

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

### **3.3.18 Rank**

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

### **3.3.19 Reference**

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

### **3.3.20 Request Type**

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

### **3.3.21 Score**

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

### **3.3.22 Source**

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

### **3.3.23 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.

### **3.3.24 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