



Tutorial for Software Developers

D4.4

BEST

Grant:

699298

Call:

H2020-SESAR-2015-1

Topic:

Sesar-03-2015

Information Management in ATM

Consortium coordinator:

SINTEF

Dissemination Level:

PU (public)

Edition date:

29 May 2018

Edition:

00.01.00

Founding Members



Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Audun Vennesland (SINTEF)	Project Member	28.05.2018
Eduard Gringinger (FRQ)	Project Member	28.05.2018
Reza Mirhossein (SLOT)	Project Member	28.05.2018

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Joe Gorman (SINTEF)	Project Coordinator	30.04.2018
Scott Wilson	Eurocontrol	23.05.2018

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

Name/Beneficiary	Position/Title	Date
Approved by all partners, in accordance with procedures defined within the consortium.		29.05.2018

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date

Document History

Edition	Date	Status	Author	Justification
00.00.01	21.02.2018	PCOS	Reza Mirhossein	Prepared document for PCOS review
00.00.02	05.03.2018	PCOS	Reza Mirhossein	Corrections from PCOS internal review
00.00.03	23.04.2018	Intermediate	Reza Mirhossein	Prepared for internal review
00.00.04	17.05.2018	External proposed	Reza Mirhossein	Add internal review comments
00.00.05	29.05.2018	Final	Reza Mirhossein	Add minor changes from internal review

00.01.00	29.05.2018	Released	Joe Gorman	Removed track changes and comments. Updated administrative information on initial pages, ready for delivery to SJU.
----------	------------	----------	------------	---



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

The purpose of this deliverable is to facilitate software development process by giving a tutorial-style overview of the project results. It explains the tools and fundamental concepts that will help software developers to perceive the primary guidelines for further development.

The document outlines the primary purposes of the developed tools/concepts as well as basic instructions to utilize them in further developments and industry use cases.

Originally this deliverable aims to address two different result groups. The first group is interested in using the software tools developed by BEST (ontologies, ontology generation scripts, ontology modularisation scripts and the AIRM compliance validator) for further developments. The second group is more interested in the concepts (such as the semantic container concept) and developed prototypes to evaluate their feasibility and usefulness. Associated sections of this deliverable serve crucial information to elaborate the objectives and implementation methodologies mainly in ontology development, modularization, compliance validator and semantic container.

The expected takeaway of this tutorial would be transferring main concepts and necessary steps to use the project results effectively in software development process.

Table of Contents

Abstract/Executive Summary	4
1 Introduction	6
1.1 Purpose	6
1.2 Intended Readership	7
1.3 Relationship to other Deliverables & Document Structure	8
1.4 Acronyms and abbreviations.....	10
2 Results Group 1: Software Developed in BEST suitable for direct use in model and software development.....	12
2.1 Ontologies and Ontology Generation.....	12
2.1.1 What is the purpose of the tools?	13
2.1.2 How do I use the tools?	13
2.2 Modularising ontologies	16
2.2.1 What is the purpose of the tools?	16
2.2.2 How do I use the tools?	18
2.3 Validating compliance and ontology matching	22
2.3.1 What is the purpose of the tool?.....	22
2.3.2 How do I use the tool?.....	22
3 Results Group 2: Concepts and Prototypes developed in BEST for illustrating the usefulness and feasibility.....	28
3.1 Semantic Container Concept.....	28
3.2 Semantic Container Management System (Prototype)	30
3.2.1 How does the Distribution and Replication Work?	30
3.2.2 How does Versioning and Consistency Management work?	32
3.2.3 How is Lineage and Provenance represented?	33
3.2.4 How does the System Architecture look like?.....	35
3.3 Integration of Semantic Containers in a SWIM Environment	36
References	37

1 Introduction¹

1.1 Purpose

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

This deliverable will describe how semantic technologies can be implemented in a SWIM environment. It will include lessons learned from the developments in BEST and be targeted towards software developers of ATM applications.

It is clear from this definition that the intended audience for this deliverable consists of software developers interested in how they might benefit from results of the BEST project. But it is unclear from this GA description what “implementing” semantic technologies would mean. Given that the deliverable is from WP4 “Stakeholder Awareness and Relevance” whose stated objective us “to accomplish the communication needs of the project”, it makes sense to interpret “implemented” as meaning something like “understood and used effectively”. With this context, we define the purpose of the deliverable more precisely as being:

to provide a tutorial-style overview of the key results of the project that are of specific interest to people responsible for development of ATM applications, providing them with advice on how the results can be used effectively.

The project results of potential interest to software developers fall into two groups, each of which needs a different focus in a tutorial document.

Result Group	Focus of information needed in a tutorial	Relevant project deliverables
Group 1: Software tools developed in the project that could be directly used by people outside the project as part of their work in developing models and/or applications.	<ul style="list-style-type: none"> • What do the tools do, what are they for? • If I want to use them, what do I need to do? 	<ul style="list-style-type: none"> • D1.1 • D1.2 • D5.2

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

Result Group	Focus of information needed in a tutorial	Relevant project deliverables
<p>Group 2: Concepts and prototypes developed in the project with a view to illustrating the usefulness/feasibility of adopting concepts used in the project, but not intended for direct use by developers in their own development processes.</p>	<ul style="list-style-type: none"> • What is the concept, and what advantages might it offer me if I use it? • What is needed to use Semantic Containers in a SWIM environment? 	<ul style="list-style-type: none"> • D2.1 • D2.2 • D3.1 • D3.2

It must be emphasised that, as an exploratory research project at TRL 1, it is absolutely *not* the role of BEST to provide prototype tools that are anywhere near the stage of becoming directly deployable, commercial software products. Thus, the “tutorial” information should not be considered as providing “manuals” for the software, such as is typically expected for software products. Rather, its purpose is to provide high-level information about what has been implemented, what it is for and how it can be used.

Full details of the various results are available in the individual deliverables. This tutorial in some cases quotes directly from these separate deliverables, in some cases provides information on parts of their contents in summary form or expressed in a more “tutorial” style, and in some cases provides completely new explanations/summaries. All of this is intended to make this tutorial easily understood by the intended readership, and to make it possible to get an overview without needing to read all the deliverables in full. However, software developers with serious plans to use any of the results would need to refer to the relevant deliverables to get all the information they would need.

1.2 Intended Readership

The tutorial is aimed at software developers who are involved in developing models or application software in the ATM domain, and who are interested in the potential benefits of semantic technology in a SWIM environment.

It may also be useful for IT developers who have a more general interest in aviation digital trends particularly in ATM.

1.3 Relationship to other Deliverables & Document Structure

The table below shows the other BEST deliverables that have been directly used in production of this deliverable. The relationship to this deliverable is in each case that the contents of the deliverable have been partially summarised/augmented with explanations aimed at the intended readership.

The table is organized according to the two “Groups” defined above in section 1.1, and shows – for each deliverable – which sections of this this deliverable provide the tutorial information. From this you can see that the structure of the deliverable reflects the two groups, and the deliverables that are related to each one.

Group	Deliverable	Content	Tutorial information in section(s)
Group 1: Directly usable software	D1.1 Experimental ontology modules formalizing concept definition of ATM data	D1.1 is the deliverable responsible for developing the BEST ontology infrastructure. The ontology infrastructure includes a monolithic ontology developed from the ATM Information Reference Model (AIRM) UML model and a set of ontology modules, each representing different sub-areas of ATM information. Furthermore, the ontologies form a baseline for the establishment of guidelines describing how semantic technologies can be applied to support information exchange in a SWIM environment.	2.1
	D5.2 Ontology Modularisation Guidelines for SWIM	D5.2 describes a set of guidelines for modularising ontologies in a SWIM setting. The ontology modularisation guidelines evaluates the “monolithic” ontology and the ontology modules developed in relation to D1.1 and provide guidelines on how modularisation best can be accomplished in a SWIM operational setting.	2.2

Group	Deliverable	Content	Tutorial information in section(s)
	D1.2 AIRM Compliance Validator	D1.2 is responsible for developing the AIRM Compliance Validator. The AIRM Compliance Validator prototype application will, using techniques from ontology matching and schema matching, contribute to detecting semantic differences between the monolithic ontology and the ontology modules. This can assist in monitoring of compliance between a reference ontology, here represented by the AIRM ontology and ontology modules (represented by the AIXM and IWXXM ontology modules).	2.3
Group 2: Concepts and prototypes illustrating feasibility	D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base	D2.1 proposes a faceted ontology-based description and discovery of semantic containers. It presents experimental results of using common semantic web technologies and the reference ontologies developed in Deliverable 1.1 for realizing the semantic container approach.	3.1 3.2
	D2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment	D2.2 extends the semantic container approach described D2.1 with mechanisms for handling distribution of containers across different nodes, adding provenance information to the administrative metadata, distinguishing between logical and physical containers for distributed allocation.	
	D3.1 Prototype Use Case Scenarios	The use case scenarios defined in D3.1 provides a scope for both the AIRM ontology and the ontology modules.	3.3

Group	Deliverable	Content	Tutorial information in section(s)
	D3.2 Prototype SWIM-enabled applications	The prototype applications developed in D3.2 will demonstrate practicality of the semantic container approach in a SWIM setting.	

1.4 Acronyms and abbreviations

Acronym/Abbreviation	Explanation
ADQ	Aeronautical Data Quality
ANSP	Air Navigation Service Provider
AIRM	ATM Information Reference Model
AIXM	Aeronautical Information Exchange Model
ATM	Air Traffic Management
DNOTAM	Digital NOTAM
F-Logic	Frame Logic
FIXM	Flight Information Exchange Model
IWXXM	ICAO Meteorological Information Exchange Model
METAR	Meteorological Aerodrome Report
NOTAM	Notice To Airmen
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
SESAR	Single European Sky ATM Research
SPARQL	SPARQL Protocol and RDF Query Language
SIGMET	Significant Meteorological Information
SQL	Structured Query Language
TAF	Terminal Aerodrome Forecast
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WSDOM	Web Service Description Ontological Model

XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

2 Results Group 1: Software Developed in BEST suitable for direct use in model and software development

2.1 Ontologies and Ontology Generation

The ontologies developed and employed in BEST are detailed in D1.1 [1]. The transformation scripts developed by BEST in order to produce the ontologies are also detailed in D1.1 [1]. These results are suitable for direct use in model and software development with the limitations obvious from a TRL 1 project's output. The XSLT scripts used in the transformation are available from:

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

The zip file contains the XSLT files, a sample XMI file (that has been processed according to section 2.1.1), a sample OWL file resulting from the transformation, and a readme file.

The OWL ontologies that are available are based on UML models representing the ATM Information Reference Model (AIRM), the Aeronautical Information Exchange Model (AIXM) and the ICAO Meteorological Information Exchange Model (IWXXM). We provide a short introduction of these models here, but further details can be found in D1.1 [1]. All ontologies can be downloaded from the BEST web site at: <http://www.project-best.eu/downloads/ontologies/ontologies.zip>

AIRM is considered the reference standard model which addresses semantic interoperability and provides a harmonised definition for information being exchanged in ATM. Semantic interoperability within ATM is facilitated by ensuring that all information being exchanged within ATM is conformant with the definitions in AIRM. Such compliance is achieved following the rules of compliance defined in the new EUROCONTROL Specification for SWIM Information Definition [2] which was based largely on the SESAR AIRM Compliance Framework [3].

The *AIRM* is decomposed into two main views having different abstraction levels:

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

In the development of the ontological infrastructure, we have only focused on the Logical Data Model since the resulting ontologies require a certain detail level. In addition to classes, the OWL representation also includes properties and associations transformed from UML. These are represented as OWL properties. In order to maintain inter-package relationships present in the original

UML model, we have included the Abstract package and Data Types package in addition to the Subject Field package.

AIXM provides a UML data model and associated XML schemas for representing the format of digitally communicated aeronautical information. *AIXM* defines information related to, among other things, airports and heliports, airspace structures, organisations (including services they provide), geographical points and navigation aids, route information and flying restrictions.

IWXXM is another exchange model that encompasses information about weather phenomenon. This includes actual and forecasted weather reports at aerodromes (METAR and TAF), weather conditions along the route (AIRMET), significant meteorological information (SIGMET), and advisories related to volcanic ash events and other extreme meteorological conditions (e.g. cyclones). As with *AIXM*, the UML model is targeted for XML schema development, something that makes it challenging for a completely automated transformation to OWL.

2.1.1 What is the purpose of the tools?

BEST provides a set of ontologies which can be picked up and used. However, as detailed above, the scope of the ontologies is limited. They can be used with these limitations in mind. However, the software developer may need to go beyond the OWL representations produced by BEST.

The purpose of the transformation scripts is to automatically obtain an OWL representation from the original UML models. The result from the transformation process is hence an OWL file representing the entire source UML model (monolithic representation), which can be useful in itself, or it can be subject to further modularisation by following the process described in section 2.2.

The AIRM and AIXM UML models are transformed into OWL format as an automated process using Extensible Stylesheet Language Transformation (XSLT) transformation scripts developed in the project. The IWXXM ontology has been developed manually. The resulting ontologies have different scope and size. The AIRM is transformed to a single (monolithic) OWL ontology, while the AIXM and IWXXM modules are represented as ontology modules, i.e. subsets of a more complete ontology representing the original UML model.

2.1.2 How do I use the tools?

The overall approach for transforming from source UML models to OWL is illustrated in Figure 1:

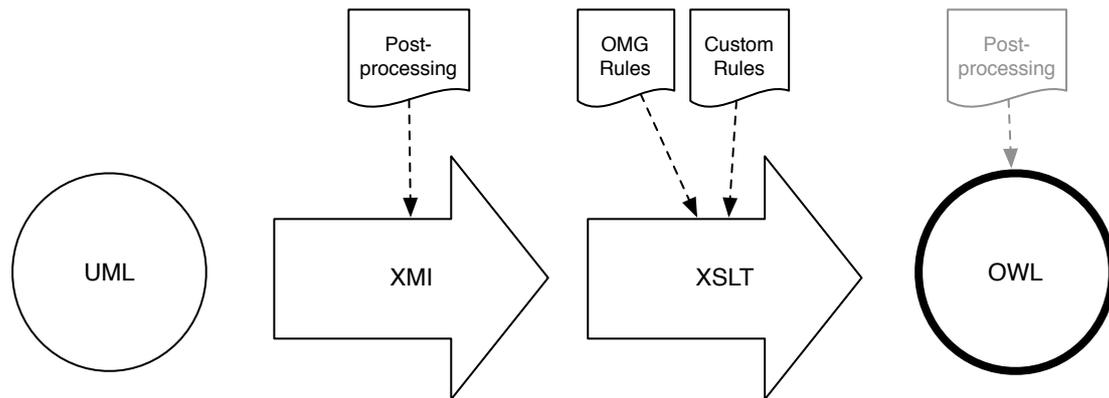


Figure 1. Transformation from UML to OWL

XMI allows for a straightforward parsing and processing of elements from the UML model. In BEST, the UML editor Sparx Enterprise Architect was used to generate an XMI file. In some cases, the XMI file resulting from the generation needs some minor post processing. For example:

Redundant data type declaration

During the transformation using OxygenXML, the SAXON parser throws an error since there are two data types declared for “PackagedElement.PackagedElement.OwnedAttribute.upperValue. This types are ‘uml:LiteralInteger’ and ‘uml:LiteralUnlimitedNatural’. This can be resolved by removing ‘uml:LiteralInteger’.

Un-needed elements

The XMI file contains elements that are irrelevant for the transformation so the following changes should be made on the XMI file both to ease manual inspection (if needed) and processing time:

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

Whitespace removal

Whitespace present in UML elements are maintained in the XMI. Especially in code list entries this was the case. This could be resolved by doing a search-replace (“ ” -> “”) of the XMI file in OxygenXML.

To ensure that the semantics of the UML constructs are correctly transformed to semantics of the OWL constructs, the rules for mapping between UML and OWL specified by OMG (Object Management Group) are followed. However, we have extended the OMG specification with a few additional rules. Table 1 describes the rules that are used to transform from a UML construct to an OWL construct.

Table 1. Mapping rules adapted from OMG [4]

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

Once an OWL representation is in place, some post-processing for some of the OWL ontologies is required. This relates to ensuring a good representation of <<choice>> constraints and UML association classes (a detailed description of this is provided in D1.1 [1]).

The main XSLT file is `airm_xslt_xmi2owl_Main.xsl` which imports the other XSLT files in the folder. These include:

- `airm_xslt_xmi2owl_Classes.xsl`
- `airm_xslt_xmi2owl_ObjectProperties.xsl`
- `airm_xslt_xmi2owl_DatatypeProperties.xsl`
- `airm_xslt_xmi2owl_Individuals.xsl`

In BEST, we have used the XML editor OxygenXML² to do the actual transformation from UML to OWL, and the OWL editor Protégé to evaluate intermediate results. The representation syntax for the resulting OWL file is RDF/XML.

² <https://www.oxygenxml.com/>

Some additional details related to the ontology development in BEST can be found in deliverable D1.1 [1] and we encourage the reader to consult this deliverable for a complete understanding of this development process.

2.2 Modularising ontologies

The modularisation tools described in this section are available on GitHub at: <https://github.com/sju-best-project/ontology-modules>

The set of guidelines used by BEST to modularise ontologies in an ATM setting are detailed in Deliverable D5.2 [5]. The starting point for the modularisation is the ontologies described in the previous section. This section provides a summary of the main principles related to ontology modularisation in BEST and describes the software developed to support the ontology modularisation. For additional clarification about the modularisation process, we refer to D5.2 [5].

2.2.1 What is the purpose of the tools?

BEST used the tools to modularise the “monolithic” ontology produced from the AIRM’s Logical Data Model. D1.1 describes the distinction of a monolithic ontology and an ontology module as follows:

“Monolithic ontologies are typically characterised as ontologies large in size and complexity, and often spanning several different topics and knowledge areas. Ontology modules on the other hand, aim to provide ontology users with the specific knowledge they require, reducing the scope as much as possible to what is strictly necessary. An ontology consists of a set of axioms, i.e. logical statements, that holds some knowledge. An ontology module encapsulates a subset of the axioms compared to the “monolithic” ontology. For example, if we are interested in only the knowledge related to the concept Aircraft in AIRM, we can represent this knowledge in an Aircraft ontology module, while disregarding other axioms from the AIRM ontology that are not relevant for expressing knowledge about an Aircraft.”

Monolithic ontologies will in many cases include more concepts, properties and instances than needed for a particular use case. Ontology modules on the other hand represent subsets of a monolithic ontology that can be customised to encompass only the entities required for describing a single knowledge domain and/or a particular purpose. For this reason, ontology modularisation is a separate research field within ontology engineering, and a quite active one as well.

Ontology modularisation is a process whereby ontology modules are automatically obtained from monolithic ontologies using a variety of techniques. The rationale for operating with modules instead of their monolithic counterparts can for example be improved performance, usability and maintainability.

There are two main strategies for splitting up a monolithic ontology into ontology modules:

1. **Ontology Partitioning.** This strategy divides the ontology into several equal pieces using quantitative approach. It does not consider the knowledge contents of each resulting module. In BEST we have decided to apply the qualitative approach to base the modularisation on the

contents of the resulting module. This may result in very large modules occasionally, yet they could be further modularised into sub-modules to achieve the desired results.

2. **Ontology Module Extraction.** Module extraction extracts modules from an ontology based on a definition of a sub-vocabulary, also called a seed signature. This signature consists of a set of entities (classes and/or properties and/or individuals) from which the technique recursively traverses through the ontology to gather related entities to be included in the module.

The modularization is useful for the maintaining the ontologies and taking care of the changes. However, there is another useful application for it if we aim to build a specific application. The specific application might not require to handle the entire ontology as built for specific purpose, so it can deal with the specific part of the ontology extracted from the whole ontology into a module. We believe that modularization criteria should be defined in terms of the applications for which the modules are catered. Accordingly, utilizing modularization software can be explained step-by-step as below:

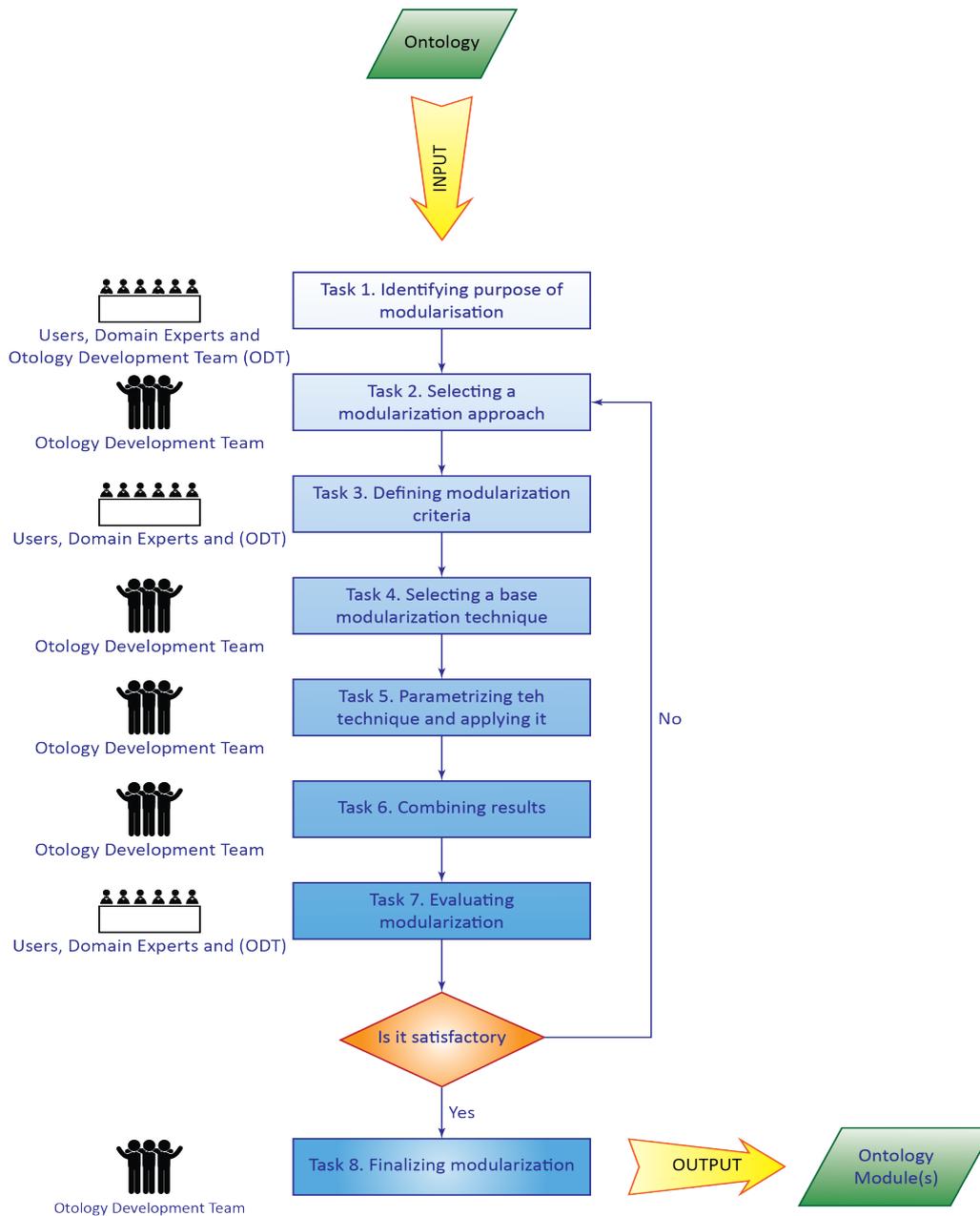


Figure 2. Framework of ontology modularization (re-used from [6])

2.2.2 How do I use the tools?

As part of the investigations in D5.2, the BEST project has developed the following software applications to support the ontology modularisation process:

- **The Module Extractor** extracts a module from the AIRM ontology given a seed signature as parameter. A seed signature can be either a class name or a property [7].

- The **Ontology Module Network Report Generator** checks if there are any classes in the resulting module for which a dependency is not declared. This tool also suggests which ontology modules should be imported to resolve the missing dependencies.
- The **Module Network Dependency Manager** acts on the analyses performed in the previous step and automatically declares the relevant import statements in the ontology module and removes the outlier classes so that there are no duplicate entries in the ontology module.
- The **Redundancy Report Generator** analyses the ontology modules for duplicate classes and presents a list of (potential) duplicates. Resolving the redundancy is a manual operation. The Redundancy Report Generator performs a pairwise ontology matching operation of a set of modules in order to identify duplicate classes.

Apache Maven³ is used for managing dependencies with required java libraries.

All software is developed in java and is made available on GitHub at: <https://github.com/sju-best-project/ontology-modules>

All ontology processing is performed with support of the OWL API [8]. Redundancy Report Generator re-uses functionality provided by the Alignment API [9].

2.2.2.1 Module Extractor

A screenshot of the Module Extractor is shown in Figure 3. The Module Extractor creates a (locality-based) module according to a seed signature from the AIRM ontology. This tool is based on an OWL API implementation of locality-based module extraction developed by the University in Manchester[7]. However, when testing this functionality in the OWL API, we discovered that the resulting modules only contained classes and individuals, all properties were omitted in the extraction. Therefore, we extended the OWL API implementation with functionality that also extracted object properties and data properties for a given ontology module from the AIRM ontology. One consequence of including the object properties is that the resulting module includes outlier classes. This happens because some of the range classes referred to in the object properties belong to other modules extracted from the AIRM ontology.

The command-line user interface prompts the user for three parameters:

- Path to the OWL file representing the ontology from which a module should be extracted from
- The name of the module to be created
- The signature from which axioms represented in the new module should be extracted

³ <https://maven.apache.org>

```

Enter path to (monolithic) ontology file: ./files/modules/v2Sept2017/airm_mono.owl
Enter name of ontology module to create: Meteorology_Module
Enter signature to create module from: #_Meteorology_

Ontology created!
Number of classes: 74
Number of object properties: 69
Number of data properties: 15
Number of individuals: 97
Number of axioms: 746

```

Figure 3. An example on how to use the modularisation tool to extract a Meteorology module from the AIRM ontology

2.2.2.2 Redundancy Report Generator

This tool checks for duplicate classes in modules, after the desired set of modules are extracted by the Module Extractor in the previous step. By pairwise matching of ontology modules using string similarity matching it identifies duplicates and creates a report that lists all duplicates among the ontology modules used as input parameters. Figure 4 shows how to interact with the Redundancy Report Generator and how results are presented.

```

Enter path to folder holding the ontology modules: ./test-files/modules/output-modules
Enter path to folder where the alignments holding duplicate classes will be stored: ./test-files/modules/output-alignments
Running Redundancy Report Generator...

stakeholders and meteorology contain 3 duplicates, and the duplicates are:
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#Aerodrome> - <http://www.project-best.eu/owl/airm-mod/meteorology.owl#Aerodrome>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#Airspace> - <http://www.project-best.eu/owl/airm-mod/meteorology.owl#Airspace>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#ValDistanceType> - <http://www.project-best.eu/owl/airm-mod/meteorology.owl#ValDistanceType>

stakeholders and navigationinfrastructure contain 11 duplicates, and the duplicates are:
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#CodeAuthorityRoleType> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#CodeAuthorityRoleType>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#ValFrequencyType> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#ValFrequencyType>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#RadioNavigationService> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#RadioNavigationService>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#Organisation> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#Organisation>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#Aerodrome> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#Aerodrome>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#CodeRadioEmissionType> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#CodeRadioEmissionType>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#SpecialNavigationSystem> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#SpecialNavigationSystem>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#ValDistanceType> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#ValDistanceType>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#RadioFrequencyArea> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#RadioFrequencyArea>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#DirectionFinder> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#DirectionFinder>
<http://www.project-best.eu/owl/airm-mod/stakeholders.owl#InformationService> - <http://www.project-best.eu/owl/airm-mod/navigationinfrastructure.owl#InformationService>

```

Figure 4. Redundancy Report Generator for identifying duplicate classes in modules

2.2.2.3 Ontology Module Network Report Generator

Figure 5 shows a screenshot of the Ontology Module Network Report Generator. This application analyses an ontology module and reports missing dependencies. Missing dependencies are discovered by searching for an ontology module that has an outlier class in its signature. From this the Ontology Module Network Report tool suggests which ontologies that the module should import and if there are any classes for which there is no relevant module, the names of these classes are presented to the user for further manual analysis of which ontology this class belongs to and consequently which ontology should be imported. Although the imports are taken care of automatically by the Module Network Dependency Manager, the relationships between the modules can bring useful knowledge about the interdependencies of the module network.

```

***Outlier classes (22)***

Flight
CodeEquipmentOperationalType
Unit
AircraftState
AircraftStand
AircraftOperator
AerospaceManufacturer
TaxiwayElement
ValMasType
ValAltitudeType
Trajectory
MatterEmission
SpeedRangeType
TrajectoryPoint
CrewMember
RunwayDirection
CodeFlightPhaseType
CodeValueInterpretationType
ValSpeedType
ValDistanceType
ValPressureType
Organisation

***This ontology module should import***

http://project-best.eu/owl/airm-mod/flight
http://project-best.eu/owl/airm-mod/stakeholders
http://project-best.eu/owl/airm-mod/common
http://project-best.eu/owl/airm-mod/aerodromeinfrastructure
http://project-best.eu/owl/airm-mod/datatypes

***Classes belonging to ontology modules outside our scope***

MatterEmission

```

Figure 5. An example report from the Ontology Module Network Report Generator tool

2.2.2.4 Module Network Dependency Manager

A screenshot showing interaction with the Module Network Dependency Manager is shown in Figure 6. This application identifies relevant ontology modules to import, declares the import statements so that the ontology module actually imports these modules, removes outlier classes, that is, those classes that previously missed a dependent ontology module.

The user is asked to provide a path to the OWL file representing the ontology module to which import declarations should be added. She is also asked to enter the path to the folder holding all ontology modules within the network, so that the Module Network Dependency Manager can process them and see if they contain the classes included in a dependency relation. If so, the necessary import declarations are included in the module and the dependency resolved. Finally, the ontology module is saved to disk with all required import declarations.

```

Enter path to ontology module for which dependencies will be resolved: ../test-files/modules/output-modules/aircraft.owl
Enter path to folder where the modules in the network resides: ../test-files/modules/output-modules

Declaring import for documentIRI:../test-files/modules/output-modules/aircraft.owl and ontologyIRI http://www.project-best.eu/owl/airm-mod/stakeholders.owl

Declaring import for documentIRI:../test-files/modules/output-modules/aircraft.owl and ontologyIRI http://www.project-best.eu/owl/airm-mod/aerodromeinfrastructure.owl

Declaring import for documentIRI:../test-files/modules/output-modules/aircraft.owl and ontologyIRI http://www.project-best.eu/owl/airm-mod/common.owl
Ontology module saved with imports!

*** Classes outside of the defined ontology network:
SpeedRangeType
TrajectoryPoint
MatterEmission
Flight
ValPressureType
ValMassType
ValDistanceType
Trajectory
AircraftState
ValSpeedType
CodeFlightPhaseType
ValAltitudeType

```

Figure 6. Module Network Dependency Manager

2.3 Validating compliance and ontology matching

The source code of the AIRM Compliance Validator can be downloaded from GitHub: <https://github.com/sju-best-project/compliancevalidator>. More detailed information about the development and evaluation of the AIRM Compliance Validator proof-of-concept application can be found in deliverable D1.2 AIRM Compliance Validator [10]. This chapter describes the main principles, details related to the application development and how to interact with the AIRM Compliance Validator.

2.3.1 What is the purpose of the tool?

The AIRM Compliance Validator is a proof-of-concept application that automatically identifies semantic correspondences between concepts of two input ontologies. The application supports two use cases:

1. **Semantic interoperability in the development of new ATM information models and services.** By suggesting semantic correspondences between models under development and the AIRM this encourages re-use of standardised information elements rather than the development of new ones.
2. **Compliance Assessment.** Once information models are developed they undergo a process to assure that they are compliant with the AIRM. The AIRM Compliance Validator supports the compliance assessment process as it through an automated process suggests semantic correspondence between elements in the information models under assessment and the AIRM.

The application is developed using principles from ontology matching research, and it includes the following main components:

- A set of metrics used for profiling ontologies to be matched;
- A set of matching algorithms that produce an alignment as a set of semantic correspondences;
- Strategies that combine the alignments in an optimal manner.

2.3.2 How do I use the tool?

The development re-uses and extends source code from the following java libraries:

- OWL API. We use the OWL API [8] for parsing the input ontologies and for retrieving statistics about the ontology constructs used in the ontology profiling.
- Alignment API. The Alignment API [9] provides interfaces and methods for supporting implementation of matching algorithms and generating alignment files according to the alignment format.
- OntoSim. OntoSim⁴ provides a library of various types of similarity algorithms. From this library we use the ISub string matching algorithm, originally developed by Stoilos [11].
- JWNL. JWNL⁵ is a java library for interacting with the WordNet [12] database. WordNet is a synonym lexicon that defines a number of different semantic relations between concepts.
- Apache POI. Apache POI⁶ is an API that is used for manipulating Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). More specifically we use Apache POI for transforming AIRM compliance mapping artefacts in Excel to reference alignments used to evaluate the quality of alignments produced by the Alignment API.
- Neo4J. Neo4J⁷ is a graph database that is used by the structural matcher Closest Parent Matcher, and has to be installed and run in order to execute the Closest Parent Matcher.

Apache Maven⁸ is used for managing dependencies with required java libraries.

The source code of the AIRM Compliance Validator can be downloaded from GitHub: <https://github.com/sju-best-project/compliancevalidator>

For interacting with the AIRM Compliance Validator we have developed a simple command-line user interface that is described in the following:

The first activity of the matching workflow is to analyse the ontologies to be matched. We have developed a component called **Ontology Profiler** that analyses the terminological, structural and lexical characteristics of the input ontologies. Figure 7 shows the user interface for the Ontology Profiler. The only input required is the two ontology files to be analysed. We refer to D1.2 AIRM Compliance Validator [10] report for an explanation of what the different metrics mean and how their results should be interpreted.

⁴ <http://ontosim.gforge.inria.fr/>

⁵ <https://sourceforge.net/projects/jwordnet/>

⁶ <https://poi.apache.org/>

⁷ <https://neo4j.com/>

⁸ <https://maven.apache.org>

```

Starting the Ontology Profiler

Enter path to ontology 1: ./files/ontologies/airm/airportheliport.owl
Enter path to ontology 2: ./files/ontologies/airm/aerodromeinfrastructure.owl

Ontology Profiling Results:

*** Number of Compounds ***
The Num Compounds (NC) of airm_airportheliport.owl is: 0.9081632653061225
The Num Compounds (NC) of aerodromeinfrastructure.owl is: 0.9383259911894273
The Num Compounds (NC) of airm_airportheliport.owl and aerodromeinfrastructure.owl is: 0.9232446282477749 (92.32 percent)

*** Annotation Coverage ***
The Annotation Coverage of airm_airportheliport.owl is: 99.97448979591837
The Annotation Coverage of aerodromeinfrastructure.owl is: 99.51541850220265
The Annotation Coverage of airm_airportheliport.owl and aerodromeinfrastructure.owl is: 99.72813238770685 (99.73 percent)

*** Inheritance Richness (Real number) ***
The Inheritance Richness (IR) of airm_airportheliport.owl is: 1.9846938775510203
The Inheritance Richness (IR) of aerodromeinfrastructure.owl is: 1.5198237885462555
The Inheritance Richness (IR) of airm_airportheliport.owl and aerodromeinfrastructure.owl is: 1.7352245862884161

*** Relationship Richness ***
The Relationship Richness (RR) of airm_airportheliport.owl is: 0.4450784593437946
The Relationship Richness (RR) of aerodromeinfrastructure.owl is: 0.5
The Relationship Richness (RR) of airm_airportheliport.owl and aerodromeinfrastructure.owl is: 0.4723220704529116 (47.23 percent)

*** WordNet Coverage ***
The WordNet Coverage (WC) of airm_airportheliport.owl is: 0.7551020408163265
The WordNet Coverage (WC) of aerodromeinfrastructure.owl is: 0.6740088105726872
The WordNet Coverage (WC) of airm_airportheliport.owl and aerodromeinfrastructure.owl is: 0.7145554256945068 (71.46 percent)

*** WordNet Synonym Coverage (Real number) ***
The WordNet Synonym Coverage of airm_airportheliport.owl is 0.5663265306122449
The WordNet Synonym Coverage of aerodromeinfrastructure.owl is 0.6387665198237885
The WordNet Synonym Coverage of airm_airportheliport.owl and aerodromeinfrastructure.owl is 0.6025465252180167

```

Figure 7. Ontology Profiler

The results from the Ontology Profiler gives an indication of the performance to be expected from different matching algorithms. Once the set of matchers to be included in the matching process is determined, the core matching process of the AIRM Compliance Validator can be executed. In the following we include screenshots and explanations for each step of this process.

2.3.2.1 Import ontologies

The entire process starts by importing the two ontologies from which semantic correspondences will be identified. The parsers implemented in the AIRM Compliance Validator will only accept OWL ontologies, and the ontologies have to reside locally on disk, not online. Next, the user is asked to provide a path to a folder where the alignment holding all semantic correspondences will be stored.

```

Starting the AIRM Compliance Validator

Enter path to ontology 1: ./files/ontologies/airm/airportheliport.owl
Enter path to ontology 2: ./files/ontologies/airm/aerodromeinfrastructure.owl
Enter path to folder where alignment file will be stored: ./files/alignments/

```

Figure 8. Import of ontologies to be matched

2.3.2.2 Match ontologies

Once the ontologies are imported and parsed, the matching of the two imported ontologies is performed with some initial configuration from the user. This includes selecting the desired type of

semantic correspondence (equivalence or other semantic correspondence types), selecting matcher(s), and configuring the confidence measure.

2.3.2.3 Select matching strategy

Once the ontologies are imported, the user is asked to select whether the AIRM Compliance Validator should identify equivalence relations or other semantic relations. Afterwards, the user is presented with a list of available matchers and combination strategies (matcher configuration). The sub-menu shown presenting the available matchers depends on whether the user has selected equivalence relations (Figure 9) or other semantic relations Figure 10).

```
Select Equivalence (1) or Other Semantic Relations (2): 1
EQ1: ISub
EQ2: Definitions Matcher
EQ3: Property Matcher
EQ4: Range Matcher
EQ5: WordNet Synonym Matcher
COM1: Weighted Sequential Combination
COM2: Simple Vote Combination
COM3: Autoweight++ Combination
```

Figure 9. Select Matching Strategy - Equivalence

```
Select Equivalence (1) or Other Semantic Relations (2): 2
SUB1: Closest Parent Matcher
SUB2: Compound Matcher
SUB3: Definitions (Subsumption) Matcher
COM1: Weighted Sequential Combination
COM2: Simple Vote Combination
COM3: Autoweight++ Combination
```

Figure 10. Select Matching Strategy - Other Semantic Correspondences

Note that in order to run the Closest Parent Matcher an instance of the Neo4J database has to be installed and running. When Neo4J is running a database to hold the graph representation of the ontologies to be matched is created automatically.

2.3.2.4 Matcher configuration

If the user has selected a combination strategy from the sub-menus, he/she is asked to provide a path to the folder holding the alignments to be combined, see Figure 11.

```

EQ1: ISub
EQ2: Definitions Matcher
EQ3: Property Matcher
EQ4: Range Matcher
EQ5: WordNet Synonym Matcher
COM1: Weighted Sequential Combination
COM2: Simple Vote Combination
COM3: Autoweight++ Combination

Select matcher configuration: COM1

Enter path to folder holding alignments to be combined: ./files/testGUI

Weighted Sequential Combination completed!

```

Figure 11. Selection of folder holding alignments to be combined

If the user has selected an individual matcher, he/she is asked to configure which confidence threshold to be applied for the matcher, see Figure 12.

```

Select matcher configuration: SUB1

Select confidence threshold (decimal value between 0.0 and 1.0): 0.6
Matching ./files/ontologies/airm/airm_airportheairport.owl and ./files/ontologies/airm/aerodromeinfrastructure.owl

Closest Parent matcher completed!

```

Figure 12. Configuring the selected matcher

Once the configuration of confidence threshold is done, the matching is executed.

2.3.2.5 Report identified semantic correspondences

The identified semantic correspondences are presented in an RDF-XML file according to the Alignment Format⁹. Figure 13 shows the output from an equivalence matching operation using the XML editor OxygenXML¹⁰. Each equivalence correspondence is represented in a *map* element, and each map element contains one *cell* element. Within each cell element the two concepts forming the semantic correspondence is represented as *entity1* and *entity2*. The type of semantic correspondence between the two concepts is expressed in the *relation* element. For equivalence correspondences the relation is '=', while specialisation (restriction) which is shown in Figure 14 is specified as < (less than). Generalisation would be specified as > (or greater than).

⁹ <http://alignapi.gforge.inria.fr/format.html>

¹⁰ <https://www.oxygenxml.com/>

```

    ▼ ● map
      ▼ ● Cell
        ● entity1 "http://www.project-best.eu/owl/aixm-mod/airport#AirportHeliport"
        ● entity2 "http://www.project-best.eu/owl/aixm-mod/aerodromeinfrastructure.owl#Aerodrome"
        ● relation =
        ● measure "http://www.w3.org/2001/XMLSchema#float" 0.9892215568862275
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://www.project-best.eu/owl/aixm-mod/airport#GuidanceLineMarking"
          ● entity2 "http://www.project-best.eu/owl/aixm-mod/aerodromeinfrastructure.owl#GuidanceLineMarking"
          ● relation =
          ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://www.project-best.eu/owl/aixm-mod/airport#ApronMarking"
          ● entity2 "http://www.project-best.eu/owl/aixm-mod/aerodromeinfrastructure.owl#TaxiwayMarking"
          ● relation =
          ● measure "http://www.w3.org/2001/XMLSchema#float" 0.9587924016282225
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://www.project-best.eu/owl/aixm-mod/airport#DeicingAreaMarking"
          ● entity2 "http://www.project-best.eu/owl/aixm-mod/aerodromeinfrastructure.owl#DeicingAreaMarking"
          ● relation =
          ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://www.project-best.eu/owl/aixm-mod/airport#GuidanceLine"
          ● entity2 "http://www.project-best.eu/owl/aixm-mod/aerodromeinfrastructure.owl#GuidanceLine"
          ● relation =
          ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
  
```

Figure 13. Semantic Correspondences in Alignment Format - Equivalence

```

    ▼ ● map
      ▼ ● Cell
        ● entity1 "http://project-best.eu/owl/iwxxm-mod/metar#AerodromeSurfaceWind"
        ● entity2 "http://www.project-best.eu/owl/aixm-mono/airm.owl#Wind"
        ● relation &lt;
        ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://project-best.eu/owl/iwxxm-mod/metar#RunwayContamination"
          ● entity2 "http://www.project-best.eu/owl/aixm-mono/airm.owl#Runway"
          ● relation &lt;
          ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
      ▼ ● map
        ▼ ● Cell
          ● entity1 "http://project-best.eu/owl/iwxxm-mod/metar#AerodromeRunwayVisualRange"
          ● entity2 "http://www.project-best.eu/owl/aixm-mono/airm.owl#RunwayVisualRange"
          ● relation &lt;
          ● measure "http://www.w3.org/2001/XMLSchema#float" 1.0
  
```

Figure 14. Semantic Correspondences in Alignment Format - Other Correspondences

3 Results Group 2: Concepts and Prototypes developed in BEST for illustrating the usefulness and feasibility

3.1 Semantic Container Concept

The semantic container approach is detailed in Deliverable D2.1. We investigate the benefits of using semantic web technologies for realizing the upcoming System Wide Information Management (SWIM) concept for Air Traffic Management (ATM). We identify kinds of ATM information and metadata that existing semantic web technologies can handle effectively, and we propose a semantic container approach for handling ATM information within SWIM. What is the purpose of the concept?

The semantic container approach, as developed in the course of the BEST project, complements the European SWIM service (instance) definitions with a means for the description of the information that a service instance uses and provides. A faceted approach using existing semantic technologies and ontology modules, developed in the same research project, allows for flexible information description. 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 contained 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 labeled 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.

Figure 15 illustrates possible derivation chains of semantic containers. Filtering, enrichment, combination, and composition of information leads to the computation of several derivation states. In order to keep the figure as readable as possible, it does not show component containers, physical containers, i.e., allocations of logical containers, versions and administrative metadata. The figure provides a high-level overview of how semantic containers can be generated from other containers.

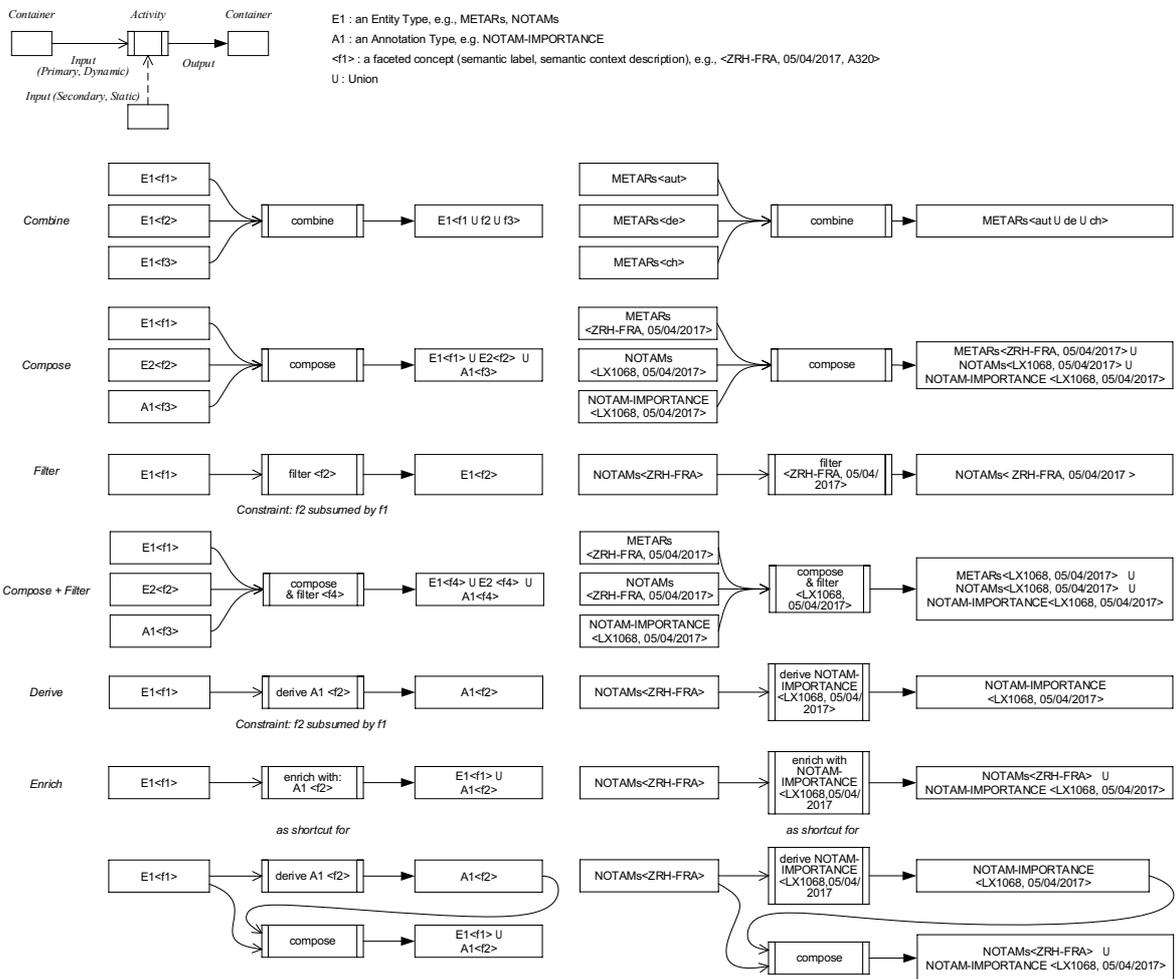


Figure 15. Possible semantic container derivation operations along with concrete examples

Possible container derivations are “combine”, “compose”, “filter”, “derive” and “enrich”. The combine operation produces a homogeneous composite container from a set of input containers with the same entity/annotation type. For example, sets of METAR containers from different countries as input yield a single composite container as output. The compose operation produces a heterogeneous composite container from a set of input containers with different entity/annotation types. For example, a METAR container, a NOTAM container, and a container with NOTAM importance as input yield a heterogeneous composite container as output. The filter operation takes a container as input and produces a container with a reduced set of data items. For example, a NOTAM container with NOTAMs for a specific route as input is filtered with respect to a specific flight on a specific date. The derive operation takes a container as input and produces a container with some annotation type as output. For example, a NOTAM container as input yields a container with a set of NOTAM importance annotations for a specific flight as output. The enrich operation is a combination of derive and compose. For example, from a container of NOTAMs relevant for a specific flight route, first all the NOTAM importance annotations relevant for a specific flight are derived, which then constitute

together with the original container the components of a heterogeneous composite container. In the following, we provide examples of derivation chains

3.2 Semantic Container Management System (Prototype)

The Semantic Container Management System (SMCS) is detailed in D3.2 Prototype SWIM-enabled applications [12] chapter 3. In this section, we describe the management of Semantic Containers in a distributed SWIM environment. We discuss the metamodel for Semantic Container Management System in UML that considers both logical and physical aspects of semantic containers which was the baseline for the prototype development. We follow different views on semantic containers, namely distribution and replication, lineage and provenance, as well as versioning and consistency management.

3.2.1 How does the Distribution and Replication Work?

A semantic container is primarily a logical unit of data items with a semantic label that states a membership condition for data items, which represents a commitment by the creator of the container: The semantic container comprises all data items that satisfy the membership condition. The metamodel of the faceted membership condition is part of the semantic label. Facets are dimensions of semantic description, and can be classified into spatial, temporal, and other semantic facets.

For example, a spatial facet may describe the geographic focus of the DNOTAMs in a semantic container; a temporal facet the time of validity of the DNOTAMs, another semantic facet may refer to the type of aircraft for which the contained DNOTAMs are relevant. The facet values that a semantic label assigns for each facet come from an ontology. For example, a DNOTAM container may contain DNOTAMs relevant for fixed-wing aircraft, with fixed-wing aircraft being represented by a concept in an ontology derived from the ATM Information Reference Model (AIRM). A single facet may be defined by multiple ontologies, and the same ontology may serve to define the same facet.

Now a semantic container is also a physical package (see Figure 16) of data items, meaning that each logical semantic container also has an allocation at a specific physical server location as well as copies at multiple other locations. For example, a semantic container with all DNOTAMs for a flight from Munich to Frankfurt may be allocated on servers at Munich airport and Frankfurt airport, or on an aircraft that conducts a flight from Munich to Frankfurt. For the proof-of-concept of the distributed architecture we implemented two different locations during the prototyping (see Figure 17).

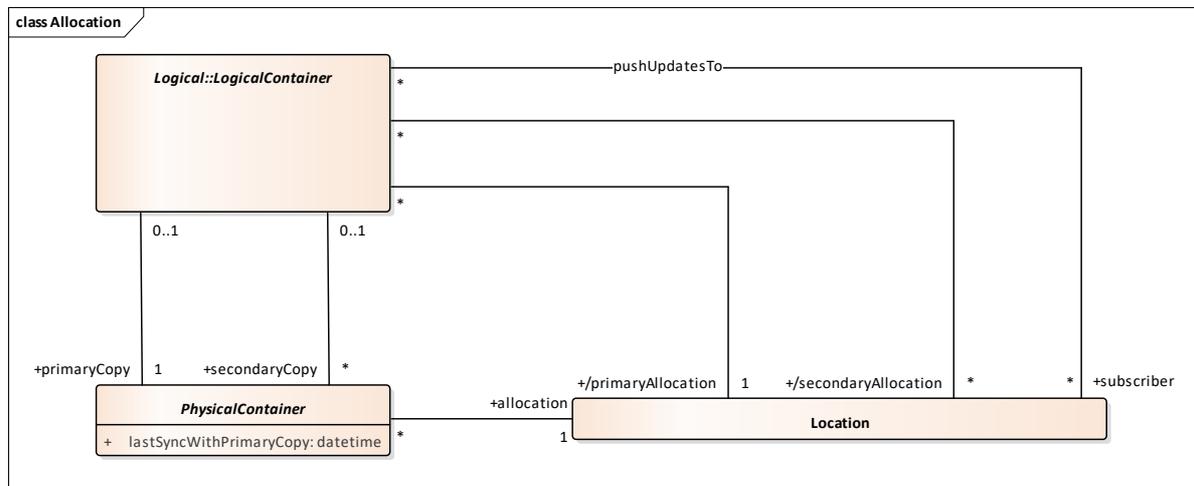


Figure 16. Allocation of physical semantic containers

We opt for a distribution and replication concept where each logical semantic container has one primary copy stored at some location, and potentially multiple replicas stored at other locations. We note, however, that also other distribution and replication concepts may be considered, including decentralized solutions and reference-only containers. The former refers to a solution where no copy of a logical container is a designated primary. The latter refers to containers that have no physical materialization but are only logical concepts materialized upon request.

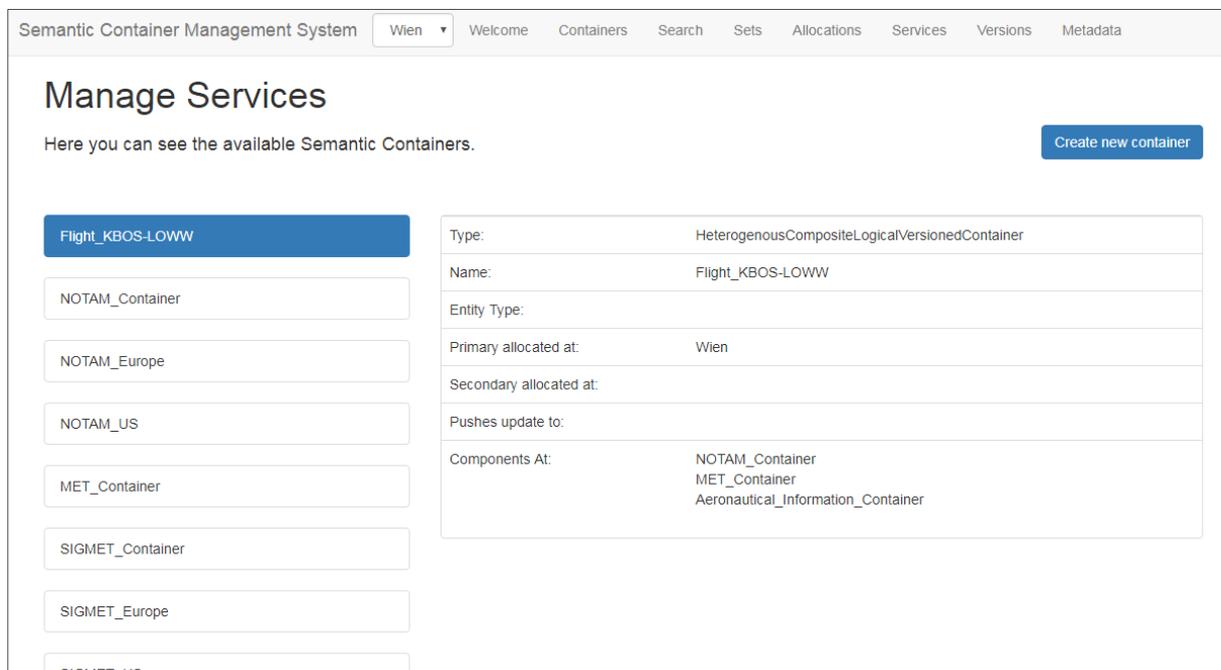
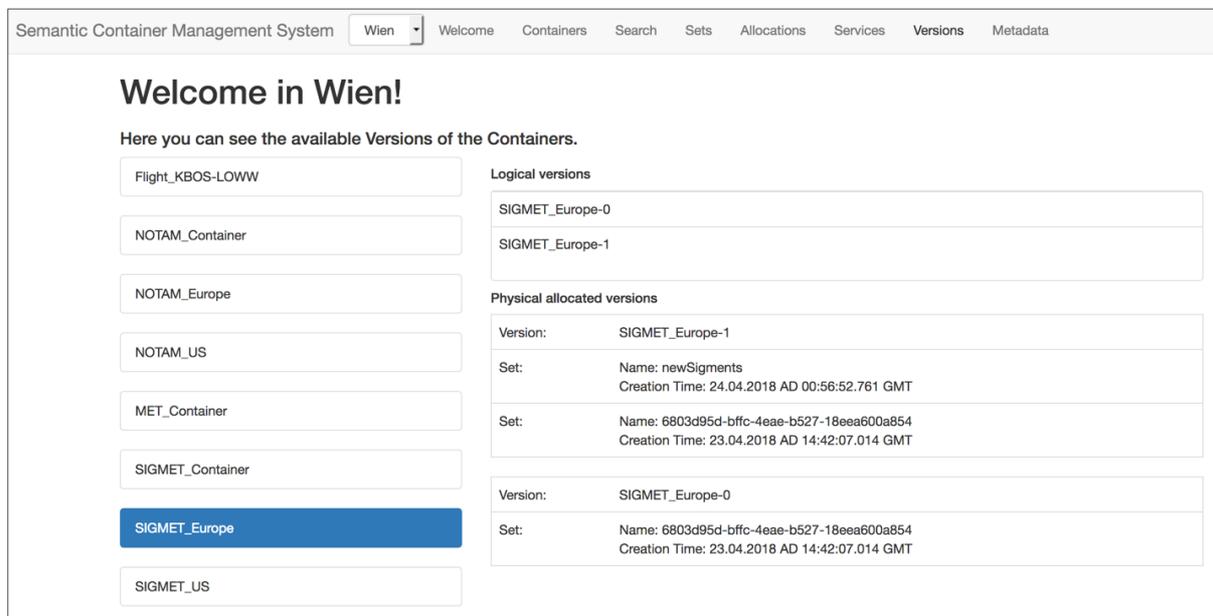


Figure 17: Semantic Container Management Platform Prototype showing two different locations.

becomes available again, the degenerated datasets can be replaced by the regular sets in the current version, but are kept for audit purposes.



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.

Container Name	Logical versions	Physical allocated versions
Flight_KBOS-LOWW	SIGMET_Europe-0	Version: SIGMET_Europe-1 Set: Name: newSigments Creation Time: 24.04.2018 AD 00:56:52.761 GMT
NOTAM_Container	SIGMET_Europe-1	Set: Name: 6803d95d-bffc-4eae-b527-18eea600a854 Creation Time: 23.04.2018 AD 14:42:07.014 GMT
NOTAM_Europe		Version: SIGMET_Europe-0 Set: Name: 6803d95d-bffc-4eae-b527-18eea600a854 Creation Time: 23.04.2018 AD 14:42:07.014 GMT
NOTAM_US		
MET_Container		
SIGMET_Container		
SIGMET_Europe		
SIGMET_US		

Figure 19: Versioning in the Semantic Container Management System

A versioned composite physical container consists of multiple component physical containers. The datasets of a composite container derive from the datasets of its components. Note that all components of a composite container are allocated on the same location together. In the following, we define how logical and physical containers are derived by services from other containers, formalizing the principles of derivation chains from the previous sections.

3.2.3 How is Lineage and Provenance represented?

The provenance of a semantic container in a derivation chain is represented in the container management system. A logical container may have multiple primary sources as well as alternative secondary sources. The primary sources are the sources of prime quality. Secondary sources usually offer degraded quality.

A container derives from a primary or secondary source through a service call. A service call has a semantic label as arguments and possibly many static containers as additional input, corresponding the concepts and static containers in the examples from the previous sections. A service call can have multiple occurrences. The call occurrences are what actually produce a dataset (see Figure 20).

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

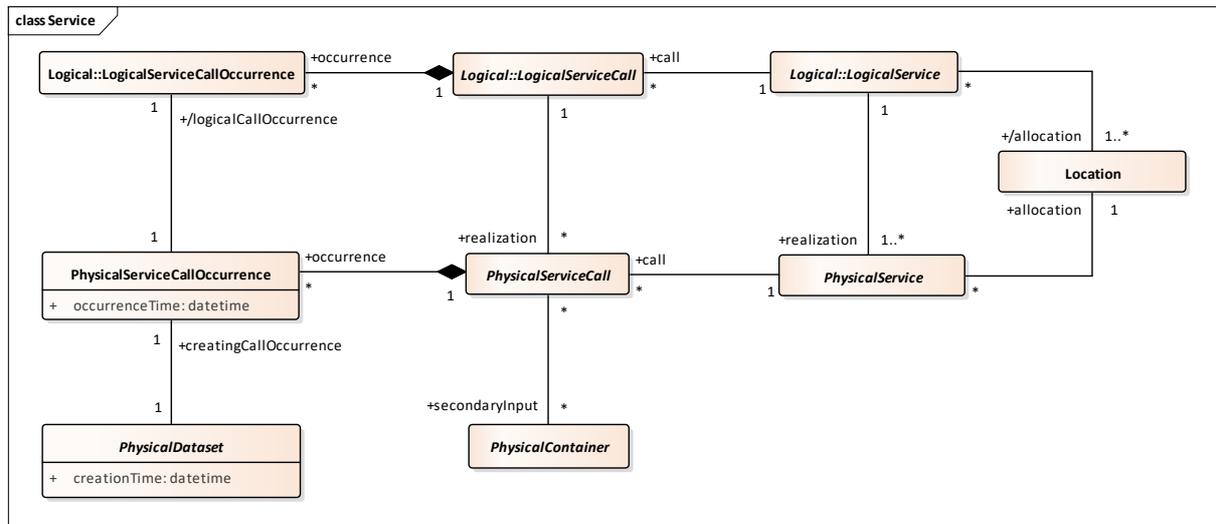


Figure 20. Class diagram “Service” in the physical view: Physical realization of container provenance

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

Figure 21: Creation of a Homogenous Composite Container in the SCMS

3.2.4 How does the System Architecture look like?

The system architecture consists of a central logical container repository and a number of decentralized physical repositories that manage the actually allocated data. Figure 22 illustrates the architecture of the container registry, with example locations. Note that each geographic location could have multiple server locations. An aircraft could also be equipped with a local container management system in order to physically allocate relevant data on the aircraft directly. During the prototype development we implemented two locations (Linz and Vienna) to proof the concept (see Figure 24).

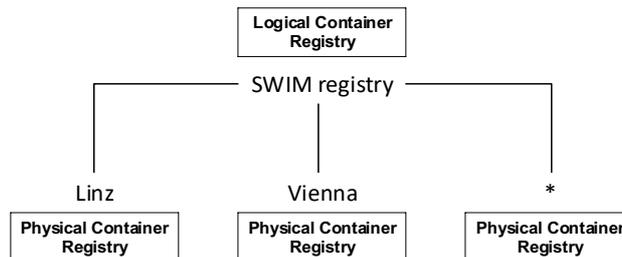


Figure 22. Central logical container registry and multiple physical container registries at different server locations

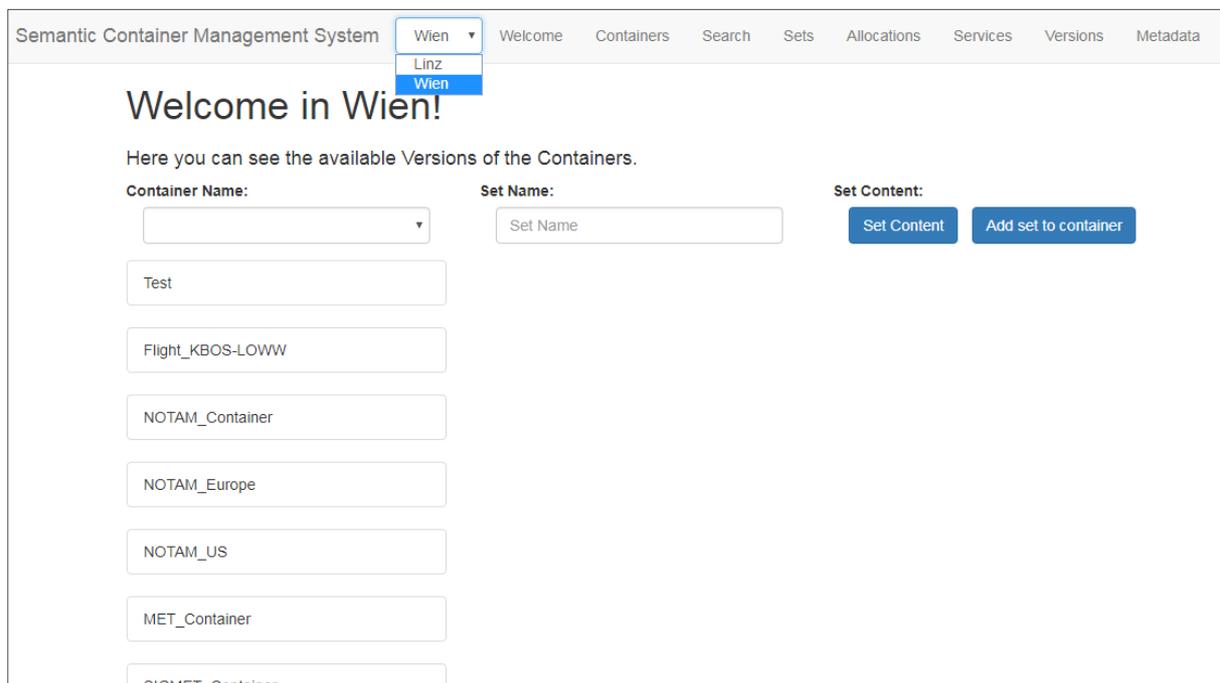


Figure 23: Multiple locations for container storage

3.3 Integration of Semantic Containers in a SWIM Environment

A detailed view of the scenario used to illustrate the integration of semantic containers in the SWIM Environment can be found in D3.2 Prototype SWIM-enabled Applications [12] chapter 5. Figure 24 gives an overview about how semantic containers can be integrated into SWIM. The SWIM Registry (see 1) was used 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 (see 2) is used to define and create containers that are then visible through the SWIM registry. On organizational level the SWIM Integration platform (Frequentis' MosaiX) is used (see 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 (see 4) from SESAR WP13.2.2.

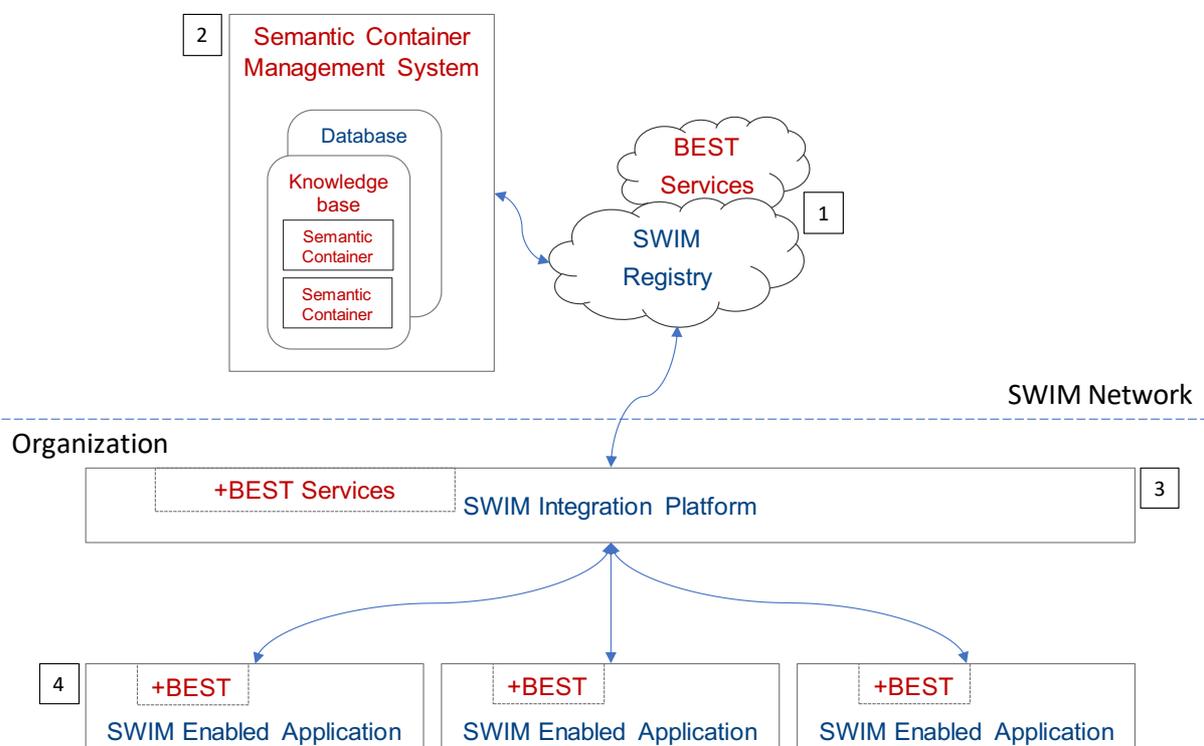


Figure 24: Integration of Semantic Containers into SWIM [12]

References

- [1] A. Vennesland, B. Neumayr, C. Schuetz, and A. Savulov, “D1.1 Experimental ontology modules formalising concept definition of ATM data,” 2017.
- [2] S. Wilson, “EUROCONTROL Specification for SWIM Information Definition version 1.0,” Brussels, Belgium, 2017.
- [3] S. Wilson, S. Keller, G. Marrazzo, and R. Suzic, “AIRM Compliance Framework,” 2015.
- [4] Object Management Group, “Ontology Definition Metamodel (ODM) v1.1,” Needham, OSA, 2014.
- [5] A. Vennesland, E. Gringinger, and A. Kocsis, “D5.2 Ontology Modularisation Guidelines,” 2018.
- [6] M. D’Aquin, “Modularizing Ontologies,” in *Ontology Engineering in a Networked World*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 213–233.
- [7] C. Del Vescovo, R. Gonçalves, B. Parsia, and U. Sattler, “OWL @ Manchester: Modularity,” 2018. [Online]. Available: <http://owl.cs.manchester.ac.uk/research/modularity/>. [Accessed: 19-Apr-2018].
- [8] M. Horridge and S. Bechhofer, “The OWL API: A Java API for OWL Ontologies,” *Semant. Web J.*, vol. 2, no. 1, pp. 11–21, 2011.
- [9] J. David, J. Euzenat, F. Scharffe, and C. T. Dos Santos, “The Alignment API 4.0,” *Semant. Web*, vol. 2, no. 1, pp. 3–10, 2010.
- [10] A. Vennesland, B. Neumayr, C. Schuetz, and E. Gringinger, “D1.2 AIRM Compliance Validator,” 2018.
- [11] S. Stoilos, Giorgos and Stamou, Giorgos and Kollias, “A string metric for ontology alignment,” in *Proceeding of the International Semantic Web Conference 2005*, 2005, pp. 624–637.
- [12] C. Fellbaum, *WordNet: An Electronical Lexical Database*. Cambridge: MIT Press, 1998.

The BEST consortium:

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