Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Département fédéral de la défense,
de la protection de la population et des sports DDPS

Office fédéral de topographie swisstopo

INTERLIS 2 –

Métamodèle

Edition du 2022-06-17 (français)

Informations et contact : www.interlis.ch, info@interlis.ch

Ce document a été rédigé initialement en allemand, les auteurs de la version française ont essayé de respecter, le plus rigoureusement possible, le texte original.

Table des matières

1 Introduction ... 3
1.1 Définition et finalité ... 3
1.2 Rapports entretenus avec d'autres normes ... 3
1.3 Lien avec le langage INTERLIS ... 4
1.3.1 Intégralité ... 4
1.3.2 Métaobjets ... 4
1.4 Conseils de lecture ... 4

2 Le métamodèle INTERLIS ... 5
2.1 Constituants de base ... 5
2.1.1 Vue d'ensemble ... 5
2.1.2 Classe MetaElement ... 5
2.1.3 Gestion d'extensions ... 6
2.1.4 Modèles et thèmes ... 6
2.1.5 Traductions ... 7
2.2 Constituants principaux ... 8
2.2.1 Vue d'ensemble ... 8
2.2.2 Structures et classes ... 8
2.2.3 Relations et références ... 9
2.2.4 Unités de données ... 9
2.2.5 OID quelconques ... 10
2.3 Types ... 10
2.3.1 Vue d'ensemble ... 10
2.3.2 Types pour Text et Blackbox ... 11
2.3.3 Types numériques et unités ... 11
2.3.4 Type booléen ... 11
2.3.5 Type de coordonnées ... 11
2.3.6 Enumérations ... 11
2.3.7 Domaines de valeurs de classes et d'attributs ... 12
2.3.8 Types d'objets ... 12
2.3.9 Unités ... 12
2.4 Domaines de valeurs génériques ... 13
2.5 Vues et graphique ... 14
2.6 Autres constituants ... 15
2.6.1 Expressions et facteurs ... 15
2.6.2 Conditions de cohérence ... 15
2.6.3 Définition de fonctions ... 16
2.6.4 Métaobjets ... 17
2.6.5 Paramètre d'exécution ... 17

3 Répartition des données de modèle en conteneurs ... 18

4 Le métamodèle INTERLIS en INTERLIS 2 ... 19

1 Introduction

1.1 Définition et finalité

Par métamodèle, on entend le modèle de données qui décrit la manière dont les descriptions de modèles sont modélisées. Les données qui respectent un métamodèle et décrivent ainsi un modèle de données sont appelées des données de modèle.

Les données de modèle possèdent fondamentalement le même contenu descriptif que les descriptions de modèles en langage INTERLIS. Leur structure et leur codage ne sont cependant pas tournés vers l'élégance descriptive et la lisibilité pour leurs utilisateurs mais vers un emploi aussi simple que possible dans des programmes, par exemple :

- des applicatifs génériques dont le fonctionnement dépend pour partie des informations du modèle
- des générateurs de parties de programmes respectant les informations du modèle
- des outils convertissant des informations du modèle d'une manière ou d'une autre (par exemple dans d'autres standards).

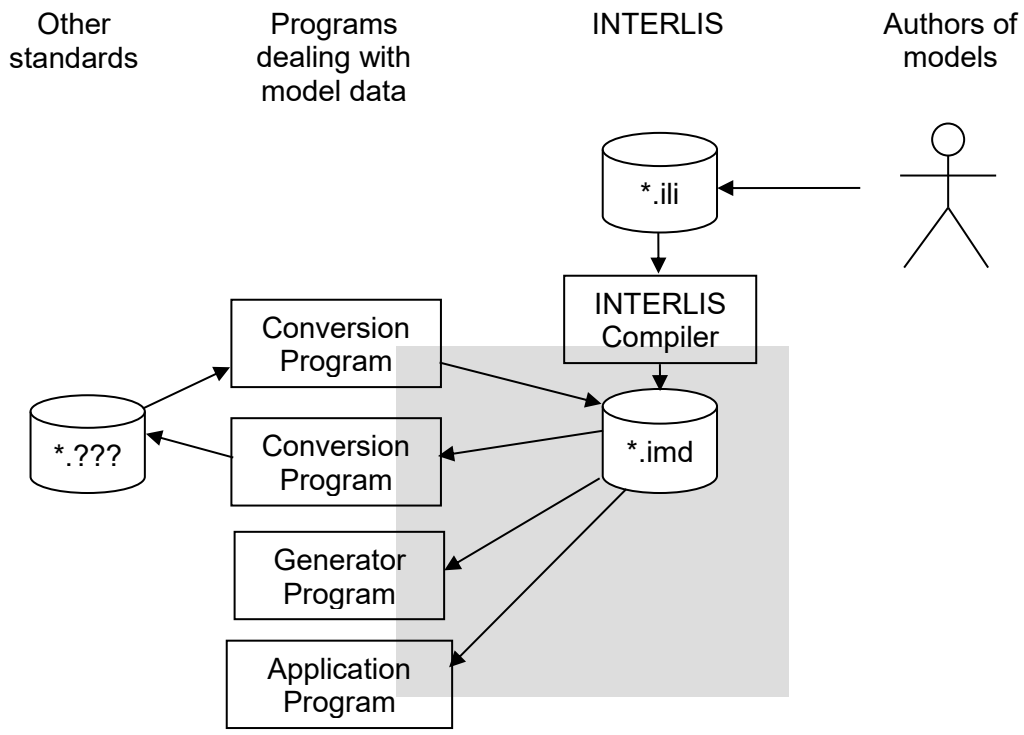

Figure 1 : importance des données de modèles

En conséquence, le présent document ne s'adresse pas à des personnes développant des modèles INTERLIS, mais à celles et ceux qui écrivent des programmes utilisant des informations de modèles ou produisant des données de modèles INTERLIS (compilateur INTERLIS ou programmes de conversion depuis d'autres standards).

1.2 Rapports entretenus avec d'autres normes

Les métamodèles et les métadonnées correspondantes existent non seulement pour INTERLIS mais également pour d'autres techniques de description de données (par exemple UML-MOF, schéma XML).

La question de la prise en charge d'un métamodèle lié à une telle technique s'est donc posée. La réponse a été négative, parce que les différences sont généralement conséquentes et que par suite, la proximité avec le langage de description INTERLIS aurait été perdue. D'autres techniques de description peuvent cependant être utilisées : des données de modèle INTERLIS peuvent être générées à partir de données de modèle d'autres techniques et réciproquement. Toutefois, une telle mise en oeuvre peut être ajoutée à la spécification INTERLIS. Il pourrait en outre se révéler judicieux, pour des standards donnés (par exemple GML), d'élaborer des modèles de base en INTERLIS 2, contenant les structures fondamentales du standard tiers considéré en INTERLIS.

1.3 Lien avec le langage INTERLIS

1.3.1 Intégralité

Diverses règles sémantiques sont liées au langage INTERLIS (cf. manuel de référence INTERLIS 2).

Le métamodèle étant cependant tourné vers l'utilisation des données de modèle au sein de programmes et non vers la génération correcte des données de modèle, les règles sémantiques ne sont d'une part pas mises en oeuvre intégralement au sein du métamodèle (par exemple sous forme de conditions de cohérence) et certaines situations sont d'autre part reproduites de façon redondante (ainsi, certaines propriétés peuvent être représentées à plusieurs reprises : en propre et par héritage, au moyen de sous-éléments directs).

En conséquence, les données conformes au métamodèle INTERLIS 2 (données du modèle en abrégé ; *.imd) ne constituent jamais la description primaire du modèle de données. Celle-ci est habituellement établie sous forme textuelle (*.ili), fournie par une autre technique de modélisation ou gérée par un éditeur de modèles (par exemple l'éditeur UML/INTERLIS). Un tel modèle de données primaire peut parfaitement contenir plus d'informations que le métamodèle INTERLIS 2 (en particulier des informations sur la représentation du modèle au sein de diagrammes).

1.3.2 Métaobjets

Le langage INTERLIS 2 utilise notamment la notion de métaobjets (cf. manuel de référence INTERLIS 2, § 2.10). Ces métaobjets décrivent cependant des systèmes de référence et des symboles, raison pour laquelle ils ne doivent pas être confondus avec des objets des modèles de données. Afin de réduire le risque de confusion, les objets des données de modèles INTERLIS sont appelés des métaéléments.

1.4 Conseils de lecture

Les diverses descriptions portant sur le métamodèle supposent la connaissance préalable des constituants d'INTERLIS (modèles, thèmes, classes, etc.), tels qu'ils sont définis dans le manuel de référence INTERLIS 2.

Les descriptions de détail supposent également que le métamodèle est toujours conditionné par la description INTERLIS 2 au chapitre 4.

Les classes et les structures sont représentées à l'identique sur les diagrammes UML. L'utilisation d'une structure au sein d'une autre structure ou classe est représentée comme une composition, le nom d'attribut étant indiqué à côté du losange plein, contrairement au cas des relations entre classes.

Les attributs de référence sont représentés de manière similaire aux relations, sous la forme d'une liaison entre les classes source et cible. L'attribut de référence est répertorié comme un attribut au sein de la classe source et la ligne figurant la relation conduit jusqu'à lui.

2 Le métamodèle INTERLIS

2.1 Constituants de base

2.1.1 Vue d'ensemble

La figure 2 présente une vue d'ensemble du métamodèle INTERLIS prenant la forme d'un diagramme de classes UML. Les données décrivant les modèles effectifs et celles qui résultent de traductions y sont clairement séparées. Les différentes classes et les relations entre elles vont être décrites dans les paragraphes suivants.

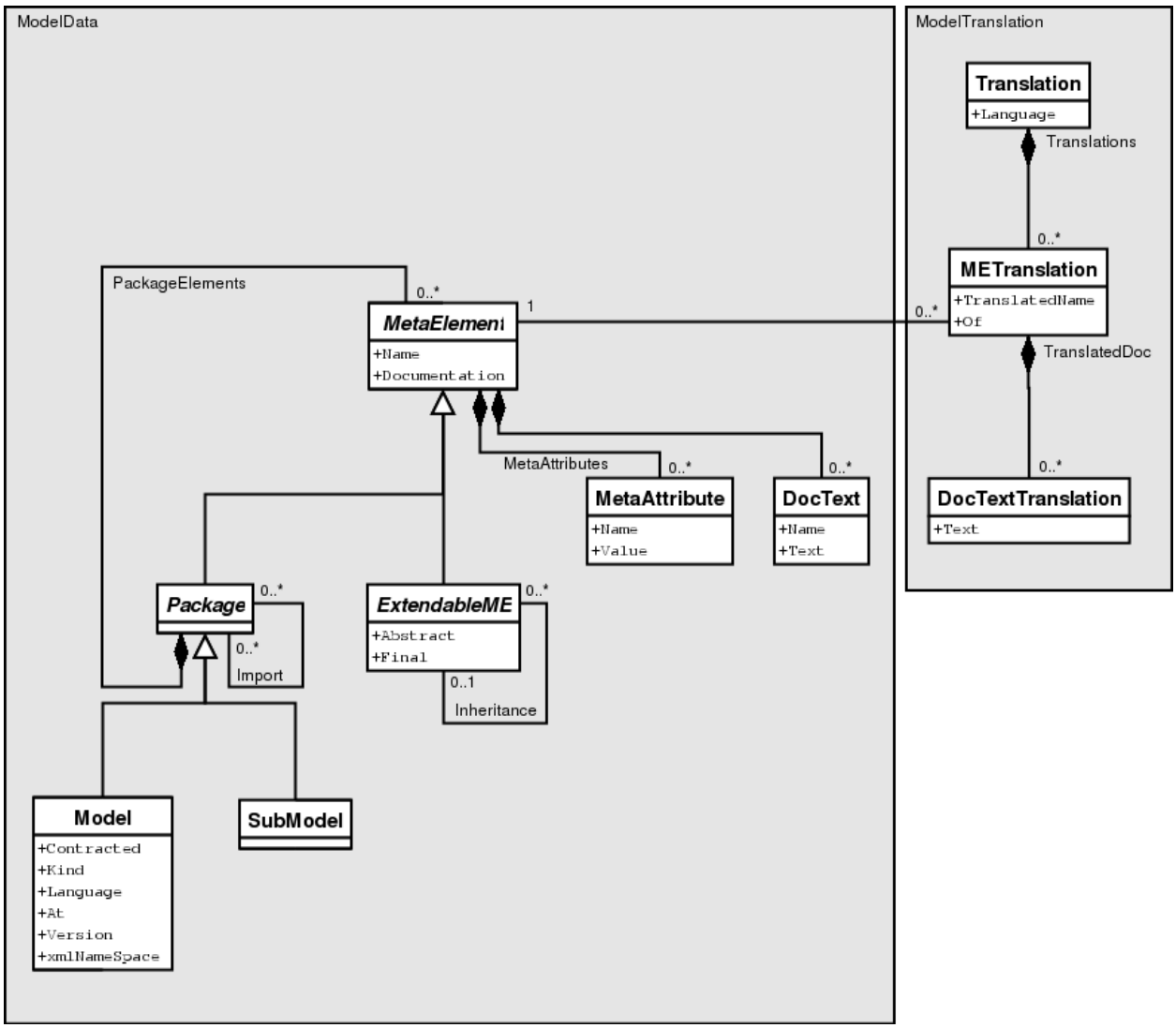

Figure 2 : vue d'ensemble des constituants de base

2.1.2 Classe MetaElement

La classe abstraite MetaElement est la classe d'objets de base pour toutes les données de modèle. On parlera ici de métaéléments et non de métaobjets afin d'éviter toute confusion avec les métaobjets du langage de description de données (cf. manuel de référence INTERLIS 2, § 2.10).

L'identification d'objet d'un métaélément se compose d'un chemin de nom comprenant les noms des métaéléments de niveau supérieur et se terminant par son propre nom (attribut "Name"). Les noms sont à chaque fois séparés par un point. Ce procédé garantit que l'identification d'objet reste également stable en

cas de compilations répétées (même en cas de modifications du modèle, pour autant qu'elles ne concernent pas la position du métaélément considéré).

Exemple : le modèle de données INTERLIS interne (cf. manuel de référence INTERLIS 2, annexe A) contient la structure UTC avec l'attribut Hours. Le métaélément de cette description d'attribut se voit donc associer l'OID "INTERLIS.UTC.Hours".

Un métaélément contient une liste (éventuellement vide) de textes de documentation (classe "DocText"). Chaque texte de documentation peut être identifié par un nom et contient le texte documentaire concerné. La manière dont les textes de documentation sont produits n'est pas régie par INTERLIS, elle relève de l'outil par lequel les données du modèle sont générées.

Des métaattributs peuvent être affectés par composition à un métaélément (classe "MetaAttribut"). Les métaattributs ne sont définis plus précisément ni par le langage INTERLIS ni par le métamodèle. Ils servent notamment à ce que des informations qui dépassent le cadre d'INTERLIS puissent être ajoutées comme des parties intégrantes des données de modèles. Les métaattributs possèdent un nom (attribut "Name") qui doit être sans équivoque au sein des métaattributs d'un même métaélément et une valeur descriptive (attribut "Value"). L'identification d'objet d'un métaattribut se compose toujours de l'OID du métaélément, complétée par le nom fixe METAOBJECT (mot-clé dans INTERLIS 2) et son propre nom. Si un métaattribut de nom X était défini dans l'exemple précédent, l'OID serait alors "INTERLIS.UTC.Hours.METAOBJECT.X". Les métaattributs ne doivent pas nécessairement se trouver dans le même conteneur que le métaélément. Il est en particulier possible de définir des conteneurs INTERLIS (cf. manuel de référence INTERLIS 2, § 1.4.5) ne comportant que des métaattributs relatifs à des métaéléments différents. C'est pourquoi le rôle MetaElement est désigné comme étant externe dans la relation MetaAttribute.

2.1.3 Gestion d'extensions

Différents constituants d'INTERLIS 2 peuvent être étendus au sens du paradigme orienté objet (par exemple les thèmes, les classes, les attributs). De tels constituants sont toujours des sous-classes directes ou indirectes de la classe de base abstraite "ExtendableME" (avec les attributs Abstract, Generic et Final). La relation Inheritance décrit le lien existant entre le métaélément de base (supérieur) et celui constituant l'extension (inférieur). Le métaélément de base pouvant être défini dans un autre conteneur, le rôle Super est désigné comme étant externe.

2.1.4 Modèles et thèmes

Afin que le métamodèle INTERLIS soit aussi proche que possible du métamodèle UML, les modèles et les thèmes sont représentés de façon un peu différente au sein du métamodèle par rapport à leur définition dans le langage INTERLIS.

De manière analogue à UML, la classe abstraite "Package" constitue la base des modèles et des sous-modèles. La relation "PackageElements" permet à un paquet – exception faite des modèles – de contenir d'autres métaéléments quelconques. Un sous-modèle (classe "SubModel") peut notamment contenir d'autres sous-modèles, même si cela n'est pas définissable avec le langage actuel (INTERLIS 2.4). Un paquet est associé aux paquets importés via la relation "Import".

Il est décrit, à l'aide de métaéléments de la classe "DataUnit", comment les unités de données (en particulier les conteneurs d'un échange de données INTERLIS) peuvent être construites et comment elles dépendent les unes des autres. La désignation de type à utiliser lors du transfert (Tag XML) est à tirer de l'attribut "DataUnitName". Du point de vue du métamodèle, seul le fait que les unités de données doivent être définies au sein d'un paquet est fixé. Les règles suivantes s'appliquent cependant pour la version actuelle du langage INTERLIS 2.4 :

- Un modèle INTERLIS est représenté par un métaélément de la classe "Model". Il contient, sous forme d'attributs, toutes les informations requises compte tenu des possibilités du langage. L'attribut xmlns est défini dans l'optique du transfert XML. Pour INTERLIS, il a la valeur http://www.interlis.ch/INTERLIS2.4.
- Un thème (Topic) du langage INTERLIS est représenté par un métaélément de chacune des deux classes "SubModel" et "DataUnit". Avec la classe SubModel, un thème peut être considéré comme un paquet au sens d'UML ; au sein de ce paquet, le métaélément "DataUnit" (Name: "BASKET", DataUnitName: Modelname"."Topicname) décrit la manière dont les conteneurs de données INTERLIS correspondants ("Basket") sont construits (cf. § 2.2.4).

Le constituant CONTRACTED ayant été abandonné dans INTERLIS 2.4, l'attribut correspondant reste disponible pour des raisons de compatibilité, mais il n'est plus obligatoire.

2.1.5 Traductions

Les modèles INTERLIS 2 peuvent être traduits (TRANSLATION OF). Toutefois, les métadonnées se rapportent toujours aux modèles originaux. Même lorsqu'un modèle importe un modèle traduit, les renvois (relations, références) se rapportent tous aux métaéléments de la langue d'origine concernée.

Des conteneurs propres sont constitués pour des traductions (thème de métamodèle "ModelTranslation"). Les objets de la classe "Translation" décrivent la langue de la traduction considérée et contiennent les traductions des métaéléments (structure "METranslation"). Celles-ci se rapportent à un métaélément donné (dans la langue source) et contiennent le nom traduit de même que les commentaires traduits, au besoin, dans le même ordre de succession que les commentaires originaux.

2.2 Constituants principaux

2.2.1 Vue d'ensemble

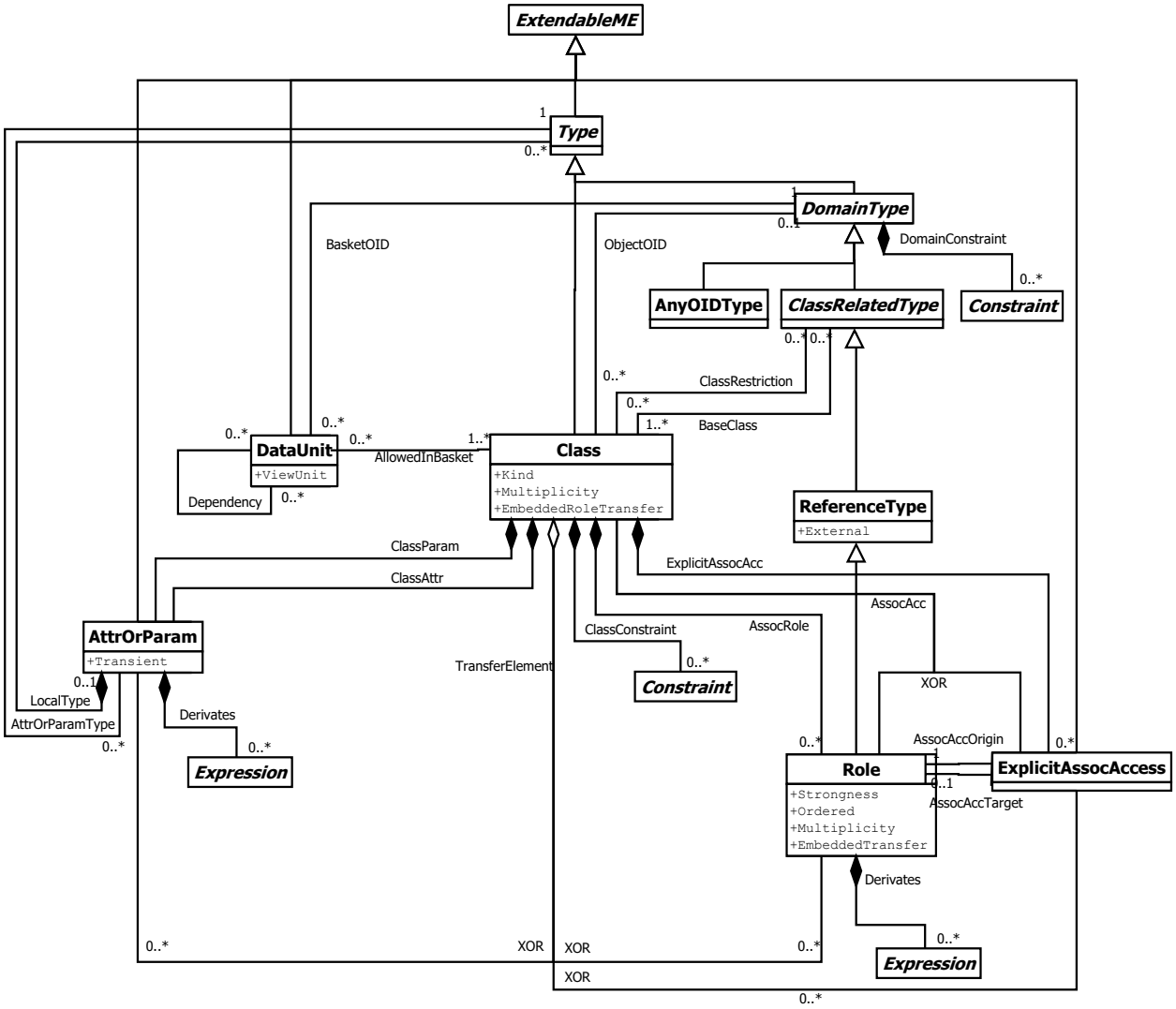

Figure 3 : vue d'ensemble des constituants principaux

2.2.2 Structures et classes

Les structures et les classes (tout comme les relations et les vues) étant définies à l'identique du point de vue formel (cf. manuel de référence INTERLIS 2, § 2.5.3), elles sont représentées par un seul constituant au sein du métamodèle, la classe "Class". Son attribut "Kind" indique le genre de constituant (structure, classe, association, vue) dont il s'agit. Pour les vues, les indications supplémentaires sont contenues dans une sous-classe (cf. § 2.5).

La propriété la plus importante d'une classe est la définition de ses attributs. Elle s'effectue à l'aide de la composition "ClassAttr". Si la classe décrite est une sous-classe de la classe INTERLIS "Metaobject", des paramètres peuvent aussi lui être affectés. La composition "ClassParam" sert à cette affectation.

Les descriptions des attributs et des paramètres étant largement identiques, elles sont décrites par la classe "AttrOrParam" dans les deux cas. De même, le nom de la classe appartient au chemin du nom de l'OID dans les deux cas. Le type est la propriété la plus importante d'un attribut ou d'un paramètre. Il est décrit via la relation "AttrOrParamType". Si les valeurs d'attributs sont définies par des "Expressions", ces dernières sont décrites par l'intermédiaire de l'attribut de liste "Derivates".

Les types qui n'ont pas été définis comme des constituants autonomes mais qui l'ont été directement, dans le cadre de la définition d'attribut (par exemple Name: TEXT*30), sont affectés à l'attribut ou au paramètre concerné via les compositions "LocalType". L'attribut "Name" contient la valeur "Type".

Si l'attribut est de type AttributeRefType, il est possible que de nouveaux types soient définis au sein des restrictions. Ceux-ci sont également affectés à l'attribut via les compositions "LocalType". L'attribut "Name" contient la valeur "Type." suivie d'un numéro d'ordre (affectation en continu débutant à 1).

Les conditions de cohérence d'une classe sont décrites par un ensemble (INTERLIS-BAG) d'éléments structurés correspondants (extension de la structure "Constraint") (cf. § 2.5.2).

Une propriété importante d'une définition de classe est l'indication des attributs et des rôles qui doivent être transférés dans le cadre d'un objet de cette classe ainsi que de l'ordre de succession à respecter à cette occasion. Pour que cette propriété soit à disposition de manière simple, elle est proposée via la relation "TransferElement", bien que cela constitue une redondance.

Une classe modélisée peut également être en relation avec d'autres classes modélisées. Le paragraphe 2.2.3 décrit cela plus avant.

2.2.3 Relations et références

Une relation est en premier lieu décrite avec un objet de la classe "Class" (Class.Kind=Association). Des objets de rôle (classe "Role") sont affectés par composition à cet objet (attribut "Name": nom du rôle). Les classes possibles des objets de référence sont affectées à la classe supérieure "ClassRelatedType" : d'une part les classes de base des variantes possibles (relation "BaseClass"), d'autre part d'éventuelles restrictions (relation "ClassRestriction"). (Les classes affectées comme des restrictions sont toujours des sous-classes de l'une des classes de base et sont donc directement ou indirectement liées à celle-ci par l'intermédiaire de la relation "Inheritance".)

Les accès à la relation liant les classes entre elles (classes cibles), sont décrits via la relation AssocAcc. Dans le cas normal, il est directement renvoyé ici au rôle via lequel les objets cibles peuvent être atteints.

Des conflits de noms étant inévitables dans le cas de relations multiples avec une même classe, des métaéléments de la classe "ExplicitAssocAccess" sont formés dans ce cas, affectés par composition à la classe au moyen de la relation ExplicitAssocAcc (attribut "Name": nom de l'accès à la relation). Un accès explicite à une relation renvoie d'une part, via la relation AssocAccOrigin, au rôle (de la même classe que celle à laquelle appartient l'accès à la relation) via lequel l'accès à la relation est atteint du point de vue de la relation. Il peut d'autre part, par l'intermédiaire de la relation AssocAccTarget, renvoyer aux rôles via lesquels l'objet de référence est atteint du point de vue de la relation. Si ce renvoi fait défaut, ce sont les objets de la relation eux-mêmes et non des objets de référence qui doivent être atteints via l'accès à la relation. Des accès explicites à des relations ne pouvant être générés ni sur la base du langage INTERLIS actuel ni sur celle du modèle de métadonnées d'UML, ils ne devraient guère avoir d'importance en pratique. Ils documentent toutefois une carence existante et la mesure requise pour y remédier.

Les attributs de référence sont représentés au moyen de la classe "ReferenceType".

2.2.4 Unités de données

Les unités de données (classe "DataUnit") décrivent la manière dont les conteneurs de données (INTERLIS-Baskets) sont construits.

La relation de dépendance "Dependencies" fait apparaître les autres unités de données dont dépend une unité de données. Le type d'OID des conteneurs découle de la relation "BasketOID". Toutes les classes pouvant se présenter dans un tel conteneur sont définies via la relation "AllowedInBasket".

La relation "DeferredGenerics" renvoie vers des domaines (Domains) génériques, pour lesquels il n'existe pas encore de définition concrète, de sorte que leur domaine concret est indiqué dans les données de transfert.

L'attribut "DataUnitName" contient la désignation de type (Tag XML) à utiliser lors du transfert.

Dans la définition actuelle du langage (INTERLIS 2.4), les unités de données ne sont pas définies de manière explicite. Elles résultent au contraire des définitions de thèmes (cf. § 2.1.4).

2.2.5 OID quelconques

"AnyOIDType" a été introduit comme étant un type un peu particulier. Il existe un objet primaire pour ce type : "INTERLIS.ANYOID". Des types d'OID explicites (textuels ou numériques) renvoient, via la relation Inheritance, soit vers cet objet, soit vers des types AnyOID explicites, lesquels renvoient à leur tour vers "INTERLIS.ANYOID".

2.3 Types

2.3.1 Vue d'ensemble

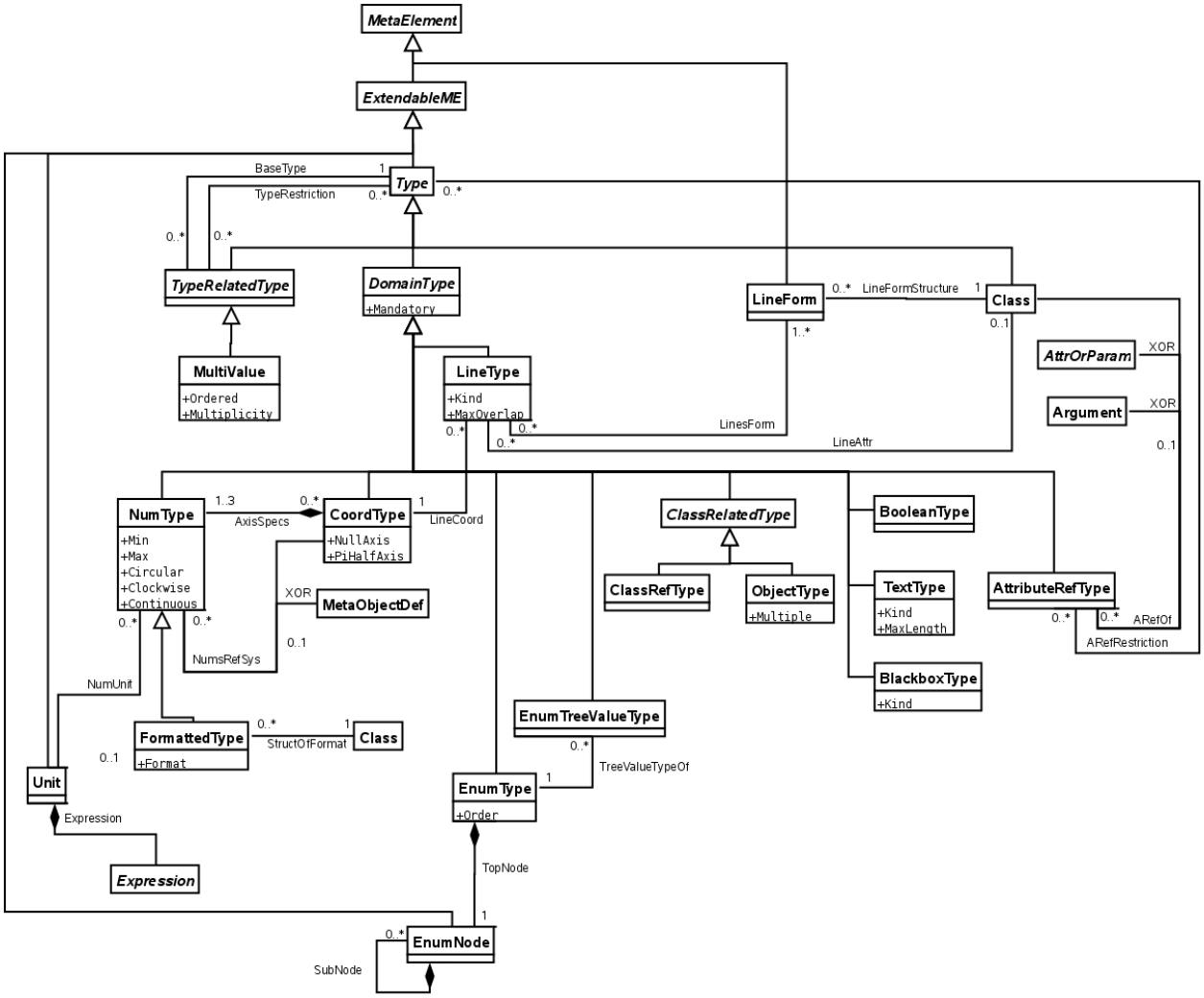

Figure 4 : vue d'ensemble des types

Le concept des types a été légèrement généralisé par rapport au langage INTERLIS 2. A présent, il correspond plutôt aux concepts de types des langages de programmation. "DomainType", "Class" et "TypeRelatedType" sont les premières extensions introduites pour le type.

Le type DomainType définit un domaine de valeurs donné. Une valeur nulle est fondamentalement admise (dans le respect du langage), sauf si cela est expressément exclu (attribut Mandatory). Dans le cadre de "DomainType", une distinction est d'abord établie entre les différents types de base (cf. § 2.3.2 à 2.3.6) et les types de lignes ("LineType").

Les structures et les classes étant très semblables du point de vue de leur construction, la classe "Class" est utilisée pour leur description (cf. § 2.2). Afin que les attributs de structure (attributs dont la construction est décrite par une structure) n'aient pas à être traités comme des cas particuliers dans le métamodèle, la classe "Class" est réalisée comme une extension du "Type". Si le type "Class" est affecté à un attribut (classe "AttrOrParam") via la relation "AttrOrParamType", il ne peut cependant s'agir que d'une structure.

Selon le manuel de référence INTERLIS 2, un type d'attribut (règle "AttrType") pris comme un tout peut être obligatoire (MANDATORY) ou facultatif. Dans le cas de structures, cela n'a toutefois aucun sens : cette déclaration doit être faite pour les différents attributs de la structure. En conséquence, "Class" est une extension directe de Type et non de DomainType.

La classe "TypeRelatedType" constitue la base pour les types qui font appel à un autre type (au moyen de la relation "BaseType"), lequel fait toutefois l'objet de restrictions supplémentaires (relation "TypeRestriction"). Il n'existe pour l'heure qu'une classe d'extension pour "TypeRelatedType", à savoir "MultiValue". Avec elle, un ensemble de sous-éléments est décrit. Il est précisé, à l'aide d'attributs, si l'ensemble est ordonné (LIST) ou non (BAG) et sa cardinalité est indiquée.

2.3.2 Types pour Text et Blackbox

Dans le cas des types "TextType" et "BlackboxType", c'est d'abord le genre du type qui est indiqué, la taille maximale étant en outre précisée pour le type "TextType". Si aucune limite n'est fixée pour la taille d'un type "TextType", l'attribut "MaxLength" reste indéfini.

2.3.3 Types numériques et unités

Le type numérique ("NumType") contient, en tant qu'attributs, les différentes indications prévues dans le langage INTERLIS.

L'unité ("Unit") et le système de référence sont affectés par le biais de relations ("NumUnit", "NumsRefSys"), pour autant qu'ils soient définis.

Les types formatés sont compris comme des types numériques particuliers, étant donné qu'ils reposent sur l'assemblage de champs numériques de structures. Si un type formaté décrit par exemple les heures, les minutes et les secondes, il peut aussi être compris comme un type numérique avec des secondes. Les heures et les minutes sont alors à convertir en conséquence. En revanche, si le type formaté décrit les mois et les jours, il est possible de le comprendre comme un type numérique avec des jours, mais sans connaissance précise de la conversion requise, tous les mois seront considérés comme comprenant 31 jours. Par suite, le type numérique considéré ne sera plus continu (attribut "Continuous").

2.3.4 Type booléen

Le format booléen est pris en charge par un type autonome ("BooleanType") et n'est pas considéré comme une énumération particulière, comme le suggère le manuel de référence.

2.3.5 Type de coordonnées

Le type de coordonnées ("CoordType") sert à la définition des types numériques associés à chacun des axes (relation "AxisSpecs") et du sens de rotation.

2.3.6 Enumérations

Chaque type d'énumération (classe "EnumType") est intégralement décrit et contient donc tous les noeuds (classe "EnumNode"). Un noeud artificiel est d'abord affecté au type d'énumération (sous le nom de "TOP")

Tous les noeuds effectifs sont alors directement et indirectement subordonnés à celui-ci (avec à chaque fois la valeur d'énumération comme nom). Dans le cas de types d'énumération ayant fait l'objet d'une extension, le type d'énumération et tous les noeuds repris sont liés au métaélément hérité correspondant (relation "Inheritance"). Si plus aucun sous-noeud ne peut être ajouté à un noeud donné, la valeur "true" est affectée à l'attribut "ExtendableME.Final". L'attribut "ExtendableME.Abstract" a toujours la valeur "false".

Le type qui admet tous les noeuds et pas seulement les feuilles comme valeur (cf. manuel de référence INTERLIS 2, § 2.8.2) est lié au type d'énumération effectif via la relation "TreeValueTypeOf".

2.3.7 Domaines de valeurs de classes et d'attributs

Le type pour le domaine de valeurs des classes (classe "ClassRefType") est une extension de la classe "ClassRelatedType", aussi utilisée dans le cadre de relations et de références où elle sera décrite plus avant.

Le type pour le domaine de valeurs des attributs (classe "AttributeRefType") peut renvoyer via la relation "ARefOf" et selon la situation, vers une classe (classe "Class"), une référence de classe (classe "ClassRefType") ou un argument (classe "Argument"). Les types admissibles pour l'attribut considéré découlent de la relation "ARefRestriction".

2.3.8 Types d'objets

Des objets isolés ou des ensembles d'objets peuvent être des arguments de fonctions. La description s'effectue via le type d'objet ("ObjectType") qui est une extension de la classe "ClassRelatedType". L'attribut "Set" sert à indiquer si un ensemble d'objets ou un objet isolé est attendu.

2.3.9 Unités

Les unités sont décrites au moyen d'objets de la classe Unit. Dans le cas d'unités abstraites, l'attribut Name prend le nom de l'unité pour valeur. Dans le cas d'unités concrètes, l'attribut Name prend l'abréviation de l'unité pour valeur. Le nom complet est décrit au moyen d'un élément DocText, dans l'intérêt de la capacité à le traduire, contenant le nom "FullName" et le nom complet comme texte.

La définition d'unités déduites ou composées est décrite à l'aide d'une expression. Seul un nombre réduit de genres de facteurs peut toutefois y survenir (références d'unités, constantes, appels de fonctions).

2.4 Domaines de valeurs génériques

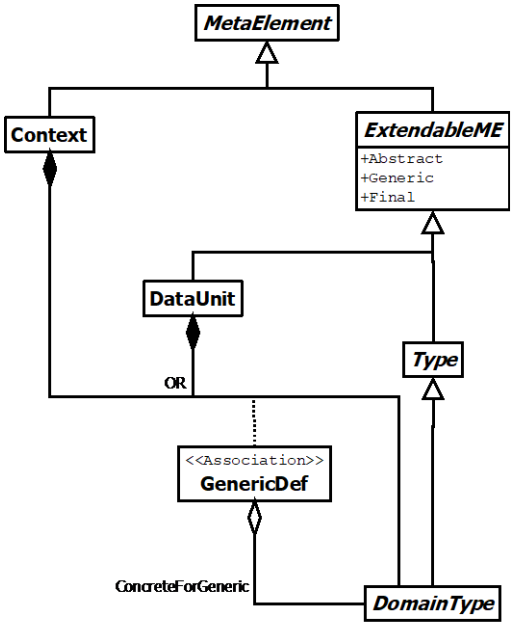

Figure 5 : domaines de valeurs génériques

Si un domaine de valeurs est générique, il en est fait état dans la super-classe "ExtendableME" via Generic = #true. Bien que des domaines de valeurs génériques soient uniquement permis pour des coordonnées dans INTERLIS2, il est renoncé à cette restriction dans le métamodèle.

Dans le cadre de contextes, des domaines de valeurs concrets peuvent être définis pour des domaines de valeurs génériques. Pour chaque domaine de valeurs générique pour lequel des définiitons concrètes sont effectuées, il existe une instance de la relation "GenericDef" appartenant par composition à l'instance du contexte et renvoyant vers le domaine de valeurs générique. Les domaines de valeurs concrets définis par l'intermédiaire de ce contexte sont attribués à cette instance via la relation "ConcreteForGeneric".

La même structure logique est réutilisée pour que les domaines de valeurs concrets applicables à des domaines de valeurs génériques soient parfaitement clairs pour une unité de données (DataUnit) spécifique : pour chaque domaine de valeurs générique rencontré dans la DataUnit, il exisite une instance de la relation, attribuée à la DataUnit par composition, et renvoyant vers le domaine de valeurs générique. Les domaines de valeurs concrets, applicables à cette DataUnit en raison des contextes qui lui sont associés, sont attribués à cette instance de la relation via "ConcreteForGeneric". Si plusieurs domaines de valeurs concrets sont définis, celui qui s'applique aux données n'est spécifié que dans les données de transfert.

2.5 Vues et graphique

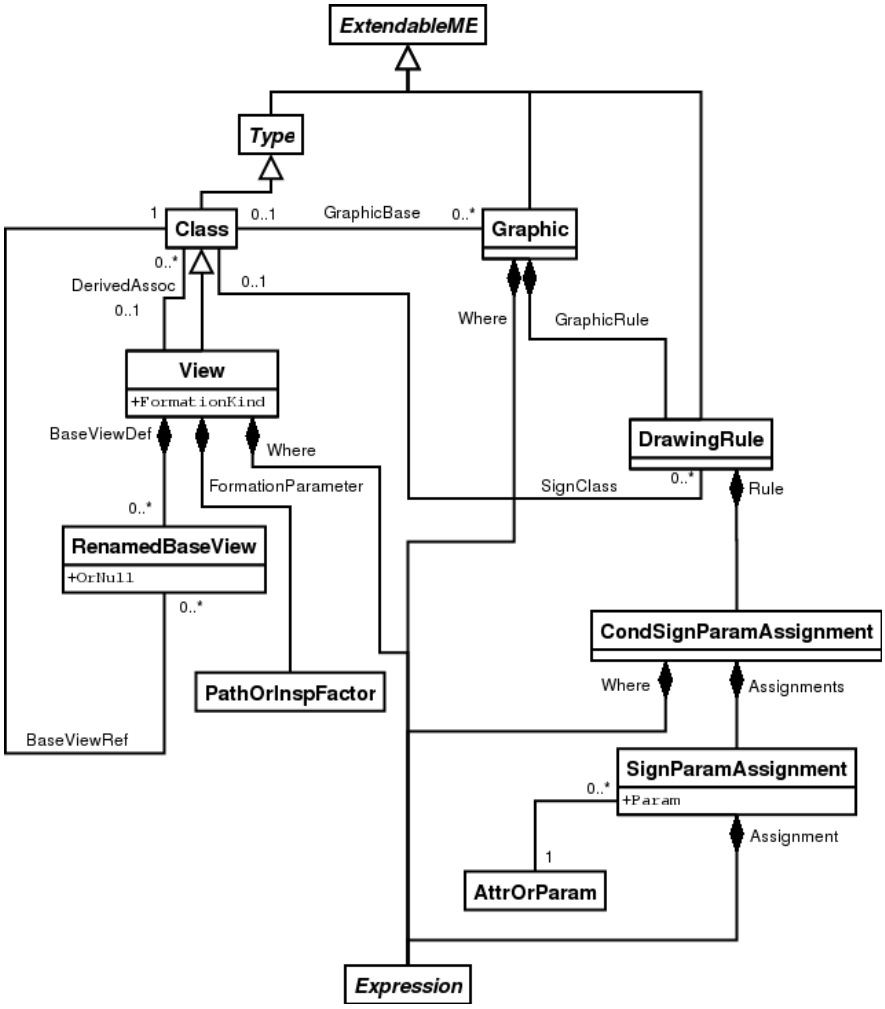

Figure 6 : vues et graphique

La description d'une vue (classe "View") est une extension de "Class" qui fournit la description des attributs. La manière dont la vue est formée est décrite à l'aide d'objets de la classe View et les classes de base apportant des contributions à la vue sont précisées (via "RenamedBaseView").

D'autres paramètres pour la formation de la vue (cf. manuel de référence INTERLIS 2 § 2.15) ne sont indiqués que pour l'agrégation (pour autant qu'EQUAL soit requis) et pour l'inspection en vue de la désignation de l'attribut à inspecter ("FormationParameter"). Si une association est de type "Outer-Join", la valeur "true" est affectée à l'attribut "OrNull" dans le cadre de la définition correspondante de la classe de base (classe "RenamedBaseView").

Dans le cas d'une agrégation, l'attribut implicite "AGGREGATES" est formé (avec ce nom). Son type est une MultiValue dont le type de base est la classe de base de l'agrégation.

Si une vue est étendue, les définitions des classes de base ne sont pas reprises dans l'extension, seules d'éventuelles définitions complémentaires étant ajoutées.

La description du graphique suit largement la structure du langage INTERLIS. Les constituants requis pour l'affectation ont été créés dans le cadre des facteurs et des expressions. La question de savoir si les constituants spéciaux doivent aussi être utilisés pour d'autres domaines que le graphique reste donc ouverte.

2.6 Autres constituants

2.6.1 Expressions et facteurs

Les facteurs et les expressions ne sont pas définis comme des objets autonomes (métaéléments) mais comme des structures. Les différents constituants correspondent assez directement aux constituants du langage.

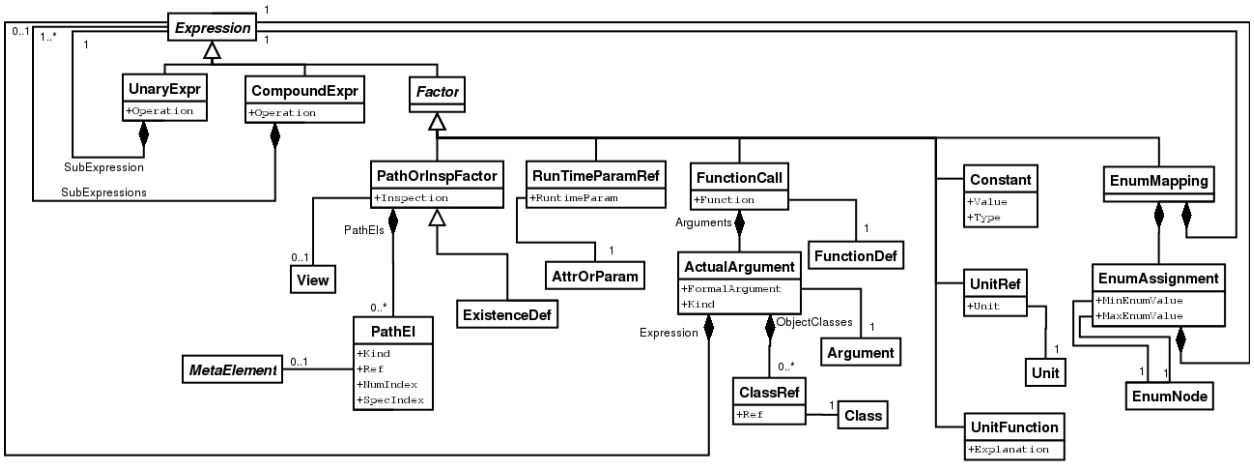

Figure 7 : expressions et facteurs

Dans le cas des expressions, une distinction est établie entre :

- "UnaryExpr" : les expressions pour lesquelles une opération (Not, Defined) se rapporte à une sous-expression ;
- "CompoundExpr" : les expressions pour lesquelles une opération (And, Or, comparaisons, InRange, CondExpr) se rapporte à un ensemble de sous-expressions ;
- "Factor" : les facteurs qui établissent le lien avec les éléments de modèle.

Remarques spéciales sur l'objet "Factor" :

- la référence de l'unité (structure UnitRef) n'est employée que dans le cadre d'unités dérivées et composées ;
- la projection (structure EnumMapping) de valeurs d'énumération sur d'autres valeurs, est avant tout utilisée dans le cadre du graphique.

2.6.2 Conditions de cohérence

Les conditions de cohérence sont modélisées comme des sous-classes de la classe "Constraint", laquelle est à son tour une sous-classe de la classe "MetaElement". Les instances de la condition de cohérence appartiennent par composition, via la relation "ClassConstraint" ou "DomainConstraint", à l'élément (classe ou domaine) qu'elles précisent Les différentes sous-classes de la classe "Constraint" correspondent assez directement aux constituants du langage.

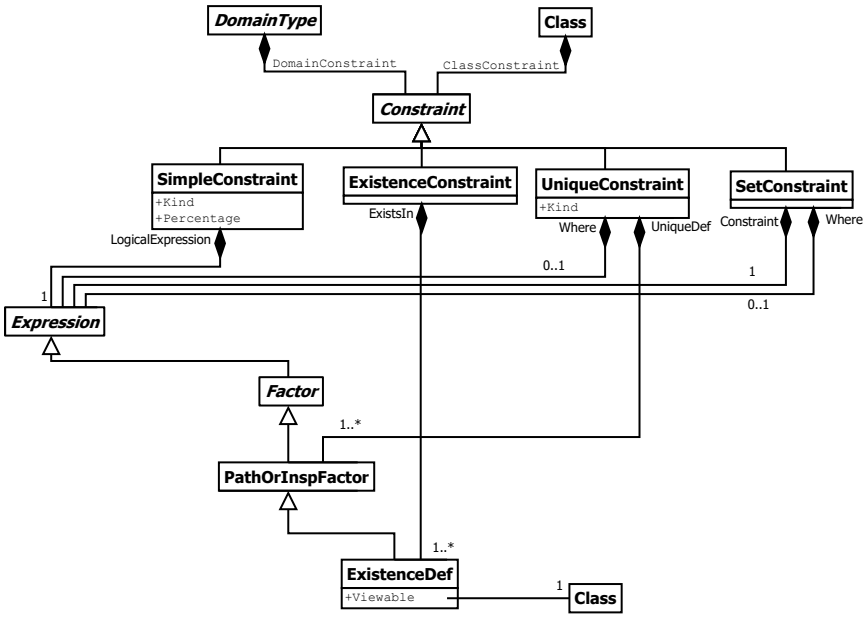

Figure 8 : conditions de cohérence

2.6.3 Définition de fonctions

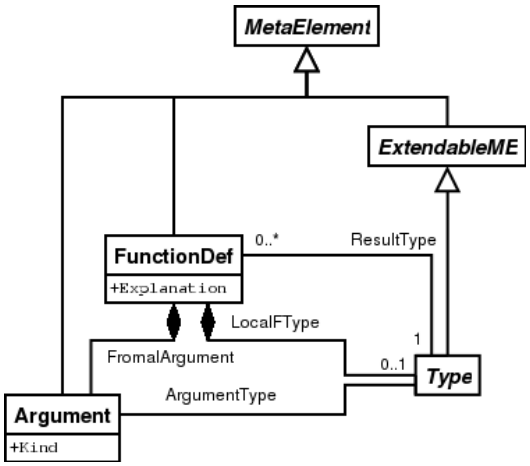

Figure 9 : définition de fonctions

A une définition de fonction (classe "FunctionDef") est associé le type du résultat via la relation "Result". Les arguments sont affectés par composition à la définition de la fonction (relation "FormalArgument"). Le chemin du nom suivra le modèle …NomFonction.NomArgument. Si des types du résultat ou des arguments sont définis localement, ils sont affectés par composition à la définition de la fonction (relation "LocalFType"). Le chemin du nom suivra le modèle …NomFonction.TYPE pour le type de résultat (TYPE étant un mot-clé du langage, aucun nom d'argument de cette nature n'est possible) ou …NomFonction.NomArgument.TYPE.

Les arguments sont liés à leur type via la relation ArgumentType.

2.6.4 Métaobjets

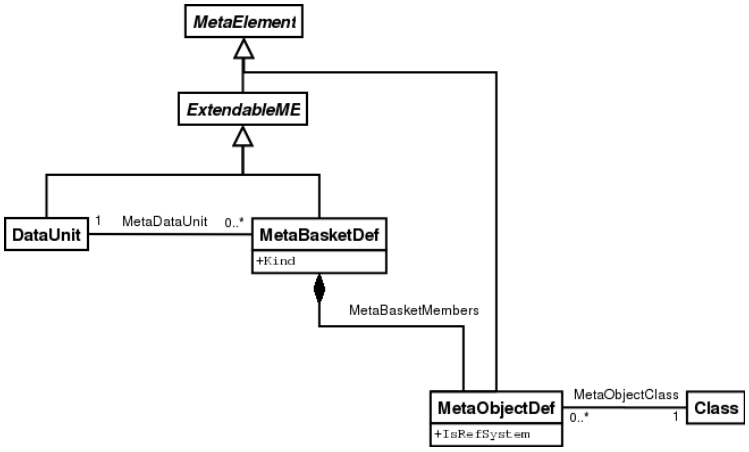

Figure 10 : métaobjets

Les classes "MetaDataBasketDef" et "MetaObjectDef" servent à la description des conteneurs de métadonnées et de métaobjets.

Le nom de la description des conteneurs de métadonnées correspond à celui du conteneur. Les métaobjets attendus dans ce conteneur sont affectés via la composition MetaBasketMembers.

2.6.5 Paramètre d'exécution

Les paramètres d'exécution (cf. manuel de référence INTERLIS 2, § 2.11) sont représentés par des objets de la classe AttrOrParam, lesquels sont directement affectés au paquet concerné.

3 Répartition des données de modèle en conteneurs

Afin que les données de modèle puissent être utilisées simplement de manière sélective, un conteneur propre (basket) est constitué par modèle (original ou traduction). L'identification de ce conteneur se compose du texte constant "MODEL" suivi d'un point et du nom du modèle. Il est ainsi garanti que l'identification du conteneur ne coïncide pas avec une identification d'objet.

4 Le métamodèle INTERLIS en 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.
```