Providing Packages of Relevant ATM Information: An Ontology-based Approach

Bernd Neumayr[a,*], Christoph G. Schuetz[a], Eduard Gringinger[b,c], Christoph Fabianek[b,c], Audun Vennesland[d], Michael Schrefl[a], Scott Wilson[e]

[a] *Institute of Business Informatics – Data and Knowledge Engineering, Johannes Kepler University Linz, Altenberger Strasse 69, 4040 Linz, Austria*
[b] *Frequentis AG, Vienna, Austria*
[c] *OwnYourData, Bad Vöslau, Austria*
[d] *SINTEF, Trondheim, Norway*
[e] *EUROCONTROL, Brussels, Belgium*

Abstract

ATM information providers publish reports and notifications of different types using standardized information exchange models. For a typical information user, e.g., an aircraft pilot, only a fraction of the published information is relevant for a particular task. Filtering out irrelevant information from different information sources is in itself a challenging task, yet it is only a first step in providing relevant information, the challenges concerning maintenance, auditability, availability, integration, comprehensibility, and traceability. This paper presents the Semantic Container approach, which employs ontology-based faceted information filtering and allows for the packaging of filtered information and associated metadata in semantic containers, thus facilitating *reuse* of filtered information at different levels. The paper formally defines an abstract model of ontology-based information filtering and the structure of semantic containers, their composition, versioning, discovery, and replicated physical allocation. The paper further discusses different usage scenarios, the role of semantic containers in SWIM, an architecture for a semantic container management system, as well as a proof-of-concept prototype. Finally the paper discusses a blockchain-based notary service to realize tamper-proof version histories for semantic containers.

Keywords: Data Mediation, Information Tailoring, System Wide Information Management, Air Traffic Management

1. Introduction

Providers of air traffic management (ATM) information increasingly employ standardized information exchange models for the publication of ATM information. The Aeronautical Information Exchange Model (AIXM) [1], the Flight Information Exchange Model (FIXM) [2], and the ICAO Weather Information Exchange Model (IWXXM) [3] are the most common examples of exchange models in ATM. Different information providers publish different types of information items, e.g., Digital NOTAMs, flight plans, or weather reports, on the SWIM network, which relies on the standardized exchange models. The SWIM network provides information consumers with a uniform technical basis for accessing the various types of ATM information published by a multitude of information providers.

A consumer of ATM information, e.g., an aircraft pilot or an air traffic controller, typically requires only a small fraction of the available information in order to prepare for a task. Providing ATM information packages, i.e., collections of relevant ATM information, encompasses retrieving all the required information by making calls to various services offered by different information providers. In order to reduce the information load, filter conditions may be sent along with the requests for information directly to the service providers. Alternatively, dedicated filtering services or end-user applications may assume the filtering task. The complexity of the filtering task, in turn, may vary, from rather simple to more complex; Digital NOTAM (DNOTAM) filtering [4] is a prime example of a complex and computationally expensive filtering task.

The overall goal of the research presented in this paper is to facilitate the provisioning and management of ATM information packages. The challenges we tackle are the following. (i) Dealing with many different service interfaces from different information providers without standardized means to specify filter criteria makes the collection of relevant information cumbersome. (ii) Information filtering is often computationally expensive and conducting the information filtering from scratch for every information need is highly inefficient. (iii) New information items must be integrated over time to keep ATM information packages up-to-date. For later auditability it is necessary to keep track of every update and its provenance. (iv) Network or service outages may hamper availability.

Solving these challenges is necessary for the scalable provision of ATM information packages which we believe

*Corresponding author

will play a central and ubiquitous role in the future SWIM network where ATM information will be provided and filtered at large-scale.

The *overall contribution* presented in this paper is a platform-independent, ontology-based approach for the provision and management of ATM information packages – the Semantic Container approach developed in project BEST [6]. A semantic container, the central construct of the approach, represents an information need or information package, and is described by type and origin of information together with filter criteria.

The Semantic Container approach tackles the aforementioned challenges as follows (using the same numbering as above): (i) Information needs and information packages are uniformly represented by semantic containers with filter criteria defined in shared ontologies to ensure a common understanding between ATM stakeholders and allowing to abstract away from particular service calls. (ii) Pre-filtered information packages can be re-used for the derivation of task-specific information packages. The description of semantic containers facilitates automated discovery of pre-filtered information packages as well as the dynamic addition and removal from a derivation chain of semantic containers. (iii) A semantic container maintains a version history to verify which information was available to the user at a particular point of time and to track the provenance of each update. (iv) Replication of information packages increases availability in case of network or service outages.

This paper is a revised and extended version of a conference paper [8]. The main additional contributions of this paper are *formal definitions* of an abstract model of ATM information, of ontology-aware information filtering, and of the main constructs of the Semantic Container approach. The formal definitions precisely describe the Semantic Container approach and the basics upon which it builds, avoiding the ambiguities of the previously mainly textual presentation of the approach. Additionally we describe a notary service that provides trusted timestamps, thus enabling trusted version histories of semantic containers, allowing, especially in case of incidents, to verify *without doubt* the contents of a semantic container at a particular location at a certain point of time in the past.

The remainder of the paper is structured as follows. Sect. 2 explains the development of OWL ontologies from widely-used, standardized ATM information exchange models which serve as basis for ontology-based ATM information filtering. Sect. 3 introduces an abstract and formal model of ATM information and of ontology-based ATM information filtering which underly the Semantic Container approach. Sect. 4 introduces a formal model of the Semantic Container approach, describing the core constructs of the approach in a precise and implementation-independent manner. Sect. 5 discusses the application potentials of the approach in SWIM and highlights the necessity for a Semantic Container Management System as a concrete realization of the approach. Sect. 6 discusses an architecture of such a system and Sect. 7 presents a proof-of-concept prototype of a basic semantic container management system and its integration with a system for integrated digital briefing in SWIM. Sect. 8 presents different usage scenarios for semantic containers. Sect. 9 discusses how blockchain technologies can provide trusted timestamps for trusted version histories of semantic containers. Sect. 10 gives an overview of related work and Sect. 11 concludes with an outlook on future work. A table of acronyms (Table 3) is given at the end of the paper.

2. ATM Information Ontologies

The ontologies used in the Semantic Container approach are based on OWL ontologies and ontology modules derived from ATM information exchange models (IWXXM and AIXM) and from the ATM Information Reference Model (AIRM) available as UML class diagrams. In this section we describe the transformation from UML to OWL, automatic extraction of modules, and the development of more fine-grained ontologies by specializing the concepts from the generated ontologies. We conclude the section with a formalization of the aspects of ontologies which are essential to the Semantic Container approach.

The ontology infrastructure includes ontologies developed from the AIRM UML model and a set of ontologies, each representing different domains of ATM information exchange, namely AIXM [1] and IWXXM [3]. All ontologies are formalized in the Web Ontology Language (OWL) as standardized by the World Wide Web Consortium. The ontologies are used as the vocabulary for describing and supporting retrieval of relevant information by applications developed in the project. Furthermore, the ontologies form a baseline for the establishment of guidelines describing how semantic technologies can be applied to support information exchange in a SWIM environment.

The ontology development basically included three sub-processes:

1. Transformation from UML to OWL
2. Automated extraction of modules
3. Development of fine-grained ontologies by extending and refining the generated ontologies and ontology modules

2.1. Transformation from UML to OWL

The first step in transforming UML to OWL was to generate an XML Metadata Interchange (XMI) representation of the UML models. In the next step, we applied a set of XSLT transformation rules to transform from XMI to OWL. The transformation rules were developed with support from the (non-normative) guidelines for mapping between UML and OWL in the OMG ODM specification (see Appendix A of [9]). Table 1 provides an overview of the transformations performed; a more detailed explanation of each transformation and its resulting OWL entity

Table 1: Overview of transformation between UML and OWL

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

is provided in [9]. The XSLT scripts used in the transformation are available online[1].

For the AIRM, the transformation from UML to OWL was fully automated using the XSLT transformation rules described above. However, the IWXXM and AIXM exchange models include many package interdependencies, intricate data typing, and other modelling conventions (e.g. XOR relationships and association classes) that made a fully automated transformation from UML (via XMI) to OWL challenging. Therefore, for these two models the transformation followed a semi-automatic ontology development consisting of two consecutive processes. In the first process, OWL classes and properties were generated automatically using approximately the same XSLT transformation that was used for the AIRM. In the second process, manual engineering using the ontology editor Protégé [11] was applied to create and verify proper relationships between classes, object properties, data properties, and individuals, before an OWL representation of the exchange models was complete and the module extraction described in the next section could take place.

2.2. Automated Extraction of Modules

Monolithic ontologies can be characterized as large-sized and complex, often spanning several different topics and knowledge domains. Developing and maintaining such monolithic ontologies is a cumbersome and sometimes overwhelming task due to their size and complexity [13]. Advantages of ontology modules on the other hand include that they promote use, re-use, more efficient processing, and simple maintenance (to name a few).

The task of automatically decomposing a monolithic ontology into a set of sub parts (modules) is called ontology modularization. There is no single approach to ontology modularization that works for all situations, it depends on the application requirements. There are however two overall strategies, namely 1) ontology partitioning and 2) ontology module extraction. Ontology partitioning consists of decomposing the full set of axioms in an ontology into a set of modules (partitions) and the union of all modules should in principle be equivalent to the original ontology.

[1] `https://w3id.org/airm-o/ontology/`

Table 2: Modules generated from AIRM

Ontology Module	Classes	Object prop.	Data prop.	Individuals
Aircraft	71	84	32	182
AerodromeInfrastructure	117	345	69	0
NavigationInfrastructure	34	70	39	0
SurveillanceInfrastructure	34	21	17	0
Obstacle	12	27	8	0
BaseInfrastructureCodelists	100	0	0	1574
Meteorology	74	69	15	97
Stakeholders	148	131	40	316
Common	78	44	19	396

For example, Stuckenschmidt and Schlicht [14] applied structural characteristics such as target module size and number of target modules to determine suitable partitions of an input ontology. Ontology 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 [13].

In BEST, we employed the latter strategy and more specifically a technique called Syntactic Locality Modularisation [14, 15], for extracting ontology modules from the AIRM, AIXM and IWXXM monolithic ontologies. The reason for this choice was primarily based on the use cases we had in the BEST project. First, all the three original models were structured according to topicality. For example, the AIRM model is organized into different subject fields, where each subject field is responsible for describing semantics about a certain topic, for example "Aircraft" or "Meteorology". Secondly, the semantic containers are described in detail in Sect. 4.

The BEST project developed a set of prototypes supporting different steps in the modularization process. The module extraction functionality was implemented in Java using the OWL API library (version 4.1.2). It is important to realize that ontology modularization is not just about extracting isolated modules from a monolithic representation. To have a consistent set of modules in the end (i.e. a network of modules), one must capture and maintain dependencies among the extracted modules and resolve any redundancy that might exist. For this reason, a set of prototype ontology modularization applications were developed [2].

2.3. Development of Fine-grained Ontologies

The ontologies and ontology modules generated from exchange models and from the AIRM contain rather abstract and generic concepts. For the purpose of ontology-based information filtering they are extended with more fine-grained concepts.

[2] `https://github.com/sju-best-project/ontology-modules`

In the remainder of this paper we take an abstract perspective on ontologies, since the Semantic Container approach only uses the set of concepts defined by the ontology and the subsumption hierarchy (also referred to as generalization hierarchy) of these concepts. For the Semantic Container approach it is not relevant how these concepts are defined and whether and how the subsumption hierarchy is derived automatically.

Definition 1 (Ontologies). *An* ontology $o \in \mathcal{O}$*, taken from the universe of ontologies* $\mathcal{O}$ *defines a set of concepts* $V_o \subseteq \mathcal{V}$ *taken from the universe of concepts* $\mathcal{V}$*. Concepts are arranged in a subsumption hierarchy* $G_o \subseteq V_o \times V_o$*. We say concept* v is properly subsumed by *concept* v'*, denoted by* $v \sqsubset v'$*, if* $(v, v') \in G_o^+$*, where* G_o^+ *is the transitive closure of* G_o*; and* v is subsumed by v'*, denoted as* $v \sqsubseteq v'$*, if* $v \sqsubset v' \vee v = v'$*.*

Example 1. An ontology *AircrOnt* defines concepts such as *Aircraft*, from the AIRM, and more fine-grained concepts such as *HeavyAircraft* (i.e., an aircraft in the HEAVY wake turbulence category), *Landplane* and *Boeing747*, i.e., $V_{AircrOnt} = \{$*Aircraft, HeavyAircraft, Landplane, Boeing747,* $\ldots\}$. These concepts are arranged, by automatic classification, in a subsumption hierarchy (also referred to as generalization hierarchy) $G_{AircrOnt} = \{$(*Boeing747*, *Landplane*), (*Boeing747*, *HeavyAircraft*), (*Landplane*,*Aircraft*), (*HeavyAircraft*, *Aircraft*), $\ldots\}$. *Boeing747* is thus properly subsumed by *Landplane*, denoted as *Boeing747* $\sqsubset$ *Landplane*, and also by *HeavyAircraft* and *Aircraft*. Note, the ontology further contains concept definitions (e.g., the necessary and sufficient conditions for an aircraft to be a *HeavyAircraft*) and properties which are not subject to the Semantic Container approach but are rather used by an ontology reasoner to derive the subsumption hierarchy of concepts. □

3. Filtering of ATM Information

In this section we define essential characteristics of ATM information and of ontology-aware information filtering that underlie the Semantic Container approach.

3.1. Abstract Model of ATM Information

We now introduce an abstract model of the information provisioned in SWIM and covered by the Semantic Container approach. Note, there are other kinds of information exchanged in SWIM which are not covered by this characterization and are thus not covered by the Semantic Container approach.

In SWIM, information is provided and exchanged based on standardized exchange models such as the Aeronautical Information Exchange Model (AIXM) [1], the Flight Information Exchange Model (FIXM) [2], and the ICAO Weather Information Exchange Model (IWXXM) [3] which, in turn, should conform to the ATM Information Reference Model (AIRM) [7] and are thus covered by the ontology modules described in Sect. 2.

In the abstract model underlying the Semantic Container approach, data items (often also referred to as messages or as information items) are the basic unit of information provided in SWIM. There are many different types of data items based on different information exchange models, such as digital notices to airmen (NOTAMs) based on the AIXM, and weather observations (METARs) or weather forecasts (TAFs) based on IWXXM. The Semantic Container approach is based on the abstract notion of data item, abstracting from the specifities of the different information exchange models and different data item types.

Once issued (i.e., created and published) at some point of time, a data item is immutable but may be replaced or canceled by another data item. Depending on its type, a data item may come with additional timing information, such as the time of observation (of a METAR), or the time a NOTAM becomes effective and the time it ceases to be effective. While these type-specific timing information may be used for type-specific filtering, the definition of the Semantic Container approach is especially dependent on the creation time, i.e., the point of time a data item is issued.

Data items are published by a data provider, acting as authoritative data source and origin. For example, GroupEAD issues NOTAMs and is the authoritative source and origin for the NOTAMs it issues. Similarly, the Met Office issues METARs and TAFs. In order to apply an information filter, one has to first specify the data item type(s) and the data origin(s) of the to-be filtered data items.

The following definition summarizes the common properties of data/information items covered by the Semantic Container approach.

Definition 2 (Data Items). *A data item* $i \in \mathcal{I}$ *from the universe of data items,* $\mathcal{I}$*, has a data item type* $d_i \in \mathcal{D}$ *taken from the universe of data item types,* $\mathcal{D}$*, an origin* $s_i \in \mathcal{S}$ *taken from the universe of data sources,* $\mathcal{S}$*, and a creation time* $t_i \in T$ *taken from the totally ordered set of timestamps* T*. Given two timestamps* t *and* t'*,* t *is either before or equal to* t'*, denoted as* $t \preceq t'$*, or* t' *is before* t*, denoted as* $t' \prec t$*.*

Example 2. An example data item *i1* is a message of type *NOTAM*, i.e., $d_{i1} = NOTAM$, issued by data source *GroupEAD*, i.e., $s_{i1} = GroupEAD$, at time 16:32 on the 15th of October, 2019, i.e., $t_{i1} = 201910151632$. Note, the content of data item *i1*, e.g., notifying of a partial runway closure at Frankfurt airport from October 20 to October 25, is not subject to the Semantic Container approach but will rather be analyzed by filter rules to decide if *i1* is relevant or not with regard to particular filter criteria. □

3.2. Ontology-aware Information Filtering

We now sketch the difference between information filtering and data selection and introduce faceted ontology-

aware information filtering, the filtering approach underlying the Semantic Container approach.

Information filtering is akin to data selection in database querying (cf. selection in relation algebra). The main difference between information filtering and data selection is the complexity of the query. While the information need underlying data selection can be directly expressed in a straightforward manner in a database query and be executed by a query engine, the information need underlying information filtering is more complex which would make it extremely difficult to formulate it directly in a query language. Instead, the information need for information filtering should be expressed in a more declarative form, with rule-based systems matching information need to available ATM information. An example of a rule-based system for ATM information filtering is SemNOTAM [4].

Faceted information filtering facilitates the specification of an information need by filter criteria specifying filter conditions for different orthogonal facets. In ontology-aware information filtering, every filter facet is associated with an ontology and the possible filter conditions are the concepts of that ontology.

The knowledge for filtering ATM information based on ontologies is not contained in the definitions of concepts in the ontologies but needs to be expressed in additional rules for matching data items and filter conditions. For each facet, a (typically rather complex) set of rules specifies which information items are relevant or are not relevant for which concepts.

We define essential properties of an ontology-aware information filtering system to be used in conjunction with the Semantic Container approach. Such a system defines a set of filter facets with the filter function (typically implemented by a set of rules) of each facet adhering to the subsumption hierarchy of the associated ontology.

Definition 3 (Filter facets). *A* facet $f \in \mathcal{F}$*, taken from the universe of facets* $\mathcal{F}$*, is a triple* (o_f, V_f, ρ_f)*, where* $o_f \in \mathcal{O}$ *is the ontology from which the facet's possible filter conditions* V_f *are taken, i.e.,* $V_f \subseteq V_{o_f}$*, and partial Boolean function* $\rho_f : \mathcal{I} \times V_f \to \{\text{true}, \text{false}\}$ *determines the relevance of data items with regard to filter conditions. The relevance of a data item* $i \in \mathcal{I}$ *with regard to a filter condition* $v \in V_f$ *is defined for facet* f*, denoted as* $(i, v) \in dom_{\text{def}}(\rho_f)$*, or is undefined, denoted as* $(i, v) \notin dom_{\text{def}}(\rho_f)$*. In the former case, data item* i *is either* relevant *with regard to* v*, denoted as* $\rho_f(i, v)$*, or* irrelevant*, denoted as* $\neg\rho_f(i, v)$*.*

Note, given a partial function $\mathfrak{f}$, we write $dom_{\text{def}}(\mathfrak{f})$ to refer to the domain of definition of $\mathfrak{f}$.

Example 3. A *aircraft* facet refers to the aircraft ontology *AircrOnt* from Ex. 1, $o_{aircraft}$ = *AircrOnt* and uses concepts $V_{aircraft}$ = { *Aircraft*, *Landplane*, *HeavyAircraft*, *Boeing747*, ... } as possible filter conditions. According to the filter rules which analyze the contents of data items, the relevance of data item *i1* from Ex. 2 with regard to filter condition *Landplane* for facet *aircraft* is undefined, i.e., $(i1, \mathit{Landplane}) \notin dom_{\text{def}}(\rho_{aircraft})$, but it is defined for heavy aircraft, i.e., $(i1, \mathit{HeavyAircraft}) \in dom_{\text{def}}(\rho_{aircraft})$, as relevant, i.e., $\rho_{aircraft}(i1, \mathit{HeavyAircraft})$. □

Each filter function ρ_f is defined by a shared set of rules (since the Semantic Container approach is agnostic towards the concrete rule mechanism we may also say that filter functions are realized by filter services) which adheres to the subsumption hierarchy of concepts as defined below. This adherence to the subsumption hierarchy facilitates a top-down evaluation of the filter function. Top-down evaluation is important for the Semantic Container approach since it facilitates the *reuse* of packages of coarse-grained filtered information as input for more fine-grained filtering. When the coarse-grained filter criteria subsume the fine-grained filter criteria, applying the fine-grained filter criteria on the coarse-grained filtered information will produce the same result as applying the fine-grained filter criteria directly on the unfiltered set of data items.

Definition 4 (Subsumption Hierarchy Adherence). *Let* $v \in V_f$ *and* $v' \in V_f$ *be two concepts used as filter conditions of filter facet* f *with concept* v *subsuming concept* v'*,* $v' \sqsubseteq v$*. If a data item* $i \in I$ *is relevant with regard to* v *in the context of facet* f *then* i *is also relevant to* v' *in the context of* f*, i.e.,* $(i, v) \in dom_{\text{def}}(\rho_f) \wedge \rho_f(i, v) \wedge v' \sqsubseteq v \Rightarrow \rho_f(i, v')$*. Thus, if* i *is relevant to* v' *it is either also relevant to* v *or its relevance regarding* v *is undefined. If a data item* $i \in I$ *is not relevant with regard to* v *then* i *is also not relevant to* v'*, i.e.,* $(i, v) \in dom_{\text{def}}(\rho_f) \wedge \neg\rho_f(i, v) \wedge v' \sqsubseteq v \Rightarrow \neg\rho_f(i, v')$*. Thus, if* i *is not relevant to* v' *it is either also not relevant to* v *or its relevance regarding* v *is undefined.*

Example 4. (continued from Ex. 3) Since *i1* is relevant for heavy aircraft it is also relevant for *Boeing747*, i.e., $\rho_{aircraft}(i1, \mathit{Boeing747})$ is also true. □

4. The Semantic Container Approach

In this section we introduce semantic containers as packages of filtered information. We begin our discussion and formalization of semantic containers with elementary logical containers, the core construct of the Semantic Container approach used to represent elementary information needs and information packages alike. We then continue with subsumption checking as a basis for searching for and reusing existing containers. Furthermore, we introduce versioning and physical allocation of semantic containers and finally discuss composite containers which collect data items of different types and/or different origins.

In the previous section we discussed the kinship between information filtering and data selection. From that perspective, semantic containers are akin to views in relational databases with the main difference that they are not

based on data selection (like views in relational databases) but rather on ontology-aware information filtering.

4.1. Ontology-based Description of Filtered Information

The core construct of the Semantic Container approach is the *elementary logical container* which represents the set of data items of a given origin, a given data item type, and filtered according to the given filter criteria. That set of data items is also referred to as the container's content.

Definition 5 (Elementary Logical Containers). *An elementary logical container $c \in \mathcal{C}$ is defined by a triple (d_c, s_c, γ_c) where $d_c \in D$ is the container's data item type, i.e., the type of the data items contained in the container; $s_c \in S$ is the origin (also referred to as authoritative source) of the container's data items, and $\gamma_c : F \to V$ are filter criteria given as a partial function mapping facets to concepts. The contents of a container, $I_c \subseteq \mathcal{I}$, are all data items of type d_c issued by s_c and not irrelevant with regard to filter criteria γ_c, i.e., $I_c \stackrel{\text{def}}{=} \{i \in I \mid d_i = d_c \wedge s_i = s_c \wedge \nexists f \in dom_{\text{def}}(\gamma_c) : \neg\rho_f(i, \gamma_c(f))\}$.*

Example 5. An elementary logical container *c1* represents *NOTAMs*, $d_{c1} = NOTAM$, issued by *GroupEAD*, $s_{c1} = GroupEAD$, filtered for flights from Frankfurt to London Heathrow conducted with a heavy aircraft, $\gamma_{c1} = \{$ *location* $\mapsto$ *FRAtoLHR*, *aircraft* $\mapsto$ *HeavyAircraft* $\}$. With data item *i1* being of type *NOTAM*, being issued by *GroupEAD*, and being neither irrelevant for location *FRAtoLHR* nor for aircraft *HeavyAircraft*, it will be one of the data items contained by *c1*, i.e., $i1 \in I_{c1}$. A special kind of elementary logical container are containers with unfiltered content, e.g., *c0* represents all *NOTAMs* issued by *GroupEAD*, i.e., $d_{c0} = NOTAM$, $s_{c0} = GroupEAD$, $\gamma_{c0} = \{\}$. □

4.2. Ontology-based Discovery of Filtered Information

The Semantic Container approach facilitates discovery of filtered ATM information (i.e., available semantic containers) to, first, fulfill a given information need or, second, to derive possible derivation chains for existing containers. In both cases, with information needs also represented as logical containers, the goal is to discover the available containers from which the contents of the given logical container can be derived by applying the given filter criteria. Based on subsumption checking, the system determines available containers which subsume the given logical container, and thus always contain a super-set of the contents sought. Based on the assumption that filter rules adhere to the concept subsumption hierarchies (see Def. 4) we can define the derivation of a subsumption hierarchy of semantic containers from concept subsumption hierarchies.

Definition 6 (Subsumption of Semantic Containers). *A container $c \in \mathcal{C}$ subsumes another container $c' \in \mathcal{C}$, denoted as $c' \sqsubseteq c$, if both have the same origin and the same data item type and for every facet of c's filter criteria, the filter condition of c subsumes the filter condition of c', i.e., $c' \sqsubseteq c \stackrel{\text{def}}{=} d_{c'} = d_c \wedge s_{c'} = s_c \wedge \forall f \in dom_{\text{def}}(\gamma_c) : \gamma_{c'}(f) \sqsubseteq \gamma_c(f)$. Container c properly subsumes c' if c subsumes c' and c and c' have different filter criteria, i.e., $c' \sqsubset c \stackrel{\text{def}}{=} c' \sqsubseteq c \wedge \gamma_c \neq \gamma_{c'}$.*

Example 6. Container $c2 \in \mathcal{C}$ represents the information need of an aircraft pilot preparing for a flight from Frankfurt to London Heathrow on Oct-19 using a Boeing 747, $\gamma_{c2} = \{$ *aircraft* $\mapsto$ *Boeing747*, *location* $\mapsto$ *FRAtoLHR*, *time* $\mapsto$ *Oct-19* $\}$. The aircraft pilot needs all the relevant *NOTAMs*, $d_{c2} = NOTAM$, issued by *GroupEAD*, $s_{c2} = GroupEAD$. Information need *c2* is subsumed by container *c1*, $c2 \sqsubseteq c1$, since they have the same data item type, $d_{c1} = d_{c2}$, the same origin $s_{c1} = s_{c2}$, and for every facet, *aircraft* and *location*, of γ_{c1}, the filter condition of *c1*, *HeavyAircraft* and *FRAtoLHR*, respectively, subsumes the filter condition of *c2*, *Boeing747* and *FRAtoLHR*, respectively. *c1* also properly subsumes *c2*, $c2 \sqsubset c1$, since their filter criteria are not the same. Now, consider an additional container *c3* with $d_{c3} = NOTAM$, $s_{c3} = GroupEAD$ and $\gamma_{c3} = \{$ *location* $\mapsto$ *FRAtoLHR* $\}$ which, based on its definition, properly subsumes both *c1* and *c2*. Obviously, container *c0* with its unfiltered content, properly subsumes *c1*, *c2*, and *c3*. □

In order to reduce the effort for further information filtering, only the most relevant information packages should be discovered. Consequently, out of a set of containers subsuming a given logical container only the most specific ones, i.e., those with the most specific filter criteria, should be selected. Such a most-specific subsumer contains all the needed information and there is no more specific container which also contains all the needed information. If there are multiple most-specific subsumers, their intersection may serve as input to further filtering.

Definition 7 (Most-Specific Subsumer). *Given a container $\bar{c} \in \mathcal{C}$ and a set of available containers $C \subseteq \mathcal{C}$, the set of most-specific subsumers of $\bar{c}$ in C, denoted by $\hat{C}_{(\bar{c},C)}$, is given by $\hat{C}_{(\bar{c},C)} \stackrel{\text{def}}{=} \{c \in C \mid \bar{c} \sqsubseteq c \wedge \nexists c' \in C : \bar{c} \sqsubseteq c' \sqsubset c\}$. Note, if $\bar{c}$ is in the set of available containers C, then $\bar{c}$ is its own most-specific subsumer in C.*

Example 7. (continued from Ex. 6) Let the set of available containers consist of containers *c0* (containing all NOTAMs from GroupEAD), *c1* (like *c0* but filtered for flights from FRA to LHR conducted with a heavy aircraft), and *c3* (like *c0* but filtered for flights from FRA to LHR). The semantic container *c2*, representing an information need of an aircraft pilot (for NOTAMs from GroupEAD for a flight from FRA to LHR with a Boeing747 on Oct-19), is properly subsumed by containers *c0*, *c1* and *c3*. Since *c0* properly subsumes *c3* which in turn properly subsumes *c1*, only *c1* is a most-specific subsumer of *c2*, denoted as $\hat{C}_{(c2,\{c0,c1,c3\})} = \{c1\}$. Container *c1* is thus the most relevant available container for information need *c2* and may

be provided as-is to the aircraft pilot or as input for further filtering. Assuming there is a further available container *c4* (like *c0* but filtered for flights on Oct-19) then the most-specific subsumers of *c2* are *c1* and *c4*, denoted as $\hat{C}_{(c2,\{c0,c1,c3,c4\})} = \{c1, c4\}$ In this case, the intersection of containers *c1* and *c4* may be provided as-is to the aircraft pilot or as input for further filtering. □

4.3. Versioning of Semantic Containers

ATM information is inherently dynamic. Data sources frequently publish new data items leading to updates of the contents of semantic containers. Semantic containers keep track of every update by keeping a version history which allows to rebuild past states of information. This is important in the follow-up of incidents to analyze decisions and their information base.

Definition 8 (Container Versions). *The set of version timestamps $T_c \subseteq T$ of a container c is the set of timestamps where the container's data origin s_c issued data items of the container's data item type d_c, i.e., $T_c \stackrel{\text{def}}{=} \{t \in T \mid \exists i \in \mathcal{I} : d_i = d_c \wedge s_i = s_c \wedge t_i = t\}$. For every version timestamp $t \in T_c$ there exists a container version $c\langle t\rangle$. A container version $c\langle t\rangle$ contains all the container's data items until t, i.e., $I_{c\langle t\rangle} \stackrel{\text{def}}{=} \{i \in I_c \mid t_i \preceq t\}$. The delta set of $c\langle t\rangle$, denoted as $I^+_{c\langle t\rangle}$, is the (possibly empty) set of data items added with version $c\langle t\rangle$, and is defined as $I^+_{c\langle t\rangle} \stackrel{\text{def}}{=} \{i \in I_c \mid t_i = t\}$.*

Example 8. (continued from Ex. 5 and Ex. 2) So far, container *c1* has a single data item *i1* as content with creation time t_{i1} = *201910151632* and thus has a single version *c1⟨201910151632⟩* and a single delta set $I^+_{c1\langle 201910151632\rangle}$ = { *i1* }. At time 20:12 on the 16h of October, 2019, *GroupEAD* issues three more NOTAMs, *i2*, *i3*, and *i4*, with $t_{i2} = t_{i3} = t_{i4} = 201910162012$ of which *i2* and *i4* fulfil the filter criteria of *c1*. Thus, a new version *c1⟨201910162012⟩* comes to existence with delta set $I^+_{c1\langle 201910162012\rangle}$ = { *i2*, *i4* } and the contents of the new container version being $I_{c1\langle 201910162012\rangle}$ = { *i1*, *i2*, *i4* }. Fifteen minutes later, *GroupEAD* issues two more NOTAMs, *i5* and *i6*, with $t_{i5} = t_{i6} = 201910162027$, which are not relevant with regard to the filter criteria of *c1*. Thus, a new version *c1⟨201910162027⟩* comes into existence with an empty delta set, $I^+_{c1\langle 201910162027\rangle} = \{\}$, and thus having the same contents as the prior version, i.e., $I_{c1\langle 201910162012\rangle} = I_{c1\langle 201910162027\rangle}$. This new version with empty delta set indicates that the latest updates from *GroupEAD*, namely NOTAMs *i5* and *i6*, have been considered but were filtered out as irrelevant. □

4.4. Distribution of Filtered Information

Semantic containers come with a simple, yet powerful, distribution and replication concept: Each logical container has a (possibly empty) set of physical copies stored at different locations, making the container's contents available also off-line or in case of network failures. It is important to note that the update cycle of a physical container may be independent of the versioning of the corresponding logical container, where the latter solely depends on when the data origin issues data items. This independence paves the way for both push- and pull-based handling of updates.

In case of unavailability of primary sources of filtered information, a physical container may be updated from secondary sources with lesser information quality producing degenerated container versions which will later be re-synchronized. The re-synchronization potentially comprises adding and/or deleting data items from the physical representation of a logical container version – in order to be able to reconstruct the information available to a user or application at a previous point in time, in such a case, physical container versions are not changed but additional physical container versions are created. It is up to the service or application which uses the contents of the container to adequately deal with potentially degenerated quality and with potential conflicts due to re-synchronization – typically, a user application will inform the user about the temporarily lower quality of information and about the eventual re-synchronization.

Definition 9 (Container Allocation). *A logical container $c \in \mathcal{C}$ has a (possibly empty) set of physical copies allocated at different locations $L_c \subseteq \mathcal{L}$ taken from the universe of locations $\mathcal{L}$. A physical container $c@l$ is container c allocated at location $l \in L_c$.*

- *A physical container has an update history with $T_{c@l} \subseteq T$ being the set of update timestamps.*
- *A physical container version $c\langle t\rangle@l\langle t'\rangle$ is the physical representation of logical container version $c\langle t\rangle$ at location l produced with the physical container update at time $t' \in T_{c@l}$.*
- *If a physical container version $c\langle t\rangle@l\langle t'\rangle$ is marked as final, denoted as $\mathsf{final}(c\langle t\rangle@l\langle t'\rangle)$, then its content $I_{c\langle t\rangle@l\langle t'\rangle}$ and its delta set $I^+_{c\langle t\rangle@l\langle t'\rangle}$ must be equal to content $I_{c\langle t\rangle}$ and delta set $I^+_{c\langle t\rangle}$, respectively, of the corresponding logical container version $c\langle t\rangle$, i.e., $\mathsf{final}(c\langle t\rangle@l\langle t'\rangle) \Rightarrow I_{c\langle t\rangle} = I_{c\langle t\rangle@l\langle t'\rangle} \wedge I^+_{c\langle t\rangle} = I^+_{c\langle t\rangle@l\langle t'\rangle}$.*

Example 9. (continued from Ex. 8) Logical container *c1* has two physical copies *c1@Fra* and *c1@Lon* allocated at locations *Fra* and *Lon*, i.e., L_{c1} = { *Fra*, *Lon* }. Physical container *c1@Fra* is updated at 20:13 producing a final physical container version *c1⟨...2012⟩@Lon⟨...2013⟩*. Note: timestamps are abbreviated, *⟨...2012⟩* stands for *⟨201910162012⟩*.

Due to a network outage the data source at *GroupEAD*, physical container *c1@Fra*, as well as the NOTAM filtering service are unavailable from location *Lon* from

time 20:11. Physical container *c1@Lon* thus becomes stale. When data source *GroupEAD* becomes available again at time 20:14, container *c1@Lon*, in order to avoid missing possibly mission-critical information, takes the unfiltered information from *GroupEAD* to produce a preliminary (and degenerated) version $c1\langle...2012\rangle@Lon\langle...2014\rangle$ with delta set $I^+_{c1\langle...2012\rangle@Lon\langle...2014\rangle} = \{i2, i3, i4\}$ and later a degenerated version with delta set $I^+_{c1\langle...2027\rangle@Lon\langle...2027\rangle} = \{i5, i6\}$.

As soon as physical container *c1@Fra* becomes available again from location *Lon* at time 20:30, the degenerated versions of $c1@Lon$ are archived for later reproducibility, and replaced with $c1\langle...2012\rangle@Lon\langle...2030\rangle$ and $c1\langle...2027\rangle@Lon\langle...2030\rangle$ by replication from $c1@Fra$ to finally get physical delta sets $I^+_{c1\langle...2012\rangle@Lon\langle...2030\rangle} = \{i2, i4\}$ and $I^+_{c1\langle...2027\rangle@Lon\langle...2030\rangle} = \{\}$. □

4.5. Composition of Semantic Containers

A consumer of ATM information in order to prepare for a task typically has a complex information need comprising various types of data items, issued by different information providers, and often also combining different sets of filter criteria. The description of such information needs and the management of corresponding information packages is made possible by composite containers.

Definition 10 (Container Composition). *Elementary containers $\mathcal{C}$ and composite containers $\mathcal{K}$ are arranged in a part-of hierarchy $P \subset (\mathcal{C} \cup \mathcal{K}) \times \mathcal{K}$. With P^+ being the transitive closure of P we derive for composite container k*

1. *its set of component elementary containers* $C_k \stackrel{\text{def}}{=} \{c \in \mathcal{C} \mid (c,k) \in P^+\}$,
2. *its contents* $I_k \stackrel{\text{def}}{=} \bigcup_{c \in C_k} I_c$.

Different classes of composite containers can be differentiated based on whether their components have all the same origins, the same filter criteria, and/or the same data item type. A composite container k is called *homogeneous* if all its components have the same data item type, i.e., $\forall c, c' \in C_k : d_c = d_{c'}$, or *heterogeneous*, otherwise. It is called *single-origin* if all its components have the same origin, i.e., $\forall c, c' \in C_k : s_c = s_{c'}$, or *multi-origin*, otherwise. It is called *conjunctively filtered* if all its components have the same filter criteria, i.e, $\forall c, c' \in C_k : \gamma_c = \gamma_{c'}$, or *disjunctively filtered*, otherwise.

Example 10. Aircraft pilots preparing for a flight from Frankfurt to London Heathrow on Oct-19 are not only interested in relevant NOTAMs provided by GroupEAD but also in weather forecasts of type TAF provided by DWD and MetOffice. Elementary containers *c4* and *c5* with $d_{c4} = d_{c5} = TAF$, and $s_{c4} = DWD$ and $s_5 = MetOffice$ and $\gamma_{c4} = \gamma_{c5} = \{location \mapsto FRAtoLHR, time \mapsto Oct\text{-}19\}$ are combined into a homogeneous composite container *k1*. Elementary container *c6* with $d_{c6} = NOTAM$ and $s_{c6} = GroupEAD$ and $\gamma_{c6} = \gamma_{c5}$ is combined with *k1* into a heterogeneous composite container *k2*. The part-of hierarchy is $P = \{(c4, k1), (c5, k1), (c6, k2), (k1, k2)\}$. The set of component elementary containers of *k2* is $C_{k2} = \{c4, c5, c6\}$ and its contents are the union of the contents of its elementary containers $I_{k2} = I_{c4} \cup I_{c5} \cup I_{c6}$. Composite container $k2$ is called a multi-origin container because its component containers have different origins, namely *GroupEAD*, *DWD*, and *MetOffice*. It is called a conjunctively-filtered container since its components all have the same filter criteria, namely $\{location \mapsto FRAtoLHR, time \mapsto Oct\text{-}19\}$. It is a heterogeneous container since its components have different data item types, namely *NOTAM* and *TAF*. □

5. Discussion: Semantic Containers and SWIM

In this section we discuss the application of semantic containers in SWIM and align the discussion with the constructs of the Semantic Container approach formally defined in the previous section.

Semantic containers encapsulate the data logic of SWIM services [16] and clearly separate it from business and presentation logic. A semantic container allows developers to organize and make sense of the provided ATM information. A semantic container provides a SWIM application or service with all the relevant ATM information, hiding the complexities of compiling the information package. Semantic containers come with ontology-based metadata that allow users, services, and applications to know what the content of the container is and assess the freshness as well as the quality of the data.

The provisioning of semantic containers for a specific purpose encompasses the discovery of existing source containers and often further filtering steps [17]. These tasks are supported by matching of information need and available semantic containers and services. Based on a formal ontology-based specification - employing the ontology infrastructure - of the information needed for an operational scenario, the semantic container management system should find existing containers that most closely fulfil the specified information need and identify missing processing steps. Note that the implementation of the corresponding algorithms of identifying missing processing steps to obtain a full match is left to future work. More information about the definition of a semantic container can be found in [18] and [19].

Effective use of the Semantic Container approach developed in BEST depends on the existence of a Semantic Container Management System (SCMS) controlling the replication, distribution and consistency of containers. In the field of distributed databases, there are many existing techniques for distribution, replication and consistency management, mostly based on a single generic data model. In BEST, we refine existing techniques using different types of models for different kinds of information [20]. The specific configuration of semantic containers and

realization of update strategies are left to the SCMS, the Semantic Container approach only serves as a basis for their realization.

High availability of information and low network load are key goals for the success of the SWIM approach. Semantic containers, supported by an SCMS, can contribute significantly to these goals. The Semantic Container approach distinguishes between logical and physical containers to indicate which containers are allocated at which nodes. The Semantic Container approach also allows for the definition of different versions of containers, supporting consistency management and different forms of synchronization. Finally, semantic containers allow for traceability of data provenance, and definition of composite containers that gather data from lower-level elementary containers. We stress that the Semantic Container approach applies to various types of ATM information as well.

In SWIM, different applications require different types of ATM information at various degrees of freshness and availability. An aircraft pilot may, for example, request current weather data. For availability's sake, consistency may be sacrificed: Slightly outdated weather information is better for a pilot than none. With respect to notifications about runway closures, on the other hand, pilots require fresh data because wrong information would entail potentially disastrous consequences. Semantic containers allow us to make the inherent trade-off between freshness and high availability tangible for the consumer of ATM information: A semantic container packages ATM information and the resulting packages can be redundantly stored at multiple locations for high availability; administrative metadata indicate freshness and data quality. As for the metadata in the core Semantic Container approach presented in Sect. 4, every semantic container has origin, data item type, and filter criteria; every version has a version timestamp; every physical container has a location; and every physical container version has an update time. Further metadata, for example describing quality and provenance, can be attached to logical and physical containers as well as logical and physical container versions. It is left open to future work to develop update strategies based on additional metadata attached to semantic containers and to realize them as part of an SCMS.

Semantic containers also increase availability of the overall system by considering multiple sources of ATM information which semantic containers may be derived from. The semantic container metamodel [17] allows for the representation of multiple data sources for the same semantic container. An SCMS may switch dynamically and transparently between different sources. Different sources may provide the same data with different quality to ensure that the consumer is alert to any reduction in quality of service. A primary source is a source with the highest data quality among the sources of the container. Secondary sources of lesser quality are only used when no primary source is available at the expected freshness. The semantic container metamodel [17] goes beyond the core of the Semantic Container approach presented in Section 4 by facilitating a distinction between primary and secondary sources of filtered information based on their assumed information quality.

An advantage of packaging ATM information in semantic containers is the possibility to allocate relevant information directly in the aircraft that operates a specific flight. The semantic container can be created a couple of days prior to the date the actual flight takes place, being filled with relevant information in advance. Shortly before the flight, at the departure airport with high bandwidth, the container can be uploaded onto the plane, and during the flight updated with only the critical information or information that requires low bandwidth. The specific configurations for en-route information provisioning based on semantic containers has to be realized as part of a SCMS.

ATM information is inherently dynamic: Government authorities and authoritative sources, e.g., GroupEAD, push new data and updates to existing data. Hence, the Semantic Container approach requires a mechanism to keep the contained ATM information up to date. The Semantic Container approach comes with a versioning mechanism which is independent of its concrete realization and which paves the way for both push- and pull-based handling of updates. The concrete handling of updates and the choice of update strategies is left to the SCMS.

Multiple service consumers may request the same ATM information from a remote entity. Typically, each request for ATM information is processed individually, thereby putting stress on the available bandwidth. With a SCMS in place, SWIM services may cache frequently requested ATM information (e.g. weather data) as semantic containers. They can even store the semantic containers at locations where they are frequently needed, thereby reducing the bandwidth and computation effort. For example, a NOTAM filtering service may cache relevant NOTAMs for the most important flight routes as semantic containers. When concrete requests for specific flights come in, rather than sifting through the whole body of NOTAMs currently in place, the service may use the pre-filtered semantic containers as a starting point for further filtering. It is left open to future work to develop algorithms which optimize, with regard to frequent information needs, the provision of pre-filtered information packages and their allocation at different locations.

6. Architecture of a Semantic Container Management System

In this section we introduce a possible architecture for a SCMS. A semantic container management system (SCMS) fits well into the decentralized, service-oriented architecture of SWIM as it builds on the same technology stack. What an SCMS adds to SWIM is a decentralized system for the management of filtered ATM information and associated metadata on which other SWIM services can build.

A semantic container intended to be reused by others, can be made available as a kind of information service.

The Semantic Container approach, as formalized in this paper, may be implemented in many different forms; the Semantic Container approach is independent of a concrete software and data distribution architecture. Other and maybe more adequate architectures may be developed in the future based on the vast literature on distributed systems (e.g., [21][22]). The proposed architecture as described in this paper serves two purposes:

1. to give a more complete picture of a globally distributed SCMS, and
2. to serve as a starting point for the development of more advanced software and data distribution architectures.

An SCMS is distributed over multiple server locations and multiple client locations. Locations are connected over the internet. Container content and metadata are allocated redundantly at multiple locations. Centrally-provided software is run independently at the different locations which cooperate to provide globally-distributed semantic container management. Container content and metadata are allocated redundantly at multiple locations. A semantic container consists of location-independent metadata (represented by the logical semantic container), location-dependent metadata (represented by the physical semantic container) and content (also referred to as data/information, e.g., a set of AIXM Digital NOTAMs).

The container metadata can be represented as RDF triples [10]. All container metadata can thus be collected into an RDF graph. This RDF graph of all semantic containers is fully replicated at every server location and partially replicated at client locations. Each location runs an RDF database management system (a.k.a. graph store) and SPARQL query engine for storing, modifying and querying (parts of) the RDF graph. Modifications of metadata at some location are replicated in an asynchronous manner to other locations to provide for redundancy of metadata in case of connection or network failures. Replica consistency of metadata is maintained by giving priority to most recent writes.

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

Each server location independently runs a software package which makes available functionality for managing and querying data and metadata via RESTful web services. A client location (or sink), e.g., an electronic flight bag on board of an aircraft, may run a client variant of the software package which provides a subset of this functionality. The software package (in its server and client variants) is distributed from a central software repository.

A client location provides functionality for:

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

A server location additionally provides functionality for:

5. Creating a semantic container, determining locations for its physical allocation
6. Calling services to derive/update the contents of semantic containers
7. Forwarding modifications of semantic containers to client containers via push and pull
8. Creating, updating and deleting semantic containers
9. Discovery of semantic containers

7. Semantic Container Proof-of-Concept

In this section a proof-of-concept prototype is described in which the Semantic Container approach is integrated into a SWIM environment. Possible users of such an approach are all ATM stakeholders sharing, consuming, and exchanging SWIM data like Airlines, Airports, and ANSPs. Figure 1 gives an overview about the various systems involved in this scenario. The goal of the scenario is to give an idea how the TRL1 concept can be used in a complete SWIM life-cycle.

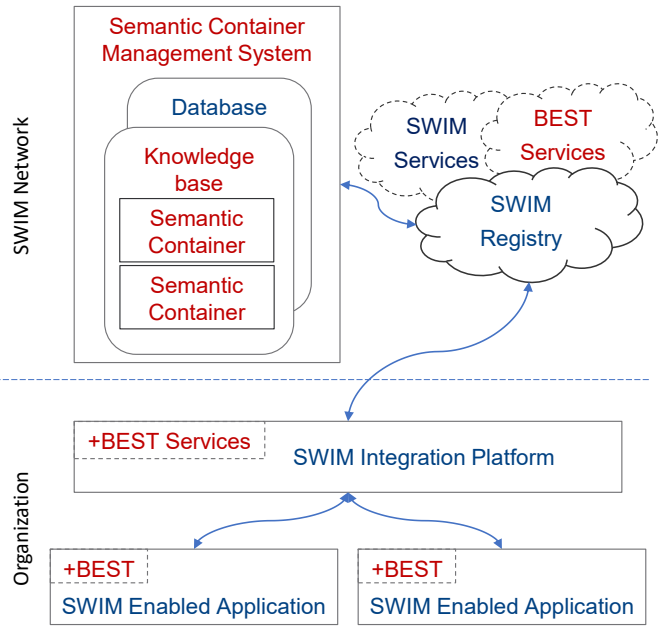

Figure 1: SWIM enhanced by Semantic Container [10]

For the scenario the Frequentis SWIM Registry (i.e., a collection of meta-data about available SWIM services

together with a web-based tool for exploring and querying these meta-data) was integrated to provide not only information about SWIM services but also about semantic containers via SWIM. The SCMS is used to define and create containers that are then visible through the SWIM registry. On an organizational level the Frequentis SWIM integration platform - called MosaiX - serves to configure organization internal the SWIM information for the specific SWIM applications. The information is ultimately accessed by a SWIM application. For the BEST integration we used an existing SESAR 1 prototype [23], namely the Integrated Digital Briefing from that project's WP13.2.2.

7.1. 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 these service providers (e.g., source, content, freshness) and harmonizes the both types in a single view. For the current use case a dedicated Frequentis Semantic Container Service Registry was adapted. This registry lists available SWIM services and semantic containers together with the functionality to show details about the services or edit entries.

Besides showing the content of existing semantic containers, it is also possible to create new containers in the SCMS. Opening the SCMS allows to view further details about the semantic containers.

7.2. Prototype: Semantic Container Management System

The SCMS is used to create and maintain semantic containers. In the section "Containers", a new container can be created. 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. It is also possible to already create a semantic container restricted to a specific aircraft type. After creating the container, the container hierarchy can be explored in the SCMS.

7.3. Integration: SWIM Integration Platform

To configure available data sources for an organization the Frequentis MosaiX SWIM Management Console is used.In this console, it is possible to establish, manage and monitor relevant data sources for an organization to provide access for those entities with legal permission.

The overall goals for this integration platform are interoperability within the heterogeneous application landscape, data integration (consistent metadata and versioning), as well as monitoring data streams especially in regard to security and compliance rules.

7.4. Integration: SWIM Integrated Digital Briefing

Based on the described use cases [19] Figure 1 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 digitally enhanced Pre-flight Information Bulletin (ePIB).

The SWIM application is managed by the organizational SWIM integration platform, 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.

The prototype has been integrated to use the BEST semantic container concept and is able to retrieve containerized filtered information to be used as such without further need of filtering. 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 added value provided by the semantic containers.

8. Usage Scenarios

To demonstrate characteristics and benefits of semantic containers in the SWIM environment, we present different usage scenarios. Business procedures were compared between current settings and the envisioned use of semantic containers. A more detailed account of this usage scenarios can be found in [10].

8.1. Pilot Briefing for a Flight from Germany to Austria

This scenario (see Fig. 2) compares centralized data distribution from Eurocontrol vs. decentralized data sources from local ANSPs for a pilot briefing application that requires specific (filtered) data. Data sources include current weather information, NOTAMs, and airport information from the start and destination airport.

While a centralized legacy services requires to handle a large number of messages and a big storage (about 45 GB for the major airports in Europe), the decentralized approach distributes this load as expected between effected ANSPs (580 MB in Austria and 5,4 GB in Germany). Since the majority of the data hardly changes (e.g., airport information) it makes sense to store this information in a distributed way.

8.2. Information Needs of a Fuelling Service

In this scenario (see Fig. 3), a fuelling service company operating at Frankfurt airport requests tailored information in order to plan and improve its operations. DFS

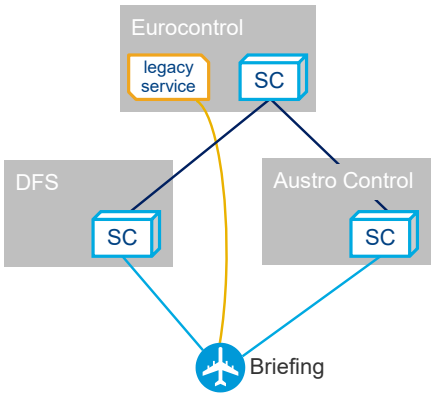

Figure 2: Pilot Briefing

provides flight plan information and Fraport AG supplies a query interface for further relevant information about the airport.

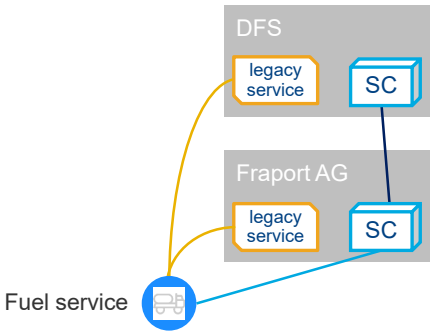

Figure 3: Fuelling Service

The benefit in this scenario is a single interface with consistent data. In case of inconsistent update intervals among services providing data, a semantic container can serve for data harmonization and also act as stable interface during updates.

8.3. Airline Managing its Fleet

In this scenario (see Fig. 4) various information providers relevant for an airline use Semantic Containers to funnel relevant data. Data sources in this scenario include flight plan information, localized weather data, NOTAMs, and airport information from ANSPs, as well as global weather information.

A solution employing semantic containers provides here standardized handling of all data to merge information from various sources and enables a 'single version of the truth' with most up-to-date data within an organisation. In addition, containers allow to collect data over a period of time and then simulate certain combined scenarios (e.g., severe weather conditions together with strikes at an airport) for an airline to study the impact on the entire fleet.

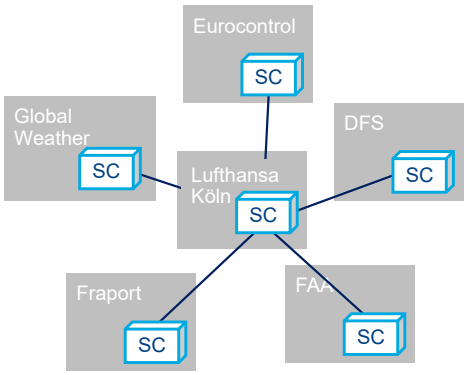

Figure 4: Airline Information Integration

8.4. Flight Data for an Intercontinental Flight

In this scenario (see Fig. 5), for an intercontinental flight from Australia to Europe, information from a number of FIRs must be collected and can be updated 'live' with a container on board the airplane. The data is again about weather information and relevant airports and NOTAMs en route together with data that is generated by the aircraft during the flight.

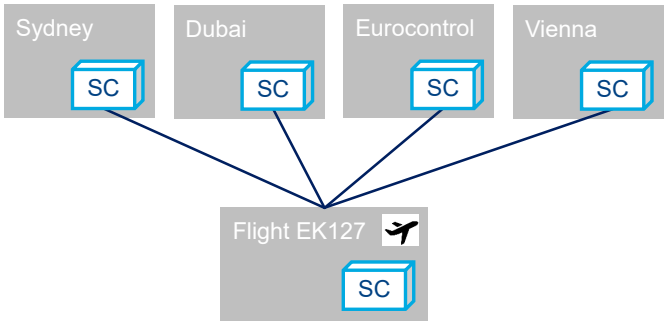

Figure 5: Flight Data for flight from Sydney to Vienna

For this scenario a legacy system does not exist, but such a solution would make instant scenario evaluation on board possible and would automatically generate a complete audit trail for each flight.

8.5. Benefits of Semantic Containers

In the five above-mentioned scenarios, three general benefits could be identified:

1. 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.
2. Improved message distribution: Data provider in the SWIM context process many requests from different applications. 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, improve reliability of

the overall network, and increases response times for SWIM applications.

3. 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 acts as a single interface with defined behaviour and thus allows a wide range of tests in a realistic environment. It would also be possible to record data traffic over a time 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.

9. Towards Trusted Semantic Containers

An SCMS providing mission-critical data and metadata requires special consideration of trustful communication to ensure *authentication*, *integrity*, and *non-repudiation* of data and metadata. Furthermore, when reappraising an incident, one needs a *trusted version history* of physical containers to be able to retrieve without doubt the version of the semantic container that was available when the incident happened.

In a decentralized system, trust can only be provided based on cryptography protocols [25]. Semantic containers can leverage cryptography protocols to provide trustful semantic container management and secure SWIM. Digital signatures can serve as basic technology to provide authentication, integrity and non-repudiation for single semantic containers.

Trusted timestamps will provide the foundation for making the update history of a physical container a trusted version history, that is, to make it tamper-proof. In a physical container with trusted version history, after each update the then current state of the container including metadata and content will get a trusted timestamp. The update timestamp (which is a container-internal non-trusted timestamp) will then be linked to the trusted timestamp and stored together in the physical container as part of its metadata (which will in turn receive a trusted timestamp together with the content after the next update).

In this section we will describe a notary service which we have built to provide such trusted timestamps. The notary service is freely available for experimentation and in turn builds on the Ethereum blockchain and a Time Stamping Authority (TSA).

9.1. The Need for a Notary Service

Digital content can be easily copied and manipulated. But when data is shared or sold, there is a need to record the exact state and timestamp of that data. Example scenarios for such requirements may be pieces of music by artists who want to prove the authorship of a tune, content on the internet such as hate postings that may be only briefly available, or the sale of data that is provided with a well-defined usage policy.

Another important aspect in the storage of this data are legal framework conditions such as the GDPR, and commercial aspects to generate such information (current state and timestamp) with only a low fee or even free.

9.2. Relevant Technical Building Blocks

There are already a number of technical solutions to individual aspects of the challenges described above and this section describes the technologies used.

- Snapshot of digital content: hash functions are well-established methods for producing an almost-unique fingerprint of a bit-stream. Currently, a SHA256 hash value is used which generates a 32-byte message.

- Distributed Ledger Technologies (DLT) allow decentralized storage of transactions that guarantee the immutability of the data contained therein. For the notary service the Ethereum blockchain is used as DLT.

- Merkle trees: Since storing a transaction in a distributed ledger is associated with a cost (mining fee), one option is to store several hash values in one transaction to reduce the price tag. At the same time, the available storage space is limited (comparable to the "Reference" field in a conventional bank transfer). Using Merkle trees (as described in RFC 6962), it is possible to combine any number of hash values in a binary tree, and only the root node to storage at a time. To verify a hash value, the root node is then necessary together with an audit proof.

- Trusted Timestamp: the algorithm described in RFC 3161 has been established in EU Regulation 910/2014 (eIDAS) as a binding method of electronic identification and trust services for electronic transactions. In this case, a provided hash value is cryptographically signed together with a current time stamp by a so-called timestamp authority. While storing a hash values in the blockchain is a decentralized way of making data immutable, Trusted Timestamps offer centralized immutability which is legally binding in the EU.

The use of a decentralized storage mechanisms (Ethereum blockchain) as trusted timestamp is not yet fully recognized in courts all over Europe and therefore it makes sense to additionally use a centralized and legally established trust service based on an EU regulation. On the other hand, if the certificate for the used Time Stamping Authority (currently Germany-based freeTSA.org) is revoked the blockchain-based timestamp is available as fallback.

9.3. Providing a Notary Service

The parts described above are combined into an easy-to-use notary service. In particular two functionalities must be provided for such a service: (i) notarization of a specified hash value, and (ii) verification of an existing hash value to retrieve its status, i.e., what is the associated timestamp. For usability reasons, these two requirements are combined in a single function so that the hash value passed is either saved if it does not exist and the respective status is always returned.

The described notary service is available via an API endpoint[3]. The following examples demonstrate its usage.

9.4. Usage Scenario

As an example, the hash value of the string "hello world" is used:

```
$ echo -n "hello world" | openssl dgst -sha256
(stdin)=b94d27b9934d3e08a52e52d7da7dabfac484efe37a538
    0ee9088f7ace2efcde9
```

9.4.1. First request

```
$ curl https://blockchain.ownyourdata.eu/api/doc?hash
    =b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee
    9088f7ace2efcde9
{
  "status": "new",
  "address": "",
  "root-node": "",
  "audit-proof": [],
  "dlt-timestamp": "",
  "tsr": "MIIN...5cXA==",
  "tsr-timestamp": "2019-08-15T20:29:59Z",
  "oyd-timestamp": "2019-08-15T20:29:59Z"
}
```

Result. The status "new" indicates that this hash value was passed for the first time and the field "oyd-timestamp" states the time provided by the notary service itself. Also, a Trusted Timestamp (field tsr) is created and returned base64 encoded. For easy readability, the timestamp "tsr-timestamp" received from the timestamp authority is also provided in the response.

The following command can be used to check the trusted timestamp (stored in the file.tsr file) for a hash value:

```
$ openssl ts -verify -data <(echo -n
    "b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee
    9088f7ace2efcde9") -in file.tsr -CAfile
    cacert.pem -untrusted tsa.crt
```

The files cacert.pem and tsa.crt, which are also required for this purpose, must be provided by the respective Timestamp Authority. The notary service uses a free timestamp service[4]. Currently the timestamp service from is used.

[3] https://blockchain.ownyourdata.eu/api/doc

[4] https://freetsa.org

9.4.2. Subsequent request

```
$ curl https://blockchain.ownyourdata.eu/api/doc?hash
    =b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee
    9088f7ace2efcde9
{
  "status": "exist",
  "address": "",
  "root-node": "",
  "audit-proof": [],
  "dlt-timestamp": "",
  "tsr": "MIIN...5cXA==",
  "tsr-timestamp": "2019-08-15T20:29:59Z",
  "oyd-timestamp": "2019-08-15T20:29:59Z"
}
```

Result. Only the "status" field has changed to "exist".

9.4.3. Request after the entry has been stored in the blockchain

Every day at 6 o'clock UTC, all entries of the notary service not previously stored in the blockchain are combined in a Merkle tree and the root node is stored in the blockchain. The mining fee is paid by a non-profit association[5].

```
$ curl https://blockchain.ownyourdata.eu/api/doc?hash
    =b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee
    9088f7ace2efcde9
{
  "status": "exist",
  "address": "0x761519431eb768f5a7d4cec38a28172ec68fc
    4cfa35c40f31cda603f4961cddc",
  "root-node": "5ca0442dfb2a277d6bbe2397994723fdfacc2
    f2002f8a143e7d77dc2bfe8b444",
  "audit-proof": "-d9fe724f791e98a2c2fdc4bbb21f5b357e
    38fbde228ada1a7dd5e5044b8610bf, -e663e58ce01a53bd
    dcea2760c0b8981118bff6094be23b6f4d63619ec48c8c15,
    -85eeb 4ac012e8fe6d977474a7735449cd924bd106dc07b9
    219a76b33df8b25b0,-970bbad0eb493c67f2c1298490fa69
    79f05874002c6401fcb9d248548d67c322,-e792a7cc91f9b
    c2314f449e45be663d0fd5a33e265ba3a521cd53efb0dc979
    33",
  "dlt-timestamp": "2019-08-16T06:00:08Z",
  "tsr": "MIIN...5cXA==",
  "tsr-timestamp": "2019-08-15T20:29:59Z",
  "oyd-timestamp": "2019-08-15T20:29:59Z
}
```

Result: by storing all entries in the blockchain, the following additional information becomes available:

1. address: address in the Ethereum blockchain where the root node was stored. The data at this address can read using free web services[6] the specified address can be read, for example https://etherscan.io/tx/0x761519431eb768f5a7d4cec38a28172ec68fc4cfa35c40f31cda603f4961cddc

[5] Verein zur Förderung der selbstständigen Nutzung von Daten, https://OwnYourData.eu Public charity to foster personal use of data

[6] for example: https://etherscan.io

2. `root-node`: the computed root node stored in the Ethereum Blockchain
3. `audit-proof`: the hash value together with the audit proof uniquely identifies the root node of the Merkel tree (as described in RFC 6962); however, it is important for each element to indicate whether it is the left (+) or right (-) node along the path, hence the +/- prefixes for each entry
4. `dlt-timestamp`: timestamp of the blockchain when the transaction was mined

To use the API features described here, there is also an easy to use web frontend[7]. Users can there calculate the hash value of a file via drag & drop and request to store it in the blockchain as well as retrieve a trusted timestamp. All necessary information will then be displayed (status, time stamp) or offered for download (TSR file and relevant certificates) as well as instructions for independent verification.

Summarizing, OwnYourData offers an open source and free service for the creation of independently verifiable data including time stamp. It offers both decentralized storage in a blockchain and legally binding processing via EU regulations. To cover the costs incurred, a time denomination of 1 day was selected for storage in the Ethereum blockchain.

10. Related Work

Semantic web technologies in general and ontologies in particular are increasingly being used for management of ATM information. We refer to Keller [26] for a survey on approaches applying semantic web technologies in ATM. A notable example of an ontology for ATM is the NASA ATM Ontology [27, 28].

The Semantic Container approach is closely related to semantic web services. Semantic annotation of web services renders the service descriptions machine-readable which, in turn, facilitates automatic discovery as well as composition of web services [29]. Among the types of semantics covered by web service descriptions are data semantics and functional semantics [30, p. 980]. The Semantic Container approach adopts a data-centric view towards web services: Semantic containers are the data products that are the output of web services, thus covering the data semantics of ATM information services. Unlike work on semantic web services, which is predominantly concerned with web service discovery, the presented approach for the description of data products explicitly considers distribution and maintenance of the data products after provisioning through the corresponding web services.

The FAA's Web Service Description Ontological Model (WSDOM) [31] allows for the semantic description of web service interfaces in the aeronautical domain. An extension of the WSDOM ontology for geospatial concepts [32] employs GeoSPARQL as representation format and query language for web service discovery. The WSDOM ontology and its extension for geospatial concepts are orthogonal to the Semantic Container approach. We do not focus on the web services as such but on the management and discovery of data sets. To this end, we introduce the notion of semantic containers and employ ontologies for the semantic description of container contents.

Related work [33] investigates the design and execution of web service workflows. In that context, metadata management has also been identified as an important topic [34]. Derivation chains of semantic containers provide a data-centric view on ATM information service workflows.

The concept of data mashups [35] is also related to semantic containers. Other work [36] propose the application of secure multiparty computation to build privacy-preserving data mashups.

The algebra of qualified relations [37], a well-established approach to distributed database management, served as a main inspiration for container versioning and consistency management in a distributed environment. We adapt the underlying concepts of the algebra for qualified relations for SWIM information services and extend the concept with semantic labels to support the management of containers, specifically the discovery but also the description of container lineage and provenance.

Zander and Schandl [38] propose to use "Semantic Web technologies to build comprehensive descriptions of user's information needs based on contextual information" and employ the descriptions to "selectively replicate data from external sources." Keeping local copies of relevant data on mobile devices, so that an application on the mobile device can operate also without network connectivity.

Replication of semantic containers is related to distribution and replication of (dynamic) XML documents [39, 40]. In comparison to existing approaches, semantic containers offer a unique combination of version management, distribution, replication, and fine-grained provenance tracking. Instead of relying on a single generic data model, e.g., XML, the Semantic Container approach uses different data models for different kinds of information – XML for data, RDF for metadata, OWL for semantic labels. In contrast to generic XML-based approaches, the Semantic Container approach leverages the specifics of ATM information exchange, with data items like NOTAMs and METARs constituting the lowest grain of fragmentation.

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

Literature on database replication distinguishes be-

[7] `https://notary.ownyourdata.eu`

tween eager and lazy replication [41]. Distributed semantic container management could follow both an eager and lazy replication approach, depending on the criticality of the data – for non-safety critical data, lazy replication may be preferable due to the lower replication costs. Lightweight approaches to versioning for database systems have also been proposed, e.g., OrpheusDB [42]. When datasets are collaboratively authored, version management is of importance and appropriate techniques for version management in the spirit of common version control systems must be developed [43].

11. 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 [24]. 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.

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

The proof-of-concept scenario has shown that the Semantic Container approach can extend the SWIM concept and add value to it by facilitating data discovery through semantic annotation, thus leveraging necessary benefits in SWIM networks. Since BEST was a TRL 1 project, future work will improve the semantic container concept and validate the SWIM integration in a comprehensive manner. This should include more scenarios, including data from an airline, an airport, and ANSPs and SWIM components such as the SESAR 2020 SWIM registry.

Information sharing via semantic containers builds on common and fine-grained ontologies providing commonly understood filter conditions. Shared rule sets realize filter functions of filter facets which can be provided as information filtering services. We expect the main effort with realizing the Semantic Container approach is with the development, maintenance, and governance of shared ontologies and rule sets. We expect fine-grained ontologies and rule sets to be developed and maintained by different communities. Faceted ontology-based information filtering allows to combine ontologies and rule sets in a modular way. This gives a high degree of flexibility supporting different strategies for development, maintenance, and governance of ontologies and rule sets.

Further details on the approach, including the full text of project deliverables (and summaries thereof), information about how to access technical results of the project (software and ontologies), and a short video explaining some technical details of parts of the work, are available on the project website (`https://project-best.eu/`).

Table 3: Acronyms

Acronym	Explanation
AIRM	ATM Information Reference Model
AIXM	Aeronautical Information Exchange Model
ANSP	Air Navigation Service Provider
ATM	Air Traffic Management
BEST	Archieving the BEnefits of SWIM by making smart use of Semantic Technologies – a SESAR project
DFS	Deutsche Flugsicherung – the ANSP for Germany
DLT	Distributed Ledger Technologies
ePIB	enhanced Pre-flight Information Bulletin
FAA	Federal Aviation Administration
FIR	Flight Information Region
FIXM	Flight Information Exchange Model
GDPR	General Data Protection Regulation
ICAO	International Civil Aviation Organization
IWXXM	ICAO Weather information Exchange Model
METAR	METeorological Aerodrome Reports
NOTAM	Notice to Airmen
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWL	Web Ontology Language
REST	Representational State Transfer
RDF	Resource Description Framework
RFC	Request for Comments
SCMS	Semantic Container Management System
SESAR	Single European Sky ATM Research
SPARQL	SPARQL Protocol and RDF Query Language
SWIM	System Wide Information Management
TAF	Terminal Aerodrome Forecast
TRL	Technology Readiness Level
UML	Unified Modeling Language
XML	Extensible Markup Language
XMI	XML Metadata Interchange
XSLT	Extensible Stylesheet Language Transformations

Acknowledgments

This research has received funding from the SESAR Joint Undertaking under grant agreement No 699298 under the European Union's Horizon 2020 research and innovation program. The views expressed in this paper are those of the authors.

This research has received funding from the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit) under grant number 869781 under program "IKT der Zukunft".

References

[1] Aeronautical Information Exchange Model 5.1.1, 2016. http://www.aixm.aero/

[2] Flight Information Exchange Model 4.1.0 2017. http://www.fixm.aero

[3] ICAO Weather Information Exchange Model 2.1.1, 2017. https://github.com/wmo-im/iwxxm

[4] F. Burgstaller, D. Steiner, B. Neumayr, M. Schrefl, E. Gringinger, "Using a model-driven, knowledge-based approach to cope with complexity in filtering of notices to airmen," Proceedings of the Australasian Computer Science Week 2016, ACM 2016.

[5] S. Ceri, G. Pelagatti, "Correctness of Query Execution Strategies in Distributed Databases," ACM Transactions on Database Systems (TODS), 8(4) 1983: 577-607.

[6] A. Vennesland, J. Gorman, BEST D6.3 Final Report, 2018. http://www.project-best.eu/publications.html

[7] ATM Information Reference Model 4.1.0, 2017. http://www.airm.aero

[8] E. Gringinger, C. Fabianek, C. G. Schuetz, B. Neumayr, M. Schrefl, A. Vennesland, S. Wilson, "The Semantic Container approach: Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base," In: Proceedings of the SESAR Innovation Days 2018 (SIDs 2018) https://www.sesarju.eu/sites/default/files/documents/sid/2018/papers/SIDs_2018_paper_78.pdf.

[9] A. Vennesland, B. Neumayr and C. Schuetz, "BEST D1.1 Experimental ontology modules formalising concept definition of ATM data," 2017. http://www.project-best.eu/publications.html

[10] E. Gringinger, C. Fabianek, C. G. Schuetz, "BEST D3.2 Prototype SWIM-enabled Applications," 2018. http://www.project-best.eu/publications.html

[11] M. A. Musen, Protégé Ontology Editor, 2015. http://protege.stanford.edu/

[12] A. Vennesland, E. Gringinger and A. Kocsis "BEST D5.2 Ontology Modularisation Guidelines," 2017. http://www.project-best.eu/publications.html

[13] M. D'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou, "Criteria and evaluation for ontology modularization techniques," in Modular ontologies, 2009, pp. 67–89.

[14] A. Schlicht and H. Stuckenschmidt, "Towards structural criteria for ontology modularization," in Proceedings of the 1st International Conference on Modular Ontologies-Volume 232, 2006, pp. 85–97.

[15] B. C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur, "Modularity and Web Ontologies," in KR, 2006, pp. 198–209.

[16] B. Neumayr, E. Gringinger, C. G. Schuetz, M. Schrefl, S. Wilson and A. Vennesland, "Semantic data containers for realizing the full potential of system wide information management," IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), St. Petersburg, FL, USA, 2017, pp. 1-10.

[17] E. Gringinger, C. G. Schuetz, B. Neumayr, M. Schrefl and S. Wilson, "Towards a value-added information layer for SWIM: The Semantic Container approach," IEEE 18th Integrated Communications Navigation Surveillance Conference (ICNS), Herndon, VA, USA, 2018, pp. 3G1-1-3G1-14.

[18] C. Schuetz, B. Neumayr, M. Schrefl and E. Gringinger, "D2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base," 2018. http://www.project-best.eu/publications.html

[19] E. Gringinger, C. Fabianek, B. Neumayr and A. Savulov, "BEST D3.1 Prototype Use Case Scenarios," 2018. http://www.project-best.eu/publications.html

[20] C. Schuetz, B. Neumayr, M. Schrefl and E. Gringinger, "BEST D2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment," 2018. http://www.project-best.eu/publications.html

[21] A. S. Tanenbaum and M. van Steen, Distributed systems: Principles and paradigms: Prentice-Hall, 2007.

[22] M. T. Özsu and P. Valduriez, Principles of distributed database systems: Springer Science & Business Media, 2011.

[23] C. G. Schuetz, B. Neumayr, M. Schrefl, E. Gringinger and S. Wilson, "Semantics-Based Summarization of ATM Information to Manage Information Overload in Pilot Briefings," 31st Congress of the International Council of Aeronautical Sciences (ICAS), Belo Horizonte, Brazil, 2018.

[24] A. Vennesland, J. Gorman, S. Wilson, B. Neumayr and C. G. Schuetz, "Automated Compliance Verification in ATM using Principles from Ontology Matching," 10th International Conference on Knowledge Engineering and Ontology Development (KEOD), Seville. Spain, 2018

[25] B. Schneier, Applied Cryptography, 2nd ed.: John Wiley and Sons, 1996.

[26] R. M. Keller, Ontologies for aviation data management, in: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), 2016 (2016). doi:10.1109/DASC.2016.7777971.

[27] R. M. Keller, The NASA Air Traffic Management Ontology, https://data.nasa.gov/ontologies/atmonto (March 2018).

[28] R. M. Keller, Building a knowledge graph for the air traffic management community, in: Companion of The 2019 World Wide Web Conference, 2019, pp. 700–704 (2019).

[29] S. A. McIlraith, T. C. Son, Honglei Zeng, Semantic web services, IEEE Intelligent Systems 16 (2) (2001) 46–53 (2001).

[30] J. Domingue, D. Fensel, J. A. Hendler, Handbook of semantic web technologies, Springer, 2011 (2011).

[31] M. Kaplun, Faa web service description ontological model (wsdom) – an introduction, https://www.faa.gov/air_traffic/technology/swim/governance/service_semantics/media/FAA%20WSDOM%20Introduction.pdf (Accessed: 16 October 2019).

[32] A. Balaban, Testbed-12 aviation semantics engineering report, Tech. rep., http://docs.opengeospatial.org/per/16-039.html (Accessed: 16 October 2019.

[33] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, C. Goble, The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud, Nucleic Acids Research 41 (W1) (2013) W557–W561 (2013). doi:10.1093/nar/gkt328.

[34] K. Belhajjame, K. Wolstencroft, O. Corcho, T. Oinn, F. Tanoh, A. William, C. Goble, Metadata management in the taverna workflow system, in: 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008, pp. 651–656 (2008). doi:10.1109/CCGRID.2008.17.

[35] G. Di Lorenzo, H. Hacid, H.-y. Paik, B. Benatallah, Data integration in mashups, SIGMOD Records 38 (1) (2009) 59–66 (2009). doi:10.1145/1558334.1558343.

[36] N. Mohammed, B. C. M. Fung, K. Wang, P. C. K. Hung, Privacy-preserving data mashup, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT 2009, 2009, pp. 228–239 (2009). doi:10.1145/1516360.1516388.

[37] S. Ceri, G. Pelagatti, Distributed Data Bases: Principles and Systems.

[38] S. Zander, B. Schandl, Context-driven rdf data replication on mobile devices, Semantic Web 3 (2) (2012) 131–155 (2012).

[39] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, T. Milo, Dynamic xml documents with distribution and replication, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD '03, 2003, pp. 527–538

(2003). doi:10.1145/872757.872821.

[40] G. Koloniari, E. Pitoura, Peer-to-peer management of xml data: Issues and research challenges, SIGMOD Records 34 (2) (2005) 6–17 (2005). doi:10.1145/1083784.1083788.

[41] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, Database replication techniques: a three parameter classification, in: Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000, 2000, pp. 206–215 (2000). doi:10.1109/RELDI.2000.885408.

[42] L. Xu, S. Huang, S. Hui, A. J. Elmore, A. Parameswaran, Orpheusdb: A lightweight approach to relational dataset versioning, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, 2017, pp. 1655–1658 (2017). doi:10.1145/3035918.3058744.

[43] A. P. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, A. G. Parameswaran, Datahub: Collaborative data science & dataset version management at scale, CoRR abs/1409.0798 (2014).
URL `http://arxiv.org/abs/1409.0798`