

Title:

<b>Title:</b>
Title: Specification – Humboldt Editor
<b>Author(s)/Organisation(s):</b>
Jan Schulze Althoff (ETH, Zürich)
<b>Working Group:</b>
<b>References:</b>

<b>Short Description:</b>
<b>Keywords:</b>

<b>History:</b>			
Version	Author(s)	Status	Comment
001	JSA	new	
002	JSA	rfc	Changes to a new structure
003	AB, JSA		Comments on XMI included

Title:

## Table of contents

1 Introduction.....	3
1.1 Purpose of the document.....	3
1.2 Used Terminology.....	3
1.3 Abbreviations and Definitions used in this document.....	3
1.4 Standards used in this document.....	4
2 Enterprise Viewpoint.....	4
2.1 Overview.....	4
2.2 Actors.....	4
2.3 Business process overview.....	5
2.4 Scenario Integration.....	5
2.5 Use Cases.....	5
2.5.1 Use Case 1 – Generate or modify Data Model.....	5
2.5.2 Use Case 2 – Generation of GML-Schema and Data Specification Documents.....	6
2.5.3 Use Case 4 – Reverse engineering.....	6
2.6 Graphical User Interface.....	7
2.6.1 Standard Layout .....	7
2.6.2 Graphical modelling and multi-image models.....	7
2.6.3 Iterative Creation Process.....	8
2.6.4 Reuse of existing models.....	8
2.6.5 Context sensitive tools.....	8
2.6.6 Graphical validation.....	9
3 Computational Viewpoint.....	9
4 Information Viewpoint .....	9
5 Literature.....	14

Title:

# 1 Introduction

## 1.1 Purpose of the document

The document describes primarily the functionality of the suggested „Humboldt Editor“. The focus is set to a short enterprise viewpoint, to describe the general functionality and to the informational viewpoint to explain, which elements the data model are build on.

## 1.2 Used Terminology

There is a quite varying terminology in the field of modelling. To ensure the correct understanding, some terms are described shortly for the usage in this document.

### Data model:

Data model = ‘model’ of the (geographic) data that is stored and/or exchanged<sup>1</sup>.

To be more concrete the abstraction of the real (geospatial) world with object types, relations, attributes, structuring elements (packages), additional restrictions. And the model is guessed to be human readable.

For this specification: The data model is such a model, which is created and managed by the Humboldt Editor.

### Data specification:

A data specification is a data model with additional information on quality criteria, extends, etc. In this document, a data specification is a ISO 19131 document.

### Encoding:

The process of encoding is describes the transferring of a conceptual data model into a logical or physical schema. Most important is GML schema, but other formats are also possible.

## 1.3 Abbreviations and Definitions used in this document

GML	Geography Markup Language	See <a href="http://www.opengeospatial.org/standards/gml">http://www.opengeospatial.org/standards/gml</a>
UML	Unified Modelling Language	See <a href="http://www.uml.org/">http://www.uml.org/</a>
EPSG	European Petroleum Survey Group (now part of the International Association of Oil & Gas Producers)	The EPSG defined a codelist for most of the used geospatial reference systems ( <a href="http://www.epsg.org/">http://www.epsg.org/</a> )
OGC	Open Geospatial Consortium	See <a href="http://www.opengeospatial.org">http://www.opengeospatial.org</a>

---

<sup>1</sup> From Marian’s Table on data model etc.

Title:

## 1.4 Standards used in this document

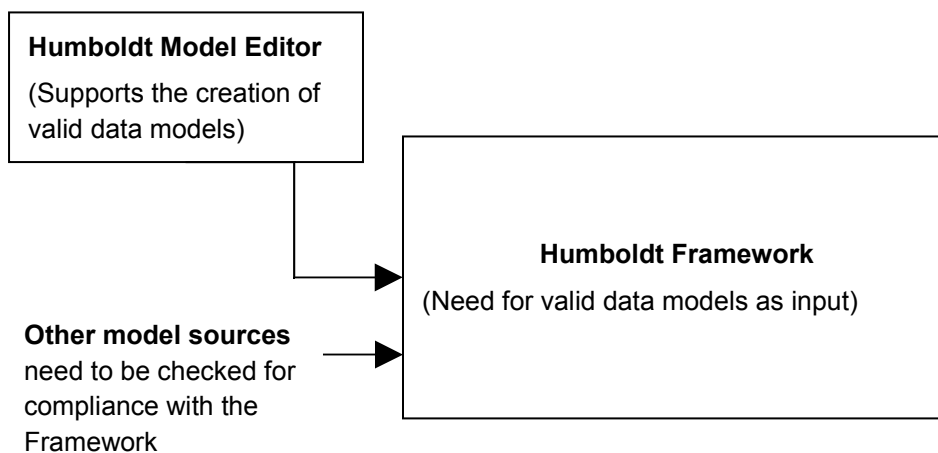
# 2 Enterprise Viewpoint

## 2.1 Overview

The Humboldt Editor is tool to support the user in producing suitable datamodels/data-specifications for geospatial demands and to support in the analysis of existing data sources. Due to the importance, also automatic GML schemas should be generated in reproducible manner. There exist tools for data modelling, mainly standard UML editors, and there are some GML generators available. But that combination is difficult to be used by non-geospatial experts (see WP3).

The Datamodels and the encoding are valuable input to the harmonisation process. In that sense, the Humboldt Editor is not designed as integrated component in a technical framework, but primary a standalone element to help the user in that task (see Figure 1).

A tight coupling to other components, especially to a repository of datamodels and to an editor for model transformation, might be useful.



**Figure 1: The Humboldt Model Editor as tool for the creation of compliant models for the framework**

## 2.2 Actors

As complete standalone desktop application, the Humboldt editor only interacts with human users. These users act in their role as "data modeller". Referring to the classification of user roles in Humboldt, following roles user-roles might interact with the editor:

- "Data Provider":  
the person providing data to the system.
- "Data Integrator":  
the person responsible to define the transformations between source and target data.

For the integrated view, the Editor might interact with other Services, especially with a repository of existing models "Model Repository".

Title:

## 2.3 Business process overview

The Humboldt Model Editor is an interactive component that helps users in the creation of consistent and valid data models to be used within the HUMBOLDT framework. The user benefits directly in having a customized GUI for that purpose and a direct model check (according to the required data model for the framework).

## 2.4 Scenario Integration

The Humboldt Model Editor is a supporting tool for the fulfilment of input requirements of the Humboldt framework. The scenarios will use the framework for the harmonisation of geospatial data. One topic is the harmonisation of data structures. This means the transformation of data structured according to a source model to data structured according to a target model.

The creation of these source and target models according to the requirements of the Humboldt framework is supported by the Humboldt Model Editor (but the models can also be created by other procedures).

For the ERISKA scenario, the model editor might be used to create the target models of the base datasets (railroad network, street network, river network). The created model will be used by framework components for the offline creation of the demanded datasets.

## 2.5 Use Cases

### 2.5.1 Use Case 1 – Generate or modify Data Model

- 1 Description  
The user creates a new datamodel.
- 2 Actors  
Data Integrator
- 3 Initial Conditions  
-
- 4 Final Result  
New datamodel in user readable and machine processable form.
- 5 Processing  
User creates a new model-file  
Modifies the model  
System provides user with error or alerts in the model  
Stores the model

Modification A:

The user opens an existing datamodel and modifies it.

Modification B:

The user includes an existing datamodel by referencing it<sup>2</sup>.

Modifications C and D need the integration of user interfaces to a model repository. It might be better to split into 2 organisational steps.

---

<sup>2</sup> External Formats for Conceptual Models should be supported. Especially XMI for the exchange of UML models will be useful.

Title:

Modification C:

Use a model repository to include or modify an existing data model.

Modification D:

Deploy the created model to a repository with adequate metadata.

## **2.5.2 Use Case 2 – Generation of GML-Schema and Data Specification Documents**

- 1 Description  
Produce exact and reproducible GML-Schema and Data Specifications from the defined data model.
- 2 Actor  
Data Integrator
- 3 Initial Conditions  
A valid data model is created or loaded (Use Case 1)
- 4 Final Result  
GML Schema and a prefilled template for data specification documents
- 5 Processing  
User starts the processing on an active datamodel  
User provides additional information for the creation of schema and specification (storage location, namespaces, schema locations, metadata...)  
Model is checked  
GML Schema and/or Data Specification is generated

## **2.5.3 Use Case 4 – Reverse engineering**

- 1 Description  
For given GML Schema, which follows a defined set of constraints – e.g. specific geometric types, specific id encoding, ... - the Editor generates the according data model
- 2 Actor  
Data Integrator
- 3 Initial Condition  
A GML Schema is available with all imported schemas
- 4 Final Result  
A conceptual datamodel representing the GML Schema in a human readable way
- 5 Processing  
User starts processing on the GML Schema  
Schema is checked for validity, available references and additional constraints like geometric types, ...  
Datamodel is generated

## **2.6 Graphical User Interface**

The Humboldt Editor offers only a limited functionality that can be expressed in standard use cases. It offers mainly a user interface to help formalising concepts of the domain data models.

---

Title:

## 2.6.1 Standard Layout

A simple standard interface with menu bars, navigation window, main editing window, context menu, detailed properties, help functionality, etc. should be offered.

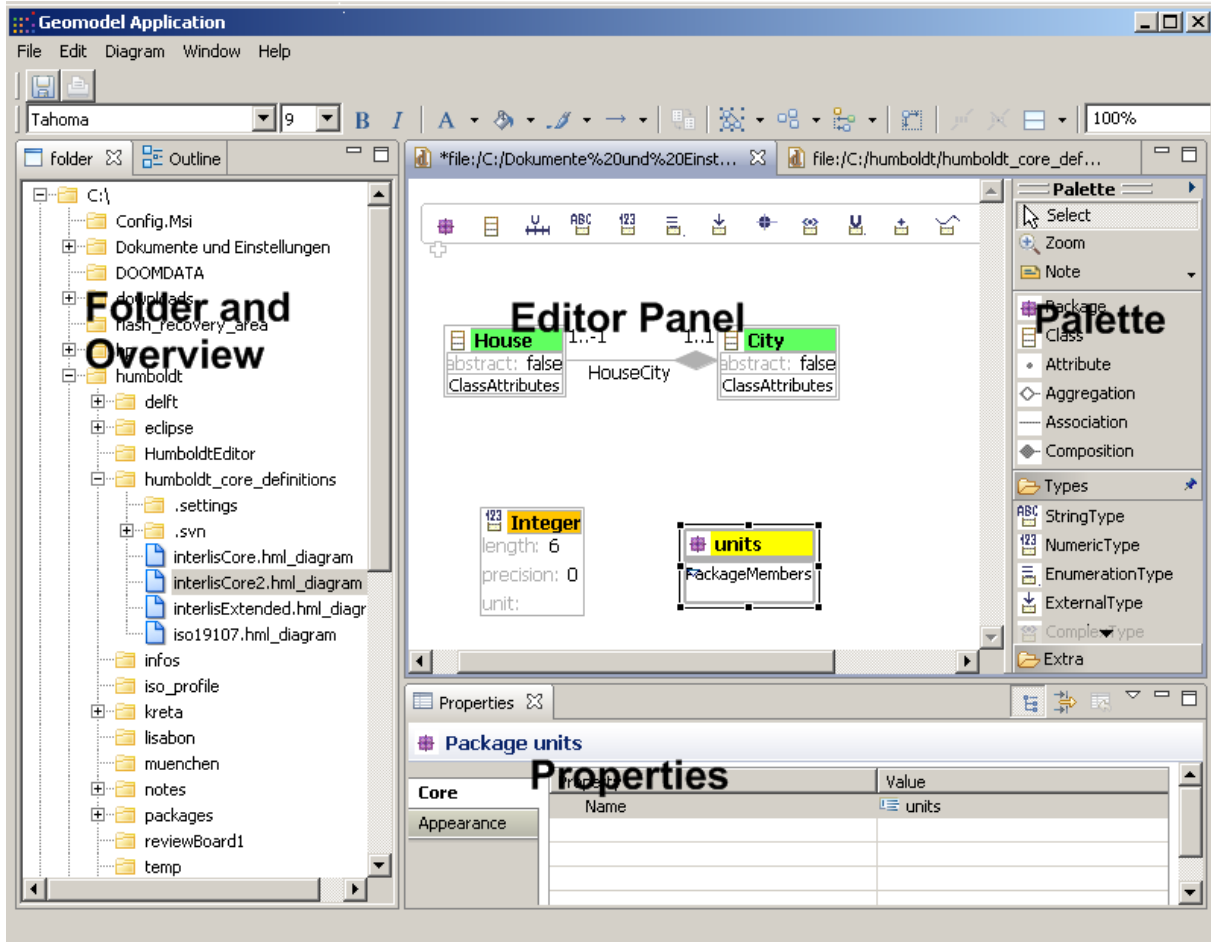


Figure 2: Standard layout from the prototype implementation.

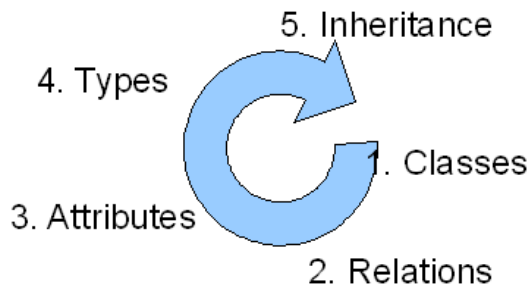
## 2.6.2 Graphical modelling and multi-image models

For the modelling part, a graphical representation is used. This allows direct mapping of relations and inheritances but is growing extremely for large and detailed models. Grouping elements (packages) are needed and multipanel editing is necessary (see Figure 2).

## 2.6.3 Iterative Creation Process

The tool supports the iterative refinement process, this means that a model can first be defined coarse and be refined stepwise; it is not necessary to complete one element after the other.

Title:

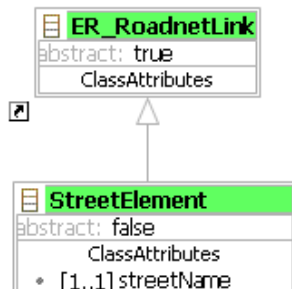


**Figure 3: Iterative modelling process**

The iterative definition process also reflects the fact, that mostly not the complete range of definition possibilities is needed. E.g. the definition of specific types is only needed in specific situations, when standard types need further refinement.

### 2.6.4 Reuse of existing models

As described in the use cases, the reuse and referencing of existing data models is highly desirable. Actually most existing data models and especially the INSPIRE models will be delivered as UML 2 class diagrams. A support for the exchange via XMI is useful. The referenced elements are visible in the graphical model and marked as external element.

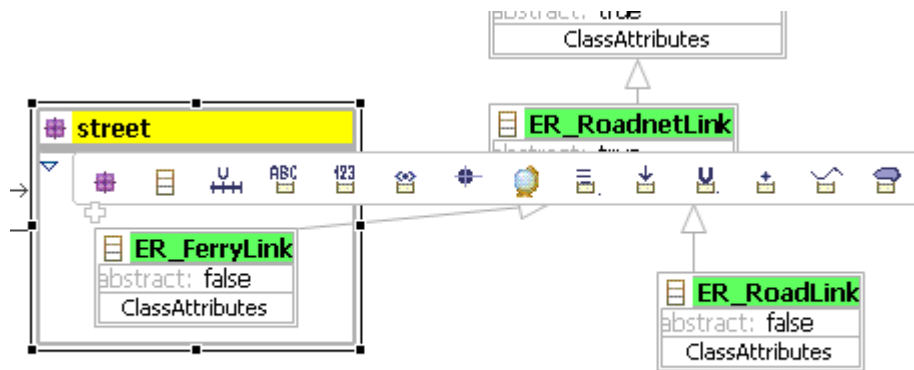


**Figure 4: The external element is marked with a small arrow and can be used directly in the data model.**

### 2.6.5 Context sensitive tools

Only valid options for specific model elements are offered to the user, to generate consistent, valid models with maximum feedback to the user.

Title:



**Figure 5: Context menu with all elements for packages**

## 2.6.6 Graphical validation

The validation of the data model offers a graphical marker on the affected elements.

## 3 Computational Viewpoint

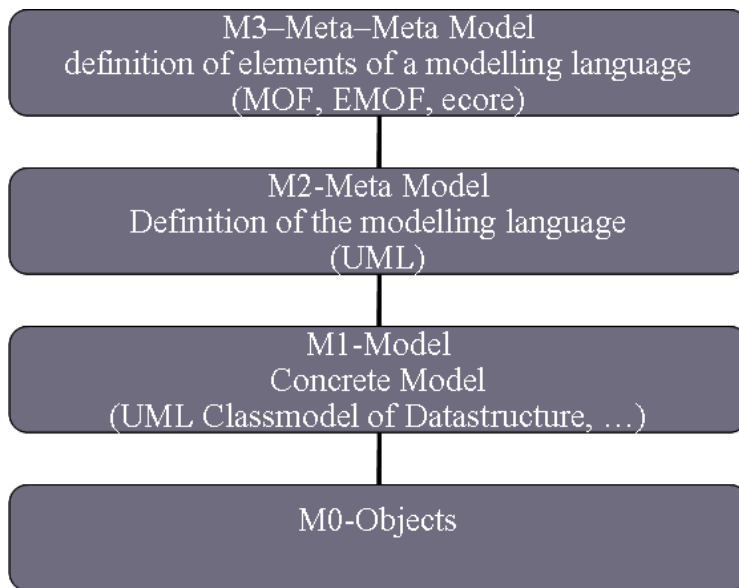
The Humboldt Editor is completely developed in a declarative way. Sources and components are made with the Eclipse Graphical Modelling Framework[Eclipse Foundation, 2008] the transformation Engine is developed with OpenArchitectureWare [Open Architecture Ware, 2008].

The internal structure of the generated source can be found in the documentation.

## 4 Information Viewpoint

The internal model of the Humboldt Editor is the „Meta-Model“of the Data-Models. This means, the internal model is build of all elements that are needed to describe a geospatial data model precisely enough to perform the targeted operations. In the OMG nomenclature, the internal model is located on Level M2.

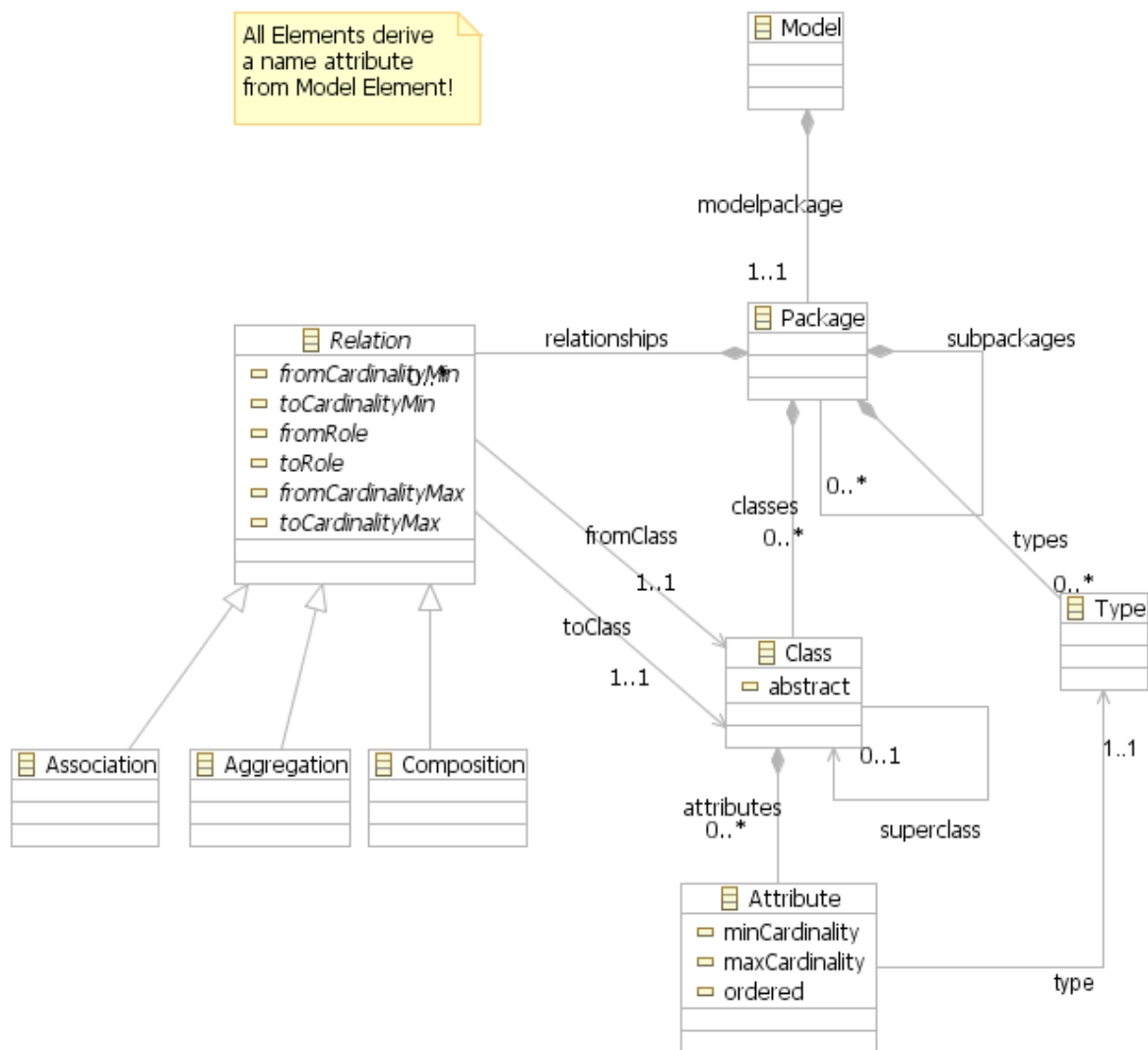
Title:



**Figure 6: The levels of abstraction according to the Model Driven Architecture approach of OMG [OMG - Object Management Group, 2003]**

The Data models should mainly describe Objects by defining Classes, the way they are characterised by Attributes and their relations to each other by inheritances and different relations. These elements are directly taken from the UML definition [OMG - Object Management Group, 2007a, b].

Title:



**Figure 7: Metamodel of the main modelling elements.**

For geospatial modelling the usage of correct types is crucial. To allow simple definition of types, 6 type elements were defined, as a blueprint for the definition of types in the editor.

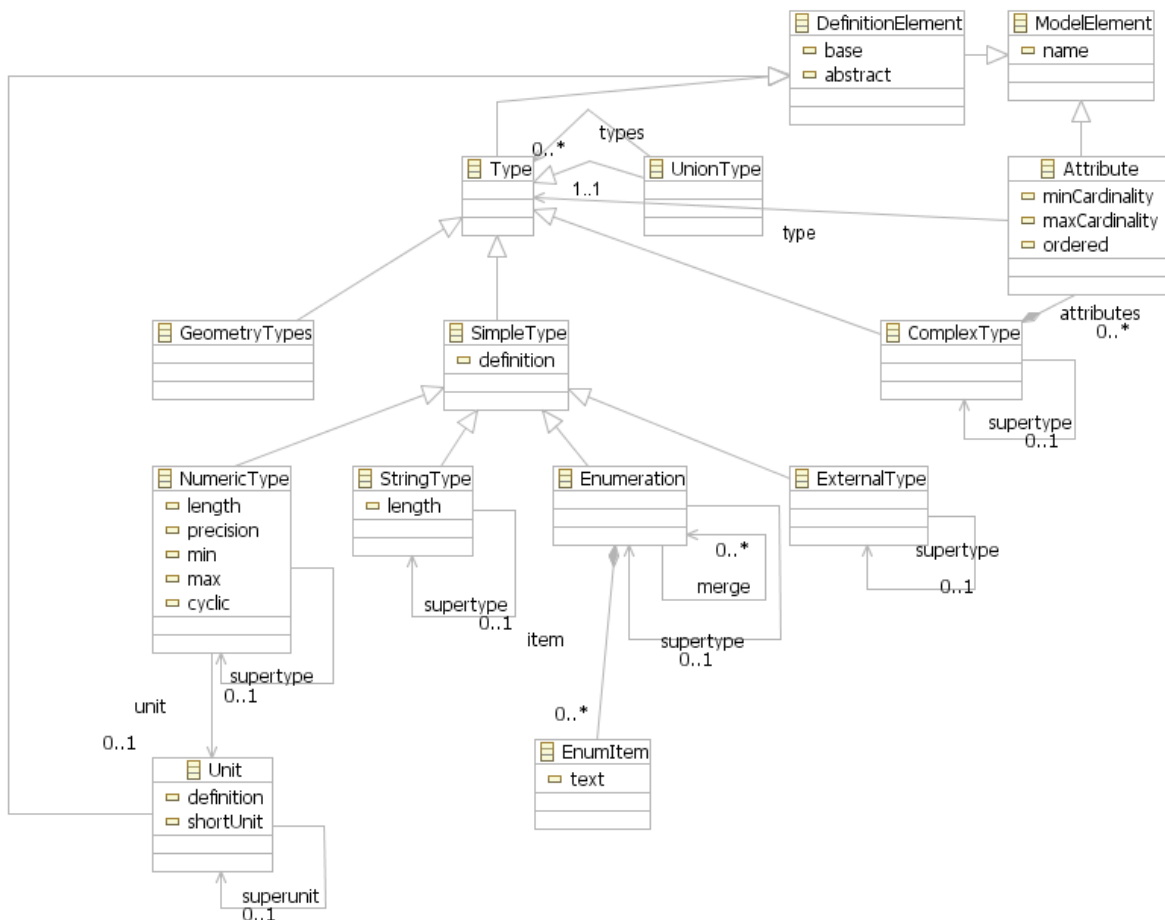
The **simple types** include:

- Numeric types with or without units and defined precisions. Numeric types can only be inherited when refining the super definition. I.e. reducing the value interval between min and max or reducing the precision of the numeric type.
- String types are simply defined by the string length. Subtypes can only have reduced length.
- Enumeration types offer despite to the simple inheritance with a reduction of Enumeration items also a merge construct to extend Enumerations. (This mechanism needs to be checked for usability and also it must be evaluated if an codelist element needs to be introduced)

The **complex types** or data types follow the differentiation made by ISO, to split into elements, which can be instantiated for themselves (class) or only act as a data container for reuse (complex type).

The helper **union type** allows offering choices of different types, e.g. either textual types or enumerations, or different types of geometric types.

Title:



**Figure 8: General type system (TODO: remove external type)**

The geometry types are split up into the 3 single geometry subtypes and 4 multi geometry types:

- **Point Type**, representing a point with a given position
- **Multi Point**, for collections of points forming one feature (it is common practise to use multiple geometry types instead of higher cardinalities for the according attribute)
- **Line Type**, a collection of points. (Note: The cardinality of 1..1 does reflect the cardinality of point types! This means, a line is characterised by 2 or more ordered Point Elements, but these Point Elements are of one Point Type). Currently there is only the possibility of linear internodials, further internodials might be useful.
- **Multi Line Type**, a collection of lines
- **(Abstract) Polygon Type**, defines a closed lines to form a polygon (with or without exclaves and islands). Similar to Line Types, the cardinality of the defining Line Types is one, as it reflects the type and not the number of concrete line elements. Abstract Polygon Type is subtyped into:
  - **Polygon Type**, the standard polygons
  - **Multi Polygon Type**, a collection of types
  - **Surface type**, the subdivision of the complete area, which is quite often used in cadastral domains or in land cover data. The attribute “MaxOverlap” indicates the

Title:

tolerance which is acceptable for real world applications, the unit of this maximum overlap (measured as maximum orthogonal size of the split polygon) is the same as defined in the underlying Coordinate type.

- **Multi Surface Type**, a collection of surfaces (not be used)

Depending on the modelling target (only generate a GML schema, derive other Conceptual or logical Model, derive a Data Specification Document, ...), resp. the required precision and quality of the data model, it might be sufficient to offer one instance of these types in a common model on level M1. But if refinement is required, following elements help definitions within the geospatial domain.

**Coordinate Systems** are simply defined by a name and free text definition elements. The definition elements will help the using software to deal with the coordinate system. As currently no concrete software is dealing with these elements, OGC well known text for coordinate systems can be used or epsg codes.

The **Axis** elements own a name and a description; they are strongly bound to the coordinate system and should have at least one possible unit.

The **Coordinate** elements are the base element for modelling within a geospatial domain. They describe a position within a 1d, 2d or 3d space and have 1, 2 or 3 elements of a coordinate numeric type.

The **Coordinate Numeric Type** inherits all elements form the numeric type, but links directly to one axis element and bridges coordinates to the defining coordinate system.

Title:

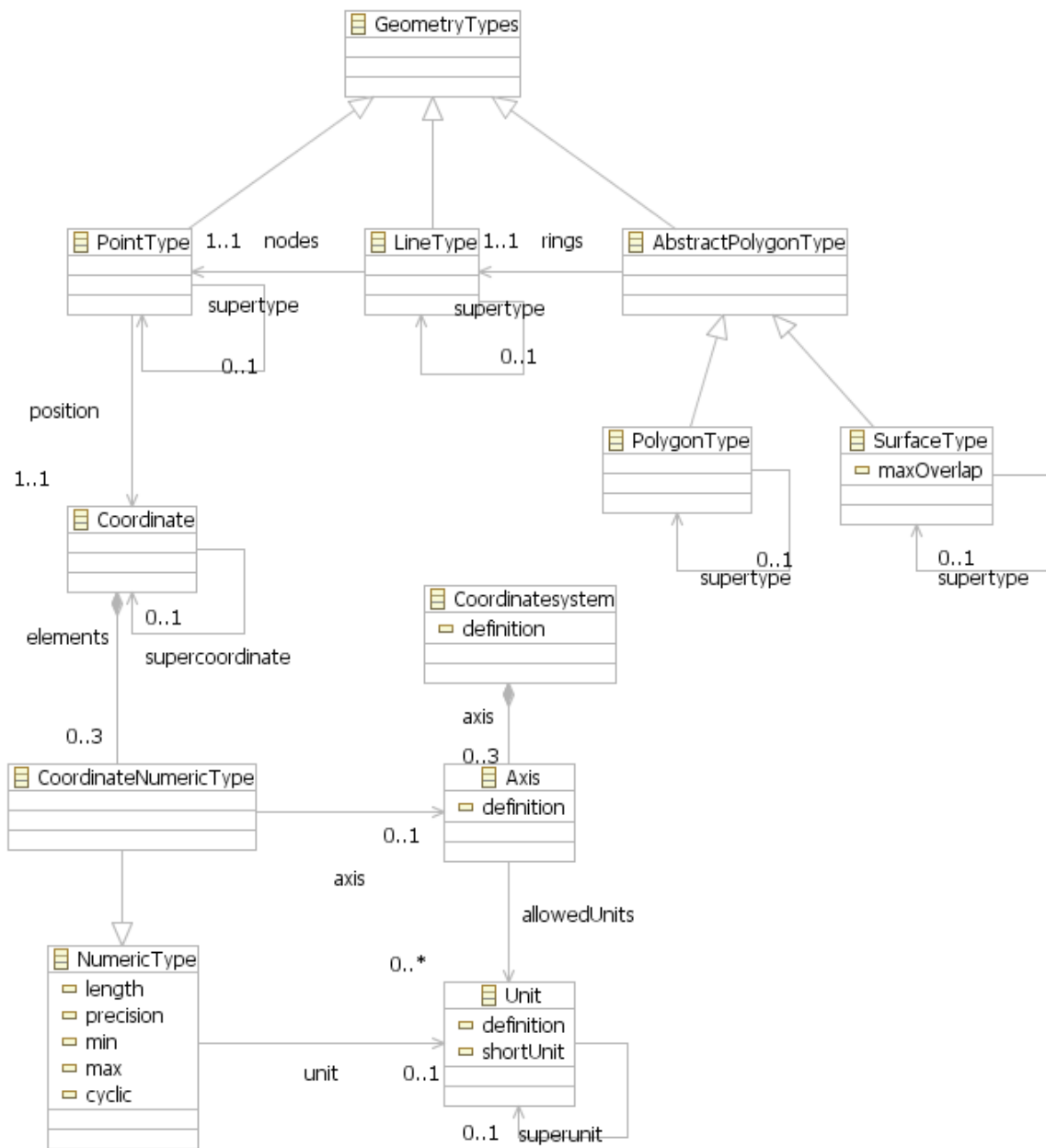


Figure 9: Geometry types and corresponding elements (TODO: Add Multigeometry Types).

## 5 Literature