The Open Group Standard

Open Universal Domain Description Language (Open UDDL) Edition 1.0

The Open Group Standard

Open Universal Domain Description Language (Open UDDL), Edition 1.0

ISBN: 1-947754-32-4

Document Number: C198

Published by The Open Group, July 2019.

Comments relating to the material contained in this document may be submitted to:

The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom

or by electronic mail to:

ogface-admin@opengroup.us

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 600 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Open Group Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/library.

This Document

This document is The Open Group Standard for the Open Universal Domain Description Language (Open UDDL). It has been developed and approved by The Open Group FACE™ Consortium.

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this document.

Principal Authors

- Ryan Brooks, Boeing
- Spencer Crosswy, NAVAIR
- James “Bubba” Davis, U.S. Army CCDC AvMC, Domain Interoperability Working Group (DAWG) Vice-Chair
- Stuart Frerking, U.S. Army CCDC AvMC
- Bill Kinahan, Skayl, Domain Interoperabilty Working Group (DAWG) Chair

Additional Contributors

- Chis Allport, Skayl
- Robert Daniels, U.S. Army PEO Aviation
- John Gray, U.S. Army PEO Aviation
- Gordon Hunt, Skayl
- Leanne May, Rockwell Collins
- Sean Mulholland, TES-SAVi
- Bill Tanner, TES-SAVi

Referenced Documents

The following documents are referenced in this Standard.

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

Normative References

Normative references for this document are defined in Section 1.4.

Informative References

- Extensible Markup Language (XML) Schema Definition (XSD), W3C; refer to: https://www.w3.org/XML/Schema
- FACE™ Technical Standard, Edition 3.0, The Open Group Standard (C17C), November 2017, published by The Open Group; refer to: www.opengroup.org/library/c17c
- Open Data Element Framework (O-DEF™), Version 1.0, The Open Group Standard (C163), May 2016, published by The Open Group; refer to: www.opengroup.org/library/c163

1 Introduction

1.1 Objective

The purpose of the Universal Domain Description Language (UDDL) is to define a data modeling language for formally describing, querying, and communicating information. The specific objectives of this document are to:

- Formally document the meaning of data with the goal of eliminating ambiguity:
  - — Document the semantic meaning of data to support machine-readability and the use of information in the engineering process; this includes distinguishing the identity of data, clearly specifying context of data, and providing measurable building blocks characterized with units and a frame of reference
  - — Separate concerns relative to the appropriate level of abstraction in order to aid understandability and embrace a model-based engineering approach
- Facilitate information exchange within any domain:
  - — Allow specification of a model subset for information exchange while maintaining the contextual integrity of the projected data
  - — Facilitate interoperability during information exchange by supporting automated data conversions and transformations and the ability to compare and merge data
- Provide extensibility:
  - — Provide flexibility in using the UDDL by allowing tailoring by other standards through a customization mechanism such as extension points
  - — Allow extension of the metamodel (i.e., allow the addition of an attribute to the existing metamodel)
- Facilitate world-wide use and distribution:
  - — Allow design time and run-time cross-platform support by avoiding constraints and assumptions that unnecessarily tie the use of the UDDL to a given platform
  - — Allow unlimited distribution and usage of the standard (royalty-free) and support internationalization

1.2 Overview

1.2.1 Background

Adequately describing domain data is a common problem in computer-based systems. Typical approaches involve creating documentation to accompany artifacts, such as models or source code, in an attempt to capture the semantics of the data exchanged. While many different techniques exist for data modeling, an analysis of available technologies showed that none met all of the needs, so a new approach for capturing data semantics was developed.

The UDDL is the result of this effort. It leverages entity and relationship modeling with a novel approach allowing for refinement and appropriate extension of semantic elements. Additionally, the UDDL provides for common data models that may be reused and shared to enhance interoperability between software development efforts.

1.2.2 Technical Approach

The UDDL utilizes a multi-level approach to modeling entities and associations at the conceptual, logical, and platform levels, enabling gradual and varying degrees of specificity. The multi-level modeling provides a context for the specification of views. A selection mechanism is provided through a *Query*, as a means of specifying a *View* that projects a set of characteristics of the entity model for communicating data. The *Query* is a specification and is not necessarily executed.

The UDDL is specified by:

- An Essential Meta-Object Facility (EMOF) metamodel in Extensible Markup Language (XML) Metadata Interchange (XMI)
- A set of Object Constraint Language™ (OCL™) constraints that supply semantic rules to which data model content adheres
- A query grammar to specify selection of data
- A set of English language constraints for the query grammar

1.2.3 Applicability

A few potential use-cases are provided to give an idea of the broad applicability of the UDDL standard:

- System integration (e.g., the FACE™ Technical Standard, a standard of The Open Group)
- System-of-systems integration
- Model-based engineering
- Industrial Internet of Things (IIoT)
- Specification development
- Software component development
- Interface data documentation (e.g., ICD)
- Data store schema development

In addition, the UDDL standard is complementary to The Open Group Open Data Element Framework (O-DEF™); a semantic interoperability standard which enables types and properties of data to be classified, so that equivalences between them can be easily determined. Data models developed in accordance with the UDDL standard can utilize the O-DEF standard to determine a common vocabulary bridge between data models, and enable gap analysis and discovery of where multiple data models can be made interoperable.

1.3 Conformance

The UDDL standard does not define a conformance process. However, a UDDL Data Model is defined to be conformant if it meets all the requirements set forth in this document.

1.4 Normative References

The following standards contain provisions which, through references in this document, constitute provisions of the UDDL standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

- Extensible Markup Language (XML), Version 1.0 (Fifth Edition), November 2008, W3C; refer to: https://www.w3.org/TR/xml
- IEEE Std 754-2008: IEEE Standard for Floating-Point Arithmetic, August 2008; refer to: https://ieeexplore.ieee.org/document/4610935
- ISO/IEC 14977:1996: Information Technology – Syntactic Metalanguage – Extended BNF; refer to: https://www.iso.org/standard/26153.html
- Meta-Object Facility™ (MOF™) Specification, Version 2.0, Object Management Group (OMG), January 2006; refer to: www.omg.org/spec/MOF/2.0
- Object Constraint Language™ (OCL™), Version 2.4, Object Management Group (OMG), February 2014; refer to: www.omg.org/spec/OCL/2.4

1.5 Terminology

For the purposes of the UDDL standard, the following terminology definitions apply:

May	Describes a feature or behavior that is optional. Its presence cannot be relied upon.
Shall	Describes a feature or behavior that is mandatory for an implementation that conforms to the standard. Its presence can be relied upon.
Should	Describes a feature or behavior that is strongly recommended for an implementation that conforms to the standard. Its presence cannot be relied upon.

1.6 Future Directions

None.

2 Definitions

For the purposes of the UDDL standard, the following terms and definitions apply. Merriam-Webster's Collegiate Dictionary should be referenced for terms not defined in this section.

2.1 Data Architecture

A set of related models, specifications, and governance policies with the primary purpose of providing an interoperable means of data exchange.

2.2 Data Model Language

A language specified as an Essential MetaObject Facility (EMOF) metamodel and Object Constraint Language (OCL) constraints used to capture data element syntax and semantics.

2.3 Conceptual Data Model (CDM)

A data model comprised of Entities, Characteristics, and Associations that provide definitions of concepts, their characteristics, and the context relating them. Observables that are fundamental to the domain and have no further decomposition are used to specify these defining features. Domain concepts are captured in the CDM through the definition of Basis Entities. A Basis Entity represents a unique domain concept and establishes a foundation from which conceptual Entities may be specialized. Observables and Basis Entities are axiomatic. This allows for separation of concerns, allowing multiple domains to be modeled.

2.4 Logical Data Model (LDM)

A data model consisting of Entities, Characteristics, and Associations that realize their definition from the CDM. It provides terms of Measurement Systems, Coordinate Systems, Reference Points, Value Domains, and Units. The principal level of detail added in an LDM is provided through frames of reference for representing characteristic values. Multiple LDM elements may realize a single CDM element.

2.5 Platform Data Model (PDM)

A data model consisting of Entities, Characteristics, and Associations that realize their counterpart definition from an LDM. In a PDM, specific representation details such as data type and precision are provided to represent characteristics. Multiple PDM elements may realize a single LDM element.

3 General Concepts

For the purposes of the UDDL standard the general concepts provided in this chapter apply.

3.1 Data Model Language Elements

A UDDL Data Model consists of Conceptual, Logical, and Platform Data Models, each of which describes data elements at progressing levels of refinement. All three levels combine to provide a complete definition of each element. The roles of the models are:

- Conceptual – defines the complete semantics of all Entities and Associations, typed with Observables – traits which can be observed, but not further characterized
- Logical – refines Entities and Associations typed with Measurements for all elements or a subset identified using using down-selection[1]

  Measurements and their associated elements refine Observables by adding detail such as frame of reference, value type, units, and constraints.
- Platform – refines Entities and Associations typed with Platform Data Types for all elements or a subset identified using down-selection

  Platform Data Types refine measurements by adding specific platform details with language binding references.

Figure 1 illustrates the UDDL Data Model elements and their refinement from abstract to more concrete (shown horizontally, from left to right using black arrows). View projection, shown vertically using gray arrows, is used to select a subset of the data model to create new structural representations of the data.

[1] Down-selection is the process of leaving out entire Entities/Associations, or a subset of their Characteristics when they are not needed for View Projection.

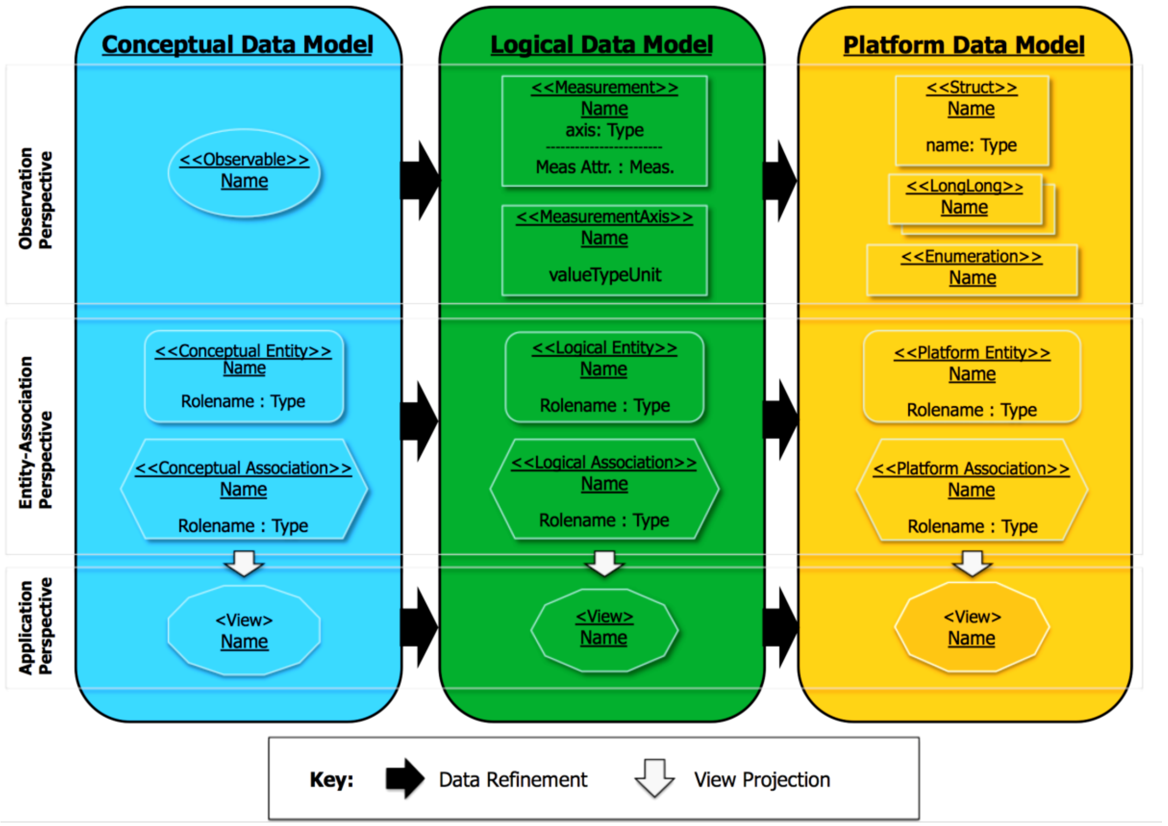
Conceptual Data Model
Logical Data Model
Platform Data Model
Observation Perspective
Entity-Association Perspective
Application Perspective
<<Observable>> Name
<<Measurement>> Name axis: Type Meas Attr. : Meas.
<<MeasurementAxis>> Name valueTypeUnit
<<Struct>> Name name: Type
<<LongLong>> Name
<<Enumeration>> Name
<<Conceptual Entity>> Name Rolename : Type
<<Logical Entity>> Name Rolename : Type
<<Platform Entity>> Name Rolename : Type
<<Conceptual Association>> Name Rolename : Type
<<Logical Association>> Name Rolename : Type
<<Platform Association>> Name Rolename : Type
<View> Name
<View> Name
<View> Name
Key: Data Refinement View Projection

Figure 1: Data Model Refinement, Perspectives, and Primary Language Elements

Figure 1 also illustrates key Data Model Language elements, organized into three perspectives that span model-level boundaries: Observation, Entity-Association, and Application. Each perspective is shown as a horizontal box with increasing refinement from left to right. These perspectives are defined as follows:

- Observation perspective: describes data elements (Observables, Measurements, Platform Data Types) used to type Entity and Association characteristics
- Entity-Association perspective: describes data elements that define the meaning and context of application concepts and their relationships
- Application perspective: describes the application (or use) of data elements by software components supporting component interface integration

While the perspectives are not actual model elements, they aid in understanding the cross-cutting aspects of data element refinement and the varying metamodel elements used to type them.

3.2 Data Model Language

The Data Model Language for the UDDL standard is formally specified as an OMG Essential Meta-Object Facility (EMOF) metamodel and OMG Object Constraint Language (OCL) constraints used to capture data elment syntax and semantics. The metamodel and associated constraints are specified in Chapter 7 and Chapter 8.

3.3 Extension by Other Standards

The UDDL is defined using EMOF which allows other standards to extend its metamodel using the EMOF reference mechanism. (See the Object Management Group (OMG) MOF Specification listed in Section 1.4.)

4 UDDL Requirements

A valid UDDL Data Model conforms to both the metamodel and associated constraints:

1. A UDDL Data Model shall be an XMI file conforming to the EMOF metamodel specified in Chapter 7.
2. A UDDL Data Model shall use the “xmi:id” attribute with a unique UUID as the ID for all elements.
3. A UDDL Data Model shall adhere to the Metamodel Constraints (OCL) in Chapter 8.
4. A UDDL Data Model shall adhere to the query grammar and constraints in Chapter 6.

5 Metamodel Description

The description in this chapter is a depiction of the metamodel defined by the normative EMOF specification in Chapter 7.

5.1 Metamodel Diagram Legend

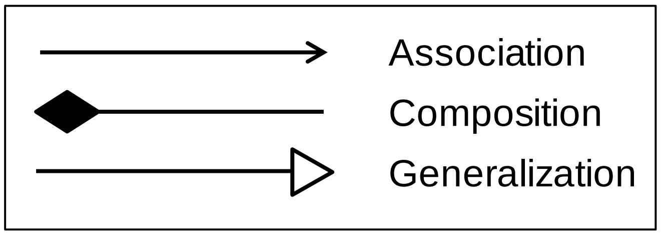

Figure 2: MOF Symbology

Metamodel diagrams in the following sections contain three basic connector types: Association, Composition, and Generalization. An *Association* type represents a relationship between elements. It has a role-name that differentiates the Association from other Associations. A *Composition* type defines a containment relationship. The element at the filled diamond end of the connector is the containing element. A *Generalization* connector represents an “is-a” relationship. The element at the plain end of the connector is considered to be of the same type as the element at the triangle terminator. Generalization does not imply inheritance in the object oriented sense. Polymorphism is not represented by the Generalization connector.

5.2 Meta-Package: datamodel

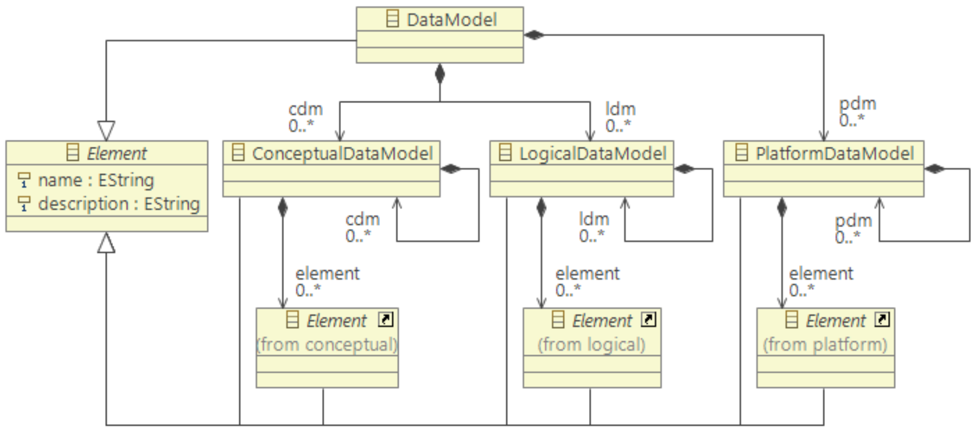

Figure 3: Data Model Metamodel “datamodel” Package

5.2.1 Meta-Class: datamodel.DataModel

Description

A DataModel is a container for ConceptualDataModels, LogicalDataModels, and PlatformDataModels. The relationships for the DataModel meta-class are listed in Table 1.

Table 1: datamodel.DataModel Relationships

Type	Name	Target	Multiplicity
Composition	cdm	ConceptualDataModel	0..*
Composition	ldm	LogicalDataModel	0..*
Composition	pdm	PlatformDataModel	0..*
Generalization		Element	

5.2.2 Meta-Class: datamodel.Element

Description

An Element is the root type for defining all named elements in the DataModel. The "name" attribute captures the name of the Element in the model. The "description" attribute captures a description for the Element. The attributes for the Element meta-class are listed in Table 2.

Table 2: datamodel.Element Attributes

Name	Type	Multiplicity
name	string	1
description	string	1

5.2.3 Meta-Class: datamodel.ConceptualDataModel

Description

A ConceptualDataModel is a container for CDM Elements. The relationships for the ConceptualDataModel meta-class are listed in Table 3.

Table 3: datamodel.ConceptualDataModel Relationships

Type	Name	Target	Multiplicity
Composition	element	datamodel.conceptual.Element	0..*
Composition	cdm	ConceptualDataModel	0..*
Generalization		Element	

5.2.4 Meta-Class: datamodel.LogicalDataModel

Description

A LogicalDataModel is a container for LDM Elements. The relationships for the LogicalDataModel meta-class are listed in Table 4.

Table 4: datamodel.LogicalDataModel Relationships

Type	Name	Target	Multiplicity
Composition	element	datamodel.logical.Element	0..*
Composition	ldm	LogicalDataModel	0..*
Generalization		Element	

5.2.5 Meta-Class: datamodel.PlatformDataModel

Description

A PlatformDataModel is a container for platform Data Model elements. The relationships for the PlatformDataModel meta-class are listed in Table 5.

Table 5: datamodel.PlatformDataModel Relationships

Type	Name	Target	Multiplicity
Composition	element	datamodel.platform.Element	0..*
Composition	pdm	PlatformDataModel	0..*
Generalization		Element	

5.3 Meta-Package: datamodel.conceptual

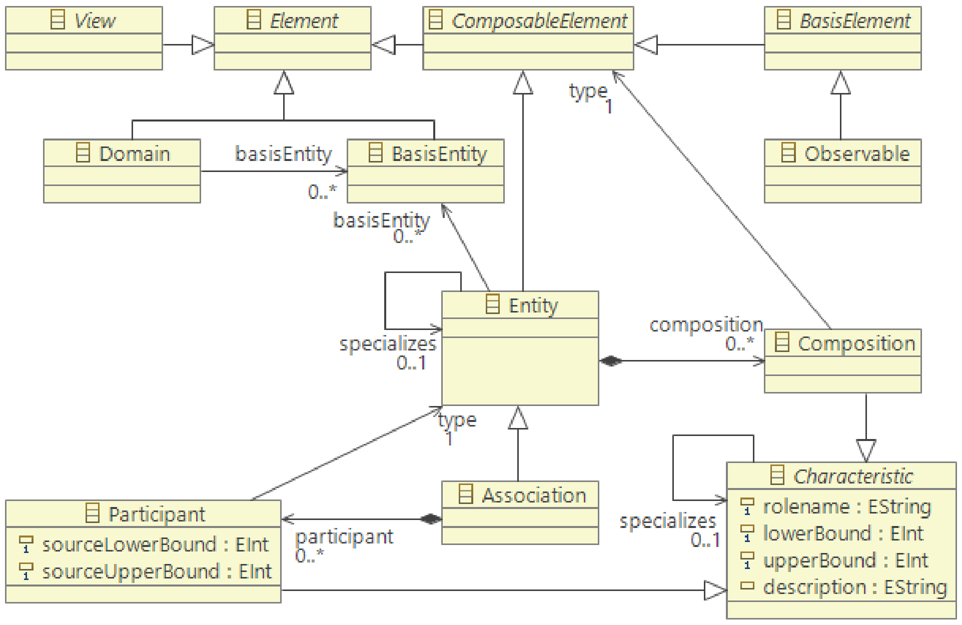

Figure 4: Data Model Metamodel "datamodel.conceptual" Package

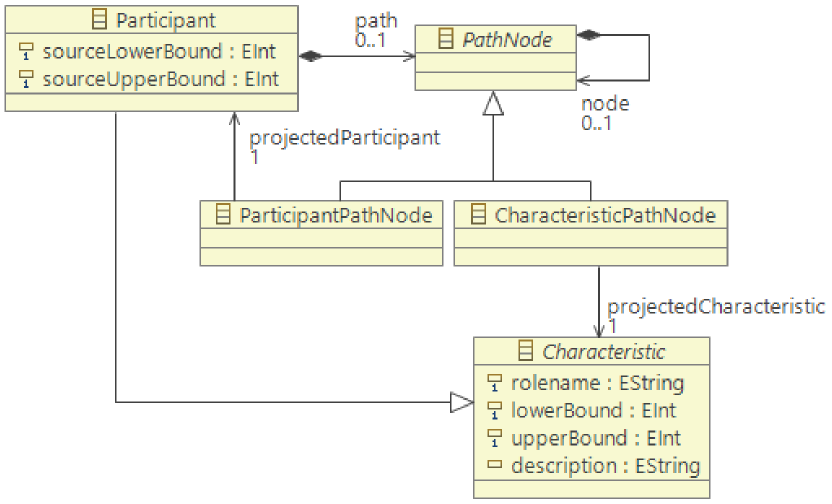

Figure 5: Data Model Metamodel "datamodel.conceptual" Package: Participant Path

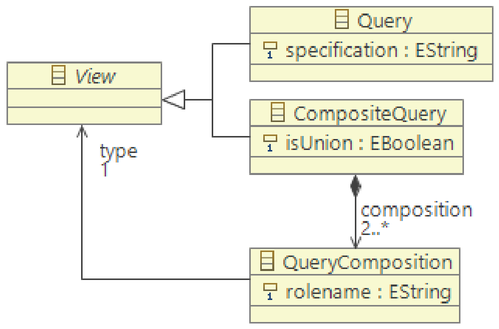

Figure 6: Data Model Metamodel "datamodel.conceptual" Package: Views

5.3.1 Meta-Class: datamodel.conceptual.Element

Description

A conceptual Element is the root type for defining the conceptual elements of a Data Model. The relationships for the Element meta-class are listed in Table 6.

Table 6: datamodel.conceptual.Element Relationships

Type	Name	Target	Multiplicity
Generalization		datamodel.Element	

5.3.2 Meta-Class: datamodel.conceptual.ComposableElement

Description

A conceptual ComposableElement is a conceptual Element that is allowed to participate in a Composition relationship. In other words, these are the conceptual Elements that may be a characteristic of a conceptual Entity. The relationships for the ComposableElement meta-class are listed in Table 7.

Table 7: datamodel.conceptual.ComposableElement Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.3.3 Meta-Class: datamodel.conceptual.BasisElement

Description

A conceptual BasisElement is a conceptual data type that is independent of any specific data representation. The relationships for the BasisElement meta-class are listed in Table 8.

Table 8: datamodel.conceptual.BasisElement Relationships

Type	Name	Target	Multiplicity
Generalization		ComposableElement	

5.3.4 Meta-Class: datamodel.conceptual.BasisEntity

Description

A BasisEntity represents a unique domain concept and establishes a basis which can be used by conceptual Entities. The relationships for the BasisEntity meta-class are listed in Table 9.

Table 9: datamodel.conceptual.BasisEntity Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.3.5 Meta-Class: datamodel.conceptual.Domain

Description

A Domain represents a space defined by a set of BasisEntities relating to well understood concepts by practitioners within a particular domain. The relationships for the Domain meta-class are listed in Table 10.

Table 10: datamodel.conceptual.Domain Relationships

Type	Name	Target	Multiplicity
Association	basisEntity	BasisEntity	0..*
Generalization		Element	

5.3.6 Meta-Class: datamodel.conceptual.Observable

Description

An Observable is something that can be observed but not further characterized, and is typically quantified through measurements of the physical world. An observable is independent of any specific data representation, units, or reference frame. For example, "length" may be thought of as an observable in that it can be measured, but at the conceptual level the nature of the measurement is not specified. The relationships for the Observable meta-class are listed in Table 11.

Table 11: datamodel.conceptual.Observable Relationships

Type	Name	Target	Multiplicity
Generalization		BasisElement	

5.3.7 Meta-Class: datamodel.conceptual.Characteristic

Description

A conceptual Characteristic contributes to the uniqueness of a conceptual Entity. The “rolename” attribute defines the name of the conceptual Characteristic within the scope of the conceptual Entity. The “lowerBound” and “upperBound” attributes define the multiplicity of the composed Characteristic. An “upperBound” multiplicity of –1 represents an unbounded sequence. The attributes for the Characteristic meta-class are listed in Table 12, and its relationships are shown in Table 13.

Table 12: datamodel.conceptual.Characteristic Attributes

Name	Type	Multiplicity
rolename	string	1
lowerBound	int	1
upperBound	int	1
description	string	0..1

Table 13: datamodel.conceptual.Characteristic Relationships

Type	Name	Target	Multiplicity
Association	specializes	Characteristic	0..1

5.3.8 Meta-Class: datamodel.conceptual.Entity

Description

A conceptual Entity represents a domain concept in terms of its Observables and other composed conceptual Entities. Since a conceptual Entity is built only from conceptual ComposableElements, it is independent of any specific data representation, units, or reference frame. The relationships for the Entity meta-class are listed in Table 14.

Table 14: datamodel.conceptual.Entity Relationships

Type	Name	Target	Multiplicity
Composition	composition	Composition	0..*
Association	specializes	Entity	0..1
Association	basisEntity	BasisEntity	0..*
Generalization		ComposableElement	

5.3.9 Meta-Class: datamodel.conceptual.Composition

Description

A conceptual Composition is the mechanism that allows conceptual Entities to be constructed from other conceptual ComposableElements. The “type” of a Composition is the ComposableElement being used to construct the conceptual Entity. The relationships for the Composition meta-class are listed in Table 15.

Table 15: datamodel.conceptual.Composition Relationships

Type	Name	Target	Multiplicity
Association	type	ComposableElement	1
Generalization		Characteristic	

5.3.10 Meta-Class: datamodel.conceptual.Association

Description

A conceptual Association represents a relationship between two or more conceptual Entities. The conceptual Entities participating in the conceptual Association may be specified locally or in its generalized types. In addition, there may be one or more conceptual ComposableElements that characterize the relationship. Conceptual Associations are conceptual Entities that may also participate in other conceptual Associations. The relationships for the Association meta-class are listed in Table 16.

Table 16: datamodel.conceptual.Association Relationships

Type	Name	Target	Multiplicity
Composition	participant	Participant	0..*
Generalization		Entity	

5.3.11 Meta-Class: datamodel.conceptual.Participant

Description

A conceptual Participant is the mechanism that allows a conceptual Association to be constructed between two or more conceptual Entities. The “type” of a conceptual Participant is the conceptual Entity being used to construct the conceptual Association. The “sourceLowerBound” and “sourceUpperBound” attributes define the multiplicity of the conceptual Association relative to the Participant. A “sourceUpperBound” multiplicity of –1 represents an unbounded sequence. The “path” attribute of the Participant describes the chain of entity characteristics to traverse to reach the subject of the association beginning with the entity referenced by the “type” attribute. The attributes for the Participant meta-class are listed in Table 17, and its relationships are shown in Table 18.

Table 17: datamodel.conceptual.Participant Attributes

Name	Type	Multiplicity
sourceLowerBound	int	1
sourceUpperBound	int	1

Table 18: datamodel.conceptual.Participant Relationships

Type	Name	Target	Multiplicity
Association	type	Entity	1
Composition	path	PathNode	0..1
Generalization		Characteristic	

5.3.12 Meta-Class: datamodel.conceptual.PathNode

Description

A conceptual PathNode is a single element in a chain that collectively forms a path specification. The relationships for the PathNode meta-class are listed in Table 19.

Table 19: datamodel.conceptual.PathNode Relationships

Type	Name	Target	Multiplicity
Composition	node	PathNode	0..1

5.3.13 Meta-Class: datamodel.conceptual.ParticipantPathNode

Description

A conceptual ParticipantPathNode is a conceptual PathNode that selects a Participant that references an Entity. This provides a mechanism for reverse navigation from an Entity that participates in an Association back to the Association. The relationships for the ParticipantPathNode meta-class are listed in Table 20.

Table 20: datamodel.conceptual.ParticipantPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedParticipant	Participant	1
Generalization		PathNode	

5.3.14 Meta-Class: datamodel.conceptual.CharacteristicPathNode

Description

A conceptual CharacteristicPathNode is a conceptual PathNode that selects a conceptual Characteristic which is directly contained in a conceptual Entity or Association. The relationships for the CharacteristicPathNode meta-class are listed in Table 21.

Table 21: datamodel.conceptual.CharacteristicPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedCharacteristic	Characteristic	1
Generalization		PathNode	

5.3.15 Meta-Class: datamodel.conceptual.View

Description

A conceptual View is a conceptual Query or a conceptual CompositeQuery. The relationships for the View meta-class are listed in Table 22.

Table 22: datamodel.conceptual.View Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.3.16 Meta-Class: datamodel.conceptual.Query

Description

A conceptual Query is a specification that defines the content of conceptual View as a set of conceptual Characteristics projected from a selected set of related conceptual Entities. The “specification” attribute captures the specification of a Query as defined by the Query grammar in Section 6.1. The attributes for the Query meta-class are listed in Table 23, and its relationships are shown in Table 24.

Table 23: datamodel.conceptual.Query Attributes

Name	Type	Multiplicity
specification	string	1

Table 24: datamodel.conceptual.Query Relationships

Type	Name	Target	Multiplicity
Generalization		View	

5.3.17 Meta-Class: datamodel.conceptual.CompositeQuery

Description

A conceptual CompositeQuery is a collection of two or more conceptual Views. The “isUnion” attribute specifies whether the composed Views are intended to be mutually exclusive or not. The attributes for the CompositeQuery meta-class are listed in Table 25, and its relationships are shown in Table 26.

Table 25: datamodel.conceptual.CompositeQuery Attributes

Name	Type	Multiplicity
isUnion	boolean	1

Table 26: datamodel.conceptual.CompositeQuery Relationships

Type	Name	Target	Multiplicity
Composition	composition	QueryComposition	2..*
Generalization		Element	
Generalization		View	

5.3.18 Meta-Class: datamodel.conceptual.QueryComposition

Description

A conceptual QueryComposition is the mechanism that allows a conceptual CompositeQuery to be constructed from conceptual Queries and other conceptual CompositeQueries. The “rolename” attribute defines the name of the composed conceptual View within the scope of the composing conceptual CompositeQuery. The “type” of a conceptual QueryComposition is the conceptual View being used to construct the conceptual CompositeQuery. The attributes for the QueryComposition meta-class are listed in Table 27, and its relationships are shown in Table 28.

Table 27: datamodel.conceptual.QueryComposition Attributes

Name	Type	Multiplicity
rolename	string	1

Table 28: datamodel.conceptual.QueryComposition Relationships

Type	Name	Target	Multiplicity
Association	type	View	1

5.4 Meta-Package: datamodel.logical

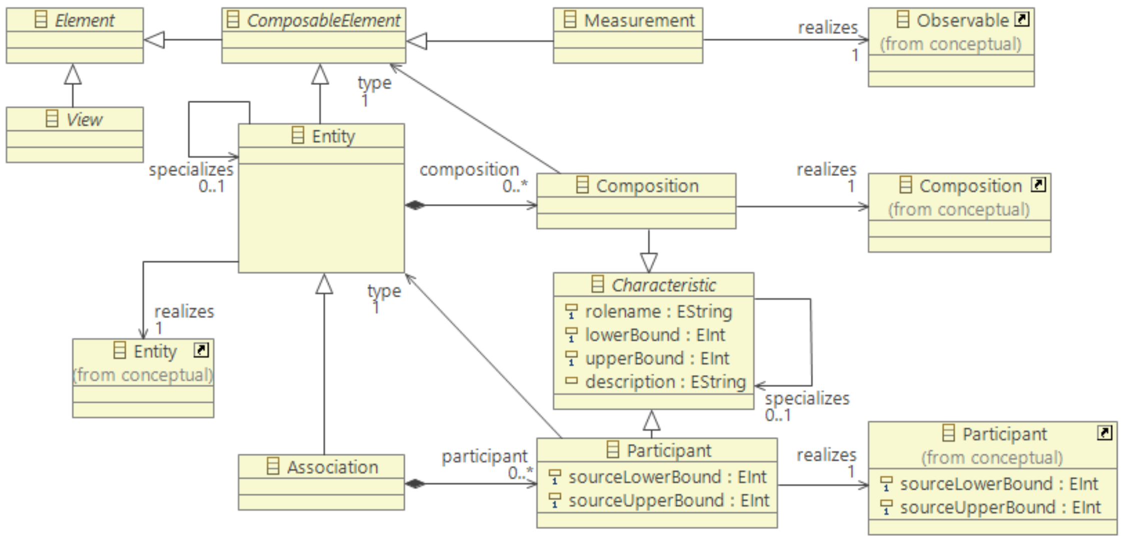

Figure 7: Data Model Metamodel "datamodel.logical" Package

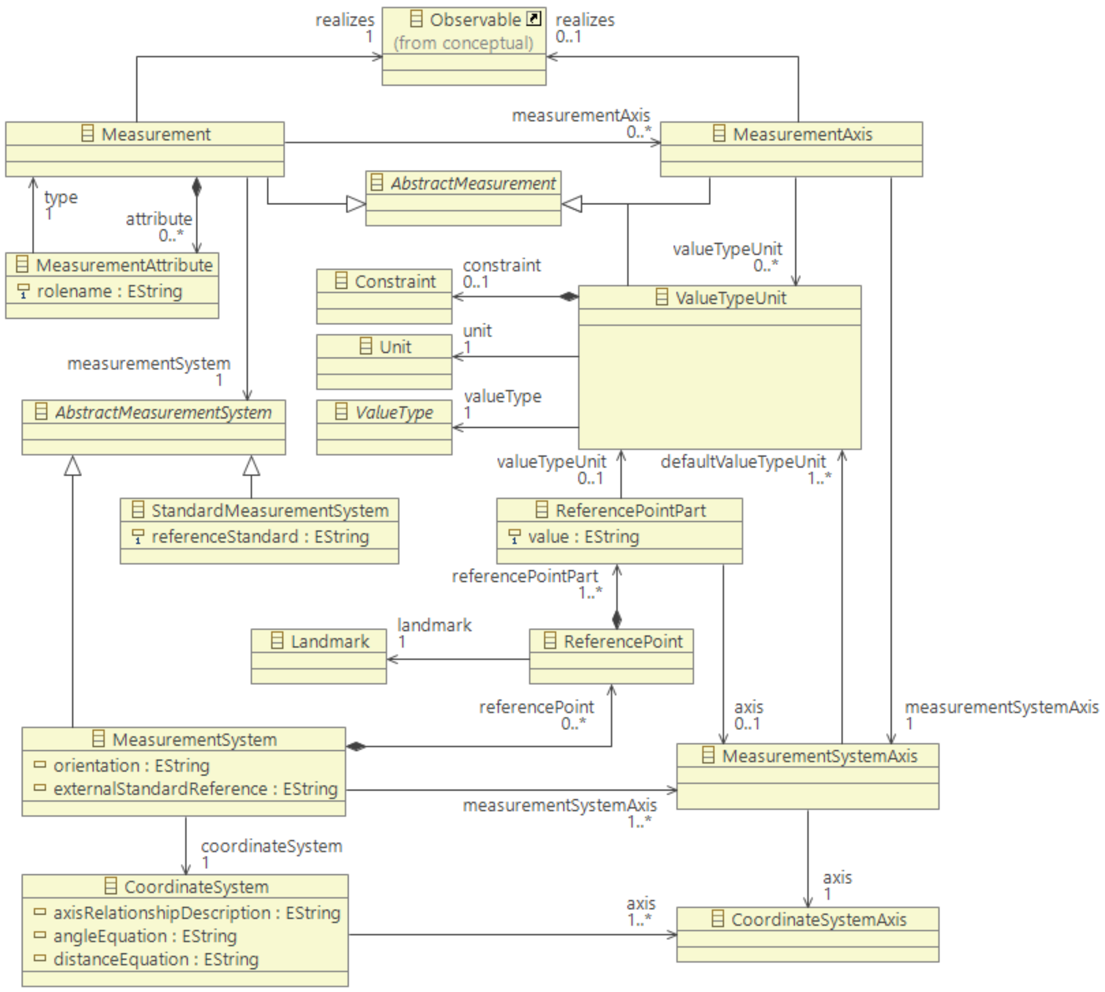

Figure 8: Data Model Metamodel "datamodel.logical" Package: Logical Basis

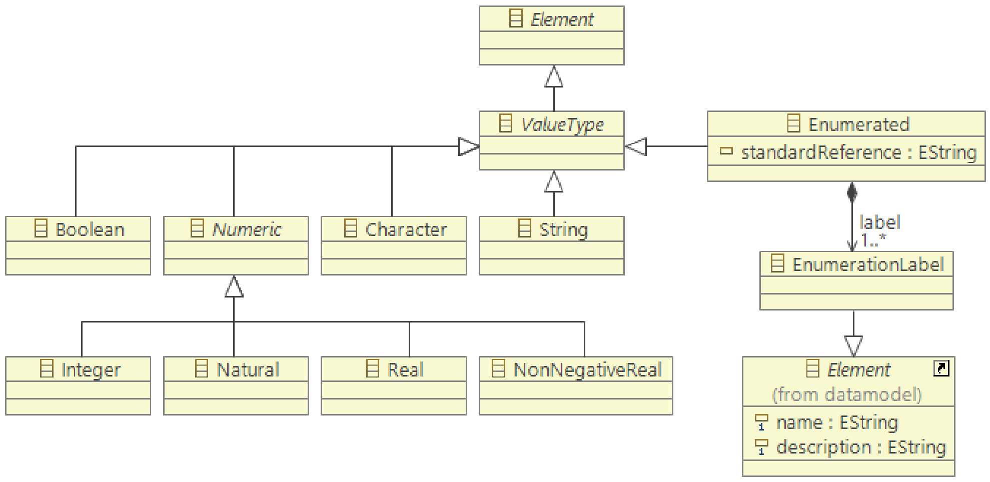

Figure 9: Data Model Metamodel "datamodel.logical" Package: Logical Value Types

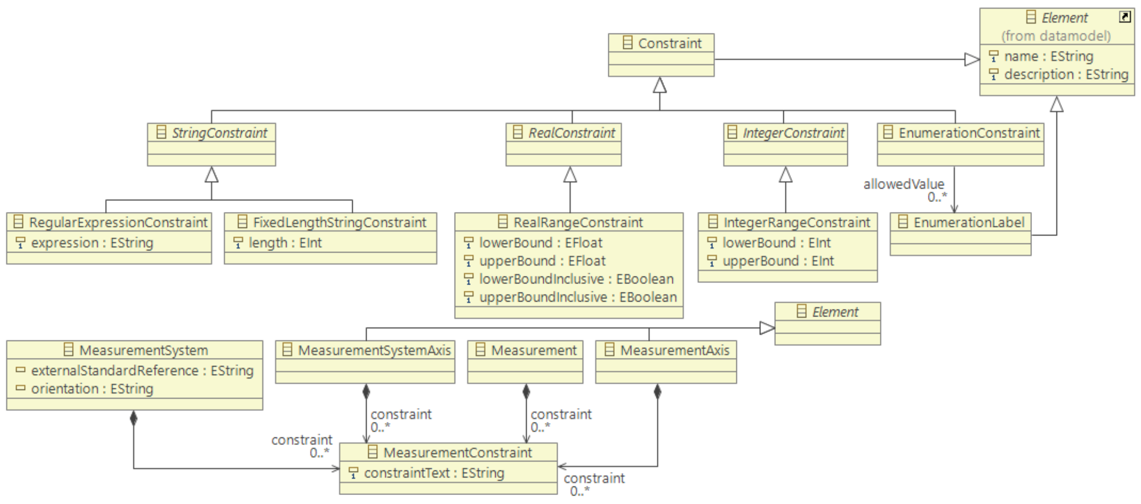

Figure 10: Data Model Metamodel "datamodel.logical" Package: Measurement Constraints

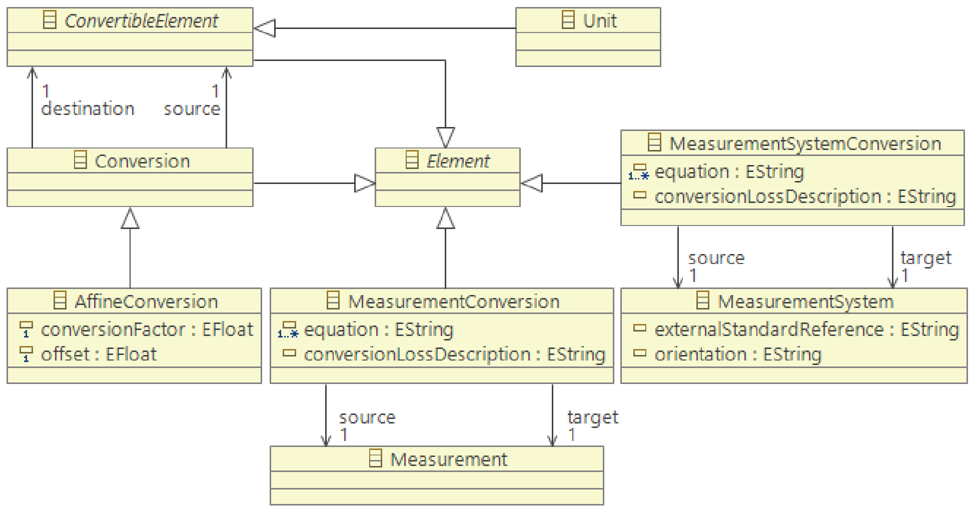

Figure 11: Data Model Metamodel "datamodel.logical" Package: Measurement Conversion

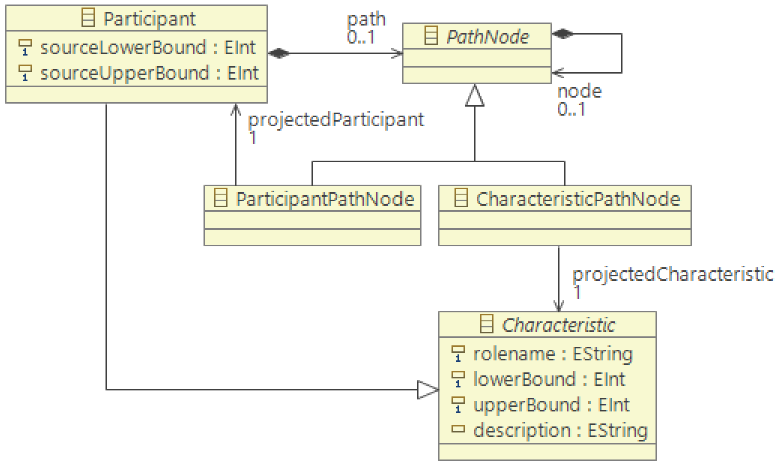

Figure 12: Data Model Metamodel "datamodel.logical" Package: Participant Path

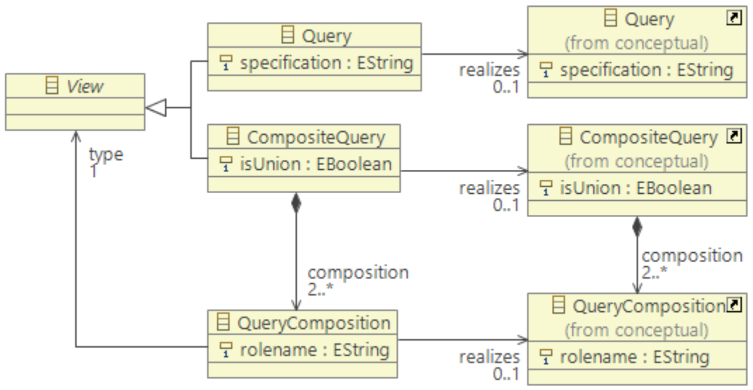

Figure 13: Data Model Metamodel "datamodel.logical" Package: Views

5.4.1 Meta-Class: datamodel.logical.Element

Description

A logical Element is the root type for defining the logical elements of a Data Model. The relationships for the Element meta-class are listed in Table 29.

Table 29: datamodel.logical.Element Relationships

Type	Name	Target	Multiplicity
Generalization		datamodel.Element	

5.4.2 Meta-Class: datamodel.logical.ConvertibleElement

Description

A ConvertibleElement is a Unit. The relationships for the ConvertibleElement meta-class are listed in Table 30.

Table 30: datamodel.logical.ConvertibleElement Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.3 Meta-Class: datamodel.logical.Unit

Description

A Unit is a defined magnitude of quantity used as a standard for measurement. The relationships for the Unit meta-class are listed in Table 31.

Table 31: datamodel.logical.Unit Relationships

Type	Name	Target	Multiplicity
Generalization		ConvertibleElement	

5.4.4 Meta-Class: datamodel.logical.Conversion

Description

A Conversion is a relationship between two ConvertibleElements that describes how to transform measured quantities between two Units. The relationships for the Conversion meta-class are listed in Table 32.

Table 32: datamodel.logical.Conversion Relationships

Type	Name	Target	Multiplicity
Association	destination	ConvertibleElement	1
Association	source	ConvertibleElement	1
Generalization		Element	

5.4.5 Meta-Class: datamodel.logical.AffineConversion

Description

An AffineConversion is a relationship between two ConvertibleElements in the form *mx+b*. The attributes for the AffineConversion meta-class are listed in Table 33, and its relationships are shown in Table 34.

Table 33: datamodel.logical.AffineConversion Attributes

Name	Type	Multiplicity
conversionFactor	float	1
offset	float	1

Table 34: datamodel.logical.AffineConversion Relationships

Type	Name	Target	Multiplicity
Generalization		Conversion	

5.4.6 Meta-Class: datamodel.logical.ValueType

Description

A ValueType specifies the logical representation of a MeasurementSystem or Measurement. Integer, Real, and String are examples of logical ValueTypes. The relationships for the ValueType meta-class are listed in Table 35.

Table 35: datamodel.logical.ValueType Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.7 Meta-Class: datamodel.logical.String

Description

A String is a value type that represents a variable length sequence of characters. The relationships for the String meta-class are listed in Table 36.

Table 36: datamodel.logical.String Relationships

Type	Name	Target	Multiplicity
Generalization		ValueType	

5.4.8 Meta-Class: datamodel.logical.Character

Description

A Character is a value type representing characters from any character set. The relationships for the Character meta-class are listed in Table 37.

Table 37: datamodel.logical.Character Relationships

Type	Name	Target	Multiplicity
Generalization		ValueType	

5.4.9 Meta-Class: datamodel.logical.Boolean

Description

A Boolean is a value type representing the two values TRUE and FALSE. The relationships for the Boolean meta-class are listed in Table 38.

Table 38: datamodel.logical.Boolean Relationships

Type	Name	Target	Multiplicity
Generalization		ValueType	

5.4.10 Meta-Class: datamodel.logical.Numeric

Description

A Numeric is a numeric ValueType. The relationships for the Numeric meta-class are listed in Table 39.

Table 39: datamodel.logical.Numeric Relationships

Type	Name	Target	Multiplicity
Generalization		ValueType	

5.4.11 Meta-Class: datamodel.logical.Integer

Description

An Integer is a value type representing integer numbers. The relationships for the Integer meta-class are listed in Table 40.

Table 40: datamodel.logical.Integer Relationships

Type	Name	Target	Multiplicity
Generalization		Numeric	

5.4.12 Meta-Class: datamodel.logical.Natural

Description

A Natural is a value type representing the non-negative integers. The relationships for the Natural meta-class are listed in Table 41.

Table 41: datamodel.logical.Natural Relationships

Type	Name	Target	Multiplicity
Generalization		Numeric	

5.4.13 Meta-Class: datamodel.logical.Real

Description

A Real is a value type representing real numbers. The relationships for the Real meta-class are listed in Table 42.

Table 42: datamodel.logical.Real Relationships

Type	Name	Target	Multiplicity
Generalization		Numeric	

5.4.14 Meta-Class: datamodel.logical.NonNegativeReal

Description

A NonNegativeReal is a value type representing non-negative real numbers. The relationships for the NonNegativeReal meta-class are listed in Table 43.

Table 43: datamodel.logical.NonNegativeReal Relationships

Type	Name	Target	Multiplicity
Generalization		Numeric	

5.4.15 Meta-Class: datamodel.logical.Enumerated

Description

An Enumerated is a value type representing a set of named values, each with specific meaning. The attributes for the Enumerated meta-class are listed in Table 44, and its relationships are shown in Table 45.

Table 44: datamodel.logical.Enumerated Attributes

Name	Type	Multiplicity
standardReference	string	0..1

Table 45: datamodel.logical.Enumerated Relationships

Type	Name	Target	Multiplicity
Composition	label	EnumerationLabel	1..* {Ordered}

Type	Name	Target	Multiplicity
Generalization		ValueType	

5.4.16 Meta-Class: datamodel.logical.EnumerationLabel

Description

An EnumerationLabel defines a named member of an Enumerated value set. The relationships for the EnumerationLabel meta-class are listed in Table 46.

Table 46: datamodel.logical.EnumerationLabel Relationships

Type	Name	Target	Multiplicity
Generalization		datamodel.Element	

5.4.17 Meta-Class: datamodel.logical.CoordinateSystem

Description

A CoordinateSystem is a system which uses one or more coordinates to uniquely determine the position of a point in an *N*-dimensional space. The coordinate system is comprised of multiple CoordinateSystemAxis which completely span the space. Coordinates are quantified relative to the CoordinateSystemAxis. It is not required that the dimensions be ordered or continuous. The attributes for the CoordinateSystem meta-class are listed in Table 47, and its relationships are shown in Table 48.

Table 47: datamodel.logical.CoordinateSystem Attributes

Name	Type	Multiplicity
axisRelationshipDescription	string	0..1
angleEquation	string	0..1
distanceEquation	string	0..1

Table 48: datamodel.logical.CoordinateSystem Relationships

Type	Name	Target	Multiplicity
Association	axis	CoordinateSystemAxis	1..*
Generalization		Element	

5.4.18 Meta-Class: datamodel.logical.CoordinateSystemAxis

Description

A CoordinateSystemAxis represents a dimension within a CoordinateSystem. The relationships for the CoordinateSystemAxis meta-class are listed in Table 49.

Table 49: datamodel.logical.CoordinateSystemAxis Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.19 Meta-Class: datamodel.logical.AbstractMeasurementSystem

Description

An AbstractMeasurementSystem is an abstract parent for StandardMeasurementSystems and MeasurementSystems. It is used for structural simplicity in the metamodel. The relationships for the AbstractMeasurementSystem meta-class are listed in Table 50.

Table 50: datamodel.logical.AbstractMeasurementSystem Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.20 Meta-Class: datamodel.logical.StandardMeasurementSystem

Description

A StandardMeasurementSystem is used to represent an open, referenced measurement system without requiring the detailed modeling of the measurement system. The reference should be unambiguous and allows for full comprehension of the underlying measurement system. The attributes for the StandardMeasurementSystem meta-class are listed in Table 51, and its relationships are shown in Table 52.

Table 51: datamodel.logical.StandardMeasurementSystem Attributes

Name	Type	Multiplicity
referenceStandard	string	1

Table 52: datamodel.logical.StandardMeasurementSystem Relationships

Type	Name	Target	Multiplicity
Generalization		AbstractMeasurementSystem	

5.4.21 Meta-Class: datamodel.logical.Landmark

Description

A Landmark is a named, recognizable or artificial feature used to locate a ReferencePoint in a measurable space. The relationships for the Landmark meta-class are listed in Table 53.

Table 53: datamodel.logical.Landmark Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.22 Meta-Class: datamodel.logical.MeasurementSystem

Description

A MeasurementSystem relates a CoordinateSystem to an origin and orientation for the purpose of establishing a common basis for describing points in an *N*-dimensional space. Defining a MeasurementSystem establishes additional properties of the CoordinateSystem including units and value types for each axis, and a set of reference points that can be used to establish an origin and indicate the direction of each axis. The attributes for the MeasurementSystem meta-class are listed in Table 54, and its relationships are shown in Table 55.

Table 54: datamodel.logical.MeasurementSystem Attributes

Name	Type	Multiplicity
externalStandardReference	string	0..1
orientation	string	0..1

Table 55: datamodel.logical.MeasurementSystem Relationships

Type	Name	Target	Multiplicity
Association	measurementSystemAxis	MeasurementSystemAxis	1..*
Association	coordinateSystem	CoordinateSystem	1
Composition	referencePoint	ReferencePoint	0..*
Composition	constraint	MeasurementConstraint	0..* {Ordered}
Generalization		AbstractMeasurementSystem	

5.4.23 Meta-Class: datamodel.logical.MeasurementSystemAxis

Description

A MeasurementSystemAxis establishes additional properties for a CoordinateSystemAxis including units and value types. The relationships for the MeasurementSystemAxis meta-class are listed in Table 56.

Table 56: datamodel.logical.MeasurementSystemAxis Relationships

Type	Name	Target	Multiplicity
Association	axis	CoordinateSystemAxis	1
Association	defaultValueTypeUnit	ValueTypeUnit	1..*

Type	Name	Target	Multiplicity
Composition	constraint	MeasurementConstraint	0..* {Ordered}
Generalization		Element	

5.4.24 Meta-Class: datamodel.logical.ReferencePoint

Description

A ReferencePoint is an identifiable point (landmark) that can be used to provide a basis for locating and/or orienting a MeasurementSystem. The relationships for the ReferencePoint meta-class are listed in Table 57.

Table 57: datamodel.logical.ReferencePoint Relationships

Type	Name	Target	Multiplicity
Composition	referencePointPart	ReferencePointPart	1..*
Association	landmark	Landmark	1
Generalization		datamodel.Element	

5.4.25 Meta-Class: datamodel.logical.ReferencePointPart

Description

A ReferencePointPart is a value for one ValueTypeUnit in a ValueTypeUnit set that is used to identify a specific point along an axis. The attributes for the ReferencePointPart meta-class are listed in Table 58, and its relationships are shown in Table 59.

Table 58: datamodel.logical.ReferencePointPart Attributes

Name	Type	Multiplicity
value	string	1

Table 59: datamodel.logical.ReferencePointPart Relationships

Type	Name	Target	Multiplicity
Association	axis	MeasurementSystemAxis	0..1
Association	valueTypeUnit	ValueTypeUnit	0..1

5.4.26 Meta-Class: datamodel.logical.ValueTypeUnit

Description

A ValueTypeUnit defines the logical representation of a MeasurementSystemAxis or MeasurementAxis value type in terms of a Unit and ValueType pair. The relationships for the ValueTypeUnit meta-class are listed in Table 60.

Table 60: datamodel.logical.ValueTypeUnit Relationships

Type	Name	Target	Multiplicity
Association	unit	Unit	1
Association	valueType	ValueType	1
Composition	constraint	Constraint	0..1
Generalization		Element	
Generalization		AbstractMeasurement	

5.4.27 Meta-Class: datamodel.logical.Constraint

Description

A Constraint limits the set of possible values for the ValueType of a MeasurementSystem or Measurement. The relationships for the Constraint meta-class are listed in Table 61.

Table 61: datamodel.logical.Constraint Relationships

Type	Name	Target	Multiplicity
Generalization		datamodel.Element	

5.4.28 Meta-Class: datamodel.logical.IntegerConstraint

Description

An IntegerConstraint specifies a defined set of meaningful values for an Integer or Natural. The relationships for the IntegerConstraint meta-class are listed in Table 62.

Table 62: datamodel.logical.IntegerConstraint Relationships

Type	Name	Target	Multiplicity
Generalization		Constraint	

5.4.29 Meta-Class: datamodel.logical.IntegerRangeConstraint

Description

An IntegerRangeConstraint specifies a defined range of meaningful values for an Integer or Natural. The “upperBound” is greater than or equal to the “lowerBound”. The defined range is inclusive of the “upperBound” and “lowerBound”. The attributes for the IntegerRangeConstraint meta-class are listed in Table 63, and its relationships are shown in Table 64.

Table 63: datamodel.logical.IntegerRangeConstraint Attributes

Name	Type	Multiplicity
lowerBound	int	1
upperBound	int	1

Table 64: datamodel.logical.IntegerRangeConstraint Relationships

Type	Name	Target	Multiplicity
Generalization		IntegerConstraint	

5.4.30 Meta-Class: datamodel.logical.RealConstraint

Description

A RealConstraint specifies a defined set of meaningful values for a Real or NonNegativeReal. The relationships for the RealConstraint meta-class are listed in Table 65.

Table 65: datamodel.logical.RealConstraint Relationships

Type	Name	Target	Multiplicity
Generalization		Constraint	

5.4.31 Meta-Class: datamodel.logical.RealRangeConstraint

Description

A RealRangeConstraint specifies a defined range of meaningful values for a Real or NonNegativeReal. The “upperBound” is greater than or equal to the “lowerBound”. The attributes for the RealRangeConstraint meta-class are listed in Table 66, and its relationships are shown in Table 67.

Table 66: datamodel.logical.RealRangeConstraint Attributes

Name	Type	Multiplicity
lowerBound	float	1
upperBound	float	1
lowerBoundInclusive	boolean	1
upperBoundInclusive	boolean	1

Table 67: datamodel.logical.RealRangeConstraint Relationships

Type	Name	Target	Multiplicity
Generalization		RealConstraint	

5.4.32 Meta-Class: datamodel.logical.StringConstraint

Description

A StringConstraint specifies a defined set of meaningful values for a String. The relationships for the StringConstraint meta-class are listed in Table 68.

Table 68: datamodel.logical.StringConstraint Relationships

Type	**Name**	**Target**	**Multiplicity**
Generalization		Constraint	

5.4.33 Meta-Class: datamodel.logical.RegularExpressionConstraint

Description

A RegularExpressionConstraint specifies a defined set of meaningful values for a String in the form of a regular expression. The attributes for the RegularExpressionConstraint meta-class are listed in Table 69, and its relationships are shown in Table 70.

Table 69: datamodel.logical.RegularExpressionConstraint Attributes

Name	**Type**	**Multiplicity**
expression	string	1

Table 70: datamodel.logical.RegularExpressionConstraint Relationships

Type	**Name**	**Target**	**Multiplicity**
Generalization		StringConstraint	

5.4.34 Meta-Class: datamodel.logical.FixedLengthStringConstraint

Description

A FixedLengthStringConstraint specifies a defined set of meaningful values for a String of a specific fixed length. The “length” attribute defines the fixed length, an integer value greater than 0. The attributes for the FixedLengthStringConstraint meta-class are listed in Table 71, and its relationships are shown in Table 72.

Table 71: datamodel.logical.FixedLengthStringConstraint Attributes

Name	**Type**	**Multiplicity**
length	int	1

Table 72: datamodel.logical.FixedLengthStringConstraint Relationships

Type	Name	Target	Multiplicity
Generalization		StringConstraint	

5.4.35 Meta-Class: datamodel.logical.EnumerationConstraint

Description

An EnumerationConstraint identifies a subset of enumerated values (EnumerationLabel) considered valid for an Enumerated value type of a MeasurementAxis. The relationships for the EnumerationConstraint meta-class are listed in Table 73.

Table 73: datamodel.logical.EnumerationConstraint Relationships

Type	Name	Target	Multiplicity
Association	allowedValue	EnumerationLabel	0..*
Generalization		Constraint	

5.4.36 Meta-Class: datamodel.logical.MeasurementConstraint

Description

A MeasurementConstraint describes the constraints over the axes of a given MeasurementSystem or Measurement or over the value types of a MeasurementSystemAxis or MeasurementAxis. The constraints are described in the “constraintText” attribute. The specific format of “constraintText” is undefined. The attributes for the MeasurementConstraint meta-class are listed in Table 74.

Table 74: datamodel.logical.MeasurementConstraint Attributes

Name	Type	Multiplicity
constraintText	string	1

5.4.37 Meta-Class: datamodel.logical.MeasurementSystemConversion

Description

A MeasurementSystemConversion is a relationship between two MeasurementSystems that describes how to transform measured quantities between those MeasurementSystems. The conversion is captured as a set of equations in the “equation” attribute. The specific format of “equation” is undefined. The loss introduced by the conversion equations is captured in the “conversionLossDescription” attribute. The specific format of “conversionLossDescription” is undefined. The attributes for the MeasurementSystemConversion meta-class are listed in Table 75, and its relationships are shown in Table 76.

Table 75: datamodel.logical.MeasurementSystemConversion Attributes

Name	Type	Multiplicity
equation	string	1..* {Ordered}
conversionLossDescription	string	0..1

Table 76: datamodel.logical.MeasurementSystemConversion Relationships

Type	Name	Target	Multiplicity
Association	source	MeasurementSystem	1
Association	target	MeasurementSystem	1
Generalization		Element	

5.4.38 Meta-Class: datamodel.logical.AbstractMeasurement

Description

An AbstractMeasurement is a Measurement, MeasurementAxis, or a ValueTypeUnit.

5.4.39 Meta-Class: datamodel.logical.Measurement

Description

A Measurement realizes an Observable as a set of quantities that can be recorded for each axis of a MeasurementSystem. A Measurement contains the specific implementation details optionally including an override of the default ValueType and Unit for each axis as well as the constraints over that space for which the MeasurementSystem is valid. The relationships for the Measurement meta-class are listed in Table 77.

Table 77: datamodel.logical.Measurement Relationships

Type	Name	Target	Multiplicity
Composition	constraint	MeasurementConstraint	0..* {Ordered}
Association	measurementAxis	MeasurementAxis	0..*
Association	measurementSystem	AbstractMeasurementSystem	1
Association	realizes	datamodel.conceptual.Observable	1
Composition	attribute	MeasurementAttribute	0..*
Generalization		ComposableElement	
Generalization		AbstractMeasurement	

5.4.40 Meta-Class: datamodel.logical.MeasurementAxis

Description

A MeasurementAxis optionally establishes constraints for a MeasurementSystemAxis and may optionally override its default units and value types. The relationships for the MeasurementAxis meta-class are listed in Table 78.

Table 78: datamodel.logical.MeasurementAxis Relationships

Type	Name	Target	Multiplicity
Association	valueTypeUnit	ValueTypeUnit	0..*
Association	measurementSystemAxis	MeasurementSystemAxis	1
Composition	constraint	MeasurementConstraint	0..* {Ordered}
Association	realizes	datamodel.conceptual.Observable	0..1
Generalization		Element	
Generalization		AbstractMeasurement	

5.4.41 Meta-Class: datamodel.logical.MeasurementAttribute

Description

A MeasurementAttribute is supplemental data associated with a Measurement. The attributes for the MeasurementAttribute meta-class are listed in Table 79, and its relationships are shown in Table 80.

Table 79: datamodel.logical.MeasurementAttribute Attributes

Name	Type	Multiplicity
rolename	string	1

Table 80: datamodel.logical.MeasurementAttribute Relationships

Type	Name	Target	Multiplicity
Association	type	Measurement	1

5.4.42 Meta-Class: datamodel.logical.MeasurementConversion

Description

A MeasurementConversion is a relationship between two Measurements that describes how to transform measured quantities between those Measurements. The conversion is captured as a set of equations in the “equation” attribute. The specific format of “equation” is undefined. The loss introduced by the conversion equations is captured in the “conversionLossDescription” attribute. The specific format of “conversionLossDescription” is undefined. The attributes for the

MeasurementConversion meta-class are listed in Table 81, and its relationships are shown in Table 82.

Table 81: datamodel.logical.MeasurementConversion Attributes

Name	Type	Multiplicity
equation	string	1..* {Ordered}
conversionLossDescription	string	0..1

Table 82: datamodel.logical.MeasurementConversion Relationships

Type	Name	Target	Multiplicity
Association	source	Measurement	1
Association	target	Measurement	1
Generalization		Element	

5.4.43 Meta-Class: datamodel.logical.ComposableElement

Description

A logical ComposableElement is a logical Element that is allowed to participate in a Composition relationship. In other words, these are the logical Elements that may be a characteristic of a logical Entity. The relationships for the ComposableElement meta-class are listed in Table 83.

Table 83: datamodel.logical.ComposableElement Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.44 Meta-Class: datamodel.logical.Characteristic

Description

A logical Characteristic contributes to the uniqueness of a logical Entity. The “rolename” attribute defines the name of the logical Characteristic within the scope of the logical Entity. The “lowerBound” and “upperBound” attributes define the multiplicity of the composed Characteristic. An “upperBound” multiplicity of –1 represents an unbounded sequence. The attributes for the Characteristic meta-class are listed in Table 84, and its relationships are shown in Table 85.

Table 84: datamodel.logical.Characteristic Attributes

Name	Type	Multiplicity
rolename	string	1
lowerBound	int	1

Name	Type	Multiplicity
upperBound	int	1
description	string	0..1

Table 85: datamodel.logical.Characteristic Relationships

Type	Name	Target	Multiplicity
Association	specializes	Characteristic	0..1

5.4.45 Meta-Class: datamodel.logical.Entity

Description

A logical Entity "realizes" a conceptual Entity in terms of Measurements and other logical Entities. Since a logical Entity is built from logical Measurements, it is independent of any specific platform data representation. A logical Entity's composition hierarchy is consistent with the composition hierarchy of the conceptual Entity that it realizes. The logical Entity's composed Entities realize one to one the conceptual Entity's composed Entities; the logical Entity's composed Measurements realize many to one the conceptual Entity's composed Observables. The relationships for the Entity meta-class are listed in Table 86.

Table 86: datamodel.logical.Entity Relationships

Type	Name	Target	Multiplicity
Composition	composition	Composition	0..*
Association	realizes	datamodel.conceptual.Entity	1
Association	specializes	Entity	0..1
Generalization		ComposableElement	

5.4.46 Meta-Class: datamodel.logical.Composition

Description

A logical Composition is the mechanism that allows logical Entities to be constructed from other logical ComposableElements. The "type" of a Composition is the ComposableElement being used to construct the logical Entity. The "lowerBound" and "upperBound" define the multiplicity of the composed logical Entity. An "upperBound" multiplicity of –1 represents an unbounded sequence. The relationships for the Composition meta-class are listed in Table 87.

Table 87: datamodel.logical.Composition Relationships

Type	Name	Target	Multiplicity
Association	type	ComposableElement	1
Association	realizes	datamodel.conceptual.Composition	1

Type	Name	Target	Multiplicity
Generalization		Characteristic	

5.4.47 Meta-Class: datamodel.logical.Association

Description

A logical Association represents a relationship between two or more logical Entities. The logical Entities participating in the logical Association may be specified locally or in its generalized types. In addition, there may be one or more logical ComposableElements that characterize the relationship. Logical Associations are logical Entities that may also participate in other logical Associations. The relationships for the Association meta-class are listed in Table 88.

Table 88: datamodel.logical.Association Relationships

Type	Name	Target	Multiplicity
Composition	participant	Participant	0..*
Generalization		Entity	

5.4.48 Meta-Class: datamodel.logical.Participant

Description

A logical Participant is the mechanism that allows a logical Association to be constructed between two or more logical Entities. The “type” of a logical Participant is the logical Entity being used to construct the logical Association. The “sourceLowerBound” and “sourceUpperBound” attributes define the multiplicity of the logical Association relative to the Participant. A “sourceUpperBound” multiplicity of –1 represents an unbounded sequence. The “path” attribute of the Participant describes the chain of entity characteristics to traverse to reach the subject of the association beginning with the entity referenced by the “type” attribute. The attributes for the Participant meta-class are listed in Table 89, and its relationships are shown in Table 90.

Table 89: datamodel.logical.Participant Attributes

Name	Type	Multiplicity
sourceLowerBound	int	1
sourceUpperBound	int	1

Table 90: datamodel.logical.Participant Relationships

Type	Name	Target	Multiplicity
Association	type	Entity	1
Association	realizes	datamodel.conceptual.Participant	1
Composition	path	PathNode	0..1

Type	Name	Target	Multiplicity
Generalization		Characteristic	

5.4.49 Meta-Class: datamodel.logical.PathNode

Description

A logical PathNode is a single element in a chain that collectively forms a path specification. The relationships for the PathNode meta-class are listed in Table 91.

Table 91: datamodel.logical.PathNode Relationships

Type	Name	Target	Multiplicity
Composition	node	PathNode	0..1

5.4.50 Meta-Class: datamodel.logical.ParticipantPathNode

Description

A logical ParticipantPathNode is a logical PathNode that selects a Participant that references an Entity. This provides a mechanism for reverse navigation from an Entity that participates in an Association back to the Association. The relationships for the ParticipantPathNode meta-class are listed in Table 92.

Table 92: datamodel.logical.ParticipantPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedParticipant	Participant	1
Generalization		PathNode	

5.4.51 Meta-Class: datamodel.logical.CharacteristicPathNode

Description

A logical CharacteristicPathNode is a logical PathNode that selects a logical Characteristic which is directly contained in a logical Entity or Association. The relationships for the CharacteristicPathNode meta-class are listed in Table 93.

Table 93: datamodel.logical.CharacteristicPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedCharacteristic	Characteristic	1
Generalization		PathNode	

5.4.52 Meta-Class: datamodel.logical.View

Description

A logical View is a logical Query or a logical CompositeQuery. The relationships for the View meta-class are listed in Table 94.

Table 94: datamodel.logical.View Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.4.53 Meta-Class: datamodel.logical.Query

Description

A logical Query is a specification that defines the content of logical View as a set of logical Characteristics projected from a selected set of related logical Entities. The “specification” attribute captures the specification of a Query as defined by the Query grammar. The attributes for the Query meta-class are listed in Table 95, and its relationships are shown in Table 96.

Table 95: datamodel.logical.Query Attributes

Name	Type	Multiplicity
specification	string	1

Table 96: datamodel.logical.Query Relationships

Type	Name	Target	Multiplicity
Association	realizes	datamodel.conceptual.Query	0..1
Generalization		View	

5.4.54 Meta-Class: datamodel.logical.CompositeQuery

Description

A logical CompositeQuery is a collection of two or more logical Views. The “isUnion” attribute specifies whether the composed Views are intended to be mutually exclusive or not. The attributes for the CompositeQuery meta-class are listed in Table 97, and its relationships are shown in Table 98.

Table 97: datamodel.logical.CompositeQuery Attributes

Name	Type	Multiplicity
isUnion	boolean	1

Table 98: datamodel.logical.CompositeQuery Relationships

Type	Name	Target	Multiplicity
Composition	composition	QueryComposition	2..*
Association	realizes	datamodel.conceptual.CompositeQuery	0..1
Generalization		Element	
Generalization		View	

5.4.55 Meta-Class: datamodel.logical.QueryComposition

Description

A logical QueryComposition is the mechanism that allows a logical CompositeQuery to be constructed from logical Queries and other logical CompositeQueries. The “rolename” attribute defines the name of the composed logical View within the scope of the composing logical CompositeQuery. The “type” of a logical QueryComposition is the logical View being used to construct the logical CompositeQuery. The attributes for the QueryComposition meta-class are listed in Table 99, and its relationships are shown in Table 100.

Table 99: datamodel.logical.QueryComposition Attributes

Name	Type	Multiplicity
rolename	string	1

Table 100: datamodel.logical.QueryComposition Relationships

Type	Name	Target	Multiplicity
Association	realizes	datamodel.conceptual.QueryComposition	0..1
Association	type	View	1

5.5 Meta-Package: datamodel.platform

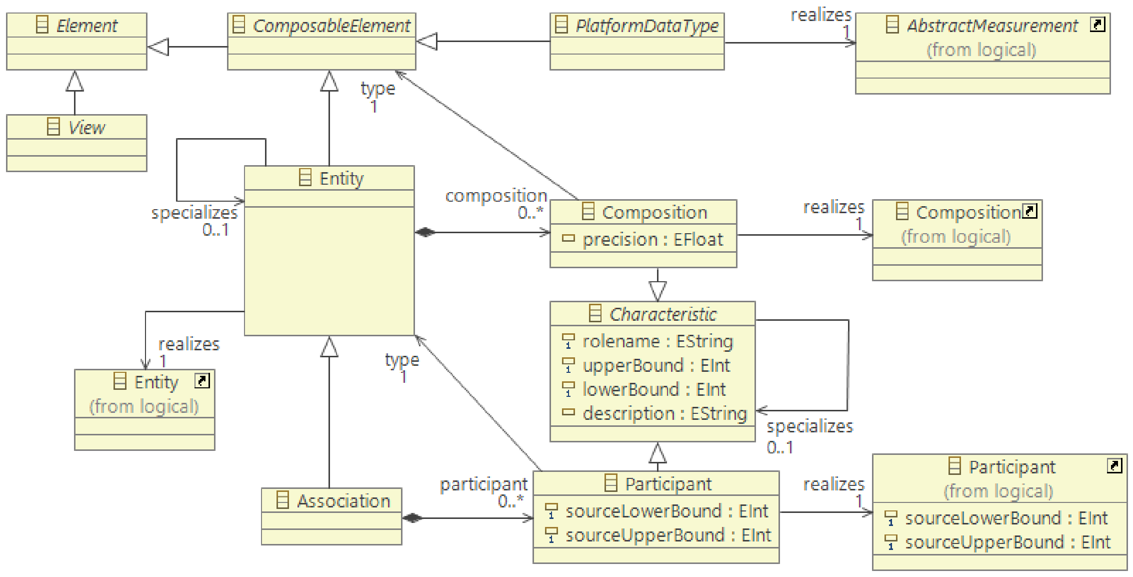

Figure 14: Data Model Metamodel "datamodel.platform" Package

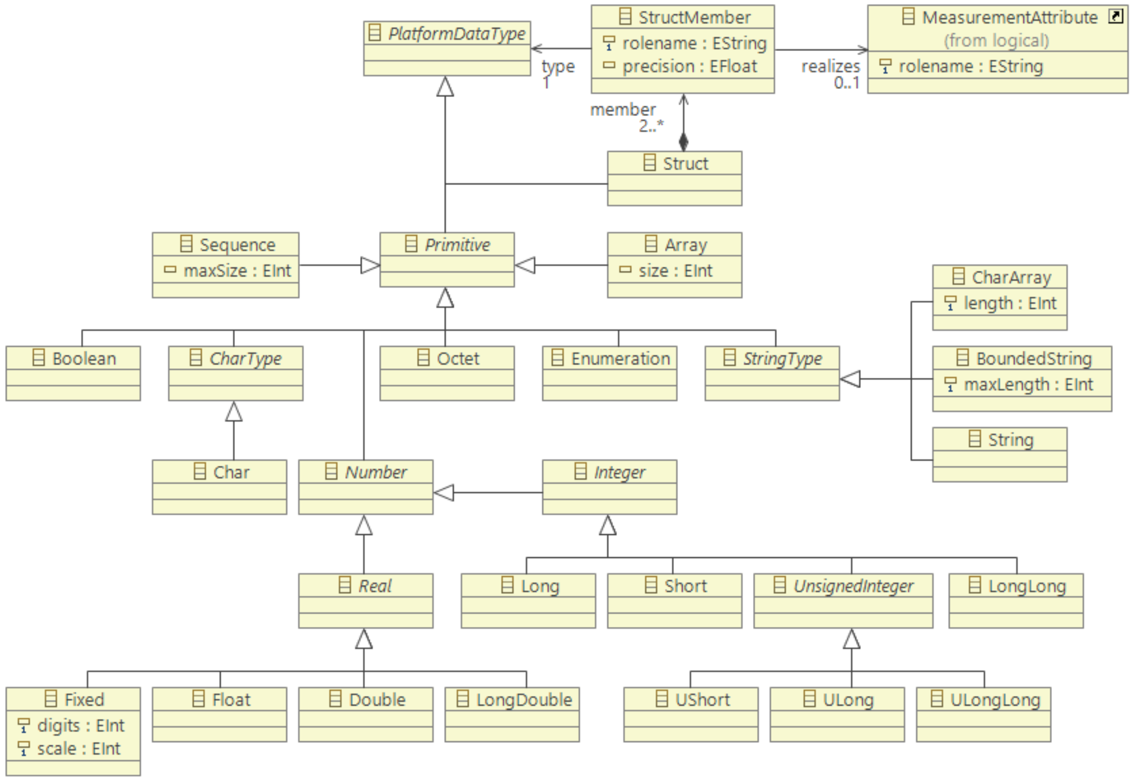

Figure 15: Data Model Metamodel "datamodel.platform" Package: Platform Value Types

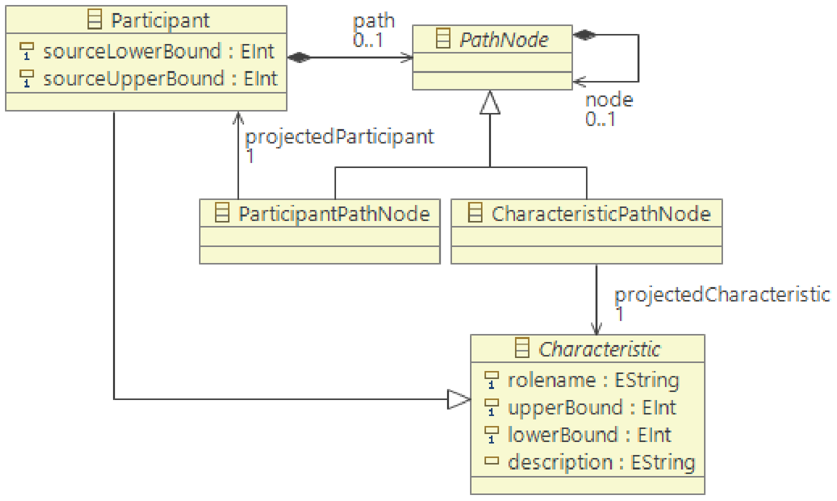

Figure 16: Data Model Metamodel "datamodel.platform" Package: Participant Path

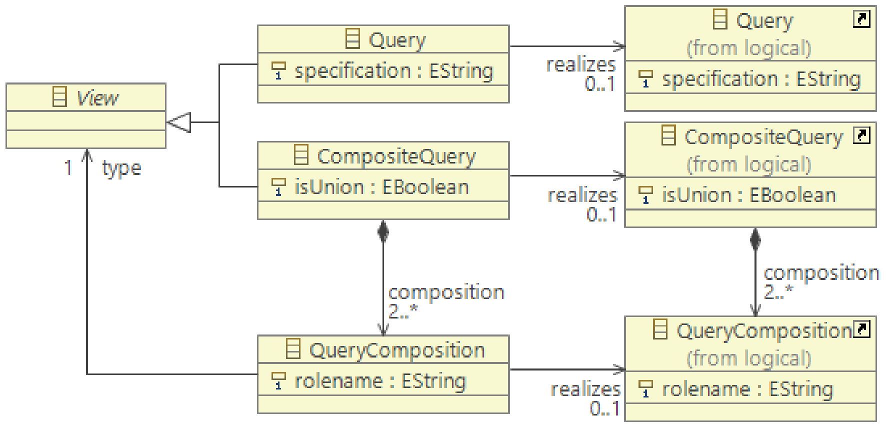

Figure 17: Data Model Metamodel "datamodel.platform" Package: Views

5.5.1 Meta-Class: datamodel.platform.Element

Description

A platform Element is the root type for defining the platform elements of a Data Model. The relationships for the Element meta-class are listed in Table 101.

Table 101: datamodel.platform.Element Relationships

Type	**Name**	**Target**	**Multiplicity**
Generalization		datamodel.Element	

5.5.2 Meta-Class: datamodel.platform.ComposableElement

Description

A platform ComposableElement is a platform Element that is allowed to participate in a Composition relationship. In other words, these are the platform Elements that may be a characteristic of a platform Entity. The relationships for the ComposableElement meta-class are listed in Table 102.

Table 102: datamodel.platform.ComposableElement Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.5.3 Meta-Class: datamodel.platform.PlatformDataType

Description

A PlatformDataType is a platform realization of a logical AbstractMeasurement and is either a Primitive or a Struct. The relationships for the PlatformDataType meta-class are listed in Table 103.

Table 103: datamodel.platform.PlatformDataType Relationships

Type	Name	Target	Multiplicity
Association	realizes	datamodel.logical.AbstractMeasurement	1
Generalization		ComposableElement	

5.5.4 Meta-Class: datamodel.platform.Primitive

Description

A platform Primitive realizes a logical AbstractMeasurement in terms of a simple data type. The relationships for the Primitive meta-class are listed in Table 104.

Table 104: datamodel.platform.Primitive Relationships

Type	Name	Target	Multiplicity
Generalization		PlatformDataType	

5.5.5 Meta-Class: datamodel.platform.Boolean

Description

A Boolean is a data type that represents the values TRUE and FALSE. The relationships for the Boolean meta-class are listed in Table 105.

Table 105: datamodel.platform.Boolean Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.6 Meta-Class: datamodel.platform.Octet

Description

An Octet is an 8-bit quantity that is guaranteed not to undergo any conversion during transfer between systems. The relationships for the Octet meta-class are listed in Table 106.

Table 106: datamodel.platform.Octet Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.7 Meta-Class: datamodel.platform.CharType

Description

A CharType is a Char. The relationships for the CharType meta-class are listed in Table 107.

Table 107: datamodel.platform.CharType Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.8 Meta-Class: datamodel.platform.Char

Description

A Char is a data type that represents characters from any single byte character set. The relationships for the Char meta-class are listed in Table 108.

Table 108: datamodel.platform.Char Relationships

Type	Name	Target	Multiplicity
Generalization		CharType	

5.5.9 Meta-Class: datamodel.platform.StringType

Description

A StringType is a BoundedString, an unbounded String, or a CharArray. The relationships for the StringType meta-class are listed in Table 109.

Table 109: datamodel.platform.StringType Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.10 Meta-Class: datamodel.platform.String

Description

A String is a data type that represents a variable length sequence of Char (all 8-bit quantities except NULL). The length is a non-negative integer, and is available at run-time. The length is not maximally bounded. The relationships for the String meta-class are listed in Table 110.

Table 110: datamodel.platform.String Relationships

Type	Name	Target	Multiplicity
Generalization		StringType	

5.5.11 Meta-Class: datamodel.platform.BoundedString

Description

A BoundedString is a data type that represents a variable length sequence of Char (all 8-bit quantities except NULL). The length is a non-negative integer, and is available at run-time. The "maxLength" attribute defines the maximum length of the BoundedString, an integer value greater than 0. The attributes for the BoundedString meta-class are listed in Table 111, and its relationships are shown in Table 112.

Table 111: datamodel.platform.BoundedString Attributes

Name	Type	Multiplicity
maxLength	int	1

Table 112: datamodel.platform.BoundedString Relationships

Type	Name	Target	Multiplicity
Generalization		StringType	

5.5.12 Meta-Class: datamodel.platform.CharArray

Description

A CharArray is a data type that represents a fixed length sequence of Char (all 8-bit quantities except NULL). The length is a positive integer, and is available at run-time. The "length" attribute defines the length of the CharArray, an integer value greater than 0. attributes for the CharArray meta-class are listed in Table 113, and its relationships are shown in Table 114.

Table 113: datamodel.platform.CharArray Attributes

Name	Type	Multiplicity
length	int	1

Table 114: datamodel.platform.CharArray Relationships

Type	Name	Target	Multiplicity
Generalization		StringType	

5.5.13 Meta-Class: datamodel.platform.Enumeration

Description

An Enumeration is a data type that represents an ordered list of identifiers. The relationships for the Enumeration meta-class are listed in Table 115.

Table 115: datamodel.platform.Enumeration Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.14 Meta-Class: datamodel.platform.Number

Description

A Number is an abstract meta-class from which all meta-classes representing numeric values derive. The relationships for the Number meta-class are listed in Table 116.

Table 116: datamodel.platform.Number Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.15 Meta-Class: datamodel.platform.Integer

Description

An Integer is an abstract meta-class from which all meta-classes representing whole numbers derive. The relationships for the Integer meta-class are listed in Table 117.

Table 117: datamodel.platform.Integer Relationships

Type	Name	Target	Multiplicity
Generalization		Number	

5.5.16 Meta-Class: datamodel.platform.Short

Description

A Short is an integer data type that represents integer values in the range –2^15 to (2^15 – 1). The relationships for the Short meta-class are listed in Table 118.

Table 118: datamodel.platform.Short Relationships

Type	Name	Target	Multiplicity
Generalization		Integer	

5.5.17 Meta-Class: datamodel.platform.Long

Description

A Long is an integer data type that represents integer values in the range –2^31 to (2^31 – 1). The relationships for the Long meta-class are listed in Table 119.

Table 119: datamodel.platform.Long Relationships

Type	Name	Target	Multiplicity
Generalization		Integer	

5.5.18 Meta-Class: datamodel.platform.LongLong

Description

A LongLong is an integer data type that represents integer values in the range –2^63 to (2^63 – 1). The relationships for the LongLong meta-class are listed in Table 120.

Table 120: datamodel.platform.LongLong Relationships

Type	Name	Target	Multiplicity
Generalization		Integer	

5.5.19 Meta-Class: datamodel.platform.Real

Description

A Real is an abstract meta-class from which all meta-classes representing real numbers derive. The relationships for the Real meta-class are listed in Table 121.

Table 121: datamodel.platform.Real Relationships

Type	Name	Target	Multiplicity
Generalization		Number	

5.5.20 Meta-Class: datamodel.platform.Double

Description

A Double is a real data type that represents an IEEE double precision floating-point number. The relationships for the Double meta-class are listed in Table 122.

Table 122: datamodel.platform.Double Relationships

Type	Name	Target	Multiplicity
Generalization		Real	

5.5.21 Meta-Class: datamodel.platform.LongDouble

Description

A LongDouble is a real data type that represents an IEEE extended double precision floating-point number (having a signed fraction of at least 64 bits and an exponent of at least 15 bits). The relationships for the LongDouble meta-class are listed in Table 123.

Table 123: datamodel.platform.LongDouble Relationships

Type	Name	Target	Multiplicity
Generalization		Real	

5.5.22 Meta-Class: datamodel.platform.Float

Description

A Float is a real data type that represents an IEEE single precision floating-point number. The relationships for the Float meta-class are listed in Table 124.

Table 124: datamodel.platform.Float Relationships

Type	Name	Target	Multiplicity
Generalization		Real	

5.5.23 Meta-Class: datamodel.platform.Fixed

Description

A Fixed is a real data type that represents a fixed-point decimal number of up to 31 significant digits. The “digits” attribute defines the total number of digits, a non-negative integer value less than or equal to 31. The “scale” attribute defines the position of the decimal point in the number, and cannot be greater than “digits”. The attributes for the Fixed meta-class are listed in Table 125, and its relationships are shown in Table 126.

Table 125: datamodel.platform.Fixed Attributes

Name	Type	Multiplicity
digits	int	1
scale	int	1

Table 126: datamodel.platform.Fixed Relationships

Type	Name	Target	Multiplicity
Generalization		Real	

5.5.24 Meta-Class: datamodel.platform.UnsignedInteger

Description

An UnsignedInteger is an abstract meta-class from which all meta-classes representing unsigned whole numbers derive. The relationships for the UnsignedInteger meta-class are listed in Table 127.

Table 127: datamodel.platform.UnsignedInteger Relationships

Type	Name	Target	Multiplicity
Generalization		Integer	

5.5.25 Meta-Class: datamodel.platform.UShort

Description

A UShort is an integer data type that represents integer values in the range 0 to (2^16 – 1). The relationships for the UShort meta-class are listed in Table 128.

Table 128: datamodel.platform.UShort Relationships

Type	Name	Target	Multiplicity
Generalization		UnsignedInteger	

5.5.26 Meta-Class: datamodel.platform.ULong

Description

A ULong is an integer data type that represents integer values in the range 0 to (2^32 – 1). The relationships for the ULong meta-class are listed in Table 129.

Table 129: datamodel.platform.ULong Relationships

Type	Name	Target	Multiplicity
Generalization		UnsignedInteger	

5.5.27 Meta-Class: datamodel.platform.ULongLong

Description

A ULongLong is an integer data type that represents integer values in the range 0 to (2^64 – 1). The relationships for the ULongLong meta-class are listed in Table 130.

Table 130: datamodel.platform.ULongLong Relationships

Type	Name	Target	Multiplicity
Generalization		UnsignedInteger	

5.5.28 Meta-Class: datamodel.platform.Sequence

Description

A Sequence is used to represent a sequence of Octets. This can be used to realize a StandardMeasurementSystem. The attributes for the Sequence meta-class are listed in Table 131, and its relationships are shown in Table 132.

Table 131: datamodel.platform.Sequence Attributes

Name	Type	Multiplicity
maxSize	int	0..1

Table 132: datamodel.platform.Sequence Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.29 Meta-Class: datamodel.platform.Array

Description

An Array is used to represent an array of Octets. This can be used to realize a StandardMeasurementSystem. The attributes for the Array meta-class are listed in Table 133, and its relationships are shown in Table 134.

Table 133: datamodel.platform.Array Attributes

Name	Type	Multiplicity
size	int	0..1

Table 134: datamodel.platform.Array Relationships

Type	Name	Target	Multiplicity
Generalization		Primitive	

5.5.30 Meta-Class: datamodel.platform.Struct

Description

A platform Struct is a structured realization of a logical AbstractMeasurement. It is composed of PlatformDataTypes (i.e., Primitives and other Structs composed of Primitives). A platform Struct's composition hierarchy is consistent with the composition hierarchy of the logical AbstractMeasurement that it realizes. Each composed PlatformDataType realizes a logical AbstractMeasurement. The relationships for the Struct meta-class are listed in Table 135.

Table 135: datamodel.platform.Struct Relationships

Type	Name	Target	Multiplicity
Composition	member	StructMember	2..* {Ordered}
Generalization		PlatformDataType	

5.5.31 Meta-Class: datamodel.platform.StructMember

Description

A StructMember is the mechanism that allows Structs to be constructed from other PlatformDataTypes. The "type" of a StructMember is the PlatformDataType being used to construct the Struct. If "type" is a Primitive, the "precision" attribute specifies a measure of the detail in which a quantity is captured. The attributes for the StructMember meta-class are listed in Table 136, and its relationships are shown in Table 137.

Table 136: datamodel.platform.StructMember Attributes

Name	Type	Multiplicity
rolename	string	1
precision	float	0..1

Table 137: datamodel.platform.StructMember Relationships

Type	Name	Target	Multiplicity
Association	type	PlatformDataType	1
Association	realizes	datamodel.logical.MeasurementAttribute	0..1

5.5.32 Meta-Class: datamodel.platform.Characteristic

Description

A platform Characteristic contributes to the uniqueness of a platform Entity. The “rolename” attribute defines the name of the platform Characteristic within the scope of the platform Entity. The “lowerBound” and “upperBound” attributes define the multiplicity of the composed Characteristic. An “upperBound” multiplicity of –1 represents an unbounded sequence. The attributes for the Characteristic meta-class are listed in Table 138, and its relationships are shown in Table 139.

Table 138: datamodel.platform.Characteristic Attributes

Name	Type	Multiplicity
rolename	string	1
upperBound	int	1
lowerBound	int	1
description	string	0..1

Table 139: datamodel.platform.Characteristic Relationships

Type	Name	Target	Multiplicity
Association	specializes	Characteristic	0..1

5.5.33 Meta-Class: datamodel.platform.Entity

Description

A platform Entity “realizes” a logical Entity in terms of PhysicalDataTypes and other platform Entities composed of PhysicalDataTypes. A platform Entity’s composition hierarchy is consistent with the composition hierarchy of the logical Entity that it realizes. The platform Entity’s composed Entities realize one to one the logical Entity’s composed Entities; the platform Entity’s composed PhysicalDataTypes realize many to one the logical Entity’s composed Measurements. The relationships for the Entity meta-class are listed in Table 140.

Table 140: datamodel.platform.Entity Relationships

Type	Name	Target	Multiplicity
Composition	composition	Composition	0..* {Ordered}
Association	realizes	datamodel.logical.Entity	1
Association	specializes	Entity	0..1
Generalization		ComposableElement	

5.5.34 Meta-Class: datamodel.platform.Composition

Description

A platform Composition is the mechanism that allows platform Entities to be constructed from other platform ComposableElements. The "type" of a Composition is the ComposableElement being used to construct the platform Entity. The "lowerBound" and "upperBound" define the multiplicity of the composed platform Entity. An "upperBound" multiplicity of –1 represents an unbounded sequence. If "type" is a Primitive, the "precision" attribute specifies a measure of the detail in which a quantity is captured. The attributes for the Composition meta-class are listed in Table 141, and its relationships are shown in Table 142.

Table 141: datamodel.platform.Composition Attributes

Name	Type	Multiplicity
precision	float	0..1

Table 142: datamodel.platform.Composition Relationships

Type	Name	Target	Multiplicity
Association	type	ComposableElement	1
Association	realizes	datamodel.logical.Composition	1
Generalization		Characteristic	

5.5.35 Meta-Class: datamodel.platform.Association

Description

A platform Association represents a relationship between two or more platform Entities. The platform Entities participating in the platform Association may be specified locally or in its generalized types. In addition, there may be one or more platform ComposableElements that characterize the relationship. Platform Associations are platform Entities that may also participate in other platform Associations. The relationships for the Association meta-class are listed in Table 143.

Table 143: datamodel.platform.Association Relationships

Type	Name	Target	Multiplicity
Composition	participant	Participant	0..* {Ordered}
Generalization		Entity	

5.5.36 Meta-Class: datamodel.platform.Participant

Description

A platform Participant is the mechanism that allows a platform Association to be constructed between two or more platform Entities. The "type" of a platform Participant is the platform

Entity being used to construct the platform Association. The “sourceLowerBound” and “sourceUpperBound” attributes define the multiplicity of the platform Association relative to the Participant. A “sourceUpperBound” multiplicity of –1 represents an unbounded sequence. The “path” attribute of the Participant describes the chain of entity characteristics to traverse to reach the subject of the association beginning with the entity referenced by the “type” attribute. The attributes for the Participant meta-class are listed in Table 144, and its relationships are shown in Table 145.

Table 144: datamodel.platform.Participant Attributes

Name	Type	Multiplicity
sourceLowerBound	int	1
sourceUpperBound	int	1

Table 145: datamodel.platform.Participant Relationships

Type	Name	Target	Multiplicity
Association	type	Entity	1
Association	realizes	datamodel.logical.Participant	1
Composition	path	PathNode	0..1
Generalization		Characteristic	

5.5.37 Meta-Class: datamodel.platform.PathNode

Description

A platform PathNode is a single element in a chain that collectively forms a path specification. The relationships for the PathNode meta-class are listed in Table 146.

Table 146: datamodel.platform.PathNode Relationships

Type	Name	Target	Multiplicity
Composition	node	PathNode	0..1

5.5.38 Meta-Class: datamodel.platform.ParticipantPathNode

Description

A platform ParticipantPathNode is a platform PathNode that selects a Participant that references an Entity. This provides a mechanism for reverse navigation from an Entity that participates in an Association back to the Association. The relationships for the ParticipantPathNode meta-class are listed in Table 147.

Table 147: datamodel.platform.ParticipantPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedParticipant	Participant	1
Generalization		PathNode	

5.5.39 Meta-Class: datamodel.platform.CharacteristicPathNode

Description

A platform CharacteristicPathNode is a platform PathNode that selects a platform Characteristic which is directly contained in a platform Entity or Association. The relationships for the CharacteristicPathNode meta-class are listed in Table 148.

Table 148: datamodel.platform.CharacteristicPathNode Relationships

Type	Name	Target	Multiplicity
Association	projectedCharacteristic	Characteristic	1
Generalization		PathNode	

5.5.40 Meta-Class: datamodel.platform.View

Description

A platform View is a platform Query or a platform CompositeQuery. The relationships for the View meta-class are listed in Table 149.

Table 149: datamodel.platform.View Relationships

Type	Name	Target	Multiplicity
Generalization		Element	

5.5.41 Meta-Class: datamodel.platform.Query

Description

A platform Query is a specification that defines the content of platform View as a set of platform Characteristics projected from a selected set of related platform Entities. The "specification" attribute captures the specification of a Query as defined by the Query grammar. The attributes for the Query meta-class are listed in Table 150, and its relationships are shown in Table 151.

Table 150: datamodel.platform.Query Attributes

Name	Type	Multiplicity
specification	string	1

Table 151: datamodel.platform.Query Relationships

Type	Name	Target	Multiplicity
Association	realizes	datamodel.logical.Query	0..1
Generalization		Element	
Generalization		View	

5.5.42 Meta-Class: datamodel.platform.CompositeQuery

Description

A platform CompositeQuery is a collection of two or more platform Views. The “isUnion” attribute specifies whether the composed Views are intended to be mutually exclusive or not. The attributes for the CompositeQuery meta-class are listed in Table 152, and its relationships are shown in Table 153.

Table 152: datamodel.platform.CompositeQuery Attributes

Name	Type	Multiplicity
isUnion	boolean	1

Table 153: datamodel.platform.CompositeQuery Relationships

Type	Name	Target	Multiplicity
Composition	composition	QueryComposition	2..* {Ordered}
Association	realizes	datamodel.logical.CompositeQuery	0..1
Generalization		Element	
Generalization		View	

5.5.43 Meta-Class: datamodel.platform.QueryComposition

Description

A platform QueryComposition is the mechanism that allows a platform CompositeQuery to be constructed from platform Queries and other platform CompositeQueries. The “rolename” attribute defines the name of the composed platform View within the scope of the composing platform CompositeQuery. The “type” of a platform QueryComposition is the platform View being used to construct the platform CompositeQuery. The attributes for the QueryComposition meta-class are listed in Table 154, and its relationships are shown in Table 155.

Table 154: datamodel.platform.QueryComposition Attributes

Name	Type	Multiplicity
rolename	string	1

Table 155: datamodel.platform.QueryComposition Relationships

Type	Name	Target	Multiplicity
Association	realizes	datamodel.logical.QueryComposition	0..1
Association	type	View	1

6 Query Language

This chapter specifies the UDDL query language.

6.1 Query Grammar

The following Extended Backus-Naur Form grammar defines the language for a Query specification.

```
(*
A query_specification represents a Query in the Data Model.
*)

query_specification = query_statement ;

(*
A query_statement is the expression of a Query, which is a declaration of a set
of data in terms of a set of Entities and their Characteristics. The
selected_entity_expression defines the context for a query_statement's data set
as a set of related Entities and optionally a set of conditions expressed over
their Characteristics that are true for all data in the data set. The
projected_characteristic_list identifies the specific Characteristics that are
the elements of the query_statement's data set. The set_qualifier DISTINCT, if
specified, indicates that instances of data in the data set are not duplicated.
Otherwise, they may be duplicated.
*)

query_statement = kw_select , [ set_qualifier ] , projected_characteristic_list
, selected_entity_expression ;

(*
A set_qualifier indicates whether instances of data in a set are unique
(kw_distinct) or not (kw_all).
*)

set_qualifier = kw_distinct | kw_all ;

(*
A projected_characteristic_list defines the set of Characteristics in a
query_statement. all_characteristics indicates that every Characteristic of
every selected_entity is included. Otherwise, the Characteristics are those
specified by the projected_characteristic_expressions. In both cases, only
those Characteristics whose types are not Entities are included.
*)

projected_characteristic_list = all_characteristics |
projected_characteristic_expression , { comma ,
projected_characteristic_expression } ;

(*
all_characteristics is a shorthand notation indicating "every Characteristic".
*)

all_characteristics = "*" ;
```

```
(*
A projected_characteristic_expression represents one or more Characteristics of
a specific Entity.
*)

projected_characteristic_expression =
selected_entity_characteristic_wildcard_reference |
explicit_selected_entity_characteristic_reference ;

(*
A selected_entity_characteristic_wildcard_reference is a shorthand notation
indicating "every Characteristic of selected_entity_reference".
*)

selected_entity_characteristic_wildcard_reference = selected_entity_reference ,
period , all_characteristics ;

(*
An explicit_selected_entity_characteristic_reference represents one
Characteristic of one Entity. projected_characteristic_alias specifies an
optional alias for the Characteristic for use elsewhere in a query_statement.
*)

explicit_selected_entity_characteristic_reference =
selected_entity_characteristic_reference , [ [ kw_as ] ,
projected_characteristic_alias ] ;

(*
A selected_entity_expression defines the context for a query_statement's data
set as a set of related Entities (using the from_clause). It may also specify a
set of conditions expressed over Characteristics of those Entities that are
true for all data in the data set (using a where_clause) and specify how data
in the data set is ordered (using an order_by_clause).
*)

selected_entity_expression = from_clause , [ where_clause ] , [ order_by_clause
] ;

(*
A from_clause identifies a set of related Entities via entity_expression.
*)

from_clause = kw_from , entity_expression ;

(*
An entity_expression identifies a set of Entities and a set of relationships
between those Entities. The set of Entities are the selected_entitys in the
entity_expression. The set of relationships are identified by the
entity_type_characteristic_equivalence_expressions in the entity_expression.
*)

entity_expression = selected_entity , { entity_join_expression } ;

(*
A selected_entity is the Entity whose name is entity_type_reference.
selected_entity_alias specifies an alias for the Entity for use elsewhere in a
query_statement.
*)

selected_entity = entity_type_reference , [ [ kw_as ] , selected_entity_alias ]
;

(*
An entity_join_expression identifies an Entity (join_entity) and one or more
relationships between it and other selected_entitys in the entity_expression.
Each relationship is identified with an
entity_type_characteristic_equivalence_expression whose
selected_entity_characteristic_reference is either a Characteristic of
join_entity whose type is another selected_entity, or is a Characteristic of
another selected_entity whose type is join_entity.
*)

entity_join_expression = kw_join , join_entity , kw_on , entity_join_criteria ;

(*
A join_entity is a selected_entity in an entity_join_expression.
*)

join_entity = selected_entity ;

(*
An entity_join_criteria identifies one or more relationships between two or
more selected_entitys in the entity_expression. Each
entity_type_characteristic_equivalence_expression identifies a relationship
between two selected_entitys.
*)

entity_join_criteria = entity_type_characteristic_equivalence_expression , {
kw_and , entity_type_characteristic_equivalence_expression } ;

(*
An entity_type_characteristic_equivalence_expression identifies a relationship
between two selected_entitys. selected_entity_characteristic_reference is a
Characteristic of one selected_entity whose type is the another selected_entity
in the entity_expression. A selected_entity_reference is used to identify a
specific selected_entity should there be more than one selected_entity whose
type is that Characteristic's type.
*)

entity_type_characteristic_equivalence_expression =
selected_entity_characteristic_reference , [ equals_operator ,
selected_entity_reference ] ;

(*
A selected_entity_characteristic_reference is a selected_entity's
Characteristic specified by characteristic_reference. A
selected_entity_reference is used to identify a specific selected_entity should
there be more than one Characteristic whose name is characteristic_reference.
*)

selected_entity_characteristic_reference = [ selected_entity_reference , period
] , characteristic_reference ;

(*
A selected_entity_reference is a reference by name to a selected_entity, where
query_identifier is either a selected_entity's entity_type_reference or its
selected_entity_alias.
*)

selected_entity_reference = query_identifier ;

(*
A where_clause specifies a set of conditions that are true for all data in the
query_statement's data set.
*)
```

```
where_clause = kw_where , criteria ;

(*
A criteria specifies a set of conditions expressed over Characteristics of
selected_entitys via boolean_expression.
*)

criteria = boolean_expression ;

(*
An order_by_clause specifies how data in the query_statement's data set is
ordered. The data set is initially ordered by the first ordering_expression's
projected_characteristic_reference. Each additional ordering_expression further
orders the data set.
*)

order_by_clause = kw_order , kw_by , ordering_expression , { comma ,
ordering_expression } ;

(*
An ordering_expression specifies a Characteristic in the
projected_characteristic_list used to order data in the query_statement's data
set. If ordering_type DESC is specified, it indicates that data is ordered
descending. Otherwise, it is ordered ascending.
*)

ordering_expression = projected_characteristic_reference , [ ordering_type ] ;

(*
A projected_characteristic_reference is a reference to a Characteristic in the
projected_characteristic_list.
*)

projected_characteristic_reference =
qualified_projected_characteristic_reference |
unqualified_projected_characteristic_reference_or_alias ;

(*
A qualified_projected_characteristic_reference is a Characteristic specified by
characteristic_reference in the Entity specified by selected_entity_reference.
*)

qualified_projected_characteristic_reference = selected_entity_reference ,
period , characteristic_reference ;

(*
An unqualified_projected_characteristic_reference_or_alias is a Characteristic
in projected_characteristic_list whose rolename or assigned
projected_characteristic_alias is query_identifier. If query_identifier happens
to match both a projected_characteristic_alias and a rolename, then the
Characteristic associated with the projected_characteristic_alias is assumed.
*)

unqualified_projected_characteristic_reference_or_alias = query_identifier ;

(*
An ordering_type specifies whether data is ordered ascending (kw_asc) or
descending (kw_desc).
*)

ordering_type = kw_asc | kw_desc ;

(*
A boolean_expression is a boolean OR expression over boolean_terms.
*)

boolean_expression = boolean_term , { kw_or , boolean_term } ;

(*
A boolean_term is a boolean AND expression over boolean_factors.
*)

boolean_term = boolean_factor , { kw_and , boolean_factor } ;

(*
A boolean_factor is a boolean_predicate. If kw_not is not present, it evaluates
the same as boolean_predicate. If kw_not is present, the evaluation is the same
but negated.
*)

boolean_factor = [ kw_not ] , boolean_predicate ;

(*
A boolean_predicate is a predicate.
*)

boolean_predicate = scalar_compare_predicate | set_compare_predicate |
set_membership_predicate | exists_predicate | left_paren , boolean_expression ,
right_paren ;

(*
A scalar_compare_predicate is a function that compares two predicate_terms. It
evaluates to TRUE if the comparison is true, FALSE otherwise.
*)

scalar_compare_predicate = predicate_term , compare_operator , predicate_term ;

(*
A set_membership_predicate is a function that checks a predicate_term for
membership in a logical_set. If kw_not is not present, it evaluates to TRUE if
predicate_term is a member of logical_set, FALSE otherwise. If kw_not is
present, the evaluation is the same but negated.
*)

set_membership_predicate = predicate_term , [ kw_not ] , kw_in , logical_set ;

(*
A logical_set is a set of data. If logical_set is a subquery with one
Characteristic in its projected_characteristic_list, the set is the data
associated with that Characteristic. If logical_set is a
characteristic_basis_set, the set is the data associated with each
characteristic_basis. If logical_set is an enum_literal_set, the set is the
specified EnumerationLabels.
*)

logical_set = subquery | characteristic_basis_set | enum_literal_set ;

(*
A characteristic_basis_set is one or more characteristic_basis.
*)

characteristic_basis_set = left_paren , characteristic_basis , { comma ,
characteristic_basis } , right_paren ;

(*
A set_compare_predicate is a function that represents a pair-wise comparison of
predicate_term with all members of compare_set. If set_compare_quantifier is
kw_all, the function evaluates to TRUE if the application of compare_operator
evaluates to TRUE for predicate_term and every member of the set, FALSE
otherwise. If set_compare_quantifier is kw_some, the function evaluates to TRUE
if the application of compare_operator evaluates to TRUE for predicate_term and
at least one member in the set, FALSE otherwise.
*)

set_compare_predicate = predicate_term , compare_operator ,
set_compare_quantifier , compare_set ;

(*
A compare_set is a subquery with a single Characteristic in its
projected_characteristic_list.
*)

compare_set = subquery ;

(*
A compare_operator is a boolean comparison operator.
*)

compare_operator = equals_operator | not_equals_operator | less_than_operator |
greater_than_operator | less_than_or_equals_operator |
greater_than_or_equals_operator ;

(*
A set_compare_quantifier indicates that a comparison applies to every (kw_all)
or any (kw_some) member of a set.
*)

set_compare_quantifier = kw_all | kw_some ;

(*
An exists_predicate is a function that evaluates to TRUE if there is any data
associated with the single Characteristic in subquery's
projected_characteristic_list, FALSE otherwise.
*)

exists_predicate = kw_exists , subquery ;

(*
A predicate_term represents a DataModel Element whose associated data is
scalar.
*)

predicate_term = characteristic_basis | enum_literal_reference_expression ;

(*
A characteristic_basis is a Characteristic whose associated data is scalar. If
characteristic_basis is a subquery with one Characteristic in its
projected_characteristic_list, then the characteristic_basis is that
Characteristic. Otherwise, the Characteristic is specified by
selected_entity_characteristic_reference.
*)

characteristic_basis = selected_entity_characteristic_reference | subquery ;

(*
A subquery is a query_statement that is nested inside another query_statement.
*)
```

```
subquery = left_paren , query_statement , right_paren ;

(*
A characteristic_reference is a Characteristic whose rolename matches
query_identifier.
*)

characteristic_reference = query_identifier ;

(*
An entity_type_reference is the Entity whose name matches query_identifier.
*)

entity_type_reference = query_identifier ;

(*
An enum_literal_set is a set of EnumerationLabels. Each member in the set is
identified by the EnumerationLabel whose name is enumeration_literal_reference
in the Enumerated whose name is enumeration_type_reference.
*)

enum_literal_set = left_brace , enumeration_type_reference , colon ,
enumeration_literal_reference , { comma , enumeration_literal_reference } ,
right_brace ;

(*
An enum_literal_reference_expression is an EnumerationLabel whose name is
enumeration_literal_reference in the Enumerated whose name is
enumeration_type_reference.
*)

enum_literal_reference_expression = left_brace , enumeration_type_reference ,
colon , enumeration_literal_reference , right_brace ;

(*
An enumeration_type_reference is the Enumerated whose name matches
query_identifier.
*)

enumeration_type_reference = query_identifier ;

(*
An enumeration_literal_reference is an EnumerationLabel whose name matches
query_identifier.
*)

enumeration_literal_reference = query_identifier ;

(*
A selected_entity_alias is an alias for a selected_entity.
*)

selected_entity_alias = query_identifier ;

(*
A projected_characteristic_alias provides an alias for a Characteristic in a
projected_characteristic_list.
*)

projected_characteristic_alias = query_identifier ;

(*
A query_identifier is an alphanumeric string used to represent an identifier in
a query_specification.
*)

query_identifier = identifier ;

(*
The following terms start with kw_ to indicate that they are keyword tokens.
*)

kw_all = "ALL" | "all" ;

kw_some = "SOME" | "some" | "ANY" | "any" ;

kw_exists = "EXISTS" | "exists" ;

kw_not = "NOT" | "not" ;

kw_in = "IN" | "in" ;

kw_select = "SELECT" | "select" ;

kw_and = "AND" | "and" ;

kw_or = "OR" | "or" ;

kw_as = "AS" | "as" ;

kw_distinct = "DISTINCT" | "distinct" ;

kw_from = "FROM" | "from" ;

kw_where = "WHERE" | "where" ;

kw_by = "BY" | "by" ;

kw_join = "JOIN" | "join" ;

kw_on = "ON" | "on" ;

kw_order = "ORDER" | "order" ;

kw_asc = "ASC" | "asc" ;

kw_desc = "DESC" | "desc" ;

(*
The following terms represent boolean operator tokens.
*)

(*
An equals_operator is the boolean equals operator.
*)

equals_operator = "=" ;

(*
A not_equals_operator is the boolean not equals operator.
*)

not_equals_operator = "<>" | "!=" ;

(*
A less_than_operator is the boolean less-than operator.
*)

less_than_operator = "<" ;

(*
A greater_than_operator is the boolean greater-than operator.
*)

greater_than_operator = ">" ;

(*
A greater_than_or_equals_operator is the boolean greater-than-or-equals
operator.
*)

greater_than_or_equals_operator = ">=" ;

(*
A less_than_or_equals_operator is the boolean less-than-or-equals operator.
*)

less_than_or_equals_operator = "<=" ;

(*
The following terms represent punctuation tokens.
*)

comma = "," ;

left_paren = "(" ;

right_paren = ")" ;

period = "." ;

left_brace = "{" ;

right_brace = "}" ;

colon = ":" ;

(*
The following terms represent identifier tokens.
*)

identifier = letter , { letter | digit_literal } ;

letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l"
| "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" |
"z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" |
"Z" | "_" ;

digit_literal = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
```

6.2 Query Constraints

In the Legend below, the left column of the table identifies and demonstrates the typography of the key elements in this specification. The key elements of this specification are distinguished by their unique typography, including color. The right column describes the meaning of the key elements of the specification.

Legend

Key Element	**Description**
term	The name of a term in the Query grammar. A set of rules may follow a term. These rules apply to all instances of this term in a Query. This term is referred to as the rule's "context term" in this legend.
term_name	The name of a term in the Query grammar. In a rule, it represents some instance of the named term in a Query. (Rules reference the context term as well as other terms in its expression.)
DATAMODELMETATYPE_NAME	The name of a DataModel meta-type. In a rule, it represents some instance of the named meta-type in a DataModel.
property_name	The name of DataModel meta-type property. In a rule, it represents a value associated with the named property of some instance of a DataModel meta-type in a DataModel.
"literal"	Represents a literal value.

query_specification

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a CONCEPTUALDATAMODEL, then:

- An ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a CONCEPTUALDATAMODEL
- An ***enum_literal_reference_expression*** must not be specified
- An ***enum_literal_set*** must not be specified

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a LOGICALDATAMODEL, then an ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a LOGICALDATAMODEL.

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a PLATFORMDATAMODEL, then an ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a PLATFORMDATAMODEL.

query_statement

A ***selected_entity_reference*** in a ***projected_characteristic_expression*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***.

If an ***explicit_selected_entity_characteristic_reference*** is specified, and ***selected_entity_reference*** is not specified in its **selected_entity_characteristic_reference**, then its ***characteristic_reference*** must match the rolename of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***.

If a ***where_clause*** is specified, then:

- There must be at least one ***boolean_predicate*** in either its criteria or in a nested subquery's ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***query_statement*** is not a subquery, then for each ***selected_entity_characteristic_reference predicate_term*** in the ***where_clause***'s criteria:

  — If a ***selected_entity_reference*** is specified, then the ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***

  — If a ***selected_entity_reference*** is not specified, then the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***query_statement*** is a subquery, then for each ***selected_entity_characteristic_reference*** **predicate_term** in the ***where_clause***'s criteria:

  — If a ***selected_entity_reference*** is specified, then the ***selected_entity_reference*** must either:

  A: Match by name a ***selected_entity_alias*** in the subquery's ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the subquery's ***entity_expression***, or

  B: Match by name a ***selected_entity_alias*** in an outer ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in an outer ***query_statement***'s ***entity_expression***.

  If both A and B above are true, then the ***selected_entity*** from A is assumed.

  — If a ***selected_entity_reference*** is not specified, then the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in either:

  1. The ***subquery's entity_expression***, or

  2. An outer ***query_statement***'s ***entity_expression***.

  If both 1 and 2 above are true, then the CHARACTERISTIC from 1 is assumed.
- If its ***criteria*** contains a ***scalar_compare_predicate***, then:

  — If both ***predicate_term***s are a ***selected_entity_characteristic_reference***, then at least one ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

  — If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is an ***enum_literal_reference_expression***, then the

selected_entity_characteristic_reference must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is a ***subquery***, then either the ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If one ***predicate_term*** is a ***enum_literal_reference_expression*** and the other ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If both ***predicate_term***s are a ***subquery***, then there must be at least one ***boolean_predicate*** in either ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***set_compare_predicate***, then:

  — If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

  — If ***predicate_term*** is an ***enum_literal_reference_expression***, then there must be at least one ***boolean_predicate*** in either the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

  — If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either that ***subquery***'s ***where_clause*** or the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***set_membership_predicate***, then:

  If ***logical_set*** is a ***subquery***, then:

  — If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference***

predicate_term that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or the ***subquery predicate_term***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***enum_literal_reference_expression***, then there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

If ***logical_set*** is a ***characteristic_basis_set***, then:

— If a ***characteristic_basis*** in ***characteristic_basis_set*** is a ***selected_entity_characteristic_reference***, then that **selected_entity_characteristic_reference** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***selected_entity_characteristic_reference characteristic_basis*** that is a CHARACTERISTIC of a **selected_entity** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either that ***subquery***'s ***where_clause*** or a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is an ***enum_literal_reference_expression***, then there must be at least one ***selected_entity_characteristic_reference characteristic_basis*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

If ***logical_set*** is a ***enum_literal_set***, then:

— If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***exists_predicate***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

If an ***order_by_clause*** is specified, then for each ***ordering_expression***:

- If a ***qualified_projected_characteristic_reference*** is specified, then:

  — Its ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the **query_statement**'s ***entity_expression***

  — Its ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC in the ***selected_entity*** referenced by ***selected_entity_reference***

  — It must be a CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***

- If an ***unqualified_projected_characteristic_reference_or_alias*** is specified, then:

  — It either must match by name a ***projected_characteristic_alias***, or it must match the **rolename** of one and only one CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***; if it matches by name a ***projected_characteristic_alias*** and also matches the **rolename** of one and only one CHARACTERISTIC in the ***projected_characteristic_list***, then the CHARACTERISTIC associated with the ***projected_characteristic_alias*** is assumed

- Two or more ***projected_characteristic_reference***s must not reference the same CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***

set_qualifier

// No contextually relevant rules for instances of this term.

projected_characteristic_list

A ***projected_characteristic_list*** must have at least one CHARACTERISTIC.

An ***explicit_selected_entity_characteristic_reference*** must not be a CHARACTERISTIC of a ***selected_entity*** referenced by a ***selected_entity_characteristic_wildcard_reference***.

Two or more **selected_entity_characteristic_wildcard_reference**s must not reference the same ***selected_entity***.

Two or more ***explicit_selected_entity_characteristic_reference***s must not reference the same CHARACTERISTIC.

All ***projected_entity_alias***es must be unique.

all_characteristics

// No contextually relevant rules for instances of this term.

projected_characteristic_expression

// No contextually relevant rules for instances of this term.

selected_entity_characteristic_wildcard_reference

The ***selected_entity_reference*** must be a ***selected_entity*** with at least one CHARACTERISTIC whose **type** is not an ENTITY.

explicit_selected_entity_characteristic_reference

The ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC whose **type** is not an ENTITY.

selected_entity_expression

// No contextually relevant rules for instances of this term.

from_clause

// No contextually relevant rules for instances of this term.

entity_expression

All ***selected_entity_alias***es must be unique.

A ***selected_entity_alias*** must not be the same as any ***selected_entity***'s ***entity_type_reference***.

A ***selected_entity_alias*** must be specified for a ***selected_entity*** if there is more than one ***selected_entity*** with the same ***entity_type_reference***.

A ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***entity_expression***.

If ***selected_entity_reference*** is not specified in a ***selected_entity_characteristic_reference***, then the ***selected_entity_characteristic_reference***'s ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***entity_expression***.

If an ***entity_expression*** has more than one ***selected_entity***, then all of its ***selected_entity***s must be connected. That is, there must at least one simple path from each ***selected_entity*** to every other ***selected_entity*** via the expressed ***entity_join_expression***s. A simple path is formed by combining one or more path segments in sequence, where:

- Each path segment is one of the two possible ordered pairs of the two ***selected_entity***s identified by the explicitly specified or unambiguously inferred

selected_entity_references of one of the ***entity_type_characteristic_equivalence_expression***s, and

- The second ***selected_entity*** of a given path segment's ordered pair and the first ***selected_entity*** of an immediately following path segment's ordered pair must be the same, and
- The first or second ***selected_entity*** of all subsequent path segment's ordered pairs must not be the same as the first ***selected_entity*** of the first path segment's ordered pair, and
- The second ***selected_entity*** of last path segment's ordered pair must not be the same as the first or second ***selected_entity*** of all preceding path segment's ordered pairs, and
- The first ***selected_entity*** of any path segment's ordered pair must not be the same as the first ***selected_entity*** of all other path segment's ordered pairs, and
- The second ***selected_entity*** of any path segment's ordered pair must not be the same as the second ***selected_entity*** of all other path segment's ordered pairs

Two or more ***selected_entity_characteristic_reference***s must not be the same CHARACTERISTIC.

If an ***entity_join_expression*** is specified, then for each ***entity_type_characteristic_equivalence_expression*** in that ***entity_join_expression***:

- If its ***selected_entity_characteristic_reference*** is a CHARACTERISTIC of ***join_entity***, then:

  — If a ***selected_entity_reference*** is specified after the ***equals_operator***, then the CHARACTERISTIC's **type**'s **name** must match by name the ***selected_entity_reference***'s ***entity_type_reference***, and the referenced ***selected_entity*** must not be ***join_entity***

  — Otherwise, that CHARACTERISTIC's **type**'s **name** must match by name the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***entity_expression***, and that ***selected_entity*** must not be ***join_entity***

- Otherwise, the ***selected_entity_characteristic_reference*** must not be a CHARACTERISTIC of ***join_entity***, and that CHARACTERISTIC's **type**'s **name** must match by name ***join_entity***'s ***entity_type_reference***; if ***selected_entity_reference*** is specified after the ***equals_operator***, then the referenced ***selected_entity*** must be ***join_entity***

selected_entity

A ***selected_entity_alias***, if specified, must not be the same as the ***entity_type_reference***.

entity_join_expression

// No contextually relevant rules for instances of this term.

join_entity

// No contextually relevant rules for instances of this term.

entity_join_criteria

// No contextually relevant rules for instances of this term.

entity_type_characteristic_equivalence_expression

The ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC whose **type** is an ENTITY.

selected_entity_characteristic_reference

If ***selected_entity_reference*** is specified, the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC in the ***selected_entity*** referenced by ***selected_entity_reference***.

selected_entity_reference

// No contextually relevant rules for instances of this term.

where_clause

// No contextually relevant rules for instances of this term.

criteria

A ***selected_entity_characteristic_reference*** **predicate_term** must not reference a CHARACTERISTIC of a ***selected_entity*** in a nested ***subquery***'s ***entity_expression***.

order_by_clause

// No contextually relevant rules for instances of this term.

ordering_expression

// No contextually relevant rules for instances of this term.

projected_characteristic_reference

// No contextually relevant rules for instances of this term.

qualified_projected_characteristic_reference

// No contextually relevant rules for instances of this term.

unqualified_projected_characteristic_reference_or_alias

// No contextually relevant rules for instances of this term.

ordering_type

// No contextually relevant rules for instances of this term.

boolean_expression

// No contextually relevant rules for instances of this term.

boolean_term

// No contextually relevant rules for instances of this term.

boolean_factor

// No contextually relevant rules for instances of this term.

boolean_predicate

// No contextually relevant rules for instances of this term.

scalar_compare_predicate

Both ***predicate_term***s must not be ***enum_literal_reference_expression***s.

If both ***predicate_term***s are a ***selected_entity_characteristic_reference***, then both ***selected_entity_characteristic_reference***s must be a CHARACTERISTIC whose **types** are the same.

If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is an ***enum_literal_reference_expression***, then:

1. The ***selected_entity_characteristic_reference*** is a CHARACTERISTIC whose **type** must either be or realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".
2. The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in 1. If an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT.
3. The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***.

If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is a ***subquery***, then the selected_entity_characteristic_reference's **type** and the subquery's projected CHARACTERISTIC's **type** must be the same.

If one ***predicate_term*** is an ***enum_literal_reference_expression*** and the other ***predicate_term*** is a ***subquery***, then:

1. The ***subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".
2. The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in 1. If an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT.

3. The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***.

If both ***predicate_term***s are a ***subquery***, then the projected CHARACTERISTIC's **type** in both ***subquery***s must be the same.

set_membership_predicate

If ***logical_set*** is a ***subquery***, then:

- There must be one and only one CHARACTERISTIC in its ***projected_characteristic_list***, and all instances of that CHARACTERISTIC's associated data must be scalar
- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then its **type** and the ***subquery***'s projected CHARACTERISTIC's **type** must be the same
- If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***logical_set subquery***'s projected CHARACTERISTIC's **type**
- If ***predicate_term*** is an ***enum_literal_reference_expression***, then:

  — The ***logical_set subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*"

  — The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined above; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT

If ***logical_set*** is a ***characteristic_basis_set***, then:

- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then its **type** and the ***characteristic_basis_set***'s **type** must be the same
- If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***characteristic_basis_set***'s **type**
- If ***predicate_term*** is an ***enum_literal_reference_expressio***n, then:

  — The ***characteristic_basis_set***'s **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is "*AbstractDiscreteSetMeasurementSystem*"

  — The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined above; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT

If ***logical_set*** is an ***enum_literal_set***, then:

- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then:

  a. The ***selected_entity_characteristic_reference*** is a CHARACTERISTIC whose **type** must either be or realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".

b. The ***enum_literal_set***'s ***enumeration_type_reference*** must match the **name** of the ENUMERATED in the MEASUREMENT defined in a.

c. Each ***enumeration_literal_reference*** in ***enum_literal_set*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED defined in b. If an ENUMERATIONCONSTRAINT is specified for the MEASUREMENT defined in a, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT.

- If ***predicate_term*** is a ***subquery***, then:

  a. The ***subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".

  b. The ***enum_literal_set***'s ***enumeration_type_reference*** must match the **name** of the ENUMERATED in the MEASUREMENT defined in a.

  c. Each ***enumeration_literal_reference*** in ***enum_literal_set*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED defined in b. If an ENUMERATIONCONSTRAINT is specified for the MEASUREMENT defined in a, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT.

- ***predicate_term*** must not be an ***enum_literal_reference_expression***

logical_set

// No contextually relevant rules for instances of this term.

characteristic_basis_set

All of the characteristic_basis' **type**s must be the same.

set_compare_predicate

There must be one and only one CHARACTERISTIC in that ***compare_set subquery***'s ***projected_characteristic_list***, and all instances of that CHARACTERISTIC's associated data must be scalar.

If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then it must be a CHARACTERISTIC whose **type** is the same as the ***compare_set subquery***'s projected CHARACTERISTIC's **type**.

If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***compare_set subquery***'s projected CHARACTERISTIC's **type.**

If ***predicate_term*** is an ***enum_literal_reference_expression***:

1. The ***compare_set subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".

2. The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in 1; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT.

3. The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***.

compare_set

// No contextually relevant rules for instances of this term.

compare_operator

// No contextually relevant rules for instances of this term.

set_compare_quantifier

// No contextually relevant rules for instances of this term.

exists_predicate

The ***subquery***'s ***projected_characteristic_list*** must be ***all_characteristics***.

predicate_term

// No contextually relevant rules for instances of this term.

characteristic_basis

If a ***characteristic_basis*** is a ***selected_entity_characteristic_referenc***e, then it must be a CHARACTERISTIC whose:

- **type** is not an ENTITY
- **lowerBound** and **upperBound** is 1
- Associated data is scalar

If a ***characteristic_basis*** is a ***subquery***, then:

- There must be one and only one CHARACTERISTIC in its ***projected_characteristic_list***
- That CHARACTERISTIC's l**owerBound** and **upperBound** must be 1
- That CHARACTERISTIC's associated data must be scalar

subquery

An ***order_by_clause*** must not be specified.

characteristic_reference

// No contextually relevant rules for instances of this term.

entity_type_reference

// No contextually relevant rules for instances of this term.

enum_literal_set

Each ***enumeration_literal_reference*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED whose **name** is ***enumeration_type_reference***.

All ***enumeration_literal_reference***s must be unique.

enum_literal_reference_expression

The ***enumeration_literal_reference*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED whose **name** is ***enumeration_type_reference***.

enumeration_type_reference

// No contextually relevant rules for instances of this term.

enumeration_literal_reference

// No contextually relevant rules for instances of this term.

selected_entity_alias

// No contextually relevant rules for instances of this term.

projected_characteristic_alias

// No contextually relevant rules for instances of this term.

query_identifier

A ***query_identifier*** is a valid String as defined by OCL constraint datamodel::Element::isValidIdentifier specified in Section 8.1: OCL Constraint Helper Methods.

7 Metamodel Specification (EMOF)

This chapter defines an Essential Meta-Object Facility (EMOF) metamodel for the UDDL in Extensible Markup Language (XML) Metadata Interchange (XMI). This chapter is the normative description of the metamodel and in the case of any discrepancy between this and Chapter 4, the authoritative description is in this chapter.

```
<emof:Package xmi:version="2.0" xmi:id="datamodel" name="datamodel"
uri="http://www.opengroup.us/datamodel/1.0"
xmlns:emof="http://schema.omg.org/spec/MOF/2.0/emof.xml"
xmlns:xmi="http://www.omg.org/XMI">
  <ownedType xmi:type="emof:Class" xmi:id="datamodel.DataModel" name="DataModel"
superClass="datamodel.Element">
    <ownedAttribute xmi:id="datamodel.DataModel.cdm" name="cdm" lower="0" upper="*"
type="datamodel.ConceptualDataModel" isComposite="true" />
    <ownedAttribute xmi:id="datamodel.DataModel.ldm" name="ldm" lower="0" upper="*"
type="datamodel.LogicalDataModel" isComposite="true" />
    <ownedAttribute xmi:id="datamodel.DataModel.pdm" name="pdm" lower="0" upper="*"
type="datamodel.PlatformDataModel" isComposite="true" />
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="datamodel.Element" name="Element"
isAbstract="true">
    <ownedAttribute xmi:id="datamodel.Element.name" name="name" isOrdered="true"
default="">
      <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
    </ownedAttribute>
    <ownedAttribute xmi:id="datamodel.Element.description" name="description"
isOrdered="true" default="">
      <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
    </ownedAttribute>
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="datamodel.ConceptualDataModel"
name="ConceptualDataModel" superClass="datamodel.Element">
    <ownedAttribute xmi:id="datamodel.ConceptualDataModel.element" name="element"
lower="0" upper="*" type="datamodel.conceptual.Element" isComposite="true" />
    <ownedAttribute xmi:id="datamodel.ConceptualDataModel.cdm" name="cdm" lower="0"
upper="*" type="datamodel.ConceptualDataModel" isComposite="true" />
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="datamodel.LogicalDataModel"
name="LogicalDataModel" superClass="datamodel.Element">
    <ownedAttribute xmi:id="datamodel.LogicalDataModel.element" name="element" lower="0"
upper="*" type="datamodel.logical.Element" isComposite="true" />
    <ownedAttribute xmi:id="datamodel.LogicalDataModel.ldm" name="ldm" lower="0"
upper="*" type="datamodel.LogicalDataModel" isComposite="true" />
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="datamodel.PlatformDataModel"
name="PlatformDataModel" superClass="datamodel.Element">
    <ownedAttribute xmi:id="datamodel.PlatformDataModel.element" name="element" lower="0"
upper="*" type="datamodel.platform.Element" isComposite="true" />
    <ownedAttribute xmi:id="datamodel.PlatformDataModel.pdm" name="pdm" lower="0"
upper="*" type="datamodel.PlatformDataModel" isComposite="true" />
  </ownedType>
  <nestedPackage xmi:id="datamodel.conceptual" name="conceptual"
uri="http://www.opengroup.us/datamodel/conceptual/1.0">
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Element" name="Element"
isAbstract="true" superClass="datamodel.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.ComposableElement"
name="ComposableElement" isAbstract="true" superClass="datamodel.conceptual.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.BasisElement"
name="BasisElement" isAbstract="true" superClass="datamodel.conceptual.ComposableElement"
/>
```

```
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.BasisEntity"
name="BasisEntity" superClass="datamodel.conceptual.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Domain" name="Domain"
superClass="datamodel.conceptual.Element">
      <ownedAttribute xmi:id="datamodel.conceptual.Domain.basisEntity" name="basisEntity"
lower="0" upper="*" type="datamodel.conceptual.BasisEntity" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Observable"
name="Observable" superClass="datamodel.conceptual.BasisElement" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Characteristic"
name="Characteristic" isAbstract="true">
      <ownedAttribute xmi:id="datamodel.conceptual.Characteristic.rolename"
name="rolename" isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.Characteristic.lowerBound"
name="lowerBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.Characteristic.upperBound"
name="upperBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.Characteristic.specializes"
name="specializes" isOrdered="true" lower="0" type="datamodel.conceptual.Characteristic"
/>
      <ownedAttribute xmi:id="datamodel.conceptual.Characteristic.description"
name="description" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Entity" name="Entity"
superClass="datamodel.conceptual.ComposableElement">
      <ownedAttribute xmi:id="datamodel.conceptual.Entity.composition" name="composition"
lower="0" upper="*" type="datamodel.conceptual.Composition" isComposite="true" />
      <ownedAttribute xmi:id="datamodel.conceptual.Entity.specializes" name="specializes"
lower="0" type="datamodel.conceptual.Entity" />
      <ownedAttribute xmi:id="datamodel.conceptual.Entity.basisEntity" name="basisEntity"
lower="0" upper="*" type="datamodel.conceptual.BasisEntity" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Composition"
name="Composition" superClass="datamodel.conceptual.Characteristic">
      <ownedAttribute xmi:id="datamodel.conceptual.Composition.type" name="type"
isOrdered="true" type="datamodel.conceptual.ComposableElement" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Association"
name="Association" superClass="datamodel.conceptual.Entity">
      <ownedAttribute xmi:id="datamodel.conceptual.Association.participant"
name="participant" lower="0" upper="*" type="datamodel.conceptual.Participant"
isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Participant"
name="Participant" superClass="datamodel.conceptual.Characteristic">
      <ownedAttribute xmi:id="datamodel.conceptual.Participant.type" name="type"
isOrdered="true" type="datamodel.conceptual.Entity" />
      <ownedAttribute xmi:id="datamodel.conceptual.Participant.sourceLowerBound"
name="sourceLowerBound" isOrdered="true" default="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.Participant.sourceUpperBound"
name="sourceUpperBound" isOrdered="true" default="-1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.Participant.path" name="path"
isOrdered="true" lower="0" type="datamodel.conceptual.PathNode" isComposite="true" />
```

```
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.PathNode"
name="PathNode" isAbstract="true">
      <ownedAttribute xmi:id="datamodel.conceptual.PathNode.node" name="node"
isOrdered="true" lower="0" type="datamodel.conceptual.PathNode" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.ParticipantPathNode"
name="ParticipantPathNode" superClass="datamodel.conceptual.PathNode">
      <ownedAttribute
xmi:id="datamodel.conceptual.ParticipantPathNode.projectedParticipant"
name="projectedParticipant" isOrdered="true" type="datamodel.conceptual.Participant" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.CharacteristicPathNode"
name="CharacteristicPathNode" superClass="datamodel.conceptual.PathNode">
      <ownedAttribute
xmi:id="datamodel.conceptual.CharacteristicPathNode.projectedCharacteristic"
name="projectedCharacteristic" isOrdered="true"
type="datamodel.conceptual.Characteristic" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.View" name="View"
isAbstract="true" superClass="datamodel.conceptual.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.Query" name="Query"
superClass="datamodel.conceptual.View">
      <ownedAttribute xmi:id="datamodel.conceptual.Query.specification"
name="specification" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.CompositeQuery"
name="CompositeQuery" superClass="datamodel.conceptual.Element
datamodel.conceptual.View">
      <ownedAttribute xmi:id="datamodel.conceptual.CompositeQuery.composition"
name="composition" lower="2" upper="*" type="datamodel.conceptual.QueryComposition"
isComposite="true" />
      <ownedAttribute xmi:id="datamodel.conceptual.CompositeQuery.isUnion" name="isUnion"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Boolean" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.conceptual.QueryComposition"
name="QueryComposition">
      <ownedAttribute xmi:id="datamodel.conceptual.QueryComposition.rolename"
name="rolename" isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.conceptual.QueryComposition.type" name="type"
isOrdered="true" type="datamodel.conceptual.View" />
    </ownedType>
  </nestedPackage>
  <nestedPackage xmi:id="datamodel.logical" name="logical"
uri="http://www.opengroup.us/datamodel/logical/1.0">
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Element" name="Element"
isAbstract="true" superClass="datamodel.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ConvertibleElement"
name="ConvertibleElement" isAbstract="true" superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Unit" name="Unit"
superClass="datamodel.logical.ConvertibleElement" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Conversion"
name="Conversion" superClass="datamodel.logical.Element">
      <ownedAttribute xmi:id="datamodel.logical.Conversion.destination"
name="destination" isOrdered="true" type="datamodel.logical.ConvertibleElement" />
      <ownedAttribute xmi:id="datamodel.logical.Conversion.source" name="source"
isOrdered="true" type="datamodel.logical.ConvertibleElement" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.AffineConversion"
name="AffineConversion" superClass="datamodel.logical.Conversion">
      <ownedAttribute xmi:id="datamodel.logical.AffineConversion.conversionFactor"
name="conversionFactor" isOrdered="true">
```

```
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.AffineConversion.offset" name="offset"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ValueType"
name="ValueType" isAbstract="true" superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.String" name="String"
superClass="datamodel.logical.ValueType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Character"
name="Character" superClass="datamodel.logical.ValueType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Boolean" name="Boolean"
superClass="datamodel.logical.ValueType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Numeric" name="Numeric"
isAbstract="true" superClass="datamodel.logical.ValueType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Integer" name="Integer"
superClass="datamodel.logical.Numeric" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Natural" name="Natural"
superClass="datamodel.logical.Numeric" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Real" name="Real"
superClass="datamodel.logical.Numeric" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.NonNegativeReal"
name="NonNegativeReal" superClass="datamodel.logical.Numeric" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Enumerated"
name="Enumerated" superClass="datamodel.logical.ValueType">
      <ownedAttribute xmi:id="datamodel.logical.Enumerated.label" name="label"
isOrdered="true" upper="*" type="datamodel.logical.EnumerationLabel" isComposite="true"
/>
      <ownedAttribute xmi:id="datamodel.logical.Enumerated.standardReference"
name="standardReference" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.EnumerationLabel"
name="EnumerationLabel" superClass="datamodel.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.CoordinateSystem"
name="CoordinateSystem" superClass="datamodel.logical.Element">
      <ownedAttribute xmi:id="datamodel.logical.CoordinateSystem.axis" name="axis"
upper="*" type="datamodel.logical.CoordinateSystemAxis" />
      <ownedAttribute
xmi:id="datamodel.logical.CoordinateSystem.axisRelationshipDescription"
name="axisRelationshipDescription" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.CoordinateSystem.angleEquation"
name="angleEquation" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.CoordinateSystem.distanceEquation"
name="distanceEquation" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.CoordinateSystemAxis"
name="CoordinateSystemAxis" superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.AbstractMeasurementSystem"
name="AbstractMeasurementSystem" isAbstract="true" superClass="datamodel.logical.Element"
/>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.StandardMeasurementSystem"
name="StandardMeasurementSystem"
superClass="datamodel.logical.AbstractMeasurementSystem">
```

```
      <ownedAttribute
xmi:id="datamodel.logical.StandardMeasurementSystem.referenceStandard"
name="referenceStandard" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Landmark" name="Landmark"
superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementSystem"
name="MeasurementSystem" superClass="datamodel.logical.AbstractMeasurementSystem">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystem.measurementSystemAxis"
name="measurementSystemAxis" upper="*" type="datamodel.logical.MeasurementSystemAxis" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystem.coordinateSystem"
name="coordinateSystem" type="datamodel.logical.CoordinateSystem" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystem.referencePoint"
name="referencePoint" lower="0" upper="*" type="datamodel.logical.ReferencePoint"
isComposite="true" />
      <ownedAttribute
xmi:id="datamodel.logical.MeasurementSystem.externalStandardReference"
name="externalStandardReference" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystem.orientation"
name="orientation" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystem.constraint"
name="constraint" isOrdered="true" lower="0" upper="*"
type="datamodel.logical.MeasurementConstraint" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementSystemAxis"
name="MeasurementSystemAxis" superClass="datamodel.logical.Element">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystemAxis.axis" name="axis"
type="datamodel.logical.CoordinateSystemAxis" />
      <ownedAttribute
xmi:id="datamodel.logical.MeasurementSystemAxis.defaultValueTypeUnit"
name="defaultValueTypeUnit" upper="*" type="datamodel.logical.ValueTypeUnit" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystemAxis.constraint"
name="constraint" isOrdered="true" lower="0" upper="*"
type="datamodel.logical.MeasurementConstraint" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ReferencePoint"
name="ReferencePoint" superClass="datamodel.Element">
      <ownedAttribute xmi:id="datamodel.logical.ReferencePoint.referencePointPart"
name="referencePointPart" upper="*" type="datamodel.logical.ReferencePointPart"
isComposite="true" />
      <ownedAttribute xmi:id="datamodel.logical.ReferencePoint.landmark" name="landmark"
isOrdered="true" type="datamodel.logical.Landmark" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ReferencePointPart"
name="ReferencePointPart">
      <ownedAttribute xmi:id="datamodel.logical.ReferencePointPart.axis" name="axis"
isOrdered="true" lower="0" type="datamodel.logical.MeasurementSystemAxis" />
      <ownedAttribute xmi:id="datamodel.logical.ReferencePointPart.value" name="value"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.ReferencePointPart.valueTypeUnit"
name="valueTypeUnit" isOrdered="true" lower="0" type="datamodel.logical.ValueTypeUnit" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ValueTypeUnit"
name="ValueTypeUnit" superClass="datamodel.logical.Element
datamodel.logical.AbstractMeasurement">
      <ownedAttribute xmi:id="datamodel.logical.ValueTypeUnit.unit" name="unit"
isOrdered="true" type="datamodel.logical.Unit" />
      <ownedAttribute xmi:id="datamodel.logical.ValueTypeUnit.valueType" name="valueType"
isOrdered="true" type="datamodel.logical.ValueType" />
```

```
      <ownedAttribute xmi:id="datamodel.logical.ValueTypeUnit.constraint"
name="constraint" isOrdered="true" lower="0" type="datamodel.logical.Constraint"
isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Constraint"
name="Constraint" superClass="datamodel.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.IntegerConstraint"
name="IntegerConstraint" isAbstract="true" superClass="datamodel.logical.Constraint" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.IntegerRangeConstraint"
name="IntegerRangeConstraint" superClass="datamodel.logical.IntegerConstraint">
      <ownedAttribute xmi:id="datamodel.logical.IntegerRangeConstraint.lowerBound"
name="lowerBound" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.IntegerRangeConstraint.upperBound"
name="upperBound" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.RealConstraint"
name="RealConstraint" isAbstract="true" superClass="datamodel.logical.Constraint" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.RealRangeConstraint"
name="RealRangeConstraint" superClass="datamodel.logical.RealConstraint">
      <ownedAttribute xmi:id="datamodel.logical.RealRangeConstraint.lowerBound"
name="lowerBound" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.RealRangeConstraint.upperBound"
name="upperBound" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.RealRangeConstraint.lowerBoundInclusive"
name="lowerBoundInclusive" isOrdered="true" default="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Boolean" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.RealRangeConstraint.upperBoundInclusive"
name="upperBoundInclusive" isOrdered="true" default="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Boolean" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.StringConstraint"
name="StringConstraint" isAbstract="true" superClass="datamodel.logical.Constraint" />
    <ownedType xmi:type="emof:Class"
xmi:id="datamodel.logical.RegularExpressionConstraint" name="RegularExpressionConstraint"
superClass="datamodel.logical.StringConstraint">
      <ownedAttribute xmi:id="datamodel.logical.RegularExpressionConstraint.expression"
name="expression" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class"
xmi:id="datamodel.logical.FixedLengthStringConstraint" name="FixedLengthStringConstraint"
superClass="datamodel.logical.StringConstraint">
      <ownedAttribute xmi:id="datamodel.logical.FixedLengthStringConstraint.length"
name="length" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.EnumerationConstraint"
name="EnumerationConstraint" superClass="datamodel.logical.Constraint">
      <ownedAttribute xmi:id="datamodel.logical.EnumerationConstraint.allowedValue"
name="allowedValue" lower="0" upper="*" type="datamodel.logical.EnumerationLabel" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementConstraint"
name="MeasurementConstraint">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementConstraint.constraintText"
name="constraintText" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class"
xmi:id="datamodel.logical.MeasurementSystemConversion" name="MeasurementSystemConversion"
superClass="datamodel.logical.Element">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystemConversion.source"
name="source" type="datamodel.logical.MeasurementSystem" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystemConversion.target"
name="target" type="datamodel.logical.MeasurementSystem" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementSystemConversion.equation"
name="equation" isOrdered="true" upper="*">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute
xmi:id="datamodel.logical.MeasurementSystemConversion.conversionLossDescription"
name="conversionLossDescription" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.AbstractMeasurement"
name="AbstractMeasurement" isAbstract="true" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Measurement"
name="Measurement" superClass="datamodel.logical.ComposableElement
datamodel.logical.AbstractMeasurement">
      <ownedAttribute xmi:id="datamodel.logical.Measurement.constraint" name="constraint"
isOrdered="true" lower="0" upper="*" type="datamodel.logical.MeasurementConstraint"
isComposite="true" />
      <ownedAttribute xmi:id="datamodel.logical.Measurement.measurementAxis"
name="measurementAxis" lower="0" upper="*" type="datamodel.logical.MeasurementAxis" />
      <ownedAttribute xmi:id="datamodel.logical.Measurement.measurementSystem"
name="measurementSystem" isOrdered="true"
type="datamodel.logical.AbstractMeasurementSystem" />
      <ownedAttribute xmi:id="datamodel.logical.Measurement.realizes" name="realizes"
isOrdered="true" type="datamodel.conceptual.Observable" />
      <ownedAttribute xmi:id="datamodel.logical.Measurement.attribute" name="attribute"
lower="0" upper="*" type="datamodel.logical.MeasurementAttribute" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementAxis"
name="MeasurementAxis" superClass="datamodel.logical.Element
datamodel.logical.AbstractMeasurement">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAxis.valueTypeUnit"
name="valueTypeUnit" lower="0" upper="*" type="datamodel.logical.ValueTypeUnit" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAxis.measurementSystemAxis"
name="measurementSystemAxis" isOrdered="true"
type="datamodel.logical.MeasurementSystemAxis" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAxis.constraint"
name="constraint" isOrdered="true" lower="0" upper="*"
type="datamodel.logical.MeasurementConstraint" isComposite="true" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAxis.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.conceptual.Observable" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementAttribute"
name="MeasurementAttribute">
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAttribute.type" name="type"
isOrdered="true" type="datamodel.logical.Measurement" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementAttribute.rolename"
name="rolename" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.MeasurementConversion"
name="MeasurementConversion" superClass="datamodel.logical.Element">
```

```
      <ownedAttribute xmi:id="datamodel.logical.MeasurementConversion.equation"
name="equation" isOrdered="true" upper="*">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute
xmi:id="datamodel.logical.MeasurementConversion.conversionLossDescription"
name="conversionLossDescription" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.MeasurementConversion.source"
name="source" isOrdered="true" type="datamodel.logical.Measurement" />
      <ownedAttribute xmi:id="datamodel.logical.MeasurementConversion.target"
name="target" isOrdered="true" type="datamodel.logical.Measurement" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ComposableElement"
name="ComposableElement" isAbstract="true" superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Characteristic"
name="Characteristic" isAbstract="true">
      <ownedAttribute xmi:id="datamodel.logical.Characteristic.rolename" name="rolename"
isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.Characteristic.lowerBound"
name="lowerBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.Characteristic.upperBound"
name="upperBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.Characteristic.specializes"
name="specializes" isOrdered="true" lower="0" type="datamodel.logical.Characteristic" />
      <ownedAttribute xmi:id="datamodel.logical.Characteristic.description"
name="description" isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Entity" name="Entity"
superClass="datamodel.logical.ComposableElement">
      <ownedAttribute xmi:id="datamodel.logical.Entity.composition" name="composition"
lower="0" upper="*" type="datamodel.logical.Composition" isComposite="true" />
      <ownedAttribute xmi:id="datamodel.logical.Entity.realizes" name="realizes"
isOrdered="true" type="datamodel.conceptual.Entity" />
      <ownedAttribute xmi:id="datamodel.logical.Entity.specializes" name="specializes"
lower="0" type="datamodel.logical.Entity" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Composition"
name="Composition" superClass="datamodel.logical.Characteristic">
      <ownedAttribute xmi:id="datamodel.logical.Composition.type" name="type"
isOrdered="true" type="datamodel.logical.ComposableElement" />
      <ownedAttribute xmi:id="datamodel.logical.Composition.realizes" name="realizes"
isOrdered="true" type="datamodel.conceptual.Composition" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Association"
name="Association" superClass="datamodel.logical.Entity">
      <ownedAttribute xmi:id="datamodel.logical.Association.participant"
name="participant" lower="0" upper="*" type="datamodel.logical.Participant"
isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Participant"
name="Participant" superClass="datamodel.logical.Characteristic">
      <ownedAttribute xmi:id="datamodel.logical.Participant.type" name="type"
isOrdered="true" type="datamodel.logical.Entity" />
      <ownedAttribute xmi:id="datamodel.logical.Participant.realizes" name="realizes"
isOrdered="true" type="datamodel.conceptual.Participant" />
```

```
      <ownedAttribute xmi:id="datamodel.logical.Participant.sourceLowerBound"
name="sourceLowerBound" isOrdered="true" default="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.Participant.sourceUpperBound"
name="sourceUpperBound" isOrdered="true" default="-1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.Participant.path" name="path"
isOrdered="true" lower="0" type="datamodel.logical.PathNode" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.PathNode" name="PathNode"
isAbstract="true">
      <ownedAttribute xmi:id="datamodel.logical.PathNode.node" name="node"
isOrdered="true" lower="0" type="datamodel.logical.PathNode" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.ParticipantPathNode"
name="ParticipantPathNode" superClass="datamodel.logical.PathNode">
      <ownedAttribute xmi:id="datamodel.logical.ParticipantPathNode.projectedParticipant"
name="projectedParticipant" isOrdered="true" type="datamodel.logical.Participant" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.CharacteristicPathNode"
name="CharacteristicPathNode" superClass="datamodel.logical.PathNode">
      <ownedAttribute
xmi:id="datamodel.logical.CharacteristicPathNode.projectedCharacteristic"
name="projectedCharacteristic" isOrdered="true" type="datamodel.logical.Characteristic"
/>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.View" name="View"
isAbstract="true" superClass="datamodel.logical.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.Query" name="Query"
superClass="datamodel.logical.View">
      <ownedAttribute xmi:id="datamodel.logical.Query.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.conceptual.Query" />
      <ownedAttribute xmi:id="datamodel.logical.Query.specification" name="specification"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.CompositeQuery"
name="CompositeQuery" superClass="datamodel.logical.Element datamodel.logical.View">
      <ownedAttribute xmi:id="datamodel.logical.CompositeQuery.composition"
name="composition" lower="2" upper="*" type="datamodel.logical.QueryComposition"
isComposite="true" />
      <ownedAttribute xmi:id="datamodel.logical.CompositeQuery.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.conceptual.CompositeQuery" />
      <ownedAttribute xmi:id="datamodel.logical.CompositeQuery.isUnion" name="isUnion"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Boolean" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.logical.QueryComposition"
name="QueryComposition">
      <ownedAttribute xmi:id="datamodel.logical.QueryComposition.realizes"
name="realizes" isOrdered="true" lower="0" type="datamodel.conceptual.QueryComposition"
/>
      <ownedAttribute xmi:id="datamodel.logical.QueryComposition.rolename"
name="rolename" isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.logical.QueryComposition.type" name="type"
isOrdered="true" type="datamodel.logical.View" />
    </ownedType>
  </nestedPackage>
  <nestedPackage xmi:id="datamodel.platform" name="platform"
uri="http://www.opengroup.us/datamodel/platform/1.0">
```

```
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Element" name="Element"
isAbstract="true" superClass="datamodel.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.ComposableElement"
name="ComposableElement" isAbstract="true" superClass="datamodel.platform.Element" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.PlatformDataType"
name="PlatformDataType" isAbstract="true"
superClass="datamodel.platform.ComposableElement">
      <ownedAttribute xmi:id="datamodel.platform.PlatformDataType.realizes"
name="realizes" isOrdered="true" type="datamodel.logical.AbstractMeasurement" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Primitive"
name="Primitive" isAbstract="true" superClass="datamodel.platform.PlatformDataType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Boolean" name="Boolean"
superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Octet" name="Octet"
superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.CharType" name="CharType"
isAbstract="true" superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Char" name="Char"
superClass="datamodel.platform.CharType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.StringType"
name="StringType" isAbstract="true" superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.String" name="String"
superClass="datamodel.platform.StringType" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.BoundedString"
name="BoundedString" superClass="datamodel.platform.StringType">
      <ownedAttribute xmi:id="datamodel.platform.BoundedString.maxLength"
name="maxLength" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.CharArray"
name="CharArray" superClass="datamodel.platform.StringType">
      <ownedAttribute xmi:id="datamodel.platform.CharArray.length" name="length"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Enumeration"
name="Enumeration" superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Number" name="Number"
isAbstract="true" superClass="datamodel.platform.Primitive" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Integer" name="Integer"
isAbstract="true" superClass="datamodel.platform.Number" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Short" name="Short"
superClass="datamodel.platform.Integer" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Long" name="Long"
superClass="datamodel.platform.Integer" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.LongLong" name="LongLong"
superClass="datamodel.platform.Integer" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Real" name="Real"
isAbstract="true" superClass="datamodel.platform.Number" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Double" name="Double"
superClass="datamodel.platform.Real" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.LongDouble"
name="LongDouble" superClass="datamodel.platform.Real" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Float" name="Float"
superClass="datamodel.platform.Real" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Fixed" name="Fixed"
superClass="datamodel.platform.Real">
      <ownedAttribute xmi:id="datamodel.platform.Fixed.digits" name="digits"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Fixed.scale" name="scale"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
```

```
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.UnsignedInteger"
name="UnsignedInteger" isAbstract="true" superClass="datamodel.platform.Integer" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.UShort" name="UShort"
superClass="datamodel.platform.UnsignedInteger" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.ULong" name="ULong"
superClass="datamodel.platform.UnsignedInteger" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.ULongLong"
name="ULongLong" superClass="datamodel.platform.UnsignedInteger" />
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Sequence" name="Sequence"
superClass="datamodel.platform.Primitive">
      <ownedAttribute xmi:id="datamodel.platform.Sequence.maxSize" name="maxSize"
isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Array" name="Array"
superClass="datamodel.platform.Primitive">
      <ownedAttribute xmi:id="datamodel.platform.Array.size" name="size" isOrdered="true"
lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Struct" name="Struct"
superClass="datamodel.platform.PlatformDataType">
      <ownedAttribute xmi:id="datamodel.platform.Struct.member" name="member"
isOrdered="true" lower="2" upper="*" type="datamodel.platform.StructMember"
isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.StructMember"
name="StructMember">
      <ownedAttribute xmi:id="datamodel.platform.StructMember.type" name="type"
isOrdered="true" type="datamodel.platform.PlatformDataType" />
      <ownedAttribute xmi:id="datamodel.platform.StructMember.rolename" name="rolename"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.StructMember.precision" name="precision"
isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.StructMember.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.logical.MeasurementAttribute" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Characteristic"
name="Characteristic" isAbstract="true">
      <ownedAttribute xmi:id="datamodel.platform.Characteristic.rolename" name="rolename"
isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Characteristic.upperBound"
name="upperBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Characteristic.lowerBound"
name="lowerBound" isOrdered="true" default="1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Characteristic.specializes"
name="specializes" isOrdered="true" lower="0" type="datamodel.platform.Characteristic" />
      <ownedAttribute xmi:id="datamodel.platform.Characteristic.description"
name="description" isOrdered="true" lower="0">
```

```
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Entity" name="Entity"
superClass="datamodel.platform.ComposableElement">
      <ownedAttribute xmi:id="datamodel.platform.Entity.composition" name="composition"
isOrdered="true" lower="0" upper="*" type="datamodel.platform.Composition"
isComposite="true" />
      <ownedAttribute xmi:id="datamodel.platform.Entity.realizes" name="realizes"
isOrdered="true" type="datamodel.logical.Entity" />
      <ownedAttribute xmi:id="datamodel.platform.Entity.specializes" name="specializes"
isOrdered="true" lower="0" type="datamodel.platform.Entity" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Composition"
name="Composition" superClass="datamodel.platform.Characteristic">
      <ownedAttribute xmi:id="datamodel.platform.Composition.type" name="type"
isOrdered="true" type="datamodel.platform.ComposableElement" />
      <ownedAttribute xmi:id="datamodel.platform.Composition.realizes" name="realizes"
isOrdered="true" type="datamodel.logical.Composition" />
      <ownedAttribute xmi:id="datamodel.platform.Composition.precision" name="precision"
isOrdered="true" lower="0">
        <type xmi:type="emof:PrimitiveType"
href="http://www.eclipse.org/emf/2002/Ecore.emof#ecore.EFloat" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Association"
name="Association" superClass="datamodel.platform.Entity">
      <ownedAttribute xmi:id="datamodel.platform.Association.participant"
name="participant" isOrdered="true" lower="0" upper="*"
type="datamodel.platform.Participant" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Participant"
name="Participant" superClass="datamodel.platform.Characteristic">
      <ownedAttribute xmi:id="datamodel.platform.Participant.type" name="type"
isOrdered="true" type="datamodel.platform.Entity" />
      <ownedAttribute xmi:id="datamodel.platform.Participant.realizes" name="realizes"
isOrdered="true" type="datamodel.logical.Participant" />
      <ownedAttribute xmi:id="datamodel.platform.Participant.sourceLowerBound"
name="sourceLowerBound" isOrdered="true" default="0">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Participant.sourceUpperBound"
name="sourceUpperBound" isOrdered="true" default="-1">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Integer" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.Participant.path" name="path"
isOrdered="true" lower="0" type="datamodel.platform.PathNode" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.PathNode" name="PathNode"
isAbstract="true">
      <ownedAttribute xmi:id="datamodel.platform.PathNode.node" name="node"
isOrdered="true" lower="0" type="datamodel.platform.PathNode" isComposite="true" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.ParticipantPathNode"
name="ParticipantPathNode" superClass="datamodel.platform.PathNode">
      <ownedAttribute
xmi:id="datamodel.platform.ParticipantPathNode.projectedParticipant"
name="projectedParticipant" isOrdered="true" type="datamodel.platform.Participant" />
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.CharacteristicPathNode"
name="CharacteristicPathNode" superClass="datamodel.platform.PathNode">
      <ownedAttribute
xmi:id="datamodel.platform.CharacteristicPathNode.projectedCharacteristic"
name="projectedCharacteristic" isOrdered="true" type="datamodel.platform.Characteristic"
/>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.View" name="View"
isAbstract="true" superClass="datamodel.platform.Element" />
```

```
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.Query" name="Query"
superClass="datamodel.platform.Element datamodel.platform.View">
      <ownedAttribute xmi:id="datamodel.platform.Query.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.logical.Query" />
      <ownedAttribute xmi:id="datamodel.platform.Query.specification"
name="specification" isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.CompositeQuery"
name="CompositeQuery" superClass="datamodel.platform.Element datamodel.platform.View">
      <ownedAttribute xmi:id="datamodel.platform.CompositeQuery.composition"
name="composition" isOrdered="true" lower="2" upper="*"
type="datamodel.platform.QueryComposition" isComposite="true" />
      <ownedAttribute xmi:id="datamodel.platform.CompositeQuery.realizes" name="realizes"
isOrdered="true" lower="0" type="datamodel.logical.CompositeQuery" />
      <ownedAttribute xmi:id="datamodel.platform.CompositeQuery.isUnion" name="isUnion"
isOrdered="true">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#Boolean" />
      </ownedAttribute>
    </ownedType>
    <ownedType xmi:type="emof:Class" xmi:id="datamodel.platform.QueryComposition"
name="QueryComposition">
      <ownedAttribute xmi:id="datamodel.platform.QueryComposition.rolename"
name="rolename" isOrdered="true" default="">
        <type xmi:type="emof:PrimitiveType"
href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String" />
      </ownedAttribute>
      <ownedAttribute xmi:id="datamodel.platform.QueryComposition.realizes"
name="realizes" isOrdered="true" lower="0" type="datamodel.logical.QueryComposition" />
      <ownedAttribute xmi:id="datamodel.platform.QueryComposition.type" name="type"
isOrdered="true" type="datamodel.platform.View" />
    </ownedType>
  </nestedPackage>
</emof:Package>
```

8 Metamodel Constraints (OCL)

This chapter specifies the UDDL metamodel constraints.

The constraints are specified using the OMG Object Constraint Language (OCL), Version 2.4.

8.1 OCL Constraint Helper Methods

```
package datamodel

  context Element
    /*
     * Helper method that determines if a string is a valid identifier.
     * An identifier is valid if it consists of alphanumeric characters.
     */
    static def: isValidIdentifier(str : String) : Boolean =
      str.size() > 0 and
      str.replaceAll('[a-zA-Z][_a-zA-Z0-9]*', '').size() = 0

endpackage
```

8.2 OCL Constraints for datamodel Package

```
package datamodel

  context Element
    /*
     * The name of an Element is a valid identifier.
     */
    inv nameIsValidIdentifier:
      Element::isValidIdentifier(self.name)

    /*
     * The following elements have a non-empty description:
     *    - Observable
     *    - Unit
     *    - Landmark
     *    - ReferencePoint
     *    - MeasurementSystem
     *    - MeasurementSystemAxis
     *    - CoordinateSystem
     *    - CoordinateSystemAxis
     *    - MeasurementSystemConversion
     *    - Boolean
     *    - Character
     *    - Numeric
     *    - Integer
     *    - Natural
     *    - NonNegativeReal
     *    - Real
     *    - String
     */
    inv nonEmptyDescription:
      (self.oclIsTypeOf(conceptual::Observable) or
       self.oclIsTypeOf(logical::Unit) or
       self.oclIsTypeOf(logical::Landmark) or
       self.oclIsTypeOf(logical::ReferencePoint) or
       self.oclIsTypeOf(logical::MeasurementSystem) or
```

```
        self.oclIsTypeOf(logical::MeasurementSystemAxis) or
        self.oclIsTypeOf(logical::CoordinateSystem) or
        self.oclIsTypeOf(logical::CoordinateSystemAxis) or
        self.oclIsTypeOf(logical::MeasurementSystemConversion) or
        self.oclIsTypeOf(logical::Boolean) or
        self.oclIsTypeOf(logical::Character) or
        self.oclIsTypeOf(logical::Numeric) or
        self.oclIsTypeOf(logical::Integer) or
        self.oclIsTypeOf(logical::Natural) or
        self.oclIsTypeOf(logical::NonNegativeReal) or
        self.oclIsTypeOf(logical::Real) or
        self.oclIsTypeOf(logical::String))
          implies
        self.description.size() > 0

  context DataModel
    /*
     * Every Element in an DataModel has a unique name.
     */
    inv hasUniqueName:
      let children : Bag(String)=
        self.cdm->collect(name.toLowerCase())->union(
        self.pdm->collect(name.toLowerCase())->union(
        self.ldm->collect(name.toLowerCase()))) in

      children->size() = children->asSet()->size()


endpackage
```

8.3 OCL Constraints for datamodel::conceptual Package

```
package datamodel::conceptual

  context Element
    /*
     * Every Conceptual Element has a unique name.
     */
    inv hasUniqueName:
      not Element.allInstances()->excluding(self)
                                 ->collect(name.toLowerCase())
                                 ->includes(self.name.toLowerCase())

  context ComposableElement
    /*
     * Helper method that determines if a ComposableElement is a
     * specialization of another ComposableElement.
     */
    def: isSpecializationOf(ce : ComposableElement) : Boolean =
      self.oclIsKindOf(Entity) and
      ce.oclIsKindOf(Entity) and
      self.oclAsType(Entity).specializes->closure(specializes)->includes(ce)

  context Entity
    /*
     * Helper method that gets the Characteristics contained in an Entity.
     */
    def: getLocalCharacteristics() : Set(Characteristic) =
      if self.oclIsTypeOf(Association) then
        self.composition
          ->union(self.oclAsType(Association).participant)
          ->oclAsType(Set(Characteristic))
      else
        self.composition->oclAsType(Set(Characteristic))
      endif

    /*
     * Helper method that gets the Characteristics of an Entity,
     * including those from specialized Entities.
```

```
 */
def: getAllCharacteristics() : Set(Characteristic) =
  let allCharacteristics : Set(Characteristic) =
    self->closure(specializes)
        ->collect(getLocalCharacteristics())
        ->asSet() in

  -- get all characteristics that have been specialized
  let specializedCharacteristics : Set(Characteristic) =
    allCharacteristics->collect(specializes)
                      ->asSet() in

   -- return all characteristics that have not been specialized
   allCharacteristics - specializedCharacteristics

/*
 * Helper method that gets the identity of a conceptual Entity.
 */
def: getEntityIdentity() : Bag(OclAny) =
  self.getAllCharacteristics()
    ->collectNested(getIdentityContribution())
    ->union(self.getBasisEntities())

/*
 * Helper method to retrieve the BasisEntities of an Entity,
 * including those from specialized Entities.
 */
def: getBasisEntities() : Bag(BasisEntity) =
  self->closure(specializes)
      ->collect(basisEntity)

/*
 * Helper method that determines whether or not
 * an Entity is part of a specialization cycle.
 */
def: isPartOfSpecializationCycle() : Boolean =
  self.specializes->closure(specializes)->includes(self)

/*
 * A Characteristic's rolename is unique within an Entity.
 */
inv characteristicsHaveUniqueRolenames:
  self.getAllCharacteristics()->isUnique(rolename)

/*
 * If Entity A' specializes Entity A, all characteristics
 * in A' specialize nothing, specialize characteristics from A,
 * or specialize characteristics from an Entity that is a generalization of
 * A. (If A' does not specialize, none of its characteristics specialize.)
 */
inv specializingCharacteristicsConsistent:
  if self.specializes = null then
    self.getLocalCharacteristics()
        ->select(specializes <> null)
        ->isEmpty()
  else
    self.getLocalCharacteristics()
        ->select(specializes <> null)
        ->forAll(c | self.specializes
                         ->closure(specializes)
                         ->collect(getLocalCharacteristics())
                         ->exists(sc | c.specializes = sc))
   endif

/*
 * An Entity is not a specialization of itself.
 */
inv noCyclesInSpecialization:
  not isPartOfSpecializationCycle()

/*
```

```
    * An Entity has at least one Characteristic defined
    * locally (not through generalization).
    */
   inv hasAtLeastOneLocalCharacteristic:
     self.getLocalCharacteristics()->size() >= 1

 context Association
   /*
    * An Association has at least two Participants.
    */
    inv hasAtLeastTwoParticipants:
      self.getAllCharacteristics()
          ->selectByKind(Participant)
          ->size() >= 2

 context Characteristic
   /*
    * Helper method that gets the rolename of a Characteristic.
    */
   def: getRolename() : String =
     if self.oclIsKindOf(Composition) then
       self.oclAsType(Composition).rolename
     else
       self.oclAsType(Participant).getRolename()
     endif

   /*
    * Helper method that gets the type of a Characteristic.
    */
   def: getType() : ComposableElement =
     if self.oclIsTypeOf(Composition) then
       self.oclAsType(Composition).type
     else
       self.oclAsType(Participant).getResolvedType()
     endif

   /*
    * Helper method that gets the contribution a Characteristic makes
    * to an Entity's uniqueness.
    */
   def: getIdentityContribution() : Sequence(OclAny) =
     if self.oclIsTypeOf(Composition) then
       self.oclAsType(Composition).getIdentityContribution()
     else
       self.oclAsType(Participant).getIdentityContribution()
     endif

   /*
    * Helper method that determines if one upper bound is
    * more restrictive than another.
    */
   def: upperBound_LTE(testUpperBound : Integer,
                       baselineUpperBound : Integer) : Boolean =
   if baselineUpperBound = -1 then
     testUpperBound >= baselineUpperBound
   else
     testUpperBound <= baselineUpperBound
   endif

   /*
    * Helper method that determines if a Characteristic is a
    * specialization of another Characteristic.
    */
   def: isSpecializationOf(cp : Characteristic) : Boolean =
     self->closure(specializes)
         ->exists(sc : Characteristic |
                  sc = cp.oclAsType(Characteristic))

   /*
    * The rolename of a Characteristic is a valid identifier.
    */
```

```
  inv rolenameIsValidIdentifier:
    self.getRolename() <> null implies
    Element::isValidIdentifier(self.getRolename())

  /*
   * A Characteristic's lowerBound is less than or equal to its upperBound,
   * unless its upperBound is -1.
   */
  inv lowerBound_LTE_UpperBound:
    self.upperBound <> -1 implies self.lowerBound <= self.upperBound

  /*
   * A Characteristic's upperBound is equal to -1 or greater than or equal to 1.
   */
  inv upperBoundValid:
    self.upperBound = -1 or self.upperBound >= 1

  /*
   * A Characteristic's lowerBound is greater than or equal to zero.
   */
  inv lowerBoundValid:
    self.lowerBound  >= 0

  /*
   * A Characteristic is specialized once in a generalization hierarchy.
   */
  inv specializeCharacteristicOnce:
    self.specializes <> null implies

    (let containingEntity
       = Entity.allInstances()->any(e | e.getLocalCharacteristics()
                                           ->includes(self)) in

     containingEntity.specializes <> null implies
     containingEntity.getAllCharacteristics()
                     ->reject(c | c = self)
                     ->forAll(c | c.specializes <> self.specializes)
    )

context Composition
  /*
   * Helper method that gets the contribution a Composition makes
   * to an Entity's uniqueness (type and multiplicity).
   */
   def: getIdentityContribution() : Sequence(OclAny) =
     Sequence{self.type,
              self.lowerBound,
              self.upperBound}

  /*
   * If a Composition specializes, its multiplicity is
   * at least as restrictive as the Composition it specializes.
   */
  inv multiplicityConsistentWithSpecialization:
    self.specializes <> null implies
    self.lowerBound >= self.specializes.lowerBound and
    upperBound_LTE(self.upperBound, self.specializes.upperBound)

  /*
   * If a Composition specializes, it specializes a Composition.
   * If Composition "A" specializes Composition "B",
   * then A's type is B's type or a specialization of B's type.
   */
  inv typeConsistentWithSpecialization:
    self.specializes <> null implies
    self.specializes.oclIsTypeOf(Composition) and
    (self.type = self.specializes.oclAsType(Composition).type or
     self.type
         .isSpecializationOf(self.specializes.oclAsType(Composition).type))

   /*
```

```
     * If a Composition specializes, its type or multiplicity is
     * different from the Composition it specializes.
     */
    inv specializationDistinct:
      self.specializes <> null and
      self.specializes.oclIsTypeOf(Composition) implies
      self.type <> self.specializes.getType() or
      self.lowerBound <> self.specializes.lowerBound or
      self.upperBound <> self.specializes.upperBound

context Participant
  /*
   * Helper method that gets a Participant's PathNode sequence.
   */
  def: getPathSequence() : OrderedSet(PathNode) =
    self.path
        ->asOrderedSet()
          ->closure(pn : PathNode |
                    let projectedParticipantPath =
                      if pn.projectsParticipant() then
                         pn.projectedParticipant().path
                      else
                        null
                      endif in

                    OrderedSet{projectedParticipantPath,
                               pn.node}
                      ->reject(oclIsUndefined()))

  /*
   * Helper method that determines if a Participant's
   * path sequence contains a cycle.
   */
  def: hasCycleInPath() : Boolean =
    self.getPathSequence()
        ->collect(getProjectedCharacteristic())
        ->includes(self)

  /*
   * Helper method that gets the element projected by a Participant.
   * Returns a ComposableElement.
   */
  def: getResolvedType() : ComposableElement =
    if self.hasCycleInPath() then
      null
    else if self.path = null then
      self.type
    else
      self.getPathSequence()->last().getNodeType()
    endif
    endif

  /*
   * Helper method that gets the rolename of a Participant.
   * (A Participant's rolename is either projected from a
   * characteristic or defined directly on the Participant.)
   */
  def: getRolename() : String =
    if self.rolename.size() > 0 then
      self.rolename
    else if self.path <> null and
            self.getPathSequence()->last()
                .oclIsTypeOf(CharacteristicPathNode) then
      self.getPathSequence()->last()
          .oclAsType(CharacteristicPathNode)
          .projectedCharacteristic.getRolename()
    else
      null
    endif
    endif
```

```
/*
 * Helper method that gets the contribution a Participant makes
 * to an Entity's uniqueness (type, path sequence, and multiplicity).
 */
 def: getIdentityContribution() : Sequence(OclAny) =
   Sequence{self.type,
            self.getPathSequence()->collect(getProjectedCharacteristic()),
            self.lowerBound,
            self.upperBound}

/*
 * Helper method that determines if a Participant's path sequence
 * is "equal" to another path sequence.
 * (A PathNode sequence "A" is "equal" a sequence "B" if
 * the projected element of each PathNode in A is the same
 * projected element of the corresponding PathNode in B.)
 */
def: pathIsEqualTo(otherPath : OrderedSet(PathNode))
       : Boolean =
  let path = self.getPathSequence() in

  path->size() = otherPath->size() and
  Sequence{1..path->size()}->forAll(index : Integer |
    let pathNode = path->at(index) in
    let specializedPathNode = otherPath->at(index) in
    pathNode.getProjectedCharacteristic()
      = specializedPathNode.getProjectedCharacteristic())

/*
 * Helper method that determines if a Participant's path sequence
 * correctly "specializes" another path sequence.
 * (A PathNode sequence "A" "specializes" a sequence "B" if
 * the projected element of each PathNode in A specializes the
 * projected element of the corresponding PathNode in B.)
 */
def: pathIsSpecializationOf(specializedPath : OrderedSet(PathNode))
       : Boolean =
  let path = self.getPathSequence() in

  path->size() > 0 and
  path->size() = specializedPath->size() and
  Sequence{1..path->size()}->forAll(index : Integer |
    let pathNode = path->at(index) in
    let specializedPathNode = specializedPath->at(index) in
    pathNode.getProjectedCharacteristic()
            .isSpecializationOf(specializedPathNode
                                  .getProjectedCharacteristic()))

/*
 * A Participant has a rolename, either projected from a
 * characteristic or defined directly on the Participant.
 */
inv rolenameDefined:
  self.getRolename() <> null

/*
 * If a Participant has a path sequence, the first PathNode in the sequence
 * is resolvable from the type of the Participant.
 */
inv pathNodeResolvable:
  self.path <> null implies
  self.path.isResolvableFromEntity(self.type)

/*
 * If a Participant specializes, its multiplicity is
 * at least as restrictive as the Participant it specializes.
 */
inv multiplicityConsistentWithSpecialization:
  self.specializes <> null and
  self.specializes.oclIsTypeOf(Participant) implies
  let specializedParticipant = self.specializes.oclAsType(Participant) in
```

```
    self.lowerBound >= specializedParticipant.lowerBound and
    self.sourceLowerBound >= specializedParticipant.sourceLowerBound and
    upperBound_LTE(self.upperBound, specializedParticipant.upperBound) and
    upperBound_LTE(self.sourceUpperBound,
                   specializedParticipant.sourceUpperBound)

  /*
   * If a Participant specializes, it specializes a Participant.
   * If Participant "A" specializes Participant "B",
   * then A's type is the same or a specialization of B's type,
   * and A's PathNode sequence is "equal to" or "specializes" B's
   * PathNode sequence (see "pathIsEqual" and
   * "pathIsSpecializationOf" helper methods).
   */
  inv typeConsistentWithSpecialization:
    self.specializes <> null and
    self.specializes.oclIsTypeOf(Participant) implies
    let specializedParticipant = self.specializes.oclAsType(Participant) in

    self.specializes.oclIsTypeOf(Participant)
      and
     (self.type = specializedParticipant.type or
      self.type.isSpecializationOf(specializedParticipant.type))
      and
     (self.pathIsEqualTo(specializedParticipant.getPathSequence()) or
      self.pathIsSpecializationOf(specializedParticipant.getPathSequence()))

   /*
    * If a Participant specializes, its type, PathNode sequence,
    * or multiplicity is different from the Participant it specializes.
    */
   inv specializationDistinct:
     self.specializes <> null and
     self.specializes.oclIsTypeOf(Participant)
       implies
     let specializedParticipant = self.specializes.oclAsType(Participant) in
     self.type <> self.specializes.getType() or
     self.pathIsSpecializationOf(specializedParticipant.getPathSequence()) or
     self.lowerBound <> self.specializes.lowerBound or
     self.upperBound <> self.specializes.upperBound or
     self.sourceLowerBound <> specializedParticipant.oclAsType(Participant)
                                                  .sourceLowerBound or
     self.sourceUpperBound <> specializedParticipant.oclAsType(Participant)
                                                  .sourceUpperBound

context CompositeQuery
  /*
   * A QueryComposition's rolename is unique within a CompositeQuery.
   */
  inv compositionsHaveUniqueRolenames:
    self.composition->collect(rolename)
                    ->isUnique(rn | rn)

  /*
   * A CompositeQuery does not compose itself.
   */
  inv noCyclesInConstruction:
    let composedQueries = self.composition
                              ->collect(type)
                              ->selectByKind(CompositeQuery)
                              ->closure(composition
                                        ->collect(type)
                                        ->selectByKind(CompositeQuery)) in

    not composedQueries->includes(self)

  /*
   * A CompositeQuery does not compose the same View more than once.
   */
  inv viewComposedOnce:
    self.composition->collect(type)->isUnique(view | view)

context QueryComposition
  /*
   * The rolename of a QueryComposition is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    Element::isValidIdentifier(self.rolename)

context PathNode
  /*
   * Helper method that gets the Characteristic projected by a PathNode.
   */
  def: getProjectedCharacteristic() : Characteristic =
    if self.oclIsTypeOf(CharacteristicPathNode) then
      self.oclAsType(CharacteristicPathNode).projectedCharacteristic
    else -- ParticipantPathNode
      self.oclAsType(ParticipantPathNode).projectedParticipant
    endif

  /*
   * Helper method that determines if a PathNode projects a Participant.
   */
  def: projectsParticipant() : Boolean =
    self.oclIsTypeOf(CharacteristicPathNode) and
    self.oclAsType(CharacteristicPathNode)
        .projectedCharacteristic
        .oclIsTypeOf(Participant)

  /*
   * Helper method that gets the Participant projected by a PathNode.
   * Returns null if no Participant is projected.
   */
  def: projectedParticipant() : Participant =
    let pp = self.oclAsType(CharacteristicPathNode)
                 .projectedCharacteristic
                 .oclAsType(Participant) in
    if not pp.oclIsInvalid() then
      pp
    else
      null
    endif

  /*
   * Helper method that gets the "node type" of a PathNode. For a
   * CharacteristicPathNode, the node type is the type of the projected
   * characteristic. For a ParticipantPathNode, the node type is the
   * Association containing the projected Participant.
   * Returns a ComposableElement.
   */
  def: getNodeType() : ComposableElement =
    if self.oclIsTypeOf(CharacteristicPathNode) then
      self.oclAsType(CharacteristicPathNode)
          .projectedCharacteristic
          .getType()
    else
      -- get Association that contains projectedCharacteristic
      Association.allInstances()
        ->select(participant->includes(self.oclAsType(ParticipantPathNode)
                                           .projectedParticipant))
        ->any(true)
    endif

  /*
   * Helper method that determines if a PathNode is resolvable from a
   * given Entity.
   */
  def: isResolvableFromEntity(entity : Entity) : Boolean =
    if self.oclIsTypeOf(CharacteristicPathNode) then
      entity.getAllCharacteristics()
            ->includes(self.oclAsType(CharacteristicPathNode)
                              .projectedCharacteristic)
      else
        entity = self.oclAsType(ParticipantPathNode)
                        .projectedParticipant
                        .type
      endif

    /*
     * Helper method that determines if the referenced characteristic
     * is a collection.
     */
    def: projectsAcrossCollection() : Boolean =
      if self.oclIsTypeOf(CharacteristicPathNode) then
        let projectedCharacteristic = self.oclAsType(CharacteristicPathNode)
                                        .projectedCharacteristic in
        projectedCharacteristic.oclIsKindOf(Characteristic) and
        (projectedCharacteristic.oclAsType(Characteristic).lowerBound <> 1 or
        projectedCharacteristic.oclAsType(Characteristic).upperBound <> 1)
      else -- ParticipantPathNode
        let projectedParticipant = self.oclAsType(ParticipantPathNode)
                                      .projectedParticipant in
        projectedParticipant.sourceLowerBound <> 1 or
        projectedParticipant.sourceUpperBound <> 1
      endif

    /*
     * If a CharacteristicPathNode projects a Characteristic with upper or
     * lower bounds not equal to 1, then it is the end of a PathNode sequence.
     * If a ParticipantPathNode projects a Participant with source lower or
     * upper bounds not equal to 1, then it is the end of a PathNode sequence.
     */
    inv noProjectionAcrossCollection:
      self.projectsAcrossCollection() implies
      self.node = null

    /*
     * If a PathNode "A" is not the last in a path sequence, the next PathNode
     * in the sequence is resolvable from the "node type" of A.
     */
    inv pathNodeResolvable:
      self.node <> null implies
      if self.getNodeType() = null then
        false
      else
        self.getNodeType().oclIsKindOf(Entity) and
        self.node.isResolvableFromEntity(self.getNodeType().oclAsType(Entity))
      endif


endpackage
```

8.4 OCL Constraints for datamodel::logical Package

```
package datamodel::logical

  context Element
    /*
     * Every Logical Element has a unique name,
     * with the exception of Constraints.
     */
    inv hasUniqueName:
      not self.oclIsTypeOf(Constraint) implies
      not Element.allInstances()->excluding(self)
                                ->collect(name.toLowerCase())
                                ->includes(self.name.toLowerCase())

  context Entity
    /*
     * Helper method that gets the Characteristics contained in an Entity.
```

```
 */
def: getLocalCharacteristics() : Set(Characteristic) =
  if self.oclIsTypeOf(Association) then
    self.composition
      ->union(self.oclAsType(Association).participant)
      ->oclAsType(Set(Characteristic))
  else
    self.composition->oclAsType(Set(Characteristic))
  endif

/*
 * Helper method that gets the Characteristics of an Entity,
 * including those from specialized Entities.
 */
def: getAllCharacteristics() : Set(Characteristic) =
  let allCharacteristics : Set(Characteristic) =
    self->closure(specializes)
        ->collect(getLocalCharacteristics())
        ->asSet() in

  -- get all characteristics that have been specialized
  let specializedCharacteristics : Set(Characteristic) =
    allCharacteristics->collect(specializes)
                      ->asSet() in

   -- return all characteristics that have not been specialized
   allCharacteristics - specializedCharacteristics

/*
 * A Characteristic's rolename is unique within an Entity.
 */
inv characteristicsHaveUniqueRolenames:
  self.getAllCharacteristics()->isUnique(rolename)

/*
 * Compositions in a logical Entity realize Compositions in
 * the conceptual Entity that the logical Entity realizes.
 */
inv compositionsConsistentWithRealization:
  self.composition
      ->collect(realizes)
      ->forAll(c | self.realizes.composition->exists(c2 | c = c2))

/*
 * An Entity does not contain two Compositions that realize the same
 * conceptual Composition unless their types are different Measurements
 * and their multiplicities are equal.
 */
inv realizedCompositionsHaveDifferentTypes:
  self.composition->forAll(c1, c2 | c1 <> c2 and
                                    c1.realizes = c2.realizes
                                      implies
                                    c1.type.oclIsTypeOf(Measurement) and
                                    c2.type.oclIsTypeOf(Measurement) and
                                    c1.type <> c2.type and
                                    c1.lowerBound = c2.lowerBound and
                                    c1.upperBound = c2.upperBound)

/*
 * If an Entity specializes, its specialization is
 * consistent with its realization's specialization.
 */
inv specializationConsistentWithRealization:
  self.specializes <> null implies
  self.specializes.realizes = self.realizes.specializes

/*
 * An Entity has at least one Characteristic defined
 * locally (not through generalization), unless the
 * Entity is in the "middle" of a generalization hierarchy.
 */
```

```
  inv hasAtLeastOneLocalCharacteristic:
    let inMiddleOfGeneralizationHierarchy =
      self.specializes <> null and
      Entity.allInstances()->collect(specializes)
                           ->includes(self) in

    not inMiddleOfGeneralizationHierarchy
      implies
    self.getLocalCharacteristics()->size() >= 1

context Association
  /*
   * Participants in a logical Association realize Participants in
   * the conceptual Association that the logical Association realizes.
   */
  inv participantsConsistentWithRealization:
    self.participant
        ->collect(realizes)
        ->forAll(ae | self.realizes.oclAsType(datamodel::conceptual
                                                            ::Association)
                          .participant
                          ->exists(ae2 | ae = ae2))

  /*
   * Participants in an Association realize unique Participants.
   */
  inv participantsRealizeUniquely:
    self.participant->forAll(p1, p2 | p1 <> p2 implies
                                  p1.realizes <> p2.realizes)

context Characteristic
  /*
   * Helper method that gets the rolename of a Characteristic.
   */
  def: getRolename() : String =
    if self.oclIsKindOf(Composition) then
      self.oclAsType(Composition).rolename
    else
      self.oclAsType(Participant).getRolename()
    endif

  /*
   * Helper method that gets the type of a Characteristic.
   */
  def: getType() : ComposableElement =
    if self.oclIsTypeOf(Composition) then
      self.oclAsType(Composition).type
    else
      self.oclAsType(Participant).getResolvedType()
    endif

  /*
   * Helper method that gets the conceptual Characteristic a
   * logical Characteristic realizes.
   */
  def: getRealizes() : datamodel::conceptual::Characteristic =
    if self.oclIsTypeOf(Composition) then
      self.oclAsType(Composition).realizes
    else
      self.oclAsType(Participant).realizes
    endif

  /*
   * The rolename of a Characteristic is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    self.getRolename() <> null implies
    Element::isValidIdentifier(self.getRolename())

  /*
   * A Characteristic's lowerBound is less than or equal to its upperBound,
   * unless its upperBound is -1.
   */
  inv lowerBound_LTE_UpperBound:
    self.upperBound <> -1 implies self.lowerBound <= self.upperBound

  /*
   * A Characteristic's upperBound is equal to -1 or greater than or equal to 1.
   */
  inv upperBoundValid:
    self.upperBound = -1 or self.upperBound >= 1

  /*
   * If a Characteristic specializes, its specialization is
   * consistent with its realization's specialization.
   */
  inv specializationConsistentWithRealization:
    self.specializes <> null implies
    self.specializes.getRealizes() = self.getRealizes().specializes

context Composition
  /*
   * A Composition's type is consistent with its realization's type.
   */
  inv typeConsistentWithRealization:
    if self.type.oclIsKindOf(Entity) then
      self.type.oclAsType(Entity).realizes = self.realizes.type
    else
    if self.type.oclIsKindOf(Measurement) then
      self.type.oclAsType(Measurement).realizes = self.realizes.type
    else
      false
    endif
    endif

  /*
   * A Composition's multiplicity is at least as
   * restrictive as the Composition it realizes.
   */
  inv multiplicityConsistentWithRealization:
    self.lowerBound >= self.realizes.lowerBound and
    if self.realizes.upperBound = -1 then
      self.upperBound >= self.realizes.upperBound
    else
      self.upperBound <= self.realizes.upperBound
    endif

  /*
   * A Composition's multiplicity is at least as
   * restrictive as the Composition it specializes.
   */
  inv multiplicityConsistentWithSpecialization:
    self.specializes <> null implies
    self.lowerBound >= self.specializes.lowerBound and
    if self.specializes.upperBound = -1 then
      self.upperBound >= self.specializes.upperBound
    else
      self.upperBound <= self.specializes.upperBound
    endif

context Participant
  /*
   * Helper method that gets a Participant's PathNode sequence.
   */
  def: getPathSequence() : OrderedSet(PathNode) =
      self.path
          ->asOrderedSet()
          ->closure(pn : PathNode |
                    let projectedParticipantPath =
                      if pn.projectsParticipant() then
                         pn.projectedParticipant().path
                      else
```

```
                         null
                       endif in

                     OrderedSet{projectedParticipantPath,
                                pn.node}
                       ->reject(oclIsUndefined()))

/*
 * Helper method that determines if a Participant's
 * path sequence contains a cycle.
 */
def: hasCycleInPath() : Boolean =
  self.getPathSequence()
      ->collect(getProjectedCharacteristic())
      ->includes(self)

/*
 * Helper method that gets the element projected by a Participant.
 * Returns a ComposableElement.
 */
def: getResolvedType() : ComposableElement =
  if self.hasCycleInPath() then
    null
  else if self.path = null then
    self.type
  else
    self.getPathSequence()->last().getNodeType()
  endif
  endif

/*
 * Helper method that gets the rolename of a Participant.
 * (A Participant's rolename is either projected from a
 * characteristic or defined directly on the Participant.)
 */
def: getRolename() : String =
  if self.rolename.size() > 0 then
    self.rolename
  else if self.path <> null and
          self.getPathSequence()->last()
              .oclIsTypeOf(CharacteristicPathNode) then
    self.getPathSequence()->last()
        .oclAsType(CharacteristicPathNode)
        .projectedCharacteristic.getRolename()
  else
    null
  endif
  endif

/*
 * A Participant has a rolename, either projected from a
 * characteristic or defined directly on the Participant.
 */
inv rolenameDefined:
  self.getRolename() <> null

/*
 * If Participant "A" realizes Participant "B",
 * then A's type realizes B's type,
 * and A's PathNode sequence "realizes" B's PathNode sequence.
 * (A PathNode sequence "A" "realizes" a sequence "B" if
 * the projected element of each PathNode in A realizes the
 * projected element of the corresponding PathNode in B.)
 */
inv typeConsistentWithRealization:
  self.type.realizes = self.realizes.type
    and
  self.getPathSequence()->collect(getProjectedCharacteristic()
                                     .getRealizes()) =
  self.realizes.getPathSequence()->collect(getProjectedCharacteristic())
  /*
   * A Participant's multiplicity is at least as
   * restrictive as the Participant it realizes.
   */
  inv multiplicityConsistentWithRealization:
    self.lowerBound >= self.realizes.lowerBound and
    if self.realizes.upperBound = -1 then
      self.upperBound >= self.realizes.upperBound
    else
      self.upperBound <= self.realizes.upperBound
    endif

    and

    self.sourceLowerBound >= self.realizes.sourceLowerBound and
    if self.realizes.sourceUpperBound = -1 then
      self.sourceUpperBound >= self.realizes.sourceUpperBound
    else
      self.sourceUpperBound <= self.realizes.sourceUpperBound
    endif

  /*
   * A Participant's multiplicity is at least as
   * restrictive as the Participant it specializes.
   */
  inv multiplicityConsistentWithSpecialization:
    (self.specializes <> null and
     self.specializes.oclIsTypeOf(Participant))
       implies
    let specializedParticipant = self.specializes.oclAsType(Participant) in

    self.lowerBound >= specializedParticipant.lowerBound and
    if specializedParticipant.upperBound = -1 then
      self.upperBound >= specializedParticipant.upperBound
    else
      self.upperBound <= specializedParticipant.upperBound
    endif

    and

    self.sourceLowerBound >= specializedParticipant.sourceLowerBound and
    if specializedParticipant.sourceUpperBound = -1 then
      self.sourceUpperBound >= specializedParticipant.sourceUpperBound
    else
      self.sourceUpperBound <= specializedParticipant.sourceUpperBound
    endif

context View
  /*
   * Helper method that gets the View realized by a View.
   */
  def: getRealizes() : datamodel::conceptual::View =
    if self.oclIsKindOf(Query) then
      self.oclAsType(Query).realizes
    else
      self.oclAsType(CompositeQuery).realizes
    endif

context CompositeQuery
  /*
   * A QueryComposition's rolename is unique within a CompositeQuery.
   */
  inv compositionsHaveUniqueRolenames:
    self.composition->collect(rolename)
                    ->isUnique(rn | rn)

  /*
   * A CompositeQuery does not compose itself.
   */
  inv noCyclesInConstruction:
    let composedQueries = self.composition
```

```
                                  ->collect(type)
                                  ->selectByKind(CompositeQuery)
                                  ->closure(composition
                                            ->collect(type)
                                            ->selectByKind(CompositeQuery)) in

      not composedQueries->includes(self)

    /*
     * A CompositeQuery does not compose the same View more than once.
     */
    inv viewComposedOnce:
      self.composition->collect(type)->isUnique(view | view)

    /*
     * QueryCompositions in a logical CompositeQuery realize QueryCompositions
     * in the conceptual CompositeQuery that the logical CompositeQuery
     * realizes.
     */
    inv compositionsConsistentWithRealization:
      if self.realizes = null
      then
        self.composition->forAll(c | c.realizes = null)
      else
        self.composition->forAll(c |
            self.realizes.composition->exists(c2 | c.realizes = c2)
        )
      endif

    /*
     *  A CompositeQuery that realizes has the same "isUnion" property
     *  as the CompositeQuery it realizes.
     */
    inv realizationUnionConsistent:
      self.realizes->forAll(realized | self.isUnion = realized.isUnion)

    /*
     * A CompositeQuery does not contain two QueryCompositions that realize the
     * same QueryComposition.
     */
    inv realizedCompositionsHaveDifferentTypes:
      self.realizes <> null implies
      self.composition->forAll(c1, c2 | c1 <> c2 implies
                                        c1.realizes <> c2.realizes)

  context QueryComposition
    /*
     * The rolename of a QueryComposition is a valid identifier.
     */
    inv rolenameIsValidIdentifier:
      Element::isValidIdentifier(self.rolename)

    /*
     * If QueryComposition "A" realizes QueryComposition "B",
     * then A's type realizes B's type.
     */
    inv typeConsistentWithRealization:
      self.realizes <> null implies
      self.type.getRealizes() = self.realizes.type

  context PathNode
    /*
     * Helper method that gets the Characteristic projected by a PathNode.
     */
    def: getProjectedCharacteristic() : Characteristic =
      if self.oclIsTypeOf(CharacteristicPathNode) then
        self.oclAsType(CharacteristicPathNode).projectedCharacteristic
      else -- ParticipantPathNode
        self.oclAsType(ParticipantPathNode).projectedParticipant
      endif
    /*
     * Helper method that determines if a PathNode projects a Participant.
     */
    def: projectsParticipant() : Boolean =
      self.oclIsTypeOf(CharacteristicPathNode) and
      self.oclAsType(CharacteristicPathNode)
          .projectedCharacteristic
          .oclIsTypeOf(Participant)

    /*
     * Helper method that gets the Participant projected by a PathNode.
     * Returns null if no Participant is projected.
     */
    def: projectedParticipant() : Participant =
      let pp = self.oclAsType(CharacteristicPathNode)
                    .projectedCharacteristic
                    .oclAsType(Participant) in
      if not pp.oclIsInvalid() then
        pp
      else
        null
      endif

    /*
     * Helper method that gets the "node type" of a PathNode. For a
     * CharacteristicPathNode, the node type is the type of the projected
     * characteristic. For a ParticipantPathNode, the node type is the
     * Association containing the projected Participant.
     * Returns a ComposableElement.
     */
    def: getNodeType() : ComposableElement =
      if self.oclIsTypeOf(CharacteristicPathNode) then
        self.oclAsType(CharacteristicPathNode)
            .projectedCharacteristic
            .getType()
      else
        -- get Association that contains projectedCharacteristic
        Association.allInstances()
          ->select(participant->includes(self.oclAsType(ParticipantPathNode)
                                              .projectedParticipant))
          ->any(true)
      endif

  context ValueTypeUnit
    /*
     * If a ValueTypeUnit "A" contains an EnumerationConstraint,
     * then A's valueType is an Enumerated, and the constraint's
     * allowedValues are EnumerationLabels from that Enumerated.
     */
    inv appropriateLabelsForEnumeratedConstraint:
      self.constraint <> null and
      self.constraint.oclIsTypeOf(EnumerationConstraint)

      implies

      self.valueType.oclIsTypeOf(Enumerated) and
      self.constraint
          .oclAsType(EnumerationConstraint)
          .allowedValue
          ->forAll(allowedValue | self.valueType
                                       .oclAsType(Enumerated)
                                       .label
                                       ->exists(label | label = allowedValue))

  context ValueType
    /*
     * A ValueType is named the same as its metatype.
     * (e.g. a String is named "String")
     */
    inv nameOfValueTypeMatchesNameOfMetaclass:
      self.oclIsTypeOf(Enumerated) or
      self.name = self.oclType().name

  context FixedLengthStringConstraint
    /*
     * A FixedLengthStringConstraint's length is greater than zero.
     */
    inv nonNegativeLength:
      self.length > 0

  context Enumerated
    /*
     * An EnumerationLabel's name is unique within an Enumerated.
     */
    inv enumerationLabelNameUnique:
      self.label->isUnique(name)

  context MeasurementSystem
    /*
     * Helper method that determines if a MeasurementSystem
     * uses an Enumerated ValueType in any of its axes.
     */
    def: hasAnEnumeratedValueType() : Boolean =
      let valueTypes: Collection(ValueType) =
            self.measurementSystemAxis.defaultValueTypeUnit.valueType in
      valueTypes->exists(vt | vt.oclIsTypeOf(Enumerated))

    /*
     * There is one MeasurementSystem that uses an Enumerated ValueType
     * in any of its axes. Its name is "AbstractDiscreteSetMeasurementSystem",
     * and it has one axis.
     */
    inv onlyOneEnumeratedMeasurementSystem:
      if self.name = 'AbstractDiscreteSetMeasurementSystem' then
        self.hasAnEnumeratedValueType() and
        self.measurementSystemAxis.defaultValueTypeUnit->size() = 1
      else
        not self.hasAnEnumeratedValueType()
      endif

    /*
     * If a MeasurementSystem "A" is based on CoordinateSystem "B",
     * then A and B have the same number of axes,
     * and every MeasurementSystemAxis in A is based on a unique
     * CoordinateSystemAxis in B.
     */
    inv measurementSystemConsistentWithCoordinateSystem:
      self.measurementSystemAxis->collect(axis)
        = coordinateSystem.axis->asBag()

    /*
     * A ReferencePoint in a MeasurementSystem contains ReferencePointParts
     * that use MeasurementSystemAxes used by that MeasurementSystem.
     */
    inv referencePointPartsConsistentWithAxes:
      self.referencePoint->forAll(rp |
        rp.referencePointPart->collect(axis)->forAll(rppAxis |
          rppAxis <> null implies
          self.measurementSystemAxis->exists(msa | msa = rppAxis)
        )
      )

    /*
     * In a MeasurementSystem, each ReferencePoints' parts use the same
     * set of VTUs as the MeasurementSystem's axes.
     */
    inv referencePointPartsCoverAllAxes:
      self.referencePoint->forAll(rp |
        rp.referencePointPart->collect(valueTypeUnit)
          = self.measurementSystemAxis->collect(defaultValueTypeUnit)
        )
    /*
     * If a MeasurementSystem has ReferencePoints, then it has
     * at least as many ReferencePoints as it has axes.
     */
    inv hasSufficientReferencePoints:
      self.referencePoint->notEmpty() implies
      self.referencePoint->size() >= self.measurementSystemAxis->size()

  context ReferencePoint
    /*
     * If two ReferencePointParts in a ReferencePoint refer to the same
     * VTU, then they refer to distinct (non-null) axes.
     */
     inv noAmbiguousVTUReference:
       let allVTUs = self.referencePointPart->collect(valueTypeUnit) in
       let vtusUsedMoreThanOnce = allVTUs->reject(vtu |
                                                  allVTUs->count(vtu) = 1) in

       vtusUsedMoreThanOnce->forAll(vtu |
         let rppsThatUseTheVTU
          = self.referencePointPart
                ->select(rpp |
                         vtusUsedMoreThanOnce->includes(rpp.valueTypeUnit)) in

         rppsThatUseTheVTU->forAll(rpp | rpp.axis <> null) and
         rppsThatUseTheVTU->collect(axis)
                          ->asSet()
                          ->size() = rppsThatUseTheVTU->collect(axis)->size()
       )

  context Measurement
    /*
     * Helper method that determines if a Measurement
     * uses an Enumerated ValueType in any of its axes.
     */
    def: hasAnEnumeratedValueType() : Boolean =
      let valueTypes: Collection(ValueType) =
            self.measurementAxis.valueTypeUnit.valueType in
      valueTypes->exists(vt | vt.oclIsTypeOf(Enumerated))

    /*
     * Helper method that determines if a Measurement is
     * based on a StandardMeasurementSystem.
     */
    def: isStandardMeasurement() : Boolean =
        self.measurementSystem.oclIsTypeOf(StandardMeasurementSystem)

    /*
     * A Measurement that uses an Enumerated ValueType in any of its axes
     * is based on the 'AbstractDiscreteSetMeasurementSystem' MeasurementSystem.
     */
    inv enumeratedMeasurementUsesEnumeratedMeasurementSystem:
      if self.hasAnEnumeratedValueType() then
        self.measurementSystem.name = 'AbstractDiscreteSetMeasurementSystem'
      else
        self.measurementSystem.name <> 'AbstractDiscreteSetMeasurementSystem'
      endif

    /*
     * If a Measurement "A" is based on MeasurementSystem "B",
     * then A and B have the same number of axes,
     * and every MeasurementAxis in A is based on a unique
     * MeasurementSystemAxis in B.
     * If a Measurement is based on a StandardMeasurementSystem,
     * then it has no axes.
     */
    inv measurementConsistentWithMeasurementSystem:
      if self.isStandardMeasurement() then
        self.measurementAxis->isEmpty()
      else
        self.measurementAxis->collect(measurementSystemAxis)
```

```
          = self.measurementSystem.oclAsType(MeasurementSystem)
                .measurementSystemAxis->asBag()
      endif

    /*
     * A Measurement does not use itself as a MeasurementAttribute.
     */
    inv noCyclesInMeasurements:
      not self.attribute.type->closure(attribute.type)->includes(self)

    /*
     * A Measurement's attributes have unique rolenames.
     */
    inv measurementAttributesHaveUniqueRolenames:
      self.attribute->isUnique(rolename)

  context MeasurementAxis
    /*
     * Helper method that gets the ValueTypeUnits used in a MeasurementAxis.
     */
    def: getValueTypeUnits() : Set(ValueTypeUnit) =
      if self.valueTypeUnit->isEmpty() then
        self.measurementSystemAxis.defaultValueTypeUnit
      else
        -- the MeasurementSystem's default ValueTypeUnit is overridden
        self.valueTypeUnit
      endif


endpackage
```

8.5 OCL Constraints for datamodel::platform Package

```
package datamodel::platform

  context Element
    /*
     * All Platform Elements have a unique name.
     */
    inv hasUniqueName:
      not Element.allInstances()->excluding(self)
                                ->collect(name.toLowerCase())
                                ->includes(self.name.toLowerCase())

  context Entity
    /*
     * Helper method that gets the Characteristics contained in an Entity.
     * (Platform Characteristics are ordered with Participants first,
     * then Compositions.)
     */
    def: getLocalCharacteristics() : OrderedSet(Characteristic) =
      if self.oclIsTypeOf(Association) and not self.oclAsType(Association)
                                                  .participant
                                                  ->isEmpty() then
        self.oclAsType(Association)
            .participant
            ->iterate(c : Characteristic;
                      acc : OrderedSet(Characteristic) = self.composition |
                      acc->append(c))
      else
        self.composition
      endif

    /*
     * Helper method that gets the Characteristics of an Entity,
     * including those from specialized Entities.
     * (Platform Characteristics are ordered with the Characteristics
     * of the "top-most" Entity in a generalization hierarchy first,
     * then the "second-top-most", etc.)
```

```
   */
  def: getAllCharacteristics() : OrderedSet(Characteristic) =
    let allCharacteristics : OrderedSet(Characteristic) =
      self->asOrderedSet()
          ->closure(specializes)
          ->collect(getLocalCharacteristics())
          ->asOrderedSet() in

    -- get all characteristics that have been specialized
    let specializedCharacteristics : OrderedSet(Characteristic) =
      allCharacteristics->collect(specializes)
                        ->asOrderedSet() in

     -- return all characteristics that have not been specialized
     specializedCharacteristics->iterate(c : Characteristic;
                                         acc : OrderedSet(Characteristic)
                                             = allCharacteristics |
                                         acc->excluding(c))

  /*
   * A Characteristic's rolename is unique within an Entity.
   */
  inv characteristicsHaveUniqueRolenames:
    self.getAllCharacteristics()->isUnique(rolename)

  /*
   * Compositions in a platform Entity realize Compositions in
   * the logical Entity that the platform Entity realizes.
   */
  inv compositionsConsistentWithRealization:
    self.composition
        ->collect(realizes)
        ->forAll(c | self.realizes.composition->exists(c2 | c = c2))

  /*
   * An Entity does not contain two Compositions that realize the same
   * logical Composition unless their types are different PlatformDataTypes
   * and their multiplicities are equal.
   */
  inv realizedCompositionsHaveDifferentTypes:
    self.composition->forAll(c1, c2 | c1 <> c2 and
                                      c1.realizes = c2.realizes
                                        implies
                                      c1.type.oclIsKindOf(PlatformDataType) and
                                      c2.type.oclIsKindOf(PlatformDataType) and
                                      c1.type <> c2.type and
                                      c1.lowerBound = c2.lowerBound and
                                      c1.upperBound = c2.upperBound)

  /*
   * If an Entity specializes, its specialization is
   * consistent with its realization's specialization.
   */
  inv specializationConsistentWithRealization:
    self.specializes <> null implies
    self.specializes.realizes = self.realizes.specializes

  /*
   * An Entity has at least one Characteristic defined
   * locally (not through generalization), unless the
   * Entity is in the "middle" of a generalization hierarchy.
   */
  inv hasAtLeastOneLocalCharacteristic:
    not (self.specializes <> null
           or
         Entity.allInstances()->collect(specializes)
                              ->includes(self))
      implies
    self.getLocalCharacteristics()->size() >= 1

context Association
```

```
  /*
   * Participants in a platform Association realize Participants in
   * the logical Association that the platform Association realizes.
   */
  inv participantsConsistentWithRealization:
    self.participant
        ->collect(realizes)
        ->forAll(ae | self.realizes.oclAsType(datamodel::logical
                                                             ::Association)
                           .participant
                           ->exists(ae2 | ae = ae2))

  /*
   * Participants in an Association realize unique Participants.
   */
  inv participantsRealizeUniquely:
    self.participant->forAll(p1, p2 | p1 <> p2 implies
                                      p1.realizes <> p2.realizes
    )

context Characteristic
  /*
   * Helper method that gets the rolename of a Characteristic.
   */
  def: getRolename() : String =
    if self.oclIsKindOf(Composition) then
      self.oclAsType(Composition).rolename
    else
      self.oclAsType(Participant).getRolename()
    endif

  /*
   * Helper method that gets the type of a Characteristic.
   */
  def: getType() : ComposableElement =
    if self.oclIsTypeOf(Composition) then
      self.oclAsType(Composition).type
    else
      self.oclAsType(Participant).getResolvedType()
    endif

  /*
   * Helper method that gets the conceptual Characteristic a
   * logical Characteristic realizes.
   */
  def: getRealizes() : datamodel::logical::Characteristic =
    if self.oclIsTypeOf(Composition) then
      self.oclAsType(Composition).realizes
    else
      self.oclAsType(Participant).realizes
    endif

  /*
   * The rolename of a Characteristic is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    self.getRolename() <> null implies
    Element::isValidIdentifier(self.getRolename())

  /*
   * A Characteristic's lowerBound is less than or equal to its upperBound,
   * unless its upperBound is -1.
   */
  inv lowerBound_LTE_UpperBound:
    self.upperBound <> -1 implies self.lowerBound <= self.upperBound

  /*
   * A Characteristic's upperBound is equal to -1 or greater than or equal to 1.
   */
  inv upperBoundValid:
    self.upperBound = -1 or self.upperBound >= 1
  /*
   * If a Characteristic specializes, its specialization is
   * consistent with its realization's specialization.
   */
  inv specializationConsistentWithRealization:
    self.specializes <> null implies
    self.specializes.getRealizes() = self.getRealizes().specializes

context Composition
  /*
   * A Composition's type is consistent with its realization's type.
   */
  inv typeConsistentWithRealization:
    if self.type.oclIsKindOf(Entity) then
      self.type.oclAsType(Entity).realizes = self.realizes.type
    else
    if self.type.oclIsKindOf(PlatformDataType) then
      self.type.oclAsType(PlatformDataType).realizes = self.realizes.type
    else
      false
    endif
    endif

  /*
   * A Composition's multiplicity is at least as
   * restrictive as the Composition it realizes.
   */
  inv multiplicityConsistentWithRealization:
    self.lowerBound >= self.realizes.lowerBound and
    if self.realizes.upperBound = -1 then
      self.upperBound >= self.realizes.upperBound
    else
      self.upperBound <= self.realizes.upperBound
    endif

  /*
   * A Composition's multiplicity is at least as
   * restrictive as the Composition it specializes.
   */
  inv multiplicityConsistentWithSpecialization:
    self.specializes <> null implies
    self.lowerBound >= self.specializes.lowerBound and
    if self.specializes.upperBound = -1 then
      self.upperBound >= self.specializes.upperBound
    else
      self.upperBound <= self.specializes.upperBound
    endif

  /*
   * A Composition whose type is a Number has
   * its precision greater than zero.
   */
  inv composedNumberHasPrecisionSet:
    self.type.oclIsKindOf(Number)
      implies
    self.precision > 0

context Participant
  /*
   * Helper method that gets a Participant's PathNode sequence.
   */
  def: getPathSequence() : OrderedSet(PathNode) =
      self.path
          ->asOrderedSet()
          ->closure(pn : PathNode |
                    let projectedParticipantPath =
                      if pn.projectsParticipant() then
                         pn.projectedParticipant().path
                      else
                        null
                      endif in

                    OrderedSet{projectedParticipantPath,
                               pn.node}
                      ->reject(oclIsUndefined()))

  /*
   * Helper method that determines if a Participant's
   * path sequence contains a cycle.
   */
  def: hasCycleInPath() : Boolean =
    self.getPathSequence()
        ->collect(getProjectedCharacteristic())
        ->includes(self)

  /*
   * Helper method that gets the element projected by a Participant.
   * Returns a ComposableElement.
   */
  def: getResolvedType() : ComposableElement =
    if self.hasCycleInPath() then
      null
    else if self.path = null then
      self.type
    else
      self.getPathSequence()->last().getNodeType()
    endif
    endif

  /*
   * Helper method that gets the rolename of a Participant.
   * (A Participant's rolename is either projected from a
   * characteristic or defined directly on the Participant.)
   */
  def: getRolename() : String =
    if self.rolename.size() > 0 then
      self.rolename
    else if self.path <> null and
            self.getPathSequence()->last()
                .oclIsTypeOf(CharacteristicPathNode) then
      self.getPathSequence()->last()
          .oclAsType(CharacteristicPathNode)
          .projectedCharacteristic.getRolename()
    else
      null
    endif
    endif

  /*
   * A Participant has a rolename, either projected from a
   * characteristic or defined directly on the Participant.
   */
  inv rolenameDefined:
    self.getRolename() <> null

  /*
   * If Participant "A" realizes Participant "B",
   * then A's type realizes B's type,
   * and A's PathNode sequence "realizes" B's PathNode sequence.
   * (A PathNode sequence "A" "realizes" a sequence "B" if
   * the projected element of each PathNode in A realizes the
   * projected element of the corresponding PathNode in B.)
   */
  inv typeConsistentWithRealization:
    self.type.realizes = self.realizes.type
      and
    self.getPathSequence()->collect(getProjectedCharacteristic()
                                     .getRealizes()) =
    self.realizes.getPathSequence()->collect(getProjectedCharacteristic())

  /*
```

```
     * A Participant's multiplicity is at least as
     * restrictive as the Participant it realizes.
     */
    inv multiplicityConsistentWithRealization:
      self.lowerBound >= self.realizes.lowerBound and
      if self.realizes.upperBound = -1 then
        self.upperBound >= self.realizes.upperBound
      else
        self.upperBound <= self.realizes.upperBound
      endif

      and

      self.sourceLowerBound >= self.realizes.sourceLowerBound and
      if self.realizes.sourceUpperBound = -1 then
        self.sourceUpperBound >= self.realizes.sourceUpperBound
      else
        self.sourceUpperBound <= self.realizes.sourceUpperBound
      endif

    /*
     * A Participant's multiplicity is at least as
     * restrictive as the Participant it specializes.
     */
    inv multiplicityConsistentWithSpecialization:
      self.specializes <> null and
      self.specializes.oclIsTypeOf(Participant)

        implies

      let specializedParticipant = self.specializes.oclAsType(Participant) in

      self.lowerBound >= specializedParticipant.lowerBound and
      if specializedParticipant.upperBound = -1 then
        self.upperBound >= specializedParticipant.upperBound
      else
        self.upperBound <= specializedParticipant.upperBound
      endif

      and

      self.sourceLowerBound >= specializedParticipant.sourceLowerBound and
      if specializedParticipant.sourceUpperBound = -1 then
        self.sourceUpperBound >= specializedParticipant.sourceUpperBound
      else
        self.sourceUpperBound <= specializedParticipant.sourceUpperBound
      endif

  context View
    /*
     * Helper method that gets the View realized by a View.
     */
    def: getRealizes() : datamodel::logical::View =
      if self.oclIsKindOf(Query) then
        self.oclAsType(Query).realizes
      else
        self.oclAsType(CompositeQuery).realizes
      endif

  context CompositeQuery
    /*
     * A QueryComposition's rolename is unique within a CompositeQuery.
     */
    inv compositionsHaveUniqueRolenames:
      self.composition->collect(rolename)
                      ->isUnique(rn | rn)

    /*
     * A CompositeQuery does not compose itself.
     */
    inv noCyclesInConstruction:
```

```
    let composedQueries = self.composition
                                ->collect(type)
                                ->selectByKind(CompositeQuery)
                                ->closure(composition
                                          ->collect(type)
                                          ->selectByKind(CompositeQuery)) in

    not composedQueries->includes(self)

  /*
   * A CompositeQuery does not compose the same View more than once.
   */
  inv viewComposedOnce:
    self.composition->collect(type)->isUnique(view | view)

  /*
   * QueryCompositions in a platform CompositeQuery realize QueryCompositions
   * in the logical CompositeQuery that the platform CompositeQuery
   * realizes.
   */
  inv compositionsConsistentWithRealization:
    if self.realizes = null
    then
      self.composition->forAll(c | c.realizes = null)
    else
      self.composition->forAll(c |
          self.realizes.composition->exists(c2 | c.realizes = c2)
      )
    endif

  /*
   *  A CompositeQuery that realizes has the same "isUnion" property
   *  as the CompositeQuery it realizes.
   */
  inv realizationUnionConsistent:
    self.realizes->forAll(realized | self.isUnion = realized.isUnion)

  /*
   * A CompositeQuery does not contain two QueryCompositions that realize the
   * same QueryComposition.
   */
  inv realizedCompositionsHaveDifferentTypes:
    self.realizes <> null implies
    self.composition->forAll(c1, c2 | c1 <> c2 implies
                                  c1.realizes <> c2.realizes)

context QueryComposition
  /*
   * The rolename of a QueryComposition is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    Element::isValidIdentifier(self.rolename)

  /*
   * If QueryComposition "A" realizes QueryComposition "B",
   * then A's type realizes B's type.
   */
  inv typeConsistentWithRealization:
    self.realizes <> null implies
    self.type.getRealizes() = self.realizes.type

context PathNode
  /*
   * Helper method that gets the Characteristic projected by a PathNode.
   */
  def: getProjectedCharacteristic() : Characteristic =
    if self.oclIsTypeOf(CharacteristicPathNode) then
      self.oclAsType(CharacteristicPathNode).projectedCharacteristic
    else -- ParticipantPathNode
      self.oclAsType(ParticipantPathNode).projectedParticipant
    endif
  /*
   * Helper method that determines if a PathNode projects a Participant.
   */
  def: projectsParticipant() : Boolean =
    self.oclIsTypeOf(CharacteristicPathNode) and
    self.oclAsType(CharacteristicPathNode)
        .projectedCharacteristic
        .oclIsTypeOf(Participant)

  /*
   * Helper method that gets the Participant projected by a PathNode.
   * Returns null if no Participant is projected.
   */
  def: projectedParticipant() : Participant =
    let pp = self.oclAsType(CharacteristicPathNode)
                 .projectedCharacteristic
                 .oclAsType(Participant) in
    if not pp.oclIsInvalid() then
      pp
    else
      null
    endif

  /*
   * Helper method that gets the "node type" of a PathNode. For a
   * CharacteristicPathNode, the node type is the type of the projected
   * characteristic. For a ParticipantPathNode, the node type is the
   * Association containing the projected Participant.
   * Returns a ComposableElement.
   */
  def: getNodeType() : ComposableElement =
    if self.oclIsTypeOf(CharacteristicPathNode) then
      self.oclAsType(CharacteristicPathNode)
          .projectedCharacteristic
          .getType()
    else
      -- get Association that contains projectedCharacteristic
      Association.allInstances()
        ->select(participant->includes(self.oclAsType(ParticipantPathNode)
                                           .projectedParticipant))
        ->any(true)
    endif

context PlatformDataType
  /*
   * Helper method that determines if a PlatformDataType realizes a MeasurementAxis.
   */
  def: realizesMeasurementAxis() : Boolean =
    self.realizes.oclIsTypeOf(datamodel::logical::MeasurementAxis)


  /*
   * Helper method that gets the MeasurementAxis realized by an PlatformDataType.
   * Returns null if the PlatformDataType does not realize a MeasurementAxis.
   */
  def: realizedMeasurementAxis() : datamodel::logical
                                                ::MeasurementAxis =
    if self.realizesMeasurementAxis() then
      self.realizes.oclAsType(datamodel::logical::MeasurementAxis)
    else
      null
    endif

  /*
   * Helper method that determines if a PlatformDataType realizes a Measurement.
   */
  def: realizesMeasurement() : Boolean =
    self.realizes.oclIsTypeOf(datamodel::logical::Measurement)

  /*
```

```
 * Helper method that gets the Measurement realized by a PlatformDataType.
 * Returns null if the PlatformDataType does not realize a Measurement.
 */
def: realizedMeasurement() : datamodel::logical::Measurement =
  if self.realizesMeasurement() then
    self.realizes.oclAsType(datamodel::logical::Measurement)
  else
    null
  endif

/*
 * A ValueTypeUnit is realized by a Primitive.
 */
inv vtuRealizedByPrimitive:
  self.realizes.oclIsTypeOf(datamodel::logical::ValueTypeUnit)
    implies
  self.oclIsKindOf(Primitive)

/*
 * An Array or Sequence realizes a
 * Measurement based on a StandardMeasurementSystem.
 */
inv collectionRealizesStandardMeasurement:
  self.oclIsTypeOf(Sequence) or self.oclIsTypeOf(Array)
    implies
  self.realizesMeasurement() and
  self.realizedMeasurement().isStandardMeasurement()

/*
 * If a MeasurementAxis has one ValueTypeUnit (VTU), then
 * it is realized by a Primitive; if it has multiple VTUs, then
 * it is realized by a Struct with one StructMember per VTU.
 * If Struct "A" realizes MeasurementAxis "B",
 * then A has the same number of StructMembers as B has VTUs,
 * and every StructMember in A realizes a unique VTU in V.
 */
inv platformDataTypeConsistentlyRealizesMeasurementAxis:
  self.realizesMeasurementAxis()

    implies

  if self.realizedMeasurementAxis().getValueTypeUnits()->size() = 1 then
    self.oclIsKindOf(Primitive)
  else
    self.oclIsTypeOf(Struct) and
    self.realizedMeasurementAxis().getValueTypeUnits()->asBag()
      = self.oclAsType(Struct).member
                                ->collect(type.realizes)
                                ->asBag()
  endif

/*
 * A Measurement is realized by a Struct with one StructMember
 * per MeasurementAxis. (Each StructMember's type realizes a unique axis
 * in the Measurement; every axis is realized.)
 * There are two exceptions:
 *   - If a Measurement has one axis with one ValueTypeUnit (VTU)
 *      and no MeasurementAttributes, it is realized by a Primitive.
 *   - If a Measurement has one axis with multiple VTUs
 *      and no MeasurementAttributes, it is realized by a Struct
 *      with one StructMember for each VTU in the axis.
 *      (Each StructMember's type realizes a unique VTU in the axis;
 *      every VTU is realized.) Each StructMember's type is consistent
 *      with the type of the VTU it realizes.
 */
inv platformDataTypeConsistentlyRealizesMeasurement:
  self.realizesMeasurement() and
  not self.realizedMeasurement().isStandardMeasurement()

    implies
```

```
      let realizedAxes = self.realizedMeasurement().measurementAxis in

      if realizedAxes->collect(getValueTypeUnits())->size() = 1 and
         self.realizedMeasurement().attribute->size() = 0 then
        self.oclIsKindOf(Primitive)
      else if realizedAxes->size() = 1 and
         self.realizedMeasurement().attribute->size() = 0 then
        realizedAxes->collect(getValueTypeUnits())
          = self.oclAsType(Struct).member->select(realizes = null)
                                            ->collect(type.realizes)
                                            ->asBag()
      else
        realizedAxes->asBag()
          = self.oclAsType(Struct).member->select(realizes = null)
                                            ->collect(type.realizes)
                                            ->asBag()
      endif
      endif

  context Struct
    /*
     * A Measurement with MeasurementAttributes is realized by a Struct
     * with one StructMember per MeasurementAttribute. (Each StructMember
     * (that realizes) realizes a unique attribute in the Measurement;
     * every attribute is realized.)
     */
    inv structMembersConsistentlyRealizeMeasurementAttributes:
      self.realizesMeasurement()

        implies

      self.member->collect(realizes)->reject(oclIsUndefined())->asBag()
       = self.realizedMeasurement().attribute->asBag()

  context StructMember
    /*
     * If a StructMember realizes a MeasurementAttribute, then
     * the StructMember's type is consistent with its realization's type.
     */
    inv typeConsistentWithRealization:
      self.realizes <> null implies
      self.type.oclAsType(PlatformDataType).realizes = self.realizes.type

    /*
     * A StructMember whose type is a Number has
     * a precision greater than zero.
     */
    inv composedNumberHasPrecisionSet:
      self.type <> null and self.type.oclIsKindOf(Number)
        implies
      self.precision > 0


endpackage
```

Glossary

Cardinality

The number of elements in a particular set or other grouping.

For characteristic compositions and participants, the minimum or maximum number of elements making up the multiplicity.

Data Model

An abstraction that describes real-world elements, their properties, and their relationships providing an interoperable means of data exchange.

Multiplicity

The range of the number of elements possible within a set. Each lower bound and upper bound is referred to as cardinality, whereas the range is the multiplicity.

Note: Refer to the metamodel classes "Characteristic" and "Participant" for further details on multiplicity.

Object Constraint Language

The language used to define the additional rules on the user's Data Model.

Path

A means by which the relational context of a participant can be narrowed to a specific characteristic.

Realize (Realization)

The relationship between similar model elements across levels of abstraction.

Acronyms

CDM	Conceptual Data Model
EMOF	Essential Meta-Object Facility
IIoT	Industrial Internet of Things
LDM	Logical Data Model
MOF	Meta-Object Facility
OCD	Interface Control Document
OCL	Object Constraint Language (OMG)
PDM	Platform Data Model
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

Index