Date: November 2009

Reference Metamodel for the EXPRESS Information Modeling Language Specification

Beta 3 (clean)

OMG Document Number: dtc/2009-12-08
Standard document URL: http://www.omg.org/spec/EXPRESS/1.0
Associated Files*:
http://www.omg.org/spec/EXPRESS/20091101
http://www.omg.org/spec/EXPRESS/20091102

---

* Original files: dtc/09-11-03 (cmof.xmi), dtc/09-11-07 (uml2 xmi)

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

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

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

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , IMM™, MOF™, OMG Interface Definition Language (IDL)™, and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

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

Table of Contents

List of Figures

Preface

About the Object Management Group

OMG

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

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

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

OMG Specifications

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

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

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

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

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

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Introduction

Background – the origins of EXPRESS

In 1984, the Standards for Exchange of Product Data (STEP) project was officially created in the International Standards Organization (ISO) as an outgrowth of standardization projects in the United States and France. The objective of the STEP project was to develop standards for the exchange of product information among software tools that supported product engineering. It rapidly came to include support for construction engineering and manufacturing systems engineering as well.

An objective of this project was to specify the information content of a product description in a way that was independent of the form of exchange, so that more than one specific exchange form could be specified, while the semantic equivalences would be retained by reference to the common model. In particular, the project members envisaged the need for both database representations and sequential file structures.

At that time, there were *no* standard information modeling languages, and the languages in common use were purely graphical. In order to specify the relationships between the information model (what we would now call a "platform independent model") and the data implementation model (a "platform specific model"), it was perceived to be a requirement that the information model have a formal text form. Such a form would enable an information model to be processed by a software tool that could generate the corresponding PSM. There being no usable standard, nor any common language, with these characteristics, the STEP project developed and standardized its own information modeling language: EXPRESS.

The information modeling language EXPRESS was standardized in 1994 as Part 11 of the ISO 10303 Standards for the Exchange of Product Data. It was revised in 1999 and in 2004. It was used for every information model in the STEP series, and in 3 other standards series in ISO TC184 (Industrial Data), and for information models in standards developed by other ISO Technical Committees. As of 2005, there were over 300 major information models for manufacturing and construction information that are formally specified in EXPRESS and standardized by ISO. These models, and the EXPRESS language are in wide use in the manufacturing industry, and the exchange models are supported by dozens of software tools.

In the more recent past, in order to make these models useful to an industry in which programmers and modelers are not commonly taught EXPRESS, further ISO projects have been undertaken to produce mappings from EXPRESS to XML Schema (ISO 10303-28) and UML (ISO 10303-25). But each of these mappings was specified entirely in text and targeted version 1 of XML Schema and UML respectively.

The MEXICO project

In 2005, the MEXICO project was created with the objective of applying OMG Model-Driven Architecture (MDA) technologies to the "EXPRESS problem." The project has three components:

- development of a MOF metamodel for the EXPRESS language
- development of a formal (MOF/QVT) mapping from the EXPRESS metamodel to the UML v2 metamodel (thus replacing ISO 10303-25 with a formal and machine-processable specification)
- development of a formal (MOF/QVT) mapping from the EXPRESS metamodel to the metamodel of OWL specified in the OMG Ontology Definition Metamodel

This specification represents the final deliverable of the first project component: the MOF metamodel of EXPRESS. Results of the other project components will be published separately.

Taken together, these elements will permit automatic generation of UML models that faithfully represent the content of any ISO standard model formulated in EXPRESS. Similarly, these elements will permit automatic generation of faithful renditions of those models in OWL, which will enable them to be used as draft ontologies and tested for logical consistency (and consistency with other models) using Semantic Web tooling. In this way, the knowledge captured in the many standard EXPRESS models can be made available and usable for 21st century technologies and practitioners.

Development of the EXPRESS metamodel

The MEXICO project has developed a complete metamodel of the EXPRESS language and tooling to support it.

NIST developed an EXPRESS compiler that accepts an EXPRESS schema (model) and produces XMI (v1.1) that corresponds to the metamodel. NIST is currently reworking that compiler to produce M1 instances of the complete CMOF model herein specified in the XMI 2.1.x form.

Fraunhofer IPK developed a MOF implementation of the metamodel and used the output of the NIST tool to populate the MOF database for a set of EXPRESS schemas.

Fraunhofer developed additional tooling to implement a mapping from the MOF population to UML (v1.4) following the guidance in ISO 10303-25. This was a first step toward the goals of the second MEXICO project component.

Eurostep developed tooling to map a subset of the metamodel to OWL. This was a first step toward the goals of the third MEXICO project component. Further work in this area is continuing with Eurostep and other partners.

At the same time, a number of other tool vendors who support the EXPRESS modeling community have developed independent internal models of EXPRESS and mappings to various languages, including UML, OWL, and XML Schema. (Many of them are listed as “supporters” of this specification.) We all agree that the time has come to standardize an XMI representation of EXPRESS, so as to permit these tools to interoperate around a common representation.

This specification is the metamodel of the semantics of the EXPRESS language that was developed and tested in the MEXICO project. It represents completion of the first subproject in the MEXICO trilogy. And it has value in its own right to other EXPRESS tool developers. For this reason, we are bringing it to OMG for standardization.

Participants in the metamodel development activity include four “technical experts” who participated in the development of the EXPRESS language itself. It also includes technical experts who were principal developers of the Part 25 (mapping to UML) and Part 28 (mapping to XML Schema) standards. This expertise gives us confidence that the metamodel is faithful to the semantic intent of the EXPRESS standard.

To be clear about what has been “tested”: For the MEXICO proof-of-concept tooling, all the tools were built to a version of the metamodel known as version 060615f. Only the NIST tool dealt with the concepts “internal to” Algorithms: Variables, Statements, and ActualTypes. Parallel work at the New University of Lisbon (UNINOVA) developed tooling for an elaborate model of Statements. The major change in this specification is the integration of the UNINOVA model, and related changes and repairs to the Algorithms Package.

Further, to satisfy the current level of technical expectations in OMG, the MOF 1.4 version has been modified to a CMOF version in this version. Several errors have been discovered and they are corrected in this version.

Acknowledgements

This specification is derived in part from early work on the development of a metamodel of EXPRESS (that was itself specified in EXPRESS) by Prof. Donald Sanderson of East Tennessee State University, Dr. Philip Spiby of Eurostep , Dr. Markus Maier of PDTEC, and Dr. Peter Wilson of Boeing Corporation (now retired).

Every organization listed as a submitter or supporter has made some technical contribution to this specification.

2 Scope and Purpose

This specification is a metamodel for the EXPRESS information modeling language, as defined by ISO 10303-11.2:2004, Product data exchange – EXPRESS Language Reference Manual. It includes all elements of the language.

The metamodel conforms to the OMG Complete Meta-Object Facility (CMOF) specification, version 2.0.

The metamodel captures the meaning of the EXPRESS syntactic constructs, not the syntactic constructs themselves. It differs from an abstract syntactic model of the language when either:

- the same syntax is used with different semantics in different contexts, or
- the syntax is more complex than the semantic content it expresses.

Some attributes of concepts in the metamodel serve only to permit the EXPRESS syntactic form to be recreated from the metamodel instance. Such attributes are so described.

The purpose of this specification is to provide a common basis for communication among tools that create or compile EXPRESS models, analyze them, and/or map them to various forms of implementation specifications.

It is also intended to serve as a basis for the definition of formal standard mappings to other modeling and implementation languages.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. Subsequent amendments to, or revisions of, any of these publications do not necessarily apply. However, users and implementors of this specification are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. ISO and OMG maintain registers of currently valid specifications.

ISO 10303, *Industrial data – Product data exchange – Part 11: EXPRESS Language Reference Manual*, ed. 2, 2004.

OMG *Meta-Object Facility (MOF) Core Specification*, v2.0, January, 2006, (formal/06-01-01)

OMG *XML Metadata Interchange (XMI) Specification*, v2.1.1, December, 2007, (formal/07-12-01)

4 Conformance

An exchange document can conform to this specification as specified in Conformance of an exchange document. A tool can conform as a producer, as specified in Conformance as a producer (pre-processor), or as a processor, as specified in Conformance as a (post-)processor, or both. In addition, it is possible for a tool to conform to one or more of the compliance points specified in Compliance points, as a part of conformance as a producer or a processor.

4.1 Conformance of an exchange document

An exchange document conforms to this specification if

- it is a valid XMI exchange document and represents a MOF M1 model that is consistent with the M2 metamodel defined in clauses 6-12 of this specification; and

- the M1 model represents a valid EXPRESS schema as defined by ISO 10303-11.2:2004.

Representation of an EXPRESS schema need not include representation of all elements of the schema. It shall include all elements of the schema that can be represented by elements of the Core Package, as defined in clause 7.

4.2 Conformance as a producer (pre-processor)

A software tool conforms to this specification *as a producer* if it produces conforming exchange documents as specified in Conformance of an exchange document.

A software tool may claim conformance to a given compliance point as a producer if the exchange document it produces for any given EXPRESS schema contains representations of all the EXPRESS model elements that correspond to that compliance point.

4.3 Conformance as a (post-)processor

A software tool conforms to this specification *as a processor* if

- it can accept any and all exchange documents that conform as specified in Conformance of an exchange document; and
- it can interpret all EXPRESS concepts modeled by elements in the Core Package.

The nature of the process performed on the EXPRESS schema that is represented by a document that it accepts is not specified by this standard, but the interpretation of the EXPRESS schema in that process shall be consistent with the interpretation given by ISO 10303-11.

A software tool may claim conformance to a given compliance point as a processor if it can also interpret all the EXPRESS model elements that correspond to that compliance point.

4.4 Compliance points

In addition to support of the elements in the Core Package, a tool may support any of the additional compliance points defined below.

4.4.1 Compliance point: Enumerations

A tool conforms to the Enumerations compliance point by producing or interpreting model elements defined in the Enumerations Package.

4.4.2 Compliance point: Algorithms

A tool conforms to the Algorithms compliance point by producing or interpreting model elements defined in the Algorithms Packages. Conformance to this compliance point requires Statements to be produced as text, if the Statements compliance point is not supported. It makes no requirements for the interpretation of Statements.

4.4.3 Compliance point: Rules

A tool conforms to the Rules compliance point by producing or interpreting model elements defined in the Rules, Algorithms and Instances Packages. Conformance to this compliance point requires Statements to be produced as text, if the Statements compliance point is not supported. It makes no requirements for the interpretation of Statements.

4.4.4 Compliance point: Expressions

A tool conforms to the Expressions compliance point by producing or interpreting model elements defined in the Expressions, Algorithms and Instances Packages.

A tool that conforms as a producer to this compliance point shall not represent any Expression solely as text. That is, it shall represent every EXPRESS expression properly as the subtype of Expression that models it. Conformance to this compliance point requires Statements to be produced as text, if the Statements compliance point is not supported. It makes no requirements for the interpretation of Statements.

4.4.5 Compliance point: Statements

A tool that conforms to the Statements compliance point shall conform to the Expressions compliance point, and shall produce or interpret model elements defined in the Statements Package as well.

A tool that conforms as a producer to this compliance point shall not represent any Statement solely as text. That is, it shall represent every EXPRESS statement properly as the subtype of Statement that models it.

4.4.6 Compliance point: Express2

A tool conforms to the Express2 compliance point shall conform to the Statements compliance point and to the Rules compliance point. A tool that conforms to the Express2 compliance point shall fully support all elements of the EXPRESS language.

5 Terms and Definitions

5.1 Unified Modeling Language (UML) Terms

The following terms are taken from the Unified Modeling Language (UML) Specification, and are used with the definitions given in that specification:

- abstract
- association
- association end
- attribute
- class
- constraint
- dependency
- derived
- enumeration
- multiplicity
- navigable
- package
- stereotype
- tagged value

5.2 EXPRESS Terms

The following terms are taken from the EXPRESS Language Reference Manual, and are used with the definitions given in that specification:

- aggregate, and aggregation
- algorithm
- constant
- domain
- entity, and entity type
- function
- identifier
- instance
- inverse
- keyword
- member
- parameter
- population
- rule
- schema
- scope
- statement
- subtype
- supertype
- type

Some of these terms have the same orthography as certain UML terms that are not used in this specification. The following terms are used in this specification with their UML interpretation and are prefixed by “EXPRESS” whenever they are used with their EXPRESS interpretation:

- abstract
- attribute
- data type
- derived
- enumeration

5.3 Terms for Model Elements

This specification defines a number of metaclasses, associations, attributes and association end names. Each of those becomes a term that may be used in other definitions and requirements.

When a term is capitalized in the text, e.g., Schema, it refers to the metaclass with that identifier, and by extension, to the semantic concept that it represents.

In the text, a term beginning with a period (.) and set in Courier font, e.g., `:namespace`, refers to the attribute or association end with that name that is owned by the class being described.

Note – Other than these conventions, some terms that refer to model elements have the same spelling as terms used in UML and EXPRESS. The convention denotes the intended distinction. In most cases, however, when the EXPRESS term and the model element identifier have the same spelling, the model element models the concept designated by the EXPRESS term.

5.4 Additional terms introduced in this specification

The following additional terms are introduced in this specification:

instance package

A UML Package that comprises only declarations of individual objects that represent fixed instances of metaclasses defined in the parent package.

namespace

the domain of interpretation of a body of EXPRESS text in which a given identifier is associated with a given model element.

6 Document Conventions

This specification is a Complete Meta-Object Facility (CMOF) specification of the EXPRESS metamodel, conforming to the OMG Meta-Object Facility Core specification v2.0, as an M2 model.

The only CMOF features (beyond those of EMOF) that are used in this specification are:

- Specialization of primitive types
- Subsetting of properties

The stereotype `«implicit»` is used only as a drawing convention to show relationships that exist as more complex paths than are actually depicted. These associations do not exist in this form in the metamodel itself. These are used only to show important conceptual relationships on the diagrams.

MOF 2.0 requires that every association be named, even those that are navigable in only one direction. In this specification, all associations are named (in the UML and CMOF XMI files), but only the names of bidirectional associations are displayed and only bidirectional associations are separately documented (as Associations).

Similarly, MOF 2.0 requires that every association end be named, even those that are not navigable. In this specification, the names of non-navigable association ends are not shown and not documented. They do appear in the UML and CMOF XMI files for the metamodel. Every navigable association end is documented as a properties of the owning class.

For derived attributes and associations, the UML model includes an <<isDerived>> stereotype that allows the attachment of the tagged-value “derivation.” Wherever the derivation is a simple navigation expression, it is given as the value of “derivation” and documented accordingly in the normative text. Where the derivation is a more complex operation, it is omitted from the UML model and described in the text. The CMOF model does not include the tagged values, but wherever the derivation expression is given in the UML model, the CMOF model contains a Constraint requiring the value of the derived property to be equal to the value of the derivation expression.

7 Overview of the EXPRESS Metamodel

This specification is a metamodel for the EXPRESS information modeling language, as defined by ISO 10303-11. The Metamodel is composed of 7 packages, which are related as shown in Figure 1.

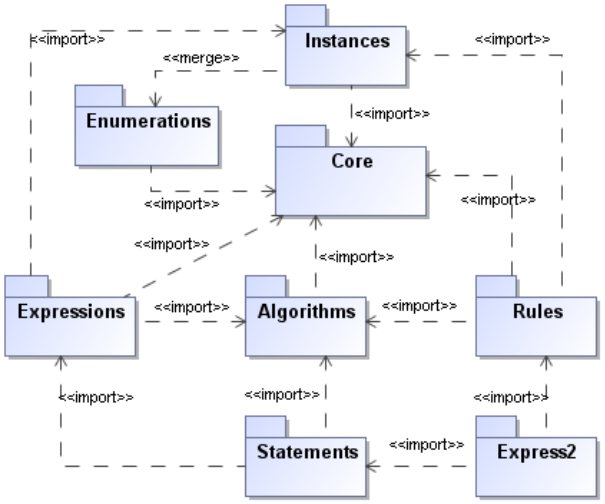

Figure 1 - EXRESS Metamodel Packages

The Core Package contains all of the generally required modeling elements of EXPRESS, along with some basic metamodel artifacts. It is the foundation on which all of the other packages are built. The Core Package is the minimal implementation of the EXPRESS Metamodel.

The Enumerations Package contains the model of EnumerationItem and the subclasses of Instance that it instantiates. Its purpose is to support a compliance point that includes schema-level declarations and EnumerationItems.

The Instances Package ~~contains~~ completes the model of instances that conform to the EXPRESS types. This package is needed to support many of the concepts in EXPRESS rules.

The Algorithms Package contains the model of function and procedure definitions. This model is needed to support Expressions, and some of its features are used in Global Rules.

The Rules Package contains the models of RULEs and SUBTYPE_CONSTRAINTS, which rely on the notion of extents of types with populations.

The Expressions Package contains a model of expressions that includes all details of value manipulation that are described in Clause 12 of ISO 10303-11.

The Statements Package contains a model of the executable statements that may be contained in the body of functions and procedures. It contains all of the concepts in Clause 13 of ISO 10303-11.

The Express2 Package contains nothing in its own right. It imports everything defined in the metamodel, and thus serves as the package that models the entire EXPRESS language.

8 Package :: Core

The Core package contains all of the generally required modeling elements of EXPRESS, including:

- Scopes and Naming concepts
- Schemas
- (Data) Types
- Entities, Attributes and Relationships
- Domain Constraints

The Core package also includes the abstract classes Expression and Instance, which serve as linking points for detailed models contained in other packages.

8.1 Dependencies

none.

8.2 MOF Metamodel Datatypes

The following basic data types from the MOF metamodel are used in this package with the interpretation given in the MOF specification.

8.2.1 Datatype: Boolean

Definition: MOF metatype for logical values.

8.2.2 Datatype: Integer

Definition: MOF metatype for numeric information.

Note – All integer values used in this metamodel are non-negative.

8.2.3 Datatype: String

Definition: MOF Metatype for arbitrary character (code) representation.

8.3 EXPRESS Language Datatypes

8.3.1 Datatype: ExpressText

Definition: Represents any EXPRESS language text, including both unparsed text and specific syntactic elements.

Note – See clause 7 of ISO 10303-11:2004.

8.3.1.1 Supertypes

Realization type is . MOF::String

The realization relationship is modeled as a generalization.

8.3.1.2 Members

none.

8.3.2 Datatype: Identifier

Definition: EXPRESS language element used for naming NamedElements.

Note – See 7.4 of ISO 10303-11:2004.

8.3.2.1 Supertypes

ExpressText.

8.3.2.2 Members

none.

8.3.3 Datatype: Keyword

Definition: EXPRESS language element used for names of built-in data types.

Note – See 7.2.1 of ISO 10303-11:2004.

8.3.3.1 Supertypes

ExpressText .

8.3.3.2 Members

none.

8.4 Schemas, Scopes and Naming

This section of the Core model introduces the naming and namespace concepts of the EXPRESS language.

An EXPRESS model consists primarily of a set of NamedElements – model elements that have (or may have) identifiers. Per Clause 10 of ISO 10303-11, every NamedElement has a Scope in which it is “visible,” that is, a collection of model contexts in which the identifier refers to that NamedElement. Such identifiers are modeled here as ScopedIds – the combination of an Identifier and the namespace (Scope) in which it is defined (see Datatype: ScopedId).

The primary Scope/namespace of an EXPRESS model is a Schema. All model elements, except those that are predefined in the EXPRESS language, are defined in some Schema. Interfacing is the mechanism by which an EXPRESS Schema includes model elements defined in other Schemas. Figure 2 - Scopes and Schemas shows the basic Scope, Schema, and Interfacing concepts of EXPRESS.

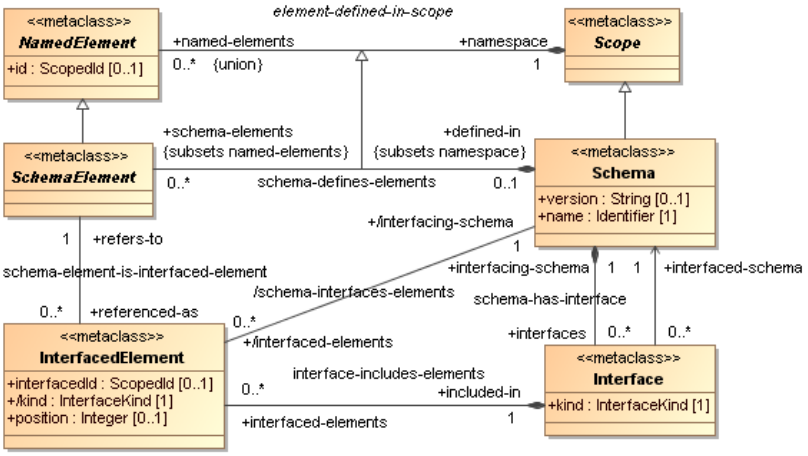

Figure 2 - Schemas and Interfacing

There are three general subclasses of Scope: Schemas, Local Scopes, and Type Scopes. These Scope concepts are shown in Figure 3. All of these concepts are defined in detail below, except for NamedType – the scope of TypeElements – which is defined in 8.6.

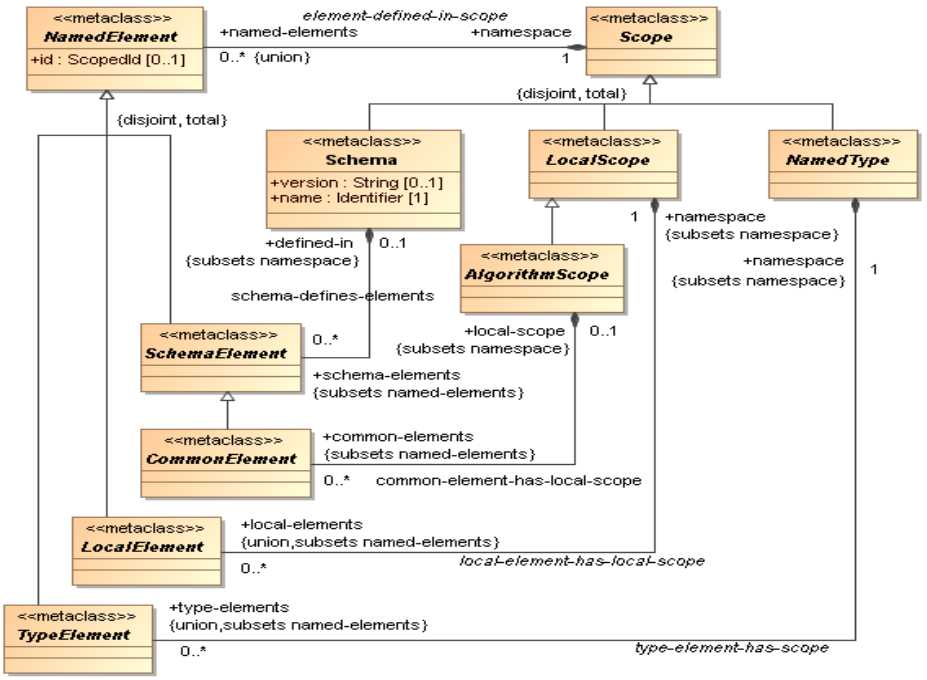

Figure 3 - EXPRESS Scope and Named Element Concepts

8.4.1 Class: AlgorithmScope

Definition: A LocalScope that can be the namespace of CommonElements.

Properties: abstract

8.4.1.1 Supertypes

LocalScope

8.4.1.2 Attributes

none.

8.4.1.3 Associations

AssociationEnd: common-elements **To: CommonElement**

via: common-element-has-local-scope

Subsets: Scope:named-elements

Definition: represents the relationship between an AlgorithmScope and the CommonElements that are defined in it. This is a refinement of the (abstract) Scope:named-elements relationship.

Note – See clause 10 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: variables **To: Algorithms::LocalVariable**

via: Algorithms::variable-defined-in-scope

Subsets: LocalScope:local-elements

Definition: represents the relationship between the AlgorithmScope and the set of LocalVariables that are defined within it.

Multiplicity: 0..* unordered

Properties: composite

8.4.1.4 Other Roles

none.

8.4.2 Class: CommonElement

Definition: a SchemaElement that can be defined in either a Schema or a LocalScope, and has (or may have) a unique identifier within that Scope. This is an artifact of the declaration and namespace rules for the EXPRESS language. NamedTypes, Algorithms, Constants, and SupertypeRules can be defined at the Schema level or within Algorithms and

GlobalRules (AlgorithmScopes).

Every CommonElement has a Scope. The Scope is either a SchemaScope or an AlgorithmScope.

Properties: abstract

8.4.2.1 Supertypes

SchemaElement

8.4.2.2 Attributes

none.

8.4.2.3 Associations

Note – The AssociationEnd: defined-in to Schema is inherited from SchemaElement.

AssociationEnd: local-scope **To: AlgorithmScope**

via: common-element-has-local-scope

Subsets: NamedElement:namespace

Definition: represents the relationship between a CommonElement that is defined in an AlgorithmScope and the scope in which it is defined; also, the scope (set of model elements) in which the id of the CommonElement refers to that CommonElement.

Note – See Clause 10 of ISO 10303-11:2004.

Multiplicity: 0..1

8.4.2.4 Other Roles

none.

8.4.2.5 Rules

Constraint has-scope (OCL)

```
exists(self->defined-in) XOR exists(self->local-scope)
```

A CommonElement has exactly one scope, either a Schema (via defined-in), or a LocalScope.

8.4.3 Class: Interface

Definition: represents the EXPRESS “interface” relationship between two Schemas that is created by a USE or REFERENCE statement.

Each EXPRESS interface statement (`USE` or `REFERENCE`) explicitly includes zero or more SchemaElements from the interfaced Schema in the interfacing Schema. Each interface statement shall be represented by an Interface object with the corresponding `:kind`. If there are multiple interface statements of the same kind for the same interfaced schema, they may all be represented by a single Interface object of that kind. Each SchemaElement that is explicitly interfaced by the statement(s)

shall be represented by exactly one InterfacedElement that is included in the Interface. Such elements are considered to be in the namespace of the interfacing Schema as well, but the identifier in the interfacing schema may be overridden in the InterfacedElement.

In addition, an EXPRESS interface statement may *implicitly* interface zero or more SchemaElements from the interfaced Schema in the interfacing Schema, in order to complete the specifications of the explicitly interfaced elements. For each interfaced schema from which one or more SchemaElements are implicitly interfaced, the interfacing Schema shall also contain one Interface object that has `:kind=IMPLICIT`, and that includes one InterfacedElement for each implicitly interfaced element from that interfaced Schema. Implicitly interfaced elements are not considered to be in the namespace of the interfacing schema, but they may appear in a corresponding population.

Note – See clause 11 of ISO 10303-11:2004. Interface models the USE and REFERENCE statements, but follows the interpretation rules given in that clause. In particular, a statement of the form

```
REFERENCE FROM <schema>;
```

explicitly interfaces every SchemaElement defined in the interfaced schema, and a statement of the form

```
USE FROM <schema>;
```

explicitly interfaces every NamedType defined in the interfaced schema.

Note – The above requires an interfaced element that is both USEd and REFERENCEd in the same interfacing schema to have two corresponding InterfacedElements, one in each of the Interface objects corresponding to the two kinds of interface statements.

Note – Per ISO 10303-11, a SchemaElement can be implicitly interfaced to define the terms used in defining explicitly interfaced SchemaElements in one USE or REFERENCE statement. The same SchemaElement can also be explicitly interfaced in another USE or REFERENCE statement. This specification does not require a SchemaElement that is explicitly interfaced to be modeled as implicitly interfaced at all. But SchemaElements that are implicitly interfaced at least once and are not explicitly interfaced at all must be modeled by InterfacedElements that are included in ~~at least one appropriate Interface~~ an Interface whose kind is implicit.

8.4.3.1 Supertypes

none.

8.4.3.2 Attributes

Attribute: kind **To: InterfaceKind**

Definition: The nature of the interface, as indicated by the interface statement the Interface represents: USE, REFERENCE, implicit.

Multiplicity: 1..1

8.4.3.3 Associations

AssociationEnd: interfaced-elements **To: InterfacedElement**

via: interface-includes-elements

Definition: the InterfacedElements that are included in the Interface. That is, the SchemaElements that are implicitly or explicitly interfaced into the interfacing schema by the USE or REFERENCE statement that is represented by the Interface.

Properties: composite

Multiplicity: 0..* unordered

AssociationEnd: interfaced-schema **To: Schema**

Definition: represents the relationship between the Interface and the Schema whose SchemaElements are being interfaced into the `.interfacing-schema`.

Multiplicity: 1..1

AssociationEnd: interfacing-schema **To: Schema**

via: schema-has-interface

Definition: represents the relationship between the Interface and the Schema in which it appears.

Multiplicity: 1..1

8.4.3.4 Other Roles

none.

8.4.4 Class: InterfacedElement

Definition: represents the EXPRESS “interface” relationship (USE, REFERENCE) between an interfacing Schema and one SchemaElement that is defined in some other Schema. It can be viewed as a “role” of the .refers-to SchemaElement in the interfacing schema. Each InterfacedElement is contained in exactly one Interface, which models one or more interface statements of the same kind for the interfaced schema. Because it is not meaningful for an interface statement to interface the same SchemaElement more than once, the combination (`:included-in`, `:refers-to`) uniquely identifies an InterfacedElement relationship.

Note – See clause 11 of ISO 10303-11:2004.

8.4.4.1 Supertypes

none.

8.4.4.2 Attributes

Attribute: interfacedId **To: ScopedId**

Definition: the new Identifier for the .refers-to SchemaElement in the interfacing schema.

Note – See clause 11 of ISO 10303-11:2004.

Multiplicity: 0..1

Attribute: kind **To: InterfaceKind**

Definition: the nature of the interface that is represented by the InterfacedElement: USE, REFERENCE, implicit. It is derived from the kind of Interface it is included in.

Multiplicity: 1..1

Properties: derived.

```
derivation = self->included-in->kind
```

8.4.4.3 Associations

AssociationEnd: included-in **To: Interface**

via: interface-includes-elements

Definition: the Interface that includes the InterfacedElement.

Multiplicity: 1..1

AssociationEnd: interfacing-schema **To: Schema**

via: schema-interfaces-elements

Definition: represents the relationship between the InterfacedElement and the Schema in which it appears. If the InterfacedElement renames the .refers-to SchemaElement, the interfacing-schema is the namespace for the .interfacedId.

Multiplicity: 1..1

Properties: derived.

TaggedValues

```
derivation = self->included-in->interfacing-schema
```

AssociationEnd: refers-to **To: SchemaElement**

via: schema-element-is-interfaced-element

Definition: represents the SchemaElement being imported (interfaced) into the interfacing schema as the InterfacedElement.

Multiplicity: 1..1

8.4.4.4 Other Roles

none.

8.4.5 Datatype: InterfaceKind

Stereotype: `enumeration`

Definition: The nature of an Interface – the EXPRESS interface relationship between two Schemas.

8.4.5.1 Supertypes

none.

8.4.5.2 Values

Value: IMPLICIT

Definition: Represents "implicit" interfacing, as defined in ISO 10303-11. A NamedElement is implicitly interfaced when it is not explicitly interfaced by any `USE` or `REFERENCE` statement but is used in the specification of a NamedElement that is explicitly interfaced, or in the specification of another element that is implicitly interfaced. Elements that are implicitly interfaced do not appear in the namespace of the interfacing schema. Instances of implicitly interfaced NamedTypes may appear in a Population governed by that Schema as if they were REFERENCEd.

Value: REFERENCE

Definition: Represents explicit interfacing by a `REFERENCE` statement. NamedElements that are explicitly interfaced have identifiers in the namespace of the interfacing schema. Instances of NamedTypes that are interfaced by a `REFERENCE` statement may exist in a Population, but only to fulfill some Attribute of an entity that is ultimately dependent on an "independent entity."

Value: USE

Definition: Represents explicit interfacing by a `USE` statement. NamedElements that are explicitly interfaced have identifiers in the namespace of the interfacing schema. Instances of every NamedType that is explicitly interfaced by a `USE` statement are permitted to be "independent entities" in a Population governed by the interfacing Schema.

8.4.6 Class: LocalElement

Definition: An abstract class, representing NamedElements whose scope is a LocalScope. No LocalElement is defined in the Core package.

Properties: abstract

8.4.6.1 Supertypes

NamedElement

8.4.6.2 Attributes

none.

8.4.6.3 Associations

AssociationEnd: namespace **To: LocalScope**

Subsets: NamedElement:namespace

Definition: the Scope in which the LocalElement is defined. Unlike SchemaElements, a LocalElement is instantiated only in the context of a particular "use" or "invocation" of the Scope in which it is defined. As a consequence, a LocalElement can be instantiated more than once in interpreting a Population under a given Schema, and each such instantiation has a "lifetime" corresponding to that use/invocation.

Multiplicity: 1..1

Properties: abstract

8.4.6.4 Other Roles

none.

8.4.7 Class: LocalScope

Definition: A Scope that is neither a Schema nor a NamedType. Terms defined in a LocalScope are not visible at the Schema level.

Note – See Clause 10 of ISO 10303-11:2004.

Properties: abstract

8.4.7.1 Supertypes

Scope

8.4.7.2 Attributes

none.

8.4.7.3 Associations

AssociationEnd: local-elements **To: LocalElement**

Subsets: Scope:named-elements

Definition: the LocalElements that are defined in the LocalScope. (A LocalScope that is an AlgorithmScope may also be the scope of CommonElements.)

Multiplicity: 0..* unordered

Properties: composite, derived union

8.4.7.4 Other Roles

none.

8.4.8 Class: NamedElement

Definition: An abstract class representing a principal modeling concept of the EXPRESS language: an object that is defined in a model, has a notion of “lifetime,” and has an identifier that refers to it in Schemas or in some nested Scope in a Schema.

Note – Every NamedElement has an :id attribute whose value is a ScopedId. Some NamedElements are not required to have identifiers, and some NamedElements can have additional identifiers. The scope of each such identifier is the Scope in which the NamedElement is defined.

Properties: abstract

8.4.8.1 Supertypes

none.

8.4.8.2 Attributes

Attribute: id **To: ScopedId**

Definition: Represents the identifier that uniquely identifies the NamedElement within the Scope that is the :namespace. Not all NamedElements are required to have identifiers.

Note – See Clause 10 of ISO 10303-11:2004.

Multiplicity: 0..1

8.4.8.3 Associations

AssociationEnd: documentation **To: Remark**

via: remark-describes-element

Definition: represents the relationship between a NamedElement and the Remarks, if any, that constitute its in-schema documentation. If the Scope (.appears-in) of the Remark is, or is contained in, a different Schema from the declaration of the NamedElement, the Remark only applies to the NamedElement as-interfaced.

Note – See 7.1.6.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: namespace **To: Scope**

Definition: represents the abstract relationship between a NamedElement and the “scope” in which it is defined, i.e., the set of model elements for which that name refers to that NamedElement.

Note – See clause 10 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: abstract

8.4.8.4 Other Roles

none.

8.4.9 Class: Schema

Definition: a Scope that represents an EXPRESS SCHEMA, i.e., a collection of SchemaElement declarations and interface declarations.

Note – “Schema” is a reserved word in EXPRESS; if this metamodel is converted to EXPRESS, this class must be renamed. See 9.3 of ISO 10303-11:2004.

8.4.9.1 Supertypes

Scope

8.4.9.2 Attributes

Attribute: name **To: Identifier**

Definition: the name of the EXPRESS schema.

Note – See clause 9.3 of ISO 10303-11:2004.

Multiplicity: 1..1

Attribute: version **To: Identifier**

Definition: the version identifier for the EXPRESS schema, if any.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..1

8.4.9.3 Associations

AssociationEnd: documentation **To: Remark**

via: remark-describes-schema

Definition: represents the relationship between a Schema and the Remarks, if any, that constitute its in-schema documentation. If the Scope (.appears-in) of the Remark is a different Schema, the Remark only applies to the Schema as-interfaced.

Note – See 7.1.6.3 of ISO 10303-11:2004. Technically the Schema is a named element of the EXPRESS language, but it has no defined Scope.

Multiplicity: 0..* unordered

AssociationEnd: interfaced-elements **To: InterfacedElement**

via: schema-interfaces-elements

Definition: represents the relationship between a Schema and the InterfacedElements it contains, that is, the SchemaElements that it imports/interfaces from other Schemas via USE and REFERENCE statements.

Properties: derived.

Multiplicity: 0..* unordered

TaggedValues

```
derivation = self->interfaces->interfaced-elements;
```

AssociationEnd: interfaces **To: Interface**

via: schema-has-interface

Definition: the Interfaces that link the Schema to the Schemas it interfaces and to the InterfacedElements they interface into the Schema.

Properties: composite

Multiplicity: 0..* unordered

AssociationEnd: schema-elements **To: SchemaElement**

via: schema-defines-elements

Subsets: Scope:named-elements

Definition: represents the relationship between the Schema and the SchemaElements that are defined in it, as distinct from those that are interfaced into it.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

8.4.9.4 Other Roles

From: Interface as interfaced-schema

From: Instances::Population as governing-schema

8.4.10 Class: SchemaElement

Definition: a NamedElement whose scope can be a Schema. This includes all CommonElements and GlobalRule. The scope of CommonElements can be a Schema, but is not required to be a Schema.

Properties: abstract

8.4.10.1 Supertypes

NamedElement

8.4.10.2 Attributes

none.

8.4.10.3 Associations

AssociationEnd: defined-in **To: Schema**

via: schema-defines-elements

Subsets: NamedElement:namespace

Definition: represents the relationship between the SchemaElement and the Schema in which it is (originally) defined.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: referenced-as **To: InterfacedElement**

via: schema-element-is-interfaced-element

Definition: represents a use of the SchemaElement in some Schema other than the one in which it is defined. Only a SchemaElement whose scope is a Schema can be referenced as an InterfacedElement.

Multiplicity: 0..* unordered

8.4.10.4 Other Roles

none.

8.4.10.5 Rules

Constraint (OCL)

```
exists(self->defined-in) OR NOT exists(self->referenced-as)
```

Only a SchemaElement that is defined-in a Schema can be referenced-as an InterfacedElement.

8.4.11 Class: Scope

Definition: any EXPRESS object that defines a namespace for the interpretation of identifiers.

Note – See clause 10 of ISO 10303-11:2004.

Properties: abstract

8.4.11.1 Supertypes

none.

8.4.11.2 Attributes

none.

8.4.11.3 Associations

AssociationEnd: includes-remarks **To: Remark**

via: remark-appears-in-scope

Definition: represents the relationship between a Schema and the Remarks that appear in it.

Note – See 7.1.6 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: named-elements **To: NamedElement**

Definition: represents the abstract relationship between a Scope and the NamedElements that are defined in it.

Note – This relationship is very much conceptual. Not every kind of NamedElement can be defined in every kind of Scope. See Clause 10 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite, derived union

8.4.11.4 Other Roles

From: Expression as interpretation-context

8.4.12 Datatype: ScopedId

Stereotypes: `structure`

Definition: The combination of an Identifier and its namespace, which together constitute a well-defined symbol for an EXPRESS ModelElement. Figure 4 shows the conceptual model of a ScopedId. A ScopedId whose Scope is a Schema is visible throughout the Schema, and possibly to other Schemas that interface the NamedElement. A ScopedId whose Scope is a LocalScope is visible only in that LocalScope. A ScopedId whose Scope is a NamedType is visible only in the declaration of that NamedType and in Expressions involving references to elements whose data type is that NamedType.

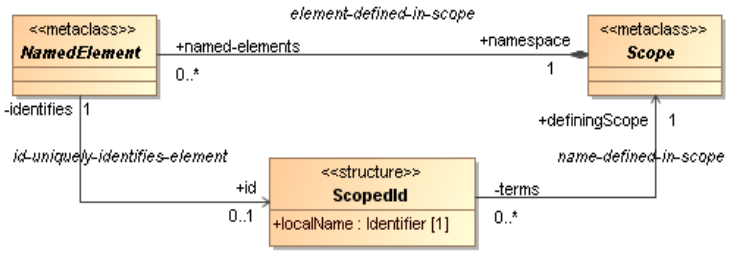

Figure 4 - Conceptual Model of ScopedId

8.4.12.1 Supertypes

none.

8.4.12.2 Members

Member: definingScope **To: Scope**

Definition: Represents the relationship between the ScopedId and the Scope in which it is defined.

Note – See Clause 10 of ISO 10303-11:2004.

Multiplicity: 1..1

Member: localName **To: Identifier**

Definition: Represents the EXPRESS identifier that uniquely identifies the NamedElement within the namespace that is the Scope.

Multiplicity: 1..1

8.4.13 Class: TypeElement

Definition: A NamedElement whose namespace is a data type (NamedType).

Note – See 8.2.2, 8.2.3, and 8.2.4 of ISO 10303-11:2004.

Properties: abstract

8.4.13.1 Supertypes

NamedElement

8.4.13.2 Attributes

none.

8.4.13.3 Associations

AssociationEnd: namespace **To: NamedType**

via: type-element-has-scope

Subsets: NamedElement:namespace

Definition: represents the relationship between the TypeElement and the NamedType in which it is defined. This is a refinement of the NamedElement:namespace and an abstraction of the specific relationships of TypeElements to their owner NamedTypes.

Multiplicity: 1..1

8.4.13.4 Other Roles

none.

8.4.14 Association: common-element-has-local-scope

Definition: represents the relationship between an AlgorithmScope and the CommonElements that are defined in it. This is a refinement of the (abstract) element-defined-in-scope relationship.

Note – See clause 10 of ISO 10303-11:2004.

8.4.14.1 Supertypes

element-defined-in-scope

8.4.14.2 Association Ends

AssociationEnd: common-elements **To: CommonElement**

Definition: the CommonElements that are defined in the AlgorithmScope.

Note – See clause 10 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: local-scope **To: AlgorithmScope**

Subsets: NamedElement:namespace

Definition: represents the relationship between a CommonElement that is defined in an AlgorithmScope and the scope in which it is defined; also, the scope (set of model elements) in which the id of the CommonElement refers to that CommonElement.

Note – See Clause 10 of ISO 10303-11:2004.

Multiplicity: 0..1

8.4.15 Association: element-defined-in-scope

Definition: represents the generic relationship between a NamedElement and the Scope in which it is defined. Every NamedElement is defined in exactly one Scope. It may be interfaced into other Scopes, and it may be visible in Scopes nested inside the Scope in which it defined and the Scopes into which it is interfaced.

Note – See clause 10 of ISO 10303-11:2004.

Note – This is an abstract relationship. Each separate form of this relationship is separately modeled.

Properties: abstract

8.4.15.1 Association Ends

AssociationEnd: named-elements **To: NamedElement**

Definition: represents the relationship between a Scope and the NamedElements that are defined in it.

Note – This relationship is very much conceptual. Not every kind of NamedElement can be defined in every kind of Scope. See clause 10 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite, derived union

AssociationEnd: namespace **To: Scope**

Definition: represents the relationship between a NamedElement and the “scope” in which it is defined, i.e., the set of model elements for which that name refers to that NamedElement.

Note – See clause 10 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: abstract

8.4.16 Association: interface-includes-elements

Definition: represents the relationship between an Interface and the InterfacedElements it contains, that is the relationship between an interface statement (USE or REFERENCE) and the SchemaElements it implicitly and explicitly interfaces.

Note – See clause 11 of ISO 10303-11:2004.

8.4.16.1 Association Ends

AssociationEnd: included-in **To: Interface**

Definition: the Interface that includes the InterfacedElement.

Multiplicity: 1..1

AssociationEnd: interfaced-elements **To: InterfacedElement**

Definition: the InterfacedElements that are included in the Interface. That is, the SchemaElements that are implicitly or explicitly interfaced into the interfacing schema by the USE or REFERENCE statement that is represented by the Interface.

Properties: composite

Multiplicity: 0..* unordered

8.4.17 Association: local-element-has-local-scope

Definition: represents the abstract relationship between a LocalScope and the LocalElements that are defined in it. (A LocalScope that is an AlgorithmScope may also be the scope of CommonElements.)

Note – This relationship is an abstraction of the actual relationships between LocalElements and Scopes. Each separate form of this relationship is separately modeled. Not every kind of LocalElement can be defined in every LocalScope. In fact, only NamedVariables can be defined in every LocalScope.

Note – See Clause 10.3 of ISO 10303-11:2004.

Properties: abstract

8.4.17.1 Supertypes

element-defined-in-scope

8.4.17.2 Association Ends

AssociationEnd: local-elements **To: LocalElement**

subsets: Scope:named-elements

Definition: the LocalElements that are defined in the LocalScope. (A LocalScope that is an AlgorithmScope may also be the scope of CommonElements.)

Multiplicity: 0..* unordered

Properties: composite, derived union

AssociationEnd: namespace **To: LocalScope**

Subsets: NamedElement:namespace

Definition: the Scope in which the LocalElement is defined. Unlike SchemaElements, a LocalElement is instantiated only in the context of a particular “use” or “invocation” of the Scope in which it is defined. As a consequence, a LocalElement can be instantiated more than once in interpreting a Population under a given Schema, and each such instantiation has a “lifetime” corresponding to that use/invocation.

Multiplicity: 1..1

Properties: abstract

8.4.18 Association: schema-defines-elements

Definition: represents the relationship between a Schema and the SchemaElements that are defined in it.

8.4.18.1 Supertypes

element-defined-in-scope

8.4.18.2 Association Ends

AssociationEnd: defined-in **To: Schema**

Subsets: NamedElement:namespace

Definition: represents the relationship between the SchemaElement and the Schema in which it is (originally) defined. Refines the (abstract) NamedElement:namespace relationship.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: schema-elements **To: SchemaElement**

subsets: Scope:named-elements

Definition: represents the relationship between the Schema and the SchemaElements that are defined in it, as distinct from those that are interfaced into it. refines the (abstract) Scope:named-elements relationship.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

8.4.19 Association: schema-element-is-interfaced-element

Definition: represents a use of the SchemaElement in some Schema other than the one in which it is defined. Only a SchemaElement whose scope is a Schema can be referenced as an InterfacedElement.

8.4.19.1 Association Ends

AssociationEnd: referenced-as **To: InterfacedElement**

Definition: represents a use of the SchemaElement in some Schema other than the one in which it is defined. Only a SchemaElement whose scope is a Schema can be referenced as an InterfacedElement.

Multiplicity: 0..* unordered

AssociationEnd: refers-to **To: SchemaElement**

Definition: represents the SchemaElement being imported (interfaced) into the interfacing schema as the InterfacedElement.

Multiplicity: 1..1

8.4.20 Association: schema-interfaces-elements

Definition: represents the EXPRESS “interface” relationships (USE, REFERENCE) between an interfacing Schema and the InterfacedElements that represent the SchemaElements that are interfaced from other Schemas.

Note – See clause 11 of ISO 10303-11:2004.

Properties: derived.

8.4.20.1 Association Ends

AssociationEnd: interfaced-elements **To: InterfacedElement**

Definition: represents the relationship between a Schema and the InterfacedElements it contains, that is, the SchemaElements that it imports/interfaces from other Schemas via USE and REFERENCE statements.

Properties: derived.

Multiplicity: 0..* unordered

TaggedValues

```
derivation = self->interfaces->interfaced-elements;
```

AssociationEnd: interfacing-schema **To: Schema**

Definition: represents the relationship between the InterfacedElement and the Schema in which it appears. If the InterfacedElement renames the .refers-to SchemaElement, the interfacing-schema is the namespace for the .interfacedId.

Properties: derived.

Multiplicity: 1..1

Tagged Values

derivation = self->included-in->interfacing-schema;

8.4.21 Association: schema-has-interface

Definition: represents the relationship between a Schema and the Interfaces it contains, and indirectly, the Schemas that it imports/interfaces.

Note – See clause 11 of ISO 10303-11:2004.

8.4.21.1 Association Ends

AssociationEnd: interfaces **To: Interface**

Definition: the Interfaces that link the Schema to the Schemas it interfaces and to the InterfacedElements they interface into the Schema.

Properties: composite

Multiplicity: 0..* unordered

AssociationEnd: interfacing-schema **To: Schema**

Definition: represents the relationship between the Interface and the Schema in which it appears.

Multiplicity: 1..1

8.4.22 Association: type-element-has-scope

Definition: represents the abstract relationship between a TypeElement and the NamedType in which it is defined. This is a refinement of the entity-defined-in-scope relationship and an abstraction of the specific relationships of TypeElements to their owner NamedTypes.

Properties: abstract.

8.4.22.1 Supertypes

element-defined-in-scope

8.4.22.2 Association Ends

AssociationEnd: type-elements **To: TypeElement**

subsets: Scope:named-elements

Definition: represents the relationship between the NamedType and the TypeElements that are defined in its scope.

Multiplicity: 0..* unordered

Properties: composite, derived union

AssociationEnd: namespace **To: NamedType**

subsets: NamedElement:namespace

Definition: represents the relationship between the TypeElement and the NamedType in which it is defined. This is a refinement of the NamedElement:namespace and an abstraction of the specific relationships of TypeElements to their owner NamedTypes.

Multiplicity: 1..1

8.5 Remarks

This section of the Core model introduces the Remark constructs that serve to document Schemas and NamedElements. Figure 5 depicts the Remark concept and its properties.

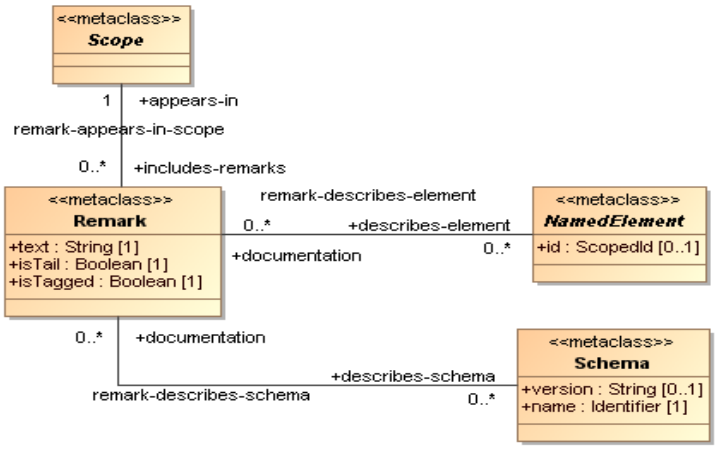

Figure 5 - Remarks

8.5.1 Class: Remark

Definition: A comment or or other documentation element that provides additional information about a model element.

8.5.1.1 Supertypes

none.

8.5.1.2 Attributes

Attribute: isTagged **To: MOF::Boolean**

Definition: Is TRUE if the Remark is "tagged" to refer to one or more NamedElements, and FALSE if the remark is not explicitly tagged.

If `.isTagged` is TRUE, the Remark should have the `.describes-element` or `.describes-schema` property.

Note – See 7.1.6.3 of ISO 10303-11:2004.

Multiplicity: 1..1

Attribute: isTail **To: MOF::Boolean**

Definition: is True if the Remark is lexically a `tail_remark`; and False if the Remark is lexically an `embedded_remark`. This distinction describes only the representation and placement of the remark in the EXPRESS syntax.

Note – See 7.1.6 of ISO 10303-11:2004.

Multiplicity: 1..1

Attribute: text **To: ExpressText**

Definition: Represents the actual text of the remark.

Note – Part 11 requires that the character set of the remark be the EXPRESS character set, but in practice a larger subset of ISO 10646-1 Basic Multilingual Plane is often used.

Note – See 7.1.6 of ISO 10303-11:2004.

Multiplicity: 1..1

8.5.1.3 Associations

AssociationEnd: appears-in **To: Scope**

via: remark-appears-in-scope

Definition: represents the relationship of a Remark to the Schema that lexically contains it.

Multiplicity: 1..1

AssociationEnd: describes-element **To: NamedElement**

via: remark-describes-element

Definition: represents the relationship between a Remark and the NamedElement(s) it describes. While a tagged remark is formally associated with one or more NamedElement(s), a processor may also ascribe a given un-tagged Remark to a given NamedElement, based on its lexical position.

Note – See 7.1.6.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: describes-schema **To: Schema**

via: remark-describes-schema

Definition: represents the relationship between a Remark that describes a Schema and the Schema it describes. The Remark may be Tagged to refer to the Schema, or it may be ascribed to the Schema if it lacks any other association. In particular, a Remark may appear in one Schema and refer to an interfaced Schema or to elements interfaced from it.

Note – See 7.1.6.3 of ISO 10303-11:2004. Technically the Schema is a named element of the EXPRESS language, but it has no defined Scope.

Multiplicity: 0..* unordered

8.5.1.4 Other Roles

none.

8.5.2 Association: remark-appears-in-scope

Definition: represents the relationship of a Remark to the Schema that lexically contains it.

8.5.2.1 Association Ends

AssociationEnd: appears-in **To: Scope**

Definition: the Schema that lexically contains the Remark.

Note – This may be the only cue as to the subject of the Remark. The first edition of EXPRESS did not specify a means for binding Remarks to model elements.

Multiplicity: 1..1

AssociationEnd: includes-remarks **To: Remark**

Definition: represents the relationship between a Schema and the Remarks that appear in it.

Note – See 7.1.6 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.5.3 Association: remark-describes-element

Definition: represents the relationship between a Remark and the NamedElement(s) it describes. While a tagged remark is formally associated with one or more NamedElements, a processor may also ascribe a given un-tagged Remark to a given NamedElement, based on its lexical position.

Note – See 7.1.6.3 of ISO 10303-11:2004.

8.5.3.1 Association Ends

AssociationEnd: describes-element **To: NamedElement**

Definition: the NamedElement(s) described by the Remark.

Note – See 7.1.6.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: documentation **To: Remark**

Definition: represents the relationship between a NamedElement and the Remarks, if any, that constitute its in-schema documentation. If the Scope (.appears-in) of the Remark is, or is contained in, a different Schema from the declaration of the NamedElement, the Remark only applies to the NamedElement as-interfaced.

Note – See 7.1.6.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.5.4 Association: remark-describes-schema

Definition: represents the relationship between a Schema and the Remarks, if any, that constitute its in-schema documentation. If the Scope (.appears-in) of the Remark is a different Schema, the Remark only applies to the Schema as-interfaced.

Note – See 7.1.6.3 of ISO 10303-11:2004. Technically the Schema is a named element of the EXPRESS language, but it has no defined Scope.

8.5.4.1 Association Ends

AssociationEnd: describes-schema **To: Schema**

Definition: represents the relationship between a Remark that describes a Schema and the Schema it describes. The Remark may be Tagged to refer to the Schema, or it may be ascribed to the Schema if it lacks any other association. In particular, a Remark may appear in one Schema and refer to an interfaced Schema or to elements interfaced from it.

Note – See 7.1.6.3 of ISO 10303-11:2004. Technically the Schema is a named element of the EXPRESS language, but it has no defined Scope.

Multiplicity: 0..* unordered

AssociationEnd: documentation **To: Remark**

Definition: represents the relationship between a Schema and the Remarks, if any, that constitute its in-schema documentation. If the Scope (.appears-in) of the Remark is a different Schema, the Remark only applies to the Schema as-interfaced.

Note – See 7.1.6.3 of ISO 10303-11:2004. Technically the Schema is a named element of the EXPRESS language, but it has no defined Scope.

Multiplicity: 0..* unordered

8.6 Overview of Types

This section of the Core model introduces the data type modeling concepts of the EXPRESS language, including the built-in types.

As is shown in Figure 6, the EXPRESS data type model consists of several dichotomies. Each of the high-level abstract types represents a group of EXPRESS data types that can play a given role in the metamodel.

DataType is the general class of types of results of Expressions. This includes all VariableTypes, together with “partial complex entity data types” (PartialEntityTypes), which can only occur as the result of an (intermediate) Expression.

VariableType is the general class of types that Variables can be declared to have. This includes all InstantiableTypes and ActualTypes, which are formal types that resolve to InstantiableTypes at the time the Variable is created.

ParameterType is the most general class of types that a model element, and in particular, Attributes and Parameters, can be declared to have. This includes all VariableTypes and GeneralizedTypes, which represent generalized requirements on the type of the element that must be specialized in actual uses.

Instantiable Types represent all the data type notions that characterize objects and properties in EXPRESS. Instantiable Types also represent all the data types that have Instances, except for PartialEntityTypes.They are subdivided into EntityTypes, which largely represent non-data objects, and ConcreteTypes, which represent data elements. They are also subdivided into NamedTypes, which are defined by declarations in the Schema, and AnonymousTypes, which are defined in the EXPRESS language and have specific syntactic designations instead of “identifiers.”

Any given object representing an EXPRESS data type is an instance of exactly one of InstantiableType, ActualType, GeneralizedType, and PartialEntityType, and in fact, it is an instance of exactly one specific instantiable subclass.

All of these concepts are defined below.

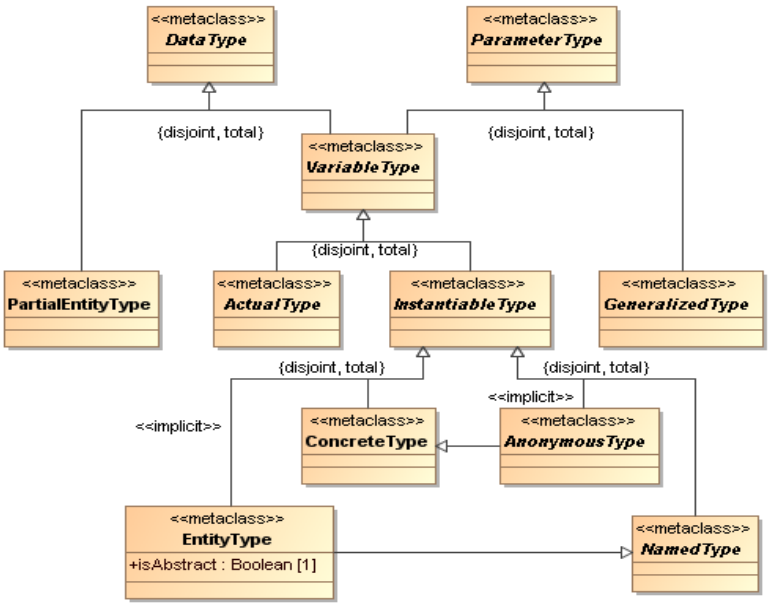

Figure 6 - Overview of EXPRESS Type concepts

Figure 6 also shows, using “implicit” subclass relationships for EntityTypes and AnonymousTypes, that there are two dichotomies for InstantiableTypes. Every InstantiableType is either an EntityType or a ConcreteType, and every InstantiableType is either a NamedType or an AnonymousType.

Figure 7 shows the model of Instantiable Types in detail. SimpleTypes, (Concrete)AggregationTypes and EntityTypes are defined in separate sections. The other classes and associations are defined below.

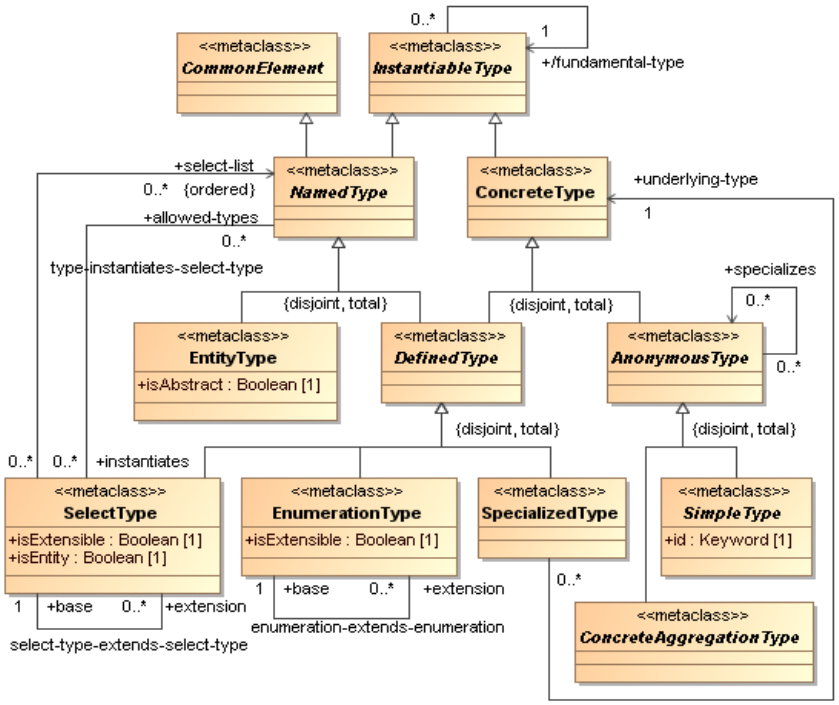

Figure 7 - NamedTypes and Instantiable Types

8.6.1 Class: ActualType

Definition: specification of an instantiable data type by reference to (a component of) the data type of the actual parameter that corresponds to a formal parameter of the Algorithm.

Each subtype of ActualType refers to a ParametricElement that is defined among the formal Parameters of the Algorithm. The ParametricElement denotes the corresponding component of the data type of the corresponding actual parameter in any given invocation. The ParametricElement is named by an EXPRESS `type_label`, and the ActualType refers to that ParametricElement via the `type_label`.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Properties: abstract

The details of ActualTypes are specified in the Algorithms Package (Clause 10.4).

8.6.1.1 Supertypes

Variable

8.6.1.2 Attributes

none.

8.6.1.3 Associations

AssociationEnd: scope **To: Algorithms::Algorithm**

Via: Algorithms::scope-of-actual-type

Definition: The Algorithm in which the ActualType is specified.

The ActualType must be the data type of a Variable or Attribute whose scope is contained in the Algorithm, and the ParametricElement that defines the `type_label` to which the ActualType refers must be defined among the formal parameters of the Algorithm.

An ActualType does not have a namespace; it defines no identifiers. The :scope of the ActualType represents the ownership of the ActualType and the lifetime of the ActualType.

Multiplicity: 1..1

8.6.1.4 Other Roles

From: Algorithms::ActualAggregationType as member-type

8.6.2 Class: AnonymousType

Definition: represents any InstantiableType that is not a NamedType.

Properties: abstract

8.6.2.1 Supertypes

InstantiableType, ConcreteType

8.6.2.2 Attributes

none.

8.6.2.3 Associations

AssociationEnd: specializes **To: AnonymousType**

Definition: represents the relationship of an AnonymousType to an AnonymousType of which it is a “specialization,” as specified in Part 11 clause 9.2.7. Unlike the specialization for defined data types, these relationships are true subtypes: the domain of the "specialization" is a subset of the domain of AnonymousType and has the same interpretation.

Multiplicity: 0..* unordered

8.6.2.4 Other Roles

From: AnonymousType as specializes

8.6.3 Class: ConcreteType

Definition: represents any InstantiableType that is not an EntityType.

Note – See 9.1 of ISO 10303-11:2004.

Properties: abstract

8.6.3.1 Supertypes

InstantiableType

8.6.3.2 Attributes

none.

8.6.3.3 Associations

none.

8.6.3.4 Other Roles

From: SpecializedType as underlying-type

8.6.4 Class: DataType

Definition: an ExpressionType that represents all the data type notions that can be declared for objects and properties in EXPRESS. Syntactically called parameter_type, it includes InstantiableTypes and GeneralizedTypes (which represent conformance rules for InstantiableTypes). It excludes PartialEntityTypes, which are only classifiers for intermediate results.

Note – See clause 8 of ISO 10303-11:2004.

Properties: abstract

8.6.4.1 Supertypes

none.

8.6.4.2 Attributes

none.

8.6.4.3 Associations

AssociationEnd: instances **To: Instance**

Definition: the modeled Instances of the DataType, if any. In general, Instances of a DataType are not modeled unless they appear directly in a Schema.

Note: For most DataTypes, navigating the association in this direction is not a required feature of the model.

Multiplicity: 0..* unordered.

8.6.4.4 Other Roles

From: Expression as data-type

8.6.5 Class: DefinedType

Definition: a NamedType representing an EXPRESS defined data type, a type declared by a type_declaration.

Note – See 8.3.2 and 9.1 of ISO 10303-11:2004.

Properties: abstract

8.6.5.1 Supertypes

ConcreteType, NamedType

8.6.5.2 Attributes

none.

8.6.5.3 Associations

none.

8.6.5.4 Other Roles

none.

8.6.6 Class: EnumerationType

Definition: a DefinedType representing an EXPRESS defined data type whose `underlying_type` is a ENUMERATION data type - a data type that has as its domain a set of named values.

Note – See 8.4.1 of ISO 10303-11:2004.

8.6.6.1 Supertypes

DefinedType

8.6.6.2 Attributes

Attribute: isExtensible **To: MOF::Boolean**

Definition: True if the EnumerationType can have additional values in a schema that interfaces it; False if not.

In the context schema for a population, the final set of possible values is known. But the set given in the defining schema may be incomplete and be extended by other EnumerationTypes for which this is the base.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.6.6.3 Associations

AssociationEnd: base **To: EnumerationType**

via: enumeration-extends-enumeration

Definition: represents the relationship of an extended EnumerationType to the EnumerationType it is BASED ON. The domain of the extended type includes all of the values of the base type and all the values defined in the extension.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: declared-items **To: Enumerations::EnumerationItem**

via: Enumerations:enumeration-declares-items

Subsets: NamedType:type-elements

Definition: represents the relationship of an EnumerationType to the EnumerationItems that are declared in its type_declaration. For extended enumeration types, this is distinct from the .values relationship, which captures all of the valid values of the type.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: extension **To: EnumerationType**

via: enumeration-extends-enumeration

Definition: represents the relationship of an EXTENSIBLE EnumerationType to the EnumerationTypes that are BASED ON it. Each extension type may add additional values to the domain, and these are considered to be values of the base type for all uses within the schema containing the extension.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: values **To: Enumerations::EnumerationItem**

via: Enumerations::value-of-EnumerationType

Definition: represents the relationship between an EnumerationType and the EnumerationItems that are valid values of the type. An EnumerationItem is a value of every EnumerationType that is related by extension to the type that declares it. This relationship can be derived recursively as the union of the values of the .declared-items attribute for the EnumerationType, for each EnumerationType in the sequence of .base relationships from the EnumerationType, and from all the extensions of the EnumerationType.

Note – See clause 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: derived

Note – The derivation of the entire list of values is a recursive operation, described in the Definition above.

8.6.6.4 Other Roles

none.

8.6.7 Class: InstantiableType

Definition: an abstract classifier, encompassing all the data type notions that characterize objects and properties in EXPRESS. InstantiableType is a proper subtype of DataType, which includes all the data types that have Instances.

Note – See 8.6.1 of ISO 10303-11:2004.

Properties: abstract

8.6.7.1 Supertypes

VariableType

8.6.7.2 Attributes

none.

8.6.7.3 Associations

AssociationEnd: fundamental-type **To: InstantiableType**

Definition: represents the relationship between the InstantiableType and the data type used to represent its values. The fundamental-type of a SpecializedType is the fundamental-type of its underlying-type; the fundamental-type of any other InstantiableType is the InstantiableType itself.

Note – ISO 10303-11 is not clear about the fundamental-type of a SelectType. The values of a SelectType are necessarily also values of one of the types in the select-list, and each value is represented according to the fundamental-type of its narrowest data type.

Note – See 13.3.2 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: derived

The derivation is a recursive operation as stated in the Definition above:

```
if self is a SpecializedType then
        self->fundamental-type = self->underlying-type->fundamental-type
else
        self->fundamental-type = self
```

8.6.7.4 Other Roles

From: InstantiableType as fundamental-type

From: InstantiableAggregationType as member-type

From: Instances::Constant as data-type

8.6.8 Class: NamedType

Definition: a CommonElement that defines a new InstantiableType.

Note – See 8.3 of ISO 10303-11:2004.

Properties: abstract

8.6.8.1 Supertypes

CommonElement , Scope, InstantiableType

8.6.8.2 Attributes

none.

8.6.8.3 Associations

AssociationEnd: domain-rules **To: DomainRule**

via: NamedType-has-DomainRule

Subsets: Core::NamedType:type-elements

Definition: a refinement of InstantiableType:constraints, represents the association of DomainRules that restrict the domain of valid values of the NamedType.

Note – See 9.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: instantiates **To: SelectType**

via: type-instantiates-select-type

Definition: represents the relationship between the NamedType and a SelectType whose domain includes it.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: type-elements **To: TypeElement**

via: type-element-has-scope

Subsets: Scope:named-elements

Definition: represents the relationship between the NamedType and the TypeElements that are defined in its scope.

Multiplicity: 0..* unordered

Properties: composite, derived union

8.6.8.4 Other Roles

From: SelectType as select-list

From: Instances::TypeName as refers-to

From: Expressions::ExtentRef as refers-to

8.6.8.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every NamedType shall have an identifier

8.6.9 Class: ParameterType

Definition: An abstract classification of Types that includes the InstantiableTypes, ActualTypes and GeneralizedTypes. That is, a ParameterType is any Type that is admissible as the declared type of a Parameter or an (abstract) ExplicitAttribute.

Note – See ISO 10303-11:2004 clause 8.6.2

Note – The lexical `parameter_type` in EXPRESS may represent an ActualType rather than a ParameterType, and it may include labeled GenericComponents that are used in ActualTypes and ActualTypeConstraints. All of these concepts are described in the Algorithms Package.

Properties: abstract

8.6.9.1 Supertypes

none.

8.6.9.2 Attributes

none.

8.6.9.3 Associations

AssociationEnd: constraints **To: DomainConstraint**

via: type-has-constraints

Definition: represents the association of DomainConstraints that restrict the value domain of the ParameterType

Note – See 8.1.6, 8.1.7, 8.2, and 9.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite, derived union

AssociationEnd: role **To: Attribute**

via: attribute-has-data-type

Definition: represents the relationship between the ParameterType and the roles (attributes of entities) that its admissible values may play.

Note – See 9.2.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.6.9.4 Other Roles

From: AGGREGATEType as member-type

From: Redeclaration as restricted-type

From: Algorithms::Parameter as formal-parameter-type

8.6.10 Class: SelectType

Definition: a DefinedType representing an EXPRESS defined data type whose `underlying_type` is a SELECT data type - a data type that has as its domain the union of the domains of a specified set of named data types.

Note – See 8.4.2 of ISO 10303-11:2004.

8.6.10.1 Supertypes

DefinedType

8.6.10.2 Attributes

Attribute: isEntity **To: MOF::Boolean**

Definition: represents a constraint on the extensions of an Extensible SelectType: True if every NamedType in the extension must be an EntityType; otherwise False.

Multiplicity: 1..1

Attribute: isExtensible **To: MOF::Boolean**

Definition: True if the SelectType is EXTENSIBLE, i.e., if it can have additional NamedTypes in the select-list when it is interfaced into another Schema; False otherwise.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 1..1

8.6.10.3 Associations

AssociationEnd: allowed-types **To: NamedType**

via: type-instantiates-select-type

Definition: represents the relationship of the SelectType to a NamedType whose values are included in the domain of the SelectType. All values in the domain of the NamedType are valid values of the SelectType.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: base **To: SelectType**

via: select-type-extends-select-type

Definition: represents the relationship of an extended select type to the (extensible) select type it is BASED ON.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: extension **To: SelectType**

via: select-type-extends-select-type

Definition: represents the relationship of an EXTENSIBLE select type to a select type BASED ON it.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: select-list **To: NamedType**

Definition: represents the appearance of the NamedType in the select list in the declaration of the SelectType. For extended and extensible SelectTypes, the NamedType should appear in exactly one of the select-lists in any set of SelectTypes related by extension. This is distinct from .allowed-types, which represents all of the NamedTypes that can validly instantiate the SelectType, including any related by extension. The select-list is said to be “ordered,” to convey the syntactic ordering. The ordering has no semantic significance.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* ordered

8.6.10.4 Other Roles

From: Instances::TypedInstance as satisfies-type

8.6.11 Class: SpecializedType

Definition: a DefinedType representing an EXPRESS defined data type whose `underlying_type` is neither an explicit ENUMERATION data type nor an explicit SELECT data type. According to ISO 10303-11 clause 9.1, a SpecializedType represents an abstract data type whose values are represented by values of the `underlying_type`; but in practice, a SpecializedType may also simply name an `underlying_type` that is an AnonymousType, or name an `underlying_type` whose domain is a subset of the domain of another DefinedType.

Note – See 9.1 of ISO 10303-11:2004.

8.6.11.1 Supertypes

DefinedType

8.6.11.2 Attributes

none.

8.6.11.3 Associations

AssociationEnd: underlying-type **To: ConcreteType**

Definition: represents the EXPRESS “specialization” relationship between a defined data type and the “underlying type” used to represent it.

Note – See 9.1 and 9.7 of ISO 10303-11:2004.

Multiplicity: 1..1

8.6.11.4 Other Roles

From: Instances::SpecializedValue as of-type

8.6.12 Class: VariableType

Definition: An abstract class representing the permissible data types of a variable: InstantiableTypes and ActualTypes.

Properties: abstract

8.6.12.1 Supertypes

ParameterType , DataType

8.6.12.2 Attributes

none.

8.6.12.3 Associations

none.

8.6.12.4 Other Roles

From: Algorithms::Variable as variable-type

From: Expressions::Coercion as target-type

From: Algorithms::ActualAGGREGATEType as member-type

8.6.13 Association: enumeration-extends-enumeration

Definition: represents the relationship of an EXTENSIBLE EnumerationType to the EnumerationTypes that are BASED ON it.

Note – See 8.4.1 of ISO 10303-11:2004.

8.6.13.1 Association Ends

AssociationEnd: base **To: EnumerationType**

Definition: represents the relationship of an extended EnumerationType to the EnumerationType it is BASED ON. The domain of the extended type includes all of the values of the base type and all the values defined in the extension.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: extension **To: EnumerationType**

Definition: represents the relationship of an EXTENSIBLE EnumerationType to the EnumerationTypes that are BASED ON it. Each extension type may add additional values to the domain, and these are considered to be values of the base type for all uses within the schema containing the extension.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.6.14 Association: select-type-extends-select-type

Definition: represents the relationship of an EXTENSIBLE select type to a select type BASED ON it.

Note – See 8.4.2 of ISO 10303-11:2004.

8.6.14.1 Association Ends

AssociationEnd: base **To: SelectType**

Definition: represents the relationship of an extended select type to the (extensible) select type it is BASED ON.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: extension **To: SelectType**

Definition: represents the relationship of an EXTENSIBLE select type to a select type BASED ON it.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.6.15 Association: type-instantiates-select-type

Definition: represents the appearance of the "generalizes" NamedType in the select list of the "instantiates" SelectType.

Note – See 8.4.2 of ISO 10303-11:2004.

8.6.15.1 Association Ends

AssociationEnd: allowed-types **To: NamedType**

Definition: represents the relationship of the SelectType to a NamedType whose values are included in the domain of the SelectType. All values in the domain of the NamedType are valid values of the SelectType.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: instantiates **To: SelectType**

Definition: represents the relationship between the NamedType and a SelectType whose domain includes it.

Note – See 8.4.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.7 Type Constraints

InstantiableTypes can have local constraints on the admissible values of their “domain.” The basic concept is shown in Figure 8. All NamedTypes can have DomainRules. AnonymousTypes have specialized constraints, which are shown in the sections for those types.

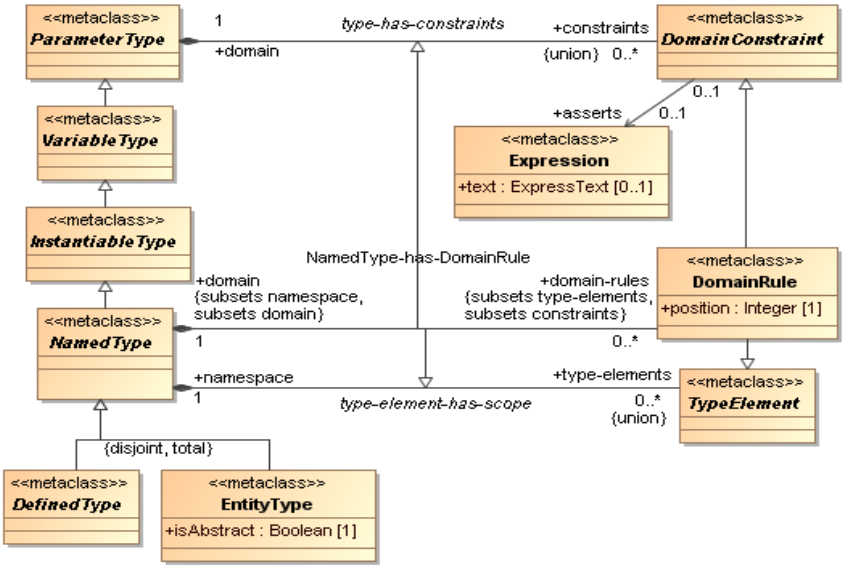

Figure 8 - Type Constraints

8.7.1 Class: DomainConstraint

Definition: represents a constraint on the allowable values of an EXPRESS data type. This concept does not appear explicitly in the EXPRESS language. Some DomainConstraints are explicit DomainRules (WHERE rules); others, such as SizeConstraints and LengthConstraints, are stated in the EXPRESS syntax for the data type. In this model, a DomainConstraint is always formulated as a (boolean) Expression, regardless of the EXPRESS syntax used to specify it.

Properties: abstract

8.7.1.1 Supertypes

none.

8.7.1.2 Attributes

none.

8.7.1.3 Associations

AssociationEnd: domain **To: ParameterType**

Definition: a dependency – represents the relationship between the DomainConstraint and the data type whose values it constrains.

Multiplicity: 1..1

Properties: abstract

AssociationEnd: asserts **To: Expression**

Definition: represents the relationship between the domain constraint and a Boolean expression that can be evaluated to determine if it holds.

While all DomainConstraints can be represented by Boolean expressions, some DomainConstraints have representations that do not require the Expression to be explicitly modeled. For this reason, `.asserts` has multiplicity 0..1. When the DomainConstraint has a simple representation (such as a fixed size that is an integer), `.asserts` may, but need not, have a value. When the DomainConstraint cannot be simply represented, `.asserts` shall have a value that is a Boolean expression that conveys the constraint.

Note – The asserts expression that formulates the DomainConstraint is wholly owned by the DomainConstraint. It is not treated as reusable.

Multiplicity: 0..1

EXAMPLE

For the EXPRESS text:

```
ENTITY roster;
  max_team: INTEGER;
  members: LIST [1:max_team+1] OF entry;
END_ENTITY;
```

The DomainConstraint representing the maximum size of the `members` list is a SizeConstraint that has no `.bound` value and has a value for `.asserts` that is an Expression of the form:

```
SizeOf(SELF.members) <= SELF.max_team + 1
```

The DomainConstraint representing the minimum size of the `members` list is a SizeConstraint that has `.bound = 1`. It is not required to have any value for `.asserts`. But, if present, the value of `.asserts` should be an Expression of the form:

```
SizeOf(SELF.members) >= 1
```

8.7.1.4 Other Roles

none.

8.7.2 Class: DomainRule

Definition: represents a DomainConstraint that is stated as an EXPRESS domain rule in a WHERE clause in the type_declaration or the entity declaration. In a type_declaration, it is a Boolean expression in terms of SELF that limits the allowable values in the domain of the data type. In an entity_declaration, it is a Boolean expression that constrains the values of one or more attributes (or other relationships) of the entity data type.

Note – See clauses 9.1 and 9.2.2.2 of ISO 10303-11:2004.

Note – Part 11 permits a DomainRule to evaluate to indeterminate ("?") and requires a rule with that evaluation to be treated as satisfied. The most common case is the evaluation of an expression involving an OPTIONAL attribute. Languages like OCL and OWL require the possibly indeterminate values to be protected by an EXISTS operation.

EXAMPLE

For the EXPRESS text:

```
ENTITY time_interval;
  begin_time: date_time;
  end_time: OPTIONAL date_time;
WHERE
  wr1: begin_time <= end_time;
END_ENTITY;
```

The EXPRESS domain rule `wr1` is represented by a DomainRule with `:id="wr1"` and `:position = 1`, and `:asserts` linked to an Expression of the form:

```
SELF.begin_time <= SELF.end_time
```

The proper translation of the EXPRESS DomainRule `wr1`, however, may require the rule to be represented as:

```
NOT EXISTS(SELF.end_time) OR (SELF.begin_time <= SELF.end_time)
```

8.7.2.1 Supertypes

TypeElement, DomainConstraint

8.7.2.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: Represents the position of the Domain Rule in the list of rules following the WHERE keyword in the entity/type declaration.

Multiplicity: 1..1

8.7.2.3 Associations

AssociationEnd: domain **To: NamedType**

via: NamedType-has-DomainRule

Subsets: Core::TypeElement:namespace, Core::DomainConstraint:domain

Definition: represents the relationship of the DomainRule to the NamedType that is the domain of values to which it applies.

Multiplicity: 1..1

8.7.2.4 Other Roles

none.

8.7.3 Association: NamedType-has-DomainRule

Definition: a refinement of type-has-constraints, representing the relationship of a NamedType to a DomainRule that restrict the domain of valid values of the NamedType.

Note – See 9.1 of ISO 10303-11:2004.

8.7.3.1 Supertypes

type-element-has-scope, type-has-constraints

8.7.3.2 Association Ends

AssociationEnd: domain **To: NamedType**

Definition: represents the relationship of the DomainRule to the NamedType that is the domain of values to which it applies.

Multiplicity: 1..1

AssociationEnd: domain-rules **To: DomainRule**

Definition: a refinement of InstantiableType:constraints, represents the association of DomainRules that restrict the domain of valid values of the NamedType.

Note – See 9.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

8.7.4 Association: type-has-constraints

Definition: an abstract relationship, represents the association between a ParameterType and a DomainConstraint that restricts the value domain of the ParameterType.

Note – See 8.1.6, 8.1.7, 8.2, and 9.1 of ISO 10303-11:2004.

Properties: abstract

Note – This is an abstract relationship. Each separate form of this relationship is separately modeled.

8.7.4.1 Association Ends

AssociationEnd: constraints **To: DomainConstraint**

Definition: represents the association of DomainConstraints that restrict the value domain of the ParameterType.

Note – See 8.1.6, 8.1.7, 8.2, and 9.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite, derived union

AssociationEnd: domain **To: ParameterType**

Definition: a dependency – represents the relationship between the DomainConstraint and the data type whose values it constrains.

Multiplicity: 1..1

Properties: abstract

8.8 Simple Types

The EXPRESS language defines “simple types” as those that carry a single conceptual information unit. Each simple type is denoted by a keyword, rather than an identifier. The simple types are BOOLEAN, INTEGER, LOGICAL, NUMBER, all BINARY types, all REAL types, and all STRING types. They are shown in Figure 9 and described below.

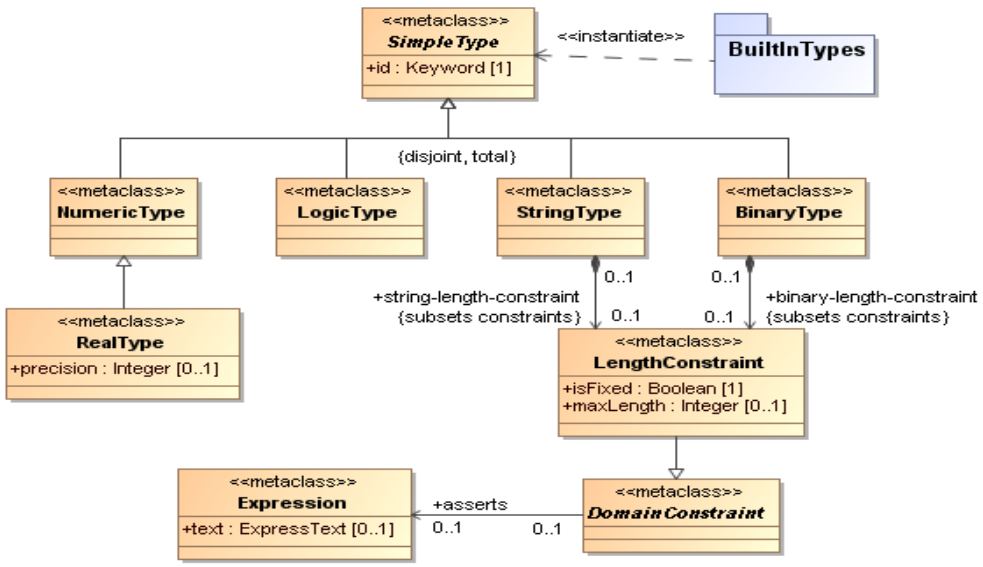

Figure 9 - Simple Types

8.8.1 Class: BinaryType

Definition: a SimpleType representing all EXPRESS BINARY data types, which are distinguished by different LengthConstraints.

By definition, every EXPRESS BINARY type with a LengthConstraint is different from every other BINARY data type. (They may be compatible with others, but not the same.) The only instance of BINARYType with no LengthConstraint is the EXPRESS data type BINARY.

Note – See 8.1.7 of ISO 10303-11:2004.

8.8.1.1 Supertypes

SimpleType

8.8.1.2 Attributes

none.

8.8.1.3 Associations

AssociationEnd: binary-length-constraint **To: LengthConstraint**

Subsets: ParameterType:constraints

Definition: represents a constraint on the length (in bits) of the values in the domain of the BINARY data type.

Note – See 8.1.7 of ISO 10303-11:2004.

Multiplicity: 0..1

8.8.1.4 Other Roles

From Instances:BinaryValue as of-type

8.8.2 Class: LengthConstraint

Definition: represents any maximum-length or fixed-length constraint on the length of the values of a STRING or BINARY type. A LengthConstraint is a DomainConstraint, considered to have an equivalent Boolean expression using the built-in Length() function.

Note – See 8.1.6 and 8.1.7 of ISO 10303-11:2004.

8.8.2.1 Supertypes

DomainConstraint

8.8.2.2 Attributes

Attribute: isFixed **To: MOF::Boolean**

Definition: True if all values of the SimpleType are required to be of the same length; False if the constraint specifies only the maximum length of the values.

Note – See 8.1.6 and 8.1.8 of ISO 10303-11:2004.

Multiplicity: 1..1

Attribute: maxLength **To: MOF::Integer**

Definition: represents a constant value specifying the required maximum/fixed length of the STRING or BINARY value. This attribute is present when the constraint expression is a “constant.”

Note – See 8.1.6 and 8.1.9 of ISO 10303-11:2004.

Multiplicity: 0..1

8.8.2.3 Associations

none.

8.8.2.4 Other Roles

From: StringType as string-length-constraint

From: BinaryType as binary-length-constraint

8.8.2.5 Rules

Constraint ()

Every LengthConstraint is either a string-length-constraint or a binary-length-constraint for exactly one SimpleType.

Constraint ()

A LengthConstraint is unique to the STRINGType or BINARYType it constrains.

8.8.3 Class: LogicType

Definition: a SimpleType representing the EXPRESS data types BOOLEAN and LOGICAL, which are the only instances of LOGICALType.

Note – See 8.1.4 of ISO 10303-11:2004.

8.8.3.1 Supertypes

SimpleType

8.8.3.2 Attributes

none.

8.8.3.3 Associations

none.

8.8.3.4 Other Roles

From Instances:LogicalValue as of-type

8.8.4 Class: NumericType

Definition: a SimpleType representing the EXPRESS data types NUMBER, INTEGER, and all REAL data types. NUMBER and INTEGER are instances of NUMBERType.

Note – See 8.1.1 of ISO 10303-11:2004.

8.8.4.1 Supertypes

SimpleType

8.8.4.2 Attributes

none.

8.8.4.3 Associations

none.

8.8.4.4 Other Roles

From Instances:NumberValue as of-type

8.8.5 Class: RealType

Definition: represents all EXPRESS REAL data types, which are distinguished from one another by different values of "precision." Type REAL (with no "precision" value) is one instance of REALType.

Note – See 8.1.2 of ISO 10303-11:2004.

8.8.5.1 Supertypes

NumericType

8.8.5.2 Attributes

Attribute: precision **To: MOF::Integer**

Definition: represents the number of significant figures in the values of the RealType, as specified in its syntactic designation. Although the value of "precision" is specified in EXPRESS to be an expression, it is assumed in this model that the value will in practice be a "constant." The only REALType for which "precision" is not present is the EXPRESS type REAL (with no precision specification).

Note – See 8.1.3 of ISO 10303-11:2004.

Multiplicity: 0..1

8.8.5.3 Associations

none.

8.8.5.4 Other Roles

none.

8.8.6 Class: SimpleType

Definition: an AnonymousType representing those EXPRESS data types defined in the language as "simple types": BINARY types, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL types, and STRING types.

Note – See 8.1 of ISO 10303-11:2004.

Properties: abstract

8.8.6.1 Supertypes

AnonymousType

8.8.6.2 Attributes

Attribute: id **To: Keyword**

Definition: represents the EXPRESS keyword denoting the SimpleType, one of: BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL, STRING.

Note – See 8.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.8.6.3 Associations

none.

8.8.6.4 Other Roles

none.

8.8.7 Class: StringType

Definition: a SimpleType representing all EXPRESS STRING data types, which are distinguished by different LengthConstraints. By definition, every EXPRESS STRING type with a LengthConstraint is different from every other STRING data type. (They may be compatible with others, but not the same.) The only instance of STRINGType with no LengthConstraint is the EXPRESS data type STRING.

Note – See 8.1.6 of ISO 10303-11:2004.

8.8.7.1 Supertypes

SimpleType

8.8.7.2 Attributes

none.

8.8.7.3 Associations

AssociationEnd: string-length-constraint **To: LengthConstraint**

Definition: represents a constraint on the length (in characters) of the values in the domain of the STRING data type.

Subsets: ParameterType:constraints

Note – See 8.1.6 of ISO 10303-11:2004.

Multiplicity: 0..1

8.8.7.4 Other Roles

From Instances:StringValue as of-type

8.9 Aggregation Types

EXPRESS “aggregation types” are types whose instances are collections of instances of a “member type.” There are four kinds of aggregation types, which represent different structures for the collections: ARRAY, BAG, LIST, SET. Figure 10 shows the overview of Aggregation types. The model elements are defined below.

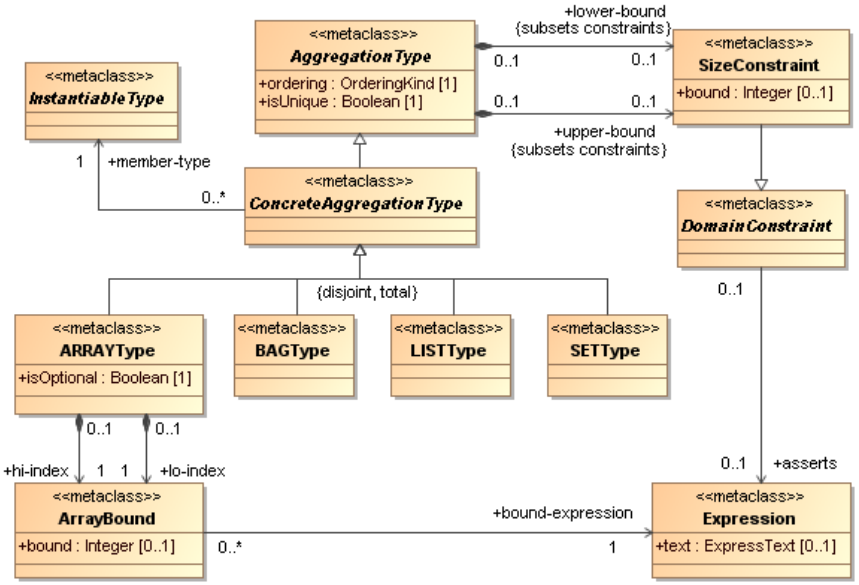

Figure 10 - Aggregation Types

8.9.1 Class: AggregationType

Definition: an AnonymousType representing an EXPRESS “aggregation type,” whose instances are collections of instances of a “member type”: ARRAY, BAG, LIST, SET.

Note – See 8.2 of ISO 10303-11:2004.

Properties: abstract

8.9.1.1 Supertypes

none.

8.9.1.2 Attributes

Attribute: isUnique **To: MOF::Boolean**

Definition: True if the members of a given instance of the type are required to be distinct; else False. isUnique is always True for a SET type, always False for a BAG type, and True for LIST and ARRAY types if and only if the UNIQUE keyword is present in the type designation.

Note – See 8.2 of ISO 10303-11:2004.

Multiplicity: 1..1

Attribute: ordering **To: OrderingKind**

Definition: Specifies the structure of the AggregationType: indexed (ARRAY), ordered (LIST), unordered (BAG, SET).

Multiplicity: 1..1

8.9.1.3 Associations

AssociationEnd: lower-bound **To: SizeConstraint**

Subsets: ParameterType:constraints

Definition: represents the appearance of a lower-bound constraint in syntactic designation for the aggregation type. Refines InstantiableType:constraints. For this purpose the appearance of an explicit zero ("0") value may be considered to represent no lower-bound constraint; and the lower-bound relationship need not appear. (The appearance of a lower-bound expression that may evaluate to zero shall always be represented by a lower-bound relationship.)

Note – See 8.2.2, 8.2.3, and 8.2.4 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: upper-bound **To: SizeConstraint**

Subsets: ParameterType:constraints

Definition: represents the appearance of an upper-bound constraint in the syntactic designation for the aggregation type. Refines InstantiableType:constraints. For this purpose the appearance of an explicit indeterminate value ("?") is considered to represent no upper-bound constraint, and shall not be represented by an upper-bound relationship. (The appearance of an upper-bound expression that may evaluate to "?" shall be represented by an upper-bound relationship.)

Note – See 8.2.2, 8.2.3, and 8.2.4 of ISO 10303-11:2004.

Multiplicity: 0..1

8.9.1.4 Other Roles

none.

8.9.2 Class: ArrayBound

Definition: represents a bound on the index domain of an ARRAY data type.

Note – See 8.2.1 of ISO 10303-11:2004.

8.9.2.1 Supertypes

none.

8.9.2.2 Attributes

Attribute: bound **To: MOF::Integer**

Definition: the integer value of the bound, when it can be determined “by inspection” of the bound expression.

Note – See 8.2.1 of ISO 10303-11:2004.

Multiplicity: 0..1

8.9.2.3 Associations

AssociationEnd: bound-expression **To: Expression**

Definition: the Expression that defines the ArrayBound.

Note – See 8.2.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.9.2.4 Other Roles

From: ARRAYType as hi-index

From: ARRAYType as lo-index

From: GeneralARRAYType as lo-index

From: GeneralARRAYType as hi-index

From: Algorithms::ActualARRAYType as lo-index

From: Algorithms::ActualARRAYType as hi-index

8.9.2.5 Rules

Constraint ()

Every ArrayBound is either a hi-index or lo-index for exactly one ARRAYType, ActualARRAYType, or GeneralARRAYType.

Constraint ()

An ArrayBound is unique to the A RRAYType (or GeneralARRAYType) and the role (hi-index/lo-index) it plays with respect to that type.

8.9.3 Class: ARRAYType

Definition: an AggregationType representing all EXPRESS ARRAY data types.

8.9.3.1 Supertypes

InstantiableAggregationType

8.9.3.2 Attributes

Attribute: isOptional **To: MOF::Boolean**

Definition: True if the member type is declared to be OPTIONAL in the syntactic designation for the ARRAYType; False otherwise. When isOptional is True, any instance of the ARRAYType is permitted to have members whose value is unspecified ("?").

Note – See 8.2.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.9.3.3 Associations

AssociationEnd: hi-index **To: ArrayBound**

Definition: represents the relationship between the ARRAYType and the upper bound on the Integer index-range of each value of the ARRAYType.

Note – See 8.2.1 and 15.11 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: lo-index **To: ArrayBound**

Definition: represents the relationship between the ARRAYType and the lower bound on the Integer index-range of each value of the ARRAYType.

Note – See 8.2.1 and 15.17 of ISO 10303-11:2004.

Multiplicity: 1..1

8.9.3.4 Other Roles

From: Instances::ARRAYValue as of-type

8.9.3.5 Rules

Constraint (OCL)

```
self->ordering = Indexed
```

8.9.4 Class: BAGType

Definition: an AggregationType representing all EXPRESS BAG data types.

Note – See 8.2.3 of ISO 10303-11:2004.

8.9.4.1 Supertypes

InstantiableAggregationType

8.9.4.2 Attributes

none.

8.9.4.3 Associations

none.

8.9.4.4 Other Roles

From: Instances::BAGValue as of-type

8.9.4.5 Rules

Constraint (OCL)

```
NOT self->isUnique
```

Constraint (OCL)

```
self->ordering = Unordered
```

8.9.5 Class: ConcreteAggregationType

Definition: an anonymous InstantiableType that is an AggregationType whose member-type is itself an InstantiableType.

Properties: abstract

8.9.5.1 Supertypes

AggregationType, AnonymousType

8.9.5.2 Attributes

none.

8.9.5.3 Associations

AssociationEnd: member-type **To: InstantiableType**

Definition: represents data type of its components (members) of the InstantiableAggregationType.

Multiplicity: 1..1

8.9.5.4 Other Roles

none.

8.9.6 Class: LISTType

Definition: an AggregationType representing all EXPRESS LIST data types.

Note – See 8.2.2 of ISO 10303-11:2004.

8.9.6.1 Supertypes

InstantiableAggregationType

8.9.6.2 Attributes

none.

8.9.6.3 Associations

none.

8.9.6.4 Other Roles

From: Instances::LISTValue as of-type

8.9.6.5 Rules

Constraint (OCL)

```
self->ordering = Ordered
```

8.9.7 Datatype: OrderingKind

Stereotypes: `enumeration`

Definition: Values that characterize the logical structure of the collections represented by an AggregationType (or a GeneralAggregationType).

8.9.7.1 Supertypes

none.

8.9.7.2 Values

Value: indexed

Definition: Specifies that the structure of the AggregateValues is an ARRAY. That is, the positions in the sequence are associated with specific (consecutive) INTEGER index values.

Value: ordered

Definition: Specifies that the structure of the AggregateValues is a LIST. That is, the position of each member-value in the sequence is significant in interpreting the AggregateValue.

Value: unordered

Definition: Specifies that the structure of the AggregateValues is a BAG or SET. That is, the position of each member-value in the sequence has no significance in interpreting the AggregateValue.

8.9.8 Class: SETType

Definition: an AggregationType representing all EXPRESS SET data types.

Note – See 8.2.4 of ISO 10303-11:2004.

8.9.8.1 Supertypes

InstantiableAggregationType

8.9.8.2 Attributes

none.

8.9.8.3 Associations

none.

8.9.8.4 Other Roles

From: Instances::SETValue as of-type

8.9.8.5 Rules

Constraint (OCL)

```
self->isUnique
```

Constraint (OCL)

```
self->ordering = Unordered
```

8.9.9 Class: SizeConstraint

Definition: A SizeConstraint represents a constraint on the number of members in each value of an EXPRESS aggregation type, stated as a bound in the syntactic designation for the type. A SizeConstraint represents either an upper-bound or a lower-bound. In the case of an ARRAY type, the value (hi-index - lo-index + 1) is both the lower-bound value and the upper-bound value. A SizeConstraint is a DomainConstraint, considered to have an equivalent Boolean expression using the built-in SizeOf() function.

Note – See 8.2.2, 8.2.3, and 8.2.4 of ISO 10303-11:2004.

8.9.9.1 Supertypes

DomainConstraint

8.9.9.2 Attributes

Attribute: bound **To: MOF::Integer**

Definition: represents a constant value specifying the (upper or lower) bound on the number of members in a valid instance of the aggregation type. This attribute is present when the bound expression is a “constant.”

Note – See 8.2.2, 8.2.3, and 8.2.4 of ISO 10303-11:2004.

Multiplicity: 0..1

8.9.9.3 Associations

none.

8.9.9.4 Other Roles

From: AGGREGATEType as upper-bound

From: AGGREGATEType as lower-bound

From: AggregationType as upper-bound

From: AggregationType as lower-bound

From: Redeclaration as upper-bound

From: Redeclaration as lower-bound

From: Role as lower-bound

From: Role as upper-bound

From: Algorithms::ActualAGGREGATEType as lower-bound

From: Algorithms::ActualAGGREGATEType as upper-bound

8.9.9.5 Rules

Constraint ()

Every SizeConstraint is either an upper-bound or a lower-bound for exactly one AggregationType or GeneralAggregationType.

Constraint ()

A SizeConstraint is unique to the AggregationType (or GeneralAggregationType) it describes and the role (upper-bound/ lower-bound) it plays with respect to that AggregationType.

8.10 Generalized Types

Generalized types are those EXPRESS data types that are “abstract,” in the sense that every actual instance is an instance of some InstantiableType(s). These types are only permitted as the data type of formal parameters and the data type of “abstract”

Attributes of ABSTRACT EntityTypes. They are shown in Figure 11.

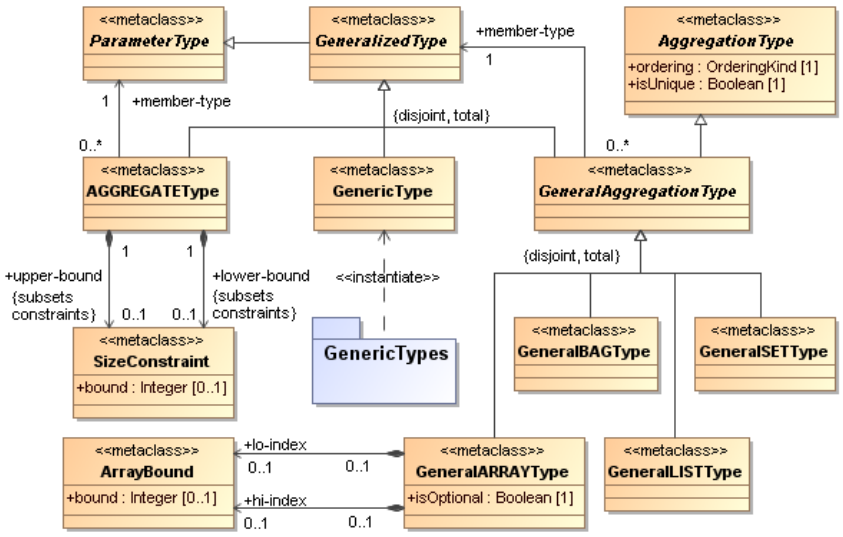

Figure 11 - Generalized Types

8.10.1 Class: AGGREGATEType

Definition: a GeneralizedType that is an abstraction of all AggregationTypes and all GeneralAggregationTypes. That is, any ARRAY, BAG, LIST, or SET Instance that satisfies the SizeConstraints (if any), whose members are of the specified member type or some specialization of it, is an instance of the AGGREGATEType. It follows that any ARRAY, BAG, LIST, or SET type whose instances are necessarily instances of the AGGREGATEType is a specialization.

Each syntactic occurrence of `AGGREGATE` is considered to be a distinct instance of AGGREGATEType, even when the bounds and member-type are the same as those of some other syntactic occurrence, because the corresponding types of the actual parameters or subtype attributes need not be the same. When the structures are required to be the same, that is represented as an ActualStructureConstraint.

Note – When the keyword AGGREGATE is followed by an EXPRESS `type_label`, there are three possible interpretations in the metamodel:

- A ParametricStructure is being defined to have that `type_label` (see x.x) and relate to the datatype of the actual parameters or instantiable attributes that correspond to the `:source`. The datatype, or component of the datatype, of the `:source` is a new AGGREGATEType. This is the interpretation of the first occurrence of the `type_label` in a parameter list or entity declaration.
- An ActualStructureConstraint is being specified that refers to the ParametricStructure with that `type_label`. The datatype denoted by the occurrence of `AGGREGATE:label` is a new AGGREGATEType that has that constraint. This is the interpretation of any later occurrence of the `type_label` in the same parameter list or entity declaration.
- A new ActualAGGREGATEType is being defined by reference to the ParametricStructure with that `type_label` (see Section 11.4.3), and the datatype of the variable, attribute, or member is the ActualAGGREGATEType. This is the interpretation of any other occurrence of the `type_label` within the same Algorithm.

Note – See 9.5.3.1 of ISO 10303-11:2004.

8.10.1.1 Supertypes

GeneralizedType

8.10.1.2 Attributes

none.

8.10.1.3 Associations

AssociationEnd: constraint **To: Algorithms::ActualStructureConstraint**

via: Algorithms::aggregate-has-constraint

Definition: the ActualStructureConstraint, if any, that applies to this component of the GeneralizedType specification.

Note – Only an AGGREGATEType that appears in the specification of the data type of a Parameter can have an ActualStructureConstraint. The AGGREGATEType has an ActualStructureConstraint only if it has a syntactic type_label and does not itself define that type_label.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: defines-parameter **To: ParametricStructure**

via: AGGREGATEType-defines-parameter

Definition: The ParametricStructure, if any, that is defined to refer to the structure of the actual data types that conform to this AGGREGATEType.

Multiplicity: 0..1

AssociationEnd: lower-bound **To: SizeConstraint**

Subsets: ParameterType:constraints

Definition: represents a lower-bound constraint on aggregate values conforming to the AGGREGATE type. If the lower-bound constraint is present, the number of members of the aggregate value shall be greater than or equal to this value. If the lower-bound is not present or evaluates to zero, there is no constraint. Unless the lower-bound specified for the AGGREGATIONType is an explicit “0,” this constraint shall appear.

Note – See 9.5.3.2 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: member-type **To: ParameterType**

Definition: represents the relationship between an AGGREGATE Type and the specification for the data type of the members of its instances. If the specification is an InstantiableType, the member-type of conforming aggregation types is required to be

exactly that data type. If the specification is a GeneralizedType, the member-type of the conforming aggregation types must conform to it.

Note – See 9.5.3.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: upper-bound **To: SizeConstraint**

Subsets: ParameterType:constraints

Definition: represents an upper-bound constraint on aggregate values conforming to the AGGREGATE type. If the upper-bound constraint is present and does not evaluate to indeterminate ("?"), the number of members of the aggregate value shall be less than or equal to this value. If the upper-bound is not present or evaluates to indeterminate, there is no constraint. Unless the upper-bound specified for the AGGREGATE type is an explicit "?", this constraint shall appear.

Note – See 9.5.3.3 of ISO 10303-11:2004.

Multiplicity: 0..1

8.10.1.4 Other Roles

none.

8.10.2 Class: GeneralAggregationType

Definition: represents a GeneralizedType whose instances are AggregateValues with a specific structure (ARRAY, BAG, LIST, or SET), but whose member-types are specializations of some specified GeneralizedType. That is, a GeneralAggregationType is an aggregation data type whose member-type is specified to be a GeneralizedType; while an (Instantiable) AggregationType is an aggregation data type whose member-type is specified to be an InstantiableType.

Any instance of a GeneralAggregationType is required to be an AggregateValue that has the specified structure and has members that are instances of some InstantiableType that conforms to the specified member-type. In addition,the instance must satisfy any DomainConstraints associated with the GeneralAggregationType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

Properties: abstract

8.10.2.1 Supertypes

AggregationType, GeneralizedType

8.10.2.2 Attributes

none.

8.10.2.3 Associations

AssociationEnd: member-type **To: GeneralizedType**

Definition: represents the relationship between a GeneralAggregationType and the conformance specification for the member-type.

Note – See 9.5.3.5 of ISO 10303-11:2004.

Multiplicity: 1..1

8.10.2.4 Other Roles

none.

8.10.3 Class: GeneralARRAYType

Definition: represents a GeneralAggregationType whose structure is an ARRAY. The hi-index and lo-index values of a conforming ARRAYInstance are required to be equal to the values given for the GeneralARRAYType.

When the GeneralARRAYType is the data type of an abstract attribute (see Section 11.4.4), the datatype of every conforming redeclaration is required to be an ARRAYType or a GeneralARRAYType whose hi-index and lo-index values are equal to the values given for the GeneralARRAYType. In addition, the .isOptional property of the redeclaration shall be as specified below.

Note – See 9.5.3.5 of ISO 10303-11:2004.

8.10.3.1 Supertypes

GeneralAggregationType

8.10.3.2 Attributes

Attribute: isOptional **To: MOF::Boolean**

Definition: When isOptional is True, any conforming ARRAYInstance is permitted to have members whose value is indeterminate ("?"). When isOptional is False, no member of a conforming ARRAYInstance is permitted to have an unspecified value.

If isOptional is True for an abstract attribute, the member type of any attribute that redeclares the abstract attribute may be declared to be OPTIONAL; if False, the member type of an attribute that redeclares the abstract attribute shall not be declared to be OPTIONAL.

Note – See 9.5.3.5 of ISO 10303-11:2004.

Multiplicity: 1..1

8.10.3.3 Associations

AssociationEnd: hi-index **To: ArrayBound**

Definition: The hi-index value of a conforming ARRAY data type is required to be equal to the hi-index value, if any, for the GeneralARRAYType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: lo-index **To: ArrayBound**

Definition: The lo-index value of a conforming ARRAY data type is required to be equal to the lo-index value, if any, for the GeneralARRAYType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

Multiplicity: 0..1

8.10.3.4 Other Roles

none.

8.10.4 Class: GeneralBAGType

Definition: represents a GeneralAggregationType whose structure is a BAG.

When the GeneralBAGType is the data type of an abstract attribute (see Section 11.4.5), the datatype of every conforming redeclaration is required to be a BAGType or a GeneralBAGType that includes or refines any DomainConstraint associated with the GeneralBAGType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

8.10.4.1 Supertypes

GeneralAggregationType

8.10.4.2 Attributes

none.

8.10.4.3 Associations

none.

8.10.4.4 Other Roles

none.

8.10.5 Class: GeneralizedType

Definition: an abstract classifier, representing those EXPRESS data types that are “abstract,” in the sense that every actual instance is an instance of some InstantiableType(s). These types are only permitted as the data type of formal parameters and the data type of “abstract” Attributes of ABSTRACT EntityTypes. GeneralizedType is a proper subclass of ParameterType that is disjoint with InstantiableType.

Note – The syntactic occurrences of EXPRESS `generalized_type` do not always denote GeneralizedTypes per se. In particular, a generalized_type that appears with a type_label may denote an ActualType or a constraint. When used as the type of a LocalVariable or FunctionResult, it denotes an ActualType (q.v.). When used as the type of a Parameter, it may be a ParametricElement that defines a reference to the data type of the corresponding actual parameter (in addition to being a GeneralizedType specification for the allowable data types of the actual parameter), or it may represent a constraint on the data type of the corresponding actual parameter that relates to the data type of another actual parameter.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Properties: abstract

8.10.5.1 Supertypes

ParameterType

8.10.5.2 Attributes

none.

8.10.5.3 Associations

none.

8.10.5.4 Other Roles

From: GeneralAggregationType as member-type

8.10.6 Class: GeneralLISTType

Definition: represents a GeneralAggregationType whose structure is a LIST.

When the GeneralLISTType is the data type of an abstract attribute (see Section 11.4.7), the datatype of every conforming redeclaration is required to be a LISTType or a GeneralLISTType that includes or refines any DomainConstraint associated with the GeneralLISTType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

8.10.6.1 Supertypes

GeneralAggregationType

8.10.6.2 Attributes

none.

8.10.6.3 Associations

none.

8.10.6.4 Other Roles

none.

8.10.7 Class: GeneralSETType

Definition: represents a GeneralAggregationType whose structure is a SET.

When the GeneralSETType is the data type of an abstract attribute (see Section 11.4.8), the datatype of every conforming redeclaration is required to be a SETType or a GeneralSETType that includes or refines any DomainConstraint associated with the GeneralSETType.

Note – See 9.5.3.5 of ISO 10303-11:2004.

8.10.7.1 Supertypes

GeneralAggregationType

8.10.7.2 Attributes

none.

8.10.7.3 Associations

none.

8.10.7.4 Other Roles

none.

8.10.8 Class: GenericType

Definition: represents the EXPRESS generalized types GENERIC and GENERIC_ENTITY.

Every data type is a specialization of the GenericType GENERIC, and every Instance is an Instance of GENERIC. Every entity data type is a specialization of the GenericType GENERIC_ENTITY. Every EntityInstance is an instance of GENERIC_ENTITY and every instance of GENERIC_ENTITY is an EntityInstance.

Note – See 9.5.3.2 and 9.5.3.3 of ISO 10303-11:2004.

Note – When the keywords GENERIC and GENERIC_ENTITY are followed by an EXPRESS `type_label`, there are three possible interpretations in the metamodel:

- A ParametricType is being defined to have that `type_label` (see Section 11.4.6) and relate to the datatype of the actual parameters or instantiable attributes that correspond to the `:source`. The datatype, or component of the datatype, of the `:source` is the GenericType. This is the interpretation of the first occurrence of the `type_label` in a parameter list.
- An ActualTypeConstraint is being specified that refers to the ParametricType with that `type_label`. The datatype denoted by the occurrence of `GENERIC:label` or `GENERIC_ENTITY:label` is the GenericType but the allowable data types that correspond to it in this usage are constrained by the ActualTypeConstraint. This is the interpretation of any later occurrence of the `type_label` in the same parameter list.

- An ActualGenericType is being identified by reference to the ParametricType with that `type_label` (see Section 11.4.6), and the datatype of the variable, attribute, or member is the ActualGenericType. This is the interpretation of any other occurrence of the `type_label` within the same Algorithm.

8.10.8.1 Supertypes

GeneralizedType

8.10.8.2 Attributes

none.

8.10.8.3 Associations

AssociationEnd: constraint **To: Algorithms::ActualTypeConstraint**

via: Algorithms::generic-has-constraint

Definition: the ActualTypeConstraint, if any, that applies to this component of the GeneralizedType specification.

Note – Only a GenericType that appears in the specification of the data type of a Parameter can have an ActualTypeConstraint. The GenericType has an ActualTypeConstraint only if it has a syntactic type_label and does not itself define that type_label.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..1

8.10.8.4 Other Roles

none.

8.10.8.5 Rules

Constraint (OCL)

```
self = GenericTypes::GENERIC OR self = GenericTypes::GENERIC_ENTITY;
```

8.11 Entities and Attributes

This section of the Core model introduces the entity and attribute concepts of the EXPRESS language.

Figure 12 shows the primary concepts associated with EXPRESS entities: EntityTypes, Attributes, UNIQUE rules, and DomainRules (WHERE rules). The SingleEntityType represents the group of attributes declared explicitly in the entity declaration (as distinct from those inherited), and is used in PartialEntityValues (see 9.5.6) that represent states of entities. PartialEntityType is a special data type that characterizes such values when they are produced in Expressions. All of these concepts are described in detail below.

DomainRules are a kind of TypeConstraint that applies to NamedTypes in general. They are described in 8.7.2. In the particular case of EntityTypes, they are used to capture constraints on the relationships among Attributes of the entity data type.

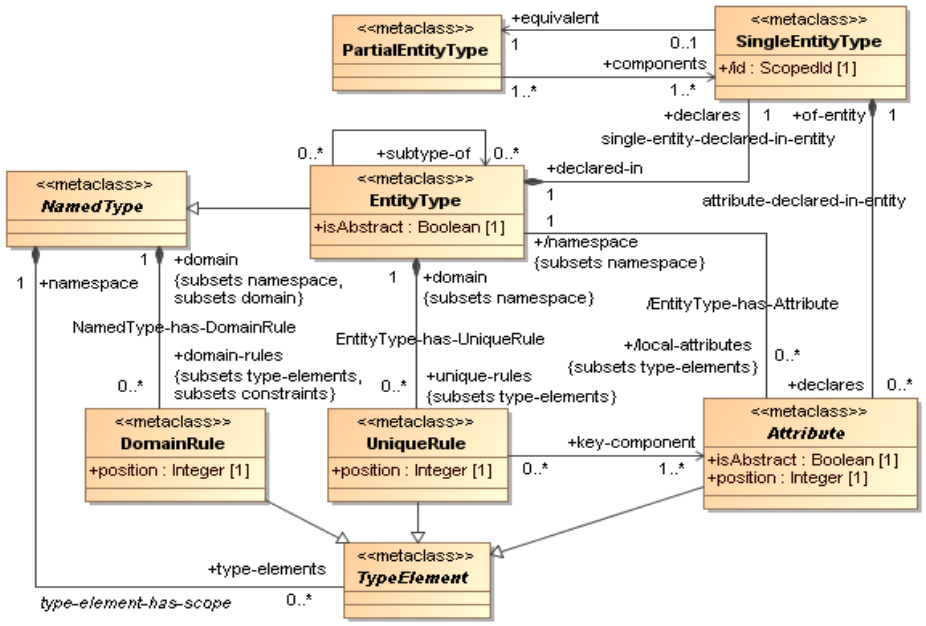

Figure 12 - Entity Types

Figure 13 depicts the concepts associated with Attributes in EXPRESS. Attributes are of three kinds: explicit, INVERSE, and DERIVEd. To facilitate modeling INVERSE attributes and relationships, this model adds the concept InvertibleAttribute. All of these concepts are described below.

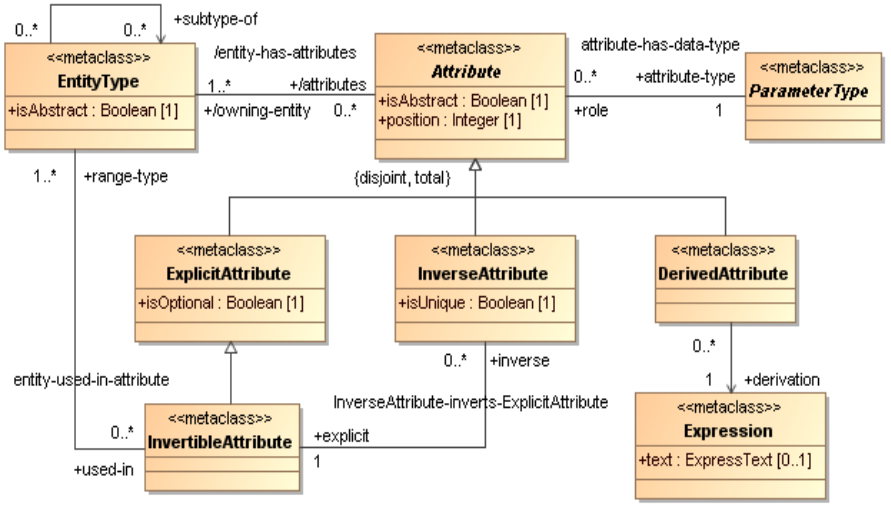

Figure 13 - Attributes

8.11.1 Class: Attribute

Definition: represents an EXPRESS attribute, i.e., a model of a property of an entity instance.

Note – See 9.2.1 of ISO 10303-11:2004.

Properties: abstract

8.11.1.1 Supertypes

TypeElement, ElementSource

8.11.1.2 Attributes

Attribute: isAbstract **To: MOF::Boolean**

Definition: True if .isAbstract is True for the owning EntityType (see .of-entity) and the attribute-type of the EXPRESS attribute is a GeneralizedType; False in all other cases. When .isAbstract is True, this Attribute must be redeclared to have an attribute-type that is an InstantiableType in any subtype of the owning EntityType that is not itself ABSTRACT.

Multiplicity: 1..1

Attribute: position **To: MOF::Integer**

Definition: Represents the position of the attribute declaration in the sequence of attribute declarations in the entity declaration.

Multiplicity: 1..1

8.11.1.3 Associations

AssociationEnd: attribute-type **To: ParameterType**

via: attribute-has-data-type

Definition: represents the required data type for all values of that Attribute in all instances of the EntityType. When EntityType that declares the Attribute is “abstract”, the attribute-type can be a GeneralizedType. When the Attribute is defined within the scope of an Algorithm, the attribute-type can be an ActualType. In these cases, the attribute-type can also be an InstantiableType, and in any other case, the attribute-type is required to be an InstantiableType.

Note – See 9.2.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: namespace **To: EntityType**

via: EntityType_has_Attribute

subsets: TypeElement::namespace

Definition: the nominal scope/namespace of the Attribute. It is included in the scopes of all subtypes of the EntityType.

Multiplicity: 1..1

Properties: derived

Tagged Values

derivation = self->of-entity->declared-in

AssociationEnd: of-entity **To: SingleEntityType**

via: attribute-declared-in-entity

Definition: represents the relationship of an Attribute to the SingleEntityType for which it was originally declared.

Multiplicity: 1..1

Note – The derived relationship `.of-entity.declared-in` is the specialization of TypeElement:namespace for Attributes. The EntityType is the namespace for the Attribute, not the SingleEntityType.

AssociationEnd: owning-entity **To: EntityType**

via: entity-has-attributes

Definition: the EntityTypes that have or inherit the Attribute, that is, the EntityType in which the Attribute is declared and all subtypes of that EntityType.

Multiplicity: 1..* unordered

Properties: derived

Note – The derivation of this relationship begins with self->namespace (i.e., self->of-entity->declared-in) and recursively adds all EntityTypes reached by supertype-of.

8.11.1.4 Other Roles

From: UniqueRule as key-component

From: EntityType as attributes

From: Redeclaration as original-attribute

From: Instances::RoleName as refers-to

From: Expressions::AttributeRef as refers-to

From: Expressions::UsedInRef as inverse-of

8.11.1.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every Attribute shall have an Identifier.

8.11.2 Class: DerivedAttribute

Definition: represents an EXPRESS DERIVE attribute = a property whose value can be determined from other attributes and relationships of the entity instance.

Note – See 9.2.1.2 of ISO 10303-11:2004.

8.11.2.1 Supertypes

Attribute

8.11.2.2 Attributes

none.

8.11.2.3 Associations

AssociationEnd: derivation **To: Expression**

Definition: the Expression that specifies how to determine the value of the DerivedAttribute from the values of other Attributes.

Note – See 9.2.1.2 of ISO 10303-11:2004.

Multiplicity: 1..1

8.11.2.4 Other Roles

none.

8.11.3 Class: EntityType

Definition: a NamedType representing an EXPRESS entity data type, a type declared by an entity_declaration.

Note – See 9.2 of ISO 10303-11:2004.

8.11.3.1 Supertypes

InstantiableType, NamedType

8.11.3.2 Attributes

Attribute: isAbstract **To: MOF::Boolean**

Definition: True if the EXPRESS entity data type is declared ABSTRACT in its original declaration, either as ABSTRACT entity or as ABSTRACT SUPERTYPE; False otherwise. The entity data type can also/later be declared “abstract” in a SUBTYPE_CONSTRAINT, e.g., in an interfacing Schema, but that is taken as a constraint on the usage of the EntityType in that context.

Note – See 9.2.4 and 9.2.5.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.11.3.3 Associations

AssociationEnd: attributes **To: Attribute**

via: entity-has-attributes

Definition: represents the relationship between an EntityType and the declared Attributes of that EntityType, including those in the entity declaration and those inherited from supertypes.

Note – See 9.2 of ISO 10303-11:2004.

Properties: derived

Multiplicity: 0..* unordered

TaggedValues

```
derivation = declares.declares + subtype-of.declares.declares
```

AssociationEnd: declares **To: SingleEntityType**

via: single-entity-declared-in-entity

Definition: the SingleEntityType that is declared in the declaration for the EntityType, i.e., the group of Attributes that is named for the EntityType.

Multiplicity: 1..1

Properties: composite

AssociationEnd: extension **To: Instances::Extent**

via: Instances::extent-of-EntityType

Definition: represents the relationship between an EntityType and its extent (the set of corresponding EntityInstances) in a given Population.

Multiplicity: 0..* unordered

AssociationEnd: local-attributes **To: Attribute**

via: EntityType_has_Attribute

Definition: the Attributes that are declared within the entity declaration, that is, the attributes that are declared in the corresponding SingleEntityType.

Subsets: NamedType:type-elements

Multiplicity: 0..* unordered

Properties: derived

Tagged Values

derivation = self->declares->declares

AssociationEnd: instances **To: Instances::EntityInstance**

via: Instances::instance-of-EntityType

Definition: represents the relationship between an EntityType (classifier) and the EntityInstances that satisfy it.

Multiplicity: 0..* unordered

AssociationEnd: plays-domain-role **To: DomainRole**

via: entity-plays-domain-role

Definition: represents the relationship between an entity type and the domain roles that its instances play.

For each InvertibleAttribute of the EntityType, the EntityType plays a corresponding DomainRole. An EntityInstance is considered to play the DomainRole once for each member of an InvertibleAttribute whose data type is an AggregationType.

Properties: derived.

Multiplicity: 0..* unordered

Note – The derivation of this property is complex. For each InvertibleAttribute `x` in self->attributes, the EntityType plays-the-domain-role that is `x->creates-relationship->domain`, i.e., the DomainRole in the Relationship that is created by the InvertibleAttribute `x`.

AssociationEnd: plays-range-role **To: RangeRole**

via: entity-plays-range-role

Definition: represents the relationship between an entity type and the range roles that its instances play. For each occurrence of the EntityType in/as the attribute-type of an InvertibleAttribute, the EntityType plays the corresponding RangeRole.

Properties: derived.

Multiplicity: 0..* unordered

Note – The derivation of plays-range-role is complex. For each InvertibleAttribute that is an instance of self->used-in, a given EntityType plays the RangeRole that is InvertibleAttribute::models-role.

AssociationEnd: redeclarations **To: Redeclaration**

via: scope-of-redeclaration-is-EntityType

Definition: represents the relationship between the EntityType and any attribute Redeclarations that appear in its declaration.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: subtype-of **To: EntityType**

Definition: represents the relationship of an entity data type to its immediate supertypes – those entity data types from whose common domain the instances of the EntityType are drawn. For compatibility with the interpretation of other features of EXPRESS, this relationship extends only to those EntityTypes that are “immediate supertypes,” i.e., those explicitly declared in the SUBTYPE OF clause for this EntityType.

Note – See 9.2.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: unique-rules **To: UniqueRule**

via: EntityType-has-UniqueRule

Subsets: NamedType:type-elements

Definition: represents the relationship between an EntityType and the local uniqueness rules that constrain the values of attributes of that EntityType

Note – See 9.2.2.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: used-in **To: InvertibleAttribute**

via: entity-used-in-attribute

Definition: represents the relationship between the EntityType and the InvertibleAttributes (of other EntityTypes) that establish relationships to it.

Multiplicity: 0..* unordered

8.11.3.4 Other Roles

From: Rules::SupertypeRule as named-supertype

From: EntityType as subtype-of

From: Instances::EntityValue as corresponds to

From: Instances::SingleLeafInstance as characterizing-type

8.11.4 Class: ExplicitAttribute

Definition: represents an EXPRESS “explicit” attribute, a model of a property of an entity instance that is not, in general, derived from other properties of that instance or other entity instances.

Note – See 9.2.1.1 of ISO 10303-11:2004.

8.11.4.1 Supertypes

Attribute

8.11.4.2 Attributes

Attribute: isOptional **To: MOF::Boolean**

Definition: True if the entity instance is permitted to have no specified value for this attribute; False if a value for this attribute is required.

Note – See 9.2.1.1 of ISO 10303-11:2004.

Multiplicity: 1..1

8.11.4.3 Associations

none.

8.11.4.4 Other Roles

From: Expressions::AttributeBinding as attribute

From: Instances::AttributeValue as attribute

From: Statements::AttributeObject as refers-to

8.11.5 Class: InverseAttribute

Definition: represents an EXPRESS INVERSE attribute = a property of each instance of this entity data type that represents a relationship between it and instances of some other entity data type, created by an InvertibleAttribute of that entity data type.

Note – See 9.2.1.3 of ISO 10303-11:2004.

8.11.5.1 Supertypes

Attribute

8.11.5.2 Attributes

Attribute: isUnique **To: MOF::Boolean**

Definition: True if the designated relationship between this instance and any given instance can occur at most once; False if it can occur more than once.

(True if the attribute-type of the INVERSE attribute is declared to be an entity data type or a SET; False if it is declared to be a BAG.)

Note – See 9.2.1.3 of ISO 10303-11:2004.

Multiplicity: 1..1

8.11.5.3 Associations

AssociationEnd: explicit **To: InvertibleAttribute**

via: InverseAttribute-inverts-ExplicitAttribute

Definition: represents the relationship of an inverse attribute of one entity data type to the explicit attribute (InvertibleAttribute) of another entity data type that models the Relationship from which the inverse attribute is derived.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: models-role **To: DomainRole**

via: InverseAttribute-models-role

Definition: represents the relationship between an Inverse Attribute and the domain-role it defines. By extension (`models-role:in-relationship`), it models the relationship of the inverse attribute to the Relationship it denotes.

Multiplicity: 1..1

8.11.5.4 Other Roles

none.

8.11.6 Class: InvertibleAttribute

Definition: An ExplicitAttribute whose attribute type is one of:

- an EntityType
- a SelectType whose select-list consists of EntityTypes
- an AggregationType whose member-type is either of the above

An InvertibleAttribute models a Relationship between two EntityTypes – the EntityType that declares the InvertibleAttribute, and the EntityType that appears in its attribute-type.

An InvertibleAttribute whose attribute-type (or its member-type) is a SelectType defines one Relationship for each EntityType in the select-list.

Note – See ISO 10303-11.2:2004 clause 9.2.1.3

8.11.6.1 Supertypes

ExplicitAttribute

8.11.6.2 Attributes

none.

8.11.6.3 Associations

AssociationEnd: creates-relationship **To: Relationship**

via: InvertibleAttribute-creates-relationship

Definition: represents the relationship between an InvertibleAttribute and the Relationship between EntityTypes that it models.

Multiplicity: 1..1

AssociationEnd: inverse **To: InverseAttribute**

via: InverseAttribute-inverts-ExplicitAttribute

Definition: represents the relationship of an explicit attribute denoting a Relationship to the inverse attribute of the range entity data type that models the same Relationship. While the inverse is conceptually unique, EXPRESS allows it to be declared differently in different subtypes of the original range entity.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: models-role **To: RangeRole**

via: InvertibleAttribute-models-role

Definition: represents the relationship between an Explicit Attribute and the RangeRole it defines.

Note – An explicit attribute defines a RangeRole (and thus a Relationship) if and only if it is an InvertibleAttribute.

Multiplicity: 0..1

AssociationEnd: range-type **To: EntityType**

via: entity-used-in-attribute

Definition: models the relationship between the InvertibleAttribute and the EntityTypes that are, or are members of, its attribute-type. These EntityTypes are the “range” of the Relationship with the “referencing” entity that is created by the InvertibleAttribute.

Multiplicity: 1..* unordered

8.11.6.4 Other Roles

none.

8.11.7 Class: PartialEntityType

Definition: a-DataType representing a collection of SingleEntityTypes. A PartialEntityType is the data type of a PartialEntityValue.

Note – See 9.2.6 of ISO 10303-11:2004.

8.11.7.1 Supertypes

DataType

8.11.7.2 Attributes

none.

8.11.7.3 Associations

AssociationEnd: components **To: SingleEntityType**

Definition: represents the relationship between the PartialEntityValue and the SingleEntityValues that make it up.

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

8.11.7.4 Other Roles

From: SingleEntityType as equivalent

From: Instances::PartialEntityValue as of-type

8.11.8 Class: SingleEntityType

Definition: the group of Attributes of a given EntityType that appear directly in the entity_declaration for that EntityType, i.e., excluding “inherited” attributes. A SingleEntityType corresponds to, and has the same id as, the EntityType whose declaration defines it.

Note – A SingleEntityType is not a DataType; it cannot be the type of an Expression result or of any other EXPRESS concept. It is only the “type” of SingleEntityValues, and they are not Instances.

Note – See 3.3.9 of ISO 10303-11:2004 (should be corrected by TC#1).

8.11.8.1 Supertypes

none.

8.11.8.2 Attributes

Attribute: id **To: ScopedId**

Definition: Represents the EXPRESS Identifier for the SingleEntityType, which is the same as the Identifier for the corresponding EntityType.

Properties: derived.

Multiplicity: 1..1

TaggedValues

```
derivation = self->derived-from->id
```

8.11.8.3 Associations

AssociationEnd: declares **To: Attribute**

via: attribute-declared-in-entity

Definition: represents the relationship between a SingleEntityType and the Attributes declared in the entity declaration for the corresponding EntityType..

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: derived-from **To: EntityType**

via: single-entity-declared-in-entity

Definition: represents the derivation of the SingleEntityType from the entity_declaration for the EntityType.

Multiplicity: 1..1

AssociationEnd: equivalent **To: PartialEntityType**

Definition: represents the relationship between the SingleEntityType and the “equivalent” PartialEntityType, namely, the PartialEntityType that consists of exactly that one SingleEntityType. For those PartialEntityTypes that are equivalent to SingleEntityTypes, the PartialEntityType:includes relationship is the inverse of this relationship.

Multiplicity: 1..1

8.11.8.4 Other Roles

From: PartialEntityType as components

From: Instances::SingleEntityValue as of-type

From: Expressions::GroupRef as refers-to

From: Expressions::PartialEntityConstructor as attribute-group

From: Statements::GroupObject as refers-to

8.11.8.5 Rules

Constraint (OCL)

```
sizeof(self->equivalent->includes) = 1
```

Constraint (OCL)

```
self->equivalent->includes[1] = self
```

8.11.9 Class: UniqueRule

Definition: represents an EXPRESS UNIQUE rule = a requirement that the combination of values of the specified “key” attributes be unique over all instances of the entity data type in a given Population.

Note – See 9.2.2.1 of ISO 10303-11:2004.

8.11.9.1 Supertypes

TypeElement

8.11.9.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: Represents the position of the Unique Rule in the list of rules following the UNIQUE keyword in the entity/type declaration.

Multiplicity: 1..1

8.11.9.3 Associations

AssociationEnd: domain **To: EntityType**

via: EntityType-has-UniqueRule

Subsets: Core::TypeElement:namespace

Definition: represents the relationship of the UniqueRule to the EntityType whose Extent is the domain of values to which it applies.

Multiplicity: 1..1

AssociationEnd: key-component **To: Attribute**

Definition: represents the relationship between the UniqueRule and the “key” attributes of the (possibly joint) key for the instances of the EntityType.

Note – See 9.2.2.1 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

8.11.9.4 Other Roles

none.

8.11.10 Association: attribute-declared-in-entity

Definition: represents the relationship between a SingleEntityType and the Attributes declared in the entity declaration for the corresponding EntityType.

8.11.10.1 Association Ends

AssociationEnd: declares **To: Attribute**

Definition: represents the relationship between a SingleEntityType and the Attributes declared in the entity declaration for the corresponding EntityType.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: of-entity **To: SingleEntityType**

Definition: represents the relationship of an Attribute to the SingleEntityType for which it was originally declared.

Multiplicity: 1..1

8.11.11 Association: attribute-has-data-type

Definition: represents the relationship between an Attribute and the ParameterType that characterizes all values of the Attribute.

Note – See 9.2.1 of ISO 10303-11:2004.

8.11.11.1 Association Ends

AssociationEnd: attribute-type **To: ParameterType**

Definition: represents the required data type for all values of that Attribute in all instances of the EntityType. The attribute-type is required to be an InstantiableType unless either:

- isAbstract is True for the EntityType, in which case the attribute-type may be a GeneralizedType, or
- the EntityType is defined in an AlgorithmScope (instead of a Schema), in which case the attribute-type may be an ActualType.

Note – See 9.2.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: role **To: Attribute**

Definition: represents the relationship between the ParameterType and the roles (attributes of entities) that its admissible values may play.

Note – See 9.2.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.11.12 Association: entity-has-attributes

Definition: represents the relationship between an EntityType and all of the Attributes that are associated with every instance of the EntityType, including instances of any of its subtypes. That is, this association relates an EntityType to the Attributes declared in the corresponding SingleEntityType and to all the Attributes declared in the SingleEntityTypes that correspond to its supertypes.

Properties: derived

8.11.12.1 Association Ends

AssociationEnd: attributes **To: Attribute**

Definition: represents the relationship between an EntityType and the declared Attributes of that EntityType, including those in the entity declaration and those inherited from supertypes.

Note – See 9.2 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: derived

Note – The derivation of this relationship is recursive, using e->subtype-of, beginning with e = self and adding the attributes of e->declares->declares for each e.

AssociationEnd: owning-entity **To: EntityType**

Definition: the EntityTypes that have or inherit the Attribute, that is, the EntityType in which the Attribute is declared and all subtypes of that EntityType.

Multiplicity: 1..* unordered

Properties: derived

Note – The derivation of this relationship begins with self->namespace (i.e., self->of-entity->declared-in) and recursively adds all EntityTypes reached by supertype-of.

8.11.13 Association: EntityType-has-Attribute

Definition: represents the relationship between an EntityType and the Attributes that are declared within the entity declaration, that is, the attributes that are declared in the corresponding SingleEntityType.

Note – This is a derived association that refines the type-element-has-scope relationship for Attribute.

8.11.13.1 Supertypes

type-element-has-scope

8.11.13.2 Association Ends

AssociationEnd: local-attributes **To: Attribute**

Definition: the Attributes that are declared within the entity declaration, that is, the attributes that are declared in the corresponding SingleEntityType.

Subsets: NamedType:type-elements

Multiplicity: 0..* unordered

Properties: derived

Tagged Values

```
derivation = self->declares->declares
```

AssociationEnd: namespace **To: EntityType**

Definition: the nominal scope/namespace of the Attribute. It is included in the scopes of all subtypes of the EntityType.

Subsets: TypeElement:namespace

Multiplicity: 1..1

Properties: derived

Tagged Values

derivation = self->of-entity->declared-in

8.11.14 Association: EntityType-has-UniqueRule

Definition: represents the relationship between an EntityType and the local uniqueness rules that constrain the values of attributes of that EntityType.

8.11.14.1 Supertypes

type-element-has-scope

8.11.14.2 Association Ends

AssociationEnd: domain **To: EntityType**

Definition: represents the relationship of the UniqueRule to the EntityType whose Extent is the domain of values to which it applies.

Multiplicity: 1..1

AssociationEnd: unique-rules **To: UniqueRule**

Definition: represents the relationship between an EntityType and the local uniqueness rules that constrain the values of attributes of that EntityType.

Note – See 9.2.2.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

8.11.15 Association: InverseAttribute-inverts-ExplicitAttribute

Definition: represents the relationship of an INVERSE attribute of one entity data type to the explicit attribute (InvertibleAttribute) of the entity data type that models the Relationship from which the inverse attribute is derived.

8.11.15.1 Association Ends

AssociationEnd: explicit **To: InvertibleAttribute**

Definition: the explicit attribute (InvertibleAttribute) of the associated entity data type that models the Relationship from which the inverse attribute is derived.

Note – The attribute-type of the InverseAttribute may be a subtype of the entity data type that defines the InvertibleAttribute.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: inverse **To: InverseAttribute**

Definition: represents the relationship of an explicit attribute denoting a Relationship to the inverse attribute of the range entity data type that models the same Relationship. While the inverse is conceptually unique, EXPRESS allows it to be declared differently in different subtypes of the original range entity.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.11.16 Association: single-entity-declared-in-entity

Definition: represents the relationship between the EntityType and the SingleEntityType that is implicitly declared in the entity_declaration for the EntityType.

8.11.16.1 Association Ends

AssociationEnd: declares **To: SingleEntityType**

Definition: the SingleEntityType that is declared in the declaration for the EntityType, i.e., the group of Attributes that is named for the EntityType.

Multiplicity: 1..1

AssociationEnd: declared-in **To: EntityType**

Definition: represents the derivation of the SingleEntityType from the entity_declaration for the EntityType.

Multiplicity: 1..1

8.12 Relationships

According to ISO 10303-11, a “distributive relationship” between entity data types is modeled by an attribute whose data type is either an entity type or an aggregation type whose member type is an entity type. This section models the “distributive relationship” concepts.

Note – The primary purpose of this subclause is to facilitate mappings to languages in which relationships, also called “associations” or “properties,” are first-class concepts from which the associated “attributes” are derived.

In EXPRESS, all relationships are directed. The entity type that is the “domain” of the relationship has an explicit attribute – an InvertibleAttribute – that denotes the relationship; the entity type that is the “range” of the relationship may have an inverse attribute that denotes the relationship, but EXPRESS always supports an implicit inverse attribute via the UsedIn function (see 12.5.3).

Figure 14 shows these concepts, and their relationship to the Attribute concepts. They are described in detail below.

Note – In Figure 14, the «implicit» relationship entity-has-attributes represents the derived association entity-has-attributes defined in 8.12.3.3, but restricted to InvertibleAttribute, which is a subclass of Attribute.

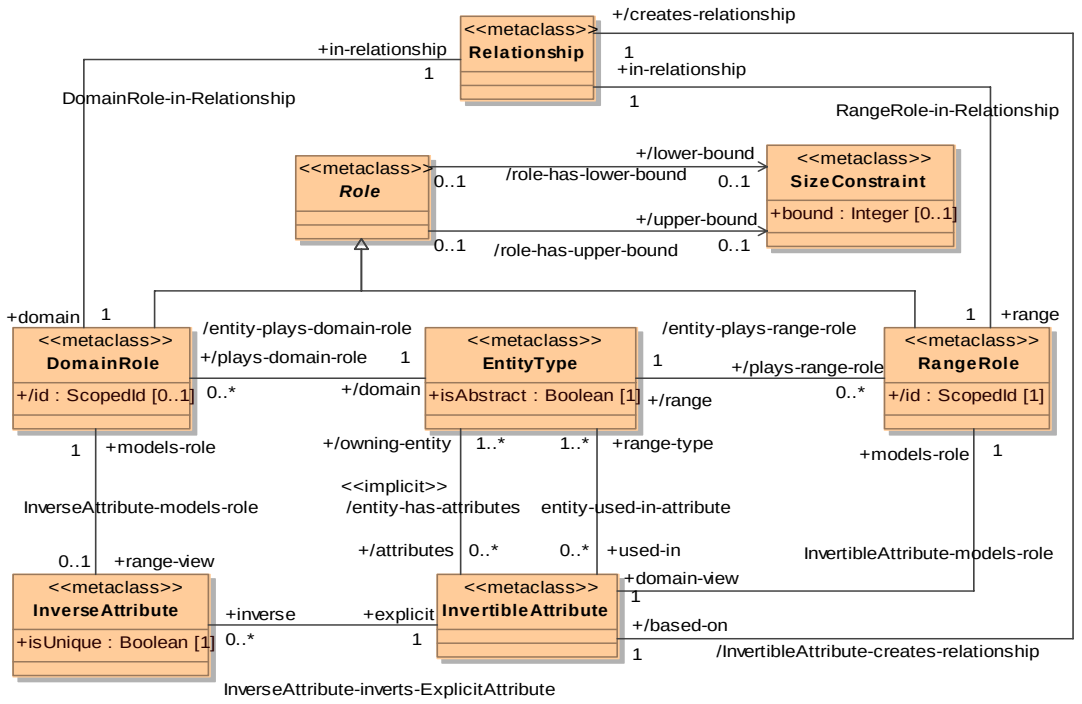

Figure 14 - Relationships

8.12.1 Class: DomainRole

Definition: a role representing the behavior of the entity instances that is designated the “domain” of the relationship.

8.12.1.1 Supertypes

Role

8.12.1.2 Attributes

Attribute: id **To: ScopedId**

Definition: Represents the “complete” identifier for the Role. The identifier for the DomainRole is derived from the identifier for the InverseAttribute, when present, including the Identifier value and the associated EntityType identifier. When there is no

InverseAttribute, :id has no proper value, but the DomainRole may be identified by the pseudo-identifier: UsedIn.<RangeRole:id>, where <RangeRol:id> is the identifier for the RangeRole in the Relationship.

Properties: derived.

Multiplicity: 0..1

TaggedValues

```
derivation = self->range-view->id
```

8.12.1.3 Associations

AssociationEnd: domain **To: EntityType**

via: entity-plays-domain-role

Definition: represents the (single) entity data type common to all instances that play the Domain Role.

Properties: derived.

Multiplicity: 1..1

TaggedValues

```
derivation = self->in-relationship->range->domain-view->of-entity
```

AssociationEnd: in-relationship **To: Relationship**

via: DomainRole-in-Relationship

Definition: represents the relationship between a Domain Role and the (unique) Relationship in which it is defined

Multiplicity: 1..1

AssociationEnd: range-view **To: InverseAttribute**

via: InverseAttribute-models-role

Definition: represents the relationship between a domain-role and the inverse attributes of the range entities that model it. Different subtypes of the primary “range” entity data type can define different views of (and constraints on) the domain role. The “range” entity has an inverse attribute that defines the “domain” role (the role of the other entity).

Multiplicity: 0..1

8.12.1.4 Other Roles

none.

8.12.2 Class: RangeRole

Definition: a role representing the behavior of the entity instances that is designated the “range” of the relationship.

8.12.2.1 Supertypes

Role

8.12.2.2 Attributes

Attribute: id **To: ScopedId**

Definition: Represents the “complete” identifier for the Role. The identifier for a RangeRole is derived from the identifier for the ExplicitAttribute that creates the relationship, including the Identifier value and the associated EntityType identifier.

Properties: derived.

Multiplicity: 1..1

TaggedValues

```
derivation = self->domain-view->id
```

8.12.2.3 Associations

AssociationEnd: domain-view **To: InvertibleAttribute**

via: InvertibleAttribute-models-role

Definition: represents the relationship between a RangeRole and the InvertibleAttribute of the domain/referencing entity that models it.

Multiplicity: 1..1

AssociationEnd: in-relationship **To: Relationship**

via: RangeRole-in-Relationship

Definition: represents the relationship between a Range Role and the (unique) Relationship in which it is defined.

Multiplicity: 1..1

AssociationEnd: range **To: EntityType**

via: entity-plays-range-role

Definition: represents the (single) entity data type common to all instances that play the Range Role.

Properties: derived.

Multiplicity: 1..1

TaggedValues

```
derivation = self->domain-view->attribute-type
```

8.12.2.4 Other Roles

none.

8.12.3 Class: Relationship

Definition: a “distributive relationship” between entity data types.

Every InvertibleAttribute creates a Relationship between two EntityTypes and creates two Roles -- one for each participating EntityType. All relationships are directed. The InvertibleAttribute is an explicit attribute of the EntityType that plays the DomainRole; the range-type of the InvertibleAttribute is the EntityType that plays the RangeRole.

The range-type may have an inverse attribute denoting the DomainRole; or the DomainRole may be referred to by the UsedIn function (see 12.5.3).

8.12.3.1 Supertypes

none.

8.12.3.2 Attributes

none.

8.12.3.3 Associations

AssociationEnd: based-on **To: InvertibleAttribute**

via: InvertibleAttribute-creates-relationship

Definition: represents the relationship between a Relationship and the InvertibleAttribute on which it is based, i.e., the Attribute that creates the Relationship.

Multiplicity: 1..1

AssociationEnd: domain **To: DomainRole**

via: DomainRole-in-Relationship

Definition: represents the relationship between the Relationship and the Role that is its DomainRole.

Multiplicity: 1..1

AssociationEnd: range **To: RangeRole**

via: RangeRole-in-Relationship

Definition: represents the relationship between the Relationship and its “range” role.

Multiplicity: 1..1

8.12.3.4 Other Roles

none.

8.12.4 Class: Role

Definition: a “slot” in a relationship, denoting the behavior of one of the Instances involved in the relationship. Since all

relationships in EXPRESS are directed, the two slots are nominally designated domain and range.

Properties: abstract

8.12.4.1 Supertypes

none.

8.12.4.2 Attributes

none.

8.12.4.3 Associations

AssociationEnd: lower-bound **To: SizeConstraint**

Definition: represents a lower-bound on the number of Relationship instances in which a given EntityInstance can play this Role. An explicit zero ("0") value may be considered to represent no lower-bound constraint; and the lower-bound relationship need not appear. A lower-bound expression that may evaluate to zero shall always be represented by a lower-bound relationship.

Note – The lower-bound on the Domain role is specified by the Explicit Attribute that models the RangeRole. The lower-bound on the Range role is specified by the Inverse Attribute that models the Domain Role, if any, or possibly by a DomainRule on the “range” EntityType involving UsedIn(SELF,).

Note – Because the ExplicitAttribute that creates the Relationship may have an aggregation data type for which isUnique does not hold, a given pair of participating entity instances may occur more than once as an instance of the Relationship. The Size constraint is on the count of pairs, not the count of distinct pairs.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Properties: derived.

Multiplicity: 0..1

AssociationEnd: upper-bound **To: SizeConstraint**

Definition: represents an upper-bound on the number of Relationship instances in which a given EntityInstance can play the Role. An explicit indeterminate value ("?") is considered to represent no upper-bound constraint, and shall not be represented by an upper-bound relationship. (An upper-bound expression that may evaluate to "?" shall be represented by an upper-bound relationship.)

Note – The upper-bound on the Domain role is specified by the Explicit Attribute that models the RangeRole. The upper-bound on the Range role is specified by the Inverse Attribute that models the Domain Role, if any, or possibly by a DomainRule on the “range” EntityType involving UsedIn(SELF,).

Note – Because the ExplicitAttribute that creates the Relationship may have an aggregation data type for which isUnique does not hold, a given pair of participating entity instances may occur more than once as an instance of the Relationship. The Size constraint is on the count of pairs, not the count of distinct pairs.

Note – See 9.2.1.3 of ISO 10303-11:2004.

Properties: derived.

Multiplicity: 0..1

8.12.4.4 Other Roles

From: Redeclaration as refined-role

8.12.5 Association: DomainRole-in-Relationship

Definition: represents the relationship between the Relationship and the Role that is its DomainRole.

8.12.5.1 Association Ends

AssociationEnd: domain **To: DomainRole**

Definition: represents the relationship between the Relationship and the Role that is its DomainRole.

Multiplicity: 1..1

AssociationEnd: in-relationship **To: Relationship**

Definition: represents the relationship between a Domain Role and the (unique) Relationship in which it is defined.

Multiplicity: 1..1

8.12.6 Association: entity-plays-domain-role

Definition: represents the relationship between an entity type and the domain roles that its instances play.

Properties: derived

8.12.6.1 Association Ends

AssociationEnd: domain **To: EntityType**

Definition: represents the (single) entity data type common to all instances that play the Domain Role.

Multiplicity: 1..1

Properties: derived

TaggedValues

```
derivation = self->in-relationship->range->domain-view->of-entity
```

AssociationEnd: plays-domain-role **To: DomainRole**

Definition: represents the relationship between an entity type and the domain roles that its instances play.

For each InvertibleAttribute of the EntityType, the EntityType plays a corresponding DomainRole. An EntityInstance is considered to play the DomainRole once for each member of an InvertibleAttribute whose data type is an AggregationType.

Multiplicity: 0..* unordered

Properties: derived

TaggedValues

```
derivation = ((self->attributes) * extent(InvertibleAttribute))->
      creates-relationship->domain
```

8.12.7 Association: entity-plays-range-role

Definition: represents the relationship between an entity type and the range roles that its instances play.

Properties: derived

8.12.7.1 Association Ends

AssociationEnd: plays-range-role **To: RangeRole**

Definition: represents the relationship between an entity type and the range roles that its instances play.

For each occurrence of the EntityType as the attribute-type, or a member of the attribute-type, of an explicit attribute (InvertibleAttribute), the EntityType plays the corresponding RangeRole (.models-role).

Multiplicity: 0..* unordered

Properties: derived.

Note – The derivation of plays-range-role is complex. For each InvertibleAttribute that is an instance of `self->used-in`, a given EntityType plays the RangeRole that is `InvertibleAttribute::models-role`.

AssociationEnd: range **To: EntityType**

Definition: represents the (single) entity data type common to all instances that play the Range Role.

Multiplicity: 1..1

Properties: derived.

TaggedValues

```
derivation = self->domain-view->attribute-type
```

8.12.8 Association: entity-used-in-attribute

Definition: represents the relationship between the EntityType and the InvertibleAttributes (of other EntityTypes) that establish relationships to it.

8.12.8.1 Association Ends

AssociationEnd: range-type **To: EntityType**

Definition: models the relationship between the InvertibleAttribute and the EntityTypes that are, or are members of, its attribute-type. These EntityTypes are the “range” of the Relationship with the “referencing” entity that is created by the InvertibleAttribute.

Multiplicity: 1..* unordered

AssociationEnd: used-in **To: InvertibleAttribute**

Definition: represents the relationship between the EntityType and the InvertibleAttributes (of other EntityTypes) that establish relationships to it.

Multiplicity: 0..* unordered

8.12.9 Association: InverseAttribute-models-role

Definition: represents the relationship between an Inverse Attribute and the domain-role it refers to.

8.12.9.1 Association Ends

AssociationEnd: models-role **To: DomainRole**

Definition: represents the relationship between an Inverse Attribute and the domain-role it defines. By extension (`models-role:in-relationship`), it models the relationship of the inverse attribute to the Relationship it denotes.

Multiplicity: 1..1

AssociationEnd: range-view **To: InverseAttribute**

Definition: represents the relationship between a domain-role and the inverse attributes of the range entities that model it. Different subtypes of the primary “range” entity data type can define different views of (and constraints on) the domain role. The “range” entity has an inverse attribute that defines the “domain” role (the role of the other entity).

Multiplicity: 0..1

8.12.10 Association: InvertibleAttribute-creates-relationship

Definition: represents the relationship between an InvertibleAttribute and the Relatiionship between EntityTypes that it models.

8.12.10.1 Association Ends

AssociationEnd: based-on **To: InvertibleAttribute**

Definition: represents the relationship between a Relationship and the InvertibleAttribute on which it is based, i.e., the Attribute that creates the Relationship.

Multiplicity: 1..1

AssociationEnd: creates-relationship **To: Relationship**

Definition: represents the relationship between an InvertibleAttribute and the Relatiionship between EntityTypes that it models.

Multiplicity: 1..1

8.12.11 Association: InvertibleAttribute-models-role

Definition: represents the relationship between an Invertible Attribute and the RangeRole it defines.

8.12.11.1 Association Ends

AssociationEnd: domain-view **To: InvertibleAttribute**

Definition: represents the relationship between a RangeRole and the InvertibleAttribute of the domain/referencing entity that models it.

Multiplicity: 1..1

AssociationEnd: models-role **To: RangeRole**

Definition: represents the relationship between an Explicit Attribute and the RangeRole it defines.

Note – An explicit attribute defines a RangeRole (and thus a Relationship) if and only if it is an InvertibleAttribute.

Multiplicity: 0..1

8.12.12 Association: RangeRole-in-Relationship

Definition: represents the relationship between a Range Role and the (unique) Relationship in which it is defined.

8.12.12.1 Association Ends

AssociationEnd: in-relationship **To: Relationship**

Definition: represents the relationship between a Range Role and the (unique) Relationship in which it is defined.

Multiplicity: 1..1

AssociationEnd: range **To: RangeRole**

Definition: represents the relationship between the Relationship and its “range” role.

Multiplicity: 1..1

8.13 Redeclarations

Redeclaration is an EXPRESS mechanism that permits a subtype to “redeclare” an inherited attribute in order to constrain its possible values in instances of the subtype. Figure 15 shows the model of this concept, and this section defines the related metamodel elements.

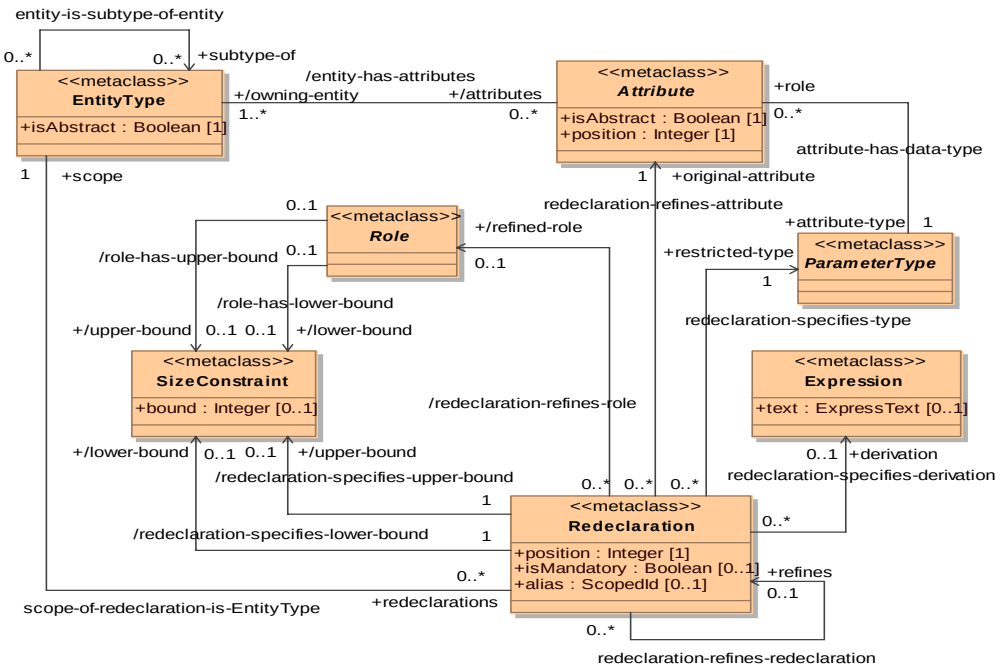

Figure 15 - Redeclarations

8.13.1 Class: Redeclaration

Definition: represents the “redeclaration” of an EXPRESS attribute in a subtype of the entity data type for which that attribute was originally declared. A redeclaration represents a refinement of the original attribute concept in the subtype, and it states corresponding constraints on the possible values of that attribute in the subtype. It may also rename the attribute for the subtype. When the attribute-type of the original-attribute is an EntityType, the Redeclaration may be seen as refining the RangeRole represented by the original-attribute for the domain restricted to the subtype.

Note – See 9.2.3.4 of ISO 10303-11:2004.

8.13.1.1 Supertypes

none.

8.13.1.2 Attributes

Attribute: alias **To: ScopedId**

Definition: an additional EXPRESS identifier that may be used to identify the original attribute in this subtype.

Note – See 9.2.2.2 of ISO 10303-11:2004.

Multiplicity: 0..1

Attribute: isMandatory **To: MOF::Boolean**

Definition: True if the entity instance is required to have a value for this attribute in this subtype; False if it is permitted to have no specified value. This attribute is only present if isOptional is True for the original attribute.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 0..1

Attribute: position **To: MOF::Integer**

Definition: Represents the position of the redeclaration in the sequence of attribute declarations in the entity declaration. By convention these follow all the new attribute declarations of each kind.

Multiplicity: 1..1

8.13.1.3 Associations

AssociationEnd: derivation **To: Expression**

Definition: When specified, represents a Redeclaration that redeclares an ExplicitAttribute to be “derived” in the .scope subtype. That is, it declares an Expression that can be used to derive (or validate) the value of the redeclared Attribute in this subtype.

Multiplicity: 0..1

AssociationEnd: lower-bound **To: SizeConstraint**

Definition: represents the minimum cardinality of the role that is stated by the Redeclaration. This is the case when the Redeclaration redeclares the ParameterType to restrict the minimum size of the aggregate values.

When the restricted-type is an AggregationType, the lower-bound SizeConstraint is the lower-bound of that AggregationType.

Multiplicity: 0..1

Properties: derived.

AssociationEnd: original-attribute **To: Attribute**

Definition: identifies the original Attribute being redeclared by the Redeclaration. If the Redeclaration redeclares another redeclared-attribute (see .refines), the .original-attribute is determined transitively. Every Redeclaration ultimately constrains an original attribute in some supertype.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: refined-role **To: Role**

Definition: represents the relationship between a Redeclaration and the Role represented by the .original-attribute.

If the Redeclaration redeclares an InvertibleAttribute, it refines the corresponding RangeRole by restricting the allowable participants in the RangeRole for the domain that is the `.scope` of the Redeclaration. If the Redeclaration redeclares an

InverseAttribute, it refines the corresponding DomainRole by restricting the allowable participants in the DomainRole for the range that is the `.scope` of the Redeclaration.

Multiplicity: 0..1

Properties: derived.

AssociationEnd: refines **To: Redeclaration**

Definition: This relationship is present only when a Redeclaration is stated as a refinement of an attribute of a subtype that itself redeclares that attribute. `.refines` refers to the Redeclaration that represents that redeclared attribute.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: restricted-type **To: ParameterType**

Definition: when specified, specifies the subtype or specialization of the data type of the original attribute to which all values of the original attribute in instances of the “scope” EntityType must conform.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: scope **To: EntityType**

via: scope-of-redeclaration-is-EntityType

Definition: represents the relationship between the Redeclaration and the entity data type to which the redeclaration applies. Values for the original attribute are constrained by the Redeclaration for instances of the .scope EntityType and all of its subtypes. The .scope EntityType is the namespace of the .alias identifier, if present.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: upper-bound **To: SizeConstraint**

Definition: represents a restriction on the maximum cardinality of the role that is stated by the Redeclaration. This is the case when the Redeclaration redeclares the ParameterType to restrict the maximum size of the aggregate values.

When the restricted-type is an AggregationType, the upper-bound SizeConstraint is the upper-bound of that AggregationType.

Multiplicity: 0..1

Properties: derived.

8.13.1.4 Other Roles

From: Redeclaration as refines

8.13.2 Association: scope-of-redeclaration-is-EntityType

Definition: represents the relationship between the Redeclaration and the entity data type to which the redeclaration applies.

8.13.2.1 Association Ends

AssociationEnd: redeclarations **To: Redeclaration**

Definition: represents the relationship between the EntityType and any attribute Redeclarations that appear in its declaration.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: scope **To: EntityType**

Definition: the entity data type to which the redeclaration applies.

Values for the original attribute are constrained by the Redeclaration for instances of the .scope EntityType and all of its subtypes. The .scope EntityType is the namespace of the .alias identifier, if present.

Note – See 9.2.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

8.14 Parametric Datatype Elements

EXPRESS permits the `generalized_type` specifications for formal parameters and attributes of abstract entity data types to contain labeled components that refer to specific elements of the data type of the corresponding actual parameters and instantiable entity subtypes. These labeled components are modeled as ParametricElements. In the declarations of other attributes of the abstract entity data type or other parameters of the same Algorithm, ActualTypeConstraints refer to these ParametricElements. In Algorithm bodies, the specifications of data types that are ActualTypes refer to ParametricElements defined in the formal parameter list. Figure 16 depicts the ParametricElement concepts.

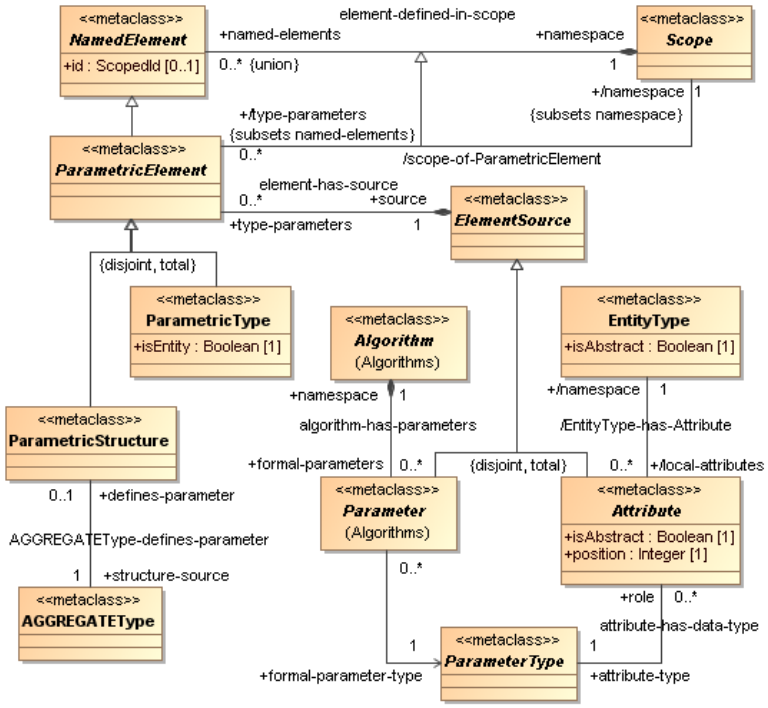

Figure 16 - Parametric Datatype Elements

Note – In the diagram the model elements that are taken from the Algorithms Package can be ignored if only the Core package is being implemented.

8.14.1 Class: ElementSource

Definition: an Attribute or Parameter. ElementSource reifies the roles of Attribute and Parameter as `:source` of ParametricElements – the container of their declarations – and as owner of ActualTypeConstraints and ActualStructureConstraints.

Note – InverseAttributes cannot have values for any of the properties of ElementSource - type-parameters, type-constraints, structure-constraints.

Properties: abstract

8.14.1.1 Supertypes

none

8.14.1.2 Attributes

none.

8.14.1.3 Associations

AssociationEnd: structure-constraints **To: ActualStructureConstraint**

Definition: the ActualStructureConstraints, if any, that constrain the allowable data types of the corresponding actual parameter.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: type-constraints **To: ActualTypeConstraint**

Definition: the ActualTypeConstraints, if any, that constrain the allowable data types of the corresponding actual parameter.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

AssociationEnd: type-parameters **To: ParametricElement**

via: element-has-source

Definition: the ParametricElements, if any, whose declarations are contained in the declared type of the ElementSource (Attribute or Parameter).

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

8.14.2 Class: ParametricElement

Definition: a NamedElement representing a parametric data type – a component of the type description for an abstract Attribute or a formal Parameter that refers to the corresponding type component of the InstantiableType or the corresponding actual parameter. The ParametricElement is denoted by an EXPRESS `type_label` that is unique within the scope of the EntityType or Algorithm.

The `:id` attribute of the ParametricElement represents the EXPRESS `type_label`.

In EXPRESS `entity_declarations`, the first occurrence of the `type_label` among the Attribute declarations defines the ParametricElement. Any later occurrence of the same `type_label` in the Attribute declarations for the same EntityType (even for the same Attribute) specifies an ActualStructureConstraint or an ActualTypeConstraint that is based on the ParametricElement. The `:source` property indicates the Attribute whose data type contains the ParametricElement definition.

In EXPRESS Algorithm declarations, the first occurrence of the `type_label` in the formal parameter list defines the ParametricElement. Any later occurrence of the same `type_label` in the formal parameter list (even in the same Parameter) specifies an ActualStructureConstraint or an ActualTypeConstraint that is based on the ParametricElement. The `:source` property indicates the Parameter whose formal parameter type contains the ParametricElement definition.

Note – An EXPRESS `type_label` is not part of the model of a GenericType or AGGREGATEType; it is an identifier for a ParametricElement that can be used in ActualTypes and ActualTypeConstraints.

Properties: abstract

8.14.2.1 Supertypes

NamedElement

8.14.2.2 Attributes

none.

8.14.2.3 Associations

AssociationEnd: namespace **To: Scope**

Definition: the EntityType or Algorithm that is the namespace of the ScopedId that is the `type_label`. This relationship is derived – the namespace of a ParametricElement is the same as the namespace of its `:source` element (Attribute or Parameter).

Multiplicity: 1..1

Properties: derived.

Note – While the derivation has the conceptual form: `self->source->namespace` in all cases, each kind of `source` inherits its `namespace` association from a different supertype.

AssociationEnd: source **To: ElementSource**

via: element-has-source

Definition: the ElementSource (Attribute or Parameter) whose declared type is or includes the ParametricElement and defines its `type_label`. The first (by `:position`) Attribute or Parameter whose declared type contains the `type_label` is the source for that ParametricElement and defines the `type_label` as its `:id`.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

8.14.2.4 Other Roles

none.

8.14.3 Class: ParametricStructure

Definition: A syntactic occurrence of AGGREGATE within a formal parameter type or attribute type that defines a `type_label`. The ParametricStructure is the first occurrence of the label among the Parameters of the Algorithm or the Attributes of the EntityType.

In an Attribute declaration, the ParametricStructure refers to the structure (ARRAY, BAG, LIST, SET) of the corresponding instantiable AggregationType in corresponding Attributes of subtypes. In a Parameter declaration, the ParametricStructure refers to the structure of the corresponding component of the corresponding ActualParameters.

Note – Later occurrences of the `type_label` in the same Scope are ActualStructureConstraints.

Example -- In the EXPRESS declaration:

```
FUNCTION check_properties(inputs: AGGREGATE:ins OF property, selectors: AGGRE-
      GATE:ins OF BOOLEAN): BOOLEAN;
```

the `AGGREGATE:ins` in the `inputs` parameter declares both an AGGREGATEType component of the formal-parameter-type of the `inputs` Parameter and a ParametricStructure that defines the `type_label` "`ins`". The `:source-structure` of the ParametricStructure is that AGGREGATEType.
(The `AGGREGATE:ins` in the `selectors` Parameter declares an AGGREGATEType component and an ActualStructureConstraint based on "`ins`".

8.14.3.1 Supertypes

ParametricElement

8.14.3.2 Attributes

none.

8.14.3.3 Associations

AssociationEnd: source-structure **to: AGGREGATEType**

via: AGGREGATEType-defines-parameter

Definition: the AGGREGATEType from whose instantiations the ParametricStructure takes its values. That is, the ParametricStructure refers to the structure of the attribute-type or the component of the actual parameter that corresponds to this AGGREGATEType.

Note – the AGGREGATEType is unique and is, or is a component of, the data type of the `:source`, which is a ParameterType. Since a ParameterType can contain more than one occurrence of `AGGREGATE`, the intended component of the actual parameter type must be explicitly identified.

Multiplicity: 1..1

8.14.3.4 Other Roles

From: ActualAGGREGATEType as refers-to

From: ActualStructureConstraint as required-structure

8.14.4 Class: ParametricType

Definition: A syntactic occurrence of `GENERIC` or `GENERIC_ENTITY` within a formal parameter type or attribute type that defines a `type_label`. The ParametricType is the first occurrence of the `type_label` among the Parameters of the Algorithm or the Attributes of the EntityType.

In an EntityType declaration, the ParametricType refers to the corresponding InstantiableType component of each corresponding Attribute. In an Algorithm declaration, the ParametricType refers to the data type of the corresponding component of each corresponding ActualParameter. Since the ElementSource (Attribute or Parameter) contains exactly one component that is a GenericType, the ParametricType is always associated with that component.

Note – The association between the ParametricType and the GenericType component is not modeled, since the GenericType component is not itself modeled (it is simply an occurrence of one of the two objects of GenericType). The association is implied, as stated above, by the (inherited) association to the ElementSource (`ParametricElement:source`).

Note – Later occurrences of the `type_label` within the same Scope are ActualTypeConstraints (q.v.).

Note – See ISO 10303-11 clause 9.5.3.4. It also requires that the ParametricType must be based on `GENERIC_ENTITY`, i.e., that `:isEntity` must be TRUE, if the `:source` of the ParametricType is an Attribute.

8.14.4.1 Supertypes

ParametricElement

8.14.4.2 Attributes

Attribute: isEntity **to: MOF::Boolean**

Definition: True if the ParametricType is based on `GENERIC_ENTITY`; False if it is based on `GENERIC`.

Multiplicity: 1..1

8.14.4.3 Associations

none.

8.14.4.4 Other Roles

From: ActualGenericType as refers-to

From: ActualTypeConstraint as required-type

8.14.4.5 Rules

Constraint (OCL):

```
not (self->source.type = Attribute) or (self->isEntity);
```

If the source is an Attribute, the ParametricType must be based on `GENERIC_ENTITY`.

8.14.5 Association: AGGREGATEType-defines-parameter

Definition: represents the relationship between a ParametricStructure and the AGGREGATEType that defines it. The ParametricStructure takes on the structure of the actual parameters that conform to this element of the formal parameter type.

8.14.5.1 Association Ends

AssociationEnd: defines-parameter **To: ParametricStructure**

Definition: The ParametricStructure, if any, that is defined to refer to the structure of the actual data types that conform to this AGGREGATEType.

Multiplicity: 0..1

AssociationEnd: structure-source **To: AGGREGATEType**

Definition: the AGGREGATEType from whose instantiations the ParametricStructure takes its values. That is, the ParametricStructure refers to the structure of the attribute-type or the component of the actual parameter that corresponds to this AGGREGATEType.

Note – the AGGREGATEType is unique and is, or is a component of, the data type of the `:source`, which is a ParameterType. Since a ParameterType can contain more than one occurrence of `AGGREGATE`, the intended component of the actual parameter type must be explicitly identified.

Multiplicity: 1..1

8.14.6 Association: element-has-source

Definition: represents the relationship between a ParametricElement and the syntactic/semantic element (ElementSource) that contains its definition.

Note – See 9.5.3.4 of ISO 10303-11:2004.

8.14.6.1 Association Ends

AssociationEnd: source **To: ElementSource**

Definition: the ElementSource (Attribute or Parameter) whose declared type is or includes the ParametricElement and defines its `type_label`. The first (by `:position`) Attribute or Parameter whose declared type contains the `type_label` is the source for that ParametricElement and defines the `type_label` as its `:id`.

Multiplicity: 1..1

AssociationEnd: type-parameters **To: ParametricElement**

Definition: the ParametricElements, if any, whose declarations are contained in the declared type of the ElementSource (Attribute or Parameter).

Multiplicity: 0..* unordered

8.15 Actual Type Constraints

EXPRESS permits the generalized_type specifications for formal parameters to contain labeled generic components that refer to specific elements of the data type of the corresponding actual parameters. These elements can be referred to in the specifications for the data types of other formal parameters. The effect of such a reference is to state a constraint on the data types of the actual parameters that correspond to the formal parameter that contains the reference to the labeled component.

This section provides a model for the capture of such constraints, herein called *ActualTypeConstraints*. The associated concepts are depicted in Figure 15 (in Section 8.14) and Figure 16.

According to clause 9.5.3.4 of ISO 10303-11, the first occurrence of a labeled component in a parameter type is the defining occurrence and subsequent occurrences are constraining references.

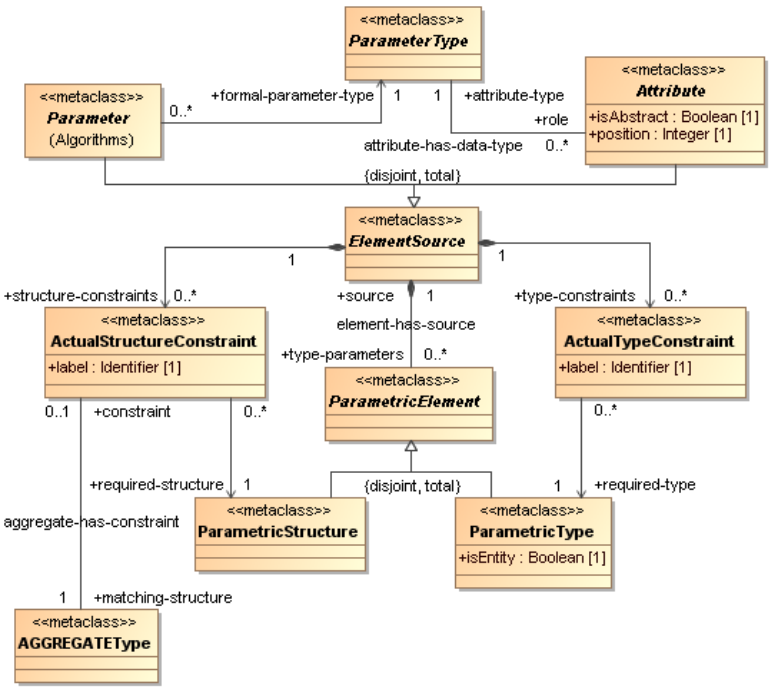

Figure 17 - Actual Type Constraints

8.15.1 Class: ActualStructureConstraint

Definition: A constraint on the structure of the ConcreteAggregationType that corresponds to a given AGGREGATEType. The constraint is declared in EXPRESS by a `type_label` on the `AGGREGATE` keyword that occurs in the specification of an attribute-type or a formal-paramater-type, but is not the definition of that `type_label` (cf. ParametricStructure). The requirement declared by the constraint is that the structure of the ConcreteAggregationType that corresponds to the AGGREGATEType that *uses* the `type_label` (the `:matching-structure`, the component in the data type of the corresponding actual parameter or corresponding instantiable attribute) must be the same as the structure referred to by the ParametricStructure that *defines* the `type_label` (the `:required-structure`).

Note – See 9.5.3.4 of ISO 10303-11:2004.

Example -- In the EXPRESS declaration:

```
FUNCTION check_properties(inputs: AGGREGATE:ins OF property, selectors:
AGGREGATE:ins OF BOOLEAN): BOOLEAN;
```

the `AGGREGATE:ins` in the `inputs` parameter declares both an AGGREGATEType component of the formal-parameter-

type of the inputs Parameter and a ParametricStructure that defines the type_label "ins." The AGGREGATE:ins in the selectors Parameter declares both an AGGREGATEType component of the formal-parameter-type of the selectors Parameter and an ActualStructureConstraint based on "ins." The matching-structure of the ActualStructureConstraint is the AggregateType of the selectors parameter, and the required-structure is the ParametricStructure declared by the inputs parameter.

8.15.1.1 Supertypes

none.

8.15.1.2 Attributes

Attribute: label **To: Core::Identifier**

Definition: the EXPRESS type_label on the AGGREGATE keyword that denotes the constraint. Any occurrence of the same type_label after the first denotes a constraint.

Multiplicity: 1..1

8.15.1.3 Associations

AssociationEnd: matching-structure **To: AGGREGATEType**

via: aggregate-has-constraint

Definition: the AGGREGATEType component to which the constraint applies, i.e., the AGGREGATEType that is denoted by the AGGREGATE keyword that uses the type_label.

Multiplicity: 1..1

AssociationEnd: required-structure **To: ParametricStructure**

Definition: the ParametricStructure that defines the EXPRESS type_label that is used to establish the constraint. The :required-structure defines the required structure (ARRAY, BAG, LIST, SET) of the ConcreteAggregationType that corresponds to the AGGREGATEType that is the :matching structure.

Multiplicity: 1..1

8.15.1.4 Other Roles

From: ElementSource as structure-constraints

8.15.2 Class: ActualTypeConstraint

Definition: a constraint on the InstantiableType that corresponds to a given GenericType component of an attribute-type or a formal-parameter-type. The constraint is declared in EXPRESS by a type_label (the :label property) on a GENERIC or GENERIC_ENTITY keyword that occurs in the specification of the formal-parameter-type, but is not defined there. The requirement declared by the constraint is that the InstantiableType that corresponds to the GenericType component that *uses* the type_label (the component in the data type of the corresponding actual parameter or corresponding instantiable attribute) must be type compatible with the InstantiableType to that corresponds to the ParametricType that *defines* the type_label (the :required-type).

If the formal parameter types of additional Parameters of the same Algorithm contain the same `type_label`, each such occurrence constitutes a distinct ActualTypeConstraint.

The data type of the ElementSource (Attribute or Parameter) that has the ActualTypeConstraint contains exactly one occurrence of a GenericType (`GENERIC` or `GENERIC_ENTITY`). That occurrence is the component that is constrained by the ActualTypeConstraint.

Note – See 9.5.3.4 of ISO 10303-11:2004.

8.15.2.1 Supertypes

none.

8.15.2.2 Attributes

Attribute: label **To: Identifier**

Definition: the EXPRESS `type_label` on the `GENERIC` or `GENERIC_ENTITY` keyword that denotes the constraint. Any occurrence of the same `type_label` after the first denotes a constraint.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 1..1

8.15.2.3 Associations

AssociationEnd: required-type **To: ParametricType**

Definition: the ParametricType that defines the EXPRESS type_label that is used to establish the constraint. The ParametricType defines the data type with which the component of the data type of the actual parameter that has the ActualTypeConstraint must be compatible.

Multiplicity: 1..1

8.15.2.4 Other Roles

From: Parameter as type-constraints

8.15.3 Association: aggregate-has-constraint

Definition: the relationship between an AGGREGATE type specification and its ActualStructureConstraint, if any.

8.15.3.1 Association Ends

AssociationEnd: constraint **To: ActualStructureConstraint**

Definition: the ActualStructureConstraint, if any, that applies to this component of the GeneralizedType specification.

Note – Only an AGGREGATEType that appears in the specification of the data type of a Parameter or an Attribute of an abstract entity data type can have an ActualStructureConstraint. The AGGREGATEType has an ActualStructureConstraint only if it has a syntactic `type_label` and does not itself define that type_label.

Note – See 9.5.3.4 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: matching-structure **To: Core::AGGREGATEType**

Definition: the AGGREGATEType component to which the constraint applies, i.e., the AGGREGATEType that is denoted by the `AGGREGATE` keyword that uses the `type_label`.

Multiplicity: 1..1

8.16 Expressions and Instances

This section of the Core model introduces the basic concepts for Expression and Instance, which are expanded in other packages. They are provided here so that implementations need not support the Expressions and Instances Packages in order to support all features of the Core model.

For Expressions, the Core package contains only the class Expression. The optional `:text` attribute allows an Expression to be represented as verbatim EXPRESS language text. The Expressions package (see clause 12) models the subclasses of Expression that represent the semantic interpretation of the parsed language text. Support for the Expressions Package is a compliance point (see Section 4.4.4).

The class Instance is abstract. The Core package contains the Instance concept solely in order to model the semantics of Expressions. The Instances Package (see clause 9) models the detailed expansion of the Instance concept, including all of the instantiable subclasses. Support for the Instances package is a compliance point (see Section 4.4.1). Implementations that do not support the Instances package do not, in general, need to provide any implementation of the Instances class, and may provide any simple implementation where needed.

Figure 18 shows the base Expression and Instance concepts, and they are described below.

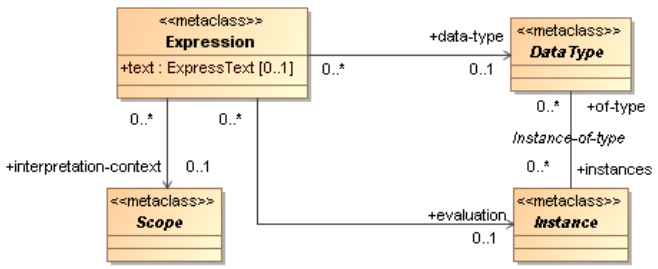

Figure 18 - Basic Expression Concepts

8.16.1 Class: Expression

Definition: In general, an Expression is the representation of an Instance by a set of computational operations that will produce that Instance when performed in the context in which the Expression occurs. An Expression is always evaluated in a context

which determines the Instances denoted by the model elements (e.g., Variables, Attributes, etc.) that appear in the Expression. This context is explicit in the model element that contains the Expression being evaluated, but it implicitly includes the Population under study. The Instance produced by the same Expression may vary from context to context. The Instance produced is said to be the *value*, or the *evaluation*, of the Expression.

Note – In general, Expressions are treated as reusable. It is recommended, however, that, except for literals and local variables, each occurrence should be a unique object. A few uses of Expression are not treated in the model as reusable, specifically those that are the definitions of Rules.

8.16.1.1 Supertypes

none.

8.16.1.2 Attributes

Attribute: text **To: ExpressText**

Definition: represents the actual EXPRESS language text denoting the Expression. The text is required if the Expressions Package is not implemented. It is optional in most cases when the Expressions Package is implemented. Certain forms of Expression (in the Expressions Package) specialize the text attribute.

Multiplicity: 0..1

8.16.1.3 Associations

AssociationEnd: data-type **To: DataType**

Definition: represents the DataType of the evaluation of the Expression. While the result of an Expression always has a DataType, it is not always possible to determine at model-analysis time what that data type is. And in many cases, even when it is known, it is not necessary to specify it.

Multiplicity: 0..1

AssociationEnd: evaluation **To: Instance**

Definition: represents the Instance (value) that results from evaluating the Expression. Since the same Expression can be evaluated in more than one “situation,” i.e., different values for the operands, the result in each situation may be a different Instance. The evaluation is included in a model, however, only when it is “constant” and can be computed at “compile time.”

Multiplicity: 0..1

AssociationEnd: interpretation-context **To: Scope**

Definition: An Expression is always evaluated in a context which determines the assignment of specific instances of model elements to symbols (e.g.,Variables, Attributes, etc.). When the Expression is represented by text only, this relationship is usually required, but in many cases it may be implicit. When the Expression is represented by the detailed model elements in the Expressions Package, the interpretation of the Text has been done, and this association is purely documentary and not required. Certain permissible EXPRESS constructs, however, only permit interpretation of certain keyword symbols to Operations in the presence of actual operand Instances.

Multiplicity: 0..1

8.16.1.4 Other Roles

From: ArrayBound as bound-expression

From: DerivedAttribute as derivation

From: DomainConstraint as asserts

From: Redeclaration as derivation

From: Algorithms::LocalVariable as initial-value

From: Expressions::AggregateIndex as index-value

From: Expressions::AttributeBinding as attribute-value

From: Expressions::BinaryIndex as first-bit

From: Expressions::BinaryIndex as last-bit

From: Expressions::BinaryOperation as right-operand

From: Expressions::BinaryOperation as left-operand

From: Expressions::Coercion as operand

From: Expressions::IndexOperation as base-value

From: Expressions::MemberBinding as member-value

From: Expressions::PassByValue as actual-value

From: Expressions::QueryExpression as aggregate-operand

From: Expressions::QueryExpression as select-condition

From: Expressions::RepeatCount as derivation

From: Expressions::Selector as entity-instance

From: Expressions::StringIndex as first-code

From: Expressions::StringIndex as last-code

From: Expressions::UnaryOperation as unary-operand

From: Instances::Constant as value-expression

From: Rules::NamedRule as asserts-expression

From: Rules::SubtypeConstraint as equivalent-rule

From: Statements::Assignment as assigned-value

From: Statements::CaseAction as label-value

From: Statements::CaseStatement as selection-expression

From: Statements::ControlVariable as bound-value

From: Statements::ControlVariable as increment

From: Statements::ControlVariable as initial-value

From: Statements::IfStatement as if-condition

From: Statements::MemberCell as index-value

From: Statements::RepeatStatement as while-expression

From: Statements::RepeatStatement as until-expression

From: Statements::ReturnStatement as return-value

8.16.1.5 Rules

Constraint ()

An Expression can only exist to fulfill a role.

8.16.2 Class: Instance

Definition: an object that is in the domain of a DataType - any real or conceptual object, information unit or data element.

Properties: abstract

8.16.2.1 Supertypes

none.

8.16.2.2 Attributes

none.

8.16.2.3 Associations

AssociationEnd: appears-in-population **To: Instances::Population**

via: Instances::instance-appears-in-population

Definition: represents the relationship between an Instance and the Populations in which it appears.

Multiplicity: 0..* unordered

AssociationEnd: of-type **To: DataType**

Definition: the DataType(s) that are instantiated in the Instance. With the exception of the Indeterminate Instance (see Section 10.2.7), every modeled Instance instantiates at least one modeled DataType; an Instance may instantiate more than one.

A modeled Instance should be modeled as an Instance of its “declared type.” It may, but need not, be modeled as an Instance of all the supertypes or SelectTypes that it instantiates.

Multiplicity: 0..*

8.16.2.4 Other Roles

From: Expression as evaluation

From: Instances::ArrayMember as member-value

From: Instances::ListMember as member-value

From: Instances::BagMember as member-value

From: Instances::SETValue as member-value

From: Instances::AttributeValue as actual-value

From: Instances::Constant as actual-value

8.16.2.5 Rules

Constraint (OCL): datatype-required

```
self = INDETERMINATE or exists(self->of-type);;
```

Every Instance except INDETERMINATE has at least one data type.

8.17 Instance Package: BuiltInTypes

This Package is a part of the Core Package. It contains required instances of subclasses of SimpleType. All of the other instances of SimpleType appear in a Schema as a SimpleType with a constraint or a precision.

Note – The purpose of making this a Package is to separate the class model from the "ground facts."

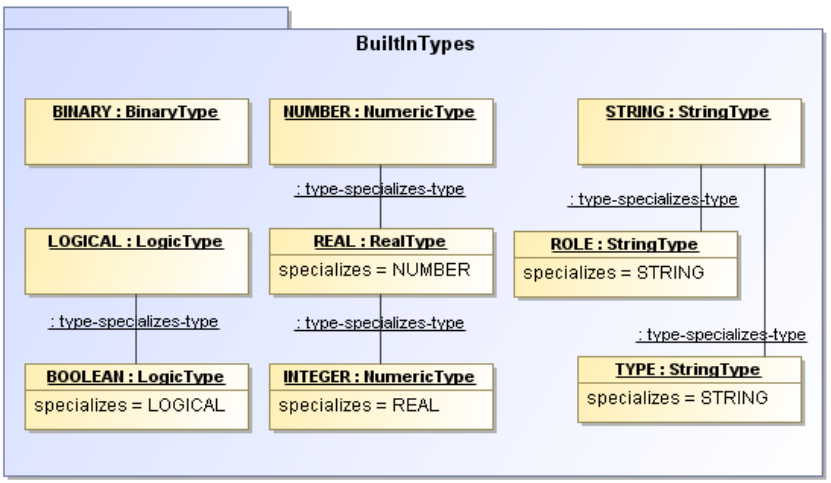

Figure 19 - Instance Model for Built-In Types

8.17.1 Dependencies

Dependency on Class: Core::SimpleType

Stereotypes: `instantiates`

This Package provides base individuals that are always instances of SimpleType (that is, instances of its subtypes).

8.17.2 Instance: BINARY

Type: Core::BinaryType

Definition: represents the EXPRESS type BINARY without length constraints.

Note – The class BinaryType also includes instances of EXPRESS BINARY that have declared length constraints.

8.17.3 Instance: BOOLEAN

Type: Core::LogicType

Definition: represents the EXPRESS type BOOLEAN

Note – BOOLEAN and LOGICAL are the only instances of LogicType.

8.17.4 Instance: INTEGER

Type: Core::NumericType

Definition: represents the EXPRESS type INTEGER

Note – INTEGER and NUMBER are the only instances of NumericType that are not RealTypes.

8.17.5 Instance: LOGICAL

Type: Core::LogicType

Definition: represents the EXPRESS type LOGICAL

Note – BOOLEAN and LOGICAL are the only instances of LogicType.

8.17.6 Instance: NUMBER

Type: Core::NumericType

Definition: represents the EXPRESS type NUMBER

Note – INTEGER and NUMBER are the only instances of NumericType that are not RealTypes.

8.17.7 Instance: REAL

Type: Core::RealType

Definition: represents the EXPRESS type REAL without a precision specification.

Note – The class RealType also includes instances of EXPRESS REAL that have precision specifications.

8.17.8 Instance: ROLE

Type: Core::StringType

Definition: ROLE is the StringType whose instances are the names of Attributes, i.e. the result of RolesOf and the formal second operand of UsedIn. These objects are data typed STRING in Part 11, but they have reserved syntax and reserved interpretation. In order to facilitate mappings to other languages, these data types are explicitly identified, and coerced to/from STRING where necessary.

Note – See Clause 15.20 of ISO 10303-11:2004.

8.17.9 Instance: STRING

Type: Core::StringType

Definition: represents the EXPRESS type STRING without constraints

Note – The class StringType also includes TYPE, ROLE and instances of EXPRESS STRING that have declared length constraints.

8.17.10 Instance: TYPE

Type: Core::StringType

Definition: TYPE is the StringType whose instances are the names of DataTypes (TypeNames), i.e., the result of TypeOf and related operands. These objects are data typed STRING in Part 11, but they have reserved syntax and reserved interpretation. In order to facilitate mappings to other languages, these data types are explicitly identified, and coerced to/from STRING where necessary.

Note – See Clause 15.25 of ISO 10303-11:2004.

8.17.11 Association: instance-of-type

Definition: represents the abstract relationship between an Instance (a value) and the DataTypes that it instantiates.

8.17.11.1 Association Ends

AssociationEnd: instances **To: Instance**

Definition: the modeled Instances of the DataType, if any. In general, Instances of a DataType are not modeled unless they appear directly in a Schema.

Note – For most DataTypes, navigating the association in this direction is not a required feature of the model.

Multiplicity: 0..* unordered.

AssociationEnd: of-type **To: DataType**

Definition: the DataType(s) that are instantiated in the Instance. Every modeled Instance instantiates at least one modeled DataType; an Instance may instantiate more than one.

Multiplicity: 1..* unordered.

8.18 Instance Package: GenericTypes

This Package is a part of the Core Package. It contains the required instances of the class GenericType. There are no other instances of the class GenericType.

Note – The purpose of making this a Package is to separate the class model from the "ground facts."

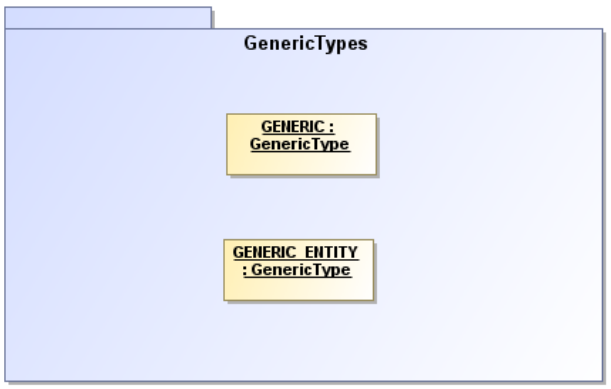

Figure 20 - Instance Model for Generic Types

8.18.1 Dependencies

Dependency on Class: Core::GenericType

Stereotypes: `instantiates`

This Package provides base individuals that are always the only instances of class GenericType.

8.18.2 Instance: GENERIC

Type: Core::GenericType

Definition: represents the EXPRESS generalized type GENERIC. Every data type is a specialization of the GenericType GENERIC, and every Instance is an Instance of GENERIC.

Note – See 9.5.3.2 of ISO 10303-11:2004.

8.18.3 Instance: GENERIC_ENTITY

Type: Core::GenericType

Definition: represents the EXPRESS generalized type GENERIC_ENTITY. Every entity data type is a specialization of GENERIC_ENTITY. Every EntityInstance is an instance of GENERIC_ENTITY and every instance of GENERIC_ENTITY is an EntityInstance.

Note – See 9.5.3.3 of ISO 10303-11:2004.

9 Enumerations

The Enumerations Package contains the EnumerationItem concept and its relationships to EnumerationType in the Core Package. It also contains the abstract subclass ConcreteType, which is a subclass of Instance and a supertype of EnumerationItem.

The purpose of the Enumerations Package is to support a compliance point consisting of the Core Package (schema declarations) and EnumerationItems, without requiring support for the full Instances Package.

9.1 Dependencies

Dependency on Package: Core

The Enumerations Package depends on the Core Package for the EnumerationType and Instance concepts.

9.2 Enumeration Items

Figure 21 shows the overall model of EnumerationItems, i.e., Instances of EXPRESS ENUMERATION data types. EnumerationItems are ConcreteValues.

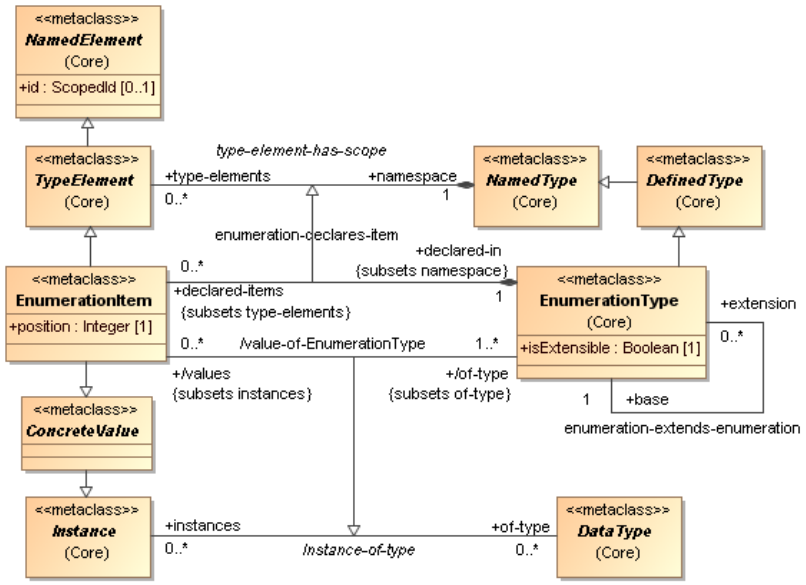

Figure 21 - Enumeration Items

9.2.1 Class: ConcreteValue

Definition: represents a data item, an Instance that is an item of information that has an explicit data representation conveying its meaning.

Properties: abstract

Note – ConcreteValue is included in this Package in order to enable a consistent “package merge” of the Enumerations Package into the Instances Package. Its real usage is described in the Instances Package.

9.2.1.1 Supertypes

Core::Instance

9.2.1.2 Attributes

none.

9.2.1.3 Associations

none.

9.2.1.4 Other Roles

none.

9.2.2 Class: EnumerationItem

Definition: a ConcreteValue representing a named value of an EnumerationType. An EnumerationItem is also a TypedInstance, because the corresponding EnumerationType has an Identifier. An EnumerationItem is also a TypeElement, in that the scope of its identifier is the EnumerationType.

Note – See 8.4.1 of ISO 10303-11:2004.

9.2.2.1 Supertypes

ConcreteValue, Core::TypeElement

9.2.2.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: Represents the position of the Enumeration Item in the list of items in the type_declaration that defines the EnumerationItem. That is, `:position` relates to the `:declared-in` EnumerationType. When the number of values of `:of-type` (the types of which this EnumerationItem is a value) is exactly 1, the position defines an ordering on the values of the EnumerationType.

Multiplicity: 1..1

9.2.2.3 Associations

AssociationEnd: declared-in **To: Core::EnumerationType**

via: enumeration-declares-items

Subsets: Core::TypeElement.namespace

Definition: represents the relationship between an EnumerationItem and the EnumerationType whose declaration defines the item.

Multiplicity: 1..1

AssociationEnd: of-type **To: Core::EnumerationType**

via: value-of-EnumerationType

Definition: represents the relationship between an EnumerationItem and the EnumerationTypes of which it is a value.

With respect to a given "governing schema" and all of the SchemaElements it defines and interfaces, each declared EnumerationItem is a value of every EnumerationType that is related by extension to the EnumerationType in which it is declared. That is, it is a value of

(a) the EnumerationType `self->declared-in`;

(b) the EnumerationType that is the `:base` of that EnumerationType, if any, and recursively of all EnumerationTypes related by `:base`, and

(c) each EnumerationType that is an .extension of any of the EnumerationTypes related by either (a) or (b) above, and recursively of all EnumerationTypes related to them by `:extension`.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

Properties: derived.

9.2.2.4 Other Roles

From Expressions::EnumItemRef as refers-to

9.2.2.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every EnumerationItem shall have an Identifier.

9.2.3 Association: enumeration-declares-items

Definition: represents the relationship between an EnumerationItem and the EnumerationType whose declaration defines the item.

This can be different from value-of-EnumerationType (see below) only when the EnumerationType is EXTENSIBLE, or is itself the extension of another EnumerationType.

9.2.3.1 Supertypes

Core::type-element-has-scope

9.2.3.2 Association Ends

AssociationEnd: declared-in **To: Core::EnumerationType**

Definition: represents the relationship between an EnumerationItem and the EnumerationType whose declaration defines the item.

Multiplicity: 1..1

AssociationEnd: declared-items **To: EnumerationItem**

Definition: represents the relationship of an EnumerationType to the EnumerationItems that are declared in its type_declaration. For extended enumeration types, this is distinct from the `:values` relationship, which captures all of the valid values of the type.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

9.2.4 Association: value-of-EnumerationType

Definition: represents the relationship between an EnumerationType and the EnumerationItems that are valid values of the type.

Properties: derived

9.2.4.1 Association Ends

AssociationEnd: of-type **To: Core::EnumerationType**

Definition: represents the relationship between an EnumerationItem and the EnumerationTypes of which it is a value.

With respect to a given “governing schema” and all of the SchemaElements it defines and interfaces, each declared EnumerationItem is a value of every EnumerationType that is related by extension to the EnumerationType in which it is declared. That is, it is a value of

(a) the EnumerationType `self->declared-in`;

(b) the EnumerationType that is the `:base` of that EnumerationType, if any, and recursively of all EnumerationTypes related by `:base`, and

(c) each EnumerationType that is an .extension of any of the EnumerationTypes related by either (a) or (b) above, and recursively of all EnumerationTypes related to them by `:extension`.

Note – See 8.4.1 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

Properties: derived.

AssociationEnd: values **To: EnumerationItem**

Definition: represents the relationship between an EnumerationType and the EnumerationItems that are valid values of the type.

An EnumerationItem is a value of every EnumerationType that is related by extension to the type that declares it. This relationship can be derived recursively as the union of the values of the `:declared-items` attribute for the EnumerationType, for each EnumerationType in the sequence of `:base` relationships from the EnumerationType, and from all the extensions of the EnumerationType.

Note – See clause 8.4.1 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: derived.

Note – The derivation of the values of `:values` is a recursive operation, described in the text above.

10 Package : Instances

The Instances Package contains all of the Instance concepts that go with the Type concepts in the Core Package. The Instances Package incorporates the model elements in the Enumerations Package, in order to do this.

The purpose of the Instances Package is to provide a model representation for specific Instances that are explicitly referred to in a Schema. A tool that supports the Expressions Package may also use Instances to represent the values of expressions that can be statically evaluated.

Note – It is possible to represent an actual Population as an instance of this package, but such a representation is “unexpected.” In MOF terminology, the EXPRESS metamodel defined in this specification is an M2 model. An EXPRESS Schema and its contents constitute an M1 population that conforms to this metamodel. A Population (in the EXPRESS sense) should be represented as an M0 population that conforms to the M1 model of the governing-schema. Representing that Population as an instance of this package would make it an M1 population that carries direct M1 links to the M1 objects representing the model elements of the EXPRESS Schema. While such a representation is (accidentally) enabled by this Package, that is not the purpose of this package, and it is not to be considered a required part of any compliance point.

10.1 Dependencies

Dependency on Package: Core

Stereotypes: `import`

The Instances Package depends on the Core Package for the InstantiableType concepts that are the data types of the individuals (Instances).

Dependency on Package: Enumerations

Stereotypes: `merge`

The Instance Package extends the ConcreteType and EnumerationItem model elements from the EnumerationsPackage, while importing the properties and associations modeled therein.

10.2 Overview of Instances

Figure 22 shows the overall model of Instances of EXPRESS data types. Instances are divided into TypedInstances, ConcreteValues, and two special categories – PartialEntityValue and Indeterminate.

TypedInstances represent instances of NamedTypes. There are three subcategories – EntityInstances, SpecializedValues and EnumerationItems. TypedInstances are the instances that can be values of SelectTypes. Each of the subcategories corresponds to one of the other subtypes of NamedType.

ConcreteValues are Instances that can be the fundamental values of SpecializedValues – the values of SpecializationTypes. There are three subcategories – SimpleValues, AggregateValues and EnumerationItems. SimpleValues correspond to the SimpleTypes. AggregateValues correspond to the ConcreteAggregationTypes. EnumerationItems correspond to EnumerationTypes, and because EnumerationTypes are NamedTypes, EnumerationItems are also TypedInstances.

Indeterminate is the class that corresponds to the EXPRESS constant "?", which can be considered to be an instance of all EXPRESS data types, or of none of them.

PartialEntityValues only arise as the results of Expressions. They are described in detail in 10.5, which deals with values of EntityTypes.

This section defines the Instance concepts associated with EXPRESS defined data types — Select types, Enumeration types, and Specializations — in detail. SimpleValues, AggregateValues, values of EntityTypes are described in subsequent sections.

While the domains of EXPRESS data types are often unbounded, only those Instances that actually occur in, or as a result of an Expression in, a Schema need to be materialized in a metamodel population that represents the Schema. Similarly, in a Population that is realized as an instance of this package, only the Instances actually occurring in that Population need to be represented.

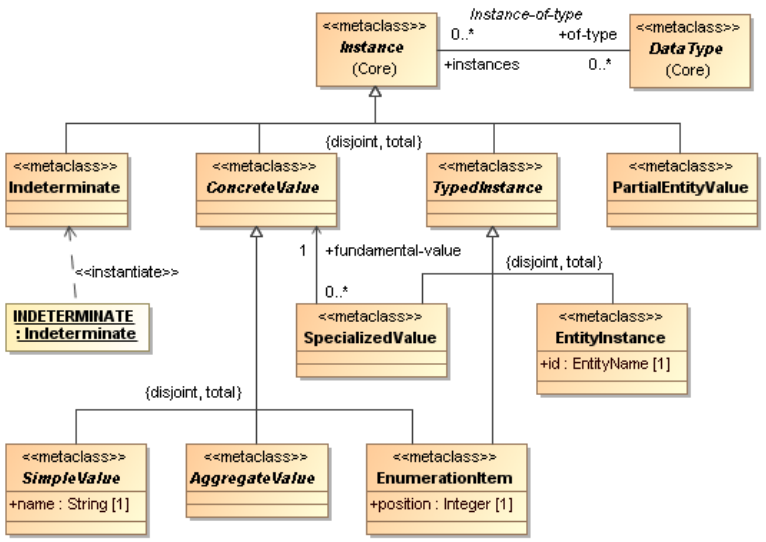

Figure 22 - Overview of Instances

10.2.1 Class Core::Instance

Definition: an object that is in the domain of a DataType - any real or conceptual object, information unit or data element.

Properties: abstract

Note – The Instance concept, and all its properties, is defined in the Core Package, so that it may be referenced in other Packages without creating interdependencies. There is no real requirement for support of Instances in the Core Package. This entry serves to define the Instance concept in the context of the Instances Package, and to provide a link to the complete specification in 8.17.2.

10.2.2 Class: ConcreteValue

Definition: represents a data item, an Instance that is an item of information that has an explicit data representation conveying its meaning.

Properties: abstract

Note – ConcreteValue is defined in the Enumerations Package. The Instances Package extends it, but only by adding a (passive) role: It represents the instances that can be fundamental-values of SpecializedValues.

10.2.2.1 Supertypes

Core::Instance

10.2.2.2 Attributes

none.

10.2.2.3 Associations

none.

10.2.2.4 Other Roles

From: SpecializedValue as fundamental-value

10.2.3 Class: EnumerationItem

Definition: a ConcreteValue representing a named value of an EnumerationType. An EnumerationItem is also a TypedInstance, because the corresponding EnumerationType has an Identifier. An EnumerationItem is also a TypeElement, in that the scope of its identifier is the EnumerationType.

Note – EnumerationItem is defined in the EnumerationsPackage. The Instances Package extends EnumerationItem by making it a subclass of TypedInstance, and inheriting those properties. There are no other changes or additions.

10.2.3.1 Supertypes

ConcreteValue, TypedInstance, Core::TypeElement

10.2.3.2 Attributes

no additions or changes

10.2.3.3 AssociationEnds

no additions or changes

10.2.3.4 Other Roles

no additions or changes

10.2.3.5 Rules

no additions or changes

10.2.4 Class: Indeterminate

Definition: The class that contains only the INDETERMINATE object (see Section 10.2.7).

Note – See 14.2 of ISO 10303-11:2004.

10.2.4.1 Supertypes

Core::Instance

10.2.4.2 Attributes

none.

10.2.4.3 Associations

none.

10.2.4.4 Other Roles

From Expressions::IndeterminateRef as refers-to

10.2.4.5 Rules

Constraint (OCL): is-singleton

```
self = INDETERMINATE;
```

The only instance of Indeterminate is the INDETERMINATE object.

10.2.5 Class: SpecializedValue

Definition: a TypedInstance that is a value of a SpecializedType. Every SpecializedValue is represented by some ConcreteValue, called its *fundamental-value*.

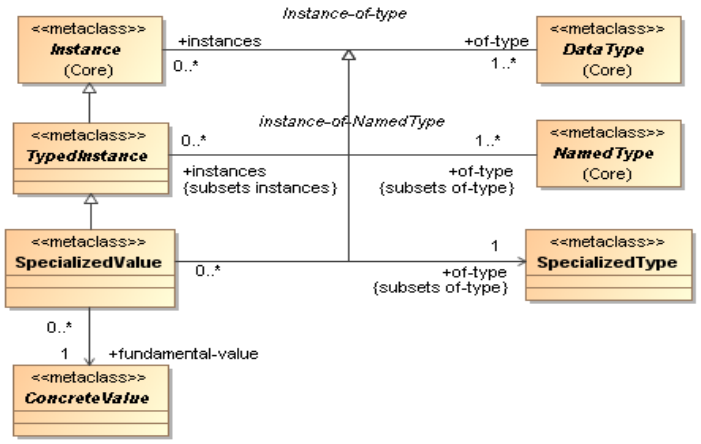

Figure 23 - Specialized Values

10.2.5.1 Supertypes

TypedInstance

10.2.5.2 Attributes

none.

10.2.5.3 Associations

AssociationEnd: fundamental-value **To: ConcreteValue**

Definition: represents the relationship between a SpecializedInstance and the “fundamental” ConcreteValue that is used to represent that Instance.

Multiplicity: 1..1

AssociationEnd: of-type **To: Core::SpecializedType**

subsets: Core::Instance:of-type

Definition: represents the relationship between a SpecializedValue and its data type.

Multiplicity: 1..1

10.2.5.4 Other Roles

none.

10.2.6 Class: TypedInstance

Definition: an abstract classifier, a subtype of Instance comprising those Instances that are instances of a NamedType. Only a TypedInstance can instantiate a SelectType.

Properties: abstract

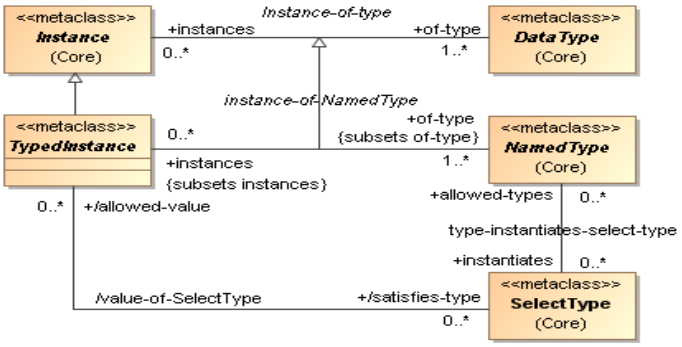

Figure 24 - TypedInstances

10.2.6.1 Supertypes

Core::Instance

10.2.6.2 Attributes

none.

10.2.6.3 Associations

AssociationEnd: satisfies-type **To: Core::SelectType**

via: value-satisfies-SelectType

subsets: Core::Instance:of-type

Definition: represents the relationship between a TypedInstance and the SelectTypes of which it is an allowable instance.

Multiplicity: 0..* unordered

10.2.6.4 Other Roles

none.

10.2.7 Instance: INDETERMINATE

Type: Instances::Indeterminate

Definition: Represents the EXPRESS “indeterminate” value, which is interpreted as “no determinable value.”

This Instance arises as the evaluation of an Expression that is the Indeterminate literal ("?"), or an Expression in which one of the operations “fails.” Indeterminate is not an instance of any data type, but it may be treated as an instance of the required data type of the Expression, if any.

Note – See clause 14.2 of ISO 10303-11:2004.

10.3 Simple Values

This section specifies the model of SimpleValues – Instances that correspond to the simple data types defined in the EXPRESS language: BINARY, BOOLEAN, LOGICAL, INTEGER, NUMBER, REAL, STRING. The model is shown in Figure 25.

It also includes two specialized classes of STRING value that have specific syntax requirements in the EXPRESS language: TypeName and RoleName. There are no EXPRESS data types for these, but certain values in Expressions are required to be instances of these classes.

There are exactly three distinct LogicalValues – FALSE, TRUE, and UNKNOWN. These are explicitly modeled as individual objects in the NamedValues package.

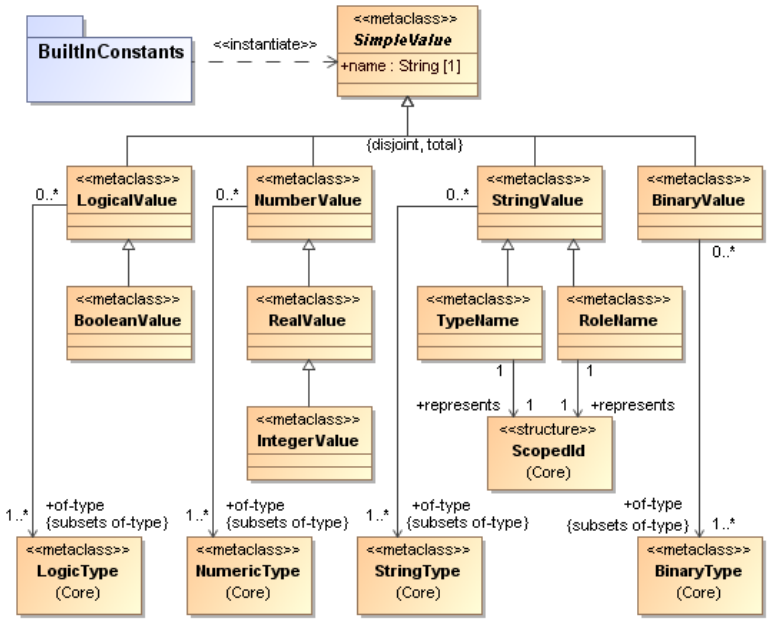

Figure 25 - Simple Values

10.3.1 Class: BinaryValue

Definition: an AggregateValue, representing a value of an EXPRESS BAG data type: a collection of instances of the member-type of the BAG, in which a given instance can appear more than once.

10.3.1.1 Supertypes

SimpleValue

10.3.1.2 Attributes

none.

10.3.1.3 Associations

AssociationEnd: of-type **To: Core::BinaryType**

subsets: Core::Instance:of-type

Definition: the BinaryType(s) that are instantiated in the BinaryValue.

Multiplicity: 1..* unordered.

10.3.1.4 Other Roles

none.

10.3.2 Class: BooleanValue

Definition: a SimpleValue, a value of the EXPRESS data type BOOLEAN: TRUE, FALSE

10.3.2.1 Supertypes

LogicalValue

10.3.2.2 Attributes

none.

10.3.2.3 Associations

none.

10.3.2.4 Other Roles

none.

10.3.2.5 Rules

Constraint

```
(self == NamedValues::TRUE) or (self == NamedValues::FALSE);
Every BooleanValue must be either TRUE or FALSE.
```

10.3.3 Class: IntegerValue

Definition: a SimpleValue, a value of the EXPRESS data type INTEGER: any mathematical integer value.

10.3.3.1 Supertypes

RealValue

10.3.3.2 Attributes

none.

10.3.3.3 Associations

none.

10.3.3.4 Other Roles

none.

10.3.4 Class: LogicalValue

Definition: a SimpleValue, a value of the EXPRESS data type LOGICAL: TRUE, UNKNOWN, FALSE.

10.3.4.1 Supertypes

SimpleValue

10.3.4.2 Attributes

none.

10.3.4.3 Associations

AssociationEnd: of-type **To: Core::LogicType**

subsets: Core::Instance:of-type

Definition: the LogicType(s) that are instantiated in the LogicalValue.

Note – The of-type relationships of the LogicalValues are explicitly modeled in the NamedValues Package.

Multiplicity: 1..* unordered.

10.3.4.4 Other Roles

none.

10.3.4.5 Rules

Constraint

```
(self == NamedValues::TRUE) or (self == NamedValues::FALSE)
      or (self == NamedValues::UNKNOWN);
```

Every LogicalValue must be one of: TRUE or FALSE or UNKNOWN.

10.3.5 Class: NumberValue

Definition: a SimpleValue, a value of the EXPRESS data type NUMBER: any numeric value with its mathematical interpretation.

10.3.5.1 Supertypes

SimpleValue

10.3.5.2 Attributes

none.

10.3.5.3 Associations

AssociationEnd: of-type **To: Core::NumericType**

subsets: Core::Instance:of-type

Definition: the NumericType(s) that are instantiated in the NumberValue.

Multiplicity: 1..* unordered.

10.3.5.4 Other Roles

none.

10.3.6 Class: RealValue

Definition: a SimpleValue, a value of the EXPRESS data type REAL: supposedly a mathematical “real” value, but properly a computational fixed or floating-point value.

10.3.6.1 Supertypes

NumberValue

10.3.6.2 Attributes

none.

10.3.6.3 Associations

none.

10.3.6.4 Other Roles

none.

10.3.7 Class: RoleName

Definition: A RoleName is a reference to an Attribute that has the form of a StringValue. It is an instance of StringType ROLE. RoleNames are produced as the result-type of the UnaryOperator RolesOf, and used as the formal parameter type for UsedIn. They have reserved syntax and reserved interpretation.

Note – The result of RolesOf is only well-defined for Attributes of EntityTypes defined in the Schema. Some problems arise with interfaced EntityTypes, renamed Attributes, and attributes of EntityTypes defined in AlgorithmScopes.

Note – See Clause 15.25 of ISO 10303-11:2004.

10.3.7.1 Supertypes

StringValue

10.3.7.2 Attributes

Attribute: represents **To: Core::ScopedId**

Definition: represents the relationship between the RoleName – a StringValue – and the (structured) TypeScopedId for the Attribute, of which it is a representation.

Multiplicity: 1..1

10.3.7.3 Associations

AssociationEnd: refers-to **To: Core::Attribute**

Definition: represents the relationship between a RoleName and the Attribute to which it refers.

Multiplicity: 1..1

10.3.7.4 Other Roles

none.

10.3.8 Class: SimpleValue

Definition: a ConcreteValue that consists of a single atomic information unit of a data type defined in the EXPRESS language itself.

Properties: abstract

10.3.8.1 Supertypes

ConcreteValue

10.3.8.2 Attributes

Attribute: name **To: MOF::String**

Definition: the representation of the value, assumed to be a character string.

Multiplicity: 1..1

10.3.8.3 Associations

none

10.3.8.4 Other Roles

From Expressions::Literal as refers-to

10.3.9 Class: StringValue

Definition: a SimpleValue, a value of the EXPRESS data type STRING: a sequence of character codes from the ISO 10646-1 Basic Multilanguage Plane.

10.3.9.1 Supertypes

SimpleValue

10.3.9.2 Attributes

none.

10.3.9.3 Associations

AssociationEnd: of-type **To: Core::StringType**

subsets: Core::Instance:of-type

Definition: the StringType(s) that are instantiated in the StringValue.

Multiplicity: 1..* unordered.

10.3.9.4 Other Roles

none.

10.3.10 Class: TypeName

Definition: A TypeName is a reference to a DataType that has the form of a StringValue. It is an instance of StringType TYPE. TypeNames are produced as the result-type of the UnaryOperator TypeOf. They have reserved syntax and reserved interpretation.

Note – The result of TypeOf is only well-defined for NamedTypes defined in the Schema, although it can also produce EXPRESS keywords. Some problems arise with interfaced NamedTypes, and NamedTypes defined in AlgorithmScopes.

Note – See Clause 15.25 of ISO 10303-11:2004.

10.3.10.1 Supertypes

StringValue

10.3.10.2 Attributes

Attribute: represents **To: Core::ScopedId**

Definition: the (structured) ScopedId for the NamedType, of which the TypeName is a String representation.

Multiplicity: 1..1

10.3.10.3 Associations

AssociationEnd: refers-to **To: Core::NamedType**

Definition: represents the relationship between a TypeName and the NamedType to which it refers.

Multiplicity: 1..1

10.3.10.4 Other Roles

none.

10.4 Aggregate Values

This section specifies the model of AggregateValues – Instances that correspond to EXPRESS aggregation types: ARRAY, BAG, LIST, SET.

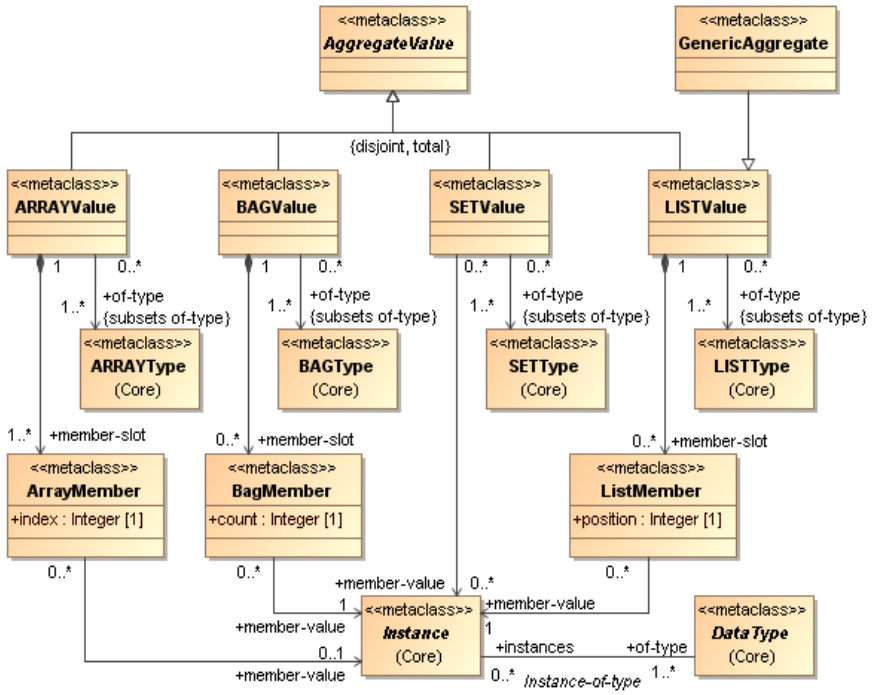

Figure 26 - Aggregate Values

10.4.1 Class: AggregateValue

Definition: a ConcreteValue that is composite, consisting of a collection of Instances from a given member DataType.

Properties: abstract

10.4.1.1 Supertypes

ConcreteValue

10.4.1.2 Attributes

none.

10.4.1.3 Associations

none.

10.4.1.4 Other Roles

none.

10.4.2 Class: ArrayMember

Definition: Represents a single element of an ARRAYValue seen as a relation. It maps one index-value to one value of the base data type (the “member” value). In the case of an ARRAY OF OPTIONAL, the member-value need not be present.

10.4.2.1 Supertypes

none.

10.4.2.2 Attributes

Attribute: index **To: MOF::Integer**

Definition: represents the index value to which the ArrayMember corresponds. In a given ARRAYValue, there is exactly one ArrayMember that corresponds to each index value.

Multiplicity: 1..1

10.4.2.3 Associations

AssociationEnd: member-value **To: Core::Instance**

Definition: for a given ARRAYValue, represents the relationship between an index value (represented by an ArrayMember) and the Instance value that is the image of that index value in the base type.

Multiplicity: 0..1

10.4.2.4 Other Roles

From: ARRAYValue as member-slot

10.4.3 Class: ARRAYValue

Definition: an AggregateValue, representing a value of an EXPRESS ARRAY data type: a set of pairs of the form (index value, domain value) where the index value is selected from a finite range of integers, and each such value occurs in exactly one pair, and the domain value is an instance of the member-type of the ARRAY.

10.4.3.1 Supertypes

AggregateValue

10.4.3.2 Attributes

none.

10.4.3.3 Associations

AssociationEnd: member-slot **To: ArrayMember**

Definition: represents the relationship between an ArrayValue and each of its distinct slots for member values.

Multiplicity: 1..* unordered

Properties: composite

AssociationEnd: of-type **To: Core::ARRAYType**

subsets: Core::Instance:of-type

Definition: represents the relationship between the ARRAYValue and the ARRAYTypes of which it is an instance.

Multiplicity: 1..* unordered

10.4.3.4 Other Roles

none.

10.4.4 Class: BagMember

Definition: Represents the relationship between a BAGValue and one value of the base data type (the “member” value). It has a “count” attribute that represents the number of times the given member-value occurs in the BAGValue.

10.4.4.1 Supertypes

none.

10.4.4.2 Attributes

Attribute: count **To: MOF::Integer**

Definition: represents the relationship between a BagMember and the number of occurrences of the member-value that it represents, i.e., the number of occurrences of that member-value in the bag.

Multiplicity: 1..1

10.4.4.3 Associations

AssociationEnd: member-value **To: Core::Instance**

Definition: represents the relationship between a BagMember and the Instance that it includes, one or more times, in the BAGValue.

Multiplicity: 1..1

10.4.4.4 Other Roles

From: BAGValue as member-slot

10.4.5 Class: BAGValue

Definition: an AggregateValue, representing a value of an EXPRESS BAG data type: a collection of instances of the member-type of the BAG, in which a given instance can appear more than once.

10.4.5.1 Supertypes

AggregateValue

10.4.5.2 Attributes

none.

10.4.5.3 Associations

AssociationEnd: member-slot **To: BagMember**

Definition: represents the relationship between a BagValue and each of its distinct member values. Each distinct member value is represented by a BagMember (slot) that counts its occurrences in the bag.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: of-type **To: Core::BAGType**

subsets: Core::Instance:of-type

Definition: represents the relationship between the BAGValue and the BAGTypes of which it is an instance.

Multiplicity: 1..* unordered

10.4.5.4 Other Roles

none.

10.4.6 Class: GenericAggregate

Definition: An AggregateValue representing the output of an AggregateInitializer. It is interpreted as a LIST value whose member-type is GENERIC, but actually constrained to the common DataType of all the Expressions in the Initializer. It can be coerced to an ARRAY, BAG, SET, or LIST value of the appropriate member-type, according to the context of its use.

Note – Certain GenericAggregate values have a syntactic parse as a LIST of instances, but no clear semantics as to data type; this is a defect in Part 11.

Note – See 12.9 of ISO 10303-11:2004.

10.4.6.1 Supertypes

LISTValue

10.4.6.2 Attributes

none.

10.4.6.3 Associations

none.

10.4.6.4 Other Roles

From Expressions::AggregateInitializer as result-value

10.4.7 Class: ListMember

Definition: represents one position in a ListValue and the instance of the member-type in that position.

10.4.7.1 Supertypes

none.

10.4.7.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: the ordinal identifier for the position in the sequence.

Multiplicity: 1..1

10.4.7.3 Associations

AssociationEnd: member-value **To: Core::Instance**

Definition: represents the relationship between a position in a LISTValue (represented by a ListMember) and the Instance that appears in that position.

Multiplicity: 1..1

10.4.7.4 Other Roles

From: LISTValue as member-slot

10.4.8 Class: LISTValue

Definition: an AggregateValue, representing a value of an EXPRESS LIST data type: a sequence of instances of the member-type of the LIST.

10.4.8.1 Supertypes

AggregateValue

10.4.8.2 Attributes

none.

10.4.8.3 Associations

AssociationEnd: member-slot **To: ListMember**

Definition: represents the relationship between a ListValue and each of its distinct slots for member values. Each member-slot represents a position in the ListValue.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: of-type **To: Core::LISTType**

subsets: Core::Instance:of-type

Definition: represents the relationship between the LISTValue and the LISTTypes of which it is an instance.

Multiplicity: 1..* unordered

10.4.8.4 Other Roles

none.

10.4.9 Class: SETValue

Definition: an AggregateValue representing a value of a SET data type.

Note – A SETValue can be viewed as a specialization of a BAGValue in which the “count” value for each BagMember is 1. But technically, the conversion of the SETValue to the corresponding BAGValue is a coercion, because the behavior of the resulting BAGValue is different. For example, the union of two SETValues is different from the union of the corresponding BAGValues.

10.4.9.1 Supertypes

AggregateValue

10.4.9.2 Attributes

none.

10.4.9.3 Associations

AssociationEnd: member-value **To: Core::Instance**

Definition: represents the relationship between a SETValue and the Instances that appear in it. Any given Instance can take this role at most once for any given SetValue.

Multiplicity: 0..* unordered

AssociationEnd: of-type **To: Core::SETType**

subsets: Core::Instance:of-type

Definition: represents the relationship between the SETValue and the SETTypes of which it is an instance.

Multiplicity: 1..* unordered

10.4.9.4 Other Roles

none.

10.5 Entity Instances and Values

This section specifies the model of EntityInstances – instances that correspond to entity data types. It also specifies the model of PartialEntityValues, which are aggregates of entity attribute values that are constructed and manipulated by some Expressions.

Figure 27 depicts the model of entity instances. In general, entity instances represent real-world objects being described by the EXPRESS schema. What is captured in the information base is an EntityValue which is a representation of the current state of the real-world object. A SingleLeafInstance is an EntityInstance that has a model as a single EntityType. A MultiLeafInstance is an EntityInstance that has a model as an allowable collection of overlapping subtypes of modeled EntityTypes.

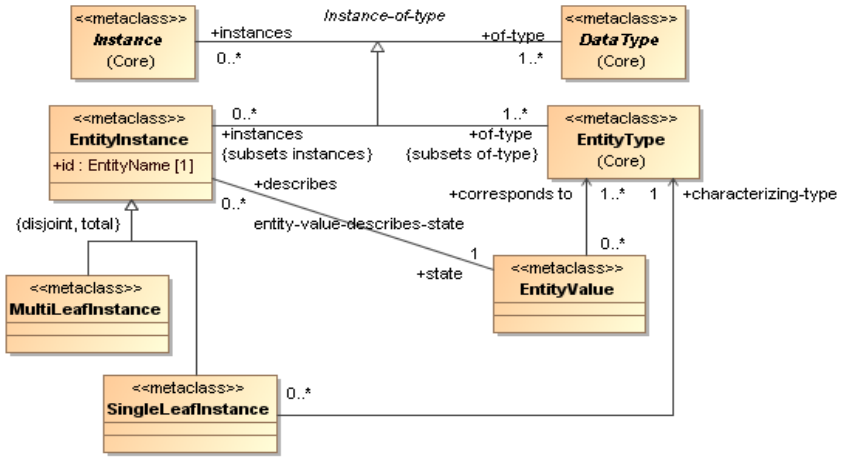

Figure 27 - Entity Instances

Figure 28 depicts the model of PartialEntityValues. A PartialEntityValue is a collection of information – assignments of values to named Attributes. Some PartialEntityValues are EntityValues, that is, they describe the state of an EntityInstance.

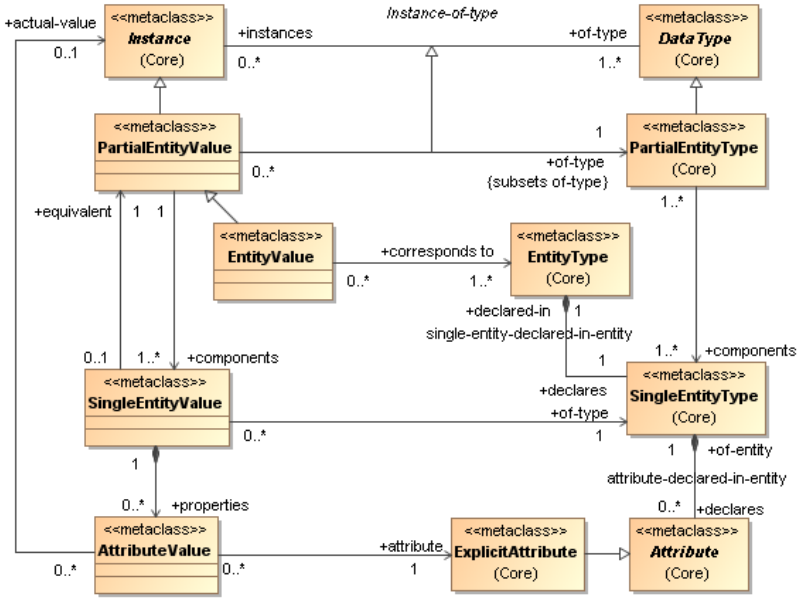

Figure 28 - PartialEntityValues

10.5.1 Class: AttributeValue

Definition: represents the assignment of a value to a given Attribute of the EntityType corresponding to the SingleEntityValue.

10.5.1.1 Supertypes

none.

10.5.1.2 Attributes

none.

10.5.1.3 Associations

AssociationEnd: actual-value **To: Core::Instance**

Definition: represents the value assigned to the Attribute by the AttributeValue. If the Attribute is declared OPTIONAL, it is possible that no value is assigned.

Multiplicity: 0..1

AssociationEnd: attribute **To: Core::ExplicitAttribute**

Definition: represents the relationship between the AttributeValue assignment and the ExplicitAttribute to which it assigns a value.

Multiplicity: 1..1

10.5.1.4 Other Roles

From: SingleEntityValue as properties

Multiplicity: 1..1 composite

10.5.2 Class: EntityInstance

Definition: a TaggedInstance that represents an EXPRESS entity instance – an instance of an entity data type, a view of an object that incorporates those properties and relationships that are significant to some particular purpose(s). The EntityInstance is distinct from the EntityValue – a collection of information about the object that represents those properties and relationships.

Note – See clause 5 of ISO 10303-11:2004.

10.5.2.1 Supertypes

TypedInstance

10.5.2.2 Attributes

Attribute: id **To: EntityName**

Definition: represents a nominal identifier for an EntityInstance that distinguishes it from other EntityInstances. The nature of this identifier is not defined in EXPRESS, but it is stated that this identifier is not necessarily constructed from any group of modeled attribute values. Each EntityName is unique within a Population, but the actual namespace of an EntityName is not specified in Part 11.

Note – See clause 5 of ISO 10303-11:2004.

Multiplicity: 1..1

10.5.2.3 Associations

AssociationEnd: of-type **To: Core::EntityType**

via: instance-of-EntityType

subsets: Core::Instance:of-type

Definition: represents the relationship between an EntityInstance and each of the EntityType classifiers it satisfies.

Multiplicity: 1..* unordered

AssociationEnd: state **To: EntityValue**

via: entity-value-describes-state

Definition: represents the relationship between the EntityInstance and the EntityValue that describes the current state of the Instance (in terms of its modeled properties) at any given time.

Multiplicity: 1..1

10.5.2.4 Other Roles

From: Rules::Extent as content

10.5.3 Datatype: EntityName

Definition: represents the unique underlying identity of an entity instance, expressed as some kind of identifier. The nature of this identifier is not defined in EXPRESS, but it is stated that this identifier is not necessarily constructed from any group of modeled attribute values. Each EntityName is unique within a Population, but the actual namespace of an EntityName is not specified in Part 11.

Note – See clause 5 of ISO 10303-11:2004.

10.5.3.1 Supertypes

Realization type is . MOF::String

The realization relationship is modeled as a generalization.

10.5.3.2 Members

none.

10.5.4 Class: EntityValue

Definition: A PartialEntityValue that completely describes an Instance of some EntityType(s).

10.5.4.1 Supertypes

PartialEntityValue

10.5.4.2 Attributes

none.

10.5.4.3 Associations

AssociationEnd: corresponds to **To: Core::EntityType**

Definition: represents the EntityType(s) whose complete modeled description comprises a set of Attributes that is contained in the EntityValue. The complete modeled description of an EntityType is a set of SingleEntityTypes, and the EntityValue contains SingleEntityValues corresponding to each of them.

Multiplicity: 1..* unordered

AssociationEnd: describes **To: EntityInstance**

via: entity-value-describes-state

Definition: represents the EntityInstances, if any, whose current state is described by the EntityValue. This direction of the association is only significant when the EntityValue is used as the means of identification of a particular EntityInstance.

Multiplicity: 0..* unordered

10.5.4.4 Other Roles

none.

10.5.5 Class: MultiLeafInstance

Definition: A (complex) EntityInstance that is a valid instance of more than one EntityType and whose state includes more SingleEntityValues than are declared for, or inherited by, any named EntityType defined in the governing Schema. The subtype/supertype graph corresponding to such an EntityInstance has multiple “leaf” nodes.

Note – This concept appears in Part 11 only in 3.3.12, but it appears in ISO 10303-21:2002 as an “uncharacterized instance” whose representation requires the “external mapping.”

10.5.5.1 Supertypes

EntityInstance

10.5.5.2 Attributes

none.

10.5.5.3 Associations

none.

10.5.5.4 Other Roles

none.

10.5.6 Class: PartialEntityValue

Definition: an Instance that is a collection of Attributes (of SingleEntityTypes) with associated values.

10.5.6.1 Supertypes

Core::Instance

10.5.6.2 Attributes

none.

10.5.6.3 Associations

AssociationEnd: components **To: SingleEntityValue**

Definition: the SingleEntityValues that make up the PartialEntityValue.

Multiplicity: 1..* unordered

Properties: composite

AssociationEnd: of-type **To: Core::PartialEntityType**

Definition: represents the relationship between a PartialEntityValue and the PartialEntityType that identifies the collection of SingleEntityTypes for which the PartialEntityValue provides values.

Multiplicity: 1..1

10.5.6.4 Other Roles

From: SingleEntityValue as equivalent

Multiplicity: 0..1

From: Expressions::PartialEntityConstructor as result-value

10.5.7 Class: SingleEntityValue

Definition: A collection of values for the explicit Attributes of exactly one SingleEntityType.

Note – A SingleEntityValue is not an Instance; it is a part of a PartialEntityValue. It cannot be the result of an Expression, nor can it be the value of any EXPRESS concept. The result of a PartialEntityConstructor is the .equivalent PartialEntityValue.

10.5.7.1 Supertypes

none.

10.5.7.2 Attributes

none.

10.5.7.3 Associations

AssociationEnd: equivalent **To: PartialEntityValue**

Definition: represents the relationship between a SingleEntityValue and the PartialEntityValue that consists of exactly that one SingleEntityValue.

Multiplicity: 1..1

AssociationEnd: of-type **To: Core::SingleEntityType**

Definition: represents the relationship between a SingleEntityValue and the SingleEntityType that declares the Attributes whose values are contained in the SingleEntityValue.

Note – While the relationship between a SingleEntityValue and a SingleEntityType appears to be an Instance-to-Type relationship, it is not treated as such in the metamodel, because SingleEntityValues are not Instances â_“ they can only appear as components of a PartialEntityValue.

Multiplicity: 1..1

AssociationEnd: properties **To: AttributeValue**

Definition: represents the relationship of the SingleEntityValue to the AttributeValue assignments it comprises.

Multiplicity: 0..* unordered

Properties: composite

10.5.7.4 Other Roles

From: PartialEntityValue as components

Multiplicity: 1..1 composite

10.5.8 Class: SingleLeafInstance

Definition: An EntityInstance that is completely characterized by a single EntityType (and all its supertypes) that is declared in the governing Schema

Note – This concept does not appear in Part 11, but is the “characterized instance” that is the basis for the “internal mapping” in ISO 10303-21:2002.

10.5.8.1 Supertypes

EntityInstance

10.5.8.2 Attributes

none.

10.5.8.3 Associations

AssociationEnd: characterizing-type **To: Core::EntityType**

Definition: represents the unique EntityType classifier that has (defines or inherits) exactly all of the Attributes present in the representation of the EntityInstance. Not every EntityInstance has a characterizing-type – it may be an “instance-of” two or more EntityTypes for which the intersection is not explicitly modeled, but permitted by the model to be non-empty.

Multiplicity: 1..1

10.5.8.4 Other Roles

none.

10.5.9 Association: entity-value-describes-state

Definition: represents the relationship between an EntityInstance and the EntityValue that describes the current state of the Instance (in terms of its modeled properties) at any given time.

10.5.9.1 Association Ends

AssociationEnd: describes **To: EntityInstance**

Definition: represents the EntityInstances, if any, whose current state is described by the EntityValue. This direction of the association is only significant when the EntityValue is used as the means of identification of a particular EntityInstance.

Multiplicity: 0..* unordered

AssociationEnd: state **To: EntityValue**

Definition: represents the relationship between the EntityInstance and the EntityValue that describes the current state of the Instance (in terms of its modeled properties) at any given time.

Multiplicity: 1..1

10.5.10 Association: instance-of-EntityType

Definition: represents the relationship between an EntityInstance and each of the EntityType classifiers it satisfies.

10.5.10.1 Association Ends

AssociationEnd: of-type **To: Core::EntityType**

subsets: Core::Instance:of-type

Definition: represents the relationship between an EntityInstance and each of the EntityType classifiers it satisfies.

Multiplicity: 1..* unordered

AssociationEnd: instances **To: EntityInstance**

subsets: Core::DataType:instances

Definition: represents the relationship between an EntityType (classifier) and the EntityInstances that satisfy it.

Multiplicity: 0..* unordered

10.6 Constants

This section defines the Constant concept. A Constant is a model element that provides a name for an instance of any data type, and allows the instance it names to be specified as the value of an Expression. Figure 29 depicts the model of Constants. The Constant class and its properties are described below.

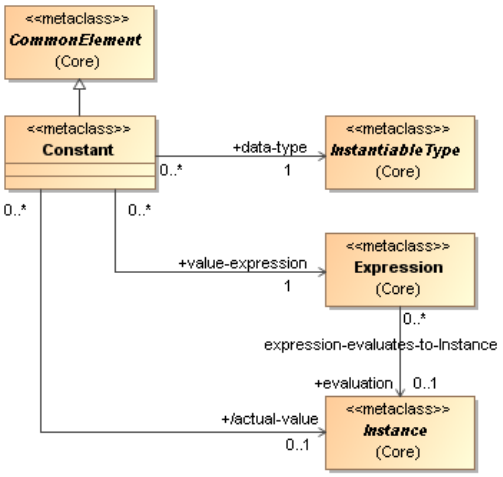

Figure 29 Constants

10.6.1 Class: Constant

Definition: a CommonElement that denotes a single instance value throughout each of its life cycles. The instance value is described by an Expression that evaluates to the value to be used in each instantiation of the Constant.

For a Constant that is defined as a SchemaElement, the value is unchanged across all algorithms and rules, and over all corresponding populations. It is a constant and names a specific Instance. Its `:value expression` may only refer to Instances and other Constants. A Constant defined in an AlgorithmScope, however, assumes a value for a given invocation of the Algorithm or Rule, but may assume different values for different invocations. Its `:value expression` may refer to parameters of the Algorithm or to elements of the population.

Note – "Constant" is a reserved word in EXPRESS; if this metamodel is converted to EXPRESS, this class must be renamed. See clause 9.4 of ISO 10303-11:2004.

10.6.1.1 Supertypes

Core::CommonElement

10.6.1.2 Attributes

none.

10.6.1.3 Associations

AssociationEnd: actual-value **To: Core::Instance**

Definition: represents the value resulting from evaluating the value-expression. This value may only be computable for a given population, or it may require computational capabilities a given agent does not have.

Multiplicity: 0..1

Properties: derived

Tagged Values

```
derivation = self->value-expression->evaluation;
```

AssociationEnd: data-type **To: Core::InstantiableType**

Definition: represents the relationship between the Constant and the DataType of the Instance denoted by the Constant.

Multiplicity: 1..1

AssociationEnd: value-expression **To: Core::Expression**

Definition: represents the Expression that specifies the value of the Constant for a given lifetime.

Multiplicity: 1..1

10.6.1.4 Other Roles

From: Expressions::ConstantRef as refers-to

10.6.1.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every Constant shall have an Identifier.

10.7 Populations

This section defines the Population concept and its relationship to Schemas and Instances. A population represents an information base that corresponds to a Schema. Figure 30 depicts the model of Population. The class Population and its associations are described below.

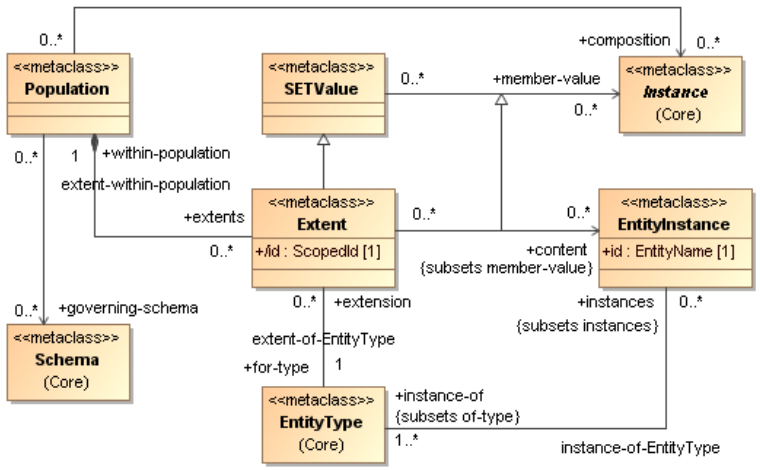

Figure 30 Populations and Instances

10.7.1 Class: Extent

Definition: the collection of all Instances in a given Population that satisfy the specified EntityType. That is, Extent is the SetValue that is the intersection of `EntityType:instances` and `Population:composition`.

Note – See 9.6 of ISO 10303-11:2004.

10.7.1.1 Supertypes

SETValue

10.7.1.2 Attributes

Attribute: id **To: Core::ScopedId**

Definition: the identifier for the EntityType, used as a name for the Extent.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: derived.

TaggedValues

```
derivation = self->for-type->id
```

10.7.1.3 Associations

AssociationEnd: content **To: EntityInstance**

Subsets: SETValue:member-values

Definition: represents the relationship between the Extent (within a Population) and the EntityInstances it contains. Extent is a SetValue and Extent:content is just the relationship between that SetValue and its members.

Multiplicity: 0..* unordered

AssociationEnd: for-type **To: Core::EntityType**

via: extent-of-EntityType

Definition: the EntityType to which the Extent corresponds.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: within-population **To: Population**

via: extent-within-population

Definition: the Population from which the Set of instances is drawn.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..1

10.7.1.4 Other Roles

none.

10.7.2 Class: Population

Definition: represents the collection of all entity instances over which the LocalRules and GlobalRules of a schema are to be evaluated.

The EXPRESS interpretation of Population is the complete closed collection of entity instances that is used for a particular purpose, such as the content of a database or an exchange document. Many distinct Populations may have the same governing-schema. The presumption is that the Population will be realized when the EntityInstances are realized, but it is not necessary that that realization will itself be represented as instance of this Package.

Note – See clause 5 of ISO 10303-11:2004.

10.7.2.1 Supertypes

none.

10.7.2.2 Attributes

none.

10.7.2.3 Associations

AssociationEnd: composition **To: Core::Instance**

via: instance-appears-in-population

Definition: represents the relationship between a Population and the Instances that make it up.

Multiplicity: 0..* unordered

AssociationEnd: extents **To: Extent**

via: extent-within-population

Definition: the collection of Extents of EntityTypes that make up the Population.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: governing-schema **To: Core::Schema**

Definition: represents the relationship between a Population and a Schema that governs (models, describes) it.

Note – See 9.3 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

10.7.2.4 Other Roles

none.

10.7.3 Association: extent-of-EntityType

Definition: represents the relationship between an EntityType and its Extent (the set of corresponding EntityInstances) in a given Population.

10.7.3.1 Association Ends

AssociationEnd: extension **To: Extent**

Definition: represents the relationship between an EntityType and its extension (the set of corresponding EntityInstances) in a given Population.

Multiplicity: 0..* unordered

AssociationEnd: for-type **To: Core::EntityType**

Definition: represents the relationship between an Extent and the EntityType to which it corresponds.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..1

10.7.4 Association: extent-within-population

Definition: represents the relationship between an Extent and the Population from which it is drawn.

10.7.4.1 Association Ends

AssociationEnd: extents **To: Extent**

Definition: the collection of Extents of EntityTypes that make up the Population.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: within-population **To: Population**

Definition: the Population from which the Set of instances constituting the Extent is drawn.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..1

10.7.5 Association: population-includes-instance

Definition: represents the relationship between an Instance and the Populations in which it appears.

10.7.5.1 Association Ends

AssociationEnd: appears-in-population **To: Population**

Definition: represents the relationship between an Instance and the Populations in which it appears.

Multiplicity: 0..* unordered

AssociationEnd: composition **To: Core::Instance**

Definition: represents the relationship between a Population and the Instances the make it up.

Multiplicity: 0..* unordered

10.8 Instance Package: BuiltInConstants

This Package represents the values of the “built-in constants” of the EXPRESS language. They are here modeled as individual objects that are instances of subtypes of SimpleValue.

Note – See clause 14 of ISO 10303-11:2004.

Note – The built-in constants are also modeled as Literals in Clause 12.11, i.e., as the Expressions that refer to these values.

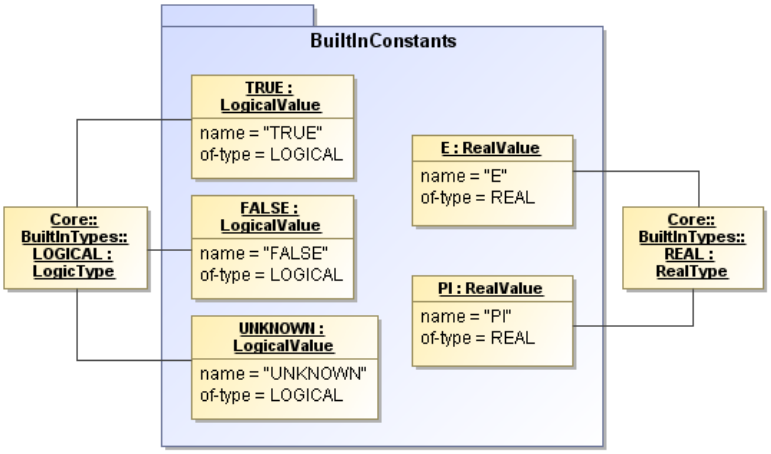

Figure 31 - Built-In Constants

10.8.1 Dependencies

Dependency on Class: Instances::SimpleValue

Stereotypes: instantiates

This Package provides base individuals that are always instances of class SimpleValue.

10.8.2 Instance: E

Type: Instances::RealValue

Definition: Represents the unique REAL number *e* such that the area above the x-axis and below the curve 1/x, for $1 \leq x \leq e$, is exactly 1.

Note – See clause 14.1 of ISO 10303-11:2004.

10.8.2.1 Slots

Attribute: name **Value: “E”**

Attribute: of-type **Values: Core::BuiltInTypes::REAL**

10.8.3 Instance: FALSE

Type: Instances::LogicalValue

Definition: Represents the LOGICAL value that is the evaluation of a proposition whose negation is asserted.

Note – See clause 14.3 of ISO 10303-11:2004.

10.8.3.1 Slots

Attribute: name **Value: “FALSE”**

Attribute: of-type **Values: Core::BuiltInTypes::LOGICAL**

10.8.4 Instance: PI

Type: Instances::RealValue

Definition: Represents the REAL value that is the ratio of the circumference of a circle to its diameter.

Note – See clause 14.4 of ISO 10303-11:2004.

10.8.4.1 Slots

Attribute: name **Value: “PI”**

Attribute: of-type **Values: Core::BuiltInTypes::REAL**

10.8.5 Instance: TRUE

Type: Instances::LogicalValue

Definition: Represents the LOGICAL value that is the evaluation of a proposition that is asserted.

Note – See clause 14.6 of ISO 10303-11:2004.

10.8.5.1 Slots

Attribute: name	**Value: “TRUE”**
Attribute: of-type	**Values: Core::BuiltInTypes::LOGICAL**

10.8.6 Instance: UNKNOWN

Type: Instances::LogicalValue

Definition: Represents the LOGICAL value that is the evaluation of an Expression that involves Indeterminate values. UNKNOWN is a specialization of the Indeterminate value that is treated only as a value of data type LOGICAL.

Note – See clause 14.7 of ISO 10303-11:2004.

10.8.6.1 Slots

Attribute: name	**Value: “UNKNOWN”**
Attribute: of-type	**Values: Core::BuiltInTypes::LOGICAL**

11 Package : Algorithms

The Algorithms Package contains the concepts related to definitions of Algorithms and Functions in EXPRESS.

11.1 Dependencies

Dependency on Package: Core

Stereotypes: `import`

The Algorithms Package depends on the Core Package for the NamedElement and Scope concepts, for data type concepts, and for the basic Expression concept.

11.2 Functions and Procedures

This section defines all the major concepts in EXPRESS Algorithm definitions, except for Variables and their data types. Those concepts are described in subsequent clauses below. Figure 32 depicts the concepts described in this section.

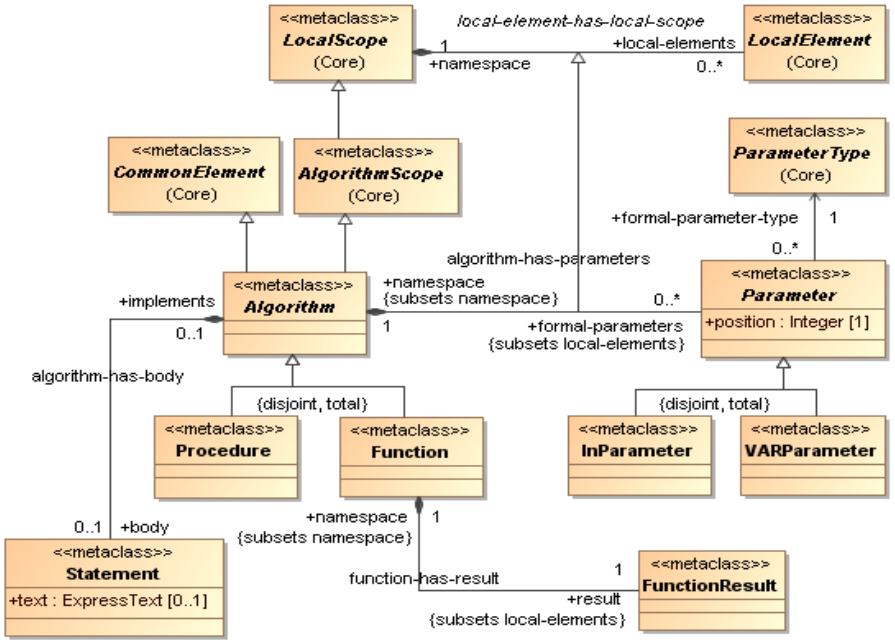

Figure 32 - Algorithms

11.2.1 Class: Algorithm

Definition: a CommonElement that represents an operation or process that transforms information. Every Algorithm is either a Procedure or a Function. Every Algorithm is also an AlgorithmScope, in that it may define CommonElements and local ModelElements.

Note – See 9.5 of ISO 10303-11:2004.

Properties: abstract

11.2.1.1 Supertypes

Core::CommonElement, Core::AlgorithmScope

11.2.1.2 Attributes

none.

11.2.1.3 Associations

AssociationEnd: actual-types **To: Core::ActualType**

via: scope-of-actual-type

Subsets: Core::LocalScope:local-elements

Definition: the ActualTypes that are defined in the Algorithm.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: body **To: Statement**

via: algorithm-has-body

Definition: represents the relationship between a (conceptual) Algorithm and a definition of the Algorithm as a Statement. In most cases, the Statement is a StatementBlock – a sequence of actions to be performed. The body of the Algorithm is modeled as optional (0..1). Support for the body is not a requirement for the support of Algorithms.

Note – See 9.5 of ISO 10303-11:2004.

Multiplicity: 0..1

Properties: composite

AssociationEnd: formal-parameters **To: Parameter**

via: algorithm-has-parameters

Subsets: Core::LocalScope:local-elements

Definition: represents the relationship between the Algorithm and its formal parameters.

Multiplicity: 0..* unordered

Properties: composite

11.2.1.4 Other Roles

From: Core::ActualType as scope

11.2.1.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every Algorithm has an identifier

11.2.2 Class: Function

Definition: an Algorithm that returns a single Instance and can appear in an Expression.

Note – "Function" is a reserved word in EXPRESS; if this metamodel is converted to EXPRESS, this class must be renamed. See 9.5.1 of ISO 10303-11:2004.

11.2.2.1 Supertypes

Algorithm

11.2.2.2 Attributes

none.

11.2.2.3 Associations

AssociationEnd: result **To: FunctionResult**

via: function-has-result

Subsets: Core::LocalScope:local-elements

Definition: represents the relationship between a Function and its FunctionResult.

Note – See 9.5.1 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: composite

11.2.2.4 Other Roles

From: Expressions::FunctionCall as invokes-function

11.2.3 Class: FunctionResult

Definition: the formal parameter representing the result Instance that is returned by the invocation of a Function. Within the body of the Function, the FunctionResult is a Variable that is denoted by the Algorithm identifier. Upon termination of the execution of the function-body, the (current) value of that Variable is returned.

Note – See 9.5.1 of ISO 10303-11:2004.

11.2.3.1 Supertypes

Variable

11.2.3.2 Attributes

none.

11.2.3.3 Associations

AssociationEnd: namespace **To: Function**

via: function-has-result

Subsets: Core::LocalElement:namespace

Definition: the Function that is the Scope in which the Function name refers to the FunctionResult.

Multiplicity: 1..1

11.2.3.4 Other Roles

From: Expressions::FunctionCall as returns-result

11.2.3.5 Rules

Constraint (OCL)

```
self->id = self->namespace->id;
```

The identifier for the function result is the identifier for the function.

11.2.4 Class: InParameter

Definition: a formal parameter to a Procedure or Function to which the ActualParameter is passed “by value.”

During an invocation of the Algorithm, the InParameter is a Variable that is initially set to the value of the corresponding ActualParameter. The value of the InParameter can be changed during the execution of the Algorithm.

An InParameter has a formal-parameter-type, which is the type specification to which the corresponding ActualParameters are required to conform. The InParameter also has a variable-type, which is the type specification for the Variable created to hold the value during invocation of the Algorithm. When the formal-parameter-type is an InstantiableType, the variable-type is the same type. When the formal parameter-type is a GeneralizedType, the variable-type is the corresponding ActualType.

Note – It is possible that the formal-parameter-type is itself an ActualType, if the Algorithm is defined within another Algorithm. In such a case, the variable-type is the same type.

Note – See 9.5.3 of ISO 10303-11:2004.

11.2.4.1 Supertypes

Parameter, Variable

11.2.4.2 Attributes

none.

11.2.4.3 Associations

none.

11.2.4.4 Other Roles

From: Expressions::PassByValue as for-parameter

11.2.5 Class: Parameter

Definition: a formal parameter – the formal description of an operand – of a Procedure or Function.

Parameters are of two kinds:

- InParameter, to which the values of the corresponding ActualParameters are passed by value.
- VarParameter, to which the corresponding ActualParameters are passed by reference

A Parameter is actually a NamedVariable whose scope is the Algorithm, and in each invocation of the Algorithm its (initial) value is set from the value or reference provided as the actual parameter. The formal-parameter-type of the Parameter constrains the types/values of the corresponding actual parameters. As a NamedVariable, it also has a variable-type, which is its data type for the purpose of operations within the body of the Algorithm. If the formal-paramater-type is an InstantiableType or an ActualType, the variable-type is the same type. If the formal-parameter-type is a GeneralizedType, the variable-type is the corresponding ActualType.

Note – See 9.5.3 of ISO 10303-11:2004.

Properties: abstract

11.2.5.1 Supertypes

Core::ElementSource

Note – Parameter is an abstract classifier. The two instantiable subclasses of Parameter – InParameter and VARParameter -- are subclasses of NamedVariable. So Parameter is an implicit subclass of NamedVariable.

11.2.5.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: A positive integer value designating the ordinal position of the Parameter in the formal-parameter-list for the Algorithm that is its :namespace. The position is used to associate ActualParameters with the formal Parameter.

Note – See 9.5.3 of ISO 10303-11:2004.

Multiplicity: 1..1

11.2.5.3 Associations

AssociationEnd: formal-parameter-type **To: Core::ParameterType**

Definition: the specification for the required data type of the actual parameters (see 12.7.1) that correspond to the formal Parameter; the data type that represents the allowable values of the Parameter.

Multiplicity: 1..1

Note – The lexical `parameter_type` in EXPRESS may refer to an InstantiableType, an ActualType (if the Algorithm is defined within an outer AlgorithmScope) or a GeneralizedType, and when it is syntactically a `generalized_type`, it may include ActualTypeConstraints.

AssociationEnd: namespace **To: Algorithm**

via: algorithm-has-parameters

Subsets: Core::LocalElement:namespace

Definition: represents the relationship between the Parameter and the Algorithm of which it is a formal parameter, and therefore the Algorithm which is the namespace for its :id.

Multiplicity: 1..1

11.2.5.4 Other Roles

From: Expressions::ParameterRef as refers-to

From: Expressions::ActualParameter as formal-parameter

11.2.5.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every Parameter has an identifier

Constraint (OCL)

```
IF typeof(self->namespace) = 'Function' THEN NOT self->inout;
```

No parameter to a Function shall be a VAR parameter.

11.2.6 Class: Procedure

Definition: an Algorithm that is executed as an action in a FunctionBody.

Note – See 9.5.2 of ISO 10303-11:2004.

Note – “Procedure” is a reserved word in EXPRESS; if this metamodel is converted to EXPRESS, this class must be renamed.

11.2.6.1 Supertypes

Algorithm

11.2.6.2 Attributes

none.

11.2.6.3 Associations

none.

11.2.6.4 Other Roles

From: Statements::ProcedureCall as invokes

11.2.7 Class: Statement

Definition: An EXPRESS Statement, a directive to perform a certain set of operations.

Note – See Clause 13 of ISO 10303-11:2004.

Note – Even though Statement is technically an abstract classifier, it is represented by direct instances with text representations when the Statements compliance point is not supported.

11.2.7.1 Supertypes

none.

11.2.7.2 Attributes

Attribute: text **To: Core::ExpressText**

Definition: Represents the EXPRESS statement verbatim.

Multiplicity: 0..1

11.2.7.3 Associations

AssociationEnd: controlled-by **To: Statements::RepeatStatement**

via: Statements::repeat-has-body

Definition: the RepeatStatement that controls the iterated execution of the actions of the Statement.

Multiplicity: 0..1

AssociationEnd: implements **To: Algorithm**

via: algorithm-has-body

Definition: represents the relationship between a Statement and the Algorithm for which it specifies an implementation.

Multiplicity: 0..1

AssociationEnd: in-block **To: Statements::StatementBlock**

via: Statements::block-sequences-statements

Definition: represents the relationship between a Statement and the StatementBlock, if any, in which it occurs.

Note – This relationship is needed for ESCAPE statements and SKIP statements, whose interpretation requires a path back to the REPEAT statement that controls them (see 13.8.3). It may also be needed to associate a RETURN statement with the Algorithm that whose implementation contains it.

Multiplicity: 0..1

11.2.7.4 Other Roles

From: Rules::GlobalRule as supporting-body

Multiplicity: 0..1

From: Statements::AliasStatement as body

Multiplicity: 0..1

From: Statements::CaseAction as action

Multiplicity: 0..1

From: Statements::IfStatement as then-action

Multiplicity: 0..1

From: Statements::IfStatement as else-action

Multiplicity: 0..1

11.2.8 Class: VARParameter

Definition: A formal parameter to a Procedure that is used as a reference to the object that is the ActualParameter in a given invocation. That is, a VARParameter represents a parameter that is "passed by reference."

During an invocation of the Algorithm, theVARParameter is a VARVariable whose referent is specified by the VARExpression that is the corresponding ActualParameter. All references to a VARParameter (in Statements and Expressions) refer to its referent.

As a Parameter, the VARParameter has a formal-parameter-type, which is the type specification to which the corresponding ActualParameters are required to conform. As a VARVariable, its data type is the type of its referent.

Note – See 9.5.3 of ISO 10303-11:2004.

11.2.8.1 Supertypes

Parameter, VARVariable

11.2.8.2 Attributes

none.

11.2.8.3 Associations

none.

11.2.8.4 Other Roles

From: Statements::PassByReference as for-parameter

11.2.8.5 Rules

Constraint (OCL)

```
typeof(self->namespace)='Procedure';
```

Only a Procedure can have a VAR Parameter.

11.2.9 Association: algorithm-has-body

Definition: represents the relationship between a (conceptual) Algorithm and a definition of the Algorithm as a StatementBlock – a sequence of actions to be performed.

Note – See 9.5 of ISO 10303-11:2004.

11.2.9.1 Association Ends

AssociationEnd: body **To: Statement**

Definition: represents the relationship between a (conceptual) Algorithm and a definition of the Algorithm as a Statement. In most cases, the Statement is a StatementBlock – a sequence of actions to be performed. The body of the Algorithm is modeled as optional (0..1). Support for the body is not a requirement for the support of Algorithms.

Note – See 9.5 of ISO 10303-11:2004.

Multiplicity: 0..1

Properties: composite

AssociationEnd: implements **To: Algorithm**

Definition: represents the relationship between a FunctionBody and the Algorithm for which it specifies an implementation.

Multiplicity: 0..1

11.2.10 Association: algorithm-has-parameters

Definition: represents the relationship between an Algorithm and its formal parameters.

11.2.10.1 Supertypes

Core::local-element-has-local-scope

11.2.10.2 Association Ends

AssociationEnd: formal-parameters **To: Parameter**

Definition: represents the relationship between the Algorithm and its formal parameters.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: namespace **To: Algorithm**

Definition: represents the relationship between the Parameter and the Algorithm of which it is a formal parameter, and therefore the Algorithm which is the namespace for its :id.

Multiplicity: 1..1

11.2.11 Association: function-has-result

Definition: represents the relationship between a Function and its FunctionResult.

Note – See 9.5.1 of ISO 10303-11:2004.

11.2.11.1 Supertypes

Core::local-element-has-local-scope

11.2.11.2 Association Ends

AssociationEnd: namespace **To: Function**

Subsets: LocalElement:namespace

Definition: the Function that is the AlgorithmScope in which the Function name refers to the FunctionResult.

Multiplicity: 1..1

AssociationEnd: result **To: FunctionResult**

Subsets: LocalScope:local-elements

Definition: represents the relationship between a Function and its FunctionResult.

Note – See 9.5.1 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: composite

11.3 Variables

This section describes the concepts associated with Variables in EXPRESS. Variables are introduced in Algorithms and GlobalRules. Figure 33 depicts the concepts described in this section.

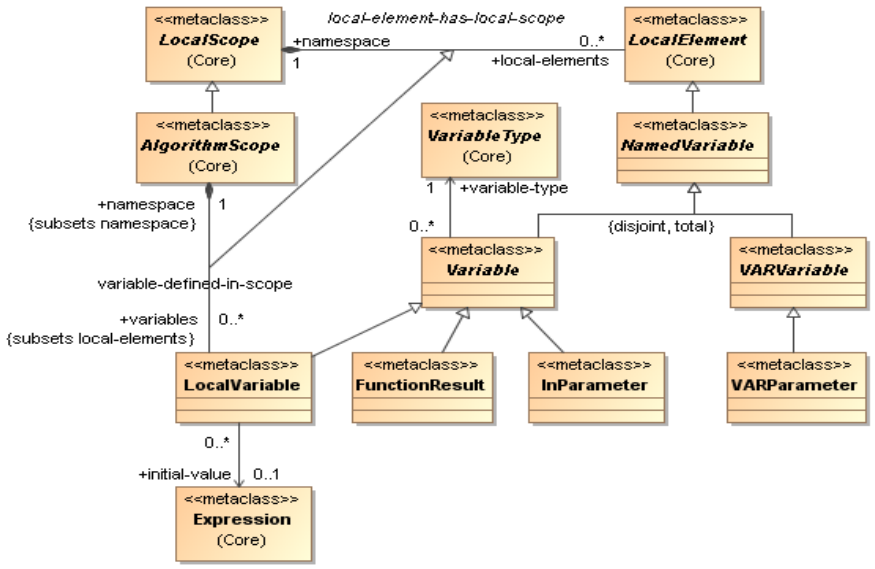

Figure 33 - Variables

11.3.1 Class: LocalVariable

Definition: a Variable that is declared as LOCAL to an Algorithm or GlobalRule and given an Identifier, and possibly an initial value, in the declaration.

Note – See 9.5.4 of ISO 10303-11:2004.

11.3.1.1 Supertypes

Variable

11.3.1.2 Attributes

none.

11.3.1.3 Associations

AssociationEnd: initial-value **To: Core::Expression**

Definition: represents the relationship between the LocalVariable and the Expression that specifies its initial-value on entry to the body of the Algorithm or GlobalRule that defines it.

Multiplicity: 0..1

AssociationEnd: namespace **To: Core::AlgorithmScope**

via: variable-defined-in-scope

Subsets: Core::LocalElement:namespace

Definition: represents the relationship between the LocalVariable and the AlgorithmScope in which it is defined. This is a refinement of the NamedElement:namespace relationship. The lifetime of a LocalVariable is exactly equal to the lifetime of the algorithm invocation or the GlobalRule evaluation that corresponds to the AlgorithmScope.

Multiplicity: 1..1

11.3.1.4 Other Roles

none.

11.3.2 Class: NamedVariable

Definition: Any EXPRESS syntactic variable: A LocalVariable, a QueryVariable, an increment ControlVariable, an AliasVariable, or a Parameter or FunctionResult. A NamedVariable is a NamedElement and always has a name/identifier. Each kind of NamedVariable has a different scope, but the scope of every NamedVariable is a LocalScope.

Every NamedVariable is either a Variable or a VARVariable.

Properties: abstract

11.3.2.1 Supertypes

Core::LocalElement

11.3.2.2 Attributes

none.

11.3.2.3 Associations

none.

11.3.2.4 Other Roles

From: Expressions::VariableRef as refers-to

11.3.2.5 Rules

Constraint (OCL)

```
exists(self->id);
```

Every NamedVariable has an identifier.

11.3.3 Class: VARVariable

Definition: A VARVariable represents a “pointer” that functions as a reference to a “cell” - a Variable, or a part of a Variable - during the execution of an Algorithm.

A VARVariable is a NamedVariable, but it is not a Variable. Unlike a Variable, it does not itself hold an Instance. Instead, it points to a cell that holds an Instance. The cell to which a VARVariable refers is called its *referent*. The referent of a VARVariable can be anything to which a VARExpression (see Section 14.10.5) can refer. The referent of a VARVariable is fixed at the time the instance of the VARVariable is created.

There are two kinds of VARVariables: VARParameter and AliasVariable.

Properties: abstract

11.3.3.1 Supertypes

NamedVariable

11.3.3.2 Attributes

none.

11.3.3.3 Associations

none.

11.3.3.4 Other Roles

From: Statements::AliasRef as refers-to

11.3.4 Class: Variable

Definition: a NamedVariable that exists during an invocation of an Algorithm or the evaluation of a GlobalRule and contains an Instance of a specified data type. (In essence, the type of a Variable specifies the structure of the object that contains the value.) During execution of an Algorithm, the Instance contained in a Variable can change.

Variables can be the objects of assignments or the referents of VARExpressions (see Section 14.10), and they have declared or implied data types that constrain their allowable values.

Note – See 9.5.4 of ISO 10303-11:2004. Part 11 uses the term “variable” to denote any of several kinds of objects that hold values, including LocalVariables, FunctionResults, Parameters, aggregate members, and ExplicitAttributes in EntityValues. The term Variable here only refers to LocalVariables, FunctionResults, and InParameters.

Properties: abstract

11.3.4.1 Supertypes

NamedVariable

11.3.4.2 Attributes

none.

11.3.4.3 Associations

AssociationEnd: variable-type **To: Core::VariableType**

Definition: the data type of the Variable - the type of the values that the Variable can contain.

In any given invocation, the data type of the Variable is an InstantiableType. If the data type of the Variable is specified as an InstantiableType, it is fixed for all invocations. If the data type of the Variable is specified as an ActualType, the actual data type varies from invocation to invocation, according to the data type of an actual parameter. If the Variable is an InParameter and its formal parameter type is a GeneralizedType, the variable-type is the corresponding ActualType.

Note – See 9.5.4 of ISO 10303-11:2004.

Multiplicity: 1..1

11.3.4.4 Other Roles

From: Statements::VariableCell as refers-to

11.3.5 Association: variable-defined-in-scope

Definition: represents the relationship between a LocalVariable and the AlgorithmScope in which it is defined. This is a refinement of the element-defined-in-scope relationship.

11.3.5.1 Supertypes

Core::local-element-has-local-scope

11.3.5.2 Association Ends

AssociationEnd: namespace **To: Core::AlgorithmScope**

Subsets: LocalElement:namespace

Definition: represents the relationship between the LocalVariable and the AlgorithmScope in which it is defined. This is a refinement of the NamedElement:namespace relationship. The lifetime of a LocalVariable is exactly equal to the lifetime of the algorithm invocation or the GlobalRule evaluation that corresponds to the AlgorithmScope.

Multiplicity: 1..1

AssociationEnd: variables **To: LocalVariable**

Subsets: LocalScope:local-elements

Definition: represents the relationship between the LocalScope and the set of LocalVariables that are defined within it.

Multiplicity: 0..* unordered

Properties: composite

11.4 Actual Types

In the simplest case, return values, variables, and other elements whose lifetime is the evaluation of the Algorithm are declared to have InstantiableTypes. But they can also be declared to be derivatives of the data types of the actual parameters in a given invocation. Figure 34 depicts the model of data types that have such declarations, herein called *ActualTypes*.

EXPRESS permits the `generalized_type` specifications for formal parameters to contain labeled generic components that refer to specific elements of the data type of the corresponding actual parameters. These labeled components are modeled as ParametricElements (see Section 8.14.2). The specifications of data types that are ActualTypes refer to ParametricElements, as shown in Figure 34.

All of these concepts are described in detail in this section.

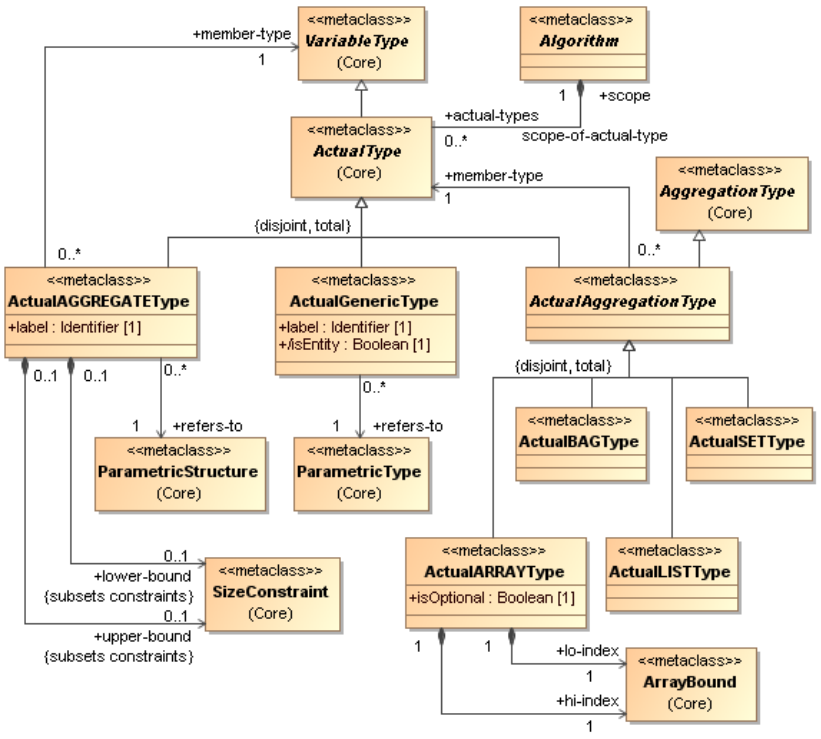

Figure 34 - Actual Types

11.4.1 Class: Core::ActualType

Definition: specification of an instantiable data type by reference to (a component of) the data type of the actual parameter that corresponds to a formal parameter of the Algorithm.

Each subtype of ActualType refers to a ParametricElement that is defined among the formal Parameters of the Algorithm. The ParametricElement denotes the corresponding component of the data type of the corresponding actual parameter in any given invocation. The ParametricElement is named by an EXPRESS `type_label`, and the ActualType refers to that ParametricElement via the `type_label`.

Note – The class ActualType is defined in the Core package (8.6.1).

11.4.2 Class: ActualAGGREGATEType

Definition: an ActualType that is an aggregation type whose structure is specified by a ParametricStructure, which refers to the structure of a (component of) an actual parameter. The .label attribute is used to determine the ParametricStructure to which it refers. The member-type of the ActualAGGREGATEType can be any VariableType (Instantiable or Actual) and need not have any relationship to the member type of the corresponding actual parameter.

Note – See 9.5.3.4 of ISO 10303-11:2004.

11.4.2.1 Supertypes

Core::ActualType

11.4.2.2 Attributes

Attribute: label **To: Core::Identifier**

Definition: Represents the EXPRESS `type_label` on the AGGREGATE type, which is used to associate it with the ParametricStructure that defines that identifier.

Note – The label on the ActualAGGREGATEType is not a definition of that symbol; it is a reference to the occurrence of that symbol as a label on a component of a formal parameter type that defines the label in the Algorithm namespace as the `id` for a ParametricStructure that defines what the actual structure is for each invocation. More than one ActualAGGREGATEType can have the same label and refer to the same structure.

Multiplicity: 1..1

11.4.2.3 Associations

AssociationEnd: lower-bound **To: Core::SizeConstraint**

Subsets: Core::ParameterType:constraints

Definition: represents a lower-bound constraint on aggregate values that are instances of the actual aggregation type corresponding to the AGGREGATE type. If the lower-bound constraint is present, the number of members of the aggregate value shall be greater than or equal to this value. If the lower-bound is not present or evaluates to zero, there is no constraint. Unless the lower-bound specified for the AGGREGATE type is an explicit “0,” this constraint shall appear.

Note – See 9.5.3.2 of ISO 10303-11:2004.

Multiplicity: 0..1

AssociationEnd: member-type **To: Core::VariableType**

Definition: represents the type of the components of the actual aggregation type that has the structure that corresponds to the AGGREGATE type. The type of the members may be an InstantiableType or an ActualType derived from a ParameterType.

Note – See 9.5.3.1 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: refers-to **To: ParametricStructure**

Definition: the ParametricStructure that defines the identifier that corresponds to the `:label` on the ActualAGGREGATEType. When instantiated, the ActualAGGREGATEType will have the structure of the component of the datatype of the ActualParameter that corresponds to this ParametricStructure.

Multiplicity: 1..1

AssociationEnd: upper-bound **To: Core::SizeConstraint**

Subsets: Core::ParameterType:constraints

Definition: represents an upper-bound constraint on aggregate values that are instances of the actual aggregation type corresponding to the AGGREGATE type. If the upper-bound constraint is present and does not evaluate to indeterminate ("?"), the number of members of the aggregate value shall be less than or equal to this value. If the upper-bound is not present or evaluates to indeterminate, there is no constraint. Unless the upper-bound specified for the AGGREGATE type is an explicit "?", this constraint shall appear.

Note – See 9.5.3.3 of ISO 10303-11:2004.

Multiplicity: 0..1

11.4.2.4 Other Roles

none.

11.4.3 Class: ActualAggregationType

Definition: An aggregation type whose member-type is an ActualType. An ActualAggregationType differs from an InstantiableAggregationType in that the data type of its components is dynamically specified.

Properties: abstract

11.4.3.1 Supertypes

Core::AggregationType , Core::ActualType

11.4.3.2 Attributes

none.

11.4.3.3 Associations

AssociationEnd: member-type **To: Core::ActualType**

Definition: represents the ActualType that is the the type of the component elements of the ActualAggregationType.

Note – If the member-type were not itself an ActualType, the ActualAggregationType would be an Instantiable AggregationType.

Multiplicity: 1..1

11.4.3.4 Other Roles

none.

11.4.4 Class: ActualARRAYType

Definition: An ActualAggregationType whose structure is an ARRAY with defined lower and upper bounds on the index.

11.4.4.1 Supertypes

ActualAggregationType

11.4.4.2 Attributes

Attribute: isOptional **To: MOF::Boolean**

Definition: True if the member type is declared to be OPTIONAL in the syntactic designation for the ARRAYType; False otherwise. When isOptional is True, any instance of the ARRAYType is permitted to have members whose value is unspecified ("?").

Note – See 8.2.1 of ISO 10303-11:2004.

Multiplicity: 1..1

11.4.4.3 Associations

AssociationEnd: hi-index **To: Core::ArrayBound**

Definition: represents the upper bound on the Integer index-range of each value of the ActualARRAYType.

Note – See 8.2.1 and 15.11 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: lo-index **To: Core::ArrayBound**

Definition: represents the lower bound on the Integer index-range of each value of the ActualARRAYType.

Note – See 8.2.1 and 15.11 of ISO 10303-11:2004.

Multiplicity: 1..1

11.4.4.4 Other Roles

none.

11.4.5 Class: ActualBAGType

Definition: An ActualAggregationType whose structure is a BAG (see 8.10.4).

11.4.5.1 Supertypes

ActualAggregationType

11.4.5.2 Attributes

none.

11.4.5.3 Associations

none.

11.4.5.4 Other Roles

none.

11.4.6 Class: ActualGenericType

Definition: an ActualType that refers to a ParametricType - the data type, or the member-type, of an actual parameter.

The `:label` attribute is used to determine the ParametricType to which it refers.

Note – See 9.5.3.4 of ISO 10303-11:2004.

11.4.6.1 Supertypes

Core::ActualType

11.4.6.2 Attributes

Attribute: isEntity **To: MOF::Boolean**

Definition: True if the ActualType is required to be an EntityType; False otherwise.

Multiplicity: 1..1

Properties: derived.

Tagged Values

```
derivation = self->refers-to->isEntity;
```

Attribute: label **To: Core::Identifier**

Definition: Represents the EXPRESS `type_label` on the GENERIC or GENERIC_ENTITY keyword, which is used to associate it with the ParametricType that defines that `type_label`.

Note – The label on the ActualGenericType is not a definition of that symbol; it is a reference to the occurrence of that symbol as a label on a component of a formal parameter type.

Multiplicity: 1..1

11.4.6.3 Associations

AssociationEnd: refers-to **To: Core::ParametricType**

Definition: the ParametricType that defines the identifier that corresponds to the `:label` on the ActualGenericType. When instantiated, the actual type will be the (component of the) datatype of the actual parameter that corresponds to this ParametricType.

Multiplicity: 1..1

11.4.6.4 Other Roles

none.

11.4.7 Class: ActualLISTType

Definition: An ActualAggregationType whose structure is a LIST. (See 8.10.6)

11.4.7.1 Supertypes

ActualAggregationType

11.4.7.2 Attributes

none.

11.4.7.3 Associations

none.

11.4.7.4 Other Roles

none.

11.4.8 Class: ActualSETType

Definition: An ActualAggregationType whose structure is a SET. (See 8.10.8)

11.4.8.1 Supertypes

ActualAggregationType

11.4.8.2 Attributes

none.

11.4.8.3 Associations

none.

11.4.8.4 Other Roles

none.

11.4.9 Association: scope-of-actual-type

Definition: represents the relationship between an ActualType and the Algorithm that is its scope.

11.4.9.1 Association Ends

AssociationEnd: scope **To: Algorithm**

Definition: The Algorithm in which the ActualType is specified. The ActualType must be the data type of a Variable or Attribute whose scope is contained in the Algorithm.

The ParametricElement that defines the `type_label` to which the ActualType refers shall be defined among the formal parameters of the Algorithm.

Note – An ActualType does not have a namespace; it defines no identifiers. The `:scope` of the ActualType represents the ownership of the ActualType and the lifetime of the ActualType.

Multiplicity: 1..1

AssociationEnd: actual-types **To: ActualType**

Definition: the set of ActualTypes that are defined in the Algorithm.

Multiplicity: 0..* unordered

Properties: composite

12 Package : Rules

The Rules Package contains the models of RULEs and SUBTYPE_CONSTRAINTS, which rely on the notion of extents of types with populations (see 9.7).

12.1 Dependencies

Dependency on Package: Core

Stereotypes: `import`

The Rules Package depends on the Core Package for the NamedElement and Scope concepts, for the EntityType concept, and for the basic Expression concept.

Dependency on Package: Algorithms

Stereotypes: `import`

The Rules Package depends on the Algorithms Package for the Variable and Statement concepts.

12.2 Global Rules

This section models the concepts used in EXPRESS RULE declarations. Figure 35 depicts the principal concepts.

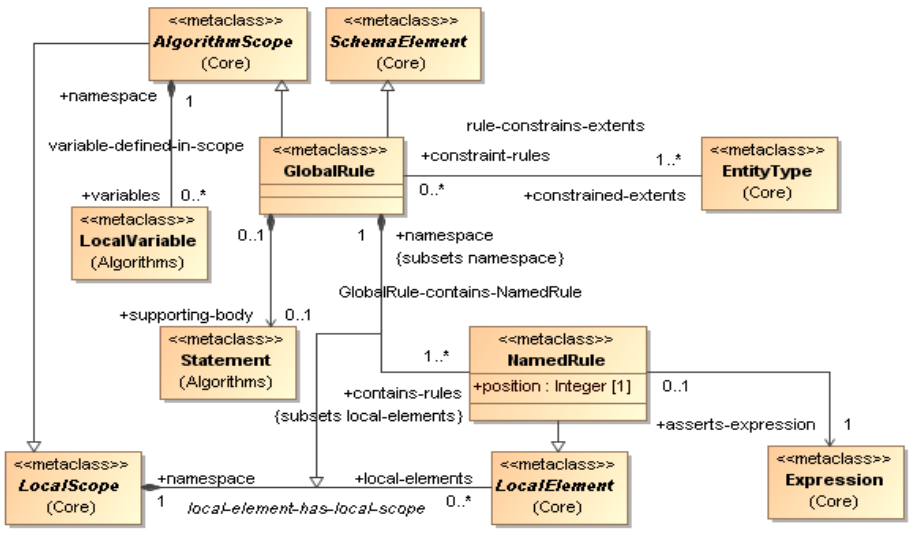

Figure 35 - Global Rules

12.2.1 Class: GlobalRule

Definition: a SchemaElement denoting a collection of NamedRules for the interaction of the Extents of one or more EntityTypes. It corresponds to the RULE declaration in EXPRESS. Every GlobalRule is also an AlgorithmScope and may define CommonElements and Variables.

Note – See 9.6 of ISO 10303-11:2004.

12.2.1.1 Supertypes

Core::AlgorithmScope, Core::SchemaElement

12.2.1.2 Attributes

none.

12.2.1.3 Associations

AssociationEnd: constrained-extents **To: Core::EntityType**

via: rule-constrains-extents

Definition: the EntityTypes whose Extents are constrained by the GlobalRule

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

AssociationEnd: contains-rules **To: NamedRule**

via: GlobalRule-contains-NamedRule

Subsets: Core::LocalScope:local-elements

Definition: represents the relationship between the GlobalRule (container) and the NamedRules it contains. Since the GlobalRule also constitutes the scope of the id (if any) for the NamedRule, this relationship is treated as a specialization of the Scope:named-elements relationship.

Multiplicity: 1..* unordered

Properties: composite

AssociationEnd: supporting-body **To: Algorithms::Statement**

Definition: represents the Statement, usually a StatementBlock, that provides values for LocalVariables used in the NamedRules that are contained in the GlobalRule.

The supporting-body of the GlobalRule can only appear if one or more LocalVariables are introduced for use in the NamedRules, and even then, the supporting-body is not required if the value of each LocalVariable is completely defined by an initializing expression.

If an implementation of the metamodel does not support the Statements compliance point, the supporting body should be captured as text when it is present.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 0..1

Properties: composite

12.2.1.4 Other Roles

none.

12.2.1.5 Rules

Constraint (OCL)

```
exists(self->defined-in);
```

Every GlobalRule shall be defined-in a Schema.

Constraint (OCL)

```
exists(self->id);
```

Every GlobalRule shall have an identifier

Constraint (OCL)

```
if exists(self->supporting-body) then exists(self->variables);
```

A GlobalRule cannot have a supporting body unless it defines LocalVariables.

12.2.2 Class: NamedRule

Definition: a constraint requiring a given Boolean Expression involving the Extents of one or more EntityTypes to evaluate to True. It corresponds to a domain rule contained in a Rule declaration in EXPRESS.

Note – See 9.6 of ISO 10303-11:2004.

12.2.2.1 Supertypes

Core::LocalElement

12.2.2.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: Represents the lexical position of the NamedRule in the sequence of NamedRules contained in the GlobalRule.

Multiplicity: 1..1

12.2.2.3 Associations

AssociationEnd: asserts-expression **To: Core::Expression**

Definition: represents the fact that every NamedRule states a Boolean expression that is required to be True for the Extents in a given Population.

Note – See 9.6 of ISO 10303-11:2004. The asserts-expression that formulates the NamedRule is wholly owned by the NamedRule. It is not treated as reusable.

Multiplicity: 1..1

AssociationEnd: namespace **To: GlobalRule**

via: GlobalRule-contains-NamedRule

Subsets: Core::LocalElement:namespace

Definition: represents the relationship between the NamedRule and the GlobalRule that contains it. This is a refinement of the NamedElement:namespace relationship to Scope. In addition to being the namespace for the id of the NamedRule, the GlobalRule identifies the EntityTypes to which the NamedRule applies (and whose Extents may be referred to in the asserts-expression) and may define Variables that are used in the asserts-expression.

Multiplicity: 1..1

12.2.2.4 Other Roles

None.

12.2.3 Association: GlobalRule-contains-NamedRule

Definition: represents the relationship between the GlobalRule (container) and the NamedRules it contains.

12.2.3.1 Supertypes

Core::local-element-has-local-scope

12.2.3.2 Association Ends

AssociationEnd: contains-rules **To: NamedRule**

Definition: represents the relationship between the GlobalRule (container) and the NamedRules it contains. Since the GlobalRule also constitutes the scope of the id (if any) for the NamedRule, this relationship is treated as a specialization of the Scope:named-elements relationship.

Multiplicity: 1..* unordered

Properties: composite

AssociationEnd: namespace **To: GlobalRule**

Definition: represents the relationship between the NamedRule and the GlobalRule that contains it. This is a refinement of the NamedElement:namespace relationship to Scope. In addition to being the namespace for the id of the NamedRule, the GlobalRule identifies the EntityTypes to which the NamedRule applies (and whose Extents may be referred to in the asserts-expression) and may define Variables that are used in the asserts-expression.

Multiplicity: 1..1

12.2.4 Association: rule-constrains-extents

Definition: represents the relationship between a GlobalRule and the EntityTypes whose Extents it constrains.

Note – See 9.6 of ISO 10303-11:2004.

12.2.4.1 Association Ends

AssociationEnd: constrained-extents **To: Core::EntityType**

Definition: represents the relationship between a GlobalRule and the Extents of the EntityTypes that it constrains.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

AssociationEnd: constraint-rules **To: GlobalRule**

Definition: represents the relationship between an EntityType and the GlobalRules that constrain it.

Note – See 9.6 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

12.3 SupertypeRules and SubtypeConstraints

This section models the concepts used in EXPRESS supertype clauses and SUBTYPE_CONSTRAINT declarations.

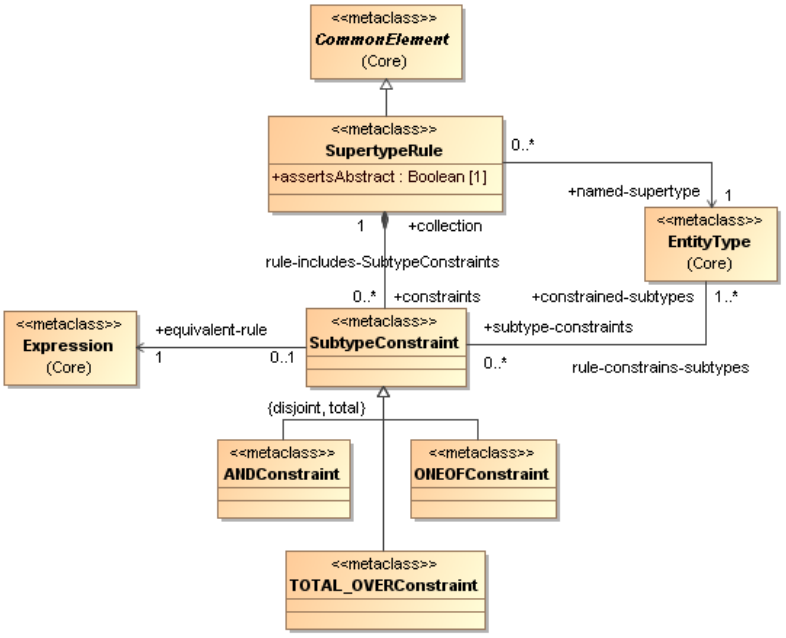

Figure 36 - Supertype Rules and Subtype Constraints

12.3.1 Class: ANDConstraint

Definition: a constraint requiring its two operands to be equal as sets. Each operand can be a single Extent or a union of Extents.

Note – See 9.2.5.4 of ISO 10303-11:2004.

12.3.1.1 Supertypes

SubtypeConstraint

12.3.1.2 Attributes

none.

12.3.1.3 Associations

none.

12.3.1.4 Other Roles

none.

12.3.2 Class: ONEOFConstraint

Definition: a constraint requiring all of its operands to be mutually exclusive. Each operand can be a single Extent or a union of Extents.

Note – See 9.2.5.2 of ISO 10303-11:2004.

12.3.2.1 Supertypes

SubtypeConstraint

12.3.2.2 Attributes

none.

12.3.2.3 Associations

none.

12.3.2.4 Other Roles

none.

12.3.3 Class: SubtypeConstraint

Definition: a Rule requiring a specific relationship among the Extents of two or more subtypes of a given supertype EntityType. The constraint can be stated as a relationship among the Extents as Sets of entity instances, and is equivalent to a NamedRule.

Note – See 9.2.5 of ISO 10303-11:2004.

12.3.3.1 Supertypes

none.

12.3.3.2 Attributes

none.

12.3.3.3 Associations

AssociationEnd: collection **To: SupertypeRule**

via: rule-includes-SubtypeConstraints

Definition: represents the relationship of a SubtypeConstraint to the SupertypeRule that contains it, which also identifies the common supertype.

Multiplicity: 1..1

AssociationEnd: constrained-subtypes **To: Core::EntityType**

via: rule-constrains-subtypes

Definition: the EntityTypes whose Extents are constrained by the SubtypeConstraint.

Note – See 9.2.5 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

AssociationEnd: equivalent-rule **To: Core::Expression**

Definition: represents the fact that every SubtypeConstraint is equivalent to a BooleanExpression involving the Extents of the EntityTypes named in the SubtypeConstraint. The Expression is required to evaluate to True. The effect is that the SubtypeConstraint is equivalent to a NamedRule.

Note – The equivalent-rule that formulates the SubtypeConstraint is wholly owned by the SubtypeConstraint. It is not treated as reusable.

Multiplicity: 1..1

12.3.3.4 Other Roles

None.

12.3.4 Class: SupertypeRule

Definition: a CommonElement representing a collection of rules requiring specific relationships among the Extents of two or more subtypes of a given supertype EntityType. The interpretation of a SupertypeRule is that all of the contained constraints shall hold. SupertypeRule corresponds to a SUBTYPE_CONSTRAINT declaration, or to the EXPRESS supertype-clause attached to an entity declaration.

A SupertypeRule shall have an `:id` value if and only if it represents an EXPRESS SUBTYPE_CONSTRAINT.

Note – This rule reflects the EXPRESS syntax. An EXPRESS supertype-clause has no identifier. An EXPRESS SUBTYPE_CONSTRAINT is required to have an identifier.

Note – See 9.2.5 and 9.7 of ISO 10303-11:2004.

12.3.4.1 Supertypes

Core::CommonElement

12.3.4.2 Attributes

Attribute: assertsAbstract **To: MOF::Boolean**

Definition: Represents a declaration in a SUBTYPE_CONSTRAINT that the .supertype EntityType is to be treated as ABSTRACT in this context, which is usually an interfacing schema.

Note – See clause 9.2.5.1 of ISO 10303-11:2004.

Multiplicity: 1..1

12.3.4.3 Associations

AssociationEnd: constraints **To: SubtypeConstraint**

via: rule-includes-SubtypeConstraints

Definition: represents the relationship between a SupertypeRule (supertype-clause or SUBTYPE_CONSTRAINT) and the individual subtype constraints it contains.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: named-supertype **To: Core::EntityType**

Definition: represents the relationship between a SupertypeRule and the EntityType that is the supertype of all the EntityTypes that appear in the SupertypeRule. This relationship is nominal for ANDConstraints and ONEOFConstraints, but significant for ABSTRACT and TOTAL_OVERConstraints.

Note – See 9.2.5 and 9.7 of ISO 10303-11:2004.

Multiplicity: 1..1

12.3.4.4 Other Roles

None.

12.3.5 Class: TOTAL_OVERConstraint

Definition: a constraint requiring the union of all of its operands to be equal to the Extent of the supertype.

Note – See 9.7.2 of ISO 10303-11:2004.

Note – The proper model of a TOTAL_OVER constraint requires that the supertype be one of the operands of the equivalent-expression and that the supertype be included among the constrained-subtypes.

12.3.5.1 Supertypes

SubtypeConstraint

12.3.5.2 Attributes

none.

12.3.5.3 Associations

none.

12.3.5.4 Other Roles

none.

12.3.6 Association: rule-constrains-subtypes

Definition: represents the relationship between a SubtypeConstraint and the Extents of the EntityTypes to which it refers.

Note – See 9.2.5 of ISO 10303-11:2004.

12.3.6.1 Association Ends

AssociationEnd: constrained-subtypes **To: Core::EntityType**

Definition: represents the relationship between a SubtypeConstraint and the EntityTypes whose Extents it constrains.

Note – See 9.2.5 of ISO 10303-11:2004.

Multiplicity: 1..* unordered

AssociationEnd: constraints **To: SubtypeConstraint**

Definition: represents the relationship between an EntityType and the SubtypeConstraints that involve it.

Note – See 9.2.5 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

12.3.7 Association: rule-includes-SubtypeConstraints

Definition: represents the relationship between a SupertypeRule (supertype-clause or SUBTYPE_CONSTRAINT) and the individual subtype constraints it contains.

12.3.7.1 Association Ends

AssociationEnd: collection **To: SupertypeRule**

Definition: represents the relationship of a SubtypeConstraint to the SupertypeRule that contains it, which also identifies the common supertype.

Multiplicity: 1..1

AssociationEnd: constraints **To: SubtypeConstraint**

Definition: represents the relationship between a SupertypeRule (supertype-clause or SUBTYPE_CONSTRAINT) and the individual subtype constraints it contains.

Multiplicity: 0..* unordered

Properties: composite

13 Package : Expressions

The Expressions Package contains the detailed modeling concepts for Expressions. The basic Expression model in the Core Package is permitted to be a syntactic string. This package provides the elements that support the operational semantics of the expression.

13.1 Dependencies

Dependency on Package: Core

Stereotypes: `import`

The Expressions Package depends on the Core Package for the basic Expression concept, for the basic Instance concept for Expression results, and for references to InstantiableTypes, SingleEntityTypes and Attributes.

Dependency on Package: Instances

Stereotypes: `import`

The Expressions Package depends on the Instances Package for the Instance concepts that correspond to Literals and other references to Constants.

Dependency on Package: Algorithms

Stereotypes: `import`

The Expressions Package depends on the Algorithms Package for the Variable concept, and for the Function and Parameter concepts used in FunctionCalls.

13.2 Overview of Expressions

Figure 37 provides the overview of Expression types. Expression and TextExpression are described in the Core package. FullExpression is the abstract class that represents the semantic model of an EXPRESS expression. It is described in this section. Each of its subclasses is described in a separate section below.

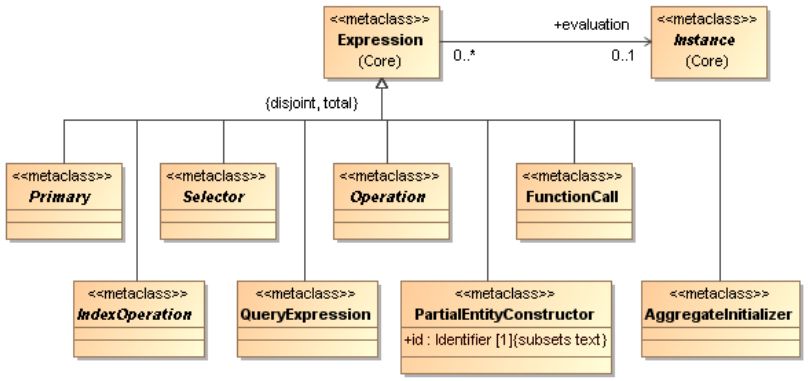

Figure 37 - Expressions

13.2.1 Class Core::Expression

Definition: In general, an Expression is the representation of an Instance by a set of computational operations that will produce that Instance when performed in the context in which the Expression occurs. An Expression is always evaluated in a context which determines the assignment of Instances to model elements (e.g., Variables, Attributes, etc.) that appear in the Expression. The Instance produced by the same Expression may vary from context to context. The Instance produced is said to be the value, or the evaluation, of the Expression.

Note – In general, Expressions are treated as reusable. It is recommended, however, that, except for literals and local variables, each occurrence should be a unique object. A few uses of Expression are not treated in the model as reusable, specifically those that are the definitions of Rules.

Note – Class Expression, and all of its properties, are defined in the Core Package, so that it can be used by other Packages, including Core, as necessary. This entry serves only to provide the Definition and a link to the complete specification in 8.17.1.

13.2.2 Class: IndexOperation

Definition: an Expression that returns a value “extracted from” a given base value.

Properties: abstract

13.2.2.1 Supertypes

Core::Expression

13.2.2.2 Attributes

none.

13.2.2.3 Associations

AssociationEnd: base-value **To: Core::Expression**

Definition: represents the base value from which the result value is to be extracted. For an AggregateIndex, the base-value Expression must evaluate to an AggregateValue. For a BinaryIndex, the base-value Expression must evaluate to a BINARY value. For a StringIndex, the base-value Expression must evaluate to a STRING Value.

Multiplicity: 1..1

13.2.2.4 Other Roles

none.

13.2.3 Class: Operation

Definition: an abstract subclass of Expression; represents the result of a well-defined mathematical operation or character manipulation.

Note – See clause 12 of ISO 10303-11:2004.

Properties: abstract

13.2.3.1 Supertypes

Core::Expression

13.2.3.2 Attributes

none.

13.2.3.3 Associations

none.

13.2.3.4 Other Roles

none.

13.2.4 Class: Primary

Definition: an abstract subclass of Expression representing a specific Instance, or the current value of an object that has a simple lexical designation.

Note – See 12.7 of ISO 10303-11:2004.

Properties: abstract

13.2.4.1 Supertypes

Core::Expression

13.2.4.2 Attributes

none.

13.2.4.3 Associations

none.

13.2.4.4 Other Roles

none.

13.2.5 Class: Selector

Definition: A FullExpression that returns the value of one or more Attributes of an EntityInstance.

Note – This concept does not appear in Part 11 per se, but the three subclasses all appear in Part 11 and have this property.

Properties: abstract

13.2.5.1 Supertypes

Core::Expression

13.2.5.2 Attributes

none.

13.2.5.3 Associations

AssociationEnd: entity-instance **To: Core::Expression**

Definition: represents the entity instance from which the Selector extracts the value of the named Attribute(s).

Note – See 12.7.3 of ISO 10303-11:2004.

Multiplicity: 1..1

13.2.5.4 Other Roles

none.

13.3 Primaries

This section describes the EXPRESS operations that return the values of named independent elements – Constants, Enumeration items, Extents, Variables, Parameters. It also includes SELF, which is a reference to the current instance of a data type, and Literals, which are specialized syntactic notations that refer to values of simple types.

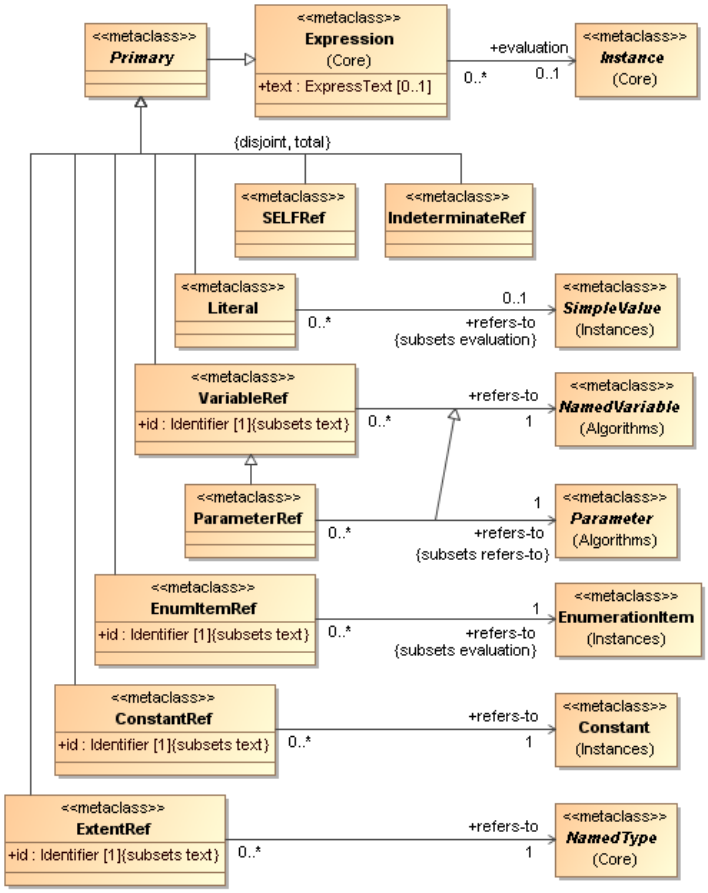

Figure 38 - Primaries

13.3.1 Class: ConstantRef

Definition: a Primary Expression that returns the (current) value of a given Constant. The :id attribute refers to an identifier for a Constant defined in, or interfaced into, the schema.

Note – See 12.7.1 of ISO 10303-11:2004.

Note – A reference to an EXPRESS “Built-in Constant” is considered to be a Literal, not a ConstantRef.

13.3.1.1 Supertypes

Primary

13.3.1.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the Reference.

Multiplicity: 1..1

13.3.1.3 Associations

AssociationEnd: refers-to **To: Instances::Constant**

Definition: represents the Constant referred to by a ConstantRef.

Note – See 12.7.1 of ISO 10303-11:2004.

Multiplicity: 1..1

13.3.1.4 Other Roles

none.

13.3.2 Class: EnumItemRef

Definition: a Primary Expression that returns an EnumerationItem (value)

Note – See 12.7.1 of ISO 10303-11:2004.

13.3.2.1 Supertypes

Primary

13.3.2.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the reference.

Multiplicity: 1..1

13.3.2.3 Associations

AssociationEnd: refers-to **To: Instances::EnumerationItem**

Subsets: Core::Expression:evaluation

Definition: represents the EnumerationItem value referred to by the EnumItemRef. This relationship specializes Expression:evaluation.

Multiplicity: 1..1

13.3.2.4 Other Roles

none.

13.3.3 Class: ExtentRef

Definition: a Primary Expression denoting the extent of a NamedType (almost always an entity data type), that is, the set of instances of that data type that appear in the population. This type of Primary is only permitted in an Expression that states a Rule.

Note – See 9.6 of ISO 10303-11:2004.

13.3.3.1 Supertypes

Primary

13.3.3.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the reference.

Multiplicity: 1..1

13.3.3.3 Associations

AssociationEnd: refers-to **To: Core::NamedType**

Definition: represents the relationship between the Extent Reference and the NamedType to which the :id value refers. The value returned is the Extent of that NamedType within the (current) Population.

Multiplicity: 1..1

13.3.3.4 Other Roles

none.

13.3.4 Class: IndeterminateRef

Definition: a Primary Expression consisting of the ‘symbol ”?”’, which always evaluates to the INDETERMINATE value (see 9.2.9).

Note – See 14.2 of ISO 10303-11:2004.

Although the Indeterminate (“?”) symbol is described as a built-in constant in ISO 10303-11, it is treated here as a distinct kind of Primary, because it refers-to (evaluates-to) an instance that is not a value of any DataType.

13.3.4.1 Supertypes

Primary

13.3.4.2 Attributes

none.

13.3.4.3 14164:Associations

none.

13.3.4.4 Other Roles

none.

13.3.5 Class: Literal

Definition: a Primary Expression consisting of a symbol that denotes a specific value of a SimpleType. The :text attribute of Expression is the representation of the value.

Note – See 7.5 of ISO 10303-11:2004.

Note – References to the built-in constants - E, PI, TRUE, FALSE, UNKNOWN - are considered to be Literals whose `:text` is the keyword.

13.3.5.1 Supertypes

Primary

13.3.5.2 Attributes

none.

13.3.5.3 Associations

AssociationEnd: refers-to **To: Instances::SimpleValue**

subsets: Core::Expression:evaluation

Definition: represents the SimpleValue value referred to by the Literal. This relationship specializes Expression:evaluation.

Multiplicity: 0..1

Note – Although every Literal refers to exactly one SimpleValue, it is not usually necessary to instantiate either the Simple-Value or the relationship.

13.3.5.4 Other Roles

none.

13.3.6 Class: ParameterRef

Definition: a Primary Expression that returns the current value associated with a given Parameter.

A ParameterRef is only permitted within the body of an Algorithm.

For an InParameter, the associated value is the current value of the InParameter..
For a VarParameter, the associated value is the current value in the referent of the VarParameter.

A ParameterRef is a subclass of VariableRef, because every Parameter is a NamedVariable, and a ParameterRef is a reference to the value of the Parameter seen as a variable in the body of the Algorithm.

Note – See 12.7.1 of ISO 10303-11:2004.

13.3.6.1 Supertypes

VariableRef

13.3.6.2 Attributes

none.

13.3.6.3 Associations

AssociationEnd: refers-to **To: Algorithms::Parameter**

Subsets: VariableRef:refers-to

Definition: the formal Parameter to which the ParameterRef refers. If the formal Parameter is an InParameter, the ParameterRef refers to its current value. If the formal Parameter is a VarParameter, the ParameterRef refers to the current value of its referent.

Note – See 12.7.1 of ISO 10303-11:2004.

Multiplicity: 1..1

13.3.6.4 Other Roles

none.

13.3.7 Class: SELFRef

Definition: A Primary Expression consisting of the symbol SELF. It refers to the value of each instance (in any Population) of the data type being defined by the declaration in which it appears. SELF is only a valid Symbol in a DomainRule.

Note – See clause 14.5 of ISO 10303-11:2004.

13.3.7.1 Supertypes

Primary

13.3.7.2 Attributes

none.

13.3.7.3 Associations

none.

13.3.7.4 Other Roles

none.

13.3.8 Class: VariableRef

Definition: a Primary Expression that returns the value currently associated with a given NamedVariable. NamedVariables include LocalVariables, QueryVariables, ControlVariables, and AliasVariables. They also include Parameters and FunctionResults seen as variables within the body of the Algorithm.

A VariableRef that refers-to a QueryVariable may occur anywhere within expressions in the owning Query.

A VariableRef that refers-to a ControlVariable may occur anywhere within the RepeatStatement that defines the ControlVariable.

A VariableRef that refers-to an AliasVariable may occur anywhere within the AliasStatement.

A VariableRef that refers-to a LocalVariable may occur anywhere within the AlgorithmScope in which it is defined:

- for a GlobalRule, it may occur anywhere within the body of the GlobalRule, or within the NamedRules contained in the GlobalRule;
- for an Algorithm, it may occur within the body of an Algorithm or within initial-value expressions for other LocalVariables.

A VariableRef that refers to a Parameter may occur anywhere within the body of the Algorithm, or within initial-value expressions for LocalVariables.

A VariableRef that refers to a FunctionResult may occur anywhere within the body of the Algorithm,

The value associated with a VariableRef that refers to aVARVariable (an AliasVariable or a VARParameter) is the current value in the referent of the VARVariable.

The value associated with any other VariableRef is the current value in the Variable to which the VariableRef refers.

Note – See 12.7.1 of ISO 10303-11:2004.

13.3.8.1 Supertypes

Primary

13.3.8.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the reference.

Multiplicity: 1..1

13.3.8.3 Associations

AssociationEnd: refers-to **To: Algorithms::NamedVariable**

Definition: represents the relationship between the VariableReference and the local Variable to which it refers.

Multiplicity: 1..1

13.3.8.4 Other Roles

none.

13.4 Indexing

This section describes the EXPRESS operations that select values that are part of Instances. Indexing operations – aggregate indexing, string indexing and binary indexing – extract component values by their numbered positions in the Instance. These concepts are shown in Figure 39.

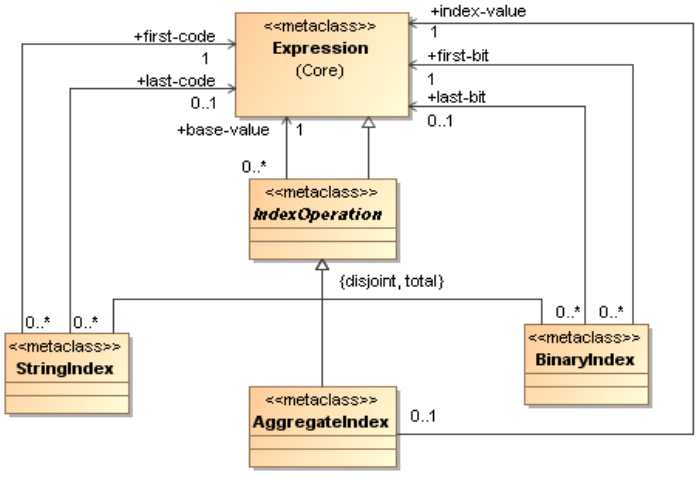

Figure 39 - Indexing Operations

13.4.1 Class: AggregateIndex

Definition: an IndexOperation that returns the value of a specified member of a given AggregateValue. .base-value evaluates to the AggregateValue. .index-value evaluates to the "position" of the member to be extracted. The interpretation of the .index-value depends on the kind of AggregateValue (Indexed, Ordered, Unordered).

Note – See 12.6.1 of ISO 10303-11:2004.

13.4.1.1 Supertypes

IndexOperation

13.4.1.2 Attributes

none.

13.4.1.3 Associations

AssociationEnd: index-value **To: Core::Expression**

Definition: represents the (Integer) index value designating the member whose value is to be extracted. The interpretation of the index value depends on the kind of AggregateValue.

Note – See 12.6.1 of ISO 10303-11:2004.

Multiplicity: 1..1

13.4.1.4 Other Roles

none.

13.4.2 Class: BinaryIndex

Definition: An IndexOperation that returns a substring of one or more bits from a BINARY value. .base-value is the BINARY value. .first-bit designates the position of the first bit to be extracted. .last-bit designates the position of the last bit to be extracted. .last-bit has no value if only one bit is to be extracted.

Note – See clause 12.3.1. of ISO 10303-11:2004.

13.4.2.1 Supertypes

IndexOperation

13.4.2.2 Attributes

none.

13.4.2.3 Associations

AssociationEnd: first-bit **To: Core::Expression**

Definition: represents the (positive integer) value that designates the position of the first bit to be extracted.

Multiplicity: 1..1

AssociationEnd: last-bit **To: Core::Expression**

Definition: represents the (positive integer) value that designates the position of the last bit to be extracted. .last-bit has no value if only one bit is to be extracted.

Multiplicity: 0..1

13.4.2.4 Other Roles

none.

13.4.3 Class: StringIndex

Definition: An IndexOperation that returns a substring of one or more characters (codes) from a STRING value. .base-value is the STRING value. .first-code designates the position of the first character (code) to be extracted. .last-code designates the position of the last character (code) to be extracted. .last-code has no value if only one character is to be extracted.

Note – See clause 12.5.1. of ISO 10303-11:2004.

13.4.3.1 Supertypes

IndexOperation

13.4.3.2 Attributes

none.

13.4.3.3 Associations

AssociationEnd: first-code **To: Core::Expression**

Definition: represents the (positive integer) value that designates the position of the first character (code) to be extracted.

Multiplicity: 1..1

AssociationEnd: last-code **To: Core::Expression**

Definition: represents the (positive integer) value that designates the position of the last character (code) to be extracted. .last-code has no value if only one character (code) is to be extracted.

Multiplicity: 0..1

13.4.3.4 Other Roles

none.

13.5 Selection

This section describes the EXPRESS operations that select values that are related to EntityInstances, or are components of PartialEntityValues. Selector operations extract values related to entity instances by the name of the relationship – attributes, implicit inverse attributes (UsedIn), and attribute-groups. In a similar way, they can be used to extract the values of attributes and attribute-groups from PartialEntityValues. The Selector operations are shown in Figure 40.

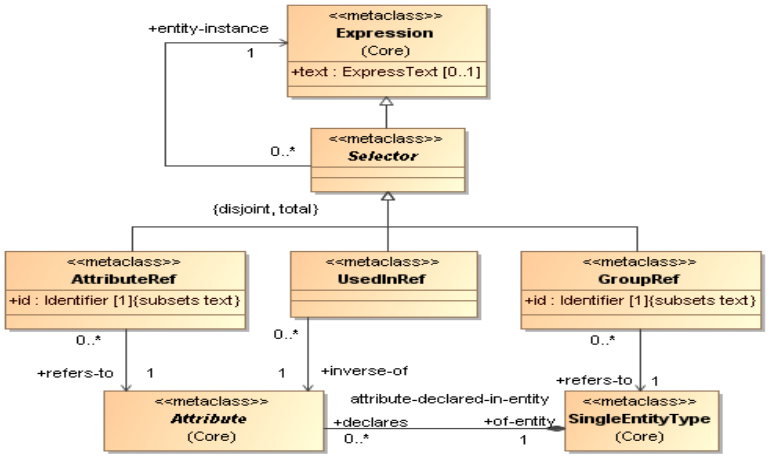

Figure 40 - Attribute and Attribute-Group Selectors

13.5.1 Class: AttributeRef

Definition: a Selector expression that returns the value of a given Attribute of a given entity instance.

Note – See 12.7.3 of ISO 10303-11:2004.

13.5.1.1 Supertypes

Selector

13.5.1.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the reference.

Multiplicity: 1..1

13.5.1.3 Associations

AssociationEnd: refers-to **To: Core::Attribute**

Definition: represents the relationship between the AttributeReference and the Attribute to which it refers.

Multiplicity: 1..1

13.5.1.4 Other Roles

none.

13.5.2 Class: GroupRef

Definition: a Selector that returns a PartialEntityValue consisting of the values of the Attributes of a given entity instance that constitute a given SingleEntityType.

Note – See 12.7.4 of ISO 10303-11:2004.

13.5.2.1 Supertypes

Selector

13.5.2.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier that is the content of the reference.

Multiplicity: 1..1

13.5.2.3 Associations

AssociationEnd: refers-to **To: Core::SingleEntityType**

Definition: represents the relationship between the GroupReference and the SingleEntityType (group of Attributes) to which it refers.

Multiplicity: 1..1

13.5.2.4 Other Roles

none.

13.5.3 Class: UsedInRef

Definition: a Selector expression that returns the Set of EntityInstances for which the given entity instance is in the range of the specified Attribute. In effect, it returns the value of the corresponding inverse attribute for the given entity instance.

Note – See clause 15.26 of ISO 10303-11:2004.

13.5.3.1 Supertypes

Selector

13.5.3.2 Attributes

none.

13.5.3.3 Associations

AssociationEnd: inverse-of **To: Core::Attribute**

Definition: represents the relationship between the UsedIn Reference and the Attribute designated by the :id value. The UsedIn Reference effectively produces the “inverse” of this Attribute.

Multiplicity: 1..1

13.5.3.4 Other Roles

none.

13.6 Operations

This section describes the Expressions that are conceptually “operations” with one operand (UnaryOperation) or two operands (BinaryOperation).

The EXPRESS syntax for Operations takes several forms. Some of the operations are denoted by infix or prefix operation symbols, such as “+” or “NOT.” Others are denoted by “built-in functions” that take one or two arguments that are the operands. In this metamodel, they are all treated as Operations. Each built-in function is represented by a corresponding BinaryOperator or UnaryOperator. There is not a one-to-one correspondence between Operations and EXPRESS operation symbols and built-in functions, because some of the symbols are “overloaded,” in that they denote different operations for operands of different data types.

This section also includes the Coercion operation, which is a special case. It has only one operand, but it also has a “meta-operand” – the data type to which the operand is to be logically or physically converted. Each EXPRESS data type, including all user-defined types, implicitly defines a Coercion operation whose target is that data type. And in that sense, the data type simply distinguishes one coercion operations from another. There is no explicit EXPRESS syntax for Coercion operations; they are inserted as part of the semantic interpretation of Expressions, when it is necessary to treat a literal or result as representing a value of a different datatype.

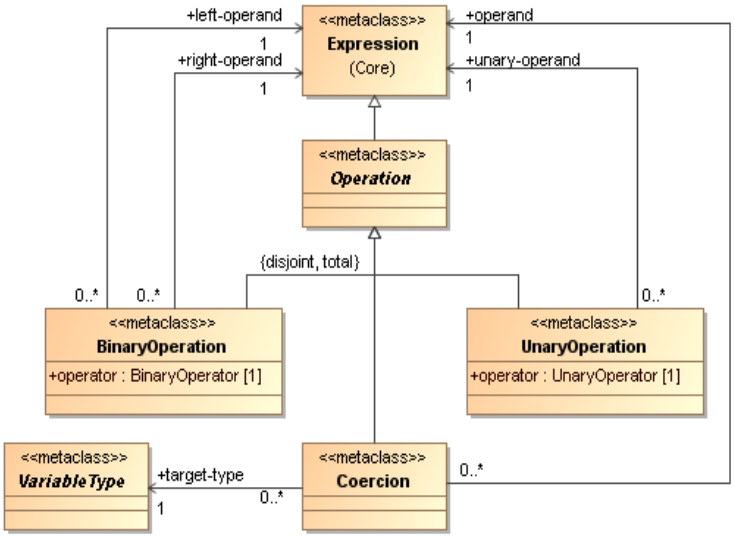

Figure 41 - Operations and Built-in Functions

13.6.1 Class: BinaryOperation

Definition: an Operation representing the result of a well-defined mathematical operation or character manipulation on two Expression operands, which are distinguished. An instance of BinaryOperation represents a usage of a value of BinaryOperator with a specific left and right operand.

Note – See clause 12 of ISO 10303-11:2004.

13.6.1.1 Supertypes

Operation

13.6.1.2 Attributes

Attribute: operator **To: BinaryOperator**

Definition: Represents the conceptual operation that is actually being performed by the BinaryOperation.

Note – See ISO 10303-11.2:2004, clause 12.

Multiplicity: 1..1

13.6.1.3 Associations

AssociationEnd: left-operand **To: Core::Expression**

Definition: represents the operand Expression that produces one input to a BinaryOperation, distinguished (if needed) as the “left” operand in the definition of the operation.

Note – See clause 12 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: right-operand **To: Core::Expression**

Definition: represents the operand Expression that produces one input to a BinaryOperation, distinguished (if needed) as the “right” operand in the definition of the operation.

Note – See clause 12 of ISO 10303-11:2004.

Multiplicity: 1..1

13.6.1.4 Other Roles

none.

13.6.2 Datatype: BinaryOperator

Stereotypes: `enumeration`

Definition: Conceptual EXPRESS language element representing the interpretation of a binary operation symbol in the context of the operand datatypes. Instances of this class are distinct operations, such as number-addition, set-union, string-compare-equal, etc. Some BinaryOperators are denoted by “built-in functions” in EXPRESS syntax.

Note – See ISO 10303-11.2:2004 clause 12 and some elements of clause 15.

13.6.2.1 Supertypes

none.

13.6.2.2 Values

Value: AND

Definition: Returns true if both operands are true, unknown if both are unknown, and false if either is false.

Value: Add

Definition: Returns the arithmetic sum of two NUMBER operands.

Value: BagAdd

Definition: Returns the BagValue resulting from adding one to the count of occurrences of the value of the second operand in the first operand, which must be a BagValue.

Value: BagRemove

Definition: Returns the BagValue resulting from subtracting one from the count of occurrences of the value of the second operand in the first operand, which must be a BagValue. If the first operand contains no occurrences of the value of the second operand, returns the value of the first operand.

Value: BagUnion

Definition: For two BAG operands with a common member type, returns the BAG value in which the number of occurrences of each value of the member type is the sum of the number of its occurrences in the two operands.

Value: BinaryAppend

Definition: Returns the BinaryValue whose bits are the bits of the value of the first operand, which must be a BinaryValue, in that order, followed by the bits of the value of the second operand, which must be a BinaryValue, in that order.

Value: DIV

Definition: For two INTEGER operands, returns the integral part of the quotient of dividing the value of the first by the value of the second.

Value: Difference

Definition: For two SET operands with a common member type, returns the SET value containing all members of the first operand except for those that are also members of the second operand. For two BAG operands with a common member type, returns the BAG value in which the number of occurrences of each value of the member type is the number of its occurrences in the first operand minus the number of its occurrences in the second operand, but not less than zero.

Value: Divide

Definition: For two NUMBER operands, returns the quotient of dividing the value of the first by the value of the second.

Value: EntityConstructor

Definition: For two operands that are PartialEntityValues, returns the PartialEntityValue that contains all of the SingleEntityValues that were present in either operand. This operation is referred to in EXPRESS as the “complex entity constructor” (||).

Note – See ISO 10303-11:2004 clause 12.10

Value: EntityValueEqual

Definition: If both operands are of a common data type and that data type is an entity data type, returns false if the value of any attribute of the first operand is NotEqual to (or EntityValueNotEqual to) the value of that attribute of the second operand, else true. If both operands are of a common data type and that data type is an aggregation type whose members are entity instances, returns false if the operands are of different sizes, or if for any of the corresponding members of the two operands, the value of any attribute of the member of the first operand is NotEqual to (or EntityValueNotEqual to) the value of that attribute of the member of the second operand, else true. If the common data type is anything else, this operator is equivalent to Equal.

Value: EntityValueNotEqual

Definition: If both operands are of a common data type and that data type is an entity data type, returns true if the value of any attribute of the first operand is NotEqual to (or EntityValueNotEqual to) the value of that attribute of the second operand, else false. If both operands are of a common data type and that data type is an aggregation type whose members are entity instances, returns true if the operands are of different sizes, or if for any of the corresponding members of the

two operands, the value of any attribute of the member of the first operand is NotEqual to (or EntityValueNotEqual to) the value of that attribute of the member of the second operand, else false. If the common data type is anything else, this operator is equivalent to NotEqual.

Value: Equal

Definition: Returns true if both operands are of a common data type and equal in value, as defined for that type, else false. For the definition of “equal in value,” see ISO 10303-11:2004 clause 12.2.1.

Value: Exponent

Definition: For two NUMBER operands, returns the the value of the first raised to the power specified by the value of the second.

Value: Greater

Definition: Returns true if both operands are of a common data type and the value of the first operand is greater than the value of the second operand, as defined for that type, else false. For the definition of “is greater than,” see ISO 10303-11:2004 clause 12.2.1.

Value: IN

Definition: Returns true if the value of the first operand is Equal to the value of any member of the second operand (which must be an AggregateValue); else false. If the first operand is an EntityInstance, “is Equal to” is interpreted as “is InstanceEqual to.”

Value: InstanceEqual

Definition: If both operands are of a common data type and that data type is an entity data type, returns true if both operands refer to the same individual, else false. If both operands are of a common data type and that data type is an aggregation type whose members are entity instances, returns false if the operands are of different sizes, or if any of the corresponding members of the two operands refer to different individuals, else true. If the common data type is anything else, this operator is equivalent to Equal.

Value: InstanceNotEqual

Definition: If both operands are of a common data type and that data type is an entity data type, returns true if the operands refer to distinct individuals, else false. If both operands are of a common data type and that data type is an aggregation type whose members are entity instances, returns true if the operands are of different sizes, or if any of the corresponding members of the two operands refer to different individuals, else false. If the common data type is anything else, this operator is equivalent to NotEqual.

Value: Intersection

Definition: For two SET operands with a common member type, returns the mathematical intersection of the two sets. For two BAG operands with a common member type, returns the BAG value in which the number of occurrences of each value of the member type is the smaller of the number of its occurrences in the two operands.

Value: LIKE

Definition: Returns true if both operands are StringValues and the value of the first operand is a match for the pattern that is the value of the second operand. For the interpretation of the pattern, see ISO 10303-11:2004 clause 12.2.5.

Value: Less

Definition: Returns true if both operands are of a common data type and the value of the first operand is less than the value of the second operand, as defined for that type, else false. For the definition of “is less than,” see ISO 10303-11:2004 clause 12.2.1.

Value: ListAddFirst

Definition: Returns the ListValue whose first member is the value of the second operand and whose subsequent members are the members of the value of the first operand, which must be a ListValue, in that order.

Value: ListAddLast

Definition: Returns the ListValue whose members are the members of the value of the first operand, which must be a ListValue, in that order, followed by the value of the second operand.

Value: ListAppend

Definition: Returns the ListValue whose members are the members of the value of the first operand, which must be a ListValue, in that order, followed by the members of the value of the second operand, which must be a ListValue, in that order.

Value: MOD

Definition: For two INTEGER operands, returns the remainder of dividing the value of the first by the value of the second.

Value: Multiply

Definition: Returns the arithmetic product of two NUMBER operands.

Value: NVL

Definition: If the value of the first operand is Indeterminate (?), returns the value of the second operand; else returns the value of the first operand.

Note – See ISO 10303-11:2004 clause 15.18.

Value: NotEqual

Definition: Returns true if both operands are of a common data type and unequal in value, as defined for that type, else false. For the definition of “equal in value,” see ISO 10303-11:2004 clause 12.2.1.

Value: NotGreater

Definition: Returns true if both operands are of a common data type and the value of the first operand is less than or equal to the value of the second operand, as defined for that type, else false. For the definition of “is less than or equal to,” see ISO 10303-11:2004 clause 12.2.1.

Value: NotLess

Definition: Returns true if both operands are of a common data type and the value of the first operand is greater than or equal to the value of the second operand, as defined for that type, else false. For the definition of “is greater than or equal to,” see ISO 10303-11:2004 clause 12.2.1.

Value: OR

Definition: Returns true if either operand is true, unknown if both are unknown, and false if both are false.

Value: SetAdd

Definition: Returns the SetValue that is the union of the value of the first operand, which must be a SetValue, with the SetValue comprising exactly one member equal (or InstanceEqual) to the value of the second operand.

Value: SetUnion

Definition: For two SET operands with a common member type, returns the mathematical union of the two sets.

Value: StringAppend

Definition: Returns the StringValue whose characters are the characters of the value of the first operand, which must be a StringValue, in that order, followed by the characters of the value of the second operand, which must be a StringValue, in that order.

Value: Subset

Definition: Returns true if every member of the value of the first operand (which must be an AggregateValue) is IN the value of the second operand (which must be an AggregateValue); else false.

Value: Subtract

Definition: For two NUMBER operands, returns the result of subtracting the value of the second from the value of the first.

Value: ValueIn

Definition: Returns true if the value of the first operand is Equal to the value of any member of the second operand (which must be an AggregateValue); else false. If the first operand is an EntityInstance, “is Equal to” is interpreted as “is EntityValueEqual to.”

Note – See ISO 10303-11:2004 clause 15.28.

Value: XOR

Definition: Returns true if one operand is true and one is false, unknown if either is unknown, and false otherwise.

13.6.3 Class: Coercion

Definition: an Operation representing the conversion of the operand to a specific data type (InstantiableType). This operation is implicit in a number of EXPRESS expressions, notably:

- in converting between a defined data type and its fundamental type (on which the operations are defined), and
- in converting an EntityValue to an EntityInstance of the corresponding EntityType.

In most cases, the Coercion does not change the “value” of the operand; rather the Coercion maps the value to the corresponding value of the related data type.

Note – See clause 12 of ISO 10303-11:2004, and the proposed revision to clause 12.10.

13.6.3.1 Supertypes

Operation

13.6.3.2 Attributes

None.

13.6.3.3 Associations

AssociationEnd: operand **To: Core::Expression**

Definition: represents the Expression whose result is to be converted to the target-type by the Coercion operation.

Multiplicity: 1..1

AssociationEnd: target-type **To: Core::VariableType**

Definition: represents the data type to which the operand of the Coercion is to be converted.

Multiplicity: 1..1

13.6.3.4 Other Roles

none.

13.6.4 Class: UnaryOperation

Definition: an Operation representing the result of a well-defined mathematical operation on a single Expression operand. A UnaryOperation models a use of a UnaryOperator with a particular operand.

Note – See clause 12 of ISO 10303-11:2004.

13.6.4.1 Supertypes

Operation

13.6.4.2 Attributes

Attribute: operator **To: UnaryOperator**

Definition: Represents the conceptual operation that is actually being performed by the UnaryOperation.

Note – See ISO 10303-11.2:2004, clause 12.

Multiplicity: 1..1

13.6.4.3 Associations

AssociationEnd: unary-operand **To: Core::Expression**

Definition: represents the operand Expression that produces the input to a UnaryOperation.

Note – See clause 12 of ISO 10303-11:2004.

Multiplicity: 1..1

13.6.4.4 Other Roles

None.

13.6.5 Datatype: UnaryOperator

Stereotypes: `enumeration`

Definition: Conceptual EXPRESS language element representing the interpretation of a unary operation symbol in the context of the operand datatype. Instances of this class are distinct operations, such as numeric-negation, boolean-negation, real-square-root, absolute-value, etc. Some UnaryOperators are denoted by “built-in functions” in EXPRESS syntax.

See ISO 10303-11.2:2004 clause 12 and some elements of clause 15.

13.6.5.1 Supertypes

none.

13.6.5.2 Values

Value: ABS

Definition: For a NUMBER operand, returns the magnitude (absolute value) of the value of the operand.

Value: ACOS

Definition: For a NUMBER operand, returns the mathematical arc cosine of the value of the operand.

Value: ASIN

Definition: For a NUMBER operand, returns the mathematical arcsine of the value of the operand.

Value: ATAN

Definition: For a NUMBER operand, returns the mathematical arctangent of the value of the operand.

Value: BinaryLength

Definition: For an operand that is a BinaryValue, returns the number of bits in the value.

Value: COS

Definition: For a NUMBER operand, returns the mathematical cosine of the value of the operand.

Value: EXISTS

Definition: Returns false if the operand is Indeterminate (?), else true.

Value: EXP

Definition: For a NUMBER operand, returns the mathematical exponential function of the value of the operand.

Value: HiBound

Definition: For an operand whose data type is an aggregation type, returns the declared upper-bound value for the size of the values, or for an ARRAY, the declared maximum index-value.

Value: HiIndex

Definition: For an operand that is an AggregateValue, returns the largest valid index-value for the value.

Value: Identity

Definition: Returns the value of the operand.

Value: LOG

Definition: For a NUMBER operand, returns the Napierian logarithm of the value of the operand.

Value: LOG10

Definition: For a NUMBER operand, returns the logarithm to the base 10 of the value of the operand, which for an INTEGER value is the number of decimal digit characters required to represent it.

Value: LOG2

Definition: For a NUMBER operand, returns the logarithm to the base 2 of the value of the operand, which for an INTEGER value is the number of bits required to represent it.

Value: LoBound

Definition: For an operand whose data type is an aggregation type, returns the declared lower-bound value for the size of the values, or for an ARRAY, the declared minimum index-value.

Value: LoIndex

Definition: For an operand that is an AggregateValue, returns the smallest valid index-value for the value.

Value: NOT

Definition: For an operand that is a LogicalValue, returns true if the value is false, unknown if the value is unknown, and false if the value is true.

Value: Negate

Definition: For a NUMBER operand, returns the additive inverse of the value of the operand.

Value: ODD

Definition: For an operand that is an INTEGERValue, returns false if the value is exactly divisible by 2 and true otherwise.

Value: RolesOf

Definition: For an EntityInstance operand, returns a set of RoleName values representing all the distinct Attributes (RangeRoles) which the operand plays in the Population.

Value: SIN

Definition: For a NUMBER operand, returns the mathematical sine of the value of the operand.

Value: SQRT

Definition: For a NUMBER operand, returns the mathematical square root of the value of the operand, or Indeterminate if it is negative.

Value: SizeOf

Definition: For an operand that is an AggregateValue, returns the number of members in the value.

Value: StringLength

Definition: For an operand that is a StringValue, returns the number of characters in the value.

Value: TAN

Definition: For a NUMBER operand, returns the mathematical tangent of the value of the operand.

Value: TypeOf

Definition: Returns a Set of TypeName values representing the data types of which the operand is an instance.

Value: VALUE

Definition: For a STRING operand, returns the NUMBER value resulting from interpreting the operand as the representation of a numeric value, or Indeterminate, if no such interpretation can be made.

Value: ValueUnique

Definition: For an operand that is an AggregateValue, returns true if no two members of the operand are Equal or EntityValueEqual.

13.7 Function Calls

This section describes the Expressions that represent invocations of schema-defined FUNCTIONs, each of which returns a FunctionResult that is the evaluation of the Expression.

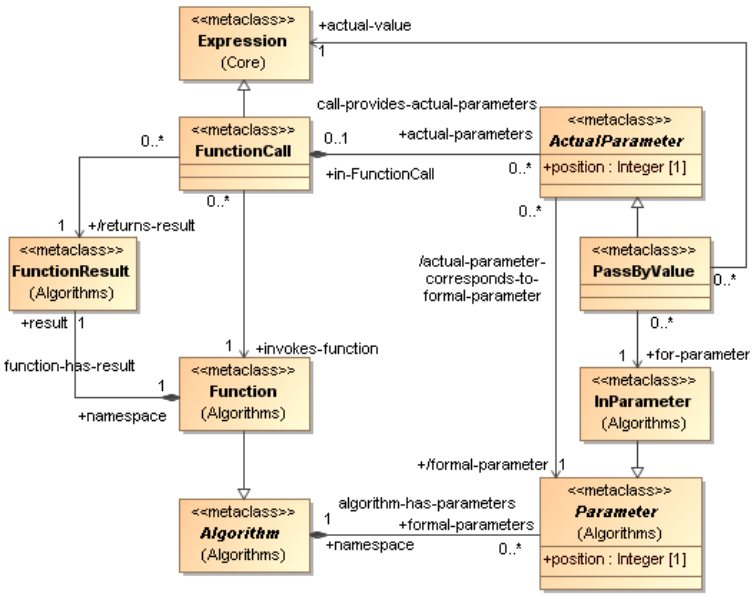

Figure 42 - Function Calls

13.7.1 Class: ActualParameter

Definition: represents the substitution of the actual parameter instance for the formal parameter and, where required, the substitution of the data type of the actual parameter for the GeneralizedType of the formal parameter and any derivatives.

ActualParameter is an abstraction of two different parameter-passing mechanisms: PassByValue and PassByReference. When the corresponding formal Parameter is an InParameter, the ActualParameter shall be a PassByValue. When the corresponding formal Parameter is a VARParameter, the ActualParameter shall be a PassByReference.

In a FunctionCall, the corresponding formal parameter is always an InParameter; a ProcedureCall can have formal parameters of either kind.

Note – PassByValue is described below. PassByReference is defined in the Statements package (a separate compliance point), because it is only used in Procedure Call statements.

Note – See 12.8 of ISO 10303-11:2004.

Properties: abstract

13.7.1.1 Supertypes

none.

13.7.1.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: represents the position in which the ActualParameter occurs in the sequence associated with the FunctionCall (used to associate the ActualParameter with a formal parameter).

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

13.7.1.3 Associations

AssociationEnd: formal-parameter **To: Algorithms::Parameter**

Definition: represents the formal parameter to which the ~~ParameterBinding applies~~ ActualParameter corresponds.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: derived union of `PassByValue:for-parameter` and `PassByReference:for-parameter`.

AssociationEnd: in-FunctionCall **To: FunctionCall**

via: call-provides-actual-parameters

Definition: the FunctionCall, if any, that contains the ActualParameter.

Multiplicity: 0..1

AssociationEnd: in-ProcedureCall **To: Statements::ProcedureCall**

via: Statements::procedure-call-provides-actual-parameters

Definition: the ProcedureCall, if any, in which the ActualParameter appears.

Multiplicity: 0..1

13.7.1.4 Other Roles

none.

13.7.1.5 Rules

Constraint

```
exists(self->in-FunctionCall) xor exists(self->in-ProcedureCall);
```

A given ActualParameter must occur in either a FunctionCall or a ProcedureCall.

13.7.2 Class: FunctionCall

Definition: an Expression that represents the instance resulting from the invocation of a Function with zero or more Expression operands called “actual parameters.”

Note – See 12.8 of ISO 10303-11:2004.

13.7.2.1 Supertypes

Core::Expression

13.7.2.2 Attributes

none.

13.7.2.3 Associations

AssociationEnd: actual-parameters **To: ActualParameter**

via: call-provides-actual-parameters

Definition: represents the relationship between a FunctionCall and the specifications for the values of its actual parameters.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: invokes-function **To: Algorithms::Function**

Definition: represents the relationship between the FunctionCall and the formal definition of the Function invoked.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: returns-result **To: Algorithms::FunctionResult**

Definition: represents the relationship between the FunctionCall and the formal definition of the FunctionResult, which describes the instance that results from the FunctionCall.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

Properties: derived

TaggedValues

```
derivation = self->invokes-function->result
```

13.7.2.4 Other Roles

none.

13.7.3 PassByValue

Definition: An ActualParameter that is passed "by value." At the time of Algorithm invocation, the `actual-value` Expression is evaluated and the resulting value is assigned to the InParameter – the local Variable within the invocation that corresponds to the formal parameter.

13.7.3.1 Supertypes

ActualParameter

13.7.3.2 Attributes

none.

13.7.3.3 Associations

AssociationEnd: actual-value **To: Core::Expression**

Definition: the Expression that specifies the value to be passed to the InParameter. This is the Expression that is syntactically the `actual_parameter` when the corresponding formal parameter is an InParameter.

The `actual-value` shall evaluate to either an instance of an InstantiableType or Indeterminate.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: for-parameter **To: Algorithms::InParameter**

Subsets: Expressions::ActualParameter:formal-parameter

Definition: the formal parameter to which the actual value is passed.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

13.7.3.4 Other Roles

none.

13.7.4 Association: call-provides-actual-parameters

Definition: represents the relationship between a FunctionCall and the specifications for the values of its actual parameters.

13.7.4.1 Association Ends

AssociationEnd: actual-parameters **To: ActualParameter**

Definition: represents the relationship between a FunctionCall and the specifications for the values of its actual parameters.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: in-FunctionCall **To: FunctionCall**

Definition: the FunctionCall, if any, that contains the ActualParameter.

Multiplicity: 0..1

13.8 Query Expressions

This section describes the QueryExpression, which models invocations of the EXPRESS built-in QUERY function, specified in section 12.6.7 of ISO 10303-11. The concepts are depicted in Figure 43.

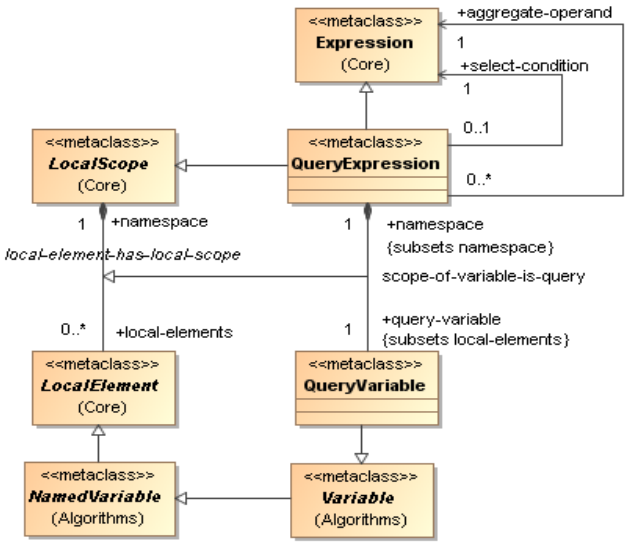

Figure 43 - Query Expressions

13.8.1 Class: QueryExpression

Definition: an Expression representing the (aggregate) instance that results from extracting from the value of the aggregate-operand (an Expression yielding an aggregate value) the corresponding collection of member instances that satisfy a given select-condition. Every QueryExpression is also the LocalScope for the QueryVariable that designates members of the aggregate value in the select-condition.

Note – See 12.6.7 of ISO 10303-11:2004.

13.8.1.1 Supertypes

Core::Expression, Core::LocalScope

13.8.1.2 Attributes

none.

13.8.1.3 Associations

AssociationEnd: aggregate-operand **To: Core::Expression**

Definition: represents the operand Expression whose result is the aggregate value from which members will be extracted by the Query operation.

Note – See 12.6.7 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: query-variable **To: QueryVariable**

via: scope-of-variable-is-query

Subsets: Core::LocalScope:local-elements

Definition: the QueryVariable associated with the QueryExpression. The QueryVariable ranges over the member elements of the aggregate-operand.

Multiplicity: 1..1

Properties: composite

AssociationEnd: select-condition **To: Core::Expression**

Definition: represents the relationship between a Query expression and the Logical Expression that defines admissibility of members in the Query result. This Expression is treated as a kind of "function definition" having a single Parameter which is the Query variable. The .select-condition "function" is invoked once for each member value of the .aggregate-value.

Note – See Clause 12.6.7 of ISO 10303-11:2004. The Expression that formulates the select-condition is owned by the QueryExpression. It is not treated as reusable.

Multiplicity: 1..1

13.8.1.4 Other Roles

none.

13.8.2 Class: QueryVariable

Definition: a Variable that ranges over the member elements of the aggregate-operand in evaluating the QueryExpression. The scope of a QueryVariable is the QueryExpression, that is, all references to it occur in the select-condition of the QueryExpression. The data-type of a QueryVariable is implicitly the data type of the member-element of the aggregate operand.

Note – See 12.6.7 of ISO 10303-11:2004.

Note – Although QueryVariable is modeled as a subclass of Variable, it is syntactically impossible for a QueryVariable to be the referent of a VariableCell.

13.8.2.1 Supertypes

Algorithms::Variable

13.8.2.2 Attributes

none.

13.8.2.3 Associations

AssociationEnd: namespace **To: QueryExpression**

via: scope-of-variable-is-query

Subsets: Core::LocalElement:namespace

Definition: the QueryExpression in which the QueryVariable is defined.

Multiplicity: 1..1

13.8.2.4 Other Roles

none.

13.8.3 Association: scope-of-variable-is-query

Definition: represents the (1-to-1) relationship between the QueryVariable and the QueryExpression in which it is defined.

13.8.3.1 Supertypes

Core::local-element-has-local-scope

13.8.3.2 Association Ends

AssociationEnd: namespace **To: QueryExpression**

Definition: the QueryExpression in which the QueryVariable is defined.

Multiplicity: 1..1

AssociationEnd: query-variable **To: QueryVariable**

Definition: the QueryVariable associated with the QueryExpression. The QueryVariable ranges over the member elements of the aggregate-operand.

Multiplicity: 1..1

Properties: composite

13.9 Aggregate Initializers

This section describes the EXPRESS operations that construct AggregateValues from component values.

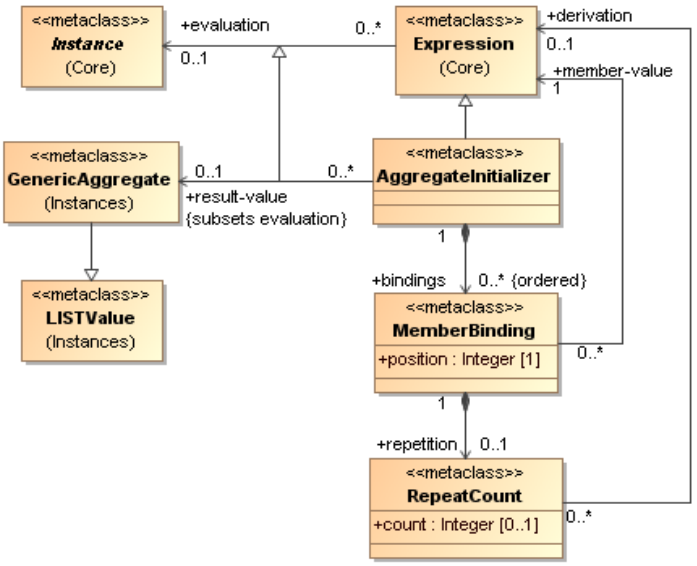

Figure 44 - Aggregate Initializers

13.9.1 Class: AggregateInitializer

Definition: represents the EXPRESS “aggregate initializer.” It produces a value of type AGGREGATE OF GENERIC, by binding a sequence of member values to positions in the generic aggregate value.

Note – See 12.9 of ISO 10303-11:2004.

13.9.1.1 Supertypes

Core::Expression

13.9.1.2 Attributes

None.

13.9.1.3 Associations

AssociationEnd: bindings **To: MemberBinding**

Definition: represents the relationship between the AggregateInitializer and the set of MemberBindings it comprises.

Note – See 12.9 of ISO 10303-11:2004.

Multiplicity: 0..* ~~un~~ordered

Properties: composite

AssociationEnd: result-value **To: Instances::GenericAggregate**

subsets: Core::Expression:evaluation

Definition: represents the aggregate value that results from the aggregate initializer. This is a refinement of Expression:evaluation.

If the AggregateInitializer expression can be evaluated without regard to any actual population (“compile time”), this value shall be present, but not otherwise.

Note – See 12.9 of ISO 10303-11:2004.

Multiplicity: 0..1

13.9.1.4 Other Roles

none.

13.9.2 Class: MemberBinding

Definition: represents the placement of a member value in one or more positions (ListMembers) in the GenericAggregate value resulting from the aggregate initializer.

If the member binding has no repetition count, the MemberBinding associates the .member-value with one ListMember in the GenericAggregate. If the member value has a repetition count, the MemberBinding associates the .member-value with one or more consecutive ListMembers in the GenericAggregate. The member-values are assigned to ListMembers in the order of the MemberBindings. The `:position` of the MemberBinding conveys the ordering of the MemberBindings (but not necessarily the position of the corresponding ListMembers).

Note – The MemberBinding may have a repetition count that depends on values in the population or the actual parameters of an Algorithm invocation, with the consequence that the relationship between the MemberBinding and ListMembers can only be determined when the AggregateInitializer is evaluated.

Note – See 12.9 of ISO 10303-11:2004.

13.9.2.1 Supertypes

none.

13.9.2.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: Represents the ordinal position of the MemberBinding specification in the AggregateInitializer.

Note – When no MemberBinding in the AggregateInitializer has a `.repetition` value, the MemberBinding:position will be the position of the member-value in the resulting GenericAggregate. Otherwise, the relationship between the positions will depend on the `.repetition` values.

Multiplicity: 1..1

13.9.2.3 Associations

AssociationEnd: member-value **To: Core::Expression**

Definition: represents the member value to be assigned to the MemberBinding position in the aggregate value, as the result of the Expression.

Note – See 12.9 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: repetition **To: RepeatCount**

Definition: represents the relationship between the MemberBinding and an associated RepeatCount, if any. If the repetition count for the .member-value is implicitly 1, or explicitly a literal “1,” this relationship shall not appear. In all other cases, this relationship shall appear.

Multiplicity: 0..1

Properties: composite

13.9.2.4 Other Roles

From: AggregateInitializer as bindings

13.9.3 Class: RepeatCount

Definition: A specification for repeating a given initial value into n consecutive ListMember slots, where n is the .count value. The repetition value is specified by the .derivation expression. If that expression is, or evaluates to, a constant (without regard to a Population), the value of .count is that constant.

Note – See 12.9 of ISO 10303-11:2004.

13.9.3.1 Supertypes

none.

13.9.3.2 Attributes

Attribute: count **To: (none)**

Definition: The number of actual ListMembers that are to be filled with the member-value. If the .derivation expression evaluates to a constant, without regard to population, .count has a value; otherwise not.

Multiplicity: 0..1

13.9.3.3 Associations

AssociationEnd: derivation **To: Core::Expression**

Definition: represents the relationship between the RepeatCount and the Expression that denotes the value of the RepeatCount. This relationship shall be present whenever the specification for the RepeatCount is not an integer literal.

Multiplicity: 0..1

13.9.3.4 Other Roles

From: MemberBinding as repetition

13.10 Partial Entity Constructors

This section describes the EXPRESS operations that construct PartialEntityValues from component values.

Note – The so-called “entity constructor” is a binary operation (See 13.6.2 Value: EntityConstructor) that produces PartialEntityValues from other PartialEntityValues. The actual operation that produces entity instances is a special case of Coercion (see 13.6.3).

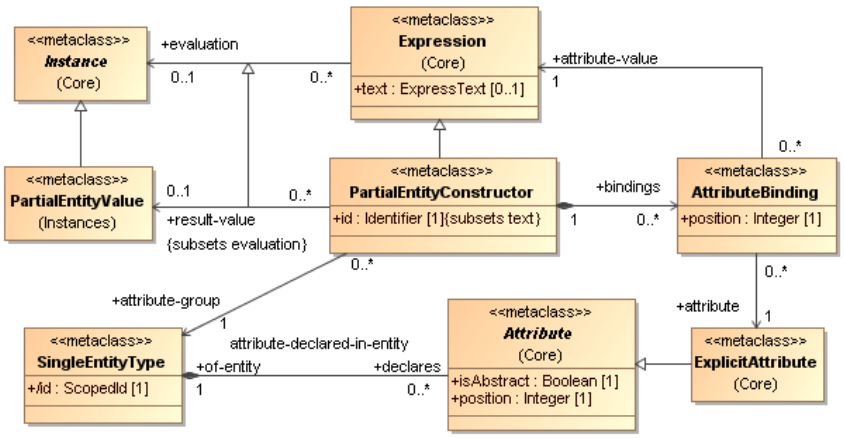

Figure 45 - Partial Entity Value Constructors

13.10.1 Class: AttributeBinding

Definition: represents the assignment of a specific value to one Attribute in the group that comprises the PartialEntityType.

Note – See 9.2.6 of ISO 10303-11:2004.

13.10.1.1 Supertypes

none.

13.10.1.2 Attributes

Attribute: position **To: MOF::Integer**

Definition: represents the position of the AttributeBinding in the constructor (and thus the association with the explicit attribute).

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 1..1

13.10.1.3 Associations

AssociationEnd: attribute **To: Core::ExplicitAttribute**

Definition: represents the explicit attribute to which the AttributeBinding assigns a value. Position is used to identify the attribute.

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: attribute-value **To: Core::Expression**

Definition: represents the value to be assigned to the explicit attribute by the AttributeBinding, as the result of the Expression.

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 1..1

13.10.1.4 Other Roles

From: PartialEntityConstructor as bindings

13.10.2 Class: PartialEntityConstructor

Definition: represents the EXPRESS “partial entity constructor” named for a “single entity data type.” It takes one actual parameter (AttributeBinding) for each ExplicitAttribute in the group of Attributes identified by the SingleEntityType, and binds the values to the ExplicitAttributes in order of their occurrence in the entity_declaration. The result is a PartialEntityValue of the partial entity data type that consists of exactly that one single entity data type.

Note – See 9.2.6 of ISO 10303-11:2004 (revised by TC#1).

13.10.2.1 Supertypes

Core::Expression

13.10.2.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: Core::Expression:text

Definition: Represents the identifier for the PartialEntityConstructor, which is the identifier for the SingleEntityType to which it refers.

Multiplicity: 1..1

13.10.2.3 Associations

AssociationEnd: attribute-group **To: Core::SingleEntityType**

Definition: represents the relationship between the PartialEntityConstructor and the SingleEntityType that defines it, i.e., the list of explicit attributes..

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: bindings **To: AttributeBinding**

Definition: represents the relationship between the PartialEntityConstructor and the set of AttributeBindings it comprises.

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: result-value **To: Instances::PartialEntityValue**

Subsets: Core::Expression:evaluation

Definition: represents the instance that results from the partial entity constructor. This is a refinement of Expression:evaluation.

If the expression can be evaluated without regard to any actual population (“compile time”), this value shall be present, but not otherwise.

Note – See 9.2.6 of ISO 10303-11:2004.

Multiplicity: 0..1

13.10.2.4 Other Roles

none.

14 Package : Statements

The Statements Package contains the detailed modeling concepts for the Statements in the EXPRESS language. The basic Statement model in the Algorithms Package is permitted to be a syntactic string. This package provides the elements that support the operational semantics of each kind of Statement.

The Statements Package depends on the Expressions Package. It is a requirement for the Statements compliance point that a complete semantic model of Expressions be supported.

14.1 Dependencies

Dependency on Package: Core

Stereotypes: `import`

The Statements Package depends on the Core Package for the basic Expression concept and for the LocalScope and LocalElement concepts.

Dependency on Package: Algorithms

Stereotypes: `import`

The Statements Package depends on the Algorithms Package for the basic Statement concept, the Variable concept, and the Procedure concept.

Dependency on Package: Expressions

Stereotypes: `import`

The Statements Package depends on the Expression Package for ActualParameter, and in most implementations, for the detailed semantic models of Expressions.

14.2 Overview of Statements

This section provides the overview of all of the EXPRESS Statement types. They are depicted in Figure 46.

The concept StatementBlock and ControlStatement are described in detail in this section. Each of the other statement types is described in its own section.

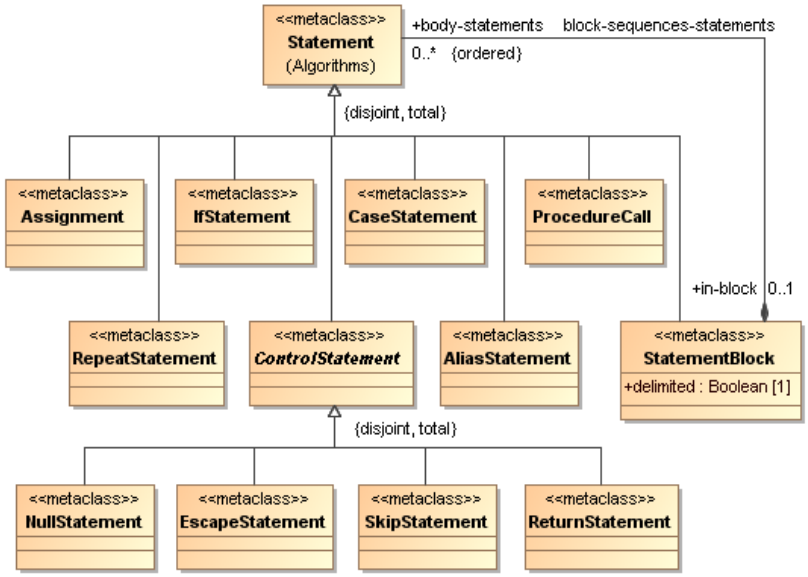

Figure 46- Statements

14.2.1 Class: Algorithms::Statement

Definition: An EXPRESS Statement, a directive to perform a certain set of operations.

Note – See Clause 13 of ISO 10303-11:2004.

Note – Even though Statement is technically an abstract classifier, it is represented by direct instances with text representations when the Statements compliance point is not supported.

Note – The class Statement, and all its properties, is specified in the Algorithms Package, which provides the primary use of Statements. This entry serves only to define the Statement class in context and provide a link to its specification in 10.2.7.

14.2.2 Class: ControlStatement

Definition: an abstract class representing EXPRESS statements whose action is “transfer of control,” i.e., a change in the sequence of execution.. This class was introduced primarily to simplify the metamodel diagram.

Properties: abstract

14.2.2.1 Supertypes

Algorithms::Statement

14.2.2.2 Attributes

none.

14.2.2.3 Associations

none.

14.2.2.4 Other Roles

none.

14.2.3 Class: NullStatement

Definition: Represents an EXPRESS Null statement. A NullStatement is just a syntactic placeholder, made necessary by grammar rules that require the presence of at least 1 statement. It has the semantics: Take no action. It is modeled here, solely to permit reconstruction of the Express Text.

Note – See Clause 13.1 of ISO 10303-11:2004.

14.2.3.1 Supertypes

ControlStatement

14.2.3.2 Attributes

none.

14.2.3.3 Associations

none.

14.2.3.4 Other Roles

none.

14.2.4 Class: StatementBlock

Definition: represents a sequence of Statements to be executed in the given order.

In EXPRESS syntax, a number of constructs contain a statement or sequence of statements, and a “compound statement” is a statement that begins with BEGIN and ends with END and contains a sequence of statements. All such sequences have the semantics of the StatementBlock. The BEGIN/END case is here modeled as .delimited = True.

Note – See Clause 13.5 of ISO 10303-11:2004.

14.2.4.1 Supertypes

Algorithms::Statement

14.2.4.2 Attributes

Attribute: delimited **To: MOF::Boolean**

Definition: Is true if the StatementBlock was delimited by BEGIN and END tokens, false if it is implicit in the body of some other Statement.

Note – The sole purpose of this attribute is to be able to reconstruct the source EXPRESS text properly.

Multiplicity: 1..1

14.2.4.3 Associations

AssociationEnd: body-statements **To: Algorithms::Statement**

via: block-sequences-statements

Definition: represents the relationship of a StatementBlock to the Statements of which the sequence consists.

Note – Every EXPRESS syntax whose semantics is a StatementBlock requires the body to consist of at least 1 statement, but it may consist solely of a Null statement. This model permits the body to be (semantically) empty – the single Null statement need not be modeled. Even the EXPRESS text reconstruction is clear without the existence of a NullStatement in this case.

Multiplicity: 0..* ordered

Properties: composite

14.2.4.4 Other Roles

none.

14.2.5 Association: block-sequences-statements

Definition: represents the relationship of a StatementBlock to the Statements of which the sequence consists.

14.2.5.1 Association Ends

AssociationEnd: body-statements **To: Algorithms::Statement**

Definition: represents the relationship of a StatementBlock to the Statements of which the sequence consists.

Note – Every EXPRESS syntax whose semantics is a StatementBlock requires the body to consist of at least 1 statement, but it may consist solely of a Null statement. This model permits the body to be (semantically) empty – the single Null statement need not be modeled. Even the EXPRESS text reconstruction is clear without the existence of a NullStatement in this case.

Multiplicity: 0..* ordered

AssociationEnd: in-block **To: StatementBlock**

Definition: represents the relationship between a Statement and the StatementBlock, if any, in which it occurs.

StatementBlocks may, but need not, occur directly in other StatementBlocks.

Multiplicity: 0..1

14.3 ALIAS Statements

This section describes the ALIAS statement. Figure 47 depicts the associated concepts.

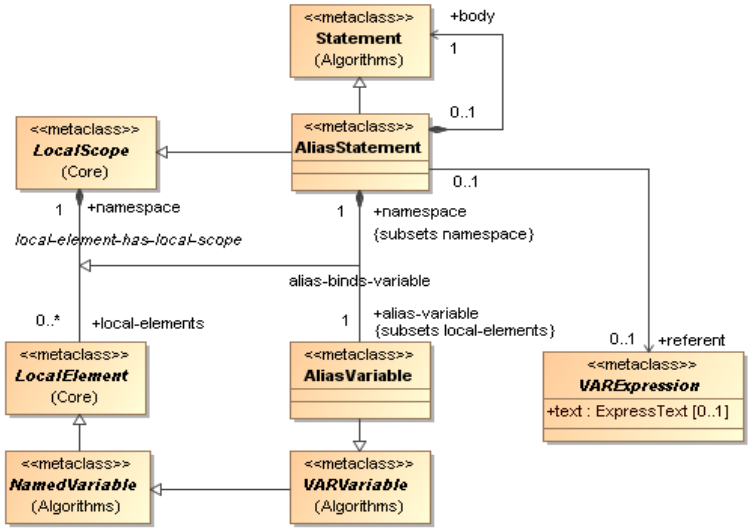

Figure 47 - ALIAS Statements

14.3.1 Class: AliasStatement

Definition: Represents an EXPRESS ALIAS statement. An ALIAS statement introduces a NamedVariable (the `alias-variable`) to represent the result of a VARExpression (the `referent`). The AliasVariable is not a Variable, and the interpretation is not assignment. The ALIAS statement creates a VARVariable that is persistently associated with the cell specified by the VARExpression over changes in the content of that cell during execution of the `body`. Within the body of the ALIAS statement, any assignment to the AliasVariable assigns the value to the referent cell, and any VariableRef that refers to the AliasVariable refers to the current value in that cell.

See Clause 13.2 of ISO 10303-11:2004.

14.3.1.1 Supertypes

Core::LocalScope, Algorithms::Statement

14.3.1.2 Attributes

none.

14.3.1.3 Associations

AssociationEnd: alias-variable **To: AliasVariable**

via: alias-binds-variable

Subsets: Core::LocalScope:local-elements

Definition: the AliasVariable that is introduced by the AliasStatement and bound to the `:referent`.

Multiplicity: 1..1

Properties: composite

AssociationEnd: body **To: Algorithms::Statement**

Definition: the Statement (or StatementBlock) specifying the action to be taken by the AliasStatement.

Note – The AliasStatement has the effect of "fixing" the referent of the alias-variable, in the case in which the Statement is a StatementBlock that includes actions that alter the values of elements of the VARExpression.

Multiplicity: 1..1

Properties: composite

AssociationEnd: referent **To: VARExpression**

Definition: the VARExpression that specifies the referent of the AliasVariable – the cell to which the AliasVariable refers during execution of the body of the ALIAS statement.

Multiplicity: 1..1

14.3.1.4 Other Roles

none.

14.3.1.5 Rules

Constraint (OCL)

```
self->alias-variable->namespace = self;
```

14.3.2 Class: AliasVariable

Definition: a NamedVariable that is created by an ALIAS statement, and whose scope is the body of the ALIAS statement. An Alias Variable is a VARVariable: it does not hold an Instance; it refers to cell that holds an Instance. The referent of the AliasVariable is specified by the value of the VARExpression assigned to it by the ALIAS statement.

Note – See Clause 13.2 of ISO 10303-11:2004.

14.3.2.1 Supertypes

Algorithms::VARVariable

14.3.2.2 Attributes

none.

14.3.2.3 Associations

AssociationEnd: namespace **To: AliasStatement**

via: alias-binds-variable

Subsets: Core::LocalElement:namespace

Definition: the AliasStatement that is the scope of the AliasVariable.

Multiplicity: 1..1

14.3.2.4 Other Roles

none.

14.3.3 Association: alias-binds-variable

Definition: represents the relationship between the AliasStatement and the AliasVariable it defines.

14.3.3.1 Supertypes

Core::local-element-has-local-scope

14.3.3.2 Association Ends

AssociationEnd: alias-variable **To: AliasVariable**

Definition: the Variable that is introduced by the AliasStatement and bound to a Reference.

Multiplicity: 1..1

Properties: composite

AssociationEnd: namespace **To: AliasStatement**

Definition: the AliasStatement that is the scope of the AliasVariable.

Multiplicity: 1..1

14.4 Assignment Statements

This section describes assignment statements. Figure 48 depicts the associated concepts.

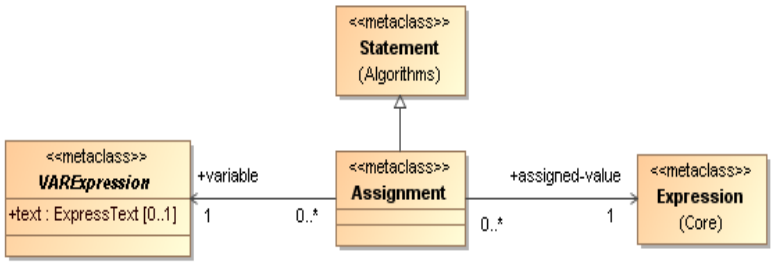

Figure 48 - Assignment Statements

14.4.1 Class: Assignment

Definition: Represents an EXPRESS assignment statement. An Assignment causes the value of the Variable that is specified by the `.variable` VARExpression to become equal to the result of the `.assigned-value` Expression.

Note – See Clause 13.3 of ISO 10303-11:2004.

14.4.1.1 Supertypes

Algorithms::Statement

14.4.1.2 Attributes

none.

14.4.1.3 Associations

AssociationEnd: assigned-value **To: Core::Expression**

Definition: the Expression whose result is the value to be assigned.

Multiplicity: 1..1

AssociationEnd: variable **To: VARExpression**

Definition: the VARExpression that designates the object whose value is to be replaced.

Note – The VARExpression must not refer to an object that is part of the state of an EntityInstance in the Population. It may, however, refer to an object that holds (a reference to) an EntityInstance, or to an object (other than an EntityInstance) that holds an EntityValue.

Multiplicity: 1..1

14.4.1.4 Other Roles

none.

14.5 CASE Statements

This section describes CASE statements. Figure 49 depicts the associated concepts.

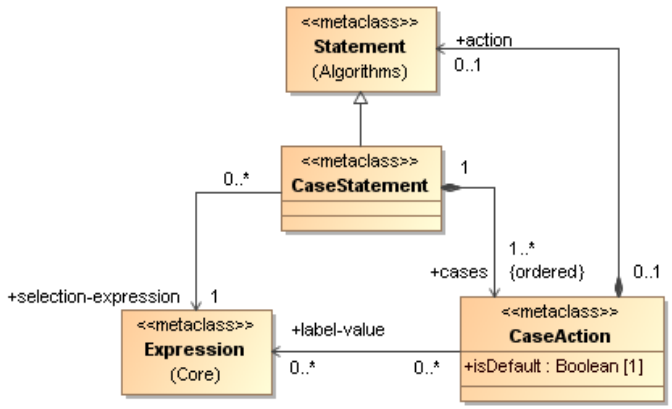

Figure 49 - CASE Statements

14.5.1 Class: CaseAction

Definition: represents a possible action to be taken, together with the .label-values that identify the case and enable it to be selected. Among the cases for a given CaseStatement, one CaseAction may be designated the “default” action, which is taken if no other action meets the selection criteria.

14.5.1.1 Supertypes

none.

14.5.1.2 Attributes

Attribute: isDefault **To: MOF::Boolean**

Definition: True if this CaseAction represents the default action to be taken if no other case label matches the value of the selection-expression; otherwise False.

Multiplicity: 1..1

14.5.1.3 Associations

AssociationEnd: action **To: Algorithms::Statement**

Definition: the Statement (or StatementBlock) that defines the actions, if any, to be executed if that case is selected.

Multiplicity: 0..1

Properties: composite

AssociationEnd: label-value **To: Core::Expression**

Definition: an Expression whose result is a case label. When the value of the .selection-expression matches the value of the Expression (which is often a Literal), the associated CaseAction defines the action to be taken by the CaseStatement.

Multiplicity: 0..* unordered

14.5.1.4 Other Roles

From: CaseStatement as cases

Multiplicity: 1..1

14.5.1.5 Rules

Constraint labels-unless-default (OCL)

```
if NOT (self->isDefault) THEN SizeOf(self->label-value) > 0;
```

Only the default CaseAction can have no label-values.

Constraint one-default (EXPRESS)

```
SizeOf(Query(c <* self.cases : c.isDefault)) <= 1;
```

At most 1 CaseAction in the list of cases for a given CaseStatement can have .isDefault = True.

14.5.2 Class: CaseStatement

Definition: represents an EXPRESS CASE statement. The CASE statement selects and executes a single CaseAction (from the list of CaseActions), based on the value of a selection-expression. The .cases are considered in order, and the first CaseAction whose label-value matches the value of the .selection-expression is the action that is taken. If no CaseAction has a label-value that matches the value of the .selection-expression, the CaseAction for which .isDefault is true, if any, is taken; otherwise, no action is taken.

Note – See Clause 13.4 of ISO 10303-11:2004.

14.5.2.1 Supertypes

Algorithms::Statement

14.5.2.2 Attributes

none.

14.5.2.3 Associations

AssociationEnd: cases **To: CaseAction**

Definition: represents the possible actions to be taken, in order of consideration, each labeled by one or more values.

Multiplicity: 1..* ordered

Properties: composite

AssociationEnd: selection-expression **To: Core::Expression**

Definition: the Expression that is used to choose the CaseAction to be taken.

Multiplicity: 1..1

14.5.2.4 Other Roles

none.

14.6 IF Statements

This section describes IF...THEN...ELSE statements. Figure 50 depicts the associated concepts.

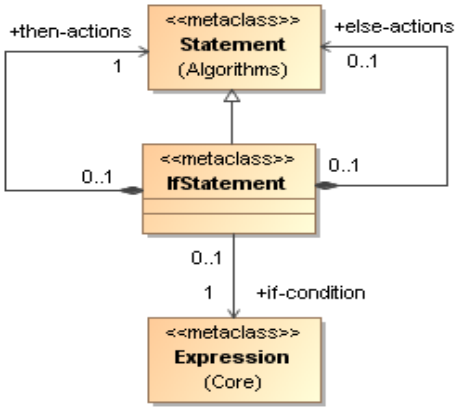

Figure 50 - IF Statements

14.6.1 Class: IfStatement

Definition: represents an EXPRESS IF...THEN...ELSE statement.

Note – See Clause 13.7 of ISO 10303-11:2004.

14.6.1.1 Supertypes

Algorithms::Statement

14.6.1.2 Attributes

none.

14.6.1.3 Associations

AssociationEnd: else-actions **To: Algorithms::Statement**

Definition: the Statement (or StatementBlock) specifying the actions to be taken when the condition is False.

Multiplicity: 0..1

Properties: composite

AssociationEnd: if-condition **To: Core::Expression**

Definition: an Expression that defines the condition used to determine whether to perform the “then-actions” or the “else-actions.”

Note – The if-condition is wholly owned by the IfStatement. It is not treated as reusable.

Multiplicity: 1..1

AssociationEnd: then-actions **To: Algorithms::Statement**

Definition: the Statement (or StatementBlock) specifying the actions to be taken when the condition is True.

Multiplicity: 1..1

Properties: composite

14.6.1.4 Other Roles

none.

14.7 Procedure Calls

This section describes procedure call statements. Figure 51 depicts the associated concepts.

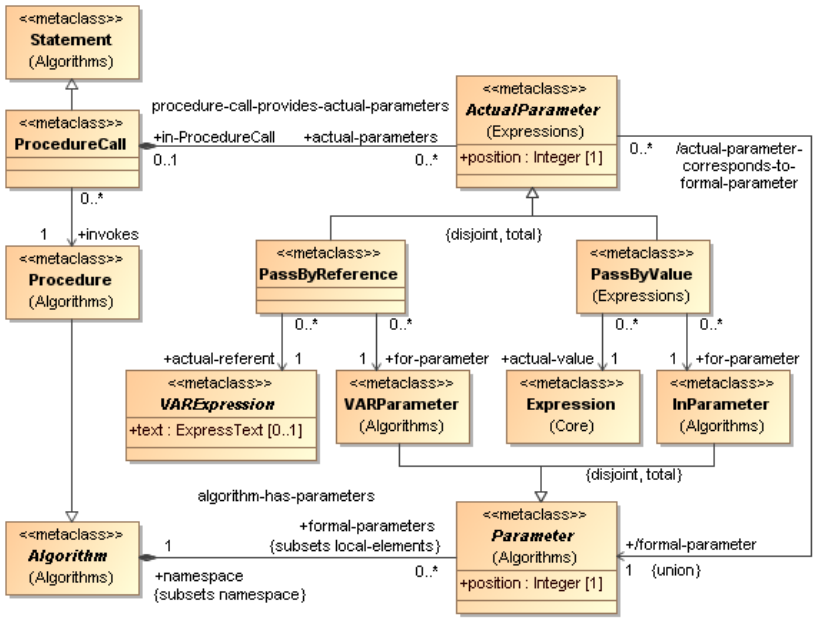

Figure 51 - Procedure Calls

14.7.1 PassByReference

Definition: An ActualParameter that is passed “by reference.” At the time of Algorithm invocation, the `actual-referent` VARExpression is evaluated to identify a cell and that cell becomes the referent of the VARParameter during execution of the Algorithm.

14.7.1.1 Supertypes

ActualParameter

14.7.1.2 Attributes

none.

14.7.1.3 Associations

AssociationEnd: actual-referent **To: Statements::VARExpression**

Definition: the VARExpression that denotes the cell that is to be the referent of the formal VARParameter during the invocation. This is the expression that is syntactically the `actual_parameter` when the corresponding formal parameter is a VAR Parameter.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

AssociationEnd: for-parameter **To: Algorithms::InParameter**

Subsets: Expressions::ActualParameter:formal-parameter

Definition: the formal parameter to which the actual referent is assigned.

Note – See 12.8 of ISO 10303-11:2004.

Multiplicity: 1..1

14.7.1.4 Other Roles

none.

14.7.1.5 Rules

Constraint

```
exists(self->inProcedureCall);
```

Every PassByReference appears in a ProcedureCall (not a FunctionCall).

14.7.2 Class: ProcedureCall

Definition: Represents an EXPRESS procedure call statement. A procedure call causes an instance of a defined Procedure to be created, and the actual parameter values to be passed to the corresponding formal parameters. The .actual-value Expression corresponding to each InParameter is evaluated and the result is copied into the corresponding InVariable. Each VARParameter is set to refer to the Variable that is the result of the VARExpression that appears as the corresponding actual parameter. Then the declared LocalVariables are instantiated, according to their declared types (which may be ActualTypes), with initial values if specified. Finally, the StatementBlock that is the algorithm body is executed.

Note – See Clause 13.8 of ISO 10303-11:2004.

14.7.2.1 Supertypes

Algorithms::Statement

14.7.2.2 Attributes

None.

14.7.2.3 Associations

AssociationEnd: actual-parameters **To: Expressions::ActualParameter**

via: procedure-call-provides-actual-parameters

Definition: the ActualParameters to be passed at the time of invocation.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: invokes **To: Algorithms::Procedure**

Definition: the Procedure that is invoked by the ProcedureCall.

Multiplicity: 1..1

14.7.2.4 Other Roles

none.

14.7.3 Association: procedure-call-provides-actual-parameters

Definition: represents the relationship between the ProcedureCall statement and the ActualParameters to be passed at the time of invocation.

14.7.3.1 Association Ends

AssociationEnd: actual-parameters **To: Expressions::ActualParameter**

Definition: the ActualParameters to be passed at the time of invocation.

Multiplicity: 0..* unordered

Properties: composite

AssociationEnd: in-ProcedureCall **To: ProcedureCall**

Definition: the ProcedureCall, if any, in which the ActualParameter appears.

Multiplicity: 0..1

14.8 REPEAT Statements

This section describes REPEAT statements, and the associated ESCAPE and SKIP statements. Figure 52 depicts the associated concepts.

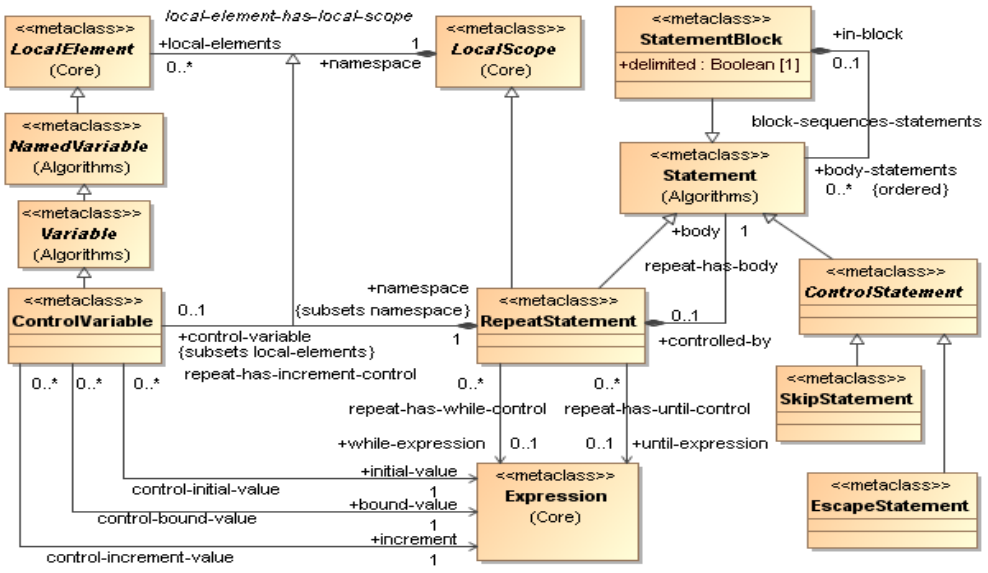

Figure 52 - REPEAT, SKIP, and ESCAPE Statements

14.8.1 Class: ControlVariable

Definition: a Variable representing the specification of for the control variable, if any, of the `REPEAT` statement.

If the REPEAT statement has an “increment control,” it introduces the control variable, whose scope is the RepeatStatement, and specifies the initial value for the control variable, a bound-value, and the increment value.

Note – In EXPRESS, the initial value, increment value and bound value are properties of the “increment control.” Here the “increment control” properties are assigned to the ControlVariable. See ISO 10303-11:2004 clause 13.9.1.

14.8.1.1 Supertypes

Algorithms::Variable

14.8.1.2 Attributes

None.

14.8.1.3 Associations

AssociationEnd: bound-value **To: Core::Expression**

Definition: the Expression whose value, taken together with the initial-value, specifies the bounds of a set of real numbers. Iteration of the repeated-body of the RepeatStatement terminates when the value of the control-variable lies outside that set.

Multiplicity: 1..1

AssociationEnd: increment **To: Core::Expression**

Definition: the Expression whose value is added to the value of the control-variable at the end of each iteration.

Multiplicity: 1..1

Note – When the EXPRESS syntax does not specify an increment value, the Expression is a Literal referring to the Integer value 1.

Note – See ISO 10303-11:2004 clause 13.9.1.

AssociationEnd: initial-value **To: Core::Expression**

Definition: the Expression that specifies the value to be assigned to the control-variable before the first iteration.

Multiplicity: 1..1

AssociationEnd: namespace **To: RepeatStatement**

via: repeat-has-increment-control

Subsets: Core::LocalElement:namespace

Definition: the RepeatStatement whose execution is controlled by the IncrementControl.

Multiplicity: 1..1

14.8.1.4 Other Roles

none.

14.8.1.5 Rules

Constraint

```
self->control-variable->namespace = self->for-loop;
```

14.8.2 Class: EscapeStatement

Definition: Represents an EXPRESS ESCAPE statement. An ESCAPE statement is always contained within the body of a RepeatStatement. Execution of an ESCAPE statement results in terminating the repetitiion of the repeated-body and continuing the control flow with the statement following the RepeatStatement.

Note – See Clause 13.11 of ISO 10303-11:2004.

14.8.2.1 Supertypes

ControlStatement

14.8.2.2 Attributes

none.

14.8.2.3 Associations

none.

14.8.2.4 Other Roles

none.

14.8.2.5 Rules

Constraint

```
exists(self->in-block->controlled-by);
```

An EscapeStatement shall only appear in the repeated-body of a RepeatStatement.

14.8.3 Class: RepeatStatement

Definition: Represents an EXPRESS REPEAT statement. The RepeatStatement defines an iteration. The execution of the repeated-body occurs zero or more times depending on the associated controls, which may be any combination of

- a increment-control (see ControlVariable)
- a while-expression
- an until-expression

If no control is specified, the iteration continues until an EscapeStatement is executed.

Note – See Clause 13.9 of ISO 10303-11:2004.

14.8.3.1 Supertypes

Core::LocalScope, Algorithms::Statement

14.8.3.2 Attributes

none.

14.8.3.3 Associations

AssociationEnd: body **To: Algorithms::Statement**

via: repeat-has-body

Definition: the Statement that specifies the actions to be iterated. When the EXPRESS text for the body includes multiple statements, the body Statement is a StatementBlock.

Multiplicity: 1..1

Properties: composite

AssociationEnd: control-variable **To: ControlVariable**

via: repeat-has-increment-control

Subsets: Core::LocalScope:local-elements

Definition: the specification for the increment control, if any. The increment control defines a control variable, its initial and final values, and the value by which it is incremented on each iteration.

Note – See ISO 10303-11:2004 clause 13.9.1.

Multiplicity: 0..1

Properties: composite

AssociationEnd: until-expression **To: Core::Expression**

Definition: the Boolean Expression that specifies a condition for terminating the iteration. If the value returned by the while-expression is True, the iteration is terminated.

Note – See ISO 10303-11:2004 clause 13.9.3.

Multiplicity: 0..1

AssociationEnd: while-expression **To: Core::Expression**

Definition: the Boolean Expression that specifies the condition for reiterating the repeated-body. If the value returned by the while-expression is False, the iteration is terminated.

Note – See ISO 10303-11:2004 clause 13.9.2.

Multiplicity: 0..1

14.8.3.4 Other Roles

none.

14.8.4 Class: SkipStatement

Definition: Represents an EXPRESS SKIP statement. A SKIP statement is always contained within the body of a RepeatStatement. Execution of a SKIP statement results in continuing the control flow with the “increment and test” operations of the RepeatStatement, skipping any intervening actions.

Note – See Clause 13.11 of ISO 10303-11:2004.

14.8.4.1 Supertypes

ControlStatement

Attributes

none.

14.8.4.2 Associations

none.

14.8.4.3 Other Roles

none.

14.8.4.4 Rules

Constraint

```
exists(self->in-block->controlled-by);
```

A SkipStatement shall only appear in the repeated-body of a RepeatStatement.

14.8.5 Association: repeat-has-body

Definition: represents the relationship between a RepeatStatement and the Statement (or StatementBlock) that specifies the actions to be iterated.

14.8.5.1 Association Ends

AssociationEnd: body **To: Algorithms::Statement**

Definition: the Statement that specifies the actions to be iterated. When the EXPRESS text for the body includes multiple statements, the body Statement is a StatementBlock.

Multiplicity: 1..1

Properties: composite

AssociationEnd: controlled-by **To: RepeatStatement**

Definition: the RepeatStatement that controls the iterated execution of the actions of the Statement.

Multiplicity: 0..1

14.8.6 Association: repeat-has-increment-control

Definition: represents the relationship between the RepeatStatement and its IncrementControl, if any.

14.8.6.1 Supertypes

Core::local-element-has-local-scope

14.8.6.2 Association Ends

AssociationEnd: control-variable **To: ControlVariable**

Definition: the specification for the control variable, if any, and its initial and final values.

Multiplicity: 0..1

Properties: composite

AssociationEnd: namespace **To: RepeatStatement**

Definition: the RepeatStatement whose execution is controlled by the IncrementControl.

Multiplicity: 1..1

14.9 RETURN Statements

This section describes RETURN statements. Figure 53 depicts the associated concepts.

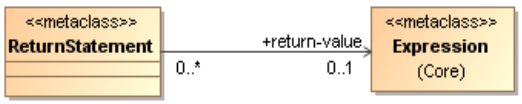

Figure 53 - RETURN Statements

14.9.1 Class: ReturnStatement

Definition: Represents an EXPRESS RETURN statement. A RETURN statement terminates the execution of a ProcedureCall or FunctionCall.

A RETURN statement that appears in the body of a Function may also specify an expression for the FunctionResult, that is, the value which is to be returned as the evaluation of a FunctionCall in which the RETURN statement is executed.

Note – See Clause 13.9 of ISO 10303-11:2004.

14.9.1.1 Supertypes

ControlStatement

14.9.1.2 Attributes

none.

14.9.1.3 Associations

AssociationEnd: return-value **To: Core::Expression**

Definition: An Expression that specifies the value to be returned as the Function result.

The result-value shall not exist for a RETURN statement that appears in the body of a Procedure. A RETURN statement that appears in the body of a Function and does not specify a result-value Expression implicitly specifies that the value of the FunctionResult variable is to be returned as the evaluation of a FunctionCall in which the RETURN statement is executed.

Multiplicity: 0..1

14.9.1.4 Other Roles

none.

14.10 VAR Expressions

This section defines the concepts associated with references to (what ISO 10303-11 calls) “variables” that may change in value during the execution of an invocation of an Algorithm or the evaluation of a GlobalRule. In general, such “variables” may be simple Variables, or more complex expressions denoting a part of a Variable. The general form of a “variable,” therefore, is modeled as a VAR Expression – an Expression that refers to an object that contains a value. Figure 54 depicts the concepts associated with VAR Expressions.

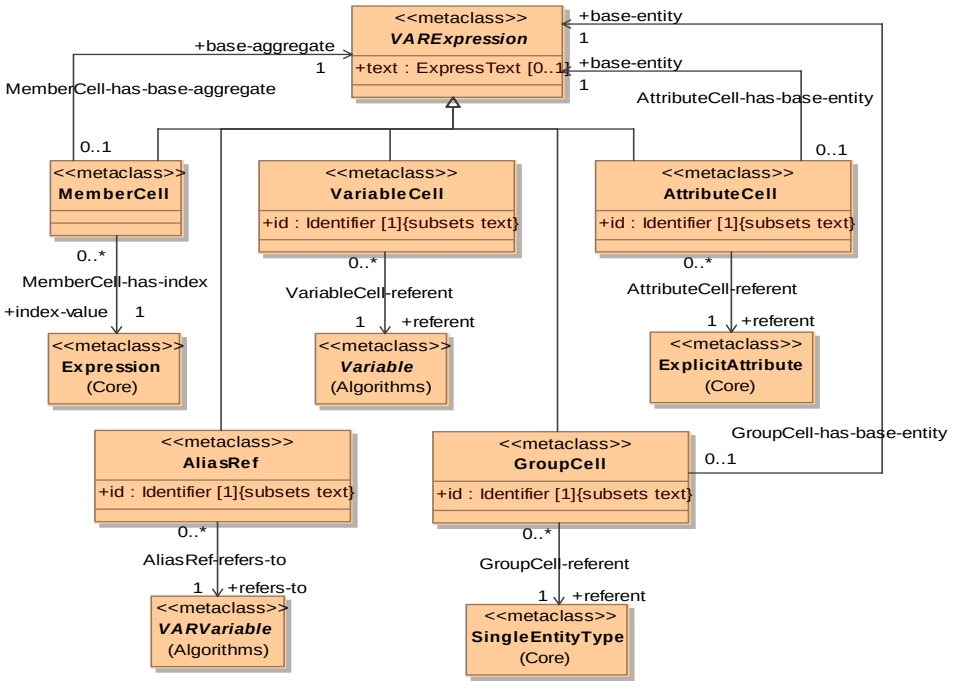

Figure 54 - VAR Expressions

VAR Expressions appear in assignment statements, in ALIAS statements and as ActualParameters that correspond to formal parameters that are VARParameters (which are permitted only in Procedure definitions).

Note – Primary Expressions, Index Expressions and Selector Expressions are similar in structure (and use the same syntax in EXPRESS), but they refer to the Instance that is the current value of the "variable" – the value currently held by that object. A VARExpression formally refers to the object (place) that holds an Instance, rather than to the Instance it contains. That is, for example, the meaning of the VariableRef is different from the meaning of the VariableCell, even though the EXPRESS syntax is the same. Because the meanings are different, they have different metamodels.

Note – A VAR Expression can never refer to an Instance in the modeled population. Instances in the Population cannot be created or modified by an EXPRESS Schema. For this reason, EXPRESS restricts the syntax for VAR Expressions to beginning with a `parameter_ref` or a `variable_ref`. This is reflected in the model.

14.10.1 Class: AttributeCell

Definition: A VARExpression whose referent is a cell (or "slot") containing the value of one ExplicitAttribute in an EntityValue or PartialEntityValue.

The referent of the `:base-entity` VARExpression shall be a cell that holds an EntityValue or PartialEntityValue that has a "slot" for the ExplicitAttribute that is the `:referent` of the AttributeCell. The cell/slot in the referent of the `:base-entity` that corresponds to that ExplicitAttribute is the referent of the AttributeCell.

Note – An EntityInstance in the Population is considered to be an object that holds an EntityValue. And therefore, an EntityInstance can be the referent of the base-entity. But it is not possible to change the value of an Attribute of an EntityInstance in the Population.

Note – An “entity-valued object” -- a Variable, Attribute, or aggregation member whose data type is an EntityType (or a SelectType whose select-list contains EntityTypes) -- may contain EntityInstances from the Population, or contain EntityValues that correspond to the EntityType, without reference to Instances in the Population. When the base-entity of an AttributeCell is an entity-valued object, it is not always clear whether it contains an EntityInstance, which is then the referent, or an EntityValue, which makes the entity-valued object the referent.

14.10.1.1 Supertypes

VARExpression

14.10.1.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: VARExpression:text

Definition: the lexical text of the identifier for the Attribute.

Multiplicity: 1..1

14.10.1.3 Associations

AssociationEnd: base-entity **To: VARExpression**

Definition: the VARExpression that identifies the cell that contains the EntityValue or PartialEntityValue that contains the referent of the AttributeCell.

Multiplicity: 1..1

AssociationEnd: referent **To: Core::ExplicitAttribute**

Definition: the ExplicitAttribute that designates the slot that is the referent of the AttributeCell.

Multiplicity: 1..1

14.10.1.4 Other Roles

none.

14.10.2 Class: GroupCell

Definition: A VARExpression whose referent is the group of cells (or “slots”) for the ExplicitAttributes that constitute a SingleEntityType within a cell that holds an EntityValue.

The referent of the `:base-entity` VARExpression shall be a cell that holds an EntityValue or PartialEntityValue that includes a SingleEntityValue for the SingleEntityType that is the `:referent` of the GroupCell. The group of cells/slots in the referent of the `:base-entity` that corresponds to that SingleEntityType is the referent of the GroupCell.

Note – An EntityInstance in the Population is considered to be an object that holds an EntityValue. And therefore, an EntityInstance can be the referent of the base-entity. But it is not possible to change the value of an Attribute of an EntityInstance in the Population.

Note – An “entity-valued object” -- a Variable, Attribute, or aggregation member whose data type is an EntityType (or a SelectType whose select-list contains EntityTypes) -- may contain EntityInstances from the Population, or contain EntityValues that correspond to the EntityType, without reference to Instances in the Population. When the base-entity of a GroupCell is an entity-valued object, it is not always clear whether it contains an EntityInstance, which is then the referent, or an EntityValue, which makes the entity-valued object the referent.

14.10.2.1 Supertypes

VARExpression

14.10.2.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: VARExpression:text

Definition: the lexical text of the identifier for the SingleEntityType.

Multiplicity: 1..1

14.10.2.3 Associations

AssociationEnd: base-entity **To: VARExpression**

Definition: the VARExpression that identifies the cell that contains the EntityValue or PartialEntityValue that contains the referent of the GroupCell.

Multiplicity: 1..1

AssociationEnd: referent **To: Core::SingleEntityType**

Definition: the SingleEntityType that designates the group of ExplicitAttribute slots that constitute the referent of the GroupCell.

Multiplicity: 1..1

14.10.2.4 Other Roles

none.

14.10.3 Class: MemberCell

Definition: A VARExpression that represents a reference to a cell that is a member of a cell whose datatype is an aggregation data type.

The cell that is the referent of the `:base-aggregate` VARExpression shall have a datatype that is an aggregation data type. The referent of the MemberCell is the member of that cell that is designated by the index or position value that is the result of the `:index-value` Expression.

14.10.3.1 Supertypes

VARExpression

14.10.3.2 Attributes

none.

14.10.3.3 Associations

AssociationEnd: base-aggregate **To: VARExpression**

Definition: the VARExpression that identifies the aggregate cell that contains the referent member cell.

Multiplicity: 1..1

AssociationEnd: index-value **To: Core::Expression**

Definition: the index or position value used to identify the member ~~object~~ cell within the aggregate ~~object~~ cell.

Multiplicity: 1..1

14.10.3.4 Other Roles

none.

14.10.4 Class: AliasRef

Definition: A VARExpression consisting only of the identifier for a VARVariable, i.e., an AliasVariable, or a VARParameter. The referent of the AliasRef VARExpression is the referent of the VARVariable designated by the `.refers-to` relationship.

Note – An AliasRef to a VARVariable produces a different result from a VariableRef to the same VARVariable. The AliasRef produces the referent of the VARVariable – the place that holds the value; the VariableRef produces the value that is currently in that place. In computer science terminology, the VariableRef "de-references" the VARVariable.

14.10.4.1 Supertypes

VARExpression

14.10.4.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: VARExpression:text

Definition: the lexical text of the identifier for the Parameter or the AliasVariable.

Multiplicity: 1..1

14.10.4.3 Associations

AssociationEnd: refers-to **To: Algorithms::VARVariable**

Definition: the AliasVariable or VARParameter whose referent is the referent of the AliasRef.

Multiplicity: 1..1

14.10.4.4 Other Roles

none.

14.10.5 Class: VARExpression

Definition: an expression that refers to a cell - a place - that contains a value.

Unlike Primary Expressions, Index Expressions and Selector Expressions, which are similar in structure, a VARExpression formally refers to the cell that holds an Instance, rather than to the Instance itself. The cell to which a VARExpression refers is called its *referent*. The type of a VARExpression is “reference to cell containing” the data type of the referent cell. The referent of a VARExpression can be:

- a LocalVariable
- an InParameter or FunctionResult
- a member of a cell whose data type is an AggregationType
- an ExplicitAttribute slot in a cell that contains an EntityValue or PartialEntityValue
- the cells that contain a SingleEntityValue in a cell that contains an EntityValue or PartialEntityValue
- the cell that is the referent of an AliasVariable or a VARParameter.

Properties: abstract

14.10.5.1 Supertypes

none.

14.10.5.2 Attributes

Attribute: text **To: Core::ExpressText**

Definition: the lexical representation of the VARExpression.

Multiplicity: 0..1

14.10.5.3 Associations

none.

14.10.5.4 Other Roles

From: PassByReference as actual-referent

From: Assignment as recipient

From: MemberCell as base-aggregate

From: AttributeCell as base-entity

From: GroupCell as base-entity

From: AliasVariable as referent

14.10.6 Class: VariableCell

Definition: A VARExpression that consists only of the identifier for a Variable. The referent of the VariableCell VARExpression is the cell that instantiates that Variable (as distinct from the value of that Variable). The Variable is designated by the `.referent` relationship.

Note – A VARExpression that consists of the identifier for an AliasVariable or a VARParameter is an AliasRef, not a VariableCell. A VariableCell differs from a VariableRef in that it refers to the place, not the value.

14.10.6.1 Supertypes

VARExpression

14.10.6.2 Attributes

Attribute: id **To: Core::Identifier**

Subsets: VARExpression:text

Definition: the lexical text of the identifier for the NamedVariable

Multiplicity: 1..1

14.10.6.3 Associations

AssociationEnd: referent **To: Algorithms::Variable**

Definition: the Variable whose instantiation is the referent object of the VariableCell VARExpression.

Multiplicity: 1..1

14.10.6.4 Other Roles

none.

15 Package : Express2

The Express2 Package has no immediate content. It simply combines the Rules Package with the full Statements Package, and thus contains all of the model elements for the language.

Figure 55 shows the complete view of the scope concepts in EXPRESS version 2. Note that the LocalScopes arise only when the Algorithms, Rules, Expressions, and Statements Packages are supported.

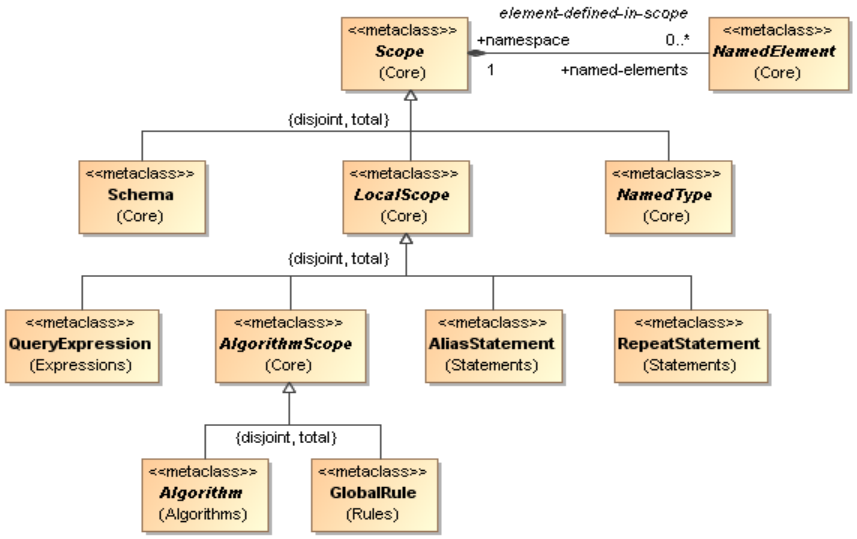

Figure 55 - Integrated Overview of Scopes

In a similar way, Figure 56 depicts the complete view of the NamedElement concepts in EXPRESS version 2, which are drawn from several packages.

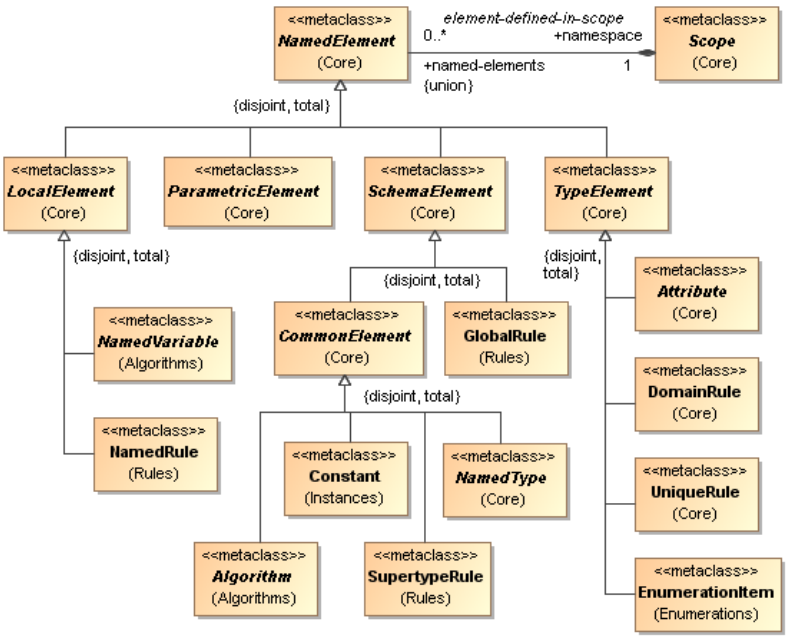

Figure 56 - Overview of Named Elements

15.1 Dependencies

Dependency on Package: Statements

Stereotypes: `import`

The Express2 Package depends on the Statements Package for complete modeling of EXPRESS Functions and Procedures. By way of the Statements Package, Express2 implicitly depends on the Expressions Package, for complete modeling of Expressions and thereby on the Algorithms, Core, and Instances Packages.

Dependency on Package: Rules

Stereotypes: `import`

The Express2 Package depends on the Rules Package in order to complete the support of all elements of the EXPRESS language. The Rules Package is the only package that is not required for the support of the Statements compliance point.

15.2 Classes and Associations

None.