SVEUČILIŠTE U ZAGREBU - GEODETSKI FAKULTET
UNIVERSITY OF ZAGREB - FACULTY OF GEODESY
Zavod za primijenjenu geodeziju; Katedra za upravljanje prostornim informacijama
Institute of Applied Geodesy; Chair of Spatial Information Management
Kačićeva 26; HR-10000 Zagreb, CROATIA
Web: www.upi.geof.hr; Tel.: (+385 1) 46 39 222; Fax.: (+385 1) 48 28 081

Graduate study of Geodesy and Geoinformatics

Orientation: Geoinformatics

MASTER THESIS

Schema transformation between different modeling languages in the INSPIRE context using GML

Author:

Ivan Mihaljević

Put kor Meterize 25

Šibenik

imihaljevic@geof.hr

Mentors: doc. dr. sc. Vlado Cetl

Dipl.-Inf. Tatjana Kutzner

Zagreb, September 2011.

Thanks:

To my mentors Dipl.-Inf. Tatjana Kutzner and doc. dr. sc. Vlado Cetl, who made this thesis possible with endless patience and shared knowledge;

To the board of Faculty of Geodesy in Zagreb, whose open-mindedness made this international project possible;

To the Technische Universität München for the warm welcome and this wonderful opportunity;

To my parents, who supported my decision to take part in this project even if they didn't agree with it;

To my sister Vlatka, for helping me traverse all the communication barriers;

To Ivan, Ksenija, Roko and Ana, for showing me how much family means;

To Helena, who makes me feel like home wherever I am.

Schema transformation between different modeling languages in the INSPIRE context using GML

Ivan Mihaljević

Abstract: *The INSPIRE Directive requires that EU member states adjust their spatial data systems in a way that will allow unifying them in interoperable geospatial web services and standardized transfer formats. European data models, called INSPIRE Data Specifications, are defined for important topics as a final goal of transformation. The Unified Modeling Language (UML) is used for the definition of the INSPIRE data models.*

At the moment, most of the countries use their own spatial data systems and also their own data models. Some countries (e.g. Germany and Switzerland) define their data models with the UML and apply UML profiles as well. By defining transformations between data models using the same UML profile it is possible to unify spatial data into a single system.

However, the spatial data models of Germany, Switzerland and INSPIRE are represented by UML profiles that don't conform to the UML specification, but expand it with their own individual elements. This additionally complicates the transformation.

One of the approaches to this problem is to define a core UML profile as an intersection of different UML profiles. The transformation between data models conforming to different UML profiles could then be defined using the core UML profile as an intermediate step.

The goal of this thesis is to explore the Geography Markup Language (GML) specification as a possible solution for the above mentioned core UML profile.

Keywords: *UML profiles, INTERLIS, INSPIRE, GML*

Transformacija shema različitih jezika za modeliranje u kontekstu INSPIRE-a korištenjem GML-a

Ivan Mihaljević

Prošireni sažetak: *INSPIRE direktiva zahtjeva od članica Europske Unije da prilagode svoje prostorne podatke na način koje će omogućiti njihovo ujedinjenje u interoperabilne geoprostorne web sevise te standardizirane formate za transfer podataka. Europski modeli podataka, zvani INSPIRE specifikacije za podatke, su definirani za važna područja kako konačni cilj transformacije. Unified Modeling language (UML) je korišten za definiciju INSPIRE modela podataka.*

Trenutno, većina država koristi svoje modele i sustave prostornih podataka. Neke države (npr. Njemačka i Švicarska) definiraju svoje modele podataka pomoću UML-a koristeći UML profile. Definiranjem transformacija između modela podataka koji koriste iste UML profile moguće je ujediniti prostorne podatke u jedan sustav.

Problem je što su modeli prostornih podataka Njemačke, Švicarske te INSPIRE-a predstavljeni UML profilima koji ne podliježu UML specifikaciji, već je proširuju svojim vlasititim elementima. Ovo dodatno otežava transformaciju.

Jedan od pristupa riješavanju ovog problema jest uvođenje osnovnog UML profila kao presjeka različitih UML profila. Tada bi mogli definirati transformaciju koja se sastoji od slijedećih koraka:

1.) Modeliranje početnih modela prema osnovnom UML profilu

2.) Modeliranje osnovnih modela zasnovanih na osnovnom profilu prema ciljnom modelu (INSPIRE) zasnovanom na osnovnom UML profilu

3.) Modeliranje ciljnog modela zasnovanog na osnovnom UML profilu prema ciljnom modelu zasnovanom na ciljnom (INSPIRE) profilu.

Ovaj rad bavi se samo prvim dijelom ovog procesa transformacije.

Model koji je odabran za transformaciju je švicarski katastarski model, MOpublic, napisan u INTERLIS jeziku. Cilj ovog rada je modelirati UML profil MOpublic modela u osnovnom UML profilu.

Smisao ovog istraživanja je istražiti sposobnost Gography markup language-a (GML-a) za ulogu osnovnog UML profila. GML ima svoju sintaksu i pravila, ali se može prevesti u UML okruženje direktnom transformacijom opisanom u GML specifikaciji. To znači da je moguće definirati skup pravila za transformaciju elemenata INTERLIS jezika u

elemente GML jezika, te pomoću tih pravila prevesti MOpublic model u GML okružje. Proizvoljno se naknadno može primijeniti direktna transformacija u UML sintaksu.

Struktura ovog rada je zamišljena na način da pruži objašnjenje pojma UML profila općenito, te spomene neke važnije standarde koji se odnose na profile. Trenutno stanje je prikazano primjerom nekoliko UML profila različitih država. Potom slijedi analiza INTERLIS jezika, te poglavlje koje govori o GML-u kao osnovnom UML profilu. Slijedeće poglavlje definira pravila prevođenja iz INTERLIS UML profila u GML. Konačno, pravila prevođenja su primijenjena na Švicarski model podataka, MOpublic. U zaključku je iznesena procijena rezultata.

Rad je napisan na engleskom jeziku zbog toga što je produkt ERASMUS međunarodne suradnje sa TUM-om (tehničkim sveučilištem u Münchenu).

Ključne riječi: *UML profili, INTERLIS, INSPIRE, GML*

Schema transformation between different modeling languages in the INSPIRE context using GML

Ivan Mihaljević

T A B L E O F C O N T E N T S

1. Introduction

INSPIRE, an European Union initiave, has set its goal to establish an European spatial data infrastructure that will potentiate accessibility and interoperability of geographical information for various purposes.

One of the main obstacles that has to be traversed before the INSPIRE Directive can be enforced is divergence between different spatial data models currently used by European countries. For interoperability to be achieved, a transformation method has to be determined between each of these mdoels and European data models (defined by INSPIRE Data Specifications).

The data models of some counties (eg. Germany, Switzerland) are based on different Unified Modeling Language profiles. INSPIRE UML profile is also defined as a profile of the target model of transformations. The problem is that these UML profiles don't conform to the UML specification, but expand it with additional elements. This complicates the process of defining model transformations.

A possible solution for this situation is introduction of a core UML profile, which would be composed of all the elements that are in common to all the UML profiles. Then the transformation could be taken apart into smaller steps:

1.) Mapping the source models to the core UML profile

2.) Mapping the source models based on the core profile to the destination model (INSPIRE) based on the core profile.

3.) Mapping the destination model based on the core profile to the destination model (INSPIRE) based on the destination (INSPIRE) profile.

This thesis deals only with the first part of this transformation process.

A selected model for the transformation is a cadastral model of Switzerland, called MOpublic, written in INTERLIS language. The goal of this thesis is to map MOpublic UML profile to a core UML profile.

The purpose of this research was to determine capabilities of Geography markup Language as a core UML profile. GML has its own syntax and rules, but it can be transformed to a UML syntax directly. This means that transformation of MOpublic model can be executed by defining mapping rules from the INTERLIS language to GML syntax. Arbitrarily, direct transformation to UML can be applied afterwards.

Structure of this thesis was designed to give an introduction about UML profiles in general, together with several important standards related to profiles. Current situation is shown by examples of several UML profiles of different countries. After that comes the analisys of INTERLIS language and introduction of GML as a core UML profile. Following chapter defines mapping rules from the INTERLIS UML profile to GML. Finally, the mapping rules are applied on the Swiss data model called MOpublic. In conclusion, evaluation of results is presented.

2. UML Profiles

Unified Modeling Language (UML) is a modeling language designed for general purposes in the software engineering. It conforms to a standard created and managed by the Object Management Group.

UML syntax is a set of graphic notation tehniques. It is used to specify, visualise, modify, construct and document the details of an object-oriented system under development.

(FOLDOC 2001. Unified Modeling Language)

2.1. Definition of UML profiles

UML profiles provide means of adopting UML to certain areas of application. This is accomplished by extension mechanisms which allow adding new elements to the sintax in a way that facilitates the application of UML on a domain of interest, while avoiding the contradiction with the standard semantics.

Profile is a collection of extension mechanisms (stereotypes, tag definitions, and constraints) that are applied to specific model elements (Classes, Attributes, Operations, and Activities). (Si Alhir, Sinan 2002. Guide to applying the UML)

2.2. UML profiles for spatial data schemas

In the field of geoinformatics, the methods of collecting, storing, manipulating, visualising and exchanging the spatial data are very important to ensure interoperability and multipurposeness of spatial information. Usually, a regulation exists on a country level that prescribes a desired way to deal with these tasks. A UML profile exists as a basis for the nataional data model. However, not every country uses the same UML profile (sometimes differences exist even on a state level, eg. In Germany). Next section describes the ISO19103 UML profile, which is a basis of the UML profiles of Germany and the EU. UML profiles of Germany, Switzerland and EU are described in the following chapters. Observations presented in these sections are a result of a study which explored these profiles in detail, exposing similarities and differences between them. (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

2.2.1. ISO19103 UML profile

Both German and EU UML profile conform to the profile described in ISO19103: „Geographic information — Conceptual schema language“. However, this profile expands the original UML specification. This in fact creates an independent modeling language that shares the syntax with the UML.

The parts of ISO19103 that are of the biggest importance to us are definitions of data types and stereotypes.

„The basic data types have been grouped into three categories, as shown in Figure 1:

a) Primitive types: Fundamental types for representing values, examples are CharacterString, Integer, Boolean, Date, Time, etc.

b) Implementation and collection types: Types for implementation and representation structures, examples are Names and Records, and types for representing multiple occurrences of other types, examples are Set, Bag and Sequence.

c) Derived types: Measure types and units of measurement.

The basic types are defined as abstract types, class name in italic, and appropriate representations will be defined by implementation and encoding mappings for the various subtypes.“ (ISO/TS 19103, 2005.)

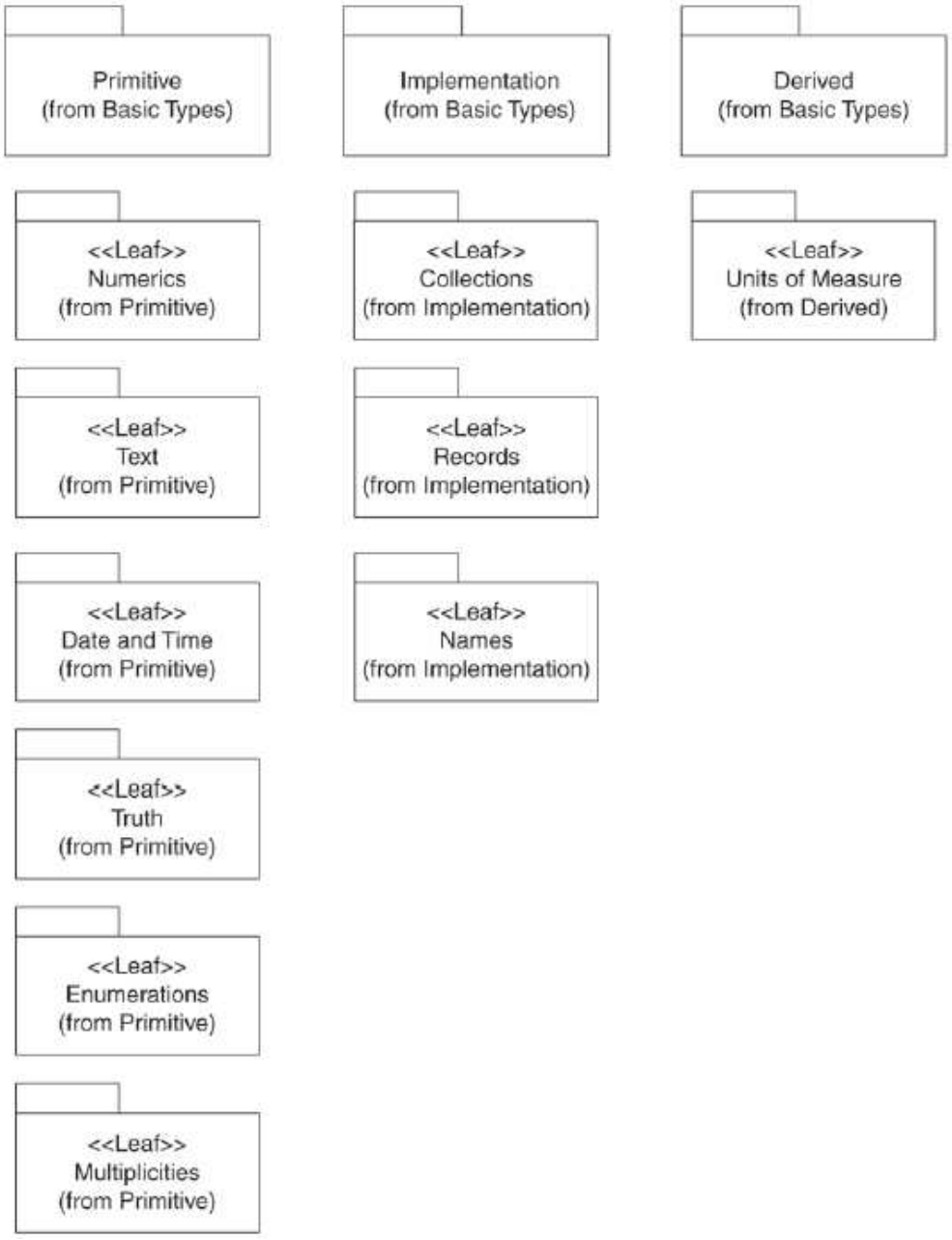

Figure 1 – The basic data types (ISO/TS 19103, 2005.)

A stereotype is an extensibility mechanism of modeling languages. It allows derive new model elements from the existing ones, hence extending the vocabulary of a language. These new model elements have specific properties that are suitable for a particular problem domain or otherwise specialized usage. (Object Management Group, UML Superstructure Specification, 2005.)

„The following stereotypes are defined in ISO/TS19103 specification:

a) <<CodeList>> is a flexible enumeration that uses string values through a binding of the Dictionary type key and return values as string types; e.g. Dictionary (String, String). If the elements of a list are completely known, an enumeration shall be used; if only the likely values of the elements are known, a code list shall be used.

b) <<Leaf>> is a package that contains definitions, without any sub-packages.

c) <<Union>> is a type consisting of one and only one of several alternatives (listed as member attributes). This is similar to a discriminated union in many programming languages. In some languages using pointers, this requires a “void” pointer that can be cast to the appropriate type as determined by a discriminator attribute.

Stereotypes are essential in the creation of code generators for UML models. Generally speaking, stereotypes act as flags to language compilers to determine how to create implementation models from the abstract. Enumeration and CodeList are prime examples of this. As opposed to the creation of a class, these classes generate small value domains, one value per “attribute” in the model. In an Enumeration, these values are usually represented as a 1-up number code for the attributes listed, and the list is assumed to be completely defined in the Abstract Specification. In a CodeList, the code values are not specified, and the list is extendible at any time. For examples, CodeLists can be used to implement the ISO country codes and language code used commonly in metadata specifications.“ (ISO/TS 19103, 2005.)

These stereotypes are the biggest deviation from the UML specification.

2.2.2. AAA UML Profile

A common application schema has been developed for german projects AFIS (Official Fixed Point Information System), ALKIS (Official Real Estate Cadastre Information System) and ATKIS (Official Topographic Cartographic Information System). (Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany, 2009. Documentation on the Modelling of Geoinformation of Official Surveying and Mapping)

This application schema (AAA application schema) is composed of the AAA basic schema, the AAA versioning schema and the AAA technical schema, the NAS operations and the AAA output catalogue. The AAA basic schema provides the

basis for modelling the AAA technical schema. The AAA technical schema represents the conceptual schemata which describe the universe of discourse in a formal way.

The AAA basic schema is modelled based on the ISO 19100 series of standards. The meta model of the AAA basic schema is the General Feature Model from ISO 19109 which however is extended by the meta class AA_ObjektOhneRaumbezug. The new meta class is derived from the meta class GF_FeatureType. By doing so, FeatureTypes which are not allowed to have a spatial reference can be defined.

The AAA application schema was defined with UML according to ISO 19103, i.e., UML is used according to version 1.4.2. Furthermore relevant rules from ISO 19109 are used to define the application schema.

The stereotypes defined in ISO 19103 are used in the AAA application schema. Furthermore, to make things more clear, the stereotype «FeatureType» from ISO 19136 Annex E is used for all FeatureTypes. Since all object types have to be derived from the meta class GM_FeatureType no objects can exist without this stereotype. Furthermore UML Tagged Values which are specified in ISO 19136 are used in the AAA application schema.

The encoding is defined according to the encoding rules from ISO 19136 Annex E and ISO 19139 (metadata). GML is used as data transfer format. The encoding from UML to GML is executed in two steps with an implementation schema as intermediate step.

1. The AAA application schema contains some UML constructs which are not supported by the encoding rules of ISO 19136 and ISO 19139. Therefore the application schema is converted into an implementation schema which conforms to the specifications of both standards. The implementation schema is generated script-based by means of the software Rational Rose.

2. Afterwards the implementation schema is converted into a GML application schema according to the encoding rules of ISO 19136 and ISO 19139. The software ShapeChange is used for this step.

In conclusion, the AAA UML profile is not a UML profile in terms of the UML profile definition of the OMG. The AAA UML profile is based on the “UML profile” from the standard ISO 19103 and can therefore not be considered a real UML profile as well. The rules for deriving the implementation schema are defined in the GeoInfoDoc. The encoding rules are taken from the standards ISO 19136 and ISO 19139. All rules are defined in a textual and informal way. (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

2.2.3. INTERLIS UML Profile

INTERLIS is a data description language and a transfer format with particular emphasis on spatial data (GIS data format) and the model-based method. (http://www.interlis.ch/content/index.php?language=e)

No standardised (de-jure) UML profile exists for INTERLIS. In the context of the software UML/INTERLIS-Editor a UML meta model extension was defined based on UML version 1.4.2; this meta model extension is generally (de-facto) used. This is not critical, because the INTERLIS schema is the normative schema and not its UML visualization. Furthermore a meta model exists which is not based on Meta-Object Facility, an OMG standard. This meta model is defined with INTERLIS and is not an extension of the UML meta model.

The data types and stereotypes are described in chapter 3.

A schema defined with INTERLIS is so precise that no manually created intermediate schema (which has to be updated regularly) in the form of an implementation schema is required to be able to derive the transfer format automatically.

For INTERLIS no UML profile in terms of the UML profile definition can exist, since INTERLIS contains language constructs which are not pure specializations of classes of the UML meta model (e.g. GraphicDef). This is analogous to the UML profile from ISO 19103, which is not a real UML profile because of «CodeList» and «Union». (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

2.2.4. INSPIRE UML Profile

The INSPIRE Generic Conceptual Model contains all definitions for the INSPIRE UML profile.

The meta model for the INSPIRE application schemata is the General Feature Model from ISO 19109.

Each INSPIRE application schema has to be defined with UML according to ISO 19103 and ISO 19109, with the exception that UML version 2.1 has to be used instead of version 1.4.2.

The INSPIRE Generic Conceptual Model furthermore defines which stereotypes are allowed to be used in the Consolidated INSPIRE UML Model. Most of the stereotypes have already been defined by other standards and are reused here.

The encoding is defined according to the encoding rules of ISO 19136 Annex E and ISO 19193 (metadata), i.e. in the same way as the encoding for the AAA application schemata. However, some additional rules are defined here. GML is used as data transfer format as well. In contrast to the AAA application schemata the encoding does not have to be executed in two steps with an implementation schema as intermediate step. The implementation schema is mentioned in the document, but for the Annex I themes no requirements have been identified which require this intermediate step. Furthermore the INSPIRE documents do not mention any software suitable for performing the encoding.

Just as the AAA “UML profile”, the INSPIRE UML profile is not a UML profile in terms of the UML profile definition of the OMG. The INSPIRE UML profile is based on the UML profile of ISO 19103 and therefore cannot be considered a real UML profile. Since no intermediate step using an implementation schema is necessary

for the Annex I themes, the INSPIRE documents do not contain any rules for deriving an implementation schema. The encoding rules are taken from ISO 19136 and ISO 19139. The rules are defined in a textual way.

Figure 2 shows how the INSPIRE UML profile relates to UML and to some of the standards from ISO 19100 series. “Modellierungssprache des TC211” is to make clear that the definition of the stereotypes «codeList» und «union» in ISO 19103 does not produce a new UML profile but a new modelling language. This new modelling language has the UML syntax but not the UML semantics; therefore it represents a new language. The same is true for “Modellierungssprache von INSPIRE”. (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

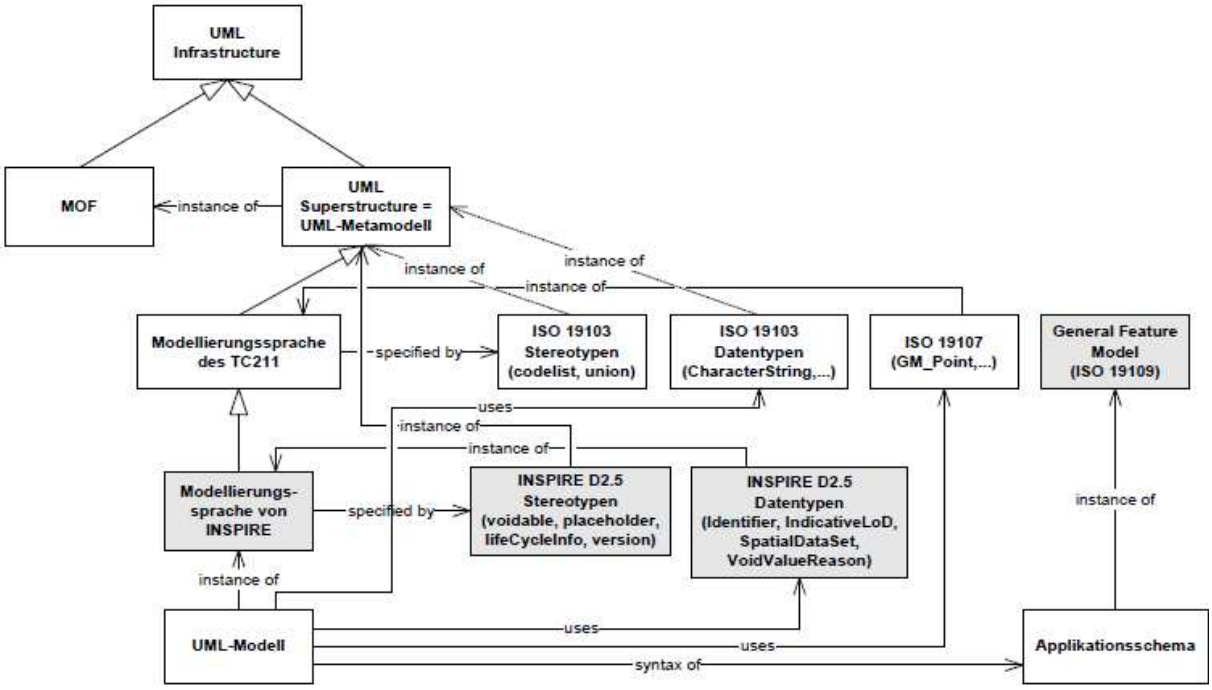

Figure 2 – Relation of INSPIRE UML profile to UML and ISO standards (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

3. INTERLIS

The goal of the thesis is to map one of the UML profiles explained in chapter 2 to a core profile. The UML profile we chose to put through the mapping process is the INTERLIS UML profile. Therefore, an introduction to INTERLS language is provided, as well as thorough descriptions of INTERLIS data types. This is important because one of the main parts of the transformation is to define how to map certain data types from one enviroment to another.

3.1.1. Introduction

"INTERLIS - A Data Exchange Mechanism for Land-Information-Systems" is a mechanism that consists of a conceptual description language and a transfer format which especially focuses on spatial data, thus enabling interoperability among various systems, as well as long-term availability, i.e. archiving and documentation of data.

INTERLIS is a standard dedicated to fulfillment of the requirements of modelling and the integration of geodata into contemporary and future geographic information systems. The usage of unified, documented spatial information and the flexible exchange possibilities results in the following advantages:

-the standardized documentation

-the compatible data exchange

-the comprehensive integration of geodata e.g. from different data owners.

-the quality proofing

-the long term data storage

-the contract-proof security and the availability of the software

(http://www.interlis.ch/general/historique_e.php)

3.2. INTERLIS 2 metamodel

This section describes the INTERLIS meta model in detail. It is composed of extracts from INTERLIS 2 – Metamodel document.

3.2.1. Introduction

The term of metamodel refers to the data model which describes the modeling of model descriptions. Data corresponding to a metamodel, and hence describing a data model, are called model data. As a basic principle the contents of model data are the same as those of model descriptions in INTERLIS language.

However their structure and coding do not aim at elegance of description and readability for human users but merely at their simplest possible application in programs, e.g.:

- Generic application programs whose functioning partially depends on model information.

- Generators of program parts according to model information.

- Tools converting model information in any way (e.g. into other standards).

(INTERLIS 2 – Metamodel, 2008.)

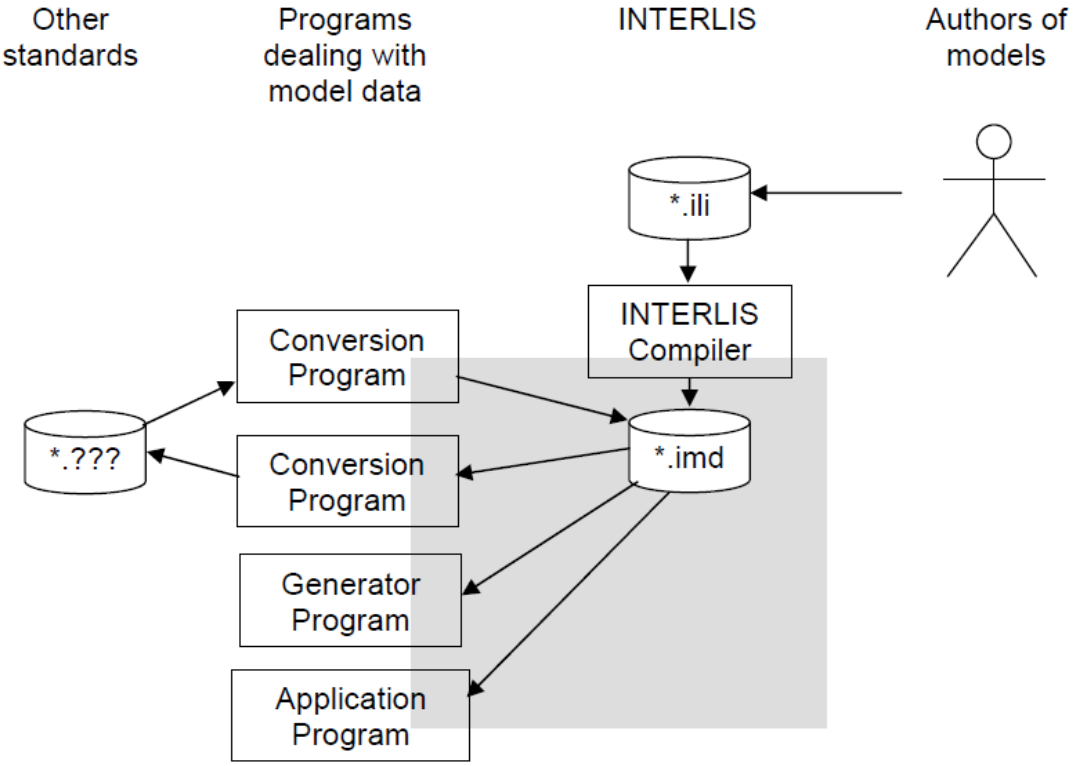

Figure 3 – Relevance of model data (INTERLIS 2 – Metamodel, 2008.)

3.2.2. Relationship with other standards

Metamodels and corresponding model data do not only exist for INTERLIS but also for other data description technologies (e.g. UML-Meta-Object Facility, XML-schema). Hence the question arised of whether the metamodel of such a technology should be adopted. This possibility was rejected on grounds of considerable differences which would have resulted in the loss of proximity to the INTERLIS-description language. Yet it is still possible to use other description technology: Based upon model data from other technologies it is possible to generate INTERLIS-model data and vice-versa. Such a transfer may also be added to the INTERLIS-specification. Furthermore it may appear convenient to

establish basic models for certain standards (e.g. GML), which would contain the basic structure of the external standard in INTERLIS. (INTERLIS 2 – Metamodel, 2008.)

3.2.3. Basic constructs

Figure 4 displays an overview of the INTERLIS-metamodel in the form of a UML class diagram.

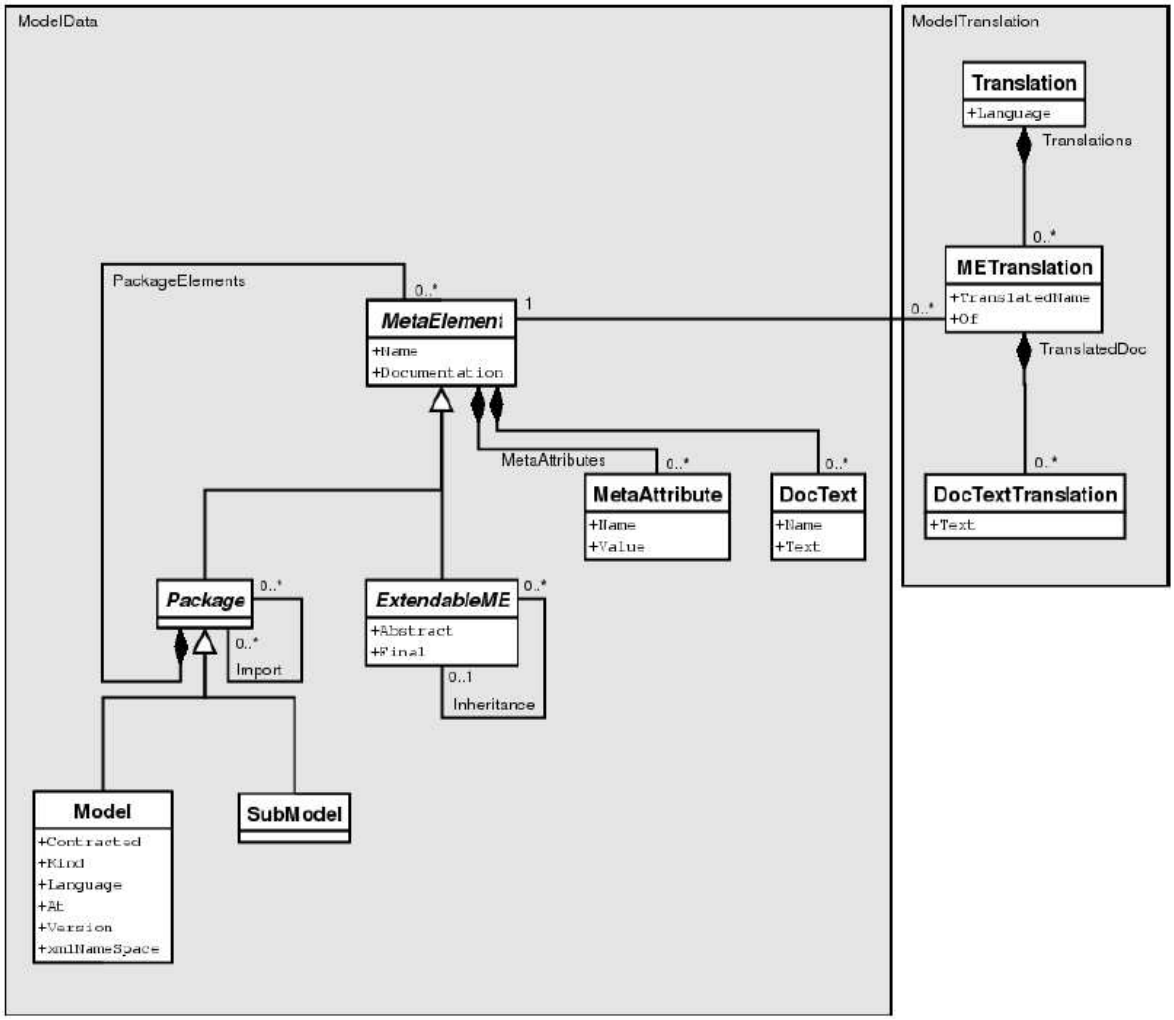

Figure 4 – Overview of the INTERLIS-metamodel (INTERLIS 2 – Metamodel, 2008.)

The abstract class MetaElement is the basic object class for all model data. Object identification of a metaelement is based upon a name path containing all names of superior metaelements plus, at the very end, its own name (attribute "Name"). Names are seperated by means of a period, thus guaranteeing stability of object identification throughout repeated compilation (even in the case of model alterations, provided they do not concern this particular metaelement.

Example: The internal INTERLIS-data model (cf. INTERLIS 2-Reference Manual, Appendice A) contains the structure UTC with the attribute Hours. Hence the metaelement of this attribute description will possess the OID "INTERLIS.UTC.Hours".

A metaelement will contain a (possibly empty) list of documentation texts (class "DocText"). Each documentation text can be specified by a name and contains the documenting text. INTERLIS does not govern the process of generating documentation texts, rather leaving this aspect to the tool generating model data.

It is possible to assign meta attributes to a metaelement by means of composition (class "MetaAttribut"). Neither the INTERLIS-language nor the metamodel provide a more precise definition of meta attributes. They mainly serve to add information beyond INTERLIS as a component of model data. Meta attributes have a name (attribute "Name"), which must be unique within the meta atributes of one metaelement, and a descriptive value (attribute "Value"). The object identification of meta attributes always consists of of the OID of the metaelement plus the permanent name METAOBJECT (a key word in INTERLIS 2) and its proper name. If, in the example above, a meta attribute by the name X, were defined, its OID would be INTERLIS.UTC.Hours.METAOBJECT.X". Meta attributes need not necessarily be stored in the same basket as the metaelement. Above all it is possible to define INTERLIS-baskets containing nothing but meta attributes of different metaelements. This is why, within the relationship MetaAttributes, the role MetaElement is specified as external. (INTERLIS 2 – Metamodel, 2008.)

3.2.4. Models and topics

In order to keep the INTERLIS-metamodel as similar as possible to the UML metamodel, models and topics in the metamodel appear slightly different from their definition in the INTERLIS-language.

Analoguous to UML it is the abstract class "Package" that sets the basis for all models and sub models. Via the relationship "PackageElements" a package may – with the exception of models – contain any number of other metaelements. Above all a sub model (class "SubModel") may also contain other sub models, even if the circumstance cannot be defined by means of the current language (INTERLIS 2.3). Via the relationship "Import", a package is linked to the imported packages.

By means of metaelements of the class "DataUnit" it is described how data units can be organized and how they depend on each other. All type designations (XML-tags) to be used during transfer can be derived from the "DataUnitName". The metamodel merely determines that data units must be defined within a package. However the following rules apply to the current language versions INTERLIS 2.3:

- An INTERLIS-model is represented by a metaelement of the class "Model". In the form of attributes it contains all information necessary according to the linguistic possibilities. In view of a possible XML transfer the attribute xmlNameSpace has been defined. As regards INTERLIS its value is http://www.interlis.ch/INTERLIS2.3 .

- A "Topic" of the INTERLIS-language is represented by one metaelement of each of the classes "SubModel" and "DataUnit". By means of the class SubModel a topic may be considered a package in the UML sense of the term;

within this package the metaelement "DataUnit" (Name: "BASKET", DataUnitName: Modelname"."Topicname) describes the structure and organization of corresponding INTERLIS-data baskets ("Basket"). (INTERLIS 2 – Metamodel, 2008.)

3.3. INTERLIS 2 data types

This section explains INTERLIS data types on a model level. Table 1 brings a summary of this section, but detailed comprehension is important for understanding of transformations, later described in chapter 5. Note that INTERLIS language rules won't be explained separately due to their intuitive nature, i.e. a reader familiar with the topic will be able to almost completely understand the meaning of the code. However, a reference manual with the language rules is enclosed on an attached CD.

All the data type descriptions in this chapter are extracts from the INTERLIS 2 Reference manual.

Table 1 : INTERLIS 2 data types (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

Data type	Explanation
TextType	Different types of text
EnumerationType	Different types of enumerations
EnumTreeValueType	Another type of enumeration
AlignmentType	Predefined list (for compatibility with INTERLIS1)
BooleanType	Predefined enumeration (true / false)
NumericType	Integers, floating point numbers, fixed point numbers (including min/max)
FormattedType	User-definable serialized data types
TimeType	Time including min/max
DateType	Date including min/max
DateTimeType	Date + time including min/max
OIDType	User configurable types of object identifiers
BlackboxType	Any XML fragments or binary objects
ClassType	Class name (primarily relevant for meta model)
AttributePathType	Attribute name (primarily relevant for meta model)
CoordiateType	Coordinates (with different consistency constraints)
PolylineType	Lines (with different consistency constraints)
SurfaceType	Individual panels (with different consistency constraints)
AreaType	Surface network (with different consistency constraints)

3.3.1. Strings

The term ‚string' depicts a series of symbols of maximum length. All the symbols supplied must be clearly defined.

Within the data type MTEXT, there are symbols 'carriage return' (#xD), 'line feed' (#xA) and 'Tabulator' (#x9), as opposed to the domain of the data type TEXT. With the data type string (TEXT), it is primarily the length of the string that is of interest. Depending on the form of definition, it is indicated explicitly or implicitly. In its explicit form (TEXT*…), the maximum length is determined in number of symbols (exceeding 0). If only the keywords TEXT or MTEXT are indicated, the number of symbols is unlimited. Within the scope of an extension its length can be shortened (lengthening would lead to a domain no longer compatible with the basic domain). An INTERLIS string length features the number of symbols as perceived by the user, but not the maximum number of memory storage spaces a system would need for the representation of such a string. Strings whose length is zero are considered to be undefined.

The name string type (keyword NAME) defines a domain that corresponds exactly to the one of INTERLIS-names. It is applied in the predefined class METAOBJECT and above all in the classes for reference systems, as well as symbols, because that is where data attributes have to coincide with description elements of models.

URI (Uniform Resource Identifier) represents a further string type, e.g. http-, ftp- and mailto-addresses. The length of a URI in INTERLIS is limited to 1023 symbols. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.2. Enumerations

By means of an enumeration, all values admissible for this type are determined. However an enumeration is not simply linear, but features a tree-like structure. The leaves of this tree (but not its knots) form the set of admissible values. Example (Figure 5):

```
DOMAIN
Colors = (red (dark_red, orange, crimson),
yellow,
green (light_green, dark_green));
```

Produces the following admissible values – described by means of constants:

#red.dark_red #red.orange #red.crimson #yellow #green.light_green #green.dark_green

A subdivision level is indicated in brackets (). The element names of each subdivision level must be unequivocal. You are at liberty to choose whatever tree depth seems convenient.

In an ordered enumeration (keyword ORDERED), the sequence of elements is clearly defined. If the enumeration is circular (keyword CIRCULAR), the sequence of elements is defined as if the enumeration were ordered. Moreover it is stated

that the last element will again be followed by the first. (INTERLIS Version 2 - Reference Manual, 2006.)

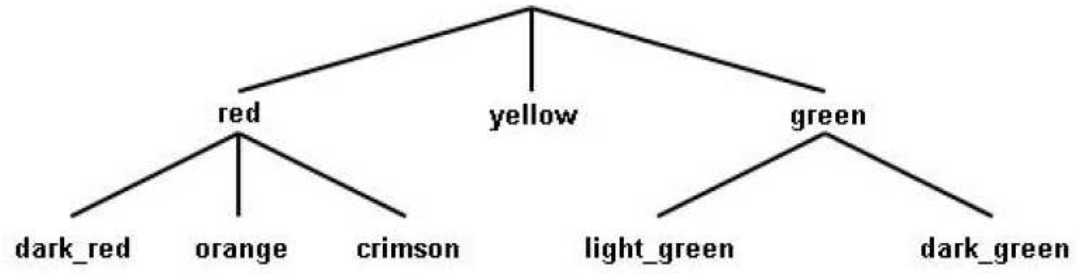

Figure 5 – Example of an enumeration (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.3. Text orientation

For the processing of plans and maps text positions must be determined. It has to be stated which point of the text the position corresponds to. The horizontal alignment determines whether the position is situated on the left hand or right hand margin or in the center of the text. The vertical alignment determines the position in the direction of the text size.

The distance between cap and base corresponds to the size of capital letters. Descenders are placed in the area base-bottom.

Horizontal and vertical alignment can be understood to signify the following, predefined enumerations:

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

Figure 6 depicts application of these rules in an example. (INTERLIS Version 2 - Reference Manual, 2006.)

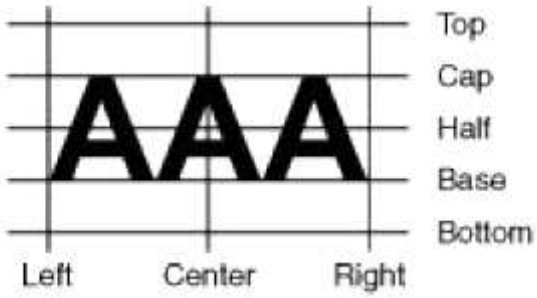

Figure 6 - Text orientation horizontally (HALIGNMENT) and vertically (VALIGNMENT) (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.4. Boolean

The type Boolean features the values true and false. It can be interpreted as the following predefined

enumeration:

```
DOMAIN
BOOLEAN (FINAL) = (false, true) ORDERED;
```

(INTERLIS Version 2 - Reference Manual, 2006.)

3.3.5. Numeric data types

The most important information in connection with numeric data types are the minimum and maximumvalue including number of decimals, as well as the scaling factor. Additionally it can be indicated that thetype is circular (keyword CIRCULAR), i.e. that in the last significant digit the maximum value increased by1 and the minimum value technically have the same meaning (e.g. with angles 0 .. 359 degrees). If the attribute has been defined as a continuous subdivision of the predecessor, the type has to be defined as circular. If the indication of minimum and maximum value should be missing (keyword NUMERIC), the domain is regarded as abstract.

```
DOMAIN
Angle1 = 0.00 .. 359.99 CIRCULAR [degree]; !! correct
Angle2 = 0.00 .. 360.00 CIRCULAR [degree]; !! syntactically
                     !!correct, but technically false, since
                     !!360.01 subsequently corresponds to the
                     !! minimal value 0.00
```

The number of digits must coincide in minimal and maximum value. By means of scaling it is possible to describe float-numbers, but then the minimum as well as the maximum value have to be indicated in mantissa type number representation, i.e. starting with zero (0) and followed by a decimal point (.) the first digit after the decimal point must differ from zero (0). The scaling of the minimum value must be inferior to the scaling of the maximum value. However the notation of minimum

and maximum value in no way qualifies as an indication on how to transfer these values (if the domain is defined with 000 .. 999 this does not mean that the value 7 is transferred as 007). Float-numbers are the exception to this rule. These are to be transferred in mantissa type number representation and with scaling. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.6. Formatted domains

Formatted domains are based on structures and use their numeric and formatted attributes within one format. On the one hand this format serves the data exchange, on the other hand the definition of both lower and upper limit of the domain. Primarily a basic definition of a formatted domain defines the structure it is built upon and the format which is being used. In addition it is possible to define both lower and upper limit of the domain. They may not extend the limits defined by the structure.

Within the scope of an extension it is possible to refer to an extension of the original structure, to supplement the format (the inherited part must figure at the beginning and in the interest of clearness it should be signalized by the keyword INHERITANCE) and to restrict the domain. On the one hand the format definition may contain constant strings, which do not start with a number (at the start, at the end or in between the individual attribute references), on the other it may contain direct or indirect (via structure attributes) attribute references. The attribute reference must either designate a numeric attribute or a structure attribute. In the case of a numeric attribute it is possible to define the number of digits on the left of the decimal point. As a result leading noughts will be introduced if necessary. The number of decimals will result from the numeric domain. With structure attributes you must define in accordance with which formatted domain they have to be formatted. The structure must either tally with the basic structure of the domain or be an extension of it. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.7. Date and time

Whenever indications of date or time do not only consist of one single value (e.g. year, second), we tend to use formatted domains.

In the interest of compatibility with XML corresponding elements are predefined by INTERLIS:

```
UNIT
Minute [min] = 60 [INTERLIS.s];
Hour [h] = 60 [min];
Day [d] = 24 [h];
Month [M] EXTENDS INTERLIS.TIME;
Year [Y] EXTENDS INTERLIS.TIME;
REFSYSTEM BASKET BaseTimeSystems ~ TIMESYSTEMS
OBJECTS OF CALENDAR: GregorianCalendar
OBJECTS OF TIMEOFDAYSYS: UTC;
```

```
STRUCTURE TimeOfDay (ABSTRACT) =
Hours: 0 .. 23 CIRCULAR [h];
CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR
[INTERLIS.s];
END TimeOfDay;
STRUCTURE UTC EXTENDS TimeOfDay =
Hours(EXTENDED): 0 .. 23 {UTC};
END UTC;
DOMAIN
GregorianYear = 1582 .. 2999 [Y] {GregorianCalendar};
STRUCTURE GregorianDate =
Year: GregorianYear;
SUBDIVISION Month: 1 .. 12 [M];
SUBDIVISION Day: 1 .. 31 [d];
END GregorianDate;
STRUCTURE GregorianDateTime EXTENDS GregorianDate =
SUBDIVISION Hours: 0 .. 23 CIRCULAR [h] {UTC};
CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR
[INTERLIS.s];
END GregorianDate;
DOMAIN XMLTime = FORMAT BASED ON UTC ( Hours/2 ":" Minutes ":"
Seconds );
DOMAIN XMLDate = FORMAT BASED ON GregorianDate ( Year "-" Month "-
" Day );
DOMAIN XMLDateTime EXTENDS XMLDate = FORMAT BASED ON
GregorianDateTime
( INHERITANCE "T" Hours/2 ":" Minutes ":" Seconds );
```

(INTERLIS Version 2 - Reference Manual, 2006.)

3.3.8. Coordinates

Coordinates can be defined one, two or three dimensionally, and hence are either a single digit, double digits or triple digits. It is permissible to add the second or third dimension only in an extension. For every dimension the numeric domain, as well as perhaps a measure unit and a coordinate system (including numbers of axis) must be indicated. Only concrete measure units can be indicated. If no reference system is indicated and if furthermore measure units are not or as length units, the program system that implements the model may presume that it is dealing with Cartesian coordinates.

Whenever a rotation definition occurs (keyword ROTATION), it is possible within the scope of definitions of zero directions to refer to such a coordinate reference system. The rotation definition determines which axis of the coordinate domain corresponds to the zero direction and which to the direction of a positive right angle. It may be missing in a concrete coordinate definition and could be added in an extension.

Any indication concerning definition of axis and rotation cannot be changed in extensions.

```
DOMAIN
CHCoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
60000.00 .. 320000.00 [m] {CHLV03[2]},
ROTATION 2 -> 1;
```

In both defined axes the admissible domain is indicated as well as the units and reference system including the number of axis the coordinates refer to. The actual axes are defined within the reference system. The rotation definition determines that the zero direction leads from axis #2 to axis #1, in other words in the Swiss Federal system where value #1 corresponds to east, and value #2 to north, the zero direction shows north and turns clockwise.

```
DOMAIN
WGS84Coord = COORD –90.00000 .. 90.00000 [Units.Angle_DMS]
{WGS84[1]},
0.00000 .. 359.99999 CIRCULAR [Units.Angle_DMS] {WGS84[2]},
-1000.00 .. 9000.00 [m] {WGS84Alt[1]};
```

Typically geographic coordinates are represented in degrees and refer to an ellipsoid coordinate system (e.g. CH1901). Then again the altitude is described in meters. It refers to a special ellipse height system with one axis. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.9. Domains of object identifications

Identifiable objects are always labeled with an object identification. In order to make it clear to the system what storage has to be supplied and how these object identifications have to be generated, corresponding domains can be defined and assigned to topics resp. classes. For the administration of object identifications, mainly also of baskets, it makes sense to have ordinary attributes with such domains.

INTERLIS 2 itself defines the following OID-domains:

```
DOMAIN
ANYOID = OID ANY;
I32OID = OID 0 .. 2147483647; !! positive integer values needing 4
Bytes memory
STANDARDOID = OID TEXT*16!! (only numbers and letters are
permitted)
UUIDOID = OID TEXT*36; !! according to ISO 11578
```

If ANYOID is used for abstract topics, resp. classes, it is required to expect an object identification whose exact definition however is to remain open. Otherwise ANYOID can only be used as an attribute domain. Consequently it is not only the OID itself that belongs to the attribute value, but also the concrete OID domain.

OID values of textual OID domains must comply with the rules of the XML-ID-type: the first symbol must be a letter or underscore, followed by letters, numbers, dots, minus sign, underscore; no colons. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.10. Blackboxes

By using this data type it is possible to model attributes whose contents cannot be specified. The XMLversion describes an attribute with XML-content and the BINARY version a binary content. This type cannot be refined in extensions. (INTERLIS Version 2 - Reference Manual, 2006.)

3.3.11. Domains of classes and attribute paths

It may make sense that data objects contain references to certain classes and attributes. By indicating structure any structure or class, by indicating class (also permissible as extension of STRUCTURE) any class (but no structures) are admitted. If only certain structures, resp. classes and their extensions are to be admitted, these have to be listed (RESTRICTION). In extensions all admissible structures, resp. classes have to be listed again. They cannot contradict the basic definition. As soon as such restrictions have been defined, STRUCTURE no longer can be extended by CLASS. By means of indicating ATTRIBUTE a certain attribute path type is admitted. It may be stated that it should belong to a class (not a subclass!) in accordance with another definition (OF). It is possible to refer either to a ClassType-Attribute or as in the case of the definition of a function to a different argument. Furthermore all possible attribute types can be restricted (RESTRICTION). The following are suitable as constants: names of the attributes of classes, structures, associations and views. The corresponding class name can be stated explicitly or is derived from the context resp. from thereference to another attribute or another argument (OF). (INTERLIS Version 2 - Reference Manual, 2006.)

3.4. Line strings

3.4.1. Geometry of the line string

Practically a curve segment is a 1-dimensional structure which has no splits, no corners and no double points of any type (see figures 7 and 8). Curve segments are smooth and unique. Straight line segments, circle arcs, segments of parabolas and clothoides are examples of curve segments. Every curve segment has two boundary points (start point and end point, which are not allowed to be identical. The other points of a curve segment are called inner points. These form the interior of the curve segment.

Figure 7 – Examples of planar curve segments (INTERLIS Version 2 - Reference Manual, 2006.)

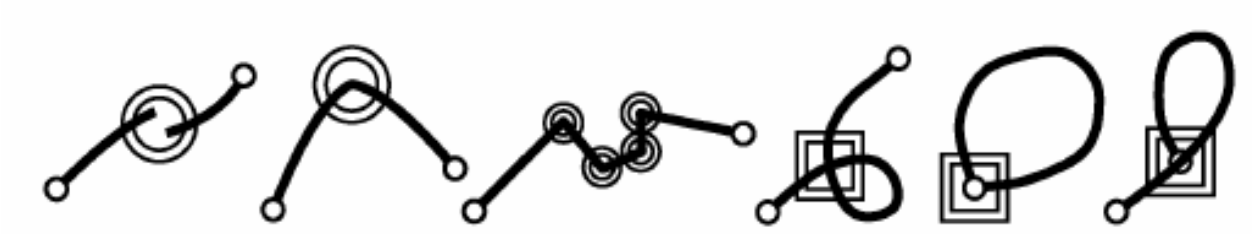

Figure 8 – Examples of planar sets not being curve segments (a double circle indicates „not smooth“ and a double square „not injective“) (INTERLIS Version 2 - Reference Manual, 2006.)

A line string is a finite sequence of curve segments. Except for the first curve segment, the start point of every curve segment corresponds to the end point of the preceding curve segment. These points are called control points of the line string. Practically a line string can have multiply used curve segments, curve segments with common base points, intersecting curve segments and curve segments starting or ending in the interior of other curve segments (see figures 9 and 10). A simple line string contains no self-intersection points (see figure 10). In addition, for a closed simple line the start point of the first curve segment and the end point of the last curve segment are identical. (INTERLIS Version 2 - Reference Manual, 2006.)

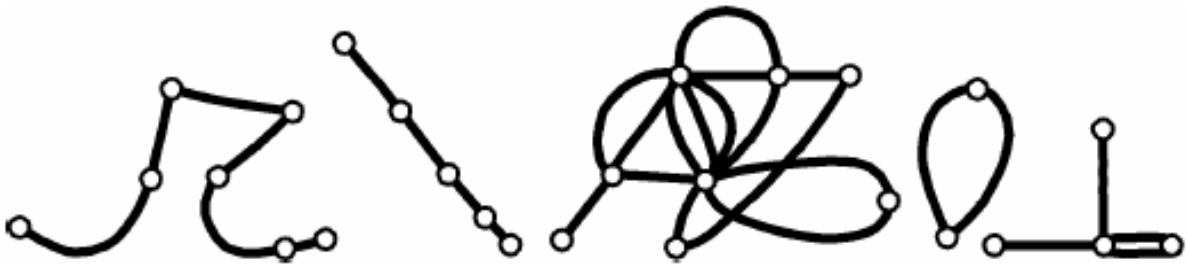

Figure 9 – Examples of planar line strings (INTERLIS Version 2 - Reference Manual, 2006.)

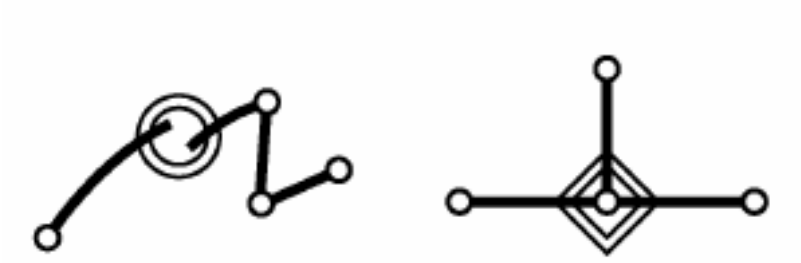

Figure 10 - Examples of planar sets that are not line strings (the double circle means "not continuous"and the double rhombus "not image of an interval") (INTERLIS Version 2 - Reference Manual, 2006.)

Figure 11 - Examples of (planar) simple line strings (INTERLIS Version 2 - Reference Manual, 2006.)

3.4.2. Line strings with straight line segments and circle arcs as predefined curve segments

Line strings exist as directed (DIRECTED POLYLINE) or undirected (POLYLINE) line strings, and may as well be used within the scope of surfaces and tessellations. The definition of a concrete value domain of a line string always requires the specification of the admissible forms of curve segments by means of enumeration, e.g. straight line segments (keyword STRAIGHTS), circle arcs (keyword ARCS) or other possibilities and furthermore the indication of the domain of the vertices. Within an abstract value domain of a line string these specifications may be omitted. The following rules apply for domain extensions:

- A line can only be reduced but not completed by new types.

- The coordinate domain indicated within the scope of a line string value domain must be a restriction of the coordinate domain of the original line string value domain, provided the latter has been defined.

Curve segments are always considered an extension of the basic structure 'LineSegment'. The coordinate domain applied therein is the same as the one defined in the definition of the line string.

```
STRUCTURE LineSegment (ABSTRACT) =
SegmentEndPoint: MANDATORY LineCoord;
END LineSegment;
STRUCTURE StartSegment (FINAL) EXTENDS LineSegment =
END StartSegment;
STRUCTURE StraightSegment (FINAL) EXTENDS LineSegment =
END StraightSegment;
STRUCTURE ArcSegment (FINAL) EXTENDS LineSegment =
  ArcPoint: MANDATORY LineCoord;
  Radius: NUMERIC [LENGTH];
END ArcSegment;
```

The first curve segment of a line string is always a start segment. The start segment only consists of the start point itself, which at the same time is the end point of the start segment. The straight-line segment has an end point and thereby determines a straight from the end point of the predecessor curve segment to its end point. Both start segment and straight line segment do not require any further specifications. Thus the corresponding extensions of the LineSegment are void. Two successive vertices (SegmentEndPoints) may not coincide in the projection.

A circle arc segment describes a curve segment that appears as a true circle arc segment in the projection. In addition to the end point an intermediate point describes the circle arc segment. It is only of significance in connection with the geometry. With 3-dimensional coordinates we use linear interpolation for the height on the circle arc segment. You may imagine this curve as the thread of a cylindrical screw in perpendicular position on the projection plane. The intermediate point is not a vertex of the line string. It should be positioned as exactly as possible in the middle between start and end point. Since the intermediate point is indicated with the same precision as the vertices, the calculated and the effective radius may differ widely. Whenever the effective radius is indicated it is relevant for the definition of the arc circle. Then the intermediate point will only determine which of the four possible circle arcs is the one desired. However even in this case the intermediate point may only differ by 2 units from the trace of the circle arc calculated from the radius. It can be required that a string line must be a simple string line, i.e. practically that it neither intersects itself and above all that multiple use of the same curve segment is impossible (keyword WITHOUT OVERLAPS). If a circle arc and a straight line (resp. a second circle arc) as successive curve segments of a line string not only have a common vertex but also a common inner point (definition see above), then

this is also permitted in the case of a simple string line, provided that the circle segment detached from the straight (resp. the double circle segment detached from the other circle arc) have a height of arrow smaller or equal the decimal specified after WITHOUT OVERLAPS> (see figure 12a). There are two reasons for this regulation: On one hand for numeric reasons and because in certain cases small overlaps in arcs cannot be avoided (tangential arcs). On the other hand, when transferring data that originally has been registered in graphics even more important overlaps (e.g. of several centimeters) have to be tolerated, unless one would be prepared to face an enormous workload to repair these overlaps. Tolerances have to be listed in the same units as vertex coordinates. For numeric reasons it must be greater than nought. They cannot be overridden and are mandatory in the case of both surfaces and tessellations.

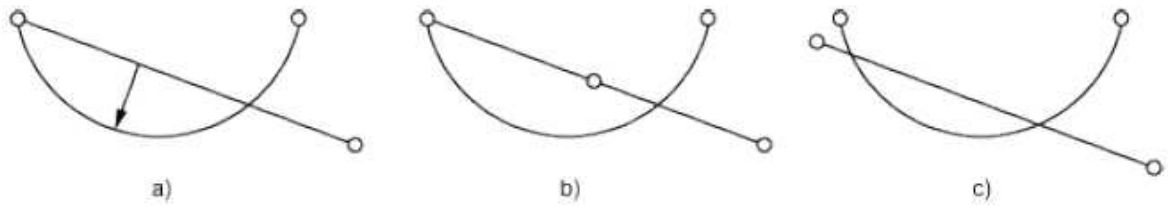

Figure 12 - a) Height parameter (of arrow) may not exceed the given tolerance; b) inadmissible verlap of polylines since another vertex is situated between vertex and intersection; c) inadmissible verlap of polylines since there exists no common vertex (INTERLIS Version 2 - Reference Manual, 2006.)

Within the scope of value domain definitions and attribute extensions, undirected string lines can beextended into directed string lines. When line strings are directed, their direction always has to be conserved (even when transferring data). For vertices the value domain of the coordinates is defined. By means of the

existence constraint REQUIRED IN it is further possible to demand that coordinates may not be arbitrary but have to correspond to the points of certain classes. If the coordinate type of the vertices is abstract, then the line string has to be declared abstract as well. In order to be able to assign different attributes to different segments of the boundary where surfaces are concerned, it is possible to define further attributes for the actual line string objects (so-called line attributes, rule LineAttrDef only with SURFACE and AREA). However this does not result in the concept of a clearly defined division of the boundary. From the conceptual point of view it is insignificant whether subsequent curve segments with equal attribute values are considered as individual line string objects or as one whole line. The structure employed for the definition may only feature local attributes and function calls. When a surface or tessellation attribute for which a line attribute has been defined by means of structure, is extended, then the extended attribute must either not feature a line attribute or its structure must be abstract extension of the basic structure. With function calls the resulting type must be a local attribute. (INTERLIS Version 2 - Reference Manual, 2006.)

3.4.3. Other forms of curve segments

Besides straight line segments and circle arcs it is possible to define other forms of curve segments. Besides their names it also has to be specified according to which structure a curve segment is described. These definitions of such curve segments are only admissible within the scope of a contract, since we may not assume that a system supports any form of curves. A line structure must always be an extension of the LineSegment as defined by INTERLIS. (INTERLIS Version 2 - Reference Manual, 2006.)

3.5. Surfaces and tesselations

3.5.1. Geometry of surfaces

In most cases planar surfaces are sufficient for the modeling of geo-data. In addition INTERLIS also supports planar general surfaces. Practically a planar general surface is limited by one exterior and possibly by one or several interior boundaries (see figure 17). The boundary lines themselves must consist of simple line strings that from a geometrical point of view can be combined into closed simple line strings. Furthermore they must be positioned in such a manner that from any point in the interior of the surface to any other point in the interior of the surface there exists a way that neither intersects a boundary line nor contains vertices of a boundary line (see figure 16). As long as this restriction is not violated, boundaries may meet in vertices. In such situations there are several conceivable possibilities that would allow dividing the boundary of the surface as a whole into individual line strings (see figure 19). INTERLIS does not insist on one specific possibility. If such a surface is transferred several times, a different possibility may occur in each different transfer. (INTERLIS Version 2 - Reference Manual, 2006.)

Figure 13 - Examples of surface elements (INTERLIS Version 2 - Reference Manual, 2006.)

Figure 14 - Examples of point sets in space, which are not surface elements (here a double circle means "not smooth") (INTERLIS Version 2 - Reference Manual, 2006.)

Figure 15 - Examples of surfaces in the space (INTERLIS Version 2 - Reference Manual, 2006.)

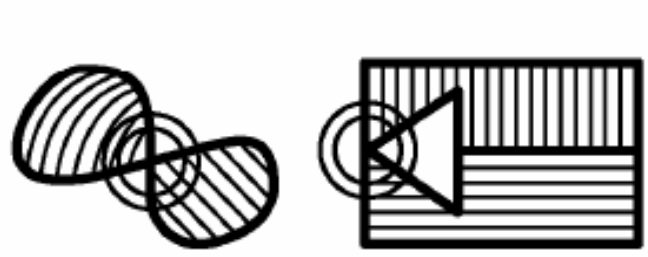

Figure 16 - Examples of planar point sets that are not surfaces (a double circle marks a "singular point") (INTERLIS Version 2 - Reference Manual, 2006.)

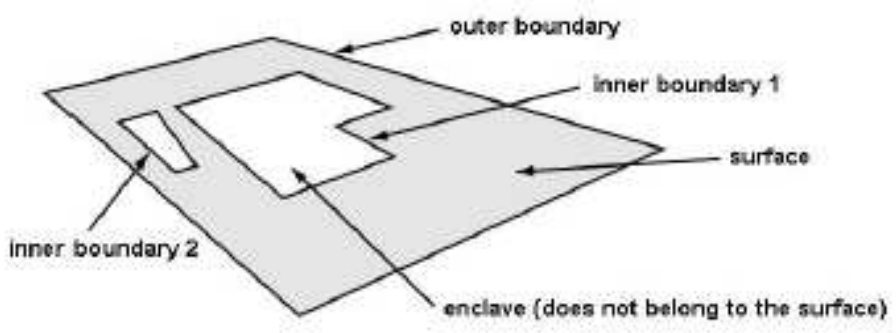

Figure 17 - Planar surface with boundaries and enclaves (INTERLIS Version 2 - Reference Manual, 2006.)

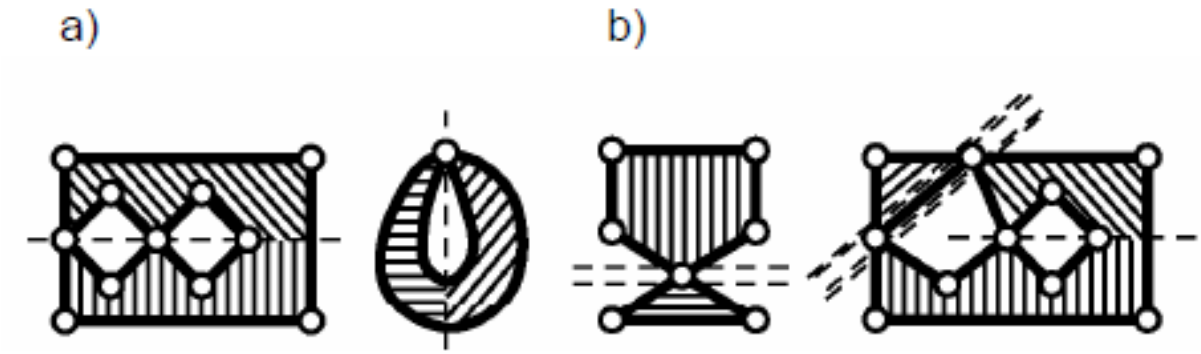

Figure 18 - a) Examples of planar general surfaces; b) Examples of planar sets that are not general surfaces, because their interior is not connected. But these planar sets can be subdivided into general surfaces ("---" shows the subdivision into surface elements and "===" the subdivision into general surfaces) (INTERLIS Version 2 - Reference Manual, 2006.)

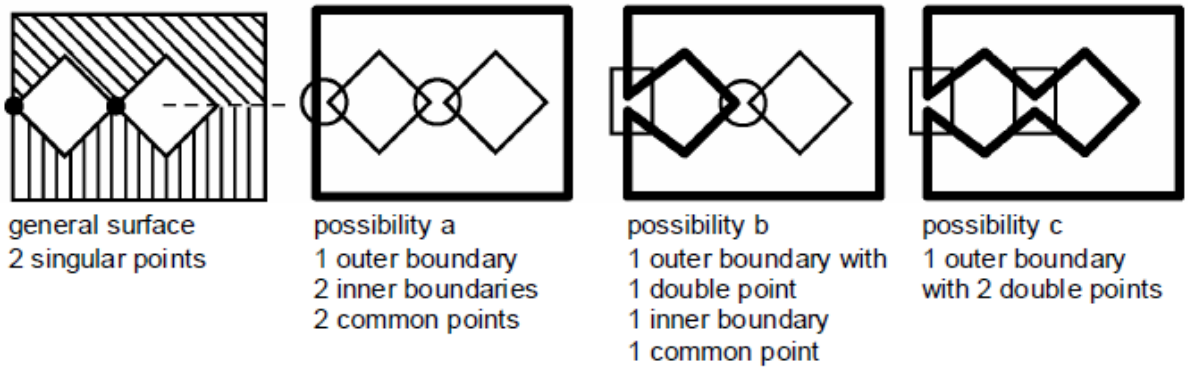

Figure 19 - Different possible subdivisions of the boundary of a general surface (INTERLIS Version 2 - Reference Manual, 2006.)

Along with the definition of (general) surfaces, resp. (general) surfaces of a tessellation we determine beyond which tolerance curve segments of the boundary may not overlap (for all concrete definitions of surfaces and tessellations, WITHOUT OVERLAPS must either be directly specified or inherited). With surfaces prohibition of overlaps, resp. intersection does only apply to curve segments of one individual line string but to all curve segments of all line strings of the surface boundary. In the case of surfaces of a tessellation it even applies to all line strings connected with the tessellation. Furthermore and in accordance with the definition of the (general) surface, line strings that are not part of the boundary of a (general) surface are excluded. (INTERLIS Version 2 - Reference Manual, 2006.)

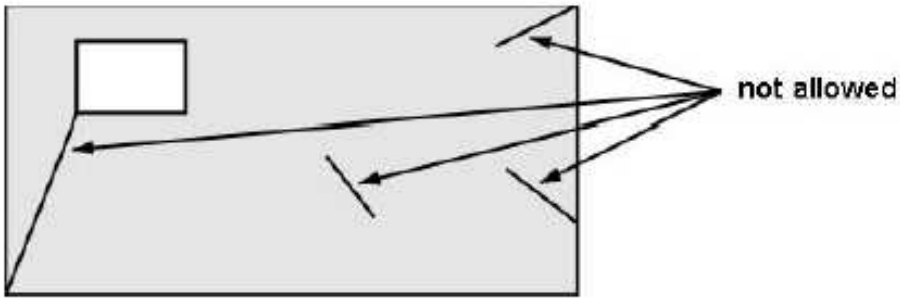

Figure 20 - Disallowed boundary configurations for tessellations (INTERLIS Version 2 - Reference Manual, 2006.)

3.5.2. Surfaces

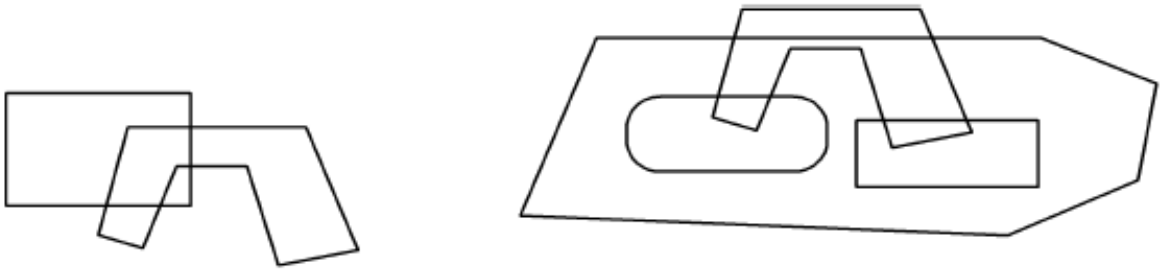

Figure 21 - Individual surfaces (SURFACE) (INTERLIS Version 2 - Reference Manual, 2006.)

For surfaces that partially or entirely overlap, i.e. which do not only have boundary points in common, the geometrical attribute type SURFACE is available (see figure 21). This type is called surface. A surface consists of one outer and possibly several inner boundaries (around the enclaves). Each boundary consists of at least one line string. Besides its geometry each line string features the defined attributes (rule LineAttrDef).

3.5.3. Surfaces of a tesselations

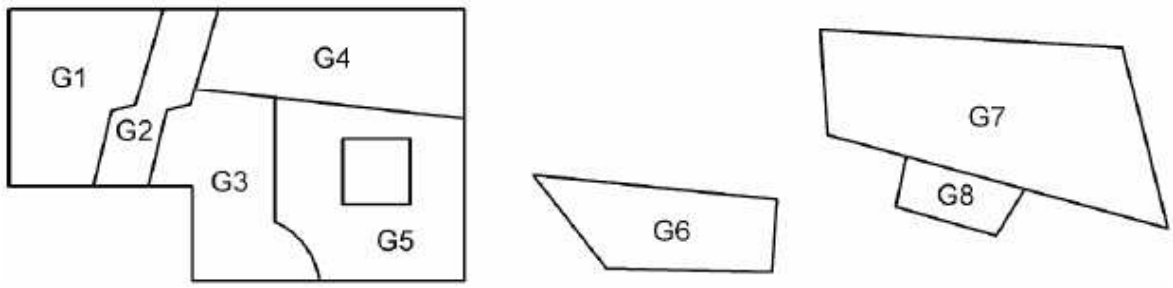

Figure 22 - Tessellation (AREA) (INTERLIS Version 2 - Reference Manual, 2006.)

(Planar) tessellation signifies a finite set of (general) surfaces and environments that cover a layer without overlaps. For tessellations the geometrical attribute type AREA is at your disposal. A maximum of one surface of the tessellation (or exactly one with the additional keyword MANDATORY), but never its environment, is assigned to the area object. It is not admissible that each of two surfaces of the tessellation with a common boundary does not correspond to an area object. Thus each individual area object corresponds to a surface. As a result the same implicit structure applies to surfaces and to area objects. However additional consistency constraints apply:

- String lines of a tessellation must always be true boundaries. Hence there are no line strings with the same surface on either side (see figure 22). This is also ruled out by the definition of a surface.

- If there are defined area objects on either side of a line string, then each curve segment (join between two vertices) of one area object must in terms of geometry and attributes correspond exactly to the curve segment of the other area object.

- If line attributes have been defined for these lines, then they must feature the same values for coinciding lines of the two neighbouring areas.

Tessellations may not occur within substructures. In order to be able to refer to line strings of a tessellation as individual objects (and namely as one object even if the line string is the boundary of two area objects), then AREA INSPECTION is at your disposal. (INTERLIS Version 2 - Reference Manual, 2006.)

3.5.4. Extensibility

Surfaces may be extended into areas. The extension of a string line into an area is inadmissible, since with a surface several string lines have to be expected, whereas the definition of the line string only implies one line string. Independent surfaces and surfaces of a tessellation can be extended in two respects:

- When it is 'SURFACE' that is primarily defined and hence overlaps are allowed, this can be replaced by 'AREA' in extensions, since this will not violate the basic definition.

- Further line attributes can be attached.

(INTERLIS Version 2 - Reference Manual, 2006.)

3.6. INTERLIS 2 stereotypes

There is a number of stereotypes in INTERLIS 2 language. The complete list of these stereotypes can be found in Table 2. However, not all of them will be discussed in detail in this thesis because there is no need in defining a mapping rule for them. This is because GML has its own elements that have the same function as some of these stereotypes. Also, GML does not deal with the means of displaying the data, so all the stereotypes that allow the user to define views, graphic rules and drawing rules etc. are unmappable.

This section brings a description of two main stereotypes of INTERLIS 2 language, ModelDef and TopicDef, extracted from the INTERLIS 2 Reference manual.

Table 2 : INTERLIS 2 stereotypes (Kutzner, Eisenhut 2010. Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE)

Stereotype	Explanation
ModelDef	Definition of an application schema (may only include UnitDef or GraphicDef)
TopicDef	Definition of a data container
MetaDataBasketDef	Definition of CRS for use in CoordinateType and of symbols for use in GraphicDef
UnitDef	Definition of units
FunctionDef	Definition of functions for use in the consistency conditions, ViewDef or GraphicDef
ViewDef	Data conversion (as the basis for complex consistency conditions, calculated attributes / associations or GraphicDef)
GraphicDef	Presentation description
DrawingRule	Part of a GraphicDef
LineFormTypeDef	Definition of new forms of a curve
RunTimeParameterDef	Definition of data from the runtime system, such as current date for use in the consistency conditions

3.6.1. Stereotype ModelDef

The term 'model' means a self-contained, complete definition. According to the type of model it may contain several constructions.

A pure type model (TYPE MODEL) may only declare measure units, domains, functions and types of lines.

A reference-system model (REFSYSTEM MODEL) should declare definitions of a type model, as well as topics and classes related to the extensions of the predefined classes AXIS, resp. REFSYSTEM. The language cannot reinforce the observation of this rule. It is up to the user to abide by it.

A symbology model (SYMBOLOGY MODEL) should declare definitions of a type model, as well as topics and classes related to the extensions of the predefined class SIGN, and furthermore run time parameters. It is up to the user to observe this rule. Symbology models can only be permitted in connection with contracts, as they have to be adapted to their treatment by the system.

Where no restrictive model properties are defined, a model may contain any conceivable construct. Following the model name the language can be indicated (optional). Whenever possible this should be done according to ISO-Norm 639 , i.e. in two letters and using small letters (see www.iso.ch/), e.g. "de" stands for German, "fr" for French, "it" for Italian, "en" for English. According to ISO-Norm 3166 a country code can be added separated by an under line to indicate a variety of language used in a specific country: "de_CH" stands for the written High German used in Switzerland. This indication is of documentary value, in connection with the possibility to declare one model translation of another (TRANSLATION OF). In their structure both models must be exactly identical, hence they can only differ in the names employed. However the declaration 'translation' is not tied to the indication of language. For example in order to support local use of language or particular trade vocabulary it is admissible to add translations in the original language. Following this the author of the model will be identified by indicating the corresponding URI. We expect the model name to be unequivocal within this context.

By means of this model version we are able to distinguish between different versions (above all different levels of development) of a model. In the comments it is possible to add further information such as remarks concerning compatibility with earlier versions. Nevertheless at a certain moment in time there should only exist one model version. That is why no version will be indicated when importing models. If one model is a translation of another, its version has to be indicated in brackets. Hence such an indication of version within the scope of a translation definition (TRANSLATION OF) will only demonstrate which basic version was used as a base for this translation, i.e. which basic version will have exactly the same structures.

Whenever a model uses language elements demanding a contract, this model has to be signalized specifically (keyword CONTRACTED).

Whenever an INTERLIS construction refers to a definition stated in another model, this model has to be imported (keyword IMPORTS). Thus topics can be extended and references to classes created. IMPORTS only offers the possibility of use. When using imported definitions they still have to be referred to with a qualified name (model, topic), unless the keyword UNQUALIFIED is applied. Topics will only belong to a, if they have been defined within this model (rule TopicDef).

A pre-defined base model "INTERLIS is connected to the language. It need not be imported. Nevertheless its elements are only available with unqualified names, if the model is introduced with IMPORTS UNQUALIFIED INTERLIS. (INTERLIS Version 2 - Reference Manual, 2006.)

3.6.2. Stereotype TopicDef

A topic (keyword TOPIC) contains all definitions necessary to describe a specific part of reality. A topic can also define types such as measure units, domains or structures or it may use those belonging to the enveloping model or an imported model. Put in parenthesis ( ) properties of inheritance can be defined. Since an extension of a topic always refers to a topic of a different name, EXTENDED would not make sense and hence is not admissible. Concerning a certain topic, which contains concrete classes, there may exist an indefinite number of baskets (databases etc.) They all possess a structure corresponding to the topic but contain different objects. A data basket only features instances of classes (and their sub-structures). If a topic contains graphic descriptions, three types of baskets can be set up.

- Data baskets.

- Baskets with basic data for graphics. Such baskets contain the instances of classes or views which lay the foundation for graphic descriptions.

- Graphic baskets. These baskets contain graphic objects that have been realized (according to the parameter definition of the symbols employed.

As a rule baskets and objects feature an object identification. Their domains result from the corresponding definition: BASKET OID AS for the object identifications of any basket, OID AS for the object identifications of any object, provided no specific definition was made with the corresponding class. Once an OID-domain has been assigned to a topic, this can no longer be modified in extensions. In many

cases it will make sense to use the standard domain STANDARDOID. Definitions concerning object identifications (BASKET OID AS, OID AS) are only admissible within the scope of contracts. If there is no OID definition for a topic or a certain class, no stable object identification will be provided for these baskets resp. objects and consequently no incremental data exchange will be possible. With exception of specific indications, one topic is, where data are concerned, independent of other topics. Data-related dependencies arise as a consequence of relationships, resp. reference attributes, which refer to a different basket, through special constraints or by means of construction of views or graphic-definitions on

classes or other views, but not as a consequence of the use of meta objects. In the interest of clear recognizability of such dependencies these must be explicitly declared already in topic heading (keyword DEPENDS ON). Detailed definitions (e.g. definitions of relationships) may not violate declarations of dependencies. Cyclic dependencies are inadmissible. Without further declaration an extended topic features the same dependencies as its base-topic. (INTERLIS Version 2 - Reference Manual, 2006.)

4. Geography Markup Language

The core profile we chose for this research is the ISO 19136 specification, which describes Geography Markup Language (GML). However, we want to look at the specification as if it would be a UML profile. Section 3.2 gives an insight in a transformation from GML to UML, which is also decribed by ISO 19136 specification.

4.1. Introduction

Geography Markup Language is an XML grammar written in XML Schema for development of application schemas with focus on geographic information, as well as the transport and storage of spatial data.

The key concepts used by Geography Markup Language for modeling the world are drawn from the ISO 19100 series of International Standards.

A feature is an abstraction of real world phenomena. A geographic feature is associated with a location relative to the Earth. A digital representation of the real world may be thought of as a set of features. Set of properties describes a state of feature, where each property has its name, type and value.

Type definition determines what kind of properties will a feature have. Geographic features with geometry have properties whose type is a geometry. A feature collection is a collection of features that may itself be treated as a feature; this means that a feature collection has a feature type and thus may have distinct properties of its own, in addition to the features it contains.

GML provides means of selecting or defining reference systems. A reference system defines a connection between values of properties and a scale of measurements, usually representing real world. A coordinate reference system is a reference system which enables us to describe position of spatial data. It consists of a set of coordinate system axes that are related to the Earth through a datum that defines the size and shape of the Earth.

A temporal reference system provides standard units for describing time and temporal length or duration.

A reference system dictionary provides definitions of pre-defined reference systems used for describing spatial or temporal properties of a feature.

A dictionary of units of measurements provides definitions of numerical measures of physical quantities, such as angle, length and temperature. It also provides definition of conversions between units. (OpenGIS® Geography Markup Language (GML) Encoding Standard, 2007.)

4.2. Mapping a GML schema to UML

Even though GML is selected as a core UML profile for this thesis, it is better to create an application schema directly in XSD schema. This is because of the thorough descriptions of encoding rules in XSD schema that are supplied in the

GML encoding standards. Another reason for this is that a study that deals with translation of INTERLIS language to a GML enviroment already exists.

However, it is important to know that GML encoding standard describes a fixed set of rules that allow direct translation of a GML application schema, created in an XSD schema, to the UML form. Next sections bring a small insight to the most basic transformation methods.

4.2.1. Encoding rules

„The schema encoding rules are based on the general idea that the corresponding type and element declarations in XML Schema are mapped to class definitions in the UML application schema, so that element structures in the XML document can be mapped to the objects in the instance model.“ (OpenGIS® Geography Markup Language (GML) Encoding Standard, 2007.)

Table 3 gives an overview of the mapping rules. The UML elements placed between „ «» „ symbols are UML stereotypes.

Table 3 : Schema encoding overview (OpenGIS® Geography Markup Language (GML) Encoding Standard, 2007.)

Table: GML → UML Application Schema Overview	
GML application schema	**UML application schema**
GML application schema	Package <<ApplicationSchema>>
GML schema document {name} XSD	Package named {name}
Object and property type and global element for any object type that is a direct or indirect extension of `gml:AbstractFeatureType`	Class with stereotype <<FeatureType>>
Object and property type and global element for any object type that is a direct or indirect extension of `gml:AbstractGMLType`, other than those that extend `gml:AbstractFeatureType`	Class with no stereotype
Data and property type and global element for any object type that is not a direct or indirect extension of `gml:AbstractGMLType` and whose content model is a sequence of properties	Class with stereotype <<DataType>>
Restriction of `xsd:string` with enumeration values	Class with stereotype <<Enumeration>>
Union of an enumeration and a pattern	Class with stereotype <<CodeList>>
Data and property type and global element for any object type that is not a direct or indirect extension of `gml:AbstractGMLType` and whose content model is a choice of properties	Class with stereotype <<Union>>
Local xsd:element of a simpleType or a complexType with simpleContent or a type that does not directly or indirectly inherit from `gml:AbstractGMLType`	UML Attribute
Local xsd:element of a type that contains `gml:AssociationAttributeGroup`	UML Association Role
Schematron constraints	Not encoded

5. Mapping the INTERLIS 2 UML profile to GML

In this chapter, the rules are defined for mapping different elements of INTERLIS language to GML. It may seem more reasonable to define mapping rules from INTERLIS profile in UML form to a UML profile based on ISO19136. However, this is not the case because the INTERLIS schema based on INTERLIS syntax is the normative schema of INTERLIS standard, while its UML profile serves more as a visualisation. It is more practical to define mapping rules between two textual languages then between a textual language and a graphic language, especially since ISO19136 defines rules of transformation from GML syntax to UML. Another argument that works for this approach to this problem is existance of „GML-Kodierungsregeln für INTERLIS“ document, which deals with the encoding of INTERLIS-syntax to GML enviroment.

5.1. Mapping data types

First step of mapping process is to define data types of target language that have the same or approximately the same capabilities as the data types in the source language. In the Table 4 there is a list of all INTERLIS 2 data types which are mappable to GML, as well as their best couple that is already in existence in the GML language. The third column states if the gml data type is a complex or a symple type in terms of XML data types. A simple type is a type whose instances can not contain any subelements. Note that Table 1 brings a somewhat different list of INTERLIS data types, i.e. there are also DateType, TimeType and DateTimeType types, which are treated as instances of FormattedType in table 4. The same goes for PolylineType, SurfaceType and AreaType, which are treated as instances of LineType in table 4.

Table 4 : Overview of the corresponding data types (GML-Kodierungsregeln für INTERLIS, 2011.)

INTERLIS	GML	Complex/Simple-Type
TextType	xsd:string	Simple
EnumerationType	xsd:string/gml:CodeType	Simple/Complex
EnumTreeValueType	xsd:string/gml:CodeType	Simple/Complex
AlignmentType	xsd:string	Simple
BooleanType	xsd:boolean	Simple
NumericType	xsd:integer / xsd:decimal / xsd:double	Simple
FormattedType	xsd:string/ xsd:date / xsd: time / xsd:dateTime	Simple
CoordinateType	gml:DirectPositionType	Complex
OidType	xsd:int / xsd:token	Simple
BlackboxType	xsd:anyType / xsd:base64Binary	Complex/Simple
ClassType	xsd:string	Simple
AttributePathType	xsd:string	Simple
LineType	gml:CurveType / gml:PolygonType	Complex

Next step is definition of the concrete mapping rules that will allow modeling as much of the original data as possible without loss of information. Following sections demonstrate these mapping rules for all of these data types.

5.1.1. Strings

Attributes / values of base type TEXT are encoded as xsd: normalizedString, MTEXT as xsd:string, NAME is encoded as xsd: string and URI as xsd:anyURI. If derived from the model, xsd: restriction is encoded.

INTERLIS

```
DOMAIN
  TestText = TEXT;
  TestTextLen = TEXT*10;
  TestMText = MTEXT;
  TestMTextLen = MTEXT*10;
  TestURI = URI;
  TestName = NAME;
```

GML

```
<xsd:simpleType name="TestText">
  <xsd:restriction base="xsd:normalizedString">
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestTextLen">
  <xsd:restriction base="xsd:normalizedString">
  <xsd:maxLength value="10"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestMText">
  <xsd:restriction base="xsd:string">
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestMTextLen">
  <xsd:restriction base="xsd:string">
  <xsd:maxLength value="10"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestURI">
  <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
<xsd:simpleType name="TestName">
  <xsd:restriction base="xsd:token">
  <xsd:maxLength value="255"/>
  <xsd:pattern value="[a-zA-Z][a-zA-Z0-9_]*"/>
  </xsd:restriction>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.2. Enumerations

In INTERLIS, enumerations can be updated and / or expanded.

Since updating nor expansion is possible when using XML schema enumerations, an INTERLIS enumeration is only mapped to an XML Schema enumeration if the range of values or the attributes is marked as FINAL. Otherwise, the list is mapped as gml:Code.

INTERLIS

```
DOMAIN
  FarbeFinal (FINAL) = (rot, gelb, gruen);
  Farbe = (rot, gelb, gruen);
  FarbePlus EXTENDS Farbe=(rot(dunkel,hell),gelb,gruen);
CLASS Fahrzeug =
  CarrosserieFarbe : Farbe;
END Fahrzeug;
CLASS Auto EXTENDS Fahrzeug =
  CarrosserieFarbe (EXTENDED) : FarbePlus;
END Auto;
```

GML

```
<xsd:simpleType name="FarbeFinal">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="rot"/>
    <xsd:enumeration value="gelb"/>
    <xsd:enumeration value="gruen"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="Farbe">
  <xsd:restriction base="gml:CodeType"/>
</xsd:complexType>
<xsd:complexType name="FarbePlus">
  <xsd:restriction base="gml:CodeType"/>
</xsd:complexType>
<xsd:element name="Fahrzeug" type="FahrzeugType"
substitutionGroup="gml:AbstractFeature"/>
<xsd:complexType name="FahrzeugType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="CarrosserieFarbe" type="Farbe"
        minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Auto" type="AutoType"
substitutionGroup="Fahrzeug"/>
<xsd:complexType name="AutoType">
```

```
  <xsd:complexContent>
    <xsd:extension base="FahrzeugType">
      <xsd:sequence>
<!—CarrosserieFarbe ist schon in FahrzeugType definiert -->
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- Dictionary -->
<gml:Dictionary gml:id="o1">
  <gml:identifier codeSpace="http://www.interlis.ch/ILIGML-
  1.0/ModelA"> Farbe</gml:identifier>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o2">
    <gml:identifier codeSpace=“http://www.interlis.ch/
    ILIGML-1.0/ModelA/Farbe">rot</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o3">
      <gml:identifier codeSpace=“http://www.interlis.ch/
      ILIGML-1.0/ModelA/Farbe">gelb</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o4">
      <gml:identifier codeSpace="http://www.interlis.ch/
      ILIGML-1.0/ModelA/Farbe">gruen</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
</gml:Dictionary>
<gml:Dictionary gml:id="o5">
  <gml:identifier codeSpace="http://www.interlis.ch/
  ILIGML-1.0/ModelA">FarbePlus</gml:identifier>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o6">
      <gml:identifier codeSpace="http://www.interlis.ch/
      ILIGML-
      1.0/ModelA/FarbePlus">rot.dunkel</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o7">
      <gml:identifier codeSpace="http://www.interlis.ch/
      ILIGML-1.0/ModelA/FarbePlus">rot.hell</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o8">
      <gml:identifier codeSpace="http://www.interlis.ch/
      ILIGML-1.0/ModelA/FarbePlus">gelb</gml:identifier>
```

```
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="o9">
      <gml:identifier codeSpace="http://www.interlis.ch/
      ILIGML-1.0/ModelA/FarbePlus">gruen</gml:identifier>
    </gml:Definition>
  </gml:dictionaryEntry>
</gml:Dictionary>
```

GML INSTANCE

```
<Fahrzeug gml:id="o1">
  <CarroserieFarbe>rot<CarroserieFarbe>
</Fahrzeug>
<Auto gml:id="o2">
  <CarroserieFarbe>rot.dunkel<CarroserieFarbe>
</Auto>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.3. Numeric data types

Numeric data types are encoded as xsd:integer, xsd:decimal or xsd:double depending on the upper and lower limit value. If derived from the model, xsd:restriction is encoded.

INTERLIS

```
DOMAIN
  TestInt = 1 .. 10;
  TestDec = 1.0 .. 10.0;
  TestDouble = 0.123e1 .. 0.234e1;
```

GML

```
<xsd:simpleType name=" TestInt">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="10"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestDec">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="1.0"/>
    <xsd:maxInclusive value="10.0"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TestDouble">
  <xsd:restriction base="xsd:double">
```

```
    <xsd:minInclusive value="1.23"/>
    <xsd:maxInclusive value="2.34"/>
  </xsd:restriction>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.4. Formatted domains

Formatted value ranges, the INTERLIS.XMLDate, INTERLIS.XMLTime and INTERLIS.XMLDateTime are all equivalents to the corresponding XML Schema date types (xsd:date, xsd:time, xsd:dateTime). All other formatted ranges are mapped in xsd:string.

INTERLIS

```
STRUCTURE GregorianDate =
  Year: 1582 .. 2999;
  SUBDIVISION Month: 1 .. 12;
  SUBDIVISION Day: 1 .. 31;
END GregorianDate;
DOMAIN
  BuchungsDatum = FORMAT INTERLIS.XMLDate
    "2002-01-01" ..
    "2007-12-31";
  StartZeit = FORMAT INTERLIS.XMLTime
    "00:00:00.000" ..
    "23:59:59.999";
  MessZeitpunkt = FORMAT INTERLIS.XMLDateTime
    "2002-01-01T00:00:00.000" ..
    "2007-12-31T23:59:59.999";
  EigenesDatum = FORMAT BASED ON GregorianDate
     ( Year "Y" Month "M" Day "D");
```

GML

```
<xsd:simpleType name="BuchungsDatum">
  <xsd:restriction base="xsd:date"/>
</xsd:simpleType>
<xsd:simpleType name="StartZeit">
  <xsd:restriction base="xsd:time"/>
</xsd:simpleType>
<xsd:simpleType name="MessZeitpunkt">
  <xsd:restriction base="xsd:dateTime"/>
</xsd:simpleType>
<xsd:simpleType name="EigenesDatum">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.5. Blackboxes

The XML variant of type BLACKBOX is encoded as xsd:any; the binary version as xsd:base64Binary.

INTERLIS

```
DOMAIN
  BlackboxXml = BLACKBOX XML;
  BlackboxBinary = BLACKBOX BINARY;
```

GML

```
<xsd:complexType name="BlackboxXml">
  <xsd:sequence>
    <xsd:any namespace="##any"
      minOccurs="0" maxOccurs="unbounded"
      processContents="lax"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="BlackboxBinary">
  <xsd:restriction base="xsd:base64Binary">
  </xsd:restriction>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.6. Class type

Class type is encoded as xsd:normalizedString.

INTERLIS

```
DOMAIN
  InterlisClassRef = CLASS;
```

GML

```
<xsd:simpleType name="InterlisClassRef">
  <xsd:restriction base="xsd:normalizedString">
  </xsd:restriction>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.7. Attribute type

Attribute type is encoded as xsd:normalizedString.

(GML-Kodierungsregeln für INTERLIS, 2011.)

INTERLIS

```
DOMAIN
  InterlisAttributeRef = ATTRIBUTE;
```

GML

```
<xsd:simpleType name="InterlisAttributeRef">
  <xsd:restriction base="xsd:normalizedString">
  </xsd:restriction>
</xsd:simpleType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.8. Coordinates

Coordinates are encoded as gml:PointPropertyType

INTERLIS

```
DOMAIN
  LKoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
    60000.00 .. 320000.00 [m] {CHLV03[2]},
    ROTATION 2 -> 1;
```

GML

```
<xsd:complexType name="LKoord">
  <xsd:complexContent>
    <xsd:restriction base="gml:PointPropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:Point"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.9. Line strings

Line strings will be coded as gml:CurvePropertyType.

INTERLIS

```
DOMAIN
  LKoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
    60000.00 .. 320000.00 [m] {CHLV03[2]}, ROTATION 2 -> 1;
  Strassenachse = POLYLINE WITH (ARCS,STRAIGHTS) VERTEX
    LKoord WITHOUT OVERLAPS>0.10;
```

GML

```
<xsd:complexType name="LKoord">
  <xsd:complexContent>
```

```
    <xsd:restriction base="gml:PointPropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:Point"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Strassenachse">
  <xsd:complexContent>
    <xsd:restriction base="gml:CurvePropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:AbstractCurve"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.1.10. Surfaces and tesselations

Individual polygons will be coded as gml:SurfacePropertyType.

INTERLIS

```
DOMAIN
  LKoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
    60000.00 .. 320000.00 [m] {CHLV03[2]}, ROTATION 2 -> 1;
  GebaeudeFlaeche = SURFACE WITH (ARCS,STRAIGHTS) VERTEX
    LKoord WITHOUT OVERLAPS>0.10;
```

GML

```
<xsd:complexType name="LKoord">
  <xsd:complexContent>
    <xsd:restriction base="gml:PointPropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:Point"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="GebaeudeFlaeche">
  <xsd:complexContent>
    <xsd:restriction base="gml:SurfacePropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:Polygon"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.2. Mapping stereotypes

This section describes mapping INTERLIS.ModelDef and INTERLIS.TopicDef stereotypes,

5.2.1. Stereotype ModelDef

INTERLIS

```
INTERLIS 2.3;
MODEL ModelA (de)
AT "mailto:ceis@localhost"
VERSION "2008-03-31" =
  IMPORTS Units;
END ModelA.
```

GML

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns=http://www.interlis.ch/ILIGML-1.0/ModelA
  targetNamespace=http://www.interlis.ch/ILIGML-1.0/ModelA
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:gml=http://www.opengis.net/gml/3.2
  xmlns:ns1="http://www.interlis.ch/ILIGML-1.0/Units">
  <xsd:annotation>
    <xsd:appinfo source="http://www.interlis.ch/ili2c">
      <ili2c:model>ModelA</ili2c:model>
      <ili2c:modelVersion>2008-03-31</ili2c:modelVersion>
      <ili2c:modelAt>mailto:ceis@localhost</ili2c:modelAt>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:import namespace="http://www.opengis.net/gml/3.2"/>
  <xsd:import namespace="http://www.interlis.ch/ILIGML-
  1.0/Units"/>
</xsd:schema>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

5.2.2. Stereotype TopicDef

INTERLIS

```
TOPIC Bodenbedeckung =
  CLASS BoFlaechen =
  END BoFlaechen;
  CLASS Gebaeude =
  END Gebaeude;
END Bodenbedeckung;
```

GML

```
<xsd:complexType name="BodenbedeckungMemberType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureMemberType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="BoFlaechen"/>
          <xsd:element ref="Gebaeude"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Bodenbedeckung" type="BodenbedeckungType"
  substitutionGroup="gml:AbstractFeature"/>
<xsd:complexType name="BodenbedeckungType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="member"
        type="BodenbedeckungMemberType"
        minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup
      ref="gml:AggregationAttributeGroup"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

GML INSTANCE

```
<Bodenbedeckung gml:id="b1">
  <member>
    <BoFlaechen gml:id="o1">
                ...
    </BoFlaechen>
  </member>
  <member>
    <BoFlaechen gml:id="o2">
                ...
    </BoFlaechen>
  </member>
</Bodenbedeckung>
```

(GML-Kodierungsregeln für INTERLIS, 2011.)

6. Applying the mapping rules to the MOpublic model

The final goal of this thesis is to convert a model created in INTERLIS language to a core UML profile based on the ISO19136 specification.

The model chosen for the mapping process is MOpublic v1.2.

MOpublic is a data set for users who would like to import Swiss cadastral surveying data in a simpler structure than the federal model DM.01-AV-CH. The data model MOpublic was defined as the Swiss standard model.

(http://www.cadastre.ch/internet/cadastre/en/home/products/mopublic.html)

6.1. The Mopublic model in INTERLIS

This chapter will describe MOpublic model in its original state, in INTERLIS language. The goal is to point out parts of structure which are most interesting in terms of mapping, as well as potentially unmappable parts.

MOpublic model is composed of reference system definition, domain definition, units definition, definition of topics of interest and finally definitions of classes, which are components of said topics. They are explained in more detail in the following sections.

6.1.1. Model declaration

At the begining, MOpublic model is declared (stereotipe ModelDef), along with its location on the web and current version. After that, INTERLIS language is imported, as well as set of rules for defining coordinate systems. The third file is a lookup table, which is supposed to contain descriptions of specific values for some of the attributes encountered throughout the model, but it is at this point empty.

```
MODEL MOpublic (en) AT http://www.cadastre.ch
  VERSION "2010-04-01" =
  IMPORTS UNQUALIFIED INTERLIS;
  IMPORTS CoordSys;
  IMPORTS LookUp;
```

What follows is a reference system definition:

```
   REFSYSTEM BASKET BCoordSys ~ CoordSys.CoordsysTopic
  OBJECTS OF GeoCartesian2D : CHLV03
  OBJECTS OF GeoHeight : SwissOrthometricAlt;
```

This cannot be mapped to a GML at a model level, because in GML a reference system is specifyed at data level.

6.1.2. Units

In the UNIT section, two derived units are defined: gradians and square meters. They are necessary later in the model to describe angular and areal attributes.The official mapping instructions, „GML-Kodierungsregeln für INTERLIS“, propose no means of translating unit definitions to GML. Because of that, UNIT section is ommited from GML version of MOpublic.

```
UNIT
  Grads = 200.0 / PI [rad];
  SquareMeters [m2] = (m * m);
```

6.1.3. Domains

DOMAIN section defines value limits for certain attributes, i.e. coordinates, altitude, bearings and accuracy. All of these can be modelled, but value limits aren't applicable.

Also, UUIDOID is defined as an OID type, which can be mapped to GML, but isn't used because GML has its own object ID system.

```
DOMAIN
  CoordP = COORD
  480000.000 .. 850000.000 [m] {CHLV03[1]},
  70000.000 .. 310000.000 [m] {CHLV03[2]},
  ROTATION 2 -> 1;

  CoordA = COORD
  480000.000 .. 850000.000 [m] {CHLV03[1]},
  70000.000 .. 310000.000 [m] {CHLV03[2]},
  -200.000 ..   5000.000 [m] {SwissOrthometricAlt[1]},
  ROTATION 2 -> 1;

  Altitude = -200.000 .. 5000.000 [m];
  Rotation = 0.0 .. 399.9 [Grads];
  Accuracy = 0.0 .. 700.0;
  UUIDOID = OID TEXT*36;  !! of ISO 11578
```

6.1.4. Topics and classes

Last section of INTERLIS MOpublic model is a set of definition of topics of interest and their classes (stereotype TopicDef). This part is most important, as it defines feature classes for feature data sets. It can be completely mapped to GML.

It is also worth noticing that UNIQUE constraint, which is here applied, isn't mentioned in „GML-Kodierungsregeln für INTERLIS“ document. However, it is easily modelled using XML's pre-defined „unique“ constraint.

```
TOPIC Control_points
```

```
  BASKET OID AS UUIDOID;
  OID AS UUIDOID;
  CLASS Control_point =
    Category : MANDATORY 0..5; !! Designation under
    LookUp.Lookup_tables.Control_point_Category
    IdentND : MANDATORY TEXT*12;
    Number : MANDATORY TEXT*12;
    Geometry : MANDATORY CoordP;
    Plan_accuracy : Accuracy;
    Geom_alt : Altitude;
    Alt_accuracy : Accuracy;
    Mark : MANDATORY 0..8; !! If the Mark is not defined,
    indicate Code 8 = undefined     !! Designation under
    LookUp.Lookup_tables.Mark_type
    FOSNr : 0 .. 9999;
  UNIQUE IdentND, Number;
  END Control_point;
END Control_points;
```

Also, some of the classes are related to each other by means of associations, aggregations and compositions. This is an example of a compositions:

```
ASSOCIATION LCSurface_PosTextPosText_of =
  LCSurface_PosText -- {0..*} LCSurface_PosText;
  PosText_of -<#> {1} LCSurface;
END LCSurface_PosTextPosText_of;
```

The „GML-Kodierungsregeln für INTERLIS“ makes no notice of aggregations and compositions, so it is decided that all of these relations will be treated as associations.

6.2. The resulting GML Schema representation of the MOpublic model

This chapter describes final product of this thesis, MOpublic model mapped to GML schema. Note that only excerpts of the code are included due to size of the final document, while the complete schema can be found attached.

A GML version of MOpublic model starts with xml declaration containing namespace declarations, followed by domain definitions, auxiliary type definitions and finally definitions of topics and classes.

6.2.1. Model declaration

First line of GML MOpublic schema defines XML version of the document, as well as encoding standard. Then comes the „schema“ element, which defines namespace prefixes. These are used to differentiate elements and attributes from different dictionaries and for avoiding possible name collisions.

Note that „GML-Kodierungsregeln für INTERLIS“ proposes default namespace for MOpublic schema document („no namespace“), however, when implementing the

GML schema, „mopublic“ namespace was introduced. This is because „unique“ has difficulties recognising a target element from default namespace. Possible solution would be to assign both namespaces to this schema document. Then it would be possible to use „no namespace“ throughout the document, but at the same time to use non-default prefix in the unique constraints. However, this is not applied at the moment.

Regarding other namespaces, xlink is used for referencing an attribute or element of one class to another class, and ili2 namespace is used for model description.

After „schema“ element, „annotation“ element is used to describe version of a model, its location on the web, as well as the current version.

„Import“ element contains precise URL of a GML schema, which contains an „import“ element to XLink schema. This way, all the elements from these two schemas that are used in MOpublic schema can be checked for validity.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:mopublic="http://www.interlis.ch/ILIGML-1.0/MOpublic"
  xmlns:gml="http://www.opengis.net/gml/3.2"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:ili2="http://www.interlis.ch/ili2c"
  targetNamespace="http://www.interlis.ch/ILIGML-
  1.0/MOpublic"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
   <xsd:annotation>
    <xsd:appinfo source="http://www.interlis.ch/ili2c">
      <ili2:model>MOpublic.en.v1.2</ili2:model>
      <ili2:modelVersion>2011-07-30</ili2:modelVersion>
      <ili2:modelAt>http://www.cadastre.ch</ili2:modelAt>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:import namespace=http://www.opengis.net/gml/3.2
  schemaLocation=
  "http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
```

6.2.2. Domain definitions

All the domains from INTERLIS MOpublic model are mapped to GML in a way proposed by „GML-Kodierungsregeln für INTERLIS“ document. However, it should be noted that

-UUIDOID is never used

-CoordP is only defined as a „gml:Point“ element, without value limits which are stated in INTERLIS version of MOpublic.

```
<xsd:complexType name="CoordP">
```

```
  <xsd:complexContent>
    <xsd:restriction base="gml:PointPropertyType">
      <xsd:sequence>
        <xsd:element ref="gml:Point"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="CoordA">
  <xsd:sequence>
    <xsd:element name="CoordP" type="mopublic:CoordP"/>
    <xsd:element name="Altitude" type="mopublic:Altitude"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="Altitude">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="-200.000"/>
    <xsd:maxInclusive value="5000.000"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Rotation">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0.0"/>
    <xsd:maxInclusive value="399.9"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Accuracy">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0.0"/>
    <xsd:maxInclusive value="700.0"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="UUIDOID">
  <xsd:restriction base="xsd:token">
    <xsd:length value="36"/>
    <xsd:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-
    [a-f0-9]{4}-[a-f0-9]{12}"/>
  </xsd:restriction>
</xsd:simpleType>
```

6.2.3. Auxiliary types

Before mapping any topics and classes in GML, 11 auxiliary types were created which were not specified in INTERLIS MOpublic model, but conform to types

required oftenly throughout the schema. They were created for purpose of making the schema file smaller and easier to read.

```
<xsd:simpleType name="FOSNrType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="9999"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="QualityType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="4"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="NumberType">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:maxLength value="12"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="RegBL_EGIDType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="999999"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Type01">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:pattern value="[0-1]"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Type02">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:pattern value="[0-2]"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Type03">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:pattern value="[0-3]"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="Number_NameType">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:maxLength value="30"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="NameType">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:maxLength value="40"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Operating_CompanyType">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:maxLength value="30"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="AssociationType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

6.2.4. Topics and classes

Topics and classes are mapped according to „GML-Kodierungsregeln für INTERLIS“ document and chapter 5 of this document. An example here is „Control_points“ topic.

```
<xsd:element name="Control_points"
type="mopublic:Control_pointsType"
substitutionGroup="gml:AbstractFeature">
  <xsd:unique name="CP_uniqueIdentND">
    <xsd:selector xpath="*/mopublic:Control_point"/>
    <xsd:field    xpath="mopublic:IdentND"/>
  </xsd:unique>
  <xsd:unique name="CP_uniqueNumber">
    <xsd:selector xpath="*/mopublic:Control_point"/>
    <xsd:field    xpath="mopublic:Number"/>
  </xsd:unique>
</xsd:element>
<xsd:complexType name="Control_pointsType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="member"
        type="mopublic:Control_pointsMemberType"
```

```
        minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup
      ref="gml:AggregationAttributeGroup"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Control_pointsMemberType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureMemberType">
      <xsd:sequence>
        <xsd:element ref="mopublic:Control_point"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Control_point"
type="mopublic:Control_pointType"
substitutionGroup="gml:AbstractFeature"/>
<xsd:complexType name="Control_pointType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="Category">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="5"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="IdentND" type
        ="mopublic:NumberType"/>
        <xsd:element name="Number" type
        ="mopublic:NumberType"/>
        <xsd:element name="Geometry" type="mopublic:CoordP"/>
        <xsd:element name="Plan_accuracy"
        type="mopublic:Accuracy" minOccurs="0"/>
        <xsd:element name="Geom_alt" type="mopublic:Altitude"
        minOccurs="0"/>
        <xsd:element name="Alt_accuracy"
        type="mopublic:Accuracy"
        minOccurs="0"/>
        <xsd:element name="Mark" default="8">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="8"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
```

```
        <xsd:element name="FOSNr" type="mopublic:FOSNrType"
        minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The above example from chapter 6.2.5. of two associated classes is mapped as follows:

```
<xsd:complexType name="LCSurface_PosTextType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="PosText_of"
        type="gml:ReferenceType">
          <xsd:annotation>
            <xsd:appinfo>
              <gml:targetElement>
              mopublic:LCSurface</gml:targetElement>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="PosText_of.LINK_DATA"
        minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element
              ref="mopublic:LCSurface_PosTextPosText_of"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
              ...
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="LCSurface_PosTextPosText_of"
type="mopublic:AssociationType"
substitutionGroup="gml:AbstractFeature"/>
```

As proposed in „GML-Kodierungsregeln für INTERLIS“ document, an instance file of this schema should be structured with an element called „TRANSFER“ as a root element. Data from separate topics should be placed in separate „basket“ elements.

```
<xsd:complexType name="TRANSFERMemberType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureMemberType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="gml:AbstractFeature"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element   name="TRANSFER"   type="mopublic:TRANSFERType"
substitutionGroup="gml:AbstractFeature"/>
<xsd:complexType name="TRANSFERType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="baskets" type=
        "mopublic:TRANSFERMemberType"
        minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup
      ref="gml:AggregationAttributeGroup"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

6.3. GML Instance of the MOpublic model

This section contains excerpt from an XML document which is an instance of the MOpublic GML schema. This excerpt instantiates the topic „Control_points“. The whole test file is enclosed as an attachment.

By means of creating an instance document, the whole MOpublic model was tested for errors. This version of test file is valid, while numerous intentional mistakes were made during the testing process to ensure that all the value limits and constraints work well.

The tools used for the validation process were *Altova XMLspy (http://www.altova.com/xmlspy.html)*, as well as *DecisionSoft*'s online *XML Schema Validator (http://tools.decisionsoft.com/schemavalidate/)*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TRANSFER gml:id="TR01" xmlns=http://www.interlis.ch/ILIGML-
1.0/MOpublic
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xlink=http://www.w3.org/1999/xlink
xsi:schemaLocation="http://www.interlis.ch/ILIGML-
1.0/MOpublic MOPublicGML_web_schemas.xml">
  <baskets>
    <Control_points gml:id="cp01">
      <member>
        <Control_point gml:id="cp21">
          <Category>2</Category>
          <IdentND>4</IdentND>
          <Number>21</Number>
          <Geometry>
            <gml:Point gml:id="p21"
            srsName=
            "http://www.opengis.net/def/crs/EPSG/0/4326">
              <gml:coordinates>45.67, 88.56</gml:coordinates>
            </gml:Point>
          </Geometry>
          <Plan_accuracy>100</Plan_accuracy>
          <Mark>6</Mark>
        </Control_point>
      </member>
    </Control_points>
  </baskets>
  <baskets>
    <Land_Cover gml:id="LC01">
      <member>
        <LCSurface gml:id="lcsurface21">
          <Geometry>
            <gml:Polygon gml:id="pollc01">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>45.256 -110.45 46.46 -109.48
                  43.84 -109.86 45.256 -110.45</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
              </gml:Polygon>
          </Geometry>
          <Quality>1</Quality>
          <Type>6</Type>
          <RegBL_EGID>9923</RegBL_EGID>
        </LCSurface>
      </member>
      <member>
        <LCSurface_PosText gml:id="pt01">
          <PosText_of xlink:href="# lcsurface21"/>
          <Type>1</Type>
```

```
          <Number_Name>Dalmacija</Number_Name>
        </LCSurface_PosText>
      </member>
    </Land_Cover>
  </baskets>
</TRANSFER>
```

7. Summary and Conclusion

The goal of this thesis was to implement a core UML profile that could unify spatial data from different data systems. Geography Markup Language (GML) was explored as an option, because of its clear syntax and ability to be completely mapped to an Unified Markup Language (UML).

The MOpublic model, written in INTERLIS language, was subjected to the transformation.

First part of the transformation itself was defining the transformation methods for the individual elements of the INTERLIS profile to GML. This process was further separated into finding the appropriate methods of transformation for all of the INTERLIS data types, and then finding the methods of modelling INTERLIS stereotypes into GML.

Second part of the process was applying those rules on the MOpublic model. The model was composed of nine topics, all of which were sucessfully mapped to GML. The parts that weren't suitable for transfer were unit definitions and coordinate reference system definition. The reason for this is that transformation methods for units aren't developed, and that GML doesn't allow choosing a coordinate reference system for the whole model. Also, aggregations and compositions are mapped the same as the associations beacause there are no solutions for mapping these relations in a different way at the moment.

When everything is taken into consideration, we can say that results are very satisfying. The MOpublic model was almost completely transferred into Geography Markap Language. All of the crucial parts of the model are preserved, with the semantics kept intact. This means that models which are encoded in INTERLIS language will have a way of transformation into a core profile. The results of this thesis ecourage use of GML as a core profile that could unify spatial data from different spatial systems that are currently in use by members of the European Union.

Further research can be done on the subject of transformation of different UML profiles to GML as a core UML profile, as well as on the transformation from GML to INSPIRE UML profile, which is a final goal of this research.

8. Attachments

Due to the length of the files, whole models and a test file were not inserted as a text in this document. Instead, only parts of files were used for the examples, with the complete files enclosed on a CD.

8.1. Contents of the supplied CD

Results of this thesis, as well as the original model used, are stored on the attached compact disc. The contents of the disc are listed in Table 5.

Table 5 : Contents of the supplied media

Nr.	File	Content
1.	Master Thesis.doc	Text of the master thesis in a word document
2.	Master Thesis.pdf	Text of the master thesis in a pdf document
3.	MOpublic_en_v1.2.ili	INTERLIS version of MOpublic model
4.	MOPublic_v1.2._GML_schema.xml	GML version of MOpublic model
5.	Test.xml	Instance file of GML MOpublic model
6.	INTERLIS 2 – Reference manual	Documentation on INTERLIS language

References:

FOLDOC (2001): Unified Modeling Language

Si Alhir, Sinan (2002): Guide to applying the UML

Kutzner, Eisenhut (2010): Comparative studies on the modeling and model transformation in the Lake Constance region in the Context of INSPIRE

Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany (2009): Documentation on the Modelling of Geoinformation of Official Surveying and Mapping

ISO 19103 (2005.): Geographic information — Conceptual schema language

OpenGIS® Geography Markup Language (GML) Encoding Standard (2007.)

INTERLIS 2 – Metamodel (2008.)

INTERLIS Version 2 - Reference Manual (2006.)

Object Management Group (2005.): UML Superstructure Specification

eCH-Fachgruppe (2011.): INTERLIS GML-Kodierungsregeln für INTERLIS

List of used URL-s:

URL 1. History of INTERLIS http://www.interlis.ch/general/historique_e.php
(August 28, 2011.)

URL 2. About MOpublic
http://www.cadastre.ch/internet/cadastre/en/home/products/MOpublic.html
(August 28, 2011.)

URL 3. INTERLIS http://www.interlis.ch/content/index.php?language=e
(August 28, 2011.)

URL 4. XML SPY http://www.altova.com/xmlspy.html
(August 22, 2011.)

URL 5. DecisionSoft's Online XML schema validation tool
http://tools.decisionsoft.com/schemavalidate/
(August 22, 2011.)

List of figures

List of tables

europass

Europass Curriculum Vitae

Personal information

First name(s) / Surname(s)	**Ivan Mihaljević**
Address(es)	25, Put kroz Meterize, 22000, Šibenik, Hrvatska
Telephone(s)	+ 385 (0)22338722 Mobile: +385 (0)955159022
Fax(es)	
E-mail	imihaljevic@geof.hr
Nationality	Croat
Date of birth	21.04.1988
Gender	Male

Desired employment / Occupational field

Geoinformatics

Work experience

Dates	September 2009 – January 2010
Occupation or position held	Demonstrator
Main activities and responsibilities	- Assisting the professor in tutoring younger students on the subject of Differential geometry - Preparing the students for the exam
Name and address of employer	Faculty of Geodesy, Chair of Mathematics and Physics, Kačićeva 26, 10 000 Zagreb
Type of business or sector	Education
Dates	June 2003
Occupation or position held	Cinema operator
Main activities and responsibilities	- Setting up and maintaining cinema equipment for the outdoor screening of the movies
Name and address of employer	International children's festival, Šibenik
Type of business or sector	Entertainment
Principal subjects/occupational skills covered	Geodesy and geoinformatics
Name and type of organisation providing education and training	University of Zagreb, Croatia
Level in national or international classification	ISCED 5

Personal skills and competences

Mother tongue(s) **Croatian**

Other language(s)

Self-assessment	Understanding				Speaking				Writing	
European level (*)	Listening		Reading		Spoken interaction		Spoken production			
English	C1	Proficient user	C2	Proficient user	B2	Independent user	C1	Proficient user	C1	Proficient user
Italian	A2	Basic user	A2	Basic user	A1	Basic user	A2	Basic user	A2	Basic user

(*) Common European Framework of Reference for Languages

Social skills and competences

Team work: I have been part of many teams in various activities, including team sports, science researches and band playing

Computer skills and competences

- Experienced with Microsoft Office programs
- Experienced with various GIS applications

Artistic skills and competences

- I have been recreationally playing guitar for 8 years
- I have been engaged in amateur film-making while in high school

Other skills and competences

Black belt in Karate

Driving licence

B category

Additional information

Personal interests: I like to play all kinds of sports, especially football and table tennis. As for regular exercise, I practice karate or go to gym. My favourite way of socialising is playing guitar and singing with people. My long-term goal is to acquire knowledge about as many cultures as possible.