



# Ontology-based techniques for data distribution and consistency management in a SWIM environment

Deliverable 2.2

BEST

Grant:	699298
Call:	H2020-SESAR-2015-1
Topic:	Sesar-03-2015
Consortium coordinator:	Information Management in ATM
Dissemination Level:	SINTEF
Edition date:	PU
Edition:	17 May 2018
	01.00.01

Founding Members



## Authoring & Approval

### Authors of the document

Name/Beneficiary	Position/Title	Date
<b>Christoph Schuetz (LINZ)</b>	Project member	17.05.2018
<b>Bernd Neumayr (LINZ)</b>	Project member	17.05.2018
<b>Michael Schrefl (LINZ)</b>	Project member	17.05.2018
<b>Eduard Gringinger (FREQUENTIS)</b>	Project member	01.02.2018

### Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Joe Gorman (SINTEF)	Project leader	27.04.2018
Audun Vennesland (SINTEF)	Project member	12.05.2018

### 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	18.05.2018

### Rejected By - Representatives of beneficiaries involved in the project

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

### Document History

Edition	Date	Status	Author	Justification
00.00.01	18.01.2017	Planned Content and Structure (PCOS)	Christoph Schuetz	Completed PCOS for internal review
00.01.00	06.06.2017	Intermediate	Christoph Schuetz	Progress report for internal review
00.02.00	13.03.2018	Intermediate	Christoph Schuetz	Progress report for internal review
00.03.00	17.04.2018	External proposed	Christoph Schuetz	Proposed final text for internal review

00.04.00	30.04.2018	External proposed	Bernd Neumayr	Incorporate Joe's Comments
Edition	Date	Status	Author	Justification
01.00.00	09.05.2018	External proposed	Christoph Schuetz	Finalize the document for another round of internal review
01.00.01	17.05.2018	External	Christoph Schuetz	Finalize the document after last round of internal review



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

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

### Abstract/Executive Summary

---

Effective use of the Semantic Container approach developed in BEST depends on the existence of a Semantic Container Management system controlling the replication, distribution and consistency of containers. The deliverable provides a metamodel describing the information needed to implement such a Semantic Container Management System.

In the field of distributed databases, there are many existing techniques for distribution, replication and consistency management, mostly based on a single generic data model. In BEST we refine this, using different types of models for different kinds of information.

High availability of information and low network load are key goals for the success of the SWIM approach. Semantic containers, supported by a Semantic Container Management System, can contribute significantly to these goals.

The metamodel distinguishes between logical and physical containers, and indicates which containers are allocated to which nodes. It also allows definition of versions of containers, supporting consistency management and different forms of synchronization. Finally, the metamodel allows for traceability of data provenance, and definition of composite containers that gather data from lower-level elementary containers.

We illustrate the proposed metamodel using NOTAMs as example. We stress, however, that the semantic container approach applies to other types of ATM information as well.

The deliverable also outlines the architecture of a Semantic Container Management system, describing the distribution of the metadata and software components needed to implement it.

## Table of Contents

Abstract/Executive Summary .....	4
<b>1 Introduction: About this document .....</b>	<b>7</b>
1.1 Purpose .....	7
1.2 Intended readership.....	7
1.3 Relationship to other deliverables .....	8
1.4 Relationship to Publications.....	8
1.5 Acronyms and abbreviations.....	8
<b>2 Background.....</b>	<b>10</b>
2.1 Distribution and replication .....	10
2.2 Consistency management .....	10
2.3 Provenance .....	11
<b>3 Semantic container distribution and SWIM .....</b>	<b>12</b>
3.1 High availability of information.....	12
3.2 Decreased network load and computation effort .....	13
<b>4 Distribution and replication .....</b>	<b>14</b>
4.1 Logical containers .....	14
4.2 Physical containers and their allocation.....	16
4.3 Determining allocation sites .....	19
<b>5 Consistency management.....</b>	<b>20</b>
5.1 Versioning of semantic containers .....	20
5.2 Push and pull synchronization .....	22
<b>6 Provenance and composition.....</b>	<b>23</b>
6.1 Lineage of logical containers .....	23
6.2 Provenance of physical datasets .....	24
6.3 Container composition.....	27
<b>7 Semantic container management system: possible architecture.....</b>	<b>34</b>
7.1 Data Distribution Architecture .....	34
7.2 Software Distribution Architecture .....	34

- 8 Conclusion.....36**
- 9 References .....37**
- 10 APPENDIX A: ATM information cubes.....39**
  - 10.1 Overview..... 39**
  - 10.2 Background ..... 41**
    - 10.2.1 Rule-based filtering and annotation of ATM information .....41
    - 10.2.2 Related work.....41
  - 10.3 Data container management using ATM information cubes..... 41**
  - 10.4 Merge of semantic containers..... 45**
  - 10.5 Abstraction of data items..... 46**

# 1 Introduction: About this document<sup>1</sup>

---

## 1.1 Purpose

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

*“This deliverable comprises an ontology-based language for describing data distribution, including data lineage and freshness requirements.”*

In this context, the term “language” means a way of describing, in a formalised way, all the information that is needed to make effective use of semantic containers in information systems (such as SWIM) where data can be replicated and distributed to multiple locations.

Effective use of semantic containers depends on the existence of a *Semantic Container Management System*: software that manages the job of caching of containers, deciding where replicas should be stored, and maintaining consistency between them. Deliverable 3.2 provides a prototype of such a system; it is the purpose of this deliverable to provide the language needed to implement the semantic container management system.

The “language” provided by the deliverable is a metamodel for semantic container management, expressed as a collection of UML diagrams. The UML diagrams constitute an ontology complementary to the various existing ATM domain ontologies (see D 1.1), and translate into an RDF vocabulary that can be used in the implementation of a prototype semantic container management system (see D 3.2). The metamodel covers aspects of semantic container distribution and replication, consistency management, and provenance (semantic container lineage). A semantic container’s quality metadata indicate actual freshness and expected freshness of the contained ATM information.

The deliverable goes beyond its original scope by also introducing (in Appendix A, see Section 10) initial ideas on the concept of *ATM information cubes*. This concept provides a form of semantic container distribution that may serve as the basis for future research.

## 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)

---

<sup>1</sup> 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. The semantic container metamodel described in this deliverable represents an ontology complementary to existing ATM domain ontologies.
D 2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base	The concept of semantic containers and the fundamentals for definition and discovery are defined in D 2.1, which are re-used in this deliverable.
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 semantic container management system in D 3.2 builds on the concepts described in this deliverable.
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	D 5.1 formally investigates 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 Relationship to Publications

A work-in-progress version of the research presented in the main body of this deliverable was published as a paper [1] in the Proceedings of the Integrated Communications Navigation and Surveillance Conference 2018. Furthermore, the material in the appendix will be presented at the 2018 Congress of the International Council of the Aeronautical Sciences; a paper is currently under review for a recommendation letter. An extended version of that paper will be submitted for a special issue of the Aeronautical Journal, following an invitation of the program committee based upon the original submission of the extended abstract. **Some parts of the text of this document are copied directly from parts of these papers.**

### 1.5 Acronyms and abbreviations

Acronym/Abbreviation	Explanation
ANSP	Air Navigation Service Provider
AIRM	ATM Information Reference Model
AIXM	Aeronautical Information Exchange Model
ATM	Air Traffic Management
DNOTAM	Digital NOTAM



Acronym/Abbreviation	Explanation
EFB	Electronic Flight Bag
F-Logic	Frame Logic
FIXM	Flight Information Exchange Model
IWXXM	ICAO Meteorological Information Exchange Model
METAR	Meteorological Aerodrome Report
NOTAM	Notice To Airmen
OLAP	Online Analytical Processing
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SESAR	Single European Sky ATM Research
SPARQL	SPARQL Protocol and RDF 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 chapter, we briefly present background information on data distribution and replication, consistency management in distributed databases, and data provenance. We briefly explain the role of semantic container distribution for SWIM in a separate chapter (Chapter 3).

### 2.1 Distribution and replication

The algebra of qualified relations [2] is a well-established approach to distributed database management, which serves as a main inspiration for container versioning and consistency management in a distributed environment. We adapt the concept for SWIM information services and extend the concept with semantic labels to support the management of containers, specifically the discovery but also the description of container lineage and provenance.

Related work investigates the integration of information related to flights in a data lake [3]. In a sense, the semantic container approach can be considered an ontology-based, structured data lake approach. Other work [4] investigates the design and execution of web service workflow, metadata management has been identified an important topic [5]. Derivation chains of semantic containers provide a data-centric view on information service workflows.

Zand and Schandl [6] propose to use “Semantic Web technologies to build comprehensive descriptions of user’s information needs based on contextual information, and employs these descriptions to selectively replicate data from external sources.” This approach is about keeping on mobile devices local copies of relevant data, so that an application on the mobile device can operate also without network connectivity.

Replication of semantic containers is related to distribution and replication of (dynamic) XML documents [7, 8]. In comparison to existing approaches, BEST semantic containers offer a unique combination of version management, distribution and replication, and fine-grained provenance tracking. Instead of relying on a single generic data model (like XML), the semantic container approach uses different data/knowledge models for different kinds of information (XML for data, RDF for metadata, OWL for semantic labels) to get the best of all worlds. In contrast to generic XML-based approaches, the Semantic Container approach leverages the specifics of ATM information exchange, with data items like NOTAMs and METARs constituting the lowest grain of fragmentation.

Metwally et al. [9] model datacenter resources for infrastructure-as-a-service (IaaS) using an ontology model and employs reasoning for the allocation of resources.

### 2.2 Consistency management

In database replication, eager and lazy approaches can be considered [10]. Distributed semantic container management could follow both an eager and lazy replication approach, depending on the criticality of the data – for non-safety critical data, lazy replication may be preferable due to the lower replication costs. Lightweight approaches to versioning also for database systems have recently been proposed, e.g., OrpheusDB [11]. When datasets are collaboratively authored, version management is of importance and appropriate techniques for version management in the spirit of common version control systems must be developed [12].

## 2.3 Provenance

The PROV-O ontology [13] considers activities that are associated with an agent and use entities that were themselves generated by activities and derived from other entities. The semantic container approach builds on that provenance concept by considering services (activities) that are associated with a service provider (agent) and use containers (entities) that were themselves generated by services (activities) and derived from other containers (entities).

## 3 Semantic container distribution and SWIM

---

With respect to data distribution as described in this deliverable, the benefits of the semantic container approach for the SWIM concept are described in D 3.1 as follows:

- *High availability of information:* Semantic containers are packages of ATM information that can be redundantly allocated on various server nodes for increased availability.
- *Decreased network load:* Redundant allocation of semantic containers also reduces network load as SWIM information services may cache frequently used packages of ATM information for reuse.

In the following, we briefly summarize these main advantages of the semantic container approach's distribution concept for SWIM according to D 3.1. We refer to D 3.1 for further information.

### 3.1 High availability of information

In SWIM, different applications require different types of ATM information at various degrees of freshness and availability. An aircraft pilot may, for example, request current weather data. For availability's sake, consistency may be sacrificed: Slightly outdated weather information is better for a pilot than none at all. With respect to notifications about runway closures, on the other hand, pilots require fresh data because wrong information would entail potentially disastrous consequences. Semantic containers allow to make the inherent trade-off between freshness and high availability tangible for the consumer of ATM information: A semantic container packages ATM information, the resulting packages can be redundantly stored at multiple locations for high availability, administrative metadata indicate freshness and data quality.

Semantic containers also increase availability of the overall system by considering multiple sources of ATM information which semantic containers may be derived from. The semantic container metamodel as presented in this deliverable allows for the representation of multiple data sources for the same semantic container. A semantic container management system (see Section 1.1 and Section 7) may switch dynamically and transparently between different sources. Different sources may provide the same data with different quality in order to ensure that the consumer is alert to any reduction in quality of service. A primary source is a source with the highest data quality among the sources of the container. Secondary sources of lesser quality are only used when no primary source is available at the expected freshness.

A particular advantage of packaging ATM information in semantic containers is the possibility to allocate relevant information directly in the aircraft that operates a specific flight. The semantic container can be created a couple of days prior to the date the actual flight takes place, being filled with relevant information in advance. Shortly before the flight, at the departure airport with high bandwidth, the container can be uploaded onto the plane, and during the flight updated with only the critical information or information that requires only a low bandwidth.

ATM information is inherently dynamic: Government authorities and authoritative sources push new data and updates to existing data. Hence, the semantic container approach requires a mechanism to keep the contained ATM information up to date. The proposed semantic container metamodel paves the way for both push- and pull-based handling of updates.

### 3.2 Decreased network load and computation effort

Semantic containers allow SWIM services to cache frequently requested ATM information (e.g., weather data) for reuse. Multiple service providers may request the same ATM information from a remote entity. Typically, each request for ATM information is processed individually, thereby putting stress on the available bandwidth. With a semantic container management system in place, SWIM services may cache frequently requested ATM information as semantic containers at locations where they are frequently needed, thereby reducing the bandwidth and computation effort, including human processing and approval. For example, a NOTAM filtering service may cache relevant NOTAMs for the most important flight routes as semantic containers. When concrete requests for specific flights come in, rather than sifting through the whole body of NOTAMs currently in place, the service may use the pre-filtered semantic containers as a starting point for further filtering.

## 4 Distribution and replication

In the following, we present UML class diagrams that translate into an RDF vocabulary for the ontology-based management of semantic container distribution and replication (see D 3.2). The UML models themselves can be considered an ontology for semantic container management, which allows for the combination with other ATM ontologies and ontology modules (see D 1.1) that serve for the faceted semantic description of packages of ATM information (see D 2.1).

### 4.1 Logical containers

In analogy to the well-known ANSI/SPARC architecture for database management systems [14] as well as theory on distributed databases which considers logical and physical fragments [2], we distinguish between logical and physical aspects of semantic container management. In this respect, the word “logical” relates to information independent from its storage location and the word “physical” relates to information that is stored at a particular site. A semantic container is primarily a logical unit of data items with a semantic label that states a membership condition for data items. The membership condition expresses a commitment by the creator of the container: The semantic container comprises all data items that satisfy the membership condition (see D 2.1).

Figure 1 illustrates the metamodel of the faceted membership condition that is part of the semantic label. Facets are dimensions of semantic description, and can be classified into spatial, temporal, and other semantic facets. For example, a spatial facet may describe the geographic focus of the DNOTAMs in a semantic container, a temporal facet the time of validity of the DNOTAMs, another semantic facet may refer to the type of aircraft for which the contained DNOTAMs are relevant.

The facet values that a semantic label assigns for each facet come from an ontology. For example, a DNOTAM container may contain DNOTAMs relevant for fixed-wing aircraft, with fixed-wing aircraft being represented by a concept in an ontology derived from the ATM Information Reference Model (AIRM). A single facet may be defined by multiple ontologies, and the same ontology may serve to define the same facet.

Figure 2 illustrates instantiation of the metamodel in Figure 1. The container *METARs<Region: LSAS>* has the *LSAS* concept from the *AIRM* ontology as value for the *Geography* spatial facet. Similarly, the container *METARs<Region: EDGG>* has the *EDGG* concept from some ontology as value for the *Geography* spatial facet.

Besides the semantic label, a logical container may also have administrative metadata (Figure 3). These administrative metadata may be technical or quality metadata (see D 2.1). The metamodel of the semantic container approach is intentionally flexible on what kind of administrative metadata is included. Logical containers may have any kind and number of administrative metadata attributes. To an administrative metadata attribute, a logical container assigns a literal value. In the logical view of semantic containers, administrative metadata may refer to the expected accuracy of the contained data items (quality metadata), e.g., of weather forecasts. Similarly, technical metadata may state permissible data formats, e.g., XML or JSON. Other quality metadata may define the expected update frequency of the semantic container.

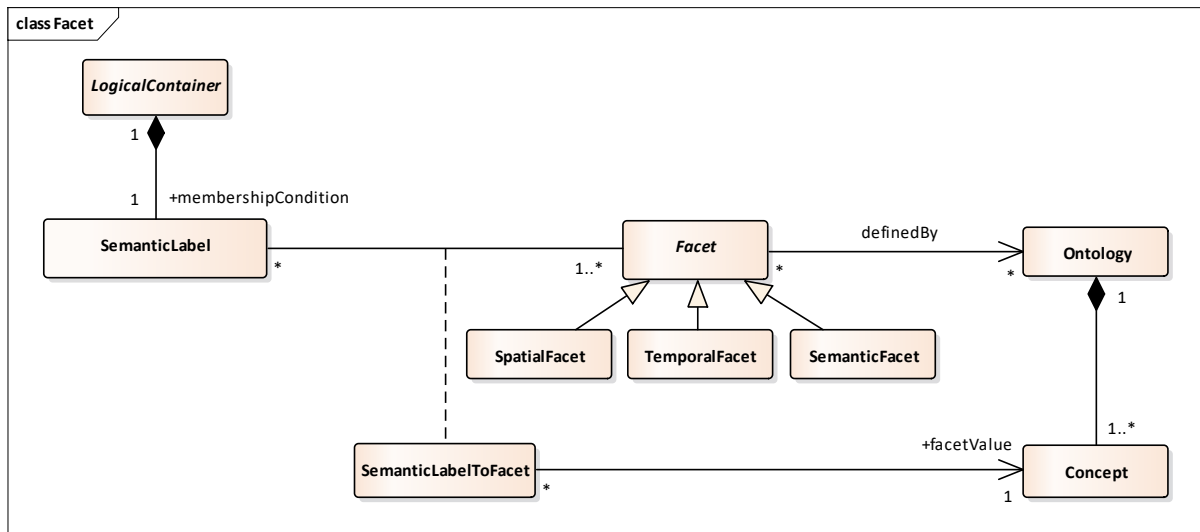


Figure 1. Class diagram “Facet” in the logical view: Logical containers and their semantic labels

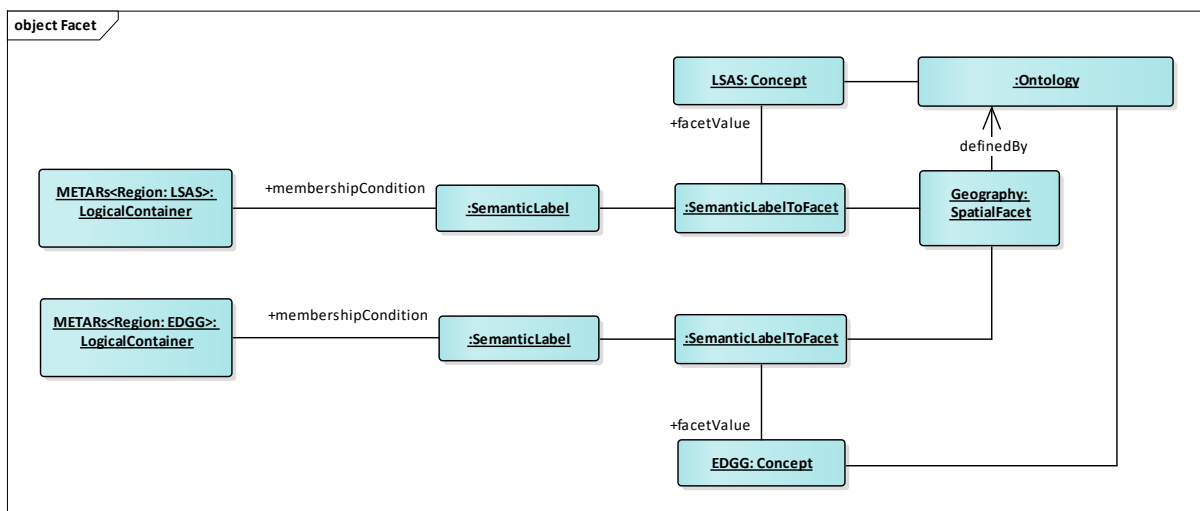


Figure 2. Object diagram “Facet”: Examples of logical containers and their semantic labels

In a distributed SWIM environment, the semantic description and the administrative metadata of the containers is important in order to find logical containers that fit a specific information need and requirements with respect to data quality and technical characteristics (such as data format). In a repository of logical containers, users and applications may look for containers with a certain content and properties. The actual content for a logical container may then be retrieved from different storage locations. Each logical container may have multiple copies, or replicas: These copies are the physical containers, which we describe in the next section.

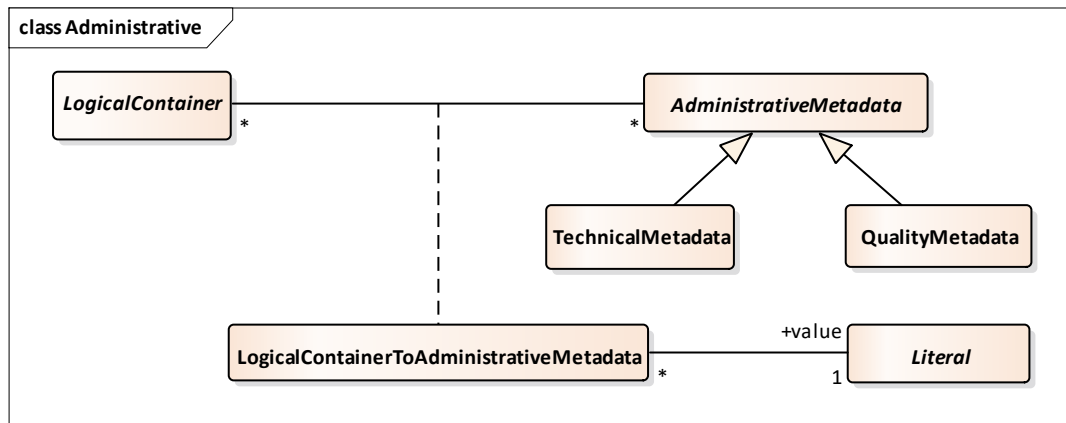


Figure 3. Class diagram “Administrative” in the logical view: Logical containers and their administrative metadata

## 4.2 Physical containers and their allocation

A semantic container can also be an actual physical package of data items, meaning that each logical semantic container also may have an allocation at a specific physical server location as well as replicas at multiple other locations (see Figure 4). For example, as illustrated in Figure 5, a logical semantic container with all DNOTAMs for a flight from Zurich to Frankfurt may be allocated on servers at Zurich airport and Frankfurt airport, or on the aircraft that conducts a specific flight from Munich to Frankfurt. Note that the *Location* class in the metamodel refers to a server or virtual machine, and not to an actual geographic location. For example, Zurich airport may have multiple server locations. The class *EntityElementaryLogicalContainer* in the object diagram refers to a container that only contains data items of a specific kind, e.g., NOTAMs, as opposed to a composite container (see Section 6.3).

We opt for a simple, yet powerful, distribution and replication concept: Each logical semantic container has one primary copy stored at some location, and potentially multiple replicas (secondary copies) stored at other locations. We note, however, that also other distribution and replication concepts may be considered, including decentralized solutions and reference-only containers. The former refers to a solution where no copy of a logical container is a designated primary. The latter refers to containers that have no physical materialization but are only logical concepts materialized upon request. We further discuss push- and pull-based consistency management in Section 5.2.

A physical container represents one copy of a logical container stored at a location. The logical container’s primary allocation is the location of the physical container that is the logical container’s primary copy. The secondary copies must be kept in sync with the primary copy. The local container management systems on each location may subscribe to receive updates for the secondary copies of a specific logical container that these locations hold. Alternatively, a pull-based approach may be followed. In that case, a physical container must store the date when the last sync with its primary copy has occurred, in order to be able to judge whether a synchronization should be attempted or not, which is also a form of administrative metadata.



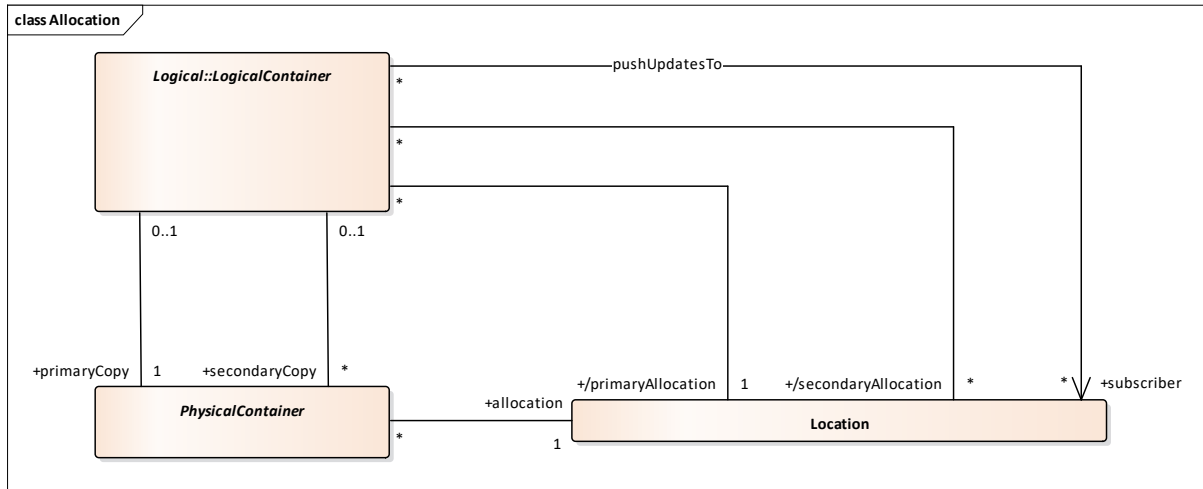


Figure 4. Class diagram “Allocation” in the physical view: Replication of semantic containers

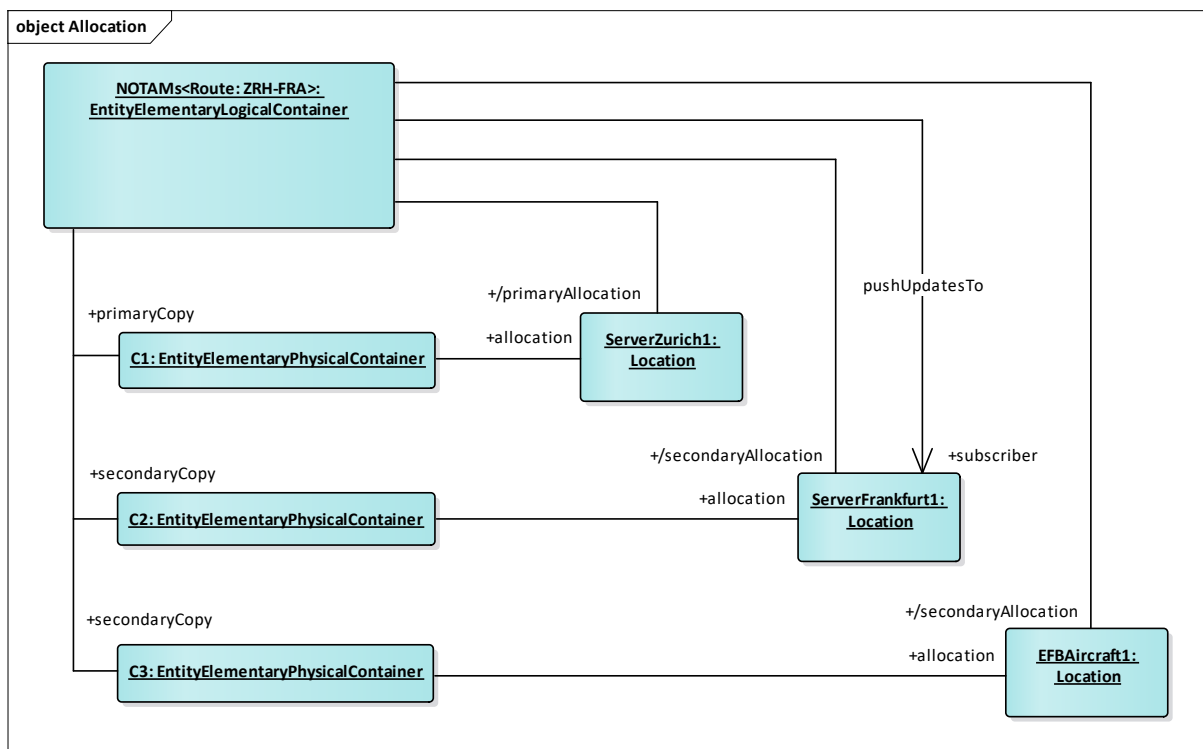


Figure 5. Object diagram “Allocation”: Examples of replication of semantic containers

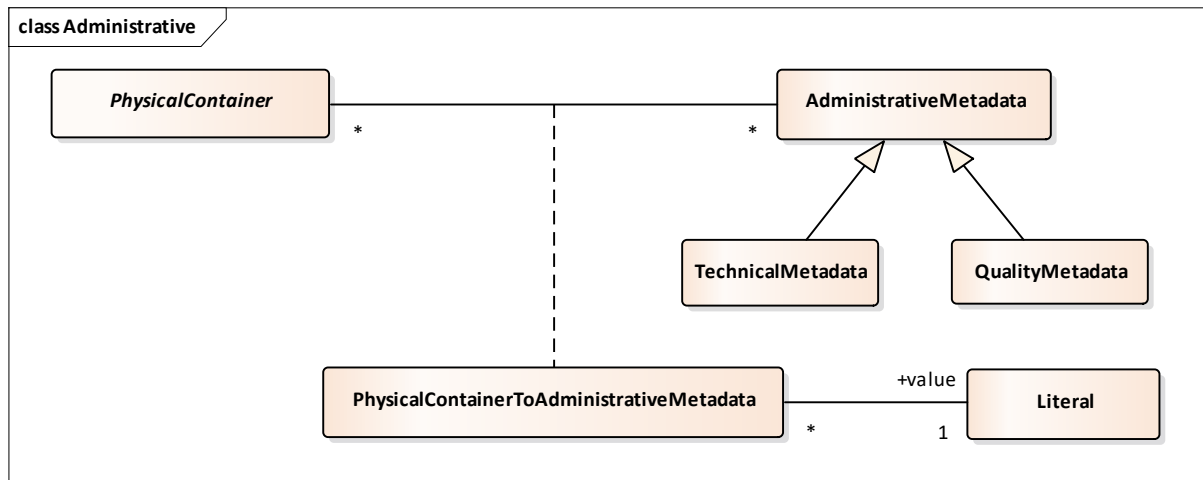


Figure 6. Class diagram “Administrative” in the physical view: Physical containers and their administrative metadata

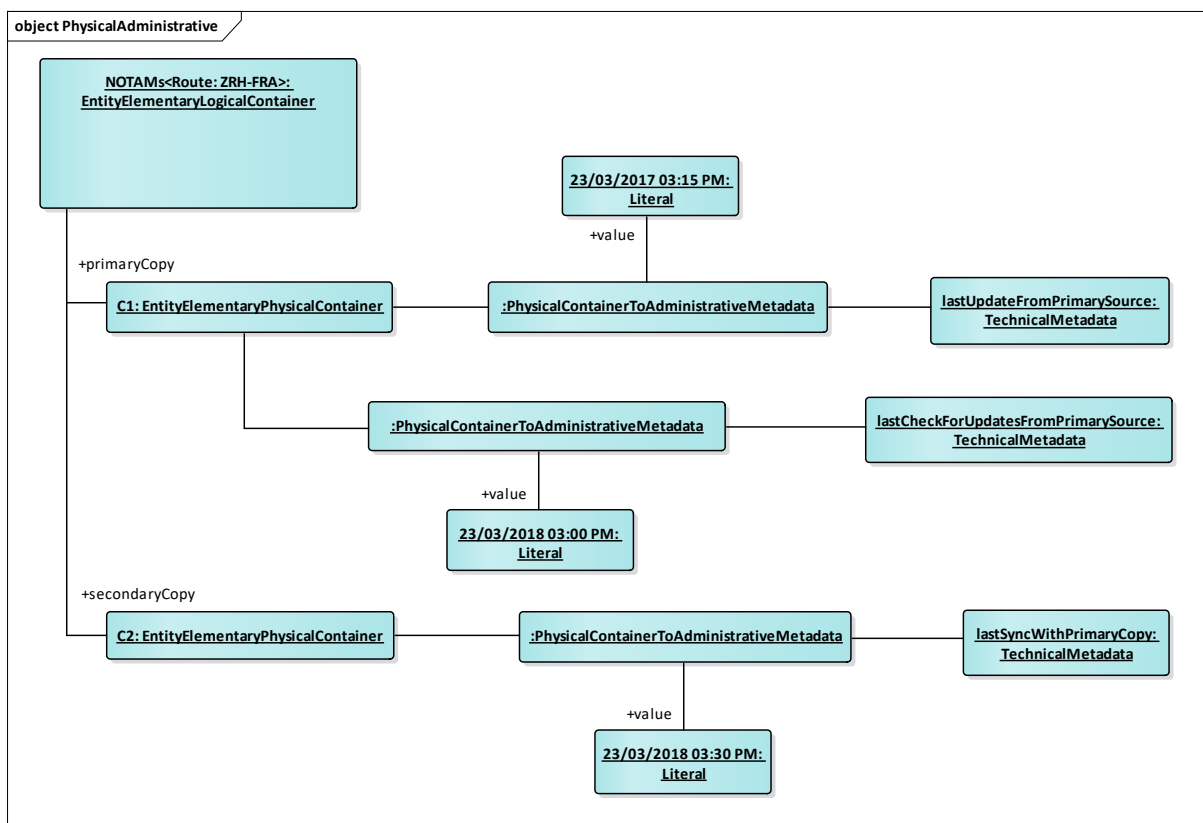


Figure 7. Object diagram “PhysicalAdministrative”: Technical metadata of physical containers

Each physical container may thus also have administrative metadata (Figure 6), independently from the logical container and the other physical replicas. An administrative metadata attribute may hence be the date and time of the last synchronization of a secondary physical container with the corresponding primary copy. An administrative metadata attribute may also be the date and time of the last update or last check for updates of a physical container from the source containers from which the container has been derived (see Chapter 6 for more information on container provenance). Figure 7 illustrates administrative data attributes of physical semantic containers. The NOTAM container for the route from Zurich to Frankfurt has a primary copy and a secondary copy at some locations (not shown in the diagram). Assume the logical container is derived from another source container. The *C1* primary copy of the container has attributes that indicate when the last update from a primary source took place and when the last check for updates from the primary source took place. The *C2* secondary copy of the container has an attribute that indicates when the last synchronization with the primary copy took place.

### 4.3 Determining allocation sites

The semantic description of the container contents may serve to determine beneficial allocation sites with the goal to improve performance, availability, or both. Spatial facet values could serve to allocate containers on different servers. For example, knowing that a container contains the data for a specific flight, the container could be allocated on servers at the departure and arrival airports, as well as nearby airports relevant for the route. In order for that approach to work, ontological concepts used for the faceted semantic description of data containers would have to be linked with server locations. Similarly, time facets may serve to identify packages of information that are only relevant for ex post analysis and are not operation-critical because the date has already passed.

We refer to literature on distributed database systems [2] for specific algorithms. The semantic facets may serve as additional parameters for deciding where to allocate a container. Containers may either be allocated at a single, most-beneficial allocation site using a best-fit algorithm, or allocated at multiple locations using an all-beneficial-sites algorithm.

The semantic container approach – and the metamodel described in this paper – allows for the consideration of the semantics of ATM information packages as criteria for the allocation decision. These decisions, however, are dependent on the specific business case.

## 5 Consistency management

---

Concerning updates to semantic containers, we distinguish containers that keep versions from containers that do not. Unversioned containers consist of contents and when that content changes, the previous content is forgotten. For auditability's sake, however, versioned containers are preferred since they allow to rebuild past states of information that led to certain decisions, which is important in the case of accidents and failures.

### 5.1 Versioning<sup>2</sup> of semantic containers

A versioned logical container has multiple versions of its contents as well as one current version (see Figure 8). Physically, only the versioned elementary containers have actual datasets (see Figure 9). A composite container's datasets (see Section 6.3) are its component physical containers' datasets. Now an elementary physical container has an initial dataset, and each update adds a delta set to the physical container. Concerning the composition of the physical container's current version, we consider two possibilities: Either the delta set adds to the set of valid data items – meaning that after the first update, the initial set plus the delta set constitute the container's current version – or the delta set replaces the previous sets and becomes the sole constituent of the current version. Either way, all delta sets are preserved for future auditability. Each data set also stores its creation time for audit purposes.

In case the primary copy is not reachable for synchronization, the secondary copies may be updated through other secondary copies, or alternative sources. In that case, however, should the alternative source be a non-primary source of information (see Chapter 6), the added dataset is a degenerated dataset. A version that consists of at least one degenerated dataset is a degenerated version. In that case, the contents of the physical version are likely of lesser quality than those of a regular physical version, or do not fully meet expected freshness requirements. Once the primary copy becomes available again, the degenerated datasets can be replaced by the regular sets in the current version, but are kept for audit purposes.

The administrative metadata captured for physical containers, although flexible, should include at least the following for the purposes of our distribution concept with one primary copy:

- **lastSynchronizationWithPrimaryCopy:** When was the physical container last synchronized with the corresponding primary copy.
- **lastCheckForSynchronizationWithPrimaryCopy:** When was the primary copy last checked for updates that must also be applied to the secondary copy.
- **lastSynchronizationWithSecondaryCopy:** When was the physical container last synchronized with a secondary copy.

A versioned composite physical container consists of multiple component (versioned) physical containers. The datasets of a composite container derive from the datasets of its components. Note

---

<sup>2</sup> In the field of configuration management, “version” is a generic term that can refer either to “revision” (changes over time, made in sequence) or “variant” (changes made in parallel, e.g. to adapt to different platforms). To be precise: in this document we are using “version” to refer to revisions. We use “version” as it is more widely understood term.

that all components of a composite container are allocated together on the same location. In Chapter 6, we define how logical and physical containers are derived by services from other containers, formalizing the principles of derivation chains as described in D 3.1.

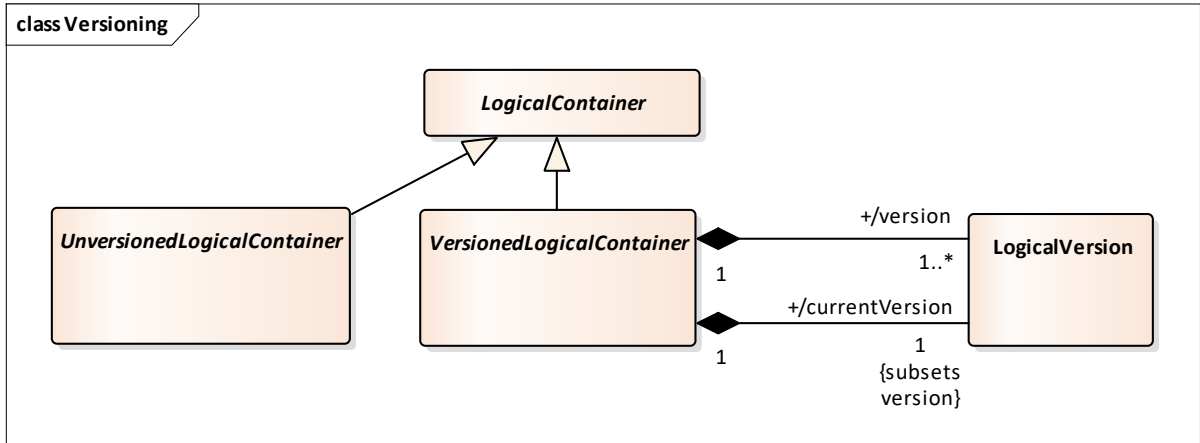


Figure 8. Class diagram “Versioning” in the logical view: Container versioning

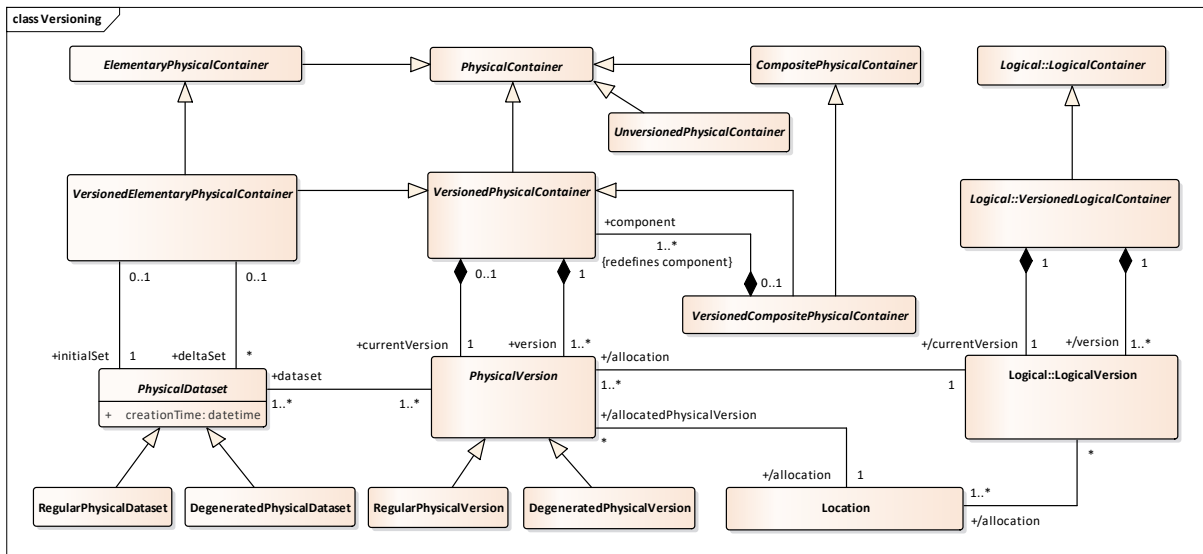


Figure 9. Class diagram “Versioning” in the physical view: Versions and physical datasets

Figure 10 illustrates an instantiation of the semantic container metamodel for versioning from the physical viewpoint. A versioned elementary physical container initially consists of a regular physical dataset. The first version is regular and consists only of the initial set. The second version consists of the initial dataset and the first delta set. The first delta set ( $S_2$ ) and the corresponding second version are regular. Then, for some reason, a degenerated physical dataset ( $S_3$ ) is added, making the third version, which consists of the first, second, and third datasets, a degenerated physical version. The delta set  $S_4$ , on the other hand, is regular again. The fourth version consists of  $S_1$ ,  $S_2$ , and  $S_4$ , but not

the generated dataset S3. Creation times of datasets, which are not shown in the example, allow for a reconstruction of the state of a physical container at any point in time, allowing for auditability in case of incidents.

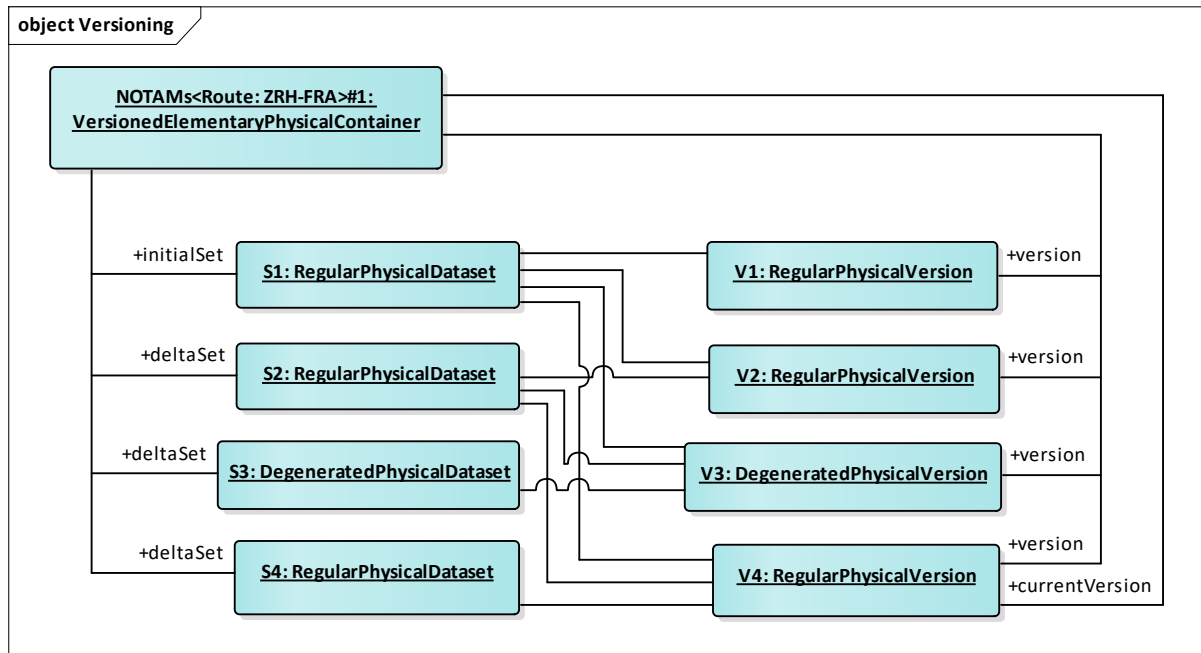


Figure 10. Object diagram “Versioning”: Example versions and physical datasets

## 5.2 Push and pull synchronization

The decision for a particular synchronization approach is orthogonal to the presented semantic container approach: The semantic container approach supports both push- and pull-based synchronization. A pull-based synchronization approach works without the source container – and management system thereof – knowing about the existence of the derived container that is to be kept in synchronization. A push-based synchronization approach lets semantic containers subscribe for updates at the source container. The source container has a list of subscribers for updates and the management system of the source container automatically pushes updates to the derived containers.

We accommodate for push-based synchronization in the metamodel by having a *pushUpdatesTo* relationship from logical container to location (see Figure 4). The location is a subscriber to updates, the container management system at a subscribing location will automatically receive updates from the primary location of the logical container.

Pull-based synchronization requires administrative metadata at the logical and physical level. Each physical container requires administrative metadata that records when the last synchronization and the last check for updates from the primary copy (and possibly secondary copy). Using the metadata, the container management system that manages the physical container can periodically check for updates, depending on the system’s settings.

## 6 Provenance and composition

The lineage of a semantic container in a derivation chain should be represented in the container management system. A logical container may have multiple primary sources as well as alternative secondary sources (Figure 11). The primary sources are the sources of prime quality. Secondary sources usually offer degraded quality.

### 6.1 Lineage of logical containers

A logical container derives from a primary or secondary source through a service call (Figure 11). Typically, a logical container has a single primary source, which is the source with the highest quality and freshness. In rare circumstance, a semantic container may have multiple alternative primary sources. Secondary sources of lesser quality are only used when no primary source is available at the expected freshness. Primary or secondary sources are again semantic containers. A service call has a semantic label as arguments and possibly many static containers as additional input. For example, static containers supplied to a digital briefing service as additional input may include a list of relevant aerodromes for a particular flight route. These containers are static in a sense that their input does not change or is only very slowly changing. The semantic label that serves as the arguments reflect the semantic label of the result container. Although omitted in the figure, the semantic label also provides a value for each associated facet.

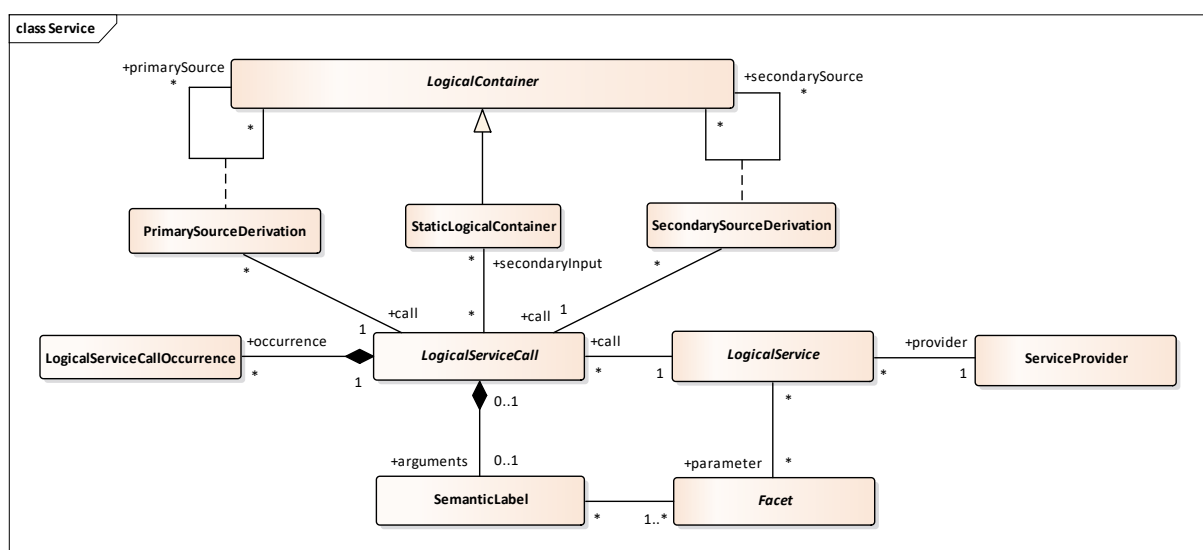


Figure 11. Class diagram “Service” in the logical view: Lineage of semantic containers

A service call can have multiple occurrences. The call occurrences, on a physical level, are then what actually produce a dataset (see Figure 13). Each service call belongs to a service, which has in turn a service provider.

Figure 12 illustrates an example lineage of semantic container according to the metamodel. The *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>* container has two (alternative) primary sources. The container may either be derived through one service call from the *NOTAMs<Region: Europe>*

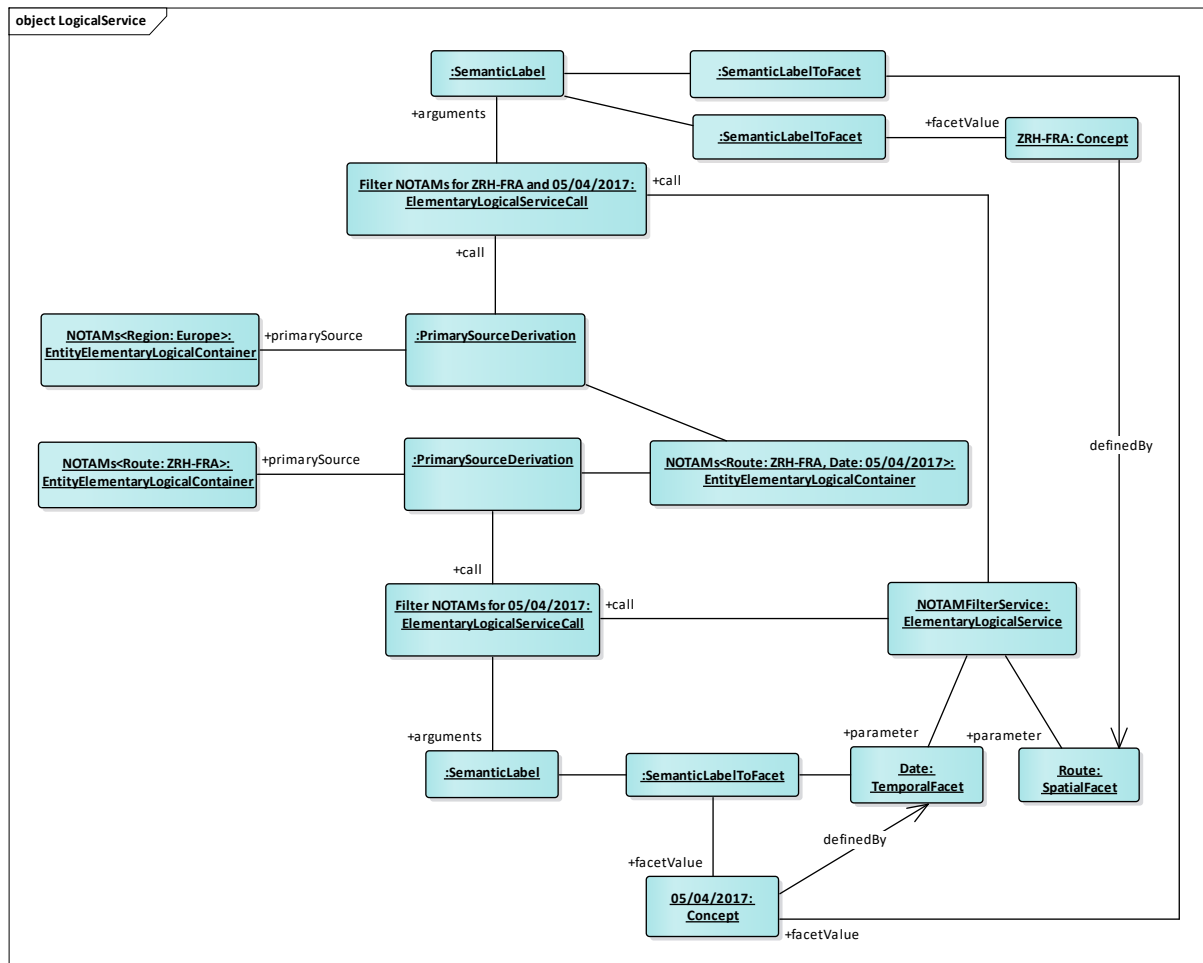


Figure 12. Object diagram “LogicalService”: Example lineage of logical containers

container, which contains all NOTAMs relevant to Europe, or through another service call from the *NOTAMs<Route: ZRH-FRA>* container, which contains all NOTAMs relevant for the route from Zurich to Frankfurt. Both calls that serve to derive the *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>* container from its sources are calls of the *NOTAMFilterService*. The *NOTAMFilterService* – we do not specify a provider in this example – has *Date* and *Route* facets as parameters, although not all service calls must supply arguments for all parameters. The *Filter NOTAMs for 05/04/2017* service call, for example, which derives the *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>* container from the *NOTAMs<Route: ZRH-FRA>* container, receives as arguments a semantic label with only the 05/04/2017 concept for facet value. The *Filter NOTAMs for ZRH-FRA and 05/04/2017* service call, on the other hand, which derives the *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>* container from the *NOTAMs<Region: Europe>* container, receives as arguments a semantic label with both the date and the *ZRH-FRA* concept for facet values.

## 6.2 Provenance of physical datasets

Services, just like containers, have logical and physical aspects that must be considered. Each logical service has a provider and may be realized as multiple physical services running at different locations. The location that produces a dataset may be different from the location where the source or result



dataset resides. The references to physical service calls and call occurrences allows to trace the creation of a dataset back to a specific service provider and physical server where the service was actually executed. Besides tracking provenance, the linking back to the creating service also allows assumptions about the data quality, since potential data quality attributes of the information service from the SWIM registry (when in place) can be used also to describe the quality of the semantic containers produced by the services.

The provenance concept of the semantic container approach was inspired by the PROV-O ontology [13]. The PROV-O ontology considers activities that are associated with an agent and use entities that were themselves generated by activities and derived from other entities. The semantic container approach builds on that provenance concept by considering services (activities) that are associated with a service provider (agent) and use containers (entities) that were themselves generated by services (activities) and derived from other containers (entities). In contrast to the PROV-O ontology, however, we distinguish between primary and secondary sources.

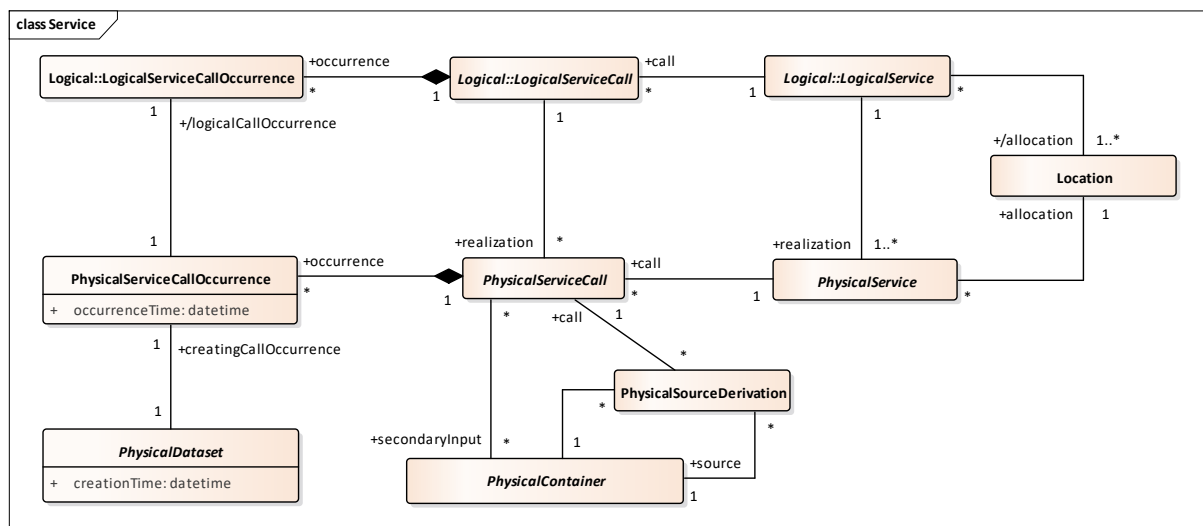


Figure 13. Class diagram “Service” in the physical view: Provenance of physical datasets

Figure 13 shows the metamodel for the definition of provenance of physical datasets. The classes *LogicalService*, *LogicalServiceCall*, and *LogicalServiceCallOccurrence* have physical counterparts. Hence, *PhysicalService* and *PhysicalServiceCall* represent the realizations of *LogicalService* and *LogicalServiceCall*, respectively. A physical service is allocated at a location, the server that hosts the service. A physical service call may have multiple physical service call occurrences (*PhysicalServiceCallOccurrence*). A physical service call occurrence has an occurrence time.

A physical dataset has a creation time and a creating call occurrence. Each physical call occurrence produces a single physical dataset.

A physical container may have multiple sources. The *PhysicalSourceDerivation* class represents the relationship between a source physical container and the result physical container. The derivation of a physical container from another physical container is done by a physical service call. Note that there can be multiple derivation links between the same physical source and result containers. A physical

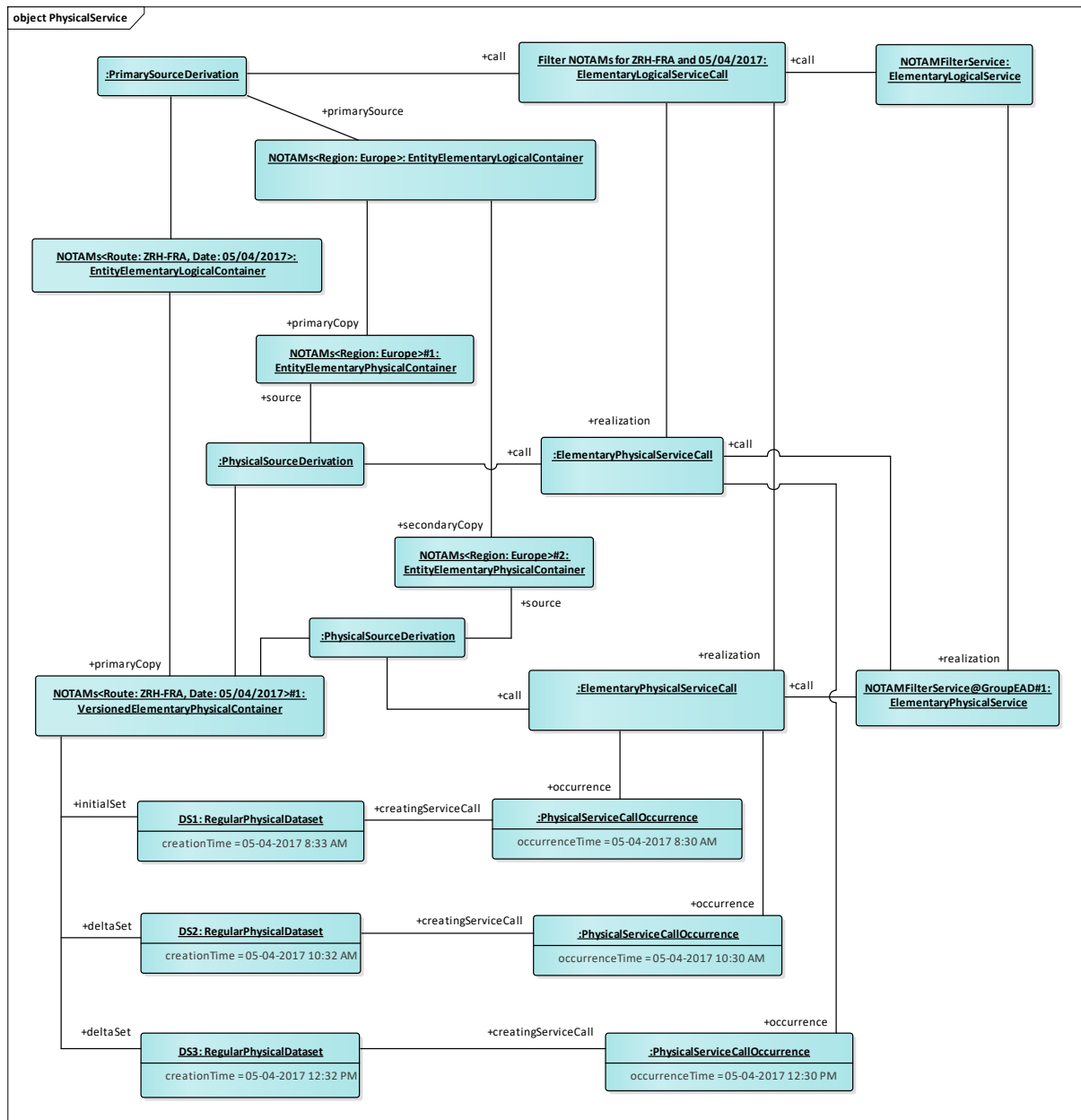


Figure 14. Object diagram “PhysicalService”: Example provenance of physical datasets

container *C1* can have one of its datasets derived from a physical container *C2* by a physical service call of a physical service *S1*. The container *C1* can have another one of its datasets derived from that same physical container *C2* by a physical service call of a physical service *S2*. Both *S1* and *S2* could be realizations at different locations of the same logical service.

Figure 14 illustrates an instantiation of the metamodel for physical dataset provenance. The logical container *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>* has the primary copy *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>#1*. The logical container derives from the primary source container *NOTAMs<Region: Europe>*, which has a primary copy *NOTAMs<Region: Europe>#1* and a secondary copy *NOTAMs<Region: Europe>#2*. The *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>#1* physical

container has an initial set and two delta sets (all regular). The *NOTAMFilterService* as physically realized by the *NOTAMFilterService@GroupEAD* was used to derive all three datasets. Two different calls, and three different call occurrences, however, were used to create the different datasets. The datasets *DS1* and *DS2* were created by occurrences of the same service call. The service call was used to create the datasets with the *NOTAMs<Region: Europe>#2* as input. The dataset *DS3* was created by a different service call, which uses the *NOTAMs<Region: Europe>#1* as input. Hence, the *NOTAMs<Route: ZRH-FRA, Date: 05/04/2017>#1* container, in its current state with three datasets, derived from the *NOTAMs<Region: Europe>#1* and *NOTAMs<Region: Europe>#2* container.

### 6.3 Container composition

Elementary containers consist of data items of a particular data item type. We distinguish between annotation elementary containers and entity elementary containers. An entity elementary container consists of data items of a particular entity type, e.g. NOTAMs, METARs. An annotation elementary container consists of data items that are annotation of a particular annotation type annotating entities of a particular entity type, e.g., NOTAM importances annotating NOTAMs. Figure 15 shows the metamodel for elementary logical containers. Figure 16 shows the metamodel for elementary physical containers.

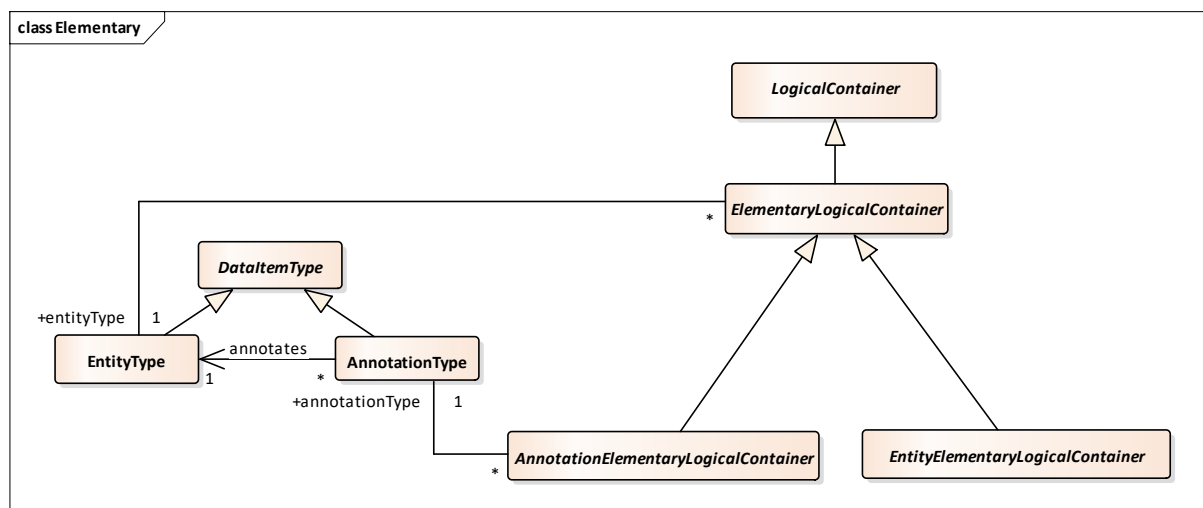


Figure 15. Class diagram “Elementary” in the logical view: Types of elementary semantic containers

Composite logical containers consist of other semantic containers (Figure 17). A composite container may either be homogeneous or heterogeneous. A homogeneous composite container consists of component containers (elementary or homogeneous composite) that all contain items of the same data item type. A heterogeneous composite container may consist of component containers with different types of data. Figure 18 illustrates an example instantiation of a heterogeneous composite logical container. The composite container has component containers that contain data items of types *METAR*, *NOTAM*, and *NOTAMPriority*. A homogeneous composite container with METARs for the EDGG and LSAS region, an entity elementary container with NOTAMs for the route from Zurich to Frankfurt, and an annotation elementary container with NOTAM priorities for a specific flight from Zurich to Frankfurt on a specific date make up the component containers of the composite container.

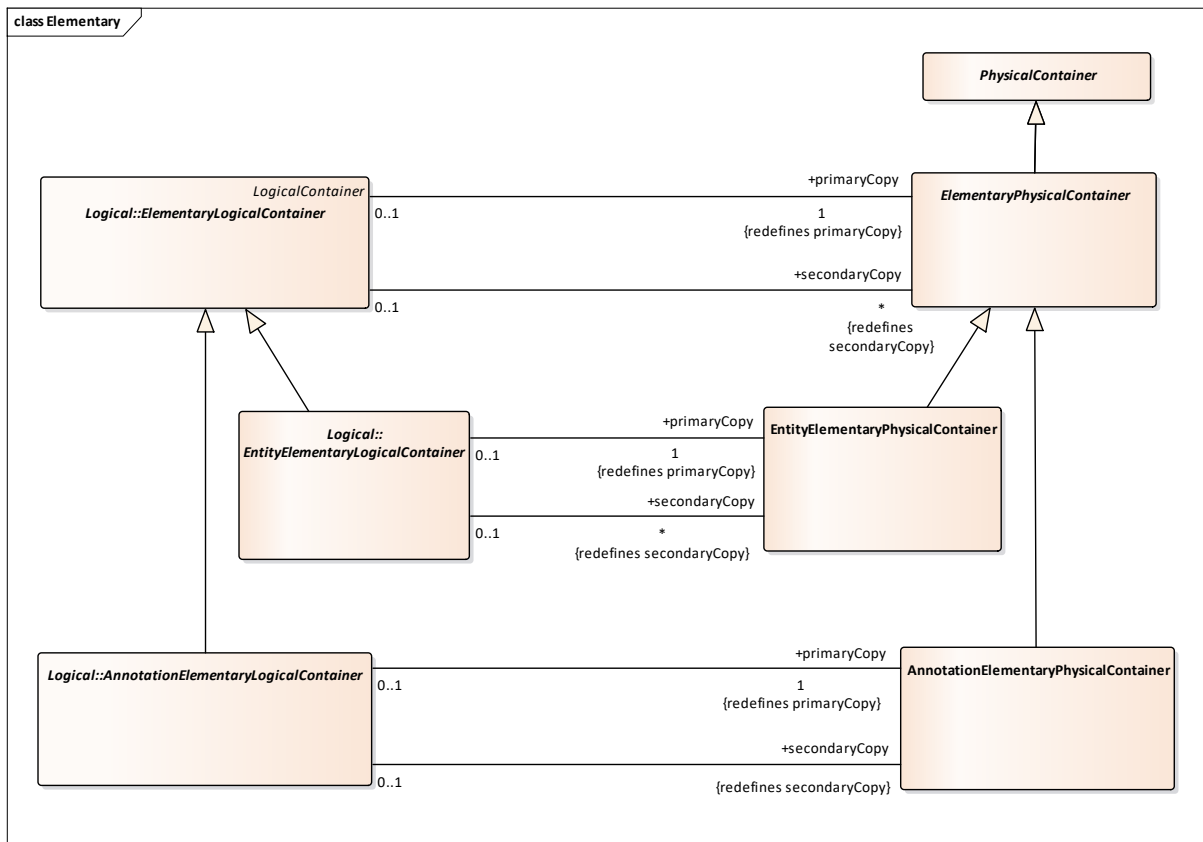


Figure 16. Class diagram “Elementary” in the physical view: Types of elementary semantic containers

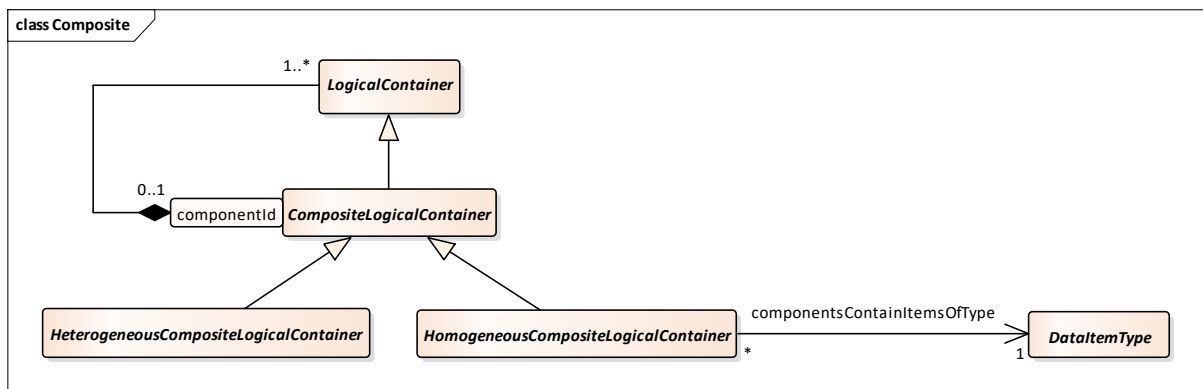


Figure 17. Class diagram “Composite” in the logical view: Types of composite semantic containers

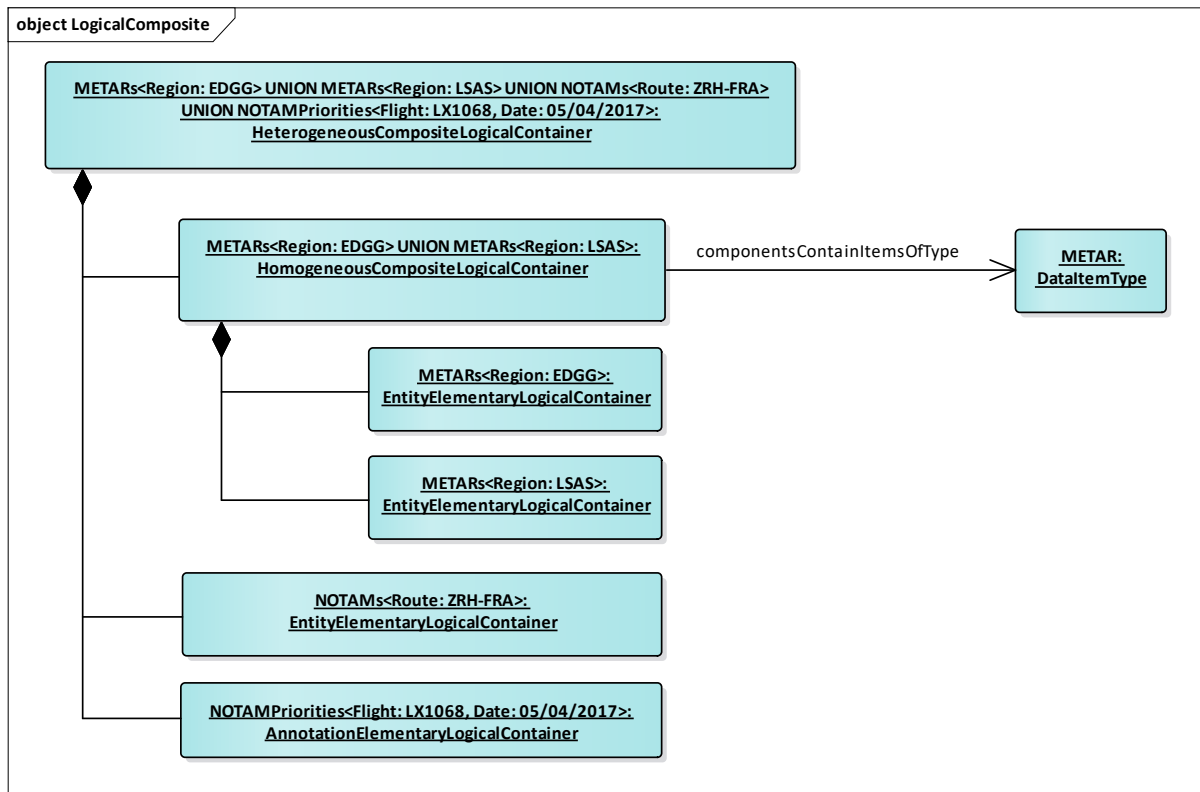


Figure 18. Object diagram “LogicalComposite”: Example composite semantic container

The physical model (Figure 19) of container composition mainly redefines the relationships of physical container.

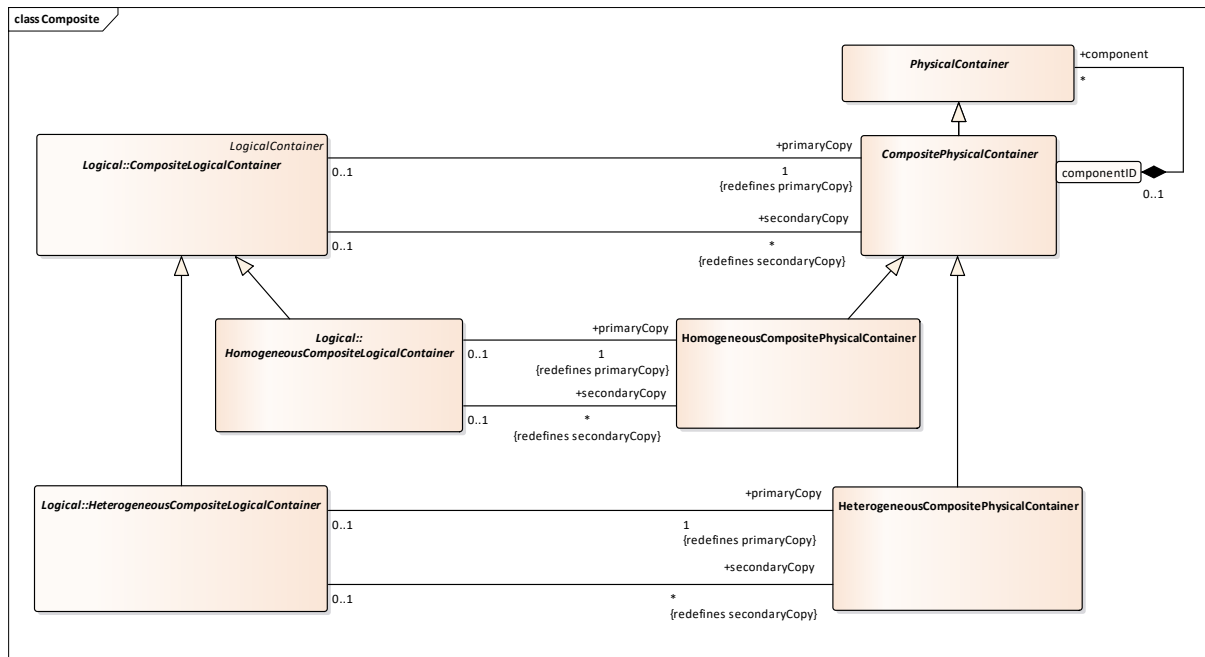


Figure 19. Class diagram "Composite" in the physical view: Types composite semantic containers

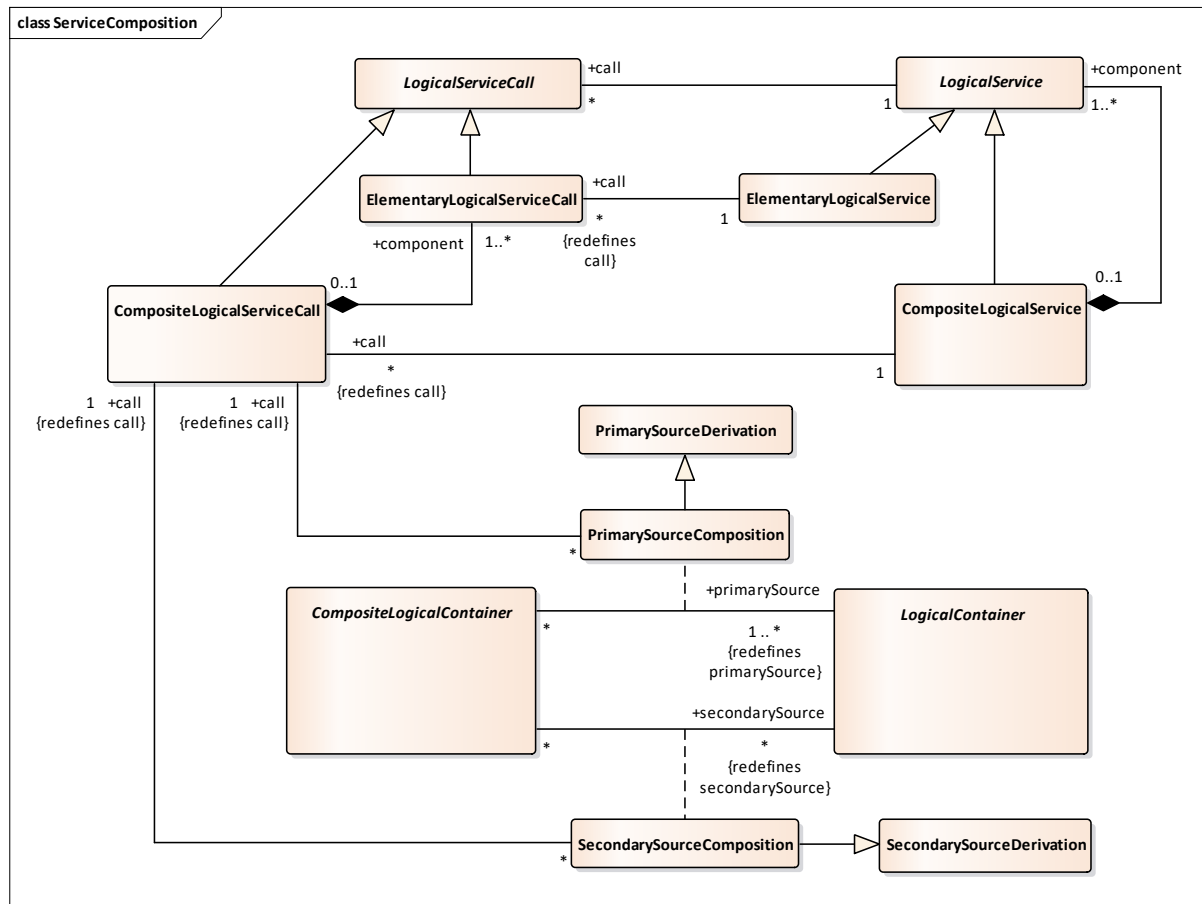


Figure 20. Class diagram “ServiceComposition” in the logical view: Composition of services

The provenance concept must likewise be adapted for composite containers: Composite logical containers are created by composite logical services. Figure 20 shows the metamodel for service composition, and it reflects the metamodel for container composition. A composite logical service consists of component logical services. A composite logical service has several composite logical service calls. The derivation of a composite logical container is a primary or secondary source composition – a specialization of primary and secondary source derivation, respectively.

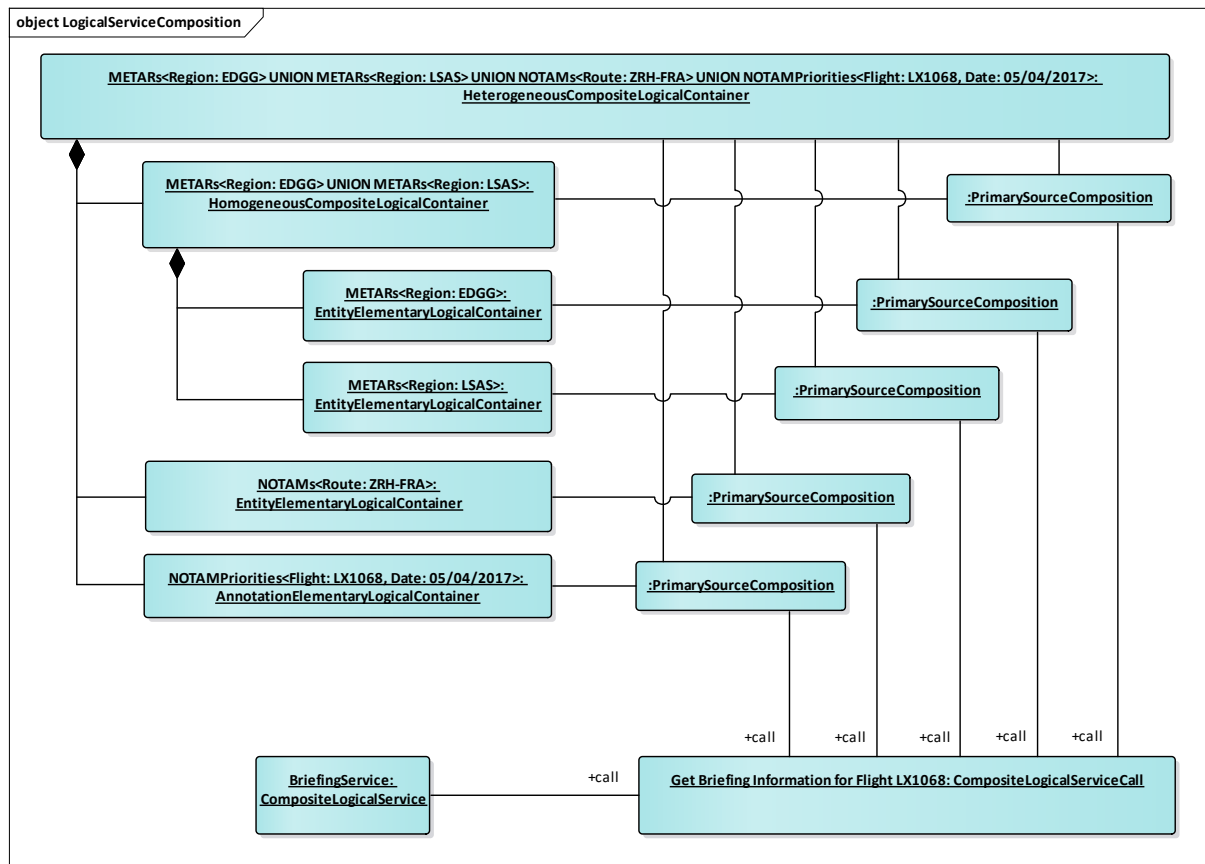


Figure 21. Object diagram “LogicalServiceComposition”: Composition of services

Figure 21 illustrates an example instantiation of the metamodel for logical service composition. A *BriefingService* instance of *CompositeLogicalService* may be called to get the briefing information for a specific flight. A heterogeneous composite logical container with METARs for regions EDGG and LSAS as well as NOTAMs for the route from Zurich to Frankfurt and NOTAM priorities for a specific flight may be derived by one such call to the briefing service. Each *PrimarySourceComposition* link makes reference to the same call. A call can only be used for one composite container: A call binds together all the primary source compositions that belong together. Another call binds together another, alternative primary or secondary derivations (compositions) of a composite semantic container.



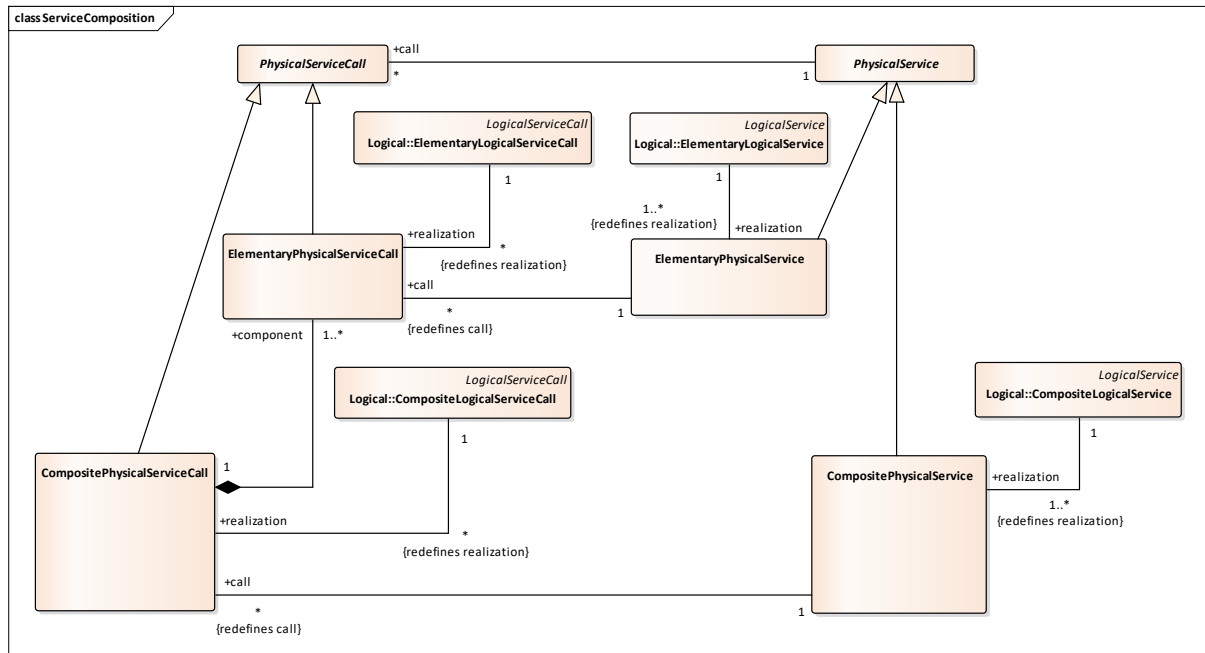


Figure 22. Class diagram “ServiceComposition” in the physical view: Composition of services

Figure 22 shows the metamodel for service composition at the physical level. Physical service composition basically redefines the relationships of physical services and physical service calls.

## 7 Semantic container management system: possible architecture

---

In this section we introduce a possible architecture for a semantic container management system. The semantic container approach as introduced in deliverable D 2.1 and the language (UML metamodel) introduced in this deliverable may be implemented in many different forms, it is independent of a concrete software and data distribution architecture. Other and maybe more adequate architectures may be developed in the future based on the vast literature on distributed systems (e.g., [15, 16]). The herein described proposed architecture serves two purposes, namely

- to give a more complete picture of a globally distributed semantic container management system, and
- to serve as a starting point for the development of more advanced software and data distribution architectures.

A semantic container management system is distributed over multiple server locations and multiple client locations. Locations are connected over the Internet. Container content and metadata are allocated redundantly at multiple locations. Centrally-provided software is run independently at the different locations which cooperate to provide globally-distributed semantic container management.

### 7.1 Data Distribution Architecture

Container content and metadata are allocated redundantly at multiple locations. A semantic container consists of location-independent metadata (represented by the logical semantic container), location-dependent metadata (represented by the physical semantic container) and content (also referred to as data or information, e.g., a set of AIXM Digital NOTAMs).

Metadata is modelled by the UML metamodel presented in this deliverable which can be translated into an RDF vocabulary (see D 3.2). The container metadata are then represented as RDF triples. All container metadata can be collected into an RDF graph. This RDF graph of all semantic containers is fully replicated at every server location and partially replicated at client locations. Each location runs an RDF database management system (a.k.a. graph store) and SPARQL query engine for storing, modifying and querying (parts of) the RDF graph. Modifications of metadata at some location are replicated in an asynchronous manner to other locations to provide for redundancy of metadata in case of connection or network failures. Replica consistency of metadata is maintained by giving priority to most recent writes.

Container contents remain in their original form (XML documents according to AIXM, IWXXM, or FIXM). Each location runs an XML database management system (a.k.a. document store) for storing and querying the contents of its allocated containers.

### 7.2 Software Distribution Architecture

Each server location independently runs a software package which makes available functionality for managing and querying data and metadata via RESTful web services. A client location (or sink), e.g., an electronic flight bag on board of an aircraft, may run a client variant of the software package which

provides a subset of this functionality. The software package (in its server and client variants) is distributed from a central software repository.

A client location provides functionality for:

- Allocating an existing semantic container
- Provisioning of semantic containers including content and metadata
- Keeping data and metadata of allocated semantic containers up-to-date via push and/or pull from their primary sources
- Keeping semantic containers up-to-date from alternative sources in case of unavailability of primary sources

A server location additionally provides functionality for:

- Creating a semantic container, storing its primary copy, deriving locations for secondary copies
- Calling services to derive/update the contents of semantic containers
- Forwarding modifications of semantic containers to client containers via push and pull
- Creating, updating and deleting semantic containers
- Discovery of semantic containers

## 8 Conclusion

---

A replication mechanism for the redundant storage of semantic containers promises higher availability of mission-critical data within SWIM while at the same time reducing the network load of SWIM. By packaging ATM information in semantic containers, SWIM information services may cache often used information and thus avoid frequent calls to other SWIM services. Applications may store local copies of important information to hedge against network outage when availability of the information is mission-critical, e.g., on an aircraft. Furthermore, semantic containers are a mechanism to retain provenance information when packaging ATM information from different SWIM information services. Thus, when a composite SWIM information service returns a composite semantic container based upon information from various other SWIM services, provenance information about the semantic container's components is preserved, which is important for auditability purposes.

A semantic container management system providing mission-critical data and metadata requires special consideration of trustful communication to ensure authentication, integrity, and nonrepudiation of data and metadata. In a decentralized system, trust can only be provided based on cryptographic protocols (see [17]). This is clearly out of scope of the BEST project. Future research needs to investigate how BEST's semantic containers can be combined with cryptographic protocols (e.g., using blockchain technology) to provide trustful semantic container management and secure SWIM.

As a concluding remark, we note that the BEST semantic container approach is not meant to rival SWIM, but to provide a data-centric layer for SWIM information services, which the SWIM services and applications may use for the management of ATM information.

## 9 References

---

- [1] E. Gringinger, C. Schuetz, B. Neumayr, M. Schrefl, and S. Wilson, "Towards a Value-Added Information Layer for SWIM: The Semantic Container Approach," in *Proceedings of the 18th Integrated Communications Navigation and Surveillance (ICNS) Conference*, 2018.
- [2] S. Ceri and G. Pelagatti, *Distributed databases: principles and systems*: McGraw-Hill, New York, 1984.
- [3] M. A. Martínez-Prieto *et al.*, Eds., *Integrating flight-related information into a (Big) data lake*. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), 2017.
- [4] K. Wolstencroft *et al.*, "The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. Web Server issue, W557-W561, 2013.
- [5] K. Belhajjame *et al.*, Eds., *Metadata Management in the Taverna Workflow System*. 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008.
- [6] S. Zander and B. Schandl, "Context-driven RDF data replication on mobile devices," *Semant. web*, vol. 3, no. 2, pp. 131–155, 2012.
- [7] S. Abiteboul *et al.*, "Dynamic XML documents with distribution and replication," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, California: ACM, 2003, pp. 527–538.
- [8] G. Koloniari and E. Pitoura, "Peer-to-peer management of XML data: Issues and research challenges," *SIGMOD Rec*, vol. 34, no. 2, pp. 6–17, 2005.
- [9] K. M. Metwally, A. Jarray, and A. Karmouch, Eds., *Two-phase ontology-based resource allocation approach for IaaS cloud service*. 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), 2015.
- [10] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, Eds., *Database replication techniques: A three parameter classification*. Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000, 2000.
- [11] L. Xu, S. Huang, S. Hui, A. J. Elmore, and A. Parameswaran, "OrpheusDB: A Lightweight Approach to Relational Dataset Versioning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago, Illinois, USA: ACM, 2017, pp. 1655–1658.
- [12] A. P. Bhardwaj *et al.*, "DataHub: Collaborative Data Science & Dataset Version Management at Scale," *CoRR*, vol. abs/1409.0798, 2014.
- [13] W3C, *PROV-O: The PROV Ontology*. [Online] Available: <https://www.w3.org/TR/2013/REC-prov-o-20130430/>.
- [14] D. A. Jardine, *The ANSI/SPARC DBMS Model; Proceedings of the Second Share Working Conference on Data Base Management Systems, Montreal, Canada, April 26-30, 1976*: Elsevier Science Inc, 1977.
- [15] A. S. Tanenbaum and M. van Steen, *Distributed systems: Principles and paradigms*: Prentice-Hall, 2007.
- [16] M. T. Özsu and P. Valduriez, *Principles of distributed database systems*: Springer Science & Business Media, 2011.
- [17] B. Schneier, *Applied Cryptography*, 2nd ed.: John Wiley and Sons, 1996.
- [18] ICAO, *Aeronautical Information Services Manual*, 6th ed.
- [19] ICAO, *Aeronautical Information Services - Annex 15*, 15th ed.

- [20] D. Steiner *et al.*, “Semantic enrichment of DNOTAMs to reduce information overload in pilot briefings,” in *Proceedings of the 16th Integrated Communications Navigation and Surveillance (ICNS) Conference*, 2016.
- [21] I. Kovacic *et al.*, Eds., *Ontology-based data description and discovery in a SWIM environment*. 2017 Integrated Communications, Navigation and Surveillance Conference (ICNS), 2017.
- [22] B. Neumayr *et al.*, Eds., *Semantic data containers for realizing the full potential of system wide information management*. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), 2017.
- [23] R. Pelchen-Medwed and E. Porosnicu, “Enhanced pilot situational awareness through the digital/graphical pre-flight briefing concept,” *HindSight*, vol. 23, pp. 66–69, 2016.
- [24] D. Steiner, F. Burgstaller, E. Gringinger, M. Schrefl, and I. Kovacic, “In-Flight Provisioning and Distribution of ATM Information,” in *Proceedings of the 30th Congress of the International Council of the Aeronautical Sciences (ICAS)*, 2016.
- [25] S. Gabree, M. Yeh, Y. J. Jo, and others, “Electronic flight bag (EFB): 2010 industry survey,” Federal Aviation Administration. Air Traffic Organization Operations Planning. Human Factors Research and Engineering Group, 2010.
- [26] A. Vaisman and E. Zimányi, *Data Warehouse Systems - Design and Implementation*: Springer, 2014.
- [27] F. Burgstaller, D. Steiner, B. Neumayr, M. Schrefl, and E. Gringinger, “Using a model-driven, knowledge-based approach to cope with complexity in filtering of notices to airmen Canberra, Australia, February 2-5, 2016,” p. 46.
- [28] C. Chen *et al.*, “InfoNetOLAP: OLAP and mining of information networks,” in *Link Mining: Models, Algorithms, and Applications*: Springer % CKR and contexts, 2010, pp. 411–438.
- [29] C. Schütz, B. Neumayr, and M. Schrefl, “Business model ontologies in OLAP cubes,” in *CAiSE 2013*, 2013, pp. 514–529.
- [30] FAA, *Federal NOTAM system airport operations scenarios*. Available: <https://notams.aim.faa.gov/FNSAirportOpsScenarios.pdf>.
- [31] R. Sherman, *Business Intelligence Guidebook*: Morgan Kaufmann, 2015.
- [32] H. -J. Lenz and A. Shoshani, “Proceedings of the 9th International Conference on Scientific and Statistical Database Management,” in *Proceedings of SSDBM 1997*, 1997, pp. 132–143.
- [33] *AIXM 5.1.1 - Data Model (UML)*.

## 10 APPENDIX A: ATM information cubes

---

In this appendix, we propose the use of semantic containers as the fundamental for data distribution in ATM information cubes. The content of this appendix is an adaption of a paper submitted to the ICAS Congress 2018 for review<sup>3</sup>, which we will further extend following an invitation by the ICAS Congress 2018 organizers to submit to the *Aeronautical Journal*<sup>4</sup> for a special issue. The research results described in this *appendix* may serve as the starting point for future research that continues the effort of the BEST project, building on the semantic container concept.

The original motivation behind ATM information cubes is as follows, but could be extended to other types of ATM information as well: Pilot briefings, in their traditional form, drown pilots in seas of information. Rather than unfocused swathes of ATM information, pilots require the information that they need for their flight at hand. To this end, we introduce the notion of ATM information cubes – in analogy to data cubes in data warehousing and OLAP. We introduce a conceptual framework for the summarization of semantic containers using merge and abstraction operations, yielding a higher-level management summary of relevant information.

### 10.1 Overview

A Pre-flight Information Bulletin (PIB) provides pilots with current Notices to Airmen relevant for a particular flight [18]. A Notice to Airmen (NOTAM) notifies aviation personnel about temporary changes regarding flight conditions [19], e.g., closure of air space or unserviceable navigation aids. PIBs are traditionally delivered on paper in textual form, with limited possibilities for structuring the data, drowning pilots in information. Digital NOTAMs (DNOTAMs), on the other hand, facilitate classification of data items along different dimensions, e.g., importance, geographic area, flight phase, and event scenario, that can be employed to flexibly structure the PIB in order to reduce information overload [20]. For example, using the classification rules developed in the *Semantic NOTAM* (SemNOTAM) project (see [20]), DNOTAMs can be packaged into *semantic containers* (see [21]), each container comprising the DNOTAMs about the same event scenario that have the same importance for a certain flight on a particular date in a given flight phase at some geographic location. Consider, for example, the semantic containers on the left-hand side of Figure 23 which, given a flight and date assumed to be fixed, contain the DNOTAMs for flight information region (FIR) segment *EDDU-01* classified as an *operational restriction* for the *cruise* flight phase and the DNOTAMs for FIR segment *EDDU-02* classified as flight critical for the *descent* flight phase, respectively.

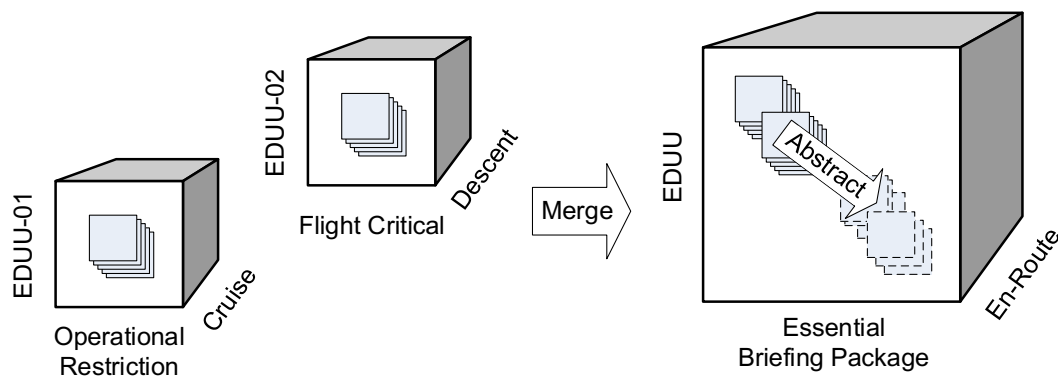
Semantic containers may also contain other types of data items relevant for pilot briefings, classified along different dimensions [22], which allows for the representation of an enhanced PIB (ePIB [23]) that also includes information beyond DNOTAMs, e.g., meteorological (MET) information such as METAR and TAF messages in digital form. Rule-based approaches similar to SemNOTAM could also be devised for messages other than DNOTAMs (cf. [24]). Indeed, an electronic flight bag (EFB) platform

<sup>3</sup> See <http://www.icas.org/> for further information.

<sup>4</sup> <https://www.cambridge.org/core/journals/aeronautical-journal>

may display various kinds of relevant ATM information for a flight, depending on the manufacturer [25]. The question is how to organize relevant information in order to enable applications to make smart use of it.

In the following, we propose a conceptual framework for further summarization of semantic containers using *merge* and *abstraction* operations, yielding a higher-level management summary of relevant information. To that end, we adapt the concept of OLAP cubes – i.e., multidimensional data structures for online analytical processing (OLAP) – and OLAP query operators (see [26] for further information) to work with information in air traffic management (ATM). We hence propose the notion of ATM information cube, which hierarchically organizes semantic containers. We assume the existence of appropriate rule-based filtering mechanisms to collect ATM information into the containers. The SemNOTAM classification rules [20], for example, provide a first reduction of the information overload in pilot briefings by packaging DNOTAMs into collections of smaller containers, analogous to OLAP cubes. These containers can be merged in order to obtain more comprehensive containers of data items. For example, individual containers with flight critical DNOTAMs and DNOTAMs about operational restrictions, respectively, are merged into a container with DNOTAMs that comprise the essential briefing package (see Figure 23); containers with DNOTAMs about different en-route segments are merged into containers about entire flight information regions (FIRs); containers with DNOTAMs relevant to the cruise flight phase or descent flight phase, respectively, are merged into a container relevant for en-route phases in general. The data items can then be further combined to obtain more abstract data items. For example, individual DNOTAMs concerning specific runway closures for landing aircraft are combined into one abstract DNOTAM indicating the existence of a runway closure for landing aircraft in a specific context, with only more general (or abstract) information about obstructions, hazards, construction activity, etc. given.



**Figure 23. Illustration of merge and abstract operations for semantic containers: The semantic containers on the left are merged into a single container, the contents of which are altered through application of some abstraction operation.**



## 10.2 Background

In this section, we present relevant background information on semantic containers as well as rule-based filtering and annotation of ATM information. We further discuss related work.

### 10.2.1 Rule-based filtering and annotation of ATM information

The metadata-centric semantic container approach needs to be complemented by information processing and reasoning techniques at the instance level – in order to derive the actual content of semantic containers – as provided by the SemNOTAM approach [20]. The SemNOTAM approach comes with a mix of techniques necessary to cope with the complexity of ATM domain knowledge and of the filtering and annotation task [27].

In the SemNOTAM approach, the SemNOTAM engine receives ATM information, i.e., a set of DNOTAMs, and the user's interest specification as input, translates the input into a representation that suits knowledge-based reasoning, selects from its knowledge base the relevant set of filtering and annotation rules and lets the knowledge-base reasoner execute these rules against the ATM information. The result of the reasoning process – a filtered and enriched set of DNOTAMs – is provided to the user, who typically is a pilot or air traffic controller.

Combining SemNOTAM with BEST's semantic container approach, the input and output of the SemNOTAM engine are the contents of semantic containers. Different instances of SemNOTAM, with different configurations and rules, may then act as information services in derivation chains of semantic containers.

### 10.2.2 Related work

Traditional OLAP systems work on multidimensional models with numeric measures (see [26]). Going beyond numeric measures, InfoNetOLAP [28], which is also known as graph OLAP, associates weighted graphs with dimension attributes. Topological and informational roll-up are the basic kinds of operations, akin to merge and abstraction operations presented in this paper. The focus of graph OLAP, however, are weighted directed graphs with highly structured and homogeneous data not suited for ATM information with its rich schema.

ATM information cubes build on the ideas developed in our previous work [29], where we propose the use of business model ontologies for the management and summarization of rich information in OLAP cubes. In that approach, the cells of an OLAP cube are associated with entire RDF graphs, each representing knowledge that applies to a particular context.

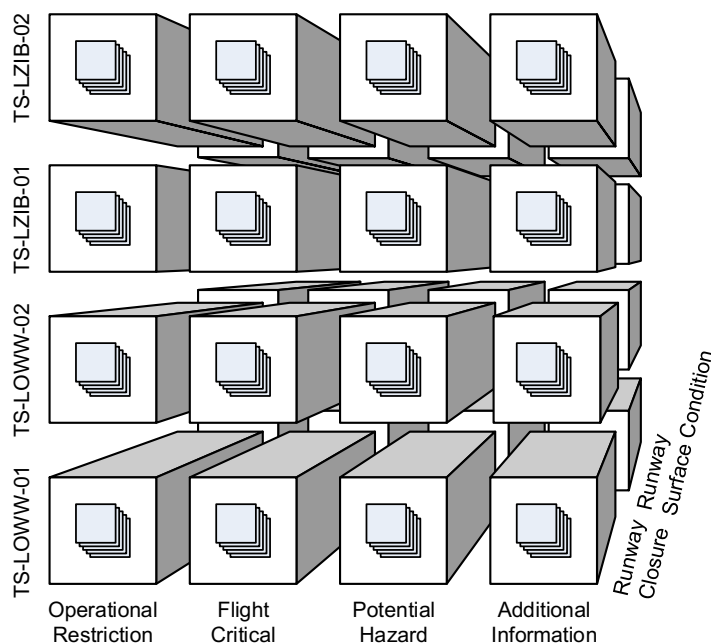
## 10.3 Data container management using ATM information cubes

A semantic container is a flexible data structure for storing ATM data items and central to our notion of ATM information cube. For the purposes of this paper, we formally define the notion of semantic container as follows.

We now arrange semantic containers in ATM information cubes along multiple dimensions (or facets) of content description. For that arrangement of semantic containers, we borrow the data cube metaphor from data warehousing and OLAP (see [26]): The dimensions span a multidimensional space where each point associates numeric values – the measures. In the case of ATM information cubes, however, the associated values are sets of ATM messages, e.g., DNOTAMs or METARs, rather than

numeric values. Each semantic container hence becomes associated with a point in a multidimensional space according to the container's semantic label.

Consider, for example, the three-dimensional ATM information cube in Figure 25. Individual DNOTAMs are collected into semantic containers along geographic, importance, and scenario dimensions. Each semantic container in that cube hence contains DNOTAMs describing a specific scenario (see [30]) for a specific geographic segment within a FIR with some importance, e.g., operational restriction or flight critical, for the flight and date which the cube has been defined for. The coordinates of a container correspond to a semantic description of the data items inside the container. For example, the point identified by *TS-LOWW-01*, *Flight Critical*, and *Runway Closure* indicates that the associated semantic container comprises all DNOTAMs about runway closures that are flight critical for the *TS-LOWW-01* transition segment. Now the attentive reader will remark two things. First, nowhere in the model has the data item type been fixed to “DNOTAM”. Second, the importance of a DNOTAM depends on many things, first and foremost on the particular flight. Yet, the specific flight is not a dimension of the cube, nor is the date or the flight phase, which also influence importance. Both of these objections could be countered by the introduction of additional dimensions: for the item type, the flight, the date, and the flight phase. The cube associates containers only with points of a certain item type, flight, or date. On the other hand, item type, flight, and date could be implicit constants that set a context for the cube. In that case, we would build a separate cube of DNOTAMs for each flight and date. The pilot could dynamically select containers of DNOTAMs along the dimensions within that context only. Furthermore, a cube of ATM information cubes could organize multiple individual cubes and explicitly represent the otherwise tacit context information (explained later).



**Figure 24. An example ATM information cube with geographic, importance, and scenario dimensions**

In order to allow for roll-up operations, i.e., viewing the contents at different levels of granularity, an OLAP cube employs hierarchically organized dimensions. In the case of ATM information cubes, the roll-up operation corresponds to a merge (see Section 10.4).

Consider, for example, the dimension hierarchies in Figure 25, which illustrates hierarchies for the dimensions of the ATM information cube from Figure 24, namely importance, geographic, and scenario dimensions. The importance dimension hierarchy follows the importance classification system for DNOTAMs from SemNOTAM [20]. The scenario dimension hierarchy follows the organization of the FAA's document on airport operations scenarios [30]. The geographic dimension hierarchy consists only of segments on transitions to airports, which are assigned to a FIR. We note that alternative roll-up relationships could be defined to allow for different geographic organization.

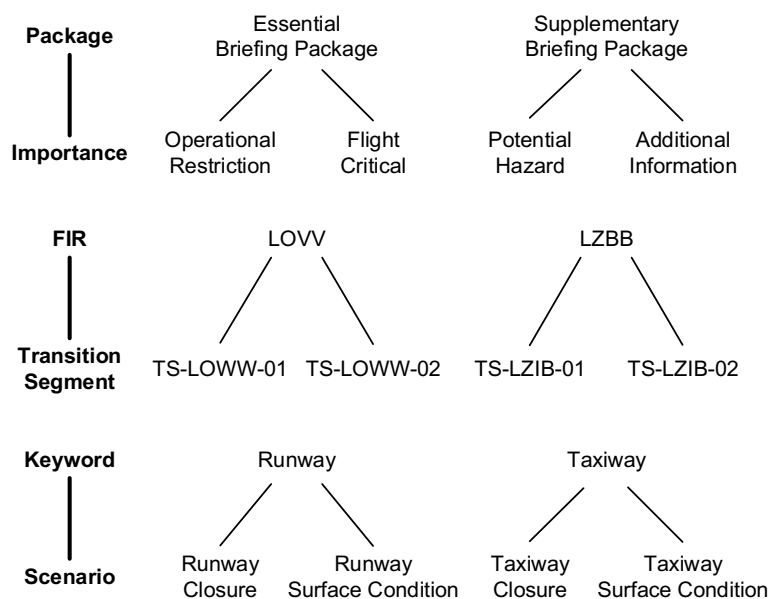


Figure 25. Example dimension hierarchies of an ATM data cube: levels (in boldface) and level members

A common issue for business intelligence applications are null values – i.e., missing, unknown, or invalid values – for dimension attributes (see [31, pp. 207-208]), which applications must handle appropriately. Otherwise, null values may result in “misleading or inaccurate results” [31, p. 208]. Similarly, when collecting ATM messages into semantic containers, e.g., along an importance dimension, in some cases, the messages may not be assigned to any one particular member, e.g., importance class (see [20]). On the one hand, there could be custom null values in the dimensions, e.g., an *unknown importance*. On the other hand, ATM messages could be directly added to a point of a higher granularity, e.g., a DNOTAM is known to be in the essential briefing package although it is not known whether the DNOTAM is flight critical or denotes merely an operational restriction. To that end, in the following, we introduce the notion of multigranular ATM information cubes.

We have defined an ATM information cube quite generally as associating semantic containers with points in a multidimensional space. Whereas the example cube in Figure 24 shows an ATM information cube that associates semantic containers only with the base granularity, i.e., the most specific granularity level, we may well imagine the existence of a multigranular ATM information cube that also associates semantic containers with coarser levels of granularity. Consider, for example, the ATM information cube in Figure 26, which illustrates a cube that associates data items explicitly with, e.g., the supplementary briefing package importance classification rather than the more specific potential

hazard or additional information importance classifications that roll up to the supplementary briefing package.

The points of an ATM information cube at non-base granularities are referred to as higher-level containers. For example, in Figure 26, the point identified by *LZBB*, *Supplementary Briefing Package*, and *Runway* has a higher-level container that contains data items associated with the coarser *FIR*, *package*, and *keyword* granularity.

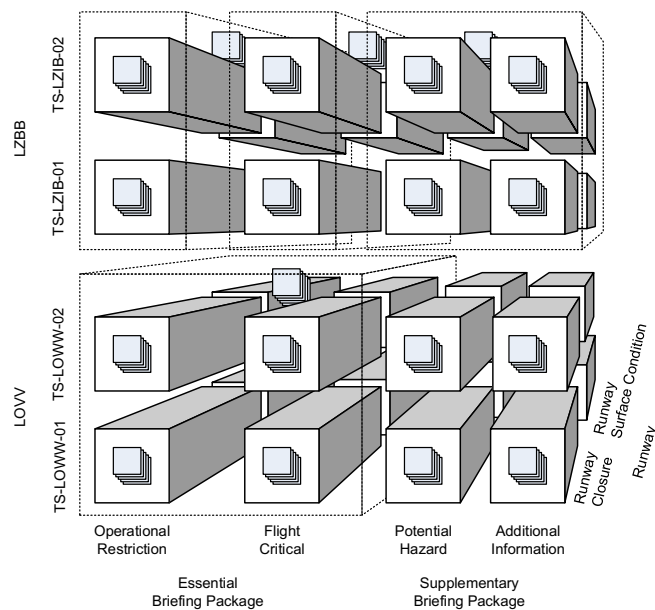


Figure 26. A multigranular ATM information cube

If some message is flight critical for a segment of the *LOVV* region then, all other things remaining unchanged, that message is also in the essential briefing package for the entire *LOVV* region.

We imagine ATM information cubes being built for a certain operational context, e.g., a specific flight on some date. In the previous sections, a cube's operational context was nowhere explicitly defined in the model. Thus, in order to externalize that context, we propose to arrange the ATM information cubes themselves into multidimensional structures (see Figure 27).

A cube of ATM information cubes (or *metacube*) hence consists of several cubes, the sets of dimensions of which will typically overlap but not necessarily be equal. A drill-across operator allows to combine the different cubes, joining via the common dimensions, with all non-common dimensions considered rolled up at the implicit *all* level. While the dimensions can be manifold, we assume flight, item type, and date/time as the typical candidates for dimensions. A point in such a metacube may contain, e.g., a cube of DNOTAMs relevant for flight OS93 on 15 March 2018.

A pivoting operator allows to obtain a self-contained ATM information cube from a metacube by adding the metacube dimensions to the component cube. For these added dimensions, the dimension member is fixed to the respective points from the metacube. For example, the DNOTAM cube for OS93 on 15 March 2018 associates containers only with points that have a value *DNOTAM* in the item type dimension, *OS93* in the flight dimension, and *15 March 2018* in the date dimension.

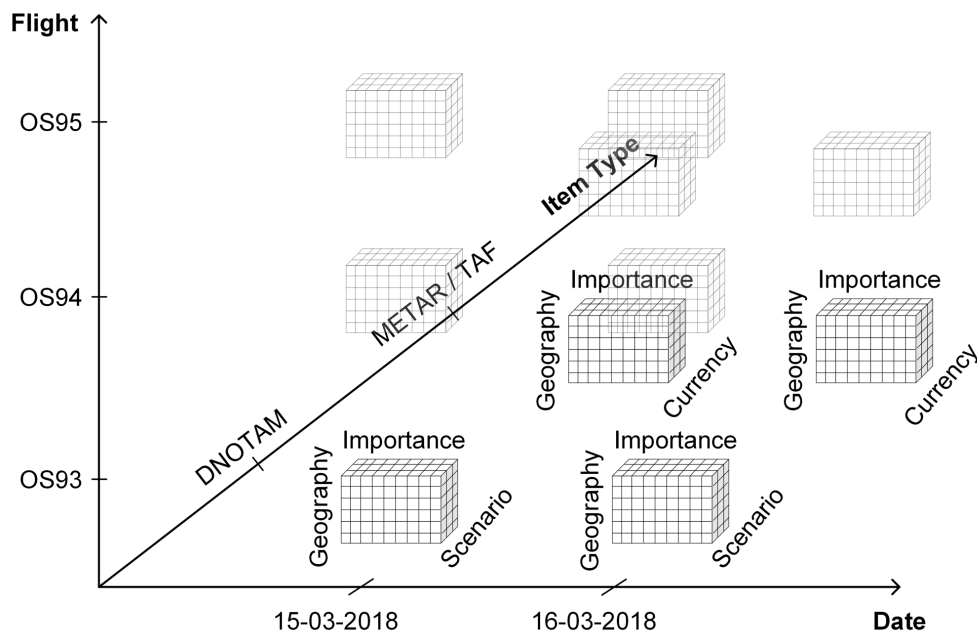


Figure 27. A multigranular ATM information cube

## 10.4 Merge of semantic containers

The organization of semantic containers into ATM information cubes allows for a wide array of possibilities with respect to container merging. We distinguish a *merge-select*, a *merge-union*, and a *merge-intersection* operator. We note, however, that other forms of merge operators may also be conceived in the future.

Intuitively, the merge-select operator applied on a cube  $q$  with a set of argument coordinates returns a single flat container of data items from the containers at the argument coordinates of the input cube  $q$ . The merge-select operator, however, is not closed with respect to ATM information cubes since the operator takes a cube as input and returns a container as output.

The merge-union and merge-intersection operators are closed with respect to ATM information cubes, i.e., take a cube as input and return a cube as output. Both operators receive a granularity level  $g$  as input and return a new cube  $q'$  that has the granularity level  $g$  as the base granularity. The containers at the base granularity  $g$  of the new cube  $q'$  are all flat, i.e., elementary and non-composite, and contain data items from the containers of the input cube  $q$  below the new base granularity  $g$ . In the case of merge-union, the operator just selects all the data items. In the case of merge-intersection, the operator only selects the data items contained in each of the merged containers. We suppose the merge-union operator is the more common for ATM information cubes.

Figure 28 illustrates the merge-union operator. The depicted cube has a coarser base granularity than the input cube as seen in Figure 24. The associated containers are elementary. In case the merge operator was applied on a cube with containers of different item types and the elementary containers must consist of a single item type, the application of the merge operator must be restricted to containers of the same item type (elementary or homogeneous composite).

In OLAP cubes from traditional data warehousing, the “double-counting problem” [26] leads to summarizability issues [32]. In that case, counting the same numeric measure twice yields erroneous results. In the case of ATM information cubes, the double-counting problem does not pose an issue. We may well imagine the same DNOTAM being included in multiple containers. The merge-union operation would then simply ignore the duplicate data items when compiling the merged semantic container.

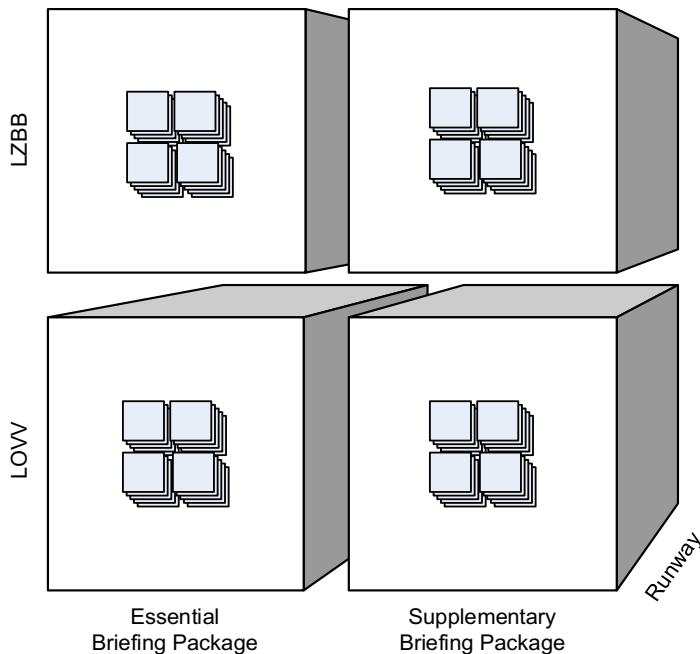


Figure 28. A merge of the semantic data containers from Figure 24 along the geographic, importance, and scenario dimension hierarchies from Figure 25

## 10.5 Abstraction of data items

The notion of *abstraction* serves as an umbrella term for a wide variety of different operations. Abstraction, as opposed to merge, is a kind of operations that alter the individual data items and their relationships to each other. Originally proposed for RDF data (see [29]), the principle of abstraction is independent from any concrete data or information model.

We employ UML object diagrams to illustrate the principle of the abstraction operation. First, consider an object diagram that illustrates DNOTAMs according to the AIXM information model (Figure 29). Note, however, that for simplicity's sake, we adapt AIXM and disregard the AIXM temporality concept (see [33]). The object diagram then shows two DNOTAMs about the surface conditions of runways, and two DNOTAMs about the closure of runway directions. The *LOWW-16/34* runway has two layers of contaminants with an overall extend of 0.32 m: dry snow (0.29 m) and ice (0.02 m). The *LOWW-11/29* runway has a layer of ice with an extent of 0.01 m. Both runways, however, have a specified length and width already cleared of contaminants while the remainder of the runway has winter

services going on, thus leading to the closure of one runway direction from each runway. The closures are due to snow and ice removal, respectively, and for each runway affect only one runway direction whereas the other remains open.

Assume the DNOTAMs in Figure 29 concern the destination airport of a particular flight. Then, a management summary suitable for the preparation and early phases of that flight might only show a single, abstracted DNOTAM that alerts the pilot that wintery conditions await at the destination airport. Consider, thus, the abstracted DNOTAM information in Figure 30, which is the result of applying some abstraction operator that we do not further specify; we leave the specification of useful abstraction operators for the ATM domain to future work. The resulting DNOTAM notes only wintery contaminant for a generic *LOWW-Runway*, and alerts of runway closure due to winter services. Summary data are given of the attributes such as the cleared length and width of the contaminated runways. While the abstracted DNOTAM has only one generic runway contamination, the *count* attribute provides documentation of the number of runway contaminations in the original model.

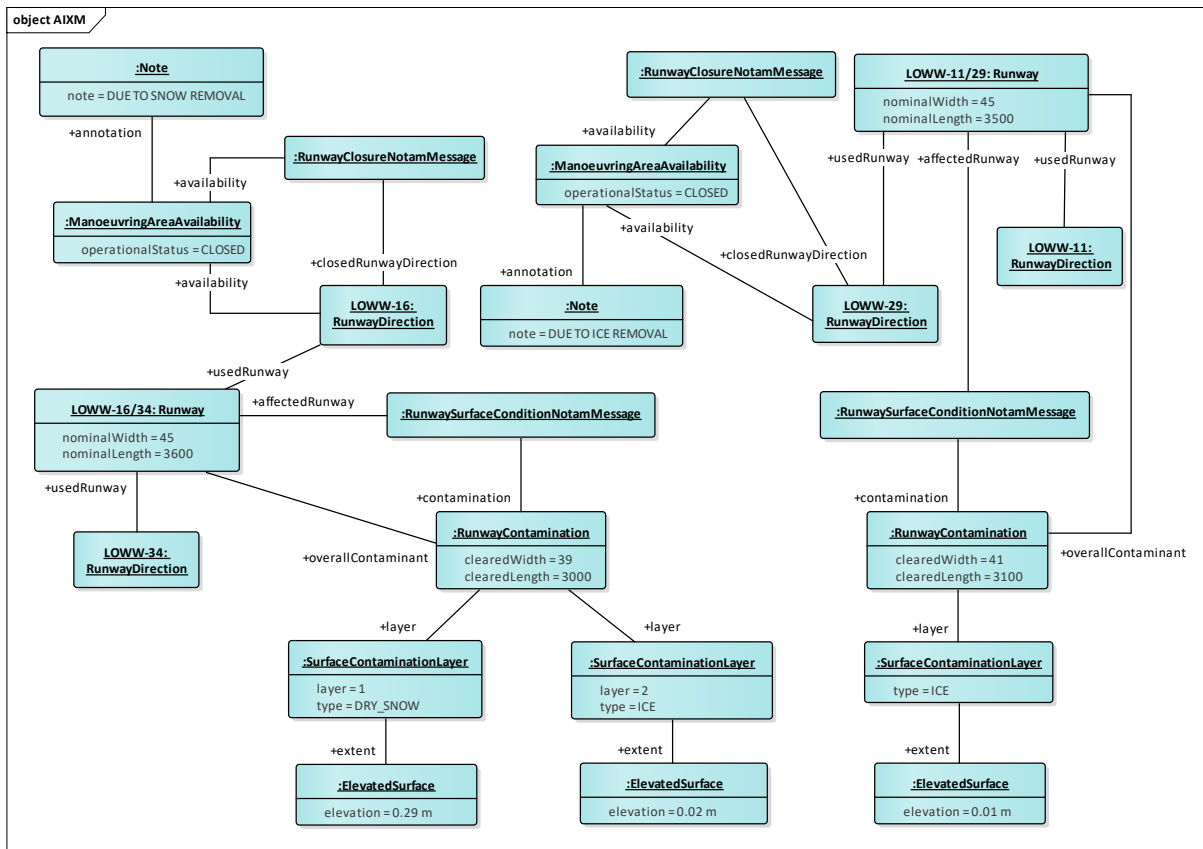


Figure 29. An object diagram of DNOTAMs, simplified and adapted from the AIXM information model [1], disregarding the temporality concept, with DNOTAM messages classified according to the FAA airport operations scenarios [2]

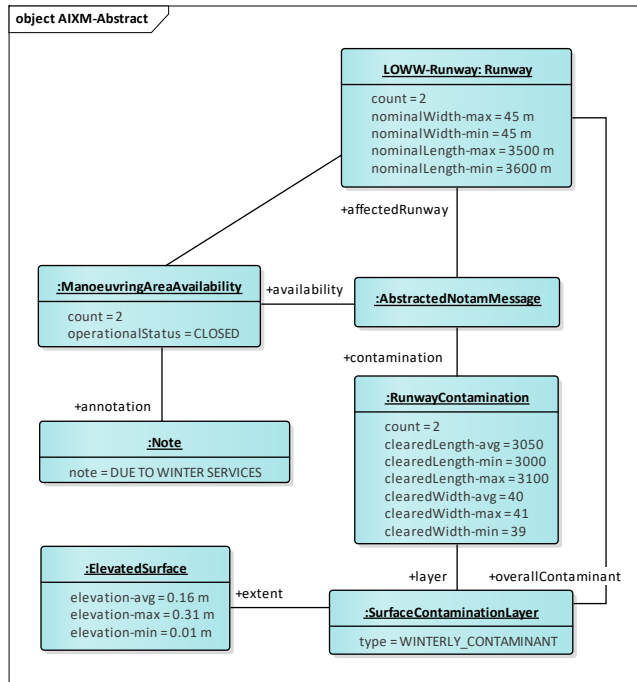






Figure 30. An example of abstracted DNOTAM information obtained from the DNOTAM information in Figure 29



The BEST consortium:

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