INTERLIS 2 –

Manuel de référence

Edition du 2003-05-13 (français)

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

Ce manuel de référence 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

Liste des figures

Avant-propos

Le présent manuel de référence s'adresse aux spécialistes familiers des systèmes d'information, en particulier des systèmes d'information géographique et d'information du territoire. Il devrait notamment intéresser les décideurs soucieux de gérer soigneusement leurs données.

C'est en 1991 que le document "INTERLIS - mécanisme d'échange de données pour systèmes d'information du territoire" a paru pour la première fois. L'objectif majeur et le but d'INTERLIS sont la description la plus précise possible de données. Le mécanisme en question comprend un langage de description conceptuel et un format de transfert séquentiel spécialement orienté vers les données géoréférencées (en un mot, géodonnées). Cette configuration permet une compatibilité entre les systèmes et une disponibilité à longue échéance, plus précisément l'archivage et la documentation des données. Une utilisation rationnelle d'INTERLIS dans les processus de décision, de planification et de gestion s'avère d'une grande utilité. Souvent, ce langage permet des économies considérables, du fait par exemple d'une utilisation multiple et d'une documentation claire des données.

Cinq ans après sa publication, INTERLIS, que nous appellerons rétroactivement version 1, ci-après INTERLIS 1, est sorti de son apparente léthargie. Toute une palette d'outils logiciels a fait son apparition dans l'intervalle. Avec ces derniers, les utilisateurs peuvent transformer des géodonnées décrites et codées en INTERLIS. Certes le langage en question a été créé parce que la mensuration officielle en avait besoin mais la gamme de ses utilisations dépasse et de loin le cadre de cette spécialité. Nous en voulons pour preuve le nombre de modèles de données et de projets, bien plus d'une centaine, recourant à INTERLIS dix ans après sa publication. Et la version de base "INTERLIS 1", sous forme de norme suisse SN 612030, continuera longtemps encore à rendre de précieux services, parallèlement à ses versions ultérieures.

Vu les exigences accrues posées par les utilisateurs, diverses extensions d'INTERLIS 1 s'avéraient indispensables comme la livraison incrémentielle de données actualisées, l'orientation structurelle objet ou encore la description formelle de la représentation graphique d'objets. 1998 a marqué les débuts d'un processus de plusieurs années auquel ont pris part une demi-douzaine de spécialistes de la recherche, de l'administration, du conseil et de l'industrie du logiciel et qui a débouché sur un consensus. Il a engendré une "extension" d'INTERLIS 1 et, du même coup, une synthèse de ces nouveaux concepts.

Le manuel de référence INTERLIS 2 n'intègre à nouveau que ce qui s'est avéré strictement nécessaire ; des exemples et des figures ne sont donnés que lorsqu'ils complètent judicieusement le texte volontairement concis. De cette manière, la spécification reste claire et facile à implémenter. Si quelques éléments linguistiques comme des vues ou des représentations graphiques semblent complexes, cela ne tient de toute évidence pas à INTERLIS mais à une situation effectivement difficile. Pour remédier à ce genre de problème, rien de tel que de donner de bons exemples, que de consacrer du temps à la formation et au perfectionnement et que de proposer des "profils", autrement dit des mélanges bien dosés des capacités d'INTERLIS en tant qu'outil.

Nous conseillons à tout lecteur de lire au moins le chapitre 1, consacré aux principes fondamentaux, pour s'assurer une bonne compréhension des concepts de base d'INTERLIS 2.

Extensions d'INTERLIS 2 par rapport à INTERLIS 1

A quelques rares exceptions près, le langage de description existant INTERLIS 1 n'est pas modifié, mais uniquement étendu. L'extension a par exemple concerné les possibilités de relations entre objets à décrire (relations vraies en tant que classe d'association et attributs de référence avec REFERENCE TO. Attention : une signification différente a été affectée à la syntaxe "->" de l'attribut relationnel d'INTERLIS 1), en veillant toutefois à ce que le passage de la version 1 à 2 d'INTERLIS se fasse sans complication inutile (cf. paragraphe 2.7 Relations vraies). Dorénavant, des relations vraies et des attributs de référence peuvent renvoyer à des objets d'autres conteneurs sous certaines conditions bien spécifiques. Le terme

de conteneur est introduit et recouvre la notion bien connue de l'organisation des objets (c.-à-d. de données décrivant les objets du monde réel, également appelées instances objets ou simplement instances) dans une banque de données. Le terme de table (TABLE) devient par exemple "classe" (CLASS) du fait de la transition du formalisme relationnel vers le formalisme orienté objet. Sans autre indication, un attribut est facultatif (OPTIONAL disparaît) et il faut indiquer s'il est obligatoire (MANDATORY). Par ailleurs, la désignation de l'unicité (IDENT) a aussi un nouveau nom (UNIQUE). Les nouveaux concepts orientés objet englobent l'héritage notamment de thèmes, de classes, de vues, de représentations graphiques et de domaines de valeurs. D'autres extensions d'importance sont des types de données de quantités (LIST, BAG), les conditions de cohérence, les vues de données, les représentations graphiques, des descriptions d'unités, de méta-objets (systèmes de coordonnées et signature graphique) et la livraison incrémentielle des données. Des extensions spéciales, spécifiques de l'utilisation, peuvent en outre être définies comme des fonctions et d'autres géométries linéaires. Mais pour ce faire, il faut passer des contrats avec les fournisseurs d'outils.

L'eXtensible Markup Language (XML) reprend le codage pour le format de transfert INTERLIS 2, ce qui constitue une nouveauté. Du fait d'une prévisible diffusion internationale, nous nous attendons à une bonne acceptation de ce format et à un grand nombre de produits logiciels compatibles.

En définitive, peu de changements sont à noter pour un utilisateur familier d'INTERLIS 1, aussi longtemps qu'il n'envisage pas de recourir aux nouveaux concepts tels que l'orientation objet ou la représentation graphique: les connaissances déjà acquises lui permettent donc de travailler avec INTERLIS 2. Des outils tels que le compilateur INTERLIS 2 disponible gratuitement l'aident à franchir le cap de la transition. Les producteurs ayant déjà veillé à la souplesse des possibilités de configuration et au respect des règles du développement logiciel (par exemple la modularité et l'abstraction) lors de l'implémentation d'INTERLIS 1 pourront conserver le bénéfice des investissements consentis; des bibliothèques de programmes disponibles gratuitement permettront en effet aux producteurs de logiciels de se concentrer sur la connexion de leurs systèmes à INTERLIS 2.

Perspectives

INTERLIS 1 est apparu à un moment où le langage relationnel de définition des données SQL-92 par exemple n'était pas encore standardisé et où l'on ne parlait quasiment pas encore d'orientation objet. Josef Dorfschmid, le père d'INTERLIS 1, a intégré quelques-uns de ces concepts dans le langage en faisant preuve d'un bel esprit d'anticipation. Avec le processus de remaniement, l'intégration orientée objet et divers concepts informatiques spécifiques, INTERLIS a atteint une nouvelle maturité. INTERLIS 2 peut donc aujourd'hui – et non pas demain seulement – être utilisé comme un outil performant.

Nous sommes conscients que la recherche du langage universel de description des données n'est pas encore menée à son terme. Mais tout atermoiement aurait des conséquences analogues à la gestion non durable des ressources de notre planète. INTERLIS mérite de ne pas servir uniquement de format d'échange pour s'inscrire résolument dans la durée : avec INTERLIS, l'exigence d'une utilisation durable de la technique s'est fait un nom !

Chaque langage doit être appris et cet apprentissage doit se fonder sur une méthode. Il est donc clair que quelques manuels de l'utilisateur doivent venir compléter le présent manuel de référence.

Remerciements

Responsable du développement d'INTERLIS et principal rédacteur du présent document, Stefan Keller (Hochschule für Technik Rapperswil, auparavant Direction fédérale des mensurations cadastrales) a d'emblée supervisé les travaux relatifs à INTERLIS 2, pleinement avant son départ pour Rapperswil puis en très grande partie une fois à la Haute Ecole Spécialisée. Qu'il soit chaleureusement remercié pour la constance de son engagement. A ces remerciements, il convient également d'associer les "piliers" de

l'équipe INTERLIS 2, pour la remarquable qualité de leur travail. Cette équipe comprenait Joseph Dorfschmid (Adasys AG), Michael Germann (infoGrips GmbH), Hans Rudolf Gnägi (Ecole polytechnique fédérale de Zurich), Jürg Kaufmann (Kaufmann Consulting), René L'Eplattenier (Service de l'aménagement du territoire du canton de Zurich), Hugo Thalmann (a/m/t software service AG) ainsi que Sascha Brawer (Adasys AG), Claude Eisenhut (Eisenhut Informatik) et Rolf Zürcher (COSIG) qui en assurait la coordination.

Dès le début 2002, la responsabilité de la rédaction du manuel de référence a été confiée à la COSIG, coordination de l'information géographique et des systèmes d'information géographique de la Confédération, laquelle reste toutefois en contact étroit avec la Direction fédérale des mensurations cadastrales. Peu après ce transfert de compétence, l'Association suisse de normalisation (SNV) a lancé une procédure de consultation officielle relative à INTERLIS 2. Elle a abouti à l'enregistrement de 109 commentaires, tous examinés par les membres de l'équipe au cours des mois suivants, lesquels leur ont, le cas échéant, réservé une suite favorable. Il en a résulté quelques modifications importantes du langage par rapport à la version de consultation 2.1 du 17 octobre 2001, avec pour conséquence qu'INTERLIS 2 portera le numéro de version interne 2.2. Fin novembre 2002, l'acte final du INB/TK 151 a déclaré INTERLIS 2 comme norme SN 612031. Après quelques dernières révisions textuelles, le manuel de référence peut être dès lors transmis au public

Aucune norme ne peut être définie par quelques individus isolés, son élaboration nécessite toujours le concours d'un grand nombre d'experts. Nos remerciements s'adressent donc à l'ensemble de ces professionnels qui vont dès lors insuffler un esprit d'innovation à leurs produits.

Wabern, avril 2003

Rolf Zürcher

1 Principes fondamentaux

1.1 Présentation générale

INTERLIS favorise l'interaction de systèmes d'information, en particulier de systèmes d'information géographique ou de systèmes d'information du territoire. Comme son nom l'indique, INTERLIS est entre (**INTER**) les "**L**and-**I**nfomations-**S**ystemen", terme allemand pour systèmes d'information du territoire. Tous les systèmes actuels perçoivent clairement les concepts de collaboration et d'interopérabilité.

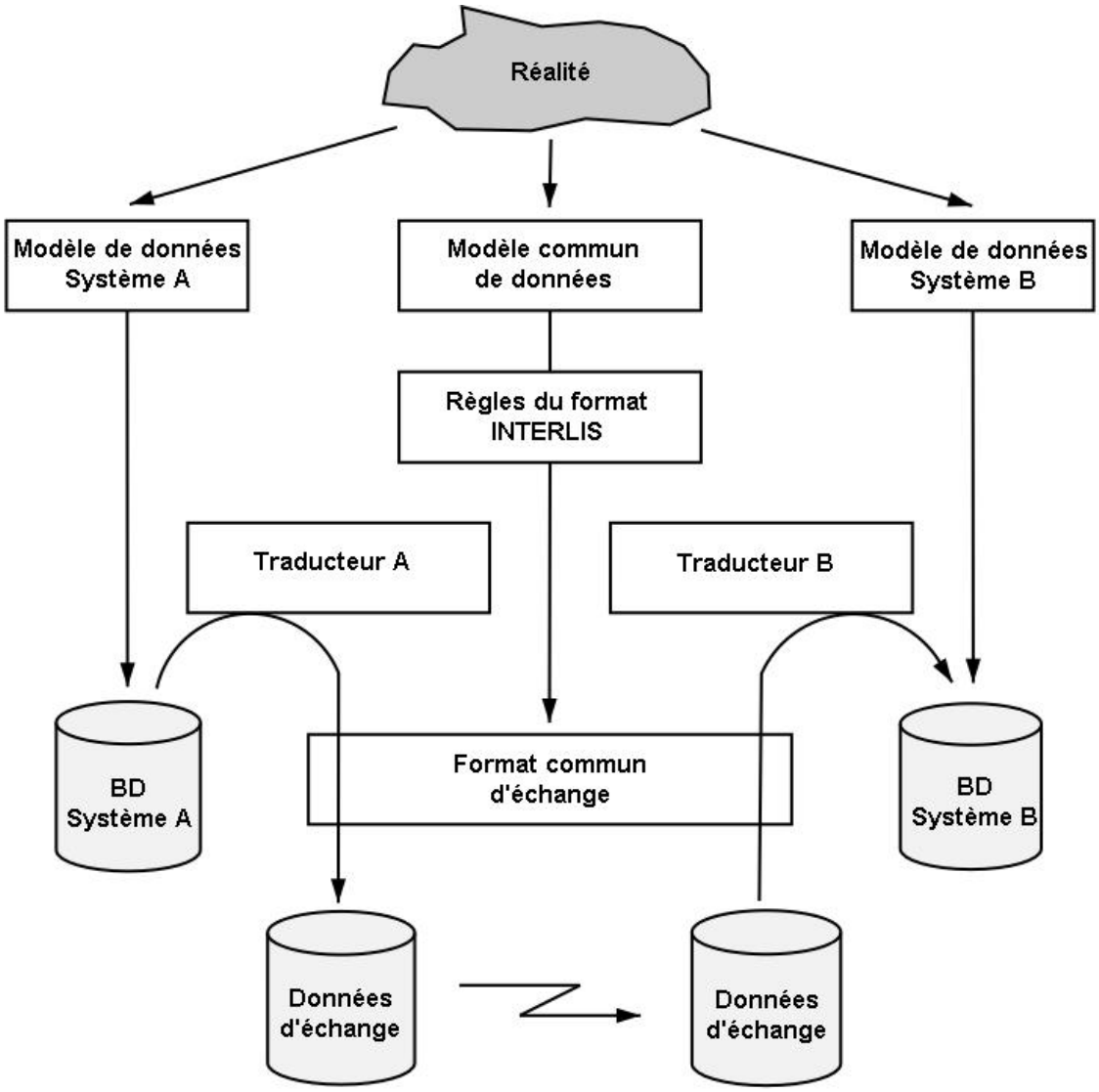


Figure 1 : Transfert de données entre plusieurs bases de données à l'aide d'un schéma commun de données (modèle de données) décrit avec un langage commun de description des données.

Pour cette raison, INTERLIS propose un langage de description conceptuel des données permettant de décrire et de modéliser le monde réel concerné par un domaine d'application défini. Une telle abstraction est appelée un "schéma conceptuel" ou un "modèle conceptuel" de données, ou plus simplement un

modèle ou un schéma. Un nombre restreint de concepts de signification précisément définie décrit (ou modélise) des classes d'objets avec leurs propriétés et leurs relations. Le langage de description d'INTERLIS permet par ailleurs d'introduire des classes d'objets déduits, définies sous forme de vues sur d'autres classes d'objets. Des classes d'objets vraies comme des classes d'objets déduites peuvent servir de base à des représentations graphiques, INTERLIS garantissant la séparation stricte entre la représentation graphique et la description de la structure de données fondamentale (modèle des objets).

INTERLIS n'est pas orienté vers une application spécifique. Le développement d'INTERLIS est axé sur des principes généraux orientés objet en veillant toutefois à ce que les concepts importants pour les systèmes d'information géographique soient particulièrement bien étayés. Ainsi des coordonnées, des lignes et des surfaces sont des types de données formant les éléments de base d'INTERLIS. De plus, des éléments syntaxiques pour décrire la précision des mesures et des unités sont à votre disposition. Mais il est également possible d'utiliser INTERLIS pour des applications non géographiques.

Les aspects fondamentaux d'un domaine d'application peuvent être décrits dans un modèle de base. Ensuite ce modèle est précisé et spécialisé, par exemple selon les besoins spécifiques d'un pays, à la faveur d'autres étapes, en tenant compte des niveaux d'une région déterminée (comme un canton, une région ou une commune, cf. Figure 2). Comme outil de spécialisation, INTERLIS 2 propose les concepts orientés objet d'héritage et de polymorphisme. On s'assure de la sorte que des définitions déjà abouties ne doivent pas être répétées ou ne soient pas remises en question par erreur.

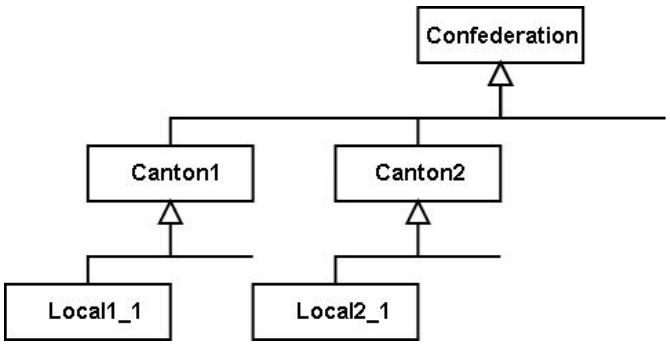


Figure 2 : Spécialisation de la modélisation d'un concept, du niveau de la Confédération à l'échelon local en passant par celui du canton (spécifique au pays).

Des applications importantes n'ont pas besoin d'être définies dans un modèle unique. Elles peuvent être subdivisées en descriptions partielles (modèles, schémas). Une description partielle peut englober plusieurs thèmes. Par souci de lisibilité des modèles, il est aussi possible de définir des modèles comme de pures traductions d'autres modèles.

1.2 Utilisation de modèles

Un modèle (ou schéma) INTERLIS se veut avant tout une aide à la compréhension pour l'utilisateur. Le langage est conçu de telle façon que la lecture d'un modèle soit aisée. Les modèles INTERLIS sont par ailleurs précis, dépourvus de toute ambiguïté et leur interprétation ne peut donner lieu à aucun malentendu. Ainsi, le langage textuel INTERLIS se présente-t-il réellement comme un complément indispensable du langage de description graphique UML (Unified Modeling Language, www.omg.org/uml).

Mais INTERLIS franchit un pas supplémentaire. Comme un modèle possède une signification formelle clairement définie, la mise en œuvre d'un processus dans un système informatique peut être tirée automatiquement de ce modèle (conceptuel). INTERLIS englobe par exemple un utilitaire de transfert basé sur XML dont les définitions sont produites à partir des modèles pertinents selon leurs règles.

L'utilisation d'une modélisation des données en étroite relation avec des services d'interface indépendants de tout système est appelée une *architecture guidée par modèle* (cf. "model-driven architecture" d'OMG, www.omg.org/mda).

Des modèles peuvent être développés en appui sur des concepts de base communs. Comme INTERLIS permet de décrire ceci de façon explicite par l'utilisation de l'héritage et du polymorphisme, on sait toujours quels concepts sont communs et lesquels sont spécifiques, ce qui revêt une grande importance dans l'optique de l'interopérabilité sémantique souhaitée. Par exemple, le fichier de transfert d'une commune peut être interprété sans difficulté par une unité administrative d'ordre supérieur (Confédération, canton) sans que les organismes concernés doivent pour cela s'entendre sur un modèle unique. Il suffit que chaque échelon construise son modèle sur celui de l'unité d'ordre supérieur.

Il est envisageable voire tout à fait souhaitable que de nouveaux services soient développés sur la base d'INTERLIS. Un compilateur INTERLIS 2 devrait considérablement faciliter cette tâche. Ce programme a pour objet de lire et d'écrire des modèles INTERLIS, de permettre leur modification et de vérifier que ces modèles se conforment aux règles syntaxiques et sémantiques propres à INTERLIS. Ce compilateur est en mesure, dans le respect du service de transfert INTERLIS actuel avec XML, de générer automatiquement, entre autres, des diagrammes XML (www.w3.org/XML/Schema) à partir de modèles INTERLIS. Ainsi, les données INTERLIS/XML concrètes peuvent-elles trouver un champ d'utilisation encore plus large grâce à des outils XML courants et appropriés. Ce compilateur INTERLIS est à disposition pour la création de nouveaux outils tant que les règles d'utilisation ne sont pas violées.

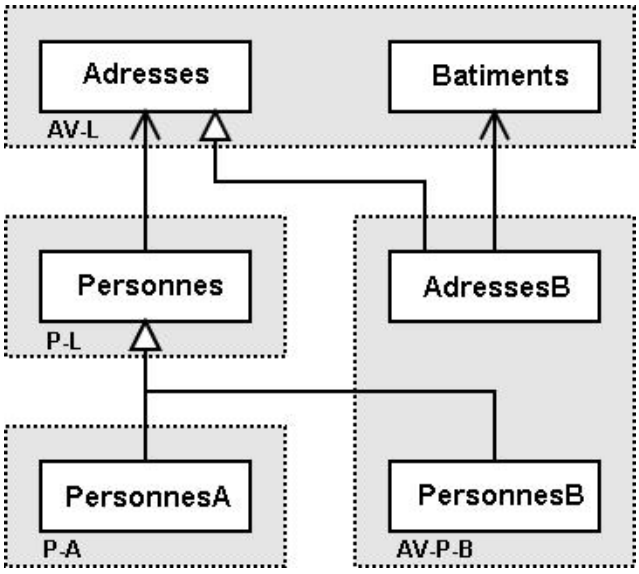


Figure 3 : Hiérarchie de transmission d'adresses, de personnes, de bâtiments.

1.3 Classification en modèles et thèmes

Un modèle (resp. schéma) décrit une représentation du monde telle qu'elle est importante dans l'optique d'une certaine application. Il est clos en lui-même mais peut utiliser ou étendre des parties d'autres modèles. Un modèle INTERLIS est donc d'une certaine manière comparable aux modules ou "packages" de bien des langages de programmation.

On peut d'abord faire la différence entre des modèles qui ne comprennent que des définitions portant sur des types (unités, domaines de valeurs, structures) et des modèles à propos desquels des données peuvent exister. Les descriptions à propos desquelles des données peuvent exister sont structurées en thèmes. Cette classification s'effectue en fonction de la logique suivante : montrer dans quelles

organisations et par qui les données sont gérées et utilisées. Des données, qui sont typiquement gérées et utilisées par des services différents, doivent aussi être définies selon différents thèmes. Ces derniers peuvent être dépendants les uns des autres. Mais de telles dépendances doivent se limiter au minimum. On évitera si possible des relations entre des thèmes dont les données sont gérées par des instances différentes, le maintien de la cohérence a un coût. Des dépendances cycliques sont en tout cas exclues. Outre les définitions de données à proprement parler, des thèmes peuvent comprendre des définitions pour des vues et des graphiques.

Un thème peut en étendre un autre. De cette manière tous les concepts du thème de base sont hérités. Ils peuvent être complétés ou précisés.

Un modèle de la mensuration officielle d'un pays pourrait par exemple comprendre notamment les thèmes "adresses" et "bâtiments" (cf. Figure 3). Ces concepts sont indépendants les uns des autres ; la relation est établie par algorithmes via des coordonnées. Des personnes ont des adresses si bien que le modèle des personnes de ce pays prend appui sur le modèle national de la mensuration officielle, le thème "personnes" dépendant du thème "adresses" du modèle de la mensuration.

Admettons que l'on veuille décrire les personnes d'une région avec plus de précision que le modèle pays - personnes ne le prévoie. Cette région A établit donc son propre modèle dans lequel le thème "personnes" est repris et précisé à partir du modèle "personnes" national.

La région B souhaite créer une relation explicite entre les bâtiments et les adresses et estime également que le modèle pays - personnes n'est pas assez précis. A nouveau les thèmes en question sont repris et précisés. Les deux précisions apportées sont réunies en un seul modèle - le modèle de la région B.

1.4 Concept-objet

1.4.1 Objets et classes

Un objet (également désigné par instance objet ou simplement instance) comprend les données d'un objet du monde réel. Normalement, de nombreux objets possèdent les mêmes propriétés et peuvent donc être réunis. Une quantité d'objets aux propriétés identiques s'appelle une classe. A chaque propriété correspond (au moins) un attribut. INTERLIS 1 utilisait le terme de table pour classe. Quantité d'entités, type d'entité ou encore feature (typ) sont autant d'autres termes employés pour classe.

La description d'une classe permet entre autres de fixer les propriétés ou caractéristiques des différents objets. Ces dernières sont appelées attributs. Les valeurs d'attributs des différents objets ne sont pas quelconques mais doivent remplir des conditions précises entrant dans la description d'un attribut.

Dans ce contexte, INTERLIS propose une palette de types de données fondamentaux (types de données de base : chaînes de caractères, types de données numériques, énumérations, coordonnées cartésiennes et elliptiques 2D ou 3D, lignes, surfaces) sur la base desquels il est possible de définir de nouvelles structures de données plus complexes. Pour préciser encore l'indication, des attributs numériques peuvent être dotés d'une unité de mesure et mis en rapport avec un système de référence ; les coordonnées peuvent être liées à un système de référence de coordonnées.

Outre ces types de données de base, un attribut peut également contenir des sous-structures. Les éléments constitutifs de la sous-structure sont à interpréter comme des éléments structurés n'existant qu'en relation avec l'objet principal auquel ils sont rattachés et ne pouvant être trouvés que par l'intermédiaire de celui-ci. L'organisation des éléments structurés est décrite de façon similaire à celle des classes et la définition d'une classe peut également intervenir dans ce cadre.

Hormis le fait que toutes les valeurs d'attributs doivent correspondre à leur type respectif, d'autres conditions peuvent être définies. INTERLIS établit une distinction entre les différents types de conditions de cohérence suivants :

- Ceux qui se réfèrent à un seul objet. Ils peuvent encore être subdivisés pour autant que chaque objet d'une classe respecte les conditions de celle-ci (conditions de cohérence "strictes"), et que les possibles violations de condition soient rares (conditions de cohérence "souples").
- Ceux qui exigent l'unicité (mono-valeur) de combinaisons d'attributs de tous les objets d'une classe.
- Ceux qui exigent l'existence d'une valeur d'attribut dans une instance d'une autre classe.
- Conditions plus complexes, lesquelles se réfèrent à des ensembles d'objets et sont formulées au moyen de vues.

1.4.2 Extension de classe et polymorphisme

Des classes sont soit indépendantes soit étendent une classe de base (spécialisation, héritage), ce qui revient à dire que la description d'une classe est soit indépendante soit contient des extensions d'une autre description transmise. Une extension de classe (aussi appelée sous-classe ou sur-classe) peut avoir des attributs et des contraintes additionnels ou rendre plus strictes des conditions héritées (types de données, contraintes).

Chaque objet appartient à une seule classe (on dit aussi : est instance objet ou instance d'une classe). Mais il remplit toujours toutes les conditions posées par les classes de base (c.-à-d. les sur-classes). Pour chaque classe, il existe un ensemble d'objets qui sont instances de cette classe ou de l'une de ses extensions. Dans le cas de classes concrètes, il existe un sous-ensemble d'instances, de taille normalement plus petite, n'appartenant qu'à cette classe.

Une classe étendue est *polymorphe* vis-à-vis de ses classes de base : partout où des instances d'une classe de base sont attendues, il est possible de trouver des instances d'une extension (polymorphisme de sous-ensembles ou principe de substitution). INTERLIS a été conçu de telle sorte qu'une lecture polymorphe soit toujours possible. Si une relation avec une classe est par exemple définie (cf. paragraphe 1.4.4 Relations entre objets), elle s'étend aux objets des extensions. L'écriture polymorphe intégrale ne fait pas partie des objectifs poursuivis par la présente version d'INTERLIS.

Les éléments de sous-structures ne constituent pas de véritables objets indépendants, mais des éléments structurés qui n'appartiennent par conséquent pas à l'ensemble des instances d'une classe quelconque.

1.4.3 Métamodèles et méta-objets

Les systèmes de référence et de coordonnées de même que les signatures graphiques se présentent comme des éléments du modèle (ou du schéma) du point de vue de l'application, pouvant intervenir dans les définitions des applications. Toutefois, comme différents systèmes de coordonnées et de référence mais surtout différentes signatures graphiques peuvent être décrits de la même manière, il est judicieux de définir également leurs propriétés à l'aide de classes, au sein de modèles. A chaque système ou signature graphique (par exemple un symbole ponctuel, un type de ligne) correspond donc un objet.

Les méta-objets sont à définir dans le cadre d'un métamodèle. Les objets utilisables doivent explicitement être identifiés comme des méta-objets (extensions de la classe prédéfinie METAOBJECT) et peuvent alors être référencés par l'application au moyen de leur nom. Il est nécessaire à cet effet que les méta-objets soient mis à la disposition de l'outil traitant la définition de l'application par l'intermédiaire de conteneurs (cf. paragraphe 1.4.5 Conteneurs, réplication et transfert de données).

1.4.4 Relations entre objets

INTERLIS 2 distingue (contrairement à INTERLIS 1) deux sortes de rapports entre les objets : une relation vraie et un attribut de référence.

On parle de *relation vraie* en présence d'un ensemble de paires d'objets (ou dans le cas général d'uplets à n objets). Le premier objet de chaque paire appartient à une première classe A, le second à une deuxième classe B. Dans ce cadre, l'attribution d'objets aux paires doit être prédéfinie, elle n'a donc qu'à être décrite, autrement dit modélisée. Comme on le montrera au paragraphe 1.5 Concept de vues, il est également possible, à l'opposé, de calculer des attributions de façon algorithmique, par exemple sur la base de valeurs d'attribut.

De telles relations sont décrites comme des constructions autonomes, appelées classes relationnelles (resp. classes d'associations), lesquelles peuvent aussi être étendues. INTERLIS 2 n'accepte pas seulement des relations duales mais permet encore des relations multiples et des relations avec des attributs propres. Ainsi, une classe relationnelle est aussi elle-même une classe d'objets.

Propriétés importantes de telles relations :

- *La cardinalité* – Combien d'objets de la classe B (ou A) peuvent être attribués à un objet de la classe A (ou B) à travers la relation ? Autrement dit, combien de paires d'objets peut-il y avoir avec un objet défini à la première place (ou à la deuxième) ? Cas général : combien de n-uplets peut-il y avoir avec des objets déterminés à n-1 positions?
- *L'intensité* – INTERLIS 2 établit une distinction entre l'association, l'agrégation et la composition. L'adressage direct des objets intervenant dans ces relations est possible dans tous les cas. S'agissant de l'association et de l'agrégation, les objets concernés existent indépendamment l'un de l'autre. Il existe une asymétrie entre les classes participantes dans les cas de l'agrégation et de la composition : les objets de l'une des classes (la sur-classe) sont appelés le tout (ou sur-objets) et les objets de l'autre classe (la sous-classe) les parties (ou sous-objets). Pour l'association, les objets disposent des mêmes droits et sont liés entre eux par des liens faibles. Sur le plan conceptuel, l'agrégation et la composition sont des relations orientées : plusieurs parties (sous-objets de la sous-classe) sont affectées à un tout (sur-objet de la sur-classe). Dans le cas de l'agrégation, au contraire de l'association, la copie d'un tout entraîne également celle de toutes les parties assignées, tandis que les parties restent conservées si le tout est supprimé. En revanche, la suppression du tout dans le cadre d'une composition entraîne la suppression des parties. Le paragraphe 2.7.2 Intensité de la relation décrit le comportement des relations avec d'autres objets lors de la copie. Attention : les sous-objets (parties) de compositions sont des objets identifiables au contraire des éléments structurés des sous-structures.
- *Le rôle* – Quelle signification est attachée aux classes participantes du point de vue de la relation ? C'est précisément ce que le rôle sert à définir pour chacune des classes participantes.

Un *attribut de référence* permet de créer le lien unissant un objet ou un élément structuré à un autre objet. Une telle relation n'est connue que de l'objet "référençant" et donc ignoré de l'objet référencé. Elle est par conséquent unilatérale.

Il est possible, sans violer l'indépendance des thèmes, de définir des relations (c.-à-d. des relations vraies et des attributs de référence) au moyen d'une identification spécifique (EXTERNAL), établissant ainsi un lien vers des objets *d'un autre conteneur* d'un même ou d'un autre thème. A condition, toutefois, que la structure dans laquelle la relation est définie, appartienne à un thème qui dépende de celui de la classe vers laquelle il renvoie.

Afin que règne un ordre clair, les relations ne peuvent se rapporter qu'à des classes déjà définies à l'endroit où la classe relationnelle (ou l'attribut de référence) est à définir.

1.4.5 Conteneurs, réplication et transfert de données

On appelle conteneur un ensemble fini d'objets appartenant à un thème ou à ses extensions. Fini veut dire que le conteneur doit comprendre tous les objets qui sont en relation au sein du thème.

Typiquement, un conteneur contient tous les objets d'une certaine région (par exemple d'une commune, d'un canton ou d'un pays tout entier). Un conteneur peut notamment comprendre également des données de diverses extensions (par exemple de divers cantons avec des extensions propres). Cela présuppose toutefois que dans le cadre d'une telle communauté de transfert, les désignations des modèles, thèmes et extensions de thèmes soient univoques.

Dans le cadre de conditions de cohérence, on parle souvent de "tous les" objets d'une classe. D'un point de vue conceptuel, on entend aussi par-là, fondamentalement, tous les objets qui ont la propriété voulue et qui existent simplement, autrement dit au-delà du conteneur. Il est toutefois manifeste pour des raisons diverses (efficacité, disponibilité, autorisation d'accès, etc.) qu'un réexamen n'est possible que dans des conteneurs accessibles localement. Dans le cas de conteneurs avec des métadonnées, les conditions de cohérence ne valent explicitement qu'au sein d'un conteneur car on peut admettre que les métadonnées ont un caractère descriptif et que donc, elles doivent être disponibles à chaque fois de façon complète (quasiment comme bibliothèque) dans un conteneur.

- On distingue différentes sortes de conteneurs : les *conteneurs de données* comprennent les instances de classes du thème.
- Des conteneurs de vues comprennent les instances de vues du thème.
- Des conteneurs avec les données de base pour le graphique englobant des instances de toutes les données ou vues qui sont utiles pour les graphiques du thème. Cela permet d'alimenter un convertisseur graphique.
- Des conteneurs avec les éléments graphiques : ils contiennent les instances de tous les objets graphiques (=signatures) qui sont utiles d'après les graphiques du thème. Cela permet d'utiliser un outil de représentation graphique (Renderer) (cf. Figure 5).

INTERLIS ne règle pas les modalités de conservation des objets dans les systèmes. Les réglementations ne concernent que l'interface entre les systèmes. A l'heure actuelle, une interface de transfert pour le transfert de conteneurs est définie comme fichier XML, permettant non seulement le transfert complet de tout le conteneur mais encore la mise à jour incrémentielle des données.

Dans le cas d'un transfert complet, on considère que le récepteur constitue des copies des objets nouvelles et indépendantes ne présentant aucun lien immédiat avec l'objet initial. En conséquence, les objets n'ont donc à être pourvus que d'une identification de transfert temporaire (abrégée en TI pour Transferidentifikator) dans le cadre du transfert complet. Cette identification est utilisée pour le transfert de relations.

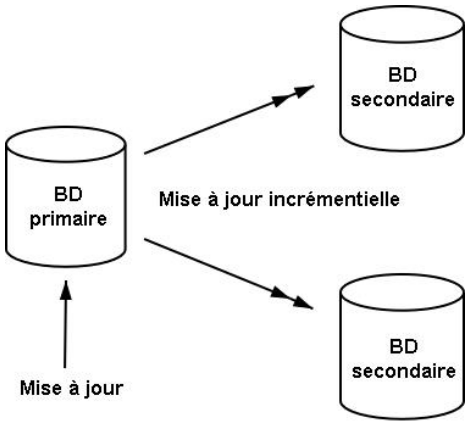


Figure 4 : Mise à jour dans la base de données primaire puis livraison complémentaire à la base de données secondaire (une flèche double indique une livraison incrémentielle).

Avec le transfert incrémentiel de données, on admet que l'émetteur fournit un état initial d'un conteneur de données (d'autres genres de conteneurs sont exclus) puis fournit des mises à jour incrémentielles au destinataire permettant à ce dernier d'actualiser ses données (cf. Figure 4). Les objets sont donc répliqués et conservent la relation avec l'objet originel, ce qui signifie qu'ils ne peuvent pas être modifiés indépendamment de l'original et qu'aucune nouvelle identification ne leur est associée.

A cet égard, on admet qu'entre les exploitants des banques de données primaire et secondaires, des accords contractuels sont passés (notamment ampleur et fréquence de la livraison ultérieure), accords qu'il est à l'heure actuelle impossible de décrire avec des instruments d'INTERLIS. Pour la livraison de données actualisées vraie, INTERLIS met toutefois les moyens requis à disposition. Des nouveaux objets sont communiqués comme lors du premier transfert. Un OID (identificateur d'objet) univoque leur est attribué. Il doit être conservé dans le temps. En cas de modification, référence est faite à cet OID univoque et tous les attributs de l'objet (y compris tous les éléments structurés des attributs structurés) livrés à nouveau. Des suppressions sont communiquées de la même manière. A cet égard, celui qui émet a la responsabilité première de la cohérence des objets (exemple : respect des conditions de cohérence, rectitude et cardinalité de relations). Pour ce faire, il annonce au destinataire les objets qui ont été modifiés ou supprimés et ceux qui ont été nouvellement créés. Avec des attributs de référence dépassant le thème, on ne part pas de l'idée – dans l'intérêt de l'autonomie des thèmes – que l'intégrité est conservée en tout temps. C'est l'affaire du destinataire de s'accommoder du fait que des incohérences provisoires existent entre thèmes de base et thèmes dépendants, autrement dit qu'un objet référencé n'existe pas.

Le cadre dans lequel l'unicité de l'OID est garantie n'est pas déterminé par INTERLIS directement. L'annexe E décrit une possibilité de structurer un tel OID. D'autres possibilités existent cependant. L'important est que l'OID soit toujours univoque dans le cadre d'une communauté de transfert, dans laquelle les différentes personnes qui saisissent des données appliquent correctement la règle de structuration de l'OID. Suivant la structure de l'OID, la mise à jour incrémentielle de données est possible dans un cadre élargi (par exemple à l'échelon planétaire) ou restreint (par exemple dans une organisation interne). La procédure de détermination de l'OID ainsi choisie définit la communauté potentielle de transfert.

Un objet ne peut être modifié que dans son conteneur originel, ou au dehors, qu'avec la permission de ceux qui le gèrent. Tous les autres conteneurs secondaires ne peuvent modifier un objet qu'à la suite d'une livraison de données actualisées. INTERLIS 2 exige donc que – dans le cadre de livraison incrémentielle -, outre les objets, les conteneurs soient univoques et durablement identifiables. Pour cette raison les conteneurs comprennent aussi un OID. Dans le cas d'un transfert complet, les conteneurs aussi ont besoin d'un seul identificateur de transfert (TID). Lorsque la distinction avec l'OID (resp. le TID) normal est prépondérante, on parle d'identificateur de conteneur BOID (resp. identificateur de transfert de conteneur BID).

Il faut admettre que divers objets sont d'abord mis dans des conteneurs, d'une commune par exemple, que ces conteneurs sont ensuite transmis en bloc au canton puis, de là, intégrés dans des conteneurs qui comprennent tout le canton, par thème. Le cas échéant, ces conteneurs sont ensuite à nouveau transmis par exemple à la Confédération. Pour que l'on connaisse le conteneur original à tout moment, on donne son BOID à chaque objet répliqué. Un récepteur peut donc bâtir une gestion des conteneurs en répertoriant, pour ses propres conteneurs, ceux dans lesquels il conserve des objets répliqués ainsi que les conteneurs d'origine de ces objets répliqués (INTERLIS 2 met également les moyens nécessaires à disposition pour la description de tels conteneurs par le langage lui-même comme pour l'échange d'objets normaux). Si le récepteur fait appel à la propriété d'INTERLIS 2, par laquelle c'est non seulement l'OID de l'objet de référence mais aussi le BOID de son conteneur d'origine qui est transféré dans le cas de relations dépassant les limites d'un thème, alors il dispose d'un moyen efficace pour déterminer lequel parmi ses conteneurs contient l'objet de référence.

1.5 Concept de vues

INTERLIS 2 permet de modéliser des vues en plus des vrais objets. En principe, les vues sont des classes virtuelles dont les instances ne sont pas des données d'un élément concret du monde réel mais sont déduites par calcul à partir d'autres objets.

Une définition de vue comporte les éléments suivants :

- *Des ensembles de base* – De quelles classes ou vues les objets utilisés pour le calcul des objets-vues proviennent-ils ? Dans le cas de classes, ce ne sont pas seulement les vraies instances qui interviennent mais également toutes les instances des extensions dans le respect du polymorphisme. INTERLIS ne définit pas les conteneurs qu'un système doit prendre en compte pour le calcul d'une vue.
- *Une règle de connexion* – Comment relier les ensembles de base entre eux ? INTERLIS 2 connaît les jonctions *(Join)*, les unions *(Union),* les regroupements ou agrégations *(Aggregation)* et les inspections de sous-structures. Du point de vue de la théorie des ensembles, les jonctions constituent l'intersection des ensembles de base et les unions leur réunion. Les agrégations permettent de regrouper des objets d'ensembles de base au sein d'un nouvel objet-vue, sous réserve du respect de critères définissables. Les inspections de sous-structures permettent quant à elles de considérer les éléments structurés d'une sous structure comme un ensemble de tels éléments. Les jonctions et les agrégations peuvent également être interprétées comme des associations virtuelles.
- *Une sélection* – Quels objets calculés doivent effectivement appartenir à la vue ? INTERLIS permet de fixer des conditions complexes dans ce cadre.

1.6 Concept graphique

Les représentations graphiques se basent sur des classes ou des vues et déclarent les signatures graphiques (par exemple un symbole ponctuel, une ligne, un remplissage de surface ou une étiquette) à affecter aux objets de la vue dans des *définitions graphiques* (cf. Figure 5 et annexe L, glossaire), afin qu'un logiciel de transformation graphique puisse générer des objets graphiques représentables. Les signatures graphiques sont définies dans le cadre d'un modèle de signatures propre, décrivant également les propriétés de ces signatures graphiques.

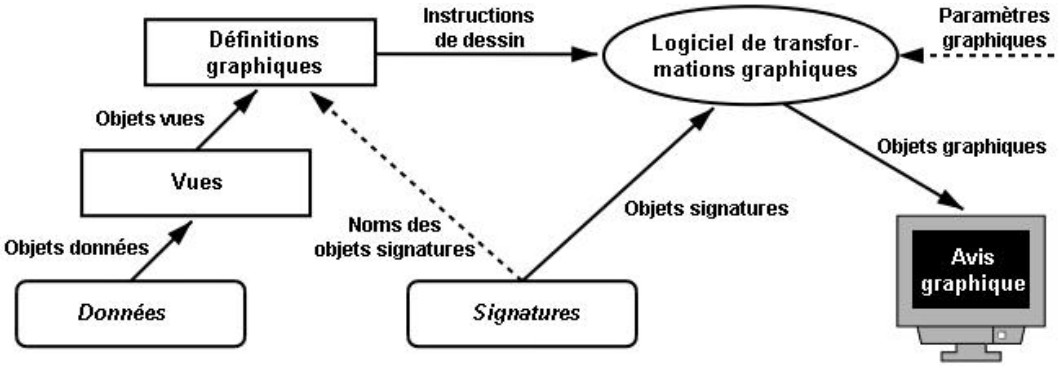


Figure 5 : Définitions graphiques construites sur des données et des vues ainsi que sur des signatures pour créer un graphique (représentation abstraite).

Le renvoi vers les signatures graphiques demandées s'effectue par le nom (cf. Noms objets signatures de la Figure 5). Les signatures graphiques elles-mêmes (également appelées objets signatures) sont stockées dans des conteneurs correspondants sous forme de méta-objets (données). Un conteneur

renfermant de tels objets signatures est fréquemment désigné par l'expression de bibliothèque de signatures.

Le modèle des signatures fixe, pour chacun des genres de signatures de la définition des classes de signatures correspondante, les paramètres (tels que la position et l'orientation de la signature) indispensables à sa représentation. Ainsi, l'interface vers le sous-système graphique (le logiciel de transformation graphique) des systèmes concernés n'est-elle pas déterminée par INTERLIS mais par les modèles de signatures. Il est par ailleurs possible de définir des paramètres d'exécution globaux (par exemple l'échelle de la représentation) qui sont connus du sous-système graphique au moment de l'exécution et peuvent influer sur le choix des signatures à utiliser pour la représentation. De telles définitions étant fortement liées aux possibilités effectives des systèmes, les paramètres concernés et les propriétés des signatures doivent être fixés dans un cadre contractuel établi en commun accord avec les producteurs de systèmes (cf. paragraphe 1.7 Contrats).

Une représentation graphique ne doit pas proposer de solution close à la transformation en signatures, laquelle doit plutôt être héritée d'une autre description de représentation. Au sein de cette dernière, des paramètres non fixés dans les définitions de base peuvent être complétés ou des instructions de représentation déjà effectuée remplacés.

1.7 Contrats

Pour qu'INTERLIS 2 puisse répondre aux exigences les plus diverses, il comprend aussi des éléments dont la mise en œuvre n'est pas réglée avec la définition, par exemple des fonctions, des formes de ligne, des signatures (on trouvera des détails à ce sujet au chapitre du langage de description). Si rien d'autre n'a été entrepris, l'objectif selon lequel une description INTERLIS peut être automatiquement transposée dans un service ne serait plus satisfait. Dans un souci de simplification, INTERLIS renonce cependant à proposer des possibilités de définition plus étendues pour de tels cas (comme un langage de programmation proprement dit pour définir des fonctions).

Pour que la mise en œuvre automatique soit possible au moins dans un cadre restreint (par exemple dans un pays ou un certain domaine d'application), de telles constructions ne doivent être autorisées que dans le cadre de modèles couverts par des contrats. Ces derniers sont des accords ou des conventions passés entre ceux qui définissent les modèles et ceux qui proposent les outils basés sur des descriptions d'INTERLIS. Normalement, seuls les modèles de base d'une branche d'activités ou d'un pays sont liés à des contrats. Pour la définition de tels modèles, des spécialistes de ces derniers et des fournisseurs d'outils coopèrent et s'entendent par ces contrats. Celui qui met l'outil à disposition est à même de réaliser les éléments demandés dans ces modèles (exemple : des fonctions) de façon indépendante de l'application concrète. Les modèles concrets peuvent ensuite être à nouveau automatiquement transposés (donc sans l'assistance du fournisseur en question).

Il importe que de telles constructions additionnelles soient formulées indépendamment des systèmes après une discussion impliquant un maximum de personnes et qu'ils soient aussi publics que le modèle lui-même. Dans le cas contraire, un des objectifs majeurs d'INTERLIS, à savoir l'ouverture et l'interopérabilité, ne peut plus être garanti.

1.8 Services, capacités des outils et conformité

INTERLIS 2 permet la description conceptuelle de données et définit un transfert de données neutre. INTERLIS 2 renonce à dessein à prescrire une implémentation et reste donc indépendant des systèmes. Dans la pratique se posera donc souvent la question de savoir si un certain objet ou un certain service est conforme à INTERLIS 2 ou non.

INTERLIS 2 ne part pas du principe que seul un oui (autrement dit satisfaction complète) ou un non (non-satisfaction) sont possibles. En effet, un service peut être conforme à INTERLIS 2 par certains aspects tandis qu'il ne remplit pas d'autres exigences.

Dans le cas le plus simple, un système donné remplit les spécifications INTERLIS pour un cas précis (pour un certain ensemble de modèles, uniquement pour lire ou uniquement pour écrire ou les deux, etc.).

Idéalement, un ensemble de modèles INTERLIS peut être transmis à un outil INTERLIS, ce qui fait que ce dernier adapte automatiquement ses services et ses capacités à la situation définie par les modèles. Dans bon nombre de cas, cette adaptation sera toutefois liée à d'autres efforts manuels (configuration du système ou même programmation). Le fait de pouvoir mettre correctement en mémoire des descriptions de modèles INTERLIS fait à coup sûr partie de la fonctionnalité de base (service de base) de tels outils. Et il entre surtout dans ce cadre le fait que les capacités des systèmes soient proposées correctement, conformément aux structures d'INTERLIS, en particulier à celles relatives à l'héritage.

Du point de vue d'INTERLIS, les capacités de tels outils peuvent être subdivisées de la manière suivante (capacités des outils, fonctionnalités ou services) :

- Mise en mémoire de données (y compris vues mémorisées sans générer d'objets-vues)
- Mise en mémoire de données (y compris vues mémorisées en engendrant des objets-vues)
- Contrôles de cohérence
- Ecriture de vues (views);
- Production de graphiques (lecture de vues (views) comprise)
- Traitement et écriture de données
- Production d'identificateurs d'objets (OID)
- Lecture de mise à jour (mise à jour incrémentielle des données)
- Ecriture de mise à jour (mise à jour incrémentielle des données)

De plus, il est tout à fait possible qu'un outil particulier ou un service déterminé ait des capacités particulières (exemple : la mise à jour successive) pour certains modèles et thèmes (données ou vues) mais ne les ait pas pour d'autres modèles ou thèmes.

Il se pose en outre la question de savoir quels contrats sont épaulés par un outil.

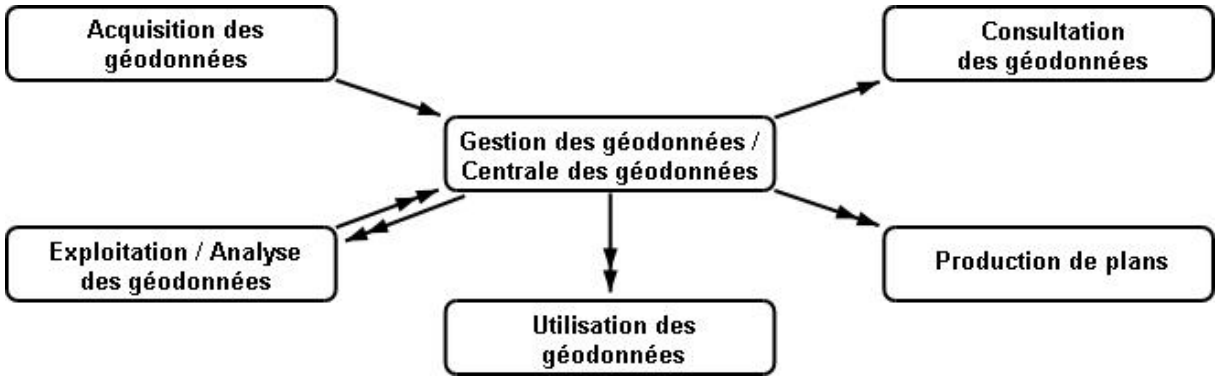


Figure 6 : Les divers domaines d'activité d'INTERLIS (une flèche double indique une livraison incrémentielle).

Si l'on considère un exemple de l'interaction entre divers acteurs concernés, il apparaît que des capacités d'outil ou des services différents d'INTERLIS 2 sont demandés (cf. Figure 6) suivant le domaine d'application et le rôle d'un acteur. Un acteur concerné peut assumer plusieurs rôles dans une application (c.-à-d. modèle INTERLIS) comprenant plusieurs thèmes (TOPICS) :

- *Acquisition des géodonnées* – Traitement et écriture de données ; génération d'OID le cas échéant.
- *Exploitation / analyse des géodonnées* (mise à jour, complément) : lecture de données ; traitement et écriture de données ; génération d'OID ; production d'incréments ; lecture d'incréments ; contrôles (locaux) de cohérence.
- *Gestion des géodonnées / centrale des géodonnées* – Lecture de données ; traitement et écriture de données ; génération d'OID ; lecture d'incréments ; contrôles (globaux) de cohérence.
- *Utilisation des géodonnées* – Lecture de données ; lecture d'incréments.
- *Consultation de géodonnées* – Lecture de données ; génération d'objets-vues et de représentations graphiques.
- *Production de plans* — Lecture de données ; lecture d'incréments ; lecture de vues et génération de représentations graphiques.

1.9 Un petit exemple Roads en guise d'introduction

L'annexe C donne un petit exemple qui présente les principaux éléments INTERLIS dans le cadre d'une application simple.

Figure 7 : Le petit exemple Roads.

1.10 Structure de ce document

Ce document de référence est structuré de la manière suivante : le chapitre 2 définit formellement notre langage conceptuel de modélisation. Le chapitre 3 spécifie le transfert séquentiel des géodonnées. Ce chapitre est divisé en une partie générale relatant les services de transfert séquentiel actuels et futurs d'INTERLIS 2, et une partie plus spécifique concernant le service de transfert séquentiel de géodonnées au moyen de XML.

Les annexes sont organisées en deux annexes *normatives* A et B, une annexe *informative* C, un index D *et des propositions d'extension des standards* E à K ainsi qu'un glossaire (annexe L).

Les annexes normatives font partie intégrante de cette spécification. Les propositions d'extensions (E à K) sont vivement recommandées pour l'implémentation, mais n'ont qu'un caractère informatif. Les applications respectives peuvent être résolues par implémentation, tant que ces implémentations demeurent conformes à INTERLIS 2. Dans un tel cas, il incombe aux parties impliquées dans le transfert de s'entendre sur cette spécification.

2 Langage de description

2.1 Syntaxe utilisée

Afin de déterminer un schéma conceptuel de données (modèle de données) et les paramètres de transfert réglant l'échange des données, un langage formel sera défini dans les chapitres suivants. Ce langage est lui-même défini de manière formelle. Les règles de syntaxe décrivent la séquence admissible des symboles.

Cette description respecte les règles usuelles en matière de langages modernes de programmation. C'est pourquoi il ne sera donné ci-dessous qu'un résumé, nécessaire à la compréhension. On consultera la bibliographie pour obtenir des détails. Une brève introduction figure par exemple dans "Programmer en Modula-2" de Niklaus Wirth.

Dans l'esprit de la "Notation Backus-Naur" (EBNF), une formule s'exprime comme suit :

```
Nom de la formule = Expression de la formule.
```

L'expression de la formule est une combinaison des éléments suivants :

- Des mots fixes (signes spéciaux inclus) du langage, écrits entre des apostrophes, par exemple 'BEGIN'.
- Des références à d'autres formules par l'indication du nom de la formule.

Entrent en ligne de compte comme combinaison :

Juxtaposition

```
a b c          d'abord a, puis b, puis c.
```

Groupement

```
( a )          des parenthèses rondes groupent des expressions de formule.
```

Choix

```
a | b | c      a, b, ou c.
```

Option

```
[ a ]          a ou rien (vide).
```

Répétition facultative

```
{ a }          Suites quelconques de a ou rien (vide).
```

Répétition obligatoire (comme complément à EBNF)

```
(* a *)        Suite quelconque de a, mais un au moins.
```

Exemples d'expressions de formule :

```
(a|b)(c|d)     ac, ad, bc ou bd
a[b]c          abc ou ac
a{ba}          a, aba, ababa, abababa, ...
{a|b}c         c, ac, bc, aac, abc, bbc, bac, ...
a(*b*)         ab, abb, abbb, abbbb, ...
(*ab|[c]d*)    ab, d, cd, abd, dab, cdab, ababddd, cdababcddcd, ...
```

On aimerait souvent utiliser une même formule syntaxique dans des circonstances différentes, à des fins diverses. Pour ce faire, il faudrait écrire une formule additionnelle :

```
Example = 'CLASS' Classname '=' Classdef.
Classname = Name.
```

Pour éviter ce détour, on utilise la notation abrégée suivante :

```
Example = 'CLASS' Class-Name '=' Classdef.
```

La formule Nom_de_la_classe n'est pas définie. Sur le plan syntaxique, la règle "Nom" est directement utilisée (cf. paragraphe 2.2.2 Noms). Quant au fond, Nom est un nom de classe. Le complément "de_la_classe" devient pratiquement un commentaire.

2.2 Symboles de base du langage

Le langage de description distingue les classes de symboles suivantes : noms, chaînes de caractères, nombres, explications, signes particuliers et mots réservés, commentaires.

2.2.1 Caractères, espaces et fin de ligne

Le langage vrai n'utilise que les caractères ASCII américains (32 à 126) imprimables. Il y a lieu de déterminer par l'application concrète du compilateur les caractères ayant valeur d'espacements en sus du caractère espace. Cette application détermine aussi les caractères ou combinaisons de caractères valant comme fin de ligne. La mise en mémoire des signes (ensemble de caractères) est l'affaire de l'application du compilateur. Elle peut différer en particulier en fonction de la plate-forme.

Des caractères complémentaires peuvent être utilisés dans des commentaires (exemple : voyelle infléchie et accents).

2.2.2 Noms

Un nom est défini comme une suite de 255 lettres, chiffres et traits de soulignement au maximum. Le premier caractère doit toutefois être une lettre. On établit une distinction entre majuscules et minuscules. Les noms coïncidant avec des mots réservés du langage (cf. paragraphe 2.2.7 Signes particuliers et mots réserées) ne sont pas admis.

Syntaxe :

```
Name = Letter { Letter | Digit | '_' }.
Letter = ( 'A' | .. | 'Z' | 'a' | .. | 'z' ).
Digit = ( '0' | '1' | .. | '9' ).
HexDigit = ( Digit | 'A' | .. | 'F' | 'a' | .. | 'f' ).
```

Des informations détaillées sur l'unicité et le domaine de validité des noms sont décrites au paragraphe 2.5.4 Espaces nominaux.

2.2.3 Chaînes de caractères

Les chaînes de signes (Strings) se manifestent en relation avec des constantes. Elles commencent et se terminent avec des guillemets et ne peuvent pas s'étendre sur plusieurs lignes. \" vaut pour un guillemet et \\ pour un "backslash" (barre oblique inversée) au sein du "string".

Une séquence de \u, directement suivie d'exactement quatre chiffres hexadécimaux annonce un quelconque signe Unicode. Des signes à partir de U+10000 doivent être désignés, comme avec la codification UTF-16, au moyen de deux codes Surrogat (cf. www.unicode.org).

Syntaxe :

```
String = '"' { <any character except '\' or '"'>
```

```
              | '\"'
              | '\\'
              | '\u' HexDigit HexDigit HexDigit HexDigit
              } '"'.
```

2.2.4 Nombres

Les nombres apparaissent sous des formes diverses : nombres positifs entiers y compris 0 (NoPos), nombres entiers (nombre), chiffres décimaux (Dec) et nombres structurés. Pour les chiffres décimaux, le facteur de multiplication peut être indiqué comme puissance de dix (exemple : 1E2 équivaut à 100, 1E-1 à 0.1). Des chiffres structurés n'ont de sens qu'en relation avec des unités et des domaines correspondants (l'heure, par exemple).

Syntaxe :

```
PosNumber = (* Digit *).
Number = [ '+' | '-' ] PosNumber.
Dec = ( Number [ '.' PosNumber ] | Float ).
Float = [ '+' | '-' ] '0.' (('1'|'2'|...|'9') [PosNumber] | (* '0' *))
        Scaling.
Scaling = ( 'e' | 'E' ) Number.
StructDec = Number (* ':' PosNumber *) [ '.' PosNumber ].
```

Exemples :

```
PosNumber:   5134523             1          23
Number:          123          -435        +5769
Dec:         123.456     0.123456e4    -0.123e-2
Float:         0.1e7   -0.123456E+4   0.987e-100
StructDec:  23:59.10       02:11:07
```

2.2.5 Ensembles de propriétés

Des propriétés doivent être assignées à un objet de description à des fins diverses. Ceci peut être effectué avec une syntaxe générale :

Syntaxe :

```
Properties = [ '('Property { ',' Property } ')' ].
```

Pour définir qu'à un endroit précis d'une règle de syntaxe de telles propriétés doivent être définies, la construction suivante est introduite dans la syntaxe :

```
'Properties' '<' Property-Keyword { ',' Property-Keyword } '>'
```

On écrit donc "Properties" et on définit les mots-clés autorisés entre des crochets. Si l'on prend pour exemple la règle ClassDef (cf. paragraphe 2.5.3 Structures et classes), les mots-clés "ABSTRACT", "EXTENDED" et "FINAL" sont acceptés pour décrire des propriétés avec "Properties<ABSTRACT,EXTENDED,FINAL>". Dans une définition INTERLIS 2, les définitions suivantes seraient notamment possibles :

```
CLASS A (ABSTRACT) = .....
CLASS A (EXTENDED, FINAL) = .....
```

2.2.6 Explications

Une explication est exigée là où un fait doit être décrit avec plus de précision. Du point de vue du mécanisme standard, cette explication n'est pas interprétée et traitée comme un commentaire. Mais il est toujours possible de formaliser des explications plus détaillées pour permettre un traitement automatique supplémentaire. Une explication est placée entre deux doubles traits obliques et ne doit par conséquent pas contenir de tels traits dédoublés.

Syntaxe :

```
Explanation = '//' any character except //  '//'.
```

2.2.7 Signes particuliers et mots réservés

Les signes particuliers et les mots réservés sont toujours écrits entre des apostrophes selon les règles syntaxiques du langage (mais pas dans le cadre d'une description de données), comme ',' ou 'MODEL'. Les mots réservés s'écrivent en majuscules. Ne pas utiliser exclusivement des majuscules pour les noms permet d'éviter facilement tout risque de conflit.

Les mots suivants sont réservés (quelques-uns d'entre eux ont été utilisés en INTERLIS 1 et demeurent réservés pour des raisons de compatibilité ; ils sont indiqués en italique avec un style normal) :

ABSTRACT	**ACCORDING**	**AGGREGATES**	**AGGREGATION**
ALL	**AND**	**ANY**	**ANYCLASS**
ANYSTRUCTURE	**ARCS**	**AREA**	**AS**
ASSOCIATION	**ATTRIBUTE**	**ATTRIBUTES**	**BAG**
BASE	**BASED**	**BASKET**	*BLANK*
BOOLEAN	**BY**	**CIRCULAR**	**CLASS**
CLOCKWISE	*CODE*	**CONSTRAINT**	**CONSTRAINTS**
CONTINUE	**CONTINUOUS**	*CONTOUR*	**CONTRACT**
COORD	*COORD2*	*COORD3*	**COUNTERCLOCKWISE**
DATA	*DATE*	*DEFAULT*	**DEFINED**
DEGREES	**DEPENDS**	*DERIVATIVES*	**DERIVED**
DIM1	*DIM2*	**DIRECTED**	**DOMAIN**
END	**EQUAL**	**EXISTENCE**	**EXTENDED**
EXTENDS	**EXTERNAL**	**FINAL**	**FIRST**
FIX	*FONT*	**FORM**	*FORMAT*
FREE	**FROM**	**FUNCTION**	*GRADS*
GRAPHIC	**HALIGNMENT**	*I16*	*I32*
IDENT	**IMPORTS**	**IN**	**INSPECTION**
INTERLIS	**ISSUED**	**JOIN**	**LAST**
LINE	*LINEATTR*	*LINESIZE*	**LIST**
LNBASE	**LOCAL**	**MANDATORY**	**METAOBJECT**
MODEL	**NAME**	*NO*	**NOT**
NULL	**NUMERIC**	**OBJECT**	**OF**
OID	**ON**	*OPTIONAL*	**OR**
ORDERED	**OTHERS**	**OVERLAPS**	**PARAMETER**
PARENT	*PERIPHERY*	**PI**	**POLYLINE**
PROJECTION	*RADIANS*	**REFERENCE**	**REFSYSTEM**
REQUIRED	**RESTRICTED**	**ROTATION**	**SIGN**
STRAIGHTS	**STRUCTURE**	**SURFACE**	**SYMBOLOGY**
TABLE	**TEXT**	**THATAREA**	**THIS**
THISAREA	*TID*	*TIDSIZE*	**TO**
TOPIC	*TRANSFER*	**TRANSIENT**	**TRANSLATION**
TYPE	**UNDEFINED**	**UNION**	**UNIQUE**
UNIT	**UNQUALIFIED**	**URI**	**VALIGNMENT**
VERTEX	*VERTEXINFO*	**VIEW**	**WHEN**
WHERE	**WITH**	**WITHOUT**	

Table 1 : Mots réservés pour INTERLIS 2.

2.2.8 Commentaires

On distingue deux formes de commentaires :

2.2.8.1 Commentaire rédigé sur une ligne

Deux points d'exclamation consécutifs ouvrent un commentaire. Ce dernier se termine à la fin de la ligne.

Syntaxe :

```
!! Line comment; goes until end of line
```

2.2.8.2 Bloc de commentaire

Le bloc de commentaire est annoncé par une barre oblique suivie d'une étoile ; il se termine par les mêmes signes, dans l'ordre inverse. Il peut s'étendre sur plusieurs lignes et contenir des commentaires du type précédemment décrit ; des blocs de commentaire emboîtés ne sont en revanche pas autorisés.

Syntaxe :

```
/* Block comment,
    additional line comment */
```

2.3 Règle principale

Chaque unité descriptive débute par l'indication de la version du langage, jetant ainsi la base de compléments ultérieurs du langage. Dans ce document, c'est la version **2.2** d'INTERLIS qui est décrite.

Les descriptions du modèle suivent alors.

Syntaxe :

```
INTERLIS2Def = 'INTERLIS' Version-Dec ';'
               { ModelDef }.
```

2.4 Héritage

Divers éléments d'INTERLIS peuvent être étendus au sens de l'orientation objet : une première définition crée la base qui pourra être ensuite spécialisée en plusieurs étapes.

Thèmes, classes, vues, descriptions graphiques, unités et domaines peuvent étendre les structures de base pertinentes (mot-clé EXTENDS ou EXTENDED) et héritent de ce fait de toutes leurs propriétés. Une structure préalablement définie peut dans certains cas être étendue par la mention EXTENDED, le nom étant conservé.

Le terme FINAL empêche l'extension d'une définition. Diverses structures peuvent être définies dans une forme encore incomplète (mot-clé ABSTRACT) ; elles seront ultérieurement complétées en une extension pour donner une définition concrète.

Pour indiquer les mots-clés autorisés dans un contexte déterminé, on utilise systématiquement le mode général d'écriture Property (cf. paragraphe 2.2.5 Ensembles de propriétés).

2.5 Modèles, thèmes, classes

2.5.1 Modèles

Par modèle, on entend une définition close et complète. En fonction de son genre, il peut comprendre diverses structures.

Un modèle type (TYPE MODEL) ne peut déclarer que des unités de mesure, des domaines, des fonctions et des formes de ligne.

Outre les définitions d'un modèle type, un modèle de système de référence (REFSYSTEM MODEL) doit déclarer uniquement des classes et des thèmes en relation avec des extensions des classes prédéfinies AXIS ou REFSYSTEM (cf. paragraphe 2.10.3 Systèmes de référence). Le respect de cette règle ne peut pas être forcé par le langage. C'est à l'utilisateur de s'y conformer.

Un modèle de signature (SYMBOLOGY MODEL) ne doit déclarer - outre les définitions d'un modèle type - que des thèmes et classes en relation avec des extensions de la classe prédéfinie SIGN, ainsi que des paramètres graphiques (cf. paragraphe 2.16 Représentations graphiques resp. 2.11 Paramètres

d'exécution). Le respect de cette règle est également l'affaire de l'utilisateur). Des modèles de signature ne sont possibles qu'en relation avec des contrats. Ils doivent en effet être harmonisés quant à la manière dont les systèmes traitent les symboles.

Aucune restriction de qualité du modèle ne sera définie, un modèle peut contenir toutes les constructions possibles.

Après le nom du modèle, la langue peut être indiquée au choix. L'indication devrait si possible être opérée à l'aide de désignations standardisées à deux lettres de la norme ISO 639 (cf. www.iso.ch) ; par exemple "de" signifie allemand, "fr" français, "it" italien, "rm" romanche et "en" anglais. Un code de pays selon la norme ISO 3166 séparé par un trait de soulignement peut être placé ensuite pour désigner la variante de langue d'un pays déterminé : "de_CH" indique l'allemand (littéraire) utilisé en Suisse. Cette indication a une valeur documentaire. Elle est en rapport avec la possibilité de déclarer un modèle comme étant la traduction d'un autre (TRANSLATION OF). Les deux modèles doivent alors concorder exactement quant à leur structure. Ils ne peuvent donc différer qu'au niveau du nom utilisé. La déclaration comme traduction n'est toutefois pas liée à l'indication linguistique. Pour appuyer des usages linguistiques par exemple locaux ou spécifiques d'une branche, on autorise notamment les traductions dans la même langue que la description originelle.

Syntaxe :

```
ModelDef = [ 'TYPE' | 'REFSYSTEM' | 'SYMBOLOGY' ]
           'MODEL' Model-Name [ '('Language-Name ')' ]
           [ 'TRANSLATION' 'OF' Model-Name ] '='
               { 'CONTRACT' 'ISSUED' 'BY' Issuer-Name [ Explanation ] ';' }
               { 'IMPORTS' [ 'UNQUALIFIED' ] Model-Name
               { ',' [ 'UNQUALIFIED' ] Model-Name } ';' }
               { MetaDataUseDef
               | UnitDef
               | FunctionDef
               | LineFormTypeDef
               | DomainDef
               | RunTimeParameterDef
               | ClassDef
               | TopicDef }
           'END' Model-Name '.'.
```

Si un modèle est lié à un contrat, l'éditeur de ce contrat est mentionné. Des précisions (telles que l'adresse postale ou l'adresse de messagerie électronique de l'éditeur) peuvent être fournies dans les commentaires.

Si un élément d'INTERLIS se rapporte à une définition réalisée dans un autre modèle, ce dernier est à importer (mot-clé IMPORTS). Ainsi, des thèmes peuvent par exemple être étendus et la référence à des classes effectuée. IMPORTS n'ouvre toutefois qu'une possibilité d'utilisation. Lors de l'emploi des définitions importées, celles-ci sont à référencer par leur nom qualifié (modèle, thème), à moins que l'on utilise le mot-clé UNQUALIFIED.

Un modèle de base prédéfini "INTERLIS" (cf. annexe A Le modèle de données interne INTERLIS) est également lié à la langue. Il n'a pas à être importé. Ses éléments ne sont cependant à disposition avec un nom non qualifié que si le modèle a été introduit par IMPORTS UNQUALIFIED INTERLIS.

2.5.2 Thèmes

Un thème (mot-clé TOPIC) contient toutes les définitions en vue de décrire un pan objectif précis du monde réel. Un thème peut aussi définir des types comme des unités de mesure, des domaines de valeurs ou des structures ou les utiliser à partir du modèle environnant ou d'un modèle importé.

Les propriétés d'héritage peuvent être définies entre parenthèses. L'extension d'un thème se référant toujours à un thème portant un nom différent, EXTENDED est sans objet et n'est de ce fait pas admis.

Syntaxe :

```
TopicDef = [ 'VIEW' ] 'TOPIC' Topic-Name
               Properties<ABSTRACT,FINAL>
               [ 'EXTENDS' TopicRef] '='
                   [ 'OID' 'AS' OID-DomainRef ]
                   { 'DEPENDS' 'ON' TopicRef { ',' TopicRef } ';' }
                   Definitions
           'END' Topic-Name ';'.

Definitions =  { MetaDataUseDef
               | UnitDef
               | DomainDef
               | ClassDef
               | AssociationDef
               | ConstraintsDef
               | ViewDef
               | GraphicDef }.

TopicRef = [ Model-Name '.' ] Topic-Name.
```

Un nombre quelconque de conteneurs (banques de données, etc.) peut exister pour un thème donné contenant des classes concrètes. Tous présentent la structure correspondante du thème mais comportent des objets différents.

Seules les instances de classes (et leurs sous-structures) apparaissent dans un conteneur de données. Si un thème contient des représentations graphiques, trois types de conteneurs peuvent être formés :

- Des conteneurs de données.
- Des conteneurs comprenant les données de base pour la représentation graphique. De tels conteneurs comprennent les instances de classes ou de vues constituant la base des représentations graphiques.
- Des conteneurs graphiques. De tels conteneurs comprennent les objets graphiques convertis (conformément à la définition des paramètres des signatures utilisées).

Dans le cas de thèmes dans lesquels seules des vues sont définies (n'intégrant donc ni classes ni définitions graphiques) et sont par ailleurs explicitement identifiées comme thèmes-vues (mot-clé VIEW), des conteneurs comportant des instances des vues du thème sont également possibles.

Les identifications des conteneurs et les objets identifiables qu'ils comprennent doivent respecter le domaine de valeurs assigné à cette fin au thème (OID AS). Si un domaine de valeur d'OID a été assigné à un thème, cette affectation ne peut plus être modifiée dans des extensions. En l'absence de toute assignation, le domaine de valeurs d'OID STANDARDOID (cf. paragraphe 2.8.8 Domaines de valeurs d'identifications d'objets et annexes A et E) s'applique implicitement. Des définitions explicites ne sont admises que dans le cadre de contrats.

Du point de vue des données et en l'absence de toute indication supplémentaire, un thème est indépendant d'autres thèmes. Des dépendances au niveau des données résultent de relations ou d'attributs de référence renvoyant à un autre conteneur, de conditions de cohérence particulières ou de la constitution de vues ou de définitions graphiques reposant sur des classes ou sur d'autres vues. De telles dépendances sont à déclarer explicitement dans l'en-tête du thème (mot-clé DEPENDS ON) afin qu'elles puissent être clairement reconnues. Les définitions détaillées (par exemple les définitions de relations) ne doivent alors pas violer cette déclaration de dépendance. Des dépendances cycliques ne sont pas

autorisées. Un thème étendu possède les mêmes dépendances que son thème de base, en l'absence de toute déclaration supplémentaire.

2.5.3 Structures et classes

Une définition de classe (mot-clé CLASS) déclare les propriétés de tous les objets qui lui appartiennent. Les définitions de classes peuvent être étendues. Une extension hérite, dans ce contexte, de tous les attributs de sa classe de base. Son domaine de valeur peut être limité. Par ailleurs, d'autres attributs complémentaires peuvent être définis.

Des conditions de cohérence peuvent être indiquées comme faisant partie d'une définition de classe ; ce sont des règles que les objets doivent satisfaire à titre complémentaire. Elles ne peuvent pas être déclarées sans effet mais s'appliquent toujours en plus de celles déclarées localement.

Les objets d'une classe sont toujours indépendants et identifiables individuellement. D'un point de vue formel, les structures (mot-clé STRUCTURE) sont définies de la même manière que les classes, mais les éléments structurés qu'elles contiennent ne sont pas indépendants et ne peuvent pas être identifiés isolément. On rencontre ces éléments au sein de sous-structures d'objets (cf. paragraphe 2.6 Attributs), mais leur existence ne peut par ailleurs être que temporaire, sous forme de résultats de fonctions. Partout où des structures sont utilisées, le renvoi à la définition d'une classe est également possible. C'est ce que l'on appelle l'utilisation structurelle de la classe. Dans ce type d'utilisation, les classes sont cependant à interpréter comme des structures de sorte que leurs objets ne sont pas indépendants.

Les classes particulières telles que celles des systèmes de référence, des axes d'un système de coordonnées et des signatures graphiques (donc des extensions de la classe prédéfinie METAOBJECT) sont décrites au paragraphe 2.10 Traitement des méta-objets.

Les propriétés d'héritage peuvent être définies entre parenthèses (règle Properties). Toutes les possibilités sont admises. Si une classe ou une structure contient des attributs abstraits, elle est à déclarer comme ABSTRACT. Des attributs abstraits doivent encore être concrétisés dans des extensions concrètes de la classe. Cependant, il est également permis de déclarer abstraites des classes dont les attributs sont totalement définis. Des instances objets ne peuvent exister que pour des classes concrètes ayant été définies au sein d'un thème. Des classes définies en dehors de thèmes (donc directement dans le modèle) ne peuvent contenir aucun attribut de référence. Il n'est par ailleurs pas permis de définir des associations avec de telles classes.

Si l'héritage porte sur une seule classe et non sur un thème complet, aucune relation avec cette classe ne peut être définie (cf. paragraphe 2.7 Relations vraies).

Si un thème en étend un autre, toutes les classes du thème hérité sont reprises. Elles deviennent alors des classes du thème actuel et portent le même nom que dans le thème hérité. Une telle classe peut également être étendue en conservant son nom (EXTENDED). Si un thème T2 étend par exemple le thème T1, contenant la classe C, T2 ne comportera qu'une seule classe C (EXTENDED), à savoir C. De nouvelles classes devant se distinguer de celles héritées par leur désignation, peuvent aussi étendre des classes héritées. Avec C2 EXTENDS C, T2 comporte alors deux classes (C et C2). INTERLIS n'acceptant que l'héritage simple dans un souci de simplicité et de clarté, EXTENDED n'est toutefois admis que si la classe de base n'a été étendue avec EXTENDS ni dans le thème de base ni dans le thème actuel. EXTENDED et EXTENDS s'excluent mutuellement dans la même définition de classe.

Syntaxe :

```
ClassDef = ( 'CLASS' | 'STRUCTURE' ) Class-Name
               Properties<ABSTRACT,EXTENDED,FINAL>
                   [ 'EXTENDS' StructureRef ] '='
               [ 'ATTRIBUTE' ] { AttributeDef }
               { ConstraintDef }
```

```
                [ 'PARAMETER' { ParameterDef } ]
            'END' Class-Name ';'.

ClassRef = [ Model-Name '.' [ Topic-Name'.' ] ] Class-Name.

StructureRef = [ Model-Name '.' [ Topic-Name '.' ] ]
               ( Structure-Class-Name | Class-Name ).
```

Le paragraphe suivant (2.5.4 Espaces nominaux) explique quels noms doivent être qualifiés (par des Model-Name et/ou par des Model-Name. Topic-Name). Des classes et des structures ne se basant sur aucune classe ou structure déjà définie ne requièrent aucune partie EXTENDS.

2.5.4 Espaces nominaux

Un espace nominal désigne un ensemble de noms (univoques). Chacun des éléments de modélisation (modèle de données, thème, élément de classe) tout comme les conteneurs de métadonnées mettent un espace nominal propre à disposition pour chacune des catégories de noms (noms de types, noms de composantes, noms de méta-objets).

Il existe trois niveaux hiérarchiques pour les éléments de modélisation :

- Le modèle (MODEL est le seul élément de modélisation au niveau le plus élevé)
- Le thème (TOPIC est le seul élément de modélisation à ce niveau)
- Les éléments de classe, à savoir les classes (CLASS), la structure (STRUCTURE), l'association (ASSOCIATION), la vue (VIEW), la définition graphique (GRAPHIC)

Les noms de conteneurs de métadonnées ouvrent l'accès aux méta-objets (cf. paragraphe 2.10 Traitement des méta-objets).

Il existe trois catégories de noms, contenant les noms suivants :

- Les noms de types sont les symboles (noms) des unités et les noms de fonctions, de types de formes de lignes, de domaines de valeurs, de structures, de thèmes, de classes, d'associations, de vues, de définitions graphiques et de conteneurs.
- Les noms de composantes sont ceux de paramètres d'exécution, d'attributs, de règles de dessin, de paramètres, de rôles, d'accès à des relations et de vues de base.
- Les noms de méta-objets désignent les méta-objets et n'existent qu'au sein de conteneurs de métadonnées.

Afin d'exclure toute possibilité de malentendu, les éléments de modélisation reprennent par ailleurs les noms des éléments de modélisation de niveau supérieur dans le respect de la catégorie du nom. Si un élément de modélisation en étend un autre, ses espaces nominaux sont ajoutés à tous les noms de l'élément de modélisation de base. Un nom défini localement dans l'élément de modélisation ne doit pas entrer en conflit avec un nom repris, sauf à ce qu'il s'agisse explicitement d'une extension (EXTENDED).

Si l'on souhaite référencer des éléments de description du modèle de données, leur nom doit normalement être indiqué de façon qualifiée, c.-à-d. précédé du nom du modèle et du thème. Les noms des espaces nominaux de l'élément de modélisation concerné peuvent être utilisés de manière non qualifiée.

2.6 Attributs

2.6.1 Généralités

Tout attribut est défini par son nom et son type. Les propriétés d'héritage peuvent être définies entre parenthèses (règle Properties). Si un attribut est une extension d'un attribut hérité, il faut le signaler expressément par EXTENDED. Si le domaine de valeurs d'un attribut est abstrait, l'attribut doit être déclaré comme étant ABSTRACT.

Syntaxe :

```
AttributeDef = Attribute-Name Properties<ABSTRACT,EXTENDED,FINAL>
                   ':' AttrTypeDef
                       [ ':=' Factor { ',' Factor } ] ';'.
```

Si la valeur de l'attribut est définie au moyen d'un facteur (cf. paragraphe 2.13 Expressions), son type de résultat doit être compatible avec une assignation à l'attribut défini, c.-à-d. qu'il doit présenter un domaine de valeur identique ou étendu, c.-à-d. spécialisé. Dans le cadre de vues, principalement pour des unions et des extensions de vues (cf. paragraphe 2.15 Vues), plusieurs facteurs peuvent être définis et des facteurs supplémentaires peuvent être ajoutés dans des extensions de vues additionnelles. C'est toujours le dernier en date auquel une valeur a été affectée qui s'applique (d'abord la base puis l'extension).

La définition d'un attribut peut être précisée de l'une des manières suivantes dans des extensions :

- En restreignant le domaine de valeurs.
- En lui affectant une constante dans le domaine de valeurs requis. Une telle définition est implicitement finale, en d'autres termes, elle ne peut plus être précisée.
- En définissant un facteur lorsque le type du résultat pourrait être admis comme extension de l'attribut. Une précision supplémentaire est autorisée dans ce cadre.

Syntaxe :

```
AttrTypeDef = ('MANDATORY' [ AttrType ]
              | AttrType
              | ( ( 'BAG' | 'LIST' ) [ Cardinality ]
                  'OF' RestrictedStructureRef ) ).

AttrType = ( Type
           | DomainRef
           | ReferenceAttr
           | RestrictedStructureRef ) .

ReferenceAttr = 'REFERENCE' 'TO'
                  Properties<EXTERNAL> RestrictedClassOrAssRef .

RestrictedClassOrAssRef = ( ClassOrAssociationRef | 'ANYCLASS' )
                                [ 'RESTRICTED' 'TO' ClassOrAssociationRef
                                    { ',' ClassOrAssociationRef } ].

ClassOrAssociationRef = ( ClassRef | AssociationRef ).

RestrictedStructureRef = ( StructureRef | 'ANYSTRUCTURE' )
                             [ 'RESTRICTED' 'TO' StructureRef
                                 { ',' StructureRef } ].
```

Dans le cadre d'extensions, il est possible d'indiquer seulement MANDATORY. C'est ensuite le type d'attribut déjà défini qui s'applique. Il est toutefois exigé que la valeur soit toujours définie.

2.6.2 Attributs avec domaines de valeur comme type

Des définitions directes de type (règle Type) et l'utilisation de domaines de valeurs déjà définis (règle DomainRef) entrent en ligne de compte comme types d'attributs. Les différentes possibilités sont mentionnées sous 2.8 Domaines de valeurs et constantes.

2.6.3 Attributs de référence

Un attribut de référence permet de définir un renvoi vers un autre objet. Les attributs de référence ne sont admis qu'au sein de structures, raison pour laquelle une structure contenant directement ou indirectement (via des sous-structures) des attributs de référence ne peut pas être étendue vers une

classe. Les liens entre objets indépendants sont à définir par l'intermédiaire de relations vraies (cf. paragraphe 2.7 Relations vraies).

Les classes dont les objets entrent en ligne de compte pour la référence peuvent être des classes d'objets ou de relations concrètes ou abstraites mais en aucun cas des structures (ces dernières n'étant pas des objets indépendants). Ainsi, toutes les classes concrètes correspondant aux classes primaires énumérées ou à l'une des classes restrictives énumérées (RESTRICTED TO) sont concernées (la classe elle-même ou une sous-classe de celle-ci). A chaque niveau de restriction (définition initiale ou extensions successives), toutes les classes encore admises doivent être énumérées. Chaque classe définie comme une restriction doit être une sous-classe d'une classe admise jusqu'alors. Une classe entrant ainsi en ligne de compte n'est toutefois admise que si elle appartient à un thème, coïncidant ou non avec celui de l'attribut de référence, dont le thème "référençant" est dépendant (DEPENDS ON). Si la référence renvoie à un objet d'un autre conteneur du même thème ou d'un autre thème (hypothèse DEPENDS ON), alors la propriété EXTERNAL doit être indiquée. S'agissant d'extensions, cette propriété peut être omise et par conséquent exclue, mais en aucun cas ajoutée. Il doit au moins exister une sous-classe concrète admise lorsque l'attribut de référence n'est pas déclaré comme étant abstrait.

2.6.4 Attribut structuré

Les attributs structurés (règle RestrictedStructureRef) comprennent un (ni LIST ni BAG requis) ou plusieurs (nombre admis dans le cadre de la cardinalité indiquée) éléments structurés ordonnés (LIST) ou non ordonnés (BAG). Les éléments structurés ne disposent d'aucun OID, n'existent qu'en relation avec leur objet principal et ne peuvent être trouvés que par l'intermédiaire de celui-ci. Leur organisation résulte de la structure ou de la classe indiquée. La totalité des éléments structurés, desquels tous les attributs structurés d'une classe sont repris, est appelée une sous-structure.

Si la structure ou la classe de la sous-structure est quelconque (ANYSTRUCTURE ou ANYCLASS), l'attribut structuré doit être déclaré comme étant abstrait, pour autant qu'il soit obligatoire ou que sa cardinalité minimale soit supérieure à zéro. ANYSTRUCTURE peut être spécialisé en ANYCLASS dans une extension, pour autant qu'aucune restriction (RESTRICTED TO) ne soit définie.

Des sous-structures peuvent être définies avec des structures concrètes ou abstraites. Ce sont en premier lieu toutes les classes ou structures primaires ou énumérées comme étant restrictives (RESTRICTED TO) de même que leurs classes ou structures étendues qui entrent en ligne de compte pour ce qui concerne les structures concrètes. Dans le thème courant, il s'agit, sans restriction supplémentaire, de toutes les classes ou structures étendues. Hors du thème courant, les classes ou structures requises doivent être explicitement énumérées. A chaque niveau de restriction (définition initiale ou extensions successives), toutes les classes ou structures encore admises doivent être énumérées. Chaque classe ou structure définie comme une extension doit être une extension d'une classe ou structure admise jusqu'alors. Il doit au moins exister une extension concrète de la structure lorsque la sous-structure n'est pas déclarée comme étant abstraite. Une sous-structure ordonnée (LIST) ne peut pas être étendue par une sous-structure non ordonnée (BAG). L'ensemble des règles s'appliquant aux relations vaut également pour la cardinalité (cf. paragraphe 2.7.3 Cardinalité).

2.7 Relations vraies

2.7.1 Généralités

Les relations vraies (au contraire des attributs de référence ; cf. paragraphe 2.6 Attributs) sont décrites comme des éléments indépendants. La propriété la plus importante d'une relation est constituée par l'énumération des classes d'objets (au nombre minimal de deux) assignées dans les rôles, complétée par les caractéristiques détaillées que sont l'intensité de la relation et sa cardinalité. Les noms des rôles doivent en principe être des substantifs. Ils peuvent coïncider avec les noms des classes d'objets

affectées, mais il ne s'agit pas là d'une obligation. La relation elle-même peut par ailleurs présenter des attributs locaux. Si le nom d'association n'est pas indiqué, il est implicitement formé à partir de la composition des noms de rôles (le premier, puis le deuxième, etc.).

Une relation constitue une classe un peu spéciale à divers titres. Les mêmes dispositions que celles valant pour les classes s'appliquent pour ce qui concerne l'héritage. Afin que la signification de la cardinalité soit claire et inchangeable, il est interdit d'ajouter des rôles supplémentaires dans les extensions. Si, dans des extensions de relations, les classes de référence admises sont restreintes par rapport à l'association de base, seules les relations énumérées de façon explicite restent alors permises. Si une relation est définie entre les classes A et B, lesquelles sont étendues par des sous-classes A1, A2, B1, B2, etc., des relations quelconques entre objets de A1, A2 et B1, B2 sont en principe admises. Si une extension de la relation est toutefois entreprise, avec des rôles prévoyant par exemple de restreindre la relation à A1 et B1, seules de telles relations sont alors admises entre les objets des classes A1 et B1. Il reste cependant permis de définir plusieurs extensions de relations de ce type, par exemple une telle extension entre A1 et B3. Ainsi, un objet d'A1 pourrait être mis en relation avec des objets de B1 ou de B3 mais ne pourrait pas l'être avec des objets de B2.

Syntaxe :

```
AssociationDef = 'ASSOCIATION' [ Association-Name ]
                     Properties<ABSTRACT,EXTENDED,FINAL>
                     [ 'EXTENDS' AssociationRef ]
                     [ 'DERIVED' 'FROM' RenamedViewableRef ] '='
                     { RoleDef }
                     [ 'ATTRIBUTE'] { AttributeDef }
                     { ConstraintDef }
                 'END' [ Association-Name ] ';'.

AssociationRef = [ Model-Name'.' [ Topic-Name'.' ]] Association-Name.

RoleDef = Role-Name Properties<ABSTRACT,EXTENDED,FINAL,ORDERED,EXTERNAL>
              ( '--'|'-<>'|'-<#>' ) [ Cardinality ]
              RestrictedClassOrAssRef
              [ ':=' Role-Factor ] '; '.

Cardinality = '{' ( '*' | PosNumber [ '..' ( PosNumber | '*' ) ] ) '}'.
```

Les mêmes règles que celles s'appliquant aux attributs de référence valent pour les classes d'objets et de relations définies comme étant admises dans le cadre des rôles (cf. paragraphe 2.6.3 Attributs de référence).

Une relation concrète entre objets peut être considérée comme un objet indépendant. Une identification d'objet propre lui est attribuée lorsque l'une des conditions suivantes est remplie :

- La relation possède plus de deux rôles.
- La relation possède deux rôles, la cardinalité maximale de chacun d'entre eux étant supérieur à un.

Si la cardinalité maximale d'au moins un rôle d'une relation duale est déjà égale à un dans la définition de base (et non au stade ultérieur d'une extension), aucune identification d'objet propre ne sera affectée à l'objet relationnel (mais l'OID de l'objet correspondant). L'objet relationnel sera considéré comme attribut structuré de l'objet correspondant, soit au rôle dont la cardinalité maximale est supérieure à un, soit au deuxième rôle, au cas où les deux rôles présenteraient une cardinalité maximale égale à un. Comme autre condition admise, il est considéré que la classe qui reprend l'attribut structuré, doit être défini dans le même thème que la classe d'association. Cela permet d'une part d'éviter la gestion d'identifications d'objets inutiles par les implémentations et ouvre d'autre part la possibilité d'une compatibilité avec INTERLIS 1 (cf. chapitre 3 Transfert séquentiel).

Normalement, les relations concrètes entre objets doivent être générées explicitement au moyen d'une application puis considérées comme des instances par le système de traitement. Mais une relation peut également être déduite d'une vue sans qu'elle en devienne une instance (DERIVED FROM). Une telle relation peut être une extension d'une relation abstraite. Elle ne peut pas être abstraite elle-même. Si elle est étendue, l'extension doit se baser sur la même vue ou sur une extension de celle-ci. Des chemins d'accès à tous les objets ou attributs de la vue doivent être affectés en conséquence à tous les rôles et attributs. Un chemin d'accès à un objet (cf. paragraphe 2.13 Expressions) doit ainsi être indiqué, désignant en définitive une classe ou une association correspondant au rôle. La cardinalité doit coïncider avec la performance de la vue, ce qui ne peut être contrôlé qu'au moment de l'exécution.

La déduction d'une relation à partir des conditions géométriques devrait constituer un exemple type d'application : dans une vue (connexion) à laquelle référence est faite dans une association, les bâtiments sont par exemple mis en relation avec les biens-fonds sur lesquels ils sont érigés, sur la base de leur géométrie (cf. paragraphe 2.15 Vues).

2.7.2 Intensité de la relation

Différentes intensités de relations sont distinguées, sur le modèle d'UML. Pour les expliquer, on décrira avant tout l'influence de l'intensité d'une relation sur la copie ou la suppression d'objets. D'autres considérations exercent par ailleurs une influence sur l'intensité d'une relation. Toute latitude est laissée aux systèmes de traitement pour prévoir des intensités de relation plus fines ou d'autres critères d'appréciation du comportement durant des opérations données.

- Association : les objets participants sont liés entre eux par un lien assez lâche. Si un objet participant est copié, sa copie sera liée aux mêmes objets que l'original. Si un objet participant est supprimé, la relation sera également supprimée, l'objet lié restant conservé. Sur le plan syntaxique, '--' est indiqué pour tous les rôles.
- Agrégation : il existe une relation de faible intensité entre un tout et ses parties. Les agrégations ne sont permises que dans le cas de relations à deux rôles. Sur le plan syntaxique, le rôle conduisant au tout doit être indiqué le premier, par un losange (-<>). Le rôle conduisant à une partie est défini avec '--'. Une classe d'objets peut apparaître dans différentes agrégations dans le rôle de la partie. Différents objets du tout peuvent également être affectés à un objet donné d'une partie. Contrairement aux associations, la génération d'une copie du tout entraîne la création de copies correspondantes des parties. Si plusieurs "tout" sont définis pour une partie, seule la référence au tout depuis lequel la demande de copie émane restera conservée dans la copie. En cas de suppression du tout, les parties et les éventuelles relations vers d'autres "tout" sont conservées.
- Composition : il existe une relation de forte intensité entre le tout et ses parties. Une classe d'objets peut apparaître comme partie dans plus d'une composition. A une partie d'objet cependant ne peut être subordonné qu'à un tout. Hormis cela, les compositions se comportent comme des agrégations, excepté le fait que la suppression du tout entraîne celle de ses parties. Un losange plein (-<#>) est indiqué pour le rôle conduisant au tout.

Les associations peuvent être étendues en agrégations et elles-mêmes en compositions, mais pas le contraire.

2.7.3 Cardinalité

La cardinalité définit le nombre minimal et maximal d'objets permis ; s'il n'y a qu'une valeur, le minimum égale le maximum. Si c'est une étoile au lieu d'un chiffre qui figure comme maximum, il n'y a pas de limite supérieure pour le nombre de sous-objets. L'indication de cardinalité {*} est équivalente à {0..*}. Si l'indication de cardinalité est supprimée, c'est normalement {0..*} qui s'applique. Pour les rôles d'agrégation et de composition, seuls {0..1} ou {1} sont autorisés (une partie peut seulement appartenir à un tout). L'absence d'indication équivaut à {1}.

La cardinalité ne peut être que limitée dans des extensions mais pas étendue. Si de cette manière une cardinalité de {2..4} est d'abord indiquée, une extension ne peut pas déclarer {2..5}, {7} ou {*}. L'omission de l'indication de cardinalité sous-entend, pour des attributs étendus, que la valeur héritée est reprise.

Suivant l'utilisation, la cardinalité a la portée suivante :

- Pour les sous-structures : nombre des éléments admis.
- Pour les relations duales : nombre d'objets de la classe, en fonction du rôle auquel appartient la cardinalité, qui doivent être attribués à chaque instance de la classe de l'autre rôle.
- En cas de relations multiples : nombre d'objets de la classe, conformément au rôle auquel appartient la cardinalité, qui peuvent être affectés à une combinaison particulière d'objets des classes des autres rôles.

2.7.4 Relations ordonnées

Si l'on veut que la relation soit conduite dans un ordre particulier, dans la perspective d'une classe de référence particulière, il faut l'exiger pour le rôle en qualité de propriété (ORDERED). Cet ordre est défini lors de l'établissement de la relation et doit être conservé en cas de transfert et au moment d'établir des copies (pour peu que la relation soit copiée également).

2.7.5 Accès à une relation

Comme accès à une relation, on peut attribuer la possibilité d'avoir accès à tous les objets subordonnés à un objet par une relation déterminée, ou d'obtenir tous les objets collectés par un objet de relation.

Les accès à une relation ne doivent pas être définis, mais ils découlent de la définition de la relation pour tous les rôles des classes qui ont été définies dans le même thème. Si une classe participant à une relation est définie dans un autre thème (relations plus vaste que le thème ou classe héritée), elle n'a pas d'accès à une relation. On évite ainsi que les classes ne soient encore modifiées par la suite (en dehors du cadre dans lequel elles ont été définies).

Lorsque, dans une définition de rôles, les classes autorisées sont limitées à certaines sous-classes des classes de bases mentionnées (RESTRICTED TO), il en découle des accès à une relation pour la classe de base qui seront, ainsi, hérités par toutes les sous-classes. Pour les sous-classes qui sont exclues par les restrictions, il existe aussi un accès à une relation. On ne peut toutefois pas le relier à un objet subordonné puisque la relation n'est pas possible du fait de la restriction.

Pour chaque rôle d'une relation, il existe, sous les conditions mentionnées dans la classe de base correspondante, un accès à une relation (AssociationAccess) auquel correspondent les autres rôles de la relation et qui contient aussi leurs noms. Les noms de ces accès à des relations appartiennent au même espace nominal que les noms des attributs de la classe, et à cause de cela ils ne doivent pas entrer en conflit avec eux. L'accès à l'objet de la relation lui-même ne reçoit pas de nom spécifique, mais celui-ci s'obtient par un accès à la relation précédé d'un backslash (\).

Si les classes A, B et C sont reliées entre elles par la relation ABC, que le rôle a est subordonné à la classe A, le rôle b à la classe B et le rôle c à la classe C, les accès à la relation de la classe A ont les noms b et c. A travers eux, on peut atteindre les objets subordonnés de la classe B ou de la classe C. L'accès aux objets de relation de la classe ABC peuvent également être atteints par \a ou \b.

2.8 Domaines de valeurs et constantes

Différents aspects sont liés à la représentation des domaines de valeurs. Un type de données doit être défini dans un premier temps. Les types de données d'INTERLIS sont indépendants de l'implémentation. C'est pourquoi nous ne parlerons pas d'entier (Integer) ou de réel (Real), mais tout simplement de types de données numériques (cf. paragraphe 2.8.5 Types de données numériques).

Une fois le type de données défini, d'autres précisions sont nécessaires ou possibles selon le type de données. Si la définition d'un domaine de valeurs n'est pas complète (s'il manque par exemple la longueur pour un type de chaîne de caractères), elle devra être déclarée en tant que définition abstraite (mot-clé ABSTRACT).

Les domaines de valeurs peuvent être, comme les autres concepts de base, hérités et ensuite étendus, jusqu'à ce qu'ils soient définis en tant que FINAL. De ce fait, il est important qu'une définition étendue soit toujours compatible avec la définition de base. En fait, l'extension de domaines de valeurs consiste soit en des précisions, soit en des restrictions. Le mot-clé EXTENDED (règle Property) n'est pas autorisé. Pour la clarté de la lecture, nous recommandons de répéter aussi les définitions des domaines de valeurs de base dans les extensions lorsqu'elles sont inchangées. Exemple :

```
DOMAIN
  Text (ABSTRACT) = TEXT;             !! Domaine de valeur abstrait
  GenName EXTENDS Text = TEXT*12;     !! Extension concrète
  SpezName EXTENDS GenName = TEXT*10; !! Correct
  SpezName EXTENDS GenName = TEXT*14; !! Faux, car incompatible
```

Une question importante concernant la définition des domaines de valeurs est de savoir si les valeurs "non définies" appartiennent ou non au domaine. Sans autre indication, une telle valeur appartient par défaut au domaine. Il peut également être exigé qu'un attribut soit toujours défini selon ce domaine de valeurs, c'est-à-dire qu'aucune valeur nulle n'est autorisée (mot-clé MANDATORY). MANDATORY seul n'est autorisé que lors d'extensions.

Lors de la définition d'un attribut d'une classe ou d'une structure (et uniquement dans ce cas), MANDATORY peut être mentionné si le domaine de valeurs a été déclaré avec FINAL et ne devrait ainsi plus être restreint.

Syntaxe :

```
DomainDef = 'DOMAIN'
                { Domain-Name Properties<ABSTRACT,FINAL>
                    ['EXTENDS' DomainRef ] '='
                ('MANDATORY' [ Type ] | Type ) ';' }.

Type = ( BaseType | LineType ).

DomainRef = [ Model-Name '.' [ Topic-Name '.' ] ] Domain-Name.

BaseType = ( TextType
           | EnumerationType
           | AlignmentType
           | BooleanType
           | NumericType
           | StructuredUnitType
           | CoordinateType
           | OIDType
           | BasketType
           | ClassType ).
```

Les valeurs d'attributs peuvent aussi être comparées avec des constantes (cf. paragraphe 2.13 Expressions). Les constantes sont définies comme suit :

```
Constant = ( 'UNDEFINED'
           | NumericConst
           | TextConst
           | StructUnitConst
           | EnumerationConst ).
```

Chaque type de données définit ses propres constantes.

2.8.1 Chaînes de caractères

Pour les chaînes de caractères (TEXT), l'information première est la longueur. Pour chacune des formes de définition, elle sera indiquée de manière explicite ou implicite. La forme explicite (TEXT*...) définit la longueur maximale en nombre de caractères (supérieure à zéro (Null)). Lorsque seul le mot-clé TEXT est indiqué, il s'agit d'une définition abstraite. Lorsqu'un domaine de valeurs de chaînes de caractères a été défini avec sa longueur, il ne peut être que raccourci par une extension (un rallongement conduirait à un domaine de valeurs qui ne serait plus compatible avec le domaine de valeurs de base).

La longueur de chaînes de caractères d'INTERLIS décrit le nombre de caractères tels qu'ils seront présentés à l'utilisateur et non le nombre de places en mémoire dont un système a besoin pour la représentation d'une chaîne de caractères.

Remarque : dans le contexte d'INTERLIS, la longueur d'une chaîne de caractères est définie comme le nombre de caractères affectés à la classe de combinaison canonique n° 0 selon le standard Unicode, après décomposition de la chaîne dans la forme canonique d'Unicode (cf. www.unicode.org/unicode/reports/tr15). Ainsi, la chaîne de caractères <LATIN CAPITAL LETTER C WITH CIRCUMFLEX><COMBINING CEDILLA> bénéficie d'une longueur 1 comme la chaîne équivalente <LATIN CAPITAL LETTER C>< COMBINING CIRCUMFLEX ACCENT><COMBINING CEDILLA>. Selon la définition ci-dessus, des lettres à ligature comme "fl" ou "ffi" ont une longueur de 1. Pour cette raison, il est déconseillé d'utiliser de telles formes de représentation pour des attributs de type chaîne de caractères.

Le type de chaîne de caractères nom (mot-clé NAME) définit un domaine de valeurs, qui est conforme à celui des noms d'INTERLIS (cf. paragraphe 2.2.2 Noms). Il est inclus dans la classe prédéfinie METAOBJECT (cf. modèle INTERLIS prédéfini) et avant tout dans les classes pour les systèmes de référence et les signatures (cf. paragraphe 2.10.3 Système de référence et paragraphe 2.16 Représentations graphiques), car c'est à cet endroit que les attributs de données doivent coïncider avec les éléments de description des modèles.

En tant que type de chaîne de caractères complémentaires, l'URI (Uniform Resource Identifier) est introduit, par exemple dans des adresses http-, ftp- ou mailto- (cf. paragraphe 1.2 Utilisation de modèles dans IETF RFC 2396 du standard Internet à l'adresse www.w3.org). La longueur d'un URI est limitée à 1023 caractères dans INTERLIS. Sa définition est la suivante :

```
DOMAIN
  URI (FINAL) = TEXT*1023;  !! ATTENTION: Selon IETF RFC 2396
  NAME (FINAL) = TEXT*255;  !! ATTENTION: Selon chapitre 2.2.2 Noms
```

Syntaxe :

```
TextType = ( 'TEXT' ['*' MaxLength-PosNumber ]
           | 'NAME'
           | 'URI' ).

TextConst = String.
```

2.8.2 Enumérations

Une énumération sert à définir les valeurs admises pour ce type. Elle n'est toutefois pas simplement linéaire, mais structurée sous forme d'un arbre dont les feuilles (et non les nœuds) forment l'ensemble des valeurs admises. Exemple :

```
DOMAIN
  Couleurs = (rouge (fonce, orange, carmin),
              jaune,
              vert (clair, fonce));
```

Les valeurs suivantes, décrites à l'aide de constantes, sont alors admises :

```
#rouge.fonce #rouge.orange #rouge.carmin #jaune #vert.clair #vert.fonce
```

Un emboîtement est indiqué entre parenthèses. Les noms des éléments de chaque emboîtement doivent être sans équivoque. Le niveau d'emboîtement est librement définissable.

Un ordre est défini pour les éléments si une énumération est ordonnée (mot-clé ORDERED). Si l'énumération est circulaire (mot-clé CIRCULAR), l'ordre des éléments est défini comme si l'énumération était ordonnée. Il est en outre indiqué que le premier élément suit à nouveau le dernier dans la séquence.

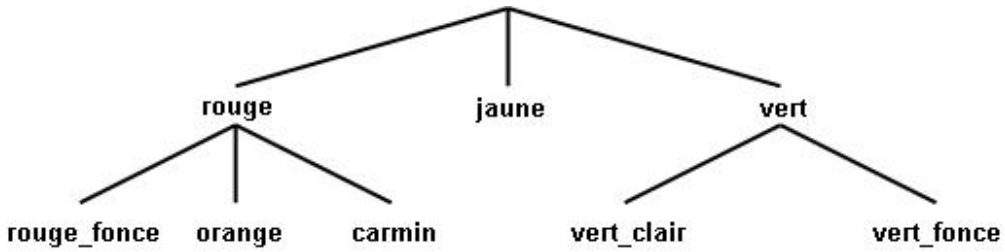


Figure 8 : Exemple d'énumération.

Exemples :

```
DOMAIN
  Position = (bas, milieu, haut) ORDERED;
  JoursSemaine = (JoursOuvrables (Lundi, Mardi, Mercredi, Jeudi,
                                  Vendredi, Samedi),
                  Dimanche) CIRCULAR;
```

Syntaxe :

```
EnumerationType = Enumeration [ 'ORDERED' | 'CIRCULAR' ].

Enumeration = '(' EnumElement { ',' EnumElement } [ ':' 'FINAL' ]
              | 'FINAL' ')'.

EnumElement = EnumElement-Name { '.' EnumElement-Name } [Sub-Enumeration].

EnumerationConst = '#' ( EnumElement-Name { '.' EnumElement-Name }
                                          [ '.' 'OTHERS' ]
                       | 'OTHERS' ).
```

Dans le cas de nouvelles définitions d'énumérations (définitions primaires, éléments complémentaires d'une extension), la composition de EnumerationConst des EnumElement n'est possible qu'à partir d'un seul nom. Plusieurs noms ne sont autorisés, que pour d'identifier une extension d'un élément de base d'une énumération.

Des énumérations peuvent, d'autre part, être élargies dans la mesure où sont définies des sous-énumérations pour des feuilles (éléments de l'énumération sans sous-énumération) de l'énumération actuelle. La définition élargie intègre désormais des nœuds, en partant de feuilles actuelles, pour lesquels aucune valeur ne peut être définie.

Toute énumération partielle peut par ailleurs être complétée dans des extensions par des éléments supplémentaires (nœuds ou feuilles). Outre les éléments nommés, les énumérations de base comprennent de ce fait des éléments supplémentaires potentiels qui ne seront définis que dans des extensions. De telles valeurs potentielles peuvent être appelées au niveau de base dans des expressions, des arguments de fonctions et des assignations de signatures (cf. paragraphe 2.13 Expressions, cf. paragraphe 2.14 Fonctions, cf. paragraphe 2.16 Représentations graphiques) via la valeur OTHERS, laquelle n'est toutefois pas une valeur admise dans le cadre de la classe à laquelle

appartient l'objet. La possibilité de pouvoir ajouter des éléments d'énumération supplémentaires dans des extensions peut être retirée en déclarant l'énumération partielle comme étant finale (FINAL), ce qui peut s'effectuer à la suite du dernier élément énuméré ou dans le cadre d'une extension, sans pour autant que de nouveaux éléments aient à être ajoutés.

Les énumérations circulaires (mot-clé CIRCULAR) ne peuvent pas être étendues.

Exemple :

```
DOMAIN
  Couleur = (rouge,
             jaune,
             vert);
  CouleurPlus EXTENDS Couleur = (rouge (fonce, orange, carmin),
                                 vert (clair, fonce: FINAL),
                                 bleu);
  CouleurPlusPlus EXTENDS CouleurPlus = (rouge (FINAL),
                                         bleu (clair, fonce));
```

Les valeurs suivantes, décrites à l'aide de constantes, sont admises pour CouleurPlus :

```
#rouge.fonce #rouge.orange #rouge.carmin #jaune #vert.clair #vert.fonce #bleu
```

Et en plus pour CouleurPlusPlus :

```
#bleu.clair #bleu.fonce
```

L'indication de FINAL dans les niveaux de vert de CouleurPlus implique qu'il n'est plus possible de définir des niveaux de vert supplémentaires dans CouleurPlusPlus. L'indication de FINAL pour la subdivision du rouge dans CouleurPlusPlus empêche l'ajout de nuances de rouge supplémentaires dans d'éventuelles extensions de CouleurPlusPlus.

2.8.3 Alignement du texte

Pour la préparation des plans et des cartes, les positions des textes doivent être fixées. Ainsi, la position du texte par rapport au point d'insertion doit-elle être définie. L'alignement horizontal définit trois positions : gauche, droite ou au milieu. L'alignement vertical définit la position du texte dans le sens de sa hauteur.

La distance Cap-Base définit la hauteur des majuscules. Le jambage inférieur d'un caractère se situe dans la zone définie par la distance Base-Bottom.

Les alignements horizontaux et verticaux peuvent être conçus en tant qu'énumérations prédéfinies :

```
DOMAIN
  HALIGNMENT (FINAL) = (Left, Center, Right) ORDERED;
  VALIGNMENT (FINAL) = (Top, Cap, Half, Base, Bottom) ORDERED;
```

Syntaxe :

```
AlignmentType = ( 'HALIGNMENT' | 'VALIGNMENT' ).
```

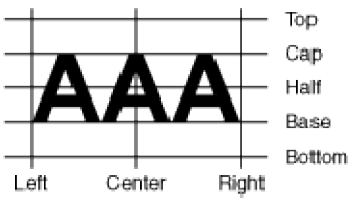


Figure 9 : Alignement horizontal (HALIGNMENT) et vertical (VALIGNMENT) du texte.

2.8.4 Booléen

Le type booléen (Boolean) comprend les valeurs vrai (true) et faux (false). Il peut être conçu en tant qu'énumération prédéfinie :

```
DOMAIN
  BOOLEAN (FINAL) = (false, true) ORDERED;
```

Syntaxe :

```
BooleanType = 'BOOLEAN'.
```

2.8.5 Types de données numériques

L'indication la plus importante pour les types de données numériques est la valeur maximale et minimale, incluant le nombre de positions (décimales) ainsi que le facteur d'échelle. Accessoirement, il est possible d'indiquer si ce type est circulaire (mot-clé CIRCULAR), c'est-à-dire que si l'on augmente d'une unité la dernière position significative de la valeur maximale, on obtient une valeur ayant par nature la même signification que la valeur minimale (exemple : 0..359 degrés pour des angles). Si l'indication des valeurs minimale et maximale est manquante (mot-clé NUMERIC), le domaine de valeurs est considéré comme abstrait.

```
DOMAIN
  Angle1 = 0.00 .. 359.99 CIRCULAR [degree];  !! correct
  Angle2 = 0.00 .. 360.00 CIRCULAR [degree];  !! syntaxiquement correct,
                                              !! mais fondamentalement faux,
                                              !! 360.01 correspond à la valeur
                                              !! minimale 0.00
```

Le nombre de chiffres doit être identique pour les valeurs minimale et maximale. Le cadrage permet la description de valeurs flottantes, mais les valeurs minimale et maximale sont alors à présenter avec une mantisse, c.-à-d. commençant par zéro (0), suivies du point décimal (.) puis d'un premier chiffre obligatoirement différent de zéro (0). Le cadrage de la valeur minimale doit être inférieur à celui de la valeur maximale. La notation des valeurs minimale et maximale ne donne toutefois aucune indication sur la manière dont les valeurs doivent être transférées (si un domaine de valeurs est par exemple défini par 000 .. 999, cela ne signifie pas que la valeur 7 devra être transférée dans le format 007). Les valeurs flottantes font exception à cette règle, étant à transférer avec une mantisse et un cadrage.

Lors d'extensions, les valeurs minimale et maximale ne peuvent qu'être restreintes. Le domaine numérique de valeurs est ainsi plus petit. Observons la situation suivante :

```
DOMAIN
  Normal = 0.00 .. 7.99;
  Precis EXTENDS Normal = 0.0000 .. 7.9949;   !! correct, car Normal est aussi
                                              !! représentable
  Precis EXTENDS Normal = 0.0000 .. 7.9999;   !! faux, car arrondi hors Normal
```

Afin d'expliquer de manière plus précise la signification des valeurs, une unité de mesure peut être indiquée (cf. paragraphe 2.9 Unités). Des unités de mesure ne sont autorisées que tant que le domaine de valeurs reste lui-même indéfini (mot-clé NUMERIC).

Les règles suivantes s'appliquent pour les extensions :

- Si un domaine de valeurs de base concret ne présente aucune unité de mesure, les extensions du domaine de valeurs de base ne peuvent pas non plus en indiquer.
- Si une unité de mesure abstraite est utilisée dans le domaine de valeurs de base, seules des unités de mesure qui sont des extensions de cette unité peuvent être utilisées dans des extensions du domaine de valeurs de base.
- Si une unité de mesure concrète est utilisée dans le domaine de valeurs de base, elle ne peut pas faire l'objet d'un dépassement de définition dans des extensions.

Exemple :

```
UNIT
  foot [ft] = 0.3048 [m];

DOMAIN
  Distance (ABSTRACT) = NUMERIC [Length];
  DistMetre (ABSTRACT) EXTENDS Distance = NUMERIC [m];
  DistPied (ABSTRACT) EXTENDS Distance = NUMERIC [ft];
  JalonMetre EXTENDS DistMetre = 0.00 .. 100.00 [m];
  JalonPied EXTENDS DistPied = 0.00 .. 100.00 [ft];
  JalonPied2 (ABSTRACT) EXTENDS JalonMetre = NUMERIC [ft]; !! faux: m vs. ft
```

Un système scalaire peut également être affecté à un domaine de valeurs numériques (cf. paragraphe 2.10.3 Systèmes de référence). Les valeurs se rapportent alors à l'origine définie par ce système scalaire, dans lequel il s'agit donc de valeurs *absolues*. Si l'unité adoptée dans la classe du système scalaire n'est pas ANYUNIT, une unité compatible avec celle du système de référence doit être indiquée pour le type de données numériques. Si l'on se réfère à un système de coordonnées, l'axe auquel les valeurs se rapportent peut être indiqué, l'unité devant être compatible avec celle de l'axe concerné. En l'absence de cette indication, la référence n'est pas définie avec plus de précision mais se déduit du domaine de spécialité (il s'agira par exemple d'altitudes ellipsoïdiques si on se réfère à un ellipsoïde dans le cas de hauteurs). Si l'on se réfère à un autre domaine de valeurs, le système de référence s'appliquant à ce dernier vaudra également pour le domaine de valeurs concerné. Dans ce cas, l'indication des axes ne peut être omise que s'il s'agit d'un domaine de valeurs numériques. L'indication des axes est obligatoire pour un domaine de valeurs de coordonnées. L'indication du système de référence ne peut plus être modifiée dans des extensions.

Si la valeur numérique représente un angle, son orientation peut être définie. Dans le cas de directions, le système de coordonnées (défini par un domaine de valeurs de coordonnées) sur lequel elles se basent peut être indiqué, imposant ainsi la direction zéro (gisement) et le sens de rotation (cf. paragraphe 2.8.7 Coordonnées). Cette indication ne peut plus être modifiée dans des extensions.

Outre les chiffres décimaux, le nombre Pi (mot-clé PI) et la base e du logarithme naturel (mot-clé LNBASE) sont définis comme des constantes numériques.

Syntaxe :

```
NumericType = ( Min-Dec '..' Max-Dec | 'NUMERIC' ) [ 'CIRCULAR' ]
              [ '[' UnitRef ']' ]
              [ 'CLOCKWISE' | 'COUNTERCLOCKWISE' | RefSys ].

RefSys = ( '{' RefSys-MetaObjectRef [ '[' Axis-PosNumber ']' ] '}'
         | '<' Coord-DomainRef [ '[' Axis-PosNumber ']' ] '>' ).

DecConst = ( Dec | 'PI' | 'LNBASE' ).

NumericConst = DecConst [ '[' UnitRef ']' ].
```

2.8.6 Domaines de valeurs structurées

Les domaines de valeurs structurées se construisent à partir des unités structurées (cf. paragraphe 2.9 Unités). Les dates et l'heure peuvent par exemple être traités ainsi. Il est nécessaire d'indiquer les valeurs minimale et maximale et l'unité structurée. Il peut de plus être fait mention d'un système de référence numérique (par exemple les fuseaux horaires).

L'unité donnée (indiquée) doit être conforme à celle définie dans le système de référence, c'est-à-dire qu'elle doit coïncider avec elle, l'étendre ou être dérivée d'elle ou d'une de ses extensions.

Lors de l'héritage, les règles concernant les types de données numériques s'appliquent. Exemple :

```
DOMAIN
  Heure = 0:00:00.000 .. 23:59:59.999 [MEZ];
```

Syntaxe :

```
StructuredUnitType = Min-StructDec '..' Max-StructDec [ 'CIRCULAR' ]
                     '[' Structured-UnitRef ']'
                     [ 'CLOCKWISE' | 'COUNTERCLOCKWISE' ]
                     [ RefSys ].

StructUnitConst = StructDec [ '[' UnitRef ']' ].
```

2.8.7 Coordonnées

Les coordonnées peuvent être définies en une, deux ou trois dimensions et se composent par conséquent d'une valeur unique, d'une paire ou d'un triplet de valeurs. Il est admis de n'introduire la deuxième ou la troisième dimension qu'au stade d'une extension. Pour chaque dimension, le domaine de valeurs numériques de même qu'une éventuelle unité de mesure et un système de coordonnées (numéros des axes compris) doivent être indiqués. Les règles valant pour les types de données numériques s'appliquent également ici. Seules des unités de mesure concrètes peuvent être indiquées. Si aucun système de référence n'est indiqué et si les unités de mesure ne sont pas définies ou le sont comme unités de longueur, alors un système logiciel implémentant le modèle peut supposer que des coordonnées cartésiennes sont utilisées.

Si une indication de rotation est fournie (mot-clé ROTATION), il peut être renvoyé à un tel système de coordonnées de référence dans le cadre de définitions de directions (cf. paragraphe 2.8.5 Types de données numériques). La définition de rotation fixe l'axe correspondant à une direction zéro et celui correspondant à un angle droit compté dans le sens positif. Elle peut également être omise dans une définition de coordonnées concrète et éventuellement être ajoutée dans une extension.

Les indications concernant les références axiales et la rotation ne peuvent pas être modifiées dans des extensions. Exemple :

```
DOMAIN
  CHLKoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
                    60000.00 .. 320000.00 [m] {CHLV03[2]},
                   ROTATION 2 -> 1;
```

Dans le cadre de la définition de chacune des deux dimensions, on a indiqué, en sus du domaine admis, les unités et le système de référence (numéros des axes compris) auxquels se rapportent les coordonnées. Les axes effectifs sont définis par le système de référence. L'information de rotation indique que la direction de l'origine va du deuxième axe vers le premier. Le système national suisse définit que le premier axe est dirigé vers l'est, que le deuxième axe est dirigé vers le nord et que la direction d'origine est celle du nord. La rotation se fait dans le sens horaire.

```
DOMAIN
  WGS84Koord = COORD –90:00:00 ..  90:00:00          [Units.Angle_DMS] {WGS84[1]},
                      0:00:00 .. 359:59:59 CIRCULAR [Units.Angle_DMS] {WGS84[2]},
                     -1000.00 .. 9000.00            [m] {WGS84Alt[1]};
```

Les coordonnées géographiques sont d'ordinaire exprimées en degrés et se rapportent à un ellipsoïde (WGS84 par exemple). Les altitudes sont en revanche exprimées en mètres et se réfèrent à un système altimétrique à axe unique basé sur un ellipsoïde.

Syntaxe :

```
CoordinateType = 'COORD' NumericalType
                     [ ',' NumericalType [ ',' NumericalType ]
                         [ ',' RotationDef ] ].
```

```
NumericalType = ( NumericType | StructuredUnitType ).

RotationDef = 'ROTATION' NullAxis-PosNumber '->'
              PiHalfAxis-PosNumber.
```

Dans le cas où il n'y a au moins pas un domaine numérique défini (avec NumeralType), l'attribut correspondant doit être déclaré abstrait.

2.8.8 Domaines de valeurs d'identifications d'objets

Les objets identifiables sont toujours pourvus d'une identification. Afin que les systèmes puissent prévoir l'espace mémoire à réserver pour leur stockage, des domaines de valeurs correspondants sont à définir et à affecter aux thèmes (cf. paragraphe 2.5.2 Thème).

Syntaxe :

```
OIDType = 'OID' ( 'ANY' | NumericType | TextType ).
```

INTERLIS 2 définit les domaines de valeurs d'OID suivants (cf. annexe A) :

```
DOMAIN
  ANYOID (ABSTRACT) = OID ANY;
  I32OID = OID 0 .. 2147483647; !! valeurs entières positives mémorisables
                                !! sur 4 octets
  STANDARDOID = OID TEXT*16;    !! conformément à l'annexe E
```

Chaque domaine de valeur d'OID numérique ou textuel est considéré comme une extension directe ou indirecte du domaine de valeur d'OID abstrait ANYOID.

2.8.9 Conteneur

Il est indispensable de pouvoir modéliser les conteneurs si une gestion de conteneur est à modéliser. Elle s'effectue en recourant au mot-clé BASKET. Sans autre indication, n'importe quel conteneur peut être décrit dans un tel attribut. Les conteneurs permis peuvent cependant être restreints. On peut d'une part, par des propriétés (Properties), indiquer les types de conteneurs (cf. 2.5.2 Thème) permis :

- DATA : conteneur de données. Seules des instances de classes (et leurs sous-structures) figurent dans les conteneurs.
- VIEW : conteneur de vues. Dans ces derniers figurent les instances (virtuelles) de vues d'un thème de vue (mot-clé VIEW TOPIC).
- BASE : conteneurs avec des instances de classes et de vues utilisées par les définitions graphiques.
- GRAPHIC : conteneurs avec signatures graphiques. Un tel conteneur sera aussi appelé bibliothèque de signatures.

Dans des extensions, les types de conteneurs ne peuvent être que plus restrictifs, mais pas complétés.

D'autre part, les conteneurs autorisés peuvent être limités par l'indication du thème admis.

Syntaxe :

```
BasketType = 'BASKET' Properties<DATA,VIEW,BASE,GRAPHIC>
                 [ 'OF' [ Model-Name '.' ] Topic-Name ].
```

La désignation du thème (nom du modèle, nom du thème), le type de conteneur (DATA, VIEW, BASE, GRAPHIC) et l'identificateur du conteneur constituent les champs de données gérés pour définir un conteneur. Par conséquent, un type de conteneur peut être interprété comme une structure présentant la définition suivante :

```
STRUCTURE Basket (ABSTRACT) =
  Model: MANDATORY NAME;
```

```
    Topic: MANDATORY NAME;
    Kind: MANDATORY (Data, View, Base, Graphic);
    Ident (ABSTRACT): MANDATORY ANYOID;
  END Basket;
```

2.8.10 Types de classes

Il peut être judicieux que des objets de données contiennent des renvois vers des classes bien définies.

Syntaxe :

```
ClassType = ( 'CLASS'
                  [ 'RESTRICTED' 'TO' ClassOrAssociationRef
                                      { ',' ClassOrAssociationRef } ]
            | 'STRUCTURE'
                  [ 'RESTRICTED' 'TO' StructureRef
                                      { ',' StructureRef } ] ).
```

L'indication de STRUCTURE a pour effet d'admettre une structure ou classe quelconque et celle de CLASS (également admise comme extension de STRUCTURE) d'autoriser une classe quelconque (mais pas de structures). Si seules des structures ou des classes données ainsi que leurs extensions peuvent être admises, alors celles-ci sont à énumérer (RESTRICTED TO). Toutes les structures ou classes admises doivent à nouveau être énumérées dans des extensions. Elles ne peuvent cependant pas présenter de contradiction avec la définition de base. Dès que de telles restrictions sont définies, STRUCTURE ne peut plus être étendu par CLASS.

2.8.11 Polylignes

2.8.11.1 Géométrie d'une polyligne

On entend concrètement par portion de courbe, une forme unidimensionnelle ne présentant ni interruption ni point anguleux ni point double d'aucune sorte (voir Figure 10 et Figure 11). Les portions de courbes sont donc régulières et dépourvues de toute ambiguïté. Des segments de droite, des arcs de cercle, de paraboles et de clothoïdes sont des exemples de portions de courbes. Chaque portion de courbe comporte deux *extrémités* (un point initial et un point final) qui ne peuvent pas coïncider. Les autres points de la portion de courbe sont appelés les *points intérieurs* et forment l'*intérieur* de la portion de courbe.

> Définition exacte (les notions mathématiques qui ne seront pas expliquées plus loin, mais dont la définition peut être trouvée dans tout ouvrage de mathématiques sont figurées "*en italique et entre guillemets"*) : une *portion de courbe* est un sous-ensemble de l'*"espace euclidien"* (simplement appelé *espace* dans la suite) *"à trois dimensions"*, *"image"* d'un *"intervalle"* (de la *"droite numérique"*) par une *"application" "injective"* et *"régulière"*. Les points initial et final de la portion de courbe sont les images des extrémités de l'intervalle. Une *portion de courbe* est dite *plane* lorsqu'elle est contenue dans un *plan* (*"sous-espace" "à deux dimensions"* de l'espace).

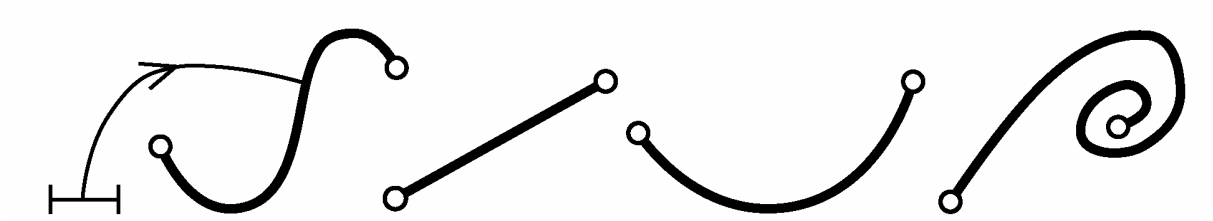


Figure 10 : Exemples de portions de courbes planes.

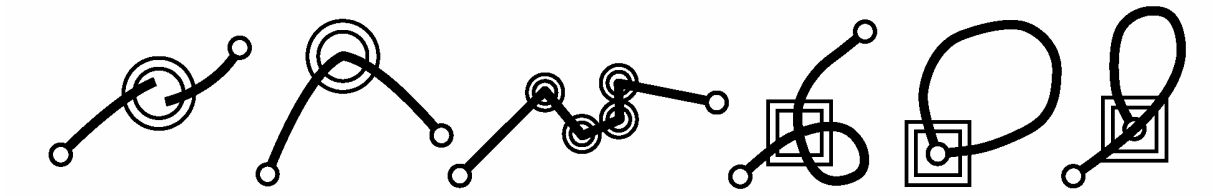


Figure 11 : Exemples d'ensembles plans qui ne constituent pas des portions de courbes (un double cercle signifie "non régulier" et un carré double signifie "non injectif").

Une polyligne est une suite finie de portions de courbes. A l'exception de la première portion de courbe, le point initial coïncide à chaque fois avec le point final de la portion de courbe précédente. Ces points sont appelés les *points d'appui* de la polyligne. En pratique, une polyligne peut contenir des portions de courbes utilisées à plusieurs reprises, des portions de courbes avec des points d'appui coïncidant, des portions de courbes se recoupant ainsi que des portions de courbes commençant ou finissant à l'intérieur d'autres portions de courbes (voir Figure 12 et Figure 13). Une *polyligne simple* ne présente aucun point d'intersection avec elle-même (voir Figure 14). Le point initial de la première portion de courbe coïncide par ailleurs avec le point final de la dernière portion dans le cas d'une *polyligne simple fermée*.

Définition exacte (les notions mathématiques qui ne seront pas expliquées plus loin, mais dont la définition peut être trouvée dans tout ouvrage de mathématiques sont figurées "*en italique et entre guillemets"*) : une *polyligne* est un sous-ensemble de l'espace, *"image"* d'un *"intervalle"* par une *"application"* (dite *associée*) "*continue"* et *"régulière par portions"* (mais pas nécessairement *"injective"*) et ne présentant qu'un nombre fini de *"points non réguliers"*, autrement dit de *points anguleux.* Les points initial et final coïncident dans le cas d'une *polyligne fermée*. L'application associée à une *polyligne simple* est également *"injective"*. L'application associée à une *polyligne simple fermée* est *"injective"*, sauf pour les points initial et final qui sont identiques.

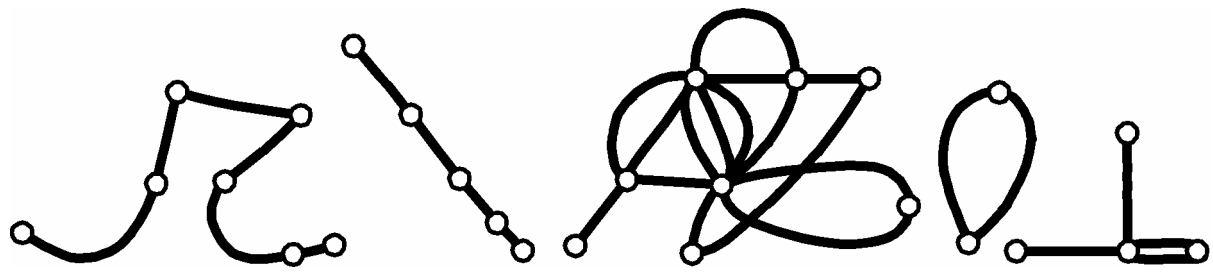


Figure 12 : Exemples de polylignes (planes).

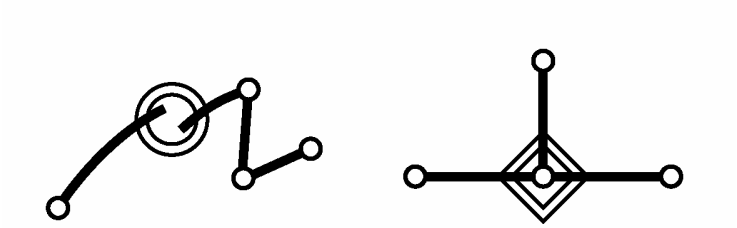


Figure 13 : Exemples d'ensembles plans qui ne constituent pas des polylignes (le double cercle signifie "non continu ", le losange signifiant "non image d'un intervalle").

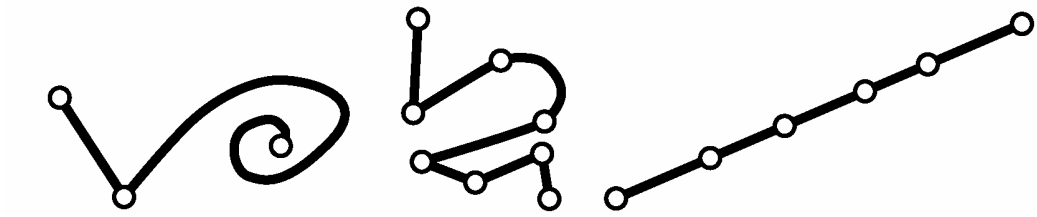


Figure 14 : Exemples de polylignes simples (planes).

2.8.11.2 Polyligne comportant des segments de droite et des arcs de cercle en tant qu'éléments de portion de courbe prédéfinis

INTERLIS 2 reconnaît des polylignes orientées (DIRECTED POLYLINE) ou non orientées (POLYLINE). Les polylignes sont en outre utilisées dans le cadre de surfaces simples et de partitions de territoires (cf. paragraphe 2.8.12 Surfaces simples et partitions de territoires).

La définition d'un domaine de valeurs concret de polyligne comprend toujours l'indication des formes de portions de courbes permises, au moyen d'une énumération, pouvant par exemple englober des segments de droite (mot-clé STRAIGHTS), des arcs de cercle (mot-clé ARCS) ou toute autre possibilité (cf. paragraphe 2.8.11.3 Formes de portions de courbes supplémentaires), de même que l'indication du domaine de valeurs des points d'appui. Ces indications peuvent être omises pour un domaine de valeurs abstrait de polyligne. Les règles suivantes s'appliquent aux extensions de domaines de valeurs :

- La forme de la portion de courbe peut être réduite mais pas complétée.
- Le domaine de valeurs de coordonnées indiqué dans le cadre d'une extension d'un domaine de valeurs d'une polyligne doit constituer une limitation du domaine de valeurs de coordonnées du domaine de valeurs de la polyligne de base, pour autant qu'un tel domaine soit défini.

Les portions de courbes sont toujours interprétées comme une extension de la structure de base LineSegment. Le domaine de valeurs de coordonnées utilisé dans ce cadre est celui indiqué dans la définition des lignes.

```
STRUCTURE LineSegment (ABSTRACT) =
  SegmentEndPoint: MANDATORY LineCoord;
END LineSegment;

STRUCTURE StartSegment (FINAL) EXTENDS LineSegment =
END StartSegment;

STRUCTURE StraightSegment (FINAL) EXTENDS LineSegment  =
END StraightSegment;

STRUCTURE ArcSegment (FINAL)EXTENDS LineSegment  =
  ArcPoint: MANDATORY LineCoord;
  Radius: NUMERIC [LENGTH];
END ArcSegment;
```

La première portion de courbe d'une polyligne est toujours du type StartSegment et se compose du seul point initial qui est également le point final de ce segment initial (Startsegment). Un segment de droite comprend un point final et définit ce faisant un tronçon linéaire reliant le point final de la portion de courbe précédente à son propre point final. Aucune autre indication n'est requise pour un segment initial (Startsegment) et un segment de droite. Les extensions correspondantes du segment de droite (LineSegment) sont par conséquent vides. Deux points d'appui consécutifs (SegmentEndPoints) ne peuvent pas se confondre dans la projection.

Un arc de cercle décrit une portion de courbe qui, dans la projection, apparaît effectivement comme un arc de cercle. Un arc de cercle est décrit par son point final et par un point intermédiaire dont seule la position planimétrique est importante. Si les coordonnées sont en 3 dimensions, la hauteur de l'arc de cercle sera interpolée linéairement. Le point intermédiaire ne constitue pas un point d'appui de la polyligne. Il doit si possible se situer à mi-chemin entre les deux extrémités. Le point intermédiaire étant indiqué avec le même niveau de précision que les points d'appui, le rayon calculé peut s'écarter considérablement du rayon effectif. Si ce dernier est indiqué, c'est donc lui qui prime pour la définition de l'arc de cercle. Le rôle du point intermédiaire ne consiste plus alors qu'à établir laquelle parmi les quatre possibilités d'arc de cercle existantes est celle recherchée. Dans ce cas également, le point intermédiaire ne peut s'écarter de plus de deux unités de la trace de l'arc de cercle déterminée à partir du rayon.

Il peut être exigé d'une polyligne qu'elle soit de type simple, en d'autres termes, qu'elle ne se recoupe pas et qu'elle n'utilise pas une même portion de courbe à de multiples reprises (mot-clé WITHOUT OVERLAPS). Un arc de cercle et un segment de droite (ou un autre arc de cercle) consécutifs d'une polyligne partageant un point intérieur (cf. ci-dessus pour la définition) en plus de leur point d'appui commun sont également admis dans le cas d'une polyligne simple si la flèche séparant les deux tronçons (segment de droite – arc de cercle, cf. ci-dessous, ou arc de cercle – arc de cercle) reste inférieure ou égale au nombre décimal indiqué pour WITHOUT OVERLAPS > (voir Figure 15a). Deux raisons expliquent cette règle : il est d'une part inévitable que de légers recouvrements se produisent pour des raisons numériques dans le cas de certains arcs de cercles (tangents par exemple) et il convient d'autre part de tolérer des recouvrements d'une certaine ampleur (de quelques centimètres par exemple) en cas de prise en charge de données initialement saisies par voie graphique si l'on souhaite s'épargner un traitement complémentaire très conséquent. L'indication de la tolérance doit s'effectuer dans la même unité que celle utilisée pour les coordonnées des points d'appui. Pour des raisons numériques, elle ne peut pas être inférieure à deux unités du dernier chiffre significatif (dernière position décimale). Toute nouvelle définition est interdite et cette tolérance est obligatoire dans le cas de surfaces indépendantes et de partitions de territoires.

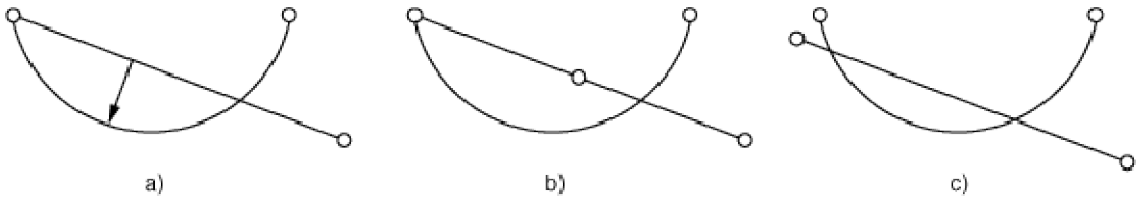


Figure 15 : a) La flèche ne doit pas excéder la tolérance indiquée ; b), c) recouvrements non permis pour une polyligne, le segment de droite et l'arc de cercle en intersection n'ayant pas leur origine en un point d'appui commun.

Des polylignes non orientées peuvent être étendues en polylignes orientées dans le cadre de définitions de domaines de valeurs et d'extensions d'attributs (cf. paragraphe 2.8.12.4 Possibilités d'extension).

Si des polylignes sont orientées, leur sens doit toujours être conservé (également dans le cas d'un transfert de données).

Le domaine de valeur des coordonnées est défini pour les points d'appui. La condition d'existence REQUIRED IN (cf. paragraphe 2.12 Règles d'intégrité et paragraphe 2.13 Expressions) permet en outre d'exiger que les coordonnées ne soient pas quelconques mais qu'elles correspondent au contraire à celles de points de classes données.

Si les coordonnées des points d'appui sont de type abstrait, la polyligne doit à son tour être déclarée comme étant abstraite.

Syntaxe :

```
LineType = ([ 'DIRECTED' ] 'POLYLINE' | 'SURFACE' | 'AREA' )
           [ LineForm ] [ ControlPoints ] [ IntersectionDef ]
           [ LineAttrDef ].

LineForm = 'WITH' '(' LineFormType {',' LineFormType} ')'.

LineFormType = ( 'STRAIGHTS' | 'ARCS'
               | [ Model-Name '.' ] LineFormType-Name ).

ControlPoints = 'VERTEX' CoordType-DomainRef.

IntersectionDef = 'WITHOUT' 'OVERLAPS' '>' Dec.
```

Il est possible de définir des attributs supplémentaires pour les objets effectifs de polylignes (appelés attributs de lignes, règle LineAttrDef permise uniquement pour SURFACE et AREA) afin que des attributs différents puissent être assignés à de portions différentes de la frontière (paragraphe 2.8.12 Surfaces simples et partitions de territoires). Une structure est indiquée pour la définition, mais elle ne peut présenter que des attributs et des appels de fonctions locaux. En cas d'extension d'un attribut de surface simple ou de partition de territoire pour lequel un attribut de ligne a été défini au moyen d'une structure, l'attribut étendu peut ne présenter aucun attribut de ligne ou sa structure doit être une extension de la structure de base. Syntaxe :

```
LineAttrDef = 'LINE' 'ATTRIBUTES' Structure-Name.
```

2.8.11.3 Formes de portions de courbes supplémentaires

Des formes de portions de courbes supplémentaires sont définissables en plus des segments de droite et des arcs de cercle. Outre le nom, il convient d'indiquer la structure dans le respect de laquelle une portion de courbe est décrite. Ces définitions de formes de portions de courbes supplémentaires et de structures correspondantes ne sont admises que dans le cadre d'un contrat, étant donné qu'il ne peut être supposé qu'un système puisse accepter toutes les formes de courbes.

Syntaxe :

```
LineFormTypeDef = 'LINE' 'FORM'
                      { LineFormType-Name ':' LineStructure-Name ';' }.
```

Une structure de ligne doit toujours être une extension de la structure de ligne LineSegment définie par l'intermédiaire d'INTERLIS (cf. paragraphe 2.8.11.2 Polyligne comportant des segments de droite et des arcs de cercle en tant qu'éléments de portion de courbe prédéfinis).

2.8.12 Surfaces simples et partitions de territoires

2.8.12.1 Géométrie de surfaces

Des surfaces planes sont généralement suffisantes pour la modélisation des géodonnées. INTERLIS accepte toutefois des surfaces planes générales. En pratique, une telle surface plane générale est délimitée par une frontière extérieure et éventuellement par une ou plusieurs frontières intérieures (voir

Figure 20). Ces lignes de frontières doivent elles-mêmes se composer de polylignes simples, lesquelles doivent pouvoir être comprises comme des polylignes simples fermées du point de vue géométrique. Elles doivent en outre être agencées de telle façon qu'il existe toujours un chemin possible reliant un point quelconque à l'intérieur de la surface à un autre point quelconque à l'intérieur de la même surface sans intersection avec une frontière ni passage par un point d'appui d'une frontière (voir Figure 19). Des points d'appui peuvent être communs à deux ou plusieurs frontières si la condition ainsi énoncée n'est pas violée. Différentes possibilités peuvent être envisagées dans de tels cas pour la partition du périmètre total de la surface en plusieurs polylignes (voir Figure 22 : Diverses partitions possibles de la frontière d'une surface générale.). INTERLIS ne fournit aucune directive quant à la sélection de l'une ou l'autre de ces possibilités. Si une telle surface est transférée à plusieurs reprises, des partitions différentes peuvent être rencontrées lors des divers transferts.

Définitions exactes (les notions mathématiques qui ne seront pas expliquées plus loin, mais dont la définition peut être trouvée dans tout ouvrage de mathématiques sont figurées "*en italique et entre guillemets"*) :

Un *élément de surface* est un sous-ensemble de l'*espace, "image"* d'un polygone plan régulier par une *"application" "régulière"* et *"injective"* (voir Figure 16 et Figure 17).

On appelle *surface* la réunion *F "connexe"* d'un nombre fini d'éléments de surface satisfaisant à la condition suivante : pour chaque point *P* de la surface, il existe un *"voisinage" déformable* en un polygone plan régulier (c.-à-d. par un *"homomorphisme"*). Si le point *P* est transféré sur la frontière du polygone lors d'une telle déformation, on dit qu'il s'agit d'un *point de la frontière de F*, dans tout autre cas, il s'agit d'un *point intérieur de F*. La *"frontière"* (c.-à-d. l'ensemble de tous les points de la frontière) d'une surface est la réunion d'un nombre fini de portions de courbes n'ayant que des points finaux en commun. Une *surface* est dite *plane* si elle constitue un sous-ensemble d'un *plan*. La *"frontière"* d'une surface plane se compose d'une *polyligne simple fermée* extérieure (appelée la *frontière extérieure*) et d'un nombre fini (pouvant le cas échéant être nul) de *polylignes simples fermées* intérieures (appelées *frontières intérieurs*). Aucun point n'est commun à la frontière extérieure et à toutes les frontières intérieures. Une portion de surface délimitée par une frontière intérieure est appelée une *enclave* (voir Figure 18, Figure 19 et Figure 20).

Une *surface* dite *générale* comporte en outre un nombre fini de *points singuliers* supplémentaires mais sa partie *intérieure* (ensemble des points intérieurs) est *"connexe"*. Un *point singulier* peut être déformé en compagnie d'un *"voisinage"* en un *ensemble* plan *en étoile* dont le point lui-même occupe le *centre*. Un *ensemble en étoile* est la réunion d'un nombre fini de surfaces triangulaires ayant le *centre* pour seul point commun. Une *surface générale* est dite *plane* lorsqu'elle constitue un sous-ensemble d'un *plan* (voir). La frontière d'une surface générale plane peut être composé de diverses manières à partir d'un nombre fini de polylignes fermées, pouvant avoir un nombre fini de points en commun et pouvant à chaque fois présenter un nombre fini de points doubles (voir Figure 22).

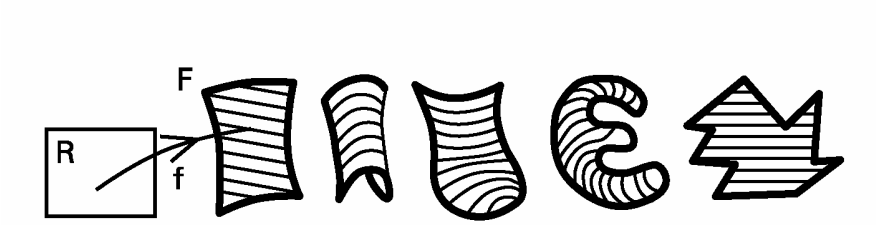


Figure 16 : Exemples d'éléments de surfaces.

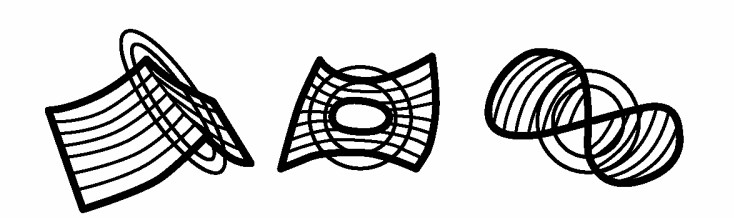


Figure 17 : Exemples d'ensembles de points dans l'espace ne constituant pas des éléments de surface (un double cercle signifie "non régulier ").

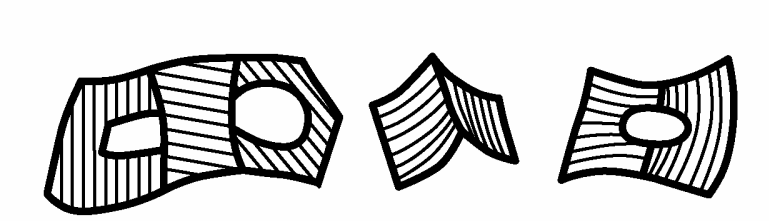


Figure 18 : Exemples de surfaces dans l'espace.

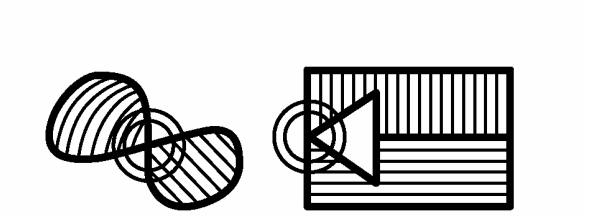


Figure 19 : Exemples d'ensembles de points plans ne constituant pas des surfaces (un double cercle signifie "point singulier").

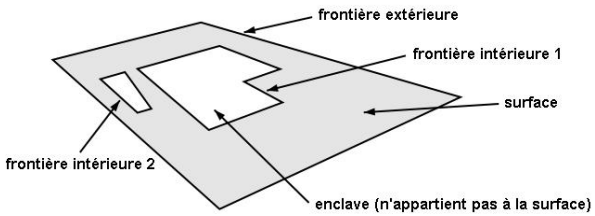


Figure 20 : Surface plane avec frontières et enclaves.

a) b)

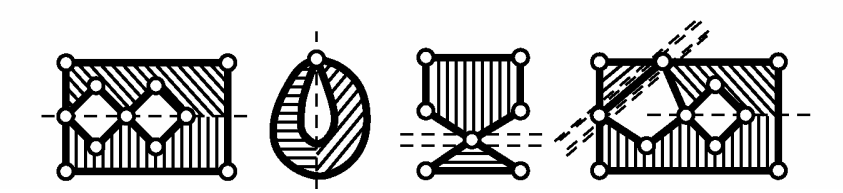


Figure 21 : a) Exemples de surfaces générales planes ; b) Exemples d'ensembles plans ne constituant pas des surfaces générales car leur intérieur n'est pas connexe. Ces ensembles plans peuvent toutefois être subdivisés en surfaces générales ("---" indique la subdivision en éléments de surface et "===" celle en surfaces générales).

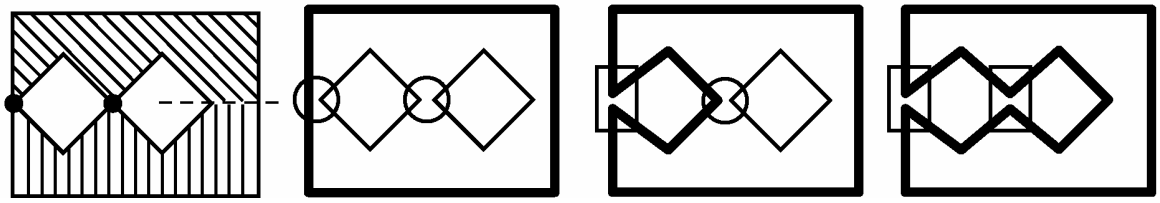


Figure 22 : Diverses partitions possibles de la frontière d'une surface générale.

La définition de surfaces indépendantes (générales) ou de surfaces (générales) de la partition d'un territoire s'accompagne de la détermination de la limite supérieure de tolérance de chevauchement pour les portions de courbe de la frontière (WITHOUT OVERLAPS doit être indiqué directement ou être hérité pour des définitions concrètes de surfaces indépendantes ou de partitions de territoires). Dans le cas des surfaces indépendantes, l'interdiction de recouvrement ou d'intersection ne s'applique pas seulement aux portions de courbes d'une polyligne donnée mais à toutes les portions de courbes de toutes les polylignes de la frontière de la surface. S'agissant de surfaces d'une partition du territoire, cette interdiction s'étend même à toutes les polylignes intervenant dans la partition du territoire. Enfin, des polylignes n'appartenant pas à la frontière d'une surface (générale) en sont exclues conformément à la définition de la surface générale.

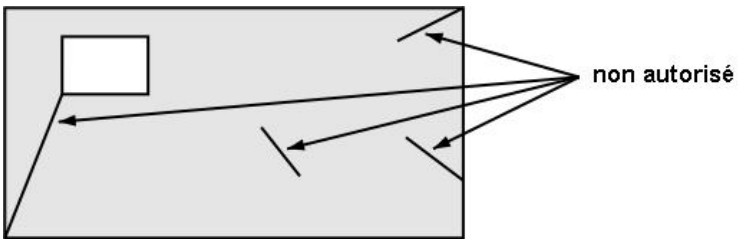


Figure 23 : Lignes non autorisées pour les surfaces.

2.8.12.2 Surfaces indépendantes

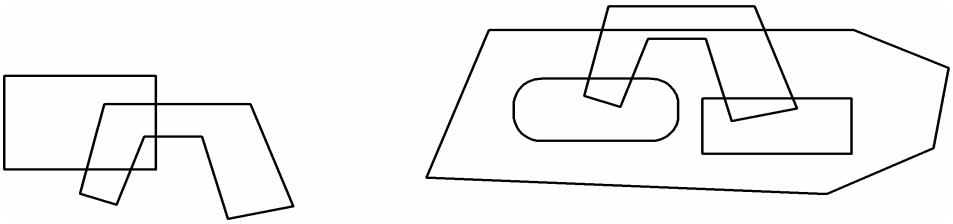


Figure 24 : Surfaces indépendantes (SURFACE).

Le type d'attribut géométrique SURFACE est à disposition (cf. Figure 24) pour des surfaces (générales) pouvant se chevaucher partiellement ou en totalité, c.-à-d. qui ne doivent pas avoir que des points de frontière en commun. Ce type est appelé surfaces indépendantes. Une surface indépendante possède une frontière extérieure et éventuellement plusieurs frontières intérieures (délimitant des enclaves). Chaque frontière comporte au moins une polyligne. Outre la géométrie, chaque polyligne présente également des attributs définis (cf. règle LinAttrDef).

2.8.12.3 Surfaces d'une partition de territoire

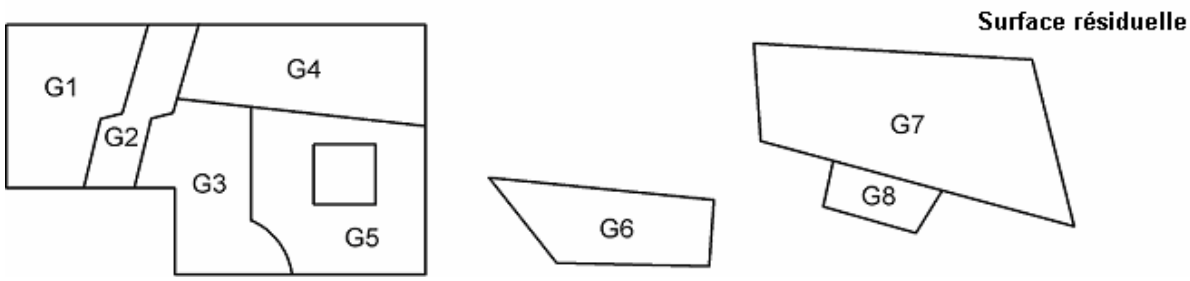


Figure 25 : Partition de territoire (AREA).

Partition du territoire (*réseau surfacique*) : c'est ainsi que l'on désigne un ensemble fini de surfaces (générales), une surface résiduelle qui recouvre le plan n'ayant que des points frontières en commun.

Le type d'attribut géométrique AREA est à disposition pour les surfaces (générales) d'une partition du territoire.

A chaque objet du territoire est affecté au plus une surface de la partition du territoire (ou exactement une, avec le mot-clé supplémentaire MANDATORY), mais pas la surface résiduelle. Ce n'est pas possible qu'aucun objet du territoire ne corresponde à chacune des deux surfaces de la partition du territoire ayant une limite commune.

Chaque objet du territoire particulier correspond de ce fait à une seule surface indépendante. Pour les objets du territoire, il en résulte donc la même structure implicite que pour les surfaces indépendantes. Les conditions de cohérence suivantes s'appliquent par ailleurs :

- Des polylignes d'une partition du territoire doivent toujours être de vraies frontières. Il ne peut donc pas exister de polyligne pour laquelle la même surface s'étende de part et d'autre de la ligne (cf. Figure 23). Une telle situation est également exclue de par la définition de la surface.
- Si des objets de territoire se trouvent des deux côtés d'une polyligne, chaque portion de courbe (connection de deux points d'appui) de l'un des objets du territoire doit correspondre, en géométrie et en attributs, à exactement une portion de courbe de l'autre objet du territoire.

Il ne peut pas y avoir de partitions du territoire dans des sous-structures.

L'instruction AREA INSPECTION est à disposition (cf. paragraphe 2.15 Vues) pour qu'il soit possible d'accéder aux polylignes d'une partition du territoire comme à des objets isolés (et des objets uniques, même dans le cas où la polyligne est frontière commune de deux surfaces voisines).

2.8.12.4 Possibilités d'extension

Des surfaces indépendantes peuvent être étendues à des partitions du territoire. L'extension d'une polyligne en une surface n'est pas autorisée puisqu'une surface doit prendre en compte plusieurs polylignes, alors que dans la définition d'une polyligne, une seule est attendue.

Des surfaces indépendantes et des surfaces d'une partition du territoire peuvent être étendues de deux manières :

- Si 'SURFACE' est défini en premier lieu, des recouvrements sont autorisés, et un remplacement par 'AREA' est permis dans des extensions car ainsi la définition de base est respectée.
- D'autres attributs de ligne peuvent être ajoutés.

2.9 Unités

Les unités sont toujours décrites par une désignation et un symbole entre crochets ([ ]). La désignation et le symbole doivent être des noms. En l'absence de toute indication de symbole, ce dernier équivaut à la désignation. Des indications supplémentaires peuvent suivre selon le type de l'unité, mais son utilisation s'effectue toujours par l'entremise de son symbole, la désignation jouant un rôle purement documentaire.

2.9.1 Unités de base

Les unités de base sont le mètre, la seconde, etc. Elles ne nécessitent pas d'autres indications. Les unités de base peuvent aussi être définies de manière abstraite (mot-clé ABSTRACT) si l'unité elle-même n'est pas encore connue alors que le contexte est clair (exemple : température, argent, etc.). Aucun symbole n'est assigné aux unités abstraites. Les unités concrètes ne peuvent plus être étendues.

Exemples :

```
UNIT
  Longueur (ABSTRACT);
  Temps (ABSTRACT);
  Monnaie (ABSTRACT);
  Temperature (ABSTRACT);
  Metre [m] EXTENDS Longueur;
  Seconde [s] EXTENDS Temps;
  FrancSuisse [CHF] EXTENDS Monnaie;
  Celsius [C] EXTENDS Temperature;
```

INTERLIS a défini une unité abstraite interne ANYUNIT. Toutes les autres unités en héritent directement ou indirectement d'elle (cf. paragraphe 2.10.2.1 Paramètres pour systèmes de référence et de coordonnées). Les unités suivantes sont déjà définies par INTERLIS :

```
UNIT
  ANYUNIT             (ABSTRACT);
  DIMENSIONLESS       (ABSTRACT);
  LENGTH              (ABSTRACT);
  MASS                (ABSTRACT);
  TIME                (ABSTRACT);
  ELECTRIC_CURRENT    (ABSTRACT);
  TEMPERATURE         (ABSTRACT);
  AMOUNT_OF_MATTER    (ABSTRACT);
  ANGLE               (ABSTRACT);
  SOLID_ANGLE         (ABSTRACT);
  LUMINOUS_INTENSITY (ABSTRACT);
  MONEY               (ABSTRACT);

  METER [m]           EXTENDS LENGTH;
```

```
KILOGRAMM [kg]     EXTENDS MASS;
SECOND [s]         EXTENDS TIME;
AMPERE [A]         EXTENDS ELECTRIC_CURRENT;
DEGREE_KELVIN [K]  EXTENDS TEMPERATURE;
MOLE [mol]         EXTENDS AMOUNT_OF_MATTER;
RADIAN [rad]       EXTENDS ANGLE;
STERADIAN [sr]     EXTENDS SOLID_ANGLE;
CANDELA [cd]       EXTENDS LUMINOUS_INTENSITY;
```

Remarque : l'annexe G "Définitions d'unités" énumère les unités les plus usuelles dans un modèle de type élargi.

2.9.2 Unités dérivées

Les unités dérivées se déduisent d'autres unités (multiplication ou division par une constante, fonction). Exemples :

```
UNIT
  Kilometre [km] = 1000 [m];
  Centimetre [cm] = 1 / 100 [m];
  Inch [in] = 0.0254 [m];         !! 1 inch = 2.54 cm
  Fahrenheit [oF] = FUNCTION // (oF + 459.67) / 1.8 // [K];
```

Des valeurs exprimées en kilomètres doivent être multipliées par mille pour obtenir des valeurs exprimées en mètres. De même, des valeurs en pouces doivent être multipliées par 2.54 pour obtenir la correspondance en mètres. On doit ajouter une constante de 459.67 à une température exprimée en degrés Fahrenheit, puis diviser cette valeur par 1.8 afin d'obtenir la même température exprimée en degrés Kelvin.

Une unité dérivée est automatiquement une extension de la même unité abstraite, comme celle dans laquelle elle peut être convertie.

2.9.3 Unités composées

Les unités composées (par exemple km par heure) s'obtiennent par la multiplication ou la division d'autres unités (unités de base, dérivées ou composées). On peut également définir des unités composées en tant qu'unités abstraites. Elles doivent alors tenir compte des autres unités abstraites.

Les unités utilisées dans l'extension concrète doivent par conséquent être des extensions concrètes des unités spécifiées dans la définition abstraite. Exemple :

```
UNIT
  Vitesse (ABSTRACT) = ( Longueur / Temps );
  KilometreHeure [kmph] EXTENDS Vitesse = ( km / h );
```

2.9.4 Unités structurées

Les unités structurées se composent de plusieurs autres unités, concrètes ou non, devant s'appuyer directement ou indirectement (unités dérivées) sur une unité de base commune (par exemple le temps). La plage de valeurs possibles doit toujours être spécifiée pour les sous-divisions de l'unité structurée (cf. exemple). Les minima et maxima sont toujours des nombres positifs. Pour la dernière sous-division, des décimales peuvent être définies, pour les autres non. Des indications cadrées ne sont pas autorisées.

Certaines unités structurées présentent des discontinuités : ainsi chaque mois n'a pas 31 jours. D'autres unités (comme le temps) sont définies en continu. Elles sont alors identifiées par le mot-clé CONTINOUS. Exemples :

```
UNIT
  Temps (ABSTRACT);
  Seconde [s] EXTENDS Temps;
  Minute = 60 [s];
  Heure = 60 [Minute];
```

```
Jour EXTENDS Temps;
Mois EXTENDS Temps;
Annee EXTENDS Temps;
Horloge = {Heure:Minute[0 .. 59]:s[0 .. 59]} CONTINUOUS;
Date = {Annee:Mois[1 .. 12]:Jour[1 .. 31]};
```

Syntaxe :

```
UnitDef = 'UNIT'
              { Unit-Name
                  [ '(' 'ABSTRACT' ')' | '[' UnitShort-Name ']' ]
                  [ 'EXTENDS' Abstract-UnitRef ]
                  [ '=' ( DerivedUnit | ComposedUnit | StructuredUnit )]
                  ';' }.

DerivedUnit = [ DecConst { ( '*' | '/' ) DecConst }
              | 'FUNCTION' Explanation ] '[' UnitRef ']'.

ComposedUnit = '(' UnitRef { ( '*' | '/' ) UnitRef } ')'.

StructuredUnit = '{' UnitRef
                     { ':' UnitRef '[' Min-Dec '..' Max-Dec ']' }
                 '}'
                     [ 'CONTINUOUS' ].

UnitRef = [ Model-Name '.' [ Topic-Name '.' ] ] UnitShort-Name.
```

Remarque : l'annexe H "Définitions temporelles" présente une proposition d'extension standard avec un exemple d'application pour des données temporelles suisses.

2.10 Traitement des méta-objets

2.10.1 Généralités

Les méta-objets, au sens d'INTERLIS 2, sont des objets importants dans le cadre de la description de modèles d'applications. Ils sont utilisés pour des systèmes de référence et pour des signatures graphiques. Les systèmes de référence ou le signatures graphiques sont référencés par leur nom dans les modèles d'applications.

La mise en place d'objets de systèmes de référence ou de signatures doit être définie dans un modèle (REFSYSTEM MODEL ou SYMBOLOGY MODEL) à l'aide des moyens usuels de classes et de structures. Les classes de système de référence, d'axes ou de signatures doivent dans ce cas constituer des extensions des classes COORDSYSTEM, REFSYSTEM, AXIS ou SIGN proposées par INTERLIS, lesquelles sont à leur tour des extensions de la classe METAOBJECT. Celle-ci présente un attribut de nom devant être univoque dans le cadre de l'ensemble des méta-objets d'un conteneur.

Comme tous les objets, les méta-objets font eux-mêmes partie intégrante de conteneurs. Un conteneur, et par suite ses méta-objets, est créé dans un modèle d'application par l'introduction d'un nom de conteneur et l'indication du thème supposé (règle MetaDataUseDef). Les objets de conteneurs de méta-données possèdent des noms et correspondent à des éléments de schéma tels que des attributs ou des classes. Un conteneur correspond à un thème et dispose ce faisant d'un espace nominal. Par conséquent, l'extension (spécialisation) d'un conteneur n'entraîne que l'extension (spécialisation) de son espace nominal. Dans ce contexte, le nom du conteneur peut également être défini comme une extension d'un nom déjà introduit (EXTENDS). Le thème associé doit donc être le même que pour le nom de base ou pour une extension de celui-ci.

Si la référence à un méta-objet (règle MetaObjectRef) est faite dans le cadre du nom de conteneur étendu, la recherche débute dans le conteneur associé. Si aucun méta-objet de ce nom n'y est trouvé, la

recherche se poursuit dans les conteneurs en fonction du nom de conteneur hérité directement ou indirectement. Des méta-objets peuvent ainsi être créés et affinés à différents niveaux. Il est par exemple possible de définir des signatures graphiques générales dans un premier temps puis de les affiner et de les compléter au niveau régional. Les modalités d'accès au conteneur concret sur la base du nom du conteneur sont du ressort de l'outil mis en œuvre à cet effet.

Dans un modèle d'application traduit (cf. paragraphe 2.5.1 Modèle), un conteneur concret ne contenant que des traductions (objets METAOBJECT_TRANSLATION ; cf. annexe A) peut également être affecté à un nom de conteneur. Ainsi, seuls sont traduits les méta-objets introduits dans le modèle de base par l'intermédiaire du conteneur correspondant. Si ce modèle de base est lui-même une traduction, un nom de conteneur peut, dans ce cas également, se voir affecter un conteneur concret ne contenant que des traductions. Si le modèle n'est pas une traduction, seul un conteneur concret ne contenant aucune traduction (c.-à-d. aucun objet de type METAOBJECT_TRANSLATION) peut être affecté à un nom de conteneur.

Syntaxe :

```
MetaDataUseDef = ('SIGN' | 'REFSYSTEM' ) 'BASKET' Basket-Name
                     Properties<FINAL>
                     [ 'EXTENDS' MetaDataUseRef ]
                         '~' Model-Name '.' Topic-Name ';'.

MetaDataUseRef = [ Model-Name '.' [ Topic-Name '.' ] ] Basket-Name.

MetaObjectRef = [ MetaDataUseRef '.' ] Metaobject-Name.
```

Si, dans le contexte courant, *un seul* nom de conteneur de métadonnées est requis, alors cette référence (MetaDataUseRef) ne doit pas être indiquée dans la référence des méta-objets.

2.10.2 Paramètres

Les propriétés qui ne concernent pas le méta-objet lui-même mais son utilisation dans l'application sont décrites à l'aide de paramètres dans le métamodèle. La définition des paramètres est introduite par le mot-clé PARAMETER. Sa structure est calquée sur celle des attributs.

2.10.2.1 Paramètres pour systèmes de référence et de coordonnées

Seul le paramètre prédéfini Unit est admis pour des systèmes de référence et de coordonnées ou leurs axes.

Si, dans la définition d'un type de données numériques (cf. paragraphe 2.8.5 Types de données numériques) ou d'un type de coordonnées (cf. paragraphe 2.8.7 Coordonnées), il est fait référence (règle RefSys) à un système de référence ou de coordonnées, alors il convient d'indiquer une unité compatible avec celle de l'axe correspondant du système de coordonnées ou l'axe unique du système de référence (cf. paragraphe 2.10.3 Systèmes de référence).

2.10.2.2 Paramètres de signatures

Les paramètres de signatures peuvent être définis librement. Ils correspondent aux indications (exemple : identification du symbole du point, position, rotation) qui doivent être transmises à un système graphique afin que celui-ci puisse engendrer sa représentation. La signature elle-même doit d'abord pouvoir être choisie. Cela passe par la définition d'un paramètre indiquant la référence vers la classe de signatures dans laquelle le paramètre est défini (METAOBJECT). On peut aussi indiquer, comme paramètre d'une signature, une référence vers un autre méta-objet (METAOBJECT OF). Dans les deux cas, on indique une référence méta-objet dans l'application (cf. paragraphe 2.16 Représentations graphiques), c'est à dire le nom de référence du conteneur et le nom du méta-objet. De cette façon, l'identification effective du conteneur et l'identification du méta-objet peuvent être déterminées par l'outil correspondant.

Hormis ces cas spécifiques, les paramètres peuvent être définis comme des attributs.

Syntaxe :

```
ParameterDef = Parameter-Name Properties<ABSTRACT,EXTENDED,FINAL>
                  ':' ( AttrTypeDef
                      | 'METAOBJECT' [ 'OF' MetaObject-ClassRef ] ) ';'.
```

2.10.3 Systèmes de référence

Sans indication supplémentaire, des valeurs numériques et des coordonnées sont considérées comme des indications différentielles, du fait de l'absence de toute définition de référence absolue. Un système de coordonnées ou un système scalaire doit être défini dans cette optique. La définition du modèle s'effectue alors dans le cadre d'un modèle de type REFSYSTEM MODEL, INTERLIS mettant les classes suivantes à disposition :

```
CLASS METAOBJECT (ABSTRACT) =
  Name: MANDATORY NAME;
UNIQUE Name;
END METAOBJECT;

STRUCTURE AXIS =
  PARAMETER
    Unit: NUMERIC [ ANYUNIT ];
END AXIS;

CLASS REFSYSTEM (ABSTRACT) EXTENDS METAOBJECT =
 END REFSYSTEM;

CLASS COORDSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  ATTRIBUTE
    Axis: LIST {1..*} OF AXIS;
END COORDSYSTEM;

CLASS SCALSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  PARAMETER
    Unit: NUMERIC [ ANYUNIT ];
END SCALSYSTEM;
```

Il est possible, dans des extensions, d'ajouter des attributs supplémentaires caractéristiques du type des systèmes de coordonnées et scalaires, l'unité pouvant par ailleurs être remplacée par une extension (syntaxe pour la définition des paramètres : cf. paragraphe 2.10.2 Paramètres et paragraphe 2.5.3 Structures et classes). L'unité définie au sein d'un domaine de valeurs doit alors être compatible avec cette dernière (cf. paragraphe 2.9 Unités).

2.11 Paramètres d'exécution

En plus des données elles-mêmes et des métadonnées, on peut également définir des éléments de données isolés qui seront mis à disposition, lors de l'exécution, par un système de traitement, d'exploitation ou de représentation. On les appelle des paramètres d'exécution.

Syntaxe :

```
RunTimeParameterDef = 'PARAMETER'
                        { RunTimeParameter-Name ':' AttrTypeDef';' }.
```

Les partitions de territoire (mot-clé AREA) ne sont pas admises comme paramètres d'exécution. Ceux-ci imposant des conditions aux systèmes dépassant le cadre du langage INTERLIS 2, ils ne peuvent être définis qu'au sein de modèles pour lesquels des contrats ont été définis.

L'échelle de la représentation et l'heure actuelle sont des exemples types de paramètres d'exécution.

2.12 Règles d'intégrité

Des règles d'intégrité (également appelées conditions de cohérence) servent à la définition des restrictions que les objets doivent respecter.

Les règles d'intégrité relatives à un objet isolé sont décrites par une expression logique se rapportant aux attributs de l'objet (cf. paragraphe suivant). Il est ainsi possible de définir des conditions de cohérence à respecter obligatoirement, s'appliquant de ce fait impérativement à tous les objets (mot-clé MANDATORY), et d'autres ne valant qu'au niveau de la règle. Dans ce dernier cas, le pourcentage d'instances de la classe devant normalement remplir cette condition sera indiqué (cf. règle PlausibilityConstraint).

La condition d'existence (Règle ExistenceConstraint) permet d'exiger que la valeur d'un attribut de chaque objet de la classe de condition apparaisse dans un attribut donné d'une instance d'autres classes, ce qui implique que l'attribut de condition soit compatible avec l'autre attribut et entraîne les conséquences suivantes :

- Si le domaine de valeurs d'un attribut de condition est identique au domaine de valeurs d'un autre attribut ou d'une extension de celui-ci, la valeur de la condition doit apparaître dans l'attribut requis d'une autre instance.
- Si le domaine de valeurs de l'attribut est une structure, tous les attributs qu'elle contient feront l'objet d'une comparaison.
- Si le domaine de valeurs de l'autre attribut est une coordonnée et le domaine de valeurs de l'attribut de condition une ligne, une surface simple ou une partition de territoire avec un domaine de coordonnées identique ou étendu, alors toutes les coordonnées des points d'appui de la ligne, de la surface simple ou de la partition du territoire devront apparaître dans l'attribut requis d'une autre instance.

Les exigences d'unicité sont introduites par l'intermédiaire du mot-clé UNIQUE (règle UniquenessConstraint).

Il peut être exigé, dans le cas d'une classe ou d'une association (appelée la classe actuelle dans la suite), qu'une combinaison donnée d'attributs ou de rôles soit univoque de manière globale. Tous les attributs et rôles accessibles via un chemin d'objet entrent en ligne de compte, donc tous ceux susceptibles d'être affectés sans équivoque à l'objet actuel.

Sur le plan conceptuel, "global" signifie que la condition concernée est à satisfaire par tous les objets existants de tous les conteneurs.

S'agissant de structures, de classes et d'associations, on peut exiger qu'une combinaison donnée d'attributs de sous-structures définis via BAG OF, LIST OF ou par l'intermédiaire des domaines de valeurs géométriques POLYLINE, SURFACE ou AREA, soit univoque localement (LOCAL), c.-à-d. dans le cadre de tous les éléments structurés affectés à l'objet ou à l'élément structuré actuel.

Exemple :

```
CLASS A =
  K: (A, B, C);
  ID: TEXT*10;
UNIQUE K, ID;
END A;
```

Si un attribut de valeur indéfinie intervient dans une condition d'unicité, la condition est réputée remplie.

Des règles d'intégrité plus approfondies sont à définir au moyen de vues (par exemple une vue reliant une classe donnée à elle-même et permettant ce faisant la comparaison de combinaisons quelconques d'attributs avec tous les autres objets de la classe). De telles vues doivent impérativement être définies au sein du modèle de données.

Des conditions de cohérence s'étendant à un grand nombre d'objets (en particulier des conditions d'unicité) ne sont pas toujours contrôlables en totalité car la vérification ne peut s'effectuer qu'avec les conteneurs disponibles localement. Sur le plan conceptuel, elles s'appliquent toutefois (hormis dans le cas de métamodèles) globalement (cf. aussi annexe F Unicité de clés utilisateurs).

Syntaxe :

```
ConstraintDef = ( MandatoryConstraint
                | PlausibilityConstraint
                | ExistenceConstraint
                | UniquenessConstraint ).

MandatoryConstraint = 'MANDATORY' 'CONSTRAINT'
                         Logical-Expression ';'.

PlausibilityConstraint = 'CONSTRAINT'
                            ( '<=' | '>=' ) Percentage-Dec '%'
                            Logical-Expression ';'.

ExistenceConstraint = 'EXISTENCE' 'CONSTRAINT'
                          AttributePath 'REQUIRED' 'IN'
                             ViewableRef ':' AttributePath
                             { 'OR' ViewableRef ':' AttributePath } ';'.

UniquenessConstraint =  'UNIQUE' ( GlobalUniqueness
                                 | LocalUniqueness ) ';'.

GlobalUniqueness = UniqueEl.

UniqueEl = ObjectOrAttributePath { ',' ObjectOrAttributePath }.

LocalUniqueness = '(' 'LOCAL' ')'
                     StructureAttribute-Name
                        { '->' StructureAttribute-Name } ':'
                                 Attribute-Name { ',' Attribute-Name }.
```

Des conditions de cohérence pour une classe ou une association donnée peuvent également n'être définies qu'a posteriori (d'ordinaire à l'issue de la définition d'une association).

Syntaxe :

```
ConstraintsDef = 'CONSTRAINTS' 'OF' ClassOrAssociationRef '='
                     { ConstraintDef }
                 'END' ';'.
```

2.13 Expressions

Des expressions sont utilisées de façon générale (Expression), par exemple en tant qu'arguments de fonctions et comme expressions logiques (Logical-Expression; le type de résultat devant être booléen) dans des règles d'intégrité et de sélection, par exemple. Elles se réfèrent à un objet contextuel, autrement dit l'objet pour lequel on formule la condition. Partant de cet objet, on peut par exemple faire référence à un attribut, un élément structurel, une fonction, etc. De tels objets ainsi que des paramètres de comparaison comme des constantes et des paramètres d'exécution entrent comme facteurs dans des prédicats. Un prédicat est une indication qui est soit vraie, soit fausse. Des prédicats peuvent être rattachés par des opérateurs booléens à une expression logique.

Syntaxe :

```
Expression = Term.
```

```
Term = Term1 { 'OR' Term1 }.

Term1 = Term2 { 'AND' Term2 }.

Term2 = Predicate [ Relation Predicate ].

Predicate = ( Factor
            | [ 'NOT' ] '(' Logical-Expression ')'
            | 'DEFINED' '(' Factor ')' ).

Relation = ( '==' | '!=' | '<>' | '<=' | '>=' | '<' | '>' ).
```

Remarques importantes :

- Conformément aux règles de syntaxe pour les termes, la comparaison (Relation) constitue le lien le plus intense, viennent ensuite AND puis OR.
- Avec NOT et l'expression logique qui suit entre parenthèses, on exige la négation de cette expression.
- Comparaison de facteurs. Suivant le type de facteurs, des comparaisons particulières sont exclues :
  - Pour des types de chaînes de caractères, l'égalité (==) et l'inégalité (!=, <>) ont la signification usuelle. Supérieur à (>) signifie que le premier facteur est le plus grand, en comparaison du second facteur, puis affiche d'autres caractères (le cas échéant [>=]). Inférieur à (<) signifie de façon analogue que le deuxième facteur est le plus grand, en comparaison du premier facteur, puis suivent d'autres caractères (le cas échéant [<=]).
  - Les comparaisons sont définies avec la signification usuelle pour les types de données numériques et les domaines de valeurs structurées. Les comparaisons de types 'supérieur à' et 'inférieur à' ne sont pas opportunes pour des types de données circulaires.
  - Les coordonnées ne peuvent être vérifiées comme un tout qu'en termes d'égalité et d'inégalité. Les autres comparaisons ne sont à disposition que pour les différentes composantes (cf. commentaire ci-après concernant la règle de syntaxe pour Factor).
  - Pour les énumérations, les comparaisons de types 'supérieur à' et 'inférieur à' ne sont admises que si l'énumération a été définie comme ordonnée. Deux valeurs d'énumération sont équivalentes si une valeur est une extension de l'autre. Ainsi, dans l'exemple du paragraphe 2.8.2 Enumérations, "jours_ouvrables" et "jours_ouvrables.mardi" sont identiques.
  - S'agissant des lignes, on ne peut vérifier que si celles-ci sont indéfinies (== UNDEFINED).
  - Un facteur peut aussi désigner un objet. On peut alors faire des vérifications sous l'angle de ce qui est défini, égal ou inégal.
- Si un facteur ne se compose pas uniquement de l'objet mais également du chemin conduisant jusqu'à lui, l'objet est présumé indéfini dès lors que l'un des attributs du chemin n'est pas défini. La fonction intégrée DEFINED (a.b) est par conséquent équivalente à (a.b != UNDEFINED).
- L'ordre dans lequel les prédicats (mais pas les règles de liaison !) reliés par AND ou OR sont exploités est laissé à la libre appréciation des systèmes.

Les facteurs peuvent être formés dans le respect des règles syntaxiques suivantes :

```
Factor = ( ObjectOrAttributePath
         | FunctionCall
         | 'PARAMETER' [ Model-Name '.' ] RunTimeParameter-Name
         | Constant ).

ObjectOrAttributePath = PathEl { '->' PathEl }.
```

```
AttributePath = ObjectOrAttributePath.

PathEl = ('THIS'
         | 'THISAREA' | 'THATAREA'
         | 'PARENT'
         | ReferenceAttribute-Name
         | AssociationPath
         | Role-Name
         | Base-Name
         | AttributeRef ).

AssociationPath = [ '\' ] AssociationAccess-Name.

AttributeRef = ( Attribute-Name ( [ '[' ( 'FIRST'
                                        | 'LAST'
                                        | AxisListIndex-PosNumber ) ']' ] )
               | 'AGGREGATES' ).

FunctionCall = [ Model-Name '.' ] Function-Name
                   '(' Argument {',' Argument } ')'.

Argument = ( Expression | ViewableRef).
```

Les facteurs peuvent se rapporter à des objets et à leurs attributs. Des chemins d'objets complets peuvent ainsi être formés pas à pas. Chacun des éléments complète à chaque fois le chemin de l'objet actuel vers le suivant. Le premier objet actuel se déduit du contexte, par exemple un objet de la classe pour laquelle une condition de cohérence va être définie.

- THIS désigne l'objet dit contextuel, c.-à-d. l'objet actuel d'une classe, d'une vue ou d'une définition graphique dans laquelle un chemin d'accès à un objet est requis. THIS est par exemple à indiquer comme argument lors de l'appel d'une fonction possédant ANYCLASS ou ANYSTRUCTURE comme paramètre.
- THISAREA et THATAREA désignent les deux objets territoriaux délimités par l'objet linéaire actuel. Ils sont du type Surface et l'emploi de THISAREA et THATAREA n'est possible que dans le cadre de l'inspection d'une partition de territoire (cf. paragraphe 2.15 Vues).
- PARENT désigne l'élément structuré de niveau supérieur ou le sur-objet de l'élément structuré ou de l'objet actuel. La vue doit de plus constituer une inspection ordinaire et non une inspection de territoire (cf. paragraphe 2.15 Vues).
- Indication de l'attribut de référence : elle décrit l'objet assigné à partir de l'objet actuel ou de la structure actuelle via l'attribut de référence donné.
- Chemin de relation : Définit l'objet (ou la donnée) associé ou l'objet associé de la classe d'association, dans lequel :
  - Si l'on veut atteindre par la relation un objet (donnée) lié, l'accès à la relation sera donné (voir chapitre 2.7.5, Accès à la relation). Ceci n'est autorisé que lorsque la lien est univoque, c'est à dire que, selon la cardinalité, il peut s'agir au plus d'un objet lié. En cas de relation double, la possibilité n'est donnée que si le rôle de l'accès à la relation demandé a une cardinalité maximum de un et, dans le cas de relations multiples, le rôle à toutes les classes de relation ont la cardinalité maximale de un.
  - Si l'on veut atteindre l'objet lié lui-même (soit l'objet de la classe d'association), on précédera l'accès à la relation d'un backslash (\). Ceci n'est autorisé que si la cardinalité n'autorise qu'un seul objet de liaison subordonné.

- Indication du rôle : elle décrit l'objet affecté via le rôle à partir de l'objet actuel de la classe d'association. Le chemin n'est admis que lorsque la cardinalité maximale du rôle est au plus égale à 1.
- Indication de la vue de base : le nom (local) de la vue de base permet de décrire l'objet (virtuel) correspondant de la vue de base dans la vue actuelle ou dans la relation actuelle déduite.

Dans la référence à un attribut, c'est la valeur de l'attribut de l'objet contextuel ou de l'objet décrit par le chemin d'accès qui intervient. Des chemins d'accès se terminant par un attribut sont par ailleurs désignés par l'expression de chemins d'attributs et sont également utilisés dans différentes règles syntaxiques, indépendamment de facteurs.

- L'indication du nom de l'attribut est suffisante dans le cas normal.
- Dans le cas en revanche d'un attribut de coordonnées, la composante concernée des coordonnées est décrite par l'indication du numéro de l'axe. La valeur 1 est associée à la première composante.
- L'attribut implicite AGGREGATES est défini dans des vues d'agrégation (cf. paragraphe 2.15 Vues) et désigne le jeu (BAG OF) des objets de base agrégés.

L'accès à des éléments isolés est possible dans le cas de sous-structures ordonnées (LIST). Les valeurs admises sont les suivantes :

- FIRST : le premier élément.
- LAST : le dernier élément.
- Numéro d'indice : l'indice indiqué doit être inférieur ou égal ou nombre maximal défini dans la cardinalité. L'indice 1 est affecté au premier élément. Il existe toujours un élément correspondant s'il est inférieur ou égal au nombre minimal défini dans la cardinalité ; l'existence de l'élément n'est pas garantie s'il est supérieur à cette valeur. Un facteur indéfini peut en résulter.

Des facteurs peuvent également être des appels de fonctions. Les arguments suivants sont possibles :

- Facteurs : le type du facteur doit être compatible avec celui de l'argument.
- Le nom d'une classe requise (règle ViewableRef) ou une fonction fournissant une classe pour résultat : le paramètre formel doit être du type CLASS.

Des appels de fonctions, des paramètres d'exécution (cf. paragraphe 2.16 Représentations graphiques) et des constantes entrent en ligne de compte en tant que valeurs de comparaison.

2.14 Fonctions

Une fonction est définie à l'aide de son nom, de paramètres formels et d'une courte description en guise d'explication. Les noms des paramètres n'ont qu'une valeur documentaire. La définition n'est admise que dans le cadre de contrats, sinon une exploitation automatique des modèles ne serait plus garantie.

Les possibilités suivantes entrent en ligne de compte pour les paramètres formels et le résultat de la fonction :

- Les types admis pour les attributs, en particulier les structures. Des facteurs appropriés (cf. règle Facteur) sont à utiliser comme arguments (c.-à-d. comme paramètres actuels).
- Si une structure est indiquée dans ce cadre, les arguments seront prioritairement des éléments structurés. Cependant, il est également possible d'indiquer des chemins d'objets conduisant à des objets constituant une extension de la structure. Des chemins d'objets quelconques peuvent en particulier être indiqués pour ANYSTRUCTURE.
- Si OBJECT OF est indiqué, les objets atteignables par l'intermédiaire d'un chemin d'objet et répondant à la définition constituent des arguments possibles. Des chemins d'objets quelconques peuvent en particulier être indiqués pour OBJECT OF ANYSTRUCTURE. De même manière que pour les références à d'autres objets (cf. paragraphe 2.6.3 Attributs de référence), la classe de

base admise et d'éventuelles restrictions peuvent être définies puis spécialisées dans des extensions. ANYSTRUCTURE peut à cette occasion être spécialisé en ANYCLASS, pour autant qu'aucune restriction (RESTRICTED TO) n'ait été définie.

Syntaxe :

```
FunctionDef = 'FUNCTION' Function-Name
                '(' Argument-Name ':' ArgumentType
               {';' Argument-Name ':' ArgumentType } ')'
               ':' ArgumentType [ Explanation ] ';'.
ArgumentType = ( AttrTypeDef
              | 'OBJECT' 'OF' (RestrictedClassOrAssRef
                              | RestrictedStructureRef
                              | ViewRef ) ).
```

Les fonctions standard suivantes sont définies :

```
FUNCTION myClass (Object: OBJECT OF ANYSTRUCTURE): STRUCTURE;
```

Elle fournit la classe de l'objet.

```
FUNCTION isSubClass (potSubClass: STRUCTURE; potSuperClass: STRUCTURE): BOOLEAN;
```

Son résultat est Vrai (true) si la classe du premier argument coïncide avec la classe ou une sous-classe du deuxième argument.

```
FUNCTION isOfClass (Object: OBJECT OF ANYSTRUCTURE; Class: STRUCTURE): BOOLEAN;
```

Son résultat est Vrai (true) si l'objet du premier argument appartient à la classe ou à une sous-classe du deuxième argument.

```
FUNCTION elementCount (bag: BAG OF ANYSTRUCTURE): NUMERIC;
```

Elle fournit le nombre d'éléments que renferme le Bag (ou la liste).

```
FUNCTION convertUnit (from: NUMERIC): NUMERIC;
```

Elle convertit la valeur numérique du paramètre "from" en valeur numérique de renvoi et tient compte des unités liées au paramètre et à l'utilisation de la valeur du résultat (d'ordinaire à l'attribut auquel le résultat est affecté). Cette fonction ne peut être utilisée que lorsque les arguments de "from" et du paramètre de renvoi sont compatibles, c.-à-d. lorsque leurs unités dérivent d'une unité commune.

2.15 Vues

Les vues (Views) sont des classes et des structures, dont les objets ne sont pas originaires : ils sont virtuels car déduits d'objets d'autres vues, classes ou encore structures. Les vues sont entre autres utilisées pour jeter les bases de graphiques et formuler des conditions de cohérence particulières. On les utilise aussi pour la transmission de données à des systèmes récepteurs sous une forme dérivée, en règle générale simplifiée.

Les vues ne sont transférées que si elles sont définies dans un thème de type VIEW TOPIC. Dans ce cas, leur transmission s'effectue comme le transfert complet (mot-clé FULL) de classes normales, de sorte qu'un récepteur de données (cf. chapitre 3 Transfert séquentiel) n'ait pas à se préoccuper de la manière dont les objets (virtuels) ont été générés. Des vues peuvent également être exclues explicitement du transfert (TRANSIENT) si elles n'ont qu'une signification locale, c.-à-d. si elles ne servent que de base pour d'autres vues. La livraison incrémentielle de vues est exclue car aucune identification d'objet n'est affectée aux objets-vues.

Les vues peuvent être abstraites (ABSTRACT) ou concrètes. Des vues concrètes peuvent également reposer sur des bases abstraites. Cependant, seuls les attributs de base concrets peuvent être appelés. Si tel n'est pas le cas, la vue elle-même doit être déclarée comme étant abstraite.

Les vues peuvent également être étendues (EXTENDED ou EXTENDS). Il n'est toutefois pas possible, dans ce cadre, de modifier la loi de formation. L'extension sert à tenir compte des extensions des vues, des classes et des structures sur lesquelles repose la vue d'une façon telle que la formulation de sélections, d'attributs et de conditions de cohérence supplémentaires soit possible.

Syntaxe :

```
ViewDef = 'VIEW' View-Name
            Properties<ABSTRACT,EXTENDED,FINAL,TRANSIENT>
            [ FormationDef | 'EXTENDS' ViewRef ]
               { BaseExtentionDef }
               { Selection }
            '='
            [ ViewAttributes ]
            { ConstraintDef }
          'END' View-Name ';'.

ViewRef = [ Model-Name'.' [ Topic-Name '.' ] ] ( View-Name ).
```

La règle de formation (FormationDef) d'une vue définit la manière dont les objets virtuels de la vue sont formés ainsi que les bases sur lesquelles s'appuie cette formation.

La vue projection (mot-clé PROJECTION OF) constitue la vue la plus simple. Elle permet de visualiser la classe de base (classe, structure ou vue) sous une forme modifiée (affichage d'une partie seulement des attributs et selon un ordre modifié, par exemple).

Dans le cas d'une *jonction* (mot-clé JOIN OF), on forme le produit cartésien (ou produit vectoriel) des classes de base (classe ou vue), c.-à-d. qu'il existe alors autant d'objets de la classe de jonction que de combinaisons d'objets des différentes classes de base. Il est également possible de définir des jonctions dites "Outer-Joins", liant des objets de la première classe de base à des objets vides (sans existence propre) des autres classes de base (instruction "(OR NULL)"). De tels objets vides sont ajoutés lorsque aucun objet de la classe supplémentaire recherchée n'a été trouvé pour une combinaison donnée des objets précédents. Tous les attributs de l'objet vide sont indéfinis. Les attributs de vues correspondants ne peuvent par conséquent pas être obligatoires.

L'*union* (mot-clé UNION OF) permet la fusion de différentes classes de base en une classe unique. Les attributs des différentes classes de base sont généralement affectés à un attribut de la vue union. Le type d'attribut de la classe de base doit être compatible avec celui de la vue union (même type ou extension de celui-ci).

Le regroupement ou agrégation (mot-clé AGGREGATION OF) permet de regrouper, en une même instance, toutes les instances d'un ensemble de base ou celles présentant une identité avec la combinaison d'attributs requise. Le jeu d'objets originaux est tenu à disposition au sein de la vue d'agrégation, sous forme de BAG, au moyen de l'attribut implicite AGGREGATES (cf. paragraphe 2.13 Expressions). Cet attribut implicite n'appartient pas aux attributs propres de la vue, raison pour laquelle il n'est pas transmis en cas de transfert. Il peut par exemple être affecté à un attribut approprié de la vue d'agrégation ou être échangé en tant qu'argument de fonction.

L'*inspection* (mot-clé INSPECTION OF) permet d'obtenir l'ensemble de tous les éléments structurés (définis via BAG OF, LIST OF ou dans le respect d'une ligne, d'une surface simple ou d'une partition d'un territoire) appartenant à un attribut de sous-structure d'une classe d'objets. L'inspection de partitions de territoire (mot-clé AREA INSPECTION) permet d'obtenir les lignes de la partition du territoire (sous forme

de structure SurfaceEdge) exactement une fois. L'accès aux territoires adjacents à cette ligne peut s'effectuer via THISAREA ou THATAREA (cf. paragraphe 2.13 Expressions). Le degré de finesse avec lequel les lignes sont subdivisées est laissé à l'appréciation de l'implémentation. Il est donc permis de signaler un objet linéaire pour un tronçon de jonction comprenant deux points d'appui et dépourvu de tout autre point d'appui intermédiaire ou de procéder à un regroupement maximum des tronçons de jonction (pour autant que les attributs soient identiques).

Une inspection normale portant sur un attribut de partition de territoire livre en revanche tous les éléments de la frontière (structure SurfaceBoundary), comme l'inspection d'un attribut de surface. Toutes les lignes sont également obtenues (structure SurfaceEdge) si une inspection complémentaire est effectuée sur l'attribut Lines. En procédant de la sorte, les lignes seront toutes détectées deux fois dans le cas d'une partition de territoire (une fois pour chaque territoire impliqué).

```
STRUCTURE SurfaceEdge =
  Geometry: DIRECTED POLYLINE;
  LineAttrs: ANYSTRUCTURE;
END SurfaceEdge;

STRUCTURE SurfaceBoundary =
  Lines: LIST OF SurfaceEdge;
END SurfaceBoundary;
```

L'inspection d'une ligne (POLYLINE) livre la structure suivante :

```
STRUCTURE LineGeometry =
  Segments: LIST OF LineSegment;
MANDATORY CONSTRAINT isOfClass (Segments[FIRST],INTERLIS.StartSegment);
END LineGeometry;
```

INTERLIS se limite à une description conceptuelle de la vue. Rien n'est entrepris pour apporter un soutien efficace à la mise en œuvre des vues. En conséquence, la génération de vues appartient à un niveau d'accomplissement spécifique du processus.

Syntaxe :

```
FormationDef = ( Projection | Join | Union | Aggregation | Inspection ).

Projection = 'PROJECTION' 'OF' RenamedViewableRef ';'.

Join = 'JOIN' 'OF' RenamedViewableRef
          (* ',' RenamedViewableRef
              [ '(' 'OR' 'NULL' ')' ] *) ';'.

Union = 'UNION' 'OF' RenamedViewableRef
           (* ',' RenamedViewableRef *) ';'.

Aggregation = 'AGGREGATION' 'OF' RenamedViewableRef
                 ( 'ALL' | 'EQUAL' '(' UniqueEl ')' ) ';'.

Inspection = [ 'AREA' ] 'INSPECTION' 'OF' RenamedViewableRef
                '->' StructureAttribute-Name
                   { '->' StructureAttribute-Name } ';'.
```

Toutes les vues de base utilisées dans une vue se voient attribuer un nom au sein de la vue utilisatrice, sous lequel elles peuvent être appelées. Ce nom correspond à celui de la vue de base, pour autant qu'aucun changement de désignation n'ait été entrepris au moyen d'un nom de base (Base-Name) explicite (local). Un tel changement de désignation est en particulier nécessaire en cas de définition de jonctions pour lesquelles la même classe de base est appelée à plusieurs reprises.

Syntaxe :

```
RenamedViewableRef = [ Base-Name '~' ] ViewableRef.

ViewableRef = [ Model-Name '.' [ Topic-Name '.' ] ]
              ( Structure-Name
              | Class-Name
              | Association-Name
              | View-Name).
```

Si l'on souhaite tenir compte d'extensions de classes de base dans une vue ou dans une extension de vue dans l'optique de la formulation d'attributs, de sélections ou de conditions de cohérence supplémentaires, il convient d'intégrer une définition d'extension appropriée (BaseExtentionDef). Elle s'appuie sur une vue de base déjà définie et décrit à son tour les extensions (qui doivent constituer des extensions de la vue de base considérée jusqu'alors) comme des vues de base. Si une telle extension de vue est utilisée au sein d'expressions, elle fournit la valeur "UNDEFINED" si l'objet de base associé à l'objet virtuel est incompatible avec cette extension de vue.

Syntaxe :

```
BaseExtentionDef = 'BASE' Base-Name 'EXTENDED' 'BY'
                       RenamedViewableRef { ',' RenamedViewableRef }.
```

L'ensemble des objets-vues défini par l'intermédiaire de la loi de formation peut être soumis à des restrictions supplémentaires par l'introduction de conditions (mot-clé WHERE).

Syntaxe :

```
Selection = 'WHERE' Logical-Expression ';'.
```

En ce qui concerne les attributs (et par suite la vue du récepteur) et les conditions de cohérence, l'organisation des vues s'appuie sur les mêmes principes que ceux prévalant pour les classes et les structures. La possibilité de prise en charge dans le même ordre (ALL OF) de tous les attributs d'une base de vues est en outre offerte pour simplifier les écritures. Cette possibilité ne s'étend pas aux unions pour lesquelles elle est dépourvue de sens et par conséquent non admise.

Syntaxe :

```
ViewAttributes = [ 'ATTRIBUTE' ]
                     { 'ALL' 'OF' Base-Name ';'
                     | AttributeDef
                     | Attribute-Name Properties <ABSTRACT,EXTENDED,FINAL>
                       ':=' Factor ';' }.
```

Dans les cas les plus simples, lorsqu'un attribut de vue de base est pris en charge, il suffit d'indiquer le nom de l'attribut et l'affectation de l'attribut de base. De telles définitions sont toujours finales et ne peuvent donc plus être étendues.

Dans le cas d'unions, pour chaque attribut, il faut indiquer complètement de quels attributs de la classe de base cet attribut est dérivé. Un attribut ne doit cependant pas se référer à toutes les classes de base, pour autant que le type d'attribut admette des valeurs indéfinies. Il est présumé indéfini pour les objets de base manquants.

L'exemple suivant présente la manière de décrire une vue permettant de définir une relation vraie (cf. paragraphe 2.7.1 Généralités du paragraphe 2.7 Relations vraies) à l'aide de l'élément DERIVED FROM.

```
DOMAIN
  CHSurface = ... ;

FUNCTION Intersect (Surface1: CHSurface;
                    Surface2: CHSurface): BOOLEAN;
```

```
CLASS A =
  a1: CHSurface;
END A;

CLASS B =
  b1: CHSurface;
END B;

VIEW ABIntersection
  JOIN OF A,B;
  WHERE Intersect (A.a1,B.b1);
  =
END ABIntersection;

ASSOCIATION IntersectedAB
  DERIVED FROM ABIntersection =
  ARole -- A := ABIntersection->A;
  BRole -- B := ABIntersection->B;
END IntersectedAB;
```

2.16 Représentations graphiques

Une représentation graphique se compose de définitions graphiques se basant toujours sur une vue ou une classe (mot-clé BASED ON). Au niveau conceptuel, une définition graphique a pour but d'affecter une signature graphique unique (symbole ponctuel, ligne, remplissage de surface, étiquette) à chaque objet de cette vue ou de cette classe (pour autant qu'il n'y ait pas eu de filtrage supplémentaire avec une sélection (mot-clé WHERE)) par l'intermédiaire d'une ou de plusieurs règles de dessin (cf. règle DrawingRule) et de générer ce faisant un ou plusieurs objets graphiques entrainant la représentation correspondante (voir Figure 5). Chaque règle de dessin doit à cet effet sélectionner une signature graphique (avec nom de méta-objet) et définir des arguments pour les paramètres associés.

Les propriétés d'héritage peuvent être définies entre parenthèses (règle Properties). Si une définition graphique est abstraite, aucun objet graphique n'en résulte. L'extension d'une définition graphique doit se baser sur la même classe que la définition graphique de base (absence de BASED ON) ou sur l'une de ses extensions.

La règle de dessin est identifiée par un nom afin qu'elle puisse être reprise et affinée dans des extensions (remarque : affinée au sens de la spécialisation mais également de valeurs de paramètres supplémentaires). S'il existe des extensions pour une règle de dessin (dans des représentations graphiques étendues), celles-ci ne génèrent pas de nouveaux objets graphiques mais exercent en revanche une influence sur les paramètres des signatures de l'objet graphique prescrit par la définition de base. Il est permis de définir plusieurs extensions pour une définition graphique, toutes exploitées (dans l'ordre de leur définition). Cette possibilité peut en particulier être utilisée pour prévoir différents secteurs d'extension pour des aspects différents (par exemple les différentes règles de dessin). Les différents paramètres de signatures sont ensuite déterminés. Cette définition peut s'effectuer en plusieurs étapes. Pour chaque paramètre, c'est la valeur définie en dernier lieu qui s'applique. La définition initiale est d'abord exploitée, puis suivent les éventuelles extensions. De plus, les affectations de paramètres peuvent encore être reliées à une condition (cf. règle CondSignParamAssignment), c.-à-d. que l'affectation n'est réalisée que lorsque la condition est remplie. Si la condition de sélection n'est pas remplie, les éventuelles sous-extensions ne seront pas non plus prises en compte.

Dès que des règles de dessin sont concrètes, la classe à laquelle appartiennent les signatures graphiques à affecter doit être définie. Dans des extensions de règles de dessin, cette classe de signatures graphiques peut être remplacée par une classe constituant une extension de la classe actuelle. La classe "responsable" de signatures graphiques est d'abord celle à laquelle appartient l'objet de signature graphique assigné (un méta-objet). Les valeurs concrètes sont à affecter aux paramètres

introduits dans la classe "responsable". Si les paramètres indiqués correspondent à une classe étendue de signatures graphiques, celle-ci devient la classe "responsable", pour autant que la classe de la signature graphique de la règle de dessin coïncide avec elle ou en soit une extension.

Dans les conditions mentionnées, les attributs d'objets (cf. AttributePath dans la règle SignParamAssignment) peuvent également être comparés à des paramètres d'exécution (cf. paragraphe 2.11 Paramètres d'exécution). Les paramètres d'exécution, importants pour la représentation graphique (par exemple l'échelle de la représentation graphique désirée), sont généralement définis dans des modèles de signatures puisqu'ils décrivent les capacités graphiques attendues d'un système, comme les paramètres des signatures. Une référence de méta-objet doit être indiquée pour un paramètre de signature graphique requérant un méta-objet (cf. paragraphe 2.10 Traitement des méta-objets).

La valeur d'un paramètre ordinaire d'une signature graphique est indiquée comme une constante ou sous forme de renvoi vers un attribut d'objet (cf. Factor dans la règle SignParamAssignment). Dans ce cadre, il est toujours renvoyé à l'attribut d'un objet de la classe de base ou de la vue de base, spécifié au moyen de BASED ON.

La représentation dépendant fréquemment d'attributs définis par l'intermédiaire d'énumérations, un élément spécifique est proposé ici : le domaine d'énumération. Il s'agit soit d'un nœud donné de l'arbre du type d'énumération, soit d'un intervalle de nœuds défini par deux nœuds de même niveau. Les définitions d'intervalles ne sont toutefois admises que s'il s'agit d'un type d'énumération ordonnée. Lorsque la valeur de l'attribut est comprise dans les limites de l'un des domaines d'énumération indiqués, la valeur correspondante est assignée au paramètre. Les signatures concrètes résultent du modèle de signatures, dans lequel les classes de signatures sont définies en compagnie des paramètres d'exécution (mot-clé PARAMETER) nécessaires pour leur utilisation. Une définition uniquement abstraite de types de données numériques est autorisée.

Syntaxe :

```
GraphicDef = 'GRAPHIC' Graphic-Name Properties<ABSTRACT,FINAL>
                 [ 'EXTENDS' GraphicRef ]
                 [ 'BASED' 'ON' ViewableRef] '='
                 { Selection }
                 { DrawingRule }
             'END' Graphic-Name ';'.

GraphicRef = [ Model-Name '.' [ Topic-Name '.' ] ] Graphic-Name.

DrawingRule = DrawingRule-Name Properties<ABSTRACT,EXTENDED,FINAL>
                    ['OF' Sign-ClassRef ]
                       ':' CondSignParamAssignment
                           { ',' CondSignParamAssignment }';'.

CondSignParamAssignment = ['WHERE' Logical-Expression ]
                '(' SignParamAssignment { ';' SignParamAssignment } ')'.

SignParamAssignment = SignParameter-Name
                         ':=' ( '{' MetaObjectRef '}'
                              | Factor
                              | 'ACCORDING' Enum-AttributePath
                                 '(' EnumAssignment {',' EnumAssignment} ')'
                              ).

EnumAssignment = ('{'MetaObjectRef '}'| Constant ) 'WHEN' 'IN' EnumRange.

EnumRange = EnumerationConst [ '..' EnumerationConst ].
```

La classe SIGN est prédéfinie par INTERLIS pour une utilisation dans des modèles de signatures :

```
CLASS SIGN (ABSTRACT) EXTENDS METAOBJECT =
  PARAMETER
    Sign: METAOBJECT;
END SIGN;
```

Cette classe de base doit être étendue pour des classes de signatures concrètes. Les données concrètes et les paramètres sont alors définis dans ce cadre.

L'exemple suivant présente la manière dont les symboles graphiques adéquats (symboles de points et étiquettes) sont définis à partir d'une classe de points comportant des coordonnées, une chaîne de caractères et une énumération sous forme d'attributs.

Le modèle de signatures est défini ainsi :

```
SYMBOLOGY MODEL SimpleSignsSymbology (en) =

  CONTRACT ISSUED BY Unknown;

  DOMAIN
    S_COORD2 (ABSTRACT) = COORD NUMERIC, NUMERIC;


  TOPIC SignsTopic =

    CLASS Symbol EXTENDS INTERLIS.SIGN =
      PARAMETER
        Pos: MANDATORY S_COORD2;
    END Symbol;

    CLASS Textlabel EXTENDS INTERLIS.SIGN =
      PARAMETER
        Pos: MANDATORY S_COORD2;
        Text: MANDATORY TEXT;
    END Textlabel;

  END SignsTopic;

END SimpleSignsSymbology.
```

Des objets (de signatures) concrets ont été saisis pour ce modèle de signatures et stockés sous le nom de bibliothèque de signatures (ou nom de conteneur) SimpleSignsBasket. Les objets de signatures saisis (classe Symbol) sont appelés Signature_point, Carre et Cercle et les types d'écritures (classe Textlabel) Ecriture1 et Ecriture2.

```
MODEL Donnees (fr) =

  DOMAIN
    CoordP = COORD
      0.000 .. 200.000 [m],
      0.000 .. 200.000 [m],
      ROTATION 2 -> 1;


  TOPIC Points =

    DOMAIN
      Genre_point = (Borne
                       (grande,
                        petite),
                     Cheville,
                     Tuyau,
                     Croix,
                     non_materialise) ORDERED;

    CLASS Point =
      Planimetrie: CoordP;      !! CoordP est un domaine de valeur de coordonnees
```

```
      Genre: Genre_point;
      Nom_point: TEXT*12;
    END Point;

  END Points;

END Donnees.


MODEL Simple_Graphique (fr) =

  CONTRACT ISSUED BY Unknown;

  IMPORTS Donnees;
  IMPORTS SimpleSignsSymbology;

  SIGN BASKET SimpleSignsBasket ~ SimpleSignsSymbology.SignsTopic;


  TOPIC Point_graphique =
    DEPENDS ON Donnees.Points;

    GRAPHIC Simple_Point_Graphique BASED ON Donnees.Points.Point =
      Symbol OF SimpleSignsSymbology.SignsTopic.Symbol: (
        Sign := {Signature_point};
        Pos := Planimetrie
      );

    END Simple_Point_Graphique;

  END Point_graphique;

END Simple_Graphique.
```

Sur ce graphique (basé sur le modèle de signatures SimpleSignsSymbology et sur la représentation graphique Simple_Graphique), des signatures de points simples (points ou dots) seront dessinées pour tous les points de la classe Point.

Il est également possible d'envisager qu'un utilisateur puisse souhaiter obtenir une représentation graphique plus évoluée, l'amélioration pouvant alors prendre différentes formes, telles que :

- L'existence de signatures supplémentaires (des signatures de points de types croix ou triangle). Une bibliothèque de signatures supplémentaires est requise à cet effet, intitulée SimpleSignsPlusBasket. Comme il s'agit d'une extension de la bibliothèque SimpleSignsBasket, les objets de signatures (ou les méta-objets) seront recherchés dans les deux bibliothèques. Si l'on étendait directement la bibliothèque SimpleSignsBasket (EXTENDED), les signatures de tous les symboles graphiques générés dans le modèle Graphique_Plus (y compris ceux hérités du modèle Simple_Graphique) seraient d'abord recherchées dans la bibliothèque étendue puis dans la bibliothèque de base (SimpleSignsBasket).
- Des signatures modulables en taille, afin qu'il soit par exemple possible de générer de petits et de grands carrés à l'aide de la même signature de point. Un modèle de signatures étendu est nécessaire à cette fin, dans lequel l'échelle de modulation de la taille des signatures est définie comme paramètre. Les classes de signatures ne présentant aucun attribut supplémentaire, l'existence de bibliothèques correspondantes n'est pas indispensable.
- Des signatures de points différentes sont à dessiner en fonction du genre de point : de petits ou de grands carrés pour les bornes, des cercles pour les chevilles, des croix pour les tuyaux et les croix. La signature de point effective peut directement être déduite du genre du point. Le facteur de modulation en taille des petits carrés pour la représentation de petites bornes est défini au moyen d'une affectation supplémentaire. Les points non matérialisés restent représentés à l'aide de

signatures de points simples, raison pour laquelle aucune nouvelle affectation n'est nécessaire dans ce cas.

```
SYMBOLOGY MODEL ScalableSignsSymbology (fr) =

  CONTRACT ISSUED BY Unknown;

  IMPORTS SimpleSignsSymbology;


  TOPIC ScalableSignsTopic EXTENDS SimpleSignsSymbology.SignsTopic =

    CLASS Symbol (EXTENDED) =
      PARAMETER
        ScaleFactor: 0.1 .. 10.0;  !! Default 1.0
    END Symbol;

  END ScalableSignsTopic;

END ScalableSignsSymbology.


MODEL Grafique_Plus (fr) =

  CONTRACT ISSUED BY Unknown;

  IMPORTS Simple_Graphique;
  IMPORTS SimpleSignsSymbology;
  IMPORTS ScalableSignsSymbology;

  SIGN BASKET SimpleSignsPlusBasket EXTENDS
    Simple_Graphique.SimpleSignsBasket ~ ScalableSignsSymbology.ScalableSignsTopic;


  TOPIC Point_Graphique_Plus_Top EXTENDS Simple_Graphique.Point_graphique =

    GRAPHIC Point_Graphique_Plus EXTENDS Simple_Point_Graphique =

      Symbol (EXTENDED) OF ScalableSignsSymbology.ScalableSignsTopic.Symbol: (
        Sign := ACCORDING Genre_point (
          {Carre} WHEN IN #Borne,
          {Cercle} WHEN IN #Cheville,
          {Croix} WHEN IN #Tuyau .. #Croix
        )
      ),
      WHERE Genre_point == #Borne.petite (
          ScaleFactor := 0.5
      );

      Text OF SimpleSignsSymbology.SignsTopic.Textlabel: (
        Sign := {Ecriture1};
        Pos := Planimetrie;
        Text := Nom_point
      );

    END Point_Graphique_Plus;

  END Point_Graphique_Plus_Top;

END Grafique_Plus.
```

3 Transfert séquentiel

3.1 Introduction

Ce chapitre présente le service de transfert séquentiel d'INTERLIS qui permet l'échange de données entre différents systèmes sans être lié à aucun système. Ce service permet l'échange tant complet qu'incrémentiel (ou différentiel) de données (réplication) et peut être appliqué à tout modèle INTERLIS. Ainsi, il est par exemple possible de transférer des données (modèle de données) et des objets de signatures (modèle de signatures) à l'aide du même mécanisme.

Pour l'heure, le service de transfert d'INTERLIS est défini comme un échange de fichiers XML (www.w3.org/XML). Il est également possible, entre autres éventualités, de générer des diagrammes XML (www.w3.org/XML/Schema) pour étendre le champ d'utilisation de ces fichiers INTERLIS/XML. Il est cependant envisageable que des services de transfert INTERLIS complémentaires soient définis à l'avenir (sur la base par exemple de services Internet ou CORBA), raison pour laquelle la description du service de transfert d'INTERLIS est subdivisée en deux sous-paragraphes, *Règles générales* et *Codage XML*. Les règles générales valent pour *tout* service de transfert *séquentiel* d'INTERLIS, indépendamment du codage concret ou de la transmission. Les règles définies dans le *Codage XML* s'appliquent au cas particulier des fichiers de transfert INTERLIS au format XML.

3.2 Règles générales pour le transfert séquentiel

3.2.1 Déductibilité à partir du modèle de données

Tout transfert INTERLIS peut être déduit du modèle de données INTERLIS associé par l'application de certaines règles (transfert de données basé sur un modèle).

3.2.2 Lecture de modèles étendus

Un transfert INTERLIS est toujours conçu de telle façon qu'un logiciel de lecture programmé ou configuré pour un certain modèle de données puisse également lire des données d'extensions de ce modèle sans disposer de connaissances relatives aux définitions étendues de ce modèle. Les données d'extensions sont alors ignorées par le logiciel de lecture sans création de messages d'erreurs.

3.2.3 Organisation d'un transfert : en-tête

Un transfert INTERLIS est un flux d'objets séquentiels, subdivisé en deux parties que sont l'en-tête et la section de données.

L'en-tête contient au moins les indications suivantes concernant le transfert :

- Le numéro de version actuel d'INTERLIS (cf. paragraphe 2.3 Règle principale).
- Le renvoi au(x) modèle(s) de données associé(s).
- La désignation de l'expéditeur (SENDER).

L'en-tête peut également contenir des commentaires en option.

L'organisation de la section de données est décrite plus en détail dans les paragraphes suivants.

3.2.4 Objets transférables

Des objets (c.-à-d. des instances objets) de classes concrètes, de relations, de vues et de définitions graphiques peuvent être transférés dans la section de données. Lors du transfert, les objets de vues sont

traités comme des objets de classes concrètes. La livraison incrémentielle de vues n'est pour l'heure pas possible. Les objets de vues ne sont transférés que si les vues associées ont été déclarées au sein d'un thème de type VIEW TOPIC ; ils ne sont pas transmis dans le cas contraire. Les vues ne sont par ailleurs pas transférables si elles ont été signalées par TRANSIENT.

3.2.5 Ordre des objets au sein de la section de données

La section de données se compose d'une suite de conteneurs (instances de thèmes) qui ne peuvent être transférés qu'en totalité. Seuls les objets modifiés ou supprimés sont transmis dans le cas d'une livraison incrémentielle. Toutefois, d'un point de vue conceptuel, c'est la totalité du conteneur qui est transmise accompagnée de l'historique, même dans le cas d'une livraison incrémentielle. Il est théoriquement possible qu'un transfert intègre des conteneurs issus de différents modèles. Un conteneur regroupe alors à son tour la totalité de ses objets. L'ordre des objets est sans importance pour le transfert, il n'est en particulier pas nécessaire que les objets au sein d'un conteneur soient ordonnés par relations ou groupés par classes (au contraire d'INTERLIS 1). Des conteneurs vides ne doivent pas être transférés.

3.2.6 Codage des objets

Tout conteneur et toute instance du flux d'objets se voit attribuer une identification. L'identification du conteneur doit être un identificateur d'objet (OID) général et stable. L'identification d'objet doit en outre être dénuée de toute ambiguïté durant la totalité du transfert. L'identification du conteneur dans lequel l'objet a été généré (conteneur initial) est par ailleurs fournie pour chaque instance objet. En cas de livraison incrémentielle ou lors du transfert initial, l'identification d'une instance objet doit être un identificateur d'objet général et stable (cf. annexe E Création d'identificateurs d'objets (OID)).

Tous les attributs d'objets (y compris COORD, SURFACE, AREA, POLYLINE, STRUCTURE, BAG OF, LIST OF, etc.) sont stockés à proximité de l'objet. Les attributs du type AREA sont codés comme ceux du type SURFACE. Les attributs du type BAG sont codés comme ceux du type LIST. STRUCTURE est codé comme LIST {1}.

La transmission de valeurs d'attributs ne s'appuie en standard que sur les caractères imprimables du jeu de caractères ASCII US (32 à 126) et sur les caractères figurant dans la table de l'annexe B.

3.2.7 Genres de transfert

Les indications suivantes doivent être livrées sur chaque conteneur :

- Indications sur le genre (KIND) de transfert : FULL, INITIAL ou UPDATE.
- Indication sur l'état initial (STARTSTATE) ou l'état final (ENDSTATE) de la mise à jour incrémentielle (seulement pour le type de transfert INITIAL ou UPDATE).

La présence de conteneurs de genres de transfert différents est admise au sein du même transfert (FULL, INTIAL et/ou UPDATE) Les genres de transfert ont l'importance suivante :

- FULL – Transfert intégral. A la réception d'un conteneur FULL, le destinataire doit d'abord initialiser un nouveau conteneur puis faire entrer tous les objets dans le conteneur au moyen d'INSERT. FULL n'est pas approprié comme base de mise à jour vu que les identificateurs d'objet ne sont valables que pour ce transfert. Des fichiers de transfert conformes à INTERLIS 1 correspondent à FULL. Seule l'opération INSERT peut apparaître dans le genre de transfert FULL.
- INITIAL – Première livraison. Elle correspond au mode de transfert FULL à cette exception près que le conteneur et les objets contenus doivent posséder des OID généraux et stables. Dans ce mode de transfert particulier, seule peut également intervenir l'opération INSERT.
- UPDATE – Mise à jour. Un conteneur UPDATE contient des objets avec des opérations INSERT, UPDATE ou DELETE. Tous les objets et conteneurs possèdent des OID généraux et stables. Des

conteneurs UPDATE ne doivent être traités par le système initial que si l'état originel (STARTSTATE) du conteneur a déjà été reçu avec l'opération INITIAL ou UPDATE.

Par ailleurs les règles de transfert supplémentaires suivantes s'appliquent au genre UPDATE :

- Le système récepteur peut partir de l'idée qu'après le traitement complet de toutes les données d'un conteneur UPDATE, un état cohérent prévaut à nouveau, autrement dit qu'un conteneur UPDATE transmet un conteneur d'un état de cohérence STARTSTATE dans un autre état de cohérence ENDSTATE.
- Un conteneur UPDATE n'est pas cohérent en soi car des relations ne peuvent en général être dissoutes qu'avec leurs antécédents.

Il faut par ailleurs indiquer, pour chaque objet, l'opération de mise à jour (cf. paragraphe 1.4.5 Conteneurs, réplication et transfert de données). Les opérations INSERT, UPDATE et DELETE ont la signification suivante :

- Pour l'opération INSERT : "introduire un nouvel objet" (insert object).
- Pour l'opération UPDATE : "mettre à jour les valeurs d'attribut d'objet" (update object). Il faut fournir tous les attributs (pas uniquement ceux qui sont modifiés).
- Pour l'opération DELETE : "supprimer l'objet" (delete object). Il faut fournir tous les attributs (pas uniquement l'OID).

3.3 Codage XML

3.3.1 Introduction

Contrairement aux règles du paragraphe 3.2 "Règles générales pour le transfert séquentiel", les règles qui s'appliquent au codage XML ne valent que pour des fichiers de transfert formatés au standard XML-1.0 (voir aussi www.w3.org/xml). Pour la formalisation des règles de dérivation du format de transfert, on recourt à une notation EBNF déjà introduite au paragraphe 2.1 "Syntaxe utilisée". Les règles suivantes sont déjà prédéfinies dans ce contexte :

```
XML-Text = Texte libre quelconque.
XML-String = Texte quelconque à une seule ligne.
XML-Value = '"' XML-String '"'.
XML-ID = '"' x { Letter | Digit} '"'.
```

Selon les règles XML-ID, 'x' ne fait pas partie de la valeur de ID, mais du codage.

Pour améliorer la lisibilité des différentes règles de déduction, on fait par ailleurs appel aux macros TAG et ETAG :

```
TAG ( Tagwert )
```

Engendre une règle partielle EBNF ayant la forme suivante :

```
'<%Tagwert%>'

TAG ( Tagwert, Attribut1, Attribut2,...)
```

Engendre une règle partielle EBNF de la forme suivante :

```
'<%Tagwert%' %Attribut1% %Attribut2% etc. '>'

ETAG ( Tagwert )
```

Engendre une règle partielle EBNF ayant la forme suivante :

```
'</%Tagwert%>'
```

La succession %Argument% doit à chaque fois être remplacée par la teneur actuelle de l'argument.

Exemples :

```
TAG (DATASECTION)                   engendre '<DATASECTION>'
TAG (HEADERSECTION,'VERSION="2.2"') engendre '<HEADERSECTION' 'VERSION="2.2"' '>'
ETAG (DATASECTION)                  engendre '</DATASECTION>
```

3.3.2 Codage de caractères

Seuls les caractères ASCII 32 à 126 ou les caractères de l'annexe B sont à disposition pour le codage de textes (XML-Text) ou de chaînes XML (XML-String). Les caractères sont codés dans le respect de la règle de codage UTF-8 ou sous forme de XML Character Reference voire XML Entity Reference. Les caractères spéciaux XML '&', '<' et '>' doivent par ailleurs être codés comme suit :

- '&' doit être remplacé par la suite de caractères '&'
- '<' doit être remplacé par la suite de caractères '<'
- '>' doit être remplacé par la suite de caractères '>'

Un résumé complet du codage des caractères avec toutes les formes possibles de codage pour chaque caractère figure en annexe B. Un logiciel d'écriture en INTERLIS 2 peut sélectionner librement une forme adéquate de codage s'il en existe plusieurs pour un caractère donné. Un logiciel de lecture en INTERLIS 2 doit pouvoir reconnaître *toutes* les formes de codage. Remarque : plusieurs formes de codage sont autorisées par caractère afin d'obtenir une compatibilité maximale avec les outils XML existants.

3.3.3 Structure générale du fichier de transfert

Un fichier de transfert INTERLIS est structuré dans le respect de la règle principale EBNF suivante :

```
Transfer = '<?xml version="1.0" encoding="UTF-8" ?>'
           TAG ( TRANSFER, 'xmlns="http://www.interlis.ch/INTERLIS2.2"' )
               HeaderSection
               DataSection
           ETAG ( TRANSFER ).
```

La règle HeaderSection crée l'en-tête du fichier de transfert et la règle DataSection le domaine de valeurs (ou section de données).

Un fichier de transfert INTERLIS créé par la règle de transfert est toujours un fichier de transfert valable (well formed) XML 1.0. De ce fait, un fichier de transfert INTERLIS peut aussi contenir un nombre quelconque de lignes de commentaires présentant la forme

```
<!-- Comment -->
```

aux endroits prévus pour cela dans XML 1.0. Le contenu des lignes de commentaires ne peut cependant pas être interprété par le logiciel de transfert. Pour le codage des signes du fichier de transfert, on fait appel au codage UTF-8. Seule peut toutefois être utilisé de façon standard la réserve de signes figurant à l'annexe B Table de caractères.

Les données sont transmises sous forme d'objets XML. Les noms de macros TAG des objets XML sont déduits des noms d'objets concernés dans le modèle de données INTERLIS. Pour des modèles de données traduits (TRANSLATION OF), cela signifie que les noms de macros TAG existent dans la langue traduite lors du transfert (des inscriptions supplémentaires doivent toutefois être effectuées dans la table d'alias).

3.3.4 En-tête

L'en-tête est structuré de la manière suivante :

```
HeaderSection = TAG ( HEADERSECTION,
```

```
                    'VERSION="2.2"',
                    'SENDER=' XML-Value )
                  Alias
                  [ Comment ]
                ETAG ( HEADERSECTION ).

Alias = TAG ( ALIAS )
           { Entries }
        ETAG ( ALIAS ).

Entries = TAG ( ENTRIES,
                'FOR=' XML-Value)
             { Tagentry | Valentry | Delentry }
          ETAG ( ENTRIES ).

Tagentry = TAG ( TAGENTRY,
                 'FROM=' XML-Value, 'TO=' XML-Value )
           ETAG ( TAGENTRY ).

Valentry = TAG ( VALENTRY,
                 'ATTR=' XML-Value, 'FROM=' XML-Value, 'TO=' XML-Value )
           ETAG ( VALENTRY ).

Delentry = TAG ( DELENTRY,
                 'TAG=' XML-Value )
           ETAG ( DELENTRY ).

Comment = TAG ( COMMENT )
             XML-Text
          ETAG( COMMENT ).
```

Il faut faire entrer les valeurs suivantes (attributs XML) dans l'élément HeaderSection :

- VERSION. Version du codage INTERLIS (pour le moment 2.2).
- SENDER. Expéditeur du lot de données.

Des entrées sont effectuées dans l'élément Alias, permettant la lecture polymorphe d'un lot de données (cf. paragraphe ci-dessous). Un commentaire peut être inséré (si on le souhaite) dans Comment. Il peut décrire le transfert de façon plus détaillée.

3.3.4.1 Importance et contenu de la table d'alias

L'élément Alias de HeaderSection est une table spéciale qui permet à un programme ad hoc de lire des modèles élargis sans connaître les extensions (lecture dite *polymorphe*). Comme XML ne connaît pas d'héritage et, partant, pas de polymorphisme non plus, la table d'alias est utilisée pour transmettre les informations complémentaires nécessaires à un programme de lecture.

La table d'alias doit comprendre une table de représentation (Entries-Element) pour chacun des modèles de données X intervenant dans le transfert. Dans chacune de ces tables de représentation, les entrées xxxENTRY (TAGENTRY, VALENTRY, DELENTRY) permettent d'indiquer les objets transférables (c.-à-d. ceux du modèle ou des extensions de ceux-ci) pouvant apparaître dans les conteneurs du modèle de données X (ou ne pas apparaître dans le cas d'inscriptions DELENTRY). Si des traductions (TRANSLATION OF) du modèle de données X sont également présentes lors du transfert, toutes les macros TAG du modèle de données traduit doivent en plus apparaître dans la table de représentation du modèle de données X (entrées TAGENTRY ou VALENTRY). Les tables de représentation des modèles de données doivent être ordonnées dans la table d'alias de telle manière que les tables de représentation des modèles de base précèdent les tables de représentation des modèles étendus ou traduits. Les différentes entrées de la table d'alias ont les significations suivantes :

- TAGENTRY. Le TAG est entré dans l'attribut FROM dans le respect du modèle original (par exemple Canton.TCanton.K1') et dans le respect du modèle actuellement considéré dans l'attribut TO (par exemple ' Confederation.TConfederation.B1'). Tous les TAG doivent également être entrés, conformément au modèle actuel. Dans ce cas, la même valeur est entrée dans les TAG FROM et TO. L'entrée TAGENTRY doit être indiquée pour des thèmes concrets, des classes, des structures, des relations, des définitions graphiques ou des vues transférables.
- VALENTRY. Le nom de l'attribut de la liste de choix est indiqué dans l'attribut ATTR (par exemple Canton. TCanton.K1.Couleur'). La valeur est entrée dans l'attribut FROM dans le respect du modèle original (par exemple 'rouge.carmin') et dans le respect du modèle actuel dans l'attribut TO (par exemple 'rouge'). Toutes les valeurs doivent également être entrées dans le respect du modèle actuel. Dans ce cas, la même valeur est entrée dans les TAG FROM et TO. L'entrée VALENTRY doit être indiquée pour tous les types d'énumération.
- DELENTRY. L'attribut TAG indique un TAG absent du modèle actuel mais éventuellement présent dans des extensions. L'entrée DELENTRY doit être indiquée pour des thèmes concrets, des classes, des structures, des relations et des définitions graphiques ou des vues transférables ainsi que pour des attributs de classes, de structures, relations et de vues transférables. Si une classe complète (ou une structure, relation ou encore une vue transférable) ne peut pas apparaître du point de vue du modèle actuel, il est permis de n'affecter DELENTRY qu'à la classe absente. Dans ce cas, les attributs de la classe (ou de la structure, relation ou encore de la vue transférable) n'ont pas à être livrés avec DELENTRY.
- Remarque relative à des relations sans identité propre : les règles mentionnées ci-dessus sont aussi valables pour des relations sans identité propre. Par exemple, les valeurs d'un attribut de dénombrement doivent être reportées dans la classe qui contient également le rôle (c'est à dire sous classe.rôle.attribut).
- Les propriétés de la table d'alias sont une nouvelle fois mises en lumière et commentées dans l'exemple suivant :

La description suivante des données :

```
INTERLIS 2.2;

MODEL Confederation =

  CLASS B (ABSTRACT) =
  END B;


  TOPIC TConfederation =

    CLASS B1 EXTENDS B =
      Couleur : (rouge, vert, bleu);
    END B1;

  END TConfederation;

END Confederation.


MODEL TConfederation TRANSLATION OF Confederation =

  CLASS TB (ABSTRACT) =
  END TB;


  TOPIC TTConfederation =

    CLASS TB1 EXTENDS TB =
      TCouleur : (trouge, tvert, tbleu);
    END TB1;
```

```
  END TTConfederation;

END TConfederation.


MODEL Canton =

  IMPORTS Confederation;


  TOPIC TCanton EXTENDS Confederation.TConfederation =

    CLASS K (ABSTRACT) EXTENDS Confederation.B =
    END K;

    CLASS K1 EXTENDS Confederation.TConfederation.B1 =
      Couleur (EXTENDED): (rouge (sombre, carmin, clair));
      Txt: TEXT*40;
    END K1;

    CLASS K2 =
      Txt: TEXT*40;
    END K2;

  END TCanton;

END Canton.


MODEL Commune =

  IMPORTS Canton;


  TOPIC TCommune EXTENDS Canton.TCanton =
  END TCommune;

END Commune.
```

implique par exemple les tables d'alias suivantes :

```
<ALIAS>

  <ENTRIES FOR="Confederation">

    <!— Inscription selon propre modele -->
    <TAGENTRY FROM="Confederation.TConfederation"
              TO="Confederation.TConfederation"></TAGENTRY>
    <TAGENTRY FROM="Confederation.TConfederation.B1"
              TO="Confederation.TConfederation.B1"></TAGENTRY>
    <VALENTRY ATTR="Confederation.TConfederation.B1.Couleur"
              FROM="rouge" TO="rouge"></TAGENTRY>
    <VALENTRY ATTR="Confederation.TConfederation.B1.Couleur"
              FROM="vert" TO="vert"></TAGENTRY>
    <VALENTRY ATTR="Confederation.TConfederation.B1.Couleur"
              FROM="bleu" TO="bleu"></TAGENTRY>

    <!-- Inscription pour TConfederation (TRANSLATION OF) -->
    <TAGENTRY FROM="TConfederation.TTConfederation"
              TO="Confederation.TConfederation"></TAGENTRY>
    <TAGENTRY FROM="TConfederation.TTConfederation.TB1"
              TO="Confederation.TConfederation.B1"></TAGENTRY>
    <VALENTRY ATTR="TConfederation.TTConfederation.TB1.TCouleur"
              FROM="trouge" TO="rouge"></TAGENTRY>
    <VALENTRY ATTR="TConfederation.TTConfederation.TB1.TCouleur"
              FROM="tvert" TO="vert"></TAGENTRY>
    <VALENTRY ATTR="TConfederation.TTConfederation.TB1.TCouleur"
```

```
                FROM="tbleu" TO="bleu"></TAGENTRY>

    <!-- Inscription selon modele Canton -->
    <TAGENTRY FROM="Canton.TCanton"
                TO="Confederation.TConfederation"></TAGENTRY>
    <TAGENTRY FROM="Canton.TCanton.K1"
                TO="Confederation.TConfederation.B1"></TAGENTRY>
    <DELENTRY TAG="Canton.TCanton.K1.Txt"></DELENTRY>
    <DELENTRY TAG="Canton.TCanton.K2"></DELENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.sombre" TO="rouge"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.carmin" TO="rouge"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.clair" TO="rouge"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur" FROM="vert" TO="vert"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur" FROM="bleu" TO="bleu"></VALENTRY>

    <!-- Inscription selon modele Commune -->
    <TAGENTRY FROM="Commune.TCommune"
                TO="Confederation.TConfederation"></TAGENTRY>
    <TAGENTRY FROM="Commune.TCommune.K1"
                TO="Confederation.TConfederation.B1"></TAGENTRY>
    <DELENTRY TAG="Commune.TCommune.K1.Txt"></DELENTRY>
    <DELENTRY TAG="Commune.TCommune.K2"></DELENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
              FROM="rouge.sombre" TO="rouge"> </VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.carmin" TO="rouge"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.clair" TO="rouge"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="vert" TO="vert"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="bleu" TO="bleu"></VALENTRY>

</ENTRIES>

  <ENTRIES FOR="Canton">

    <!-- Inscription selon propre modele -->
    <TAGENTRY FROM="Canton.TCanton" TO="Canton.TCanton"></TAGENTRY>
    <TAGENTRY FROM="Canton.TCanton.K1" TO="Canton.TCanton.K1"></TAGENTRY>
    <TAGENTRY FROM="Canton.TCanton.K2" TO="Canton.TCanton.K2"></TAGENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.sombre" TO="rouge.sombre"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.carmin" TO="rouge.carmin"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="rouge.clair" TO="rouge.clair"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="vert" TO="vert"></VALENTRY>
    <VALENTRY ATTR="Canton.TCanton.K1.Couleur"
                FROM="bleu" TO="bleu"></VALENTRY>

    <!-- Inscription selon modele Commune -->
    <TAGENTRY FROM="Commune.TCommune" TO="Canton.TCanton"></TAGENTRY>
    <TAGENTRY FROM="Commune.TCommune.K1" TO="Canton.TCanton.K1"></TAGENTRY>
    <TAGENTRY FROM="Commune.TCommune.K2" TO="Canton.TCanton.K2"></TAGENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.sombre" TO="rouge.sombre"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.carmin" TO="rouge.carmin"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.clair" TO="rouge.clair"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="vert" TO="vert"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
```

```
                FROM="bleu" TO="bleu"></VALENTRY>

  </ENTRIES>

  <ENTRIES FOR="Commune">

    <!-- Inscription propre modele -->
    <TAGENTRY FROM="Commune.TCommune" TO="Commune.TCommune"></TAGENTRY>
    <TAGENTRY FROM="Commune.TCommune.K1" TO="Commune.TCommune.K1"></TAGENTRY>
    <TAGENTRY FROM="Commune.TCommune.K2" TO="Commune.TCommune.K2"></TAGENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.sombre" TO="rouge.sombre"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.carmin" TO="rouge.carmin"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="rouge.clair" TO="rouge.clair"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="vert" TO="vert"></VALENTRY>
    <VALENTRY ATTR="Commune.TCommune.K1.Couleur"
                FROM="bleu" TO="bleu"></VALENTRY>

  </ENTRIES>

</ALIAS>
```

Un programme de lecture qui a été écrit ou configuré pour le modèle fédéral peut désormais lire des données Confederation, TConfederation, Canton ou Commune de la manière suivante :

- Tous les Tags de Confederation sont associés à eux-mêmes (par exemple Confederation.TConfederation à Confederation.TConfederation).
- Tous les Tags de TConfederation (TRANSLATION OF) sont associés à leur pendant dans Confederation (par exemple TConfederation.TTConfederation à Confederation.TConfederation).
- Un objet XML <Canton.TCanton.K1> a pour image <Confederation.TConfederation.B1> via l'entrée Tagentry. L'objet <Confederation.TConfederation.B1> est connu du logiciel de lecture du modèle fédéral de sorte qu'il peut interpréter l'objet de manière appropriée.
- Un objet XML <Commune.TCommune.K1> a pour image <Confederation.TConfederation.B1> via l'entrée Tagentry. L'objet <Confederation.TConfederation.B1> est connu du logiciel de lecture du modèle fédéral de sorte qu'il peut interpréter l'objet de manière appropriée.
- La valeur "rouge.carmin" de l'attribut d'énumération Canton.TCanton.K1.Couleur ou Commune.TCommune.Couleur a pour image "rouge" via l'entrée Valentry. La valeur "rouge" est valable selon le modèle fédéral.
- La classe abstraite Canton.TCanton.K ou Commune.TCommune.K n'a pas à être entrée comme Tagentry car aucune instance Canton.TCanton.K ou Commune.TCommune.K ne peut être rencontrée dans le jeu de données.
- L'attribut <Canton.TCanton.K1.Txt> ou <Commune.TCommune.K1.Txt> doit être ignoré car il n'existe pas dans le modèle fédéral.
- La classe <Canton.TCanton.K2> ou <Commune.TCommune.K2> doit être ignorée car des instances de <Canton.TCanton.K2> ou <Commune.TCommune.K2> n'existent pas du point de vue du modèle fédéral. Remarque : l'attribut <Canton.TCanton.K2.Txt> ou <Commune.TCommune.K2.Txt> n'a pas à être spécifiquement entré avec Delentry, puisque c'est l'ensemble de la classe qui n'existe pas du point de vue du modèle fédéral.

Remarques :

- La table d'alias peut être générée avec le compilateur INTERLIS 2.

- Pour chaque modèle de données compris dans le lot de données (y compris tous les modèles de base), il faut inscrire un Entry-Element. Le nom du modèle doit être introduit dans l'attribut XML FOR.
- Les noms de Tag, qui d'après la représentation via la table d'alias, ne conduisent à aucun nom de Tag connu du point de vue du modèle à lire, doivent être déclarés comme erreurs par le programme de lecture.
- Les valeurs d'attribut, qui d'après la représentation via la table d'alias, ne conduisent à aucune valeur connue du point de vue du modèle à lire, doivent être déclarés comme erreurs par le programme de lecture.

3.3.5 Section de données

La section de données se structure de la manière suivante :

```
DataSection = TAG ( DATASECTION )
                { Basket }
              ETAG ( DATASECTION ).
```

3.3.6 Codage de thèmes

Les conteneurs sont des instances d'un TOPIC ou VIEW TOPIC concret. Les conteneurs sont codés comme suit:

```
Basket = TAG (%Model.Topic%,
                'BID=' XML-ID,
                [ 'TOPICS=' XML-Value ],
                [ 'KIND=' XML-Value ],
                [ 'STARTSTATE=' XML-Value ],
                [ 'ENDSTATE=' XML-Value ] )
              { Object }
         ETAG ( %Model.Topic% ).
```

La valeur %Model.Topic% doit être remplacée pour chaque Topic concret correspondant (par ex. catalogue_de_donnees.points_fixes). Les attributs XML du conteneur ont la signification suivante :

- BID. L'identification du conteneur doit figurer dans le BID. Il doit s'agir, pour la livraison incrémentielle, d'un OID.
- TOPICS. Tous les thèmes (Topics), exceptés les topic de base, apparaissant effectivement dans le conteneur sont répertoriés dans TOPICS sous forme de liste dont les éléments sont séparés par des virgules (exemple : "Canton1.Topic1,Canton2.Topic2"). Les Topics attribués doivent être des extensions du topic de base commun %Model.Topic% (hérité à la rigueur en plusieurs niveaux)
- KIND. Genre de transfert (valeurs possibles : FULL, UPDATE, INITIAL). FULL est la valeur par défaut en l'absence de cet attribut.
- STARTSTATE. Etat initial du conteneur, avant le transfert (uniquement dans le cas de livraisons incrémentielles).
- ENDSTATE. Etat final du conteneur, au terme du transfert (uniquement dans le cas de livraisons incrémentielles).

3.3.7 Codage de classes

Les instances objets d'une classe concrète sont codées de la manière suivante :

```
Object = TAG ( %Model.Topic.Class%,
                'TID=' XML-ID,
                [ 'BID=' XML-ID ],
                [ 'OPERATION= 'XML-Value ] )
              (* Attribute | Role | RoleStruct | ReferenceAttribute *)
         ETAG ( %Model.Topic.Class% ).
```

La valeur %Model.Topic.Class% doit être substituée de façon appropriée pour chaque classe concrète (exemple : Modelededonnes.Pointsfixes.PFP). Chaque instance de classe – et de la sorte chaque instance objet – reçoit implicitement une identification de transfert (attribut XML TID) en plus des attributs définis dans le modèle. Avec le type de transfert 'FULL', tous les TID, inclus tous les BID, doivent être univoques dans le transfert actuel. Dans le type de transfert INITIAL ou UPDATE, les TID et BID doivent être des OID. L'identification du conteneur dans lequel l'objet a été créé à l'origine (conteneur originel) est signalée dans BID. Si l'objet se trouve dans son conteneur d'origine, il est possible d'omettre BID. Chaque objet se voit par ailleurs assigner un attribut pour l'opération de mise à jour (attribut XML OPERATION), dans les modes de transfert INITIAL et UPDATE. Cet attribut XML peut admettre les valeurs INSERT, UPDATE ou DELETE. Sans indication d'OPERATION, on admet la valeur INSERT.

Des paramètres ne sont pas transmis, hormis le cas mentionné au paragraphe 3.3.11 Codage de définitions graphiques. La succession des différents attributs dans le transfert est identique à l'ordre de la définition des attributs dans la classe concernée. Si des classes sont des extensions de classes de base, les attributs des classes élargies suivent les attributs des classes de base dans le transfert.

3.3.8 Codage de vues et de projections de vues

Vous voudrez bien vous reporter au paragraphe 3.2.4 Objets transférables pour le codage de vues. Les attributs XML TID et BID sont transmis, mais pas OPERATION. Seuls les attributs indiqués de manière explicite dans la vue par ATTRIBUTE ou de façon implicite par ALL OF sont transmis en tant qu'attributs de l'objet vue.

3.3.9 Codage de relations pourvues d'une identité propre

Les instances objets de relations concrètes ayant une identité propre (cf. paragraphe 2.7.1Généralités) sont transmises comme des instances objets de classes. Remarque : Pour des relations sans nom explicite, le nom (de la classe) se formé par chacun des noms de rôles liés (par ex. %RoleName1RoleName2%).

Les rôles sont traités comme des attributs. Les rôles eux-mêmes sont codés comme suit :

```
Role = TAG ( %RoleName%,
         ('REF=' XML-ID | 'EXTREF=' XML-ID 'BID=' XML-ID),
         [ 'NEXT_TID=' XML-ID ] )
       ETAG ( %RoleName% ).
```

Si la référence pointe vers un objet du même conteneur, elle est codée par REF. L'identification de transfert de l'objet référencé est alors inscrite dans REF.

Si la référence pointe vers un objet d'un autre conteneur (dans le même transfert ou en dehors de celui-ci), elle est codée par EXTREF et BID. L'identification de transfert de l'objet référencé est alors inscrite dans EXTREF et l'identification de son conteneur dans BID.

Dans des relations ordonnées, l'attribut NEXT_TID renvoie à l'objet relationnel suivant.

3.3.10 Codage de relations dépourvues d'identité propre

Les instances objets de (deux) relations concrètes dépourvues d'identité propre (cf. paragraphe 2.7.1Généralités) sont codées comme des sous-structures de l'un des deux objets associés. La sous-structure est organisée comme suit :

```
RoleStruct = TAG ( %RoleName%,
               ('REF=' XML-ID | 'EXTREF=' XML-ID 'BID=' XML-ID),
                [ 'NEXT_TID=' XML-ID ])
               [ StructureValue ]
             ETAG ( %RoleName% ).
```

Le nom du rôle renvoyant à l'objet principal (l'autre rôle n'est pas codé) doit être indiqué pour %RoleName%. D'éventuels attributs de la relation sont codés dans StructureValue. Les attributs XML REF, EXTREF, NEXT_TID et BID ont la même signification que dans le cas de relations ayant une identité propre. Lors d'une relation 1-1, le deuxième rôle doit être pris pour %RoleName%.

3.3.11 Codage de définitions graphiques

Lors du transfert, les classes de signatures (Sign-ClassRef) référencées par la définition graphique sont transmises pour chaque définition graphique. Les instances objets des classes de signatures sont générées par l'exécution des définitions graphiques sur un lot de données d'entrée concrètes. A cette occasion, les paramètres sont codés comme des attributs.

3.3.12 Codage d'attributs

3.3.12.1 Règles générales

Chaque attribut d'une instance objet (y compris des attributs complexes comme POINT, POLYLINE, SURFACE, AREA, STRUCTURE, LIST OF, BAG OF, etc.) est codé de la façon suivante :

```
Attribute = [ TAG ( %AttributeName% )
                AttributeValue
              ETAG ( %AttributeName% ) ].

AttibuteValue = ( TextValue | EnumValue | NumericValue | StructDecValue |
                  BasketValue | ClassTypeValue | StructureValue | BagValue |
                  ListValue | CoordValue | PolylineValue | SurfaceValue).
```

Lorsqu'une valeur d'attribut est indéfinie, l'attribut n'est pas transmis. L'unité de mesure de la valeur d'attribut n'est pas codée. Exemple d'un attribut simple :

```
<Nombre>12345</Nombre>
```

3.3.12.2 Codage d'une chaîne de signes, URI et NAME

Des attributs du type de base TEXT*N sont codés de la manière suivante :

```
TextValue = XML-String.
```

3.3.12.3 Codage d'énumérations

Les énumérations sont codées de la manière suivante :

```
EnumValue = ( EnumElement-Name { '.' EnumElement-Name } ) | 'OTHERS'.
```

Pour le codage d'énumérations, on utilise la syntaxe de constantes d'énumération (règle EnumValue). Le signe # est omis. Les types d'orientation de texte prédéfinis HALIGNMENT et VALIGNMENT sont codés comme des énumérations. Le type BOOLEAN est de même transmis comme une énumération.

3.3.12.4 Codage de types de données numériques

Les données numériques sont codées comme suit :

```
NumericValue = NumericConst.
```

Remarque : les zéros ne sont pas autorisés en tête pour des nombres entiers (007 est transféré comme 7). Dans le cas de nombres réels, un seul zéro est autorisé en tête (exemple : 00.07 n'est pas permis mais 0.07 l'est). Les nombres flottants peuvent être reportés dans différentes représentations (avec ou sans mantisse). Le mieux est que le domaine de valeur du nombre flottant corresponde simplement à votre déclaration. Ainsi, par exemple, 100 peut être reporté comme 10.0e1 ou comme 1.0e2.

3.3.12.5 Codage de domaines de valeurs structurés

Les domaines de valeurs structurés sont codés comme les constantes structurées :

```
StructDecValue = StructDec.
```

3.3.12.6 Codage de BASKET

Les valeurs d'attribut de type BASKET sont codées de la manière suivante :

```
BasketValue = TAG ( BASKETVALUE,
                    'TOPIC=' XML-Value,
                    'KIND=' XML-Value,
                    'BID=' XML-ID )
              ETAG ( BASKETVALUE ).
```

Les différents attributs XML ont la signification suivante :

- TOPIC. Désignation du thème sous la forme Model.Topic.
- KIND. Genre du conteneur (valeurs possibles : DATA, VIEW, BASE et GRAPHIC).
- BID. Identification du conteneur.

3.3.12.7 Codage de CLASS

Les attributs du type CLASS sont codés comme suit :

```
ClassTypeValue = XML-String.
```

La chaîne (string) XML se voit attribuer le nom de classe qualifié complet (par exemple JeuDesDonneesDeBase.Pointsfixes.LFP).

3.3.12.8 Codage de STRUCTURE

Les valeurs d'attributs du type STRUCTURE sont codées comme suit :

```
StructureValue = TAG ( %StructureName% )
                   (* Attribute *)
                 ETAG ( %StructureName% ).
```

StructureName est formé en tant que Model.StructureName pour des structures au niveau du modèle et en tant que Model.Topic.StructureName pour des structures au niveau du thème.

3.3.12.9 Codage de BAG OF et LIST OF

Des valeurs d'attribut du type BAG OF ou LIST OF sont codées comme suit :

```
BagValue = (* StructureValue *).
ListValue = (* StructureValue *).
```

La succession des éléments de la ListValue ne peut pas être modifiée lors du transfert.

3.3.12.10 Codage de coordonnées

Des valeurs d'attribut de type COORD sont codées comme suit :

```
CoordValue = TAG ( COORD ),
               TAG ( C1 ) NumericConst ETAG ( C1 )
               [ TAG ( C2 ) NumericConst ETAG ( C2 )
                 [ TAG ( C3 ) NumericConst ETAG ( C3 ) ]
               ]
             ETAG ( COORD ).
```

Les divers sous-objets XML doivent être constitués comme suit :

- C1. Première composante de coordonnée (codée comme une valeur numérique).
- C2. Deuxième composante de coordonnée (codée comme une valeur numérique, uniquement pour des coordonnées 2D et 3D).
- C3. Troisième composante de coordonnée (codée comme une valeur numérique, uniquement pour des coordonnées 3D).

3.3.12.11 Codage de POLYLINE

Des valeurs d'attribut de type POLYLINE sont codées de la manière suivante :

```
PolylineValue = TAG ( POLYLINE )
                  [ LineAttr ]
                  SegmentSequence
                ETAG ( POLYLINE ).

StartSegment = CoordValue.

LineSegment = CoordValue.

ArcSegment = TAG ( ARC )
               TAG ( C1 ) NumericConst ETAG ( C1 )
               TAG ( C2 ) NumericConst ETAG ( C2 )
               [ TAG ( C3 ) NumericConst ETAG ( C3 ) ]
               TAG ( A1 ) NumericConst ETAG ( A1 )
               TAG ( A2 ) NumericConst ETAG ( A2 )
               [ TAG ( R ) NumericConst ETAG ( R ) ]
             ETAG ( ARC ).

LineFormSegment = StructureValue.

SegmentSequence = StartSegment { LineSegment
                                 | ArcSegment
                                 | LineFormSegment}.

LineAttr = TAG ( LINEATTR )
             StructureValue
           ETAG ( LINEATTR ).
```

Les segments de droite d'une polyligne sont codés dans le respect de la règle LineSegment alors que la règle ArcSegment s'applique au cas des arcs de cercles. Des segments de droite définis avec LINE FORM sont codés comme une structure (LineStructure).

Remarques : le rayon (attribut XML optionnel R) est transmis en plus des coordonnées de point intermédiaire (A1/A2) dans le cas des arcs de cercles (règle ArcSegment), introduisant de ce fait une redondance. Le point intermédiaire d'un arc de cercle n'a d'importance qu'en planimétrie. Son altitude doit être interpolée linéairement entre celles du point initial et du point final de l'arc de cercle. Si ce dernier est défini dans le sens horaire (du point initial vers le point final), le rayon est précédé d'un signe "+" ; il est précédé d'un signe "-" dans le cas d'une définition dans le sens anti-horaire. En cas de différences entre le rayon et les valeurs des coordonnées, c'est le rayon qui prévaut (cf. paragraphe 2.8.11.2 Polyligne comportant des segments de droite et des arcs de cercle en tant qu'éléments de portion de courbe). L'altitude des points d'appui (C3) n'est à transmettre que dans le cas de polylignes en 3D.

3.3.12.12 Codage de SURFACE et AREA

SURFACE et AREA sont codés de la manière suivante :

```
SurfaceValue = TAG ( SURFACE )
                 OuterBoundary
```

```
                { InnerBoundary }
              ETAG ( SURFACE ).

OuterBoundary = Boundary.

InnerBoundary = Boundary.

Boundary = TAG ( BOUNDARY )
             (* PolylineValue *)
           ETAG ( BOUNDARY ).
```

Les surfaces sont transmises comme une suite de frontières (Boundaries). Une frontière est une suite de lignes de frontière, une ligne de frontière commençant toujours par le point final de la ligne de frontière la précédant. Le point final de la dernière ligne de frontière est identique au point initial de la première ligne de frontière. Ensemble, les lignes de frontière composent donc une polyligne fermée (polygone).

La première frontière d'une surface (OuterBoundary) est sa frontière extérieure. Les frontières intérieures de la surface qui suivent éventuellement (InnerBoundary) délimitent les îlots de la surface. D'un point de vue géométrique, les frontières internes doivent se situer entièrement dans la frontière externe. Les différentes frontières d'une surface ne doivent pas se recouper les unes les autres.

Si la surface (de type SURFACE ou AREA) est définie avec des attributs de ligne, l'élément structuré de l'attribut de ligne doit être transmis avec chaque PolylineValue (règle LineAttr dans PolylineValue).

Pour une partition du territoire (AREA), toutes les lignes de frontière de la surface doivent coïncider avec les lignes de frontière de la surface voisine, pour autant qu'elles n'appartiennent pas au périmètre du réseau surfacique. Deux lignes de frontière sont identiques lorsque, tronçon par tronçon, les points d'appui d'une ligne de frontière sont tous identiques à leurs homologues de la surface voisine. Pour les points d'appui des arcs de cercle, seul le signe précédant le rayon de l'arc de cercle peut être différent. Si des attributs de ligne ont été définis pour le réseau surfacique, les valeurs d'attributs de ligne doivent systématiquement être identiques par paire pour les lignes de frontière.

3.3.12.13 Codage de références

Des attributs de type REFERENCE TO sont codés de la manière suivante :

```
ReferenceAttribute = [ TAG ( %AttributeName%,
                         ('REF=' XML-ID |
                          'EXTREF=' XML-ID 'BID=' XML-ID))
                       ETAG ( %AttributeName% ) ].
```

Les attributs XML REF, EXTREF et BID ont la même signification que dans le cas de relations vraies.

3.3.12.14 Codage de METAOBJECT et METAOBJECT OF

Des attributs du type METAOBJECT (cf. la classe appropriée à l'annexe A Le modèle de données interne d'INTERLIS) sont codés selon les indications du paragraphe 3.3.12.9 Codage de BAG OF et LIST OF. Des paramètres du type METAOBJECT (règle de syntaxe ParameterDef) ne sont toutefois *pas* transmis. Des paramètres du type METAOBJECT OF sont transmis comme des attributs du type NAME.

3.4 Utilisation d'outils XML

Comme le transfert INTERLIS 2 se fonde totalement sur XML 1.0, il est en principe possible d'utiliser des outils INTERLIS ou XML pour le traitement (ou l'analyse) d'objets INTERLIS. Il faut toutefois veiller aux différences suivantes :

- Les outils INTERLIS 2 connaissent des modèles de données INTERLIS associés, en plus du lot de données XML. Ainsi, un outil de contrôle INTERLIS peut, par exemple, exécuter des tests plus pointus sur les données qu'un outil XML pur ne le pourrait.
- Les outils INTERLIS 2 connaissent les types de données spécifiques à INTERLIS comme COORD, POLYLINE, SURFACE etc. Par conséquent, un navigateur INTERLIS 2 pourra aussi représenter des lots de données d'un point de vue graphique. Un navigateur XML pur ne peut en revanche visualiser que la construction du document.
- Les outils INTERLIS 2 assistent la lecture polymorphe de données via la table d'alias. De cette manière, un programme d'importation INTERLIS peut lire des données d'un modèle élargi comme des données du modèle de base. Avec un outil XML pur, la lecture polymorphe n'est pas possible en l'état.
- Les outils INTERLIS 2 peuvent par ailleurs accepter la traduction de noms de schémas via la table d'alias. Cette possibilité n'est pas non plus disponible actuellement avec des outils XML purs.

Malgré ces différences, des outils généraux XML peuvent être utilisés directement à des fins multiples. A titre d'exemple, on peut citer le filtrage de lots de données, l'édition des lots de données avec des éditeurs XML, l'analyse de lots de données de transfert, la traduction dans d'autres formats, etc.

Annexe A (Norme) Le modèle de données interne d'INTERLIS

L'ensemble du modèle de données interne INTERLIS 2 est présenté ci-après. Ce document peut être réalisé par simple exécution du compilateur INTERLIS 2. Si les différents éléments du modèle sont déjà connus du compilateur INTERLIS 2, il n'est possible de le compiler, mais il sert uniquement à l'illustration. Les éléments du modèle sont déjà connus du compilateur INTERLIS 2 fourni par COGIS.

```
!! File INTERLIS2.2.ili 2003-03-18

INTERLIS 2.2;

TYPE MODEL INTERLIS (en) =

  LINE FORM
    STRAIGHTS;
    ARCS;

  UNIT
    ANYUNIT (ABSTRACT);
    DIMENSIONLESS (ABSTRACT);
    LENGTH (ABSTRACT);
    MASS (ABSTRACT);
    TIME (ABSTRACT);
    ELECTRIC_CURRENT (ABSTRACT);
    TEMPERATURE (ABSTRACT);
    AMOUNT_OF_MATTER (ABSTRACT);
    ANGLE (ABSTRACT);
    SOLID_ANGLE (ABSTRACT);
    LUMINOUS_INTENSITY (ABSTRACT);
    MONEY (ABSTRACT);

    METER [m] EXTENDS LENGTH;
    KILOGRAM [kg] EXTENDS MASS;
    SECOND [s] EXTENDS TIME;
    AMPERE [A] EXTENDS ELECTRIC_CURRENT;
    DEGREE_KELVIN [K] EXTENDS TEMPERATURE;
    MOLE [mol] EXTENDS AMOUNT_OF_MATTER;
    RADIAN [rad] EXTENDS ANGLE;
    STERADIAN [sr] EXTENDS SOLID_ANGLE;
    CANDELA [cd] EXTENDS LUMINOUS_INTENSITY;

  DOMAIN
    URI (FINAL) = TEXT*1023;
    NAME (FINAL) = TEXT*255;
    INTERLIS_1_DATE (FINAL) = TEXT*8;
    BOOLEAN (FINAL) = (
      false,
      true) ORDERED;
    HALIGNMENT (FINAL) = (
      Left,
      Center,
      Right) ORDERED;
    VALIGNMENT (FINAL) = (
      Top,
      Cap,
      Half,
      Base,
      Bottom) ORDERED;
    ANYOID (ABSTRACT) = OID ANY;
    I32OID = OID 0 .. 2147483647;
    STANDARDOID = TEXT*16;
    LineCoord (ABSTRACT) = COORD NUMERIC, NUMERIC, NUMERIC;
```

```
FUNCTION myClass (Object: OBJECT OF ANYSTRUCTURE): STRUCTURE;
FUNCTION issubClass (potSubClass: STRUCTURE; potSuperClass: STRUCTURE):
  BOOLEAN;
FUNCTION isofClass (Object: OBJECT OF ANYSTRUCTURE; Class: STRUCTURE): BOOLEAN;
FUNCTION elementCount (bag: BAG OF ANYSTRUCTURE): NUMERIC;
FUNCTION convertUnit (from: NUMERIC): NUMERIC;

STRUCTURE LineSegment (ABSTRACT) =
  SegmentEndPoint: MANDATORY LineCoord;
END LineSegment;

STRUCTURE StartSegment (FINAL) EXTENDS LineSegment =
END StartSegment;

STRUCTURE StraightSegment (FINAL) EXTENDS LineSegment =
END StraightSegment;

STRUCTURE ArcSegment (FINAL) EXTENDS LineSegment =
  ArcPoint: MANDATORY LineCoord;
  Radius: NUMERIC [LENGTH];
END ArcSegment;

STRUCTURE SurfaceEdge =
  Geometry: DIRECTED POLYLINE;
  LineAttrs: ANYSTRUCTURE;
END SurfaceEdge;

STRUCTURE SurfaceBoundary =
  Lines: LIST OF SurfaceEdge;
END SurfaceBoundary;

STRUCTURE LineGeometry =
  Segments: LIST OF LineSegment;
MANDATORY CONSTRAINT isOfClass (Segments[FIRST],StartSegment);
END LineGeometry;

STRUCTURE Basket (ABSTRACT) =
  Model: MANDATORY NAME;
  Topic: MANDATORY NAME;
  Kind: MANDATORY (Data, View, Base, Graphic);
  Ident (ABSTRACT): MANDATORY ANYOID;
END Basket;

CLASS METAOBJECT (ABSTRACT) =
  Name: MANDATORY NAME;
UNIQUE Name;
END METAOBJECT;

CLASS METAOBJECT_TRANSLATION =
  Name: MANDATORY NAME;
  NameInBaseLanguage: MANDATORY NAME;
UNIQUE Name;
UNIQUE NameInBaseLanguage;
END METAOBJECT_TRANSLATION;

STRUCTURE AXIS =
  PARAMETER
    Unit: NUMERIC [ANYUNIT];
END AXIS;

CLASS REFSYSTEM (ABSTRACT) EXTENDS METAOBJECT =
END REFSYSTEM;

CLASS COORDSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  ATTRIBUTE
    Axis: LIST {1..*} OF AXIS;
END COORDSYSTEM;
```

```
    CLASS SCALSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
      PARAMETER
        Unit: NUMERIC [ANYUNIT];
    END SCALSYSTEM;

    CLASS SIGN (ABSTRACT) EXTENDS METAOBJECT =
      PARAMETER
        Sign: METAOBJECT;
    END SIGN;

  END INTERLIS.
```

Annexe B (Norme pour la Suisse) Table des caractères

La table suivante énumère tous les caractères, caractères spéciaux, accents et signes diacritiques disponibles de façon standard dans INTERLIS 2 de même que leur codage dans un transfert INTERLIS. Plusieurs formes de codage coexistent pour certains caractères et toutes les formes de codage possibles du caractère sont indiquées dans de tels cas. Un programme d'écriture en INTERLIS 2 peut librement sélectionner l'une des différentes possibilités de codage proposées s'il en existe plusieurs pour un caractère donné. En revanche, un programme de lecture en INTERLIS 2 doit pouvoir reconnaître toutes les formes de codage possibles d'un tel caractère.

Cette table ne s'applique qu'à un contenu XML (c.-à-d. chaîne (String) ou valeur (Value) XML). Les tags XML sont exclusivement transmis sous forme de caractères en codage ASCII conformément à la syntaxe définie au chapitre 3.

D'autres caractères peuvent également être utilisés dans des applications INTERLIS 2, en plus des caractères standard répertoriés dans la table. Dans une telle éventualité, un contrat doit obligatoirement être conclu entre les différentes parties impliquées.

UCS Hexa.	UCS Déci.	Codage UTF-8 Octet Hexa.	Codage XML Character Reference Déci.	Codage XML Character Reference Hexa.	Codage XML Entity Reference	Représentation
0020	32	20				
0021	33	21				!
0022	34	22				"
0023	35	23				#
0024	36	24				$
0025	37	25				%
0026	38		&#38;	&#x26;	&amp;	&
0027	39	27				'
0028	40	28				(
0029	41	29				)
002A	42	2A				*
002B	43	2B				+
002C	44	2C				,
002D	45	2D				-
002E	46	2E				.
002F	47	2F				/
0030	48	30				0
0031	49	31				1
0032	50	32				2
0033	51	33				3
0034	52	34				4
0035	53	35				5
0036	54	36				6
0037	55	37				7
0038	56	38				8
0039	57	39				9
003A	58	3A				:
003B	59	3B				;
003C	60		&#60;	&#x3c;	&lt;	<
003D	61	3D				=

UCS Hexa.	UCS Déci.	Codage UTF-8 Octet Hexa.	Codage XML Character Reference Déci.	Codage XML Character Reference Hexa.	Codage XML Entity Reference	Représentation
003E	62	3E	&#62;	&#x3e;	&gt;	>
003F	63	3F				?
0040	64	40				@
0041	65	41				A
0042	66	42				B
0043	67	43				C
0044	68	44				D
0045	69	45				E
0046	70	46				F
0047	71	47				G
0048	72	48				H
0049	73	49				I
004A	74	4A				J
004B	75	4B				K
004C	76	4C				L
004D	77	4D				M
004E	78	4E				N
004F	79	4F				O
0050	80	50				P
0051	81	51				Q
0052	82	52				R
0053	83	53				S
0054	84	54				T
0055	85	55				U
0056	86	56				V
0057	87	57				W
0058	88	58				X
0059	89	59				Y
005A	90	5A				Z
005B	91	5B				[
005C	92	5C				\
005D	93	5D				]
005E	94	5E				^
005F	95	5F				_
0060	96	60				`
0061	97	61				a
0062	98	62				b
0063	99	63				c
0064	100	64				d
0065	101	65				e
0066	102	66				f
0067	103	67				g
0068	104	68				h
0069	105	69				i
006A	106	6A				j
006B	107	6B				k
006C	108	6C				l
006D	109	6D				m
006E	110	6E				n
006F	111	6F				o
0070	112	70				p
0071	113	71				q
0072	114	72				r

UCS Hexa.	UCS Déci.	Codage UTF-8 Octet Hexa.	Codage XML Character Reference Déci.	Codage XML Character Reference Hexa.	Codage XML Entity Reference	Représentation
0073	115	73				s
0074	116	74				t
0075	117	75				u
0076	118	76				v
0077	119	77				w
0078	120	78				x
0079	121	79				y
007A	122	7A				z
007B	123	7B				{
007C	124	7C				|
007D	125	7D				}
007E	126	7E				~
00C4	196	C3 84	Ä	Ä		Ä
00C6	198	C3 86	Æ	Æ		Æ
00C7	199	C3 87	Ç	Ç		Ç
00C8	200	C3 88	È	È		È
00C9	201	C3 89	É	É		É
00D1	209	C3 91	Ñ	Ñ		Ñ
00D6	214	C3 96	Ö	Ö		Ö
00DC	220	C3 9C	Ü	Ü		Ü
00E0	224	C3 A0	à	à		à
00E1	225	C3 A1	á	á		á
00E2	226	C3 A2	â	â		â
00E4	228	C3 A4	ä	ä		ä
00E6	230	C3 A6	æ	æ		æ
00E7	231	C3 A7	ç	ç		ç
00E8	232	C3 A8	è	è		è
00E9	233	C3 A9	é	é		é
00EA	234	C3 AA	ê	ê		ê
00EB	235	C3 AB	ë	ë		ë
00EC	236	C3 AC	ì	ì		ì
00ED	237	C3 AD	í	í		í
00EE	238	C3 AE	î	î		î
00EF	239	C3 AF	ï	ï		ï
00F1	241	C3 B1	ñ	ñ		ñ
00F2	242	C3 B2	ò	ò		ò
00F3	243	C3 B3	ó	ó		ó
00F4	244	C3 B4	ô	ô		ô
00F5	245	C3 B5	õ	õ		õ
00F6	246	C3 B6	ö	ö		ö
00F9	249	C3 B9	ù	ù		ù
00FA	250	C3 BA	ú	ú		ú
00FB	251	C3 BB	û	û		û
00FC	252	C3 BC	ü	ü		ü

Table 2 : Caractères UCS/Unicode admis par INTERLIS 2 et leur codage.

Remarques :

- Le code UCS (ou Unicode) du caractère (hexadécimal ou décimal) est indiqué dans les colonnes UCS Hexa. ou UCS Déci.
- Le codage du caractère selon UTF-8 en octets de 8 bits est indiqué en notation hexadécimale dans la colonne Codage UTF-8. Les caractères >= Hex 80 sont codés sous forme de suites de plusieurs

octets. Remarque : la notation hexadécimale n'est utilisée ici que pour l'explication du codage. Seuls les octets en codage binaire seront transmis lors du transfert.

- Le codage du caractère sous forme "XML Character Reference" est indiqué dans les colonnes Codage XML Character Reference (Déci.) ou Character Reference (Hexa.) (variante décimale ou hexadécimale). La valeur est une suite de caractères ASCII et doit être utilisée en l'état lors du transfert. Un programme d'écriture devrait si possible sélectionner le codage Character Reference pour un caractère. Ce codage a pour avantage de pouvoir être affiché sur tous les types de plates-formes (Unix, PC, etc.) puis modifié à l'aide d'éditeurs ASCII simples. Remarque: les lettres a – f peuvent être indiquées en majuscules ou en minuscules dans le cas de la solution de codage hexadécimale (cependant, la lettre x doit toujours être une minuscule).
- Le codage XML (Entity Reference) n'est permis que pour un nombre limité de caractères spéciaux. La valeur est une suite de caractères ASCII devant être utilisée en l'état lors du transfert. Remarque : des entités (Entities) XML telles que ü (pour le caractère ü) ne sont pas permises dans un fichier de transfert INTERLIS 2 car aucun DTD n'est référencé par un fichier de transfert INTERLIS 2 (les entités (Entities) utilisables telles que & sont prédéfinies par la spécification XML 1.0).
  - La colonne Représentation indique la représentation du caractère dans un éditeur compatible avec UCS ou Unicode.

Annexe C (Information) Le petit exemple Roads

Introduction

Pour une initiation plus facile à INTERLIS 2, nous présentons dans cette annexe un petit exemple complet. Il décrit un jeu de données qui sera utilisé pour un transfert complet [FULL]. Pour des exemples d'application de mise à jour incrémentielle (incl. OID), un jeu de données et un manuel de l'utilisateur sont en préparation.

Cet exemple comprend les parties suivantes :

- Les modèles de données RoadsExdm2ben_10 et RoadsExdm2ien_10.
- Le jeu de données XML RoadsExdm2ien (fichier RoadsExdm2ien.xml) contenant les objets selon le modèle de données RoadsExdm2ien_10.
- Le modèle graphique RoadsExgm2ien_10. Ce modèle graphique propose une représentation graphique possible du modèle de données RoadsExdm2ein_10 (remarque : pour un même modèle, plusieurs représentations sont possibles).
- Une collection d'objets de signatures (bibliothèque de signatures) dans RoadsExgm2ien_Symbols (fichier RoadsExgm2ien_Symbols.xml). La bibliothèque de signatures est un jeu de données XML selon les spécifications du modèle de signatures StandardSymbology (cf. Annexe K Modèle de signatures). La bibliothèque de signatures sera utilisée dans le modèle graphique RaodsExgm2ien_10 pour la représentation des données de l'exemple à partir du fichier RoadsExdm2ien.xml.

Le nom "RoadsExdm2ben_10" est l'abréviation de "**Roads Ex**ample, **D**ata **M**odel, INTERLIS **2**, **B**asic Model, **en**glish, Release **_10**". Les différentes parties de ce modèle sont décrites avec plus de détails ci-après.

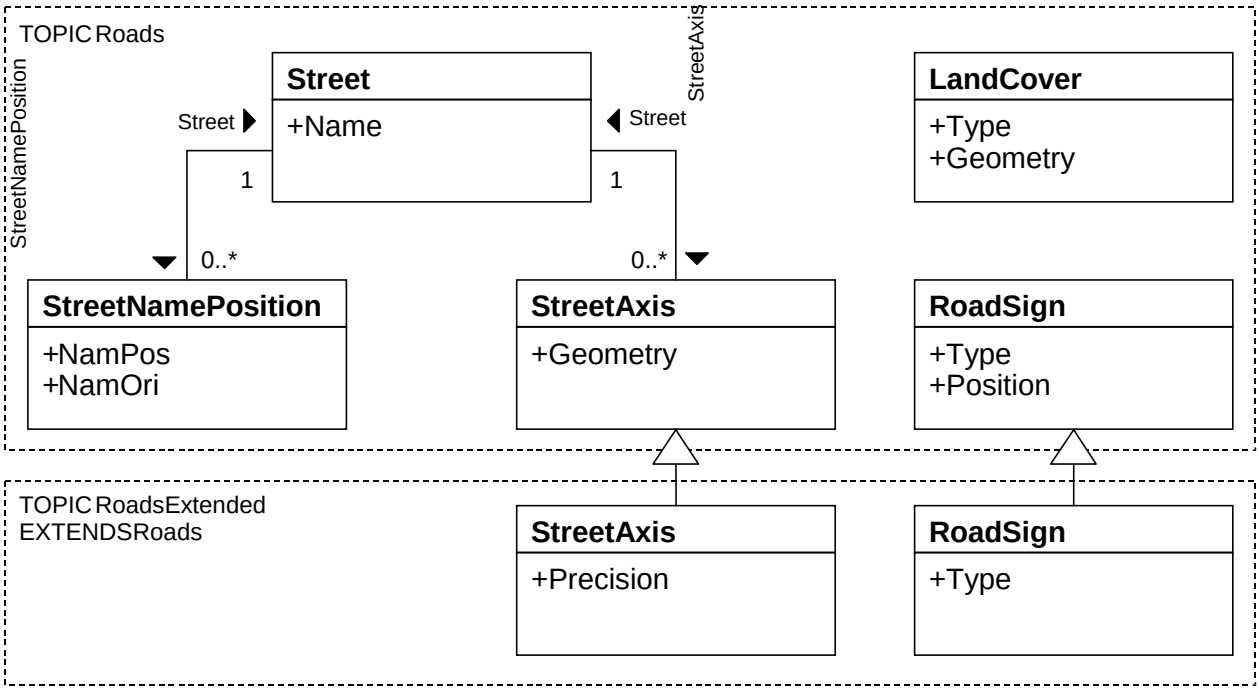


Figure 26 : Modèle conceptuel UML du "petit exemple Roads".

Modèles de donnée RoadsExdm2ben_10 et RoadsExdm2ien_10

Le modèle de données RoadsExdm2ben_10 contient les objets LandCover (surfaces de la couverture du sol), StreetAxis (axes de rues), StreetName (noms de rues) et PointObject (objets ponctuels). Le modèle de données RoadsExdm2ien_10 constitue une extension de RoadsExdm2ben_10. Les modèles de données sont récapitulés dans le modèle conceptuel UML (cf. Figure 26 : Modèle conceptuel UML).

Remarque : cet exemple est volontairement d'une très grande simplicité et ne prétend par conséquent pas être exhaustif. Les modèles associés décrits en INTERLIS 2 se présentent comme suit :

```
!! File RoadsExdm2ben_10.ili Release 2003-02-26

INTERLIS 2.2;

MODEL RoadsExdm2ben_10 (en) =

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    Point2D = COORD
      0.000 .. 200.000 [INTERLIS.m], !! Min_East  Max_East
      0.000 .. 200.000 [INTERLIS.m], !! Min_North Max_North
      ROTATION 2 -> 1;
    Orientation = 0.0 .. 359.9 CIRCULAR [Angle_Degree];


  TOPIC Roads =

    STRUCTURE LAttrs =
      LArt: (
        welldefined,
        fuzzy);
    END LAttrs;

    CLASS LandCover =
      Type: MANDATORY (
        building,
        street,
        water,
        other);
      Geometry: MANDATORY SURFACE WITH (STRAIGHTS)
        VERTEX Point2D WITHOUT OVERLAPS > 0.100
        LINE ATTRIBUTES LAttrs;
    END LandCover;

    CLASS Street =
      Name: MANDATORY TEXT*32;
    END Street;

    CLASS StreetAxis =
      Geometry: MANDATORY POLYLINE WITH (STRAIGHTS)
        VERTEX Point2D;
    END StreetAxis;

    ASSOCIATION StreetAxisAssoc =
      Street -- {1} Street;
      StreetAxis -- StreetAxis;
    END StreetAxisAssoc;

    CLASS StreetNamePosition =
      NamPos: MANDATORY Point2D;
      NamOri: MANDATORY Orientation;
    END StreetNamePosition;

    ASSOCIATION StreetNamePositionAssoc =
      Street -- {0..1} Street;
```

```
      StreetNamePosition -- StreetNamePosition;
    END StreetNamePositionAssoc;

    CLASS RoadSign =
      Type: MANDATORY (
        prohibition,
        indication,
        danger,
        velocity);
      Position: MANDATORY Point2D;
    END RoadSign;

  END Roads; !! of TOPIC

END RoadsExdm2ben_10. !! of MODEL


!! File RoadsExdm2ien_10.ili Release 2003-02-26

INTERLIS 2.2;

MODEL RoadsExdm2ien_10 (en) =

  IMPORTS RoadsExdm2ben_10;


  TOPIC RoadsExtended EXTENDS RoadsExdm2ben_10.Roads =

    CLASS StreetAxis (EXTENDED) =
      Precision: MANDATORY (
        precise,
        unprecise);
    END StreetAxis;

    CLASS RoadSign (EXTENDED) =
      Type (EXTENDED): (
        prohibition (
          noentry,
          noparking,
          other));
    END RoadSign;

  END RoadsExtended; !! of TOPIC

END RoadsExdm2ien_10. !! of MODEL
```

Jeu de données RoadsExdm2ien selon le modèle de données RoadsExdm2ien_10

Un exemple de jeu de données est indiqué ci-après pour le modèle de données RoadsExdm2ien_10. Le formatage XML a été déduit du modèle de données RoadsExdm2ien_10 conformément aux règles du chapitre 3 Transfert séquentiel.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- File RoadsExdm2ien.xml 2003-02-26 -->

<TRANSFER xmlns="http://www.interlis.ch/INTERLIS2.2"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.interlis.ch/INTERLIS2.2
                              RoadsExdm2ien_10.xsd">
  <HEADERSECTION VERSION="2.2" SENDER="V+D">
    <ALIAS>
      <ENTRIES FOR="RoadsExdm2ben_10">
        <TAGENTRY FROM="RoadsExdm2ben_10.Roads"
                  TO="RoadsExdm2ben_10.Roads"/>
        <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended"
```

```
            TO="RoadsExdm2ben_10.Roads"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.LAttrs"
            TO="RoadsExdm2ben_10.Roads.LAttrs"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.LandCover"
            TO="RoadsExdm2ben_10.Roads.LandCover"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.Street"
            TO="RoadsExdm2ben_10.Roads.Street"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.StreetAxis"
            TO="RoadsExdm2ben_10.Roads.StreetAxis"/>
  <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended.StreetAxis"
            TO="RoadsExdm2ben_10.Roads.StreetAxis"/>
  <DELENTRY TAG="RoadsExdm2ien_10.RoadsExtended.StreetAxis.Precision"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.StreetAxisAssoc"
            TO="RoadsExdm2ben_10.Roads.StreetAxisAssoc"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.StreetNamePosition"
            TO="RoadsExdm2ben_10.Roads.StreetNamePosition"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.StreetNamePositionAssoc"
            TO="RoadsExdm2ben_10.Roads.StreetNamePositionAssoc"/>
  <TAGENTRY FROM="RoadsExdm2ben_10.Roads.RoadSign"
            TO="RoadsExdm2ben_10.Roads.RoadSign"/>
  <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended.RoadSign"
            TO="RoadsExdm2ben_10.Roads.RoadSign"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LAttrs.LArt"
            FROM="welldefined" TO="welldefined"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LAttrs.LArt"
            FROM="fuzzy" TO="fuzzy"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LandCover.Type"
            FROM="building" TO="building"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LandCover.Type"
            FROM="street" TO="street"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LandCover.Type"
            FROM="water" TO="water"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.LandCover.Type"
            FROM="other" TO="other"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.RoadSign.Type"
            FROM="prohibition" TO="prohibition"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.RoadSign.Type"
            FROM="indication" TO="indication"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.RoadSign.Type"
            FROM="danger" TO="danger"/>
  <VALENTRY ATTR="RoadsExdm2ben_10.Roads.RoadSign.Type"
            FROM="velocity" TO="velocity"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.noentry" TO="prohibition"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.noparking" TO="prohibition"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.other" TO="prohibition"/>
</ENTRIES>

<ENTRIES FOR="RoadsExdm2ien_10">
  <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended"
            TO="RoadsExdm2ien_10.RoadsExtended"/>
  <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended.StreetAxis"
            TO="RoadsExdm2ien_10.RoadsExtended.StreetAxis"/>
  <TAGENTRY FROM="RoadsExdm2ien_10.RoadsExtended.RoadSign"
            TO="RoadsExdm2ien_10.RoadsExtended.RoadSign"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.StreetAxis.Precision"
            FROM="precise" TO="precise"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.StreetAxis.Precision"
            FROM="unprecise" TO="unprecise"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.noentry" TO="prohibition.noentry"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.noparking" TO="prohibition.noparking"/>
  <VALENTRY ATTR="RoadsExdm2ien_10.RoadsExtended.RoadSign.Type"
            FROM="prohibition.other" TO="prohibition.other"/>
</ENTRIES>
```

```
    </ALIAS>

    <COMMENT>
      example dataset ili2 refmanual appendix C
    </COMMENT>
  </HEADERSECTION>

  <DATASECTION>
    <RoadsExdm2ien_10.RoadsExtended BID="xREFHANDB00000001">

      <!-- === LandCover === -->
      <RoadsExdm2ben_10.Roads.LandCover TID="x16">
        <Type>water</Type>
        <Geometry>
          <SURFACE>
            <BOUNDARY>
              <POLYLINE>
                <LINEATTR>
                  <RoadsExdm2ben_10.Roads.LAttrs>
                    <LArt>welldefined</LArt>
                  </RoadsExdm2ben_10.Roads.LAttrs>
                </LINEATTR>
                <COORD><C1>39.038</C1><C2>60.315</C2></COORD>
                <COORD><C1>41.200</C1><C2>59.302</C2></COORD>
                <COORD><C1>43.362</C1><C2>60.315</C2></COORD>
                <COORD><C1>44.713</C1><C2>66.268</C2></COORD>
                <COORD><C1>45.794</C1><C2>67.662</C2></COORD>
                <COORD><C1>48.766</C1><C2>67.408</C2></COORD>
                <COORD><C1>53.360</C1><C2>64.115</C2></COORD>
                <COORD><C1>56.197</C1><C2>62.595</C2></COORD>
                <COORD><C1>57.818</C1><C2>63.862</C2></COORD>
                <COORD><C1>58.899</C1><C2>68.928</C2></COORD>
                <COORD><C1>55.927</C1><C2>72.348</C2></COORD>
                <COORD><C1>47.955</C1><C2>75.515</C2></COORD>
                <COORD><C1>42.281</C1><C2>75.388</C2></COORD>
                <COORD><C1>39.308</C1><C2>73.235</C2></COORD>
                <COORD><C1>36.741</C1><C2>69.688</C2></COORD>
                <COORD><C1>35.525</C1><C2>66.268</C2></COORD>
                <COORD><C1>35.661</C1><C2>63.735</C2></COORD>
                <COORD><C1>37.957</C1><C2>61.455</C2></COORD>
                <COORD><C1>39.038</C1><C2>60.315</C2></COORD>
              </POLYLINE>
            </BOUNDARY>
          </SURFACE>
        </Geometry>
      </RoadsExdm2ben_10.Roads.LandCover>

      <RoadsExdm2ben_10.Roads.LandCover TID="x18">
        <Type>building</Type>
        <Geometry>
          <SURFACE>
            <BOUNDARY>
              <POLYLINE>
                <LINEATTR>
                  <RoadsExdm2ben_10.Roads.LAttrs>
                    <LArt>welldefined</LArt>
                  </RoadsExdm2ben_10.Roads.LAttrs>
                </LINEATTR>
                <COORD><C1>101.459</C1><C2>65.485</C2></COORD>
                <COORD><C1>108.186</C1><C2>69.369</C2></COORD>
                <COORD><C1>102.086</C1><C2>79.936</C2></COORD>
                <COORD><C1>95.359</C1><C2>76.053</C2></COORD>
                <COORD><C1>101.459</C1><C2>65.485</C2></COORD>
              </POLYLINE>
            </BOUNDARY>
          </SURFACE>
        </Geometry>
      </RoadsExdm2ben_10.Roads.LandCover>
```

```
<RoadsExdm2ben_10.Roads.LandCover TID="x20">
  <Type>building</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>60.489</C1><C2>49.608</C2></COORD>
          <COORD><C1>79.900</C1><C2>55.839</C2></COORD>
          <COORD><C1>75.351</C1><C2>70.932</C2></COORD>
          <COORD><C1>67.678</C1><C2>68.781</C2></COORD>
          <COORD><C1>69.938</C1><C2>61.721</C2></COORD>
          <COORD><C1>57.582</C1><C2>58.029</C2></COORD>
          <COORD><C1>60.489</C1><C2>49.608</C2></COORD>
        </POLYLINE>
     </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x22">
  <Type>street</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>45.067</C1><C2>58.655</C2></COORD>
          <COORD><C1>50.669</C1><C2>42.579</C2></COORD>
          <COORD><C1>57.060</C1><C2>44.638</C2></COORD>
          <COORD><C1>51.432</C1><C2>60.469</C2></COORD>
          <COORD><C1>45.067</C1><C2>58.655</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x24">
  <Type>other</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>114.027</C1><C2>99.314</C2></COORD>
          <COORD><C1>31.351</C1><C2>99.314</C2></COORD>
          <COORD><C1>31.140</C1><C2>92.530</C2></COORD>
          <COORD><C1>70.419</C1><C2>86.177</C2></COORD>
          <COORD><C1>86.481</C1><C2>79.710</C2></COORD>
          <COORD><C1>114.027</C1><C2>99.314</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
```

```
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x26">
  <Type>other</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>113.559</C1><C2>62.880</C2></COORD>
          <COORD><C1>114.027</C1><C2>99.314</C2></COORD>
          <COORD><C1>86.481</C1><C2>79.710</C2></COORD>
          <COORD><C1>94.381</C1><C2>66.289</C2></COORD>
          <COORD><C1>96.779</C1><C2>57.177</C2></COORD>
          <COORD><C1>113.559</C1><C2>62.880</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>108.186</C1><C2>69.369</C2></COORD>
          <COORD><C1>101.459</C1><C2>65.485</C2></COORD>
          <COORD><C1>95.359</C1><C2>76.053</C2></COORD>
          <COORD><C1>102.086</C1><C2>79.936</C2></COORD>
          <COORD><C1>108.186</C1><C2>69.369</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x29">
  <Type>street</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>100.621</C1><C2>24.239</C2></COORD>
          <COORD><C1>109.729</C1><C2>24.239</C2></COORD>
          <COORD><C1>105.640</C1><C2>48.068</C2></COORD>
          <COORD><C1>96.779</C1><C2>45.088</C2></COORD>
          <COORD><C1>100.621</C1><C2>24.239</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x31">
  <Type>other</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
```

```
              <RoadsExdm2ben_10.Roads.LAttrs>
                <LArt>welldefined</LArt>
              </RoadsExdm2ben_10.Roads.LAttrs>
            </LINEATTR>
            <COORD><C1>30.900</C1><C2>24.478</C2></COORD>
            <COORD><C1>100.621</C1><C2>24.239</C2></COORD>
            <COORD><C1>96.779</C1><C2>45.088</C2></COORD>
            <COORD><C1>30.900</C1><C2>24.478</C2></COORD>
          </POLYLINE>
        </BOUNDARY>
      </SURFACE>
    </Geometry>
  </RoadsExdm2ben_10.Roads.LandCover>

  <RoadsExdm2ben_10.Roads.LandCover TID="x33">
    <Type>other</Type>
    <Geometry>
      <SURFACE>
        <BOUNDARY>
          <POLYLINE>
            <LINEATTR>

              <RoadsExdm2ben_10.Roads.LAttrs>
                <LArt>welldefined</LArt>
              </RoadsExdm2ben_10.Roads.LAttrs>
            </LINEATTR>
            <COORD><C1>31.110</C1><C2>83.750</C2></COORD>
            <COORD><C1>31.140</C1><C2>36.458</C2></COORD>
            <COORD><C1>50.669</C1><C2>42.579</C2></COORD>
            <COORD><C1>45.067</C1><C2>58.655</C2></COORD>
            <COORD><C1>51.432</C1><C2>60.469</C2></COORD>
            <COORD><C1>57.060</C1><C2>44.638</C2></COORD>
            <COORD><C1>87.839</C1><C2>54.138</C2></COORD>
            <COORD><C1>85.282</C1><C2>63.410</C2></COORD>
            <COORD><C1>78.847</C1><C2>73.433</C2></COORD>
            <COORD><C1>67.549</C1><C2>77.788</C2></COORD>
            <COORD><C1>31.110</C1><C2>83.750</C2></COORD>
          </POLYLINE>
        </BOUNDARY>
        <BOUNDARY>
          <POLYLINE>
            <LINEATTR>
              <RoadsExdm2ben_10.Roads.LAttrs>
                <LArt>welldefined</LArt>
              </RoadsExdm2ben_10.Roads.LAttrs>
            </LINEATTR>
            <COORD><C1>41.200</C1><C2>59.302</C2></COORD>
            <COORD><C1>39.038</C1><C2>60.315</C2></COORD>
            <COORD><C1>37.957</C1><C2>61.455</C2></COORD>
            <COORD><C1>35.661</C1><C2>63.735</C2></COORD>
            <COORD><C1>35.525</C1><C2>66.268</C2></COORD>
            <COORD><C1>36.741</C1><C2>69.688</C2></COORD>
            <COORD><C1>39.308</C1><C2>73.235</C2></COORD>
            <COORD><C1>42.281</C1><C2>75.388</C2></COORD>
            <COORD><C1>47.955</C1><C2>75.515</C2></COORD>
            <COORD><C1>55.927</C1><C2>72.348</C2></COORD>
            <COORD><C1>58.899</C1><C2>68.928</C2></COORD>
            <COORD><C1>57.818</C1><C2>63.862</C2></COORD>
            <COORD><C1>56.197</C1><C2>62.595</C2></COORD>
            <COORD><C1>53.360</C1><C2>64.115</C2></COORD>
            <COORD><C1>48.766</C1><C2>67.408</C2></COORD>
            <COORD><C1>45.794</C1><C2>67.662</C2></COORD>
            <COORD><C1>44.713</C1><C2>66.268</C2></COORD>
            <COORD><C1>43.362</C1><C2>60.315</C2></COORD>
            <COORD><C1>41.200</C1><C2>59.302</C2></COORD>
          </POLYLINE>
        </BOUNDARY>
        <BOUNDARY>
```

```
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>79.900</C1><C2>55.839</C2></COORD>
          <COORD><C1>60.489</C1><C2>49.608</C2></COORD>
          <COORD><C1>57.582</C1><C2>58.029</C2></COORD>
          <COORD><C1>69.938</C1><C2>61.721</C2></COORD>
          <COORD><C1>67.678</C1><C2>68.781</C2></COORD>
          <COORD><C1>75.351</C1><C2>70.932</C2></COORD>
          <COORD><C1>79.900</C1><C2>55.839</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x37">
  <Type>street</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>31.140</C1><C2>92.530</C2></COORD>
          <COORD><C1>31.110</C1><C2>83.750</C2></COORD>
          <COORD><C1>67.549</C1><C2>77.788</C2></COORD>
          <COORD><C1>78.847</C1><C2>73.433</C2></COORD>
          <COORD><C1>85.282</C1><C2>63.410</C2></COORD>
          <COORD><C1>87.839</C1><C2>54.138</C2></COORD>
          <COORD><C1>96.779</C1><C2>57.177</C2></COORD>
          <COORD><C1>94.381</C1><C2>66.289</C2></COORD>
          <COORD><C1>86.481</C1><C2>79.710</C2></COORD>
          <COORD><C1>70.419</C1><C2>86.177</C2></COORD>
          <COORD><C1>31.140</C1><C2>92.530</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<RoadsExdm2ben_10.Roads.LandCover TID="x39">
  <Type>other</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>113.811</C1><C2>51.168</C2></COORD>
          <COORD><C1>105.640</C1><C2>48.068</C2></COORD>
          <COORD><C1>109.729</C1><C2>24.239</C2></COORD>
          <COORD><C1>114.269</C1><C2>24.017</C2></COORD>
          <COORD><C1>113.811</C1><C2>51.168</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>
```

```
<RoadsExdm2ben_10.Roads.LandCover TID="x41">
  <Type>street</Type>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <LINEATTR>
            <RoadsExdm2ben_10.Roads.LAttrs>
              <LArt>welldefined</LArt>
            </RoadsExdm2ben_10.Roads.LAttrs>
          </LINEATTR>
          <COORD><C1>105.640</C1><C2>48.068</C2></COORD>
          <COORD><C1>113.811</C1><C2>51.168</C2></COORD>
          <COORD><C1>113.559</C1><C2>62.880</C2></COORD>
          <COORD><C1>96.779</C1><C2>57.177</C2></COORD>
          <COORD><C1>87.839</C1><C2>54.138</C2></COORD>
          <COORD><C1>57.060</C1><C2>44.638</C2></COORD>
          <COORD><C1>50.669</C1><C2>42.579</C2></COORD>
          <COORD><C1>31.140</C1><C2>36.458</C2></COORD>
          <COORD><C1>30.900</C1><C2>24.478</C2></COORD>
          <COORD><C1>96.779</C1><C2>45.088</C2></COORD>
          <COORD><C1>105.640</C1><C2>48.068</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</RoadsExdm2ben_10.Roads.LandCover>

<!-- === Street === -->
<RoadsExdm2ben_10.Roads.Street TID="x1">
  <Name>Austrasse</Name>
</RoadsExdm2ben_10.Roads.Street>

<RoadsExdm2ben_10.Roads.Street TID="x2">
  <Name>Eymattstrasse</Name>
</RoadsExdm2ben_10.Roads.Street>

<RoadsExdm2ben_10.Roads.Street TID="x3">
  <Name>Feldweg</Name>
</RoadsExdm2ben_10.Roads.Street>

<RoadsExdm2ben_10.Roads.Street TID="x4">
  <Name>Seeweg</Name>
</RoadsExdm2ben_10.Roads.Street>

<!-- === StreetAxis / StreetAxisAssoc === -->
<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x8">
  <Geometry>
    <POLYLINE>
      <COORD><C1>55.600</C1><C2>37.649</C2></COORD>
      <COORD><C1>15.573</C1><C2>25.785</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x1"></Street>
  <Precision>precise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x9">
  <Geometry>
    <POLYLINE>
      <COORD><C1>55.600</C1><C2>37.649</C2></COORD>
      <COORD><C1>94.990</C1><C2>50.109</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x1"></Street>
  <Precision>precise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>
```

```
<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x10">
  <Geometry>
    <POLYLINE>
      <COORD><C1>94.990</C1><C2>50.109</C2></COORD>
      <COORD><C1>101.099</C1><C2>52.279</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x1"></Street>
  <Precision>precise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x11">
  <Geometry>
    <POLYLINE>
      <COORD><C1>101.099</C1><C2>52.279</C2></COORD>
      <COORD><C1>126.100</C1><C2>62.279</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x1"></Street>
  <Precision>precise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x12">
  <Geometry>
    <POLYLINE>
      <COORD><C1>94.990</C1><C2>50.109</C2></COORD>
      <COORD><C1>89.504</C1><C2>65.795</C2></COORD>
      <COORD><C1>83.594</C1><C2>75.598</C2></COORD>
      <COORD><C1>71.774</C1><C2>80.712</C2></COORD>
      <COORD><C1>11.423</C1><C2>91.154</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x2"></Street>
  <Precision>precise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x13">
  <Geometry>
    <POLYLINE>
      <COORD><C1>101.099</C1><C2>52.279</C2></COORD>
      <COORD><C1>107.400</C1><C2>14.603</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x3"></Street>
  <Precision>unprecise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<RoadsExdm2ien_10.RoadsExtended.StreetAxis TID="x15">
  <Geometry>
    <POLYLINE>
      <COORD><C1>55.600</C1><C2>37.649</C2></COORD>
      <COORD><C1>49.359</C1><C2>56.752</C2></COORD>
    </POLYLINE>
  </Geometry>
  <Street REF="x4"></Street>
  <Precision>unprecise</Precision>
</RoadsExdm2ien_10.RoadsExtended.StreetAxis>

<!-- === StreetNamePosition / StreetNamePositionAssoc === -->
<RoadsExdm2ben_10.Roads.StreetNamePosition TID="x5">
  <NamPos>
    <COORD><C1>71.660</C1><C2>45.231</C2></COORD>
  </NamPos>
  <NamOri>15.0</NamOri>
  <Street REF="x1"></Street>
</RoadsExdm2ben_10.Roads.StreetNamePosition>
```

```
        <RoadsExdm2ben_10.Roads.StreetNamePosition TID="x6">
          <NamPos>
            <COORD><C1>58.249</C1><C2>85.081</C2></COORD>
          </NamPos>
          <NamOri>351.0</NamOri>
          <Street REF="x2"></Street>
        </RoadsExdm2ben_10.Roads.StreetNamePosition>

        <RoadsExdm2ben_10.Roads.StreetNamePosition TID="x7">
          <NamPos>
            <COORD><C1>106.095</C1><C2>33.554</C2></COORD>
          </NamPos>
          <NamOri>280.0</NamOri>
          <Street REF="x3"></Street>
        </RoadsExdm2ben_10.Roads.StreetNamePosition>

        <RoadsExdm2ben_10.Roads.StreetNamePosition TID="x14">
          <NamPos>
            <COORD><C1>53.031</C1><C2>51.367</C2></COORD>
          </NamPos>
          <NamOri>291.3</NamOri>
          <Street REF="x4"></Street>
        </RoadsExdm2ben_10.Roads.StreetNamePosition>

        <!-- === RoadSign === -->
        <RoadsExdm2ien_10.RoadsExtended.RoadSign TID="x501">
          <Type>prohibition.noparking</Type>
          <Position>
            <COORD><C1>69.389</C1><C2>92.056</C2></COORD>
          </Position>
        </RoadsExdm2ien_10.RoadsExtended.RoadSign>

        <RoadsExdm2ien_10.RoadsExtended.RoadSign TID="x502">
          <Type>prohibition.noparking</Type>
          <Position>
            <COORD><C1>80.608</C1><C2>88.623</C2></COORD>
          </Position>
        </RoadsExdm2ien_10.RoadsExtended.RoadSign>

        <RoadsExdm2ien_10.RoadsExtended.RoadSign TID="x503">
          <Type>prohibition.noparking</Type>
          <Position>
            <COORD><C1>58.059</C1><C2>93.667</C2></COORD>
          </Position>
        </RoadsExdm2ien_10.RoadsExtended.RoadSign>

        <RoadsExdm2ien_10.RoadsExtended.RoadSign TID="x504">
          <Type>danger</Type>
          <Position>
            <COORD><C1>92.741</C1><C2>38.295</C2></COORD>
          </Position>
        </RoadsExdm2ien_10.RoadsExtended.RoadSign>
      </RoadsExdm2ien_10.RoadsExtended>
      <!-- end of basket REFHANDB00000001 -->
    </DATASECTION>
  </TRANSFER>
```

Description graphique RoadsExgm2ien_10

La représentation graphique du modèle de données est définie par la description graphique RoadsExgm2ien_10 suivante :

```
!! File RoadsExgm2ien_10.ili Release 2003-02-26

INTERLIS 2.2;

MODEL RoadsExgm2ien_10 (en) = !! Roads graphics
```

```
CONTRACT ISSUED BY Unknown;  !! Contractor(s) have to be defined!

IMPORTS RoadsExdm2ben_10;
IMPORTS RoadsExdm2ien_10;
IMPORTS StandardSymbology;

SIGN BASKET StandardSymbology ~ StandardSymbology.StandardSigns;


TOPIC Graphics =
  DEPENDS ON RoadsExdm2ben_10.Roads, RoadsExdm2ien_10.RoadsExtended;

  GRAPHIC Surface_Graphics
    BASED ON RoadsExdm2ien_10.RoadsExtended.LandCover =

    Building OF StandardSymbology.StandardSigns.SurfaceSign:
      WHERE Type == #building (
        Sign := {Building};
        Geometry := Geometry;
        Priority := 100);

    Street OF StandardSymbology.StandardSigns.SurfaceSign:
      WHERE Type == #street (
        Sign := {Street};
        Geometry := Geometry;
        Priority := 100);

    Water OF StandardSymbology.StandardSigns.SurfaceSign:
      WHERE Type == #water (
        Sign := {Water};
        Geometry := Geometry;
        Priority := 100);

    Other OF StandardSymbology.StandardSigns.SurfaceSign:
      WHERE Type == #other (
        Sign := {Other};
        Geometry := Geometry;
        Priority := 100);

  END Surface_Graphics;

  VIEW Surface_Boundary
    INSPECTION OF RoadsExdm2ien_10.RoadsExtended.LandCover -> Geometry;
    =
  ATTRIBUTE
    ALL OF LandCover;
  END Surface_Boundary;

  VIEW Surface_Boundary2
    INSPECTION OF Base ~ Surface_Boundary -> Lines;
    =
  ATTRIBUTE
    Geometry := Base -> Geometry;
    LineAttr := Base -> LineAttrs;
  END Surface_Boundary2;

  GRAPHIC SurfaceBoundary_Graphics
    BASED ON Surface_Boundary2 =

    Boundary OF StandardSymbology.StandardSigns.PolylineSign: (
      Sign := {continuous};
      Geometry := Geometry;
      Priority := 101);

  END SurfaceBoundary_Graphics;

  GRAPHIC Polyline_Graphics
```

```
        BASED ON RoadsExdm2ien_10.RoadsExtended.StreetAxis =

        Street_precise OF StandardSymbology.StandardSigns.PolylineSign:
          WHERE Precision == #precise (
            Sign := {continuous};
            Geometry := Geometry;
            Priority := 110);

        Street_unprecise OF StandardSymbology.StandardSigns.PolylineSign:
          WHERE Precision == #unprecise (
            Sign := {dotted};
            Geometry := Geometry;
            Priority := 110);

      END Polyline_Graphics;

      GRAPHIC Text_Graphics
        BASED ON RoadsExdm2ien_10.RoadsExtended.StreetNamePosition =

        StreetName OF StandardSymbology.StandardSigns.TextSign: (
            Sign := {Linefont_18};
            Txt := Street -> Name;
            Geometry := NamPos;
            Rotation := NamOri;
            Priority := 120);

      END Text_Graphics;

      GRAPHIC Point_Graphics
        BASED ON RoadsExdm2ien_10.RoadsExtended.RoadSign =

        Tree OF StandardSymbology.StandardSigns.SymbolSign:
          WHERE Type == #prohibition.noparking (
            Sign := {NoParking};
            Geometry := Position;
            Priority := 130);

        GP OF StandardSymbology.StandardSigns.SymbolSign:
          WHERE Type == #danger (
            Sign := {GP};
            Geometry := Position;
            Priority := 130);

      END Point_Graphics;

    END Graphics;

  END RoadsExgm2ien_10.
```

Le modèle graphique RoadsExgm2ien_10 se réfère à des signatures de la bibliothèque RoadsExgm2en_Symbols (fichier RoadsExgm2ien_Symbols.xml). La bibliothèque de signatures est décrite dans la section suivante.

Bibliothèque de signatures RoadsExgm2ien_Symbols.xml

La bibliothèque de signatures RoadsExgm2ien_Symbols (fichier RoadsExgm2ien_Symbols.xml) est décrite dans la suite avec la syntaxe XML. Elle contient des définitions de symboles pour les points fixes et les arbres, ainsi que des signatures de lignes, d'étiquettes et de surfaces. Le modèle de signatures correspondant est décrit dans l'annexe K Modèle de signatures (StandardSymbology).

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- File RoadsExgm2ien_symbols.xml 2003-04-14 -->

<TRANSFER xmlns="http://www.interlis.ch/INTERLIS2.2"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.interlis.ch/INTERLIS2.2
                            RoadsExgm2ien_Symbols.xsd">
  <HEADERSECTION VERSION="2.2" SENDER="V+D">
    <ALIAS>
      <ENTRIES FOR="RoadsExdm2ben">
        <TAGENTRY FROM="RoadsExdm2ben.Roads"
                  TO="RoadsExdm2ben.Roads"/>
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended"
                  TO="RoadsExdm2ben.Roads"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.LAttrs"
                  TO="RoadsExdm2ben.Roads.LAttrs"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.LandCover"
                  TO="RoadsExdm2ben.Roads.LandCover"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.Street"
                  TO="RoadsExdm2ben.Roads.Street"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.StreetAxis"
                  TO="RoadsExdm2ben.Roads.StreetAxis"/>
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended.StreetAxis"
                  TO="RoadsExdm2ben.Roads.StreetAxis"/>
        <DELENTRY TAG="RoadsExdm2ien.RoadsExtended.StreetAxis.Precision"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.StreetAxisAssoc"
                  TO="RoadsExdm2ben.Roads.StreetAxisAssoc"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.StreetNamePosition"
                  TO="RoadsExdm2ben.Roads.StreetNamePosition"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.StreetNamePositionAssoc"
                  TO="RoadsExdm2ben.Roads.StreetNamePositionAssoc"/>
        <TAGENTRY FROM="RoadsExdm2ben.Roads.RoadSign"
                  TO="RoadsExdm2ben.Roads.RoadSign"/>
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended.RoadSign"
                  TO="RoadsExdm2ben.Roads.RoadSign"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LAttrs.LArt"
                  FROM="welldefined" TO="welldefined"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LAttrs.LArt"
                  FROM="fuzzy" TO="fuzzy"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LandCover.Type"
                  FROM="building" TO="building"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LandCover.Type"
                  FROM="street" TO="street"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LandCover.Type"
                  FROM="water" TO="water"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.LandCover.Type"
                  FROM="other" TO="other"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.RoadSign.Type"
                  FROM="prohibition" TO="prohibition"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.RoadSign.Type"
                  FROM="indication" TO="indication"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.RoadSign.Type"
                  FROM="danger" TO="danger"/>
        <VALENTRY ATTR="RoadsExdm2ben.Roads.RoadSign.Type"
                  FROM="velocity" TO="velocity"/>
        <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
                  FROM="prohibition.noentry" TO="prohibition"/>
        <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
                  FROM="prohibition.noparking" TO="prohibition"/>
        <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
                  FROM="prohibition.other" TO="prohibition"/>
      </ENTRIES>

      <ENTRIES FOR="RoadsExdm2ien">
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended"
                  TO="RoadsExdm2ien.RoadsExtended"/>
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended.StreetAxis"
                  TO="RoadsExdm2ien.RoadsExtended.StreetAxis"/>
        <TAGENTRY FROM="RoadsExdm2ien.RoadsExtended.RoadSign"
                  TO="RoadsExdm2ien.RoadsExtended.RoadSign"/>
        <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
                  FROM="prohibition.noentry" TO="prohibition.noentry"/>
        <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
```

```
            FROM="prohibition.noparking" TO="prohibition.noparking"/>
  <VALENTRY ATTR="RoadsExdm2ien.RoadsExtended.RoadSign.Type"
            FROM="prohibition.other" TO="prohibition.other"/>
</ENTRIES>

<ENTRIES FOR="AbstractSymbology">
  <TAGENTRY FROM="AbstractSymbology.Signs"
            TO="AbstractSymbology.Signs"/>
  <TAGENTRY FROM="AbstractSymbology.Signs.TextSign"
            TO="AbstractSymbology.Signs.TextSign"/>
  <TAGENTRY FROM="AbstractSymbology.Signs.SymbolSign"
            TO="AbstractSymbology.Signs.SymbolSign"/>
  <TAGENTRY FROM="AbstractSymbology.Signs.PolylineSign"
            TO="AbstractSymbology.Signs.PolylineSign"/>
  <TAGENTRY FROM="AbstractSymbology.Signs.SurfaceSign"
            TO="AbstractSymbology.Signs.SurfaceSign"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.Color"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.PolylineAttrs"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.FontSymbol_Polyline"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.FontSymbol_Surface"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.FontSymbol"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.LineStyle_solid"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.DashRec"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.LineStyle_dashed"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.Pattern_Symbol"/>
  <DELENTRY TAG="StandardSymbology.StandardSigns.LineStyle_Pattern"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns"
            TO="AbstractSymbology.Signs"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.TextSign"
            TO="AbstractSymbology.Signs.TextSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.SymbolSign"
            TO="AbstractSymbology.Signs.SymbolSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.PolylineSign"
            TO="AbstractSymbology.Signs.PolylineSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.SurfaceSign"
            TO="AbstractSymbology.Signs.SurfaceSign"/>
</ENTRIES>

<ENTRIES FOR="StandardSymbology">
  <TAGENTRY FROM="StandardSymbology.StandardSigns.Color"
            TO="StandardSymbology.StandardSigns.Color"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.PolylineAttrs"
            TO="StandardSymbology.StandardSigns.PolylineAttrs"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.FontSymbol_Polyline"
            TO="StandardSymbology.StandardSigns.FontSymbol_Polyline"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.FontSymbol_Surface"
            TO="StandardSymbology.StandardSigns.FontSymbol_Surface"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.FontSymbol"
            TO="StandardSymbology.StandardSigns.FontSymbol"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.LineStyle_solid"
            TO="StandardSymbology.StandardSigns.LineStyle_solid"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.DashRec"
            TO="StandardSymbology.StandardSigns.DashRec"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.LineStyle_dashed"
            TO="StandardSymbology.StandardSigns.LineStyle_dashed"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.Pattern_Symbol"
            TO="StandardSymbology.StandardSigns.Pattern_Symbol"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.LineStyle_Pattern"
            TO="StandardSymbology.StandardSigns.LineStyle_Pattern"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns"
            TO="StandardSymbology.StandardSigns"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.TextSign"
            TO="StandardSymbology.StandardSigns.TextSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.SymbolSign"
            TO="StandardSymbology.StandardSigns.SymbolSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.PolylineSign"
            TO="StandardSymbology.StandardSigns.PolylineSign"/>
  <TAGENTRY FROM="StandardSymbology.StandardSigns.SurfaceSign"
```

```
                TO="StandardSymbology.StandardSigns.SurfaceSign"/>
      <VALENTRY ATTR="StandardSymbology.StandardSigns.Join"
                FROM="bevel" TO="bevel"/>
      <VALENTRY ATTR="StandardSymbology.StandardSigns.Join"
                FROM="round" TO="round"/>
      <VALENTRY ATTR="StandardSymbology.StandardSigns.Join"
                FROM="miter" TO="miter"/>
      <VALENTRY ATTR="StandardSymbology.StandardSigns.Caps"
                FROM="round" TO="round"/>
      <VALENTRY ATTR="StandardSymbology.StandardSigns.Caps"
                FROM="butt" TO="butt"/>
    </ENTRIES>
  </ALIAS>

  <COMMENT>
    example symbology dataset ili2 refmanual appendix C
  </COMMENT>
</HEADERSECTION>

<DATASECTION>
  <StandardSymbology.StandardSigns BID="xREFHANDB00000002">

    <!-- Color Library -->
    <StandardSymbology.StandardSigns.Color TID="x1">
      <Name>red</Name>
      <L>40.0</L>
      <C>70.0</C>
      <H>0.0</H>
      <T>1.0</T>
    </StandardSymbology.StandardSigns.Color>

    <StandardSymbology.StandardSigns.Color TID="x2">
      <Name>green</Name>
      <L>49.4</L>
      <C>48.5</C>
      <H>153.36</H>
      <T>1.0</T>
    </StandardSymbology.StandardSigns.Color>

    <StandardSymbology.StandardSigns.Color TID="x3">
      <Name>light_gray</Name>
      <L>75.0</L>
      <C>0.0</C>
      <H>0.0</H>
      <T>1.0</T>
    </StandardSymbology.StandardSigns.Color>

    <StandardSymbology.StandardSigns.Color TID="x4">
      <Name>dark_grey</Name>
      <L>25.0</L>
      <C>0.0</C>
      <H>0.0</H>
      <T>1.0</T>
    </StandardSymbology.StandardSigns.Color>

    <StandardSymbology.StandardSigns.Color TID="x5">
      <Name>dark_blue</Name>
      <L>50.3</L>
      <C>43.5</C>
      <H>261.1</H>
      <T>1.0</T>
    </StandardSymbology.StandardSigns.Color>

    <StandardSymbology.StandardSigns.Color TID="x6">
      <Name>black</Name>
      <L>0.0</L>
      <C>0.0</C>
      <H>0.0</H>
```

```
  <T>1.0</T>
</StandardSymbology.StandardSigns.Color>

<StandardSymbology.StandardSigns.Color TID="x7">
  <Name>white</Name>
  <L>100.0</L>
  <C>0.0</C>
  <H>0.0</H>
  <T>1.0</T>
</StandardSymbology.StandardSigns.Color>

<StandardSymbology.StandardSigns.PolylineAttrs TID="x4001">
  <Width>0.01</Width>
  <Join>round</Join>
  <Caps>butt</Caps>
</StandardSymbology.StandardSigns.PolylineAttrs>

<StandardSymbology.StandardSigns.PolylineAttrs TID="x4002">
  <Width>0.01</Width>
  <Join>round</Join>
  <MiterLimit>2.0</MiterLimit>
  <Caps>butt</Caps>
</StandardSymbology.StandardSigns.PolylineAttrs>

<!-- Font/Symbol Library -->
<StandardSymbology.StandardSigns.FontSymbol TID="x101">
  <Name>Triangle</Name>
  <Geometry>
    <StandardSymbology.StandardSigns.FontSymbol_Surface>
      <Geometry>
        <SURFACE>
          <BOUNDARY>
            <POLYLINE>
              <COORD><C1>-0.5</C1><C2>-0.5</C2></COORD>
              <COORD><C1>0.0</C1><C2>0.5</C2></COORD>
              <COORD><C1>0.5</C1><C2>-0.5</C2></COORD>
            </POLYLINE>
          </BOUNDARY>
        </SURFACE>
      </Geometry>
    </StandardSymbology.StandardSigns.FontSymbol_Surface>
    <StandardSymbology.StandardSigns.FontSymbol_Polyline>
      <Geometry>
            <POLYLINE>
              <COORD><C1>-0.5</C1><C2>0.0</C2></COORD>
              <ARC><C1>0.5</C1><C2>0.0</C2>
                 <A1>0.0</A1><A2>0.5</A2><R>0.5</R></ARC>
              <ARC><C1>-0.5</C1><C2>0.0</C2>
                 <A1>0.0</A1><A2>-0.5</A2><R>0.5</R></ARC>
            </POLYLINE>
      </Geometry>
    </StandardSymbology.StandardSigns.FontSymbol_Polyline>
  </Geometry>
  <Font REF="x10"></Font>
</StandardSymbology.StandardSigns.FontSymbol>

<StandardSymbology.StandardSigns.FontSymbol TID="x102">
  <Name>NoParking</Name>
  <Geometry>
    <StandardSymbology.StandardSigns.FontSymbol_Polyline>
      <Color REF="x6">
      </Color>
       <Geometry>
            <POLYLINE>
              <COORD><C1>-0.5</C1><C2>0.0</C2></COORD>
              <ARC><C1>0.5</C1><C2>0.0</C2>
                 <A1>0.0</A1><A2>0.5</A2><R>0.5</R></ARC>
              <ARC><C1>-0.5</C1><C2>0.0</C2>
```

```
            <A1>0.0</A1><A2>-0.5</A2><R>0.5</R></ARC>
        </POLYLINE>
  </Geometry>
</StandardSymbology.StandardSigns.FontSymbol_Polyline>
<StandardSymbology.StandardSigns.FontSymbol_Surface>
  <FillColor REF="x1">
  </FillColor>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
          <ARC><C1>0.325</C1><C2>-0.233</C2>
             <A1>0.283</A1><A2>0.283</A2><R>0.4</R></ARC>
          <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</StandardSymbology.StandardSigns.FontSymbol_Surface>
<StandardSymbology.StandardSigns.FontSymbol_Surface>
  <FillColor REF="x1">
  </FillColor>
  <Geometry>
    <SURFACE>
     <BOUNDARY>
        <POLYLINE>
         <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
          <ARC><C1>-0.327</C1><C2>0.238</C2>
             <A1>-0.283</A1><A2>-0.283</A2><R>0.4</R></ARC>
          <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
</StandardSymbology.StandardSigns.FontSymbol_Surface>
<StandardSymbology.StandardSigns.FontSymbol_Surface>
  <FillColor REF="x5">
  </FillColor>
  <Geometry>
    <SURFACE>
      <BOUNDARY>
        <POLYLINE>
          <COORD><C1>-0.5</C1><C2>0.0</C2></COORD>
          <ARC><C1>0.5</C1><C2>0.0</C2>
             <A1>0.0</A1><A2>0.5</A2><R>0.5</R></ARC>
          <ARC><C1>-0.5</C1><C2>0.0</C2>
             <A1>0.0</A1><A2>-0.5</A2><R>0.5</R></ARC>
        </POLYLINE>
      </BOUNDARY>
      <BOUNDARY>
        <POLYLINE>
          <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
          <ARC><C1>0.325</C1><C2>-0.233</C2>
             <A1>0.283</A1><A2>0.283</A2><R>0.4</R></ARC>
          <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
      <BOUNDARY>
        <POLYLINE>
         <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
          <ARC><C1>-0.327</C1><C2>0.238</C2>
             <A1>-0.283</A1><A2>-0.283</A2><R>0.4</R></ARC>
          <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
        </POLYLINE>
      </BOUNDARY>
    </SURFACE>
  </Geometry>
```

```
      </StandardSymbology.StandardSigns.FontSymbol_Surface>
      <StandardSymbology.StandardSigns.FontSymbol_Polyline>
        <Color REF="x7">
        </Color>
        <LineAttrs REF="x4001">
        </LineAttrs>
          <Geometry>
              <POLYLINE>
                <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
                <ARC><C1>0.325</C1><C2>-0.233</C2>
                   <A1>0.283</A1><A2>0.283</A2><R>0.4</R></ARC>
                <COORD><C1>-0.233</C1><C2>0.325</C2></COORD>
              </POLYLINE>
        </Geometry>
      </StandardSymbology.StandardSigns.FontSymbol_Polyline>
      <StandardSymbology.StandardSigns.FontSymbol_Polyline>
        <Color REF="x7">
        </Color>
        <LineAttrs REF="x4001">
        </LineAttrs>
         <Geometry>
              <POLYLINE>
               <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
                <ARC><C1>-0.327</C1><C2>0.238</C2>
                   <A1>-0.283</A1><A2>-0.283</A2><R>0.4</R></ARC>
                <COORD><C1>0.228</C1><C2>-0.324</C2></COORD>
              </POLYLINE>
        </Geometry>
      </StandardSymbology.StandardSigns.FontSymbol_Polyline>
    </Geometry>
    <Font REF="x10"></Font>
</StandardSymbology.StandardSigns.FontSymbol>

<!-- interner Symbolfont "Symbols" -->
<StandardSymbology.StandardSigns.Font TID="x10">
    <Name>Symbols</Name>
    <Internal>true</Internal>
    <Type>symbol</Type>
</StandardSymbology.StandardSigns.Font>

<!-- externer Textfont "Leroy" -->
<StandardSymbology.StandardSigns.Font TID="x11">
    <Name>Leroy</Name>
    <Internal>false</Internal>
    <Type>text</Type>
    <BottomBase>0.3</BottomBase>
</StandardSymbology.StandardSigns.Font>

<!-- LineStyles -->
<StandardSymbology.StandardSigns.LineStyle_Solid TID="x21">
    <Name>LineSolid_01</Name>
    <Color REF="x6"></Color>
    <LineAttrs REF="x4001"></LineAttrs>
</StandardSymbology.StandardSigns.LineStyle_Solid>

<StandardSymbology.StandardSigns.LineStyle_Dashed TID="x22">
    <Name>LineDashed_01</Name>
    <Dashes>
      <StandardSymbology.StandardSigns.DashRec>
        <DLength>0.1</DLength>
      </StandardSymbology.StandardSigns.DashRec>
      <StandardSymbology.StandardSigns.DashRec>
        <DLength>0.1</DLength>
      </StandardSymbology.StandardSigns.DashRec>
    </Dashes>
    <Color REF="x6"></Color>
    <LineAttrs REF="x4002"></LineAttrs>
</StandardSymbology.StandardSigns.LineStyle_Dashed>
```

```
      <!-- Text Signs -->
      <StandardSymbology.StandardSigns.TextSign TID="x1001">
        <Name>Linefont_18</Name>
        <Height>1.8</Height>
        <Font REF="x11"></Font>
      </StandardSymbology.StandardSigns.TextSign>

      <!-- Symbol Signs -->
      <StandardSymbology.StandardSigns.SymbolSign TID="x2001">
        <Name>GP</Name>
        <Scale>1.0</Scale>
        <Color REF="x2"></Color>
        <Symbol REF="x101"></Symbol>
      </StandardSymbology.StandardSigns.SymbolSign>

      <StandardSymbology.StandardSigns.SymbolSign TID="x2002">
        <Name>NoParking</Name>
        <Scale>1.0</Scale>
        <Symbol REF="x102"></Symbol>
      </StandardSymbology.StandardSigns.SymbolSign>

      <!-- Polyline Signs -->
      <StandardSymbology.StandardSigns.PolylineSign TID="x3001">
        <Name>continuous</Name>
        <Style REF="x21">
          <StandardSymbology.StandardSigns.PolylineSignLineStyleAssoc>
            <Offset>0.0</Offset>
          </StandardSymbology.StandardSigns.PolylineSignLineStyleAssoc>
        </Style>
      </StandardSymbology.StandardSigns.PolylineSign>

      <StandardSymbology.StandardSigns.PolylineSign TID="x3002">
        <Name>dotted</Name>
        <Style REF="x22">
          <StandardSymbology.StandardSigns.PolylineSignLineStyleAssoc>
            <Offset>0.0</Offset>
          </StandardSymbology.StandardSigns.PolylineSignLineStyleAssoc>
        </Style>
      </StandardSymbology.StandardSigns.PolylineSign>

      <!-- Surface Signs -->
      <StandardSymbology.StandardSigns.SurfaceSign TID="x5001">
        <Name>Building</Name>
        <FillColor REF="x4"></FillColor>
      </StandardSymbology.StandardSigns.SurfaceSign>

      <StandardSymbology.StandardSigns.SurfaceSign TID="x5002">
        <Name>Street</Name>
        <FillColor REF="x3"></FillColor>
      </StandardSymbology.StandardSigns.SurfaceSign>

      <StandardSymbology.StandardSigns.SurfaceSign TID="x5003">
        <Name>Water</Name>
        <FillColor REF="x5"></FillColor>
      </StandardSymbology.StandardSigns.SurfaceSign>

      <StandardSymbology.StandardSigns.SurfaceSign TID="x5005">
        <Name>Other</Name>
        <FillColor REF="x2"></FillColor>
      </StandardSymbology.StandardSigns.SurfaceSign>
    </StandardSymbology.StandardSigns>
    <!-- end of basket REFHANDB00000002 -->
  </DATASECTION>
</TRANSFER>
```

Représentation graphique de l'exemple

A partir du jeu de données RoadsExdm2ien (fichier RoadsExdm2ien.xml), des descriptions du modèle graphique RoadsExgm2ien_10 (fichier RoadsExgm2ien_10.ili) et de la bibliothèque de signatures RoadsExgm2ien_Symbols (fichier RoadsExgm2ien_Symbols.xml), un processeur graphique INTERLIS 2 produit la représentation suivante :

Figure 27 : Représentation réalisée à partir des descriptions graphiques et de données.

Annexe D (Information) Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Les mots réservés sont indiqués, ci-dessus, en majuscule (cf. paragraphe 2.2.7 Signes particuliers et mots réservés), les définitions de syntaxe sont indiqués en écriture normale (cf. paragraphe 2) et les définitions de syntaxe relatives au transfert sont indiqués en écriture cursive (cf. paragraphe 3). Les

numéros de pages indiqués en gras désignent l'endroit au sein du manuel de référence où la notion est décrite de la façon la plus détaillée.

Annexe E (Proposition d'extension) Création d'identificateurs d'objets (OID)

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Introduction

La disponibilité sans cesse croissante de géodonnées entraîne une demande de plus en plus forte de mise à jour ainsi que d'intégration dans diverses banques de données. A terme, cela pose le problème de la réglementation des *identificateurs d'objets* (OID) et justifie les démarches actuelles visant à leur homogénéisation. Un OID est un outil permettant d'identifier une instance objet de sa création jusqu'à sa disparition, même en cas de changement des valeurs de ses attributs. Au contraire des clés utilisateurs (cf. annexe F *Unicité de clés utilisateurs* du manuel de référence d'INTERLIS 2), l'OID est à considérer comme un attribut "muet" ou "transparent" pour l'utilisateur, généralement géré par l'intermédiaire de fonctions système.

Un OID doit être au moins univoque, unique et invariable au sein d'une communauté de transfert.

L'attribution et l'utilisation d'OID sont soumises aux exigences suivantes :

- Univocité, unicité et invariabilité (stabilité) indépendamment de la quantité de données.
- Indépendance par rapport aux produits matériels et logiciels.
- Indépendance par rapport à la plate-forme.
- Aussi bien utilisable sur les postes mono utilisateur que multi-utilisateurs ou sur des systèmes autonomes (sur le terrain, par exemple).
- Faible espace mémoire requis et optimisation possible.
- Implémentation simple.

D'autres exigences posées ne revêtent pas nécessairement un caractère technique, par exemple : une organisation aussi légère que possible, la conservation de l'attribution du contrôle (national), une utilisation compatible avec des systèmes plus anciens et une large acceptation par les producteurs de systèmes. Il s'agit là d'exigences particulièrement élevées et parfois contradictoires. Une exigence particulière prévoit qu'un OID puisse être attribué au moins dix millions de fois par un producteur de systèmes. Une autre condition impose qu'il ait une longueur fixe afin d'en faciliter la gestion (ce qui exclut d'autres méthodes connues telle qu'une désignation dite URI en préfixe). Un chiffre de contrôle n'est pas nécessaire : on suppose que des niveaux de communication inférieurs mettent des moyens correspondants à disposition.

En principe, l'unicité d'un OID s'appuie toujours sur un mécanisme central. Les deux extrêmes, soit l'attribution de tout OID par un service central d'une part, et un service totalement décentralisé d'autre part, générant les OID de manière autonome, conduisent tous deux à des résultats insatisfaisants. Un OID qui est par exemple assigné via un numéro de carte de réseau et un tampon temporel passe pour non praticable et non viable, d'autant plus que chaque appareil devrait posséder une carte réseau et qu'il n'est par ailleurs pas exclu que cette technologie puisse être dépassée dans les prochaines années par de nouveaux développements. Cette spécification a une longue histoire. Pendant plusieurs années, les

consultations, études et réunions à ce sujet se sont succédées. Une partie des documents nés de ce contexte peut être remise aux intéressés (commande sur www.interlis.ch ou info@interlis.ch).

Construction d'un identificateur d'objet (OID)

Un identificateur d'objet (OID) est constitué d'un préfixe et d'un suffixe et fait état d'une longueur totale de 16 positions alphanumériques dans sa forme représentable. L'OID est toujours traité comme une unité sur l'interface utilisateur ou vis-à-vis de l'utilisateur. Le domaine de valeurs STANDARDOID de l'OID du modèle INTERLIS correspond à cette définition. Toutefois, il ne définit que la longueur totale et non pas le détail de sa structure.

```
OIDDef = Prefix Postfix.
```

Préfixe

Le préfixe est généré par une instance centrale. Ainsi, l'unicité à l'intérieur d'une communauté d'échanges est garantie. Un nouveau préfixe est généralement nécessaire pour chaque conteneur (par exemple un processus de base de données qui gère des données d'un thème concret). Comme lieu de détermination du préfixe, on utilise le pays du processus engendrant le préfixe. Ce processus n'intervient pas forcément au même lieu que le système de production qui engendre tout l'OID.

Le préfixe est composé de 8 positions avec les caractères suivants.

```
Prefix = Letter { Letter | Digit }.  !! Suite de 8 signes
Letter = ( 'A' | .. | 'Z' | 'a' | .. | 'z' ).
Digit = ( '0' | '1' | .. | '9' ).
```

Un préfixe est défini comme une suite de lettres et de chiffres, où le premier caractère doit être une lettre (cf. aussi la construction de nom de tag (balise) XML ou le paragraphe 2.2.2 Nom dans INTERLIS 2 – Manuel de référence).

La règle selon laquelle les deux premières positions du préfixe sont fixées conformément aux prescriptions de la norme ISO 3166 s'applique. Pour les préfixes créés en Allemagne, Autriche ou Suisse, on inscrit par exemple à cet endroit les lettres "de", "oe" ou "ch". Pour engendrer un préfixe, il existe donc pour chaque position 62 solutions différentes au total (0..9 : ASCII 48 à 57 ; A..Z : ASCII 65 à 90 ; a..z : ASCII 97 à 122). La combinaison des 62 signes avec le nombre de positions donne un ensemble d'OID qui est vraisemblablement supérieur à ce qu'utilisaient la plupart des anciennes applications.

Suffixe

Un suffixe est géré par le producteur de données ou par le système de production lui-même. Il est constitué d'une succession de 8 signes. Par l'approche compatible ASCII et orientée colonne, on exige que les positions éventuellement "libres" soient occupées par des zéros ("0") (cf. le premier et le troisième exemple d'un OID ci-dessous). La plus petite valeur ordinale possible de la partie du suffixe est donc représentée comme "0000000000".

```
Postfix = { Letter | Digit }.  !! Suivi de 8 signes
```

Il convient au besoin de régler des restrictions supplémentaires de la partie préfixe ou suffixe dans le cadre de spécifications complémentaires.

Résumé et exemples d'application

OID	*Longueur*	*Importance*	*Observations*

OID	*Longueur*	*Importance*	*Observations*
Préfixe	2 + 6 car.	Reconnaissance du pays + une partie d'identification "globale" donnée une fois pour toute par une instance compétente et centrale	Identification unique valant dans le monde entier, telle que de (Allemagne), oe (Autriche), ch (Suisse), selon la norme ISO 3166. D'autres limitations sont à régler dans des spécifications complémentaires.
Suffixe	8 car.	Séquence (numérique ou alphanumérique) du système de production comme partie d'identification "locale"	D'autres limitations comme le tampon temporel avec numéro de séquence sont à régler dans des spécifications additionnelles.

Exemples

```
1234567812345678   Remarque
----------------   ---------

A000000000000000   Le plus petit OID possible en théorie

zwzzzzzzzzzzzzz   Le plus grand OID possible avec zw pour le Zimbabwe

deg5mQXX2000004a   OID choisi de façon arbitraire, pour l'Allemagne (de)

chgAAAAAAAAA0azD   OID choisi de façon arbitraire, pour la Suisse(ch)
```

Organisation

Une instance, par exemple fédérale, reconnue par la communauté d'échanges, entretient un service central pour la génération d'OID. Les producteurs de données peuvent y requérir un ou plusieurs préfixes via les canaux de communications appropriés. Ce service pourrait être accessible par Internet et relié par messagerie électronique. Un tel service peut être relativement sûr et protégé contre les abus.

C'est à l'implémentation du système source et cible d'utiliser les caractéristiques explicitement établies dans cette spécification et d'utiliser de manière appropriée un OID, par exemple pour un tri ou une optimisation interne. Une telle optimisation peut être atteinte par la gestion des préfixes à l'intérieur d'un système localisé à un seul endroit. De plus, les différents objets pourraient ne contenir que les suffixes et les relations aux préfixes généraux. D'autres économies peuvent être obtenues si le préfixe est mémorisé dans le système sous forme d'un nombre binaire.

Pour une utilisation dans le cadre d'un exercice:

- Les OID pour les conteneurs (Basket), le préfix OID "chB00000".
- Pour tous les autres OID, le préfix OID "ch100000".

Annexe F (Proposition d'extension) Unicité des clés utilisateurs

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Alternatives de modélisation

Lorsque l'on exige l'unicité des clés utilisateurs, la question se pose toujours de savoir dans quel cadre celle-ci s'applique. D'un point de vue purement technique, il est tout à fait évident que l'unicité doit pouvoir être garantie à l'intérieur d'un conteneur donné, puisque les autres conteneurs ne sont pas accessibles. Du point de vue de la modélisation, le conteneur n'a aucune signification, puisque rien ne mentionne sa délimitation.

Il convient d'abord de se demander si l'unicité doit vraiment être exigée dans un modèle de base. En prenant pour exemple la structure hiérarchique de l'administration (au niveau fédéral par exemple), on peut penser que l'unicité ne s'applique pas à l'ensemble, mais uniquement au modèle interne fédéral de données, par exemple.

Deux variantes proposant une solution pour ce problème de l'unicité des clés utilisateurs sont présentées:

- Variante Réglementation centrale.
- Variante Réglementation décentralisée (principe de délégation).

Variante Réglementation centrale

Une réglementation centrale est privilégiée si aucune autre considération n'intervient. Une instance centrale définit pour tous les objets d'une classe que la clé utilisateur donnée doit être unique sur l'ensemble du territoire. Ceci peut se faire avec des mesures d'organisation où tous les intéressés ont accès à la même base de données centrale.

```
TOPIC BienFonds =

  CLASS Parcelle =
    Numero: 1 .. 99999;
    Geometrie: AREA WITH (STRAIGHTS, ARCS) VERTEX CHLKoord
               WITHOUT OVERLAPS > 0.005;
  UNIQUE
    Numero;
  END Parcelle;

END BienFonds.
```

Souvent, l'instance centrale définit une partition du territoire pour laquelle tous les numéros de territoires attribués sont uniques. Les parcelles se trouvant à l'intérieur de ce territoire doivent être uniques. Ainsi la clé utilisateur doit être la combinaison du numéro du territoire et de celui de la parcelle :

```
  CLASS Parcelle =
    NumeroRegion: 1 .. 9999;
    Numero: 1 .. 99999;
    Geometrie: AREA WITH (STRAIGHTS, ARCS) VERTEX CHLKoord
      WITHOUT OVERLAPS > 0.005;
  UNIQUE
    NumeroRegion, Numero; !! Clé utilisateur
  END Parcelle;
```

Variante Réglementation décentralisée (principe de délégation)

Si des structures de données appropriées sont prévues dans un modèle de données, il est possible que des objets soient d'abord saisis dans des conteneurs plus petits (par exemple un conteneur par commune), ensuite regroupés sans problème au sein d'un conteneur plus vaste (par exemple un conteneur pour un canton).

Admettons que l'instance fédérale ait défini que les numéros de parcelles doivent comporter 5 chiffres, mais n'impose aucun cadre quant à leur unicité. Si un canton définit alors qu'ils doivent être uniques au niveau des communes, la modélisation suivante est possible :

```
MODEL Confederation =


  TOPIC BienFonds =

    CLASS Parcelle =
      Numero: 1 .. 99999;
      Geometrie: AREA WITH (STRAIGHTS, ARCS) VERTEX CHLKoord
                 WITHOUT OVERLAPS > 0.005;
    END Parcelle;

  END BienFonds;

END Confederation.



MODEL CantonA =

  IMPORTS Confederation;


  TOPIC StructureOrg =

    CLASS Commune =
      Nom: TEXT*30;
    UNIQUE
      Nom;
    END Commune;

  END StructureOrg;


  TOPIC BienFonds EXTENDS Confederation.BienFonds =
    DEPENDS ON StructureOrg;

    ASSOCIATION ParcelleCom =
      Commune (EXTERNAL) -<> StructureOrg.Commune;
      Parcelle -- Parcelle;
    END ParcelleCom;

    CONSTRAINT OF Parcelle =
      UNIQUE
        Numero, Commune;
    END;
```

```
    END BienFonds;

  END CantonA.
```

Les noms des communes doivent être uniques dans le cadre de tous les objets de la classe, conformément à la définition. Il est sans importance de savoir si cette condition est effectivement vérifiable du fait de la partition en conteneurs concrets. L'exigence s'applique, si l'on considère la situation d'un point de vue concret.

Pour fixer l'unicité du numéro de parcelle au sein d'une commune, une relation est établie entre la parcelle et la commune et il est exigé que la combinaison de la commune et du numéro soit univoque. Il est à nouveau sans importance de savoir si un conteneur englobe une partie d'une commune, une commune entière ou plusieurs communes. Considérée du point de vue de la modélisation, l'exigence s'applique.

Si l'on prend par exemple pour hypothèse qu'un système contient les parcelles d'une commune donnée, il est possible que l'on renonce à la relation à la commune pour les parcelles au sein du système et que l'on ne l'ajoute que dans le cas d'un transfert de données à d'autres systèmes.

Annexe G (Proposition d'extension) Définitions d'unités

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Le modèle des types

Le modèle des types ci-après fait la synthèse des unités les plus courantes. Il étend les unités directement définies par INTERLIS (cf. à ce sujet l'annexe A Le modèle de données interne d'INTERLIS).

```
!! File Units.ili Release 2003-03-18

INTERLIS 2.2;

TYPE MODEL Units (en) =

  UNIT
    !! abstract Units
    Area (ABSTRACT) = (INTERLIS.LENGTH*INTERLIS.LENGTH);
    Volume (ABSTRACT) = (INTERLIS.LENGTH*INTERLIS.LENGTH*INTERLIS.LENGTH);
    Velocity (ABSTRACT) = (INTERLIS.LENGTH/INTERLIS.TIME);
    Acceleration (ABSTRACT) = (Velocity/INTERLIS.TIME);
    Force (ABSTRACT) = (INTERLIS.MASS*INTERLIS.LENGTH/INTERLIS.TIME/INTERLIS.TIME);
    Pressure (ABSTRACT) = (Force/Area);
    Energy (ABSTRACT) = (Force*INTERLIS.LENGTH);
    Power (ABSTRACT) = (Energy/INTERLIS.TIME);
    Electric_Potential (ABSTRACT) = (Power/INTERLIS.ELECTRIC_CURRENT);
    Frequency (ABSTRACT) = (INTERLIS.DIMENSIONLESS/INTERLIS.TIME);

    Millimeter [mm] = 0.001 [INTERLIS.m];
    Centimeter [cm] = 0.01 [INTERLIS.m];
    Decimeter [dm] = 0.1 [INTERLIS.m];
    Kilometer [km] = 1000 [INTERLIS.m];

    Square_Meter [m2] EXTENDS Area = (INTERLIS.m*INTERLIS.m);
    Cubic_Meter [m3] EXTENDS Volume = (INTERLIS.m*INTERLIS.m*INTERLIS.m);

    Minute [min] = 60 [INTERLIS.s];
    Hour [h] = 60 [min];
    Day [d] = 24 [h];

    Kilometer_per_Hour [kmh] EXTENDS Velocity = (km/h);
    Meter_per_Second [ms] = 3.6 [kmh];
    Newton [N] EXTENDS Force = (INTERLIS.kg*INTERLIS.m/INTERLIS.s/INTERLIS.s);
    Pascal [Pa] EXTENDS Pressure = (N/m2);
    Joule [J] EXTENDS Energy = (N*INTERLIS.m);
    Watt [W] EXTENDS Power = (J/INTERLIS.s);
    Volt [V] EXTENDS Electric_Potential = (W/INTERLIS.A);

    Inch [in] = 2.54 [cm];
    Foot [ft] = 0.3048 [INTERLIS.m];
    Mile [mi] = 1.609344 [km];

    Are [a] = 100 [m2];
    Hectare [ha] = 100 [a];
    Square_Kilometer [km2] = 100 [ha];
    Acre [acre] = 4046.873 [m2];
```

```
      Liter [L] = 1 / 1000 [m3];
      US_Gallon [USgal] = 3.785412 [L];

      Angle_Degree = 180 / PI [INTERLIS.rad];
      Angle_Minute = 1 / 60 [Angle_Degree];
      Angle_Second = 1 / 60 [Angle_Minute];
      Angle_DMS = {Angle_Degree:Angle_Minute[0 .. 59]:Angle_Second[0 .. 59]}
                  CONTINUOUS;

      Gon = 200 / PI [INTERLIS.rad];

      Gram [g] = 1 / 1000 [INTERLIS.kg];
      Ton [t] = 1000 [INTERLIS.kg];
      Pound [lb] = 0.4535924 [INTERLIS.kg];

      Calorie [cal] = 4.1868 [J];
      Kilowatt_Hour [kWh] = 0.36E7 [J];

      Horsepower = 746 [W];

      Techn_Atmosphere [at] = 98066.5 [Pa];
      Atmosphere [atm] = 101325 [Pa];
      Bar [bar] = 10000 [Pa];
      Millimeter_Mercury [mmHg] = 133.3224 [Pa];
      Torr = 133.3224 [Pa]; !! Torr = [mmHg]

      Decibel [dB] = FUNCTION // 10**(dB/20) * 0.00002 // [Pa];

      Degree_Celsius [oC] = FUNCTION // oC+273.15 // [INTERLIS.K];
      Degree_Fahrenheit [oF] = FUNCTION // (oF+459.67)/1.8 // [INTERLIS.K];

      CountedObjects EXTENDS INTERLIS.DIMENSIONLESS;

      Hertz [Hz] EXTENDS Frequency = (CountedObjects/INTERLIS.s);
      KiloHertz [KHz] = 1000 [Hz];
      MegaHertz [MHz] = 1000 [KHz];

      Percent = 0.01 [CountedObjects];
      Permille = 0.001 [CountedObjects];

      !! ISO 4217 Currency Abbreviation
      USDollar [USD] EXTENDS INTERLIS.MONEY;
      Euro [EUR] EXTENDS INTERLIS.MONEY;
      SwissFrancs [CHF] EXTENDS INTERLIS.MONEY;

  END Units.
```

Exemples

Cf. paragraphe 2.9.1 Unités de base dans le manuel de référence INTERLIS 2.

Annexe H (Proposition d'extension) Définitions temporelles

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Le modèle temporel

```
!! File Time.ili Release 2003-03-18

INTERLIS 2.2;

REFSYSTEM MODEL Time (en) =  !! en = 2-letter code (ISO 639)

  CONTRACT ISSUED BY Unknown;  !! Contractor(s) have to be defined!

  IMPORTS Units;

  UNIT
    HM = {Units.h : Units.min[0 .. 59]} CONTINUOUS;
    HMS = {Units.h : Units.min[0 .. 59] : INTERLIS.s[0 .. 59]} CONTINUOUS;

    Year [Y] EXTENDS INTERLIS.TIME;
    Month [M] EXTENDS INTERLIS.TIME;

    DayOfYear [MD] = {M:Units.d[1 .. 31]};
    Date [YMD] = {Y:M[1 .. 12]:Units.d[1 .. 31]};

  DOMAIN
    Year = 1582 .. 2999 [Y]; !! Gregorian Calendar
    Date = 1582:10:15 .. 2999:12:31 [YMD];
    DayOfYear = 1:1 .. 12:31 [MD];

    WeekDay = (WorkingDay (Monday, Tuesday, Wednesday,
              Thursday, Friday, Saturday), Sunday);

    HMOfDay = 0:0 .. 23:59 [HM];
    DifferenceToUTC = MANDATORY -13:00 .. 13:00 [HM]; !! UTC := LocTime + Diff

  FUNCTION AppropriateDate (dayOfYear: MANDATORY DayOfYear;
                            weekDay: WeekDay): DayOfYear
    // returns first parameter if second is undefined
       returns first day from (incl) first parameter being the
       requested weekday //;

  FUNCTION DSTOrdered (day1: DayOfYear; day2: DayOfYear) : BOOLEAN
    // returns TRUE if the second parameter comes after the
       first parameter or if both parameters are equal //;

  STRUCTURE DSTransition =
    TransitionDSTime: MANDATORY HMOfDay;
    FirstDate: MANDATORY DayOfYear;
    DayOfWeek: WeekDay;
    TransitionDate: DayOfYear := AppropriateDate (FirstDate, DayOfWeek);
  END DSTransition;

  STRUCTURE DaylightSavingPeriod =
    DSToUTC: DifferenceToUTC;
    From: MANDATORY Year;
```

```
      To:   MANDATORY Year;
      DSStart: MANDATORY DSTransition;
      DSEnd: MANDATORY DSTransition;
    MANDATORY CONSTRAINT
      DSTOrdered (DSStart, DSEnd);
    MANDATORY CONSTRAINT
      To >= From;
    END DaylightSavingPeriod;

    FUNCTION DSPOverlaps (periods: BAG {1..*} OF DaylightSavingPeriod) : BOOLEAN
      // returns TRUE if any one of the periods overlap //;


    TOPIC TimeZone =

      CLASS TimeZone (ABSTRACT) EXTENDS INTERLIS.SCALSYSTEM =
      PARAMETER
        Unit (EXTENDED): NUMERIC [INTERLIS.TIME];
      END TimeZone;

      CLASS BaseTimeZone EXTENDS TimeZone = !! TimeZone without daylight saving
        DiffToUTC: DifferenceToUTC;
      END BaseTimeZone;

      CLASS DaylightSavingTZ EXTENDS TimeZone =
        Periods: BAG {1..*} OF DaylightSavingPeriod;
      MANDATORY CONSTRAINT
        NOT ( DSPOverlaps (Periods) );
      END DaylightSavingTZ;

      ASSOCIATION DaylightSavingTZOf =
        BaseTZ -<> BaseTimeZone;
        DSTZ -- DaylightSavingTZ;
      END DaylightSavingTZOf;

    END TimeZone;

  END Time.
```

Exemples de données relatives au modèle temporel

L'exemple ci-dessous est adapté au modèle temporel précédent.

```
  <?xml version="1.0" encoding="UTF-8"?>

  <!-- File SwissTimeData.xml Release 2003-03-18 -->

  <TRANSFER xmlns="http://www.interlis.ch/INTERLIS2.2"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.interlis.ch/INTERLIS2.2
                                SwissTimeData.xsd">
    <HEADERSECTION VERSION="2.2" SENDER="V+D">
      <ALIAS>
        <ENTRIES FOR="Time">
          <TAGENTRY FROM="Time.DayOfYear" TO="Time.DayOfYear"/>
          <TAGENTRY FROM="Time.HMDiffWithinDay" TO="Time.HMDiffWithinDay"/>
          <TAGENTRY FROM="Time.DSTransition" TO="Time.DSTransition"/>
          <TAGENTRY FROM="Time.DaylightSavingPeriod" TO="Time.DaylightSavingPeriod"/>
          <TAGENTRY FROM="Time.TimeZone" TO="Time.TimeZone"/>
          <TAGENTRY FROM="Time.TimeZone.BaseTimeZone"
                    TO="Time.TimeZone.BaseTimeZone"/>
          <TAGENTRY FROM="Time.TimeZone.DaylightSavingTZ"
                    TO="Time.TimeZone.DaylightSavingTZ"/>
          <TAGENTRY FROM="Time.TimeZone.DaylightSavingTZOf"
                    TO="Time.TimeZone.DaylightSavingTZOf"/>
        </ENTRIES>
      </ALIAS>
```

```
      <COMMENT>
        example dataset ili2 refmanual appendix H
      </COMMENT>
    </HEADERSECTION>

    <DATASECTION>
      <Time.TimeZone BID="xBTimeZones">
        <Time.TimeZone.BaseTimeZone TID="xBTimeZonesMEZ">
          <Name>MEZ</Name>
          <DiffToUTC>-1:00</DiffToUTC>
        </Time.TimeZone.BaseTimeZone>

        <Time.TimeZone.DaylightSavingTZ TID="xBTimeZonesMESZ">
          <Name>MESZ</Name>
          <Periods>
            <Time.DaylightSavingPeriod>
              <DSToUTC>-2:00</DSToUTC>
              <From>1983</From>
              <To>1995</To>
              <DSStart>
                <Time.DSTransition>
                  <TransitionDSTime>3:00</TransitionDSTime>
                  <FirstDate>3:25</FirstDate>
                  <DayOfWeek>Sunday</DayOfWeek>
                </Time.DSTransition>
              </DSStart>
              <DSEnd>
                <Time.DSTransition>
                  <TransitionDSTime>3:00</TransitionDSTime>
                  <FirstDate>9:24</FirstDate>
                  <DayOfWeek>Sunday</DayOfWeek>
                </Time.DSTransition>
              </DSEnd>
            </Time.DaylightSavingPeriod>
            <Time.DaylightSavingPeriod>
              <DSToUTC>-2:00</DSToUTC>
              <From>1996</From>
              <To>2999</To>
              <DSStart>
                <Time.DSTransition>
                  <TransitionDSTime>3:00</TransitionDSTime>
                  <FirstDate>3:25</FirstDate>
                  <DayOfWeek>Sunday</DayOfWeek>
                </Time.DSTransition>
              </DSStart>
              <DSEnd>
                <Time.DSTransition>
                  <TransitionDSTime>3:00</TransitionDSTime>
                  <FirstDate>10:25</FirstDate>
                  <DayOfWeek>Sunday</DayOfWeek>
                </Time.DSTransition>
              </DSEnd>
            </Time.DaylightSavingPeriod>
          </Periods>
        </Time.TimeZone.DaylightSavingTZ>

        <Time.TimeZone.DaylightSavingTZOf TID="xDaylightSavingTZOf">
          <BaseTZ REF="xBTimeZonesMEZ"></BaseTZ>
          <DSTZ REF="xBTimeZonesMESZ"></DSTZ>
        </Time.TimeZone.DaylightSavingTZOf>
      </Time.TimeZone>
    </DATASECTION>
  </TRANSFER>
```

Exemples d’application

Le paragraphe 2.9 Unités du manuel de référence d'INTERLIS 2 fournit des exemples de base pour les indications temporelles.

Voici deux exemples d'application se basant sur ce modèle temporel et les données XML précédentes :

Exemple 1 : la définition de domaine suivante décrit un type temporel basé sur l'heure de l'Europe centrale (MEZ), sans prise en compte de l'heure d'été d'un pays.

```
DOMAIN
  Heure = 0:00:00.000 .. 23:59:59.999 [MEZ];
```

Exemple 2 : la définition suivante décrit un type temporel basé sur l'heure de l'Europe centrale (MEZ) et incluant l'heure d'été de la Suisse.

```
DOMAIN
  Heure = 0:00:00.000 .. 23:59:59.999 [MESZ];
```

Annexe I (Proposition d'extension) Définition des couleurs

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Introduction

Cette spécification explique en détail pourquoi un certain espace chromatique appelé $L^*C^*_{ab}h^*_{ab}$ est le mieux adapté pour les définitions de couleurs. Elle donne une description exhaustive de cet espace chromatique, cite des formules de conversion entre les autres espaces chromatiques et donne des instructions sur la manière d'implémenter une transformation de coordonnées de $L^*C^*_{ab}h^*_{ab}$ dans le système de coordonnées des couleurs d'un écran ou d'une imprimante. De plus, elle motive le choix des domaines de valeurs et des précisions et indique des coordonnées pour des exemples sélectionnés.

Les couleurs permettant, entre autres, la description de représentations graphiques par INTERLIS 2, leur spécification doit être rendue possible. Une définition indépendante de tout système et de tout appareillage se révèle toutefois être d'une surprenante complexité et suppose par ailleurs la maîtrise de notions généralement étrangères aux non-spécialistes.

Une couleur est produite par la lumière (= stimuli de couleur), les yeux (=composante chromatique) et le cerveau (=information sensorielle). Il n'est en fait pas possible de décrire une couleur de façon chiffrée de manière à ce que deux personnes comprennent la même chose. Toutefois les couleurs peuvent être mesurées d'une manière universellement acceptée, permettant une compréhension précise entre spécialistes.

La spécification des couleurs sous forme de chaîne de caractères doit satisfaire à plusieurs critères :

- **Indépendance du matériel** — La couleur correspondant effectivement à une indication donnée doit être clairement définie. C'est uniquement ainsi que l'on pourra assurer que le résultat satisfait aux attentes exprimées indépendamment du matériel utilisé.
- **Expressivité** — Il doit être possible de spécifier toutes les couleurs que les matériels (hardware) "normaux" (en particulier des imprimantes et plotters de bonne qualité) sont capables de représenter. Le spectre des couleurs à représenter devrait être aussi large que possible. Idéalement, il devrait comprendre toutes les couleurs qu'un être humain peut percevoir.
- **Intelligibilité intuitive** — En lisant une description d'une couleur, une personne devrait intuitivement avoir la notion de la couleur décrite. Un modèle INTERLIS a toujours un certain caractère documentaire et devrait être compréhensible par les personnes concernées sans effort majeur.
- **Indépendance du système** — Les méthodes de représentation des couleurs devraient être indépendantes des systèmes (SIG, système d'exploitation, matériel) et ne nécessiter l'acquisition d'aucun appareil spécial.

Espace chromatique

Le tableau suivant montre l'adéquation d'espaces chromatiques pour une application dans des représentations graphiques d'INTERLIS :

Espace chromatique	Indépendance-matérielle	Expressivité	Intelligibilité intuitive	Indépendance du système
RGB	–	–	–	–
HLS	–	–	+ +	–
HSV	–	–	+ +	–
CMY(K)	–	–	–	–
XYZ	+	+	– –	+
SRGB	+	–	–	–
L a* b**	+	+	+	+
L C*ab h*ab*	+	+	+ +	+

Figure I.1 : Adéquation des espaces chromatiques pour INTERLIS.

Le dernier espace chromatique mentionné sur la figure I.1 *L* C*ab h*ab* (c.-à-d. *L*a*b** avec des coordonnées polaires) remplit au mieux toutes les conditions mentionnées plus haut.

L*a*b*

L'espace chromatique *L*a*b** (parfois appelé CIELAB) largement utilisé par l'industrie graphique, peut être dérivé par transformation à partir de XYZ comme le montre la figure I.2.

$$L^* = 116 \cdot f\left(\frac{Y}{Y_n}\right) - 16$$

$$a^* = 500 \cdot \left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right]$$

$$b^* = 200 \cdot \left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right]$$

, où $f(x) = \begin{cases} \sqrt[3]{x} & \text{si } x > 0.008856; \\ 7.787x + \frac{16}{116} & \text{autrement} \end{cases}$

Figure I.2 : Conversion de XYZ en L*a*b*.

Dans le calcul de la figure I.2 un "blanc de référence" est introduit au moyen de ‹X_n, Y_n, Z_n› afin de compenser une éventuelle teinte de lumière. Très souvent, les valeurs CIE des sources standard de lumière (principalement D50, occasionnellement D65) sont employées. Les coordonnées XYZ de ces sources de lumière peuvent être trouvées par exemple dans [Sangwine/Horne, 1989], Tableau 3.1.

Cet espace possède bon nombre de propriétés utiles :

- **Indépendance matérielle** — L*a*b* est dérivé du système XYZ et c'est pourquoi il est indépendant de tout matériel. La couleur associée à un triplet L*a*b* est définie de manière univoque.
- **Expressivité** — Chaque couleur qui peut être émise par une surface réfléchissante est associée à un point dans L*a*b*.
- **Intelligibilité intuitive** — *L** est la luminosité, une surface totalement noire (ne reflétant pas du tout la lumière) possédant un *L** nul et un réflecteur parfait (reflétant la totalité de la lumière) un *L**

de 100. Un observateur humain jugera qu'une couleur pour laquelle L* = 50 est de luminosité moyenne. a* est l'axe rouge-vert : une couleur avec a* = 0 est interprétée comme n'ayant aucune composante rouge et aucune composante verte, une couleur avec un a* négatif est verte, une couleur avec un a* positif est rouge. De manière analogue, b* correspond à l'axe bleu-jaune. Sur un plan défini par les axes a* et b*, plus la distance entre le point origine des axes et la valeur de couleur est grande, plus la couleur devient saturée.

- **Indépendance du système** — L*a*b* est indépendant de tout système ; étant un standard international, l'espace chromatique est indépendant d'une quelconque firme.
- **Utilisation croissante**— L'utilisation de L*a*b* dans les imprimeries professionnelles est très répandue. Des programmes comme Adobe Photoshop ou Acrobat (PDF) recourent à l'espace chromatique L*a*b*.
- **Transformation aisée en RGB** — Les triplets L*a*b* peuvent être transformés en valeurs RGB de n'importe quel écran par la multiplication avec une matrice 3x3, suivie par l'élévation à une puissance supérieure (correction gamma), laquelle peut être efficacement réalisée au moyen d'une table (cf. [Adobe, 1992], chapitre 23). Ainsi, les efforts des développeurs systèmes seront réduits au minimum.
- **Bonne compressibilité** — Indiquons en passant, que L*a*b* est mieux adapté que la norme RGB pour certains processus de compression d'image impliquant des pertes d'information. Cependant du point de vue d'INTERLIS, c'est insignifiant.

$$C^*_{ab} = \sqrt{\left(a^*\right)^2 + \left(b^*\right)^2} \qquad h^*_{ab} = \tan^{-1}\left(\frac{b^*}{a^*}\right)$$

Figure I.3 : Conversion de l'espace cartésien L*a*b vers la forme polaire L*$C^*_{ab}h^*_{ab}$ (d'après [Sangwine/Horne, 1998]).

L*$C^*_{ab}h^*_{ab}$

Comme décrit ci-dessus, chaque axe *L**(foncé — clair), *a**(vert — rouge), *b**(bleu — jaune) de l'espace L*a*b* correspond à des propriétés chromatiques percevables.

Toutefois, l'intelligibilité intuitive peut être encore accentuée en indiquant les coordonnées chromatiques dans un système polaire au lien d'un système cartésien (cf. Figure I.4).

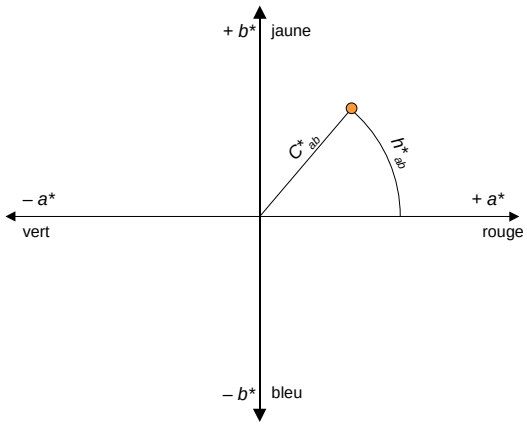


Figure I.4 : L'espace chromatique L*$C^*_{ab}h^*_{ab}$ fonctionne avec des coordonnées polaires basées sur L*a*b*.

La formule de la figure I.3 pour h^*_{ab} n'est applicable que pour des valeurs positives a* et b*, une version corrigée devrait prendre en considération les différences de chaque quadrant. Ce système polaire combine l'intelligibilité intuitive de HLS et HSV avec les nombreux avantages de L*a*b* décrits plus haut, puisqu'il signifie que les axes L* (*luminosité*), C* (*chroma*) et h* (nuance, en anglais *hue*) deviennent disponibles séparément.

Lorsque des indications précises sur les couleurs sont souhaitées dans des modèles INTERLIS, elles devraient être faites sur la base de ce système de coordonnées chromatiques.

Précision exigée

Cela fait partie d'un modèle INTERLIS que d'indiquer le degré de précision à appliquer pour l'enregistrement de valeurs numériques. L'espace L*a*b* est défini de telle manière que la différence entre deux couleurs est à peine perceptible, si la valeur calculée, comme illustré à la figure I.5, est égale à 1.

Remarque : [Has/Newman, o.D.] a mis en évidence le fait que la perception des différences chromatiques dépend aussi de la quantité de temps à disposition pour la comparaison. L'article relate une expérience où le temps nécessaire pour remarquer des différences était mesuré dans le cas d'un observateur inexpérimenté. Les résultats mentionnés sont 5 secondes pour ΔE_{ab} = 15, 10 secondes pour ΔE_{ab} = 10 et 15 secondes pour ΔE_{ab} = 5.

$$\Delta E_{ab} = \sqrt{\Delta L^{*2} + \Delta a^{*2} + \Delta b^{*2}}$$

Figure I.5 : Calcul de différences chromatiques dans un espace cartésien L*a*b*.

Précision de l'axe L* : pour la luminosité, une précision d'une décimale est suffisante.

Précision des axes C^*_{ab} et h^*_{ab} : théoriquement a* et b* peuvent être librement choisis, mais dans la pratique un arrondi de ±128 des nombres entiers est considéré comme largement suffisant (cf. [Adobe, 1992]). Ainsi, quel degré de précision est nécessaire pour C^*_{ab} et h^*_{ab} afin d'assurer que l'imprécision dans le plan a*/b* n'excède pas 1 ?

L'imprécision introduite par l'indication angulaire augmente avec l'accroissement de la distance au point zéro. Ainsi, la précision peut encore être considérée comme étant suffisante tant que ‹127, 128› et ‹128, 128› peuvent être distingués dans le plan a*/b*. Comme illustré sur la figure I.6, une décimale peut suffire dans ce cas extrême. Il s'agit de deux nuances d'orange à peine distinguables, toutefois saturées à un tel degré qu'il paraît improbable qu'un appareil puisse les reproduire.

a^*	b^*	C^*_{ab}	h^*_{ab}
127	128	180.3	45.2
128	128	181.0	45.0

Figure I.6 : Coordonnées cartésiennes et polaires d'une couleur extrêmement éloignée du point d'origine (conversion voir figure I.3).

Combinaison avec des noms

Les noms de couleurs sont plus faciles à manipuler que leurs codes (c.-à-d. des nombres), mais présentent toutefois le désavantage de ne mettre qu'un nombre limité de couleurs à disposition. Dans

INTERLIS des noms peuvent être combinés avec une spécification numérique, permettant à l'utilisateur de définir ses propres noms de couleurs et de les échanger par les moyens usuels.

Grâce à cette définition, il est également possible d'employer INTERLIS — si besoin est — dans la documentation et l'utilisation de systèmes de noms de couleurs existants ou catalogues d'exemples de couleurs, tels que le système Pantone ou HKS.

Ceci demande la création d'une méta-classe (cf. paragraphe 1.4.3 dans le manuel de référence d'INTERLIS 2). Ses instances, appelées méta-objets, sont contenues dans un fichier de transfert spécial et sont lues par le compilateur d'INTERLIS 2. Elles sont disponibles pour les modèles de données INTERLIS et peuvent ainsi être utilisées dans des définitions graphiques afin de déterminer la couleur de certains symboles, par exemple.

Exemple d'application dans des modèles INTERLIS

```
!! Partie du modèle de signatures

SYMBOLOGY MODEL SymbologyExample EXTENDS BasicSymbology =

  CONTRACT ISSUED BY Unknown; !! Contractor(s) have to be defined!


TOPIC Signs =

  CLASS LChColor EXTENDS INTERLIS.METAOBJECT =
    !! Attribut "Nom" herite de INTERLIS.METAOBJECT
    Luminance = MANDATORY 0.0 .. 100.0;
    Chroma = MANDATORY 0.0 .. 181.1;
    Hue = MANDATORY 0.0 .. 359.9 CIRCULAR [DEGREE] COUNTERCLOCKWISE;
  END  LChColor;

    ...

  !! Partie de la definition des classes de signatures dans modele de signatures
  CLASS CouleurSignature EXTENDS SIGN =
    ...
  PARAMETER
    Couleur: -> METAOBJECT OF SymbologyExample.LChColor;
  END CouleurSignature;

  ...

END Signs;
...
```

Dans une instruction graphique (ici PointSimpleGr) définie par l'utilisateur, la couleur d'une signature peut par exemple se présenter comme suit (cf. paragraphe 2.16 Représentations graphiques du manuel de référence d'INTERLIS 2) :

```
...
MODEL GraphiqueSimple =

CONTRACT ISSUED BY Unknown;

IMPORTS SymbologyExample, Donnees;

SIGN BASKET SimpleSigns ~ SymbologyExample.Signs


  TOPIC CouleurPointGraphique =
    DEPENDS ON Donnes.Points;

    GRAPHIC PointSimpleGr BASED ON Données.Points.Point =
```

```
        Symbol OF SymbologyExample.Signs. CouleurSignature: (
          Sign := { Point };
          Pos := Position;
          Couleur := Brun
           );
      END PointSimpleGr;

    END CouleurPointGraphique;

  END GraphiqueSimple.
  ...
```

Nous n'avons pas voulu donner un exemple complet, ni représenter la métatable nécessaire et vous invitons à vous reporter à l'exemple figurant à l'annexe C *Le petit exemple Roads* du manuel de référence d'INTERLIS 2.

Exemple de valeurs

La figure I.7 énumère quelques couleurs avec leurs coordonnées respectives. Comme il n'est pas sûr que ce document ait été écrit sur un système (et ait été imprimé sur un périphérique) en mesure de reproduire correctement les couleurs, il nous faut malheureusement renoncer ici à représenter les couleurs correspondantes "en couleur".

Nom	L^*	a^*	b^*	C^*_{ab}	h^*_{ab}
Noir	0.0	0	0	0.0	0.0
Gris foncé	25.0	0	0	0.0	0.0
Gris moyen	50.0	0	0	0.0	0.0
Gris clair	75.0	0	0	0.0	0.0
Blanc	100.0	0	0	0.0	0.0
Fuchsia	40.0	70	0	70.0	0.0
Bleu clair	80.0	0	-30	30	270.0
Jaune foncé	90.0	0	100	100.0	90.0
Brun	50.0	30	50	58.3	59.0
Lilas	50.0	50	-50	70.7	315.0

Figure I.7 : Coordonnées cartésiennes et polaires de quelques couleurs.

Un exemple concret d'application avec des définitions de couleurs se trouve à l'annexe C *Le petit exemple Roads*.

Remarques pour les développeurs

Les développeurs de systèmes conformes à INTERLIS doivent composer avec la question suivante : comment transformer les valeurs chromatiques du système indépendant L^* C^*_{ab} h^*_{ab} dans un système de coordonnées chromatiques d'un écran ou d'une imprimante spécifique ?

Un format de fichier standardisé vous permettra d'enregistrer les distorsions chromatiques d'un appareil donné dans ce qu'on appelle des profils de correspondance d'appareil ou de couleur (appelé ICC-Profil-Format). Entre autres informations, ces fichiers contiennent les paramètres nécessaires à la conversion

d'un espace chromatique indépendant dans un système chromatique spécifique à un appareil. Pour les premiers nommés, il s'agit de *XYZ* ou de *L*a*b**, de *RGB* ou de *CMYK* pour les derniers nommés. Le format et les fonctions de conversion nécessaires sont définis dans [ICC, 1996].

Ainsi, un développeur sera capable d'accepter directement les profils ICC dans son produit. Le format du fichier est relativement simple et les fonctions de conversion sont facilement implémentables. Pour quelques plateformes, des bibliothèques de programmes déjà prêtes (telles que *Apple ColorSync* ou *Kodak KCMS*) sont disponibles.

Dans ce contexte, nous voudrions attirer l'attention du lecteur sur le fait que PDF accepte directement l'espace chromatique L*a*b. PostScript vous permet même de définir vos propres domaines de couleurs en termes de n'importe quelle transformation donnée à partir de XYZ. La fonction inverse de la formule présentée sur la figure I.2 peut être trouvée à l'exemple 4.11 dans [Adobe, 1990]. Il est relativement simple de programmer une fonction analogue en PostScript qui accepterait directement $L^* C^*_{ab} h^*_{ab}$.

Bibliographie

[Adobe, 1990] Adobe Systems: PostScript Language Reference Manual. 2nd Ed., 1990. ISBN 0-201-18127-4. 764 pages.

*Le manuel de référence du PostScript fournit également des recommandations pour le traitement des couleurs et des différentes méthodes de conversion disponibles en PostScript. L'exemple 4.11 à la page 191 définit l'espace chromatique L*a*b* en PostScript.*

[Adobe, 1992] Adobe Developers Association : TIFF Revision 6.0. http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf

*Le chapitre 23 définit une variante du format TIFF pour les images avec l'espace chromatique L*a*b* et énumère un nombre d'avantages en comparaison avec RGB. De plus, vous trouverez la description d'une méthode rapide pour la conversion de L*a*b* vers RGB.*

[Apple, 1998] Apple Computer, Inc.: Introduction to Color and Color Management Systems. In: *Inside Macintosh — Managing Color with ColorSync.* http://developer.apple.com/techpubs/macosx/Carbon/graphics/ColorSyncManager/ManagingColorSync/index.html

Introduction facile à comprendre aux différents espaces chromatiques, illustrés avec des graphiques.

[Apple, o.D.] Apple Computer, Inc.: A Brief Overwiew Of Color. www.apple.com/colorsync/benefits/training/overview.html

Introduction concise, facile à comprendre aux différents concepts en relation avec les couleurs. Pour les non-spécialistes.

[Has/Newman, o.D.] Michael Has, Todd Newman: Color Management: Current Practice and The Adoption of a New Standard. www.color.org/wpaper1.html

Nomme les mesures pour les points de référence rouge, vert et bleu de deux écrans standard d'ordinateur et montre qu'elles diffèrent largement des valeurs xy du standard de couleurs NTSC souvent annoncées. Donne une transformation de XYZ vers RGB d'un certain écran.

[ICC, 1996] International Color Consortium: ICC Profile Format Specification. www.color.org/profiles.html

Définit un format de fichier pour la caractérisation de n'importe quel appareil donné en rapport avec sa représentation chromatique. L'annexe A commente différents espaces chromatiques.

[Poynton, 1997] Charles A. Poynton: Frequently Asked Questions about Color. ftp://ftp.inforamp.net/pub/users/poynton/doc/colour/ColorFAQ.pdf

Explique au paragraphe 36, pourquoi HLS et HSV ne sont pas appropriés pour la spécification des couleurs.

[Sangwine/Horne, 1998] Sangwine, Stephen J. et Horne, Robin E. N. [Hrsg.]: The Colour Imaging Processing Handbook. Chapman & Hall: London […], 1998. ISBN 0-412-80620-7. 440 pages.

Introduction fouillée sur les fondements scientifiques de la perception des couleurs et son application au traitement d'images.

[Stokes et al., 1996] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar und Ricardo Motta: A Standard Defalut Color Space for the Internet — sRGB. November 1996. www.color.org/sRGB
Spécifications de sRGB.

Annexe J (Proposition d'extension) Systèmes de références et systèmes de coordonnées

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Introduction

Les coordonnées décrivent la position d'un point dans l'espace, pour autant qu'un système de coordonnées ait été défini. Un système de coordonnées positionné de manière fixe par rapport à la Terre est appelé un système de coordonnées de référence. Toutefois, les coordonnées ne déterminent pas seulement la position, mais aussi les quantités métriques qui peuvent être déduites des coordonnées, telles que les distances, surfaces, volumes, angles et directions, ainsi que d'autres propriétés comme les pentes et les courbures.

Il existe de nombreuses classes (types) de systèmes de coordonnées et un plus grand nombre encore d'objets, c.-à-d. de réalisations (instances) de systèmes de coordonnées (cf. par exemple [Voser 1999]). Les coordonnées du système fédéral suisse se basent par exemple sur une instance particulière d'un système de coordonnées de référence [Gubler et al 1996], qui peut être dérivé d'un système de référence géodésique par une projection cartographique [Snyder 1987, Bugayevskiy 1995]. Ces systèmes de référence géodésique forment une catégorie propre de systèmes de coordonnées de référence décrivant la géométrie du modèle de la Terre. Une sphère ou un ellipsoïde sur lequel des coordonnées géographiques peuvent être définies est par exemple utilisé pour décrire une position bidimensionnelle. Cela se complique encore un peu dès que l'on introduit la notion d'altitude. On emploie comme modèle géométrique et physique de la Terre un géoïde [Marti 1997] ou un modèle gravimétrique [Torge 1975] qui définit des altitudes orthométriques, resp. normales. Mais il arrive très souvent que seules des altitudes usuelles soient employées en pratique.

Puisque les géodonnées d'applications géomatiques présentent toujours une référence spatiale, chaque jeu de géodonnées doit être basé sur un système de coordonnées. Les systèmes de coordonnées individuels différant largement, il est nécessaire de préciser le système de référence correspondant avec les géodonnées. C'est pourquoi INTERLIS permet de décrire des données appartenant à un système de coordonnées.

C'est seulement par la connaissance du système de coordonnées sous-jacent qu'il est possible de transférer des géodonnées dans un autre système de coordonnées. Ceci est également nécessaire si les géodonnées provenant de différents systèmes de coordonnées doivent être utilisées ensembles [Voser 1996].

D'abord, nous traiterons des systèmes de coordonnées de manière générale, puis des relations (représentations) entre systèmes de coordonnées avant d'introduire des systèmes de coordonnées de référence et de les appliquer.

Systèmes de coordonnées

Un *système de coordonnées* permet la "mesure" d'un espace métrique. Un système de coordonnées possède une origine, des axes de coordonnées (leur nombre correspond à la dimension de l'espace couvert) ainsi que les unités de mesure attribuées aux axes. Selon que l'espace considéré a une, deux ou trois dimensions, le système de coordonnées attribue un seul nombre, deux nombres ou trois nombres à chaque point de l'espace comme coordonnées.

L'espace euclidien à une, deux ou trois dimensions est défini par ses 1, 2 ou 3 axes rectilignes. Les espaces curvilignes demandent des paramètres supplémentaires pour définir l'insertion de leurs axes curvilignes dans l'espace euclidien. Pour des besoins géodésiques, des espaces ellipsoïdaux bidimensionnels ainsi que divers systèmes altimétriques, tous traités comme des cas spéciaux d'espaces euclidiens unidimensionnels, sont nécessaires en plus des espaces euclidiens de différentes dimensions. C'est à cela que servent un modèle de gravité ou un géoïde.

D'une manière légèrement différente de l'utilisation faite jusqu'alors en géodésie, nous employons ici l'expression de *datum géodésique* comme synonyme de *système géodésique de référence*, ne désignant rien d'autre qu'un système de coordonnées spécial qui est un système cartésien 3D positionné par rapport à la Terre. Ceci peut être réalisé de deux manières :

(a) Le centre de gravité moyen de la Terre est défini comme point zéro (origine) du système de coordonnées, le premier axe est défini par la direction de l'axe moyen de rotation de la Terre, le deuxième axe est perpendiculaire au précédent dans la direction du méridien moyen de Greenwich et le troisième axe est également perpendiculaire aux deux premiers, créant ainsi un système direct. Le système de coordonnées WGS84 est par exemple défini de cette manière.

(b) La surface d'une région donnée de la Terre (généralement un pays) est approchée de manière optimale par une sphère ou un ellipsoïde dont l'axe de rotation est parallèle à la direction moyenne de l'axe de rotation de la Terre. Cet ellipsoïde définit un système cartésien de coordonnées par son demi-petit axe parallèle à l'axe de la Terre, par son demi-grand axe et par le troisième axe perpendiculaire aux deux premiers, créant ainsi un système horaire.

Un système de coordonnées 3D cartésien positionné selon les principes (a) ou (b) est appelé un *datum géodésique* ou un *système géodésique de référence*.

Origine différente des systèmes de coordonnées en géomatique

Différentes expériences et connaissances conduisent à des définitions variées des systèmes de coordonnées :

Domaine des capteurs : les capteurs de saisie de données en géodésie classique (par exemple avec des théodolites) ainsi qu'en photogrammétrie et en télédétection recourent à un système de coordonnées (local) adapté à la méthode de saisie concernée.

Géopositionnement : la description de la position sur la Terre au moyen d'un modèle (géodésique) terrestre. Il existe trois différents types de modèle géodésique terrestre [Voser 1999] :

- *physique :* le modèle terrestre est décrit soit par le champ gravitationnel, soit par un géoïde.
- *mathématique :* le modèle terrestre est un corps symétrique (par exemple une sphère ou un ellipsoïde).
- *topographique :* le modèle terrestre prend également en considération les montagnes et les vallées (modèle surfacique).

Les modèles mentionnés ci-dessus correspondent à différents systèmes de coordonnées.

Positionnement cartographique : les surfaces des modèles précédents étant courbes voire plus complexes, le calcul des distances, des angles etc. est très difficile. C'est pourquoi nous utilisons des

projections cartographiques qui représentent la surface bidimensionnelle dans un plan. Une projection cartographique est une opération géométrique clairement définie permettant de représenter la surface d'un modèle mathématique terrestre dans un plan. Cette procédure inclut des déformations qui peuvent toutefois être déterminées à l'avance et contrôlées.

Applications entre systèmes de coordonnées

Les géodonnées sont généralement saisies dans des systèmes de coordonnées différents ou gérées par des institutions différentes dans d'autres systèmes, de sorte qu'il est nécessaire de connaître les méthodes permettant le passage de données exprimées dans un système d'origine A dans un système cible Z. Cette opération est appelée application du système de coordonnées A, resp. de l'espace défini par A, dans le système de coordonnées Z, resp. l'espace défini par Z. L'application entre deux systèmes de coordonnées, resp. entre les espaces qu'ils définissent, est déterminée par les classes des deux systèmes de coordonnées concernés.

Nous devons distinguer entre deux formes d'applications fondamentalement différentes entre systèmes de coordonnées pour autant que l'origine des formules et de leurs paramètres soit concernée : les conversions et les transformations.

La *conversion* est une application entre deux systèmes de coordonnées strictement définie par des formules et leurs paramètres. Ces formules et spécialement les valeurs des paramètres nécessaires sont déterminées à l'avance [Voser 1999]. Dans la catégorie des conversions on trouve par exemple les projections cartographiques, c.-à-d. la représentation de la surface de l'ellipsoïde dans un plan, ainsi que la conversion des coordonnées ellipsoïdiques en coordonnées cartésiennes correspondantes, ou vice-versa, avec origine au centre de l'ellipsoïde.

Une *transformation* est une application entre deux systèmes de coordonnées pour laquelle des règles (formules) sont déterminées sur la base d'hypothèses et des paramètres sont calculés au moyen d'analyses statistiques de mesures faites dans les deux systèmes de coordonnées [Voser 1999]. Des transformations sont d'ordinaire effectuées lorsqu'un modèle géodésique en remplace un autre (transformation de datum géodésique) ou lorsque l'on ajuste des coordonnées locales dans un système de rang supérieur, c'est à dire, dans le cas d'une numérisation, la transformation de coordonnées liées à une feuille ou à la tablette de numérisation dans un système de projection défini.

Systèmes de coordonnées de référence

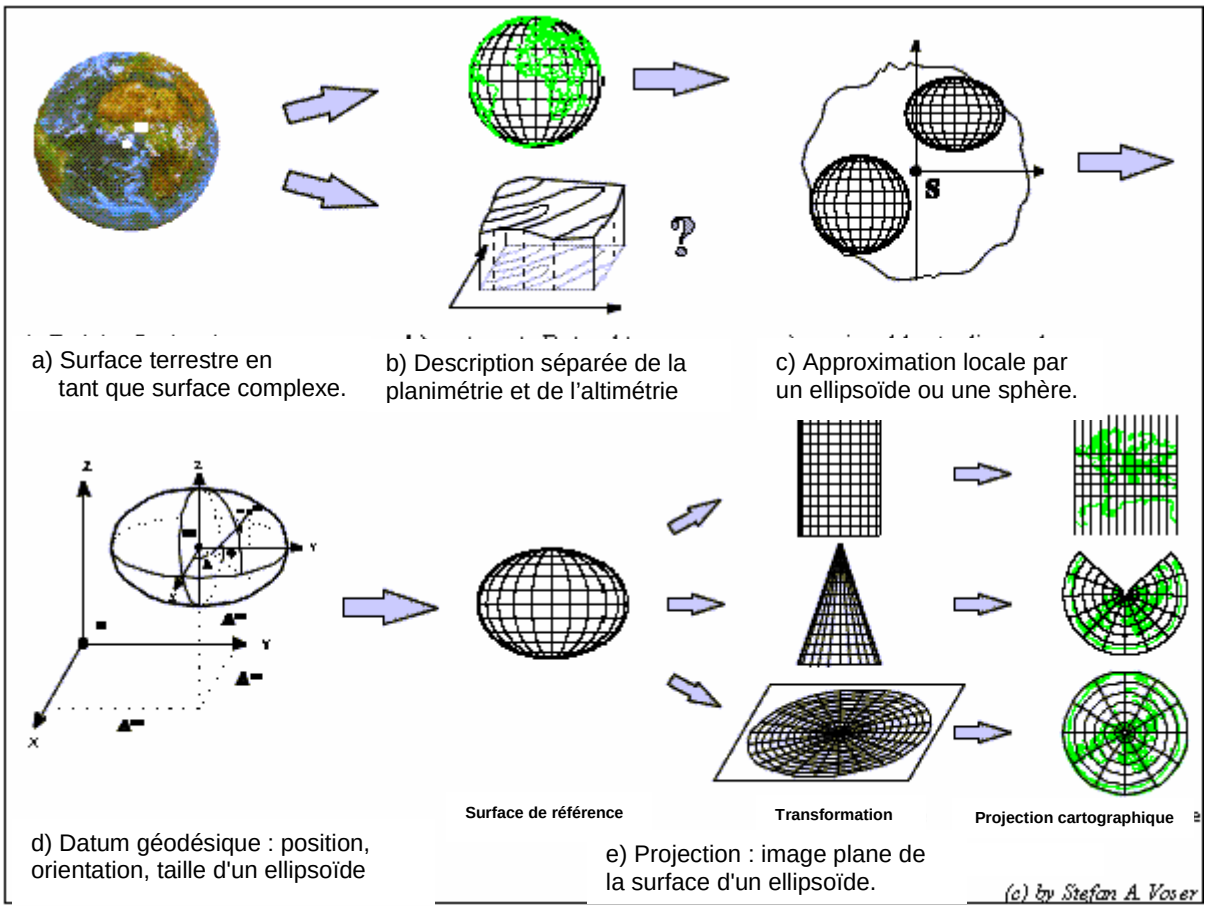
L'expression de *système de coordonnées de référence* décrit un système de coordonnées qui peut être dérivé d'un système de référence géodésique (datum géodésique) par conversion via une suite de systèmes de coordonnées intermédiaires.

Les systèmes géodésiques de référence (ou datum géodésique) sont les systèmes de coordonnées de référence les plus importants. Ils se réfèrent à un modèle géodésique de la Terre (cf. plus haut).

Evaluation des systèmes de coordonnées de référence les plus importants

La figure J.1 présente quelques systèmes de coordonnées de référence géodésiques et cartographiques importants. La Terre elle-même est à l'origine de chaque séquence de systèmes de coordonnées ou de transformations. Nous avons essayé de lui attribuer un modèle géométrique nous permettant de décrire des positions à sa surface. Dans un premier temps, un système de coordonnées cartésien 3D est affecté à la Terre avec le centre de gravité de la Terre pour origine (cf. méthode (a) du chapitre Systèmes de coordonnées précédent). La position et l'altitude d'un point seront traitées séparément dans la suite. Intéressons-nous d'abord à ce qui a doit être fait afin de déterminer la position d'un point. Des mesures géodésiques apportent l'information nécessaire pour déterminer les dimensions et la forme d'un ellipsoïde qui approcherait localement la surface terrestre de manière optimale. Conformément à la

méthode (b) du chapitre Systèmes de coordonnées, un datum géodésique peut être assigné à cet ellipsoïde. Bon nombre de modèles terrestres choisis pour des mensurations nationales sont positionnés "localement", c'est-à-dire que le centre de l'ellipsoïde ne coïncide par avec le centre de gravité de la Terre. Toutefois, comme établi plus haut ((a) du chapitre Système de coordonnées), il existe des systèmes géodésiques de référence ayant le centre de gravité terrestre pour origine. C'est la raison pour laquelle il est relativement facile, actuellement, de déterminer les paramètres d'une transformation de datum vers un tel système de rang supérieur. Si l'on a pris la décision d'utiliser un ellipsoïde de rotation local, il est possible de projeter sa surface sur un plan au moyen d'une projection cartographique appropriée répondant aux exigences posées.



a) Surface terrestre en tant que surface complexe.

b) Description séparée de la planimétrie et de l'altimétrie

c) Approximation locale par un ellipsoïde ou une sphère.

d) Datum géodésique : position, orientation, taille d'un ellipsoïde

e) Projection : image plane de la surface d'un ellipsoïde.

Figure J.1 : Transformation de la surface terrestre en coordonnées 2D [Voser 1998].

Structure de données pour des systèmes de coordonnées et applications entre elles

Le schéma conceptuel proposé pour les données nécessaires à la description des systèmes de coordonnées et aux relations entre elles n'est pas limité aux systèmes de coordonnées de référence mais a été conçu pour les systèmes de coordonnées en général. Notre intention est de permettre également la description des systèmes de coordonnées des tablettes de numérisation et des écrans ou des systèmes de coordonnées de signatures sans aucune référence explicite à la surface de la Terre.

Les systèmes de coordonnées et les représentations qui les relient sont les deux concepts-clés d'une caractérisation exacte de la référence spatiale des géodonnées. En conséquence, le modèle conceptuel (ou schéma conceptuel) de structure de données présente deux groupes de classes principaux, à savoir les "Systèmes de coordonnées pour les besoins géodésiques" et les "Correspondances entre systèmes de coordonnées". La troisième dimension, l'altimétrie, est traitée comme suit : dans un système de

coordonnées cartésien 3D, l'altitude est implicitement intégrée en tant que troisième coordonnée. Toutefois, les systèmes de coordonnées utilisés dans la pratique quotidienne sont généralement une combinaison d'un système horizontal 2D et d'un système supplémentaire 1D pour l'altimétrie. Les données de systèmes de coordonnées de ce type sont décrites par deux jeux de données indépendants, d'abord les données du système de coordonnées 2D (cartésien ou ellipsoïdique 2D) puis les données du système altimétrique approprié (normal, orthométrique ou ellipsoïdique).

Comment les structures de données proposées aident-elles efficacement à la correspondance entre les systèmes de coordonnées ? De la manière suivante : les systèmes de coordonnées forment les nœuds et les relations entre eux constituent les arêtes d'une structure de graphe. Le nom du système de coordonnées utilisé est indiqué dans la section DOMAIN d'un modèle d'application (schéma d'application). Si des coordonnées source doivent être projetées dans un autre système de coordonnées ou si des paramètres GeoTIFF correspondant à une telle transformation doivent par exemple être calculés, alors un programme approprié doit trouver, dans la structure du graphe, le chemin le plus court possible allant du nœud du système de coordonnées source (conformément à DOMAIN) au nœud du système cible. Ensuite, les transformations nécessaires entre le système source et le système cible, via d'éventuels systèmes de coordonnées intermédiaires, doivent être calculées.

Deux classes internes et deux mots-clés sont disponibles dans INTERLIS pour la description des systèmes de coordonnées : AXIS et COORDSYSTEM. Ils sont utilisés dans le modèle conceptuel de données (ou modèle de système de référence) "CoordSys" (cf. ci-après). Pour plus de détails, consulter le paragraphe 2.10.3 Systèmes de référence du manuel de référence d'INTERLIS 2.

Bibliographie

[Bugayevskiy 1995] Bugayevskiy Lev M., Snyder John P.: Map Projections, A Reference Manual, Taylor&Francis, Londres, Bristol 1995.

[Gubler et al. 1996] Gubler E., Gutknecht D., Marti U., Schneider D. Signer Th., Vogel B., Wiget A.: Die neue Landesvermessung der Schweiz LV95; MPG 2/96.

[Marti 1997] Marti, Urs : Geoid der Schweiz 1997. Geodätisch-geophysikalische Arbeiten in der Schweiz, Commission Géodésique Suisse, Volume 56, Zurich, 1997.

[Torge 1975] Torge, Wolfgang : Geodäsie. Sammlung Göschen 2163, de Gruyter, Berlin – New York, 1975.

[Snyder 1987] Snyder, John P.: Map Projections – A Working Manual, U.S. Geological Survey Professional Paper 1395, Washington, 1987.

[Voser 1996] Voser, Stefan A ; Anforderungen an die Geometrie zur gemeinsamen Nutzung unterschiedlicher Datenquellen; 4. deutsche Arc/Info-Anwender-Konferenz, Proceedings, mars 1996, Freising.

[Voser 1998] Voser S. A.: Schritte für ein automatisiertes Koordinatensystemmanagement in GIS und Kartographie. Nachrichten aus dem Karten- und Vermessungswesen, Reihe I, Heft Nr. 118, S. 111-125. Bundesamt für Kartographie und Geodäsie, Francfort-sur-le-Main, 1998.

[Voser 1999] Voser, S. A.: MapRef - The Internet Collection of Map Projections and Reference Systems for Europe; 14. ESRI European User Conference, Presentation and Proceedings; 15.-17. novembre 1999 ; Munich; www.mapref.org/.

Le modèle de système de référence

Structure de données pour des systèmes de coordonnées et des systèmes de coordonnées de référence ainsi que les relations entre eux. Modèle conceptuel de données (schéma conceptuel) avec INTERLIS.

```
!! File CoordSys.ili Release 2003-03-18

INTERLIS 2.2;

REFSYSTEM MODEL CoordSys (en) =  !! 2-letter code (ISO 639)

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];
    Angle_Minute = 1 / 60 [Angle_Degree];
    Angle_Second = 1 / 60 [Angle_Minute];
    Angle_DMS = {Angle_Degree:Angle_Minute[0 .. 59]:Angle_Second[0 .. 59]}
                CONTINUOUS;


  TOPIC CoordsysTopic =

    !! Special space aspects to be referenced
    !! ************************************

    CLASS Ellipsoid EXTENDS INTERLIS.REFSYSTEM =
      EllipsoidAlias: TEXT*70;
      SemiMajorAxis: MANDATORY 6360000.0000 .. 6390000.0000 [INTERLIS.m];
      InverseFlattening: MANDATORY 0.00000000 .. 350.00000000;
      !! The inverse flattening 0 characterizes the 2-dim sphere
      Remarks: TEXT*70;
    END Ellipsoid;

    CLASS GravityModel EXTENDS INTERLIS.REFSYSTEM =
      GravityModAlias: TEXT*70;
      Definition: TEXT*70;
    END GravityModel;

    CLASS GeoidModel EXTENDS INTERLIS.REFSYSTEM =
      GeoidModAlias: TEXT*70;
      Definition: TEXT*70;
    END GeoidModel;


    !! Coordinate systems for geodetic purposes
    !! ****************************************

    STRUCTURE LengthAXIS EXTENDS INTERLIS.AXIS =
      ShortName: TEXT*12;
      Description: TEXT*255;
    PARAMETER
      Unit (EXTENDED): NUMERIC [INTERLIS.LENGTH];
    END LengthAXIS;

    STRUCTURE AngleAXIS EXTENDS INTERLIS.AXIS =
      ShortName: TEXT*12;
      Description: TEXT*255;
    PARAMETER
      Unit (EXTENDED): NUMERIC [INTERLIS.ANGLE];
    END AngleAXIS;

    CLASS GeoCartesian1D EXTENDS INTERLIS.COORDSYSTEM =
      Axis (EXTENDED): LIST {1} OF LengthAXIS;
    END GeoCartesian1D;

    CLASS GeoHeight EXTENDS GeoCartesian1D =
      System: MANDATORY (
        normal,
```

```
    orthometric,
    ellipsoidal,
    other);
  ReferenceHeight: MANDATORY -10000.000 .. +10000.000 [INTERLIS.m];
  ReferenceHeightDescr: TEXT*70;
END GeoHeight;

ASSOCIATION HeightEllips =
  GeoHeightRef -- {*} GeoHeight;
  EllipsoidRef -- {1} Ellipsoid;
END HeightEllips;

ASSOCIATION HeightGravit =
  GeoHeightRef -- {*} GeoHeight;
  GravityRef -- {1} GravityModel;
END HeightGravit;

ASSOCIATION HeightGeoid =
  GeoHeightRef -- {*} GeoHeight;
  GeoidRef -- {1} GeoidModel;
END HeightGeoid;

CLASS GeoCartesian2D EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {2} OF LengthAXIS;
END GeoCartesian2D;

CLASS GeoCartesian3D EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {3} OF LengthAXIS;
END GeoCartesian3D;

CLASS GeoEllipsoidal EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {2} OF AngleAXIS;
END GeoEllipsoidal;

ASSOCIATION EllCSEllips =
  GeoEllipsoidalRef -- {*} GeoEllipsoidal;
  EllipsoidRef -- {1} Ellipsoid;
END EllCSEllips;


!! Mappings between coordinate systems
!! **********************************

ASSOCIATION ToGeoEllipsoidal =
  From -- {1..*} GeoCartesian3D;
  To -- {1..*} GeoEllipsoidal;
  ToHeight -- {1..*} GeoHeight;
MANDATORY CONSTRAINT
  ToHeight -> System == #ellipsoidal;
MANDATORY CONSTRAINT
  To -> EllipsoidRef -> Name == ToHeight -> EllipsoidRef -> Name;
END ToGeoEllipsoidal;

ASSOCIATION ToGeoCartesian3D =
  From2 -- {1..*} GeoEllipsoidal;
  FromHeight-- {1..*} GeoHeight;
  To3 -- {1..*} GeoCartesian3D;
MANDATORY CONSTRAINT
  FromHeight -> System == #ellipsoidal;
MANDATORY CONSTRAINT
  From2 -> EllipsoidRef -> Name == FromHeight -> EllipsoidRef -> Name;
END ToGeoCartesian3D;

ASSOCIATION BidirectGeoCartesian3D =
  From -- {1..*} GeoCartesian3D;
```

```
      To2 -- {1..*} GeoCartesian3D;
      Precision: MANDATORY (
        exact,
        measure_based);
      ShiftAxis1: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
      ShiftAxis2: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
      ShiftAxis3: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
      RotationAxis1: -90:00:00.000 .. 90:00:00.000 [Angle_DMS];
      RotationAxis2: -90:00:00.000 .. 90:00:00.000 [Angle_DMS];
      RotationAxis3: -90:00:00.000 .. 90:00:00.000 [Angle_DMS];
      NewScale: 0.000001 .. 1000000.000000;
    END BidirectGeoCartesian3D;

    ASSOCIATION BidirectGeoCartesian2D =
      From -- {1..*} GeoCartesian2D;
      To -- {1..*} GeoCartesian2D;
    END BidirectGeoCartesian2D;

    ASSOCIATION BidirectGeoEllipsoidal =
      From4 -- {1..*} GeoEllipsoidal;
      To4 -- {1..*} GeoEllipsoidal;
    END BidirectGeoEllipsoidal;

    ASSOCIATION MapProjection (ABSTRACT) =
      From5 -- {1..*} GeoEllipsoidal;
      To5 -- {1..*} GeoCartesian2D;
      FromCo1_FundPt: MANDATORY -90:00:00.00 .. +90:00:00.00 [Angle_DMS];
      FromCo2_FundPt: MANDATORY -90:00:00.00 .. +90:00:00.00 [Angle_DMS];
      ToCoord1_FundPt: MANDATORY -10000000 .. +10000000 [INTERLIS.m];
      ToCoord2_FundPt: MANDATORY -10000000 .. +10000000 [INTERLIS.m];
    END MapProjection;

    ASSOCIATION TransverseMercator EXTENDS MapProjection =
    END TransverseMercator;

    ASSOCIATION SwissProjection EXTENDS MapProjection =
      IntermFundP1: MANDATORY -90:00:00.00 .. +90:00:00.00 [Angle_DMS];
      IntermFundP2: MANDATORY -90:00:00.00 .. +90:00:00.00 [Angle_DMS];
    END SwissProjection;

    ASSOCIATION Mercator EXTENDS MapProjection =
    END Mercator;

    ASSOCIATION ObliqueMercator EXTENDS MapProjection =
    END ObliqueMercator;

    ASSOCIATION Lambert EXTENDS MapProjection =
    END Lambert;

    ASSOCIATION Polyconic EXTENDS MapProjection =
    END Polyconic;

    ASSOCIATION Albus EXTENDS MapProjection =
    END Albus;

    ASSOCIATION Azimutal EXTENDS MapProjection =
    END Azimutal;

    ASSOCIATION Stereographic EXTENDS MapProjection =
    END Stereographic;

    ASSOCIATION HeightConversion =
      FromHeight -- {1..*} GeoHeight;
      ToHeight -- {1..*} GeoHeight;
      Definition: TEXT*70;
    END HeightConversion;

  END CoordsysTopic;
```

```
END CoordSys.
```

Le fichier MiniCoordSysDaten dont le nom peut apparaître dans MetadataUseDef, contient les données suivantes dans le format de transfert d'INTERLIS 2 :

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- File MiniCoordSysData.xml Release 2003-03-18 -->

<TRANSFER xmlns="http://www.interlis.ch/INTERLIS2.2"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.interlis.ch/INTERLIS2.2
                              MiniCoordSysData.xsd">
  <HEADERSECTION VERSION="2.2" SENDER="V+D">
    <ALIAS>
      <ENTRIES FOR="CoordSys">
        <TAGENTRY FROM="CoordSys.Angle_DMS_S"
                  TO="CoordSys.Angle_DMS_S"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic"
                  TO="CoordSys.CoordsysTopic"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Ellipsoid"
                  TO="CoordSys.CoordsysTopic.Ellipsoid"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GravityModel"
                  TO="CoordSys.CoordsysTopic.GravityModel"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoidModel"
                  TO="CoordSys.CoordsysTopic.GeoidModel"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.LengthAXIS"
                  TO="CoordSys.CoordsysTopic.LengthAXIS"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.AngleAXIS"
                  TO="CoordSys.CoordsysTopic.AngleAXIS"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoCartesian1D"
                  TO="CoordSys.CoordsysTopic.GeoCartesian1D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoHeight"
                  TO="CoordSys.CoordsysTopic.GeoCartesian1D"/>
        <DELENTRY TAG="CoordSys.CoordsysTopic.GeoHeight.System"/>
        <DELENTRY TAG="CoordSys.CoordsysTopic.GeoHeight.ReferenceHeight"/>
        <DELENTRY TAG="CoordSys.CoordsysTopic.GeoHeight.ReferenceHeightDescr"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoHeight"
                  TO="CoordSys.CoordsysTopic.GeoHeight"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.HeightEllips"
                  TO="CoordSys.CoordsysTopic.HeightEllips"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.HeightGravit"
                  TO="CoordSys.CoordsysTopic.HeightGravit"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.HeightGeoid"
                  TO="CoordSys.CoordsysTopic.HeightGeoid"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoCartesian2D"
                  TO="CoordSys.CoordsysTopic.GeoCartesian2D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoCartesian3D"
                  TO="CoordSys.CoordsysTopic.GeoCartesian3D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.GeoEllipsoidal"
                  TO="CoordSys.CoordsysTopic.GeoEllipsoidal"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.EllCSEllips"
                  TO="CoordSys.CoordsysTopic.EllCSEllips"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.ToGeoEllipsoidal"
                  TO="CoordSys.CoordsysTopic.ToGeoEllipsoidal"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.ToGeoCartesian3D"
                  TO="CoordSys.CoordsysTopic.ToGeoCartesian3D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.BidirectGeoCartesian2D"
                  TO="CoordSys.CoordsysTopic.BidirectGeoCartesian2D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.BidirectGeoCartesian3D"
                  TO="CoordSys.CoordsysTopic.BidirectGeoCartesian3D"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.BidirectGeoEllipsoidal"
                  TO="CoordSys.CoordsysTopic.BidirectGeoEllipsoidal"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.TransverseMercator"
                  TO="CoordSys.CoordsysTopic.TransverseMercator"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.SwissProjection"
                  TO="CoordSys.CoordsysTopic.SwissProjection"/>
```

```
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Mercator"
                  TO="CoordSys.CoordsysTopic.Mercator"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.ObliqueMercator"
                  TO="CoordSys.CoordsysTopic.ObliqueMercator"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Lambert"
                  TO="CoordSys.CoordsysTopic.Lambert"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Polyconic"
                  TO="CoordSys.CoordsysTopic.Polyconic"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Albus"
                  TO="CoordSys.CoordsysTopic.Albus"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Azimutal"
                  TO="CoordSys.CoordsysTopic.Azimutal"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.Stereographic"
                  TO="CoordSys.CoordsysTopic.Stereographic"/>
        <TAGENTRY FROM="CoordSys.CoordsysTopic.HeightConversion"
                  TO="CoordSys.CoordsysTopic.HeightConversion"/>
      </ENTRIES>
    </ALIAS>

    <COMMENT>
      example dataset ili2 refmanual appendix J
    </COMMENT>
  </HEADERSECTION>

  <DATASECTION>
    <CoordSys.CoordsysTopic BID="xBCoordSys">
      <CoordSys.CoordsysTopic.Ellipsoid TID="xBCoordSysBessel">
        <Name>Bessel</Name>
        <EllipsoidAlias>Bessel(1841)</EllipsoidAlias>
        <SemiMajorAxis>6377397.1550</SemiMajorAxis>
        <InverseFlattening>299.1528128</InverseFlattening>
        <Remarks>Dok L+T 19031266</Remarks>
      </CoordSys.CoordsysTopic.Ellipsoid>

      <CoordSys.CoordsysTopic.Ellipsoid TID="xBCoordSysWGS72">
        <Name>WGS72</Name>
        <EllipsoidAlias>World Geodetic System 1972</EllipsoidAlias>
        <SemiMajorAxis>6378135.000</SemiMajorAxis>
        <InverseFlattening>298.2600000</InverseFlattening>
      </CoordSys.CoordsysTopic.Ellipsoid>

      <CoordSys.CoordsysTopic.GravityModel
        TID="xBCoordSysCHLotabweichung">
        <Name>CHLotabweichung</Name>
        <Definition>Siehe Software LAG L+T</Definition>
      </CoordSys.CoordsysTopic.GravityModel>

      <CoordSys.CoordsysTopic.GeoidModel TID="xBCoordSysCHGeoid">
        <Name>CHGeoid</Name>
        <Definition>Siehe neues CH Geoid L+T</Definition>
      </CoordSys.CoordsysTopic.GeoidModel>

      <CoordSys.CoordsysTopic.GeoHeight
        TID="xBCoordSysSwissOrthometricAlt">
        <Name>SwissOrthometricAlt</Name>
        <Axis>
          <CoordSys.CoordsysTopic.LengthAXIS>
            <ShortName>h</ShortName>
            <Description>CHOrthometricAlt</Description>
          </CoordSys.CoordsysTopic.LengthAXIS>
        </Axis>
        <System>orthometric</System>
        <ReferenceHeight>373.600</ReferenceHeight>
        <ReferenceHeightDescr>Pierre du Niton</ReferenceHeightDescr>
        <EllipsoidRef REF="xBCoordSysBessel"/>
        <GeoidRef REF="xBCoordSysCHGeoid"/>
        <GravityRef REF="xBCoordSysCHLotabweichung"/>
      </CoordSys.CoordsysTopic.GeoHeight>
```

```
<CoordSys.CoordsysTopic.GeoHeight
  TID="xBCoordSysSwissEllipsoidalAlt">
  <Name>SwissEllipsoidalAlt</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>H</ShortName>
      <Description>CHEllipsoidalAlt</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <System>ellipsoidal</System>
  <ReferenceHeight>0.000</ReferenceHeight>
  <ReferenceHeightDescr>Meereshoehe</ReferenceHeightDescr >
  <EllipsoidRef REF="xBCoordSysBessel"/>
  <GeoidRef REF="xBCoordSysCHGeoid"/>
  <GravityRef REF="xBCoordSysCHLotabweichung"/>
</CoordSys.CoordsysTopic.GeoHeight>

<CoordSys.CoordsysTopic.GeoCartesian2D TID="xBCoordSysCOORD2">
  <Name>COORD2</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>X</ShortName>
      <Description>X-Axis</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>Y</ShortName>
      <Description>Y-Axis</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <Definition>Mathematisches 2D Refsystem</Definition>
</CoordSys.CoordsysTopic.GeoCartesian2D>

<CoordSys.CoordsysTopic.GeoCartesian2D TID="xBCoordSysCHLV03">
  <Name>CHLV03</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>Y</ShortName>
      <Description>Ost-Wert</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>X</ShortName>
      <Description>Nord-Wert</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <Definition>Geodaetisches 2D Refsystem</Definition>
</CoordSys.CoordsysTopic.GeoCartesian2D>

<CoordSys.CoordsysTopic.GeoCartesian3D TID="xBCoordSysCOORD3">
  <Name>COORD3</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>X</ShortName>
      <Description>X-Axis</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>Y</ShortName>
      <Description>Y-Axis</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>Z</ShortName>
      <Description>Z-Axis</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <Definition>Mathematisches 3D Refsystem</Definition>
</CoordSys.CoordsysTopic.GeoCartesian3D>

<CoordSys.CoordsysTopic.GeoCartesian3D TID="xBCoordSysCH1903">
```

```
  <Name>CH1903</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>XC</ShortName>
      <Description>Aequator Greenwich</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>YC</ShortName>
      <Description>Aequator East</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>ZC</ShortName>
      <Description>North</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <Definition>Geodaetisches 3D Refsystem CH</Definition>
</CoordSys.CoordsysTopic.GeoCartesian3D>

<CoordSys.CoordsysTopic.GeoCartesian3D TID="xBCoordSysWGS84">
  <Name>WGS84</Name>
  <Axis>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>XW</ShortName>
      <Description>Aequator Greenwich</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>YW</ShortName>
      <Description>Aequator_East</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
    <CoordSys.CoordsysTopic.LengthAXIS>
      <ShortName>ZW</ShortName>
      <Description>North</Description>
    </CoordSys.CoordsysTopic.LengthAXIS>
  </Axis>
  <Definition>World Geodetic System 1984</Definition>
</CoordSys.CoordsysTopic.GeoCartesian3D>

<CoordSys.CoordsysTopic.GeoEllipsoidal
  TID="xBCoordSysSwitzerland">
  <Name>Switzerland</Name>
  <Axis>
    <CoordSys.CoordsysTopic.AngleAXIS>
      <ShortName>Lat</ShortName>
      <Description>Breite</Description>
    </CoordSys.CoordsysTopic.AngleAXIS>
    <CoordSys.CoordsysTopic.AngleAXIS>
      <ShortName>Long</ShortName>
      <Description>Laenge</Description>
    </CoordSys.CoordsysTopic.AngleAXIS>
  </Axis>
  <Definition>Koordinaten auf dem CH Ellipsoid 1903</Definition>
  <EllipsoidRef REF="xBCoordSysBessel"/>
</CoordSys.CoordsysTopic.GeoEllipsoidal>

<CoordSys.CoordsysTopic.ToGeoEllipsoidal
  TID="xBCoordSysFromCH1903toSwitzerland">
  <From REF= "xBCoordSysCH1903"></From>
  <To REF= "xBCoordSysSwitzerland"></To>
  <ToHeight REF= "xBCoordSysSwissEllipsoidalAlt"></ToHeight>
</CoordSys.CoordsysTopic.ToGeoEllipsoidal>

<CoordSys.CoordsysTopic.ToGeoCartesian3D
  TID="xBCoordSysFromSwitzerlandToCH1903">
  <From2 REF="xBCoordSysSwitzerland"></From2>
  <FromHeight REF="xBCoordSysSwissEllipsoidalAlt"></FromHeight>
  <To3 REF="xBCoordSysCH1903"></To3>
</CoordSys.CoordsysTopic.ToGeoCartesian3D>
```

```
      <CoordSys.CoordsysTopic.BidirectGeoCartesian3D
        TID="xBCoordSysWGS84toCH1903">
        <From REF= "xBCoordSysWGS84"></From>
        <To2 REF= "xBCoordSysCH1903"></To2>
        <Precision>measure_based</Precision>
        <ShiftAxis1>-660.077</ShiftAxis1>
        <ShiftAxis2>-13.551</ShiftAxis2>
        <ShiftAxis3>-369.344</ShiftAxis3>
        <RotationAxis1>-0:0:2.484</RotationAxis1>
        <RotationAxis2>-0:0:1.783</RotationAxis2>
        <RotationAxis3>-0:0:2.939</RotationAxis3>
        <NewScale>0.99444</NewScale>
      </CoordSys.CoordsysTopic.BidirectGeoCartesian3D>

      <CoordSys.CoordsysTopic.BidirectGeoCartesian3D
        TID="xBCoordSysCH1903toWGS84">
        <From REF= "xBCoordSysCH1903"></From>
        <To2 REF= "xBCoordSysWGS84"></To2>
        <Precision>measure_based</Precision>
        <ShiftAxis1>660.077</ShiftAxis1>
        <ShiftAxis2>13.551</ShiftAxis2>
        <ShiftAxis3>369.344</ShiftAxis3>
        <RotationAxis1>0:0:2.484</RotationAxis1>
        <RotationAxis2>0:0:1.783</RotationAxis2>
        <RotationAxis3>0:0:2.939</RotationAxis3>
        <NewScale>1.00566</NewScale>
      </CoordSys.CoordsysTopic.BidirectGeoCartesian3D>

      <CoordSys.CoordsysTopic.TransverseMercator
        TID="xBCoordSysFromCH1903ToSwitzerland">
        <From5 REF= "xBCoordSysSwitzerland"></From5>
        <To5 REF= "xBCoordSysCHLV03"></To5>
        <FromCo1_FundPt>46:57:08.66</FromCo1_FundPt>
        <FromCo2_FundPt>7:26:22.50</FromCo2_FundPt>
        <ToCoord1_FundPt>600000</ToCoord1_FundPt>
        <ToCoord2_FundPt>200000</ToCoord2_FundPt>
      </CoordSys.CoordsysTopic.TransverseMercator>

      <CoordSys.CoordsysTopic.HeightConversion
        TID="xBCoordSysElliphToOrth">
        <FromHeight REF= "xBCoordSysSwissEllipsoidalAlt"></FromHeight>
        <ToHeight REF= "xBCoordSysSwissOrthometricAlt"></ToHeight>
      </CoordSys.CoordsysTopic.HeightConversion>

      <CoordSys.CoordsysTopic.HeightConversion
        TID="xBCoordSysOrthToElliph">
        <FromHeight REF= "xBCoordSysSwissOrthometricAlt"></FromHeight>
        <ToHeight REF= "xBCoordSysSwissEllipsoidalAlt"></ToHeight>
      </CoordSys.CoordsysTopic.HeightConversion>
    </CoordSys.CoordsysTopic>
  </DATASECTION>
</TRANSFER>
```

Exemple

Que faudrait-il indiquer au sein d'un modèle d'application (ou d'un schéma d'application) pour identifier sans équivoque le système de coordonnées ou le système de coordonnées de référence utilisé ?

```
!! File CoordSysEx.ili Release 2003-03-18

INTERLIS 2.2;

MODEL Exemple (fr) =  !! 2-letter code (ISO 639)

  IMPORTS CoordSys;

  REFSYSTEM BASKET BCoordSys ~ CoordSys.CoordsysTopic;
```

```
  UNIT
    Meter = [INTERLIS.m];

  DOMAIN
    LKoord = COORD
      480000.000 .. 850000.000 [INTERLIS.m] {CHLV03[1]},
       60000.000 .. 320000.000 [INTERLIS.m] {CHLV03[2]},
      ROTATION 2 -> 1;
    Hoehe = COORD
      -200.000 .. 5000.000 [INTERLIS.m] {SwissOrthometricAlt[1]};
    HKoord = COORD
      480000.000 .. 850000.000 [INTERLIS.m] {CHLV03[1]},
       60000.000 .. 320000.000 [INTERLIS.m] {CHLV03[2]},
        -200.000 ..   5000.000 [INTERLIS.m] {SwissOrthometricAlt[1]},
      ROTATION 2 -> 1;


  TOPIC T =

    CLASS PointFixe =
      Nom: TEXT*20;
      Position: CoordP;
    END PointFixe;

  END T;

END Exemple.
```

Annexe K (Proposition d'extension) Modèle de signatures

Remarque

Les spécifications suivantes ne font pas partie intégrante de la norme INTERLIS. Il s'agit d'une proposition d'extension du manuel de référence d'INTERLIS 2 sous forme de recommandation. Il est prévu de transformer cette proposition, après une large discussion, en un règlement définitif. Pour des exemples d'application, veuillez vous reporter au manuel de l'utilisateur d'INTERLIS 2.

Modèle de signatures abstrait

Description du modèle de signatures abstrait.

```
!! File AbstractSymbology.ili Release 2003-03-18

INTERLIS 2.2;

SYMBOLOGY MODEL AbstractSymbology (en) =

  CONTRACT ISSUED BY Unknown; !! Contractor(s) have to be defined!

  UNIT
    Millimeter [mm] = 0.001 [INTERLIS.m];
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    Style_COORD2 (ABSTRACT)   = COORD NUMERIC, NUMERIC;
    Style_COORD3 (ABSTRACT)   = COORD NUMERIC, NUMERIC, NUMERIC;
    Style_POLYLINE (ABSTRACT) = POLYLINE WITH (STRAIGHTS, ARCS)
                                VERTEX Style_COORD2;  !! {Planar}?
    Style_SURFACE (ABSTRACT)  = SURFACE WITH (STRAIGHTS, ARCS)
                                VERTEX Style_COORD2;
    Style_INT (ABSTRACT)      = NUMERIC; !! [Units?]
    Style_FLOAT (ABSTRACT)    = NUMERIC; !! [Units?]
    Style_ANGLE (ABSTRACT)    = 0.000 .. 359.999 CIRCULAR [Angle_Degree]
                                COUNTERCLOCKWISE;  !! RefSystem?


  TOPIC Signs =

    !! Graphic interface

    CLASS TextSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Txt      : MANDATORY TEXT;
      Geometry : MANDATORY Style_COORD2;
      Rotation : Style_ANGLE; !! Default 0.0
      HAli     : HALIGNMENT;  !! Default Center
      VAli     : VALIGNMENT;  !! Default Half
    END TextSign;

    CLASS SymbolSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_COORD2;
      Scale    : Style_FLOAT;
      Rotation : Style_ANGLE; !! Default 0.0
    END SymbolSign;

    CLASS PolylineSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_POLYLINE;
    END PolylineSign;
```

```
    CLASS SurfaceSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_SURFACE;
    END SurfaceSign;

  END Signs;

END AbstractSymbology.
```

Modèle de signatures standard

Description du modèle de signatures standard étendu, basé sur le modèle de signatures abstrait.

```
!! File StandardSymbology.ili Release 2003-03-18

INTERLIS 2.2;

SYMBOLOGY MODEL StandardSymbology (en) =

  !! Extended symbology model with symbol libraries and priorities.

  CONTRACT ISSUED BY Unknown;  !! Contractor(s) have to be defined!

  IMPORTS AbstractSymbology;

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    SS_Priority = 0 .. 9999;
    SS_Float    = -2000000000.000 .. 2000000000.000;
    SS_Angle    = 0.000 .. 359.999
                  CIRCULAR [Angle_Degree] COUNTERCLOCKWISE;
    SS_Coord2   = COORD -2000000000.000 .. 2000000000.000 [INTERLIS.m],
                        -2000000000.000 .. 2000000000.000 [INTERLIS.m],
                  ROTATION 2 -> 1;
    SS_Polyline = POLYLINE WITH (STRAIGHTS, ARCS)
                  VERTEX SS_Coord2;
    SS_Surface  = SURFACE WITH (STRAIGHTS, ARCS)
                  VERTEX SS_Coord2;


  TOPIC StandardSigns EXTENDS AbstractSymbology.Signs =

    !! StandardSigns contains symbol libraries and symbol interfaces.
    !! The libraries (colors, fonts/symbols and line patterns) form the
    !! base for the construction of concrete symbols. The symbol interfaces
    !! extend the symbol interfaces of AbstractSymbology by priorites.

    !! Library section
    !! +++++++++++++++

    !! Color library
    !! =============
    !! Colors are defined by LCh values with transparency.

    CLASS Color =
      Name: TEXT*40; !! name of color, i.e. "light green"
      L: MANDATORY 0.0 .. 100.0; !! Luminance
      C: MANDATORY 0.0 .. 181.1; !! Chroma
      H: MANDATORY 0.0 .. 359.9 CIRCULAR [Angle_Degree] COUNTERCLOCKWISE; !! Hue
      T: MANDATORY 0.000 .. 1.000; !! Transparency: 0=totally transparent, 1=opaque
    END Color;

    !! Polyline attributes
    !! ++++++++++++++++++
    !! Presentation parameters for simple continuous lines. Polyline attributes
```

```
!! are used by all other polyline definitions (see also below).

CLASS PolylineAttrs =
  Width       : SS_Float;
  Join        : ( !! connection form for line segments
    bevel,
    round,
    miter
  );
  MiterLimit  : 1.0 .. 1000.0; !! only for Join = miter
  Caps        : (  !! termination form at end of line
    round,
    butt
  );
END PolylineAttrs;

!! Font- and symbol library
!! ========================
!! Symbols are a collection of lines and surfaces. Symbols are
!! organized in fonts. A font can be either a text font or a symbol
!! font. If the font is a text font (Type = #text), every symbol
!! (Character) has an UCS4 code (Unicode) and a spacing parameter assigned.

STRUCTURE FontSymbol_Geometry (ABSTRACT) =
  !! Basic structure for uniform treatment of all symbol geometries.
END FontSymbol_Geometry;

STRUCTURE FontSymbol_Polyline EXTENDS FontSymbol_Geometry =
  Color       : REFERENCE TO Color; !! only for symbols
  LineAttrs   : REFERENCE TO PolylineAttrs;
  Geometry    : MANDATORY SS_Polyline;
END FontSymbol_Polyline;

STRUCTURE FontSymbol_Surface EXTENDS FontSymbol_Geometry =
  FillColor   : REFERENCE TO Color; !! only for symbols
  Geometry    : MANDATORY SS_Surface;
  !! Remark: Has no line symbology, because the boundary is *not* part
  !! of the surface. With FillColor you define only the color of the
  !! surface filling.
END FontSymbol_Surface;

CLASS FontSymbol =
  !! All font symbols are defined for size 1.0 and scale 1.0.
  !! The value is measured in user units (i.e. normally [m]).
  Name        : TEXT*40; !! Symbol name, if known
  UCS4        : 0 .. 4000000000; !! only for text symbols (characters)
  Spacing     : SS_Float; !! only for text symbols (characters)
  Geometry    : LIST OF FontSymbol_Geometry;
END FontSymbol;

CLASS Font =
  Name        : MANDATORY TEXT*40; !! Font name or name of external font
  Internal    : MANDATORY BOOLEAN; !! Internal or external font
                                   !! Only for internal fonts the geometric
                                   !! definitions of the symbols is contained
                                   !! in FontSymbol.
  Type        : MANDATORY (
    symbol,
    text
  );
  BottomBase  : SS_Float; !! Only for text fonts, measured relative to text
                          !! height 1.0
END Font;

ASSOCIATION FontAssoc =
  Font -<#> {1} Font;
  Symbol -- {0..*} FontSymbol;
END FontAssoc;
```

```
!! Line symbology library
!! ======================
!! With the line sybology library the user can define continuos, dashed or
!! patterned lines. It is also possible to define multi line symbologies.
!! Each line in a multi line symbology can be continuous, dashed or patterned
!! for itself. The offset indicates the distance from the middle axis. All
!! are stored in the library relative to the width 1.0. The width can be over
!! written by the symbology parameter Width in the symbology interface. For
!! continuous lines the Width parameter defines the total width of the line,
!! for multi lines the parameter Width scales the attribute value offset.

CLASS LineStyle (ABSTRACT) =
  Name        : MANDATORY TEXT*40;
END LineStyle;

CLASS LineStyle_Solid EXTENDS LineStyle =
END LineStyle_Solid;

ASSOCIATION LineStyle_SolidColorAssoc =
  Color -- {0..1} Color;
  LineStyle -- {1} LineStyle_Solid;
END LineStyle_SolidColorAssoc;

ASSOCIATION LineStyle_SolidPolylineAttrsAssoc =
  LineAttrs -- {0..1} PolylineAttrs;
  LineStyle -- {1} LineStyle_Solid;
END LineStyle_SolidPolylineAttrsAssoc;

STRUCTURE DashRec =
  DLength     : SS_Float; !! Length of dash
END DashRec;

CLASS LineStyle_Dashed EXTENDS LineStyle =
  Dashes      : LIST OF DashRec; !! 1. dash is continuous
                                 !! 2. dash is not visible
                                 !! 3. dash is continuous
                                 !! etc.
END LineStyle_Dashed;

ASSOCIATION LineStyle_DashedColorAssoc =
  Color -- {0..1} Color;
  LineStyle_Dashed -- {1} LineStyle_Dashed;
END LineStyle_DashedColorAssoc;

ASSOCIATION LineStyle_DashedLineAttrsAssoc =
  LineAttrs -- {0..1} PolylineAttrs;
  LineStyle_Dashed -- {1} LineStyle_Dashed;
END LineStyle_DashedLineAttrsAssoc;

STRUCTURE Pattern_Symbol =
  FontSymbRef : MANDATORY REFERENCE TO FontSymbol;
  ColorRef    : REFERENCE TO Color;
  Weight      : SS_Float; !! Width for symbol lines
  Scale       : SS_Float; !! Default: 1.0
  Dist        : MANDATORY SS_Float; !! Distance along polyline
  Offset      : MANDATORY SS_Float; !! Vertical distance to polyline axis
END Pattern_Symbol;

CLASS LineStyle_Pattern EXTENDS LineStyle =
  PLength     : MANDATORY SS_Float;
  Symbols     : LIST OF Pattern_Symbol;
  !! after PLength the pattern is repeated
END LineStyle_Pattern;

!! Symbology interface
!! ++++++++++++++++++
```

```
!! Text interface
!! ==============

CLASS TextSign (EXTENDED) =
  Height      : MANDATORY SS_Float;
  Weight      : SS_Float; !! line width for line fonts
  Slanted     : BOOLEAN;
  Underlined  : BOOLEAN;
  Striked     : BOOLEAN;
  ClipBox     : SS_Float; !! Defines a rectangular surface around the text
                          !! with distance ClipBox from text.
PARAMETER
  Priority    : MANDATORY SS_Priority;
END TextSign;

ASSOCIATION TextSignFontAssoc =
  Font -- {1} Font;
  TextSign -- {0..*} TextSign;
MANDATORY CONSTRAINT
  Font -> Type == #text;
END TextSignFontAssoc;

ASSOCIATION TextSignColorAssoc =
  Color -- {0..1} Color;
  TextSign -- {0..*} TextSign;
END TextSignColorAssoc;

ASSOCIATION TextSignClipFontAssoc =
  ClipFont -- {0..1} Font;
  TextSign2 -- {0..*} TextSign;
END TextSignClipFontAssoc;

!! Symbol interface
!! ================

CLASS SymbolSign (EXTENDED) =
  Scale       : SS_Float;
  Rotation    : SS_Angle;
PARAMETER
  Priority    : MANDATORY SS_Priority;
END SymbolSign;

ASSOCIATION SymbolSignSymbolAssoc =
  Symbol -- {1} FontSymbol;
  SymbolSign -- {0..*} SymbolSign;
END SymbolSignSymbolAssoc;

ASSOCIATION SymbolSignClipSymbolAssoc =
  ClipSymbol -- {0..1} FontSymbol;
  SymbolSign2 -- {0..*} SymbolSign;
END SymbolSignClipSymbolAssoc;

ASSOCIATION SymbolSignColorAssoc =
  Color -- {0..1} Color;
  SymbolSign -- {0..*} SymbolSign;
END SymbolSignColorAssoc;

!! Polyline interface
!! ==================

CLASS PolylineSign (EXTENDED) =
  !! The parameter Width of the interface influences the width *and*
  !! the scale of start- and endsymbols.
PARAMETER
  Priority    : MANDATORY SS_Priority;
  Width       : SS_Float; !! Width of line symbology, default = 1.0
END PolylineSign;
```

```
      ASSOCIATION PolylineSignLineStyleAssoc =
        Style -- {1} LineStyle;
        PolylineSign -- {0..*} PolylineSign;
      ATTRIBUTE
        Offset      : SS_Float; !! Default 0.0
      END PolylineSignLineStyleAssoc;

      ASSOCIATION PolylineSignColorAssoc =
        Color -- {0..1} Color;
        PolylineSign -- {0..*} PolylineSign;
      END PolylineSignColorAssoc;

      ASSOCIATION PolylineSignClipStyleAssoc =
        ClipStyle -- {0..1} LineStyle; !! Used as a mask for clipping
        PolylineSign2 -- {0..*} PolylineSign;
      END PolylineSignClipStyleAssoc;

      ASSOCIATION PolylineSignStartSymbolAssoc =
        StartSymbol -- {0..1} SymbolSign; !! Symbol at start of line in opposite
                                          !! direction of line
        PolylineSign -- {0..*} PolylineSign;
      END PolylineSignStartSymbolAssoc;

      ASSOCIATION PolylineSignEndSymbolAssoc =
        EndSymbol -- {0..1} SymbolSign; !! Symbol at end of line in same
                                        !! direction as line
        PolylineSign3 -- {0..*} PolylineSign;
      END PolylineSignEndSymbolAssoc;

      !! Surface interface
      !! =================

      CLASS SurfaceSign (EXTENDED) =
        Clip        : (
          inside,
          outside
        );
        HatchOffset : SS_Float;
      PARAMETER
        Priority    : MANDATORY SS_Priority;
        HatchAng    : SS_Angle;  !! Default 0.0
        HatchOrg    : SS_Coord2; !! Default 0.0/0.0, Anchor point for hatching
                                 !! or filling
      END SurfaceSign;

      ASSOCIATION SurfaceSignColorAssoc =
        FillColor -- {0..1} Color; !! Fill color
        SurfaceSign -- {0..*} SurfaceSign;
      END SurfaceSignColorAssoc;

      ASSOCIATION SurfaceSignBorderAssoc =
        Border -- {0..1} PolylineSign; !! Border symbology
        SurfaceSign -- {0..*} SurfaceSign;
      END SurfaceSignBorderAssoc;

      ASSOCIATION SurfaceSignHatchSymbAssoc =
        HatchSymb -- {0..1} PolylineSign; !! Hatch symbology
        SurfaceSign2 -- {0..*} SurfaceSign;
      END SurfaceSignHatchSymbAssoc;

    END StandardSigns;

  END StandardSymbology.
```

Exemple

Voir annexe C Le petit exemple Roads.

Annexe L (Information) Glossaire

Abréviations courantes, veuillez vous reporter aux définitions pour les abréviations du domaine de spécialité.

Abr.	Abréviation.
Art.	Article (d'un texte de loi).
Al.	Alinéa (d'un texte de loi).
Déf.	Définition.
de	deutsch (allemand).
en	english (anglais).
fr	français.
Syn.	Synonyme.
INTERLIS	La mention INTERLIS comme INTERLIS 2.6.4 signifie que des informations complémentaires sur ce terme figurent au point 2.6.4 du document de référence INTERLIS 2 (SN 612031).
→ A	Le terme A est défini sous A dans ce glossaire.

Définitions

Abstraction de données

Retirer, omettre des détails peu importants sur des données.

Syn. data abstraction (en); Datenabstraktion (de).

Remarque 1 : séparation du quoi ? (→ interface de classe, → type), du comment ? (→ classe, implémentation concrète). → généralisation et → spécialisation sont des principes d'abstraction possibles.

Remarque 2 : la mise en œuvre effective des → opérations et la construction interne de → l'objet ou → de l'élément structuré sont cachées, c.-à-d. que l'on considère les propriétés de façon abstraite sans tenir compte de l'implémentation effective.

Accès à une relation

Conditions préalables et possibilités de référencement d' → objets relationnels et d' → objets de → classes (usuelles) via des → chemins d'accès à l'aide des premiers nommés.

Agrégation

→ Relation vraie orientée entre une → classe supérieure et une → classe inférieure. A un tout (sur- → objet de la classe supérieure) sont affectées plusieurs parties (sous- → objets) de la classe inférieure. Plusieurs "tout" peuvent également être affectés à une partie. En copiant un tout, toutes les parties attribuées sont copiées en même temps. A la suppression d'un tout, les parties qui lui sont assignées continuent d'exister.

Remarque 1 : avec l'agrégation, on décrit la relation entre un tout et ses parties (exemple : auto / moteur). Le → rôle de la classe inférieure peut être décrit avec "fait-partie-de", "is-part-of" (en).

Remarque 2 : dans INTERLIS 2, l'agrégation est indiquée au moyen d'un losange (vide) (-<>), par analogie avec la notation de modèle conceptuel → UML.

Remarque 3 : cf. → composition.

Altitude

→ Altitude ellipsoïdique, → normale ou → orthométrique.

Altitude ellipsoïdique

Distance euclidienne d'un point par rapport à l'ellipsoïde mesurée le long de la normale à la surface passant par ce point.

Altitude normale

Longueur de la courbe pondérée, de gravité normale décrite par la trajectoire orthogonale de l'ellipsoïde normal passant par le point et séparant le zéro de l'ellipsoïde normal et le point.

Altitude orthométrique H

Longueur de la courbe suivant la direction (incurvée) de la verticale séparant le → géoïde et le point.

Altitude usuelle

Somme des mesures de nivellement (différences altimétriques) le long d'un cheminement de nivellement d'un point de la → cote 0 au point ayant l'altitude usuelle recherchée.

Application

(d'un espace A, définie au moyen d'un → système de coordonnées dans un autre espace Z, définie par un deuxième → système de coordonnées :) règle qui attribue à chaque point a de A exactement un point z de Z.

Remarque: la → transformation et la → conversion sont des applications particulières.

Argument

Valeur d'un → paramètre.

Association

→ relation vraie qui ne limite pas l'indépendance des → classes concernées. Les → objets assignés peuvent être copiés et effacés indépendamment les uns des autres.

Syn. association (en), Assoziation (de).

Remarque 1 : en INTERLIS 2, la → classe d'association est à disposition pour la description de l'association.

Remarque 2 : cf. aussi → attribut de référence, → agrégation et → composition.

Association bidirectionnelle

Voir la définition sous → association.

Attribut

Eléments de données (ou données) correspondant à une propriété spécifique → d'objets d'une → classe et → d'éléments structurés d'une → structure (INTERLIS 2.6.4). Un attribut possède un nom et un → domaine de valeurs lui est associé.

Syn. → Merkmal (de), attribute (en).

Remarque : chaque → objet d'une → classe contient de même un → élément de données d'un attribut auquel une → valeur individuelle est associée. Concrètement, un attribut correspond à la colonne d'une → table.

Attribut de condition

→ Attribut pour lequel une → condition de cohérence est formulée.

Attribut de référence

→ Relation qui n'est connue que du premier → objet de chaque paire d'objets de la → relation.

Syn. → relation unilatérale.

Attribut de signature

Syn. de → règle de dessin.

Attribut dérivé

→ Attribut dont le → domaine de valeurs est calculé par la règle d'une fonction (→ expression logique, calcul).

Remarque 1 : des attributs dérivés ne peuvent pas être modifiés directement.

Remarque 2 : avec INTERLIS 2, la règle d'une fonction est définie via une → fonction.

Syn. derived attribute (en) ; Abgeleitetes Attribut (de).

Attribut structuré

→ Attribut avec le type de données INTERLIS 2 BAG ou LIST.

Remarque : au contraire de la définition d'une composition à l'aide de la → classe d'association, les → éléments structurés d'un attribut structuré ne sont pas référençables, ce qui signifie qu'ils n'ont pas d'identité en dehors du sur-objet.

Banque de données

Unité de gestion logique pour le traitement et le stockage durable d' → objets.

Remarque : plusieurs banques de données peuvent être exploitées sur un même → système. Il est également possible qu'une banque de données soit répartie entre plusieurs → systèmes. Abr. BD.

Banque de données de mutation

→ Banque de données temporaire avec les → objets de laquelle des → mutations sont effectuées. Une banque de données de mutation reçoit ses → objets d'une → banque de données primaire à laquelle elle les restitue au terme du traitement (→ mise à jour).

Remarque : une banque de données de mutation peut être gérée sur le même → système que la → banque de données primaire (banque de données de mutation interne) ou sur un autre → système (banque de données de mutation externe).

Banque de données primaire

→ Banque de données dans laquelle les → objets de certains → thèmes d'un territoire donné sont gérés à long terme.

Banque de données secondaire

Copie d'un → état de banque de données d'une → banque de données primaire.

Remarque : en principe, la banque de données secondaire n'est pas gérée sur le même → système que la → banque de données primaire.

Bibliothèque de signatures

Réunion de → signatures graphiques structurées conformément à un → modèle de signatures.

Syn. Signaturenbibliothek, Symbolbibliothek (de), symbology library (en).

Remarque 1 : une bibliothèque de signatures est toujours un → conteneur, c.-à-d. un fichier XML.

Remarque 2 : une bibliothèque de signatures désigne généralement une réunion concrète de → signatures graphiques, spécifique à une application donnée.

Bibliothèque de symboles

Déf. cf. sous → bibliothèque de signatures.

Bord de carte

Domaine dans lequel le contenu de la carte est représenté.

Remarque : des domaines de recouvrement échelonnés peuvent être définis contre le bord extérieur.

Bord de plan (layout)

Description d'un plan par les → métadonnées titre, → légende, description de celui qui établit le plan, date d'établissement, description des caractères typographiques et la représentation graphique d'autres éléments, comme les systèmes d'axes de coordonnées et la flèche indiquant le nord.

Syn. Planlayout, Kartenrahmen (de); border (layout) of the plan (en).

Cardinalité

Nombre d' → objets de la → classe B (ou A) pouvant être affecté à un → objet de la → classe A (ou B) par l'intermédiaire de la → relation entre les → classes A et B.

Syn. Multiplizität, Kardinalität (de), cardinality, multiplicity (en).

Remarque : en langage → UML, la notion de → multiplicité est également utilisée dans ce cadre ; la "cardinalité" désigne alors le nombre concret de → relations d'objets entre → instances objets.

Catalogue de données

Syn. de → catalogue d'objets.

Catalogue d'objets

Enumération informelle de → classes avec des définitions du langage courant (nom et description de la → classe) des objets de données d'importance pour une application.

Abr. OK (de).

Syn. catalogue de données.

Remarque 1 : des indications sur le degré de détail, sur les exigences en matière de qualité (en particulier sur la qualité géométrique) et éventuellement sur les règles de saisie appartiennent au catalogue d'objet.

Remarque 2 : le catalogue d'objets est une étape préliminaire et un complément du → modèle de données conceptuel.

Catégorie de nom

Sous-ensemble des noms d'un → schéma de données conceptuel. Il existe trois catégories de noms, à savoir les → noms de type, → les noms de structure de données et les → noms de méta-objets.

Remarque : la catégorie de nom et l'→ élément de modélisation définissent l'→ espace nominal.

Chemin d'accès

Suite de noms d' → attributs et/ou de → classes et/ou de → rôles de → classes d'associations définissant un → objet ou la → valeur d'un → attribut, à traiter par l'intermédiaire d'une → expression logique.

Classe

Ensemble d' → objets aux propriétés et → opérations identiques. Chaque propriété est décrite par un → attribut, chaque → opération par sa → signature d'interface.

Syn. Objektklasse, Entitätsmenge, Objekttyp (de), feature_type, feature, class (en).

Remarque 1 : une classe décrite avec → INTERLIS 2 correspond à une classe → UML avec des attributs uniquement visibles.

Remarque 2 : voir aussi → classe supérieure, → classe inférieure, → table et → élément de classe.

Remarque 3 : une classe ne doit pas nécessairement contenir d'objets. On parle d'une → classe concrète si elle peut contenir des objets et d'une → classe abstraite dans le cas contraire.

Classe abstraite

→ Classe qui ne peut comprendre aucun → objet.

Remarque : une classe abstraite est toujours incomplète et constitue la base de → classes inférieures (autrement dit pour des spécialisations) dont la quantité d'objets ne doit ensuite pas être vide.

Syn. abstract class (en); Abstrakte Klasse (de).

Classe concrète

→ Classe qui peut contenir des → objets.

Syn. concrete class (en); konkrete Klasse (de).

Remarque: cf. → classe abstraite.

Classe d'association

→ Elément de classe pour décrire une → association, → agrégation ou → composition.

Classe d'interfaces

Syn. de classe d'interfaces de classes. Cf. → interface de classe pour la définition.

Classe d'objet

Syn. de → classe.

Classe de base

Syn. de → classe supérieure.

Classe de base

→ Classe dont les → objets participent à la formation d'une → vue.

Classe de condition

→ Classe pour laquelle une → condition de cohérence (règle d'intégrité) est formulée.

Classe implémentée

Module logiciel exécutable avec des → opérations mises en œuvre en guise de → méthodes.

Classe inférieure

Cf. → relation d'héritage pour la définition.

Syn. Unterklasse, Unterobjektklasse, Subobjektklasse (de), subclass (en).

Classe supérieure

Cf. → relation d'héritage pour la définition.

Syn. Superklasse (de), Oberklasse (de), super class (en).

Communauté de transfert

Communauté d' → émetteurs et de → récepteurs qui prennent part à un → transfert de données.

Compilateur INTERLIS

Programme déduisant le format de transfert INTERLIS associé à partir d'un → schéma de données en → IDDL. La correction syntaxique du → schéma de données est vérifiée à cette occasion (processus dit de Parsing). Cf. INTERLIS annexe A.

Composition

→ Relation vraie orientée entre une → classe supérieure et une → classe inférieure. A un tout (sur- → objet de la classe supérieure) sont assignées plusieurs parties (sous- → objets de la classe inférieure) alors qu'un tout, au plus, peut être attribué à une partie. En copiant un tout, toutes les parties attribuées sont copiées en même temps. En supprimant un tout, toutes les parties assignées sont également supprimées.

Syn. Komposition (de), composition (en).

Remarque 1 : les parties n'ont pas d'autonomie mais font fermement partie du tout. Les → classes concernées n'établissent donc pas de → relation d'égal à égal mais bien une hiérarchie tout/parties (en : consists-of).

Remarque 2 : dans INTERLIS 2, la composition est définie comme → classe d'association.

Remarque 3 : Cf. Attribut structuré.

Condition

Syn. de → condition de cohérence (règle d'intégrité).

Condition de cohérence (règle d'intégrité)

Limitation que des → objets doivent satisfaire.

Syn. Konsistenzbedingung, Bedingung, Randbedingung, Zusicherung (de), constraint (en); contrainte (fr).

Remarque : des conditions de cohérence particulières sont prédéfinies en INTERLIS 2. D'autres conditions de cohérence sont définissables formellement avec des → fonctions, des → expressions logiques ou des règles et doivent faire l'objet d'un contrat.

Conteneur

Collection d'→ objets appartenant à un → thème ou à ses → extensions.

Syn. Basket (en), Behälter (de).

Contrat

Accord conclu avec des producteurs d'outils logiciels.

Remarque : Par exemple si → INTERLIS exige 2 modèles de données dans lesquels des → Fonctions, → modèles de signature non prédéfinis ou des → types de forme de lignes non définis sont utilisés.

Conversion

→ Application d'un → système de coordonnées (ou de son espace) dans un autre → système de coordonnées (ou dans son espace), rigoureusement définie par des formules et leurs → paramètres.

Syn. conversion (en) ; Konversion (de).

Remarque : on utilise à l'occasion le terme de conversion comme synonyme de reformatage de fichiers de transfert.

Couche d'information

Ensemble non vide de → thèmes.

Date

Donnée temporelle (par exemple 2002-06-25).

Datum

Syn. → datum géodésique.

Datum géodésique

→ Système de coordonnées cartésien tridimensionnel dont les axes occupent une position et une orientation fixe par rapport au centre des masses et à l'axe de rotation de la Terre.

Syn. datum (de).

Définition d'attribut de signature

Recommandation dénommée et conceptuelle en termes de dessin dans une → définition graphique comme attribution vraie d'objets pour des → signatures graphiques.

Remarque : ne pas confondre avec → attribut d'une → classe dans un → modèle de signatures.

Définition graphique

→ Elément de classe d'un → thème graphique, c.-à-d. que tout → thème graphique d'une → représentation graphique est une réunion de définitions graphiques (et non de → classes !). Toute définition graphique appartient à une → classe (BASED ON) du → thème des données correspondantes, assigne une ou plusieurs → signatures graphiques aux → objets de cette → classe au moyen de → règles de dessin et définit les arguments des → signatures graphiques conformément aux → données des → objets.

Syn. graphic definition (en).

Description de données

Syn. non univoque → schéma de données et → modèle de données.

Description graphique

Syn. de → représentation graphique.

Diagramme de classes

Représentation graphique de → classes et de leurs → relations.

Syn. class diagram (en); Klassendiagramm (de).

Domaine de définition d'un nom

→ Espace nominal de la → catégorie nominale de ce nom vers l'→ élément de modélisation dans lequel le nom est défini.

Remarque 1 : dans le domaine de définition d'un nom, chaque nom ne peut avoir qu'une définition/portée. En revanche, le même nom peut par exemple être défini une seule fois dans l' → espace nominal de chaque → catégorie de nom du même → élément de modélisation.

Remarque 2 : le domaine de définition d'un nom fait partie du → domaine de visibilité d'un nom.

Domaine de valeurs

Ensemble d'→ éléments de données similaires. Un élément de donnée d'un domaine de valeurs s'appelle une → valeur.

Syn. → Type de données.

Remarque 1 : cf. → type de donnée de base.

Remarque 2 : un domaine de valeurs peut également se composer des → éléments structurés d'une → sous-structure.

Domaine de valeurs structuré

Elément linguistique d'INTERLIS destiné à la description d' → attributs composés tels que la → date ou l'heure.

Domaine de visibilité d'un nom

Ensemble des → espaces nominaux à partir desquels le nom peut être référencé de façon non qualifiée. Le domaine de visibilité d'un nom se compose de son → domaine de définition et des → espaces nominaux de sa → catégorie de nom dans tous les → éléments de modélisation qui sont subordonnés hiérarchiquement à l'→ élément de modélisation de son → domaine de définition.

Remarque : à l'exception de l'espace nominal de son → domaine de définition, un nom peut être redéfini dans chaque → espace nominal de son domaine de visibilité. Cet → espace nominal devient de la sorte le nouveau → domaine de définition du nom. Ce nouveau → domaine de définition et le domaine de visibilité qui lui est assigné viennent se "surimposer" à une partie du domaine de visibilité originel en ce sens que seule la nouvelle définition/signification du nom s'applique encore dans ce domaine partiel (qui compose un arbre partiel de la hiérarchie des éléments de modélisation).

Elément

Terme fondamental de la théorie des ensembles. Un ensemble comprend des éléments.

Remarque : cf. → élément de modélisation ou → élément graphique.

Syn. → Instance.

Elément de classe

→ Elément de modélisation "du niveau de modélisation de la classe". Plus précisément : les éléments de classes sont appelés → classe, → structure, → classe d'association, → vue, → vue-projection et → définition graphique.

Elément de données

Veuillez consulter des ouvrages d'informatique pour la déf. Cf. également → domaine de valeurs.

Elément de modélisation

→ Elément de schéma particulier. Il existe trois éléments de modélisation, à savoir le → modèle de données, le → thème et l' → élément de classe.

Remarque : l'élément de modélisation et la → catégorie de nom définissent l' → espace nominal.

Elément de schéma

Schéma partiel d'un → schéma de données conceptuel portant un nom.

Remarque : tous les → éléments de modélisation sont des éléments de schémas.

Elément graphique

Représentation graphique d'un → objet tenant compte de la géométrie planimétrique et d'autres → attributs de cet objet, prête pour l'impression moyennant un périphérique adapté, après un éventuel traitement.

Syn. Grafikobjekt (de), graphic element (en).

Elément structuré

→ Données d'un objet du monde réel associées à des → opérations pouvant être exécutées avec ces → données sans toutefois pouvoir les modifier et dépourvues d' → identification d'objet.

Remarque : un élément structuré est l' → instance d'une → structure.

Elément thématique

Les → éléments de modélisation suivants sont des éléments thématiques : → Topic, → Pattern.

Emetteur

Déf. cf. → transfert de données.

Ensemble d'entités

Syn. de → classe.

Entité

Syn. d' → objet.

Espace nominal

Ensemble des noms (univoques) d'une → catégorie de noms dans un → élément de modélisation.

Syn. namespace (en).

Remarque : l'espace nominal est requis pour l'établissement du → domaine de définition et du → domaine de visibilité d'un nom.

Etat de banque de données

Ensemble de toutes les → données et → relations d'une → banque de données à un instant donné. Un nom est associé à tout état de banque de données.

Remarque : une → banque de données est transférée (→ mise à jour) d'un état de banque de données dans un autre par l'intermédiaire d'une ou de plusieurs → mutations.

Expression

Syn. d' → expression logique.

Expression logique

Prédicats liés par des opérateurs booléens.

Syn. logical expression (en), logischer Ausdruck (de).

Extension

Syn. de → spécialisation.

Feature

Syn. de → objet resp. souvent aussi de → classe.

Feature type

Syn. de → classe.

Fichier

Veuillez consulter des ouvrages d'informatique pour la déf.

Syn. file (en) ; Datei (de).

Fichier de transfert

→ Fichier préparé pour le → transfert de données dans le → format de transfert approprié.

Fonction

→ Application de → domaines de valeurs de paramètres d'entrée dans le → domaine de valeurs d'un paramètre de sortie via une règle de calcul préétablie (→ paramètres).

Syn. function (en); Funktion (de).

Remarque : certaines fonctions sont prédéfinies dans INTERLIS 2, d'autres doivent être réglées dans le cadre d'un contrat.

Format de transfert

Subdivision d'un → fichier de transfert en champs de données.

Syn. Format.

Généralisation

→ Rôle de la → classe supérieure dans une → relation d'héritage.

Syn. generalization (en); Generalisierung (de).

Remarque 1 : on utilise parfois la généralisation comme synonyme d' → héritage (même si on entend par là, la relation inverse).

Remarque 2 : en cartographie, on appelle généralisation toutes les activités qui résultent de la projection à une échelle plus petite d'objets de la réalité.

Géoïde

Surface équipotentielle du champ de gravitation.

Remarque : le géoïde fournit un modèle terrestre physique qui s'adapte au champ de gravitation de la Terre. Sa forme est irrégulière puisqu'il tient compte de la répartition irrégulière de la masse terrestre. Il peut être compris comme la surface du niveau moyen des mers prolongée sous les continents.

Héritage

Méthode visant à définir des → relations d'héritage entre des → classes supérieures et des → classes inférieures. Elles sont une → spécialisation de classe et une → spécialisation d'attribut.

Syn. inheritance (en) ; Vererbung (de).

Remarque 1 : concrètement, des → classes inférieures procèdent de la même idée ; elles ont les mêmes propriétés que leurs classes supérieures et les spécialisent.

Remarque 2 : on établit une distinction entre → héritage simple et → héritage multiple. Dans le premier cas (en : single inheritance; de : Einfachvererbung), une classe inférieure hérite seulement d'une classe supérieure directe. Avec l'héritage multiple, une classe hérite de plusieurs classes supérieures.

Remarque 3 : INTERLIS 2 n'autorise qu'un héritage simple (comme Java).

Héritage multiple

→ Relation d'héritage qui assigne plus d'une → classe supérieure à une → classe inférieure.

Remarque : l'héritage multiple n'est pas prévu dans INTERLIS.

Héritage simple

Déf. cf. sous → héritage.

ILI

Abréviation d'INTERLIS.

Remarque: Est aussi souvent un complément de nom de → fichiers qui contiennent un → schéma de données décrit en → INTERLIS (Version 1 et 2).

IDDL

Abréviation de INTERLIS-Data Description Language = → langage de description de données INTERLIS.

Identité d'objet

Syn. d' → identification d'objet.

Identificateur

Syn. d' → identification.

Identificateur d'objet

Syn. d' → identification d'objet.

Identification

→ Attribut ou combinaison d'attributs dont la → valeur caractérise de façon univoque un → objet dans sa → classe.

Abr. ID.

Syn. identifiant, identité.

Remarque : au sein d'un → fichier de transfert INTERLIS 2, chaque objet reçoit, outre les valeurs d'attribut décrites dans le → schéma de données, une identification (identité) qui le caractérise de façon univoque au sein de ce fichier, appelée identification de transfert (TID). Si une telle TID est une identification → générale et → stable, alors on l'appelle identification d'objet (OID).

Identification d'objet

→ Identification → générale et → stable.

Abr. OID.

Syn. Objektidentifikator, Objektidentität (de), object identifier (en), object identity (en).

Remarque 1 : normalement, l'identification d'objet n'est modifiée que par un système, pas par l'utilisateur. C'est une propriété qui différencie un objet de tous les autres, même en cas de possession éventuelle des mêmes valeurs d'attribut.

Remarque 2 : cf. → identification de transfert.

Remarque 3 : L'annexe E du manuel de référence INTERLIS 2 contient une proposition pour une identification d'objet.

Identification de transfert

Déf. cf. → Identification.

Abr. TID.

Identification générale

→ Identification univoque pour tous les objets (modélisés) d'une → communauté de transfert.

Remarque : Voir aussi → identification d'objet.

Identification stable

→ Identification non tributaire du temps, autrement dit qui ne peut être modifiée pendant le cycle de vie d'un → objet. L'identification stable d'un → objet supprimé ne doit plus être utilisée.

Remarque : Cf. → Identification d'objet.

Identité

Syn. d' → identification.

Instance

Syn. d' → élément (concret) d'un ensemble (abstraction).

Syn. Instanz (de) ; instance (en).

Remarque : exemples d'instance : une → valeur est une instance d'un → type de données. Un → objet est une instance d'une → classe. Un → conteneur est une instance d'un → thème. Une paire d'objets est une instance d'une → classe d'association.

Instance objet

Syn. d' → objet.

Intégrité référentielle

Règle qui fixe ce qui se passe avec une → relation d'objet ou avec les → objets concernés lorsque l'un d'eux ou la relation elle-même est supprimée.

Syn. referential integrity (en).

Intensité

Rattachement des parties (sous- → objets de la → classe inférieure) au tout (sur- → objet de la → classe supérieure) dans le cas d'une → relation vraie.

Interface

Syn. non univoque d' → interface de classe, d' → interface utilisateur ou d' → interface de données.

Syn. Schnittstelle (de), interface (en).

Interface de classe

Accès à une partie ou à la totalité des → opérations d'une → classe.

Syn. Schnittstelle (de), interface (en, fr).

Remarque 1 : une → classe peut posséder plusieurs interfaces de classes. Pour chacune de celles-ci, une → classe d'interfaces séparée peut être définie. Le → schéma → conceptuel d'une → classe d'interfaces ne contient que des → signatures d'interfaces.

Remarque 2 : cf. également → interface utilisateur et → interface de données.

Interface de données

Programme de changement de format de → fichiers de transfert ou → protocole pour le → transfert de données.

Syn. → interface.

Remarque: Voir aussi → interface de classe et → interface utilisateur.

Interface utilisateur

Interface permettant la gestion et l'exécution d'un programme informatique par l'utilisateur.

Syn. → Schnittstelle (de), graphic user interface (en).

Remarque : cf. également → interface de classe et → interface de données.

INTERLIS 2

→ Mécanisme de transfert de données prévu pour des géodonnées et se composant du → langage de description de données INTERLIS (→ IDDL), du format de transfert INTERLIS-XML (IXML) et de règles de déduction d'IXML pour une structure de données décrite via → IDDL. → IDDL, IXML et les règles de conversion sont définies dans la → norme suisse SN 612031.

Abr. "INTER Land-Informations-Systeme" (c.-à-d. "entre les → SIG").

Langage de description de données

Langage formel pour la description exacte de données.

Syn. Data description language (DDL), conceptual schema language (CSL).

Langage de description de données INTERLIS

→ Langage de description de données (conceptuel) du → mécanisme de transfert de données INTERLIS.

Syn. INTERLIS Data Description Language (en abrégé → IDDL).

Remarque : un → schéma de données décrit par → IDDL peut être stocké sous forme de fichier (texte). La terminaison "ILI" est d'usage pour les extensions de tels fichiers schémas. Exemple : le fichier schéma du modèle de données de la mensuration officielle est appelé MD01MO.ILI.

Layer

Désignation courante dans le domaine de la DAO pour le regroupement de → données graphiques d'un certain → type. Occasionnellement utilisée dans le domaine des → SIG pour désigner un → thème.

Légende

Ecritures et explication d'une carte ou d'un plan et des → signatures graphiques qui sont utilisées pour ce faire.

Syn. legend (en).

Remarque : voir aussi → représentation graphique et → bibliothèque de signatures.

Livraison complémentaire

→ Transfert de données → intégral ou → incrémentiel d'un → état de banque de données de la → banque de données primaire vers une → banque de données secondaire.

Remarque 1 : une livraison complémentaire se déroule toujours de façon séquentielle, c.-à-d. qu'une → banque de données secondaire ne reçoit jamais plusieurs → livraisons complémentaires simultanément.

Remarque 2 : cf. également → synchronisation.

Maquette maître

→ schéma conceptuel visant à décrire des éléments fondamentaux, autrement dit des structures de données qui peuvent prendre place dans divers → thèmes.

Syn. Muster, Anwendungsmuster (de); design pattern, pattern (en).

Remarque : dans INTERLIS 2, le mot-clé PATTERN sert à définir les maquettes maîtres, le mot-clé USES sert à l'utilisation de maquettes maîtres.

Mécanisme de transfert de données

→ Langage de description de données (conceptuel), → format de transfert (physique) et règles de déduction d'un tel → format de transfert des données décrite à l'aide du → langage de description de données.

Métadonnées

→ Données sur des → données.

Syn. metadata (en), Metadaten (de).

Remarque : en géomatique surtout, les métadonnées sont des données qui désignent notamment la description d'objets dans le langage courant, la saisie des objets, leur subdivision, la référence spatiale, la qualité, la disponibilité, l'origine, etc.

Métamodèle

→ Modèle de → métadonnées.

Méta-objet

→ Objet dont la concrétisation dans le monde réel est un ensemble d‘ → objets.

Remarque 1 : un méta-objet se compose de → métadonnées. Il existe des méta-objets pour des → objets isolés et/ou pour tous les → objets d'un → élément de modélisation.

Remarque 2 : les → métadonnées relatives aux → valeurs d‘ → attributs d‘ → objets sont des → attributs supplémentaires de la → classe de ces → objets.

Méthode

Implémentation d'une → opération par une suite d'instructions (autrement dit par un programme).

Syn. method (en); Methode (de).

Remarque : terme à plusieurs sens. Souvent utilisé comme syn. d'→ opération.

Mise à jour

Une ou plusieurs → mutations d'une → banque de données primaire. Une mise à jour conduit la → banque de données primaire d'un → état de banque de données vers le suivant.

Remarque : les → mutations de la → banque de données primaire peuvent être effectuées simultanément. La → banque de données primaire doit garantir la cohérence du résultat en cas de → mutations simultanées.

Modèle

Syn. de → modèle de données.

Remarque: La modélisation orientée objet fait la distinction entre un modèle-objet (synonyme pour la partie d'un → schéma de données qui décrit le contenu et l'organisation des → données) et un modèle d'application (synonyme pour la partie du → schéma de données qui décrit les → opérations qui seront appliquées aux → données.

Modèle d'application

Syn. de → modèle.

Modèle de données

Description exacte de données (appelée → schéma de données → conceptuel) complète et close. Le modèle de données est l' → élément de modélisation hiérarchiquement le plus élevé.

Syn. modèle, description de données.

Remarque 1 : attention ! Dans la théorie des banques de données, le modèle de données est couramment synonyme de formalisme conceptuel (ce qui signifie qu'un modèle de données est considéré comme une méthode pour la production d'un → schéma conceptuel).

Remarque 2 : un modèle de données est composé d'au moins un → thème.

Remarque 3 : un modèle de données est désigné par le mot-clé MODEL dans INTERLIS 2. Le → paquet (Package) qui correspond au modèle de données est au-dessus de tous les paquets qui correspondent aux → thèmes d'un modèle de données.

Modèle de gravité

Description du champ de gravité terrestre.

Modèle de signatures

→ Schéma → conceptuel décrivant des → signatures graphiques et les → paramètres qui leur sont associées.

Syn. Symbologiemodell (de), symbology model (en).

Remarque 1 : des → contrats sont requis pour les modèles de signatures.

Remarque 2 : l'annexe K du manuel de référence d'INTERLIS 2 contient une suggestion de modèle de signatures étendu.

Remarque 3 : cf. également → bibliothèque de signatures.

Modèle graphique

Syn. de → représentation graphique.

Multiplicité

Syn. de → cardinalité.

Mutation

→ Opération effectuée sur une → banque de données en vue de la conservation de sa cohérence.

Nom d'attribut

→ Catégorie de nom composée de → nom de paramètre graphique, → nom d'attribut, → nom de paramètre.

Nom de composantes

→ Catégorie de noms composée des noms des → paramètres d'exécution, → d'attributs, → de règles de dessin, → de paramètres, → de rôles, → d'accès aux relations et → de vues de base.

Nom de méta-objet

→ Catégorie de noms comprenant exclusivement des noms de → méta-objets.

Nom de structure de données

→ Catégorie de nom composée de → nom de thème, → nom de classe, → nom d'association, → nom de vue, → nom de projection de vue, → nom de définition graphique et → nom de vue de base.

Nom de type

→ Catégorie de nom composée des noms de → thèmes, → classes, → associations, → vues, → définitions graphiques, → conteneurs, → unités, → fonctions, → types de formes de lignes, → domaines de valeurs, → structures.

Norme

Une norme 'de jure' (ou norme) est une prescription technique fixée par des associations nationales et internationales de normalisation. Une norme 'de facto' est une prescription technique jouissant d'une large reconnaissance et utilisée par le plus grand nombre, moins contraignante qu'une norme 'de jure'.

Syn. standard (en), Norm (de).

Remarque 1 : une loi est une prescription au-dessus d'une norme 'de jure' ou 'de facto'.

Remarque 2 : en allemand, "standard" est synonyme de norme 'de facto'. En anglais, le terme de norme se traduit par 'standard' dans tous les cas de figure, ce qui peut être source d'ambiguïtés.

Objet

Données d'un élément du monde réel couplées avec les → opérations qui peuvent être exécutées avec ces données et avec une → identification d'objet.

Syn. Objekt, Entität, Tupel, Objektinstanz (de), object instance, feature (en), instance objet (fr).

Remarque 1 : cf. → instance → classe.

Remarque 2 : au contraire d'une → valeur, un objet a une identité, existe dans l'espace et le temps, est modifiable tout en conservant son identité, et peut être utilisé conjointement par des renvois. Un objet est concret. Il est lié à l'existence de choses réelles.

Remarque 3 : dans la bibliographie orientée objet, on trouve la description imagée suivante : un objet est une unité existant concrètement avec une identité propre (inaltérable) et des limites définies (au sens figuré), qui retient l'état et le comportement. L'état est représenté par des → attributs et des → relations, le comportement par des → opérations. Chaque objet appartient à une → classe exactement. La structure définie de leurs attributs vaut de la même manière pour tous les objets d'une → classe, tout comme le comportement. Les → valeurs des attributs sont toutefois individuelles pour chaque objet.

Objet de signature

Syn. de → signature graphique.

Objet graphique

Syn. d‘ → élément graphique.

Objet relationnel

Cf. → relation pour la définition.

OID

Abr. → identification d’objet.

Ontologie

Syn. de → schéma de données.

Remarque 1: L'ontologie est "une spécification formelle explicite d'un concept commun (englais : shared)", ou, dit de manière illustrée, un classeur (repository) de concepts.

Remarque 2: L'ontologie utilisie un langage UML/OCL ou un langage spécifique, tel que DAM/OIL (DARPA Agent Markup Language + Ontology Interchange Language). L'ontologie se compose typiquement d'un → schàma de données conceptuel, d'une hiérarchie taxonomique provenant d'un schéma de données conceptuel, d'une hiérarchie taxonomique de → classes (dictionnaire, thésaurus) et d'axiomes qui restreignent les interprétations possibles des termes définis (généralement dans un langage logique). L'ontologie doit (à l'avenir) trouver une utilisation comme abstraction supérieure dans l'utilisation de → schémas de données pour la spécifiqcation de software et pour la communication entre les humains.

Opération

Application des domaines de valeur d’attributs d’une → classe et/ou des → domaines de valeurs de paramètres d’entrée dans le domaine de valeur d’un paramètre de sortie.

Remarque 1 : l’implémentation d’un objet par une suite d’instructions (autrement dit par un programme) s’appelle une méthode.

Remarque 2 : la description d’une opération s’appelle une → signature d’interface et se compose de noms d’opérations et de la description des → paramètres.

Opération de vue

Règle de déf. de nouveaux → objets à partir des → objets de → classes de base ou de → vues de base.

Remarque : Les opérations de vues d' → INTERLIS 2 sont la projection (projection), la jonction (join), l'union (union), l'agrégation (aggregation) et l'inspection (inspection). Ensuite, la quantité d'objet peut encore être limitée avec une sélection (selection).

Optionnel

Facultatif, n'a pas impérativement à être présent ou utilisable. Contraire : non-optionnel, c.-à-d. obligatoire.

Remarque 1 : les → attributs sont optionnels lorsqu'ils n'est pas requis qu'ils soient obligatoires (mandatory). Le mot-clé MANDATORY est à disposition dans → IDDL pour les → attributs obligatoires.

Remarque 2 : dans → IDDL, "non obligatoirement présent" se rapporte au → fichier de transfert.

Paramètre d'exécution

→ Paramètre dont la → valeur est mise à disposition par un système de traitement, d'exploitation ou de représentation lors de l'exécution de la tâche requise.

Remarque : Comme par exemple, échelle de représentation, → date.

Paramètre graphique

Syn. de → paramètre d'une → signature graphique.

Paramètres

Eléments de données (ou données) dont la → valeur est transmise à une → fonction, une → opération ou un → méta-objet et/ou restitués par des → fonctions ou des → opérations. A chaque paramètre est associé un nom, un → domaine de valeurs et , dans le cas de → fonctions ou d' → opérations, une direction de transmission (in, out, inout). La → valeur concrète d'un paramètre est appelée un → argument.

Remarque 1 : cf. également → paramètre d'exécution.

Remarque 2 : les paramètres servent à la description des propriétés de → méta-objets ne concernant pas le → méta-objet lui-même mais son utilisation.

Paquet

Elément linguistique UML pour décrire des → modèles, → des thèmes et des parties de thèmes.

Syn. Paket (de), package (en).

Remarque 1 : un paquet définit un → espace nominal, ce qui veut dire qu'au sein d'un paquet les noms des schémas intégrés doivent être univoques. Chaque élément du schéma peut être référencé dans d'autres paquets, mais appartient exactement à un paquet (origine).

Remarque 2 : dans le langage UML, les paquets peuvent à leur tour contenir des paquets. Le paquet supérieur comprend le système complet correspondant au modèle de données d'INTERLIS 2.

Plan (miroir du plan)

Domaine dans lequel le contenu d'un plan sera représenté.

Syn. Carte (miroir de la carte).

Remarque : En bordure extérieure du plan, on peut décrire des bandes de recouvrement.

Polymorphisme d'objets

Des → objets d'une → extension peuvent aussi être trouvés partout où des → objets d'une → classe de base sont attendus.

Syn. Teilmengen-Polymorphismus, Teilmengen-Polymorphie, Substitutionsprinzip (de), polymorphism (en).

Remarque 1 : Cf. → Polymorphisme d'opérations.

Remarque 2 : il est principalement fait référence au polymorphisme d' → objets dans → INTERLIS 2.

Polymorphisme d'opérations

La → signature d'interface permet à des → objets de classes → différentes de répondre à des noms d'opérations identiques (informations), c.-à-d. d'être traités par des → opérations portant des noms identiques.

Syn. Polymorphie (de), polymorphism (en).

Remarque 1 : Cf. → Polymorphisme d'objets.

Remarque 2 : Dans → INTERLIS 2 est surtout utilisé le polymorphisme d' → objets.

Projection cartographique

→ Conversion d'un espace (ellipsoïde ou sphère) dans un plan euclidien.

Projection de vue

→ Classe dont les → objets sont sélectionnés à partir des → objets d'une autre → classe, → vue ou → projection de vue en complétant leurs → attributs. Des → attributs (virtuels) supplémentaires dont les → valeurs sont fixées par des → fonctions peuvent en particulier être définis.

Syn. viewprojection (en).

Remarque 1 : des → extensions de projections de vues sont possibles. Cependant, leurs → objets restent toujours des sous-ensembles de l'ensemble des objets de la → classe de base, → vue de base ou projection de vue de base.

Remarque 2 : cf. également → élément de classe.

Propriété

Syn. d' → attribut.

Syn. property (en).

Protocole

Ensemble des → opérations appartenant à un ensemble de → classes.

Récepteur

Cf. → transfert de données pour la définition.

Syn. → système de destination.

Règle de dessin

Elément linguistique d'une → définition graphique. Une règle de dessin affecte une → signature graphique (aux → objets d'une) à une → classe et fixe les → arguments de signature graphique correspondants conformément aux → valeurs d'attributs (c.-à-d. aux → données) des → objets.

Syn. → attribut de signature.

Relation

Ensemble de paires d'objets (ou dans le cas général d'uplets à n objets, qui s'appellent aussi → objets relationnels). Le premier → objet de chaque paire appartient à une première → classe A, le deuxième à une deuxième classe B. A cet égard, l'attribution d'objets aux paires doit être prédéfinie. Elle ne doit donc qu'être définie, autrement dit modélisée. On établit une distinction entre relations vraies (surtout → association, → agrégation, → composition), → relation d'héritage et → attribut de référence.

Syn. relationship (en); Beziehung (de).

Remarque 1 : comme le concept des vues le montre, il est, à l'opposé, possible de calculer des attributions avec des algorithmes, sur la base de valeurs d'attributs, par exemple.

Remarque 2 : cf. → relation objet.

Remarque 3 : pour une relation vraie, l' → intensité et la → cardinalité sont définies.

Relation d'héritage

→ Relation orientée entre une → classe supérieure, appelée → classe supérieure et une → classe inférieure, appelée → classe inférieure définie par → héritage. Le → rôle de la classe supérieure s'appelle → généralisation, celui de la classe inférieure → spécialisation.

Remarque 1 : les → objets de la → classe supérieure sont des généralisations des objets de la → classe inférieure. Les objets de la → classe inférieure sont des restrictions (→ spécialisations, → extensions) des objets de la → classe supérieure.

Remarque 2 : la relation d'héritage est la relation de sous-ensemble, les → objets de la → classe inférieure composent un sous-ensemble des → objets de la → classe supérieure. Pour les → objets de la → classe inférieure, il ne s'agit donc pas de nouveaux → objets mais d'une partie ou d'une subdivision des → objets de la → classe supérieure. Les deux → objets d'une paire d'objets de la relation d'héritage ont le même → OID.

Relation d'objet

Deux → objets reliés entre eux par une → relation entre les → classes auxquelles ils appartiennent.

Syn. link (en).

Relation orientée

→ Agrégation, → composition ou → attribut de référence ou → relation d'héritage.

Relation unilatérale

Syn. d' → attribut de référence.

Relation vraie

Déf. cf. sous → relation.

Répliquer

Copier, l' → objet copié ne pouvant toutefois pas être modifié indépendamment de l'original.

Remarque : On l'utilise surtout avec la → mise à jour.

Représentation graphique

→ Schéma → conceptuel décrivant l'affectation de → signatures graphiques à des → objets et se composant de → thèmes graphiques. Les → objets peuvent être sélectionnés dans une → vue.

Syn. modèle graphique, description graphique.

Remarque 1 : une représentation graphique dans → INTERLIS 2 comprend des → thèmes graphiques correspondant chacun à un thème de données (DEPENDS ON). Un → thème graphique est une réunion de → définitions graphiques (et non de → classes !). Chaque → définition graphique appartient à une → classe ou à une → vue (BASED ON) du thème de données correspondant, affecte une → signature graphique à chacun des → objets de la → classe ou de la → vue au moyen de → règles de dessin et définit les → arguments de la → signature graphique, dans le respect des → données des → objets. Les → données de ces → signatures graphiques, à savoir leurs noms et leur représentation graphique sont stockés dans une → bibliothèque de signatures dont la structure de données est décrite dans un → modèle de signatures correspondant.

Remarque 2 : la représentation graphique peut elle-même contenir des → schémas de données (par exemple des → classes décrivant des positions de textes).

Rôle

Importance des → objets d'une → classe dans une → relation.

Remarque : dans une → relation vraie, le rôle de chaque → classe concernée est décrit par son nom, → son intensité et sa → cardinalité. Un → attribut de référence décrit le rôle de la classe avec cet attribut. Dans la → relation d'héritage, les rôles sont définis de manière implicite.

Schéma

Syn. de → schéma de données.

Schéma conceptuel

Syn. → schéma de données conceptuel. Déf. cf. → schéma de données, remarque 2.

Syn. conceptual schema (en); konzeptionelles Schema (de).

Schéma de données

Description du contenu et de la classification des données qui caractérisent un extrait de la réalité spécifique, en termes d'application, ainsi que des règles qui s'appliquent dans ce cadre et d'→ opérations qui peuvent être exécutées avec les données.

Syn. Description de données, schéma conceptuel, ontologie.

Remarque 1 : pluriel : schémas de données.

Remarque 2 : selon le niveau d'abstraction auquel on décrit les → données, on établit une distinction entre → schéma conceptuel, schéma logique et schéma physique. Pour formuler un schéma de données, il existe des → langages de description des données adaptés.

Remarque 3 : pour les → banques de données, le schéma logique, correspondant au schéma conceptuel et formulé selon les possibilités de structuration spécifiques du système, est aussi appelé schéma interne. Par ailleurs, les schémas logiques et physiques d'appareils périphériques ou de fichiers d'échange sont souvent désignés par "schémas externes ou schémas formatés".

SIG

Abr. de système d'information géographique.

Signature

Syn. non univoque de → signature d'interface et → signature graphique.

Signature d'interface

Description de l'appel d'une → opération se composant du nom de l' → opération, du → type de données et éventuellement du nom des → paramètres et de l'indication d'un type de données de retour.

Syn. Signatur (de).

Signature de cartes

Syn. de → Signature graphique.

Signature graphique

→ Données pour la représentation graphique d'→ objets encore indépendants de la géométrie en planimétrie et d'autres valeurs d'attributs de ces → objets. Les → paramètres de signatures graphiques sont également appelés → paramètres graphiques en abrégé.

Syn. Symbol, Kartensignatur, Signatur, Signaturobjekt (de), signature, symbol, style (en).

Remarque 1 : il existe quatre types de signatures graphiques : les écritures ou signatures de textes (également désignées parfois par écritures de texte ou simplement écritures), les symboles ponctuels (parfois appelés signatures de points ou plus simplement → symboles ou pictogrammes), les signatures de lignes et les signatures de surfaces (isolées).

Remarque 2 : dans → INTERLIS 2, des données et d'éventuels → paramètres d'une signature graphique sont décrites dans le → modèle des signatures et les → données correspondantes sont regroupées au sein d'une → bibliothèque de signatures. Les signatures graphiques sont référencées dans une → définition graphique par l'intermédiaire de noms de signatures graphiques et des → arguments adéquats sont définis pour d'éventuels → paramètres.

SN

Abréviation de Schweizer → Norm (norme suisse).

Sous-structure

→ Domaine de valeurs défini à l'aide d'une → structure.

Remarque : Cf. → Attribut structuré.

Spécialisation

→ Rôle de la → classe inférieure dans une → relation d'héritage, également et bien souvent synonyme d' → héritage.

Syn. Spezialisierung (de), Erweiterung (de), extension (en).

Remarque 1 : cf. → spécialisation de classe et → spécialisation d'attribut.

Remarque 2 : parce qu'il faut davantage de texte pour décrire la spécialisation de → classes ou d' → attributs que pour la → classe supérieure ou l'attribut originel, on parle souvent, également, d'extension (même terme en anglais), plutôt que de spécialisation.

Spécialisation d'attribut

Restriction du → domaine de valeurs d'un → attribut.

Remarque : la spécialisation d'attribut est également utilisée pour définir des → relations d'héritage.

Spécialisation de classes

Limitation d'une → classe par des → attributs, → relations, → conditions de cohérence ou → spécialisations d'attributs complémentaires.

Remarque : une spécialisation des classes est utilisée pour la déf. de → relations d'héritage.

Standard

Syn. en allemand de → norme 'de facto' et en anglais de → norme 'de facto' ou 'de jure'.

Structure

Ensemble d' → éléments structurés dont les propriétés et les → opérations sont identiques. Seules des → opérations ne modifiant pas les → données des → éléments structurés sont permises. Chaque propriété est décrite par un → attribut, chaque → opération par sa → signature d'interface.

Remarque 1 : les structures se rencontrent au sein d'attributs de types LIST ou BAG (→ sous-structure) ou n'existent que temporairement en tant que résultats de → fonctions.

Remarque 2 : cf. aussi → élément de classe.

Structure graphique

Eléments d'un → système de signes cartographiques qui règle l'interaction d' → éléments graphiques.

Remarque : concrètement, une structure graphique précise comment une carte particulière est conçue, par exemple la couleur dans laquelle des → éléments graphiques concrets sont imprimés à tel ou tel endroit sur le papier.

Symbole

Syn. de → signature graphique, symbole de langue ou symbole sémiotique.

Symbologie

Sous-ensemble d'éléments d'un → système de signes cartographiques composé de → signatures graphiques, d'écritures, de diagrammes, de demi-teintes.

Remarque : cf. → bibliothèque de signatures.

Synchronisation

Ajustement automatique et régulier des → états de banque de données de deux → banques de données.

Système

Ensemble de tous les composants appartenant à un poste de traitement des informations (matériel et logiciels), utilisés dans un but déterminé.

Système de coordonnées

Base d'un espace vectoriel euclidien ou image originelle de la base de l'espace vectoriel euclidien assignée en cas d'homomorphisme cartographique (cf. analyse vectorielle pour des détails).
Syn. coordinate system (en).

Remarque : du point de vue des → données, un système de coordonnées est défini par ses axes qui sont soit des droites (appelées LengthAXIS dans INTERLIS 2) soit des courbes gauches (AngleAXIS) suivant le genre de l'espace qu'ils permettent de mesurer.

Système de coordonnées cartésien

→ Système de coordonnées de l'espace euclidien dont les axes sont des droites perpendiculaires deux à deux.

Système de coordonnées sur l'ellipsoïde

→ Système de coordonnées sur la surface bidimensionnelle d'un ellipsoïde (de rotation) tridimensionnel.

Système de destination

Syn. → récepteur.

Système de référence

→ Système de coordonnées à la conclusion d'une suite de → systèmes de coordonnées et de → conversions, dans laquelle existe un seul → datum géodésique formant le début de la suite.

Syn. reference system (en), Referenzsystem (de).

Système de référence de coordonnées

Syn. de → système de référence.

Système de référence géodésique

Syn. de → datum géodésique.

Système de signes cartographiques

Ensemble de possibilités graphiques de représentation pour → signatures graphiques.

Syn. cartographic sign system (en).

Remarque 1 : un → élément graphique concret représenté à l'écran ou sur papier est le résultat d'un processus en plusieurs étapes à la faveur duquel des → objets sont sélectionnés (selection), représentés avec des → signatures graphiques (mapping), réunis, établis graphiquement (rendering) et représentés (display).

Remarque 2 : dans INTERLIS 2, les deux premiers niveaux sont réglés via une → représentation graphique, l'implémentation des autres étant l'affaire des systèmes ou de "pilotes" ; il existe parfois à ce sujet certains standards graphiques (tels PostScript, HPGL, OpenGL, Java2D, SVG).

Table

→ Classe pour les → objets de laquelle aucune → opération ne peut être définie explicitement.

Syn. relation.

Thème

Ensemble de → classes dont les → données vont, dans un certain sens, ensemble, lesquelles ont une relation, ou appartiennent au même organe de gestion des données, ou possède un rythme de mise à jour semblable. Les → instances de thèmes sont des → conteneurs.

Syn. topic (en), Thema (de).

Remarque 1 : un thème est décrit avec → INTERLIS 2 comme TOPIC. Un TOPIC est décrit en UML via un → paquet au sein d'un modèle de données avec la signification supplémentaire que

ce paquet (a) possède un → espace nominal propre et (b) peut être dépendant d'autres paquets (par exemple étendu). Un paquet UML assigné à un thème peut contenir d'autres paquets (emboîtés).

Remarque 2 : attention : un → layer, expression utilisée dans le domaine de la DAO pour désigner une "couche", désigne la réunion de → données graphiques. Un thème peut englober plusieurs → couches (graphiques) ainsi que des attributs structurés.

Thème graphique

Déf. → représentation graphique.

TID

Abr. → identification de transfert.

Topic

Syn. de → thème.

Trait de rappel

→ Elément graphique linéaire qui relie entre eux deux éléments graphiques ou un → élément graphique et une écriture.

Syn. help line (en) ; Hilfslinie (de).

Remarque : le trait de rappel représente fréquemment une relation entre une signature de ligne, de surface et une écriture associée ou encore les lignes de cotation y afférentes.

Transfert

Syn. de → transfert de données.

Transfert de données

Transfert de → données d'une → banque de données A vers une autre → banque de données Z. A est appelé système initial, source, → émetteur, système émetteur, Z système cible, → récepteur, cible. La livraison des → données à transférer par le → système A est également appelée exportation, la prise en charge par le → système Z importation.

Syn. Tranfert, transmission de données.

Transfert de données incrémentiel

→ Transfert de données de la différence entre deux → états de banque de données, de l' → émetteur vers le → système de destination.

Transfert de données intégral

→ Transfert d'un → état de banque de données complet de l' → émetteur vers le → récepteur.

Transformation

→ Application d'un → système de coordonnées (ou de son espace) dans un autre → système de coordonnées (ou dans son espace), si la règle d'application (formule) est définie sur la base d'hypothèses et les → paramètres calculés par analyse statistique de mesures dans les deux → systèmes de coordonnées.

Syn. transformation (en/de).

Transformation de datum

→ Transformation d'un → datum géodésique (ou de l'espace défini par lui) dans un autre → datum géodésique (ou dans son espace).

Tupel

Syn. de → Objet.

Type

Syn. de → type de données (autrement dit → domaine de valeurs), d' → interface de classe ou de → signature d'interface.

Type d'objet

Syn. de → Classe.

Type de données

Syn. de → domaine de valeurs.

Type de données de base

→ Domaine de valeurs prédéfini comme TEXT ou BOOLEAN (INTERLIS 2).

Type de forme de ligne

Forme des portions de courbes composant une polyligne (segment de droite, arc de cercle, autres géométries de liaison). Vous voudrez bien vous reporter à INTERLIS 2.8.11 et 2.8.12 pour la déf. des géométries d'objets acceptées par INTERLIS 2.

UML

Abréviation de Unified Modeling Language.

Cf. www.OMG.org/ pour la définition.

Unité

Elément de base d'une échelle de mesure (exemples : mètre, seconde).

Syn. unit (en) ; Einheit (de).

Valeur

→ Elément de donnée d'un → domaine de valeurs.

Vue

→ Classe dont les → objets résultent de la combinaison et de la sélection (par des → opérations de vues pour être précis) d' → objets d'autres → classes ou vues.

Syn. view (en).

Remarque 1 : les → objets d'une vue n'en sont pas "originaires", puisqu'ils ne correspondent pas directement à un objet du monde réel. Une vue est par conséquent un genre de → classe virtuelle.

Remarque 2 : cf. également → élément de classe.

Vue de base

→ Vue dont les → objets participent à la formation d'une nouvelle → vue.