OGC® DOCUMENT: 23-050

External identifier of this OGC® document: http://www.opengis.net/doc/PER/T19-D021

OGC TESTBED-19 AGILE REFERENCE ARCHITECTURE ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2023-12-06
Approval Date: 2024-01-25
Publication Date: 2024-04-26
Editor: Lucio Colaiacomo

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

I EXECUTIVE SUMMARY

The concepts of agile architecture and reference architecture may not be new ideas in information or geospatial technologies, but what is meant by the term Agile Reference Architecture?

Agile Reference Architecture is the long-term vision of the complex and changing nature of how problems will be solved in the future within the location-referenced and geospatial realms. This includes consideration of network availability, as containers integrated with Linked Data, and Application Programming Interfaces (APIs) serve data as secure, trusted, and self-describing resources.

While the Open Geospatial Consortium (OGC) focuses on geospatial information and technologies, that community is also dependent on the overall state of information and communications technology (ICT), including developing cyber, cryptographic, and internet technologies.

In today's infrastructures, the collection, exchange, and continuous processing of geospatial resources typically happens at pre-defined network endpoints of a spatial data infrastructure. Each participating operator hosts some capability at a network endpoint. Whereas some network operator endpoints may provide data access, other endpoints provide processing functionality and other endpoints may support the uploading of capabilities. In other words, such an infrastructure is not agile in the sense that it cannot adapt by itself to meet the needs of the moment. One of the biggest challenges resulting from the static characteristics is ensuring effective and efficient operations of the overall system and at the same time maintaining trust and provenance.

This OGC Testbed 19 Engineering Report (ER) outlines novel concepts for establishing a federated agile infrastructure of collaborative trusted systems (FACTS) that is capable of acting autonomously to ensure fit-for-purpose cooperation across the entire system. One of the key objectives is to not create a new data product, but instead a collaborative object is offered leveraging FACTS that allows for obtaining the data product via well-defined interfaces and functions provided by the collaborative object.

Trust and assurance are two key aspects when operating a network of collaborative objects leveraging STANAG 4774/4778. STANAG 4774 outlines the metadata syntax required for a confidentiality label to better facilitate and protect sensitive information sharing. In addition, STANAG 4778 defines how a confidentiality label is bound to the data throughout its lifecycle and between the sharing parties.The agile aspect is achieved by the object's ability to activate, deactivate, and order well-defined capabilities from other objects. These capabilities are encapsulated in building blocks. Each building block is well defined in terms of accessibility, functionality, and ordering options. This allows building blocks to roam around collaborative objects as needed to ensure a well-balanced network load and suitable processing power of individual nodes from the network.

Equally trusted partners in the infrastructure participate in FACTS. They can collect data from other partners and create derived products via collaborative objects. The sharing of data products is only possible directly, meaning direct communication with data consumer and it is only possible via the objects. This guarantees that fundamental trust operations are applied

to the data and provenance records are produced before the data product is made available to others. The use of Blockchain technology and Smart contracts is one example of how this fundamental behavior can be planted into collaborative objects. As in trusted networks that are using Evaluation Assurance Level (EAL) approved hardware and software components, the objects will have to undergo a similar assurance process.

For ensuring the acceptance and interoperability of an agile reference architecture, built on top of FACTS with collaborative objects and building blocks, standardization is a key aspect. In particular, the core (fundamental) requirements for FACTS as well as the interfaces and capabilities of the collaborative objects and pluggable building blocks should be standardized. The OGC provides a consensus based collaborative standardization environment fits these requirements very well.

II KEYWORDS

The following are keywords to be used by search engines and document catalogues.

testbed, architecture, Agile Reference Architecture

III CONTRIBUTORS

All questions regarding this document should be directed either to the editor or to the contributors.

NAME	ORGANIZATION	ROLE
Lucio Colaiacomo	EU SatCen	Editor
Greg Buehler	OGC	Contributor
David Habgood	KurrawongAI	Contributor
Christophe Noël	Spacebel s.a.	Contributor
Clemens Portele	interactive instruments GmbH	Contributor
Yves Coene	Spacebel s.a.	Contributor

1

INTRODUCTION

1 INTRODUCTION

The term Agile Reference Architecture (ARA) refers to the long-term vision of the complex and changing nature of how problems will be solved in the future within the location-referenced and geospatial realms, with or without network availability, as containers mix with Linked Data, and as APIs and data become more secure, trusted, and self-describing. In addition to relying on OGC Standards for enabling geospatial interoperability, the geospatial community also depends on the overall state of information and communications technology (ICT), including developing cyber, crypto, and internet technologies. In the OGC Testbed 19 ARA task as documented in this Engineering Report (ER), the reader is encouraged to begin a journey to define where the industry is with the current reference architecture. This discussion includes where the industry is headed in the near term as technology and ideas are developed (next generation), and ultimately, to determine a suitable direction for the generation-after-next of the geospatial community. This ER will not answer all these questions but is intended to provide a baseline, upon which future initiatives will build. This is required in order to evolve into the next, more flexible reference architecture, and ultimately into the agile reference architecture of the generation-after-next.

1.1. Problem statement

In recent years the trend in Information Technology Security — the methods, tools and personnel used to defend an organization's digital assets — developed normative references that require revision of how digital information (including geospatial data) is produced, managed and served. Trust of Data and Services is no longer an implementation issue but more and more an issue for the implementation and adoption of geospatial standards. Moreover, the OGC Standards development process has typically assumed a static networking model in the sense that each operator publishes interface instances or APIs with a given set of functionality. The following issues therefore need to be considered. Proxy caching is a feature of proxy servers that stores content on the proxy server itself, allowing web services to share those resources to more users. The proxy server coordinates with the source server to cache documents such as files, images and web pages. So creating a data space out of the control of the serber (or API). Another issue is how to discover new or updated capabilities provided by the APIs. OGC API Standards support synchronous or asynchronous communication, but still require using HTTP/S and/or MQTT protocols. Another issue is how to establish autonomous interactions between systems assuring trust. OGC API Standards are concerned with managing data, providing access to data, or processing data. For example, consider the current OGC API Standards baseline. A user can access a sensor through implementations of current OGC API Standards. However, how does the user determine the chain of commands for a given activity? How is Trust managed in OGC Standards? How can the provenance of the information be accessed via implementations of OGC Data Encoding Standards? Consider a GML instance document or a map stored in GeoTIFF or GMLJP2. How can the user of that information validate the integrity of the information and be assured about the authenticity of the original author? How is provenance documented? Consider for example a GML document or a GeoTIFF file that was modified by a user in good faith (e.g., updating feature properties to reflect updates). Once the information is saved to storage there is no recording that the modification happened! More than loosely coupled

APIs are required to support the requirements as identified above. An ecosystem of trusted collaborating systems, of which implementations of the OGC API and Web Service Standards can be a part of, needs to be defined. In the D123-128 parts of this docment you will see examples of what is possible to manage with current standards and definitions but in an environment that is neither trusted nor secure.

1.2. Possible path towards Next Generation Architecture

The objective is that any interaction on data inevitably produces a verifiable trace (provenance + identity) and that data itself is secured (principles of data centric security applied). The idea is to introduce Collaborative Objects (CO) that are capable of negotiating relevant business (in particular not data) between each other based on Smart Contracts. For example, Smart Contracts integrated into a Blockchain can assert fundamental communications to produce metadata and provenance. This enables working together in agilely in a self-sovereign / adaptable way. All interactions are controlled via Smart Contracts. F.A.C.T.S. (Federated Adaptive (Infrastructure of) Collaborating Trusted Systems) as the ecosystem establishes trust and provenance based on Collaborative Objects (COs). These COs could be implemented, as an example, via Docker Images using Content Trust. This ensures that there is basic trust in COs, which is required to realize the adaptive Trusted System. Many current implementations of OGC API Standards are not adaptive – they cannot self-adapt because there is no “built-in” logic other than providing access to data, metadata, and processes! They are simply not designed to do so. However, OGC API Standards are important for realizing the vision of data access in F.A.C.T.S. In the context of defining the next generation architecture, it is clear that there is a need to define a whole ecosystem for increased flexibility. Such a new Agile Reference Architecture can also adapt to an increased number of data/processes that are also derived by algorithms creating new COs in near real time services. The main steps for the next activities are Data Centric Security (IPT based), OGC building block definition with IT Security constraints considered (IPT enabled), OGC Standards harmonization to consider IPT.

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2. This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec. For the purposes of this document, the following additional terms and definitions apply.

2.1. Terms and definitions

2.1.1. **Building Block**

A building block is a package of functionality defined to meet specific business needs. The way in which functionality, products, and custom developments are assembled into building blocks will vary widely between individual architectures

2.1.2. Collaborative Object

A Collaborative Object is self-contained and contains data products, services/processes, and collaborates in the exchange of events, as well as the invocation of operations.

2.1.3. Data-centric Security

Data-centric security is an approach to security that emphasizes the dependability of the data itself rather than the security of networks, servers, or applications [Wikipedia].

2.1.4. F.A.C.T.S.

FACTS (Federated infrastructure of Agile Collaborative Trusted Systems) establishes trust based on Collaborative Objects (COs) such as a collection of Docker Images using Content Trust.

2.1.5. Smart Certificate

A Smart Certificate ensures that the F.A.C.T.S. Collaborative Object is doing what it is supposed to do, supporting verified attestation and processes. A Smart Certificate assures, for example, that the APIs on the Collaborative Object interact with F.A.C.T.S. Similar to the Smart Contract, where the recording of the processing is published on the Blockchain BEFORE the data product is published, a Smart Certificate ensures that the Verifiable Attestation is issued before the data product is published.

2.1.6. Smart Contract

A Smart Contract is a computer program or a transaction protocol that is intended to automatically execute, control, or document events and actions according to the terms of a contract or an agreement. [Wikipedia]. A F.A.C.T.S. Smart Contract for example ensures that the recording of the processing metadata (provenance information) is published on the Blockchain BEFORE the service (Building Block) is published.

2.1.7. Verifiable Attestation

A type of verifiable credential containing claims about certain attributes of an entity for uses other than identification or authentication (EBSI definition). https://code.europa.eu/ebsi/json-schema/-/tree/main/schemas/ebsi-attestation

2.2. Abbreviated terms

ABB	Architecture Building Block
API	Application Programming Interface
ARA	Agile Reference Architecture
BB	Building Block
CO	Collaborative Object
CQL	Common Query Language
DCS	Data Centric Security
DDIL	Denied Disrupted Intermittent Limited
DGIWG	Defense Geospatial Information Working Group
DID	Decentralized Identifier
DIF	Decentralized Identity Foundation
DMF	DGIWG Metadata Foundation
FACTS	Federated Agile Collaborating Trusted Systems
GDAL	Geospatial Data Abstraction Library
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IPT	Identity Provenance Trust
ISO	International Organization for Standardization
JSON	JavaScript Object Notation

MGCP	Multinational Geospatial Co-production Program
NGA	US National Geospatial Intelligence Agency
NSG	US National System for Geospatial Intelligence
OGC	Open Geospatial Consortium
OS	Ordnance Survey (Great Britain)
RDF	Resource Description Framework
REST	Representational State Transfer
RM-ODP	Reference Model of Open Distributed Processing
SatCen	European Union Satellite Center
SBB	Solution Building Block
SC	Smart Certificate
SHACL	Shapes Constraint Language
SSI	Self-Sovereign Identity
SWG	Standard Working Group
TIE	Technology Integration Experiment
TOGAF	The Open Group Architecture Framework
TP	Trust and Provenance
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VA	Verifiable Attestation
XML	eXtensible Markup Language

3

REFERENCE ARCHITECTURE

3 REFERENCE ARCHITECTURE

3.1. Actual status

Currently the OGC Reference Model (ORM) architecture is used as the basis for OGC Standards work (OGC 08-062r7). The ORM is defined using Reference Model for Open Distributed Processing (RM-ODP) which is an international standard for architecting open, distributed processing systems. Recent advances in various technologies are causing a self-reflection on the way geospatial systems architecture can be adapted for the next generation of geospatial systems and for the generation-after-next. Below in Clause 3.2, is a short but not exhaustive list of the reference architectures that are in use and can be considered within the scope of this Engineering Report.

3.2. Examples of architectures in use

The **OGC RM** has the following purposes (OGC 03-040):

- provides a foundation for coordination and understanding (both internal and external to OGC) of ongoing OGC activities and the OGC Technical Baseline;
- update/Replacement of parts of the 1998 OpenGIS Guide (https://www.ogc.org/standards/orm/);
- describes the OGC requirements baseline for geospatial interoperability;
- describes the OGC architecture framework through a series of non-overlapping viewpoints: including existing and future elements; and
- regularizes the development of domain-specific interoperability architectures by providing examples.

DGIWG Geospatial Reference Architecture (DGRA)

The DGRA defines a set of standards, implementation guides, and industry practices which together form an ideal framework for achieving geospatial interoperability in a Defense context (DGIWG 933). The DGRA is particularly relevant to this engineering report because of its application in the Defense community and its use of ISO/IEC 10746 1-3 “Information Technology — Open Distributed Processing — Reference Model” (RM-ODP).

Open Distributed Processing — Reference Model (RM-ODP)

The RM-ODP defines a model that portrays a reference architecture from the following viewpoints.

- Enterprise: Defines the purpose, scope and policies of the system.
- Information: Describes the semantics of information used within the system, e.g., Vector, Imagery, Metadata, Portrayal, and their relevant standards.
- Computational: Describes the systems individual interfaces, e.g., the standards and the operations they use for each function.
- Engineering: Describes the system components, their relationship functions and standards.
- Technology: Describes the technology choices available to realize systems in terms of their compliance to specifications described in other viewpoints.

Geospatial Interoperability reference architecture

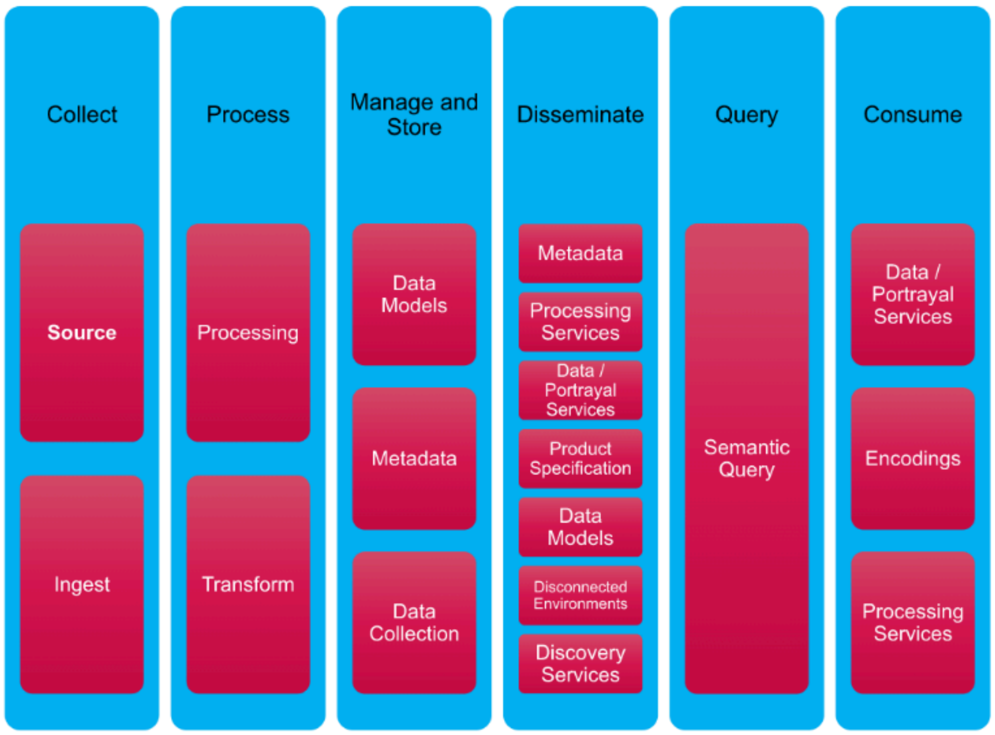


Figure 1 — actual working brainstorming architecture (logical workflow)

OGC is in the process of addressing the need for a modernized reference architecture through its work on OGC API Standards. The definition of the concept of OGC API 'Building block', is still not standardized. To be more precise it is defined in the OGC Technical Commettee Policies and Procedures 7.2.1 where it is stated "There is no firm definition for the content or scope of a building block, but the building block must fulfill a function that can operate in the larger context of an implementation,...". As properly referred in that document the implementation part that is connected with security of data and services by normative reference (last but not least EU Digital Act) impose a redefinition of the architecture. The main disadvantage is that – as with the existing OGC Web Services Standards – security is not directly integrated into the API. As such, security needs to be added as part of the implementation or managed externally. Actual implementations of OGC API Standards are based on OGC building blocks (some of which are not yet standards) and there is no security capability yet for this concept. Where a component is specific to a given OGC API Standard, this is not an issue. However, this can result in definitions that are not unique. Because the building blocks currently do not use agreed cross API semantics, definitions, and concepts, the interfaces defined and in use are likely be developed on an ad hoc basis, resulting in stove-piped solutions that may not be fully interoperable.

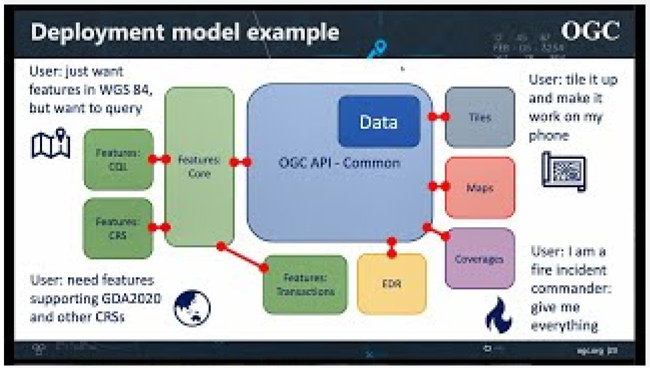

Deployment model example
OGC
User: just want features in WGS 84, but want to query
Features: CQL
Features: CRS
Features: Core
Data
OGC API - Common
Tiles
Maps
Coverages
Features: Transactions
EDR
User: tile it up and make it work on my phone
User: need features supporting GDA2020 and other CRSs
User: I am a fire incident commander: give me everything

Figure 2 — architecture deployment example

Current architectures — and in particular for geospatial systems — exhibit a strong dependency on data types (meaning metadata describing different encodings) that could be reduced as much as possible to enable real interoperable processes and services. The managing of new data formats and their standardization requires effort that some organizations cannot afford. Today the situation is that from the moment a new data format is available (produced from the market or developed) up to the moment it is standardized (OGC, ISO…) the delay is considerable and does not allow an agile approach. Even though this development is asynchronous to software development and an OGC API implementation instance may use the encoding once standardized, there is the risk of not being interoperable.

The drawback of this approach is that the creation of connected services (consider the draft OGC API — Connected Systems (https://ogcapi.ogc.org/connectedsystems/)) or persistent monitoring capability is not straight forward. The building blocks approach enables modularity but requires basic elements such as Data Centric Security (DCS) in their development and life cycle. Therefore, DCS should be considered as an element for actual architectures wherever applicable and mainly for the next and generation after next. Many OGC encodings and API Standards do not specify security constraints. Perhaps it is impossible because there are too many use cases to be considered? This implies difficulty in establishing data trust and provenance to data/services/processes of a single operator and implies ‘mission impossible’ when trying to build workflows with secured services as in a federated setup.

Service Orchestration and Automation Platforms (SOAP)

Considering Service Orchestration and Automation Platforms (SOAP), Gartner says: “…SOAPs enable I&O leaders to design and implement business services. These platforms combine workflow orchestration, workload automation and resource provisioning across an organization’s hybrid digital infrastructure. Increasingly, they are central to an organization’s ability to deploy workloads and to optimize deployments as a part of cost and availability initiatives. SOAPs provide a unified administration console and an orchestration engine to manage workloads and data pipelines and to enable event-driven application workflows. Most tools expose APIs enabling scheduling batch processes, monitoring task statuses and alerting users when new events are triggered that can be integrated into DevOps pipelines to increase delivery velocity. SOAPs expand the role of traditional workload automation by adapting to use cases that deliver and extend into data pipelines, cloud-native infrastructure and application architectures. These tools complement and integrate with DevOps toolchains to provide customer-focused agility and cost savings, operational efficiency and process standardization." (source: https://www.gartner.com/reviews/vendor/storidge).

The Open Group Architecture framework (TOGAF)

A very detailed Architecture document is "The Open Group Architecture Framework" (TOGAF). The proposal is to have the following definition for a Building Block, which was defined by the Testbed-19 participants. There are two main aspects to consider, Architecture building block (see definition below) and solution building block (see below) that could better fit the actual OGC API concept and leave room for IPT adaptation.

3.3. Building blocks

3.3.1. Official OGC definition of a Building Block

Clause 7.2.1 of the OGC Technical Committee's Policies and Procedures (OGC 05-020r29) defines a Standard Building Block as follows:

"Many OGC Standards are structured with modular sets of requirements (or requirement classes) that collectively function as a reusable building block. There is no firm definition for the content or scope of a building block, but the building block must fulfill a function that can operate in the larger context of an implementation, including combination with other OGC building blocks to create novel implementations.

Building blocks developed for one Standard can be reused in another Standard. To facilitate such reuse, a Standard constructed of building blocks shall identify each building block and publish a definition of the building block to OGC's Registries and web resources. The definition will be in the form most suitable for the type of building block (e.g., Open API for a Standardized API), reference the owning Standard, and be adequately documented to be used in reference.

OGC Standards that reuse building blocks from other Standards must include in the Normative References a reference to the owning Standard of the building block(s) and a direct reference to the registered building block(s) content. In this fashion, implementers of the Standard reusing these building blocks need to only access specific parts (the building blocks) of the referenced Standard, not the entire document."

3.3.2. Generic characteristics of a building block

This engineering report therefore identifies the generic characteristics of a building block as follows:

- a package of functionality defined to meet the business needs across an organization;
- has published interfaces to access the functionality;
- may interoperate with other, inter-dependent building blocks;
- considers implementation and usage, and evolves to exploit technology and standards;
- may be assembled from other building blocks;
- may be a subassembly of other building blocks;
- ideally is re-usable, replaceable, and well-specified; and
- may have multiple implementations but with different inter-dependent building blocks.

A building block is therefore simply a package of functionality defined to meet specific business needs. The way in which functionality, products, and custom developments are assembled into building blocks will vary widely between individual architectures. Every organization must decide for itself what arrangement of building blocks works best for their use cases. A good choice of building blocks can lead to improvements in legacy system integration, interoperability, and flexibility in the creation of new systems and applications. Systems are built from collections of building blocks, so most building blocks must interoperate with other building blocks. Wherever that is true, it is important that the interfaces to a building block are published and reasonably stable and persistent. Building blocks can be defined at various levels of detail, depending on what stage of architecture development has been reached. For instance, at an early stage, a building block can simply consist of a grouping of functionality such as a customer database and some retrieval tools. Building blocks at this functional level of definition are described in TOGAF as Architecture Building Blocks (ABBs). Later, real products or specific custom developments replace these simple definitions of functionality, and the building blocks are then described as Solution Building Blocks (SBBs).

3.4. Architecture and Solution Building Blocks (TOGAF definition)

The following content has been copied from the TOGAF specification [20] and represents a possible definition of what a building block could be at the architecture level and at solution level. This definition could help in maintaining the actual implementations and opening the door to the introduction of the IPT concept.

3.4.1. Architecture Building Blocks

Architecture Building Blocks (ABBs) relate to the Architecture Continuum (https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap18.html#tag_19_01), and are defined or selected as a result of the application of the Architecture Development Method (ADM). The ADM is a generic method for architecture development, which has been designed to deal with most system and organizational requirements.

Characteristics

The following are characteristics of Architecture Building Blocks:

- define what functionality will be implemented;
- capture business and technical requirements;
- are technology aware; and
- direct and guide the development of SBBs.

Specification Content

ABB specifications include the following as a minimum:

- fundamental functionality and attributes: semantic, unambiguous, including security capability and manageability;
- interfaces: selected, supplied (APIs, data formats, protocols, hardware interfaces, standards);
- dependent building blocks with required functionality and named user interfaces; and
- mapped to business/organizational entities and policies.

3.4.2. Solution Building Blocks

Solution Building Blocks (SBBs) relate to the Solutions Continuum (https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap18.html#tag_19_02), and may be either procured or developed.

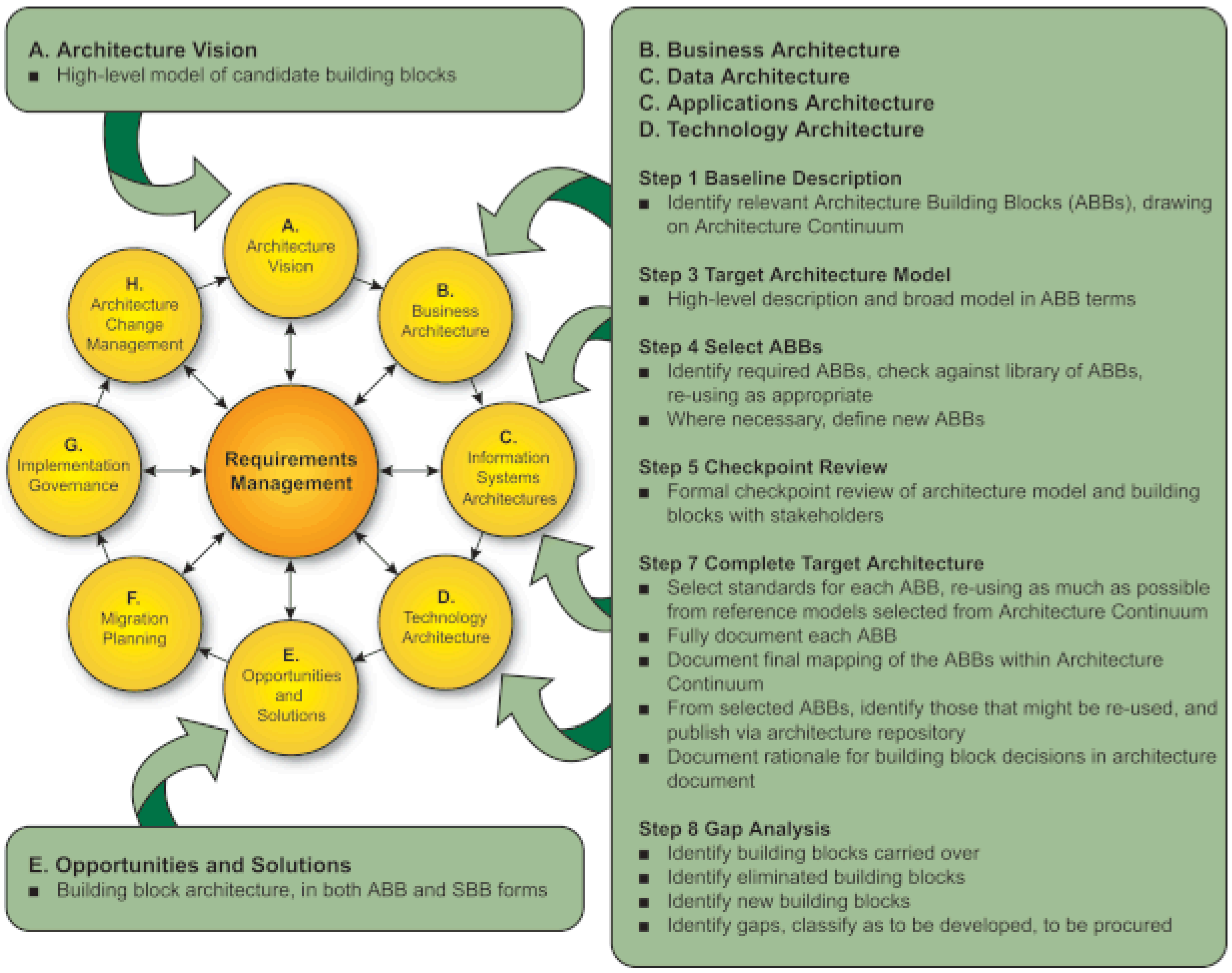

A. Architecture Vision
High-level model of candidate building blocks
B. Business Architecture
C. Data Architecture
C. Applications Architecture
D. Technology Architecture
Step 1 Baseline Description
Identify relevant Architecture Building Blocks (ABBs), drawing on Architecture Continuum
Step 3 Target Architecture Model
High-level description and broad model in ABB terms
Step 4 Select ABBs
Identify required ABBs, check against library of ABBs, re-using as appropriate
Where necessary, define new ABBs
Step 5 Checkpoint Review
Formal checkpoint review of architecture model and building blocks with stakeholders
Step 7 Complete Target Architecture
Select standards for each ABB, re-using as much as possible from reference models selected from Architecture Continuum
Fully document each ABB
Document final mapping of the ABBs within Architecture Continuum
From selected ABBs, identify those that might be re-used, and publish via architecture repository
Document rationale for building block decisions in architecture document
Step 8 Gap Analysis
Identify building blocks carried over
Identify eliminated building blocks
Identify new building blocks
Identify gaps, classify as to be developed, to be procured
A. Architecture Vision
B. Business Architecture
C. Information Systems Architectures
D. Technology Architecture
E. Opportunities and Solutions
F. Migration Planning
G. Implementation Governance
H. Architecture Change Management
Requirements Management
E. Opportunities and Solutions
Building block architecture, in both ABB and SBB forms

Figure 3 — architecture vision (The Open Group Architecture Framework)

Characteristics

The following are characteristics of Solution Building Blocks:

- define what products and components will implement the functionality;
- define the implementation of the building block;
- fulfill business requirements; and
- are product or vendor-aware.

Specification Content (example not to be considered all applicable for OGC context)

SBB specifications include the following, as a minimum:

- specific functionality and attributes;
- interfaces: the implemented set;
- required SBBs used with required functionality and names of the interfaces used;

- mapping from the SBBs to the IT topology and operational policies;
- specifications of attributes shared across the environment (not to be confused with functionality) such as security, manageability, localizability, and scalability;
- performance and configurability;
- design drivers and constraints, including the physical architecture; and
- relationships between SBBs and ABBs.

3.5. Comparison based on the above elements

Having analyzed the status of the various architectures available, the following can be assumed.

1. The current situation based on a mix of commercial and open source solutions (partially implementing OGC and other standards) do not allow easy chaining of services and processes. With complex systems it is even more difficult, if using dynamic solutions, to enable certain flexibility if adapting systems dynamically is required.
2. When ensuring security aspects such as identity+integrity, provenance and trust (Data Centric Security) should be considered as one of the pillars to build future architectures. Currently — as demonstrated in previous OGC Testbeds — there is compatibility but not real implementation.
3. Derived from the previous points regarding trust on the data, it can be said that trust=identity+integrity+provenance. As such an IPT-enabled system can be based on identity and provenance.
4. The use of data to train models as used in some Artificial Intelligence workflows is not currently defined in OGC Standards and should follow IPT criteria (unless provenance in OGC Training ML). One recent exception is OGC TrainingDML-AI Standard that specifies the requirement for detailed metadata for formalizing the information model of training data. This includes, but is not limited to, the following aspects: how the training data is prepared, such as provenance or quality, etc.

So there is a need to start defining a standardization for building blocks and possible implementations of OGC API Standards that support an IPT based Data Centric Security approach.

4

NEXT GENERATION ARCHITECTURE

4 NEXT GENERATION ARCHITECTURE

The Reference Architecture chapter outlined the limitations of the current architecture. Basically, the need to chain data and services is limited by the data encoding dependency while service interfaces are strongly related to the data type (features, maps, etc.). Basically, it is not feature type agnostic. Moreover, data and services, if chained, must be trusted and identifiable. Otherwise, the results cannot be trusted and so any results have limited trustworthiness.

The significance of not having identifiable provenance for data and services is that it does not allow for agile connection between services and processes. This is because identification, trustiness, and provenance are fundamental elements for enabling trustworthy chaining of services. Identity and trustiness of data and services are also a key factor for developing and deploying Machine Learning models and algorithms that are not only accurate, but also explainable, FAIR, privacy-preserving, causal, robust, and trustworthy. To establish a viable definition of the next generation architecture, the following points should at least be considered:

- the architecture should provide a level of abstraction to manage the complexity of the system providing also communication and orchestration among building blocks;
- the architecture should provide a solution that considers performance and security criteria;
- there is a need for definition of building blocks and their interfaces;
- building Block definition and specification, if agreed, should be an OGC standard;
- there is a need to consider Smart Certificate definition adoption by OGC; and
- there is a need to consider Smart Contract definition adoption by OGC.

The following is derived from prototyping-focused projects conducted by EU SatCen and are provided here only as a reference.

4.1. Federated Agile Collaborating Trusted Systems (FACTS)

In today's infrastructures, the collection, exchange, and continuous processing of geospatial data takes place at pre-defined network endpoints of a spatial data infrastructure. Each participating operator hosts predefined static functionality at a network endpoint: Some operator network endpoints may provide data access, other endpoints provide processing functionality, and other endpoints uploading capabilities. In other words, such an infrastructure is not agile in the sense that it cannot adapt by itself to meet more real time needs. One of the biggest challenges resulting from these static characteristics is ensuring effective and efficient operations of the overall system and at the same time maintaining trust and provenance.

This chapter outlines novel concepts for establishing federated agile infrastructure of collaborative trusted systems (FACTS) that is capable of acting autonomously for ensuring fit-for-purpose cooperation across the entire system. One of the key objectives is, for example, that a data product is not made available, but instead a collaborative object is offered leveraging FACTS that supports retrieval of the data product via well-defined interfaces and functions provided by the collaborative object.

Trust and assurance are two key aspects when operating a network of collaborating objects leveraging STANAG 4774/4778.

The agile aspect is achieved by the object's ability to activate, deactivate, and order well-defined capabilities from other objects. These capabilities are encapsulated in building blocks. Each building block is well-defined in terms of accessibility, functionality, and ordering options. This allows building blocks to roam around collaborative objects as needed to ensure a well-balanced network load and processing power of individual nodes from the network.

Equally trusted partners in the infrastructure participate in FACTS. They are capable of collecting data from other partners and creating derived products via collaborating objects. The sharing of data products is not directly possible. It is only possible via the objects. This guarantees that fundamental trust operations applied to the data and provenance records are produced before the data product is made available to others. The use of the Blockchain technology and Smart contracts is one example of how this fundamental behavior can be planted into collaborative objects. As in trusted networks that are using EAL approved hardware and software components, the objects will have to undergo a similar assurance process.

Building blocks define capabilities that can be activated, de-activated, or ordered from other objects in the FACTS network. Even though the actual capabilities of a building block are subject to configuration (based on need), there are fundamental APIs that a building block must support. Considering that collaborative objects get distributed and executed via Kubernetes (https://kubernetes.io/) or Helm (https://helm.sh/), for example, there is a key requirement of trust. One approach to manage the fundamental trust in FACTS can be achieved via The Update Framework (TUF) (https://theupdateframework.io/), which is called content trust in the Docker environment.

Enhancing FACTS as an existing network of collaborative objects with additional capabilities such as knowledge generation from Artificial Intelligence and Ontologies, the provenance for and trust of training data must be considered. Without applying trust and provenance to AI training data, the best algorithm is useless,, or even dangerous if trained with fraudulent data.

With FACTS, each participating entity can make available data or processing capabilities as it meets their quality and security requirements. The ability to use data and processing capabilities throughout FACTS strengthen the common capabilities because trustworthy can be chained on the fly. Additional capabilities, available via a Licensing Building Block, may support expressing re-use conditions.

Data products may become available to many participating entities. What is shared and under which re-use conditions are at the discretion of the creating entity. Anything that is shared must have re-use conditions to ensure proper and legitimate uptake. For example, leveraging the Creative Commons licensing framework allows an entity to waive all rights but also allows expressing concrete re-use conditions ranging from simple attribution, non-commercial, or preventing deriving own work conditions.

The use of FACTS can be compared with middleware that ensures integrity, provenance, and trust of geospatial data products as created and distributed by any organization. For example, data products encoded as GMLJP2, GeoPDF, geodb, and GeoTiff would be made available via collaborative objects and thereby benefit from collaboration in the entire network. By using FACTS, other EU member states could create derived work from the products generated. The fundamental trust capabilities of the collaborative objects ensure that any modification to a generated product can be detected and that the provenance capability enables tracing of the lineage of derived products towards the original source. For ensuring the acceptance and interoperability of an agile reference architecture, built on top of FACTS with collaborative objects and building blocks, standardization is a key aspect. In particular, the core (fundamental) requirements for FACTS as well as the interfaces and capabilities of the collaborative objects and pluggable building blocks should be standardized.

Projects running in Europe like the European Blockchain Services Infrastructure (EBSI) (https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home), Data Act (https://www.europarl.europa.eu/doceo/document/TA-9-2023-0069_EN.pdf), create the baseline to start revising the Architecture of Geospatial (and other) systems based at least on Security and, if possible, making the architecture more flexible. OGC provides a consensus based collaborative standardization environment that fits very well in this vision and could propose such concepts to try to find a way forward towards a proper discussion across the geospatial community.

4.2. Requirements for next generation architecture

This section presents a possible system architecture that can answer the above use case and reflects the requirements for a next generation architecture. The architecture should support possible interaction between APIs (services) and data independently of pre-existing systems.

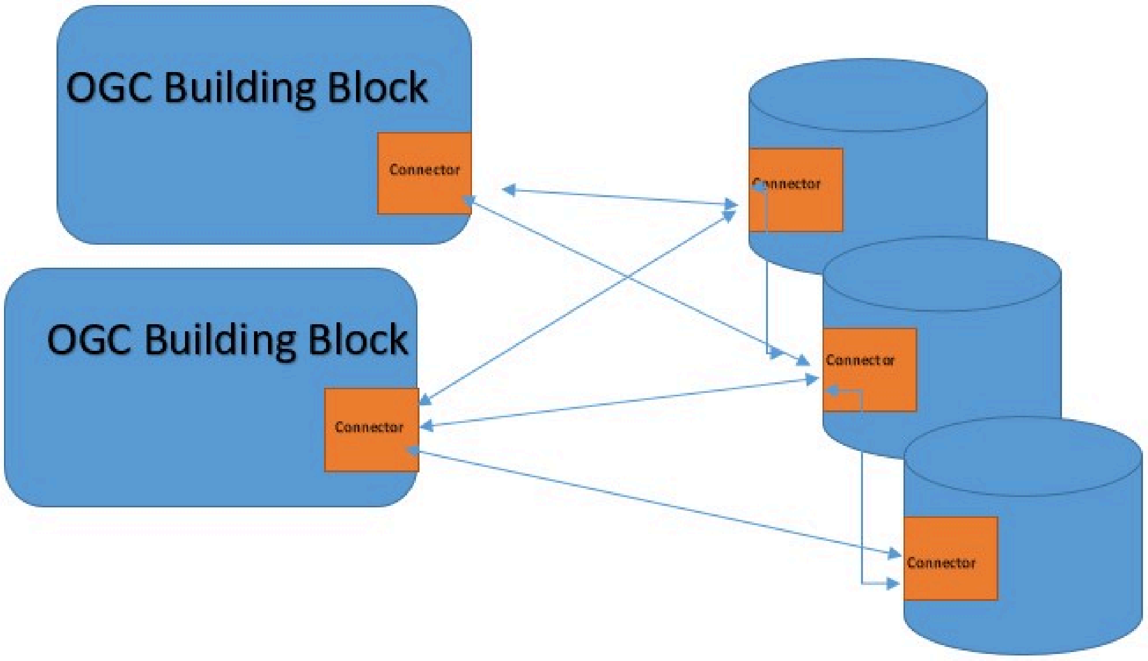


Figure 4 — possible reference architecture

REQUIREMENT 1	
STATEMENT	The new architecture should be, as much as possible, independent of data encodings. The encoding could just offer a file that uses Key Value Pairs (KVP) to present metadata that describes all the relevant information in a way that is similar to the DGIWG Metadata Foundation (DMF).

REQUIREMENT 2	
STATEMENT	Building blocks should be as service-agnostic as possible (without specialization according to data types such as maps, features, and coverages)

REQUIREMENT 3	

STATEMENT	All source of information should at least be defined in terms of trust and provenance to support the proper assessment of validity through a digital signature as shown below:
	1. signature
	2. signature value e.g,
	`<gmljp2:extension>`
	`    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">`
	`      <ds:SignedInfo>`

REQUIREMENT 3

```
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#rsa-sha1"/>
        <ds:Reference URI="">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/
xmldsig#enveloped-signature"/>
            <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-
19991116">
              <ds:XPath xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:
err="http://www.w3.org/2005/xqt-errors" xmlns:fn="http://www.w3.org/2005/
xpath-functions" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:gmlcov=
"http://www.opengis.net/gmlcov/1.0" xmlns:gmljp2="http://www.opengis.net/
gmljp2/2.0" xmlns:math="http://www.w3.org/2005/xpath-functions/math" xmlns:
swe="http://www.opengis.net/swe/2.0" xmlns:xs="http://www.w3.org/2001/
XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">/gmljp2:
GMLJP2CoverageCollection/gmlcov:metadata[1]/gmlcov:Extension[1]/gmljp2:
eopMetadata[1]</ds:XPath>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#sha1"/>
          <ds:DigestValue>PozvauWPsaua10zZ0cfnw4cTJu4=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
        SQhuJ0FQzHPh4I0VTgUdtvdNc9TREL7q2WyZb5FLby0XNPFZ6h9r/
ZiukgUyrryGpLBqyOvGprE pv4+cvrurbZcUik7Z4BoN2hxNs9T35P92sMjf9BGiCy5dgxSho9sI
L29Hf0u9b6rfoQAj03NC7FJ/rR1EGAN6T5AMK4bBT/iG/fNWfZKC9DimNwCLvezj3sryodrrl+D0
RfOrU7mL7d7IMsV75g5uklz/kilosBaQbkek6R+UINP8bY+yv1SD+Imyii+xO17TU9FPRh9puEwL
raauDm7RePPwZ4n5kdu2l5yg+/b1kRZAMbZIHWBbYslbMoEz21keRVjXeHjA==
      </ds:SignatureValue>
    </ds:Signature>
  </gmljp2:extension>
```

REQUIREMENT 4

STATEMENT	All building blocks should support at least the same approach of the simple management of trust and provenance. Building blocks should implement a meta description enabling automatic activation on specific data sources (See OGC 20-089r1, OGC Best Practice for Earth Observation Application Package, for an example).

REQUIREMENT 5

STATEMENT	Data should be discoverable and queryable depending on the IPT and releasibility/accessibility.

REQUIREMENT 6

STATEMENT	A schemaless Data Model would be needed. An example would be the Open Street Map (OSM) dataset mapped to a new data dictionary with an Excel spreadsheet and then schema recovered automatically. Note that SAFE FME (Http://www.safe.com) and Hootenanny are examples of data

REQUIREMENT 6	
	conflation tools that can facilitate automated and semi-automated conflation of critical Foundation GEOINT features in the topographic domain. In short, it merges multiple maps into a single seamless map.

REQUIREMENT 7	
STATEMENT	Context AI tools and other processes following extraction guidelines used in the implementation of the processes themselves. These guidelines provide valuable information. As an example, the definition of a beach: on a shore, the area on which the waves break and over which shore debris (for example: sand, shingle, and/or pebbles) accumulate. These definitions could enable the use of RDF/Turtle.

REQUIREMENT 8	
STATEMENT	Trust and provenance validation could be implemented following a decentralized approach with criteria starting from a majority vote of the various nodes up to 100% requirement. But this can work also in the configuration of networks in Denied, Degraded, Intermittent, or Limited Bandwidth (DDIL) environments ensuring local IPT, and further checking once connected to a node of the system.

REQUIREMENT 9	
STATEMENT	Building blocks should enable streaming of information. Signaling of information available in a stream to other building blocks should be active. This could happen in an asynchronous or synchronous way and/or in accumulation mode. In this context a building block could be considered as “Data as a Service”, for example.

REQUIREMENT 10	
STATEMENT	Data should be discoverable and queryable depending on the IPT and releasability/accessibility.

Considering the above requirements it could be assumed that Identity (+ integrity) Provenance and Trust (IPT) layering is a required action to the next ARA, because the Architecture should at least manage IPT tuple. The IPT management is not affecting the open nature (if any) of the data and service but establish a minimal liability for the usage and further processing, e.g., usage of machine learning algorithm with information with no provenance and not trusted source.

The above requirements leads to the following issues.

1. Managing of different data types and services in distributed environments

2. File systems limitations with new virtualized environment (e.g., docker container, etc..)

3. Data files and streaming

4. Cloud/edge/local configuration.

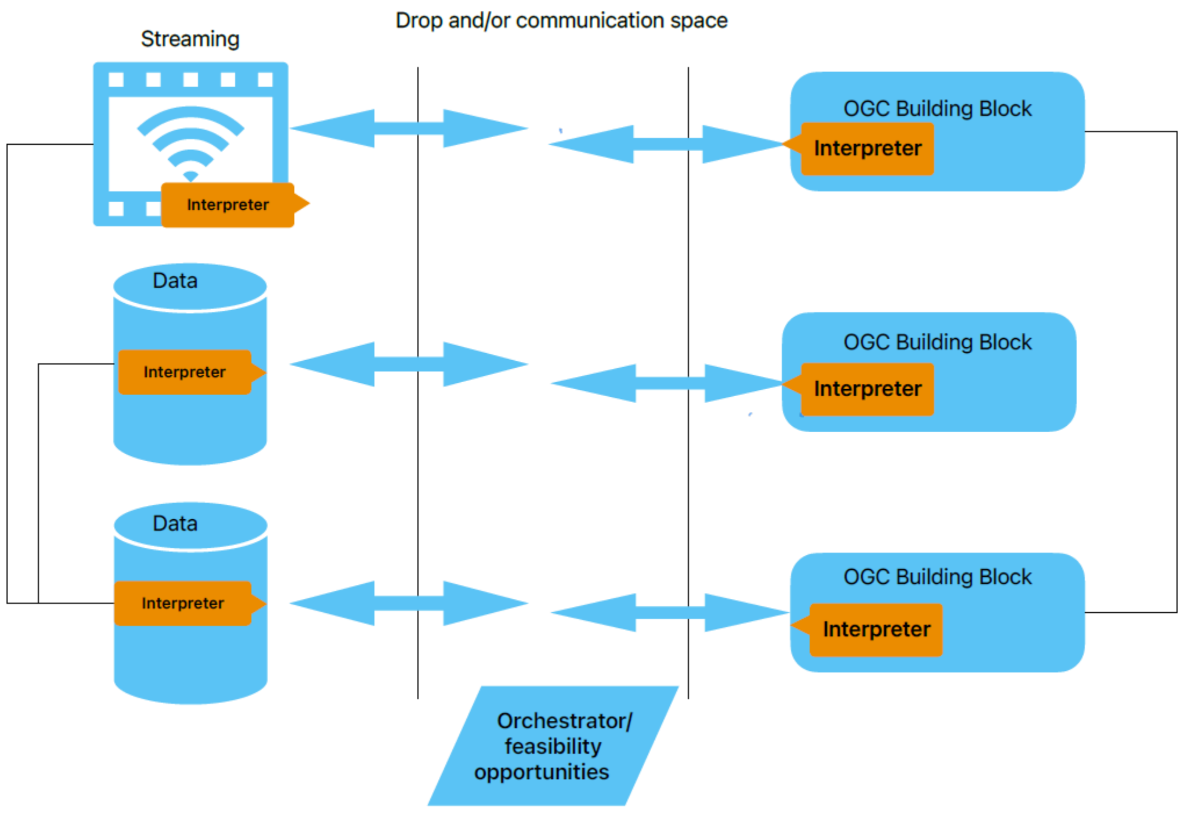


Figure 6 — next generation architecture

4.3. Building block definition — further considerations

In the TOGAF specification, “a Building Block is a package of functionality defined to meet business needs across an organization”. There is a type corresponding to a TOGAF metamodel. In this context, a building block (BB) is a “thing” (e.g., company, server, etc.) with well-defined interfaces, boundaries, and specifications to enable reusability. Moreover, BBs can be classified into Architectural and Solution BBs (technology/vendor aware). The first drives the development of the second. One or more building blocks can be integrated into existing/novel web applications. Each building block represents a testable interface component.

The BB definition should support interaction between data and services and provide at least the following parameters (dimensions, locations, domain, range values, and types (null and interpolation, etc.)) after checking the availability of information related to the BB via functions.

The Data container should be provided with a “description” enabling BBs to interact via a set of functions and determine possible workflows. This could also be achieved with algorithms that could be integrating part of the orchestrator (i.e., orchestration is the coordination and management of multiple computer systems, applications, and/or services, stringing together multiple tasks in order to execute a larger workflow or process (http://databrticks.com)).

Considering experience acquired in the implementation of the OGC Web Processing Service (WPS) and Web Coverage Processing Service (WCPS) Standards, the following can be stated.

1. WPS supports any kind of geoprocessing, whereas WCPS focuses on coverage processing.
2. WPS consists only of a low-level framework for procedural invocation, whereas WCPS gives a high-level, concrete, and concise service specification.
3. WPS specifies static services, whereas WCPS provides the flexibility of dynamic ad-hoc query formulation. In other words, a WPS extension requires client and server-side programming, whereas with WCPS this means composing a new string on the client side, without any changes to the server.
4. WCPS supports phrasing of analytically expressible algorithms. WPS, on the other hand, by definition is Turing complete; As experience shows, WCPS offers a high potential for automatic chaining and optimization. WPS typically requires manual server-side intervention, such as code tuning in supercomputing centers.

Therefore, mechanisms for automation already exist and it can be assumed that they could be integrated with AI/ML. If there is an orchestrator (maybe referring to a register for the list of available building blocks), then there is already a possible scenario for the next architecture. The above can only be implemented leaving the flexibility to be automatically updated as stated before, otherwise the result is a static approach. This orchestrator can be considered as another API that is able to integrate different BBs.

To properly reflect the dynamic approach, an event driven mechanism (see event driven architecture Pub/Sub) should be considered. However, this compounds the fact that more and more streaming of information and algorithms can create active information signaling in a streaming environment.

This new approach redefines the way information is shared between data and services. It can be assumed that, first, an interaction space is required where data and building blocks can interact.

The building block, which could be defined using yaml, should include the following.

1. Metadata (it could be DGIWG Metadata File)
2. Specification (ogc-api reference)
3. Configuration (possible data values or streaming to be handled, etc.).

The above can be managed by an admission webhook (e.g., a listener waiting for a new BB to be published, a registry service) that is validating and registering the Building Block.

4.4. Interaction space

The interaction space could be a distributed object storage system, software defined, and should operate with disconnected or limited connection capability.

The above elements should provide identity management, encryption, and possibly distribution.

The basic idea is to use W3C Decentralized Identifiers v1.0 (e.g., https://www.w3.org/TR/did-core/). As an example, the Hyperledger Indy (https://www.hyperledger.org/projects/indy) provides tools, libraries, and reusable components for providing digital identities rooted on blockchains or other distributed ledgers so that they are interoperable across administrative domains, applications, and any other silo. Indy is interoperable with other blockchains or can be used standalone powering the decentralization of identity.

As an example, Hyperledger Fabric (https://www.hyperledger.org/projects/fabric) (for managing provenance) is intended as a foundation for developing applications or solutions with a modular architecture. Hyperledger Fabric allows components, such as consensus and membership services, to be plug-and-play. Fabric's modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy. Using Hyperledger, the yaml file previously described could be substituted with a Smart Certificate.

While information security nowadays represents a core concern for any organization, Trust Management is usually less elaborated and is only important when two or more organizations cooperate towards a common objective.

For example, the overall Once-Only Principle Project (TOOP) (https://toop.eu/architecture) relies on the concept of trusted sources of information and on the existence of a secure exchange channel between the Data Providers and the Data Consumers in this interaction framework. Trust and information security are two cross-cutting concerns of paramount importance. These two concerns are overlapping, but not identical and they span all of the interoperability layers, from the legal down to the technical, passing through organizational and semantic layers.

While information security aims at the preservation of integrity, confidentiality, and availability of information, the establishment of trust guarantees that the origin and the destination of the data and documents are trustworthy (trustworthiness) and authentic (authenticity), and that data and documents are secured against any modification by untrusted parties (integrity).

Keeping in mind that the above are just examples and it would be interesting see different implementations based on other concepts and tools.

4.5. Way ahead

The next-generation architecture should be based on Data Centric Security (DCS). The DCS concept is implemented through the adoption of the Identity Provenance and Trust concept (IPT). The concept can be improved with Integrity having an I2PT. The ecosystem that is proposed, which will be based on W3C Decentralized Identifiers (DID) and e.g., Hyperledger (https://www.hyperledger.org/projects/indy), will enable both data and building blocks. The entire architecture could be based on the concept of Kubernetes K8s volume abstraction concept (https://kubernetes.io/docs/concepts/storage/volumes/) but can be any other space where data and deployed APIs interact together. For data, W3C DID will be adopted to create identity and other elements for provenance and trust in order to compose a Smart Certificate. This is an active Certificate and can be chained with any other data or BB through an orchestrator or registry. Building blocks (such as those of OGC API Standards) have to be compatible with the IPT ecosystem. This is easier because by modifying the OGC Compliance & Interoperability Testing Environment (CITE) it will be possible to certify BB to be compatible with such an ecosystem, and so through a Smart Contract they can be registered in the main registry /orchestrator. In this way both data and BB can be chained to perform specific operations to be proposed to users or to other services.

The ecosystem that provides agile processing is based on Smart Contracts that run on a Distributed Ledger, such as Hyperledger Fabric. The conditions in a Smart Contract enforce that the provenance of data processing gets recorded on the distributes ledgers (Fabric nodes). As such, the Hyperledger Fabric is concerned with making processing results transparent by capturing provenance. The use of DID (W3C Recommendation) and VA (Verifiable Attestations) is the essential part for establishing the integrity of assets, e.g., data products, metadata records, etc., basically, anything that can be hashed. The issuing of DIDs for users and the recording of immutable “Smart Certificates” a.k.a. verifiable attestations + some business logic, can be implemented using, e.g., Hyperledger Indy. The combination of Hyperledger Indy and Fabric builds the complete ecosystem to support agile IPT.

To clarify the basic architecture elements required to implement the next generation architecture, below are implementation examples. The idea for using a Kubernetes persistent volume example is just to show actual capabilities and consider in the future where any space of interaction could be possible.

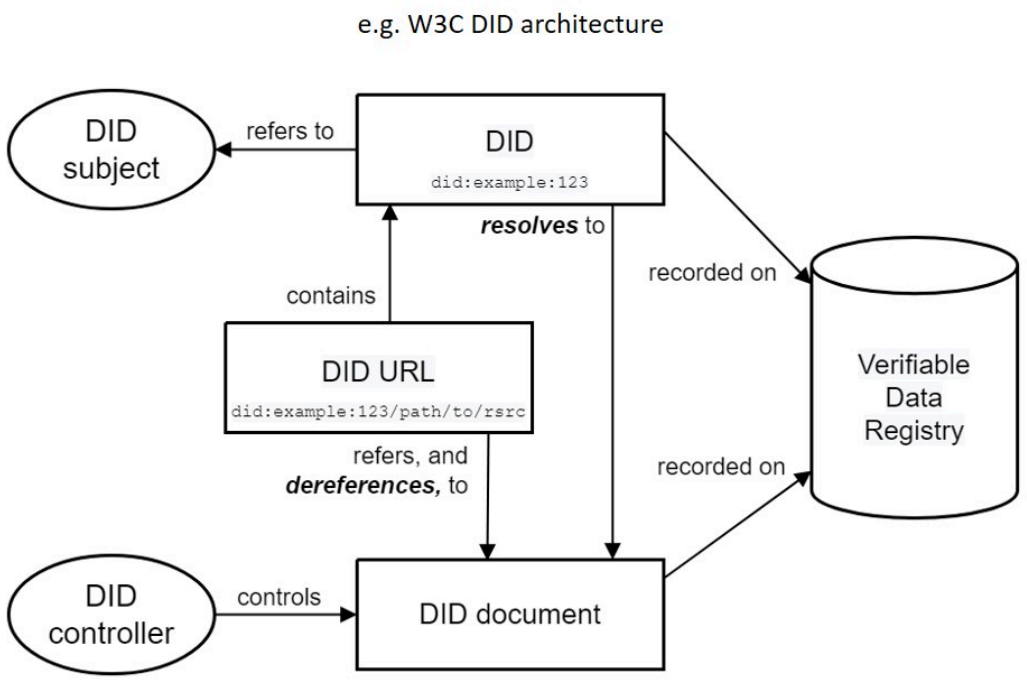


Figure 7 — w3c did sample architecture

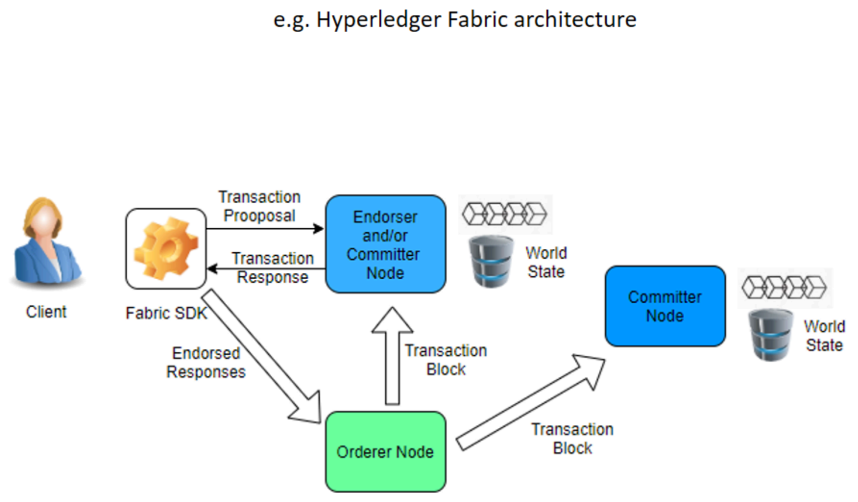


Figure 8 — hyperledger fabric sample architecture

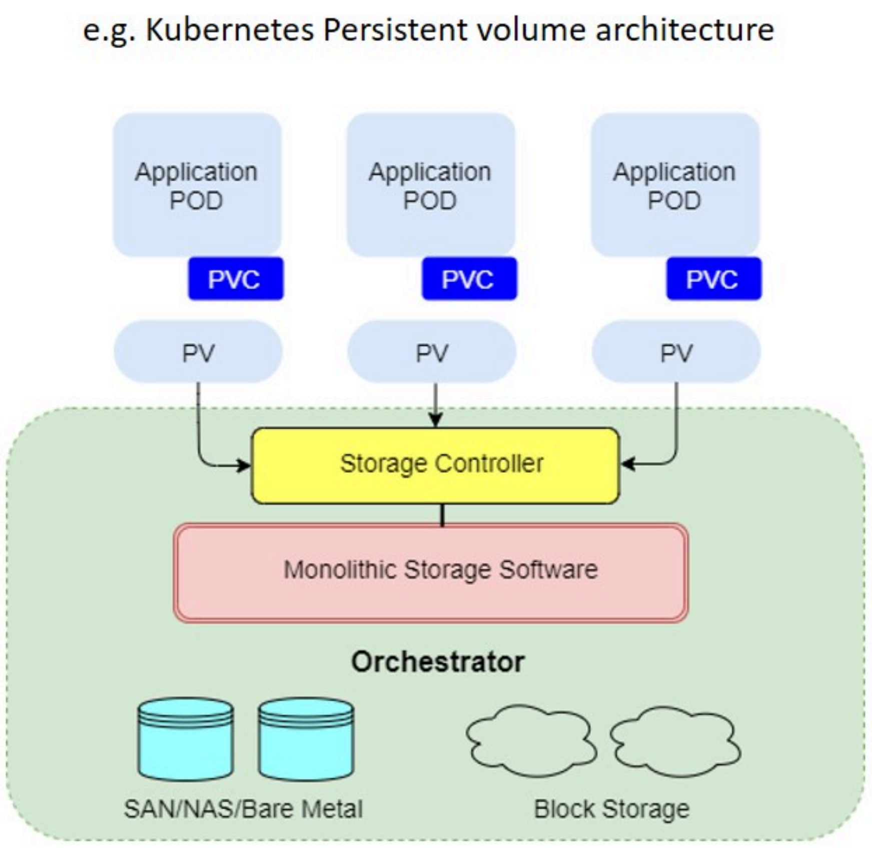


Figure 9 — kubernetes persistent volume sample architecture

Transition from actual OGC data format (any) to a new version IPT enabled with Smart Certificate

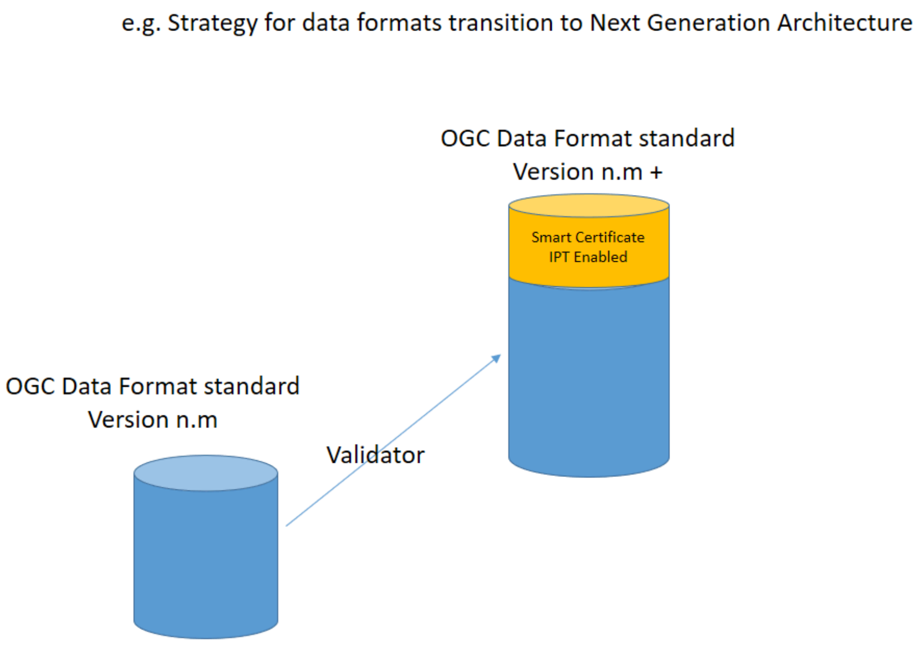

e.g. Strategy for data formats transition to Next Generation Architecture
OGC Data Format standard Version n.m +
Smart Certificate IPT Enabled
OGC Data Format standard Version n.m
Validator

Figure 10 — ogc data formats transition

Transition from an actual OGC API (any) to a new version that is IPT enabled with a Smart contract

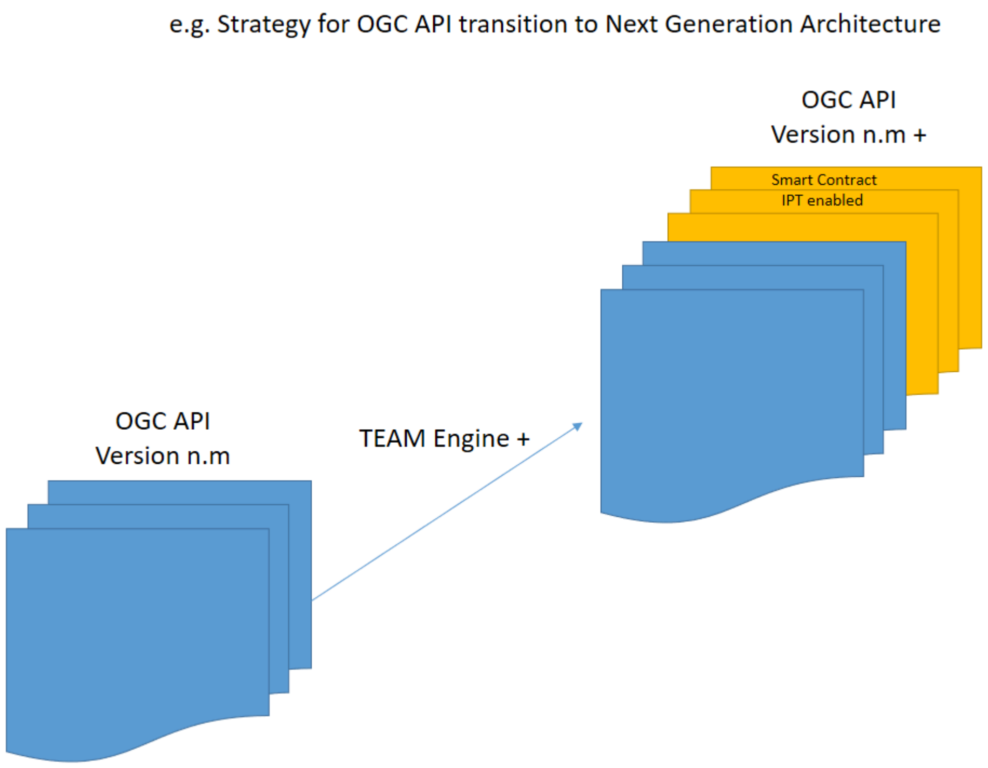


Figure 11 — ogc api(s) transition schema

4.5.1. Possible implementation steps

Several steps for the transition to the next generation architecture are required as follows.

1. Data being moved to the new ecosystem (IPT based). Revision of data formats standards to implement the new ecosystem.

2. Building block as per the definition provided in the previous section to be moved to the new concept. This would require few changes from the current versions if considering implementation of Smart Contract in the registry/orchestrator.

3. A standard Building Block definition based on OGC API standardization work with an approved definition.

4. Registry/orchestrator: Evolution of the OGC WCPS Standard to interact with Smart Contracts, or a confluence between pub/sub and WCPS. For the generation-after-next architecture, a mechanism is envisioned to be implemented using Smart Contract in which BB can actively interact among them and with data (evolution of the WCPS/pubsub standard). Any data processing service MUST record metadata on the processing (details to be defined in the standardization process). This takes place by executing the associated Smart Contract that is deployed on the Fabric Ledger. Also, when a processed data product is published, a Verifiable Attestation is generated that can be used afterwards to validate the Integrity of the data product. So, a WCPS instance executes Smart Contract and

only if that operation is successful does the VA gets created upon making the data product available for supporting verification of the data set.

The data and the OGC API BB are automatically identified through their Smart Certificates and Contracts properties. Solution building blocks (the ones used in the specific workflow) can also be considered as a Smart Contract Component (SCC).

The solution workflow is generated and registered if all the components (data/building blocks) are valid.

A practical example follows: To have the entire process start we need Data IPT enabled. Schemas to issue certificates are registered (e.g. blockchain) and can be as many as required and easily modifable.

Code required to generate an IPT based data (pdf, imagery, vector, processes..) in python

```
from io import BytesIO
            import sys
            import datetime
            import uuid

            from endesive.pdf import cms
            from pypdf import PdfWriter

            date = datetime.datetime.utcnow()
            date = date.strftime("%Y-%m-%dT%H:%M:%SZ")

            with open(out_pdf + ".pdf", "rb") as fh:
                bytes_stream = BytesIO(fh.read())

            writer = PdfWriter(clone_from=bytes_stream)
            writer.add_metadata(
            {
                "/IPT-Identifier": str(uuid.uuid4()),
                "/IPT-Publisher": "EU SatCen",
                "/IPT-Created": date,
                "/IPT-Releaseability": "CC-BY",
                "/IPT-Classification": "ultra open",
                "/IPT-AOI": "POINT(0 0)",
                "/IPT-CRS": "CRS84",
                "/IPT-Holder-DID": "FpsXsfj64R8N5gRYJjPdSE",
                "/IPT-CredentialDefinition": "3zC3MBQ31EV5Wom3UwUamj:3:CL:69:
PDF-1.0"
            }
            )
            writer.write(bytes_stream)
            datau = bytes_stream.getbuffer().tobytes()
            out_pdf = out_pdf.replace(".pdf", "-IPT.pdf")
            with open(out_pdf, "wb") as fp:
                fp.write(datau)
```

Figure 12 — IPTcreation example

If the infrastructure (e.g. blockchain based), data and services (processes) are properly registered to ensure provenance (e.g. registered in the blockchain), then trust of data is automatically enabled. Once the IPT-based infrastructure is in place the records provenance is like the following:

```
{
  "assetId": "ff6d96ea-12ce-4213-9dfd-f92629d820e3",
  "docType": "asset",
  "input": {
    "water": {
      "hash": "1e9d7c27c8bbc8ddf0055c93e064a62fa995d177fee28cc8fa949bc8a4db06f4
",
      "source": "Mangfall",
      "amount": 0.78211224
    },
    "image": {
      "smartCertificate": {
        "cred_def_id": "2hTqXuc1rS5YK4YwKH71Ag:3:CL:178:Teabag-2.0",
        "connection_invite_url": ""
      },
      "hash": "8e8b07089d97a16b1d7b9a23caed98caacb8d0c2858573343ea394ad8d87c08f
"
    }
  },
  "output": {
    "amount": 0.78211224,
    "type": "lemon",
    "brand": "Earl Grey",
    "hash": "3fa87694a746f1d08179c7523d832489124347b619e298154108b67d92a0554d"
  },
  "process": {
    "name": "Trusted Tea and Coffee Pot",
    "description": "I brew coffee and tea from trusted input impages",
    "developer": "Long John Silver",
    "smartCertificate": {
      "cred_def_id": "2hTqXuc1rS5YK4YwKH71Ag:3:CL:178:Teabag-2.0",
      "connection_invite_url": ""
    },
    "hash": "ad8d84f416aee607ecbf459a3d8e881c280b2c0d7c93a49283665ed59927ed16"
  },
  "description": "I just created a product of brand Earl Grey and type lemon",
  "productionDate": "2024-03-09T09:06:58.226005",
  "processingTime": 163.60591,
  "temperature": 99.71069,
  "kind": "TEA"
}
```

Figure 13 — Provenance definition

4.6. Artificial Intelligence

The approach of using Data Centric Security in a IPT makes geospatial systems more reliable and secure. This approach can be applied to any other IT system, thus enabling more reliable solutions for public use as well as moderating AI data production and usage. The information/services properly IPT certified enable derived products and/or services to be easily consumed by users because the products and/or services will be wearing a score about their validity in IPT terms. The score could be represented with a simple 3 color code (e.g., red, yellow, green for fully validated). This aspect obviously has an impact on the entire IT infrastructure and not only for geospatial information.

An important aspect to be considered is the managing of real time data and its streaming. All new architectures should consider the fact that more and more data are available and their quick and reliable managing requires flexible architectures.

4.7. Use cases

In this section possible uses cases for the application of the Agile Reference Architecture are described.

Image processing:

1. Raw image data (reception of new data from a generic sensor, but following Smart Cerificate specification, the data are made available in the communication space).
2. Building block for orthorectification (An OGC API endpoint detects with the Smart Object specification the presence of the data (or through the orchestrator) and makes available the possibility to apply the orthorectification algorithm).
3. Building block histogram equalization (An OGC API endpoint detects with the Smart Object specification the presence of the data (or through the orchestrator) and makes available the possibility to apply the histogram equalization algorithm).
4. Orthorectified and equalized data available as a result of the chaining and with the Smart Certificate defined considering the entire processing workflow.

5

GENERATION AFTER NEXT

5 GENERATION AFTER NEXT

5.1. Open issues

The previously described next-generation architecture leaves open several issues as follows.

- Real time data streaming.
- Quantum Computing/transmission:
    1. sensors (are already available, as well as quantum radar in test phase);
    2. data and processes applied (being the direct communication with the entangled pair, all the actual mechanisms need to be revised); and
    3. quantum computing.
- IPT ready data and Building blocks.
- Federation of systems (authority) collaboration with EU project like EBSI, etc. (Data Act).
- Limited or disconnected environments (the IPT can work standalone with a private key but procedures for checking out/in to be established).

5.2. Future work

5.2.1. Data Centric Security

Extending and applying the Data Centric Security (DCS) scenarios and solutions for authentication, authorization, and cryptographic key exchange from Testbed-18 (OGC 22-014 and OGC 22-018) in the context of Self-Sovereign Identity (SSI) with W3C Decentralized Identifiers (DID) and Verifiable Credentials (VC) ecosystem. For authorization requests, OpenID for Verifiable Credential Issuance may be considered. While OGC 22-014 relies on a Key Management Service (KMS) as the central component, the use of a Decentralized Key Management System [18] may be envisaged for achieving an agile reference architecture relying on OGC API-based building blocks. This topic requires more detailed investigation and prototyping.

5.2.2. Discovery of Decentralized Applications

The current document proposed, in Clause 4.2, `decentralized applications` as a possible path towards a Next Generation Architecture.

Decentralized applications are essentially smart contract-powered applications. Smart contracts are contracts coded and stored on the blockchain. They automate agreements between the creator and recipient. The contract execution is triggered automatically when conditions for contract execution are satisfied. Contract details are eventually recorded on the blockchain ledger as “on-chain” metadata. In such contexts, the integrity and provenance of contract inputs and outputs is provided by the blockchain technology (e.g., Burzykowska et al. [19]).

A prototype implementation of such `Decentralized applications` cooperating to solve a typical use case (e.g., around the D123 process), a detailed analysis as to where the OGC API (Building Blocks) and/or containerized applications and data fit in such `decentralized application`, and determining the remaining role of “discovery” through a registry/catalogue are included in future work.

A

ANNEX A (INFORMATIVE) COMPONENT DELIVERABLES

A ANNEX A (INFORMATIVE) COMPONENT DELIVERABLES

The following sections are descriptions of the work performed during OGC Testbed 19 to show the current state of the architecture and determine shortcomings in design or functionality.

A.1. Component D121: An integrated knowledge base linking machine-readable specifications required to implement the target DDIL Use Case

Two key scenarios were implemented to support the DDIL Use Case.

1. Presentation of example building blocks in an integrated knowledge base.
2. Automatic documentation generation from machine-readable components.

A.1.1. Presentation of example building blocks in an integrated knowledge base

The following assumptions were made about the building blocks:

- building blocks are made available in RDF formats; and
- APIs for the querying and retrieval of RDF can then be used to select building blocks.

To support the Testbed 10 DDIL Use Case, the authors of this ER assert that for a system delivering information on building blocks to be useful, the building blocks must be able to be searched and viewed in domain specific ways, such that a user can make sense of and assemble a domain specific architecture composed of building blocks.

OGC provides a number of existing standards which offer methods in which spatial data and metadata can be queried. These include the OGC API — Features Standard and OGC API — Records Candidate Standard and specifically, one component: the Common Query Language version 2 (CQL2). CQL is a formal language for representing queries to information retrieval systems such as web indexes, bibliographic catalogs, and museum collection information. As the building blocks are expressed as RDF, it is worth exploring whether existing systems which seek to deliver spatial data in an OGC standards conformant manner can be extended to also deliver the building blocks themselves.

The benefits of this approach are numerous.

- Full provenance and interrogation of the reference building blocks can be made available alongside data delivered in conformance with the building blocks; at a minimum data or systems can reference the building blocks.
- Granular parts of data or services can reference parts of the building blocks.

The set of basic use cases that can be supported by this approach are:

1. a user can search for building blocks by name, description, or other metadata;
2. a user can search for building blocks by spatial extent;
3. a user can search for building blocks by temporal extent;
4. a user can view a building block in a domain specific way, for example as a map or as a table; and
5. a user can view a set of building blocks in a conformance specific way.

The Testbed participants attempted to demonstrate that the information *required* to generate a map or table in a domain specific way can be returned — ignoring any specific implementation of the map or table itself.

To explore the above use case, an existing system that the Testbed participants are familiar with, Prez was extended to be compliant with the draft OGC API — Records Standard. Compliance with OGC API — Records covers use cases 1-3, and existing functionality around profiles, built to the ConnegP Recommendation, through extension, covers use cases 4 and 5.

To achieve OGC API — Records compliance, the participants determined the best method would be to provide a CQL2 to SPARQL conversion. Initially, this was implemented in the D122 use case using RDFrame, to assist in confirming that the profiling was correct. The mapping from CQL2 to SPARQL was done via a common Python model, using Pydantic (validation library for Python). This model allows for the creation of SPARQL queries using Jinja2 templates based on a limited set of parameters targeted at describing objects, listing objects, and profiled views of the object. Jinja2 is a web template engine for the Python programming language. To achieve CQL2 compliance specifically, the test cases from the CQL specification were used to drive the development of the model. These test cases are found here Building Blocks as JSON, within the OGC API — Features Standard, which is referred to in the draft OGC API — Records Standard.

A.1.2. Demonstration system showing Building Blocks in a navigable knowledge base

The demonstration system is available here

The system is based on the following components.

- A FastAPI based API, which provides the RDF content at endpoints which are conformant with, and an extension to, the draft OGC API — Records Standard. FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.8+ based on standard Python type hints. As part of the Records compliance, CQL2 filtering is provided at the individual API endpoints.
- A CQL2 endpoint is provided, specifically for CQL2 JSON, to support more complex or longer queries to be sent to a server via the API. This allows CQL2 queries to be made against the RDF content, including spatial, temporal, and queries which filter on properties.

A.1.3. Auto generation of documentation from machine-readable components

Machine-readable building blocks as RDF also include human-readable properties (via “annotations”), that is, labels, descriptions, and other literals which are human-readable text. As the human and machine-readable formats are tied together, the ability to programmatically combine building blocks based on user defined inputs, profiles, or even manual selections is possible.

The PyLode tool was used for the creation of human-readable specifications directly from RDF Ontologies (in place of building blocks in the current context). That is, a nicely formatted printout of classes and properties within an ontology. For Testbed 19, the participants sought to extend this tool to provide programmatic capability to support the ability to dynamically construct specifications based on logic, profiles, or other user defined inputs. The changes to PyLode to support the DDIL Use Case are described below.

A “Supermodel Profile” designed to represent multiple ontologies as a unified “supermodel” was introduced prior to Testbed 19 — it provided a model basis but did not automate the creation of documentation which used this model. It was demonstrated through the creation of an updated Cadastral Survey Data Model for New Zealand, which was demonstrated within the testbed. Within the current testbed, a profiling mechanism was introduced, allowing for modular and scalable ontology documentation. At its core, these profiles enable users to construct detailed models from components.

The key model changes/additions are summarized in the table below.

Table A.1

PROPERTY	DESCRIPTION
`prof:isProfileOf`	Denotes the association of a component to its overarching profile
`lode:componentModel`	Defines a “Component Model,” a model that is a component of a broader model, such as a supermodel.

PROPERTY	DESCRIPTION
`lode:ignoreClass`	Signals specific classes to be overlooked during the documentation process.
`lode:isQualifiedProfileOf`	Links to qualified nodes that are associated with a profile, ensuring only relevant classes within a profile are loaded.

By emphasizing a component-driven design, the profiles feature ensures that ontology models have the flexibility to be both intricate and organized.

A.2. Component D122: The Agile Reference Architecture represented in RDF/Turtle format

The Agile Reference Architecture represented in RDF/Turtle format (i.e., a description of how the components and specifications are related in the form of a reusable pattern that can be adapted to new circumstances)

A.2.1. Introduction

In Testbed 19, the participants demonstrated the inherent interoperability of building blocks expressed in RDF/Turtle format. The participants demonstrated a new tool developed in this Testbed, RDFrame. RDFrame supports the composition and viewing of the implementations of the reference architecture described in this Engineering Report. Concretely, RDFrame provides profiled views of building blocks according to programmatic logic, either explicit interoperability declarations, or more complex, rule based logic.

The participants further explored the extension of existing documentation tooling for the automated creation of building blocks.

The participants also demonstrated a use case for maintaining the building blocks in RDF/Turtle format: composition of human-readable specifications from the machine-readable building blocks. As part of this process, specification of the relevant parts (also expressed as data) allows the creation of a domain specific architecture.

A.2.2. RDF Context

In many symbolic systems, context provides meaning to symbols, allowing them to gain meaning beyond their individual representation. For geospatial data, each resource in RDF — whether a datum, process, or standard — can be viewed as such a symbol. Using RDF metadata, these resources self-define, indicating their roles, relationships, and functions. Notably, there is no need for an external reference architecture: The building blocks themselves convey compatibility and together establish the architecture.

The interoperability of the building blocks in general is supported by the interoperability of the underlying linked data standards, which are themselves building blocks, and allow for arbitrary extensibility. At its core, RDF allows "anyone to make a statement about anything." There is a loose distinction between metadata and data — both are represented in the same way. In a very direct sense, the statement, "David's report has content about RDF," and the time at which such a statement was made would be recorded in an RDF database within the same (and only) kind of data structure, a triple. In this context, more effort can be placed on reusing existing standards and building blocks, rather than creating new ones, and only revising or extending the more general use cases to support project specific needs. This prevents the proliferation of new standards, data silos, and non-interoperable systems.

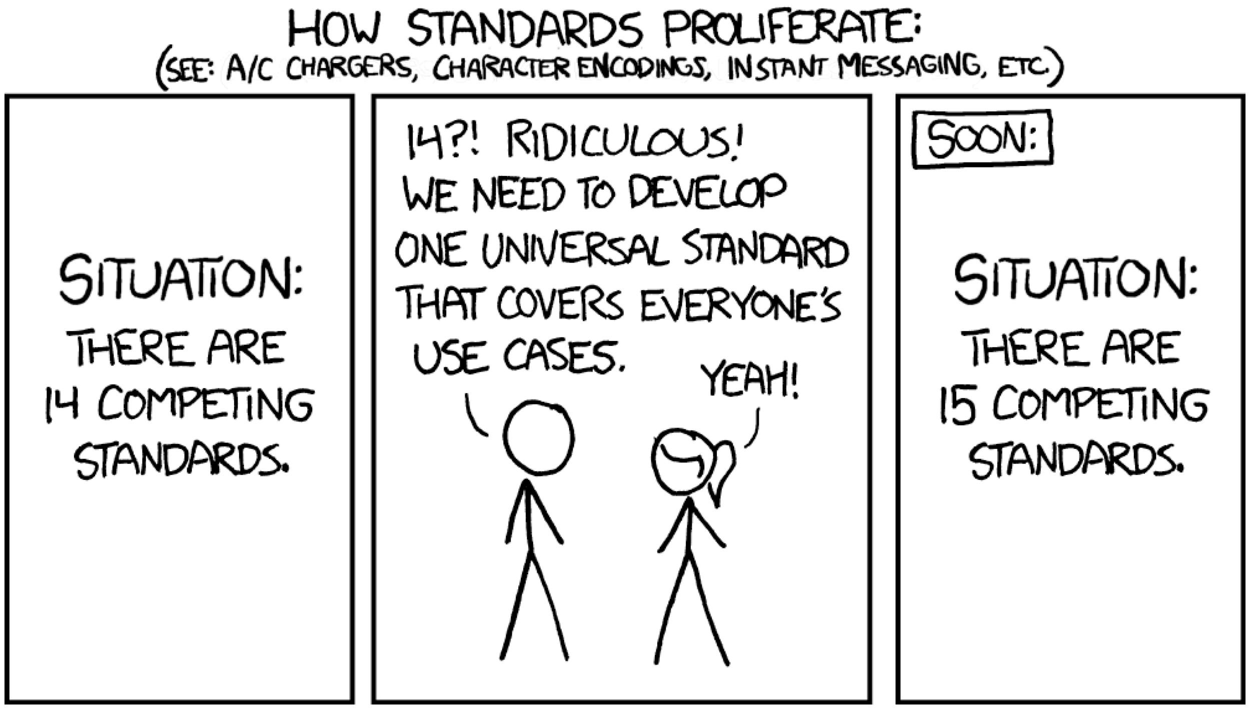

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Figure A.1

A.2.3. Reusability

While the task of specifying how and where specific building blocks can be used, and therefore reused, is left to domain experts, some level of declarative reusability statements between building blocks is assumed. That is to say, a given set of building blocks, to be reused must all be compatible with each other. This could take a number of forms, a given building block "A" could declare it is compatible with building block "B" only, or building blocks of type "Class 1" could declare they are compatible with building blocks of type "Class 2." More complex statements including conditional logic could also be considered, however these are not considered here. Some basic use cases are shown below.

The machine readability of RDF supports the use of automated reasoning to determine the reusability of building blocks. This is a key advantage of using RDF as a representation format, as it allows for the use of automated reasoning to determine the reusability of building blocks. The work done in D121 is related. In this work, software was developed to automatically create a human-readable standard from a set of building blocks. There was an implicit or assumed compatibility between the building blocks, which is based on the knowledge of the user creating the standard. In this Testbed, as in deliverable D122, the participants explored making this compatibility explicit. Use cases of the machine readability could include validation of specific architectures composed of building blocks, and recommendations of building blocks to use in a given context.

To explore this idea further, a few illustrative examples of the compatibility are represented below.

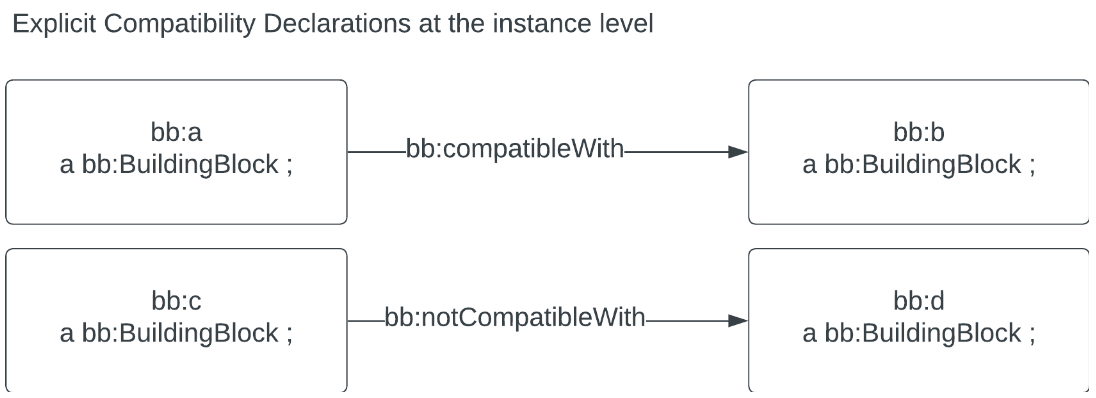


Figure A.2

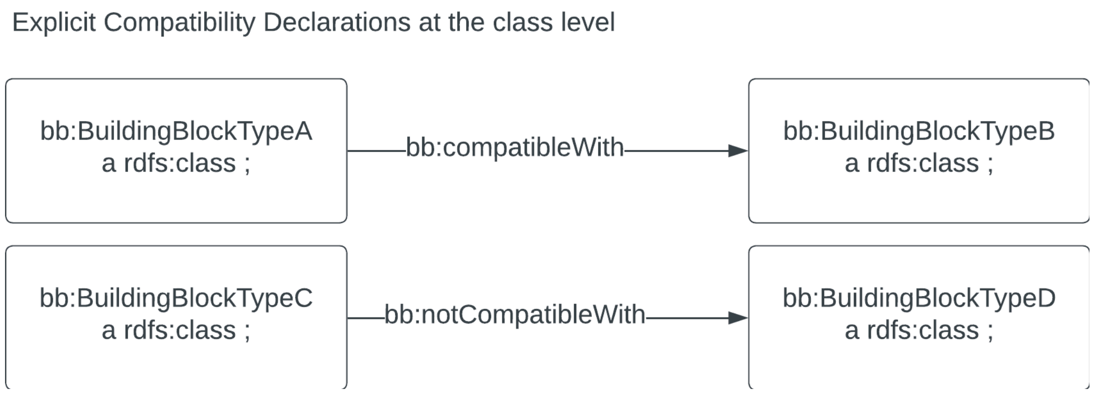
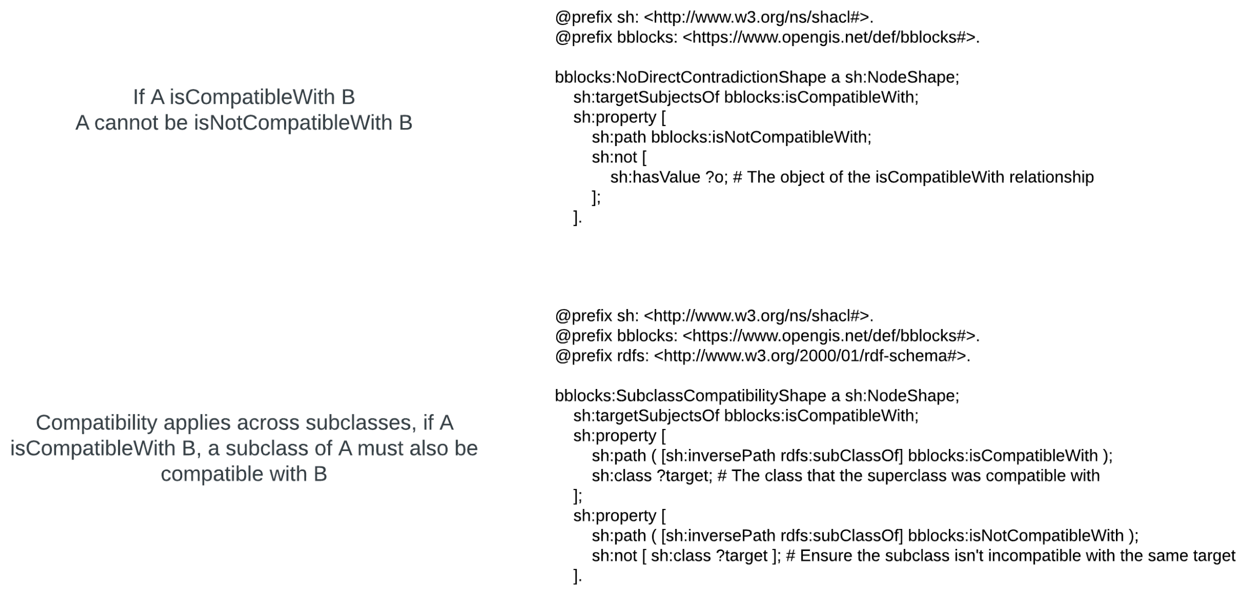


Figure A.3

The building blocks can then be tested for compatibility using a SPARQL query, and the validation or rules can be represented in the Shapes Constraints Language, or SHACL. However, this is not strictly necessary with the explicit compatibility declarations given above. The SHACL rules below provide logic to test the compatibility of building blocks in the two scenarios outlined above.



If A isCompatibleWith B
A cannot be isNotCompatibleWith B

```
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix bblocks: <https://www.opengis.net/def/bblocks#>.

bblocks:NoDirectContradictionShape a sh:NodeShape;
    sh:targetSubjectsOf bblocks:isCompatibleWith;
    sh:property [
        sh:path bblocks:isNotCompatibleWith;
        sh:not [
            sh:hasValue ?o; # The object of the isCompatibleWith relationship
        ];
    ].
```

Compatibility applies across subclasses, if A isCompatibleWith B, a subclass of A must also be compatible with B

```
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix bblocks: <https://www.opengis.net/def/bblocks#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

bblocks:SubclassCompatibilityShape a sh:NodeShape;
    sh:targetSubjectsOf bblocks:isCompatibleWith;
    sh:property [
        sh:path ( [sh:inversePath rdfs:subClassOf] bblocks:isCompatibleWith );
        sh:class ?target; # The class that the superclass was compatible with
    ];
    sh:property [
        sh:path ( [sh:inversePath rdfs:subClassOf] bblocks:isNotCompatibleWith );
        sh:not [ sh:class ?target ]; # Ensure the subclass isn't incompatible with the same target
    ].
```

Figure A.4

Compatibility tooling was developed to demonstrate how machine-readable definitions of compatibility can be used dynamically to query conformance between building blocks. The tool developed for Testbed 19 was called RDFrame, suggesting that the tool frames or profiles RDF in a particular way, similar to JSON-LD framing. A simple UI was created for the tool, comprising the following displays.

1. Runtime information for user interfaces where building blocks are browsed, or otherwise dynamically delivered by an API, provides context such as search terms, specific building block identifiers for retrieval, and spatially or temporally scoped information.

2. The selection of “target” or “focus nodes” in RDF: Within the scope of this ER, these would primarily be building blocks expressed in RDF. This is primarily done with `sh:targetNode` where the building block of interest is known; sh:targetClass where building blocks of a particular Class or Subclass are known, and the use of more detailed subqueries where a more complex description of the building blocks to be selected is required.

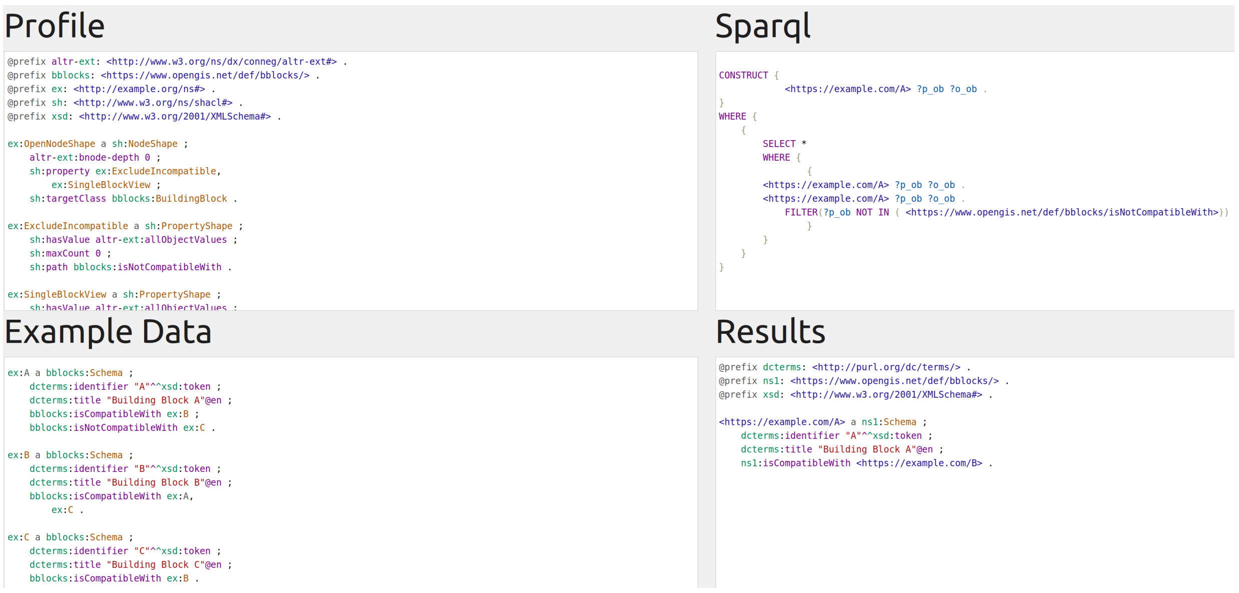
3. A profile definition in RDF: The profiles are used to shape descriptions of Building Blocks, or related entities, based on the selected “nodes” or “building blocks” from the first point. This again primarily uses SHACL.

4. A SPARQL query generated from the target selection, profile and runtime information.

5. Example data — in this case the reference architecture building blocks.

6. The resulting “Framed” or “Profiled” RDF is shown in Figure A.5. Not shown in the quadrants is the supply of runtime information. This will be added as an enhancement to the tool. Runtime information could include a “focus block” which could provide information according to a particular profile. One example

is a class of building block which could list instances. Another is a “focus block”, from which, by relationships, could list related blocks. This runtime information is synonymous with the profiles themselves with the difference being this information is typically only known “at runtime” or when a user interacts with the system, whereas the profiles define “general saved view” that users find useful. A screenshot of the tool is shown below.



Profile

```
@prefix altr-ext: <http://www.w3.org/ns/dx/conneg/altr-ext#> .
@prefix bblocks: <https://www.opengis.net/def/bblocks/> .
@prefix ex: <http://example.org/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:OpenNodeShape a sh:NodeShape ;
    altr-ext:bnode-depth 0 ;
    sh:property ex:ExcludeIncompatible,
        ex:SingleBlockView ;
    sh:targetClass bblocks:BuildingBlock .

ex:ExcludeIncompatible a sh:PropertyShape ;
    sh:hasValue altr-ext:allObjectValues ;
    sh:maxCount 0 ;
    sh:path bblocks:isNotCompatibleWith .

ex:SingleBlockView a sh:PropertyShape ;
    sh:hasValue altr-ext:allObjectValues ;
```

Sparql

```
CONSTRUCT {
            <https://example.com/A> ?p_ob ?o_ob .
}
WHERE {
    {
        SELECT *
        WHERE {
                {
        <https://example.com/A> ?p_ob ?o_ob .
        <https://example.com/A> ?p_ob ?o_ob .
            FILTER(?p_ob NOT IN ( <https://www.opengis.net/def/bblocks/isNotCompatibleWith>))
                }
        }
    }
}
```

Example Data

```
ex:A a bblocks:Schema ;
    dcterms:identifier "A"^^xsd:token ;
    dcterms:title "Building Block A"@en ;
    bblocks:isCompatibleWith ex:B ;
    bblocks:isNotCompatibleWith ex:C .

ex:B a bblocks:Schema ;
    dcterms:identifier "B"^^xsd:token ;
    dcterms:title "Building Block B"@en ;
    bblocks:isCompatibleWith ex:A,
        ex:C .

ex:C a bblocks:Schema ;
    dcterms:identifier "C"^^xsd:token ;
    dcterms:title "Building Block C"@en ;
    bblocks:isCompatibleWith ex:B .
```

Results

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ns1: <https://www.opengis.net/def/bblocks/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://example.com/A> a ns1:Schema ;
    dcterms:identifier "A"^^xsd:token ;
    dcterms:title "Building Block A"@en ;
    ns1:isCompatibleWith <https://example.com/B> .
```

Figure A.5 — Example of profiled RDF and associated SPARQL

A.2.4. Adaptation to new circumstances

Use of RDF as the Agile Reference Architecture allows for reuse of the Simple Knowledge Organization System, or SKOS. SKOS is an established ontological framework for modeling concepts, schemes, and collections of these. The reference building blocks supplied are, in fact, classed both building block types and concepts providing an immediate framework to extend building blocks to incorporate existing knowledge or concepts.

As the data and metadata is expressed in RDF, both are capable of being used with APIs, enhancing ‘queryability’ across large collections of building blocks, including spatial, free text, and organizational hierarchies (such as OGC API — Records and OGC API — Features, and general ontologies such as SKOS.)

A.3. Component D123: An instance of OGC API – Processes

This section discusses the context, plan, and motivation for the enhancement of the OGC API — Processes component within the realm of the Agile Reference Architecture.

An activity included in this Testbed was the implementation of an instance of OGC API - Processes for demonstrating the improvement in a Technology Integration Experiment (TIE) which is described later in this section.

A.3.1. Context

The OGC API — Processes Standard plays a pivotal role in offering a standardized interface for the discovery, execution, and retrieval of processing functionalities.

A.3.1.1. Target Objectives

An Agile Reference Architecture aims to be adaptable, modular, and future-proof. From the range of activities outlined for research in the Testbed 19 ARA thread, including resilience, universal access, etc., the participants specifically focused on addressing integration, interoperability, and the transformation of data into locally-useful forms.

- Integration ensures that different parts of an ecosystem can work harmoniously, sharing data and executing functions without redundancy or conflicts.
- Interoperability extends beyond integration. It ensures that different systems, perhaps developed independently and with varying standards, can communicate and work together. In the context of geospatial data and services, this means that data and services from different sources or standards can be combined, processed, and presented uniformly.

With the proliferation of data sources in diverse formats and from varied origins, the challenge is not just collecting data, but transforming it into a **locally-useful form**. This transformation is not merely about data conversion, but about ensuring that the transformed data is meaningful, actionable, and optimized for the local context in which it is used.

A.3.1.2. OGC API — Processes Enhancement

Given the potential and existing challenges of the OGC API — Processes, the focus of the participants in this Testbed ARA thread included the following.

- Dynamic Chaining of Processes: Infusing agility principles into the OGC API - Processes Standard to support dynamic chaining, leveraging abstraction layers (building blocks) atop I/O, and enabling protocol negotiation for enhanced interoperability.

- Interoperability of I/O Formats: Addressing the potential challenges when chaining nearly compatible processing units and ensuring smooth data format transitions without necessitating application modifications.
- Data Transformation: Ensuring that data can be seamlessly transformed between heterogeneous sources into locally-useful formats, considering potential needs for reformatting or scaling.

This refined focus was derived from weekly meetings in which the participants established a clear scope for the Deliverable OGC API — Processes implementation (D123) and identified concrete enhancements that contribute to the review of a new agile architecture.

A.3.1.3. Motivation

A.3.1.3.1. Strengths of OGC API — Processes

The current OGC API — Processes Standard has several strengths as follows.

1. Discoverability: Users can effortlessly discover services. For instance, once Alice (a user persona) registers an application in an OGC application package, Bob (another user persona), without any in-depth knowledge of its implementation, can discover and interact with the service.
2. Execution: An implementation of the OGC API — Processes Standard provides a standardized operation for executing processes. This process handles the intricate details, including staging of inputs, scheduling, execution within the container, and results stage-out.
3. Retrieval: Users can retrieve results in a streamlined and consistent manner, ensuring the users always know where and how to obtain their data.

A.3.1.3.2. Gaps for chaining processes

The OGC API — Processes Standard also has some weaknesses. The Standard falls short when providing requirements and guidance for **chaining processes** that are nearly compatible but have slight differences. For instance, the passing of a subset region of a coverage to a second process is not supported.

- Interoperability of I/O Formats: Different processes may have distinct input and output formats. For instance, the ability to chain a process that produces an output in the Esri Shapefile (SHP) format with another one expecting GeoJSON as input is not straightforward.
- Data Subsetting: There are instances where only a temporal or spatial subset (or other tailoring) of the data might be necessary. The current OGC API — Processes Standard does not inherently cater to such subsetting needs.

A.3.1.3.3. Fictional Use Case: URBA and CITYSTATS

To illustrate the gaps, consider a chaining scenario involving two fictional processes: URBA and CITYSTATS:

URBA: — Description: An environmental analysis software application tracking urbanization in cities. — Input: Vector representations (roads, buildings, parks, etc.) spanning the last 20 years. — Output: A heatmap pinpointing rapid urban growth areas, provided in SHP file format.

CITYSTATS: — Description: A geospatial tool yielding neighborhood project stats. — Input: Various data, including URBA's heatmap, but in GeoJSON format. — Output: A project score, presented in TXT format.

When attempting to chain URBA and CITYSTATS, the issue arises from the differing I/O formats. While URBA outputs a SHP file, CITYSTATS expects GeoJSON as input. The current OGC API — Processes, Standard does not easily support such chaining without extensive modifications or interventions from developers.

A.3.2. Proposed Solution

The goal of the participants was to draft and prototype the enhancement of the current capabilities of the OGC API — Processes Standard to align with the objectives of the Agile Reference Architecture. This alignment requires a solution that addresses the complexities of process chaining, data transformation, and interoperability. The detailed approach is described in the following subsections.

A.3.2.1. Integration of OGC Building Blocks

> **Note**: For the purpose of this ER and this aspect of the Testbed 19, a `building block' strictly adheres to the OGC's official definition presented in Clause 3.3.1.

Within the context of a processes chain, a building block can play a critical role in facilitating the interoperability of processes. By supporting building blocks as input, a system would have a standardized method to access and manipulate data (still enabling the process native format).

The results of each process can be provisioned as separate building blocks, ensuring ready availability for subsequent processes or data transformations.

The `building blocks' provide an abstract data store as they allow data to be stored in a format-agnostic manner, making it more adaptable to varying processes. With data stored as building blocks, operations such as subsetting or reformatting can be handled more easily.

A.3.2.2. Provisioning of Transient Building Blocks

The solution relies on a set of assumptions for the building block components that can be provisioned with the output data of processes.

First, it is crucial for the provisioned building block components to serve data dynamically. As processes generate outputs, these outputs can be immediately provisioned as building blocks without waiting for batch operations or manual intervention.

This ensures that the vector data can be dynamically mounted to and from containers.

Also, the components should handle a variety of native formats, granting us greater flexibility in process chaining.

A.3.2.3. Implementation of a Processing Server

The server implementation must be designed to pull data from building blocks, exposing the building-block specific parameters of the API. However, the data should be retrieved in the correct local format required for any given process.

Once a process has been run, the server will take the outputs and dynamically provision an OGC API Features building block, ensuring the results are readily accessible.

A.3.3. Implementation

The Testbed-19 OGC API — Processes Proof of Concept (PoC) is a comprehensive implementation designed to demonstrate the new capabilities elaborated above, based on the OGC API — Processes Standard version 1.0.0.

The following operations are integral to this implementation.

- List Processes (`GET /processes`): This operation enumerates all the processes the server supports, acting as an entry point for users to explore available functionalities.
- Describe Process (`GET /processes/{id}`): This operation, associated with the HTTP resource `GET /processes/{id}`, where `{id}` is the unique identifier for a process, provides detailed information about a specific process, which is essential for users to understand process capabilities and requirements.
- Execute (`POST /processes/{id}/execution`): Triggered by `POST /processes/{id}/execution`, where `{id}` identifies the process, this operation initiates the execution of a process. It requires a JSON-formatted payload detailing the necessary inputs and desired outputs for the process. Inputs can be either direct values or references, while outputs specify the data types expected from the process execution.
- Jobs Status (`GET /jobs/{id}`): Linked to `GET /jobs/{id}`, with `{id}` being the job identifier, this operation provides real-time status updates for a job. The status codes

include ‘accepted,’ ‘running,’ ‘successful,’ ‘failed,’ or ‘dismissed,’ offering clear insights into the job’s lifecycle.

- Jobs Results (`GET /jobs/{id}/results`): Associated with `GET /jobs/{id}/results`, where `{id}` is the job identifier, this operation is used to retrieve the results post job execution. The results may be provided as either inline data or as references (links) to the data.
- List Jobs (`GET /jobs/`): This operation lists all jobs, both currently executing and previously executed by the server, providing a comprehensive view of server activity.
- Cancel Job (`DELETE /jobs/{id}`): Corresponding to `DELETE /jobs/{id}`, this operation specifies the dismissal of an ongoing job execution, effectively canceling the job execution and marking its status as ‘dismissed.’

A.3.3.1. Design Overview

The OGC API — Processes implementation is a microservice written in Java and comprises the following architectural components.

- Java Application: This application utilizes the Spring Boot Framework (version 3.0.0) for its simplicity and efficiency serving as the core of the implementation, facilitating the execution of various operations defined by the OGC API — Processes Standard.
- Data Persistence & PostgreSQL: Hibernate is employed for database operations. Hibernate facilitates the management and persistence of data, offering a powerful, high-performance Object/Relational persistence and query service.
- Kubernetes Integration: Fabric8 toolkit for Kubernetes simplifies the deployment and management of Kubernetes resources, streamlining the process of running and monitoring Kubernetes jobs which execute the core processes.
- LDProxy Server: LDProxy is an OGC API — Features implementation which steps in to expose and manage the source and the resulting geographic features, acting as a bridge between the Kubernetes jobs and end-user accessibility.

The entire suite, including the OGC API — Processes, PostgreSQL database, and LDProxy server, is seamlessly deployed on a Kubernetes cluster hosted at Spacebel.

The following specific Docker images were used.

- LDProxy: `iide/ldproxy:3.5.0`
- PostgreSQL: `postgres:15-alpine`
- Spring Boot Application: `eclipse-temurin:17-jre-alpine`

A.3.3.2. Components Configuration

The OGC API — Processes server is configured with the following properties.

- SPRING_DATASOURCE_URL: The JDBC URL used to connect to the PostgreSQL database. A JDBC URL provides a way of identifying a database so that the appropriate driver recognizes and connects to the database. KUBECONFIG: The Kubernetes configuration file used to connect to the Kubernetes cluster in order to manipulate K8S resources (Jobs).

The LD Proxy server requires two volume mounts:

- /ldproxy/data/api-resources/features: path used to store the features exposed by the server; and
- /ldproxy/data/store: path used to store the configuration files used to expose the features.

For each feature exposed by the LD proxy server, two configuration files must be provided: - /store/entities/features/providers/my-product.yml : configuration file describing the feature; and - /store/entities/features/services/my-product.yml: configuration file describing the service used to expose the features.

The dataset used in the frame of the proof of concept is the 'Daraa' dataset. This is a test dataset was used in the Open Portrayal Framework thread in OGC Testbed-15. The dataset is based on OpenStreetMap data from the region of Daraa, Syria, converted to the NGA Topographic Data Store schema.

A.3.3.3. Reproject Process

The "Reproject" process in the Testbed-19 OGC API — Processes Proof of Concept is designed to transform geospatial features from one coordinate reference system (CRS) to another. This process is critical in scenarios where data interoperability and integration across different geospatial data sources are needed. Implementation of process ensures that geospatial features can be utilized effectively across various systems and applications.

The process accepts two types of inputs.

- Input-features consists of geospatial features provided in the GeoJSON format.
- Reprojection-code defines the target CRS to which the input features will be transformed. Typically represented as an EPSG code, this string input directs the process in aligning the features to the desired geospatial reference framework.

```
{
    "title": "Reproject process",
    "description": "Reproject features",
    "keywords": ["geopackage","geojson","reproject"],
    "id": "reproject",
    "version": "1.0",
    "jobControlOptions": ["sync-execute"],
    "outputTransmission": ["value"],
    "inputs": {
        "reprojection-code": {
            "title": "reprojection-code",
            "keywords": [
                "input"
            ],
            "minOccurs": 1,
            "maxOccurs": 1,
            "schema": {
                "type": "string",
                "contentMediaType": "plain/text",
                "contentEncoding": "text"
            }
        },
        "input-features": {
            "title": "input-features",
            "keywords": [
                "input"
            ],
            "minOccurs": 1,
            "maxOccurs": 10,
            "schema": {
                "type": "string",
                "format": "ogc.geo.features.featureCollection",
                "contentMediaType": "application/geo+json",
                "contentEncoding": "text"
            }
        }
    },
    "outputs": {
        "reprojected-features": {
            "title": "reprojected-features",
            "description": "reprojected features",
            "keywords": [
                "output"
            ],
            "schema": {
                "type": "string",
                "format": "ogc.geo.features.featureCollection",
                "contentMediaType": "application/geopackage",
                "contentEncoding": "binary"
            }
        }
    }
}
```

Figure A.6

A.3.3.4. Execution Sequence

The execution of the “Reproject” process aligns with the Agile Reference Architecture’s objectives, such as integration, interoperability, and transforming data into locally-useful forms.

Initially, users select the input features through the LD-Proxy web interface, accessing the Daraa test dataset. This platform allows for nuanced selection based on geographical zones or time periods. The chosen GeoJSON URL from this selection process becomes the `href` value in the payload for `input-features.`

The users also define the reprojection parameters by specifying the desired CRS. While EPSG:4326 is the default, the parameters can be modified to any preferred CRS.

```
{
    "inputs":{
        "input-features": {
            "href": "http://172.17.20.10:30088/rest/services/daraa/collections/
AeronauticCrv/items?f=json&bbox=36.4005%2C32.6950%2C36.4111%2C32.7047&datetime=
2011-03-16T14%3A51%3A12Z%2F2013-12-27T12%3A47%3A07Z"
        },
        "reprojection-code": "EPSG:4296"
    },
    ...
}
```

Figure A.7

Upon completion of these steps, an HTTP POST request is made to `/processes/reproject/execution` with the defined payload. This action initiates the backend process where a Kubernetes job is launched. This job is responsible for:

- downloading the GeoJSON resource;
- reprojecting the data according to the specified CRS using GDAL ogr2ogr command;
- storing the reprojected data in a new volume; and
- provisioning a new instance of IDProxy server based on the newly created volume.

The response of the server includes a URL pointing to the reprojected features. This URL can be accessed via a web browser to visualize the transformed data.

```
{
    "reprojected-features": {
        "href": "http://172.17.20.10:30188/rest/services/newproduct/
collections"
    }
}
```

Figure A.8

A.4. Component D124: An instance of OGC API – Features serving OpenStreetMap data

A.4.1. Overview

The requirement for this deliverable was:

> An instance of OGC API — Features serving OpenStreetMap data represented according to the NSG Topographic Data Store schema

interactive instruments provided two API instances that implemented OGC API — Features Standard to access OpenStreetMap data. One of the API endpoints represented the OpenStreetMap data according to the NSG Topographic Data Store schema.

Both API implementations use ldproxy, an OGC Reference Implementation for the OGC API — Features Standard.

A.4.2. Daraa API

A.4.2.1. Overview

Link to the API Landing Page

The first API instance is an existing Features API implementation that interactive instruments maintains as one of the ldproxy demonstration APIs. The Daraa dataset has been used in previous OGC testbeds and pilots. It is based on OpenStreetMap data from the region of Daraa, Syria, converted to the NGA NSG Topographic Data Store schema.

This API implementation also provides resources from the OGC API — Tiles Standard and the draft OGC API — Styles Standard.

A.4.2.2. Dataset

Download as GeoPackage

The Daraa dataset was provided by the US National Geospatial Intelligence Agency (NGA) for development, testing, and demonstration in initiatives of the Open Geospatial Consortium (OGC). For any reuse of the data, please contact NGA.

A.4.2.3. ldproxy Configuration

To deploy one or more APIs with ldproxy, configuration files for the deployment are needed. The configuration is typically maintained in a git repository.

For the demo.ldproxy.net deployment, which includes the Daraa API, the configuration is available at https://github.com/ldproxy/demo.

Since it is a demonstration deployment, the repository also contains additional documentation about the different APIs.

A.4.2.4. Deployment

ldproxy is only distributed as a Docker image. The configuration repository also includes a Docker Compose file to simplify the process of starting a local deployment.

A.4.3. Tunisia API

A.4.3.1. Overview

Link to the API Landing Page (requires credentials)

The second Testbed 19 ARA API implementation is based on OpenStreetMap data from Tunisia, converted to a variant of the MGCP schema used by European Union Satellite Center (SatCen).

A.4.3.2. Dataset

The Tunisia dataset was provided by SatCen to interactive instruments for use in Testbed-19 in the Esri File Geodatabase format. This dataset is not publicly available.

A.4.3.3. Initial ldproxy Configuration

Initially, a minimal configuration was created, consisting of the following files:

```
api-resources/features/tunisia0523.gpkg
cfg.yml
store/entities/providers/tunisia.yml
store/entities/services/tunisia.yml
```

Figure A.9

For more information about ldproxy configurations, see the ldproxy documentation.

The GeoPackage file `api-resources/features/tunisia0523.gpkg` was generated from the Esri File Geodatabase using GDAL.

```
store:
 mode: READ_ONLY
server:
  externalUrl: ${EXTERNAL_URL:-https://t19.ldproxy.net}
logging:
  level: ERROR
  appenders:
    - type: console
      timeZone: Europe/Berlin
  loggers:
    de.ii: INFO
```

Figure A.10 — Global configuration (`cfg.yml`)

```
id: tunisia
entityStorageVersion: 2
providerType: FEATURE
providerSubType: SQL
connectionInfo:
  database: api-resources/features/tunisia0523.gpkg
  dialect: GPKG
auto: true
```

Figure A.11 — Feature provider (`store/entities/providers/tunisia.yml`)

```
id: tunisia
serviceType: OGC_API
entityStorageVersion: 2
metadata:
  attribution: "Copyright \xA9 by the European Union Satellite Centre (EU
SatCen), 2023. All rights reserved."
  contactEmail: portele@interactive-instruments.de
  contactName: Clemens Portele, interactive instruments GmbH
  creatorName: European Union Satellite Centre (EU SatCen)
  licenseName: All rights reserved
  publisherName: interactive instruments GmbH
  publisherUrl: https://www.interactive-instruments.de
auto: true
```

Figure A.12 — API building blocks (`store/entities/services/tunisia.yml`)

A.4.3.4. Initial Deployment

The following Docker Compose file was used to test a local deployment.

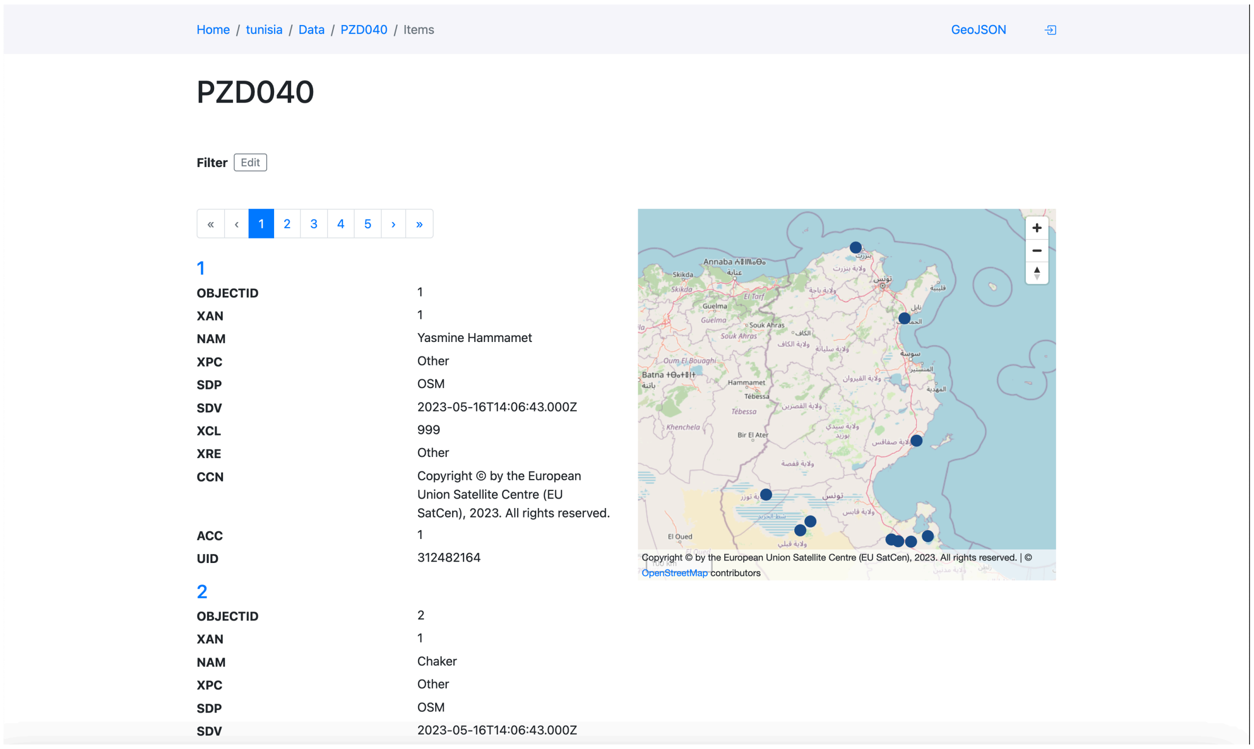
```
version: '3.9'
services:
  ldproxy:
    image: iide/ldproxy:next
    container_name: ldproxy_t19
    ports:
      - "7080:7080"
    volumes:
      - .:/ldproxy/data
    environment:
      - EXTERNAL_URL=http://localhost:7080/rest/services
```

Figure A.13 — Docker Compose file (`docker-compose.yml`)

When starting the API instance with `docker compose up`:

- the feature types with their properties were derived from the database; and
- the default ldproxy API building blocks were enabled (basically all building blocks from OGC API — Features Part 1 and 2) with their default configuration.

The API endpoint was then ready to serve the feature data. The following is a screenshot of the first HTML page of collection PZD040:



Home / tunisia / Data / PZD040 / Items   GeoJSON

PZD040

Filter Edit

« ‹ 1 2 3 4 5 › »

1

OBJECTID	1
XAN	1
NAM	Yasmine Hammamet
XPC	Other
SDP	OSM
SDV	2023-05-16T14:06:43.000Z
XCL	999
XRE	Other
CCN	Copyright © by the European Union Satellite Centre (EU SatCen), 2023. All rights reserved.
ACC	1
UID	312482164

2

OBJECTID	2
XAN	1
NAM	Chaker
XPC	Other
SDP	OSM
SDV	2023-05-16T14:06:43.000Z

Copyright © by the European Union Satellite Centre (EU SatCen), 2023. All rights reserved. | © OpenStreetMap contributors

Figure A.14 — Initial PZD040 features representation

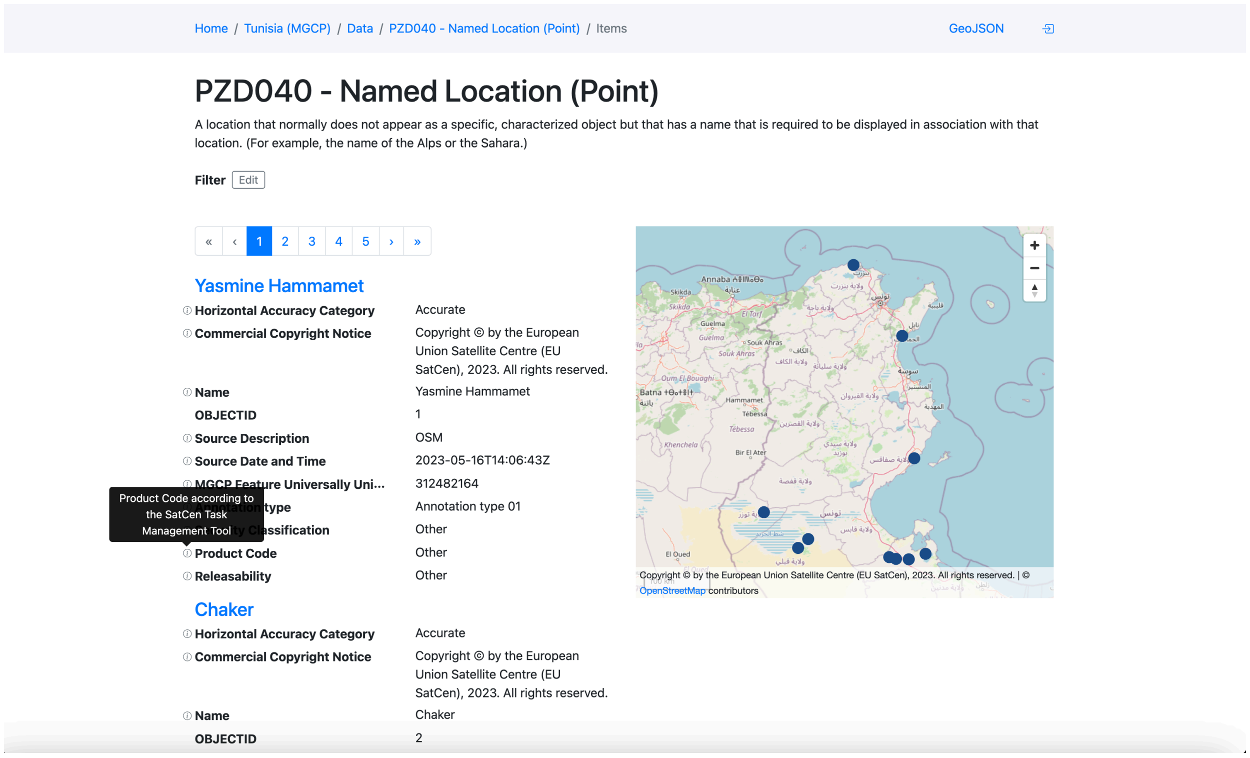
Such an API is only useful, if the user / client understands what a PZD040 feature is or what the meaning of an attribute ACC with a value 1 is.

A.4.3.5. Updated configuration and deployment

SatCen maintains a data dictionary for their datasets that provides this schema information. The data dictionary was provided to interactive instruments as a XML file.

Using a Python script, the configuration for the feature types was updated to include titles and descriptions for feature types and attributes, constraints on attributes and the codelists for coded values. In addition, additional API building blocks were enabled.

After restarting the API endpoint, the first HTML page of collection PZD040 included additional information about the feature type, attributes, and value.


Home / Tunisia (MGCP) / Data / PZD040 - Named Location (Point) / Items
GeoJSON
Filter Edit
« ‹ 1 2 3 4 5 › »
Copyright © by the European Union Satellite Centre (EU SatCen), 2023. All rights reserved. | © OpenStreetMap contributors

PZD040 - Named Location (Point)

A location that normally does not appear as a specific, characterized object but that has a name that is required to be displayed in association with that location. (For example, the name of the Alps or the Sahara.)

Yasmine Hammamet

Horizontal Accuracy Category	Accurate
Commercial Copyright Notice	Copyright © by the European Union Satellite Centre (EU SatCen), 2023. All rights reserved.
Name	Yasmine Hammamet
OBJECTID	1
Source Description	OSM
Source Date and Time	2023-05-16T14:06:43Z
MGCP Feature Universally Uni...	312482164
Annotation type	Annotation type 01
Security Classification	Other
Product Code	Other
Releasability	Other

Product Code according to the SatCen Task Management Tool

Chaker

Horizontal Accuracy Category	Accurate
Commercial Copyright Notice	Copyright © by the European Union Satellite Centre (EU SatCen), 2023. All rights reserved.
Name	Chaker
OBJECTID	2

Figure A.15 — Updated PZD040 features

The ldproxy Schema building block was also enabled in the configuration and enables clients to determine the schema of the features in the dataset. The following is the schema for the PZD040 features.

NOTE: Since not all feature types and attributes in the dataset were included in the data dictionary, some feature types and attributes are lacking schema information such as title, description, and constraints.

```
{
  "title" : "PZD040 - Named Location (Point)",
  "description" : "A location that normally does not appear as a specific,
characterized object but that has a name that is required to be displayed
in association with that location. (For example, the name of the Alps or the
Sahara.)",
  "properties" : {
    "ACC" : {
      "title" : "Horizontal Accuracy Category",
      "description" : "A general evaluation of the horizontal accuracy of the
geographic position of a feature, as a category.",
      "enum" : [ 1, 2 ],
      "type" : "integer"
    },
    "CCN" : {
      "title" : "Commercial Copyright Notice",
      "description" : "A description of any commercial (or similar) copyright
notice applicable to information regarding the feature or data set. ",
      "type" : "string"
    },
    "NAM" : {
      "title" : "Name",
      "description" : "A textual identifier or code that is used to denote a
feature.",
      "type" : "string"
    },
    "NFI" : {
      "title" : "Named Feature Identifier",
      "description" : "The unique named feature identifier element in the NGA
Geographic Names Data Base (GNDB). (Typically used together with Attribute:
'Name Identifier' to provide a unique index into the NGA Geographic Names Data
Base (GNDB) from which NGA draws all of its feature name information.)",
      "type" : "string"
    },
    "NFN" : {
      "title" : "Name Identifier",
      "description" : "The unique name identifier element in the NGA
Geographic Names Data Base (GNDB). (Typically used together with Attribute:
'Named Feature Identifier' to provide a unique index into the NGA Geographic
Names Data Base (GNDB) from which NGA draws all of its feature name
information.)",
      "type" : "string"
    },
    "OBJECTID" : {
      "x-ogc-role" : "id",
      "type" : "integer"
    },
    "SDP" : {
      "title" : "Source Description",
      "description" : "A description of the data set that was used to define
the digital representation of the feature or data set. (No restriction is
placed on the length of the description.)",
      "type" : "string"
    },
    "SDV" : {
      "title" : "Source Date and Time",
      "description" : "The date and, optionally, time of collection of the
data set that was used to define the digital representation of the feature
or data set. (Midnight is understood to be 00:00:00 (the beginning of a
day); when the time is not specified then midnight in the local time zone is
typically implied.)",
      "x-ogc-role" : "primary-instant",
      "format" : "date-time",
      "type" : "string"
    },
    "SRT" : {
      "title" : "Source Type",
      "description" : "The type(s) of the data set(s) that were used to define
the digital representation of the feature or data set. (For example, based on
a data product specification.)",
      "enum" : [ 0, 1, 10, 11, 110, 111, 112, 113, 114, 115, 116, 117, 118,
119, 120, 121, 16, 17, 18, 19, 2, 20, 21, 22, 24, 25, 26, 27, 29, 3, 30, 31,
32, 33, 34, 36, 37, 38, 39, 4, 40, 41, 42, 43, 44, 45, 46, 47, 48, 5, 50, 51,
 52, 53, 54, 55, 56, 57, 59, 6, 60, 61, 62, 63, 64, 65, 7, 8, 85, 92, 93, 94,
95, 996, 997, 998, 999 ],
      "type" : "integer"
    },
    "Shape" : {
      "x-ogc-role" : "primary-geometry",
      "format" : "geometry-point"
    },
    "TXT" : {
      "title" : "IA Comments",
      "description" : "A narrative or other textual description associated
with a feature or data set.",
      "type" : "string"
    },
    "UID" : {
      "title" : "MGCP Feature Universally Unique Identifier",
      "description" : "A unique identifier for each instance of MGCP Feature
assigned by national system in accordance with ISO /IEC 9834-8  standard.
The UUID shall be represented by a string of hexadecimal digits, using two
hexadecimal digits for each octet of the binary form.",
      "type" : "string"
    },
    "XAN" : {
      "title" : "Annotation type",
      "description" : "Different types of text, in essence, labels, on a map
hat display useful information.",
      "enum" : [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ],
      "type" : "integer"
    },
    "XAS" : {
      "type" : "string"
    },
    "XCL" : {
      "title" : "Security Classification",
      "description" : "Security level assigned to document, file, or record
based on the sensitivity or secrecy of the information.",
      "enum" : [ 2, 3, 4, 5, 6, 0, 999 ],
      "type" : "integer"
    },
    "XFA" : {
      "type" : "string"
    },
    "XPC" : {
      "title" : "Product Code",
      "description" : "Product Code according to the SatCen Task Management
Tool",
      "type" : "string"
    },
    "XRE" : {
      "title" : "Releasability",
      "description" : "Capable of being deliverable to other institutions.
Example: \"Releasable to….\"",
      "type" : "string"
    },
    "XSD" : {
      "type" : "string"
    },
    "XSI" : {
      "title" : "Imagery Source Universally Unique Identifier",
      "description" : "A unique identifier for the associated imagery
source used for the features. The USID shall be represented by a string of
hexadecimal digits, using two hexadecimal digits for each octet of the binary
form.",
      "type" : "string"
    },
    "XTA" : {
      "type" : "string"
    }
```

```
  },
  "type" : "object",
  "required" : [ "ACC", "CCN", "NAM", "SDP", "SDV", "SRT", "XAN", "XCL", "XPC",
 "XRE" ],
  "$schema" : "https://json-schema.org/draft/2020-12/schema",
  "$id" : "https://t19.ldproxy.net/tunisia/collections/PZD040/schema"
}
```

Figure A.16 — Schema of the PZD040 features

A.5. Component D125: An instance of OGC API — Features supporting real-time observations

A.5.1. Overview

The requirement for this deliverable was:

> an instance of OGC API — Features (and/or STA and/or EDR) — supporting real-time observations of some phenomena relevant to the Use Case, supports sufficient semantic annotation to identify necessary contextual information to support reuse.

interactive instruments provided a Features API endpoint that publishes surface-based weather observations that are harvested from the WMO Information System (WIS) 2.0. In addition to accessing the data using the OGC API — Features building blocks, the API also supported the draft OGC API — Environmental Data Retrieval — Part 2: Publish-Subscribe Workflow building blocks for feature resources.

The API uses ldproxy, an OGC Reference Implementation for OGC API — Features. As part of Testbed 19, interactive instruments extended the ldproxy to support PubSub building blocks.

A.5.2. Components

A.5.2.1. Overview

In order to publish observation data via the PubSub extension for OGC API — Features, a system consisting of several components was developed and deployed. Figure A.17 provides an overview of the relevant components and their interactions.

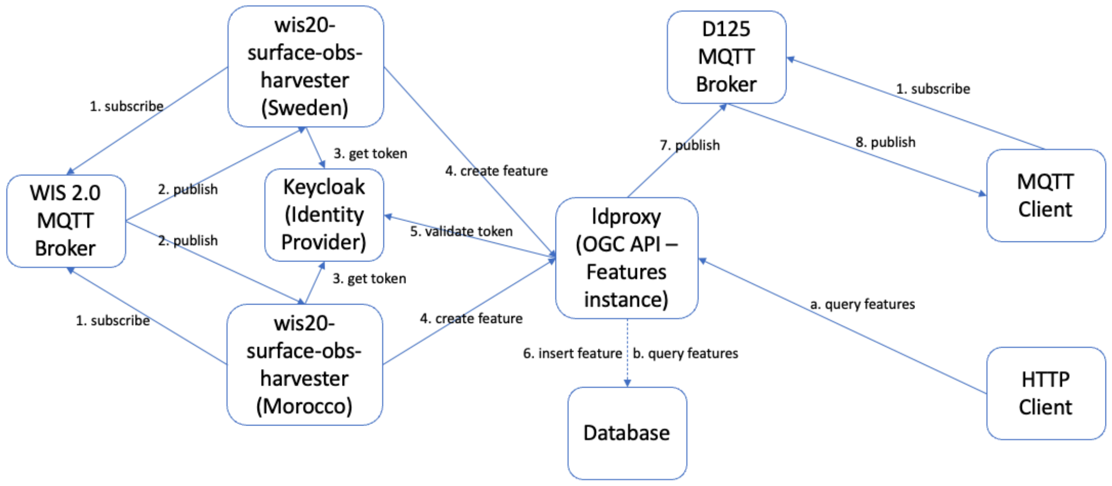


Figure A.17 — Components in the system (from interactive instruments) that provides observation data via the PubSub extension for OGC API - Features

The *WIS 2.0 MQTT broker* is an external component. It publishes weather observations from countries across the globe. MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT).

The *WIS 2.0 surface observation harvester*, implemented by interactive instruments in Testbed 19, subscribes to surface weather observations published by the WIS 2.0 MQTT broker, processes them, and posts the results to the *ldproxy* instance. The implementation of the draft using OGC API — Features — Part 4: Create, Replace, Update and Delete is used to create new observation features that are stored in a *PostgreSQL/PostGIS database*. Two harvesters had been deployed for Testbed-19: one to gather surface weather observations from Sweden, and the other to do so for Morocco.

Write access to the ldproxy instance was restricted to authenticated clients. In the testbed, interactive instruments used *Keycloak* as an identity provider and the OpenID Connect Client Credentials Flow to ensure that only harvester instances could create new observations.

NOTE: At the time of writing, Keycloak is not yet deployed. This will be done in the upcoming weeks.

Once a new observation feature was created, messages were published via a *Mosquitto MQTT Broker*. Such messages could be received using any MQTT Client.

The harvesters, ldproxy, Keycloak, the database, and the MQTT broker were deployed in a cloud environment, using docker containers — see Figure A.18. The deployment was managed using *Portainer*.

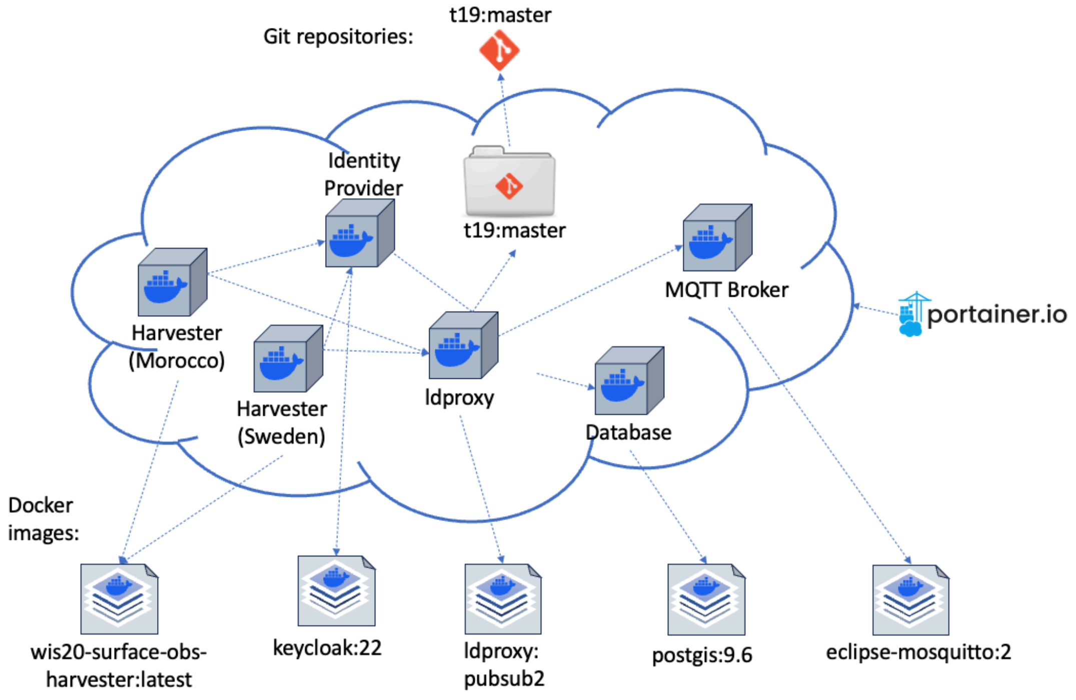


Figure A.18 — Deployment of the components

A.5.2.2. WIS 2.0 MQTT broker

The WIS 2.0 MQTT broker is an external component, operated by Meteo France, that publishes WIS 2.0 weather observation data from multiple countries. In Testbed 19, the broker from Meteo France was used.

Each WIS 2.0 message is encoded as a GeoJSON feature. The actual weather data is provided in WMO's BUFR format. The data typically needs to be downloaded using a link contained in the GeoJSON feature. Some data providers also encode the data directly using a base64 encoding in the GeoJSON feature. See *WIS 2.0 Notification Message Format* for further details about the WIS 2.0 messages.

The WIS 2.0 MQTT broker publishes weather observations on a range of different MQTT topics. For example, surface-based weather observations are published on topic(s): `origin/a/wis2/+/+/data/core/weather/surface-based-observations/synop`. The +-sign thereby serves as a wildcard. The first is for the country, the second for the center-id. See *WIS 2.0 topic hierarchy* for further details about the WIS 2.0 topics.

A.5.2.3. WIS 2.0 surface observation harvester

A.5.2.3.1. Overview

The harvester component is a Java application that subscribes for surface-based weather observations from a particular country at the Annex A.5.2.2. In Testbed 19, data from Sweden and Morocco was harvested.

NOTE: Since the WIS 2.0 weather data was used in Testbed 19 as a source for event data, to demonstrate the PubSub extension for OGC API — Features, the subscription was created with MQTT Quality of Service (QoS) level 0, meaning that not all messages published by the broker may be received and processed by the harvester. To avoid message loss, a higher QoS level would need to be chosen, which is entirely possible. However, it would increase the resource consumption for both the broker and the harvester, which is why QoS level 0 was used for the Testbed 19 demonstrator.

A.5.2.3.2. Workflow

Receipt of a new observation triggers the following workflow.

1. The observation (in WIS 2.0 GeoJSON format) is parsed.
2. If the weather data is not encoded in BUFR format (Binary Universal Form for the Representation of meteorological data), the observation is dismissed. Otherwise, the BUFR data is decoded or downloaded and temporarily stored locally.
3. The BUFR file is converted to a number of GeoJSON files using the bufr2geojson package.
4. The resulting GeoJSON files are parsed. If a GeoJSON feature does not have a 2D point geometry, or the data is given with unit "associated units," the feature is dropped. Otherwise, the feature is posted to the ldproxy instance. More specifically, the feature is posted to a particular collection at the OGC API — Features implementation. A new feature is created in the collection, as specified by the draft OGC API — Features — Part 4: Create, Replace, Update and Delete — which is supported by ldproxy.
5. Finally, the temporary BUFR file, as well as the GeoJSON files created by the bufr2geojson tool, are deleted.

A.5.2.3.3. Build and deployment

The harvester Java application is packaged as a Docker image. The image is based on the bufr2geojson:latest image (which itself is based on the wmoim/dim_eccodes_baseimage:2.28.0 image). All programs required by the application are thus directly available. When the Docker image is deployed and run as a Docker container, the topic to subscribe to at the Annex A.5.2.2 can be configured, to be used as a command parameter that is added to the invocation of the harvester application.

A.5.2.4. ldproxy

A.5.2.4.1. Overview

The Features API implementation was configured to implement selected conformance classes from the first four parts of OGC API — Features.

- OGC API — Features — Part 1: Core (supported conformance classes: Core, GeoJSON, HTML and OpenAPI 3.0)
- OGC API — Features — Part 2: Coordinate Reference Systems by Reference (supported conformance classes: CRS)
- OGC API — Features — Part 3: Filtering *Draft* (supported conformance classes: Queryables, Queryables as Query Parameters, Filter, Features Filter)
- OGC API — Features — Part 4: Create, Replace, Update, and Delete *Draft* (supported conformance classes: create-replace-delete, features)
- Common Query Language 2 *Draft* (supported conformance classes: Basic CQL2, Advanced comparison operators, Case-insensitive comparison, Basic spatial operators, Spatial operators, Temporal operators, Array operators, Property-property comparison, CQL2-JSON, CQL2-Text)

A.5.2.4.2. Database

The observations that are created using the Create Feature operations are stored in a PostgreSQL/PostGIS database. They can be retrieved by clients using implementations of the OGC API — Features building blocks that provide access to features.

The schema of the observation features is derived from the schema of the features created by bufr2geojson.

```
CREATE TABLE wisobservation (

   _id bigserial NOT NULL PRIMARY KEY,
   position geometry(POINT,4326) NOT NULL,
   dataid text NOT NULL,
   reportid text,
   wigos_station_identifier text,
   phenomenontime timestamp with time zone NOT NULL,
   resulttime timestamp with time zone,
   name text,
   value numeric NOT NULL,
   units text NOT NULL,
   description text,
   index integer,
   fxxyyy text
);
```

Figure A.19 — SQL DDL of the WIS 2.0 surface observation features

A.5.2.4.3. Publication of feature events

In addition to providing access to the observation features using Web API based on the HTTP protocol, the API also supported asynchronous, event-driven access using the proposed OGC API — EDR Part 2: PubSub building blocks for feature resources.

With the definition of an OGC API -EDR: Part 2 PubSub extension, clients will be able to subscribe to events related to the observation data. In Testbed 19, the events (new observations) were published by the Features API via an MQTT broker in an efficient, low-latency manner, with configurable Quality of Service. Without the PubSub extension, clients would have had to regularly poll the Features API for new observation data. With the PubSub extension, new observation data was automatically pushed to subscribed clients as soon as such data becomes available.

The work on a PubSub extension for the OGC API Standards suite began at the time of the Testbed 19 initiative. The implementation used in this Testbed supported two different ways to subscribe to observations as follows.

- A single topic to publish all new observations as a GeoJSON feature
- MQTT topic: `ogcapi/t19.ldproxy.net/wis20/collections/wisobservation/items`
- One topic for each combination of the weather station and the observed property with just the value of the observation
- MQTT topic: `ogcapi/t19.ldproxy.net/wis20/collections/wisobservation/{wigos_station_identifier}/{observed_property}`

AsyncAPI was used to specify the PubSub capabilities so that clients were able to subscribe to new observations. AsyncAPI is a suite of Open-Source tools to easily build and maintain event-driven architectures.

```
{
  "info": {
    "title": "WIS 2.0 Surface-based Weather Observations",
    "description": "Surface-based weather observations for Sweden and Morocco.
The weather observations are published through the Weather Information System
2.0 (WIS 2.0) of the UN World Meteorological Organization (WMO). The data
is harvested from one of the WIS 2.0 Global Brokers and published via an API
implementing the OGC API - Features Standard. This API is developed as part
of <a href=\"https://www.ogc.org/projects/initiatives/t19\" target=\"_blank\">
OGC Testbed-19</a>. The API is experimental, a work-in-progress and subject to
change.",
    "version": "1.0.0",
    "contact": {
      "name": "interactive instruments GmbH",
      "email": "mail@interactive-instruments.de"
    },
    "license": {
      "name": "Unspecified"
    }
  },
  "servers": {
    "t19": {
      "protocol": "secure-mqtt",
      "protocolVersion": "3.1.1",
      "url": "t19.ldproxy.net:8883",
      "bindings": {
        "clientId": "t19.ldproxy.net",
        "cleanSession": true,
        "bindingVersion": "0.1.0"
      }
    }
  },
  "channels": {
    "ogcapi/t19.ldproxy.net/wis20/collections/wisobservation/items": {
      "subscribe": {
        "operationId": "featureChange_wisobservation_items",
        "bindings": {
          "qos": 0,
          "retain": false,
          "bindingVersion": "0.1.0"
        },
        "message": {
          "$ref": "#/components/messages/featureChange_wisobservation"
        }
      },
      "servers": [
        "t19"
      ]
    },
    "ogcapi/t19.ldproxy.net/wis20/collections/wisobservation/{wigos_station_
identifier}/{observed_property}": {
      "subscribe": {
        "operationId": "featureChange_wisobservation_wigos_station_identifier_
observed_property",
        "bindings": {
          "qos": 0,
          "retain": true,
          "bindingVersion": "0.1.0"
        },
        "parameters": {
          "observed_property": {
            "type": "string"
          },
          "wigos_station_identifier": {
            "type": "string"
          }
        },
        "message": {
          "$ref": "#/components/messages/valueChange_wisobservation_value"
        }
      },
      "servers": [
        "t19"
      ]
    }
  },
  "components": {
    "messages": {
      "featureChange_wisobservation": {
        "name": "featureChangeMessage",
        "title": "Feature Change",
        "summary": "Information about a new, updated or deleted feature.",
        "contentType": "application/geo+json",
        "payload": {
          "type": "object"
        }
      },
      "valueChange_wisobservation_value": {
        "name": "valueChangeMessage",
        "title": "Value Change",
        "summary": "Information about a new or updated feature property.",
        "contentType": "plain/text",
        "payload": {
          "type": "number"
        }
      }
    }
  },
  "asyncapi": "2.6.0"
}
```

Figure A.20 — AsyncAPI definition

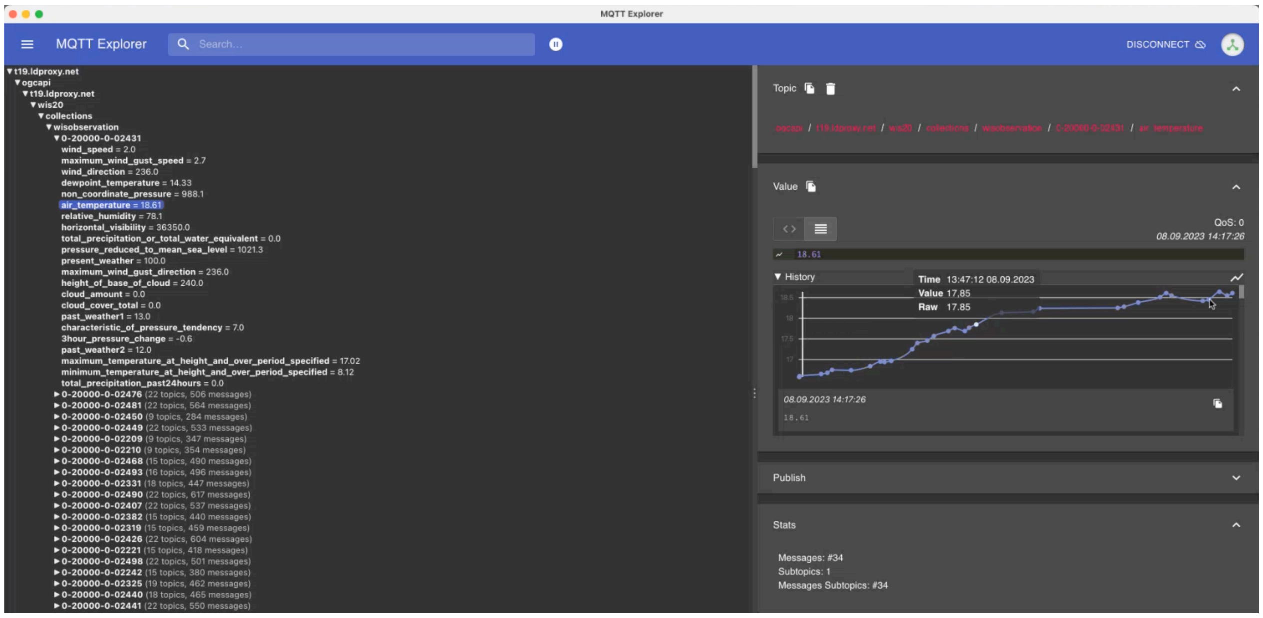
Once a new observation feature was created, two messages were published via the D125 MQTT Broker:

- one message with the new observation feature in GeoJSON; and
- one message with just the numeric value in a topic for the weather station and observed property.

The GeoJSON message consisted of the observation feature with additional properties as specified by the draft OGC API Pub Sub extension. The implementation deviated from the draft OGC API Pub Sub extension in the following ways.

- `$id`: a UUID for the message (the draft OGC API Pub Sub extension stored this information in the top-level “id” GeoJSON property, which would have conflicted with the existing feature identifier).
- `$pubTime`: the timestamp of the publication (to avoid name clashes, a “$” prefix has been added the property name specified by the draft OGC API Pub Sub extension).
- `$operation`: always `"create"`, since observations are only created, never changed or deleted (to avoid name clashes, a “$” prefix has been added the property name specified by the draft OGC API Pub Sub extension).

Such messages could be received using MQTT Clients, for example, MQTT Explorer — see Figure A.21.



```
▼t19.ldproxy.net
 ▼ogcapi
  ▼t19.ldproxy.net
   ▼wis20
    ▼collections
     ▼wisobservation
      ▼0-20000-0-02431
        wind_speed = 2.0
        maximum_wind_gust_speed = 2.7
        wind_direction = 236.0
        dewpoint_temperature = 14.33
        non_coordinate_pressure = 988.1
        air_temperature = 18.61
        relative_humidity = 78.1
        horizontal_visibility = 36350.0
        total_precipitation_or_total_water_equivalent = 0.0
        pressure_reduced_to_mean_sea_level = 1021.3
        present_weather = 100.0
        maximum_wind_gust_direction = 236.0
        height_of_base_of_cloud = 240.0
        cloud_amount = 0.0
        cloud_cover_total = 0.0
        past_weather1 = 13.0
        characteristic_of_pressure_tendency = 7.0
        3hour_pressure_change = -0.6
        past_weather2 = 12.0
        maximum_temperature_at_height_and_over_period_specified = 17.02
        minimum_temperature_at_height_and_over_period_specified = 8.12
        total_precipitation_past24hours = 0.0
      ▶0-20000-0-02476 (22 topics, 506 messages)
      ▶0-20000-0-02481 (22 topics, 564 messages)
      ▶0-20000-0-02450 (9 topics, 284 messages)
      ▶0-20000-0-02449 (22 topics, 533 messages)
      ▶0-20000-0-02209 (9 topics, 347 messages)
      ▶0-20000-0-02210 (9 topics, 354 messages)
      ▶0-20000-0-02468 (15 topics, 490 messages)
      ▶0-20000-0-02493 (16 topics, 496 messages)
      ▶0-20000-0-02331 (18 topics, 447 messages)
      ▶0-20000-0-02490 (22 topics, 617 messages)
      ▶0-20000-0-02407 (22 topics, 537 messages)
      ▶0-20000-0-02382 (15 topics, 440 messages)
      ▶0-20000-0-02319 (15 topics, 459 messages)
      ▶0-20000-0-02426 (22 topics, 604 messages)
      ▶0-20000-0-02221 (15 topics, 418 messages)
      ▶0-20000-0-02498 (22 topics, 501 messages)
      ▶0-20000-0-02242 (15 topics, 380 messages)
      ▶0-20000-0-02325 (19 topics, 462 messages)
      ▶0-20000-0-02440 (18 topics, 465 messages)
      ▶0-20000-0-02441 (22 topics, 550 messages)
```

Topic

ogcapi / t19.ldproxy.net / wis20 / collections / wisobservation / 0-20000-0-02431 / air_temperature

Value

QoS: 0
08.09.2023 14:17:26

18.61

▼ History

Time 13:47:12 08.09.2023
Value 17,85
Raw 17.85

08.09.2023 14:17:26
18.61

Publish

Stats

Messages: #34
Subtopics: 1
Messages Subtopics: #34

Figure A.21 — Air temperature observations at a Swedish weather station in MQTT Explorer

A.5.2.4.4. Configuration

```
id: wis20
entityStorageVersion: 2
providerType: FEATURE
providerSubType: SQL
nativeCrs:
  code: 4326
  forceAxisOrder: LON_LAT
nativeTimeZone: Europe/Berlin
connectionInfo:
  connectorType: SLICK
  host: db
  database: wis20
  user: # not shown
  password: # not shown
  dialect: PGIS
  pool:
    initFailTimeout: 10s
queryGeneration:
  computeNumberMatched: false
sourcePathDefaults:
  primaryKey: _id
  sortKey: _id
types:
  wisobservation:
    sourcePath: /wisobservation
    type: OBJECT
    objectType: WisObservation
    label: WIS 2.0 Surface-based Weather Observations
    properties:
      oid:
        sourcePath: _id
```

```
          type: INTEGER
          role: ID
        position:
          sourcePath: position
          type: GEOMETRY
          role: PRIMARY_GEOMETRY
          geometryType: POINT
          constraints:
            required: true
        dataId:
          sourcePath: dataid
          type: STRING
          constraints:
            required: true
        reportId:
          sourcePath: reportid
          type: STRING
        wigos_station_identifier:
          sourcePath: wigos_station_identifier
          type: FEATURE_REF
          valueType: STRING
          refUriTemplate: 'https://oscar.wmo.int/surface/#/search/station/
stationReportDetails/{{value}}'
        phenomenonTime:
          sourcePath: phenomenontime
          type: DATETIME
          role: PRIMARY_INSTANT
          constraints:
            required: true
        resultTime:
          sourcePath: resulttime
          type: DATETIME
        name:
          sourcePath: name
          type: STRING
        value:
          sourcePath: value
          type: FLOAT
          constraints:
            required: true
        units:
          sourcePath: units
          type: STRING
          constraints:
            required: true
        description:
          sourcePath: description
          type: STRING
        index:
          sourcePath: index
          type: INTEGER
        fxxyyy:
          sourcePath: fxxyyy
          type: STRING
```

Figure A.22 — Feature provider (`store/entities/providers/wis20.yml`)

```
id: wis20
entityStorageVersion: 2
label: WIS 2.0 Surface-based Weather Observations
```

```
description: 'Surface-based weather observations for Sweden and Morocco. The
 weather observations are published through the Weather Information System
2.0 (WIS 2.0) of the UN World Meteorological Organization (WMO). The data is
 harvested from one of the WIS 2.0 Global Brokers and published via an API
implementing the OGC API - Features Standard. This API is developed as part
of <a href="https://www.ogc.org/projects/initiatives/t19" target="_blank">OGC
 Testbed-19</a>. The API is experimental, a work-in-progress and subject to
change.'
enabled: true
serviceType: OGC_API
metadata:
  keywords:
  - Testbed-19
  - D125
  - weather
  - observation
  contactName: interactive instruments GmbH
  contactEmail: mail@interactive-instruments.de
  creatorName: WMO Members
  creatorUrl: https://public.wmo.int/
  publisherName: interactive instruments GmbH
  publisherUrl: https://www.interactive-instruments.de/
  licenseName: Unspecified # WMO is unclear about the license of the data
  attribution: WMO Members, interactive instruments GmbH
accessControl:
  enabled: false
api:
- buildingBlock: QUERYABLES
  enabled: true
- buildingBlock: SORTING
  enabled: true
- buildingBlock: FILTER
  enabled: true
- buildingBlock: CRUD
  enabled: true
- buildingBlock: PUB_SUB
  enabled: true
  brokers:
    t19:
      host: t19.ldproxy.net
      port: 8883
  publisher: t19.ldproxy.net
  publications:
    items:
      broker: t19
      mqttQos: AT_MOST_ONCE
    '{wigos_station_identifier}/{observed_property}':
      parameters:
        wigos_station_identifier: wigos_station_identifier
        observed_property: name
      property: value
      broker: t19
      mqttQos: AT_MOST_ONCE
      retain: true
collections:
  wisobservation:
    id: wisobservation
    label: wisobservation
    enabled: true
    api:
    - buildingBlock: QUERYABLES
      included:
      - dataId
      - name
      - reportId
      - value
      - units
      - wigos_station_identifier
    - buildingBlock: SORTING
      included:
      - dataId
      - name
      - reportId
      - value
      - units
      - wigos_station_identifier
    - buildingBlock: FEATURES_HTML
      transformations:
        phenomenonTime:
        - dateFormat: dd.MM.yyyy HH:mm:ss
        resultTime:
        - dateFormat: dd.MM.yyyy HH:mm:ss
```

Figure A.23 — API building blocks (`store/entities/services/wis20.yml`)

A.5.2.5. D125 MQTT broker

The D125 MQTT broker is a standard deployment of Mosquitto, a popular MQTT broker software.

A.6. D127: An instance of OGC API – Tiles serving OS Open Zoomstack data

A.6.1. Overview

The requirement for this deliverable was:

> An instance of OGC API Tiles serving OS Open Zoomstack data.

OS Open Zoomstack is a comprehensive vector basemap showing coverage of Great Britain at a national level, right down to street-level detail. interactive instruments provided the API using ldproxy, an implementation of OGC API Tiles.

A.6.2. OS Open Zoomstack API

A.6.2.1. Overview

Link to the API Landing Page

This is an existing API endpoint that interactive instruments maintains as one of the ldproxy demonstration APIs. The OS Open Zoomstack dataset is provided by Ordnance Survey as GeoPackage (features) and MBTiles (vector tiles).

This API endpoint also provides resources from OGC API — Features and the draft OGC API — Styles Standards.

A.6.2.2. Dataset

The dataset is available from the Ordnance Survey website.

In addition, stylesheets and associated resources (i.e., fonts and symbols) are available on GitHub.

A.6.2.3. ldproxy Configuration

To deploy one or more APIs with ldproxy, configuration files for the deployment are needed. The configuration is typically maintained in a git repository.

For the demo.ldproxy.net deployment, which includes the OS Open Zoomstack API, the configuration is available at https://github.com/ldproxy/demo.

Since this is a demonstration deployment, the repository also contains additional documentation about the different APIs.

A.6.2.4. Deployment

ldproxy is only distributed as a Docker image. The configuration repository also includes a Docker Compose file to simplify the process of starting a local deployment.

The image below is a screenshot of a web map using the OS Open Zoomstack vector tiles and the “Road” style as served by the API.

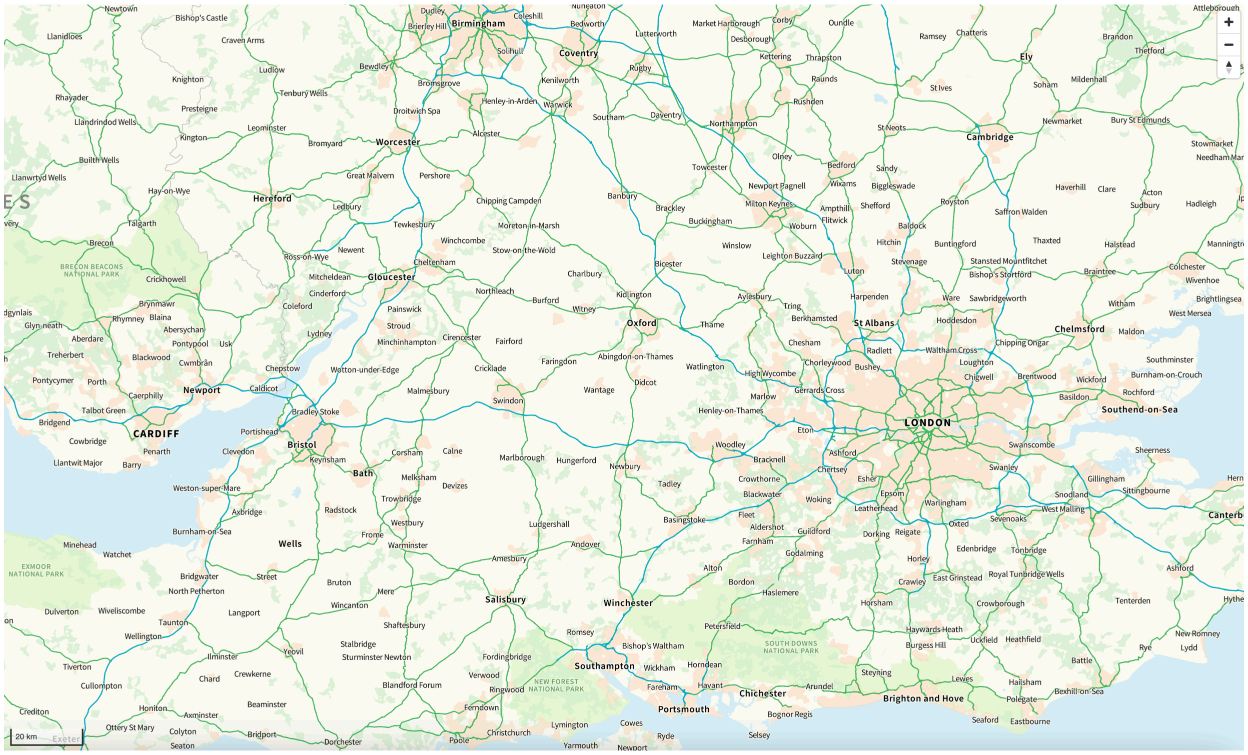


Figure A.24 — MapLibre web map of the OS Open Zoomstack vector tiles with the "Road" style. Contains OS data © Crown copyright and database right 2023.

A.7. Component D128: An instance of OGC API – Records

A.7.1. Purpose

This component provides metadata about all of the software and data components available in the (prototype) system.

RM-ODP defines five viewpoints (on a system) that yield a specification of the whole system related to a particular set of concerns. The five viewpoints defined by RM-ODP have been chosen to be both simple and complete, covering all the domains of architectural design.

In the Agile Reference Architecture (ARA) approach, the assumption is that a system is built from reusable ARA Blocks, or components. Depending on the RM-ODP viewpoint chosen, they are described in the catalog from a different perspective.

- Information Viewpoint: ARA Blocks are described as Datasets or Dataset series metadata records.

- Computational Viewpoint: ARA Blocks are described as Service metadata records offering specific interfaces.

The two viewpoints describing an ARA Block are not independent. Information viewpoint and computational viewpoint metadata related to the same ARA Block can be modeled as “coupled resources” and/or related through one of more “offerings” as defined in the OWS Context Conceptual Model OGC 12-080r2 and corresponding encodings, e.g., OGC 14-055r2.

Table A.2 — Resources — ARA Blocks

TYPE	IDENTIFIER	DESCRIPTION	OFFERINGS	COUPLED RESOURCES
Dataset	OSM	OSM Dataset, Data Container	OGC API-Features, OGC API-Maps, Docker data container	-
Dataset	RTD	Real-time Dataset	OGC API-Features	-
Dataset	DD	SatCen Data Dictionary	OGC API-Features, Docker data container	-
Dataset	ARA	TB19 ARA Blocks	OGC API-Records	-
Service	D123	OGC API-Processes instance	OGC API-Processes	
Service	D124	OGC API-Features instance serving OSM data	OGC API-Features	OSM
Service	D125	OGC API-Features instance serving real-time data	OGC API-Features	RTD
Service	D128	OGC API-Records instance serving ARA Blocks metadata	OGC API-Records	ARA

A.7.2. Embedded metadata

While ARA Blocks are described with metadata records in the API-Records catalog server, it is proposed that the containerized ARA blocks contain embedded metadata as well.

Metadata can be included in containers and `pods` as described in D. Meyer et al., [15] and [16]. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. The embedded metadata is still to be defined (e.g., in YAML format which is equivalent to JSON), but expected to cover additional RM-ODP viewpoint information, in particular platform and distribution-related information.

The embedded metadata (YAML file) and catalog metadata records are proposed to contain “trust” and “provenance” information.

A.7.3. Metadata response formats

Some metadata formats are more suitable than others for describing coupled-resources, offerings, and provenance information.

A.7.3.1. Coupled resources

In the ISO 19139 Geographic information XML schema implementation Part 1: Encoding rules Standard and other metadata formats, service metadata can refer to the target datasets of the described service by reference, i.e., through a URL that points to the metadata record of the data on which the service operates.

```
<gmd:identificationInfo>
   <srv:SV_ServiceIdentification>
      <srv:operatesOn xlink:href="https://emc.spacebel.be/collections/tb19-osm?
httpAccept=application%2Fvnd.iso.19139%2Bxml#tb19-osm"/>
   </srv:SV_ServiceIdentification>
</gmd:identificationInfo>
```

Figure A.25 — ISO19139 coupled resource

The `dcat:servesDataset` and `dcat:service` allow a similar coupling in GeoDCAT-AP metadata.

```
{
  "@type": "dcat:DataService",
  "dct:type": "http://inspire.ec.europa.eu/metadata-codelist/ResourceType/
service",
  "dct:identifier": "tb19-d124",
  "dcat:servesDataset": [
        {
           "@type": "dcat:Dataset",
           "dct:identifier": "tb19-osm",
           "@id": "https://emc.spacebel.be/collections/series/items/tb19-osm"
        }
  ],
}
```

Figure A.26 — GeoDCAT-AP coupled resource

A.7.3.2. Offerings

The offering concept was defined in the OGC OWS Context Conceptual Model Standard [OGC 12-080r2] and OGC OWS Context GeoJSON Encoding Standard [OGC 14-055r2]. In the OGC Testbed-15: Catalog and Discovery Engineering Report OGC 19-020r1 the set of available offering identifiers was proposed to be extended with OGC “offerings” for OGC API interfaces and (Docker) containers.

```
{
        "type": "Offering",
        "code": "http://www.opengis.net/spec/eopad-geojson/1.0/req/docker/
image",
        "contents": [
                {
                        "type": "text/plain",
                        "content": "docker.tb19.org/data/osm:latest"
                }
        ]
}
```

Figure A.27 — Docker container offering

The same specification provides an example of an offering for an implementation of OGC API — Processes.

```
{
        "type": "Offering",
        "code": "http://www.opengis.net/spec/eopad-geojson/1.0/req/ogc-api-
processes",
        "operations": [
                {
                        "code": "LandingPage",
                        "method": "GET",
                        "href": "http://172.17.20.10:30080/ogcapi/",
                        "type": "application/json"
                },
                {
                        "code": "Service",
                        "method": "GET",
                        "href": "http://172.17.20.10:30080/ogcapi/api/",
                        "type": "application/openapi+json;version=3.0"
                },
                {
                        "code": "Conformance",
                        "method": "GET",
                        "href": "http://172.17.20.10:30080/ogcapi/conformance/
",
                        "type": "application/json"
                },
                {
                        "code": "Processes",
                        "method": "GET",
                        "href": "http://172.17.20.10:30080/ogcapi/processes/",
                        "type": "application/json"
                },
                {
                        "code": "DescribeProcess",
                        "method": "GET",
                        "href": "http://172.17.20.10:30080/ogcapi/processes/
reproject",
                        "type": "application/json",
                        "result": {
                                "type": "application/json",
                                "content": {
                                        "process": {
                                                "id": "reproject",
                                                "title": "...",
                                                ...
                                        }
                                }
                        }
                }
        ]
}
```

Figure A.28 — OGC API-Processes offering (GeoJSON)

In GeoDCAT-AP (JSON-LD) the same offerings are encoded as `dcat:endpointDescription` as defined in the EO Collection GeoJSON(-LD) Encoding Standard [OGC17-084r1].

```
"dcat:endpointDescription": [{
    "@type": "owc:Offering",
    "owc:code": {"@id": "http://www.opengis.net/spec/eopad-geojson/1.0/req/ogc-
api-features"},
    "owc:operations": [
      {
        "owc:href": "https://t19.ldproxy.net/wis20?f=json",
        "@type": "owc:Operation",
        "owc:type": "application/json",
        "owc:code": "LandingPage",
        "owc:method": "GET"
      },
      {
        "owc:href": "https://t19.ldproxy.net/wis20/api?f=json",
        "@type": "owc:Operation",
        "owc:type": "application/openapi+json;version=3.0",
        "owc:code": "Service",
        "owc:method": "GET"
      },
      {
        "owc:href": "https://t19.ldproxy.net/wis20/conformance",
        "@type": "owc:Operation",
        "owc:type": "application/json",
        "owc:code": "Conformance",
        "owc:method": "GET"
      }
    ]
  }],
```

Figure A.29 — OGC API-Features offering (GeoDCAT-AP)

A.7.3.3. Provenance

Different levels of provenance information details can be included in ISO19139, DCAT v2, GeoDCAT-AP, W3C PROV or other metadata encodings.

- https://semiceu.github.io/GeoDCAT-AP/drafts/latest/#properties-for-provenance-statement
- https://www.w3.org/TR/vocab-dcat-2/#examples-dataset-provenance
- https://www.w3.org/TR/prov-overview/
- https://www.w3.org/TR/vc-data-model-2.0

```
{
    "@type": "dcat:Dataset",
    "@id": "https://emc.spacebel.be/collections/series/items/tb19-osm",
    "dct:type": "http://inspire.ec.europa.eu/metadata-codelist/ResourceType/
series",
    "dct:identifier": "tb19-osm",
    "dct:provenance": [{
      "rdfs:label": "The dataset was provided by the US National Geospatial
Intelligence Agency (NGA) for development, testing and demonstrations in
initiatives of the Open Geospatial Consortium (OGC). For any reuse of the
data, please contact NGA.",
      "@type": "dct:ProvenanceStatement"
    }]
}
```

Figure A.30 — GeoDCAT-AP Provenance Statement

```
<gmd:dataQualityInfo>
    <gmd:DQ_DataQuality>
      <gmd:scope>
 <gmd:DQ_Scope>
   <gmd:level>
     <gmd:MD_ScopeCode codeList="http://standards.iso.org/iso/19139/resources/
gmxCodelists.xml#MD_ScopeCode" codeListValue="series"/>
   </gmd:level>
 </gmd:DQ_Scope>
      </gmd:scope>
      <gmd:lineage>
        <gmd:LI_Lineage>
          <gmd:statement>
          <gco:CharacterString>
            The dataset was provided by the US National Geospatial
Intelligence Agency (NGA) for development, testing and demonstrations in
initiatives of the Open Geospatial Consortium (OGC). For any reuse of the
data, please contact NGA.
          </gco:CharacterString>
          </gmd:statement>
        </gmd:LI_Lineage>
      </gmd:lineage>
    </gmd:DQ_DataQuality>
  </gmd:dataQualityInfo>
</gmd:dataQualityInfo>
```

Figure A.31 — ISO19139 Provenance Statement

Alternatively, the provenance information can be described using the W3C PROV vocabularies and refer to an external description.

```
{
    "@type": "dcat:Dataset",
    "@id": "https://emc.spacebel.be/collections/series/items/tb19-osm",
    "dct:type": "http://inspire.ec.europa.eu/metadata-codelist/ResourceType/
series",
    "dct:identifier": "tb19-osm",
    "prov:wasGeneratedBy": [
        {
            "@id": "https://server.org/provenance-info.jsonld"
        }
    ]
}
```

Figure A.32 — GeoDCAT-AP External Provenance Information

In a decentralized environment, provenance information may be encoded as verifiable claims. A JSON-LD/RDF encoding is available in the Verifiable Credentials Data Model v2.0 from W3C.

A.7.3.4. Integrity

The integrity and authenticity (trust) might be supported by using the DCS (Data Centric Security) approach proposed in the OGC Testbed-18: Secure Asynchronous Catalog Engineering Report OGC 22-018. For example, JWT tokens or JWS or JWS/CT signatures computed over data or container image bytes and embedded in corresponding (dataset/component) metadata records and/or description files. The JWS/CT (Clear Text) approach allows adding the signature while preserving the readability.

In OGC 22-018, the .well-known/jwks.json URL on a server was used to discover public keys. When decentralized identifiers (DID) are used (see below), the key information can be obtained from the DID document instead.

For JSON-LD (RDF) encodings, the spdx:checksum property may be used to associate a checksum with a digital object. See also https://github.com/ESIPFed/science-on-schema.org/blob/master/guides/Dataset.md#checksum.

A.7.4. Implementation

The OGC API — Records implementation supports obtaining search results and metadata records in various representations using the HTTP query parameter `httpAccept` or using the HTTP header parameter `Accept` (content negotiation). Supported metadata formats include the following.

- OGC 19-020r1, OGC 17-084r1
- ISO19139
- GeoDCAT-AP in JSON-LD, RDF/XML or Turtle encoding.

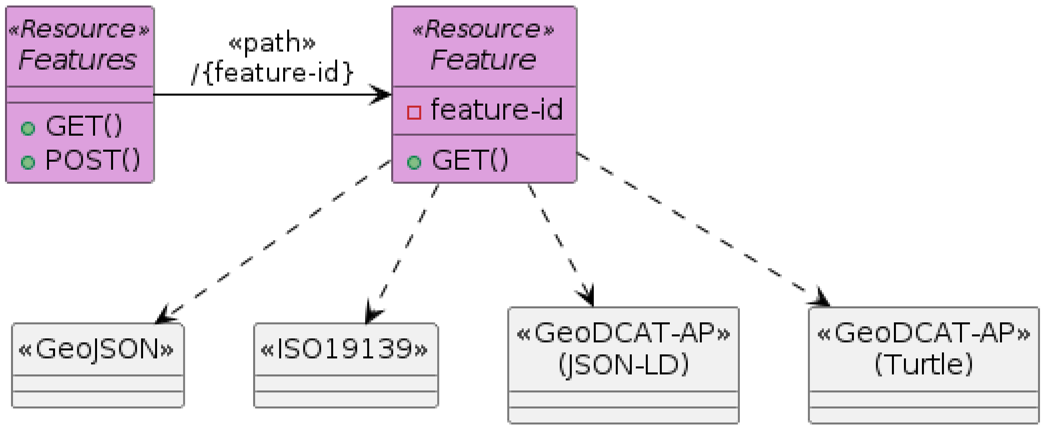


Figure A.33 — Supported metadata formats

The implementation supports CQL2 Text and Basic CQL.

Metadata Record Examples:

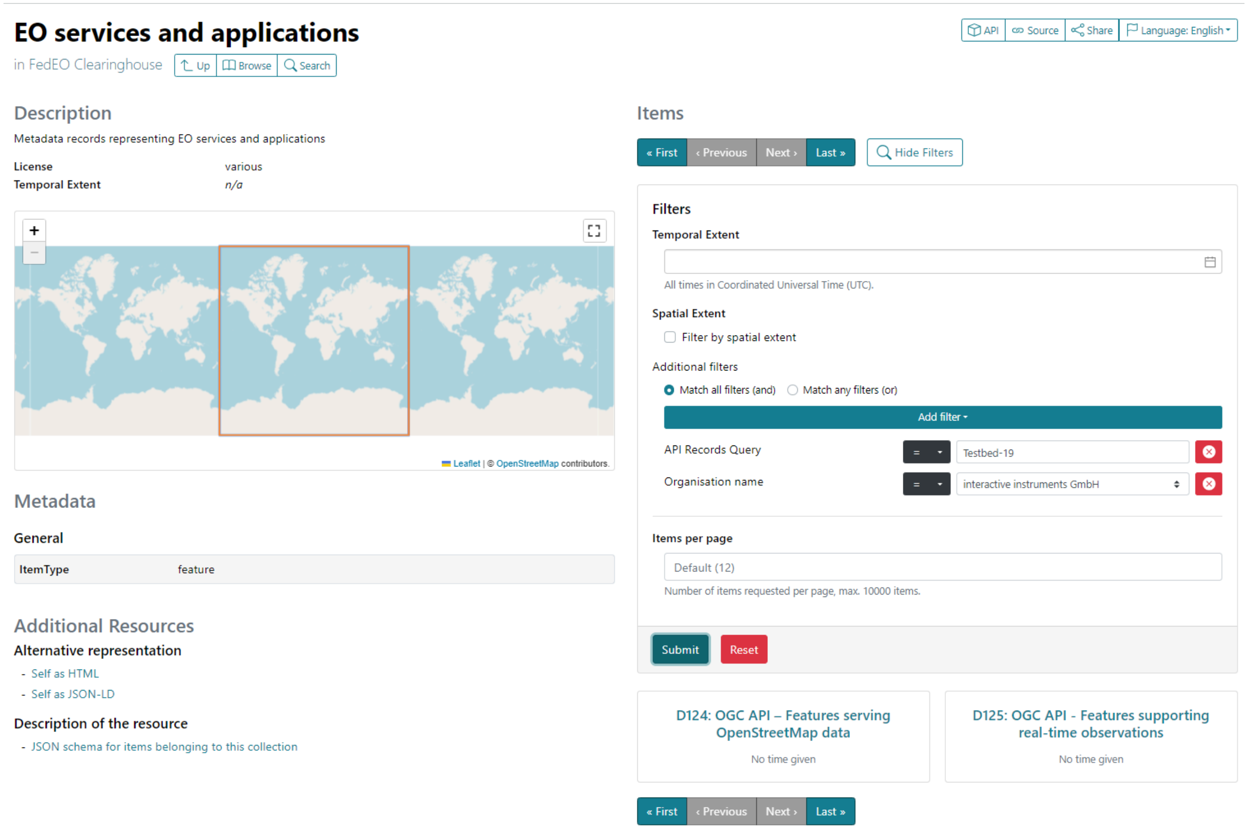
- GeoJSON
  - /collections/services/items/tb19-d123 (API-Processes)
  - /collections/services/items/tb19-d124 (API-Features)
  - /collections/services/items/tb19-d125 (API-Features)
  - /collections/series/items/tb19-osm (Dataset Series)
  - /collections/series/items/tb19-rtd (Dataset Series)
- ISO19139
  - /collections/services/items/tb19-d123?httpAccept=application/vnd.iso.19139%2Bxml
  - /collections/services/items/tb19-d124?httpAccept=application/vnd.iso.19139%2Bxml
  - /collections/services/items/tb19-d125?httpAccept=application/vnd.iso.19139%2Bxml
  - /collections/series/items/tb19-osm?httpAccept=application/vnd.iso.19139%2Bxml
  - /collections/series/items/tb19-rtd?httpAccept=application/vnd.iso.19139%2Bxml
- GeoDCAT-AP (JSON-LD)
  - /collections/services/items/tb19-d123?httpAccept=application/ld%2Bjson;profile=http://data.europa.eu/930/
  - /collections/services/items/tb19-d124?httpAccept=application/ld%2Bjson;profile=http://data.europa.eu/930/
  - /collections/services/items/tb19-d125?httpAccept=application/ld%2Bjson;profile=http://data.europa.eu/930/
  - /collections/series/items/tb19-osm?httpAccept=application/ld%2Bjson;profile=http://data.europa.eu/930/
  - /collections/series/items/tb19-rtd?httpAccept=application/ld%2Bjson;profile=http://data.europa.eu/930/
- GeoDCAT-AP (Turtle)
  - /collections/services/items/tb19-d123?httpAccept=text/turtle;profile=http://data.europa.eu/930/
  - /collections/services/items/tb19-d124?httpAccept=text/turtle;profile=http://data.europa.eu/930/

- /collections/services/items/tb19-d125?httpAccept=text/turtle;profile=http://data.europa.eu/930/
- /collections/series/items/tb19-osm?httpAccept=text/turtle;profile=http://data.europa.eu/930/
- /collections/series/items/tb19-rtd?httpAccept=text/turtle;profile=http://data.europa.eu/930/

Search Examples:

- /collections/services/items?q=Testbed-19
- /collections/services/items?externalId=tb19-d123
- /collections/services/items?filter=organisationName%20=%20%27interactive%20instruments%20GmbH%27

The open-source software StacBrowser can be used to access the catalog server. The below example shows the search result of a search request combining free-text (`q`) and `organisationName` with a CQL2 filter.



EO services and applications

in FedEO Clearinghouse

Description

Metadata records representing EO services and applications

License	various
Temporal Extent	n/a

Metadata

General

ItemType	feature

Additional Resources

Alternative representation

- Self as HTML
- Self as JSON-LD

Description of the resource

- JSON schema for items belonging to this collection

Items

Filters

Temporal Extent

All times in Coordinated Universal Time (UTC).

Spatial Extent

Filter by spatial extent

Additional filters

Match all filters (and) Match any filters (or)

API Records Query	=	Testbed-19
Organisation name	=	interactive instruments GmbH

Items per page

Default (12)

Number of items requested per page, max. 10000 items.

D124: OGC API – Features serving OpenStreetMap data

No time given

D125: OGC API - Features supporting real-time observations

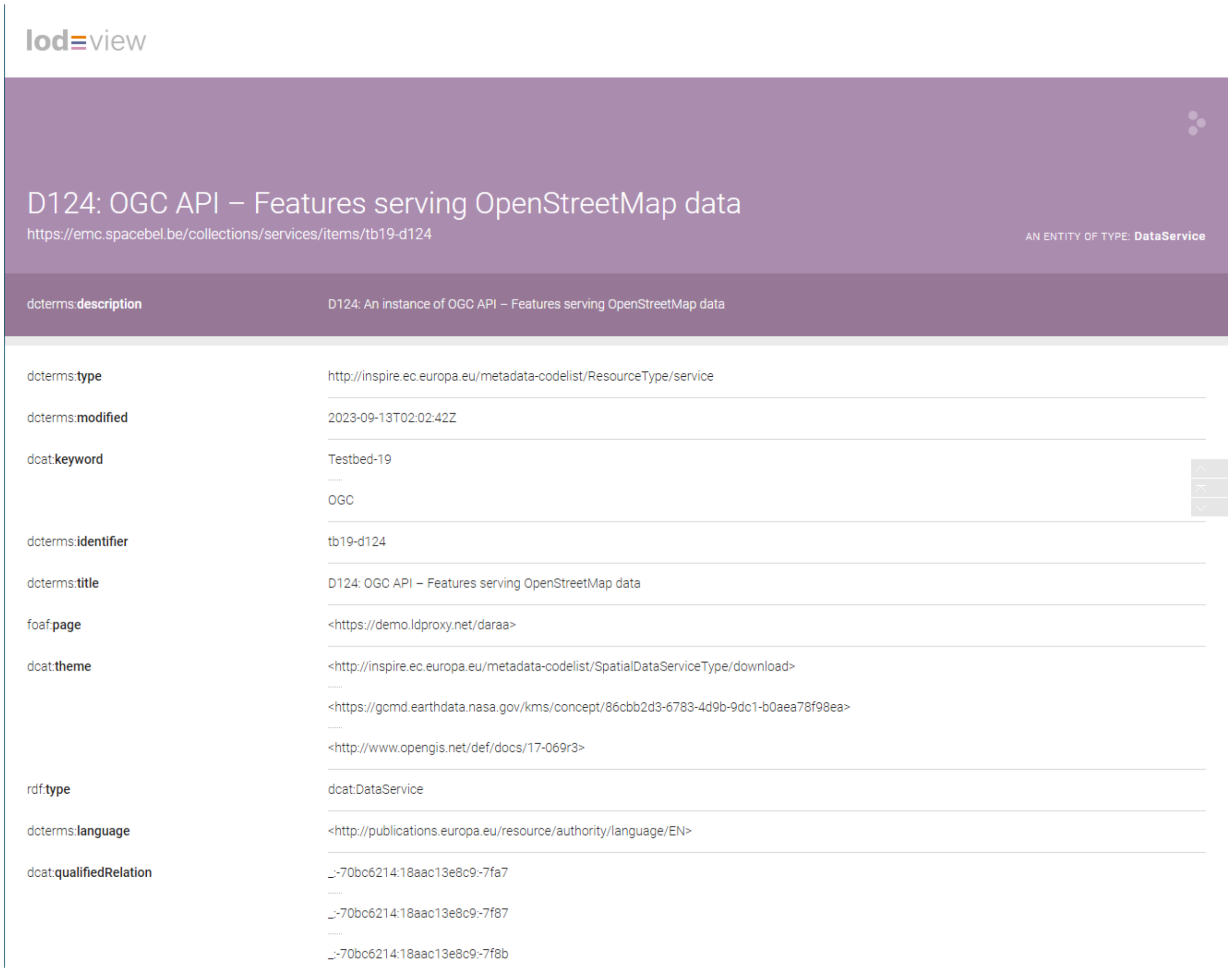
No time given

Figure A.34 — Accessing the API-Records catalog with StacBrowser Client

A.7.5. OGC Building Blocks

The GeoDCAT-AP responses of the API-Records implementation can be browsed as Linked Data, e.g., using the LODView open-source software. They are integrated with the knowledge graph behind OGC RAINBOW as shown below.

For example, https://emc.spacebel.be/lodview/ext/?uri=https%3A%2F%2Femc.spacebel.be%2Fcollections%2Fservices%2Fitems%2Ftb19-d124 accesses the /collections/services/items/tb19-d124 response as Linked Data.



D124: OGC API – Features serving OpenStreetMap data

https://emc.spacebel.be/collections/services/items/tb19-d124

dcterms:**description**	D124: An instance of OGC API – Features serving OpenStreetMap data
dcterms:**type**	http://inspire.ec.europa.eu/metadata-codelist/ResourceType/service
dcterms:**modified**	2023-09-13T02:02:42Z
dcat:**keyword**	Testbed-19
	OGC
dcterms:**identifier**	tb19-d124
dcterms:**title**	D124: OGC API – Features serving OpenStreetMap data
foaf:**page**	<https://demo.ldproxy.net/daraa>
dcat:**theme**	<http://inspire.ec.europa.eu/metadata-codelist/SpatialDataServiceType/download>
	<https://gcmd.earthdata.nasa.gov/kms/concept/86cbb2d3-6783-4d9b-9dc1-b0aea78f98ea>
	<http://www.opengis.net/def/docs/17-069r3>
rdf:**type**	dcat:DataService
dcterms:**language**	<http://publications.europa.eu/resource/authority/language/EN>
dcat:**qualifiedRelation**	_:-70bc6214:18aac13e8c9:-7fa7
	_:-70bc6214:18aac13e8c9:-7f87
	_:-70bc6214:18aac13e8c9:-7f8b

Figure A.35 — Accessing D124 metadata as Linked Data (GeoDCAT-AP)

The GeoJSON, JSON-LD, RDF/XML or Turtle representations of the metadata records reference the OGC API Standard they implement as shown below.

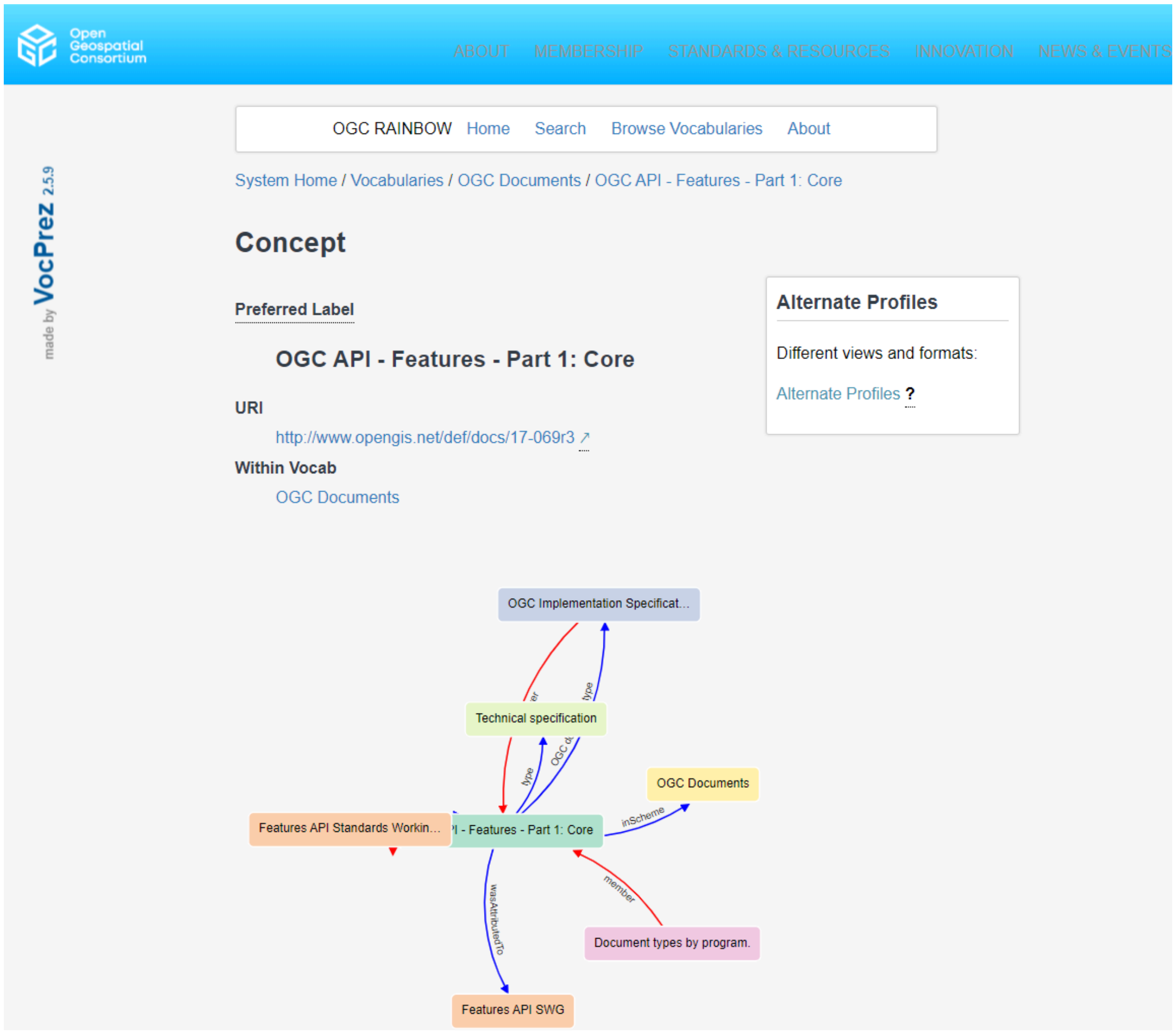
```
"categories": [
      {
        "scheme": "https://inspire.ec.europa.eu/metadata-codelist/
ProtocolValue",
        "term": "http://www.opengis.net/def/docs/18-062r2",
        "label": "OGC API-Processes"
      }
```

```
    ]
```

Figure A.36 — Reference to API description in RAINBOW (GeoJSON)

```
"dcat:theme": [
      {
        "skos:prefLabel": "OGC API-Features",
        "@type": "skos:Concept",
        "@id": "http://www.opengis.net/def/docs/17-069r3",
        "skos:inScheme": "https://inspire.ec.europa.eu/metadata-codelist/
ProtocolValue"
      }
   ]
```

Figure A.37 — Reference to API description in RAINBOW (JSON-LD)



System Home / Vocabularies / OGC Documents / OGC API - Features - Part 1: Core

Concept

Preferred Label

OGC API - Features - Part 1: Core

URI

http://www.opengis.net/def/docs/17-069r3

Within Vocab

OGC Documents

Alternate Profiles

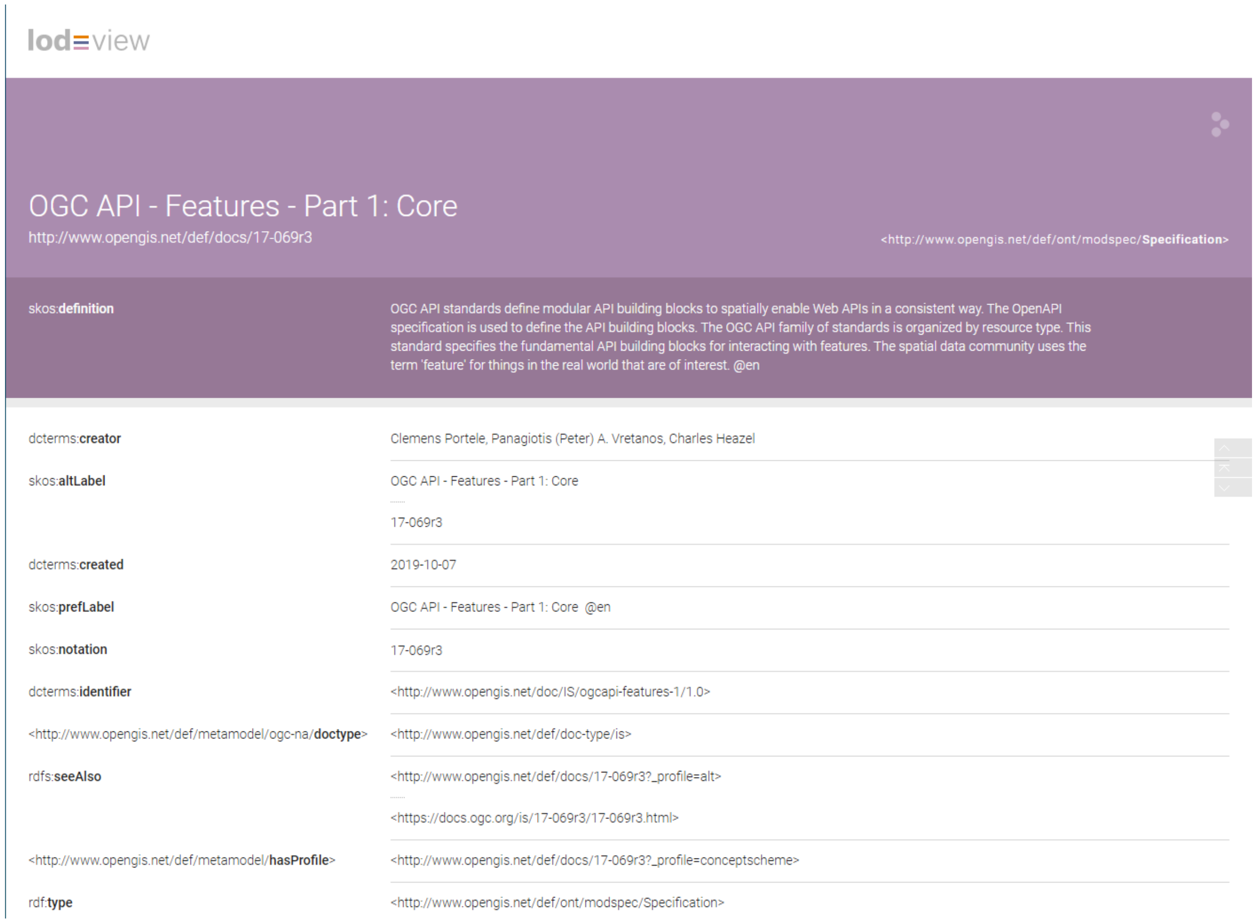
Different views and formats:

Alternate Profiles ?

Figure A.38 — Description OGC API - Features - Part 1 in RAINBOW

Following the corresponding reference in the LODView client allows navigating to the corresponding definition in the OGC RAINBOW. OGC RAINBOW publishes a range of resources managed by OGC and identified by unique stable web addresses (IRIs or URIs). The following RAINBOW elements are relevant in the metadata records.

- http://defs.opengis.net/vocprez/object?uri=http://www.opengis.net/def/docs/18-062r2
- http://defs.opengis.net/vocprez/object?uri=http://www.opengis.net/def/docs/17-069r3



OGC API - Features - Part 1: Core

http://www.opengis.net/def/docs/17-069r3

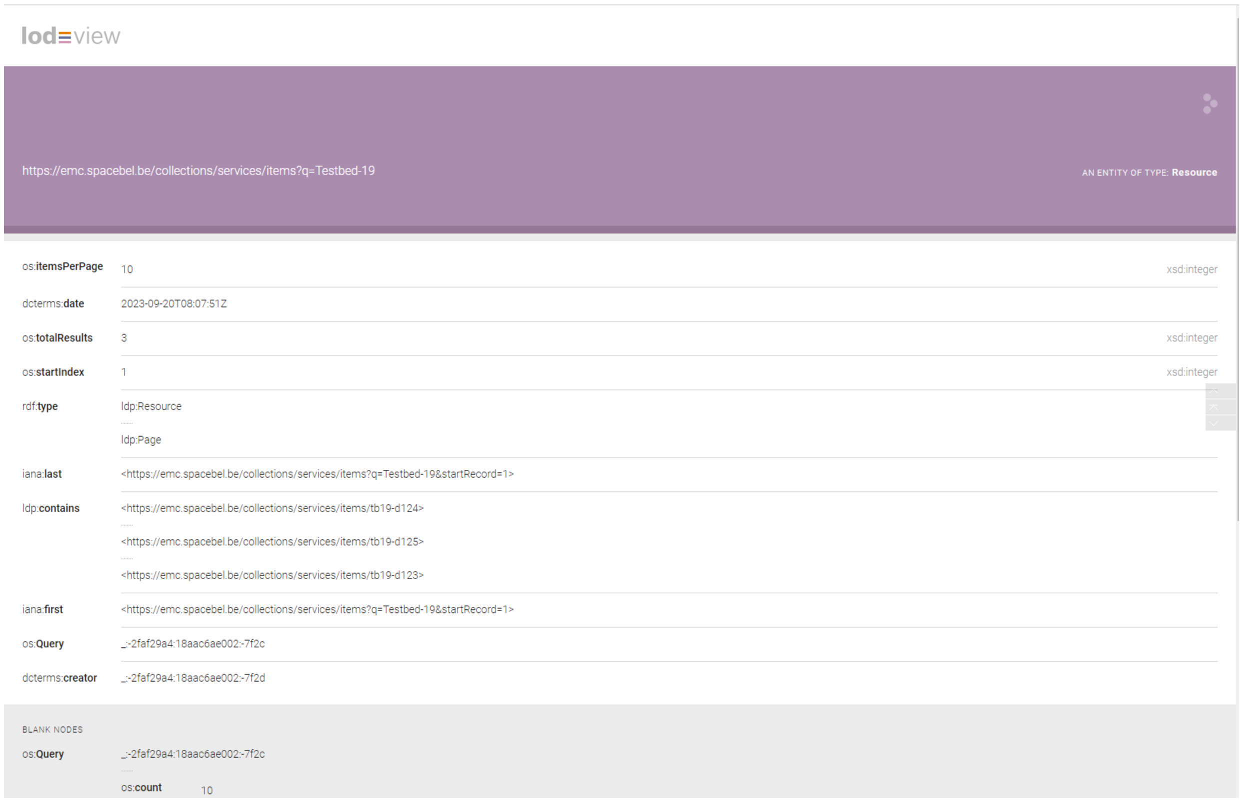
<http://www.opengis.net/def/ont/modspec/Specification>

skos:definition	OGC API standards define modular API building blocks to spatially enable Web APIs in a consistent way. The OpenAPI specification is used to define the API building blocks. The OGC API family of standards is organized by resource type. This standard specifies the fundamental API building blocks for interacting with features. The spatial data community uses the term 'feature' for things in the real world that are of interest. @en
dcterms:creator	Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel
skos:altLabel	OGC API - Features - Part 1: Core
	17-069r3
dcterms:created	2019-10-07
skos:prefLabel	OGC API - Features - Part 1: Core @en
skos:notation	17-069r3
dcterms:identifier	<http://www.opengis.net/doc/IS/ogcapi-features-1/1.0>
<http://www.opengis.net/def/metamodel/ogc-na/doctype>	<http://www.opengis.net/def/doc-type/is>
rdfs:seeAlso	<http://www.opengis.net/def/docs/17-069r3?_profile=alt>
	<https://docs.ogc.org/is/17-069r3/17-069r3.html>
<http://www.opengis.net/def/metamodel/hasProfile>	<http://www.opengis.net/def/docs/17-069r3?_profile=conceptscheme>
rdf:type	<http://www.opengis.net/def/ont/modspec/Specification>

Figure A.39 — Accessing Building Block metadata as Linked Data (RAINBOW)

In a similar way, any D128 API Records search response can be obtained in RDF format (linked data) via the LODView Client as the catalog supports JSON-LD, RDF/XML, and Turtle encodings of search responses in addition to GeoJSON and HTML.

- https://emc.spacebel.be/lodview/ext/?uri=https%3A%2F%2Femc.spacebel.be%2Fcollections%2Fservices%2Fitems%3Fq%3DTestbed-19



lod≡view

https://emc.spacebel.be/collections/services/items?q=Testbed-19

AN ENTITY OF TYPE: **Resource**

os:**itemsPerPage**	10	xsd:integer
dcterms:**date**	2023-09-20T08:07:51Z	
os:**totalResults**	3	xsd:integer
os:**startIndex**	1	xsd:integer
rdf:**type**	ldp:Resource ldp:Page	
iana:**last**	<https://emc.spacebel.be/collections/services/items?q=Testbed-19&startRecord=1>	
ldp:**contains**	<https://emc.spacebel.be/collections/services/items/tb19-d124> <https://emc.spacebel.be/collections/services/items/tb19-d125> <https://emc.spacebel.be/collections/services/items/tb19-d123>	
iana:**first**	<https://emc.spacebel.be/collections/services/items?q=Testbed-19&startRecord=1>	
os:**Query**	_:-2faf29a4:18aac6ae002:-7f2c	
dcterms:**creator**	_:-2faf29a4:18aac6ae002:-7f2d	

BLANK NODES

os:**Query**	_:-2faf29a4:18aac6ae002:-7f2c	
	os:**count**	10

Figure A.40 — Accessing API Records search responses as Linked Data (GeoDCAT-AP)

A.7.6. Decentralized identifiers

W3C Decentralized Identifiers (DID) can be used to find binding information for services without the need to retrieve it from a centralized catalog. The identifiers also support the dereferencing of the service requests, increasing location independence, and data portability that can be combined with cryptographical verification.

For example `did:web:emc.spacebel.be` is a `DID` identifier referring to the organization 'Spacebel'. The DID resolves to a `DID document` published at https://emc.spacebel.be/.well-known/did.json as can be checked with https://dev.uniresolver.io/1.0/identifiers/did:web:emc.spacebel.be.

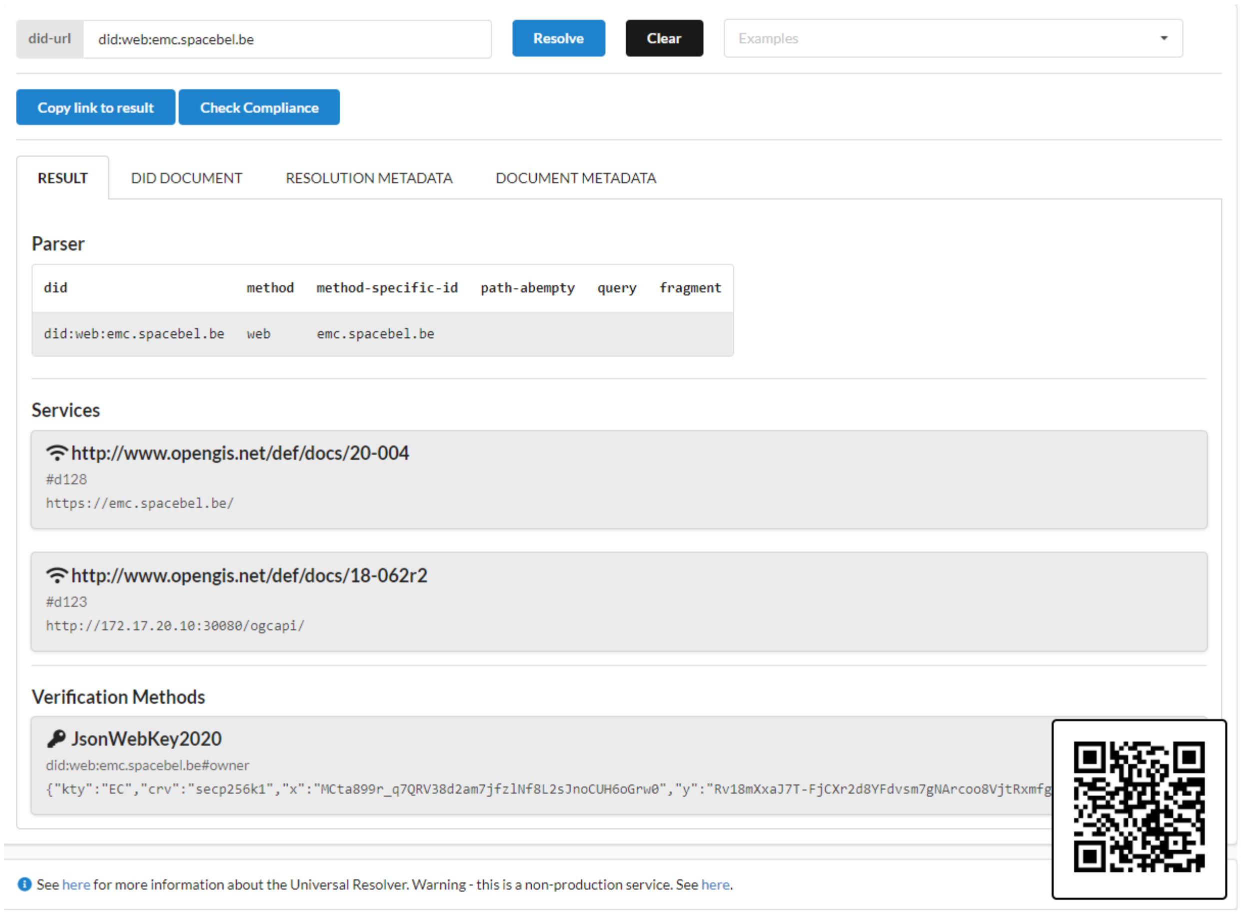
```
{
 "@context": [
  "https://www.w3.org/ns/did/v1",
  "https://w3id.org/security/suites/jws-2020/v1"
 ],
 "id": "did:web:emc.spacebel.be",
 "verificationMethod": [
  {
   "id": "did:web:emc.spacebel.be#owner",
   "type": "JsonWebKey2020",
   "controller": "did:web:emc.spacebel.be",
   "publicKeyJwk": {
    "kty": "EC",
    "crv": "secp256k1",
    "x": "MCta899r_q7QRV38d2am7jfzlNf8L2sJnoCUH6oGrw0",
    "y": "Rv18mXxaJ7T-FjCXr2d8YFdvsm7gNArcoo8VjtRxmfg"
   }
  }
 ],
 "authentication": [
  "did:web:emc.spacebel.be#owner"
 ],
 "assertionMethod": [
  "did:web:emc.spacebel.be#owner"
 ],
 "service": [
  {
   "id": "#d128",
   "type": "http://www.opengis.net/def/docs/20-004",
   "serviceEndpoint": "https://emc.spacebel.be/"
  },
  {
   "id": "#d123",
   "type": "http://www.opengis.net/def/docs/18-062r2",
   "serviceEndpoint": "http://172.17.20.10:30080/ogcapi/"
  }
 ]
}
```

Figure A.41 — DID document with service information

The OGC Testbed-19 services published by Spacebel are identified in the above `DID document`. A DID resolver can access the published services via the following DID URLs.

- `did:web:emc.spacebel.be#d123` or `did:web:emc.spacebel.be?service=d123`
- `did:web:emc.spacebel.be#d128` or `did:web:emc.spacebel.be?service=d128`

The image below depicts the output of the DIF Universal Resolver for the above Spacebel DID.



Parser

did	method	method-specific-id	path-abempty	query	fragment
did:web:emc.spacebel.be	web	emc.spacebel.be			

Services

http://www.opengis.net/def/docs/20-004
#d128
https://emc.spacebel.be/

http://www.opengis.net/def/docs/18-062r2
#d123
http://172.17.20.10:30080/ogcapi/

Verification Methods

JsonWebKey2020
did:web:emc.spacebel.be#owner
```
{"kty":"EC","crv":"secp256k1","x":"MCta899r_q7QRV38d2am7jfzlNf8L2sJnoCUH6oGrw0","y":"Rv18mXxaJ7T-FjCXr2d8YFdvsm7gNArcoo8VjtRxmfg
```

See here for more information about the Universal Resolver. Warning - this is a non-production service. See here.

Figure A.42 — DIF Universal Resolver

In the above implementation, the D123 service metadata record refers back to its DID identifier as shown below.

```
"links": [
    {
      "rel": "describes",
      "href": "did:web:emc.spacebel.be#d123",
      "type": "application/did+ld+json",
      "title": "DID Document"
    }
 ]
```

Figure A.43 — Catalog record referring to resource DID identifier/document.

Some more examples of dereferencing DID URLs with another universal resolver:

- https://api.godiddy.com/0.1.0/universal-resolver/identifiers/did:web:emc.spacebel.be
- https://api.godiddy.com/0.1.0/universal-resolver/identifiers/did:web:emc.spacebel.be?service=d128&relativeRef=collections/services/items/tb19-d125
- https://api.godiddy.com/0.1.0/universal-resolver/identifiers/did:web:emc.spacebel.be?service=d128&relativeRef=api

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B ANNEX B (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	PERSON	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-12-06	.9	Lucio Colaiacomo	all	Initial version
2024-01-16	.9	Carl Reed	all	Apply edits from detailed review

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Yves Coene, Christophe Noel: OGC 22-018, *Testbed-18: Secure Asynchronous Catalog Engineering Report*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/PER/T18-D007.

[2] Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1.

[3] George Percivall: OGC 03-040, *OGC Reference Model*. Open Geospatial Consortium (2003).

[4] George Percivall: OGC 08-062r7, *OGC Reference Model*. Open Geospatial Consortium (2011). http://www.opengis.net/doc/orm/2.1.

[5] Yves Coene: OGC 19-020r1, *OGC Testbed-15: Catalogue and Discovery Engineering Report*. Open Geospatial Consortium (2019). http://www.opengis.net/doc/PER/t15-D010.

[6] Scott Simmons: OGC 05-020r29, Technical Committee Policies and Procedures, version 29.0, https://docs.ogc.org/pol/05-020r29/05-020r29.html

[7] Panagiotis (Peter) A. Vretanos, Tom Kralidis, Charles Heazel: OGC 20-004, **OGC API – Records – Part 1: Core**, 2019 https://docs.ogc.org/DRAFTS/20-004.html

[8] SEED Standard Definition — https://ngageoint.github.io/seed/seed.html.

[9] W3C: **Decentralized Identifiers (DIDs) v1.0, Core architecture, data model, and representations**, W3C Recommendation 19 July 2022, Version 1.0 https://www.w3.org/TR/did-core

[10] DGIWG : DGIWG 114, **DGIWG Metadata Foundation**, 12 July 2017, Edition 2.0, https://portal.dgiwg.org/files/67565

[11] SEMIC: **GeoDCAT-AP — Version 2.0.0, A geospatial extension for the DCAT application profile for data portals in Europe**, Version 2.0 https://semiceu.github.io/GeoDCAT-AP/releases/2.0.0/

[12] ITU-T: X.901-1997, **Information technology – Open distributed processing – Reference Model: Overview**, https://rm-odp-new.lcc.uma.es/files/X.901-1997.pdf

[13] DGIWG : DGIWG 933, **DGIWG Geospatial Reference Architecture (DGRA)**, 18 May 2023, Edition 1.0, https://portal.dgiwg.org/files/?artifact_id=73392

[14] ISO: ISO 19139:2007, **Geographic Information – Metadata XML**, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32557

[15] Docker object labels — https://docs.docker.com/config/labels-custom-metadata/.

[16] Kubernetes Labels and Selectors — https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/.

[17] Decentralized Identity Foundation: Decentralized Web Node — https://identity.foundation/decentralized-web-node/spec/.

[18] Indy: Decentralized Key Management — https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/design/005-dkms/README.html.

[19] Anna Burzykowska (ESA), Michele Iapaolo (Randstad), Priit Anton (Guardtime), Andreas Sisask (Guardtime), EO Data Provenance with KSI Blockchain, February 2020. — https://eo4society.esa.int/wp-content/uploads/2020/03/EO-data-provenance-with-KSI-blockchain-Feb-2020.pdf.

[20] **The Open Group Architecture Framework**, https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap32.html#tag_33_03