



# Überlegungen zu einer gemeinsamen Schema-Transformations- und Daten- Konversions-Sprache



Ein Dokument der Serie "Model-driven Method Memos" (MM-Serie)

Prof. Stefan F. Keller, sfkeller@hsr.ch  
Center int>e>gis, Institut ITA, Hochschule Rapperswil

Version 1.1de vom 2003-04-21

Dokument 'MM\_SchemaTrans\_v01d.doc'

Aktuellste Version jeweils auf [www.integis.ch](http://www.integis.ch)

## Inhaltsüberblick

<b>1</b>	<b>Allgemeines</b> .....	<b>2</b>
1.1	Einleitung: Transformation und Konversion.....	2
1.2	Nutzen.....	4
1.3	Begriffe.....	4
<b>2</b>	<b>Anforderungen</b> .....	<b>5</b>
2.1	Funktionale Anforderungen.....	5
2.2	Usability-Anforderungen.....	6
<b>3</b>	<b>Stand der Technik</b> .....	<b>6</b>
3.1	Softwarewerkzeuge.....	6
3.2	Sprachen.....	7
<b>4</b>	<b>Diskussion im Hinblick auf einen Entwurf</b> .....	<b>8</b>
4.1	Allgemeines.....	8
4.2	Warum nicht XSLT, bzw. XML?.....	8
4.3	Hauptaspekte einer möglichen Sprache.....	8
4.4	Entwurf einer Sprache.....	9
<b>5</b>	<b>Szenarien und Use Cases</b> .....	<b>12</b>
5.1	Szenarien und Use Cases.....	12
5.2	Beispiel Adressen.....	12
	<b>Anhang: Literatur</b> .....	<b>14</b>



Fig. 1. Ausschnitt aus einer Werbung (OO-DB versus Relational...)

## 1 Allgemeines

### 1.1 Einleitung: Transformation und Konversion

Die sogenannte 'Schema-Transformation' (-Abbildung, -'Mapping') ist überall dort notwendig, wo Daten transferiert, integriert oder ausgetauscht werden sollen und wo das den Daten zugrunde liegende Quellschema nicht dem Zielschema entspricht. 'Schema-Transformation' bedeutet, dass ein bestehendes Schema in ein neues Schema umgewandelt wird - und damit auch die Daten selber.

Auf Daten-Ebene spricht man eher von einer Konversion, wobei diese auch stattfinden kann ohne Schema-Transformation, wie das bei Format-Konversionen der Fall sein kann. Hier bleibt das Schema gleich und man spricht von einer 1:1-Konversion.

Ein typisches Beispiel für eine 'echte' Transformation mit entsprechender Daten-Konversion ist, wenn z.B. zwei Personen ihre Adressen austauschen wollen und im Quellschema Name und Vorname zusammen in einem Attribut verwaltet werden und im Zielschema Name und Vorname je als eigenes Attribut erwartet werden. Zusätzlich könnte gewünscht sein, dass eine Klasse (aufgrund ihrer Attributwerte) in mehrere aufgeteilt werden muss oder umgekehrt (vgl. Kapitel Szenarien und Use Cases). Ein anderes typisches Problem ist, wenn ein konzeptionelles Schema (z.B. als UML-Diagramm beschrieben) in ein internes Schema (z.B. ein Datenbankmodell) umwandeln muss. In diesem Falle müssen z.B. assoziierte Klassen in Tabellen mit Identifikatoren und Primär-/Fremdschlüssel abgebildet werden.

Damit diese und andere Prozesse verständlich kommuniziert, dokumentiert und ausgetauscht werden können, wäre eine 'Schema-Transformations'-Sprache hilfreich.

Eine formale Schema-Transformations-Sprache dient der Beschreibung der Abbildung von Objekten zwischen vordefinierten konzeptionellen Schemata, die gemeinsame semantische Elemente enthalten aber für deren Modellierung unterschiedliche Formen gewählt wurden.

Eine solche genormte Sprache gibt es bislang nicht und auch entsprechende Initiativen sind noch rar oder unvollständig [CWM].

Dabei ist klar, dass es Grenzen gibt bei der Umtransformation von Schemas. Daher folgende Klassifikation:



Figure 4.1: Scale of schema relationships

Fig. 2. Klassifikation der Mengen-Beziehungen zwischen zwei Schemas [Kindingstad].

Werden zwei Schemas miteinander verglichen, können vier verschiedene Möglichkeiten eintreten (vgl. auch Figur 1).

1. Die zwei Schemas sind völlig identisch (engl. Equivalent).
2. Schema B ist eine Teilmenge vom Schema A (engl. Inclusion).
3. Die beiden Schemas haben nur eine gemeinsame Schnittmenge (engl. Overlapping).
4. Die Schemas haben keine Gemeinsamkeiten (engl. Disjoint).

Um die Äquivalenz oder Nähe zweier Schemas zu bezeichnen ist vielleicht folgende "semantische Äquivalenz-Kategorien" hilfreich:

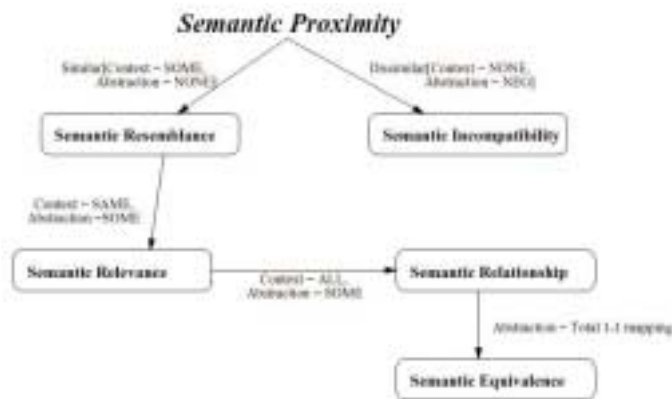


Figure 4.2: Semantic proximity: a taxonomy

Fig. 3. Taxonomie von "semantischen Äquivalenz-Kategorien" [Kindingstad].

Davon sind diejenigen Schemas mit 'semantischer Äquivalenz' und 'semantischer Beziehung', die Erfolgversprechendsten für eine automatische Transformation.

Strukturelle Konflikte müssen nicht immer eine entsprechende semantische "semantische Äquivalenz-Kategorie" zugeordnet haben. Trotzdem ist eine grobe Zuordnung evtl. sinnvoll:

Comparison Structural/Semantical	
Synonymy	Semantic equivalence
Homonymy	Semantic incompatibility
Data representation conflicts	Semantic equivalence
Data unit conflicts	Semantic equivalence
Data precision conflicts	Semantic relationship
Default value conflicts	Semantic relevance
Integrity constraint conflicts	Semantic resemblance

Table 4.6: Structural conflicts and their semantic proximity

Fig. 4. Strukturelle Konflikte und ihre 'semantische Nähe' [Kindingstad].

## 1.2 Nutzen

Vorteile und Nutzen einer gemeinsamen Schema-Transformations-Sprache sind:

- Aus allgemeiner Sicht - vor allem derjenigen der Datenherren und Nutzer - gelten hier die gleichen Vorteile wie bei der modell-basierten Methode, da es sich um ein weiteres 'Familienmitglied' der genormten, modell-basierten konzeptionellen Sprachen handelt:
  - Gemeinsame Verständigung
  - Langfristige Dokumentation, Investitionsschutz (in Daten, Modelle und Konfigurationen), Unabhängigkeit schnellelebiger 'de facto-Standards'
- Aus Sicht eines Softwareherstellers:
  - können damit die eigenen Kunden mit ihrer gewohnten Oberfläche einfacher Skripts erzeugen,
  - sie können diese für andere Konverter-Tools verwenden (die z.B. andere Formate beherrschen und damit Daten für das eigene System erschliessen).

## 1.3 Begriffe

Im folgenden eine ungeordnete Sammlung von wichtigen Begriffen in diesem Kontext.

Die Verweise (->) beziehen sich entweder auf diese Sammlung oder auf das (auf der Norm SN 612031 und weitere offizielle Quellen basierenden)

Online-Glossar auf <http://www.integis.ch> > INTERLIS-Treffpunkt.

Konventionen und Abkürzungen siehe dort.

*Quellschema* (en: *Source schema*) und *Zielschema* (en: *Target schema*; *Senke*):

Ausgangs- und Ende-Punkt einer -> Konversion oder Transformation.

*Transformation*

Mehrdeutig; vgl. auch u.a. -> Koordinatensystem; Linguistik; Physik.

Begriff für die Abbildung von Elementen eines konzeptionellen Quellschemas in Elemente gemäss einem (anderen konzeptionellen) Zielschema mittels -> Regeln; damit kann direkt eine -> Konversion von Objekten verbunden sein.

Syn. (en:) 'Mapping', Translation. Enthält (Schema-)Transformationsregeln

*Schema-Transformations-Sprache*

dient der Beschreibung der Abbildung von Objekten zwischen vordefinierten konzeptionellen Schemata (-> Quell- und Zielschematas), die gemeinsame semantische Modellierungs-Elemente enthalten aber für deren Modellierung unterschiedliche Formen gewählt wurden.

*Konversion*

Mehrdeutig; vgl. u.a. auch -> Koordinatensysteme.

Umformatieren von Transferdateien, d.h. Formatumbau der strukturellen Repräsentation von (z.B. serialisierten) Objekten. Könnte auch als 'Transformation' von Objekten gemäss intern-physikalischem Schema bezeichnet werden. Substantive: Übersetzer, Konverter, Translator.

*Integration*

[Def. ?]

Verlangt und enthält Konversion, oft auch Transformationen (mit gleichzeitiger Konversion).  
Substantive: Integrationsprogramm, Mediator

### Regel

Abbildungsvorschrift (mathem. Formel) gemäss einer formalen Sprache

## 2 Anforderungen

### 2.1 Funktionale Anforderungen

Hier eine nicht ganz systematische Zusammenstellung funktionaler Anforderungen, die sich z.T. widersprechen; eine Gewichtung folgt anschliessend.

1. 'Matchen' ganzer Quell-Schema-Elementen (Model/Topic/Klasse/Attribut) durch ihre Namen.
2. 'Matchen' von Elementen mittels Typen (auch Instanzen von of Subtypen, wie LIST/BAG OF) und exakte Typen (würde Instanzen von Subtypen ausschliessen).
3. Filtern der gefundenen 'matched' Objektmenge mittels Assoziationen, Attributwerte, und anderen Kriterien. Einfacher Umgang mit optionalen Attributwerten.
4. Definieren von verschiedenen (nicht-uniformen) Transformationen. Z.B. könnte verlangt sein, dass die 'matched'-Elemente eine Rangordnung erhalten, die von völlig unabhängigen Funktion abhängen; Transformations-Regeln müssen Vorgänger, Nachfolger, Index, erstes und letztes Element in einer Menge identifizieren können.
  - Aggregation und 'Splitting' von Klassen (inkl. der entsprechenden Assoziationen)
  - Umgang mit Assoziationen; Beziehungen (Identifikatoren) hinzufügen zwischen Quell- und Ziel-Schema-Elemente (evtl. implizit).
  - Umdefinieren von Typen, Aufzähltypen und Wertebereichen (Nachkommastellen...)
  - Initialisieren von Attributen mit Defaults, Erzeugen von TIDs
  - Weitere Transformationen angelehnt an [WAD] sind: Umbenennung und Typänderung; Klasse in Assoziation und umgekehrt; Generalisierung/Spezialisierung; Verschmelzen von zwei Klassen zu einer Klasse und umgekehrt das Abspalten von Klassen; Umwandlung der Stärken von Beziehungen: Assoziation, Aggregation und Komposition; Verschiebung von Attributen in einer Vererbungsstruktur Veränderung der Kardinalitäten; Erzeugen einer Klasse, eines Attributes oder einer Assoziation; Löschen.
5. Explizit benannte (und gebundene/mit Werten vordefinierten) Variablen, z.B. um Konstanten hinzuzufügen.
6. (Funktionale) Sprache für Berechnungen/Umrechnungen, z.B. Einheiten umrechnen.
7. Einbettung (via Deklaration) von externen Funktionen.
8. Objektorientierte Vererbungs-'Mechanismus' (Transformationsregel A EXTENDS B).
9. Datenformat-spezifische Angaben (z.B. bei DXF)
10. Evtl. herstellerspezifische Erweiterbarkeit für (ähnlich via stereotypes, 'pragma'; z.B. die GUI-Statements von FME).

11. Handhabung rekursiver Strukturen mit beliebigen Schachtelungen; Umwandlung von Bäumen in Listen und umgekehrt (levels of nesting; vgl. z.B. XSLT/XML).
12. Bi-direktionale Abbildungen.

Aus dieser Sammlung entfallen unserer Meinung nach die letzten beiden Punkte.

## 2.2 Usability-Anforderungen

Entwurfsziele einer solchen Sprache ist Lesbarkeit und Ausdrucksstärke:

- Deklarativ: Ergibt einfachere semantische Modelle um die Transformationsregeln zu verstehen (im Gegensatz zu prozeduralen Sprachen damit ist bei deklarativen z.B. die Reihenfolge der Regeln und Abbruchkriterien irrelevant).
- Regeln gruppieren/modularisieren (Transformationrule-Topics/Packages?)
- Mehrfache Ziel-Elemente in einer einzigen Regel angeben (verbessert die Lesbarkeit aber nicht die Ausdrucksstärke, denn man könnte jederzeit zwei Regeln schreiben).
- Zwischen- und Hilfsregeln (Views) definieren, die mehrere Zwischenschritte erlauben.
- Natürliche explizite 'Einbettung' von Bedingungen, Berechnungs-Ausdrücken und (schachtelbare) Funktionen.
- Nähe zu Normen, wie INTERLIS und MOF für den einfacheren Einstieg ausgehend vom bekannten UML
- Potentielle Kompatibilität zu XSLT, XQuery/XPath, SQL (XQL), etc.

Ein Hauptentscheid ist die Bevorzugung eines deklarativen Stils durch seine hohe Abstraktionsebene und die Eigenschaft, dass dieser Stil unabhängiger von konkreten (letztlich prozeduralen) Konvertern macht. Dabei soll uns aber bewusst sein, dass mit einer deklarativen Sprache nicht alle Probleme lösen kann. D.h. dass auf irgendeine Art und Weise trotzdem auch prozedurale Sprachelemente vorhanden sein sollten.

# 3 Stand der Technik

## 3.1 Softwarewerkzeuge

Es gibt einige Werkzeuge, wie ICS, FME oder InterlisStudio. Diese haben aber einige Nachteile, bzw. Unzulänglichkeiten: Sie haben z.T. kein grafisches Benutzerinterface und deren Konfigurations-Sprachen/Skripts sind weder wirklich konzeptionell-deklarativ und einige sind binär oder gar nicht dokumentiert.

Ideal für die einfache Bedienung ist ein GUI-Dialog mittels dem man mit "Steckern" Klassen eines Quell- und einem Ziel-Schema einander zuordnen kann (ähnlich wie das BizTalk (auf Format-Ebene) die "FME Workbench" macht (oder noch besser...)).

### 3.2 Sprachen

In Gerber et al. [Gerber] "Transformation: The Missing Link of MDA" gibt es eine aktuelle Übersicht über Arten von Schema-Transformations-Sprachen und interessante Ausführungen zur Bedeutung für die Integration, ja für den eigentlichen Erfolg der "Model Driven Architecture" von OMG.

Hier eine Zusammenstellung ohne Anspruch auf Vollständigkeit:

- Prozedurale Programmiersprachen:
  - Java, C++, C, VisualBasic, etc.
- Prozedurale Skriptsprachen:
  - ICS-Skript; FME-Skript; Perl, Awk, Editoren
- Funktionale Sprachen:
  - Object Constraint Language (OCL) 2.0 [OCL].
- Deklarative Programmiersprachen:
  - EXPRESS-X (deklarativ): vormals EXPRESS-M; siehe Literatur.
  - ODL-M [ODL-M] Sehr einfache aber interessante, deklarative Sprache, die als Resultat einer Diplomarbeit jedoch leider experimentell geblieben ist (es gibt nur wenig Referenzen).
  - XSLT (deklarativ, aber maschinenorientiert): Ist z.Zt. einfach en vogue; ist aber sehr auf das (hierarchische) XML zugeschnitten.
- (Deklarative) Datenbank-Views: Man fasst das Mapping als Sicht der Zieldaten auf die Quelldaten auf.
  - SQL-View oder INTERLIS-ViewDef
- Graphisch-deklarative Sprachen:
  - Erweitertes UML (deklarativ, aber grafisch; bzw. XML, d.h. wäre damit wieder maschinenorientiert).
  - Graphen gemäss Graphen-Theorie.
- Reines Objektmodell: vgl. Fig. 5 unten.

Erste Recherchen ergaben, dass es etwa drei Hauptaktivitäten gibt: Eine eher akademische rund um ODL/OQL, eine rund um STEP (EXPRESS-X) und neu eine beim OMG unter dem Namen "MOF transformations". Leute rund um das W3C werden einerseits XSLT bevorzugen und evtl. von Ontologien sprechen (...).

MOF und CWM können noch nicht richtig eingeordnet werden (MOF-Query, not yet issued), has the potential to play a key role in the MDA vision. CWM-Transformation-Meta-Schema (deklarativ, aber grafisch; bzw. XML): "Common Warehouse Metamodel (CWM) Specification" (Kap. 10, eine Art "UML-Profil" von der OMG-Initiative Model-driven Architecture) regelt keine Transformationssprache.)

Eine Alternative (bzw. Rückfallposition im wahren Sinne des Wortes) zu diesen Normierungsbemühungen ist immer eine proprietär definierte und evtl. binär gespeicherte Sprache.

## 4 Diskussion im Hinblick auf einen Entwurf

### 4.1 Allgemeines

Bei Schema-Transformationen stellt sich auch die Frage der Korrektheit. Es gibt verschiedene Aspekte der Korrektheit. Zunächst wird angenommen, dass die Quell- und Zielschemas selber formal korrekt sind (was z.B. von XSLT alleine nicht verlangt wird). Das einfachste Kriterium ist die syntaktische und strukturelle Korrektheit der Transformation selber (Wohlgeformtheit und Validität); diese kann mit Compiler geprüft werden. Bei Transformationen stellt sich aber die Frage, ob das Resultat einer Transformation immer auch wohlgeformt und valid ist, auch wenn von einem wohlgeformten und validen Quellschema ausgegangen wurde? Noch komplizierter ist die Frage, ob das Zielschema semantisch korrekt angenommen, dass das Quellschema semantisch korrekt ist? In der Realität hat man es oft sogar mit unvollständigen und nicht korrekten Quellschemas zu tun.

### 4.2 Warum nicht XSLT, bzw. XML?

Eine der ersten Fragen zu dieser Aufgabenstellung wird sein, warum dafür nicht XSLT verwendet wird. In folgender Literatur sind dazu einige Argumente zusammengestellt (aus [MTRANS]):

"...It seems that XSLT can be used to express model transformation if models are represented by XML. Nevertheless, It appears difficult to use directly XSLT on a real system for some reasons:

- Writing an XSLT program is long and painful. For instance, at France Telecom R&D, we use XLST to transform a model into another and this program takes about 7000 lines and it is unreadable. One important problem with XSLT is its poor readability and the high cost of maintenance for associated programs
- Writing an XSLT program implies good skills in the MOF and XML specification, because when using XPATH in XSLT, we must take into account the deep structure of models that depends on meta-models which are themselves highly dependent on MOF and XML. Few people which are experts with MOF and XML and we would welcome a more straightforward way of expressing these transformations.
- Executing an XSLT program is not user-friendly for models transformation. There are no error messages that depend on the application domain. For example, if we want to transform a concept that does not exist, the processor XSLT does not inform the user.
- Finally it is interesting to have the meta-models in line when generating code. This will allow the compiler to generate more clever patterns of XSLT code."

### 4.3 Hauptaspekte einer möglichen Sprache

Elemente einer deklarativen Sprache:

1. Explizites deklarieren der Quell- und Ziel-Elemente,
2. Das 'matchen' von Mengen von Quell-Schema-Elementen,
3. Filtern des Sets,
4. Transformations-Ausdrücke,
5. Definieren einer Menge von Ziel-Schema-Elementen und deren Attribute.

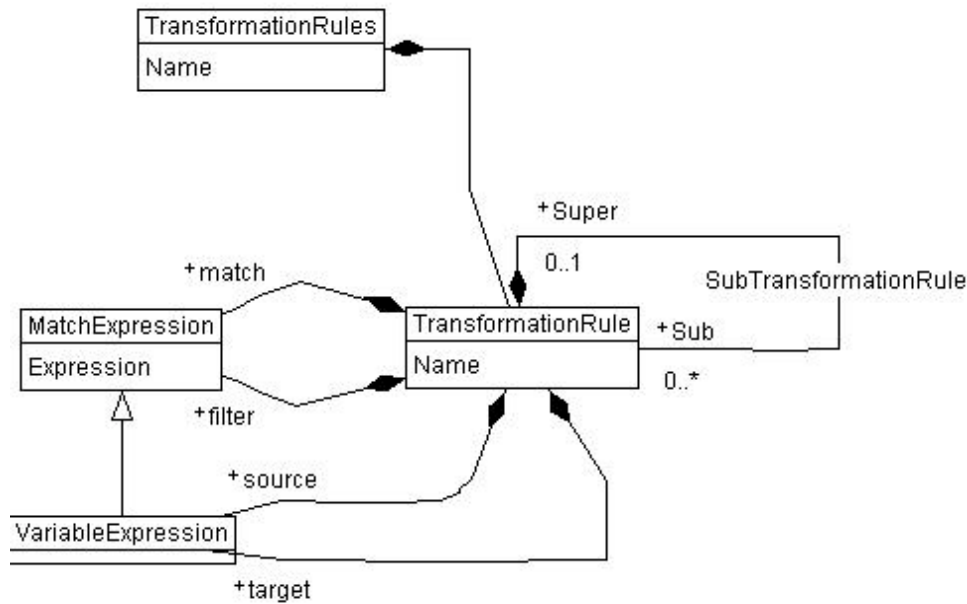


Fig. 5. Sprache statisch in Form eines Objektmodells (deren Elemente sind nicht auf menschliche Lesbarkeit ausgerichtet, sondern auf Maschinenlesbarkeit).

#### 4.4 Entwurf einer Sprache

Die experimentelle Sprache ODL-M [ODL-M] erfüllt einige Anforderungen; könnte aber noch mächtiger sein und wurde nach der Diplomarbeit nicht mehr weitergeführt.

Überlegungen zum Sprachentwurf:

- Deklarativ, angelehnt an GraphicDef und ViewDef von INTERLIS (INTERLIS-like) und mit eBNF-Regeln spezifiziert [ISO/IEC\_14977] (extended EBNF wegen der <Properties>-Regel).
- Die GraphicDef-Regel geht in die gewünschte Richtung (v.a. den Umgang mit Parametern); wir haben es aber nicht mit denselben Metaobjekten zu tun (Modell-Elemente und keine Grafiksignaturen). Als neue Syntax wird daher eine TransformationDef-EBNF-Regel eingeführt (**TRANSFORMATION**), die sehr **ähnlich der GraphicDef-EBNF-Regel** gehalten ist. Diese enthält eine oder mehrere TransformationRuleDefs (**Trans-Name 'OF' Trans-ClassRef ':' CondTransAssignment ';'** ).
- Die **TransformationRuleDef**-Syntax ist **analog DrawingRule** gehalten.
- Der Grund, dass mehrere TransformationRuleDefs (welche die eigentliche Abbildungsanweisung enthalten) in einer TransformationDef zusammengefasst werden, ist das Problem, wenn Objekte gemäss Quell-Klasse A (z.B. Adresse) zwei oder mehreren Klassen B (C, etc., z.B. Privatadresse, Geschäftsadresse) gemäss Zielschema erzeugt werden müssen (meist abhängig von einem Attributwert, z.B. Art==#privat, bzw. Art==#geschaeft): Das bedingt die Erzeugung zusätzlicher TIDs, die für die Assoziationen benötigt werden (vgl. das Beispiel Adressen unten).
- Inwiefern die ganze Expression-EBNF-Regel von INTERLIS 2 übernommen werden muss, muss noch geklärt werden.
- Durch die Benennung von TransformationDef und TransformationRuleDef wird auch Vererbung ermöglicht.
- Neu ist auch die **FORMAT**-Syntax: Damit wird Quell- oder Zielschema bestimmt, sowie das Format und allenfalls spezifische Formatangaben. Diese baut aber auf Metaobject und reiht sich daher in einen möglichen (künftigen) Metaobjekt-'Mechanismus' von INTERLIS 2 ein, der sich

langsam abzeichnet als gemeinsames Konstrukt von Referenzsystemen, Darstellungsbeschreibungen (inkl. Symbologie) und weiteren Modellierungs-Muster.

- Funktionen werden semantisch unverändert übernommen, syntaktisch auch - ausser dass neu **Funktionen als Parameter** (ArgumentType-Regel) zugelassen sind (Frage: Warum haben wir das nicht schon im aktuellen INTERLIS 2.2 zugelassen?).
- Views werden unverändert übernommen.

Wichtig für die Semantik der TransformationDef sind die Regelungen, wie sie in den ersten Abschnitten im Kapitel 2.16 des INTERLIS-Referenzhandbuch für GraphicDef-Regeln festgehalten wurden. Demnach gilt folgendes:

Die Transformationsregel wird mit einem Namen identifiziert, damit sie in Erweiterungen aufgegriffen und verfeinert werden kann. Existieren zu einer Transformationsregel Erweiterungen, erzeugen diese keine neuen (Ziel-)Objekte, sondern beeinflussen nur die Parameter der durch die Basisdefinition vorgegebenen Regel. Für jeden Parameter gilt derjenige Wert, der als letztes definiert wurde.

Offene Fragen:

- Geht's noch einfacher...?

**Fett** die neuen Sprachkonstrukte:

```
!! Folgende Klasse sei in INTERLIS zusätzlich vordefiniert für Transformation:
CLASS FORMAT (ABSTRACT) EXTENDS METAOBJECT = !! erbt Attribut Name
  PARAMETER
    Direction: MANDATORY (Source, Target); !! Quell- oder Zielschema
    Model: NAME; !! bezeichnet optional das dem Format zugeordnete Modell
  END FORMAT;

!! Datei INTERLIS1-Format.ili (= physikalisches Schema des INTERLIS 1-Formats):
MODEL INTERLISFormate =

  !! INTERLIS 2/XML-Format:
  CLASS IXML EXTENDS FORMAT =
    PARAMETER
      Name (EXTENDED): NAME := "IXML"; !! bezeichnet das Format
    END IXML;

  !! INTERLIS 1/ITF-Format. Diese Angaben sind eigentlich schon im
  !! INTERLIS 1-Schema enthalten; hier aber nochmals explizit aufgeführt:
  CLASS ITF EXTENDS FORMAT =
    PARAMETER
      Name (EXTENDED): NAME := "ITF"; !! bezeichnet das Format
      Format: MANDATORY (Free, Fix);
      Linesize: [1..9999]; !! bei Fix-Format
      Tidsize: [1..999]; !! bei Fix-Format
      MANDATORY CONSTRAINT (Format == #Fix) == DEFINED ( Linesize );
      MANDATORY CONSTRAINT (Format == #Fix) == DEFINED ( Tidsize );
    END ITF;

END INTERLISFormate.

!! Datei TransModell.ilt
!! *** Die eigentliche INTERLIS-Transformation ***
TRANSFORMATION MODEL TransModell (de_CH) =

  CONTRACT ISSUED BY HSR; !! Wegen Gebrauch von FUNCTION

  IMPORTS Modella; !! Modella in der Rolle als Quellschema
```

```

IMPORTS ModellB; !! hier als Zielschema
IMPORTS INTERLISFormate; !! Format für beide Modelle

!! Zwingende Angabe: muss pro Modell konkretisiert werden
!! und es muss in der Konfiguration mindestens FORMATE geben,
!! die je ein Attributwert #Source und #Target enthalten.
FORMAT INTERLISFormate.IXML =
  Direction := #Source;
  Model := { ModellA };
END INTERLISFormate.IXML;

FORMAT INTERLISFormate.ITF =
  Direction := #Target;
  Model := { ModellB };
  Format := #Fix;
  Linesize := 75;
  Tidsize := 10;
END INTERLISFormate.ITF;

FUNCTION substring( Source: TEXT, from: NUMBER, length: NUMBER): TEXT
  // returns part of the source string with position and
  length //;

STRUCTURE ListStructure =
  Container: LIST OF TEXT;
END ListStructure;

FUNCTION split( source: TEXT, delimiter: TEXT ): ListStructure;
  // Returns list element at index starting with 0 //;

FUNCTION index( arrayList: ListStructure): TEXT
  // Returns list element at index starting with 0 //;

FUNCTION getTelTyp( telefonnummer: TEXT*3 ): (Fix, Mobile)
  // Returns Mobile if String is between "079" .. "076"
  else Fix //;

TOPIC TransTopic = !! Gruppiert Regeln
DEPENDS ON ModellA, ModellB, INTERLISFormate;

VIEW PrivatansichtView
  PROJECTION OF Base ~ ModellA.Adressen.Adresse
  WHERE Art == #privat;
=
ATTRIBUTE
  ALL OF Base;
END PrivatansichtView;

VIEW TelefonnummernView
  INSPECTION OF Base ~ ModellA.Adressen.Adresse:Telefonnummer;
=
ATTRIBUTE
  ALL OF Base;
END TelefonnummernView;

!! TransformationDef:
!! Neue Syntax 'TRANSFORMATION' ähnlich GraphicDef.

TRANSFORMATION PrivatansichtRegel
  FROM PrivatansichtView, TelefonnummernView; !! (* ClassRef/ViewRef *)
  = !! TransformationRuleDef-Syntax analog DrawingRule
  Trans1 OF ModellB.Adressen.PrivatAdresse: (
    !! Vorname und Name abtrennen:
    Vorname := index( split( PrivatansichtView.VornameName, " "), 0 );
    Name := index( split( PrivatansichtView.VornameName, " "), 1 );
    PLZ := PrivatansichtView.PLZ;
  );

```

```

Trans2 OF ModellB.Adressen.Telefonnummer: (
  Landesvorwahl := "CH";  !! Konstante!
  Nummer := TelefonnummernView.Nummer;
  Typ := getTelTyp( substring( TelefonnummernView.Nummer, 0, 3 );
);
Trans3 OF ModellB.Adressen.TelZuAdressBez: (
  PrivatAdresseBez := Transl.OID;
  TelefonnummerBez := Trans2.OID;
), WHERE ( Transl.OID == Trans2.OID );
!! Hat Zugriff auf OIDs, da in derselben TransformationDef

END PrivatAdresseRegel;

END TransTopic;
END TransModell.

```

## 5 Szenarien und Use Cases

### 5.1 Szenarien und Use Cases

Siehe [Ulrich].

### 5.2 Beispiel Adressen

Beispiel mit folgenden Transformationen:

- Umbenennen von Klassennamen und Attributen
- Attributsplit aufgrund Inhalt (VornameName zu Vorname und Name)
- Klassensplit aufgrund eines Attributwerts (Adresse.Art zu Privatadresse)
- Klassensplit aufgrund des Typs (Transformation von LIST OF in zwei Klassen mit Komposition)
- Zusätzliche Konstante im Zielschema

```

INTERLIS 2.2;

MODEL ModellA =  !! Quellmodell
  TOPIC Adressen =

  STRUCTURE Telefonnummer =
    Nummer: TEXT*10;
  END Telefonnummer;

  CLASS Adresse =
    Art: (privat, geschaeftlich);
    VornameName: TEXT*40;
    PLZ: 1000..9999;
    Telefonnummer: LIST {0..*} OF Telefonnummer;
  END Adresse;
END Adressen;
END ModellA.

MODEL ModellB =  !! Zielmodell
  TOPIC Adressen =

  CLASS PrivatAdresse =
    Vorname: TEXT*20;
    Name: TEXT*20;
    PLZ: 1000..9999;
  END PrivatAdresse;

```

```
CLASS Telefonnummer =
  Landesvorwahl: TEXT*3;  !! z.B.
  Nummer: TEXT*10;
  Typ: (Fix, Mobile);
END Telefonnummer;

ASSOCIATION TelZuAdressBez =
  PrivatAdresseBez -<#> {0..1} PrivatAdresse;  !! immer {1}
  TelefonnummerBez -- {0..*} Telefonnummer;
END TelZuAdressBez;
END Adressen;
END ModellB.
```

## Anhang: Literatur

Siehe auch [www.integis.ch](http://www.integis.ch) > Publikationen.

[Akehurst] David H. Akehurst, Model Translation: A UML-based specification technique and active implementation approach, Doktorarbeit, 2001; [cs.ukc.ac.uk.pdf](http://cs.ukc.ac.uk.pdf).

[BizTalk] BizTalk: Microsoft, <http://www.biztalk.org/>

[CWM] CWM Partners. Common Warehouse Metamodel (CWM) Specification. OMG Documents: ad/01-02-f 01,02,03 g , Feb. 2001. <http://www.omg.org/>

[EXPRESS-X] EXPRESS-X. <http://www.nist.gov/sc4/step/parts/expressx/n002/>

[FME] FME: Feature Manipulation Engine der Safe Corp., Canada. <http://www.safe.com>

[Gerber]. Transformation: The Missing Link of MDA. Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood; CRC for Enterprise Distributed Systems (DSTC). <http://www.dstc.edu.au/Research/Projects/Pegamento/publications/icgt2002.pdf>

[igTools] igTools: INTERLIS Tools von InfoGrips <http://www.infogrips.ch/>

[INTERLIS1] Eidg. Vermessungsdirektion (1991): INTERLIS - Ein Datenaustauschmechanismus für Land-Informationssysteme.

[INTERLIS2] <http://www.interlis.ch>.

[ISO/IEC\_14977]. ISO/IEC 14977:1996; Information technology -- Syntactic metalanguage -- Extended BNF; ISO/IEC Copyright Office, Geneva.; 1996.

[MOF] Meta-Object Facility (MOF) specification version 1.3, OMG Document ad/00-04-03, March 2000.

[MOF-Query] MOF 2.0 Query/Views/Transformations, Request for Proposal: MOF 2.0 Core RFP. OMG Document: ad/01-11-05, Nov. 2001. Not yet issued.

[MTRANS] M. Peltier, J. B' ezivin, and G. Guillaume. MTRANS: A general framework, based on XSLT, for model transformations. In WTUML'01, Proceedings of the Workshop on Transformations in UML, Genova, Italy, Apr. 2001. ([wtuml\\_2001.pdf](http://wtuml_2001.pdf))

[OCL] Request for Proposal: UML 2.0 OCL RFP. OMG Document: ad/00-09-03, Sept. 2000.

[OCL-Tutorial] [http://neptune.irit.fr/Biblio/ocl\\_1\\_4.shtml](http://neptune.irit.fr/Biblio/ocl_1_4.shtml).

[ODL-M] ODL-M - A Mapping Language for Schema Integration in Object-Oriented Multidatabase Systems. Steinar A. Kindingstad, Master thesis submitted to the Cand. Scient degree in Informatics at the Department of Informatics, University of Oslo. August 1996.

[RoseTool] RationalRose Suite: RationalRose, <http://www.rational.com/rose/>

[SN612031] SN 612031 INTERLIS 2 - Ein Datenaustauschmechanismus. Referenzhandbuch. [www.snv.ch](http://www.snv.ch)

[Ulrich02] Bericht Werkzeuge zur Geodaten-Schema-Abbildung, P. Ulrich, 199b, Informatik-Seminar, Sommersemester 2002. [http://www.integis.ch/documents/ISem\\_Ulrich\\_Schema\\_Mapping\\_v1.4.pdf](http://www.integis.ch/documents/ISem_Ulrich_Schema_Mapping_v1.4.pdf)

[UML] UML 1.3, OMG, <http://www.omg.org/>

[UML-HUTN] OMG. Human-Usable Textual Notation. OMG Document: ad/02-03-02, Apr. 2002.  
<http://www.omg.org/>

[WAD] <http://www.uni-paderborn.de/cs/ag-schaefer/Lehre/PG/VARLET/documents/Wad98.pdf>

[XMI] OMG, eXtensible Metadata Interchange (XMI) specification version 1.0, Document ad/00-06-01, June 2000.

[XML] W3C, eXtensible Markup Language (XML) specification version 1.0 (second edition), October 2000.

[XSLT] <http://www.w3.org/TR/xslt>.

[TokTok] DSTC. TokTok – The Language Generator.  
<http://www.dstc.edu.au/Research/Projects/Pegamento/TokTok/>