



Experimental ontology modules formalising concept definition of ATM data

Deliverable 1.1

BEST

Grant:

699298

Call:

H2020-SESAR-2015-1

Topic:

Sesar-03-2015

Information Management in ATM

Consortium coordinator:

SINTEF

Dissemination Level:

PU

Edition date:

31 May 2017

Edition:

01.03.00 Final Version

Founding Members



Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Audun Vennesland (SINTEF)	Project Member	31.05.2017
Bernd Neumayr (LINZ)	Project Member	30.05.2017
Christoph Schuetz (LINZ)	Project Member	30.05.2017
Alex Savulov (FRQ)	Project Member	22.07.2016

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Scott Wilson (ECTRL)	Project Member	23.05.2017
Eduard Gringinger (FRQ)	Project Member	17.05.2017
Bernd Neumayr (LINZ)	Project Member	19.05.2017

Approved for submission to the SJU By — Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
Approved by consortium in accordance with procedures defined in Project Handbook.	Consortium	31.05.2017

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
------------------	----------------	------

Document History

Edition	Date	Status	Author	Justification
00.00.01	10.07.2016	Document created	Audun Vennesland	First draft
00.00.02	09.09.2016	Planned Content and Structure proposed	Audun Vennesland	Prepared document for internal review (Planned Content and Structure)
00.00.03	26.09.2016	Planned Content and Structure approved	Audun Vennesland	Addressing comments from internal review (Planned Content and Structure)

00.01.01	24.02.2017	Intermediate proposed	Audun Vennesland	Prepared document for internal review (intermediate proposed)
01.00.00	16.05.2017	Final draft	Audun Vennesland	Addressing comments from internal review (intermediate proposed) and circulated to FRQ and LINZ for internal review
01.01.00	21.05.2017	Final	Audun Vennesland	Addressing comments from FRQ and LINZ
01.02.00	30.05.2017	Final	Audun Vennesland	Included input from LINZ
01.03.00	31.05.2017	Final	Audun Vennesland	Final version addressing comments from final internal review



Achieving the **BE**nefits of **SWIM** by making smart use of **Semantic Technologies**

This deliverable is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 699298 under the European Union's Horizon 2020 research and innovation programme.

Executive Summary

This document details the BEST ontology infrastructure developed in task 1.1 of the BEST Project. The ontology infrastructure includes a monolithic ontology developed from the ATM Information Reference Model (AIRM) UML model and a set of ontology modules, each representing different sub-areas of ATM information exchange, namely Aeronautical Information Exchange Model (AIXM) and ICAO Meteorological Information Exchange Model (IWXXM). All ontologies are formalised in the OWL ontology language. The ontologies will be used as vocabulary for describing and supporting retrieval of relevant aeronautical information by applications developed in other work packages of the project. Furthermore, the ontologies form a baseline for the establishment of guidelines describing how semantic technologies can be applied to support information exchange in a SWIM environment.

Table of Contents

Executive Summary	4
1 Introduction: About this document	7
1.1 Purpose	7
1.2 Intended Readership	7
1.3 Relationship to other deliverables	8
1.4 Acronyms and terminology	9
2 Background	10
2.1 Ontologies	10
2.2 Other relevant semantic technologies	11
2.3 Ontology Modularisation	12
2.4 AIRM	13
2.5 Exchange models for communication of digital ATM data	14
2.5.1 AIXM 5.1	14
2.5.2 IWXXM 1.1	15
2.5.3 FIXM Core 4.0.0	16
2.6 Transforming from UML to OWL	17
3 Development approach	18
3.1 Overall approach	18
3.2 Transformation from UML to OWL	19
3.2.1 XMI Representation of the AIRM UML Model	19
3.2.2 Transformation rules	20
4 The BEST Ontology Infrastructure	34
4.1 Presentation of the Monolithic Ontology	34
4.1.1 OWL Classes	34
4.1.2 Object Properties	35
4.1.3 Data Properties	36
4.1.4 Individuals	36
4.2 Presentation of the ontology modules	37
4.2.1 AIRM Modules	37
4.2.2 AIXM Modules	38
4.2.3 IWXXM Modules	40
4.3 Validation of the ontology infrastructure	42
5 Conclusions and future work	44
6 References	45

APPENDIX A: *OMG Mapping Guidelines*..... 47
APPENDIX B: *Post-processing of XMI* 48

1 Introduction: About this document¹

1.1 Purpose

This document details the BEST ontological infrastructure. The ontology infrastructure consists of a monolithic ontology developed using the AIRM (ATM Information Reference Model), as well as a set of ontology modules extracted from the ATM Information Reference Model (AIRM), Aeronautical Information Exchange Model (AIXM) and ICAO Meteorological Information Exchange Model (IWXXM). We have developed both a monolithic ontology and ontology modules in order to have a varied ontology infrastructure that other tasks in the project can benefit from. For example, an AIRM Compliance Validator will be developed in work package 1 that will help identify semantic discrepancies between AIRM, here represented by the monolithic ontology, and the ontology modules representing various sub-topics of the previously mentioned exchange models. Furthermore, work package 5 will establish a set of guidelines on scalability characteristics when applying semantic technologies in ATM information exchange and our assumption is that using monolithic ontologies compared to using ontology modules will impact on scalability metrics. The ontologies have been syntactically transformed from their respective UML models and as such they should be considered lightweight ontologies, without logical class constructors facilitating the expression of more complex knowledge. The ontologies described in this deliverable will be used, and if needed, semantically enriched, by other technical developments in the project. These developments include techniques for supporting information retrieval and data distribution strategies in work package 2; and prototype applications demonstrating how semantic technologies can be applied in a SWIM environment in work package 3. Furthermore, experiences gathered in developing the ontologies as well as when applying them in applications will help formulate guidelines describing how ontologies can be employed to support ATM information exchange.

1.2 Intended Readership

This document is primarily targeted towards people having an interest in:

- ATM information exchange
- Application of semantic technologies in ATM
- System-wide Information Management (SWIM)

¹ The opinions expressed herein reflect the author's view only. Under no circumstances shall the SESAR Joint Undertaking be responsible for any use that may be made of the information contained herein.

1.3 Relationship to other deliverables

Table 1. Relationship to other deliverables

Deliverable	Relationship
D1.2 AIRM Compliance Validator	The compliance validator prototype application will, using techniques from ontology matching and schema matching, contribute to detecting semantic differences between the monolithic ontology and the ontology modules. This can assist in monitoring of compliance between a reference ontology, here represented by the AIRM ontology and ontology modules (represented by the AIXM and IWXXM ontology modules).
D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base	The techniques described in D2.1 uses the ontologies described in D1.1 for describing aeronautical data products, supporting the formulation of information need, and aeronautical data discovery and retrieval.
D2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment	The techniques described in D2.2 will use the ontologies from D1.1 (among other ontologies) for supporting data distribution and consistency management.
D3.1 Prototype Use Case Scenarios	The use case scenarios defined in D3.1 provides a scope for both the AIRM ontology and the ontology modules.
D3.2 Prototype SWIM-enabled applications	The prototype applications developed in relation to D3.2 will utilise the ontologies developed in relation to D1.1.
D4.4 Tutorial for Software Developers	The tutorial for software developers will describe how semantic technologies, including the ontologies developed in relation to D1.1, can be applied when developing applications for SWIM.
D5.1 Scalability Guidelines for Semantic SWIM-based Applications	The scalability analysis performed in relation to D5.1 will consider how semantic technologies, including the ontologies described in D1.1, influence on the scalability of software applications using such technologies.
D5.2 Ontology Modularisation Guidelines for SWIM	The ontology modularisation guidelines will evaluate the “monolithic” ontology and the ontology modules developed in relation to D1.1 and provide guidelines on how this could be best accomplished in a SWIM operational setting.

1.4 Acronyms and terminology

Table 2. Acronyms and terminology

Definition	Explanation
AIRM	ATM Information Reference Model
AIXM	Aeronautical Information Exchange Model
FIXM	Flight Information Exchange Model
F-Logic	Frame Logic
IWXXM	ICAO Meteorological Information Exchange Model
METAR	Meteorological Aerodrome Report
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SESAR	Single European Sky ATM Research
SPARQL	SPARQL Protocol and RDF Query Language
SWIM	System-wide Information Management
TAF	Terminal Aerodrome Forecast
UML	Unified Modelling Language
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XSLT	EXtensible Stylesheet Language

2 Background

This section describes some of the background knowledge relevant for the development of the ontology infrastructure in BEST. The ontology infrastructure consists of a monolithic ontology and a set of ontology modules. A monolithic ontology is typically characterised as an ontology where all knowledge used for describing a complete domain, possibly encompassing several knowledge domains, is contained within a single model. There are normally no external references to other ontologies. In contrast, ontology modules are defined as self-contained, but coherent knowledge models, each responsible for describing a single, narrower knowledge domain, and typically taking part in a network of interdependent modules in order to represent a larger knowledge domain.

2.1 Ontologies

“An ontology describes a hierarchy of concepts related by subsumption relationships (a.k.a. specialization or is-a relationships), where suitable rules or statements (a.k.a. axioms) express other relationships between concepts and to constrain their intended interpretation. Following from this, an ontology O is defined as a tuple $\langle C, HC, RC, HR, I, RI, iC, iR, A \rangle$ where ontology concepts C are arranged in a subsumption hierarchy HC . Relations RC exist between pairs of concepts. The relations themselves can also be arranged in a hierarchy HR . Instance data is constituted by individuals I of specific concepts, and these individuals are interconnected by relational instances RI . Individuals and relational individuals are connected to concepts and relations by instantiations iC and iR respectively. A represents a set of axioms that can be used to further express constraints and induce logic from the ontology structure and associated instances” [1].

OWL (Web Ontology Language) [2] is a popular ontology language, and is the main formalism we use for the ontologies developed in the BEST project. As a very short introduction to later sections in this report, the main building blocks of OWL are entities and axioms. Entities are represented as either classes, object properties, data properties or individuals and they are all identified using IRIs (Internationalized Resource Identifiers). Classes are lightweight objects that in themselves do not hold information about definitions that may apply to themselves; this information is taken care of by the ontology object through axioms relating to the class level. Object properties are binary associations between individuals (real instances), and compared to UML associations OWL object properties can have additional characteristics (e.g. that an object property expression is transitive, inverse, ir-/reflexive, a-/symmetric, and so on). Data properties relate an individual to a concrete data value (for example a value of type `xsd:string`). As with classes, both object properties and data properties can be organised hierarchically. Axioms represent facts explicitly stated in the ontology, and there are several different categories of axioms. Declaration axioms declare the content (i.e. the entities) within the ontology. Logical axioms express logical assertions, such as specialisation (super-sub class) hierarchies, that two entities in an (or several) ontologies are equivalent, and property characteristics (e.g. that a property is transitive), among others. Annotation axioms merely associate information to entities, and do not affect the semantics of the ontology.

In BEST we will use ontologies in combination with semantic reasoning for supporting retrieval and filtering of ATM information. This reasoning consists of subsumption reasoning (i.e. discovering the

most specific yet relevant set of instance data through utilising the specialisation hierarchy expressed in ontologies) and membership reasoning (determining if an individual belongs to a particular class through inference). A research paper written as part of the BEST project describes in a detailed manner how ontologies are applied in combination with reasoning for supporting ATM information exchange in a SWIM setting [3].

There are a number of different tools for editing ontologies. For manual ontology engineering, inspection of the resulting BEST ontologies, and visualising extracts of the BEST ontology infrastructure later in this report, we have used the Protégé ontology editor [4] which is a free, open source ontology editor for OWL.

It is important to emphasise that the BEST ontologies described in this report aim to preserve the semantics expressed in the source UML models. For example, they do not include more advanced concepts such as complex classes (for example expressing intersection of classes), additional property characteristics (e.g. symmetric properties), and more advanced use of data types (e.g. facets). During the implementation of the more advanced techniques in WP 2 and prototype applications in WP 3, we may however see the need for incorporating more advanced expressions that enrich the semantics of the ontologies as well as introduce other semantic technologies that complement OWL. A short inventory of other relevant semantic technologies is presented in the next section.

2.2 Other relevant semantic technologies

The choice of semantic web technology depends on the reasoning task at hand and the type of represented information. In the semantic container approach, developed in WP 2 [5], one must decide whether a given semantic container contains more specific information than the other (subsumption reasoning) and whether individual data items belong to a specific semantic container (membership reasoning). The general-purpose language OWL allows to declaratively describe a broad range of knowledge while remaining decidable. Yet, for the representation of certain types of information, e.g., geospatial information, OWL is less suited. Thus, for certain types of information, more specialized ontology languages and domain-specific extension are preferable. GeoSPARQL [6], for example, provides representational concepts and query facilities for geospatial information in the semantic web. For other types of information, e.g., provenance, a simpler language such as RDF(S) [7] is sufficient, along with the corresponding query language – SPARQL [8]. Frame Logic (F-Logic) [9], on the other hand, is an ontology language similar to Datalog [10] in deductive database systems. The advantage of F-Logic is its concise object-oriented syntax as well as its rule-based reasoning capabilities which allow for the representation of information not expressible in OWL. The developed rules, however, must then be maintained.

For realizing the semantic container approach developed in WP 2, a viable technology choice seems to employ a mixture of OWL, RDF(S), SPARQL, GeoSPARQL, and F-Logic, whereby each technology serves a different purpose [5]. OWL could serve for the representation of base vocabulary and semantic description of container contents, and an OWL reasoner takes care of semantic container discovery. GeoSPARQL seems to be a viable alternative for the representation and querying of geospatial data, which also integrates well into OWL and RDF(S). RDF(S) could serve for the

representation of provenance information and other administrative metadata; SPARQL queries allow for a further restriction of discovered semantic containers based on the administrative metadata. F-Logic could serve for membership reasoning, i.e., actually populating the semantic containers with data items.

2.3 Ontology Modularisation

We distinguish between monolithic ontologies, which are typically characterised as ontologies large in size and complexity, and often spanning several different topics and knowledge areas, and ontology modules, which aim at providing ontology users with the knowledge they require, reducing the scope as much as possible to what is strictly necessary [11]. As mentioned earlier, an ontology consists of a set of axioms, i.e. logical statements, that holds some knowledge. An ontology module represents a particular subset of these axioms, and encapsulates a subset of the axioms compared to the “monolithic” ontology. For example, if we are interested in only the knowledge about the concept Aircraft in AIRM, we can represent this knowledge in an Aircraft ontology module, while disregarding other axioms from the AIRM ontology that are not relevant for expressing knowledge about an Aircraft.

There are a number of good incentives for operating with ontology modules rather than monolithic representations of ontologies. Ontology modules intuitively promote reuse, simpler maintenance, enable distributed engineering over different geographical locations and different areas of expertise, enable effective management and navigability, and will (in most cases) result in faster processing of reasoning operations. There are however several challenges related to modularisation too. One thing is finding the appropriate size of the modules. If they become too large, many of the issues with monolithic ontologies still remain. If they become too small, there might be too many modules in the network to manage, and maintaining an overview and keeping the ontology module network consistent can become challenging. There are also challenges related to defining the boundaries of the modules, and especially if the starting point is an already developed ontology that needs to be partitioned and potentially a large number of interdependencies between entities across different themes. Deciding on criteria for size and thematic boundaries depends on each particular use case. Once these criteria are settled, there are a number of different modularisation techniques to choose from. We briefly discuss two of them, ontology partitioning and ontology extraction. Ontology partitioning consists of decomposing the full set of axioms in an ontology into a set of modules and the union of all modules is equivalent to the original ontology. For example, Stuckenschmidt and Schlicht [12] applied structural characteristics of the ontology to determine suitable partitions of an input ontology. Representing the ontology as a weighted graph the technique computed the strength of the dependency between the different entities by analysing the ontology structure. Module extraction extracts modules from an ontology based on a definition of a sub-vocabulary or also called a seed signature. This signature consists of a set of entities (classes and/or properties and/or individuals) from which the technique recursively traverses through the ontology to gather related entities to be included in the module [13]. As a part of developing ontology modules in BEST, we partly use the latter approach.

2.4 AIRM

AIRM is a reference model that addresses semantic interoperability through harmonised and agreed definitions of the information being exchanged in ATM [14]. Semantic interoperability within ATM is accomplished by ensuring that all information being exchanged within ATM should be conformant with the definitions in AIRM. Such compliance is achieved following the rules of compliance defined in the SESAR AIRM Compliance Framework [15].

AIRM is formalised in UML and the UML model is decomposed into two main views having different abstraction levels:

- The Information Model, which defines information elements used in European ATM and their interrelations. In this view, the information elements are defined as entities without detailing their properties.
- The Logical Data Model, which refines the content of the Information Model to be used in more “operational” settings to support system and service development. In this view, the fundamental structure of the models and their entities is the same as in the information model, but each entity includes more detail, such as class properties and clearly defined association roles.

Within each view there is a decomposition of the model into different subject fields, where each subject field includes elements for particular areas of ATM. As an example there is a subject field for information elements describing the aircraft, another subject field for information elements describing meteorological information, etc. Figure 1 shows the decomposition of subject fields in AIRM.

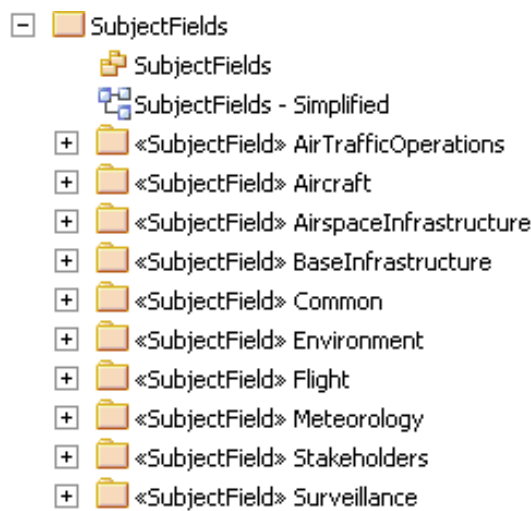


Figure 1. Subject Fields of the AIRM Data Model

In the development of the ontological infrastructure, we have only focused on the Logical Data Model. The reason for this is that the ontologies require a certain detail level in order to be useful for the applications developed in work package 2 and work package 3. This detail level is accomplished by transforming also the properties and associations in addition to the entities (UML classes) from the Logical Data Model. In addition to the Subject Field packages, the transformation also needs to include the Abstract package and the Data Types package since these contain properties and types

that apply to all elements in the Subject Fields. From AIRM we have developed a monolithic OWL ontology (see chapter 3) and in addition we have modularised this monolithic ontology into a set of ontology modules consisting of:

- Aircraft
- BaseInfrastructure
- Meteorology
- Common
- Stakeholders

2.5 Exchange models for communication of digital ATM data

While AIRM provides a reference model of all information to be exchanged within ATM to ensure semantic interoperability, there are different exchange models that define the structure and content and operationalise this for actual transmissions of digital ATM information. As mentioned in section 2.4, the information communicated must be compliant with AIRM to accomplish semantic interoperability. The exchange models may use different terminology, but there has to be a semantic trace between the information defined by these exchange models and AIRM, and semantic interoperability must be demonstrated following the rules defined by the AIRM Compliance Framework. The following subsections describe the exchange models most relevant to the BEST project.

2.5.1 AIXM 5.1

AIXM provides a conceptual data model and associated XML schemas for representing the format of digitally communicated aeronautical information. There is a direct link between the conceptual model and the XML schemas [16], and some of the modelling conventions applied represents a challenge for automated transformation to OWL. The requirements underlying the data model are defined in the AIX (Aeronautical Information Exchange) specification from Eurocontrol [17]. AIXM defines information related to, among other things, airports and heliports, airspace structures, organisations (including services they provide), geographical points and navigation aids, route information and flying restrictions. The basic information unit is a feature, which corresponds to a real world entity in the aeronautical environment (see list of packages in Figure 2). The features are considered dynamic, similarly to the environment they exist within. This dynamic is reflected by the notion of time slices for each feature which defines how a feature can either permanently or temporarily change in time [18]. In our work we have primarily focused on the AirportHeliport feature, but since there are strong dependencies to some of the other features we have also included these into our scope. The packages represent modules in our setting and the list of modules developed from AIXM are:

- AirportHeliport
- Geometry
- Obstacle
- Organisation
- Shared

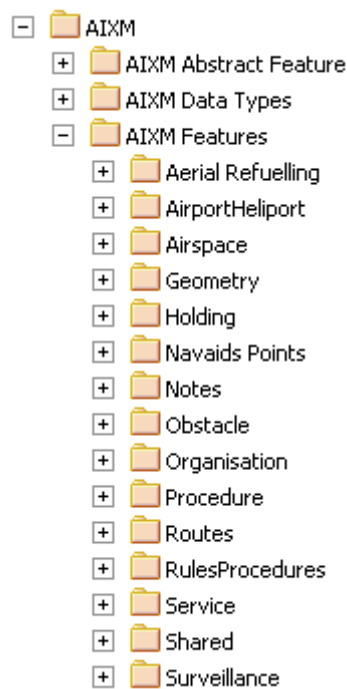


Figure 2. Structure of AIXM Conceptual UML Model

2.5.2 IWXXM 1.1

IWXXM is an exchange model that encompasses information about weather phenomenon. This includes actual and forecasted weather reports at aerodromes (METAR and TAF), weather conditions along the route (AIRMET), significant meteorological information (SIGMET), and advisories related to volcanic ash events and other extreme meteorological conditions (e.g. cyclones). As with AIXM, the UML model is targeted for XML schema development, something that makes it challenging for a completely automated transformation to OWL. In this work we have only focused on three of the IWXXM models, namely the METAR and TAF reports, and the Common package, which are all represented as ontology modules.

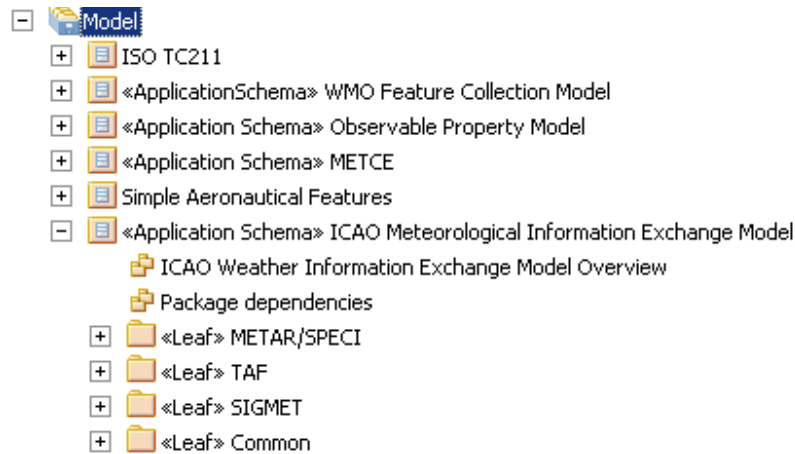


Figure 3. IWXXM UML Model Structure

2.5.3 FIXM Core 4.0.0

The Flight Information Exchange Model (FIXM) is a standardised model for the global exchange of flight information [19]. The model encompasses concepts such as flight and en route information, Arrival and Departure information, and basic information about other entities involved in a flight (Aircraft, Cargo, Trajectory, etc.), see Figure 3 for an overview of the FIXM UML model. As with AIXM and IWXXM, the UML model of FIXM is intended to be encoded to XML Schemas. While having an OWL representation of FIXM would be very interesting, this is outside of the defined scope of BEST, so this model has not been transformed to OWL.

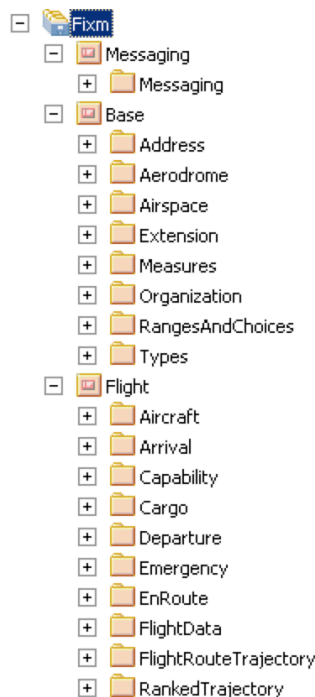


Figure 4. FIXM UML Model Structure

2.6 Transforming from UML to OWL

The overall approach used when transforming from the source UML models to OWL is illustrated in Figure 5.

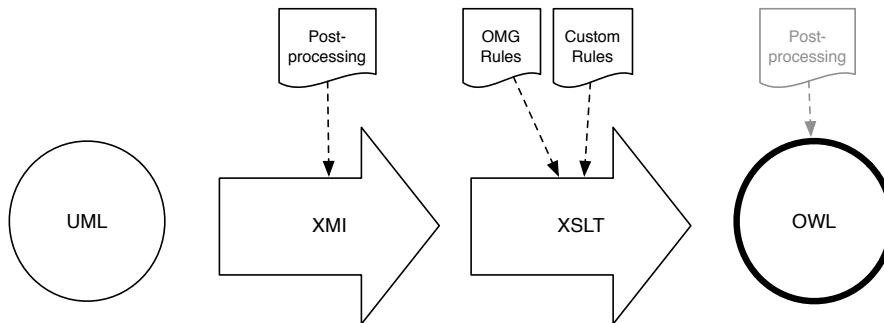


Figure 5. Overall transformation process from UML to OWL

When transforming the content of a UML model to OWL in an automated manner one needs to have a representation of the UML constructs that allows for a straightforward parsing and processing of them. Such representation is provided by XMI [20], a format that represent UML constructs in XML. The XMI file resulting from the generation from the UML editor Sparx Systems Enterprise Architect [21], needed some minor post processing. This post processing is described in Appendix B. The transformation from XMI to OWL is performed using XSLT. XSLT is a “rule-based” transformation scripting language typically used for transforming one XML representing into another XML representation [22], although other target representations are also possible (e.g. HTML). As an OWL ontology can be represented in XML (RDF-XML and OWL-XML), XSLT represents a viable option for our purposes. With regard to the rules guiding the transformation, OMG, a technology standards developing organisation (responsible for the standardisation of UML, among others) provides guidelines on how to map from UML to OWL [23] and these guidelines have helped formulate most of the rules applied in our transformation to OWL. However, we have extended the OMG specification with a few additional rules. These rules are explained in detail in section 3. Once an OWL representation is in place, some post-processing for some of the OWL ontologies is required. This relates to ensuring a good representation of <<choice>> constraints and UML association classes. This post-processing is further described in chapter 3.2.2.

3 Development approach

The development of a monolithic ontology and a set of ontology modules has followed three different approaches, primarily caused by different complexity of the UML structures of AIRM and the exchange models, as mentioned in the previous chapter, but also for the sake of experimenting with different techniques.

1. The monolithic version of the AIRM ontology has been transformed from UML automatically using XSLT.
2. Ontology modules of AIRM have been developed by taking the monolithic ontology as a starting point, and then apply an automated modularisation technique for extracting a set of modules.
3. The iWXXM and AIXM UML exchange models include a large number of package interdependencies, intricate data typing, and some other modelling conventions (e.g. XOR relationships and association classes) that makes it challenging to completely automate a transformation from UML to OWL. Therefore, the development of the ontology modules from IWXXM and AIXM has been performed semi-automatically in the sense that much of the laborious class and property axioms in OWL are established automatically using XSLT and the same set of rules as for AIRM. After this we have enhanced and structured the content manually in an ontology editor Protégé [4].

3.1 Overall approach

The overall approach is illustrated in Figure 6. Both the AIRM UML model and the exchange models are exported to XMI from the UML editor (Sparx Systems Enterprise Architect [21]) in the same manner. In the next step, we ran the XSLT transformation which transformed the XMI representation of AIRM to a monolithic ontology and AIXM and IWXXM (see section 3.2.2). In principle, more or less the same transformation rules are applied for all models in order to get to a monolithic AIRM ontology and an intermediate OWL representation for the exchange models, but as mentioned there are some differences in modelling procedures that prevents a completely generic approach among the three. The intermediate OWL ontologies generated from the XSLT transformation consist of all most entities present in the UML models, however quite a bit of manual engineering is still required in order to organise the proper relationship between classes, object properties, data properties and individuals, before the ontology modules are complete.

On the left hand side of Figure 6, after the monolithic AIRM ontology is generated, a seed signature guides the extraction of AIRM modules. The seed signature we composed consisted of a subject field name (e.g. BaseInfrastructure), since the UML package names for the subject fields are transformed to OWL classes in the resulting ontology. The module extraction functionality was implemented in Java using the OWL API library (version 4.1.2) and the theories behind it are described in [24]–[26].

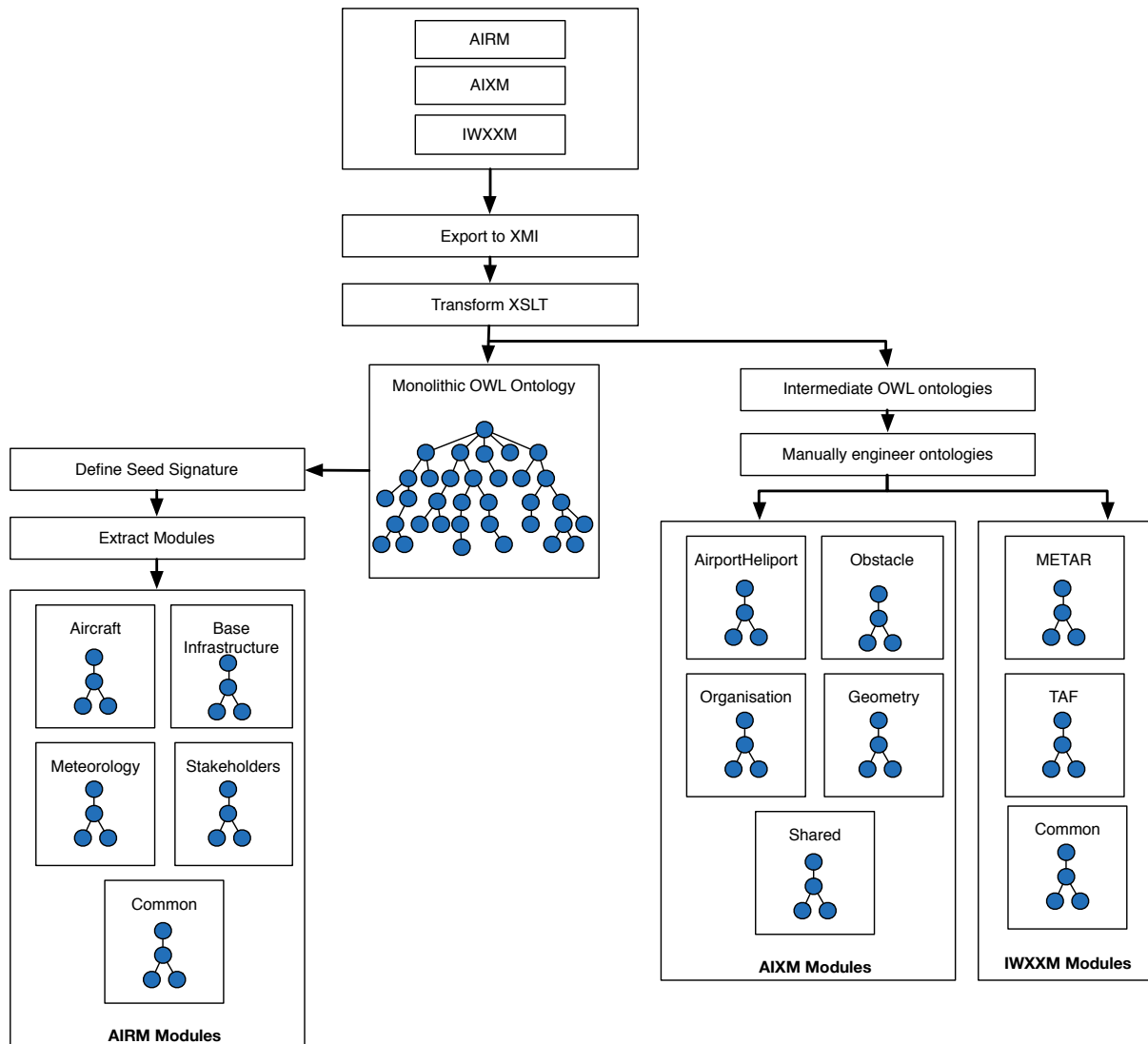


Figure 6. Overall approach

3.2 Transformation from UML to OWL

3.2.1 XMI Representation of the AIRM UML Model

The UML models are generated into separate XMI files which are used as a basis for generating the OWL ontology files. Figure 7 shows the nodes of the XMI which are important for the transformation. The numbers in parenthesis are used as reference numbers for the explanation of the transformation rules in the next section.



Figure 7. Structure of the XMI generated from the AIRM UML Model.

In the following the transformation rules are explained. These rules are applied both when developing the monolithic ontology and the intermediate OWL representation of the ontology modules.

3.2.2 Transformation rules

As described in section 3.1 the generation of the ontological infrastructure was performed using XSLT transformation from UML via XMI to OWL. The transformation rules are developed with support from the (non-normative) guidelines for mapping between UML and OWL in the OMG ODM specification (see section 2.6 and Appendix A). Each of the following transformation rules describe

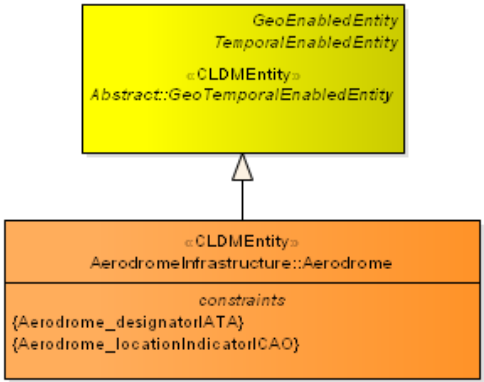
how we transform from a UML construct to an OWL construct. Table 3 below provides an overview of the transformations performed and a more detailed explanation of each transformation and its resulting OWL entity is provided in the subsequent parts of this chapter. The XSLT scripts used in the transformation are available from: <http://project-best.eu/downloads/ontologies/xslt/xslt.zip>

Table 3. Overview of transformations between UML and OWL

UML Construct	OWL Construct
UML Class	OWL Class
UML Generalization	OWL SubClassOf
UML Boolean attribute	OWL Class
UML Attribute with complex data type	OWL Object Property
UML Association	OWL Object Property
UML Aggregation (AIRM only)	OWL Object Property
UML Composition (AIXM and IWXXM)	OWL Object Property
UML Attribute with simple data type	OWL Data Property
UML Code List	OWL Class
UML Code List values	OWL Individuals

OWL Classes

- A UML Class is transformed to an OWL Class
- The name of the OWL Class is extracted from element/@name (7) if element/@xmi-type="uml:Class" (6) or element/@xmi-type="uml:Package". The reason for also transforming the UML packages to OWL Class is to maintain the order of elements in the UML model also in the resulting OWL ontology. This makes navigation in the ontology easier as the packages act as placeholders for content relevant for a particular thematic. OWL Classes representing UML Packages are underscored, so for example the AerodromeInfrastructure UML Package is represented as `_AerodromeInfrastructure_` in the OWL Ontology to distinguish it from regular OWL Classes.
- `subClassOf` statements are created if there is a match between the value of element/@name (7) and the value of connector/source/model/@name (26), and if the properties/@ea_type (36) is 'Generalization'. If so the value in any target/model/@name (31) is included as (super) classes in the `subClassOf` statements.

UML Representation	OWL Representation
 <pre> classDiagram class GeoEnabledEntity { <<CLDMEntity>> TemporalEnabledEntity Abstract::GeoTemporalEnabledEntity } class AerodromeInfrastructure { <<CLDMEntity>> AerodromeInfrastructure::Aerodrome constraints {Aerodrome_designator(ATA)} {Aerodrome_locationIndicator(CAO)} } GeoEnabledEntity < -- AerodromeInfrastructure </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Aerodrome --> <owl:Class rdf:about="&airm;Aerodrome"> <rdfs:subClassOf rdf:resource="&airm;GeoTemporalEnabledEntity"/> <rdfs:subClassOf rdf:resource="&airm;_AerodromeInfrastructure_"/> </owl:Class> </pre>

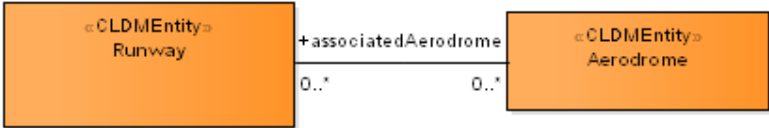
Boolean attributes

- UML Boolean attributes are transformed to OWL Classes and are represented as sub-classes of the class they belong to in UML.
- Boolean attributes are identified by checking if attribute/properties/@type="Boolean" (20).
- The attribute name is extracted from attribute/@name (16)
- The OWL Class name is defined by a combination of [OWL class representing the UML class the attribute belongs to] – [attribute name]

UML Representation	OWL Representation
<pre> classDiagram class CLDMEntity { + certificationDate: Date [0..1] + certificationExpirationDate: Date [0..1] + certificationICAO: CodeICAOCertificationType [0..1] + controlType: CodeMilitaryOperationsType [0..1] + country: CharacterString [0..1] + dateMagneticVariation: Date [0..1] + designator: CharacterString [0..1] + designatorATA: CharacterString [0..1] + fieldElevation: ValVerticalDistanceType [0..1] + fieldElevationAccuracy: ValVerticalDistanceType [0..1] + isAbandoned: Boolean [0..1] + isMagneticCheckedLocationAvailable: Boolean [0..1] + isLandingDirectionIndicatorAvailable: Boolean [0..1] + isPrivateUse: Boolean [0..1] + isSecondaryPowerSupplyAvailable: Boolean [0..1] + isWindDirectionIndicatorAvailable: Boolean [0..1] + locationIndicatorCAO: CharacterString [0..1] + lowestTemperature: ValTemperatureType [0..1] + magneticVariation: Angle [0..1] + magneticVariationAccuracy: Angle [0..1] + magneticVariationChange: Angle [0..1] + name: CharacterString [0..1] + referenceTemperature: ValTemperatureType [0..1] + transitionAltitude: ValVerticalDistanceType [0..1] + transitionLevel: Integer [0..1] + type: CodeAerodromeType [0..1] + verticalDatum: CodeVerticalDatumType [0..1] } </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Aerodrome-isAbandoned --> <owl:Class rdf:about="&airm;Aerodrome-isAbandoned"> <rdfs:subClassOf rdf:resource="&airm;Aerodrome"/> <rdfs:comment>An indicator that the aerodrome is no longer in operational use, but its infrastructure is still present and visible from the air.</rdfs:comment> </owl:Class> </pre>

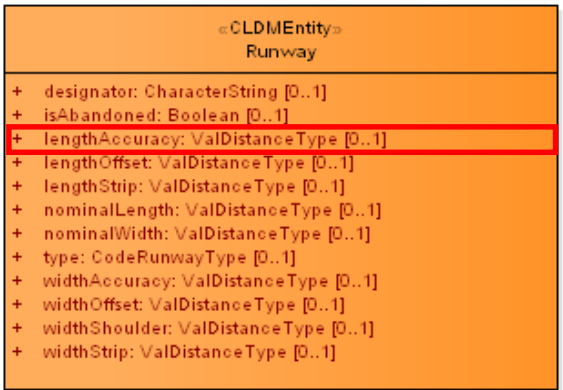
Associations

- UML Associations are represented as OWL Object Properties
- The domain of the OWL Object Property is extracted from connector/source/model/@name (26) and the range class from connector/target/model/@name (31)
- The association name is extracted from connector/target/role/@name (34)
- The subPropertyOf statement ensures that this object property is grouped as a subProperty (along with other object properties having Runway as domain class) under the main Runway_OP_ object property.
- The naming convention used for the OWL Object Property is [Domain class name] – [association name]

UML Representation	OWL Representation
 <pre> classDiagram class Runway { <<CLDMEntity>> } class Aerodrome { <<CLDMEntity>> } Runway "0..*" -- "0..*" Aerodrome : +associatedAerodrome </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Runway-associatedAerodrome --> <owl:ObjectProperty rdf:about="&airm;Runway-associatedAerodrome"> <rdfs:range rdf:resource="&airm;Aerodrome" /> <rdfs:domain rdf:resource="&airm;Runway" /> <rdfs:subPropertyOf rdf:resource="&airm;Runway_OP_" /> </owl:ObjectProperty> </pre>

Attributes with complex data types

- Attributes with complex data types are transformed to OWL Object Properties
- An assumption made by the transformation script is that in AIRM all complex data types ends with "...Type"
- The attribute name is extracted from attributes/attribute/@name (16)
- The domain class is extracted from element/@name (7) and the range class from attributes/attribute/properties/@type (20)
- The naming convention used is [*Domain class name*] – [*attribute name*]

UML Representation	OWL Representation
 <pre> classDiagram class Runway { +designator: CharacterString [0..1] +isAbandoned: Boolean [0..1] +lengthAccuracy: ValDistanceType [0..1] +lengthOffset: ValDistanceType [0..1] +lengthStrip: ValDistanceType [0..1] +nominalLength: ValDistanceType [0..1] +nominalWidth: ValDistanceType [0..1] +type: CodeRunwayType [0..1] +widthAccuracy: ValDistanceType [0..1] +widthOffset: ValDistanceType [0..1] +widthShoulder: ValDistanceType [0..1] +widthStrip: ValDistanceType [0..1] } </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Runway-lengthAccuracy --> <owl:ObjectProperty rdf:about="&airm;Runway-lengthAccuracy"> <rdf:type rdf:resource="&owl;FunctionalProperty"/> <rdfs:domain rdf:resource="&airm;Runway"/> <rdfs:subPropertyOf rdf:resource="&airm;Runway_OP_"/> <rdfs:range rdf:resource="&airm;ValDistanceType"/> </owl:ObjectProperty> </pre>

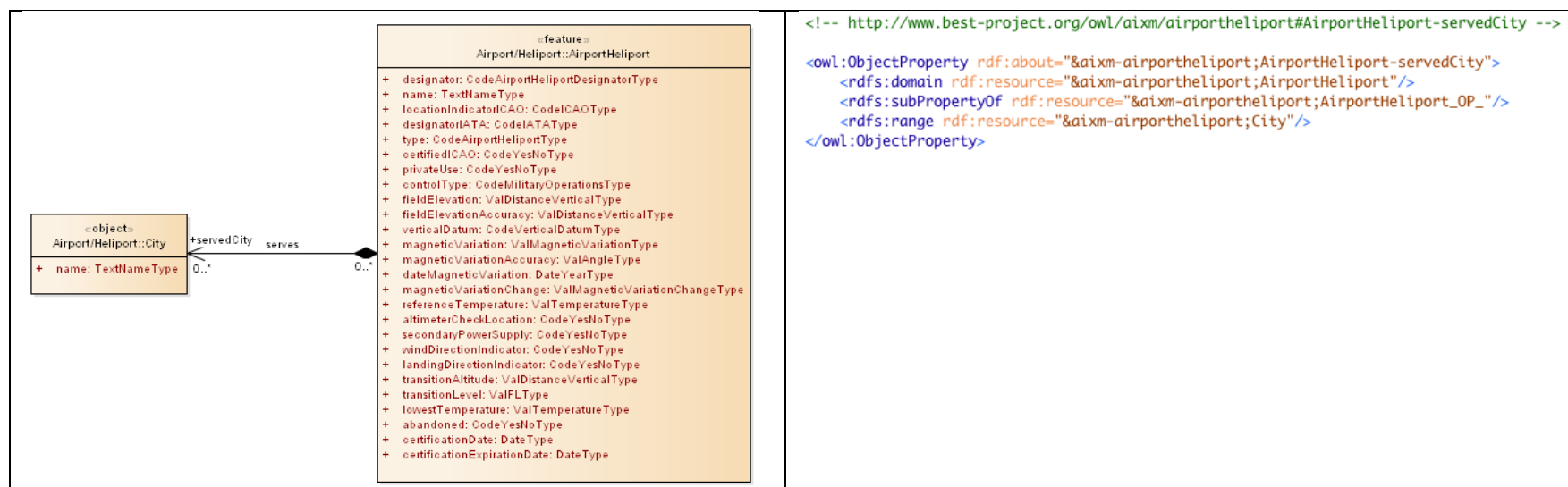
Aggregation relationships

- UML Aggregation is transformed to an OWL Object Property
- An aggregation relationship is identified by connector/source/type/@aggregation="shared" (28)
- The aggregation name is retrieved from connector/target/role/@name (34)
- The domain class is extracted from connector/source/model/@name (26) and the range class from connector/target/model/@name (31)

UML Representation	OWL Representation
<pre> classDiagram class Runway["<CLDMEntity>\nRunway"] class SurfaceCharacteristics["<CLDMObject>\nSurfaceCharacteristics"] Runway o-- "0..1" SurfaceCharacteristics : +surfaceProperties </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Runway-surfaceProperties --> <owl:ObjectProperty rdf:about="&airm;Runway-surfaceProperties"> <rdf:type rdf:resource="&owl;FunctionalProperty"/> <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/> <rdfs:domain rdf:resource="&airm;Runway"/> <rdfs:subPropertyOf rdf:resource="&airm;Runway_OP_"/> <rdfs:range rdf:resource="&airm;SurfaceCharacteristics"/> </owl:ObjectProperty> </pre>

Composition relationships

AIXM and IWXMM do not use aggregation relationships, but composition relationships. This is considered a stronger composition than aggregation (which is used in AIRM). However, as the rationale for using composition rather than aggregation is to ensure that the transformation from UML to XML schemas is performed unambiguously² according to ISO 19118 rules, and not for expressing life-cycle dependency between classes, we have transformed such relationships to normal OWL object properties.



² ISO 19118 defines encoding rules when transforming from UML to XML Schemas. Here, a regular association is transformed to an xlink, a composition association is transformed to an inline element, while an aggregation association can become either one.

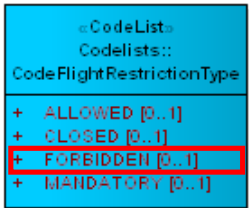
Attributes with simple data types

- UML Attributes with simple data types are transformed to OWL Data Properties
- The attribute name is extracted from attributes/attribute/@name (16)
- The domain class is extracted from element/@name (7). For the range we have converted all occurrences of CharacterString to xsd:string.

UML Representation	OWL Representation
<pre> classDiagram class Runway { +designator: CharacterString [0..1] +isAbandoned: Boolean [0..1] +lengthAccuracy: ValDistanceType [0..1] +lengthOffset: ValDistanceType [0..1] +lengthStrip: ValDistanceType [0..1] +nominalLength: ValDistanceType [0..1] +nominalWidth: ValDistanceType [0..1] +type: CodeRunwayType [0..1] +widthAccuracy: ValDistanceType [0..1] +widthOffset: ValDistanceType [0..1] +widthShoulder: ValDistanceType [0..1] +widthStrip: ValDistanceType [0..1] } </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#Runway-designator --> <owl:DatatypeProperty rdf:about="&airm;Runway-designator"> <rdf:type rdf:resource="&owl;FunctionalProperty"/> <rdfs:label>designator</rdfs:label> <rdfs:comment>The designator of a runway, used to uniquely identify <rdfs:subPropertyOf rdf:resource="&airm;Runway_DP_" /> <rdfs:range rdf:resource="&xsd:string" /> </owl:DatatypeProperty> </pre>

Code lists and their values

- UML Code lists are represented as OWL Classes while their values are represented as OWLNamedIndividuals³
- The name of the code list is extracted from element/@name (7) and element/properties/@stereotype="CodeList"
- The code list value is retrieved from attributes/attribute/@name (16)
- The name of the OWL Class is defined by [Code list name] – [Code list value]

UML Representation	OWL Representation
 <pre> classDiagram class CodeList { <<CodeList>> CodeLists: CodeFlightRestrictionType + ALLOWED [0..1] + CLOSED [0..1] + FORBIDDEN [0..1] + MANDATORY [0..1] </pre>	<pre> <!-- http://www.best-project.org/owl/AIRM#CodeFlightRestrictionType-FORBIDDEN --> <owl:NamedIndividual rdf:about="&airm;CodeFlightRestrictionType-FORBIDDEN"> <rdf:type rdf:resource="&airm;CodeFlightRestrictionType"/> <rdfs:label>FORBIDDEN</rdfs:label> <rdfs:comment>Forbidden</rdfs:comment> </owl:NamedIndividual> </pre>

³ Alternatively, the code lists could be represented as Enumerations expressed in OWL, capturing that they define a fixed set of values, or as a datatype with an enumeration of values.

Data types and code lists

Data type and code lists are represented differently in AIXM and IWXXM and AIRM, and this must be reflected by the transformation rules. The data type construction in AIXM consists of a two-step creation procedure, as illustrated in Figure 8. When these models are transformed to XSD there is as a first step created a simple type that grounds the code list/data type to its simple data type (in this example string and date) and as a second step a complex type is created that defines the nilReason attribute. A similar convention is applied for some of the data types in IWXXM. So for all code lists there is also a corresponding data type class “nilReason” attribute for expressing reasons why the attribute referencing the code list is empty. The reasons for “nil” are defined as values in an enumeration called “NilReasonEnumeration”.

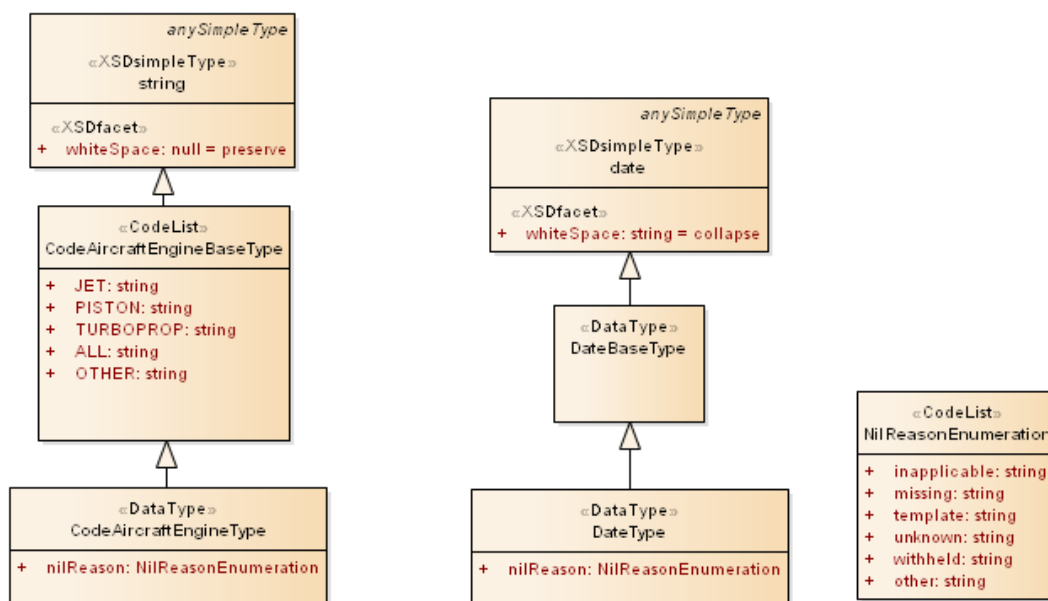


Figure 8. Modelling of code lists and data types in AIXM. To the left how a code list is first grounded to a simple XSD type (string) and as a second step the

We have simplified this for the data properties in the OWL ontology and have traced each complex data type back to its simple data type. So for example DateType is represented by xsd:date in the OWL ontology. The code lists employed in AIXM remain intact in the ontology and are processed as described in section 3.2.2. Code lists in IWXXM are not populated with values in the UML model, only notes suggesting example values are provided, so an automated transformation from UML to OWL was not possible for these. An example of this is shown in Figure 9.

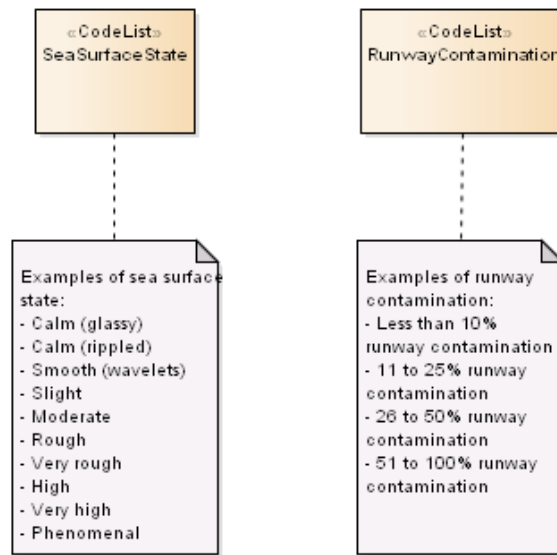


Figure 9. Code list values provided as examples in notes in the IWXXM UML model

XOR Relationships

XOR relationships, where the “deciding” class is stereotyped with <<choice>>, signify that at least one but also maximum one of the associated classes apply.

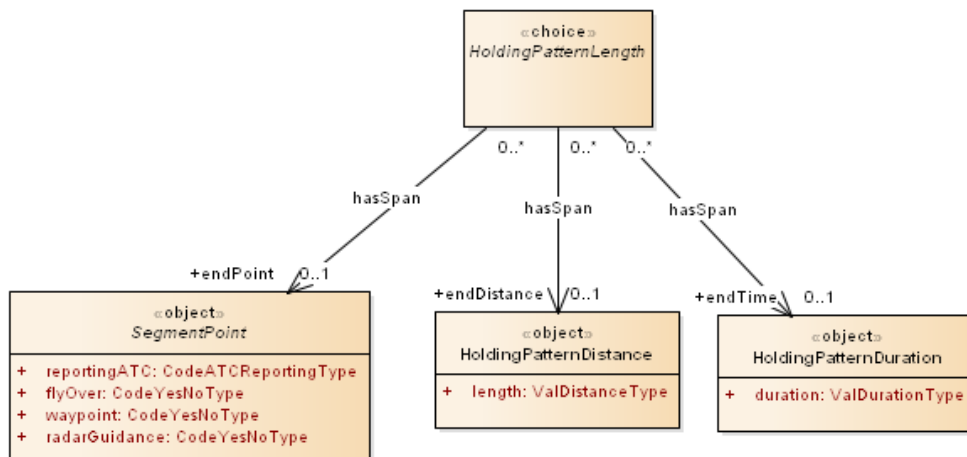


Figure 10. Use of pattern constraints to define XOR (choice)

These structures were managed in the ontology editor after transformation from XMI and were implemented as one functional object property <classname>-CHOICE with a set of sub-properties (one sub-property per possible choice) and specified with the class that it must have at least one of these sub-properties. So using the UML model depicted in Figure 10 as example, we have defined the following OWL constructs in the Protégé OWL editor to express XOR:

```

ObjectProperty: HoldingPatternLength-CHOICE
  Characteristics: Functional
  Domain: HoldingPatternLength
  Range: HoldingPatternDistance or HoldingPatternDuration or SegmentPoint

ObjectProperty: HoldingPatternLength-endDistance
  SubPropertyOf:
    HoldingPatternLength-CHOICE
  Range:
    HoldingPatternDistance

ObjectProperty: HoldingPatternLength-endPoint
  SubPropertyOf:
    HoldingPatternLength-CHOICE
  Range:
    SegmentPoint

ObjectProperty: HoldingPatternLength-endTime
  SubPropertyOf:
    HoldingPatternLength-CHOICE
  Range:
    HoldingPatternDuration

Class: HoldingPatternLength
  SubClassOf:
    (HoldingPatternLength-endDistance
     some owl:Thing) or
    (HoldingPatternLength-endPoint
     some owl:Thing) or
    (HoldingPatternLength-endTime some
     owl:Thing)
    
```

Association classes

Association classes are introduced when additional information about a relationship is required and are modelled as shown in Figure 11.

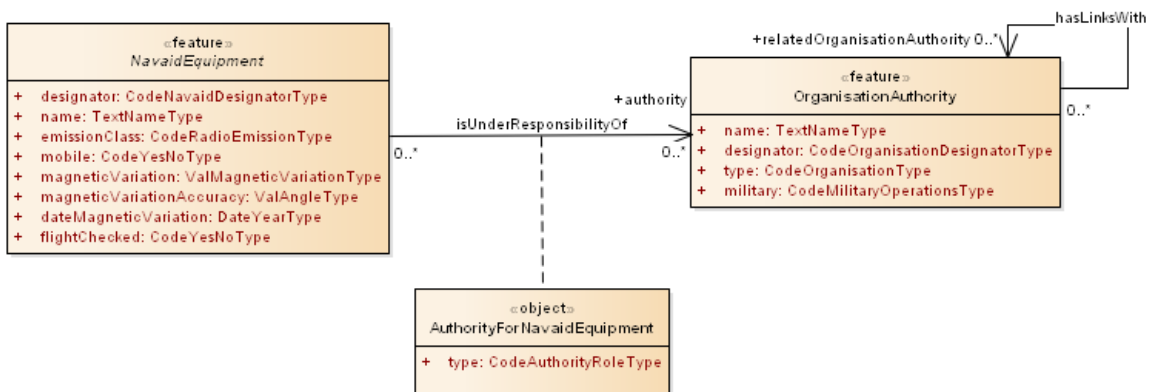


Figure 11. Association classes in AIXM

Here we implemented only one OWL class (AuthorityForNavaidEquipment) representing both the association and the association class and an object property per association end.

```
Datatype: CodeAuthorityRoleType
Class: AuthorityForNavaidEquipment
Class: NavaidEquipment
Class: OrganisationAuthority
ObjectProperty: AuthorityForNavaidEquipment-navaidEquipment
  Characteristics: Functional
  Domain: AuthorityForNavaidEquipment
  Range: NavaidEquipment
ObjectProperty: AuthorityForNavaidEquipment-authority
  Characteristics: Functional
  Domain: AuthorityForNavaidEquipment
  Range: OrganisationAuthority
DataProperty: AuthorityForNavaidEquipment-type
  Characteristics: Functional
  Domain: AuthorityForNavaidEquipment
  Range: CodeAuthorityRoleType
```

Additionally, we defined that the association-ends are mandatory properties:

```
Class: AuthorityForNavaidEquipment
  SubClassOf:
    AuthorityForNavaidEquipment-NavaidEquipment some NavaidEquipment,
    AuthorityForNavaidEquipment-authority some OrganisationAuthority
```

And that the association ends together form a compound key for the association class (i.e., two links with the same source and target are one and the same link).

```
Class: AuthorityForNavaidEquipment
  HasKey:
    AuthorityForNavaidEquipment-NavaidEquipment,
    AuthorityForNavaidEquipment-authority
```

4 The BEST Ontology Infrastructure

The monolithic ontology and the ontology modules can be downloaded as a zip file from:

<http://project-best.eu/downloads/ontologies/ontologies.zip>

4.1 Presentation of the Monolithic Ontology

In total the AIRM ontology consists of 1177 classes, 3272 object properties, 1972 data properties and 3727 individuals.

4.1.1 OWL Classes

Figure 12 shows the overall class structure of the AIRM OWL Ontology. One of our aims has been to mimic the structure of the AIRM UML model to the extent feasible, and the OWL classes including underscores represent the package hierarchy in UML, while classes not underscored represent “regular” classes. This convention keeps the transformed classes in the intended order thus makes it easier to navigate in the resulting ontology using an editor such as Protegé [4] for instance.

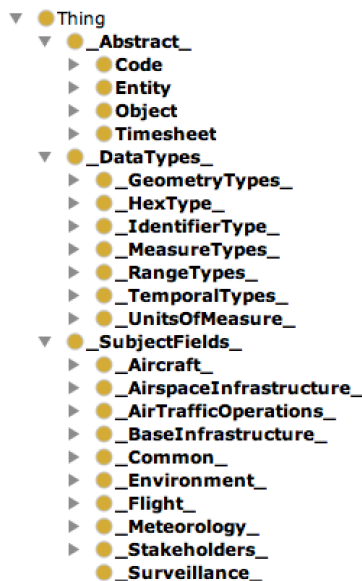


Figure 12. Overall class structure of the AIRM Ontology

There are four types of OWL classes in the AIRM Ontology:

- Classes representing UML package (as described above).
- Classes transformed from UML classes. See Figure 13.
- Classes transformed from Boolean attributes in UML. These are represented as sub-classes of the class they belong to as attributes in UML. See Figure 14.
- Classes transformed from code lists in UML. See Figure 15.

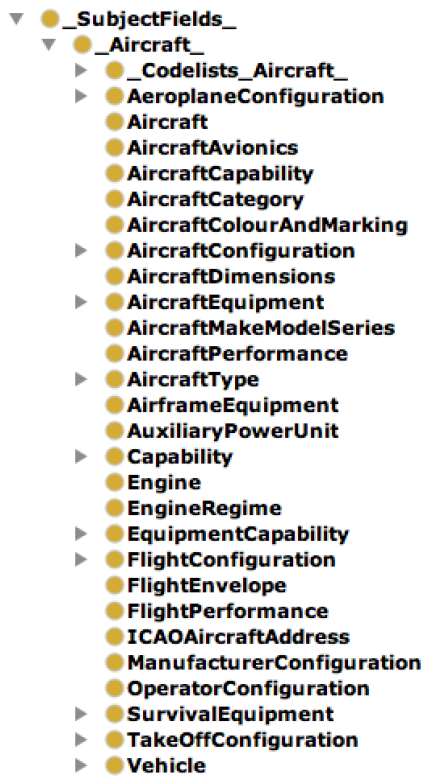


Figure 13. OWL classes representing the AIRM Aircraft subject field

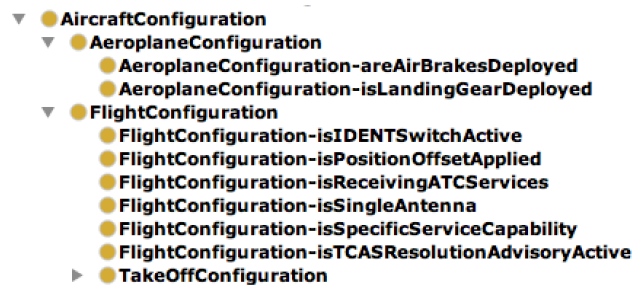


Figure 14. Classes representing Boolean attributes

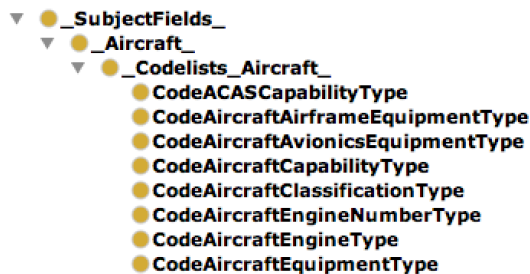


Figure 15. Classes representing code lists. As in the AIRM UML model there is a code list package for every subject field

4.1.2 Object Properties

As with the OWL classes, there are different types of object properties generated from UML.

- Regular UML associations, see Figure 16.
- Aggregation associations, see Figure 17.
- Attributes with complex data types, see Figure 18.



Figure 16. A regular UML association is transformed to an OWL object property (with domain Runway and range Aerodrome)



Figure 17. A UML aggregation relationship is transformed to an OWL Object Property (with domain Runway and range SurfaceCharacteristics)

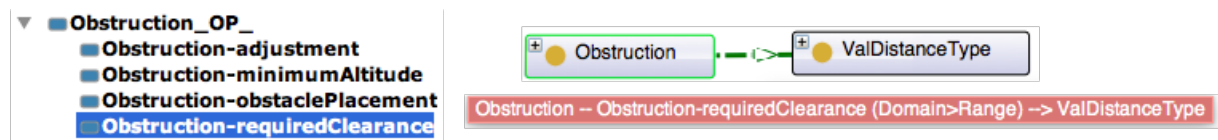


Figure 18. A UML Complex data type is transformed to an OWL Object Property. The domain class is the class holding the attribute referring to the complex data type and the range is the class transformed from the complex data type class in UML

4.1.3 Data Properties

UML attributes with simple data types are transformed to OWL data properties.

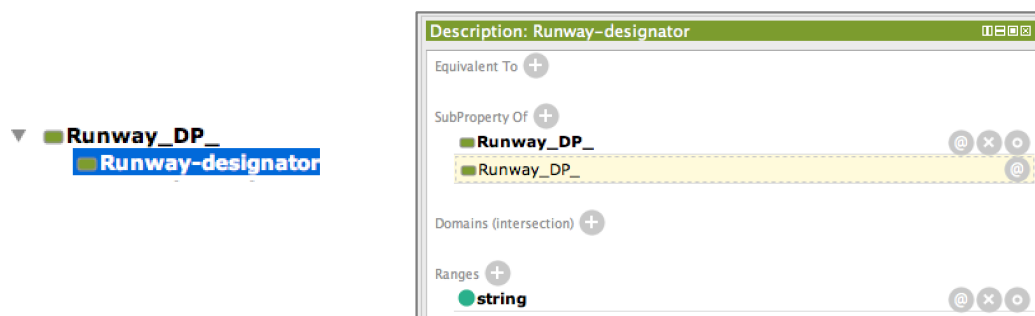


Figure 19. A UML simple attribute is transformed to an OWL Data Property. The range in this case is string and since all attributes in AIRM have cardinality 0..1 this becomes a functional property

4.1.4 Individuals

OWL Individuals are used to values from the code lists in AIRM. These are prefixed by the name of the code list to support navigation.

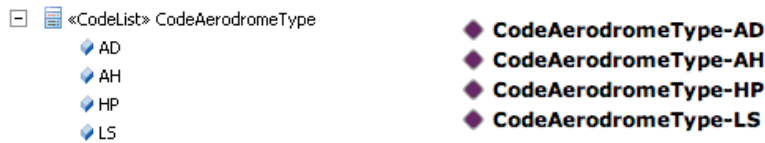


Figure 20. UML code list values are transformed to OWL individuals. The code list values are prefixed with the name of the code list they belong to and have the code list as type

4.2 Presentation of the ontology modules

This section lists the ontology modules extracted from the AIRM monolithic ontology along with some statistics to get a sense of their size and complexity.

4.2.1 AIRM Modules

The **Aircraft ontology module** consists of 93 classes, 84 object properties, 33 data properties and 182 individuals.

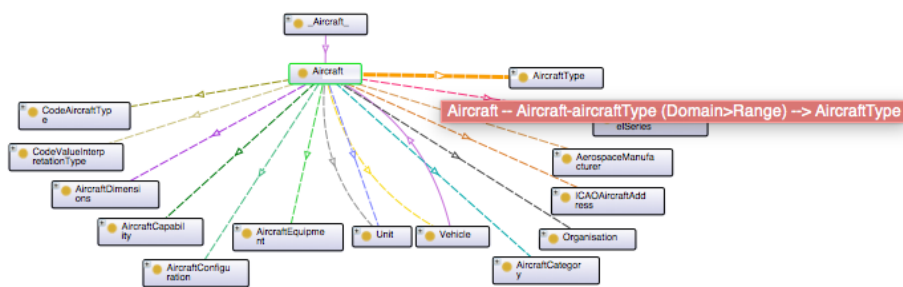


Figure 21. OWL representation of the Aircraft class along with object properties related to other OWL classes

The **BaseInfrastructure ontology module** consists of 357 classes, 463 object properties, 133 data properties and 1574 individuals.

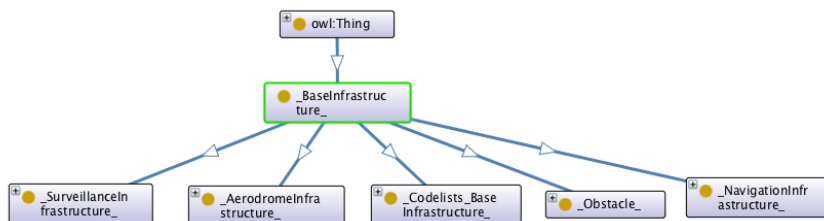


Figure 22. OWL representation of the BaseInfrastructure ontology module and its sub classes

The **Common ontology module** consists of 78 classes, 44 object properties, 19 data properties and 396 individuals.

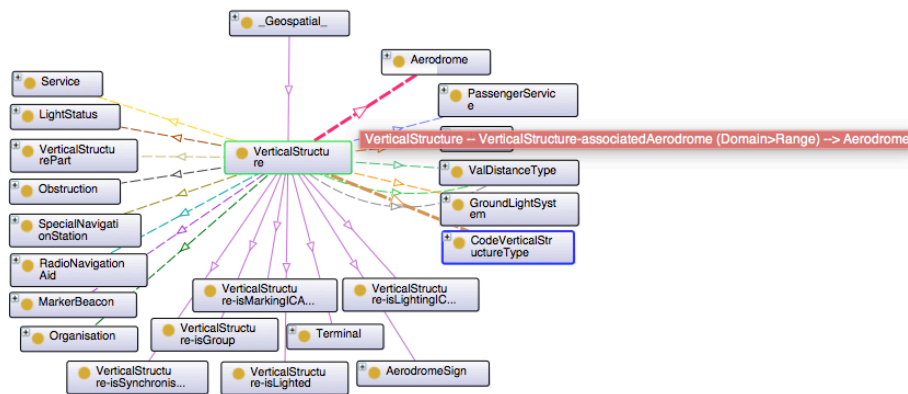


Figure 23. Example showing the VerticalStructure OWL Class from the Commons OWL ontology module

The **Stakeholders ontology module** consists of 148 classes, 131 object properties, 40 data properties and 316 individuals.

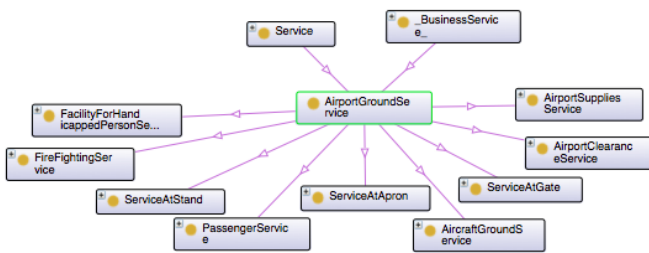


Figure 24. Example showing the AirportGroundService OWL Class as a concept in the Stakeholders ontology module

The **Meteorology ontology module** consists of 74 classes, 69 object properties, 15 data properties and 97 individuals.

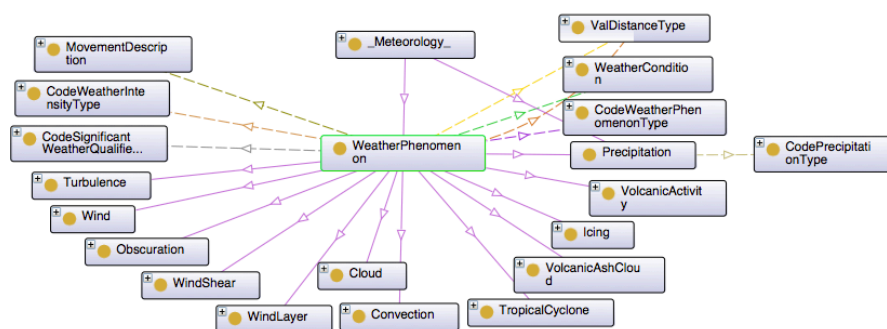


Figure 25. Example showing the WeatherPhenomenon OWL Class as a concept in the Meteorology ontology module

4.2.2 AIXM Modules

The **AirportHeliport ontology module** consists of 196 classes, 312 object properties, 133 data properties and 569 individuals.

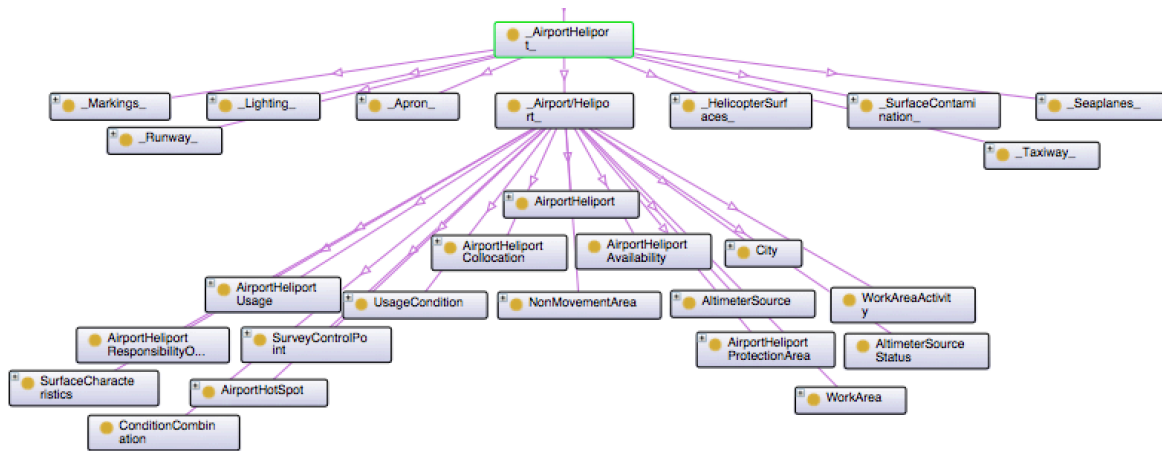


Figure 26. OWL representation of the AirportHeliport ontology module from AIXM

The **Obstacle ontology module** consists of 24 classes, 35 object properties, 10 data properties and 132 individuals.

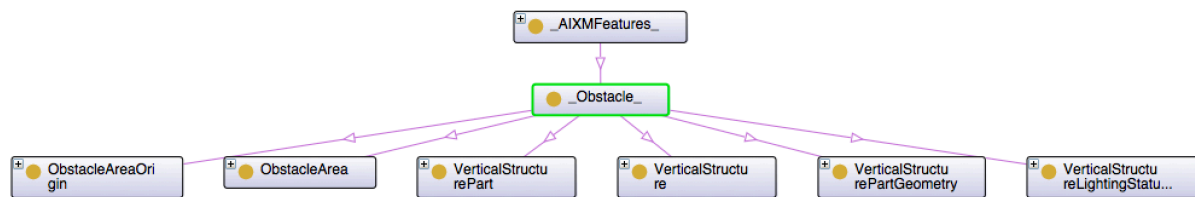


Figure 27. OWL representation of the Obstacle ontology module from AIXM

The **Organisation ontology module** consists of 15 classes, 22 object properties, 8 data properties and 23 individuals.

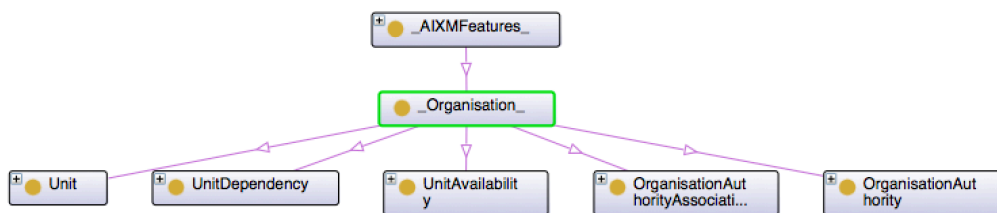


Figure 28. OWL representation of the Organisation ontology module from AIXM

The **Geometry ontology module** consists of 11 classes, 8 object properties, 19 data properties and 4 individuals.

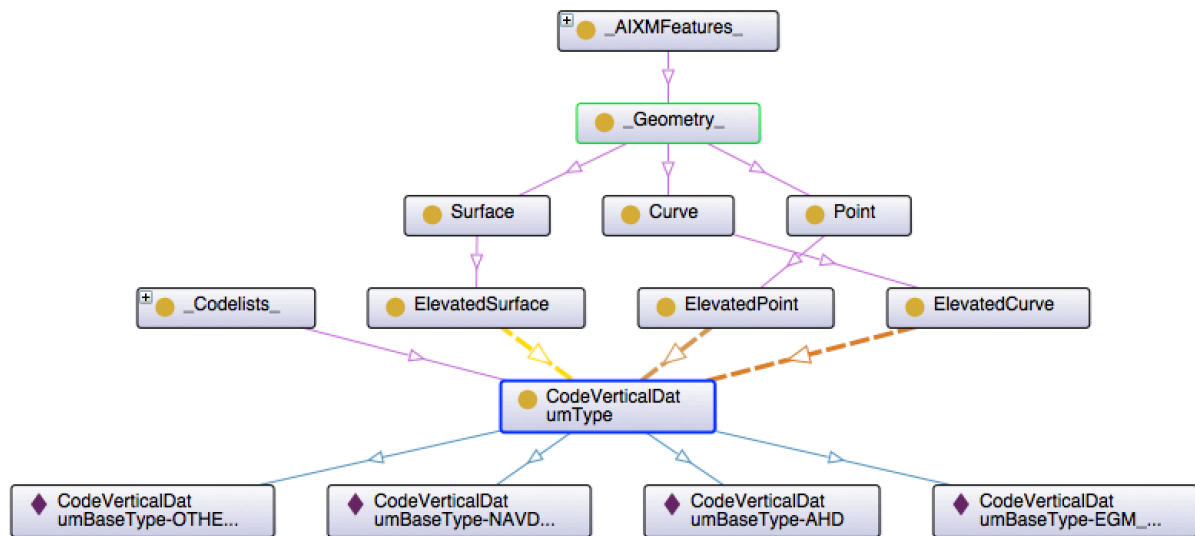


Figure 29. OWL representation of the Geometry ontology module from AIXM

The **Shared ontology module** consists of 33 classes, 39 object properties, 36 data properties and 103 individuals.

From the Shared UML package in AIXM we have extracted a single module for the following sub-packages:

- Address
- Light Element
- Meteorology
- Schedules

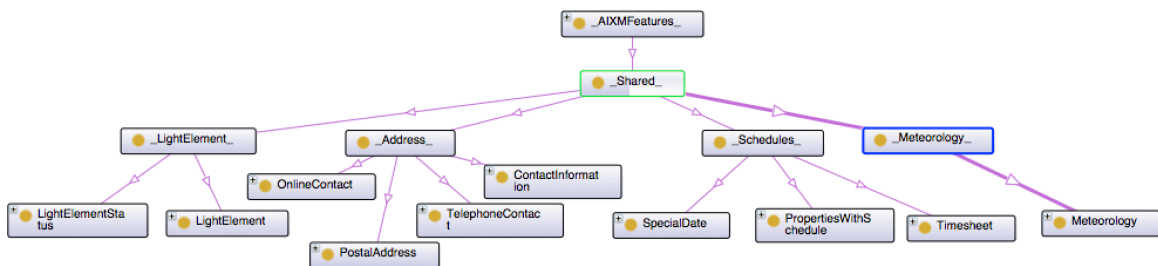


Figure 30. OWL representation of the Shared ontology module from AIXM

4.2.3 IWXXM Modules

The IWXXM modules imports the external W3C Time ontology for describing temporal aspects. The W3C Time ontology is using the following namespace: <http://www.w3.org/2006/time#>.

The **METAR ontology module** consists of 56 classes, 70 object properties, 53 data properties and 25 individuals.

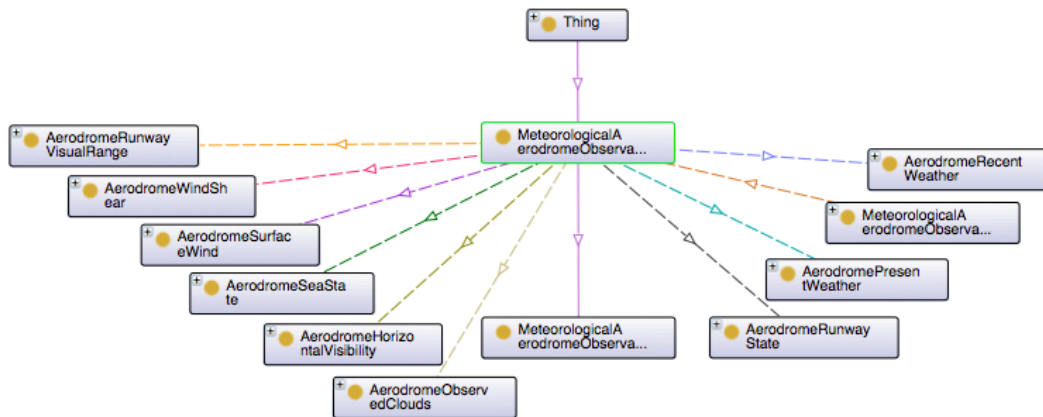


Figure 31. Example showing the MeteorologicalObservationRecord OWL class in the METAR ontology module

The **TAF ontology module** consist of 38 classes, 56 object properties, 32 data properties and 28 individuals. This ontology imports the W3C time ontology for expressing temporal aspects.

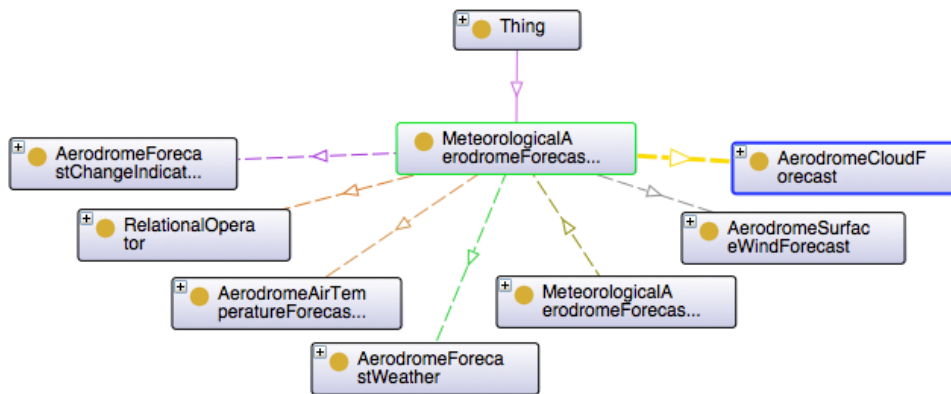


Figure 32. Example showing the MeteorologicalAerodromeForecastRecord OWL class from the TAF ontology module

The **Common ontology module** consists of 10 classes, 2 object properties, 0 data properties and 0 individuals.

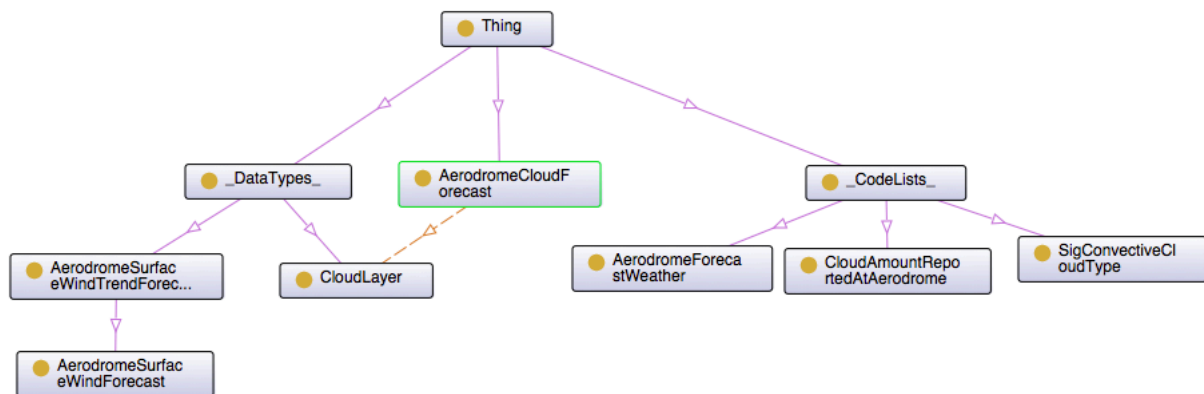


Figure 33. The full Common ontology module

4.3 Validation of the ontology infrastructure

Due to the sheer size of the ontologies, a proper validation of the BEST ontology infrastructure is a challenge, and needs to be performed in several iterations. The validation will hence take place in two overall rounds. The first round of validation has been performed during and after the development of the ontologies. Here, both the monolithic ontology and the ontology modules have been validated using the reasoning facilities in Protégé and by manually comparing the ontologies against the UML models they are sourced from, as well as XML schemas and instances in the case of the AIXM and IWXXM modules. The Protégé reasoner can detect logical inconsistencies in the ontologies, for example that the model is constructed in a fashion that prevents them from having individuals associated to them. In the second validation round, the ontologies will undergo validation when they are being employed in the technical implementation in work packages 1 (development of the AIRM Compliance Validator) and 3 (development of a prototype SWIM-enabled application), and a desktop validation will take place in work packages 2 (techniques for ontology-based data description and discovery; and techniques for data distribution and consistency management), 4 (development of a tutorial for software developers) and 5 (producing guidelines for scalability of semantic technologies). Lessons learned from the validation activity and the final set of ontologies in the ontology infrastructure will be summarised in deliverable D5.2, “Ontology Modularisation Guidelines for SWIM” close to the end of the project.

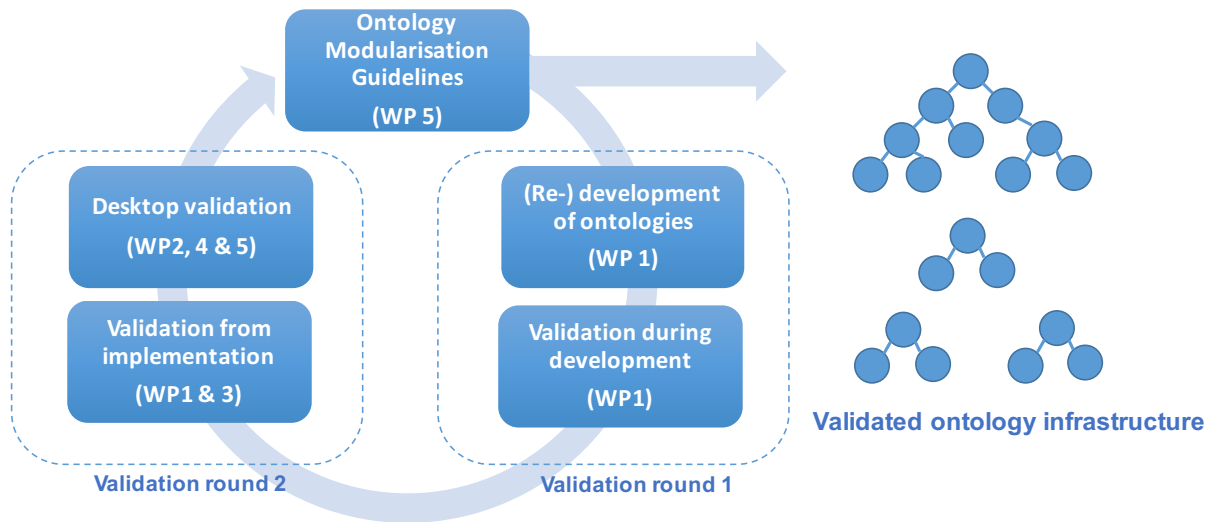


Figure 34. Validation process of BEST ontology infrastructure

5 Conclusions and future work

This report has described the ontology infrastructure developed as a part of task 1.1 in the BEST project. The ontology infrastructure consists of a monolithic ontology automatically generated from the ATM Information Reference Model (AIRM), and a set of ontology modules semi-automatically developed from the ATM exchange models Aeronautical Information Exchange Model (AIXM) and ICAO Meteorological Information Exchange Model (IWXXM). This ontology infrastructure will be used as a basis for development of software applications in other technical work packages in the project as well as for formulating guidelines on how semantic technologies can be applied in a SWIM environment. The automatic transformation from UML models to an OWL representation has resulted in ontologies that primarily maintain the semantics expressed in the source UML models. As such the resulting ontologies should be considered lightweight ontologies that do not incorporate more advanced OWL constructs such as complex classes or more advanced property restrictions. During the implementation of the ontologies in other work packages, we expect that the ontologies will be semantically enriched to facilitate expression of more complex knowledge. The validation of the ontologies will be performed iteratively, which is challenging due to their size. During the development reported in this deliverable, the ontologies have been validated using consistency checking with a reasoner, by manually comparing the resulting ontologies against their baseline models in UML as well as XML Schemas and instances for the ontologies produced from the exchange models (AIXM and IWXXM). Another round of validation will be performed when the ontologies are applied in the project's application development and when guidelines for using semantic technologies in a SWIM environment are produced. The final set of validated ontologies will be delivered along with guidelines for modularisation close to the end of the project when deliverable D5.2 (Ontology Modularisation Guidelines for SWIM) is finalised.

6 References

- [1] M. Ehrig, S. Staab, and Y. Sure, “Bootstrapping ontology alignment methods with APFEL,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3729 LNCS, pp. 186–200, 2005.
- [2] P. Hitzler, M. Krotzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, “OWL 2 Web Ontology Language Primer (Second Edition),” 2012. [Online]. Available: <https://www.w3.org/TR/owl2-primer/>. [Accessed: 12-May-2017].
- [3] I. Kovacic, D. Steiner, C. Schuetz, B. Neumayr, F. Burgstaller, M. Schrefl, and S. Wilson, “Ontology-based Data Description and Discovery in a SWIM Environment,” in *Proceedings of the Integrated Communications Navigation and Surveillance Conference*, 2017.
- [4] M. A. Musen, “Protégé Ontology Editor,” 2015. [Online]. Available: <http://protege.stanford.edu/>.
- [5] C. Schuetz, B. Neumayr, and M. Schrefl, “D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base,” 2017.
- [6] M. Perry and J. Herring, “GeoSPARQL - A Geographic Query Language for RDF Data,” 2012. [Online]. Available: <http://www.opengeospatial.org/standards/geosparql>. [Accessed: 28-May-2017].
- [7] W3C, “RDF Schema 1.1 – W3C Recommendation 25 February 2014,” 2014. [Online]. Available: <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>. [Accessed: 28-May-2017].
- [8] W3C, “SPARQL 1.1 Query Language,” 2013. .
- [9] M. Kifer and G. Lausen, “F-logic: a higher-order language for reasoning about objects, inheritance, and scheme,” *ACM SIGMOD Rec.*, vol. 18, no. 2, pp. 134–146, 1989.
- [10] S. Ceri, G. Gottlob, and L. Tanca, “What you always wanted to know about Datalog (and never dared to ask),” *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 1, pp. 146–166, 1989.
- [11] S. B. Abbes, T. Meilender, and M. D’Aquin, “Characterizing modular ontologies,” in *Conference on Formal Ontologies in Information Systems-FOIS*, 2012.
- [12] H. Stuckenschmidt and A. Schlicht, “Structure-Based Partitioning of Large Ontologies,” in *Modular ontologies*, Springer, 2009, pp. 187–210.
- [13] M. D’Aquin, “Modularizing Ontologies,” in *Ontology Engineering in a Networked World*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 213–233.
- [14] S. Wilson, R. Suzic, and S. Van der Stricht, “The SESAR ATM information reference model within the new ATM system,” in *2014 Integrated Communications, Navigation and Surveillance Conference (ICNS) Conference Proceedings*, 2014.
- [15] S. Wilson, R. Suzic, J. Pinto, S. Keller, and G. Marazzo, “SESAR AIRM Compliance Framework,” Brussels, 2015.
- [16] EUROCONTROL and FAA, “AIXM UML to XML Schema Mapping,” 2010.
- [17] EUROCONTROL, “EUROCONTROL Specifications for Aeronautical Information Exchange,” Brussels, 2012.
- [18] EUROCONTROL, “AIXM 5 Temporality Model,” Brussels, 2010.
- [19] www.fixm.aero, “FIXM Model Development Guidelines,” 2014. [Online]. Available:

- https://www.fixm.aero/documents/FIXM_Development_Guidelines_v2.pdf. [Accessed: 16-May-2017].
- [20] Object Management Group, “XML Metadata Interchange v2.5.1,” 2015.
- [21] Sparx Systems, “Enterprise Architect (Software),” 2017. [Online]. Available: <https://www.sparxsystems.eu/enterprise-architect/new-edition/>. [Accessed: 22-Feb-2017].
- [22] J. Clark, “XSL Transformations (XSLT) Version 1.0,” 1999.
- [23] Object Management Group, “Ontology Definition Metamodel (ODM) v1.1,” Needham, OSA, 2014.
- [24] U. Sattler, T. Schneider, and M. Zakharyashev, “Which Kind of Module Should I Extract?” *Descr. Logics*, 2009.
- [25] E. Jimenez-Ruiz, B. C. Grau, U. Sattler, T. Schneider, and R. Berlanga-Llavori, “Safe and Economic Re-Use of Ontologies: A Logic-Based Methodology and Tool Support,” in *ESWC 2008*, 2008.
- [26] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, “Modular Reuse of Ontologies: Theory and Practice,” *J. Artif. Intell. Res.*, pp. 272–318, 2008.

APPENDIX A: OMG Mapping Guidelines

UML elements	Package	OWL elements	Comment
class, property ownedAttribute, type ^a	7.3.7 Classes 7.3.8 Classifiers 7.3.32 Multiplicities	class	
instance	7.3.22 Instances	individual	OWL individual independent of class
ownedAttribute, binary association	7.3.7 Classes	property	OWL property can be global
subclass, generalization	7.3.7 Classes 7.3.8 Classifiers	subclass subproperty	
N-ary association, association class	7.3.7 Classes 7.3.4 Association Classes	class, property	
enumeration	7.3.11 Datatypes	oneOf	
disjoint, cover	7.3.21 Generalization sets	disjointWith, unionOf	
multiplicity	7.3.32 Multiplicities	minCardinality maxCardinality	OWL cardinality declared only for range
package	7.3.37 Packages	ontology	
dependency	7.3.12 Dependencies	reserved name RDF:property	

APPENDIX B: Post-processing of XMI

Issues with generated XMI

- The SAXON parser throws an error since there are two data types declared for “PackagedElement.PackagedElement.OwnedAttribute.upperValue. This types are ‘uml:LiteralInteger’ and ‘uml:LiteralUnlimitedNatural’
 - Resolved by removing ‘uml:LiteralInteger’

Un-needed elements

- Remove the uml:Model branch of the XMI, since this is duplicates of the xmi:Extension
- Remove the <diagrams> elements since they do not contain any information relevant for the transformation
- Remove the top-level UML packages (e.g. “AIXM_v.5.1.1”) as we do not want that as a part of the OWL.





Whitespace removal

- Typically for code values there is an extra whitespace in the end of the code value name. This is not accepted by the Protégé OWL editor.
- Resolved by removing all whitespace in names using the search-replace function in Oxygen XML Editor.

Data type conversion

- The HermiT reasoner, which is used in the ontology module extraction does not support xsd:duration as it is not a part of the OWL 2 datatype map. All instances of xsd:duration are converted to xsd:string for the time being.
- The HermiT reasoner, which is used in the ontology module extraction does not support xsd:date as it is not a part of the OWL 2 datatype map. All instances of xsd:date are converted to xsd:string for the time being.

The BEST consortium:

SINTEF	
Frequentis AG	
Johannes Kepler Universität (JKU) Linz	
SLOT Consulting	
EUROCONTROL	