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

Comparison of different language paradigms for modeling geospatial data

Author:

Helena Petković

Vinka Maglice 50

Šibenik

hpetkovic@geof.hr

Supervisors: 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, for their patience and shared knowledge;

To the Technische Universität München for the great opportunity and unforgetable three months;

To my parents for unconditional support;

To all my friends who made studying easy and fun;

To Ivan who was there for me in good and bad times.

Comparison of different language paradigms for modeling geospatial data

Helena Petković

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 capability to communicate and transfer data, called interoperability, is needed to ensure successful spatial data exchange between two or more different information systems. To achieve that interoperability, one main aspect must be fulfilled: to specify system independent data structure. The formal description of this data structure and content is provided by an application schema. According to the ISO 19100 series of geographic information standards, UML (Unified Modeling Language) and GML (Geography Markup Language) are formal languages recommended to describe an application schema. UML is an object-oriented language used for spatial data modeling, while GML uses the XML syntax to express geographical features including features, coordinate reference systems, geometry, topology, time, units of measure etc.
The transformation from UML application schema to the GML application schema (according to ISO 19109) is based on a set of encoding rules that are presented and explained in this thesis.

Keywords: *language paradigm, ISO standard, UML, GML, FME, General Feature Model*

Usporedba različitih jezičnih paradigmi za modeliranje geoprostornih podataka

Helena Petković

Produženi sažetak: INSPIRE Direktiva zahtjeva da zemlje članice Europske Unije usklade svoje sustave prostornih podataka na način koji će ujediniti te sustave u svrhu interoperabilnosti geoprostornih web podataka i standardiziranih formata razmjene. Europski modeli podataka, pod nazivom «INSPIRE Data Specifications», su definirani za važne teme čiji su glavni cilj transformacije. Interoperabilnost je definirana kao sposobnost za komunikaciju i prijenos podataka i potrebna je za osiguranje uspješne razmjene prostornih podataka između dva različita informacijska sustava. Za postizanje interoperabilnosti, potrebno je ispuniti jedan glavni zahtjev: uspostaviti sustav neovisne strukture podataka. Formalni opis takve strukture podataka i njegova sadržaja pruža aplikacijska shema. Prema ISO 19100 serijama standarda za geoprostorne podatke, UML (Unified Modeling Language) i GML (Geography Markup Language) su formalni jezici koji se preporučaju za definiranje aplikacijske sheme. UML je objektno orijentirani jezik koji se koristi za modeliranje prostornih podataka, dok GML koristi XML sintaksu za izražavanje prostornih značajki kao što su referentni koordinatni sustav, topologija, vrijeme, jedinice za mjere itd. Transformacija iz UML aplikacijske sheme u GML aplikacijku shemu (uređena pravilima koje pruža ISO 19109 standard) se temelji na skupu pravila koji su prezentirani i objašnjeni u ovom radu. Osim transformacije između UML i GML shema, ovaj rad izlaže još jednu analizu: analizu razlike relacijske paradigme na kojoj se bazira program FME (Feature Manipulation Engine) te GML aplikacijske sheme definirane na XML paradigmi. Razlike u strukturi, načinu nasljeđivanja između pojedinih elemenata, prikazu atributa itd. Samo su neke u nizu mnogobrojnih razlika između navedenih paradigmi. Ova analiza pruža uvid u pojedinačnu strukturu korištenih alata za usporedbu, kao i u prednosti i mane manipuliranja podacima ovisno o različitoj paradigmi.

GIS grupa na Tehničkom Sveučilištu u Münchenu trenutačno provodi istraživanje pod nazivom “Prototypical transformation of spatial data from the cross-border Lake Constance region to INSPIRE” (Prototip transformacije prostornih podataka za granično područje Jezera Constance za INSPIRE). Projekt se fokusira na integraciju prostornih podataka s područja Jezera Constance gdje Njemačka, Švicarska i Austrija dijele granicu. U svakoj od navedenih zemalja podaci vezani za to područje su pohranjeni u različite sustave, koriste različite prostorne referentne sustave i formate za razmjenu podataka, a povrh svega su im drukčije sheme koje određuju strukturu i semantiku prostornih podataka. Da bi se projekt realizirao, kartografske agencije pojedine zemlje pružaju prostorne podatke sa područja Jezera Constance. Podatke koji su korišteni u ovom diplomskom radu osigurava Bavarska Agencija za Izmjeru i Geoinformacije (Landesamt für Vermessung und Geoinformation Bayern) i oni pokrivaju administrativno područje Lindau-

a. Shema za korištene podatke je German Digital Landscape Model (ATKIS Base-DLM) shema kao dio AFIS-ALKIS-ATKIS Referentnog Modela (AAA). Format za prijenos podataka je NAS (Normbasierte Austauschschnittstelle), sučelje za razmjenu podataka koje se temelji na standardima.

Diplomski rad pisan je na engleskom jeziku zbog toga što je nastao u suradnji sa TUM-om (tehničkim sveučilištem u Münchenu) koju je omogućio ERASMUS program međunarodne suradnje.

Ključne riječi: jezična paradigma, ISO standard, UML, GML, FME, General Feature Model

Comparison of different language paradigms for modeling geospatial data

Helena Petković

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

1. Introduction

1.1. Motivation and objective

INSPIRE Directive, established in 2007 by the EU, set its goals on establishing an infrastructure for spatial information in Europe. INSPIRE enables the sharing of spatial information among public sector and facilitates public sector access to the spatial information. Inside INSPIRE, many topical and tehnical themes have been developed. One of them is INSPIRE Data Specification, defining rules regarding interoperability of spatial data sets and services. Data Specification covers different themes such as Protected sites, Hidrography, Coordinate Reference System etc. Each of that congregation of data is organized into data models, that are defined by means of modeling language, each language conforming to a certain language paradigm.

UML (Unified Modeling Language) is a standardized modeling language used to create visual models of object - oriented software systems. With UML, geospatial features and their properties can be described in the form of diagrams independently from a certain data format. It has been established as a standard for modeling spatial data in Germany, Switzerland and the EU. UML conforms to a object - oriented language paradigm.

GML (Geography Markup Language) is written in XML Schema using the XML syntax for the description of application schemas with focus on the geographical information. As it is stated and described in this paper, GML application schema is used as an application schema for system-independent data exchange and file format NAS.

The aim of this thesis was to bring different language paradigms (object-oriented, relational and xml) together and compare them. The data used in this thesis was provided by the Landesamt für Vermessung und Geoinformation Bayern (Bavarian Agency for Surveying and Geoinformation) and covers the administrative district Lindau.

Structure of the thesis was designed to give an introduction about language paradigms in general: Object – oriented, relational and xml language paradigm. ISO standards that are stated and described in the thesis are basis for the entire practical part, giving the rules for spatial data schemas, geometries and GML. The fourth chapter is about FME (The Feature Manipulation Engine) – the software used for the comparision in the first part of practical work. In this chapter, data used for this work and two schemas :AAA NAS 6.0 Schema (GML application schema) and AAA UML schema, are also described. After that, the practical part of the thesis is exposed, where two comparisions have been made: Differences between NAS schema and FME; and Differences between NAS schema and UML schema, with impact of the General Feature Model on this analysis.

1.2. Related works

Different standards and rules are created that define the modeling of data. Some of these standards are from the ISO 19100 series of standards, such as ISO 19103, 19107, 19109, 19136 and some are general IT standards such as UML, XML, RDF etc. By definition, modeling of data is process of creating the data model using data modeling methods, such as entity-relationship or semantic data modeling. All of these concepts are based on different language paradigms, and it is not an easy work to combine them. One work that gives UML-to-GML encoding rules is described below:

Experiences of UML-to-GML Encoding (Grønmo, Solheim, Skogan, 2001): SINTEF started the project called *GeNorway – Model-based infrastructure for living geospatial data in eNorway.* The purpose of the project is to test the practicability of selected standards from ISO/TC 211 in an implementation of a Web Feature Server (WFS) according to OGC's specification. GML is used in Web service to represent spatial information. The final product of the project are UML-to-GML encoding rules and tools for translating UML class models to GML Schemas.

Also relevant work for this thesis is Comparative Studies Regarding Modeling and Schema Translation in the Lake Constance Region (Kutzner T., Eisenhut C., 2011), especially the part about AAA application schema. That chapter states data types and geometries that are used in AAA application schema, with description of the schema and its components in general.

.

2. Language paradigms for modelling spatial data

A paradigm of a programming language defines the concepts and abstractions used to represent the elements of a program. In the following chapter, three language paradigms are described and they present the core of this thesis.

2.1. Object oriented paradigm

The object−oriented paradigm focuses on the behavioral and structural characteristics of entities as complete units. It is concept−centric in that it focuses on all the types of features that constitute any given concept. The paradigm encompasses and supports the following principles: (Sinan Si Alhir "UML in a Nutshell: A Desktop Quick Reference". O'Reilly Associates, Inc., 1998.)

- Abstraction involves the formulation of representations by focusing on similarities and differences among a set of entities to extract essential characteristics (relevant common features) and avoid incidental characteristics (irrelevant distinguishing features) in order to define a single representation having those characteristics that are relevant to defining every element in the set.

- Encapsulation involves the packaging of representations by focusing on the hiding of details to facilitate abstraction, where specifications are used to describe what an entity is and what an entity does and implementations are used to describe how an entity is realized.

- Inheritance involves the relating and reusing of existing representations to define new representations.

- Polymorphism involves the ability of new representations to be defined as variations of existing representations, where new implementations are introduced but specifications remain the same such that a specification has many implementations.(Sinan Si Alhir. "UML in a Nutshell : A Desktop Quick Reference". O'Reilly Associates, Inc., 1998.)

2.1.1. Objects and classes

Objects and classes abstract entities.

Objects are representational constructs of entities. Objects encapsulate structural characteristics known as attributes and behavioral characteristics known as operations. Attributes are representational constructs of structural characteristics of entities and determine the possible states of an object. Operations are representational constructs of behavioral characteristics of entities and determine the possible behaviors of an object as invoked in response to receiving a

message. Objects have identity and are instances of classes. Fundamentally, objects are abstracted entities that encapsulate state and behavior. Classes are descriptions of objects with a common implementation. Classes are concerned with the implementation of uniform structural characteristics and behavioral characteristics. Fundamentally, classes are descriptions of objects with common attributes, operation implementations, semantics, associations, and interactions.

Types are descriptions of objects with a common specification. Types are concerned with the specification of uniform structural characteristics and behavioral characteristics. Types may be explicitly related to classes, where a class receives an interface from a type and provides an implementation for the interface, or types may be implicitly related to classes, where a class defines an interface and provides an implementation. Fundamentally, types are descriptions of objects with common attributes, operation interfaces, semantics, associations, and interactions. (Sinan Si Alhir. "UML in a Nutshell : A Desktop Quick Reference". O'Reilly Associates, Inc., 1998.)

2.1.2. Links and associations

Links and associations abstract relationships among entities.

Links are representational constructs of entities that relate other entities. Links are instances of associations.Fundamentally, links are abstracted relationships among objects.

Associations are descriptions of links with a common implementation. Aggregations are associations that specify a whole−part relationship among an aggregate and component parts.

Compositions are aggregations with strong ownership and coincident lifetime constraints among a composite and component parts.

Generalizations are associations specifying a relationships that relate more general representational constructs and more specific representational constructs. The more specific representational constructs are derived from existing more general representational constructs and acquire the characteristics of the more general representational constructs via inheritance.Because the more specific representational constructs receive the characteristic of the more general representational constructs, instances of the more specific representational constructs may be substituted for instances of the more general representational constructs.

Polymorphism enables the same message (operation interface) to invoke the appropriate method (operation implementation) based on the class of the receiver

when a more specific instance is substituted for a more general instance. Fundamentally, associations are descriptions of linkswith common attributes, operation implementations, semantics, associations, and interactions. (Sinan Si Alhir. "UML in a Nutshell : A Desktop Quick Reference". O'Reilly Associates, Inc., 1998.)

2.1.3. Scenarios and interactions

Scenarios and interactions abstract occurrences among entities.

Scenarios are representational constructs of entities that are conduits for a sequence of message exchanges among other entities. Scenarios are instances of interactions. Fundamentally, scenarios are abstracted message exchanges among objects.

Interactions are descriptions of scenarios with a common implementation. Message exchanges involve a sender who is said to apply an operation on a receiver by sending a message (operation interface) that invokes a method

(operation implementation) within the receiver. Fundamentally, interactions are descriptions of scenarios with common message exchange sequences, classes, and associations. (Sinan Si Alhir. "UML in a Nutshell : A Desktop Quick Reference". O'Reilly Associates, Inc., 1998.)

2.2. UML (Unified Modeling Language)

The UML was released by the Object Management Group (OGC) with the purpose of providing a stable and common design language that could be used in developing computer applications. Today it became a standard modeling language because it is programming – language independent (UML modeling tools can have different appliances) and because it is a language, not a methodology. The second characteristic is important because language, as opposed to a methodology, can be applied into any company's way of conducting bussines without changes. UML provides several types of diagrams: class, activity, use – case, etc. In the following text, class diagram will be explained because it is important for this thesis. (URL 1)

2.2.1. Class diagram

A class diagram is a type of static structure diagram that describes the structure of a system in the form of system's classes, attributes, methods and relationships between them. Figure 1 shows class diagrams on the example of UML schema used in the practical part of this work..

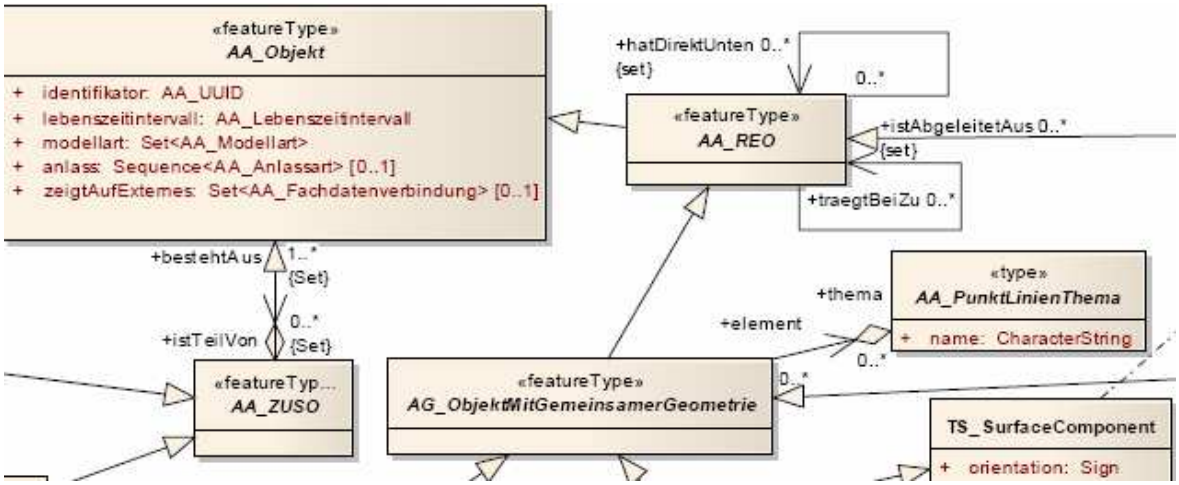


Figure 1: Example of Class diagram

Classes

A class is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics. A class is modeled concept. Depending on the kind of model, the concept may be based on the real world (for a conceptual model), or it may be based upon implementation of either platform independent system concepts (for specification models) or platform specific system concepts (for implementation models). A UML class has a name, a set of attributes, a set of operations and constraints. A class may participate in associations.

The visibility of an attribute, operation, or association of a class may be limited in various ways. Private visibility ('-') means that an attribute or operation is only visible within the class within which it is defined; an association is only navigable from the class at its source end. Protected visibility ('#') means an attribute or operation is visible only within the class within which it is defined and to its subclasses; an association is only navigable from the class at its source end and from its subclasses. Public visibility ('+') means that an attribute or operation is visible externally to any class within the same package as the class within which it is defined; an association is navigable from any class associated with the class within which it is defined. Private and protected visibilities are normally not used in the ISO geographic information standards. Figure 2 shows symbols for derived element, visibility and class level definitions.

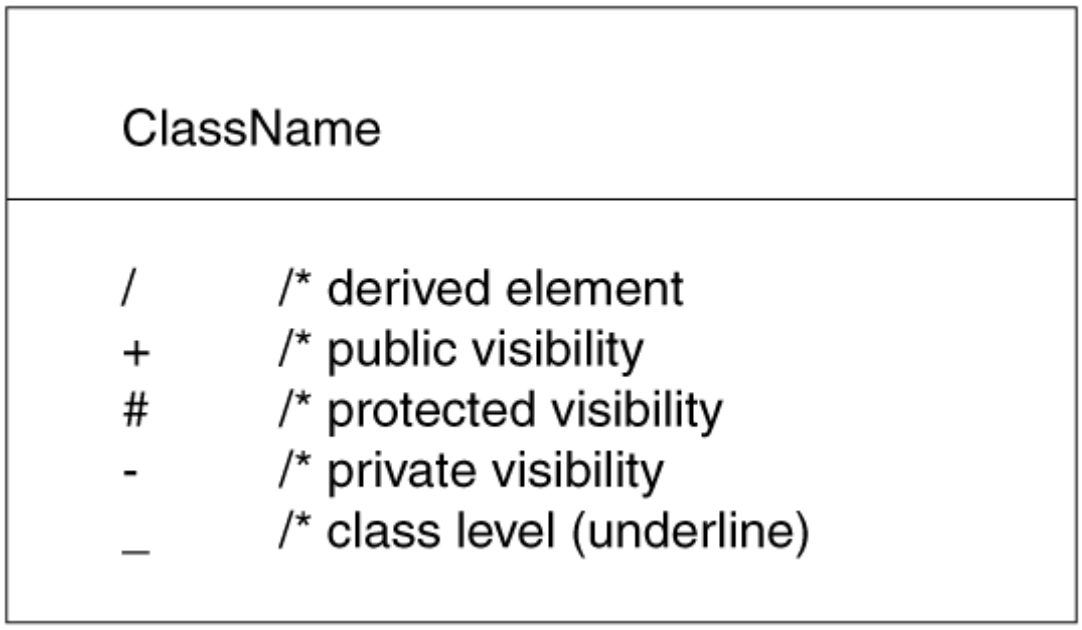


Figure 2: Symbols for derived element, visibility and class level definitions (ISO/TS 19103:2005(E))

Attributes

UML notation for an attribute has the form:

[<<stereotype>>] [visibility] name [multiplicity] [:type] [= initial value] [{property-string}]

where optional elements are enclosed in brackets, and:

stereotype is the name of the stereotype (see *stereotypes* below) to which the attribute belongs, if any.

visibility specifies the visibility of the attribute.

name is a character string that identifies the name of the attribute

multiplicity specifies the number of values) that an instance of a class may have for a given attribute.

type-expression identifies the data type of the attribute.

initial value specifies the default value for the attribute.

Properties can be user-defined. An attribute should be unique within the context of a class. A data type must always be specified; there is no default type. If no explicit multiplicity exists, the multiplicity is assumed to be 1. An attribute may define a default value, which is used when an object of that data type is created. Default values are defined by explicit default values in the UML definition of the attribute. (ISO/TS 19103:2005(E))

Operations

Operations are presented in UML class diagrams in compliance with the UML Notation Guide. An operation is specified by a text string:

visibility name (parameter-list) : return-type-expression [{ property-string }]

where optional elements are enclosed in brackets, and:

visibility specifies the visibility of the operation.

name is a character string that identifies the name of the operation.

parameter-list is a list of parameters .

return-type-expression specifies the data type or types of the value returned by the operation.

property-string contains any tagged values applied to the operation.

An element of the property-list has the form:

[kind] name : type-expression [= default-value] where:

kind is *in*, *out*, or *inout*, with the default *in* if absent.

name is the name of the parameter.

Type-expression identifies the data type of the parameter.

Default-value is an initial value for the parameter.

The values for *kind* have the following meanings:

in – The parameter value is read, but not modified, by the operation.

out – The parameter value is created by the operation and may be read upon the operation's return to the point of call.

inout – The parameter is read and modified by the operation and may be read upon the operation's return to the point of call. Exceptions can be defined as class elements using the stereotype (see *stereotypes*) <<Exception>>. (ISO/TS 19103:2005(E))

Relationships

A relationship in UML is a connection among model elements. Different kinds of relationships include association, generalization, aggregation/ composition, and several kinds grouped under dependency. The following chapter describes

association, composition and aggregation. Figure 3 shows different kind of relationships in UML among model elements.

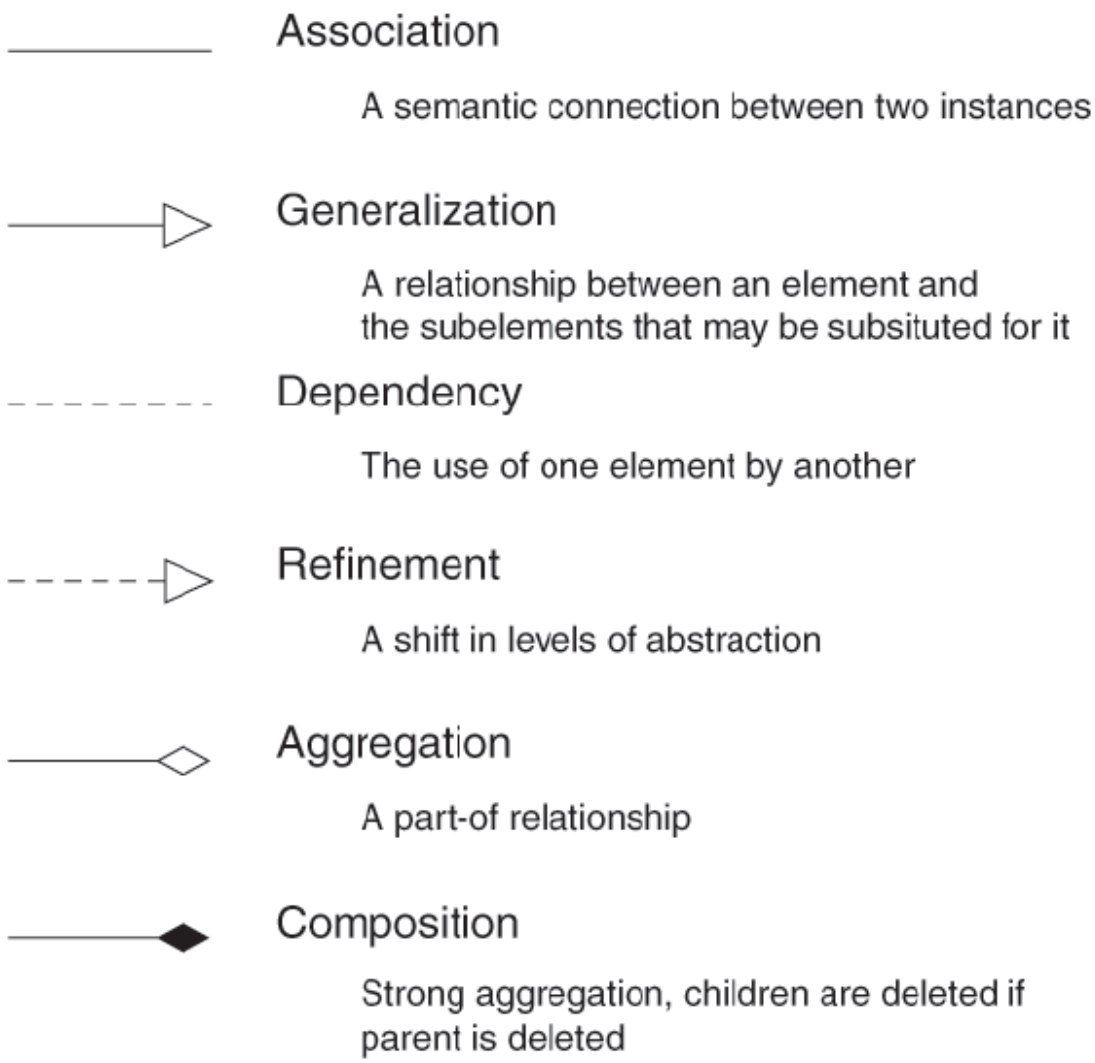


Figure 3: Different kinds of relationships (ISO/TS 19103:2005(E))

Association, composition and aggregation

An association is used to describe a relationship between two or more classes. In addition to an ordinary association, UML defines two special types of associations called aggregation and composition. The three types have different semantics. An ordinary association should be used to represent a general relationship between two classes. The aggregation and composition associations should be used to create part-whole relationships between two classes. A binary association has a name and two association-ends. An association-end has a role name, a multiplicity statement, an optional aggregation symbol. An association-end should always be

connected to a class. Figure 4 shows association. An aggregation association is a relationship between two classes, in which one of the classes plays the role of container and the other plays the role of a containee. Figure 5 shows aggregation. A composition association is a strong aggregation. In a composition association, if a container object is deleted then all of its containee objects are deleted as well. Composition should be used to have the semantic effect of containment and its application construct should be considered within the context of a model. (ISO/TS 19103:2005(E))

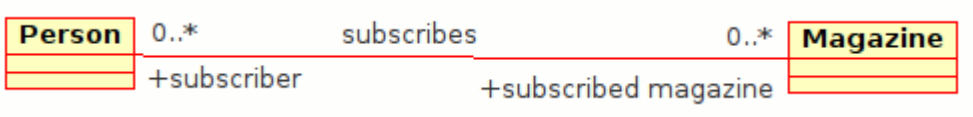


Figure 4: association (URL 3)

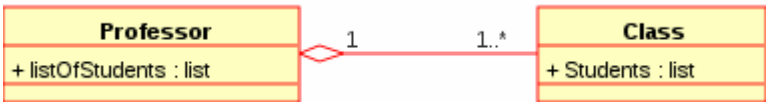


Figure 5: aggregation (URL 3)

Stereotypes

Stereotypes allow designers to extend the vocabulary of UML in order to create new model elements, derived from existing ones, but that have specific properties that are suitable for a particular problem or specialized usage.(URL 2)

The following stereotypes are used: (ISO/TS 19103:2005(E))

a) <<Interface>> a definition of a set of operations that is supported by objects having this interface.

b) <<Type>> a stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A type may have attributes and associations.

c) <<Control>> for a class whose primary purpose is to provide a service and does not represent particular data in itself — a standard UML extension from the Unified Software Development Process.

d) <<Entity>> meaning a class representing information-carrying, potentially persistent objects. A standard UML extension from the Unified Software Development Process.

e) <<Boundary>> meaning a class representing an external interface for a system. A standard UML extension from the Unified Software Development Process.

f) <<Enumeration>> A data type whose instances form a list of named literal values. Both the enumeration name and its literal values are declared. Enumeration means a short list of well-understood potential values within a class.

g) <<Exception>> a signal the underlying execution machinery raises in response to behavioural faults.

h) <<MetaClass>> A class whose instances are classes. Metaclasses are typically used in the construction of metamodels. The meaning of metaclass is an object class whose primary purpose is to hold metadata about another class. For example, “FeatureType” and “AttributeType” are metaclasses for “Feature” and “Attribute”.

i) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with few or no operations whose primary purpose is to hold the abstract state of another class for storage, encoding or transmission.

2.3. Relational paradigm

2.3.1. Language constructs

In relational model: (URL 2)

- A relation is a data structure which consists of a heading and an unordered set of tuples which share the same type.
- When Edgar F. Codd invented the relational model, he generalized the concept of binary relation (mathematical relation) to *n*-ary relation. Relation is a fundamental concept in relational model.
- A relation has zero or more tuples.
- A relation value is an instance of a relation.
- A relation variable (relvar) is a variable which has a relation value.

Figure 6 shows organization of relational model

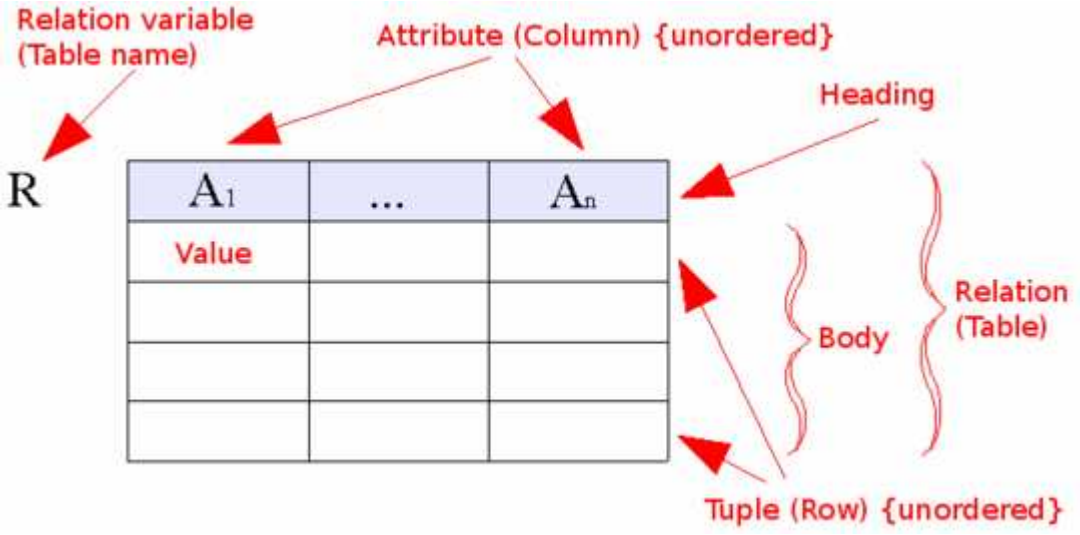


Figure 6: Relational model concepts including relation (URL 4)

The fundamental assumption of the relational model is that all data is represented as mathematical *n*-ary relations, an *n*-ary relation being a subset of the Cartesian product of *n* domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either *true* or *false* (and in particular no third value such as *unknown*, or *not applicable*, either of which are often associated with the concept of NULL).

The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared *constraints* in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives.

The basic relational building block is the domain or data type, usually abbreviated nowadays to *type*. A *tuple* is an ordered set of *attribute values*. An attribute is an ordered pair of *attribute name* and *type name*. An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

A relation consists of a *heading* and a *body*. A heading is a set of attributes. A body (of an *n*-ary relation) is a set of *n*-tuples. The heading of the relation is also the heading of each of its tuples. (URL 2)

2.3.2. Application to databases

A data type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values *true* and *false*, and so on. The corresponding type names for these types might be the strings "int", "char", "date", "boolean", etc. It is important to understand, though,

that relational theory does not dictate what types are to be supported; nowadays provisions are expected to be available for *user-defined* types in addition to the *built-in* ones provided by the system. (URL 2)

Attribute is the term used in the theory for what is commonly referred to as a column. Similarly, table is commonly used in place of the theoretical term relation (though in SQL the term is by no means synonymous with relation). A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column.

A tuple is basically the same thing as a row, except in an SQL DBMS, where the column values in a row are ordered. (Tuples are not ordered; instead, each attribute value is identified solely by the attribute name and never by its ordinal position within the tuple.)

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table. The heading of its assigned value at any time is as specified in the table declaration and its body is that most recently assigned to it by invoking some update operator (typically, INSERT, UPDATE, or DELETE). The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query. (URL 2)

2.4. XML language paradigm

XML is a standard, simple, self-describing way of encoding both text and data so that content can be processed with relatively little human intervention and exchanged across diverse hardware, operating systems, and applications.

XML offers a widely adopted standard way of representing text and data in a format that can be processed without much human or machine intelligence. Information formated in XML can be exchanged across platforms, languages, and applications, and can be used with a wide range of development tools and utilities. (URL 3)

2.4.1. XML benefits

(URL 4)

Simplicity
Information coded in XML is easy to read and understand, plus it can be processed easily by computers.

Openness
XML is a W3C standard, endorsed by software industry market leaders.

Extensibility
There is no fixed set of tags. New tags can be created as they are needed.

Self-description
In traditional databases, data records require schemas set up by the database administrator. XML documents can be stored without such definitions, because they contain meta data in the form of tags and attributes. XML provides a basis for author identification and versioning at the element level. Any XML tag can possess an unlimited number of attributes such as author or version.

Contains machine-readable context information
Tags, attributes and element structure provide context information that can be used to interpret the meaning of content, opening up new possibilities for highly efficient search engines, intelligent data mining, agents, etc. This is a major advantage over HTML or plain text, where context information is difficult or impossible to evaluate.

Separates content from presentation
XML tags describe meaning not presentation. The motto of HTML is: "I know how it looks", whereas the motto of XML is: "I know what it means, and you tell me how it should look." The look and feel of an XML document can be controlled by XSL style sheets, allowing the look of a document (or of a complete Web site) to be changed without touching the content of the document. Multiple views or presentations of the same content are easily rendered.

Supports multilingual documents and Unicode
This is important for the internationalization of applications.

Facilitates the comparison and aggregation of data
The tree structure of XML documents allows documents to be compared and aggregated efficiently element by element.

Can embed multiple data types
XML documents can contain any possible data type - from multimedia data (image, sound, video) to active components (Java applets, ActiveX).

Can embed existing data
Mapping existing data structures like file systems or relational databases to XML is simple. XML supports multiple data formats and can cover all existing data structures.

Provides a 'one-server view' for distributed data
XML documents can consist of nested elements that are distributed over multiple remote servers. XML is currently the most sophisticated format for distributed data.

2.4.2. Language constructs

Every xml document must have a root element and that element is a start of a tree structure that is the basic form of elements in xml document . All elements can have sub – elements, text content and attributes. Figure 7 shows root element and its elements that contain text and attributes. (URL 3)

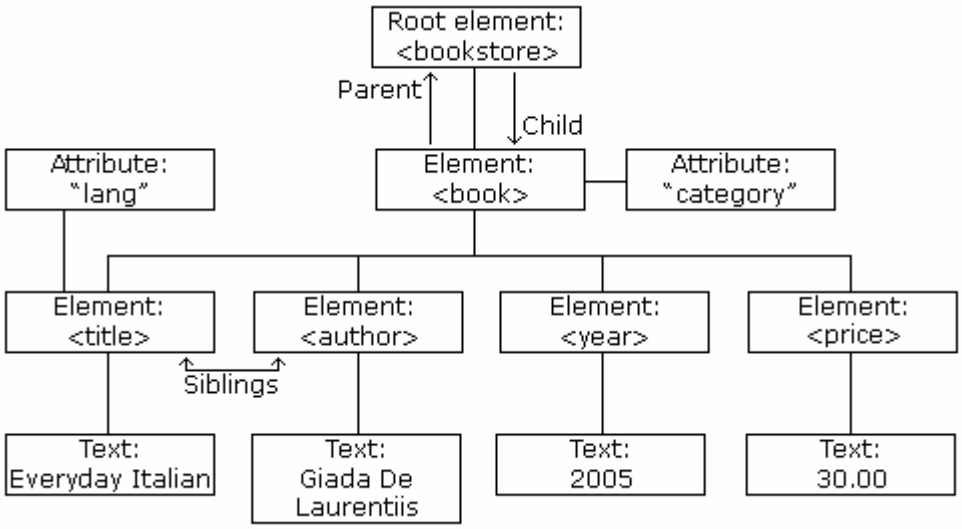


Figure 7: Root element and its elements (URL 3)

In xml, all elements must be properly nested. This means, for example, that if <a> element is opened inside <b> element, it must be closed inside <b> element. Also, elements must follow these naming rules: names can contain letters, numbers, and other characters, but cannot contain spaces; they cannot start with a number, punctuation character or the letters xml. Attributes of an element provide additional information about the element and must always be quoted. (URL 3)

3. ISO standards related to language paradigms

3.1. ISO 19103 – The UML profile for spatial data schemas

ISO 19103:2005 provides rules and guidelines for the use of a conceptual schema language within the ISO geographic information standards. The chosen conceptual schema language is the Unified Modeling Language (UML). ISO 19103:2005 also provides a UML profile that serves for customizing UML models for particular domains. In addition, it provides guidelines on how UML should be used to create standardized geographic information as well as basic parts of UML thet are used within the standard.(URL 1)

3.1.1. Data types (ISO 19103:2005)

The basic data types have been grouped into three categories, as shown in Figure 8:

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.

Data types presented in this standard are defined independently of any particular programming language and can be used to describe interfaces to existing libraries without having to specify the language.

Figure 8: Basic data types (ISO 19103:2005)

3.1.2. Primitive types (ISO 19103:2005)

Primitive data types are fundamental types for representing values. These types include boolean values, numeric values, strings, basic date, time objects, etc. Figure 9 shows primitive types

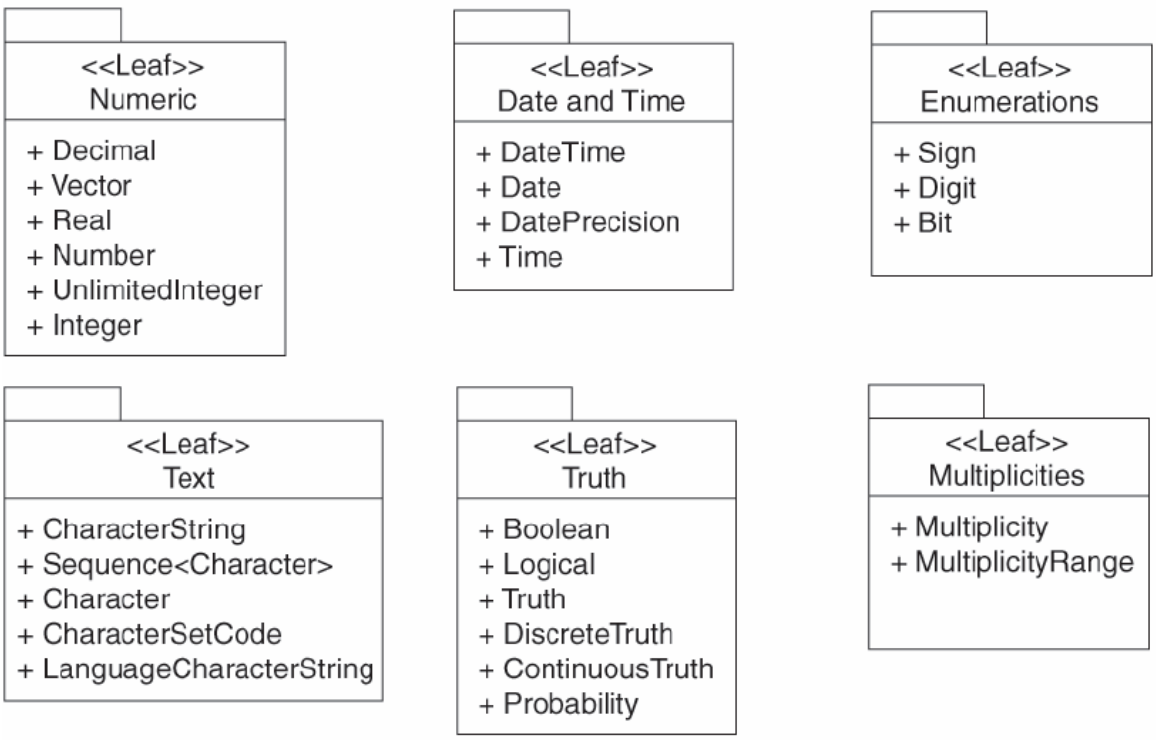


Figure 9: Primitive types (ISO 19103:2005)

Number

Number is the base type for all number data, giving the basic algebraic operations.

Integer

Integer is a signed integer number; the length of an integer is encapsulation and usage dependent. It consists of an exact integer value, with no fractional part.

Decimal

Decimal is a decimal data type in which the number represents an exact value, as a finite representation of a decimal number. This is a class capable of representing the quantity 0.1 with no error.

Real

Real is a floating point number consisting of a mantissa (part of floating – point number that contains its significant digits) and an exponent, which represents a value to a precision given by the number of digits shown, but is not necessarily the exact value. The length of a real is usage dependent. The common binary real implementation uses base 2.

CharacterString

CharacterString is a sequence of characters. The maximum length depends on encapsulation and usage.

Date

Date gives values for year, month and day.

Time

A time is determined by an hour, minute and second.

DateTime

A DateTime is a combination of Date and Time.

Boolean

The boolean type has two values: TRUE or FALSE.

Logical

The logical type has three values: TRUE, FALSE or MAYBE.

Probability

Probability is a number that is greater then or equal to 0 and smaller than or equal to 1.

Multiplicity

Multiplicity defines the lower and upper bounds of the number of possible instances.

3.1.3. Collection and dictionary types (ISO 19103:2005)

There are a few types explained in this section in ISO 19103:2005, but only *sequence* and *dictionary* type are relevant for this thesis and will be explained in the following section.

Sequence

A Sequence is defined to order the elements. The element may be repeated in a sequence. Sequences may be used as either lists or arrays.

Dictionary

Dictionary is similar to an array, and it consists of index type (KeyType) and return type (ValueType). The typical use inside ISO 19103:2005 is to use character strings as the KeyType and numbers as the ValueType.

3.1.4. Enumerated types (ISO 19103:2005)

An enumerated type defines a list of identifiers with memory. Attributes of this type can take values only from the mentioned list. There are a few types explained in

this section in ISO 19103:2005, but only *enumeration* and *codelist* type are relevant for this thesis and will be explained in the following section.

Enumeration

Enumerations are modelled as classes that are stereotyped (see stereotypes) as <<Enumeration>>. An enumeration class can contain only simple attributes which represent the enumeration values. Other information within an enumeration class is void. An enumeration is a user-definable data type whose instances form a list of named literal values. Usually, both the enumeration name and its literal values are declared. Figure 10 shows enumeration.

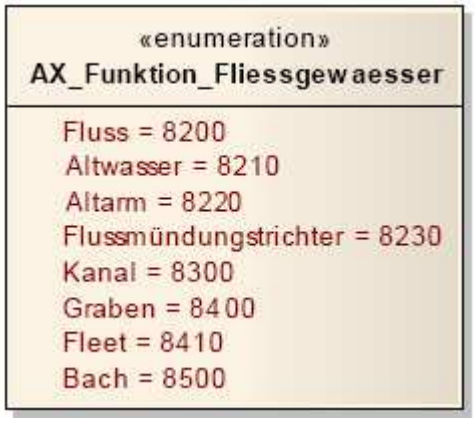


Figure 10: Enumeration (from UML schema)

CodeList

CodeList can be used to describe an open enumeration. This means that it needs to be represented in such a way that it can be extended during system runtime. <<CodeList>> is a flexible enumeration that uses string values for Dictionary type key (see *Dictionary*) and return values as string types. They are useful for expressing a long list of potential values. An enumeration will be used if all elements or the likely values of an element are known. The chosen representation is to represent the value pairs as attributes with a stereotyped <<CodeList>>. Figure 11 shows CodeList.

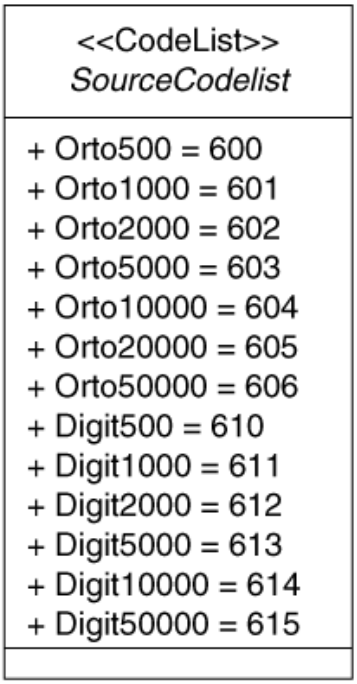


Figure 11: CodeList (ISO 19103:2005)

3.1.5. Names (ISO 19103:2005)

This section is about names for objects and attributes.

NameSpace

NameSpace defines a domain in which “names” given by character strings can be mapped to objects via a getObject operation.

GenericName

GenericName is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.

ScopedName

ScopedName contains a LocalName as head and a GenericName, which might be a LocalName or a ScopedName, as tail.

LocalName

A LocalName references a local object directly accessible from the NameSpace.

TypeName

This is a container for the name of a type.

MemberName

A MemberName is a LocalName that references an attribute, operation, or association role in an object instance or a type description in some schema.

3.1.6. Stereotypes (ISO 19103:2005)

Stereotypes are essential in the creation of the implementation models from the abstract.

In the ISO 19103:2005, the following stereotypes are defined:

a) <<*CodeList*>> is a flexible enumeration that uses string values for Dictionary type key (see *Dictionary*) and return values as string types. They are useful for expressing a long list of potential values. An enumeration will be used if all elements or the likely values of an element are known.

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

c) <<*Union*>> is a type in which the user can choose only one of the listed values (listed as member attributes).

3.2. ISO 19107 – The Standard for Geometries

ISO 19107 specifies conceptual schemas for describing the spatial characteristics of geographic features, and a set of spatial operations consistent with these schemas. In the model defined in this International Standard, spatial characteristics are described by one or more spatial attributes whose value is given by a geometric object (GM_Object) or a topological object (TP_Object). Geometry provides the means for the quantitative description of the spatial characteristics of features, including dimension, position, size, shape, and orientation. The mathematical functions used for describing the geometry of an object depend on the type of coordinate reference system used to define the spatial position. Geometry is the only aspect of geographic information that changes when the information is transformed from one geodetic reference system or coordinate system to another. Topology deals with the characteristics of geometric figures that remain invariant if the space is deformed. (ISO 19107:2003(E))

There are 39 options for application schemas that define types for the instantiation of geometric or topological objects. They are differentiated on the basis of three criteria (ISO 19107:2003(E)):

The first criterion is level of data complexity. Four levels are identified:

- Geometric primitives
- Geometric complexes
- Topological complexes
- Topological complexes with geometric realization

The second criterion is dimensionality. There are four levels for simple geometry:

- 0-dimensional objects
- and 1-dimensional objects
- 0-, 1-, and 2-dimensional objects
- 0-, 1-, 2- and 3-dimensional objects

The third criterion is level of functional complexity. There are three levels.

- Data types only
- Simple operations
- Complete operations

Geometry packages

Geometry root class contains base class GM_Object as a common superclass for all geometries, e.g. every object must always belong to a class GM_Object. Figure 12 shows class GM_Object and its subclasses.

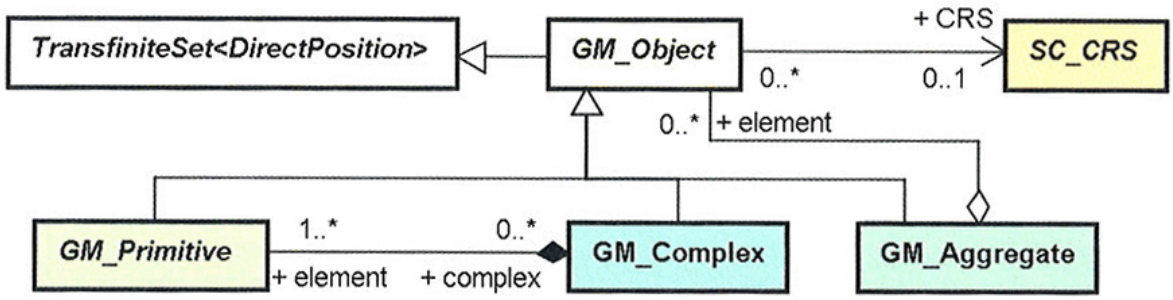


Figure 12: Class GM_Object and its subclasses (Andrae, C., Fitzke, J., Zipf, A.: Spatial Schema - ISO 19107 und ISO 19137 vorgestellt und erklärt. Wichmann Verlag, 2008)

GM_Primitive presents superclass for all primitive classes, e.g. all primitive classes inherit from GM_Primitive class (as shown in Figure 13)

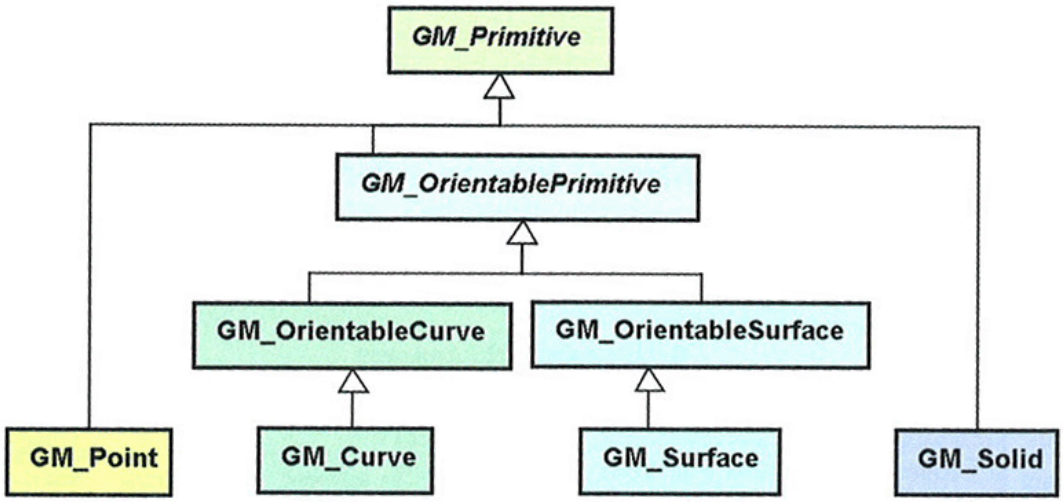


Figure 13: GM_Primitive class and its subclasses (Andrae, C., Fitzke, J., Zipf, A.: Spatial Schema - ISO 19107 und ISO 19137 vorgestellt und erklärt. Wichmann Verlag, 2008)

GM_Complex inherits direct from GM_Object class (see Figure 14) and it contains subclass GM_Composite. GM_Composite is a composition of similar primitives that are geometrically linked together so that the complex can behave like a primitive. Figure 15 shows district complex that is made of two subcomplexes.

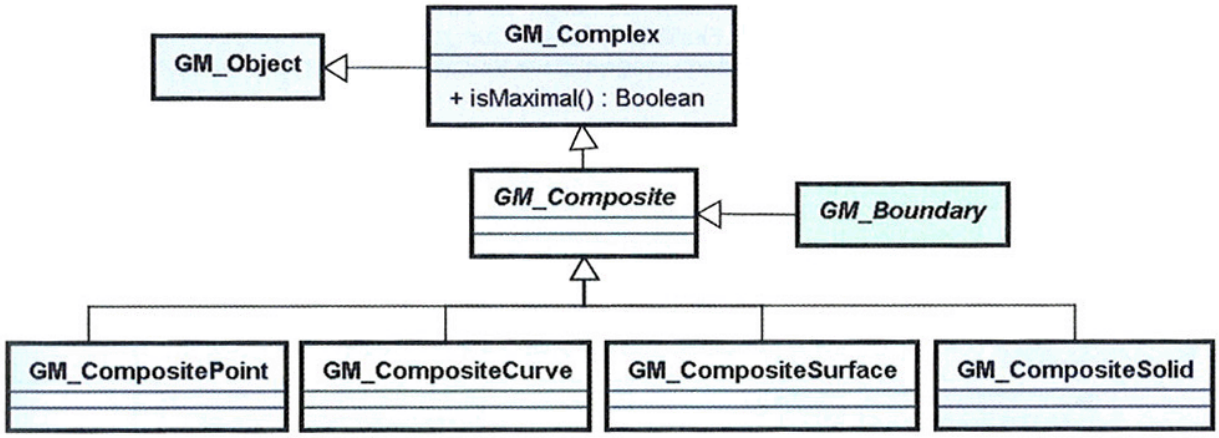


Figure 14: GM_Complex (Andrae, C., Fitzke, J., Zipf, A.: Spatial Schema - ISO 19107 und ISO 19137 vorgestellt und erklärt. Wichmann Verlag, 2008)

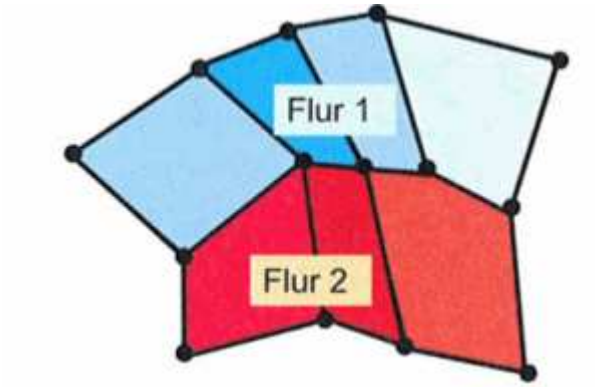


Figure 15: District complex made of two subcomplexes (Andrae, C., Fitzke, J., Zipf, A.: Spatial Schema - ISO 19107 und ISO 19137 vorgestellt und erklärt. Wichmann Verlag, 2008)

GM_Aggregate (see Figure 16) is also a superclass that contains class GM_MultiPrimitive. GM_MultiPrimitive is the root class for all primitive aggregates.

Figure 16 shows root class, superclasses and its subclasses in the composititon that describes their relations and mutual dependencies.

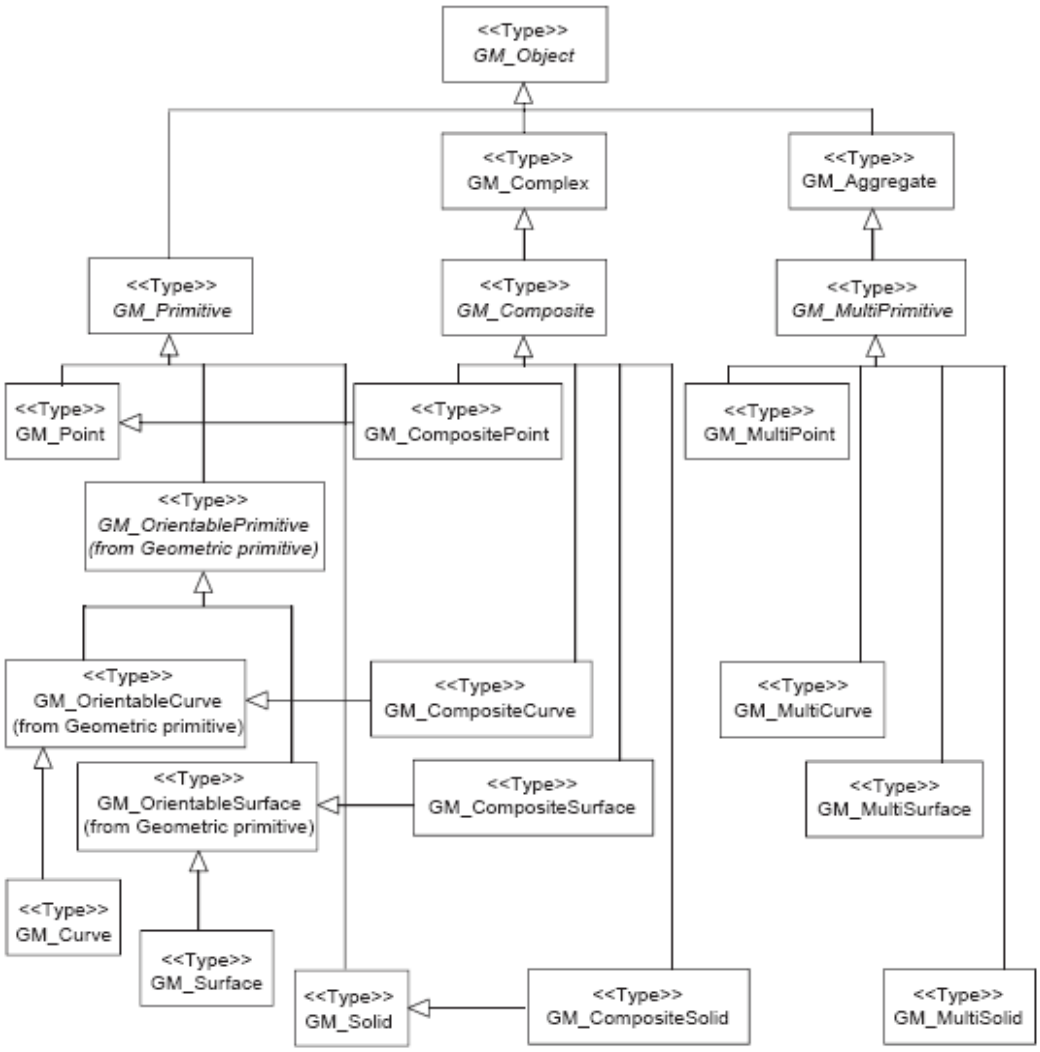


Figure 16: Geometry basic classes with specialization relations (ISO 19107:2003)

3.3. ISO 19109 – The General Feature Model

In ISO 19109 is described a conceptual model that identifies and describes the concepts used to define features. That model is called the General Feature Model (GFM) and it is expressed in UML class diagrams. (ISO 19109:2006 (E))

3.3.1. The purpose of GFM

The things we want to classify are features; the relations between feature types are feature association types and inheritance. Feature types have properties that are feature attributes, feature operations and feature association roles. All these concepts are expressed as UML metaclasses in the GFM.

The GFM is a metamodel of feature types.

The GFM may also serve as the conceptual model of the feature-catalogue structure, but the feature catalogue has additional concepts for documenting feature types. There is a Feature Catalogue Model (FCM) that realizes the GFM concepts and also adds some more concepts (defined in ISO 19110).
An application schema of GFM is expressed in CSL (Conceptual schema language), that is in UML class diagrams. It describes the structure and content of the dataset. The GFM specifies the requirements for the classification of features, but is not a CSL. This means that we have to use an existing CSL to define the application schema. (ISO 19109:2006: E)

3.3.2. Mapping GFM to UML

The GFM defines the structure for classifying features. The mapping from GFM to UML is a one-way mapping; it is not possible to map backwards. For example, the application schema has UML classes. Some of these classes are GFM feature types and some are the datatypes for feature attributes; it is not possible to keep these things apart. The GFM does not define feature-attribute values to the depth that is needed. That is not necessary to do as the GFM only specifies the structure and content of definition of features. The conclusion is that the GFM is a metamodel for definition of features that also is used to define the structure of feature catalogues. The chosen CSL metamodel (e.g. the UML metamodel) is the metamodel for an application schema. As the application schema deals with data representing features, the structure of the GFM has to be kept in mind while creating the application schema. (ISO 19109:2006: E)

3.3.3. The main structure of GFM

Figure 17 shows the concepts used to define types of features (GF_FeatureType).

A feature type is described by its name, description and the following properties (GF_PropertyType):

- feature attributes (GF_AttributeType);
- feature association roles (GF_AssociationRole) characterizing the feature type; and
- defined behaviour of the feature type.

Additional concepts are

- feature associations (GF_associationRole) between the feature type and itself or other feature types;
- generalization and specialization relationships to other feature types; and
- constraints on the feature type

(EN ISO 19109:2006: E)

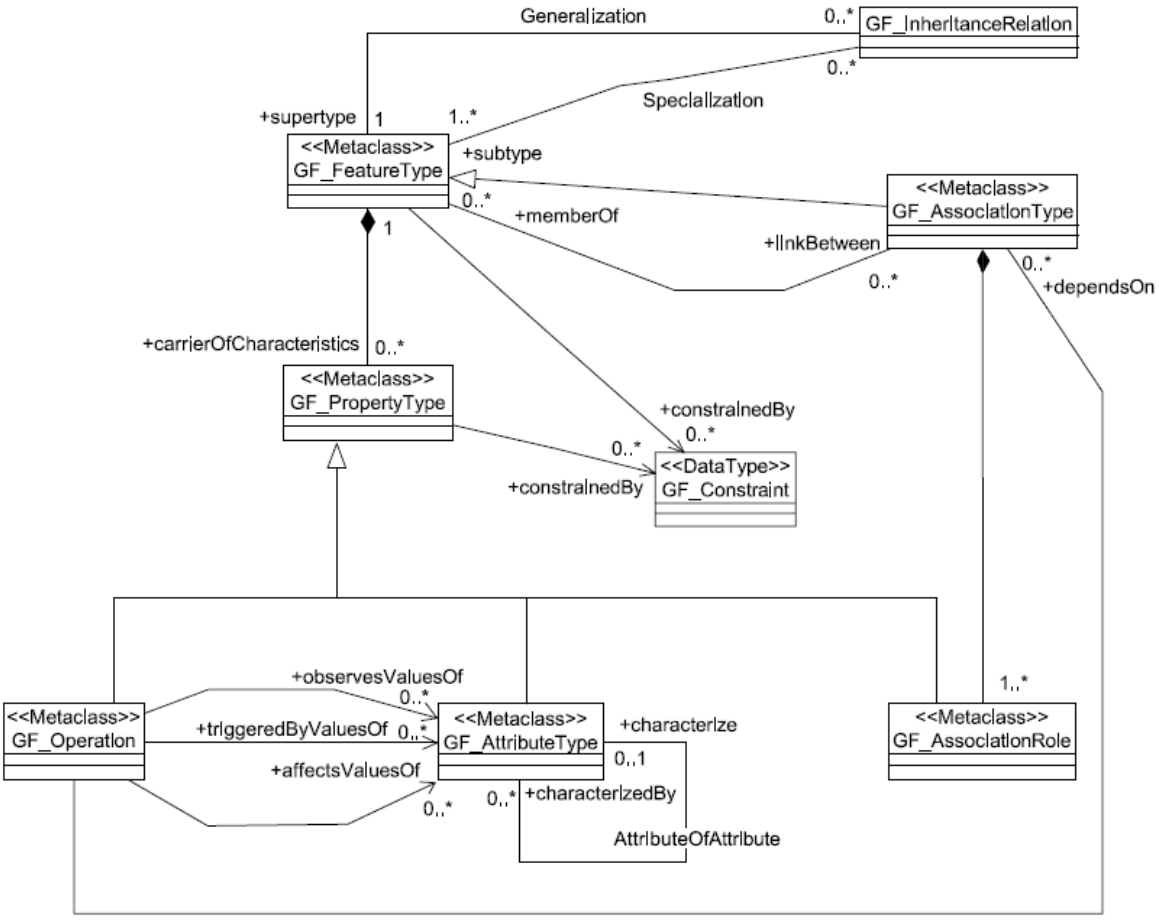


Figure 17: Extract from the General Feature Model (EN ISO 19109:2006: E)

3.4. ISO 19136 – The Geography Markup Language

The Geography Markup Language (GML) was originally developed within the Open Geospatial Consortium, Inc. (OGC).

Geography Markup Language is written in XML Schema using the XML syntax for the description of application schemas as well as the transport and storage of geographic information. (OGC 07-036)

3.4.1. GML schema

The relevant conceptual models for GML schema include those defined in:

- ISO/TS 19103 — Conceptual schema language (units of measure, basic types);
- ISO 19107 — Spatial schema (spatial geometry and topology);
- ISO 19108 — Temporal schema (temporal geometry and topology, temporal reference systems);
- ISO 19109 — Rules for application schemas (features);
- ISO 19111 — Spatial referencing by coordinates (coordinate reference systems);
- ISO 19123 — Schema for coverage geometry and functions (coverages, grids).

The GML schema includes the components (XML elements, attributes, simple types, complex types, attribute groups, etc.) that are described in this International Standard. (OGC 07-036)

3.4.2. GML application schemas

Designers of GML application schemas may extend or restrict the types defined in the GML schema to define appropriate types for an application domain. Non-abstract elements, attributes and types from the GML schema may be used directly in an application schema, if no changes are required.

In accordance with ISO 19109, a GML application schema will be specified in XML Schema and import the GML schema. Design of the schema can be constructed in one of two ways: (ISO 19109:2006 (E))

- By complying with the rules for GML application schemas specified in Clause 21 for creating a GML application schema directly in XML Schema.

- By complying with the rules specified in ISO 19109 for application schemas in UML, and conforming to both the constraints on such schemas and the rules for mapping them to GML application schemas specified in Annex E of this International Standard. The mapping from an ISO 19109 conformant Application Schema in UML to the corresponding GML application schema is based on a set of encoding rules.

Both ways are valid approaches to construct GML application schemas.

3.4.3. XML Schema definition of GML language

The GML schema consists of W3C XML Schema components that define types and declare

- XML elements to encode GML objects with identity,
- XML elements to encode GML properties of those objects, and
- XML attributes qualifying those properties.

A GML object is an XML element of a type derived directly or indirectly from *gml:AbstractGMLType*. From this derivation, a GML object will have a *gml:id* attribute.

A GML property will not be derived from *gml:AbstractGMLType*, and will not have a gml:id attribute, or any other attribute of XML type *ID*.

An element is a GML property if and only if it is a child element of a GML object.

All XML attributes declared in the GML schema are defined without namespace, the only exception is the *gml:id* XML attribute. (OGC 07-036)

3.4.4. Base objects (OGC 07-036)

AbstractObject

An abstract convenience element *gml:AbstractObject* is declared as follows:

```
<element name="AbstractObject" abstract="true"/>
```

This element has no type defined, and is therefore implicitly (in accordance with the rules of W3C XML Schema) an XML Schema *anyType*. It is used as the head of an XML Schema substitution group which unifies complex content and certain

simple content elements used for datatypes in GML, including the *gml:AbstractGML* substitution group.

AbstractGML, AbstractGMLType

The most basic components for representations of identifiable objects are described in the schema as follows:

```
<element name="AbstractGML" type="gml:AbstractGMLType"
abstract="true" substitutionGroup="gml:AbstractObject"/>
<complexType name="AbstractGMLType" abstract="true">
<sequence>
<group ref="gml:StandardObjectProperties"/>
</sequence>
<attribute ref="gml:id" use="required"/> </complexType>
<group name="StandardObjectProperties">
<sequence>
<element ref="gml:metaDataProperty" minOccurs="0"
maxOccurs="unbounded"/>
<element ref="gml:description" minOccurs="0"/>
<element ref="gml:descriptionReference" minOccurs="0"/>
<element ref="gml:identifier" minOccurs="0"/>
<element ref="gml:name" minOccurs="0" maxOccurs="unbounded"/>
</sequence> </group>
```

The abstract element *gml:AbstractGML* is any GML object having identity. It acts as the head of an XML Schema substitution group, which may include any element which is a GML feature, or other object, with identity. This is used as a variable in content models in GML core and application schemas. It is effectively an abstract superclass for all GML objects.

The pairing of *gml:AbstractGML* and *gml:AbstractGMLType* shows a basic pattern used in the GML schema, whereby each GML object type is represented by a global element declaration, which has an associated XML Schema type definition. The name of an element representing a GML object indicates the conceptual meaning of the object. Generic element names in GML include *gml:AbstractObject, gml:AbstractGML, gml:AbstractFeature, gml:AbstractValue, gml:AbstractCoverage, gml:AbstractTopology* and *gml:AbstractCRS*. These other generic elements representing objects are defined elsewhere in this International Standard.

The child XML elements and XML attributes of a GML object are properties of that object. Thus an object represented by an *gml:AbstractGML* element has five non-deprecated properties: *gml:identifier, gml:description, gml:descriptionReference*, *gml:name* and *gml:id*.

GML properties

The term property is used to refer to a GML property, which is any characteristic of a GML object. An element in a GML document or data stream is a GML property if and only if it is a child element of a GML object element. The meaning of each property shall be indicated by the name of the element that instantiates it.

GML objects may have an unlimited number of properties, in addition to those inherited from *gml:AbstractGMLType*. A property may be defined to have either simple or complex content. A property with simple content has an XML Schema simple content type, as illustrated by the case of the standard property elements *gml:description* and *gml:name*. A property with complex content has an XML Schema complex content type. Property elements may use two modes:

- inline: the value of the property is represented directly, as the content of the property element. This method is used by the standard property *gml:name* and may be used for *gml:description*

- by reference: the value of the property is available elsewhere, and is identified by the value of an *xlink:href* attribute on the property element. This alternative method shall be used for the standard property *gml:descriptionReference*

AssociationAttributeGroup

XLink components are the standard method to support hypertext referencing in XML. An XML Schema attribute group, *gml:AssociationAttributeGroup*, is provided to support the use of Xlinks as the method for indicating the value of a property by reference in a uniform manner in GML. This attribute group is defined as follows:

```
<attributeGroup name="AssociationAttributeGroup">
<attributeGroup ref="xlink:simpleLink"/>
<attribute name="nilReason" type="gml:NilReasonType"/>

<attribute ref="gml:remoteSchema"/> </attributeGroup>
with the following definitions from Xlink :
<attributeGroup name="simpleLink">
<attribute name="type" type="string" fixed="simple"
form="qualified"/>
<attribute ref="xlink:href"/>
<attribute ref="xlink:role"/>
<attribute ref="xlink:arcrole"/>
<attribute ref="xlink:title"/>
<attribute ref="xlink:show"/>
<attribute ref="xlink:actuate"/> </attributeGroup>
```

The value of a GML property that carries an *xlink:href* attribute is the resource returned by traversing the link.

The *nilReason* attribute may be used in a property element that is nillable to indicate a reason for a nil value.

abstractAssociationRole, AssociationRoleType

To support the encoding of properties that may have complex content, a basic pattern for property elements is provided in the GML schema as follows:

```
<element name="abstractAssociationRole"
type="gml:AssociationRoleType" abstract="true"/> <complexType
name="AssociationRoleType">
<sequence minOccurs="0">
<any namespace= ##any />
</sequence>
<attributeGroup ref="gml:OwnershipAttributeGroup"/>
<attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

Applying this pattern shall restrict the multiplicity of objects in a property element using this content model to exactly one. An instance of this type shall contain an element representing an object, or serve as a pointer to a remote object.

abstractReference, ReferenceType

In order to support the encoding of properties whose value is provided remotely by-reference, the following components are provided:

```
<element name="abstractReference" type="gml:ReferenceType"
abstract="true"/> <complexType name="ReferenceType">
<sequence/>
<attributeGroup ref="gml:OwnershipAttributeGroup"/>
<attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

The element *gml:abstractReference* is abstract, and thus may be used as the head of a substitution group of more specific elements providing a value by-reference.

Properties representing the same relationship

If the value of an object property is another object and that object contains also a property for the association between the two objects, then this name of the reverse property may be encoded in a *gml:reversePropertyName* element in an *appinfo* annotation of the property element to document the constraint between the two properties. The value of the element shall contain the qualified name of the property element.

```
<element name="reversePropertyName" type="string"/>
```

3.4.5. Standard properties of GML objects

Derivation from AbstractGMLType

XML Schema types for all GML objects derive directly or indirectly from *gml:AbstractGMLType*. This means that all GML objects inherit certain standard properties that are included in the content model of *gml:AbstractGMLType*.

name, identifier

The *gml:name* property provides a label or identifier for the object, commonly a descriptive name. An object may have several names, typically assigned by different authorities. *gml:name* uses the *gml:CodeType* content model. The authority for a name is indicated by the value of its (optional) *codeSpace* attribute. The name may or may not be unique, as determined by the rules of the organization responsible for the *codeSpace*. In common usage there will be one name per authority, so a processing application may select the name from the c*odeSpace* that it prefers.

```
<element name="name" type="gml:CodeType"/>
```

Often, a special identifier is assigned to an object by the authority that maintains the feature with the intention that it is used in references to the object. For such cases, the *codeSpace* shall be provided. That identifier is usually unique either globally or within an application domain *gml:identifier* is a predefined property for such identifiers.

Id

The attribute *gml:id* supports provision of a handle for the XML element representing a GML object. Its use is mandatory for all GML objects.

```
<attribute name="id" type="ID"/>
```

It is of XML type *ID*, so is constrained to be unique in the XML document within which it occurs. An external identifier for the XML element representing the GML object in the form of a URI may be constructed using standard methods (IETF RFC 2396). This is done by concatenating the URI for the document, a fragment separator —# , and the value of the attribute of XML type ID. (OGC 07-036)

3.4.6. Collections of GML objects (OGC 07-036)

AbstractMemberType and derived property types

To create a collection of GML objects that are not all features, a property type shall be derived by extension from *gml:AbstractMemberType*.

```
<complexType name="AbstractMemberType" abstract="true">
<sequence/>
<attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

This abstract property type is intended to be used only in object types where software shall be able to identify that an instance of such an object type is to be interpreted as a collection of objects.

By default, this abstract property type does not imply any ownership of the objects in the collection. The owns attribute of *gml:OwnershipAttributeGroup* may be used on a property element instance to assert ownership of an object in the collection. A collection shall not own an object already owned by another object

GML object collections, AggregationAttributeGroup

A GML object collection is any *gml:AbstractObject* with a property element in its content model whose content model is derived by extension from *gml:AbstractMemberType*.

In addition, the complex type describing the content model of the GML object collection may also include a reference to the attribute group *gml:AggregationAttributeGroup* to provide additional information about the semantics of the object collection. This information may be used by applications to group GML objects, and optionally to order and index them.

```
<attributeGroup name="AggregationAttributeGroup">
<attribute name="aggregationType"
type="gml:AggregationType"/>
</attributeGroup>
```

The allowed values for the *aggregationType* attribute are defined by *gml:AggregationType.*

Xlinks

Xlink components are used in GML to implement associations between objects by reference. GML property elements may carry *Xlink* attributes, which support the encoding of an association relationship by reference, the name of the property

element denoting the target role in the association. The most important *Xlink* component is:

xlink:href
- identifier of the resource which is the the association, given as a URI

The other Xlink components are used to indicate additional semantics of the relationship. The most useful of these are

xlink:role
- description of the nature of the target resource, given as a URI

xlink:arcrole
- description of the role or purpose of the target resource in relation to the present resource, given as a URI

xlink:title
- description of the association or the target resource, given as text

Features

A GML feature is a feature encoded using GML.

AbstractFeatureType

The basic feature model is given by the *gml:AbstractFeatureType*, defined in the schema as follows:

```
<complexType name="AbstractFeatureType" abstract="true">
<complexContent>
<extension base="gml:AbstractGMLType">
<sequence>
<element ref="gml:boundedBy" minOccurs="0"/>
<element ref="gml:location" minOccurs="0"/>
</sequence> </extension> </complexContent> </complexType>
```

The content model for *gml:AbstractFeatureType* adds two specific properties: the value of the *gml:boundedBy* property describes an envelope that encloses the entire feature instance; the value of the *gml:location* property describes the extent, position or relative location of the feature.

AbstractFeature

The element gml:AbstractFeature is declared as follows:

```
<element name="AbstractFeature"
type="gml:AbstractFeatureType" abstract="true"
substitutionGroup="gml:AbstractGML"/>
```

This abstract element serves as the head of a substitution group which may contain any elements whose content model is derived from *gml:AbstractFeatureType*. *Gml:AbstractFeature* may be used to define variables or templates in which the value of a GML property is any feature.

Geometry properties

Application-specific names will be chosen for geometry properties in GML application schemas and they will express the semantics of the value. There are no inherent restrictions in the type of geometry property a feature type may have as long as the property value is a geometry object substitutable for *gml:AbstractGeometry.*

3.4.7. Feature collections (OGC 07-036)

GML feature collections

A GML feature collection is a collection of GML feature instances.

A GML feature collection is any GML feature with a property element in its content model whose content model is derived by extension from *gml:AbstractFeatureMemberType*. In addition, the complex type describing the content model of the GML feature collection may also include a reference to the attribute group *gml:AggregationAttributeGroup* to provide additional information about the semantics of the object collection.

AbstractFeatureMemberType and derived property types

In the purpose of creating a collection of GML features, a property type will be derived by extension from *gml:AbstractFeatureMemberType.*

```
<complexType name="AbstractFeatureMemberType"
abstract="true">
<sequence/>
<attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

The owns attribute of *gml:OwnershipAttributeGroup* may be used on a property element instance to assert ownership of a feature in the collection.

3.4.8. Composite geometries (OGC 07-036)

General representation of composites in GML

The members of a geometric composite will form a homogeneous collection of geometric primitives whose union would be the core geometry of the composite. The member properties in *gml:CompositeCurve, gml:CompositeSurface* and *gml:CompositeSolid* represent a subset of the *gml:element* property of *gml:GeometricComplex.*

As XML Schema does not support the concept of multiple inheritance which is used in ISO 19107 to express the duality of the geometric composites (as an open primitive and as a closed complex) in the GML schema, the composites derive from *gml:AbstractGeometricPrimitiveType* only. However, by using a *<choice>* element in the property type *gml:GeometricComplexPropertyType,* a composite can be used in any property which expects a *gml:GeometricComplex* as its value.

CompositeCurveType, CompositeCurve

```
<complexType name="CompositeCurveType"> <complexContent>

<extension base="gml:AbstractCurveType">
<sequence>
<element ref="gml:curveMember" maxOccurs="unbounded" />
</sequence>
<attributeGroup ref="gml:AggregationAttributeGroup"/>
</extension>
</complexContent>
</complexType>
<element name="CompositeCurve" type="gml:CompositeCurveType"
substitutionGroup="gml:AbstractCurve" />
```

gml:CompositeCurve implements ISO 19107 *GM_CompositeCurve*

A *gml:CompositeCurve* is represented by a sequence of curves such that each curve in the sequence terminates at the start point of the subsequent curve in the list. The curves are contiguous, the collection of curves is ordered. For that reason the *aggregationType* attribute will have the value *sequence.*

CompositeSurfaceType, CompositeSurface

```
<complexType name="CompositeSurfaceType"> <complexContent>
<extension base="gml:AbstractSurfaceType">
<sequence>
<element ref="gml:surfaceMember" maxOccurs="unbounded" />
```

```
</sequence>
<attributeGroup ref="gml:AggregationAttributeGroup"/>
</extension>
</complexContent>
 </complexType>

<element name="CompositeSurface"
type="gml:CompositeSurfaceType"
substitutionGroup="gml:AbstractSurface" />
```

A *gml:CompositeSurface* is represented by a set of orientable surfaces. It is a geometry type with all the geometric properties of a (primitive) surface. Tools used for comparing the paradigms.

4. Tools used for comparing the paradigms

Software used in the thesis is FME. FME allows easy manipulation and validation of data "on-the-fly" to give people immediate access to the exact data they need. It can convert data in over 250 spatial and non-spatial formats and It supports Open Geospatial Consortium (OGC) standards. FME conforms to the relational paradigm. Also, two schemas are used for comparing the paradigms: AAA conceptual schema that conforms to the object – oriented paradigm; and AAA NAS schema that conforms to the xml paradigm.

4.1. The Feature Manipulation Engine (FME): Relational paradigm

FME provides format support for data translation, integration and flexibility in data model transformation and distribution. Data model in FME is designed to cover all possible attribute types and geometries. You can accurately restructure the schema of your data as it moves from the source to the destination, without losing semantic information. You can work with an entire dataset or isolate specific feature types or attributes, and then add transformers to manipulate the data so you get the output that you want. (URL 5)

4.1.1. Primary FME components

FME Workbench

FME Workbench is the primary tool for data translations in FME. You can transform and translate any data you want into a wide range of formats. Using Workbench, you can graphically adjust the way your data flows from its source to the destination. Figure 18 shows multiple input and output formats and details about transformation in FME Workbench. (URL 5)

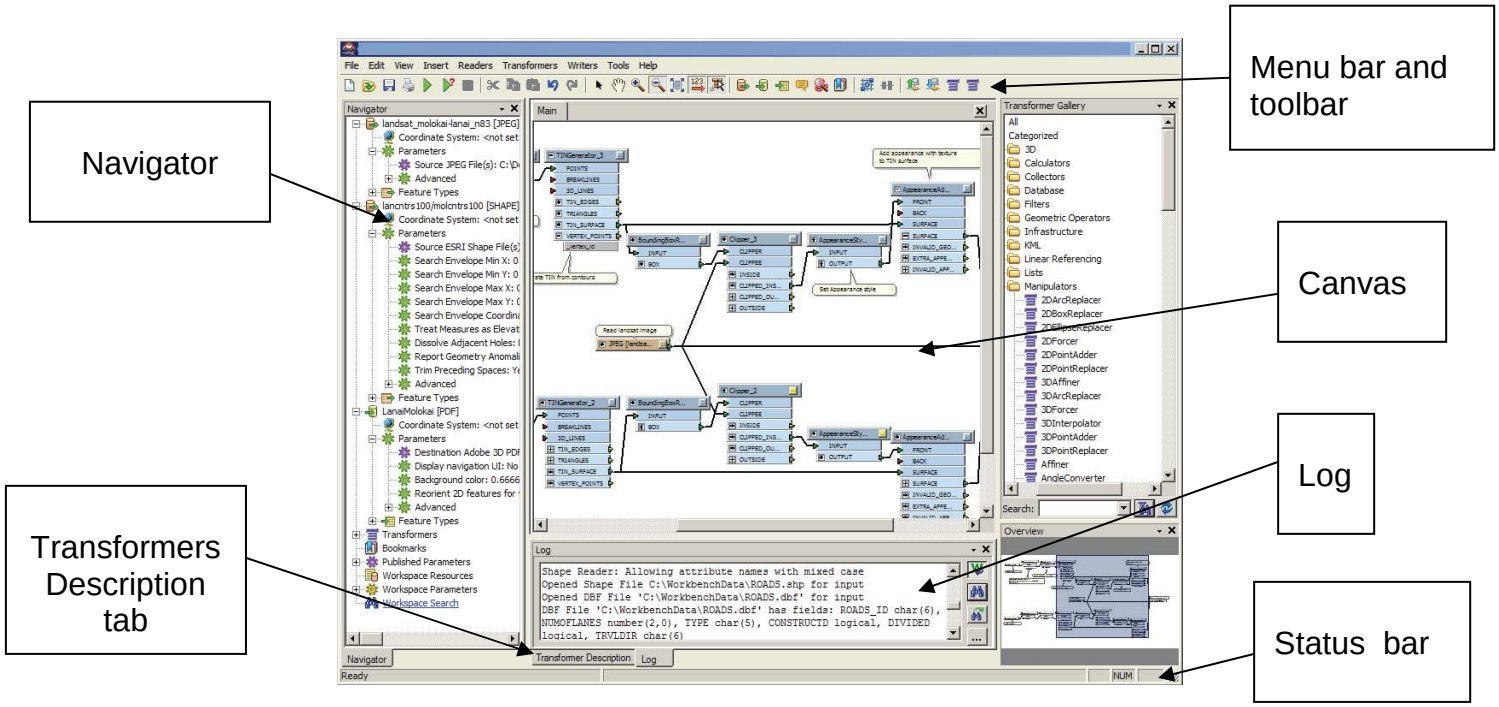


Figure 18: Input and output formats in FME Workbench (FME Tutorial)

4.1.2. FME Universal Viewer

FME Universal Viewer allows quick viewing of data in any of the FME-supported formats. You can use it for data validation and quality assurance. It allows you to preview the data before translation, or review it after translation. Figure 19 shows spatial objects and information about them in FME Universal viewer. (URL 5)

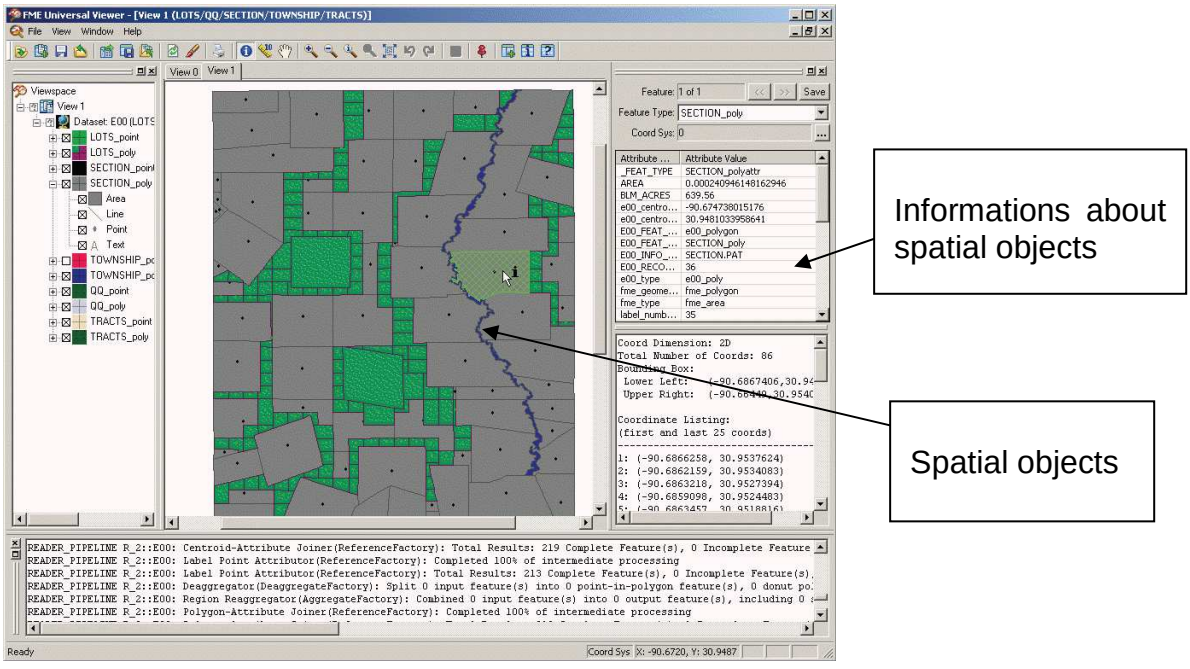


Figure 19: Spatial objects and information about them in FME Universal viewer (FME Tutorial)

4.1.3. Schema mapping

In FME Workbench, one side of the workspace shows the source schema (which already exists) and the other side shows the destination schema (which doesn't yet exist). Schema mapping is the process of connecting the source schema (Reader feature type) to the destination schema (Writer feature type) in a way that ensures the correct Reader feature types are sent to the right Writer feature types and the correct source attributes are sent to the correct destination attributes. (URL 5)

4.1.4. Feature type and attribute mapping

Feature type mapping is the process of connecting Reader feature types to Writer feature types. Attribute mapping is the process of connecting Reader (source) attributes to Writer (destination) attributes. Figure 20 shows feature type and attribute mapping in FME Workbench. Feature type mapping connections are shown with a thick, black line. Attribute mapping connections are shown with a thinner, gray line. (URL 5)

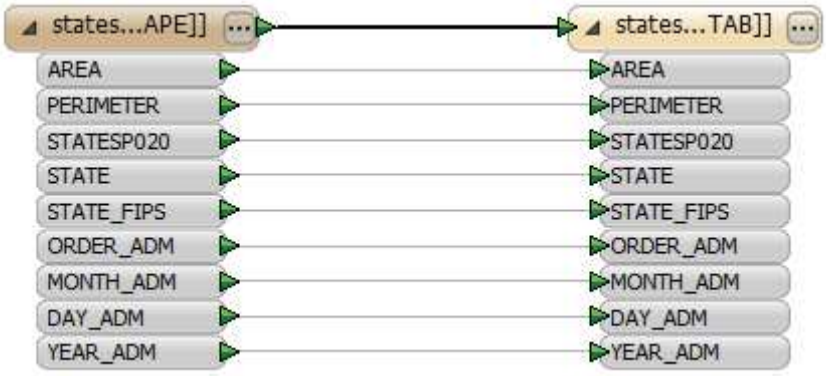


Figure 20: Feature type and attribute mapping in FME Workbench (FME Tutorial)

4.1.5. Feature Type Properties

The Feature Type Properties dialog contains detailed information about the Feature Type.

As shown in Figure 21, the **General tab** under this dialog shows the name of the feature type, allowed geometries and which dataset is used. (URL 5)

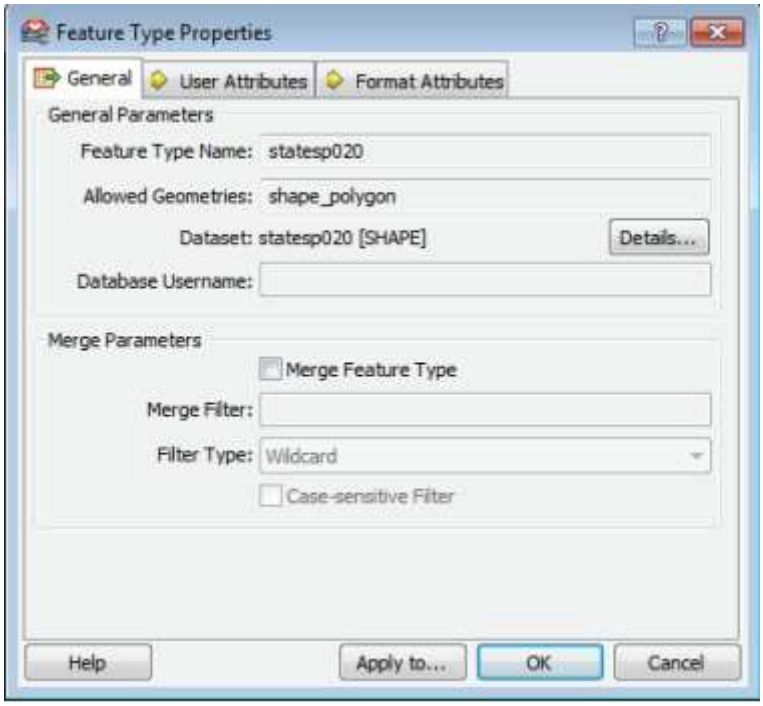


Figure 21: The General tab (FME Tutorial)

As shown in Figure 22, the **User Attributes** tab under this dialog shows a list of attributes that feature type contains. Each attribute is defined by its name, data type, width and number of decimal places. (URL 5)

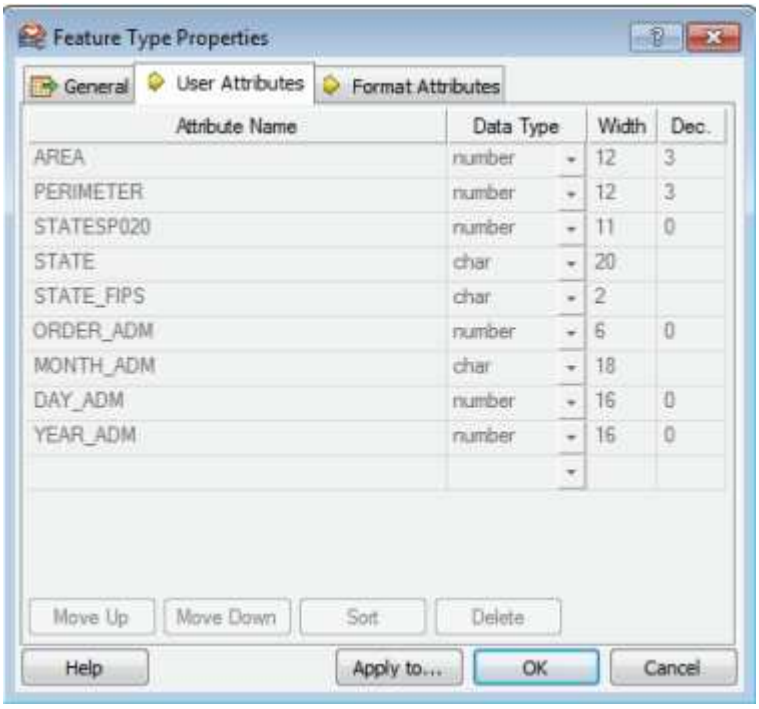


Figure 22: The User Attributes tab (FME Tutorial)

As shown in Figure 23, the **Format Attributes** tab under this dialog shows the attributes that are specific for this format. (URL 5)

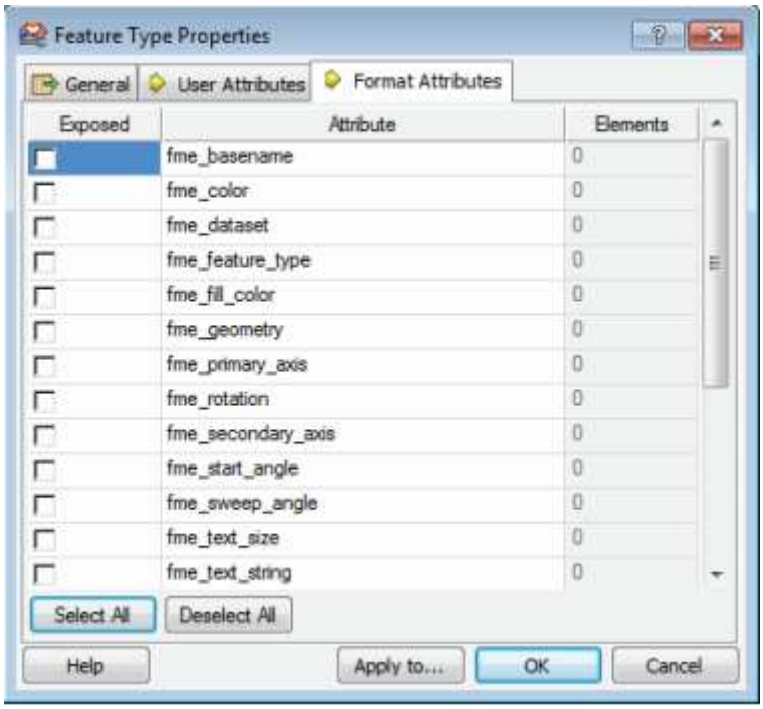


Figure 23: The Format Attributes tab (FME Tutorial)

4.1.6. FME and the Relational paradigm

FME is based on a relational paradigm, as can be seen in the way it stores attributes inside a feature type – in form of a relation. Each feature type presents a relation, and the feature type attributes are the attributes of the relation. Feature attributes are usually a primitive type: integers, floats, characters. Features are deleted as they are retrieved from the database.

4.2. The spatial data and its schemas

The GIS Group at the Technische Universität München currently conducts a project called “Prototypical transformation of spatial data from the cross-border Lake Constance region to INSPIRE”. The project focuses on the integration of spatial data from the Lake Constance region where Germany, Switzerland and Austria share borders. In each country the data are stored in different systems, use different spatial reference systems and transfer formats and above all are based on different schemas determining the structure and semantics of the spatial data.
To carry out the project the mapping agencies of each country provided spatial data from the Lake Constance region. The data used in this thesis was provided by the Landesamt für Vermessung und Geoinformation Bayern (Bavarian Agency for Surveying and Geoinformation) and covers the administrative district Lindau.

The schema of the data is the German Digital Landscape Model (ATKIS Base-DLM) as part of the AFIS-ALKIS-ATKIS Reference Model (AAA). The transfer format is called NAS (Normbasierte Austauschschnittstelle, standards-based data exchange interface).

4.2.1. AAA conceptual schema: Object-oriented paradigm

The German AFIS (Official Control Station Information System), ALKIS (Official Real Estate Cadastre Information System) and ATKIS (Official Topographic Cartographic Information System) are associated with each other in a common reference model, e.g. common application schema (AAA application schema). The AAA application schema is composed of the *AAA basic schema*, the *AAA versioning schema*, 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 UML is chosen for describing the AAA application schema and the feature catalogue (according to ISO 19103 and ISO 19109).

The AAA application schema uses object – oriented modelling approach for AFIS-ALKIS-ATKIS.

The meta model for AAA Basic schema is General Feature Model. ISO 19107 provides geometries for AAA application schema. When an application schema is created, concrete classes have to be derived from these abstract classes by inheritance, which then can be used in the application schema. (Kutzner T., Eisenhut C. (2011): Comparative Studies Regarding Modeling and Schema Translation in the Lake Constance Region, Technische Universität München)

4.2.2. AAA NAS 6.0 schema: XML-paradigm

In purpose of exchanging geoinformations within AAA application schema , a system-independent data exchange and file format is developed. That Format is called NAS (Standards-based data exchange interface). NAS is in fact a GML application schema that additionally considers AAA-specific requirements. NAS encoding rules (ISO 19118) map the conceptual AAA tehnical and basic model on GML application schema and the conceptual schema of NAS operations in the XML schema. There are rules in GML for using XML schema in order to define a application schema, so GML software is able to interpretate AFIS-ALKIS-ATKIS-objects by analysing the GML application schema of the NAS. (GeoInfoDoc version 6.0.1)

5. Differences between NAS (GML) schema and FME

5.1. Data used for the analysis

- AAA NAS 6.0 data file 601_DLM25_clip_n.xml OO/XML paradigm
- The data are provided by the Bavarian Agency for Surveying and Geoinformation (Landesamt für Vermessung und Geoinformation Bayern)
- The data cover the area of the administrative district Lindau, which is situated at the Lake Constance
- The structure of the data is defined by the Digitales Basis-Landschaftsmodell (AAA-Modellierung) Basis-DLM (AAA)

Relevant object types:

Hydrography:

- AX_Fliessgewaesser (Watercourse)
- AX_Gewaesserachse (Water axis)
- AX_Hafenbecken (Basin)
- AX_StehendesGewaesser (Standing water)
- AX_Wasserlauf (Watercourse)

Protected Sites:

- AX_NaturUmweltOderBodenschutzrecht (Nature, environment or soil protection law)

Relevant schemas:

- AAA NAS 6.0 Schema (GML application schema) OO/XML paradigm
- AAA UML schema OO paradigm

5.2. Structure of elements and subelements in GML and FME

Because of the similarity between the structure of the data, I have taken object type AX_Fliessgewaesser and one of its instances for the comparison.

Object type: AX_Fliessgewaesser

GML (NAS):

```
<AX_Fliessgewaesser gml:id="DEBY8424003HM009">
<gml:identifier codeSpace="http:/www.adv-
online.de/">urn:adv:oid:DEBY8424003HM009</gml:identifier>
 <lebenszeitintervall>
        <AA_Lebenszeitintervall>
            <beginnt>2010-02-26T08:31:32Z</beginnt>
          </AA_Lebenszeitintervall>
        </lebenszeitintervall>
        <modellart>
          <AA_Modellart>
            <advStandardModell>Basis-DLM</advStandardModell>
          </AA_Modellart>
        </modellart>
     <istTeilVon xlink:href="urn:adv:oid:DEBY8424003HM000" />
        <position>
          <gml:Surface gml:id="_10502"
srsName="urn:adv:crs:DE_DHDN_3GK4" srsDimension="2">
```

FME (Universal Viewer):

Table 1: Data of one instance of the object type AX_Fliessgewaesser

Feature Type	AX_Fliessgewaesser
Coordinate System	DE_DHDN_3GK4
Attribute Name	**Attribute Value**
fme_geometry	fme_polygon
fme_type	fme_area
gml_id	DEBY8424003HM004
istTeilVon{0}.owns	false
istTeilVon{0}.xlink_href	urn:adv:oid:DEBY8424003HM000

lebenszeitintervall.AA_Lebenszeitintervall.beginnt	2010-02-26T08:31:32Z
modellart{0}.AA_Modellart.advStandardModell	Basis-DLM
nas_transaction	insert
xml_type	xml_area

Since GML is based on XML paradigm, it's hierarchical structure allows elements to own subelements, and also that they themselves can be the subelements of another element. FME is based on a relational paradigm which means that the data corresponds to data stored in tables of relational databases. Because of this characteristic, it places every element and subelement of an object hierarchically equal into one table, and it treats them like attributes of that object. Only the names of these attributes tell us on which hierarchical level these attributes were in GML, since the name of each attribute consists of names of every one of its parent elements. These "sub-names" inside the attribute name are separated with dots.

For example:

FME (Workbench):

Table 2: Some of the attributes of the object type AX_Fliessgewaesser

Attribute Name	Data Type	Width	Dec.
Lebenszeitintervall.AA_Lebenszeitintervall.beginnt	xml_char	30	
Lebenszeitintervall.AA_Lebenszeitintervall.endet	xml_char	30	

GML (Base schema):

```
<element name="AA_Lebenszeitintervall"
substitutionGroup="gml:AbstractObject"
type="adv:AA_LebenszeitintervallType"/>
  <complexType name="AA_LebenszeitintervallType">
    <sequence>
      <element name="beginnt" type="dateTime"/>
      <element minOccurs="0" name="endet" type="dateTime"/>
```

```
</sequence>
```

FME doesn't have the option to store some attribute's default value for the entire model, but has to write down every feature's value of said attribute independently.

For example:
Attribute "*owns*" is not declared manually for every object in GML, but only for those objects whose value is different than the default value, which is "false". (When the value of the "owns" attribute is "*true*", the existence of inline or referenced object(s) depends upon the existence of the parent object.)

An "*xlink:href*" attribute on a GML geometry property means that the value of the property is the resource referenced in the link. It is a location attribute of an "anyURI" type with a defined value in GML and also in FME.

5.3. Geometry types

In this example, gml:Surface is chosen as a geometry object type. According to ISO 19107:2003 it is "a subclass of GM_Primitive and is the basis for 2-dimensional geometry." FME chooses generic attributes fme_type and fme_geometry, both related to the geometry of a feature. For the <position> element FME uses the generic FME attributes fme_geometry and fme_type. Fme_geometry indicates the geometry type of the actual coordinates, whereas fme_type specifies how that geometry is to be interpreted. For gml:Surface, the values of these attributes are fme_polygon and fme_area.

5.4. Change functions

Because GML itself provides no elements for updating operations, the definitions from the OGC's *Web Feature Service* (WFS) are used for this purpose. As well as the transaction mechanism, the WFS specification defines 3 change functions: <Insert> (insert new object), <Replace> (amend, overwrite object) and <Delete> (delete object). This applies to both changes in the attribute values and relations and also to geometric changes.

Example:

```
<wfs:Transaction version="1.0.0" service="WFS">
  <wfs:Insert>
    <AX_Strassenverkehrsanlage gml:id="DEBY83260049W001">

                        ...

</AX_Strassenverkehrsanlage>
    <AX_Strassenachse gml:id="DEBY84230014A003">

                        ...
```

```
  </AX_Strassenachse>
    </wfs:Insert>
 </wfs:Transaction>
```

FME described this function as a format specific nas_transaction attribute with an attribute value insert.

5.5. Displayed attributes

GML (Application Schema):

```
  <element name="AX_Lagebezeichnung"
substitutionGroup="gml:AbstractObject"
type="adv:AX_LagebezeichnungType"/>
  <complexType name="AX_LagebezeichnungType">
    <choice>
  <element name="unverschluesselt" type="string"/>
     <element name="verschluesselt"
type="adv:AX_VerschluesselteLagebezeichnungPropertyType"/>
    </choice>

                  ...

type="adv:AX_VerschluesselteLagebezeichnungType"/>
  <complexType name="AX_VerschluesselteLagebezeichnungType">
    <sequence>
      <element name="land" type="string"/>
      <element     minOccurs="0"     name="regierungsbezirk"
type="string"/>
      <element name="kreis" type="string"/>
      <element name="gemeinde" type="string"/>
      <element name="lage" type="string"/>
    </sequence>
```

FME:

Table 3: Some of the attributes of the object type AX_Fliessgewaesser

Attribute Name	Data Type	Width
name.AX_Lagebezeichnung.unverschluesselt		
name.AX_Lagebezeichnung.verschluesselt.AX_VerschluesselteLagebezeichnung.land		
name.AX_Lagebezeichnung.verschluesselt.AX_VerschluesselteLagebezeichnung.regierungsbezirk		
name.AX_Lagebezeichnung.verschluesselt.AX_VerschluesselteLagebezeichnung.kreis		
name.AX_Lagebezeichnung.verschluesselt.AX_VerschluesselteLagebezeichnung.gemeinde		
name.AX_Lagebezeichnung.verschluesselt.AX_VerschluesselteLagebezeichnung.lage		

GML offers a choice between two mutually exclusive options, while in FME there is no constraint that would indicate that these attributes shouldn't be used simultaneously. Furthermore, FME reads the complete xsd schema even if there are much less data in the actual data set. This means that in FME workbench all attributes inside the feature type are displayed even if they don't have values.

GML:

```
<gml:identifier codeSpace="http:/www.adv-online.de/">
     urn:adv:oid:DEBY84250046B001
</gml:identifier>
```

FME:

gml_identifier
gml_identifier.codeSpace

GML elements become user attributes in FME and attributes of GML elements (here codeSpace) become separate attributes in FME. So when you look at the user attributes alone you don't know how codeSpace is encoded in GML, as attribute of gml:identifier or it could also be encoded as subeelement of gml:identifier.

6. Differences between NAS (GML) schema and UML schema

6.1. UML to GML Conversion rules

6.1.1. Trivial conversion rules

Conversion rules are the rules that explain how one language schema is converted to another language schema. This conversion rules are defined in accordance with ISO- standards (ISO 19103 Conceptual schema language, ISO 19109 Rules for application schema, ISO 19107 Spatial Schema). Below are some trivial and non-trivial conversion rules.

Tables showing examples of trivial conversions of ISO-specific constructs:

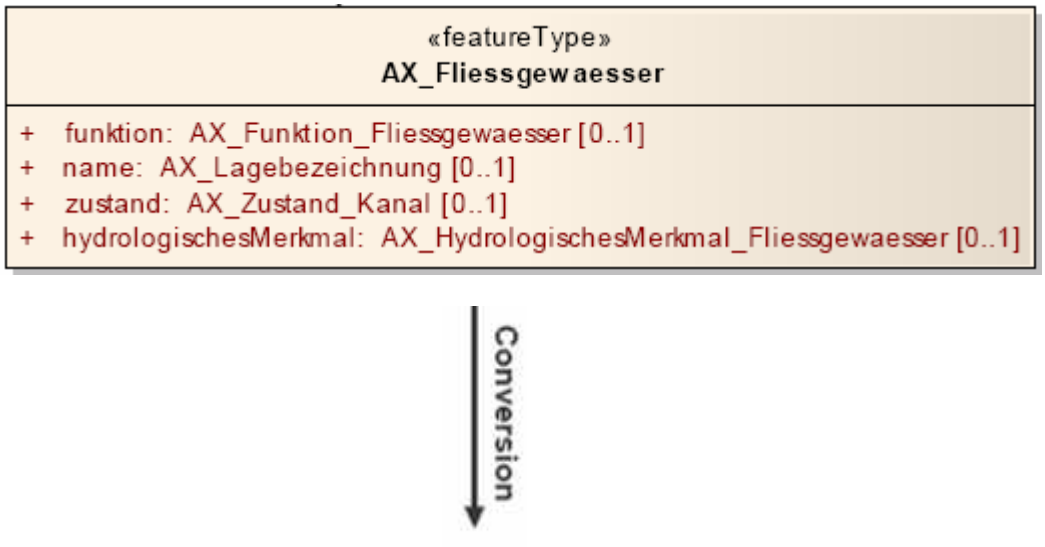
Table 3: ISO basic types converted to XML (R. Grønmo, I. Solheim, D. Skogan, Experiences of UML-to-GML Encoding)

Basic Type according to ISO 19103	XML Schema Basic Type
CharacterString	string
Integer	integer
Date	date
Boolean	boolean
Real	decimal

Table 4: ISO Spatial Schema types converted to GML (R. Grønmo, I. Solheim, D. Skogan, Experiences of UML-to-GML Encoding)

Spatial Type according to ISO 19107	GML Type
GM_Point	PointPropertyType
GM_Curve	LineStringPropertyType
GM_CompositeSurface	PolygonPropertyType

6.1.2. Converting an ISO-conform UML class to a GML Schema




```
<element name="AX_Fliessgewaesser"
substitutionGroup="adv:AX_TatsaechlicheNutzung"
type="adv:AX_FliessgewaesserType"/>
  <complexType name="AX_FliessgewaesserType">
    <complexContent>
      <extension base="adv:AX_TatsaechlicheNutzungType">
        <sequence>
          <element minOccurs="0" name="funktion"
type="adv:AX_Funktion_FliessgewaesserType"/>
          <element minOccurs="0" name="name"
type="adv:AX_LagebezeichnungPropertyType"/>
          <element minOccurs="0" name="zustand"
type="adv:AX_Zustand_KanalType"/>
          <element minOccurs="0" name="hydrologischesMerkmal"
type="adv:AX_HydrologischesMerkmal_FliessgewaesserType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

6.1.3. Non-trivial conversion rules

Non-trivial conversion rules are necessary for encoding inheritance, associations and order of attributes and associations.

Table 5: UML constructs converted to GML (R. Grønmo, I. Solheim, D. Skogan, Experiences of UML-to-GML Encoding)

UML Construct	Conversion to GML
Package	Packages are ignored. (the AAA_Basisschema contains fourteen packages and each package several classes. All classes are put into AAA-Basisschema.xsd, the packages in contrast do not exist in this xsd.)
Class	(1) Classes with stereotype Enumeration or CodeList are converted to: <simpleType name="UMLCLASSNAMEType"> <restriction base="string"> (2) Classes that inherit from a superclass are converted to a complexType and an element declaration: <complexType name="UMLCLASSNAMEType"> <complexContent> <extensionbase="UMLSUPERCLASSNAMEType"> ... <attributeGroupref="gml:AssociationAttributeGroup "/> ... <element name="UMLCLASSNAME" type="UMLCLASSNAMEType" substitutionGroup="UMLSUPERCLASSNAME"/> (3) Classes that do not inherit from other classes and have at least one navigable association to another class, are converted to a GML collection type and an element declaration: <complexType name=" UMLCLASSNAMEType "> <complexContent> <extensionbase="gml:AbstractFeatureCollectionBase

	Type"> ... <element name="UMLCLASSNAME" type="UMLCLASSNAMEType" substitutionGroup="gml:_FeatureCollection"/> (4) All other classes are converted to a GML feature type and an element declaration: <complexType name=" UMLCLASSNAMEType "> <complexContent> <extension base="gml:AbstractFeatureType"> ... <element name="UMLCLASSNAME" type="UMLCLASSNAMEType" substitutionGroup="gml:_Feature"/> All classes that are abstract are converted to an abstract XML element type.
Attribute	(5) Attributes within classes of stereotype Enumeration or CodeList are converted *to* *<enumeration value="UMLATTRIBUTENAME"/>* within the *<restriction base="string">* element within the *<simpleType>* of the corresponding class. Attributes within all other classes are converted to *<elementname="UMLATTRIBUTENAME"* within the *<sequence>* of the *<complexType>* of the corresponding class.
Attribute type	(6) Attribute types are ignored for attributes within classes stereotyped as Enumeration or Codelist. Attribute types that are identified as ISO/TC 211 basic types or ISO/TC 211 spatial types are converted according to Tables 1 and 2. All other types are assumed to be user-defined types within the UML model as class names. The converted type will be *UMLATTRIBUTETYPEType*. (If these types are not defined as class names within the UML model, the GML Schema will not be a legal XML Schema.) The resulting type, *CONVERTED_UMLATTRTYPE*, is inserted as the value of the type attribute within the *<element>* of the

	corresponding attribute *(<element name="UMLATTRIBUTENAME"* *type="CONVERTED_UMLATTRIBUTETYPE)"*
Association	Composition, aggregation and association are treated the same way. Navigable UML class associations are converted to explicit GML featureAssociation types and an element declaration (Two-way associations result in two type and two element declarations): <complexType name="UMLCLASSNAME.ROLENAMEATTHEOTHERCLASSType"> <complexContent> <restriction base="gml:FeatureAssociationType"> <sequence minOccurs="0"> <element ref="UMLOTHERCLASSNAME"/> </sequence> <attributeGroup ref="gml:AssociationAttributeGroup"/> ... <element name="UMLCLASSNAME.ROLENAMEATTHEOTHERCLASS" type="UMLCLASSNAME.ROLENAMEATTHEOTHERCLASSType" substitutionGroup="gml:featureMember"/> This explicit GML featureAssociation type will be part of the sequence of the UML class that has the navigable association: <complexType name=" UMLCLASSNAMEType"> <sequence> <element ref="UMLCLASSNAME.ROLENAMEATTHEOTHERCLASS">
Cardinality	(7) Attribute and association cardinalities are converted to values of the minOccurs and maxOccurs attributes within the corresponding <element>.Integer values are converted to themselves, * is converted to the xml value: unbounded.

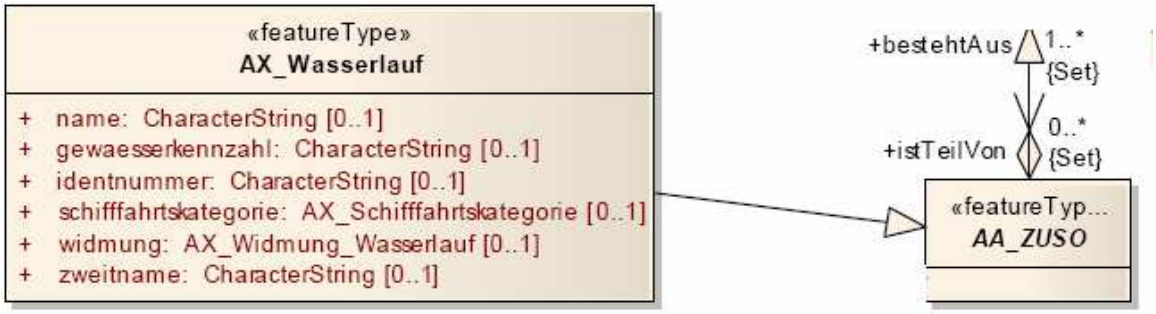
Inheritance	(2) UML class inheritance is converted to XML element type inheritance by <extension> elements: <complexType name="UMLSUBCLASSNAMEType"> <complexContent> <extension base=" UMLSUPERCLASSNAMEType"> Multiple inheritance is not supported.
Other UML constructs (including operations)	Ignored.

Examples for the rules listed above using data from : AAA NAS 6.0 Schema (GML application schema) OO/XML paradigm and AAA UML schema OO paradigm

(1)


```
<simpleType name="AX_Widmung_WasserlaufType">
  <restriction base="string">
    <enumeration value="1310"/>
    <enumeration value="1320"/>
    <enumeration value="1330"/>
    <enumeration value="1340"/>
  </restriction>
</simpleType>
```

(2)




```
  <element name="AX_Wasserlauf"
substitutionGroup="adv:AA_ZUSO"
type="adv:AX_WasserlaufType"/>
  <complexType name="AX_WasserlaufType">
    <complexContent>
      <extension base="adv:AA_ZUSOType">
```

(3)

```
<element abstract="true" name="AA_Objekt"
substitutionGroup="gml:AbstractFeature"
type="adv:AA_ObjektType"/>
  <complexType abstract="true" name="AA_ObjektType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
```

(4)

```
<element name="AX_Fliessgewaesser"
substitutionGroup="adv:AX_TatsaechlicheNutzung"
type="adv:AX_FliessgewaesserType"/>
  <complexType name="AX_FliessgewaesserType">
```

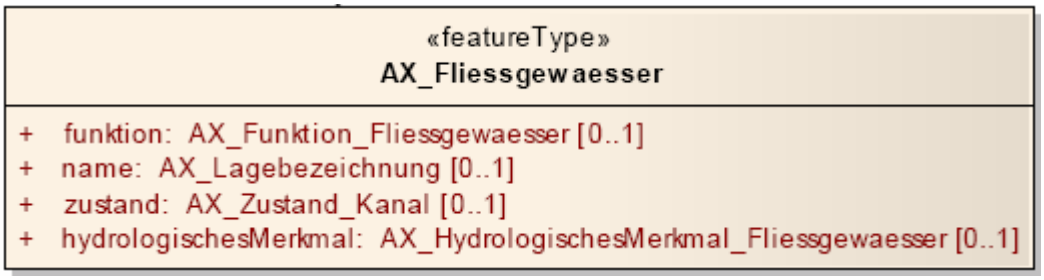
(5)

```
 <simpleType
name="AX_HydrologischesMerkmal_FliessgewaesserType">
    <restriction base="string">
      <enumeration value="2000"/>
```

(6)

```
<element name="AX_Fliessgewaesser"
substitutionGroup="adv:AX_TatsaechlicheNutzung"
type="adv:AX_FliessgewaesserType"/>
```

(7)




```
        <sequence>
          <element minOccurs="0" name="funktion"
type="adv:AX_Funktion_FliessgewaesserType"/>
          <element minOccurs="0" name="name"
type="adv:AX_LagebezeichnungPropertyType"/>
          <element minOccurs="0" name="zustand"
type="adv:AX_Zustand_KanalType"/>
          <element minOccurs="0" name="hydrologischesMerkmal"
type="adv:AX_HydrologischesMerkmal_FliessgewaesserType"/>
        </sequence>
```

6.1.4. Impact of the General Feature Model on this analysis

A feature is encoded as an XML element with the name of the feature type (GF_FeatureType becomes an XML element). Other identifiable objects are encoded as XML elements with the name of the object type.

Each feature attribute and feature association role is a property of a feature. Feature properties are encoded in an XML element (GF_AttributeType and GF_AssociatonType become properties of a feature and are encoded in an XML element).

The GML model has a straightforward representation using the UML profile used in the ISO 19100 series of International Standards (defined in ISO/TS 19103). This is described in detail in Annex D and Annex E, but can be summarized approximately and briefly as follows: (ISO 19136)

Features are represented

- in UML by objects, where the name of the feature type is used as the name of the object class;
- in GML instances by XML elements, where the name of the feature type is used as the name of the element. This means that the features are syntactically XML elements, but semantically objects.

Feature properties are represented

- in UML by association roles with feature type classes, and attributes of feature type classes, where the property semantics are given by the association role name or attribute name;
- in GML instances by sub-elements (known as property elements) of feature elements, where the property semantics are given by the property element name. This means that the feature properties are syntactically XML sub-elements, but semantically attributes of the feature object.

The result is a layered XML document, in which XML elements corresponding to features, objects or values occur interleaved with XML elements corresponding to the properties that relate them. The function of a feature, object or value in context can always be determined by inspecting the name of the property element which directly contains it, or which carries the reference to it.

<<FeatureType>> is a new stereotype which does not appear in ISO/TS 19103 or ISO 19109, and is used to indicate that the type is a realization of GF_FeatureType and a specialization from AbstractFeature. (ISO 19136)

The Feature Model concepts are closely related to object-oriented data modelling, involving inheritance, polymorphism, labelled associations, etc, and GML

documents may be modelled using UML. In fact, in modelling application domains ISO 19103 requires the use of a conceptual schema language that supports object-oriented principles, and strongly recommends UML. Of course, since XML is a static data encoding, "operations" or "methods" are not available in GML descriptions of features. But apart from this, the Feature Model can be understood to be just an application of object-modelling.

As said, the General Feature Model supports the concept of 'operations'. However, implementation of features using W3C XML Schema only allows definition of a static XML document; defining only the structure and properties. There is no mechanism allowing the description of operations that a feature can be invoked on it. The concept of 'processing affordance' is proposed to enable description of operational interfaces. The semantics of processing affordance are equivalent to those of an interface in Object Oriented programming. The 'interface' defines a declaration of intent, describing a series of operations that can be invoked. Where a feature supports the 'interface', it implies that that feature is able to provide sufficient information to execute the operation described in the interface. Or expressed mathematically: the feature has the attributes p,q,r to support f(p,q,r). Furthermore, a feature may be able to support multiple interfaces, thus enabling polymorphism. While the properties of features can be defined in W3C XML schema, only limited definition of relationships between feature types (and other objects) is allowed (i.e. single inheritance, association). This means the capability to make associations between features and the processing affordance (interface or interfaces) must be provided by other means. One possible methodology is to define the processing affordance as an object in a registry (i.e. a catalogue with appropriate governance) and associate it with feature instances or feature types. (URL 6)

7. Attachments

Data and schema used in the process of making this work are attached on this media. They are logicaly structured by meaning.

Table 6: The content of the attached media

Nr.	File	Content
1.	Master Thesis.doc	Text of the master thesis
2.	Master Thesis.pdf	Text of the master thesis
3.	601_DLM25_clip_n.xml	Data used in practical part
4.	UML schema	UML schema of the stated data

8. Summary

During the working procces on this thesis, I adopted a new knowledge and skills on the field of language paradigms and their application within various languages for modelling and transformation. The comparisons that are the result of this thesis are usefull not only for better understanding of the tools that are used in them – FME, GML and UML schema, but for resolving and connecting different kind of complex problems such as transformation or modeling data.

Rules provided by ISO 19100 series of standards present the basis for this work and are thoroughly described and listed.

Two crucial languages for manipulating spatial data are – UML (Unified Modeling Language) and GML (Geography Markup Language). They are described and compared, so the reader can understand them separately and in the same time can see diversity between their language paradigms: UML is based on Object – oriented, while GML is based on Xml paradigm.

The final products of this work are differences between FME and GML (NAS) schema that compare language paradigms on which they are built on and differences between GML schema and UML schema providing not only the differences between their paradigms, but encoding rules for their schemas. In addition, the impact of the General Feature Model, as a metamodel of feature types, on this analisys is described.

The next step of this work might include transformation languages into comparison of laguage paradigms. When transforming from one model to another the transformation language might be in a different paradigm and it should be analysed if differing language paradigms of modelling and transformation languages can cause problems in the transformation.

References:

Sinan Si Alhir. (1998): "UML in a Nutshell : A Desktop Quick Reference". O'Reilly Associates, Inc.

Open Geospatial Consortium Inc. (2007-08-27): OpenGIS Geography Markup Language (GML) Encoding Standard, version 3.2.1

Geographic information – Rules for application schema (ISO 19109:2005); English version EN ISO 19109:2006

International standard ISO 19107 (2003-05-01): Geographic information — Spatial Schema

Technical specification ISO 19103 (2005-07-15): Geographic information — Conceptual schema language

Kutzner T., Eisenhut C. (2011): Comparative Studies Regarding Modeling and Schema Translation in the Lake Constance Region, Technische Universität München

Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV) (2009): Documentation on the Modelling of Geoinformation of Official Surveying and Mapping (GeoInfoDoc), Version 6.0.1

Geir Myrind (2006): Application Schemas–From conceptual models implementation, StandardizationWorkshop, Nordic GIS Conference in Helsinki

R. Grønmo, I. Solheim, D. Skogan (2001): Experiences of UML-to-GML Encoding, SINTEF Telecom and Informatics, Oslo, Norway

Andrae, C., Fitzke, J., Zipf, A. (2008): Spatial Schema - ISO 19107 und ISO 19137 SINTEF vorgestellt und erklärt. Wichmann Verlag

LIST of URL-s:

URL 1. UML Basic, http://www.ibm.com/developerworks/rational/library/769.html, (31. 08. 2011.)

URL 1. Xml paradigm, http://communities.softwareag.com/ecosystem/communities/public/Developer/webmethods/products/tamino/faq/XMLStarter/XMLBasics.html, (26. 08. 2011.)

URL 3. XML Tutorial, http://www.w3schools.com/xml/ (23.08.2011.)

URL 4. XML Benefits, http://communities.softwareag.com (23.08.2011.)

URL 5. FME Tutorial , http://docs.safe.com/fme/pdf/FMEGettingStarted.pdf (19.08.2011.)

URL 6. The General Feature Model, https://www.seegrid.csiro.au/wiki/AppSchemas/SchemaMapping, (23.08.2011.)

List of figures

List of tables

europass

Europass Curriculum Vitae

Personal information

First name(s) / Surname(s)	**Helena Petković**
Address(es)	50, Vinka Maglice, 22000, Šibenik, Hrvatska
Telephone(s)	+ 385 (0)22212043 Mobile: +385 (0)981688439
Fax(es)	
E-mail	hpetkovic@geof.hr
Nationality	Croat
Date of birth	26.01.1988
Gender	Female

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 Differencial 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 2004
Occupation or position held	Assistent
Main activities and responsibilities	- Assisting English teacher in translating to foreign guests - Guiding guests
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 work groups. I was on Erasmus excange students program in München for three months where I wrote my Master Thesis.

Computer skills and competences

- Experienced with Microsoft Office programs
- Experienced with various GIS applications
- Little experience in FME (Feature Manipulation Engine)

Artistic skills and competences

- I am very good at drawing

Other skills and competences

Additional information

Personal interests: I like to play all kinds of sports, especially tennis and table tennis. As for regular exercise, I do jogging every day. My favourite way of socialising is playing cards with my friends. My long-term goal is to travel as much as possible and to find a job that will fulfil me as a person.