FAIR Data APIs in the FAIR in Vivo Data Sharing Platform*

Felix Schwagereit[1,*,†], Martin Romacker[1,*,†], Fabien Richard[1,†], Robert Trypuz[1,†], Thomas Liener[1,†] and Olivier Roche[1,†]

[1]Roche Pharma Research and Early Development, Data & Analytics, Roche Innovation Center Basel, Switzerland

Abstract

Exposing and integrating data and its metadata in an industry production setting is challenging due to the amount of data and the number of systems, technologies, and people involved. We propose FAIR data APIs, based on REST endpoints and JSON-LD, as a method to accomplish machine-actionable interoperability and semantic awareness. This paper demonstrates a concrete implementation of such a FAIR data API as we discuss Roche's FAIR in vivo data sharing platform (FISH).

Keywords

FAIR Principles, FAIR Data API, FAIR API, Ontology, Roche Terminology Service, Model-driven APIs

1. Introduction

Data, the collection of digital objects, is an asset that brings tremendous value to an organization if it can be reused at any time and in any way without investing any major additional financial and human resources [1, 2, 3]. To accomplish this, the data must be well understood externally and internally. By *external understanding*, we mean machine-actionable[1] information about ownership, license conditions, permissions, formats, access points, and everything that can be said about the data as an information artifact. In [4, p.4], it is named "the contextual metadata surrounding a digital object ('what is it?')". By *internal understanding*, we mean an ontology that provides meaning to the data content and the means that make this data machine-actionable through an ontology—in [4, p.4], it is named "the content of the digital object itself ('how do I process it/integrate it?')". As pointed out in the [4, p.3], technology is no longer the reason why data still lacks proper external and internal understanding within an organization; "the reason is, that we do not pay our valuable digital objects the careful

SEMANTICS 2022 EU: 18th International Conference on Semantic Systems, September 13-15, 2022, Vienna, Austria

*Corresponding authors.

†These authors contributed equally.

felix.schwagereit@roche.com (F. Schwagereit); martin.romacker@roche.com (M. Romacker); fabien.richard@roche.com (F. Richard); robert.trypuz@roche.com (R. Trypuz); thomas.liener@roche.com (T. Liener); olivier.roche@roche.com (O. Roche)

0000-0003-2696-672X (F. Schwagereit); 0000-0001-6898-0226 (M. Romacker); 0000-0001-8192-3023 (F. Richard); 0000-0003-4042-9947 (R. Trypuz); 0000-0003-3257-9937 (T. Liener)

[1]We understand the phrase 'machine actionable' in the same way as in [4, p.3], i.e., as "a continuum of possible states wherein a digital object provides increasingly more detailed information to an autonomously-acting, computational data explorer."

attention they deserve." The FAIR Guiding Principles have been proposed to help with better data management, providing data owners with the means to expose proper machine-actionable external and internal understanding. However, to our knowledge, implementing these principles when building a FAIR system in the industry has not been described in detail yet.

This paper aims to show how the FAIR Guiding Principles have been followed when building the FAIR in vivo data sharing platform (henceforward, FISH) at Roche. We will present how the FAIRness of FISH has been achieved using application ontologies/models and carefully designed API endpoints to expose FAIR data and metadata to other systems (henceforward, FAIR Data APIs).

The paper is written with the following structure. Section 2 describes the FISH application models, their design, and their rationale. Section 3 introduces the concept of FAIR Data API and its usage in the software development process. Section 4 is dedicated to assessing the FAIRness of FISH, focusing on the FAIR Data API as an essential component contributing to FISH's FAIRness. Finally, section 5 concludes the paper and presents an outlook on possible enhancements of the FAIR Data API concept to turn it into a FAIR API.

2. Application ontologies with local restrictions

The FISH platform exposes FAIR data about non-clinical in vivo studies and consists of several components (henceforward, applications) where each of them manages and stores data about one primary digital object e.g. the Study Registration System (SRS) for studies, the Animal Registration System (ARS) for animals, the Study Designer (SD) for study designs and executions, the Biospecimen Registration System (BRS), and Assay Structure Definer (ASD) for assay protocols. FISH platform architecture consists of microservices between and within the FISH applications, relational databases for back-ends, JSON-based REST FAIR Data APIs, and graphic user interfaces.

2.1. Application ontologies

With FAIRness in mind, the first step was to create the semantic application ontologies on which the relational database schemas of FISH have been based. Semantic FISH ontologies are modularized in the same way as the FISH application is. Every application ontology is self-contained, i.e., contains only classes and properties relevant to the distinct FISH application and specifies the meaning of vocabulary used in the application. However, FISH application ontologies can share classes and properties when several FISH applications use those classes and properties. For instance, the SD ontology contains selected properties of the class 'Study' that are specified by/in the Study Registration System's ontology.

Regarding the expressive power of the FISH application ontologies, they are RDFS ontologies extended with OWL classes and all types of OWL properties. Each application ontology is stored in a separate named graph. Application ontologies are created through the Roche Terminology Service (RTS) (see figure 1). Nevertheless, they can be serialized, e.g., to Turtle, and used by external ontology editors.

In RTS, classes and properties of an application ontology (called "application model") come from a reference ontology, called "Reference Model" (see the left column of the window in figure

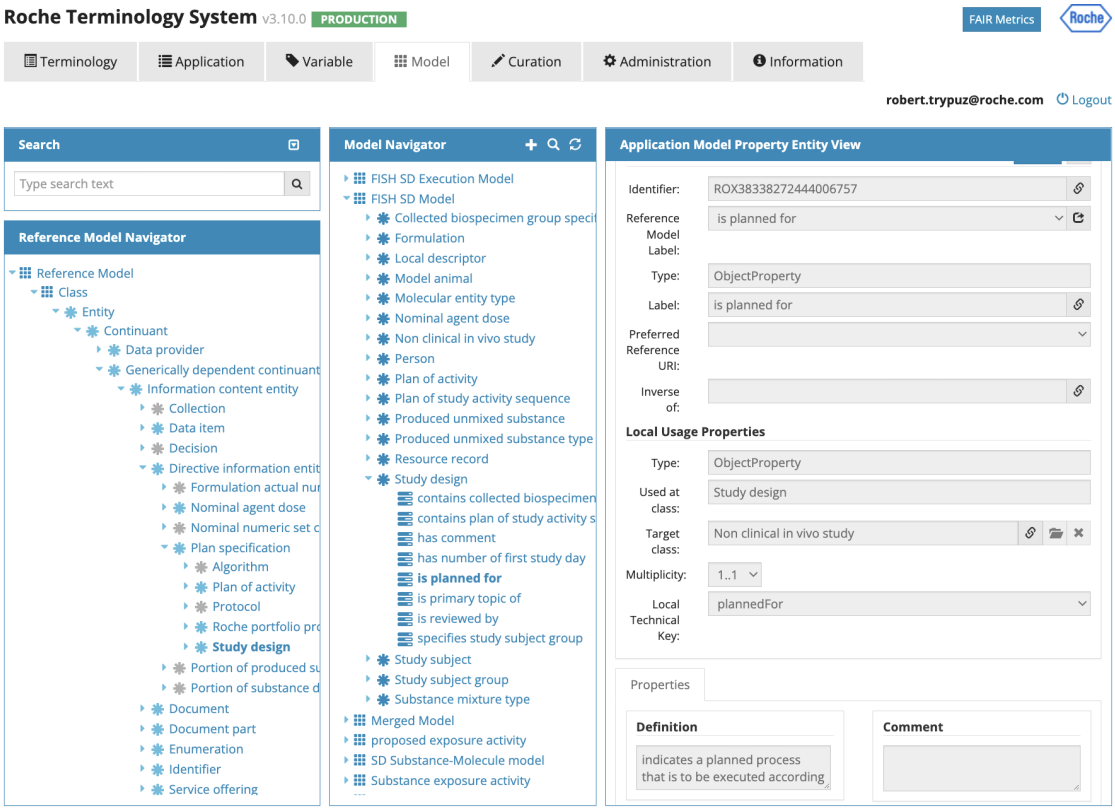

Figure 1: Study design and its properties in RTS

1), that is founded on the Basic Formal Ontology and contains all the classes and properties used by the RTS application models. The Reference Model contributes to data harmonization and interoperability across all the Roche applications described by the RTS application models. Classes and properties are organized into taxonomies and have labels and definitions compliant with the Naming and Textual Definitions conventions of the Open Biological and Biomedical Ontologies (OBO) Foundry.

An RTS application model can enforce a more restrictive meaning of classes and properties in the context of a given application. For instance, the FISH SD application requires that a study design is planned for exactly one non-clinical in vivo study (see figures 1 and 2). This constraint may not generally be valid outside of the FISH SD application. That is why we decided to use SHACL[2] node shapes instead of OWL restrictions to specify cardinality constraints, domains, and ranges of properties within application models (see figure 3). The SHACL shapes of an application model are stored in the named graph of the model. RTS allows for translating the application models with the SHACL shapes into OWL ontologies with OWL restrictions (see figure 2) and serializing them to Turtle. However, it should be remembered that such obtained OWL restrictions do not specify the general meaning of the classes and properties but reflect

[2]https://www.w3.org/TR/shacl/

local business rules specific to the application model.

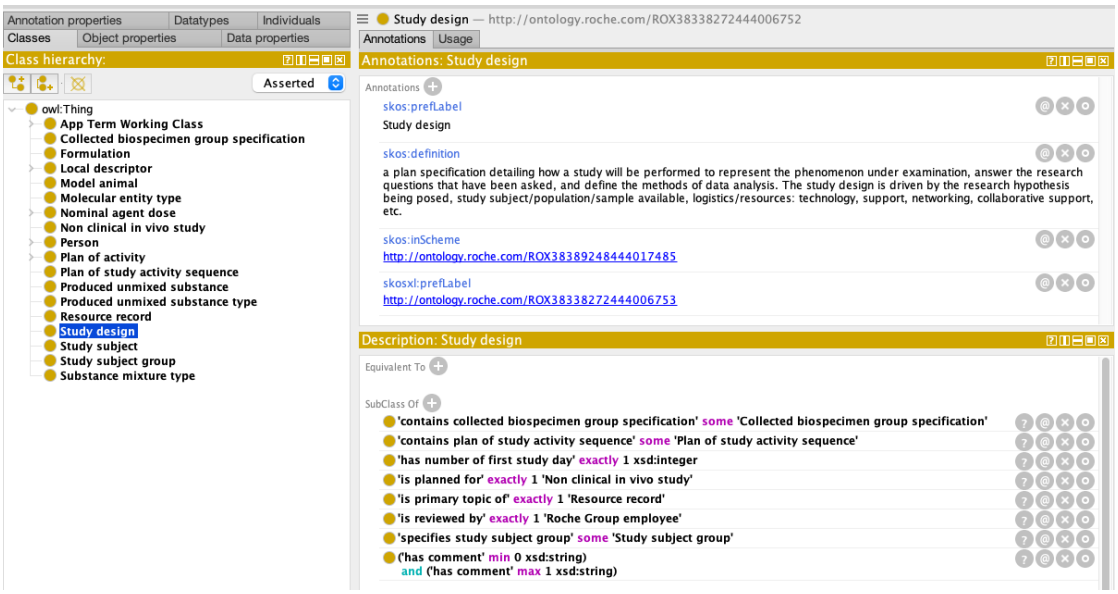

Figure 2: Study design htbpand OWL restriction

The application models' role extends beyond being conceptual schemas for the FISH databases and specifying the business rules since they also provide a vocabulary for the FISH's REST APIs. Namely, the payloads of the REST APIs are required to be serialized as JSON-LD and compliant with the application models. Section 3 will elaborate on this topic.

2.2. GUPRIs for application ontology elements and data instances

There is no FAIR without Globally Unique, Persistent and Resolvable Identifiers (GUPRIs). The FISH architecture decisions were that each FISH application is the reference system of some FISH application ontology classes and provides GUPRIs for the data instances classified by these classes. For instance, the Study Registration System (SRS) is the only authorized system to provide GUPRIs for the class 'Study' data instances. All the other systems must use the SRS GUPRIs when using or exposing the class 'Study' data instances. RTS is another example of a reference system and provides the FISH applications with the GUPRIs of the RTS application models' elements, i.e., the classes, properties, terminologies, and their digital objects. The FISH GUPRIs have the following schemas: when a class is the primary digital object defined by a FISH application), the GUPRI schema for the data instances of this class is https://id.roche.com/[FISH_component_code]/accession (e.g. https://id.roche.com/a2/123 for the FISH Study No 123), when a class is not a primary digital object, the GUPRI schema for the data instances of this class is https://id.roche.com/[FISH_component_code]/[classname]/accession, where accession is the primary key of a class instance generated by a FISH application database. The RTS GUPRI schema is https://ontology.roche.com/ROX[uniquenumber], for defining the

prefix objects within the application model the dash:stem[3] property is used.

3. FAIR Data APIs

Data stored in a semantic-aware database like a triple store can be easily retrieved using SPARQL. However, underlying systems often use more traditional tabular-based databases, as in FISH's case. To exchange data between systems, the de-facto standard in a microservice architecture is JSON-based API. We wrote above that FISH data stored in the relational databases is application ontology-compliant. The challenge was making the data machine-actionable through the JSON-based APIs, letting the external agents know what data FISH provides, and letting the external systems understand the APIs' payloads so they can benefit from FISH data. We propose the idea of FAIR Data APIs that use the application ontologies and JSON-LD to expose FAIR data. We believe semantically aware APIs should be vital to any industry-scale microservice architecture.

3.1. JSON-LD for payload

FISH FAIR Data APIs use JSON-LD for their payloads (i.e., the APIs' requests and responses). JSON-LD is a serialization format based on JSON. It is a W3C recommendation and is extensively used by many important projects to encode knowledge (e.g., schema.org and Google Knowledge Graph). By turning JSON into JSON-LD, it is possible to add semantics to the API reply, which adds much value: instead of "just" data, FAIR data is retrieved from databases via the API. JSON-LD combines three advantages: 1) it can serialize semantic triples, 2) it is JSON-based and therefore allows the use of the existing tools supporting JSON, and 3) it is widely used by application developers. Thus, JSON-LD provides a common ground where (semantic) data modelers and (application) developers can meet and communicate.

The application ontologies restrict requests and responses of the FAIR Data APIs. Similarly to schema.org they specify what classes and properties can be used and determine the properties' domains and ranges. Additionally, the application ontologies specify cardinality restrictions by SHACL shapes (see figure 3).

Below we present a FISH GET API response that finds study designs by the local study design identifier (id). For study design id equal 1 we have:

Listing 1: FISH GET API response for study design id equal 1.

```
{
   "@context":{
      "@import":"https://ontology-service.roche.com/rts2-api/v3/appmodels/ROX38389248444017485/context
          ?version=2022-06-08T09%3A05%3A30.000Z"
   },
   "@id":"https://id.roche.com/a6/1",
   "@type":"StudyDesign",
   "plannedFor":{
      "@id":"https://id.roche.com/a2/1",
      "@type":"NonClinicalInVivoStudy"
   },
   "numberOfFirstStudyDay":1,
   "comment":"some comment",
```

[3] https://datashapes.org/constraints.html#StemConstraintComponent

```
    "reviewedBy":{
       "@id":"https://id.roche.com/xyz/fishcur1",
       "@type":"Employee",
       "userName":"GLO FISH CURATOR"
    },
    "realizedBy":{
       "@id":"https://id.roche.com/ao/8",
       "@type":"StudyDesignExecution"
    },
    "isPrimaryTopicOf":{
       "@id":"https://id.roche.com/a6/DatasetRecord/1"
       "@type":"DatasetRecord",
       "createdBy":{
          "@id":"https://id.roche.com/xyz/max007",
          "@type":"Employee",
          "userName":"Max Mustermann"
       },
       "lastModifiedBy":{
          "@id":"https://id.roche.com/xyz/max007",
          "@type":"Employee",
          "userName":"Max Mustermann"
       },
       "createdOn":"2022-05-27T14:49:23.904+00:00",
       "lastModifiedOn":"2022-05-31T11:52:28.197+00:00",
       "status":{
          "@id":"ROX38098944443956511",
          "prefLabel":"Archived"
       }
    }
}
```

In the above example of a JSON-LD response from a FAIR Data API we find GUPRIs of the instance data e.g. https://id.roche.com/a6/1 for the study design `id=1` and https://id.roche.com/ao/8 for a study design execution `id=8` that the FISH applications has created according to the GUPRI schemas (see section 2.2). It is strongly recommended to make the GUPRI and the instance type explicit via `@id` and `@type`.

The payload is structured by the usage of keys and values, e.g., `"@type"` and `"StudyDesign"`. The keys are usually not directly defined as GUPRIs but as speaking character strings. JSON-LD allows for using strings instead of GUPRIs as long as a mapping from strings to GUPRIs is defined in the JSON-LD context (see listing 2). In RTS we assign and record a key (called "local technical key" e.g. `studyDesign`) for the GUPRI of a class or a property within an application model. Local technical keys are defined by means of SHACL's node shapes (see figure 3). Each shape can have a property called `prefLocalTechId`. We have the obvious requirement that the local technical keys are unique within one application model; otherwise, they could not be used as local keys to identify a shape in the application model (e.g. the node shape provided below) or generate a valid JSON-LD context from the application model.

The local technical keys are used to establish a mapping between SHACL shapes and key-value pairs in JSON-LD. Such mapping can be expressed in the JSON-LD context. Using the local technical keys, we have implemented a method to generate a context for a SHACL model automatically. The principle for the context generation is to create for each local technical key mapping to the class or property GUPRI (see listing 2) that the node shape or Property Shape is defined on. These contexts can always be generated on the fly, and RTS offers a context

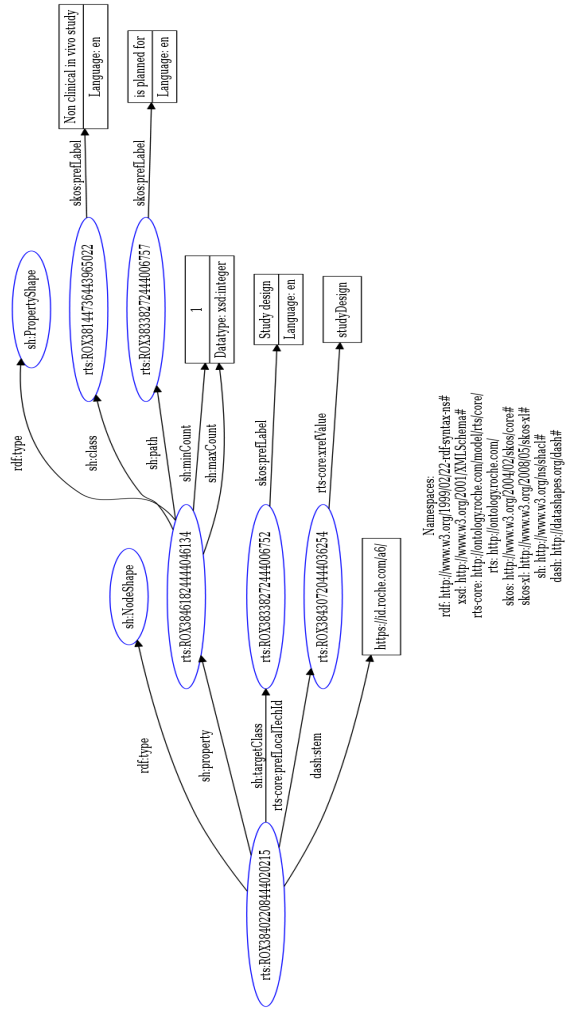

Figure 3: SHACL shape expressing a local restriction that every study design is planned for exactly one non-clinical in vivo study. It also provides the local technical key and the DASH stem/GUPRI prefix.

service for other applications. In JSON-LD payloads, our context generation REST endpoint can be referenced via @import with some agreed parameters so that the context for a specific SHACL model can always be looked up (see figure 3). A fragment of the context imported by the payload above looks like in listing 2.

Listing 2: An example of context

```
{
   "@context":{
      "@base":"http://ontology.roche.com/",
      "id":"@id",
      "type":"@type",
      "rts":"http://ontology.roche.com/",
      "xsd":"http://www.w3.org/2001/XMLSchema#",
      "StudyDesign":"rts:ROX383382724440067 52",
      "plannedFor":{
```

```
        "@id":"rts:ROX38338272444006757",
        "@type":"@id"
      },
      "numberOfFirstStudyDay":{
        "@id":"rts:ROX38347776444009619",
        "@type":"xsd:integer"
      },
      "comment":{
        "@id":"rts:ROX38350368444009851",
        "@type":"xsd:string"
      },
      ...
  }
}
```

With the generated context, we are able to translate a given JSON-LD back into triples with the application ontologies' URIs. Based on the generated context, we can offer a validation service that accepts a JSON-LD payload and responds to whether the payload fulfills a specific SHACL model (or not). The service is currently implemented with the RDF4J-SHACL library.

3.2. JSON-LD payload examples

To support the implementation and usage of FAIR data APIs and, in particular, to support the developers in their daily work, we offer one more helpful way to use the information we have specified within the application models. We have implemented a simple heuristic to generate valid JSON-LD payloads that cover all shapes in the application model by generating artificial (i.e., exemplary) objects and triples. In this algorithm, we use classes, properties, datatypes of datatype properties, and known object URIs to auto-generate a payload that mimics a real payload that a real REST API endpoint can implement.

We want to highlight a particular aspect due to one fundamental difference between a JSON-LD representation and an RDF representation of triples. Both are directed graphs, but JSON-LD is also a tree with a root element. In typical REST API endpoints, the root element is the object the endpoint is working on. E.g., an endpoint to retrieve details on a particular id of a registered object of type Study will retrieve a JSON-LD object as the root element of type Study. We made the node shape a parameter of our payload generation algorithm so that the algorithm starts traversing the graph of SHACL shapes at this shape further down to other shapes connected via Property Shapes to generate the example payload until a given depth is reached.

Listing 3: An example of automatically generated payload

```
{
  "@context":"https://ontology-services.roche.com/rts2-api/v3/appmodels/ROX38389248444017485/context?
      version=2022-04-07T14%3A46%3A15.000Z",
  "@graph":{
    "@id":"https://id.roche.com/a6/1",
    "@type":"StudyDesign",
    "comment":"string",
    "containsPlanOfActivitySequence":[
      {
        "@type":"PlanOfActivitySequence",
        "@id":"https://id.roche.com/a6/1/planActivitySeq/1",
        "doneOn":{
          "@type":"SubjectGroup",
          "@id":"https://id.roche.com/a6/1/studyGroups/1"
```

```
            }
         },
         ...
      ],
      "isPrimaryTopicOf":{
         "@id":"https://id.roche.com/a6/DatasetRecord/1"
         "@type":"DatasetRecord",
         "createdBy":{
            "@id":"https://id.roche.com/xyz/bond_007",
            "@type":"Employee"
         },
         "createdOn":"2021-08-26T14:44:11.882Z",
         "lastlyModifiedBy":{
            "@id":"https://id.roche.com/xyz/bond_007",
            "@type":"Employee"
         },
         "lastlyModifiedOn":"2021-08-26T14:44:11.882Z",
         "status":{
            "@id":"rts:ROX1446033330206",
            "prefLabel":"Drafted"
         }
      },
      "numberOfFirstStudyDay":123,
      "plannedFor":{
         "@type":"NonClinicalInvivoStudy",
         "hasParticipant":[
            {
               "@type":"Subject",
               "@id":"https://id.roche.com/a6/1/subjects/1"
            },
            ...
         ]
      },
      ...
   }
}
```

3.3. Usage in the software development process

The building blocks described above have been made part of the software development process as outlined in Figure 4. We support this process by implementing these building blocks within our RTS application that the software developers can directly use.

Based on the functional requirements, the application model is developed by model engineers with on-

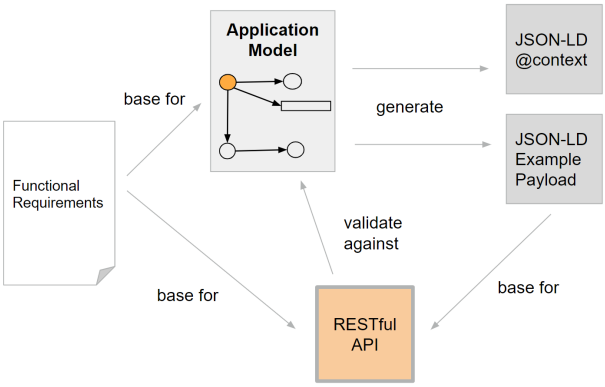

Figure 4: Model-driven API development

tology elements from the Reference Model. The application model should cover all the data elements received as input or provided as output from the API that will be built. Once designed, the application model is used to auto-generate a single JSON-LD in the RTS application. This generated context is referenced in all JSON-LD payloads of the API (e.g., in Listing 1). On demand, the API developers can also use the RTS application to auto-generate JSON-LD example payloads for objects based on the application model as a blueprint for implementing or interpreting payloads within the API. When an API implementation has been coded, the developer can then use functionality in RTS to auto-validate the payloads of their particular endpoints against the application model to check if the implementation is aligned with the application model.

4. FAIR Data APIs and the FAIR Guiding Principles

It is evident that to have a FAIR data ecosystem, the API itself plays an essential role. No less important are modeling, preparing, and cleaning the data. Also, additional services like a terminology service might be necessary to accomplish a FAIR Data API. The FAIR maturity indicators have been published to help assess the FAIRness of data or systems (see e.g. [5, 6, 7]). Many FAIR maturity indicators—namely **F1**, **F2**, **F3**, **A2**, **I2** and **I3**—need to be fulfilled as a prerequisite *for implementing a FAIR Data API*, and be taken into account during the design phase of a FAIR system. Some FAIR maturity indicators can be facilitated and supported *by implementing the FAIR Data API* e.g. **F4**, **A1**, **A1.1**, **A1.2**, and **I1**. Below we show how each FAIR maturity indicator is addressed by FISH and its FAIR Data APIs.

- **F1.** *(meta)data are assigned a globally unique and persistent identifier*
  Both FISH GUPRIs, for instance, data (see section 2.2), and RTS GUPRIs, for metadata and onotology elements, are resolvable, unique, and persistent.
- **F2.** *data are described with rich metadata (defined by R1 below)*
  Primary digital objects are described with the resource (aka dataset) records by using properties “has status”, “is created by”, “is created on”, “is lastly modified by”, “is lastly modified on”, and “is locked by source system” (see lines 22-40 in listing 1)
- **F3.** *metadata clearly and explicitly include the identifier of the data it describes*
  Primary digital objects and the resource records are linked with the property “is primary topic of” (see line 22 in listing 1).
- **F4.** *(meta)data are registered or indexed in a searchable resource*
  FAIR Data APIs dedicated to search make this principle fulfilled. Responses of the APIs are serialized in JSON-LD and comply with the application models.
- **A1**, **A1.1**, **A1.2** *(meta)data are retrievable by their identifier using a standardized communications protocol*, *the protocol is open, free, and universally implementable*, and *the protocol allows for an authentication and authorization procedure, where necessary*
  HTTPs is used as a communication protocol, and usage of the API can be restricted to groups and users.
- **A2.** *metadata are accessible, even when the data are no longer available*
  FISH data and metadata are stored in separate databases and can be accessed by different APIs.

- **I1.** *(meta)data use a formal, accessible, shared and broadly applicable language for knowledge representation*
  The application models describe metadata and are expressed in the RDFS extended with some OWL elements (see section 2.1); FISH FAIR Data APIs' responses—that are serialized in JSON-LD and are application models compliant (see section 3.1)– guarantee that I1 is fulfilled for FISH.
- **I2.** *(meta)data use vocabularies that follow FAIR principles*
  RTS controls the application models, which strive to be FAIR even though RTS is not open. FISH FAIR Data APIs are compliant with the FISH application models (see section 2.1), which makes FISH (meta)data machine-actionable.
- **I3.** *(meta)data include qualified references to other (meta)data*
  The rich catalog of cross-references that RTS provides connects the in-house digital objects to the reference ontology ones.
- **R1**, **R1.1**, **R1.2**, **R1.3** *meta(data) are richly described with a plurality of accurate and relevant attributes*, *(meta)data are released with a clear and accessible data usage license*, *(meta)data are associated with detailed provenance*, and *(meta)data meet domain-relevant community standards*
  They are fulfilled by re-using community standards (e.g DCAT, SKOS) as much as possible and including provenance data in the application models. As FISH is an internal system, licensing is not applicable.

5. Conclusion

The FAIR Guiding Principles are essential to maximizing the value of data, especially in a data-heavy and complex landscape like the life science industry. We have described an implementation of an API designed to expose FAIR data using ontologies and JSON-LD to reveal the meaning and context of digital objects. The shown example, the FAIR in vivo data sharing platform, is a system in production that takes full advantage of semantic technologies.

The sheer size and complexity of a company like Roche and the "in silo" implementation of numerous systems make data fragmentation inevitable. We see FAIR Data APIs as a way to have a semantic layer on top of data silos, enabling semantic data and system integration. The FAIR Guiding Principles should be considered for data, processes, and technical solutions like API design to accomplish the vision of a company-wide integrated data ecosystem. In a company with a data-centric culture digital objects, their exchange and governance must be treated as first-class assets.

We discussed FAIR data APIs in this paper. However, we did not discuss the FAIRness of the API itself. We believe there is a difference between a FAIR Data API and a FAIR API. We propose that a FAIR API has to contain 1) FAIR metadata about itself (e.g., owned by, contact, is about, description), 2) must not but should contain FAIR data, and 3) must contain HATEOAS links to support machine interpretability: given a root endpoint, one should be able to explore the API without prior knowledge. The semantic meaning of keys and values in a FAIR API should be captured, e.g., via JSON-LD context.

Acknowledgments

We would like to thank Joachim Rupp and Eng. Krzysztof Majewski for their insightful comments and suggestions while working on the FAIR Data API concept. We would also like to extend our thanks to the project teams of RTS and FISH, particularly Christian Blumenröhr, for their professional approach and openness to implementing the FAIR Guiding Principles.

References

[1] Maximizing data value for biopharma through FAIR and quality implementation: FAIR plus Q, Drug Discovery Today 27 (2022) 1441–1447.

[2] H. van Vlijmen, A. Mons, A. Waalkens, et al., The need of industry to go FAIR, Data Intelligence 2 (2020) 276 – 284.

[3] J. Wise, A. G. de Barron, A. Splendiani, et al., Implementation and relevance of FAIR data principles in biopharmaceutical R&D, Drug Discovery Today 24 (2019) 933–938.

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

[5] M. D. Wilkinson, M. Dumontier, et al., Evaluating FAIR Maturity Through a Scalable, Automated, Community-Governed Framework, Scientific Data 6 (2019).

[6] FAIR Data Maturity Model: specification and guidelines, 2020. URL: https://www.rd-alliance.org/group/fair-data-maturity-model-wg/outcomes/fair-data-maturity-model-specification-and-guidelines-0.

[7] The Pistoia Alliance FAIR Toolkit, 2022. URL: https://fairtoolkit.pistoiaalliance.org.