Title:

A5.2-D3 [3.3] Conceptual Schema Specification and Mapping

Author(s)/Organisation(s):

Thorsten Reitz (FhG-IGD), Marian de Vries (TUD), Daniel Fitzner (FhG-IGD)

Working Group:

Architecture Team / WP5

References:

A5.2-D3 [3.0] A Lightweight Introduction to the HUMBOLDT Framework V3
A5.2-D3 [3.1] Specification Introduction and Overview V3
A5.2-D3 [3.3.1] The HUMBOLDT Alignment Editor
A5.2-D3 [3.4] Context Service Specification
A5.2-D3 [3.6] Processing Components General Model and Implementations
A5.3-D3 Humboldt Commons Specification / Framework Common Data Model V3

Quality Assurance:

- ☑ Review WP Leader: Thorsten Reitz (FhG-IGD)
- ☐ Review dependent WP leaders:
- ☐ Review Executive Board:
- ☑ Review others:

Delivery Date: **02.12.2009**

Short Description:

This document provides the common framework for the HUMBOLDT software dealing with conceptual schema harmonisation. It describes general issues encountered, ways to resolve them and details the mapping language that we're using.

Keywords:

Harmonisation, geoprocessing, web services, schema mapping, conceptual schema translation, ontology

History:

Version	*Author(s)*	*Status*	*Comment*
001	Thorsten Reitz, Marian de Vries, Astrid Fichtinger	RFC	Initial version based on WIKI version, updated [3.6] document and the gOML description
002	Daniel Fitzner	FINAL	Review

Table of contents

List of Figures

List of Tables

1 Introduction

This document provides the common framework for all HUMBOLDT software dealing with the resolution of heterogeneities on the conceptual schema level. This includes the definition of target models, the mapping of source to target schemas using specialized software and the execution of schema translation on actual geodata.

1.1 Purpose of this document

This document describes processing components, an essential group of software components within the HUMBOLDT Data Harmonisation Framework. It also establishes common architectural elements for processing services to be used in the HUMBOLDT Framework, especially for harmonisation processing services, to make them useable by other components such as the Mediator Service (MS) and the Workflow Design and Construction Service (WDCS).The terms processing component and Transformer are used synonymously in the following.

Specifically, this document contains information relevant for the following applications and services:

- **Conceptual Schema Transformer (CST):** The CST is a service that is able to apply a schema transformation to a geodata set (expressed in a specific source Application Schema (A)) in order to provide a geodata set (expressed in a specific target Application Schema B). A schema mapping between schema A and schema B has to be defined beforehand in order to accomplish the transformation. The CST can only work when a target conceptual schema has been defined and when a mapping of the source to the target conceptual schema has been created.
- **HUMBOLDT Model Editor:** This is a restricted graphical UML editor that enables users to define the source and target data models (application schemas). The schemas can be:
    - created from scratch,
    - or imported,
    - or generated (reverse engineered) in case of the source application schemas.
- **HUMBOLDT Alignment Editor:** The HUMBOLDT Alignment Editor (HALE) offers users an interactive graphical user interface to specify the mapping rules between source and target application schemas. Because this has to be done by domain experts, not by programmers, the following criteria are important:
    - the way to specify the mappings between source and target schemas must be easy (good GUI);
    - all kinds of mappings must be supported (see point 3 below), also those involving geometric processing (thru functions/procedures);
    - preferably direct feedback is given to the user whether the mapping is well specified (no mistakes are made), and whether it is adequate and consistent (TODO: define measures for this);
    - the mappings must be encoded in such a way that the actual transformation code can be derived from it.

- **HUMBOLDT Model Repository:** A distributed infrastructure that stores and provides conceptual schemas with all their dependencies as well as mapping defined to translate data from one of these schemas to another.

1.2 Abbreviations used in this document

This section summarizes the abbreviations used specifically for this service component. It does not repeat information found in the introduction and specification overview document [3.0].

1.3 Definitions valid in this document

Any definitions that are given as part of the Specification Introduction and Overview document [3.0] are also valid in this document. In addition, the following definitions are relevant for this document:

1.3.1 Model, schema

Model (in general): an abstraction of a collection of things or of one thing, not the real-world (real or virtual) objects themselves.

- *Data model:* a model of the (geographic) data that is stored and/or exchanged.
- *Schema:* a schema is a 'kind-of' model. Some use the words as synonyms, for others there is a distinction.

In the ISO 19100 series, the two terms are distinguished as follows [ISO 19101]:

- *conceptual model*: "model that defines concepts of a universe of discourse"
- *conceptual schema:* "formal description of a conceptual model". A conceptual schema is described in a conceptual schema language (in case of the ISO 19100 series in the respective UML profile described in ISO 19103).

The ISO definition of conceptual schema is very broad. Under this definition also an ontology is a conceptual schema.

Looking not at ISO, but at data modelling and data management practice: the word 'schema' is chosen often in the context of data management and model implementation (by database administrators for example), the word 'model' is used more by system designers and data modellers in the context of design and development. This different roles and subtle distinctions between 'schema' and 'model' can also be seen in Table 1 below, where the following pattern becomes apparent: a schema is lower at the abstraction scale than the model with the same name: what is called a 'conceptual schema' in data management, is called a 'logical data model' by data modellers.

1.3.1.1 Levels and scope of models: a matrix

For the discussion in WP5 especially the following terms needed clarification: What is a 'conceptual schema' in HUMBOLDT? What is the relation with the term 'application schema' in HUMBOLDT?

When we compare the two terms we actually compare two dimensions of a data model: abstraction level (or: platform independent vs. platform dependent), and scope (generic vs. application-specific). The matrix in Table 1 is meant to clarify this also with examples.

With scope of a data model is meant: from generic to application-specific data models (from left to right in the table).

With abstraction level is meant: from conceptual view on the information content to implementation-specific scripts and schemas (from top to bottom in the table). The abstraction level of data models has to do with:

- the data model being implementation-neutral (in software engineering aka Platform Independent Model) or implementation-specific (aka Platform Specific Model). In database modeling and implementation the 3 levels 'conceptual, logical and physical' is more common.
- phases in data modeling: from first sketch of the information content and how it can be structured, to a final model where all question marks are decided on.

In the cells there are examples of data modeling artifacts that belong on that spot in the matrix.

1.3.1.2 Note about the relation with ISO definitions

The term "application schema" is used widely in the ISO/OGC and INSPIRE world. For that reason we also use it in HUMBOLDT. The term will be used as synonym for *HUMBOLDT's 'application-specific data model'* (see deliverable 7.1D1*). Application schema* or *application-specific data model* is then: the data model for a specific application, such as a HUMBOLDT Scenario.

The term *'conceptual schema'* is used in ISO/OGC standards, and also in GIS practice and discussions, to express the contrast with the physical data format (the actual encoding/storage format of the data). A conceptual schema gives (logical/conceptual) information about the content and structure of the data, not how it is actually (physically) stored.

According to ISO, an application schema is a "conceptual schema for data required by one or more applications" [ISO 19101]. The last part about 'required by one or more applications' accords to the HUMBOLDT definition. But the 'conceptual schema' part of the definition can lead to misunderstanding. In practice in ISO/OGC and INSPIRE the term 'application schema' is used at two levels: at the conceptual schema level (in INSPIRE as UML model), and also at the logical/physical schema level, as for example in 'GML application schema'.

	Abstraction level	*Explanation*	*Scope: Generic -> application-specific*		
			information field	*application domain*	*application*
PIM	Ontology (of concepts)	Can take different forms: from thesaurus in plain English (loose structure), to ontology in ontology language (processable by software)	The INSPIRE Feature Concept Dictionary can evolve into a cross-application ontology for the geographic information field.	For example the thesaurus about Marine terms and knowledge maintained by NERC/BODC.	Codelists and classifications can be published on a Scenario portal in the form of an ontology.

PIM	Conceptual data model = The 'what', and first 'how': what is the relevant information for us (in our information field, application domain or application), and how can it be divided into classes, attributes, and relations between classes.	How humans see the information that is involved in their use cases. Does not have to be specified in detail yet (this is the second aspect: phases in data modeling).	ISO 19109 (General Feature Model), ISO 19123 (Coverages): text and UML models.	Classification schemes (or simpler: standardized enumeration lists) for specific application domains: NUTS, CORINE, Habitat classifications, Risk levels, …	HUMBOLDT Scenario-specific conceptual data model, details can be left out, implementation platform does not have to be taken into account. Purpose: this model 'freezes' what information will be used as input and/or published as output in the Scenario.
PIM/ PSM	Logical data model, or: conceptual schema = Describes and prescribes the logical data structure and constraints on attribute values using a conceptual schema language	Precise enough to be 'unambiguously understood' by software ('deterministic') Takes implementation constraints into account, for example yes/no multiple inheritance, composite attributes or repeating groups.	UML models of ISO 19107 (spatial schema), ISO 19108 (temporal schema).	The INSPIRE Annex Theme data models, in two representations: UML data model, called 'UML application schema' Feature Catalogue	HUMBOLDT Scenario-specific application schema in UML (to discuss: also as ISO 19110 Feature Catalogue ?). This must be precise enough (for example with unambiguous data types) so that software can process it.
PSM	Physical data model A. Logical schema: description of data structure and constraints in a Data Definition Language (DDL)	Can be derived by software from the logical data model. Needed for that: rules/mappings for vertical schema translation from	Examples: XML Schema files (*.xsd) that express the GML specification. 'create spatial data types and metadata	In case of GML encoding: the 'GML application schema' (official OGC term), often abbreviated to 'GML schema'. Examples: SensorML.xsd	Same terms as for application-domain. Examples: our Scenario xsd's in case of GML, inline headers in

	such as XML Schema, or SQL 'Create table etc' constructs. Which DDL to use depends on the data storage format that is chosen (XML, ascii, database category)	the higher level to this level.	tables' scripts for PostGIS.	CityGML.xsd NetCDF.xsd	case of NetCDF files, file-specific metadata in case of image/raster files, sql create table scripts in case of Oracle or PostGIS.
PSM	B. Physical schema = The 'how' for a concrete implementation.	The above (physical data model), plus indexes, table space allocation, MIME-type settings, etc. This depends on the chosen encoding format, and on 'local' conditions.			

Table 1: Matrix of data modeling artifacts relevant to HUMBOLDT

1.4 Standards used in this document

The Mediator Service Component makes use of some of the core standards in geoinformation for the provision of maps and other products, as well as raw data. Most notably, these are:

1.4.1 OGC Geographic Markup Language (GML)

GML 3.1 and 3.2.1 are used as primary geodata exchange model and encoding for the processing service components.

1.4.2 The Unified Modelling Language (UML) 2.0

The UML provides a very detailed conceptual schema language to express models with their class, attributes, operations and relations, including constraints.

1.4.3 The Ontology Mapping Language (OML)

This mapping language (alternative name 'alignment language') is the result of a number of European projects: DIP, SEKT and - most recent - Knowledge Web. It is largely independent of the Conceptual Schema Language used and is very expressive. It is being standardized in the context of the OMWG:

OMWG D7.2: Ontology Mapping Language RDF/XML Syntax (DERI OMWG Working Draft last update 21 January 2007), http://www.omwg.org/TR/d7/rdf-xml-syntax/

1.4.4 OGC Web Processing Service (WPS)

The WPS is used as a basic interface for the Transformer specification, which is used as the interface from the Mediator Service to processing capabilities.

2 Enterprise Viewpoint

The Enterprise viewpoint describes the functional purpose of the component and its integration with business processes. This chapter first of all provides directly relevant requirements and then makes use of BPMN to outline the overall process in which these components are used.

2.1 Requirements

Many of the requirements that were collected from the HUMBOLDT scenarios directly indicate the need to perform conceptual schema translation. These requirements were consolidated and assigned the relevant components. For the Conceptual Schema Specification and Mapping set of components, the following common requirements were deemed relevant:

2.1.1 Metadata Requirements Cluster

- *METADATA01*: The SYSTEM shall be able to transform Metadata from existing metadata schemas employed in the scenarios to the HUMBOLDT, ISO, and INSPIRE schemas (OS_001, BSS_080).

2.1.2 Conceptual Schema Translation Requirements Cluster

- *CST01*: The SYSTEM shall be able to perform transformations of data structure to transform geographic information to a harmonised schema as defined in a scenario (PAS_001, ES_004, OS_004, HAR_013).
- *CST02*: The SYSTEM shall be able to reclassify attributes according to different classifications (ES_015)
- *CST03*: The SYSTEM shall guide the USER and ask him/her for checking/validating the trans-coding case by case (PAS_002).
- *CST04*: The SYSTEM shall be able to select attributes of individual features or of sets of features out of a data set by applying Filters.
- *CST05*: The SYSTEM shall be able to perform renaming of types and of attributes (see 2. in CST definition).
- *CST06*: The SYSTEM shall be able to perform reclassification (see 3. in CST definition).
- *CST07*: The SYSTEM shall be able to perform merging and splitting of Features (see 4. in CST definition).
- *CST08*: The SYSTEM shall be able to perform type conversions, specifically spatial conversions, UoM conversions and alphanumeric conversions (see 6. in CST definition).
- *CST09*: The SYSTEM shall be able to assign derived spatial properties such as centroids or bounding boxes.
- *CST10*: The SYSTEM shall be able to assign default and null values where required.

2.1.3 General Requirements

- *PCG01:* When any transformation, the SYSTEM shall add information on all modifications performed to the data set's metadata. Adding metadata to individual features is not required (ES_005, PA and others).

2.2 Business Processes

Processing components support a wide range of very different business processes, different within each scenario. Nonetheless, at least for the harmonisation processing components which are the focus point of this document, there are common business processes or at least, activities.

2.2.1 Business process overview

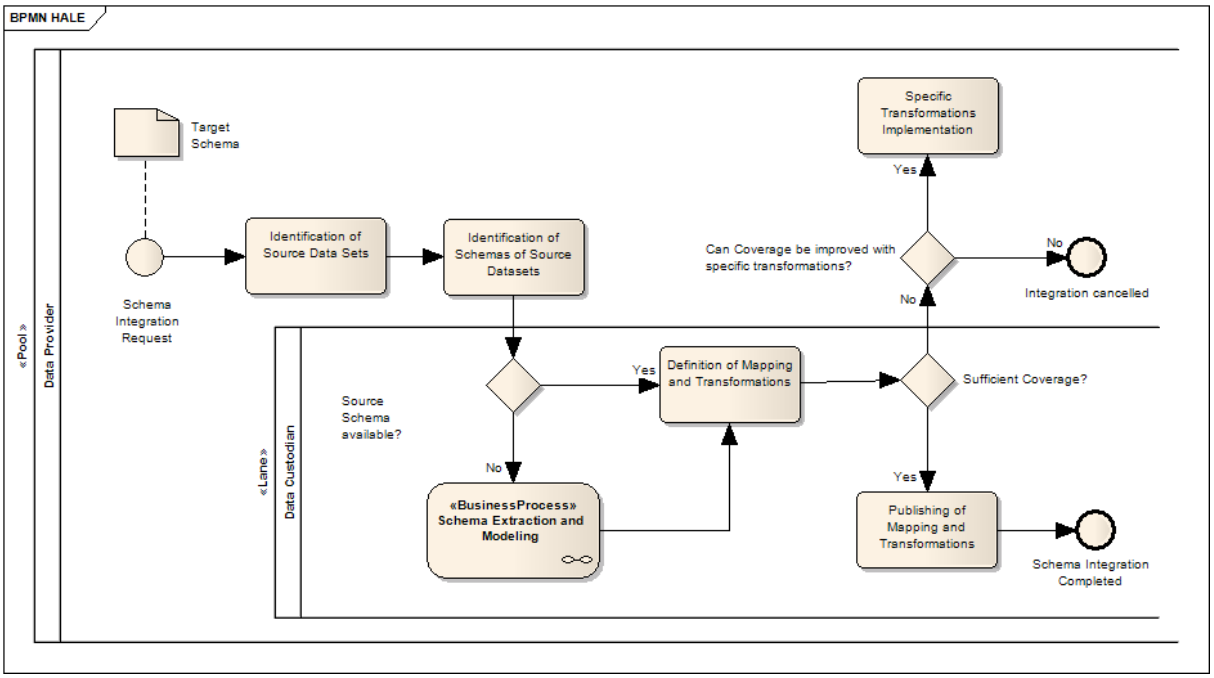

Figure 1: The overall business process associated with the schema specification mapping activities

The general business process of schema integration usually starts with a data integration request. This request can be one-time or recurring and is accompanied with information about the target schema into which available data needs to be integrated. Examples for such a start event include the necessity to make data available via an INSPIRE download service, but can also includes cases where the target schema is specific to a Data Provider's or Data Users' infrastructure and applications, as it is the case in the HUMBOLDT scenarios.

After the identification of relevant data sets that have to be transformed and the identification of their schemas, the Model Editor can be used to create or enrich the source schema, e.g. starting from a simple shapefile's attributive data, or from a relational database or an XML schema. When both source and target schemas contain the required amount f information, the conceptual schema mapping process can be started.

This definition of mappings and transformations can be done using the HUMBOLDT Alignment Editor. The result of this work is a mapping file, which can later be used by the Conceptual Schema Transformer to actually transform a set of geographical objects.

For more information on the organisational and technical process of creating the target schema, please refer to Deliverables A7.0-D2 and A7.1-D2.

3 Computational Viewpoint

The computational viewpoint is omitted from this document; please refer to the individual component specification documents.

4 Information Viewpoint

Please note that the UML class diagrams in this chapter are based on the Java API for the gOML, while the code listings are based on the XML/RDF serialization of the OML.

4.1 The Geographic Profile of the Ontology Mapping Language (gOML)

The specification of an adequate language for conceptual schema mapping was one of the highest priorities for version 2 and version 3 specification works. Towards this, detailed requirements analyses were conducted by trying to recreate matching tables created in INSPIRE transformation testing using different languages. One of the first such experiments for the conceptual schema transformation experiments was with HS-ERiskA data, where the mapping rules were directly written in XSLT.

Better would be to have an implementation-neutral mapping language, from which the actual transformation code can be derived, depending on the requirements in that particular case. This fits better with the model driven approach that was discussed at HUMBOLDT meetings (e.g. Muenchen 2007, Zuerich 2008).

n addition, XSLT cannot be used in case of non-XML input. Relying on XSLT as model2model mapping language would not work for data harmonisation in especially the Ocean and Galileo scenarios.

The criteria for this model2model mapping language are in short (also see the description in the Conceptual Schema Transformer, and the concept text of 7.0-D2):

- Expressive enough: it must support renaming of classes and attributes, restructuring, reclassification, and also a number of general (non-geo), geometric and topological functions to transform geographic data;
- The actual mapping code for different platforms/implementations can be derived from it, for example XSLT or XQuery for XML/GML, or Java code, to do the actual data transformation;
- Preferably it builds on existing standards or initiatives.

After some preliminary literature research and testing of tools one candidate for this mapping language is: the ontology mapping language (OML) proposed by Scharffe, Euzenat et al. This mapping language (alternative name 'alignment language') is the result of a number of European projects: DIP, SEKT and - most recent - Knowledge Web.

See these references for a description and examples:

- Euzenat, J., F. Scharffe, et al. (2007). D2.2.10: Expressive alignment language and implementation. Project deliverable 2.2.10, Knowledge Web NoE (FP6-507482), 2007.
- Scharffe, F. (2008). Correspondence Patterns Representation. Faculty of Mathematics, Computer Science and Physics, University of Innsbruck. PhD.
- OMWG D7.2: Ontology Mapping Language RDF/XML Syntax (DERI OMWG Working Draft last update 21 January 2007), http://www.omwg.org/TR/d7/rdf-xml-syntax/

The ontology mapping language (OML) is not the only approach, but seems the most expressive at the moment, and it's suitability was confirmed by the implementation of the CST and HALE. The class diagram below gives a first impression of the structure of OML.

4.1.1 The *align* Package

The align package of the OML represents the main mapping constructs: The *Alignment* as the complete set of mappings defined between two conceptual schemas, a definition of the *Schemas* itself including a definition of the used *Formalism*, the *Cell* as the unit of mapping and the *Entity* as the abstract element being mapped.

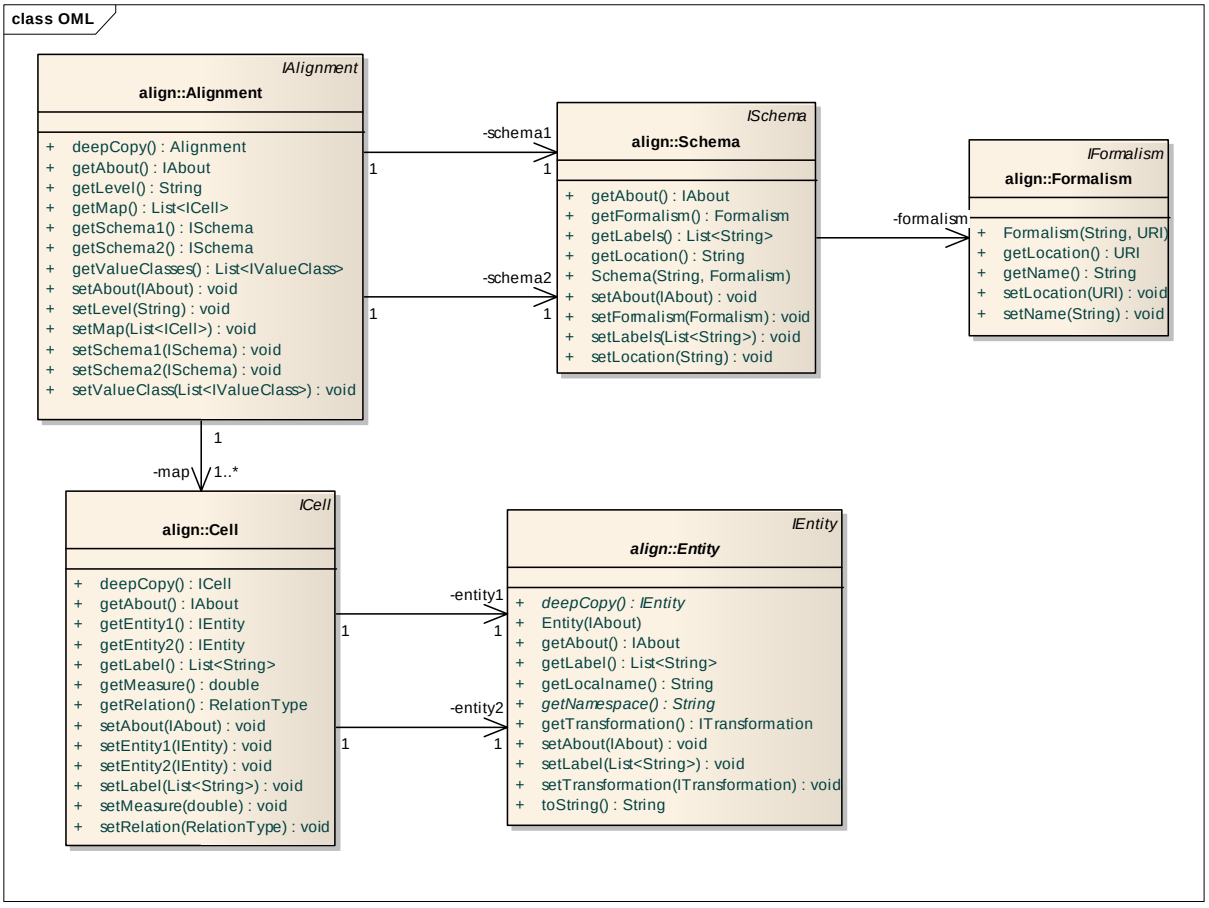

Figure 2: The classes of the align package of the gOML

4.1.1.1 Alignment, Schema and Formalism

The Alignment represents all mappings defined in between two conceptual schemas. It comprises references to information about those two schemas, plus some metadata on the alignment itself. This metadata includes an About object that can be used to identify the alignment, a list of mappings, a list of ValueClasses used to encode enumerations in OML, and a value called Level which can be used to indicate the level of concreteness that this alignment represents. The following listing provides an example for such an Alignment element (with some omissions detailed later on).

```
<Alignment xmlns:omwg="http://www.omwg.org/TR/d7/ontology/alignment"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:goml="http://www.esdi-humboldt.eu/schemas/goml"
```

```
xmlns:align="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml/">
  <align:level></align:level>

  <align:onto1>
    <align:Ontology>
      <align:location>
        http://www.esdi-humboldt.org/waterVA
      </align:location>
      <align:formalism>...</align:formalism>
    </align:Ontology>
  </align:onto1>
  <align:onto2>
    <align:Ontology>
      <align:location>
        urn:x-inspire:specification:gmlas-v31:Hydrography:2.0
      </align:location>
      <align:formalism>...</align:formalism>
    </align:Ontology>
  </align:onto2>
  <align:map>...</align:map>
</Alignment>
```

For GML Application schemas, all HUMBOLDT schema mapping/translation applications expect the following formalism element.

```
<align:Formalism>
  <align:uri>http://www.opengis.net/gml/3.2.1/</align:uri>
  <align:name>GML 3.2.1 Application Schema</align:name>
</align:Formalism>
```

Alternatively, GML 3.1 or GML 2.1.2 declarations are also allowable.

4.1.1.2 Cell

A Cell contains a mapping between two Entities, such as FeatureClasses or Property objects. It represents the basic unit of conceptual schema mapping. The following listing provides an example for a simple equivalence mapping of two FeatureClasses.

```
<align:Cell>
  <omwg:entity1>
    <omwg:Class rdf:about="
      http://www.esdi-humboldt.org/waterVA/Watercourses_VA">
    <omwg:transf rdf:resource="…RenameFeatureFunction "/>
    </omwg:Class>
  </omwg:entity1>
  <omwg:entity2>
    <omwg:Class rdf:about="
      urn:x-inspire:specification:gmlas-v31:Hydrography:2.0/Watercourse">
    <omwg:transf/>
    </omwg:Class>
  </omwg:entity2>
  <align:relation>Equivalence</align:relation>
</align:Cell>
```

It should be noted that the *Relation* element is optional in the case of an equivalence relation, but should still be given always. The OML also contains an optional measure element which is not used in HUMBOLDT. This element can be used to save a confidence that a human or an algorithm would assign any given mapping.

The next listing provides an example of a simple *Property* mapping, where no relation element is needed. When a *Property* is mapped, a *transf* element always has to be present to indicate any required transformations of the type. In this case, a simple *NAME* attribute is used to set up a *geographicalName* attribute as required in the INSPIRE schema.

```
<align:Cell>
  <omwg:entity1>
    <omwg:Property rdf:about="
         http://www.esdi-humboldt.org/waterVA/Watercourses_VA/NAME">
      <omwg:transf rdf:resource="...CreateInspireGeoName">
      </omwg:transf>
    </omwg:Property>
  </omwg:entity1>
  <omwg:entity2>
    <omwg:Property rdf:about="urn:x-inspire:specification:
           gmlas-v31:Hydrography:2.0/SurfaceWater/geographicalName">
      <omwg:transf/>
    </omwg:Property>
  </omwg:entity2>
</align:Cell>
```

Cells can also be conditional. The following example provides a Cell that defines a mapping between two FeatureClasses that is only applicable when a certain Restriction on an attribute/property is met. This is also one of the places where we have defined an extension to the OML, namely the inclusion of filters based on OGC Common Query Language (CQL) expressions. These simplify the structure that the OML foresees for defining filters:

```
<align:Cell>
  <omwg:entity1>
    <omwg:Class rdf:about="
      http://www.esdi-humboldt.org/waterVA/Watercourses_VA">
      <omwg:transf rdf:resource="...RenameFeatureFunction"/>
      <omwg:attributeValueCondition>
        <omwg:Restriction>
          <goml:cqlStr>LEVEL > 200</goml:cqlStr>
        </omwg:Restriction>
      </omwg:attributeValueCondition>
    </omwg:Class>
  </omwg:entity1>
  <omwg:entity2>
    <omwg:Class rdf:about="urn:x-inspire:
      specification:gmlas-v31:Hydrography:2.0/Watercourse">
      <omwg:transf/>
    </omwg:Class>
  </omwg:entity2>
</align:Cell>
```

Furthermore, Cells can also define instance split and merge conditions. An instance split is a case where from one entity represented in the source schema, multiple entities in the target schema are created. For instance merges, the opposite is true: multiple source features are used to create a single target feature. The next listings provides an example for a split case:

```
<align:Cell>
  <omwg:entity1>
    <omwg:Class rdf:about="
      http://www.esdi-humboldt.org/waterVA/Watercourses_VA">
      <omwg:transf rdf:resource="…RenameFeatureFunction ">
        <param>
          <name>InstanceSplitCondition</name>
          <value>foreach LineString in waterVA:the_geom</value>
```

```
        </param>
      </omwg:transf>
    </omwg:Class>
  </omwg:entity1>
  …
</align:Cell>
```

Example merge case:

```
<align:Cell>
  <omwg:entity1>
    <omwg:Class rdf:about="
      http://www.esdi-humboldt.org/waterVA/Watercourses_VA">
      <omwg:transf rdf:resource="…RenameFeatureFunction ">
        <param>
          <name>InstanceMergeCondition</name>
          <value>groupBy waterVA:LEVEL</value>
        </param>
      </omwg:transf>
    </omwg:Class>
  </omwg:entity1>
  …
</align:Cell>
```

Finally, there can also be so-called augmentation Cells, which define transformations on Entity2 instead of Entity1. These have the special characteristic that they don't need any of the information available in the source features and they are executed after all other mappings have been applied. The next listing provides an example for such an augmentation function.

```
<align:Cell>
  <omwg:entity1>
    <Class rdf:about="null"></Class>
  </omwg:entity1>
  <omwg:entity2>
    <omwg:Class rdf:about="
      http://www.esdi-humboldt.org/waterVA/Watercourses_VA">
      <omwg:transf rdf:resource="…NilReasonFunction">
        <param>
          <omwg:name>NilReasonType</omwg:name>
          <omwg:value>unpopulated</omwg:value>
        </param>
      </omwg:transf>
    </omwg:Class>
  </omwg:entity2>
  <align:relation>Equivalence</align:relation>
</align:Cell>
```

As can be seen from the example, augmentation cells also have the special characteristic that they use a "null" entity for entity1.

4.1.1.3 Entity

Entity is the abstract superclass for all elements of a schema that can be mapped. These elements are explained in detail in the next section, but there are some characteristics all share:

- Entities have an *About* attribute that identifies them, using the namespace and the local name. In case of a Property, the *FeatureClass* local name that aggregates the property is also part of the *About*. Examples for all these cases are provided with the listings above.

- Entities can have a transformation function assigned. Normal transformations are defined on Entity1, whereas Augmentation transformations (functions that don't need any values from the source objects) are defined on Entity2.

4.1.2 The omwg Package

The omwg package contains a set of classes that allows expressing different types of entities to be mapped, ranging from classes (called *FeatureClasses* here to avoid naming conflicts) over *Properties* to *ComposedFeatureClasses* and *ComposedProperties*. Also, *Restrictions* (filters on a mapping) and *Relations* (mappings that form entities themselves) are modelled here.

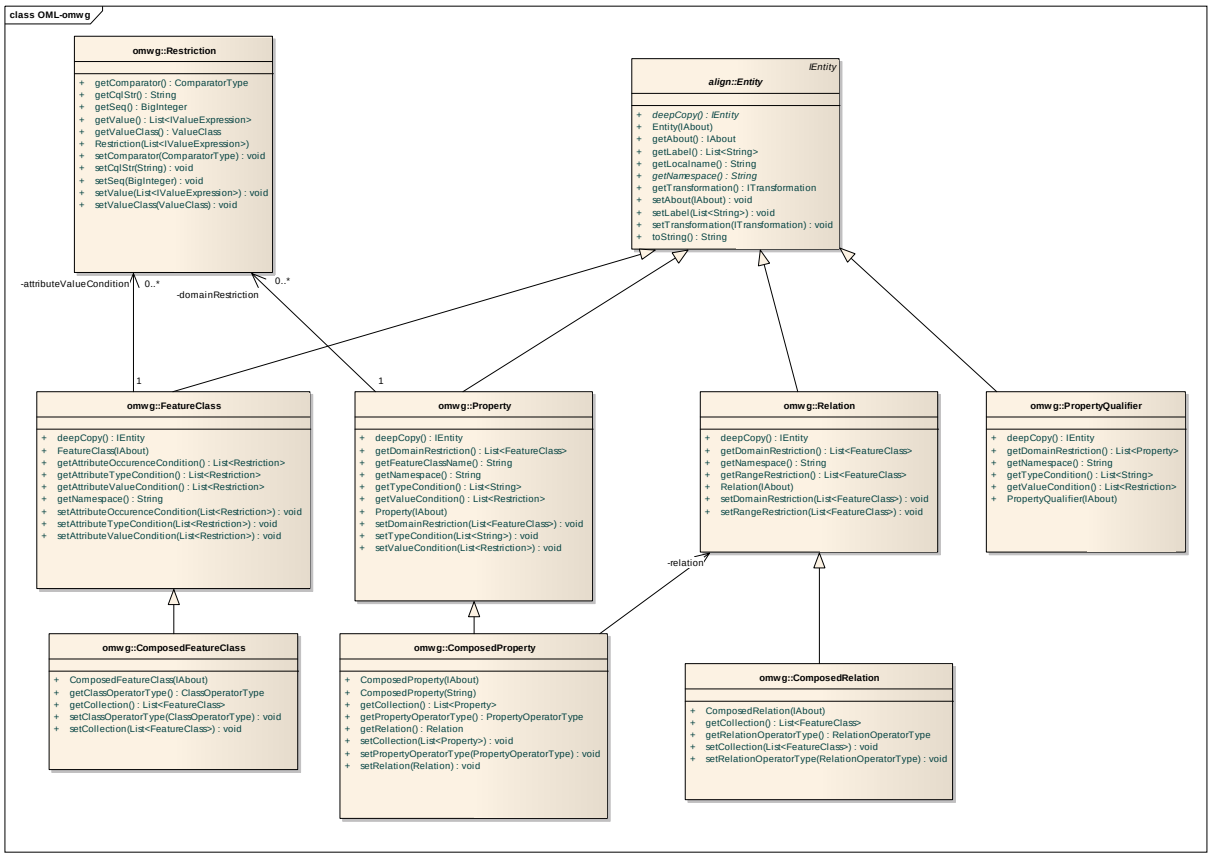

Figure 3: The classes of the omwg package of the gOML

4.1.2.1 FeatureClass

A *FeatureClass* is representative of a class-level element of a schema. In INSPIRE terms, it is equal to a Spatial Object Type; in OGC terms it can be considered equal to a *FeatureType*. Since the OML stems from semantic web research, the *FeatureClass* can also represent a concept from an ontology.

4.1.2.2 ComposedFeatureClass

A *ComposedFeatureClass* can be used to make statement like that the union of two types in a schema is equal to one type in another schema. The following listing provides an example for a ComposedFeatureClass.

```
<align:Cell>
  <align:Entity1>
```

```
    <omwg:ComposedFeatureClass rdf:About="…">
      <omwg:ClassOperatorType>AND</omwg:ClassOperatorType>
      <omwg:Collection>
        <omwg:FeatureClass> ... </omwg:FeatureClass>
        <omwg:FeatureClass> ... </omwg:FeatureClass>
        <omwg:FeatureClass> ... </omwg:FeatureClass>
      </omwg:Collection>
    </omwg:ComposedFeatureClass>
  </align:Entity1>
...
</align:Cell>
```

As with ComposedProperties (see below), *ComposedFeatureClasses* have artificial local name keys, in this case, a random UUID.

4.1.2.3 Property

A Property is representative of a single attribute of a FeatureClass; an attribute can be of primitive type (ie. contain a literal value, such as a String or a number), or it can be a complex attribute.

4.1.2.4 ComposedProperty

ComposedProperties are required when a value in a target schema property cannot be derived from a single source value. As an example, consider the mathematical expression that uses the average of three input values to determine the output value in this listing:

```
<omwg:entity1>
  <omwg:Property rdf:about="http://www.esdi-humboldt.org/waterVA/
    d374454b-ff2c-4a79-8efe-e1077d48a82f">
    <omwg:transf rdf:resource=" …GenericMathFunction">
      <omwg:param>
        <omwg:name>math_expression</omwg:name>
        <omwg:value>(LAENGE_ARC + LAENGE_ROU + LENGTH) / 3</omwg:value>
      </omwg:param>
    </omwg:transf>
    <omwg:propertyComposition>
    <omwg:collection>
      <omwg:item>
        <omwg:Property rdf:about="http://www.esdi-humboldt.org/waterVA/
          Watercourses_VA/LAENGE_ARC">
          <omwg:transf/>
        </omwg:Property>
      </omwg:item>
      <omwg:item>
        <omwg:Property rdf:about="http://www.esdi-humboldt.org/waterVA/
          Watercourses_VA/LAENGE_ROU">
          <omwg:transf/>
        </omwg:Property>
      </omwg:item>
      <omwg:item>
        <omwg:Property rdf:about="http://www.esdi-humboldt.org/waterVA/
          Watercourses_VA/LENGTH">
          <omwg:transf/>
        </omwg:Property>
      </omwg:item>
    </omwg:collection>
    <omwg:Relation/>
    </omwg:propertyComposition>
  </omwg:Property>
</omwg:entity1>
```

ComposedProperties have artificial keys, in this case, a random UUID.

4.1.2.5 Restriction

Restrictions are used by all other Entity types for multiple purposes, such as filtering the validity of a mapping.

4.1.3 The rdf and ext Packages

The rdf elements are two basic elements taken from the Resource Description Framework. The ext package can be used for the definition of new elements that should extend the functionality of the OML.

4.1.3.1 rdf:About and rdf:Resource

These two elements are used for identification of all kinds of elements of a OML document. In general, About is used, but in the transf element, the grounding of the transformation (i.e. the address of the concrete service implementing it) is represented by a Resource attribute.

Valid Resource values may be either an URL of a Web Processing Service with the ProcessIdentifier that is relevant, or it may be a local resource name, like a fully qualified java class name.

4.1.3.2 ext:transf and ext:transfPipe

These elements allow to provide a schema translator with more concrete instructions about how to execute mappings defined in cells, and are mainly important for property mappings between complex properties, such as geometries. *Transf* provides a single transformation definition, whereas *transfPipe* can be used to define a chain of transformations.

5 Interface Control Document for Schema Mapping

5.1.1 Geography Profile of the Ontology Mapping Language

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
      xmlns:align="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
      xmlns:omwg="http://www.omwg.org/TR/d7/ontology/alignment"
      xmlns:goml="http://www.esdi-humboldt.eu/goml"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      elementFormDefault="qualified">

   <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="tmprdf.xsd" />
   <xs:import namespace="http://www.omwg.org/TR/d7/ontology/alignment"
schemaLocation="omwg.xsd" />
   <xs:import namespace="http://humboldt/goml" schemaLocation="goml.xsd" />

   <xs:element name="Alignment" type="align:AlignmentType" />
   <xs:complexType name="AlignmentType">
      <xs:sequence>
         <xs:element name="level" type="xs:string" />
         <xs:element name="onto1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="align:Ontology"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="onto2">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="align:Ontology"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="map" minOccurs="0" maxOccurs="unbounded" >
            <xs:complexType >
               <xs:sequence>
                  <xs:element ref="align:Cell" maxOccurs="1"/>
               </xs:sequence>
               <xs:attribute ref="rdf:about" use="optional"/>
            </xs:complexType>
         </xs:element>
         <xs:element ref="goml:ValueClass" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute ref="rdf:about" use="optional"/>
   </xs:complexType>

   <xs:element name="Ontology" type="align:OntologyType" />
   <xs:complexType name="OntologyType">
      <xs:sequence>
         <xs:element ref="omwg:label" minOccurs="0" maxOccurs="unbounded" />
         <xs:element name="location" type="xs:string" />
         <xs:element name="formalism" >
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="align:Formalism"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
```

```
      </xs:sequence>
      <xs:attribute ref="rdf:about" use="optional"/>
   </xs:complexType>


   <xs:element name="Formalism" type="align:FormalismType" />
   <xs:complexType name="FormalismType">
      <xs:sequence>
         <xs:element name="uri" type="xs:string" />
         <xs:element name="name" type="xs:string" />
      </xs:sequence>
   </xs:complexType>

   <xs:element name="Cell" type="align:CellType" />
   <xs:complexType name="CellType">
      <xs:sequence>
        <xs:element ref="omwg:label" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="omwg:entity1"/>
        <xs:element ref="omwg:entity2"/>
        <xs:element ref="align:measure" minOccurs="0" />
        <xs:element name="relation" type="align:relationEnumType" minOccurs="0"  />
      </xs:sequence>
      <xs:attribute ref="rdf:about" use="optional"/>
   </xs:complexType>

   <xs:simpleType name="relationEnumType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Equivalence"/>
         <xs:enumeration value="Subsumes"/>
         <xs:enumeration value="SubsumedBy"/>
         <xs:enumeration value="InstanceOf"/>
         <xs:enumeration value="HasInstance"/>
         <xs:enumeration value="Disjoint"/>
         <xs:enumeration value="PartOf"/>
         <xs:enumeration value="Extra"/>
         <xs:enumeration value="Missing"/>
      </xs:restriction>
   </xs:simpleType>

   <xs:element name="measure" type="xs:float" />
   <!--xs:element name="measure">
     <xs:complexType>
       <xs:simpleContent>
         <xs:extension base="xs:float">
           <xs:attribute ref="rdf:datatype" use="optional"/>
         </xs:extension>
       </xs:simpleContent>
     </xs:complexType>
   </xs:element-->

</xs:schema>
```