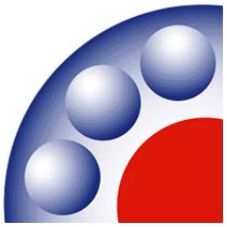
swisstopo
KOGIS

SumSuG Funktionen Test

(Systemunabhängiger, modellbasierter Strukturumbau von Geodaten, Funktionen Test)

Projektbericht



Impressum

Titel
SumSuG, Funktionen Test

Auftraggeber
swisstopo / KOGIS

Auftragnehmer
RMP Consulting AG

Status
Definitiv

Version / Datum
30.08.2022

Projektleitung
Hans Rudolf Gnägi

Autor
Hans Rudolf Gnägi

Zitierweise:
Gnägi, H. R.. *SumSuG, Funktionen Test (Systemunabhängiger, modellbasierter Strukturumbau von Geodaten, Funktionen Test).* Projektbericht, swisstopo / KOGIS 2022

Version	Änderung	Verfasser	Datum
1.1	Minimale Formatkorrekturen, Dank ergänzen	HRG	2022-11-22
1.0	Definitive Version	HRG	2022-06-26
0.92	Schlussredaktion	MM, HRG	2022-06-22
0.91	Version mit Änderungen gemäss Korrekturvorschlägen von PROK, sichtbar im Korrekturmodus von Word	PROK, HRG	2022-06-11
0.9	Version bereit zur Überprüfung durch PROK	HRG	2022-05-20
0.7	Technische Überarbeitung, Detailkorrekturen, alle Abbildungskommentare 1-zeilig, SW-Versionen aktuell	MM, HRG	2022-05-15
0.6	Test-Dokumentation für Anhang 7 bereinigen, Konsistenz aller Abbildungen sicherstellen	HRG	2022-05-14
0.5	Ergänzung Kap 3.1, Kap. 4 neu gliedern. Details zu Kap. 1, 2 (insbesondere Abb.1), 4.1, 4.2, 4.3, 4.4, 4.5, 4.6. Pendenzen 5.1/2, 5.3, Anhang 5 neu, Detailkorrekturen 5.4, 7.1, 7.2, 6.1/2/3/4, Abkürzungen und Glossar, Anhang 8, Konsistenzversuch für 7.2 (Erkenntnisse) und 7.3 (Handlungsfelder), Abb. 1 neu nummerieren für Ordnung der Doks in Anhang 8, diesen vervollständigen.	HRG	2022-03-31
0.4	Umbau von Kapitel 4.4: Umbaudefinition mit UMLT, Variante 1, Kritik, Variante 2: 4.4.1, 4.4.2, 4.4.3. Umbau von Kapitel 5: Zu 5.1 Existierende Funktionen neu 5.2 neu Variante 1, 5.3 neu Variante 2. Start Kap. 8 Literatur, Kap. 9 Abkürzungen und Glossar, sowie Anhänge 4, 5, 6.	HRG	2022-03-05
0.3	Technische Überarbeitung	MM	2021-08-31
0.2	Umstrukturierung gemäss Projektbedürfnissen, Aufbereiten und Einfüllen der bereits vorhandenen Resultate	HRG	2021-08-29
0.1	Ersterstellung Struktur auf Basis des SumSuG-Berichtes	HRG	2021-08-27

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Management Summary

Wie beim Projekt SumSuG [11] festgehalten ist, wird der Austausch von Geodaten im Zuge der starken Vernetzung immer wichtiger. Damit die Daten effizient genutzt werden können, ist es wesentlich, diese in einer standardisierten Struktur und in einem Standardformat bereitzustellen. Dieser Bedarf wurde in der Schweiz bereits vor geraumer Zeit erkannt. Die Standardmodelle sind mit den minimalen Geodatenmodellen (MGDM) verfügbar und als Standardformate werden INTERLIS 2-XML-Transferformat (XTF) oder die Geography Markup Language (GML) verwendet.

Ein wesentliches Hindernis ist der grosse Aufwand für die Bereitstellung standardisierter Geodaten aus bestehenden GIS, da die Daten üblicherweise in den GIS nicht gemäss MGDM strukturiert sind. Daher müssen die Daten umstrukturiert und passend formatiert werden.

Das Projekt SumSuG wollte primär zeigen, dass mit UMLT / ILIT und weiterer gratis und offencode Software eine objektorientierte, modulare und effiziente Methode für diesen Struktur- und Formatumbau existiert. Diese Methode wird beschrieben und es wird gezeigt, dass und wie sie funktioniert. Ferner werden Handlungsempfehlungen für die Behebung von Mängeln der aktuellen Tools abgegeben.

Als Anwendungsbeispiel wurde im Projekt SumSuG die Hochwasser-Gefahrenkarte des Kantons Aargau vom proprietären Modell und Format auf das entsprechende MGDM und das dazugehörige GML-Format umgebaut. Das ist ein kleines und überblickbares Beispiel. Dies genügt, um die Funktionsfähigkeit der Methode und die Details der einzelnen Schritte zu zeigen. Für den Strukturumbau werden dabei nur wenige einfache Umbaufunktionen verwendet.

Beim aktuellen Projekt „SumSuG Funktionen Test“ geht es nun darum, abzuklären, ob auch für umfangreichere Anwendungen beim Strukturumbau der vorhandene Funktionensatz genügt, insbesondere beim Strukturumbau geometrischer Objekte. Als Anwendungsbeispiel werden die Gleisdaten der Schweizerischen Bundesbahnen (SBB) vom proprietären Modell und Format mit SumSuG auf die Gleisdaten der Deutschen Bahn (DB) in deren proprietäres Modell und Format transformiert. Dabei werden die Untersuchungen beschränkt auf die Phase Strukturumbau vom Startmodell ins Zielmodell. Die Herleitung der beiden Modelle durch Analyse der entsprechenden proprietären Formate wird nur angedeutet und der Formatumbau vom proprietärem Startformat in den Standard-Startformat bzw. vom Standard-Zielformat in den proprietären Zielformat kann gemäss Projekt SumSuG erfolgen (siehe Kap. 6).

Hauptergebnisse des Projektes:

- Der aktuelle Satz von Umbaufunktionen genügt nicht, um in dem umfangreicheren Anwendungsbeispiel Gleisdaten aus SBB-Struktur in DB-Struktur zu transformieren. Es braucht mindestens noch Funktionen zum Zerlegen von Polylinien in deren Segmente, zur eindeutigen Zuordnung von Nummern zu Koordinaten für Punkte und zur Sammlung von Punktnummern zu Listen.
- Neue Funktionen `SplitLinToLinEl`, `PointNrGenerator`, `NodeNrGenerator`, `NumToListCollector` und `Incrementer` wurden im Detail analysiert, entworfen, in Java implementiert und an einem umfangreichen Datenbeispiel getestet.
- Da die im Projekt SumSuG eingesetzte UMLT/ILIT Implementierung hier nicht für die Tests zur Verfügung stand, wurde ein Ersatz-Testrahmen in Java realisiert. Dafür mussten auch die schon existierenden und benötigten Funktionen `OidGenerator`, `ValueMapper` und `StringConcatenator` mit Java programmiert werden.
- Die gesetzten Ziele sind erreicht: Der aktuelle Funktionsumfang von UMLT/ILIT ist beschrieben (Kap. 5.1), die Liste zusätzlicher Funktionen zur Transformation eines aktuellen umfangreichen Beispiels erstellt (Kap. 5.3), die Methodik zur Definition und Implementierung neuer Funktionen beschrieben (Kap. 5.4) und der Ergebnisbericht liegt vor.

2. Einleitung

2.1 Ausgangslage

Mit dem Projekt SumSuG wurde nur ein kleiner Teil der bei UMLT / ILIT definierten Strukturumbaufunktionen verwendet. Nämlich die folgenden: `Dissolver`, `ValueMapper` und `StringConcatenator`.

Man möchte zunächst wissen, ob die gesamte Liste der aktuellen Trafofunktionen genügt, um umfangreichere Strukturumbauten zu definieren, bei denen insbesondere auch Geometrieelemente umzubauen sind, und – wenn nein – welche weiteren Trafofunktionen minimal benötigt werden.

Die letzte Schlussfolgerung im Bericht des SumSuG-Projektes hat den Titel „Erweiterung des Umfangs an Abbildungsfunktionalitäten“ und lautet: Der UMLT-Editor und der UMLTApplier-Transformer implementieren derzeit nur solche Abbildungsfunktionalitäten (z.B. `Dissolver`, `StringConcatenator`, `ValueMapper`), die sich aus den Anforderungen der vorangegangenen Forschungsprojekte und des hier beschriebenen Projektes ergeben haben. Es ist zu erwarten, dass diese Menge an Funktionalitäten für einen umfassenden Praxiseinsatz noch nicht ausreichend ist und entsprechend erweitert werden muss.

2.2 Projektziele

Mit dem Projekt sollen folgende Ziele erreicht werden:

1. Liste aktueller Trafo-Funktionen
2. Liste minimal nötiger zusätzlicher Trafo-Funktionen
3. Beschreibung der Methode zur Definition und Implementierung weiterer Trafo-Funktionen
4. Zusammenfassung der Resultate in einem Ergebnisbericht

3. SumSuG-Prinzip: Systemunabhängig und modellbasiert

3.1 Detaillierte Funktionsweise, Übersicht

Wie im SumSuG-Bericht [11] zeigt Abbildung 1, welche Daten vorgegeben sind (weiß), welchen Input die fünf nötigen Aktivitäten (rot) brauchen, welche Ergebnis-Modelle und Ergebnis-Formatbeschreibungen sie liefern (blau), welche Prozesse entstehen (grün) und wie diese Prozesse die Daten umbauen (Datenfluss = dicke Pfeile).

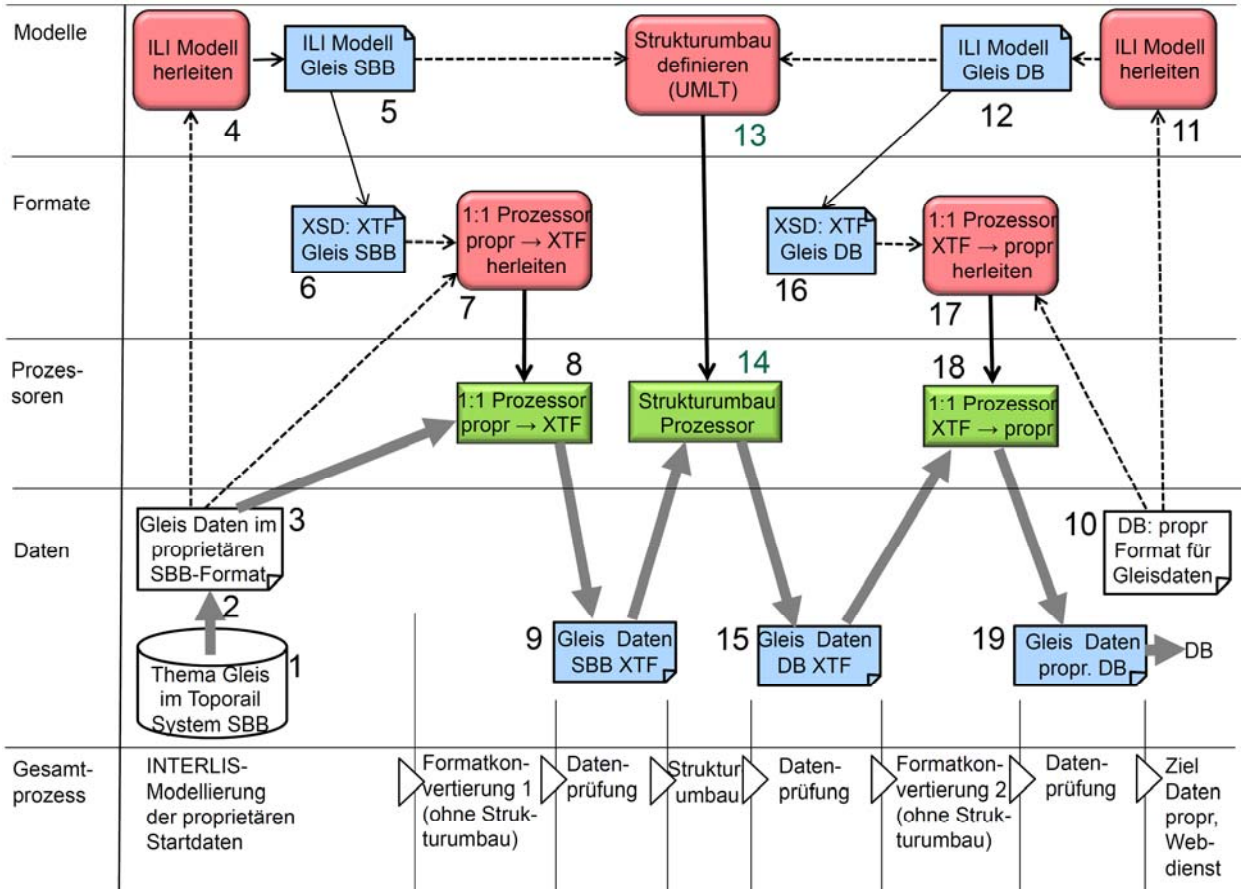


Abbildung 1: SumSuG: Detaillierte Funktionsweise, Prozessdetails und Ergebnisse

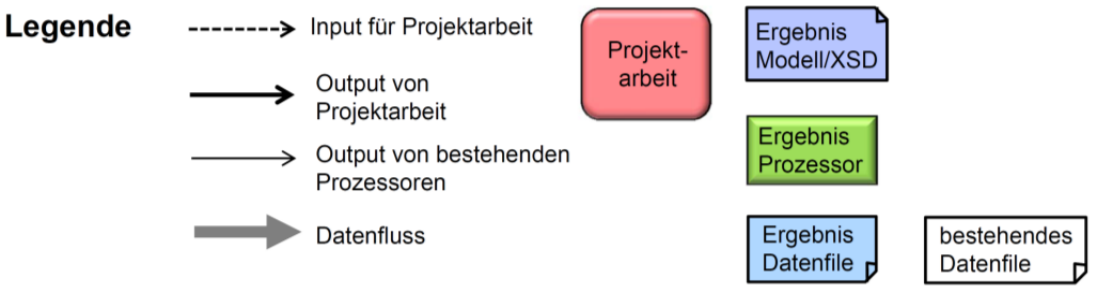


Abbildung 2: Verwendete Symbole in Abbildung 1

Nachfolgend wird in Klammern () auf die in obiger Abbildung verwendeten und mit Nummern versehenen Prozessschritte referenziert.

3.2 Beschreibung der Funktionsweise

Ausgangspunkt sind die Geodaten (3) der SBB-Gleise, wie sie vom gängigen System Toporail der SBB (1) in seinem proprietären Transferformat geliefert werden (2). Das proprietäre Datenmodell (5) der Datenstruktur hinter diesem Datenformat wird in INTERLIS formuliert (4). Die technische Datenstruktur des gängigen Systems selbst muss nicht in Betracht gezogen werden. Analog wird auf Seite der DB-Zieldaten mit Hilfe von deren Formatbeschreibung (10) das proprietäre Datenmodell (12) hinter dem Zieldatenformat hergeleitet (11).

Als Nächstes gilt es, das proprietäre Format der SBB-Daten (3) in das standardisierte Transferformat XTF umzuwandeln. Dazu liefert uns der INTERLIS-Compiler aus dem proprietären SBB-Datenmodell (5) die Formatbeschreibung als XML-Schema (6), mit welchem die Umformatierung entworfen werden kann (7). Das resultierende Umformatierungsprogramm nennen wir einen 1:1-Prozessor (8), da nur das Format, nicht aber die Struktur, verändert wird.

Eine Prüfung stellt sicher, dass die vom 1:1-Prozessor umgewandelten Quelldaten (9) dem standardisierten Transferformat (6) entsprechen. Eine weitere Prüfung zeigt auf, ob die SBB-Daten (9) konform zum proprietären SBB-Modell (5) sind. Es können daher Mängel in der Datenqualität aufgedeckt werden, z.B. Verletzungen von Wertebereichen von Attributen, Eindeutigkeitsbedingungen, Beziehungsstrukturen sowie Anforderungen an die Geometrie.

Der Strukturumbau vom proprietären Start-Modell (5) ins Ziel-DB-Modell (12) wird auf konzeptioneller Ebene objektorientiert definiert. Dieser Schritt wird ausserhalb des gängigen Systems vorgenommen, weshalb SumSuG als systemunabhängige Methode bezeichnet wird. Der für diesen Strukturumbau entwickelte UMLT-Editor erlaubt eine graphische Eingabe des Strukturumbaus und speichert die Umbauregeln in der standardisierten Umbausprache UMLTtex [7].

Aus der Umbaudefinition (13) wird automatisch das Programm zum Umbau der Daten erzeugt (14). Bei der aktuellen Implementierung von UMLT / ILIT wird dafür eine FME-Workbench generiert, welche die Quelldaten (9) gemäss SBB-Modell in DB-Zieldaten (15) gemäss DB-Modell umbaut. Sowohl Input als auch Output sind im standardisierten Transferformat XTF codiert.

Analog der Prüfung vor dem Strukturumbau werden die Zieldaten (15) nach dem Strukturumbau auf ihre Konformität zur Formatbeschreibung (16) und zum Modell (12) geprüft.

Eine zweite Umformatierung wandelt die Zieldaten (15) im XTF-Transferformat in das proprietäre DB-Format (19) um, damit sie vom DB-System übernommen werden können. Der INTERLIS-Compiler liefert dafür aus dem DB-Modell (12) die Formatbeschreibungen als XML-Schema (16) für das XTF-Transferformat. Davon ausgehend und mit der Beschreibung des proprietären DB-Formats (10) wird die Umformatierung entworfen (17). Das entstehende Programm (18) ist wiederum ein 1:1-Prozessor, der die Daten ohne Strukturumbau umformatiert.

Auch nach der Durchführung dieser zweiten Formatkonvertierung durch den zweiten 1:1-Prozessor muss eine Prüfung sicherstellen, dass die umgewandelten Zieldaten (19) der Beschreibung ihres Transferformats (10) entsprechen. Eine weitere Prüfung gegenüber dem Modell (12) ist nicht nötig, da der 1:1-Prozesseor keine Änderung an der Struktur vornahm.

Kernpunkt der vorliegenden Arbeit ist die Umbaudefinition (13). Damit dies sinnvoll und korrekt erfolgt, braucht es für die praktische Überprüfung der Funktionen auch den 1:1-Prozessor vom proprietären SBB-Format in ein INTERLIS-Format (7, 8, 9), den Strukturumbau-Prozessor (14) sowie – für dessen interne DB-Zieldaten – einen 1:1-Prozessor (Writer) auf ein lesbares INTERLIS-Format (15). Details dazu im Kapitel 6.

4. SumSuG-Anwendung: Gleisdatenumbau SBB nach DB

Als Anwendungsbeispiel dienen die Gleisdaten – einerseits der SBB, andererseits der DB. Der Blick in die Realität zeigt, dass es sich dabei offensichtlich um dieselben Realweltobjekte handelt. Hingegen sind die – historisch gewachsenen – Datenstrukturen zur Verwaltung der entsprechenden Daten unglaublich verschieden, daher bestens geeignet für einen SumSuG-Test.

4.1 Startdaten Gleise der SBB

Analyse des Toporail Transferformats

Gemäss der Toporail Formatbeschreibung enthalten die Gleisdaten der SBB die exakte und umfassende Gleisgeometrie. Deren Hauptelemente sind Weichen mit Weichenpunkten und Weichenlinien sowie Gleisabschnitte. Da es hier primär um Funktionen zum Geometrieumbau geht, werden vor allem Geometrieobjekte und Geometrieattribute in Betracht gezogen und zur Vereinfachung auch nur Punkte und Verläufe, letztere ohne Überhöhungen, Längsprofile und Gradienten. Auch beschränken wir uns auf 2D-Koordinaten.

Abbildung 3 zeigt einen Ausschnitt aus der SBB-Startdatei (entsprechend (3) in Abbildung 1, ab Zeile Nr. 2279, siehe Anhang 7). Jede Zeile beginnt mit einem einzelnen Zeichen, der Zeilen-Identifikation. Eine Zeile mit Zeilen-Identifikation x nennen wir kurz Zid_x.

```
2279 6 I STR     8         30 CM N  EW    IV    300          1:12            F  H  L  STAR G    0 9005     0
2280 7  1  706663.90278  279055.38906 365.07012       0.000
2281 7  2  706646.54790  279083.77979 365.07012       0.000 WES
2282 7  3  706645.11181  279082.81773 359.77718       0.000 WEA
2283 7 01  706657.39457  279066.03579   0.00000
2284 7 43  706641.50209  279087.76207  62.43100
2285 7 42  706643.35428  279089.00257 262.43100
2286 7 53  706632.86568  279102.65911   0.00000       3.500
2287 8 C 13  706663.90278  279055.38906   24.94237 365.07012    -300.000
2288 8 D 13  706650.02468  279076.10528    8.31825 359.77718
2289 6 I STR  9010         16 CM    GE
2290 7  1  706570.32307  279200.76075 143.53530       0.000
2291 N I N  JE      1       1                  JE      1       2
2292 1 D CM      322090  685097.17325  278778.46491   26.54600  80.08532
2293 2 V  685097.17325  278778.46491  437.87722   26.54600    1.20000
2294 3  685097.17325  278778.46491   26.54600    0.0    0.0 120
2295 N I N  JE      2       1                  JE      2       2
2296 1 D CM      322090  685122.43097  278786.63424   41.59458  80.08532
2297 2 V  685122.43097  278786.63424  437.90907   41.59458    1.20000
2298 3  685122.43097  278786.63424   41.59458    0.0    0.0 120
2299 N I N  JE      2       2                  JE      8       2
```

Abbildung 3: Ausschnitt aus SBB-Startdatei

Der Dateiausschnitt enthält folgende Zeilentypen:

- Zid_6 mit thematischen Daten von Weichen.
- Zid_7 mit Koordinaten von Weichenpunkten. Von diesem interessieren uns nur die den Weichenenden entsprechenden Weichenpunkte (mit Leerstelle an Pos 3 und 1, 2, 3 an Pos 4).
- Zid_8 sind Weichenlinienstücke.
- Zid_N sind Startzeilen von Gleisabschnitten und
- Zid_1 sind deren Stützpunkte samt Verbindungsinformation zum folgenden Stützpunkt.

Details sind in Anhang 7 Abschnitt 3_SBB_GleisdatenProprFmt zu finden.

INTERLIS-Modell herleiten

Bei der SBB war kein konzeptionelles Datenmodell der Gleisdaten erhältlich, weshalb dieses im Rahmen des Projektes auf Basis der Quelldaten hergeleitet wurde.

Das hergeleitete Datenmodell SBB beinhaltet ein Thema (Topic) SBB_gleisnetz und darin 7 verschiedene Klassen. Entsprechend den oben erwähnten beiden Hauptelementen heissen die Hauptklassen Weiche (mit WeichenPunkt und WeichenLinie) sowie Gleisabschnitt. Das Datenmodell enthält noch drei weitere Klassen, nämlich Gradient, Ueberhoehung und Laengsprofil. Diese drei Klassen bieten keine weiteren wesentlichen Probleme und werden daher für den Strukturumbau nicht berücksichtigt Abbildung 4 zeigt das Datenmodell in UML. Das INTERLIS-Modell ist in Anhang 2 zu finden.

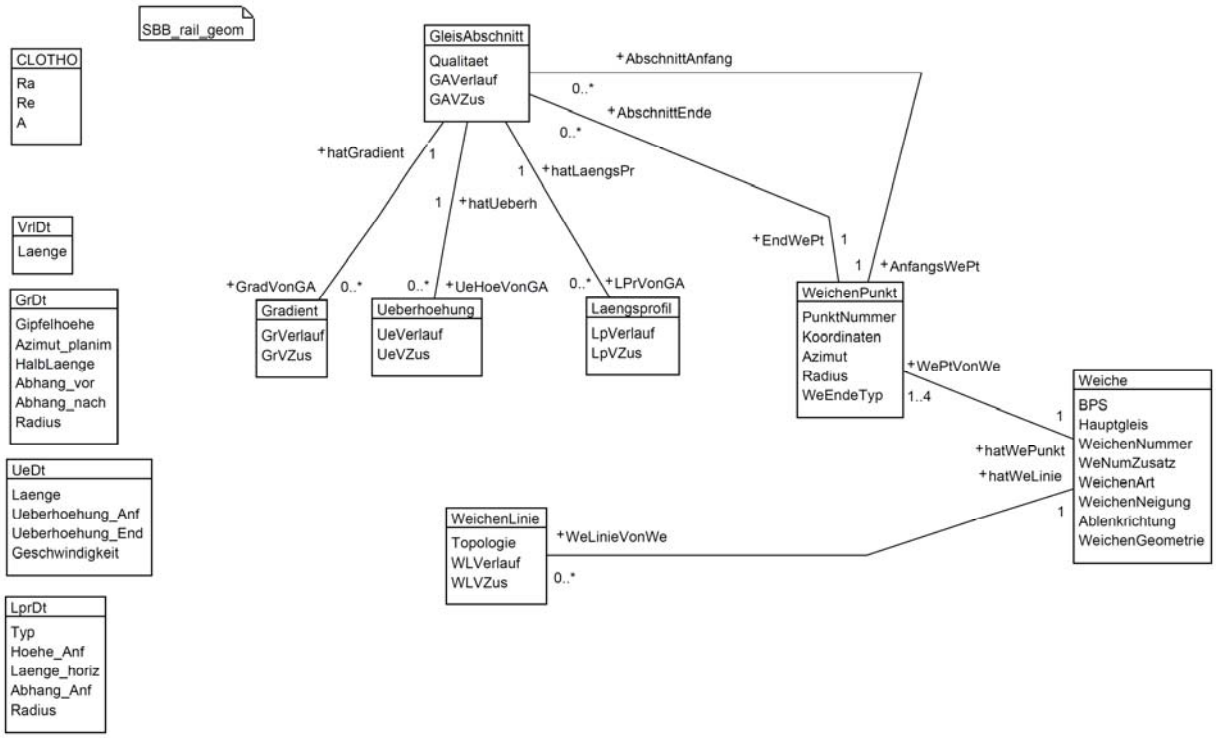


Abbildung 4: UML-Diagramm der SBB-Startdatei (SBB_rail_geom.uml, Modell SBB)

Die «Klassenkästchen» am linken Rand von Abbildung 4 sind Symbole für INTERLIS-Strukturen. Da bei der SBB (und auch bei der DB) bei den Gleisen Klothoiden vorkommen, muss dafür ein Datentyp definiert werden. Dazu dient die Struktur CLOTHO. Die übrigen Strukturen sind Attributstypen für Zusatzdaten zu den Stützpunkten der Polylinien namens GAVerlauf, GrVerlauf, UeVerlauf und LpVerlauf. Diese Zusatzdaten befinden sich in den Listen GAVZus, GrVZus, UeVZus und LpVZus, sind also modelliert, obschon sie – wie oben erwähnt und begründet – nicht verarbeitet werden. Da die Daten nur einen Ausschnitt des Gleisnetzes betreffen und daher nicht vollständig konsistent sind, wurden gewisse Kardinalitäten von Beziehungen abgeschwächt, um zu viele Fehlermeldungen zu vermeiden. So ist etwa nicht jeder Weichenpunkt Start- oder Endpunkt eines Gleisabschnittes (Rolle AbschnittEnde von Klasse GleisAbschnitt mit Kardinalität {0..*} statt {1..*}). Da es zu vielen Weichen keine Weichenlinien gibt und da viele Weichen nicht 3 oder 4 Weichenpunkte haben (als Enden von einfachen Weichen oder Kreuzungsweichen) sondern nur 1 oder 2, liegt die Vermutung nahe, dass mit „Weiche“ ein allgemeinerer Gleisübergang bezeichnet wird. Dies wäre abzuklären (siehe Kap. 7.2). Der Vollständigkeit halber sei erwähnt, dass einige Attribute im UML-Diagramm fehlen, die jedoch im INTERLIS-Modell noch als Kommentare vorkommen. So WePunktNum_Anf, WePunktNum_End in Klasse GleisAbschnitt, die mit Beziehung gelöst wurden, sowie WePunktID und WeicheID, deren Bedeutung nicht klar ist.

4.2 Zieldaten Gleisstruktur der DB

Analyse des DB Transferformats und der Datenbank

Die Deutsche Bahn unterhält eine „Datenbank zur Verwaltung von geometrischen und topologischen Daten von Gleisanlagen der Deutschen Bundesbahn“. Die Datenbasis besteht aus geometrischen sowie topologischen Daten. Für eine saubere Modellierung der Gleisnetzdaten werden beide Datentypen benötigt. Die geometrischen Daten beinhalten die einzelnen Punkte und Gleisabschnitte, in den topologischen Daten sind zum Beispiel die Informationen enthalten, wie die einzelnen Abschnitte zu einer Gleiskante (auch Strecke genannt) zusammengesetzt werden. Die folgende Abbildung 5 (entsprechen dem Punkt (10) aus Abbildung 1) zeigt einen Ausschnitt aus der Beschreibung des proprietären Transferformats GND der DB Gleisdaten gemäss [12]. Dabei geht es um die allgemeinen Punktdaten (Datenobjekte der Klasse `Punkt-Allgemein`).

Feld	Feldname	von	bis	Format	Ein-gabe	Beschreibung
	SATZTYP	1	2	I2	11	Satzart
1	PAD	3	13	A11	X	Punktadresse
2	PART	14	17	A4	X	Punktart bzw. Objektklasse DB-GIS
3	VERMART	18	20	I3	*	Vermarkungsart
4	STABIL	21	21	I1	*	Stabilitätskennzeichen
5	STATION	22	34	F13.3	X	Bei PSTRRIKZ<>4: Strecken-Kilometrierung des Punktes Bei PSTRRIKZ=4: optionaler Stationswert (= Längenbezug zur zugehörigen Kante) Standardwert = '0', Werte >'0' werden von DB-GIS originär verwaltet und fortgeführt
6	PDATUM	35	42	A8	X	Datum der letzten Änderung
7	PBEARB	43	50	A8	X	Bearbeiter der letzten Änderung
8	PAUFTR	51	58	A8	*	Auftrag
9	PPROG	59	66	A8	*	erzeugendes Programm
10	PTEXT	67	86	A20	*	Kommentar
11	PSTRECKE	87	90	I3A1	X	Bei PSTRRIKZ<>4: Streckenbezeichnung Bei PSTRRIKZ=4: Gleisnummer Bahnhofsgleis oder '0'
12	PSTRRIKZ	91	91	I1	X	Richtungskennzeichen
13	PID					Punkt –ID des Referenzpunktes, bei Auswahl PART=DBRF wird PID-Feld geöffnet und muß gefüllt werden mit einer ID aus ID-Katalog.

Abbildung 5: ASCII-Schnittstelle der Satzart 11 - Punkt Allgemein

In Anhang 7 Abschnitt 10_DB-GleisdatenProprFmt befinden sich Details zu den DB-Daten aller verwendeten Klassen.

INTERLIS Modell herleiten

Abbildung 6 zeigt als Resultat der Aktion „ILI-Modell herleiten“ ((11) in Abbildung 1) die Struktur der DB Gleisdaten grafisch im UML-Diagramm ((12) in Abbildung 1). Dieses Datenmodell als INTERLIS-Text findet sich in Anhang 3. Das Modell enthält auch die Klasse `PunktAllgemein`, die der ASCII-Schnittstelle von Abbildung 5 entspricht. Beim Entwurf des Datenmodells wurden verschiedene Attributsnamen übernommen (z.B. PAD), andere wurden etwas aussagefähiger umformuliert (z.B. `Punktart` statt PART, `Richtung` statt PSTRRIKZ, `BStreckenId` (= Bahn-Strecken ID) statt PSTRECKE [als `TEXT*13` statt `I3A1`]). Zwecks vorläufiger Vereinfachung wurde PAD als Zahlenbereich `1..999999` statt als `TEXT*11` definiert, ferner die `Punktart` als Aufzählytp mit besser verständlichen Texten statt mit Abkürzungen von 4 Zeichen (z.B. mit `LageElPt` statt PHPG). Wie bei den SBB-Startdaten wurden auch hier zu Gunsten der Geometrie fast alle thematischen Attribute nicht modelliert (z.B. Vermarkungs-

art, Stabilitätskennzeichen, Datum und Bearbeiter der letzten Änderung, Auftrag, erzeugendes Programm, Kommentar). Auch wird auf die kurvilineare Koordinate verzichtet (DB-Attribut `STATION`, Strecken-Kilometrierung des Punktes, M-Wert).

Die Beziehung `GleisPunkt` zwischen den Klassen `GleiskanteStrecke` und `PunktAllgemein` wurde vorerst nicht realisiert, da die Liste `LaPt` von allen Lagepunkten eines `GleiskanteStrecke` Objektes ihre Punktnummer `PAD` enthält. Daher ist die Kardinalität der Rolle `KanteMitPunkt` jetzt `{0..4}` statt `{1..4}` und von der Rolle `PunkteAufKante` jetzt `{0..*}` statt `{2..*}`.

Auch im DB-Modell braucht es eine Struktur, das Klassenkästchen `PAdrS` links aussen. In INTERLIS 2.3 wird eine Nummernliste (wie Attribut `LaPt` in Klasse `GleiskanteStrecke`) beschrieben durch `LIST {} OF` Strukturelemente (`PAdrS`) mit Nummer (`AdS`).

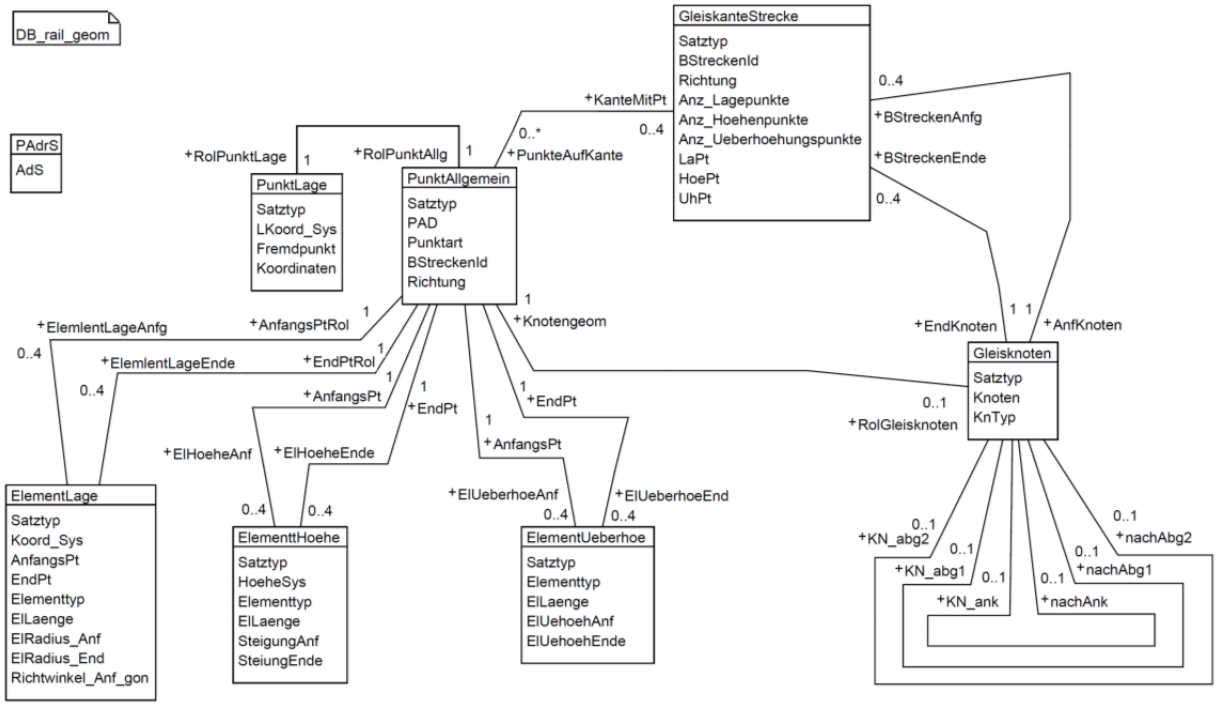


Abbildung 6: UML-Diagramm der DB-Zieldatei (`DB_rail_geom.uml`, Modell DB)

Ergebnisse der Analyse:

Der Vergleich des Modells DB mit dem Modell SBB zeigt markante Unterschiede. Während bei der Punktklasse `WeichenPunkt` der SBB die Punktkoordinaten mit dem Attribut `Koordinaten` unmittelbar in der Punktklasse vorkommen, sind bei der DB Koordinaten und thematische Attribute auf zwei verschiedenen Klassen `PunktAllgemein` und `PunktLage` verteilt, die durch eine 1:1-Beziehung miteinander verknüpft sind.

Bei SBB und DB ist die Abstraktion des Gleisverlaufs eine Polylinie. Beim SBB-Modell gibt es bei Polylinien-Objekten (wie `WeichenLinie` oder `GleisAbschnitt`) ein Verlauf-Attribut (wie `WLVerlauf` oder `GAVerlauf`) vom INTERLIS-Typ `POLYLINE`, d.h. mit einer Folge von Stützpunkten samt Verbindungsgeometrie. Darin sind die Stützpunkte nicht nummeriert, es sind lediglich deren Koordinaten vorhanden. Ganz anders bei der DB-Struktur. Dort gibt es bei jedem Polylinien-Objekt (wie `GleiskanteStrecke`) die Anzahl der Stützpunkte (`Anz_Lagepunkte`) und auch eine Liste der Stützpunkte, die aber nur die Nummern dieser sog. Lagepunkte enthält (Attribut `LaPt`). Deren Koordinaten findet man über die Nummer via Klasse `PunktAllgemein` in der Klasse `PunktLage`. Die Verbindungs-Geometrien der Polylinien befinden sich

segmentweise in der Klasse `ElementLage`. Dort findet sich für zwei benachbarte Stützpunkte (sog. Lagepunkte) der Polylinie die detaillierte Beschreibung für die Geometrie der Verbindung zwischen diesen beiden Lagepunkten, d.h. ob es sich dabei um ein Geradenstück, um einen Kreisbogen oder um einen Klothoidenbogen handelt (Attribut `Elementtyp`) samt den entsprechend benötigten Detailparametern. Zu finden sind die `ElementLage`-Objekte von einem `GleiskanteStrecke`-Objekt mit den Punktnummern der Liste `LaPt`. Das sind die Nummern der Anfangspunkte der `ElementLage`-Objekte. Das bedeutet, dass die `ElementLage`-Objekte schliesslich in der DB-Datenbank nicht nur über ihre Objektidentifikation gefunden werden können, sondern auch über die Nummer ihres Anfangspunktes (Attribut `AnfangsPt`). Da im DB-Modell alle Lagepunkte eine Nummer haben, müssen alle Stützpunkte von Polylinien eine eindeutige Nummer erhalten.

Noch zur Klasse `Gleisknoten` der Knotenpunkte: Bei der GND-Spezifikation von Gleiskante Strecke (GS) gibt es die Attribute AKNOTEN und EKNOTEN. Dafür findet man in der GND-Spezifikation von Gleisknoten (KN) drei Knotentypen, nämlich Weiche, Gleisende und Streckenwechsel, d.h. pro Weiche gibt es einen Knoten. Ferner gibt es dort drei Attribute KN0, KN1, KN2 für Anschlussknoten des ankommenden, des ersten und des zweiten abgehenden Gleises. Die Erläuterung für Weichen lautet: KN0 ist der Knoten vor dem Weichenanfang, KN1 derjenige hinter dem Weichenende, KN2 ist nicht weiter erläutert. Diese Knotenabstraktion scheint sich an der einfachen Weiche zu orientieren. Dann ist aber unklar, wie Kreuzungsweichen in mehrere einfache Weichen aufzuteilen sind. Im DB-Modell sind diese Attribute für Anschlussknoten als Beziehungen der Klasse `Gleisknoten` auf sich selbst formuliert. Da die erwähnten Unklarheiten nicht beseitigt werden konnten, wurde auf die Transformation der Klassen `Weiche` und `Weichenlinie` verzichtet und nur beim Umbau der Klasse `Gleisabschnitt` werden Objekte der Klasse `Gleisknoten` hergestellt, als Anfangs- bzw. End-Knoten des entstehenden `GleiskanteStrecke` Objektes.

4.3 Strukturumbau mit UMLT/ILIT, Prinzip der Abbildungsregeln

Der Strukturumbau mit UMLT/ILIT umfasst zurzeit im Wesentlichen zwei Schritte:

1. Die Definition der Transformation auf Modellebene zwischen dem Quellmodell SBB und dem Zielmodell DB mittels der Abbildungssprache UMLT (UML-Transformationsdefinition).
2 Automatischer Umbau der mittels UMLT definierten Transformationsregeln in konkrete Transformationsprogramme, die dann den eigentlichen Strukturumbau der Daten durchführen.

Beide Schritte werden im SumSuG Projektbericht [11] näher beschrieben. Im aktuellen Funktionen-Test-Bericht ist mit Schwergewicht der erste Schritt beschrieben. Da die aktuelle UMLT/ILIT-Implementierung nicht für Tests verwendet werden konnte, wird in Kap. 6 auch der zweite Schritt beschrieben.

Im ersten Schritt wird systemunabhängig konzeptionell definiert, wie das Quellmodell (hier das proprietäre Modell `SBB` aus 4.1) in das Zielmodell (hier das proprietäre Modell `DB` aus Kap. 4.2) umzubauen ist. Dabei geht es in diesem Projekt primär um den Umbau geometrischer Attribute. Mit dem Umbau nichtgeometrischer (sog. thematischer) Attribute hat sich das Projekt SumSuG [11] ausführlich beschäftigt. Auch wird wegen der Unklarheiten bei den Knoten (siehe Kap. 4.2) vorerst darauf verzichtet, die Details des Strukturumbaus von SBB-Weichen und -Weichenlinien in die entsprechenden DB-Elemente auszuführen (mit UMLTtex). Damit die Übersicht vollständig bleibt, werden die entsprechenden Strukturumbauelemente in den UMLT-Diagrammen beibehalten (insbesondere die TrafoAction `WLWE2GS` samt ihren Links zu Pins und anderen TrafoActions).

Die konzeptionelle Definition wird mittels der Abbildungssprache UMLT definiert. Für eine Einführung in die Sprache UMLT sei auf [10], [6] und [9] verwiesen. Praktische Erfahrungen, die hier mit UMLT gemacht wurden und die zum Verständnis von UMLT beitragen können, sind in Anhang 6 zusammengestellt.

4.4 Version 1: TrafoActions `WP2PAPLGK` und `GA2GS`

Im Projekt wurde mit dieser Version 1 gestartet, die sich – wie man in Kap. 4.5 sieht – als ungenügend erwies. Zum besseren Verständnis des Strukturumbaus mit UMLT/ILIT und der Gründe für die Entwicklung des definitiven Funktionensatzes (siehe Kap. 4.6) wird auch die Version 1 der Transformation und der dazu benötigten Funktionen beschrieben.

UMLT erlaubt die Definition der Abbildungsregeln in grafischer Form; die erste Version (V1) der vollständig definierten Transformationsdefinition ist in Abbildung 7 dargestellt.

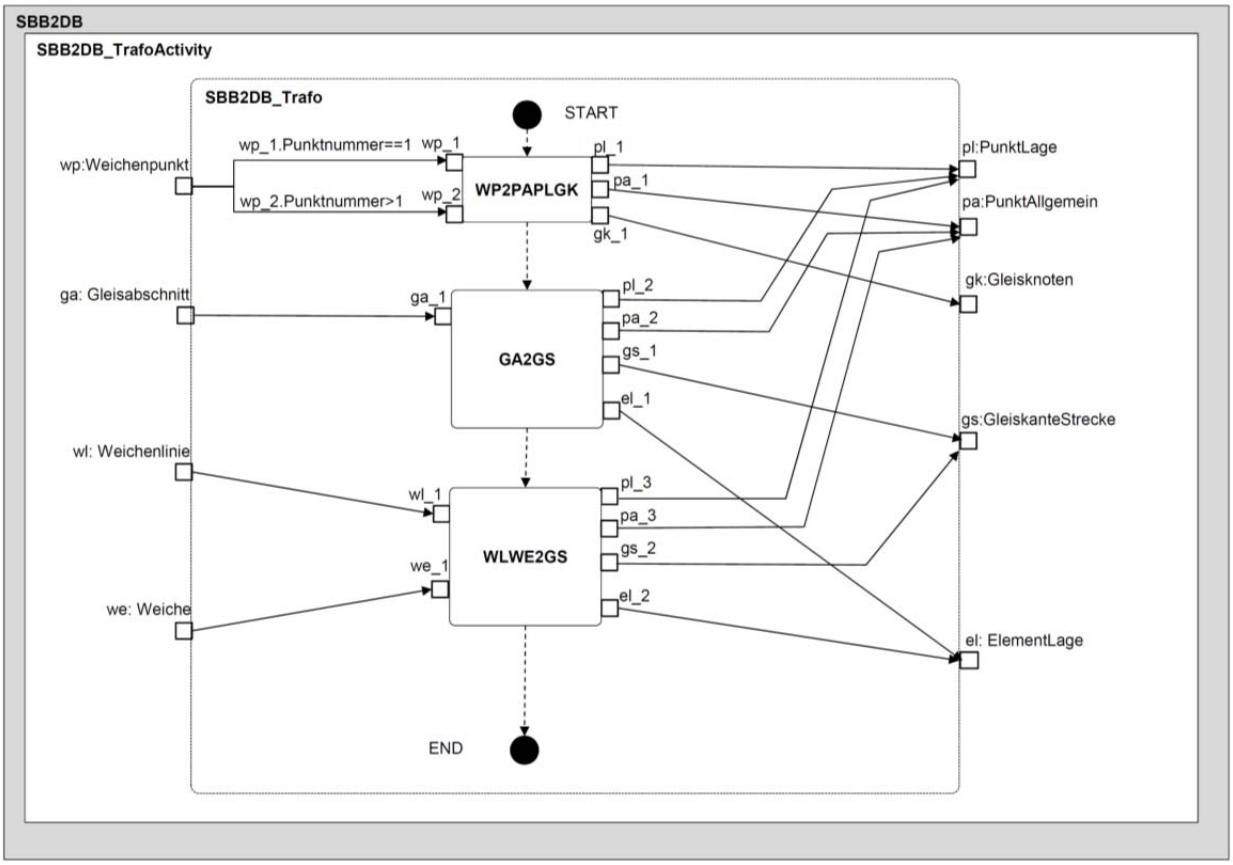


Abbildung 7: UMLT-Transformation (Version 1) vom Quellmodell SBB zum Zielmodell DB

In die Transformation (hier `SBB2DB_Trafo`) hinein fliessen Objekte der Klassen `Weichenpunkt`, `Gleisabschnitt`, `Weichenlinie`, und `Weiche` des Quellmodells, aus der Transformation heraus fliessen Objekte der Klassen `PunktLage`, `PunktAllgemein`, `Gleisknoten` , `GleiskanteStrecke` und `ElementLage` des Zielmodells.

Die Transformationsdefinition enthält drei TrafoActions (`WP2PAPLGK`, `GA2GS` und `WLWE2GS`), in welchen die Wertzuweisungen von Attributen aus den Quell- zu den Zielklassen definiert werden. Diese Wertzuweisungen sind in Abbildung 8 dargestellt für `WP2PAPLGK` und `GA2GS`. Die Transformationsdefinition könnte darüber hinaus sogenannte ValueMaps enthalten, mittels derer die Abbildungen zwischen Werten aus vordefinierten Wertetabellen (Aufzähltypen, Enumerations) festgelegt werden können.

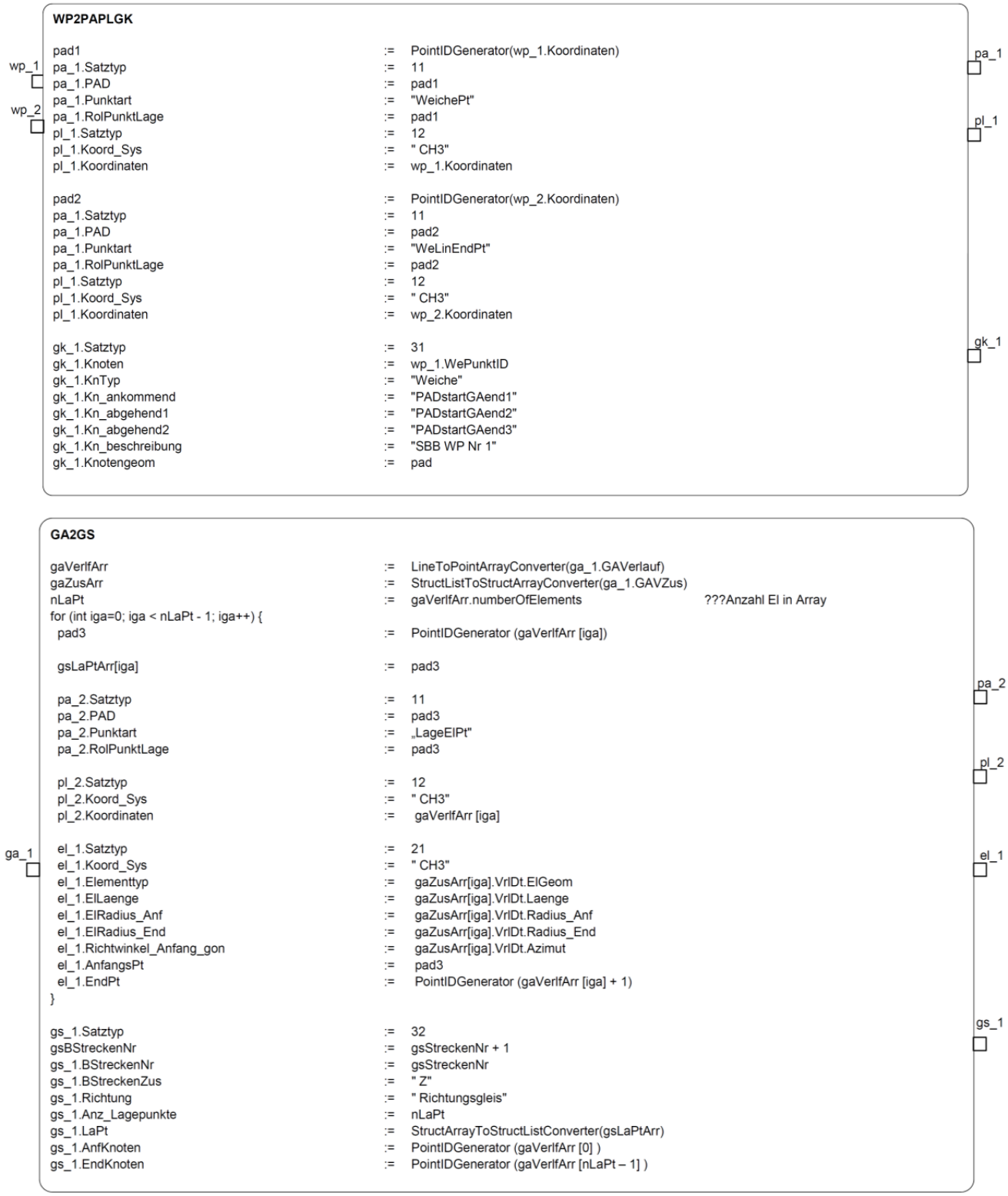


Abbildung 8: Wertzuweisungen in den TrafoActions (Version 1) mit HUTN UMLTtex

Für diese Wertzuweisungen (sog. Assignments) allenfalls mit Funktionsaufrufen (und ValueMaps) kommt ein für Menschen lesbarer und verständlicher textueller Formalismus zum Einsatz (sog. HUTN := Human Useable Textual Notation). Dafür wäre eigentlich die aus INTERLIS weiterentwickelte Sprache ILIT geeignet. Da aber ILIT noch nicht präzis definiert war, wurde für den aktuellen Prototyp des UMLT-Editors eine spezielle HUTN entwickelt und eingesetzt. Wir

nennen sie UMLTtex. Über den Zusammenhang zwischen einzelnen Sprachelementen von UMLTtex und INTERLIS (bzw. ILIT) siehe Anhang 6.

Die UMLT-Transformationsdefinition kann nun grafisch mittels des UMLT-Editors definiert werden. Anschliessend ist je nach zur Verfügung stehenden Tools die Transformationsdefinition im Format ILIT oder XMI zu exportieren. Das Format XMI wird benötigt, wenn die Transformationsdefinition im zweiten Schritt in FME eingelesen werden soll. Das ist für den aktuellen Stand der Software der Fall.

4.5 Probleme der Trafo-Definition Version 1

Dass bei den Wertzuweisungen in UMLTtex der TrafoAction `GA2GS` von Version 1 eine DO-Schleife gebraucht wird, ist ein Widerspruch zu Grundsatz 2 von UMLT, wonach der Benützer nicht komplex programmieren müssen soll (siehe Anhang 6). Diese DO-Schleife wird benötigt, weil aus der INTERLIS-Polylinie (etwa des Gleisverlaufs im SBB-Modell) die einzelnen Segmente herausgeholt werden müssen – die etwa von der DB-Klasse `GleiskanteStrecke` benötigt werden. Damit eine solche DO-Schleife nicht durch den UMLT-Benützer programmiert werden muss, sind Strukturumbauten, wie diese Aufteilung einer INTERLIS-Polylinie in Segmente, als Funktion zur Verfügung zu stellen. Wie Version 2 zeigt, braucht es dazu nicht nur andere Funktionen, sondern es ist dann auch die TrafoAction-Struktur anzupassen.

Weiter werden in beiden TrafoActions Werte zugewiesen zu Rollen-Attributen von Beziehungen. In der 1:1-Beziehung `PunktLagePAD` der Klassen `PunktAllgemein` und `PunktLage` (siehe Anhang 3) hat die Rolle der Klasse `PunktLage` den Namen `RolPunktLage`. Da in der DB-Datenstruktur das hauptsächlich verwendete Referenzattribut eines Punktes seine Nummer ist (in der Klasse `PunktAllgemein`), werden die Koordinaten des Punktes normalerweise von der Nummer aus gesucht. Daher muss bei jedem Objekt der Klasse `PunktAllgemein` ein Pointer gespeichert sein auf das entsprechende Objekt der Klasse `PunktLage`, das die zur Nummer passenden Koordinaten enthält. Bei der Herleitung der Beschreibung des INTERLIS-XML Transferformats XTF erfindet daher der Compiler ein zusätzliches Attribut in der „Haupt“-Klasse `PunktAllgemein` mit dem Namen `RolPunktLage`, das in den Daten mit der Objektidentifikation des Koordinaten-Objekts aus Klasse `PunktLage` gefüllt werden muss, das dem Nummern-Objekt aus Klasse `PunktAllgemein` entspricht. Das wird in der TrafoAction (Version 1) `WP2PAPLGK` so gelöst, dass dem Rollenattribut `RolPunktLage` die eben berechnete eindeutige Punktnummer `pad1` zugewiesen wird. Das ist aber nicht die Objektidentifikation des Koordinatenobjekts. Bei der Suche nach derselben und entsprechenden Fragen an UMLT/ILIT-„Natives“ zeigte sich, dass die Zuordnung der Rollenattributwerten zu den entsprechenden Zielobjekten durch UMLT automatisch erfolgt auf Grund des Start- und Zielmodells (Details dazu sind in Anhang 6 zu finden). Aus der TrafoAction `WP2PAPLGK` können daher die unnötigen (und auch falschen) Zuordnungen von Rollenattributwerten entfernt werden, ebenso in der TrafoAction `GA2GS`.

Die Programmierung, die zu vermeiden ist, betrifft auch, dass in beiden TrafoActions Hilfsvariable eingeführt werden wie `pad1` (in `WP2PAPLGK`) bzw. `gaVerlfArr` und `nLaPt` (in `GA2GS`).

Die bei der Besprechung der Klasse `Gleisknoten` der DB in Kapitel 4.2 erwähnten noch fehlenden Wertzuweisungen in UMLTtex für den Umbau von SBB-Weichen in die entsprechenden DB-Elemente betreffen die TrafoAction `WLWE2GS` in Abbildung 7.

Dem allgemeinen Problem, dass in ein Zielmodell nur Daten abgefüllt werden können, welche im Startmodell auch vorhanden sind, entspricht die Tatsache, dass das SBB-Quellmodell nicht alle Informationen beinhaltet, welche beim DB-Zielmodell vorkommen. Daher können nicht alle Klassen und Attribute des DB-Zielmodells mit Quelldaten gefüllt werden. Fehlen zu gewissen

Zieldaten entsprechende Quelldaten, dann werden für die Zieldaten Defaultwerte eingesetzt, falls solche existieren.

4.6 Version 2: TrafoActions `WA2PAPL`, `GA2LINEL`, `LINEL2PAPLEL`, `PAL2GSKN`

Die mit UMLT in grafischer Form definierte Transformationsdefinition der zweiten Version (V2) von Abbildungsregeln ist in Abbildung 9 dargestellt. In die Transformation hinein fliessen wie bei der ersten Version (V1) Objekte der Klassen `Weichenpunkt`, `Gleisabschnitt`, `Weichenlinie`, und `Weiche` des Quellmodells, aus der Transformation heraus fliessen Objekte der Klassen `PunktLage`, `PunktAllgemein`, `Gleisknoten` , `GleiskanteStrecke` und `ElementLage` des Zielmodells. Die Transformationsdefinition enthält fünf TrafoActions (`WP2PAPL` [als korrigierte Version von `WP2PAPLGK`], `GA2LINEL`, `LINEL2PAPLEL`, `PAL2KNGS` und `WLWE2GS`, die letzte nach wie vor ohne Details), in welchen die Wertzuweisungen von Attributen aus den Quell- zu den Zielklassen definiert werden. Diese Wertzuweisungen sind in Abbildung 10 dargestellt für die Trafo Actions `WP2PAPL`, `GA2LINEL`, `LINEL2PAPLEL` und `PAL2KNGS`. Die Wertzuweisungen für `LINEL2PAPLEL` enthalten darüber hinaus eine ValueMap, mittels der die Abbildung zwischen Werten aus vordefinierten Wertetabellen (Aufzähltypen, Enumerations) festgelegt werden kann. Hier wird durch die ValueMap `Connection-Type_ValueMap` der Aufzähltyp `LinienGeomTyp` des SBB-Modells in den Aufzähltyp `Verbindungsgeom` des DB-Modells abgebildet (Abbildung 11).

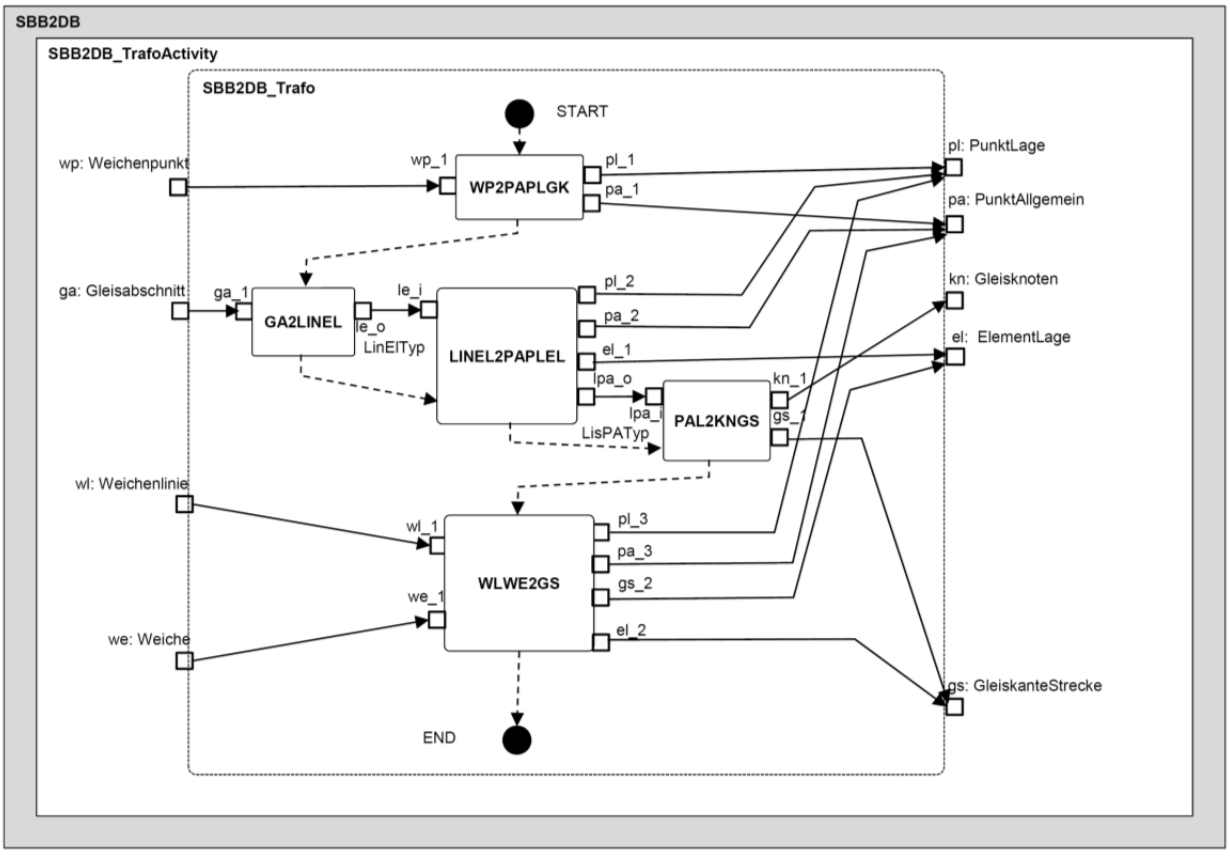


Abbildung 9: UMLT-Transformation (Version 2) vom Quellmodell SBB zum Zielmodell DB

Die in Kapitel 4.4 beschriebenen Mängel der Trafodefinition Version 1 sind hier wie folgt behoben:

- Auf die Programmierung einer DO-Schleife kann jetzt verzichtet werden. Die Zerlegung einer INTERLIS-Polylinie in Segmente erfolgt mit der neuen Funktion `SplitLinToLinEl`.
- Es zeigte sich, dass für die Berechnung der Objekte der DB-Klassen `PunktLage`, `PunktAllgemein`, `Gleisknoten` und `ElementLage` von den einzelnen Segmenten des Verlaufs eines SBB-Gleisabschnittes die Daten vorliegen müssen. Wie die Abbildung 9 zeigt, braucht es deshalb eine eigene TrafoAction `GA2LINEL`, welche diese Segmente über ihren Output-Pin `le_o` liefert. Diese TrafoAction `GA2LINEL` besteht aus einer einzigen Wertzuweisung in UMLTtex (Abbildung 10) mit dem Aufruf der Funktion `SplitLinToLinEl`.
- Die Codierung der Beziehungen durch Zuordnung von Werten zu Rollen-Attributen wurde aus allen TrafoActions herausgestrichen. Dadurch sind auch gerade – als eigentlich unerwünschte Programmierelemente – die Hilfsvariablen `pad1`, `pad2`, `pad3` verschwunden.
- Auch in Version 2 müssen alle vorkommenden Punkte eine eindeutige Punktnummer haben, auch die Stützpunkte von SBB-Polylinien, die zunächst nur Koordinaten haben. Für die Berechnung dieser Nummern braucht es auch in Version 2 die Funktion `PointNrGenerator`.
- Da ein Objekt der DB-Klasse `GleiskanteStrecke` für den entsprechenden `Gleisabschnitt` aus dem SBB-Modell eine Liste dieser Punktnummern braucht in der gleichen Reihenfolge, in welcher bei der SBB-Polylinie die entsprechenden Koordinaten vorkommen, braucht es noch eine weitere neue Funktion `CollectorNumToList`, welche diese Punktnummern sammelt zu einer richtig geordneten Liste.

- Bei Version 1 war ein Problem gar nicht aufgetreten, nämlich das von Zusatzattributen, die weder im Start- noch im Ziel-Modell definiert sind, deren Werte aber von einer TrafoAction an eine andere übergeben werden sollten. Denn es gab keinen Bedarf für den Austausch solcher Zusatzattribute zwischen TrafoActions (siehe Abbildung 7).

  In Version 2 sind nun an zwei Stellen Daten von einer TrafoAction zu einer anderen via Output- und Input-Pin zu übertragen, die weder im Start- noch im Ziel-Modell beschrieben sind. Nämlich zwischen `GA2LINEL` und `LINEL2PAPLEL` via die beiden Pins `le_o` und `le_i` und zwischen `LINEL2PAPLEL` und `PAL2GS` via die Pins `lpa_o` und `lpa_i` (Abbildung 9). Zunächst wurde daran gedacht, das Start- oder das Ziel-Modell zu ergänzen um neue Klassen mit denjenigen Attributen, deren Werte übertragen werden sollen, so dass dann über die Pins die entsprechenden Objekte zu transferieren wären. Dagegen sprechen drei Dinge:

  1. Korrektur der Modelle darf nicht nötig sein, erst recht nicht um Daten, die gar nicht zu den Modellen sondern zu deren Transformation gehören.
  2. Objekte brauchen OIDs, die dann auch noch konsistent verwaltet werden müssten.
  3. Widerspruch zu Grundsatz 3 von UMLT, dass beim Datentransfer über Pins zwischen TrafoActions keine Schnittstellenprogrammierung nötig sein darf.

  In beiden erwähnten Fällen handelt es sich beim Datentransfer zwischen TrafoActions um den Transfer von Funktions-Resultaten, zwischen `GA2LINEL` und `LINEL2PAPLEL` um das Ergebnis der Funktion `SplitLinToLinEl`, zwischen `LINEL2PAPLEL` und `PAL2GS` um das Ergebnis der Funktion `CollectorNumToList`. Alle Funktionen sind im INTERLIS-Modell `ILITfct` definiert und damit auch alle Attributstypen ihrer Parameter. In diesem Modell, das wie das Start- und Zielmodell im Textbereich von UMLT (mit UMLTtex oder ILIT) importiert werden muss, können allenfalls auch Attributstypen definiert und damit der Transformationsbeschreibung von UMLT zugänglich gemacht werden, die nicht Parameter von Funktionen sind. Diese können dann auch zum Datenaustausch via Pins zwischen TrafoActions verwendet werden.

- Derselbe Punkt kann in den SBB-Daten mehrfach mit denselben Koordinaten vorkommen, zum Beispiel als Endpunkt eines Gleisabschnittes und als Weichenpunkt. Die aus der konzeptionellen Umstrukturierungsdefinition mit UMLT hergeleitete Software zum physischen Datenumbau muss sicherstellen, dass so ein Punkt immer dieselbe Punktnummer zugeteilt bekommt und dass genau ein Punktobjekt den Klassen `PunktLage` und `PunktAllgemein`

übergeben wird. Der Entwerfer des Strukturumbaus mit UMLT sollte sich nicht um solche Probleme kümmern müssen.

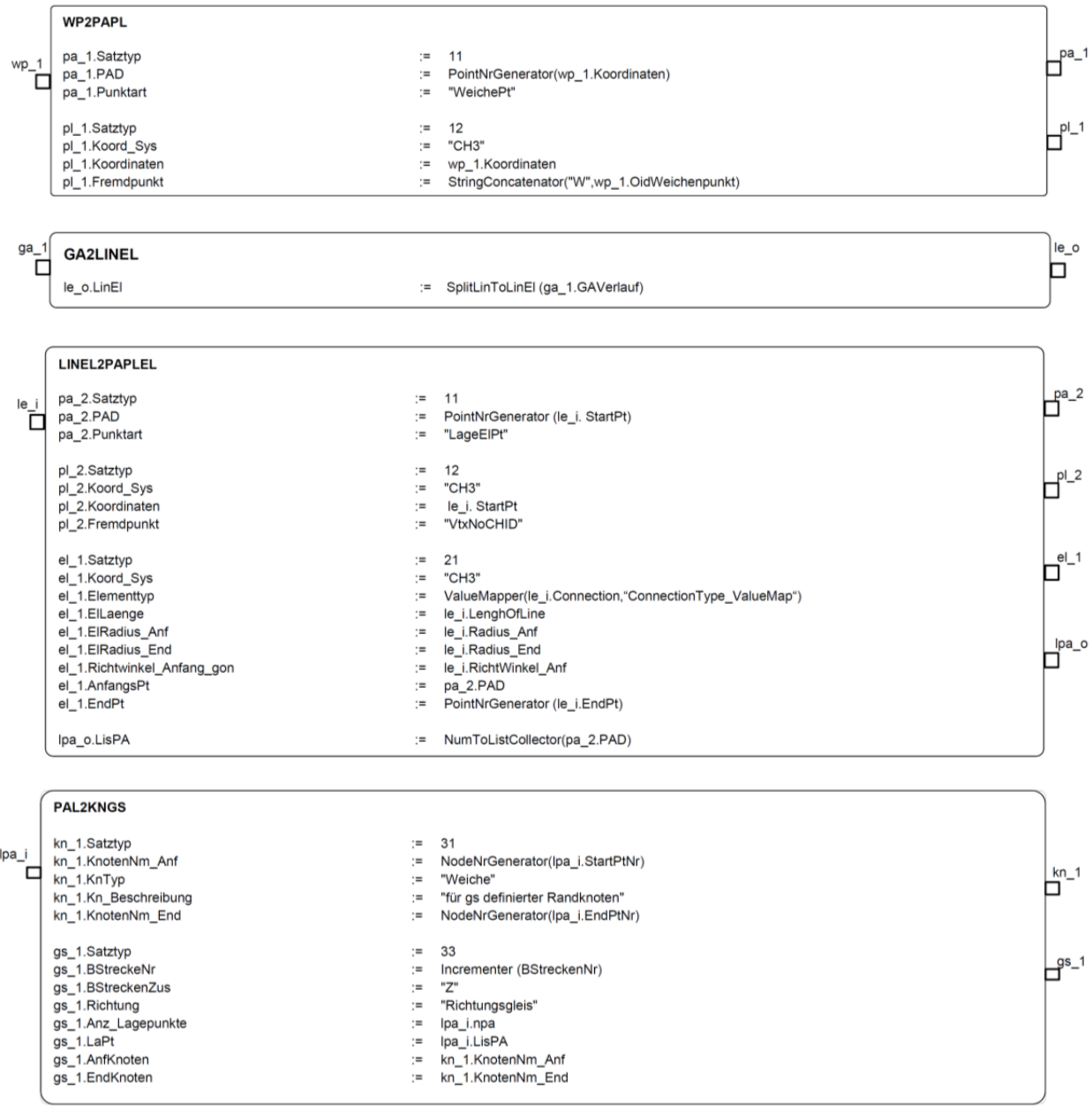


Abbildung 10: Wertzuweisungen der TrafoActions (Version 2) in UMLTtex

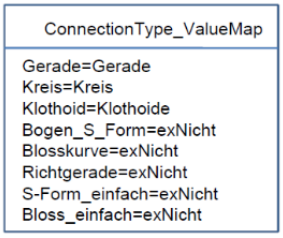


Abbildung 11: ValueMap ConnectionType_ValueMap

5. Transformationsfunktionen

5.1 Existierende Funktionen

Funktion	Signatur	Beschreibung
Accumulator	`Accumulator (attributeToAnalyze : String, attributeToGroupBy : String, typeOfStatistics : String) : float`	Berechnet Statistiken für das Attribut `attributeToAnalyze` der eingehenden Objekte. Der Parameter `attributeToGroupBy` gruppiert Objekte, um Statistiken für Objektgruppen zu berechnen. Die zu berechnende Statistik wird mittels des Parameters `typeOfStatistics` festgelegt (Mögliche Werte sind z.B. `sum, mean, minimum, maximum`),
AreaCalculator	`AreaCalculator (geometry : Polygon) : int`	Berechnet den ganzzahligen Flächeninhalt von Objekten. Mittels des Parameters `geometry` wird das Attribut übergeben, das die Geometrie vom Typ `Polygon` (d.h. INTERLIS-Typ AREA oder SURFACE) enthält.
Dissolver	`Dissolver (geometry : Polygon[], groupBy : String) : Polygon[]`	Vereinigt flächenhafte Objekte, die gemeinsame Randlinien aufweisen, zu gemeinsamen Flächenobjekten. Mittels des Parameters `geometry` wird das Attribut übergeben, das die Geometrie vom Typ `Polygon` (d.h. INTERLIS-Typ AREA oder SURFACE) enthält. – Der Parameter `groupBy` dient zur Gruppierung von Flächenobjekten.
PointInPolygon Extractor	`PointInPolygonExtractor (geometry : Polygon) : Point`	Bestimmt einen Punkt im Innern flächenhafter Objekte. Mittels des Parameters `geometry` wird das Attribut übergeben, das die Geometrie vom Typ `Polygon` (d.h. INTERLIS-Typ AREA oder SURFACE) enthält.
PolygonToLine Converter	`PolygonToLineConverter (geometry : Polygon) : Line`	Zerlegt die Randlinien flächenhafter Objekte in Polylinien. – Mittels des Parameters `geometry` wird das Attribut übergeben, das die Geometrie vom Typ `Polygon` (d.h. INTERLIS-Typ AREA oder SURFACE) enthält.
PolygonToNon RedundantLine Converter	`PolygonToNonRedundantLine Converter (geometry : Polygon) : Line`	Zerlegt die Randlinien flächenhafter Objekte in Polylinien, aber liefert von doppelt vorkommenden Polylinien nur eine. – Mit dem Parameter `geometry` wird das Attribut übergeben, das die Geometrie vom Typ `Polygon` (d.h. INTERLIS-Typ AREA oder SURFACE) enthält.
StringConcatenator	`StringConcatenator (sourceAttribute1 : String, sourceAttribute2 : String) : String`	Die Zeichenketten `sourceAttribute1` und `sourceAttribute2` werden zusammengehängt zu einer einzigen Zeichenkette.
SubStringer	`SubStringer (sourceAttribute : String, startIndex : int, endIndex : int) : String`	Von der Zeichenkette `sourceAttribute` werden die Zeichen Nr `startIndex` bis Nr `endIndex` herauskopiert.
UUIDGenerator	`UUIDGenerator() : String`	Berechnet einen neuen Universally Unique Identifier.
ValueMapper	`ValueMapper (sourceAttribute : String, valueMapName : String) : String`	Dem Wert des Attributs `sourceAttribute` wird gemäss der Zuordnungstabelle namens `valueMapName` ein neuer Wert zugeordnet.

Abbildung 12: Die aktuellen UML/UMLTtex Transformationsfunktionen (FME-implementiert)

5.2 Neue benötigte Funktionen, Version 1

Wie den TrafoActions der Version 1 (in Abbildung 8) zu entnehmen ist, werden die folgenden zusätzlichen Transformationsfunktionen benötigt. In den DB-Daten muss jeder Punkt eine Nummer haben, das ist aber z.B. für Stützpunkte von Polylinen-Daten der SBB nicht der Fall. Daher braucht es die neue Funktion `PointNrGenerator`, welche über ein ganzes Operat eindeutige Punktnummern generiert. Ferner braucht es einen individuellen Zähler, z.B. um die Anzahl der Segmente der Polylinie einer DB-Gleisstrecke festzustellen. Dazu dient die neue Funktion `Incrementer`. Um aus SBB-Polylinien die benötigten Segmente für die DB bestimmen zu können, muss man von der Polylinie den Array der Stützpunkte der Polylinie zur Verfügung haben. Dies liefert die Funktion `LineToPointArrayConverter`. Da zunächst nicht klar war, ob innerhalb des DO-Loops der TrafoAction GA2GS bei der Programmierung mit UMLTtex auf die Elemente einer INTERLIS-`LIST` zugegriffen werden kann, wie auf Elemente eines Arrays, wurden noch 2 neue Funktionen eingeführt, um eine INTERLIS-`LIST` von STRUCTUREs in einen Array von STRUCTUREs umzubauen mit `StructListToStructArrayConverter` bzw. umgekehrt mit `StructArrayToStructListConverter` einen Array von STRUCTUREs in eine INTERLIS-`LIST` von STRUCTUREs.

Funktion	Signatur (in UMLTtex)	Beschreibung
LineToPointArrayConverter	`LineToPointArray Converter (geometry : Line) : Point[]`	Umwandlung des Objekts `geometry` vom Typ `Line` (d.h. INTERLIS-Typ `POLYLINE`) in einen Array (= Folge) der Stützpunkte vom Typ `Point` (d.h. INTERLIS-Typ `COORD`).
PointNrGenerator	`PointNrGenerator (geometry : Point) : String`	Prüft, ob der Punkt vom Typ `Point` (d.h. INTERLIS-Typ `COORD`) mit seinen Koordinaten bereits vorhanden ist in der Hash-Datei der Punktkoordinaten. Falls Ja, wird die bereits existierende Punkt Nr zurückgegeben. Falls Nein, wird eine neue Punkt Nr erzeugt, zurückgegeben und mit den Koordinaten des neuen Punktes in der Hash-Datei eingefügt.
StructListToStructArrayConverter	`StructListToStructArray Converter (structList : LIST OF {} STRUCTURE) : STRUCTURE[]`	Umwandlung des Objekts `structList` vom Typ `LIST OF {} STRUCTURE` in einen Array der Struktur-Elemente vom Typ `STRUCTURE`.
StructArrayToStructListConverter	`StructArrayToStructList Converter (structArray : STRUCTURE[]) : LIST OF {} STRUCTURE`	Umwandlung des Objekts `structArray` vom Typ `STRUCTURE[]` in eine Liste der Struktur-Elemente vom Typ `LIST OF {} STRUCTURE`..
Incrementer	`Incrementer (counter : int) : int`	Der Zähler `counter` vom Typ `int` wird beim ersten Aufruf mit dieser `counter`-Variable (wie prüfen?) auf 1 gesetzt und bei jedem weiteren Aufruf um 1 erhöht. Dieser Wert wird zurückgegeben.

Abbildung 13: Neue UMLT/UMLTtex-Funktionen (Version 1), nicht FME-Implementiert

5.3 Neue benötigte Funktionen, Version 2

Auch für die TrafoActions der Version 2 (in Abbildung 10) werden zusätzliche Transformationsfunktionen benötigt. Wie bei Version 1 muss in den DB-Daten jeder Punkt eine Nummer haben (u.a. sind Stützpunkte von Polylinen-Daten der SBB nicht nummeriert). Daher braucht es auch die neue Funktion `PointNrGenerator`, welche über ein ganzes Operat eindeutige Punktnummern generiert. Wie bei Version 1 braucht es den individuellen Zähler (um die Anzahl der Segmente einer DB-Gleisstrecke festzustellen). Dazu dient die neue Funktion `Incrementer`.

Hingegen braucht es nun nicht mehr die drei in Version 1 eingeführten neuen Funktionen `LineToPointArrayConverter`, `StructListToStructArrayConverter`, `StructArrayToStructListConverter`.

In Version 2 werden jedoch drei andere neue Funktionen benötigt: `SplitLinToLinEl`, `CollectorNumToList`, `NodeNrGenerator` (siehe Abbildung 14 und Anhang 4 mit Definition der Funktionssignaturen in INTERLIS sowie Begründungen in Kap. 4.6 nach Abbildung 9).

Funktion	Signatur (in UMLTtex)	Beschreibung

PointNrGenerator	`PointNrGenerator`
	`(geometry : Point)`
	`: int`
		Prüft, ob `geometry` vom Typ `point` bereits vorhanden ist in der HashMap der Punktkoordinaten. Falls Ja, wird die bereits existierende Punkt Nr Typ `int` zurückgegeben. Falls Nein, wird eine neue Punkt Nr erzeugt, zurückgegeben und mit den Koordinaten des neuen Punktes in der HashMap eingefügt.

NodeNrGenerator	`NodeNrGenerator`
	`(refPointNr : int)`
	`: int`
		Prüft, ob die Referenzpunkt-Nummer `refPointNr` des neuen Knotens bereits vorhanden ist in der HashMap der Referenzpunkt-Nummern. Falls Ja, wird die bereits existierende Knoten Nr Typ `int` zurückgegeben. Falls Nein, wird eine neue Knoten Nr erzeugt, zurückgegeben und mit der Referenzpunkt Nr des neuen Knotens in der HashMap eingefügt.

Incrementer	`Incrementer`
	`(counter: String)`
	`: int;`
		Initialisieren und Erhöhen des ganzzahligen Zählers `counter` um 1. Beim ersten Aufruf der Funktion mit dem Zählernamen `counter` wird dieser Name gespeichert und der entsprechende Zähler auf 1 gesetzt. Bei jedem weiteren Aufruf wird dieser Zähler um 1 erhöht.

SplitLinToLinEl	`SplitLinToLinEl`
	`(geometry: Line,`
	`nSegm: int)`
	`: LinElTyp[];`
	
	`Type LinElTyp =`
	`coord_Sys: String;`
	`StartPt: Point;`
	`EndPt: Point;`
	`gmtyp: String;`
	`elLaenge: float;`
	`elRadius_Anf: float;`
	`elRadius_End: float;`
	`elRichtWinkel_Anf_gon:`
	`float;`
		Zerlegt eine Polylinie in Segmente. Mit dem Parameter `geometry` wird das Attribut übergeben, das vom Geometrietyp `Line` ist (INTERLIS Typ `PolylineGeneral`, d.h. `POLYLINE` mit Geradenstücken, Kreisbogen und Klothoiden). Der zweite Parameter `nSegm` enthält die Anzahl der Segmente in der Linie `geometry` (jetzt nötig aus programmtechnischen Gründen!!). Der Rückgabeparameter ist ein Array (INTERLIS `LIST OF`) vom Typ Segment (= Linienelement), der durch den UMLTtex-Type (INTERLIS `STRUCTURE`) `LinElTyp` definiert ist.

CollectorNumToList	`CollectorNumToList`
	`(number: int)`
	`: int[];`
		Sammelt ganzzahlige Werte `number` zu einem Array. Nach Eingabe des letzten Wertes `number` ist als Rückgabewert der Array (INTERLIS `LIST OF`) der gesammelten ganzzahligen Werte verfügbar.

Abbildung 14: Neue UMLT/UMLTtex-Funktionen (Version 2), nicht FME-implementiert

5.4 Definition und Implementierung neuer Trafo-Funktionen

Falls sich bei einem Strukturumbau entsprechende Bedürfnisse ergeben so dass zu einem späteren Zeitpunkt noch weitere Trafo-Funktionen eingeführt werden müssten, empfiehlt sich das folgende Vorgehen:

1. Zunächst ist das Bedürfnis nach einer neuen Funktion zu formulieren, und zwar mit einer möglichst präzisen Beschreibung (Beispiele in Kap. 5.2 und 5.3, Abbildung 13 und Abbildung 14).
2. Dann gilt es abzuklären, ob nicht eine Kombination bereits existierender Funktionen der Beschreibung genügt. Falls ja, muss lediglich diese Kombination beschrieben werden. Falls nein, ist eine entsprechende neue Funktion herzuleiten.
3. Sobald das Konzept der neuen Funktion vorliegt, ist es zweckmässig, die INTERLIS Signatur der neuen Funktion zu formulieren (wie in Anhang 4), damit Klarheit herrscht über Eingabeparameter und Rückgabewerte.
4. Die Implementierung einer neuen Funktion richtet sich nach dem Stand der UMLT/ILIT Software. Für die UMLT/UMLTtex-Implementierung, die an der TU München zum Strukturumbau von proprietären Modellen nach INSPIRE-Methoden zum Einsatz kommt, braucht es eine entsprechende FME-Workbench (siehe dazu [8]). In unserem Java-Projekt “UMLT-Funktionen-Testbed“ als Testrahmen-Ersatz braucht eine neue Funktion eine neue Java-Klasse. Als Beispiel ist im Anhang 5 die Funktion `PointNrGenerator` zu finden.

6. Test mit Testrahmen-Ersatz

6.1 Testrahmen ohne UMLT/ILIT-Software

Ziel des Tests ist, zu zeigen, dass mit den vorhandenen und neu vorgeschlagenen Funktionen die gegebenen SBB-Daten gemäss der vorgesehenen UMLT/UMLTtex-Strukturumbau-Definition in die DB-Struktur transformiert werden können.

Beim SumSuG-Projekt [11] kamen für den Strukturumbau Softwarewerkzeuge zum Einsatz, die im Rahmen der Forschungsprojekte „Modellbasierter Ansatz für den Web-Zugriff auf verteilte Geodaten am Beispiel grenzübergreifender GIS-Anwendungen (mdWFS)“ [1] und „Prototypische Transformation von Geodaten nach INSPIRE in der grenzüberschreitenden Region Bodensee“ [7] entwickelt worden waren. Das waren die Folgenden:

- UMLT-Editor: Editor zur grafischen Definition der Umbaufunktionen mittels UMLT und zum Export der Umbaufunktionen im Format XMI.
- UMLT-Applier: FME-Transformer zur Überführung der in XMI exportierten Umbaudefinition von UMLT in eine FME-Workbench [8].
- XMI-Reader: FME-Reader zum Einlesen der Quell- und Zielmodelle im Format XMI in FME.

Die Forschungsprojekte wurden von den Landesvermessungsbehörden der Schweiz (swisstopo), von Bayern, Baden-Württemberg, Österreich und dem deutschen Bundesamt für Kartographie und Geodäsie im Zeitraum 2006 bis 2012 finanziert und von der TUM zeitweise in Kooperation mit der ETH Zürich durchgeführt.

Diese vorhandene Test-Implementierung von UMLT/UMLTtex konnte im terminlichen und finanziellen Rahmen des Projektes hier (NGDI 21-24) nicht benützt werden. Daher wurde ein minimaler Ersatz-Testrahmen erstellt. Für die grafische Formulierung des konzeptionellen UMLT-Diagramms wurde statt des UMLT-Editors ein einfaches grafisches Programm verwendet (Microsoft-Office Powerpoint). Dieses Programm wurde auch für die Formulierung der entsprechenden Wertzuweisungen als Text in der Sprache UMLTtex eingesetzt. Für den Datenumbau auf Format-Niveau entsprechend der Umbaudefinition mit UMLT auf konzeptionellem Niveau wurde an Stelle von FME ein Java-Programm entwickelt mit dem Java-Projektnamen `UMLT-Funktionen-Testbed`. Die Übersetzung vom konzeptionellen Niveau auf das Daten-Niveau erfolgte sowohl für die Datenmodelle als auch für das Transformationsmodell nicht mit UMLT-Applier und XMI-Reader, sondern von Hand. Dieser Ersatz-Testrahmen soll im Folgenden kurz erläutert werden.

Zunächst galt es die (vorkommenden!) INTERLIS-Sprachelemente nach Java zu übersetzen:

INTERLIS -Klasse	→	Java-`class`
INTERLIS-Attribute	→	Java-Attribut
elementare INTERLIS-Attributstypen	→	Java-Datentypen
INTERLIS-Struktur	→	Java-Klasse/-Typ
INTERLIS-Beziehungen (nur 1:mc)	→	Werte von Rollenattributen
Geometrietyp COORD	→	`coord` als `double`
COORD 2D oder 3D	→	(Struktur-/Typ-)Klasse mit 2 (oder 3) Koordinaten, je als `double`
Alle POLYLINEs	→	2 ArrayLists, die eine mit allen Segmenten (End Pt Koordinate, Typ B=Startsegment, D=Geradenstück, C=Kreisbogen, R=Klothoide, Parameter gemäss Typ), die andere mit einem Element für jede einzelne POLYLINE (mit Anfangs- und End-Index der zugehörigen Segmente in der Segment-ArrayList). Grund: Offenbar können ArrayList Elemente in Java nicht selber Arrays enthalten.

Die Übersetzungen der INTERLIS Datenmodelle nach Java befinden sich im Package `fktT` des Ersatz-Testrahmen Projektes `UMLT-Funktionen-Testbed`. Das Package `fktT` umfasst damit schon mal die Datenmodelle `SBB` und `DB`. Aber das Package `fktT` umfasst auch die Übersetzung des Strukturumbaus mit UMLT nach Java. Diese erfolgt mit Hilfe der Klasse von `fktT` mit dem Namen `TestRahmenFktT`. Diese enthält auch die `main`-Methode.

Dabei werden aus UMLT / ILIT / UMLTtex die Sprachelemente des Strukturumbaus wie folgt übernommen:

- **TrafoModell** heisst das oberste Sprachelement der hierarchischen Struktur. Das entsprechende ILIT-Schlüsselwort ist `MAPPING MODEL`. In unserem Beispiel von Abbildung 9 hat das TrafoModell den Namen `SBB2DB`. Im Ersatz-Testrahmen entspricht dem TrafoModell einerseits das ganze Package `fktT`, andererseits die Klasse `TestRahmenFktT` aus diesem Package `fktT`. Denn die Klassendefinition von `TestRahmenFktT` enthält vor dem `class` Statement, sozusagen auf Package-Level, den Import aller benötigten Daten-, Attributs- und Struktur-Java-Klassen, wie im ILIT-TrafoModell.
- **Thema** heisst, wie in jedem INTERLIS-Datenmodell, das dem TrafoModell direkt untergeordnete UMLT/ILIT-Sprachelement. Dessen ILIT-Schlüsselwort ist `TOPIC`. In den ILIT-TrafoModellen von [5] (S. 62) und [9] (S. 226ff) gibt es jeweils nur ein Thema, es ist auch derselbe Name möglich wie beim TrafoModell. TrafoModell und Thema zusammen bilden den äussersten Rahmen im UMLT-Diagramm Abbildung 9, mit dem Namen `SBB2DB`.
- **TrafoActivity** ist das dem Thema in der Hierarchie folgende UMLT/ILIT-Sprachelement und bildet im UMLT-Diagramm den nächstinneren Rahmen. Das entsprechende ILIT-Schlüsselwort ist `ACTIVITY`. In Abbildung 9 gibt es nur eine TrafoActivity mit dem Namen `SBB2DB_TrafoActivity`. Im Ersatz-Testrahmen entspricht der TrafoActivity auch noch die Java-Klasse `TestRahmenFktT`. Hier werden für die in der TrafoActivity definierte Transformation `SBB2DB_Trafo` die Input-Pins `wp`, `ga`, `wl`, we aufgefüllt durch den 1:1-Prozessor `sbb1to1_AsciiToJava`, realisiert als Methode der Klasse `TestRahmenFktT`.
- **Transformation** ist das nächste Sprachelement mit dem nächstinneren Rahmen im UMLT-Diagramm. Das entsprechende ILIT-Schlüsselwort ist `TRANSFORMATION`. Es gibt in Abbildung 9 eine Transformation, sie heisst `SBB2DB_Trafo`. Auch für die Transformation dient im Ersatz-Testrahmen die Klasse `TestRahmenFktT`, indem sie über die externen Input-Pins (etwa `wp`) die Startdaten zu den TrafoActions verschiebt, und in den externen Output-Pins (etwa `gs`) die Zieldaten sammelt.
- **TrafoAction** heisst das Basiselement, das effektive Transformationselement der Strukturumbau-Definition mit UMLT, grafisch dargestellt durch die innersten Kästchen des UMLT-Diagramms. Dort sind die Wertzuweisungen definiert von den Objekten der Input-Pin-Klassen auf die Objekte der Output-Pin-Klassen. Die TrafoActions sind als Methoden der Klasse `TestRahmenFktT` realisiert. Aus dieser Perspektive entspricht die Klasse `TestRahmenFktT` genau dem Rahmen der Transformation `SBB2DB_Trafo`, welcher die Kästchen der TrafoActions umfasst. Wegen der Java-Unklarheit betreffend Schachtelung von Methoden sind TrafoActions mit inneren Input-Pins samt inneren Loops nicht als eigene Methoden von `TestRahmenFktT` implementiert, sondern als einfache Fortsetzungs-Programmteile der „übergeordneten“ TrafoAction mit (äusserem) Input-Pin und entsprechendem äusseren Loop.
- **Pin** heisst ein Eingangstor (Input-Pin) bzw. ein Ausgangstor (Output-Pin) einer Transformation oder einer TrafoActivity. Externe Pins (Input- und Output-Pins einer Transformation, z.B. von `SBB2DB_Trafo`, und die entsprechenden direkt damit verknüpften Input- und Output-Pins von TrafoActions) sind als ArrayLists von Objekten der entsprechenden Klassen realisiert. Für Input-Pins ist die Iteration über die Objekte dieser ArrayLists inbegriffen, mit `for`-Loops. Interne Pins (Input- oder Output-Pins von TrafoActions ohne Verknüpfung mit einem externen Pin, wie die Pins `le_i` und `lpa_o` der TrafoActivity `LINEL2PAPLEL` [= Linienele-

ment zu Punkt allgemein, Punkt Lage und Element Lage]) sind Einzelwerte oder Arrays mit Listen von Werten. Input Pins enthalten auch hier die Iteration über die Liste von Werten (mit `for`-Loops), falls es sich um Arrays handelt. (Innere) Loops innerhalb anderer (äusserer Loops) weisen auf geschachtelte TrafoActions hin (siehe dazu die Bemerkung am Schluss des Abschnittes zur TrafoAction weiter oben).

- Dann braucht es 1:1-Prozessoren, wie oben bei der TrafoActivity erwähnt, etwa vom proprietären SBB-CSV/ASCII-Transferformat nach SBB-Java und dann auch von SBB-Java nach SBB-XTF. Diese sind als eigenständige Methoden der Klasse `TestRahmenFktT` programmiert.
- Der Ablauf des Datenumbaus mit Hilfe des Ersatz-Testrahmens ist festgelegt durch die `main`-Methode ist in der Klasse `TestRahmenFkt`. Hier werden die Methoden der Klasse `TestRahmenFktT` für die einzelnen TrafoActions in der richtigen Reihenfolge gestartet.

6.2 Der 1:1-Prozessor von SBB-CSV-Daten nach Java

Die Umformatierung der SBB-CSV-Daten nach Java-Datenstrukturen erfolgt mit einem: Java-Programm, realisiert als Methode `sbb1to1_AsciiToJava` des Hauptprogramms `TestRahmenFktT`. An sich unkompliziert: Die aus den CSV-Daten erzeugten Objekte des SBB-Datenmodells werden klassenweise in ArrayLists gespeichert, deren Elemente Struktur-Attributstypen gemäss der Klassendefinition im SBB-Modell sind. Problematisch ist nur, dass verschiedene wesentliche Ausnahmeregeln nirgends formuliert sind.

Bei SBB-CSV-Daten von Polylinien-Objekten folgen sich in der CSV-Datei die Zeilen mit den Daten der einzelnen Segmente. Diese Segmente entsprechen im Prinzip den Segmenten der entsprechenden POLYLINE, allerdings sind sie etwas anders organisiert als in INTERLIS. Je Segment kommen die Koordinaten des Anfangspunktes sowie die Parameter der Verbindungsgeometrie zum nächsten Stützpunkt. Diesen Endpunkt des Segments liefert aber erst der Anfangspunkt des nächsten Segments.

Die nachfolgend aufgeführten Problemstellungen, zu denen eine Beschreibung nicht zu finden war, wurden erkannt und behoben:

- Problem Weichenlinien: Es sind nicht alle Weichenlinien vorhanden, die erste Weichenlinie jeder Weiche fehlt und muss konstruiert werden als einziges Geradensegment von Weichenpunkt 1 zu Weichenpunkt 2. Von den übrigen Weichenlinien fehlt immer der letzte Punkt. Er ist ein Weichenpunkt, dessen Nummer sich aus der Weichenlinien-Nummer bestimmen lässt.
- Problem Gleisabschnitte: Es gibt allgemeine Daten zu Gleisabschnitten auf den Zeilen mit Zeilenidentifikation `N` , und je Gleisabschnitts-Segment gibt es eine Zeile namens `Lagegeometrie` mit Zeilenidentifikation `1`. Das Ende einer Gleisabschnitt-Linie ist dadurch definiert, dass die nächste Zeile in der SBB-CSV-Datei eine andere Zeilenidentifikation (≠ `1`) hat als die Zeilen mit Gleisabschnitt-Segmenten.
- Problem Weichenlinien-Abschluss: Das ist gleich wie bei den eben besprochenen Gleisabschnitten, aber zusätzlich noch komplizierter, da pro Weiche verschiedene Weichenlinien nacheinander kommen können, von denen jeweils die vorhergehende abgeschlossen werden muss, bevor die neue beginnt.

6.3 Der 1:1-Prozessor von SBB-Java nach XTF

Der 1:1-Prozessor von SBB-Java nach XTF ist ein einfaches Java-Programm, realisiert als Methode `sbb1to1_JavaToXTF` des Hauptprogramms `TestRahmenFktT`. Je Klasse des

SBB-Modells können die in der entsprechenden ArrayList gespeicherten Objekte sequentiell gelesen und im passenden Format in die XTF-Datei geschrieben werden.

6.4 Der Strukturumbau SBB-Java nach DB-Java

Der Strukturumbau wurde auch als Java-Programm realisiert gemäss den am Schluss von Abschnitt 6.1 erläuterten Prinzipien. Die Trafo Action `WP2PAPL` wurde als Methode `WP2PAPL()` des Hauptprogramms `TestRahmenFktT` realisiert. Die UMLTtex-Anweisungen konnten sozusagen „wörtlich“ nach Java übernommen werden.

7. Fazit des Projekts

7.1 Zielerreichung

Die Ziele des Projekts konnten erreicht werden:

Ziel 1: Liste aktueller Trafo-Funktionen
Diese Liste findet sich in Kapitel 5.1, Abbildung 12.

Ziel 2: Liste minimal nötiger zusätzlicher Trafo-Funktionen
Diese zusätzlichen Funktionen sind aufgelistet und besprochen in Kapitel 5.3 „Neue benötigte Funktionen, Version 2“. Diese Version 2 ist die Schlussversion der zusätzlich nötigen Trafo-Funktionen. Zum besseren Verständnis des Strukturumbaus mit UMLT/ILIT und der Gründe für die Entwicklung des definitiven Funktionensatzes ist in Kapitel 5.2 auch die «Startversion» des Projektes als Version 1 der Transformation und der dazu benötigten Funktionen beschrieben.

Ziel 3: Beschreibung der Methode zur Definition und Implementierung weiterer Trafo-Funktionen
Die vier zweckmässigen Schritte des Vorgehens sind in Kapitel 5.4 beschrieben.

Ziel 4: Ergebnisbericht
Der Ergebnisbericht liegt hiermit vor.

7.2 Erkenntnisse

Mit der konsequent modellbasierten Methode können auch komplex strukturierte Daten zum selben Inhalt einfach ineinander umgebaut werden, wenn die nötigen Funktionen zur Verfügung stehen und die praktisch brauchbaren Werkzeuge. Den Strukturumbau auf konzeptioneller Ebene zwischen objektorientierten Datenmodellen zu definieren und dann auf physischer Ebene automatisch ausführen zu lassen, ist die zweckmässige Methode, um eine funktionierende nationale Geodateninfrastruktur (NGDI) zu realisieren, die erlaubt, Daten über Grenzen zwischen grossen historisch gewachsenen Systemen zu nutzen, basierend auf minimalen Geodatenmodellen (MGDM).

Es brauchte zusätzliche Funktionen. Die existierenden Funktionen genügten nicht, um den Strukturumbau konzeptionell zu definieren für die hier wesentlich komplexeren Datenmodelle (Gleise von SBB und DB) als die im SumSuG-Projekt [11] bearbeiteten (Gefahrenkarte Kanton Aargau und MGDM). Es brauchte eine neue geometrische Funktion (`SplitLinToLinEl`) und verschiedene „administrative“ Funktionen (`CollectorNumToList`, `PointNrGenerator`, `NodeNrGenerator`, `Incrementer`). Ausgehend einerseits von den praktischen Bedürfnissen gegeben durch Start- und Zielmodell und andererseits davon, was dem konzeptionellen „Umstrukturierer“ an Basisprogrammierkenntnissen zuzumuten ist, ergaben sich rasch Funktionsumfang und Parameter (Kapitel 5.3). Die dabei gemachten Erfahrungen konnten zu einer ersten Checkliste von 4 Punkten zusammengefasst werden für die Definition und Implementierung neuer Trafo-Funktionen (Kapitel 5.4). Zusätzliche Funktionen sind also gut zu bewerkstelligen.

Wirklich schwerwiegend ist hingegen der Mangel an praktisch brauchbaren Werkzeugen für die konzeptionelle Definition und die physische Implementierung von Umstrukturierung. Dabei gibt es sogar zwei verschiedene gangbare Wege zu solchen (siehe 7.3, Handlungsfelder).

Eine grosse Herausforderung war, dass die an der TU München weiterentwickelte Implementierung von UMLT/UMLTtex bei diesem Projekt 21-24 auf Grund des knappen Zeitrahmens und damit auch der knappen Finanzen, für Tests nicht zur Verfügung stand. Es musste daher ein eigener Ersatz-Testrahmen programmiert werden. Mit der Programmierung dieses Testrahmens wurde versucht, den Programmcode so zu organisieren, dass sichtbar ist, was von UMLTtex (oder ILIT) direkt übernommen werden kann und was an technischem Hintergrund-Programm dazu organisiert werden muss. Damit sind erste Voraussetzungen geklärt für die automatische Herleitung des Programmtextes für den Datenumbau direkt aus dem Umbau-Modell in UMLT/UMLTtex bzw. in Zukunft besser in UMLT/ILIT.

Neben SumSuG gibt es auch ConvConf zur Vereinfachung des Strukturumbaus mit der modellbasierten Methode. Es wäre wichtig, herauszufinden, wie es sich mit dem Funktionsumfang, mit der einfachen Benutzbarkeit und allenfalls der gegenseitigen Ergänzungsmöglichkeiten der beiden Werkzeuge verhält.

Bei der Programmierung in Java wurden verschiedene Probleme augenfällig, etwa folgende:

- Die Methode `.length` einer Array-Variable liefert immer die definitorisch festgelegte maximale Grösse. Die Anzahl effektiv besetzter Elemente muss zusätzlich ermittelt und geliefert werden. Praktischer ist daher die Verwendung von ArrayList (statt Array), deren Methode `.size()` die Anzahl effektiv besetzter Elemente liefert.
- Es ist unklar, wie Funktionen zur Verarbeitung variabel langer Listen zu programmieren sind, wenn nach dem letzten Listenelement noch Zusatzarbeiten zu erledigen sind.
- Die Frage, wie die Beziehungsdefinitionen in INTERLIS zu übersetzen sind in Java-Code, damit die Rollenwerte in den Daten des Startmodells automatisch richtig interpretiert und die Rollenwerte in den Daten des Zielmodells automatisch richtig berechnet werden können, ist ein weiteres hier nicht endgültig gelöstes Problem.
- Zur Frage, wie genau die Pins von UMLT/ILIT auf dem Niveau Datenumbauprogramm zu realisieren sind, ist eine Lösung eingesetzt worden. Input-Pins enthalten nicht nur die Objekte der entsprechenden Input-Klasse sondern starten auch die Iteration über diese Objekte. Wie sich das einerseits auf die grafische Darstellung mit UMLT auswirkt, andererseits auf die Schachtelung von Java-Methoden (entsprechend den TrafoActions) konnte nicht befriedigend geklärt werden.
- In den SBB-Daten gibt es Probleme (siehe Kapitel 6.2), die nur in Zusammenarbeit mit SBB-Spezialisten behoben werden können, allenfalls müssen auch die Datenmodelle angepasst werden. Die Kontrolle der produzierten XTF-Dateien mit dem ilivalidator mit entsprechend nötigen Korrekturen der Modelle und/oder des Programmcodes und/oder der Daten konnten nicht bis zur leeren ilivalidator-Fehlerliste vorangetrieben werden.

7.3 Ausblick und Handlungsfelder

Aufgrund dieser Erkenntnisse ergeben sich Handlungsfelder für zukünftige Arbeiten, welche die Vernetzung von Geodaten weiter fördern und vereinfachen:

Mit nicht allzu grossem Aufwand können die erwähnten Probleme zwischen Java-Code und konzeptioneller Modell- bzw. Strukturumbau-Definition behoben werden (siehe Kaptiel 7.2). Die Lösung dieser Probleme kann auch einen wesentlichen Beitrag zur Klärung der Definition von UMLT/UMLTtex bzw. UMLT/ILIT liefern. Die klar formulierbaren Probleme zeigen deutlich, dass eine Klärung der Definition dieser Strukturumbausprachen nicht top-down, startend mit einer grossen Theorie stattfinden sollte, sondern wirklich eher bottom-up durch Analyse und Implementierung von Lösungen für konkrete praktische Aufgaben.

Ganz dringend ist der Vergleich von SumSuG und ConvConf mit minimalen Test-Implementierungen, damit die Bedeutung, die praktische Brauchbarkeit und allenfalls die gegenseitige Nutzbarkeit von Elementen der beiden Konzepte geklärt werden kann. Mit der vorliegenden

Ersatz-Testumgebung steht ein „Hand“-Werkzeug für die Implementierung von SumSuG zur Verfügung. Für ConvConf sollte dringend ebenfalls eine minimale Experimentier-Implementierung gemacht werden, wie sie im NGDI-Projekt 21-21 „Erstellung AutoConverter-Prototyp“ vorgeschlagen wurde.

Fehlerbehebung und offensichtliche Verbesserungen – wie sie etwa in Kapitel 5.3 besprochen sind – sollten unbedingt bei der Ersatz-Testumgebung `fktT` durchgeführt werden. Denn auch dies ist mit kleinem Aufwand möglich und bringt weitere Klärungen zur UMLT/ILIT-Definition.

Nun zum Mangel an praktisch brauchbaren Werkzeugen für die konzeptionelle Definition und die physische Implementierung von Umstrukturierung. Es gibt (mindestens) zwei Wege zu praktisch brauchbaren UMLT/ILIT-Werkzeugen:

(A) Die FME-basierte Variante TUM ist besser auf die ILI-Grundlage zu stellen und zu verbessern, beziehungsweise

(B) die FME-unabhängige Variante nach der Art der vorliegenden Ersatz-Testumgebung ist schrittweise zu entwickeln.

Es sind Schritte zu planen und der entsprechende Aufwand ist abzuschätzen für die Weiterentwicklung von SumSuG mit FME-Basierung bzw. für die FME-unabhängigen Weiterentwicklung der Ersatz-Testumgebung von fktT. Und solche Schritte gibt es.

Betrachten wir zuerst die Variante (A) der FME-basierten Lösung:

- Da gibt es Medienbrüche der Modellierungstools, die zu eliminieren sind durch verbesserte ILI-Konformität: Die gegebenen bzw. mit dem UML-Editor erzeugten INTERLIS-Modelle müssen derzeit in Enterprise Architect nochmals erstellt werden, damit sie vom UMLT Applier-Transformer eingelesen werden können. Der Transformer nutzt dazu den XMI-Reader von FME, der dahingehend angepasst werden sollte, dass er INTERLIS-Modelle direkt lesen kann. Möglicherweise kann hierfür der ili2fme- [3] oder der ili2db-Reader/Writer [4] aus FME wiederverwendet werden, der bereits eine umfassende Abbildung von INTERLIS-Modellen auf FME implementiert, so dass der Aufwand der Anpassung auf ein Minimum reduziert werden kann.
- Für die Definition und Ausführung von SumSuG basierend auf FME müssen verschiedene Werkzeuge mit unterschiedlichen Benutzerschnittstellen bedient werden. Für benutzerfreundliche Anwendung braucht es ein GIU für alle Arbeiten.
- Für UMLT wurde neben der jetzt bei Variante (A) verwendeten textuellen Notation UMLTtex und XMI als Austauschformat ursprünglich auch eine INTERLIS-basierte Notation entwickelt, HUTN (Human Usable Textual Notation) bzw. ILIT (INTERLIS TrafoDefinition) genannt [10], die jedoch in den erwähnten Forschungsprojekten aufgrund der verstärkten Ausrichtung auf die ISO-Normenreihe 191xx für Geoinformation nicht weiter zum Einsatz kam. Bei einer Verbesserung der ILI-Konformität wäre zu überlegen, den für (A) jetzt existierenden Eclipse-basierten UMLT-Editor dahingehend anzupassen, dass er die Transformationsdefinitionen im Format ILIT exportiert, und entsprechend den UMLT Applier-Transformer, dass er die Trafo Definitionen im Format ILIT liest.

Nun zur Variante (B), FME-unabhängige Lösung:

- Als konsequentere Lösung zeigt sich jedoch (für (A) und (B) brauchbar), den aktuellen Eclipse-basierten UMLT-Editor für die Strukturumbau-Definition zu ersetzen durch Erweiterung der bestehenden ILI-Software: Den grafischen UML/ILI-Editor erweitern zum UMLT-Editor, der die textuelle Beschreibung des entwickelten Trafomodells in ILIT liefert, so wie er jetzt zum UML-Datenmodell die textuelle Beschreibung in INTERLIS liefert. Den INTERLIS-Compiler erweitern, so dass er auch ILIT-Trafomodelle überprüft.
- Die Beseitigung der erwähnten Java-Unklarheiten und Überlegungen zur sauberen Konzeption modularer Software-Komponenten für die Ersatz-Test-Umgebung `fktT` können den Ausgangspunkt bilden für die Programmierung eines FME-unabhängigen UMLT/ILIT Ap-

plier-Transformers. Der erledigt selbständig gemäss dem TrafoModell in ILIT den Umbau der Startdaten in XTF gemäss Start-ILI-Modell in die Zieldaten in XTF gemäss Ziel-ILI-Modell. Dabei ist UMLT/ILIT nicht theoretisch zu konzipieren, sondern praktisch schrittweise mit typischen Beispielen aufzubauen.

8. Literaturverzeichnis

Autor (en)	Titel	Version / Datum
[1] Donaubauer, A., Kutzner, T., Gnägi, H. R., Henrich, S. und Fichtinger, A. (2010).	Webbasierte Modelltransformation in der Geoinformatik.	In G. Engels, D. Karagiannis, und H. C. Mayr (Hrsg.), *Modellierung 2010* (Bd. 161 Lecture Notes in Informatics-Proceedings, S. 269–284). Klagenfurt: Gesellschaft für Informatik.
[2] eCH-Fachgruppe Geonormen. (2017).	eCH-0031 INTERLIS 2 - Referenzhandbuch.	eCH-Standard No. eCH-0031. Verfügbar am 2022-03-31 unter https://www.ech.ch > eCH-Standards > Übersicht nach eCH-Nr
[3] Eisenhut, C. (2014, März 2013)	ili2fme - INTERLIS-plugin for FME.	Verfügbar am 2022-05-24 unter https://ili2fme.ch/
[4] Eisenhut, C. (2017)	ili2db: interlis import/export to relational databases.	Verfügbar am 2022-06-03 unter https://www.interlis.ch/en/downloads/ili2db
[5] Gnägi, H. R., Henrich, S., Münster, M., Rüegg, R., Eisenhut, C. (2009)	MDAinSVT: Einsatz modellbasierter Datentransfernormen (INTERLIS) in der Strassenverkehrstelematik am Beispiel der Verkehrsdaten.	Bundesamt für Strassen ASTRA, Bericht 1267 zum Forschungsauftrag VSS 2007/902
[6] Kutzner, T. (2016)	Geospatial Data Modelling and Model-driven Transformation of Geospatial Data based on UML Profiles	PhD, TU München, München, https://mediatum.ub.tum.de/node?id=1280909
[7] Kutzner, T., Donaubauer, A., Müller, M., Feichtner, A. und Goller, S. (2014)	Erfolgreiche Transformation von Geodaten nach INSPIRE	Zeitschrift für Geodäsie, Geoinformation und Landmanagement, 139. Jg.(2/2014), 103–109. https://geodaesie.info/zfv/heftbeitrag/3170
[8] Safe Software Inc. (2016)	FME Desktop Advanced Training Manual 2016.1	Verfügbar unter https://cdn.safe.com/training/course-materials/FME-Desktop-Advanced-Training-Manual.pdf
[9] Staub, P. (2009)	Über das Potential und die Grenzen der semantischen Interoperabilität von Geodaten.	DISS. ETH Nr. 18201, IGP Mitteilungen Nr. 102
[10] Staub, P., Gnägi, H.R. und Morf, A. (2008)	Semantic Interoperability through the Definition of Conceptual Model Transformations	Transactions in GIS, 12(2), 193–207
[11] Strösslin, T., Gnägi, H.R., Kutzner, T., Loidold, M., und Stahl, M. (2017)	SumSuG: Systemunabhängiger, modellbasierter Strukturumbau von Geodaten (Projektbericht)	Schweizerische Organisation für Geo-Information (SOGI), https://goo.gl/T3PV1s > SumSuG_Projektbericht.pdf
[12] Winter, W., Leckenbusch, L., Colle, J. (2002)	Spezifikation Bearbeiten und Prüfen von Gleisnetzdaten	DB Netz AG, Version 08.10.2002

9. Abkürzungen und Glossar

Abkürzungen

Abkürzung	Bedeutung
→ *A*	Begriff A ist im Glossar erklärt
2D	zweidimensional
3D	dreidimensional
Abb.	Abbildung
abg	abgehend
Abk.	Abkürzung
ank	ankommend
ASCII	American Standard Code for Information Interchange, 7 Bit-Zeichencodierung, US-Variante ISO 646
AV	(mehrdeutig) Appareil de voie (=→ *Weiche*), auch: Amtliche Vermessung
Bem.	Bemerkung
bzw.	beziehungsweise
CSV	Comma Separated Value, Bezeichnung für Feld-orientierte Datenformate
DB	(mehrdeutig) Deutsche Bahn, auch: Datenbank
de	Deutsch
d.h.	das heisst
Def.	Definition
El	→ *Element*
en	Englisch
etc.	et cetera, und so weiter
ev.	eventuell, möglicherweise
FME	Feature Manipulation Engine von Safe Software
fr	Französisch
g2l	Gnägis second law (→ *semantisches Modell)*
GML	→ *Geography Markup Language*
GIS	Geografisches Informationssystem
GND	Gleisnetzdaten (der DB)
HUTN	→ *Human Usable Textual Notation*
ID	Identifikation
IDDL	→ *INTERLIS* Data Definition Language
IG	InfoGrips AG, Software Entwicklungs- und Beratungs-Firma
ILI	→ *INTERLIS*
ILIT	→ *INTERLIS TrafoDefinition*
ITF	→ *INTERLIS* 1 Transfer Format, objektrelationaler → *CSV* → *ASCII* → *Transferformat*
Kap	Kapitel
KGK	Konferenz der kantonalen Geoinformations- und Katasterstellen
KN, kn	→ *Knoten*
KOGIS	Koordination, Geoinformation und Services (swisstopo)
Koord.	→ *Koordinate(n)*
MDA	Model Driven Approach (→ *Modellbasiertes Vorgehen*)
MGDM	Minimale Geo Daten Modelle
Nr	Nummer
OGC	Open Geospatial Consortium
OID, oid	→ *Objektidentifikation*

Abkürzung	Bedeutung
PAD	Punktadresse, Punktnummer in DB-Daten
Pos	Position
PROK	Projektkoordinator des Auftraggebers (swisstopo)
propr	proprietär
Pt	→ *Punkt*
RVT	Reference Value Transformation, → *Referenzwert-Transformation*
SBB	Schweizerische Bundesbahnen
sog.	sogenannt/e/er/es
SumSuG	Systemunabhängiger modellbasierter Strukturumbau von Geodaten
Syn	Synonym
Sys	→ *System*
Trafo	→ *Transformation*
TU	Technische Univerrsität
UML	→ *Unified Modeling Language*
UMLT	→ *UML TrafoDefinition*
UMLTtex	→ *UML TrafoDefinition mit Text*
usw.	und so weiter
XML	eXtensible Markup Language
XML-Schema	→ *XSD*
XSD	→ *XML* Schema Description Language
XTF	→ *XML* basierter → *INTERLIS 2* → *Transferformat*
z.B.	zum Beispiel
Zid_x	Zeile mit x als Zeilen-Identifikation (d.h. als erstes Zeichen einer Zeile einer CSV-Datei im ASCII-Format)

Glossar

Begriff	Bedeutung
1:1-Prozessor	Programm zur Änderung des →*Transferformats* ohne → *Strukturumbau*
Abbildung	(aus einer Menge A in eine Menge Z:) Vorschrift, die einem Element $a \in A$ genau ein Element $z \in Z$ zuordnet, z wird dann als $f(a)$ bezeichnet. A heisst Definitionsbereich, Z heisst Wertebereich der Abbildung. Die Menge aller $z \in Z$, die einem $a \in A$ zugeordnet sind, heisst Bild(menge) der Abbildung, kurz $f(A)$. Die Menge aller $a \in A$, denen ein $z \in Z$ zugeordnet ist, heisst Urbild(menge), kurz $f^{-1}(f(A))$, wobei f^{-1} im Allgemeinen nicht das Symbol einer Abbildung ist. Die Menge der Paare (a, z) mit $z = f(a)$ heisst Graph der Abbildung. Syn. Funktion (de), map, mapping (en), application (fr) Bem. 1: Bei Zahlenmengen A und Z kann der Graph der Abbildung in einem → *Koordinatensystem* dargestellt werden. Bem. 2: Es gibt → *surjektive*, → *injektive*, → *bijektive* → *stetige*, → *glatte* A. Präzise Definition siehe Mathematik, anschauliche Erläuterung siehe bei den einzelnen Begriffen. Bem. 3: Eine besonderen A.ist die → *Koordinatentransformation*, wobei die Menge A und die Menge Z je ein → *Referenzbereich* ist, definiert durch ein → *Koordinatensystem*. Bem. 4: Eine andere spezielle A. ist die → *Operation*, nämlich eine A., bei welcher die Menge A aus → *Wertebereichen* von → *Attributen* bzw. von Eingabe→ *Parametern* besteht, Z aus → *Wertebereichen* von Ausgabe → *Parametern.*
Abbildung auf	Syn. für → *surjektiv.*
Abstraktion	Zweite Phase des → *modellbasierten Vorgehens*, in der festgelegt wird, mit welcher mathematischen Theorie der → *Realitätsausschnitt* bearbeitet wird. Syn. (zu vermeiden, da ungenau weil mehrdeutig): Modell Bem.: Beinhaltet der → *Realitätsausschnitt* Gegenstände der Realwelt und insbesondere deren räumliche Ausdehnung und Positionierung, dann ist dessen A. sinnvollerweise die Geometrie des → *Raumes*, d.h. des 3-dimensionalen Euklidischen Raumes.

Begriff	Bedeutung
Appareil de voie	Syn (fr CH SBB) für → *Weiche* Abk. (mehrdeutig) AV
Argument	→ *Wert* eines → *Parameters.*
Assoziation	Eigentliche → *Beziehung.* Bemerkung: oft auch Syn. für → *Beziehung* allgemein.
Attribut	Daten(elemente) entsprechend einer spezifischen Eigenschaft von → *Objekten* einer → *Klasse*. Ein A. hat einen A.-Namen und einen → *Wertebereich.* Syn. Merkmal (de); attribute (en).
Bezeichner	Text zur Beschreibung eines Datenfeldes, meist in < … > vor und nach dem Datenfeld. Syn.: tag (en)
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. Assoziation (dt), relationship (en).
Beziehungsobjekt	Def → *Beziehung*
Bijektiv	Eigenschaft einer → *Abbildung*: → *injektiv* und → *surjektiv*..
Bild(menge)	Def → *Abbildung*
Bucket	Menge von → *Objekten* , die bei einer → *HashMap* einem → *Schlüssel* zugeordnet ist.
Datei	Def. siehe Informatik. Syn. file (en), fichier (fr).
Daten	Def. siehe Informatik. Syn. data (en), données (fr).
Datenbeschrei-bungssprache	→ *Formale Sprache* zur exakten Beschreibung von → *Daten.* Syn.: Data description language (DDL), conceptual schema language (CSL) Abk.: DDL, CSL.
Datenelement	Kleinster Bestandteil von → *Daten*, der mit Hilfe eines → *Systems* bearbeitet werden kann. Bemerkung: Details dazu siehe Informatik.
Datenmodell	Vollständiges und in sich geschlossenes → *konzeptionelles* → *Datenschema* . Bemerkung 1: Ein D. ist also eine exakte Beschreibung von → *Daten* bzw. genauer von deren → *Daten-struktur* , die vollständig und in sich geschlossen ist. Bemerkung 2: Das D. ist das hierarchisch höchste → *Modellierungselement.* Syn. Modell, Datenbeschreibung.
Datenobjekt	Syn. von → *Objekt*
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. Bem. 1: Zur Formulierung eines D. gibt es geeignete → *Datenbeschreibungssprachen*. Bem. 2: Es gibt konzeptionelle, logische und physische D.s. Bem. 3: Ein besonderes D. ist das → *Datenmodell.*
Datenschnittstelle	Zugriff auf → *Daten* eines → *Systems* definiert durch das → *Datenmodell* der verfügbaren → *Daten*, durch das → *Transferformat* derselben und durch den Transferprozess. Syn. (mehrdeutig) → *Schnittstelle* Bem. 1: D. heisst etwa auch ein Programm zum Umformatieren von → *Transferdateien.* Bem. 2: Im objektorientierten Systemaufbau ist die D. gegeben durch die → *Schnittstellensignaturen* der → *Operationen*, mit welchen Daten eingelesen / ausgegeben werden können.
Datenstruktur	Gliederung von → *Daten* in → *Datenelemente* . D. wird präzis beschrieben durch ein → *Datenmodell* .
Datentransfer	Verschiebung von → *Daten* von einem → *System* $\mathcal{S}$ zu einer anderen → *System* $\mathcal{Z}$. $\mathcal{S}$ wird bezeichnet als Startsystem, Ausgangssystem, Quelle, Sender, Sendersystem, Source, $\mathcal{Z}$ als Zielsystem, Empfänger, Target. Die Lieferung der zu transferierenden → *Daten* durch → *System* $\mathcal{S}$ wird auch als Export bezeichnet, die Übernahme durch → *System* $\mathcal{Z}$ als Import. Syn. Transfer, Datenübertragung.

Begriff	Bedeutung
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* .
Datenübertragung	Syn. für → *Datentransfer* .
Dissolver	Bisherige Funktion für den Strukturumbau (siehe Kap. 5.1**Error! Reference source not found.**)
Ebene	2-dimensionaler Unterraum des →*Raumes*. Bemerkung: Details siehe lineare Algebra.
Ecke	Nicht glatter → *Punkt* eines → *Linienzugs*. Syn. Stützpunkt, Lagepunkt
Eindeutiges Attribut	→ *Attribut* einer → *Klasse*, das bei jedem → *Objekt* der → *Klasse* genau einen → *Wert* hat und alle diese → *Werte* sind verschieden voneinander. Syn.: Schlüssel
Element	Grundbegriff der Mengenlehre. Eine Menge besteht aus E. Syn. Instanz Abk.: El
ElementLage	Syn. für → *Segment* (Deutsche Bahn DB)
Extensible Markup Language	→ *Transferformat* mit → *Bezeichnern* Syn.: XML-Transferformat Abk.: XML
Extensible Markup Language Schema Description	→ *Formatbeschreibungssprache* für den → *XML-Transferformat* . Syn.: XML-Schema Abk.: XSD
Formale Sprache	Beispiele: Programmiersprache, Modellierungssprache
Format	Syn. für → *Transferformat*
Formatbeschreibungssprache	Formale Sprache zur exakten Beschreibung von → *Transferformaten*.
Funktion	Syn. für → *Abbildung,* speziell für → *Operation.*
Geography Markup Language	→ *Transferformat* für Geodaten von OGC und ISO/TC211 entwickelt, → *XML*-Dialekt. Abk.: GML
Gerade	1-dimensionaler Unterraum des →*Raumes* oder einer →*Ebene.* Bemerkung: Details siehe lineare Algebra.
Glatt	Eigenschaft einer → *Abbildung*, Def siehe Mathematik. Bem.: Anschaulich, der → *Graph der Abbildung* hat keine Ecken, Spitzen
GleiskanteStrecke	→ *Kante* des → *Graphs* für die → *Topologie* der DB-Geleise Abk.: GS, gs
Gleisknoten	→ *Knoten* des → *Graphs* für die → *Topologie* der DB-Geleise Abk.: KN, kn
Gleisnetz	→ *Netz* von → *Gleiskanten* und → *Gleisknoten*.
Graph	Besteht aus → *Knoten* und (orientierten bzw. nicht orientierten) → *Kanten*, wobei zu jeder → *Kante* ein (geordnetes bzw. ungeordnetes) Paar von → *Knoten* gehört, nämlich ihre beiden Endknoten
Graph einer Abbildung	Def → *Abbildung.*
HashMap	Datenorganisation, die → *Schlüssel* zu → *Elementen* zuordnet. Als → *Elemente* kommen in Frage einzelne → *Objekte* oder → *Buckets.*
Human Usable Textual Notation	Für Menschen lesbare und verständliche → *formale Sprache* als → *Datenbeschreibungssprache* oder → *Formatbeschreibungssprache* oder → *Strukturumbaubeschreibungssprache*) Abk.: HUTN
Injektiv	Eigenschaft einer → *Abbildung* aus A in Z: Für jedes $z \in Z$ gilt, entweder gibt es genau ein $a \in A$, so dass $f^{-1}(\{z\}) = \{a\}$ oder dann ist $f^{-1}(\{z\}) = \emptyset$. Syn. eineindeutig, umkehrbar eindeutig, Injektion, Einbettung Bem.: Anschaulich, zu jedem Element der Bildmenge $z \in f(A)$ gehört nur genau ein Element $a \in A$, dessen Bild $f(a)$ das Element z ist.

Begriff	Bedeutung
Instanz	Syn. für → *Element* (konkretes Exemplar) einer Menge (Abstraktion). Syn. instance (en). 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 → *Assoziation*.
INTERLIS.	→ *Datentransfer-Mechanismus* (auch für Geodaten) bestehend aus der → *INTERLIS-Datenbeschreibungssprache* (IDDL) und dem INTERLIS-XML-Transferformat (XTF) sowie Regeln für die Herleitung des XTF für eine mit IDDL beschriebene Datenstruktur. IDDL, XTF und Umsetzungsregeln sind definiert in der eCH-Norm eCH-0031 [2]. Abk.: ILI, ili
INTERLIS-Datenbeschreibungssprache	(Konzeptionelle) → *Datenbeschreibungssprache* des → *Datentransfer-Mechanismus* → *INTERLIS*.
INTERLIS Trafo-Definition	Konzeptionelle textuelle → *Transformationsbeschreibungssprache*. Abk.: ILIT Bemerkung: Erweiterung der → *INTERLIS Datenbeschreibungssprache* für die konzeptionell präzise Beschreibung der → *semantischen Transformation* mit Text.
Kante	Def siehe → *Graph*. Syn.: arête (fr), edge (en) Bemerkung 1: Anschaulich ist die K. das lineare Element eines → *Graph*. Bemerkung 2: Eine K. kann (aber muss nicht) ein → *Linienzug* im → *Raum* sein. Bei der → *Topologie* der DB-Geleise ist das der Fall. Eine solche K. zwischen zwei aufeinanderfolgenden → *Knoten* heisst→ *GleiskanteStrecke*.
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).
Knoten	Def. → *Graph*. Abk.: KN, kn Bemerkung: Anschaulich ist der K. das punktförmige Element der → *Topologie* → *Graph*.
Konzeptuell	Syn. für → *konzeptionell*.
Koordinate	→ *Referenzwert* durch ein → *Koordinatensystem* definiert. Syn.: Direct position (en, ISO 19148)
Koordinatensystem	→ *Referenzsystem*, das Basis ist des → *Raumes* (falls der ganze → *Raum* → *Referenzbereich* ist). Syn. coordinate reference system (en), système de coordonnées (fr).
Koordinaten-Transformation	→ *Referenzwert-Transformation* zwischen → *Koordinatensystemen*. D.h.: → *Abbildung* von einem → *Koordinatensystem* (bzw. von seinem → *Referenzbereich*) auf ein anderes → *Koordinatensystem* (bzw. auf dessen → *Referenzbereich*), 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 de coordonnées (fr).
Lagepunkt	Syn. für → *Ecke*
Linie	Syn. für → *Polylinie* (in UMLTtex)
Linienelement	Syn. für → *Segment* (in UMLTtex)
Linienzug	Teilmenge des → *Raumes*, die Bildmenge einer stetigen und stückweise glatten (aber nicht notwendigerweise injektiven) → *Abbildung* eines Intervalls ist (der sogenannten zugeordneten → *Abbildung*) und nur endlich viele nicht glatte → *Punkte* (sogenannte → *Ecken*) aufweist. Bemerkung: Begriffe siehe Mathematik (allgemeine Topologie). Beispiel für L.: → *Polylinie*.
Metaobjekt	→ *Objekt*, das in einem → *Modell* gebraucht wird
Modell	Syn. für → *Datenmodell* bzw. für → *Trafomodell*. 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).
Modellbasiertes Vorgehen	Vorgehensweise, um von einem → *Realitätsausschnitt* mit → *Abstraktion* über ein → *Datenmodell* zu → *Daten* und Programmen für deren Bearbeitung zu gelangen. Syn. Model Driven Approach, Model Driven Architcture Abk. MDA

Begriff	Bedeutung
Modellierungselement	Beschreibungsmöglichkeit einer → *formalen Sprache*, die als Schlüsselwort zur Verfügung steht.
Netz	Syn von → *zusammenhängender Graph*
NumToListCollector	Neue Strukturumbau-Funktion (siehe Kap. 5.3)
Objekt	Daten eines → *Realweltobjektes* mit den → *Operationen*, die mit diesen Daten ausgeführt werden können, und mit einer → *Objektidentifikation*. Syn. Datenobjekt
Objektidentifikation	→ *Eindeutiges Attribut* , das dauerhaft ist (d.h. keine Änderung im Laufe der Zeit und keine Wiederverwendung, falls das Objekt gelöscht wird) und generell (d.h. eindeutig nicht nur innerhalb der → *Klasse* sondern innerhalb einer Transfergemeinschaft).
Operat	Gesamtheit aller bearbeiteten Daten
Operation	→ *Abbildung* aus den Attributswertebereichen einer → *Klasse* und/oder aus → *Wertebereichen* von Eingabe → *Parametern* in den → *Wertebereich* eines Ausgabe → *Parameters.* Syn.: Funktion
Pin	Sprachelement von → *UML-TrafoDefinition*.
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
Polylinie	→ *Linienzug,* dessen → *Segmente* Geradenstücke oder Kreisbogen oder Klothoiden sind.
Programm	Syn. für Computerprogramm, Def. siehe Informatik
Prozessor	Syn. für → *Programm (Software)*.
Punkt	(Mengen-)Element des → *Raumes* (als Menge betrachtet)
Raum	3-dimensionaler Euklidischer Raum. Bemerkung: Def. des 3D Euklidischen Raumes siehe Mathematik.
Realitätsausschnitt	Teil der Realwelt, für die Bearbeitung eines Themas oder Themenbereiches wesentlich. Bem. 1: Ein Gegenstand der Realwelt kann zu verschiedenen Realitätsausschnitten gehören und entsprechend können sehr verschiedene Eigenschaften wesentlich sein. Beispiel: Strasse für Strassenbau und für Adressenverwaltung. Bem. 2: Für die Beschreibung des R. in Umgangssprache wird oft auch der Begriff → *semantisches Modell* gebraucht.
Realweltobjekt	Gegenstand der realen Welt Bemerkung: Ein R. kann digital beschrieben werden durch ein → *(Daten)Objekt*
Referenzbereich	Teilmenge des → *Raumes,* in der durch ein → *Referenzsystem* jedem → *Punkt* eindeutig ein → *Referenzwert* zugeordnet werden kann.
Referenzrahmen	Materialisierung eines → *Referenzsystems* in der Realwelt Beispiel: Die Materialisierung erfolgt durch Auswahl und besondere Kennzeichnung von punktförmigen → *Realweltobjekten* (z.B. von Steinen, Bolzen etc.). deren → *Referenzwerte* im → *Referenzsystem* bekannt sind. Syn.: Bezugsrahmen (de CH L+T swisstopo)
Referenzsystem	Werkzeug, um die Lage von → *Objekten* eindeutig zu beschreiben (durch → *Referenzwerte*) Syn. (mehrdeutig) → *Koordinatensystem.* Bemerkung: Ein besonderes R. ist das → *Koordinatensystem.*
Referenzwert	→ *Wert*, welcher der Lage eines → *Objekts* im → *Referenzbereich* eines → *Referenzsystems* eindeutig zugeordnet ist, aus dem → *Wertebereich*, der für das → *Referenzsystem* definiert ist. Syn: Bezug (de CH, VSS Normen), Lokalisation (de), localisation (en), position (en), repérage (fr CH)
Referenzwert-Transformation	→ *Abbildung* von einem → *Referenzsystem* (bzw. von seinem → *Referenzbereich*) auf ein anderes → *Referenzsystem* (bzw. auf dessen → *Referenzbereich*). Syn.: → *Transformation* (mehrdeutig), Reference Value Transformation (en) Abk.: RVT Bem.: Eine spezielle R. ist die → *Koordinatentransformation.*
Repérage	Syn. (fr CH) für → *Referenzwert*
Schlüssel	Syn. für → *Eindeutiges Attribut* Beispiel: Bei → *HashMaps* wird jedem Element ein S. zugeordnet, mit Hilfe dessen das Element gesucht werden kann.

Begriff	Bedeutung
Schnittstelle	Syn. (mehrdeutig) für → *Datenschnittstelle,* Klassenschnittstelle, Benutzerschnittstelle. Bem.: Klassenschnittstelle und Benutzerschnittstelle kommen in diesem Bericht nicht vor. Syn. (en, fr): interface
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.
Segment	Glatter Abschnitt eines → *Linienzugs* zwischen zwei benachbarten → *Ecken.* Syn. Linienelement (SBB), ElementLage (Deutsche Bahn DB)
Semantik	Syn. für Bedeutung
Semantik erhaltende Transformation	Syn. für → *semantische Transformation*
Semantisches Modell	Beschreibung des → *Realitätsausschnitts* in Umgangssprache. Bemerkung: Der Term ist verwirrend, da er suggeriert, dass Semantik nur in Umgangssprache beschrieben werden kann, entgegen unserer Erfahrung, wonach durch das präzise → *konzeptionelle Schema* mindestens 95% der Semantik erfasst werden kann (= g2l).
Semantische Transformation	Definition des Umbaus eines → *Datenmodells* in ein inhaltlich (semantisch) äquivalentes → *Datenmodell* und daraus automatische Herleitung des entsprechenden Datenumbaus.. Syn: Semantik erhaltende Transformation, Strukturumbau
Signatur	Syn. von → *Schnittstellensignatur*
SplitLinToLinEl	Neue Strukturumbau-Funktion (siehe Kap. 5.3)
Stetig	Eigenschaft einer → *Abbildung,* Def siehe Mathematik. Bem.: Anschaulich, der → *Graph der Abbildung* hat keine Lücken, Risse
Strecke	Mehrdeutig: Mathematik: Abgeschlossene und zusammenhängende Teilmenge einer → *Geraden*. Bahn: Verlauf einer Bahnlinie zwischen zwei → *Knoten*.
StringConcatenator	Bisherige Strukturumbau-Funktion (siehe Kap. 5.1)
Structure	Sprachelement (STRUCTURE) der → *INTERLIS-Datenbeschreibungssprache*, geeignet zur Definition komplexer → *Wertebereiche.*
Struktur	Syn. (mehrdeutig) für → *Datenstruktur* und → *Structure*
Strukturumbau	Syn. für → *semantische Transformation*
Stützpunkt	Syn. für → *Ecke*
Surjektiv	Eigenschaft einer → *Abbildung*: $f(A) = Z$. Syn. → *Abbildung* **auf** (Z).
System	Gesamtheit aller zu einer EDV-Anlage gehörenden Komponenten (Hardware und Software), die für einen bestimmten Zweck genutzt werden.
tag	Syn. en für → *Bezeichner.*
Topologie	Im Bereich der → *Geleise* braucht man die sogenannte kombinatorische T, um z.B. → *Gleisnetze* zu bearbeiten. Die verwendeten Grundbegriffe sind → *Graph* und → *Netz.*
Topologische Struktur	Syn. von → *Graph* (im Falle der DB-Gleise)
TrafoAction	(innerstes) Sprachelement der → *Transformationsbeschreibungssprachen* → *INTERLIS-TrafoDefinition* (ILIT), → *UML-TrafoDefinition* (UMLT) und → *UML-TrafoDefinition mit Text* (UMLTtex) Bemerkung: Enthält Zuweisungen (Assignments), ev. mit Funktionsaufrufen.
TrafoModell	(äusserstes) Sprachelement der → *Transformationsbeschreibungssprachen* → *INTERLIS-TrafoDefinition* (ILIT), → *UML-TrafoDefinition* (UMLT) und → *UML-TrafoDefinition mit Text* (UMLTtex)
Transfer	Syn. für → *Datentransfer.*
Transferdatei	Zum → *Datentransfer* vorbereitete → *Datei* in geeignetem → *Transferformat.*
Transferformat	Gliederung einer → *Transferdatei* in Datenfelder. Syn. Format.
Transformation	Mehrdeutiges Syn. für → *Referenzwert-Transformation,* → *Koordinaten-Transformation* und → *semantische Transformation.* Bemerkung: In diesem Bericht meist als Abkürzung verwendet für → *semantische Transformation*.

Begriff	Bedeutung
Transformations-beschreibungs-sprache	→ *Formale Sprache* zur exakten Beschreibung von → *semantischer Transformation*. Bemerkung: Siehe → *INTERLIS TrafoDefinition*, → *UML TrafoDefinition* und → *UML TrafoDefinition mit Text*.
Umbaufunktion	→ *Operation* zur Vereinfachung der Definition der → *semantischen Transformation.*
Umkehrbar eindeutig	Syn von → *Injektiv*
UML TrafoDefinition	Konzeptionelle grafische → *Transformationsbeschreibungssprache*. Abk.: UMLT Bemerkung: Auf → *UML* basieren, Element des → *Datentransfermechanismus* → *INTERLIS*.
UML TrafoDefinition mit Text	Konzeptionelle textuelle → *Transformationsbeschreibungssprache*. Abk.: UMLTtex Bemerkung: Variante statt → *INTERLIS TrafoDefinition ILIT*
Unified Modelking Language	Grafische → *Datenbeschreibungssprache*. Abk.: UML Bemerkung: Bei diesem Projekt wird vor allem das Klassendiagramm von UML und die auf UML basierende → *UML-TrafoDefinition* verwendet.
ValueMapper	Bisherige Strukturumbaufunktion (siehe Kapitel 5.1).
Weg	Endliche Folge von paarweise verschiedenen → *Knoten* v_i und (daher auch paarweise verschiedenen) → *Kanten* k_i der Art $(v_1, k_1, v_2, k_2, \dots, v_{n-1}, k_{n-1}, v_n)$, wobei (v_i, v_{i+1}) das zur → *Kante* k_i gehörige Paar von → *Knoten* ist. Syn: path (en).
Weiche	(Bei Bahnen) Element des Gleisverlaufs, meist mit Abzweigung. Syn. Appareil de voie (fr CH SBB) Abk. (mehrdeutig) AV
Weichenlinie	→ *Polylinie* einer → *Weiche* , d.h. → *Abstraktion* eines Stücks Gleisverlauf in einer → *Weiche* .
Wert	→ *Datenelement* eines → *Wertebereichs.*
Wertebereich	Menge gleichartiger → *Datenelemente*. Ein → *Datenelement* eines W. heisst → *Wert.* Syn: Datentyp.
Wertebereich einer Abbildung	Def → *Abbildung*
XML-Schema	Syn von → *Extensible Markup Language Schema Description*
XML-Transferformat	Syn von → *Extensible Markup Language*
Zusammenhängender Graph	Zu je zwei verschiedenen → *Knoten* des → *Graphs* gibt es einen → *Weg*, dessen Endknoten sie sind, d.h. je zwei verschiedene → *Knoten* des→ *Graphs* sind verbunden durch einen → *Weg.* Syn: Netz (de), réseau (fr), connected graph, net (en)

Anhang 1 Verwendete Hilfsmittel

Folgende Software-Tools kommen in diesem Projekt zum Einsatz:

Software	Version	Verwendungszweck	Lizenzbedingungen
INTERLIS-Compiler	5.2.2	Erstellen XTF- und GML-Schemata aus ili-Modell	LGPL (Lesser GNU Public License)
UML/INTERLIS Editor	3.6.1	Entwurf der Datenmodelle SBB und DB	LGPL (Lesser GNU Public License)
ILI-Validator	1.11.13	Prüfen einer INTERLIS-XTF-Datei auf Konformität mit dem entsprechenden INTERLIS-Datenmodell	LGPL (Lesser GNU Public License).
INTERLIS Checker	Version 2020.0; Update vom 23.03.2021	Prüfen einer INTERLIS-XTF-Datei auf Konformität mit dem entsprechenden INTERLIS-Datenmodell	WORLD WIDE SINGLE USER LICENSE
Java JDK	jre1.8.0_241	Wird zur Ausführung von Eclipse und zur Entwicklung des UMLT-Editors und der FME-Plugins UMLTApplier und XMI-Reader benötigt.	Oracle Binary Code License Agreement.0
Eclipse Modeling Tools	4.5.2 (Mars.2)	Entwicklungsumgebung zur Implementierung des UMLT-Editor und der FME-Plugins UMLTApplier und XMI-Reader Beinhaltet die für die Entwicklung benötigten Plugins - Eclipse EMF, Version 2.7.0 - Eclipse GMF, Version 1.5.0 - Eclipse UML2, Version 3.2.1	EPL (Eclipse Public License)
Microsoft Office Powerpoint	14.0.7268.5000 (32 Bit)	Zum Entwurf des UMLT-Diagramms, als Vorstufe zum UMLT-Editor gedacht, hier als UMLT-Ersatz-Editor verwendet	Microsoft

Anhang 2 INTERLIS-Modell Gleisdaten Struktur SBB

```
INTERLIS 2.3;

CONTRACTED MODEL SBB (de) AT "http://www.gis.ethz.ch" VERSION "2022-05-10" =
  IMPORTS INTERLIS;

  UNIT
    NeuGrad [gon] = 200.0 / PI [INTERLIS.rad];

  DOMAIN
    Azimuttyp = 0.00000 .. 399.99999 CIRCULAR [gon];
    Koord2 = COORD 480000.00000 .. 850000.00000 [INTERLIS.m],
                   60000.00000 .. 320000.00000 [INTERLIS.m],
                   ROTATION 2 -> 1;
    Laengentyp = -99999.00000 .. 999999.00000 [INTERLIS.m];
    LinienGeomTyp = (Gerade,Kreis,Klothoide);
    Radiustyp = -99999.9999 .. 999999.0000 [INTERLIS.m];

  STRUCTURE CLOTHO (FINAL) EXTENDS INTERLIS.LineSegment =
    Ra : MANDATORY Radiustyp;
    Re : MANDATORY Radiustyp;
    A : MANDATORY Azimuttyp;
  END CLOTHO;

  LINE FORM
    CLOTHOIDES : CLOTHO;

  TOPIC SBB_Gleisnetz =

    STRUCTURE GrDt =                    !!Gradient
      Gipfelhoehe : SBB.Laengentyp;
      Azimut_planim : SBB.Azimuttyp;
      HalbLaenge : SBB.Laengentyp;
      Abhang_vor : TEXT*20;
      Abhang_nach : TEXT*20;
      Radius : SBB.Laengentyp;
    END GrDt;

    STRUCTURE LprDt =                   !!Längsprofil
      Typ : TEXT*20;
      Hoehe_Anf : SBB.Laengentyp;
      Laenge_horiz : SBB.Laengentyp;
      Abhang_Anf : TEXT*20;
      Radius : SBB.Laengentyp;
    END LprDt;

    STRUCTURE UeDt =                   !!Überhöhung
      Laenge : SBB.Laengentyp;
      Ueberhoehung_Anf : SBB.Laengentyp;
      Ueberhoehung_End : SBB.Laengentyp;
      Geschwindigkeit : TEXT*20;
    END UeDt;
```

```
    STRUCTURE VrlDt =                  !!Verlauf, M-Wert
      Laenge : MANDATORY SBB.Laengentyp;
    END VrlDt;

    CLASS Weiche =
      BPS : MANDATORY TEXT*4;
      Hauptgleis : BOOLEAN;
      WeichenNummer : MANDATORY 0 .. 9999;
      WeNumZusatz : TEXT*5;
      WeichenArt : MANDATORY (EW,GE,OGE,GTP,MS,ZV,EKW,GD,SW,DW,DKW);
      WeichenNeigung : (N17,N18,N19);
      Ablenkrichtung : (DG,DD);
      WeichenGeometrie : (G,B,O,C);  !!vorläufig nicht MANDATORY
!!    WeichenID : TEXT*20; !!Bedeutung unklar 2022-02-24
      UNIQUE BPS, WeichenNummer, WeNumZusatz;
    END Weiche;

    CLASS Gradient =
      GrVerlauf : MANDATORY POLYLINE WITH (STRAIGHTS) VERTEX SBB.Koord2
       WITHOUT OVERLAPS>0;
      GrVZus : MANDATORY STRUCTURE RESTRICTION (SBB.SBB_Gleisnetz.GrDt);
    END Gradient;

    CLASS Laengsprofil =
      LpVerlauf : MANDATORY POLYLINE WITH (STRAIGHTS) VERTEX SBB.Koord2
       WITHOUT OVERLAPS>0;
      LpVZus : MANDATORY STRUCTURE RESTRICTION (SBB.SBB_Gleisnetz.LprDt);
    END Laengsprofil;

    CLASS Ueberhoehung =
      UeVerlauf : MANDATORY POLYLINE WITH (STRAIGHTS) VERTEX SBB.Koord2
       WITHOUT OVERLAPS>0;
      UeVZus : MANDATORY STRUCTURE RESTRICTION (SBB.SBB_Gleisnetz.UeDt);
    END Ueberhoehung;

    CLASS WeichenPunkt =
      PunktNummer : MANDATORY 1 .. 54;
      Koordinaten : MANDATORY SBB.Koord2;
      Azimut : MANDATORY SBB.Azimuttyp;
      Radius : SBB.Radiustyp;
      WeEndeTyp : (WEA,WES);
 !!   WePunktID : TEXT*14;       !!Bedeutung unklar 2022-02-24
    END WeichenPunkt;

    ASSOCIATION WePunktZuWe =
      hatWePunkt -- {1} Weiche;
      WePtVonWe -- {1..4} WeichenPunkt; !!offenbar allgemein Übergänge
    END WePunktZuWe;   !!WePtVonWe-Kardinalität für Weichen wäre {3..4}

    CLASS WeichenLinie =
      Topologie : MANDATORY 12 .. 544;
      WLVerlauf : MANDATORY POLYLINE WITH (STRAIGHTS,ARCS,CLOTHOIDES)
       VERTEX SBB.Koord2 WITHOUT OVERLAPS>0;
      WLVZus : LIST {0..*} OF SBB.SBB_Gleisnetz.VrlDt;
    END WeichenLinie;
```

```
    ASSOCIATION WeLinieZuWe =
      hatWeLinie -- {1} Weiche;
      WeLinieVonWe (ORDERED) -- {0..*} WeichenLinie;  !!eigentlich: {1..*}
    END WeLinieZuWe;                                  !!wenn Daten vollständig

    CLASS GleisAbschnitt =
!!    WePunktNum_Anf : MANDATORY 1 .. 4; -> Beziehung GleisAbchnittAnfang
!!    WePunktNum_End : MANDATORY 1 .. 4; -> Beziehung GleisAbchnittEnde
      Qualitaet : MANDATORY (M,DM,CM);
      GAVerlauf : MANDATORY POLYLINE WITH (STRAIGHTS,ARCS,CLOTHOIDES)
       VERTEX SBB.Koord2 WITHOUT OVERLAPS>0;
      GAVZus : LIST {0..*} OF SBB.SBB_Gleisnetz.VrlDt;
    END GleisAbschnitt;

    ASSOCIATION GleisAbschnittAnfang =
      AnfangsWePt -- {1} WeichenPunkt;
      AbschnittAnfang -- {0..*} GleisAbschnitt;
    END GleisAbschnittAnfang;

    ASSOCIATION GradiZuGA =
      hatGradient -- {1} GleisAbschnitt;
      GradVonGA -- {0..*} Gradient;
    END GradiZuGA;

    ASSOCIATION LaengpZuGA =
      hatLaengsPr -- {1} GleisAbschnitt;
      LPrVonGA -- {0..*} Laengsprofil;
    END LaengpZuGA;

    ASSOCIATION UeberhZuGA =
      hatUeberh -- {1} GleisAbschnitt;
      UeHoeVonGA -- {0..*} Ueberhoehung;
    END UeberhZuGA;

    ASSOCIATION GleisAbschnittEnde =
      EndWePt -- {1} WeichenPunkt;
      AbschnittEnde -- {0..*} GleisAbschnitt;
    END GleisAbschnittEnde;

  END SBB_Gleisnetz;

END SBB.
```

Anhang 3 INTERLIS-Modell Gleisdaten Struktur DB

```
INTERLIS 2.3;

MODEL DB (de) AT "http://www.gis.ethz.ch" VERSION "2022-05-13"  =

  UNIT
    NeuGrad [gon] = 200.0 / PI [INTERLIS.rad];

  DOMAIN
    Azimutart = 0.00000 .. 399.99999 CIRCULAR [gon];
    BStreckenart = (Eingleisstrecke,Richtungsgleis,Gegenrichtungsgleis,
      Kilometrierungslinie,Bahnhofsgleis,keinGleis);
    DKm = 0.000 .. 200000000.000;
    Gleisart = (unbekannt, Streckengleis, Bahnhofsgleis,
      Streckengleis_ausserBetrieb, Bahnhofsgleis_ausserBetrieb);
    HoehVrbTyp = (Gerade,quadratParabel,ueberhoeWeZwang);
    Knotenart = (Weiche,Gleisende,Streckenwechsel);
    Koord2 = COORD 480000.00000 .. 850000.00000,
                   60000.00000 .. 320000.00000,
                   ROTATION 2 -> 1;
    Laengenart = 0.00000 .. 999999.99999 [INTERLIS.m];
    PAdr = 1 .. 999999; !!Vorläufig nicht TEXT*11, wie DB GND-Definition von 2002
    Radiusart = -30000.00000 .. 30000.00000 [INTERLIS.m];
    Stabilitaet = (unbekannt, gut, mittel gering);
    UeberhoeVrbTyp = (gleichblbUeh, geradeRampe, SFormRampe, BlossRampe,
      GleisschereSForm, GleisschereBloss);
    Verbindungsgeom = (Gerade, Kreis, Klothoide, Bogen_S_Form, Blosskurve,
      Richtgerade, S_Form_einfach, Bloss_einfach);
    Vermarkungsart = (unvermarkt, FBolzen, InMauer, Bodenvermarkung,
      Tunnelvermarkung, Lochstein, Draenrohr, Eisenrohr, indirekt, Hoehenbolzen,
      TPStein, TPPlatte, Grenzstein, Grenzmarke, Nagel, Kreuz, IDBolzen);

   STRUCTURE PAdrS =
     AdS : MANDATORY DB.PAdr;
   END PAdrS;
   !! In ILI2.3 muss STRUCTURE definiert werden für Verwendung in LIST
   !! (oder BAG), wie in Klasse GleiskanteStrecke benötigt


  TOPIC DB_gleisnetz =

    CLASS ElementtHoehe =
      Satztyp : MANDATORY 22 .. 22;
      HoeheSys : MANDATORY TEXT*20;
      Elementtyp : MANDATORY DB.HoehVrbTyp;
      ElLaenge : MANDATORY DB.Laengenart;
      SteigungAnf : MANDATORY 0 .. 1;
      SteiungEnde : MANDATORY 0 .. 1;
    END ElementtHoehe;
```

```
CLASS ElementUeberhoe =
  Satztyp : MANDATORY 23 .. 23;
  Elementtyp : MANDATORY DB.UeberhoeVrbTyp;
  ElLaenge : MANDATORY DB.Laengenart;
  ElUehoehAnf : MANDATORY DB.Laengenart;
  ElUehoehEnde : MANDATORY DB.Laengenart;
END ElementUeberhoe;

CLASS PunktAllgemein =
  Satztyp : MANDATORY 11 .. 11;
  PAD : MANDATORY DB.PAdr;
  Punktart : MANDATORY (LageElPt,KmLinPt,HoehPt,WeichePt,GleisVkPt,
    RefFestPt,andere);
  BStreckenId : TEXT*13;
  Richtung : DB.BStreckenart;
  UNIQUE PAD;
END PunktAllgemein;

CLASS PunktLage =
  Satztyp : MANDATORY 12 .. 12;
  LKoord_Sys : MANDATORY TEXT*3;
  Fremdpunkt : TEXT*14;
  Koordinaten : MANDATORY DB.Koord2;
END PunktLage;

ASSOCIATION PtZuHoeheAnf =
  ElHoeheAnf -- {0..4} ElementtHoehe;
  AnfangsPt -- {1} PunktAllgemein;
END PtZuHoeheAnf;

ASSOCIATION PtZuHoeheEnde =
  ElHoeheEnde -- {0..4} ElementtHoehe;
  EndPt -- {1} PunktAllgemein;
END PtZuHoeheEnde;

ASSOCIATION PtZuUeberhoeAnf =
  ElUeberhoeAnf -- {0..4} ElementUeberhoe;
  AnfangsPt -- {1} PunktAllgemein;
END PtZuUeberhoeAnf;

ASSOCIATION PtZuUeberhoeEnde =
  ElUeberhoeEnd -- {0..4} ElementUeberhoe;
  EndPt -- {1} PunktAllgemein;
END PtZuUeberhoeEnde;

ASSOCIATION PunktLagePAD =
  RolPunktLage -- {1} PunktLage;
  RolPunktAllg -- {1} PunktAllgemein;
END PunktLagePAD;
```

```
CLASS ElementLage =
  Satztyp : MANDATORY 21 .. 21;
  Koord_Sys : MANDATORY TEXT*3;
  AnfangsPt: DB.PAdr;
  EndPt: DB.PAdr;
  Elementtyp : MANDATORY DB.Verbindungsgeom;
  ElLaenge : MANDATORY DB.Laengenart;
  ElRadius_Anf : MANDATORY DB.Radiusart;
  ElRadius_End : MANDATORY DB.Radiusart;
  Richtwinkel_Anf_gon : MANDATORY DB.Azimutart;
  UNIQUE Satztyp, AnfangsPt, EndPt;
END ElementLage;

ASSOCIATION PtZuElLageAnf =
  AnfangsPtRol -- {1} PunktAllgemein;
  ElemlentLageAnfg -- {0..4} ElementLage;
END PtZuElLageAnf;

ASSOCIATION PtZuElLageEnde =
  EndPtRol -- {1} PunktAllgemein;
  ElemlentLageEnde -- {0..4} ElementLage;
END PtZuElLageEnde;

CLASS Gleisknoten =            !!Im Moment (2022-03-31) ist der Zusammenhang
  Satztyp : MANDATORY 31 .. 31;!!zwischen DB- und SBB-Knoten noch nicht klar,
  KnotenNr : MANDATORY 1..999999;!!vorerst wird daher auf Trafo der Netze
  KnTyp : MANDATORY DB.Knotenart; !!verzichtet und das DB-KN-Modell schwach.
  KN_ankommend : TEXT*15; !!Vorerst nicht MANDATORY, Bez: KNzuKNankommend
  KN_abgehend1 : TEXT*15; !!Vorerst nicht MANDATORY, Bez: KNzuKNabgehend1
  KN_abgehend2 : TEXT*15; !!Zukunftlösung mit Beziehungen: KNzuKNabgehend2
  KN_Beschreibung : MANDATORY TEXT*40;
  UNIQUE KnotenNr;
END Gleisknoten;

ASSOCIATION GleisknotenPAD =
  Knotengeom -- {1} PunktAllgemein;
  RolGleisknoten -- {0..1} Gleisknoten;
END GleisknotenPAD;

ASSOCIATION KNzuKNabgehend1 =
  KN_abg1 -- {0..1} Gleisknoten;    !!Vorerst nicht {1}
  nachAbg1 -- {0..1} Gleisknoten;   !!Vorerst nicht {1}
END KNzuKNabgehend1;

ASSOCIATION KNzuKNabgehend2 =
  nachAbg2 -- {0..1} Gleisknoten;
  KN_abg2 -- {0..1} Gleisknoten;     !!Vorerst nicht {1}
END KNzuKNabgehend2;

ASSOCIATION KNzuKNankommend =
  KN_ank -- {0..1} Gleisknoten;       !!Vorerst nicht {1}
  nachAnk -- {0..1} Gleisknoten;      !!Vorerst nicht {1}
END KNzuKNankommend;
```

```
    CLASS GleiskanteStrecke =
      Satztyp : MANDATORY 33 .. 33;
      BStreckenId : MANDATORY 1 .. 999999;  !!Vorerst statt TEXT*13
      Richtung : MANDATORY DB.BStreckenart;
      Anz_Lagepunkte : MANDATORY 0 .. 999;
      Anz_Hoehenpunkte : MANDATORY 0 .. 999;
      Anz_Ueberhoehungspunkte : MANDATORY 0 .. 999;
      LaPt : LIST {2..*} OF DB.PAdrS;
      HoePt : LIST {0..*} OF DB.PAdrS;
      UhPt : LIST {0..*} OF DB.PAdrS;
!!    UNIQUE StreckeKnotenAnf.AnfKnoten, StreckeKnotenEnd.EndKnoten;
    END GleiskanteStrecke;

    ASSOCIATION StreckeKnotenAnf =
      AnfKnoten -- {1} Gleisknoten;
      BStreckenAnfg -- {0..4} GleiskanteStrecke;
    END StreckeKnotenAnf;

    ASSOCIATION StreckeKnotenEnd =
      EndKnoten -- {1} Gleisknoten;
      BStreckenEnde -- {0..4} GleiskanteStrecke;
    END StreckeKnotenEnd;

   ASSOCIATION GleisPunkt =
     PunkteAufKante (ORDERED) -- {0..*} PunktAllgemein; !!Vorerst statt {2..*}
     KanteMitPt -- {0..4} GleiskanteStrecke;            !!Vorerst statt {1..4}
   END GleisPunkt;

  END DB_gleisnetz;

END DB.
```

Anhang 4 INTERLIS-Modell neue UMLT/ILIT-Funktionen

```
INTERLIS 2.3;

CONTRACTED MODEL ILITfct (de) AT "http://www.gis.ethz.ch" VERSION "2022-03-31" =

  UNIT
    NeuGrad [gon] = 200.0 / PI [INTERLIS.rad];

  DOMAIN
    Azimuttyp = 0.00000 .. 399.99999 CIRCULAR [gon];
    Koord2 = COORD 480000.00000 .. 850000.00000 [INTERLIS.m],
                   60000.00000 .. 320000.00000 [INTERLIS.m],
                   ROTATION 2 -> 1;
    Laengentyp = -99999.00000 .. 999999.00000 [INTERLIS.m];
    LinienGeomTyp = (Gerade,Kreis,Klothoide);
    NaturalaZ = 0 .. 9999999;   !!Natural and Zero
    PAdr = 1 .. 999999;   !!vorläufig nicht TEXT*11, wie in DB GND für PAD;
    Radiustyp = -99999.9999 .. 999999.0000 [INTERLIS.m];
    Real = -9999999.9999 .. 9999999.9999;

  STRUCTURE CLOTHO (FINAL) EXTENDS INTERLIS.LineSegment =
    Ra : MANDATORY Radiustyp;
    Re : MANDATORY Radiustyp;
    A : MANDATORY Azimuttyp;
  END CLOTHO;

  LINE FORM
    CLOTHOIDES : CLOTHO;

  DOMAIN
    PolylineStraight = POLYLINE WITH (STRAIGHTS) VERTEX Koord2
     WITHOUT OVERLAPS>0;
    PolylineGeneral = POLYLINE WITH (STRAIGHTS, ARCS, CLOTHOIDES) VERTEX
     Koord2 WITHOUT OVERLAPS>0;
    SurfaceGeneral = SURFACE WITH (STRAIGHTS, ARCS, CLOTHOIDES) VERTEX
     Koord2 WITHOUT OVERLAPS>0;


  STRUCTURE PAdrS =
    AdS : MANDATORY PAdr;
  END PAdrS;

  STRUCTURE LisPATyp =
    LsPA : LIST {2..*} OF PAdrS;
    nPA : MANDATORY 2 .. 9999;
  END LisPATyp;

  STRUCTURE LgTyp =
    LgT : MANDATORY Laengentyp;
  END LgTyp;
```

```
  STRUCTURE LinElTyp =
    coord_Sys : TEXT;
    StartPt : MANDATORY Koord2;
    EndPt : MANDATORY Koord2;
    gmtyp : MANDATORY LinienGeomTyp;
    elLaenge : MANDATORY Laengentyp;
    elRadius_Anf : Radiustyp;
    elRadius_End : Radiustyp;
    elRichtWinkel_Anf_gon : Azimuttyp;
  END LinElTyp;


  FUNCTION Incrementer
   (counter: TEXT)
   : NaturalaZ;
  /* Initialisieren und Erhöhen des ganzzahligen Zählers counter um 1. Beim
     ersten Aufruf der Funktion mit dem Zählernamen counter wird dieser Name
     gespeichert und der entsprechende Zähler auf 1 gesetzt. Bei jedem
     weiteren Aufruf wird dieser Zähler um 1 erhöht.
*/
  FUNCTION CollectorNumToList
   (number: PAdr)
  : LIST {1..*} OF PAdrS;
  /* Sammelt ganzzahlige Werte number zu einem Array. Nach Eingabe des letzten
     Wertes number ist als Rückgabewert der Array (INTERLIS LIST OF) der
     gesammelten ganzzahligen Werte verfügbar.
 */
  FUNCTION PointNrGenerator
   (geometry: Koord2)
  : PAdr;
  /*Prüft, ob geometry vom Typ Koord2 bereits vorhanden ist in der HashMap der
    Punktkoordinaten. Falls Ja, wird die bereits existierende Punkt Nr Typ PAdr
    zurückgegeben. Falls Nein, wird eine neue Punkt Nr erzeugt, zurückgegeben und
    mit den Koordinaten des neuen Punktes in der HashMap eingefügt.
  */
  FUNCTION NodeNrGenerator
   (refPointNr : PAdr)
  : PAdr;
  /*Prüft, ob die Referenzpunkt-Nummer refPointNr des neuen Knotens bereits
    vorhanden ist in der HashMap der Referenzpunkt-Nummern. Falls Ja, wird die
    bereits existierende Knoten Nr Typ PAdr zurückgegeben. Falls Nein, wird eine
    neue Knoten Nr erzeugt, zurückgegeben und mit der Referenzpunkt Nr des neuen
    Knotens in der HashMap eingefügt.
  */
  FUNCTION SplitLinToLinEl
   (geometry: PolylineGeneral;
    nSegm: 1 .. 35)
  : LIST {1..*} OF LinElTyp;
  /*Zerlegt eine Linie in Linienelemente. Mit dem Parameter geometry wird das
    Attribut übergeben, das vom Geometrietyp Line ist (INTERLIS Typ PolylineGe-
    neral, d.h. POLYLINE mit Geradenstücken, Kreisbogen und Klothoiden). Der
    zweite Parameter nSegm enthält die Anzahl der Segmente in der Linie geometry
    (jetzt nötig aus programmtechnischen Gründen!!). Der Rückgabeparameter ist
    ein Array (INTERLIS LIST OF) vom Typ Linienelement, der durch die INTERLIS
    Struktur LinElTyp definiert ist.
  */
END ILITfct.
```

Anhang 5 Die neue Funktion `PointNrGenerator` in Java

```
package fktT;

import java.util.HashMap;

public class F_PointNrGenerator {   //Point Number Generator (NOT oid!)
  String hashKey = "";
  int    ptID;           //fortlaufende neue Identifikation
  int    ptIDex = 0;     //existierende in ptStore gespeicherte Nummer
  final  HashMap<String,Integer> ptStore = new HashMap<String,Integer>();
  String strCoord1 = "";
  String strCoord2 = "";
  boolean YKeyIsPresent; //ist für diese Koord schon PtNr da? Ja = true

  public F_PointNrGenerator (){    //Point Number Generator
    ptID = 0;
  }

  public int getNr(double coord1I, double coord2I) {
    hashKey = String.valueOf(coord1I) + "C" + String.valueOf(coord2I);
    YKeyIsPresent = ptStore.containsKey(hashKey);
    if (YKeyIsPresent)  ptIDex = ptStore.get(hashKey);
    else {
      ptID = ptID + 1;
      ptStore.put(hashKey,ptID);
      ptIDex = ptID;
    }
    return ptIDex;
  }

  public boolean getYpaExisted() {
    return YKeyIsPresent;
  }

}
```

Anhang 6 UMLT/ILIT, Elemente der Sprachdefinition

UMLT verfolgt drei wesentliche Grundsätze:

Grundsatz 1: Die Transformationsdefinition zwischen objektorientierten Datenmodellen soll ebenfalls objektorientiert stattfinden. Der Benützer von UMLT soll seine objektorientierten Umbauideen nicht selbst auf den objektrelationalen oder relationalen Formalismus reduzieren müssen, wie es etwa bei der Strukturumbau-Definition direkt mit FME nötig ist.

Grundsatz 2: Der Benützer von UMLT soll nicht aufwendig programmieren müssen. Die notwendigen Transformationselemente stehen als Funktionen zur Verfügung. Die „Programmierung“ vom UMLT besteht darin, den Ziel-Attributs-Werten die passenden Ausgangs-Attributs-Werte zuzuordnen und dabei allenfalls geeignete Umbaufunktionen aufzurufen. Zur übersichtlichen Gliederung des gesamten Strukturumbaus können diese Zuordnungen (mit Funktionsaufrufen) zu sogenannten Transformationsaktivitäten (kurz TrafoActions) zusammengefasst werden, die über Output- und Input-Pins Attribut-Werte austauschen.

Grundsatz 3: Wenn zwei TrafoActions über Pins Attribut-Werte austauschen, dann gilt: Was der o-Pin der Start-TrafoAction liefert, wird über den i-Pin von der Ziel-TrafoAction genau verstanden, d.h., die Beschreibung des transferierten Wertes stimmt bei der Start-TrafoAction und bei der Ziel-TrafoAction genau überein. Der Benützer von UMLT kann die entsprechenden Pins zusammenhängen ohne zusätzlich die Schnittstelle programmieren zu müssen.

Im Prinzip sollten die Beziehungsrollen-Werte automatisch berechnet werden können, wenn man Start- und Ziel-Datenmodell zur Verfügung hat. Es gibt eine Beschreibung, wie bei der Realisierung des mit UMLT beschriebenen Strukturumbaus für den Datenumbau mit Hilfe von FME Beziehungen zwischen Start- und Ziel-Klassen zu berücksichtigen sind. Diese Beschreibung ist nur für einen Detailkenner von FME hilfreich bei der Übersetzung der UMLT-Umbaudefinition auf ein Java-Datenumbauprogramm.

Die Funktionsdefinitionen mit Signaturen und Parametern sind in Kapitel **Error! Reference source not found.** in UMLTtex beschrieben und in Anhang 4 in INTERLIS mit dem Datenmodell `ILITfct`. Zum Zusammenhang zwischen den beiden (textuellen) formalen Sprachen UMLTtex und INTERLIS / ILIT gibt die Abbildung 15 eine unvollständige Liste von Beispielen.

Nr	UMLTtex	INTERLIS DDL, ILIT
1	`int`	`1..*`
2	`float`	`-9999.99 .. 9999.99`
3	`String`	`TEXT, TEXT*12`
4	`Point`	`LKoord, HKoord` (mit Datentyp `COORD` definiert)
5	`Line`	`POLYLINE` (mit Details)
6	`Polygon`	SURFACE oder AREA (mit Details)
7	`Polygon[]` (= Array von `Polygon`)	`LIST OF {} SurfaceStructure`
8	`PAL2KNGS` (Start der TrafoAction)	`TRAFO_ACTION PAL2KNGS =` (Start der TrafoAction) `END PAL2KNGS;` (Ende der TrafoAction)
9	`Kn_1.KnotenNm_Anf := NodeNrGenerator(lpai.StartPtNr)`	`Kn_1->KnotenNm_Anf := NodeNrGenerator(lpai->StartPtNr);`

Abbildung 15: Vergleich der formalen Sprachen (und HUTN) UMLTtex und ILIT (Beispiele)

Start- und Zielmodell sollten nicht abgeändert werden müssen, um die Definition der Trafo mit Strukturumbau zu ermöglichen. Für die Definition spezieller Werte, die über Pins zwischen TrafoActions auszutauschen sind eignet sich ebenfalls das Datenmodell `ILITfct`, das wie die Datenmodelle der Start- und Ziel-Daten in die UMLTtex- bzw. ILIT-Beschreibung des Strukturumbaus (Nr. (13) in Abbildung 1) und in den Strukturumbau Prozessor (Nr. (14) in Abbildung 1) importiert wird.

Anhang 7 Projektergebnisse

Die Projektergebnisse sind auf USB-Stick in Ordnern verfügbar, deren Nummerierung und Titel denjenigen von Abbildung 1 in Kapitel 3 entsprechen.

3_SBB_GleisdatenProprFmt

- `SBB_Gleisschnittstelle_Toporail_V2002.docx`: Formatbeschreibung der SBB Startdaten vom CSV-Typ und ASCII-codiert.
- `allModifV1.txt`: Testdatei mit SBB-Startdaten Raum Schaffhausen.

5_SBB_UML-ILI-Modell

- `SBB_railV8_geom.uml`: UML/INTERLIS-Editor Datei mit SBB-Datenmodell.
- `SBB_railV8_geom_UML.pdf`: UML-Klassendiagramm des SBB-Modells in pdf-Format.
- `SBB_railV8_geom.ili`: INTERLIS-Datei des SBB-Startmodells.

8_SBB_1To1Proc-ProprFmt2Java2XTF

Der Testrahmen-Ersatz enthält drei 1:1-Prozessoren für die SBB-Startdaten:

- `Methode_sbb1to1_AsciiToJava.java`: SBB-Daten-Reader zur Umformatierung der SBB-Startdaten im ASCII-CSV-Format in die interne Datenstruktur des Java-Programms für den Testrahmen-Ersatz.
- `Methode_sbb1to1_JavaToITF.java`: Aus der internen Java-Struktur die SBB-Startdaten im INTERLIS 1 Transferformat ITF herstellen.
- `Methode_sbb1to1_JavaToXTF.java`: Aus der internen Java-Struktur die SBB-Daten im INTERLIS 2 Transferformat XTF herstellen zwecks Prüfung mit ilivalidator und IG-Checker.

Diese 1:1-Prozessoren sind realisiert als Methoden der Java-Klasse `TestRahmenFktT.java`. Details zu dieser Java-Klasse siehe im Abschnitt 14_StrukturumbauProzessor.

9_SBB_GleisdatenXTF

- `SBB_T52.xtf`: Vollständige Testdaten, insbesondere mit Klothoiden.
- `SBB_T54_noClotho.xtf`: Mit Geradenstücken anstelle von Klothoiden.
- `Test51_SBBxtfValidiert.txt`: ilivalidator-Protokoll der vollständigen Testdaten. Jedes Auftreten von Klothoiden (mit Tag <SBB.CLOTHO>) wird als Fehler gemeldet, da das Prüfprogramm ilivalidator (wie auch der IG-Checker) benutzerdefinierte Verbindungsgeometrien noch nicht bearbeiten können
- `Test53_SBBxtfChecker_noClotho.log`: IG-Checker-Protokoll für SBB-XTF-Daten mit Geradenstücken statt Klothoiden. Nach Entfernen der Klothoiden kommen noch einige inhaltliche Fehler der SBB-Daten besser zum Vorschein (sie sind auch im ilivalidator-Protokoll Test51_SBBxtfValidiert.txt zu finden): 4 Weichenlinien haben gar kein Liniensegment und 5 Gleisabschnitte haben nur ein Segment der Länge 0.

10_DB-GleisdatenProprFmt

- `DB_PunktAllgemein-PunktLage_GND-Spezifikation.pdf`: ASCII-Schnittstellen Beschreibung für Satzart 11 Punkt allgemein und Satzart 12 Punkt Lage aus [12] „Spezifikation Bearbeiten und Prüfen von Gleisnetzdaten“.
- `DB_ElementLage_GND-Spezifikation.pdf`: ASCII-Schnittstellen Beschreibung für Satzart 21 Element Lage aus [12].

- `DB_Gleisknoten_GND-Spezifikation.pdf`: ASCII-Schnittstellen Beschreibung für Satzart 31 Knoten aus [12].
- `DB_GleiskanteStrrecke_GND-Spezifikation.pdf`: ASCII-Schnittstellen Beschreibung für Satzart 33 Gleiskante Strecke aus [12].

12_DB_UML-ILI-Modell

- `DB_railV8_geom.uml`: UML/INTERLIS-Editor Datei mit DB-Datenmodell.
- `DB_railV8_geom_UML.pdf`: UML-Klassendiagramm des DB-Modells in pdf-Format.
- `DB_railV8_geom.ili`: INTERLIS-Datei des DB-Zielmodells.

13_StrukturumbauDef-UMLT

- `TrafoUMLT_RailSBB2DB_V8.pptx`: Definition des Strukturumbaus auf konzeptionellem Niveau vom Startmodell der SBB-Gleisdatenstruktur ins Zielmodell der DB-Gleisdatenstruktur grafisch mit UMLT und als Text mit UMLTtex. Da die Test-Implementierung des UMLT-Editors dazu nicht verwendet werden konnte, findet sich hier nur die Entwurfsvorstufe der UMLT / UMLTtex Strukturumbau-Definition mit Hilfe von MS Powerpoint.
- `TrafoUMLT_RailSBB2DB_V8`.pdf: pdf-Version der obigen pptx-Datei.

14_StrukturumbauProzessor

Da die beim Projekt SumSuG [11] verwendete Implementierung von UMLT/UMLTtex hier nicht zur Verfügung stand, musste im Ersatz-Testrahmen auch der Strukturumbau-Prozessor (in Abbildung 1 mit (14) als Nummer) programmiert werden: Die dazu entwickelte Software ist ein Java-Projekt mit dem Namen `UMLT-Funktionen-Testbed` in der Eclipse Entwicklungsumgebung. Das Java-Projekt `UMLT-Funktionen-Testbed` enthält als Java-Package `fktT` das Demobeispiel mit Gleisdatenumbau von SBB nach DB. Dieses Java-Package `fktT` umfasst die folgenden Klassen mit ihren Methoden. Diese Liste enthält auch eine Kurzbeschreibung der Bedeutung von deren Namen und Aufgaben.

- `TestRahmenFktT.java` ist das Hauptprogramm mit folgenden Methoden:
  - `sbb1to1_AsciiToJava` ist der SBB-Daten-Reader: Die SBB-Daten im proprietären CSV-ASCII-Format werden eingelesen in die interne Java-Datenstruktur.
  - `sbb1to1_JavaToITF`: Aus der internen Java-Datenstruktur wird das INTERLIS 1 Transferformat ITF hergestellt für die SBB-Daten.
  - `sbb1to1_JavaToXTF`: Analog das INTERLIS 2 Transferformat XTF herstellen für die SBB-Daten.
  - `WP2PAPL`: TrafoAction baut Weichenpunkte um nach Punkte allgemein und Punkte Lage.
  - `GA2LINEL`: TrafoAction zerlegt Gleisabschnitte in Linienelemente.
  - `LINEL2PAPLEL`: TrafoAction baut Linienelemente um in Punkte allgemein, Punkte Lage und Elemente Lage.
  - `PAL2KNGS`: TrafoAction baut Punkte allgemein (als Liste) um in Knoten und Gleiskanten Strecken.
  - `db1to1_JavaToITF`: 1:1-Prozessor, der die DB-Daten aus der internen Java-Struktur umformatiert in das INTERLIS 1 Transferformat ITF.
  - `db1to1_JavaToXTF`: Analoger 1:1-Prozessor zum Umformatieren der DB-Daten aus der internen Java-Struktur in das INTERLIS 2 Transferformat XTF.
  - `main`: Steuert den Gesamtablauf.

- Neben dieser Hauptklasse `TestRahmenFktT.java` gibt es im Java-Package `fktT` weitere Typen / Klassen mit der Namensstruktur `A_xxx` für den komplexen Attributstyp `xxx`:
  - `A_GaObj.java`: Gleisabschnitt-Objekt mit Startindex und Anzahl Segmenten des Gleisabschnitts in der allgemeinen ArrayList `gasegs` der POLYLINE-Segmente.
  - `A_LinElTyp.java`: Linienelement-Typ für die durch Zerlegung von INTERLIS `POLYLINE`s entstandenen Linienelemente als Vorbereitung für die DB-Objekte Element Lage mit Koordinaten der beider Endpunkte und Verbindungsinformation, aber noch ohne Punktnummern.
  - `A_SegPly.java`: Segmente von INTERLIS `POLYLINE`s entsprechend dem INTERLIS-Datentyp `POLYLINE WITH (STRAIGHTS,ARCS,CLOTHOIDES)`.
  - `A_WlObj.java`: Weichenlinien-Objekte: Startindex und Anzahl Segmente für jede Weichenlinie in der allgemeinen ArrayList `wlsegs` der POLYLINE-Segmente.

- Ferner gibt es im Package `fktT` Klassen mit der Namensstruktur `F_yyy.java` für die bei der Umbaudefinition verwendete Funktion `yyy`:
- `F_CollectorNumToList.java`: Sammelt Nummern und bildet daraus eine Liste.
- `F_Incrementer.java`: Initialisiert und erhöht Zähler.
- `F_NodeNrGenerator.java`: Generiert eindeutige Knotennummer.
- `F_OidGenerator.java`: Generiert Objektidentifikator.
- `F_PointNrGenerator.java`: Generiert eindeutige Punktnummer.
- `F_SplitLinToLinEl.java`: Zerlegt Linien in Linienelemente.
- F_StringConcatenator.java: Hängt Texte zusammen.
- `F_ValueMapper.java`: Macht Wertzuweisungen gemäss Korrespondenztabelle.

- Weiter enthält der Ersatz-Testrahmen im Java-Package `fktT` gemäss den Input-Pins von UMLT/UMLTtex die folgenden Input-(Objekt-)Klassen der SBB:
  - `we_Weiche.java`
  - `wp_Weichenpunkt.java`
  - `wl_Weichenlinie.java`
  - `ga_Gleisabschnitt.java`

- Und schliesslich gibt es in `fktT` auch noch die folgenden Output-(Objekt-)Klassen für die DB entsprechend den Output-Pins von UMLT/UMLTtex
  - `pa_PunktAllgemein.java`
  - `pl_punktLage.java`
  - `el_ElementLage.java`
  - `kn_Gleisknoten.java`
  - `gs_GleikanteStrecke.java`

- Die folgende Hilfsklasse zerlegt die Zeilen der SBB-ASCII-CSV-Datei in die einzelnen Felder
  - `ReadCSVFile.java`

15_DB-GleisdatenXTF

- `DB_T61.xtf`: Vollständige Testdaten.
- `Test63_DBxtfValidator.txt`: ilivalidator-Protokoll der vollständigen DB-Testdaten
- `Test63_DBxtfChecker.log`: IG-Checker-Protokoll für DB-XTF-Daten

Beide Prüfprogramme melden, dass drei Objekte der Klasse `ElementLage` die Eindeutigkeitsbedingung verletzen. Die Attribute `Satztyp, AnfangsPt, EndPt` haben in allen drei Objekten dieselben Werte. Im Abschnitt „3_SBB_GleisdatenProprFmt“ zeigte die Kontrolle der SBB-Originaldaten, dass in Startdatei `allModifV1.txt` zwei Gleisabschnitte in drei (mittleren) Segmenten genau übereinstimmen, obschon ihr Name verschieden ist. Zur Lösung dieses Problems brauchte es Rücksprache mit den entsprechenden SBB-Fachleuten.

Projektbericht

- `SumSuGFunktionenTest_BerichtV1.1.pdf`: Definitiver Projektbericht

Dank

Die Durchführung dieses 1-Mann-Projektes war auf mannigfache Unterstützung angewiesen. Mein herzlicher Dank dafür geht zunächst an Rolf Buser, Christine Najar und Rolf Zürcher von swisstopo / KOGIS für den Auftrag zu diesem Projekt und für dessen Finanzierung sowie an die Firma RMP Consulting als Auftragnehmer mit perfektem Projektmanagement und unzähligen praktischen Hinweisen durch Martina Münster. Unentbehrliche Voraussetzung für die Bewilligung dieses Projektes war die Empfehlung des Themas durch die GIS-Fachstelle des Kantons Bern und deren Leiter Thomas Hardmeier. Merci vielmals.

Verschiedene Gespräche mit den UMLT/UMLTtex/ILIT–„Natives“ Tatjana Kutzner (TU München), Peter Staub (KGK) und Claude Eisenhut (Eisenhut Informatik AG) haben zur Klärung wesentlicher Fragen beigetragen, herzlichen Dank. Ebenfalls herzlich danken möchte ich Nicole Stahel für die wertvollen Vorarbeiten zu 1:1-Prozessoren im Rahmen des GeoRail-Projektes. Der Projektkoordinator Stefan Henrich (moflex Infra GmbH) hat durch präzise und kritische Lektüre der ersten eingereichten Berichtsversion darin verschiedene Lücken und Unverständlichkeiten festgestellt, deren Beseitigung den Text wesentlich zu verbessern erlaubten, auch dafür herzlichen Dank. Ferner geht mein Dank an die SBB und an die DB. Beide Bahnen haben die Durchführung dieses Projektes überhaupt erst ermöglicht, indem sie die Format-Beschreibungen für den Transfer ihrer Gleisdaten zur Verfügung stellten, die SBB zusätzlich umfangreiche Test-Daten aus dem Grenzraum Schaffhausen.

Schliesslich möchte ich auch meiner Frau Marianna herzlich danken für die not-wendige und geduldige Unterstützung.