Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base

Deliverable 2.1

BEST	
Grant:	**699298**
Call:	**H2020-SESAR-2015-1**
Topic:	**Sesar-03-2015** **Information Management in ATM**
Consortium coordinator:	**SINTEF**
Dissemination Level:	**PU**
Edition date:	**[09 March 2018]**
Edition:	**[01.02.01]**

Founding Members
EUROPEAN UNION EUROCONTROL
best ACHIEVING THE BENEFITS OF SWIM BY MAKING SMART USE OF SEMANTIC TECHNOLOGIES
SESAR JOINT UNDERTAKING

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Christoph Schuetz (LINZ)	Project member	
Bernd Neumayr (LINZ)	Project member	
Michael Schrefl (LINZ)	Project member	

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Audun Vennesland (SINTEF)	Project member	30.05.2017
Eduard Gringinger (FRQ)	Project member	26.01.2018
Audun Vennesland (SINTEF)	Project member	26.02.2018

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	All partners	01.06.2017
Revision approved by consortium, in accordance with procedures defined in Project Handbook	All partners	09.03.2018

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date

Document History

Edition	Date	Status	Author	Justification
00.00.01	21.10.2016	Document created	Christoph Schuetz	
00.00.02	21.10.2016	Planned Content and Structure (PCOS)	Christoph Schuetz	Completed PCOS for internal review
00.00.03	22.12.2016	PCOS	Christoph Schuetz	Revision of PCOS based on ongoing research
00.01.00	13.02.2017	Intermediary	Christoph Schuetz	Finished background and overview for intermediary review.

Edition	Date	Status	Author	Justification
00.02.00	15.05.2017	Intermediary	Christoph Schuetz	Final structure, rough content
00.03.00	25.05.2017	Intermediary	Christoph Schuetz	Draft version, most content included
00.04.00	29.05.2017	Final draft	Christoph Schuetz	Draft version for final internal review
01.00.00	31.05.2017	Final	Christoph Schuetz	
01.00.01	01.06.2017	Final revised	Christoph Schuetz	Minor changes for public release
01.01.00	10.01.2018	Final revised	Christoph Schuetz	Clarification of relationship to SWIM and addressing comments from SJU review
01.02.00	12.02.2018	Final revised	Christoph Schuetz	Taking into account comments by Eduard Gringinger
01.02.01	26.02.2018	Final revised	Christoph Schuetz	Minor revision, ready for submission to SJU
01.02.01	09.03.2018	Released	Joe Gorman	No changes in content; updated dates and approval status.

BEST

Achieving the BEnefits 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.

Abstract/Executive Summary

The upcoming System Wide Information Management (SWIM) in air traffic management mandates access to required information via information services in order to ensure common situational awareness among stakeholders. Developing value-added information services and applications in SWIM will encompass finding, selecting, filtering and composition of data/information from different sources (the 'data logic'). In this respect, semantic containers are a means to encapsulate the data logic and clearly separate it from business and presentation logic; the separation of data logic from business and presentation logic is a commonly accepted principle in software engineering. The provisioning of semantic containers for a specific purpose encompasses the discovery of existing source containers and often further value-adding processing steps such as filtering and annotating. Common semantic web technologies are well-suited to implementing the semantic container approach; a mix of semantic web technologies will have to be employed.

Table of Contents

1 Introduction: About this document[1]

1.1 Purpose

The Grant Agreement describes the content of this deliverable as follows:

This deliverable comprises

- *a method for the semantic description of data products building on ontologies from WP1*
- *techniques for discovering most relevant data products for a given information need.*

In preparing the detailed Project Management Plan at project initiation, this was extended to:

Technical report on techniques for the ontology-based description of data products and information needs and for the discovery of data products that fulfil a given information need.

This deliverable comprises

- *a method for the semantic description of data products building on ontologies from WP1*
- *techniques for discovering most relevant data products for a given information need.*

In this document, we propose the semantic container approach for handling aeronautical data. We investigate fitness of semantic web technologies for semantic container management and discovery. In particular, we propose a *faceted* ontology-based description and discovery of semantic containers. We present experimental results of using common semantic web technologies and the reference ontologies developed in Deliverable 1.1 for realizing the semantic container approach.

A work-in-progress version of the research presented in this deliverable was published as a paper (Kovacic et al. 2017) in the Proceedings of the Integrated Communications Navigation and Surveillance Conference 2017. Furthermore, this deliverable, along with Deliverable 3.1, served as the fundamental for a paper (Neumayr et al. 2017) in the Proceedings of the Digital Avionics Systems Conference 2017.

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

Deliverable	Relationship
D 1.1 Experimental ontology modules formalising concept definition of ATM data	The ontology modules developed in D 1.1 can serve as the fundamental for the faceted ontology-based description of semantic containers. We employ the ontologies developed in D 1.1 for our experiments in order to study feasibility.
D 2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment	In D 2.2 we will extend the semantic container approach with mechanisms for handling distribution of containers across different nodes, adding provenance information to the administrative metadata, distinguishing between logical and physical containers for distributed allocation.
D 3.1 Prototype Use Case Scenarios	The scenarios described in D 3.1 provide the scope for the semantic container approach.
D 3.2 Prototype SWIM-enabled applications	The prototype applications in D 3.2 will demonstrate practicality of the semantic container approach in a SWIM setting.
D 4.4 Tutorial for Software Developers	The tutorial will describe how software developers can write SWIM applications using semantic containers for data management and discovery.
D 5.1 Scalability Guidelines for Semantic SWIM-based Applications	While we conduct experiments concerning principal feasibility, D 5.1 will formally investigate scalability aspects of the semantic container approach.
D 5.2 Ontology Modularisation Guidelines for SWIM	The guidelines will describe how to develop ontology modules for the semantic container approach.

1.4 Acronyms and abbreviations

Acronym/Abbreviation	Explanation
ADQ	Aeronautical Data Quality
ANSP	Air Navigation Service Provider
AIRM	ATM Information Reference Model
AIXM	Aeronautical Information Exchange Model
ATM	Air Traffic Management
DNOTAM	Digital NOTAM
EFB	Electronic Flight Bag
F-Logic	Frame Logic
FIXM	Flight Information Exchange Model
IWXXM	ICAO Meteorological Information Exchange Model

Acronym/Abbreviation	Explanation
METAR	Meteorological Aerodrome Report
NOTAM	Notice To Airmen
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
SESAR	Single European Sky ATM Research
SI	Système international d'unités
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TAF	Terminal Aerodrome Forecast
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WSDOM	Web Service Description Ontological Model
XML	Extensible Markup Language

2 Background

In this section we give the background to the remainder of the deliverable. In particular, we explain the kinds of aeronautical datasets and the aeronautical information service economy that produces and transforms these datasets. We classify the idea of semantic descriptions of data containers in the broader context of metadata. We also briefly present the fundamentals of semantic web technologies as relevant for the understanding of this deliverable. Finally, we relate the deliverable with existing work on semantic web services.

2.1 Data, metadata, and value-added data in SWIM

SWIM offers an "information sharing" approach to ATM information management and its adoption offers advantages for better situational awareness and information management. SWIM means in principle that all aviation related information will be available for those who need it from the source that is best placed to provide it. This represents a real shift from today's bilateral aviation ICT environment to a real network based approach.

Sets of data in the upcoming SWIM originate from different authoritative sources (such as Eurocontrol) and are combined and enriched by different data providers (such as GroupEAD). They are further filtered, combined and enriched specifically by organizations (Airlines, ANSPs, Manufacturers) and provided to end-users for the accomplishment of specific tasks (such as the data in an EFB for a pilot conducting a specific flight).

Interoperability at the level of plain *data* in SWIM is facilitated by standardized information exchange models like AIXM (Aeronautical Information Exchange Model), FIXM (Flight Information Exchange Model), and IWXXM (ICAO Weather Information Exchange Model). These models are specified in the Unified Modeling Language (UML) and come with a standardized serialization in the Extensible Markup Language (XML). These exchange models provide the *schema metadata* for data sets in SWIM.

Data sets in SWIM will not only consist of collections of plain data from AIXM, FIXM, and iWXXM, but may be enriched with *value-added data*, such as annotations and classifications of primary data, such as the importance of a METAR for a particular flight. Another important kind of value-add is alignment of data at the instance level, for example two data items may adhere to the same schema from AIXM but use different measure units which hinders interoperability. A transformation to common measure units can overcome this problem and adds value to the data.

The importance of metadata in ATM, in addition to schema metadata, has been recognized. We refer to these kinds of metadata as *administrative metadata*. Several attempts exist to define metadata for the aeronautical domain. The Aeronautical Data Quality (ADQ) Implementing Rule issued by the European Commission identifies the need for a minimum set of metadata (Eurocontrol 2010). Metadata is identified as a driver for interoperability since it allows to find data and to make decisions based on the associated metadata which, e.g., can indicate the quality or relevance of the data by describing temporal and geographical facets (Porosnicu 2013). Considering the ISO 19100 standards for geographical information and geomatics, work has been ongoing to define and standardize metadata. This effort resulted in the development of the ADQ Metadata profile based on guidance and requirements from the Open Geospatial Consortium (OGC) (Wilson 2011).

Information reference models and ontologies (see below) are typically used as *intensional metadata*, specifying the correct interpretation of data sets. In order to support the automation of the discovery and the combination of datasets to fulfil a given information need one additionally needs *extensional metadata* which specifies the set of data items contained in some data container in order to answer questions like: "Does data set X contain all the relevant weather messages for today's flight from Linz to Trondheim?". The distinction between intensional and extensional metadata is discussed by Catarci and Lenzerini (1993)

The preparation and maintenance of data sets is often expensive and sometimes cannot be fully automated. Often human experts or computationally-expensive expert systems are involved. It is thus important to foster reuse of existing data sets. A dataset's metadata gives guarantees and hints about the data quality, especially how fresh the data are. There will be more and more datasets in SWIM, system-wide or within a project. Automatic discovery of data sets based on their metadata avoid creation from scratch if others have already prepared applicable datasets.

In BEST, we introduce semantic containers as a principled approach for organising and reusing sets of plain ATM data (AIXM, FIXM, iWXXM) and value-added data based on extensive use of all kinds of metadata attached to these semantic containers.

2.2 ATM Information Reference Model (AIRM)

AIRM (ATM Information Reference Model) is a standardised information model for ATM designed to ensure that the information communicated in SWIM is clearly and uniquely defined and well understood. The AIRM comes as a package containing explanatory material and Unified Modelling Language (UML) models. The AIRM UML models are structured to satisfy the needs of several different audiences and its use as a common reference. The models promote semantic interoperability between operational experts, systems and services within the European ATM Network.

Semantic interoperability ensures that the precise meaning of exchanged information is preserved and understood by all parties. The AIRM is recognised in the ICAO Global Air Navigation Plan and in the European Union's Pilot Common Project. The AIRM will become a Eurocontrol specification.

The BEST project employs these models and related specifications as a basis for the development of ontologies which, in turn, are the basis for the metadata describing semantic containers.

2.3 Ontologies and semantic reasoning

Conceptual domain models, such as the AIRM, are typically developed and expressed using the Unified Modeling Language (UML). UML is well-suited to the development of small- to medium-sized domain models, yet practically bound to proprietary tools with only limited support for consistency checking and querying of large models. Furthermore, UML models are typically only used in the requirements engineering and development phase, without proper support for using and changing the models after system deployment and at runtime.

Ontologies are formal conceptual domain models with precise and unambiguous semantics, fixing the meaning of types, properties, and interrelationships of the entities that exist for a particular domain of discourse. By using an ontology, the various actors in a decentralized information system commit to the shared understanding of concepts as defined in the ontology, thus avoiding misunderstandings in their communications. Expressing a conceptual domain model or information reference

model (like the AIRM) in terms of an ontology language, such as the web ontology language OWL, with formal semantics avoids the semantic ambiguities of a modelling language like the UML and yields a lightweight domain ontology, often also referred to as vocabulary.

Ontologies often additionally come with detailed axioms on the use of classes and properties and their correct interpretation. Arguably the two most important kinds of axioms in an ontology are those attached to a class as "necessary condition" (every instance of the class fulfils the condition) or as a "necessary and sufficient condition" (every instance of the class fulfils the condition, and every object that fulfils the condition is an instance of the class). The latter is also referred to as a concept definition.

Based on the formal semantics of the ontology language, semantic reasoners are used for

- **Satisfiability/consistency checking**: automatically check the internal consistency of an ontology, i.e. that the statements expressed in the ontology do not violate each other, or the satisfiability of a single class, i.e. that a class may have instances according to its definition.
- **Subsumption reasoning**: derive the subsumption hierarchy of classes, also known as specialization/generalization/inheritance hierarchy. Subsumption reasoning is about deciding whether two classes are in a specialization (or subset) relationship.
- **Membership reasoning** and **query answering**: given a class and an individual object, find out if the individual is member of the class. Derive additional statement (attributes and relationships) of individual objects based on the ontology's axioms.

Take for example a very simple ontology where two classes 'airplane' and 'helicopter' are defined as disjoint. The class 'aircraft' is defined (by a necessary and sufficient condition) as union of 'airplane' and 'helicopter' and, meaning that every airplane and every helicopter is also an aircraft and every aircraft is either an airplane or a helicopter. Further class 'aircraft' has as necessary condition 'can fly'. The semantic reasoner derives (*subsumption reasoning*) that airplane as well as helicopter is subsumed by aircraft and by things that 'can fly'. For every instance of 'airplane' the reasoner can derive (*membership reasoning*) that it is also an instance of 'aircraft' and thus is an instance of the things that 'can fly'. Based on this ontology, a class 'grounded airplane' defined as airplane that cannot fly would be inconsistent (*class satisfiability checking*). Declaring an individual object to be instance of 'grounded airplane' would make the ontology inconsistent (*ontology satisfiability checking*). Since airplanes can be grounded temporarily, the definition would have to be adapted in order to obtain a consistent ontology. For such a very simple ontology, these reasoning tasks are straightforward, with more complex ontologies with many classes and relationships between classes the reasoning tasks become very complex.

Consistency-checking and subsumption reasoning are two reasoning services that have proven indispensable in the development and maintenance of very large ontologies. Membership reasoning is especially relevant for using semantic reasoners at the instance level, to derive additional statements used for answering queries.

In BEST, the AIRM is transformed from its UML representation to a lightweight ontology (D 1.1) which we use, in this deliverable, to specify contents of semantic containers as well as information needs by necessary and sufficient conditions. Subsumption reasoning then serves to derive a hierarchy of semantic containers and to find semantic containers that subsume a given information need.

2.4 Semantic web technologies

The Semantic Web extends the World Wide Web with machine-readable descriptions of resources. Quite literally, the "Semantic Web is a machine-readable Web" (Domingue et al. 2011), where resources have precisely defined meaning. Conversely, Semantic Web technologies are those technologies that serve for the definition and interpretation of resource semantics. Semantic web technologies comprise languages for data interchange (RDF), querying (SPARQL), and lightweight as well as rich ontologies (RDFS and OWL). There is a couple of open source and production-ready implementations of these technologies as database and query engines and automated reasoners.

XML. The Extensible Markup Language (XML) is often used as exchange format for semantic web data and ontologies (with OWL and RDF data serialized in XML). XML schema datatypes are used by other semantic web technologies. In many cases, including BEST, instance data is encoded in XML and only metadata is encoded in RDF and OWL.

RDF(S) and SPARQL. The Resource Description Framework (RDF) defines a versatile format for knowledge representation that consists of triples of subject, predicate, and object (W3C 2014a). These triples express statements over resources which are identified by International Resource Identifiers (IRIs, think a generalized form of the ubiquitous URLs). The resources represent objects and their properties. Consider, for example, the following RDF triple from DBpedia , which states that Airbus A300 has as successor Airbus A310: `dbr:Airbus_A300 dbo:successor dbr:Airbus_A310`. The RDF data model in its basic form is schemaless, i.e., it does not distinguish between classes and objects and does not impose any restrictions on properties. RDF Schema (RDFS) allows for the definition of classes and relationships which in turn allows for the definition of simple ontologies (W3C 2014b). SPARQL, in turn, is the query language for RDF data. We examine RDF(S) and SPARQL more closely in Section 5.1.

OWL and SWRL. The Web Ontology Language (OWL) is arguably the most popular ontology language and standardized by the World Wide Web Consortium (W3C). OWL builds in parts on RDF(S); OWL ontologies are serialized using RDF. Various variants of OWL, called profiles, exist which vary in the degree of expressivity. Most OWL profiles trade expressivity for reduced complexity and decidability. The OWL profiles are based on different description logics. The Semantic Web Rule Language (SWRL) extends OWL with support for rule-based reasoning (Horrocks et al. 2004). We examine OWL and SWRL more closely in Section 5.2.

Rules (F-Logic and RIF). The Rule Interchange Format (RIF) is also part of the set of semantic web standards (from the W3C) and based on rule-based languages such as Datalog and production rules. With regard to knowledge structuring it is heavily influenced by F-Logic, a formalism used in the SemNOTAM project (Steiner et al. 2016).

Semantic web services. A semantic web service is a web service with a description of the allowed inputs and the results of a service call as metadata (see McIlraith et al. 2001). The semantic description of web services is machine-readable which, in turn, facilitates automatic discovery as well as composition of web services. Among the types of semantics covered by web service descriptions are data semantics and functional semantics (see Domingue et al. 2011, p. 980), which are similar to the metadata we consider for semantic containers.

With semantic web technologies, there is no “one size fits all”. Different technologies come with very different characteristics regarding expressivity, computational complexity and performance. One has to make very careful technology choices for each task that should be supported by semantic technologies. In BEST we analyse these technology choices and propose a technology mix including XML (or JSON) for plain and value-added ATM data, RDF for describing semantic containers including administrative metadata, OWL for the representation of AIRM and the definition of the contents of semantic containers and information needs, rules[2], and ideas from semantic web services. Semantic web services have a service-centric view. The semantic container approach complements the service-centric SWIM with a data-centric approach facilitating reuse of data sets.

2.5 Related Work

The presented approach bears similarities to semantic web services and their discovery. Semantic annotation of web services renders the service descriptions machine-readable which, in turn, facilitates automatic discovery as well as composition of web services (see McIlraith et al. 2001). Among the types of semantics covered by web service descriptions are data semantics and functional semantics (see Domingue et al. 2011, p. 980). We adopt a data-centric view towards web services: We describe the data products that are the output of web services, thus covering the data semantics of aeronautical information services. Unlike work on semantic web services, which is predominantly concerned with web service discovery, the presented approach for data product description explicitly considers distribution and maintenance of the data products after provisioning through the corresponding web services.

Ongoing research (Balaban 2016) aims at extending the WSDOM ontology, which allows for the semantic description of web service interfaces in the aeronautical domain, with support for geospatial concepts. To this end, GeoSPARQL serves as representation and query language for web service discovery. The WSDOM ontology as well as the proposed framework for handling geospatial information are orthogonal to the semantic container approach as presented in this deliverable. We do not focus on the web services as such but on the management and discovery of data sets. To this end, we introduce the notion of semantic containers and employ ontologies for the semantic description of container contents.

There are several approaches to applying ontologies for aviation data management; see Keller (2016) for a survey of existing projects. Keller et al. (2016) integrate different kinds of ATM data from different sources into an RDF triple store, first constructing an RDF-based ATM ontology and then transforming all the ATM data into RDF triples according to this ontology. In the described research, only aviation data for one particular day was transformed, leaving open the question how this approach scales with larger amounts of data. In contrast, our approach applies ontologies and semantic technologies for metadata only, without the necessity to integrate the large amounts of instance data residing in these semantic containers using semantic technologies, i.e., converting ATM data into RDF triples. This does not mean, however, that semantic containers cannot be materialized in their native (or any other) format (see Chapter 3 for more information).

[2] We refer to the SemNOTAM project for reasoning about instance data.

3 Semantic containers and SWIM

System Wide Information Management (SWIM) describes a service-oriented architecture (SOA) for ATM applications. As such, SWIM defines a set of information services, e.g., aeronautical information feature service or METAR service. Each service definition in SWIM also contains a basic definition of the information that a service works with and returns as output, e.g., a METAR service returns a set of METARs. A specific service instance, in turn, is a concrete implementation of a SWIM service. For example, the DWD METAR service is an instance of METAR service. Preferably, there should also exist a machine-readable description of the specific information that a service instance works on and returns as result. For example, the DWD METAR service provides only METARs relevant for Germany and surrounding regions and each service call returns a subset of these METARs depending on the specific arguments supplied by the end user.

A semantic container consists of a membership condition, administrative metadata, and a set of data items – the container's contents (see Chapter 4 for detailed information). The membership condition describes the set of data items: Each data item in that set satisfies the membership condition. The administrative metadata provide additional information about provenance and freshness of the contained data. We also refer to the ensemble of membership condition and administrative metadata as the *semantic description* of a container.

The semantic description may serve to describe individual SWIM service instances or, more specifically, the information that a SWIM service instance works on. Semantic descriptions also serve to describe the contents that a service instance invocation returns as result. In addition, SWIM services may receive input information and return output information in the form of semantic containers, yielding *semantically enriched SWIM services*. The advantages of semantically enriched SWIM services are as follows:

- A semantically rich description of SWIM service instances will facilitate searching of information services in the SWIM registry that most closely fit an end user's information need. The SWIM registry must have a description of how the information that a SWIM service works upon looks like should the SWIM registry aim to provide rich search capabilities. Such descriptions must be machine-readable. Hence, using semantic technologies is a natural fit. In this deliverable, we investigate the available options for semantic technologies and propose a faceted approach to semantic descriptions.
- Using semantic containers to feed input data into a service and receive output data from the service facilitates exchange, reuse, and redundant storage of ATM information. Semantic descriptions could be stored in a knowledge base, the actual data items could be stored in a database. Such a semantic container store may be realized as a part of the SWIM registry or a SWIM service itself (see Deliverable 3.1 for further information). Note that the semantic container approach relies on SWIM as far as security aspects of the semantic container management system are concerned; an investigation of security aspects is outside of the scope of this deliverable.

Figure 3 illustrates how semantic containers fit into the notion of SWIM information services. The information services interact with a semantic container management system that consists of a knowledge base for the semantic descriptions and a database containing the actual data; the

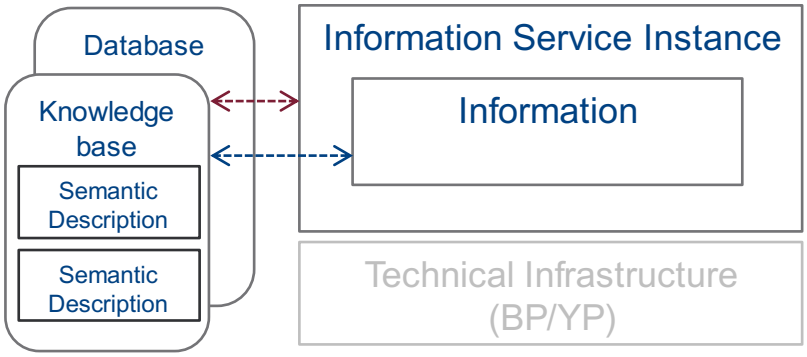


Figure 1. Semantically enriched SWIM information services

semantic descriptions and the actual data constitute the semantic containers. The semantic descriptions describe the information that an information service instance works on, i.e., receives as input, and that an information service instance invocation returns as output. The information service instance may also use the semantic container management system to store and retrieve semantic containers, providing a standard way of packaging information in SWIM.

The upper part of Figure 2 provides a schematic description of the SWIM service metamodel. A service definition has an information definition for input and output. A service is an instance of a service definition and operates on a set of base information, which is an instance of the information definition. A service invocation/call then receives information as input and returns information as output. For example, NOTAMService is a service definition that receives NOTAMs as input and returns NOTAMs as output. NOTAMService-FAA and NOTAMService-GroupEAD are instances of NOTAMService that operate on different base information. NOTAMService-FAA operates on NOTAMs-NorthAmerica whereas NOTAMService-GroupEAD operates on NOTAMs-Europe+MidEast. A specific invocation/call of NOTAMService-GroupEAD may return the set of NOTAMs relevant for the route from Dubai to Vienna. These set of NOTAMs may be kept for later re-use in other service invocations, which receive that set as input for further filtering, e.g., to produce the set of NOTAMs relevant for a particular flight. In that scenario, the semantic container approach fits in as follows. First, semantic descriptions of the contents may serve to describe the base information that a service instance operates on; the semantic description allows for searching SWIM services based on the requested data. The semantic container approach is a flexible proposal to realize description of the information offered by a service. Second, semantic descriptions allow for the description of the contents of packages of information, which can be stored and retrieved for later re-use. Existing semantic containers can be found based on the semantic description of the contents and do not have to be recomputed every time a service is invoked.

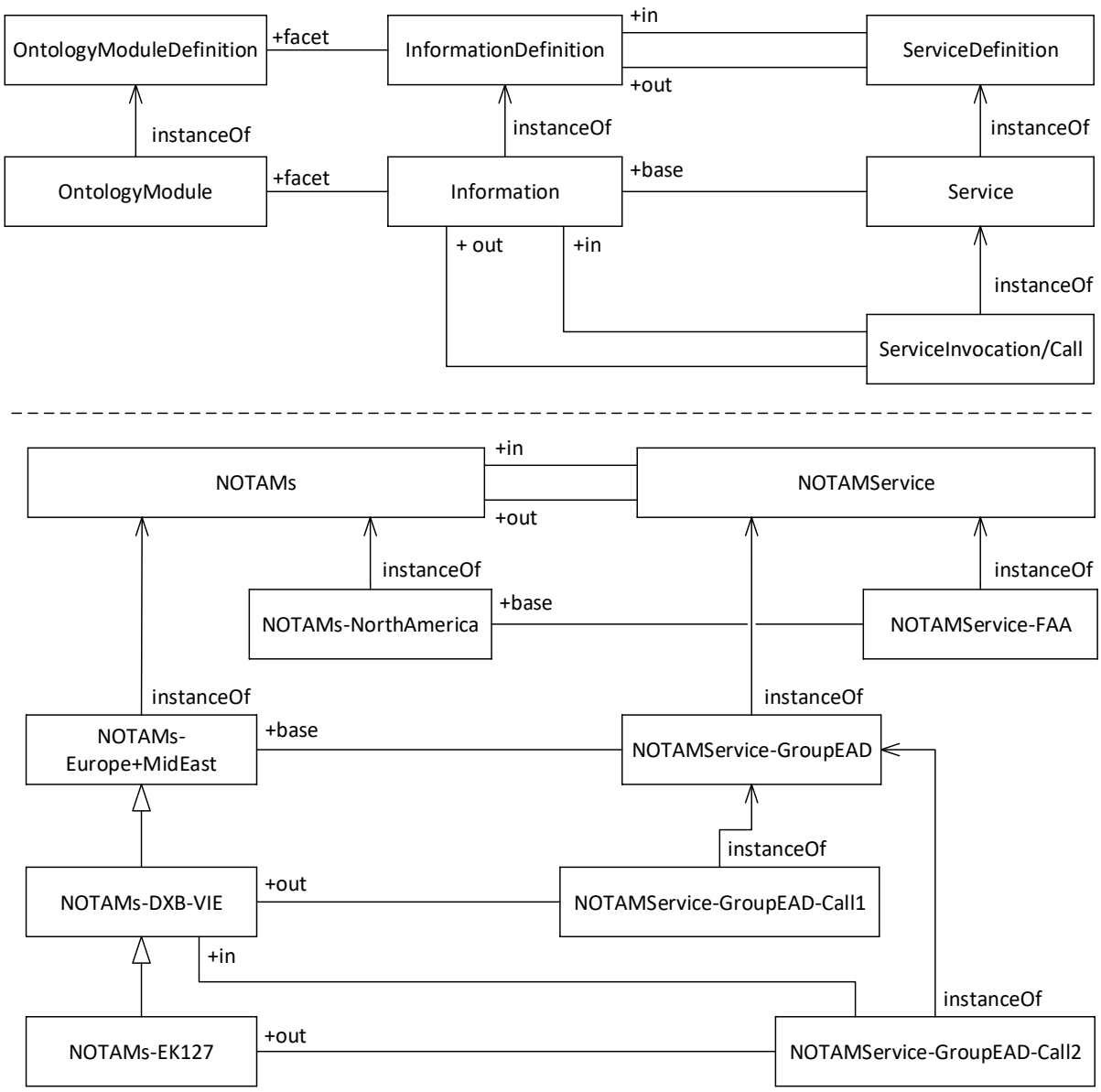


Figure 2. Semantically enriched SWIM service metamodel, SWIM service definitions, SWIM service instances, and SWIM service invocations/calls. Semantic containers help to describe the base information that a service instance works on and allows for the automatic determination of the generalization/subsumption relationships between these sets of data/information. Semantic containers also allow to describe the data/information that a service invocation/call receives as input and returns as output.

4 The semantic container approach

In this chapter, we discuss the key concepts of the semantic container approach. Section 4.1 motivates the need for ontology-based data description and discovery in SWIM. Section 4.2 defines our notion of semantic container (restricted to elementary containers). Section 4.3 gives an overview of the different kinds of metadata used to describe a semantic container. The purpose of data discovery is to find relevant data, in the semantic container approach this means matching information needs to existing semantic containers based on semantic reasoning, which is discussed in Section 4.4. Section 4.5 discusses the key ideas of combining different semantic containers into a composite semantic container. Section 4.6 introduces the basic ideas of semantic-container-based management of value-added data, like annotations, classifications, and alignments of primary data. Subsequent chapters will give details on how to realize these concepts.

4.1 Motivation

The upcoming System Wide Information Management (SWIM) within the aviation industry grants access to data/information via information services to ensure common situational awareness among stakeholders. Standardized exchange models like the Aeronautical Information Exchange Model (AIXM), the Flight Information Exchange Model (FIXM), the ICAO Weather Information Exchange Model (IWXXM), or semantic models like the ATM Information Reference Model (AIRM) already affect software architecture and software development in a positive manner.

Developing value-added data services and applications in SWIM will encompass finding, selecting, filtering and composition of data from different sources (the ‘data logic’). Without a semantic description of the information/data, the data logic will likely be hard-coded in applications and service implementations, intertwined with business and presentation logic, which hinders reuse. In this regard, SWIM can be imagined as a gigantic whiteboard where different authorities write data, making it difficult for stakeholders to focus on the data relevant for a specific purpose in the needed quality. The complexities of the data logic will likely absorb most of a developer’s attention, restraining them from developing novel applications and value-added services.

We introduce semantic containers as a means to encapsulate the data logic of SWIM services and clearly separate it from business and presentation logic. A semantic container serves as ‘magic goggles to look at the gigantic whiteboard’. A semantic container provides a SWIM application or service with all the relevant data, hiding the complexities of providing these data. Semantic containers come with ontology-based metadata that allow users, services, and applications to judge the freshness and quality of the data.

The provisioning of semantic containers for a specific purpose encompasses the discovery of existing source containers and often further value-adding processing steps such as filtering and annotating. These tasks are supported by semi-automatic matching of information need and available data containers and services. Based on a formal ontology-based specification of the information needed for an operational scenario, the semantic container system could identify the missing processing steps necessary to generate a data container that fulfils the specified information need, although the implementation of the corresponding algorithms is left to future work.

4.2 What is a semantic container?

A semantic container has content and description. The content is a set of data items of a specific type, such as NOTAM or METAR; the content can be materialized or just referenced. The description includes a membership condition and administrative metadata, such as provenance, quality, and technical metadata. A semantic container should contain all and only data items that fulfil the membership condition. Quality metadata (such as a last-update timestamp) gives indications of possible deviations of actual content from the membership condition.

Constituents of a semantic container:

- **Set of data items**. The **content** of the semantic container.
- **Membership condition**. Every data item that fulfils the condition is member of the data set.
- **Administrative metadata**. Quality, freshness, provenance, and technical metadata.

An elementary semantic container is a semantic container with all contained data items being instances of the same data item type and having the same origin. Composite semantic containers (see Section 4.5) overcome this limitation. Note, the content and the administrative metadata of a semantic container may change over time, but the membership condition remains stable.

An example semantic container is depicted in Figure 1. It contains all data items of type METAR originating from DWD (Deutscher Wetterdienst) and relevant for flight route MUC-FRA (Munich to Frankfurt) on February 23, 2017. The content is serialized in XML format and was last changed at 11am. Since then, new METARs may have been published by DWD and have not been included in the content of the semantic container.

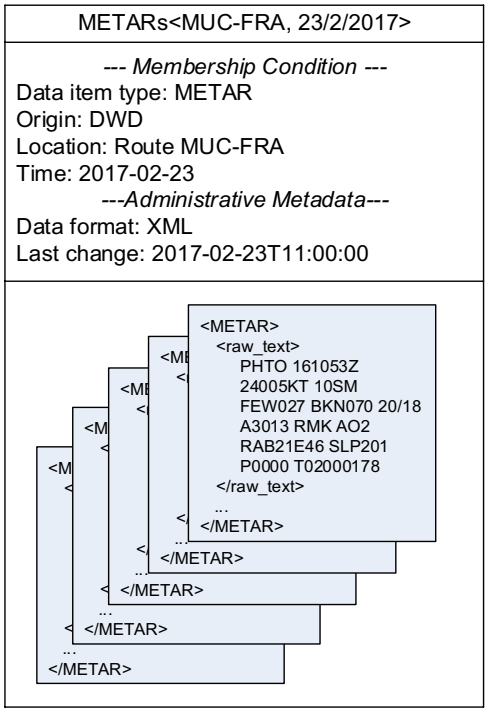


Figure 3. A semantic container

How a semantic container is populated (filled with data items) is not prescribed by the semantic container approach. Its membership condition is to be understood as a contract that the container contains all data items that fulfil the membership condition. How that is realized is in the responsibility of the provider of the semantic container (e.g., MET office or an information service provider that aggregates and refines source data, etc.) and indicated as part of the administrative metadata of the semantic container. It is possible that a container is filled by a human expert who does for example the filtering. In simple settings, a membership condition or parts of the membership condition may be translated to an SQL query which can be directly executed on a database. In many cases the membership condition will be too complex to be directly transformed to a database query, instead a rule-based system that encodes expert knowledge (with SemNOTAM as a notable example) can be used for the population of semantic containers.

4.3 Using semantic containers for data description

In this section we introduce the key concepts of the description of semantic containers focusing on elementary semantic containers of primary data items. Semantic containers of secondary data items add additional descriptive elements which will be discussed in Section 4.6. The description of composite semantic containers is given by the descriptions of its components.

Constituents of the description of an elementary semantic container

- **Data item type**
- **Membership condition** (content definition)
    - Semantic facets
    - Temporal facets
    - Spatial facets
- **Administrative metadata**
    - Technical metadata
    - Quality metadata
    - Provenance metadata

There are many different data item types in SWIM, such as METAR, TAF, and NOTAM. A primary data item type typically corresponds to a class in AIRM of stereotype IMMessage. The semantic container approach further supports secondary data item types, such as annotations of type ‘NOTAM Importance’.

We distinguish technical metadata, quality metadata, and provenance metadata. Technical metadata include the description of a semantic container’s data format. The data format itself consists of syntax and data model. Syntax may be JSON, XML, some RDF notation, etc. The data model may be AIXM, WXXM, FIXM, some RDFS/OWL ontology. Quality metadata includes timestamps describing the freshness of the data, such as last update and last check for updates of the container. Provenance metadata may include the service that was used to populate the container with data items.

Independent of a semantic container’s quality, provenance, and technical metadata, a semantic container’s content is defined by a membership condition. This content definition can be understood as a contract of what data items the semantic container contains, in the sense that every data item that fulfils the content definition is part of the semantic container.

The membership condition defines which data items go into a semantic container and, therefore, characterizes the content of a semantic container. Membership conditions may be used to populate a semantic container or reason about subsumption of semantic containers, i.e., if one semantic con-

tainer is more specific than the other, with the more general container having all the content that is in the more specific container, and possibly more.

The membership condition (e.g, 'METAR originating from DWD and relevant for flight route MUC—FRA on February 23rd, 2017') is to be understood as a defined concept which can be split up into orthogonal facets (e.g., 'relevant for route MUC-FRA'). Splitting up membership conditions into orthogonal facets supports ontology modularization. This kind of modularization allows for splitting up the reasoning tasks into smaller independent subtasks which is a key to improved scalability (see Section 6). Membership reasoning (for populating containers) can be done separately for each facet and then combined. Similarly, subsumption reasoning (for deriving a hierarchy of semantic containers) can be done separately for each facet and then combined.

The facet values (such as 'Location: Route MUC-FRA' or 'Valid Time: 2017-02-23') are to be understood as concepts (e.g., 'data item relevant for route MUC-FRA' or 'data item valid on 2017-02-23') that can be interpreted as sets of data items. These facet values/concepts are organized in subsumption hierarchies (e.g., 'Location: Route MUC-FRA' is subsumed by 'Location: Germany' meaning that every 'data item relevant for route MUC-FRA' is also a 'data item relevant for flights in Germany').

The facets of a membership condition are grouped into temporal facets, such as valid time, spatial facets, such as location, and semantic facets, an umbrella term for other facets such as the aircraft type that the data is relevant for. The facets characterize the contents of semantic containers, and are predefined in order to allow for a decentralized network of semantic containers that can be queried. Every facet refers to an ontology. For some ontologies, the subsumption hierarchy of concepts can be generated automatically. For other ontologies, an external reasoner must compute subsumption hierarchies. For example, external reasoners are used to derive subsumption hierarchies of GML shapes or of complex temporal concepts. The choice of reasoner depends on the facet, each facet may come with a specific reasoner, which may be an automatic reasoning engine or a human domain expert.

Based on facet-specific subsumption hierarchies, membership conditions of semantic containers may be organized into subsumption hierarchies by automated reasoners. These subsumption hierarchies of membership conditions yield a hierarchy of the corresponding semantic containers from more general to more specific. Consider, for example, the membership conditions of semantic containers in Figure 4, which are based on the scenario described in BEST Deliverable 3.1 of the flight from Dubai (DXB) to Vienna (VIE). The membership condition of the container with label 'NOTAMs<DXB-VIE, 2017>' subsumes the membership condition of the container with label 'NOTAMs<DXB-VIE, 23/2/2017>'. The former semantic container contains all NOTAMs relevant for the flight route DXB-VIE valid in year 2017. The latter is more specific a date description and contains NOTAMs relevant for the same route but only those with a valid time intersecting with day 23/2/2017.

Concerning the physical implementation of the semantic container approach, two options exist. First, the semantic container approach could be merely a logical concept where the membership conditions and administrative metadata serve to describe SWIM services (or, more specifically, the data/information that service works with). On the other hand, semantic containers could also be actual physical data packages used to exchange data/information, which can also be redundantly stored in a distributed network (see BEST Deliverable 2.2).

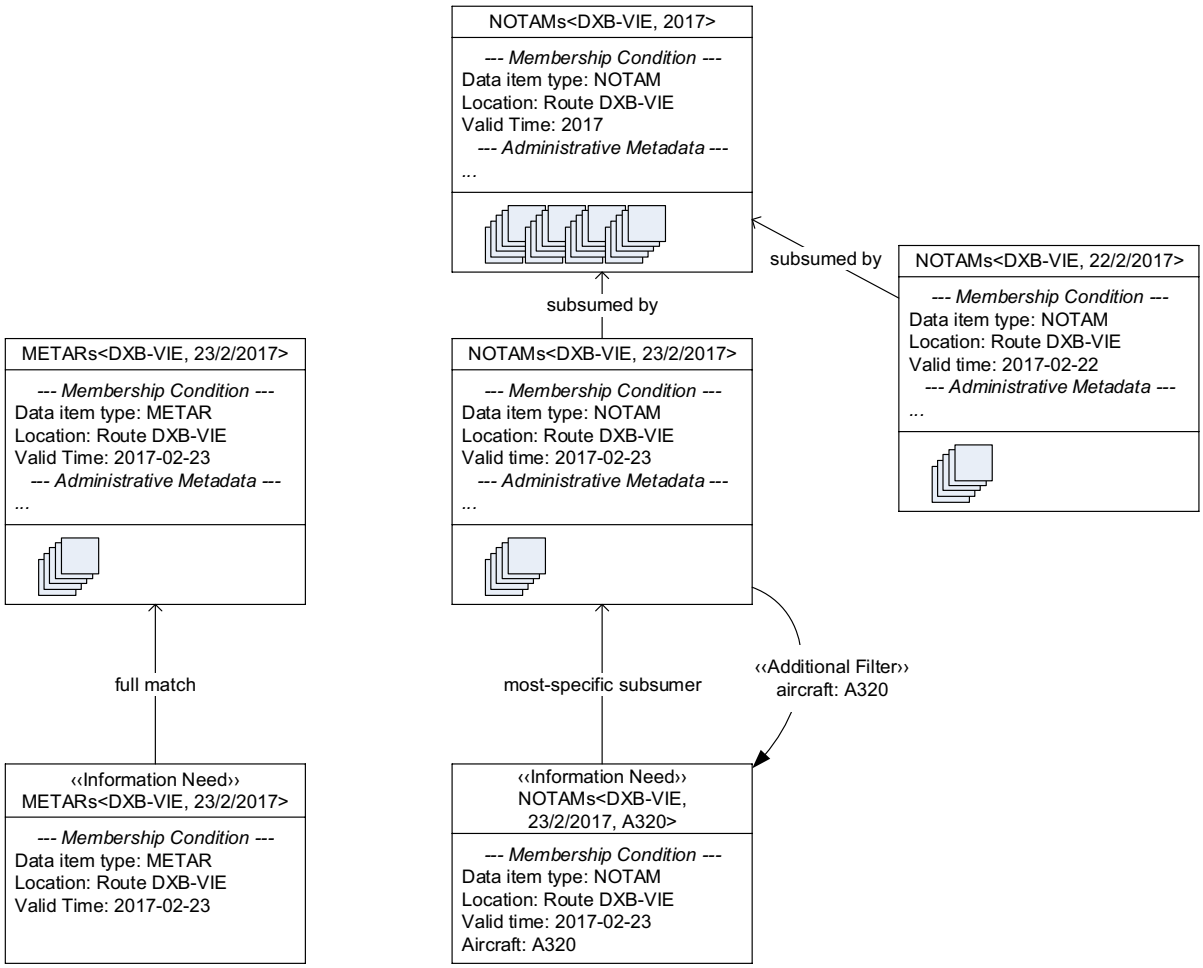


Figure 4. Subsumption reasoning over the membership conditions allows for the hierarchization of semantic containers. A semantic container satisfies an information need if the translation of the information need into a container description is a subset of said data container.

4.4 Using semantic containers for data discovery

In order to fulfil a particular task, a service, application or user has a particular information need (i.e., the data/knowledge needed in order to be able to fulfil the task). For example, a pilot preparing for a flight from Dubai to Vienna on the 23rd of February needs the METARs and NOTAMs relevant for this route and day.

Key concepts for data discovery in the semantic container approach are:

- **Information need** of a service, application or user; expressed as a membership condition.
- **Full match**. A semantic container that contains exactly the data to fulfil the information need, i.e., the semantic container's membership condition is equivalent to the information need's membership condition.
- **Most-specific subsumer.** A semantic container that contains all the data to fulfil the information need, i.e., the semantic container's membership condition subsumes the information need's membership condition.

- **Additional filters**. The missing processing steps that are necessary to transform the most-specific subsuming semantic container into a semantic container that fully matches the information need. The missing processing steps may, in principle, be deduced automatically.

In order to determine the semantic container that best fits the information need required for a particular ATM task, the existing containers' membership condition must be analyzed. To this end, the information need must first be expressed as a membership condition and a subsumption reasoner can determine the semantic container that is the best match of the information need. This may be a full match, i.e. a container with a membership condition that is equivalent to the information need, or a most-specific subsumer, this is the semantic container with a membership condition that is more general than the informantion need. In the latter case, the system should indicate which additional filters are to be applied on the most-specific subsumer to produce a semantic container that fully matches the information need. In case these additional filters cannot be directly expressed in a query or call of an existing service, it is to be understood as an input for the developer of the 'data logic' which will take care of the implementation of the filter.

For example (see Figure 4), a pilot's information need is expressed by two information needs, labelled 'METARs<DXB-VIE, 23/2/2017>' and 'NOTAMs<DXB-VIE, 23/2/2017, A320>'. For the former the reasoner finds a full match. For the latter the reasoner finds a most-specific subsumer, namely the container labelled 'NOTAMs<DXB-VIE, 23/2/2017>' and identifies 'Aircraft: A320' as additional filter necessary to produce a full match. The semantic description serves as the fundamental for the identification of missing processing steps but the actual formulation and implementation of algorithms is left to future work.

4.5 Composition of semantic containers: key concepts

Typically, a user's or service's information need is not satisfied by a single semantic container with all data items of the same type and from the same origin. Rather a set of semantic containers with data of different types and different provenance is needed. Composite semantic containers allow to express such complex information needs and their realization as semantic containers. Regarding composition, we consider the following types of semantic containers:

- **Elementary Semantic Container** ('Fragment' in the proposal). A set of data items of the same data item type (e.g., METARs) and same data model (e.g., IWXXM), data format (e.g., XML), provenance, quality, and freshness. All data items in an elementary semantic container share the same administrative metadata which can in turn be provided 'in bulk' with the container.
- **Homogeneous Composite Semantic Container** ('Semantic Data Container'). A set of data items of the same data item type (e.g., METARs), possibly with differing administrative metadata (provenance, quality, freshness). A homogeneous semantic container is a set of elementary semantic containers (fragments) of the same data item type, data model, and data format.
- **Heterogeneous Composite Semantic Container** ('Semantic Data Product'). A set of data items of different data item types. A set of homogeneous composite semantic containers.

One key requirement of the semantic container approach is to keep track of data provenance and quality/freshness metadata to allow judgements about the quality of the data. Additionally, in the spirit of database normalization, which aims to reduce redundancies in order to foster consistency, it

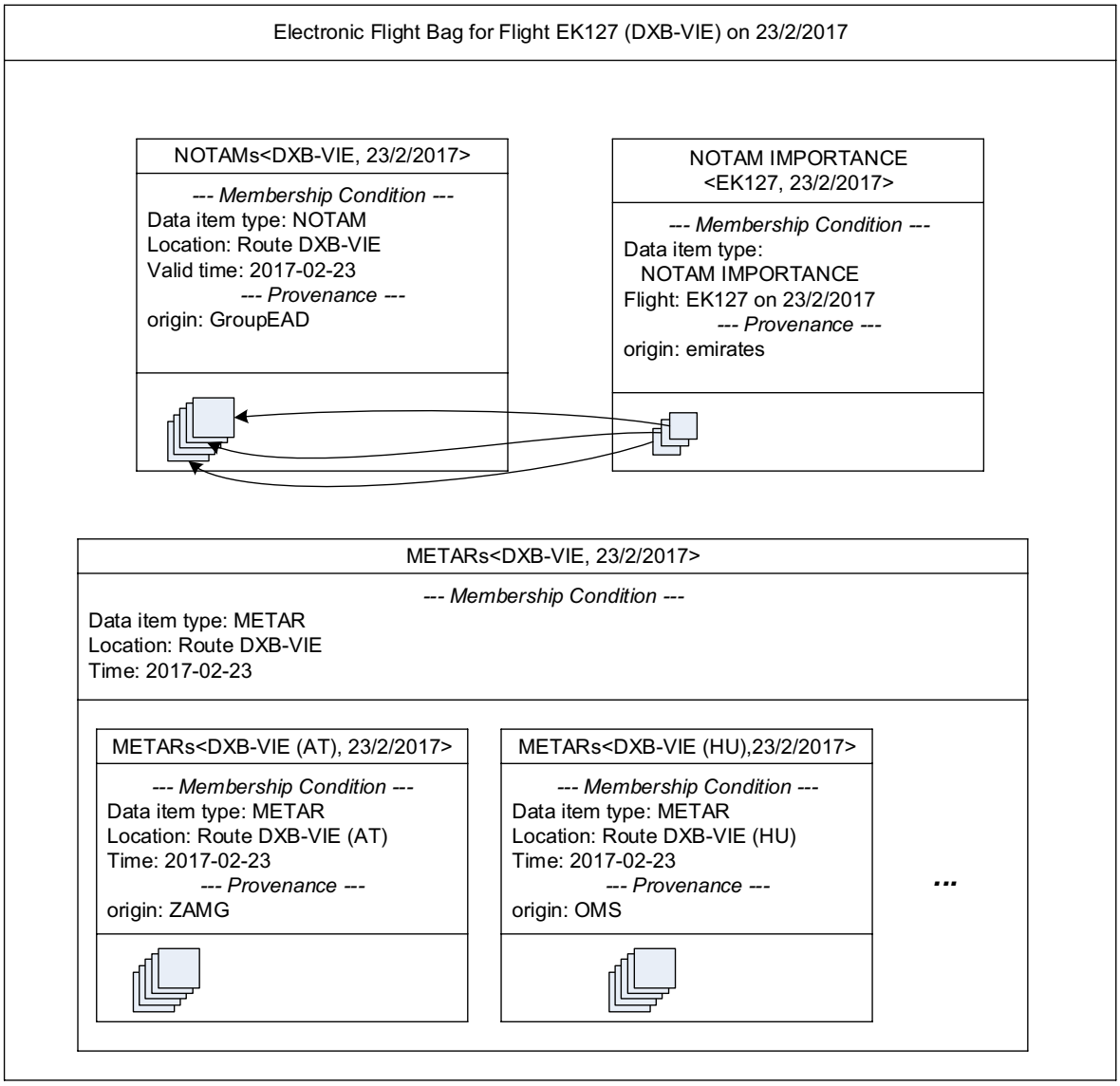


Figure 5. A composite semantic container

should be avoided to attach redundant metadata to every data item. To avoid metadata redundancy, the elementary semantic container is the finest grain where semantic metadata is attached.

Elementary containers with the same data item type but different provenance (and thus maybe different update cycles and data quality) may be combined into a homogeneous composite semantic container. Often, the components of a homogeneous composite container have a membership condition that only differs in one facet and then one constructs a simple membership condition for the composite container. A simple membership condition is one that has one value (which may be a complex class expression) per facet, with the membership condition being the conjunction of these facet values. For example, two elementary containers with membership conditions “METAR<Location: AT, Time: 23/2/2017>” and “METAR<Location: HU, Time: 23/2/2017>” can be

combined into a homogeneous semantic container with membership condition “METAR<Location: (AT or HU), Time: 23/2/2017>”.

The composition of elementary or composite semantic containers of different data item types yields a heterogeneous semantic container, such as the electronic flight bag represented in Figure 5, composed of an elementary semantic container of data item type NOTAM, an elementary semantic container of secondary data item type “NOTAM IMPORTANCE”, and a homogeneous composite semantic container of data item type “METAR”.

The membership condition of a heterogeneous composite semantic container is the disjunction of the membership conditions of its components. Regarding freshness, the oldest freshness date among the freshness values of the semantic containers becomes the composite container’s freshness value. The component semantic containers in a homogeneous composite container, as opposed to a heterogeneous composite container, have the same data format, possibly after prior conversion.

4.6 Value-added data in semantic containers

Primary data item in SWIM are entities and messages standardized by the information exchange models AIXM, FIXM, or IWXXM and with AIRM as a common reference model. One of the assumptions of BEST is that information services may also provide secondary data which enrich the primary data. In BEST, such ‘value-added’ data are the result of semantic transformation (e.g., transforming all measurements to SI units), annotation, or classification. A secondary data item is attached to another data item (which it enriches), typically a primary data item.

Kinds of data items with regard to added value:

- **Primary data items**. A data item (entity, message) from AIXM, FIXM, IWXXM.
- **Secondary data items** (Value-added data).
    - **Direct secondary data items.** A secondary data item that is associated with a single primary data item.
    - **Associative secondary data items**. A secondary data item that is associated with a primary data item but additionally with regard to one or more other data items.

Direct secondary data items are attached to a single primary data item. Examples of direct secondary data items are a) the event scenario classification of a NOTAM, and b) the transformation of a NOTAM to standard SI units. The membership condition of a semantic container of direct secondary data items is similar to a semantic container of primary data items with the main difference to have a secondary data item type. Consider, for example, the semantic container assigning NOTAM importance as illustrated in Figure 5. Each NOTAM importance item is a direct secondary data item that refers to a specific NOTAM in another semantic container, as indicated by the arrows from the NOTAM importance container to the NOTAM container.

Associative secondary data items are attached also to a single data item but taking into account one or more other data items serving as context of the annotation or classification. An example of an associative secondary data item is the importance of a NOTAM with regard to a specific flight plan segment.

4.7 Derivation chains of activities and semantic containers

ATM data in the upcoming SWIM originates from different authoritative sources (such as Eurocontrol) and is combined by and enriched by different data providers (such as GroupEAD). The data are further filtered, combined and enriched specifically by organizations (Airlines, ANSPs, Manufactures) and provided to end-users for the accomplishment of specific tasks (such as the data in an EFB for a pilot conducting a specific flight). Currently, the relationships between these different data products are hidden in APIs or non-standardized interfaces of web services. This makes the discovery and combination of data products an intricate task with a lot of intervention of IT personnel.

The focus of BEST is the description and discovery of data products as semantic containers. Further, the approach also allows to model the whole derivation chain of semantic containers, not only focusing the static aspects of semantic containers but also the activities that derive one container from other containers.

In order to model derivation chains, BEST distinguishes four kinds of activities:

- **Filter**: reduce the number of data items using rules based on a membership condition
- **Enrich**: an activity that derives secondary data items
- **Combine**: an activity with two or more semantic containers of the same data item type as input and one homogeneous composite container as output
- **Compose**: an activity with two or more semantic containers of possibly different data item types as input and one heterogeneous composite container as output. The composition may involve the filtering of one container regarding another (including semi-joins)

Figure 6 shows an example of a derivation chain with one combine activity (“Combine METARs for Europe and Middle East”), three similar compose activities (“Compose EFB for …”), three similar enrich activities (“Prioritize NOTAMs …”) and four filter activities (“Filter NOTAMs …”, “Filter METARs …”). The example is partially based on the scenario of the flight from Dubai (DXB) to Vienna (VIE) presented in BEST Deliverable 3.1. Certain containers can be reused to compose EFBs for different flights. For example, the METAR container for Europe and the Middle East can also be the starting point for the processing of information for other European flights.

This modelling approach is agnostic with regard to the implementation of the activities. For example, the “Prioritize NOTAMs<Flight: EK127, Time: 05/04/2017>” activity could be realized by a knowledge-based system like SemNOTAM or also by a human expert who manually derives priorities for NOTAMs with regard to a flight (think of an expert writing some exclamation marks “!” and “!!” on a NOTAM printout which is given to a pilot).

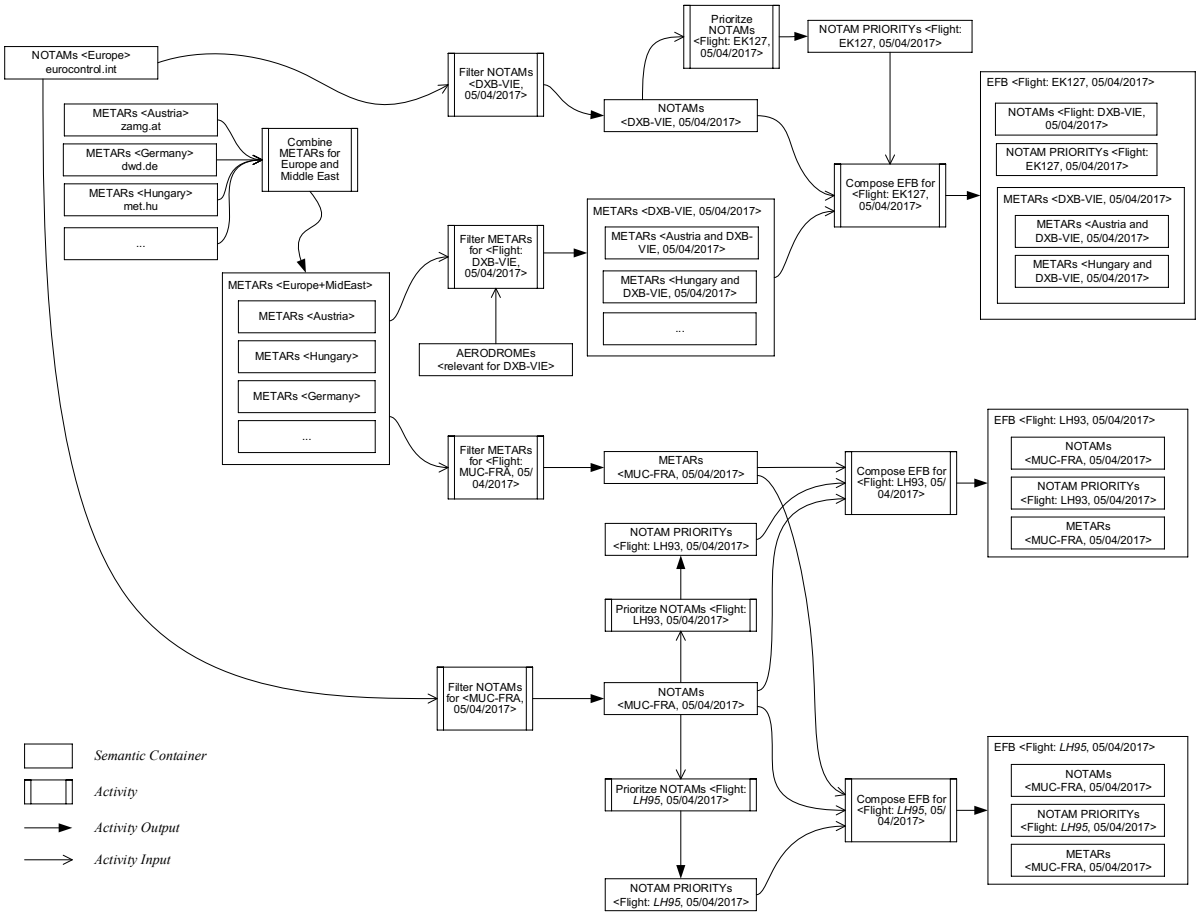


Figure 6. A derivation chain of semantic containers and activities based on the scenario in BEST Deliverable 3.1

5 Fitness of semantic web technologies for container management and discovery

In this section, we discuss the fitness of common semantic web technologies for semantic container management and discovery. Specifically, we consider RDF(S) and SPARQL, OWL and SWRL as well as F-Logic and RIF as possible technology choices for realizing the semantic container approach. These technologies can be used to describe containers and container content; logical reasoning then allows discovering and populating semantic containers. Concerning reasoning, one must distinguish between subsumption reasoning, i.e., deciding whether one semantic container is more general/specific than another, and membership reasoning, i.e., deciding whether a specific aeronautical data item belongs to a specific semantic container.

5.1 RDF(S) and SPARQL

The Resource Description Framework (RDF) as one of the semantic web's foundational technologies may serve as a simple and efficient means for expressing semantic container descriptions. The versatile organization of data into triples of subjects, predicates and objects makes RDF the ideal format for the representation of semi-structured, complex data and knowledge. The reliance on the International Resource Identifier (IRI) to denote and uniquely identify entities and properties of entities paves the way for linking RDF data sources with each other, allowing for a modular representation of knowledge. Since RDF typically also serves as the serialization format for ontologies, the knowledge thus encoded in the ontologies then complements the available semantic container descriptions in RDF format.

RDF Schema (RDFS) allows for the definition of a simple vocabulary, a basic ontology with very limited expressive power but efficient mechanisms for inference of new knowledge. Basically, RDFS allows for the assertion of subclass relationships between entity classes as well as the definition of domain and range of properties. These definitions allow RDFS reasoners to efficiently infer class membership of entities, albeit in a limited way due to the limited expressiveness of RDFS compared to more comprehensive ontology languages.

In the semantic container approach, RDF and RDFS may serve to define facets and facet values of semantic containers. In that case, a semantic container corresponds to an RDF resource, with the container's content being described by properties that correspond to the different facets. For example, the resource `Container_1` with a `relevantToAircraft` property referencing the `FixedWingAircraft` resource signifies that the specific semantic container represented by the `Container_1` resource contains aeronautical information relevant to fixed wing aircrafts. The domain of the `relevantToAircraft` property is then a class/resource `Container`, the range a class/resource `Aircraft`. This `relevantToAircraft` property represents the *Aircraft* facet of semantic container descriptions, characterizing the content of the semantic container with respect to the type of aircraft the content is relevant for. The values for this facet are then inferred to be instances of `Aircraft`., e.g., `FixedWingAircraft` is an instance of `Aircraft`. Now, given a specific individual aircraft `Aircraft_O-1234`, if one knows that the aircraft's type is fixed-wing aircraft, one knows that `Container_1` is relevant to `Aircraft_O-1234`.

Handling specialization hierarchies of aeronautical concepts is of paramount importance for management and discovery of semantic containers. For example, there are different types of aircraft

which are in various specialization relationships with each other, e.g., sea plane is a kind of fixed-wing aircraft, helicopter a kind of rotary-wing aircraft, and specific aircrafts then typically belong to multiple types of aircraft. Given such specialization relationships of aircraft types, one must determine whether a semantic container that is relevant for a specific type of aircraft is relevant for a specific individual aircraft. Similarly, given a semantic container relevant for a specific type of aircraft, e.g., fixed-wing aircraft, one must decide whether the semantic container is also relevant for another type of aircraft, e.g., sea planes. Generally speaking, unless there is a mechanism to efficiently handle specialization hierarchies of aeronautical concepts, managing semantic containers becomes error-prone, determining whether a semantic container is relevant for a specific information need becomes cumbersome.

Consider, for example, the semantic containers in Figure 7a, which contain METAR objects relevant to fixed-wing aircrafts, rotary-wing aircrafts, and sea planes, respectively. The seaplane-relevant METAR container is subsumed by the METAR container relevant to fixed-wing aircraft. The subsumption relationship between semantic containers can be derived from the specialization relationships between the concepts that are used as facet values (Figure 7b), e.g., the *SeaPlane* concept is a specialization of *FixedWingAircraft*, the *Helicopter* concept a specialization of *RotaryWingAircraft*; thus can be derived the seaplane-relevant METAR container is subsumed by the METAR container relevant to fixed-wing aircraft. Listing 1 then contains the RDFS class hierarchy that corresponds to the concept hierarchy in Figure 7b, along with an individual plane `Aircraft_O-1234` that belongs to the `SeaPlane` class. An RDFS reasoner may now derive class membership of `Aircraft_O-1234` to `FixedWingAircraft`. Notice that each class is marked an instance of itself[3], which serves querying purposes and is discussed later. Listing 2 contains the RDF representation of the METAR containers in Figure 7a, the facet values of which defined by the `relevantToAircraft` property using the previously defined RDFS classes; the `METAR_Container` resource, in this example, denotes the class of semantic containers that contain METARs.

The RDFS reasoner alone cannot determine subsumption hierarchies of semantic containers: To this end, we must resort to a query language. SPARQL is the standard query language for RDF data and may serve for the discovery of semantic containers, the descriptions of which are represented using RDF and RDFS. The queries consist of triple patterns that specify the characteristics of the requested data. SPARQL is a database query language and as such operates under closed-world assumption: Queries retrieve from the specified data sources those data items with given characteristics. SPARQL complements the capabilities for logical inference provided by automatic reasoners, which may populate the RDF database with additional inferred data which subsequently are accessible to SPARQL queries.

Subsumption reasoning for semantic containers may be encoded into SPARQL queries which then serve to retrieve the containers that satisfy the given information need. Consider, for example, the SPARQL query in Listing 3: The query returns all resources that denote semantic containers relevant to a specific individual aircraft, `Aircraft_O-1234`, using the `relevantToAircraft` facet.

[3] The `a` property is short for `rdf:type` which denotes class membership of a resource.

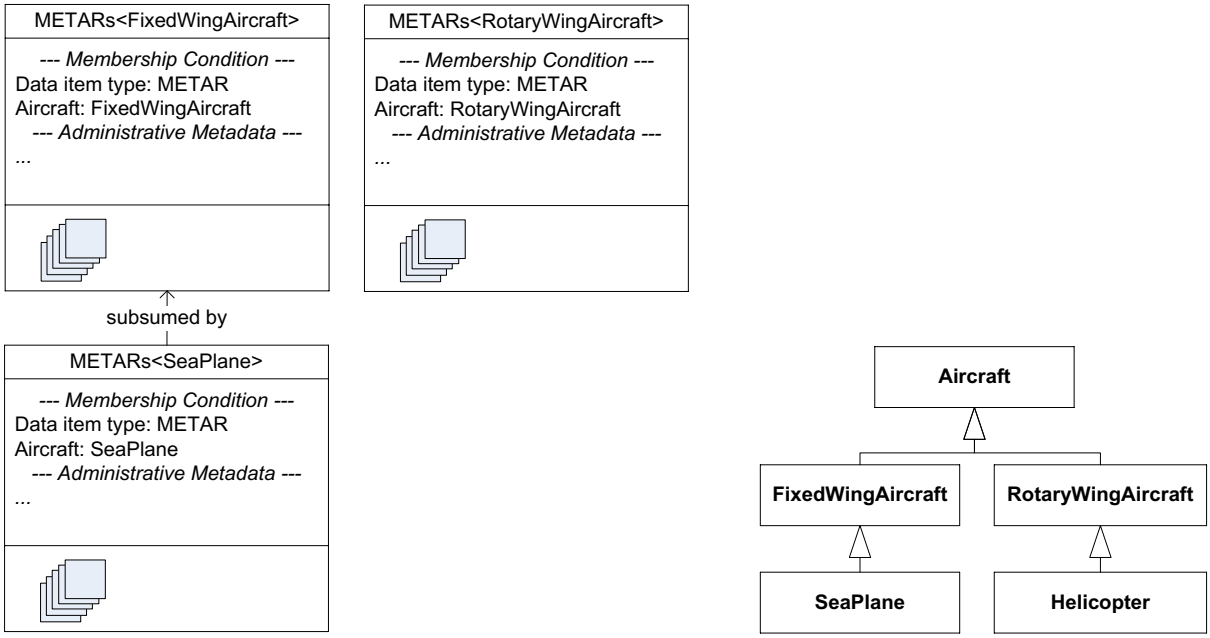


Figure 7. Semantic containers and concepts for container description

```
:FixedWingAircraft rdfs:subClassOf :Aircraft ;
                   a :FixedWingAircraft .
:SeaPlane rdfs:subClassOf :FixedWingAircraft ;
          a :SeaPlane .
:RotaryWingAircraft rdfs:subClassOf :Aircraft ;
                    a :RotaryWingAircraft .
:Helicopter rdfs:subClassOf :RotaryWingAircraft ;
            a :Helicopter .
:Aircraft_O-1234 a :SeaPlane .
```

Listing 1. The RDFS class hierarchy that corresponds to the concept hierarchy in Figure 7b. Each class is also an asserted instance of itself in order to facilitate container discovery in case the interest focus is on a type rather than an actual individual aircraft.

```
:Container_1 a :METAR_Container ;
             :relevantToAircraft :FixedWingAircraft .
:Container_2 a :METAR_Container ;
             :relevantToAircraft :SeaPlane .
:Container_3 a :METAR_Container ;
             :relevantToAircraft :RotaryWingAircraft .
```

Listing 2. RDFS container definitions corresponding to the containers in Figure 7a

```
SELECT ?container WHERE {
  ?container a :METAR_Container .
  ?container :relevantToAircraft ?aircraft .
  :Aircraft_O-1234 a ?aircraft .
}
```

Listing 3. A SPARQL query that returns all containers relevant to the specific aircraft `Aircraft_O-1234`

?container
:Container_1
:Container_2

Table 1. Result of the SPARQL query in Listing 3 executed on the RDF data in Listing 1 and Listing 2with RDFS reasoning

```
SELECT ?container WHERE {
  ?container a :METAR_Container .
  ?container :relevantToAircraft ?aircraft .
  :FixedWingAircraft a ?aircraft.
}
```

Listing 4. A SPARQL query that returns all containers relevant to an aircraft type

?container
:Container_1

Table 2. Result of the SPARQL query in Listing 4 executed on the RDF data in Listing 1 and Listing 2 with RDFS reasoning

When performed on the RDF data in Listing 1 and Listing 2, the result (Table 1) includes the containers `Container_1` and `Container_2`, relevant to fixed-wing aircraft and sea planes, respectively. The relevance of `Container_2` follows straight from the assertion that `Aircraft_O-1234` is a sea plane. The relevance of `Container_1` follows from the inferred classification of `Aircraft_O-1234` as a fixed-wing aircraft. The SPARQL query in Listing 3 can be easily extended to cope with multiple facets by adding additional triple patterns in the `WHERE` clause corresponding to the respective facet.

In Listing 1, we declared each RDFS facet class to be an instance of itself. The semantics of such a declaration is as follows: In its role as an instance, a facet class denotes some generic instance of the aircraft type. Then the same query pattern as with individual instances of the facet classes works for retrieval of relevant semantic containers. Consider, for example, an information need for a METAR container relevant to fixed-wing aircraft, i.e., containing all the necessary METARs for this type of aircraft. The corresponding SPARQL query (Listing 4) follows the same pattern as before and, when performed on Listing 1 and Listing 2, yields the result illustrated in Table 2.

Aeronautical information systems are inherently geographic information systems that must be able to handle geographic data well. RDF and SPARQL as such are ill-equipped to handling geographic information. But, the GeoSPARQL (Perry and Herring 2012) extension of RDF and SPARQL provides advanced concepts and query operators for working with geographic entities. For example, Geo-

SPARQL provides predicates for comparing two geographic shapes with respect to their relationship, i.e., distinctiveness, overlap, containment, and so on. Thus, for geography facets of semantic containers, GeoSPARQL might be a viable technology choice. Related work (Balaban 2016) employs GeoSPARQL for discovery of aeronautical web services.

The main disadvantage of RDFS is its rather limited expressiveness and thus limited capabilities for automated reasoning. More complex concepts must be expressed in SPARQL rather than in logical terms. For some facets, however, the limited expressiveness of RDFS is unproblematic, e.g., provenance and freshness. In that case, users may predominantly work with simple ad hoc queries to get the freshest container out of a logically determined set of containers that fulfil the information need. For example, a semantic container could have a data property `lastUpdate` with a date/time as value. A typical query would then just ask for the `lastUpdate` value to be after some particular date.

Concerning storage of semantic container descriptions, native triple stores offer convenient storage facilities for RDF data. A SWIM registry for semantic containers could be realized upon such RDF triple stores. Typically, such triple stores also offer built-in reasoning support which is then also available in the SPARQL query engine.

5.2 OWL and SWRL

Realizing the semantic container approach with RDF and RDFS comes with a downside: Conditions for container subsumption as well as more complex concepts must be encoded into SPARQL queries and manually maintained; finding the container that most closely matches the information need also requires more complex SPARQL queries. More comprehensive ontology languages than RDFS, on the other hand, would offer more flexible, declarative definition of concepts, along with reasoning support that would facilitate finding the closest match to an information need.

The Web Ontology Language (OWL) is a popular ontology language based on RDF and RDFS. Different profiles exist, with different degrees of expressivity – expressivity in different profiles reduced for the sake of efficiency and decidability. OWL Description Logic (DL) is among the most common OWL profiles, presenting some restrictions in order to make OWL decidable (W3C 2012b). OWL EL (W3C 2012c), in turn, further trades off expressiveness, such as disjunction, for the sake of increased performance and scalability. OWL distinguishes classes and properties. Concerning properties, OWL distinguishes data properties and object properties, the former's range being an atomic type, the latter's range being an OWL class. With respect to RDFS, OWL allows for more advanced concept definitions, allowing classes to be defined equal to more complex logical expressions.

A first advantage of OWL over more simple languages such as RDFS is the ability to uniformly and unambiguously define certain terms which are used to describe the contents of semantic containers. For example, the term *heavy aircraft* lacks a standard definition in the aeronautical domain. Every information service provider, every aeronautical actor could use that term with a different meaning. Using OWL, each information service provider can publish their definition of terms and actors will know what the term exactly means. Consider, for example, the OWL class hierarchy in Figure 8 which shows different aircraft types ordered hierarchically. These aircraft types have different characteristics which can be expressed in OWL as properties, such as the maximum payload by object property `hasMaxPayload`. Using the properties, OWL classes can then be defined equivalent to a certain expression, e.g., `HeavyAircraft` is equivalent to an aircraft with a maximum payload greater or equal 150 metric tons. Note that different information service provider may come up with different

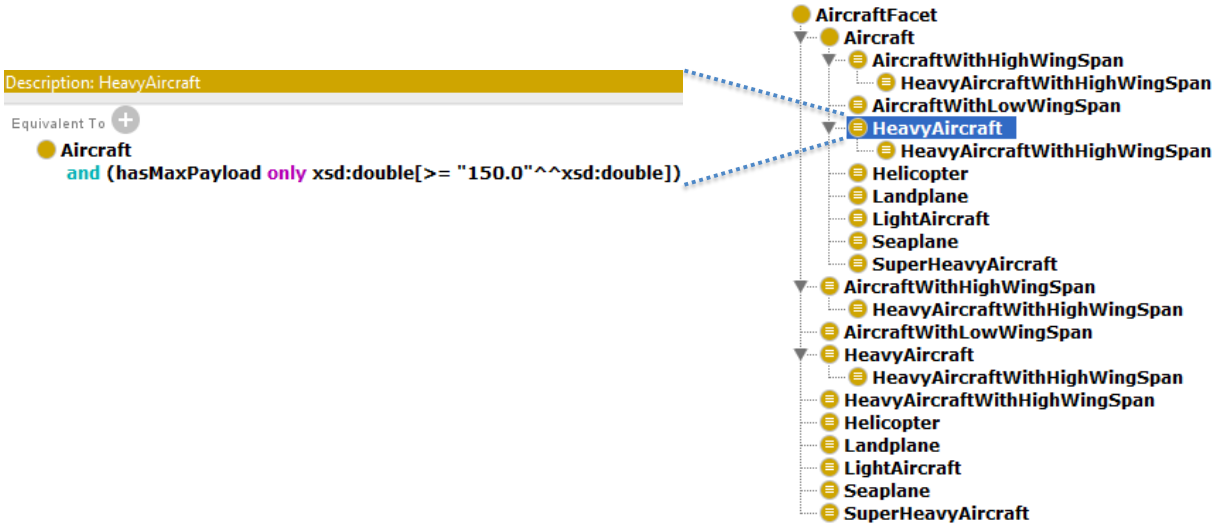


Figure 8. Definition of an OWL class using a logical expression

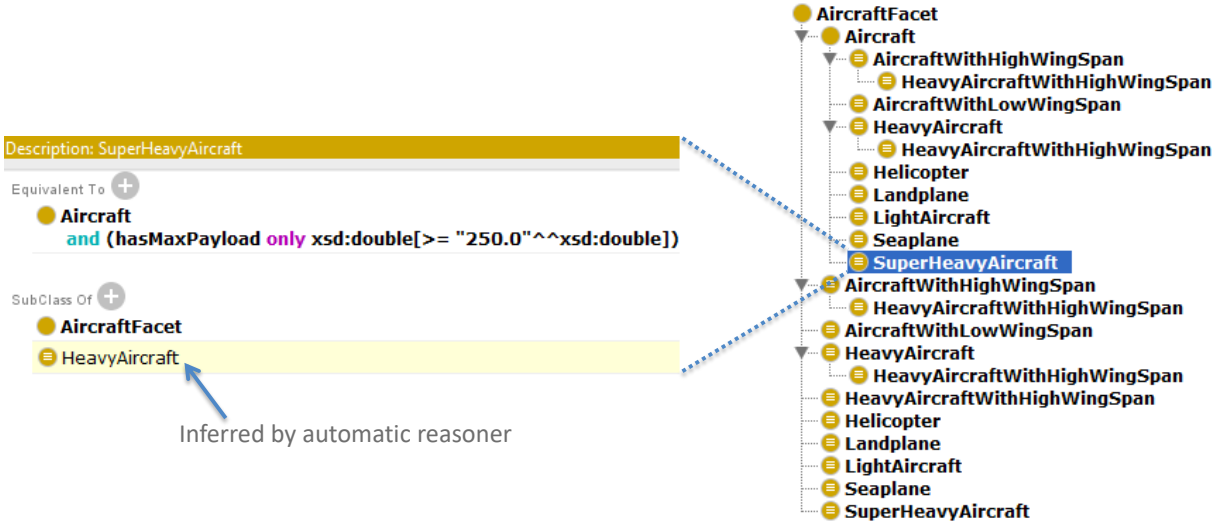


Figure 9. Derivation of subclass relationship through automated reasoning

definitions of what constitutes a heavy aircraft that they use in their semantic container descriptions. When published, air traffic actors can view these definitions and use them for the querying.

Based on the class definitions, an OWL reasoner can automatically perform subsumption reasoning for classes, i.e., a derivation of specialization relationships between OWL classes. Consider, for example, the definition of `SuperHeavyAircraft` as an aircraft with a maximum payload greater or equal to 250 metric tons. An OWL reasoner can derive that class to be a subclass of the previously defined `HeavyAircraft` class. Thus, the use of OWL greatly facilitates management of concept definitions and can contribute towards unambiguous communication between information service providers and air traffic actors. Each information service provider can publish its ontology that ex

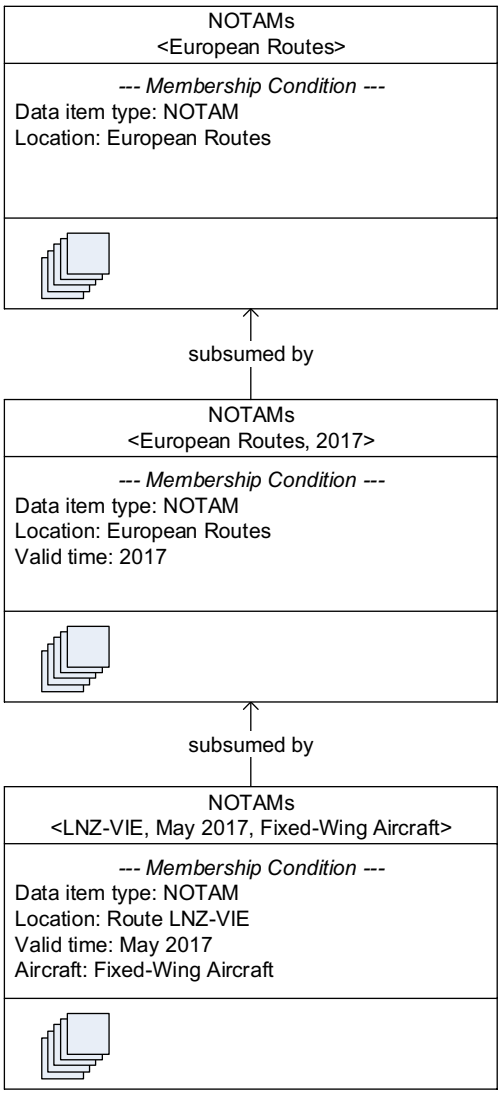


Figure 10. Semantic containers with NOTAM data items

plains what the different terms mean; the automatic reasoner organizes these terms in a specialization hierarchy for increased manageability and comprehensibility.

OWL class definitions and properties can then be employed for the definition of semantic container descriptions. Consider, for example, the semantic containers illustrated in Figure 10. These Digital NOTAM (DNOTAM) containers are in a subsumption hierarchy. The most general container has NOTAMs for all European routes, then there's the container for DNOTAMs for European routes valid in 2017, followed by the container for DNOTAMs that concern fixed-wing aircraft on the route from Linz to Vienna in May 2017. These semantic containers then translate into OWL classes the definitions of which employ object properties that correspond to the different facets. For example, in Figure 11, the `DNOTAM_Container1` class corresponds to the DNOTAM container for European routes. In Figure 12, the `DNOTAM_Container2` class corresponds to the DNOTAM container for European routes in 2017. In Figure 13, the `DNOTAM_Container3` class corresponds to the DNOTAM container for fixed-wing aircraft for the route from Linz to Vienna in May 2017.

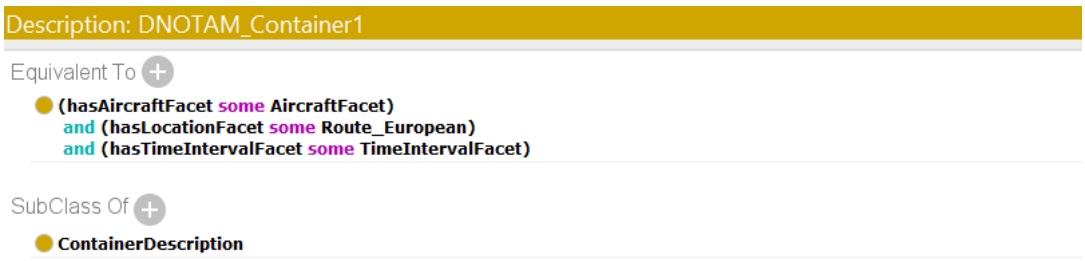


Figure 11. Semantic container for information concerning European routes

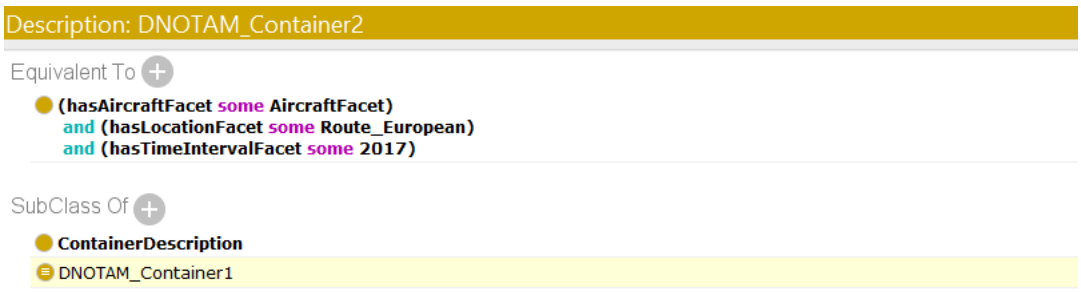


Figure 12. Semantic container for information concerning European routes in 2017

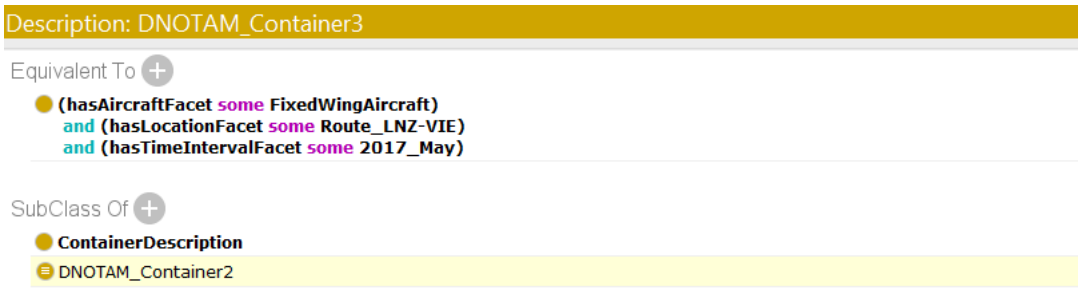


Figure 13. Semantic container for information concerning the route from Linz to Vienna in May 2017

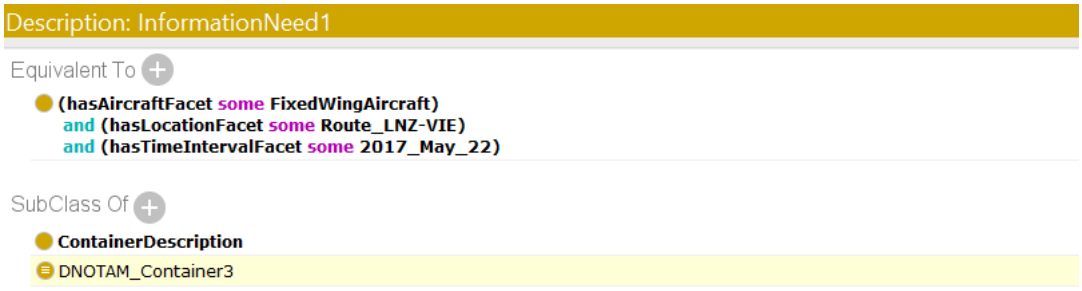


Figure 14. Information need expressed as semantic container description

Container descriptions in OWL allow for the use of standard OWL reasoners for the derivation of specialization relationships between semantic containers which, in turn, allows for discovery of semantic containers that fit a certain information need. The information need can be expressed as an OWL class and classified into the subsumption hierarchy of semantic container descriptions by the automatic reasoner. Consider, for example, the class definition in Figure 14, which expresses the information need for a container with data items relevant for fixed-wing aircraft on the route from Linz to Vienna on the 22 May 2017. The OWL reasoner can then infer subclass relationships as with any other semantic container. In this example, `InformationNeed1` is inferred to be a subclassOf `DNOTAM_Container3`.

OWL has limited support for expressing geographic concepts and performing reasoning over these concepts. With respect to expressing geographic areas, bounding-box approximations work for OWL, although concerning scalability, there is a limit with respect to the number of classes that can be handled efficiently (see Section 6). Consider, for example, the previous example with European NOTAMs and NOTAMs relevant for the route from Linz to Vienna. The automatic reasoner should automatically determine whether two geographic areas are in a subsumption relationship. This can be done by expressing the concepts by coordinates that span a bounding box on the map, as illustrated in Figure 15. Bounding boxes, unlike GML shapes, can be easily expressed in OWL and handled by the automatic reasoner. To this end, the `hasLongitude` and `hasLatitude` data properties can be used in class expressions to define a class as equivalent to a certain bounding box (Figure 16). The range of the `hasLongitude` and `hasLatitude` properties for the thus specified geographic entities must be restricted, by means of subclass assertion, to a range of -180 to +180 for longitudes and -90 to +90 for latitudes. The route from Linz to Vienna can then likewise be expressed as a bounding box and inferred to be a subclass of the European area (Figure 17).

The Semantic Web Rule Language (SWRL) can be used to express concepts that are not expressible in OWL, e.g., an overloaded aircraft as an individual where the current payload exceeds its maximum allowed payload. Another alternative are SPARQL-based rules. If OWL is also used for membership reasoning, i.e., decide whether a data item belongs to a containe, then in many cases SWRL or another rule language will be needed. Using these rule languages, new information can be derived by means of evaluation of declarative rules. For example, it could be expressed that given some property values, a data item is to be classified as important or not important. Likewise, containers could be considered relevant for an information need if their facet properties satisfy certain conditions.

Since RDF serves as the serialization format for OWL, RDF triple stores can also serve to realize the technical infrastructure of an OWL-based semantic container registry. SPARQL queries could then complement reasoning-based container discovery. For example, provenance and freshness may be expressed as RDF properties and then, after filtering for content using the OWL reasoner on the facets of the semantic containers, a SPARQL query may select those containers that satisfy a given condition on freshness and provenance. Similarly, GeoSPARQL could be used to express geographic concepts and combined with OWL as GeoSPARQL provides a set of RDF/OWL concepts in order to represent geographic entities and their relationship to each other. Thus, the choice between RDF(S)/SPARQL and OWL/SWRL is not a clear-cut this-or-that decision but often times a combination of these technologies will have to be employed in practice.

Figure 15. Bounding boxes for Europe as well as the relevant area for the flight from Linz to Vienna

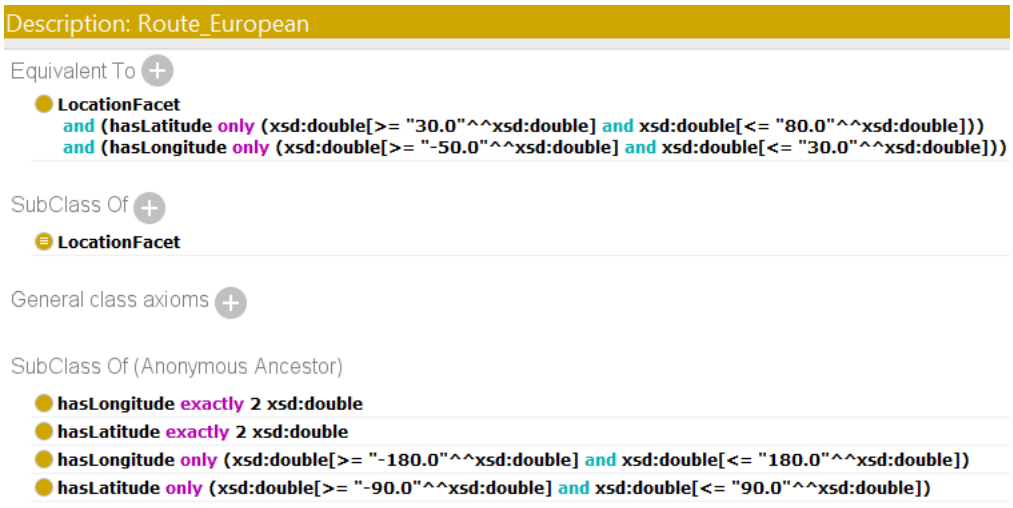


Figure 16. Bounding-box concept for European routes

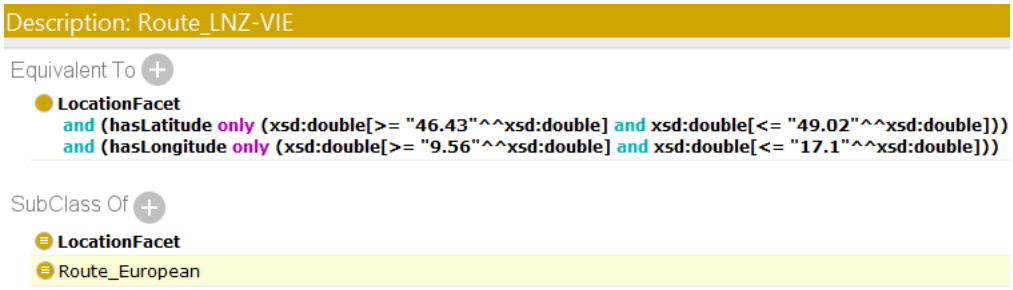


Figure 17. Bounding-box concept for the route from Linz to Vienna

5.3 F-Logic and RIF

Frame Logic (F-Logic) is a logic-based knowledge representation and query language which can be informally described as a kind of “object-oriented Datalog”. F-Logic, in its dialect ObjectLogic, has been successfully employed in the aeronautical domain for building an expert system for the filtering and prioritization of Digital NOTAMs (Steiner et al. 2016) in the course of the SemNOTAM project. The SemNOTAM approach, where aeronautical data items (NOTAMs) are represented in F-Logic and filtering/prioritization rules are realized as predicates, can be adapted for other types of aeronautical data items, e.g., METARs. The SemNOTAM approach, however, realizes only membership reasoning, i.e., determining whether a data item is part of a specific container. While the query and prioritization rules used to populate a NOTAM container with SemNOTAM could then serve as semantic container description, subsumption reasoning becomes hard. In that case, a more viable option would be the translation of the F-Logic rules to an RDF or OWL representation with reduced expressivity which can then be handled more efficiently.

The Rule Interchange Format (RIF) is a W3C recommendation for expressing and exchanging rules. The RIF Basic Logic Dialect (BLD) incorporates aspects of F-Logic, namely frames and objects (W3C 2013a), and thus introduces these aspects of F-Logic to the semantic web. RIF is compatible with RDF and OWL (W3C 2013b) and, therefore, a RIF-based approach could complement an approach that employs OWL and RDF for management and discovery of semantic containers. Again, the choice between F-Logic/RIF and other semantic web technologies is not a clear-cut this-or-that decision but often times a combination of these technologies will have to be employed in practice.

5.4 Guidelines for technology choice

With respect to technology choice, one must distinguish between subsumption and membership reasoning. The former refers to determining the subsumption relationships between containers, i.e., whether one container is more specific than the other, and which containers then satisfy a given information need. The latter refers to determining whether an individual data item belongs to a given semantic container. Table 3 then provides an overview of the roles semantic web technologies may assume in the semantic container approach. In particular, the following criteria and questions guide the choice for particular technologies:

- **Expressiveness**
  - What are the facets, e.g., geospatial and time facets, of semantic container descriptions? These facets differ by entity type contained in the semantic containers. For example, NOTAM containers have different facets for content description than METAR containers. The facets influence the choice of technology since the ontology language influences what kind of concepts can be expressed.
  - What is the granularity of each facet? For example, spatial facets may represent geographic entities as GML shapes (high granularity) or bounding boxes (low granularity). The granularity influences how precise container content and information need may be expressed. Granularity also influences performance: High granularity typically requires more effort for management and discovery of semantic containers; on the other hand, container content then matches the information need more closely, thus leading to reduced effort for the end user application.
  - How are the facets combined, e.g., simple conjunction of facet values that apply to the semantic container’s contents or complex disjunctions?

- **Workload**
  - What is the total number of semantic containers? Compared to traditional query processing, automatic reasoning with open world assumption performs less efficient on large datasets. In case of large datasets, using plain SPARQL queries rather than automated reasoning may be the more viable option to semantic container management and discovery. Scalability will be examined in BEST Deliverable 5.1.
  - What is the expected number of new semantic containers in a specified time period? Since new semantic containers have to be integrated into the subsumption hierarchy by the automatic reasoner, the frequency of container additions influences technology choice. Only if incremental reasoning in automatic reasoners can be realized efficiently, OWL becomes a viable option for handling high-frequency additions of semantic containers.
  - How many queries for semantic container discovery are issued in a specified time period? In reasoning-based container discovery, the query corresponds to the integration of a new semantic container into the subsumption hierarchy.
  - What is the expected complexity of semantic container descriptions? This boils down to the question how small or large semantic containers can become with respect to the content: Will there be relatively few containers with quite large amounts of data items or will there be lots of containers that quite closely match the information need and thus contain relatively few data items?
- **Response time for different tasks**
  - Publishing of semantic containers
  - Container discovery: Will there be a need for real-time and emergency-critical discovery of semantic containers, or are containers for potential emergency situations discovered beforehand and then quickly retrieved from some local index?
- **Off-the-shelf vs. custom-made tools**
  - Single tool vs. multiple tools: A single tool or tool suite from a single vendor facilitates interoperability between the different components.
  - Amount of customization required
  - Proprietary tools: Some semantic web technologies are backed by proprietary tools, e.g., ObjectLogic (as a dialect of F-Logic) requires OntoStudio and OntoBroker to work with and develop applications.
- **Time-to-market**
- **Maintenance**

In general, common semantic web technologies are fit for semantic container management in the aeronautical domain. Various factors influence technology choice; scalability aspects are more closely considered in BEST Deliverable 5.1. Future development of a semantic container management system, registry, and tool suite may well employ a mixture of existing semantic web technologies. A viable option seems to be a reliance on a combination of RDF(S), OWL, SPARQL and GeoSPARQL for subsumption reasoning as well as F-Logic and RIF for membership reasoning. To what degree each of these technologies is employed depends on the laid out factors.

Subsumption / Membership	RDF(S) and SPARQL	OWL and SWRL	F-Logic and RIF
RDF(S) and SPARQL	• Very flexible but rather low-level implementation • Most reasoning must be realized manually using SPARQL queries	• Semantic container description with expressivity limited to the expressivity of OWL • Off-the-shelf reasoners can be used for subsumption reasoning, possibly complemented by SWRL rules • A series of custom SPARQL queries realizes membership reasoning	• F-Logic rules for determining subsumption relationships of filtering parameters used to populate container • A series of custom SPARQL queries realizes membership reasoning
OWL and SWRL	• Data items must be expressed in OWL • SWRL allows for the formulation of filtering rules • SPARQL queries select relevant containers	• Data items must be expressed in OWL • SWRL allows for the formulation of filtering rules • Off-the-shelf reasoners can be used for subsumption reasoning, possibly complemented by SWRL rules	• F-Logic rules for determining subsumption relationships of filtering parameters used to populate container • Data items must be expressed in OWL • SWRL allows for the formulation of filtering rules
F-Logic and RIF	• Semantic containers are populated by expert systems like SemNOTAM, using F-Logic for representation of data items and filtering rules • Translation of the F-Logic rules used to populate the container into (possibly simplified) RDF representation	• Semantic containers are populated by expert systems like SemNOTAM, using F-Logic for representation of data items and filtering rules • Translation of the F-Logic rules used to populate the container into simplified OWL representation	• Semantic containers are populated by expert systems like SemNOTAM, using F-Logic for representation of data items and filtering rules • F-Logic rules for determining subsumption relationships of filtering parameters used to populate container

Table 3. Overview of the different semantic web technologies and their potential roles in semantic container approach

6 Faceted ontology-based description and discovery of semantic containers

A semantic container's content is characterized by a *semantic description* which has different facets (Figure 18). For each facet, the semantic description has as the value a concept from some ontology. Different (aeronautical) ontologies may serve as source for such facet values. Information service providers may employ their own ontologies, building on existing reference ontologies (see Deliverable 1.1) in order to facilitate interoperability.

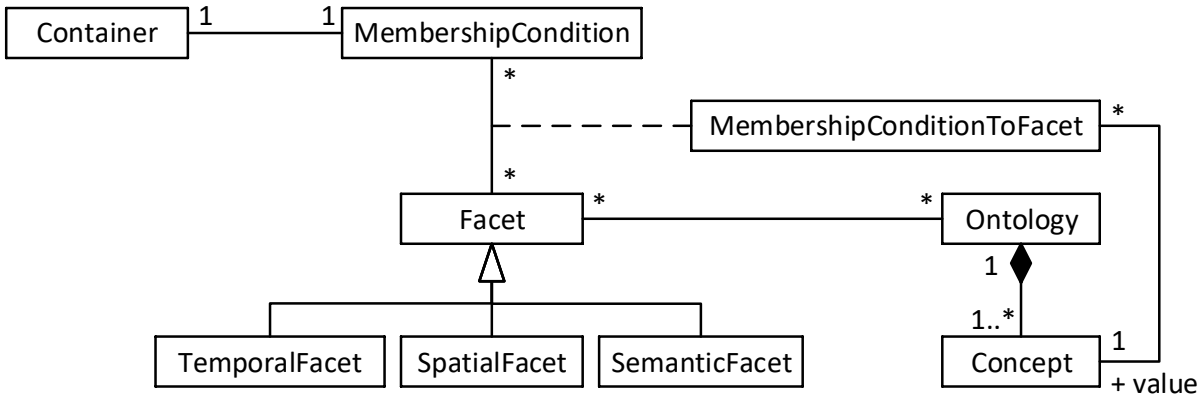


Figure 18. Abstract data model of a container's semantic description

We distinguish three types of facets, namely *temporal*, *spatial* and *semantic* facets. Semantic containers typically contain data with a specific temporal and spatial focus; these are captured by temporal and spatial facets, respectively. The term "semantic facets", in turn, denotes an umbrella category of facets covering diverse domain-specific aspects of the contents of semantic containers, such as the aircraft type that the contained data refer to.

The contents of a data container refer to time periods in various ways. For example, NOTAMs have a validity time as well as an application time, METARs have an observation time, and TAFs have a validity time. Different from temporal facets are indicators of freshness, such as the update time, which are part of the administrative metadata (see Section 7).

A note about interpretation of semantic descriptions: A container contains *all* the relevant data items for the context identified by the semantic description's facet values. For example, if a semantic description has a location facet with *Frankfurt airport* then a NOTAM container with this semantic description contains all the NOTAMs relevant for the Frankfurt airport.

6.1 Experimental setting: OWL API and HermiT reasoner

In order to evaluate the technical feasibility of the presented faceted ontology-based approach and to determine the trade-offs of various design choices we conducted the experiments as presented in this section.

The experiments were conducted on a machine with the following characteristics:

- Operating system: Windows 10 Pro, 64 Bit
- CPU: Intel® Core™ i7-5600, 2,6 GHz
- RAM: 16 GB

The programs are written in Java (JavaSE-1.8) making use of the HermiT OWL reasoner (version 1.3.8) together with the OWL API (version 3.4.3). The OWL API[4] and the HermiT[5] reasoner are bundled.

The OWL API is a reference implementation for working with OWL ontologies in Java. The OWL API allows to create and manipulate OWL ontologies programmatically in Java and to parse and serialize OWL ontologies in the various OWL syntaxes. The OWL API is released as open-source software under the GNU Lesser General Public License (LGPL) or the Apache License.

HermiT is an open-source reasoner for OWL ontologies, covering all language constructs of OWL 2 (W3C 2012e) together with SWRL, restricted to DL-safe rules (Motik et al. 2005). As a reasoner based on Description Logics, HermiT supports OWL 2 direct semantics (W3C 2012a) and not OWL 2 RDF-based semantics (W3C 2012d).

Given an OWL ontology, HermiT can, among others,

- check the satisfiability of the ontology
- check the satisfiability of a class in the ontology
- derive the subsumption hierarchy of classes
- derive the members of a class

The OWL ontology used as input for the reasoner can be extended (based on an OWL file) or fully generated programmatically using the OWL API. This is important to generate various ontologies for the experiments.

6.2 Semantic facet ontologies based on the AIRM ontology

Concepts used as facet values in semantic descriptions of semantic containers are defined in facet-specific ontologies using vocabulary (classes, properties, codelists) from the AIRM ontology (see deliverable D1.1).

6.2.1 Defined classes based on the AIRM ontology

We demonstrate the use of the AIRM ontology (see deliverable D1.1) for the definition of facet-specific ontologies using a very small and rather simple facet-specific ontology (see below). The classes defined in this ontology can be used as facet-values of semantic facet 'Aircraft' in semantic descriptions.

Terms taken from the AIRM ontology are shown in bold font, for example, the defined classes `Helicopter`, `Balloon` and `Aeroplane` are defined using object property `AircraftIcaoAircraftCategory` and individuals aircraft categories `HELICOPTER`, `BALLOON`, `AEROPLANE` taken from the AIRM ontology. Class `Aircraft-with-EnhancedVision-`

[4] http://owlapi.sourceforge.net/

[5] http://www.hermit-reasoner.com/.

System is defined using object properties Aircraft-aircraftEquipment and Aircraft-Avionics-type and individual ENHANCED_VISION_SYSTEM taken from the AIRM ontology. Classes BlueAircraft and RedAircraft are defined using object properties Aircraft-configuration and OperatorConfiguration-colourAndMarking together with data property AircraftColourAndMarking-aircraftColour from the AIRM ontology.

Such classes with a definition based on the AIRM ontology can in turn be used for the definition of further defined classes. For example, class Aeroplane-or-Helicopter is defined as the union of Aeroplane and Helicopter. Class Aeroplane-with-EnhancedVisionSystem is defined as the intersection of Aeroplane and Aircraft-with-EnhancedVisionSystem. Classes BlueBalloon and RedBalloon are defined as the intersection of Balloon and BlueAircraft or RedAircraft, respectively.

```
Ontology: <http://www.project-best.eu/owl/facets/aircraft>

Import: < http://www.project-best.eu/owl/airm-mono>

Class: Helicopter
    EquivalentTo:
        Aircraft-icaoAircraftCategory value HELICOPTER

Class: Balloon
    EquivalentTo:
        Aircraft-icaoAircraftCategory value BALLOON

Class: Aeroplane
    EquivalentTo:
        Aircraft-icaoAircraftCategory value AEROPLANE

Class: Aeroplane-or-Helicopter
    EquivalentTo:
        Aeroplane or Helicopter


Class: Aircraft-with-EnhancedVisionSystem
    EquivalentTo:
        Aircraft-aircraftEquipment some (
          AircraftAvionics-type value ENHANCED_VISION_SYSTEM
        )

Class: Aeroplane-with-EnhancedVisionSystem
```

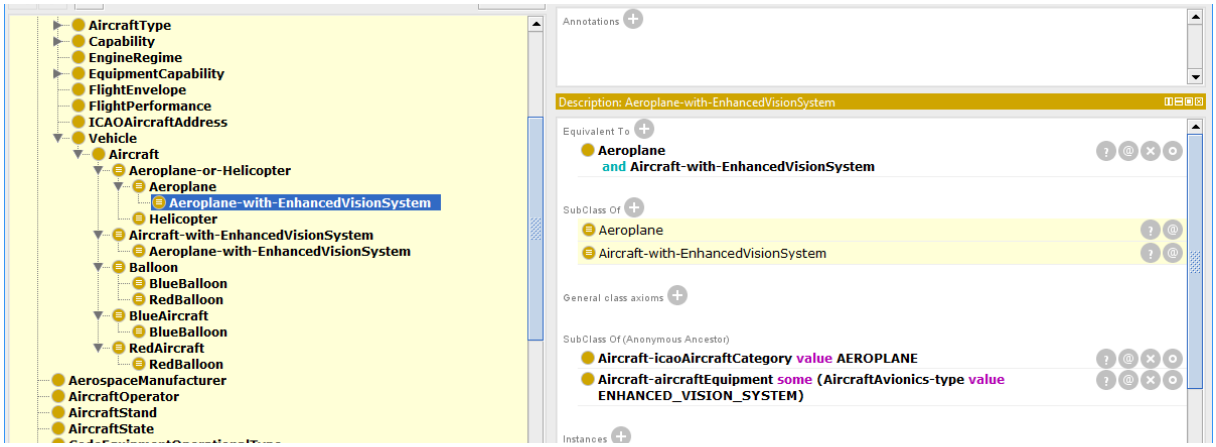
```
    EquivalentTo:
        Aeroplane
         and Aircraft-with-EnhancedVisionSystem

Class: BlueAircraft
    EquivalentTo:
        Aircraft-configuration some (
          OperatorConfiguration-colourAndMarking some (
            AircraftColourAndMarking-aircraftColour
              value "blue"^^xsd:string
          )
        )

Class: RedAircraft
    EquivalentTo:
        Aircraft-configuration some (
          OperatorConfiguration-colourAndMarking some (
            AircraftColourAndMarking-aircraftColour
              value "red"^^xsd:string
          )
        )

Class: BlueBalloon
    EquivalentTo:
        Balloon
         and BlueAircraft

Class: RedBalloon
    EquivalentTo:
        Balloon
         and RedAircraft
```

6.2.2 Subsumption hierarchies of classes based on the AIRM ontology

The ultimate goal of using OWL and semantic reasoners for semantic container management is to organize semantic containers in a subsumption hierarchy (a subclass/superclass hierarchy) and to detect semantic containers that fulfil a given information need. One of the subtasks involved is subsumption reasoning over each of the facet-specific ontologies. Facet-specific ontologies based on the AIRM need to import the AIRM ontology so that the reasoner can consider the semantics of concepts from AIRM, e.g., domain and range constraints of properties.

The following screenshot from Protégé shows the derived subsumption hierarchy of the classes defined above, arranged together with the classes from the AIRM ontology. For example, defined class

`Aeroplane-with-EnhancedVisionSystem` is subsumed by defined classes `Aircraft-with-EnhancedVisionSystem` and `Aeroplane`. Class `Aeroplane` is in turn subsumed by defined class `Aeroplane-or-Helicopter` which is subsumed by AIRM class `Aircarft` which is subsumed by AIRM class `Vehicle`.

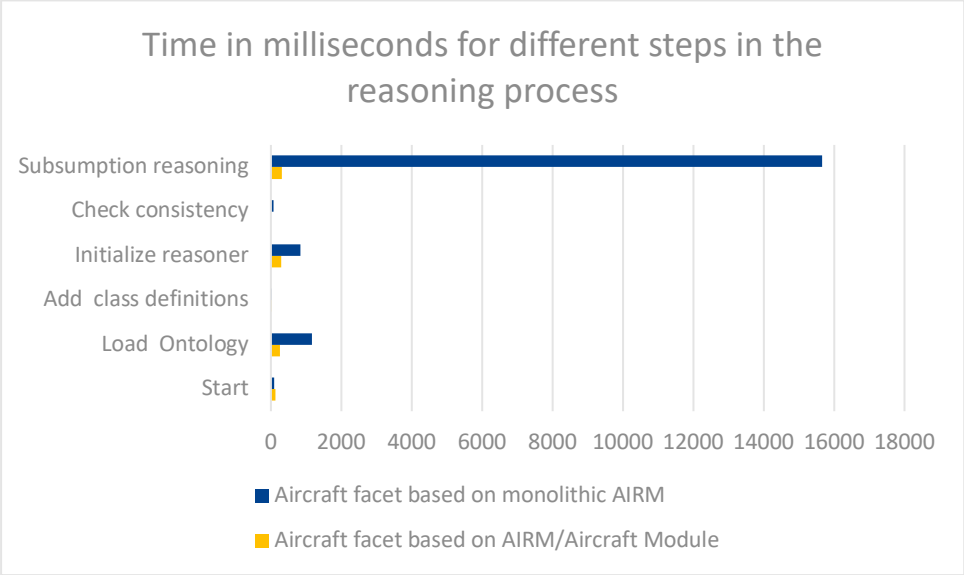


6.2.3 Experiment: Performance benefits of modularizing the AIRM ontology

In most cases, a facet-specific ontology (e.g., for the aircraft facet) only requires some parts of the AIRM ontology. In this experiment we tried to find out whether importing only a subject-field-specific ontology module instead of the monolithic AIRM ontology affects the performance of the reasoning process.

We compared the runtime performance of consistency and subsumption reasoning for the aircraft-facet-specific ontology from above (1) based on the monolithic AIRM ontology and (2) based on the aircraft ontology module. In the first case the facet-specific ontology imports the whole AIRM ontology. In the second case, the facet-specific ontology imports only the aircraft ontology module.

The result of the comparison is shown in the following diagram. Subsumption reasoning is the most time-consuming task. By only importing the aircraft ontology module instead of the whole AIRM, the time for subsumption reasoning can be reduced by orders of magnitude. In this small example the time for subsumption reasoning is reduced from 15.6 seconds to 0.3 seconds, the overall time is reduced from about 18 seconds to 1 second.



6.3 Using OWL for spatial facets

A location individual has a latitude and a longitude. A bounding box (see Section 5.2) can be represented as OWL class expression and is interpreted by the set of location individuals that are within the bounding box's boundaries. The subsumption hierarchy of classes, which represent bounding boxes, corresponds to the bounding box containment hierarchy.

In principle, a general purpose OWL reasoner, like HermiT, can be used to derive such containment hierarchies. The experiments in this section, however, show:

- Reasoning performance depends a lot on the way bounding boxes are represented
- A general purpose OWL reasoner is only suitable for deriving a hierarchy of a small set of bounding boxes (approx. <500 bounding boxes)

6.3.1 Different representations of bounding boxes

We investigate different ways of representing bounding boxes based on the underlying conceptualization of locations that every individual location is specified by a latitude and a longitude. A bounding box is a class of such individual locations. Note, there are other possible ways of conceptualizing bounding boxes, one is discussed in Section 5.2 where bounding boxes are considered as individuals.

In the **variant A of the generic part** of a spatial facet ontology, it is defined that every Location has a latitude and a longitude and that latitude and longitude are functional (i.e., an individual can only have one latitude value and one longitude value):

```
Datatype: xsd:double
DataProperty: hasLatitude
    Characteristics:
        Functional
DataProperty: hasLongitude
    Characteristics:
```

```
        Functional
Class: Location
    SubClassOf:
      (hasLatitude some xsd:double) and
      (hasLongitude some xsd:double)
```

In the alternative **variant B of the generic part** of a spatial facet ontology, we omit any constraints on the hasLatitude and hasLongitude properties. Omitting constraints is often a way to speed up reasoning tasks:

```
DataProperty: hasLatitude
DataProperty: hasLongitude
Class: Location
```

Bounding boxes are represented as defined classes, with equivalent class axioms constraining the hasLatitude and hasLongitude values to a data range with a maximum and a minimum value. Every location within this boundaries is considered a member of the bounding box class.

In **variant I of the bounding box pattern** we use existential quantification (expressed by keyword **some**) to constrain the hasLatitude and hasLongitude values, for example:

```
Class: bb__23_10__53_44__99_54__83_74
    EquivalentTo: (
      hasLatitude some xsd:double[
        >= "23.1"^^xsd:double ,
        <= "53.44"^^xsd:double
      ]
    ) and (
      hasLongitude some xsd:double[
        >= "99.54"^^xsd:double ,
        <= "83.74"^^xsd:double
      ]
    )
    SubClassOf: Location
```

In the alternative **variant II of the bounding box pattern** we use universal quantification (expressed by keyword **only**) to constrain the hasLatitude and hasLongitude values, for example:

```
Class: bb__11_1__66_11__8_37__100_73
  EquivalentTo: (
    hasLatitude only xsd:double[
      >= "11.01"^^xsd:double ,
      <= "66.11"^^xsd:double
    ]
```

```
  ) and (
    hasLongitude only xsd:double[
      >= "8.370000000000001"^^xsd:double ,
      <= "100.73"^^xsd:double
    ]
  )
  SubClassOf: Location
```

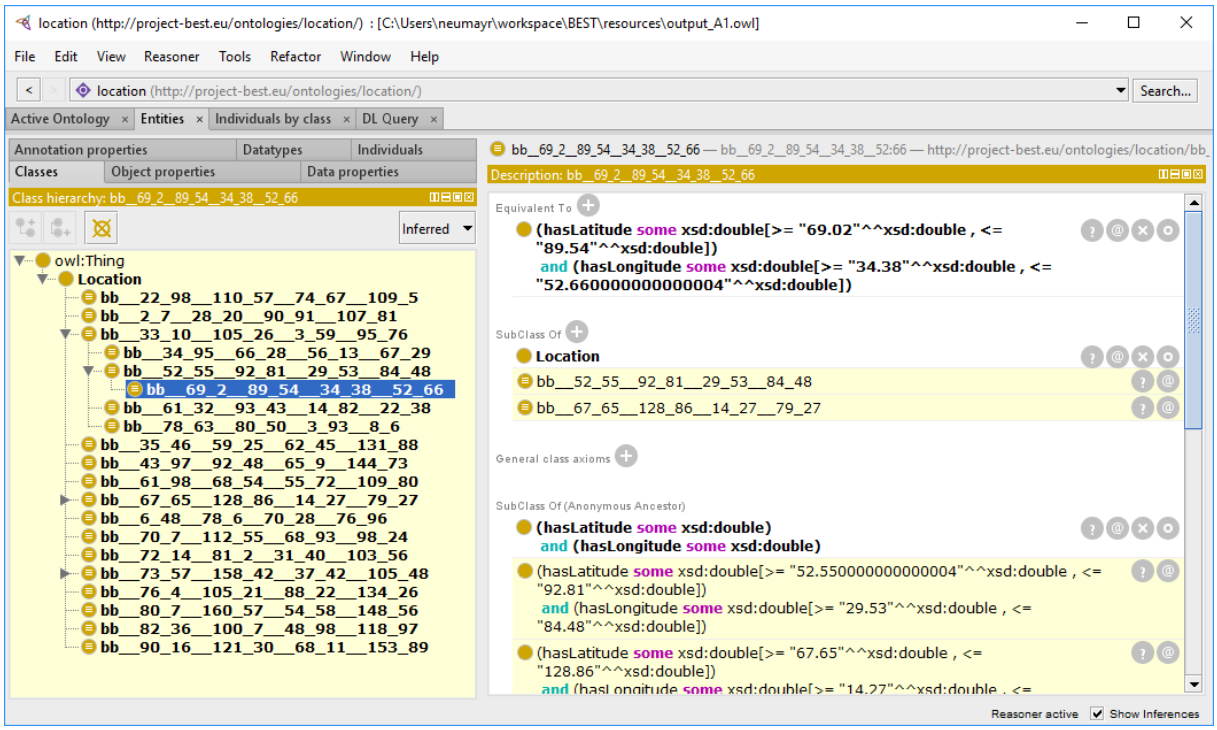
The combination of the two variants (A and B) for the generic part and the two variants (I and II) for the bounding box pattern yield four alternatives for the representation of bounding-box-based location classes:

	Bounding box pattern I	*Bounding box pattern II*
Generic part A	Alternative **A-I**	Alternative **A-II**
Generic part B	Alternative **B-I**	Alternative **B-II**

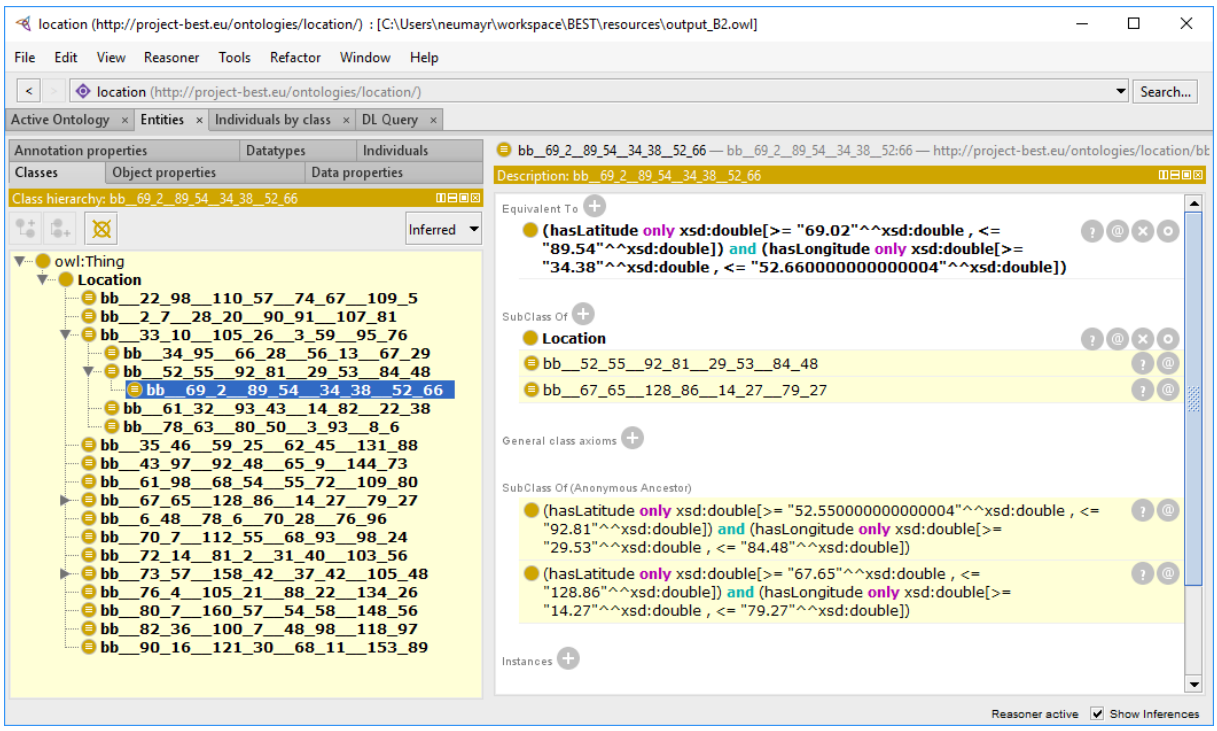
6.3.2 Subsumption hierarchies of bounding boxes

As long as one is only interested in subsumption hierarchies of bounding-box-based location classes and not using negation in class definitions, the different representation choices (A-I, A-II, B-I, B-II) are equivalent with regard to the resulting subsumption hierarchy of bounding box classes.

The following screenshot from Protégé shows the class for the bounding box with latitude=69.02—89.54 and longitude 34.38—52.66 together with subsuming bounding boxes for alternative **A-I**.

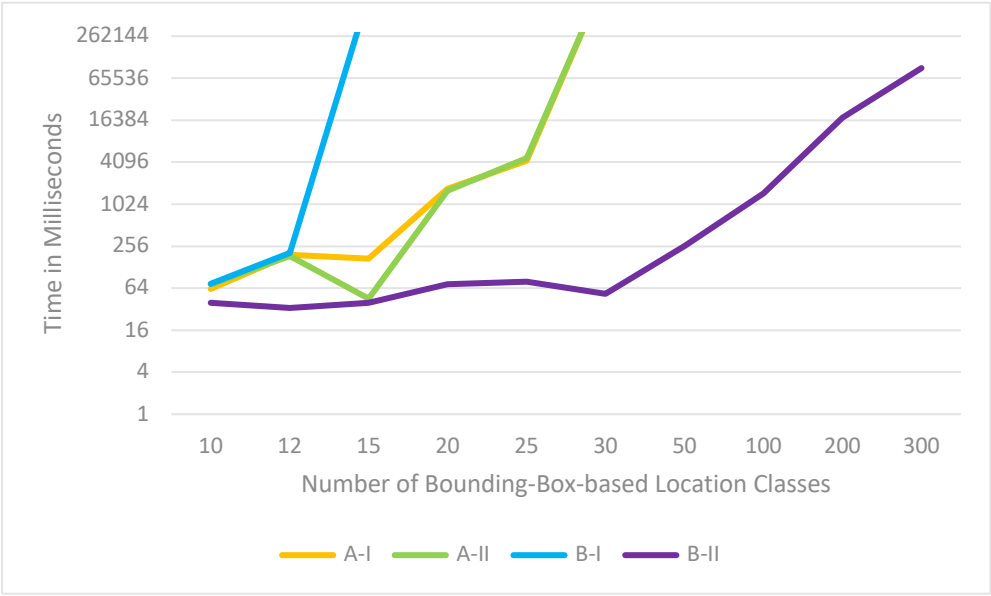


For variant **B-II** (see the following screenshot) the derived subsumption hierarchy is the same:



6.3.3 Experiment: Reasoning with different bounding box represenations

Different representations may result in very different reasoning performance. The following diagramshows that with representation B-I, subsumption reasoning is only feasible for up to 15 bounding-box-based location classes. With representations A-I and A-II, the reasoner can handle up to around 30 such classes. With representation B-II up to 500 classes are feasible.



While variant B-II is the fastest, in the envisioned setting of semantic container management restricting the size of spatial facet ontologies to 300—500 bounding boxes is typically not acceptable. To cover larger sets of bounding boxes external derivation of bounding box hierarchies will be employed. Actually, deriving a subsumption hierarchy of bounding boxes can easily be implemented in Java or using rules expressed in SPARQL. The externally-derived subsumption hierarchy of bounding box classes can be expressed in OWL as asserted subclass hierarchy and be seamlessly used for the definition of semantic descriptions of semantic containers.

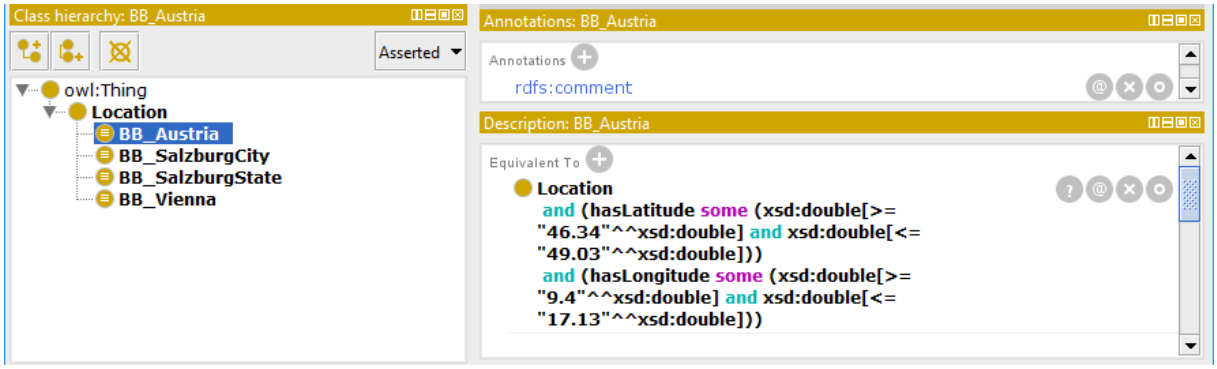
6.4 Materialized and externally-derived subsumption hierarchies

The facet values of semantic descriptions refer to classes in facet-specific ontologies. The subsumption hierarchy of facet values can be derived independently of the subsumption hierarchies of semantic container descriptions. This simplifies and thus speeds up the derivation of subsumption hierarchies of semantic container descriptions. Further, it does not matter whether the facet-specific subsumption hierarchy is derived using a general purpose off-the-shelf OWL reasoner or a special-purpose custom-made engine, as long as the resulting hierarchy is represented as an OWL class hierarchy (with subsumption relationships represented as asserted subclass axioms).

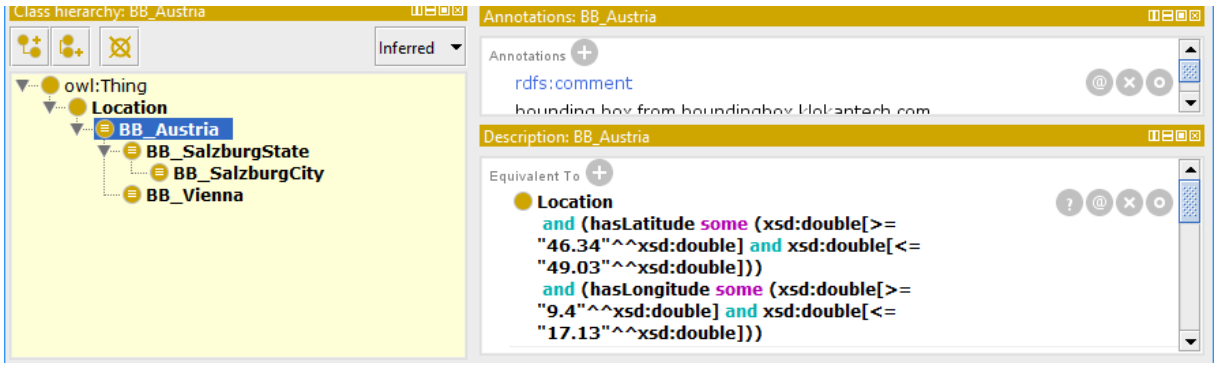
6.4.1 Semantic container ontologies and facet-specific ontologies

In the simple setting of this example, semantic containers have only two facets: time and location. Facet values are defined in facet-specific ontologies ‘Time’ and ‘Location’ and are imported in ontology ‘Multidimensional’ which contain the semantic container descriptions.

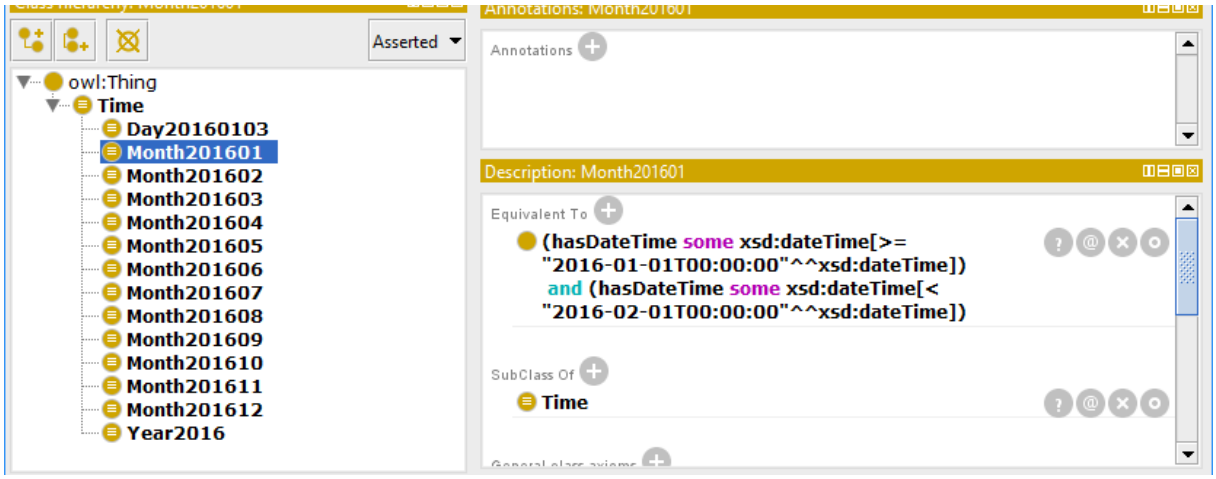
In the simple example, the location facet ontology contains 4 bounding-box-based classes.



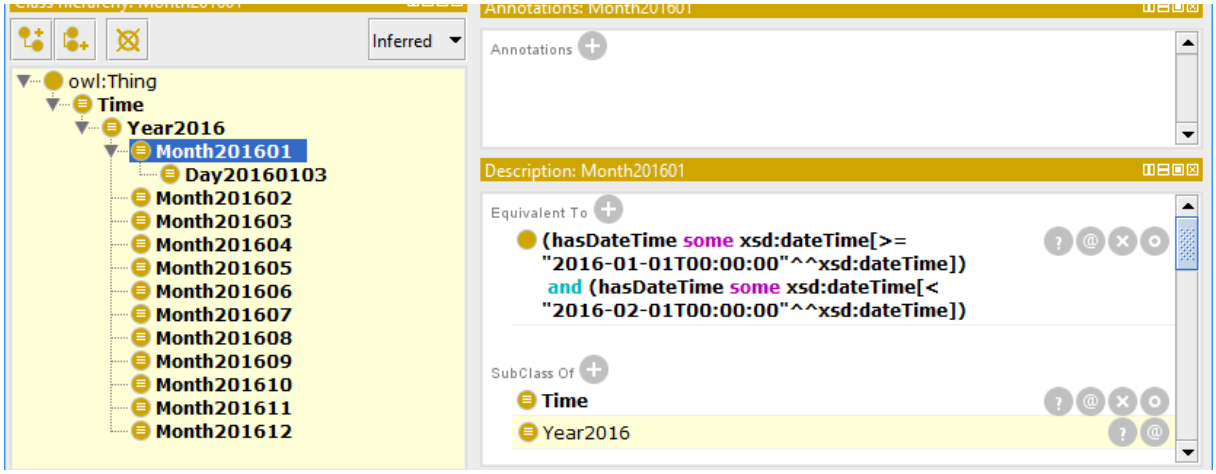
From their definition, the OWL reasoner derives a subsumption hierarchy:



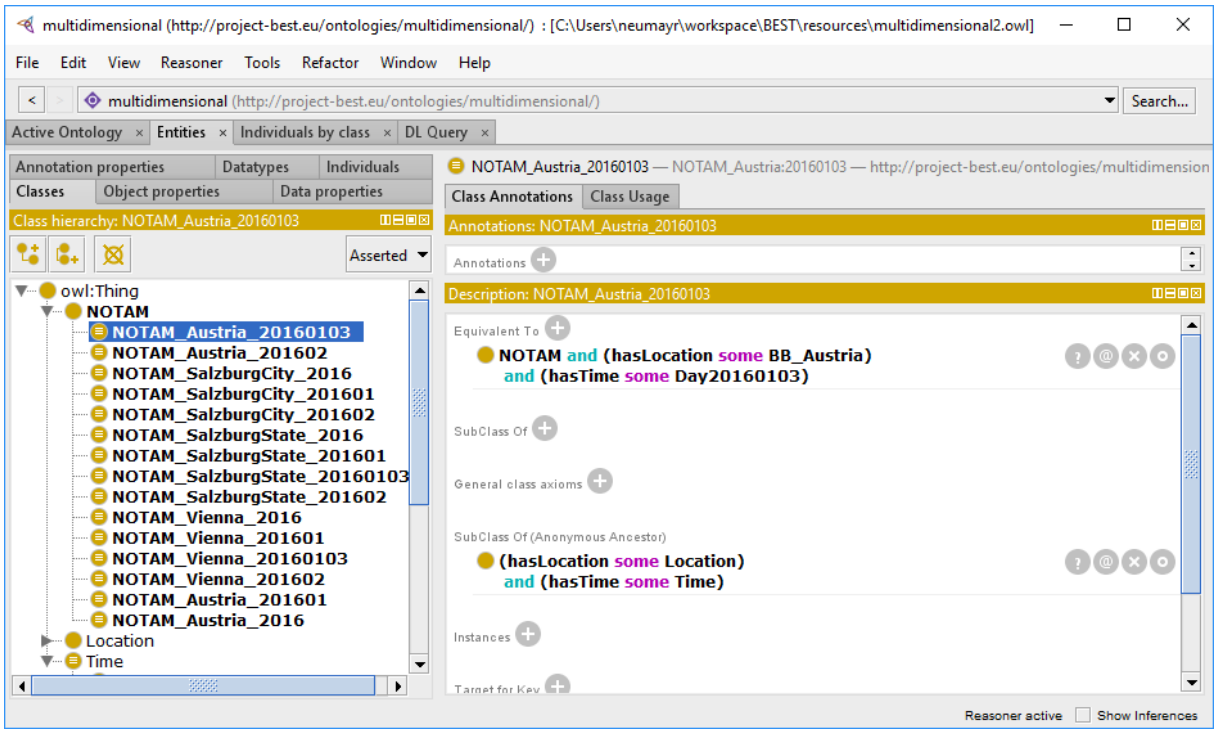
Likewise, the time-facet ontology contains 14 timespan classes:



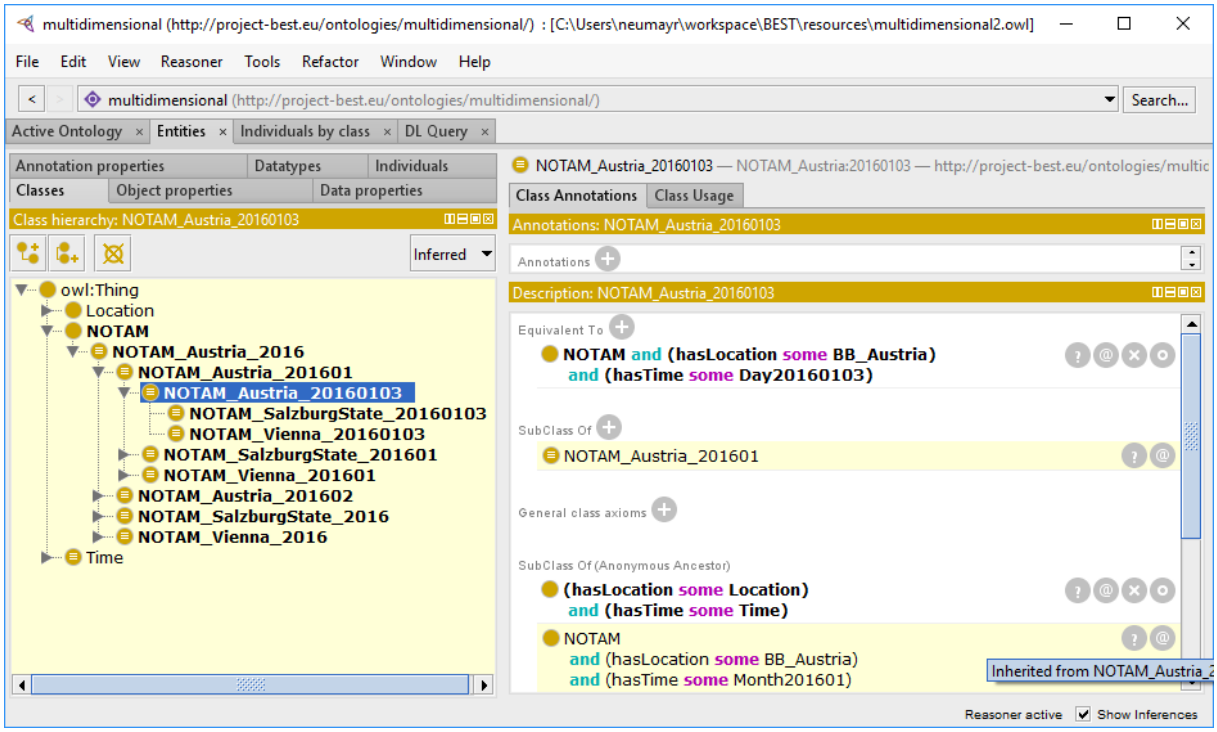
Using the definitions of these timespan classes, the reasoner derives a subsumption hierarchy:



Membership conditions of semantic containers now combine time and location concepts and the type of data item (e.g., NOTAM).



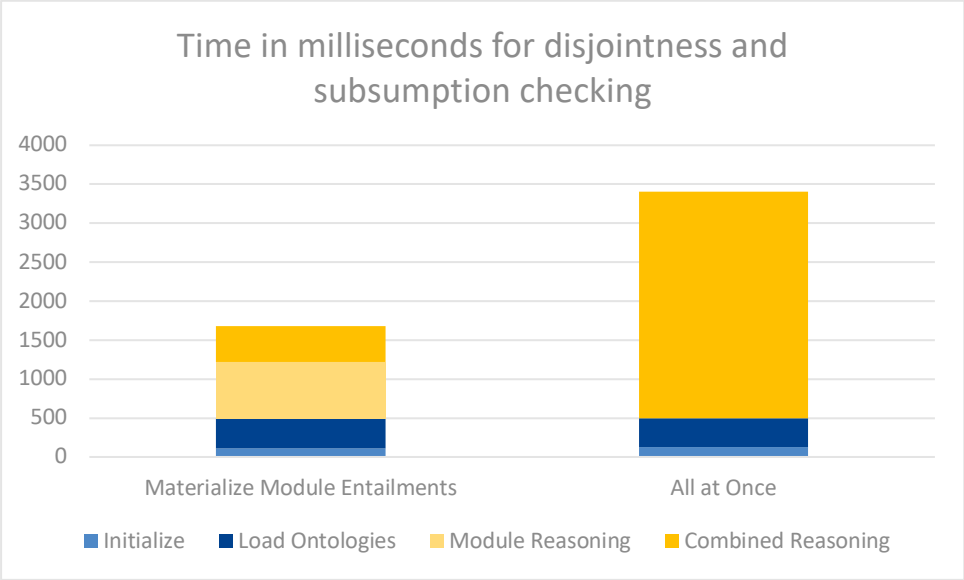
The reasoner derives a subsumption hierarchy of such multi-faceted membership conditions.



6.4.2 Experiment: benefits of materializing subsumption hierarchies

In this experiment we evaluate the benefits of materializing entailments of ontology modules before doing the reasoning over semantic container descriptions (which combines the reasoning results of multiple facet-specific ontologies).

We compared the runtime performance of disjointness and subsumption reasoning for membership conditions (1) based on materialized module entailments (2) without materialized module entailments, giving the reasoner everything at once. The result of the comparison is shown in the following diagram. Even with such a small example, splitting up the reasoning process into smaller chunks increases performance considerably. In this example, the overall time was reduced from 3.4 seconds to 1.7 seconds.



6.5 Matching information need and semantic containers

The most important reasoning task for semantic container discovery is to find, given a (huge) set of elementary semantic containers and an elementary information need, the most specific subsuming semantic containers. Matching composite information needs should be broken down to elementary ones. We assume that the set of semantic containers does not change very often (their content is highly dynamic but their membership condition is stable) but there a lot of different information needs emerging dynamically. In such a setting, this reasoning task should be splitted in two parts. First, the subsumption hierarchy of the membership conditions of the semantic containers is derived. Second, the direct subsumers of the information need (which is represented as OWL class expression) are derived.

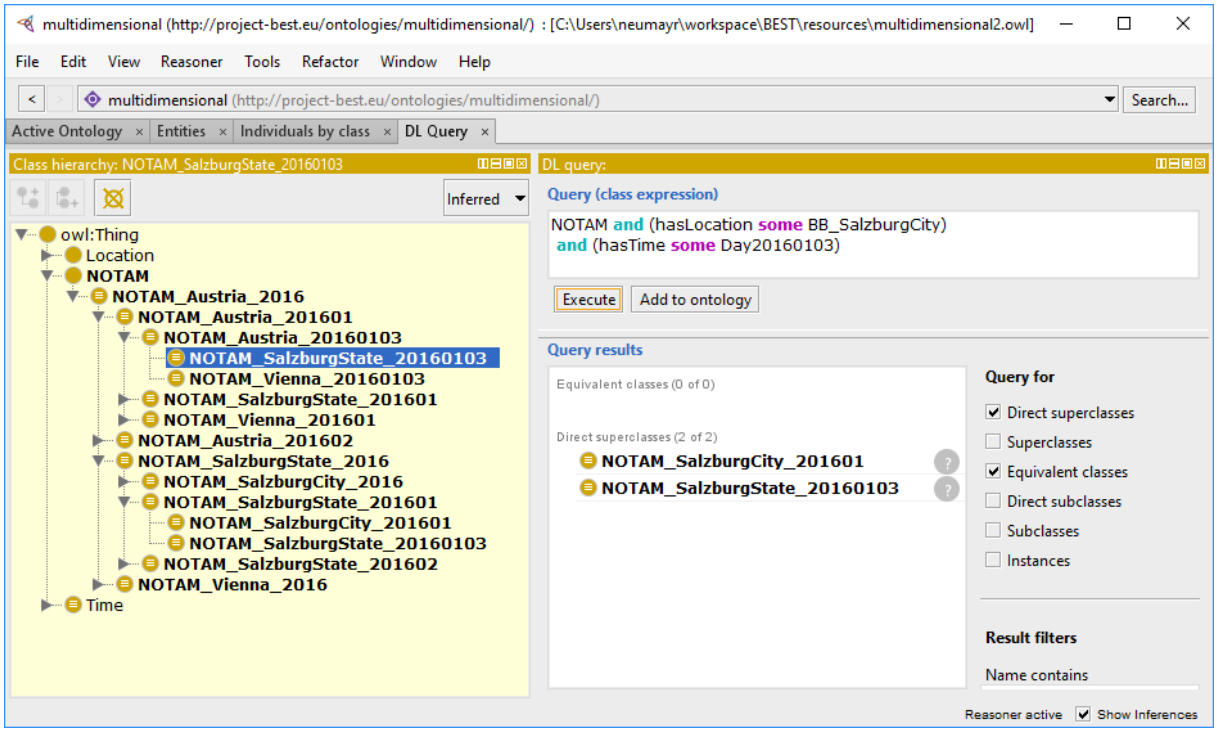
6.5.1 Information needs represented by OWL class expressions

Information needs are expressed similarly to semantic container descriptions. In contrast to the semantic description of container desciptions which are are asserted as defined class in the semantic

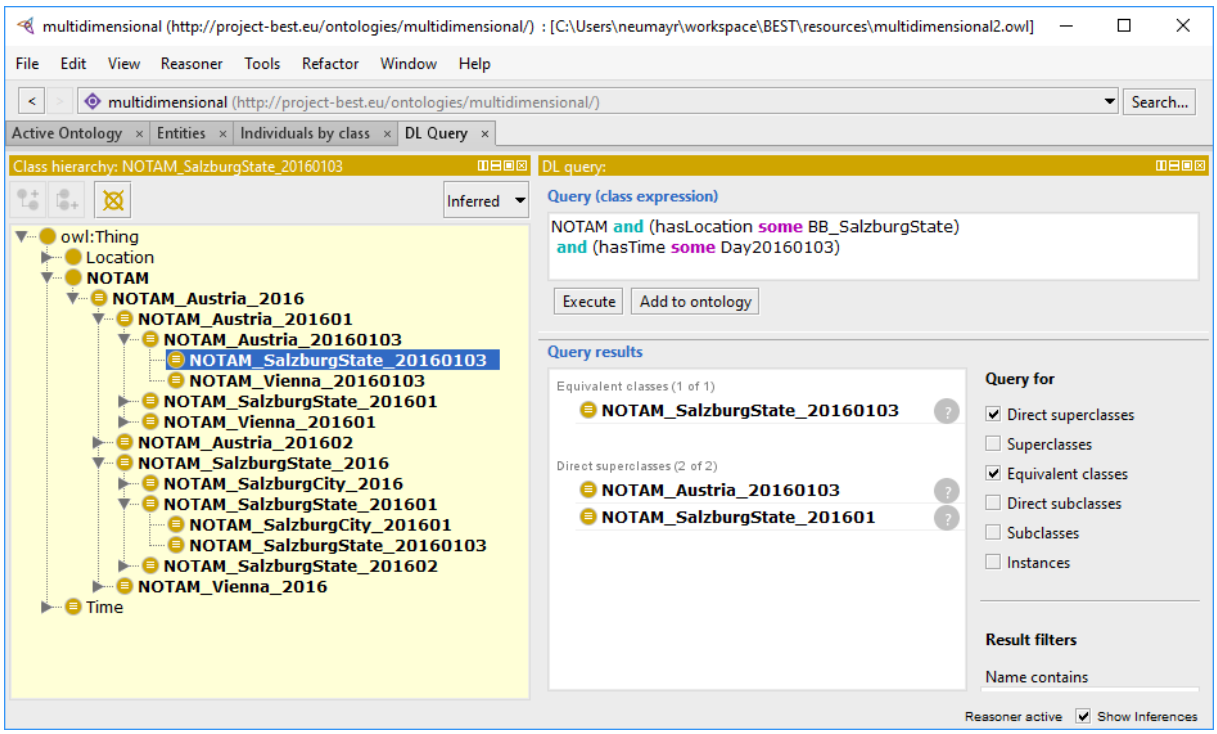
container ontology, information needs are only expressed as class expressions which are used to query the ontology for direct superclasses and equivalent classes (fully matching semantic containers). Direct superclasses represent most-specific subsuming containers. Equivalent classes represent fully matching semantic containers.

The following two Protégé screenshots exemplify how to query an ontology for direct superclasses and equivalent classes. Note, the example has been modified from the previous section and does not contain the class `SalzburgCity_20160103` anymore, in order to show an example where there is no full match).

In the first screenshot, an information need represented by class expression `NOTAM and(hasLocation some BB_SalzburgCity) and (hasTime some Day2016-0103)` has two direct subsumers (most-specific subsuming containers), namely `NOTAM_SalzburgCity_201601` and `NOTAM_SalzburgState_20160103`.



In the second screenshot, an information need represented by class expression `NOTAM and (hasLocation some BB_SalzburgState) and (hasTime some Day20160103)` has two direct subsumers (most-specific subsuming containers), namely `NOTAM_Austria_20160103` and `NOTAM_SalzburgState_201601` and one equivalent class (full match), namely `NOTAM_SalzburgState_20160103`.



6.5.2 Experiment: Scalability of semantic container discovery

In this example we investigated how semantic container discovery scales with the size of the semantic container repository, i.e., with the number of semantic containers.

For the experiment, semantic descriptions and information needs are generated randomly, each taking one facet-value from each of three facet-specific ontologies. Each generated facet-specific ontology has 364 classes (possible facet-values) arranged in a subclass hierarchy with a depth of five. In this experiment, these subclass hierarchies form a tree. As shown in the previous experiment "Benefits of materializing ontology module entailments" subsumption hierarchies of facet-specific ontologies should be precomputed and for semantic container reasoning we only consider the precomputed subsumption hierarchy. Combining these facet-values give 48228544 possible membership conditions.

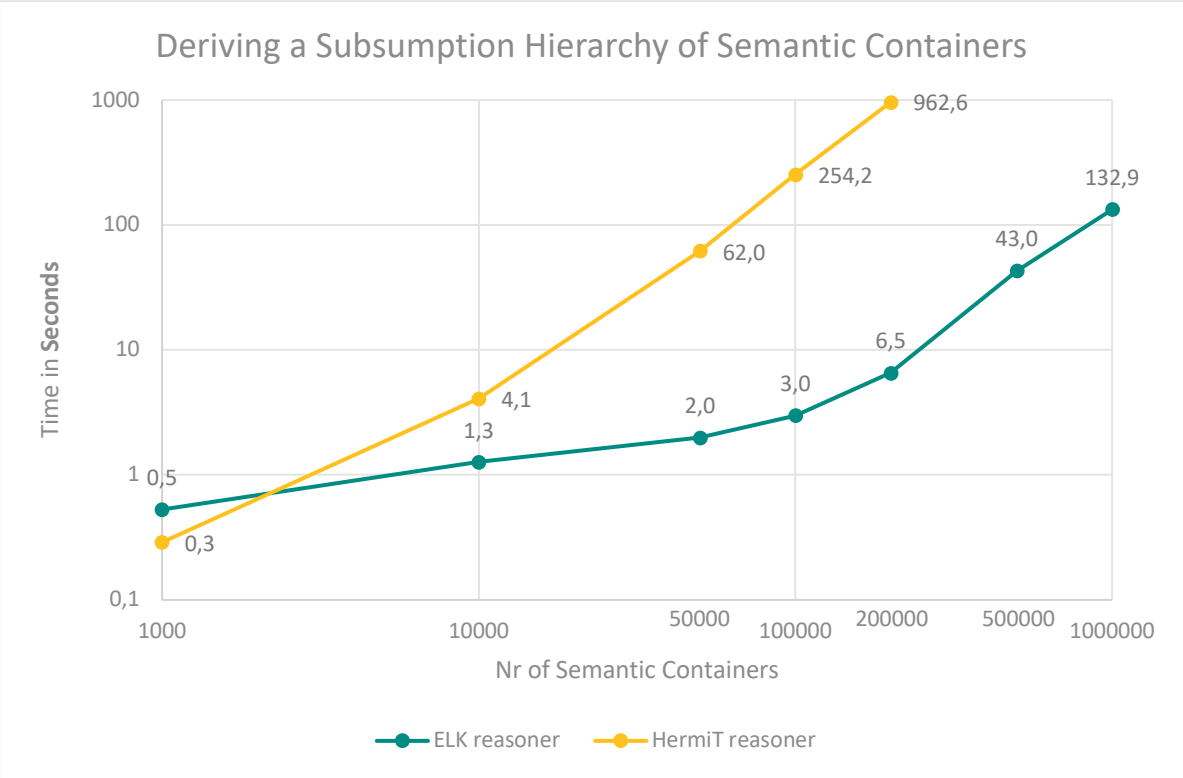
For this experiment, in contrast to previous experiments we choose a more 'reasoner-friendly' representation of semantic containers. With this representation, membership conditions are represented as intersections of facet-values. The generated OWL ontology thus only uses a small subset of the constructs of OWL and is in the OWL EL profile (see W3C 2012c). OWL EL is a profile especially well-suited for very large ontologies with rather simple class definitions and allows for very efficient subsumption reasoning. ELK[6] is an open-source reasoner for OWL 2 EL ontologies and supports very fast subsumption reasoning. OWL EL, however, trades off certain features, such as disjunction, which

[6] https://www.cs.ox.ac.uk/isg/tools/ELK/

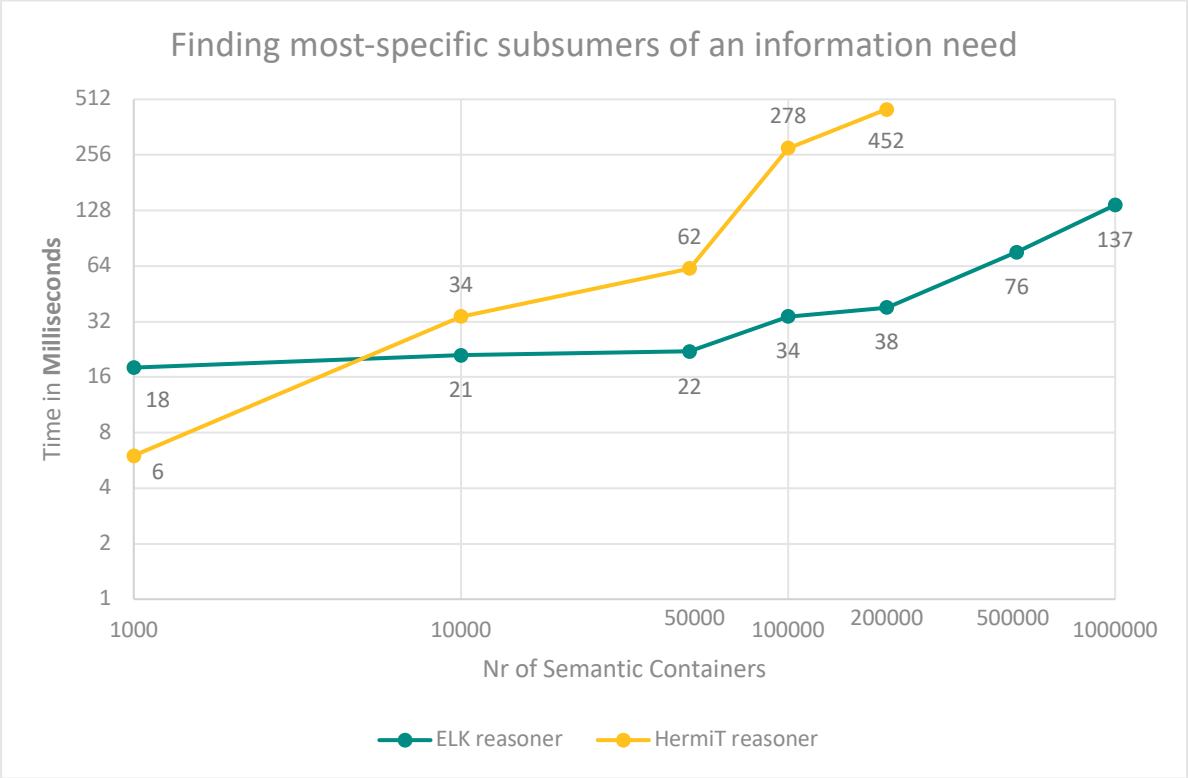
could be useful in semantic container description. In cases where such features are needed in the semantic container approach, custom (rule-based) reasoning must be employed, which should work efficiently in the limited cases where needed, or an alternative representation must be chosen. For example, the flight information region for the whole of Germany then cannot be defined as the union of smaller flight information regions but the smaller flight information regions must be modelled as subclasses of the Germany region. Also, heterogeneous composite containers cannot be represented using disjunction, but disjunction must be handled separately through rule-based reasoning, e.g., using SPARQL-based rules, and specific facets for heterogeneous composite containers.

We ran the experiment with different numbers of semantic containers (1000, 10000, 50000, 100000, 200000, 500000, 1000000 semantic containers) expressed as OWL classes and with two different OWL reasoners (HermiT and ELK). In this experiment, ontologies (sets of semantic container descriptions expressed as OWL classes) were generated in Java using the OWL API. The generated ontologies were written to OWL files and imported with Protégé, starting reasoners HermiT and ELK from within Protégé. For the experiments regarding most-specific subsumers, we used Protégé's "DL query" tab. The measured times are taken from Protégé's output to the command line. The measurements should be considered with care and are only meant to give some first indication about performance characteristics.

The first reasoning task was to derive the subsumption hierarchy for the set of semantic containers. As this experiments shows, the first task is quite expensive yet scales up to one million semantic containers on a single desktop machine (with ELK taking 133 seconds for the derivation). The more general HermiT reasoner ran out of memory with 500k semantic containers. Using ELK, it takes only 6.5 seconds to derive the subsumption hierarchy of 200000 containers while HermiT already takes 1000 seconds. For smaller sets of semantic containers (1000—10000 containers) both reasoners perform well, taking up to 4 seconds for subsumption reasoning.



The second reasoning task was to find for each of four generated information needs (each expressed as an OWL class expression) the most specific subsuming containers (i.e., the direct subsumers of the OWL class expressions). Based on the derived subsumption hierarchy, finding the direct subsumers of a class expression is very fast with both reasoners: Given a subsumption hierarchy of 200000 containers and an information need it took HermiT 452 milliseconds and ELK only 38 milliseconds to get the set of most specific subsuming containers. Even with a million of semantic containers, it took ELK only 137 milliseconds to get the set of most-specific subsumers.



6.6 Conclusions

We have conducted a set of experiments to analyse the runtime behaviour (performance) of general-purpose OWL reasoners for the reasoning tasks and kind of knowledge relevant for BEST. We came to the following conclusions:

- Modularizing a large domain ontology (like the AIRM ontology) is beneficial from the viewpoint of more specialized ontologies (like an aircraft-facet specific ontology for semantic container description) which use only some parts of the large domain ontology. Importing only the used modules can improve reasoning performance by orders of magnitude (see Section 6.2.3).
- Splitting complex concepts (like the membership condition of a semantic container) into orthogonal facets (modules) allows to reason over each facet independently. The experiment in Section 6.4.2 shows that in such cases it is, with current technology, better not to leave optimization decisions to the general purpose OWL reasoner but to explicitly encode in the system which intermediate reasoning results to materialize before continuing the reasoning process. We expect that off-the-shelf reasoners will in the future provide better built-in support for such modularized reasoning, relieving system developers from the need to hard-code such optimization techniques.
- The performance of a semantic reasoner depends on the way knowledge is represented and depending on which ontology language constructs are used. It is often beneficial to not express all constraints that exist in a domain, but rather only the knowledge that is necessary for a certain reasoning task (see Section 6.3.3). For very large ontologies (with large numbers

of classes), restricting class definitions to the OWL 2 EL profile is worthwhile. Special OWL reasoners for the EL profile (like the ELK reasoner) allow to realize huge performance gains in subsumption reasoning (see Section 6.5.2).

- "One-size fits all" does not apply to semantic reasoning: general purpose OWL reasoners like HermiT are not the perfect solution for specific reasoning tasks, such as the derivation of containment hierarchies of bounding boxes. Special purpose engines (external reasoners) should be applied for such specific reasoning tasks. Yet, the outcome of these specific reasoning tasks can be expressed (materialized) as an OWL subclass hierarchy (with asserted `SubclassOf` axioms) and used by a general purpose reasoner (see Section 6.3.3).
- The key reasoning task, namely matching information needs with membership conditions of semantic containers to find the most specific subsuming containers, scales well with the size of the repository (see Section 6.5.2).

7 Administrative metadata

The importance of metadata in ATM has been recognized before. Several attempts exist to define metadata for the aeronautical domain. The Aeronautical Data Quality (ADQ) Implementing Rule issued by the European Commission identifies the need for a minimum set of metadata (Eurocontrol 2010). Metadata is identified as a driver for interoperability since it allows to find data and to make decisions based on the associated metadata which, e.g., can indicate the quality or relevance of the data by describing temporal and geographical facets (Porosnicu 2013).

Considering the ISO 19100 standards for geographical information and geomatics, work has been ongoing to define and standardize metadata. This effort resulted in the development of the ADQ Metadata profile based on guidance and requirements from the Open Geospatial Consortium (OGC) (Wilson 2011). The definition of the ADQ Metadata profile is still ongoing but nevertheless provides valuable input for the presented metadata description. Note however that a complete coverage of these standardization efforts is not the goal of the presented approach.

The first step in realizing the benefits of semantic labels by employing ontologies is to define the types of metadata that should be covered. As stated before, the semantic description of a container can be used to describe a data set regarding the data source, the membership conditions, and its quality and freshness. Depending on the concerned aspect, this can be achieved by annotating properties or by assigning concepts from the AIRM ontology.

We distinguish two groups of metadata, namely administrative metadata and descriptive metadata (the semantic description's membership condition). The former provides information supporting the management of the data containers such as technical, quality, and provenance information. The latter is used to describe the content of the data set, thereby answering the question what data items are in the data set. The descriptive metadata corresponds to the semantic description's membership condition. The main purpose of descriptive metadata is to support the discovery of the data containers that satisfy a consumer's information need, whereas administrative metadata can be considered as additional selection criteria.

Consider, for example, the semantic containers in Figure 19, which contain weather forecasts (TAF data items) from the Deutsche Wetterdienst (DWD) relevant for the route from Munich to Frankfurt (MUC-FRA route) with overlapping time periods covered by the weather forecasts. The first container covers a time period from 23-26 February 2017, the second container covers a time period from 24-26 February 2017, meaning that the data containers contain all DWD TAFs relevant for the specified route in the specified time period. Suppose a consumer is interested in the forecasts for the MUC-FRA route on the 25 February 2017. In that case, the contents of both data containers in Figure 19 are relevant. The administrative metadata can then be used to select a particular container.

7.1 Technical metadata

Technical metadata describes technical characteristics of a semantic container's data set and comprises (1) data format, (2) encoding, (3) volume, (4) location, and a (5) checksum. Possible data formats include, e.g., XML and JSON. Encoding describes the character encoding, e.g., UTF-8. Both, data format and encoding are relevant for further processing especially if the data set is used as input for

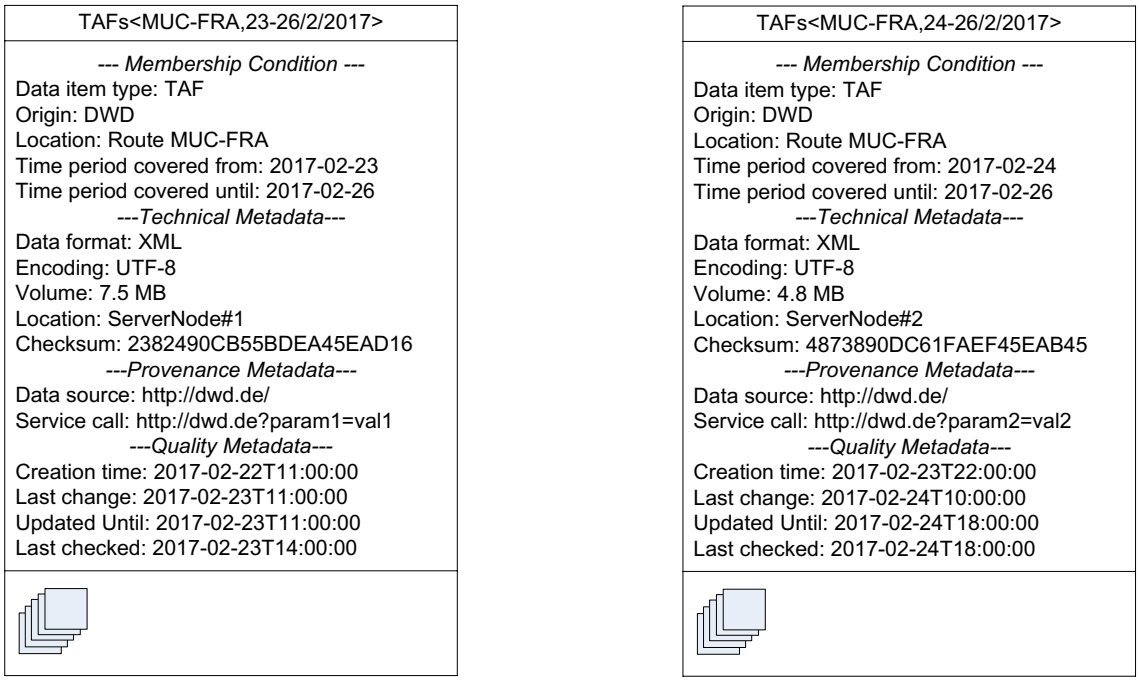


Figure 19. Administrative metadata of semantic containers that contain TAF data items

other services. The data volume represents the size of the data set and location refers to the physical location of the data set; BEST Deliverable 2.2 will investigate allocation of semantic containers. The checksum serves to verify the integrity of the data set after transmission. In addition, technical metadata describe whether the refresh is conducted based on polling, a publish/subscribe architecture, or any variation of these approaches. Detailed descriptions on how this can be established are provided by Patrick et al. (2003).

Listing 5 shows a possible RDF representation of the technical metadata of the semantic containers in Figure 19. In this representation, the technical metadata become RDF properties of the data containers. These properties are not considered by the OWL reasoner, i.e., during the faceted ontology-based discovery of data containers, but become selection criteria for semantic containers once the relevant semantic containers have been identified or provide valuable information for post-processing, e.g., conversion into another format and encoding.

7.2 Provenance metadata

Provenance describes the data origin and captures performed data processing steps. This is captured for every secondary data container by specifying its (1) data source and (2) a data service call which together produce the container's data set. The data service call consists of the executed service with optional parameters and credentials that were used to retrieve the data. The example in Figure 19 serves for illustration purposes only. The service call could be a complex data object with multiple, very complex parameters. We will investigate provenance more closely in Deliverable 2.2 along with the allocation of semantic containers.

```
:Container_1 :dataFormat :XML ;
             :encoding :UTF_8 ;
             :volume 7.5 ;
             :location ServerNode_1 ;
             :checksum "2382490CB55BDEA45EAD16"^^xs:string .

:Container_2 :dataFormat :XML ;
             :encoding :UTF_8 ;
             :volume 7.5 ;
             :location ServerNode_2 ;
             :checksum "4873890DC61FAEF45EAB45"^^xs:string .
```

Listing 5. Technical metadata of the semantic containers in Figure 19

In addition to the already introduced metadata, further metadata might be useful when selecting a data container from a list of suitable containers. For example, the organization which provides the data container can be considered. A consumer may, for example, have more trust in data provided by some organization over others. Furthermore, the role of the information service providers can be incorporated in the metadata in order to be able to distinguish between data originators and processors. Additional aspects are the access limitations, i.e., if the data container contains restricted or classified information, and the access condition, i.e., whether the data container can be used freely or if the usage is charged.

7.3 Quality metadata

In a decentralized marketplace of information services the description of data sets also includes a description of the data quality. Data quality is a multidimensional concept which varies depending on the stakeholders assessing the data quality (Pipino et al. 2002); various metrics for data quality exist. We focus on a subset of four metrics, namely (1) population completeness, (2) relevance, and (3) timeliness. Concerning population completeness and relevance, data sets provided by SWIM information producers contain data items which fulfil predefined conditions expressed using specific temporal, spatial or semantic concepts. Since these conditions represent membership conditions we expect that the provided data set is complete with respect to the described conditions, e.g., if the content of a data set is described as the DNOTAMs for airplanes in Austria during January 2017, then it is expected that this data set contains all relevant airplane DNOTAMs for Austria during January 2017. Assuming that one hundred percent recall is provided, i.e., all relevant data items are retained, even if there are superfluous data items included; the amount of superfluous data may vary between producers. In this respect, a data container's precision can only be measured relative to another data container: Assuming one hundred percent recall, the data container with fewer data items more precisely satisfies the same information need.

The crucial metric for our approach is timeliness, which considers the freshness of the data container and the freshness requirement of the information need. To allow SWIM information consumers to assess the timeliness of a data container we consider the (1) creationTime of data items and the (2) lastChange, (3) updatedUnil, and (4) lastCheck timestamp of a data container. The creationTime is

```
:Container_1 :creationTime 2017-02-22T11:00:00"^^xs:dateTime ;
             :lastChange 2017-02-23T11:00:00"^^xs:dateTime ;
             :updatedUntil 2017-02-23T11:00:00"^^xs:dateTime ;
             :lastChecked 2017-02-23T14:00:00"^^xs:dateTime .

:Container_2 :creationTime 2017-02-23T22:00:00"^^xs:dateTime ;
             :lastChange 2017-02-24T10:00:00"^^xs:dateTime ;
             :updatedUntil 2017-02-24T10:00:00"^^xs:dateTime ;
             :lastChecked 2017-02-24T18:00:00"^^xs:dateTime .

:InformationNeed_X rdfs:subClassOf Container_1 .
:InformationNeed_X rdfs:subClassOf Container_2 .
```

Listing 6. Quality metadata of the semantic containers in Figure 19 along with inferred knowledge (in italics) which states that both semantic containers satisfy a given information need `InformationNeed_X`

```
SELECT ?container WHERE {
  :InformationNeed_X rdfs:subClassOf ?container .
  ?container :lastChecked ?lastChecked .
  FILTER(?lastChecked >= "2017-02-24T08:00:00"^^xs:dateTime)
}
```

Listing 7. SPARQL query to retrieve all containers defined in Listing 6 that satisfy a given information need (previously determined by reasoner) and then filter the relevant containers by their lastChecked property

?container
:Container_2

Table 4. Result of the SPARQL query In Listing 7 applied on the RDF data in Listing 6

the time when the data item was first published in a primary data container. The lastChange timestamp is the time when the data set was last updated with new data; lastChange corresponds to the creationTime of the last-added data item. The updatedUntil timestamp guarantees that all data items with a creationTime prior to updatedUntil have been considered. For example, a SWIM consumer can be sure that when a data container is labelled with the updatedUntil timestamp of January 5th that all the available data items up to this point were considered during the creation of the data set.

New data items are created and published via primary data containers. Maintaining the timeliness of secondary data containers requires checking the source container for updates periodically according to a specified interval (refreshInterval). The lastCheck timestamp indicates when the last check for updates of the source container was performed. Secondary data containers may be only kept up-to-date until a certain point (refreshUntil).

Listing 6 shows a possible RDF representation of the quality metadata for the semantic containers in Figure 19. Again, the quality metadata are expressed as RDF properties such as `creationTime`, `lastChange`, `updatedUntil`, and `lastChecked`. The range of these properties is `xs:dateTime`, an standard XML data type for the representation of timestamps. Notice the triples in italic font, these are triples derived by the OWL reasoner, stating that the two containers `Container_1` and `Container_2` both subsume the given information need represented by the `InformationNeed_X` class.

We can then query the RDF data as shown in Listing 6 using the SPARQL query in Listing 7 in order to retrieve all semantic containers that satisfy `InformationNeed_X` – information previously derived by the OWL reasoner using subsumption matching – and also present specific characteristics with respect to the `lastChecked` property. Only those semantic containers that satisfy the given information and were last checked for updates after 24 February 2017, 8:00 am, are considered as a result for this query.

8 Composition of semantic containers

In previous sections, we considered semantic containers with a single type of data item, e.g., METAR containers, DNOTAM containers. In this section, we present considerations about composite semantic containers, which many practical operational scenarios require (see Deliverable 3.1).

8.1 Types of semantic containers

We distinguish elementary and composite semantic containers (Figure 20). Elementary semantic containers consist of data items. These data items are either entities from some aeronautical ontology or annotations. Each annotation refers to an entity of a specific type, e.g., a NOTAM importance refers to a NOTAM data item. Figure 21 illustrates the abstract data model of elementary semantic containers: Each elementary container refers to an entity type, an elementary container of annotations (*AnnotatedElementaryContainer*) also refers to an annotation type. Composite semantic containers consist of multiple elementary containers or other composite containers.

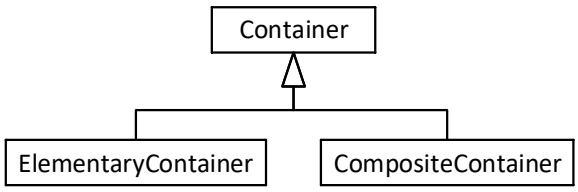


Figure 20. Types of semantic containers: elementary and composite containers.

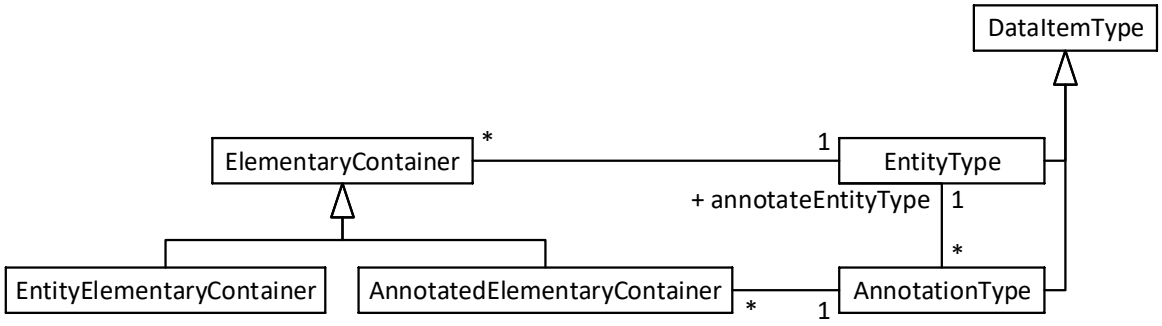


Figure 21. Abstract data model of elementary semantic containers

We distinguish two types of composite semantic containers (Figure 22): homogeneous and heterogeneous. Homogeneous composite containers comprise several other semantic containers as components. Each component container of a homogeneous composite container contains data of the same data item type or homogeneous composite containers of the same data item type. For example, a homogeneous composite METAR container contains several semantic containers of the METAR entity type. A heterogeneous composite container, on the other hand, has component containers of

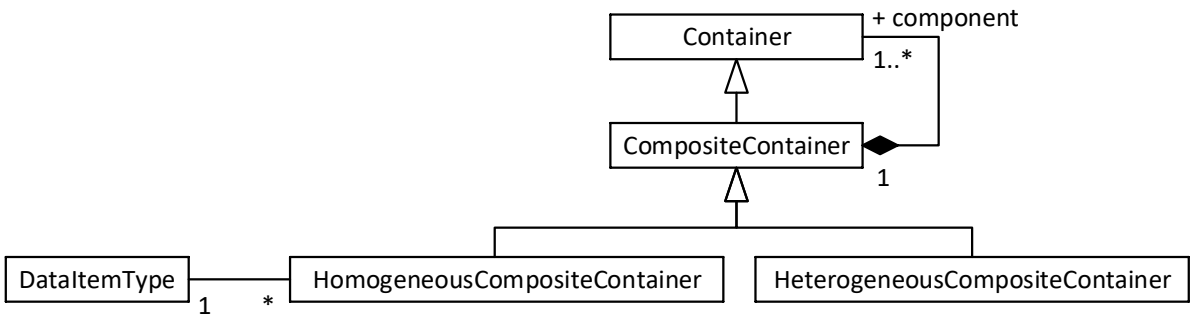


Figure 22. Abstract data model of composite semantic containers

different data item types. For example, an EFB container comprises component containers of the NOTAM, METAR, TAF, etc. types. The different types of composite containers are used for different operational scenarios, as we briefly discuss in the following.

8.2 Combining fine-grained semantic containers

Some semantic containers will be composed of several smaller containers of the same data item type which contain a subset of the required data items. For example, multiple semantic DNOTAM containers, one per flight route segment, compose a homogeneous composite semantic container containing all the relevant DNOTAMs for the whole flight.

The facet values of a composite semantic container that is just a combination of several smaller fine-grained containers is the union of the component containers' facet values. No additional facets must be attached to the composite container. The quality metadata of the composite container contain the minimum quality characteristics from the component containers.

8.3 Composition of weather, flight, and other aeronautical data

Typical operational scenarios that require composite semantic containers are sector-less air traffic control and flight rerouting (cf. BEST Deliverable 3.1). Operating a flight requires different types of data items such as DNOTAMs, weather data and forecasts, and flight plans. In sector-less air traffic control, controllers are provided with a composite container that contains all the DNOTAMs, METARs and TAFs, etc. that are relevant for a specific flight. Similarly, for different routes, different composite containers with all the data items relevant for conducting a specific flight on that route can be pre-computed so that, in case of rerouting, the required information is readily available.

A heterogeneous composite semantic container supports provisioning of all the different types of data relevant for a particular flight. A heterogeneous composite container can contain all DNOTAMs, METARs, TAFs, flight plans, etc. required for conducting a particular flight. The resulting heterogeneous composite containers "inherit" the facet values from the component containers as semantic descriptions. These inherited facet values, however, must be indicated to stem from a container of a certain data item type, e.g., DNOTAM, METAR, in order to be able to correctly interpret the semantics of the facet values in the particular context. In that case, the data item types of the component containers become the facets of the composite container, the facet values are then the disjunction of the membership conditions of the component containers of the respective data item type.

8.4 Composition with value-added data

In a service-oriented SWIM architecture, data providers annotate existing data items with supplemental information and thus create value-added data. A value added can be the annotation of priority of individual data items.

Semantic containers with value-added data supplied by information service providers are realized as heterogeneous composite containers. These containers consist of elementary or composite containers of a particular entity type along with containers of a particular annotation type. For example, a SemNOTAM-filtered container comprises an elementary container with DNOTAMs along with a semantic container of annotations that asserts the priority of each DNOTAM as an annotation object.

The annotations are key-value pairs where the key is a data item identifier, e.g., the identifier of an individual DNOTAM or METAR, and the value is some annotation, e.g., a “high”, “medium”, or “low” priority.

8.5 Combining semantic containers from different sources

In a decentralized SWIM setting, where different information service providers co-exist and complement each other, composite semantic containers also merge containers from different sources, e.g., NOTAMs issued by the FAA and GroupEAD. In that case, preservation of provenance information of the various component containers becomes especially important. In particular, the data quality and freshness of the semantic containers from the different providers may differ. The composite container then has freshness and data quality properties that correspond to the minimum freshness and the minimum data quality, respectively, of the component containers. Deliverable 2.2 will investigate representation of provenance of semantic containers.

9 Lessons learned

In this deliverable we have presented the semantic container approach and investigated fitness of common semantic web technologies for realizing this approach. We have built on the results of Deliverable 1.1 and considered the use case scenarios laid out in Deliverable 3.1. Given these results, we summarize the lessons learned as follows:

- Based on the functional requirements and backed by experimental evaluation, we conclude that semantic web technologies are in general well-suited for realizing the semantic container approach.
- With respect to technology choice, a mix of semantic web technologies seems appropriate. Different aspects of semantic container management and discovery require different degrees of expressiveness and, therefore, must be realized with different technologies.
- Different OWL profiles, in varying degrees, trade off expressiveness for the sake of decidability, scalability, and performance. In the narrow context of semantic container management and discovery, custom-made reasoners and rule-based reasoning could efficiently emulate the required features dismissed by the various OWL profiles.
- In general, technology choice is always a trade-off between expressiveness, scalability as well as development and maintenance effort. For example, while RDF places no restrictions on expressiveness, the corresponding reasoning tasks must be solved by custom-built software which also results in potentially higher maintenance effort. OWL, on the other hand, trades off expressiveness for general decidability and scalability but off-the-shelf reasoners then support many tasks that would otherwise have to be custom-built.
- The AIRM ontology developed in WP1 is a well-suited foundation for defining facet-specific ontologies which are in turn used for defining membership conditions of semantic containers. Importing only relevant AIRM ontology modules instead of importing the monolithic AIRM ontology not only better supports knowledge organization but also helps to speed up semantic reasoning.
- Modularization of the ontologies for semantic container description into orthogonal facet-specific ontologies allows to splitting the reasoning process into tractable subtasks. Subsumption hierarchies are derived independently for each facet and are then combined for organizing semantic containers in derived subsumption hierarchies and for matching information needs with available semantic containers. Additionally, this kind of modularization facilitates the use of different reasoning techniques for different modules, e.g., for deriving a subsumption hierarchy of a spatial facet it is more efficient to use a special-purpose reasoner instead of a general purpose OWL reasoner.
- Speed and scalability of semantic reasoning depends on the complexity of the ontology, especially on the used language constructs. For large-scale reasoning, e.g, subsumption reasoning over a million semantic containers, one has to invest considerable thought into representations that allow for efficient and scalable reasoning. Restricting the language constructs used for membership conditions of semantic containers to the OWL 2 EL profile makes data discovery (matching information needs with available semantic containers) fast and scalable.

10 References

Balaban, A. 2016. *Testbed-12 Aviation Semantics Engineering Report*. http://docs.opengeospatial.org/per/16-039.html. Accessed 28 May 2017.

Catarci, T., and Lenzerini, M. 1993. "Representing and using interschema knowledge in cooperative information systems," *International Journal of Cooperative Information Systems* (2:4), pp. 375–398.

Domingue, J., Fensel, D., and Hendler, J. A. 2011. *Handbook of semantic web technologies*, Heidelberg, New York: Springer.

Eurocontrol. 2010. *Aeronautical Data Quality (ADQ) Implementing Rule*. http://www.eurocontrol.int/adq. Accessed 28 May 2017.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. 2004. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML - W3C Member Submission 21 May 2004*. https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

Keller, R. M. 2016. "Ontologies for aviation data management," in *Proceedings of the 35th IEEE/AIAA Digital Avionics Systems Conference (DASC)*.

Keller, R. M., Ranjan, S., Wei, M. Y., and Eshow, M. M. 2016. "Semantic representation and scale-up of integrated air traffic management data," in *Proceedings of the International Workshop on Semantic Big Data*, pp. 1–6.

Kovacic, I., Steiner, D., Schuetz, C., Neumayr, B., Burgstaller, F., Schrefl, M., and Wilson, S. (eds.). 2017. *Ontology-based data description and discovery in a SWIM environment*. 2017 Integrated Communications, Navigation and Surveillance Conference (ICNS).

McIlraith, S. A., Son, T. C., and Zeng, H. 2001. "Semantic web services," *IEEE intelligent systems* (16:2), pp. 46–53.

Motik, B., Sattler, U., and Studer, R. 2005. "Query Answering for OWL-DL with rules," *Rules Systems* (3:1), pp. 41–60.

Neumayr, B., Gringinger, E., Schuetz, C. G., Schrefl, M., Wilson, S., and Vennesland, A. (eds.). 2017. *Semantic data containers for realizing the full potential of system wide information management*. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC).

Patrick, T. E., Pascal, A. F., Rachid, G., and Anne-Marie, K. 2003. "The many faces of publish/subscribe," *ACM Comput. Surv.* (35:2), pp. 114–131.

Perry, M., and Herring, J. 2012. *GeoSPARQL - A Geographic Query Language for RDF Data*. http://www.opengeospatial.org/standards/geosparql. Accessed 28 May 2017.

Pipino, L. L., Lee, Y. W., and Wang, R. Y. 2002. "Data quality assessment," *Communications of the ACM* (45:4), pp. 211–218.

Porosnicu, E. 2013. *Metadata - Existing Guidelines*. https://www.eurocontrol.int/sites/default/files/content/documents/single-sky/mandates/20131210-adq-aixm-workshop-3-b-metadata-guidelines.pdf. Accessed 28 May 2017.

Steiner, D., Kovacic, I., Burgstaller, F., Schrefl, M., Friesacher, T., and Gringinger, E. 2016. "Semantic enrichment of DNOTAMs to reduce information overload in pilot briefings," in *Proceedings of the 16th Integrated Communications Navigation and Surveillance (ICNS) Conference*.

W3C. 2012a. *OWL 2 Web Ontology Language Direct Semantics (Second Edition) - W3C Recommendation 11 December 2012*. https://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/.

W3C. 2012b. *OWL 2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012*. https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.
W3C. 2012c. *OWL 2 Web Ontology Language Profiles (Second Edition) - W3C Recommendation 11 December 2012*. https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.
W3C. 2012d. *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition) - W3C Recommendation 11 December 2012*. https://www.w3.org/TR/owl2-rdf-based-semantics/.
W3C. 2012e. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition) - W3C Recommendation 11 December 2012*. https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.
W3C. 2013a. *RIF Basic Logic Dialect (Second Edition) - W3C Recommendation 5 February 2013*. https://www.w3.org/TR/2013/REC-rif-bld-20130205/.
W3C. 2013b. *RIF RDF and OWL Compatibility (Second Edition) - W3C Recommendation 5 February 2013*. https://www.w3.org/TR/2013/REC-rif-rdf-owl-20130205/.
W3C. 2014a. *RDF 1.1 Primer – W3C Working Group Note 24 June 2014*. https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.
W3C. 2014b. *RDF Schema 1.1 – W3C Recommendation 25 February 2014*. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.
Wilson, S. 2011. *Requirements for Aviation Metadata*. http://portal.opengeospatial.org/files/?artifact_id=41667. Accessed 28 May 2017.

The BEST consortium:

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