INTERLIS 2 –

Referenzhandbuch

Ausgabe vom 2003-05-13 (deutsch)

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

Inhalt

Figurenverzeichnis

Vorwort

Dieses Referenzhandbuch richtet sich an Fachleute, die sich mit Informationssystemen, insbesondere mit Geo-Informationssystemen oder Landinformationssystemen befassen. Es dürfte insbesondere auch für Entscheidungsträger von Interesse sein, für die der sorgfältige Umgang mit den Daten ein Anliegen ist.

Im Jahre 1991 erschien erstmals "INTERLIS - ein Datenaustausch-Mechanismus für Land-Informationssysteme". Hauptziel und Zweck von INTERLIS ist die möglichst präzise Beschreibung von Daten. Dieser Mechanismus besteht aus einer konzeptionellen Beschreibungssprache und einem sequentiellen Transferformat mit besonderer Berücksichtigung raumbezogener Daten (kurz Geodaten). Damit wird die Kompatibilität unter den Systemen und eine langfristige Verfügbarkeit, d.h. Archivierung und Dokumentation der Daten ermöglicht. Wird INTERLIS bei Entscheidungs-, Planungs- und Verwaltungs-Prozessen sinnvoll eingesetzt, kann daraus ein grosser Nutzen entstehen. Oft lassen sich - zum Beispiel durch Mehrfachverwendung und einheitlicher Abgabe von dokumentierten und geprüften Daten - erhebliche Einsparungen erzielen.

Fünf Jahre nach seiner Veröffentlichung ist INTERLIS, das wir rückblickend Version 1, bzw. INTERLIS 1 nennen, aus seinem "Dornröschenschlaf" geweckt worden. Seither ist eine beachtliche Palette von Softwarewerkzeugen verfügbar, mit denen die Benutzer in INTERLIS beschriebene und codierte Geodaten verarbeiten können. INTERLIS ist aus einem Bedürfnis der amtlichen Vermessung heraus entstanden, sein Anwendungsspektrum ist aber wesentlich umfassender. Das zeigen die weit über hundert Datenmodelle und Projekte, die zehn Jahre nach der Publikation von INTERLIS damit arbeiten. Der Standard "INTERLIS 1" wird als Schweizer Norm SN 612030 - parallel zu seinen Nachfolge-Versionen - noch eine Weile von Nutzen sein.

Aufgrund von gestiegenen Benutzeranforderungen drängten sich verschiedene Erweiterungen von INTERLIS 1 auf, wie zum Beispiel die inkrementelle Nachlieferung, die strukturelle Objektorientierung oder die formale Beschreibung der grafischen Abbildung von Objekten. 1998 war der Beginn eines mehrjährigen Konsensprozesses, für den ein halbes Dutzend Fachleute aus Forschung, Verwaltung, Beratung und Softwareindustrie zusammengearbeitet haben. Herausgekommen ist ein Produkt, das man eine Erweiterung von INTERLIS 1 und gleichzeitig eine Synthese neuester Konzepte bezeichnen darf.

Beim INTERLIS 2-Referenzhandbuch wurde wieder darauf geachtet, dass nur das absolut Notwendigste festgelegt wird: Beispiele und Figuren erscheinen nur dort, wo der knappe Text sinnvoll ergänzt wird. Damit bleibt die Spezifikation übersichtlich und einfach implementierbar. Wenn einige Sprachelemente, wie z.B. Sichten oder Darstellungsbeschreibungen, anspruchsvoll wirken, so liegt das höchst wahrscheinlich nicht an INTERLIS selbst, sondern am komplexen Sachverhalt. Zur Lösung dieses Problems sind uns folgende Wege bekannt: gute Beispiele, Aus- und Weiterbildung, sowie so genannte "Profile", d.h. Untermengen von wohl definierten INTERLIS-Werkzeugfähigkeiten.

Für ein erstes Grundverständnis von den INTERLIS 2-Konzepten wird jedem Leser empfohlen, mindestens das Kapitel 1 Grundprinzipien durchzulesen.

Erweiterungen von INTERLIS 2 gegenüber INTERLIS 1

Die bestehende INTERLIS 1-Beschreibungssprache wird bis auf wenige Ausnahmen erweitert und nicht abgeändert. Erweitert wurden zum Beispiel die Möglichkeiten, Beziehungen zwischen Objekten zu beschreiben (eigentliche Beziehungen als Assoziationsklasse und Referenzattribute mit REFERENCE TO. Hinweis: Die "->"-Syntax des INTERLIS 1-Beziehungsattributs erhielt eine andere Bedeutung), wobei darauf geachtet wurde, dass der Übergang von INTERLIS 1 auf 2 nicht unnötig erschwert wird (vgl. Kapitel 2.7 Eigentliche Beziehungen). Eigentliche Beziehungen und Referenzattribute können neu mit bestimmten Bedingungen auf Objekte in anderen Behältern verweisen. Behälter ist ein neuer Begriff für die uns wohlbekannte Organisation von Objekten (d.h. von Daten, die Realweltobjekte beschreiben, auch

Objektinstanzen oder Instanzen genannt) in einer Datenbank. Verändert hat sich der Begriff Tabelle (TABLE), der neu Klasse (CLASS) heisst, entsprechend dem Übergang vom relationalen zum objekt-orientierten Formalismus. Ohne weitere Angaben gilt ein Attribut als fakultativ (OPTIONAL entfällt) und es muss angegeben werden, wenn es obligatorisch ist (MANDATORY). Ferner hat die Eindeutigkeitsbezeichnung (IDENT) ebenfalls einen neuen Namen bekommen (UNIQUE). Die neuen objekt-orientierten Konzepte umfassen Vererbung unter anderem von Themen, Klassen, Sichten, Darstellungsbeschreibungen und Wertebereichen. Weitere mächtige Erweiterungen sind Mengen-Datentypen (LIST, BAG), Konsistenzbedingungen, Datensichten, Darstellungsbeschreibungen, Beschreibung von Einheiten, Beschreibung von Meta-Objekten (Koordinatensysteme und Grafiksignaturen) und die inkrementelle Nachlieferung. Zudem sind spezielle, anwendungsspezifische Erweiterungen wie z.B. Funktionen und weitere Liniengeometrien definierbar. Dafür sollen aber Kontrakte, d.h. Vereinbarungen mit den Werkzeuganbietern eingegangen werden.

Neu übernimmt die eXtensible Markup Language (XML) die Codierung für das INTERLIS 2-Transferformat. Durch die absehbare internationale Verbreitung von XML erwarten wir eine gute Akzeptanz dieses Formats und eine grosse Anzahl kompatibler Softwareprodukte.

Für den mit INTERLIS 1 vertrauten Benutzer ändert sich nicht viel, solange er die neuen Konzepte, wie Objekt-Orientierung oder Darstellungsbeschreibung nicht nutzen will: er kann seine bisherigen Kenntnisse in INTERLIS 2 weiterhin einsetzen. Werkzeuge, wie der frei erhältliche INTERLIS 2-Compiler, unterstützen ihn beim Umsteigen. Diejenigen Hersteller, welche bereits bei der Implementierung von INTERLIS 1 auf flexible Konfigurationsmöglichkeiten und auf die Regeln der Software-Entwicklungskunst (z.B. Modularisierung und Abstraktion) geachtet haben, werden ihre bisherigen Investitionen erhalten können; durch frei erhältliche Programmbibliotheken können sich die Softwarehersteller auf die Anbindung ihrer Systeme an INTERLIS 2 konzentrieren.

Ausblick

INTERLIS 1 erschien zu einem Zeitpunkt, in dem beispielsweise die relationale Datendefinitionssprache SQL-92 noch nicht standardisiert und von Objektorientierung noch kaum die Rede war. Einige dieser heute etablierten Konzepte hat der Urheber von INTERLIS 1, Joseph Dorfschmid, in vorausschauender Weise in die Sprache einfliessen lassen. Mit dem Überarbeitungsprozess und der Einbindung objekt-orientierter und spezifischer Informatikkonzepte hat INTERLIS einen neuen Reifegrad erreicht. INTERLIS 2 kann damit heute – und nicht erst morgen – als effizientes Werkzeug eingesetzt werden.

Es ist uns bewusst, dass die Suche nach der universellen Datenbeschreibungssprache noch nicht abgeschlossen ist. Jedes Zuwarten wäre aber mit ähnlichen Folgen verbunden, wie der kurzsichtige Umgang mit den Ressourcen unserer Erde. INTERLIS hat es verdient, nicht nur als Austauschformat sondern auch als nachhaltiges Werkzeug wahrgenommen zu werden: Mit INTERLIS bekam die Forderung nach nachhaltigem Umgang mit der Technik einen Namen!

Jede Sprache muss auch erlernt und in eine Methodik eingebettet werden. So ist es klar, dass zur Ergänzung des vorliegenden Referenzhandbuchs einige Benutzerhandbücher folgen müssen.

Dank

Als Verantwortlicher für die Weiterentwicklung von INTERLIS und als Hauptredaktor dieses Dokuments hat Stefan Keller (Hochschule für Technik Rapperswil, vorher Eidgenössische Vermessungsdirektion) die Arbeiten an INTERLIS 2 von Beginn weg bis zu seinem Wechsel an die Fachhochschule vollumfänglich, anschliessend immer noch zu einem beträchtlichen Teil, betreut. Dafür möchten wir ihm herzlich danken. In diesen Dank einschliessen möchten wir auch das "INTERLIS 2-Kernteam", das eine hervorragende und einzigartige Arbeit geleistet hat. Zum diesem Team gehörten Joseph Dorfschmid (Adasys AG), Michael Germann (infoGrips GmbH), Hans Rudolf Gnägi (Eidgenössische Technische Hochschule), Jürg

Kaufmann (Kaufmann Consulting), René L'Eplattenier (Amt für Raumordnung und Vermessung des Kantons Zürich), Hugo Thalmann (a/m/t software service AG) sowie Sascha Brawer (Adasys AG), Claude Eisenhut (Eisenhut Informatik) und für die Koordination Rolf Zürcher (KOGIS).

Seit Anfang des Jahres 2002, liegt die Verantwortung für die Redaktion des Referenzhandbuchs – in enger Zusammenarbeit mit der Vermessungsdirektion - bei KOGIS, der Koordinationsstelle für Geoinformation und geographischen Informationssysteme beim Bund. Kurz nach diesem Verantwortungswechsel hat eine öffentliche Vernehmlassung der Schweizerischen Normenvereinigung SNV zu INTERLIS 2 stattgefunden. Eingegangen sind 109 Kommentare, die in den anschliessenden Monaten vom Kernteam geprüft und allenfalls umgesetzt werden mussten. Dies hat zu einigen grösseren Änderungen der Sprache gegenüber der Vernehmlassungsversion 2.1 vom 17. Oktober 2001 geführt, mit der Folge, dass INTERLIS 2 die interne Version 2.2 tragen wird. Ende November 2002 hat die Schlussabstimmung des INB / TK 151 stattgefunden, wo INTERLIS 2 zur Norm SN 612031 erklärt wurde. Nach letzten textlichen Überarbeitungen kann das Referenzhandbuch hiermit der Öffentlichkeit übergeben werden.

Kein Standard kann von Individuen alleine festgelegt werden, dazu ist die Hilfe von vielen Fachleuten nötig. Wir danken diesen professionellen Kräften an dieser Stelle und freuen uns auf innovative Produkte.

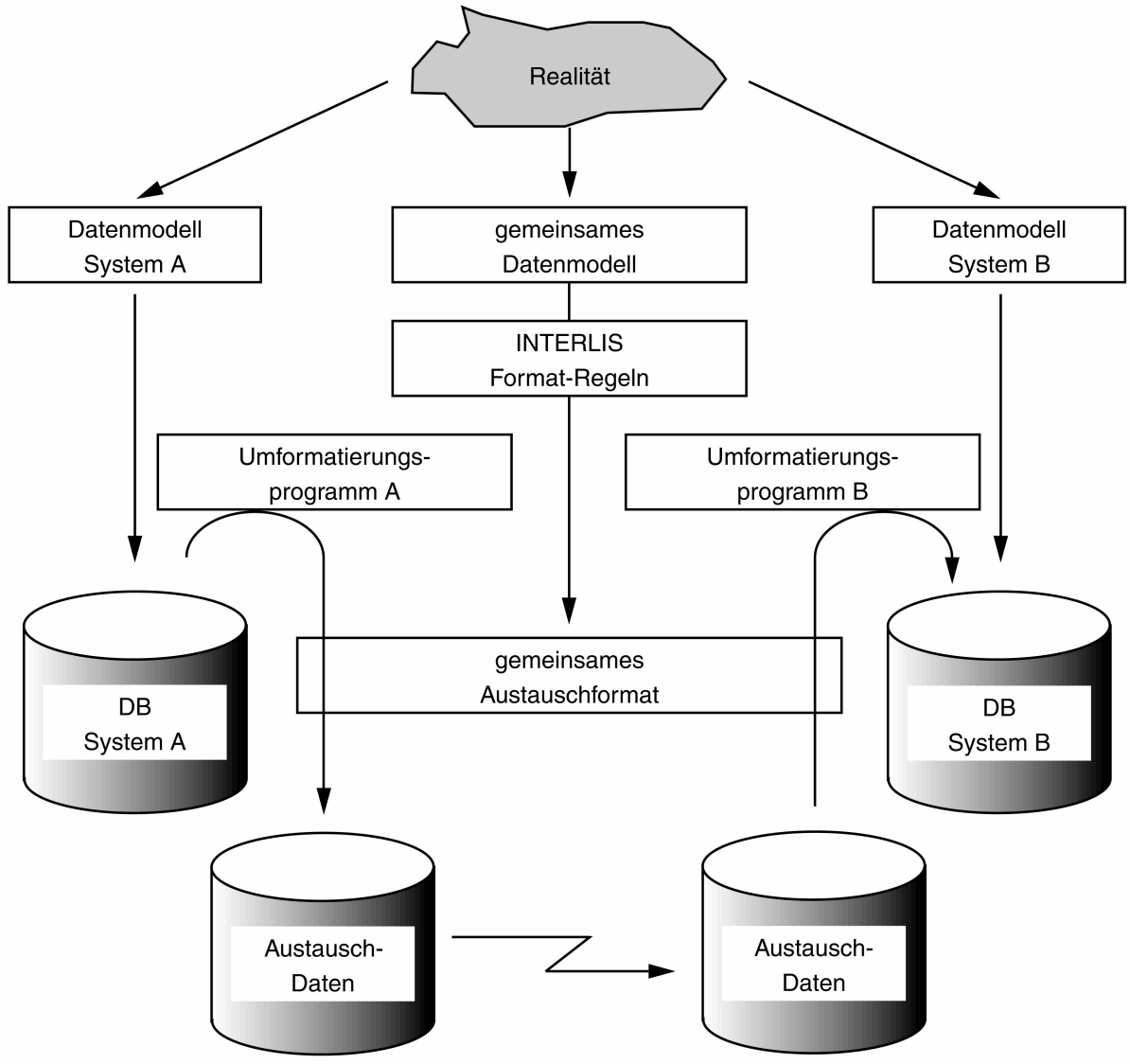
Wabern, April 2003

Rolf Zürcher

1 Grundprinzipien

1.1 Übersicht

INTERLIS dient der Zusammenarbeit von Informationssystemen, insbesondere von geografischen Informationssystemen oder Landinformationssystemen. Wie der Name sagt, steht INTERLIS zwischen (**INTER**) **L**and-**I**nfomations-**S**ystemen. Zentral hierfür ist, dass alle beteiligten Systeme eine klare Vorstellung von jenen Konzepten besitzen, die für die Zusammenarbeit relevant sind.



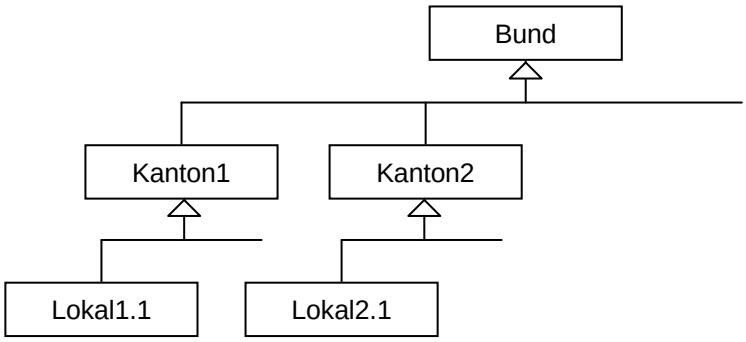
Figur 1: Datentransfer zwischen verschiedenen Datenbanken über gemeinsames Datenmodell (Datenschema) beschrieben mit gemeinsamer Datenbeschreibungssprache.

INTERLIS umfasst aus diesem Grund eine konzeptionelle Beschreibungssprache. Mit dieser Sprache kann der Ausschnitt der Realwelt beschrieben werden, der für eine bestimmte Anwendung von Interesse

ist. Eine solche Beschreibung heisst (konzeptionelles) Anwendungsmodell oder Anwendungsschema, bzw. kurz Modell oder Schema. Mit wenigen Konzepten mit genau definierter Bedeutung werden Klassen von Objekten mit ihren Eigenschaften und ihren Beziehungen beschrieben (modelliert). Zudem erlaubt die Beschreibungssprache von INTERLIS, Klassen von abgeleiteten Objekten einzuführen, die als Sichten auf andere Klassen von Objekten definiert werden. Sowohl echte als auch abgeleitete Klassen von Objekten können als Grundlage von Darstellungsbeschreibungen dienen, wobei INTERLIS eine strikte Trennung von Darstellungsbeschreibung und Beschreibung der zu Grunde liegenden Datenstruktur (Objektmodell) sicherstellt.

INTERLIS ist nicht auf eine bestimmte Anwendung ausgerichtet. Beim Entwurf, der sich an allgemeinen objekt-orientierten Prinzipien orientiert, wurde jedoch darauf geachtet, dass jene Konzepte besonders gut unterstützt werden, die für geografische Informationssysteme wichtig sind. So sind Koordinaten, Linien und Flächen als Datentypen Grundkonstrukte von INTERLIS, und es stehen Sprachelemente zur Beschreibung von Messgenauigkeiten und Einheiten zur Verfügung. Dennoch können durchaus auch nichtgeografische Anwendungen mit INTERLIS bearbeitet werden.

Aspekte, die für ein Anwendungsgebiet grundlegend sind, können in einem Basismodell beschrieben werden. Dieses Basismodell wird anschliessend z.B. gemäss den spezifischen Bedürfnissen eines Landes, in weiteren Stufen gemäss denen einer bestimmten Gegend (wie Kanton, Region oder Gemeinde) spezialisiert (Figur 2). INTERLIS 2 bietet als Mittel zur Spezialisierung die objekt-orientierten Konzepte der Vererbung und des Polymorphismus an. So ist sichergestellt, dass bereits erfolgte Definitionen nicht wiederholt werden müssen oder gar irrtümlicherweise in Frage gestellt werden können.



Figur 2: Spezialisierung der Modellierung eines Konzepts von Stufe Bund über kantonale (länderspezifische) bis lokale Stufe.

Grössere Anwendungen müssen nicht in einer einzigen Beschreibung definiert werden. Sie können vielmehr in mehrere Beschreibungseinheiten (Modelle, Schemas) aufgeteilt werden. Eine Beschreibungseinheit kann mehrere Themen umfassen. Im Interesse der Lesbarkeit der Modelle ist es auch möglich, Modelle als reine Übersetzungen von anderen Modellen zu definieren.

1.2 Nutzung von Modellen

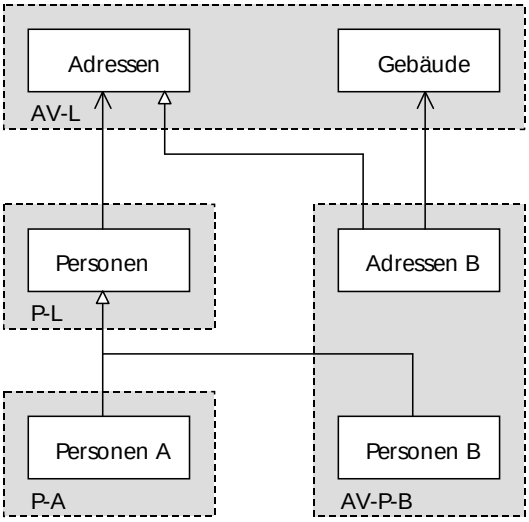
Ein INTERLIS-Modell (bzw. INTERLIS-Schema) ist primär eine Verständigungshilfe für Anwender. Die Sprache ist so gestaltet, dass ein Modell leicht von Menschen gelesen werden kann. Dennoch sind INTERLIS-Modelle präzis, eindeutig und ohne Missverständnisse interpretierbar. Die textuelle INTERLIS-Sprache bietet sich daher geradezu an als notwendige Ergänzung der grafischen Beschreibungssprache Unified Modeling Language (UML, www.omg.org/uml).

INTERLIS geht aber noch einen Schritt weiter: Da ein Modell eine formal klar definierte Bedeutung besitzt, ist die Implementation eines Dienstes in einem Computersystem automatisch aus diesem (konzeptionellen) Modell ableitbar. Beispielsweise umfasst INTERLIS einen XML-basierten Transfer-Dienst, dessen Definitionen regelbasiert aus den jeweiligen Modellen erzeugt werden. Die Nutzung der Datenmodel-

lierung in enger Verbindung mit systemneutralen Schnittstellendiensten nennt man *den modell-basierten* oder den *modell-getriebenen Ansatz* (siehe "model-driven architecture" von OMG, www.omg.org/mda/).

Modelle können auf gemeinsamen Basiskonzepten aufbauen. Da INTERLIS erlaubt, dies durch die Benutzung von Vererbung und Polymorphismus explizit zu beschreiben, ist jederzeit klar, welche Konzepte gemeinsam und welche jeweils spezifisch sind. Dies ist im Hinblick auf eine angestrebte semantische Interoperabilität von grosser Bedeutung. Beispielsweise kann die Transferdatei einer Gemeinde problemlos von einer übergeordneten Verwaltungseinheit (Kanton, Bund) interpretiert werden, ohne dass sich die Beteiligten dafür auf ein einziges Modell einigen müssten. Es genügt, wenn jede Stufe ihr Modell auf jenem der übergeordneten Einheit aufbaut.

Es ist denkbar und durchaus erwünscht, dass neue Dienste auf der Basis von INTERLIS entwickelt werden. Dies wird erheblich durch einen INTERLIS 2-Compiler erleichtert. Dieses Programm liest und schreibt INTERLIS-Modelle, erlaubt deren Änderung und überprüft, ob Modelle den syntaktischen und semantischen Bedingungen von INTERLIS entsprechen. Dieser Compiler kann - gemäss dem aktuellen INTERLIS-Transferdienst mit XML – unter anderem aus INTERLIS-Modellen automatisch XML-Schema-Dokumente (www.w3.org/XML/Schema) generieren. Damit können die konkreten INTERLIS/XML-Daten durch entsprechende allgemeine XML-Werkzeuge noch breiter genutzt werden. Solange die Nutzungsbedingungen nicht verletzt werden, steht dieser INTERLIS-Compiler zum Erstellen neuer Werkzeuge zur Verfügung.



Figur 3: Vererbungs-Hierarchie von Adresse, Person und Gebäude.

1.3 Gliederung in Modelle und Themen

In einem Modell (bzw. Schema) wird eine Vorstellung der Welt beschrieben, wie sie für eine bestimmte Anwendung von Bedeutung ist. Ein Modell ist in sich abgeschlossen, darf aber Teile von anderen Modellen verwenden oder erweitern. Ein INTERLIS-Modell ist dadurch in gewisser Weise mit den Modulen oder "Packages" mancher Programmiersprachen vergleichbar.

Primär können Modelle unterschieden werden, die nur typenmässige Definitionen (Einheiten, Wertebereiche, Strukturen) enthalten und solchen, zu denen Daten existieren können. Die Beschreibungen, zu denen Daten existieren können, werden in Themen gegliedert. Diese Gliederung erfolgt gemäss den Vorstellungen, in welchen Organisationseinheiten und durch wen die Daten verwaltet und gebraucht werden. Daten, die typischerweise durch unterschiedliche Stellen verwaltet oder gebraucht werden, sollen auch in

verschiedenen Themen definiert werden. Themen dürfen voneinander abhängig sein. Solche Abhängigkeiten sollen sich jedoch auf das Minimum beschränken. Beziehungen zwischen Themen, deren Daten durch verschiedene Stellen verwaltet werden, sollen möglichst vermieden werden, da für die Erhaltung der Konsistenz mit speziellem Aufwand zu rechnen ist. Zyklische Abhängigkeiten sind in jedem Fall ausgeschlossen. Themen können nebst den eigentlichen Datendefinitionen auch Definitionen für Sichten und Grafiken umfassen.

Ein Thema kann ein anderes erweitern. Damit werden alle Konzepte, welche das Basisthema definiert, übernommen und können ergänzt bzw. spezialisiert werden.

Beispielsweise könnte ein Modell der amtlichen Vermessung eines Landes unter anderem die Themen "Adressen" und "Gebäude" umfassen (Figur 3). Diese Konzepte sind voneinander unabhängig; der Bezug wird algorithmisch über Koordinaten hergestellt. Personen weisen Adressen auf, so dass das Personen-Modell dieses Landes auf dem Landesmodell der amtlichen Vermessung aufbaut, wobei das Thema "Personen" vom Thema "Adressen" des Vermessungsmodells abhängt.

Nun möchte man die Personen in einer Gegend A präziser beschreiben, als es das Landes-Personen-Modell vorsieht. Diese Gegend A verfasst daher ihr eigenes Modell, in dem das Thema "Personen" aus dem landesweiten Personen-Modell übernommen und erweitert wird.

In einer Gegend B will man einerseits eine explizite Beziehung zwischen Gebäuden und Adressen herstellen, andererseits wird auch hier das Landes-Personen-Modell als zu grob erachtet. Wiederum werden die jeweiligen Themen übernommen und spezialisiert. Beide Erweiterungen werden in einem einzigen Modell - der "Weltsicht" der Gegend B - zusammengefasst.

1.4 Objekt-Konzept

1.4.1 Objekte und Klassen

Ein Objekt (auch Objektinstanz oder einfach Instanz genannt) besteht aus den Daten eines Gegenstandes der realen Welt und ist eindeutig identifizierbar. Typischerweise besitzen zahlreiche Objekte gleichartige Eigenschaften und können daher zusammengefasst werden. Eine Menge von Objekten (Objektmenge) mit gleichartigen Eigenschaften wird als Klasse bezeichnet. Jeder Eigenschaft entspricht (mindestens) ein Attribut. INTERLIS 1 verwendete anstelle des Begriffs Klasse den Begriff Tabelle. Andere Begriffe für Klasse sind Entitätsmenge, Entitätstyp, Feature(typ).

Mit der Beschreibung einer Klasse wird unter anderem auch festgehalten, welche Eigenschaften oder Merkmale die einzelnen Objekte tragen. Diese werden Attribute genannt. Die Attributwerte der einzelnen Objekte sind dabei nicht beliebig, sondern müssen bestimmten Bedingungen genügen, die zur Beschreibung eines Attributes gehören.

INTERLIS bietet dafür eine Reihe von grundlegenden Datentypen an (Basis-Datentypen: Zeichenketten, numerische Datentypen, Aufzählungen, kartesische und elliptische 2D- bzw. 3D-Koordinaten, Linien, Flächen), auf deren Basis neue, komplexere Datenstrukturen definiert werden können. Um die Aussage noch zu präzisieren, können numerische Attribute mit einer Masseinheit versehen und auf ein Referenzsystem bezogen werden; Koordinaten können auf ein Koordinatensystem bezogen werden.

Neben diesen Basis-Datentypen kann ein Attribut auch Unterstrukturen enthalten. Die einzelnen Elemente der Unterstruktur sind als Strukturelemente aufzufassen, die nur im Zusammenhang mit ihrem Hauptobjekt existieren und auch nur über dieses auffindbar sind. Der Aufbau der Strukturelemente wird ähnlich wie jener von Klassen beschrieben, und es kann dazu auch die Definition einer Klasse herangezogen werden.

Abgesehen davon, dass alle Attributwerte ihrem jeweiligen Typ entsprechen müssen, können weitere Bedingungen formuliert werden. INTERLIS unterscheidet zwischen folgenden Arten von Konsistenzbedingungen:

- Solchen, die sich auf ein einzelnes Objekt beziehen. Diese werden weiter untergliedert in Bedingungen, die jedes Objekt einer Klasse erfüllen *muss* ("harte" Konsistenzbedingungen), und Bedingungen, deren Verletzung zwar an sich möglich, aber selten ist ("weiche" Konsistenzbedingungen).
- Solchen, welche die Eindeutigkeit von Attributkombinationen aller Objekte einer Klasse fordern.
- Solchen, welche die Existenz eines Attributwertes in einer Instanz einer anderen Klasse fordern.
- Kompliziertere Bedingungen, die sich auf Objektmengen beziehen und mittels Sichten formuliert werden.

1.4.2 Klassenerweiterung und Polymorphismus

Klassen sind entweder eigenständig oder erweitern (spezialisieren, erben) eine Basisklasse. D.h. die Beschreibung einer Klasse ist entweder unabhängig oder enthält Erweiterungen von einer anderen, ererbten Beschreibung. Eine Klassenerweiterung (auch Unterklasse oder Subklasse genannt) kann einerseits zusätzliche Attribute und Konsistenzbedingungen haben, andererseits übernommene Bedingungen (Datentypen, Konsistenzbedingungen) verschärfen.

Jedes einzelne Objekt gehört zu genau einer Klasse (man sagt auch: ist Objektinstanz oder Instanz einer Klasse). Es erfüllt aber immer auch sämtliche Bedingungen, welche die Basisklassen (d.h. die Oberklassen) stellen. Zu jeder Klasse gibt es eine Menge von Objekten, die Instanzen der Klasse oder einer ihrer Erweiterungen sind. In Falle konkreter Klassen besteht eine normalerweise kleinere Untermenge von Instanzen, die exakt zu dieser Klasse gehören.

Eine erweiterte Klasse ist *polymorph* zu ihren Basisklassen: Überall dort, wo Instanzen einer Basisklasse erwartet werden, können auch Instanzen einer Erweiterung stehen (so genannter Teilmengen-Polymorphismus oder Substitutionsprinzip). INTERLIS wurde so gestaltet, dass das polymorphe Lesen immer möglich ist. Wird beispielsweise eine Beziehung zu einer Klasse definiert (vgl. Kapitel 1.4.4 Beziehungen zwischen Objekten), haben auch Objekte von Erweiterungen diese Beziehung. Das vollständige polymorphe Schreiben ist nicht Ziel der vorliegenden INTERLIS-Version.

Die Elemente von Unterstrukturen sind keine eigentlichen selbständigen Objekte, sondern Strukturelemente und gehören damit auch nicht zur Menge der Instanzen irgendeiner Klasse.

1.4.3 Meta-Modelle und Meta-Objekte

Koordinaten- und Referenzsysteme sowie Grafiksignaturen stellen sich aus der Sicht der Anwendung als Modellelement (bzw. Schemaelement) dar, die in den Anwendungsdefinitionen verwendet werden können. Da aber verschiedene Koordinaten- und Referenzsysteme und vor allem verschiedene Grafiksignaturen auf die gleiche Art beschrieben werden können, macht es Sinn, ihre Eigenschaften ebenfalls innerhalb von Modellen mittels Klassen zu beschreiben. Jedem einzelnen System, bzw. jeder Grafiksignatur (z.B. ein Punktsymbol, eine Linienart) entspricht dann ein Objekt.

Meta-Objekte müssen in einem Meta-Modell definiert sein. Die verwendbaren Objekte müssen explizit als Meta-Objekte gekennzeichnet sein (Erweiterungen der vordefinierten Klasse METAOBJECT) und können dann von der Anwendung über den Namen referenziert werden. Dafür ist es nötig, dass die Meta-Objekte dem Werkzeug, das die Anwendungsdefinition verarbeitet, mittels Behältern (vgl. Kapitel 1.4.5 Behälter, Replikation und Datentransfer) zur Verfügung stehen.

1.4.4 Beziehungen zwischen Objekten

INTERLIS 2 unterscheidet (im Gegensatz zu INTERLIS 1) zwei Arten von Beziehungen zwischen Objekten: eigentliche Beziehung und Referenzattribut.

Als *eigentliche Beziehung* wird eine Menge von Objektpaaren (bzw. im allgemeinen Fall von Objekt-n-Tupeln) bezeichnet. Das erste Objekt jedes Paares gehört zu einer ersten Klasse A, das zweite zu einer zweiten Klasse B. Dabei soll die Zuordnung von Objekten zu den Paaren vorgegeben sein, sie muss also nur beschrieben, d.h. modelliert werden. Wie weiter unten im Kapitel 1.5 Sichten-Konzept gezeigt wird, ist es im Gegensatz dazu auch möglich, Zuordnungen algorithmisch z.B. aufgrund von Attributwerten zu berechnen.

Eigentliche Beziehungen werden als eigenständige Konstrukte, so genannte Beziehungsklassen (bzw. Assoziationsklassen), beschrieben, die auch selbst wieder erweiterbar sind. INTERLIS 2 unterstützt nicht nur Zweierbeziehungen, sondern erlaubt auch mehrfache Beziehungen und solche mit eigenen Attributen. Eine Beziehungsklasse ist damit auch selbst wieder eine Objektklasse.

Wichtige Eigenschaften solcher Beziehungen sind:

- *Kardinalität* – Wie viele Objekte der Klasse B (bzw. A) können einem Objekt der Klasse A (bzw. B) durch die Beziehung zugeordnet werden? D.h. wie viele Paare von Objekten kann es mit einem bestimmten Objekt an erster bzw. zweiter Stelle geben. Im allgemeinen Fall: Wie viele n-Tupel kann es mit bestimmten Objekten an n-1 Stellen geben?
- *Stärke* – INTERLIS 2 unterscheidet zwischen Assoziation, Aggregation und Komposition. In allen Fällen sind die beteiligten Objekte individuell ansprechbar. Bei Assoziation und Aggregation existieren die beteiligten Objekte unabhängig voneinander. Bei Aggregation und Komposition gibt es eine Asymmetrie zwischen den beteiligten Klassen: Die Objekte der einen Klasse (der Oberklasse) werden als Ganze (bzw. als Oberobjekte)) bezeichnet, die Objekte der anderen Klasse (der Unterklasse) als Teile (bzw. als Unterobjekte). Bei Assoziation sind die Objekte gleichberechtigt und lose miteinander verbunden. Aggregation und Komposition sind konzeptionell gerichtete Beziehungen: Einem Ganzen (Oberobjekt der Oberklasse) sind mehrere Teile (Unterobjekte der Unterklasse) zugeordnet. Anders als bei der Assoziation werden im Fall der Aggregation beim Kopieren eines Ganzen auch alle zugeordneten Teile mitkopiert, während bei der Löschung des Ganzen die Teile erhalten bleiben. Für eine Komposition gilt gegenüber der Aggregation zusätzlich, dass im Falle der Löschung des Ganzen auch die Teile gelöscht werden. Wie sich Beziehungen zu anderen Objekten beim Kopieren verhalten, ist im Kapitel 2.7.2 Stärke der Beziehung beschrieben. Hinweis: Die Unterobjekte (Teile) von Kompositionen sind im Gegensatz zu Strukturelementen von Unterstrukturen identifizierbare Objekte.
- *Rolle* – Welche Bedeutung besitzen die beteiligten Klassen aus der Sicht der Beziehung? Das wird für jede der beteiligten Klassen mit Hilfe der Rolle festgelegt.

Mit einem *Referenzattribut* wird der Bezug von einem Objekt bzw. einem Strukturelement zu einem anderen Objekt geschaffen. Eine solche Beziehung ist nur dem referenzierenden, nicht aber dem referenzierten Objekt bekannt. Sie ist also einseitig.

Ohne die Unabhängigkeit der Themen zu verletzen, können mittels spezieller Kennzeichnung (EXTERNAL) auch Beziehungen (d.h. eigentliche Beziehungen und Referenzattribute) definiert werden, die einen Bezug zu Objekten *eines anderen Behälters* des gleichen oder eines anderen Themas schaffen. Voraussetzung dazu ist allerdings, dass die Struktur, in der die Beziehung definiert wird, zu einem Thema gehört, das von demjenigen, zu dessen Klasse es verweist, abhängig ist.

Im Interesse einer klaren Ordnung können sich Beziehungen nur auf Klassen beziehen, die an der Stelle wo die Beziehungsklasse (bzw. das Referenzattribut) definiert wird, bereits bekannt sind.

1.4.5 Behälter, Replikation und Datentransfer

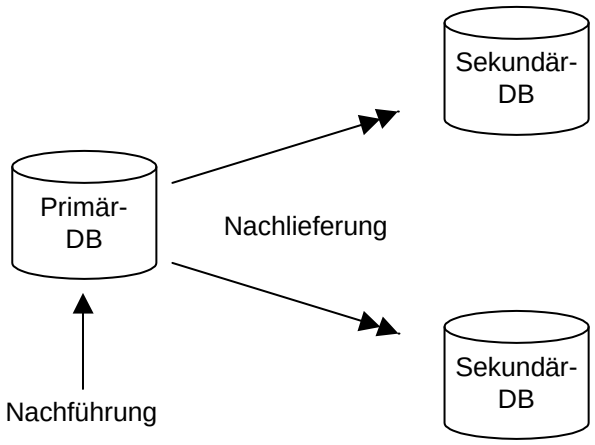
Als Behälter wird eine abgeschlossene Sammlung von Objekten bezeichnet, die zu einem Thema oder zu dessen Erweiterungen gehören. Abgeschlossen heisst, dass ein Behälter alle Objekte enthalten soll, die innerhalb des Themas zueinander in Beziehung stehen. Typischerweise wird ein Behälter alle Objekte einer bestimmten Gegend (z.B. einer Gemeinde, eines Kantons oder gar des ganzen Landes) enthalten. Ein Behälter kann insbesondere auch Daten verschiedener Erweiterungen (z.B. verschiedener Kantone mit eigenen Themenerweiterungen) enthalten. Dies bedingt allerdings, dass im Rahmen einer solchen Transfergemeinschaft, die Bezeichnungen der Modelle, Themen und Themenerweiterungen eindeutig sind.

Im Rahmen von Konsistenzbedingungen wird häufig von "allen" Objekten einer Klasse gesprochen. Konzeptionell meint man damit auch grundsätzlich alle Objekte, die die gewünschte Eigenschaft haben und die es überhaupt, d.h. behälterübergreifend gibt. Es ist aber aus verschiedenen Gründen (Effizienz, Verfügbarkeit, Zugriffsberechtigung, etc.) offensichtlich, dass eine Überprüfung nur innerhalb der lokal zugänglichen Behälter möglich ist. Im Falle von Behältern mit Meta-Objekten gelten die Konsistenzbedingungen ausdrücklich nur innerhalb eines Behälters, da davon ausgegangen wird, dass die Meta-Daten beschreibenden Charakter haben und deshalb jeweils vollständig (quasi als Bibliothek) in einem Behälter zur Verfügung stehen müssen.

- Es werden verschiedene Arten von Behältern unterschieden: *Daten-Behälter* – Umfasst die Instanzen von Klassen des Themas.
- *Sicht-Behälter* – Umfasst die Instanzen von Sichten des Themas.
- *Behälter mit den Basis-Daten für die Grafik* – Umfasst die Instanzen aller Daten oder Sichten, die für die Grafiken des Themas benötigt werden. Damit kann eine Grafik-Umsetzer-Software bedient werden.
- *Behälter mit den Grafik-Elementen* – Umfasst die Instanzen aller Grafikobjekte (= Signaturen), die gemäss den Grafiken des Themas benötigt werden. Damit kann eine Grafik-Darstellungs-Software (Renderer) bedient werden (vgl. Figur 5).

INTERLIS regelt nicht wie die Objekte in den Systemen gehalten werden müssen. Die Regelungen betreffen nur die Schnittstelle zwischen den Systemen. Aktuell ist eine Schnittstelle zum Transfer von Behältern als XML-Dateien definiert. Dabei wird nicht nur der vollständige Transfer des ganzen Behälters, sondern auch die inkrementelle Nachlieferung unterstützt.

Beim vollständigen Transfer wird davon ausgegangen, dass der Empfänger neue, eigenständige Objektkopien aufbaut, die keinen unmittelbaren Zusammenhang zum Ursprungsobjekt aufweisen. Im Rahmen des vollständigen Transfers müssen die Objekte darum nur mit einer temporären Transfer-Identifikation (Transferidentifikator, abgekürzt TID) versehen werden. Diese wird zum Transferieren von Beziehungen genutzt.



Figur 4: Nachführung in der Primär-Datenbank und anschliessende Nachlieferung an Sekundär-Datenbanken (ein doppelter Pfeil bedeutet inkrementelle Nachlieferung).

Beim inkrementellen Transfer wird davon ausgegangen, dass der Sender einmal einen Initialzustand eines Datenbehälters (andere Behälterarten sind ausgeschlossen) liefert und den Empfänger anschliessend mit Nachlieferungen versorgt, die diesem erlauben, seine Daten zu aktualisieren (Figur 4). Die Objekte werden damit repliziert und behalten den Zusammenhang zum Originalobjekt, d.h. sie können nicht unabhängig vom Original verändert werden und erhalten keine neue Identifikation.

Dabei geht man davon aus, dass zwischen den Betreibern der Primär- und der Sekundär-Datenbanken vertragliche Abmachungen getroffen werden (unter anderem Umfang, Häufigkeit der Nachlieferung), die aktuell nicht mit Instrumenten von INTERLIS beschrieben werden können. Für die eigentliche Nachlieferung stellt INTERLIS aber die nötigen Mittel zur Verfügung. Neue Objekte werden wie beim ersten Transfer mitgeteilt. Ihnen ist eine eindeutige Objekt-Identifikation (Objektidentifikator, abgekürzt OID) zugeordnet, die über die Zeit erhalten bleiben muss. Bei Änderungen wird auf diesen eindeutigen OID Bezug genommen und alle Attribute des Objektes (inkl. aller Strukturelemente von Strukturattributen) werden neu geliefert. Auf die gleiche Art, werden auch Löschungen mitgeteilt. Dabei trägt primär der Sender die Verantwortung für die Konsistenz der Objekte (z.B. Einhaltung von Konsistenzbedingungen, Korrektheit und Kardinalität von Beziehungen). Er meldet dazu den Empfängern, welche Objekte geändert oder gelöscht und welche neu erzeugt wurden. Bei themenübergreifenden Referenzattributen wird – im Interesse der Unabhängigkeit der Themen – nicht vorausgesetzt, dass die Integrität jederzeit gewährleistet ist. Es ist Sache des Empfängers, damit zurechtzukommen, dass vorübergehend Inkonsistenzen zwischen Basisthemen und abhängigen Themen bestehen, d.h. dass ein referenziertes Objekt nicht existiert.

Der Rahmen, in dem die Eindeutigkeit des OID gewährleistet ist, ist durch INTERLIS selbst nicht bestimmt. Eine Möglichkeit, wie ein solcher OID aufgebaut sein kann, ist in Anhang E beschrieben. Andere Möglichkeiten sind aber ohne weiteres denkbar. Wichtig ist, dass der OID durch die verschiedenen Datenerfasser einer Transfergemeinschaft bei korrekter Anwendung der Regel zum Aufbau des OID immer im Rahmen der Transfergemeinschaft eindeutig ist. Je nach Aufbau des OID ist der inkrementelle Datenaustausch in einem grösseren (z.B. weltweit) oder in einem kleineren (z.B. organisationsintern) Rahmen möglich. Das gewählte Verfahren zur Bestimmung des OID definiert somit die potenzielle Transfergemeinschaft.

Ein Objekt darf nur in seinem Originalbehälter bzw. ausserhalb nur mit Erlaubnis dessen Verwalters verändert werden. Alle anderen, sekundären Behälter dürfen ein Objekt nur als Folge einer Nachlieferung verändern. INTERLIS 2 verlangt darum, dass – im Rahmen der inkrementellen Nachlieferung – nebst den Objekten auch die Behälter eindeutig und dauerhaft identifizierbar sein müssen. Die Behälter erhalten dann ebenfalls einen OID. Beim vollständigen Transfer braucht auch der Behälter nur eine Transferidenti-

fikation (TID). Wo die Unterscheidung zum normalen OID (bzw. TID) wesentlich ist, wird vom Behälteridentifikator BOID (bzw. vom Behältertransferidentifikator BID) gesprochen.

Es muss davon ausgegangen werden, dass verschiedene Objekte zunächst in Behältern z.B. einer Gemeinde geführt werden, diese Behälter dann als ganzes an den Kanton übermittelt und dort in Behälter, die pro Thema den ganzen Kanton enthalten, integriert werden. Allenfalls werden diese Behälter dann wieder z.B. an den Bund weitergegeben. Damit jederzeit klar ist, welches der Originalbehälter ist, wird dessen BOID bei jedem replizierten Objekt mitgegeben. Ein Empfänger kann damit eine Behälter-Verwaltung aufbauen, in dem er festhält, in welchen eigenen Behältern er replizierte Objekte von welchen Originalbehältern aufbewahrt (INTERLIS 2 bietet auch die nötigen Mittel an, dass solche Behälter mit INTERLIS selbst beschrieben und wie normale Objekte ausgetauscht werden können). Nutzt der Empfänger die Eigenschaft von INTERLIS 2, dass bei themenübergreifenden Beziehungen nicht nur die OID des Bezugsobjektes sondern auch die BOID dessen Originalbehälters transferiert wird, kann er auf effiziente Art bestimmen, in welchem seiner Behälter das Bezugsobjekt liegt.

1.5 Sichten-Konzept

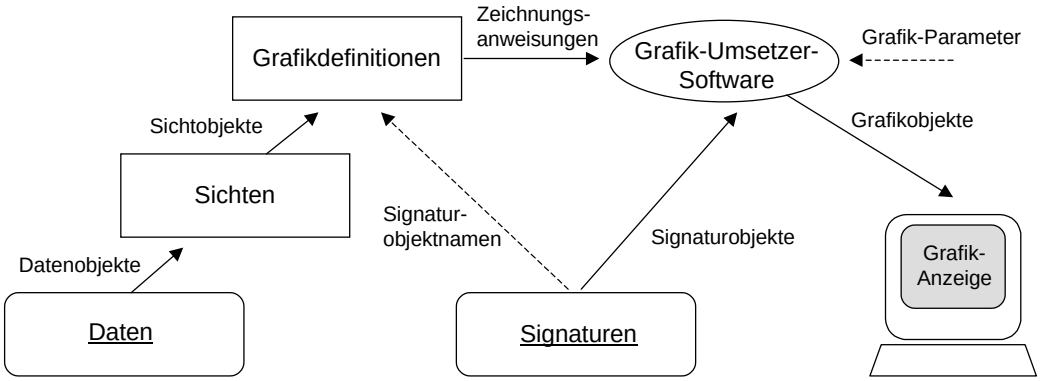
Mit INTERLIS 2 können neben den eigentlichen Sachobjekten auch Sichten modelliert werden. Sichten sind im Prinzip virtuelle Klassen, deren Instanzen nicht Daten eines eigentlichen Gegenstandes der realen Welt sind, sondern rechnerisch aus anderen Objekten abgeleitet werden.

Eine Sichtdefinition besteht aus folgenden Teilen:

- *Basismengen* – Von welchen Klassen bzw. Sichten fliessen die Objekte in die Berechnung der Sicht-Objekte ein? Bei Klassen werden jeweils nicht nur die eigentlichen Instanzen herangezogen, sondern auch im Sinne des Polymorphismus alle Instanzen von Erweiterungen. INTERLIS definiert nicht, welche Behälter ein System zum Errechnen einer Sicht berücksichtigen muss.
- *Verknüpfungsvorschrift* – Wie werden die Basismengen verknüpft? INTERLIS 2 kennt Verbindungen *(Join)*, Vereinigungen *(Union),* Zusammenfassungen *(Aggregation)* und Inspektionen von Unterstrukturen. Mengentheoretisch gesehen, bilden Verbindungen das Kreuzprodukt und Vereinigungen die Vereinigungsmenge der Basismengen. Zusammenfassungen erlauben es, Objekte der Basismenge zu einem neuen Sichtobjekt zusammenzufassen, wenn sie definierbare Kriterien erfüllen. Inspektionen von Unterstrukturen ermöglichen es, die Strukturelemente einer Unterstruktur als Menge von Strukturelementen zu sehen. Verbindungen und Zusammenfassungen können auch als virtuelle Assoziationen aufgefasst werden.
- *Selektion* – Welche errechneten Objekte sollen tatsächlich zur Sicht gehören? INTERLIS erlaubt es, hierfür komplexe Bedingungen anzugeben.

1.6 Grafik-Konzept

Darstellungsbeschreibungen bauen auf Klassen oder Sichten auf und deklarieren in so genannten *Grafikdefinitionen* (vgl. Figur 5 und Anhang L Glossar) welche Grafiksignaturen (z.B. Punktsymbol, Linie, Flächenfüllung oder Textanschrift) den Objekten der Sicht zugeordnet werden sollen, damit ein Grafikumsetzer darstellbare Grafikobjekte erzeugen kann. Die Grafiksignaturen sind in einem eigenen Signaturenmodell definiert, wo auch die Signatur-Eigenschaften beschrieben sind.



Figur 5: Grafikdefinitionen, die einerseits auf Daten und Sichten und andererseits auf Signaturen aufbauen, um daraus eine Grafik zu erzeugen (abstrahierte Darstellung).

Der Verweis auf die gefragten Grafiksignaturen erfolgt mittels Namen (vgl. Signaturobjektnamen in Figur 5). Die Grafiksignaturen selbst (auch Signaturobjekte genannt) sind als Meta-Objekte (Daten) in entsprechenden Behältern enthalten. Einen Behälter mit solchen Signaturobjekten nennt man oft auch Signaturenbibliothek.

Im Signaturenmodell wird für jede Signaturart in der entsprechenden Signaturklassen-Definition festgelegt, welche Parameter (z.B. Lage und Orientierung einer Signatur) für die Darstellung nötig sind. Damit wird die Schnittstelle zum Grafiksubsystem (der Grafik-Umsetzer-Software) der jeweiligen Systeme nicht durch INTERLIS selbst, sondern durch die Signaturenmodelle festgelegt. In diesem Rahmen können zudem globale Laufzeit-Parameter deklariert werden (z.B. der Darstellungsmassstab), die zur Laufzeit dem Grafiksubsystem bekannt sind und den Entscheid, mit welchen Signaturen die Darstellung erfolgen soll, beeinflussen können. Da solche Festlegungen eng mit den effektiven Möglichkeiten der Systeme zusammenhängen, müssen diese Parameter und die Eigenschaften von Signaturen im Rahmen von Verträgen mit den Systemherstellern festgelegt werden (vgl. folgendes Kapitel 1.7 Kontrakte).

Eine Darstellungsbeschreibung muss die Umsetzung in Signaturen nicht abschliessend regeln. Sie kann vielmehr durch eine andere Darstellungsbeschreibung geerbt werden. Darin können Parameter, die in Basisdefinitionen offen gelassen worden sind, ergänzt werden oder es können bereits erfolgte Darstellungsanweisungen ersetzt werden.

1.7 Kontrakte

Damit INTERLIS 2 verschiedensten Ansprüchen genügen kann, enthält es auch Konstrukte, deren Implementation mit der Definition nicht geregelt sind, z.B. Funktionen, Linienformen, Signaturen (Einzelheiten sind im Kapitel 2 Beschreibungssprache aufgeführt). Würde nichts Weiteres unternommen, wäre die Zielsetzung, wonach eine INTERLIS-Beschreibung automatisch in einen Dienst umgesetzt werden kann, nicht mehr erfüllt. Im Sinne der Einfachheit verzichtet INTERLIS aber dennoch darauf, für derartige Fälle weitergehende Definitionsmöglichkeiten anzubieten (wie z.B. eine eigentliche Programmiersprache zur Definition von Funktionen).

Damit die automatische Umsetzung mindestens in einem reduzierten Rahmen (z.B. in einem Land oder für einen bestimmten Anwendungsbereich) möglich ist, sollen solche Konstrukte aber nur innerhalb von Modellen zulässig sein, die durch Kontrakte abgedeckt sind.

Kontrakte sind Vereinbarungen oder Verträge zwischen denjenigen, die Modelle definieren und denjenigen, die Werkzeuge anbieten, die auf INTERLIS-Beschreibungen aufbauen. Typischerweise werden nur die Basis-Modelle einer Branche oder eines Landes an Kontrakte gebunden. Für die Definition solcher Basis-Modelle arbeiten Modellierer und Werkzeug-Anbieter zusammen und vereinbaren so einen Kon-

trakt. Die Werkzeug-Anbieter sind nun in der Lage, die in diesen Modellen geforderten Elemente (z.B. Funktionen) noch unabhängig von der konkreten Anwendung zu realisieren. Die konkreten Modelle sind dann wieder automatisch (also ohne Beihilfe des Werkzeug-Anbieters) umsetzbar.

Wichtig ist, dass solche Zusatzkonstrukte nach einer möglichst breit geführten Diskussion systemneutral formuliert werden und ebenso öffentlich wie das Modell selbst zur Verfügung stehen. Sonst kann eines der wichtigsten Ziele von INTERLIS, nämlich die Offenheit und Interoperabilität, nicht mehr gewährleistet werden.

1.8 Dienste, Werkzeugfähigkeiten und Konformität

INTERLIS 2 ermöglicht die konzeptionelle Beschreibung von Daten und definiert einen systemneutralen Daten-Transfer. INTERLIS 2 verzichtet bewusst darauf, die Implementation vorzuschreiben und bleibt damit systemunabhängig. In der Praxis wird sich damit häufig die Frage stellen, ob ein bestimmtes Werkzeug oder ein bestimmter Dienst INTERLIS 2-konform ist oder nicht.

INTERLIS 2 geht nicht davon aus, dass nur ein Ja (d.h. vollständige Erfüllung) oder ein Nein (d.h. Nichterfüllung) möglich ist. Vielmehr kann ein Dienst für bestimmte Aspekte INTERLIS 2-konform sein, während er andere Anforderungen nicht erfüllt.

Im einfachsten Fall erfüllt ein bestimmtes System die INTERLIS-Spezifikationen für einen bestimmten Fall (für eine bestimmte Menge von Modellen, nur für das Lesen oder nur für das Schreiben oder beides, etc.).

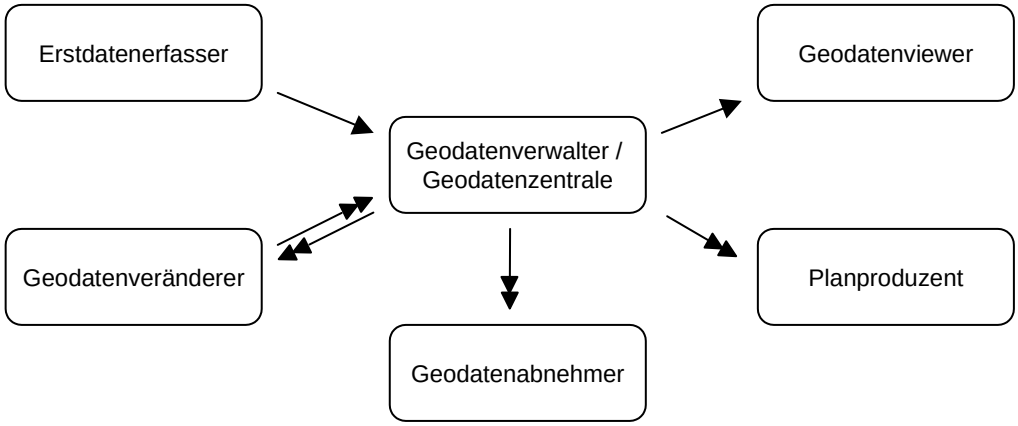
Im Idealfall kann einem INTERLIS-Werkzeug eine Menge von INTERLIS-Modellen übergeben und damit erreicht werden, dass das Werkzeug damit seine Dienste und Fähigkeiten automatisch der durch die Modelle definierten Situation anpasst. In vielen Fällen wird diese Anpassung jedoch mit weiteren manuellen Tätigkeiten (System-Konfigurierung oder gar Programmierung) verbunden sein. Zur Basisfunktionalität (Basisdienst) solcher Werkzeuge gehört sicher, dass sie INTERLIS-Modell-Beschreibungen korrekt einlesen können. Dazu gehört vor allem, dass die Systemfähigkeiten korrekt gemäss den INTERLIS-Konstrukten, insbesondere den Vererbungskonstrukten angeboten werden.

Aus der Sicht von INTERLIS können die Fähigkeiten solcher Werkzeuge wie folgt gegliedert werden (so genannte Werkzeugfähigkeiten, Funktionalitäten oder Dienste):

- Daten einlesen (inkl. gespeicherte Sichten, ohne Sicht-Objekte erzeugen)
- Daten einlesen (inkl. gespeicherte Sichten, mit Sicht-Objekte erzeugen)
- Konsistenzen prüfen
- Sichten (Views) schreiben
- Grafik erzeugen (inkl. Sichten (Views) lesen)
- Daten bearbeiten und schreiben
- Objektidentifikatoren (OID) erzeugen
- Nachlieferung lesen (inkrementelle Nachlieferung)
- Nachlieferung schreiben (inkrementelle Nachlieferung)

Es ist auch durchaus möglich, dass ein bestimmtes Werkzeug oder ein bestimmter Dienst für einzelne Modelle und Themen (Daten oder Sichten) bestimmte Fähigkeiten (z.B. inkrementelle Nachlieferung) aufweist, sie aber bei anderen Modellen oder Themen nicht unterstützt.

Zudem stellt sich die Frage, welche Kontrakte durch ein Werkzeug unterstützt werden.



Figur 6: Die verschiedenen Einsatzgebiete von INTERLIS (ein doppelter Pfeil bedeutet inkrementelle Nachlieferung).

Betrachtet man ein Beispiel des Zusammenwirkens verschiedener Beteiligter, ist es offensichtlich, dass je nach Einsatzgebiet und Rolle eines Beteiligten unterschiedliche Werkzeugfähigkeiten oder Dienste von INTERLIS 2 gefragt sind (Figur 6). Ein einzelner Beteiligter kann in einer Anwendung (d.h. INTERLIS-Modell) mit verschiedenen Themen (TOPICS) mehrere Rollen einnehmen:

- *Erstdatenerfasser* – Daten bearbeiten und schreiben; gegebenenfalls OID erzeugen.
- *Geodatenveränderer* (Nachführen, Ergänzen): Daten einlesen; Daten bearbeiten und schreiben; OID erzeugen; Inkremente produzieren; Inkremente lesen; Konsistenzen (lokal) prüfen.
- *Geodatenverwalter/Geodatenzentrale* – Daten einlesen; Daten bearbeiten und schreiben; OID erzeugen; Inkremente lesen; Konsistenzen prüfen (global).
- *Geodatenabnehmer* – Daten einlesen; Inkremente lesen.
- *Geodatenviewer* – Daten einlesen; Sicht-Objekte und grafische Darstellungen erzeugen.
- *Planproduzent* — Daten einlesen; Inkremente lesen; Sichten lesen und grafische Darstellungen erzeugen.

1.9 Das kleine Beispiel Roads als Einstieg

Im Anhang C ist ein kleines Beispiel beigefügt, das die wichtigsten INTERLIS-Sprachelemente im Rahmen einer einfachen Anwendung vorstellt.

Figur 7: Das kleine Beispiel Roads.

1.10 Weitere Gliederung des Referenzhandbuchs

Im nächsten Kapitel 2 wird die Beschreibungssprache formal definiert. Im Kapitel 3 wird der sequentielle Transfer von Geodaten spezifiziert. Dieses Kapitel enthält einen allgemeinen Teil, der für alle aktuellen und künftigen sequentiellen INTERLIS 2-Transferdienste gilt und einen speziellen Teil, der für den INTERLIS 2-Transferdienst mit XML gilt.

Die Anhänge gliedern sich in zwei *normative* Anhänge A und B, ergänzt durch einen *informativen* Anhang C, einen Index (Anhang D) und in die *Standard-Erweiterungsvorschläge* E bis K sowie ein Glossar (Anhang L).

Die normativen Anhänge sind integrierender Bestandteil dieser Spezifikation. Die Standard-Erweiterungsvorschläge (Anhänge E bis K) sind sehr zur Implementation empfohlen. Es sind dies informative (nicht-normative) Anhänge mit eigenständigem Charakter. Die entsprechenden Fragestellungen können von Implementationen auch anders gelöst werden, sie sind dadurch immer noch INTERLIS 2-konform. In diesen Fällen ist es dann aber Sache der am Transfer Beteiligten, sich über die Spezifikation zu einigen. Für viele dieser Fragestellungen muss ein Kontrakt abgeschlossen werden.

2 Beschreibungssprache

2.1 Verwendete Syntax

Zur Festlegung des konzeptionellen Datenmodells (Datenschemas) und der Transferparameter eines Datentransfers wird in den folgenden Kapiteln eine formale Sprache definiert. Diese Sprache ist selbst formal definiert. Syntaxregeln beschreiben dabei die zulässige Reihenfolge von Symbolen.

Die Beschreibung folgt damit der Art und Weise, die bei der Beschreibung moderner Programmiersprachen üblich ist. Hier wird deshalb nur eine knappe, für das Verständnis nötige Darstellung angegeben. Für Einzelheiten wird auf die Literatur verwiesen. Eine kurze Einführung befindet sich z.B. in "Programmieren in Modula-2" von Niklaus Wirth.

Eine Formel wird im Sinne der erweiterten Backus-Naur-Notation (EBNF) wie folgt aufgebaut:

```
Formel-Name = Formel-Ausdruck.
```

Der Formel-Ausdruck ist eine Kombination von:

- fixen Wörtern (inkl. Spezialzeichen) der Sprache. Sie werden in Apostrophe eingeschlossen, z.B. 'BEGIN'.
- Referenzen von anderen Formeln durch Angabe des Formelnamens.

Als Kombination kommen in Frage:

Aneinanderreihung

```
a b c          zuerst a, dann b, dann c.
```

Gruppierung

```
( a )          runde Klammern gruppieren Formel-Ausdrücke.
```

Auswahl

```
a | b | c      a, b, oder c.
```

Option

```
[ a ]          a oder nichts (leer).
```

Fakultative Wiederholung

```
{ a }          Beliebige Folgen von a oder nichts (leer).
```

Obligatorische Wiederholung (als Zusatz zur EBNF)

```
(* a *)        Beliebige Folge von a aber mindestens Eins.
```

Beispiele von Formel-Ausdrücken:

```
(a|b)(c|d)     ac, ad, bc oder bd
a[b]c          abc oder 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, ...
```

Häufig möchte man eine syntaktisch gleiche Formel in verschiedenen Zusammenhängen, für verschiedene Zwecke verwenden. Um diesen Zusammenhang auch ausdrücken zu können, müsste man eine zusätzliche Formel schreiben:

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

Damit dieser Umweg vermieden werden kann, wird folgende abgekürzte Schreibweise angewendet:

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

Die Formel Class-Name wird nicht definiert. Syntaktisch kommt direkt die Regel "Name" zur Anwendung (vgl. Kapitel 2.2.2 Namen). Von der Bedeutung her ist der Name jedoch ein Klassenname. "Class" wird damit quasi zu einem Kommentar.

2.2 Grundsymbole der Sprache

Die Beschreibungssprache weist die folgenden Symbol-Klassen auf: Namen, Zeichenketten, Zahlen, Erläuterungen, Sonderzeichen, reservierte Wörter und Kommentare.

2.2.1 Zeichenvorrat, Zwischenräume und Zeilenenden

Die eigentliche Sprache verwendet nur die druckbaren US-ASCII-Zeichen (32 bis 126). Welche Zeichen nebst dem Leerzeichen als Zwischenräume gelten, ist durch die konkrete Compiler-Implementation festzulegen. Diese legt auch fest, welche Zeichen oder Zeichenkombinationen als Zeilenende gelten. Wie die Zeichen gespeichert werden (Zeichensatz) ist Sache der Implementation des Compilers. Sie kann insbesondere je nach Plattform unterschiedlich sein.

Im Rahmen von Kommentaren dürfen auch weitere Zeichen (z.B. Umlaute und Akzente) vorkommen.

2.2.2 Namen

Ein Name ist definiert als eine Folge von maximal 255 Buchstaben, Ziffern und Unterstrichen, wobei das erste Zeichen ein Buchstabe sein muss. Gross- und Kleinbuchstaben werden dabei unterschieden. Namen, die mit reservierten Wörtern der Sprache (vgl. Kapitel 2.2.7 Sonderzeichen und reservierte Wörter) zusammenfallen, sind unzulässig.

Syntaxregeln:

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

Näheres zur Eindeutigkeit und zum Gültigkeitsbereich von Namen ist im Kapitel 2.5.4 Namensräume beschrieben.

2.2.3 Zeichenketten

Zeichenketten (Strings) kommen im Zusammenhang mit Konstanten vor. Sie beginnen und enden mit einem Anführungszeichen und dürfen sich nicht über mehrere Zeilen erstrecken. \" steht für ein Anführungszeichen und \\ für einen Backslash innerhalb des Strings.

Eine Sequenz von \u, unmittelbar gefolgt von exakt vier Hexadezimalziffern, steht für ein beliebiges Unicode-Zeichen. Zeichen ab U+10000 sind wie in der UTF-16-Codierung mit zwei Surrogatcodes zu bezeichnen (siehe www.unicode.org/).

Syntaxregel:

```
String = '"' { <any character except '\' or '"'>
             | '\"'
             | '\\'
             | '\u' HexDigit HexDigit HexDigit HexDigit
             } '"'.
```

2.2.4 Zahlen

Zahlen kommen in verschiedener Weise vor: Positive ganze Zahlen inklusive 0 (PosNumber), ganze Zahlen (Number), Dezimalzahlen (Dec) und strukturierte Zahlen. Bei Dezimalzahlen kann die Skalierung als Zehnerpotenz angegeben werden (z.B. 1E2 ist 100, 1E-1 ist 0.1). Strukturierte Zahlen machen nur im Zusammenhang mit entsprechenden Einheiten und Wertebereichen (z.B. Uhrzeit) einen Sinn. Wesentlich ist der Wert der Zahl, nicht deren Darstellung. So ist etwa 007 dieselbe Zahl wie 7.

Syntaxregeln:

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

Beispiele:

```
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 Eigenschaftsmengen

Für verschiedene Zwecke müssen einem Beschreibungsgegenstand Eigenschaften zugeordnet werden. Dies kann mit einer generellen Syntax erfolgen:

Syntaxregel:

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

Um zu definieren, dass an einer bestimmten Stelle einer Syntaxregel solche Eigenschaften definiert werden sollen, wird in der Syntax folgendes Konstrukt eingefügt:

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

Man schreibt also "Properties" und definiert in spitzen Klammern die zulässigen Schlüsselwörter. Nimmt man als Beispiel die Regel ClassDef (vgl. Kapitel 2.5.3 Klassen und Strukturen) werden mit "Properties<ABSTRACT,EXTENDED,FINAL>" die Schlüsselwörter "ABSTRACT", "EXTENDED" und "FINAL" für die Beschreibung von Eigenschaften akzeptiert. In einer INTERLIS 2-Definition wären dann unter anderem folgende Definitionen möglich:

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

2.2.6 Erläuterungen

Erläuterungen werden dort verlangt, wo ein Sachverhalt näher beschrieben werden muss. Aus der Sicht des Standard-Mechanismus wird die Erläuterung nicht weiter interpretiert, also als Kommentar aufge-

fasst. Es ist aber durchaus zulässig, Erläuterungen detaillierter zu formalisieren und damit einer weitergehenden maschinellen Verarbeitung zugänglich zu machen. Als Anfang und Abschluss einer Erläuterung wurde // gewählt. Innerhalb der Erläuterung dürfen zwei aufeinander folgende Schrägstriche nie vorkommen.

Syntaxregel:

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

2.2.7 Sonderzeichen und reservierte Wörter

Sonderzeichen und reservierte Wörter sind in den Syntaxregeln der Sprache (nicht aber bei einer Datenbeschreibung) immer zwischen Apostrophen geschrieben, z.B. ',' oder 'MODEL'. Die reservierten Wörter werden grundsätzlich mit Grossbuchstaben geschrieben. Konflikte zwischen Namen und reservierten Wörtern sind vermeidbar, wenn Namen nicht ausschliesslich aus Grossbuchstaben bestehen.

Es werden folgende reservierten Wörter verwendet (einige davon wurden in INTERLIS 1 benutzt und bleiben aus Gründen der Kompatibilität reserviert; sie sind nicht fett und *kursiv* dargestellt):

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**	

Tabelle 1: Reservierte Wörter in INTERLIS 2.

2.2.8 Kommentare

Es werden zwei Kommentarformen angeboten:

2.2.8.1 Zeilenkommentar

Ein Zeilenkommentar wird mit zwei Ausrufezeichen eröffnet, die unmittelbar aufeinander folgen. Der Zeilenkommentar wird durch das Zeilenende abgeschlossen.

Syntaxregel:

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

2.2.8.2 Blockkommentar

Der Blockkommentar wird durch einen Schrägstrich und einen Stern eingeleitet; abgeschlossen wird er durch einen Stern und einen Schrägstrich. Er darf sich über mehrere Zeilen hinweg erstrecken und seinerseits Zeilenkommentare enthalten. Geschachtelte Blockkommentare sind zugelassen.

Syntaxregel:

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

2.3 Hauptregel

Jede Beschreibungseinheit beginnt mit der Angabe der Sprach-Version. Damit wird die Basis für spätere Sprachzusätze gelegt. Die in diesem Dokument beschriebene Version von INTERLIS ist **2.2**.

Nachher folgen die Modellbeschreibungen.

Syntaxregel:

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

2.4 Vererbung

Verschiedene Konstrukte von INTERLIS können im Sinne der objekt-orientierten Denkweise erweitert werden: Eine erste Definition schafft die Grundlage, die dann in mehreren Schritten spezialisiert werden kann.

Themen, Klassen, Sichten, Grafikdefinitionen, Einheiten und Wertebereiche können die entsprechenden Grundkonstrukte erweitern (Schlüsselwort EXTENDS bzw. EXTENDED) und erben damit alle ihre Eigenschaften. Ein zuvor definiertes Konstrukt kann in bestimmten Fällen durch Angabe von EXTENDED unter Beibehaltung des Namens erweitert werden.

Mit FINAL wird das Erweitern einer Definition verhindert. Verschiedene Konstrukte können in einer noch unvollständigen Form (Schlüsselwort ABSTRACT) definiert werden; sie werden später in einer Erweiterung zu einer konkreten Definition ergänzt.

Um die in einem bestimmten Kontext zulässigen Schlüsselwörter anzugeben, wird jeweils die generelle Property-Schreibweise verwendet (vgl. Kapitel 2.2.5 Eigenschaftsmengen).

2.5 Modelle, Themen, Klassen

2.5.1 Modelle

Als Modell wird eine in sich geschlossene, vollständige Definition bezeichnet. Je nach Art des Modells kann es verschiedene Konstrukte enthalten.

Ein reines Typenmodell (TYPE MODEL) darf nur Masseinheiten, Wertebereiche, Funktionen und Linienformen deklarieren.

Ein Referenzsystem-Modell (REFSYSTEM MODEL) soll nebst den Definitionen eines Typenmodells nur Themen und Klassen deklarieren, die im Bezug zu Erweiterungen der vordefinierten Klassen AXIS bzw. REFSYSTEM stehen (vgl. Kapitel 2.10.3 Referenzsysteme). Die Einhaltung dieser Regel kann durch die Sprache nicht erzwungen werden. Es ist Sache des Anwenders, sich daran zu halten.

Ein Signaturenmodell (SYMBOLOGY MODEL) soll nebst den Definitionen eines Typenmodells nur Themen und Klassen, die im Bezug zu Erweiterungen der vordefinierten Klasse SIGN stehen, sowie Laufzeitparameter deklarieren (vgl. Kapitel 2.16 Darstellungsbeschreibungen bzw. 2.11 Laufzeitparameter). Die Einhaltung dieser Regel ist ebenfalls Sache des Anwenders. Signaturenmodelle sind nur im Zusammenhang mit Kontrakten zulässig, da sie darauf abgestimmt sein müssen, wie die Systeme damit umgehen.

Wird keine solche einschränkende Modelleigenschaft definiert, darf ein Modell alle möglichen Konstrukte enthalten.

Nach dem Modellnamen kann optional die Sprache angegeben werden. Wenn möglich sollte die Angabe anhand der durch die ISO-Norm 639 standardisierten zwei-buchstabigen Bezeichnungen in Kleinbuchstaben erfolgen (vgl. www.iso.ch/); zum Beispiel steht "de" für Deutsch, "fr" für Französisch, "it" für Italienisch, "rm" für Rätoromanisch und "en" für Englisch. Durch einen Unterstrich getrennt darf ein Ländercode gemäss ISO-Norm 3166 nachgestellt werden, um die in einem bestimmten Land benutzte Varietät einer Sprache zu bezeichnen: "de_CH" steht für das in der Schweiz verwendete (Schrift-)Deutsch. Diese Angabe hat dokumentarischen Wert. Sie steht im Zusammenhang mit der Möglichkeit, ein Modell als eine Übersetzung eines anderen zu deklarieren (TRANSLATION OF). Die beiden Modelle müssen dann strukturell exakt übereinstimmen. Sie dürfen sich also nur in den verwendeten Namen unterscheiden. Die Deklaration als Übersetzung ist aber nicht an die Sprachangabe gebunden. Um z.B. lokale oder branchenspezifische Sprachgebräuche zu unterstützen, sind insbesondere auch Übersetzungen in die gleiche Sprache wie die Ursprungsbeschreibung zulässig.

Syntaxregel:

```
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 '.'.
```

Ist ein Modell mit einem Kontrakt verbunden, wird der Herausgeber des Kontraktes aufgeführt. In der Erläuterung können Präzisierungen (wie beispielsweise Postadresse, E-Mail-Adresse des Herausgebers) angegeben werden.

Bezieht sich ein INTERLIS-Konstrukt auf eine Definition, die in einem anderen Modell vorgenommen wurde, muss dieses Modell importiert werden (Schlüsselwort IMPORTS). So können beispielsweise Themen erweitert und der Bezug auf Klassen geschaffen werden. IMPORTS eröffnet aber nur die Möglichkeit des Gebrauchs. Bei der Verwendung der importierten Definitionen, sind diese dennoch mit qualifi-

ziertem Namen (Modell, Thema) zu referenzieren, ausser man verwendet das Schlüsselwort UNQUALIFIED.

Mit der Sprache verbunden ist auch ein vordefiniertes Basis-Modell "INTERLIS" (vgl. Anhang A Das interne INTERLIS-Datenmodell). Dieses muss nicht importiert werden. Hingegen stehen auch dessen Elemente nur dann mit unqualifizierten Namen zur Verfügung, wenn das Modell mit IMPORTS UNQUALIFIED INTERLIS eingeführt wird.

2.5.2 Themen

Ein Thema (Schlüsselwort TOPIC) enthält alle zur Beschreibung eines bestimmten sachlichen Teils der Realwelt nötigen Definitionen. Ein Thema kann auch Typen wie Masseinheiten, Wertebereiche oder Strukturen definieren oder diese vom umhüllenden Modell oder einem importierten Modell benützen.

In runden Klammern können die Vererbungseigenschaften definiert werden. Da sich eine Erweiterung eines Themas immer auf ein Thema mit einem anderen Namen bezieht, macht EXTENDED keinen Sinn und ist deshalb nicht zulässig.

Syntaxregeln:

```
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.
```

Zu einem bestimmten Thema, das konkrete Klassen enthält, können beliebig viele Behälter (Datenbanken etc.) existieren. Sie weisen alle die dem Thema entsprechende Struktur auf, enthalten aber unterschiedliche Objekte.

In einem Datenbehälter kommen dabei nur die Instanzen von Klassen (und ihrer Unterstrukturen) vor. Enthält ein Thema Darstellungsbeschreibungen, können drei Arten von Behältern gebildet werden:

- Datenbehälter.
- Behälter mit den Basisdaten für die Grafik. Solche Behälter enthalten die Instanzen von Klassen oder Sichten, welche die Grundlage der Darstellungsbeschreibungen bilden.
- Grafikbehälter. Solche Behälter enthalten die umgesetzten Grafikobjekte (entsprechend der Parameter-Definition der verwendeten Signaturen).

Zu Themen, in denen nur Sichten, nicht aber Klassen oder Grafikdefinitionen definiert sind und zudem explizit als Sicht-Themen (Schlüsselwort VIEW) gekennzeichnet sind, können auch Behälter bestehen, die Instanzen der Sichten des Themas enthalten.

Die Identifikationen der Behälter und ihrer identifizierbaren Objekte muss dem Wertebereich entsprechen, der dem Thema dafür zugeordnet ist (OID AS). Wurde einem Thema ein OID-Wertebereich zugeordnet,

kann die Zuordnung in Erweiterungen nicht mehr geändert werden. Erfolgte keine Zuordnung gilt implizit der OID-Wertebereich STANDARDOID (vgl. Kapitel 2.8.8 Wertebereiche von Objektidentifikationen sowie Anhänge A und E). Explizite Definitionen sind nur im Rahmen von Kontrakten zulässig.

Ohne weitere Angaben ist ein Thema datenmässig unabhängig von anderen Themen. Datenmässige Abhängigkeiten entstehen als Folge von Beziehungen bzw. Referenzattributen, die in einen anderen Behälter verweisen, durch spezielle Konsistenzbedingungen oder durch den Aufbau von Sichten oder Grafikdefinitionen auf Klassen oder anderen Sichten. Im Interesse der klaren Erkennbarkeit solcher Abhängigkeiten, müssen diese aber bereits im Themenkopf explizit deklariert werden (Schlüsselwort DEPENDS ON). Die detaillierten Definitionen (z.B. Beziehungsdefinitionen) dürfen dann die Abhängigkeitsdeklaration nicht verletzen. Zyklische Abhängigkeiten sind nicht erlaubt. Ein erweitertes Thema besitzt ohne zusätzliche Deklarationen dieselben Abhängigkeiten wie sein Basis-Thema.

2.5.3 Klassen und Strukturen

Eine Klassendefinition (Schlüsselwort CLASS) deklariert die Eigenschaften aller zugehörigen Objekte. Klassendefinitionen sind erweiterbar, wobei eine Erweiterung primär sämtliche Attribute ihrer Basis-Klasse erbt. Deren Wertebereiche dürfen eingeschränkt werden. Zudem dürfen weitere Attribute definiert werden.

Als Teil einer Klassendefinition können Konsistenzbedingungen angegeben werden. Die Bedingungen stellen Regeln dar, welchen die Objekte zusätzlich genügen müssen. Ererbte Konsistenzbedingungen können nicht ausser Kraft gesetzt werden, sondern gelten immer zusätzlich zu den lokal deklarierten.

Die Objekte einer Klasse sind immer selbständig und individuell identifizierbar. Strukturen (Schlüsselwort STRUCTURE) sind zwar formal gleich wie Klassen definiert, ihre Strukturelemente sind jedoch unselbständig und können nicht einzeln identifiziert werden. Sie kommen entweder innerhalb von Unterstrukturen von Objekten vor (vgl. Kapitel 2.6 Attribute), oder sie existieren nur temporär als Ergebnisse von Funktionen. Überall dort, wo Strukturen verwendet werden, kann auch auf die Definition einer Klasse verwiesen werden. Man nennt das den strukturellen Gebrauch der Klasse. In dieser Anwendungsform besitzen Klassen aber die Bedeutung von Strukturen, ihre Objekte sind also unselbständig.

Spezielle Klassen wie diejenigen für Referenzsysteme, Koordinatensystem-Achsen und Grafik-Signaturen (also Erweiterungen der vordefinierten Klasse METAOBJECT) werden im Kapitel 2.10 beschrieben.

In runden Klammern (Regel Properties) können die Vererbungseigenschaften definiert werden. Es sind alle Möglichkeiten zulässig. Enthält eine Klasse oder Struktur abstrakte Attribute, ist sie als ABSTRACT zu deklarieren. Abstrakte Attribute müssen dann in konkreten Erweiterungen der Klasse noch konkretisiert werden. Es ist aber auch zulässig, Klassen als abstrakt zu erklären, deren Attribute vollständig definiert sind. Objektinstanzen können nur für konkrete Klassen existieren, die innerhalb eines Themas definiert wurden. Klassen, die ausserhalb von Themen (also direkt im Modell) definiert sind, dürfen keine Referenzattribute enthalten. Es ist auch nicht zulässig, Assoziationen zu solchen Klassen zu definieren.

Wird eine einzelne Klasse und nicht ein ganzes Thema geerbt, dürfen keine Beziehungen (vgl. Kapitel 2.7 Eigentliche Beziehungen) zu ihr definiert sein.

Erweitert ein Thema ein anderes, werden alle Klassen des geerbten Themas übernommen. Sie werden also zu Klassen das aktuellen Themas und haben denselben Namen wie im geerbten Thema. Eine solche Klasse kann auch unter Beibehaltung ihres Namens erweitert werden (EXTENDED). Erweitert z.B. ein Thema T2 das Thema T1, das die Klasse C enthält, gibt es mit C (EXTENDED) innerhalb von T2 nur eine Klasse, nämlich C. Neue Klassen, die sich namensmässig von den geerbten unterscheiden müssen, dürfen auch geerbte erweitern. Mit C2 EXTENDS C, gibt es dann in T2 zwei Klassen (C und C2). Da INTERLIS im Interesse von Einfachheit und Klarheit nur Einfachvererbung unterstützt, ist EXTENDED

allerdings nur zulässig, wenn weder im Basis-Thema noch im aktuellen Thema die Basis-Klasse mit EXTENDS erweitert wurde. EXTENDED und EXTENDS schliessen sich in der gleichen Klassendefinition aus.

Syntaxregeln:

```
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 ).
```

Welche Namen qualifiziert werden müssen (durch Model-Name bzw. durch Model-Name.Topic-Name) ist am Schluss des folgenden Abschnitts (2.5.4 Namensräume) erklärt. Klassen und Strukturen, die nicht auf einer bereits definierten Klasse oder Struktur aufbauen, brauchen keinen EXTENDS-Teil.

2.5.4 Namensräume

Als Namensraum bezeichnet man eine Menge von (eindeutigen) Namen. Jedes Modellierungselement (Datenmodell, Thema, Klassenelement) sowie die Metadaten-Behälter stellen jeweils einen eigenen Namensraum für ihre Namenskategorien (Typnamen, Bestandteilnamen, Metaobjektnamen) bereit.

Modellierungselemente gibt es auf drei Hierarchiestufen:

- Modell (MODEL ist einziges Modellierungselement auf oberster Stufe)
- Thema (TOPIC ist einziges Modellierungselement auf dieser Stufe)
- Klassenelemente sind Klasse (CLASS), Struktur (STRUCTURE), Assoziation (ASSOCIATION), Sicht (VIEW), Grafikdefinition (GRAPHIC)

Metadaten-Behälter-Namen eröffnen den Zugang zu den Metaobjekten (vgl. Kapitel 2.10 Umgang mit Metaobjekten).

Es gibt drei Namenskategorien, die folgende Namen enthalten:

- Typnamen sind die Kurzzeichen (Namen) von Einheiten und die Namen von Funktionen, Linienformtypen, Wertebereichen, Strukturen, Themen, Klassen, Assoziationen, Sichten, Grafikdefinitionen, Behältern.
- Bestandteilnamen heissen die Namen von Laufzeitparametern, Attributen, Zeichnungsregeln, Parametern, Rollen, Beziehungszugängen und Basissichten.
- Metaobjektnamen heissen die Namen von Metaobjekten. Sie existieren nur innerhalb von Metadatenbehältern.

Um Missverständnisse auszuschliessen übernehmen Modellierungselemente zudem die Namen des übergeordneten Modellierungselementes entsprechend der Namenskategorie. Erweitert ein Modellierungselement ein anderes, werden seinen Namensräumen weiter alle Namen des Basis-Modellierungselementes zugefügt. Ein lokal im Modellierungselement definierter Name darf nicht mit einem übernommenen Namen kollidieren, es sei denn es handle sich ausdrücklich um eine Erweiterung (EXTENDED).

Will man Beschreibungselemente des Datenmodells referenzieren, muss ihr Name normalerweise qualifiziert, d.h. mit vorangestelltem Modell- und Thema-Namen angegeben werden. Unqualifiziert können die Namen der Namensräume des jeweiligen Modellierungselementes verwendet werden.

2.6 Attribute

2.6.1 Allgemeines

Jedes Attribut wird durch seinen Namen und seinen Typ definiert. In runden Klammern (Regel Properties) können die Vererbungseigenschaften definiert werden. Ist ein Attribut eine Erweiterung eines geerbten Attributes, muss dies mit EXTENDED ausdrücklich angemerkt werden. Ist der Wertebereich eines Attributs abstrakt, muss das Attribut als ABSTRACT deklariert werden.

Syntaxregel:

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

Wird der Attributwert mittels eines Faktors (vgl. Kapitel 2.13 Ausdrücke) festgelegt, muss dessen Ergebnistyp zuweisungskompatibel zum definierten Attribut sein, d.h. es muss den gleichen Wertebereich oder einen erweiterten, d.h. spezialisierten Wertebereich aufweisen. Im Rahmen von Sichten - vor allem bei Vereinigungen und Sichterweiterungen (vgl. Kapitel 2.15 Sichten) – können mehrere Faktoren festgelegt werden und in zusätzlichen Sichterweiterungen noch zusätzliche beigefügt werden. Es gilt jeweils der letzte (Basis zuerst, Erweiterung anschliessend), dessen Wert definiert ist.

Ein Attribut kann in Erweiterungen wie folgt übersteuert werden:

- durch einen eingeschränkten Wertebereich.
- durch eine Konstante aus dem verlangten Wertebereich. Eine solche Definition ist implizit final, d.h. sie kann nicht mehr weiter übersteuert werden.
- durch einen Factor, wenn der Typ des Ergebnisses als Erweiterung des Attributs zulässig wäre. Eine weitere Übersteuerung ist zulässig.

Syntaxregeln:

```
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 } ].
```

Im Rahmen von Erweiterungen ist es zulässig, nur MANDATORY anzugeben. Es gilt dann der bereits definierte Attributtyp. Es wird aber verlangt, dass der Wert immer definiert ist.

2.6.2 Attribute mit Wertebereichen als Typ

Als Typ eines Attributs kommen direkte Typendefinitionen (Regel Type) und die Verwendung bereits definierter Wertebereiche (Regel DomainRef) in Frage. Die verschiedenen Möglichkeiten sind im Kapitel 2.8 Wertebereiche und Konstanten aufgeführt.

2.6.3 Referenzattribute

Mit einem Referenzattribut kann der Verweis zu einem anderen Objekt geschaffen werden. Referenzattribute sind nur innerhalb von Strukturen zulässig. Eine Struktur, die direkt oder indirekt (über Unterstrukturen) Referenzattribute enthält, darf darum nicht zu einer Klasse erweitert werden. Zusammenhänge zwischen eigenständigen Objekten sind mittels eigentlichen Beziehungen (vgl. Kapitel 2.7) zu definieren.

Die Klassen, deren Objekte für den Bezug in Frage kommen, dürfen konkrete oder abstrakte Objekt- oder Beziehungsklassen, jedoch keine Strukturen sein (da diese keine eigenständigen Objekte sind). Dabei kommen alle konkreten Klassen in Frage, die der aufgeführten primären bzw. einer der aufgeführten einschränkenden (RESTRICTED TO) Klassen entsprechen (Klasse selbst oder Unterklasse davon). Auf jeder Restriktionsstufe (Erstdefinition oder in Erweiterungsschritten) müssen jeweils alle noch zulässigen Klassen aufgeführt werden. Jede als Einschränkung definierte Klasse muss Unterklasse einer bisher zulässigen Klasse sein. Eine so in Frage kommende Klasse ist allerdings nur zulässig, wenn sie zum selben Thema wie das Referenzattribut oder zu einem Thema gehört, von denen das referenzierende Thema abhängig (DEPENDS ON) ist. Soll die Referenz auf ein Objekt eines anderen Behälters des gleichen oder eines anderen (DEPENDS ON vorausgesetzt) Themas verweisen dürfen, muss die Eigenschaft EXTERNAL angegeben werden. In Erweiterungen kann diese Eigenschaft weggelassen und damit ausgeschlossen, nicht aber zugefügt werden. Es muss mindestens eine zulässige konkrete Unterklasse geben, wenn das Referenzattribut nicht als abstrakt deklariert ist.

2.6.4 Strukturattribut

Strukturattributwerte (Regel RestrictedStructureRef) bestehen aus einem (weder LIST noch BAG verlangt) oder mehreren (zulässige Anzahl im Rahmen der angegebenen Kardinalität) geordneten (LIST) oder ungeordneten (BAG) Strukturelementen. Strukturelemente haben keine OID, existieren nur im Zusammenhang mit ihrem Hauptobjekt und sind auch nur über dieses auffindbar. Ihr Aufbau ergibt sich durch die angegebene Struktur oder Klasse. Die Gesamtheit der Strukturelemente, aus der die Strukturelemente aller Strukturattributswerte einer Klasse entnommen sind, nennt man Unterstruktur.

Ist die Struktur oder Klasse der Unterstruktur beliebig (ANYSTRUCTURE oder ANYCLASS), muss das Strukturattribut als abstrakt deklariert werden, sofern es obligatorisch ist bzw. eine minimale Kardinalität grösser null hat. ANYSTRUCTURE darf in Erweiterung zu ANYCLASS spezialisiert werden, sofern keine Restriktionen (RESTRICTED TO) definiert sind.

Unterstrukturen dürfen mit konkreten oder abstrakten Strukturen definiert werden. Als konkrete Strukturen kommen zunächst alle primär oder einschränkend (RESTRICTED TO) aufgeführten Klassen bzw. Strukturen und ihre erweiterten Klassen bzw. erweiterten Strukturen in Frage. Im aktuellen Thema sind dies ohne weitere Einschränkung alle konkreten erweiterten Klassen bzw. erweiterten Strukturen. Ausserhalb des aktuellen Themas müssen die verlangten Klassen oder Strukturen explizit aufgeführt werden. Auf jeder Restriktionsstufe (Erstdefinition oder in Erweiterungsschritten) müssen jeweils alle noch zulässigen Klassen oder Strukturen aufgeführt werden. Jede als Erweiterung definierte Klasse oder Struktur muss Erweiterung einer bisher zulässigen Klasse oder Struktur sein. Es muss mindestens eine konkrete Erweiterung der Struktur geben, wenn die Unterstruktur nicht als abstrakt deklariert ist. Eine geordnete Unter-

struktur (LIST) darf nicht durch eine ungeordnete (BAG) erweitert werden. Für die Kardinalität gelten die gleichen Regeln wie bei Beziehungen (vgl. Kapitel 2.7.3 Kardinalität)**.**

2.7 Eigentliche Beziehungen

2.7.1 Allgemeines

Eigentliche Beziehungen (im Gegensatz zu Referenzattributen; vgl. Kapitel 2.6 Attribute) werden als eigenständige Konstrukte beschrieben. Die wichtigste Eigenschaft einer Beziehung besteht in der Auflistung der in den Rollen zugeordneten Objektklassen (mindestens deren zwei) inklusive der Detaileigenschaften wie Stärke der Beziehung und Kardinalität. Die Rollennamen sollen typischerweise Substantive sein. Sie können, müssen aber nicht mit den Namen der zugeordneten Objektklassen übereinstimmen. Zudem kann die Beziehung selbst lokale Attribute aufweisen. Ist der Assoziationsname nicht angegeben, wird er implizit aus der Zusammensetzung der Rollennamen (erster, dann zweiter, usw.) gebildet.

In verschiedener Hinsicht ist eine Beziehung eine etwas spezielle Klasse. Was die Vererbung betrifft, gelten darum die gleichen Festlegungen wie bei Klassen. Damit die Bedeutung der Kardinalität klar und unveränderlich ist, dürfen in Erweiterungen keine zusätzlichen Rollen beigefügt werden. Werden in Erweiterungen von Beziehungen, die zulässigen Bezugsklassen gegenüber der Basis-Assoziation eingeschränkt, sind nur noch diejenigen Beziehungen erlaubt, die explizit aufgeführt sind. Wird zwischen den Klassen A und B eine Beziehung definiert und die Klassen A und B um Unterklassen A1, A2, B1, B2, etc. erweitert, sind zunächst beliebige Beziehungen zwischen Objekten von A1, A2 und B1, B2 zulässig. Gibt es aber eine Beziehungserweiterung z.B. deren Rollen die Beziehung auf A1 und B1 einschränken, sind zwischen Objekten der Klassen A1 und B1 nur solche Beziehungen zulässig. Es ist aber erlaubt, mehrere solcher Beziehungserweiterungen zu definieren, z.B. auch eine solche zwischen A1 und B3. Dann könnte ein Objekt von A1 also mit Objekten von B1 oder B3 nicht aber mit solchen von B2 verbunden werden.

Syntaxregeln:

```
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 | '*' ) ] ) '}'.
```

Für die im Rahmen der Rollen als zulässig definierten Objekt- und Beziehungsklassen gelten die gleichen Regeln wie bei den Referenzattributen (vgl. Kapitel 2.6.3).

Eine konkrete Beziehung zwischen Objekten kann als eigenständiges Objekt betrachtet werden. Es erhält eine eigene Objektidentifikation, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Beziehung hat mehr als zwei Rollen.
- Die Beziehung hat zwei Rollen, deren maximale Kardinalität je grösser als eins ist.

Ist die maximale Kardinalität mindestens einer Rolle einer Zweierbeziehung bereits in der Basisdefinition (nicht erst in einer Erweiterung) eins, erhält das Beziehungsobjekt keine eigene Objektidentifikation (aber die OID des umschliessenden Objektes). Das Beziehungsobjekt wird als Strukturattribut desjenigen Objektes aufgefasst, das der Rolle mit maximaler Kardinalität grösser eins bzw. der zweiten Rolle entspricht, wenn beide Rollen die maximale Kardinalität eins aufweisen. Als weitere Bedingung gilt, dass die Klasse, die das Strukturattribut zu übernehmen hat, im selben Thema definiert sein muss wie die Beziehungsklasse. Einerseits wird damit erreicht, dass Implementierungen nicht unnötig Objektidentifikationen führen müssen. Andererseits wird die Möglichkeit der Kompatibilität mit INTERLIS 1 eröffnet (vgl. Kapitel 3 Transfer).

Normalerweise müssen die konkreten Beziehungen zwischen Objekten mittels einer Anwendung explizit erstellt und dann durch das Bearbeitungssystem als Instanz festgehalten werden. Eine Beziehung kann aber auch aus einer Sicht abgeleitet werden, ohne dass sie instanziiert wird (DERIVED FROM). Eine solche Beziehung kann eine Erweiterung einer abstrakten Beziehung sein. Sie kann nicht selbst abstrakt sein. Wird sie erweitert, muss die Erweiterung auf der gleichen Sicht oder einer Erweiterung davon aufbauen. Allen Rollen und Attributen müssen entsprechend Objektpfade oder Attributpfade der Sicht zugewiesen sein. Dabei muss ein Objektpfad (vgl. Kapitel 2.13 Ausdrücke) angegeben werden, der letztlich eine der Rolle entsprechende Klasse oder Assoziation bezeichnet. Die Kardinalität muss mit der Leistung der Sicht übereinstimmen. Dies kann aber nur zur Laufzeit geprüft werden.

Ein typischer Anwendungsfall dürfte die Herleitung einer Beziehung aus den geometrischen Verhältnissen sein: In einer Sicht (Verbindung), auf die in der Assoziation Bezug genommen wird, werden z.B. Gebäude auf Grund der Geometrie mit den Liegenschaften, auf denen sie stehen, in Beziehung gebracht (vgl. Kapitel 2.15 Sichten).

2.7.2 Stärke der Beziehung

In Anlehnung an UML werden verschiedene Beziehungsstärken unterschieden. Für ihre Erklärung wird vor allem beschrieben, welchen Einfluss die Beziehungsstärke beim Kopieren und Löschen von Objekten hat. Darüber hinaus gibt es noch andere Überlegungen, die die Beziehungsstärke beeinflussen. Insbesondere ist es den Bearbeitungssystemen überlassen, feinere Beziehungsstärken oder gar andere Kriterien für das Verhalten bei bestimmten Operationen vorzusehen.

- Assoziation: Die beteiligten Objekte sind lose miteinander verbunden. Wird ein beteiligtes Objekt kopiert, ist die Kopie mit denselben Objekten verbunden wie das Original. Wird ein beteiligtes Objekt gelöscht, wird die Beziehung ebenfalls gelöscht, das verbundene Objekt bleibt aber bestehen. Syntaktisch wird bei allen Rollen '--' angegeben.
- Aggregation: Es besteht eine schwache Beziehung zwischen einem Ganzen und seinen Teilen. Aggregationen sind nur in Beziehungen mit zwei Rollen erlaubt. Syntaktisch muss die Rolle, die zum Ganzen führt, die erste sein und mit einem Rhombus (-<>) angegeben sein. Die Rolle, die zum Teil führt, wird mit '--' definiert. Eine Objektklasse kann in verschiedenen Aggregationen in der Teile-Rolle auftreten. Einem bestimmten Teile-Objekt können dann auch verschiedene Ganze-Objekte zugeordnet sein. Anders als bei Assoziationen werden als Folge der Erstellung einer Kopie des Ganzen auch entsprechende Kopien der Teile erstellt. Sind für einen Teil mehrere Ganze definiert, bleibt in der Kopie nur der Bezug zu demjenigen Ganzen erhalten, von dem die Kopie veranlasst wurde. Bei der Löschung des Ganzen bleiben die Teile und allfällige Beziehungen zu anderen Ganzen erhalten.
- Komposition: Es besteht eine starke Beziehung zwischen dem Ganzen und seinen Teilen. Eine Objektklasse darf in mehr als einer Komposition in der Teile-Rolle auftreten. Einem bestimmten Teile-Objekt darf aber höchstens ein Ganzes zugeordnet sein. Sonst verhalten sich Kompositionen gleich wie Aggregationen, ausser dass bei der Löschung des Ganzen auch seine Teile gelöscht

werden. Bei der Rolle, die zum Ganzen führt, wird ein gefüllter Rhombus (-<#>) angegeben.
Assoziationen dürfen zu Aggregationen, diese zu Kompositionen erweitert werden, nicht aber umgekehrt.

2.7.3 Kardinalität

Die Kardinalität definiert die minimale und die maximale Anzahl erlaubter Objekte; steht nur ein Wert, ist das Minimum gleich dem Maximum. Steht als Maximum ein Stern anstatt einer Zahl, gibt es keine obere Schranke für die Anzahl der Unterobjekte. Die Kardinalitätsangabe {*} ist äquivalent mit {0..*}. Falls die Kardinalitätsangabe weggelassen wird, gilt normalerweise {0..*}. Bei Aggregations- und Kompositionsrollen ist nur {0..1} oder {1} zugelassen (ein Teil kann nur zu einem Ganzen gehören). Fehlt die Angabe gilt {1}.

Die Kardinalität darf in Erweiterungen nur eingeschränkt, nicht jedoch erweitert werden. Wird also zunächst eine Kardinalität von {2..4} angegeben, darf eine Erweiterung nicht {2..5}, {7} oder {*} deklarieren. Das Weglassen der Kardinalitätsangabe wird bei erweiterten Attributen als Übernehmen des ererbten Wertes verstanden.

Je nach Verwendung hat die Kardinalität folgende Bedeutung:

- Bei Unterstrukturen: Anzahl der zulässigen Elemente.
- Bei Zweierbeziehungen: Anzahl der Objekte der Klasse gemäss der Rolle zu der die Kardinalität gehört, die jeder Instanz der Klasse der anderen Rolle zugeordnet werden dürfen.
- Bei Mehrfachbeziehungen: Anzahl der Objekte der Klasse gemäss der Rolle zu der die Kardinalität gehört, die einer bestimmten Kombination von Objekten der Klassen der übrigen Rollen zugeordnet werden dürfen.

2.7.4 Geordnete Beziehungen

Will man erreichen, dass die Beziehung aus der Sicht einer bestimmten Bezugsklasse in einer bestimmten Ordnung geführt wird, muss dies bei der Rolle als Eigenschaft (ORDERED) verlangt werden. Diese Ordnung wird beim Etablieren der Beziehung definiert und muss bei Transfers und beim Erstellen von Kopien (sofern die Beziehung überhaupt mitkopiert wird) erhalten bleiben.

2.7.5 Beziehungszugänge

Als Beziehungszugang wird die Möglichkeit bezeichnet, Zugang zu allen einem Objekt über eine bestimmte Beziehung zugeordneten Objekten bzw. Zugang zu allen zugeordneten Beziehungsobjekten zu erhalten.

Beziehungszugänge müssen nicht definiert werden, sondern entstehen mit der Definition einer Beziehung für alle über Rollen zugeordneten Klassen, die im gleichen Thema wie die Beziehung definiert wurden. Ist eine an einer Beziehung beteiligte Klasse in einem anderen Thema definiert (themenübergreifende Beziehung oder geerbte Klasse) erhält sie keine Beziehungszugänge. Damit wird vermieden, dass die Klasse nachträglich (d.h. ausserhalb des Rahmens, in dem sie definiert wurde) nochmals eine Änderung erfährt.

Werden bei einer Rollendefinition die zulässigen Klassen auf bestimmte Subklassen der erwähnten Basisklasse beschränkt (RESTRICTED TO), entstehen die Beziehungszugänge für die Basisklasse und werden damit an alle Subklassen vererbt. Bei denjenigen Subklassen, die durch die Beschränkung ausgeschlossen sind, besteht der Beziehungszugang also auch. Über ihn kann aber nie ein zugeordnetes Objekt angesprochen werden, weil die Zuordnung auf Grund der Beschränkung nicht möglich ist.

Für jede Rolle einer Beziehung, entstehen unter der erwähnten Bedingung in der zugeordneten Basisklasse Beziehungszugänge (AssociationAccess), die den anderen Rollen der Beziehung entsprechen und auch deren Namen erhalten. Die Namen dieser Beziehungszugänge gehören zum selben Namens-

raum wie die Namen der Attribute der Klasse, dürfen also nicht mit diesen kollidieren. Der Zugang zur Beziehung selbst erhält keinen eigenen Namen sondern ergibt sich aus einem Beziehungszugang mit einem vorangestellten Backslash ('\').

Sind die Klassen A, B und C über die Beziehung ABC miteinander verbunden, wobei der Rolle a die Klasse A, der Rolle b die Klasse B und der Rolle c die Klasse C zugeordnet ist, haben die Beziehungszugänge der Klasse A die Namen b und c. Über sie können die zugeordneten Objekte der Klasse B bzw. der Klasse C erreicht werden. Der Zugang zu den Beziehungsobjekten der Klasse ABC wird gleichberechtigt über \a bzw. \b erreicht.

2.8 Wertebereiche und Konstanten

Mit der Vorstellung eines Wertebereichs sind verschiedene Aspekte verbunden. Primär muss ein Datentyp festgelegt werden. Die INTERLIS-Datentypen sind unabhängig von der Implementation. Es wird deshalb z.B. nicht von Integer oder Real, sondern einfach von numerischen Datentypen gesprochen (vgl. Kapitel 2.8.5 Numerische Datentypen).

Ist der Datentyp festgelegt, sind – je nach Datentyp – noch weitere Präzisierungen nötig oder möglich. Ist eine Wertebereich-Definition noch unvollständig (fehlt z.B. bei einem Zeichenketten-Domain noch die Länge), muss sie als abstrakt deklariert werden (Schlüsselwort ABSTRACT, Regel Properties).

Wertebereiche können – wie andere Konstrukte – auch geerbt und dann erweitert werden, sofern sie nicht als FINAL definiert wurden. Wichtig ist dabei der Grundsatz, dass eine erweiterte Definition immer mit der Basis-Definition kompatibel sein muss. Bei Wertebereichen sind Erweiterungen (Schlüsselwort EXTENDS) somit eigentlich Präzisierungen bzw. Einschränkungen. Das Schlüsselwort EXTENDED (Regel Property) ist nicht zulässig. Im Interesse der Lesbarkeit wird empfohlen, Definitionsteile von Basis-Wertebereichen (z.B. die Masseinheit) in der Erweiterung auch dann zu wiederholen, wenn sie unverändert sind. Beispiel:

```
DOMAIN
  Text (ABSTRACT) = TEXT;             !! abstrakter Wertebereich
  GenName EXTENDS Text = TEXT*12;     !! konkrete Erweiterung
  SpezName EXTENDS GenName = TEXT*10; !! in Ordnung
  SpezName EXTENDS GenName = TEXT*14; !! falsch, da unverträglich
```

Eine wichtige Frage bei der Definition von Wertebereichen ist, ob der Wert "Undefiniert" auch zum Wertebereich gehört oder nicht. Ohne weitere Angabe gehört er dazu. Es kann aber verlangt werden, dass er nicht dazu gehört, d.h. dass ein Attribut mit diesem Wertebereich immer definiert sein muss (Schlüsselwort MANDATORY). MANDATORY allein ist nur bei Erweiterungen zulässig.

Bei der Definition eines Attributs einer Klasse oder Struktur (und nur dort) darf auch dann MANDATORY stehen, wenn der Wertebereich als FINAL deklariert wurde und somit eigentlich nicht weiter eingeschränkt werden dürfte.

Syntaxregeln:

```
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 ).
```

In Vergleichsoperationen (vgl. Kapitel 2.13 Ausdrücke) können Attributwerte auch mit Konstanten verglichen werden. Diese sind wie folgt definiert:

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

Die typenspezifischen Konstanten sind bei den einzelnen Datentypen definiert.

2.8.1 Zeichenketten

Beim Datentyp Zeichenkette (TEXT) ist primär die Länge der Zeichenkette von Interesse. Je nach der Form der Definition wird sie explizit oder implizit angegeben. Bei der expliziten Form (TEXT * ...) wird die maximale Länge in Anzahl Zeichen festgelegt (grösser Null). Wird nur das Schlüsselwort TEXT angegeben, gilt die Definition als abstrakt. Ist ein Zeichenketten-Wertebereich mit seiner Länge definiert, kann die Länge im Rahmen einer Erweiterung nur noch verkürzt werden (eine Verlängerung würde zu einem Wertebereich führen, der mit dem Basis-Wertebereich nicht mehr verträglich ist).

Die INTERLIS-Zeichenkettenlänge bezeichnet die Zahl der Zeichen, wie sie von Benutzern wahrgenommen wird, nicht aber die Zahl der Speicherstellen, die ein System maximal zur Repräsentation einer Zeichenkette benötigt.

Bemerkung: Im Zusammenhang mit INTERLIS ist die Länge einer Zeichenkette als Anzahl jener Zeichen definiert, welche gemäss Unicode-Standard die kanonische Kombinationsklasse Nr. 0 besitzen, nachdem die Zeichenkette in die kanonisch dekomponierte Form von Unicode gebracht wurde (vgl. www.unicode.org/unicode/reports/tr15/). So besitzt etwa eine Zeichenkette, die aus der Kette <LATIN CAPITAL LETTER C WITH CIRCUMFLEX><COMBINING CEDILLA> besteht, ebenso die Länge 1 wie die äquivalente Zeichenkette <LATIN CAPITAL LETTER C>< COMBINING CIRCUMFLEX ACCENT><COMBINING CEDILLA>. Gemäss der obigen Definition besitzen Ligaturen für "fl" oder "ffi" die Länge 1. Es wird aber davon abgeraten, solche Darstellungsformen überhaupt für Zeichenkettenattribute zu verwenden.

Der Namen-Zeichenkettentyp (Schlüsselwort NAME) definiert einen Wertebereich, der genau demjenigen der INTERLIS-Namen entspricht (vgl. Kapitel 2.2.2 Namen). Er wird in der vordefinierten Klasse METAOBJECT (vgl. vordefiniertes Basismodell INTERLIS) und damit vor allem in den Klassen für Referenzsysteme sowie Signaturen eingesetzt (vgl. Kapitel 2.10.3 Referenzsysteme sowie Kapitel 2.16 Darstellungsbeschreibungen), weil dort Datenattribute mit Beschreibungselementen von Modellen übereinstimmen müssen.

Als weiterer Zeichenkettentyp wird der URI (Uniform Resource Identifier) geführt, z.B. http-, ftp- oder mailto-Adressen (vgl. Kapitel 1.2 im Internet-Standard IETF RFC 2396 in www.w3.org/). Die Länge eines URI ist in INTERLIS auf 1023 Zeichen beschränkt. Er entspricht damit folgender Definition:

```
DOMAIN
  URI (FINAL) = TEXT*1023;  !! ACHTUNG: gemäss IETF RFC 2396
  NAME (FINAL) = TEXT*255;  !! ACHTUNG: gemäss Kapitel 2.2.2 Namen
```

Syntaxregeln:

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

TextConst = String.
```

2.8.2 Aufzählungen

Mit einer Aufzählung werden die für diesen Typ zulässigen Werte festgelegt. Die Aufzählung ist jedoch nicht einfach linear, sondern im Sinne eines Baumes strukturiert. Die Blätter des Baumes (nicht aber die Knoten) bilden die Menge der zulässigen Werte. Beispiel:

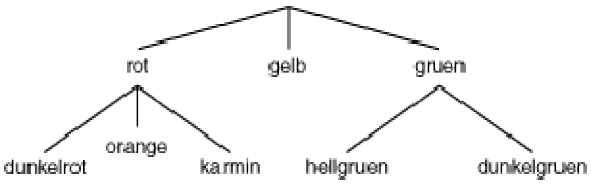
```
DOMAIN
  Farben = (rot (dunkel, karmin, orange),
            gelb,
            gruen (hell, dunkel));
```

ergibt die folgenden - mittels Konstanten beschriebenen - zulässigen Werte:

```
#rot.dunkel #rot.karmin #rot.orange #gelb #gruen.hell #gruen.dunkel
```

Eine Schachtelung wird in runden Klammern angegeben. Die Elementnamen jeder Schachtelung müssen eindeutig sein. Die Schachtelungstiefe ist frei wählbar.

Ist eine Aufzählung geordnet (Schlüsselwort ORDERED), ist eine Reihenfolge der Elemente definiert. Ist die Aufzählung zirkulär (Schlüsselwort CIRCULAR), ist die Reihenfolge der Elemente definiert, wie wenn die Aufzählung geordnet wäre. Zudem wird ausgesagt, dass nach dem letzten Element wieder das erste folgt.



Figur 8: Beispiel einer Aufzählung.

Beispiele:

```
DOMAIN
  Lage = (unten, mitte, oben) ORDERED;
  Wochentage = (Werktage (Montag, Dienstag, Mittwoch,
                          Donnerstag, Freitag, Samstag),
                Sonntag) CIRCULAR;
```

Syntaxregeln:

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

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

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

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

Im Rahmen von Neudefinitionen von Aufzählungen (Primärdefinition, zusätzliche Elemente einer Erweiterung) darf die EnumerationConst des EnumElement nur aus einem Namen bestehen. Mehrere Namen sind nur zulässig, um für eine Erweiterung ein bisheriges Aufzählungselement zu identifizieren.

Aufzählungen können einerseits erweitert werden, indem für Blätter (also Aufzählungselemente, die keine Unter-Aufzählung aufweisen) der bisherigen Aufzählung Unter-Aufzählungen definiert werden. In der erweiterten Definition werden aus bisherigen Blättern neu Knoten, für die keine Werte definiert werden dürfen.

Andererseits kann jede einzelne Teilaufzählung in Erweiterungen durch weitere Elemente (Knoten oder Blätter) ergänzt werden. Die Basisaufzählungen umfassen dadurch nebst den genannten Elementen immer auch noch potenziell weitere Elemente, die erst in Erweiterungen definiert werden. Solche potenziellen Werte können auf der Basisstufe in Ausdrücken, Funktionsargumenten und Signaturzuweisungen (vgl. Kapitel 2.13 Ausdrücke, Kapitel 2.14 Funktionen, Kapitel 2.16 Darstellungsbeschreibungen) mit dem Wert OTHERS angesprochen werden. OTHERS ist jedoch kein zulässiger Wert im Rahmen der Klasse, zu der das Objekt gehört. Die Möglichkeit, in Erweiterungen zusätzliche Aufzählelemente anfügen zu können, kann unterbunden werden, indem die Teilaufzählung als abschliessend erklärt wird (FINAL). Dies erfolgt entweder nach dem letzten aufgeführten Element oder im Rahmen einer Erweiterung auch ohne dass neue Elemente angefügt werden.

Zirkuläre Aufzählungen (Schlüsselwort CIRCULAR) können nicht erweitert werden.

Beispiel:

```
DOMAIN
  Farbe = (rot,
           gelb,
           gruen)
  FarbePlus EXTENDS Farbe = (rot (dunkel, karmin, orange),
                             gruen (hell, dunkel: FINAL),
                             blau);
  FarbePlusPlus EXTENDS FarbePlus = (rot (FINAL),
                                     blau (hell, dunkel));
```

ergibt für FarbePlus die folgenden - mittels Konstanten beschriebenen - zulässigen Werte:

```
#rot.dunkel #rot.karmin #rot.orange #gelb #gruen.hell #gruen.dunkel #blau
```

für FarbePlusPlus zusätzlich:

```
#blau.hell #blau.dunkel
```

Durch die Angabe von FINAL bei den Grünstufen von FarbePlus ist es in FarbePlusPlus nicht zulässig weitere Grünstufen zu definieren. Mit der Angabe von FINAL für die Unterteilung von rot in FarbePlusPlus wird verhindert, dass in möglichen Erweiterungen von FarbePlusPlus noch weitere Rotvarianten angefügt werden Durch die Angabe von FINAL bei den Grünstufen von FarbePlus ist es in FarbePlusPlus nicht zulässig weitere Grünstufen zu definieren. Mit der Angabe von FINAL für die Unterteilung von rot in FarbePlusPlus wird verhindert, dass in möglichen Erweiterungen von FarbePlusPlus noch weitere Rotvarianten angefügt werden können.

2.8.3 Textausrichtungen

Für die Aufbereitung von Plänen und Karten müssen die Positionen von Texten festgehalten werden. Dabei muss festgelegt werden, welcher Stelle des Textes die Position entspricht. Mit dem horizontalen Alignment wird festgelegt, ob die Position auf dem linken oder rechten Rand des Textes oder in der Textmitte liegt. Das vertikale Alignment legt die Position in Richtung der Texthöhe fest.

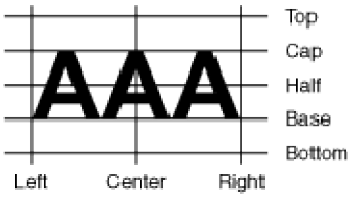
Der Abstand Cap-Base entspricht der Höhe der Grossbuchstaben. Unterlängen befinden sich im Bereich von Base-Bottom.

Horizontales und vertikales Alignment können als folgende vordefinierte Aufzählung verstanden werden:

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

Syntaxregel:

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



Figur 9: Textausrichtung horizontal (HALIGNMENT) und vertikal (VALIGNMENT).

2.8.4 Boolean

Der Typ Boolean weist die Werte false und true auf. Er kann als folgende vordefinierte Aufzählung verstanden werden:

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

Syntaxregel:

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

2.8.5 Numerische Datentypen

Die wichtigste Angabe bei numerischen Datentypen ist der Minimal- und der Maximal-Wert inklusive Stellenzahl (Nachkommastellen) sowie der Skalierungsfaktor. Zusätzlich kann angegeben werden, dass der Typ zirkulär ist (Schlüsselwort CIRCULAR), d.h. dass der in der letzten signifikanten Stelle um 1 erhöhte Maximalwert und der Minimalwert sachlich die gleiche Bedeutung haben (z.B. bei Winkeln 0 .. 359 Grad). Fehlt die Angabe des Minimal- und Maximal-Wertes (Schlüsselwort NUMERIC), gilt der Wertebereich als abstrakt.

```
DOMAIN
  Winkel1 = 0.00 .. 359.99 CIRCULAR [degree]; !! richtig
  Winkel2 = 0.00 .. 360.00 CIRCULAR [degree]; !! syntaktisch zwar richtig,
                                              !! sachlich aber falsch, da damit
                                              !! 360.01 dem Minimalwert 0.00
                                              !! entspricht.
```

Die Stellenzahl muss beim Minimal- und beim Maximal-Wert übereinstimmen. Mit Hilfe der Skalierung können Float-Zahlen beschrieben werden, aber dann sind sowohl der Minimal- als auch der Maximal-Wert in Mantissendarstellung anzugeben, d.h. beginnend mit Null (0) und gefolgt vom Dezimalpunkt (.) muss die erste Ziffer nach dem Dezimalpunkt von Null (0) verschieden sein. Die Skalierung des Minimalwertes muss kleiner sein als die Skalierung des Maximalwertes. Die Schreibweise von Minimal- und Maximalwert bedeutet aber keineswegs eine Anweisung, wie die Werte transferiert werden sollen (ist ein Wertebereich mit 000 .. 999 definiert, bedeutet das nicht, dass der Wert 7 als 007 transferiert wird). Eine

Ausnahme von dieser Regel bilden die Float-Zahlen. Diese sind in Mantissendarstellung und mit Skalierung zu transferieren.

Bei Erweiterungen dürfen die Maximal- bzw. Minimalwerte nur eingeschränkt werden. Der numerische Bereich wird damit also kleiner. Man beachte dabei folgende Situation:

```
DOMAIN
  Normal = 0.00 .. 7.99;
  Genau EXTENDS Normal = 0.0000 .. 7.9949;    !! richtig, da auch
                                              !! Normal darstellbar
  Genau EXTENDS Normal = 0.0000 .. 7.9999;    !! falsch, da gerundet
                                              !! ausserhalb Normal
```

Um die Bedeutung des Wertes genauer zu erklären kann eine Masseinheit angegeben werden (vgl. Kapitel 2.9 Einheiten). Abstrakte Masseinheiten sind nur zulässig, solange der Wertebereich selbst noch undefiniert ist (Schlüsselwort NUMERIC).

Für Erweiterungen gelten folgende Regeln:

- Weist ein konkreter Basis-Wertebereich keine Masseinheit auf, darf auch in Erweiterungen des Basis-Wertebereichs keine angegeben werden.
- Verwendet der Basis-Wertebereich eine abstrakte Masseinheit, dürfen in Erweiterungen des Basis-Wertebereichs nur Masseinheiten verwendet werden, die Erweiterungen der Masseinheit sind.
- Verwendet der Basis-Wertebereich eine konkrete Masseinheit, kann sie in Erweiterungen nicht übersteuert werden.

Beispiele:

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

DOMAIN
  Distanz (ABSTRACT) = NUMERIC [Length];
  MeterDist (ABSTRACT) EXTENDS Distanz = NUMERIC [m];
  FussDist (ABSTRACT) EXTENDS Distanz = NUMERIC [ft];
  KurzeMeter EXTENDS MeterDist = 0.00 .. 100.00 [m];
  KurzeFuesse EXTENDS FussDist = 0.00 .. 100.00 [ft];
  KurzeFuesse2 (ABSTRACT) EXTENDS KurzeMeter = NUMERIC [ft]; !! falsch: m vs. ft
```

Einem numerischen Wertebereich kann auch ein Skalarsystem zugeordnet werden (vgl. Kapitel 2.10.3 Referenzsysteme). Damit beziehen sich die Werte auf den durch das Skalarsystem bestimmten Nullpunkt. Es sind also *absolute* Werte in diesem Skalarsystem. Ist in der Klasse des Skalarsystems die Einheit nicht ANYUNIT, muss beim numerischen Datentyp eine Einheit angegeben werden, die mit jener des Referenzsystems verträglich ist. Bezieht man sich auf ein Koordinatensystem, kann die Achse angegeben werden, auf die sich die Werte beziehen. Die Einheit muss mit jener der entsprechenden Achse verträglich sein. Fehlt diese Angabe, ist der Bezug nicht genauer definiert, sondern ergibt sich aus dem Fachgebiet (z.B. bezieht man sich bei einer Höhe auf ein Ellipsoid, meint man ellipsoidische Höhen). Bezieht man sich auf einen anderen Wertebereich, soll das gleiche Referenzsystem gelten wie bei diesem Wertebereich. In diesem Fall darf die Angabe der Achse nur fehlen, wenn es sich um einen numerischen Wertebereich handelt. Bei einem Koordinatenwertebereich ist die Achsenangabe obligatorisch. Die Angabe des Referenzsystems kann in Erweiterungen nicht mehr geändert werden.

Stellt der numerische Wert einen Winkel dar, kann sein Richtungssinn festgelegt werden. Im Falle von Richtungen kann angegeben werden, auf welches Koordinatensystem (definiert durch einen Koordinaten-Wertebereich) sich die Richtung bezieht. Damit ist bekannt, wie die Nullrichtung (Azimut) und der Drehsinn definiert sind (vgl. Kapitel 2.8.7 Koordinaten). Diese Angabe kann in Erweiterungen nicht mehr geändert werden.

Als numerische Konstanten sind nebst den Dezimalzahlen auch die Zahlen Pi (Schlüsselwort PI) und e – Basis des natürlichen Logarithmus - (Schlüsselwort LNBASE) definiert.

Syntaxregeln:

```
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 Strukturierte Wertebereiche

Strukturierte Wertebereiche bauen auf strukturierten Einheiten (vgl. Kapitel 2.9 Einheiten) auf. So können z.B. Datum, Zeit behandelt werden. Es müssen Minimal- und Maximalwert und die strukturierte Einheit angegeben werden. Zudem kann ein Bezug zu einem numerischen Referenzsystem (z.B. Zeitzone) gemacht werden. Die direkt angegebene Einheit muss mit der beim Referenzsystem definierten verträglich sein, d.h. mit ihr übereinstimmen oder sie erweitern oder aus ihr oder einer Erweiterung abgeleitet sein.

Für die Vererbung gelten die gleichen Regeln wie für numerische Datentypen. Beispiel:

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

Syntaxregeln:

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

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

2.8.7 Koordinaten

Koordinaten können ein-, zwei- oder dreidimensional definiert werden und sind entsprechend eine Einzelzahl, ein Zahlenpaar oder ein Zahlentripel. Es ist zulässig, dass die zweite oder dritte Dimension erst in einer Erweiterung beigefügt wird. Für jede Dimension muss der numerische Wertebereich sowie allenfalls eine Masseinheit und ein Koordinatensystem (inkl. Achsnummern) angegeben werden. Es gelten die gleichen Regeln wie bei den numerischen Datentypen. Es können nur konkrete Masseinheiten angegeben werden. Wird kein Referenzsystem angegeben und sind die Masseinheiten entweder nicht oder als Längeneinheit definiert, darf ein Programmsystem, das das Modell implementiert, davon ausgehen, dass es sich um kartesische Koordinaten handelt.

Wird eine Rotationsangabe gemacht (Schlüsselwort ROTATION) kann im Rahmen von Richtungsdefinitionen (vgl. Kapitel 2.8.5 Numerische Datentypen) auf ein solches Koordinaten-Referenzsystem verwiesen werden. Die Rotationsdefinition legt fest, welche Achse der Nullrichtung und welche der Richtung eines positiven, rechten Winkels entsprechen. Sie darf auch in einer konkreten Koordinatendefinition fehlen und dann allenfalls in einer Erweiterung beigefügt werden.

Die Angaben betreffend Achsbezug und Rotation können in Erweiterungen nicht geändert werden. Beispiel:

```
DOMAIN
```

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

Bei beiden definierten Dimensionen wird nebst dem zulässigen Bereich angegeben, auf welche Einheiten und welches Referenzsystem samt Achsennummer sich die Koordinaten beziehen. Die eigentlichen Achsen sind beim Referenzsystem definiert. Die Rotationsangabe legt fest, dass die Nullrichtung von der zweiten zur ersten Achse führt, beim Schweizerischen System, wo der erste Wert der Ostwert, der zweite der Nordwert ist, zeigt die Nullrichtung nach Norden und dreht im Uhrzeigersinn.

```
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]};
```

Geografische Koordinaten sind typischerweise in Grad dargestellt und beziehen sich auf ein Ellipsoid-System (z.B. WGS84). Die Höhe andererseits ist in Meter beschrieben. Sie bezieht sich auf ein spezielles Ellipsoid-Höhen-System mit einer Achse.

Syntaxregeln:

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

NumericalType = ( NumericType | StructuredUnitType ).

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

Falls nicht mindestens ein numerischer Bereich definiert ist (mit NumericalType), ist das entsprechende Attribut als abstrakt zu deklarieren.

2.8.8 Wertebereiche von Objektidentifikationen

Identifizierbare Objekte werden immer mit einer Objektidentifikation versehen. Damit für die Systeme klar ist, welcher Speicherplatz dafür vorgesehen werden muss, können entsprechende Wertebereiche definiert und diese den Themen (vgl. Kapitel 2.5.2 Themen) zugeordnet werden.

Syntaxregel:

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

INTERLIS 2 selbst definiert die folgenden OID-Wertebereiche (vgl. Anhang A):

```
DOMAIN
    ANYOID (ABSTRACT) = OID ANY;
    I32OID = OID 0 .. 2147483647; !! positive in 4 Bytes speicherbare Integerwerte
    STANDARDOID = OID TEXT*16;    !! gemäss Anhang E
```

Jeder numerische oder textliche OID-Wertebereich gilt als direkte oder indirekte Erweiterung des abstrakten OID-Wertebereichs ANYOID.

2.8.9 Behälter

Soll eine Behälterverwaltung modelliert werden, ist es nötig, die Behälter modellieren zu können. Dies erfolgt mit dem Schlüsselwort BASKET. Ohne weitere Angaben, können in einem solchen Attribut, beliebige Behälter beschrieben werden. Die zulässigen Behälter können aber eingeschränkt werden. Einerseits kann mittels Properties angegeben werden, welche Arten von Behältern (vgl. Kapitel 2.5.2 Themen) zulässig sind:

- DATA: Datenbehälter. In den Behältern kommen nur Instanzen von Klassen (und ihren Unterstrukturen) vor.
- VIEW: Sichtbehälter. In den Behältern kommen die (virtuellen) Instanzen von Sichten eines Sichtthemas (Schlüsselwort VIEW TOPIC) vor.
- BASE: Behälter mit Instanzen von Klassen und Sichten, die durch die Grafikdefinitionen gebraucht werden.
- GRAPHIC: Behälter mit Grafiksignaturen. Ein solcher Behälter wird auch Signaturenbibliothek genannt.

In Erweiterungen können die Behälterarten nur noch eingeschränkt, nicht aber ergänzt werden.

Andererseits können die zulässigen Behälter durch die Angabe des zulässigen Themas eingeschränkt werden.

Syntaxregel:

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

Als Datenfelder eines Behälters werden die Bezeichnung des Themas (Modell-Name.Thema-Name), die Art des Behälters (DATA, VIEW, BASE, GRAPHIC) und der Identifikator des Behälters geführt. Ein Behältertyp kann darum auch als Struktur mit folgender Definition aufgefasst werden:

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

2.8.10 Klassentypen

Es kann Sinn machen, dass Datenobjekte Verweise auf bestimmte Klassen enthalten.

Syntaxregel:

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

Mit der Angabe von STRUCTURE wird eine beliebige Struktur oder Klasse, mit CLASS (auch als Erweiterung von STRUCTURE zulässig) eine beliebige Klasse (aber keine Strukturen) zugelassen. Sollen nur bestimmte Strukturen bzw. Klassen und ihre Erweiterungen zugelassen sein, sind diese aufzuführen (RESTRICTED TO). In Erweiterungen müssen erneut alle zulässigen Strukturen bzw. Klassen aufgeführt werden. Sie dürfen aber nicht im Widerspruch zur Basisdefinition sein. Sobald solche Einschränkungen definiert sind, kann darum STRUCTURE nicht mehr durch CLASS erweitert werden.

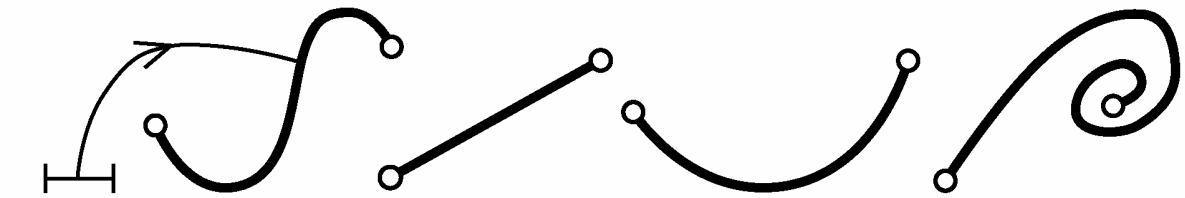
2.8.11 Linienzüge

2.8.11.1 Geometrie des Linienzugs

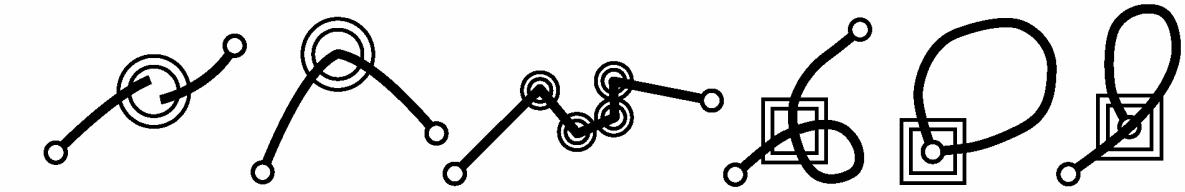
Anschaulich ist ein Kurvenstück ein 1-dimensionales Gebilde, das keine Risse, keine Ecken und keine Doppelpunkte jeglicher Art hat (siehe Figuren 10 und 11). Kurvenstücke sind also glatt und eindeutig. Strecken, Kreisbogen, Parabel- und Klothoidenstücke sind Beispiele von Kurvenstücken. Jedes Kurven-

stück hat zwei *Randpunkte* (Anfangs- und Endpunkt), die nicht zusammenfallen dürfen. Die übrigen Punkte des Kurvenstückes heissen *innere Punkte*. Diese bilden das *Innere* des Kurvenstückes.

> Exakte Definition (mathematische Begriffe, die nicht weiter erklärt werden, deren Definition man aber in Lehrbüchern findet, werden "*kursiv und in Anführungszeichen"* geschrieben): *Kurvenstück* heisst eine Teilmenge des *"3-dimensionalen" "Euklidischen Raumes"* (im folgenden kurz *Raum* genannt), die *"Bildmenge"* einer *"glatten"* und *"injektiven" "Abbildung"* eines *"Intervalls"* (der *"Zahlengerade"*) ist. Anfangs- und Endpunkt des Kurvenstücks sind die Bilder der Intervallenden. *Ebenes Kurvenstück* heisst ein Kurvenstück, das in einer *Ebene* (*"2-dimensionaler" "Unterraum"* des Raumes) liegt.



Figur 10: Beispiele von ebenen Kurvenstücken.

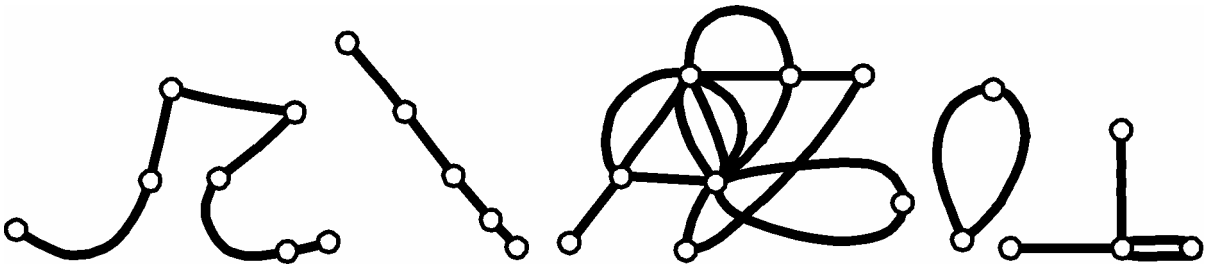


Figur 11: Beispiele von ebenen Mengen, die nicht Kurvenstücke sind (ein doppelter Kreis bedeutet "nicht glatt" und ein doppeltes Rechteck "nicht injektiv").

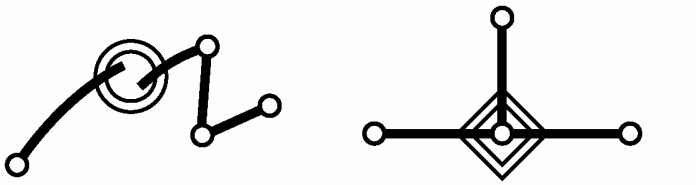
Ein Linienzug ist eine endliche Folge von Kurvenstücken. Ausser beim ersten Kurvenstück stimmt der Anfangspunkt jeweils mit dem Endpunkt des Vorgänger-Kurvenstückes überein. Diese Punkte heissen *Stützpunkte* des Linienzuges. Anschaulich kann ein Linienzug mehrfach benützte Kurvenstücke, Kurvenstücke mit zusammenfallenden Stützpunkten, sich schneidende Kurvenstücke und im Innern von Kurvenstücken endende oder startende Kurvenstücke enthalten (siehe Figuren 12 und 13). Ein *einfacher Linienzug* weist keinerlei Selbst-Schnittpunkte auf (siehe Figur 14). Bei einem *einfach geschlossenen Linienzug* stimmt zudem der Anfangspunkt des ersten Kurvenstücks mit dem Endpunkt des letzten überein.

> Exakte Definition (mathematische Begriffe, die nicht weiter erklärt werden, deren Definition man aber in Lehrbüchern findet, werden "*kursiv und in Anführungszeichen"* geschrieben): *Linienzug* heisst eine Teilmenge des Raumes, die *"Bildmenge"* einer "*stetigen"* und *"stückweise glatten"* (aber nicht notwendigerweise *"injektiven"*) *"Abbildung"* eines *"Intervalls"* ist (der so genannten *zugeordneten Abbildung*) und nur endlich viele *"nicht glatte Stellen"* aufweist. Eine *"nicht glatte Stelle"* heisst *Ecke.* Bei einem *geschlossenen Linienzug* stimmen Anfangs- und Endpunkt überein. *Einfacher Linienzug* heisst ein Linienzug, dessen zugeordnete Abbildung auch *"injektiv"* ist. *Einfach geschlossener Linienzug* heisst

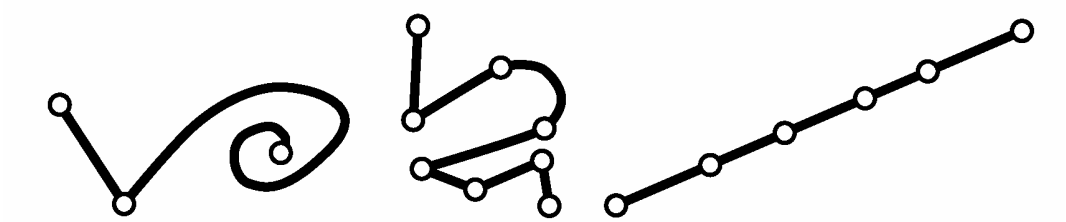
ein Linienzug, dessen zugeordnete Abbildung auch *"injektiv"* ist, abgesehen von seinem Anfangs- und Endpunkt, die übereinstimmen.



Figur 12: Beispiele von (ebenen) Linienzügen.



Figur 13: Beispiele von ebenen Mengen, die nicht Linienzüge sind (ein doppelter Kreis bedeutet hier "nicht stetig" und der Rhombus "nicht Bild eines Intervalls").



Figur 14: Beispiele von (ebenen) einfachen Linienzügen.

2.8.11.2 Linienzug mit Strecken und Kreisbogen als vordefinierte Kurvenstücke

INTERLIS 2 kennt gerichtete (DIRECTED POLYLINE) oder ungerichtete (POLYLINE) Linienzüge. Zudem werden Linienzüge im Rahmen von Einzelflächen und Gebietseinteilungen (vgl. Kapitel 2.8.12 Einzelflächen und Gebietseinteilungen) verwendet.

Zur Definition eines konkreten Linienzug-Wertebereichs gehört immer die Angabe der erlaubten Kurvenstück-Formen mittels Aufzählung, z.B. Strecken (Schlüsselwort STRAIGHTS), Kreisbogen (Schlüsselwort ARCS) oder weitere Möglichkeiten (siehe Kapitel 2.8.11.3 Weitere Kurvenstück-Formen), und die Angabe des Wertebereichs der Stützpunkte. In einem abstrakten Linienzug-Wertebereich dürfen diese Angaben fehlen. Für Wertebereichserweiterungen gelten folgende Regeln:

- Die Kurvenstück-Form darf nur reduziert, nicht aber ergänzt werden.
- Der Koordinaten-Wertebereich, der im Rahmen einer Erweiterung eines Linienzug-Wertebereichs angegeben wird, muss eine Einschränkung des Koordinaten-Wertebereichs des Basis-Linienzug-Wertebereichs sein, sofern ein solcher definiert ist.

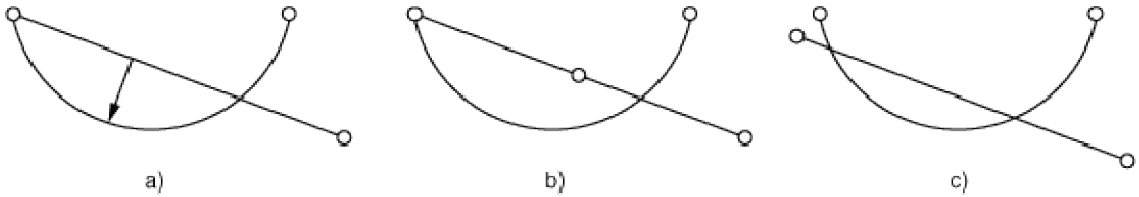
Die Kurvenstücke werden immer als Erweiterung der Grundstruktur LineSegment aufgefasst. Der darin verwendete Koordinatenwertebereich ist der in der Liniendefinition angegebene.

```
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;
```

Das erste Kurvenstück eines Linienzuges ist immer ein StartSegment. Das Startsegment besteht nur aus dem Startpunkt selbst, der zugleich auch Endpunkt des Startsegments ist. Das Geradenstück hat einen Endpunkt und definiert dadurch eine Strecke vom Endpunkt des vorherigen Kurvenstücks zu seinem Endpunkt. Startsegment und Geradenstücke brauchen keine weiteren Angaben. Die entsprechenden Erweiterungen von LineSegment sind darum leer. Zwei aufeinander folgende Stützpunkte (SegmentEndPoints) dürfen in der Projektion nicht aufeinander fallen.

Ein Kreisbogenstück beschreibt ein Kurvenstück, das in der Projektion als echtes Kreisbogenstück erscheint. Ein Kreisbogenstück wird zusätzlich zum Endpunkt mit einem Zwischenpunkt beschrieben. Dieser ist nur in der Lage von Bedeutung. Bei dreidimensionalen Koordinaten wird die Höhe auf dem Kreisbogenstück linear interpoliert. Man kann sich die Kurve als Gewindestück einer zylindrischen Schraube vorstellen, die senkrecht auf der Projektionsfläche steht. Der Zwischenpunkt ist kein Stützpunkt des Linienzuges. Er soll möglichst exakt in der Mitte zwischen Anfangs- und Endpunkt liegen. Da der Zwischenpunkt in der gleichen Genauigkeit angegeben wird wie die Stützpunkte, kann der berechnete Radius erheblich vom effektiven Radius abweichen. Wird der effektive Radius angegeben, ist er für die Kreisbogendefinition massgebend. Der Zwischenpunkt legt nur noch fest, welcher der vier möglichen Kreisbogen der gewünschte ist. Der Zwischenpunkt darf aber auch in diesem Fall um höchstens 2 Einheiten von der Spur des aus dem Radius gerechneten Kreisbogens abweichen.

Es kann verlangt werden, dass es sich bei einem Linienzug um einen einfachen Linienzug handelt, d.h. anschaulich, dass er sich nicht mit sich selbst schneiden darf und insbesondere mehrfache Benützung desselben Kurvenstücks ausgeschlossen ist. (Schlüsselwort WITHOUT OVERLAPS). Wenn ein Kreisbogen und eine Strecke (bzw. ein anderer Kreisbogen) als aufeinander folgende Kurvenstücke eines Linienzuges neben dem gemeinsamen Stützpunkt auch noch einen inneren Punkt (Definition siehe oben) gemeinsam haben, so ist das auch bei einem einfachen Linienzug erlaubt, falls das von der Strecke abgeschnittene Kreissegment (bzw. das vom anderen Kreisbogen abgeschnittene Doppel-Kreissegment) eine Pfeilhöhe aufweist, die kleiner oder gleich ist wie die nach WITHOUT OVERLAPS > angegebene Dezimalzahl (siehe Figur 15a). Diese Regelung erfolgt aus zwei Gründen: Einerseits sind bei Kreisbogen kleine Überschneidungen aus numerischen Gründen in gewissen Fällen nicht vermeidbar (z.B. bei tangentialen Kreisbogen). Andererseits sind bei der Übernahme von Daten, die ursprünglich grafisch erfasst wurden, auch grössere Überlappungen (z.B. einige Zentimeter) zu tolerieren, will man nicht einen enormen Nachbearbeitungsaufwand in Kauf nehmen. Die Angabe der Toleranz muss in den gleichen Einheiten, wie die der Stützpunktkoordinaten erfolgen. Sie darf aus numerischen Gründen nicht kleiner als 2 Einheiten der letzten signifikanten Dezimalstelle sein. Sie kann nicht übersteuert werden und ist bei Einzelflächen und Gebietseinteilungen obligatorisch.



Figur 15: a) Die Pfeilhöhe darf nicht grösser als die angegebene Toleranz sein; b), c) unzulässige Überschneidungen eines Linienzuges, da Strecke und Kreisbogen, die sich treffen, nicht von einem gemeinsamem Stützpunkt ausgehen.

Im Rahmen von Wertebereichsdefinitionen und Attributserweiterungen können ungerichtete Linienzüge zu gerichteten Linienzügen erweitert werden (vgl. ferner Kapitel 2.8.12.4 Erweiterbarkeit).

Sind Linienzüge gerichtet, muss ihr Richtungssinn immer (auch bei einem Datentransfer) erhalten bleiben.

Für die Stützpunkte wird der Wertebereich der Koordinaten definiert. Mittels der Existenzbedingung REQUIRED IN (vgl. Kapitel 2.12 Konsistenzbedingungen und Kapitel 2.13 Ausdrücke) kann zudem gefordert werden, dass die Koordinaten nicht beliebig sein dürfen, sondern denjenigen der Punkte bestimmter Klassen entsprechen müssen.

Ist der Koordinatentyp der Stützpunkte abstrakt, muss der Linienzug seinerseits als abstrakt deklariert werden.

Syntaxregeln:

```
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.
```

Damit bei Flächen (Kapitel 2.8.12 Einzelflächen und Gebietseinteilungen) verschiedenen Abschnitten der Umrandung unterschiedliche Attribute zugeordnet werden können, ist es möglich, für die eigentlichen Linienzugsobjekte noch weitere Attribute zu definieren (so genannte Linienattribute, Regel LineAttrDef nur bei SURFACE und AREA erlaubt). Für die Definition wird eine Struktur angegeben, die aber nur lokale Attribute und Funktionsaufrufe aufweisen darf. Wird ein Einzelflächen- oder Gebietseinteilungs-Attribut, für das ein Linienattribut mittels Struktur definiert ist, erweitert, dann darf das erweiterte Attribut entweder kein Linienattribut aufweisen oder dessen Struktur muss eine Erweiterung der Basisstruktur sein.

Syntaxregel:

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

2.8.11.3 Weitere Kurvenstück-Formen

Nebst Geradenstücken und Kreisbogen sind weitere Kurvenstück-Formen definierbar. Nebst dem Namen muss angegeben werden, gemäss welcher Struktur ein Kurvenstück beschrieben wird. Diese Definitionen

von weiteren Kurvenstück-Formen und entsprechenden Strukturen sind nur im Rahmen eines Kontraktes zulässig, da nicht angenommen werden kann, dass ein System irgendeine Kurvenform unterstützt.

Syntaxregel:

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

Eine Linienstruktur muss immer eine Erweiterung der durch INTERLIS definierten Struktur LineSegment sein (vgl. Kapitel 2.8.11.2 Linienzug mit Strecken und Kreisbogen als vordefinierte Kurvenstücke).

2.8.12 Einzelflächen und Gebietseinteilungen

2.8.12.1 Geometrie von Flächen

Für die Modellierung von Geodaten genügen meist ebene Flächen. INTERLIS unterstützt darüber hinaus ebene allgemeine Flächen. Anschaulich ist eine ebene allgemeine Fläche durch eine äussere und allenfalls eine oder mehrere innere Randlinien begrenzt (Figur 20). Die Randlinien selbst müssen aus einfachen Linienzügen bestehen, die aus geometrischer Sicht jeweils zu einfach geschlossenen Linienzügen zusammengefasst werden können. Sie müssen zudem so angeordnet sein, dass es von einem beliebigen Punkt im Innern der Fläche immer einen Weg zu einem beliebigen anderen Punkt im Innern der Fläche gibt, der weder eine Randlinie schneidet noch einen Stützpunkt einer Randlinie enthält (vgl. Figur 19). Soweit diese Bedingung nicht verletzt wird, dürfen sich Ränder in Stützpunkten berühren. In solchen Situationen kann man sich verschiedene Möglichkeiten vorstellen, wie die Umrandung der Fläche als Ganzes in einzelne Linienzüge aufgeteilt wird (vgl. Figur 22). INTERLIS macht keine Vorschriften, welche Möglichkeit gewählt wird. Wird eine solche Fläche mehrmals transferiert, dürfen in den verschiedenen Übertragungen auch unterschiedliche Aufteilungen vorkommen.

Exakte Definitionen (mathematische Begriffe, die nicht weiter erklärt werden, deren Definition man aber in Lehrbüchern findet, werden *"kursiv und in Anführungszeichen"* geschrieben):

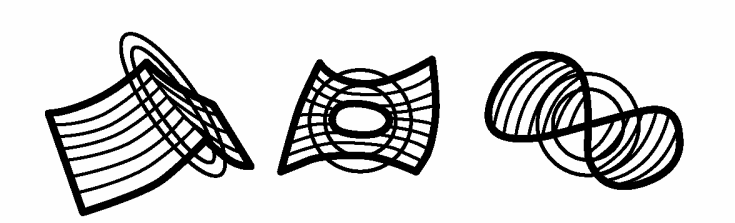
Flächenelement heisst eine Teilmenge des *Raumes,* die *"Bildmenge"* einer *"glatten"* und *"injektiven" "Abbildung"* eines ebenen regulären Vielecks ist (siehe Figuren 16 und 17).

Fläche heisst die Vereinigung *F* von endlich vielen Flächenelementen, die *"zusammenhängend"* ist und folgender Bedingung genügt: Zu jedem Punkt *P* der Fläche gibt es eine *"Umgebung"*, die sich in ein ebenes reguläres Vieleck *deformieren* (d.h. *"homöomorph abbilden"*) lässt. Wenn bei einer solchen Deformation der Punkt *P* in den Rand des Vielecks übergeführt wird, heisst er *Randpunkt von F*, andernfalls *innerer Punkt von F*. Es gilt: Der *"Rand"* (d.h. die Menge aller Randpunkte) einer Fläche ist die Vereinigung von endlich vielen Kurvenstücken, die nur Endpunkte gemeinsam haben. Eine *ebene Fläche* ist eine Fläche, die Teilmenge einer *Ebene* ist. Es gilt: Der *"Rand"* einer ebenen Fläche besteht aus einem äusseren *einfach geschlossenen Linienzug* (genannt der *äussere Rand*) und aus endlich vielen (evtl. auch keinen) inneren *einfach geschlossenen Linienzügen* (den so genannten *inneren Rändern*). Der äussere und alle inneren Ränder haben keine Punkte gemeinsam. Ein durch einen inneren Rand ausgespartes Flächenstück heisst *Enklave* (siehe Figuren 18, 19 und 20).

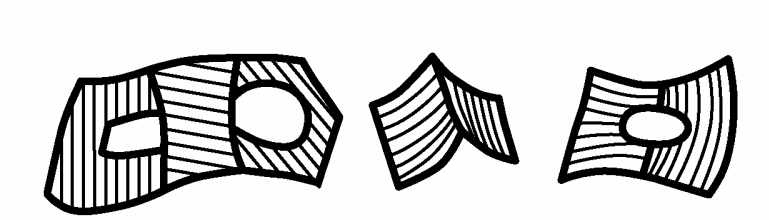
Eine *allgemeine Fläche* ist eine Fläche mit zusätzlich endlich vielen *singulären Punkten* aber mit *"zusammenhängendem" Inneren* (Menge der inneren Punkte). Ein *singulärer Punkt* kann zusammen mit einer *"Umgebung"* in eine ebene *Propellermenge* deformiert werden, er selbst ins *Zentrum*. *Propellermenge* heisst die Vereinigung endlich vieler Drei-

ecksflächen, die genau einen Punkt gemeinsam haben, das *Zentrum.* *Ebene allgemeine Fläche* heisst eine allgemeine Fläche, die Teilmenge einer *Ebene* ist (siehe Figur 21). Es gilt: Der Rand einer ebenen allgemeinen Fläche kann auf verschiedene Art zusammengesetzt werden aus endlich vielen geschlossenen Linienzügen, die höchstens endlich viele Punkte gemeinsam haben und je höchstens endlich viele Doppelpunkte aufweisen (siehe Figur 22).

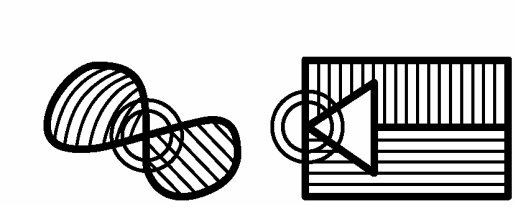
Figur 16: Beispiele von Flächenelementen.



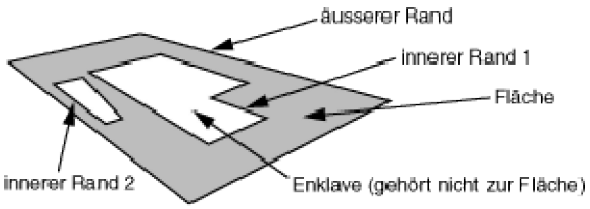
Figur 17: Beispiele von Punktmengen im Raum, die nicht Flächenelemente sind (ein doppelter Kreis bedeutet hier "nicht glatt").



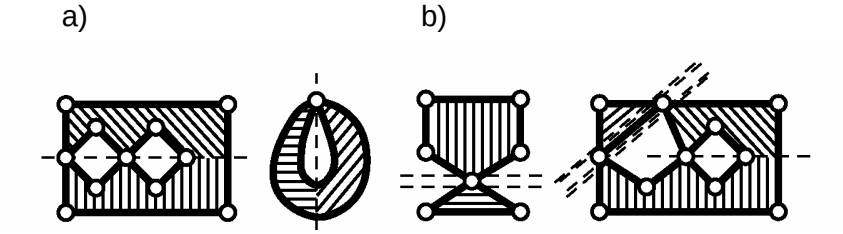
Figur 18: Beispiele von Flächen im Raum.



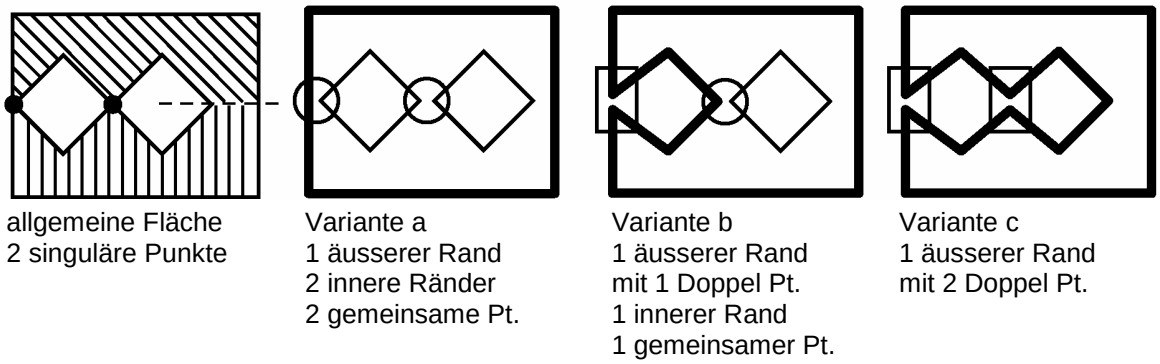
Figur 19: Beispiele ebener Punktmengen, die nicht Flächen sind (ein doppelter Kreis bedeutet "singulärer Punkt").



Figur 20: Ebene Fläche mit Rändern und Enklaven.

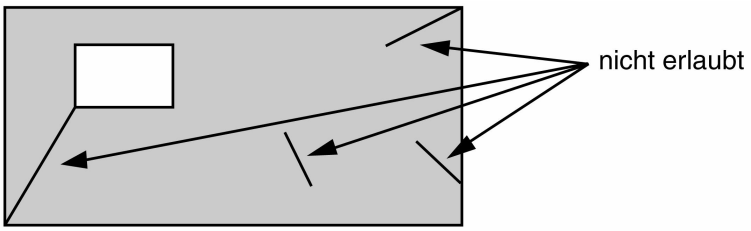


Figur 21: a) Beispiele von allgemeinen ebenen Flächen; b) Beispiele von ebenen Mengen, die nicht allgemeine Flächen sind, weil ihr Inneres nicht zusammenhängend ist. Diese ebenen Mengen können aber in allgemeine Flächen unterteilt werden ("---" zeigt die Unterteilung in Flächenelemente und "===" die Unterteilung in allgemeine Flächen).



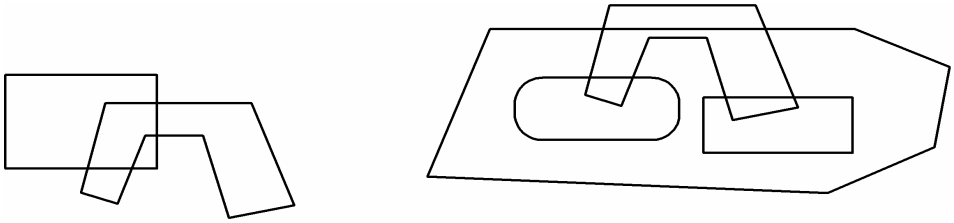
Figur 22: Verschiedene mögliche Aufteilungen des Randes einer allgemeinen Fläche.

Mit der Definition von (allgemeinen) Einzelflächen bzw. (allgemeinen) Flächen einer Gebietseinteilung wird auch festgelegt, oberhalb welcher Toleranz sich die Kurvenstücke des Randes nicht überlappen dürfen (WITHOUT OVERLAPS muss für konkrete Definitionen von Einzelflächen oder Gebietseinteilungen entweder direkt angegeben oder geerbt werden). Das Überlappungs- bzw. Schnittverbot gilt bei Einzelflächen nicht nur zwischen den Kurvenstücken eines einzelnen Linienzuges sondern zwischen allen Kurvenstücken aller Linienzüge des Flächenrandes. Für Flächen einer Gebietseinteilung gilt es sogar für alle an der Gebietseinteilung beteiligten Linienzüge. Zudem sind Linienzüge, die nicht zum Rand einer (allgemeinen) Fläche gehören, gemäss Definition der (allgemeinen) Fläche ausgeschlossen.



Figur 23: Nicht erlaubte Linien von Flächen.

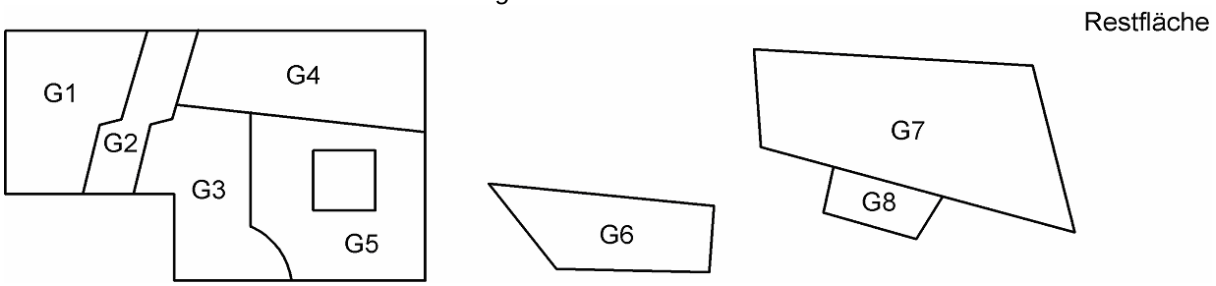
2.8.12.2 Einzelflächen



Figur 24: Einzelflächen (SURFACE).

Für (allgemeine) Flächen, die sich ganz oder teilweise überlappen dürfen, d.h. die nicht nur Randpunkte gemeinsam haben dürfen, steht der geometrische Attributtyp SURFACE zur Verfügung (siehe Figur 24). Dieser Typ wird Einzelflächen genannt. Eine Einzelfläche hat einen äusseren und allenfalls mehreren inneren Ränder (um Enklaven). Jeder Rand besteht aus mindestens einem Linienzug. Jeder Linienzug weist nebst der Geometrie die definierten Attribute auf (vgl. Regel LineAttrDef).

2.8.12.3 Flächen einer Gebietseinteilung



Figur 25: Gebietseinteilung (AREA).

Gebietseinteilung (*Flächennetz*) heisst eine endliche Menge von (allgemeinen) Flächen und Restflächen, welche die Ebene überlappungsfrei überdecken.

Für Gebietseinteilungen steht der geometrische Attributtyp AREA zur Verfügung.

Jedem Gebietsobjekt ist höchstens eine Fläche der Gebietseinteilung (oder genau eine mit Zusatz-Schlüsselwort MANDATORY), nicht aber die Restfläche, zugeordnet. Es ist nicht zulässig, dass zwei Flächen der Gebietseinteilung mit einem gemeinsamen Rand je *keinem* Gebietsobjekt entsprechen.

Jedes einzelne Gebietsobjekt entspricht damit einer Einzelfläche. Für Gebietsobjekte ergibt sich damit auch die gleiche implizite Struktur wie für Einzelflächen. Zusätzlich gelten aber Konsistenzbedingungen:

- Linienzüge einer Gebietseinteilung müssen immer echte Grenzlinien sein. Es dürfen also keine Linienzüge existieren, bei denen auf beiden Seiten die gleiche Fläche liegt (Figur 23). Dies ist auch durch die Definition der Fläche ausgeschlossen.
- Liegen auf beiden Seiten eines Linienzuges definierte Gebietsobjekte, muss jedes Kurvenstück

(Verbindung zweier Stützpunkte) des einen Gebietsobjektes in Geometrie und Attributen genau einem Kurvenstück des anderen Gebietsobjektes entsprechen.

Gebietseinteilungen dürfen nicht innerhalb von Unterstrukturen vorkommen.

Damit die Linienzüge einer Gebietseinteilung auch als einzelne Objekte angesprochen werden können (und zwar als ein Objekt, auch wenn der Linienzug zwei Gebietsobjekte begrenzt), steht die AREA INSPECTION zur Verfügung (vgl. Kapitel 2.15 Sichten).

2.8.12.4 Erweiterbarkeit

Einzelflächen können zu Gebietseinteilungen erweitert werden. Die Erweiterung eines Linienzuges zu einer Fläche ist unzulässig, da bei einer Fläche mit mehreren Linienzügen gerechnet werden muss, während mit der Definition eines Linienzuges nur einer erwartet wird.

Unabhängige Flächen und Flächen einer Gebietseinteilung können in zweierlei Hinsicht erweitert werden:

- Ist primär 'SURFACE' definiert, sind also Überlappungen zugelassen, darf dies in Erweiterungen durch 'AREA' ersetzt werden, da damit die Grunddefinition nicht verletzt wird.
- Es können weitere Linienattribute beigefügt werden.

2.9 Einheiten

Einheiten werden immer durch eine Bezeichnung und (in [ ]-Klammern) ein Kurzzeichen beschrieben. Bezeichnung und Kurzzeichen müssen Namen sein. Fehlt die Kurzzeichen-Angabe, ist sie gleich der Bezeichnung. Je nach Art der Einheit können weitere Angaben folgen. Der Gebrauch einer Einheit erfolgt immer über das Kurzzeichen. Die Bezeichnung hat damit nur dokumentarischen Charakter.

2.9.1 Basiseinheiten

Basis-Einheiten sind Meter, Sekunden, etc. Sie brauchen keine weiteren Angaben mehr. Basis-Einheiten können aber auch abstrakt (Schlüsselwort ABSTRACT) definiert werden, wenn die Einheit selbst noch nicht bekannt ist, wohl aber der zu bezeichnende Gegenstand (z.B. Temperatur, Geld). Abstrakten Einheiten ist noch kein Kurzzeichen zugeordnet. Konkrete Einheiten können nicht mehr erweitert werden.

Beispiele:

```
UNIT
  Laenge (ABSTRACT);
  Zeit (ABSTRACT);
  Geld (ABSTRACT);
  Temperatur (ABSTRACT);
  Meter [m] EXTENDS Laenge;
  Sekunde [s] EXTENDS Zeit;
  SchweizerFranken [CHF] EXTENDS Geld;
  Celsius [C] EXTENDS Temperatur;
```

Durch INTERLIS selbst ist die abstrakte Einheit ANYUNIT definiert. Alle anderen Einheiten erben sie direkt oder indirekt (vgl. Kapitel 2.10.3 Referenzsysteme), ohne dass dies definiert werden muss. Folgende Einheiten sind durch INTERLIS direkt (d.h. intern) definiert:

```
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;
```

Hinweis: Im Anhang G Einheiten-Definitionen sind die gebräuchlichsten Einheiten in einem erweiterten Typenmodell zusammengefasst.

2.9.2 Abgeleitete Einheiten

Abgeleitete Einheiten sind durch Multiplikationen bzw. Divisionen mit Konstanten oder Funktion in eine andere Einheit umrechenbar. Beispiele:

```
UNIT
  Kilometer [km] = 1000 [m];
  Centimeter [cm] = 1 / 100 [m];
  Inch [in] = 0.0254 [m];         !! 1 Zoll ist 2.54 cm
  Fahrenheit [oF] = FUNCTION // (oF + 459.67) / 1.8 // [K];
```

Werte in Kilometer müssen mit Tausend multipliziert werden, um Werte in Meter zu werden. Werte in Inch müssen mit 2.54 multipliziert werden um Werte in Zentimeter zu werden. Werte in Grad Fahrenheit müssen um 459.67 erhöht und durch 1.8 dividiert werden um zu Kelvin-Werten zu werden.

Eine abgeleitete Einheit ist automatisch eine Erweiterung der gleichen abstrakten Einheit, wie diejenige in die sie umgerechnet werden kann.

2.9.3 Zusammengesetzte Einheiten

Zusammengesetzte Einheiten (z.B. km pro h) ergeben sich durch Multiplikationen oder Divisionen von anderen Einheiten (Basis-Einheit, abgeleitete oder zusammengesetzte Einheit). Zusammengesetzte Einheiten können auch als abstrakte Einheiten definiert werden. Sie müssen sich dann vollständig auf andere abstrakte Einheiten beziehen.

Die bei der konkreten Erweiterung verwendeten Einheiten müssen dann konkrete Erweiterungen der in der abstrakten Definition verwendeten Einheiten sein. Beispiel:

```
UNIT
  Geschwindigkeit (ABSTRACT) = ( Laenge / Zeit );
  Stundenkilometer [kmph] EXTENDS Geschwindigkeit = ( km / h );
```

2.9.4 Strukturierte Einheiten

Strukturierte Einheiten setzen sich aus mehreren anderen, konkreten Einheiten zusammen, die direkt oder indirekt (abgeleitete Einheit) auf einer gemeinsamen Basis-Einheit (z.B. Zeit) aufbauen müssen. Bei den niederwertigen Anteilen der strukturierten Einheit muss immer angegeben werden, in welchem Wertebereich die Angabe erfolgt. Die jeweiligen Minima und Maxima müssen immer positive Zahlen sein. Beim letzten Anteil dürfen Dezimalstellen angegeben werden, bei den andern nicht. Skalierungsangaben sind nicht erlaubt.

Es gibt strukturierte Einheiten, die Unstetigkeiten enthalten: So besitzt nicht jeder Monat 31 Tage. Andere Einheiten (wie die Uhrzeit) sind über das gesamte Kontinuum definiert, was mit dem Schlüsselwort CONTINUOUS angezeigt wird. Beispiele:

```
UNIT
  Zeit (ABSTRACT);
  Sekunde [s] EXTENDS Zeit;
  Minute = 60 [s];
  Stunde = 60 [Minute];
  Tag EXTENDS Zeit;
  Monat EXTENDS Zeit;
  Jahr EXTENDS Zeit;
  Uhrzeit = {Stunde:Minute[0 .. 59]:s[0 .. 59]} CONTINUOUS;
  Datum = {Jahr:Monat[1 .. 12]:Tag[1 .. 31]};
```

Syntaxregeln:

```
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.
```

Hinweis: Im Anhang H Zeit-Definitionen findet sich ein Standard-Erweiterungsvorschlag mit einem Anwendungsbeispiel für Schweizer Zeitangaben.

2.10 Umgang mit Metaobjekten

2.10.1 Allgemeines

Metaobjekte im Sinne von INTERLIS 2 sind Objekte, die im Rahmen der Beschreibung von Anwendungsmodellen von Bedeutung sind. Dies wird für Referenzsysteme und Grafiksignaturen genutzt. Referenzsysteme oder Grafiksignaturen werden in Anwendungsmodellen über ihren Namen angesprochen.

Der Aufbau von Referenzsystem- oder Signaturobjekten muss in einem REFSYSTEM MODEL oder einem SYMBOLOGY MODEL mit den üblichen Mitteln von Klassen und Strukturen definiert sein. Referenzsystem-, Achsen- bzw. Signatur-Klassen müssen dabei Erweiterungen der durch INTERLIS angebotenen Klassen COORDSYSTEM, REFSYSTEM, AXIS bzw. SIGN sein, die ihrerseits Erweiterungen der Klasse METAOBJECT sind. Diese weist ein Attribut Name auf, das im Rahmen aller Metaobjekte eines Behälters eindeutig sein muss.

Die Metaobjekte selbst sind – wie alle Objekte – Bestandteile von Behältern. Ein Behälter – und damit seine Metaobjekte – wird in einem Anwendungsmodell verfügbar gemacht, indem ein Behältername eingeführt und das dabei vorausgesetzte Thema angegeben wird (Regel MetaDataUseDef). Objekte in Metadaten-Behältern haben Namen und entsprechen Schemaelementen, wie Attributen oder Klassen. Ein Behälter entspricht einem Thema und hat damit einen Namensraum. Erweiterung (Spezialisierung) eines Behälters heisst damit ausschliesslich Erweiterung (Spezialisierung) von dessen Namensraum. Der Behältername kann dabei auch als Erweiterung eines bereits eingeführten Namens definiert werden

(EXTENDS). Das zugehörige Thema muss dann das gleiche wie beim Basisnamen oder eine Erweiterung davon sein.

Wird der Bezug auf ein Metaobjekt (Regel MetaObjectRef) unter dem erweiterten Behälternamen gemacht, wird zuerst im zugehörigen Behälter gesucht. Wird dort kein solches Metaobjekt gefunden, wird die Suche in den Behältern gemäss den direkt oder indirekt geerbten Behälternamen fortgesetzt. Damit können Metaobjekte in mehreren Stufen bereitgestellt und verfeinert werden. Zum Beispiel können zunächst allgemeine Grafiksignaturen definiert werden, die dann auf regionaler Stufe verfeinert und ergänzt werden. Wie der konkrete Behälter auf Grund des Behälternamens angesprochen werden kann, ist dabei Sache des eingesetzten Werkzeugs.

In einem übersetzten Anwendungsmodell (vgl. Kapitel 2.5.1 Modelle), kann einem Behälternamen auch ein konkreter Behälter zugewiesen werden, der nur Übersetzungen enthält (METAOBJECT_TRANSLATION Objekte; vgl. Anhang A). Damit werden diejenigen Metaobjekte übersetzt, die durch den entsprechenden Behälter im zu Grunde liegenden Modell eingeführt wurden. Ist das zu Grunde liegende Modell selbst eine Übersetzung, kann auch da dem Behälternamen ein konkreter Behälter zugewiesen sein, der nur Übersetzungen enthält. Ist das Modell keine Übersetzung, kann einem Behälternamen nur ein konkreter Behälter zugewiesen werden, der keine Übersetzungen (d.h. keine METAOBJECT_TRANSLATION Objekte) enthält.

Syntaxregeln:

```
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.
```

Wird im aktuellen Kontext nur *ein* Metadaten-Behältername benötigt, muss diese Referenz (MetaDataUseRef) in der Metaobjekt-Referenz nicht angegeben werden.

2.10.2 Parameter

Mittels Parametern werden im Metamodell diejenigen Eigenschaften bezeichnet, die nicht das Metaobjekt selbst sondern dessen Gebrauch in der Anwendung betreffen. Ihre Definition ist durch das Schlüsselwort PARAMETER eingeleitet und ist ähnlich aufgebaut wie diejenige der Attribute.

2.10.2.1 Parameter für Referenz- und Koordinatensysteme

Für Referenz- und Koordinatensysteme bzw. ihre Achsen ist nur der vordefinierte Parameter Unit zulässig.

Wird in der Definition eines numerischen Datentyps (vgl. Kapitel 2.8.5 Numerische Datentypen) oder eines Koordinatentyps (vgl. Kapitel 2.8.7 Koordinaten) auf ein Referenz- oder Koordinatensystem verwiesen (Regel RefSys) muss eine Einheit angegeben sein, die derjenigen der entsprechenden Achse des Koordinatensystems bzw. der einzigen Achse des Referenzsystems (vgl. Kapitel 2.10.3 Referenzsysteme) verträglich ist.

2.10.2.2 Parameter von Signaturen

Die Parameter von Signaturen sind frei definierbar. Diese Parameter entsprechen den Angaben (z.B. Punktsymbol-Identifikation, Position, Drehung), die einem Grafiksystem übergeben werden müssen, damit es die Grafik erzeugen kann. Primär muss die Signatur selbst gewählt werden können. Dies erfolgt durch die Definition eines Parameters für die Referenz zur Signaturklasse, in der der Parameter definiert

ist (METAOBJECT). Als Parameter einer Signatur kann auch eine Referenz zu einem anderen Meta-Objekt angegeben werden (METAOBJECT OF). In beiden Fällen wird in der Anwendung (vgl. Kapitel 2.16 Darstellungsbeschreibungen) eine Metaobjekt-Referenz angegeben, d.h. es werden der Behälterreferenzname und der Metaobjektname angegeben. Damit können die effektive Behälter-Identifikation und Metaobjekt-Identifikation durch das jeweilige Werkzeug bestimmt werden.

Nebst diesen speziellen Parametern können Parameter gleich wie Attribute definiert werden.

Syntaxregel:

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

2.10.3 Referenzsysteme

Ohne weitere Angaben sind numerische Werte und Koordinaten Differenzangaben; sie haben keinen definierten absoluten Bezug. Um dies zu erreichen, muss ein Koordinatensystem bzw. ein Skalarsystem definiert sein. Die Modelldefinition erfolgt dabei in einem REFSYSTEM MODEL. INTERLIS stellt dafür die folgenden Klassen zur Verfügung:

```
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;
```

In Erweiterungen können weitere für die Art der Skalar- und Koordinatensysteme typische Attribute beigefügt werden und auch die Einheit durch eine Erweiterung ersetzt werden (Syntax für die Parameterdefinition vgl. Kapitel 2.10.2 Parameter und Kapitel 2.5.3 Klassen und Strukturen). Die in einem Wertebereich definierte Einheit muss dann mit ihr verträglich sein (vgl. Kapitel 2.9 Einheiten).

2.11 Laufzeitparameter

Nebst den eigentlichen Daten und den Metadaten können auch einzelne Datenelemente definiert werden, bei denen erwartet wird, dass sie von einem Bearbeitungs-, Auswerte- oder Darstellungssystem zur Laufzeit bereitgestellt werden. Sie heissen Laufzeitparameter.

Syntaxregel:

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

Gebietseinteilungen (Schlüsselwort AREA) sind für Laufzeitparameter nicht zugelassen. Da Laufzeitparameter Voraussetzungen an die Systeme definieren, die über die Sprache INTERLIS 2 hinausgehen, dürfen sie nur innerhalb von Modellen definiert werden, für die Kontrakte abgeschlossen sind.

Typische Laufzeitparameter sind zum Beispiel der Darstellungsmassstab und das aktuelle Datum.

2.12 Konsistenzbedingungen

Konsistenzbedingungen dienen der Definition von Einschränkungen, welchen die Objekte genügen müssen.

Konsistenzbedingungen, die sich auf ein einzelnes Objekt beziehen, werden durch einen logischen Ausdruck beschrieben, der sich auf die Attribute des Objektes bezieht (vgl. nächstes Kapitel). Dabei können Konsistenzbedingungen definiert werden, die obligatorisch sind und somit für alle Objekte zwingend gelten (Schlüsselwort MANDATORY), und solche, die nur in der Regel gelten. Im letzteren Fall wird angegeben, welcher Prozentanteil der Instanzen der Klasse die Bedingung normalerweise erfüllen soll (vgl. Regel PlausibilityConstraint).

Mit der Existenzbedingung (Regel ExistenceConstraint) kann gefordert werden, dass der Wert eines Attributes jedes Objektes der Bedingungs-Klasse in einem bestimmten Attribut einer Instanz anderer Klassen vorkommt. Das bedingt, dass das Bedingungsattribut mit dem anderen Attribut kompatibel ist und hat folgende Wirkung:

- Ist der Wertebereich des Bedingungsattributes gleich dem Wertebereich des anderen Attributes oder einer Erweiterung davon, muss der Bedingungswert im verlangten Attribut einer anderen Instanz vorkommen.
- Ist der Wertebereich des Attributes eine Struktur, werden alle darin enthaltenen Attribute verglichen
- Ist der Wertebereich des anderen Attributes eine Koordinate und der Wertebereich des Bedingungsattributes eine Linie, Einzelfläche oder Gebietseinteilung mit dem gleichen oder einem erweiterten Koordinatenbereich, müssen alle Koordinaten der Stützpunkte der Linie, Einzelfläche oder Gebietseinteilung im verlangten Attribut einer anderen Instanz vorkommen.

Eindeutigkeitsforderungen werden mit dem Schlüsselwort UNIQUE eingeleitet (Regel UniquenessConstraint).

Bei einer Klasse oder Assoziation (im Folgenden die aktuelle Klasse genannt) kann gefordert werden, dass eine bestimmte Kombination von Attributen bzw. von Rollen global eindeutig ist. In Frage kommen alle Attribute bzw. Rollen, die über einen Objektpfad erreichbar sind, d.h. die dem aktuellen Objekt eindeutig zugeordnet werden können.

Konzeptionell bedeutet "global", dass alle in beliebigen Behältern existierenden Objekte dieser Bedingung genügen müssen.

Bei Strukturen, Klassen und Assoziationen kann gefordert werden, dass eine bestimmte Kombination von Attributen von Unterstrukturen, die mit BAG OF oder LIST OF oder durch die geometrischen Wertebereiche POLYLINE, SURFACE oder AREA definiert sind, lokal (LOCAL), d.h. im Rahmen aller dem aktuellen Objekt oder Strukturelement zugeordneten Strukturelemente, eindeutig sein müssen.

Beispiel:

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

Ist an einer Eindeutigkeitsbedingung ein Attribut beteiligt, dessen Wert undefiniert ist, gilt die Bedingung als erfüllt.

Weitergehende Konsistenzbedingungen müssen mittels Sichten (z.B. Sicht, die eine bestimmte Klasse mit sich selbst verbindet und damit beliebige Attributkombinationen mit allen anderen Objekten der Klasse vergleichbar machen) definiert werden. Solche Sichten müssen zwingend innerhalb des Datenmodells definiert werden.

Konsistenzbedingungen, die sich auf eine Vielzahl von Objekten erstrecken (insbesondere Eindeutigkeitsbedingungen), sind nicht immer vollständig prüfbar, weil die Prüfung nur mit den lokal verfügbaren Behältern durchgeführt werden kann. Konzeptionell gelten sie aber (ausser bei Metamodellen) dennoch global (vgl. auch Anhang F Eindeutigkeit von Benutzerschlüsseln).

Syntaxregeln:

```
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.
```

Konsistenzbedingungen für eine bestimmte Klasse oder Assoziation können auch erst nachträglich (typischerweise nach der Definition einer Assoziation) definiert werden.

Syntaxregel:

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

2.13 Ausdrücke

Ausdrücke werden allgemein (Expression) z.B. als Argument von Funktionen und als logische Ausdrücke (Logical-Expression; der Ergebnistyp muss Boolean sein) z.B. in Konsistenz- und Selektionsbedingungen verwendet. Sie beziehen sich auf ein Kontextobjekt, d.h. das Objekt, für das man die Bedingung formu-

liert. Ausgehend von diesem kann z.B. ein Attribut, ein Strukturelement, eine Funktion, etc. angesprochen werden. Solche Gegenstände sowie Vergleichsgrössen wie Konstanten und Laufzeitparameter fliessen als Faktoren in Prädikate ein. Ein Prädikat ist eine Aussage, die entweder wahr oder falsch ist. Prädikate können mittels boolescher Operatoren zu einem logischen Ausdruck verknüpft werden.

Syntaxregeln:

```
Expression = Term.

Term = Term1 { 'OR' Term1 }.

Term1 = Term2 { 'AND' Term2 }.

Term2 = Predicate [ Relation Predicate ].

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

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

Bemerkungen zur Bedeutung der Syntaxregeln:

- Entsprechend der Syntaxregeln für die Terme bindet der Vergleich (Relation) am stärksten, dann AND, dann OR.
- Mit NOT und dem darauf in Klammer folgenden logischen Ausdruck wird die Verneinung dieses Ausdrucks verlangt.
- Vergleich von Faktoren. Je nach Typ des Faktors sind einzelne Vergleiche ausgeschlossen:
  - Bei Zeichenkettentypen haben Gleichheit (==) und Ungleichheit (!=, <>) die übliche Bedeutung. Grösser (>) bedeutet, dass der erste Faktor gemäss der zweiten Angabe beginnt und dann (allenfalls [>=]) weitere Zeichen aufweist. Kleiner (<) bedeutet analog, dass die zweite Angabe gemäss dem ersten Faktor beginnt und dann (allenfalls [<=]) weitere Zeichen aufweist.
  - Bei numerischen Datentypen und strukturierten Wertebereichen sind die Vergleiche im üblichen Sinn definiert. Grösser- und Kleiner-Vergleiche sind bei zirkulären Datentypen nicht zweckmässig.
  - Koordinaten können als Ganzes nur auf Gleichheit oder Ungleichheit geprüft werden. Die anderen Vergleiche stehen nur für die einzelnen Komponenten zur Verfügung (vgl. untenstehender Kommentar zu Syntaxregel Factor).
  - Bei Aufzählungen sind Grösser- und Kleiner-Vergleiche nur zulässig, wenn die Aufzählung als geordnet definiert wurde. Zwei Aufzählungswerte gelten auch dann als gleich, wenn einer eine Erweiterung des anderen ist. Damit sind beispielsweise gemäss dem im Kapitel 2.8.2 Aufzählungen benutzten Beispiel "Werktage" und "Werktage.Dienstag" gleich.
  - Bei Linien kann nur geprüft werden, ob sie undefiniert (== UNDEFINED) sind.
  - Ein Faktor kann auch ein Objekt bezeichnen. Dann kann auf Definiertheit und Gleichheit bzw. Ungleichheit geprüft werden.
- Besteht ein Faktor nicht nur aus dem unmittelbaren Gegenstand sondern auch aus dem Pfad, der zu diesem führt, gilt der Gegenstand immer als undefiniert, wenn irgendein Attribut des Pfades nicht definiert ist. Die eingebaute Funktion DEFINED (a.b) ist damit gleichwertig mit (a.b != UNDEFINED).
- Die Reihenfolge der Auswertung von Prädikaten (nicht aber die Bindungsregel!), die mit AND bzw. OR verknüpft sind, ist jedem System freigestellt.

Faktoren können gemäss den folgenden Syntaxregeln gebildet werden:

```
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).
```

Faktoren können sich auf Objekte und ihre Attribute beziehen. Dabei können schrittweise ganze Objektpfade gebildet werden. Jedes Konstrukt eröffnet den Weg vom jeweils aktuellen Objekt zum nächsten. Das erste aktuelle Objekt ergibt sich aus dem Kontext, z.B. ein Objekt der Klasse, für die eine Konsistenzbedingung definiert wird.

- THIS: Bezeichnet das so genannte Kontextobjekt, d.h. das aktuelle Objekt einer Klasse, einer Sicht oder einer Grafikdefinition, in dem ein Objektpfad verlangt wird. THIS ist z.B. als Argument anzugeben beim Aufruf einer Funktion, die ANYCLASS oder ANYSTRUCTURE als Parameter hat.
- THISAREA und THATAREA: Bezeichnen die beiden Gebietsobjekte, die vom aktuellen Linienobjekt umrandet werden. Diese sind vom Typ Surface. Der Einsatz von THISAREA und THATAREA ist nur möglich im Rahmen einer Inspektion einer Gebietseinteilung (vgl. Kapitel 2.15 Sichten).
- PARENT: Bezeichnet das Ober-Strukturelement oder Ober-Objekt des aktuellen Strukturelementes oder Objektes. Die Sicht muss dazu eine gewöhnliche Inspektion (keine Gebietsinspektion) sein (vgl. Kapitel 2.15 Sichten).
- Referenzattribut-Angabe: Bezeichnet das Objekt, das vom aktuellen Objekt bzw. von der aktuellen Struktur aus über das gegebene Referenzattribut zugeordnet ist.
- Beziehungspfad: Bezeichnet das assoziierte (Daten-) Objekt oder das assoziierte Objekt der Assoziationsklasse. Dabei gilt:
  - Will man ein über die Beziehung verbundenes (Daten-) Objekt erreichen, wird der entsprechende Beziehungszugang angegeben (vgl. Kapitel 2.7.5 Beziehungszugänge). Dies ist nur zulässig, wenn die Zuordnung eindeutig ist, d.h. wenn es auf Grund der Kardinalität höchstens ein zugeordnetes Objekt geben kann. Bei Zweierbeziehungen ist die Zulässigkeit gegeben, wenn die Rolle des verlangten Beziehungszugangs die maximale Kardinalität eins hat, bei Mehrfachbeziehungen wenn die Rollen zu allen Bezugsklassen die maximale Kardinalität eins haben.

  - Will man das Beziehungsobjekt selbst erreichen (d.h. das Objekt der Assoziationsklasse), wird dem Beziehungszugang ein Backslash ('\') vorangestellt. Dies ist nur zulässig, wenn es auf Grund der Kardinalität höchstens ein zugeordnetes Beziehungsobjekt geben kann.
- Rollenangabe: Bezeichnet das vom aktuellen Objekt der Assoziationsklasse aus über die Rolle zugeordnete Objekt. Der Pfad ist nur zulässig, wenn die maximale Kardinalität der Rolle höchstens 1 ist.
- Basis-Sicht-Angabe: Mit dem (lokalen) Namen der Basis-Sicht wird in der aktuellen Sicht bzw. in der aktuellen abgeleiteten Beziehung das entsprechende (virtuelle) Objekt der Basis-Sicht bezeichnet.

Beim Bezug auf ein Attribut, meint man den Wert des Attributes des Kontextobjekts oder des durch den Pfad bezeichneten Objekts. Zusätzlich werden Pfade, die mit einem Attribut enden, als Attributpfade bezeichnet und auch unabhängig von Faktoren in verschiedenen Syntaxregeln verwendet.

- Im Normalfall genügt die Angabe des Attributnamens.
- Handelt es sich um ein Koordinatenattribut bezeichnet man durch Angabe der Nummer der Achse die entsprechende Komponente der Koordinate. Die erste Komponente hat den Index 1.
- Das implizite Attribut AGGREGATES ist in Aggregationssichten (vgl. Kapitel 2.15 Sichten) definiert und bezeichnet den Satz (BAG OF) der aggregierten Basisobjekte.

Bei geordneten Unterstrukturen (LIST) können einzelne Elemente angesprochen werden. Zulässige Indizes sind:

- FIRST: das erste Element.
- LAST: das letzte Element.
- Index-Nummer: Der angegebene Index muss kleiner oder gleich der in der Kardinalität festgelegten maximalen Anzahl sein. Das erste Element hat den Index 1. Ist er kleiner oder gleich der in der Kardinalität festgelegten minimalen Anzahl, existiert immer ein entsprechendes Element; ist er grösser ist die Existenz des Elementes nicht gewährleistet. Der Faktor kann als Folge undefiniert werden.

Faktoren können auch Funktionsaufrufe sein. Als ihre Argumente kommen in Frage:

- Faktoren: Der Typ des Faktors muss mit dem Argumenttyp kompatibel sein.
- Der Name einer verlangten Klasse (Regel ViewableRef) oder eine Funktion, die als Ergebnis eine Klasse liefert: Der formale Parameter muss vom Typ CLASS sein.

Als Vergleichswerte kommen Funktionsaufrufe, Laufzeitparameter (vgl. Kapitel 2.16 Darstellungsbeschreibungen) und Konstanten in Frage.

2.14 Funktionen

Eine Funktion wird mittels ihrem Namen, den formalen Parametern sowie einer kurzen Funktionsbeschreibung als Erläuterung definiert. Die Namen der Parameter haben nur dokumentarischen Wert. Die Definition ist nur innerhalb von Kontrakten zulässig, da sonst eine automatische Auswertung der Modelle nicht mehr gewährleistet ist.

Als formale Parameter und Funktionsresultat kommen in Frage:

- Die für Attribute zulässigen Typen, insbesondere auch Strukturen. Als Argumente (d.h. aktuelle Parameter) kommen entsprechende Faktoren (vgl. Regel Factor) in Frage.
- Wird dabei eine Struktur angegeben, kommen als Argumente primär Strukturelemente in Frage. Es können aber auch Objektpfade angegeben werden, die zu Objekten führen, die eine Erweiterung der Struktur ist. Insbesondere können bei ANYSTRUCTURE beliebige Objektpfade angegeben

werden.

- Wird OBJECT OF angegeben, kommen als Argumente die mittels eines Objektpfads erreichbaren Objekte in Frage, die der Definition entsprechen. Insbesondere können bei OBJECT OF ANYSTRUCTURE beliebige Objektpfade angegeben werden. Ähnlich wie bei den Bezügen zu anderen Objekten (vgl. Kapitel 2.6.3) können die zulässige Basisklasse und allfällige Einschränkungen definiert und in Erweiterungen spezialisiert werden. ANYSTRUCTURE kann dabei zu ANYCLASS spezialisiert werden, sofern keine Einschränkungen (RESTRICTED TO) definiert sind.

Syntaxregeln:

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

Es sind folgende Standardfunktionen definiert:

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

Liefert die Klasse des Objektes.

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

Liefert true, wenn die Klasse des ersten Argumentes der Klasse oder einer Unterklasse des zweiten Argumentes entspricht.

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

Liefert true, wenn das Objekt des ersten Argumentes zur Klasse oder zu einer Unterklasse des zweiten Argumentes gehört.

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

Liefert die Anzahl Elemente, die der Bag (oder die Liste) enthält.

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

Rechnet den numerischen Wert des Parameters "from" in den numerischen Rückgabewert um und berücksichtigt dabei die Einheiten, die mit dem Parameter und mit der Verwendung des Resultatwertes (typischerweise mit dem Attribut, dem das Resultat zugewiesen wird) verbunden sind. Diese Funktion darf nur angewendet werden, wenn die Argumente von "from" und vom Rückgabeparameter verträglich sind, d.h. wenn ihre Einheiten von einer gemeinsamen Einheit abgeleitet werden.

2.15 Sichten

Sichten (Views) sind Klassen und Strukturen, deren Objekte nicht originär sondern virtuell sind, da sie aus Objekten anderer Sichten oder Klassen bzw. Strukturen abgeleitet werden. Sichten werden unter anderem dazu verwendet, die Grundlagen für Grafiken und spezielle Konsistenzbedingungen zu formulieren. Eine weitere Anwendung ist die Weitergabe der Daten an Empfängersysteme in einer abgeleiteten, in der Regel vereinfachten Form.

Sichten werden nur transferiert, wenn sie in einem VIEW TOPIC definiert sind. In diesem Fall erfolgt ihre Übertragung wie der vollständige Transfer (Schlüsselwort FULL) von normalen Klassen, so dass sich ein Datenempfänger (vgl. Kapitel 3 Sequentieller Transfer) nicht darum zu kümmern braucht, wie die (virtuellen) Objekte erzeugt wurden. Sichten können auch explizit vom Transfer ausgeschlossen werden (TRANSIENT), wenn sie nur lokale Bedeutung haben, d.h. wenn sie nur als Basis für andere Sichten dienen. Die inkrementelle Nachlieferung von Sichten ist ausgeschlossen, da Sichtobjekten keine Objektidentifikation zugeordnet wird.

Sichten können abstrakt (ABSTRACT) oder konkret sein. Konkrete Sichten können auch auf abstrakten Grundlagen beruhen. Dabei können aber nur diejenigen Grundlagenattribute angesprochen werden, die konkret sind. Ist dies nicht der Fall, muss die Sicht selbst als abstrakt erklärt werden.

Sichten können auch erweitert werden (EXTENDED oder EXTENDS). Dabei ist es allerdings nicht möglich das Bildungsgesetz zu verändern. Die Erweiterung dient dazu, Erweiterungen der Sichten, Klassen und Strukturen, auf der die Sicht aufbaut, so zur Kenntnis zu nehmen, dass weitere Selektionen, Attribute und Konsistenzbedingungen formuliert werden können.

Syntaxregeln:

```
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 ).
```

Mit dem Bildungsgesetz (FormationDef) einer Sicht wird definiert, wie und aus welchen Grundlagen die virtuellen Objekte der Sicht gebildet werden.

Die Sicht-Projektion (Schlüsselwort PROJECTION OF) ist die einfachste Sicht. Mit ihr wird die Basis-Klasse (Klasse, Struktur oder Sicht) in veränderten Form (z.B. Attribute nur teilweise und in veränderter Reihenfolge) gesehen.

Mit der *Verbindung* (Schlüsselwort JOIN OF) wird das kartesische Produkt (oder Kreuzprodukt) der Basis-Klassen (Klasse oder Sicht) gebildet, d.h. es gibt so viele Objekte der Verbindungs-Klasse wie es Kombinationen von Objekten der verschiedenen Basis-Klassen gibt. Es können auch so genannte "Outer-Joins", also Verbindungen von Objekten der ersten Basisklasse mit (an sich inexistenten) Leerobjekten der weiteren Basisklassen definiert werden (Angabe "(OR NULL)"). Solche Leerobjekte werden beigefügt, wenn zu einer bestimmten Kombination der vorangegangenen Objekte kein Objekt der gewünschten weiteren Klasse gefunden wird. Alle Attribute des Leerobjektes sind undefiniert. Die entsprechenden Sicht-Attribute dürfen darum nicht obligatorisch sein.

Die *Vereinigung* (Schlüsselwort UNION OF) ermöglicht die Verschmelzung verschiedener Basis-Klassen zu einer einzigen Klasse. Einem Attribut der Vereinigungs-Sicht werden typischerweise die Attribute der verschiedenen Basis-Klassen zugewiesen. Der Attributtyp der Basis-Klasse muss mit demjenigen der Vereinigungs-Sicht verträglich sein (gleicher Typ oder Erweiterung davon).

Mit der Zusammenfassung (Schlüsselwort AGGREGATION) werden alle oder diejenigen Instanzen einer Basismenge zu einer Instanz zusammengefasst, bei denen die verlangte Attributkombination identisch ist. Innerhalb der Aggregationssicht steht mittels dem impliziten Attribut AGGREGATES (vgl. Kapitel 2.13 Ausdrücke) der zugehörige Satz von Originalobjekten als BAG zur Verfügung. Dieses implizite Attribut

gehört nicht zu den eigentlichen Attributen der Sicht und wird darum auch dann nicht transferiert, wenn ein Transfer verlangt ist. Es kann z.B. einem entsprechenden Attribut der Aggregationssicht zugewiesen oder als Argument einer Funktion übergeben werden.

Mit der *Inspection* (Schlüsselwort INSPECTION OF) erhält man die Menge aller Strukturelemente (mit BAG OF oder LIST OF) oder gemäss Linie, Einzelfläche oder Gebietseinteilung definierte), die zu einem Unterstrukturattributeiner Objekt-Klasse gehören. Mit der Inspection von Gebietseinteilungen (Schlüsselwort AREA INSPECTION) erhält man die Linien der Gebietseinteilung (als Struktur SurfaceEdge) genau einmal. Die Gebiete, die an die Linie angrenzen, können mit THISAREA bzw. THATAREA angesprochen werden (vgl. Kap 2.13 Ausdrücke). Es ist dabei Sache der Implementation, wie fein die Linien unterteilt werden. Es ist also zulässig pro Verbindungsstück von zwei Stützpunkten ohne weiteren Zwischenstützpunkt ein Linienobjekt zu melden oder Verbindungsstücke soweit als möglich (Attribute müssen gleich sein) zusammenzufassen.

Eine normale Inspection auf ein Gebietseinteilungsattribut liefert hingegen wie die Inspection eines Flächenattributs alle Randelemente (Struktur SurfaceBoundary). Wird eine weitere Inspection auf das Attribut Lines gemacht, erhält man ebenfalls alle Linien (Struktur SurfaceEdge). Auf diese Art kommen sie aber im Falle der Gebietseinteilung doppelt vor (einmal für jedes beteiligte Gebietsobjekt).

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

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

Die Inspection einer Linie (POLYLINE) liefert folgende Struktur:

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

Mit INTERLIS wird nur eine konzeptionelle Beschreibung der Sicht gemacht. Es wird ausdrücklich nichts unternommen, um eine effiziente Realisierung der Sichten zu unterstützen. Sichten generieren gehört deshalb auch zu einer speziellen Erfüllungsstufe.

Syntaxregeln:

```
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 } ';'.
```

Alle in einer Sicht verwendeten Basissichten erhalten innerhalb der verwendenden Sicht einen Namen, unter dem sie angesprochen werden können. Dieser Name entspricht dem Namen der Basissicht, sofern keine Umbenennung mittels eines expliziten (lokalen) Base-Namens definiert wird. Eine solche Umbenennung ist insbesondere nötig, wenn Verbindungen definiert werden, bei denen mehrmals die gleiche Basisklasse angesprochen wird.

Syntaxregeln:

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

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

Will man in einer Sicht bzw. einer Sichterweiterung Erweiterungen von Basisklassen berücksichtigen und damit zusätzliche Attribute, Selektionen oder Konsistenzbedingungen formulieren, muss eine entsprechende Erweiterungsdefinition (BaseExtentionDef) aufgeführt werden. Sie geht von einer bereits definierten Basissicht aus und beschreibt die Erweiterungen (müssen Erweiterungen der bisherigen Basissicht sein) selbst wieder als Basissichten. Wird eine solche Sichterweiterung innerhalb von Ausdrücken verwendet, liefert sie den Wert "UNDEFINED", wenn das zum virtuellen Objekt gehörige Basisobjekt nicht zu dieser Sichterweiterung passt.

Syntaxregel:

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

Die durch das Bildungsgesetz definierte Menge der Sichtobjekte kann mittels Bedingungen (Schlüsselwort WHERE) zusätzlich eingeschränkt werden.

Syntaxregel:

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

Was die Attribute (und damit die Empfängersicht) und die Konsistenzbedingungen betrifft, sind Sichten grundsätzlich gleich aufgebaut, wie Klassen und Strukturen. Im Sinne einer Schreiberleichterung wird zusätzlich die Möglichkeit angeboten, alle Attribute einer Sichtbasis in der gleichen Reihenfolge zu übernehmen (ALL OF). Dies ist bei Vereinigungen unsinnig und darum auch unzulässig.

Syntaxregel:

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

In den einfachen Fällen, wo ein Attribut einer Basissicht übernommen wird, genügt es den Attributnamen und die Zuordnung des Basisattributes anzugeben. Solche Definitionen sind immer final, können also nicht mehr erweitert werden.

Bei Vereinigungen muss für jedes Attribut vollständig angegeben werden, aus welchen Attributen der Basis-Klassen es abgeleitet wird. Ein Attribut muss sich aber nicht auf alle Basis-Klassen beziehen, sofern der Attributtyp undefinierte Werte zulässt. Für die fehlenden Basis-Objekte gilt es als undefiniert.

Das folgende Beispiel zeigt, wie eine Sicht beschrieben werden kann, die ermöglicht, mit Hilfe des Konstrukts DERIVED FROM eine eigentliche Beziehung zu definieren (vgl. dazu Kapitel 2.7.1 Allgemeines zu 2.7. Eigentliche Beziehungen).

```
DOMAIN
  CHSurface = .... ;

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

CLASS A =
  a1: CHSurface;
END A;

CLASS B =
  b1: CHSurface;
END B;

VIEW CLASS 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 Darstellungsbeschreibungen

Eine Darstellungsbeschreibung besteht aus Grafikdefinitionen, die immer auf einer Sicht oder einer Klasse basieren (Schlüsselwort BASED ON). Mit einer Grafikdefinition wird konzeptionell versucht, jedem Objekt dieser Sicht oder Klasse – sofern es nicht mit einer Selektion (Schlüsselwort WHERE) noch ausgefiltert wurde – durch eine oder mehrere Zeichnungsregeln (vgl. Regel DrawingRule) je eine Grafiksignatur zuzuordnen (Punktsymbol, Linie, Flächenfüllung, Textanschrift) und damit ein oder mehrere Grafikobjekte zu erzeugen, welche die entsprechende Darstellung auslösen (vgl. Figur 5). Dazu muss jede Zeichnungsregel eine Grafiksignatur (mit Metaobjekt-Namen) auswählen und Argumente festlegen für die dazugehörigen Parameter.

In runden Klammern (Regel Properties) können die Vererbungseigenschaften definiert werden. Ist eine Grafikdefinition abstrakt, entstehen aus ihr keine Grafikobjekte. Die Erweiterung einer Grafikdefinition muss auf der gleichen Klasse basieren wie die Basisgrafikdefinition (BASED ON fehlt) oder auf einer ihrer Erweiterungen.

Die Zeichnungsregel wird mit einem Namen identifiziert, damit sie in Erweiterungen aufgegriffen und verfeinert werden kann (Anmerkung: verfeinert im Sinne der Spezialisierung aber auch zusätzlicher Parameterwerte). Existieren zu einer Zeichnungsregel Erweiterungen (in erweiternden Darstellungsbeschreibungen), erzeugen diese keine neuen Grafikobjekte, sondern beeinflussen nur die Signaturparameter des durch die Basisdefinition vorgegebenen Grafikobjektes. Es ist zulässig zu einer Grafikdefinition mehrere Erweiterungen zu definieren. Sie werden alle (in der Reihenfolge ihrer Definition) ausgewertet. Dies kann insbesondere genutzt werden, um für verschiedene Aspekte (z.B. die verschiedenen Zeichnungsregeln) verschiedene Erweiterungsstapel vorzusehen. Anschliessend werden die verschiedenen Signaturparameter bestimmt. Die Definition kann dabei in mehreren Schritten erfolgen. Für jeden Parameter gilt derje-

nige Wert, der als letztes definiert wurde. Dabei wird zuerst die primäre Definition ausgewertet, dann allfällige Erweiterungen. Parameter-Zuweisungen können zudem noch an eine Bedingung (vgl. Regel CondSignParamAssignment) geknüpft werden, d.h. die Zuweisung erfolgt nur, wenn die Bedingung erfüllt ist. Wird die Selektionsbedingung nicht erfüllt, fallen auch allfällige Untererweiterungen ausser Betracht.

Sobald Zeichnungsregeln konkret sind, muss definiert sein, zu welcher Klasse die zuzuordnenden Grafiksignaturen gehören. In Erweiterungen von Zeichnungsregeln kann diese Klasse von Grafiksignaturen durch eine Klasse ersetzt werden, welche eine Erweiterung der bisherigen ist. "Verantwortliche" Klasse von Grafiksignaturen ist zunächst diejenige, zu der die zugeordnete Grafiksignatur-Objekt (ein Metaobjekt) gehört. Den in der "verantwortlichen" Klasse eingeführten Parameter müssen die konkreten Werte zugewiesen werden. Entsprechen die angegebenen Parameter einer erweiterten Klasse von Grafiksignaturen, wird diese zur "verantwortlichen" Klasse, sofern die Klasse der Grafiksignatur der Zeichnungsregel mit ihr übereinstimmt oder eine Erweiterung von ihr ist.

In den erwähnten Bedingungen dürfen Objekt-Attribute (vgl. AttributePath in Regel SignParamAssignment) auch mit Laufzeit-Parametern verglichen werden (vgl. Kapitel 2.11 Laufzeitparameter). Laufzeitparameter, die für die Grafik relevant sind (z.B. der Massstab der gewünschten Grafik), werden typischerweise in Symbologiemodellen definiert, da sie wie die Parameter der Signaturen die von einem System erwarteten Grafikfähigkeiten beschreiben. Für einen Parameter einer Grafiksignatur, der ein Metaobjekt verlangt, muss eine Metaobjekt-Referenz angegeben werden (vgl. dazu Kapitel 2.10 Umgang mit Metaobjekten).

Der Wert eines gewöhnlichen Parameters einer Grafiksignatur wird als Konstante oder als Verweis auf ein Objekt-Attribut angegeben (vgl. Factor in Regel SignParamAssignment). Dabei wird immer auf das Attribut eines Objektes aus der Basis-Klasse oder Basis-Sicht verwiesen, welches mit Hilfe von BASED ON spezifiziert wurde.

Da die Darstellung häufig von Attributen abhängt, die mittels Aufzählungen definiert sind, wird dafür ein spezielles Konstrukt angeboten: Der Aufzählbereich. Ein Aufzählbereich ist entweder ein einzelner Knoten des Aufzähltyp-Baums oder ein Intervall von Knoten definiert durch zwei Knoten der gleichen Stufe. Intervalldefinitionen sind allerdings nur zulässig, wenn es sich um einen geordneten Aufzähltyp handelt. Wenn der Attributwert in einem der angegebenen Aufzählbereiche liegt, wird der entsprechende Parameter-Wert gesetzt. Die konkreten Signaturen ergeben sich aus dem Signaturenmodell. Dort werden die Signaturklassen samt den für ihre Anwendung nötigen Laufzeitparametern (Schlüsselwort PARAMETER) definiert. Es ist zulässig, dass numerische Datentypen nur abstrakt definiert sind.

Syntaxregeln:

```
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 ].
```

Für die Verwendung in Signaturenmodellen ist die Klasse SIGN durch INTERLIS vordefiniert:

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

Für konkrete Signaturenklassen ist diese Basis-Klasse zu erweitern. Dabei werden einerseits die konkreten Daten, andererseits die Parameter definiert.

Das folgende Beispiel skizziert, wie aus einer Punktklasse mit Koordinaten, Zeichenkette und einer Aufzählung als Attribute die entsprechenden Grafiken (Punktsymbole und Textanschriften) definiert werden.

Das Signaturenmodell sei wie folgt definiert:

```
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.
```

Zu diesem Signaturenmodell seien konkrete (Signaturen-)Objekte erfasst und unter dem Signaturenbibliotheks-Namen (d.h. Behälter-Namen) SimpleSignsBasket abgelegt worden. Die erfassten Signaturobjekte (Klasse Symbol) heissen Punktsignatur, Quadratsignatur und Kreissignatur, die Schriftarten (Klasse Textlabel) Schrift1 und Schrift2.

```
MODEL DatenModell (de) =

  DOMAIN
    LKoord = COORD
      0.000 .. 200.000 [m],
```

```
        0.000 .. 200.000 [m],
        ROTATION 2 -> 1;


    TOPIC PunktThema =

      DOMAIN
        Punktart = (Stein
                     (gross,
                      klein),
                    Bolzen,
                    Rohr,
                    Kreuz,
                    unvermarkt) ORDERED;

      CLASS Punkt =
        Lage: LKoord;      !! LKoord sei ein Koordinaten-Wertebereich
        Art: Punktart;
        PunktName: TEXT*12;
      END Punkt;

    END PunktThema;

  END DatenModell.


  MODEL SimpleGrafik (de) =

    CONTRACT ISSUED BY Unknown;

    IMPORTS DatenModell;
    IMPORTS SimpleSignsSymbology;

    SIGN BASKET SimpleSignsBasket ~ SimpleSignsSymbology.SignsTopic;


    TOPIC PunktGrafikenThema =
      DEPENDS ON DatenModell.PunktThema;

      GRAPHIC SimplePunktGrafik BASED ON DatenModell.PunktThema.Punkt =

        Symbol OF SimpleSignsSymbology.SignsTopic.Symbol: (
          Sign := {Punktsignatur};
          Pos := Lage
        );

      END SimplePunktGrafik;

    END PunktGrafikenThema;

  END SimpleGrafik.
```

Mit dieser Grafik (basierend auf dem Symbologiemodell SimpleSignsSymbology und auf der Darstellungsbeschreibung SimpleGrafik) werden für alle Punkte aus Klasse Punkt einfache Punktsignaturen (dots) gezeichnet.

Nun kann man sich auch vorstellen, dass jemand eine verbesserte Grafik wünscht. Die Verbesserung kann dabei in verschiedener Hinsicht erfolgen, zum Beispiel:

- Es soll zusätzliche Signaturen geben (Punktsignaturen Kreuzsignatur, Dreiecksignatur). Dafür braucht es eine zusätzliche Signaturenbibliothek mit dem Namen SimpleSignsPlusBasket. Da sie die Bibliothek SimpleSignsBasket erweitert, werden die Signaturobjekte (bzw. Metaobjekte) in beiden Bibliotheken gesucht. Würde man die Bibliothek SimpleSignsBasket direkt erweitern (EXTENDED) würde für alle im Modell GrafikPlus erzeugten Grafiken - inklusive derjenigen, die

vom Modell SimpleGrafik geerbt wurden - die Signaturen zuerst in der erweiterten Bibliothek, dann in der Basisbibliothek (SimpleSignsBasket) gesucht.

- Die Signaturen sollen skalierbar sein, damit z.B. grosse und kleine Quadrate mit der gleichen Punktsignatur erstellt werden können. Dafür braucht es ein erweitertes Signaturenmodell, in dem der Skalierungsmassstab der Signaturen als Parameter definiert ist. Da die Signaturklassen keine zusätzlichen Attribute aufweisen, müssen nicht notwendigerweise entsprechende Bibliotheken existieren.
- Je nach Punktart sollen verschiedene Punktsignaturen gezeichnet werden: Steine als grosse bzw. kleine Quadrate, Bolzen als Kreise und Kreuze und Rohre mit der Kreuzsignatur. Die eigentliche Punktsignatur kann direkt aus der Punktart abgeleitet werden. Der Skalierungsfaktor für kleine Quadrate zur Darstellung von kleinen Steinen, wird mit einer zusätzlichen Zuweisung erreicht. Unvermarkte Punkte werden weiterhin als einfache Punktsignaturen gezeichnet. Darum braucht es für diesen Fall keine neue Zuweisung.

```
SYMBOLOGY MODEL ScalableSignsSymbology (en) =

  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 GrafikPlus (de) =

  CONTRACT ISSUED BY Unknown;

  IMPORTS SimpleGrafik;
  IMPORTS SimpleSignsSymbology;
  IMPORTS ScalableSignsSymbology;

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


  TOPIC PunktGrafikenPlusTop EXTENDS SimpleGrafik.PunktGrafikenThema =

    GRAPHIC PunktGrafikPlus EXTENDS SimplePunktGrafik =

      Symbol (EXTENDED) OF ScalableSignsSymbology.ScalableSignsTopic.Symbol: (
        Sign := ACCORDING Art (
          {Quadratsignatur} WHEN IN #Stein,
          {Kreissignatur} WHEN IN #Bolzen,
          {Kreuzsignatur} WHEN IN #Rohr .. #Kreuz
        )
      ),
      WHERE Art == #Stein.klein (
          ScaleFactor := 0.5
      );

      Text OF SimpleSignsSymbology.Signs.Textlabel: (
        Sign := {Schrift1};
```

```
          Pos := Lage;
          Text := PunktName
        );

      END PunktGrafikPlus;

    END PunktGrafikenPlusTop;

  END GrafikPlus.
```

3 Sequentieller Transfer

3.1 Einleitung

In diesem Kapitel wird der sequentielle INTERLIS-Transferdienst beschrieben. Damit können Datenbestände zwischen verschiedenen Systemen systemneutral ausgetauscht werden. Der INTERLIS-Transferdienst unterstützt den vollständigen wie auch den inkrementellen (bzw. differenziellen) Austausch von Datenbeständen (Replikation). Der Transferdienst kann auf jedes INTERLIS-Modell angewendet werden. Damit ist es z.B. möglich Daten (Datenmodell) und Signaturobjekte (Signaturenmodell) mit dem gleichen Mechanismus zu transferieren.

Im Moment ist der INTERLIS-Transferdienst als Austausch von XML-Dateien definiert (www.w3.org/XML/). Zur breiteren Nutzung dieser INTERLIS/XML-Dateien ist es unter anderem auch möglich, XML-Schema-Dokumente (www.w3.org/XML/Schema) zu erzeugen. In Zukunft ist es jedoch denkbar, dass weitere INTERLIS-Transferdienste definiert werden (z.B. auf Basis Webservices oder CORBA). Aus diesem Grund ist die Beschreibung des INTERLIS-Transferdienstes in die Unterabschnitte *Allgemeine Regeln* und *XML-Codierung* unterteilt. Die allgemeinen Regeln gelten für *jeden sequentiellen* INTERLIS-Transferdienst, unabhängig von der konkreten Codierung oder Übermittlung. Die Regeln unter *XML-Codierung* gelten speziell für XML-formatierte INTERLIS-Transferdateien.

3.2 Allgemeine Regeln für den sequentiellen Transfer

3.2.1 Ableitbarkeit aus dem Datenmodell

Jeder INTERLIS-Transfer kann aus dem dazugehörigen INTERLIS-Datenmodell durch die Anwendung von Regeln abgeleitet werden (modell-basierter Datentransfer).

3.2.2 Lesen von erweiterten Modellen

Ein INTERLIS-Transfer ist immer so aufgebaut, dass ein Leseprogramm das für ein bestimmtes Datenmodell programmiert oder konfiguriert worden ist, auch Daten von Erweiterungen dieses Datenmodells ohne Kenntnis der erweiterten Modelldefinitionen lesen kann. Daten aus Erweiterungen werden dabei von dem Leseprogramm ohne Fehlermeldung ignoriert.

3.2.3 Aufbau eines Transfers: Vorspann

Ein INTERLIS-Transfer ist ein sequentieller Objektstrom. Der Objektstrom ist in die Teile Vorspann und Datenbereich unterteilt.

Im Vorspann sind mindestens folgende Angaben zum Transfer enthalten:

- Angabe der aktuellen INTERLIS-Versionsnummer (siehe Kapitel 2.3 Hauptregel).
- Verweis auf das/die zugehörige(n) Datenmodell(e).
- Bezeichnung des Absenders (SENDER).

Optional können im Vorspann Kommentare enthalten sein.

Der Aufbau des Datenbereichs ist in den folgenden Abschnitten näher beschrieben.

3.2.4 Transferierbare Objekte

Im Datenbereich können Objekte (d.h. Objektinstanzen) von konkreten Klassen, Beziehungen, Sichten und Grafikdefinitionen transferiert werden. Objekte von Sichten werden auf dem Transfer wie Objekte von konkreten Klassen behandelt. Die inkrementelle Nachlieferung von Sichten ist vorläufig nicht möglich. Objekte von Sichten werden nur transferiert, wenn die zugehörigen Sichten innerhalb eines VIEW TOPIC deklariert wurden, sonst werden sie nicht übermittelt. Ausserdem werden Sichten nicht übermittelt, wenn sie mit TRANSIENT markiert wurden.

3.2.5 Reihenfolge der Objekte im Datenbereich

Der Datenbereich besteht aus einer Folge von Behältern (Themen-Instanzen). Behälter können nur vollständig übertragen werden. Bei inkrementeller Nachlieferung werden nur die geänderten bzw. gelöschten Objekte übertragen. Zusammen mit der Vorgeschichte wird jedoch auch bei der inkrementellen Nachlieferung konzeptionell der ganze Behälter übertragen. Es ist grundsätzlich möglich, dass ein Transfer Behälter aus verschiedenen Modellen enthält. Ein Behälter enthält wiederum alle seine Objekte. Die Reihenfolge der Objekte im Transfer ist beliebig, insbesondere müssen die Objekte innerhalb eines Behälters nicht unbedingt nach Beziehungen geordnet oder nach Klassen gruppiert sein (im Gegensatz zu INTERLIS 1). Leere Behälter müssen nicht übertragen werden.

3.2.6 Codierung der Objekte

Im Objektstrom erhält jeder Behälter und jedes Objekt eine Identifikation. Die Identifikationen von Behältern und Objekten müssen über den gesamten Transfer eindeutig sein. Zu jedem Objekt wird ausserdem die Behälter-Identifikation mitgeliefert, in dem das Objekt ursprünglich erzeugt wurde (Originalbehälter). Bei inkrementeller Nachlieferung, bzw. beim initialen Transfer, muss die Identifikation des Behälters und der Objektinstanz generell und stabil sein, d.h. ein Objektidentifikator (vgl. Anhang E Aufbau von Objektidentifikatoren (OID)).

Alle Objektattribute (inkl. COORD, SURFACE, AREA, POLYLINE, STRUCTURE, BAG OF, LIST OF, etc.) werden unmittelbar zum Objekt gespeichert. Attribute vom Typ AREA werden wie Attribute vom Typ SURFACE codiert. Attribute vom Typ BAG werden wie Attribute vom Typ LIST codiert. STRUCTURE wird wie LIST {1} codiert.

Als Zeichenvorrat für die Übermittlung von Attributwerten stehen standardmässig nur die druckbaren Zeichen des US-ASCII Zeichensatzes (32 bis 126) und die Zeichen gemäss Zeichentabelle im Anhang B zur Verfügung.

3.2.7 Transferarten

Zu jedem Behälter müssen folgende Angaben geliefert werden:

- Angaben zur Art (KIND) des Transfers: FULL, INITIAL oder UPDATE.
- Angaben zum Anfangs- (STARTSTATE) bzw. Endzustand (ENDSTATE) der Nachlieferung (nur bei Transferart INITIAL oder UPDATE).

Es ist zulässig, dass im gleichen Transfer Behälter mit verschiedenen Transferarten (FULL, INTIAL und/oder UPDATE) vorkommen. Die Transferarten haben folgende Bedeutung:

- FULL – Vollständiger Transfer. Beim Erhalt eines FULL-Behälters, muss der Empfänger einen neuen Behälter zuerst initialisieren und dann alle Objekte mit INSERT in den Behälter einfügen. FULL ist als Basis für Nachlieferungen ungeeignet, da die Objektidentifikationen nur für diesen Transfer gültig sind. Transferdateien gemäss INTERLIS 1 entsprechen FULL. In der Transferart FULL kann nur die Operation INSERT vorkommen.
- INITIAL – Erstlieferung. Entspricht der Transferart FULL mit dem Unterschied, dass der Behälter und die darin enthaltenen Objekte generelle und stabile OID's besitzen müssen. In der Transferart

INITIAL kann ebenfalls nur die Operation INSERT vorkommen.

- UPDATE – Nachlieferung. Ein UPDATE-Behälter enthält Objekte mit INSERT-, UPDATE- oder DELETE-Operationen. Alle Objekte und der Behälter besitzen generelle und stabile OID's. UPDATE-Behälter dürfen vom Zielsystem nur verarbeitet werden, wenn der Anfangszustand (STARTSTATE) des Behälters bereits mit INITIAL oder UPDATE empfangen wurde.

Für die Transferart UPDATE gelten ausserdem folgende zusätzlichen Transferregeln:

- Das Empfängersystem kann davon ausgehen, dass nach der vollständigen Verarbeitung aller Daten eines UPDATE-Behälters wieder ein konsistenter Zustand herrscht, d.h. ein UPDATE-Behälter führt einen Behälter von einem konsistenten Zustand STARTSTATE in einen anderen konsistenten Zustand ENDSTATE über.
- Ein UPDATE-Behälter ist für sich allein gesehen nicht konsistent, da Beziehungen meist nur zusammen mit der Vorgeschichte aufgelöst werden können.

Ausserdem muss zu jedem Objekt die Nachlieferungsoperation angegeben werden (vgl. dazu auch 1.4.5 Behälter, Replikation und Datentransfer). Die Operationen INSERT, UPDATE und DELETE bedeuten folgendes:

- Die Operation INSERT hat die Bedeutung von "neues Objekt einfügen" (insert object).
- Die Operation UPDATE bedeutet "Objektattributwerte aufdatieren" (update object). Es müssen alle Attribute (nicht nur die geänderten) geliefert werden.
- Die Operation DELETE bedeutet "Objekt löschen" (delete object). Es sollen alle Attribute (nicht nur der OID) geliefert werden.

3.3 XML-Codierung

3.3.1 Einleitung

Im Gegensatz zu den Regeln im Kapitel 3.2 Allgemeine Regeln für den sequentiellen Datentransfer gelten die Regeln unter XML-Codierung nur für gemäss XML-1.0 Standard formatierte Transferdateien (siehe auch www.w3.org/XML/). Für die Formalisierung der Transferformat-Ableitungsregeln wird die bereits in Kapitel 2.1 Verwendete Syntax eingeführte EBNF-Notation benutzt. Folgende Regeln werden hier bereits vordefiniert:

```
XML-Text = beliebiger freier Text.
XML-String = beliebiger einzeiliger Text.
XML-Value = '"' XML-String '"'.
XML-ID = '"' x { Letter | Digit} '"'.
```

Bei der Regel XML-ID ist das 'x' nicht Teil des ID-Wertes, sondern Teil der Codierung.

Um die Leserlichkeit der einzelnen Ableitungsregeln zu verbessern, werden zusätzlich die Makros TAG und ETAG verwendet:

```
TAG ( Tagwert )
```

generiert eine EBNF-Teilregel der Form:

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

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

generiert eine EBNF-Teilregel der Form:

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

```
ETAG ( Tagwert )
```

generiert eine EBNF-Teilregel der Form:

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

Die Folge %Argument% muss jeweils durch den aktuellen Inhalt des Arguments ersetzt werden.

Beispiele:

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

3.3.2 Zeichencodierung

Als Zeichen für die Codierung von XML-Text bzw. XML-String stehen nur die ASCII Zeichen von 32 bis 126 bzw. die Zeichen aus Anhang B zur Verfügung. Die Zeichen werden gemäss der Codierungsregel UTF-8 oder als XML Character Reference bzw. XML Entity Reference codiert. Ausserdem müssen die XML-Sonderzeichen '&', '<' und '>' wie folgt codiert werden:

- '&' muss durch die Zeichenfolge '&' ersetzt werden.
- '<' muss durch die Zeichenfolge '<' ersetzt werden.
- '>' muss durch die Zeichenfolge '>' ersetzt werden.

Eine vollständige Zusammenfassung der Zeichenkodierung mit allen möglichen Codierungsformen pro Zeichen ist in Anhang B enthalten. Ein INTERLIS 2 Schreibprogramm, kann bei mehreren Codierungsformen pro Zeichen eine geeignete Form frei auswählen. Ein INTERLIS 2-Leseprogramm muss *alle* Codierungsformen erkennen können. Bemerkung: Mehrere Codierungsformen pro Zeichen werden zugelassen um maximale Kompatibilität mit bestehenden XML-Werkzeugen zu erreichen.

3.3.3 Allgemeiner Aufbau der Transferdatei

Eine INTERLIS-Transferdatei ist gemäss folgender EBNF-Hauptregel aufgebaut:

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

Die Regel HeaderSection erzeugt den Vorspann der Transferdatei und die Regel DataSection den Datenbereich.

Eine durch die Transferregel erzeugte INTERLIS-Transferdatei ist immer auch eine gültige (well formed) XML 1.0-Transferdatei. In einer INTERLIS-Transferdatei können daher auch beliebig viele Kommentarzeilen der Form

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

an den in XML 1.0 dafür vorgesehenen Stellen vorkommen. Der Inhalt der Kommentarzeilen darf von der Transfersoftware jedoch nicht interpretiert werden. Für die Codierung der Zeichen der Transferdatei wird die UTF-8-Codierung verwendet. Es darf standardmässig jedoch nur der Zeichenvorrat gemäss Anhang B Zeichentabelle verwendet werden.

Die Daten werden als XML-Objekte übertragen. Die Tagnamen der XML-Objekte werden jeweils aus den Objektnamen im INTERLIS-Datenmodell abgeleitet. Für übersetzte Datenmodelle (TRANSLATION OF) bedeutet das, dass die Tagnamen in der übersetzten Sprache im Transfer vorhanden sind (es müssen allerdings zusätzliche Einträge in der Alias-Tabelle gemacht werden).

3.3.4 Vorspann

Der Vorspann ist wie folgt aufgebaut:

```
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 ).
```

Im HeaderSection-Element müssen folgende Werte (XML-Attribute) eingetragen werden:

- VERSION. Version der INTERLIS-Codierung (im Moment **2.2**).
- SENDER. Absender des Datensatzes.

Im Element Alias werden Einträge gemacht welche das polymorphe Lesen eines Datensatzes ermöglichen (siehe auch nächster Abschnitt). In Comment kann (optional) ein Kommentar angegeben werden, in dem der Transfer näher beschrieben wird.

3.3.4.1 Bedeutung und Inhalt der Alias-Tabelle

Das Element Alias in der HeaderSection ist eine spezielle Tabelle, welche einem Leseprogramm das Lesen von erweiterten Modellen ohne Kenntnis der Erweiterungen gestattet (so genanntes *polymorphes* Lesen). Da XML selbst keine Vererbung und damit auch keinen Polymorphismus kennt, wird die Alias-Tabelle benutzt, um die notwendigen Zusatzinformationen einem Leseprogramm zu übermitteln.

In Alias muss pro Datenmodell X, welches im Transfer vorkommt, eine Abbildungstabelle (Entries-Element) enthalten sein. In jeder Abbildungstabelle wird via die xxxENTRY-Einträge (TAGENTRY, VALENTRY, DELENTRY) angegeben, welche transferierbaren Objekte (d.h. die eigenen, bzw. Erweiterungen davon) in den Baskets des Datenmodell X vorkommen können (bzw. nicht vorkommen können bei DELENTRY Einträgen). Sind von einem Datenmodell X auch Übersetzungen (TRANSLATION OF) des Datenmodells X im Transfer vorhanden, müssen zusätzlich alle Tags des übersetzten Datenmodells in der Abbildungstabelle des Datenmodells X mit TAGENTRY bzw. VALENTRY eingetragen werden. Die Abbildungstabellen der einzelnen Datenmodelle müssen in der Alias-Tabelle so angeordnet sein, dass

Abbildungstabellen von Basismodellen vor den Abbildungstabellen der erweiterten bzw. übersetzten Modelle eingetragen werden. Die einzelnen Einträge der Alias-Tabelle haben folgende Bedeutung:

- TAGENTRY. Im FROM-Attribut wird der Tag gemäss Originalmodell eingetragen (z.B. 'Kanton.TKanton.K1') im TO-Attribut der Tag gemäss dem aktuell betrachteten Modell (z.B. 'Bund.TBund.B1'). Es müssen auch alle Tags gemäss dem aktuellen Modell eingetragen werden. In diesem Fall wird im FROM- bzw. TO-Tag der gleiche Wert eingetragen. Der TAGENTRY-Eintrag muss für konkrete Topics, Klassen, Strukturen, Beziehungen und Grafikdefinitionen bzw. transferierbare Views angegeben werden.
- VALENTRY. Im ATTR-Attribut wird der Name des Aufzählungsattributs angegeben (z.B. 'Kanton.TKanton.K1.Farbe'). Im FROM-Attribut wird der Wert gemäss Originalmodell eingetragen (z.B. 'rot.karmin') im TO-Attribut der Tag gemäss aktuellem Modell (z.B. 'rot'). Es müssen auch alle Werte gemäss dem aktuellen Modell eingetragen werden. In diesem Fall wird der gleiche Wert für den FROM- bzw. TO-Tag angegeben. Der VALENTRY-Eintrag muss für alle Aufzählungstypen angegeben werden.
- DELENTRY. Im TAG-Attribut wird der Tag angegeben, welcher aus der Sicht des aktuellen Modells nicht vorkommen, in Erweiterungen jedoch vorhanden sein kann. Der DELENTRY-Eintrag muss für konkrete Topics, Klassen, Strukturen, Beziehungen und Grafikdefinitionen bzw. transferierbare Views und für Attribute von Klassen, Strukturen, Beziehungen und transferierbaren Views angegeben werden. Falls eine ganze Klasse (bzw. Struktur, Beziehung oder transferierbarer View) aus der Sicht des aktuellen Modells nicht vorkommen kann, ist es zulässig nur die nicht vorhandene Klasse mit DELENTRY zu bezeichnen. Die Attribute der Klasse (bzw. Struktur, Beziehung oder transferierbarer View) müssen in diesem Fall mit DELENTRY nicht geliefert werden.
- Hinweis zu Beziehungen ohne eigene Identität: Die obigen Regeln gelten sinngemäss auch für Beziehungen ohne eigene Identität. So müssen z.B. die Werte eines Aufzählungsattributs zu der Klasse eingetragen werden, in welcher auch die Rolle eingetragen wird (d.h. unter Klasse.Rolle.Attibut).

Im folgenden Beispiel werden die Eigenschaften der Alias-Tabelle nochmals verdeutlicht bzw. kommentiert.

Der folgende Datenbeschrieb:

```
MODEL Bund =

  CLASS B (ABSTRACT) = ...;


  TOPIC TBund =

    CLASS B1 EXTENDS B =
      Farbe : (rot, gruen, blau);
    END B1;

  END TBund.

END Bund.


MODEL TBund = TRANSLATION OF Bund;

  CLASS TB (ABSTRACT) = ...;


  TOPIC TTBund =

    CLASS TB1 EXTENDS TB =
      TFarbe : (trot, tgruen, tblau);
```

```
      END TB1;

    END TTBund.

  END TBund.


  MODEL Kanton = IMPORTS Bund;


    TOPIC TKanton EXTENDS Bund.TBund =

      CLASS K (ABSTRACT) EXTENDS Bund.B = ...;

      CLASS K1 EXTENDS Bund.TBund.B1 =
        Farbe (EXTENDED): (rot (dunkel, karmin, hell));
        Txt: TEXT*40;
      END K1;

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

    END TKanton.

  END Kanton.


  MODEL Gemeinde = IMPORTS Kanton;


    TOPIC TGemeinde EXTENDS Kanton.TKanton =

    END TGemeinde.

  END Gemeinde.
```

führt zur folgenden Alias-Tabelle:

```
<ALIAS>

  <ENTRIES FOR="Bund">

    <!-- Eintraege gemaess eigenem Modell -->
    <TAGENTRY FROM="Bund.TBund" TO="Bund.TBund"></TAGENTRY>
    <TAGENTRY FROM="Bund.TBund.B1" TO="Bund.TBund.B1"></TAGENTRY>
    <VALENTRY ATTR="Bund.TBund.B1.Farbe" FROM="rot" TO="rot"></TAGENTRY>
    <VALENTRY ATTR="Bund.TBund.B1.Farbe" FROM="gruen" TO="gruen"></TAGENTRY>
    <VALENTRY ATTR="Bund.TBund.B1.Farbe" FROM="blau" TO="blau"></TAGENTRY>

    <!-- Eintraege fuer TBund (TRANSLATION OF) -->
    <TAGENTRY FROM="TBund.TTBund" TO="Bund.TBund"></TAGENTRY>
    <TAGENTRY FROM="TBund.TTBund.TB1" TO="Bund.TBund.B1"></TAGENTRY>
    <VALENTRY ATTR="TBund.TTBund.TB1.TFarbe" FROM="trot" TO="rot"></TAGENTRY>
    <VALENTRY ATTR="TBund.TTBund.TB1.TFarbe" FROM="tgruen" TO="gruen"></TAGENTRY>
    <VALENTRY ATTR="TBund.TTBund.TB1.TFarbe" FROM="tblau" TO="blau"></TAGENTRY>

    <!-- Eintraege gemaess Modell Kanton -->
    <TAGENTRY FROM="Kanton.TKanton" TO="Bund.TBund"> </TAGENTRY>
    <TAGENTRY FROM="Kanton.TKanton.K1" TO="Bund.TBund.B1"> </TAGENTRY>
    <DELENTRY TAG="Kanton.TKanton.K1.Txt"> </DELENTRY>
    <DELENTRY TAG="Kanton.TKanton.K2"> </DELENTRY>
    <VALENTRY ATTR="Kanton.TKanton.K1.Farbe" FROM="rot.dunkel" TO="rot"> </VALENTRY>
    <VALENTRY ATTR="Kanton.TKanton.K1.Farbe" FROM="rot.karmin" TO="rot"> </VALENTRY>
    <VALENTRY ATTR="Kanton.TKanton.K1.Farbe" FROM="rot.hell" TO="rot"> </VALENTRY>
    <VALENTRY ATTR="Kanton.TKanton.K1.Farbe" FROM="gruen" TO="gruen"> </VALENTRY>
    <VALENTRY ATTR="Kanton.TKanton.K1.Farbe" FROM="blau" TO="blau"> </VALENTRY>
```

```
      <!-- Eintraege gemaess Modell Gemeinde -->
      <TAGENTRY FROM="Gemeinde.TGemeinde" TO="Bund.TBund"> </TAGENTRY>
      <TAGENTRY FROM="Gemeinde.TGemeinde.K1" TO="Bund.TBund.B1"> </TAGENTRY>
      <DELENTRY TAG="Gemeinde.TGemeinde.K1.Txt"> </DELENTRY>
      <DELENTRY TAG="Gemeinde.TGemeinde.K2"> </DELENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.dunkel" TO="rot"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.karmin" TO="rot"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.hell" TO="rot"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="gruen" TO="gruen"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="blau" TO="blau"> </VALENTRY>

  </ENTRIES>

    <ENTRIES FOR="Kanton">

      <!-- Eintraege gemaess eigenem Modell -->
      <TAGENTRY FROM="Kanton.TKanton" TO="Kanton.TKanton"> </TAGENTRY>
      <TAGENTRY FROM="Kanton.TKanton.K1" TO="Kanton.TKanton.K1"> </TAGENTRY>
      <TAGENTRY FROM="Kanton.TKanton.K2" TO="Kanton.TKanton.K2"> </TAGENTRY>
      <VALENTRY ATTR="Kanton.TKanton.K1.Farbe"
                FROM="rot.dunkel" TO="rot.dunkel"> </VALENTRY>
      <VALENTRY ATTR="Kanton.TKanton.K1.Farbe"
                FROM="rot.karmin" TO="rot.karmin"> </VALENTRY>
      <VALENTRY ATTR="Kanton.TKanton.K1.Farbe"
                FROM="rot.hell" TO="rot.hell"> </VALENTRY>
      <VALENTRY ATTR="Kanton.TKanton.K1.Farbe"
                FROM="gruen" TO="gruen"> </VALENTRY>
      <VALENTRY ATTR="Kanton.TKanton.K1.Farbe"
                FROM="blau" TO="blau"> </VALENTRY>

      <!-- Eintraege gemaess Modell Gemeinde -->
      <TAGENTRY FROM="Gemeinde.TGemeinde" TO="Kanton.TKanton"> </TAGENTRY>
      <TAGENTRY FROM="Gemeinde.TGemeinde.K1" TO="Kanton.TKanton.K1"> </TAGENTRY>
      <TAGENTRY FROM="Gemeinde.TGemeinde.K2" TO="Kanton.TKanton.K2"> </TAGENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.dunkel" TO="rot.dunkel"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.karmin" TO="rot.karmin"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.hell" TO="rot.hell"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="gruen" TO="gruen"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="blau" TO="blau"> </VALENTRY>

    </ENTRIES>

    <ENTRIES FOR="Gemeinde">

      <!-- Eintraege eigenem Modell -->
      <TAGENTRY FROM="Gemeinde.TGemeinde" TO="Gemeinde.TGemeinde"> </TAGENTRY>
      <TAGENTRY FROM="Gemeinde.TGemeinde.K1" TO="Gemeinde.TGemeinde.K1"> </TAGENTRY>
      <TAGENTRY FROM="Gemeinde.TGemeinde.K2" TO="Gemeinde.TGemeinde.K2"> </TAGENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.dunkel" TO="rot.dunkel"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.karmin" TO="rot.karmin"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
                FROM="rot.hell" TO="rot.hell"> </VALENTRY>
      <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
```

```
            FROM="gruen" TO="gruen"> </VALENTRY>
    <VALENTRY ATTR="Gemeinde.TGemeinde.K1.Farbe"
            FROM="blau" TO="blau"> </VALENTRY>

  </ENTRIES>

</ALIAS>
```

Ein Leseprogramm, welches für das Bundesmodell geschrieben, bzw. konfiguriert wurde, kann nun Bund, TBund, Kantons bzw. Gemeindedaten wie folgt lesen:

- Alle Tags aus Bund werden auf sich selbst abgebildet (z.B. Bund.TBund nach Bund.TBund).
- Alle Tags von TBund (TRANSLATION OF) werde auf ihr Gegenstück in Bund abgebildet (z.B. TBund.TTBund nach Bund.TBund).
- Ein XML-Objekt <Kanton.TKanton.K1> wird über den Tagentry-Eintrag nach <Bund.TBund.B1> abgebildet. Das Objekt <Bund.TBund.B1> ist dem Leseprogramm für das Bundesmodell bekannt, so dass es das Objekt entsprechend interpretieren kann.
- Ein XML-Objekt <Gemeinde.TGemeinde.K1> wird über den Tagentry-Eintrag nach <Bund.TBund.B1> abgebildet. Das Objekt <Bund.TBund.B1> ist dem Leseprogramm für das Bundesmodell bekannt, so dass es das Objekt entsprechend interpretieren kann.
- Der Wert "rot.karmin" des Aufzählungsattributs Kanton.TKanton.K1.Farbe bzw. Gemeinde.TGemeinde.Farbe wird über den Valentry-Eintrag nach "rot" abgebildet. Der Wert "rot" ist ein gültiger Wert gemäss Bundesmodell.
- Die abstrakte Klasse Kanton.TKanton.K bzw. Gemeinde.TGemeinde.K muss nicht als Tagentry eingetragen werden, da im Datensatz keine Instanzen Kanton.TKanton.K bzw. Gemeinde.TGemeinde.K vorkommen können.
- Das Attribut <Kanton.TKanton.K1.Txt> bzw. <Gemeinde.TGemeinde.K1.Txt> muss ignoriert werden, da dieses Attribut im Bundesmodell nicht existiert.
- Die Klasse <Kanton.TKanton.K2> bzw. <Gemeinde.TGemeinde.K2> muss ignoriert werden, da Instanzen von <Kanton.TKanton.K2> bzw. <Gemeinde.TGemeinde.K2> aus der Sicht Bundesmodell nicht existieren. Bemerkung: Das Attribut <Kanton.TKanton.K2.Txt> bzw. <Gemeinde.TGemeinde.K2.Txt> muss nicht speziell mit Delentry eingetragen werden, da ja die ganze Klasse aus der Sicht Bundesmodell nicht existiert.

Bemerkungen:

- Die Alias-Tabelle kann mit dem INTERLIS 2-Compiler generiert werden.
- Für jedes im Datensatz enthaltene Datenmodell (inkl. aller Basismodelle) muss ein Entry-Element eingetragen werden. Der Namen des Modells muss im XML-Attribut FOR eingetragen werden.
- Tagnamen, welche nach der Abbildung über die Alias-Tabelle zu keinem bekannten Tagnamen aus der Sicht des zu lesenden Modells führen, müssen vom Leseprogramm als Fehler ausgegeben werden.
- Attributwerte, welche nach der Abbildung über die Alias-Tabelle zu keinem bekannten Wert aus der Sicht des zu lesenden Modells führen, müssen vom Leseprogramm als Fehler ausgegeben werden.

3.3.5 Datenbereich

Der Datenbereich ist wie folgt aufgebaut:

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

3.3.6 Codierung von Themen

Behälter sind Instanzen eines konkreten TOPIC bzw. VIEW TOPIC. Behälter werden wie folgt codiert:

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

Der Wert %Model.Topic% muss für jedes konkrete Topic entsprechend substituiert werden (z.B. Grunddatensatz.Fixpunkte). Die XML-Attribute des Behälters haben folgende Bedeutung:

- BID. In BID muss die Behälteridentifikation eingetragen werden. Die Behälteridentifikation muss bei inkrementeller Nachlieferung eine OID sein.
- TOPICS. In TOPICS werden alle, ausser dem Basistopic, effektiv im Behälter vorkommenden Topics über eine kommaseparierte Liste angegeben (z.B.: "Kanton1.Topic1,Kanton2.Topic2"). Die angegebenen Topics müssen Erweiterungen des (allenfalls über mehrere Stufen geerbten) gemeinsamen Basistopic %Model.Topic% sein.
- KIND. Transferart (mögliche Werte: FULL, UPDATE, INITIAL). Falls das Attribut fehlt, wird FULL angenommen.
- STARTSTATE. Anfangszustand des Behälters vor dem Transfer (nur bei inkrementeller Nachlieferung).
- ENDSTATE. Endzustand des Behälters nach dem Transfer (nur bei inkrementeller Nachlieferung).

3.3.7 Codierung von Klassen

Die Objektinstanzen einer konkreten Klasse werden wie folgt codiert:

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

Der Wert %Model.Topic.Class% muss für jede konkrete Klasse entsprechend substituiert werden (z.B. Grunddatensatz.Fixpunkte.LFP). Jede Klasse - und damit jede Objektinstanz - erhält zusätzlich zu den im Modell definierten Attributen implizit eine Transferidentifikation (XML-Attribut TID). In der Transferart 'FULL' müssen alle TID's inkl. alle BID's innerhalb des gesamten Transfers eindeutig sein. In der Transferart INITIAL oder UPDATE müssen die TID's und BID's OID's sein. In BID wird die Behälteridentifikation angegeben, in welchem das Objekt ursprünglich erzeugt wurde (Originalbehälter). Falls sich das Objekt in seinem Originalbehälter befindet, kann BID weggelassen werden. Zudem erhält jedes Objekt in den Transferarten INITIAL und UPDATE ein Attribut für die Nachlieferungsoperation (XML-Attribut OPERATION). Das XML-Attribut OPERATION kann die Werte INSERT, UPDATE oder DELETE annehmen. Ohne Angabe von OPERATION wird der Wert INSERT angenommen.

Parameter werden nicht übertragen mit einer Ausnahme, wie sie im Kapitel 3.3.11 Codierung von Grafikdefinitionen angegeben ist. Die Reihenfolge der einzelnen Attribute im Transfer ist gleich der Definitionsreihenfolge der Attribute in der entsprechenden Klasse. Falls Klassen Erweiterungen von Basisklassen sind, folgen die Attribute der erweiterten Klassen im Transfer nach den Attributen der Basisklassen.

3.3.8 Codierung von Sichten

Zur Codierung von Sichten siehe Kapitel 3.2.4 Transferierbare Objekte. Es werden die XML-Attribute TID und BID, nicht aber OPERATION übermittelt. Als Attribute des Sichtobjekts werden nur diejenigen Attribute übertragen, welche in der Sicht explizit unter ATTRIBUTE bzw. implizit mit ALL OF angegeben wurden.

3.3.9 Codierung von Beziehungen mit eigener Identität

Objektinstanzen von konkreten Beziehungen mit eigener Identität (siehe auch Kapitel 2.7.1) werden wie Objektinstanzen von Klassen übertragen. Hinweis: Für Beziehungen ohne expliziten Namen wird der (Klassen)Name durch zusammenhängen der einzelnen Rollennamen gebildet (d.h. z.B. %RoleName1RoleName2%).

Rollen werden wie Attribute behandelt. Die Rollen selbst werden wie folgt codiert:

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

Zeigt die Referenz auf ein Objekt im gleichen Behälter wird die Referenz mit REF kodiert. In REF wird dabei die Transferidentifikation des referenzierten Objekts eingetragen.

Zeigt die Referenz auf ein Objekt in einem anderen Behälter (im gleichen Transfer oder sogar ausserhalb), wird die Referenz mit EXTREF und BID kodiert. In EXTREF wird dabei die Transferidentifikation bzw. in BID die Behälteridentifikation des referenzierten Objekts eingetragen.

In geordneten Beziehungen verweist das Attribut NEXT_TID auf das nächste Beziehungsobjekt.

3.3.10 Codierung von Beziehungen ohne eigene Identität

Objektinstanzen von konkreten (Zweier-)Beziehungen ohne eigene Identität (siehe auch Kapitel 2.7.1) werden als Unterstrukturen des einen der beiden beteiligten Objekte kodiert. Die Unterstruktur hat folgenden Aufbau:

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

Für %RoleName% muss der Name der Rolle angegeben werden welche auf das Hauptobjekt verweist (die andere Rolle wird nicht codiert). In StructureValue werden allfällige Attribute der Beziehung codiert. Die XML-Attribute REF, EXTREF, NEXT_TID und BID haben die gleiche Bedeutung wie bei Beziehungen mit eigener Identität. Bei 1-1 Beziehungen ist für %RoleName% die zweite Rolle zu nehmen.

3.3.11 Codierung von Grafikdefinitionen

Für jede Grafikdefinition werden im Transfer die von der Grafikdefinition referenzierten Signaturklassen (Sign-ClassRef) übertragen. Die Objektinstanzen der Signaturklassen werden durch das Ausführen der Grafikdefinitionen auf einem konkreten Inputdatensatz erzeugt. Parameter werden dabei wie Attribute codiert.

3.3.12 Codierung von Attributen

3.3.12.1 Allgemeine Regeln

Jedes Attribut einer Objektinstanz (einschliesslich komplexer Attribute wie POINT, POLYLINE, SURFACE, AREA, STRUCTURE, LIST OF, BAG OF, etc.) wird wie folgt codiert:

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

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

Bei undefiniertem Attributwert wird das Attribut nicht übertragen. Die Masseinheit des Attributwerts wird nicht codiert. Beispiel für ein einfaches Attribut:

```
<Nummer>12345</Nummer>
```

3.3.12.2 Codierung von Zeichenkette, URI und NAME

Attribute vom Basistyp TEXT*N werden wie folgt codiert:

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

3.3.12.3 Codierung von Aufzählungen

Aufzählungen werden wie folgt codiert:

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

Für die Codierung von Aufzählungen wird die Syntax für Aufzählungskonstanten angewendet (Regel EnumValue). Das Zeichen # wird dabei weggelassen. Die vordefinierten Textausrichtungstypen HALIGNMENT und VALIGNMENT werden wie Aufzählungen codiert. Ebenso wird der Typ BOOLEAN wie eine Aufzählung übertragen.

3.3.12.4 Codierung von numerischen Datentypen

Numerische Werte werden wie folgt codiert:

```
NumericValue = NumericConst.
```

Bemerkung: Bei ganzen Zahlen sind keine führenden Nullen erlaubt (007 wird als 7 transferiert). Bei reellen Zahlen ist maximal eine führende 0 erlaubt (z.B. nicht 00.07 sondern 0.07). Float-Zahlen können in verschiedenen Darstellungen übertragen werden (mit oder ohne Mantisse). Wesentlich ist lediglich, dass der Wertebereich der Float-Zahl zu ihrer Deklaration passt. Damit kann z.B. 100 als 10.0e1 oder 1.0e2 übertragen werden.

3.3.12.5 Codierung von strukturierten Wertebereichen

Strukturierte Wertebereiche werden wie strukturierte Konstanten codiert:

```
StructDecValue = StructDec.
```

3.3.12.6 Codierung von BASKET

Attributwerte vom Typ BASKET werden wie folgt codiert:

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

Die einzelnen XML-Attribute haben folgende Bedeutung:

- TOPIC. Bezeichnung des Themas in der Form Model.Topic.

- KIND. Art des Behälters (mögliche Werte: DATA, VIEW, BASE und GRAPHIC).
- BID. Behälteridentifikation.

3.3.12.7 Codierung von CLASS

Attribute vom Typ CLASS werden wie folgt codiert:

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

Der XML-String enthält den vollständig qualifizierten Klassennamen (z.B. Grunddatensatz.Fixpunkte.LFP).

3.3.12.8 Codierung von STRUCTURE

Attributwerte vom Typ STRUCTURE werden wie folgt codiert:

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

StructureName wird für Strukturen auf Niveau Modell als Model.StructureName und für Strukturen auf Niveau Thema als Model.Topic.StructureName gebildet.

3.3.12.9 Codierung von BAG OF und LIST OF

Attributwerte vom Typ BAG OF bzw. LIST OF werden wie folgt codiert:

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

Die Reihenfolge der ListValue-Elemente darf beim Transfer nicht verändert werden.

3.3.12.10 Codierung von Koordinaten

Attributwerte vom Typ COORD werden wie folgt codiert:

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

Die einzelnen XML-Unterobjekte müssen wie folgt gefüllt werden:

- C1. Erste Komponente der Koordinate (codiert wie numerischer Wert).
- C2. Zweite Komponente der Koordinate (nur bei 2D- und 3D-Koordinaten, codiert wie numerischer Wert).
- C3. Dritte Komponente der Koordinate (nur bei 3D-Koordinaten, codiert wie numerischer Wert).

3.3.12.11 Codierung von POLYLINE

Attributwerte vom Typ POLYLINE werden wie folgt codiert:

```
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 ).
```

Gerade Kurvenstücke eines Linienzugs werden gemäss Regel LineSegment codiert, für kreisbogenförmige Segmente gilt die Regel ArcSegment. Mit LINE FORM definierte Liniensegmente werden wie eine Struktur (LineStructure) kodiert.

Bemerkungen: Für Kreisbogensegmente (Regel ArcSegment) wird der Radius (optionales XML-Attribut R) redundant zur Zwischenpunktkoordinate (A1/A2) übermittelt. Der Zwischenpunkt eines Kreisbogens ist nur für die Lage von Bedeutung. Seine Höhe muss zwischen Anfangs- und Endpunkt linear interpoliert werden. Falls der Kreisbogen im Uhrzeigersinn (vom Startpunkt zum Endpunkt) definiert ist, hat der Radius ein positives Vorzeichen, sonst ein negatives. Bei Differenzen zwischen Radius und Koordinatenwerten gilt der Radius (vgl. Kapitel 2.8.11.2 Linienzug mit Strecken und Kreisbogen als vordefinierte Kurvenstücke). Die Stützpunkthöhe (C3) muss nur bei 3D-Polylines übertragen werden.

3.3.12.12 Codierung von SURFACE und AREA

SURFACE und AREA werden wie folgt codiert:

```
  SurfaceValue = TAG ( SURFACE )
                   OuterBoundary
                   { InnerBoundary }
                 ETAG ( SURFACE ).

  OuterBoundary = Boundary.

  InnerBoundary = Boundary.

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

Flächen werden als Folge von Rändern (Boundaries) übertragen. Ein Rand ist eine Folge von Randlinien, wobei jeweils die nächste Randlinie mit dem Endpunkt der vorhergehenden Randlinie beginnt. Der Endpunkt der letzten Randlinie ist identisch mit dem Anfangspunkt der ersten Randlinie. Die Randlinien bilden also zusammen einen geschlossenen Linienzug (Polygon).

Der erste Rand einer Fläche (OuterBoundary) ist der äussere Rand der Fläche. Die allenfalls folgenden inneren Ränder (InnerBoundary) der Fläche begrenzen die Inseln der Fläche. Die inneren Ränder müssen geometrisch vollständig innerhalb des äusseren Rands liegen. Die einzelnen Ränder einer Fläche dürfen sich gegenseitig nicht überschneiden.

Falls die SURFACE oder AREA mit Linienattributen definiert wurde, muss das Strukturelement des Linienattributs mit jedem PolylineValue übertragen werden (Regel LineAttr in PolylineValue).

Bei der Gebietseinteilung (AREA) müssen alle Randlinien der Fläche deckungsgleich mit den Randlinien der Nachbarfläche(n) sein, sofern sie nicht zum Perimeter des Flächennetzes gehören. Zwei Randlinien sind identisch, wenn für jeden Abschnitt der Randlinie alle Stützpunkte mit dem entsprechenden Abschnitt der Nachbarfläche identisch sind. Bei Kreisbogenstützpunkten darf lediglich das Vorzeichen des Kreisbogenradius verschieden sein. Falls für das Flächennetz Linienattribute definiert wurden, müssen die Linienattributwerte für Randlinien jeweils paarweise identisch sein.

3.3.12.13 Codierung von Referenzen

Attribute vom Typ REFERENCE TO werden wie folgt codiert:

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

Die XML-Attribute REF, EXTREF und BID habe die gleiche Bedeutung wie bei eigentlichen Beziehungen.

3.3.12.14 Codierung von METAOBJECT und METAOBJECT OF

Attribute vom Typ METAOBJECT (vgl. die entsprechende Klasse in Anhang A Das interne INTERLIS-Datenmodell) werden gemäss Kapitel 3.3.12.9 Codierung von BAG OF und LIST OF codiert. Parameter vom Typ METAOBJECT (Syntaxregel ParameterDef) werden jedoch *nicht* übermittelt. Parameter vom Typ METAOBJECT OF werden wie Attribute vom Typ NAME übertragen.

3.4 Verwendung von XML-Werkzeugen

Da der INTERLIS 2-Transfer vollständig auf XML 1.0 beruht, können für die Bearbeitung (bzw. Analyse) von INTERLIS-Objekten grundsätzlich INTERLIS- oder XML-Werkzeuge eingesetzt werden. Es müssen jedoch folgende Unterschiede beachtet werden:

- INTERLIS 2-Werkzeuge kennen neben dem XML-Datensatz auch die zugehörigen INTERLIS-Datenmodelle. Ein INTERLIS-Prüfwerkzeug kann daher z.B. im Allgemeinen schärfere Tests auf den Daten durchführen als dies einem reinen XML-Werkzeug möglich wäre.
- INTERLIS 2-Werkzeuge kennen die speziellen INTERLIS-Datentypen wie z.B. COORD, POLYLINE, SURFACE etc. Ein INTERLIS 2-Browser wird daher Datensätze auch grafisch darstellen können. Ein reiner XML-Browser kann hingegen nur den Aufbau des Dokuments visualisieren.
- INTERLIS 2-Werkzeuge unterstützen das polymorphe Lesen von Daten über die Alias-Tabelle. Damit kann ein INTERLIS-Importprogramm Daten eines erweiterten Modells wie Daten des Basismodells lesen. Mit einem reinen XML-Werkzeug ist polymorphes Lesen jedoch nicht ohne weiteres möglich.
- INTERLIS 2-Werkzeuge können ausserdem die Übersetzung von Schema-Namen über die Alias-Tabelle unterstützen. Auch dies ist mit reinen XML-Werkzeugen nicht ohne weiteres möglich.

Trotz dieser Unterschiede können allgemeine XML-Werkzeuge für viele Zwecke direkt eingesetzt werden. Dazu zählen zum Beispiel die Filterung von Datensätzen, das Editieren von Datensätzen mit XML-Editoren, die Prüfung der Transferdatensätze, die Übersetzung in andere Formate, etc.

Anhang A (normativ) Das interne INTERLIS-Datenmodell

Im Folgenden ist das gesamte interne INTERLIS 2-Datenmodell nochmals zusammenfassend abgebildet. Dieses kann auch durch einen eigenen Aufruf des INTERLIS 2-Compilers erzeugt werden. Wenn die verschiedenen Elemente des Modells dem INTERLIS 2-Compiler bereits bekannt sind, ist es nicht kompilierbar, sondern dient nur der Illustration. Dem von KOGIS zur Verfügung gestellten INTERLIS 2-Compiler sind die Elemente des Modells bereits bekannt.

```
!! 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: STRUCTURE;
  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.
```

Anhang B (normativ für CH) Zeichentabelle

Mit der hier aufgeführten Tabelle werden alle standardmässig in INTERLIS 2 verfügbaren Zeichen, Sonderzeichen, Umlaute und diakritische Zeichen und ihre Codierung in einem INTERLIS Transfer angegeben. Für einige Zeichen stehen mehrere Codierungsformen zur Verfügung. In diesem Fall sind alle möglichen Codierungsformen des Zeichens angegeben. Bei mehreren Codierungsmöglichkeiten pro Zeichen kann ein INTERLIS 2 Schreibprogramm eine der möglichen Codierungsformen frei auswählen. Ein INTERLIS 2-Leseprogramm muss alle möglichen Codierungsformen des Zeichens erkennen.

Die Tabelle gilt nur für XML-Content (d.h. XML-String bzw. XML-Value). XML-Tags werden ausschliesslich als ASCII-codierte Zeichen gemäss Syntax aus Kapitel 3 übertragen.

Neben den in der Tabelle enthaltenen Standardzeichen können auch weitere Zeichen in INTERLIS 2 Anwendungen benutzt werden. Dazu muss jedoch immer ein Kontrakt zwischen den beteiligten Parteien vereinbart werden.

UCS Hex	UCS Dez	UTF-8 Codierung Octet Hex	XML Codierung Character Reference Dez	XML Codierung Character Reference Hex	XML Codierung Entity Reference	Darstellung
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				=
003E	62	3E	&#62;	&#x3e;	&gt;	>
003F	63	3F				?

UCS Hex	UCS Dez	UTF-8 Codierung Octet Hex	XML Codierung Character Reference Dez	XML Codierung Character Reference Hex	XML Codierung Entity Reference	Darstellung
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
0073	115	73				s

UCS Hex	UCS Dez	UTF-8 Codierung Octet Hex	XML Codierung Character Reference Dez	XML Codierung Character Reference Hex	XML Codierung Entity Reference	Darstellung
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	&#196;	&#xc4;		Ä
00C6	198	C3 86	&#198;	&#xc6;		Æ
00C7	199	C3 87	&#199;	&#xc7;		Ç
00C8	200	C3 88	&#200;	&#xc8;		È
00C9	201	C3 89	&#201;	&#xc9;		É
00D1	209	C3 91	&#209;	&#xd1;		Ñ
00D6	214	C3 96	&#214;	&#xd6;		Ö
00DC	220	C3 9C	&#220;	&#xdc;		Ü
00E0	224	C3 A0	&#224;	&#xe0;		à
00E1	225	C3 A1	&#225;	&#xe1;		á
00E2	226	C3 A2	&#226;	&#xe2;		â
00E4	228	C3 A4	&#228;	&#xe4;		ä
00E6	230	C3 A6	&#230;	&#xe6;		æ
00E7	231	C3 A7	&#231;	&#xe7;		ç
00E8	232	C3 A8	&#232;	&#xe8;		è
00E9	233	C3 A9	&#233;	&#xe9;		é
00EA	234	C3 AA	&#234;	&#xea;		ê
00EB	235	C3 AB	&#235;	&#xeb;		ë
00EC	236	C3 AC	&#236;	&#xec;		ì
00ED	237	C3 AD	&#237;	&#xed;		í
00EE	238	C3 AE	&#238;	&#xee;		î
00EF	239	C3 AF	&#239;	&#xef;		ï
00F1	241	C3 B1	&#241;	&#xf1;		ñ
00F2	242	C3 B2	&#242;	&#xf2;		ò
00F3	243	C3 B3	&#243;	&#xf3;		ó
00F4	244	C3 B4	&#244;	&#xf4;		ô
00F5	245	C3 B5	&#245;	&#xf5;		õ
00F6	246	C3 B6	&#246;	&#xf6;		ö
00F9	249	C3 B9	&#249;	&#xf9;		ù
00FA	250	C3 BA	&#250;	&#xfa;		ú
00FB	251	C3 BB	&#251;	&#xfb;		û
00FC	252	C3 BC	&#252;	&#xfc;		ü

Tabelle 2: In INTERLIS 2 zugelassene UCS/Unicode-Zeichen und deren Codierung.

Bemerkungen:

- In den Kolonnen UCS/Hex bzw. UCS/Dezimal ist der UCS (bzw. Unicode) Code des Zeichens angegeben (hexadezimal bzw. dezimal).
- In der Kolonne UTF-8 Codierung ist die Codierung des Zeichens nach UTF-8 als 8bit Bytes (Octet) in hexadezimaler Schreibweise angegeben. Die Zeichen >= Hex 80 werden als Mehrbytefolgen codiert. Bemerkung: Die hexadezimale Schreibweise wird hier nur für die Erläuterung der Codie-

rung benutzt. Auf dem Transfer werden nur die binär codierten Octete übertragen.

- In der Kolonnen XML Codierung Character Reference (Dez) bzw. Character Reference (Hex) ist die Codierung des Zeichens als XML Character Reference angegeben (dezimale bzw. hexadezimale Variante). Der Wert ist eine ASCII-Zeichenfolge und muss genau so im Transfer verwendet werden. Falls möglich, sollte ein Schreibprogramm die Character Reference Codierung für das Zeichen wählen. Die Character Reference Codierung hat den Vorteil, dass sie auf allen Plattformen (Unix, PC, etc.) bzw. mit einfachen ASCII-Editoren angezeigt bzw. editiert werden kann. Bemerkung: Bei der hexadezimalen Codierungsvariante können die Buchstaben a-f in Gross- oder Kleinschrift angegeben werden (das x in der hexadezimalen Variante muss jedoch immer klein sein).
- Die XML Codierung (Entity Reference) ist nur für wenige Spezialzeichen erlaubt. Der Wert ist eine ASCII-Zeichenfolge, welche genau so im Transfer verwendet werden muss. Bemerkung: XML-Entities wie ü (für Zeichen ü) sind in einer INTERLIS 2-Transferdatei nicht erlaubt, weil von einer INTERLIS 2-Transferdatei kein DTD referenziert wird (die benutzbaren Entities wie z.B.: & sind durch die XML 1.0 Spezifikation vordefiniert).
- In der Kolonne Darstellung ist die Darstellung des Zeichens in einem UCS- bzw. Unicode kompatiblen Editor angegeben.

Anhang C (informativ) Das kleine Beispiel Roads

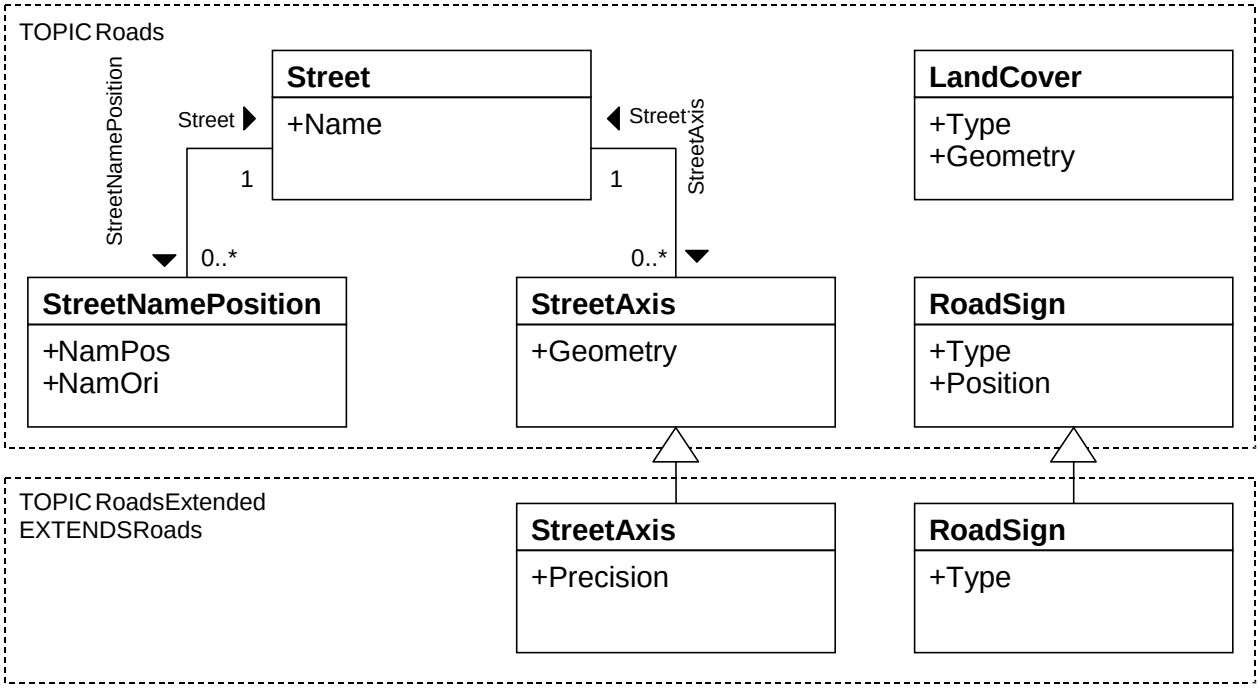
Einleitung

Zum leichteren Einstieg in INTERLIS 2 ist hier ein kleines aber vollständiges Beispiel zusammengestellt. Dieses Beispiel beschreibt einen Datensatz, der für einen vollständigen Datentransfer (FULL) ausgelegt worden ist. Für Anwendungsbeispiele mit inkrementeller Nachlieferung (inkl. OID) werden entsprechende Beispieldatensätze und Benutzerhandbücher zur Verfügung gestellt.

Das Beispiel besteht aus folgenden Teilen:

- Den Datenmodellen RoadsExdm2ben_10 und RoadsExdm2ien_10.
- Dem XML-Datensatz RoadsExdm2ien (Datei RoadsExdm2ien.xml) welcher Objekte gemäss dem Datenmodell RoadsExdm2ien_10 enthält.
- Dem Grafikmodell RoadsExgm2ien_10. Im Grafikmodell ist eine mögliche Darstellung zum Datenmodell RoadsExdm2ien_10 definiert (Bemerkung: Zum gleichen Datenmodell sind beliebig viele Darstellungen möglich).
- Einer Sammlung von Signaturobjekten (Signaturenbibliothek) in RoadsExgm2ien_Symbols (Datei RoadsExgm2ien_Symbols.xml). Die Signaturenbibliothek ist ein XML-Datensatz gemäss dem Signaturenmodell StandardSymbology (siehe auch Anhang K Signaturenmodelle). Die Signaturenbibliothek wird im Grafikmodell RoadsExgm2ien_10 für die Darstellung der Beispieldaten aus dem RoadsExdm2ien.xml-Datensatz benutzt.

Der Name "RoadsExdm2ben_10" ist eine Abkürzung von "**Roads Ex**ample, **D**ata **M**odel, INTERLIS **2**, **B**asic Model, **en**glish, Release **_10**". Die einzelnen Teile sind im Folgenden näher beschrieben.



Figur 26: UML-Klassendiagramm der Datenmodelle.

Datenmodelle RoadsExdm2ben_10 und RoadsExdm2ien_10

Im Datenmodell RoadsExdm2ben_10 sind die Objekte LandCover (Bodenbedeckungsflächen), StreetAxis (Strassenachsen), StreetName (Strassennamen) und PointObject (Punktobjekte) enthalten. Das Datenmodell RoadsExdm2ien_10 ist eine Erweiterung von RoadsExdm2ben_10. Die Datenmodelle sind im UML-Klassendiagramm (Figur 26) übersichtsmässig zusammengestellt.

Bemerkung: Das Beispiel ist bewusst sehr einfach gehalten und erhebt daher keinerlei Anspruch auf Vollständigkeit. Die zugehörigen in INTERLIS 2 beschriebenen Modelle lauten wie folgt:

```
!! 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
```

Datensatz RoadsExdm2ien gemäss Datenmodell RoadsExdm2ien_10

Nachfolgend ist ein Beispieldatensatz zum Datenmodell RoadsExdm2ien_10 angegeben. Die XML-Formatierung wurde über die Regeln gemäss Kapitel 3 Sequentieller Transfer aus dem Datenmodell RoadsExdm2ien_10 hergeleitet.

```
<?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>
```

Grafikbeschreibung RoadsExgm2ien_10

Zum Datenmodell wird eine Darstellung mit Hilfe der Grafikbeschreibung RoadsExgm2ien_10 definiert. Das Grafikmodell lautet wie folgt:

```
!! 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.
```

Das Grafikmodell RoadsExgm2ien_10 greift auf Signaturen in der Signaturenbibliothek RoadsExgm2ien_Symbols (Datei RoadsExgm2ien_Symbols.xml) zurück. Die Signaturenbibliothek ist im folgenden Abschnitt beschrieben.

Signaturenbibliothek RoadsExgm2ien_Symbols.xml

Nachfolgend ist die Signaturenbibliothek RoadsExgm2ien_Symbols als XML-Datensatz dargestellt (Datei RoadsExgm2ien_Symbols.xml). Die Signaturenbibliothek enthält Symboldefinitionen für Fixpunkte und Bäume, sowie Linien-, Textanschrift- und Flächensignaturen. Das zugehörige Signaturen-Datenmodell ist im Anhang K Signaturenmodelle (StandardSymbology) enthalten.

```
<?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>
```

Grafische Darstellung des Beispiels

Aus dem Datensatz RoadsExdm2ien (Datei RoadsExdm2ien.xml), den Beschreibungen im Grafikmodell RoadsExgm2ien_10 (Datei RoadsExgm2ien_10.ili) und der Signaturenbibliothek RoadsExgm2ien_Symbols (Datei RoadsExgm2ien_Symbols.xml) erzeugt ein INTERLIS 2-Grafikprozessor folgende Grafik:

Figur 27: Grafik, erzeugt aus den Daten- und Grafikbeschreibungen.

Anhang D (informativ) Index

Der Index zeigt die reservierten Wörter in Grossschrift (vgl. Kapitel 2.2.7 Sonderzeichen und reservierte Wörter), die Syntaxdefinitionen der Beschreibungssprache in Normalschrift (vgl. Kapitel 2) und die Syntaxdefinitionen des Transfers in Kursivschrift (vgl. Kapitel 3). Die fett angezeigte Seitenzahl nennt diejenige Stelle im Referenzhandbuch, an der der Begriff am umfassendsten definiert ist.

Anhang E (Standard-Erweiterungsvorschlag) Aufbau von Objektidentifikatoren (OID)

Vorbemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Einleitung

Mit der immer grösseren Verfügbarkeit von Geodaten wird vermehrt auch deren Nachführung (Fortführung) und Integration in verschiedene Datenbanken verlangt. Dies sind einige Gründe für den Bedarf nach einer einheitlichen Regelung von *Objektidentifikatoren* (OID): Ein OID identifiziert eine Objektinstanz von deren Entstehung bis zu ihrem Untergang auch wenn die Attributwerte sich ändern. Im Gegensatz zu den Benutzerschlüsseln (vgl. Anhang F *Eindeutigkeit von Benutzerschlüsseln* im INTERLIS 2-Referenzhandbuch) ist der OID vom Anwender als "nicht-sprechendes" ("opaques") Attribut zu betrachten, das typischerweise über Systemfunktionen verwaltet wird.

Ein OID muss zumindest innerhalb einer Transfergemeinschaft *eindeutig*, *einmalig* und *unveränderbar* sein.

An die Vergabe und die Nutzung von OID werden unter anderem folgende Anforderungen gestellt:

- Eindeutig (generell), einmalig und unveränderbar (stabil) - auch bei grossen Datenmengen.
- Unabhängig von Hardware- und Softwareproduzenten.
- Unabhängig von Plattformen.
- Im Mehrplatz- als auch im Einzelplatz-Betrieb, bzw. in autonomen Systemen nutzbar (z.B. im Felde).
- Wenig Platzbedarf und nach Bedarf optimierbar.
- Einfach implementierbar.

Weitere Anforderungen sind nicht unbedingt technischer Art, wie z.B.: möglichst wenig Organisationsaufwand, Vergabe unter eigener (nationaler) Kontrolle, nutzbar auch mit älteren Systemen und Akzeptanz bei den Systemherstellern. Dies sind anspruchsvolle, teilweise entgegen gesetzte Anforderungen. Eine spezielle Vorgabe ist, dass ein OID von einem Produzentensystem mindestens zehn Millionen Mal vergeben werden kann. Als weitere Vorgabe gelte, dass er eine feste Länge hat damit die Handhabung erleichtert wird (dies schliesst andere bekannte Verfahren, z.B. eine so genannte URI als Präfix aus). Eine Prüfziffer ist nicht vorzusehen: es wird angenommen, dass tiefere Kommunikationsebenen entsprechende Mittel bereitstellen.

Grundsätzlich wird die Eindeutigkeit eines OID immer über einen zentralen Mechanismus erreicht. Die zwei Extreme - d.h. die zentrale Vergabe jedes einzelnen OID auf der einen und die vollständig dezentrale, autonome Generierung von OID auf der anderen Seite - führen zu unbefriedigenden Lösungen. Einen OID, der z.B. über eine Netzwerkkartennummer und einen Zeitstempel vergeben wird, wird für nicht praktikabel und zukunftsweisend gehalten, zumal dann jedes Gerät eine Netzwerkkarte besitzen muss und nicht abzusehen ist, ob diese Technologie in den nächsten Jahren nicht durch andere Entwicklungen überholt wird.

Die Entstehung dieser Spezifikation hat eine lange Vorgeschichte. Es wurden über mehrere Jahre verschiedene Vernehmlassungen, Studien und Sitzungen durchgeführt. Ein Teil der dabei entstandenen Dokumente kann Interessierten abgegeben werden (Bestellung unter anderem auf www.interlis.ch, bzw. info@interlis.ch).

Aufbau des Objektidentifikators (OID)

Ein Objektidentifikator (OID) besteht aus einem Präfix- und einem Postfix-Anteil und hat als Gesamtheit eine Länge von 16 alphanumerischen Stellen in seiner darstellbaren Form. Der OID wird namentlich auf der Datenschnittstelle oder dem Anwender gegenüber immer als Einheit behandelt. Der OID-Wertebereich STANDARDOID des INTERLIS Modells entspricht dieser Definition. Er definiert aber nur die Gesamtlänge und nicht den detaillierten Aufbau.

```
OIDDef = Praefix Postfix.
```

Präfix

Das Präfix wird von einer zentralen Stelle vergeben. Damit wird die Eindeutigkeit innerhalb einer Transfergemeinschaft gegeben. Typischerweise ist für jeden Behälter (z.B. ein Datenbank-Prozess, der Daten eines konkreten Themas verwaltet) ein neues Präfix erforderlich. Als Bestimmungsort für das Präfix gilt das Land des Präfix-erzeugenden Prozesses. Dieser Prozess ist nicht unbedingt am gleichen Ort wie das Produzentensystem, das den gesamten OID erzeugt.

Das Präfix besteht aus einer Folge von 8 Zeichen mit folgendem zulässigen Zeichenvorrat:

```
Praefix = Letter { Letter | Digit }.  !! Folge von 8 Zeichen
Letter = ( 'A' | .. | 'Z' | 'a' | .. | 'z' ).
Digit = ( '0' | '1' | .. | '9' ).
```

Ein Präfix ist demnach definiert als eine Folge von Buchstaben und Ziffern, wobei das erste Zeichen ein Buchstabe sein muss (vgl. auch den Aufbau von XML-Tag-Namen oder Kapitel 2.2.2 Namen im INTERLIS 2-Referenzhandbuch).

Weiter gilt die Regelung, dass die ersten zwei Präfix-Stellen gemäss der ISO-Norm 3166 festgelegt werden. Für in Deutschland, in Österreich oder der Schweiz erzeugte Präfixe sind dort z.B. die Buchstaben "de", "oe" und "ch" festgelegt. Für die Erzeugung eines Präfixes stehen damit pro Stelle insgesamt 62 verschiedene Varianten zur Verfügung (0..9: ASCII 48 bis 57; A..Z: ASCII 65 bis 90; a..z: ASCII 97 bis 122). Die Kombination der 62 Zeichen mit der Anzahl Stellen ergibt eine Menge von OID's, die wahrscheinlich grösser ist, als die meisten Anwendungen jemals benötigen.

Postfix

Ein Postfix wird durch den Datenproduzenten, bzw. das Produzentensystem selber verwaltet. Er besteht seinerseits aus einer Folge von 8 Zeichen. Durch den ASCII-kompatiblen, kolonnenorientierten Ansatz wird verlangt, dass die allenfalls "freien" Stellen links mit Nullen ("0") besetzt werden (vgl. das erste und dritte Beispiel eines OID's unten). Der kleinstmögliche Ordinalwert des Postfix-Anteils wird damit als "0000000000" repräsentiert.

```
Postfix = { Letter | Digit }.  !! Folge von 8 Zeichen
```

Weitere Einschränkungen des Präfix- oder Postfix-Anteils sind bei Bedarf in zusätzlichen Spezifikationen zu regeln.

Zusammenfassung und Anwendungsbeispiele

OID	*Länge*	*Bedeutung*	*Bemerkungen*
Präfix	2 + 6 Char.	Länderkennung + ein von einer zuständigen, zentralen Stelle einmalig vergebener 'globaler' Identifikations-Anteil	Weltweit eindeutige Länderkennung, z.B. de (Deutschland), oe (Österreich), ch (Schweiz), entsprechend ISO-Norm 3166. Weitere Einschränkungen sind in zusätzlichen Spezifikationen zu regeln.
Postfix	8 Char.	Sequenz (numerisch oder alphanumerisch) des Produzentensystems als 'lokaler' Identifikations-Anteil	Weitere Einschränkungen, wie z.B. Zeitstempel mit Sequenznummer, sind in zusätzlichen Spezifikationen zu regeln.

Beispiele

```
1234567812345678    Bemerkung
----------------    ---------

A000000000000000    Theoretisch kleinstmöglicher OID

zwzzzzzzzzzzzzz    Grösstmöglicher OID, mit zw für Zimbabwe

deg5mQXX2000004a    Willkürlich gewählter OID aus Deutschland (de)

chgAAAAAAAAA0azD    Willkürlich gewählter OID aus der Schweiz (ch)
```

Organisation

Eine zentrale Stelle (etwa eine Bundesstelle), die von der Transfergemeinschaft anerkannt wird, unterhält einen zentralen Dienst für die Vergabe von OID's. Die Datenproduzenten können von dort einen oder mehrere Präfix-Anteile über geeignete Kommunikationskanäle beziehen. Dies könnte zum Beispiel eine Internet-Seite sein, die mit einem E-Mail-Dienst verbunden ist. Ein solcher Dienst kann relativ sicher gemacht und gegen Missbrauch geschützt werden.

Es ist den Implementationen in den Sender- und Empfängersystemen überlassen, die in dieser Spezifikation ausdrücklich genannten Eigenschaften auszunutzen und den OID z.B. zu Sortierzwecken zu verwenden oder um interne Optimierungen vorzunehmen. Eine solche Optimierung könnte z.B. darin bestehen, dass der Präfix-Anteil im System an einem zentralen Ort verwaltet wird und die verschiedenen Objekte nur den Postfix-Anteil enthalten und sich zusätzlich auf einen (gemeinsamen) Präfix-Anteil beziehen. Eine andere Sparmassnahme ist die systeminterne Speicherung des Postfix-Anteils in Binärcodierung.

Für Übungszwecke sind zu verwenden:

- bei OID's für Baskets der OID-Präfix "chB00000".
- für alle andern OID's der OID-Präfix "ch100000".

Anhang F (Standard-Erweiterungsvorschlag) Eindeutigkeit von Benutzerschlüsseln

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Modellierungs-Alternativen

Wird von Benutzerschlüsseln verlangt, dass sie eindeutig sind, stellt sich immer Frage, in welchem Rahmen die Eindeutigkeit gilt. Rein technisch gesehen ist es häufig klar, dass die Eindeutigkeit nur innerhalb eines bestimmten Behälters garantiert werden kann, weil auf die anderen Behälter gar kein Zugriff besteht. Vom Modellierungsstandpunkt aus ist der Behälter aber bedeutungslos, da nichts über seinen Umfang ausgesagt werden kann.

Zunächst einmal ist zu fragen, ob in einem Basismodell wirklich eine Eindeutigkeit verlangt werden muss. Aus Sicht der übergeordneten Stelle (z.B. Bund) ist es durchaus denkbar, dass die Eindeutigkeit nicht für alle gilt, sondern zum Beispiel nur für das (Bundes-) interne Datenmodell.

Im Weiteren werden zwei Varianten vorgestellt, die das gegebene Problem der eindeutigen Benutzerschlüssel lösen:

- Variante Zentrale Regelung.
- Variante Dezentrale Regelung (Delegationsprinzip).

Variante Zentrale Regelung

Ohne weitergehende Überlegungen wird eine zentrale Regelung im Vordergrund stehen. Dabei legt eine zentrale Stelle für alle Objekte einer Klasse fest, dass ein bestimmter Benutzerschlüssel über das gesamte Gebiet eindeutig sein muss. Dies kann mit organisatorischen Massnahmen geschehen oder alle Beteiligten greifen auf eine zentrale Datenbank zu.

```
TOPIC Grundeigentum =

  CLASS Parzelle =
    Nummer: 1 .. 99999;
    Geometrie: AREA WITH (STRAIGHTS, ARCS)
      VERTEX CHLKoord WITHOUT OVERLAPS > 0.005;
  UNIQUE
    Nummer;
  END Parzelle;

END Grundeigentum.
```

Oft legt die zentrale Stelle eine Gebietseinteilung fest, bei der alle Gebietsnummern über das gesamte Gebiet eindeutig sind. Sollen die Parzellen, die sich ja wiederum innerhalb dieser Gebiete befinden, über das Gesamte eindeutig sein, dann muss der Benutzerschlüssel aus der Kombination von Gebietsnummern und Parzellennummer bestehen:

```
CLASS Parzelle =
    Gebietsnummer: 1 .. 9999;
    Nummer: 1 .. 99999;
```

```
    Geometrie: AREA WITH (STRAIGHTS, ARCS)
      VERTEX CHLKoord WITHOUT OVERLAPS > 0.005;
  UNIQUE
    Gebietsnummer, Nummer; !! Benutzerschluessel
  END Parzelle;
```

Variante Dezentrale Regelung (Delegationsprinzip)

Wenn entsprechende Datenstrukturen in einem Datenmodell vorgesehen sind, ist es möglich, dass Objekte primär in kleineren Behältern (z.B. ein Behälter pro Gemeinde) erfasst werden, um sie dann ohne Probleme zu grösseren Behältern (z.B. einen für einen ganzen Kanton) zusammenzufassen.

Nimmt man weiter an, dass der Bund festlegt, dass Parzellennummern fünfstellige Zahlen sein sollen, aber offen lässt, in welchem Rahmen sie eindeutig sein sollen, während ein Kanton zusätzlich festlegt, dass sie im Rahmen einer Gemeinde eindeutig sein sollen, ist folgende Modellierung möglich:

```
MODEL Bund =


  TOPIC Grundeigentum =

    CLASS Parzelle =
      Nummer: 1 .. 99999;
      Geometrie: AREA WITH (STRAIGHTS, ARCS)
        VERTEX CHLKoord WITHOUT OVERLAPS > 0.005;
    END Parzelle;

  END Grundeigentum;

END Bund.


MODEL KantonA =

  IMPORTS Bund;


  TOPIC OrgStruktur =

    CLASS Gemeinde =
      Name: TEXT*30;
    UNIQUE
      Name;
    END Gemeinde;

  END OrgStruktur;


  TOPIC Grundeigentum EXTENDS Bund.Grundeigentum =
    DEPENDS ON OrgStruktur;

    ASSOCIATION GdeParzelle =
      Gemeinde -<> (EXTERNAL) OrgStruktur.Gemeinde;
      Parzelle -- Parzelle;
    END GdeParzelle;

    CONSTRAINT OF Parzelle =
      UNIQUE
        Nummer, Gemeinde;
    END;

  END Grundeigentum;

END KantonA.
```

Die Namen der Gemeinden müssen gemäss Definition im Rahmen aller Objekte der Klasse eindeutig sein. Es ist dabei irrelevant, ob diese Bedingung auf Grund der Aufteilung in konkrete Behälter wirklich überprüfbar ist. Sachlich gesehen gilt die Anforderung.

Um festzulegen, dass die Parzellennummer im Rahmen einer Gemeinde eindeutig sein muss, wird zwischen Parzelle und Gemeinde eine Beziehung und verlangt, dass die Kombination von Gemeinde und Nummer eindeutig ist. Dabei ist es wiederum nicht relevant, ob ein Behälter einen Teil einer Gemeinde, eine ganze Gemeinde oder mehrere Gemeinden umfasst. Vom Modellierungstandpunkt aus gilt die Anforderung.

Geht man z.B. davon aus, dass ein System die Parzellen einer bestimmten Gemeinde enthält, ist es durchaus möglich, dass bei den Parzellen systemintern auf die Beziehung zur Gemeinde verzichtet wird und diese nur für die Datenübergabe an andere Systeme beigefügt wird.

Anhang G (Standard-Erweiterungsvorschlag) Einheiten-Definitionen

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Das Typen-Modell

Das folgende Typen-Modell fasst die gebräuchlichsten Einheiten zusammen. Es erweitert die durch INTERLIS direkt definierten Einheiten (siehe dazu Anhang A Das interne INTERLIS-Datenmodell).

```
!! 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.
```

Beispiele

Siehe Kapitel 2.9.1 Basiseinheiten im INTERLIS 2-Referenzhandbuch.

Anhang H (Standard-Erweiterungsvorschlag) Zeit-Definitionen

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Das Zeit-Modell

```
!! 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.
```

Beispieldaten zum Zeit-Modell

Das folgende Beispiel passt zum vorhergehenden Zeit-Modell.

```
  <?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>
```

Beispiele

Einige grundlegende Beispiele für Zeitangaben sind in Kapitel 2.9 Einheiten im INTERLIS 2-Referenzhandbuch angegeben.

Bezug nehmend auf dieses Zeit-Modell und die vorhergehenden XML-Daten, sind hier zwei Anwendungsbeispiele angegeben:

Beispiel 1: In der folgenden Domain-Definition, wird ein Zeittyp beschrieben, der sich auf MEZ bezieht, dass heisst auf die Mitteleuropäische Zeit, jedoch ohne die Sommerzeit eines Landes zu berücksichtigen.

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

Beispiel 2: In der untenstehenden Domain-Definition, wird ein Zeittyp beschrieben, der sich auf MEZ inklusive der Schweizer Sommerzeit bezieht.

```
DOMAIN
  UhrZeit = 0:00:00.000 .. 23:59:59.999 [MESZ];
```

Anhang I (Standard-Erweiterungsvorschlag) Farben-Definitionen

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Einleitung

Diese Spezifikation begründet detailliert, dass sich ein $L^*C^*_{ab}h^*_{ab}$ genannter Farbraum am besten für Farben-Definitionen eignet. Es stellt diesen Farbraum ausführlich vor, zitiert Umrechnungsformeln zu anderen Farbräumen und gibt Hinweise darauf, wie eine Transformation von $L^*C^*_{ab}h^*_{ab}$-Koordinaten in das Farb-Koordinatensystem eines konkreten Bildschirms oder Druckers implementiert werden kann. Zudem begründet es die nachfolgend gewählten Wertebereiche und Genauigkeiten und gibt die Koordinaten ausgewählter Beispielwerte an.

Da es INTERLIS 2 unter anderem ermöglicht, Grafiken zu beschreiben, muss es möglich sein, Farben zu spezifizieren. Eine system- und geräteneutrale Definition von "Farbe" ist jedoch erstaunlich komplex und setzt das Verständnis von Konzepten voraus, die nicht allgemein bekannt sind.

Farbe ist ein Produkt aus Licht (= Farbreiz), Auge (= Farbvalenz) und Gehirnfunktion (= Sinneseindruck). Farben lassen sich eigentlich mit Zahlen nicht exakt beschreiben, sodass zwei Menschen darunter dasselbe verstehen. Es ist aber möglich, Farbwerte zu messen und sich auf eine Messung zu einigen, so dass eine präzise Verständigung unter Fachleuten möglich ist.

Ein Weg zur Spezifikation von Farben als Zeichenkette sollte mehreren Anforderungen genügen:

- **Geräteunabhängigkeit** — Es sollte klar definiert sein, welche Farbe mit einer bestimmten Angabe tatsächlich gemeint ist. Nur so kann sichergestellt werden, dass das Ergebnis auf allen Geräten den Erwartungen entspricht.
- **Ausdrucksstärke** — Es sollte möglich sein, alle Farben zu spezifizieren, die ein "normales" Gerät (insbesondere auch qualitativ gute Drucker und Plotter) darstellen kann. Das Spektrum spezifizierbarer Farben sollte also möglichst gross ein. Idealerweise umfasst es alle Farben, die ein Mensch wahrnehmen kann.
- **Intuitivität** — Beim Lesen einer Farbangabe sollte ein Mensch intuitiv eine ungefähre Vorstellung davon erhalten, was für eine Farbe gemeint ist. Ein INTERLIS-Modell besitzt immer auch Dokumentationscharakter und sollte für Menschen ohne grösseren Aufwand verständlich sein.
- **Systemneutralität** — Die Art und Weise der Farbangabe sollte kein System (GIS, Betriebssystem, Hardware) bevorzugen oder gar die Anschaffung besonderer Hilfsmittel bedingen.

Farbraum

Die untenstehende Tabelle fasst die Eignung verschiedener Farbräume für die Anwendung in INTERLIS-Darstellungsbeschreibungen zusammen:

Farbraum	Geräte-unabhängigkeit	Ausdrucks-stärke	Intuitive Verständlichkeit	Systemneutrali-tät
RGB	–	–	–	–
HLS	–	–	+ +	–
HSV	–	–	+ +	–
CMY(K)	–	–	–	–
XYZ	+	+	– –	+
SRGB	+	–	–	–
L a* b**	+	+	+	+
$L^* C^*_{ab} h^*_{ab}$	+	+	+ +	+

Figur I.1: Eignung verschiedener Farbräume für die Zwecke von INTERLIS.

Der letzte der in Figur I.1 erwähnten Farbräume $L^* C^*_{ab} h^*_{ab}$ (d.h. *L*a*b** mit Polarkoordinaten) erfüllt die oben postulierten Anforderungen am besten.

L*a*b*

In der grafischen Branche recht verbreitet ist der Farbraum *L*a*b** (manchmal auch CIELAB genannt), der über die in Figur I.2 beschriebene Transformation aus *XYZ* ableitbar ist.

$$L^* = 116 \cdot f\left(\frac{Y}{Y_n}\right) - 16$$

$$a^* = 500 \cdot \left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right]$$

$$b^* = 200 \cdot \left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right]$$

, wobei $f(x) = \begin{cases} \sqrt[3]{x} & \text{falls } x > 0.008856; \\ 7.787x + \frac{16}{116} & \text{sonst} \end{cases}$

Figur I.2: Die Umrechnung von XYZ zu L*a*b*.

In die Berechnung von Figur I.2 fliesst mit ‹X_n, Y_n, Z_n› ein "Referenzweiss" ein, um eine allfällige Färbung des Lichts zu kompensieren. Häufig werden die Werte von CIE-Standardlichtquellen (meist D50, gelegentlich D65) eingesetzt. Die XYZ-Koordinaten dieser Lichtquellen finden sich beispielsweise in [Sangwine/Horne, 1989], Tabelle 3.1.

Dieser Raum besitzt eine Reihe von nützlichen Eigenschaften:

- **Geräteunabhängigkeit** — *L*a*b** ist von *XYZ* abgeleitet und hängt damit nicht von einem bestimmten Gerät ab. Es ist eindeutig definiert, welche Farbe zu einem *L*a*b**-Tripel gehört.
- **Ausdrucksstärke** — Jeder Farbe, die eine reflektierende Fläche abgeben kann, ist in *L*a*b** ein Punkt zugeordnet.
- **Intuitive Verständlichkeit** — *L** ist die Helligkeit, wobei eine vollkommen schwarze Fläche (die keinerlei Licht reflektiert) ein *L** von 0 und ein perfekter Reflektor (der alles Licht reflektiert) ein *L**

von 100 besitzt. Ein menschlicher Betrachter empfindet eine Farbe mit *L** = 50 als mittlere Helligkeit. a* ist die Rot-Grün-Achse: Farben mit *a** = 0 werden als weder rot noch grün empfunden, Farben mit negativem *a** sind grün, Farben mit positivem *a** rot. Analog ist *b** die Blau-Gelb-Achse. Je weiter eine Farbe in der durch *a** und *b** aufgespannten Ebene vom Nullpunkt entfernt ist, umso gesättigter ist sie.

- **Systemneutralität** — *L*a*b** ist vollkommen systemneutral; als internationaler Standard ist der Farbraum unabhängig von einer bestimmten Firma.
- **Zunehmende Verbreitung** — *L*a*b** findet im professionellen Druck zunehmende Verbreitung. Programme wie Adobe Photoshop oder Acrobat (PDF) unterstützen den *L*a*b**-Farbraum.
- **Leichte Transformierbarkeit zu RGB** — *L*a*b**-Tripel sind durch Multiplikation mit einer 3x3-Matrix, gefolgt von einer Potenzierung (Gammakorrektur), die effizient mittels einer Tabelle erfolgen kann, in die RGB-Werte eines beliebigen Bildschirms transformierbar (vgl. [Adobe, 1992], Kapitel 23). Der Aufwand für Systemimplementierer ist somit sehr gering.
- **Gute Komprimierbarkeit** — Nur am Rande erwähnt sei, dass sich *L*a*b** besser als RGB für verlustbehaftete Verfahren zum Komprimieren von Bildern eignet. Im Zusammenhang mit INTERLIS ist dies jedoch irrelevant.

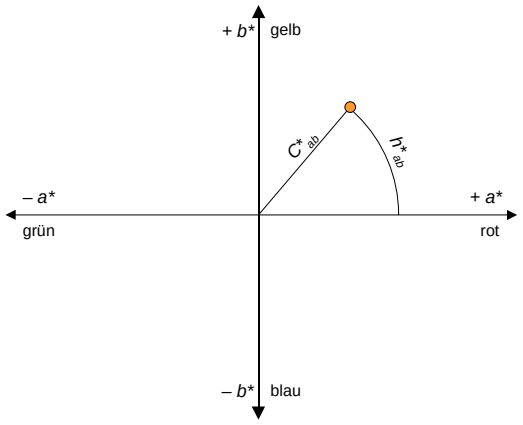
$$C^*_{ab} = \sqrt{\left(a^*\right)^2 + \left(b^*\right)^2} \qquad h^*_{ab} = \tan^{-1}\left(\frac{b^*}{a^*}\right)$$

Figur I.3: Umrechnung vom kartesischen L*a*b*-Raum in die polare Form L*C*$_{ab}$h*$_{ab}$ (nach [Sangwine/Horne, 1998]).

L*C*$_{ab}$h*$_{ab}$

Wie oben beschrieben, entsprechen im *L*a*b**-Raum die einzelnen Achsen *L** (dunkel — hell), *a** (grün — rot) und *b** (blau — gelb) unmittelbar wahrnehmbaren Eigenschaften der Farben.

Die *intuitive Verständlichkeit* kann jedoch noch gesteigert werden, wenn die Farbkoordinaten nicht über ein kartesisches, sondern über ein polares System angegeben werden (vgl. Figur I.4).



Figur I.4: Der Farbraum L*C*$_{ab}$h*$_{ab}$ arbeitet mit polaren Koordinaten auf L*a*b*.

Die Formel in Figur I.3 für h*$_{ab}$ trifft nur für positive a* und b* zu; eine korrekte Version würde Fallunterscheidungen für die einzelnen Quadranten enthalten. Dieses polare System verbindet die intuitive Verständlichkeit von HLS und HSV mit den oben beschriebenen zahlreichen Vorteilen von L*a*b*, da nun die

Achsen L* (Helligkeit, engl. *Luminance*), C* (Farbsättigung, engl. *Chroma*) und h* (Farbton, engl. *hue*) separat verfügbar sind.

Sind präzise Farben-Angaben gewünscht, dann sollen diese in INTERLIS-Modellen auf der Basis dieses Farb-Koordinatensystems gemacht werden.

Nötige Genauigkeit

Teil eines INTERLIS-Modells ist die Angabe, mit welcher Genauigkeit numerische Grössen festzuhalten sind. Nun ist der *L*a*b**-Raum so definiert, dass der Unterschied zwischen zwei Farben gerade noch wahrnehmbar ist, wenn der gemäss Figur I.5 berechnete Wert gleich 1 ist.

Bemerkung: [Has/Newman, o.D.] weist darauf hin, dass die Wahrnehmbarkeit von Farbunterschieden auch davon abhängt, wie viel Zeit zum Vergleich zur Verfügung steht. Der Artikel zitiert ein Experiment, bei dem gemessen wurde, wie lange ein unerfahrener Betrachter benötigt, bis er den Unterschied bemerkt. Es werden Zahlen von 5 Sekunden für $\Delta E_{ab} = 15$, 10 Sekunden für $\Delta E_{ab} = 10$ und 15 Sekunden für $\Delta E_{ab} = 5$ genannt.

$$\Delta E_{ab} = \sqrt{\Delta L^{*2} + \Delta a^{*2} + \Delta b^{*2}}$$

Figur I.5: Berechnung des Farbunterschieds im kartesischen L*a*b*-Raum.

Genauigkeit der *L**-Achse: Für die Helligkeit reicht damit eine Genauigkeit von einer Nachkommastelle aus.

Genauigkeit der C^*_{ab}- und h^*_{ab}-Achsen: Zwar sind a* und b* theoretisch unbeschränkt, aber in der Praxis werden Grenzen von ±128, auf ganze Zahlen gerundet, als mehr als ausreichend betrachtet (vgl. [Adobe, 1992]). Welche Genauigkeit sind damit für C^*_{ab} und h^*_{ab} nötig, damit die Ungenauigkeit in der a*/b*-Ebene nicht grösser als 1 wird?

Die durch die Winkelangabe eingeführte Ungenauigkeit steigt mit zunehmender Distanz vom Nullpunkt. Damit kann eine Genauigkeit als ausreichend erachtet werden, wenn ‹127, 128› und ‹128, 128› in der a*/b*-Ebene noch unterscheidbar sind. Wie aus Figur I.6 ersichtlich ist, reicht für diesen extremen Fall eine Nachkommastelle aus. Es handelt sich dort um zwei gerade noch unterscheidbare Orangetöne, die allerdings so stark gesättigt sind, dass sie von kaum einem Gerät darstellbar sein dürften.

*a**	*b**	C^*_{ab}	h^*_{ab}
127	128	180.3	45.2
128	128	181.0	45.0

Figur I.6: Kartesische und polare Koordinaten einer sehr weit vom Nullpunkt entfernten Farbe (zur Umrechnung vgl. Figur I.3).

Kombination mit Namen

Farbnamen sind einfacher als Farbcodes (d.h. Zahlen) zu benutzen, besitzen aber den Nachteil, dass nur eine beschränkte Menge von Farben verwendet werden kann. In INTERLIS sind nun Namen mit einer numerischen Spezifikation verbindbar, so dass die Benutzer ihre eigenen Farbnamen definieren und untereinander mit den üblichen INTERLIS-Mitteln austauschen können.

Mit dieser Definition ist es auch möglich, dass bei Bedarf existierende Farbnamensysteme und Farbmusterkataloge — wie das Pantone- oder HKS-System — mit INTERLIS dokumentiert und genutzt werden können.

Dies erfolgt durch die Definition einer Metaklasse (vgl. Kapitel 1.4.3 Meta-Modelle und Meta-Objekte im INTERLIS 2-Referenzhandbuch). Deren Instanzen, so genannte Meta-Objekte, werden in einer besonderen Transferdatei festgehalten und vom INTERLIS 2-Compiler eingelesen. Sie stehen für INTERLIS-Datenmodelle zur Verfügung und können daher in Grafik-Definitionen benutzt werden, um beispielsweise die Farbe einer Signatur zu bestimmen.

Einsatzbeispiel in INTERLIS-Modellen

```
!! Bestandteil des Signaturenmodells

SYMBOLOGY MODEL SymbologyExample EXTENDS BasicSymbology =

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


  TOPIC Signs =

    CLASS LChColor EXTENDS INTERLIS.METAOBJECT =
      !! Attribut "Name" ererbt von 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;

    ...

    !! Bestandteil der Signatur-Klassen-Definition im Signaturenmodell
    CLASS BunteSignatur EXTENDS SIGN =
      ...
    PARAMETER
      Farbe: METAOBJECT OF SymbologyExample.LChColor;
    END BunteSignatur;

  ...
  END Signs;
  ...
```

In einer benutzerdefinierten Darstellungsanweisung (hier SimplePunktGr genannt) könnte dann z.B. die Farbe einer benutzerdefinierten bunten Signatur wie folgt angegeben werden (vgl. auch Kapitel 2.16 Darstellungsbeschreibungen im INTERLIS 2-Referenzhandbuch):

```
...

MODEL SimpleGrafik =

  CONTRACT ISSUED BY Unknown;

  IMPORTS SymbologyExample, Daten;

  SIGN BASKET SimpleSigns ~ SymbologyExample.Signs;


  TOPIC BuntePunktGrafik =
    DEPENDS ON Daten.Punkte;

    GRAPHIC SimpleBuntePunktGr BASED ON Daten.Punkte.Punkt =
      Symbol OF SymbologyExample.Signs.BunteSignatur: (
        Sign := { Punkt };
```

```
          Pos := Lage;
          Farbe := Braun;
        );
      END SimpleBuntePunktGr;

    END BuntePunktGrafik;

  END SimpleGrafik.
  ...
```

Auf die Angabe eines kompletten Beispiels wie auch die Anzeige der nötigen Metatabelle wird hier verzichtet; es sei stattdessen auf Anhang C Das kleine Beispiel Roads verwiesen.

Beispielwerte

Figur I.7 nennt einige Farben mitsamt ihren Koordinaten. Da unsicher ist, ob dieses Dokument auf einem System geschrieben (und wohl auch gedruckt) wurde, das in der Lage ist, Farben korrekt wiederzugeben, muss an dieser Stelle leider darauf verzichtet werden, die Farben bunt darzustellen.

Name	*L**	*a**	*b**	C^*_{ab}	h^*_{ab}
Schwarz	0.0	0	0	0.0	0.0
Dunkelgrau	25.0	0	0	0.0	0.0
Mittelgrau	50.0	0	0	0.0	0.0
Hellgrau	75.0	0	0	0.0	0.0
Weiss	100.0	0	0	0.0	0.0
Fuchsiarot	40.0	70	0	70.0	0.0
Hellblau	80.0	0	-30	30.0	270.0
Sattes Gelb	90.0	0	100	100.0	90.0
Braun	50.0	30	50	58.3	59.0
Lila	50.0	50	-50	70.7	315.0

Figur I.7: Kartesische und polare Koordinaten einiger Farben.

Ein konkretes Anwendungsbeispiel mit Farben-Definitionen ist im Anhang C Das kleine Beispiel Roads zu finden.

Hinweise für Systemimplementierer

Den Implementierern von INTERLIS-konformen Systemen stellt sich die Frage, wie die Farbwerte aus dem geräteunabhängigen *L* C*ab h*ab*-System in das Farb-Koordinatensystem eines konkreten Bildschirms oder Druckers zu transformieren sind.

Ein standardisiertes Dateiformat ermöglicht es, die Farbverzerrung eines bestimmten Gerätes in Form so genannter *Geräte-* oder *Farbprofile* festzuhalten (so genanntes ICC-Profil-Format). Unter anderem enthalten diese Dateien Parameter, welche in die Umrechnung von einem geräteunabhängigen Farbraum zum gerätespezifischen Farb-Koordinatensystem einfliessen. Ersteres ist entweder *XYZ* oder *L*a*b**, Letzteres typischerweise *RGB oder CMYK*. Das Format und die nötigen Umrechnungsfunktionen werden durch [ICC, 1996] definiert.

Ein Systemimplementierer kann nun in seinem Produkt direkt ICC-Profile unterstützen. Das Dateiformat ist relativ einfach aufgebaut, und die Umrechnungsfunktionen sind schnell implementiert. Für einige Plattformen stehen fertige Programmbibliotheken (wie *Apple ColorSync* oder *Kodak KCMS*) zur Verfügung.

Es sei in diesem Zusammenhang darauf hingewiesen, dass PDF den Farbraum *L*a*b** direkt unterstützt. PostScript ermöglicht es sogar, eigene Farbräume als beliebige Transformationen aus *XYZ* zu definieren. Die Umkehrfunktion der in Figur I.2 angegebenen Formel findet sich als Beispiel 4.11 in [Adobe, 1990]. Es ist relativ einfach, eine analoge Funktion in PostScript zu programmieren, welche direkt $L^* C^*_{ab} h^*_{ab}$ entgegennimmt.

Literaturhinweise

[Adobe, 1990] Adobe Systems: PostScript Language Reference Manual. 2nd Ed., 1990. ISBN 0-201-18127-4. 764 Seiten.

*Das Referenzhandbuch für PostScript unter anderem auch mit Hinweisen auf die Behandlung von Farben und verschiedene Umrechnungsmethoden, die in PostScript verfügbar sind. Beispiel 4.11 auf Seite 191 definiert den L*a*b*-Farbraum in PostScript.*

[Adobe, 1992] Adobe Developers Association: TIFF Revision 6.0. http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf

*Kapitel 23 definiert eine Variante des TIFF-Formats für Bilder im L*a*b*-Farbraum und nennt eine Anzahl von Vorteilen gegenüber RGB. Ausserdem wird eine schnelle Methode zur Umrechnung von L*a*b* zu RGB skizziert.*

[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

Allgemein verständliche Einführung in verschiedene Farbräume, mit illustrativen Grafiken.

[Apple, o.D.] Apple Computer, Inc.: A Brief Overview Of Color. www.apple.com/colorsync/benefits/training/overview.html

Sehr kurz gehaltene, allgemein verständliche Grobeinführung in verschiedene Konzepte im Zusammenhang mit Farben. An Laien gerichtet.

[Has/Newman, o.D.] Michael Has, Todd Newman: Color Management: Current Practice and the Adoption of a New Standard. www.color.org/wpaper1.html

Nennt Messdaten für den Rot-, Grün- und Blaureferenzpunkt von zwei typischen Computermonitoren und zeigt, dass sie stark von den häufig zitierten xy-Werte der NTSC-Standard-Phosphorfarben abweichen. Gibt eine Transformation von XYZ zum RGB-Raum eines bestimmten Bildschirms an.

[ICC, 1996] International Color Consortium: ICC Profile Format Specification. www.color.org/profile.html

Definiert ein Datei-Format, mit dem beliebige Geräte im Hinblick auf ihre Farbwiedergabe charakterisiert werden können. Anhang A diskutiert verschiedene Farbräume.

[Poynton, 1997] Charles A. Poynton: Frequently Asked Questions about Color. ftp://ftp.inforamp.net/pub/users/poynton/doc/colour/ColourFAQ.pdf

Begründet in Abschnitt 36, wieso sich HLS und HSV nicht zur Spezifikation von Farben eignen.

[Sangwine/Horne, 1998] Sangwine, Stephen J. und Horne, Robin E. N. [Hrsg.]: The Colour Imaging Processing Handbook. Chapman & Hall: London […], 1998. ISBN 0-412-80620-7. 440 Seiten.

Seriöse Einführung in die wissenschaftlichen Grundlagen der Farbwahrnehmung und deren Anwendungen im Bereich der Bildverarbeitung.

[Stokes et al., 1996] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar und Ricardo Motta: A Standard Default Color Space for the Internet — sRGB. November 1996.
http://www.color.org/sRGB.html
Spezifikation von sRGB.

Anhang J (Standard-Erweiterungsvorschlag) Koordinatensysteme und Koordinaten-Referenzsysteme

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Einleitung

Koordinaten beschreiben die Position eines Punktes in einem Raum, für den ein Koordinatensystem eingeführt ist. Ist ein Koordinatensystem bezüglich der Erde fest positioniert – man sagt auch: gelagert – dann nennt man es ein Koordinaten-Referenzsystem. Durch Koordinaten werden aber nicht nur Positionen festgehalten, sondern auch metrische Grössen, die aus Koordinaten ableitbar sind. Dazu gehören unter anderem Distanzen, Flächen, Volumen, Winkel und Richtungen, sowie weitere Eigenschaften wie Steigungen und Krümmungen.

Es gibt eine Vielzahl von Klassen (Typen) von Koordinatensystemen, und eine noch viel grössere Anzahl von Objekten, d.h. von Realisierungen (Instanzen) von Koordinatensystemen (siehe z.B. [Voser1999]). Die schweizerischen Landeskoordinaten beispielsweise stützen sich auf eine besondere Instanz (Objekt) eines Koordinaten-Referenzsystems [Gubler et al. 1996], das über eine Kartenprojektion aus einem geodätischen Referenzsystem hergeleitet werden kann [Snyder 1987, Bugayevskiy 1995]. Diese geodätischen Referenzsysteme bilden eine eigene Kategorie von Koordinaten-Referenzsystemen, welche die Geometrie eines Erdmodells beschreiben. Dabei wird z.B. zur Beschreibung der 2-dimensionalen Lage eine Kugel oder ein Ellipsoid zu Hilfe genommen und auf deren Oberfläche werden geografische Koordinaten definiert. Etwas komplizierter sieht es für die Höhe aus: Als geometrisch-physikalisches Erdmodell wird das Geoid [Marti 1997] bzw. ein Schweremodell [Torge 1975] verwendet, welches orthometrische bzw. normale Höhen definiert. Doch oft sind in der Praxis nur Gebrauchshöhen in Verwendungen.

Da Geodaten von Geomatik-Anwendungen einen Raumbezug aufweisen, muss jedem Geodatensatz ein Koordinatensystem zu Grunde liegen. Da sich einzelne Koordinatensysteme wesentlich voneinander unterscheiden, so ist es notwendig, entsprechende Referenzdaten mit den Geodaten mitzuliefern. Aus diesem Grund ermöglicht auch INTERLIS, diese Daten zum Koordinatensystem zu beschreiben.

Erst durch die Kenntnis des zu Grunde liegenden Koordinatensystems wird es möglich, die darin beschriebenen Geodaten in ein anderes Koordinatensystem überzuführen. Dies wiederum ist notwendig, um Geodaten, welche in unterschiedlichen Koordinatensystemen vorliegen, gemeinsam nutzen zu können [Voser 1996].

Wir betrachten zuerst Koordinatensysteme allgemeiner Art, dann die Beziehungen (Abbildungen) zwischen (allgemeinen) Koordinatensystemen, und anschliessend führen wir die Koordinaten-Referenzsysteme ein und beschäftigen uns damit.

Koordinatensysteme

Ein *Koordinatensystem* ermöglicht es, einen metrischen Raum "auszumessen". Ein Koordinatensystem hat einen Ursprung, Koordinatenachsen (deren Anzahl der Dimension des aufgespannten Raumes entspricht), sowie Masseinheiten, die den Achsen zugeordnet sind. Je nachdem es sich bei dem aufge-

spannten Raum um einen 1-, 2- oder 3-dimensionalen Raum handelt, wird durch das Koordinatensystem jedem Punkt des Raumes eine Einzelzahl, ein Zahlenpaar oder ein Zahlentripel als Koordinate(n) zugeordnet.

Der Euklidische 1-, 2- bzw. 3-dimensionale Raum ist definiert durch seine 1, 2 bzw. 3 geraden Achsen. Gekrümmte Räume brauchen zusätzliche Parameter zur Definition der Einbettung ihrer gekrümmten Achsen in einen Euklidischen Raum. Für geodätische Zwecke braucht man neben den Euklidischen Räumen verschiedener Dimension noch die 2-dimensionalen ellipsoidischen Räume sowie verschiedene Höhensysteme, welche als spezielle 1-dimensionale Euklidische Räume behandelt werden. Für diese braucht man das Schweremodell oder das Geoid.

Etwas abweichend vom bisherigen Gebrauch in der Geodäsie verwenden wir den Begriff *geodätisches Datum* als Synonym für *geodätisches Referenzsystem* und bezeichnen damit nichts anderes als ein besonderes Koordinatensystem, nämlich ein 3-dimensionales kartesisches Koordinatensystem, das bezüglich der Erde positioniert ist. Das kann auf zwei Arten geschehen:

(a) Man definiert das mittlere Gravitationszentrum der Erde als Nullpunkt des Koordinatensystems, die erste Achse durch die mittlere Rotationsachse der Erde, die zweite Achse senkrecht dazu durch den mittleren Meridian von Greenwich, und die dritte Achse senkrecht zu den beiden ersten, so dass insgesamt ein rechtsdrehendes System entsteht. So wird z.B. das Koordinatensystem WGS84 definiert.

(b) Die Erdoberfläche eines bestimmten Gebietes (meist eines Landes) wird optimal angenähert durch eine Kugel oder ein Rotationsellipsoid, dessen Drehachse parallel zur mittleren Rotationsachse der Erde verläuft. Dieses Rotationsellipsoid definiert ein kartesisches Koordinatensystem durch seine kleine Halbachse parallel zur Erdachse, durch eine seiner grossen Halbachsen und durch eine dritte Achse senkrecht zu den beiden ersten, so dass wiederum ein rechtsdrehendes System entsteht.

Ein gemäss (a) oder (b) auf der Erde positioniertes 3-dimensionales kartesisches Koordinatensystem nennen wir *geodätisches Datum* oder *geodätisches Referenzsystem.*

Unterschiedliche Herkunft von Koordinatensystemen in der Geomatik

Sehr unterschiedliche Gründe führen zur Definition von Koordinatensystemen in der Geomatik:

Sensorik: Die Datenerfassung der klassischen Geodäsie (z.B. mit dem Theodolit) aber auch Photogrammetrie und Fernerkundung verwenden in ihren Erfassungssensoren das der jeweiligen Methode entsprechende (lokale) Koordinatensystem für die Erfassung.

Geopositionierung: Die Positionsbeschreibung auf der Erde mit Hilfe eines (geodätischen) Erdmodells. Es gibt dabei drei Arten von (geodätischen) Erdmodellen [Voser 1999]:

- *physikalisch*: Das Erdmodell wird durch das Schwerefeld beschrieben oder durch das Geoid.
- *mathematisch*: Das Erdmodell ist ein symmetrischer Körper (z.B. eine Kugel oder ein Ellipsoid).
- *topografisch*: Das Erdmodell berücksichtigt auch Gebirge und Täler (Geländeoberflächenmodell).

Den genannten Erdmodellen entsprechen unterschiedliche Koordinatensysteme.

Kartenpositionierung: Da die Oberflächen der genannten Erdmodelle gekrümmte oder noch komplexere Gestalt annehmen, wird die Berechnung von Distanzen, Winkeln etc. sehr schwierig. Aus diesem Grund verwendet man Kartenprojektionen, welche die 2-dimensionale Fläche in eine Ebene abbilden. Eine Kartenprojektion ist eine geometrisch klar definierte Abbildungsvorschrift von der Oberfläche eines mathematischen Erdmodells in die Ebene. Bei diesem Vorgang treten Verzerrungen auf. Diese sind jedoch im Voraus bestimmbar und kontrollierbar.

Abbildungen zwischen Koordinatensystemen

Da viele Geodaten in unterschiedlichen Koordinatensystemen erfasst werden, oder von unterschiedlichen Institutionen in anderen Systemen verwaltet werden, ist es notwendig, die Methoden zu kennen, um die in einem Ausgangs-Koordinatensystem A vorliegenden Daten in ein gewünschtes Ziel-Koordinatensystem Z umzurechnen. Diese Umrechnung nennt man Abbildung des Koordinatensystems A bzw. des durch A definierten Raumes ins Koordinatensystem Z bzw. in den durch Z definierten Raum. Die Abbildung zwischen zwei Koordinatensystemen bzw. zwischen den beiden durch sie definierten Räumen ist bestimmt durch die Klassen der beiden Koordinatensysteme.

Es sind zwei wesentlich verschiedene Abbildungen von Koordinatensystemen zu unterscheiden, wenn man die Herkunft der Formeln und ihrer Parameter in Betracht zieht, nämlich Konversionen und Transformationen.

Eine *Konversion* ist eine durch Formeln und deren Parameter fest definierte Abbildung zwischen zwei Koordinatensystemen. Die Formeln und vor allem die Werte der notwendige Parameter sind im Voraus festgelegt [Voser 1999]. Zu den Konversionen gehören beispielsweise die Kartenprojektionen, das heisst z.B. die Abbildungen von der Ellipsoidoberfläche in die Ebene. Ebenfalls in die Kategorie der Konversionen fällt die Umrechnung von ellipsoidischen Koordinaten in die zugehörigen kartesischen Koordinaten mit dem Ursprung im Ellipsoidzentrum oder umgekehrt.

Als *Transformation* bezeichnet man eine Abbildung zwischen zwei Koordinatensystemen, wenn die Abbildungsvorschrift (Formel) auf Grund von Annahmen (Hypothesen) festgelegt wird und die Parameter durch statistische Analyse von Messungen in beiden Koordinatensystemen ermittelt werden [Voser 1999]. Typische Transformationen werden beim Übergang zwischen unterschiedlichen geodätischen Erdmodellen vorgenommen (geodätische Datumstransformationen), oder bei der Einpassung lokaler Koordinaten in übergeordnete Systeme, wie beispielsweise beim Digitalisieren die Umrechnung von Blatt- oder Digitalisiertisch-Koordinaten in Projektionskoordinaten.

Koordinaten-Referenzsysteme

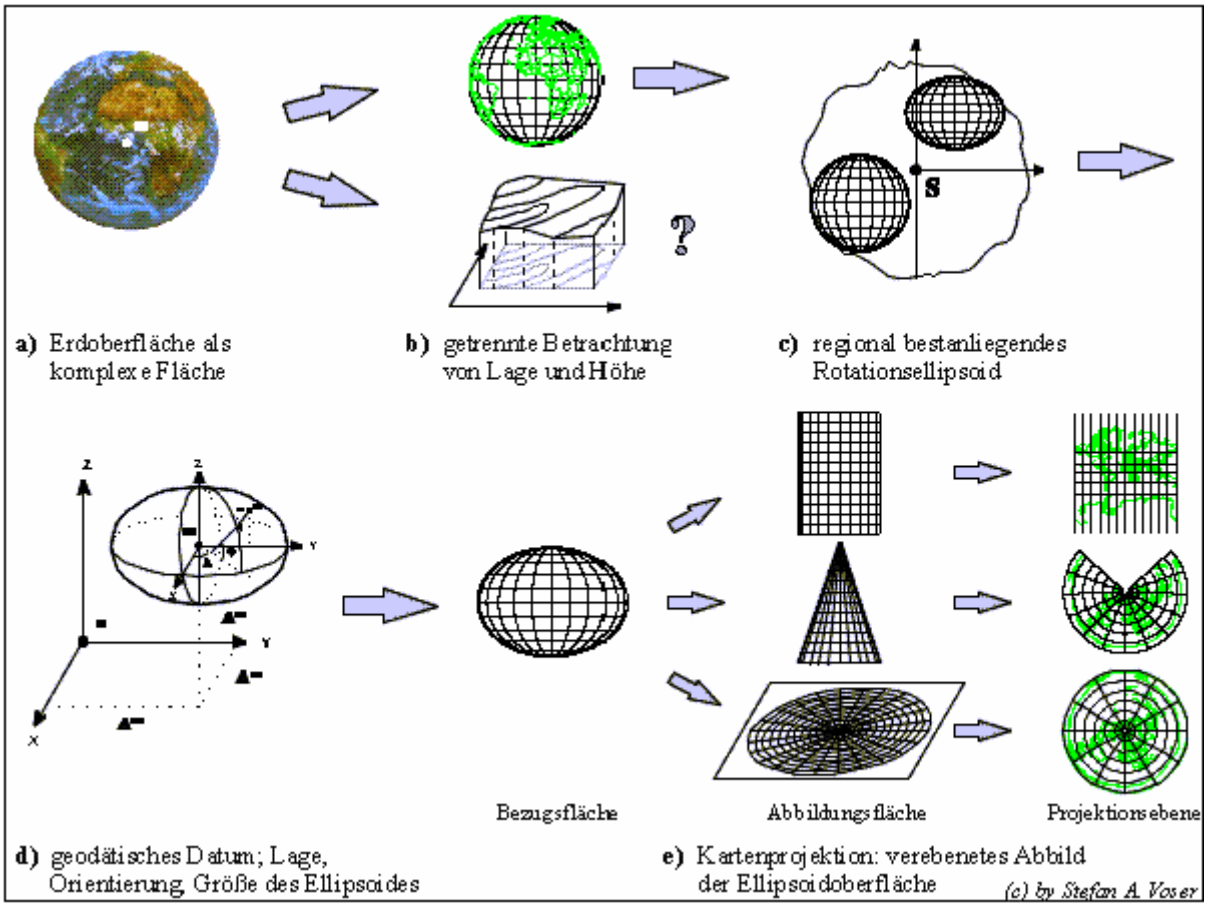
Koordinaten-Referenzsystem heisst ein Koordinatensystem, das über eine Folge von Zwischen-Koordinatensystemen durch Konversionen aus einem geodätischen Referenzsystem (d.h. aus einem geodätischen Datum) hergeleitet werden kann.

Die geodätischen Referenzsysteme (oder geodätischen Datums) sind selbst die wichtigsten Koordinaten-Referenzsysteme. Diese beziehen sich auf ein geodätisches Erdmodell (siehe oben).

Übersicht wichtiger Koordinaten-Referenzsysteme

In der Figur J.1 unten sind einige wichtige geodätische und kartografische Ausprägungen von Koordinaten-Referenzsystemen dargestellt. Am Anfang der Folge von Koordinatensystemen und Abbildungen steht die Erde. Man versucht dieser ein geometrisches Erdmodell zuzuweisen, um damit Positionen auf ihr zu beschreiben. Zunächst kann man der Erde als Ganzes ein 3-dimensionales kartesisches Koordinatensystem zuordnen mit Nullpunkt im Gravitationszentrum der Erde (siehe Methode (a) im Kapitel Koordinatensysteme weiter oben). In der Folge bearbeitet man allerdings die Lage und die Höhe eines Punktes

unabhängig voneinander. Betrachten wir zunächst nur, was zur Bestimmung der Lage zu unternehmen ist: Aus geodätischen Messungen werden die Grösse und die Form eines Rotationsellipsoides bestimmt, das die Erdoberfläche lokal optimal annähert. Diesem Rotationsellipsoid kann nach der Methode (b) im Kapitel Koordinatensysteme ein geodätisches Datum zugeordnet werden. Viele der für nationale Landesvermessungen ausgewählten Erdmodelle sind "lokal" gelagert, d.h. so, dass der Mittelpunkt des Ellipsoides nicht mit dem Gravitationszentrum der Erde (Schwerpunkt, Physikalischer Erdmittelpunkt) zusammenfällt. Nun gibt es aber, wie wir oben ((a) im Kapitel Koordinatensysteme) gesehen haben, geodätische Referenzsysteme, welche im Schwerpunkt gelagert sind. Aus diesem Grund ist es heutzutage relativ einfach möglich die Parameter der Datums-Transformation zu einem solch übergeordneten System zu bestimmen. Hat man sich für ein lokales Rotationsellipsoid entschieden, so kann andererseits dessen Oberfläche je nach Anforderung durch eine geeignete Kartenprojektion in eine Ebene abgebildet werden.



Figur J.1: Von der Erdoberfläche zu zweidimensionalen Lagekoordinaten.

Datenstruktur für Koordinatensysteme und Abbildungen zwischen diesen

Das vorgeschlagene konzeptionelle Schema für die Daten, welche zur Beschreibung von Koordinatensystemen und von Abbildungen zwischen diesen benötigt werden, beschränkt sich bewusst *nicht* auf Koordinaten-Referenzsysteme sondern ist allgemein für Koordinatensysteme konzipiert. Denn auch beispielsweise Digitalisiertisch- und Bildschirm-Koordinatensysteme oder Signaturen-Koordinatensysteme ohne expliziten Bezug zur Erdoberfläche sollen beschrieben werden können.

Koordinatensysteme und Abbildungen zwischen diesen sind die beiden Schlüsselkonzepte für die exakte Charakterisierung der räumlichen Referenzierung von Geodaten. Entsprechend weist das konzeptionelle Modell (oder konzeptionelle Schema) der Datenstruktur zwei Hauptgruppen von Klassen auf, nämlich

"Coordinate systems for geodetic purposes" und "Mappings between coordinate systems". Die dritte Dimension, Höhe, wird wie folgt bearbeitet: In einem 3-dimensionalen kartesischen Koordinatensystem ist die Höhe implizit integriert als dritte Koordinate. Aber Koordinatensysteme des täglichen Gebrauchs sind normalerweise die Kombination eines 2-dimensionalen Lage-Koordinatensystems und eines zusätzlichen 1-dimensionalen Höhensystems. Die Daten von Koordinatensystemen dieses Typs werden beschrieben durch zwei unabhängige Datensätze, zunächst durch die Daten eines 2-dimensionales Koordinatensystems (2-dimensional kartesisch oder ellipsoidisch) und zusätzlich durch die Daten eines Höhensystems von passendem Typ (normal, orthometrisch oder ellipsoidisch).

Wie werden mit Hilfe der vorgeschlagenen Datenstrukturen Abbildungen zwischen Koordinatensystemen realisiert? Auf die folgende Art: Koordinatensysteme bilden die Knoten und Abbildungen zwischen ihnen bilden die Kanten in einer Graphenstruktur. Im DOMAIN-Abschnitt eines Anwendungsmodells (Anwendungsschemas) findet man den Namen des verwendeten Koordinatensystems. Sollen nun die gegebenen Koordinaten abgebildet werden auf ein anderes Koordinatensystem oder sind beispielsweise GeoTIFF-Parameter zu berechnen, die einer solchen Abbildung entsprechen, dann hat ein geeignetes Programm in der Graphenstruktur von Koordinatensystemen und Abbildungen den kürzesten Weg zu finden vom Knoten des gegebenen Koordinatensystems (gemäss DOMAIN) zum Knoten des Zielsystems und dann die nötigen Abbildungen zu berechnen vom Ausgangssystem über evtl. Zwischen-Koordinatensysteme bis zum Zielsystem.

Für die Beschreibung von Koordinatensystemen stellt INTERLIS zwei interne Klassen und Schlüsselwörter zur Verfügung, nämlich: AXIS und COORDSYSTEM. Diese kommen im konzeptionellen Datenmodell (dem Koordinatensystem-Modell oder Koordinatensystem-Schema) "CoordSys" zum Einsatz (siehe unten). Details dazu sind zu finden im Kapitel 2.10.3 Referenzsysteme im INTERLIS 2-Referenzhandbuch.

Literaturhinweise

[Bugayevskiy 1995] Bugayevskiy Lev M., Snyder, John P.: Map Projections, A Reference Manual, Taylor&Francis, London, 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; VPK 2/96.

[Marti 1997] Marti, Urs: Geoid der Schweiz 1997. Geodätisch-geophysikalische Arbeiten in der Schweiz, Schweizerische Geodätische Kommission, Volume 56, Zürich, 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, März 1996, Freising.

[Voser 1999] Voser, Stefan A.: MapRef - The Internet Collection of Map Projections and Reference Systems for Europe; 14. ESRI European User Conference, Presentation and Proceedings; 15.-17. Nov. 1999; Munich; www.mapref.org/.

Das Referenzsystem-Modell

Datenstruktur für Koordinatensysteme und Koordinaten-Referenzsysteme sowie Abbildungen zwischen ihnen. Konzeptionelles Datenmodell (konzeptionelles Schema) mit 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.
```

Die Datei MiniCoordSysDaten, deren Namen in MetadataUseDef vorkommen kann, enthält die folgenden Daten im INTERLIS 2-Transferformat

```
<?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>
```

Beispiel

Was müsste innerhalb eines Applikationsmodells (bzw. Applikationsschemas) angegeben werden, um das verwendete Koordinatensystem, bzw. Koordinaten-Referenzsystem eindeutig zu identifizieren?

```
!! File CoordSysEx.ili Release 2003-03-18
```

```
INTERLIS 2.2;

DATA MODEL Beispiel (de) =  !! 2-letter code (ISO 639)

  IMPORTS CoordSys;

  REFSYSTEM BASKET BCoordSys ~ CoordSys.CoordsysTopic;

  UNIT
    Meter = [m];

  DOMAIN
    LKoord = COORD
      480000.000 .. 850000.000 [m] {CHLV03[1]},
       60000.000 .. 320000.000 [m] {CHLV03[2]},
      ROTATION 2 -> 1;
    Hoehe = COORD
      -200.000 .. 5000.000 [m] {SwissOrthometricAlt[1]};
    HKoord = COORD
      480000.000 .. 850000.000 [m] {CHLV03[1]},
       60000.000 .. 320000.000 [m] {CHLV03[2]},
        -200.000 ..   5000.000 [m] {SwissOrthometricAlt[1]},
      ROTATION 2 -> 1;


  TOPIC T =

    CLASS Fixpunkt =
      Name: TEXT*20;
      Position: LKoord;
    END Fixpunkt;

  END T;

END Beispiel.
```

Anhang K (Standard-Erweiterungsvorschlag) Signaturenmodelle

Bemerkung

Die folgende Spezifikation ist nicht normativer Bestandteil von INTERLIS. Dies ist ein Standard-Erweiterungsvorschlag auf der Basis des INTERLIS 2-Referenzhandbuches im Sinne einer Empfehlung. Es ist geplant, diesen Vorschlag nach einer breiten Diskussion in eine definitive Regelung umzuwandeln. Für Anwendungsbeispiele siehe auch die entsprechenden INTERLIS 2-Benutzerhandbücher.

Abstraktes Signaturenmodell

Beschreibung des abstrakten Signaturenmodells.

```
!! 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.
```

Standard-Signaturenmodell

Beschreibung des auf dem abstrakten Signaturenmodell aufbauenden, erweiterten Standard- Signaturenmodells.

```
!! 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.
```

Beispiel

Siehe Anhang C Das kleine Beispiel Roads.

Anhang L (informativ) Glossar

Abkürzungen der Umgangssprache, fachtechnische Abkürzungen siehe Definitionen

Abk.	Abkürzung.
Art.	Artikel (in Gesetzestexten).
Abs.	Absatz (in Gesetzestexten).
Def.	Definition.
de	deutsch.
en	english.
fr	français.
Syn.	Synonym.
INTERLIS	Der Vermerk INTERLIS wie z.B. INTERLIS 2. 6.4 bedeutet, dass im Abschnitt 2.6.4 dieses INTERLIS 2-Referenzdokuments (SN 612031) weitere Informationen zu diesem Begriff zu finden sind.
→ A	A ist ein Begriff, der in diesem Glossar definiert ist.

Definitionen

Abbildung

(aus einem Raum A, definiert durch ein → Koordinatensystem in einen anderen Raum Z, definiert durch ein zweites → Koordinatensystem:) Vorschrift, die jedem Punkt a aus A genau einen Punkt z aus Z zuordnet.

Bemerkung: Besondere A. sind → Transformation und → Konversion.

Abgeleitetes Attribut

→ Attribut, dessen → Wertebereich durch eine Funktionsvorschrift (→ logischer Ausdruck, Berechnung) berechnet wird.

Syn. derived attribute (en); attribut dérivé (fr).

Bemerkung 1: Abgeleitete A. können nicht direkt geändert werden.

Bemerkung 2: In → INTERLIS 2 wird die Funktionsvorschrift über eine → Funktion definiert.

Abstrakte Klasse

→ Klasse, die keine → Objekte enthalten kann.

Syn. abstract class (en); classe abstraite (fr).

Bemerkung: Eine A. ist immer unvollständig und bildet die Basis für → Unterklassen (d.h. für → Spezialisierungen), deren Objektmenge dann nicht leer sein muss.

Aggregation

Gerichtete → eigentliche Beziehung zwischen einer übergeordneten → Klasse und einer untergeordneten → Klasse. Einem Ganzen (Ober-Objekt der übergeordneten → Klasse) sind mehrere Teile (Unter-Objekte) der untergeordneten → Klasse zugeordnet. Einem Teil können auch mehrere Ganze zugeordnet sein. Beim Kopieren eines Ganzen werden alle zugeordneten Teile mitkopiert. Beim Löschen eines Ganzen können alle zugeordneten Teile weiter existieren.

Bemerkung 1: Mit Hilfe der A. wird die → Beziehung zwischen einem Ganzen und seinen Teilen beschrieben (z.B. Auto/Motor). Die → Rolle der → Unterklasse kann bezeichnet werden mit "ist-Teil-von", "is-part-of" (en).

Bemerkung 2: In → INTERLIS 2 wird die A. in Analogie zur → UML-Klassendiagramm-Notation mit einem (leeren) Rhombus (-<>) angegeben.

Bemerkung 3: Siehe auch → Komposition.

Argument

→ Wert eines → Parameters.

Assoziation

eigentliche Beziehung, welche die Unabhängigkeit der beteiligten → Klassen nicht einschränkt. Die zugeordneten → Objekte können unabhängig voneinander kopiert und gelöscht werden.

Syn. association (en, fr).

Bemerkung 1: In → INTERLIS 2 steht für die Beschreibung der A. die → Assoziationsklasse zur Verfügung.

Bemerkung 2: Siehe auch → Referenzattribut, → Aggregation und → Komposition.

Assoziationsklasse

→ Klassenelement zur Beschreibung einer → Assoziation, → Aggregation oder → Komposition.

Attribut

Daten(elemente) entsprechend einer spezifischen Eigenschaft von → Objekten einer → Klasse und von → Strukturelementen einer → Struktur (INTERLIS 2.6.4). Ein A. hat einen A.-Namen und einen → Wertebereich.

Syn. Merkmal (de), attribute (en).

Bemerkung: Jedes → Objekt einer → Klasse enthält gleichermassen ein → Datenelement eines A. mit einem individuellen → Wert. Anschaulich entspricht ein A. der Kolonne einer → Tabelle.

Attributspezialisierung

Einschränkung des → Wertebereichs eines → Attributes.

Bemerkung: A. wird auch verwendet zur Def. von → Vererbungsbeziehungen.

Ausdruck

Syn. für → logischer Ausdruck.

Basis-Klasse

→ Klasse, deren → Objekte an der Bildung einer → Sicht beteiligt sind.

Basis-Sicht

→ Sicht, deren → Objekte an der Bildung einer neuen → Sicht beteiligt sind.

Basis-View

Syn. für → Basis-Sicht.

Basisdatentyp

vordefinierter → Wertebereich wie z.B. TEXT oder BOOLEAN (→ INTERLIS 2).

Basisklasse

Syn. für → Oberklasse.

Bedingung

Syn. für → Konsistenzbedingung.

Bedingungsattribut

→ Attribut, für das eine → Konsistenzbedingung formuliert ist.

Bedingungsklasse

→ Klasse, für die eine → Konsistenzbedingung formuliert ist.

Behälter

Sammlung von → Objekten, die zu einem → Thema oder zu dessen → Erweiterungen gehören.

Syn. Basket (en).

Benutzerschnittstelle

Bedienungsoberfläche eines Computerprogramms.

Syn. Schnittstelle (de), graphic user interface (en).

Bemerkung: Siehe auch → Klassenschnittstelle und → Datenschnittstelle.

Bestandteilnamen

→ Namenskategorie bestehend aus den Namen von → Laufzeitparametern, → Attributen, → Zeichnungsregeln, → Parametern, → Rollen, → Beziehungszugängen und → Basis-Sichten.

Beziehung

Menge von Objektpaaren (bzw. im allgemeinen Fall von Objekt-n-Tupeln, die auch Beziehungsobjekte heissen). Das erste → Objekt jedes Paares gehört zu einer ersten → Klasse A, das zweite zu einer zweiten → Klasse B. Dabei soll die Zuordnung von → Objekten zu den Paaren vorgegeben sein, sie muss also nur beschrieben, d.h. modelliert werden. Man unterscheidet → eigentliche B. (nämlich → Assoziation, → Aggregation, → Komposition), → Vererbungsbeziehung und → Referenzattribut.

Syn. relationship (en); relation (fr).

Bemerkung 1: Wie das Sichten-Konzept zeigt, ist es im Gegensatz dazu auch möglich, Zuordnungen algorithmisch z.B. aufgrund von Attributwerten zu berechnen.

Bemerkung 2: Siehe auch → Objektbeziehung.

Bemerkung 3: Für eine eigentliche B. sind → Stärke und → Kardinalität definiert.

Beziehungsobjekt

Def. siehe bei → Beziehung.

Beziehungszugang

Voraussetzungen und Möglichkeiten um → Beziehungsobjekte und mit Hilfe derselben auch → Objekte von (gewöhnlichen) → Klassen über → Pfade zu referenzieren.

Bidirektionale Assoziation

Def. siehe bei → Assoziation.

Darstellungsbeschreibung

→ Konzeptionelles → Schema, das die Zuordnung von → Grafiksignaturen zu → Objekten beschreibt und aus grafischen → Themen besteht. Die → Objekte können in einer → Sicht selektiert werden.

Syn. Grafikmodell, Grafikbeschreibung.

Bemerkung 1: Eine D. in → INTERLIS 2 besteht aus grafischen → Themen, die je einem Daten-Thema entsprechen (DEPENDS ON). Ein grafisches → Thema ist eine Sammlung von → Grafikdefinitionen (nicht von → Klassen!). Jede → Grafikdefinition gehört zu einer → Klasse oder → Sicht (BASED ON) des entsprechenden Daten-Themas, ordnet mittels → Zeichnungsregeln den → Objekten dieser → Klasse oder → Sicht je eine → Grafiksignatur zu und legt die → Grafiksignatur → Argumente fest, entsprechend den → Daten der → Objekte. Die → Daten dieser → Grafiksignaturen, d.h. ihre Namen und ihre grafische Darstellung befinden sich in einer → Signaturenbibliothek, die in einem entsprechenden → Signaturenmodell beschrieben ist.

Bemerkung 2: Die D. kann selber auch → Datenschemas enthalten (z.B. → Klassen, die Textpositionen beschreiben).

Datei

Def. siehe Informatik.

Syn. file (en), fichier (fr).

Datenabstraktion

Abstrahieren (unter anderem weglassen) von unwichtigen Details über → Daten.

Syn. data abstraction (en); abstraction de données (fr).

Bemerkung 1: Das Trennen des Was? (→ Klassenschnittstelle, → Typ) vom Wie? (→ Klasse, konkrete Implementierung). → Generalisierung und → Spezialisierung sind mögliche Abstraktionsprinzipien.

Bemerkung 2: Die tatsächliche Realisierung der → Operationen und der innere Aufbau des → Objektes oder → Strukturelementes werden verborgen, d.h. man betrachtet abstrakt die Eigenschaften und lässt die tatsächliche Implementierung ausser Acht.

Datenbank

Logische Verwaltungseinheit für die Bearbeitung und dauerhafte Speicherung von → Objekten. Abk. DB.

Bemerkung: Auf einem → System können mehrere D. betrieben werden. Es ist auch möglich, dass eine D. über mehrere → Systeme verteilt ist.

Datenbankzustand

Gesamtheit aller → Daten und → Beziehungen einer → Datenbank zu einem bestimmten Zeitpunkt. Jeder D. hat einen Namen.

Bemerkung: Eine → Datenbank wird durch eine oder mehrere → Mutationen von einem D. in den nächsten übergeführt (→ Nachführung).

Datenbeschreibung

Mehrdeutiges Syn. für → Datenschema und → Datenmodell.

Datenbeschreibungssprache

Formale Sprache zur exakten Beschreibung von Daten.

Syn. Data description language (DDL), conceptual schema language (CSL).

Datenelement

Def. siehe Informatik. Vergleiche dazu → Wertebereich.

Datenkatalog

Syn. für → Objektkatalog.

Datenmodell

Exakte Beschreibung von Daten (so genanntes → konzeptionelles → Datenschema), die vollständig und in sich geschlossen ist. Das D. ist das hierarchisch höchste → Modellierungselement.

Syn. Modell, Datenbeschreibung.

Bemerkung 1: Vorsicht! In der Datenbanktheorie ist D. gebräuchlich als Syn. für konzeptionellen Formalismus (d.h. ein D. wird als → Methode für die Herstellung eines → konzeptionellen Schemas betrachtet).

Bemerkung 2: Ein D. besteht aus mindestens einem → Thema.

Bemerkung 3: In → INTERLIS 2 durch das Schlüsselwort MODEL bezeichnet. Das → Package, das dem D. entspricht, ist oberhalb aller → Packages, die den → Themen eines D. entsprechen.

Datenschema

Beschreibung von Inhalt und Gliederung von → Daten, die einen anwendungsspezifischen Ausschnitt der Realität charakterisieren, sowie von Regeln, die dafür gelten und von → Operationen, welche mit den → Daten ausgeführt werden können.

Syn. Datenbeschreibung, Schema, konzeptionelles Schema, Ontologie.

Bemerkung 1: Mehrzahl: Datenschemata oder Datenschemas.

Bemerkung 2: Entsprechend dem Abstraktionsniveau, auf dem man die → Daten beschreibt, unterscheidet man das → konzeptionelle Schema, das logische Schema und das physische Schema. Zur Formulierung eines D. gibt es geeignete → Datenbeschreibungssprachen.

Bemerkung 3: Bei → Datenbanken wird das dem → konzeptionellen Schema entsprechende und gemäss den systemspezifischen Gliederungsmöglichkeiten formulierte logische Schema auch internes Schema genannt. Logische oder auch physische Schemata von peripheren Geräten oder Austauschdateien heissen oft auch externe Schemata oder Formatschemata.

Datenschnittstelle

Programm zum Umformatieren von → Transferdateien oder → Protokoll für den → Datentransfer.

Syn. Schnittstelle (de).

Bemerkung: Siehe auch → Klassenschnittstelle und → Benutzerschnittstelle.

Datentransfer

Verschiebung von → Daten von einer → Datenbank A zu einer anderen → Datenbank Z. A wird bezeichnet als Ausgangssystem, Quelle, → Sender, Sendersystem, Source, Z als → Zielsystem, → Empfänger, Target. Die Lieferung der zu transferierenden → Daten durch → System A wird auch als Export bezeichnet, die Übernahme durch → System Z als Import.

Syn. Transfer, Datenübertragung.

Datentransfer-Mechanismus

(Konzeptionelle) → Datenbeschreibungssprache und (physisches) → Transferformat sowie Regeln zur Herleitung eines solchen → Transferformats für Daten, die mit der → Datenbeschreibungssprache beschrieben sind.

Datentyp

Syn. für → Wertebereich.

Datum

Mehrdeutiges Syn. für → geodätisches Datum, Zeitangabe (z.B. 2002-06-25) und Singular des Wortes 'Daten'.

Datumstransformation

→ Transformation von einem → geodätischen Datum (bzw. vom dadurch definierten Raum) auf ein anderes → geodätisches Datum (bzw. auf dessen Raum).

Definitionsbereich eines Namens

→ Namensraum der → Namenskategorie dieses Namens zum → Modellierungselement, in welchem der Name definiert wird.

Bemerkung 1: Im D. darf jeder Name nur eine Definition/Bedeutung haben. Hingegen darf derselbe Name z.B. im → Namensraum jeder → Namenskategorie desselben → Modellierungselementes je einmal definiert werden.

Bemerkung 2: Der D. ist Teil des → Sichtbarkeitsbereiches eines Namens.

Eigentliche Beziehung

Def. siehe bei → Beziehung.

Einfachvererbung

Def. siehe bei → Vererbung.

Einheit

Basiselement einer Mess-Skala (Beispiele: Meter, Sekunde).

Syn. unit (en).

Einseitige Beziehung

Syn. für → Referenzattribut.

Element

Grundbegriff der Mengenlehre. Eine Menge besteht aus E.

Syn. Instanz.

Bemerkung: Siehe auch → Modellierungselement oder → Grafikelement.

Ellipsoidische Höhe

Euklidische Distanz eines Punktes vom Ellipsoid gemessen entlang der Flächennormalen durch diesen Punkt.

Ellipsoidisches Koordinatensystem

→ Koordinatensystem auf der 2-dimensionalen Randfläche eines 3-dimensionalen (Rotations-) Ellipsoids.

Empfänger

Def. siehe → Datentransfer.

Syn. Zielsystem.

Entität

Syn. für → Objekt.

Entitätsmenge

Syn. für → Klasse.

Erweiterung

Syn. für → Spezialisierung.

Feature

Syn. für → Objekt bzw. oft auch für → Klasse.

Feature type

Syn. für → Klasse.

Fortführung

Syn. für → Nachführung (in Deutschland und Österreich gebräuchlich).

Funktion

→ Abbildung aus → Wertebereichen von Eingabe-Parametern in den → Wertebereich eines Ausgabe-Parameters mittels einer Berechnungsvorschrift (→ Parameter).

Syn. function (en); fonction (fr).

Bemerkung: In → INTERLIS 2 sind bestimmte F. vordefiniert, weitere müssen in einem → Kontrakt geregelt sein.

Gebrauchshöhe

Summe der Nivellementmessungen (Höhendifferenzen) längs einem Nivellementweg von einem Punkt der → Höhe 0 zum Punkt mit gesuchter G.

Generalisierung

→ Rolle der → Oberklasse in einer → Vererbungsbeziehung.

Syn. generalization (en); généralisation (fr).

Bemerkung 1: G. wird gelegentlich als Syn. für → Vererbung verwendet (obschon damit eigentlich die Gegenrichtung gemeint ist).

Bemerkung 2: In der Kartografie bezeichnet man mit G. alle Tätigkeiten, die sich durch die massstäblich verkleinerte → Abbildung von → Objekten der Realität ergeben.

Generelle Identifikation

→ Identifikation, die für alle (modellierten) → Objekte einer → Transfergemeinschaft eindeutig ist.

Bemerkung: Siehe auch → Objektidentifikation.

Geodätisches Datum

3-dimensionales → kartesisches Koordinatensystem, dessen Achsen eine feste Position und Orientierung bezüglich Massenmittelpunkt und Rotationsachse der Erde haben.

Syn. für → Datum (de).

Geodätisches Referenzsystem

Syn. für → geodätisches Datum.

Geoid

Äquipotentialfläche des Schwerefeldes.

Bemerkung: Das G. liefert ein physikalisches Erdmodell, welches sich dem Schwerefeld der Erde anpasst. Es hat eine unregelmässige Form, da es die unregelmässige Massenverteilung der Erde berücksichtigt. Es kann als die unter den Kontinenten weitergeführte mittlere Meeresoberfläche verstanden werden.

Gerichtete Beziehung

→ Aggregation oder → Komposition oder → Referenzattribut oder → Vererbungsbeziehung.

GIS

Abk. für Geo-Informationssystem oder Geografisches Informationssystem.

Grafikbeschreibung

Syn. für → Darstellungsbeschreibung.

Grafikdefinition

→ Klassenelement eines → grafischen Themas, d.h. jedes → grafische Thema einer → Darstellungsbeschreibung ist eine Sammlung von G. (nicht von → Klassen!). Jede G. gehört zu einer → Klasse (BASED ON) des entsprechenden Daten-Themas, ordnet mittels → Zeichnungsregeln den → Objekten dieser → Klasse eine oder mehrere → Grafiksignaturen zu und legt die → Grafiksignatur → Argumente fest, entsprechend den → Daten der → Objekte.

Syn. Graphic definition (en).

Grafikelement

Grafische Darstellung eines → Objektes unter Berücksichtigung der Lagegeometrie und weiterer → Attribute dieses → Objektes, nach eventueller Bearbeitung bereit für die Ausgabe durch ein passendes Peripheriegerät.

Syn. Grafikobjekt (de), graphic element (en).

Grafikmodell

Syn. für → Darstellungsbeschreibung.

Grafikobjekt

Syn. für → Grafikelement.

Grafikparameter

Syn. für → Parameter einer → Grafiksignatur.

Grafiksignatur

→ Daten für die grafische Darstellung von → Objekten noch unabhängig von der Lagegeometrie und weiteren Attributwerten dieser → Objekte. → Parameter von G. heissen auch kurz → Grafikparameter.

Syn. Symbol, Kartensignatur, Signatur, Signaturobjekt (de), signature, symbol, style (en).

Bemerkung 1: Es gibt vier Typen von G.: Schriften bzw. Textsignaturen (manchmal auch mit Textbeschriftungen oder einfach mit Beschriftungen bezeichnet), Punktsymbole (manchmal auch mit Punktsignatur oder einfach mit → Symbol oder Piktogramm bezeichnet), Liniensignaturen und (Einzel-) Flächensignaturen.

Bemerkung 2: In → INTERLIS 2 sind die → Daten und allfällige → Parameter einer G. im → Signaturenmodell beschrieben und die entsprechenden → Daten in einer → Signaturenbibliothek zusammengefasst. G. werden in einer → Grafikdefinition über G.-Namen referenziert und dabei werden für allfällige → Parameter entsprechende → Argumente definiert.

Grafisches Thema

Def. siehe → Darstellungsbeschreibung.

Hilfslinie

Lineares → Grafikelement, das zwei → Grafikelemente miteinander verbindet oder ein → Grafikelement mit einer Beschriftung.

Syn. help line (en); trait de rappel (fr).

Bemerkung: Typisch ist die H. als Darstellung einer Verbindung von einer Linien- oder Flächensignatur zu einer Beschriftung oder zur dazugehörenden Bemassungslinie.

Höhe

Entweder → ellipsoidische Höhe oder → Normalhöhe oder → orthometrische Höhe.

IDDL

Abk. für INTERLIS-Data Description Language = → INTERLIS-Datenbeschreibungssprache.

Identifikation

→ Attribut oder Attributskombination, deren → Wert ein → Objekt in seiner → Klasse eindeutig kennzeichnet.

Abk. ID.

Syn. Identifikator, Identität.

Bemerkung: Innerhalb einer INTERLIS 2-Transferdatei erhält jedes → Objekt zusätzlich zu den im → Datenschema beschriebenen Attributswerten eine I., die es innerhalb der → Transferdatei eindeutig kennzeichnet, die so genannte → Transferidentifikation (→ TID). Ist eine solche → TID eine → generelle und → stabile I., dann nennt man sie eine → Objektidentifikation (→ OID).

Identifikator

Syn. für → Identifikation.

Identität

Syn. für → Identifikation.

ILI

Abk. für INTERLIS.

Bemerkung: Ist auch als Dateinamenzusatz von → Dateien üblich, die ein in → INTERLIS (Version 1 und 2) beschriebenes → Datenschema enthalten.

Implementierte Klasse

Ausführbares Softwaremodul mit als → Methoden realisierten → Operationen.

Informationsebene

Nichtleere Menge von → Themen.

Inkrementeller Datentransfer

→ Datentransfer der Differenz zwischen zwei → Datenbankzuständen vom → Sender zum → Zielsystem.

Instanz

Syn. für → Element (konkretes Exemplar) einer Menge (Abstraktion).

Syn. instance (en; fr).

Bemerkung: Beispiele für I.: Ein → Wert ist eine I. eines → Datentyps. Ein → Objekt ist eine I. einer → Klasse. Ein → Behälter ist eine I. eines → Themas. Ein Objektpaar ist eine I. einer → Assoziationsklasse.

Interface

Syn. (en) für → Schnittstelle.

INTERLIS 2

→ Datentransfer-Mechanismus für Geodaten bestehend aus der → INTERLIS-Datenbeschreibungssprache (→ IDDL) und dem INTERLIS-XML-Transferformat (IXML) sowie Regeln für die Herleitung des IXML für eine mit → IDDL beschriebene Datenstruktur. → IDDL, IXML und Umsetzungsregeln sind definiert in der Schweizer → Norm SN 612031.

Abk. für "INTER Land-Informations-Systeme" (d.h. "zwischen den → GIS").

INTERLIS-Compiler

Programm, das aus einem → Datenschema in → IDDL die Beschreibung des zugehörigen ITF herleitet. Dabei wird die syntaktische Richtigkeit des → Datenschemas überprüft (so genanntes Parsing). Siehe INTERLIS Anhang A.

INTERLIS-Datenbeschreibungssprache

(Konzeptionelle) → Datenbeschreibungssprache des → Datentransfer-Mechanismus INTERLIS.

Syn. INTERLIS Data Description Language (kurz → IDDL).

Bemerkung: Ein in → IDDL beschriebenes → Datenschema kann als (Text-) Datei gespeichert werden. Für solche Schema-Dateien ist das Kürzel "ILI" als Dateinamenzusatz üblich. Beispiel: Die Schema-Datei des Grunddatensatzes der amtlichen Vermessung heisst DM01AV.ILI.

Kardinalität

Anzahl → Objekte der → Klasse B (bzw. A), die einem → Objekt der → Klasse A (bzw. B) durch die → Beziehung zwischen den → Klassen A und B zugeordnet werden können.

Syn. Multiplizität (de), cardinality, multiplicity (en).

Bemerkung: In → UML wird dafür auch der Begriff der → Multiplizität verwendet; mit "Kardinalität" will man dort die konkrete Anzahl der → Objektbeziehungen zwischen → Objektinstanzen bezeichnen.

Kartenprojektion

→ Konversion von einem ellipsoidischen oder sphärischen Raum in eine Euklidische Ebene.

Kartensignatur

Syn. für → Grafiksignatur.

Kartesisches Koordinatensystem

→ Koordinatensystem des Euklidischen Raumes, dessen Achsen Geraden sind, die paarweise senkrecht stehen.

Kartografisches Zeichensystem

Menge von grafischen Darstellungsmöglichkeiten für → Grafiksignaturen.

Syn. cartographic sign system (en).

Bemerkung 1: Ein konkretes auf dem Bildschirm dargestelltes oder auf Papier gedrucktes → Grafikelement ist das Resultat eines mehrstufigen Prozesses, bei dem → Objekte selektiert (selection), auf → Grafiksignaturen abgebildet (mapping), zusammengestellt, grafisch aufgebaut (rendering) und dargestellt werden (display).

Bemerkung 2: In → INTERLIS 2 werden über eine → Darstellungsbeschreibung die ersten zwei Stufen geregelt, die restlichen Stufen sind Implementationssache der Systeme, bzw. "Treiber";

teilweise existieren dort bestimmte Grafikstandards (wie z.B. PostScript, HPGL, OpenGL, Java2D, SVG).

Klasse

Menge von → Objekten mit gleichen Eigenschaften und → Operationen. Jede Eigenschaft wird durch ein → Attribut beschrieben, jede → Operation durch ihre → Schnittstellensignatur.

Syn. Objektklasse, Entitätsmenge, Objekttyp (de), feature type, feature, class (en), classe (fr).

Bemerkung 1: Eine mit → INTERLIS 2 beschriebene K. entspricht einer → UML-K. mit lauter öffentlichen ("public", d.h. sichtbaren) → Attributen.

Bemerkung 2: Siehe auch → Oberklasse, → Unterklasse, → Tabelle sowie → Klassenelement.

Bemerkung 3: Eine K. muss nicht → Objekte enthalten. Wenn sie → Objekte enthalten kann, spricht man von einer → konkreten K., wenn nicht, von einer → abstrakten K.

Klassendiagramm

Grafische Darstellung von → Klassen und ihren → Beziehungen.

Syn. class diagram (en); diagramme des classes (fr).

Klassenelement

→ Modellierungselement "des Modellierungsniveau Klasse". Genau: K. heissen → Klasse, → Struktur, → Assoziationsklasse, → Sicht, → Sicht-Projektion und → Graphikdefinition.

Klassenschnittstelle

Zugriff auf einen Teil oder die Gesamtheit der → Operationen einer → Klasse.

Syn. Schnittstelle (de), interface (en, fr).

Bemerkung 1: Eine → Klasse kann mehrere K. haben. Für jede derselben kann eine separate → Schnittstellenklasse definiert werden. Das → konzeptionelle → Schema einer → Schnittstellenklasse enthält nur → Schnittstellensignaturen.

Bemerkung 2: Siehe auch → Benutzerschnittstelle und → Datenschnittstelle.

Klassenspezialisierung

Einschränkung einer → Klasse durch zusätzliche → Attribute, → Beziehungen, → Konsistenzbedingungen oder → Attributspezialisierungen.

Bemerkung: K. wird verwendet zur Def. von → Vererbungsbeziehungen.

Komposition

Gerichtete → eigentliche Beziehung zwischen einer übergeordneten → Klasse und einer untergeordneten → Klasse. Einem Ganzen (Ober-Objekt der übergeordneten → Klasse) sind mehrere Teile (Unter-Objekte der untergeordneten → Klasse) zugeordnet, während einem Teil höchstens ein Ganzes zugeordnet sein kann. Beim Kopieren eines Ganzen werden alle zugeordneten Teile mitkopiert. Beim Löschen eines Ganzen werden alle zugeordneten Teile ebenfalls gelöscht.

Syn. composition (en, fr).

Bemerkung 1: Die Teile haben dabei keine Selbständigkeit, sondern gehören fest zum Ganzen. Die beteiligten → Klassen führen also keine gleichwertige → Beziehung, sondern stellen eine Ganzes-Teile-Hierarchie (en: consists-of), dar.

Bemerkung 2: In → INTERLIS 2 wird die K. als → Assoziationsklasse definiert.

Bemerkung 3: Siehe auch → Strukturattribut.

Konkrete Klasse

→ Klasse, die → Objekte enthalten kann.

Syn. concrete class (en); classe concrète (fr).

Bemerkung: Siehe auch → abstrakte Klasse.

Konsistenzbedingung

Einschränkung, welcher → Objekte genügen müssen.

Syn. Bedingung, Randbedingung, Zusicherung (de), constraint (en); contrainte (fr).

Bemerkung: Bestimmte K. sind in → INTERLIS 2 vordefiniert. Weitere K. sind mit → Funktionen, → logischen Ausdrücken oder Regeln formal definierbar und müssen in einem → Kontrakt geregelt sein.

Kontrakt

Vereinbarung mit Software-Werkzeuganbietern.

Bemerkung: Wird z.B. in → INTERLIS 2-Datenmodellen verlangt, in denen nicht vordefinierte → Funktionen, → Signaturenmodelle oder nicht vordefinierte → Linienformtypen verwendet werden.

Konversion

→ Abbildung von einem → Koordinatensystem (bzw. von dessen Raum) auf ein anderes → Koordinatensystem (bzw. auf dessen Raum), die durch Formeln und deren → Parameter fest definiert ist.

Syn. conversion (en, fr).

Bemerkung: K. wird gelegentlich auch verwendet als Syn. für Umformatieren von → Transferdateien.

Konzeptionelles Schema

Syn. für konzeptionelles → Datenschema. Def. siehe bei → Datenschema Bemerkung 2.

Syn. conceptual schema (en), schéma conceptionelle (fr).

Koordinaten-Referenzsystem

Syn. für → Referenzsystem.

Koordinatensystem

Basis eines Euklidischen Vektorraumes bzw. Urbild der Basis des zugeordneten Euklidischen Vektorraumes beim Kartenhomöomorphismus einer Mannigfaltigkeit (Details siehe Vektoranalysis).

Syn. coordinate system (en).

Bemerkung: Aus Sicht der → Daten ist ein K. definiert durch seine Achsen, die entweder Geraden sind (in → INTERLIS 2 so genannte LengthAXIS) oder gekrümmte Kurven (so genannte AngleAXIS) entsprechend der Art des Raumes, den sie auszumessen erlauben.

Laufzeitparameter

→ Parameter, dessen → Wert von einem Bearbeitungs-, Auswerte- oder Darstellungs-System zur Laufzeit bereitgestellt wird.

Bemerkung: Beispiele sind Darstellungs-Massstab, → Datum.

Layer

Im CAD-Bereich übliche Bezeichnung für die Zusammenfassung grafischer → Daten eines bestimmten → Typs. Gelegentlich auch in → GIS verwendet für → Thema.

Legende

Beschriftung und Erklärung einer Karte, bzw. eines Plans und der dabei verwendeten → Grafiksignaturen.

Syn. legend (en).

Bemerkung: Siehe auch → Darstellungsbeschreibung sowie → Signaturenbibliothek.

Linienformtyp

Form der Kurvenstücke, aus denen ein Linienzug zusammengesetzt ist (Gerade, Kreisbogen, andere Verbindungsgeometrien). Zur Def. der von → INTERLIS 2 unterstützten Objekt-Geometrien siehe INTERLIS 2.8.11 und 2.8.12.

Logischer Ausdruck

Mittels boolescher Operatoren verknüpfte Prädikate.

Syn. Ausdruck (de), logical expression (en), expression logique (fr).

Mehrfachvererbung

→ Vererbungsbeziehung, die einer → Unterklasse mehr als eine → Oberklasse zuordnet.

Bemerkung: M. ist in INTERLIS nicht vorgesehen.

Merkmal

Syn. für → Attribut.

Syn. property (en).

Metadaten

→ Daten über → Daten.

Syn. metadata (en), métadonnées (fr).

Bemerkung: Speziell in der Geoinformatik sind M. → Daten, die unter anderem Objektbeschreibung in Umgangssprache, Erfassung der → Objekte, Gliederung, Raumbezug, Qualität, Verfügbarkeit und Herkunft, usw. bezeichnen.

Metamodell

→ Datenmodell von → Metadaten.

Metaobjekt

→ Objekt, dessen Gegenstand der realen Welt eine Menge von → Objekten ist.

Bemerkung 1: Ein M. besteht also aus → Metadaten. M. gibt es zu einzelnen → Objekten und/oder zu allen → Objekten eines → Modellierungselementes.

Bemerkung 2: → Metadaten zu den → Werten einzelner → Attribute von → Objekten sind zusätzliche → Attribute der → Klasse dieser → Objekte.

Metaobjektnamen

→ Namenskategorie, die ausschliesslich aus Namen von → Metaobjekten besteht.

Methode

Implementierung einer → Operation durch eine Folge von Anweisungen (d.h. durch ein Programm).

Syn. method (en); méthode (fr).

Bemerkung: Mehrdeutiger Begriff. Oft als Syn. für → Operation verwendet.

Modell

Syn. für → Datenmodell.

Bemerkung: Die objektorientierte Modellierung unterscheidet Objekt-M. (als Syn. für den Teil eines → Datenschemas, der Inhalt und Gliederung der → Daten beschreibt) und Verhaltens-M. (als Syn. für den Teil eines → Datenschemas, der die → Operationen beschreibt, die mit den → Daten ausgeführt werden können).

Modellierungselement

Besonderes → Schemaelement. Es gibt drei M., nämlich → Datenmodell, → Thema und → Klassenelement.

Bemerkung: M. und → Namenskategorie definieren den → Namensraum.

Multiplizität

Syn. für → Kardinalität.

Mutation

Konsistenzerhaltende → Operation auf einer → Datenbank.

Mutationsdatenbank

Temporäre → Datenbank, mit deren → Objekten → Mutationen durchgeführt werden. Eine M. nimmt ihre → Objekte von einer → Primärdatenbank entgegen und gibt sie nach der Bearbeitung wieder an diese zurück (→ Nachführung).

Bemerkung: Eine M. kann auf dem gleichen → System wie die → Primärdatenbank (interne M.) oder auf einem anderen → System (externe M.) betrieben werden.

Nachführung

Eine oder mehrere → Mutationen auf einer → Primärdatenbank. Eine N. führt die → Primärdatenbank von einem → Datenbankzustand in den nächsten über.

Bemerkung: Die → Mutationen auf der → Primärdatenbank können zeitlich parallel ausgeführt werden. Die → Primärdatenbank muss bei parallelen → Mutationen die Konsistenz des Resultats gewährleisten.

Nachlieferung

→ Vollständiger oder → inkrementeller → Datentransfer eines → Datenbankzustands der → Primärdatenbank auf eine → Sekundärdatenbank.

Bemerkung 1: Die N. läuft immer sequenziell ab, d.h. eine → Sekundärdatenbank muss nie gleichzeitig mehrere → Nachlieferungen empfangen.

Bemerkung 2: Siehe auch → Synchronisation.

Namenskategorie

Teilmenge der Namen eines konzeptionellen → Datenschemas. Es gibt drei N., nämlich → Typnamen, → Bestandteilnamen und → Metaobjektnamen.

Bemerkung: N. und → Modellierungselement definieren den → Namensraum.

Namensraum

Menge der (eindeutigen) Namen einer → Namenskategorie in einem → Modellierungselement.

Syn. namespace (en).

Bemerkung: Der N. wird benötigt zur Festlegung des → Definitionsbereichs und des → Sichtbarkeitsbereichs eines Namens.

Norm

Eine 'de jure' N. (oder kurz N.) ist eine technische Vorschrift, die von nationalen oder internationalen Normenverbänden festgelegt wird. Eine 'de facto' N. ist eine allgemein anerkannte und mehrheitlich genutzte technische Vorschrift; weniger verbindlich als eine 'de jure' N.

Syn. standard (en), norme (fr).

Bemerkung 1: Ein Gesetz ist eine Vorschrift, die über 'de jure' und 'de facto' N. steht.

Bemerkung 2: Deutsches Syn. von 'de facto' N. ist 'Standard'. Die englische Übersetzung von N. ist (gleich geschrieben und fast gleich lautend) 'standard', womit in der deutschen Sprache nicht immer klar wird, ob jetzt von 'de facto' oder von 'de jure' N. die Rede ist.

Normalhöhe

Normalschwere gewichtete Kurvenlänge der Orthogonaltrajektorie der Normalellipsoidschar durch den Punkt zwischen dem Null-Normalellipsoid und dem Punkt.

Oberklasse

Def. siehe bei → Vererbungsbeziehung.

Syn. Superklasse (de), super class (en), classe supérieure (fr).

Objekt

Daten eines Gegenstandes der realen Welt zusammen mit den → Operationen, die mit diesen Daten ausgeführt werden können, und mit einer → Objektidentifikation.

Syn. Entität, Tupel, Objektinstanz (de), object instance, feature, feature instance (en), instance d'un object (fr).

Bemerkung 1: Siehe auch → Instanz, → Klasse.

Bemerkung 2: Ein O. hat im Gegensatz zu einem → Wert eine → Identität, existiert in Raum und Zeit, ist veränderbar bei Wahrung der → Identität und kann über Verweise gemeinsam benutzt werden. Ein O. ist konkret. Es ist an die Existenz realer Dinge gebunden.

Bemerkung 3: In der objekt-orientierten Literatur findet man folgende blumige Umschreibung des Begriffs O.: Eine konkret vorhandene → Einheit mit eigener (unveränderbarer) → Identität und definierten Grenzen (im übertragenen Sinne), die Zustand und Verhalten kapselt. Der Zustand wird repräsentiert durch → Attribute und → Beziehungen, das Verhalten durch → Operationen. Jedes O. gehört zu genau einer → Klasse. Die definierte → Struktur ihrer → Attribute gilt für alle O. einer → Klasse gleichermassen, ebenso das Verhalten. Die → Werte der → Attribute sind jedoch individuell für jedes O.

Objektbeziehung

Zwei → Objekte, die einander zugeordnet sind durch eine → Beziehung zwischen den → Klassen, denen sie angehören.

Syn. link (en).

Objektidentifikation

→ generelle und → stabile Identifikation.

Abk. OID.

Syn. Objektidentifikator, Objektidentität (de), object identifier, object identity (en).

Bemerkung 1: Die O. wird normalerweise nur von einem → System und nicht vom Anwender verändert. Die O. ist eine Eigenschaft, die ein → Objekt von allen anderen unterscheidet, auch wenn es möglicherweise die gleichen Attributwerte besitzt.

Bemerkung 2: Siehe auch → Transferidentifikation.

Bemerkung 3: Anhang E zum INTERLIS 2-Referenzhandbuch enthält einen Vorschlag für eine O.

Objektidentifikator

Syn. für → Objektidentifikation.

Objektidentität

Syn. für → Objektidentifikation.

Objektinstanz

Syn. für → Objekt.

Objektkatalog

Informelle Aufzählung von → Klassen mit umgangssprachlichen Definitionen (Name und Beschreibung der → Klasse) der für eine Anwendung relevanten Datenobjekte.

Abk. OK.

Syn. Datenkatalog.

Bemerkung 1: Zum OK gehören Angaben zum Detaillierungsgrad und zu den Qualitätsanforderungen (insbesondere zur geometrischen Qualität) sowie evtl. zu Erfassungsregeln.

Bemerkung 2: Der OK ist eine Vorstufe und eine Ergänzung des konzeptionellen → Datenmodells.

Objektklasse

Syn. für → Klasse.

Objekttyp

Syn. für → Klasse.

OID

Abk. für → Objektidentifikation.

Ontologie

Syn. für → Datenschema.

Bemerkung 1: O. ist "eine explizite formale Spezifikation einer gemeinsamen (en: shared) Konzeptualisierung"; d.h. bildlich gesprochen, eine Ablage (en: repository) von Konzepten.

Bemerkung 2: O. verwenden UML/OCL oder eigene Sprachen, wie z.B. DAML/OIL (DARPA Agent Markup Language + Ontology Interchange Language). O. bestehen typischerweise aus einem → konzeptionellen Datenschema, einer taxonomischen Hierarchie von → Klassen (Vokabular, Thesaurus) und Axiomen, welche die möglichen Interpretationen der definierten Terme einschränkt (meistens mit einer Logik-Sprache). O. sollen (in Zukunft) als höhere Abstraktion von → Datenschemas Anwendung finden für die Spezifikation von Software und für die Kommunikation zwischen Menschen.

Operation

→ Abbildung aus den Attributswertebereichen einer → Klasse und/oder aus → Wertebereichen von Eingabe-Parametern in den → Wertebereich eines Ausgabe-Parameters.

Bemerkung 1: Die Implementierung einer O. durch eine Folge von Anweisungen (d.h. durch ein Programm) heisst → Methode.

Bemerkung 2: Die Beschreibung einer O. heisst → Schnittstellensignatur und besteht aus Operationsnamen und Beschreibung der → Parameter.

Optional

Muss nicht zwingend vorhanden oder anwendbar sein, ist fakultativ. Gegenteil: Nicht-optional, d.h. obligatorisch.

Bemerkung 1: → Attribute sind o., wenn nicht gefordert ist, dass sie obligatorisch (mandatory) sind. Für obligatorische → Attribute steht in → IDDL das Schlüsselwort MANDATORY zur Verfügung.

Bemerkung 2: In → IDDL bezieht sich "nicht zwingend vorhanden" auf die → Transferdatei.

Orthometrische Höhe

Kurvenlänge der (gekrümmten) Lotlinie zwischen → Geoid und Punkt.

Package

UML-Sprachelement zur Beschreibung von → Modellen, → Themen und Teilen von Themen.

Syn. Paket (de).

Bemerkung 1: Ein P. definiert einen → Namensraum, d.h. innerhalb eines P. müssen die Namen der enthaltenen benannten → Schemaelemente eindeutig sein. Jedes benannte → Schemaelement kann in anderen P. referenziert werden, gehört aber zu genau einem (Heimat-) P.

Bemerkung 2: Bei → UML können die P. wiederum P. enthalten. Das oberste P. beinhaltet das Gesamtsystem entsprechend dem → Datenmodell von → INTERLIS 2.

Parameter

Daten(elemente), deren → Wert einer → Funktion, einer → Operation oder einem → Metaobjekt übergeben und/oder von → Funktionen oder → Operationen zurückgegeben werden. Zu jedem P. gehört ein Name, ein → Wertebereich und - bei → Funktionen oder → Operationen - eine Übergaberichtung (in, out, inout). Der konkrete → Wert eines P. heisst → Argument.

Bemerkung 1: Siehe auch → Laufzeitparameter.

Bemerkung 2: Mittels P. werden diejenigen Eigenschaften von → Metaobjekten bezeichnet, die nicht das → Metaobjekt selber, sondern dessen Gebrauch in der Anwendung betreffen.

Pfad

Folge von Namen von → Attributen und/oder → Klassen und/oder → Rollen von → Assoziationsklassen, welche ein → Objekt oder den → Wert eines → Attributes definiert, die durch einen → logischen Ausdruck zu bearbeiten sind.

Planrahmen

Beschreibung eines Plans durch die → Metadaten Titel, → Legende, Erstellerbeschreibung, Erstellungsdatum, Schriftartbezeichnung und die grafische Darstellung weiterer → Elemente, wie Koordinatenkreuze und Nordpfeil.

Syn. Kartenrahmen, Planlayout (de); layout of the plan (en), bord de plan (fr).

Planspiegel

Bereich, in dem der Inhalt eines Plans dargestellt wird.

Syn. Kartenspiegel.

Bemerkung: Gegen den äusseren Rand zu können abgestufte Abdeckungsbereiche definiert werden.

Polymorphismus von Objekten

Überall dort, wo → Objekte einer → Basisklasse erwartet werden, können auch → Objekte einer → Erweiterung stehen.

Syn. Teilmengen-Polymorphismus, Teilmengen-Polymorphie, Substitutionsprinzip (de), polymorphism (en).

Bemerkung 1: Siehe auch → P. von Operationen.

Bemerkung 2: In → INTERLIS 2 wird vor allem auf P. von → Objekten Bezug genommen.

Polymorphismus von Operationen

Aufgrund der → Schnittstellensignatur können → Objekte unterschiedlicher → Klassen auf identische Operationen-Namen (Nachrichten) antworten, d.h. durch → Operationen mit identischen Namen bearbeitet werden.

Syn. Polymorphie (de), polymorphism (en).

Bemerkung 1: Siehe auch → P. von Objekten.

Bemerkung 2: In → INTERLIS 2 wird vor allem P. von → Objekten verwendet.

Primärdatenbank

→ Datenbank in der die → Objekte bestimmter → Themen eines bestimmten Gebiets längerfristig verwaltet werden.

Protokoll

Menge der → Operationen, die einer Menge von → Klassen zugehören.

Referentielle Integrität

Regel, die festlegt, was mit einer → Objektbeziehung bzw. mit den betroffenen → Objekten passiert, wenn eines der beteiligten → Objekte oder die → Beziehung selbst gelöscht wird.

Syn. referential integrity (en).

Referenzattribut

→ Beziehung, die nur dem ersten → Objekt jedes Objektpaars der → Beziehung bekannt ist.

Syn. einseitige → Beziehung.

Referenzsystem

→ Koordinatensystem, das am Schluss einer Folge von → Koordinatensystemen und → Konversionen steht, in der genau ein → geodätisches Datum vorkommt, das den Anfang der Folge bildet.

Syn. Reference system (en), system de référence (fr).

Relation

Syn. für → Tabelle.

Replizieren

Kopieren, wobei das kopierte → Objekt nicht unabhängig vom Original verändert werden darf.

Bemerkung: Wird vor allem im Zusammenhang mit der → Nachlieferung verwendet.

Rolle

Bedeutung der → Objekte einer → Klasse in einer → Beziehung.

Bemerkung: In einer → eigentlichen Beziehung wird die R. jeder beteiligten → Klasse beschrieben durch ihren Namen, ihre → Stärke und ihre → Kardinalität. Ein → Referenzattribut beschreibt die R. der → Klasse mit diesem → Attribut. In der → Vererbungsbeziehung sind die R. implizit definiert.

Schema

Syn. für → Datenschema (Mehrzahl: Schemata oder neu auch Schemas).

Schemaelement

Teilschema eines konzeptionellen → Datenschemas, das einen Namen hat.

Bemerkung: Alle → Modellierungselemente sind S.

Schnittstelle

Mehrdeutiges Syn. für → Klassenschnittstelle, → Benutzerschnittstelle und → Datenschnittstelle.

Syn. interface (en, fr).

Schnittstellenklasse

Syn. für Klassenschnittstellen-Klasse. Def. siehe bei → Klassenschnittstelle.

Schnittstellensignatur

Beschreibung des Aufrufs einer → Operation, setzt sich zusammen aus dem Namen der → Operation, den → Datentypen und allenfalls Namen ihrer → Parameter und evtl. der Angabe eines Rückgabe-Datentyps.

Syn. Signatur (de).

Schweremodell

Beschreibung des Schwerefeldes der Erde.

Sekundärdatenbank

Kopie eines → Datenbankzustands einer → Primärdatenbank.

Bemerkung: Die S. befindet sich normalerweise nicht auf dem gleichen → System wie die → Primärdatenbank.

Sender

Def. siehe → Datentransfer.

Sicht

→ Klasse, deren → Objekte durch Kombination und Auswahl (genau: durch → Sicht-Operationen) aus → Objekten anderer → Klassen oder S. entstehen.

Syn. view (en).

Bemerkung 1: → Objekte einer S. sind nicht "originär" in dem Sinne, als sie nicht direkt einem Realweltobjekt entsprechen. Eine S. ist also eine Art virtuelle → Klasse.

Bemerkung 2: Siehe auch → Klassenelement.

Sicht-Operation

Vorschrift zur Def. neuer → Objekte aus den → Objekten von → Basis-Klassen bzw. → Basis-Sichten.

Bemerkung: S.-O. von → INTERLIS 2 sind Projektion (projection), Verbindung (join), Vereinigung (union), Zusammenfassung (aggregation) und Inspektion (inspection). Anschliessend kann die Objektmenge mit einer Selektion (selection) wieder eingeschränkt werden.

Sicht-Projektion

→ Klasse, deren → Objekte durch Ergänzung der → Attribute aus den → Objekten einer anderen → Klasse, → Sicht oder S.-P. ausgewählt werden. Insbesondere können weitere (virtuelle) → Attribute definiert werden, deren → Werte durch → Funktionen festgelegt werden.

Syn. viewprojection (en).

Bemerkung 1: → Erweiterungen von S.-P. sind möglich. Deren → Objekte bleiben aber immer Teilmengen der Objektmenge der → Basis-Klasse, → Basis-Sicht oder Basis-Sicht-Projektion.

Bemerkung 2: Siehe auch → Klassenelement.

Sichtbarkeitsbereich eines Namens

Menge der → Namensräume, aus denen der Name unqualifiziert referenziert werden kann. Der S des Namens besteht aus seinem → Definitionsbereich und aus den → Namensräumen seiner → Namenskategorie in allen → Modellierungselementen, die dem → Modellierungselement seines → Definitionsbereiches hierarchisch untergeordnet sind.

Bemerkung: Abgesehen vom → Namensraum seines → Definitionsbereichs kann ein Name in jedem → Namensraum seines S. neu definiert werden. Dieser → Namensraum wird damit neuer → Definitionsbereich des Namens. Dieser neue → Definitionsbereich und sein zugeordneter S. "überschreiben" einen Teil des ursprünglichen S. des Namens in dem Sinne, dass in diesem Teilbereich (der einen Teilbaum der Modellierungselemente-Hierarchie bildet) nur noch die neue Definition/Bedeutung des Namens gilt.

Signatur

Mehrdeutiges Syn. für → Schnittstellensignatur und → Grafiksignatur.

Signaturattribut

Syn. für → Zeichnungsregel.

Signaturenbibliothek

Sammlung von → Grafiksignaturen, die gemäss einem → Signaturenmodell strukturiert sind.

Syn. Symbolbibliothek (de), symbology library (en).

Bemerkung 1: Eine S. ist immer ein → Behälter, d.h. eine XML-Datei.

Bemerkung 2: Mit S. ist meistens eine konkrete, anwendungsspezifische Sammlung von → Grafiksignaturen gemeint.

Signaturenmodell

→ konzeptionelles → Schema, das → Grafiksignaturen und deren → Parameter beschreibt.

Syn. Symbologiemodell (de), symbology model (en).

Bemerkung 1: Für S. werden → Kontrakte verlangt.

Bemerkung 2: Anhang K zum INTERLIS 2-Referenzhandbuch enthält einen Vorschlag für ein erweitertes S.

Bemerkung 3: Siehe auch → Signaturenbibliothek.

Signaturobjekt

Syn. für → Grafiksignatur.

SN

Abk. für Schweizer → Norm.

Spezialisierung

→ Rolle der → Unterklasse in einer → Vererbungsbeziehung, oft auch Syn. für → Vererbung.

Syn. Erweiterung (de), extension (en), specialisation (fr).

Bemerkung 1: Siehe → Klassenspezialisierung und → Attributspezialisierung.

Bemerkung 2: Weil zur Beschreibung von → Klassen- oder → Attributspezialisierung mehr Text benötigt wird als bei der → Oberklasse oder beim Ausgangsattribut, spricht man oft auch von → Erweiterung (en: extension) statt von S.

Stabile Identifikation

→ Identifikation, die zeitunabhängig ist, d.h. während dem Lebenszyklus eines → Objektes nicht verändert werden kann. Die s. I. eines gelöschten → Objektes darf nicht mehr verwendet werden.

Bemerkung: Siehe auch → Objektidentifikation.

Standard

Syn. (de) für 'de facto' → Norm und (en) für 'de facto' oder 'de jure' → Norm.

Stärke

Bindung der Teile (Unter-Objekte der untergeordneten → Klasse) an das Ganze (Ober-Objekt der übergeordneten → Klasse) bei einer → eigentlichen Beziehung.

Struktur

Menge von → Strukturelementen mit gleichen Eigenschaften und → Operationen. Es sind nur → Operationen erlaubt, welche die → Daten der → Strukturelemente nicht verändern. Jede Eigenschaft wird durch ein → Attribut beschrieben, jede → Operation durch ihre → Schnittstellensignatur.

Bemerkung 1: S. kommen entweder innerhalb von LIST- oder BAG-Attributen vor (→ Unterstruktur) oder existieren nur temporär als Ergebnisse von → Funktionen.

Bemerkung 2: Siehe auch → Klassenelement.

Strukturattribut

→ Attribut mit dem INTERLIS 2-Datentyp BAG oder LIST.

Bemerkung: Im Gegensatz zur Def. einer → Komposition mit Hilfe der → Assoziationsklasse sind bei einem S. die → Strukturelemente aber nicht referenzierbar, d.h. haben ausserhalb des → Objektes, zu dessen S.-Wert sie gehören, keine → Identität.

Strukturelement

→ Daten eines Gegenstandes der realen Welt mit → Operationen, die mit diesen → Daten ausgeführt werden können, diese aber nicht verändern dürfen, und ohne → Objektidentifikation.

Bemerkung: Ein S. ist die → Instanz einer → Struktur.

Strukturierter Wertebereich

INTERLIS-Sprachelement zur Beschreibung zusammengesetzter → Attribute wie → Datum oder Zeit.

Symbol

Mehrdeutiges Syn. für → Grafiksignatur, Sprachsymbol oder semiotisches S.

Symbolbibliothek

Syn. für → Signaturenbibliothek.

Symbologie

Teilmenge von Elementen eines → kartografischen Zeichensystems bestehend aus → Grafiksignaturen, Schriften, Diagrammen, Halbtönen.

Bemerkung: siehe auch → Signaturenbibliothek.

Symbologiemodell

Syn. für → Signaturenmodell.

Synchronisation

Automatische und regelmässige Abgleichung der → Datenbankzustände zweier → Datenbanken.

System

Gesamtheit aller zu einer EDV-Anlage gehörenden Komponenten (Hardware und Software), die für einen bestimmten Zweck genutzt werden.

Tabelle

→ Klasse für deren → Objekte keine → Operationen explizit definierbar sind.

Syn. Relation.

Thema

Menge von → Klassen, deren → Daten in gewissem Sinne zusammengehören, die z.B. eine Beziehung haben, oder zu derselben Datenverwaltungsstelle gehören, oder einen ähnlichen Nachführungs-Rhythmus besitzen. Die → Instanzen von T. sind → Behälter.

Syn. topic (en), thème (fr).

Bemerkung 1: Ein T. wird mit → INTERLIS 2 als → Topic beschrieben. Ein → Topic wird in → UML durch ein → Package unterhalb eines → Datenmodells beschrieben, mit der zusätzlichen Bedeutung, dass dieses → Package (a) einen eigenen → Namensraum hat und (b) von anderen → Packages abhängig (z.B. erweitert) sein kann. Ein UML-Package, das einem T. zugeordnet ist, kann weitere (geschachtelte) → Packages enthalten.

Bemerkung 2: Beachte: Mit → Layer, dem im CAD-Bereich gebräuchlichen Ausdruck für "Ebene", wird eine Zusammenfassung grafischer → Daten bezeichnet. Ein T. kann mehrere (grafische) → Layer umfassen und zusätzlich strukturierte Sachdaten.

TID

Abk. für → Transferidentifikation.

Topic

Syn. für → Thema.

Transfer

Syn. für → Datentransfer.

Transferdatei

Zum → Datentransfer vorbereitete → Datei in geeignetem → Transferformat.

Transferformat

Gliederung einer → Transferdatei in Datenfelder.

Syn. Format.

Transfergemeinschaft

Gemeinschaft von → Sendern und → Empfängern, die sich an einem → Datentransfer beteiligen.

Transferidentifikation

Def. siehe bei → Identifikation.

Abk. TID.

Transformation

→ Abbildung von einem → Koordinatensystem (bzw. von seinem Raum) auf ein anderes → Koordinatensystem (bzw. auf dessen Raum), wenn die Abbildungsvorschrift (Formel) auf Grund von Annahmen (Hypothesen) festgelegt wird und die → Parameter durch meist statistische Analyse von Messungen in beiden → Koordinatensystemen ermittelt werden.

Syn. transformation (en/fr).

Tupel

Syn. für → Objekt.

Typ

Mehrdeutiges Syn. für → Datentyp (d.h. → Wertebereich), → Klassenschnittstelle und → Schnittstellensignatur.

Typnamen

→ Namenskategorie bestehend aus den Namen von → Themen, → Klassen, → Assoziationen, → Sichten, → Grafikdefinitionen, → Behältern, → Einheiten, → Funktionen, → Linienformtypen, → Wertebereichen, → Strukturen.

UML

Abk. für Unified Modeling Language.

Def. siehe www.omg.org/.

Unit

Syn. für → Einheit.

Unterklasse

Def. siehe → Vererbungsbeziehung.

Syn. Unterobjektklasse, Subobjektklasse (de), subclass (en), classe inférieure (fr).

Unterstruktur

→ Wertebereich, der mit Hilfe einer → Struktur definiert ist.

Bemerkung: Siehe auch → Strukturattribut.

Vererbung

→ Methode zur Def. von → Vererbungsbeziehungen zwischen → Oberklassen und → Unterklassen. Diese → Methoden sind → Klassenspezialisierung und → Attributsspezialisierung.

Syn. inheritance (en), héritage (fr).

Bemerkung 1: Anschaulich entsprechen → Unterklassen derselben Idee, sie haben dieselben Eigenschaften wie ihre → Oberklassen und spezialisieren diese.

Bemerkung 2: Man unterscheidet → Einfachvererbung und → Mehrfachvererbung. Bei einer → Einfachvererbung (en: single inheritance; fr: héritage singulaire) erbt eine → Unterklasse nur von einer direkten → Oberklasse. Bei der → Mehrfachvererbung erbt eine → Klasse von mehreren → Oberklassen.

Bemerkung 3: → INTERLIS 2 lässt nur einfache V. zu (wie z.B. Java).

Vererbungsbeziehung

→ Gerichtete Beziehung zwischen einer übergeordneten → Klasse, genannt → Oberklasse, und einer untergeordneten → Klasse, genannt → Unterklasse, definiert durch → Vererbung. Die → Rolle der → Oberklasse heisst → Generalisierung, die → Rolle der → Unterklasse heisst → Spezialisierung.

Bemerkung 1: Die → Objekte der → Oberklasse sind Verallgemeinerungen (→ Generalisierungen) der → Objekte der → Unterklasse. Die → Objekte der → Unterklasse sind Einschränkungen (→ Spezialisierungen, → Erweiterungen) der → Objekte der → Oberklasse.

Bemerkung 2: Die V. ist die Teilmengenbeziehung, die → Objekte der → Unterklasse bilden eine Teilmenge der → Objekte der → Oberklasse. Es handelt sich also bei den → Objekten der → Unterklasse nicht um neue → Objekte, sondern um einen Teil oder eine Unterteilung der → Objekte der → Oberklasse. Die beiden → Objekte eines Objektpaars der V. haben dieselbe → OID.

View

Syn. für → Sicht.

View-Projektion

Syn. für → Sicht-Projektion.

Vollständiger Datentransfer

→ Datentransfer eines vollständigen → Datenbankzustandes vom → Sender zum → Empfänger.

Wert

→ Datenelement eines → Wertebereichs.

Wertebereich

Menge gleichartiger → Datenelemente. Ein → Datenelement eines W. heisst → Wert.

Syn. Datentyp.

Bemerkung 1: Siehe auch → Basisdatentyp.

Bemerkung 2: Ein W. kann auch aus den → Strukturelementen einer → Unterstruktur bestehen.

Zeichnungsregel

Sprachelement einer → Grafikdefinition. Eine Z. ordnet (den → Objekten) einer → Klasse eine → Grafiksignatur zu, und legt die entsprechenden → Grafiksignatur-Argumente fest gemäss den → Attribut-Werten (d.h. → Daten) der → Objekte.

Syn. Signaturattribut.

Zielsystem

Syn. für → Empfänger.