PRACTICE AND IMPLEMENTATION PAPER

The FAIR Data Point: Interfaces and Tooling

Oussama Mohammed Benhamed[1], Kees Burger[2], Rajaram Kaliyaperumal[2], Luiz Olavo Bonino da Silva Santos[2,3], Marek Suchánek[4], Jan Slifka[4], Mark D Wilkinson[1†]

[1]Departamento de Biotecnología-Biología Vegetal, Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas, Centro de Biotecnología y Genómica de Plantas UPM-INIA, Universidad Politécnica de Madrid (UPM) - Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA), Madrid, ES 28223, Spain

[2]Leiden University Medical Center, Leiden, 2333 ZC, the Netherlands

[3]University of Twente, Enschede, 7513 GB, the Netherlands

[4]Czech Technical University in Prague, Prague, CZ 15000, Czech Republic

Keywords: FAIR Data; Linked data; Semantic Web; Metadata; User interfaces; Tooling; Semantic query

Citation: Mohammed Benhamed O., Burger K., Kaliyaperumal R., et al.: The FAIR Data Point: Interfaces and Tooling. Data Intelligence 5(1), 184-201 (2023). doi: 10.1162/dint_a_00161

Received: February 28, 2022; Revised: May 7, 2022; Accepted: June 8, 2022

ABSTRACT

While the FAIR Principles do not specify a technical solution for 'FAIRness', it was clear from the outset of the FAIR initiative that it would be useful to have commodity software and tooling that would simplify the creation of FAIR-compliant resources. The FAIR Data Point is a metadata repository that follows the DCAT(2) schema, and utilizes the Linked Data Platform to manage the hierarchical metadata layers as LDP Containers. There has been a recent flurry of development activity around the FAIR Data Point that has significantly improved its power and ease-of-use. Here we describe five specific tools—an installer, a loader, two Web-based interfaces, and an indexer—aimed at maximizing the uptake and utility of the FAIR Data Point.

1. INTRODUCTION

Most general-purpose repositories such as Zenodo, Figshare, and Dataverse have focused significant efforts on ensuring they have well-designed and rich metadata surrounding every data deposit, facilitating their discovery and reuse. Nevertheless, there remains a "long tail" of independent scholarly data publishers, including boutique databases where the data is more useful being served by an API or is too large for a deposit, or registries, germplasm banks, and biobanks where the data content may be highly sensitive, and

† Corresponding author: Mark D Wilkinson (Email: mark.wilkinson@upm.es; ORCID: 0000-0001-6960-357X).

therefore not able to be publicly deposited. Nevertheless, to be considered FAIR resources, these must identify a mechanism for creating and publishing the layers of metadata that would enable a machine to discover the existence of these data and determine the mechanism by which access could be achieved or requested. Five years ago, an early prototype of a general-purpose metadata repository was designed and published, with the authors being strictly guided by the FAIR Principles such that the repository would facilitate adherence to all facets of FAIRness. This became known as the FAIR Data Point, and could now be considered "commodity", production-quality, ready-to-use software. While the architecture of the contemporary FAIR Data Point will be described in a separate work, here we focus on the tooling that has evolved around this core codebase that enhances and simplifies its use.

2. BACKGROUND

2.1 Linked Open Data, the Semantic Web, and FAIR

Tim Berners-Lee coined the vision of the Semantic Web: *"an extension of the current Web, where machines can talk to each other, while understanding and manipulating data without human interaction* [1]*"*. Implementation of the Semantic Web relies on two core technologies: Resource Description Framework (RDF) [2] and Ontologies, generally represented in Web Ontology Language (OWL) [3]. Datasets following RDF patterns are referred to as Linked Data, and when these data are openly published on the Web, following the principles of Open Data—that data should be openly available for reuse by any third-party—they become part of the Linked Open Data (LOD) "cloud" [4, 5]. In 2014, a group of stakeholders with an interest in scholarly data publication came together for a Lorenz Workshop in Leiden, The Netherlands, to discuss approaches to data publication that would render the data Findable, Accessible, Interoperable, and Reusable (FAIR), with particular emphasis on FAIR for machines. Two years later the ideas from that meeting were formalized into the FAIR Data Principles [6]. Beyond the Linked Open Data and Semantic Web principles—which are largely technical/implementation principles—FAIR focused on "data behaviors": What properties should data have that will enable machines to discover, interact-with, interpret, and correctly reuse those data? Thus, the primary focus of FAIR was on guidance towards provision of rich and extensive metadata that could lead a machine to more accurately find and evaluate the utility of LOD (and non-LOD) on the Web. To enhance the ability of a machine to interpret these metadata records, it is necessary to use ontologies and other widely accepted semantic and syntactic standards within the metadata to describe not only the nature of the data, but also the context within which it was generated, and how to properly cite or acknowledge the producer/publisher when reusing it.

2.2 History of the FAIR Data Point

Almost immediately after the Lorenz Workshop, a subgroup of 22 attendees, primarily representing the software development and scholarly data publishing communities spanning the EU and the USA, created a working group—the "Skunkworks" team [7]—that aimed to identify ways to operationalize the FAIR Principles. This involved a series of teleconferences where a general skeleton of a metadata architecture emerged that could represent the multi-layered repositories that are common, particularly in the Life

Sciences, in a manner such that a computational agent could identify a repository that was potentially appropriate for its task, and then "drill-down" to find the precise data record it required. Conceptually, the consensus design took the form of layers of largely identical (structurally) metadata records, with a reliable path for an agent to follow from one layer down to the next (and back)—with Matryoshka dolls being an analogy for the design. We then explored existing and emergent technologies and standards that could fulfill these two requirements. The Data Catalog Vocabulary (DCAT) specification became a World Wide Web Consortium (W3C) standard in 2014 [8]. DCAT was designed to support interoperability between data catalogs published on the Web by providing an RDF specification broken into three main metadata layers—Catalog, Dataset, and Distribution—where these three kinds of digital records had largely overlapping (though not identical) metadata. Though we recognized early-on that DCAT did not have sufficient layers for many of our use-cases, we selected this as the initial target standard for our metadata representation platform. Knowing that we would eventually need to extend the number of layers, the problem of providing a predictable path between those layers was outstanding. In 2015, the Linked Data Platform (LDP) became a W3C recommendation [8]. LDP had three main features relevant to our problem: First, it had the concept of a "container of containers", similar to DCAT, but more akin to Matryoshka dolls in that the recursion was unlimited. Second, it had a specific property that linked a container to its contents ("ldp:contains"), but more importantly, could impose that same behavior on other properties. For example, the property that links a DCAT Catalog to its Datasets (dcat:dataset), and a DCAT Dataset to its Distributions (dcat:distribution), could both be defined as having the same interpretation, from the perspective of an automated agent, as "ldp:contains". Thus, we could use LDP to design a predictable path up-and-down metadata layers that could be followed by a Web crawler. Third, LDP is a read/write interface, giving us a simple REST interface for creating and interconnecting new metadata layers.

The first prototype of the combination of these two technologies was published in 2017 [10]. In parallel, production-ready software with these same behaviours was being written, and this was named the FAIR Data Point [11], with the objective of making the creation and publication of FAIR metadata as simple as possible by pre-selecting a set of FAIR metadata models, data representation formats, and Web behaviours. The FAIR Data Point publication just cited was the report of an early version of the software that had limited functionality. In the subsequent five years, the software has improved dramatically in both its functionality, and its ease-of-use. In the following section we describe, in detail, the contemporary functionalities and accessory tooling around modern FAIR Data Points, how these have been implemented, and their availability.

3. THE FAIR DATA POINT TOOLS AND INTERFACES

An overview of all tools and interfaces described in this section, and how they relate to one another, is provided in Figure 1. They will be covered in more detail in the following sections.

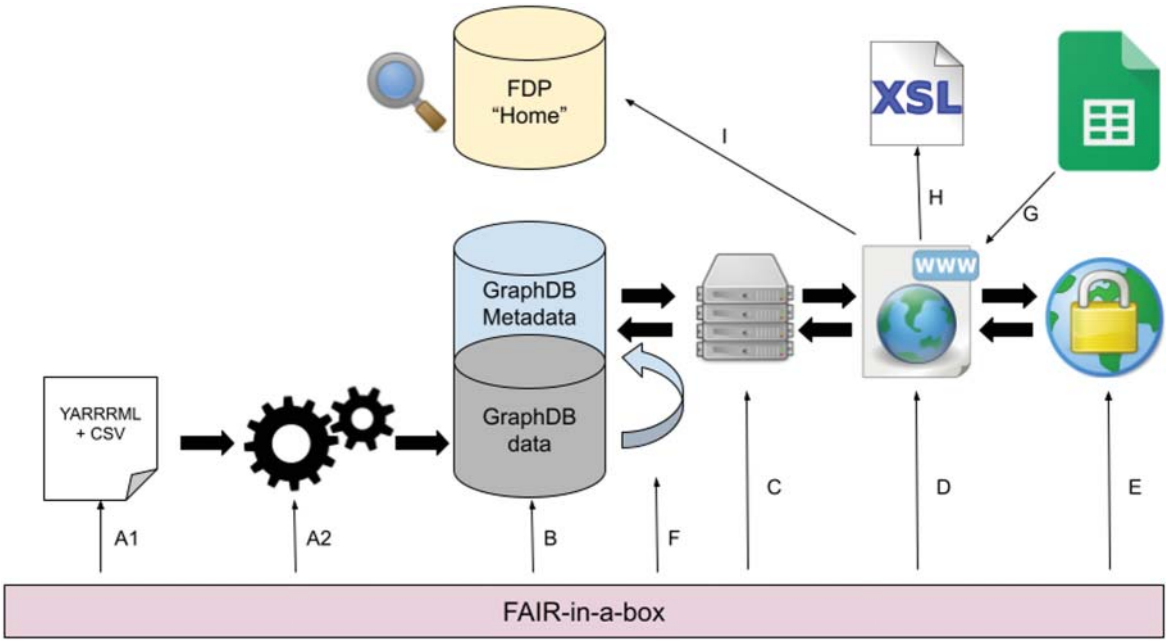

Figure 1. Diagram of Key Components and Tools. **FAIR-in-a Box: A.** Template based transformation pipeline based on YARRRML/CSV (A1) and SDM-RDFizer (A2). **B**. Two GraphDB databases are bootstrapped—one to contain the FDP Metadata, and a second to contain data output from the transformation pipeline (A1/2). Additional components of the FDP required for back-end functionality are also installed (not shown). **C.** FDP Server serves metadata to/from the default FDP Client. **D.** Default FDP Client serves metadata to the Web, and includes input forms to collect metadata from the host custodian. **E.** Hitch and Varnish are (optionally) installed and pre-configured to communicate with the default FDP client to proxy HTTPS traffic. **F.** Metadata updater is triggered by the completion of the transformation pipeline to update the FDP Metadata with e.g. last update time and a variety of ontological annotations that are contained in the data layers, to enahnce search/disovery of the FDP. **Additional Tools and Accessories: G.** MS Excel-based template transformer/loader for bootstrapping the full FDP metadata record. **H.** Alternative RDF-XML/XSLT-based visualization of data from the FDP client to create a harmonized human-readable/machine-readable interface. **I.** FDP "home"—the registry and search interface for deployed FDPs.

3.1 Tools

3.1.1 FAIR-in-a-box: An FDP Installer, and More

Behavior

The core FDP consists of several components:

- GraphDB: used to store the RDF metadata, and optionally, any Linked Data that they might describe.
- mongoDB: used to store administrative data of the FDP such as access control and resource definitions.
- FDP Server: a set of code modules that facilitate interactions with the administrative and metadata layers, and their database back-ends.
- FDP Client: a set of code modules that produce the human-friendly Web front-end for the FDP.

Historically, these components were difficult to install, bootstrap and coordinate. Subsequently, these components have all been published as Docker images, and were used within the European Joint Programme on Rare Disease (EJP-RD) to power their publication of rare disease Common Data Elements (CDEs), named "CDE-in-a-box" [12]. This was further refined and expanded to produce "FAIR-in-a-box" (Figure 1), which we introduce for the first time here. The FAIR-in-a-box installer script now runs all installation and bootstrapping/configuration steps automatically other than the download of GraphDB, since its license does not allow redistribution, thus requiring that every user request their own license. FAIR-in-a-box installs the GraphDB metadata repository (Figure 1B), the FDP Server (Figure 1C), and the FDP Web Client (Figure 1D). Beyond the metadata that is served by the default FDP, FAIR-in-a-box has an additional pipeline—based on YARRRML (Figure 1.A1; [13]) and RDF Mapping Language (RML) (Figure 1.A2; [14])/SDM-RDFizer [15]—that supports the creation and import of FAIR data into a separate database in GraphDB. FAIR-in-a-box has two further components, one that updates the metadata layers automatically when the underlying data changes (Figure 1F), and a second that provides a pre-configured solution, based on Hitch+Varnish [16], to enable a Secure Sockets Layer (SSL) proxy (Figure 1E) to the FDP Client, thus simplifying provision of HTTPS access to the FDP.

Implementation

Unlike CDE-in-a-box, FAIR-in-a-box takes advantage of Docker volumes, providing more security and flexibility in deployment. A bash script executes and coordinates between the installation and bootstrapping docker-compose files that result in a correctly configured FAIR Data Point and can be re-executed as many times as required for testing and custom-configuration until the desired result is achieved. Subsequently, a stand-alone docker-compose file (FAIR-ready-to-go) coordinates between the final 10 docker images required to run the fully functional FAIR-in-a-box, allowing the user to delete all of the installation folders and move the installation anywhere they wish on their system.

Custom configuration options include the color scheme of the default FDP homepage, a custom branding image on the homepage, and a custom icon. Customization simply involves edits to configuration files in a folder that is mounted in the docker image during start-up.

Repository

https://github.com/markwilkinson/FAIR-in-a-box

3.1.2 FDP Metadata Loader and Updater

Behavior

While the default FDP front-end (Figure 1D) allows manual entry of new records via a Web form, not all DCAT fields are available through that interface. Moreover, an FDP custodian may wish to exercise some curation over the entry of new data records into their FDP; correct metadata entry requires some experience with FAIR, but the FDP custodian will often not have the required domain-knowledge to undertake all facets

of the metadata entry themselves. This need for collaboration between the data creator, and the FDP custodian limits the ability for FDPs to be loaded in-bulk.

Within the EJP-RD there are 35 participating countries and 87 beneficiaries, most of which will have one or more patient registries or biobanks. With the plan to deploy FDPs for all these participants, we have developed a templating system (Figure 1G) that can be distributed to the participants, and then curated by the FDP custodian, prior to a fully automated load of that curated metadata.

Finally, the FDP is intended to be a metadata reflection of the content of a certain datastore. For those using the FAIR data transformation and storage capabilities of the FDP (for example, via FAIR-in-a-box) an "Updater" component (Figure 1F) can be loaded as part of the FDP docker-compose (or deployed separately), such that changes in the data layer are automatically extracted and published in the FDP metadata, ensuring that the two layers stay synchronized.

Implementation

For the FDP Metadata Loader, an MS Excel template has been designed that captures (most) elements from the DCAT schema. It is documented with the kind of data that should be entered, and some examples. An end-user will complete whichever fields for which they have metadata (leaving blank the rest) and provide this to the FDP custodian for curation. The FDP custodian may choose to make edits to the provided data (e.g., for consistency or language). They then save the file with a defined name and execute a docker-compose. The docker image is a Ruby application that extracts the fields from the Excel file, executes some "sanity checking" on the content of the file, and then uses a set of Ruby objects to build a set of Catalog, Dataset, and Distribution records. It also automatically builds a SKOS Concept Scheme from all ontological annotations provided about a Dataset and connects the concepts into that scheme (something that is encouraged by the DCAT specification, but currently not supported by the default FDP front-end). The application then uses the FDP REST API to first submit a skeleton Catalog, Dataset, and Distribution to the FDP, each of which is assigned a Globally Unique Identifier by the FDP registry. Those identifiers are then entered into the full datasets in order to properly link them to one another and give them a variety of "identifier" attributes, as required by the FAIR Principles. Finally, the full records are HTTP PUT into the FDP to replace the skeleton and are marked as "published" via the FDP API.

Using this loader, we have created FDP records that achieve near-perfect scores when assessed by the FAIR Evaluator [17], being some of the highest-scoring digital objects ever tested. This, therefore, is currently the optimal way to create metadata records that meet almost all FAIR requirements.

The FDP Updater is implemented as a Ruby Sinatra application in a Docker image, which can be deployed independently of the FDP docker-compose or added to it. It can be "triggered" manually by calling its Web interface, or automatically (for example, from a cron) to update on a regular basis. The Updater executes a series of SPARQL queries on the FAIR data triplestore, trying to extract ontological annotations. These are, by their nature, not personally identifiable and therefore can be safely exported into the metadata layer. The Updater retrieves the "Dataset" FDP metadata record describing that triplestore and

replaces any existing ontological annotations with the new annotations, then pushes it back into the FDP. It similarly retrieves the "Distribution" record and updates the "modified" property with the current time. Thus, the Updater largely eliminates the need for manual FDP curation.

Repository

Metadata Loader: https://github.com/markwilkinson/FAIR-in-a-box/tree/main/FDP%20Metadata%20Loader

FDP Updater: https://github.com/markwilkinson/FAIR-in-a-box/tree/main/FDP-Distribution-updater

3.2 Interfaces

3.2.1 Default FDP Front-End

Behavior

The FDP Client (Figure 1D) is one of the default components that provide a human-friendly front-end to the FDP. It allows people to browse the metadata records in a highly configurable responsive web UI. Each metadata record clearly shows its key information (configured to be displayed) and lists of child records, potentially of multiple types. Moreover, it always provides links to visit or download underlying RDF representation of the record in Turtle, JSON-LD, or RDF/XML format. The search functionality also shows a list of records based on the user-entered search query. A screenshot of the FDP Website for the Duchenne Parent Project is shown in Figure 2, where the ontological annotations describing the data deposit are rendered as ovals, and other key metadata facets are visible.

This component is not intended only for browsing but also for manual metadata entry via Web forms. Based on their role and permissions, authenticated users may manage (create, edit, or delete) metadata records. New records are first created as a draft that can be immediately or later published. Finally, a user who owns the metadata record may manage its permissions, such as adding other users as owners or as data providers in case of a catalog.

Another crucial part of the FDP Client is the administration of the FDP Server (Figure 1C). Users with the administrator role can easily manage the FDP instance, including user management, resource definitions, and shapes specification, basic settings, or resetting the FDP to defaults. The specification of resource definitions includes provision of schematic data models (called "shapes") that are defined using the Shapes Constraint Language (SHACL [18]). Shapes are essential for the Client, as they affect both how metadata records are rendered within its detail and list views as well as guide the creation of data-entry forms.SHACL shapes may also contain Data Shapes Vocabulary (DASH [19]) statements to extend what can be expressed in SHACL alone. The FDP Client renders metadata records and forms based on DASH viewers and editors. The FDP Client allows publishing shapes and importing them from other FDP instances, promoting their reusability.

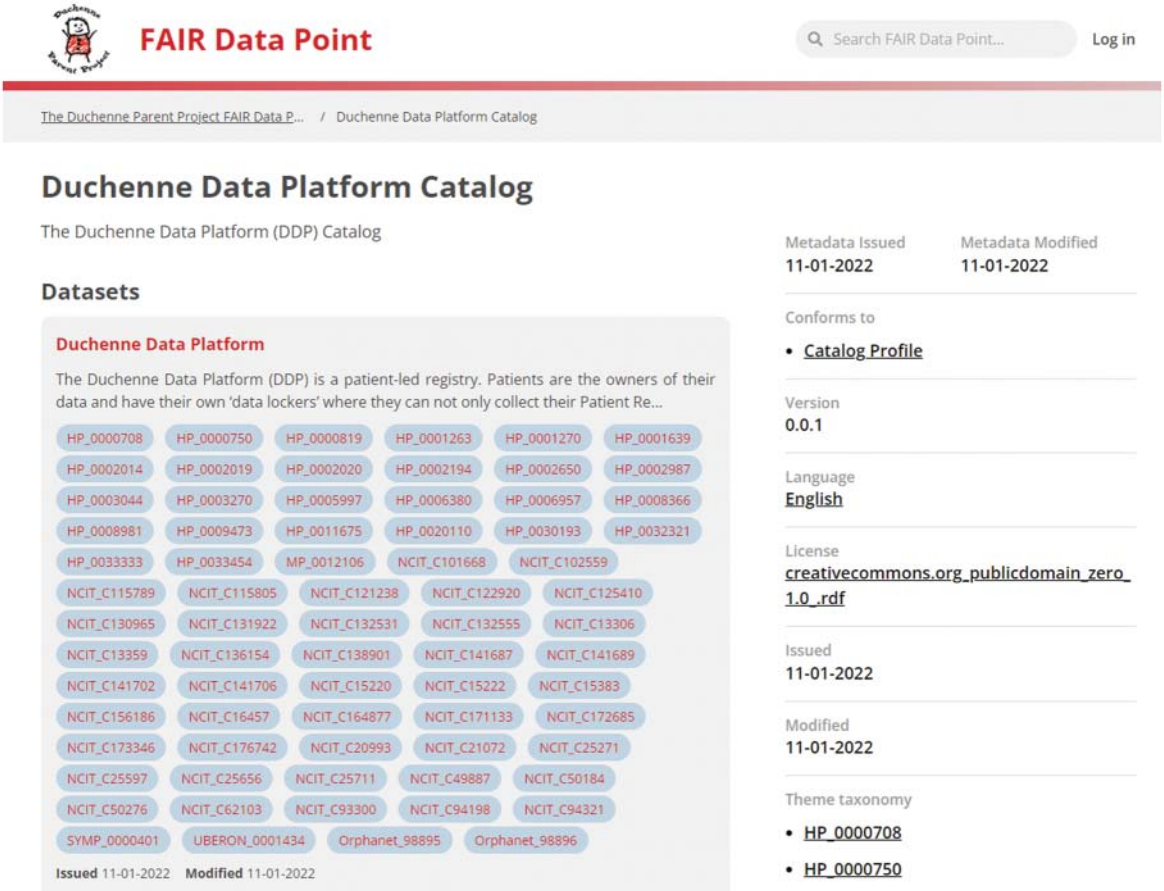

Figure 2. The FAIR Data Point Website as generated by the default FDP front-end. An overview of the deposit and its ontological annotations is provided in the center of the page. Key metadata is provided on the right panel. The display is customizable by color-scheme and logo, as shown.

Shapes are not used directly but through their resource definitions, which allows the composition of multiple shapes and supports their reusability in multiple resource definitions. There is also additional information in a resource definition, such as name, URL prefix, external links, and last but not least, specification of child relations (links to other resource definitions with necessary details, e.g., predicate and name). Resource definitions, therefore, are specifications of metadata records that also serve for the FDP Client to render metadata records and allow their creation or editing. Thus, the structure of the metadata record is flexible and not hard coded in the FDP Client component.

Implementation

The FDP Client is implemented as a Web user interface using the Vue.js framework and Bootstrap components. As all the metadata in the FDP Server can be configured during runtime, the client needs to

be able to work closely with them. The rdflib.js library (a JavaScript library that can parse and work with RDF) is used to render the correct metadata and forms defined in SHACL shapes and DASH.

The visual style itself is defined using the Syntactically Awesome Style Sheet (SCSS) standard. The colors and other variables are defined as SCSS variables that can be easily modified to customize how the FDP Client looks. This setup allows changing the default variables (such as changing the primary color), setting up a custom logo, or overriding some styles from scratch. Therefore, it is easy to customize the FDP Client to match the users' desired style.

The application is built into a Docker image that uses nginx to serve the client assets (such as HTML or JavaScript files). When the client is deployed together with FDP, it serves as a reverse proxy in front of the FDP itself (Figure 3). Based on the prepared nginx configuration, the requests are either served by the client nginx or forwarded to the FDP. Therefore, bots' requests for machine-readable formats can be served directly by FDP while the requests from the browser show a human-friendly user interface. The usual production deployment includes another reverse proxy for the client to handle HTTPS certificates and keep all other components secured in the internal Docker network.

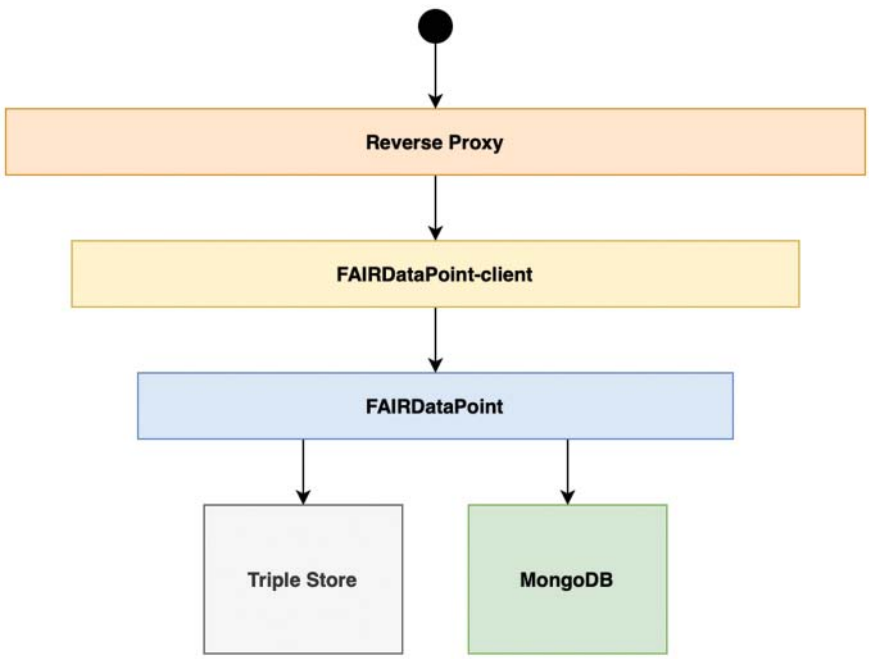

Figure 3. Overview of the deployment architecture of FDP with FDP Client, showing the reverse proxy in front of the client.

Repository

https://github.com/FAIRDataTeam/FAIRDataPoint-client

3.2.2 Extensible FDP Browser

Behavior

Since the initial publication of DCAT, a new version (DCAT2) has become available. One of the most important differences between DCAT and DCAT2 is the ability to extend the DCAT model through inheritance from the basic DCAT "Resource" abstract class. Thus, DCAT2, much more closely matches our initial requirement—that is, multiple layers of metadata, and specific sub-types of metadata record, beyond the three supported by the original DCAT. Consequently, FDP now supports DCAT2. However, this immediately raised a problem: given that DCAT2 records are exclusively metadata, with no pre-defined visualization, the inherited classes allowed in DCAT2 could not reliably be rendered in the default FDP front-end described above, since they would have distinct and unpredictable data elements. While this is not problematic for mechanized exploration, it is difficult to make these data useful to a human.

To respond to this challenge, we have created an extensible "generic" FDP browser (Figure 1H), which relies on the largely predictable structure of a DCAT record, or its DCAT2 inherited classes, to execute an XML Stylesheet Transform of metadata in an RDF-XML serialization into HTML that is rendered in the browser. The resulting visualization has been successfully demonstrated over numerous FDPs from a variety of sources (including sources not using the "official" FDP server software, but rather, just mimicking its API). We also demonstrate that, beyond providing a "generic" view over new kinds of data resources, the FDP Browser is extensible to generate resource-type specific views. For example, an inherited resource class can be rendered in a specific manner simply by adding a type-specific XSLT file into the FDP Browser's repository. An example of a page rendered using this browser is shown in Figure 4.

Implementation

The extensible browser is written as a Ruby Sinatra Web application that can be deployed on any server that runs Docker. It acts as a proxy, connecting to any FAIR Data Point and modifying the output before returning it to the browser. The precise sequence of events are:

1. The user enters their desired FDP address
2. The Web app then engages with that FDP endpoint via content-negotiation, to retrieve Linked Data
3. That linked data is queried by SPARQL to determine which DCAT Resource type, or inherited type, it is.
4. The type is matched against a repository of XSLTs
5. The Linked Data is transformed into RDF/XML syntax via the Ruby Distiller [20] libraries.
6. The resulting XML is then manipulated to add a stylesheet reference(s) to its header, referring to the selected XSLT and, optionally, a CSS.
7. The RDF/XML is then passed to the browser and is rendered into HTML client-side.

The XSLTs are designed such that DCAT and LDP-specific links are passed back into the proxy, ensuring that the extensible browser can be used to fully explore an FDP. Other links found in the metadata record are rendered as normal outward-pointing hyperlinks.

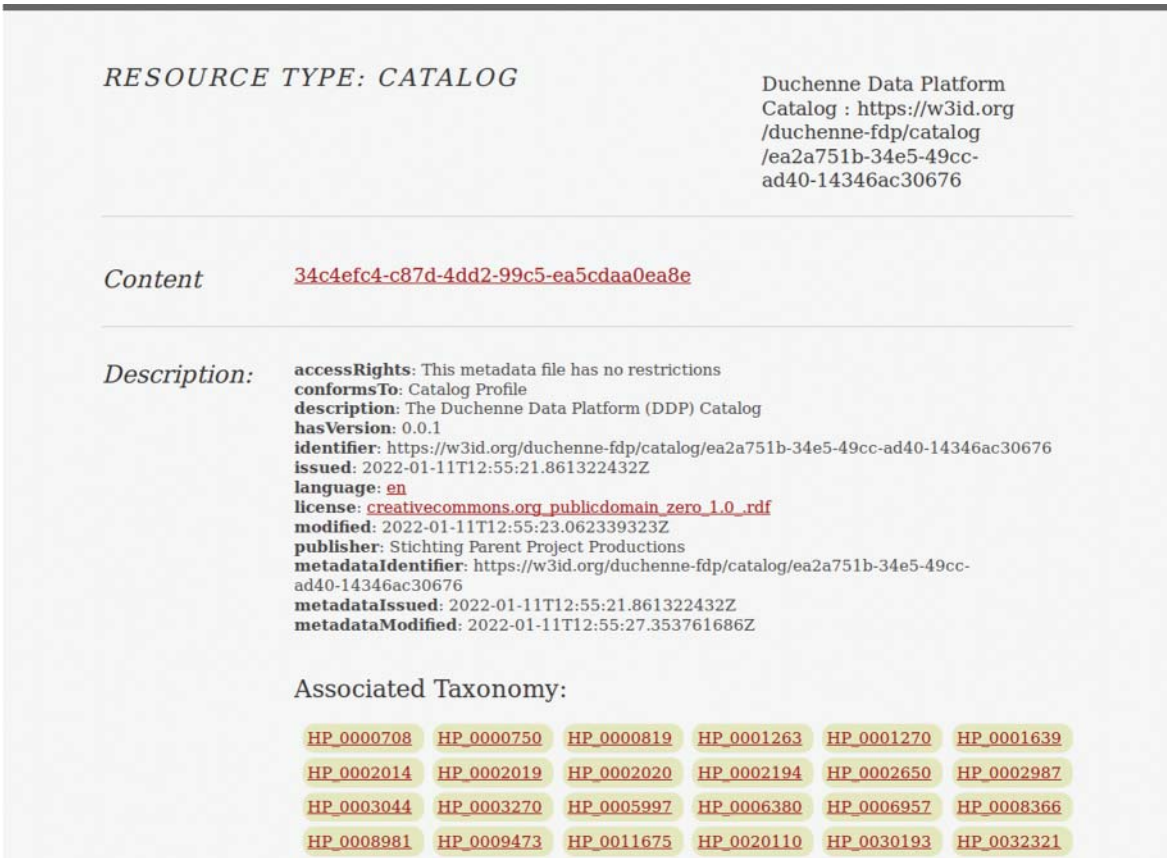

Figure 4. The Duchenne Parent Project FAIR Data Point rendered in the "generic" browser. The XSLT stylesheet has specific templates for the key components of the DCAT record, such as the pointer from a DCAT Catalog to its contained Datasets, and the ontological annotations associated with each dataset. Other metadata is simply iterated over, with the key being displayed in bold, and the value being either plain text, or a hyperlink.

Repository

https://github.com/markwilkinson/FDP-Browser

3.2.3 FDP Registry and Index

Behavior

Upon installation and initial start-up, an FDP will attempt to connect to an FDP registry (Figure 1I). This connection triggers the new FDP to be registered and, once metadata are entered and/or updated, the FDP registry can harvest and index the content, allowing all registered FDPs to have their content discovered. The FDP registry monitors the status and availability of known FDPs, including testing them for compliance with DCAT and the FDP API. Moreover, the FDPs notify the FDP registry on updates in their metadata

content, allowing the registry to have its indexed metadata as up to date as possible. The interface to the FDP registry is shown in Figure 5 (the second-to-last entry is the FDP shown in Figure 2).

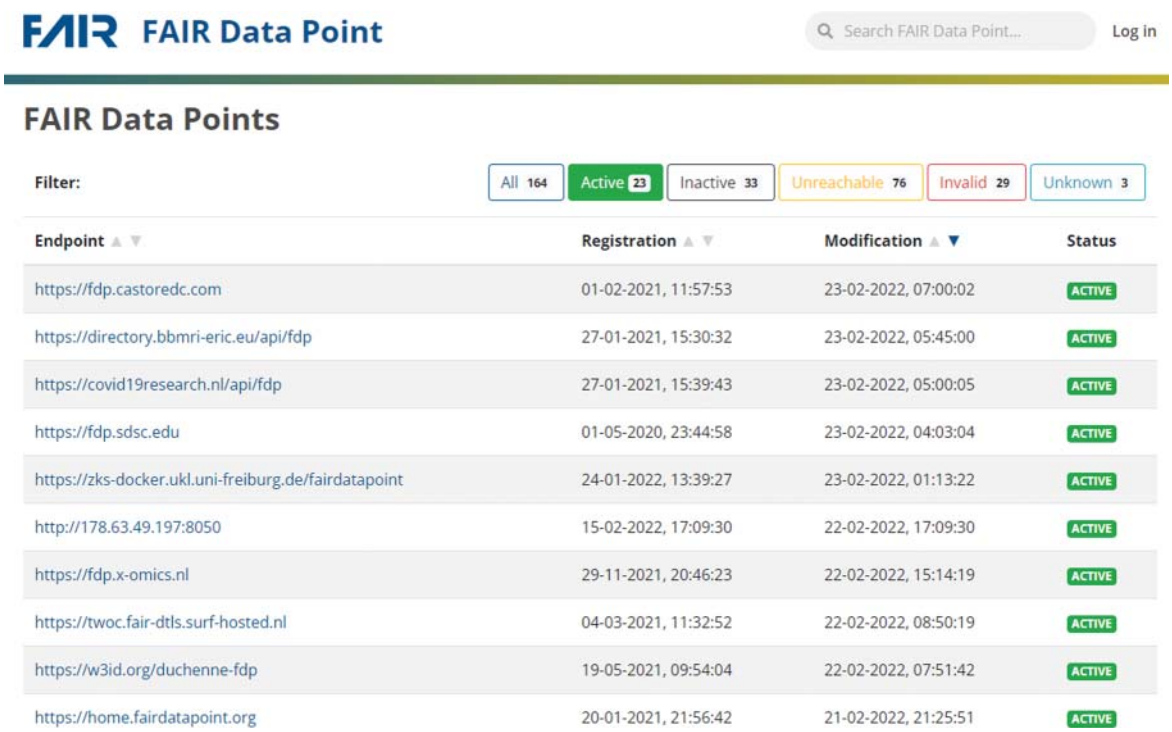

Figure 5. The FAIR Data Point index homepage. Every active FDP is listed, together with a link to its homepage. The status of every known endpoint is known by the index, and the visible list can be filtered based on status, including those that are active, but out-of-spec ("invalid") based on screening for required DCAT features. This page can be retrieved in Linked Data format to be processed by machines for the purpose of federated query.

Implementation

In the FDP reference implementation, the "call home" function, which registered the FDP in the registry, can be disabled. Moreover, multiple registries may be used by a single FDP. This allows the FDP registry to also be distributed, i.e., there is no dependency to a central FDP registry. This not only supports flexibility in the deployment topography but also addresses the risk of having a single point of failure. Using this approach, we can have a variety of different scenarios. For instance, in one scenario we could have a number of domain-specific FDP registries, each one indexing the content of FDPs of a particular domain. Then, a domain-independent FDP registry registers and indexes the content of the domain-specific registries, allowing users to search for content spanning different domains.

The FDP registry has been implemented on top of a regular FDP. Therefore, its content (the harvested metadata records of the registered FDPs) can also be navigated as a regular FDP. Moreover, the search

capabilities of the FDP registry are also available in regular FDPs. The only difference is that, in a regular FDP, the search is applied on the metadata records in that FDP while in the FDP registry, the search is applied on the harvested metadata records of all FDPs registered in the FDP registry.

Repository

https://github.com/FAIRDataTeam/FAIRDataPoint

Deployment (demonstration)

https://home.fairdatapoint.org/

4. DISCUSSION AND CONCLUSIONS

It is hard to be FAIR! This is largely because the primary target "user" for FAIR is the machine, rather than the human. As such, precision and richness of metadata are required, arguably, far more than would be for a human, and this is achieved through the use of languages and syntaxes that were never intended for human consumption. Yet with FAIRness increasingly becoming a requirement from journals, funding agencies, and other oversight and regulatory bodies, there is an urgent need to bring achievement of the FAIR Principles within-reach of the non-technical expert. This has been the primary focus of this article.

The FAIR Data Point was, to our knowledge, the first metadata publication platform that made FAIRness it's primary goal. The FDP could now well be considered "commodity software", having achieved a very high level of quality and usability in the past few years. Here we described additional tooling that simplifies its installation and use even further, with a particular focus on doing the most difficult parts of FAIR in an automated manner such that well-resourced users will not be at an "unfair" advantage compared to their peers who may not have the ability to hire technologists or professional data stewards, yet still face FAIR requirements from a variety of agencies.

This article also speaks to the primacy of the data in the FAIR ecosystem, versus human-oriented displays of that data. The FDP Browser imagines a world that consists exclusively of data—with no (tightly) associated Web pages—designed to be explored by machines. In this world, visualization for humans is secondary, and moreover, unlinked from the data itself, allowing a wide range of visualizations or layouts to be utilized to enhance human understanding of the data, when necessary. This is, in fact, a return to the original vision of the Web, where Tim Berners Lee proposed "*we should work toward a universal linked information system, in which generality and portability are more important than fancy graphics techniques and complex extra facilities.*" [21]. Our approach to solving this problem—the application of XSLT over RDF/XML—is not novel, and is used by other prominent resources in the Semantic Web in Life Sciences space, such as the SemanticScience Integrated Ontology (SIO) [22] and Ontobee [23]. The difference, and challenge, in our scenario was that unlike SIO and Ontobee we were not the originating source of the data that needed to be transformed, and in fact, there are numerous independent sources. Given that there are many ways to

serialize RDF into RDF/XML, and given that our data sources were varied, including sources not using the core FDP software, our solution involved ingesting the RDF from the source (in whatever format they provide), and then re-serializing it into RDF/XML using a common serialization engine. This was surprisingly effective, and the (so far) reliably consistent serialized XML structure allowed us to build a "generic" browser that could represent all manner of FDPs. Nevertheless, while we were surprised by the apparent stability of this approach, it is no doubt tightly linked to the fact that FDPs emit predictably structured metadata—based on DCAT and LDP—thus we do not suggest that this is a more general solution. A transformation language similar to XSLT, but based on Linked Data technologies rather than XML, seems to be a clear but unmet need.

And finally, FAIR exists to encourage and support data reuse. The network of FDPs assembled at the index of registries can be cross-searched easily in order to find all possible locations that might contain problem-relevant data. This is an important advance towards building an Internet where machines can find, access, and reuse data autonomously.

For FAIR to achieve the universality required for its goals to be achieved, it is necessary to support the broad community of scholarly data publishers that need to implement FAIRness over their own resources, for both humans and machines. These tools, we believe, lower the barrier to participation, and help maximize the utility of FAIR data publications.

ACKNOWLEDGEMENTS

OMB is funded by a fellowship from the Algerian Ministry of Higher Education and Scientific Research. MDW and RK are funded via the European Union's Horizon 2020 research and innovation programme under the EJP RD COFUND-EJP No. 825575. The research activities of MS and JS are supported by Czech Technical University in Prague grant No. SGS20/209/OHK3/3T/18. LOBSS, RK and KB are partially funded by funding from the Horizon2020 projects FAIRsFAIR grant No. 831558.

AUTHOR CONTRIBUTIONS

Oussama Mohammed Benhamed (oussama.benhamed@alumnos.upm.es) and Mark Wilkinson (mark.wilkinson@upm.es) are primarily responsible for FAIR-in-a-Box, the FDP Metadata Loader, and FDP Extensible Browser. Kees Burger (c.a.burger@lumc.nl) is the primary implementation engineer for the FDP architecture, reference implementation development lead; Rajaram Kaliyaperumal (r.kaliyaperumal@lumc.nl) assisted in design of the FDP architecture, semantic modeling,and creating the FDP ontology; Luiz Olavo Bonino da Silva Santos (l.o.boninodasilvasantos@utwente.nl) is the FDP architecture and design lead of the FDP architecture and ontology; Marek Suchanek (marek.suchanek@fit.cvut.cz) and Jan Slifka (jan.slifka@fit.cvut.cz) are engineers involved in reference implementation development. Kees, Rajaram, Luiz, Marek and Jan are directly involved on the FDP Server, FDP default Web client and the FDP index and registry. All authors contributed to the article by writing, revising, designing figures, and running tests.

REFERENCES

[1] Berners-Lee ,T.: The Semantic Web: A New Form of Web Content that is Meaningful to Computers Will Unleash a Revolution of New Possibilities (2002)

[2] World Wide Web Consortium: RDF—Semantic Web Standards, https://www.w3.org/RDF/

[3] World Wide Web Consortium: OWL—Semantic Web Standards, https://www.w3.org/OWL/

[4] Yu, L.: Linked Open Data. In: A Developer's Guide to the Semantic Web, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 409–466 (2011)

[5] McCrae, John P., Abele, Andrejs., Buitelaar, P., et al.: The linked open data cloud. Available at:https://lod-cloud.net. Accessed Jan. 10, 2022

[6] Wilkinson, M.D., et al.: The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data 3, 160018 (2016),

[7] Wilkinson, M.D.: FAIR Data Prototype—Interoperability and FAIRness through a novel combination of Web technologies. Available at: https://www.slideshare.net/markmoby/fair-data-prototype-interoperability-and-fairness-through-a-novel-combination-of-web-technologies. Accessed Jan. 10, 2022

[8] Maali, F., Erickson, J., Archer, P.: Data catalog vocabulary (DCAT). W3C Recommendation 16, (2014)

[9] Speicher, S., Arwe, J., Malhotra, A. (2015): Linked data platform. W3C.

[10] Wilkinson, M.D., et al.: Interoperability and FAIRness through a novel combination of Web technologies. [Epub ahead of print], doi: https://doi.org/10.7717/peerj-cs.110 (2017)

[11] Bonino, D.S.S., Wilkinson, M.D., Kuzniar, A., Kaliyaperumal, R., Thompson, M., Dumontier, M., Burger, K.: FAIR Data Points Supporting Big Data Interoperability. In: Enterprise Interoperability in the Digitized and Networked Factory of the Future, Zelm M, Doumeingts G, and Mendonça JP, eds. iSTE Press, pp. 270–279 (2016)

[12] Kaliyaperumal, R., Wilkinson, M.D., Alarcón, P., et al.: Semantic modelling of Common Data Elements for Rare Disease registries, and a prototype workflow for their deployment over registry data. http://medrxiv.org/lookup/doi/10.1101/2021.07.27.21261169 (2021)

[13] Heyvaert, P., De Meester, B., Dimou, A., et al.: Declarative Rules for Linked Data Generation at Your Fingertips!, http://dx.doi.org/10.1007/978-3-319-98192-5_40 (2018)

[14] Dimou, A., Vander Sande, M., Colpaert, P., et al.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: LDOW, researchgate.net (2014).

[15] Iglesias, E., Jozashoori, S., Chaves-Fraga, D., et al.: Vidal M-ESDM-RDFizer. http://dx.doi.org/10.1145/3340531.3412881, (2020).

[16] Hitch. Available at: https://www.varnish-software.com/community/hitch/. Accessed Jan. 10, 2022

[17] Wilkinson, M.D., Dumontier, M., Sansone, S-A., et al.: Evaluating FAIR maturity through a scalable, automated, community-governed framework. Scientific Data 6, 174 (2019)

[18] Gayo, J.E.L., Prud'hommeaux, E., Boneva, I., et al.: Validating RDF Data. Springer (2017)

[19] DASH Data Shapes Vocabulary. Available at: https://datashapes.org/dash. Accessed Jan., 10, 2022

[20] Kellogg, G.: Ruby Distiller. Available at: http://rdf.greggkellogg.net/distiller. Accessed Jan., 10, 2022

[21] The original proposal of the WWW. Available at: https://www.w3.org/History/1989/proposal.html. Accessed Jan., 10, 2022

[22] Dumontier, M., Baker, C.J., Baran, J., et al.: R: The Semanticscience Integrated Ontology (SIO) for biomedical research and knowledge discovery. Journal of Biomedical Semantics 5, 14 (2014).

[23] Ong, E., Xiang, Z, Zhao, B., et al.: Ontobee: A linked ontology data server to support ontology term dereferencing, linkage, query and integration. Nucleic Acids Research 45, D347–D352 (2017)

AUTHOR BIOGRAPHY

Oussama Mohammed Benhamed has a Master's degree in Nutrition and Dietetics. Oussama is a Ph.D. student at the Universidad Politécnica de Madrid, Spain. He is also a software and a full stack web developer.
ORCID: 0000-0002-2567-1914

Kees Burger is a software engineer at the Leiden University Medical Center. He is leading the development of FAIR infrastructure reference implementations, and has been involved with FAIR developments since the inception of FAIR.
ORCID: 0000-0002-5437-779X

Rajaram Kaliyaperumal was born in Pondicherry, India. He received a B.Tech degree in Biomedical Engineering from Pondicherry University, India in 2008 and an M.Sc degree in Biomedical Engineering from Linköping University, Sweden in 2011. In 2012 he joined the department of Computer and Information Science, Linköping University as a software engineer. During this time, he developed methods and tools to align and repair ontologies. In 2013 he joined the Biosemantics group, Leiden, in the Netherlands as a software developer. His current research activities include investigating the use of semantic web technology in the context of FAIR data and developing prototypes to demonstrate generating and the use of FAIR data.
ORCID: 0000-0002-1215-167X

Luiz Olavo Bonino, PhD, is Associate Professor of the Services and CyberSecurity group at the University of Twente and Associate Professor of the BioSemantics group at the Leiden University Medical Centre. Luiz has a background in ontology-driven conceptual modeling, semantic interoperability, service-oriented computing, requirements engineering and context-aware computing. In the past 8 years Luiz has worked on designing and developing technologies, methods and tools to support making, publishing, indexing, searching and annotating FAIR (meta)data.
ORCID: 0000-0002-1164-1351

Marek Suchánek is a PhD student and associate researcher at FIT CTU in Prague focusing on conceptual modelling and model-driven development in software engineering. He also works as a software developer and analyst, contributing to FAIR Data Point or Data Stewardship Wizard. He always pursues the application of research ideas in practical use cases. His current interests are mainly FAIR data management, machine-actionability, and designing services to support FAIR and open science.
ORCID: 0000-0001-7525-9218

Jan Slifka is a PhD student at the Czech Technical University in Prague, focusing on model-driven software development, user interfaces and sustainability. He is currently also working as a Software Engineer on tools related to open science, namely Data Stewardship Wizard or FAIR Data Point.
ORCID: 0000-0002-4941-0575

Mark D. Wilkinson, Ph.D., is Issac Peral Senior Investigator at the Center for Plant Biotechnology and Genomics of the Universidad Politécnica de Madrid, Spain. Mark was co-author on the original article describing the FAIR Data Principles, and continues to lead a research laboratory focused on the implementation of FAIR, and maximizing the exploitation of FAIR data.
ORCID: 0000-0001-6960-357X