



# D3.2 Prototype SWIM-enabled Applications

**D3.2**

**BEST**

**Grant:**

699298

**Call:**

H2020-SESAR-2015-1

**Topic:**

Sesar-03-2015

Information Management in ATM

**Consortium coordinator:**

SINTEF

**Dissemination Level:**

PU

**Edition date:**

08 May 2018

**Edition:**

01.01.04 Final Version

Founding Members



**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
Eduard Gringinger (FREQUENTIS)	Project Member	2018-04-30
Christoph Fabianek (FREQUENTIS)	Project Member	2018-04-30
Christoph Schütz (LINZ)	Project Member	2018-04-30

### Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Scott Wilson (EUROCONTROL)	Project Member	2018-05-07
Bernd Neumayr (LINZ)	Project Member	2018-05-08
Michael Schrefl (LINZ)	Project Member	2018-05-07
Audun Vennesland (SINTEF)	Project Member	2018-05-04

### 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	2018-05-14

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

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

### Document History

Edition	Date	Status	Author	Justification
00.00.01	2017-05-26	PCOS	Gringinger	PCOS review
00.01.01	2017-10-17	Intermediate proposed	Gringinger/Fabianek	Intermediate draft
00.01.03	2017-12-22	Intermediate Approved	Gringinger/Neumayr	Intermediate review
01.00.00	2018-03-30	Final draft	Gringinger/Fabianek	Scenario defined
01.01.03	2018-04-30	Final internal review	Gringinger/Fabianek	Final version for internal review
01.01.04	2018-05-08	Final version	Gringinger/Neumayr	Input from 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

---

This deliverable describes the exploratory development (TRL 1) performed in the exploratory research project BEST. The document gives a quick overview about the Semantic Container concept defined in D2.1 & D2.2 and briefly introduces the use cases defined in D3.1. In the motivation chapter an analysis of the actual information that is distributed via SWIM is given. This should underline the mentioned benefits in D3.1. The experimental prototypes developed are discussed in a separate chapter. In the end a full SWIM scenario with Semantic Containers is shown. This includes an integration of the Semantic Containers into the Frequentis SWIM registry, the 2<sup>nd</sup> evaluation of the exploratory prototype, the integration of the Semantic Containers into the Frequentis Integration Platform (MosaiX) and the integration of a SWIM application using Semantic Containers (the Frequentis SESAR 1 Integrated Digital Briefing prototype). This document is public, but the source code of the software is not.

# Table of Contents

Abstract/Executive Summary.....	3
<b>1 Introduction: About this document .....</b>	<b>5</b>
1.1 Purpose .....	5
1.2 Intended Readership .....	5
1.3 Relationship to other deliverables.....	6
<b>2 Background .....</b>	<b>7</b>
<b>3 Semantic Container: Analyses &amp; Motivation.....</b>	<b>8</b>
<b>3.1 SWIM Information Analyses .....</b>	<b>8</b>
3.1.1 Pilot Briefing for a Flight from Germany to Austria.....	11
3.1.2 Information Needs of a Fuelling Service at an Airport .....	13
3.1.3 Airline Managing its Fleet.....	14
3.1.4 All flight data for a flight from Sydney to Vienna via Dubai.....	15
3.1.5 Running the Network Manager Operations Centre .....	16
<b>3.2 Evaluation of Semantic Container Deployment .....</b>	<b>16</b>
<b>4 Semantic Container: Experimental Prototypes.....</b>	<b>18</b>
<b>4.1 Experimental Prototype Evolution 1: Semantic Container Management Platform.....</b>	<b>18</b>
4.1.1 Setting up a DNOTAM Container .....	19
4.1.2 Conclusion .....	26
<b>4.2 Experimental Prototype Evolution 2: Semantic Container Management Platform.....</b>	<b>27</b>
4.2.1 Frontend.....	30
4.2.2 Interface .....	40
4.2.3 Conclusion .....	54
<b>5 Semantic Container: Scenario.....</b>	<b>55</b>
<b>5.1 BEST Integration: SWIM Registry .....</b>	<b>56</b>
<b>5.2 Experimental Prototype Evolution 2: Semantic Container Management System .....</b>	<b>59</b>
<b>5.3 BEST Integration: SWIM Integration Platform .....</b>	<b>62</b>
<b>5.4 BEST integration in a SWIM Application: Integrated Digital Briefing .....</b>	<b>63</b>
5.4.1 SESAR 2020 Pj17.01 EXE 1 .....	68
<b>6 Conclusion and Future Work .....</b>	<b>69</b>
<b>7 References.....</b>	<b>70</b>
<b>8 Table of Figures .....</b>	<b>72</b>
<b>9 Abbreviations .....</b>	<b>74</b>
<b>10 APPENDIX A: Tools &amp; Libraries for Semantic Container Management System .....</b>	<b>77</b>
<b>11 APPENDIX B: Integrated Digital Briefing.....</b>	<b>79</b>

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

---

BEST is a TRL1 project where “*basic principles are observed*” [1]. The experimental prototypes developed and integrated for demonstrating the use cases of BEST described in D3.1 [2] are beyond this TRL level. The prototypes assume that SWIM is implemented and ready to use within the European ATM network. The main goal of this document is to show the possibilities and benefits of the BEST Semantic Container concept as outlined in Deliverable D2.1 [3] and D2.2 [4] in a first proof of concept (e.g.: TRL level 3 “*experimental proof of concept*”).

The dissemination level of this deliverable is shown as “PU” (Public), in accordance with table WT2 in section 1.3.2 of Part A of the Grant Agreement. That refers to the information provided in this report. However, the table shown in section 3.4 “IPR Management” of Part B of the Grant Agreement shows the “IPR Principle” for result R1 as being “Closed source software”. Result “R1” forms part of the result numbering scheme used in Part B, and refers to D3.2. Thus: the source code of the software that is described in this document is not public, and will not be made available to any third parties. Executable versions of the software can be made available on request to anyone interested in experimenting with the software.

## 1.1 Purpose

The primary focus of BEST work package 3 is to develop experimental prototypes to show the benefits of the BEST Semantic Container concept based on the use case scenarios, derived from generic use cases, described D3.1 [2]. To achieve this, the deliverable illustrates the capabilities of BEST Semantic Containers used by experimental prototypes in a SWIM enabled environment. The prototype enables experimentation and evaluation of using semantic technologies in a SWIM enabled environment. The Grant Agreement describes the content of this deliverable as follows:

3.2 Task Description: “*The aim of this task is to develop a set of experimental prototype applications that will demonstrate how the specifications and developments in WP 1 and WP 2 can contribute to realise benefits of semantic technologies in a SWIM setting. The prototypes will be based on the use case scenarios defined in Task 3.1.*”

3.2 Deliverable Description: “*This deliverable is provided as a set of proof-of-concept prototypes that will demonstrate the potential of the semantic-based approach used in BEST. The prototypes will implement the ontological structures developed in WP 1 and the decentralised SWIM Data and Knowledge Base from WP 2.*”

## 1.2 Intended Readership

The intended readerships of this document are operational people that will benefit from SWIM, SWIM developers, relevant SWIM industry and future SWIM (data) providers (e.g.: ANSPs, Airlines, Weather Service, stakeholders outside of the ATM world that could benefit from SWIM, etc.).

---

<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

The experimental prototypes demonstrate the wide scale impact of the BEST Semantic Container concept from the set of Use Cases described in D3.1 [2]. The Prototypes developed are using semantic components specified and developed in WP 1 (Ontology Development and Compliance Validation) and WP 2 (Semantic Container Management).

For example, the exchanged information identified in the use cases enable to identify which exchange information model were used within D1.1 [5] and which granularity of the ontology modules is necessary to support the use cases besides technological aspects. The use cases (D3.1) provided input for the composition of Semantic Containers defined in D2.1 [3], D2.2 and influenced this deliverable.

Deliverable	Relationship
D1.1 Experimental ontology modules formalizing concept definition of ATM data	The ontology modules developed in D1.1 can serve as the fundamental for the faceted ontology-based description of Semantic Containers. The ontology modules developed in D1.1 are used for the baseline for the use case scenarios.
D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base	The definitions applied in D2.1 in order to initialize the Semantic Container approach are essential for this deliverable to show the possibilities of Semantic Containers and their benefits.
D2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment	In D2.2 defines the Semantic Container approach by 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 which are used for the Semantic Container Management Platform.
D3.1 Prototype Use Case Scenarios	The use cases defined D3.1 demonstrate practicality of the Semantic Container approach in a SWIM setting.
D4.4 Tutorial for Software Developers	The tutorial describes how software developers can use ontologies to create things like the Semantic Container.
D5.1 Scalability Guidelines for Semantic SWIM-based Applications	While we conduct experiments concerning principal feasibility, D5.1 formally investigates scalability aspects of the Semantic Container approach.
D5.2 Ontology Modularisation Guidelines for SWIM	The guideline describes how to develop ontology modules for the Semantic Container approach. Without the modularisation the Semantic Container approach would not be feasible.

## 2 Background

---

In this chapter, background information about the Semantic Container concept and the use cases are briefly presented. Furthermore, the motivation of the Semantic Container distribution for SWIM is explained in chapter 3. The Semantic Container approach, as defined in D2.1, D2.2 and D3.1, is a flexible way of compartmentalizing ATM information which complements the service-oriented architecture of SWIM with techniques for ontology-based data description and discovery [6]. Ontology-based reasoning facilitates the matching of end user's or application's information needs with available Semantic Containers.

A Semantic Container is a data product with content and description. The content is a set of data items (i.e., messages) 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 [7]. A Semantic Container should contain all those 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. Semantic Containers are consumed and produced by SWIM services and applications and can be arranged in derivation chains [8]. The lineage of value-added information is tracked along derivation chains of Semantic Containers. The packaging of information into Semantic Containers allows for the caching of information and its subsequent discovery for later re-use.

ATM information packaged into Semantic Containers can be stored redundantly on different server nodes for increased availability. The metadata expressed using semantic technologies allows for the replication of information and the subsequent discovery and re-use in a distributed environment. A Semantic Container may also derive from other containers, combining the information represented in these containers. The semantic description of information allows for the updating of such combinations of containers in a distributed environment where different services produce and update the source information. The semantic description may also be beneficial for deciding where to allocate information in a distributed SWIM environment.

The concept of Semantic Containers focuses on enriching data by collecting individual data items into sets of data items labelled with semantic metadata about, for example, freshness, quality, localization, and time. SWIM applications may use this information to be more efficient. Generic filtering and clustering of SWIM data will help SWIM developers to reduce redundancies. Collections of messages based on the established standards AIXM, IWXXM and FIXM, such as DNOTAMs, TAFs, METARs, SIGMETs and flight plans, are prepared as BEST Semantic Containers with semantic labels, which can be further processed by applications. Future SWIM applications will only need to focus on the necessary operation-specific filtering and prioritizing of the data, based on operational rules. The concept enables a generic way of filtering according to temporal, spatial, and other semantic aspects such as the quality of data and freshness.

Based on the above description of Semantic Containers a few use cases are evaluated in this document. We investigate the applicability of the used technologies and possible benefits regarding technical and organizational consequences. Chapter 3 describes several use cases and estimates outcomes/benefits for the respective scenario while chapter 4 describes the Semantic Container: Experimental Prototypes. In chapter 5 a detailed use case for a flight from Boston, MA (KBOS) to Vienna (LOWW) is described step-by-step, demonstrating the developed prototype as well as adjacent systems and their interplay.

## 3 Semantic Container: Analyses & Motivation

---

With respect to data distribution as described in D2.2 [4], the benefits of the Semantic Container approach for the SWIM concept are “*defined quality of information, high availability of information, and decreased network load*” as described in section 3.6 in D3.1 [2]. This chapter describes the motivation for the Semantic Containers starting with an analysis of the actual information that is distributed via SWIM to underline the mentioned benefits. In the final section the Semantic Container distribution evaluation outcome is discussed.

### 3.1 SWIM Information Analyses

This section addresses questions regarding benefits (incl. storage and bandwidth requirements) when using Semantic Containers on top of the SWIM platform. It lists relevant messages, investigate their average size and makes assumptions about typical message quantities. With those assumptions established various scenarios are compared when using Semantic Containers vs. legacy services.

The following message types are used in the next scenarios:

- Meteorological data: METAR, TAF, SIGMET
- NOTAM
- FPL
- Aeronautical information (Airport information and FIR information)

For each data type the following information is relevant – see Table 1:

- *locality*: on which geographical level (continent, region, country, airport) is the information made available, processed, and consumed
- *provider*: who is the legal entity making the message currently available
- *quantity*: how many messages are typically available at any time (the maximum number of the 3rd quartile is used)
- *size*: the average size of a message in Byte
- *storage*: product of quantity and at the respective location times message size
- *update frequency*: how often are all messages updated at least once on average
- *bandwidth*: minimum required bandwidth to update all messages within the update frequency (i.e., storage divided by update frequency in seconds)

Assumptions for data in Table 1:

- we assume a request/response pattern to pull information from sources; however, in SWIM it is possible to subscribe for updates and data providers push new messages as soon as they become available; timeliness of information will therefore be increased but storage requirements and bandwidth assumptions are valid for both scenarios
- only the raw message size is given; when using a mark-up language (e.g., XML) a factor for catering the syntactic overhead needs to be considered



	<i>provider</i>	<i>quantity</i>	<i>message size</i>	<i>storage</i>	<i>update frequency</i>	<i>bandwidth</i>
<b>METAR</b>			70 Bytes		30 minutes	
Europe	Eurocontrol	10.500		735 KB		408 Bytes/s
US	FAA	10.000		700 KB		389 Bytes/s
Germany	DFS	1.078		75 KB		42 Bytes/s
Frankfurt		2		140 Bytes		-
Austria	Austro Control	116		8 KB		-
Schwechat		2		140 Bytes		-
<b>TAF</b>			150 Bytes		6 hours	
Europe	Eurocontrol	21.500		3,2 MB		150 Bytes/s
US	NOAA	6.000		900 KB		42 Bytes/s
Germany	DFS	2.150		320 KB		15 Bytes/s
Frankfurt		4		600 Bytes		-
Austria	Austro Control	232		34 KB		2 Byte/s
Schwechat		4		600 Bytes		-
<b>SIGMET</b>			650 Bytes		20 minutes	
Europe	Eurocontrol	80		52 KB		48 Bytes/s
US	NOAA	150		98 KB		90 Bytes/s
Germany	DFS	10		6,5 KB		6 Bytes/s
Austria	Austro Control	2		1,3 KB		1 Bytes/s
<b>NOTAM</b>			300 Bytes		4 hour	
Europe		20.000		6 MB		417 Bytes/s
US		20.000		6 MB		417 Bytes/s
Germany	DFS	2.500		750 KB		52 Bytes/s
Austria	Austro Control	300		90 KB		6 Bytes/s
<b>FPL</b>			500 Bytes		1 hour	
Europe		29.000		14,5 MB		168 Bytes/s
US		42.700		21,4 MB		247 Bytes/s
Germany	DFS	8.000		4,4 MB		51 Bytes/s
Austria	Austro Control	3.000		1,5 MB		17 Bytes/s
<b>Airport Information</b>			10 MB		1 year	
Europe		4.500		45 GB		1,4 kB/s
US		5.000		50 GB		1,6 kB/s
Germany	DFS	539		5,39 GB		171 Bytes/s
Austria	Austro Control	58		580 MB		18 Bytes/s
<b>FIR</b>			1 MB		1 year	
Europe		118		118 MB		4 Bytes/s
US		22		22 MB		1 Bytes/s
Germany	DFS	5		500 KB		-
Austria	Austro Control	1		100 KB		-

**Table 1:** Message Types and Attributes exchanged via SWIM

Sources for data in Table 1 are gathered from various official sources and where not available public via Wikipedia:

- Message size for weather information:  
<https://aviationweather.gov/adds/dataserver/current>  
message size for METAR, TAF and SIGMET was calculated based on average message size on Feb 19, 2018 using 4.952 METAR message, 2.797 TAF messages, 79 SIGMET messages
- Assumption for Europe if numbers are not available: 10x Germany (for simplification)
- Number of NOTAMs:
  - 20.000 active NOTAMs: <http://www.eurocontrol.int/articles/digital-notam-phase-3-p-21>
  - about 2m NOTAMs issued by FAA in 2016:  
<https://www.faa.gov/files/gslac/library/documents/2017/Sep/135837/NOTAM%20101%20Back%20to%20Basics.pdf>
- Number of flight plans:
  - [https://www.dfs.de/dfs\\_homepage/de/Unternehmen/Zahlen%20und%20Daten/Statistiken/ATFM\\_Report\\_year\\_2017.pdf](https://www.dfs.de/dfs_homepage/de/Unternehmen/Zahlen%20und%20Daten/Statistiken/ATFM_Report_year_2017.pdf)
  - [https://www.dfs.de/dfs\\_homepage/de/Unternehmen/Zahlen%20und%20Daten/Statistiken/LVS\\_01\\_2018.pdf](https://www.dfs.de/dfs_homepage/de/Unternehmen/Zahlen%20und%20Daten/Statistiken/LVS_01_2018.pdf)
  - [https://www.bmvit.gv.at/verkehr/gesamtverkehr/gvp/faktenblaetter/international/fb\\_luftsektor.pdf](https://www.bmvit.gv.at/verkehr/gesamtverkehr/gvp/faktenblaetter/international/fb_luftsektor.pdf)
- Number of airports:
  - [https://en.wikipedia.org/wiki/List\\_of\\_airports\\_in\\_Austria](https://en.wikipedia.org/wiki/List_of_airports_in_Austria)
  - <https://www.cia.gov/library/publications/the-world-factbook/geos/gm.html>
- Number of FIR regions:
  - <http://gis.icao.int/gallery/ICAOFIR2015linkWEB2.htm>
  - [https://en.wikipedia.org/wiki/Flight\\_information\\_region](https://en.wikipedia.org/wiki/Flight_information_region)

The following entities are considered in developing scenarios:

- **Airports:** relevant data to handle flights to and from the airport and service providers operating there
- **Airlines:** data for managing all planes and flights
- **Airplanes:** documenting preparation and operation of a flight
- **Network Manager:** all relevant data (weather, flight plans) for ongoing and planned flights in the next 24 hours

For each scenario the following steps are performed:

- list required data sources
- describe container layout involved (i.e., which institutions are operating containers and what are the data flows between those containers)
- compare scenario with Semantic Containers vs. legacy services

The following scenarios are investigated in regard to storage and bandwidth requirements:

- Pilot Briefing for a flight from Germany to Austria
- Information needs of a fuelling service at an airport
- Airline managing its fleet
- All flight data for a flight from Sydney to Vienna via Dubai
- Running the Network Manager Operations Center

### 3.1.1 Pilot Briefing for a Flight from Germany to Austria

#### Data Sources

- Austro Control provides the following information for the Austrian area:
  - METAR, TAF and SIGMET messages
  - NOTAMs
  - Austrian FIR and airport information
- DFS provides the following information for the German area:
  - METAR, TAF and SIGMET messages
  - NOTAMs
  - German FIR and airport information

#### Container Layout

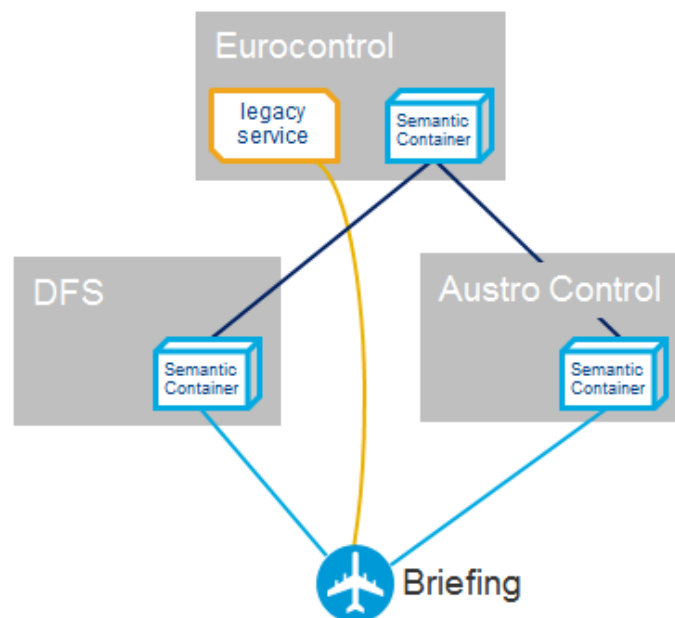


Figure 1: Container Hierarchy for a Flight from Germany to Austria

#### Behaviour of legacy services

Legacy services are compared to Semantic Containers in respect to:

- **Storage:** size of containers at each level / SWIM repository is the sum of the respective types in Table 1
- **download:** is the minimal overall bandwidth (from Table 1) necessary to keep all messages up to date, i.e., downloading messages into the container or service
- **upload:** is the minimal overall bandwidth (from Table 1) necessary from a data provider to serve all requests

Additionally, the column “Messages” provides the number of relevant messages at a given time for the briefing application.

	Messages	Storage	Download	Upload
<b>Eurocontrol</b>	56.698	45,1 GB	2,5 kB/s	17,5 MB/s
<b>Briefing application</b>	122	52 MB	-	n.a.

Table 2: Behaviour of Legacy Services

Comments:

- **Eurocontrol:** the 45 GB storage size stems from 10MB airport information per airport – all numbers are based on data data given in Table 1
- **Eurocontrol Download:** bandwidth required to keep the data (provided in the column “Storage”) current - since this is mostly static information only a low download bandwidth is necessary
- **Eurocontrol Upload:** is the product of the size of a typical briefing times the average numbers of flight plans in Europe – a rather high bandwidth requirement that illustrates the bottleneck of the legacy service architecture with an increasing number of data services
- **Briefing application:** a briefing for a typical flight processes 122 messages on average and holds about 50 MB of data; the limiting factor to download those ~50 MB is available bandwidth and upload speed of the service provider; the briefing application as service consumer does not upload any data (therefore, “n.a.” in the upload column)

#### Benefits of Semantic Containers

	Messages	Storage	Download	Upload
<b>Eurocontrol</b>	56.698	45,1 GB	2,5 kB/s	122,6 kB/s
<b>DFS</b>	6.282	5,4 GB	0,3 kB/s	5,3 MB/s
<b>Austro Control</b>	709	581 MB	32 Bytes/s	1,8 MB/s
<b>Briefing application</b>	122	52 MB	-	n.a.

Table 3: Benefits of Semantic Containers

Comments:

- **Eurocontrol Upload:** the most significant difference between the legacy service architecture and usage of Semantic Containers is the reduced upload requirements for a central authority like Eurocontrol: 123 kB/s with Semantic Containers vs. 17 MB/s with legacy services; the necessary upload for Eurocontrol is the product of download bandwidth times the number of European countries that operate a Semantic Container (assumption: 50) – instead of serving each briefing application the load is distributed to the national authorities (like DFS and Austro Control) and then those authorities serve data for the service consumers (e.g., a briefing application)
- **Eurocontrol Messages & Storage:** about 57.000 messages are relevant at a given point in time and the 45 GB storage size stems from 10MB airport information per airport; since this is mostly static information only a low download bandwidth is necessary to keep the data current
- **DFS / Austro Control upload:** is the product of the size of a typical briefing (storage of briefing application) times the average number of flight plans in the respective country

- **Briefing application:** message numbers and storage requirements are of course the same as with the legacy service since the introduction of Semantic Containers is completely transparent to service consumers

### 3.1.2 Information Needs of a Fuelling Service at an Airport

#### Data Sources

- DFS provides FPL for German area
- Fraport AG provides airport information

#### Container Layout

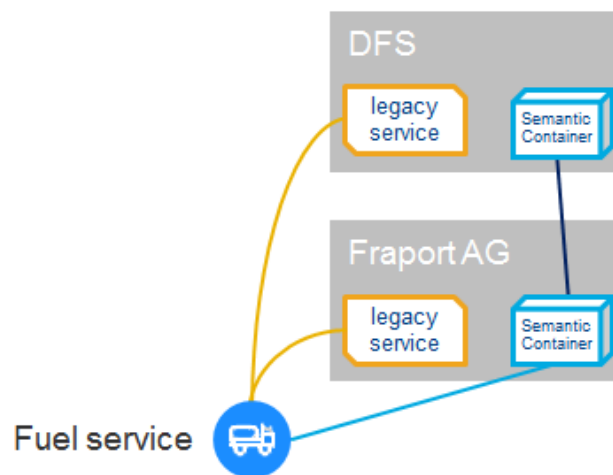


Figure 2: Container Hierarchy of a Fuelling Service at an Airport

#### Behaviour of legacy services

- multiple legacy interfaces
- data inconsistencies possible

#### Benefits of Semantic Containers

- single Interface with defined data quality
- in sync with other services on the airport

### 3.1.3 Airline Managing its Fleet

#### Data Sources

- Eurocontrol, FAA and DFS provide the following messages:
  - FPL
  - METAR, TAF and SIGMET messages
  - NOTAMs
  - FIR and airport information
- Global Weather provides
  - METAR, TAF and SIGMET messages
- Fraport AG provides
  - airport information

#### Container Layout

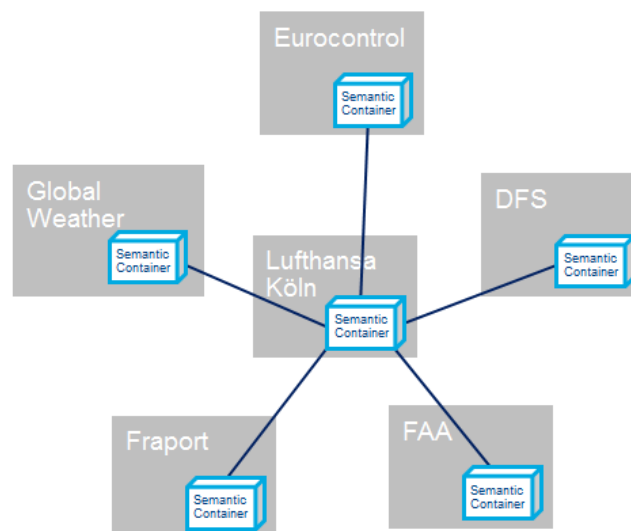


Figure 3: Container Hierarchy for an Airline

#### Behaviour of legacy services

- individual solution based on legacy interfaces

#### Benefits of Semantic Containers

- standardized handling of all data to
  - merge information from various sources
  - use most up-to-date data
  - have “single truth” within organisation

### 3.1.4 All flight data for a flight from Sydney to Vienna via Dubai

#### Data Sources

- Airservices Australia, General Civil Aviation Authority, and Eurocontrol provide the following messages:
  - METAR, TAF and SIGMET messages
  - NOTAMs
  - FPL
  - FIR and airport information

#### Container Layout

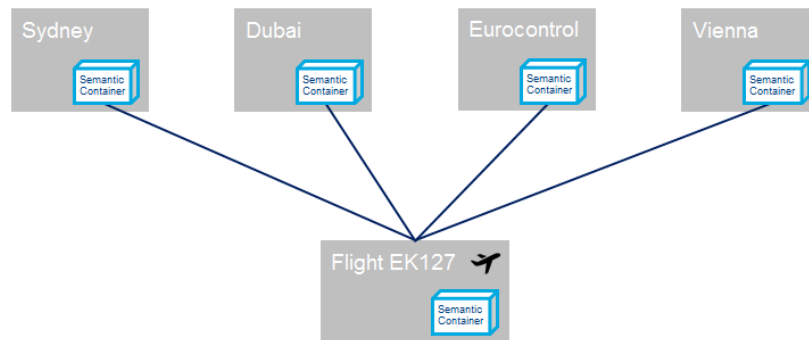


Figure 4: Container Hierarchy for a Flight from Sydney to Vienna via Dubai

#### Behaviour of legacy services

Not available right now

#### Benefits of Semantic Containers

- information available on board for instant scenario evaluation
- complete audit trail

### 3.1.5 Running the Network Manager Operations Centre

#### Data Sources

- Eurocontrol provides the following messages:
  - FPL

#### Container Layout

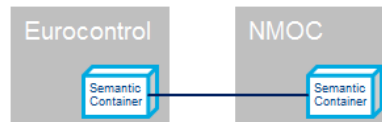


Figure 5: Container Hierarchy for Network Manager Operations Centre

#### Behaviour of legacy services

- tight integration with Eurocontrol services

#### Benefits of Semantic Containers

- established data flows in complex organizations can be represented with Semantic Containers but do not yet provide much benefits

## 3.2 Evaluation of Semantic Container Deployment

In the above sections we discussed possible application scenarios for semantic containers as well as benefits and shortcomings in different areas. In particular, we evaluated the following aspects:

- **Network Traffic:**  
Describe an example topology of a network between data consumer and provider;  
calculate number of messages and bandwidth consumption between nodes
- **Response Time**  
Time impact of caching mechanisms in Semantic Containers for providing data to SWIM applications
- **Storage Requirements**  
Necessary storage on nodes in the network that host Semantic Containers
- **Organizational Consequences**  
Process implications when implementing BEST on top of SWIM

Besides the benefits and shortcomings in the individual scenarios the following list summarizes general benefits of the Semantic Container approach:

- **Decoupling of Services**  
Semantic containers decouple information consumers from information service providers and in this way make it easier to replace and maintain SWIM components. For example, an information consumer needs information in a particular message format as provided by some semantic container. When the information service that acts as source of the semantic container is updated to a new message format, the provenance of the semantic container will be changed to also call a transformation service which will translate the messages to the old format. In this way, the information consumer is shielded from changes in the information service's output format. As soon



as the information consumer is ready for the new message format a new semantic container is created with content in the new message format.

- **Optimization for message distribution**

Data provider in the SWIM context process many requests from different applications. Table 3 gives an indication about the relevant message quantities involved in generating the pilot briefing for a single flight. With Semantic Containers providers can package and compress those usually small messages to a single response and deliver the necessary data in a more efficient way than answering a large number of small requests from various applications. This in turn improves reliability of the overall network and increases response times for SWIM applications.

- **Easier Testing and Monitoring of End-to-End Workflows in SWIM Networks**

Semantic Containers can act as black boxes in a SWIM network and allow shielding functionalities behind. When testing a new data provider or consumer a Semantic Container act as a single interface with defined behaviour and thus allow a wide range of tests in a realistic environment. It would also be possible to record data traffic over a time period and then replay this traffic in a test scenario. Additionally, Semantic Containers occupy critical nodes in a SWIM network and allow therefore monitoring data traffic at the relevant points.

## 4 Semantic Container: Experimental Prototypes

This chapter explains the experimental prototypes developed as a proof-of-concept of the Semantic Container concept. It is split into two sections describing the different experimental prototypes that have been developed.

### 4.1 Experimental Prototype Evolution 1: Semantic Container Management Platform

The first experimental prototype covers only DNOTAM sets and allows creating, updating, and deleting of Semantic Containers. The goal was to implement the use cases described by D3.1 [2] in section “3.1 Use Case: Aeronautical Information” and “3.3 Use Case: Digital NOTAM”. It was based on the initial (big and monolithic) ontology that was later replaced by a modular and more fine-grained version. Developing the first prototype led to valuable insights used in the second prototype described in section 4.2.

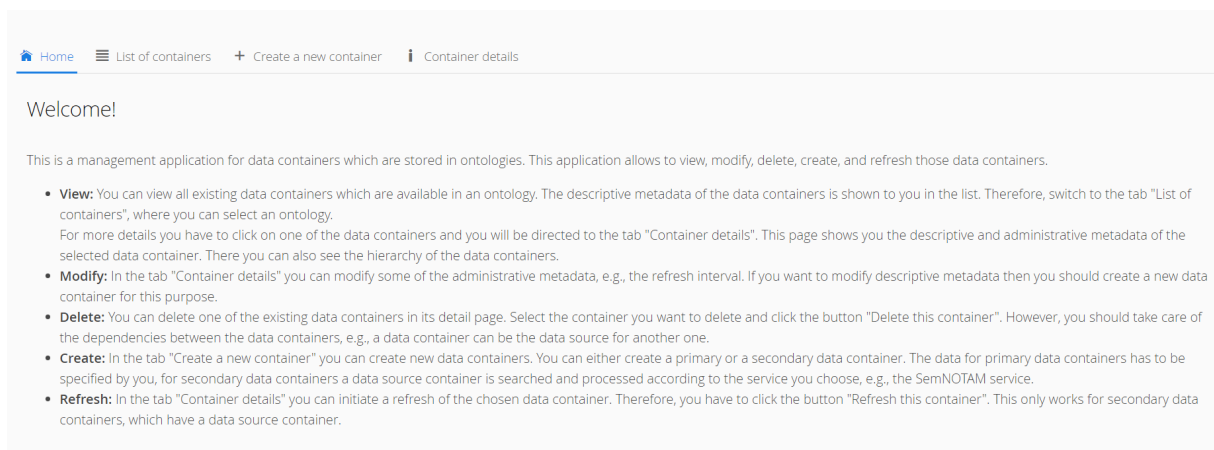
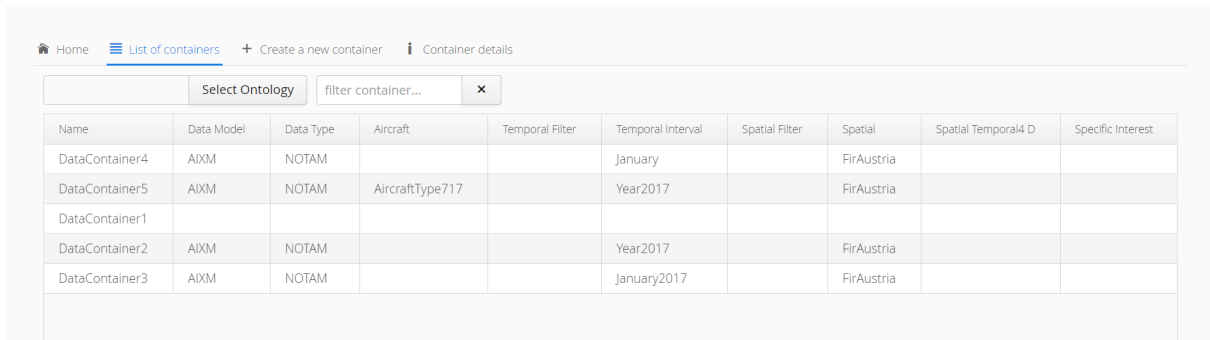


Figure 6: Start page of the Semantic Container Management (1<sup>st</sup> Evolution).

As shown in Figure 6, a web application was developed to support the user in the course of creating and maintaining Semantic Containers. Therefore, it supports to view, refresh, delete or modify existing containers, and to create new primary or secondary containers. A primary Semantic Container can be seen as a source of origin, such as an information authority in the aeronautical domain. Therefore, a primary container does not have a data source representing another Semantic Container; instead they require a data set. In contrast to this, a secondary Semantic Container depends on other either primary or secondary Semantic Container as a data source. These differences also affect the creation and the maintenance of them as shown in the following figures.

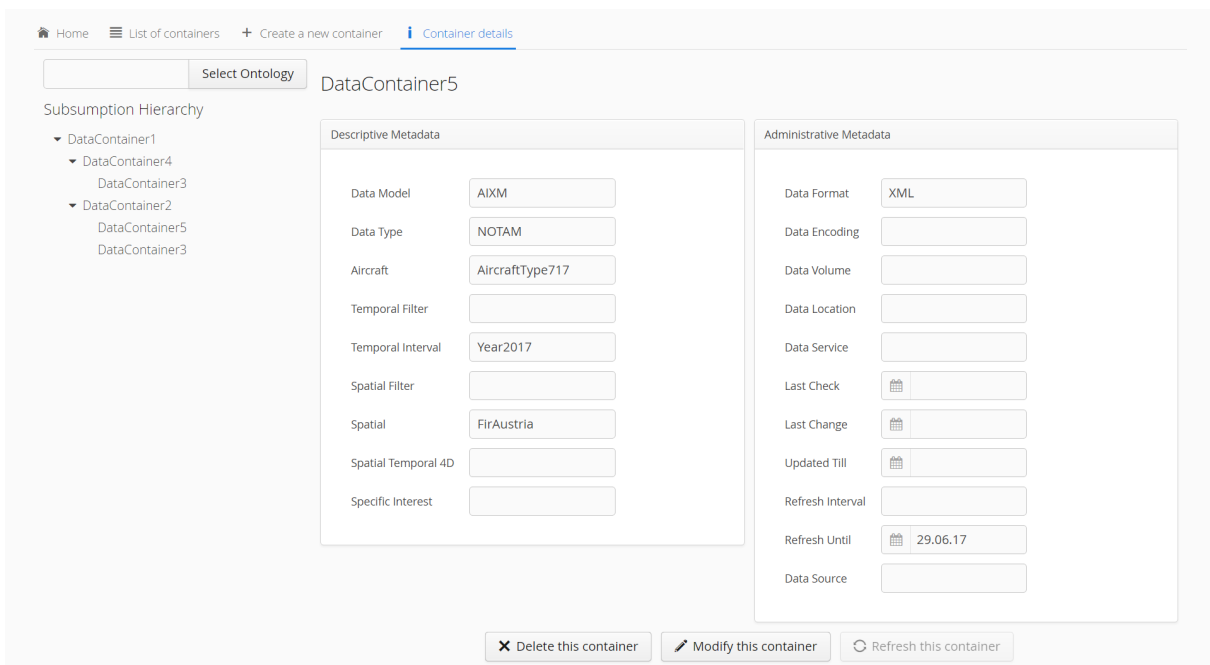


Name	Data Model	Data Type	Aircraft	Temporal Filter	Temporal Interval	Spatial Filter	Spatial	Spatial Temporal 4 D	Specific Interest
DataContainer4	AIXM	NOTAM			January		FirAustria		
DataContainer5	AIXM	NOTAM	AircraftType717		Year2017		FirAustria		
DataContainer1									
DataContainer2	AIXM	NOTAM			Year2017		FirAustria		
DataContainer3	AIXM	NOTAM			January2017		FirAustria		

Figure 7: List of all available Semantic Containers

To get an overview of existing Semantic Containers and their features respectively facets, a list of all Semantic Containers is provided and depicted in Figure 7. This view allows filtering containers by name or to show existing Semantic Containers from other ontologies. This overview provides only information about the descriptive metadata without providing insights about the administrative metadata. The descriptive metadata is defined by characterizing the data included in the Semantic Container, hence selecting facets describing the data. As shown in Figure 7, DataContainer4 includes data of data type NOTAM which are based on the AIXM data model and covers all DNOTAMs available in January for the FIR Austria. To display also the administrative metadata the view must be switched.

### 4.1.1 Setting up a DNOTAM Container



Home | List of containers | Create a new container | **Container details**

Select Ontology: DataContainer5

Subsumption Hierarchy

- ▼ DataContainer1
  - ▼ DataContainer4
    - DataContainer3
  - ▼ DataContainer2
    - DataContainer5
    - DataContainer3

**Descriptive Metadata**

Data Model:

Data Type:

Aircraft:

Temporal Filter:

Temporal Interval:

Spatial Filter:

Spatial:

Spatial Temporal 4D:

Specific Interest:

**Administrative Metadata**

Data Format:

Data Encoding:

Data Volume:

Data Location:

Data Service:

Last Check:

Last Change:

Updated Till:

Refresh Interval:

Refresh Until:

Data Source:

Figure 8: Detailed view of a Semantic Container

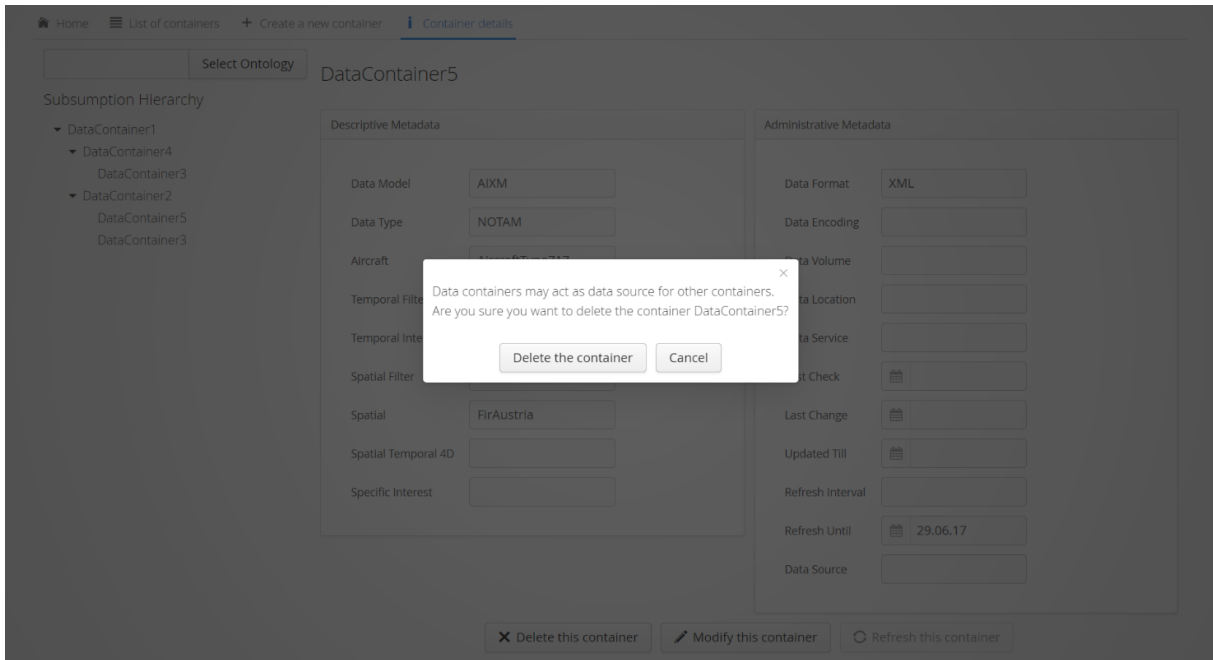
By double clicking the desired Semantic Container the view to its detailed description is opened. As depicted in Figure 8, this detailed view displays the descriptive metadata as well as the administrative

metadata. The administrative metadata is not used for subsumption reasoning but for administrative and maintaining purposes. Besides the data source and the data set characteristics the data maintenance metadata is stored. The displayed Semantic Container DataContainer5 represents a primary Semantic Container which is why the refresh option is not enabled.

Furthermore, a subsumption hierarchy is provided which indicates which Semantic Containers are subsumed by other existing Semantic Containers based on its data characteristics respectively its descriptive metadata. As depicted in Figure 9, DataContainer1 subsumes all other Semantic Containers which indicates that this Semantic Container represents a primary Semantic Container. However, other primary container can be subsumed as well as shown by the primary Semantic Container DataContainer5 which is not characterized by a data source.

**Figure 9: Modifying the properties of a Semantic Container**

Besides simply displaying the Semantic Container's details it is also possible to modify a Semantic Container. These modifications, however, are limited to the administrative metadata respectively to the *Refresh Interval* and the *Refresh Until* fields. Other administrative metadata are not editable since they represent properties of the data set itself or properties which are set in course of the refresh process.



**Figure 10: Delete a Semantic Container**

If it is required to change other properties as well then, the Semantic Container must be deleted, as shown in Figure 10, and/or a new Semantic Container created as depicted in Figure 11. To create a Semantic Container a user must provide the facets used to characterize the data content within a Semantic Container. Therefore, a user can specify a unique Semantic Container name along with a data model, a data type, or an aircraft concept. Further spatial and temporal concepts specifying different filter granularities can be specified. A specific interest can be selected to detail and restrict the data covered by the newly created Semantic Container. As described in D2.1 those concepts are then mapped to interests and integrated into an interest specification as well as a Semantic Container description stored in an ontology.

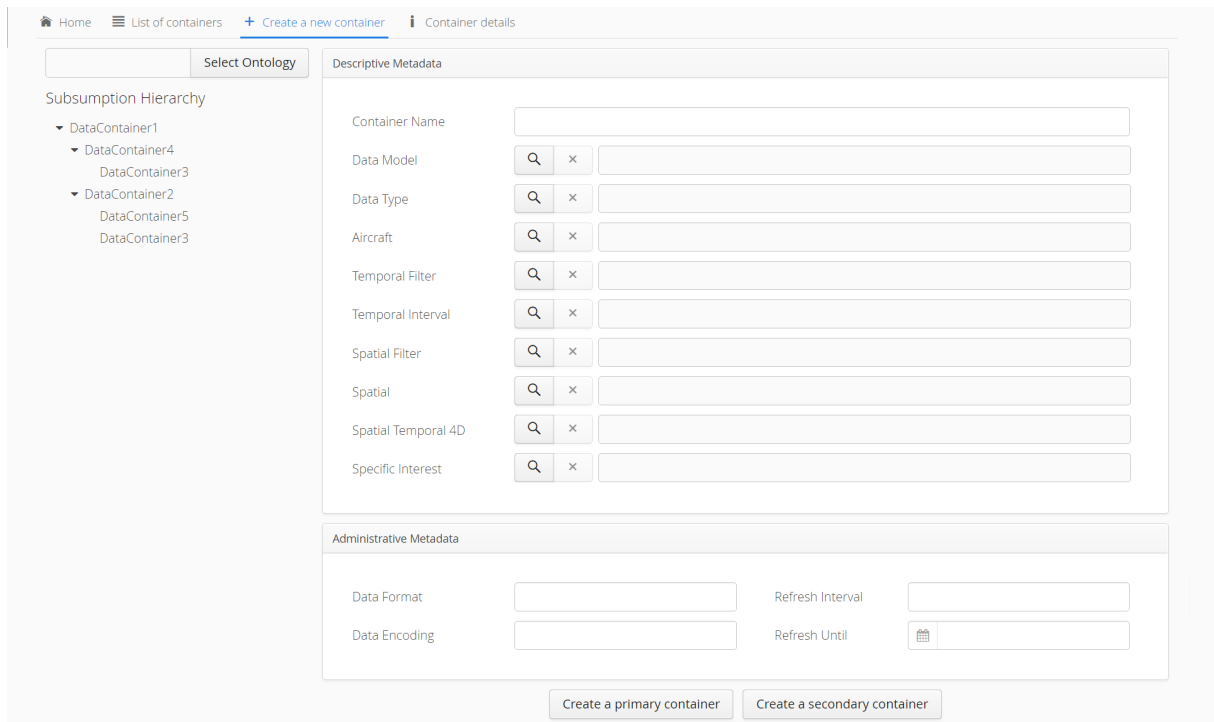


Figure 11: Create either a new primary or secondary container

Figure 12 depicts the selection of concepts for the facets. It is possible to select concepts for a specific facet, e.g., such as the selection of an aircraft concept for the aircraft facet shown in Figure 12. These concepts can be stored in different ontologies and further be mapped to a common concept representation.

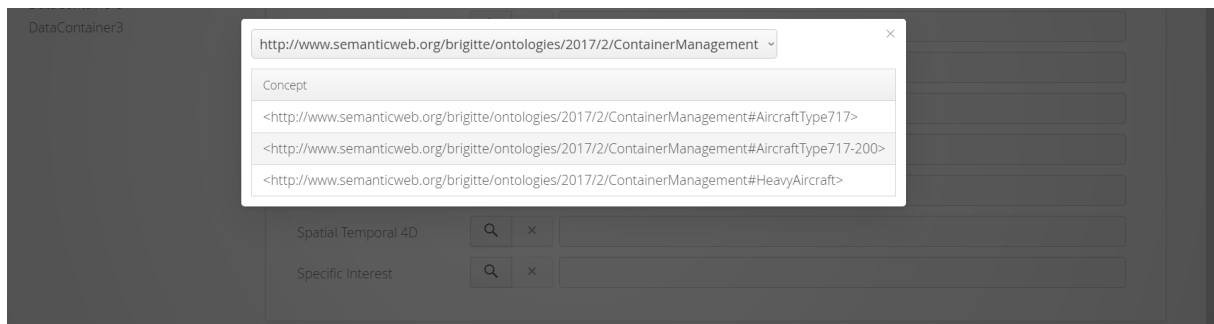


Figure 12: Select concepts for facets

The creation of a Semantic Container is split into two use cases. Either a primary Semantic Container or a secondary Semantic Container is created. The creation of a primary container requires, besides the selection of concepts for the facets, the provision of an input data set. The Semantic Container management facilitates this by providing an upload mechanism as shown in Figure 13. Therefore, primary Semantic Containers are usually only updated manually.

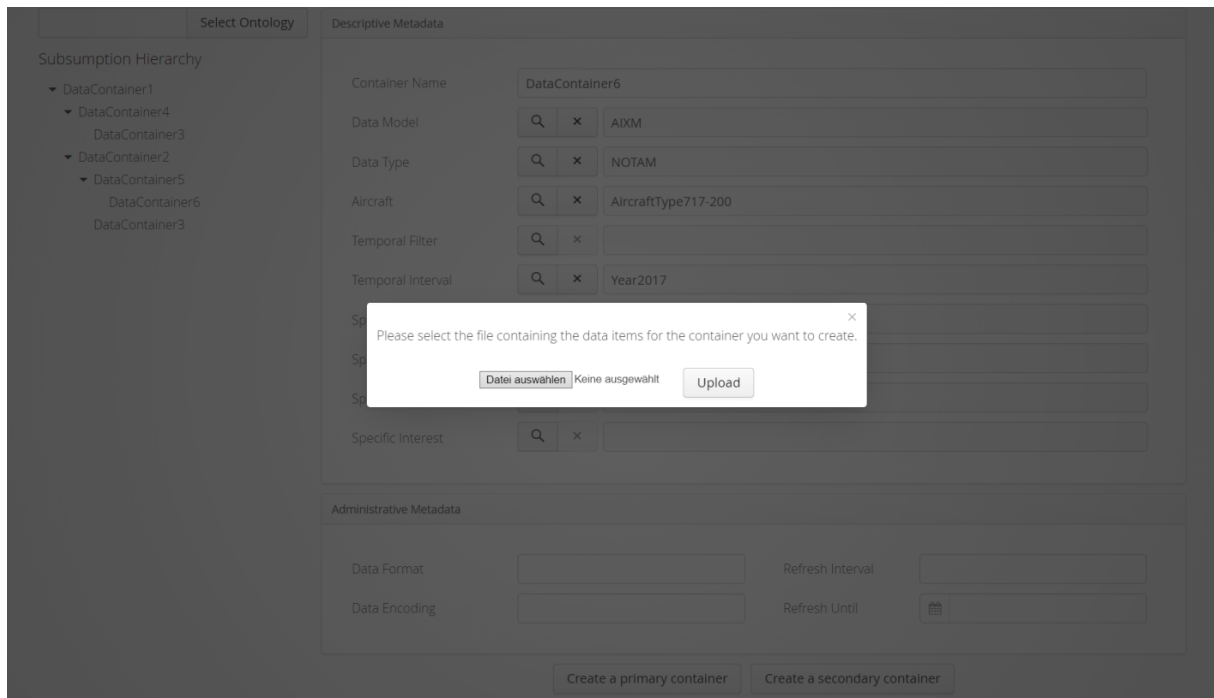


Figure 13: Select data set for primary container

In contrast to primary Semantic Containers, secondary Semantic Containers are created by selecting other suitable Semantic Containers as data sources. Irrespective of the type, both the primary and secondary Semantic Container can be used here. The Semantic Container representing the data source, however, must fulfil certain aspects. They must represent at least a super set of the data set defined in the newly created secondary Semantic Container. To determine this subsumption hierarchy the web application employs the Hermit reasoner to interpret the OWL description of the newly created secondary container. This OWL description is characterized by the concepts selected for the facets which are thereby used for performing subsumption reasoning. A detailed description of this process is available in D2.1. The subsumption reasoning results in a list of Semantic Containers which include a data set representing the most specific super set described by the newly created Semantic Container.

Figure 14 depicts the most specific super sets for the secondary data DataContainer7 with the facets AIXM for the data model, NOTAM for the data type, 717-200 for the aircraft, and January2017 for the temporal interval. The Semantic Containers DataContainer6 and DataContainer3 are suggested because both represent a super set. DataContainer6 is a super set since all facets correspond to the facets of the newly created secondary container except the temporal interval. DataContainer3 is a super set since all facets correspond to the facets of the newly created secondary container except the aircraft facets. Since both Semantic Containers only differ regarding one facet, both of them are suggested as a most specific super set.

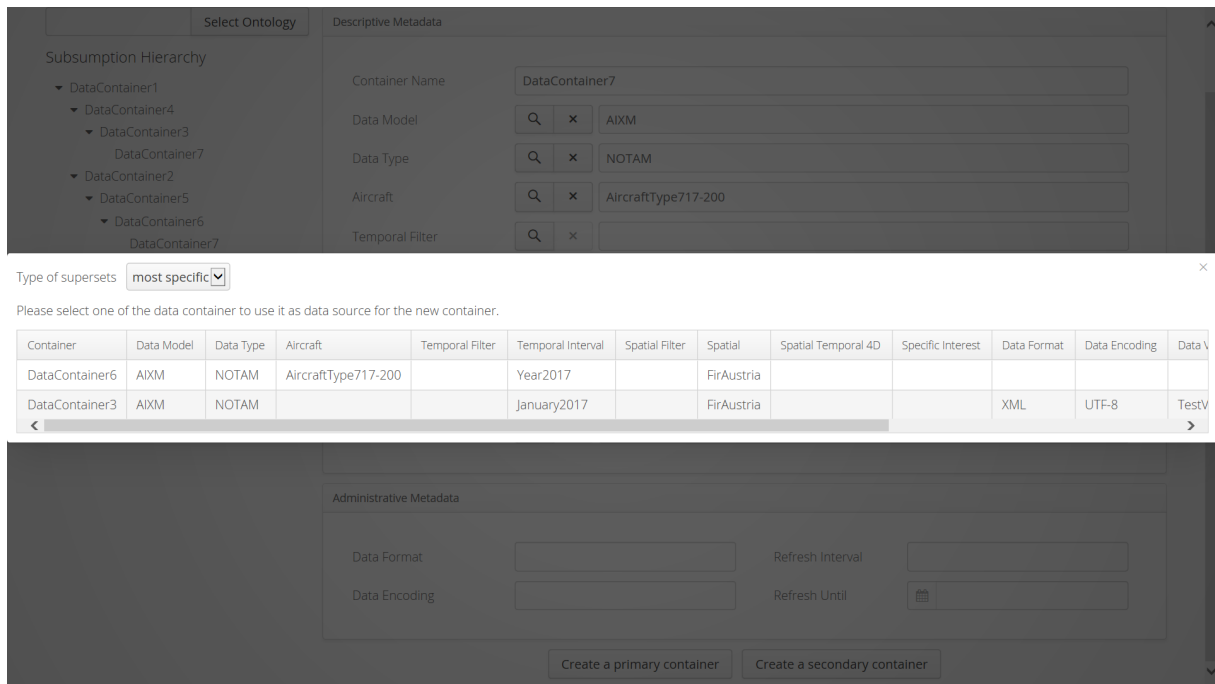


Figure 14: Select the most specific Semantic Container as the input data set

The selection of the most specific super set is extended because the drawback of a potentially bigger dataset can be counterbalanced if it provides, e.g., the data set in the required data format such as depicted on Figure 15. Following this approach administrative metadata which is usually not comprised during the subsumption reasoning can be incorporated during the selection of a suitable data source. The data source selected for the newly created secondary Semantic Container is further used as the input for a service, filtering, transforming, or mapping the data to the desired description.

As described in D2.1 the creation of a new Semantic Container leads to the definition of an OWL description which is used for subsumption reasoning as well as an interest specification representing this OWL description respectively the selected concepts as interests. The interest specification contains a detailed description of the annotations and groupings to be made and spatial, temporal, and semantic interest descriptions to be considered during evaluation. Thereby the dataset of the selected (most specific) super set Semantic Container is used as the input for further services. The current prototype implementation web service architecture to create Semantic Containers respectively provides their datasets. Figure 15 shows the selection of the evaluation web service. The data set of the data source and the generated interest specification are used as parameter of the web service. The returned relevant DNOTAMs are used as the data set for the newly created secondary Semantic Container.



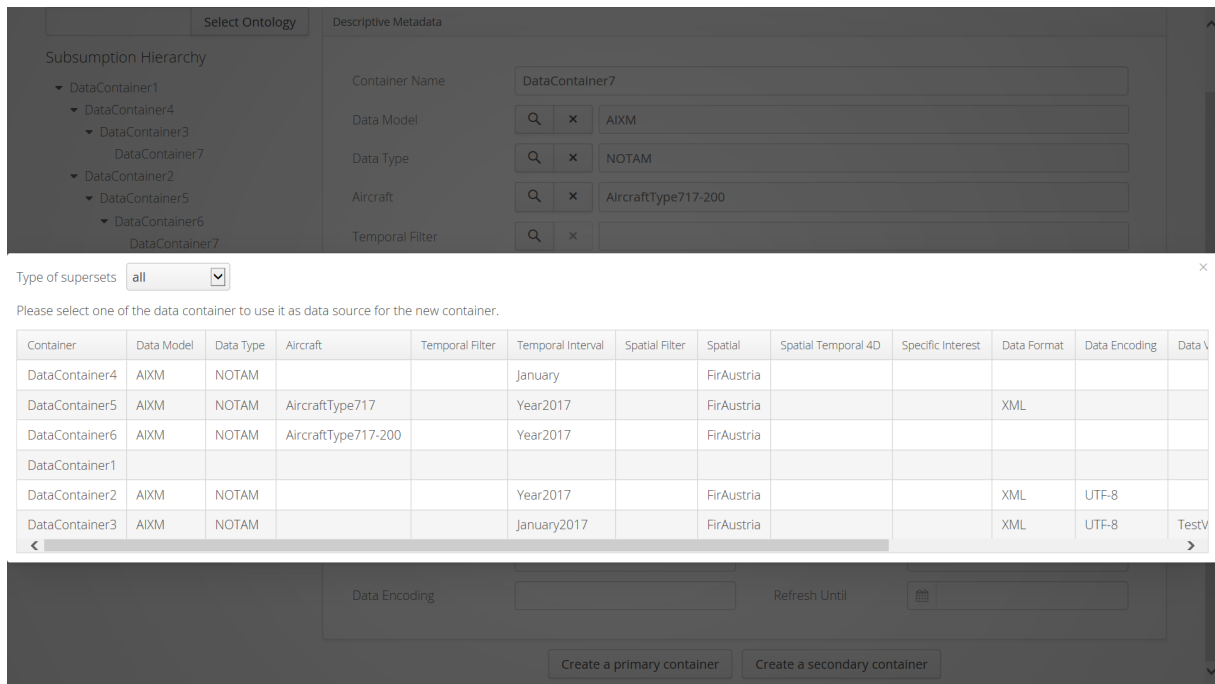


Figure 15: Select a suitable Semantic Container as the input data set

The original Integrated Digital Briefing prototype was already capable of helping to interactively prepare and produce a useful briefing and multiple ways to filter the information to be included in the final product. With adapting to the BEST Semantic Container concept, a lot of calculation time (on the queries) that had to be done in the past is now covered by the containers. Pre-selected information retrieve through the containers leads to faster usability on the end user side.

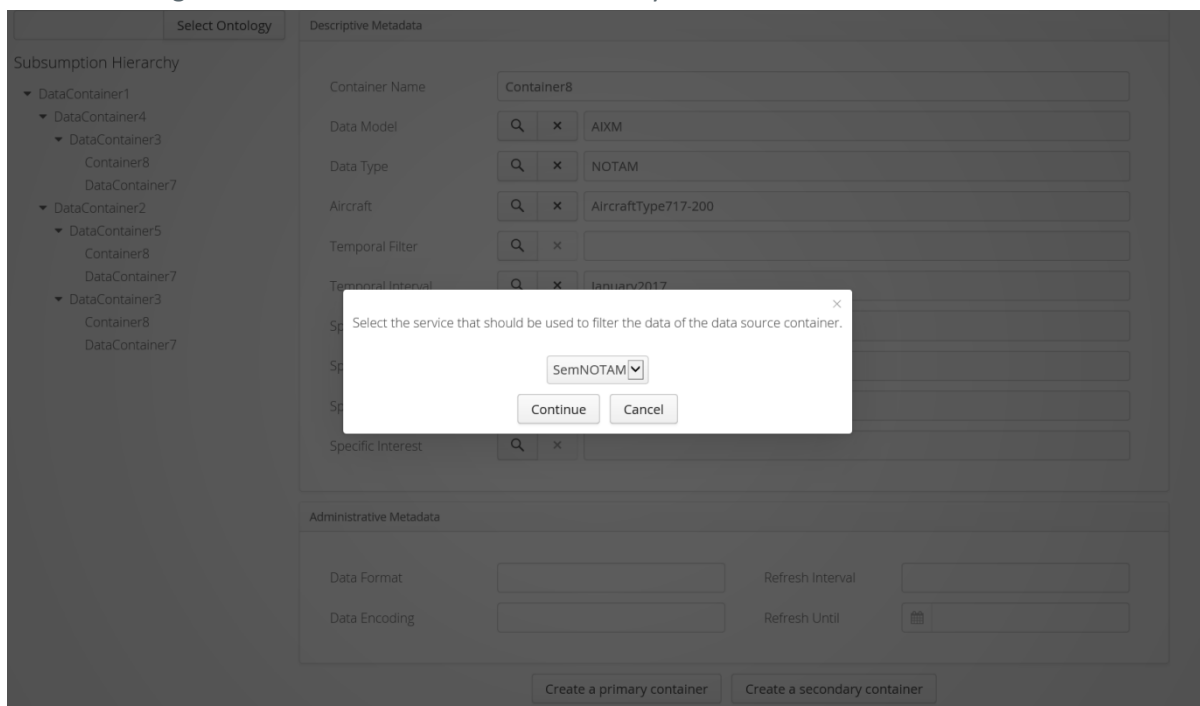


Figure 16: Service selection for the secondary container

### 4.1.2 Conclusion

Given the fact that several briefing systems from different vendors will query the same routes, the same DNOTAMs, etc. repeatedly the impact of BEST Semantic Container concept will rise. The processing time will be improved and the number of service request will drop as well. This will not only relieve the SWIM network but also boost the performance of all SWIM applications that make use of the BEST Semantic Container concept.

The addition of BEST analysis and the semantic preselection and sorting is closing that last gap, by applying the ontologies and semantic reasoning.

This first experimental prototype developed within BEST shows the benefits of the Semantic Container concept. It was later superseded by the second prototype covering not only DNOTAMs but a wider range of messages and using the modular BEST ontology.

## 4.2 Experimental Prototype Evolution 2: Semantic Container Management Platform

In this section the second experimental prototype is described. The Semantic Container Management System (SCMS) is designed to run on multiple locations which are connected through the internet. Each of those locations of the Semantic Container management has its own physical Fuseki server where the physical model is stored and a BaseX server, where the data of the specific location is stored (for example the NOTAMs, TAFs, etc.).

The physical model stores the information about the physical container and its corresponding sets. Those sets are stored in the BaseX server. Therefore, the physical model can be used to identify which set is stored in which container. Furthermore, the current version as well as the version which existed on a specific location is documented in the physical model and are therefore not available at any other location. A container exists in the physical model of a certain location only if it was allocated at this location.

The logical Fuseki server stores the data which is shared for all instances of the container management system. The Fuseki servers store the local model. The local model contains the registry of the services providers and services, the facets, the ontology, the administrative metadata, the semantic label, the assignment between a container and an administrative data as well as the logical container model. This means that all containers are registered in the shared instance. In practice, there can be multiple logical Fuseki servers which are synchronized. However, in the prototype there is only one Fuseki server.

For convenience, the container management system provides an application which starts multiple locations on one single machine. It can be configured by using the `config.properties` file. Therefore, it starts one logical Fuseki server and as many locations as it finds in the `locations` array. First, it starts the logical server. The properties which configure the host and the port of the logical Fuseki server are the properties `logicalHost` and `logicalPort`.

As mentioned before the `location` array defines the amount of locations which are started by running the app. Each location consists of a physical server and an BaseX server. Therefore, the host and port of the BaseX server and of the physical Fuseki server must be configured. For the BaseX server an arbitrary database name, username and password has to be defined. Finally, the `location` array contains the internal host and port where the backend is hosted (`privateRestHost`, `privateRestPort`) and the external host and port where it can be accessed from outside (`restHost`, `restPort`).

### Distribution and Replication

Distribution and replication consists of two different types: a logical and a physical allocation. The reason why there is a logical and a physical allocation is that the information which exists about a certain container is separated from the actual data which is stored in a container. The logical allocation holds the information which exists about a container and the physical allocation holds the actual data and also some information about it. Both, the logical and some physical information are available at every instance.

The logical allocation holds semantic information about a certain container, the provenance and the structure of a container. For example, the logical allocation stores the name of a container, the site

where it is allocated, the administrative and descriptive metadata and the different versions which exists. The database management system synchronizes the logical allocation automatically with each other. This is due to the fact that the data can only be added, except for the meta-information which can be updated as well. Therefore, the information must be only transferred to another location and the database management system can update its content at the target location.

Each instance of the database management system has also a physical allocation. The physical allocation stores the actual datasets. Not all datasets of all containers are stored at specific location but only the datasets of the containers which were allocated at the specific location. To achieve this, the physical allocation must be aware of containers which are allocated at its location and which datasets are contained by a specific container. This information is stored in a database which handles the physical information's for a site. For example, it stores if a specific set is degenerated or regular or additional information about a specific set. The actual content is stored in an XML database.

The advantage of this approach is that every instance of the database management system is aware about every available container. This is possible, due the fact that the logical model is stored in a single graph in the logical allocation which is running on ever instance. Therefore, a search for a container can be done with a certain information need. Another advantage of this approach is that it optimizes the storage usage. Not every container which exists logically must be allocated physically at every location. Each physical allocation can consume a lot of storage because there could be a big amount of data stored in one physical allocation.

Each container is then distributed as its own service. Figure 17 gives a high-level overview how the containers collect the information from the various services. Composite containers are implemented as inherited precursor (generalizations).

This architecture slightly differs from the one to-be proposed in D2.2. The main difference is that in the simple architecture proposed in D2.2 also the description of physical containers is fully replicated (there is one fully replicated RDF graph which describes not only logical containers but also physical containers). The motivation for the D2.2-architecture was to make easier container discovery with information needs including freshness and quality requirements (which need the description of physical metadata).

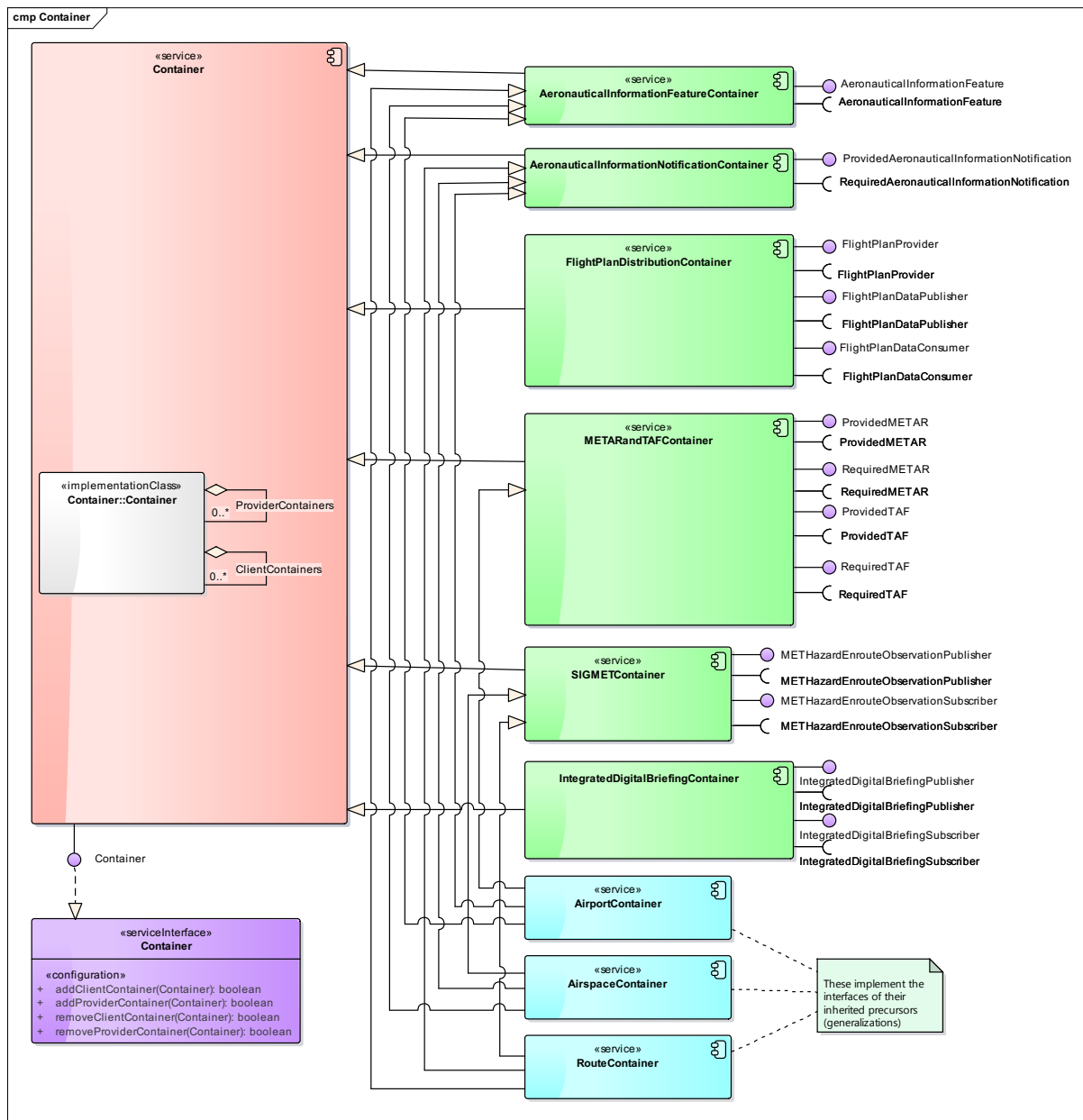


Figure 17: Semantic Container Management Platform

### Consistency Management

To ensure consistency of the containers they are synchronized. There is a primary allocation which contains the desired state of a container. Each secondary allocation synchronizes itself with the primary allocation by pulling the content from the container (PULL-Updates) or by registering itself at the primary allocation for PUSH-Updates. Whenever a container is registered for PUSH-Updates and a new set is added the primary allocation distributes this set.

If a set is added to a versioned container a new version is instantiated. If the set was added to a primary allocation it is called a regular version. This means the version represents a desired state. However, if a set is added to a secondary container the container differs from the desired stated. Therefore, this

version is called a degenerated version. Whenever a new regular version (regular set) arrives at a secondary allocation - via PULL or PUSH updates - then the desired state is inherited and the degenerated version is dropped. However, the consistency management registers every version which ever existed on a certain location. Therefore, degenerated version are kept on a certain allocation but are not deployed on to other allocations.

### **Provenance and Composition**

In the container management system, one can create a certain structure of containers. The basic structure is an elementary container which stores the actual data of the container management system. There can be an annotated elementary container and an entity elementary container. A container which contains other container information is grouped together to composites. There can be composite container which only contains containers which store the same type of data. Those containers are called homogenous composites. If a composite holds containers of different types it is called a heterogeneous composite.

The provenance of the data is ensured by referencing which service produced the data and which service inserted the data. Therefore, for each container derivation a service can be recognized.

## **4.2.1 Frontend**

This section describes how to create a container, allocate it and add datasets to it. Furthermore, it is explained how to add an ontology, add facets of the ontology to the container management system and how to register containers by using the web frontend.

### **4.2.1.1 Distribution and Replication**

#### **Register an Ontology**

In the frontend of the Semantic Container Management System an ontology can be added under the menu entry Metadata. An ontology can contain multiple facets with the corresponding concepts. The figure below shows an ontology with two facets the AircraftFacet and the EventFacet with its corresponding concepts.

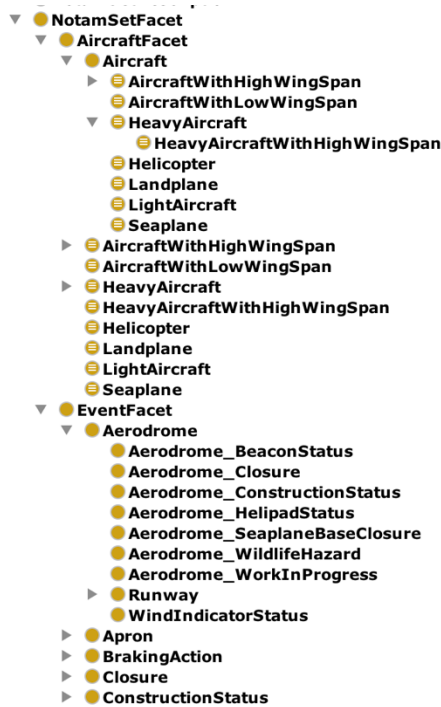


Figure 18: Ontology with Facets and Concepts

This ontology can be added by using the function “Add Ontology” - see in Figure 19. First the ontology name must be chosen, next an ontology file must be selected, then the representation must be selected and finally the ontology can be added to the system. The ontology will then be available for all locations because it is stored in the logical model and not in the physical model of the selected location.

Figure 19: Add an Ontology

### Register a Facet

A facet is a superclass of an ontology which has different concepts (subclasses) – see Figure 18 for an example of a facet. Example: first select the ontology, then select the aircraft facet, and finally add the facet to the logical model of the container management system by pressing the button “Add selected Facet”. The facets are stored in the logical model of the container management system and are available for all locations once they are added and are displayed under “Available Facets” on the user interface.

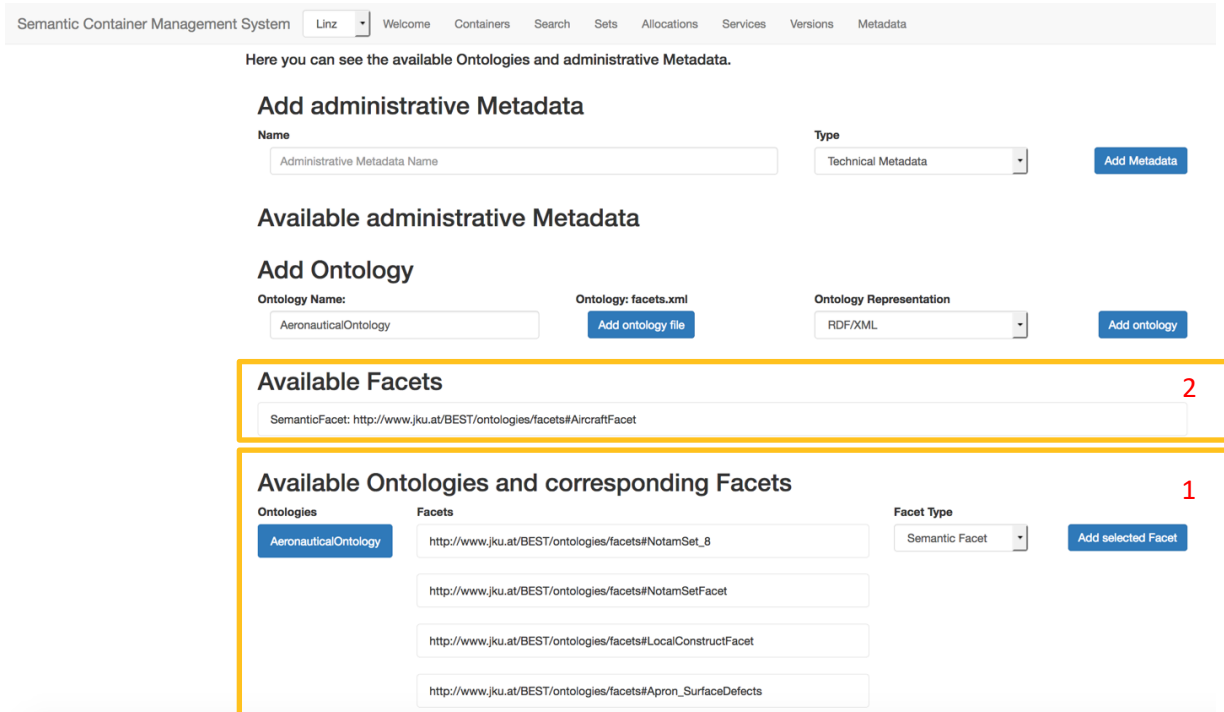


Figure 20: Add Facet to a Semantic Container Management System

### Register Administrative Metadata

Administrative metadata represents specific information like the time when the last update was performed. To add administrative metadata only a name must be specified and a type must be selected. Finally, the metadata can be added by pressing the “Add Metadata” button. Once added the administrative metadata appears under the administrative metadata label. The administrative metadata is also part of the logical model.

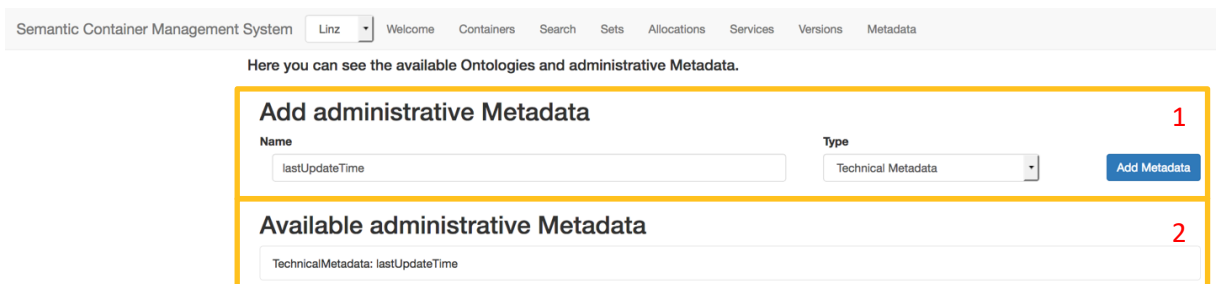


Figure 21: Administrative Metadata



**Use Facets and Administrative Metadata when creating a container**

A container is created under the menu entry Containers -> “Create New Container” and there facets and administrative Metadata can also be specified. To specify a facet, add descriptive metadata to a container during its creation process, specify the relevant, and finally choose a concept from the list.

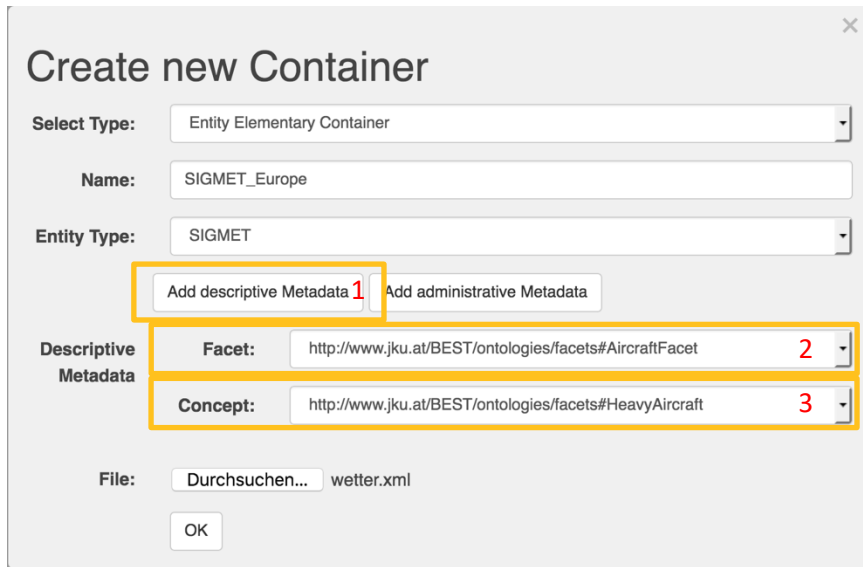


Figure 22: Specify Facet

To add administrative metadata to a container press “Add administrative Metadata” (see 1), then choose from the list of predefined administrative metadata, and finally set its characteristics by applying a value under Literal. (Literal is a mandatory value; if the literal exists, the container management system references an existing literal otherwise a new literal is created.)

Figure 23: Add Administrative Data to a Container

The administrative metadata is stored in the logical model and is therefore available for all locations.

### Finding a container through facets

To find a container use the functionality under search -> “New Search Request” and specify the needed information. This allows the containers to be filtered by a specific entity type, an annotation type and a specific facet. The search algorithm works as follows:

- If the Entity Type is not “NONE” only the containers are returned with the given entity type.
- If an Annotation Type is given, only annotated containers are returned which have the specific entity type.
- If the search specifies a concept for a facet, then a container may only have a more general concept as value for this facet (or no value for this facet). For example, if the needed information is specified by one facet (EventFacet: Runway) and a container contains all information relevant to both heavy aircrafts and aerodrome events (AircraftFacet: HeavyAircraft, EventFacet: Aerodrome), the container is not returned because it does only represent a subset of the data which is searched for. However, if there is a search for (EventFacet: Runway, AircraftFacet: HeavyAircraft) the container would be returned. The container fulfills the information need because HeavyAircraft is equal to HeavyAircraft and the concept Aerodrome is a superclass of the concept Runway.

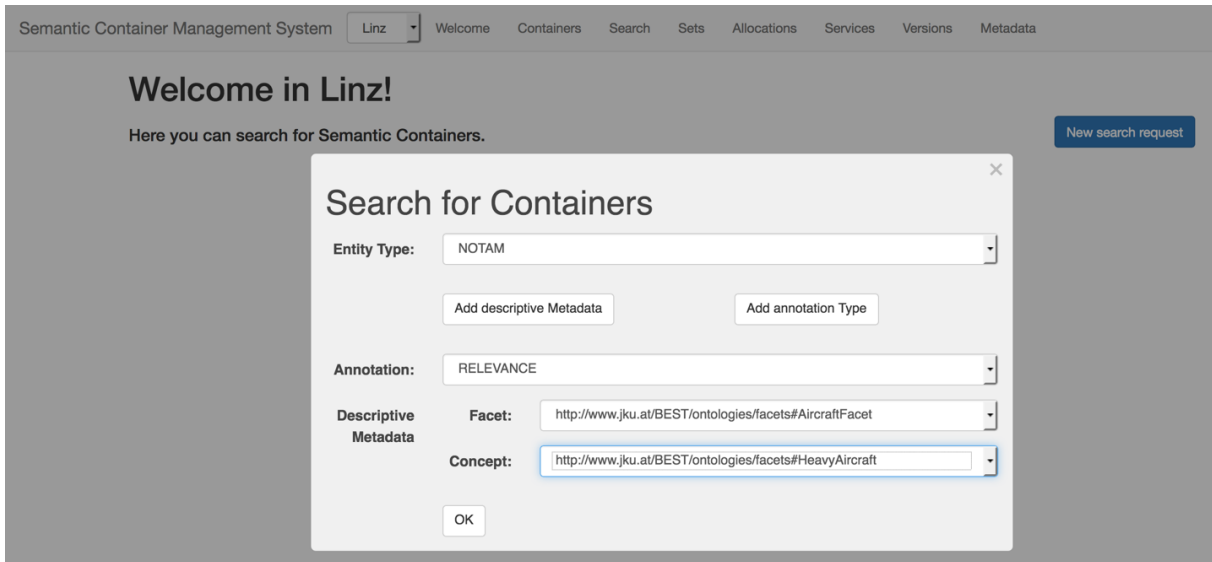


Figure 24: Example Search

### Allocate a Container

A container is allocated at a location when the physical data which is hold by a container is available at this location. The difference between the physical and the logical model is that the logical model is equal for all locations whereas the physical model can differ, depending on the allocated containers. If a container is allocated at a specific location the data is copied to that location in an initial process and afterwards the data is kept synchronized with either a PULL or PUSH mechanism.

A container can be allocated in the menu item Allocations. First choose the location where to allocate the container. Next choose the container which should be allocated. Then select whether PULL or PUSH mechanism should be used. Finally, the container can be added.

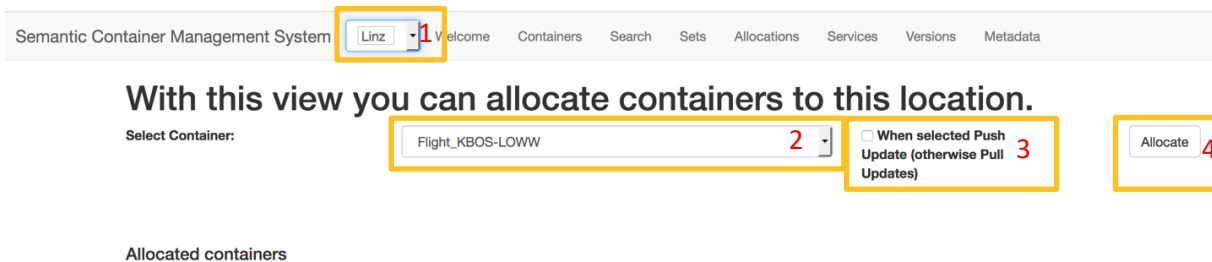


Figure 25: Allocate Container

A container is only available for allocation if it is not allocated at a specific location. Furthermore, it must be the top container of a composite container or an entity container which is not part of a composite. This is due to the constraint that composites are only allowed to be allocated as an entity.

Semantic Container Management System Wien Welcome Containers Search Sets Allocations Services Versions Metadata

**With this view you can allocate containers to this location.**

Select Container:   When selected Push Update (otherwise Pull Updates)

**Allocated containers**

FIR\_Container

METAR\_Container

SIGMET\_Container

AIRMET\_US

AIRMET\_Europe

Airports\_Container

Figure 26: Allocation view with Allocated Containers

#### 4.2.1.2 Consistency Management

The consistency management is about data synchronization between different allocations. When a container is created at a specific location then an entry in the logical model and an entry in the physical model are created. The logical information can be accessed from all locations. The information in the physical model is only specific to a certain location. The first location where a container is created is the primary allocation. This means that this is the root location which contains the desired state. All secondary allocated containers are synchronized to this state. A secondary allocation is an allocation where the content of the primary allocation is replicated to.

The data of a container is represented as a set. A set contains one or more entities of a specified entity type. For example, a NOTAM container can have a set which contains multiple NOTAMs. A set can also contain metadata about its content. Only elementary containers can have sets. To identify the sets of a composite container, one has to build the union of all sets contained by the elementary containers of the composite. The sets of a secondary allocation can differ from the sets the container holds in the primary allocation. If for example, the primary location is not available anymore the secondary allocation does not have a desired state anymore. Nevertheless, it should be kept up-to-date. Therefore, it can have sets which were not issued by primary source (so called degenerated sets.) These degenerated sets are available as long as the container is not synchronized with the primary allocation.

Whenever a set is added a new version is generated. If a set is added to a primary allocation then the new version is called a regular version and it is recorded in the logical model. If the set is added to a secondary allocation it is a degenerated set. Therefore, also the resulting version is degenerated. A degenerated version only exists at a certain location.

#### Add set to a container at the secondary allocation

To add a set to a container choose this container in the tab Sets. Only elementary containers are allowed. Next, specify the set name and its content. Finally, add the container. This results in a degenerated physical set.

Semantic Container Management System Linz | Welcome Containers Search Sets Allocations Services Versions Metadata

## Welcome in Linz!

Here you can see the available Versions of the Containers.

**Container Name:** 1

**Set Name:** 2

**Set Content: wetter.xml**

**Set Content:** 3

**Add set to container:** 4

Flight\_KBOS-LOWW

NOTAM\_Container

NOTAM\_Europe

NOTAM\_US

MET\_Container

SIGMET\_Container

SIGMET\_Europe

**Sets**

RegularPhysicalSet	6803d95d-bffc-4eae-b527-18eea600a854	<input type="button" value="View Set"/>
DegeneratedPhysicalSet	newNotams	<input type="button" value="View Set"/>

Figure 27: Add set to a Secondary Allocation

### Add set to a container at the primary allocation

The same procedure can be performed at the primary location. This results in a RegularPhysicalSet. When the containers are synchronized with each other than the DegeneratedPhysicalSet at the secondary allocation will be hidden.

### Inspect historical versions

In order to inspect which version existed for a container at a specific location use the Versions view. By clicking on a container the version which existed of a specific container will be shown. Those versions are separated into physical versions and logical versions. The logical versions are loaded from the logical model whereas the physical versions are forming the physical model of the location accessed by the frontend. The sets can only be viewed from the physical model because the sets are not included in the logical model. In Figure 28 a container with two versions is depicted. One version with one set and another one with two sets. The versions never differ from the versions which are registered in the logical model.

Semantic Container Management System Wien Welcome Containers Search Sets Allocations Services Versions Metadata

## Welcome in Wien!

Here you can see the available Versions of the Containers.

Flight\_KBOS-LOWW

NOTAM\_Container

NOTAM\_Europe

NOTAM\_US

MET\_Container

SIGMET\_Container

**SIGMET\_Europe**

SIGMET\_US

**Logical versions**

SIGMET_Europe-0
SIGMET_Europe-1

**Physical allocated versions**

Version:	SIGMET_Europe-1
Set:	Name: newSigments Creation Time: 24.04.2018 AD 00:56:52.761 GMT
Set:	Name: 6803d95d-bffc-4eae-b527-18eea600a854 Creation Time: 23.04.2018 AD 14:42:07.014 GMT
Version:	SIGMET_Europe-0
Set:	Name: 6803d95d-bffc-4eae-b527-18eea600a854 Creation Time: 23.04.2018 AD 14:42:07.014 GMT

Figure 28: Inspect Versions

### 4.2.1.3 Provenance and Composition

One goal of the container management system is to create container hierarchies, i.e., combine containers and also trace provenance. It must be traceable which server derives a container and which set was added by which service.

#### Create composition

There are two types of compositions and two types of elementary containers. There is a homogenous composite which only holds containers of the same entity type; and there is the heterogeneous composite container. A heterogeneous composite container can hold containers of different types. The elementary containers contain the data and are the leafs of the composition. Each composite must have a component.

Add a container by pressing at containers -> "Create new container"; the process can be repeated. First select the container type and add one or more sub containers (if it is a composite container).

Figure 29: Creation of a Homogenous Container

### Register a service

In the container management system an elementary service can be registered: In the Service menu enter a service name and a service provider and press “Add service”. Already existing services are there displayed as well.

Figure 30: Add a Service

## 4.2.2 Interface

This part of the documentation describes the REST interface which can be accessed from outside. This is also the REST interface which is used by the fronted. The REST interface offers the following paths:

- config (access about the location configuration)
- container (the container which are available on each location)
- metadata (the metadata which is provided about the container at a specific location)
- service (the services which are registered in the Semantic Container Management System)

The paths are grouped differently. Therefore, the path container groups all operation which can be done on a container. However, there can be operations of the Distribution and Replication (container/allocation), of the Provenance and Composition structure and of the Consistency Management. The metadata perspective only contains operation for the Distribution and Replication perspective; add ontologies and facets by using the interface this path provides. Finally, in the service perspective service providers can be registered which can be used to replicate containers.

### 4.2.2.1 Configuration

#### Returns the current location of the Container Management System

GET /config/location

Description: This method returns the location where the actual Container Management System is allocated.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Location</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

#### Returns all locations

GET /config/locations

Description: This method returns all locations where a Container Management System is allocated.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Location</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json



### 4.2.2.2 Container

#### Returns all available Logical Versioned Containers

GET /container

Description: This method returns all Logical Versioned Containers which can be found on the global Fuseki server. It groups them so that containers which are hierarchically related are grouped together.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

#### Returns all allocated containers

GET /container/allocate

Description: Returns all containers which are allocated at this location.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

#### Adds/allocates a physical container

POST /container/allocate/{containerName}

Description: This method adds/allocates an physical container to this location.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container which should be allocated.	string
<b>Body</b>	<b>pushUpdatesTo</b> <i>required</i>	Specifies weather the primary source pushes updates to it.	boolean

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalVersionedContainer</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container name was not found.	No Content

<b>409</b>	The physical container already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

### Returns a specific allocated container

GET /container/allocate/{containerName}

Description: Returns a specific allocated physical container. The physical container contains all sets which are already synchronized to this location.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container which should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalVersionedContainer</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

### Returns all sets of a physical versioned container

GET /container/allocate/{containerName}/physicalset

Description: Returns the basic and delta sets. The basic set is always on index 0.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container where the sets should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalSet</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Adds a physical set to an allocated container**

POST /container/allocate/{containerName}/physicalset/{physicalSetName}

Description: This method adds a physical set to a container.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the physical container where the sets should be added.	string
<b>Path</b>	<b>physicalSetName</b> <i>required</i>	The name of the sets which should be added.	string
<b>Body</b>	<b>physicalSet</b> <i>required</i>	The physical set.	<a href="#">PhysicalSet</a>

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalSet</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container was not found.	No Content
<b>409</b>	The sets already exist.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

**Returns a specific set of a physical versioned container.**

GET /container/allocate/{containerName}/physicalset/{physicalSetName}

Description: Returns a specific set of physical versioned container. The set name is specified in the path.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the physical container where the sets should be added.	string
<b>Path</b>	<b>physicalSetName</b> <i>required</i>	The name of the sets which should be added.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalSet</a>

<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container or the set was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Adds a version to an allocated container**

POST /container/allocate/{containerName}/version/{versionName}

Description: This method adds a physical version to a container. This method should be used when a logical version was already created and the physical version was not deployed at this location.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the physical container where the sets should be added.	string
<b>Path</b>	<b>versionName</b> <i>required</i>	The name of the sets which should be added.	string
<b>Body</b>	<b>physicalVersion</b> <i>required</i>	The physical version.	<a href="#">PhysicalVersion</a>

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalVersion</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container and the version name was not found.	No Content
<b>409</b>	The physical version already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

**Returns all containers which are allowed to be allocated.**

GET /container/allowedtoallocate

Description: This are all containers which are not part of a composite container.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Returns all not allocated containers**

GET /container/notallocate

Description: Returns all containers which are not allocated at this location.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

### Returns matching containers

GET /container/search/{searchSpecification}

Description: This method returns all specific containers matching the specified parameters.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>searchSpecification</b> <i>required</i>	Contains all the necessary search data.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

### Adds a container pair

POST /container/{containerName}

Description: Adds a container pair (Logical and corresponding physical container) to the location.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container which should be added.	string
<b>Body</b>	<b>containerPair</b> <i>required</i>	The logical and the physical container.	<a href="#">PhysicalLogicalContainerPair</a>

Responses:

HTTP Code	Description	Schema
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container name was not found.	No Content
<b>409</b>	The logical container already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

### Returns a specific container

GET /container/{containerName}

Description: This method returns the specific container specified by the parameter.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container which should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container was not found by the given name.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Returns the root container of a composite.**

GET /container/{containerName}/rootcontainer

Description: If the container is part of a composite, it will return the root container. Otherwise it will return an empty json.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container of which the root container should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalVersionedContainer</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container name was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Logical versions of logical container.**

GET /container/{containerName}/version

Description: Returns all logical versions of a specific container.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container where the versions should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalLogicalVersionsPair</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container name was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

#### Physical version of container.

GET /container/{containerName}/version/{versionName}

Description: Returns a specific physical version of a specific container.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>containerName</b> <i>required</i>	The name of the container where the version should be returned.	string
<b>Path</b>	<b>versionName</b> <i>required</i>	The name of the version.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">PhysicalVersion</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The container name or the version was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

### 4.2.2.3 Metadata

#### Returns all administrative metadata

GET /metadata/administrativ

Description: Returns all administrative metadata of the system.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Ontology</a>
<b>400</b>	There was an error in the request.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json



### Adds an administrative metadata type.

POST /metadata/administrativ/{administrativeName}

Description: This method adds an administrative metadata type to the system which then can be selected later. For each container an administrative metadata can be added.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>administrativeName</b> <i>required</i>	The name of the administrative metadata which should be added.	string
<b>Body</b>	<b>administrativeMetadata</b> <i>required</i>	The metadata which should be added.	<a href="#">AdministrativeMetadata</a>

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Ontology</a>
<b>400</b>	There was an error in the request.	No Content
<b>409</b>	The administrative metadata already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

### Returns all facets

GET /metadata/facet

Description: Returns all facets which are registered as facets. A facet belongs to a specific OWL ontology.

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Facet</a>
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

### Adds a facet

POST /metadata/facet/{facetName}

Description: Adds a facet to the container management system. A facet is a superclass within an ontology representing a temporal, a spatial or a semantic characteristic.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>facetName</b> <i>required</i>	The name of the facet which should be added.	string
<b>Body</b>	<b>facet</b> <i>required</i>	The facet which should be added.	<a href="#">Facet</a>

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Facet</a>
<b>400</b>	There was an error in the request.	No Content
<b>409</b>	The facet already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

**Returns concepts for a facet**

GET /metadata/facet/{facetName}/concept

Description: Returns all possible concepts for a facet of a specific OWL ontology. This means that all subclasses of the facet are returned.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>facetName</b> <i>required</i>	The name of the facet where the concepts should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Concept</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The facet was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Returns all ontologies**

GET /metadata/ontology

Description: Returns all ontologies of the system, which were recorded by the system.

Responses:

HTTP

Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Ontology</a>
<b>400</b>	There was an error in the request.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

**Adds an ontology**

POST /metadata/ontology/{ontologyName}

Description: Adds an ontology to the container management system.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>ontologyName</b> <i>required</i>	The name of the ontology which should be added.	string
<b>Body</b>	<b>ontology</b> <i>required</i>	The ontology which should be added.	<a href="#">Ontology</a>

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Ontology</a>
<b>400</b>	There was an error in the request.	No Content
<b>409</b>	The ontology already exists.	No Content
<b>500</b>	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

**Returns facets of an OWL ontology**

GET /metadata/ontology/{ontologyName}/facet

Description: Returns all possible facets of a specific OWL ontology.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>ontologyName</b> <i>required</i>	The name of the ontology where the facets should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">Facet</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The ontology name was not found.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

#### 4.2.2.4 Service

##### Returns all services

GET /service

Description: Returns the services registered by the Management System with their providers.

Responses:

HTTP Code	Description	Schema
200	successful operation	<a href="#">Ontology</a>
400	There was an error in the request.	No Content
500	An unknown error occurred.	No Content

Produces: application/json

##### Add a service

POST /service/{serviceName}

Description: Adds a service to the Container Management System.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>serviceName</b> <i>required</i>	The name of the service which should be added.	string
<b>Body</b>	<b>servicePair</b> <i>required</i>	The service which should be added.	<a href="#">PhysicalLogicalServicePair</a>

Responses:

HTTP Code	Description	Schema
200	successful operation	<a href="#">LogicalService</a>
400	There was an error in the request.	No Content
409	The service already exists.	No Content
500	An unknown error occurred.	No Content

Consumes: application/json

Produces: application/json

##### Returns a specific service

GET /service/{serviceName}

Description: This method returns the service specified by the parameter.

Parameters:

Type	Name	Description	Schema
<b>Path</b>	<b>serviceName</b> <i>required</i>	The name of the service which should be returned.	string

Responses:

HTTP Code	Description	Schema
<b>200</b>	successful operation	<a href="#">LogicalService</a>
<b>400</b>	There was an error in the request.	No Content
<b>404</b>	The service was not found by the name.	No Content
<b>500</b>	An unknown error occurred.	No Content

Produces: application/json

### 4.2.3 Conclusion

Semantic Containers overcome the problem of stakeholders to query different information providers. A Semantic Container holds all required information for specific information need (e.g., flight from A to B, or service deliver at an airport) and contains data from a single data provider or from multiple data providers. Stakeholders do not have to find services which suit them but request a container with defined data quality (e.g., freshness, locality, and/or aircraft type) and semantic technology allows automatic identification of the best suited container in the available pool.

The second experimental prototype developed within BEST extends the data structures from the first prototype and covers now various meteorological and general aeronautical message types (SIGMET, AIRMET, TAF, METAR, as well as information about FIRs and airports). Additionally, it is based on the modular BEST ontology developed in D1.1.

The next section demonstrates the use of Semantic Containers in an end-to-end scenario and showcases the use of the prototype.

## 5 Semantic Container: Scenario

In this chapter a scenario is described in which BEST is integrated into the SWIM world. Figure 31 gives an overview about the various systems involved in the scenario provided in this task. The goal of the scenario is to give an idea how the TRL1 concept can be used in a complete SWIM lifecycle. For the scenario the Frequentis SWIM Registry **1** was integrated to provide not only information about SWIM services but also about Semantic Containers via SWIM. The BEST Experimental Prototype Evolution 2: Semantic Container Management System **2** is used to define and create containers that are then visible through the SWIM registry. On organizational level the Frequentis SWIM Integration platform (MosaiX) is used **3** to configure organization internal the SWIM information for the specific SWIM applications. And finally, the information is then accessed by a SWIM application. For the BEST integration we used an existing SESAR 1 prototype, namely the Integrated Digital Briefing used **4** from WP13.2.2.

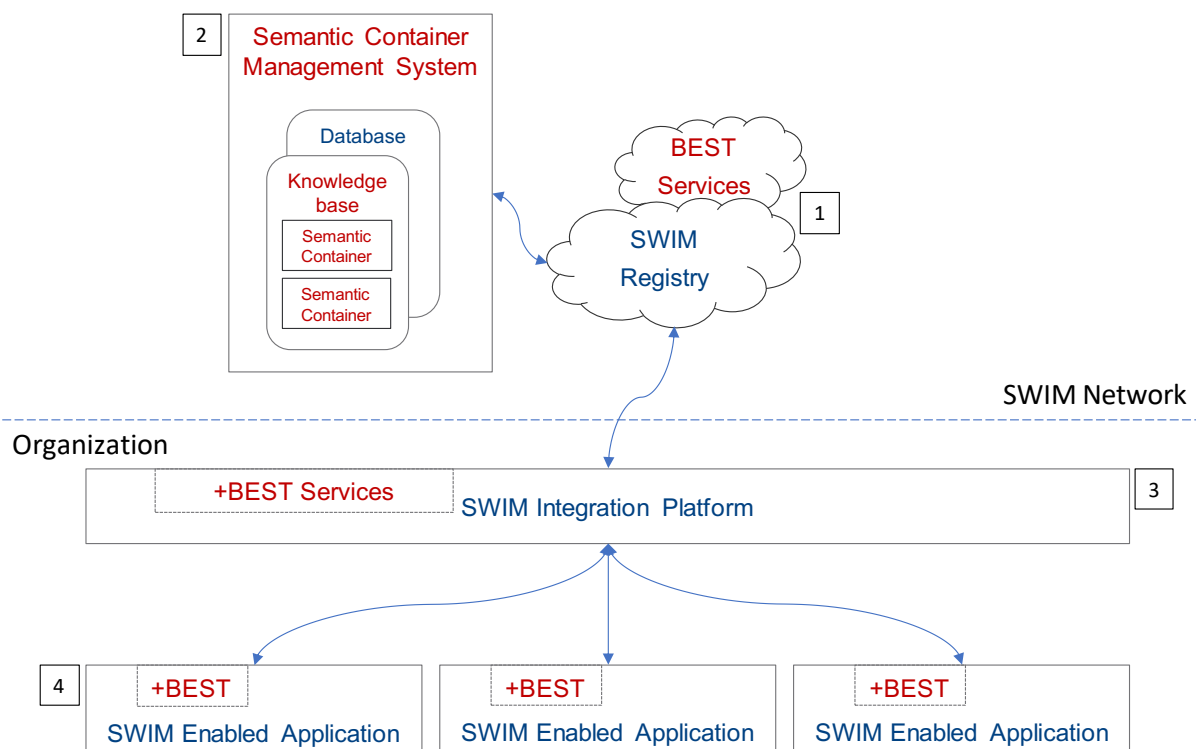


Figure 31: Scenario Overview

## 5.1 BEST Integration: SWIM Registry

As a starting point to demonstrate setup and use of Semantic Containers a SWIM service registry is introduced. This registry provides a list of available SWIM services and Semantic Containers. It also allows to query information about this service providers, e.g., source, content, freshness.

For the current use case a dedicated Frequentis Semantic Container Service Registry was adapted and is available at <https://registry-best.projects.frequentis.com/>.

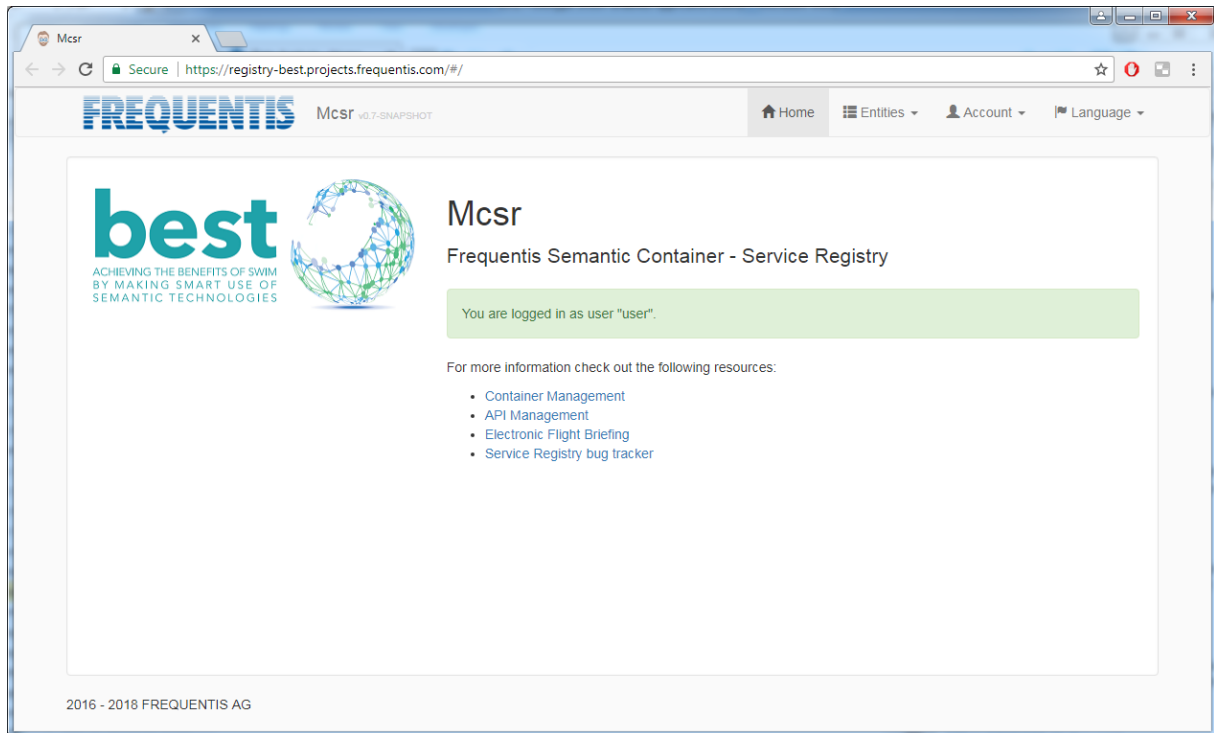


Figure 32: Frequentis Semantic Container – Service Registry

A list of available SWIM services and Semantic Containers is available under Entities > Instances together with the functionality to show details about the services or edit entries.



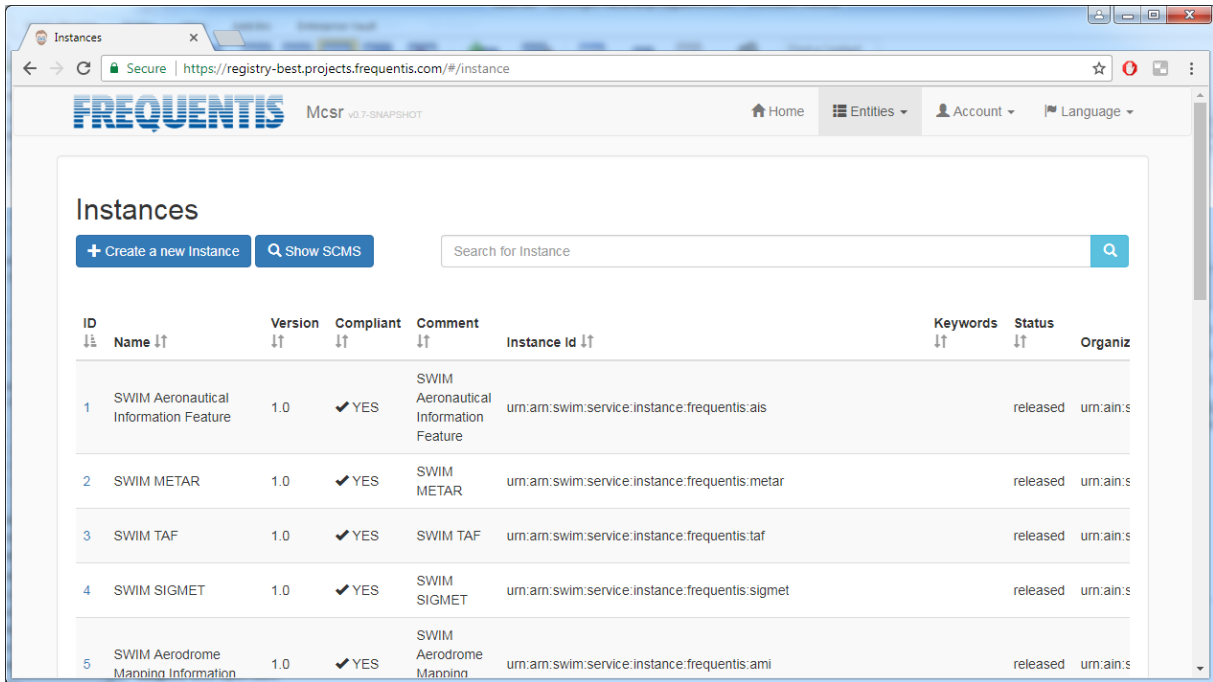


Figure 33: Instances in the Service Registry

Opening the Semantic Container Management System (SCMS) allows showing further details about Semantic Containers: <https://container-best.projects.frequentis.com/#/manageSets>. Please refer to Section 4.2 for a description about the SCMS.

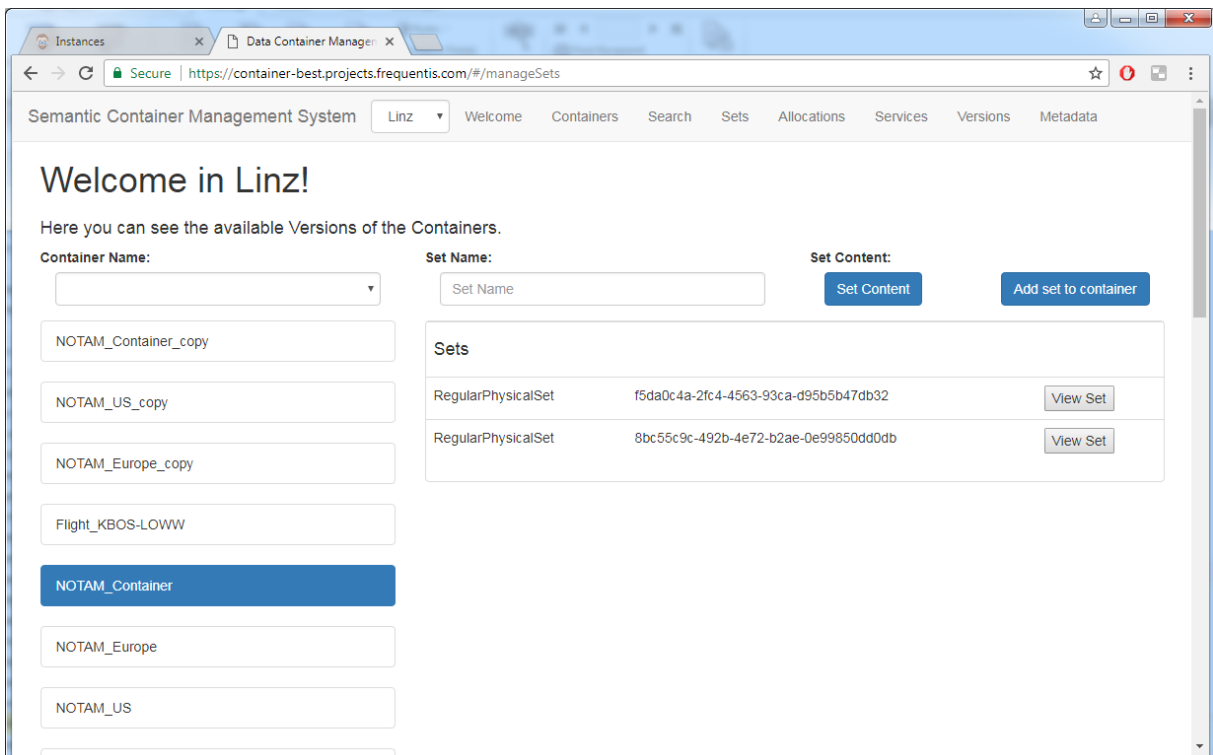


Figure 34: Detailed Container Information in the Semantic Container Management System

Selecting a container allows to show the raw messages stored in the Semantic Container.

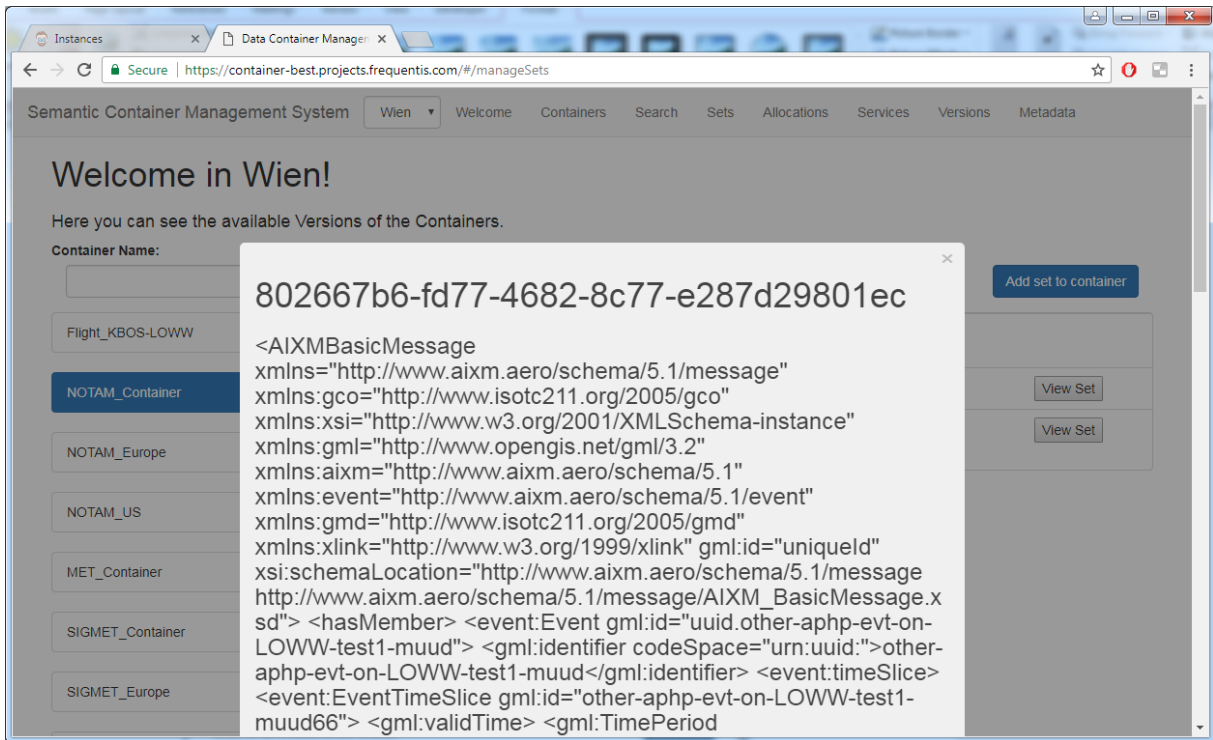


Figure 35: Detailed Container Information in the Semantic Container Management System

Besides showing the content of existing Semantic Containers, it is also possible to create new containers in the “Containers” section of the SCMS as depicted in Figure 36.

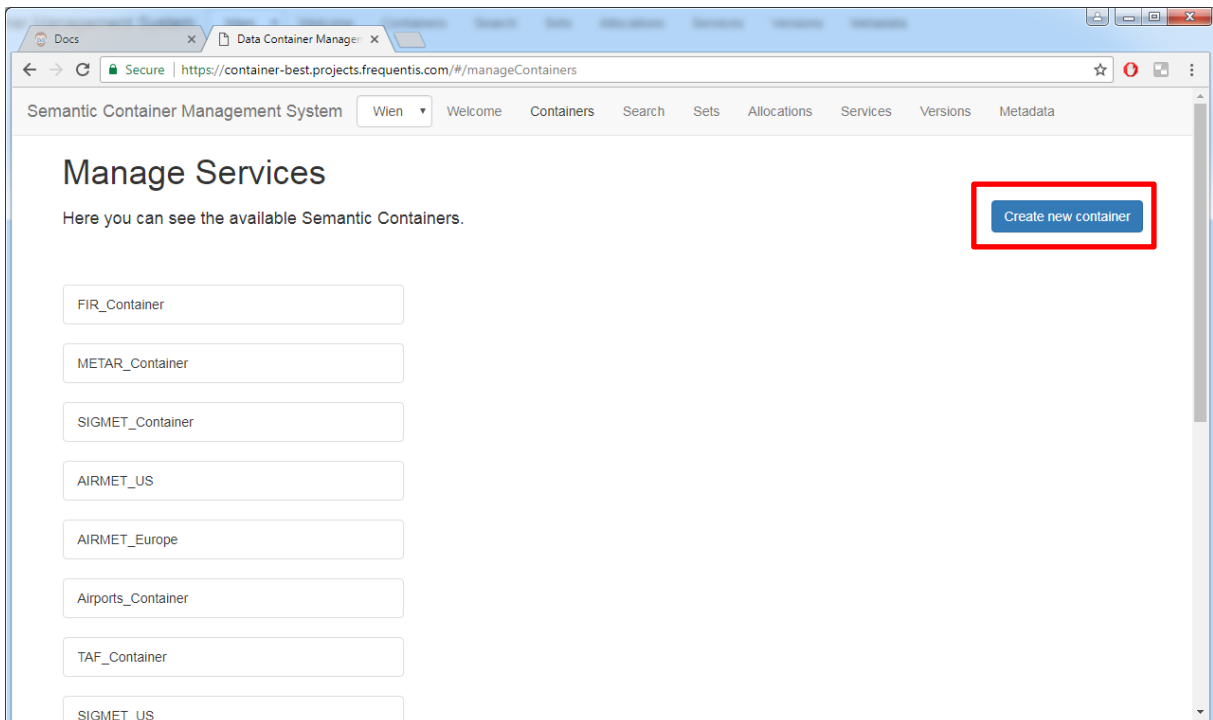


Figure 36: List of Semantic Containers with a Button to create a new Container

## 5.2 Experimental Prototype Evolution 2: Semantic Container Management System

The Semantic Container Management System is used to create and maintain Semantic Containers. In the section “Containers” a new Semantic Container can be created.

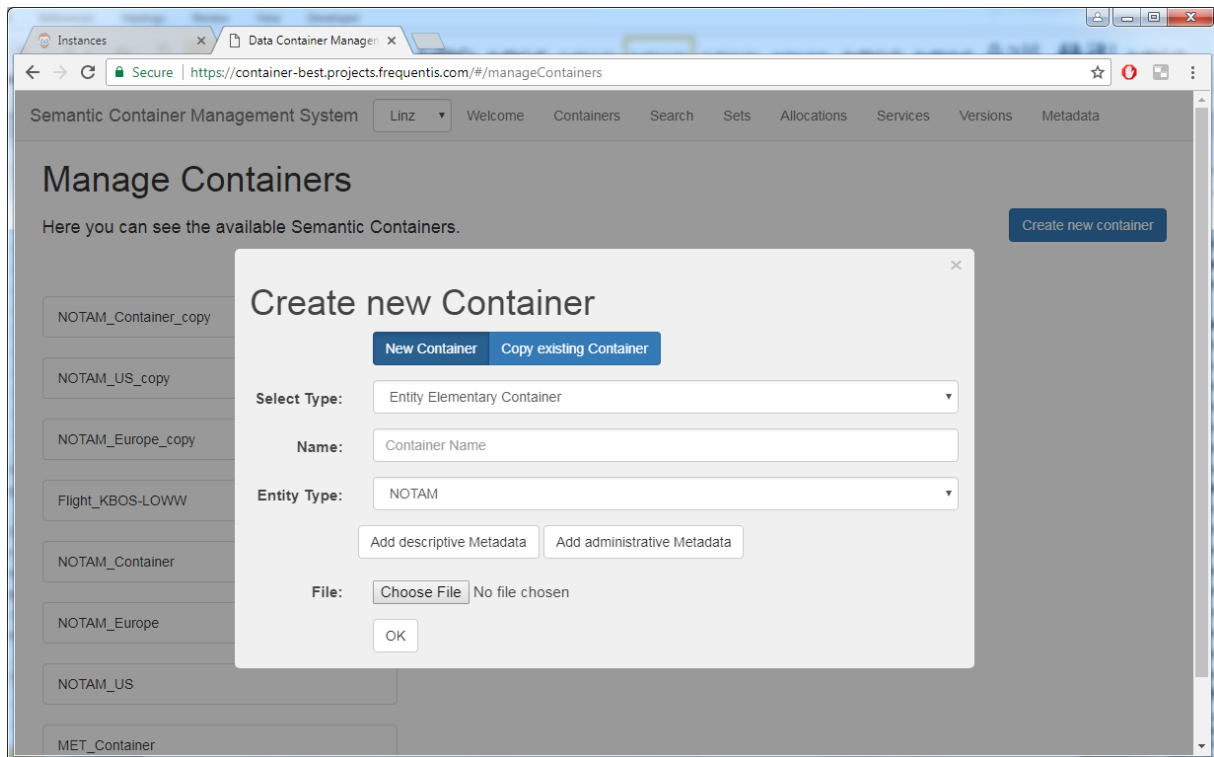


Figure 37: Creating a new Semantic Container

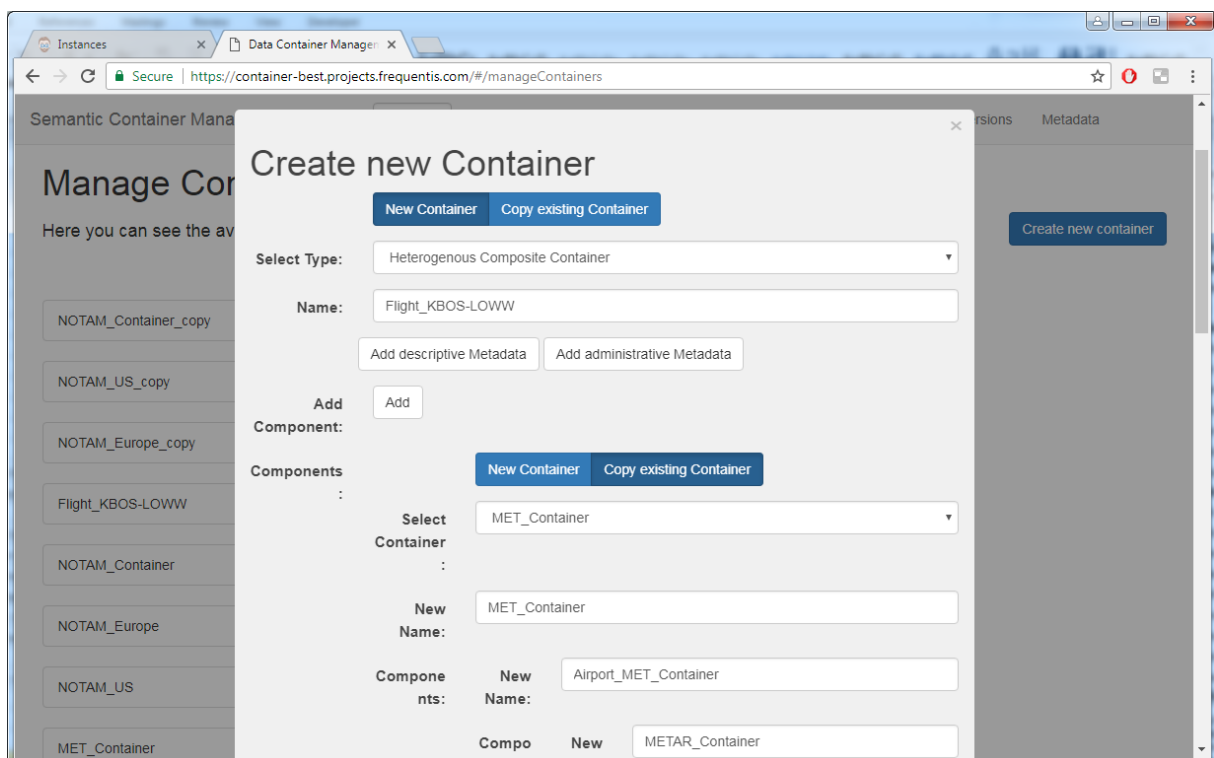
In the presented dialog the user can either create a new container and select an XML file to be used as payload or copy an existing container to create compound containers – see Section 4.2 for a description about the SCMS.

For the scenario presented in this section the following container hierarchy is used:

- Flight\_KBOS-LOWW (Heterogeneous Composite Container)*
  - *MET\_Container (Heterogeneous Composite Container)*
    - *SIGMET\_Container (Homogeneous Composite Container)*
      - *SIGMET\_Europe (Entity Elementary Container)*
      - *SIGMET\_US (Entity Elementary Container)*
    - *AIRMET\_Container (Homogeneous Composite Container)*
      - *AIRMET\_Europe (Entity Elementary Container)*
      - *AIRMET\_US (Entity Elementary Container)*
    - *Airport\_MET\_Container (Heterogeneous Composite Container)*
      - *METAR\_Container (Entity Elementary Container)*
      - *TAF\_Container (Entity Elementary Container)*

- *NOTAM\_Container (Homogeneous Composite Container)*
  - *NOTAM\_Europe (Entity Elementary Container)*
  - *NOTAM\_US (Entity Elementary Container)*
- *Aeronautical\_Information\_Container*
  - *Airports\_Container (Entity Elementary Container)*
  - *FIR\_Container (Entity Elementary Container)*

Note: It is also possible to already create a Semantic Container restricted to a specific aircraft type. In this case it would be necessary to already use a tailored NOTAM container for the specific aircraft type.



**Figure 38: Semantic Container Setup for the Scenario “Flight from KBOS to LOWW”**

After creating the Semantic Container, the container hierarchy can be explored in the SCMS: select a container and display the information in the right-hand side panel as depicted in Figure 39.

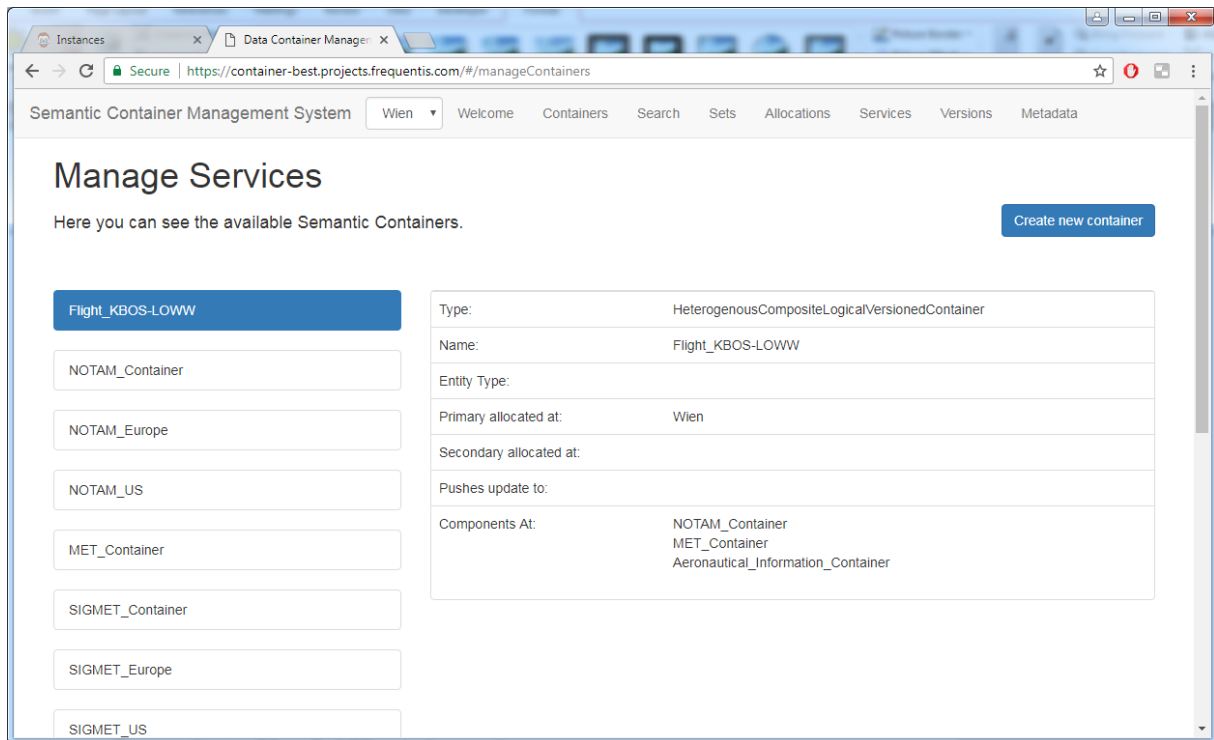


Figure 39: Container hierarchy of composite container

Based on the assumptions in section 3.1 SWIM Information Analyses the following storage and bandwidth requirements for a flight from Boston (KBOS) to Vienna (LOWW) can be assumed.

	Storage	Download	Upload
<b>Eurocontrol</b>	45,1 GB	2,45 kB/s	122,6 kB/s
<b>FAA</b>	50,0 GB	2,52 kB/s	55,5 kB/s

Table 4: ANSP Storage and Bandwidth Requirements for a Flight from KBOS to LOWW

Comment:

- **Storage:** 45 / 50 GB storage size are based on estimated 10MB airport information per airport – all numbers rely on data given in Table 1; since this is mostly static information only a low download bandwidth is necessary to keep the data current
- **Upload:** the upload is the product of download bandwidth times the number of FIR regions (118 in Europe and 22 in the US); the assumption here is that every FIR region receives data from Eurocontrol or FAA and then serves data to service consumers

### 5.3 BEST Integration: SWIM Integration Platform

To configure available data sources for an organization the Frequentis MosaiX SWIM Management Console is used: <https://management-best.projects.frequentis.com/hawtio/login>

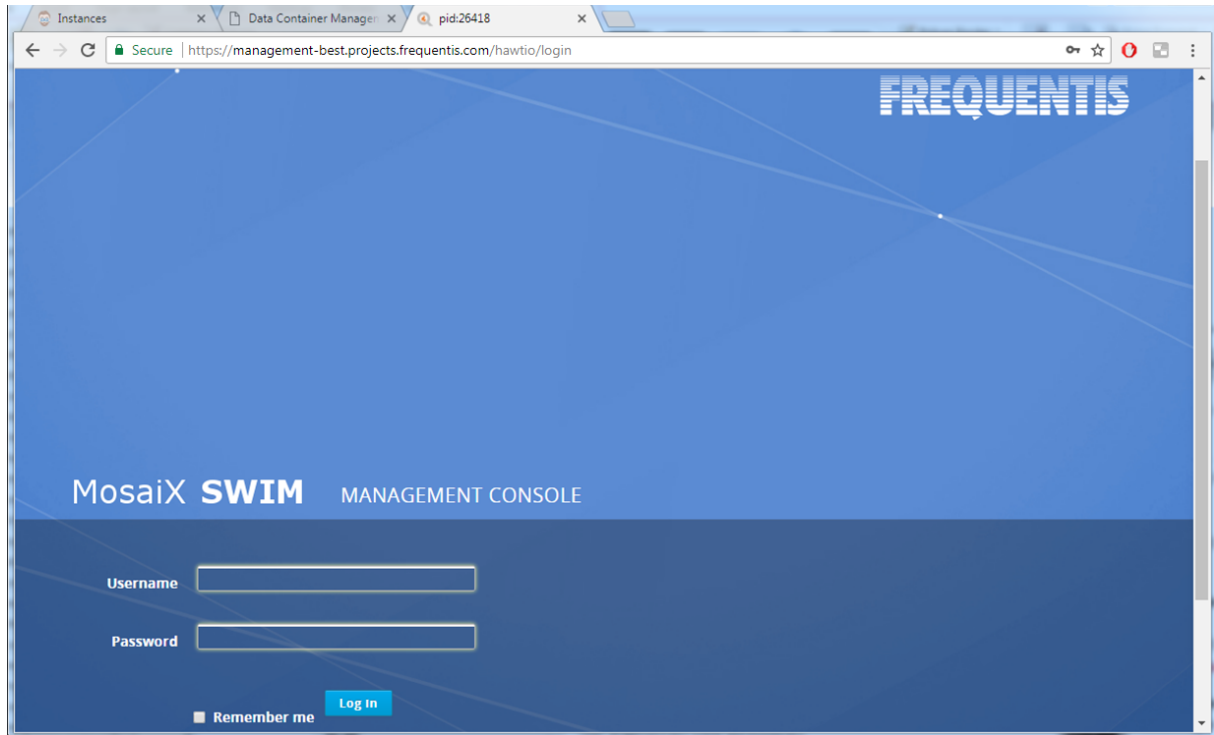


Figure 40: Frequentis MosaiX SWIM Management Console

There it is possible to establish, manage and monitor relevant data sources for an organization to provide access for those entities with legal permission.

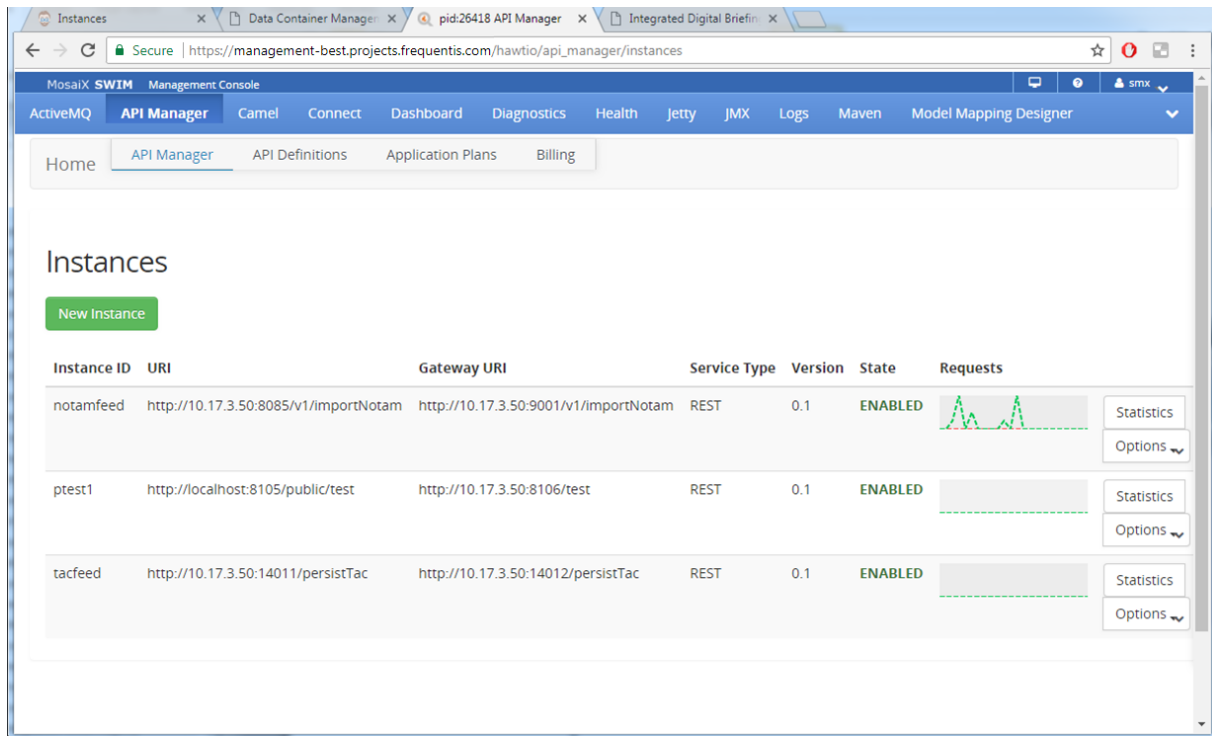


Figure 41: Frequentis MosaiX SWIM Management Console

## 5.4 BEST integration in a SWIM Application: Integrated Digital Briefing

Based on the described use cases in D3.1 Figure 42 shows the components of the SWIM application with integrated Semantic Containers. In red one can see the SWIM services that are used to fill the Semantic Containers needed for the ePIB. The flight plan used is always implied, there was no connection to any flight planning instance.

The Integrated Digital Briefing prototype is shown in green. Its main components were the actual ePIB composer, the Aeronautical Maps layers composer and the GeoServer that provided the maps information as WFS or WMS. The prototype was itself a service. The ePIB HMI contains the web UI components that provide the interactive platform for the preparation of the ePIB including the graphical/map, the text, and the control widgets.

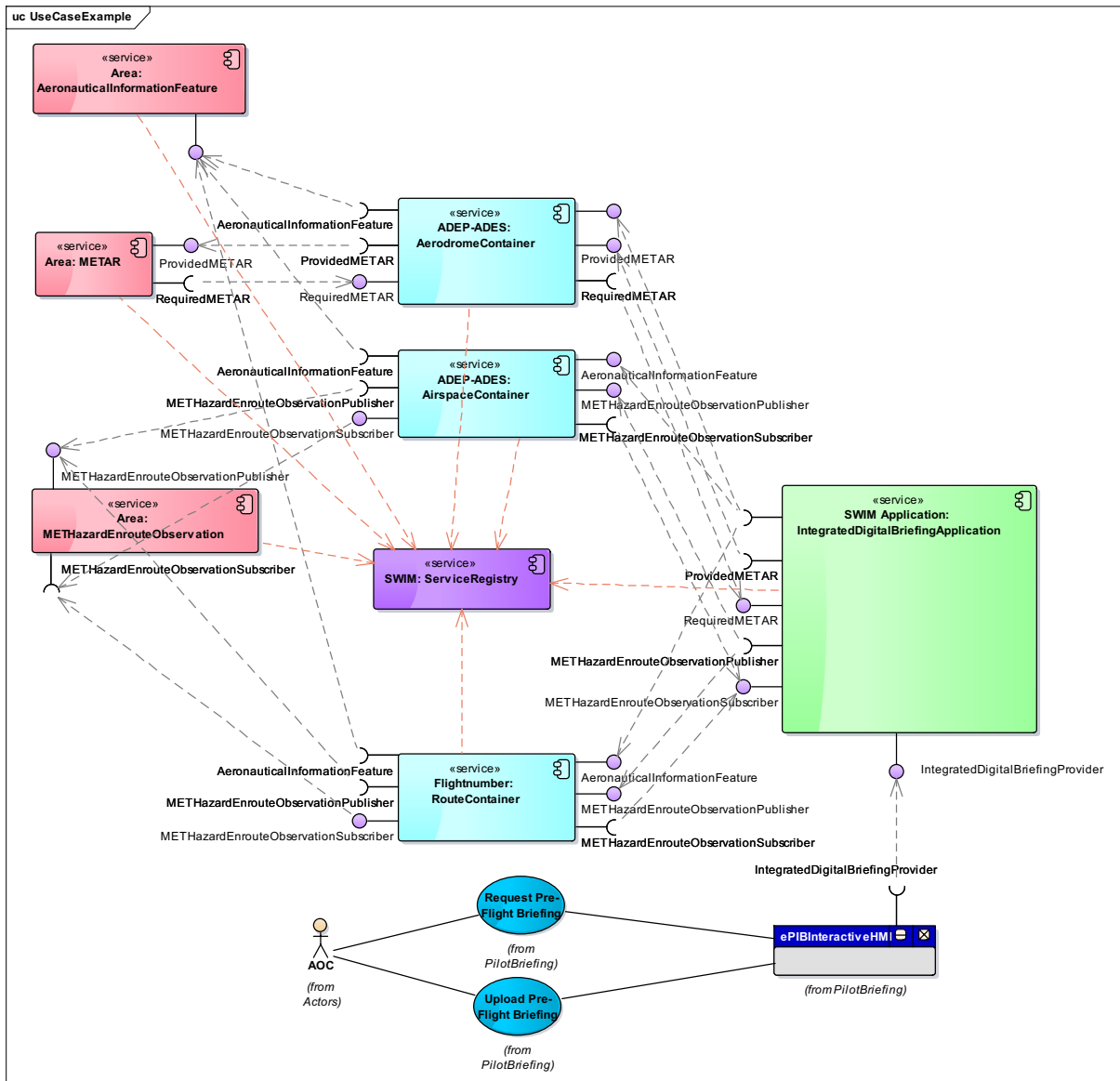


Figure 42: Integrated Digital Briefing BEST Use Case

The SWIM application is managed by the organizational SWIM integration platform (see Figure 31), which is responsible for the service management, data mediation and other configuration options. Since the SWIM Integration Platform is also used as the access point to the SWIM Registry all registered SWIM services and Semantic Container services are available. For SWIM applications it is completely transparent to connect to either a SWIM service or a Semantic Container. However, to benefit from additional functionality provided in Semantic Containers (i.e., requesting a defined data quality like freshness or locality) also SWIM applications must be adapted for that purpose.





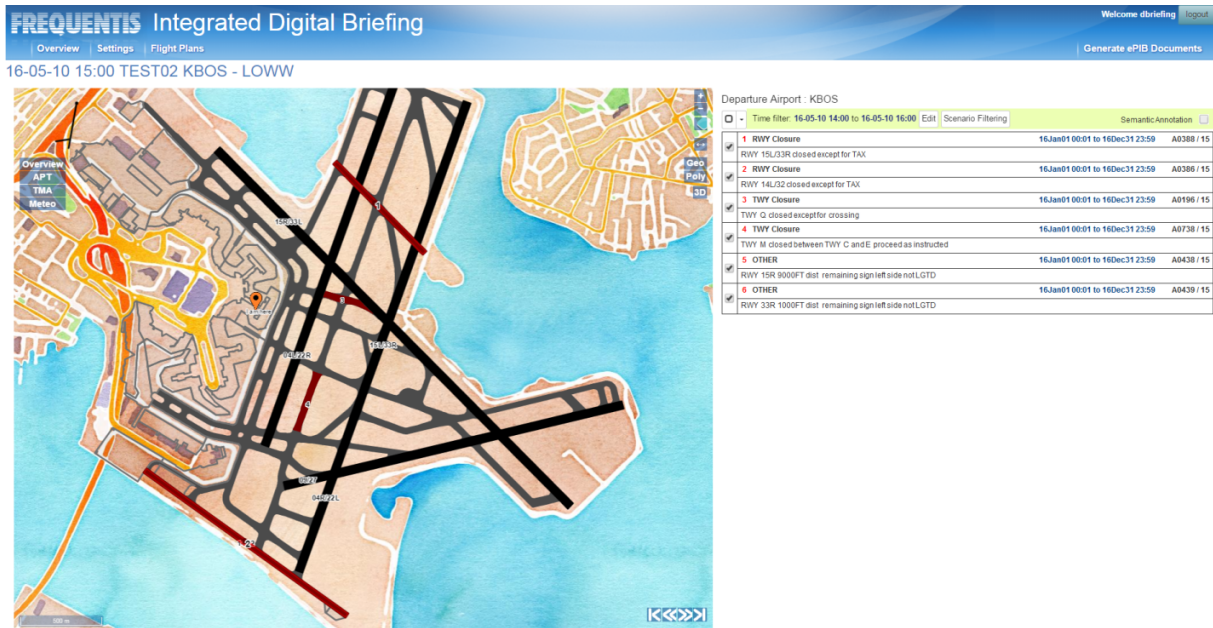


Figure 44: Airport relevant NOTAMs

The prototype has been integrated to use the BEST Semantic Container concept and is able to retrieve containerized information to be used and save calculation time the application normally would need.

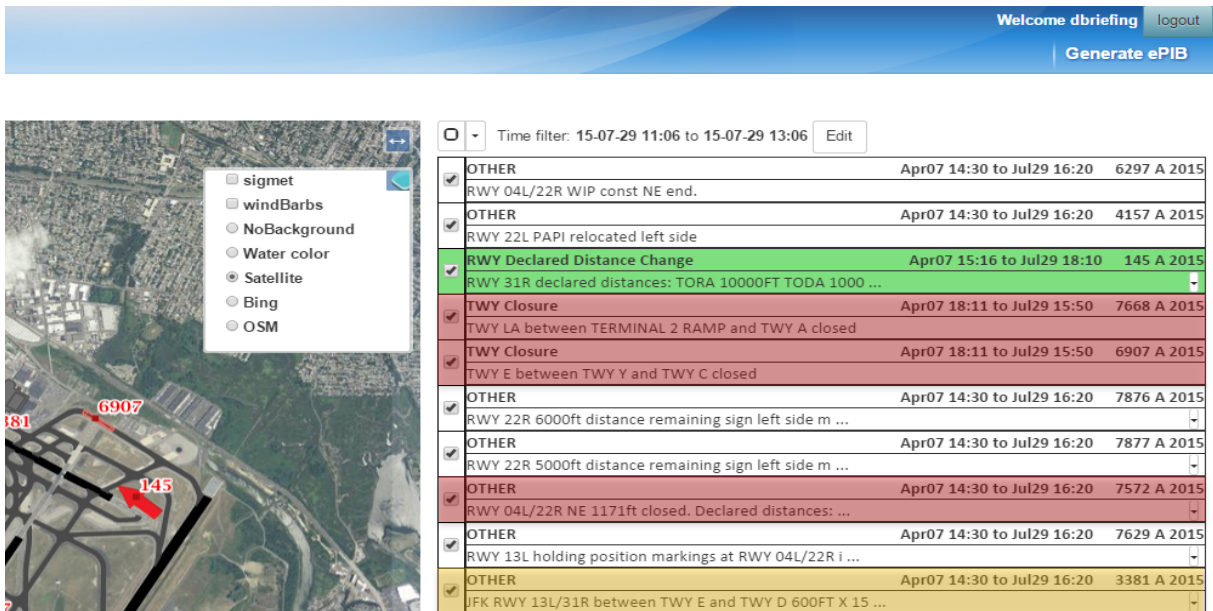


Figure 45: Airport NOTAM retrieved via the related Semantic Container

The integration of the Semantic Container concept into an existing SWIM application showed that it can be used without any changes and only little integration is necessary to visualize the add-value provided by the Semantic Containers.

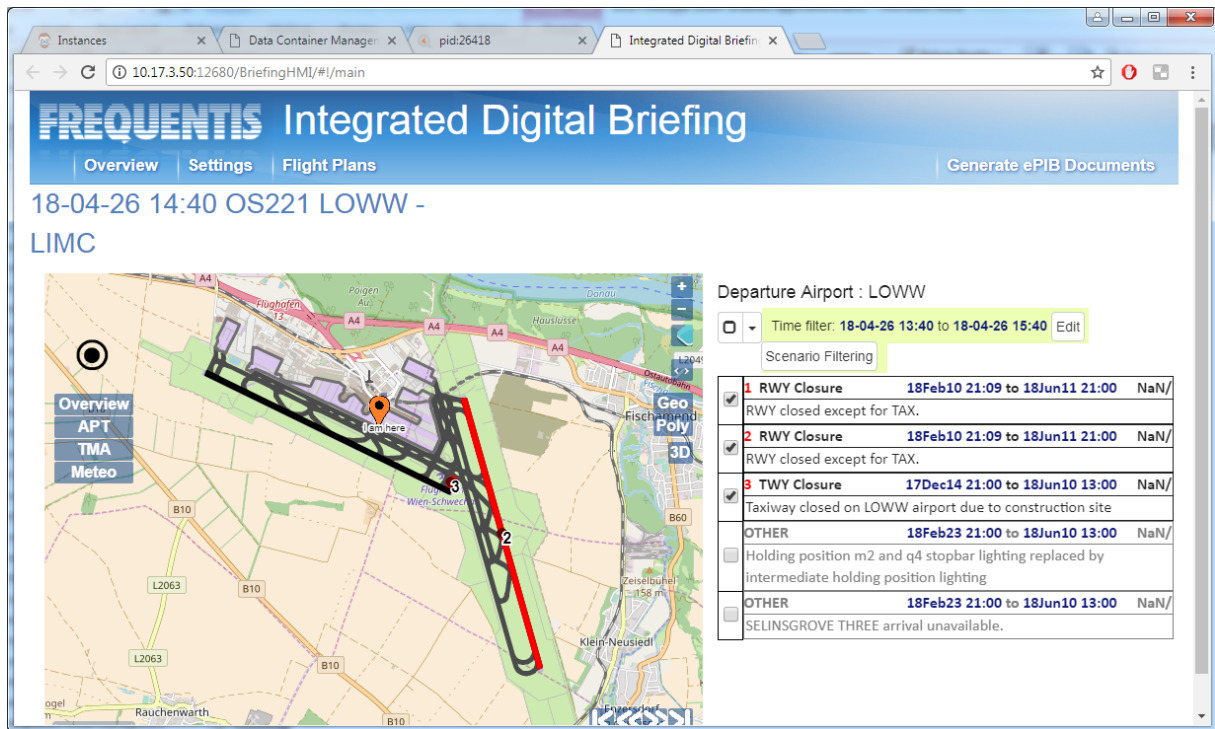


Figure 46: Results for Flight KBOS-LOWW

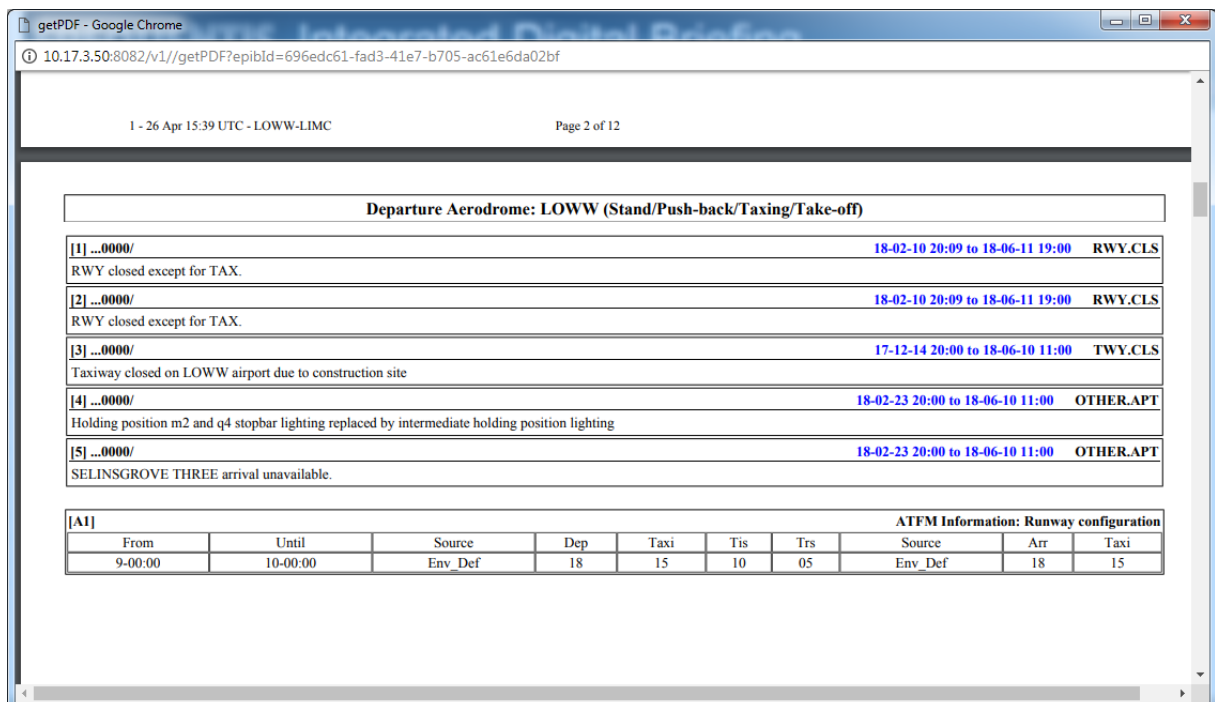


Figure 47: Briefing PDF Export

### 5.4.1 SESAR 2020 Pj17.01 EXE 1

The Semantic Containers have successfully been used in the SESAR 2020 Pj17.01 EXE1 as an integrated part of the SWIM application “Integrated Digital Briefing”. The pre-collected information was then used by other SWIM systems participating in this exercise. The diagram shown in Figure 48 has been delivered as part of PJ.17-01 TRL4 TVALP edition 00.01.00 [9] to the SJU.

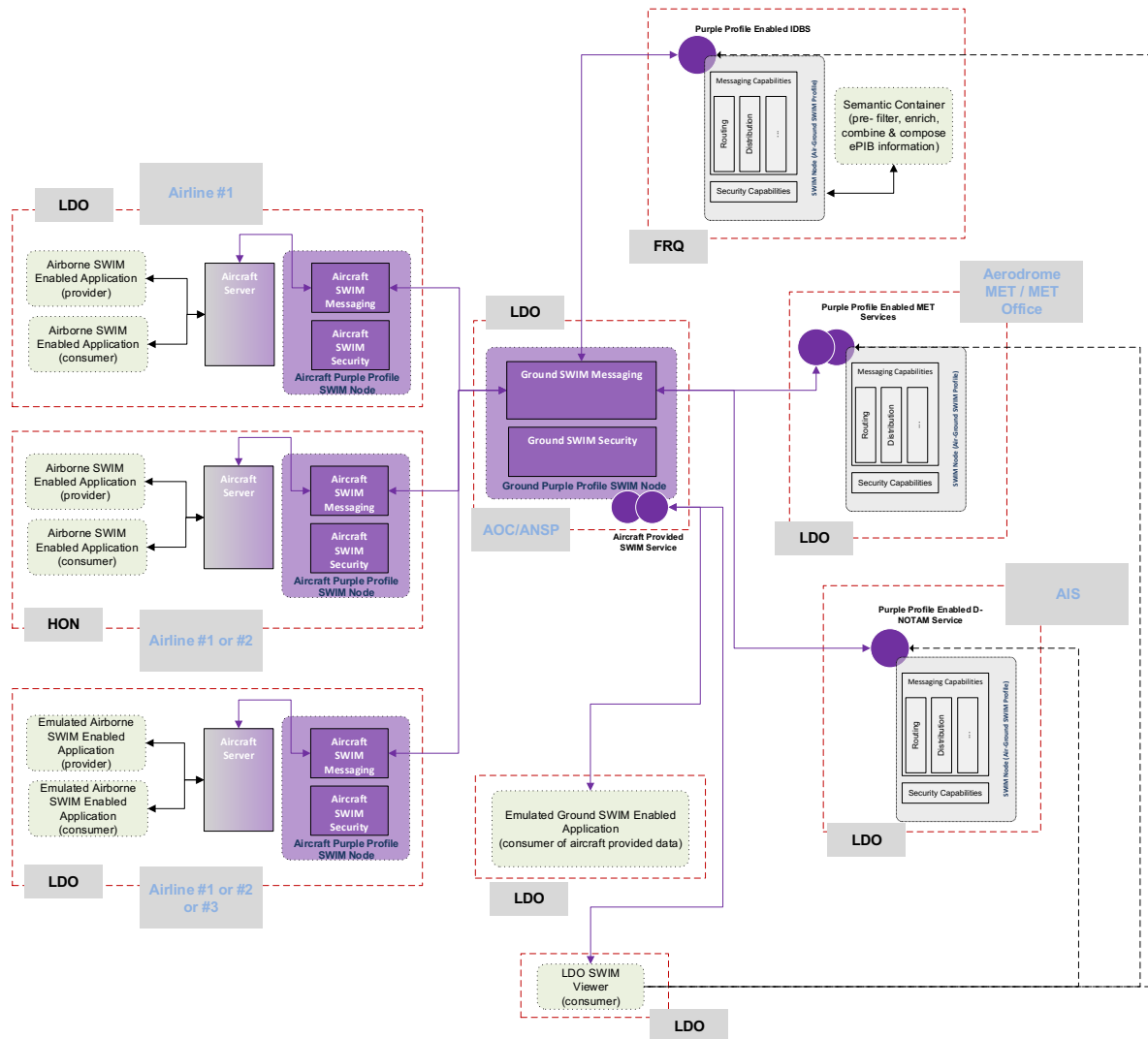


Figure 48: Integration of BEST in Pj17.01 EXE1 [9]

## 6 Conclusion and Future Work

---

The implementation of the SWIM concept enables direct ATM business benefits to be generated by assuring the provision of commonly understood quality information delivered to the right people at the right time. Semantic Containers as described in the BEST project build on this concept and establish additional patterns in such an information network. However, considering that as of today still only a limited number of SWIM services is operational we need to acknowledge that any service on top – like Semantic Containers – will require even more time before they become operational. Nevertheless, more and more SWIM services will become operational over time and it makes sense to already think now about addressing foreseeable bottlenecks that can be solved with Semantic Containers.

This deliverable started with the motivation for the Semantic Containers. Through analysis of the data that is distributed via SWIM an evaluation was conducted that underlined the benefits claimed by the Semantic Container concept. In a separate chapter, the two evolutions of the experimental prototypes of the Semantic Container Management System have been introduced. It has been shown in a proof-of-concept scenario that Semantic Container can extend the SWIM concept and add value to it by data discovery through semantic annotation and leverage necessary benefits in SWIM networks.

Since BEST is a TRL 1 project, the future work shall improve the Semantic Container concept and validate the SWIM integration in a comprehensive manner. This shall include more scenarios, including data from an airline, an airport, and ANSPs and SWIM components like the SESAR2020 SWIM registry.

## 7 References

---

- [1] W. P. HORIZON 2020, “General Annexes - Technology readiness levels (TRL),” European Commission, Brussels, 2015.
- [2] E. Gringinger, C. Fabianek, B. Neumayr and A. Savulov, D3.1 Prototype Use Case Scenarios, 2018.
- [3] C. Schuetz, B. Neumayer, M. Schrefl and E. Gringinger, “D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base,” 2018.
- [4] C. Schuetz, B. Neumayr, M. Schrefl and E. Gringinger, D2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment, 2018.
- [5] A. Vennesland, B. Neumayr, C. Schuetz and A. Savulov, “D1.1 Experimental ontology modules formalising concept definition of ATM data,” 2017.
- [6] I. Kovacic, D. Steiner, C. Schuetz, B. Neumayr, F. Burgstaller, M. Schrefl and W. Scott, “Ontology-based data description and discovery in a SWIM environment.,” in *In Proceedings of the 17th Integrated Communications, Navigation and Surveillance Conference (ICNS)*, Washington D.C., 2017.
- [7] B. Neumayr, E. Gringinger, C. Schuetz, M. Schrefl, W. Scott and A. Vennesland, “Semantic data containers for realizing the full potential of system wide information management.,” in *In Proceedings of the 36th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, St. Petersburg, 2017.
- [8] E. Gringinger, C. Schuetz, B. Neumayr, M. Schrefl and W. Scott, “Semantic data containers for realizing the full potential of system wide information management.,” in *In Proceedings of the 18th Integrated Communications, Navigation and Surveillance Conference (ICNS)*, Washington D.C., 2018.
- [9] SESAR 2020 Pj17.01, “TRL4 TVALP edition 00.01.00,” SJU, Brussels, 2018.
- [10] E. Gringinger, “FREQUENTIS Participates in EUROCAE WG-76 Aeronautical Information and Meteorological Data Link Services,” Frequentis Research Bulletin, Issue 03, 2014.
- [11] ECCAIRS, 1.3.0.12 Data Definition Standard, ICAO.
- [12] Eurocontrol, “ATFCM Operations Manual - Network Operations Handbook,” Network Manager, Brussels, 2017.
- [13] Eurocontrol, “AIXM 5.1,” Eurocontrol, 2014. [Online]. Available: <http://www.aixm.aero/>.

- [14] B. Korn, C. Edinger, S. Tittel, T. Pütz and B. Mohrhard, "SECTORLESS ATM – Analysis and Simulation Results," 27. ICAS (International Council of the Aeronautical Sciences) Conference, 2010.
- [15] K. Reinhardt, M. Poppe and M. Herr, "Increasing Capacity or Productivity with Controller Assistance Tools in High Complexity Airspace," 34th Digital Avionics Systems Conference (DASC), Prague, 2015.
- [16] Eurocontrol, "FIXM 4.0," 2016. [Online]. Available: <http://www.fixm.aero>.
- [17] WMO/ICAO, "ICAO Meteorological Information Exchange Model (IWXXM)," [Online]. Available: <http://schemas.wmo.int/iwxxm/2.1/>. [Accessed 17 10 2017].
- [18] SESAR 2020 Pj10.1b, "Project Description 1.0," SJU, Brussels, 2017.

## 8 Table of Figures

Figure 1: Container Hierarchy for a Flight from Germany to Austria .....	11
Figure 2: Container Hierarchy of a Fuelling Service at an Airport .....	13
Figure 3: Container Hierarchy for an Airline .....	14
Figure 4: Container Hierarchy for a Flight from Sydney to Vienna via Dubai .....	15
Figure 5: Container Hierarchy for Network Manager Operations Centre .....	16
Figure 6: Start page of the Semantic Container Management (1 <sup>st</sup> Evolution) .....	18
Figure 7: List of all available Semantic Containers .....	19
Figure 8: Detailed view of a Semantic Container .....	19
Figure 9: Modifying the properties of a Semantic Container.....	20
Figure 10: Delete a Semantic Container .....	21
Figure 11: Create either a new primary or secondary container .....	22
Figure 12: Select concepts for facets.....	22
Figure 13: Select data set for primary container.....	23
Figure 14: Select the most specific Semantic Container as the input data set .....	24
Figure 15: Select a suitable Semantic Container as the input data set .....	25
Figure 16: Service selection for the secondary container.....	25
Figure 17: Semantic Container Management Platform.....	29
Figure 18: Ontology with Facets and Concepts.....	31
Figure 19: Add an Ontology .....	31
Figure 20: Add Facet to a Semantic Container Management System .....	32
Figure 21: Administrative Metadata.....	32
Figure 22: Specify Facet .....	33
Figure 23: Add Administrative Data to a Container .....	34
Figure 24: Example Search.....	35
Figure 25: Allocate Container.....	35
Figure 26: Allocation view with Allocated Containers.....	36
Figure 27: Add set to a Secondary Allocation .....	37
Figure 28: Inspect Versions .....	38
Figure 29: Creation of a Homogenous Container.....	39
Figure 30: Add a Service.....	39
Figure 31: Scenario Overview.....	55
Figure 32: Frequentis Semantic Container – Service Registry .....	56
Figure 33: Instances in the Service Registry.....	57
Figure 34: Detailed Container Information in the Semantic Container Management System .....	57
Figure 35: Detailed Container Information in the Semantic Container Management System .....	58
Figure 36: List of Semantic Containers with a Button to create a new Container .....	58
Figure 37: Creating a new Semantic Container .....	59
Figure 38: Semantic Container Setup for the Scenario “Flight from KBOS to LOWW” .....	60
Figure 39: Container hierarchy of composite container.....	61
Figure 40: Frequentis MosaiX SWIM Management Console .....	62
Figure 41: Frequentis MosaiX SWIM Management Console .....	63
Figure 42: Integrated Digital Briefing BEST Use Case .....	64
Figure 43: Flight Overview .....	65
Figure 44: Airport relevant NOTAMs.....	66



Figure 45: Airport NOTAM retrieved via the related Semantic Container ..... 66

Figure 46: Results for Flight KBOS-LOWW ..... 67

Figure 47: Briefing PDF Export ..... 67

Figure 48: Integration of BEST in Pj17.01 EXE1 [9] ..... 68

Figure 49: Digital Integrated Briefing concept highlighting the Dispatcher Briefing [10]..... 79

## 9 Abbreviations

Acronym/Abbreviation	Explanation
ACC	Area Control Centre
ADEP	Departure Airport Operator
ADES	Destination Airport Operator
AIM	Aeronautical Information Management (Data Provider)
AIXM	Aeronautical Information Exchange Model
AIRM	ATM Information Reference Model
ALT	Alternate Airport Operator
ANSP	Air Navigation Service Provider
AOC	Airline Operations Centre
AOWIR	Aircraft Operator What-If-Reroute
APP	Approach
ARR	Arrival
ATC	Air Traffic Control Operator
ATFCM	Air Traffic Flow and Capacity Management
ATS	Air Traffic Services
ATM	Air Traffic Management
BEST	Achieving the BEnefits of SWIM by making smart use of Semantic Technologies
CPT	Captain (Operating Pilot)
CRCO	Central Route Charges Office
CTO	Controlled Time of Arrival
CTOT	Calculated Take-Off Time
CTR	Centre
DEP	Departure
DFS	Deutsche Flugsicherung
DNOTAM	Digital NOTice To AirMen
EAD	European AIS Database
EN-RTE	En-Route
EOBT	Estimated Off Block Time
ePIB	enhanced-PIB

ETA	Estimated Time of Arrival
ETFMS	Enhanced Tactical Flow Management System
FDP	Flight Data Processing
FEI	Flight Efficiency Initiative
FIR	Flight Information Regions
FIXM	Flight Information Exchange Model
FMP	Flow Management Positions
FO	First Officer (Operating Co-Pilot)
FPL	Flight Plan
GCAA	General Civil Aviation Authority
GND	Ground
HMI	Human-Machine Interface
IATA	International Air Transport Association
ICAO	International Civil Aviation Organization
IFR	Instrument Flight Rules
ISRM	Information Service Reference Model
IWXXM	ICAO Meteorological Information Exchange Model
LOWL	Linz Airport (IATA: LNZ, ICAO: LOWL)
LOWW	Vienna International Airport (IATA: VIE, ICAO: LOWW)
MEP	Message Exchange Pattern
MET	Meteorological Information (Data Provider)
METAR	Meteorological Terminal Air Report
NCMS	National Center of Meteorology & Seismology
NID	Network Impact Display
NM	Network Management Operator
NMOC	Network Manager Operations Centre
NOAA	National Oceanic and Atmospheric Administration
NOTAM	NOTice To AirMen
OMDB	Dubai International Airport (IATA: DXB, ICAO: OMDB)
PIB	Pre-flight Information Bulletin
RRP	Rerouting Proposal
RFP	Replacement Flight Plan Procedure
SCMS	Semantic Container Management System

SIGMET	Significant Meteorological Information
SWIM	System Wide Information Management
TAF	Terminal Area Forecast)
TMA	Traffic Management Advisor
TRL	Technical Readiness Level
TWR	Tower
WP	Work Package
WXXM	Weather Information Exchange Model
YSSY	Sydney Kingsford Smith Airport (IATA: SYD, ICAO: YSSY)

## 10 APPENDIX A: Tools & Libraries for Semantic Container Management System

This section lists tools and libraries used in the Semantic Container Management System. Each tool and library is stated with the corresponding version and license.

Tools	Version	License
Atom	1.26.1	MIT Licence
Bower	1.8.0	MIT License
Firefox Quantum	59.0.2	Mozilla Public License 2

Table 5: Frontend Tools

Library	Version	License
angular	1.5.8	MIT License
angular-cookies	1.5.8	MIT License
angular-route	1.5.8	MIT License
bootstrap	3.3.7	MIT License
jquery	1.5.8	Open Source License from JS Foundation and other contributors
ng-dialog	0.6.4	MIT License

Table 6: Frontend Libraries

Tools	Version	License
Eclipse IDE for Java Developers Version	Oxygen.3a Release (4.7.3a) Build id: 20180405-120	Eclipse Public
Apache Maven	3.5.3	Apache License, version 2.0
Java JDK	1.8.0_51	Oracle Binary Code License
Java JRE	1.8.0_51	Oracle Binary Code License

Table 7: Backend Tools

Library	Version	License
jersey-container-grizzly2-http	2.26	COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.1
jersey-container-grizzly2-servlet	2.26	COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.1
jersey-hk2	2.26	COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.1
jackson-jaxrs-json-provider	2.9.3	Apache License Version 2.0
swagger-jersey2-jaxrs	1.5.18	Apache License Version 2.0
apache-jena-libs	3.6.0	Apache License Version 2.0
jena-fuseki-embedded	3.6.0	Apache License Version 2.0
basex	8.6.7	The 2-Clause BSD License
xqj-api	1.0	<a href="https://github.com/ligasgr/intellij-xquery/blob/master/licenses/xqj-api-license.txt">https://github.com/ligasgr/intellij-xquery/blob/master/licenses/xqj-api-license.txt</a>
basex-xqj	9.0	Apache License Version 2.0
slf4j-api	1.7.5	<a href="https://opensource.org/licenses/mit-license.php">https://opensource.org/licenses/mit-license.php</a>
slf4j-log4j12	1.7.5	<a href="http://www.opensource.org/licenses/mit-license.php">http://www.opensource.org/licenses/mit-license.php</a>

Table 8: Backend Libraries

# 11 APPENDIX B: Integrated Digital Briefing

For SESAR 1 SWP 13.02.02 Frequentis has produced an interactive Integrated Digital Briefing prototype application. The prototype’s focus was set on commercial pilots flying under instrument rules. The main function of the prototype was the interactive preparation of an enhanced Pre-flight Information Bulletin (ePIB). To be noted, the ePIB prepared by the original SESAR 13.02.02 prototype was already filtered and organized in phases of flight. Limited temporal and geographical filtering was already available and to be provided by another service.

What the original prototype lacked was the semantic pre-analysis of the already filtered information to be organized and ordered based on its relevance to a specific flight configuration into Semantic Containers. The Semantic Container contains the information set for the operational application. This section shows how existing SWIM applications can be integrated to use semantical enriched benefits provided by a Semantic Container.

A brief view on the main purpose for Integrated Digital Briefing is given below in Figure 49. The main function of the Digital Briefing Application is to retrieve all the data necessary for preparing the ePIB from various sources available over a distributed (SWIM) network, concentrate that data and produce a briefing package for the end users, mainly the pilots, airline dispatchers and/or the airport reporting offices (ARO). A more particular case is the briefing of ATC/ATM operators, but that is out of scope here.

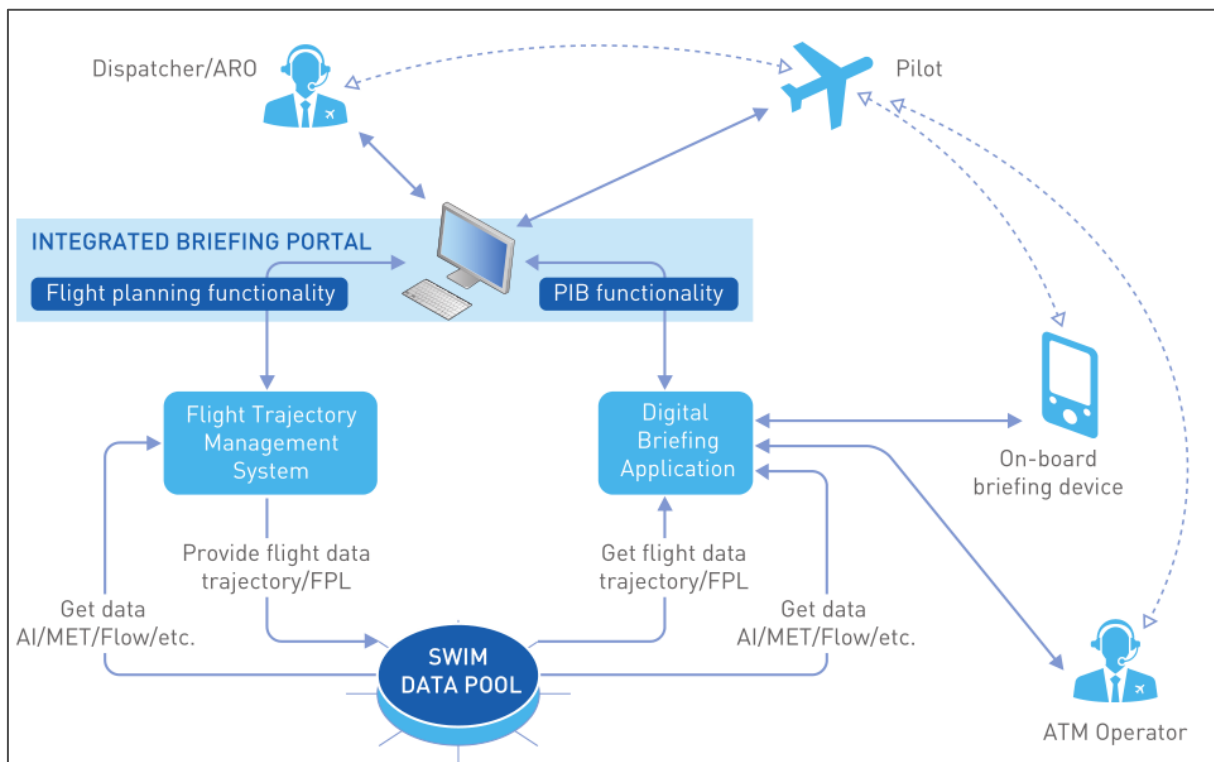



Figure 49: Digital Integrated Briefing concept highlighting the Dispatcher Briefing [10]

## The BEST consortium:

<p><b>SINTEF</b></p>	
<p><b>Frequentis AG</b></p>	
<p><b>Johannes Kepler Universität (JKU) Linz</b></p>	
<p><b>SLOT Consulting</b></p>	
<p><b>EUROCONTROL</b></p>	