Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement für Verteidigung,
Bevölkerungsschutz und Sport VBS

Bundesamt für Landestopographie swisstopo

INTERLIS 2 –

Metamodell

Ausgabe vom 2022-06-17 (deutsch)

Informationen und Kontakt: www.interlis.ch, info@interlis.ch

Inhalt

1 Einleitung

1.1 Begriff und Zweck

Unter Metamodell wird das Datenmodell verstanden, das beschreibt, wie Modellbeschreibungen modelliert sind. Daten, die einem Metamodell entsprechen und somit ein Datenmodell beschreiben, werden Modelldaten genannt.

Modelldaten haben grundsätzlich denselben Beschreibungsinhalt wie Modellbeschreibungen in der INTERLIS-Sprache. Ihre Struktur und Codierung orientieren sich aber nicht nach Beschreibungseleganz und Lesbarkeit für Menschen, sondern nach möglichst einfacher Verwendung in Programmen, z.B.:

- Generische Anwendungsprogramme, deren Funktionieren zum Teil von der Modellinformation abhängt.
- Generatoren von Programmteilen gemäss Modellinformationen.
- Werkzeuge, die Modellinformationen in irgendeiner Weise konvertieren (z.B. in andere Standards).

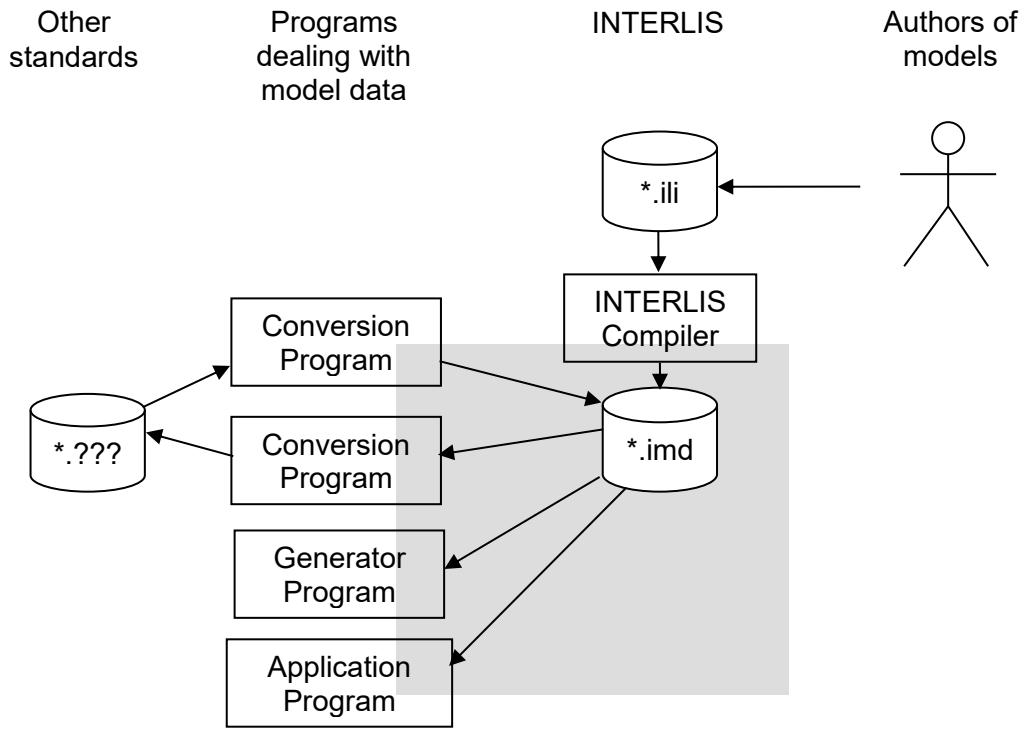

Abbildung 1: Relevanz von Modelldaten

Entsprechend richtet sich dieses Dokument nicht an Personen, die INTERLIS-Modelle entwickeln, sondern an diejenigen, welche Programme schreiben, die Modellinformationen ausnützen bzw. INTERLIS-Modelldaten erzeugen (INTERLIS-Compiler oder Konversionsprogramme aus anderen Standards).

1.2 Verhältnis zu anderen Normen

Metamodelle und entsprechende Modelldaten gibt es ausser für INTERLIS auch für andere Datenbeschreibungstechniken (z.B. UML-MOF, XML-Schema). Es stellte sich damit die Frage, ob ein Metamodell einer solchen Technik übernommen werden soll. Dies wurde verworfen, weil die Differenzen meist erheblich sind und die Nähe zur INTERLIS-Beschreibungssprache verloren gegangen wäre. Andere

Beschreibungstechniken können aber dennoch genutzt werden: Aus Modelldaten anderer Techniken können INTERLIS-Modelldaten erzeugt werden und umgekehrt. Eine solche Umsetzung kann allenfalls auch der INTERLIS-Spezifikation hinzugefügt werden. Zudem könnte es Sinn machen, für bestimmte Standards (z.B. GML) Basismodelle in INTERLIS 2 zu erstellen, welche die Grundstrukturen des Fremdstandards in INTERLIS enthalten.

1.3 Zusammenhang mit der INTERLIS-Sprache

1.3.1 Vollständigkeit

Mit der INTERLIS-Sprache (vgl. INTERLIS 2-Referenzhandbuch) sind verschiedene semantische Regelungen verbunden.

Da sich das Metamodell an der Nutzung der Modelldaten in Programmen und nicht an der korrekten Erzeugung der Modelldaten orientiert, sind einerseits die semantischen Regelungen nicht vollständig (z.B. als Konsistenzbedingungen) im Metamodell umgesetzt. Andererseits sind verschiedene Sachverhalte redundant ausgewiesen (z.B. werden verschiedentlich nicht nur die eigenen, sondern auch geerbte Eigenschaften mittels direkter Unterelemente dargestellt).

Daten im INTERLIS 2-Metamodell (kurz Modelldaten; *.imd) stellen somit nie die primäre Beschreibung des Datenmodells dar. Diese ist typischerweise in textueller Form (*.ili) verfasst, durch eine andere Modellierungstechnik gegeben oder durch einen Modell-Editor (z.B. der UML/INTERLIS-Editor) verwaltet. Ein solches primäres Datenmodell kann durchaus mehr Information (insbesondere Information für die Darstellung des Modells in Diagrammen) als das INTERLIS 2-Metamodell enthalten.

1.3.2 Metaobjekte

Die INTERLIS 2 – Sprache spricht u.a. von Metaobjekten (vgl. INTERLIS 2-Referenzhandbuch, Kap. 2.10). Diese Metaobjekte beschreiben aber Referenzsysteme und Signaturen und dürfen darum nicht mit Objekten der Modelldaten verwechselt werden. Um die Gefahr der Verwechslung zu reduzieren, werden die Objekte der INTERLIS-Modelldaten als Metaelemente bezeichnet.

1.4 Lesehinweise

Die verschiedenen Beschreibungen zum Metamodell setzen die Kenntnis der INTERLIS-Konstrukte (Modelle, Topics, Klassen, etc.) voraus, wie sie im Referenzhandbuch für INTERLIS 2 definiert sind.

Detailbeschreibungen setzen immer auch voraus, dass das Metamodell in Form der INTERLIS 2-Beschreibung gemäss Kap. 4 einbezogen wird.

In den UML-Diagrammen werden Klassen und Strukturen gleich dargestellt. Die Verwendung einer Struktur in einer anderen Struktur oder Klasse wird als Komposition dargestellt, wobei – im Gegensatz zu Beziehungen zwischen Klassen – der Attributname neben dem gefüllten Rhombus angeschrieben wird.

Referenzattribute werden ähnlich wie Beziehungen – als Verbindung zwischen der Ursprungs- und der Zielklasse - dargestellt. Bei der Ursprungsklasse wird das Referenzattribut als Attribut aufgeführt und die Beziehungslinie zu diesem Attribut geführt.

2 Das INTERLIS-Metamodell

2.1 Grundlegende Konstrukte

2.1.1 Übersicht

Abbildung 2 zeigt eine Übersicht über das INTERLIS-Metamodell in Form eines UML-Klassendiagramms. Die Daten, welche die eigentlichen Modelle beschreiben und diejenigen, die sich aus Übersetzungen ergeben, sind dabei klar getrennt. Die einzelnen Klassen und deren Beziehungen unter einander werden in den folgenden Abschnitten beschrieben.

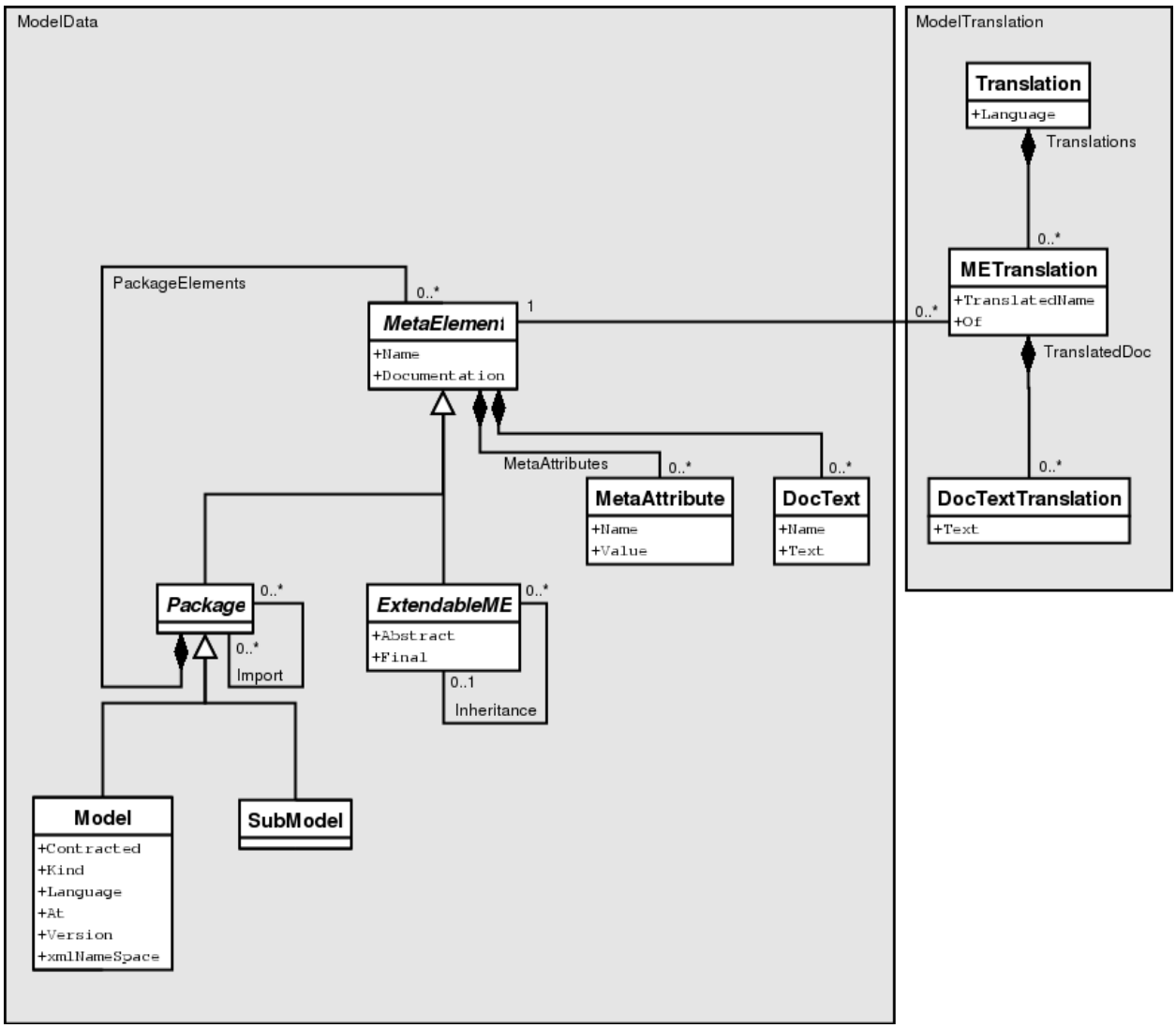

Abbildung 2: Übersicht grundlegende Konstrukte

2.1.2 Klasse MetaElement

Die abstrakte Klasse MetaElement ist die grundlegende Objektklasse für alle Modelldaten. Um Verwechslungen mit den MetaObjekten im Sinne der Datenbeschreibungssprache (vgl. INTERLIS 2-Referenzhandbuch, Kap. 2.10) zu vermeiden, wird hier von Metaelementen statt Metaobjekten gesprochen.

Die Objektidentifikation eines Metaelementes besteht aus einem Namenspfad, der die Namen der übergeordneten Metaelemente und schliesslich den eigenen Namen (Attribut "Name") enthält. Die Namen sind jeweils durch einen Punkt abgetrennt. Damit wird erreicht, dass die Objektidentifikation auch bei

mehrmaliger Compilation (selbst bei Modelländerungen, soweit sie die Stellung dieses Metaelementes nicht betreffen) stabil bleibt.

Beispiel: Das interne INTERLIS-Datenmodell (vgl. INTERLIS 2-Referenzhandbuch, Anhang A) enthält die Struktur UTC mit dem Attribut Hours. Das Metaelement dieser Attributbeschreibung hat somit die OID "INTERLIS.UTC.Hours".

Ein Metaelement enthält eine (allenfalls leere) Liste von Dokumentationstexten (Klasse "DocText"). Jeder Dokumentationstext kann durch einen Namen gekennzeichnet sein und enthält den dokumentierenden Text. Wie die Dokumentationstexte zustande kommen, ist durch INTERLIS nicht geregelt. Dies ist vielmehr Sache des Werkzeuges das die Modelldaten erzeugt.

Einem Metaelement können kompositorisch Metaattribute zugeordnet sein (Klasse "MetaAttribut"). Metaattribute sind weder durch die Sprache INTERLIS noch durch das Metamodell genauer definiert. Sie dienen insbesondere dazu, dass Informationen, die über INTERLIS hinausgehen, als Bestandteil der Modelldaten beigefügt werden können. Metaattribute haben einen Namen (Attribut "Name"), der innerhalb der Metaattribute desselben Metaelementes eindeutig sein muss, und den beschreibenden Wert (Attribut "Value"). Die Objektidentifikation eines Metaattributes besteht immer aus der OID des Metaelementes ergänzt um den festen Namen METAOBJECT (ist in INTERLIS 2 ein Schlüsselwort) und den eigenen Namen. Wäre zum obigen Beispiel ein Metaattribut mit dem Namen X definiert, hiesse die OID "INTERLIS.UTC.Hours.METAOBJECT.X". Metaattribute müssen nicht im selben Behälter wie das Metaelement enthalten sein. Insbesondere ist es möglich, INTERLIS-Behälter (vgl. INTERLIS 2-Referenzhandbuch, Kap. 1.4.5) zu definieren, die nur Metaattribute zu verschiedenen Metaelementen enthalten. Darum ist die Rolle MetaElement in der Beziehung MetaAttribute als extern bezeichnet.

2.1.3 Umgang mit Erweiterungen

Verschiedene INTERLIS 2-Konstrukte können im Sinne des objektorientierten Paradigmas erweitert werden (z.B. Themen, Klassen, Attribute). Solche Konstrukte sind immer direkte oder indirekte Subklassen der abstrakten Grundklasse "ExtendableME" (mit den Attributen Abstract, Generic und Final). Die Beziehung Inheritance beschreibt den Zusammenhang zwischen dem zugrundeliegenden Metaelement (super) und dem erweiternden Metaelement (sub). Da das zugrundeliegende Metaelement in einem anderen Behälter definiert sein kann, ist die Rolle Super als external bezeichnet.

2.1.4 Modelle und Topics

Damit das INTERLIS-Metamodell dem UML-Metamodell möglichst ähnlich ist, werden Modelle und Topics im Metamodell etwas anders dargestellt als sie in der INTERLIS-Sprache definiert sind.

Analog zu UML bildet die abstrakte Klasse "Package" die Grundlage für Modelle und Untermodelle. Über die Beziehung "PackageElements" kann ein Paket – abgesehen von Modellen – beliebige weitere Metaelemente enthalten. Ein Submodell (Klasse "SubModel") kann insbesondere auch weitere Submodelle enthalten, auch wenn dies mit der aktuellen Sprache (INTERLIS 2.4) nicht definierbar ist. Ein Paket ist mit den importierten Paketen über die "Import"-Beziehung verbunden.

Mit Metaelementen der Klasse "DataUnit" wird beschrieben, wie Dateneinheiten (also insbesondere die Behälter eines INTERLIS-Datenaustausches) aufgebaut sein können und wie sie voneinander abhängig sind. Die im Transfer zu verwendende Typenbezeichnung (XML-Tag) ist dem Attribut "DataUnitName" zu entnehmen. Aus Sicht des Metamodells ist nur festgelegt, dass Dateneinheiten innerhalb eines Paketes definiert sein müssen. Für die aktuelle Sprachversion INTERLIS 2.4 gelten jedoch folgende Regelungen:

- Ein INTERLIS-Modell wird durch ein Metaelement der Klasse "Model" dargestellt. Es enthält als Attribute alle Informationen, die gemäss den sprachlichen Möglichkeiten nötig sind. Im Hinblick auf den XML-Transfer ist das Attribut xmlns definiert. Für INTERLIS hat es den Wert http://www.interlis.ch/INTERLIS2.4.

- Ein "Topic" der INTERLIS-Sprache wird durch je ein Metaelement der Klassen "SubModel" und "DataUnit" dargestellt. Mit der Klasse SubModel kann ein Topic als Package im Sinne von UML betrachtet werden; innerhalb dieses Packages beschreibt das Metaelement "DataUnit" (Name: "BASKET", DataUnitName: Modellname"."Topicname), wie entsprechende INTERLIS-Datenbehälter ("Basket") aufgebaut sind (vgl. Kap. 2.2.4).

Da in INTERLIS2.4 das Konstrukt CONTRACTED aufgegeben wurde, ist das entsprechende Attribut aus Kompatibilitätsgründen zwar noch vorhanden aber nicht mehr obligatorisch.

2.1.5 Übersetzungen

INTERLIS 2-Modelle können übersetzt werden (TRANSLATION OF). Die Metadaten beziehen sich jedoch immer auf die Originalmodelle. Selbst wenn ein Modell ein übersetztes Modell importiert, beziehen sich sämtliche Verweise (Beziehungen, Referenzen) auf die Metaelemente der jeweiligen Originalsprache.

Für Übersetzungen werden eigene Behälter gebildet (Metamodell-Topic "ModelTranslation"). Objekte der Klasse "Translation" beschreiben jeweils die Sprache der Übersetzung und enthalten die Metaelement-Übersetzungen (Struktur "METranslation"). Diese beziehen sich auf ein bestimmtes Metaelement (in der Originalsprache) und enthalten den übersetzten Namen sowie nötigenfalls die übersetzten Kommentare in der gleichen Reihenfolge wie die Originalkommentare.

2.2 Hauptkonstrukte

2.2.1 Übersicht

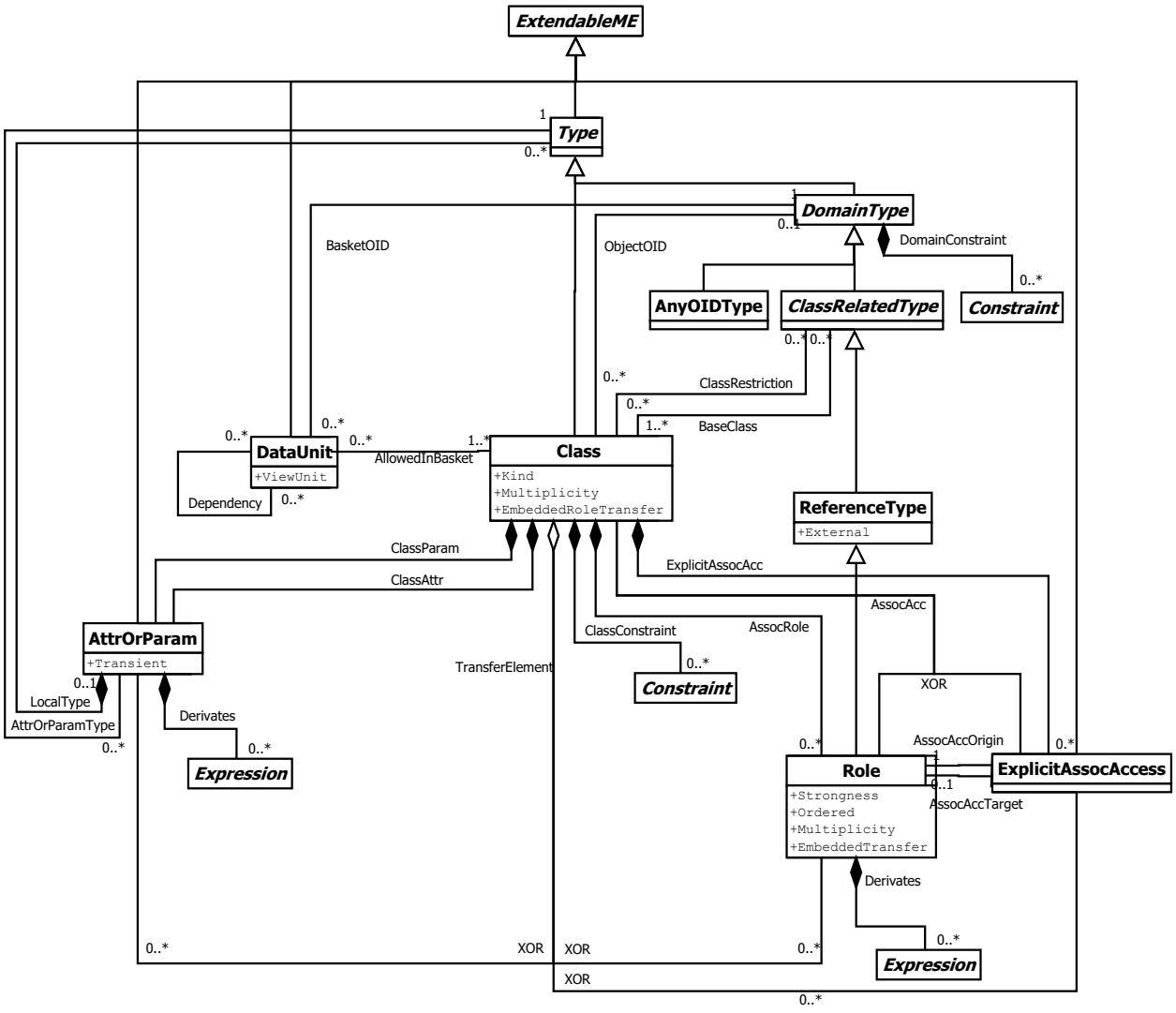

Abbildung 3: Übersicht Hauptkonstrukte

2.2.2 Strukturen und Klassen

Da Strukturen und Klassen (wie auch Beziehungen und Sichten) formal gleich definiert sind (vgl. INTERLIS 2-Referenzhandbuch, Kap. 2.5.3), werden sie im Metamodell durch ein einziges Konstrukt, die Klasse "Class" dargestellt. Deren Attribut "Kind" sagt aus, um welche Art (Struktur, Klasse, Assoziation, Sicht) es sich handelt. Bei Views sind die weiteren Angaben in einer Unterklasse enthalten (vgl. Kap. 2.5).

Die wichtigste Eigenschaft einer Klasse ist die Definition der Attribute der Klasse. Sie erfolgt mit der Komposition "ClassAttr". Ist die beschriebene Klasse eine Unterklasse der INTERLIS-Klasse Metaobject können ihr auch Parameter zugeordnet sein. Dies erfolgt mit der Komposition "ClassParam".

Da die Beschreibungen von Attributen und Parametern weitgehend identisch sind, wird beides mit der Klasse "AttrOrParam" beschrieben. Der Klassenname gehört bei beiden zum Namenspfad der OID. Die wichtigste Eigenschaft eines Attributes oder eines Parameters ist der Typ. Er wird über die Beziehung "AttrOrParamType" beschrieben. Sind die Werte von Attributen durch "Expressions" definiert, werden diese über das List-Attribut "Derivates" beschrieben.

Typen, die nicht als eigenständige Konstrukte, sondern direkt im Rahmen der Attributdefinition definiert wurden (z.B. Name: TEXT*30), werden über die Kompositionen "LocalType" dem Attribut oder Parameter zugeordnet. Das Attribut "Name" erhält den Wert "Type".

Ist das Attribut ein AttributeRefType, ist es möglich, dass innerhalb der Restriktionen neue Typen definiert werden. Diese werden ebenfalls über die Kompositionen "LocalType" dem Attribut zugeordnet. Das Attribut "Name" erhält den Wert "Type." gefolgt von einer Laufnummer (beginnend mit 1).

Die Konsistenzbedingungen einer Klasse werden durch eine Menge (INTERLIS-BAG) entsprechender Strukturelemente (Erweiterung der Struktur "Constraint") beschrieben (vgl. Kap. 2.5.2).

Eine wichtige Eigenschaft einer Klassendefinition ist die Angabe, welche Attribute und Rollen im Rahmen eines Objektes dieser Klasse in welcher Reihenfolge transferiert werden müssen. Damit diese Eigenschaft auf einfache Art zur Verfügung steht wird sie über die Beziehung "TransferElement" angeboten, obwohl dies eine Redundanz darstellt.

Eine modellierte Klasse kann auch Beziehungen zu anderen modellierten Klassen aufweisen. Dies ist im Kapitel 2.2.3 näher beschrieben.

2.2.3 Beziehungen und Referenzen

Eine Beziehung ist primär mit einem Objekt der Klasse "Class" beschrieben (Class.Kind=Association). Diesem Objekt sind Rollenobjekte (Klasse "Role") kompositorisch zugeordnet (Attribut "Name": Rollenname). Der Superklasse "ClassRelatedType" sind die möglichen Klassen der Bezugsobjekte zugeordnet: Einerseits die Basisklassen der möglichen Varianten (Beziehung "BaseClass"), andererseits allfällige Einschränkungen (Beziehung "ClassRestriction"). (Die als Einschränkungen zugeordneten Klassen sind immer Subklassen einer der Basisklassen und damit mit dieser über die Beziehung "Inheritance" direkt oder indirekt verbunden.)

Die Beziehungszugänge, die bei den durch die Beziehung verbundenen Klassen (Zielklassen) entstehen, sind mittels der Beziehung AssocAcc beschrieben. Im Normalfall wird dabei direkt auf die Rolle verwiesen, über welche die Zielobjekte erreicht werden können.

Da bei Mehrfachbeziehungen zur gleichen Klasse Namenskonflikte unvermeidbar sind, werden dann Metaelemente der Klasse "ExplicitAssocAccess" gebildet, die der Klasse mittels der Beziehung ExplicitAssocAcc kompositorisch zugeordnet sind (Attribut "Name": Beziehungszugangsname). Ein expliziter Beziehungszugang verweist einerseits über die Beziehung AssocAccOrigin auf die Rolle (derselben Klasse, zu welcher auch der Beziehungszugang gehört), mit welcher der Beziehungszugang aus Sicht der Beziehung erreicht wird. Andererseits kann er mittels der Beziehung AssocAccTarget auf die Rolle verweisen, mit welcher das Bezugsobjekt aus Sicht der Beziehung erreicht wird. Fehlt dieser Verweis, sollen mit dem Beziehungszugang nicht Bezugsobjekte, sondern Beziehungsobjekte selbst erreicht werden. Da explizite Beziehungszugänge weder auf Grund der aktuellen INTERLIS-Sprache noch auf Grund des Metadatenmodells von UML enstehen können, dürften sie kaum von praktischer Bedeutung sein. Sie dokumentieren aber einen bestehenden Mangel und die zur Behebung nötige Massnahme.

Referenzattribute werden mittels der Klasse "ReferenceType" dargestellt.

2.2.4 Dateneinheiten

Dateneinheiten (Klasse "DataUnit") beschreiben, wie Datenbehälter (INTERLIS-Baskets) aufgebaut sind.

Die Dependency-Beziehung "Dependencies" zeigt, von welchen anderen Dateneinheiten eine Dateneinheit abhängig ist. Der OID-Typ der Behälter ergibt sich über die Beziehung "BasketOID". Über die Beziehung "AllowedInBasket" sind alle Klassen definiert, die in einem solchen Behälter vorkommen dürfen.

Die Beziehung "DeferredGenerics" verweist auf generische Domains, für die noch keine konkrete Definition vorliegt, deren konkreter Domain also in den Transferdaten angegeben wird.

Das Attribut "DataUnitName" enthält die im Transfer zu verwendende Typenbezeichnung (XML-Tag).

In der aktuellen Sprachdefinition INTERLIS 2.4 werden die Dateneinheiten nicht expizit definiert. Sie ergeben sich vielmehr aus den Topic-Definitionen (vgl. Kap. 2.1.4).

2.2.5 Beliebige OID

Als etwas spezieller Typ wurde "AnyOIDType" eingeführt. Zu diesem Typ gibt es ein primäres Objekt: "INTERLIS.ANYOID". Explizite OID-Typen (Text oder numerisch) verweisen über die Inheritance-Beziehung entweder auf dieses Objekt oder auf explizite AnyOID-Typen, welche ihrerseits auf "INTERLIS.ANYOID" verweisen.

2.3 Typen

2.3.1 Übersicht

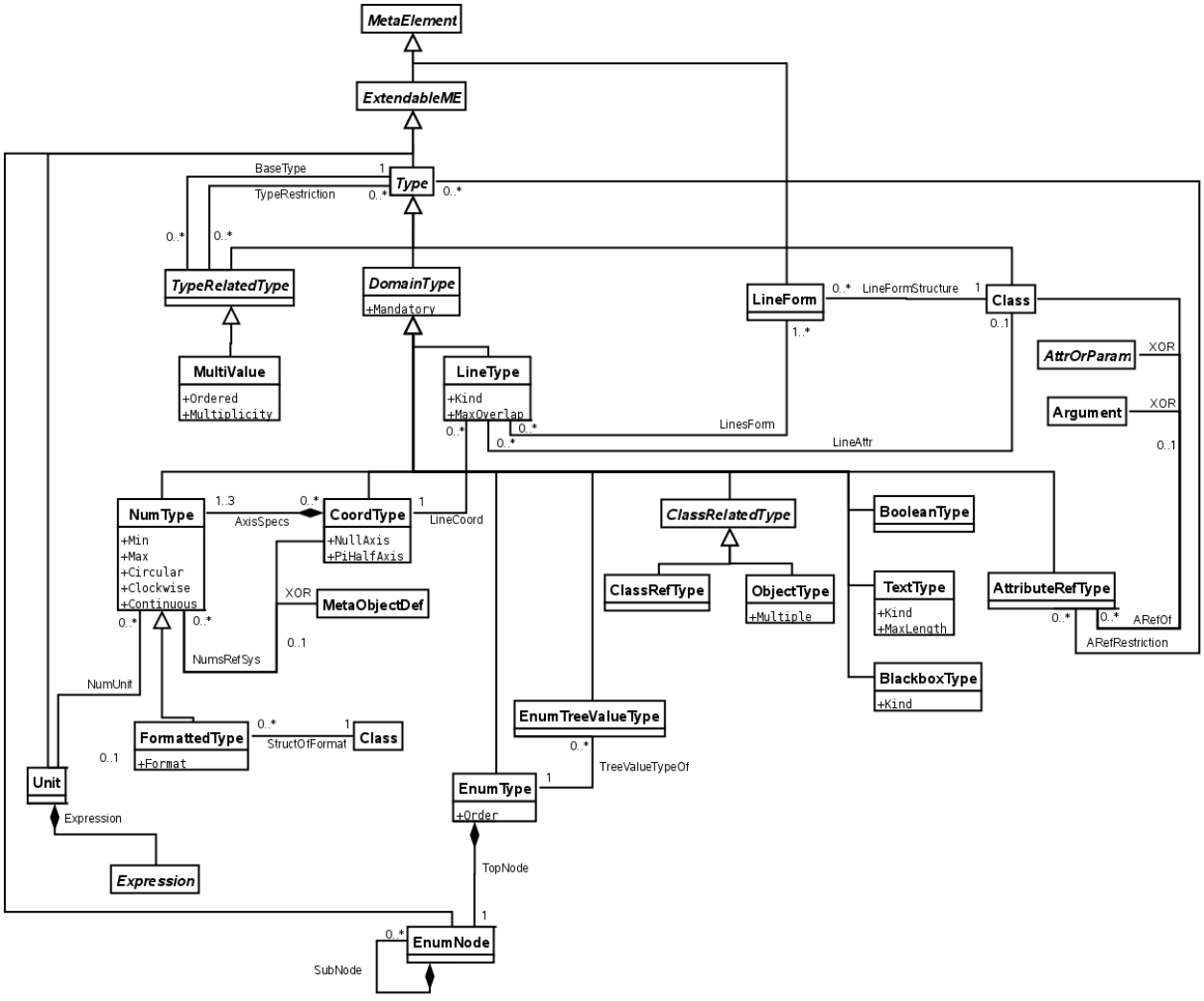

Abbildung 4: Übersicht Typen

Das Typenkonzept wurde gegenüber der INTERLIS 2-Sprache etwas verallgemeinert. Es entspricht so eher den Typenkonzepten von Programmiersprachen. Als erste Erweiterungen von Typ werden "DomainType", "Class" und "TypeRelatedType" eingeführt.

Der DomainType definiert einen bestimmten Wertebereich. Dabei ist (entsprechend der Sprache) ein NULL-Wert grundsätzlich zugelassen, es sei denn dies ist ausdrücklich ausgeschlossen (Attribut

Mandatory). Im Rahmen von "DomainType" werden primär die verschiedenen Basistypen (vgl. Kap. 2.3.2 bis 2.3.6) und die Linientypen ("LineType") unterschieden.

Da Strukturen und Klassen in ihrem Aufbau sehr ähnlich sind, wird zu ihrer Beschreibung die Klasse "Class" verwendet (vgl. Kap. 2.2). Damit Strukturattribute (Attribut, dessen Aufbau durch eine Struktur beschrieben wird) im Metamodell nicht als Spezialfall behandelt werden müssen, ist die Klasse "Class" als Erweiterung von "Type" realisiert. Ist einem Attribut (Klasse "AttrOrParam") über die Beziehung "AttrOrParamType" der Typ "Class" zugeordnet, kann es sich aber nur um eine Struktur handeln.

Gemäss INTERLIS 2-Referenzhandbuch kann ein Attribut-Typ (Regel "AttrType") als Ganzes obligatorisch (MANDATORY) oder fakultativ sein. Bei Strukturen macht dies jedoch keinen Sinn: diese Aussage muss bei den einzelnen Attributen der Struktur gemacht werden. Class ist darum eine direkte Erweiterung von Type und nicht von DomainType.

Die Klasse "TypeRelatedType" bildet die Grundlage für Typen, welche von einem anderen Typ (mittels der Beziehung "BaseType") Gebrauch machen, der allenfalls zusätzliche eingeschränkt ist (Beziehung "TypeRestriction"). Zurzeit gibt es zu "TypeRelatedType" nur eine erweiternde Klasse: "MultiValue". Mit ihr wird eine Menge von Unterelementen beschrieben. Mittels Attributen wird beschrieben, ob die Menge geordnet (LIST) ist oder nicht (BAG), und welche Kardinalität sie aufweist.

2.3.2 Typen für Text und Blackbox

Bei den Typen "TextType" und "BlackboxType" wird primär die Art des Typs, beim Text-Typ zudem die maximale Grösse, angegeben. Ist die Grösse eines TextType unbeschränkt, bleibt das Attribut "MaxLength" undefiniert.

2.3.3 Numerische Typen und Einheiten

Der numerische Typ ("NumType") enthält als Attribute die verschiedenen Angaben, wie sie in der INTERLIS-Sprache vorgesehen sind.

Einheit ("Unit") und Referenzsystem sind über Beziehungen ("NumUnit", "NumsRefSys") zugeordnet, sofern sie definiert sind.

Die formatierten Typen werden als spezielle numerische Typen aufgefasst, da sie auf einer Zusammensetzung von numerischen Feldern von Strukturen beruhen. Beschreibt ein formatierter Typ z.B. Stunden, Minuten und Sekunden, kann er auch als numerischer Typ mit Sekunden aufgefasst werden. Stunden und Minuten sind entsprechend umzurechnen. Handelt es sich beim formatierten Typ z.B. um Monate und Tage, kann er zwar als numerischer Typ mit Tagen aufgefasst werden. Ohne spezielle Kenntnis der nötigen Umrechnung, müssen aber alle Monate mit 31 Tagen angenommen werden. Der numerische Typ ist als Folge nicht mehr kontinuierlich (Attribut "Continuous").

2.3.4 Booleantyp

Boolean wird mit einem eigenständigen Typ ("BooleanType"), und nicht etwa wie gemäss Bemerkung im Referenzhandbuch, als spezielle Aufzählung abgebildet.

2.3.5 Koordinatentyp

Mit dem Koordinatentyp ("CoordType") werden die pro Achse zugehörigen numerischen Typen (Beziehung "AxisSpecs") und die Drehsinnangaben definiert.

2.3.6 Aufzählungen

Jeder Aufzähltyp (Klasse "EnumType") wird vollständig beschrieben, enthält also alle Knoten (Klasse "EnumNode"). Dem Aufzähltyp ist zunächst ein künstlicher Knoten zugeordnet (mit dem Namen "TOP") Diesem sind dann direkt und indirekt alle echten Knoten untergeordnet (jeweils mit dem Aufzählwert als Namen). Bei erweiterten Aufzählungstypen sind sowohl der Aufzähltyp wie alle übernommenen Knoten mit

dem entsprechenden geerbten Metaelement verbunden (Beziehung "Inheritance"). Dürfen zu einem Knoten keine weiteren Unterknoten mehr angefügt werden, ist das Attribut "ExtendableME.Final" auf den Wert "true" gesetzt. Das Attribut "ExtendableME.Abstract" hat immer den Wert "false".

Der Typ, der als Wert alle Knoten und nicht nur die Blätter zulässt, (vgl. INTERLIS 2-Referenzhandbuch, Kap. 2.8.2), ist über die Beziehung "TreeValueTypeOf" mit dem eigentlichen Aufzähltyp verbunden.

2.3.7 Wertebereiche von Klassen und Attributen

Der Typ für den Wertebereich von Klassen (Klasse "ClassRefType") ist eine Erweiterung der Klasse "ClassRelatedType", die auch im Rahmen von Beziehungen und Referenzen verwendet und dort näher beschrieben wird.

Der Typ für den Wertebereich von Attributen (Klasse "AttributeRefType") kann über die Beziehung "ARefOf" je nach Situation auf eine Klasse (Klasse "Class"), eine Klassenreferenz (Klasse "ClassRefType") oder ein Argument (Klasse "Argument") verweisen. Über die Beziehung "ARefRestriction" ergeben sich die für das Attribut zulässigen Typen.

2.3.8 Objekt Typen

Einzelobjekte oder Objektmengen können Argumente von Funktionen sein. Die Beschreibung erfolgt mit dem Objekt Typ ("ObjectType"), einer Erweiterung der Klasse "ClassRelatedType". Mit dem Attribut "Set" wird angezeigt, ob eine Objektmenge oder ein Einzelobjekt erwartet wird.

2.3.9 Einheiten

Einheiten werden mit Objekten der Klasse Unit beschrieben. Bei abstrakten Einheiten erhält das Attribut Name den Unit-Namen als Wert. Bei konkreten Einheiten erhält das Attribut Name den Kurznamen der Einheit als Attribut. Der volle Name wird im Interesse der Übersetzbarkeit mittels eines DocText-Elementes beschrieben, das den Namen "FullName" und als Text den vollen Namen enthält.

Die Definition von abgeleiteten, bzw. zusammengesetzten Einheiten wird mittels eines Ausdrucks beschrieben. Darin können jedoch nur wenige Arten von Faktoren vorkommen (Unit-Referenzen, Konstanten, Funktionsaufrufe).

2.4 Generische Wertebereiche

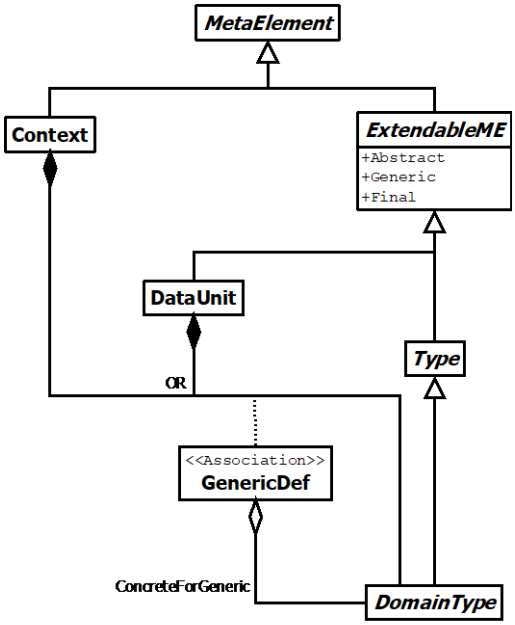

Abbildung 5: Generische Wertebereiche

Ist ein Wertebereich generisch, wird dies in der Superklasse "ExtendableME" mittels Generic= #true angemerkt. Obwohl generische Wertebereiche in INTERLIS2 nur für Koordinaten zulässig sind, wird im Metamodell auf diese Einschränkung verzichtet.

Im Rahmen von Kontexten können zu generischen Wertebereichen konkrete Wertebereiche definiert werden. Für jeden generischen Wertebereich, zu dem konkrete Definitionen gemacht werden, existiert eine Instanz der Beziehung "GenericDef", die kompositorisch zur Kontext-Instanz gehört und auf den generischen Wertebereich verweist. Dieser Beziehungsinstanz sind über die Beziehung "ConcreteForGeneric" die konkreten Wertebereiche zugeordnet, die durch diesen Kontext definiert sind.

Damit für eine DataUnit klar ist, welche konkreten Wertebereiche für generische Wertebereiche zur Anwendungen kommen, wird dasselbe Konstruktur wiederum verwendet: Zu jedem generischen Wertebereich, der in der DataUnit vorkommt, exisitiert eine Beziehungs-Instanz, die der DataUnit kompositorisch zugeordnet ist und auf den generischen Wertebereich verweist. Dieser Beziehungsinstanz sind über die Beziehung "ConcreteForGeneric" die konkreten Wertebereiche zugeordnet, die für diese DataUnit auf Grund der dafür gültigen Kontexte gelten. Ist mehr als ein konkreter Wertebereich definiert, wird erst in den Transferdaten festgelegt, welcher für die Daten gilt.

2.5 Sichten und Grafik

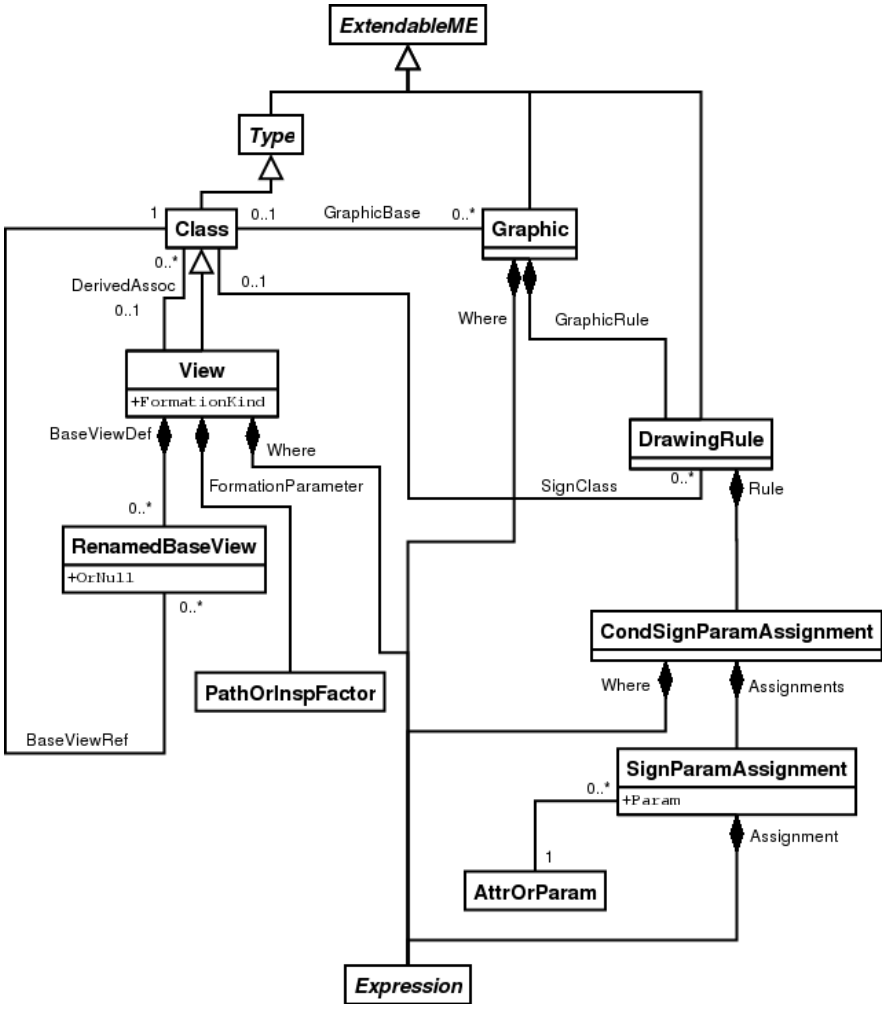

Abbildung 6: Sichten und Grafik

Die Beschreibung einer Sicht (Klasse "View") ist eine Erweiterung von "Class", welche die Beschreibung der Attribute leistet. Mit Objekten der Klasse View wird beschrieben, wie die Sicht gebildet wird und (über "RenamedBaseView"), welche Basisklassen Beiträge an die Sicht leisten.

Weitere Parameter zur Bildung der Sicht (vgl. INTERLIS 2-Referenzhandbuch Kap. 2.15) werden nur für die Aggregation (sofern EQUAL verlangt) und für die Inspektion zur Bezeichnung des zu inspizierenden Attributes angegeben ("FormationParameter"). Ist eine Verknüpfung eine sogenannte "Outer-Join", wird bei der entsprechenden Definition der Basisklasse (Klasse "RenamedBaseView") dem Attribut "OrNull" der Wert "true" zugewiesen.

Im Falle einer Aggregation wird das implizite Attribut "AGGREGATES" gebildet (mit diesem Namen). Dessen Typ ist ein MultiValue, dessen Basistyp die Basisklasse der Aggregation ist.

Wird eine Sicht erweitert, werden in der Erweiterung die Definitionen der Basisklassen nicht wiederholt, sondern nur allfällige Erweiterungsdefinitionen angefügt.

Die Grafikbeschreibung folgt weitgehend der Struktur der INTERLIS-Sprache. Die für die Zuweisung nötigen Konstrukte wurden im Rahmen der Faktoren und Ausdrücke geschaffen. Ob die speziellen Konstrukte auch in anderen Bereichen als der Grafik zur Anwendung kommen sollen, ist damit offen.

2.6 Weitere Konstrukte

2.6.1 Ausdrücke und Faktoren

Faktoren und Ausdrücke sind nicht als eigenständige Objekte (Metaelemente), sondern als Strukturen definiert. Die verschiedenen Konstrukte entsprechen ziemlich direkt den Sprachkonstrukten.

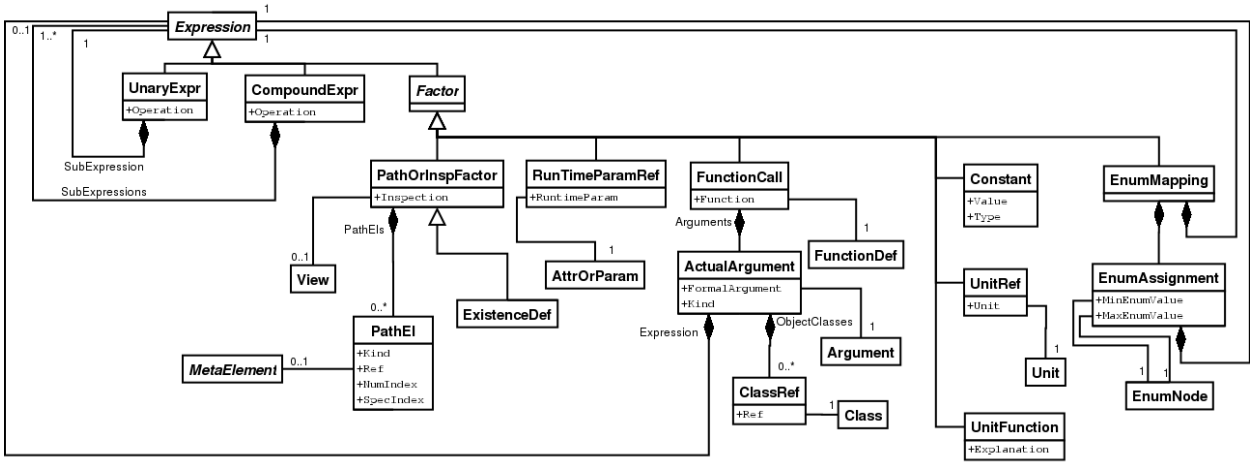

Abbildung 7: Ausdrücke und Faktoren

Bei den Ausdrücken wird unterschieden zwischen:

- "UnaryExpr": Ausdrücke, bei denen sich eine Operation (Not, Defined) auf einen Unterausdruck bezieht.
- "CompoundExpr": Ausdrücke, bei denen sich eine Operation (And, Or, Vergleiche, InRange, CondExpr) auf eine Menge von Unterausdrücken bezieht.
- "Factor": Faktoren, die den Bezug zu den Modellelementen herstellen.

Spezielle Hinweise zu "Factor":

- Die Unitreferenz (Struktur UnitRef) kommt nur im Rahmen von abgeleiteten sowie zusammengesetzten Einheiten vor.
- Die Abbildung (Struktur EnumMapping) von Aufzählungswerten in andere Werte, wird vor allem im Rahmen der Grafik benötigt.

2.6.2 Konsistenzbedingungen

Konsistenzbedingungen sind als Subklassen der Klasse "Constraint" modelliert. Diese ist ihrerseits eine Subklasse der Klasse "MetaElement". Instanzen von Konsistenzenbedingung gehören über die Beziehung "ClassConstraint" bzw. "DomainConstraint" kompositorisch zum Element (Klasse bzw. Domain), das sie präzisieren. Die verschiedenen Unterklassen der Klasse "Constraint" entsprechen ziemlich direkt den Sprachkonstrukten.

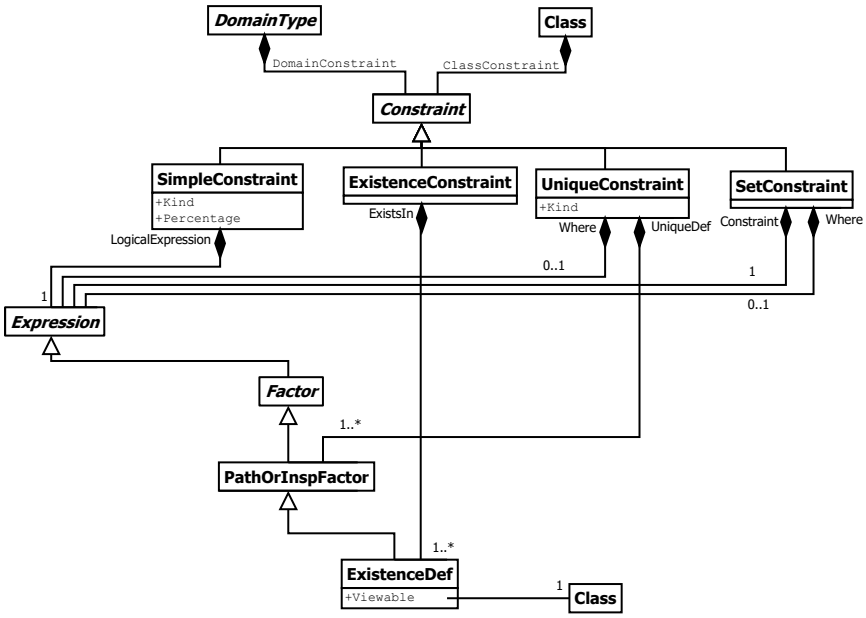

Abbildung 8: Konsistenzbedingungen

2.6.3 Definition von Funktionen

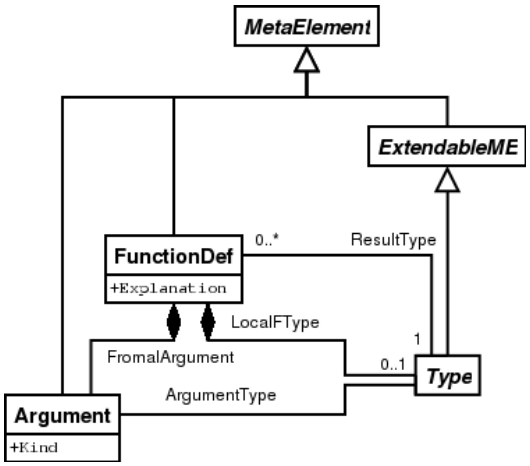

Abbildung 9: Definition von Funktionen

Einer Funktionsdefinition (Klasse "FunctionDef") ist der Typ des Resultats über die Beziehung "Result" zugeordnet. Die Argumente sind der Funktionsdefinition kompositorisch zugeordnet (Beziehung "FormalArgument"). Der Namenspfad ergibt sich aus …Funktionsname.Argumentname. Sind Typen des Resultats oder der Argumente lokal definiert, sind sie der Funktionsdefinition kompositorisch zugeordnet (Beziehung "LocalFType"). Der Namenspfad ergibt sich mit …Funktionsname.TYPE für den Resultattyp (da TYPE ein Schlüsselwort der Sprache ist, kann es keinen solchen Argumentnamen geben), bzw. mit …Funktionsname.Argumentname.TYPE.

Die Argumente sind mit ihrem Typ über die Beziehung ArgumentType verbunden.

2.6.4 Metaobjekte

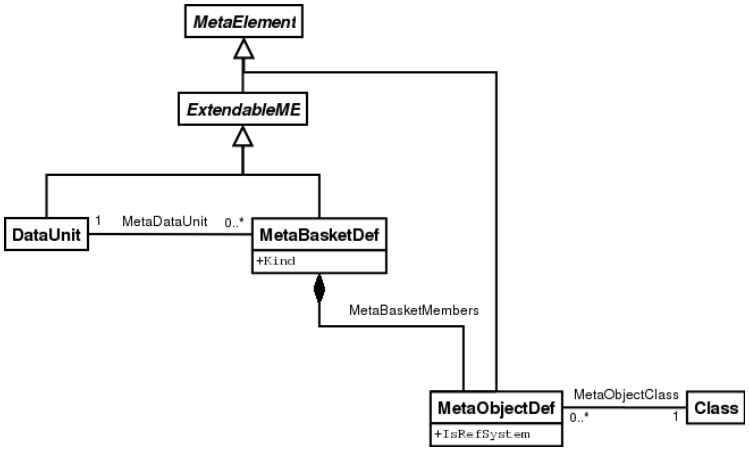

Abbildung 10: Metaobjekte

Die Klassen "MetaDataBasketDef" und "MetaObjectDef" dienen der Beschreibung der Metadatenbehälter und der Metaobjekte.

Der Name der Metadatenbehälter-Beschreibung entspricht dem Behälternamen. Die in diesem Behälter erwarteten Metaobjekte sind über die Komposition MetaBasketMembers zugeordnet.

2.6.5 Laufzeitparameter

Laufzeitparameter (vgl. INTERLIS 2-Referenzhandbuch, Kap. 2.11) werden mittels der Objekte der Klasse AttrOrParam dargestellt, die direkt dem jeweiligen Package zugeordnet sind.

3 Aufteilung der Modelldaten in Baskets

Damit Modelldaten auf einfache Art und Weise selektiv verwendet werden können, wird pro Modell (Original oder Übersetzung) ein eigener Behälter (Basket) gebildet. Dieser erhält als Basketidentifikation den konstanten Text "MODEL" gefolgt von einem Punkt und dem Modellnamen. Damit ist gewährleistet, dass die Basketidentifikation nicht mit einer Objektidentifikation übereinstimmen kann.

4 Das INTERLIS-Metamodell 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.
```