Date: July 2018

UML Profile for the Relational Oriented Systems Engineering Technology Trade-off and Analysis Framework (ROSETTA)

FTF Beta 1

OMG Document Number: ptc/2018-07-01

Standard document URL: http://www. omg.org/spec/UPR/PDF

Machine Consumable File(s): https://www.omg.org/spec/UPR/20180219/UPR.xmi

---

This OMG document replaces the submission document (ad/18-02-02, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org December 21, 2018.

You may view the pending issues for this specification from the OMG revision issues web page https://issues.omg.org/issues/lists.

The FTF Recommendation and Report for this specification will be published in March 2019. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to https://issues.omg.org/issues/create-new-issue

1 Scope

1.1 Purpose

This specification of a UML™ profile adds capabilities to UML for a comprehensive facility to structure information in support of model based analysis for architecture optimization and system design. This extension is called UPR: the UML profile for ROSETTA, in reference to its underlying mathematical foundation (the Relational Oriented Systems Engineering Technology Tradeoff and Analysis framework, for which an overview is provided in Clause 6.2). The relational orientation of the framework further supports model transformation between architecture design and detailed system design.

1.2 Challenges being addressed

A comprehensive facility has been lacking to support information structuring and integration for architecture optimization and Constraint-Driven Design analysis across distributed domain experts in a system design team. The facility must also support efficient and effective transformation of the system architecture into detailed system design. In current international standards [REF1], system architecture description is conceptualized using elements and relationships between elements.

1.3 Meeting the challenges

UPR provides an object-oriented facility to structure information in support of a variety of systems analyses for relational oriented architecture description and transformation into design. The core concepts refined in this profile build on existing OMG models and descriptions. The intent is *not* to define new techniques for model based analysis and optimization; but rather to provide new facilities to support analysis. To accomplish this, the profile:

- Supports information structuring and integration for architecture optimization and Constraint-Driven Design analysis
- Provides a facility for identification of conflicting and harmonious Design Objectives
- Implements a mathematically based framework using precise object-oriented syntax and semantics for relational orientation and transformation

The profile facilitates annotation of models to identify information required to perform analysis. The underlying framework enables precise and efficient structuring of the information. No new diagrams are proposed for UPR. The contributors believe that this is a vendor specific concern that would best be served by a longer term RFP, should there be interest, if and when the UPR RFC is finalized.

1.4 Demonstration of the general applicability

UPR should be beneficial to software and system architects in their collaboration with systems engineers and designers. Two case studies will be used to demonstrate the general applicability of the facility:

- Embedded system design and analysis
- Engineering a complex system across distributed domain experts

These case studies are provided in the annexes to the profile. The embedded system case study can be used to demonstrate how the relational viewpoint on models in UPR can be used to provide the software architect and the system engineer an integrated system of models that supports the design and development of a real-time embedded system. The second case study provides an end-to-end example of how UPR can support architecture analysis and Constraint-Driven Design of a complex system across a distributed team of domain experts.

2 Conformance

Conforming implementations of the profile must conform to UML 2.5 (or greater). In addition, conforming implementations must comply with the abstract syntax, i.e., all stereotypes that are defined in the normative clauses, of UPR. A tool demonstrating abstract syntax conformance provides a user interface that enables instances of concrete UPR stereotypes (which are applications of stereotypes to instances of UML metaclasses) to be created, read, updated, and deleted. The tool shall also provide a way to validate the stereotypes against their constraints defined in the profile.

3 References

3.1 Normative References

The following normative documents are referenced in this specification

Reference	Description
[OCL]	Object Constraint Language (OCL™). Available at http://www.omg.org/spec/OCL/
[UML]	Unified Modeling Language™ (UML®). Available at http://www.omg.org/spec/UML/

3.2 Non-normative References

The following informative documents are referenced in this specification:

Reference	Description
[REF1]	ISO/IEC/IEEE 42010:2011, *Systems and software engineering - Architecture description*, Technical Committee ISO/IEC JTC 1/SC 7, December 2011
[REF2]	Mavris, D. N., Griendling, K., & Dickerson, C. E. (2013). Relational-oriented systems engineering and technology tradeoff analysis framework. *Journal of Aircraft*, *50*(5), 1564-1575.
[REF3]	Dickerson, C. E. (2011, June). Mathematical foundations for relational oriented systems engineering (ROSE). In *System of Systems Engineering (SoSE), 2011 6th International Conference on* (pp. 197-202). IEEE.
[REF4]	Dickerson, C. E., & Mavris, D. N. (2011, June). Relational oriented systems engineering (ROSE): Preliminary report. In *System of Systems Engineering (SoSE), 2011 6th International Conference on* (pp. 149-154). IEEE.
[REF5]	Tarski, A. (1954, 1945). *Contributions to the theory of models I, II, III* (*Vol 57*, pp. 572-581, 582-588; *Vol58*, pp. 56-64). Nederl. Aka. Wetensch. Proc. Ser. A.
[REF6]	Bell, J. L. & Slomson, A. B. (1969). *Models and Ultraproducts*. North-Holland / American Elsevier, New York
[REF7]	Dickerson, C. E., & Mavris, D. (2013). A brief history of models and model based systems engineering and the case for relational orientation. *IEEE Systems Journal*, *7*(4), 581-592.
[REF8]	Dickerson, C. E. (2013). A relational oriented approach to system of systems assessment of alternatives for data link interoperability. *IEEE Systems Journal*, *7*(4), 549-560.

[REF9]	Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations* (Vol. 13). Pacific Grove: Brooks/Cole.
[REF10]	Dickerson, C., & Holden, T. (2014). Relational oriented systems engineering framework for flight training. *The Journal of Defense Modeling and Simulation*, *11*(2), 103-113.
[REF11]	Rudakov, S., & Dickerson, C. E. (2017). Harmonization of IEEE 1012 and IEC 60880 standards regarding verification and validation of nuclear power plant safety systems software using model-based methodology. *Progress in Nuclear Energy*, *99*, 86-95.
[REF12]	Dickerson, C. E., Ji, S., & Roslan, R. (2016, June). Formal methods for a system of systems analysis framework applied to traffic management. In *System of Systems Engineering Conference (SoSE), 2016 11th* (pp. 1-6). IEEE.
[REF13]	Akao, Y. (1990). *Quality function deployment*. Cambridge, MA: Productivity Press
[REF14]	ISO/IEC/IEEE 15288:2015, *Systems and software engineering -- System life cycle processes*, Technical Committee ISO/IEC JTC 1/SC 7, May 2015

4 Terms and Definitions

This clause provides the definitions and explanation of the terms and concepts used in the UPR. To assist the readers to have a collective understanding of these terms, they are not organized in the conventional alphabetical order.

Constraint-Driven Design	A system design approach in which design constraints are a central concern. A solution obtained from such a design approach aims to satisfy all design constraints.
Variable	An attribute (used to describe system property or parameter) that can take on a range of values.
Mathematical Constraint	A mathematical relation that associates a variable with a constant value in a way that limits the range of values of the variable, e.g. with a comparison operator to express equalities and inequalities.
Design Objective	A measurable attribute, such as performance or a physical property that can be expressed as a variable and used to compare systems.
Design Objective Constraint	A mathematical constraint applied to a Design Objective.
Design Variable	A variable associated with a system that is used to define the system; and is under the design authority of the engineer.
Design Variable Constraint	A mathematical constraint applied to a Design Variable. It is further categorized into Design Range and Feasible Range in UPR.
Design Range	A defined range of values of a Design Variable; mathematically, this is the domain of the variable.
Design Space	A (multi-dimensional) Space formed by the collective Design Ranges.
Feasible Range	The Feasible Range refines the Design Range of a Design Variable to provide a subset of feasible solutions. In Constraint Driven Design this subset of the Design Space is implied by a given set of Design Objective Constraints.
Feasible Design Space	A (multi-dimensional) Space formed by Feasible Ranges.
Sensitivity	A mathematical-based relation between two variables that indicate how the variation in one variable affects the variation in the other variable. In UPR, Sensitivity is specified through using Polynomial coefficients.
Polynomial	An expression consisting of variables and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents of variables. e.g. $x^2 + 3y^4 + z$. A Design Objective, within the scope of UPR, can be expressed as a univariate polynomial function of a Design Variable.
Univariate Polynomial	A polynomial that has only one variable, e.g. $1 + x + x^2 + 3x^4$.
Relational Structure	A defined collection of relations.
Relational Transformation	Relational Transformation refers to transformations that preserve Relational Structure. There are two types of Relational Transformation defined and specified in this profile: Binary (Clause 6.2.4) and Unary (Clause 6.2.5).
Matrix Representation	A Matrix Representation in this profile refers to a graphical representation that the structure of which can be represented by a mathematical n-by-m matrix.

5 Symbols

Acronym	Name, description
DO	DesignObjective (acronym used to shorten certain stereotype names)
DoE	Design of Experiment
DV	DesignVariable (acronym used to shorten certain stereotype names)
DPF	Diesel Particulate Filter
ELM	Element (refer to system element, acronym used to shorten certain stereotype names)
FE	Fuel Economy
MARTE	Modeling and Analysis of Real-Time Embedded Systems
MBSE	Model Based Systems Engineering
MDS	Minimum Detectable Signal
MTL	Minimum Triggering Level
OCL	Object Constraint Language
PM	Particulate Matter
QFD	Quality Function Deployment
ROSETTA	Relational Oriented System Engineering Technology Trade-off Analysis
RT-B	Binary Transformation
RT-U	Unary Transformation
UML	Unified Modeling Language
UPR	UML Profile for ROSETTA
SysML	Systems Modeling Language

6 Additional Information

6.1 Related Standards

UPR is related to the following normative OMG specifications:

- UML: Like any other UML profile, UPR is defined on the basis of the metamodel provided in the UML 2.5 Specification.
- OCL: UPR uses OMG's Object Constraint Language (OCL) to precisely formalize the constraints stated in natural language for the abstract syntax.

6.2 Background for Constraint-Driven Design

6.2.1 ROSETTA Framework

UPR is an object-oriented implementation of an underlying mathematical formalism that is referred to as the Relational Oriented Systems Engineering Technology Tradeoff and Analysis framework [REF2]. When the term *framework* is used in association with ROSETTA, it will refer to the mathematical formalism; whereas when the term *profile* is used, it will refer to an object-oriented implementation. One of the challenges of developing UPR has been to specify the profile in a way that is consistent with the underlying mathematical concepts.

This clause offers a brief background on the underlying formalism and its relation to mathematical modeling and systems engineering. When reading this 6.2.1 clause, it is important to keep in mind that the framework is written in the language of mathematical logic and set theory. The conceptualization that follows must be read from this point of view if the reader is to understand it. The concepts in the subsequent clauses in the RFC are written in the language of UML.

The framework is a model based formalism that integrates and extends the concepts of relational structures and relational homomorphism in universal algebra to systems engineering [REF3, REF4]. It admits a graphical interpretation that lends itself to the graphical representations of models in UML and SysML. The formalism of the framework is rooted in the Tarski theory of models in mathematical logic [REF5, REF6] but has been adapted to the practice of engineering [REF2, REF4]. Put succinctly, a model in the Tarski theory is defined to be a relational structure (in the sense of mathematical set theory) that is the image of an injective mapping of one or more sentences in the first order predicate calculus (a sentence being a fully quantified well-formed formula). In other words, the model 'faithfully' realizes the sentences. This concept of *model* in mathematical logic is similar to a PIM – PSM transformation in MDA™ for software development.

The ROSETTA formalism further supports the rigorous development of structures for the design of systems and the assemblage of systems of systems that mathematically interpret the methods of Nam Suh on Axiomatic Design theory. It is also similar to the Quality Function Deployment (QFD) House of Quality; but can translate expert opinion into mathematical relations [REF2, REF7]. For this reason, the name of the (mathematical) framework bears an intentional similarity to the Rosetta stone which provided the means to interpret between the Greek, Egyptian and Hieroglyphics demotic languages. Similarly, ROSETTA provides a facility for interpretation of concepts between systems engineers and software and system architects.

The relational oriented viewpoint on systems supports a general systems methodology that employs a principle of *model specification and relational transformation* for the purpose of system description, analysis, and design. As summarized in [REF8], in the relational viewpoint, the specification of a model associated with a system is the specification of:

- Entities associated with the system
- Sentences (declarations) about the entities
- Modeling elements to instantiate the sentences
- A semantic structure on the modeling elements
- Interpretations of the sentences into the semantic structure

Entities are abstractions that admit logical or physical existence. The entities of the system can include attributes, classes (components), and the functionality of the system. There can also be entities associated with the system which are not part of it, e.g. the operating environment and context of the system.

The sentences are the basis for system specification. A system model is valid when the interpretation of each sentence is true within the semantic structure of the model. The validation process is facilitated by two types of semantic structures: relational structures (i.e. a collection of mathematical relations) and graphical representations (e.g. an Activity diagram or a Conceptual Graph). A *graphical representation of a model* is a collection of vertices and edges for encoding the semantic information captured by the sentences. The modeling elements in this case are the vertices. The edges, which represent relationships between vertices, are represented as pairs of vertices. The underlying structure is a (discrete) graph which therefore also has a matrix representation.

Semantic structure is a concept which seeks to formalize the intended meaning of natural language through some type of mathematical organization. It can be thought of as structured knowledge about the system.

The term *structure* is offered without definition but is assumed to have locations and relations between the locations. A predicate in the first order logic is an example of a structure. The symbols for the variables and constants are locations (for information/ interpretation) and the Predicate Letter is the symbol for a relation between the variables and constants. By itself, a predicate is not semantic; it requires interpretation, e.g. into a relational structure. The combination of a predicate sentence with knowledge (e.g. ontology) can provide a language that can express relationships about the entities of a domain of interest [REF9]. An architectural domain relates to a type of knowledge about the system or to one or more system components.

Relational frames are specified as ordered pairs (*M*, R). The modeling elements are specified by *M* and the semantic structure is specified by R. This type of frame provides a static structure for organizing knowledge about the system using predefined internal relations specified by the model that reflect the relational structure of the semantic information captured by the sentences and their interpretation into the structure. This follows a line of reasoning for knowledge representation similar to the use of *frames* to represent structure as expressed in [REF9].

This is also a mathematical representation of knowledge similar to the concept of object oriented frames used in software engineering, which is primarily concerned with the classes and objects of software architecture, their responsibilities and collaborations, and the threads of control. The mathematical representation in relational orientation admits matrix representation that can be used by systems engineers in practical ways to employ the concepts of relational structures and relational homomorphism. If (*M*, R) is a relational frame, then a matrix representation of the frame will be denoted as M. Similarly, for another frame, (*N*, S), the notation would be N.

Model elements are related by belonging to: (i) n-ary mathematical relations, (ii) hierarchical decomposition, or (iii) association with elements of another model by transformation. The special case n = 1 for n-ary relations is called a *unary relation*, i.e. a defined subset of elements. The case n = 2 is a *binary relation*. The types (i) and (ii) of mathematical relation are concerned with the internal structure of a model. The associated relational frames will be referred to simply as *frames*. The third type of relation is an association external to a model, although hierarchical decomposition into sub-models can also admit transformations. The type (iii) frame will be referred to as a *transformational frame* and denoted as **Q** = (*M*, *N*; Q) where *M* and *N* are the model elements of two models and Q is an association between the elements of the frames. A matrix notation for the transformational frame **Q** will be denoted as Q. Note that although the frame is an ordered triple, the association Q is not preferential to the direction of transformation.

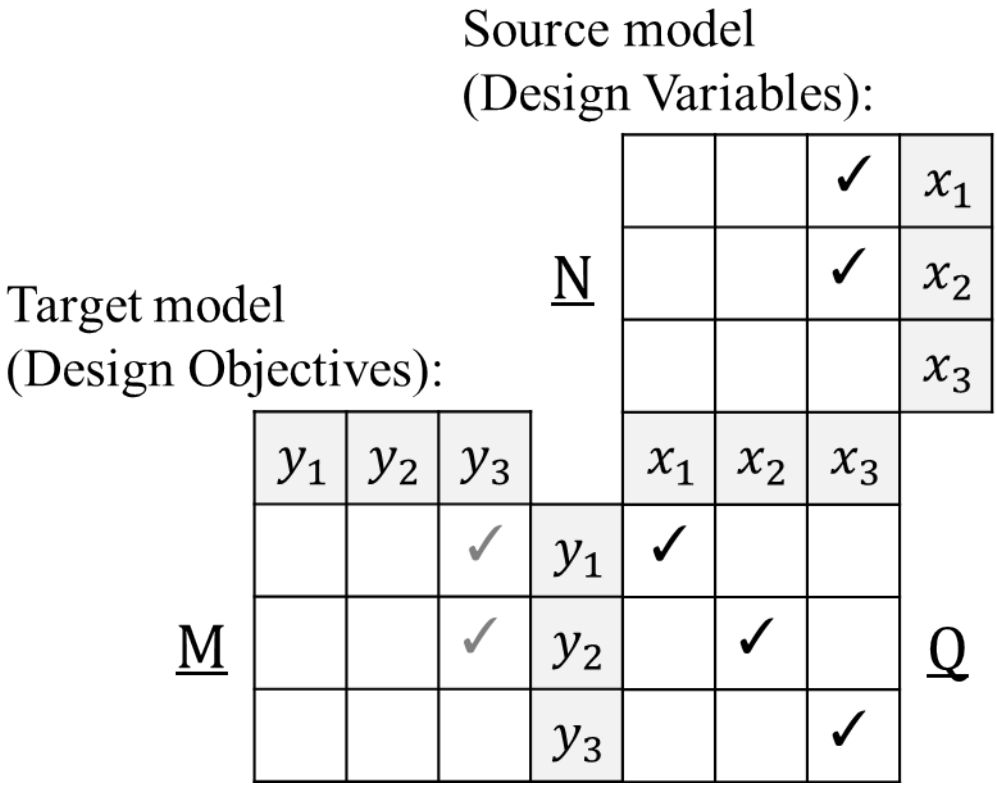

Figure 6-1 Simple example of a Transformational Frame (Q) in matrix form

A simple transformational frame in matrix form is depicted in Figure 6-1. The symbols y_1, y_2, and y_3 in this example represent Design Objectives. Similarly, x_1, x_2, and x_3 represent Design Variables. In an automotive emissions problem, the objectives might be metrics for fuel economy, particulate matter (in the emissions), and the efficiency of a Diesel Particulate Filter (DPF). On the other hand, the mathematical modeling elements of a system specification might be metrics associated with two functionalities of the system, such as operating the engine at a high temperature and burning the soot; while the third variable could be associated with a choice of DPF technology. Two of the three frames in matrix form, i.e. N and Q, in the figure are therefore specified explicitly. The semantic of the (x_i, x_j) binary relation is that x_1 has a dependency on x_j. For example, (x_2, x_3) means the burning of soot depends on the DPF technology. Similarly, the Q frame states that the Design Objectives depend on the Design Variables. For example, (y_3, x_3) means that the DPF efficiency depends on the DPF technology.

We shall see that a semantic structure on the remaining frame (M) is implied by the other two (c.f. Clause 6.2.4). Specifically this structure is the pair of binary relations in the matrix M depicted in Figure 6-1.

A *relational transformation* is specified as an association between the elements of two models of a system that induces a mapping between the relationships expressed in the models and preserves the structure of these relationships. For example, a relational transformation of a graph can be understood as a multi-valued association of the vertices of the graph (of a source model) that preserves the relationships of the edges to the vertices (in the target model).

An example can be seen in the emissions example, where the association Q when viewed as an injective mapping from *N* to *M*, gives rise to the two new binary relations in M. Thus, dependencies between the Design Variables give rise to dependencies in the Design Objectives, which otherwise had been considered to be independent. Note that these relations of dependency could be specified by expert opinion (as with QFD [REF13]) or could be derived using chain rule calculations for multivariate calculus (c.f. Clause 6.2.4).

Note that the figure is the form of the structures and transformations used in Tarski model theory. In the figure, the source model represents the binary relations that would be the predicates represented by the matrix N. The Q matrix is injective. The result of the transformation would be a matrix representation of the binary structure of the mathematical relation into the 'target model' that images the predicates in the 'source model'.

When two models of a system are related by a relational transformation, the collective three frames will be referred to as a *framework*. The M-Q-N structure of the example in the figure is representative of one form of a ROSETTA framework.

The matrix representation provides a mathematical facility to structure knowledge for a variety of systems analyses. Reference [REF2] is an example in aerospace; [REF8] is an example in data links; [REF10] is in flight training; [REF11] is commercial nuclear standards; and reference [REF12] is an example of emerging formal methods for safety analysis in urban traffic management systems.

In relational orientation, systems are modeled using multiple frameworks, which represent the various knowledge domains and components of the system. Frameworks are integrated into a *framework structure* by sharing common frames or by transformations between frames. The specification of frameworks is complete when they form a framework structure that is adequate for system specification, analysis and design. This resultant framework structure provides an abstraction that can be used for specifying and integrating the models of the system.

6.2.2 Mathematical-based Constraint Modeling

In the context of engineering, constraints can be classified into two types. On the one hand, there is the type of constraints that can be expressed mathematically by equalities and inequalities, e.g. the mass of a laptop must not be greater than two kilograms, which can be mathematically written as $M \leq 2kg$. On the other hand, there is the other type of constraints that cannot be easily expressed mathematically, e.g. the component must be in the shape of a triangle. UPR models mathematical-based constraints, referred to as *Mathematical Constraints*, using the following generic mathematical relation: 'a variable' is related to 'a value' by a 'comparison operator', where:

- A *Variable* represents the element being constrained. It can take on any real number when being unconstrained.
- A *Comparison Operator*, such as '=' and '<', are mathematical symbols defined to indicate equality and inequalities.
- A *Value* is a real number. In UPR, it is unitless.

The example, $M \leq 2$, without the unit being specified, is an example that follows the above construct[1]. There can be multiple constraints on one variable. For instance, as mass is always positive, this creates an additional constraint on M, which says, $M \geq 0$. Although, mathematically, it is possible to combine these two constraints

[1] Note that the unit here is an intrinsic property of the element that the variable represents, and can be modeled through other means than UPR, for example, using the SysML UnitAndQuantityKind model library.

into a single mathematical expression as $0 \leq M \leq 2$. However, this does not follow the construct adopted by UPR. In this case, UPR requires two constraints to be defined separately on the same variable.

To facilitate the modeling of Constraint-Driven Design (CDD) problem, UPR focuses on developing a facility to accommodate the modeling of mathematical-based constraints in a structured way. In brief, Mathematical Constraints are divided into two types:

- *Design Objective Constraints*: These are constraints on Design Objectives. An example of a Design Objective Constraint can be 'a vehicle must achieve a top speed of greater than 100 miles per hour (mph)'. A *Design Objective* is measurable attribute, such as performance or a physical property that can be expressed as a variable and used to compare systems. Sometimes, Design Objectives are to be optimized. For example, the cost of the system to be designed needs to be minimized, or the power output of the system to be designed shall be maximized. In these cases, there are no limits (constraints) specified for these Design Objectives. Hence, they can be considered as unbounded or unconstrained. CDD is only concerned with Design Objectives that are constrained. However, it is worth noting that adopting a CDD approach to system design is not in conflict with design approaches such as design optimization techniques.

- *Design Variable Constraints*: A *Design Variable* is a variable associated with a system that is used to define the system; and is under the design authority of the engineer. Using the examples stated above, system designer may want to consider finding combinations of engine power and engine torque to achieve the desired top speed. In this case, engine power and engine torque can be defined as Design Variables. A Design Variable Constraints is simply a constraint applied to the Design Variable that limits the possible design choices on that Design Variable. Further details on the classification of Design Variable Constraints are discussed in sub clause 6.2.5.

It is worth noting that although we have divided Mathematical Constraints into two types, there is not a fine line between them unless a CDD problem is formulated. Design Variable Constraint in one CDD problem may become a Design Objective Constraint in another CDD problem. This is because a Design Variable after being specified in one design problem may become a Design Objective (a requirement) in the design of another system or subsystem. This is often seen in engineering complex system that has multiple hierarchical levels. For instance, following the example stated above, after the engine power and engine torque have been specified by the vehicle designer, this may then become a Design Objective Constraint (requirement) for the engine designers and manufacturers.

6.2.3 Mathematical-based Relation Modeling

In the context of engineering, relations between variables can sometimes be quantified mathematically. Where an exact mathematical relation between two variables is known, it is possible to express such a relation using analytics. For example, based on Newton's 2nd Law, to achieve a desired acceleration, a, an object with mass, m, would need a net force of, $F = ma$. Where an exact mathematical relation is unknown, there may be a need to express a qualitative representation of the relations between the variables using a subjective scale. For instance, using the correlation concept used in Quality Functional Deployment (QFD) [REF13], on a scale from -3 to +3, it is possible to express two variables having a weak positive correlation by a correlation value, +1; and two variables having a strong negative correlation by a correlation value, -3.

In either of the above cases, the relation between two variables can be expressed by an abstract expression, referred to as the *Sensitivity* in UPR. In general, Sensitivity is understood as a mathematical-based relation between two variables that indicate how the variation in one variable affects the variation in the other variable. For two generic variables, *x* and *y*, partial differentiation can be used to calculate the Sensitivity between them as,

$$\text{Sensitivity} = \frac{\partial y}{\partial x}$$

Equation 6-1[2]

For instance, based on Newton's 2nd Law, the Sensitivity between the forces required to increase the velocity of a mass at a constant acceleration of $5ms^{-2}$ can be calculated as,

[2] Note that the partial derivative becomes exact if y only depends on x

$$\text{Sensitivity} = \frac{\partial F}{\partial m} = a = 5\text{ms}^{-2}$$

Equation 6-2

For the case of using the subjective correlation in using QFD technique, the Sensitivity is equivalent to the defined correlation value, e.g. Sensitivity = 3 for two strongly correlated variables.

Sometimes, the mathematical relation between two variables *y* and *x* can be non-linear. In this case, the Sensitivity value, as calculated by using Equation 6-1, will be dependent on the value of *x*. For example, let us consider a 2nd order polynomial of the form,

$$\text{y} = 2x + 3x^2$$

Equation 6-3

Using Equation 1, it can be concluded that at $x = 1$, Sensitivity is 8; while at $x = 2$, Sensitivity is 14. Due to the varying Sensitivity, differentiating the polynomial does not provide a useful result. However, by keeping the coefficients in the polynomial, Sensitivity can be obtained as long as these coefficients are available. For an n-th order polynomial that reads,

$$\text{y} = c_0 + c_1 x + c_2 x^2 + \cdots = \sum_{i=0} c_i x^i$$

Equation 6-4

the Sensitivity can be generically expressed as,

$$\text{Sensitivity} = \frac{\partial y}{\partial x} = c_1 + 2c_2 x + \cdots = \sum_{i=1} i c_1 x^{i-1}$$

Equation 6-5

In addition to facilitating the modeling of single valued Sensitivity, UPR thereby also aims to facilitate the modeling of the coefficients in polynomials of arbitrary degree. A single valued Sensitivity, such as the ones in previous examples using QFD correlations, can be approximated as a 1st order polynomial with $c_0 = 0$ and c_1 being the Sensitivity specified. This then allows the modeling of Sensitivity in the two different cases to be integrated by using polynomial coefficients alone.

6.2.4 Binary Transformation for Dependency Analysis

In systems engineering and design, knowing the relationships between Design Objectives can be useful. For example, this knowledge is essential in determining what tradeoff analysis will be most valuable. However, without empirical evidence or detailed knowledge of the system, it is almost impossible to determine which of these relationships (conflicting, harmonious, or independent) are present. In SysML, Design Objectives can be modeled using Requirements in a SysML Requirement diagram. To better facilitate the modeling of requirement relations, SysML further extends UML Dependency to include additional Dependency types such as Verify and Refine. These Dependency types are very useful in representing requirement traceability. However, neither UML nor SysML are capable of expressing that harmonious, conflicting, or no significant relationship exists between two Design Objectives. UPR aims to provide a facility to model these relationships as well as facilitate transformations that can be used to identify these relationships.

Where prior knowledge of harmony, conflict and independence is not available, a means to establish this knowledge is of great value. Sensitivities introduced in the UPR shall facilitate the identification of harmonious and conflicting relations between Design Objectives. Although identification method is not part of the UPR profile, a particular method, referred to as *Binary Transformation* (RT-B), is introduced in the rest of this sub clause to show how UPR can facilitate this transformation for the identification of relationships between Design Objectives. RT-B is originated from the ROSETTA Framework [REF2]. And note that this method is informative only. The principle of the stereotypes defined in UPR is to support such an identification method.

In general, a relational transformation is specified as an association between the elements or parameters of two models of a system, that induces a further mapping between the relationships in the models. The mathematical formalism of RT-B can be expressed as,

$$(x_i, x_j) \in \mathbf{N} \text{ with } (x_i, y_k), (x_i, y_l) \in \mathbf{Q} \text{ implies} (y_k, y_l) \in \mathbf{M}$$

Equation 6-6 Binary Relation Transformation

where x_i and x_j are used to denote two system elements or parameters; and y_k and y_l are used to denote two Design Objectives. Then, Equation 6-6 says that if the two system elements (parameters) are related as captured in a relational set, ***N***; and if the system element x_i relates to the Design Objective y_k and that the system element x_j relates to the other Design Objective y_l, which are both captured in a relational set, **Q**; then there is an *implied* relation between the two Design Objectives, defined as $M \equiv N\mathrm{Q}$.

Using the RT-B, it is then possible to identify whether two Design Objectives are related or not by tracing their relationships to system elements or parameters. The semantic meaning, i.e. the type of the dependency (relation), of the implied relation between the Design Objectives, depends on the semantic meanings of the existing relations used. Here, as we are interested in whether the related Design Objectives are mathematically harmonious, conflicting or independent, we can use the concept of Sensitivity introduced in Clause 6.2.2. For example, for the relation between Design Objective, y_k and system element, x_i , we have

$$\text{Sensitivity} = \frac{\partial y_k}{\partial x_i}$$

Equation 6-7

Using the same concept, it is possible to mathematically determine the Sensitivity between the two related Design Objectives by

$$\frac{dy_k}{dy_l} = \frac{\partial y_k}{\partial x_i}\frac{\partial x_i}{\partial x_j}\frac{\partial x_j}{\partial y_l} + \frac{\partial y_k}{\partial x_j}\frac{\partial x_j}{\partial x_i}\frac{\partial x_i}{\partial y_k}$$

Equation 6-8

If this equation yields a positive value, a (mathematically) harmonious relation is identified; while if it yields a negative value, a (mathematically) conflicting relation is identified. Sometimes, it can be useful to normalize the yield values into a unitless scale for the sake of comparing the strength of relations among Design Objectives.

Again, the above transformational method is not normative. Users of UPR can use any method to conduct the dependency analysis by using the information modeled. The stereotypes introduced in UPR facilitate transformational methods of similar kinds to achieve the identification of harmonious and conflicting relations between Design Objectives.

6.2.5 Unary Transformation to determine Feasible Designs

When a Sensitivity (in terms of polynomial coefficients) is defined between a Design Objective and a Design Variable, e.g. as captured by 1st order polynomial, on the one hand, if a design specification is provided, then it is possible to verify the design against the Design Objective Constraints by substituting the current design value (as an initial design) into a polynomial equation formed by the specified polynomial coefficients. On the other hand, it is of interest to system designers that if Sensitivity can be used inversely to obtain design solutions that satisfy a given set of Design Objective Constraints. We introduce two types of Design Variable Constraints: namely Design Range and Feasible Range to explain this inverse mapping.

The following simple example illustrates how a Design Objective Constraint imposes a constraint on a Design Variable. Let us consider a Design Objective, y, is constrained by two Design Objective Constraints, $y < 10$ and $y > 4$. This Design Objective is related to a Design Variable by a first order polynomial such that,

$$y = 2x$$

Equation 6-9

The Design Variable has a physical limit such that it can only be positive numbers and has to be less than 8. These can be expressed as two Design Variable Constraints, $x > 0$ and < 8 . These types of constraint are referred to as *Design Ranges*. Constraints on Design Variables that are derived from empirical knowledge or previous designs are also considered as *Design Range*. Basically, the collective Design Ranges on a Design Variable defines the solution search space. For multiple Design Variables, the collection of all Design Ranges is referred to as a *Design Space*.

Now, substituting the two Design Objective Constraints into the polynomial, it can be determined that the Design Variable must be less than 5, i.e. $x < 5$ and at the same time, greater than 2, i.e. $x > 2$, in order to not violate the Design Objective Constraints. From the above calculation, it is realized that Design Objective Constraints can impose new constraints on Design Variables. The type of these new constraints on the Design Variable is referred to as *Feasible Range*. The word 'feasible' is used to indicate that design solutions lie within the Feasible Range, which will satisfy the Design Objective Constraints. For multiple Design Variables, the

collection of all Feasible Ranges is referred to as a *Feasible Design Space*. For a multi-dimensional Design Space, a continuous Feasible Design Space may not be unique.

Depending on the detailed specification of the constraints in the CDD problem, through an inverse mapping, a Design Range can sometimes become a Feasible Range. In this case, instead of revising the Design Range into a Feasible Range, UPR recommends keeping the original Design Range and defining a new Feasible Range. This is particularly useful for having a multi-dimensional Design Space, where Feasible Design Spaces, as discussed, may not be unique.

To help visualize the above solved CDD problem, Figure 6-2 depicts the inverse mapping, while the other information relevant to this CDD problem are captured in Table 6-1.

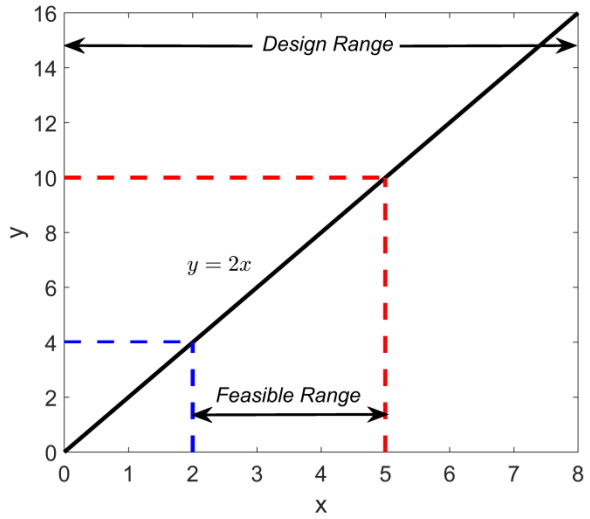

Figure 6-2 An inverse mapping: from Design Objective Constraints into Feasible Ranges (Design Variable Constraints)

Design Objective	Design Objective Constraints	Polynomial Relation	Design Variable	Design Variable Constraints	
				Design Ranges	Feasible Ranges
y	$y < 10$	$y = 2x$	x	$x > 0$	$x < 5$
	$y > 4$			$x < 8$	$x > 2$

Table 6-1 Design information related to the CDD problem

An inverse mapping that allows the transformation of constraints on Design Objective into constraints on Design Variables (Feasible Ranges) is referred to as a *Unary Transformation* (RT-U) in UPR. For complex CDD problems that involve multi-Design Objectives and multi-Design Variables, inverse mapping is no longer straightforward, while the determination of Feasible Ranges requires more advanced design algorithms based on the concept of RT-U. Nonetheless, UPR aims to facilitate the modeling and structuring information required to conduct advanced transformation algorithms, in order to determine Feasible Ranges.

6.3 Language Architecture

Figure 6-3 depicts the relation of UPR to OMG Specifications.

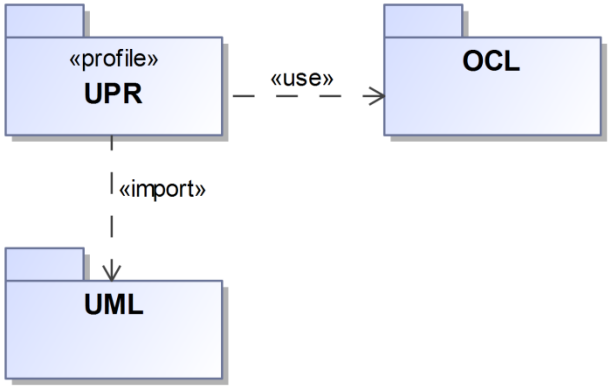

Figure 6-3 UPR Language Architecture

UPR consists of five normative packages: Foundations, Operators, Design Objective Constraint, Design Variable Constraint, and Relational Structure and Design packages. The five sub profiles are on the same hierarchical level.

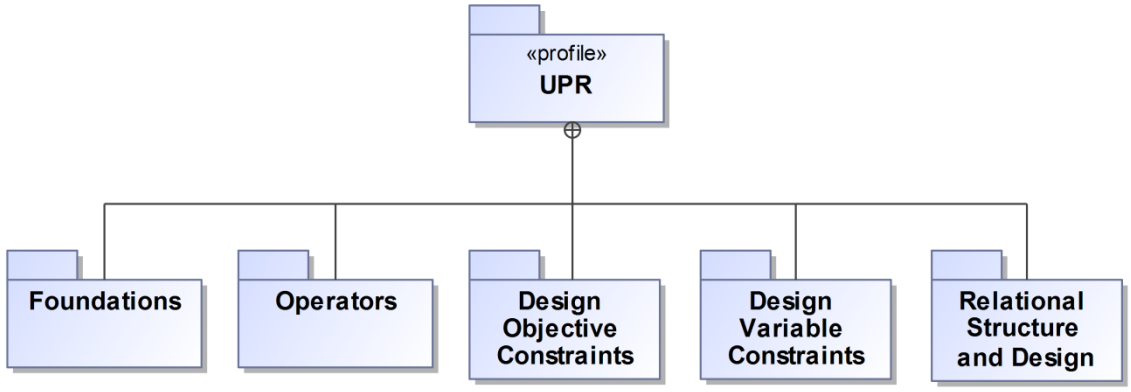

Figure 6-4 UPR Profile Packages

6.4 How to Read this Specification

6.4.1 Technical Clause Structure

The rest of this specification consists of technical clauses (Clause 7 to Clause 11). Each technical clause covers a particular profile package and is organized in the following way:

- Each clause starts with a sub clause to provide an overview of the UPR package.
- Then, each clause continues with a sub clause providing a domain view in which the domain model (where relevant) is provided and explained to illustrate the modeling concept. It is recommended to the reader of this profile to refer to Clause 6.2 for explanations on the theoretical aspects and the underlying mathematical construct of the modeling concept.
- In its last sub clause, each technical clause introduces UML stereotypes contained in the profile package. A profile diagram consisting of the stereotype is provided. Stereotype extensions always include the abstract syntax that identifies which metaclasses a stereotype extends. Each stereotype is defined with a general description, and the semantics are explained in terms of extensions, generalizations, association ends, attributes, operations and constraints. Each constraint consists of a textual description and is followed by a formal constraint expressed in OCL. If there is any ambiguity between the two, the OCL statement of the constraint takes precedence. If relevant, the sub clause then provides simple examples to show how defined stereotypes can be applied to annotate UML and/or SysML models.

6.4.2 Annexes

The Annexes of this profile do not introduce new stereotypes or metaclasses. These annexes provide examples[3] to show applications of UPR to practical engineering problems. In Annex A, a straightforward application of UPR stereotypes in the design of an elevator dispatcher is provided to demonstrate how UPR can be used to model a CDD problem in UML. Annex B demonstrates how multiple CDD problems (defined in different domains based on engineering practices) can be modeled based on a high-level vehicle architecture modeled in SysML. This annex also illustrates how UPR stereotypes facilitate design integration and tradeoff analysis.

6.4.3 Conventions and Typography

In this specification, the following conventions have been adopted:

- While referring to stereotypes, metaclasses, metaassociations, metaattributes, etc. in the text, the exact names as they appear in the model are always used.
- If a sub clause is not applicable, it is not included.
- Stereotype, metaclass and meta-association names: initial embedded capitals are used (e.g., “ModelElement,” “ElementReference”.
- Enumeration types: always end with “Type” (e.g., “OperatorType”).

6.4.4 Acknowledgments

The following companies/organizations submitted and/or supported parts of this specification:

- ZTI Systems Ltd
- Loughborough University
- Airbus
- National Institute of Standards and Technology (NIST)
- IBM

The following persons were members of the core team that originally designed and wrote this specification (sorted in alphabetical order): Yves Bernard, Charles Dickerson, Zahir Ismail, Siyuan Ji, Mole Li, and David Mulvaney.

In addition, the following persons contributed valuable ideas and feedback that significantly improved the content and the quality of this specification (sorted in alphabetical order): Graham Bleakley, Zhenyu Chen, and Peter Denno.

This work was partially sponsored by the Programme for Simulation Innovation (PSi), a partnership between Jaguar Land Rover and UK EPSRC grant EP/K014226/1.

[3] Note that real world engineering correctness of the examples provided is not of relevance for the illustration of the modeling capabilities provided by this profile.

7 UPR Foundations

7.1 Overview

This Clause aims to establish the foundation for UPR to facilitate the modeling of CDD problems based on discussion provided in Clause 6.2. To achieve comprehensive and sophisticated modeling of a CDD problem, UPR introduces two abstract stereotypes. Constraints, which are often defined mathematically, are the central elements in the definition of a CDD problem. UPR extends UML Constraints to facilitate the modeling of mathematical-based constraints. Then, in order to determine how designs are affected by the defined constraints, UPR extends UML Dependencies to facilitate the modeling of mathematical-based relations between entities that can be system elements and variables.

7.2 Domain View

The domain model for the Foundation package is depicted in Figure 7-1.

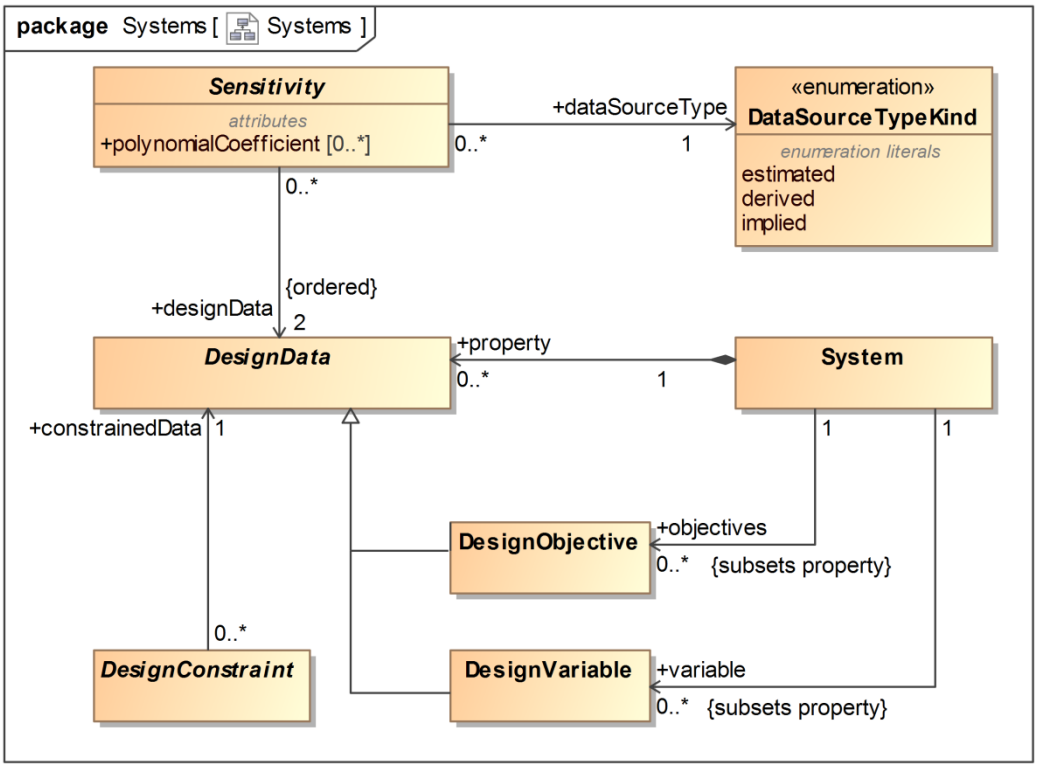

Figure 7-1 Systems (UPR Foundations) Domain Model

There are various definitions of the word: system. For instance, based on ISO/IEC/IEEE 15288 [REF14], a *System* is defined as “a combination of interacting elements organized to achieve one or more stated purposes”. Despite whichever definition is used, a System has properties, which are observables that can be described by some metrics. For example, a property of a vehicle can be fuel consumption; and a specific system (vehicle) can have a very low value on this property. In this example, “very low” is the metric used in the process of describing the fuel consumption. Similarly, in some situations, quantifiable metrics can be used to specify the properties, e.g. the average fuel consumption of the vehicle is 80mpg (miles per gallon). *DesignData* is an abstract class introduced to refer to the properties of a System. In particular, it can be specialized into two subsets: *DesignObjective* and *DesignVariable*. The definition of Design Objective and Design Variable are provided in Clause 4. In simple terms, in a design process, DesignObjectives refer to the properties that the System aims to achieve while DesignVariables refer to what the designer can change to achieve the DesignObjectives.

DesignData can be mathematically constrained by *DesignConstraints*. For example, in the design of a vehicle, customer demand on vehicle 0-60mph acceleration time to be less than 5 seconds can be considered as a constrained DesignObjective. Mathematically, this constraint can be expressed by using inequality as $\boldsymbol{acc} <$

$\mathbf{5s}$, where $\boldsymbol{acc}$ is a variable used to denote 0-60mph time and s denotes second. A constraint on DesignObjective may impose limitations in the design specification of the system. Following the above example on vehicle design, a constrained DesignObjective on the acceleration time may then restrict possible design in the mass of the vehicle and the engine power to ensure the desired acceleration time is met. As such, we conclude that two DesignVariables, vehicle mass and engine power, are identified for the designer to decide in order to achieve the constrained DesignObjective.

An instance of DesignData can be mathematically-dependent on other instance or instances of DesignData. This mathematical dependency is captured by an abstract concept, *Sensitivity* (c.f. Clause 6.2.3). This captures how sensitive a DesignData is to another. Sensitivity provides an intuitive indication on direction of improvement or otherwise. Again, with the example on vehicle design, we know that 0-60mph time is sensitive to the mass of the vehicle: the heavier the vehicle, the longer it will take the vehicle to reach 60mph from stationary. Then, this positive correlation can be modeled by a positive, single valued *polynomialCoefficient* in Sensitivity, e.g. 5. Similarly for engine power, the higher the engine power, the rate of acceleration of the vehicle is increased, hence reducing the 0-60mph time. This negative correlation can be modeled by a negative, single valued Sensitivity, e.g. -3, on a same scale. Instead of using a subjective metric to specify the Sensitivity, the modeler may also want to use historical data or simulation models to derive a more accurate measure of the Sensitivity. In this case, using regression techniques, to the second order, the 0-60mph time-to-vehicle mass Sensitivity may be derived as a polynomial that reads $\boldsymbol{acc = 0.3 + 6.85m + 0.1m^2}$, where $\boldsymbol{m}$ is the vehicle mass in ton[4]. The three constants in this polynomial are then modeled using polynomialCoefficients of Sensitivity. Depending on the source of the modeled polynomialCoefficients, different sensitivity analysis techniques can be used for the purpose of design. The source of the polynomialCoefficients is defined through the enumeration, *DataSourceTypeKind*. Its specific types of data sources are explained in the following:

When a DataSourceTypeKind is defined as *estimated*, it indicates that the value(s) defined for the Sensitivity is based on subjective quantification. These values, in practice, are often obtained from empirical evidence and tacit knowledge of the system designers. Although, in principle, there is no restriction on the order of the polynomial that one can *estimate* up to, given the uncertainty nature of estimation, it is more reasonable to use a single value only to model the Sensitivity as a first order polynomial coefficient.

When a DataSourceTypeKind is defined as *derived*, it indicates that the value(s) defined for the Sensitivity is based on applying techniques such as polynomial regression on a data set. The coefficient(s) derived from data sets provide a higher fidelity in comparison to estimated value(s) in understanding how sensitive one entity is to another.

When a DataSourceTypeKind is defined as *implied*, it indicates that the value defined for the Sensitivity is not specified by the modeler, but is obtained from an implementation of the UT-B as defined in Clause 4 and explained in detail in Clause 9. The implied Sensitivity, which is a single value, is obtained from applying the UT-B on Sensitivities (that are estimated or derived into first order polynomial) defined by the modeler.

7.3 UML Representation

This clause describes the UML extensions required to support the concept of constraint modeling and sensitivity modeling.

7.3.1 Profile Diagrams

Figure 7-2 depicts the UML extensions for modeling UPRConstraints and Sensitivities. The descriptions for the introduced stereotypes and their properties are provided in the following sub clause.

[4] Note that this polynomial model is for illustration only.

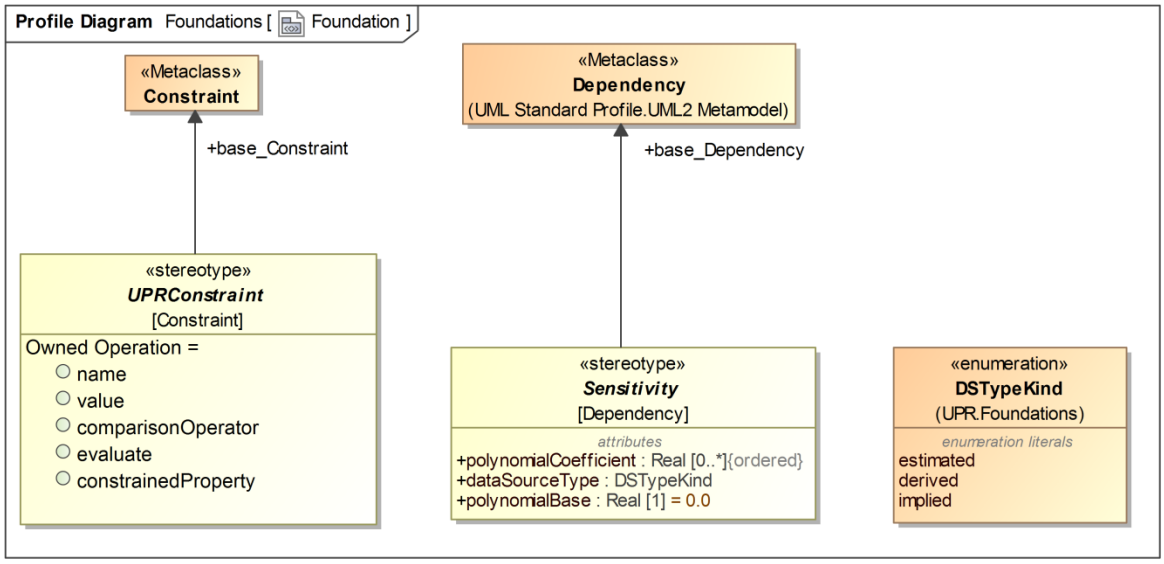

Figure 7-2 Foundations Profile Diagram[5]

7.3.2 Profile Elements Description

7.3.2.1 DSTypeKind (Enumeration)

Literals

- estimated
  Specifies that the attribute owned by the sensitivity stereotype is an estimated value based on subjective decision.
- derived
  Specifies that the set of values in Sensitivity::polynomialCoefficient are derived from data.
- implied
  Specifies that the attribute owned by the sensitivity stereotype is an implied value determined from the Binary Transformation (RT-B).

7.3.2.2 UPRConstraint (Abstract Stereotype)

An UPR constraint stereotype extends the UML Constraint metaclass which is used for specifying Design Objective Constraints and Design Variable Constraints. The specification of the UPR constraint is in the form of a comparison operator and a real number. Note that the UPR library provides a set of predefined comparison operators.

Extends

- Constraint (from UML)

Association Ends

- base_Constraint : Constraint [1]

Operations

- comparisonOperator () : invalid [1]
  `bodyCondition:`

[5] Note that (i) package membership of a class from an external package is shown in brackets underneath the name of the class; and (ii) Operations defined for the stereotypes are depicted in the "Owned Operation" rather than the conventional Class Operation due to limitation of the specific modeling tool used.
The above two conventions are also applied to profile diagrams in the rest of this profile.

```
UPR::Operators::OperatorSemantics.allInstances()->any(o |
o.notation.symbol()-
>includes(self.base_Constraint.specification.stringValue()))
```

- constrainedProperty () : Property [1]

```
bodyCondition:
self.base_Constraint.constrainedElement-
>at(1).oclAsType(UML::Property)
```

- evaluate () : Boolean [1]

```
bodyCondition:
self.comparisonOperator.applyTo(self.base_Constraint)
```

- name () : String [1]

```
bodyCondition:
self.base_Constraint.name
```

- value () : ValueSpecification [1]

```
bodyCondition:
self.base_Constraint.constrainedElement-
>at(2).oclAsType(UML::ValueSpecification)
```

Constraints

- constrainedElement1_ property
  The first constrained element shall be a property

```
self.base_Constraint.constrainedElement->at(1).oclIsKindOf(Property)
```

- constrainedElement2_ value_specification
  The first constrained element shall be a value specification

```
self.base_Constraint.constrainedElement-
>at(2).oclIsKindOf(ValueSpecification)
```

- two_constrained_elements
  An UPRConstraint constraint shall have 2 constrained elements

```
self.base_Constraint.constrainedElement->size()=2
```

7.3.2.3 Sensitivity (Abstract Stereotype)

The Sensitivity abstract stereotype extends the UML Dependency to model the Sensitivity between two NamedElements.

Extends

- Dependency (from UML)

Attributes

- dataSourceType : DSTypeKind [1]
  Represent the data source (DSTypeKind enumeration) used to specify Sensitivity
- polynomialBase : Real [1]
  Base of the polynomial, i.e. the coefficient of the polynomial term with degree 0. If not specified 0 is used as the default value.
- polynomialCoefficient : Real [0..*]
  Provide the list of coefficients to be used for the univariate polynomial approximation of the mathematical relation between two variables. First element maps to the coefficient of the term of degree 1, the second to that of the term of degree 2, and so on. The base of the polynomial is supported by the specific attribute: 'polynomialBase'.

Association Ends

- base_Dependency : Dependency [1]

Constraints

- is_binary
  A Sensitivity is a binary relationship

```
self.base_Dependency.supplier->size() = 1 and
self.base_Dependency.client->size() = 1
```

This page intentionally left blank.

8 Operators

8.1 Overview

A mathematical-based constraint may require the use of different comparison operators. To facilitate the use of various kinds of comparison operators to model mathematical-based constraints, this clause establishes an extendable library of comparison operators. Each defined operator has a unique text-based name and a notation. Implementers and users of UPR can refine and extend the library as needed.

8.2 UML Representation

This clause describes the UML extensions required to support the construction of the Operator Library.

8.2.1 Profile Diagrams

Figure 8-1 depicts the UML extensions contained in Operators package and Figure 8-2 depicts the currently defined Operators in the Library. The descriptions corresponding to the introduced stereotypes and their properties are provided in the following sub clause.

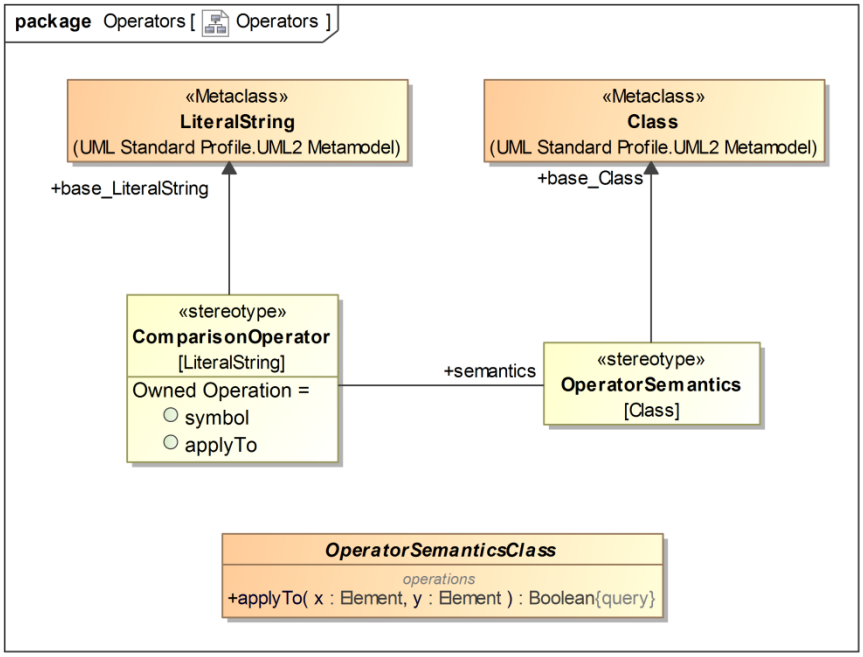

Figure 8-1 Operators

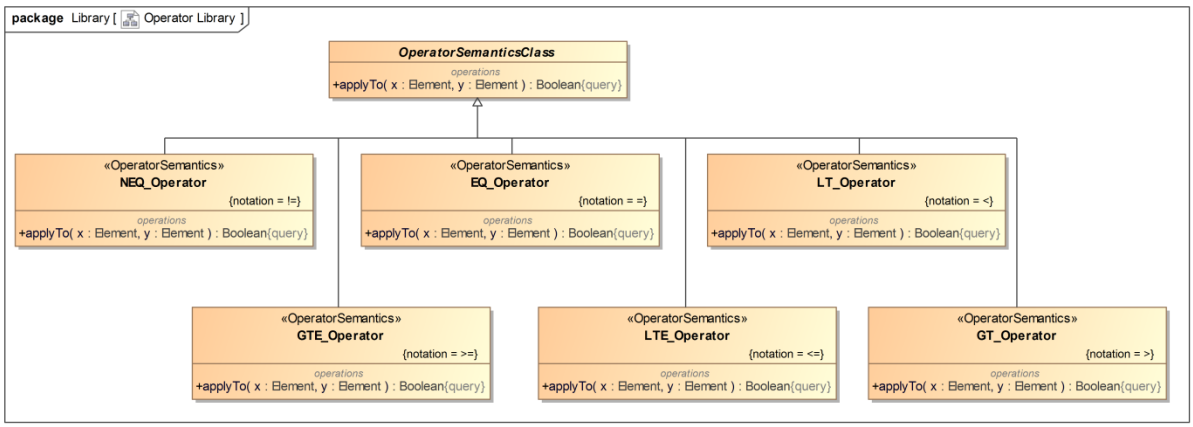

Figure 8-2 Operator Library

8.2.2 Profile Elements Description

8.2.2.1 ComparisonOperator (Stereotype)

A ComparisonOperator stereotype allows defining a logical operator that can be used with UPRConstraints in order to quantify a relationship between a variable and a specific value. The Literal String that this stereotype extends defines the symbol to use in order to refer to that operator. The corresponding semantics is specified by the semantics attributes of the stereotype.

Extends

- LiteralString (from UML)

Association Ends

- base_LiteralString : LiteralString [1]
- semantics : OperatorSemantics [1]

Operations

- applyTo (in c : UPRConstraint) : Boolean [1]
  This operation evaluates an UPRConstraint base on the semantics specified for that operator

```
bodyCondition:
self.applyTo(c.base_Constraint.constrainedElement->at(1),
c.base_Constraint.constrainedElement->at(2))
```

- symbol () : String [1]

```
bodyCondition:
self.base_LiteralString.value
```

8.2.2.2 OperatorSemantics (Stereotype)

An OperatorSemantics stereotype is applied on a Class that can be used for specifying the semantics of a ComparisonOperator that a UPRConstraints can refer to. Typically the Base Class is a specialization of the OperatorSemanticsClass

Extends

- Class (from UML)

Association Ends

- base_Class : Class [1]
- notation : ComparisonOperator [0..*]

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
self.base_Class.oclAsType(OperatorSemanticsClass).applyTo(x, y)
```

Constraints

- has_applyto_operation

```
self.base_Class.oclIsKindOf(OperatorSemanticsClass)
```

8.2.2.3 OperatorSemanticsClass (Abstract Class)

Provides an abstract base class for defining the semantics of a ComparisonOperator

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]
  This operation is used for specifying how the result of the comparison shall be computed

8.2.2.4 Library (Package)

8.2.2.4.1 EQ_Operator (Class)

This class defines the semantics for an “equal to” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) = y.oclAsType(Real)
else
 OclInvalid
endif
```

8.2.2.4.1 GTE_Operator (Class)

This class defines the semantics for a “greater than or equal to” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) >= y.oclAsType(Real)
else
 OclInvalid
endif
```

8.2.2.4.1 GT_Operator (Class)

This class defines the semantics for a “greater than” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) > y.oclAsType(Real)
else
 OclInvalid
endif
```

8.2.2.4.1 LTE_Operator (Class)

This class defines the semantics for a “less than or equal to” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) <= y.oclAsType(Real)
else
 OclInvalid
endif
```

8.2.2.4.1 LT_Operator (Class)

This class defines the semantics for a “less than” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) < y.oclAsType(Real)
else
 OclInvalid
endif
```

8.2.2.4.1 NEQ_Operator (Class)

This class defines the semantics for a “not equal to” relationship

Generalizations

- OperatorSemanticsClass (from Operators)

Operations

- applyTo (in x : Element, in y : Element) : Boolean [1]

```
bodyCondition:
if (x.oclIsKindOf(Real) and y.oclIsKindOf(Real)) then
 x.oclAsType(Real) <> y.oclAsType(Real)
else
 OclInvalid
endif
```

9 Design Objective Constraints Modeling

9.1 Overview

This clause describes the Design Objective Constraints package of UPR. This package specializes the stereotypes defined in the Foundations package to facilitate:

- the modeling of mathematical-based Design Objective Constraints;
- the modeling of mathematical-based harmonious and conflicting relations between system elements and variables, e.g. Design Objectives;
- and the identification of mathematical-based harmonious and conflicting relations between Design Objectives.

9.2 Domain View

The domain model for the Design Variable package is depicted in Figure 9-1.

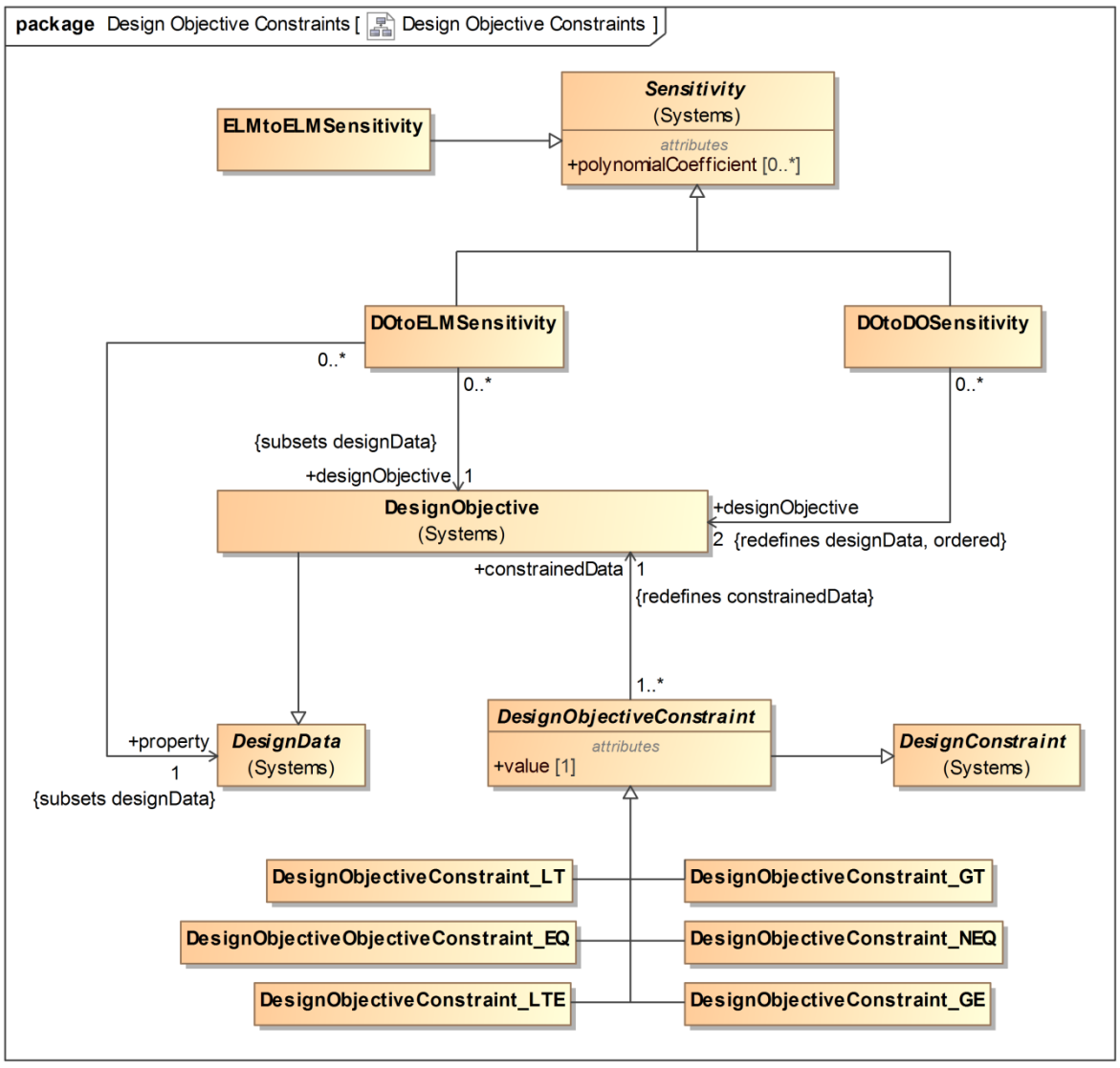

Figure 9-1 Design Objective Constraints Domain Model

The Design Objective Constraints package is concerned with the modeling of constraints on Design Objectives and relations between System::DesignData such as a harmonious relation between two System::DesignObjectives.

A *DesignObjectiveConstraint*, being a specialized type of System::DesignConstraint, defines a mathematical constraint on a System::DesignObjective by using a comparison operator (listed as below) and a *value* (c.f. Clause 6.2.2 for the defined structure of mathematical constraints on variables within the scope of UPR). There can be multiple DesignObjectiveConstraints defined on the same DesignObjective. Based on the comparison operator used, the DesignObjectiveConstraint can be classified into the following forms including but not limited to:

- *DesignObjectiveConstraint_EQ*: an equal constraint, e.g. gear modes = 5;
- *DesignObjectiveConstraint_NEQ*: a not-equal constrain[6];
- *DesignObjectiveConstraint_GT*: a greater than constraint, e.g. vehicle frontal area (m^2) ≥ 2.2
- *DesignObjectiveConstraint_GE*: a greater than or equal to constraint, e.g. vehicle top speed (mph) ≥ 200
- *DesignObjectiveConstraint_LT* : a less than constraint, e.g. CO emission (g/km) < 0.5
- *DesignObjectiveConstraint_LE*: a less than or equal to constraint, e.g. vehicle weight (kg) ≤ 2000

Sensitivities between two System::DesignObjectives are specialized as *DOtoDOSensitivities*. Sensitivities between a System::DesignObjective and a DesignData (a system property that is neither defined as a Design Objective nor as a Design Variable, c.f. Clause 4) are defined as a *DOtoELMSensitivities*. Sensitivities between two DesignData (system properties that both are neither defined as Design Objectives nor as Design Variables) are defined as *ELMtoELMSensitivities*. These specialized Sensitivities are essential for carrying out a RT-B for the identification of harmonious, conflicting or independent relations between two Design Objectives (c.f. Clause 6.2.4).

9.3 UML Representation

This clause describes the UML extensions required to support the modeling of Design Objective Constraints.

9.3.1 Profile Diagrams

Figure 9-2 depicts the UML extensions contained in the Design Objective Constraints package for modeling Design Objective Constraints and their relations. The semantic descriptions corresponding to the introduced stereotypes and their properties are provided in the next sub clause.

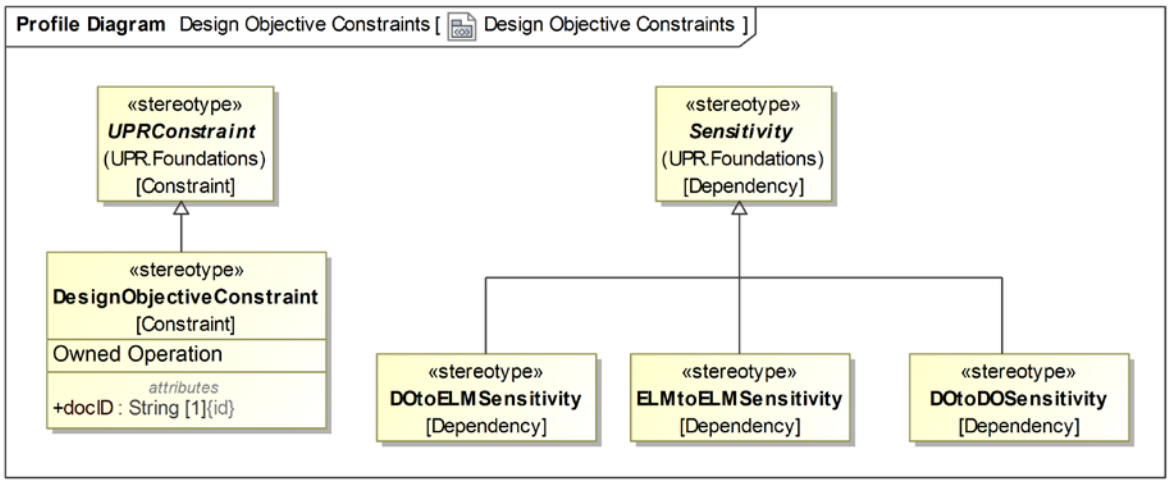

Figure 9-2 Design Objective Constraints Profile Diagram

[6] In practical engineering, a not-equal constraint is rarely used. However, for the sake of completeness, this type of DesignObjectiveConstraint is still included.

9.3.2 Profile Elements Description

9.3.2.1 DOtoDOSensitivity (Stereotype)

The DOtoDOSensitivity stereotype specializes the Sensitivity stereotype to model a sensitivity-based relation between two Design Objectives that are constrained.

Extends

- Dependency (from UML)

Generalizations

- Sensitivity (from Foundations)

Constraints

- client_and_supplier_designobjectives
  Both the client and the supplier shall be constrained by a DesignObjectiveConstraint

```
self.base_Dependency.client->union(self.base_Dependency.supplier)->forAll(p | DesignObjectiveConstraint.allInstances().constrainedProperty()->includes(p))
```

9.3.2.2 DOtoELMSensitivity (Stereotype)

The DOtoELMSensitivity stereotype specializes the Sensitivity stereotype to model a sensitivity-based relation between a Design Objective that is constrained and a NamedElement.

Extends

- Dependency (from UML)

Generalizations

- Sensitivity (from Foundations)

Constraints

- client_designobjective
  The client shall be constrained by a DesignObjectiveConstraint

```
self.base_Dependency.client->forAll(p | DesignObjectiveConstraint.allInstances().constrainedProperty()->includes(p))
```

9.3.2.3 DesignObjectiveConstraint (Stereotype)

The DesignObjectiveConstraint stereotype specializes the UPRConstraint stereotype for specifying a Design Constraint.

Extends

- Constraint (from UML)

Generalizations

- UPRConstraint (from Foundations)

Attributes

- docID : String [1]
  This attribute represents the unique Identification Number of a Design Objective Constraint.

9.3.2.4 ELMtoELMSensitivity (Stereotype)

The ELMtoELMSensitivity stereotype extends the UML Dependency to model a sensitivity-based relation between two NamedElements.

Extends

- Dependency (from UML)

Generalizations

- Sensitivity (from Foundations)

9.3.3 Example

This sub-clause illustrates how Sensitivity stereotypes can be used to model mathematical-based relations between system elements. It also demonstrates how Sensitivity information can be used to identify harmonious and conflicting relations between Design Objectives that are constrained.

To achieve the above, the vehicle design problem discussed in Clause 6.2.1 is elaborated in this sub clause. In particular, we model three Design Objective Constraints:

1) Vehicle Fuel Economy (FE) to be not less than a value of 23 Mile per Gallon (MPG) in a standard drive cycle. Mathematically, we have the Design Objective, FE (MPG), constrained by a Design Objective Constraint, $FE \geq 23$.

2) Particulate Matter (PM) emission to be less than 0.005g/km as regulated by Euro 6 Emissions Standard. Mathematically, we have the Design Objective, PM (g/km), constrained by a Design Objective Constraint, $PM < 0.005$.

3) The Diesel Particulate Filter (DPF) to have a filtration efficiency of greater than 90%. Mathematically, we have the Design Objective, DPF-eff (%), constrained by a Design Objective Constraint, $DPF\text{-}eff > 90$.

For domain experts, particularly Chemical Engineers who work on vehicle after-treatment systems, it is known to them that:

1) Design Objectives 2 and 3 are harmonious to each other since a high efficiency of DPF helps to keep the emissions of Particulate Matter at a significantly low level;

2) Design Objective 1 is in general, conflicting with both of Design Objective 2 and Design Objective 3. This is because the way a DPF works is to filter and store the PM from the exhaust gas over time. When the stored PM reaches a limit, the engine will be requested to run at very high temperature such that the high temperature exhaust gas can burn the stored PM. When the engine operates at very high temperature, it has a greater fuel consumption compared to the normal operating condition. Hence, reducing PM in general means decreasing FE.

The above points are referred to as information 1 and 2 respectively later. However, to demonstrate the use of the Design Objective Constraints package, we assume that the user of UPR has no prior knowledge to the above relations between the Design Objectives. The system architecture that captures the relationships between system elements and Design Objectives are captured in Figure 9-3. The set of Design Objective Constraints are provided in the table depicted in Figure 9-4. They are traceable to requirements modeled in SysML Requirements[7] as shown in Figure 9-3 via satisfy Dependencies.

[7] Textual descriptions of Design Objectives are modeled in SysML Requirements. However, this is not mandatory. Also, UPR can be applied to SysML models even though there is no direct relation to SysML as in Conformance Clause.

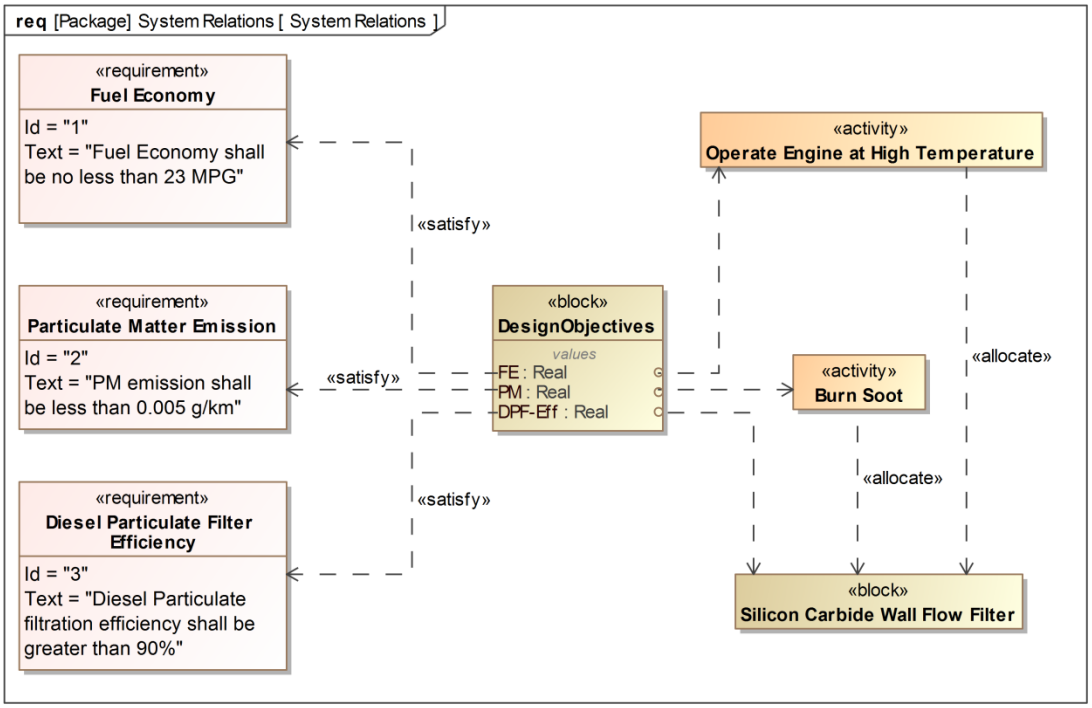

Figure 9-3 System Relations

#	Applied Stereotype	△ ○ docID	Name	Specification	Constrained Element
1	«» DesignObjectiveConstraint	1	{} FE	>=	FE : Real FE = 23.0
2	«» DesignObjectiveConstraint	2	{} PM	<	PM : Real PM = 0.005
3	«» DesignObjectiveConstraint	3	{} DPF-Eff	>	DPF-Eff : Real DPF-Eff = 90.0

Figure 9-4 Design Objective Constraints

After modeling the Design Objective Constraints, we continue to establish a set of stereotypes to capture Sensitivity information and to facilitate RT-B for identification of relations between Design Objectives. For simplicity, in the following discussion, we will use a Sensitivity scale (estimated) of -1, 0 and +1 to show if two model elements are conflicting, independent and harmonious. A complex scale can be used if the degree of conflict and harmony can be reasonably modeled. Using this scale, for the modeled relations shown in the System Relations (Figure 9-3), we have:

1) Design Objective, FE, relates to a system function modeled by an Activity as to “Operate Engine at High Temperature”. When the engine operates at a high temperature, FE decreases. Therefore, based on the DesignObjectiveConstraint 1 in which it is required to increase FE to meet the target, the relation between FE and the “Operate Engine at High Temperature” Activity is stereotyped into a DOtoELMSensitivity with an estimated single valued Sensitivity, -1, to show that they are conflicting with each other.

2) Design Objective, PM, relates to a system function modeled by an Activity as to “Burn PM”. This function is one of the most viable methods to reduce PM. The function converts PM chemically through burning. Hence, from a verification perspective, this Activity ensures that DesignObjectiveConstraint can be satisfied. For this reason, we stereotype the relation between PM and the Activity, “Burn PM” into a DOtoELMSensitivity with an estimated positive single valued Sensitivity, 1.

3) Design Objective, DPF-Eff, relates to a system component modeled by a Block, named “Silicon Carbide Wall Flow Filter”, which is a type of DPF that can achieve filtration efficiency over 90%. For the same reason as discussed in 2) above, we stereotype the relation between DEP-Eff and the Block, “Silicon Carbide Wall Flow Filters” into a DOtoELMSensitivity with an estimated positive single valued Sensitivity, 1.

It is worth noting that the three relations modeled above correspond to the three relations captured in the matrix Q in Figure 6-1

Now, looking at these system elements, when modeling the allocation of system functions to system components, it can be observed that both of the Activities are allocated to the same Block, “Silicon Carbide Wall Flow Filter”. We therefore stereotype these allocation Dependencies into ELMtoELMSensitivities with an estimated positive single valued Sensitivity, 1, also to show the harmoniousness. It is worth noting that the two allocation Dependencies correspond to the two relations captured in the matrix N in Figure 6-1.

The above modeled Sensitivities are summarized in the table depicted in Figure 9-5.

#	Applied Stereotype	Source	Target	polynomialCoefficient	dataSourceType	polynomialBase
1	«» Allocate [Abstractior «» ELMtoELMSensitivity	Burn S...	Silicon C...	1.0	estimated	0.0
2	«» Allocate [Abstractior «» ELMtoELMSensitivity	Operat...	Silicon C...	1.0	estimated	0.0
3	«» DOtoELMSensitivity	DPF-Eff...	Silicon C...	1.0	estimated	0.0
4	«» DOtoELMSensitivity	FE : Real	Operate...	-1.0	estimated	0.0
5	«» DOtoELMSensitivity	PM : Real	Burn S...	1.0	estimated	0.0

Figure 9-5 Sensitivities Modeling (estimated)

With the modeled DesignObjectiveConstraints, DOtoELMSensitivities, and ELMtoELMSensitivities, we now apply RT-B based on Equation 6-6 in Clause 6.2.4 to identify implied DOtoDOSensitivities and their associated Sensitivity values. It is important to emphasize here again that UML and SysML, as modeling languages, are not platforms that offer the implementation of analysis capabilities such as relational transformations, e.g. RT-B. Therefore, RT-B, as an analysis capability, has to be implemented in a third party tool. This, however, is not a conformance point for UPR. In this case, this tool shall read the sensitivities and implemented RT-B in a way as discussed in Clause 6.2.4. Then, the resultant sensitivity after the relational transformation is annotated on the relevant parts (DOtoDOSensitivities as in this example) of the model.

The resultant (implied) DOtoDOSensitivities obtained from the transformation are depicted in the table depicted in Figure 9-6. Firstly, PM and DPF-Eff are identified to be positively correlated due to a multiplication of three +1s. The outcome result agrees with the previously discussed information item (1). Secondly, FE and DPF-Eff are identified to be negatively correlated due to a multiplication of two +1s and a -1. The outcome result agrees with the previously discussed information item (2). As these DOtoDOSensitivities are obtained through RT-B rather than original modeled by the user of UPR, the DataSourceType should be ‘implied’. It is worth noting that the two implied relations correspond to the two relations captured in the matrix M in Figure 6-1.

In addition to the two ‘implied’ Sensitivities, it is also possible to specify the DOtoDOSensitivity between FE and PM by using a set of polynomial coefficients ‘derived’ from the stoichiometry during engine combustion. Here, a 3rd order polynomial representing the relation of FE to PM is provided (see row 3 in the table in Figure 9-6); and note that the values provided here are for demonstration only.

#	Applied Stereotype	Source	Target	polynomialCoefficient	dataSourceType	polynomialBase
1	«» DOtoDOSensitivity	FE : Real	DPF-Eff...	-1.0	implied	0.0
2	«» DOtoDOSensitivity	PM : Real	DPF-Eff...	1.0	implied	0.0
3	«» DOtoDOSensitivity	FE : Real	PM : Real	2.54 -2.29 0.51	derived	5.9

Figure 9-6 Sensitivities Modeling (implied and derived)

10 Design Variable Constraints Modeling

10.1 Overview

This clause describes the Design Variable Constraints package of UPR. This package specializes the UPRConstraint stereotype defined in the Foundations package to facilitate the modeling of Design Variable Constraints.

10.2 Domain View

The domain model for the Design Variable Constraints package is depicted in Figure 10-1.

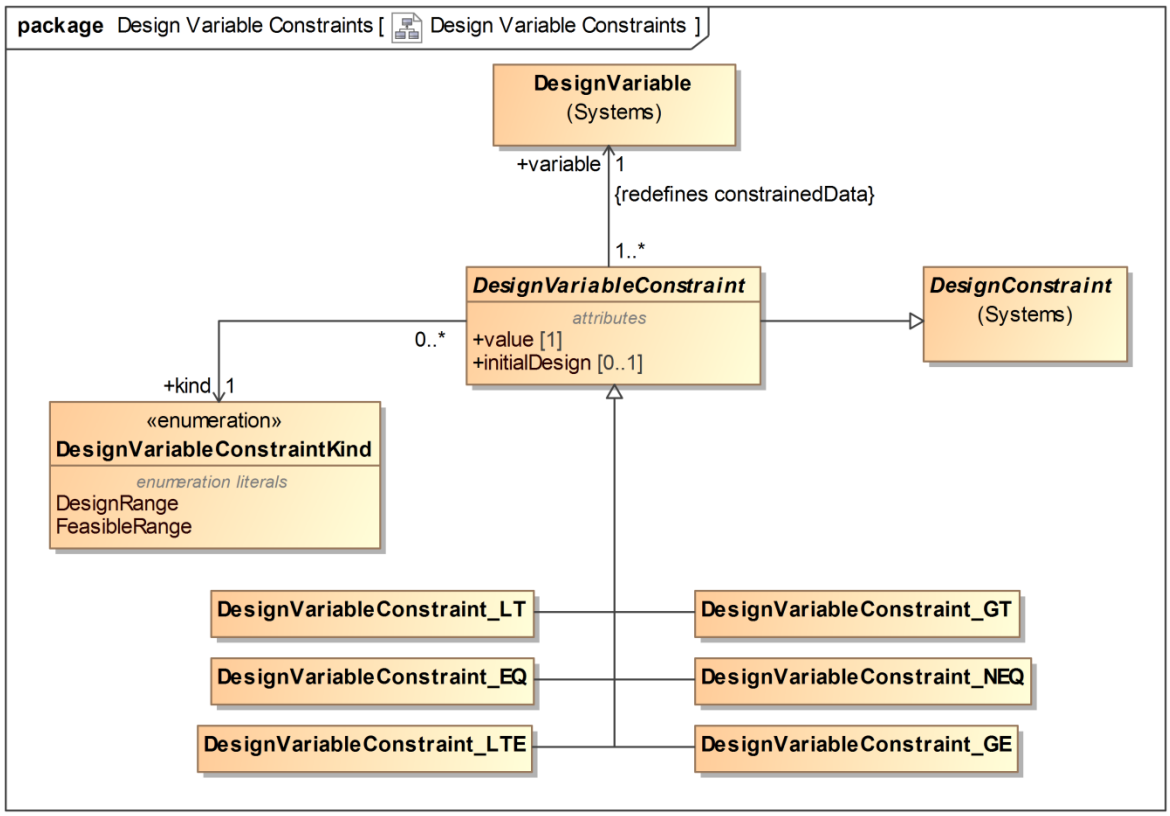

Figure 10-1 Design Variable Constraints Domain Model

The Design Objective Constraints package is concerned with the modeling of constraints on Design Variables.

A *DesignObjectiveConstraint*, being a specialized type of System::DesignConstraint, defines a mathematical constraint on a System::DesignVariable by using a comparison operator (listed as below) and a *value*. There can be multiple DesignVariableConstraints defined on the same DesignVariable. Based on the comparison operator used, the DesignObjectiveConstraint can be classified into the following forms including but not limited to:

- *DesignVariableConstraint _EQ*
- *DesignVariableConstraint_NEQ*
- *DesignVariableConstraint_GT*
- *DesignVariableConstraint_GE*
- *DesignVariableConstraint_LT*
- *DesignVariableConstraint_LE*

A *DesignVariableConstraint* has two different kinds in nature: *DesignRange* and *FeasibleRange*. When a DesignVariableConstraint describes a *DesignRange*, it means that the constraint specifies a design limit on the corresponding Design Variable without the consideration of any DesignObjectiveConstraints. This is often derived from a physical limit or a limit that is inherited from legacy designs. When DesignObjectiveConstraints

are considered, certain design choices within the specified DesignRange (design limit) may no longer be viablem, due to violation of the defined DesignObjectiveConstraints. Using UPR terminology (c.f. Clause 4 and Clause 6.2.5), the initial design limit on a Design Variable is referred to as Design Range; and a refined Design Range representing a subset of the feasible design choices is referred to as a Feasible Range. A Feasible Range is modeled by a DesignVariableConstraint with its kind specified as *FeasibleRange*.

The current design choice on the Design Variable is referred to as an InitialDesign in the Domain Model.

10.3 UML Representation

This clause describes the UML extensions required to support the modeling of Design Variable Constraints.

10.3.1 Profile Diagrams

Figure 10-2 depicts the UML extensions for modeling Design Variables. The semantic descriptions corresponding to the introduced stereotypes and their properties are provided in the following sub clause.

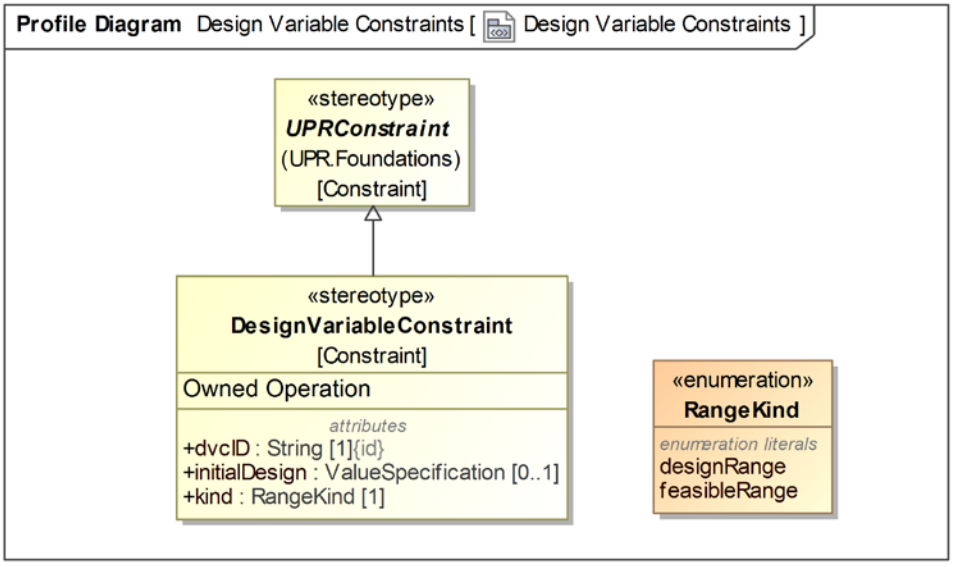

Figure 10-2 Design Variable Constraints Profile Diagram

10.3.2 Profile Elements Description

10.3.2.1 RangeKind (Enumeration)

A RangeKind describes the nature of the specified mathematical constraint on a Design Variable. Note that there can be multiple DesignVariableConstraints specified for one Design Variable, but each DesignVariableConstraint stereotype is only associated to one of the RangeKind as specified below:

Literals

- designRange
  Indicates that the constraint specified for the Design Variable is a Design Range that is attributed to physical limit or design knowledge. Not every design point within this range is necessarily a feasible solution.
- feasibleRange
  Indicates that the constraint specified for the Design Variable is a Feasible Range that is imposed by a set of Design Objective Constraints. Every design point within this range is a feasible solution.

10.3.2.2 DesignVariableConstraint (Stereotype)

The DesignVariableConstraint stereotype specializes the UPRConstraint stereotype for specifying Design Range(s) and Feasible Range(s) for a Design Variable. In addition, the initial value to be used for the design optimization is provided by the initialDesign property.

Extends

- Constraint (from UML)

Generalizations

- UPRConstraint (from Foundations)

Attributes

- dvcID : String [1]
  This attribute represents the unique Identification Number of a Design Variable Constraint.
- initialDesign : ValueSpecification [0..1]
  This attribute represents the initial design choice for the Design Variable.
- kind : RangeKind [1]
  This attribute represents the kind of range, i.e. designRange or feasibleRange, being specified for the Design Variable Constraint.

10.3.3 Example

Examples for Design Variable Constraints definition are given in Clause 10.3.3 in conjunction with the application of stereotypes defined in the Relational Structure and Design package in Clause 10. This is because that complete usage of DesignVariableConstraint stereotype requires the definition of Feasible Ranges on a Design Variable, which are constraints that are imposed by Design Objective Constraints through the mathematical relations, e.g. polynomials, between the Design Objective and the Design Variable.

This page intentionally left blank.

11 Relational Structure and Design Modeling

11.1 Overview

Having both the Design Objective Constraint and Design Variable Constraints defined, to complete the formulation of a CDD problem, it is required to define the mathematical relations between the specified Design Objectives and Design Variables. This clause is concerned with the modeling of these mathematical relations in the form of polynomial coefficients (c.f. Clause 6.2.3). The collective representation of these mathematical relations creates a Relational Structure. This structure facilitates Unary Transformation (RT-U) to determine feasible design solutions for a CDD problem.

The objective of this clause is to describe the Relational Structure and Design package of UPR. This package specializes the Sensitivity abstract stereotype defined in the Foundations package to facilitate:

- the modeling of the relational structure between Design Objective Constraints and Design Variable Constraints;
- and the determination of Feasible Ranges for the Design Variables with a given set of Design Objective Constraints, by using Unary Transformation (RT-U).

11.2 Domain View

The domain model for the Relational Structure and Design is depicted in Figure 11-1.

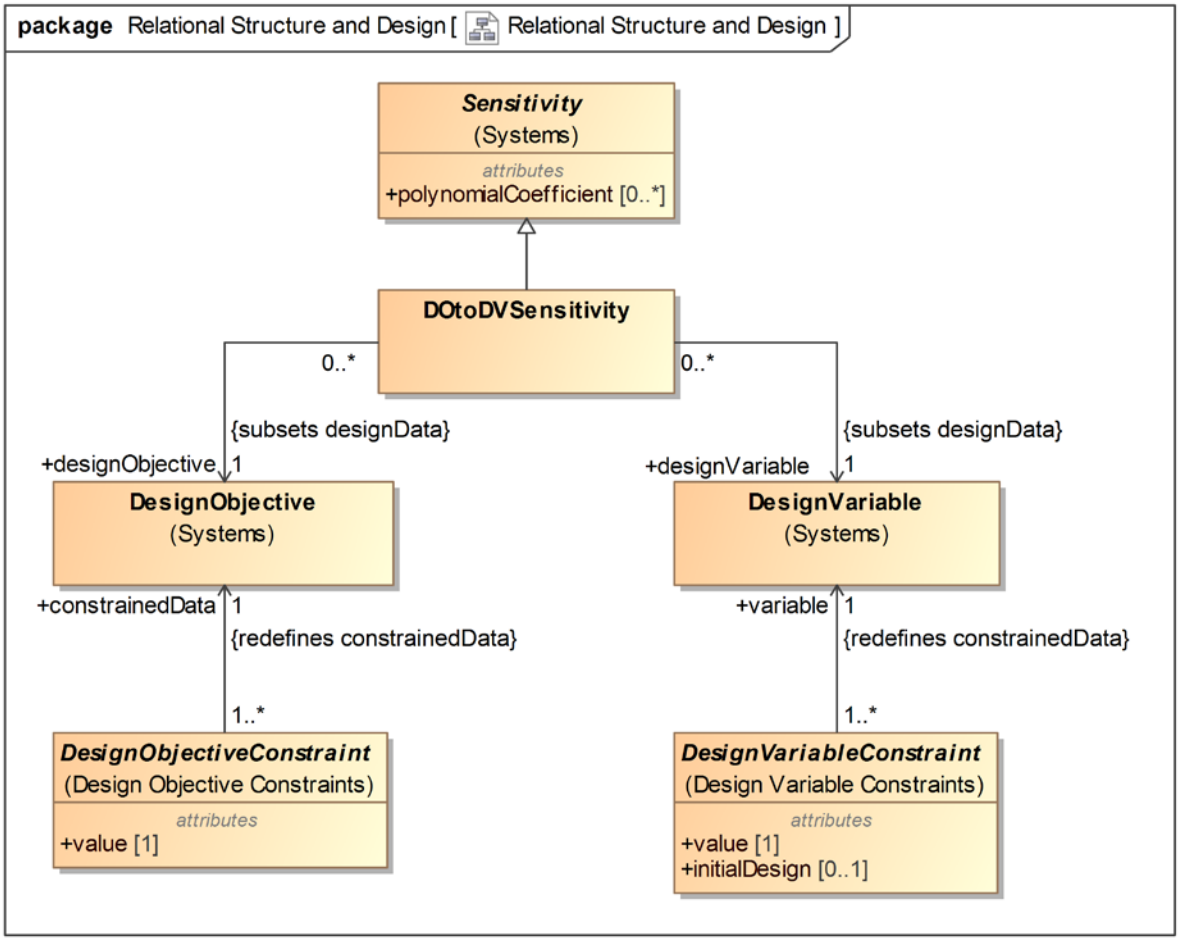

Figure 11-1 Relational Structure and Design Domain Model

Having introduced both DesignObjectiveConstraint and DesignVariableConstraint in the domain models in previous clauses, the Relational Structure and Design package concerns with the modeling of Sensitivities between a Design Objective and Design Variable by using a *DOtoDVSensitivity* that specializes

System::Sensitivity. As seen in the domain model, a System::DesignObjective and a System::DesignVariable may be related to each other via a DOtoDVSensitivity, where the DesignObjective is constrained by one or more DesignObjectiveConstraints and the DesignVariable is constrained by one or more DesignVariableConstraints.

11.3 UML Representation

This clause describes the UML extensions required to support the modeling of Relational Structure and Design.

11.3.1 Profile Diagrams

Figure 9.1 depicts the UML extensions for modeling the Relational Structure and Design. The semantic descriptions corresponding to the introduced stereotypes and their properties are provided in the next sub clause.

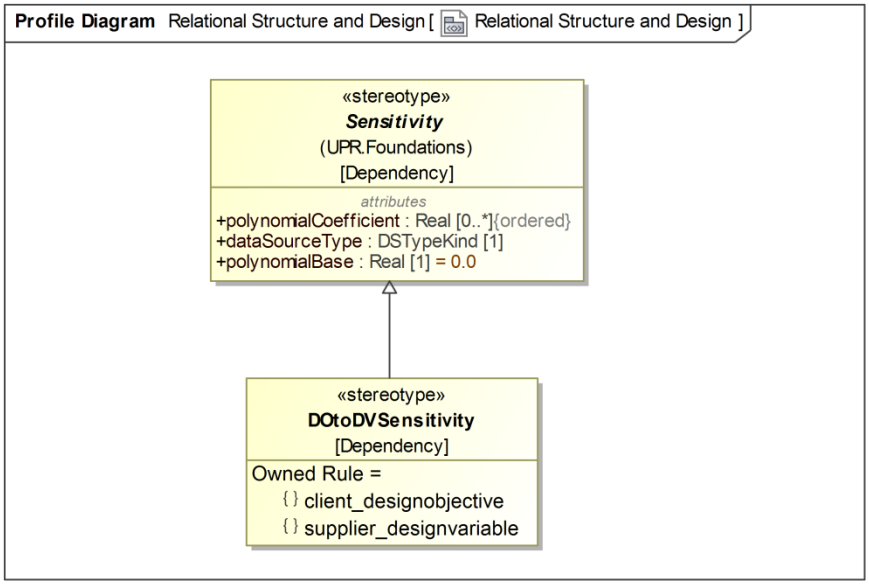

Figure 11-2 Relational Structure and Design Profile Diagram

11.3.2 Profile Elements Description

11.3.2.1 DOtoDVSensitivity (Stereotype)

The DOtoDVSensitivity stereotype specializes the Sensitivity stereotype to model a sensitivity-based relation between a Design Objective that is constrained by DesignObjectiveConstraints and a Design Variable that is constrained by DesignVariableConstraints.

Extends

- Dependency (from UML)

Generalizations

- Sensitivity (from Foundations)

Constraints

- client_designobjective
  The client shall be constrained by a DesignObjectiveConstraint

```
self.base_Dependency.client->forAll(p |
DesignObjectiveConstraint.allInstances().constrainedProperty()-
>includes(p))
```

- supplier_designvariable
  shall be constrained by a DesignVariableConstraint

```
self.base_Dependency.supplier->forAll(p |
```

```
DesignVariableConstraint.allInstances().constrainedProperty()-
>includes(p))
```

11.3.3 Example

In this sub clause, we will be using a simple constraint-driven Radar design problem as an example to demonstrate the application of the Relational Structure and Design package. As the DOtoDVSensitivity stereotype connects a DesignObjectiveConstraint and a DesignVariableConstraint, we start with modeling the Design Objective Constraints and Design Variable Constraints of the system based on a simplified Radar system requirements and architecture modeled in SysML. The Design Objectives of interest are depicted in the SysML Requirement diagram in Figure 11-3. These are the power received at the aircraft transponder, which must exceed the Minimum Triggering Level (MTL) and the power received at the radar antenna, which must exceed the Minimum Detectable Signal (MDS). Note the units used to measure these variables are rescaled into decibel (dB).

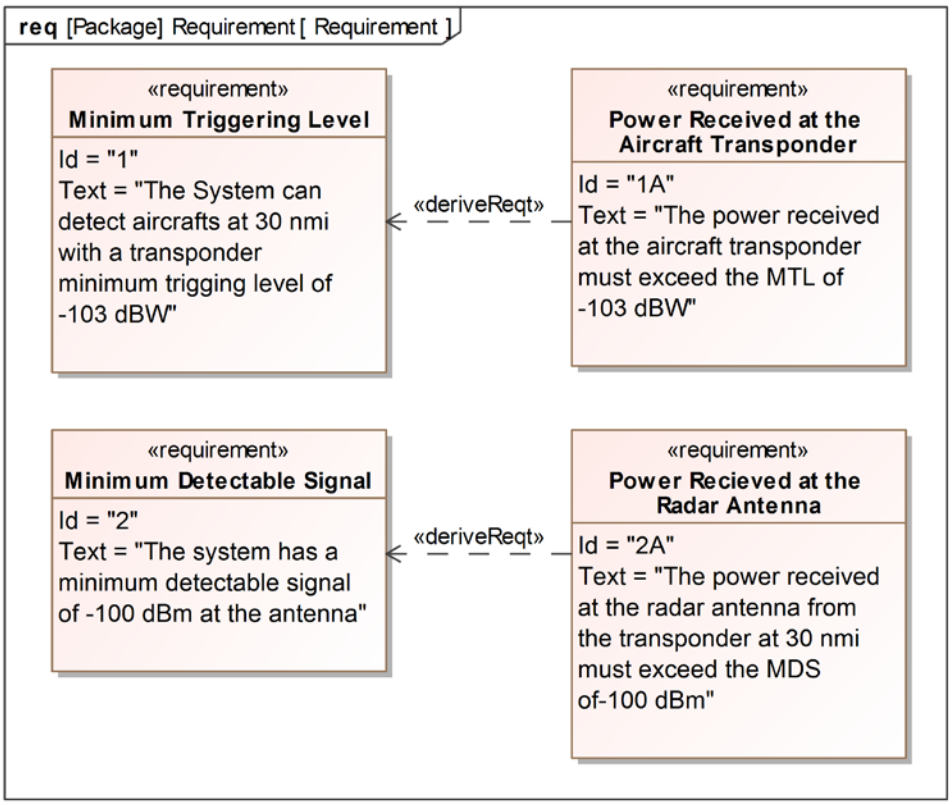

Figure 11-3 Design Objectives (Requirements) for a Radar System

Based on the texts given in the Requirements, constraints on these Design Objectives are modeled by the DesignObjectiveConstraint stereotype and are provided in the table depicted in Figure 11-4.

#	Applied Stereotype	docID	Name	Specification	Constrained Element
1	«» DesignObjectiveConstraint	1	{} P_RX	>=	V RX : Real S RX = -103.0
2	«» DesignObjectiveConstraint	2	{} P_RA	>=	V RA : Real S RA = -100.0

Figure 11-4 Design Objective Constraints for the Radar System

The modeled radar system components and their relations are presented in a SysML internal block diagram, as depicted in Figure 11-5. We identify two Design Variables that are affected by the defined Design Objective Constraints. They are:

- The power of the transmitter, P_T. This Property directly affects whether or not the signal transmitted to the aircraft can trigger the transponder with a MTL of -103dBW.
- The aperture of the Antenna, A_e. This Property directly affects whether or not the Radar system can receive a signal from an aircraft with -100 dBm.

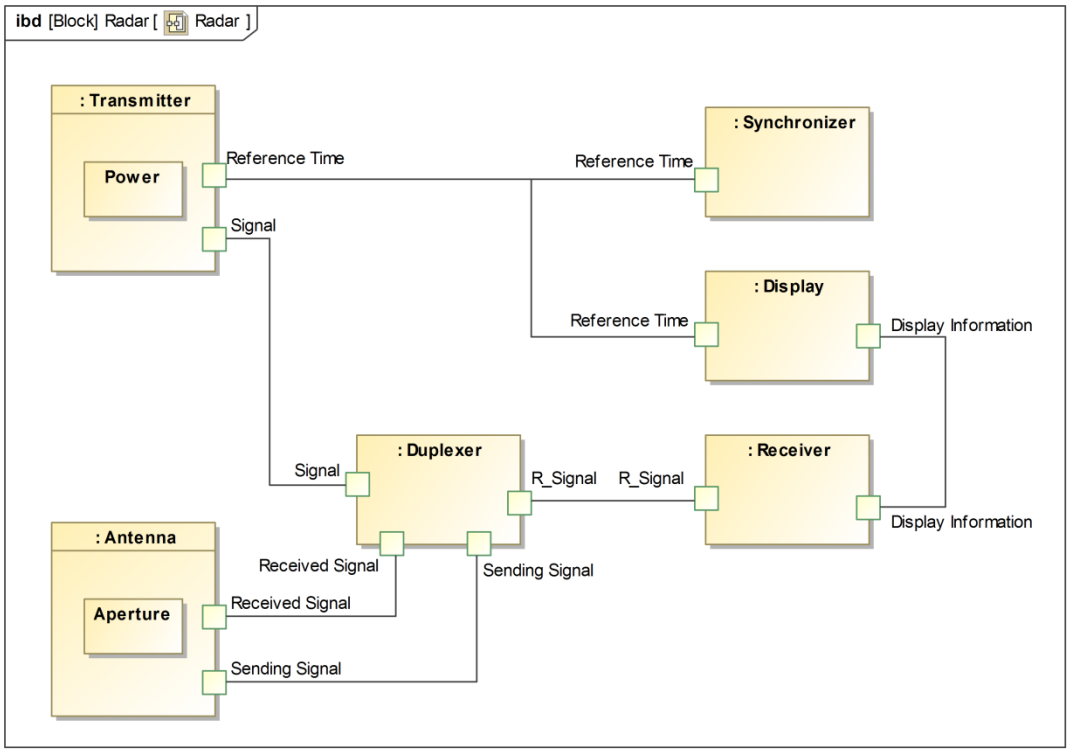

Figure 11-5 Radar system component architecture

Based on legacy designs or empirical knowledge, Design Ranges for the two Design Variables can be defined. These Design Ranges are modeled by the DesignVariableConstraints (c.f. Clause 10) and are provided in the table depicted in Figure 11-6. As shown, the Design Range specified for the power of the radar transmitter in dBW is less than or equal to 40, i.e. $P_T \leq 40$. And the initial design specified for P_T is 20. The Design Range specified for the aperture of the radar antenna in dBsm is less than or equal to 10, i.e. $A_e \leq 10$. And the initial design specified for A_e is 3.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	kind	initialDesign
1	«» DesignVariableConstrain	1.1	{ } P_T	<=	Power Power = "40"	designRange	20.0
2	«» DesignVariableConstrain	2.1	{ } A_e	<=	Aperture Aperture = "10"	designRange	3.0

Figure 11-6 Design Variable Constraints for the Radar System

To model the polynomial coefficients of the DOtoDVSensitivities, radar physics principles are used, where the P_{RX} and P_{RA} are related to the P_T and A_e through the following set of mathematical models:

$$P_{RX} = \frac{P_T A_e}{4\pi \mathrm{R}^2}$$

Equation 11-1

$$P_{RA} = \frac{P_{TX} A_e}{4\pi \mathrm{R}^2}$$

Equation 11-2

where P_{TX} is the power of the aircraft transponder with an assumed value of 20 dBW, R is the distance between the aircraft and the radar, and in this case, is equal to 30nmi based on the given requirements.

For typical Radar design problems, it is convenient to work with decibel (dB) as this linearizes the mathematical model. For instance, $4\pi R$ with $R = 30\text{nmi}$, can be converted into 106 dBsm, where 'sm' stands for square meter. The linearized model in dB then reads,

$$\begin{cases} P_{RX} = -106\ \text{dBsm} + P_T\ \text{dBW} + A_e\ \text{dBsm} \\ P_{RA} = -86\ \text{dBsm} + A_e\ \text{dBsm} \end{cases}$$

Equation 11-3

At given initial design values for P_T and A_e, based on the above linearized mathematical models, a first order univariate polynomial can be constructed for each of the mathematical relations between Design Objectives and Design Variables. For example, at $P_T = 20$ and $A_e = 3$, P_{RX} can be written as a first order polynomial of A_e as

$$P_{RX} = -86 + A_e$$

Equation 11-4

where the units are dropped for simplicity. The coefficients (including the constant term) can now be modeled by polynomialCoefficient and polynomialBase as shown in the first row of the table provided in Figure 11-7. The derivation of the rest of the polynomials will not be given, but their coefficients are also provided in this table.

#	Applied Stereotype	Client	Supplier	polynomialCoefficient	dataSourceType	polynomialBase
1	«» DOtoDVSensitivity	RX : Real	Power	1.0	derived	-103.0
2	«» DOtoDVSensitivity	RX : Real	Aperture	1.0	derived	-86.0
3	«» DOtoDVSensitivity	RA : Real	Aperture	1.0	derived	-86.0

Figure 11-7 Relational Structure for the Design of the Radar System

Using the information captured by the stereotypes, a CDD problem can be formulated and a RT-U can be applied to determine Feasible Ranges for the Design Variables. Using the initial designs as an initial condition, the results obtained are provided in the table depicted in Figure 11-8.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	kind	initialDesign
1	«» DesignVariableConstraint	1.1	{} P_TU	<=	Power Power = "40"	designRange	20.0
2	«» DesignVariableConstraint	1.2	{} P_TL	>=	Power Power = "17"	designRange	
3	«» DesignVariableConstraint	2.1	{} A_eU	<=	Aperture Aperture = "10"	designRange	3.0
4	«» DesignVariableConstraint	2.2	{} A_eL	>=	Aperture Aperture = "-14"	designRange	

Figure 11-8 Feasible Ranges on Radar System Design Variables

This page intentionally left blank.

Annex A Elevator Dispatcher System

(informative)

In this annex, UPR stereotypes are applied to model a CDD of an elevator dispatcher. The aim of this annex is to illustrate how UPR is used to annotate an existing system model to formulate a CDD problem. The system (elevator dispatcher) is modeled in UML. UPR stereotypes are applied to define Design Objective Constraints, Design Variables Constraints and Sensitivities for every Design Objective - Design Variable pair.

System Narrative

An elevator dispatcher needs to provide commands to a group of elevators so that they move in a manner to best meet the hall call and car call demands that emanate from passengers using the system. The system is real-time in the sense that it needs to be responsive to the passengers who arrive and leave the system and take into account the dynamics involved in moving the elevators themselves. A typical elevator dispatch system will re-calculate (or at least partially re-calculate) a new set of elevator demands many times in a second. Figure A-1 depicts the use cases of an elevator dispatcher system.

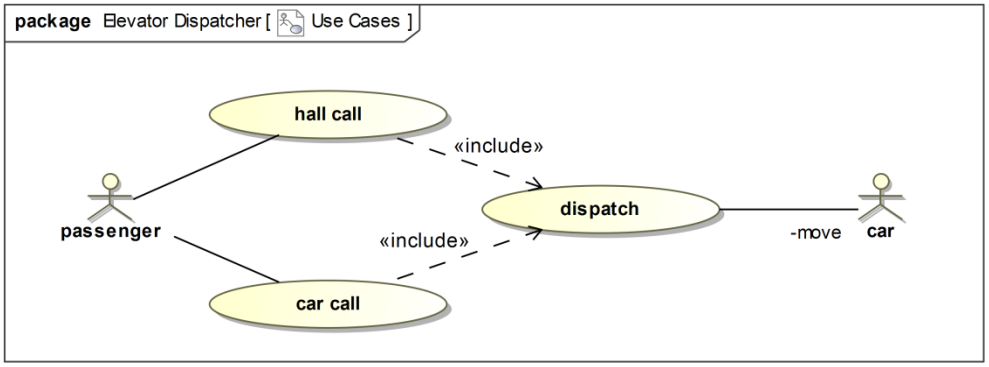

Figure A-1 Elevator Dispatcher Use Case Diagram

System Behavior

A real-time embedded elevator system needs to be capable of responding to both asynchronous and synchronous events. In this example, the arrival of passenger is an asynchronous event and this is signaled by the pressing of the hall call button. The car call could be considered as a second asynchronous event, but here it is assumed that the individual cars will monitor these events as part of their movement calculations. The synchronous events are the regular recalculation of the destinations of the cars by the dispatcher and the regular updating of the car destination (such a calculation will definitely require a model of the car dynamics if the system is being simulated, and will probably still require such a model if operating on a real physical elevator system). The activity diagram illustrating the real-time interrupt-driven nature of this application and incorporating the recording of objective values is shown in Figure A-2.

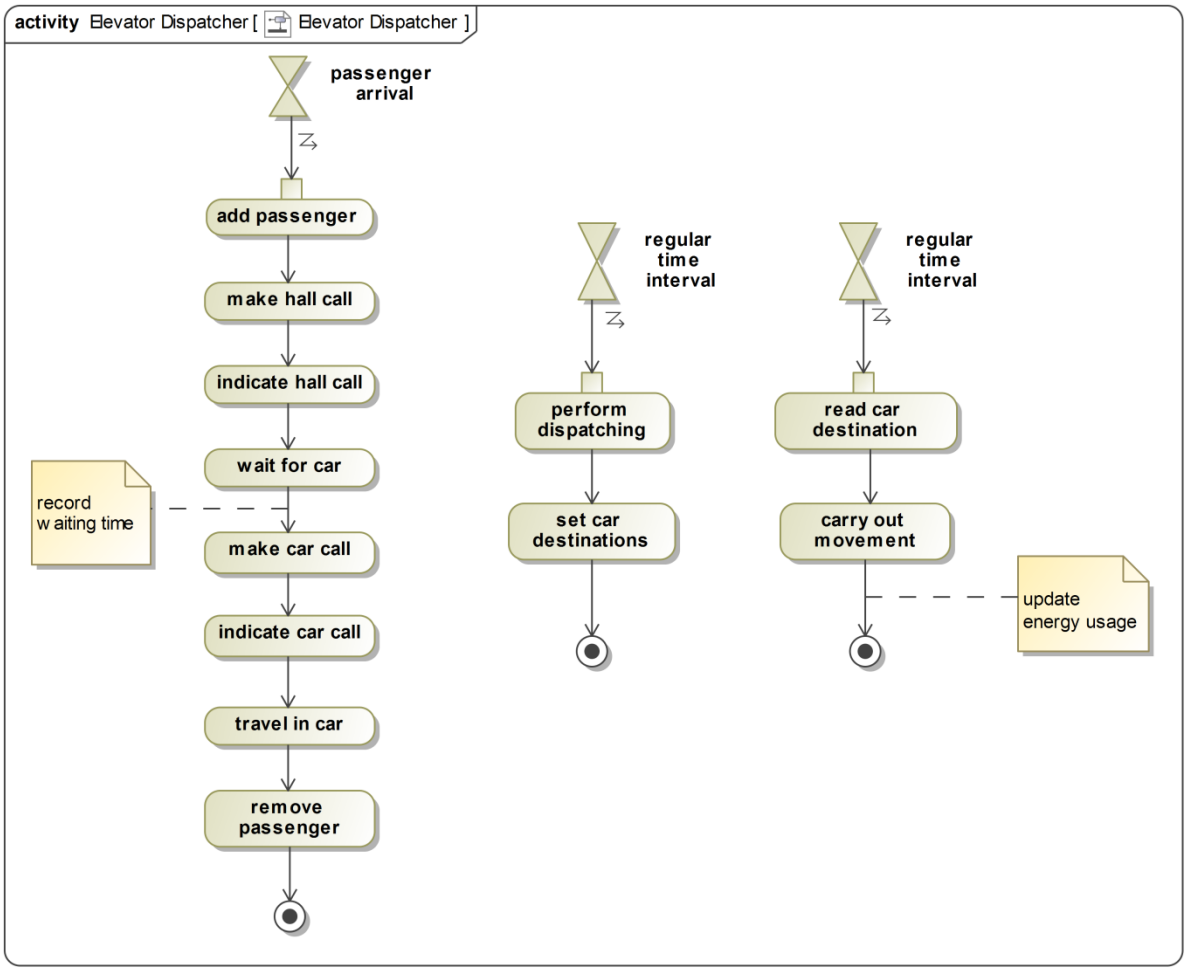

Figure A-2 Elevator Dispatcher Activity Diagram

System Structure

A Class Diagram for the elevator dispatcher is shown in Figure A-3 where the system components are modeled in Classes. Note that hall and car call buttons and indicators are good candidates for additional classes, but these have been incorporated into the *hall* and *car* classes respectively to keep the design simple and allow concentration on the relational aspects of the problem.

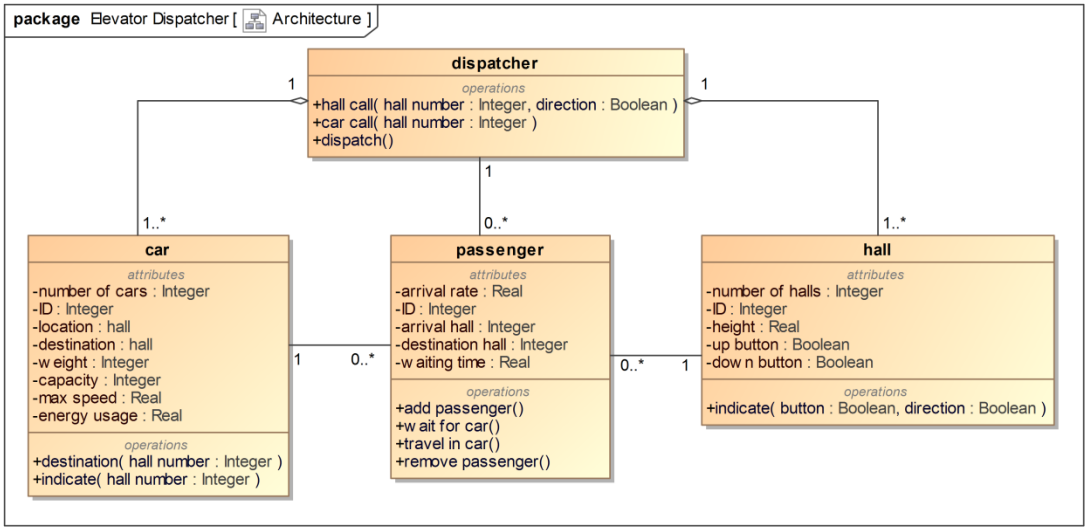

Figure A-3 Elevator Dispatcher Class Diagram

There are many Design Objective Constraints that are limiting the design space of an elevator system, and also many system elements that can be considered as Design Variables which are affected by Design Objective Constraints. For the purpose of illustration, only passenger waiting time and car energy usage are considered as Design Objective Constraints, and only car capacity (in terms of the maximum number of passengers in each car) and the number of cars (fewer cars means less usable floor space is taken in the building) are considered as relevant Design Variables.

A list of specified Design Objective Constraints and Design Variables Constraints are captured in the tables depicted in Figure A-4 and Figure A-5 respectively.

#	Applied Stereotype	docID	Name	Specification	Constrained Element
1	«» DesignObjectiveConstraint	1	{} Energy Usage	<=	-energy usage : Real energy usage = 25.0
2	«» DesignObjectiveConstraint	2	{} Waiting Time	<=	-waiting time: Real waiting time = 10.0

Figure A-4 Design Objective Constraints annotation with UPR stereotypes

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	kind	initialDesign
1	M «» DesignVariableConstraint	1.1	{} NoCL	>=	-number of cars : Integer number of cars = 1	designRange	[2,0] 4
2	M «» DesignVariableConstraint	1.2	{} NoCU	<=	-number of cars : Integer number of cars = 8	designRange	[2,0] 4
3	M «» DesignVariableConstraint	2.1	{} ARL	>=	-arrival rate : Real arrival rate = 1.0	designRange	[2,0] 2.3
4	M «» DesignVariableConstraint	2.2	{} ARU	<=	-arrival rate : Real arrival rate = 10.0	designRange	[2,0] 2.3

Figure A-5 Design Variable Constraints annotation with UPR stereotypes

Consequently, a suitable relational structure can be created. This is given in the table shown in Figure A-6. As the exact mathematical model for the relations is not known to the authors, for the purpose of demonstration, estimated polynomial coefficients are used to model Sensitivities.

#	Applied Stereotype	Source	Target	polynomialCoefficient	dataSourceType	polynomialBase
1	«» DOtoDVSensitivity	-energy us...	-number of c...	3.0	derived	0.0
2	«» DOtoDVSensitivity	-energy us...	-arrival rate...	0.8	derived	0.0
3	«» DOtoDVSensitivity	-waiting time...	-number of c...	-20.0	derived	0.0
4	«» DOtoDVSensitivity	-waiting time...	-arrival rate...	13.0	derived	0.0

Figure A-6 Relational structure (Sensitivities) annotation with UPR stereotypes

This page intentionally left blank.

Annex B Vehicle Design

(informative)

In this annex, we demonstrate how UPR is applied to multiple domains in order to model domain-specific CDD problems, while also demonstrating how the stereotypes facilitate cross-domain tradeoffs. The example used in this annex is primarily for demonstrating UPR, therefore any engineering values or design is purely for illustrative purposes only.

The vehicle system can be considered as a system of systems. The vehicle design task in practice is often decomposed into several domain-specific designs. Each design team is given a particular design task to design a subsystem of the vehicle. The design of different subsystems may be driven by different Design Objectives (that may or may not be constrained) as well as the same Design Objectives. In addition, two different design teams may have shared or dependent Design Variables. As such, when integrating subsystems designs together to form the overall vehicle design, there may be potential issues. For instance, there can be conflicts between the recommendations on solutions to Design Variables; there can also be situations where the design solution for one subsystem causes the solution of another subsystem to fail a Design Objective Constraints. We shall demonstrate how UPR can facilitate design tradeoffs without the need of building cross-domain analytics and simulations models.

B.1 Vehicle System Models

The vehicle system architecture is modeled in SysML as shown in a SysML Block Definition Diagram in Figure B-1 and an Activity Diagram Figure B-2. In Figure B-1, major vehicle components and their properties are provided. This model is a greatly simplified compared to real world vehicle design. However, the components are representative and sufficient for the purpose of demonstrating how UPR facilitates the modeling and analysis of CDD problems. In Figure B-2, typical vehicle behavior during a drive cycle is given. It involves major vehicle functions relevant to the drive cycle emissions test.

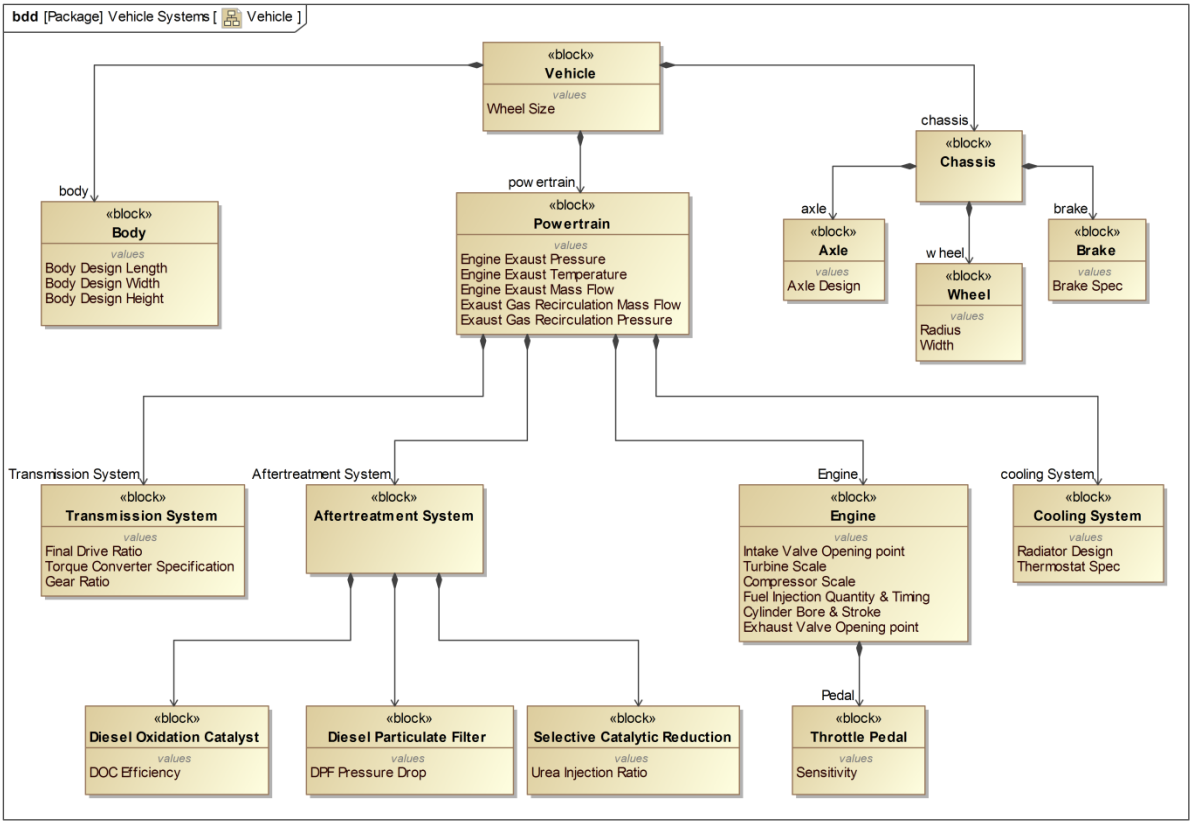

Figure B-1 Vehicle subsystems and properties

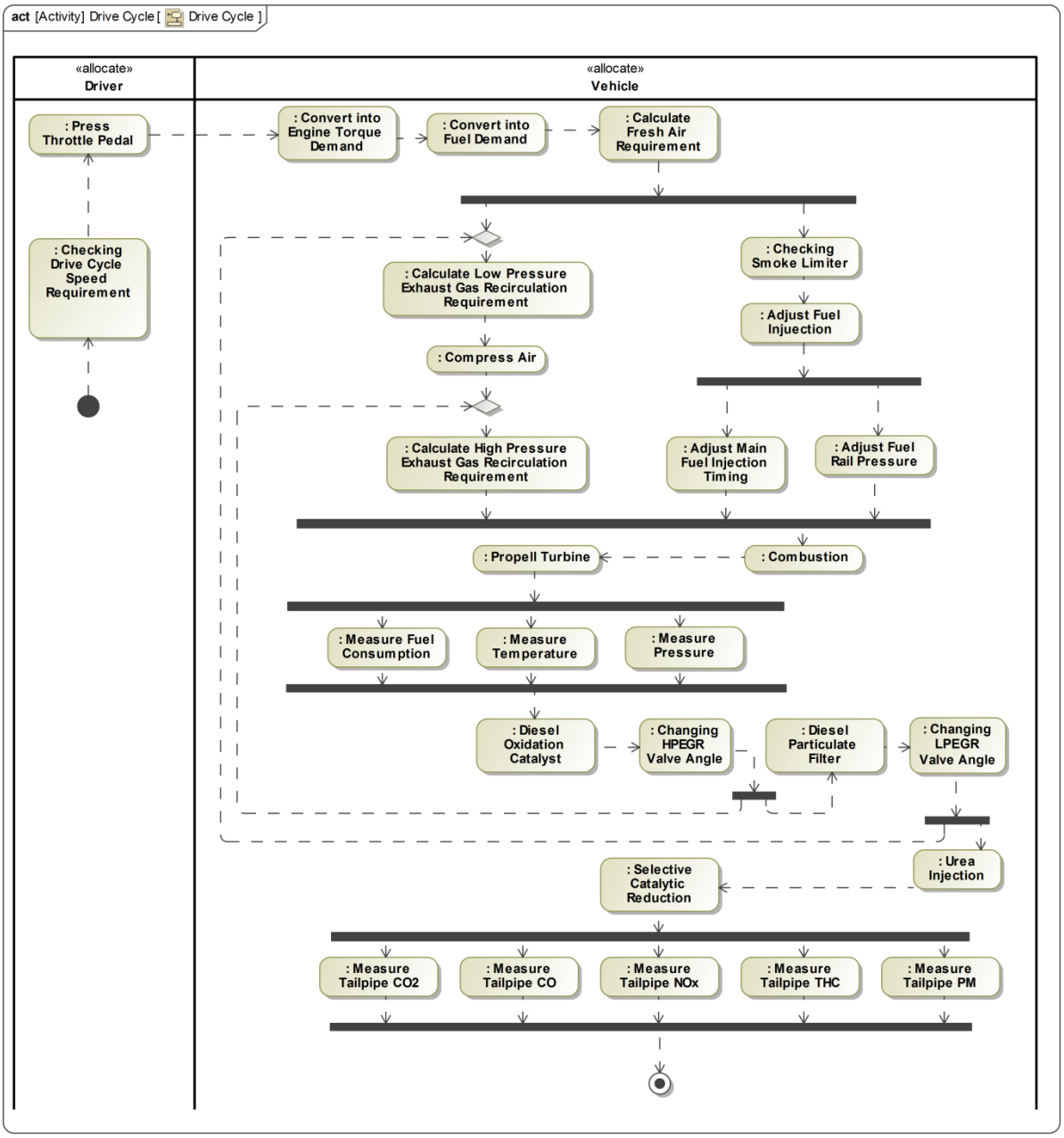

Figure B-2 Vehicle behavior in a drive cycle

The following system requirements, as depicted in the SysML Requirement diagram in Figure B-3, are provided as the basis to define system-level Design Objectives (captured by value properties as in Blocks, Figure B-3) and Design Objective Constraints using UPR. The constraints are provided in the table in Figure B-4.

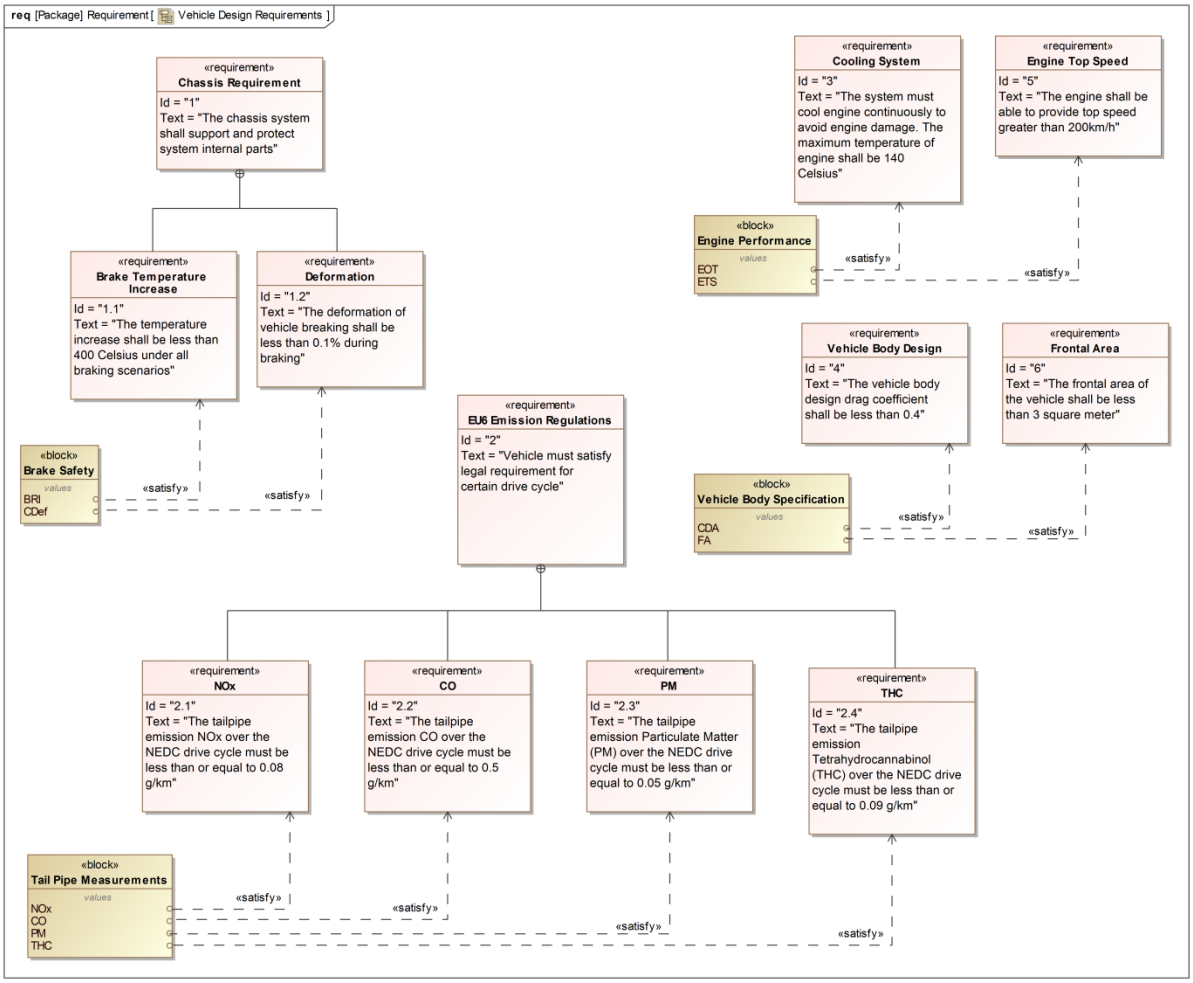

Figure B-3 Vehicle system-level requirements

#	Applied Stereotype	△ ○ docID	Name	Specification	Constrained Element
1	«» DesignObjectiveConstraint	1.1	{} BRI	<	V BRI S BRI = "400"
2	«» DesignObjectiveConstraint	1.2	{} CDef	<	V CDef S CDef = "0.1"
3	«» DesignObjectiveConstraint	2.1	{} NOx	<=	V NOx S NOx = "0.08"
4	«» DesignObjectiveConstraint	2.2	{} CO	<=	V CO S CO = "0.5"
5	«» DesignObjectiveConstraint	2.3	{} PM	<=	V PM S PM = "0.005"
6	«» DesignObjectiveConstraint	2.4	{} THC	<=	V THC S THC = "0.09"
7	«» DesignObjectiveConstraint	3.1	{} EOT	<=	V EOT S EOT = "140"
8	«» DesignObjectiveConstraint	3.2	{} ETS	>	V ETS S ETS = "200"
9	«» DesignObjectiveConstraint	4.1	{} CDA	<	V CDA S CDA = "0.4"
10	«» DesignObjectiveConstraint	4.2	{} FA	<	V FA S FA = "3"

Figure B-4 Vehicle system-level Design Objective Constraints

Based on typical engineering practices and the models provided, the following subsystem design can be defined:

1. Body design
2. Engine design
3. Aftertreatment systems design
4. Transmission system design
5. Chassis design
6. Cooling system design

To limit the scope of this example, only the first three subsystem designs will be considered. However, it is worth emphasizing that the application of UPR is not limited by the scope and that increasing the number of subsystem designs does not add modeling complexity, with UPR, to individual domains.

B.2 Vehicle Body Design

In the Body design, the following table in Figure B-5 provides a list of modeled Design Variables Constraints.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	initialDesign	kind
1	«» DesignVari	1.1	{} BH	<=	Body Design Height Body Design Height = "1.42"	1.4	designRange
2	«» DesignVari	1.2	{} BL	<=	Body Design Length Body Design Length = "4.7"	4.6	designRange
3	«» DesignVari	1.3	{} BW	<=	Body Design Width Body Design Width = "1.85"	1.85	designRange

Figure B-5 Vehicle Body Design Variable Constraints

A relational structure for the Body CDD problem is then modeled using a set of DOtoDVSensitivities in the table depicted in Figure B-6. The polynomial coefficients presented are illustrative figures.

#	Applied Stereotype	Source	Target	polynomialCoefficient	polynomialBase	dataSourceType
1	«» DOtoDVSensitivity	NOx	Body Design Height	-0.1429 0.6143	-0.5171	derived
2	«» DOtoDVSensitivity	NOx	Body Design Width	-8.1633 28.449	-24.6237	derived
3	«» DOtoDVSensitivity	FA	Body Design Width	1.42	0.0	derived
4	«» DOtoDVSensitivity	NOx	Body Design Length	1.42	6.7449	derived
5	«» DOtoDVSensitivity	FA	Body Design Height	1.85	0.0	derived

Figure B-6 Relational structure for the vehicle Body CDD

B.3 Vehicle Engine Design

In the Engine design, the following table in Figure B-7 provides a list of modeled Design Variables Constraints.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	initialDesign	kind
1	«» DesignVari	2.1.1	{ } EXTL	>=	Engine Exaust Temperature Engine Exaust Temperature = "400"	600.0	designRange
2	«» DesignVari	2.1.2	{ } EXTU	<=	Engine Exaust Temperature Engine Exaust Temperature = "800"		designRange
3	«» DesignVari	2.2.1	{ } EXMFL	>=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "20"	595.0	designRange
4	«» DesignVari	2.2.2	{ } EXMFU	<=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "600"		designRange
5	«» DesignVari	2.3.1	{ } IntakeVL	>=	Intake Valve Opening point Intake Valve Opening point = "440"	460.0	designRange
6	«» DesignVari	2.3.2	{ } IntakeVU	<=	Intake Valve Opening point Intake Valve Opening point = "480"		designRange
7	«» DesignVari	2.4.1	{ } OutVU	<=	Exhaust Valve Opening point Exhaust Valve Opening point = "280'	260.0	designRange
8	«» DesignVari	2.4.2	{ } OutVL	>=	Exhaust Valve Opening point Exhaust Valve Opening point = "240'		designRange

Figure B-7 Vehicle Engine Design Variable Constraints

A relational structure for the engine systems CDD problem is then modeled using a set of DOtoDVSensitivities in the table depicted in Figure B-8. The polynomial coefficients presented are illustrative figures.

#	Applied Stereotype	Source	Target	polynomialCoefficient	polynomialBase	dataSourceType
1	«» DOtoDVSensitivity	ETS	Intake Valve Opening point	-0.175 161.7	-3.7103	derived
2	«» DOtoDVSensitivity	ETS	Exhaust Valve Opening point	-0.25 129.0	-1.6391	derived
3	«» DOtoDVSensitivity	NOx	Engine Exaust Temperature	-5.0E-4	0.34	derived
4	«» DOtoDVSensitivity	NOx	Engine Exaust Mass Flow	-0.83	0.173	derived
5	«» DOtoDVSensitivity	NOx	Intake Valve Opening point	0.0028	-1.1933	derived
6	«» DOtoDVSensitivity	NOx	Exhaust Valve Opening point	-0.0031	0.8067	derived
7	«» DOtoDVSensitivity	PM	Engine Exaust Temperature	-1.5E-5	0.011	derived
8	«» DOtoDVSensitivity	PM	Engine Exaust Mass Flow	0.025	0.001	derived
9	«» DOtoDVSensitivity	PM	Intake Valve Opening point	-8.3E-5	0.042	derived
10	«» DOtoDVSensitivity	PM	Exhaust Valve Opening point	7.9E-5	-0.018	derived

Figure B-8 Relational structure for the vehicle Engine CDD

B.4 Vehicle Aftertreatment Systems Design

In the Aftertreatment Systems design, the following table in Figure B-9 provides a list of modeled Design Variables Constraints.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	initialDesign	kind
1	«» DesignVari	3.1.1	{ } EXTL	>=	Engine Exaust Temperature Engine Exaust Temperature = "400"	600.0	designRange
2	«» DesignVari	3.1.2	{ } EXTU	<=	Engine Exaust Temperature Engine Exaust Temperature = "800"		designRange
3	«» DesignVari	3.2.1	{ } EXMFL	>=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "20"	595.0	designRange
4	«» DesignVari	3.2.2	{ } EXMFU	<=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "600"		designRange
5	«» DesignVari	3.3.1	{ } CCL	>=	DOC Efficiency DOC Efficiency = "65"	90.0	designRange
6	«» DesignVari	3.3.2	{ } CCU	<=	DOC Efficiency DOC Efficiency = "98"		designRange
7	«» DesignVari	3.4.1	{ } DPFL	>=	DPF Pressure Drop DPF Pressure Drop = "2"	9.0	designRange
8	«» DesignVari	3.4.2	{ } DPFU	<=	DPF Pressure Drop DPF Pressure Drop = "13"		designRange
9	«» DesignVari	3.5.1	{ } SCRL	>=	Urea Injection Ratio Urea Injection Ratio = "0.98"	1.0	designRange
10	«» DesignVari	3.5.2	{ } SCRU	<=	Urea Injection Ratio Urea Injection Ratio = "1.02"		designRange

Figure B-9 Vehicle Aftertreatment System Design Variable Constraints

A relational structure for the vehicle aftertreatment systems CDD problem is then modeled using a set of DOtoDVSensitivities in the table depicted in Figure B-10. The polynomial coefficients presented are illustrative figures.

#	Applied Stereotype	Source	Target	polynomialCoefficient	polynomialBase	dataSourceType
1	«» DOtoDVSensitivity	NOx	Engine Exaust Temperature	-5.0E-4	0.34	derived
2	«» DOtoDVSensitivity	NOx	Engine Exaust Mass Flow	-0.83	0.173	derived
3	«» DOtoDVSensitivity	NOx	DPF Pressure Drop	-1.5385E-7	0.1569	derived
4	«» DOtoDVSensitivity	NOx	DOC Efficiency	0.0833	0.0583	derived
5	«» DOtoDVSensitivity	NOx	Urea Injection Ratio	-0.7143	0.7971	derived
6	«» DOtoDVSensitivity	PM	DPF Pressure Drop	4.6154E-9	0.0015	derived
7	«» DOtoDVSensitivity	PM	DOC Efficiency	-0.0125	0.0143	derived
8	«» DOtoDVSensitivity	PM	Urea Injection Ratio	0.0071	-0.0026	derived

Figure B-10 Relational structure for the vehicle Aftertreatment Systems CDD

B.5 Design Integration

Using derived polynomials, it is possible to determine Feasible Ranges with design algorithms that implement RT-U. However, this is not within the scope of UPR. Nonetheless, when integrating the domain designs into the final vehicle design, based on the above models, it is recognized that EXT and EXMF are two Design Variables shared by both the Engine design team and the Aftertreatment System design team. It shall now be assumed that the following Feasible Ranges[8] , as shown in the table depicted in Figure B-11, are determined:

[8] These numbers illustrative purposes only and do not represent a real product.

#	Applied Stereotype	dvcID	Name	Specification	Constrained Element	kind
1	«» DesignVari	D.E1.1	{} EXTL_Engine	>=	Engine Exaust Temperature Engine Exaust Temperature = "420"	feasibleRange
2	«» DesignVari	D.E1.2	{} EXTU_Engine	<=	Engine Exaust Temperature Engine Exaust Temperature = "630"	feasibleRange
3	«» DesignVari	D.A1.1	{} EXTL_AT	>=	Engine Exaust Temperature Engine Exaust Temperature = "500"	feasibleRange
4	«» DesignVari	D.A1.2	{} EXTU_AT	<=	Engine Exaust Temperature Engine Exaust Temperature = "700"	feasibleRange
5	«» DesignVari	D.E2.1	{} EXMFL_Engine	>=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "250"	feasibleRange
6	«» DesignVari	D.E2.2	{} EXMFU_Engine	<=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "350"	feasibleRange
7	«» DesignVari	D.A2.1	{} EXMFL_AT	>=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "400"	feasibleRange
8	«» DesignVari	D.A2.2	{} EXMFU_AT	<=	Engine Exaust Mass Flow Engine Exaust Mass Flow = "550"	feasibleRange

Figure B-11 Design integration for shared Design Variables

In this table, it is recognized that there is an overlap between the two Feasible Ranges on EXT determined by the two domain teams. Therefore, it is relatively obvious to conclude that an integrated solution range should be simply $500 \leq \text{EXT} \leq 630$. And it is obvious that any point-based solution within this solution range will ensure that relevant Design Objective Constraints will be satisfied. There is certainly no need to create a mathematical model that combines the two domains to obtain feasible designs that satisfies Design Objective Constraints affecting the two domains.

However, it is also recognized that there is not such a similar overlap between the two Feasible Ranges on EXMF. In these cases, a further tradeoff analysis is required. For instance, the engine team can iterate their design solutions and adjust Feasible Ranges on other Design Variables, to create larger margins for EXMF. If both teams sacrifice some of the margins in other Design Variables that are not shared between the two domains, it is very likely that an overlapping region will suffice. This tradeoff analysis again does not require a new mathematical model to be built.

Finally, for completeness, it is possible to integrate all domain-specific CDDs into a single CDD by merging all the DesignObjectiveConstraints, DesignVariableConstraints, and DOtoDVSensitivities into one design package.