INTERLIS 2 Relational

Einführung

Im folgenden Text ist ein Entwurf für ein INTERLIS 2 Relational Profil. Es wird eine Untermenge von INTERLIS 2 definiert, die einfach durch ein relationales GIS implementiert werden kann. Zur Vereinfachung der Implementierung wurden auch Elemente weggelassen, die nicht spezifisch für das objektorientierte Paradigma sind.

Syntaxregeln

Im Folgenden sind die Syntaxregeln gem. INTERLIS 2 Referenzhandbuch aufgeführt. Nicht mehr zulässige Syntaxelemente sind durchgestrichen; z.T. sind vereinfachte Alternativformulierungen eingefügt worden (unterstrichener Text).

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

Nur ein Modell pro ili-Datei.

```
ModelDef = [ 'CONTRACTED' ] [ 'TYPE' | 'REFSYSTEM' | 'SYMBOLOGY' ]
           'MODEL' Model-Name [ '(' Language-Name ')' ]
                 'AT' URI-String
                 'VERSION' ModelVersion-String [ Explanation ]
           [ 'TRANSLATION' 'OF' Model-Name '[' ModelVersion-String ']' ]
'='
               { 'IMPORTS' [ 'UNQUALIFIED' ] Model-Name
               { ',' [ 'UNQUALIFIED' ] Model-Name } ';' }
               { MetaDataBasketDef
               | UnitDef
               | FunctionDef
               | LineFormTypeDef
               | DomainDef
               | RunTimeParameterDef
               | ClassDef
               | StructureDef
               | TopicDef }
           'END' Model-Name '.'.
```

Kein IMPORTS, d.h. keine Basismodelle.

Um auf UnitDef verzichten zu können, muss man das Modell INTERLIS (Anhang A im Referenzhandbuch) um UnitDef für GRADS, DEGREES, u.a. ergänzen.

Strukturdefinitionen sind für Linienattribute notwendig. Strukturattribute sind aber nicht zulässig.

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

```
Definitions =  { MetaDataBasketDef
               | UnitDef
               | FunctionDef
               | DomainDef
               | ClassDef
               | StructureDef
               | AssociationDef
               | ConstraintsDef
               | ViewDef
               | GraphicDef }.

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

ClassDef = 'CLASS' Class-Name
             Properties<ABSTRACT,EXTENDED,FINAL>
               [ 'EXTENDS' ClassOrStructureRef ] '='
               [ ( 'OID' 'AS' OID-DomainRef | 'NO' 'OID' ) ';' ]
             ClassOrStructureDef
           'END' Class-Name ';'.

StructureDef = 'STRUCTURE' Structure-Name
                 Properties<ABSTRACT,EXTENDED,FINAL>
                   [ 'EXTENDS' StructureRef ] '='
                 ClassOrStructureDef
               'END' Structure-Name ';'.
```

Strukturdefinitionen werden zur Definition von Linienattributen benötigt.

```
ClassOrStructureDef = [ 'ATTRIBUTE' ] { AttributeDef }
                      { ConstraintDef }
                      [ 'PARAMETER' { ParameterDef } ].

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

StructureRef = [ Model-Name '.' [ Topic-Name '.' ] ] Structure-Name.

ClassOrStructureRef = ( ClassRef | StructureRef ).

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

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

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

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

RestrictedClassOrAssRef = ( ClassOrAssociationRef | 'ANYCLASS' )
                              [ 'RESTRICTION' '(' ClassOrAssociationRef
                                  { ';' ClassOrAssociationRef } ')' ].
```

```
ClassOrAssociationRef = ( ClassRef | AssociationRef ).

RestrictedStructureRef = ( StructureRef | 'ANYSTRUCTURE' )
                              [ 'RESTRICTION' '(' StructureRef
                                  { ';' StructureRef } ')' ].

RestrictedClassOrStructureRef = ( ClassOrStructureRef | 'ANYSTRUCTURE' )
                                   [ 'RESTRICTION' '(' ClassOrStructureRef
                                     { ';' ClassOrStructureRef } ')' ].

AssociationDef = 'ASSOCIATION' [ Association-Name ]
                     Properties<ABSTRACT,EXTENDED,FINAL,OID>
                     [ 'EXTENDS' AssociationRef ]
                     [ 'DERIVED' 'FROM' RenamedViewableRef ] '='
                     [ ( 'OID' 'AS' OID-DomainRef | 'NO' 'OID' ) ';' ]
                     { RoleDef RoleDef}
                     [ 'ATTRIBUTE'] { AttributeDef }
                     [ 'CARDINALITY' '=' Cardinality ';' ]
                     { ConstraintDef }
                 'END' [ Association-Name ] ';'.
```

Nur genau zwei RoleDef.

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

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

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

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

Type = ( BaseType | LineType ).

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

BaseType = ( TextType
           | EnumerationType
           | EnumTreeValueType
           | AlignmentType
           | BooleanType
           | NumericType
           | FormattedType
           | CoordinateType
           | OIDType
           | BlackboxType
           | ClassType
           | AttributeType ).

Constant = ( 'UNDEFINED'
           | NumericConst
           | TextConst
           | FormattedConst
           | EnumerationConst
           | ClassConst
```

```
            | AttributeConst ).

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

TextConst = String.

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

EnumTreeValueType = 'ALL' 'OF' Enumeration-DomainRef.

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

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

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

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

BooleanType = 'BOOLEAN'.

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

Verweis auf Einheiten sind möglich um DIM1, DIM2, DEGREES, GRADS Wertebereich abbilden zu können. Modell INTERLIS (Anhang A vom Referenzhandbuch) muss aber entsprechend erweitert warden.

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

CRS (LV03 oder LV95) nicht erkennbar/beschreibbar.

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

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

FormattedType = [ 'FORMAT' 'BASED' 'ON' StructureRef FormatDef ]
                [ Min-String '..' Max-String ]
          | 'FORMAT' FormattedType-DomainRef Min-String
                     '..' Max-String.
```

Datum- und Zeit-Wertebereiche via Verweis auf Definition in Modell INTERLIS (Anhang A Referenzhandbuch).

```
FormatDef = '(' [ 'INHERITANCE' ]
                [ NonNum-String ] { BaseAttrRef NonNum-String }
                                  BaseAttrRef [ NonNum-String ] ')'.


BaseAttrRef = ( NumericAttribute-Name [ '/' IntPos-PosNumber ]
              | StructureAttribute-Name '/' Formatted-DomainRef ).

FormattedConst = String.
```

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

Nur 2d oder 3d

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

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

BlackboxType = 'BLACKBOX' ( 'XML' | 'BINARY' ).

ClassType = ( 'CLASS'
                  [ 'RESTRICTION' '(' ViewableRef
                                      { ';' ViewableRef } ')' ]
            | 'STRUCTURE'
                  [ 'RESTRICTION' '(' ClassOrStructureRef
                                      { ';' ClassOrStructureRef } ')' ]
).

AttributeType = ( 'ATTRIBUTE'
                    [ 'OF' ( ClassType-AttributePath | '@' Argument-Name
) ]
                    [ 'RESTRICTION' '(' AttrTypeDef
                                        { ';' AttrTypeDef } ')' ] .

ClassConst = '>' ViewableRef.

AttributeConst = '>>' [ ViewableRef '.' ] Attribute-Name.

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

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

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

ControlPoints = 'VERTEX' CoordType-DomainRef.

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

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

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

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

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

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

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

```
MetaDataBasketDef = ( 'SIGN' | 'REFSYSTEM' ) 'BASKET' Basket-Name
                    Properties<FINAL>
                    [ 'EXTENDS' MetaDataBasketRef ]
                      '~' TopicRef
                    { 'OBJECTS' 'OF' Class-Name ':' MetaObject-Name
                      { ',' MetaObject-Name } } ';'.

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

MetaObjectRef = [ MetaDataBasketRef '.' ] Metaobject-Name.

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

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

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

Konsistenzbedingungen nur im Umfang von Interlis 1 IDENT.

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

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

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

UniquenessConstraint = 'UNIQUE' [ 'WHERE' Logical-Expression ':' ]
                       ( GlobalUniqueness | LocalUniqueness )
';'.

GlobalUniqueness = UniqueEl.

UniqueEl = ObjectOrAttributePath { ',' ObjectOrAttributePath }.

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

SetConstraint = 'SET' 'CONSTRAINT' [ 'WHERE' Logical-Expression ':' ]
                Logical-Expression ';'.

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

Expression = Term.
```

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

Term1 = Term2 { 'AND' Term2 }.

Term2 = Predicate [ Relation Predicate ].

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

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

Factor = ( ObjectOrAttributePath
         | (Inspection | 'INSPECTION' Inspection-ViewableRef) [ 'OF'
ObjectOrAttributePath ]
         | FunctionCall
         | 'PARAMETER' [ Model-Name '.' ] RunTimeParameter-Name
         | Constant ).

ObjectOrAttributePath = PathEl { '->' PathEl }.

AttributePath = ObjectOrAttributePath.

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

AssociationPath = [ '\' ] AssociationAccess-Name.

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

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

Argument = ( Expression
           | 'ALL' [ '(' RestrictedClassOrAssRef | ViewableRef ')' ] ).

FunctionDef = 'FUNCTION' Function-Name
                  '(' Argument-Name ':' ArgumentType
                  { ';' Argument-Name ':' ArgumentType } ')'
                  ':' ArgumentType [ Explanation ] ';'.

ArgumentType = ( AttrTypeDef
               | ( 'OBJECT' | 'OBJECTS' )
                           'OF' ( RestrictedClassOrAssRef | ViewRef )
               | 'ENUMVAL'
          | 'ENUMTREEVAL'  ).

ViewDef = 'VIEW' View-Name
            Properties<ABSTRACT,EXTENDED,FINAL,TRANSIENT>
            [ FormationDef | 'EXTENDS' ViewRef ]
```

```
{ BaseExtensionDef }
{ Selection }
            '='
            [ ViewAttributes ]
            { ConstraintDef }
          'END' View-Name ';'.

ViewRef = [ Model-Name '.' [ Topic-Name '.' ] ] View-Name.

FormationDef = ( Projection
               | Join
           | Union
           | Aggregation
               | Inspection ) ';'.

Projection = 'PROJECTION' 'OF' RenamedViewableRef.

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

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

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

Inspection = [ 'AREA' ] 'INSPECTION' 'OF' RenamedViewableRef
               '->' StructureOrLineAttribute-Name
                    { '->' StructureOrLineAttribute-Name }.

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

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

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

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

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

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

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

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

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

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

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

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