A Representation Model and a Mapping Language for INTERLIS

Table of Content

1. Introduction

1.1 Starting Point

The Swiss federal surveying authority (V+D) is responsible for the development and maintenance of the modeling language INTERLIS [1]. After successful completion of the incremental data exchange project, the V+D decided to implement a new INTERLIS Version 2 standard [2]. In the subsequent release process more than 15 requests for change (RFC) have been submitted by interested individuals and organizations. One of the requests, RFC-1011 proposes a representation model and a new mapping language for INTERLIS [3]. The intention of this paper is to transform RFC-1011 into a working standard.

1.2 Problems to solve

Several users requested the extension of INTERLIS Version 1 to meet visualization requirements of geographic data. In this paper the following points have to be investigated:

- Implementation of a representation model formulated in INTERLIS. The representation model describes object symbology (symbols, patterning, surface fills) in a system independent way.
- Definition of a new mapping language which enables the user to specify the correspondence between data model objects and representation model objects.
- Acquisition of Examples to clarify the new concepts.

The problem domain formulated above is still quite general, so we impose the following additional constraints:

- With the representation model we like to solve primarily the visualization needs of surveying and utility services mapping.
- The proposed visualization methods should be extensible by the user.
- We do not want to replace established graphics standard like PostScript, HP-GL etc. Therefore we have to show how to integrate existing standards in our concepts.
- In this work we consider only 2D graphics.

1.3 Benefits

The new INTERLIS extensions will result in the following benefits for the user:

- Symbology objects (symbols, patterns, hatching, etc.) have to be defined only once. The symbology objects can be shared by any system complying to the INTERLIS representation standard.
- In data transfers the symbology definitions are transferred together with the data model objects. No separate transmission of symbology libraries is necessary.
- The new mapping language allows the user to specify the correspondence between data model and representation model objects. The user can do this now in a system independent way. Up till now every system had to implement the correspondence in internal mapping tables.

1.4 Paper Organization

This paper is organized as follows:

- In chapter 2 we will give an overview of our solution.
- In chapter 3 we present a first example.
- In chapter 4 we describe the representation model formulated in INTERLIS.
- In chapter 5 we describe the mapping language in detail.
- In chapter 6 we describe the procedure necessary to convert INTERLIS graphics to concrete graphic standards like PostScript or HP-GL.
- In chapter 7 we give hints how complex graphic representation can be separated from base data models.
- The appendix contains a bibliography and the representation model in INTERLIS.

1.5 Related Work

Besides RFC-1011 this paper is related to RFC-1008 (object oriented INTERLIS). At the moment OO-INTERLIS is integrated in the new INTERLIS Version 2 standard. Our proposals use heavily the new features of OO-INTERLIS. We do not repeat or explain OO-INTERLIS in this paper. For details concerning OO-INTERLIS see the related papers [2] and [5].

2. Solution Overview

2.1. Graphics, what's that ?

Because a graphic representation is the visible part of an application (i.e. a plot or a computer screen display), users have normally a very clear understanding of the graphic capabilities of their systems. But still the question remains: *What is graphic in general*, i.e. independent of system implementation? If you analyze for example a plot, you see quickly that even the most complex graphic representations are composed by small graphic primitives. In the 2D case we differentiate the following primitives:

- **Text**. Text primitives have properties like size, font, color etc.
- **Line**. Line properties are width, dash style, patterning style, etc.
- **Surface**. Surfaces may be filled, outlined, hatched etc.
- **Symbol**. Symbols consist of a fixed text, line, surface combination.

Graphic primitives are of course also objects that can be modeled with INTERLIS (representation model).

2.2. The Representation Model

To describe graphic primitives and their properties we introduce a standard representation model described in INTERLIS (chapter 4). A representation model is always in some way depended of the application. In our case we have optimized the standard representation model for surveying and utility service applications, but through the possibilities of OO-INTERLIS a user can adapt or even replace the proposed standard representation model.

2.3. The Mapping Language

Objects of a graphic representation (graphic primitives) have a well known relation to data model objects. A graphic representation can therefore be seen as mapping of data model objects to representation model objects:

graphic representation = F(data model)

At the moment we have no possibilities to describe this relation. In chapter 5 we will hence introduce the new INTERLIS mapping language.

Remarks on the mapping function F():

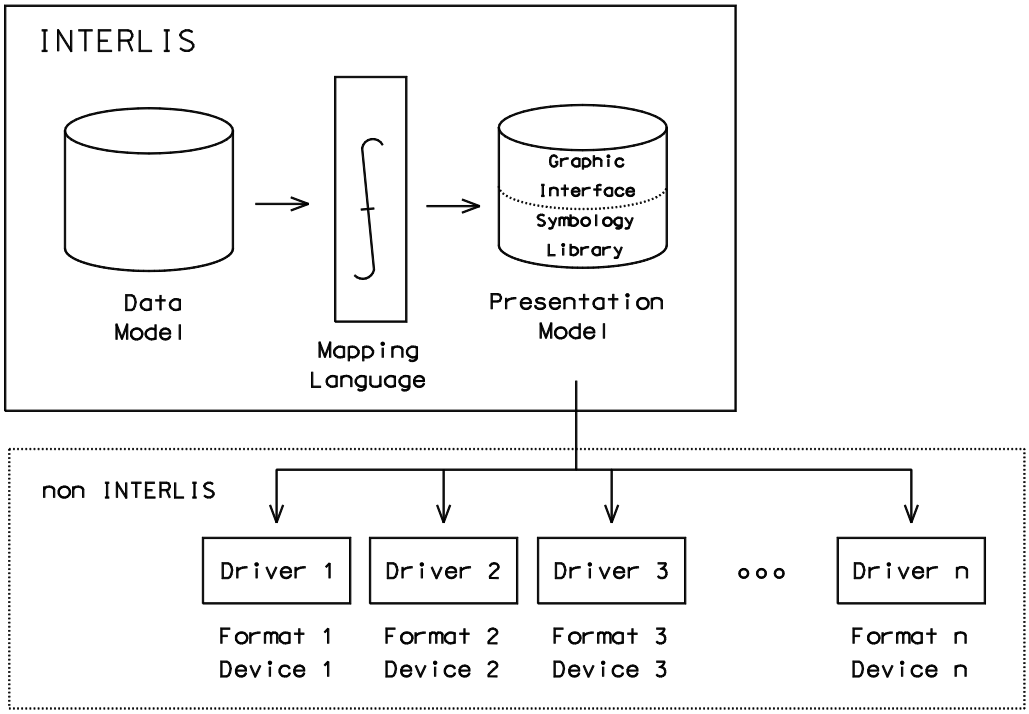
- The graphic primitives generated by F() exist only temporary. There's no need to store them in a database because they can always be generated by F() from data model objects.
- In some CAD systems there's nothing like a data model. It seems that there are only graphic primitives stored directly in the system. But in our view there is some internal fixed data model from which graphic objects are generated by a system depended mapping function F() at runtime.

2.4 From neutral to concrete graphics

We will describe graphic primitives in chapter 4 in a system independent way, but finally every graphic representation has to be created on some existing device. Here the established graphic standards come into play. We will describe the translation process from system neutral graphics to a concrete graphic format in chapter 6.

2.5. Summary

The next picture shows a summary of our solution.



Remarks:

- Through the mapping function F() we map data model objects to representation model objects. For the specification of F() we introduce the new INTERLIS mapping language.
- The representation model consists of a graphic interface and a symbology library. The representation model is defined in INTERLIS. Through the possibilities of OO-INTERLIS a user can extend or even replace the standard representation model.
- The graphic interface describes the graphic primitives text, line, surface and symbol. The graphic primitives exist only temporary.
- The symbology library contains text, line, surface and symbol definitions. The symbology library determines the rendering of a graphic primitive to an output device.
- Graphic primitives are rendered by a driver program to an output format or output device (see also chapter 6).

3. A first Example

3.1 Problem Description

Before we start to describe the representation model and the mapping language in chapter 4 and 5, we think it is time to give a simple but complete example. The problem we like to solve in our example is defined as follows:

- Triangulation points (`Fixpunkte`) have to be presented in scale 1:500 and 1:10'000.
- In scale 1:500 we like to display a text and a symbol for each triangulation point. Different symbols should be used for different kinds of triangulation points (i.e. `LFP1` or `LFP2`).
- In scale 1:10'000 we display the same symbol for all triangulation points, but we do not display a text.

The example presented in this chapter is intentionally very simple. The mapping functions declarations are integrated directly in the data model. In complex data models with diverse graphic representations it is desirable to separate mapping definitions from data model definitions. We will come back to this issue in chapter 7.

3.2 Solution

The solution in INTERLIS is presented next (additions to the standard model `Grunddatensatz` [4] appear in bold text):

```
MODEL Grunddatensatz EXTENSION OF ILI_REPRESENTATION

    DOMAIN
    !! Wertebereichsdefinitionen ...

    REPRESENTATION !! define possible representations
      M500;M10000;

    TOPIC Fixpunkte =
      TABLE LFP =
        Entstehung: OPTIONAL -> LFPNachfuehrung; !! Beziehung 1-mc
        Nummer: TEXT*12; !! Vergabe durch Landestopographie
        NumPos: LKoord;
        NumOri: OPTIONAL SchriftOri; !! Default: 100.0
        NumHAli: OPTIONAL HALIGNMENT; !! Default: Center
        NumVAli: OPTIONAL VALIGNMENT; !! Default: Half
        SymbolOri: OPTIONAL SchriftOri; !! Default: 0.0
        Geometrie: HKoord;
        !! etc.
        Art: (LFP1, LFP2);
        Herkunft: OPTIONAL TEXT*30; !! Vergabe durch Kanton
      CONSTRAINT
        UNIQUE Nummer;
        UNIQUE Geometrie;
      MAPPING
        M500: !! definition of F() for M500
          IF Art = "LFP1" THEN
            RSYMBOL(Geometrie,SymbolOri,"Fix500_LFP1");
          ELSIF Art = "LPF2" THEN
            RSYMBOL(Geometrie,SymbolOri,"Fix500_LFP2");
          END;
          RTEXT(Nummer,NumPos,NumOri,NumHAli,NumVAli,"Fix500_LFP");
```

```
          M10000: !! definition of F() for M10000
            RSYMBOL(Geometrie,SymbolOri,"Fix10000_LFP");
        END LFP;
      END Fixpunkte

    END Grunddatensatz.
```

Explanations:

- Because we like to define graphic representations for model `Grunddatensatz` we have to derive `Grunddatensatz` from the standard representation model `ILI_REPRESENTATION`.
- The graphic interface and the symbol library are inherited from `ILI_REPRESENTATION` (see also chapter 4 and appendix A2).
- All representations (`M500` and `M10000`) have to be declared first in the `REPRESENTATION` clause.
- For all objects of table `Fixpunkte.LFP` we define the mapping function F() in the `MAPPING` section of the table.
- For each representation (`M500` or `M10000`) we define F() by graphic interface function calls (`RSYMBOL`() and `RTEXT`()). The graphic interface is inherited from `ILI_REPRESENTATION`.
- In the `MAPPING` section it is possible to define more than one representation for a single table (`M500` and `M10000`).
- Symbology is assigned to graphic primitives by a unique name (`"FIX500_LFP1"`, `"FIX500_LFP2"`, `"FIX500_LFP"` and `"FIX10000_LFP"` in `RSYMBOL()` and `RTEXT()` function calls).
- The symbology library is inherited from the representation model `ILI_REPRESENTATION`.

4. The Representation Model

4.1 Introduction

In this chapter we describe the detailed structure of the representation model. Because all available graphic primitives are described by the representation model, all possible representations are determined by the structure of the representation model. The proposed standard INTERLIS representation model has the following general properties:

- The representation model is completely described in INTERLIS.
- The representation model consists of two main parts: the graphic interface and the symbology library.
- The graphic interface describes the graphic primitives text, line, surface and symbol. It is implemented by `INTERFACE` tables because it defines the interface between system independent INTERLIS graphics and the driver programs.
- The symbology library contains additional information necessary to render a graphic primitive to an output device. It contains for example line dash or surface fill definitions.

Remark: A representation model is always depended on it's concrete applications. The standard INTERLIS representation model is therefore optimized for the primary applications surveying and utility service mapping. However it is possible to adapt or even replace the standard representation model by OO-INTERLIS.

4.2. General Structure

The standard INTERLIS representation model has the following structure:

```
MODEL ILI_REPRESENTATION

   !! declarations

   !! graphic interface

   TOPIC SYMBOLOGY
      !! symbology definitions
   END SYMBOLOGY.

END ILI_REPRESENTATION.
```

Explanation:

- The declaration section contains definitions of graphic types and standard graphic functions (see also appendix A2 for details).
- The graphic interface is implemented by `INTERFACE` tables. The objects of the graphic interface (graphic primitives) are only virtual, i.e. they are not stored here.
- The symbology library is implemented in topic `SYMBOLOGY`. Topic `SYMBOLOGY` is automatically inherited by every model that extends `ILI_REPRESENTATION`.

4.3. The Graphic Interface

The graphic primitives defines the basic properties of the graphic primitives text, line, symbol and surface. It defines the INTERLIS interface to the driver programs that render INTERLIS graphic objects to graphic objects in a standard format (i.e. PostScript). Each graphic primitive has a reference to a symbology library

object. Together with it's symbology a graphic primitive can be rendered by a driver program on an output device. The INTERLIS definition of the graphic primitive `Text` is displayed next (see appendix A2 for all graphic interface definitions):

```
INTERFACE RTEXT =
  TXT       : TEXT; !! text
  GEOMETRY  : COORD2; !! position
  ROTATION  : DEGREES; !! rotation
  HALI      : HALIGNMENT; !! horizontal alignment
  VALI      : VALIGNMENT; !! vertical alignment
  SYMBOLOGY : RTEXTID; !! reference to symbology library
END RTEXT;
```

Explanations:

- The graphic interface contains only the basic properties (content, position, rotation, alignment) of the graphic primitive.
- Trough the property `SYMBOLOGY` the graphic primitive is connected to a symbology library object.
- The INTERFACE tables are only virtual, i.e. the graphic primitives are not stored here. They are sent directly to a driver program (see also chapter 6).

4.4. The Symbology Library

The symbology library supports the following symbology types:

- **Text Symbology** consisting of font, color, size, etc.
- **Symbol Symbology** consisting of text, line and surface primitives.
- **Line Symbology** with the possibility to define multiple lines, line dashes and patterned lines.
- **Surface Symbology** consisting of border symbology surface fills, surface hatching and surface patterning.

Each symbology has also the following general properties:

- **Clipping Range**. Through the clipping range a graphic primitive can be clipped against other graphic primitives.
- **Priority**. With priority the user can influence the ordering in which graphic primitives are send to an output device for final display.

The complete symbology library is specified in INTERLIS in appendix A2 (topic `SYMBOLOGY`).

4.4.1 Properties of Text Symbologies

A text symbology consist of a text font (`FONT`), a text height (`HEIGHT`) and clipping properties (`CLIPBOX`, `CLIPDIST`). The clipping properties have the following meaning:

- `CLIPBOX`: With `CLIPBOX` the user can specify a rectangular clipping region around a text primitive.
- `CLIPDIST`: With `CLIPDIST` we can define a clipping mask following each character of a text primitive.

4.4.2 Properties of Symbol Symbologies

A symbol symbology consists of text, line and surface geometries which are treated as a single entity. For each symbol we can define one or more clipping surfaces (`SYMBOL_CLIPSURF`). The symbol geometry

is defined relative to it's origin (`0.000/0.000`). The origin is used later as insertion point of the symbol geometry.

4.4.3 Properties of Line Symbologies

A line can be represented by a single or by multiple parallel line parts. Each line part has an perpendicular offset (`OFFSET`) from the line axis. For each line part a user can select one of the following line styles:

- `NORMAL`: Solid line. In `WIDTH` the user can specify the width of the line.
- `DASHED`: The line will be dashed. The line pattern is defined in the table `LINE_DASH`. The dash pattern is repeated after `DLENGTH` user units along the line.
- `PATTERNED`: Several symbols can be placed along a line. The symbol pattern is repeated after `DLENGTH` user units. All symbols are oriented according to the curvature of the line.

The user can also specify a symbol at the start (`SSYMBOL`) and at the end (`ESYMBOL`) of the line. For each line part the clip range can specified in `CLIPDIST`.

4.4.4 Properties of Surface Symbologies

A surface can be filled by a single color, hatched with lines or filled by a symbol pattern. For the border of the surface the user can select any line symbology. The fill styles have the following meaning:

- `FILL_COLOR`: The surface is solid filled with a color.
- `HATCH_SYMB`: The surface is filled with a hatch pattern. The distance between hatch lines is defined in `HATCH_OFFSET`. The hatching of a surface primitive can also be influenced by the `HATCH_ANG` and `HATCH_ORG` parameters in `RSURFACE()` function calls.
- `PATTERN_SYMB`: The surface is filled with a symbol pattern. The distance between symbols is defined in `PATTERN_DIST` and `PATTERN_OFFSET`. As with `HATCH_SYMB` the hatching can be influenced by the `RSURFACE()` parameters `PATTERN_ANG` and `PATTERN_ORG`.

Surfaces can have a clipping range. The clipping range can be defined to be `inside` or `outside` of the actual surface.

4.5. Using the Representation Model

The representation model can be used from any data model by inheriting it's definitions from standard representation model `ILI_REPRESENTATION(Model <Name> EXTENSION OF ILI_REPRESENTATION)`. By inheriting from `ILI_REPRESENTATION` all necessary definitions become available in the data model.

5. The Mapping Language

5.1 Introduction

With the new INTERLIS mapping language we can specify the mapping function F() in a formal way. The mapping language is responsible for the following tasks:

- Selection of data model objects (*What* is represented?).
- Mapping of selected data model objects to representation model objects (*How* is it represented?).

Because we can normally define more than one representation for a single data model, the mapping language allows us also to define the context of mapping function F().

5.2 INTERLIS Extensions

The INTERLIS mapping language is not implemented as a separate language, but integrated as an extension of (OO-)INTERLIS. The advantage of this approach is that all features of INTERLIS Version 2 can be used directly also in the new mapping language. The following INTERLIS extensions are introduced:

- At model level: The `REPRESENTATION` expression (context). With the `REPRESENTATION` expression we can declare all possible representations by a unique name. If necessary, a representation definition can contain parameters.
- At table level: The new `MAPPING` section (*What* and *How*). Together with the `IF` expression (*What*) and function calls (*How*) we can specify the mapping function F() for all INTERLIS tables.

5.2.1 The REPRESENTATION Expression

The `REPRESENTATION` expression has the following syntax (new syntax parts are represented in bold text):

```
Datenmodell = 'MODEL' Modell-Name '='
                 [ Modell-Wertebereichsdef ]
                 [ Darstellungsdef ]
                 [ Funktionsdef ]
                 (* Thema *)
              'END' Modell-Name '.'

Darstellungsdef = 'REPRESENTATION'
                     (* Darstellung ';' *)

Darstellung = Darstellungsname [ Parameterliste ]

Parameterliste = '(' Parameterdef ' (* ',' Parameterdef *) ')'

Parameterdef = Parametername ':' Wertebereich
```

The `REPRESENTATION` expression has the following semantics:

- The `REPRESENTATION` expression allows us to declare all possible representations for a given data model by a name.
- The mapping function F() can only be defined for declared representations.
- A driver program receives the representation name as a parameter. By the representation name a driver program knows which representation it has to generate from a data set.

- If a representation has been declared with parameters, the parameters can be used in the MAPPING section as variables. The variables can be used to fine tune the graphic output generated by a driver program.

5.2.2 The MAPPING Section

The MAPPING section has the following syntax (new syntax parts appear in bold text):

```
Tabelle = 'TABLE' Tabellen-Name '='
             Attribute
             Konsistenzdef
             Abbildungsregeln
          'END' Tabellen-Name ';'

View = 'VIEW' View-Name 'ON' Tabellenliste [ Erläuterung ] '='
          [ Attribute ]
          Abbildungsregeln
       'END' View-Name ';'

Abbildungsregeln =
   'MAPPING'
       (* Abbildungsregel *)

Abbildungsregel =
   Darstellungsname ':'
       (* Abbildungsanweisung *)

Abbildungsanweisung =
   (Funktionsaufruf | IF_Anweisung) ';'

IF_Anweisung =
   'IF' Logischer_Ausdruck 'THEN'
       (* Abbildungsanweisung *)
   (* 'ELSIF' Logischer_Ausdruck 'THEN'
       (* Abbildungsanweisung *) *)
   ['ELSE'
       (* Abbildungsanweisung *) ]
   'END'
```

The MAPPING section has the following semantics:

- In the MAPPING section we define the mapping function F() for a single table or view. The mapping function F() is only valid in a certain context. The context is specified by the representation name (Darstellungsname).
- The function calls in the MAPPING section create graphic primitives in the representation model. The call to RTEXT() for example creates a text primitive. All possible function calls and their parameters are defined by the graphic interface.
- To create a representation by a representation program all tables or views of the data model that have MAPPING sections for this representation are invoked and executed. The created graphic primitives are rendered by the driver program to concrete graphics (see also chapter 6).

6. From neutral to concrete Graphics

6.1 Introduction

We still have to answer how neutral INTERLIS graphics defined by the representation model are rendered to a concrete graphic format like PostScript, HP-GL etc. The necessary transformation steps are not defined by INTERLIS itself, and the rendering process is considered to be outside of the INTERLIS standard. Still we think it is worth to look in this chapter at the rendering task a driver program has to master.

6.2 The Driver Program

Through a driver program INTERLIS graphic primitives are rendered to a concrete graphic format. The rendering process is described by the following algorithm:

```
sort graphic primitives by priority.

FOR priority := 1 TO 100 DO

   FOR obj IN graphic primitives WHERE obj.PRIORITY=priority DO
      clip graphic primitive.
   END_FOR

   FOR obj IN graphics primitives WHERE obj.PRIORITY=priority DO
      apply symbology to graphics primitive.
   END_FOR

END_FOR

display graphic primitives on output device.
```

Each step is explained in more detail in the following paragraphs.

6.2.1 Sorting by Priority

If the driver program supports priorities all graphic primitives have to be sorted according to their priority. Graphic primitives with a low priority will be sent first to graphic buffer. Graphic primitives with a high priority will be processed later. The graphic buffer is a temporary workspace where graphic objects are stored before they are sent to the graphic device.

6.2.2 Clipping of Objects

First the driver program has to calculate the clipping ranges according to the definitions in the symbology library. Next the clipping ranges have to be applied in the graphic buffer.

6.2.3 Rendering of Objects

Symbology definitions are applied to each graphic primitive (i.e. patterning, hatching, symbol generation). The resulting graphic objects are stored in the graphic buffer.

6.2.4 Display on Output Device

Finally the whole graphic buffer is submitted to the output device (i.e. screen, vector file, raster file, etc.). Because the graphic buffer is build up in user coordinates (i.e. m), the objects in the graphic buffer have to be transformed to device coordinates for final output.

6.3 Types of graphic Drivers

It is possible that a driver program does not support all features of the INTERLIS representation model. We can differentiate the following categories of drivers:

- **type-1**. This driver type supports all features of the INTERLIS representation model including priorities and clipping.
- **type-2**. This driver type supports all features of the INTERLIS representation model but it does not support priorities or clipping.
- **type-3**. This driver type does not support the INTERLIS representation model.

All representation and transfer programs available today are of type 3. We expect first to appear type 2 and later type 3 drivers. All producers of driver programs have to specify which parts of the representation model are supported by their drivers.

7. Structuring complex Graphics

In complex data models with lots of different graphic representations it is often desirable to separate the graphic definitions completely from data model definitions. For the separation of data model and graphic definitions we do not introduce any new concepts. Instead we use the already existing facilities of OO-INTERLIS. To show this we extend our example from chapter 2 as follows:

- The data model `Grunddatensatz` has to be separated in a base data model and a graphic data model.
- All base data should be contained in the data model `Grunddatensatz_Kern`.
- Graphic data for scale 1:1500 and 1:10'000 should be implemented in the separate graphic model `Grunddatensatz_Graphik`.

The solution is explained in the following paragraphs.

7.1 Base Data Model

The base data model `Grunddatensatz_Kern` is shown below:

```
MODEL Grunddatensatz_Kern

    DOMAIN
    !! domain ranges

    TOPIC Fixpunkte =
      TABLE LFP =
        Entstehung: OPTIONAL -> LFPNachfuehrung; !! relation 1-mc
        Nummer: TEXT*12;
        Geometrie: HKoord;
        Art: (LFP1, LFP2);
        Herkunft: OPTIONAL TEXT*30;
      CONSTRAINT
        UNIQUE Nummer;
        UNIQUE Geometrie;
      END LFP;
    END Fixpunkte

  END Grunddatensatz_Kern.
```

Explanations:

- The base data model `Grunddatensatz_Kern` contains no graphic data anymore.
- All graphic attributes (for example `NumPos`, `NumOri`, `NumHAli`, `NumVAli`, `SymbolOri`) have been deleted.

7.2 Graphic Model

The graphic model `Grunddatensatz_Graphik` is shown next:

```
MODEL Grunddatensatz_Graphik

    EXTENSION OF ILI_REPRESENTATION
    DEPENDING ON Grunddatensatz_Kern

    REPRESENTATION !! possible representations
      M500,M10000;
```

```
    TOPIC Fixpunkte =
      TABLE LFP =
        Objekt: -> Grunddatensatz_Kern.Fixpunkte.LFP;
        NumPos: LKoord;
        NumOri: OPTIONAL SchriftOri; !! Default: 100.0
        NumHAli: OPTIONAL HALIGNMENT; !! Default: Center
        NumVAli: OPTIONAL VALIGNMENT; !! Default: Half
        SymbolOri: OPTIONAL SchriftOri; !! Default: 0.0
      MAPPING
        M500: !! representation M500
          IF Art = "LFP1" THEN
            RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix500_LFP1");
          ELSIF Art = "LPF2" THEN
            RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix500_LFP2");
          END;
          RTEXT(Objekt.Nummer,NumPos,NumOri,NumHAli,NumVAli,
          "Fix500_LFP");
        M10000: !! representation M10000
          RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix10000_LFP");
      END LFP;
    END Fixpunkte

  END Grunddatensatz_Graphik.
```

Explanations:

- The graphic model has been derived from `ILI_REPRESENTATION`.
- The graphic model contains only graphic data (`NumPos`, `NumOri`, `SymbolOri` etc.) and `MAPPING` sections.
- To reference objects from the base data model we have introduced a model reference with `DEPENDING ON`.

7.3 2nd Version

Sometimes it is not possible to change an existing data model (i.e. model Grunddatensatz). In this case the solution presented in 7.1 and 7.2 is not feasible, but we can help us as follows (changes are marked in bold text):

```
MODEL Grunddatensatz_Graphik2

    EXTENSION OF ILI_REPRESENTATION
    DEPENDING ON Grunddatensatz

    REPRESENTATION !! mögliche Darstellungsarten
      M500,M10000;

    TOPIC Fixpunkte =
      VIEW LFP ON Grunddatensatz.Fixpunkte.LFP =
      MAPPING !! Definition von F() für M500 und M10000
        M500:
          IF Art = "LFP1" THEN
            RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix500_LFP1");
          ELSIF Art = "LPF2" THEN
            RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix500_LFP2");
          END;
          RTEXT(Objekt.Nummer,NumPos,NumOri,NumHAli,NumVAli,
          "Fix500_LFP");
        M10000:
```

```
            RSYMBOL(Objekt.Geometrie,SymbolOri,"Fix10000_LFP");
        END LFP;
      END Fixpunkte

    END Grunddatensatz_Graphik2.
```

Explanations:

- The data model `Grunddatensatz` has not been changed, i.e. the graphic object attributes (`NumPos`, `NumOri`, `NumVAli`, `NumHAli`, `SymbolOri`) have not been separated from the base model.
- Instead of tables we use views in this version. The MAPPING section can also be defined for views.

Appendix

A1 Bibliography

[1] Eidg. Vermessungsdirektion. INTERLIS ein Daten-Austausch-Mechanismus für Land-Informations-Systeme, Oktober 1991

[2] Eidg. Vermessungsdirektion. INTERLIS ein Datenaustausch-Mechanismus für Land-Informationssysteme. Draft Version 2, Januar 1998

[3] Stefan Keller, Eidg. Vermessungsdirektion. RFC-1011: Ein Darstellungsmodell und eine Abbildungssprache für INTERLIS, August 1997

[4] Eidg. Justizdepartement. Datensatz der amtl. Vermessung, 1993

[5] Josef Dorfschmid, ADASYS AG. Expertise betreffend OO-INTERLIS -einer objektorientierten Erweiterung von INTERLIS, Januar 1998

A2 Representation Model in INTERLIS

```
!! INTERLIS representation model
!!
!! Version 0.11
!!
!! Stand 30.1.1998

MODEL ILI_REPRESENTATION

   DOMAIN

      RCOORD2   = COORD2 0.000 0.000 1000000.000 1000000.000;
      RLIN      = POLYLINE WITH (STRAIGHTS, ARCS) VERTEX RCOORD2;
      RSURF     = SURFACE WITH (STRAIGHTS, ARCS) VERTEX RCOORD2;
      RINT      = [-2000000000 .. 2000000000];
      RFLOAT    = [-2000000000.000 .. 2000000.000];
      RANGLE    = DEGREES 0.000 .. 360.000;
      RPRIORITY = [1 .. 100];
      RGB       = TEXT*6;

      RLINEID = TEXT*20;
      RTEXTID = TEXT*20;
      RSYMBID = TEXT*20;
      RSURFID = TEXT*20;

   !! Allgemeine Graphikfunktionen

   FUNCTION

      OFFSET2(p:COORD2,y:RFLOAT,x:RFLOAT):COORD2 =
         // Addiert den Offset y/x zum Punkt p //;
      OFFSET3(p:COORD3,y:RFLOAT,x:RFLOAT,z:RFLOAT):COORD3 =
         // Addiert den Offset y/x/z zum Punkt p //;
      SUBSTRING(t:TEXT;index1:RINT;index2:RINT):TEXT =
         // Liefert von t den Teilstring von index1 bis index2.
            Bemerkung: Der Index des 1. Buchstabens ist 1 //;
      FORMAT(format:TEXT, ...):TEXT =
         // Liefert einen gemaess format formatierten Text
            (analog zur C-Funktion printf()) //;
```

```
!! Graphikinterface

INTERFACE RTEXT =
   TXT       : TEXT;
   GEOMETRY  : COORD2;
   ROTATION  : DEGREES;
   HALI      : HALIGNMENT;
   VALI      : VALIGNMENT;
   SYMBOLOGY : RTEXTID;
END RTEXT;

INTERFACE RSYMBOL =
   GEOMETRY  : COORD2;
   ROTATION  : DEGREES;
   SYMBOLOGY : RSYMBID;
END RSYMBOL;

INTERFACE RLINE =
   GEOMETRY  : RLIN;
   SYMBOLOGY : RLINEID;
END RLINE;

INTERFACE RSURFACE =
   GEOMETRY    : RSURF;
   HATCH_ANG   : OPTIONAL DEGREES;
   HATCH_ORG   : OTIIONAL COORD2;
   PATTERN_ANG : OPTIONAL DEGREES;
   PATTERN_ORG : OPTIONAL COORD2;
   SYMBOLOGY   : RSURFID;
END RSURFACE;

!! Signaturbibliothek

TOPIC SYMBOLOGY =

   !! text symbology

   TABLE TEXT_SYMBOLOGY =
      NAME     : RTEXTID;
      PRIORITY : RPRIORITY;
      COLOR    : RGB;
      FONT     : TEXT*30;
      HEIGHT   : RFLOAT;
      CLIPBOX  : OPTIONAL RFLOAT;
      CLIPDIST : OPTIONAL RFLOAT;
   CONSTRAINT
      UNIQUE NAME;
   END TEXT_SYMBOLOGY;

   !! symbol symbology

   TABLE SYMBOL_SYMBOLOGY =
      NAME     : RSYMBID;
      PRIORITY : RPRIORITY;
   CONSTRAINT
      UNIQUE NAME;
   END SYMBOL_SYMBOLOGY;

   TABLE SYMBOL_TEXT =
      OBJECT    : -> SYMBOL_SYMBOLOGY;
      COLOR     : RGB;
      FONT      : RTEXT;
      HEIGHT    : RFLOAT;
      TXT       : RTEXT;
      GEOMETRY  : COORD2;
      ROTATION  : RANGLE;
      HALI      : HALIGNMENT;
      VALI      : VALIGNMENT;
   END SYMBOL_TEXT;

   TABLE SYMBOL_LINE =
      OBJECT   : -> SYMBOL_SYMBOLOGY;
      COLOR    : RGB;
      WIDTH    : RFLOAT;
      GEOMETRY : RLIN;
   END SYMBOL_LINE;

   TABLE SYMBOL_SURFACE =
      OBJECT    : -> SYMBOL_SYMBOLOGY;
      FILLCOLOR : RGB;
      GEOMETRY  : RSURF;
```

```
END SYMBOL_SURFACE;

TABLE SYMBOL_CLIPSURF =
   OBJECT   : -> SYMBOL_SYMBOLOGY;
   GEOMETRY : RSURF;
END SYMBOL_CLIPSURF;

!! line symbology

TABLE LINE_SYMBOLOGY =
   NAME     : RLINEID;
   PRIORITY : RPRIORITY;
   SSYMBOL  : OPTIONAL -> SYMBOL_SYMBOLOGY;
   ESYMBOL  : OPTIONAL -> SYMBOL_SYMBOLOGY;
CONSTRAINT
   UNIQUE NAME;
END LINE_SYMBOLOGY;

TABLE LINE_NORMAL =
   OBJECT   : -> LINE_SYMBOLOGY;
   OFFSET   : RFLOAT;
   COLOR    : RGB;
   WIDTH    : RFLOAT;
   CLIPDIST : OPTIONAL RFLOAT;
END LINE_NORMAL;

TABLE LINE_DASHED =
   OBJECT   : -> LINE_SYMBOLOGY;
   OFFSET   : RFLOAT;
   COLOR    : RGB;
   WIDTH    : RFLOAT;
   DLENGTH  : RFLOAT;
   CLIPDIST : OPTIONAL RFLOAT;
END LINE_DASHED;

TABLE LINE_DASH =
   OBJECT : LINE_DASHED;
   SDIST  : RFLOAT;
   EDIST  : RFLOAT;
   CLIPDIST : OPTIONAL RFLOAT;
CONSTRAINT
   SDIST <= EDIST;
END LINE_DASH;

TABLE LINE_PATTERN =
   OBJECT   : -> LINE_SYMBOLOGY;
   OFFSET   : RFLOAT;
   PLENGTH  : RFLOAT;
   CLIPDIST : OPTIONAL RFLOAT;
END LINE_PATTERN;

TABLE LINE_PATTERN_SYMBOL =
   OBJECT   : -> LINE_PATTERN;
   SYMBOL   : -> SYMBOL_SYMBOLOGY;
   DIST     : RFLOAT;
   OFFSET   : RFLOAT;
   ROTATION : RANGLE;
   SCALE    : RFLOAT;
END LINE_SYMBOLOGY_SYMBOL;

!! surface symbology

SURFACE_SYMBOLOGY =

   NAME           : SURFID;
   PRIORITY       : RPRIORITY;
   FILL_COLOR     : OPTIONAL RGB;
   BORDER         : OPTIONAL -> LINE_SYMBOLOGY;
   CLIP           : OPTIONAL (inside,outside);

   HATCH_SYMB     : OPTIONAL -> LINE_SYMBOLOGY;
   HATCH_OFFSET   : OPTIONAL RFLOAT;

   PATTERN_SYMB   : OPTIONAL -> SYMBOL_SYMBOLOGY;
   PATTERN_DIST   : OPTIONAL RFLOAT;
   PATTERN_OFFSET : OPTIONAL RFLOAT;

CONSTRAINT
   UNIQUE NAME;
END SURFACE_SYMBOLOGY;
```

```
   END SYMBOLOGY.
END ILI_REPRESENTATION.
```