Open Data Management in Agriculture and Nutrition

This e-learning course is the result of a collaboration between ***GODAN Action*** *partners, including* ***Wageningen Environmental Research*** *(WUR),* ***AgroKnow****,* ***AidData****,* ***the Food and Agriculture Organization of the United Nations*** *(FAO),* ***the Global Forum on Agricultural Research*** *(GFAR), and* ***the Institute of Development Studies*** *(IDS),* ***the Land Portal****,* ***the Open Data Institute*** *(ODI) and* ***the Technical Centre for Agriculture and Rural Cooperation*** *(CTA).*

GODAN ACTION
Global Open Data for Agriculture & Nutrition

UKaid
from the British people

GODAN Action is a three-year project UK's Department for International Development to enable data users, producers and intermediaries to engage effectively with open data and maximise its potential for impact in the agriculture and nutrition sectors. In particular we work to strengthen capacity, to promote common standards and best practice and to improve how we measure impact. [www.godan.info]

UNIT 4: SHARING OPEN DATA

LESSON 4.3: STRUCTURAL AND ARCHITECTURAL INTEROPERABILITY

Photo by Mario Antonio Pena Zapatería licensed under CC BY SA 2.0

Aims and learning outcomes

This lesson aims to;

- explain the basics of structural interoperability:data formats and data structures
- explain the basics of architectural interoperability: protocols, technical frameworks.
- provide guidance on the advantages and disadvantages of specific solutions.

After studying this lesson, you should be able to;

- understand the basics of structural interoperability and current best practices
- assess formats and protocols that better fit their needs
- (guide developers to) adopt the most interoperable solutions.

Contents

List of figures

1. Structural interoperability

We have said that this is the level where (meta)data become machine-readable. However, 'machine-readable' in its most literal sense is not enough for structural interoperability: in order for machines to understand the structure of the (meta)data, i.e. which are the labels/metadata elements/column names/variables and which are the values, and whether there's a hierarchy, the (meta)data have to be structured enough for a machine to extract individual values.

In other words, (meta)data have to be not just readable but 'parsable' by machines. Since 'parsing' means 'splitting a file or other input into pieces of data that can be easily stored or manipulated', the more regular and rigorous a format is, the easier it is to parse it. Ideally, parsable formats have a published specification so that developers know how the structure is built and can write parsers accordingly. So this level of interoperability is achieved through the choice of the most ease to parse data format. The next chapters will illustrate the most common of these formats.

At this level of interoperability, machines can split the file and identify the values and their roles, not understand their meaning. However, while *per se* the data structures we shall describe do not convey any meaning and do not allow for meaningful processing, they can be made semantic and therefore understandable by applying the technologies described in lesson 4.4 on semantic interoperability.

1.1. Serialisation of (meta)data: data structures into file formats

A data structure (hierarchical, relational, tabular…) can be represented in a file in many ways. The way data are physically stored in a file is called the 'serialisation' of the data: 'In computer science, in the context of data storage, serialisation is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer)'[1]

There are many file formats that are parsable by machines strictly speaking, any file format can be parsed by some software, otherwise it would be useless. What makes a file format more easily parsable and therefore more interoperable is:

- The simplicity of the structure: the fewer the elements of the structure and the fewer the possible constructs are, the easier it is for a machine to parse the file without too complex reasoning. (On the other hand, the format should also cater for complex data structures: the most suitable

[1] From Wikipedia: https://en.wikipedia.org/wiki/Serialization

file format is the one that combines simplicity and enough flexibility to represent complex data structures.)
- The rigorousness and 'regularity' of the structure: the fewer the options to serialise the same thing in a different way, the easier it is for a machine to parse the file and identify the role of each element.
- The existence of a clear and open specification for the format, which makes it easy for any developer to write parsers. (Many formats are tied to a software product and can be correctly and fully parsed only by that product. This makes their interoperability level very low.)
- An additional help is the existence of software libraries and APIs that can parse the format.

The formats that are considered the most interoperable against the criteria above are CSV, XML and JSON.

Binary array-based formats like NetCDF and HDF5 retain a special place for their use by researchers. We will not look into them here for a few reasons: they are more tied to specific software libraries (however many tools can read them); they are still more oriented towards compactness and efficiency of data transmission than towards broader interoperability, especially semantic interoperability; and some work has already been done to represent NetCDF in CSV (the CEDA BADC-CSV[2] format) and in XML (the UNIDATA NcML[3]).

CSV (comma-separated values) files have a simple tabular format with a header record with column names, commas to mark the field boundaries and line feeds to mark the record boundaries.

CSV is the simplest data format and very much used for typical tabular data like statistics or observations, but it cannot represent complex structures and is not documented in a machine-readable way: 'There is no mechanism within CSV to indicate the type of data in a particular column, or whether values in a particular column must be unique. It is therefore hard to validate and prone to errors such as missing values or differing data types within a column.'[4] Some work to improve the interoperability of CSV files has been done by the W3C Working Group on 'CSV on the Web: Use Cases and Requirements'[5].

XML (eXtensible Markup Language[6]) is 'a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML emphasise simplicity, generality, and usability across the Internet. […] Although the design of XML focuses on

[2] http://help.ceda.ac.uk/article/105-badc-csv

[3] http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/ncml/#NcML22

[4] W3C. CSV on the Web: A Primer. https://www.w3.org/TR/tabular-data-primer/

[5] https://www.w3.org/TR/csvw-ucr/

[6] https://www.w3.org/TR/REC-xml/

documents, the language is widely used for the representation of arbitrary data structures…'[7] (see Figure 1).

The most interesting aspect of XML in terms of interoperability is the support of the definition of 'schemas' to which a specific XML document can declare to conform. Schemas are machine-readable documents that specify how to name and organise the elements in an XML document and can define hierarchies, constraints etc. We will see in lesson 4.4 how schemas make it possible to embed semantic vocabularies in XML documents.

```
<dataset>
  <metadata>
    <dataset-metadata1>ValueA</dataset-metadata1>
    <dataset-metadata2>ValueB</dataset-metadata2>
   ...
  </metadata>
  <records>
    <record>
      <variable1>Value1</variable1>
      <variable2>Value2</variable2>
      <variable3>Value3</variable3>
      ...
    </record>
    <record>
      <variable1>Value4</variable1>
      <variable2>Value5</variable2>
      <variable3>Value6</variable3>
      ...
    </record>
    ...
  </records>
</dataset>
```

Figure 1 Example of a generic XML structure for a dataset

Dataset metadata in XML can reach a high level of complexity, as in this example from the US Geological Service: https://sofia.usgs.gov/metadata/sflwww/SOFIA_Cape_Sable.xml, where besides extensive metadata on the dataset (creators, conditions, citations), the <attr> elements describe all the variables used in the dataset (the dataset metadata file in this case is separated from the data file).

JSON (JavaScript Object Notation) is 'an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serialisable value)'[8].

JSON has a good combination of simplicity and flexibility in terms of data structures (it was born as a format to represent programming data objects of any kind) and it now has support for the definition of schemas for validation purposes[9]. Given the ease of use of JSON structures in programming languages (the need for parsing is minimal and the structure reflects object-oriented

[7] From Wikipedia: https://en.wikipedia.org/wiki/XML

[8] From Wikipedia: https://en.wikipedia.org/wiki/JSON

[9] http://JSON-schema.org/

programming practices), JSON is the serialisation format preferred by programmers.

```
{"dataset": {
   "dataset-metadata1": "ValueA",
   "dataset-metadata2": "ValueB",
   "records": {
     "record": [
     {"variable1": "Value1", "Variable2": "Value2", "Variable3": "Value3"},
     "record": [
     {"variable1": "Value4", "Variable2": "Value5", "Variable3": "Value6"},
     ...
     ]
   }
}}
```

Figure 2 Example of JSON structure

CSV, XML and JSON are the most interoperable formats also in view of what we will see about the application of semantic technologies: they are the formats to which these technologies are most easily applied (see lesson 4.4).

Of the three, XML is traditionally the one considered as the best combination of (a) simplicity; (b) flexibility for complex data structures; and (c) support of schema definitions that set constraints and make them easier to understand. However, considering that JSON also supports schemas and that the JSON-LD (JSON for Linking Data[10]) specification provides a method of encoding Linked Data, JSON can be considered as suitable as XML.

Other formats that are extremely well interoperable are the main native RDF formats, Turtle and N-Triples. Since they are the native serialisation of the RDF grammar, and the RDF grammar deserves a separate treatment, we will describe them in the next section.

1.2. Beyond file formats: RDF, a rigorous 'grammar' behind the format

The Resource Description Framework (RDF) is 'a general method for **conceptual description or modeling of information** that is implemented in web resources, using a variety of syntax notations and data serialisation formats'[11]. As such, it is not a format and not tied to a specific format: any format that can represent the basic RDF 'grammar' can implement RDF.

The RDF grammar is based on statements made of **subject – predicate – object**; each statement is a 'triple' and the assumption is that combinations of such triples can represent and describe everything. The 'glue' of such triples are the Unique Resource Identifiers (URIs) that always refer univocally to the

[10] 'JSON for Linking Data'. https://json-ld.org/

[11] From Wikipedia: https://en.wikipedia.org/wiki/Resource_Description_Framework

same entity, thus allowing complex description structures to be split into simpler triples where the same resource is either the subject or the object of the statement and meaningful predicates link the subject to the object.

The triples syntax is very simple:

Resource A URI – has title – 'War and Peace'
Resource A URI – has author – Person A URI
Person A URI – has name – 'Lev Tolstoj'

The triple statements are normally represented as graphs with nodes and arcs, where subject and object are nodes, while predicates are arcs (also called edges: the arrows, the relationships). Each combination of node-arc-node is a distinct triple. All components of the triple are ideally identified by URIs, even predicates/arcs (see how these URIs make it possible to embed semantic vocabularies in RDF statements in lesson 4.4. In the figure below, the predicate/arc is the URI of the property as defined in an existing metadata vocabulary which was in turn formalised as RDF, which means all classes and properties have URIs).

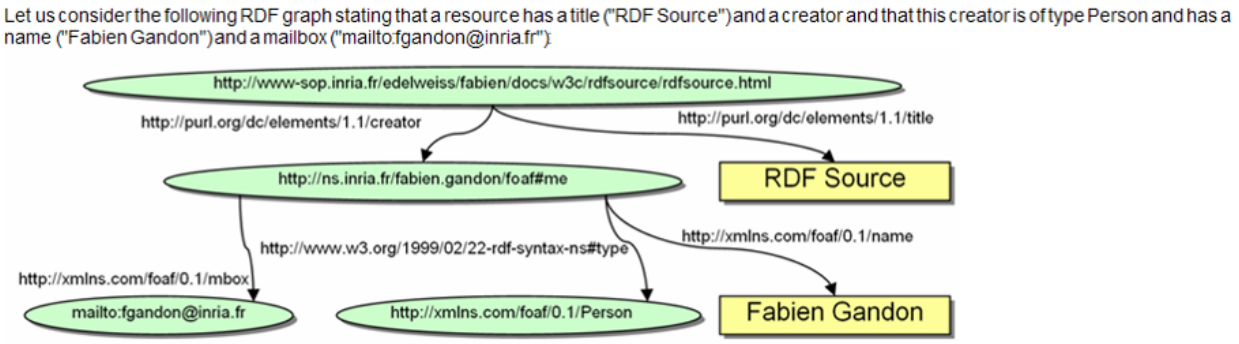

Figure 3 RDF represented as a graph, from W3C[12]

As we said, any format that allows for the triple construct can implement RDF. However, new native formats, created solely to serialise triples, with a rigorous syntax that allows only for the triples construct, are Turtle[13] and N-Triples[14].

```
<http://www-sop.inria.fr/edelweiss/fabien/docs/w3c/rdfsource/rdfsource.html> dc:title   "RDF Source"
<http://www-sop.inria.fr/edelweiss/fabien/docs/w3c/rdfsource/rdfsource.html> dc:creator <http://ns.inria.fr/fabien.gandon/foaf#me>
<http://ns.inria.fr/fabien.gandon/foaf#me>    rdf:type   foaf:Person
<http://ns.inria.fr/fabien.gandon/foaf#me>    foaf:name  "Fabien Gandon"
<http://ns.inria.fr/fabien.gandon/foaf#me>    foaf:mbox  <mailto:fgandon@inria.fr>f:mbox rdf:resource="mailto:fgandon@inria.fr"/>
```

Figure 4 Example of RDF serialisation in N-Triples of the graph above, from same W3C page

Beyond this, the RDF grammar has been successfully applied to XML: XML/RDF is not really a new 'format' (it is still formally XML), but rather an XML that uses

[12] https://www.w3.org/Submission/rdfsource/
[13] https://en.wikipedia.org/wiki/Turtle_(syntax)
[14] https://www.w3.org/TR/n-triples

specific constraints that enforce the triple logic. These 'constraints' are defined in the 'RDF 1.1 XML Syntax' W3C specification[15].

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 <rdf:Description rdf:about="http://www-sop.inria.fr/edelweiss/fabien/docs/w3c/rdfsource/rdfsource.html">
  <dc:title>RDF Source</dc:title>
  <dc:creator>
    <foaf:Person rdf:about="http://ns.inria.fr/fabien.gandon/foaf#me">
     <foaf:name>Fabien Gandon</foaf:name>
     <foaf:mbox rdf:resource="mailto:fgandon@inria.fr"/>
   </foaf:Person>
  </dc:creator>
 </rdf:Description>
</rdf:RDF>
```

Figure 5 Example of RDF serialisation in XML (XMLRDF) of the graph above, from the same page

A quick comparison between XML files following a non-RDF schema and RDF/XML files shows that using the RDF grammar makes XML files simpler and more rigorous. The technical explanation for this would require too much time, but suffice it to say that some of the possible sources of ambiguity in parsing XML (nested elements can represent sub-properties or linked entities, attributes can represent properties or meta-metadata) is overcome through the so-called 'striped syntax' (elements ***must*** alternately represent nodes and edges) and there is much less ambiguity in the use of attributes and nested elements for the same function.

The simplicity of the grammar, the rigorousness of the constraints and the simultaneous flexibility of mechanism of the triple statements that can be combined to create complex data structures make RDF probably the best combination of interoperability and flexibility against the criteria listed in section 1.

2. Architectural interoperability

Architectural interoperability is closer to what HIMSS defines as 'foundational' interoperability, but it is not related to the basic general transmission protocols (TCP/IP, HTTP, FTP) as much as to higher-level data exchange protocols designed for (meta)data sharing.

Besides data formats and shared semantics, data sharing at a more advanced level can also entail compliance with some architectural requirements, especially if it is planned to participate in initiatives and partnerships that adopt a specific architecture. In such cases, besides being made available as a simple file that can be parsed directly, (meta)data are often expected to be served through services.

[15] https://www.w3.org/TR/rdf-syntax-grammar/

The advantage of exposing (meta)data through services are: (a) (meta)data can be queried and selected subsets can be retrieved; (b) additional metadata about a linked resource can be retrieved; and (c) (meta)data can be paged.

The generic name for this trend of exposing data through services is 'Data-as-a-Service' (Daas), which goes from the simple exposure of data by a data provider to big architectures of data providers and consumers.

Here we will limit ourselves to what it means to expose data through a service and we will use the examples of the two most used protocols for doing this, but it is important to be aware of the fact that there are many ways of implementing Daas. In a similar way to what we said about data formats and semantics, the more used a protocol is (in general or in the community where you want to share your data), the more interoperable your data will become if you implement it.

2.1. Most used protocols for data sharing

The most popular protocols for exposing data as a service are the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and SPARQL, but more standardised RESTful[16] APIs are also being more and more used. OAI-PMH and SPARQL are technically stateless APIs and like all APIs they expose 'methods' which accept a number of parameters and these methods can be called via an HTTP request and return a machine-readable response.

OAI-PMH was born in the library environment and it was conceived mainly for metadata, but it can be used to expose any type of 'records'. As long as the response format is XML and the container XML follows the OAIPMH schema, the <metadata> element under each <record> can contain XML using any schema, even XML/RDF, so it can be suitable for instance for exposing metadata records of datasets using a dataset metadata vocabulary (like the W3C Data Catalog Vocabulary, DCAT).

The methods of the OAI-PMH API are designed to enable a series of calls that allow the caller (an application) to preliminarily check some metadata about the repository (the name, what metadata schemas are supported, which subsets can be retrieved…) and then retrieve records filtered according to metadata format, date ranges, subsets. Records can be retrieved in smaller batches using a 'resumption token'.

[16] 'Representational state transfer (REST) or RESTful web services is a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.' From Wikipedia: https://en.wikipedia.org/wiki/Representational_state_transfer

Figure 6 The OAI-PMH workflow[17]

Although (meta)data exposed through OAI-PMH can be made highly interoperable including semantics and even using RDF and URIs, the protocol *per se* has a couple of limitations: (a) the ability to query the data is limited to a few accepted parameters; and (b) records in the <ListRecords> section are in general expected to all be of the same type and have the same structure (though technically this is not enforced), so different calls for different types of entities have to be made.

SPARQL (SPARQL Query Language for RDF) is a language for querying RDF data, but it is also the name of the protocol that enables the query to be sent via an HTTP request and get the response in several RDF-enabled formats (RDF/XML, Turtle, JSON…).

Compared to OAI-PMH, SPARQL allows for much more complex queries built using the powerful triple grammar and can expose triples with any type of subject (a dataset, an organisation, a place, a topic…) all from the same interface. This makes it possible to filter data very granularly and look up other properties (e.g. labels) of resources referenced by URIs. SPARQL clients can also send the same query to several SPARQL engines and combine the responses: the use of URIs allows the client to merge duplicates and consolidate properties for the same entity coming from different systems.

SPARQL as a protocol is usable *per se* but it also one of the components of the broader Linked Data architecture.

The term **Linked Data** 'refers to a set of best practices for publishing and interlinking structured data for access by both humans and machines via the

[17] https://www.oaforum.org/tutorial/index.php

use of the RDF (Resource Description Framework) family of standards for data interchange and SPARQL for query.'[18]

These best practices are meant to help implement the Linked Data principles proposed by Tim Berners-Lee:

1. Use URIs to name things;
2. Use HTTP URIs so that things can be referred to and looked up ('dereferenced') by people and user agents;
3. When someone looks up a URI, provide useful information, using the open Web standards such as RDF, SPARQL;
4. Include links to other related things using their URIs when publishing on the Web.

While exposing (meta)data as RDF using URIs to identify things and providing a SPARQL interface will satisfy most of the Linked Data principles, implementing the principle at point 2 above requires an additional step. Linked Data prescribes that URIs are used to 'name' things but also to look up things: best practices therefore recommend using URLs for URIs, so that a URI is always 'dereferenced' to a URL address that can be 'looked up' through HTTP. In addition, however, it also prescribes that the URL should serve a response for both 'people and user agents', which means that depending on the request (either from a web browser for human viewers or from an RDF client as a user agent) the URL should serve an HTML response or an RDF response (also responding with the 303 redirect code while redirecting to the other resource). This is normally achieved through a complex technique called 'content negotiation', which exploits the 'content-type' header metadata in HTTP requests to trigger different responses[19].

These protocols and architectures are very generic and one (OAI-PMH) was originally created in and for the library community (it is now very much used for cultural heritage in general) and the other (SPARQL) arose in the more IT-oriented RDF community and around the ideas of Tim Berners-Lee.

Other protocols have been developed in scientific communities, building on common scientific data exchange practices, focusing on efficiency of data transfer (catering for large amounts of data) and leveraging traditional exchange formats like NetCDF and HDF5. An example is OPeNDAP.

OPeNDAP ('Open-source Project for a Network Data Access Protocol') is 'a data transport architecture and protocol widely used by earth scientists. The protocol is based on HTTP and […] includes standards for encapsulating structured data, annotating the data with attributes and adding semantics that describe the data. […] An OPeNDAP client sends requests to an OPeNDAP server, and receives various types of documents or binary data as a response.

[18] W3C. Best Practices for Publishing Linked Data. https://www.w3.org/TR/ld-bp/

[19] http://linkeddata.org/conneg-303-redirect-code-samples

[…] Data on the server is often in HDF or NetCDF format, but can be in any format including a user-defined format. Compared to ordinary file transfer protocols (e.g. FTP), a major advantage using OPeNDAP is the ability to retrieve subsets of files, and also the ability to aggregate data from several files in one transfer operation.'[20]

The choice of protocols to implement to share one's data should be definitely influenced primarily by the community with which data are expected to be shared and then by technical considerations such as ease of implementation, support for specific formats and general support for the data sharing principles outlined in lesson 4.1.

For example, OPeNDAP is widely used by governmental agencies such as NASA and NOAA to serve satellite, weather and other observed earth science data, so it could be a good choice for institutions sharing these or similar types of data. On the other hand, if wider sharing across communities is desired, OAI-PMH could be the simplest choice, or SPARQL and a Linked Data-enabled environment if being part of the Semantic Web and the Internet of Things is a priority.

As a conclusion to this excursus on different types of data interoperability, it is important to note that implementing all the technical requirements can be very difficult: unless data are manually curated, manually converted to interoperable formats and manually annotated with semantics (which can only happen for a very small repository), the exposure of (meta)data is done through a software tool. In most cases, rather than developing *ad hoc* software, it may be very convenient to use existing tools. Since there may be no tool that implements all the requirements (especially in terms of semantic interoperability), it is important to evaluate existing tools against all the criteria explained in this lesson and to prioritise criteria depending on the specific data sharing needs. See lesson 3.3 for more information on data repository tools.

[20] From Wikipedia: https://en.wikipedia.org/wiki/OPeNDAP