Scalability Guidelines for Semantic SWIM-based Applications

Deliverable 5.1

BEST

Grant:	699298
Call:	H2020-SESAR-2015-1
Topic:	Sesar-03-2015 Information Management in ATM
Consortium coordinator:	SINTEF
Dissemination Level:	PU
Edition date:	[April 27 2018]
Edition:	[01.01.00]

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Gunnar Brataas (SINTEF)	Project member	
Bernd Neumayr	Project member	
Christoph Schuetz	Project member	
Audun Vennesland (SINTEF)	Project member	

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Scott Wilson	Project member	21.09.2018
Scott Wilson	Project member	09.01.2018
Scott Wilson	Project member	27.04.2018

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

Name/Beneficiary	Position/Title	Date
Approved by consortium in accordance with procedures defined in Project Handbook.	All partners	Thu 27.04.2018

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date

Document History

Edition	Date	Status	Author	Justification
00.00.01	13.09.17	Document created	Gunnar Brataas	
00.00.01	21.09.17	PCOS Approved	Gunnar Brataas	
00.50.00	09.01.18	Intermediate Proposed	Gunnar Brataas, Bernd Neumayr, Christoph Schuetz, Audun Vennesland	

00.51.00	10.01.2018	Intermediate Approved	Gunnar Brataas, Bernd Neumayr, Christoph Schuetz, Audun Vennesland	Approval given by internal reviewer after modifications made
00.90.00	21.03.2018	External proposed	Gunnar Brataas, Bernd Neumayr, Christoph Schuetz, Audun Vennesland	
00.91.00	23.03.2018	External approved	Gunnar Brataas, Bernd Neumayr, Christoph Schuetz, Audun Vennesland	
01.00.00	29.03.2018	Released	Joe Gorman	Formal changes about document edition, approval etc.
01.01.00	27.04.2018	External proposed	Gunnar Brataas, Bernd Neumayr, Christoph Schuetz, Audun Vennesland	Clarification of scalability guidelines addressing SJU comments. For re-submission to SJU.

BEST

Achieving the BEnefits of SWIM by making smart use of Semantic Technologies

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

Abstract/Executive Summary

Semantic container management is a promising approach to organize data. However, the scalability of this approach is challenging. By scalability in this deliverable, we mean the expressivity and size of the semantic containers we can handle, given a suitable quality threshold. In this deliverable, we derive scalability characteristics of the semantic container approach in a structured way. We also describe actual experiments where we vary the number of available CPU cores and quality thresholds. Based on these experiments we propose scalability guidelines for the semantic container approach. Moreover, based on the work in D2.1 we propose guidelines on how to apply semantic technologies in a scalable way.

The real contribution of this deliverable is the proposal of a framework that allows for the development of more scalability guidelines. In this deliverable, this scalability framework for semantic container management allows us to make measurements, which could be formulated as initial guidelines. These measurements can be extended in the future for producing a mature set of guidelines, for example to guide the choice of specific semantic technologies to realize the semantic container approach.

In this deliverable the measurements result in two guidelines. When also considering input from D2.1, four more scalability guidelines are presented.

This deliverable quotes material from an accepted paper. This paper was presented at the International Conference on Performance Engineering (ICPE) 9-13 April 2018 in Berlin.

Table of Contents

1 Introduction: About this document[1]

1.1 Purpose

Semantic technologies are used to perform powerful reasoning on complex information. It is essential that the semantic technologies can still reach a minimum quality threshold as the size and complexity of the information grows. This document makes an assessment on the scalability of the semantic technologies used in BEST, and is related to objective 2 in the DoW: *How can we ensure that ATM solutions developed using semantic technologies have scalability characteristics that allow them to be used successfully even when data volumes, complexity and load increase?*

In this deliverable we focus on assessing the scalability of the semantic container approach, because scalability here is more challenging compared to the scalability of semantic filtering.

According to the Grant Agreement: *This deliverable will describe a set of guidelines on how applications using semantic technology can be applied in the ATM domain with good scalability characteristics.* This deliverable provides guidelines at the end of Section 5.2. The semantic container approach as proposed in the context of this exploratory research project is not yet mature enough to make definitive scalability guidelines. In this regard, the real contribution of this deliverable is the proposal of a framework that allows for the development of more scalability guidelines. In this deliverable, this scalability framework for semantic container management allows us to make initial measurements, which could be formulated as initial guidelines. These measurements can be extended in the future for producing a mature set of guidelines, for example to guide the choice of specific semantic technologies to realize the semantic container approach.

From Section 2 to Section 7, inclusive, this deliverable directly quotes material from an accepted paper [3], which was presented at the International Conference on Performance Engineering (ICPE) in Berlin April 9 – 13, 2018.

This accepted paper is an open access paper which means that the copyright is held by the BEST project and the authors.

There are some differences between the paper and this deliverable because of different audiences, the most important of which are:

- Section 2 does not include some material which was present in the paper corresponding to explaining the semantic container approach. In the context of BEST this is better done by a reference to D2.1.
- Some lines of text in Section 4 have been added in this deliverable explaining scalability which was not required in a paper targeted to a research community, in the performance and scalability of software and hardware systems.
- In this deliverable we propose some guidelines at the end of Section 6 which we did not manage to get into the paper before the print deadline.

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

- The conclusion and further work in Section 7 is also reformulated based on different audiences as well as adapted as a result of more refinements in this deliverable in Section 5.2.
- More work has been performed on scalability guidelines. Therefore Section 6 is new in this deliverable compared to the ICPE-paper.

1.2 Intended Readership

This document is primarily targeted towards people having an interest in

- Scalability of application of semantic technologies in ATM

1.3 Relationship to other deliverables

Deliverable	Relationship
D 1.1 Experimental ontology modules formalizing concept definition of ATM data	We will not consider scalability of the ontologies per se, as we are interested in how the ontologies are used in practice. As a result, there is no direct relationship to D1.1. We refer to D2.1 for ontology concepts. Indirectly, we will refer to D1.1, since it is referenced by D2.1.
D 2.1 Techniques for ontology-based data description and discovery in a decentralized SWIM knowledge base	Describes the semantic container approach. In this deliverable, we will investigate scalability aspects of the semantic container approach.
D 2.2 Ontology-based techniques for data distribution and consistency management in a SWIM environment	Consistency requirements are related to scalability. Practical issues like distribution of containers across different nodes are also relevant.
D 3.1 Prototype Use Case Scenarios	The scenarios described in D 3.1 provide the scope for the scalability investigation.
D 3.2 Prototype SWIM-enabled applications	The prototype applications in D 3.2 will give us practical insight into scalability.
D 4.4 Tutorial for Software Developers	The scalability guidelines will give software developers insight into applying semantic technologies in their software developments.

1.4 Acronyms and abbreviations

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

2 Introduction to the work

Semantic technologies help to create and manage conceptual models -- also referred to as ontologies -- and to apply conceptual models in large-scale and decentralized information systems in order to foster a common understanding of data and metadata.

BEST has introduced the semantic container approach as an ontology-based approach to organize data sets and to automate the discovery of data sets that fulfil a particular information need [9], see D2.1 for details. There is a concern that the computational complexity of semantic reasoning may lead to poor scalability of the semantic container approach.

We refer to scalability as a system's ability to increase the capacity by consuming more hardware and software resources [1]. The capacity is the highest workload fulfilling the quality thresholds (e.g. response times). Scalability analysis then investigates the scalability implications of higher load (more traffic), more work (computationally harder operations and/or more data), and stricter quality thresholds (e.g. shorter response times). In this context, we refer to scalability implications as the amount of additional hardware (CPUs with cores, primary and secondary memory, network capacity) and software resources (software licences) that are required to handle increasing amounts of load or work. For example, if doubling the work requires a tenfold increase of underlying resources to keep within service level agreements, the system does not scale. Scalability problems are even worse if a system is not able to handle an increase in load regardless the amount of additional hardware or software resources. Such scalability problems should be identified early in the development process in order to discover the sources of the problems as well as possible solutions. Before depending on a system architecture, it is therefore important to know its scalability implications.

3 State of the Art

This section gives a short introduction to the state of the art in scalability of semantic technologies. The Web Ontology Language (OWL 2), which D2.1 considers as a possible ontology language for the semantic container approach, is a standard ontology language for the (semantic) web, and OWL 2 DL is the most expressive decidable subset of the full language [6]. Further considerations on reasoning performance of OWL 2 ontologies led to the definition of a set of sublanguages -- the OWL 2 profiles [7] -- with reduced expressivity but also lower time complexity of common reasoning tasks when compared to OWL 2 DL. For example, the OWL 2 EL profile, which is sufficient in many practical situations, where the ELK reasoner [8] serves as an efficient implementation. Reasoning performance in practice also depends on the characteristics of the ontology.

Ensuring the scalability of reasoning tasks has been identified as a major challenge for implementing real-world applications using semantic technologies due to an "inherent trade-off between the expressivity of a logical representation language and scalability of reasoning" [5, p.524]. Parallelization of reasoning tasks is a common strategy to ensure scalability. In this deliverable, we investigate reasoning capacity for semantic container management with the ELK off-the-shelf reasoner depending on the number of available processors (CPU cores) on a single machine.

4 Scalability Requirements

Scalability is characterized by the general concepts shown in Figure 1. In this section, we apply the general scalability concepts to the semantic container approach. This means that some concepts, such as configuration parameters, will not be discussed further. For other concepts the relation between the figure and the text is complex, e.g. for system, described in the next sub section, which refers to services and resources in the figure.

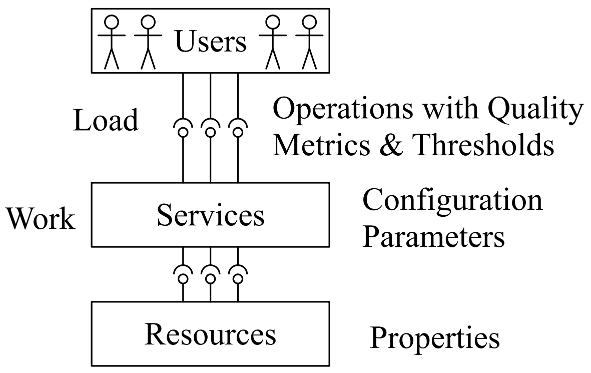

Figure 1 Scalability concepts

4.1 System

We use the term “system” to encompasses services and resources. When analysing scalability, we must define which services (and resources) are inside and outside of our system boundary. These boundaries also define how response times are measured. In this deliverable, we focus on reasoning about metadata which serves for adding/updating and querying containers. We do not investigate the population of the containers with actual data since the population of semantic containers highly depends on the specific application scenario. For example, a semantic container for Notices to Airmen may be filled with actual data using prioritization and filtering rules [2] specific to a particular airline which might be much more complex than the rules employed by another airline or the rules for another type of data such as weather forecasts. Reasoning about the metadata of containers, on the other hand, is less dependent on the specific application scenario.

4.2 Operations

An operation defines a unique and relatively similar way in which a user interacts with a service; an operation corresponds to a request class in the context of queuing networks. An operation has quality metrics and thresholds as illustrated in **Figure 1** and explained in 4.5.

For the system described in Section 4.1 for reasoning about metadata, the following three operations are essential, i.e., must be present in order to productively employ a semantic container management system: Make a subsumption hierarchy, extend the subsumption hierarchy, and find individual semantic containers. We elaborate these operations in the following sections.

4.2.1 Initialize: Make a Subsumption Hierarchy

The subsumption (or generalization) hierarchy of semantic containers serves as an index for the retrieval of semantic containers. A more general (or broader) semantic container subsumes a more specific (or narrower) semantic container. For example, a semantic container with Notices to Airmen (NOTAMs) for the European airspace subsumes a semantic container with only the NOTAMs for the route from Vienna to Frankfurt.

The subsumption hierarchy derives from faceted membership conditions; a membership condition refers to one value for each facet such as geographic area and temporal scope. For example, a semantic container with NOTAMs relevant for heavy-wake aircraft on the route from Vienna to Frankfurt on the 13 January 2018 has three facets: geography, temporal, and aircraft. For each of these facets, the semantic container refers to one concept from the corresponding ontology: a concept *RouteVIE-FRA* from a geography ontology, a concept *13-01-2018* from a temporal ontology, and a concept *HeavyWakeAircraft* from an aircraft ontology.

Making the subsumption hierarchy of semantic containers is a two-step process. First, the reasoner derives concept hierarchies for the ontologies used for the facets of the semantic container description. For example, the reasoner determines that *SuperHeavyWakeAircraft* is more specific than *HeavyWakeAircraft* and that *A380* is more specific than *SuperHeavyWakeAircraft*. For each facet ontology, the concept hierarchy may come from a separate reasoner, or be asserted.By “asserted” we mean predefined for that ontology rather than inferred through logical properties. Then, having the subsumption hierarchy for each facet ontology, the reasoner determines the subsumption hierarchy of semantic containers, which have a reference to one concept for each facet. For example, a semantic container has a value *HeavyWakeAircraft* for the aircraft facet and *January-2018* for the temporal facet. Another semantic container has facet values *Aircraft* and *2018*; the latter container subsumes the former.

4.2.2 Add Semantic Container: Extend the Subsumption Hierarchy

The subsumption hierarchy of semantic containers needs to be updated to accommodate new semantic containers. The employed reasoning algorithm works only for a certain complexity of the semantic container descriptions: The more expressive the ontology, the more complex the expressed semantic container descriptions, the more complex the reasoning algorithm. For simple ontologies, we can do incremental reasoning, whereas for more complex ontologies we may have to make the subsumption hierarchy from scratch. In this deliverable, we employ the ELK reasoner to study scalability of the semantic container approach. The ELK reasoner is capable of incremental reasoning. Adding or removing an axiom does not necessitate a full recalculation of the subsumption hierarchy.

4.2.3 Find Individual Semantic Containers

Different semantic containers fulfil different information needs. Technically, an information need is represented by a membership condition in the same ontology language as the subsumption hierarchy of semantic container. The task of finding semantic containers that satisfy a given information need is referred to as semantic container discovery. In this deliverable, we analyse scalability of semantic container discovery using OWL EL class expressions (see Section 5.2, where we describe experiments about making the subsumption hierarchy, which is the first step in semantic container discovery) for semantic container descriptions, which are also considered as a possibility in D 2.1. In that scenario, the user asks the reasoner for direct subsumers of an information need that is expressed as an ontology concept.

4.3 Work

Work characterizes the amount of data to be processed, stored or communicated when invoking one operation. Ultimately, work characterizes the amount of hardware resources consumed when invoking one operation. The set of operations is of course an important part of the characterization of work. In addition, when considering scalability, we are also interested in how the work for one operation varies. This variation is connected to sizes of relevant objects, e.g., the number of documents and their average size; such parameters are referred to as work parameters. For scalability, the highest values of the work parameters are most relevant. Whereas load typically goes up and down during the day, week and month in complex patterns, work parameters are simpler as they typically only increase in value. For operations which encompass ontological reasoning, the most relevant work parameters are ontology expressivity and ontology size.

4.3.1 Ontology Expressivity

Ontology expressivity concerns the complexity of the axioms in the ontology and thus the complexity of automatic reasoning. A more expressive ontology language allows to describe more precisely the contents of a semantic container as well as information needs. This often comes with high computational costs. Ontology language profiles restrict the expressivity of ontologies in order to allow for more efficient reasoning. The OWL EL ontology language profile disallows, for example, the use of the OR operator in class expressions. In this deliverable, we consider the use of OWL EL, although an actual implementation might use a technology mix as explained in D2.1. In our experiments, we further restrict the expressivity to class hierarchies (subclassOf axioms) and to class definitions with intersectionOf class expressions.

4.3.2 Ontology Size

In the context of semantic container management, the work parameter of ontology size can be further broken down into the following work sub-parameters:

- Number of containers. The actual size of the container is not relevant, since we are working with metadata.
- Number of facets: A facet is a property of all the data items in the container, for example location and time. The complexity of the container hierarchy is determined by the complexity of the facet hierarchy.
- Number of classes per facet: For a spatial facet the number of locations (represented by bounding boxes) will determine the number of classes.
- Depth and complexity of facet hierarchy: The classes of each facet will form a hierarchy -- a tree or a (semi-) lattice -- possibly at different levels, where the number of levels is the depth of the hierarchy.

4.4 Load

Load is how often an operation is invoked. The term load refers to the frequency of invocation of an operation by its users. In this deliverable, we focus on work and leave the analysis of the influence on load to future work. Load is most important in the case of finding semantic containers since that operation is more frequently invoked in day-to-day work than making the subsumption hierarchy.

4.5 Quality Metrics and Thresholds

A quality metric defines how we measure a certain quality and is a key part of an SLA (Service Level Agreement). At an overall level, response times and throughput are traditional scalability quality metrics. Quality thresholds (QTs) describe the border between acceptable and non-acceptable quality for each operation.

Quality thresholds in our domain are measured in 90 percentile response times, since in this way outliers will not affect the capacity. Quality thresholds will be measured in seconds and hours. We can tolerate a longer time for making the subsumption hierarchy than extending the subsumption hierarchy. Finding data containers should be done even faster.

4.6 Resources and Capacity

Figure 1 illustrates that resources have properties. We have active as well as passive resources. Active resources are hardware for processing (CPUs with CPU cores), storage (primary memory (RAM), flash memories, disks) and communication (network). Passive resources represent semaphores, buffers and pools, typically associated with storage. When considering scalability, passive resources are crucial, and not surprisingly, storage often represent scalability limitations. The cost of software licenses may also be important.

The property deemed most crucial in analysing the scalability of the semantic container approach was the capacity of the resources. The highest workload fulfilling quality thresholds is the capacity of a system. In our case we want to vary one work parameter. To get one single number of capacity, we must fix the remaining work parameters as well as quality metrics and quality thresholds. Then, the highest work parameter for the average operation which fulfils the quality thresholds, becomes the capacity.

5 Experiments

We conducted experiments for the operation “make a subsumption hierarchy”. We were interested in how the capacity varied with the number of CPU cores (varied from 1 to 16 cores) and a given quality threshold. We employed ’90 percentile response time' as our quality metric and varied the quality threshold between 0.5 sec, 5 sec and 20 seconds. Capacity is the number of semantic containers for which the subsumption hierarchy can be derived within the time given by the quality threshold.

5.1 Setup

As our hardware configuration, we used a Sun Fire X4600 (a machine from year 2008) with 8 CPUs (16 cores) of type AMD Dual-Core Operton 885 2.6 GHz and with 58 GB of RAM.

As our software configuration, we use Linux (CentOS Release 6.9), Java (JRE 8 Update 151), ELK reasoner (Version 0.4.3) and a custom-made Java program which is invoked with a maximum heap size of 50 GB.

A shell script turns on and off cores (to vary the number of cores from 1 to 16) and invokes the Java program with different quality thresholds (0.5, 5, and 20 seconds).

To find the capacity for a given number of cores and a given quality threshold, a custom-made Java program performs binary search. In the binary search, for a number of semantic containers (the tested capacity), the "make a subsumption hierarchy" operation was executed up to ten times allowing one run exceeding the time limit (0.5 sec, 5 sec, or 20 sec). To save experimentation time, the binary search was stopped when an additional round of ten runs would affect the resulting capacity by less than 10 percent.

For each run, the Java program generates an OWL EL ontology according to given work parameters and performs subsumption reasoning for this ontology. In this experiment, we varied the number of semantic containers during the binary search and fixed the other parameters as follows: 3 facets with a hierarchy depth (number of levels) of 5, and 3 children per parent (this gives a tree of 364 classes per facet).

5.2 Results

Our results are shown in **Figure 2**. The x-axis is the number of cores while the y-axis is the highest number of containers where the quality thresholds are still obeyed. We measure this for 0.5 sec, 5 sec and 20 sec.

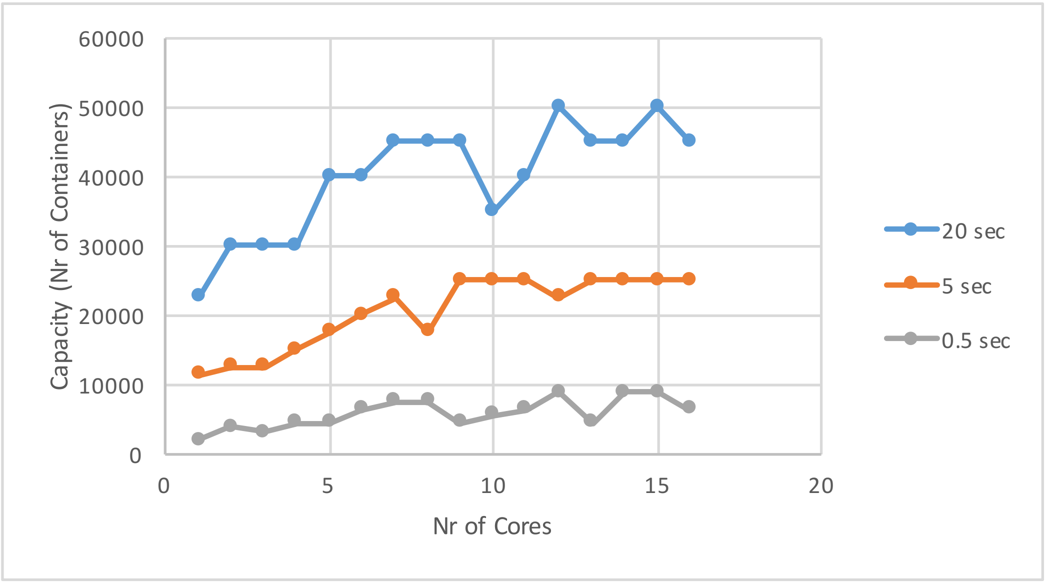

Figure 2 Capacity measured in number of containers with 0.5 sec, 5 sec and 20 sec quality thresholds

Based on the measurements in **Figure 2**, we make the following observations:

1. With a 40-times increase in the quality threshold, the capacity increases from 9000 to 50000 containers, i.e. approximately a six-time increase.
2. When it comes to the number of cores it is not easy to draw any conclusions based on these measurements, except that until approximately 7 cores the capacity increases.
3. A 7-time increase in the number of cores does not lead to a 7-time increase in capacity, especially with the 20-second quality threshold. With a 20-second quality threshold a 7-time increase in the number of cores results only in a doubling of capacity.
4. After 7 cores the capacity does not seem to increase anymore, i.e., the system does not scale.

The trend observed in **Figure 2** can have two justifications:

1. The trend in observation 1 is not surprising since the number of possible subsumption relationships between containers is roughly quadratic in the number of semantic containers. The ELK reasoner works as a black-box and makes optimizations we do not know about in detail, but $(50 / 9)^2 = 31$ which is comparable to 40.
2. The reason for the lack of scalability with respect to the number of cores, mentioned in Points 2-5 above, is as follows: You cannot just split a large ontology into 7 parts, do the reasoning independently on each part, and then just put the results together. The 7 parts are not independent from each other. The more parts into which you split an ontology, the more communication will be required between the different parts.

These two observations are converted into two guidelines in the next section.

6 Guidelines for Scalability

In this deliverable, we have developed conceptually consistent requirements for scalability of semantic containers. Experiments based directly on these requirements lead us to define the following two guidelines:

1) **Response time not linear with number of containers**: Increasing the number of semantic containers with a factor of X will lead to an increase in execution time with a factor of roughly X^2.
2) **Limited CPU core scalability:** With a small number of cores it is easy to get parallelization benefits, but probably not proportional to the number of cores. However, if more cores are added then the effect may be marginal.

A weakness with the these two guidelines is that they are valid only within the context of our measurements. More work has to be done to make more robust guidelines. This is discussed in the next section.

Based on work done in D2.1, the following additional guidelines for scalability can be formulated:

3) **Use a technology mix** (there is no one-size-fits-all in semantic technologies): Make careful decisions about what technologies (F-Logic, OWL, RDF, XML) to employ. Different semantic technologies have very different scalability characteristics. RDF and SPARQL are highly scalable, and can for example be implemented on top of Apache Spark clusters [10], but are not expressive enough for semantic container reasoning. OWL and ontology reasoning is more problematic as explained in guidelines 1 and 2.
4) **Use External/Specialized Reasoners** (for some parts of the ontology): For example, OWL reasoners are not suited for containment checking of GML shapes; it is better to use external reasoners to derive subsumption hierarchies of GML shapes and materialize the results.
5) **Limit ontology expressivity:** The OWL EL ontology language profile, for example, limits the expressivity of OWL. The ELK reasoner is an efficient automatic reasoner for this language profile which allows efficient reasoning over 100,000s of classes.
6) **Limit Ontology Size** (by MODULARIZATION): Runtime performance of automatic reasoners is acceptable as long as reasoning is performed separately over rather small ontology modules with the results being combined. This relates to guideline 1. Ontology size can also be reduced by only representing some of the data/metadata as part of the OWL ontology and keeping other parts (which do not need ontology reasoning) in RDF and/or XML.
7) **Limit ontology reasoning at run time:** Most of the automatic reasoning can be done when developing/maintaining the ontology with the results being materialized.

7 Conclusions and Further Work

In this deliverable, we have described how structured scalability analysis techniques can be applied to semantic technologies. We have also described the experiments that we conducted, and the set of measurements we were able to extract, within the resource constraints of the project. Based on these experiments we produced a set of basic guidelines. More experimental results would be needed in order to provide more robust guidelines.

Within the scope of the current measurements, the precision of the current experiments could be improve by replicating the measurements so that reasonable confidence intervals could be established. In our measurements, we used quality thresholds in the order of seconds. With a higher and more realistic quality threshold, in the order of hours, the time to do these measurements would increase, but then the value of these experiments could also be higher.

To provide more robust guidelines we would have to extend the scope of investigation to include, for example, different reasoning algorithms. Then, we should be able explore how different semantic technologies with different ontology expressivity affect the ontology sizes (described in Section 4.3.2) we are able to handle. This is important because the more expressive the ontology is, the more precisely we can describe the content of the semantic container.

8 References

[1] Gunnar Brataas and Tor Erlend Fægri. 2017. Agile Scalability Requirements. In ICPE, Conf. on Performance Eng. ACM.

[2] Gunnar Brataas, Nikolas Herbst, Simon Ivansek, and Jure Polutnik. 2017. Scalability Analysis of Cloud Software Services. In 2017 IEEE International Conference on Autonomic Computing (ICAC). IEEE, 285–292.

[3] Gunnar Brataas, Bernd Neumayr, Christoph G. Schuetz, Audun Vennesland. 2018. Toward Scalability Guidelines for Semantic Data Container Management. In ICPE, Conf. on Performance Eng. ACM.

[4] Felix Burgstaller, Dieter Steiner, Bernd Neumayr, Michael Schre , and Eduard Gringinger. 2016. Using a model-driven, Knowledge-based approach to cope with complexity in Filtering of Notices to Airmen. In Proceedings of the Australasian Computer Science Week Multiconference (ACSW 2016). 46:1–46:10.

[5] Dieter Fensel, Frank van Harmelen, Bo Andersson, Paul Brennan, Hamish Cunningham, Emanuele Della Valle, Florian Fischer, Zhisheng Huang, Atanas Kiryakov, Tony Kyung-il Lee, Lael Schooler, Volker Tresp, Stefan Wesner, Michael Witbrock, and Ning Zhong. 2008. Towards LarKC: a platform for web-scale reasoning. In Proceedings of the 2nd IEEE International Conference on Semantic Computing. 524–529.

[6] Pascal Hitzler, Peter Patel-Schneider, Sebastian Rudolph,Markus Krötzsch,and Bijan Parsia. 2012. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation. W3C. http://www.w3.org/TR/2012/REC-owl2-primer- 20121211/.

[7] Ian Horrocks, Zhe Wu, Achille Fokoue, Boris Motik, and Bernardo Cuenca Grau. 2012. OWL 2 Web Ontology Language Pro les (Second Edition). W3C Recommendation. W3C. http://www.w3.org/TR/2012/REC-owl2-pro les-20121211/.

[8] Yevgeny Kazakov, Markus Krötzsc, and Frantisek Simancik.2014.The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with EL Ontologies. J. Autom. Reasoning 53, 1 (2014), 1–61.

[9] Bernd Neumayr, Eduard Gringinger, Christoph Schuetz, Michael Schre, Scott Wilson, and Audun Vennesland. 2017. Semantic Data Containers for Realizing the Full Potential of System Wide Information Management. In Proceedings of the IEEE/AIAA 36th Digital Avionics Systems Conference.

[10] Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, Georg Lausen: S2RDF: RDF Querying with SPARQL on Spark. PVLDB 9(10): 804-815 (2016)

The BEST consortium:

SINTEF	SINTEF
Frequentis AG	FREQUENTIS
Johannes Kepler Universität (JKU) Linz	JKU JOHANNES KEPLER UNIVERSITY LINZ
SLOT Consulting	Slot onsulting
EUROCONTROL	EUROCONTROL