Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Federal Department of Defence,
Civil Protection and Sport DDPS

Federal Office of Topography swisstopo

INTERLIS 2 – Metamodel

Edition 2022-06-17 (English)

Information and contact: www.interlis.ch, info@interlis.ch

This document was initially drafted in German, the authors of its English version have attempted to respect as much as possible the original text.

Contents

1 Introduction ... 3
1.1 Concept and objectives ... 3
1.2 Relationship with other standards ... 3
1.3 Relationship with INTERLIS-language ... 4
1.3.1 Completeness ... 4
1.3.2 Metaobjects ... 4
1.4 Reading tips ... 4

2 The INTERLIS-Metamodel ... 5
2.1 Basic constructs ... 5
2.1.1 Overview ... 5
2.1.2 Class MetaElement ... 5
2.1.3 How to deal with extensions ... 6
2.1.4 Models and topics ... 6
2.1.5 Translations ... 7
2.2 Main constructs ... 7
2.2.1 Overview ... 7
2.2.2 Structures and classes ... 8
2.2.3 Relationships and references ... 8
2.2.4 Data units ... 9
2.2.5 Arbitrary OID ... 9
2.3 Types ... 10
2.3.1 Overview ... 10
2.3.2 Types for text and blackbox ... 11
2.3.3 Numeric types and units ... 11
2.3.4 Booleantyp ... 11
2.3.5 Coordinate type ... 11
2.3.6 Enumerations ... 11
2.3.7 Domains of classes and attributes ... 11
2.3.8 Object types ... 12
2.3.9 Units ... 12
2.4 Generic domains ... 12
2.5 Views and graphics ... 13
2.6 Other constructs ... 14
2.6.1 Expressions and factors ... 14
2.6.2 Consistency constraints ... 14
2.6.3 Definition of functions ... 15
2.6.4 Metaobjects ... 16
2.6.5 Run time parameters ... 16

3 Sub-dividing model data into baskets ... 17

4 The INTERLIS-Metamodel in INTERLIS 2 ... 18

1 Introduction

1.1 Concept and objectives

The term metamodel refers to the data model which describes the modelling of model descriptions. Data corresponding to a metamodel, and hence describing a data model, are called model data.

As a basic principle the contents of model data are the same as those of model descriptions in INTERLIS-language. However their structure and coding do not aim at elegance of description and readability for users but merely at their simplest possible application in programs, e.g.:

- Generic application programs whose functioning partially depends on model information.
- Generators of program parts according to model information.
- Tools converting model information in any way (e.g. into other standards).

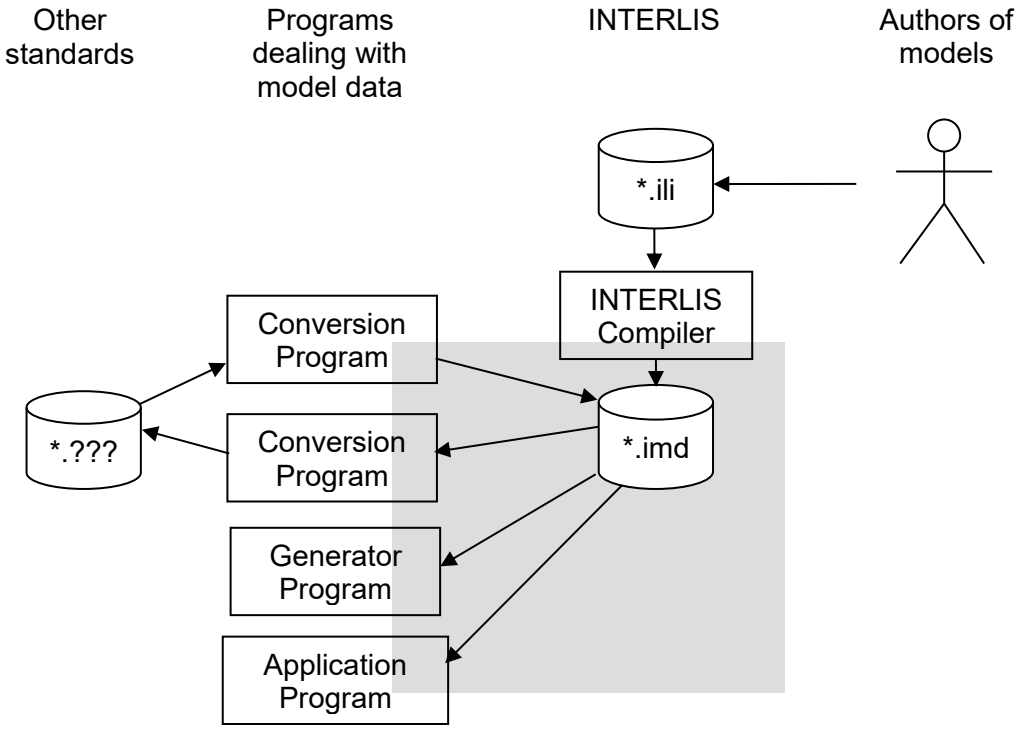

Figure 1: Relevance of model data

Consequently this document is not intended for persons who develop INTERLIS-models, but for those who write programs making use of model information, resp. generating INTERLIS-model data (INTERLIS-compiler or conversion programs of other standards).

1.2 Relationship with other standards

Metamodels and corresponding model data do not only exist for INTERLIS but also for other data description technologies (e.g. UML-MOF, XML-schema). Hence we faced the question of whether the metamodel of such a technology should be adopted. This possibility was rejected on grounds of considerable differences which would have resulted in the loss of proximity to the INTERLIS-description language. Yet it is still possible to use other description technology: Based upon model data from other technologies it is possible to generate INTERLIS-model data and vice-versa. Such a transfer may also be

added to the INTERLIS-specification. Furthermore it may appear convenient to establish basic models for certain standards (e.g. GML), which would contain the basic structure of the external standard in INTERLIS.

1.3 Relationship with INTERLIS-language

1.3.1 Completeness

Several semantic rules are already connected with the INTERLIS-language (cf. INTERLIS 2-Reference Manual).

Since the metamodel is orientated towards the application of model data in programs rather than towards correct generating of meta data, on the one hand semantic rules have not been applied on all levels (e.g. as consistency constraints) within the metamodel. On the other hand certain matters are considered redundant (e.g. on several occasions not only inherent properties but also inherited properties are represented by means of direct sub-elements).

Thus data within the INTERLIS 2-metamodel (short Model data; *.imd) never represent the primary description of the data model. Most commonly this description appears in text form (*.ili), is determined by another modeling technique or is controlled by a modeling editor (e.g. UML/INTERLIS-editor). Such a primary data model may contain more information than the INTERLIS 2-metamodel (above all regarding the representation of models in diagrams).

1.3.2 Metaobjects

In INTERLIS 2-language the term metaobject (cf. INTERLIS 2-Reference Manual, chapter 2.10) is often used. However such metaobjects describe reference systems and signatures and should not be confounded with objects of modeling data. In order to minimize the risk of mistakes objects of INTERLIS-model data are called metaelements.

1.4 Reading tips

All descriptions concerning the metamodel presume a profound knowledge of INTERLIS-constructs (models, topics, classes) as defined in the Reference Manual for INTERLIS 2.

Detailled descriptions also proceed on the assumption that the metamodel is included in the form of the INTERLIS 2-description according to chapter 4.

In UML diagrams classes and structures are represented in the same way. The application of a structure within a different structure or class is represented as a composition, the attribute name appearing next to the filled-in rhombus – contrary to relationships between classes.

Reference attributes are represented, in a way similar to relationships, as links between their respective classes of origin and target. In the case of classes of origin the reference attribute is listed as an attribute and the relationship line leading to this attribute.

2 The INTERLIS-Metamodel

2.1 Basic constructs

2.1.1 Overview

Figure 2 displays an overview of the INTERLIS-metamodel in the form of a UML class diagram. Data describing the actual models and data derived from translations are clearly seperated. Each class and all relationships between classes will be described in the following paragraphs.

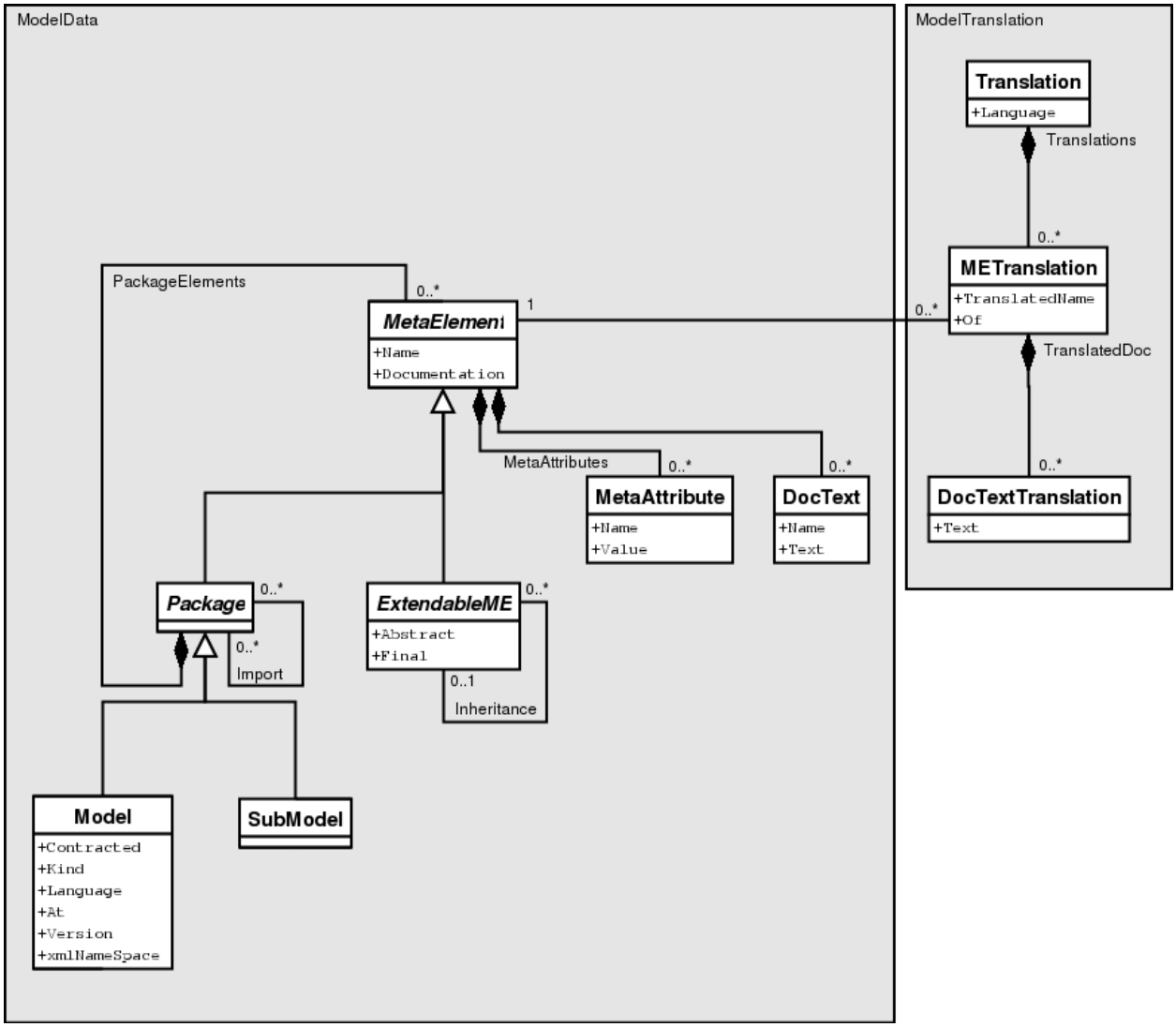

Figure 2: Overview basic constructs

2.1.2 Class MetaElement

The abstract class MetaElement is the basic object class for all model data. In order to avoid confusions with MetaObjects as perceived by the data description language (cf. INTERLIS 2-Reference Manual chapter 2.10), we call them metaelements rather than metaobjects.

Object identification of a metaelement is based upon a name path containing all names of superior metaelements plus, at the very end, its own name (attribute "Name"). Names are seperated by means of a period, thus guaranteeing stability of object identification throughout repeated compilation (even in the case of model alterations, provided they do not concern this particular metaelement.

Example: The internal INTERLIS-data model (cf. INTERLIS 2-Reference Manual, Appendice A) contains the structure UTC with the attribute Hours. Hence the metaelement of this attribute description will possess the OID "INTERLIS.UTC.Hours".

A metaelement will contain a (possibly empty) list of documentation texts (class "DocText"). Each documentation text can be specified by a name and contains the documenting text. INTERLIS does not govern the process of generating documentation texts, rather leaving this aspect to the tool generating model data.

It is possible to assign meta attributes to a metaelement by means of composition (class "MetaAttribut"). Neither the INTERLIS-language nor the metamodel provide a more precise definition of meta attributes. They mainly serve to add information beyond INTERLIS as a component of model data. Meta attributes have a name (attribute "Name"), which must be unique within the meta atributes of one metaelement, and a descriptive value (attribute "Value"). The object identification of meta attributes always consists of of the OID of the metaelement plus the permanent name METAOBJECT (a key word in INTERLIS 2) and its proper name. If, in the example above, a meta attribute by the name X, were defined, its OID would be "INTERLIS.UTC.Hours.METAOBJECT.X". Meta attributes need not necessarily be stored in the same basket as the metaelement. Above all it is possible to define INTERLIS-baskets (cf. INTERLIS 2-Reference Manual, chapter 1.4.5) containing nothing but meta attributes of different metaelements. This is why, within the relationship MetaAttributes, the role MetaElement is specified as external.

2.1.3 How to deal with extensions

Several INTERLIS 2-constructs may be extended as defined by the object-oriented paradigma (e.g. topics, classes, attributes). Such constructs always are direct or indirect sub classes of the abstract basic class "ExtendableME" (with the attributes Abstract, Generic and Final). The relationship Inheritance describes the connection between the basic metaelement (super) and the extended metaelement (sub). Since the basic metaelement can be defined within a different basket, the role Super is specified as external.

2.1.4 Models and topics

In order to keep the INTERLIS-metamodel as similar as possible to the UML metamodel, models and topics in the metamodel appear slightly different from their definition in the INTERLIS-language.

Analoguous to UML it is the abstract class "Package" that sets the basis for all models and sub models. Via the relationship "PackageElements" a package may – with the exception of models – contain any number of other metaelements. Above all a sub model (class "SubModel") may also contain other sub models, even if the circumstance cannot be defined by means of the current language (INTERLIS 2.4). Via the relationship "Import", a package is linked to the imported packages.

By means of metaelements of the class "DataUnit" we describe how data units (i.e. especially baskets of an INTERLIS-data exchange) can be organized and how they depend on each other. All type designations (XML-tags) to be used during transfer can be derived from the "DataUnitName". The metamodel merely determines that data units must be defined within a package. However the following rules apply to the current language versions INTERLIS 2.4:

- An INTERLIS-model is represented by a metaelement of the class "Model". In the form of attributes it contains all information necessary according to the linguistic possibilities. In view of a possible XML transfer the attribute xmlns has been defined. As regards INTERLIS its value is http://www.interlis.ch/INTERLIS2.4.
- A "Topic" of the INTERLIS-language is represented by one metaelement of each of the classes "SubModel" and "DataUnit". By means of the class SubModel a topic may be considered a package in the UML sense of the term; within this package the metaelement "DataUnit" (Name: "BASKET",

DataUnitName: Modelname"."Topicname) describes the structure and organization of corresponding INTERLIS-data baskets ("Basket") (cf. chapter 2.2.4).

Because the CONTRACTED construct has been discontinued in INTERLIS 2.4, the corresponding attribute is no longer mandatory, but is still available for compatibility reasons.

2.1.5 Translations

INTERLIS 2-models can be translated (TRANSLATION OF). However all meta data always refer to the original models. Even when importing a translated model, all relationships and references will refer to the metaelements of the respective original language.

Specific baskets are generated for all translations (metamodel topic "ModelTranslation"). Objects of the class "Translation" describe the language of the translation and contain all metaelement translations (structure "METranslation"). They refer to a certain metaelement (in the original language) and contain the translated name and, if necessary, translated comments in the same order as original comments.

2.2 Main constructs

2.2.1 Overview

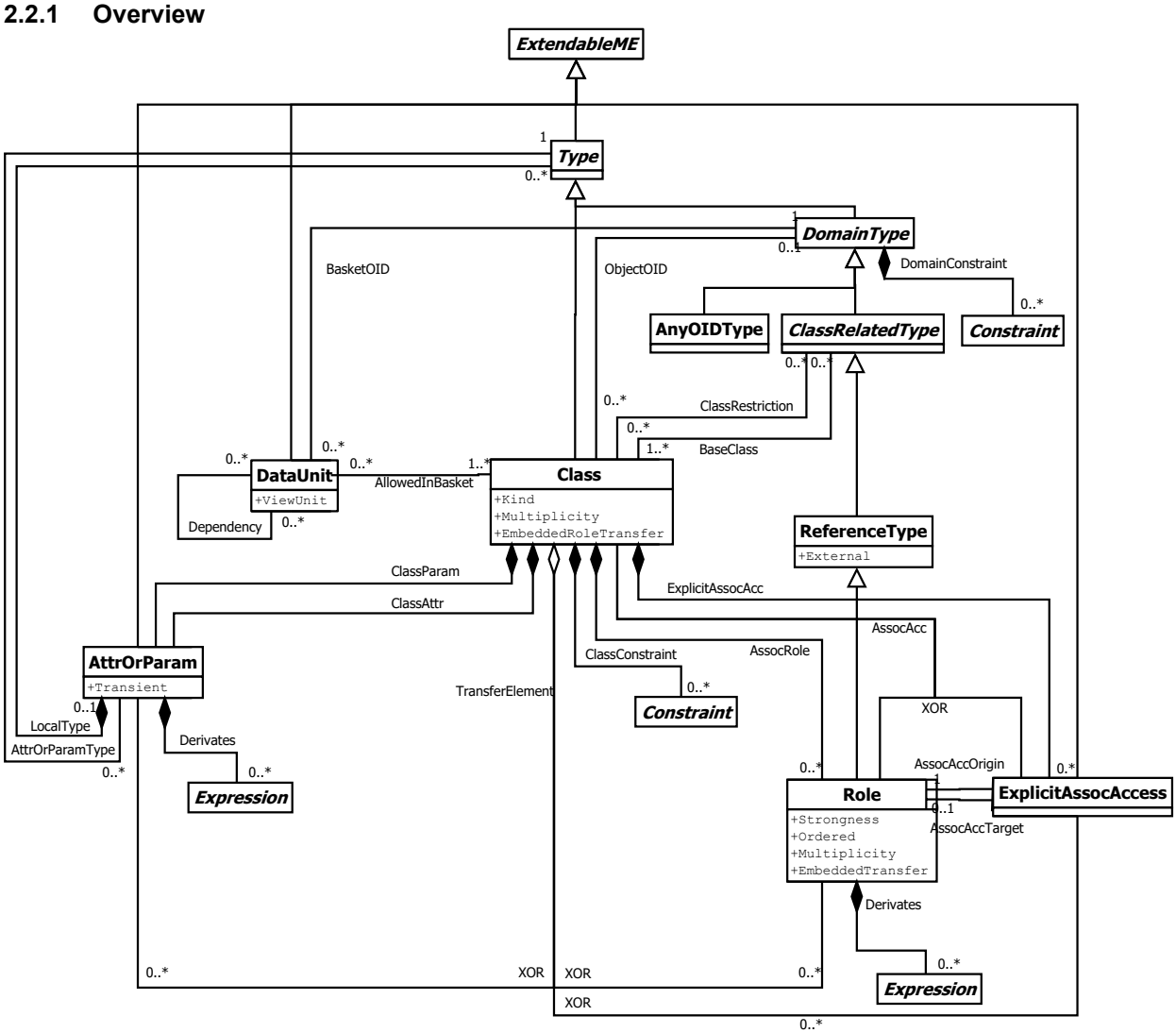

Figure 3: Overview main constructs

2.2.2 Structures and classes

As structures and classes (much the same as relationships and views) have the same formal definition (cf. INTERLIS 2-Reference Manual, chapter 2.5.3), in the metamodel they are represented by one single construct, the class "Class". Its attribute "Kind" states what kind (structure, class, assocation, view) we are dealing with. Under views (cf. chapter 2.5), further information is listed in a sub-class.

The most important property of a class is the definition of its attributes. It is achieved by means of the composition "ClassAttr". If the class described is a sub class of the INTERLIS-class Metaobject, it is also possible to assign it parameters. This is achieved by means of the composition "ClassParam".

Since descriptions of attributes and parameters are, to a large extent, identical; both are described by means of the class "AttrOrParam". In both cases the class name is part of the name path of the OID. The most important property of both attribute and parameter is its kind. It is described by means of the relationship "AttrOrParamType". If the values of attributes are defined by "Expressions", these are described by means of the list attribute "Derivates".

All types that were not defined as independent constructs but within the scope of attrtbute definition (e.g. (z.B. Name: TEXT*30), are assigned to either the attribute or parameter by means of the compositions "LocalType". The value "Type" is assigned to the attribute "Name".

If the attribute is an AttributeRefType, then it is possible to define new types within certain restrictions. By means of the compsitions "LocalType" these are also assigned to the attribute. The value "Type." is assigned to the attribute "Name" followed by a sequence number (starting at 1).

The consistency constraints of a class are described by a set (INTERLIS-BAG) of corresponding structure elements (extension of the structure "Constraint") (cf. chapter 2.5.2).

An important property of a class description is the information concerning the order in which attributes and roles within the scope of an object of this same class are to be transfered. In order to make this property available in the simplest possible way it is accessible via the relationship "TransferElement", even though this results in redundance.

Any modeled class may display relationships to other modeled classes. See chapter 2.2.3 for further information.

2.2.3 Relationships and references

Primarily any relationship will be described by means of an object of the class "Class" (Class.Kind=Association). There is a compositional assignation (Attribut "Name": role name) of role objects (class "Role") to this object. All possible classes of reference objects are assigned to the super class "ClassRelatedType": On the one hand the base classes of all possible variants (relationship "BaseClass"), on the other hand possibly occuring restrictions (relationship "ClassRestriction"). (Classes assigned as restrictions always are sub-classes of the base classes and thus related directly or indirectly by means of the relationship "Inheritance".)

By means of the relation AssocAcc we describe relation accesses which are generated in the classes (target classes) joined by the relation. Under ordinary circumstances there is direct reference to the role leading to any target object.

In the case of multiple relationships with the same class name conflicts are unavoidable. Hence we introduce metaelements of the class "ExplicitAssocAccess" and establish a compositional assignation to the class by means of the relationship ExplicitAssocAcc (attribute "Name": relation access name). There is an explicit relation access via the relationship AssocAccOrigin establishing reference to the role (of the class that also features the relation access) which from the relationship’s point of view enables the relation access. By means of the relationship AssocAccTarget it can also refer to the role which allows to attain the target object from the relation’s point of view. Where this reference is missing, the relation access should

not enable to reach reference objects but relationship objects. Since neither the current INTERLIS-language nor the UML meta data model permit the generating of explicit relation accesses, they hardly are of practical relevance. However they document an actual shortcoming and measures to be taken for its rectification.

Reference attributes are represented by means of the class "ReferenceType".

2.2.4 Data units

Data units (class "DataUnit") describe the structure and organization of data baskets (INTERLIS-Baskets).

The dependency relationship "Dependencies" displays dependencies between data units. The OID type of the basket is a result of the relationship "BasketOID". The relationship "AllowedInBasket" defines all classes permitted to occur in such a container.

The relationship "DeferredGenerics" refers to generic domains for which no specific definition exists yet, so their concrete domain is specified in the transfer data.

The attribute "DataUnitName" features the type designation (XML-tag) to be used for transfer.

The current language definition INTERLIS 2.4 does not give an explicit definition of data units. They rather are a result of topic definitions (cf. chapter 2.1.4).

2.2.5 Arbitrary OID

With "AnyOIDType" we have introduced a special type of OID. There is a primary object corresponding to this type: "INTERLIS.ANYOID". By means of an inheritance relationship explicit OID types (text or numerical) refer either to this object or to explicit AnyOID types, which then refer to "INTERLIS.ANYOID".

2.3 Types

2.3.1 Overview

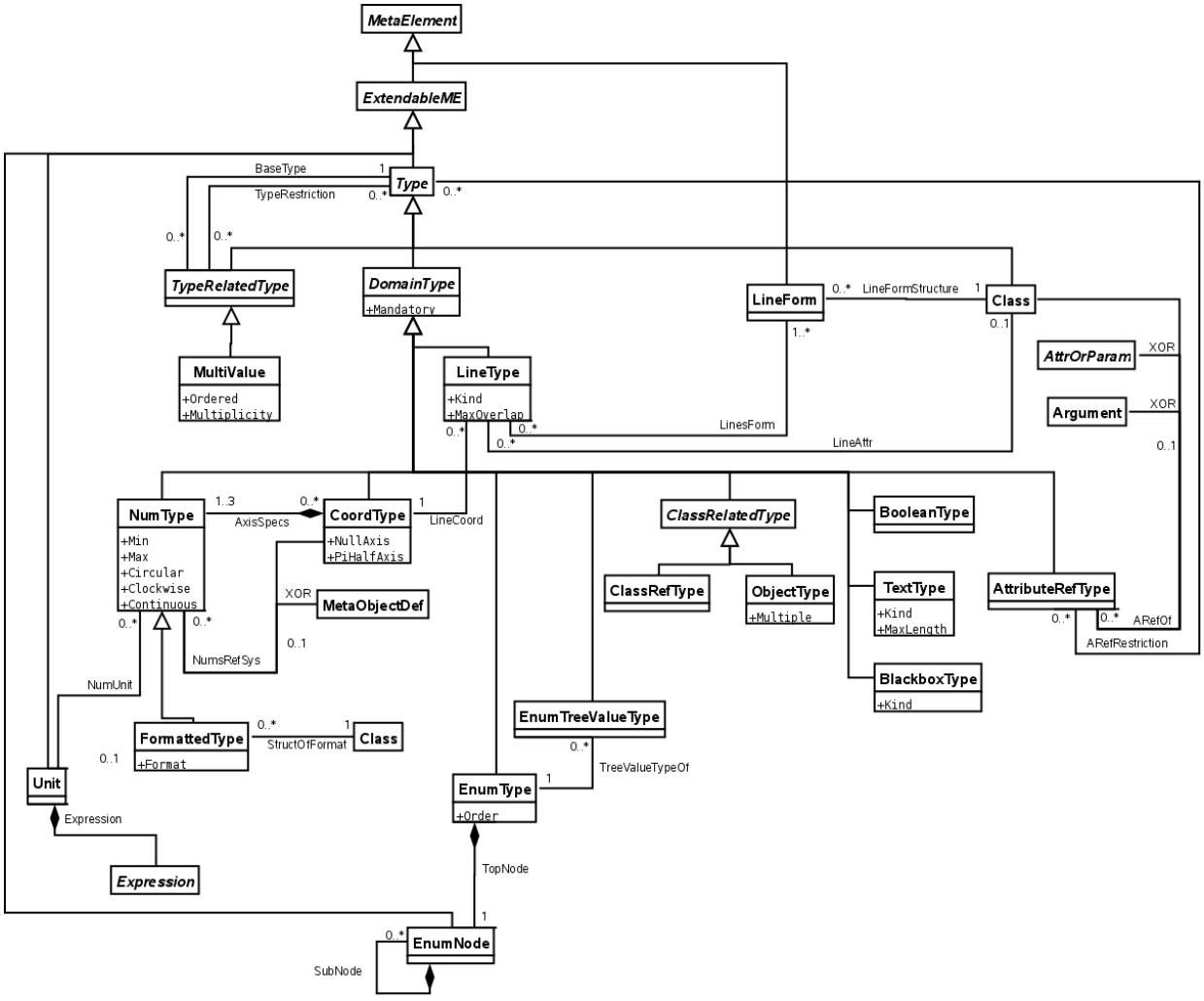

Figure 4: Overview types

In comparison with the INTERLIS 2-language we have somewhat generalized our type concept. Thus it is more in accordance with type concepts of programming languages. To start with we have introduced the following extensions of type: "DomainType", "Class" and "TypeRelatedType".

The DomainType defines a certain domain. According to the language a ZERO-value in general is permitted, unless this possibility is specifically excluded (attribute Mandatory). Within the scope of the "DomainType" we primarily distinguish the various base types (cf. chapter 2.3.2 to 2.3.6) and the line types ("LineType").

Since the organization of structures and that of classes are quite similar, we use the class "Class" for their description (cf. chapter 2.2). To avoid the necessity to treat structure attributes (attributes whose organization is described by the structure) as special cases in the metamodel, the class "Class" was realized as an extension of "Type". If, by means of the relationship "AttrOrParamType", the type "Class" is assigned to an attribute (class "AttrOrParam"), it can be nothing other than a structure.

According to the INTERLIS 2-Reference Manual an attribute type (rule "AttrType") as a whole can be either compulsory (MANDATORY) or optional. However in the case of structures this does not make sense: this statement must be made for every one of the attributes of the structure. Hence Class is a direct extension of Type and not of DomainType.

The class "TypeRelatedType" serves as basis for all types, which use another type (by means of the relationship "BaseType"), which again might be restricted (relation"TypeRestriction"). Currently there is only one extending class for "TypeRelatedType": "MultiValue". It serves to describe a set of sub elements. Attributes further describe whether this set is ordered (LIST) or not (BAG), and what cardinality it features.

2.3.2 Types for text and blackbox

With the types "TextType" und "BlackboxType" we primarily indicate the kind of type, with text type also its maximum length. If the size of a TextType is unlimited, the attribute "MaxLength" remains undefined.

2.3.3 Numeric types and units

The attributes contained in the numeric type ("NumType") equal information as intended in the INTERLIS-language.

Both units ("Unit") and reference system, provided they have been defined, are assigned by means of relationships ("NumUnit", "NumsRefSys").

Formatted types are regarded as special numeric types, since they are based on a composition of numeric fields of structures. If a formatted type describes e.g. hours, minutes and seconds, it can also be regarded as a numeric type with seconds. Thus hours and minutes must be converted accordingly. If the formatted type deals with e.g. months and days, this may be considered as a numeric type with days. However, without specific conversion guidelines, we must assume all months to have 31 days. Consequently the numeric type no longer is continuous (attribute "Continuous").

2.3.4 Booleantyp

Boolean is represented with a distinct type ("BooleanType") and not, contrary to statements in the Reference Manual, as a special enumeration.

2.3.5 Coordinate type

The coordinate type ("CoordType") defines all numeric types corresponding to an axe (relationship "AxisSpecs") and information concerning their rotation.

2.3.6 Enumerations

Each enumeration type (class "EnumType") is exhaustively described, i.e. it contains all nodes (class "EnumNode"). To start with an artificial node is assigned to the enumeration type (by the name of "TOP"). All real nodes are direct or indirect subordinates of this artificial node (each having its enumeration value as name). In the case of extended enumeration types both the enumeration type and all transfered nodes are linked with the corresponding inherited metaelement (relation "Inheritance"). If it is no longer permitted to add further sub nodes to a node, the attribute "ExtendableME.Final" features the value "true". The attribute "ExtendableME.Abstract" can have no other value than "false".

The type representing a value which admits all nodes and not only leaves (cf. INTERLIS 2-Reference Manual, chapter 2.8.2) is linked to the actual enumeration type by means of the relationship "TreeValueTypeOf".

2.3.7 Domains of classes and attributes

The type for the domain of classes (class "ClassRefType") is an extension of the class "ClassRelatedType", which is used within the scope of relationships and references and will be described more in detail in the corresponding paragraph.

Depending on the particular situation the type for the domain of attributes (class "AttributeRefType") can, by means of the relationship "ARefOf", refer to a class (class "Class"), a class reference (class

"ClassRefType") or an argument (class "Argument"). All admissible types of the attribute are derived by means of the relationship "ARefRestriction".

2.3.8 Object types

Individual objects or set of objects may be arguments of functions. Its description is effected by means of the object type ("ObjectType"), an extension of the class "ClassRelatedType". The attribute "Set" indicates whether a set of object or an individual object is to be expected.

2.3.9 Units

Units are described by objects of the class Unit. In the case of abstract units, the attribute Name is issued the unit name as its value. With concrete units the attribute Attribut is issued the abbreviation of the unit as attribute. In the interest of translatability the full name is described by a DocText element named "FullName" and containing the full name as text.

The definition of derived, resp. composed units is described by means of an expression. However this may only contain a limited variety of factors (unit references, constants, function calls).

2.4 Generic domains

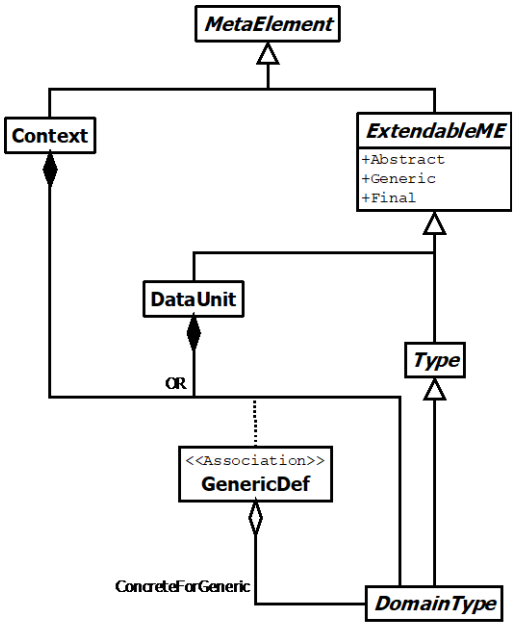

Figure 5: Generic domains

If a domain is generic, this is indicated in the "ExtendableME" super class as Generic = #true. Although in INTERLIS 2, generic domains are only eligible for coordinates, this restriction is waived in the metamodel.

For generic domains, specific domains can be defined in the framework of contexts. For each generic domain for which specific definitions are made, an instance of the relationship "GenericDef" exists, which compositionally belongs to the context instance and refers to the generic domain. The specific domains that are defined via this context are allocated to the relationship instance via the "ConcreteForGeneric" relationship.

To ensure that for a DataUnit it is clear which specific domains apply for generic domains, the same construct is used: for each generic domain that exists in the DataUnit, a relationship instance exists that is compositionally allocated to the DataUnit and refers to the generic domain. The specific domain that apply for this DataUnit on the basis of the corresponding valid contexts are allocated to the relationship instance

via the "ConcreteForGeneric" relationship. If more than one specific domain is defined, the one that applies to the data is only specified in the transfer data.

2.5 Views and graphics

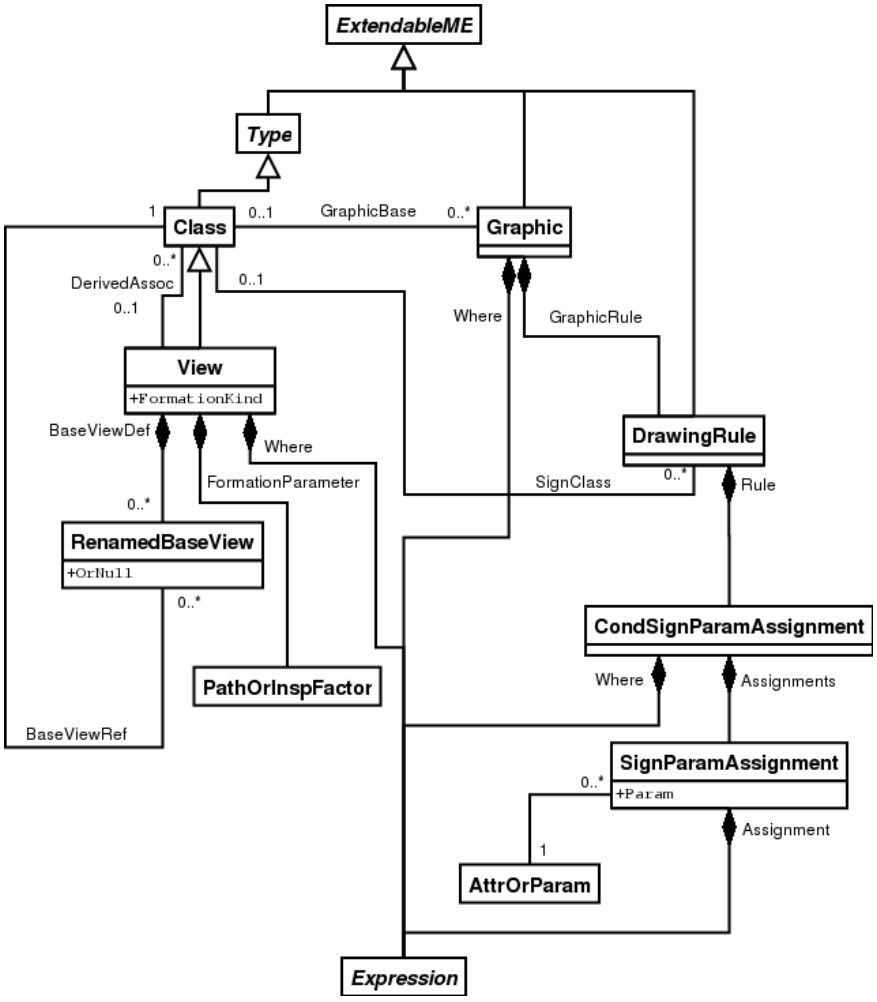

Figure 6: Views and graphics

The description of a view (class "View") is an extension of "Class", which achieves the description of the attributes. Objects of the class View describe how the view is contrived and (via "RenamedBaseView") which base classes make contributions to the view.

Further parameters serving the realization of a view (cf. INTERLIS 2-Reference Manual, chapter 2.15) are only indicated in the case of aggregation (provided EQUAL is requested) and for inspections aiming at the designation of the attribute to be inspected ("FormationParameter"). If a join is a so-called "Outer-Join", the value "true" is assigned to the attribute "OrNull" when defining the base class (class "RenamedBaseView").

In the case of an aggregation the implicit attribute "AGGREGATES" is generated (carrying this name). Its type is a MultiValue whose base type is the base class if the aggregation.

When extending a view we omit repeating definitions of the base class within the extensions, but merely supplement possible definitions of extensions.

As far as possible the graphic description follows the structure of the INTERLIS-language. All constructs necessary for assignation were generated within the scope of factors and expressions. The question,

whether or not these special constructs can also be applied in domains other than that of graphics, remains open.

2.6 Other constructs

2.6.1 Expressions and factors

Factors and Expressions are not defined as individual objects (metaelements), but as structures. The various constructs correspond more or less directly to language constructs.

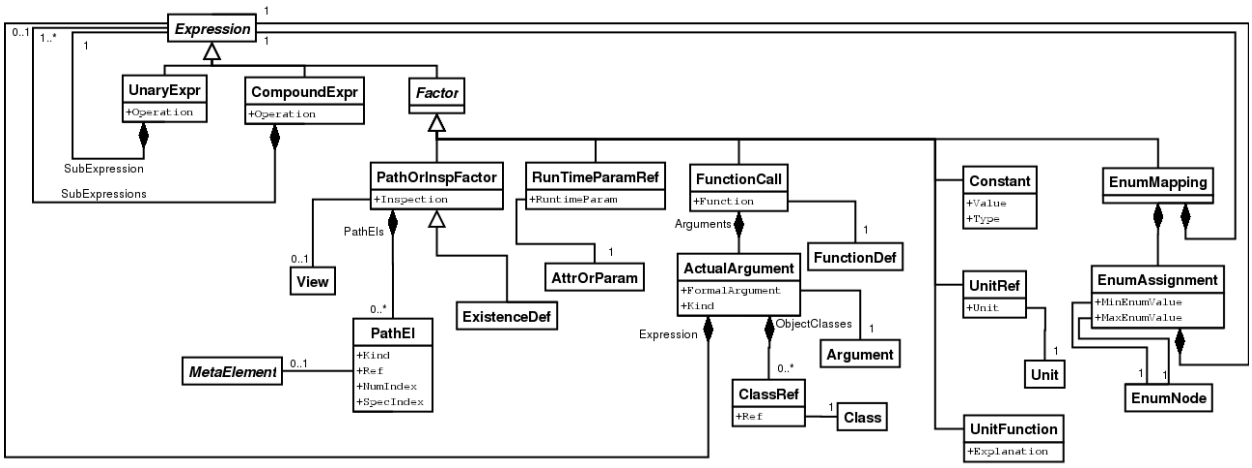

Figure 7: Expressions and factors

In the case of expressions we distinguish between:

- "UnaryExpr": Expressions, where an operation (Not, Defined) refers to a sub-expression.
- "CompoundExpr": Expressions, where an operation (And, Or, Vergleiche, InRange, CondExpr) refers to a set of sub-expressions.
- "Factor": Factors, which establish a reference to model elements.

Special remarks concerning "Factor":

- The unit reference (structure UnitRef) can only occur within the scope of derived and composed units.
- The mapping (structure EnumMapping) of enumeration values onto other values is mainly used within the scope of graphics.

2.6.2 Consistency constraints

Consistency constraints are modelled as sub-classes in the "Constraint" class. This in turn is a sub-class of the "MetaElement" class. Compositionally, instances of consistency constraints belong to the element (class or domain) that specifies them via "ClassConstraint" or "DomainConstraint". The various sub-classes of the "Constraint" class correspond more or less directly to the language constructs.

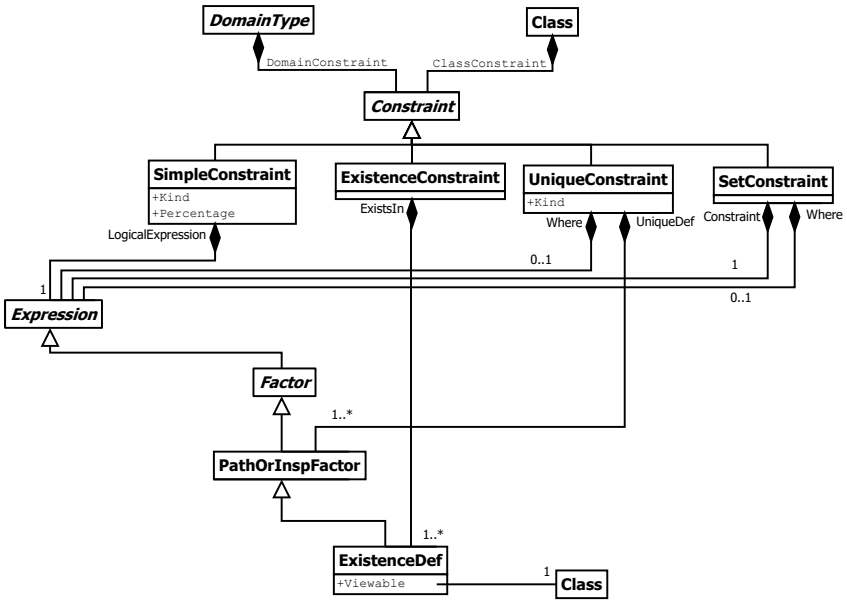

Figure 8: Consistency constraints

2.6.3 Definition of functions

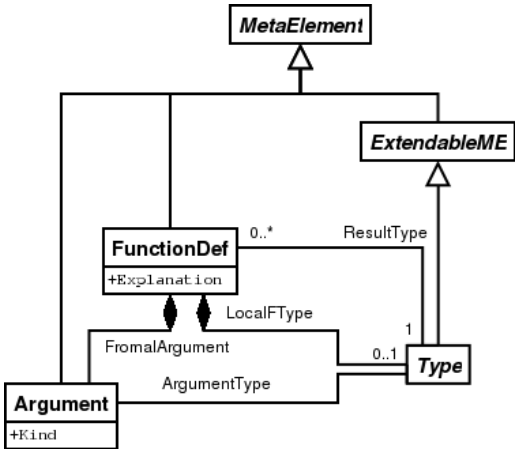

Figure 9: Definition of functions

By means of the relationship "Result" the type of result is assigned to the function definition (class "FunctionDef"). Compositional assignation is applied between arguments and function definition (relation "FormalArgument"). The name path is a result of …FunctionName.ArgumentName. Where there is a local definition of result types or of arguments, there is also compositional assignation to the function definition (relation "LocalFType"). The name path is a result of …FunctionName.TYPE for the result type (since TYPE is a key-word of the language, there can be no such argument name), resp. of …FunctionName.ArgumentName.TYPE.

Arguments and their type are linked via the relationship ArgumentType.

2.6.4 Metaobjects

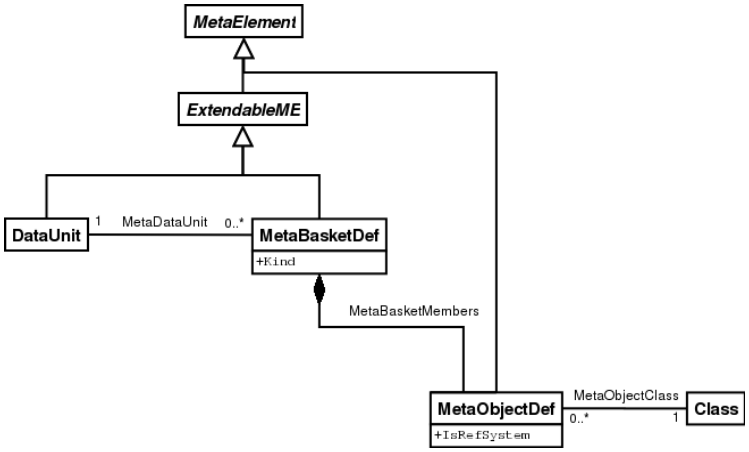

Figure 10: Metaobjects

The classes "MetaDataBasketDef" and "MetaObjectDef" serve to describe meta data baskets and metaobjects.

The name of the description of the meta data baskets corresponds to the basket name. All metaobjects expected in this basket are assigned via the composition MetaBasketMembers.

2.6.5 Run time parameters

Run time parameters (cf. INTERLIS 2-Reference Manual, chapter 2.11) are represented by means of objects of the class AttrOrParam, which are directly assigned to the corresponding package.

3 Sub-dividing model data into baskets

To ensure a simple procedure permitting selective usage of model data, a seperate container (Basket) is generated for every model (original or translation). As a basket identification it contains the constant text "MODEL", followed by a period and the respective model name, thus guaranteeing that the basket identification cannot coincide with an object identification.

4 The INTERLIS-Metamodel in INTERLIS 2

```
INTERLIS 2.4;

/** Base: IlisMeta07 VERSION "2022-04-28"
 *  Modifications for I2.4:
 *  - generics!
 *  - constraints as meta elements
 *  - ...
 */
MODEL IlisMeta16 (en) AT "http://models.interlis.ch" VERSION "2022-06-17" =

  DOMAIN
    BasketOID = OID TEXT;  !! Filename
    MetaElemOID = OID TEXT;  !! Namepath: Model...
    LanguageCode = TEXT*5;


  TOPIC ModelData =
    BASKET OID AS IlisMeta16.BasketOID;
    OID AS IlisMeta16.MetaElemOID;

    DOMAIN
      Code = 0..255;  !! 1 byte code
      MultRange = 0..2147483647;  !! Max Int32
      LengthRange EXTENDS MultRange = 1..2147483647;

    /** MetaElements in general
     */

    STRUCTURE DocText =
      Name: TEXT;
      Text: MANDATORY MTEXT;
    END DocText;

    CLASS MetaElement (ABSTRACT) =
      /** OID: <Parent-OID>.Name
       */
      Name: MANDATORY TEXT;
      Documentation: LIST OF DocText;
    END MetaElement;

    CLASS MetaAttribute =
      /** OID: <Parent-OID>.METAOBJECT.Name
       */
      Name: MANDATORY TEXT;
      Value: MANDATORY TEXT;
    END MetaAttribute;

    ASSOCIATION MetaAttributes =
      MetaElement (EXTERNAL) -<#> MetaElement;
      MetaAttribute -- MetaAttribute;
    END MetaAttributes;

    CLASS ExtendableME (ABSTRACT) EXTENDS MetaElement =
      Abstract: MANDATORY BOOLEAN;
      Generic: MANDATORY BOOLEAN;  !! 2.4
      Final: MANDATORY BOOLEAN;
    END ExtendableME;

    ASSOCIATION Inheritance =
```

```
  Sub -- ExtendableME;
  Super (EXTERNAL) -- {0..1} ExtendableME;
END Inheritance;

/** Models
 */

CLASS Package (ABSTRACT) EXTENDS MetaElement =
END Package;

STRUCTURE Ili1Format =
  isFree: MANDATORY BOOLEAN;
  LineSize: LengthRange;  !! defined when isFree == False
  tidSize: LengthRange;  !! defined when isFree == False
  blankCode: MANDATORY Code;
  undefinedCode: MANDATORY Code;
  continueCode: MANDATORY Code;
  Font: MANDATORY TEXT;
  tidKind: MANDATORY (TID_I16, TID_I32, TID_ANY, TID_EXPLANATION);
  tidExplanation: TEXT;  !! defined when tidKind == TID_EXPLANATION
END Ili1Format;

CLASS Model EXTENDS Package =
  /** MetaElement.Name := ModelName as defined in the INTERLIS-Model
   */
  iliVersion: MANDATORY TEXT;
  Contracted: BOOLEAN;
  Kind: MANDATORY (NormalM, TypeM, RefSystemM, SymbologyM);
  Language: LanguageCode;
  At: TEXT;
  Version: TEXT;
  NoIncrementalTransfer: BOOLEAN;  !! 2.4
  CharSetIANAName: TEXT;  !! 2.4
  xmlns: TEXT;  !! 2.4
  ili1Transfername: TEXT;
  ili1Format: Ili1Format;
END Model;

CLASS SubModel EXTENDS Package =
  /** MetaElement.Name := TopicName as defined in the INTERLIS-Model
   */
END SubModel;

ASSOCIATION PackageElements =
  ElementInPackage -<#> {0..1} Package;
  Element -- MetaElement;
MANDATORY CONSTRAINT
  NOT (INTERLIS.isOfClass(Element, >IlisMeta16.ModelData.Model));
END PackageElements;

ASSOCIATION Import =
  ImportingP -- Package;
  ImportedP (EXTERNAL) -- Package;
END Import;

CLASS Type (ABSTRACT) EXTENDS ExtendableME =
END Type;

STRUCTURE Expression (ABSTRACT) =
END Expression;
```

```
STRUCTURE Multiplicity =
  Min: MANDATORY MultRange;
  Max: MultRange;
END Multiplicity;

CLASS Constraint (ABSTRACT) EXTENDS MetaElement =  !! 2.4
END Constraint;

CLASS DomainType (ABSTRACT) EXTENDS Type =
  /** MetaElement.Name :=
   * DomainName if defined explicitly as a domain,
   * "Type" if defined within an attribute definition
   */
  Mandatory: MANDATORY BOOLEAN;
END DomainType;

ASSOCIATION DomainConstraint =  !! 2.4
  ToDomain -<#> DomainType;
  Constraint -- {0..*} Constraint;
END DomainConstraint;

/** Classes
 */

CLASS Class EXTENDS Type =
  /** MetaElement.Name := StructureName, ClassName,
   *                      AssociationName, ViewName
   *                      as defined in the INTERLIS-Model
   */
  Kind: MANDATORY (Structure, Class, View, Association);
  Multiplicity: Multiplicity;  !! for associations only
  EmbeddedRoleTransfer: MANDATORY BOOLEAN;
  ili1OptionalTable: BOOLEAN;  !! INTERLIS 1 only
END Class;

CLASS AttrOrParam EXTENDS ExtendableME =
  /** MetaElement.Name := AttributeName, ParameterName
   *                      as defined in the INTERLIS-Model
   */
  SubdivisionKind: (NoSubDiv, SubDiv, ContSubDiv);
  Transient: BOOLEAN;
  Derivates: LIST OF Expression;
END AttrOrParam;

ASSOCIATION ClassConstraint =  !! 2.4
  ToClass -<#> Class;
  Constraint -- {0..*} Constraint;
END ClassConstraint;

ASSOCIATION LocalType =
  LTParent -<#> AttrOrParam;
  LocalType -- {0..*} Type;
END LocalType;

ASSOCIATION AttrOrParamType =
  AttrOrParam -- AttrOrParam;
  Type (EXTERNAL) -- {1} Type;
END AttrOrParamType;

ASSOCIATION ClassAttr =
  AttrParent -<#> Class;
```

```
  ClassAttribute (ORDERED) -- AttrOrParam;
MANDATORY CONSTRAINT DEFINED(ClassAttribute->SubdivisionKind) AND
                     DEFINED(ClassAttribute->Transient);
END ClassAttr;

ASSOCIATION ClassParam =
  ParamParent -<#> Class;
  ClassParameter (ORDERED) -- AttrOrParam;
MANDATORY CONSTRAINT NOT (DEFINED(ClassParameter->SubdivisionKind) OR
                          DEFINED(ClassParameter->Transient));
END ClassParam;

/** Types related to other types
 */

CLASS TypeRelatedType (ABSTRACT) EXTENDS DomainType =
END TypeRelatedType;

ASSOCIATION BaseType =
  TRT -- TypeRelatedType;
  BaseType (EXTERNAL) -- {1} Type;
END BaseType;

ASSOCIATION TypeRestriction =
  TRTR -- TypeRelatedType;
  TypeRestriction (EXTERNAL) -- Type;
END TypeRestriction;

/** Bag type
 */

CLASS MultiValue EXTENDS TypeRelatedType =
  /** MetaElement.Name := "Type" because always defined
   *                      within an attribute definition
   */
  Ordered: MANDATORY BOOLEAN;
  Multiplicity: Multiplicity;
END MultiValue;

/** References and associations
 */

CLASS ClassRelatedType (ABSTRACT) EXTENDS DomainType =
END ClassRelatedType;

ASSOCIATION BaseClass =
  CRT -- ClassRelatedType;
  BaseClass (EXTERNAL) -- Class;
END BaseClass;

ASSOCIATION ClassRestriction =
  CRTR -- ClassRelatedType;
  ClassRestriction (EXTERNAL) -- Class;
END ClassRestriction;

CLASS ReferenceType EXTENDS ClassRelatedType =
  External: MANDATORY BOOLEAN;
END ReferenceType;

CLASS Role EXTENDS ReferenceType =
  /** MetaElement.Name := RoleName as defined in the INTERLIS-Model
```

```
   */
  Strongness: MANDATORY (Assoc, Aggr, Comp);
  Ordered: MANDATORY BOOLEAN;
  Multiplicity: Multiplicity;
  Derivates: LIST OF Expression;
  EmbeddedTransfer: MANDATORY BOOLEAN;
END Role;

ASSOCIATION AssocRole =
  Association -<#> Class;
  Role (ORDERED) -- Role;
MANDATORY CONSTRAINT Association->Kind == #Association;
END AssocRole;

CLASS ExplicitAssocAccess EXTENDS ExtendableME =
END ExplicitAssocAccess;

ASSOCIATION ExplicitAssocAcc =
  AssocAccOf -<#> Class;
  ExplicitAssocAcc (ORDERED) -- ExplicitAssocAccess;
END ExplicitAssocAcc;

ASSOCIATION AssocAccOrigin =
  UseAsOrigin -- ExplicitAssocAccess;
  OriginRole -- {1} Role;
END AssocAccOrigin;

ASSOCIATION AssocAccTarget =
  UseAsTarget -- ExplicitAssocAccess;
  TargetRole -- {0..1} Role;
END AssocAccTarget;

ASSOCIATION AssocAcc =
  Class -- Class;
  AssocAcc -- Role OR ExplicitAssocAccess;
END AssocAcc;

/** Information for easy transfer
 */

ASSOCIATION TransferElement =
  TransferClass -- Class;
  TransferElement (EXTERNAL, ORDERED) -- AttrOrParam OR
                                         ExplicitAssocAccess OR
                                         Role;
END TransferElement;

ASSOCIATION Ili1TransferElement =
  Ili1TransferClass -- Class;
  Ili1RefAttr (ORDERED) -- AttrOrParam OR Role;
END Ili1TransferElement;

/** DataUnits
 */

CLASS DataUnit EXTENDS ExtendableME =
  Name (EXTENDED): TEXT := "BASKET";
  ViewUnit: MANDATORY BOOLEAN;
  DataUnitName: MANDATORY TEXT;
END DataUnit;
```

```
    ASSOCIATION Dependency =
      Using -- DataUnit;
      Dependent (EXTERNAL) -- DataUnit;
    END Dependency;

    ASSOCIATION AllowedInBasket =
      OfDataUnit -- DataUnit;
      ClassInBasket (EXTERNAL) -- Class;
    END AllowedInBasket;

    /** Generics and Contexts
     */  !! 2.4

    CLASS Context EXTENDS MetaElement =
      /** MetaElement.Name := ContextName as defined in the INTERLIS-Model
       */
    END Context;

    ASSOCIATION GenericDef =
       OID AS MetaElemOID;  !! <Context-OID>.<GenericDomain->Name>
       Context -<#> Context OR DataUnit;
       GenericDomain -- DomainType;
    MANDATORY CONSTRAINT
       GenericDomain->Generic;
    END GenericDef;

    ASSOCIATION ConcreteForGeneric =
       GenericDef -<> GenericDef;
       ConcreteDomain -- DomainType;
    MANDATORY CONSTRAINT
       NOT (ConcreteDomain->Abstract) AND NOT (ConcreteDomain->Generic);
    END ConcreteForGeneric;

    /** Units
     */

    CLASS Unit EXTENDS ExtendableME =
      /** MetaElement.Name := ShortName as defined in the INTERLIS-Model
       */
      Kind: MANDATORY (BaseU, DerivedU , ComposedU);
      Definition: Expression;
    MANDATORY CONSTRAINT ((Kind != #BaseU) == DEFINED(Definition));
    END Unit;

    /** MetaObjects
     */

    CLASS MetaBasketDef EXTENDS ExtendableME =
      /** MetaElement.Name := BasketName as defined in the INTERLIS-Model
       */
      Kind: MANDATORY (SignB, RefSystemB);
    END MetaBasketDef;

    ASSOCIATION MetaDataUnit =
      MetaBasketDef -- MetaBasketDef;
      MetaDataTopic (EXTERNAL) -- {1} DataUnit;
    END MetaDataUnit;

    CLASS MetaObjectDef EXTENDS MetaElement =
      /** MetaElement.Name := MetaObjectName as defined in the INTERLIS-
Model
```

```
   */
  IsRefSystem: MANDATORY BOOLEAN;
END MetaObjectDef;

ASSOCIATION MetaBasketMembers =
  MetaBasketDef -<#>  MetaBasketDef;
  Member -- MetaObjectDef;
END MetaBasketMembers;

ASSOCIATION MetaObjectClass =
  MetaObjectDef --  MetaObjectDef;
  Class -- {1} Class;
END MetaObjectClass;

/** Base types
 */

CLASS BooleanType EXTENDS DomainType =
END BooleanType;

CLASS TextType EXTENDS DomainType =
  Kind: MANDATORY (MText, Text, Name, Uri);
  MaxLength: LengthRange;
END TextType;

CLASS BlackboxType EXTENDS DomainType =
  Kind: MANDATORY (Binary, Xml);
END BlackboxType;

CLASS NumType EXTENDS DomainType =
  /** MetaElement.Name :=
   *  DomainName if defined explicitly as a domain,
   *  "Type" if defined within an attribute definition,
   *  "C1", "C2", "C3" if defined within a coordinate type
   */
  Min: TEXT;
  Max: TEXT;
  Circular: BOOLEAN;
  Clockwise: BOOLEAN;
  /** Constraint, that Clockwise or Refsys
   */
END NumType;

ASSOCIATION NumUnit =
  Num -- NumType;
  Unit (EXTERNAL) -- {0..1} Unit;
END NumUnit;

DOMAIN
  AxisInd = 1..3;

CLASS CoordType EXTENDS DomainType =
  NullAxis: AxisInd;
  PiHalfAxis: AxisInd;
  Multi: BOOLEAN;  !! 2.4
END CoordType;

ASSOCIATION AxisSpec =
  CoordType -- CoordType;
  Axis (ORDERED, EXTERNAL) -- {1..3} NumType;
END AxisSpec;
```

```
ASSOCIATION NumsRefSys =
  NumType -- NumType;
  RefSys (EXTERNAL) -- {0..1} MetaObjectDef OR CoordType;
  Axis: AxisInd;
END NumsRefSys;

CLASS FormattedType EXTENDS NumType =
  Format: MANDATORY TEXT;
END FormattedType;

ASSOCIATION StructOfFormat =
  FormattedType -- FormattedType;
  Struct (EXTERNAL) -- {1} Class;
END StructOfFormat;

/** OID Definition
 */

CLASS AnyOIDType EXTENDS DomainType =
END AnyOIDType;

ASSOCIATION ObjectOID =
  Class -- Class;
  Oid (EXTERNAL) -- {0..1} DomainType  !! No OID, if no assoc
                           RESTRICTION (TextType; NumType; AnyOIDType);
END ObjectOID;

ASSOCIATION BasketOID =
  ForDataUnit -- DataUnit;
  Oid (EXTERNAL) -- {0..1} DomainType
                           RESTRICTION (TextType; NumType; AnyOIDType);
END BasketOID;

/** Functions
 */

CLASS FunctionDef EXTENDS MetaElement =
  /** MetaElement.Name := FunctionName as defined in the INTERLIS-Model
   */
  Explanation: TEXT;
END FunctionDef;

ASSOCIATION LocalFType =
  LFTParent -<#> FunctionDef;
  LocalType -- {0..1} Type;
END LocalFType;

ASSOCIATION ResultType =
  Function -- FunctionDef;
  ResultType (EXTERNAL) -- {1} Type;
END ResultType;

CLASS Argument EXTENDS MetaElement =
  /** MetaElement.Name := ArgumentName as defined in the INTERLIS-Model
   */
  Kind: MANDATORY (Type, EnumVal, EnumTreeVal);
END Argument;

ASSOCIATION FormalArgument =
  Function -<#> FunctionDef;
```

```
  Argument (ORDERED) -- Argument;
END FormalArgument;

ASSOCIATION ArgumentType =
  Argument -- Argument;
  Type (EXTERNAL) -- {0..1} Type;
MANDATORY CONSTRAINT (Argument->Kind == #Type);
END ArgumentType;

/** Class and attribute reference types
 */

CLASS ClassRefType EXTENDS ClassRelatedType =
END ClassRefType;

CLASS ObjectType EXTENDS ClassRelatedType =
  Multiple: MANDATORY BOOLEAN;
END ObjectType;

CLASS AttributeRefType EXTENDS DomainType =
END AttributeRefType;

ASSOCIATION ARefOf =
  ForARef -- AttributeRefType;
  Of (EXTERNAL) -- {0..1} Class OR AttrOrParam OR Argument;
END ARefOf;

ASSOCIATION ARefRestriction =
  ARef -- AttributeRefType;
  Type (EXTERNAL) -- Type;
END ARefRestriction;

/** Enumerations
 */

CLASS EnumType EXTENDS DomainType =
  Order: MANDATORY (Unordered, Ordered, Circular);
END EnumType;

CLASS EnumNode EXTENDS ExtendableME =
  /** MetaElement.Name := "TOP" for topnode,
   *                      enumeration value (without constant prefix #)
   *                      for all real nodes
   */
END EnumNode;

ASSOCIATION TopNode =
  EnumType -<#> EnumType;
  TopNode (ORDERED) -- {1} EnumNode;
END TopNode;

ASSOCIATION SubNode =
  ParentNode -<#> EnumNode;
  Node (ORDERED) -- EnumNode;
END SubNode;

/** Extended enumerations have complete definitions.
 *  Inherited nodes refer to base node
 */

CLASS EnumTreeValueType EXTENDS DomainType =
```

```
END EnumTreeValueType;

ASSOCIATION TreeValueTypeOf =
  ETVT -- EnumTreeValueType;
  ET (EXTERNAL) -- {1} EnumType;
END TreeValueTypeOf;

/** Line types
 */

CLASS LineForm EXTENDS MetaElement =
  /** MetaElement.Name := LineFormName as defined in the INTERLIS-Model
   */
END LineForm;

ASSOCIATION LineFormStructure =
  LineForm -- LineForm;
  Structure (EXTERNAL) -- {1} Class;
MANDATORY CONSTRAINT (Structure->Kind == #Structure);
END LineFormStructure;

CLASS LineType EXTENDS DomainType =
  Kind: MANDATORY (Polyline, DirectedPolyline, Surface, Area);
  MaxOverlap: TEXT;
  Multi: BOOLEAN;  !! 2.4
END LineType;

ASSOCIATION LinesForm =
  LineType -- LineType;
  LineForm (EXTERNAL) -- {1..*} LineForm;
END LinesForm;

ASSOCIATION LineCoord =
  LineType -- LineType;
  CoordType (EXTERNAL) -- {0..1} CoordType;
END LineCoord;

ASSOCIATION LineAttr =
  LineType -- LineType;
  LAStructure (EXTERNAL) -- {0..1} Class;
END LineAttr;

/** Views
 */

CLASS View EXTENDS Class =
  FormationKind: MANDATORY (Projection, Join, Union,
                            Aggregation (All, Equal),
                            Inspection (Normal, Area));
  FormationParameter: LIST OF Expression;  !! PathOrInspFactor only
                      /** Aggr.Equal: UniqueEl
                       *  Inspection: Attributepath
                       */
  Where: Expression;
  Transient: MANDATORY BOOLEAN;
END View;

CLASS RenamedBaseView EXTENDS ExtendableME =
  /** MetaElement.Name := Name as defined in the INTERLIS-Model
   */
  OrNull: BOOLEAN;
```

```
END RenamedBaseView;

ASSOCIATION BaseViewDef =
  View -<#> View;
  RenamedBaseView (ORDERED) -- RenamedBaseView;
END BaseViewDef;

ASSOCIATION BaseViewRef =
  RenamedBaseView -- RenamedBaseView;
  BaseView (EXTERNAL) -- {1} Class;
END BaseViewRef;

ASSOCIATION DerivedAssoc =
  DeriAssoc -- Class;
  View (EXTERNAL) -- {0..1} View;
MANDATORY CONSTRAINT
  (DeriAssoc->Kind == #Association) AND (View->Kind == #View);
END DerivedAssoc;

/** Expressions, factors
 */

STRUCTURE UnaryExpr EXTENDS Expression =
  Operation: (Not, Defined);
  SubExpression: Expression;
END UnaryExpr;

STRUCTURE CompoundExpr EXTENDS Expression =
  Operation: (Implication, And, Or, Mult, Div,  !! 2.4: Implication
              Relation (Equal, NotEqual,
              LessOrEqual, GreaterOrEqual,
              Less, Greater));
  SubExpressions: LIST OF Expression;
END CompoundExpr;

STRUCTURE Factor (ABSTRACT) EXTENDS Expression =
END Factor;

STRUCTURE PathEl =
  Kind: (This, ThisArea, ThatArea, Parent,
         ReferenceAttr, AssocPath, Role, ViewBase,
         Attribute, MetaObject) ORDERED;
  Ref: REFERENCE TO (EXTERNAL) MetaElement;
  NumIndex: MultRange;
  SpecIndex: (First, Last);
MANDATORY CONSTRAINT (Kind >= #ReferenceAttr) == DEFINED(Ref);
END PathEl;

STRUCTURE PathOrInspFactor EXTENDS Factor =
  PathEls: LIST OF PathEl;
  Inspection: REFERENCE TO (EXTERNAL) View;
END PathOrInspFactor;

STRUCTURE EnumAssignment =
  ValueToAssign: Expression;
  MinEnumValue: MANDATORY REFERENCE TO (EXTERNAL) EnumNode;
  MaxEnumValue: REFERENCE TO (EXTERNAL) EnumNode;
END EnumAssignment;

STRUCTURE EnumMapping EXTENDS Factor =
  EnumValue: PathOrInspFactor;
```

```
  Cases: LIST OF EnumAssignment;
END EnumMapping;

STRUCTURE ClassRef =
  Ref: MANDATORY REFERENCE TO (EXTERNAL) Class;
END ClassRef;

STRUCTURE ActualArgument =
  FormalArgument: MANDATORY REFERENCE TO (EXTERNAL) Argument;
  Kind: MANDATORY (Expression, AllOf);
  Expression: BAG {0..1} OF Expression;
  ObjectClasses: BAG {0..*} OF ClassRef;
END ActualArgument;

STRUCTURE FunctionCall EXTENDS Factor =
  Function: MANDATORY REFERENCE TO (EXTERNAL) FunctionDef;
  Arguments: LIST OF ActualArgument;
END FunctionCall;

STRUCTURE RuntimeParamRef EXTENDS Factor =
  RuntimeParam: MANDATORY REFERENCE TO
                          IlisMeta16.ModelData.AttrOrParam;
END RuntimeParamRef;

STRUCTURE Constant EXTENDS Factor =
  Value: MANDATORY TEXT;
  Type: MANDATORY (Undefined, Numeric, Text, Enumeration);
END Constant;

STRUCTURE ClassConst EXTENDS Factor =
  Class: REFERENCE TO (EXTERNAL) Class;
END ClassConst;

STRUCTURE AttributeConst EXTENDS Factor =
  Attribute: REFERENCE TO (EXTERNAL) AttrOrParam;
END AttributeConst;

STRUCTURE UnitRef EXTENDS Factor =
  Unit: REFERENCE TO (EXTERNAL) IlisMeta16.ModelData.Unit;
END UnitRef;

STRUCTURE UnitFunction EXTENDS Factor =
  Explanation: MANDATORY TEXT;
END UnitFunction;

/** Constraints
 */

CLASS SimpleConstraint EXTENDS Constraint =
  Kind: (MandC, LowPercC, HighPercC);
  Percentage: 0.00 .. 100.00;
  LogicalExpression: Expression;
END SimpleConstraint;

CLASS ExistenceConstraint EXTENDS Constraint =
  Attr : MANDATORY PathOrInspFactor;
END ExistenceConstraint;

ASSOCIATION ExistenceDef =   !! 2.4
  ExistenceConstraint -<#> ExistenceConstraint;
  ExistsIn -- Class;
```

```
  END ExistenceDef;

  CLASS UniqueConstraint EXTENDS Constraint =
    Where: BAG {0..1} OF Expression;
    Kind: MANDATORY (GlobalU, BasketU, LocalU);  !! 2.4: BasketU
    UniqueDef: LIST {1..*} OF PathOrInspFactor;
  END UniqueConstraint;

  CLASS SetConstraint EXTENDS Constraint =
    Where: BAG {0..1} OF Expression;
    Constraint: Expression;
  END SetConstraint;

  /** Graphic
   */

  CLASS Graphic EXTENDS ExtendableME =
    /** MetaElement.Name := Name as defined in the INTERLIS-Model
     */
    Where: Expression;
  END Graphic;

  ASSOCIATION GraphicBase =
    Graphic -- Graphic;
    Base (EXTERNAL) -- {0..1} Class;
  END GraphicBase;

  STRUCTURE SignParamAssignment =
    Param: MANDATORY REFERENCE TO (EXTERNAL) AttrOrParam;
    Assignment: Expression;
  END SignParamAssignment;

  STRUCTURE CondSignParamAssignment =
    Where: Expression;
    Assignments: LIST OF SignParamAssignment;
  END CondSignParamAssignment;

  CLASS DrawingRule EXTENDS ExtendableME =
    /** MetaElement.Name := Name as defined in the INTERLIS-Model
     */
    Rule: LIST OF CondSignParamAssignment;
  END DrawingRule;

  ASSOCIATION GraphicRule =
    Graphic -<#> Graphic;
    DrawingRule -- DrawingRule;
  END GraphicRule;

  ASSOCIATION SignClass =
    DrawingRule -- DrawingRule;
    Class -- {0..1} Class;
  END SignClass;

END ModelData;


TOPIC ModelTranslation =
  DEPENDS ON IlisMeta16.ModelData;

  STRUCTURE DocTextTranslation =
    Text: MANDATORY MTEXT;
```

```
      END DocTextTranslation;

      STRUCTURE METranslation =
        Of: MANDATORY REFERENCE TO (EXTERNAL)
                      IlisMeta16.ModelData.MetaElement;
        TranslatedName: TEXT;
        TranslatedDoc: LIST OF DocTextTranslation;
      END METranslation;

      CLASS Translation =
        NO OID;
        Language: MANDATORY LanguageCode;
        Translations: BAG OF METranslation;
      END Translation;

    END ModelTranslation;

  END IlisMeta16.
```