Bundesamt für Raumplanung
Office fédéral de l'aménagement du territoire
Ufficio federale della pianificazione del territorio
Uffizi federal da planisaziun dal territori

Federal Directorate of Cadastral Surveying
Eidgenössische Vermessungsdirektion
Direction fédérale des mensurations cadastrales
Direzione federale delle misurazioni catastali
Direcziun federala da mesiraziun

Eidg. Justiz- und Polizeidepartement
Département fédéral de justice et police
Dipartimento federale di giustizia e polizia
Departament federal da giustia e polizia

Center of Competence INTERLIS
Kompetenzzentrum INTERLIS/AVS
Centre de compétence INTERLIS/IMO
Centro di competenza INTERLIS/IMU
Center da cumpetenza INTERLIS/IMU

INTERLIS

A Data Exchange Mechanism

for Land-Information-Systems

Version 1 Revision 1a, November 1997

Contents

List of figures

1. Overview

The fundamental idea of INTERLIS is that information exchange for a geographic information system is only possible when the participating parts have an exact and uniform idea of the type of the exchange geodata. INTERLIS is therefore mainly concerned with the exact description of geodata models, and subsequently with the arrangement of the exchange format.

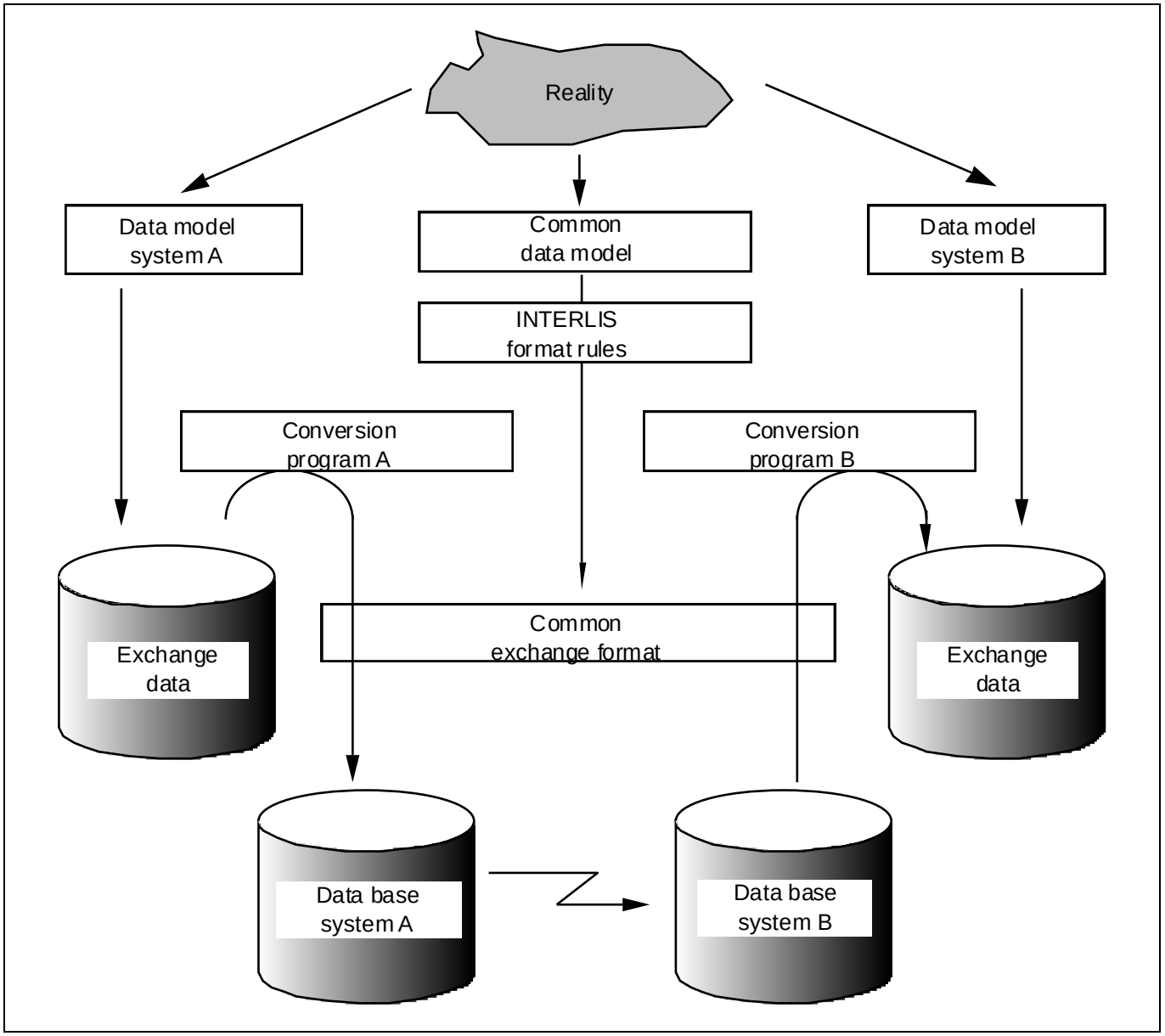

Figure 1: Data exchange using a common geodata model and a common data description language in between different geodata bases

No specific application is employed for the description of the data model. The model can be defined for any specific application much more effectively with a description language (cf. ch. 2). INTERLIS is therefore available for the entire range of applications which can be described with the proposed basic elements.

Concerning the description principle, INTERLIS follows the relationship data model, but it expands this through elements which are typical of geographic information systems. The language syntax follows that of modern programming languages such as PASCAL, MODULA2.

However, in order to make specific data exchange possile, a transfer protocol (exchange format) must be arranged beyond the data model. In order that the flexibility of the model description is not lost, the transfer protocol is formulated in such a way that it adapts itself to the specific data definition. The data exchange is currently defined at the stage of text data (cf. ch. 3).

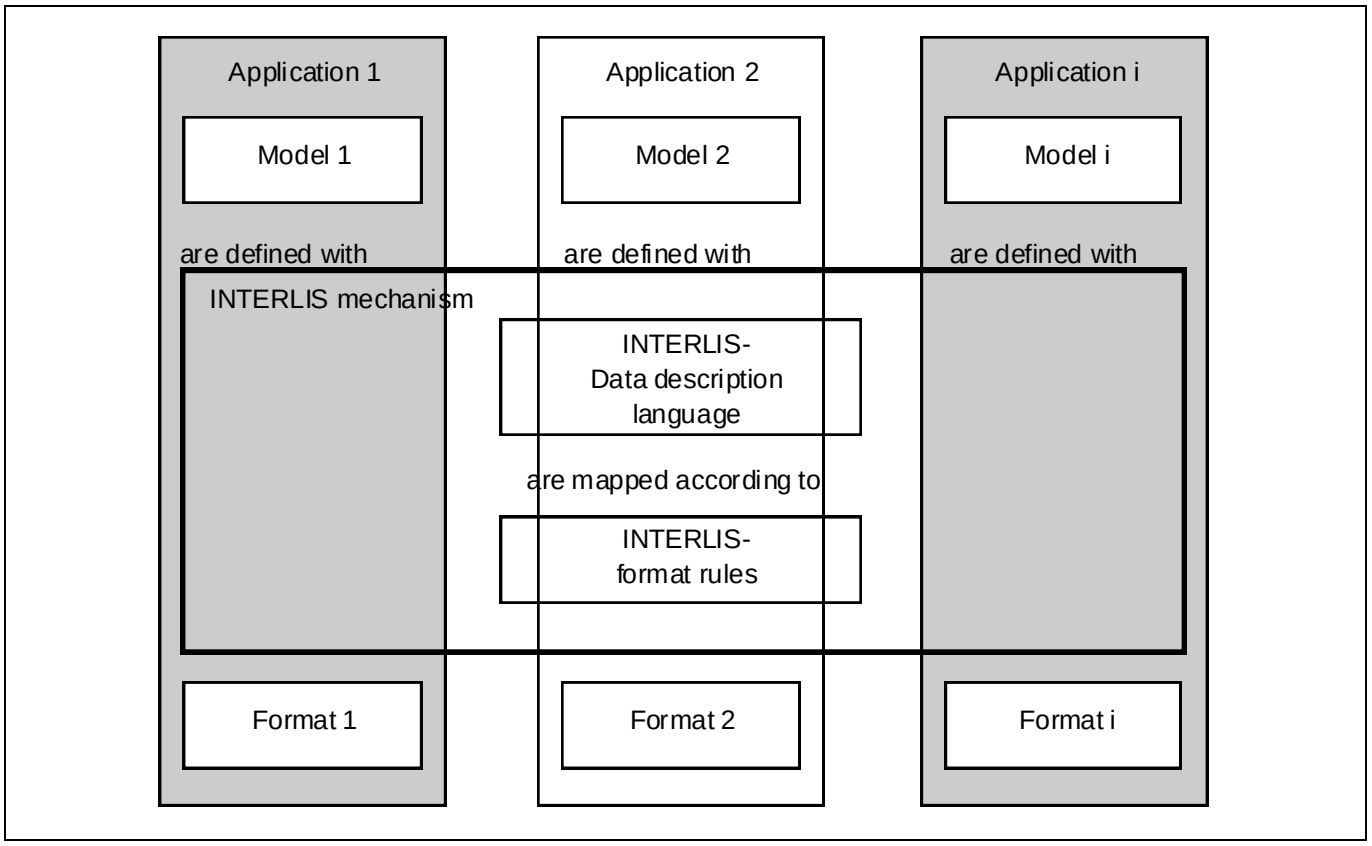

Figure 2: INTERLIS data description language and INTERLIS format rules

In order to verify that a model - which has been described in INTERLIS - is formally correct a compiler is available. Additionally it produces a description of the transfer format which corresponds to the model (cf. ch. 4).

Chapter 5 contains a simple example. In chapter 6 some notes to important questions are composed.

2. Description language

2.1. Syntax used

A formal language will be defined in the following chapters for creating a data model and the transfer parameters for data transfer. This language is itself formally defined. The syntax rules describe the admissible order of symbols.

The description follows the type and method which is usual for the description of modern programming languages. Only a succinct representation necessary for understanding is therefore given here. The literature should be consulted for particulars. A short introduction, for example, can be found in “Programming in Modula-2” by Niklaus Wirth.

A formula in the sense of the expanded Backus-Naur-Notation (EBNF) is built up as follows:

```
formula-name = formula-expression.
```

The formula-expression is a combination of:

- fixed words (incl. special characters) of the language. They are enclosed in apostrophes, eg. ‘BEGIN’
- references of other formulas by giving the formula name.

Valid combinations are as follows:

Stringing together

```
a b c              first a, then b, then c.
```

Obligatory choice

```
(a | b | c)        a or b or c
```

Optional choice

```
[a | b | c]        a, b or c or nothing
```

Optional repetition

```
{a | b | c}        Any order of a, b or c
```

It is also admissible that neither a nor b nor c occur. Eg: aaaaaa, abbc, accccab

Obligatory repetition (as addition to EBNF)

```
(* a | b | c *)
```

Any order of a, b or c. At least one occurrence is required. Everything is therefore admissible. (* a *) is therefore the same as a { a , but facilitates an easier notation method.

One often wants to use a syntactically equal formula in different contexts for different aims. In order to express this aim one has to write an additional formula:

```
Example = ‘TABLE’ tableName ‘=’ tableDef.
tableName = name.
```

In order that this indirectness can be voided, the following abbreviated notation method is used:

```
Example = ‘TABLE’ table-name = tableDef.
```

The formula table-name is not defined. Syntactically the name rule is directly applied. It is important that the name is nevertheless a table name. “Table” thereby becomes virtually a commentary.

2.2. Definition of the description language

2.2.1. Basic language symbols

The description language shows the following symbol classes: names, numbers, descriptions, special characters and reserved words, commentaries.

2.2.1.1. Names

```
name = letter { letter | numeral | ' ' }.
letter = ('A' | .. | 'Z' | 'a' | .. | 'z' ).
numeral = ( 0  | '1' | .. | '9').
```

A name is defined as a list of letters, numerals and underlines, of which the first character must be a letter. Umlauts and special characters are not allowed in names!

2.2.1.2. Numbers

Numbers occur in connection with the definition of values and with the establishing of the code of a specific character. Secondly, it is also helpful if the hexadecimal representation is available for that.

```
posNumber = (* numeral *).
number = ['+'|'-'] posNumber.
dec = number ['.'posNumber] [scaling].
scaling = 'S'number .
code = ( posNumber | hexNumber ) .
hexNumeral = ( numeral |'A'|'B'|'C'|'D'|'E'|'F'|'a'|'b'|'c'|'d'|'e'|'f').
hexNumber = 'O' ('x'|'X') (* hexNumeral *).
```

Examples:

```
posNumber:  5134523         1           23
number:         123      -435        +5769
dec:            123.456   123.456S4    123.456S-2
code:          1234       OXA2
```

Scaling on `dec` numbers is used to keep numbers in a sensible range, and to suppress superfluous noughts. 123.45654 means, eg. 123.456 * -10**4 in other words 1234560.

2.2.1.3. Explanations

Explanations are required when a fact has to be described more specifically. From the point of view of the standard mechanism the explanation is not further interpreted, ie. taken as comment. However, it is possible to formalise explanations in a more detailed way, and thereby facilitate automatic processing. For the start and close of an explanation // are therefore chosen. Two neighbouring diagonal strokes must never occur within the explanation.

```
explanation = '//' any_except_// '//'.
```

2.2.1.4. Special characters and reserved words

Following the syntax rules of the language special characters and reserved words are always written between apostrophes, eg. ';' or 'MODEL' . - In principal, the reserved words are written in

capital letters. If capital letters are exclusively used for reserved words and not for the names, then conflicts can be avoided.

The following reserved words are used:

```
ANY          ARCS         AREA         BASE         BLANK
CODE         CONTINUE     CONTOUR      COORD2       COORD3
DATE         DEFAULT      DEGREES      DERIVATIVES  DIM1
DIM2         DOMAIN       END          FIX          FONT
FORMAT       FREE         GRADS         HALIGNMENT   I16
I32          IDENT        LINEATTR     LINESIZE     MODEL
NO           OPTIONAL     OVERLAPS     PERIPHERY    POLYLINE
RADIANS      STRAIGHTS    SURFACE      TABLE        TEXT
TID          TIDSIZE      TOPIC         TRANSFER     UNDEFINED
VALIGNMENT   VERTEX       VERTEXINFO   VIEW         WITH
WITHOUT
```

Table 1: Reserved words.

2.2.1.5. Comment

```
!! up to line end
```

Two adjoining exclamation marks open a comment. The comment is completed by the end of the line.

2.2.1.6. Division of individual symbols

Adjoining symbols are normally divided by an intervening space (one or more empty characters, tabulators or line ends). However, this is only really necessary if a single symbol were to arise from the lacking of an intervening space.

2.2.2. A small example to get you started

A straightforward description of ground cover type will provide an example (the official survey of the Swiss, ‘AV93’, uses the name ‘surface cover’). In chapter 5 the same example will be explained more closely. The corresponding data exchange for that and a concrete data item are given there.

```
TRANSFER Example;

MODEL Example

  DOMAIN
    LCoord = COORD2  480000.00   60000.00
                     850000.00  320000.00;

  TOPIC groundCover =

    TABLE groundSurface =
      type : (building, fixed, humusrich, water,
              stocked, vegetationless);
      form : AREA WITH (STRAIGHTS, ARCS) VERTEX LCoord
             WITHOUT OVERLAPS > 0.10;
    NO IDENT
      !! search via geometry or building
    END groundSurface;
```

```
    TABLE building =
      assuranceNo : TEXT*6;
      surface : -> groundSurface // type = building //;
    IDENT
      assuranceNo; !! assumption that assuranceNo is unambiguous
      surface;     !! building are allocated an exact surface
    END building;

  END groundCover.
END Example.

FORMAT FREE;

CODE
  BLANK = DEFAULT, UNDEFINED = DEFAULT, CONTINUE = DEFAULT;
  TID = ANY;
END.
```

2.2.3. Main structure of the language

2.2.3.1. The main parts

```
transferDef = 'TRANSFER' transfer-Name';'
              [global-valueRangeDef]
              dataModel [evaluations]
              {arrangement} format coding.
```

A transfer definition is comprised by the value range definitions, which are valid for all the definitions (optional), the description of the data model and the giving of transfer parameters (format, coding). With the data model the type and structure of the data to be exchanged are precisely described. With the transfer parameters additional information is added for the transfer.

2.2.3.2. Value range definition

```
valueRangeDef = 'DOMAIN'
                  (* valueRange-name '=' attributeType ';' *).
```

With a value range definition a specific data type with all its characteristics is established. It can later be used for the definition of the attribute characteristics. The value range definitions are available for the attribute definitions as follows:

- in the whole definition, if the value range definition takes place in the context of the transferDef. The name remains reserved in the whole definition.
- within the model or an evaluation. The name is then reserved in this context.
- within a theme. The name is then only reserved up to the closure of the theme.

2.2.3.3. The data model

```
dataModel = 'MODEL' model-name
            [ model-valueRangeDef ]
            (* theme *)
            'END' model-name '.'.
```

The data model is described by means of

- a model name
- value range definitions, which are applicable to the whole model
- the definitions of the various themes.

2.2.3.4. The theme

```
theme = 'TOPIC' theme-name '='
          (* table | local-valueRangeDef *)
        'END' theme-name '.'.
```

A theme is primarily a collection of tables. From the point of view of data transfer the themes are completely independent of each other. A system receiving data must also be in a position to utilise the data even when all the themes have not been provided. With the capture of an object from a specific theme no assumptions can be made about the existence of objects from other themes.

This data division ensures that the transfer of an object is independant and will not transfer an entire set of further objects. Transfer of partial geodata bases is thereby possible. However, no comments are made here concerning the measures to be taken on the recipient system for containing possible inconsistencies between the themes and for clearing them up after the transfer is complete.

2.2.3.5. The table

```
table = [ 'OPTIONAL' ] 'TABLE' table-name '='
                         attribute identifications
                       'END' table-name ';'.
identifications = ( 'NO' 'IDENT' | 'IDENT' (* identDef *) ).
identDef = attribute-name { ',' attribute-name } ';'.
```

A table is a collection of data objects with the same characteristics. These are described by means of the attributes and the identifications. The attributes describe the exact structure of the data of the objects. The information of the identifications determines which attributes or attribute combinations definitively identify an object.

Example:

```
TABLE object =
  a: TEXT*8;
  b: TEXT*8;
  c: TEXT*8;
IDENT
  a;
  b,c;
END object;
```

With the first line after `IDENT` it is indicated that attribute a the object identifies, with the second line the combination of b and c is identifying (b or c alone need not be unique).

'OPTIONAL' shows that the table does not specifically have to be available.

2.2.3.6. The attribute

```
attributes = (* attribute-name ':' ['OPTIONAL']
                (localAttribute | relAttribute )
                [consistency-requirement] ';' *).
```

```
consistency-requirement = explanation.
```

Every attribute is described by its name, its structure and, if needed, information about special consistency requirements. Furthermore, 'OPTIONAL' can specify that it is admissible not to define the attribute. Optionally, an attribute definition can be closed by consistency instructions. However, these are not formalised, but given as an explanation. From the structure the attributes can then be divided into two groups.

Local attributes

A local attribute only has significance within the table. For the description of the characteristics of an attribute, various attribute types are available. They can roughly be divided into basic types for the various kinds of geometry. The exact specifications are given in the following chapters.

```
localAttribute = valueRange.
valueRange = (valueRange-name | attributeType).
attributeType = (basicType | lineType | surfaceAreaType).
```

Relationship attributes

Relations between objects are created by means of the relationship attributes. One should not worry about the way in which the relationships are created. One only has to show to which table, and therefore to which objects the relation applies. However, in order to retain the independence of the planes, relationship attributes are only permitted for relations within the same theme.

```
relAttribute = '->' table-name.
```

2.2.4. Basic types

```
basicType = ( coord2
            | coord3
            | length
            | area_measurement
            | angle
            | range
            | text
            | date
            | enumeration
            | horizAlignment
            | vertAlignment).
```

Coordinates, lengths and units of area measurements are all built in the unit of length. In order to retain clarity over the meaning of the transferred data, a uniform base unit (normally the meter) must be established for these types. All the entries for lengths, areas measurements and coordinates must then take place in this unit.

Giving minimum and maximum values with the various types, this defines the range as well as the accuracy. Decimal points and scaling must correnspond between the minimum and maximum values. If, for example, a length type has to be defined with an upper limit of one kilometer, and an accuracy of one millimeter, this is shown as:

```
DIMl 0.000  1000.000
```

2.2.4.1. Coordinates

```
coord2 = 'COORD2' Emin_dec Nmin-dec Emax_dec Nmax_dec.
coord3 = 'COORD3' Emin_dec Nmin_dec Hmin_dec Emax_dec Nmax_dec Hmax_dec.
```

The permitted value range for every component is established with coordinates. Without any indications (as commentary) the first component is the easting, the second component is the northing value, and the third is the height component.

2.2.4.2. Length and area measurement

```
length = 'DIMl' min_dec max_dec.
area_measurement = 'DIM2' min_dec max_dec.
```

2.2.4.3. Angle

```
angle = ( 'RADIANS' | 'GRADS' | 'DEGREES' ) min_dec max_dec.
```

Without further entries (as commentary) it is implied that GRADS and DEGREES are defined in the survey technical coordinate system, and RADIANS in the mathematical system.

2.2.4.4. Range

```
range = '[' min_dec '..' max_dec ']'.
```

A number range is defined without the exact meaning being established by the type. It will result either from the existing attribute name or from an additional commentary.

2.2.4.5. Text

```
text = 'TEXT' '*' max_length-posNumber.
```

Text means any string of characters, which must not exceed the given maximum character number. To avoid problems with the data exchange the rules and requirements according to ch. 3.4.5.5. must be adhered to.

2.2.4.6. Date

```
date = 'DATE'.
```

In order to enable a uniform handling of the date, it is not entered simply as free text. The date identifies the day. The date field therefore consists of year, month and day.

2.2.4.7. Enumeration

```
enumeration = '(' element { ',' element } ')'.
element = element-name [sub-enumeration].
```

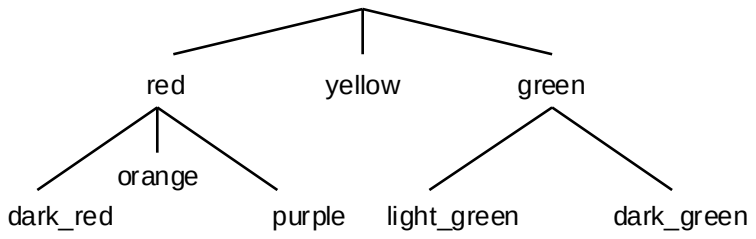

Figure 3: Example of an enumeration

An enumeration establishes the admissible values for this type. However, the list is not a simple linear one but structured in the form of a tree. For example:

```
(red (dark_red, carmine, orange), yellow, green (bright_green, dark_green))
```

gives the following admissible values:

```
red-dark_red red-carmine red-orange yellow green-bright_green green-dark_green
```

The structuring is shown by round brackets, the depth of which is freely definable.

2.2.4.8. Text alignment

```
horizAlignment = 'HALIGNMENT'.
vertAlignment = 'VALIGNMENT'.
```

The text positions must be kept in place for the preparation of plans. Furthermore, it has to be established where the text lies in relation to the coordinate. The horizontal alignment establishes whether the position lies on the left, right or in the middle of the text. The vertical alignment establishes the various height relationships.

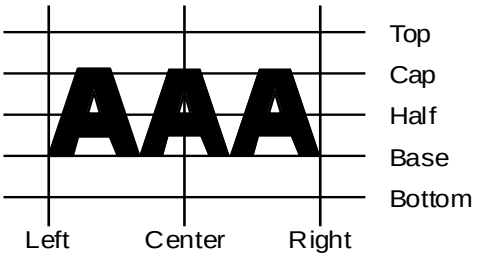

Figure 4: Text alignment horizontally and vertically

The distance cap-base corresponds to the height of the capital letters. Descenders occur in the area base-bottom.

Horizontal and vertical alignment can be understood as pre-defined enumerations. The effect is the same as if the following have been defined in the global values definition:

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

2.2.5. Line type

A line means an uninterrupted list of line segments, a single line string. More precise descriptions are required about the form of the line string (which types of geometry connections are admissible; can line strings intersect?) and information about the characteristics of the control points.

```
lineType = 'POLYLINE' form controlPoints [intersectionDef].
form = 'WITH' '(' formType { ',' formType } ')'.
formType = ('STRAIGHTS' | 'ARCS' | explanation ).
```

Figure 5: Some lines

Combinations of straight lines (STRAIGHTS), circular arcs (ARC), and special forms can thus be explained as admissible connection geometries.

The circular arcs are represented on the transfer file by means of an intermediate point in the middle of the circular arc. Circular arcs with very large central angles (more or less complete circles) thereby become numerically unstable. They must therefore be avoided. An intermediate point does not have the same meaning as a control point. He serves only the definition of the form of the circular arc. For the same circular arc different intermediate points can be used. But they all need to be nearby the middle of the circular arc. With this it is guaranteed that the systems involved in the transfer are free how they represent a circular arc internally.

More complicated connection geometries are not fixed by means of INTERLIS. Although the possibility for description and transfer nevertheless exists, special cases can be given as explanations (for instance for interpolation curves). The type of description and the coding on the transfer file is thereby the responsibility of those involved with the transfer.

```
intersectionDef = 'WITHOUT' 'OVERLAPS' '>' dec.
```

It can be determined that a line is not allowed to intersect neither with itself nor with a line of the same attribute of another object. Intersections which lie within the given tolerance are allowed in the context of circular arcs, if the circular arc intersects a line with the fact that there exists no control point on either side between the start and end point of the circular arc. This rule is needed because of two reasons: On one hand with circular arcs small intersections are not avoidable in special cases (tangential circular arcs). On the other hand larger overlaps (e.g. some centimeters) need to be tolerated during an import from data which was originally captured graphically. Otherwise this data would need a enormous amount of postprocessing resources. The indication of the tolerance need to be defined in the same units as the control coordinates.

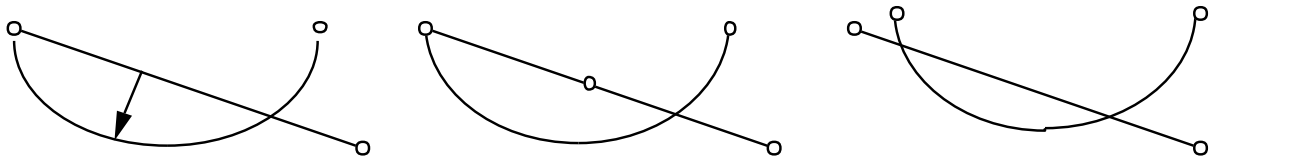

Figure 6: Tolerance (a) and not admissible overlaps (b+c) of lines.

```
controlPoints = 'VERTEX' (coord2| coord3| coordType-name) ['BASE' explanation].
```

These points are labeled as control points which have a special meaning (i.e. start and end points as well as bending points).

The value ranges of the coordinates are described first. The possibility for describing additional constraints exists via an explanation. In this way, for instance, it can be requested that the coordinates are not of a fixed type but must correspond to the points of a specific table.

The intermediate points of circular arcs are not considered as control points. Their coordinates do have the same value range (domain) as the control points.

2.2.6. Single surface and area

Surface as general notion means a bordered massive continuous and plane area. A surface always has a continuous exterior boundary. If it is the case that not the whole area inside this exterior boundary belongs to the general surface then there are one or several additional continuous boundaries, so-called interior boundaries of the surface. The areas separated in this way are called ‘enclaves’.

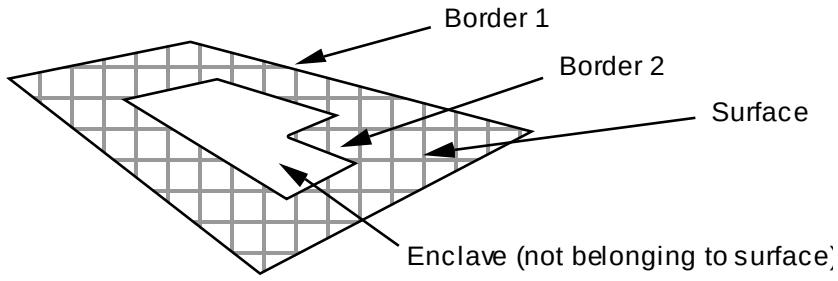

Figure 7: Area with border and enclaves

Partial surfaces only belong to the same surface if they are connected massively. Do the touch only in a point, they build two separate surfaces.

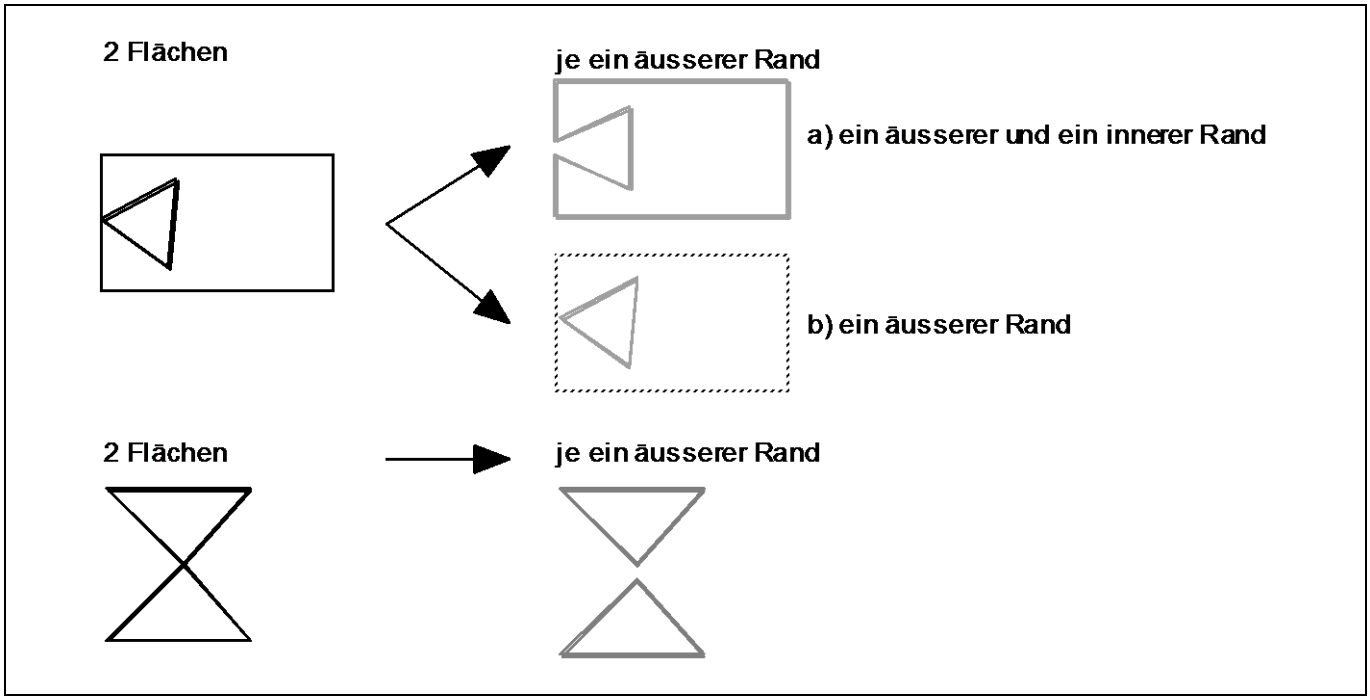

Figure 8: Examples of partial surfaces. INTERLIS allows variant a and b.

With INTERLIS one can describe whether general surfaces of different objects of the same table may overlap or not.

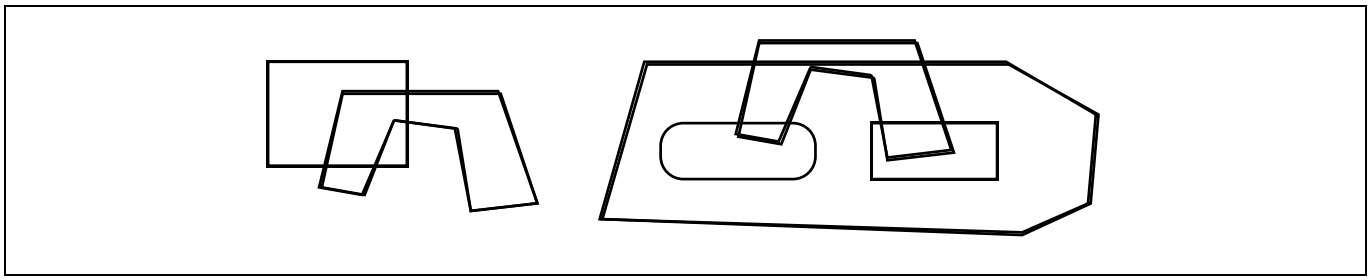

Figure 9: Single surfaces (SURFACE)

Single surface

For general surfaces which overlap completely or partially, i.e. which may not only have common border points (nodes), there exists the geometric attribute type SURFACE (polygon, see figure 9 with examples).

Area partition

Area partition describes a collection of surfaces, which form a non-disrupted, non-overlapping plane.

The area partition has an exterior surface with an unlimited number of enclaves. All other surfaces cover entirely the enclaves of the exterior surface. For this type of surface there exists the keyword AREA.

With that, an area object is connected to precisely one surface of the area partition. Every surface of the area partition is allocated to at most one area object. Therefore, it is not allowed that two surfaces of the area partition with a common boundary both do not belong to an area object.

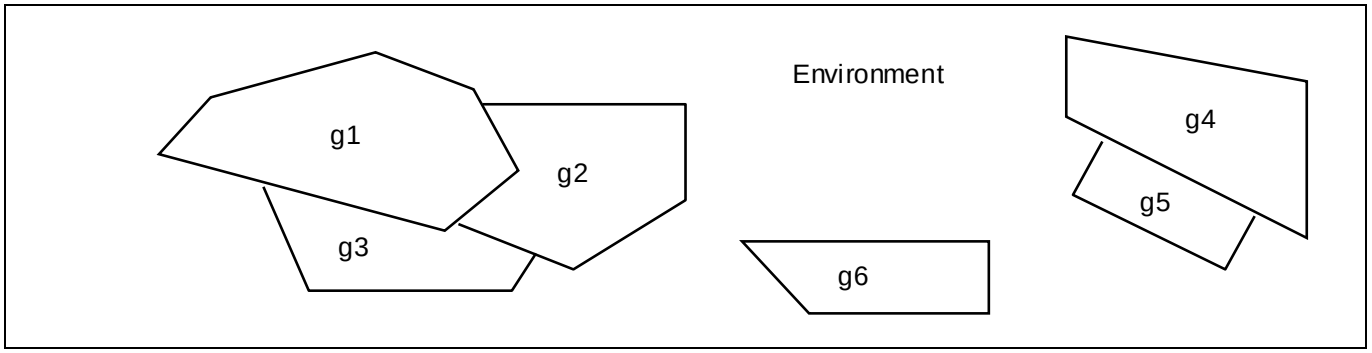

Figure 10: Area partition (AREA)

Syntax

```
surfaceAreaType = ( 'SURFACE' form controlPoints [intersectionDef]
                  | 'AREA' form controlPoints intersectionDef )
                  [ lineattrDef ].
lineattrDef = 'LINEATTR' '='
                attributes [ identifications ]
              'END'.
```

In combination with the geometric attribute type AREA it is not allowed to make it optional.

Table structure

In contrast to all the other attribute types, a further table is implicitly formed with single surfaces (SURFACE) and area partitions (AREA) which contains the lines. For table-name <main-table-name>_<attribute-name> is taken.

Any division of boundaries into individual line strings and their respective direction can be chosen and may change with different data transfers.

Attributes can also be established (lineattrdef) if required for the line strings. But these attributes cannot be permitted to be again of area or surface attribute types.

In order to link surfaces to single surface objects themselves a relation attribute is defined for each line object which points to the SURFACE object. The link between surfaces and AREA objects is described below.

Lines of area partitions

Line objects from an area partition need to be always real boundary lines. Therefore it is not allowed that lines exist where on their both sides the same area object lies (figure 11).

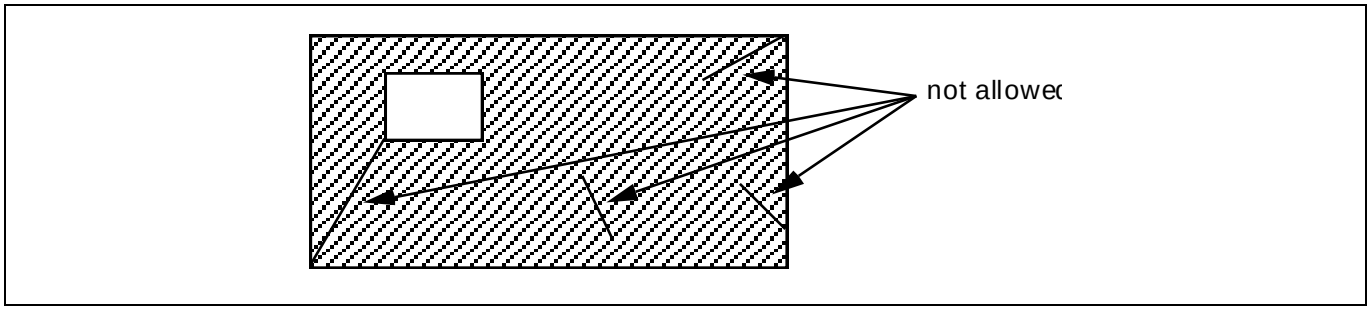

Figure 11: Not allowed area configurations.

Along the whole length of a line object the adjacent surfaces are not permissible to change. Therefore, lines must not exceed real edges (i.e. more than two intersecting lines).

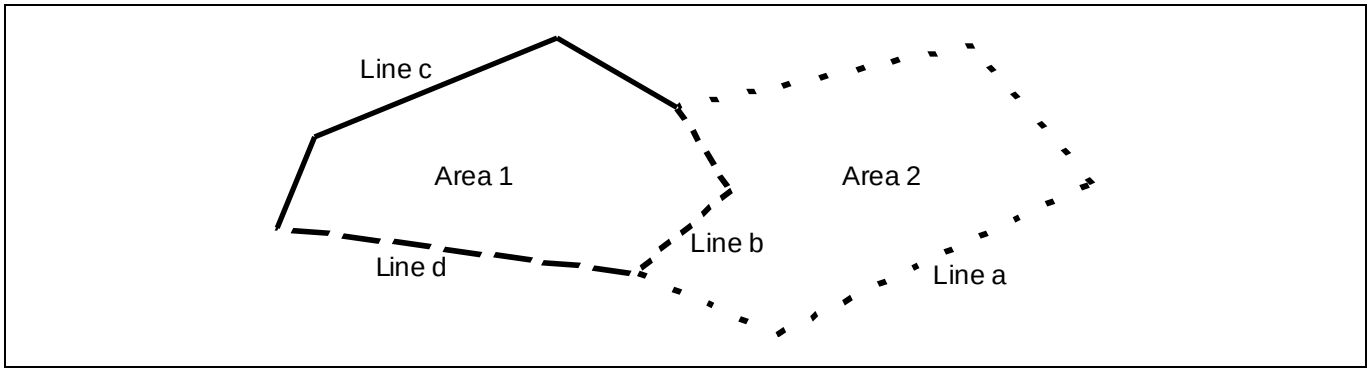

Figure 12: Line objects a, b, c, d with surface 1 and 2 of the area partition

In figure 12 the main table contains the objects surface 1 and surface 2 and the line table contains the objects line a, line b, line c and line d.

The link between the area objects and the surfaces – which in turn are build up with line objects – is geometrically established by indicating for each area object his area attribute an “area reference point” (centroid). This centroid is not allowed to be positioned within the overlap tolerance of lines. The area within the dotted line is no problem (cf. figure 13). The position of the centroid may change with different data transfers.

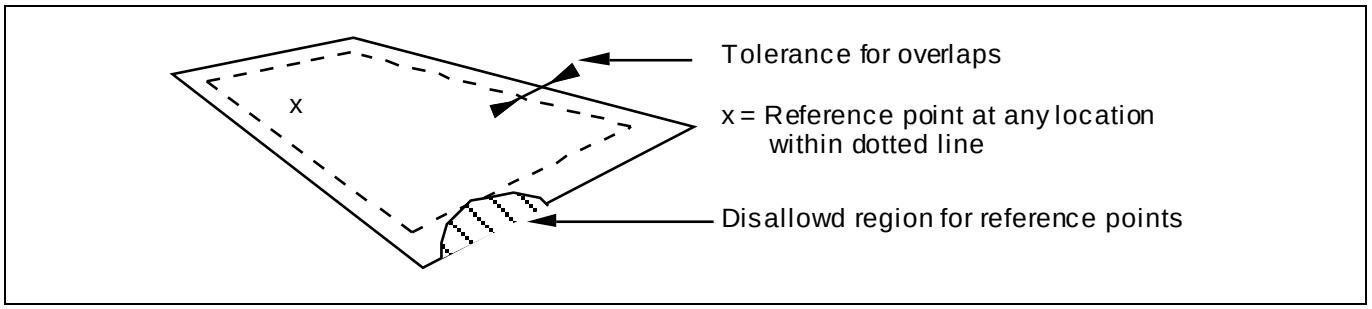

Figure 13: Allowed area of area reference points (centroids)

2.2.7. Evaluations

```
evaluations = 'DERIVATIVES' evaluation-name
                global-valueRangeDef (* theme *)
              'END' evaluation-name '.'.
```

Evaluations have the same structural build as the model. However, they do not define any new data. The individual attribute values are evaluations, functions of the model data.

2.2.8. Arrangements

The arrangements describe how the data of the model and the evaluations are arranged on the transfer file. Four things can be influenced:

- With the output of lines it can be required that apart from the point coordinates, additional information is also put out (VERTEXINFO).

- Where relations exist between tables (relation attribute), requirements can be made that the table shown is output, not singly but as attributes of the upper table (<-).

Example:

```
MODEL X
  TOPIC Y =
    TABLE A =
      a1: TEXT*10;
      a2: TEXT*5;
    IDENT
      a1;
    END A;

    TABLE B =
      b1: TEXT * 12;
      b2: -> A;
    IDENT
      b1;
    END B;
    :
    :
  VIEW  X
        Y.A <- B.b2;
```

Within the treatment of table A in addition to object a the objects of B which refer to a are output as derivated attributes (cf. 3.4.7). If all relationships of a table are output the table itself is not output.

- With area partitions and single surfaces it can be required that the geometry are output, not in the context of the line table, but directly as evaluation attributes (CONTOUR) (cf. 3.4.7.).
- With area partitions the question furthermore arises, whether the area lying outside the exterior surface should be output as an object or not. With WITH PERIPHERY this is desirable.

```
arrangement = 'VIEW' model-name
                {theme-name '.' table-name ':' arrangementDef
                {',' arrangementDef } ';'}
              'END' model-name '.'.
arrangementDef = ('VERTEXINFO' lineattrDef-name explanation
                  | 'WITH' 'PERIPHERY' AreaType name
                  | 'CONTOUR' surfaceAreaType-name [ 'WITH' 'PERIPHERY ]
                  | '<-' table-name '.' attribute-name ) .
```

2.2.9. Format

It must be established for the transfer whether the file is built up according to the rules of the fixed or free format (cf. ch. 3.2.). With fixed format the line lengths and the lengths of the identifications must be established.

```
format = 'FORMAT' ('FREE' |
                   'FIX' 'WITH' 'LINESIZE' '=' posNumber ','
                     'TIDSIZE' '=' posNumber) ';' .
```

2.2.10. Coding

In order that the characters can be recognised without doubt on the target system, the character set used must be documented. If no font definition is provided the 8-bit-character set of the ISO norm is valid (ISO 8859-1:1987). Otherwise the admissible character set has to be defined with an unique explanation. For the permissible characters in text attribute reference is made to chapter 3.4.5.5.

For some facts special characters are used on the transfer file. INTERLIS only defines suggestions for that (DEFAULT). However, for particular cases they can be changed. The code number of the desired character must then be entered within the chosen character set.

The following rules shall apply:

Empty characters can occur within text attributes. In order that the logical structure in the fields will nevertheless not be disturbed (cf. ch. 3.1.), a substitute character is used instead of the empty character. INTERLIS normally (DEFAULT) uses the underscore (_) character for that (ASCII 0x5f).

With attributes which are defined as optional, the values can be undefined. Instead of a value a special character is then inserted. INTERLIS normally uses @ (ASCII 0x40) for that.

It can happen that the character length is not sufficient to include all the fields. The logical character is then divided into further physical characters (cf. ch. 3.1.). In order to make it clear that the logical character has not yet been closed, a continuation character is added directly in front of the character end. INTERLIS normally uses \ (ASCII 0xSc) for that.

Within the transfer every object receives an identification which is unambiguous within the table, a so called object or transfer identification (TID). On the one hand a relation can be defined in a straightforward way, on the other the identification can also be of great help with all kinds of problems. The build-up of the transfer identification is fundamentally of free choice. It is taken as text of any length. However, with fixed format a maximal length must be established (cf. above). In order to facilitate the production of relations on the target system it is useful to further specify the type of identification. With I16 or I32 it is stated that the identification is an integer-number with an precision of 16 or 32 bit.

```
coding = ‘CODE’ font specCharacter transferId ‘END’ ‘.’.
font = ‘FONT’ ‘=’ explanation  ‘;’.
specCharacter = ‘BLANK’ ‘=’ ( ‘DEFAULT’ | code ) ‘,’
                ‘UNDEFINED ‘=’ ( ‘DEFAULT’ | code ) ‘,’
                ‘CONTINUE’ ‘=’ ( ‘DEFAULT’ | code ) ‘;’
transferId =’TID’ ‘=’ ( ‘I16’ | ‘I32’ | ‘ANY’ | explanations ) ‘;’.
```

3. Transfer file construction

3.1. System oriented structuring

A transfer file consists of a series of logical lines, independent of the actual content. In the following descriptions the line ends will be shown as NL. The build-up of a logical line can be seen from two different points of view:

Physical structure

A logical line can be divided into any number of physical lines. The start of the first one and the end of the last physical line does not require any special consideration. At the end of a physical line which does not stand at the end of a logical line, the line continuation character must be placed immediately in front of the end of the line. Normally **** is used for this. With the description of the coding (cf. ch. 2.2.10) another character can be defined for this. At the start of a continuation line and the first field to be inserted as field division is the line token CONT and an empty character. All characters before the continuation character and the characters after the continuing lines are belonging to the logical line but not the continutation character, not the start sign CONT and not the immediately succeeding space character.

The manner in which a transfer file is divided into physical lines is not arranged by INTERLIS. It has to be defined on a deeper interface level. Without further agreements the physical linesare closed with LF (line feed, ASCII 0xA). Immediately before the LF there may be an optional CR-character (ASCII 0xD).

Logical structure

The logical line consists of a row of fields. The arrangement of the fields on the line and the relation to the physical structure is dependent on the type of formatting of the data part of the transfer file (cf. ch. 2.2.9).

The following description of data structuring and the coding of the individual fields only builds on the logical structuring. The method by which the individual fields are separated from each other depends on whether the file is built up according to free or fixed formatting (cf. ch. 3.2). In either case, two fields in the same logical line must be separated by a dividing character (space or tab).

3.2. Free and fixed format

The data section of a transfer file can be formatted,free or fixed. Free formatting is especially suited to modern programming languages such as PASCAL, MODULA, while the fixed format is particularly applicable with languages such as BASIC and FORTRAN.

3.2.1. Free format

In free format the individual fields must be divided by at least an space character or a tabulator. The field length is always the same as the length of the actual value. A logical line can be divided at any spot, particularly in the middle of a field (cf. ch. 3.1). The characters necessary for the division therefore have to be eliminated during decoding, in order to regain the character flow of the logical line.

3.2.2. Fixed format

In fixed format the individual fields are divided by an space character. The individual field has the following build-up:

First character

(in so far as the corresponding attribute is optional) Characters for display, whether the value is defined or undefined. If the value is defined, the position contains a space character. With an undefined value the undefined character (cf. ch. 3.4.4) is inserted.

Further characters

further characters for the value. Enough characters have to be reserved in order to accommodate the maximum value. The concrete definitions are given in chapter 3.4. Numerical fields are always arranged right justified, text fields left justified. Empty positions at the end of a field are always represented as spaces, those at the start or within a text field with the substitute character (cf. CH. 3.4.5. and 2.2.10).

If all the remaining fields do not fit within the line length established for the fixed format (cf. ch. 2.2.9) then the line must be subdivided. It is subdivided in such a way that the field will fit completely on a line. A continuation line is only begun when the actual field (provided it is not the last in the logical line) plus the characters for field division (empty character) can not fit into the actual line. Field division and continuation characters must therefore always occur on the old line.

The fixed format is then built up in such a way that it can also be read according to the rules of the free format.

With regard to reading in the fixed format, sequences of the lines and line tokens are chosen in such a way that the next line can always be read with the current format. Repeated, internal reading (reread) is therefore not necessary. In many cases lines with the same line token occur in multiple order. The termination of the sequence is made by a line which only consists of a line token typical of the termination. Thus one always reads with the same format, and tests whether the line token does not signal the termination. Furthermore, in line geometry, the same format can be expected, but one must start from different line token for different connecting geometric types.

3.3. Functional structuring

A transfer file always has the following construction:

```
transfer-file = 'SCNT' NL contents-description
                ( 'MOTR' NL model- and transfer-description
                | 'MTID' transfer-name NL)
                data
                'ENDE' NL.
description = { descriptionLine } '////' NL .
descriptionLine = any_apart_from_////_on_linestart NL.
data = { modelD } .
modelD = 'MODL' model-name NL
         { themeD }
         'EMOD' NL .
```

A transfer file further contains information about the contents, namely the geographical extract. This information is defined as pure text without formatting. Those involved in the transfer can of course establish formalisations themselves. The transfer description then follows (cf. ch. 2.2), or an indication, which clearly identifies a transfer description known to both sides. The dates follow next.

These are out put by levels, in the order of the model definition. It is~ however, admissible that not all the themes suited to a model are conveyed. If, however, a theme is conveyed, then it must be done completely.

```
themeD = 'TOPI' theme-name NL { tableD } 'ETOP' .
tableD = 'TABL'table-name NL
         { objectD }
         [ perimeterDecl ]
         'ETAB' NL .
objectD = 'OBJE' transferId-text attributeD .
attributeD = { basisattribute | centroid-coordinate } NL
             { linesequence } { surfaceA | areaA } { tableA}.
perimeterdecl = 'PERI' transferId-text NL.
```

The order of the tables results from the order of the table definitions within the themes. Tables for single surface and area partition attributes follow the main table in the order of attributes, in so far as it is actually out put as a table. For each task all the objects are output. The order is optional. In the case of area partitions, after the last object it is also shown which object forms the perimeter (PERI), provided the perimeter actually has to be denoted (cf. ch. 2.2.8).

The order of the attributes per object corresponds primarily the order of the attributes in the description. For basis attributes this rules is valid without exception.

With area partition and single surface attributes the following rules apply:

Output of geometry of surfaces within the line table:

The line table is output immediately after the main table. Are there several surface attributes the order of the line tables is derived from the oder of the line attributes.

Nearby the line object a relation attribute is denoted which contains the link to the surface object.

Output of geometry of area partitions whithin the line table:

The line table is output immediately before the main table.

Within the area object instead of the area line objects the coordinates of the centroid is output which lie inside the surface (cf. ch. 2.2.6).

Output of geometry with the area partition or single surface object (CONTOUR):

The sequence is derived from the order of the attribute definition. The designation of the perimeter only occurs if the object too is indicated, i.e. when WITH PERIPHERY is requested.

Outputs of surface line attributes and evaluation attributes (surface, area attributes, table derivations) are not directly output on the object line. Their output follows the basis attributes and centroid coordinates of area attributes. First the line attributes are output, then the area centroid coordinates and finally the table with the evaluation attributes, each in the order how the are organised.

3.4. Coding definition

3.4.1. Line token

The first field of every logical line contains the line token. It always has four characters (signs).

3.4.2. Theme and table names

The maximum value lengths of theme and table names is 24 characters. Further characters are ignored.

3.4.3. Transfer identification

A transfer identification (`transferId-Text`) covers a field. In structure it is handled in basically the same way as a text. The maximum value length for fixed formats is described in the transfer parameters (ch. 2.2.9.). With free formats it does not have to be established.

3.4.4. Undefined attributes

Attributes which are marked as “OPTIONAL” in the model description can show undefined values. The undefined-character serves for display. Normally the character @ is used. In the context of the description of the coding (cf. ch. 2.2.10) it can, however, be redefined. With free formats the undefined-character is output instead of the value. With a fixed format it is displayed at the site reserved for it. This site is independent of the place reserved for the value, so that the undefined-character cannot collide with the format instruction (cf. ch. 3.2.2.). Space characters are output at the places intended for the attribute values.

3.4.5. Basis attributes

The relation attributes and the attributes with a basis type as value range are denoted as basis attributes.

3.4.5.1. Relation attributes

Relation attributes cover one field. The transfer-identification of the object which is pointed to is output as the attribute value (cf. ch. 3.4.3).

3.4.5.2. Decimal numbers attributes

Attributes, whose values are given with “dec” (coordinates, lengths, etc) are always output according to the value range definition. The format is the declaration, without the scaling part. The decimal point is placed in so far as it is given in the definition.

3.4.5.3. Coordinates

Coordinates consist of two (COORD2) or three (COORD3) fields. The first field contains the east value, the second the north value, the third the height value. If the coordinates are undefined, all the fields must contain the undefined-character. The maximum value length is established for every component.

3.4.5.4. Lengths, units of square measurement and angles

Lengths, surfaces and angles consist of a field.

3.4.5.5. Values

Values of numbering ranges cover a field.

3.4.5.6. Text

A text attribute consists of a field. So that possible empty characters (spaces) within a text are not taken as the termination of a field, the empty characters within the text must be put redefined to

another character. Without further denotation the understroke (ASCII-value 0X5F) is used. Deviations can be defined in the description of the transfer parameters (cf. ch. 2.2.10). Because the different character sets do not always necessarily contain the same characters, it is advisable to do without further agreements and beyond the 7-bit ASCII-characters only to use the following alphabetic characters (cf. ISO 6937/2-1983) of the ISO-8859-1 character set (contains all umlauts and accents also in upper case).
Proposed characters from the PC chararcter set:

```
Д, в, д, а, б, ж,
З, з,
Й, к, л, и, й,
о, п, м, н,
С, с,
Ц, ф, ц, т, у,
Ь, ы, ь, щ, ъ
```

3.4.5.7. Date

The date covers a field. It is output in the form YYYYMMDD (YYYY stands for the year, MM for the number of the month [01 to 12], DD for the day). Thus 1 August 2001 is given as 20010801.

3.4.5.8. Enumeration

An enumeration value covers a fleld. The information takes place according to the order of numbers. With that all admissible values are numbered, ie. all the elements of the definition which do not show a sub-listing. The numbering starts with 0. The maximum value length is the maximum order of numbers.

Example:

In the case of an enumeration

```
colour: (red (darkred, carmine, orange), yellow, green (brightgreen, darkgreen),
blue, violet);
```

the following values occur.

```
O for red-darkred 1 for red-carmine ... 3 for yellow 4 for brightgreen ... 6 for
blue 7 for violet.
```

3.4.5.9. Horizontal and vertical alignment

Because horizontal and vertical alignment can be taken as pre-defined enumerations, the results are output according to the rules of enumeration.

3.4.6. Line attributes

Attributes whose value ranges are defined with lines are denoted as line attributes. The individual line pieces of the line string are put out as line sequence. Thereby one line results per control point. Additional lines occur according to the type of the connecting geometry. With arcs a supplementary point is output in the vicinity of the centre of the arc. For special connections the coding, in the context of the general coding rules of INTERLIS, is left up to choice. So that one can read with the same format, a line should have the construction “linetoken point further_information NL”.

```
linesequence = [startpt (* connection *)] ‘ELIN’ NL.
```

```
startpt = 'STPT' start-point NL.
connection = (straight | arc | specialconnection ) 'LIPT' point NL.
straight = .
arc = 'ARCP' intermediate-point NL.
specialconnection = specialconnection-foreign_rule.
point = coordinate [ pointinfo ].
```

For the data transfer to different systems it is practical to give further point information as well as the point coordinates. The definition is described in Chapter 2.2.7 (VERTEXINFO). The coding is left to those in charge of transfer, but must follow the general rules from chapter 3.1 and 3.2. If one wants to read with fixed format, it must be born in mind that in places where different input is to be expected, that one can still read using the same format.

3.4.7. Evaluation attributes

Attributes which do not directly belong to actual tables but result from the evaluations of other tables, are denoted as evaluation attributes. This applies especially for the types single surfaces and area partitions, which imply a further table. Beyond the evaluation of single surfaces and area partitions the evaluation of other tables and any evaluations are available. The definition possibilities are described in Chapter 2.2.8. Only the corresponding file construction is given here.

```
evaluationattribute = ( surfaceA | areaA | tableA | specialA ).
surfaceA = (* edge *) 'EFLA' NL.
areaA = surfaceA.
edge = 'EDGE' NL
       (* lineAttr linesequence *)
       'EEDG' NL.
lineAttr = 'LATT' { basisattribute } NL.
```

The order of the edges is undefined. An edge can be subdivided into any individual line strings. The order of the line string must be chosen in such a way that the described surface always lies to the right of the actual line (thus the exterior boundary will go round in a clockwise direction, the enclaves (islands) in an anti-clockwise direction). If further attributes are established for the lines, they are always represented before the line sequence (line token LATT). If line attributes are defined (cf. ch. 2.2.6) they will follow (LATT part).

```
tableA = 'TABA' table-name NL
         { 'SOBJ' transferId-text attributeD NL }
         'ETBA' NL.
```

All the objects of the table to be evaluated, which have a relation (relation attribute) to the actual object, are given. Per object the following sequence occurs `SOBJ transferID-text attributD NL`. The relation-attribute itself is not output.

4. The INTERLIS compiler

In order to test a schema definition (model) the INTERLIS compiler can be used.

The compiler produces in its simplest form from a schema definition a list with formats which occur in the definition and a list with the input schema and an error file in the case where the definition is incorrect.

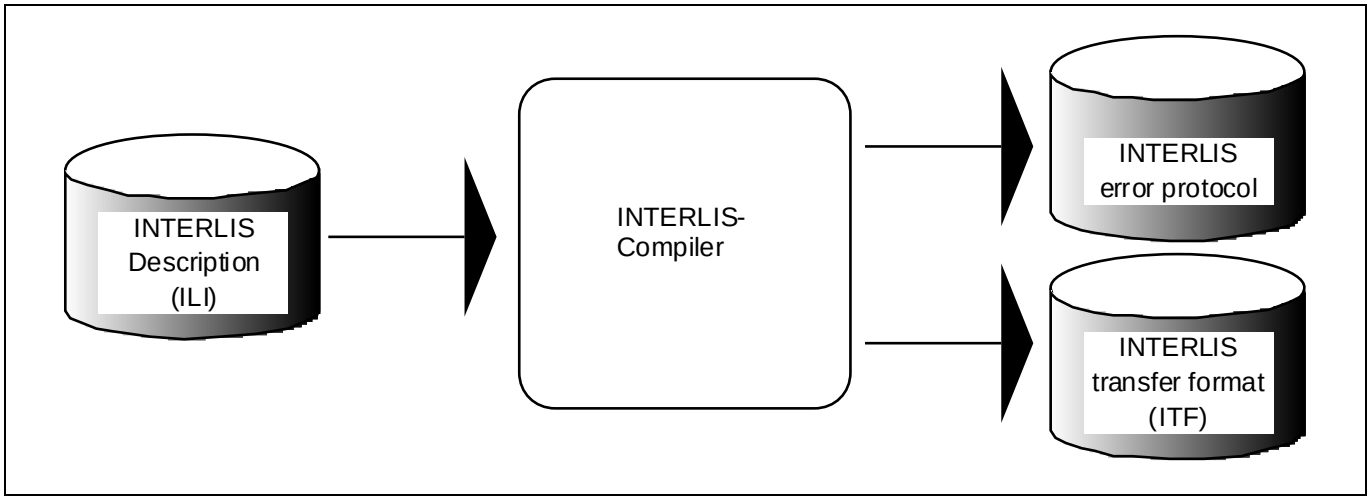

Figure 14: The INTERLIS compiler

The compiler checks the transfer description (the model or schema definition) and creates a file with errors – if there are any – and given a error-free input it produces a file which contains all tables to be transferred (i.e. modeled at the physical schema level).

Was there a transfer definition with free format specified, there will be a format description anyway. It will take the folloing assumptions:

- line length is 60 characters
- length of the transfer-identification is 1 character.

5. Example

The example already given in chapter 2.2.2. is taken up again here and explained further.

```
TRANSFER Example;

MODEL Example

  DOMAIN
    LCoord = COORD2 480000.00   60000.00
                    850000.00  320000.00;

  TOPIC groundCover =

    TABLE groundSurface =
      type : (building, fixed, humusrich, water,
              stocked, vegetationless);
      form : AREA WITH (STRAIGHTS, ARCS) VERTEX LCoord
             WITHOUT OVERLAPS > 0.10;
    NO IDENT
      !! search via geometry or building
    END groundSurface;

    TABLE building =
      assuranceNo: TEXT*6;
      surface:     -> groundSurface // type = building //;
    IDENT
      assuranceNo; !! assumption that assuranceNo is unambiguous
      surface;     !! building are allocated an exact surface
    END building;

  END groundCover.
END Example.

FORMAT FREE;

CODE
  BLANK = DEFAULT, UNDEFINED = DEFAULT, CONTINUE = DEFAULT;
   TID = ANY;
END.
```

In the context of the relational data model the structure can be represented schematically as follows:

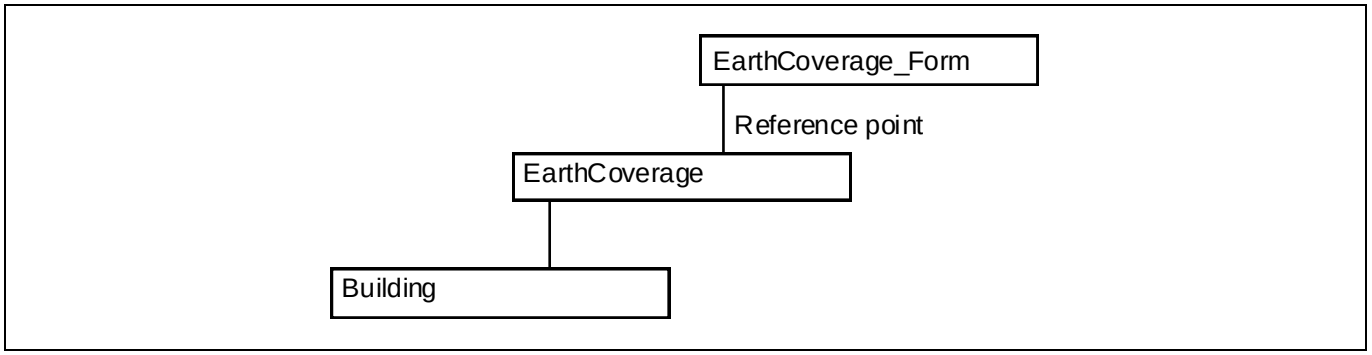

Figure 15: Schematic representation of the relationships in the example

Thus, ground cover surfaces are primarily described. In order to distinguish their characteristics, composition, the attribute `type` is introduced. In the case of building further information must be described, namely the assurance number. Because this information is only obtainable for building and not for all ground cover surfaces, a special table is constructed, relating to the ground cover surfaces.

In order to show a real transfer file, we will choose a simple real situation:

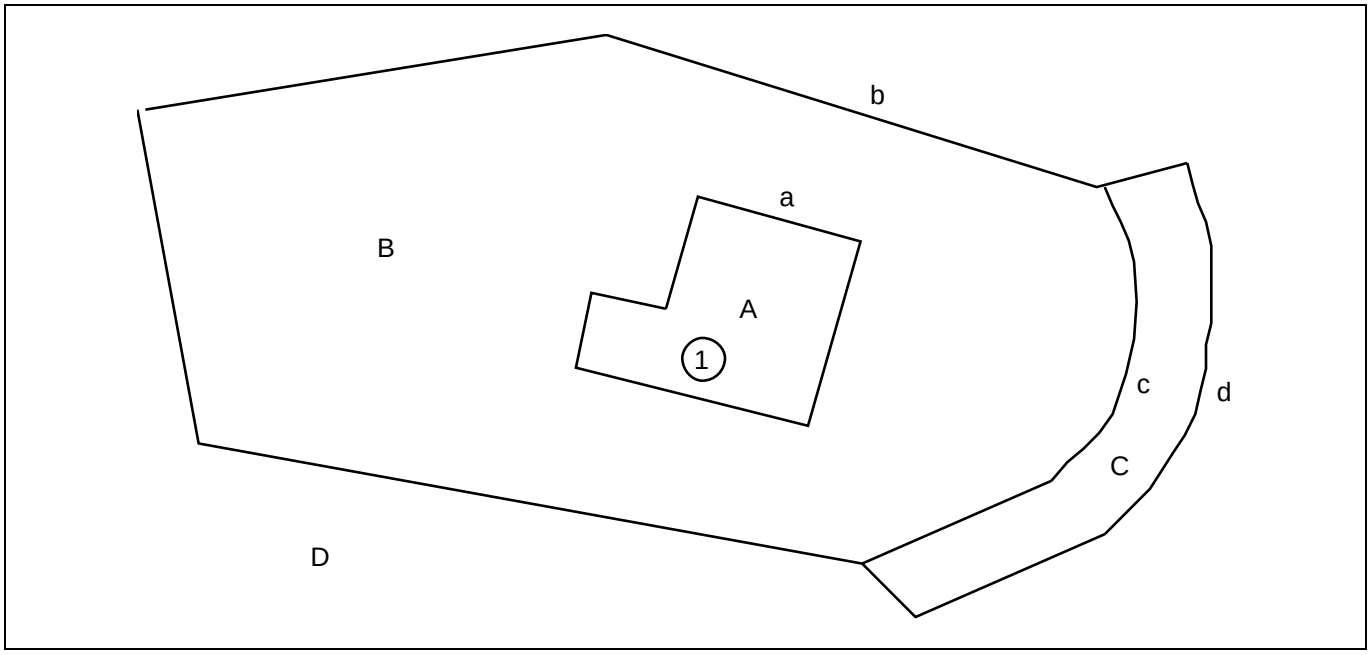

Figure 16: Example

A house (with the assurance number 958) is situated inside a humus-rich area, which borders a fixed area (street).

The given digits and letters indicate the identification on the transfer file (cf. below). With lower case letters the lines are indicated, objects from the area objects of groundSurface are indicated by capital letters, those from the table building form with digits inside circles.

The INTERLIS compiler translates the transfer description in the following format:

```
Topic groundCover

Table groundSurface_form

OBJE-Format
OBJE 1
1: Objectident          objekt identification of the chain
Followed by
Line-Records of line-attr   Line:

llab  111111.11  222222.22
1: Coordinate
2: Coordinate

Sequence of objects closed by
ETAB-Record

Table groundSurface

OBJE-Format
OBJE 1  2   333333.33   444444.44

1: Objectident          the names of the indicated fields is given
2: type
3: form                 the easting component of the area centroid
4: form                 the northing component of the area centroid

Followed by
ETAB-Record

Table building

OBJE-Format
OBJE 1 222222 3
1: Objectident          Objekt-identification des Gebaeudes
2: assuranceNr
3: surface              reference to groundSurface
Sequence of objects closed by
ETAB-Record
```

From the example the following transfer file results:

```
SCNT
transfer-file of the example
////
MTID according to description of example
MODL Example
TOPI groundCover
TABL groundSurface_form
OBJE a                          line-id
STPT 600146.92 200174.98
LIPT 600138.68 200187.51
LIPT 600147.04 200193.00
LIPT 600149.79 200188.82
LIPT 600158.15 200194.31
LIPT 600163.64 200185.96
LIPT 600146.92 200174.98
ELIN
OBJE b
STPT 600140.69 200156.63
LIPT 600118.19 200179.82
LIPT 600113.00 200219.97
LIPT 600148.30 200228.97
LIPT 600186.38 200206.82
ELIN
OBJE c
STPT 600186.38 200206.82
ARCP 600183.52 200188.65
LIPT 600170.18 200176.00
LIPT 600140.69 200156.63
ELIN
OBJE d
STPT 600186.38 200206.82
LIPT 600194.26 200208.19
ARCP 600190.75 200185.21
LIPT 600174.10 200169.00
LIPT 600145.08 200149.94
LIPT 600140.69 200156.63
ELIN
ETAB
TABL groundSurface
OBJE A 0 600148.20 200183.48  A is object-Id; O for building; centroid
OBJE B 2 600133.95 200206.06  B is object-Id; 1 for fixed; centroid
OBJE C 1 600168.27 200170.85  C is object-Id; 2 for humusrich; centroid
ETAB
TABL building
OBJE 1 958 A
ETAB
ETOP
EMOD
ENDE
```

Index

The example in chapter 5 has been corrected on 17.08.2016 by COGIS.