M17 – Compliance with the Metadata Quality Assurance (MQA) tool for datasets

Milestone Lead	UNITUS
Milestone due date	2024/07/31
Status	FINAL
Version	V1.0
Project	SEBASTIEN

Co-financed by the Connecting Europe Facility of the European Union

Agreement: INEA/CEF/ICT/A2020/2373580 **Action**: 2020-IT-IA-0234

DOCUMENT INFORMATION

Title	Milestone 17
Agreement	INEA/CEF/ICT/A2020/2373580
Action	2020-IT-IA-0234
Creator	UNITUS
Milestone Description	Compliance with the Metadata Quality Assurance (MQA) tool for datasets
Means of verification	Relevant datasets (including metadata) resulting from the action are published on a national portal or catalogue that is harvested by the European Data Portal, under which the MQA can be performed; confirmation in the final report submitted to HaDEA
Contributors	Marco Milanesi (UNITUS), Gabriel Cerveira (CMCC), Alessandro D'Anca (CMCC)
Requested deadline	M31
Reviewer	Giuseppe Trotta (CINECA)

SEBASTIEN

Table of content

1. Introduction

This document describes the procedures and techniques used within the SEBASTIEN project in order to ensure the MQA compliance with the European Data Portal of the datasets produced and their dissemination through a national portal (in particular the Italian open data portal) that is harvested by the European Data Portal.

In particular, Chapter 2 describes the characteristics required to guarantee MQA compliance with the European Data Portal while Chapter 3 presents the implementation features of the solution identified to guarantee compliance with the DCAT-AP API specifications. Chapter 4 reports the results obtained through the validation of the datasets with the SHACL validator made available by the European Data Portal. Finally, Chapter 5 presents information on the development required to guarantee the on-boarding of the datasets on the Italian open data portal through compliance with the DCAT-AP IT API.

2. Metadata Quality Assurance

In an effort to improve the quality of open data and to help data providers understand the strength and weaknesses of their metadata, the European Data Portal implemented a process called Metadata Quality Assessment (MQA). This tool periodically calculates a score to determine the quality of the information about the datasets published through the portal's harvester. The methodology applied for the score calculation is derived from the four dimensions of the FAIR principles (https://www.go-fair.org/fair-principles/) and from the DCAT-AP (https://www.w3.org/TR/vocab-dcat/) specification.

Since most of the data online today is produced, found and consumed through automated processes, the portal's MQA focuses on evaluating the quality through this perspective; this is where the dimensions of the FAIR principles (Wilkinson et al., 2016) become useful. The four FAIR dimensions are:

- Findability, which says the data should be easy to find, and therefore it should be machine-readable;
- Accessibility, which says there should be clear information on how the data should be accessed;
- Interoperability, which says the data should be operable through different applications and workflows;
- Reusability, which says the data should be easily replicable in different settings.

The most fundamental layer of the MQA is the metadata's compliance to the DCAT-AP specification. This is an application profile for data portals in Europe, derived from the Data Catalogue vocabulary (DCAT). The DCAT-AP was created with the goal of making data searchable and usable across sectors and to allow interoperability with applications that use other established vocabularies.

The combination of the DCAT-AP with the FAIR principles' dimensions allowed the European Data Portal to determine the criteria for the scoring in the MQA process, which uses DCAT-AP fields that correspond to the FAIR principles, as can be seen in *Table 1*.

FAIR Dimension	DCAT-AP Field	Weight
Findability	dcat:keyword	30
Findability	dcat:theme	30
Findability	dct:spatial	20
Findability	dct:temporal	20
Accessibility	dcat:accessURL	50
Accessibility	dcat:downloadURL	50
Interoperability	dct:format	50
Interoperability	dcat:mediaType	20
Interoperability	DCAT-AP compliance	30
Reusability	dct:license	30
Reusability	dct:accessRights	15

Reusability	dcat:contactPoint	20
Reusability	dct:publisher	10

Table 1: Relation of fields considered for the MQA score, along with the related FAIR dimension and the respective weight of the field on the overall score.

In order to be compliant with the DCAT-AP profile, there are only two mandatory fields: *dct:title* and *dct:description.* These fields are not included in the MQA score, since without them the dataset couldn't be published in the portal. The fields in the above table are non-mandatory according to the DCAT-AP, but are considered highly recommended by the European Data Portal in order to guarantee the high quality of the published metadata. The complete specification for the DCAT-AP can be found here: https://semiceu.github.io/DCAT-AP/releases/3.0.0/.

3. DCAT-AP in SEBASTIEN datasets

In the scope of the SEBASTIEN project, the metadata for the generated datasets was originally provided through a REST API, which provides this metadata in the JSON format. The original metadata can be fetched either as a list comprising information about all the datasets (https://sebastien-datalake.cmcc.it/api/v2/datasets) or as a single JSON for a specific dataset (https://sebastien-datalake.cmcc.it/api/v2/datasets/{dataset}). Table 2 shows a list of the dataset IDs, as defined in the SEBASTIEN API, which were converted for harvesting by the European Data Portal.

ID	Description
blue-tongue	Blue Tongue monthly data
iot-animal	IoT Animal data
pasture	Pasture total quantity and dry substance derived from Remote Sensing Indices
pi	Production Indices derived from MISTRAL COSMO-2I weather forecast data
pi-long-term	Production Indices derived from VHR-PRO future

	scenario
thi	Thermohygrometric Indices derived from MISTRAL COSMO-2I weather data

Table 2: List of IDs and description of datasets that were converted.

To reach the goal of publishing the SEBASTIEN metadata to the European Data Portal, several steps were followed, in particular:

1) Compare the original metadata schema with the DCAT-AP specification, to determine which fields were missing and which could be converted to a compliant field;

2) From the formats accepted by the European Data Portal, determine which one would be most effective to adopt;

3) After determining the format, develop the code for the conversion to DCAT-AP;

4) Provide the converted metadata through an API and validate the content for compliance with DCAT-AP.

On the first step, upon analysing the current schema and fields available on the original JSON metadata, it was evident that there were fields equivalent to the mandatory DCAT-AP fields and to some of the recommended fields for MQA improvement, as can be seen in *Table 3*. After determining what needed to be done to convert the metadata, two methods of providing it to the European Data Portal were evaluated: one was providing it through a CKAN (https://ckan.org/) instance, using an extension for DCAT-AP, and the other, which was chosen, was to provide it in the RDF (Resource Description Framework)/XML format, through the OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) protocol.

Original SEBASTIEN field	**DCAT-AP Equivalent**
metadata.description	dct:description
metadata.contact.name	dcat:contactPoint
metadata.label	dct:title
metadata.doi	dct:identifier
metadata.publication_date	dct:issued

metadata.id	dct:identifier

Table 3: Listing of original SEBASTIEN fields that could be directly translated to equivalent DCAT-AP fields

The OAI-PMH is a method for repository interoperability, it aims to provide a structured way for data providers (repositories) to expose their metadata and for service providers to harvest it. OAI-PMH defines a set of six verbs that are invoked via HTTP, facilitating the exchange of metadata between systems.

For the purposes of integration with the European Data Portal, and following their requirements, only the ListRecords verb was implemented. This verb allows for the batch harvesting of metadata records, which is sufficient for providing the SEBASTIEN datasets' metadata to the portal. The decision to use OAI-PMH was made because it allowed for easy extension of the existing API with this new capability, minimizing the need for extensive changes to the current infrastructure, such as setting up a separate data management system like CKAN.

The implementation process involved developing a Python-based solution for converting the original JSON metadata into the DCAT-AP compliant RDF/XML format. This conversion process utilized a library specifically designed for working with RDF data, called RDFlib, which ensured accurate transformation of the metadata fields according to the DCAT-AP specification. The code for the conversion can be seen below.

```
from rdflib import Graph, Literal, Namespace, RDF, URIRef, BNode
from rdflib.namespace import DCAT, DCTERMS, FOAF, RDF
import logging
from datetime import datetime

# Logging config
logging.basicConfig(level=logging.DEBUG)

# Namespaces for DCAT-AP, to be binded to the RDF graph
DCAT = Namespace("http://www.w3.org/ns/dcat#")
DCT = Namespace("http://purl.org/dc/terms/")
FOAF = Namespace("http://xmlns.com/foaf/0.1/")
VCARD = Namespace("http://www.w3.org/2006/vcard/ns#")
EDP = Namespace("https://europeandataportal.eu/voc#")
SPDX = Namespace("http://spdx.org/rdf/terms#")
ADMS = Namespace("http://www.w3.org/ns/adms#")
DQV = Namespace("http://www.w3.org/ns/dqv#")
SKOS = Namespace("http://www.w3.org/2004/02/skos/core#")
SCHEMA = Namespace("http://schema.org/")
# Namespace for DCAT-AP IT
DCATAPIT = Namespace("http://dati.gov.it/onto/dcatapit#")

# Define classes for DCAT-AP entities (Dataset, Distribution and ContactPoint)
```

```
class ContactPoint:
    def __init__(self, name=None, email=None, webpage=None):
        self.name = name
        self.email = email
        self.webpage = webpage

class Distribution:
    def __init__(self, access_url=None, description=None, download_url=None,
                 media_type=None, format=None, rights=None, license=None,
identifier=None):
        self.access_url = access_url
        self.description = description
        self.download_url = download_url
        self.media_type = media_type
        self.format = format
        self.rights = rights
        self.license = license
        self.identifier = identifier

class DatasetDCAT:
    def __init__(self, uri, title=None, description=None, issued=None,
identifier=None, contact_point=None):
        self.uri = uri
        self.title = title
        self.description = description
        self.issued = issued
        self.identifier = identifier
        self.contact_point = contact_point
        self.distributions = []

    def add_distribution(self, distribution):
        self.distributions.append(distribution)

    # Build the RDF graph for the dataset
    def to_graph(self, g):
        dataset = URIRef(self.uri)
        g.add((dataset, RDF.type, DCAT.Dataset))
        logging.debug(f"Adding to graph {g.identifier}: {dataset} a type
{DCAT.Dataset}")

        if self.title:
            g.add((dataset, DCT.title, Literal(self.title)))
        if self.description:
            g.add((dataset, DCT.description, Literal(self.description)))
        if self.issued:
            g.add((dataset, DCTERMS.issued, Literal(self.issued,
datatype=DCTERMS.W3CDTF)))
        if self.identifier:
            g.add((dataset, DCTERMS.identifier, Literal(self.identifier)))
```

```
        if self.contact_point:
            contact_bnode = BNode()
            g.add((dataset, DCAT.contactPoint, contact_bnode))
            g.add((contact_bnode, RDF.type, VCARD.Kind))
            if self.contact_point.name:
                g.add((contact_bnode, VCARD.fn, Literal(self.contact_point.name)))
            if self.contact_point.email:
                g.add((contact_bnode, VCARD.hasEmail,
URIRef(f"mailto:{self.contact_point.email}")))
            if self.contact_point.webpage:
                g.add((contact_bnode, VCARD.hasURL,
URIRef(self.contact_point.webpage)))

        for dist in self.distributions:
            distribution_bnode = BNode()
            g.add((dataset, DCAT.distribution, distribution_bnode))
            g.add((distribution_bnode, RDF.type, DCAT.Distribution))
            if dist.access_url:
                g.add((distribution_bnode, DCAT.accessURL, URIRef(dist.access_url)))
            if dist.description:
                g.add((distribution_bnode, DCTERMS.description,
Literal(dist.description)))
            if dist.download_url:
                g.add((distribution_bnode, DCAT.downloadURL,
URIRef(dist.download_url)))
            if dist.media_type:
                g.add((distribution_bnode, DCTERMS.mediaType,
URIRef(dist.media_type)))
            if dist.format:
                g.add((distribution_bnode, DCTERMS.format, URIRef(dist.format)))
            if dist.rights:
                rights_bnode = BNode()
                g.add((distribution_bnode, DCTERMS.rights, rights_bnode))
                g.add((rights_bnode, RDF.type, DCTERMS.RightsStatement))
                g.add((rights_bnode, DCTERMS.rights, URIRef(dist.rights)))
            if dist.license:
                license_bnode = BNode()
                g.add((distribution_bnode, DCTERMS.license, license_bnode))
                g.add((license_bnode, RDF.type, DCTERMS.LicenseDocument))
                g.add((license_bnode, DCTERMS.license, URIRef(dist.license)))
            if dist.identifier:
                g.add((distribution_bnode, DCTERMS.identifier,
Literal(dist.identifier)))

        return g

def convert_to_dcat_ap(data, url):
    logging.debug("Starting convert_to_dcat_ap function")
```

```
    g = Graph()

    # Bind namespaces
    g.bind("dcat", DCAT)
    g.bind("DCT", DCT)
    g.bind("foaf", FOAF)
    g.bind("vcard", VCARD)
    g.bind("edp", EDP)
    g.bind("spdx", SPDX)
    g.bind("adms", ADMS)
    g.bind("dqv", DQV)
    g.bind("skos", SKOS)
    g.bind("schema", SCHEMA)

    # Placeholder URI
    dataset_uri = url


    if not isinstance(data, list):
        data = [data]

    for dataset in data:
        # Check if "dataset" key is present, if it isn't, wrap the dict in it
        if "dataset" not in dataset:
            dataset = {"dataset": dataset}

        # Add the URL to the data
        dataset["url"] = url

        # Create dataset and convert the original field names to DCAT-AP
        metadata = DatasetDCAT(
            uri=f'{dataset_uri}/{dataset.get("dataset", {}).get("metadata",
{}).get("id")}',
            title=dataset.get("dataset", {}).get("metadata", {}).get("label"),
            description=dataset.get("dataset", {}).get("metadata",
{}).get("description"),
            issued=dataset.get("dataset", {}).get("metadata",
{}).get("publication_date"),
            identifier=dataset.get("dataset", {}).get("metadata", {}).get("id"),
        )

        # Create contact point and convert the original field names to DCAT-AP
        contact = dataset.get("dataset", {}).get("metadata", {}).get("contact")
        contact_point = ContactPoint(
            name=contact.get("name"),
            email=contact.get("email"),
            webpage=contact.get("webpage"),
        )
```

```
        metadata.contact_point = contact_point

        # Create distributions and convert the original field names to DCAT-AP
        products = dataset.get("dataset", {}).get("products", {}).get("monthly", {})
        distribution = Distribution(
            access_url=url,
            description=products.get("description"),
        )
        metadata.add_distribution(distribution)

        # Add dataset to graph
        metadata.to_graph(g)

    return g
```

To serve the converted metadata through the OAI-PMH protocol, a server implementation was developed using Python, with the pyoai library. This server handles OAI-PMH requests, specifically the ListRecords verb, and responds with the appropriate DCAT-AP compliant metadata in RDF/XML format. The server was integrated with FastAPI to provide the necessary HTTP endpoints for the OAI-PMH protocol. The endpoint for providing the DCAT-AP metadata (i.e. https://sebastien-datalake.cmcc.it/api/v2/oai/blue-tongue?verb=ListRecords for the blue-tongue dataset) was set up in such a way that when a GET or POST request is sent to it, it sends a ListRecords request to the OAI-PMH repository, and fetches the information for the required dataset. The relevant code for these functionalities is below.

```
# Define OAI-PMH endpoint route
@app.get("/oai/{dataset_id}")
@app.post("/oai/{dataset_id}")
def oai(request: Request, dataset_id: str):
    params = dict(request.query_params)

    # Add dataset_id to the parameters as "set_", which is a parameter from the
OAI-PMH protocol
    params['set'] = dataset_id

    # Making sure it uses the dcat_ap metadata prefix
    if 'metadataPrefix' not in params:
        params['metadataPrefix'] = 'dcat_ap'

    # handleRequest points the request to the appropriate method in
metadata_provider.py
    response = oai_server.oai_server.handleRequest(params)
    logging.debug(f"OAI-PMH Response: {response}")
    # Replace date in datestamp by empty string
    response = re.sub(b'<datestamp>.*</datestamp>', b'', response)
```

```
    return Response(content=response, media_type="text/xml")
```

```
# Defining listRecords method, only method used by data.europa harvester
    def listRecords(self, metadataPrefix='dcat_ap', from_=None, until=None,
set=None):
        logging.debug("Fetching data from API")

        if set:
            dataset_url = f"{BASE_URL}/{set}"
        else:
            dataset_url = BASE_URL

        # Fetch data from the dataset endpoint
        data = main.fetch_data(
            dataset_url
        )
        logging.debug(f"Fetched data: {data}")

        # Convert to RDF graph with proper DCAT-AP fields (URL is being used to
fill the accessURL field)
        rdf_graph = convert_to_dcat_ap(data, dataset_url)

        # Serialize the RDF graph into a string, 'pretty-xml' format makes it
more readable
        rdf_string = rdf_graph.serialize(format='pretty-xml')
        logging.debug(f"RDF string: {rdf_string}")

        # Create a header (mandatory for OAI-PMH)
        header_element = Element("header")
        header = Header(deleted=False, element=header_element, identifier="",
datestamp=datetime.utcnow(), setspec=[])

        # Create metadata element and fill it with the RDF/XML string
        metadata_element = Element("metadata")
        metadata = Metadata(element=metadata_element, map={"rdf": rdf_string})


        return [(header, metadata, [])], None
```

This approach allowed for a seamless integration between the existing SEBASTIEN data infrastructure and the requirements of the European Data Portal. By implementing the OAI-PMH protocol and converting the metadata to DCAT-AP compliant RDF/XML, we ensured that the

SEBASTIEN datasets could be easily harvested and integrated into the broader European data ecosystem.

The next steps in the process involved testing and validating the converted metadata and the OAI-PMH server implementation. This included verifying the correctness of the DCAT-AP compliant metadata, through the portal's provided SHACL validator (https://data.europa.eu/api/mqa/shacl/), and ensuring that the server correctly responded to OAI-PMH requests from potential harvesters, including the European Data Portal itself.

4. SHACL validation results

After implementing the conversion process and the OAI-PMH server to provide DCAT-AP compliant metadata, the next important step was to verify the compliance of the generated metadata. This verification process was essential to ensure that the SEBASTIEN datasets' metadata met the standards required by the European Data Portal.

To validate the DCAT-AP compliance of our metadata, we utilized the European Data Portal's "DCAT-AP SHACL validation service API". SHACL (Shapes Constraint Language) is a language for validating RDF graphs against a set of conditions. It's particularly well-suited for validating DCAT-AP compliance, as it can check both the structure and content of the metadata against the DCAT-AP specification.

The validation process involved the following steps:

1. Preparation of the metadata: we ensured that our OAI-PMH server was correctly serving the DCAT-AP compliant metadata in RDF/XML format for each dataset.
2. Setting up the validation requests: we used Insomnia (https://insomnia.rest/), a popular API testing tool, to prepare and send requests to the European Data Portal's validation service. This allowed us to easily manage and repeat the validation process for multiple datasets.
3. Sending requests to the validation endpoint: The European Data Portal provides a specific endpoint for DCAT-AP SHACL validation: https://data.europa.eu/api/mqa/shacl/validation/report
4. Analysing the validation results: For each dataset tested, we sent a validation request and received a response from the API.

The validation process returned good results. For all the SEBASTIEN datasets tested, the European Data Portal's SHACL validation service returned a positive result, indicating that our metadata was indeed DCAT-AP compliant.

The validation service responds with a concise JSON object, which includes a boolean flag indicating whether the metadata is valid. In our case, this flag was consistently true across all tested datasets, confirming the success of our implementation.

Here's an example of a typical response received from the validation service:


```
{
	"@id": "_:b0",
	"shacl:conforms": {
		"@value": "true",
		"@type": "http://www.w3.org/2001/XMLSchema#boolean"
	},
	"@type": "shacl:ValidationReport",
	"@context": {
		"shacl": "http://www.w3.org/ns/shacl#"
	}
}
```


This successful validation across multiple datasets demonstrates the effectiveness of our approach in converting the original SEBASTIEN metadata to DCAT-AP compliant format. It confirms that:

1. Our metadata conversion process accurately transforms the original JSON metadata into DCAT-AP compliant RDF/XML.
2. The implemented OAI-PMH server correctly serves the metadata in a format that meets the European Data Portal's requirements.
3. All mandatory fields as per the DCAT-AP specification are correctly included and formatted in our metadata.

These positive results are a crucial milestone in the process of integrating SEBASTIEN datasets with the European Data Portal. They ensure that our metadata will be correctly interpreted and utilized when harvested by the portal, enhancing the discoverability and usability of SEBASTIEN data within the broader European data ecosystem.

The successful SHACL validation also provides confidence in the robustness of our implementation, indicating that it can handle various datasets consistently while maintaining DCAT-AP compliance.

5. DCAT-AP IT conversion

To reach the requested milestone, it is requested that relevant datasets (including metadata) resulting from the Action are published on a national portal or catalog that is harvested by the European Data Portal. Specifically, the Italian open data portal was chosen because the datasets

produced are focused on Italy, as the stakeholders identified and the potential audience of users of the portal.

The implementation for the Italian data portal, AGID, differed from the European Data Portal in a few key aspects. Unlike the European Data Portal, AGID doesn't require the use of OAI-PMH for providing metadata repositories. This allowed for a simpler approach in serving the converted RDF file.

To accommodate AGID's requirements, the existing FastAPI implementation was extended with a regular HTTP endpoint. This endpoint serves the converted RDF file directly, without the need for the OAI-PMH protocol, the resulting RDF can be seen in this URL: https://sebastien-datalake.cmcc.it/api/v2/dcatapit. The simplicity of this approach made it straightforward to integrate with the existing SEBASTIEN API infrastructure.

For the conversion to the Italian specification, known as DCAT-AP IT, the official documentation provided by AGID was followed. This documentation, available at https://www.dati.gov.it/sites/default/files/2020-02/linee-guida-cataloghi-dati-profilo-dcat-ap-it-2.pdf, outlines the specific requirements for metadata in the Italian context.

The DCAT-AP IT specification has some notable differences compared to the standard DCAT-AP used by the European Data Portal. It includes more mandatory fields and features a slightly different structure. These differences can be seen in Table 4, which provides a list of the mandatory fields.

Class	Field Name
dcatapit:Catalog	dct:title
dcatapit:Catalog	dct:description
dcatapit:Catalog	dct:publisher
dcatapit:Catalog	dct:modified
dcatapit:Catalog	dcat:dataset
dcatapit:Dataset	dct:identifier
dcatapit:Dataset	dct:title
dcatapit:Dataset	dct:description
dcatapit:Dataset	dct:modified
dcatapit:Dataset	dcat:theme

dcatapit:Dataset	dct:rightsholder
dcatapit:Dataset	dct:accrualPeriodicity
dcatapit:Dataset	dcat:distribution
dcatapit:Distribution	dct:format
dcatapit:Distribution	dcat:accessURL
dcatapit:Distribution	dct:license

Table 4: Mandatory classes and fields in the DCAT-AP IT specification

The conversion process for DCAT-AP IT built upon the existing Python codebase used for the DCAT-AP conversion. A new function was implemented to take the graph generated for DCAT-AP and rework it into the DCAT-AP IT format. This approach allowed for efficient reuse of code while accommodating the specific requirements of the Italian specification, the function can be seen below.

```
# Function to convert to DCAT-AP IT format
def convert_to_dcat_ap_it(graph, catalog_uri):
    g = graph

    # Bind DCATAPIT namespace to graph
    g.bind("dcatapit", DCATAPIT)

    # Create catalog and add it to the graph
    catalog = URIRef(catalog_uri)
    g.add((catalog, RDF.type, DCATAPIT.Catalog))
    g.add((catalog, DCTERMS.title, Literal("Sebastien Catalog")))
    g.add((catalog, DCTERMS.description, Literal("A catalog of Sebastien datasets")))
    g.add((catalog, DCTERMS.publisher, URIRef("https://www.cmcc.it/")))
    g.add((catalog, DCTERMS.modified, Literal(datetime.now().strftime("%Y-%m-%d"),
datatype=DCTERMS.W3CDTF)))

    # Find all datasets in graph
    for dataset_uri in g.subjects(RDF.type, DCAT.Dataset):
        # Create dcatapit:Dataset node
        dcatapit_dataset_node = BNode()
        g.add((dcatapit_dataset_node, RDF.type, DCATAPIT.Dataset))

        # Wrap existing dataset elements under dcatapit:Dataset
        for s, p, o in g.triples((dataset_uri, None, None)):
            if p != RDF.type:
                g.remove((dataset_uri, p, o))
                g.add((dcatapit_dataset_node, p, o))

        # Remove original dcat:Dataset node
```

```
        g.remove((dataset_uri, RDF.type, DCAT.Dataset))

        # Add new dcat:dataset relation to the catalog, pointing to the
dcatapit:Dataset
        g.add((catalog, DCAT.dataset, dcatapit_dataset_node))

        # Add mandatory fields with placeholder values
        g.add((dcatapit_dataset_node, DCAT.theme,
URIRef("http://publications.europa.eu/resource/authority/data-theme/AGRI")))
        g.add((dcatapit_dataset_node, DCTERMS.rightsHolder,
URIRef("https://www.cmcc.it/")))
        # Add accrualPeriodicity based on dataset name
        dataset_name = dataset_uri.split("/")[-1]
        if dataset_name in ACCRUAL_PERIODICITY:
            g.add((dcatapit_dataset_node, DCTERMS.accrualPeriodicity,
URIRef(f"http://publications.europa.eu/resource/authority/frequency/{ACCRUAL_PERIODIC
ITY[dataset_name]}")))
        else:
            g.add((dcatapit_dataset_node, DCTERMS.accrualPeriodicity,
URIRef("http://publications.europa.eu/resource/authority/frequency/UNKNOWN")))

        # Change Distribution namespace to DCATAPIT
        for s, p, o in g.triples((dcatapit_dataset_node, DCAT.distribution, None)):
            g.remove((s, p, o))
            g.add((s, DCATAPIT.distribution, o))
            g.add((o, RDF.type, DCATAPIT.Distribution))

    return g
```

One key difference in the implementation for AGID is that the metadata is provided as a comprehensive list containing information for all datasets, rather than individual metadata files for each dataset. This approach aligns with AGID's harvesting preferences and allows for more efficient bulk processing of the SEBASTIEN metadata.

The implementation of the DCAT-AP IT conversion and the new HTTP endpoint ensures that the SEBASTIEN datasets can be effectively harvested and integrated into the Italian national data portal. This approach allows for the metadata to be provided as a comprehensive list containing information for all datasets, which aligns with AGID's harvesting preferences. By extending the existing codebase and adapting it to the DCAT-AP IT requirements, we were able to efficiently reuse code while meeting the specific needs of the Italian specification. This implementation expands the visibility and accessibility of the SEBASTIEN data within the Italian open data ecosystem, furthering the project's goal of wide data dissemination.

6. Conclusions

In the context of the SEBASTIEN project, a series of datasets, indicators and indices were produced in order to contribute to making the livestock sector more environmentally and socio-economically sustainable.

The datasets produced and exposed (see also Milestone M7 - Report on released database of sectoral indicators/indices) were exposed by the developed datalake to ensure easy use by users and stakeholders through the SEBASTIEN web service portal (https://dds.sebastien-project.eu/app/catalog). In addition, these datasets were i) made compliant with the specifications of the European data portal to ensure MQA compliance and ii) exposed through the necessary APIs to ensure compliance with the DCAT-AP IT specifications to facilitate onboarding on the Italian AGID Open Data Portal.

References

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., ... & Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, *3*(1), 1-9.