FDO-Ops: A Model for Machine-Actionable FAIR Digital Objects to Enhance Interoperability

Nicolas Blumenröhr*, Philipp Ost, Rossella Aversa, Felix Kraus, Maximilian Inckmann, Achim Streit

Karlsruhe Institute of Technology, Scientific Computing Center Kaiserstraße 12, 76131 Karlsruhe, Germany

Keywords: FAIR Digital Objects; Metadata management; Machine actionability; FAIR principles; Interoperability; Object-oriented; Entity-relationship

Citation: Blumenröhr N., Ost P., Aversa R., et al.: FDO-Ops: A Model for Machine-Actionable FAIR Digital Objects to Enhance Interoperability. Data Intelligence, 2026, Vol. 8, Art. No.: 20250345. DOI: https://doi.org/10.3724/2096-7004.di.2025.0345

ABSTRACT

Interoperability is the key to interacting with data across systems and technologies, thereby facilitating knowledge interpretation in different scientific domains. The FAIR Principles provide guidelines for this aspect, and Knowledge Organization Systems with associated technologies provide practical solutions. However, interoperability is rarely fully achieved in reality due to limited capabilities with respect to machine actionability of these technologies.

FAIR Digital Objects, when associated with operations, can fill this gap. We introduce FDO-Ops, a model and framework that includes a specification of operations and a procedure for associating and executing them on FAIR Digital Objects. On this basis, we derive a generic description of the machine-actionable entity a FAIR Digital Object constitutes. To demonstrate our approach, we provide a prototype implementation of FDO-Ops that we test with several use cases with data from the domains of energy research and digital humanities. Our evaluation indicates that machine-actionable FAIR Digital Objects based on FDO-Ops, used in conjunction with existing technologies, greatly enhance the interoperability of the digital resources they represent at the technical and syntactic levels. In addition, they offer perspectives for improving semantic, legal, and organizational interoperability.

1. INTRODUCTION

A major portion of scientific work consists not only of analyzing the state-of-the-art literature but also in assessing data as a form of resource. Administrative information retrieval (e.g., usage licenses), content

*Corresponding author: Nicolas Blumenröhr (E-mail: nicolas.blumenroehr@kit.edu).

assessment (e.g., interpretation of associated vocabularies), and preprocessing (e.g., format conversion) are general prerequisites for the reusability of data resources. Collecting data across domains for machine learning applications faces similar challenges and therefore is a prime example for the relevance of these tasks [1]. However, due to the wide variety of data and metadata formats, as well as the lack of interoperability, the acquisition, inspection, and comprehension of data resources can be extremely laborious [2].

Knowledge Organization Systems (KOSs) based on standardized metadata schemas [3], Web service specifications such as (RESTful) Web Application Programming Interfaces (APIs) [4], and Semantic Web methods provide a way to unify information management and access in repositories [5-6]. They enable improved information assessment for humans and a certain degree of machine-interpretability. However, their alignment to enable interoperability is difficult due to the lack of a universal solution, the diversity of standards, and the evolving nature of technologies. Therefore, achieving interoperability remains a challenging task, particularly across domains [7]. The FAIR Principles [8] provide guidelines for making data interoperable on the basis of findability and accessibility, ultimately resulting in improved reusability. The FAIR Principles align with the vision of the Semantic Web by promoting data resources that are shared, discoverable, and machine-readable, through the addition of semantics to Linked Data [9-11].

Although the FAIR principles provide guidelines on data management practices, the concept of FAIR Digital Objects (FDOs) offers an implementation of them. FDOs focus on the management of individual data resources by integrating principles also used in Linked Data, i.e., resource linkage on the Web, and Object-Oriented Programming (OOP), i.e., abstraction, encapsulation, and typing through objects [12-13]. FDOs constitute an approach to a harmonized representation of diverse artifacts. They abstract the technical, syntactical, and semantic complexity of these artifacts in a thoroughly typed, technology-, domain-, and use-case-independent format. However, the FDO core data model only fulfills the aspects of findability and accessibility through the enrichment of persistent information for the data they represent. Interoperability and reusability are less pronounced due to the lack of a mechanism for the integration of operations to make these objects machine-actionable [14]. Based on their high-level representation and technology-agnostic character, FDOs are suitable to utilize a variety of available operation technologies, e.g. Web APIs and scripts. To achieve technology- and domain interoperability, operations need to be represented uniformly to be identifiable and applicable by digital clients, regardless of their individual focus. This needs to be supported by an FDO infrastructure, i.e., base services, communication protocols, and advanced tooling [15].

We introduce FDO-Ops, a process model extending the FDO core data model which specifies operations and enables their association and execution. We provide a prototype implementation for this model using the existing FDO infrastructure, advanced tooling, and the Digital Object Interface Protocol (DOIP) to demonstrate its usability based on a set of use cases with data from the domains of energy research and digital humanities. We evaluate the applicability of this model, which interoperability levels are improved in certain ways thanks to it, and its compatibility with existing KOSs and associated technologies. We argue that FDO-Ops enables a machine-actionable framework for FDOs to yield an

interoperable data space of digital resources including data and executable software. This in turn enables systematic automation of data processing tasks, ultimately contributing to improved data reuse.

2. RELATED WORK AND PROBLEM DESCRIPTION

2.1 Interoperability in the Context of FAIR

In the FAIR Principles, the term interoperability is defined as "the ability of data or tools from non-cooperating resources to integrate or work together with minimal effort" [8]. In addition to the conditions for data resources being findable and accessible, "data becomes interoperable when it is machine-actionable" [2] and the reusability of data can be understood as a result of interoperable systems.

In more detail, there exist different categories, often referred to as levels of interoperability [16]. A set of FAIR-compliant recommendations to achieve them is provided by the fourlevel (technical, semantic/syntactic, legal, organizational) interoperability model of the European Interoperability Framework (EIF), also applied by the European Open Science Cloud (EOSC) Interoperability Framework (EOSC IF) [17]. Briefly summarized, technical interoperability facilitates interaction at the application level within the infrastructure; semantic and syntactic interoperability concern the shared understanding of data elements and their meaning, and the ability of systems to communicate with each other through data formats and communication protocols, respectively; legal interoperability ensures collaboration between organizations operating under various legal frameworks, policies, and strategies; organizational interoperability focuses on aligning business processes. The EIF thereby connects to other interoperability measures such as the Levels of Conceptual Interoperability Model (LCIM) which takes a more in-depth approach to evaluate interoperability between two systems [18], or the technological, social-technical and cross-cutting types of interoperability described in the tertiary study of Systems Interoperability Types [19]. In [15], the authors propose a more granular model of semantic interoperability, i.e., terminological and propositional interoperability, to achieve an Internet of FAIR Data and Services (IDFS), which they also incorporate as extension to the FAIR Principles criteria and propose to realize it with FDOs using dedicated FAIR Services.

2.2 Existing Concepts and Technologies for Interoperable Systems

In this subsection, we describe the state-of-the-art concepts and technologies that contribute to interoperable systems in the spirit of FAIR, highlighting their focus, strengths, and limitations. Taken together, existing KOSs and executable specifications enhance different (especially semantic and syntactic) interoperability levels but are technology dependent, often tailored to a particular field and thus not widely adopted, have a large technical overhead and a limited scope with respect to machine actionability on data resources and their metadata.

2.2.1 Knowledge Organization Systems

Metadata standards are universally recognized vocabulary-based schemas that ensure uniform data description, semantics, and structure across (persistent) storage systems, including general-purpose and domain-specific repositories [2, 20]. They are often used in conjunction with data catalogs and metadata management systems, such as Apache Atlas or Amundsen. Although many repositories adopt accepted standards, the diversity of resource types and communities comes with the frequent use of specialized schemas. To facilitate interoperability, crosswalks between schemas can be defined. This typically requires deeper domain knowledge and is often not trivial due to contextual ambiguities in the content [21].

Ontologies provide a formal structure for defining entities, relationships, and rules, as well as a way to model complex relationships between entities, especially enabling semantic interoperability and integration across diverse digital resources [22]. They are the basis for Knowledge Graphs and Linked Data Platforms (LDP), and are also used to describe the properties and capabilities of Web Services in machine-interpretable form, e.g., the OWL-S ontology [23], and the Web Service Modeling Ontology (WSMO) [24]. However, their adoption and applicability are often limited by their complexity, required granularity of domain knowledge, and high maintenance costs [25].

2.2.2 Executable Specifications

Developments such as LDScript [26], ActiveRDF [27], or the FAIR Data Point [2] specify the access to Linked Data, enhancing integration of and interoperability within Resource Description Framework (RDF)-based systems. Their RDF-centric usability and focus on Semantic Web technologies make it difficult to work with other data formats, thus limiting broader interoperability in diverse, multi-domain data ecosystems.

Web API specifications, such as the OpenAPI Specification (OAS)[①], RDF-based OAS [28], HATEOAS [29], and Hydra [30], as well as command-line tool interfaces and workflow specifications, such as the Common Workflow Language (CWL) [31], enhance interoperability through standardization, cross-platform portability, and multi-engine support. Nonetheless, they receive their input at runtime, requiring different carrier technologies to enable machine-actionable processes on data resources.

2.2.3 FAIR Digital Objects

The concept of Digital Objects was first introduced by [32] and later identified as consistent with the requirements of the FAIR Principles, providing a strategy for their implementation. This gave rise to the concept of FAIR Digital Object [33], with the potential to facilitate a wider abstraction of interoperability in data management [13]. The concept can be understood as a holistic high-level approach to leverage and combine the capabilities of existing and widely adopted solutions.

A formalized FDO core data model based on the conceptual model of Digital Objects [34] and the Research Data Alliance (RDA) recommendations on Persistent Identifier (PID) Kernel Information [35]

① https://www.openapis.org/.

was introduced by [36]. This details how FDOs serve as abstract representations for various digital resources that are requested using DOIP [37]. However, without the association of operations, FDOs are not inherently machine-actionable, which is crucial to reaching interoperability at the level of the digital resources they represent. In the same work, operations are closely associated with an underlying type system based on PID Information Types (PITs) [38-39] making them machine-interpretable, which is the prerequisite for machine-actionability [40]. On this basis, different typing mechanisms that can be used to associate a set of operations with an FDO are described in [41]. The work of [42] proposes an advanced typing model for FDOs on the basis of PITs, but focuses on modeling associations between types and operations purely within a Data Type Registry (DTR) (s. [43]), and does not provide a solution for the execution of operations. Related implementations of FDOs either do not specify a type system-based association of operations beyond the generic Create, Read, Update, Delete (CRUD) operations or lack a representation for operations to make them reusable across data storage and processing systems (e.g. [44-46]). Other modeling approaches for FDOs such as the FDO Framework (FDOF) [47], the EOSC Interoperability Framework (EOSC IF) [17], or webby FDOs implemented using Research Object Crates and Signposting [48] face a similar problem, as also pointed out in [14].

3. FDO-OPS: MACHINE-ACTIONABLE FAIR DIGITAL OBJECTS

In this section, we describe the FDO-Ops model that specifies the different FDO operations, how they are applied in the FDO workflow, and how they are associated and executed on FDOs based on a type system.

3.1 Operation Categories

Depending on their role, we classify operations into three categories: *Management-, Information Record-,* and *Data Resource Operations*, which are described in more detail in the following subsections.

3.1.1 Management Operations

In general, these operations act on behalf of a particular service that manages a set of FDOs and is used to provide a basic infrastructure to maintain an FDO space. Therefore, they implement the core mechanisms defined in the FDO core data model, including basic CRUD operations to validate FDOs, and as a way to utilize operations from other categories.

3.1.2 Information Record Operations

These operations use and process the metadata contained in the information record of an FDO on the basis of a type system. They do not access the represented bit sequence and do not modify the content of the information record at the entity level, which is done by *Management Operations*. However, they may process any set of key-value pairs in the record to gain insights into the characteristics of the represented bit sequence; for example, by assessing an FDO's relationships to other Web entities including other FDOs, and take decisions accordingly.

3.1.3 Data Resource Operations

These operations act on the bit sequence of the data resource that is represented by the FDO, also taking into account the metadata in the information record, with data access information as a minimum. A copy of the bit sequence itself may be manipulated by these operations, e.g., in the frame of a given application case. In case a permanent manipulation of the bit sequence is performed that changes the characteristics described in the information record, it must be accompanied by a *Management Operation* to properly modify its information record, i.e., updating the entire FDO.

3.2 Operation Phases

We propose applying these operation categories sequentially in three phases. In phase I, there must be a way to initially search and filter for FDOs. In this phase, FDOs are aggregated by the text-based content of their information records without using any types. Conceptually, this phase involves only the category of *Management Operations*.

In phase II, the FDOs retrieved in phase I are further processed. The operations are applied to the typed metadata attributes contained in the information record and provide information in addition to the purely text-based content assessment in phase I. This may be the type-dependent evaluation of attribute values that reference related FDO entities, which can be retrieved and also processed. In this phase, both *Management* and *Information Record Operations* are involved.

Finally, in phase III, operations are applied not only to the contents of the information record but also to the bit sequence of the represented data resource. The typed metadata is used to initiate an operation that acts directly on the bit sequence; this comprises the retrieval of the original data resource, a modified version of it, or any result that is yielded from its processing. In this phase, all categories of *Management-*, *Information Record-* and *Data Resource Operations* are involved.

The relations between these three phases and the three categories of operation are summarized in Table 1. Whilst Management Operations must be implemented directly in a service that aims to manage FDOs, Information Record- and Data Resource Operations could, in principle, be implemented anywhere and made available to the client through the management service and FDO types.

3.3 Associating FDOs and Operations

We assume that FDOs are modeled according to the FDO core data model introduced by [36], providing a precise definition accompanied by a set of assertions for how an FDO entity is created and structured analogously to object instantiation as known from OOP. The comparative analysis conducted in [41] provides insights into the trade-offs between the three typing mechanisms defined to associate FDOs and operations, i.e., Record-, Profile-, and Attribute Typing. The work of [42] supplements these typing mechanisms by introducing type inheritance for the data types involved in each mechanism. In the context of interoperability challenges faced by domainspecific KOSs, we evaluate their traits as follows:

Table 1. An overview of the three operation phases for FDOs and their relation to the three categories of operations.

Operation categories use in phases	Management Operations	Information Record Operations	Data Resource Operations
Phase I	Retrieve entities, possibly based on the record content without using the metadata types.	Not used.	
Phase II	Read, interpret, and act on record content using the metadata types.		Not used.
Phase III	Read, interpret, and act on record content and the data resource considering the metadata types and the bit sequence.		

- Record Typing: maximizes simplicity and efficiency but fixes the operation set to FDOs at instantiation, which limits their application to the frame of their domain and original (re)use cases. KOSs that are integrated within, or represented by these FDOs are only machine-actionable in the frame of their original scope and interoperable with systems that know how to interpret the results of these operations.
- Profile Typing: improves reuse and governance by binding operations to profile classes. Combined with type inheritance, this supports family-wide behavior, yet it remains profilebound and tends to proliferate profiles when KOS and non-KOS facets must be combined per object. This risks that again only domain-specific operations are available for a given FDO that integrates with, or represents KOSs.
- Attribute Typing: operations declare applicability over typed attributes that can appear across profiles and thus in FDOs independent of their original domain. Operations become executable whenever those typed attributes or their inherited types (assuming subtyping modeled by the corresponding DTR) are present in an FDO, independent of the object's profile lineage. This allows for dynamic association and reuse of operations that can process KOSs contents when the corresponding FDOs provide the required kernel information. Type inheritance, when using a corresponding typing model, thereby strengthens reuse by lifting operations and constraints along data-type hierarchies. However, this complements Attribute Typing but does not remove the need for runtime composition across heterogeneous KOS. Furthermore, role-based typing can be realized when proper authentication and authorization related attributes are integrated within an FDO's information record.

Given its emphasis on dynamic, flexible and versatile association of FDOs and operations, we chose the Attribute Typing association model. This makes use of the typed elements that appear in an FDO information record to specify *requirements* for the association. Types may be structured according to the originally defined PITs, or to advanced inheritance hierarchies. This depends on the maturity of the registry model and is independent of the following, why we stick with the generic notion of PITs.

To specify the associations in alignment with the assertions of the core model, we have to recap some terms and define new ones for the operation categories introduced earlier.

Definition 1 (Association Requirements). Let O be the set of all FDO operations, $O_m \subseteq O$ be the set of *Management Operations*, and $O_{rr} \subseteq O$ be the set of all *Information Record Operations* and *Data Resource Operations*. Denote F as the set of all existing FDOs, and R_f as the set of the information record of an FDO $f \in F$. Let $R_f \supseteq A_f = K \times V$ be a set of typed attribute key-value pairs in an FDO information record, with K being the set of all existing keys of typed attributes, and V being the set of all validated possible values for any $k \in K$. To formalize the associations, we define the set of all R_f as $\mathcal{R}\{R_f \mid f \in F\}$ and the function

$$Associated : O \times \mathcal{R} \rightarrow \{\text{true},\ \text{false}\}$$

that assesses whether an FDO is associated with an operation based on the elements of the information record.

By definition, *Management Operations* are associated with any FDO, irrespective of its type or content, if they do not fall under any specific negation condition $\mathcal{N}_n(o_m)$ (s. Section 3.6), with $n \in \mathbb{N}$, for which it holds that:

$$\forall o_m \in O_m\ \forall f \in F : \neg[\mathcal{N}_1(o_m) \vee \ldots \vee \mathcal{N}_n(o_m)] Associated(o_m,\ R_f) = true \tag{1}$$

Assuming that *Information Record-* and *Data Resource Operations* are applicable to a specific FDO type, according to its PITs, the following applies:

$$\forall o_{rr} \in O_{rr}\ \exists f \in F\ \exists a_f \in A_f : Associated(o_{rr},\ R_f;\ a_f) = true \tag{2}$$

Note that Equation (2) also holds for management operations $o_m \in O_m$ if at least one of $\mathcal{N}_i(o_m),\ 1 \leq i \leq n,$ is true. Therefore, Equation (2) formalizes the association condition for any operation o (including o_m if negation conditions apply) based on the presence of specific typed attributes a_f. The notation o_{rr} is used here for clarity within the immediate context of discussing non-management operations.

The assessment procedure of the *Associated* function is based on the Attribute Typing mechanism, where the *requirements* of an association are given by an array C_o that is specified for an operation $o \in O$ using one or more *condition* arrays $c_o \in C_o$. These *conditions* consider the presence of one or more key-value pairs using the variable x_o. While each *condition* must specify at least one key, the specification of a value for this key is optional, such that:

$$x_o \in \{(k,\ v) \mid k \in K,\ v \in V \bigcup \{\varnothing\}\} \tag{3}$$

Each *condition* c_o is then an array of tuples in the *requirements* array C_o for which the following holds:

$$\forall c_o \in C_o\ \exists x_o \in c_o : x_o \neq \varnothing \tag{4}$$

When assessing the association between an operation and an FDO, if $dim(C_o) > 1$ and $dim(c_o) > 1$, the elements in the arrays C_o and c_o are interpreted as arguments for the disjunction and conjunction terms, respectively, as described in Algorithm 1. This approach seamlessly aligns with the given data structure of an FDO information record consisting of key-value pairs. A generic minimal working example is given in Appendix A.

Algorithm 1. The logic of the function Associated for assessment of the association between an operation and its targeted FDO.

Input: o and R_f

Output: Boolean

if $(o \in O_m)$ ***and*** $(\neg[\mathcal{N}_1(o_m) \vee \ldots \vee \mathcal{N}_n(o_m)])$ **then**
    return TRUE;
// Check if any *condition* c_o of the *requirements* C_o is fulfilled (disjunctive term)
for $c_o \in C_o$
    $r \leftarrow$ TRUE;
    // Check if each *condition* in c_o is fulfilled (conjunctive term)
    // $x_o[k]$ and $x_o[v]$ denote the key and value of x_o as defined in Equation (3)
    for $x_o \in c_o$ **do**
        if value of $x_o[k] \notin R_f$ **or** $(x_o[v] \neq \varnothing$ ***and*** $(x_o[k] \in R_f) \neq x_o[v])$ **then**
            $r \leftarrow$ FALSE;
            Break;
    // Return as soon as one of the *conditions* in C_o is evaluated to TRUE
    if r = TRUE **then**
        return r;
return FALSE;

3.4 Modeling Operation FDOs

In order to implement the approach for associating operations via Attribute Typing (s. Algorithm 1), operations must be modeled using a concrete data structure. Following the idea of FDOs to abstract all kinds of digital resources, we can also represent an operation as an FDO, that we call *Operation FDO*, which is applied to *Target FDOs*. In addition to the association *requirements* (s. Definition 1), an *Operation FDO* can also specify the underlying technology used for its implementation via an *execution protocol*

(s. Section 3.5) and the type of output it generates. Furthermore, an *Operation FDO* is itself operable by other operations and can therefore, depending on the context of the application, also be considered a *Target FDO*. Since *Management Operations* are crucial for interactions with the FDO ecosystem, they are already required to create any *Operation FDO*. *Management Operations* are applicable to any FDO by definition (s. Equation (1)) and are executed by the individual service. We therefore apply the notion of *Operation FDOs* only to the categories of *Information Record Operations* and *Data Resource Operations*. Consequently, an *execution protocol* is also only present in operations that fall under these categories.

To store the associations of *Target FDOs* and *Operation FDOs* in a graph structure for efficient querying, we can represent them as triples in a graph model. From this point of view, FDOs are (Web) entities that are identified by their PIDs and related to other FDO entities by their typed attributes that specify these relationships semantically, similar to RDF. As described in [36], PID-triples can be directly derived from those typed attributes in an FDO's information record that reference other FDOs. In the following, we derive a similar constellation for Target-Operation FDO relationships. This also addresses the drawbacks of less efficient and complex Attribute Typing which is highly cost intensive when FDO-operation associations must be dynamically determined each time upon request. Representing the associations as a simplified graph data model enables resource-efficient storage in a graph database.

Definition 2 (Graph Models). Let I be the set of all existing PIDs, $\tilde{F} \subset I$ the set of PIDs that identify a set of FDOs, $\tilde{O} \subset \tilde{F}$ the set of PIDs that identify a set of *Operation FDOs*, and $\tilde{K} \subset I$ the set of all PIDs of typed attributes (PITs). Let $i_1 \in \tilde{F}$ be the PID of a *Target FDO*, $i_2 \in \tilde{K}$ the PID of a typed attribute in the information record of the FDO associated with i_1, and $i_3 \in \tilde{K}$ the PID of a typed attribute defining an association *condition* (s. Equation (4)). Let $i_4 \in \tilde{O}$ be the PID of an Operation FDO, which contains a typed attribute associated with i_3. Let L = {isOperationFor, hasOperation} be a set of edge labels. Denote $Q \subseteq \tilde{F} \times \tilde{K} \times \tilde{K} \times \tilde{O}$ as the set of all existing PID-quadruples and $T \subseteq (\tilde{F} \bigcup \tilde{O}) \times L \times (\tilde{F} \bigcup \tilde{O})$ as the set of all existing triples from the projection of valid quadruples to (target,operation) pairs. Let G_H be a PID hypergraph that can be transformed into a labeled PID digraph G_D.

G_H is defined as:

$$G_H := (V_H, E_H) \tag{5}$$

where $V_H = \tilde{F} \bigcup \tilde{K} \bigcup \tilde{O}$ is the set of vertices and $E_H = \{\{i_1, i_2, i_3, i_4\} \mid (i_1, i_2, i_3, i_4) \in Q\}$ the set of edges.

In order to construct the graph G, consider the projection of valid quadruples to (target, operation) pairs in the set T given by:

$$T = \{(i_1, \text{hasOperation}, i_4), (i_4, \text{isOperationFor}, i_1) \mid \exists i_2, i_3 : (i_1, i_2, i_3, i_4) \in Q\}$$

which collapses duplicates from different i_2, i_3 over quadruples that have the same i_1, i_4. The corresponding elements can be derived by the output of the function *Associated* in Algorithm 1.

G_D is defined as:

$$G_D := (V_D,\ E_D) \tag{6}$$

where $V_D = \tilde{F} \bigcup \tilde{O}$ is the set of vertices and $E_D = T \subseteq V_D \times L \times V_D$ the set of labeled edges.

This allows us to represent FDO entities and their relationships in the following way:

$$\text{Target FDO} \underset{\text{isOperationFor}}{\overset{\text{hasOperation}}{\rightleftarrows}} \text{Operation FDO} \tag{7}$$

3.5 Executing FDO Operations

When an associated operation is applied to an FDO, its execution must be triggered with the technical details. For this, we introduce the notion of an *execution protocol*, which contains information on how the specific implementation technology of the operation is executed. Therefore, it serves as a harmonized representation of the technical execution details of different technologies that implement an operation. Similarly to the association *requirements*, the protocol must specify which typed attributes in the *Target FDO* information record contain the content for the protocol parameters. For example, if the protocol specifies an operation that evaluates the semantic description of the data resource based on a *topic* attribute, then it expects a value of the corresponding PIT. The actual values must be therefore extracted from the information record of the *Target FDO*.

Definition 3 (Execution Protocol). Let $F_o \subseteq F$ be the set of all *Operation FDOs*, $F_t \subseteq F$ be the set of all *Target FDOs*, $e_{f_o} \in A_{f_o}$ be the execution protocol of an $f_o \in F_o$ that represents an operation $o \in O$. Since $e_{f_o} = (k_e,\ v_e)$, let $k_e \in K$ be an *execution_protocol_type* which is represented by the PID of a complex PID Information Type, and $v_e \in V$ a protocol parameter object denoted as set P_e that consists of key-value pairs of the form

$$(\textit{parameter\_type},\ (\textit{parameter\_key},\ \textit{parameter\_value},\ \textit{merge\_mul\_entries})) \in K \times V.$$

Let *parameter_type* be the PID of a complex PIT that specifies this protocol parameter, and (*parameter_key*, *parameter_value*, *merge_mul_entries*) the corresponding triple that contains the values for this parameter. Assuming $\textit{Associated}(o,\ R_{f_t}) = \textit{true}$ (s. Equation (2)) for a *Target FDO* $f_t \in F_t$, a protocol parameter $p_e \in P_e$ may specify an attribute $(k,\ v) \in A_{f_t} \subseteq R_{f_t}$ that contains input for the operation's execution, which is given by a mapping function $m : P_e \to K$ such that:

$$\exists p_e \in P_e\ \exists (k,\ v) \in A_{f_t} : m(p_e) = k$$

Definition 4 (Execution Protocol Parameters). More specifically, *parameter_key* is a string that specifies the key that is used in the actual execution for the value to be mapped, and *parameter_value* either specifies the PIT of the typed attribute in the *Target FDO* information record which contains this value, static values, information that must be supplemented by client input, or a sub-execution protocol, where combinations are possible; *merge_mul_entries* is a tuple that specifies if and how multiple entries of *parameter_value*, in case it references the key k of an element $(k,\ v) \in A_{f_t}$, are merged on the basis of a

specific (*prefix, delimiter, suffix*) pattern. The merged values are then used as part of the same request during the execution process. In case they are not supposed to be merged, each value is used in a separate request. This is necessary because a *Target FDO* may have repeated entries for the same typed attribute key specified in the *parameter_value*. The structure is defined as:

$$merge\_mul\_entries = \begin{cases} (i_1,\ i_2) & \text{if } parameter\_value \text{ references PIT,} \\ \varnothing & \text{if } parameter\_value \text{ does not reference PIT.} \end{cases}$$

For the first case, *merge_mul_entries* is specified as:

$$merge\_mul\_entries = \{(i_1,\ i_2) |\quad i_1 \in \{\text{true, false}\},\ i_2 = \begin{cases} (\text{prefix, delimiter, suffix}) & \text{if } i_1 = \text{true,} \\ \varnothing & \text{if } i_1 = \text{false.} \end{cases}\}$$

The mapping function *m*, as employed in Algorithm 2, is used to fill all parameters of the *execution protocol* of an *Operation FDO* with the required information from a *Target FDO* to generate the operation requests, and transfers them into an *execution_map* that is ready to be further processed by a software component. A minimal generic example for this mapping procedure is given in Appendix A.

Algorithm 2. Mapping of the execution-protocol e_{f_o} and *Target FDO* information record R_{f_t} to an *execution_map*.

Input: e_{f_o} and R_{f_t}

Output: execution_map

FnFunction: Mapping (e_{f_o}, R_{f_t}):

    // Initialize the execution map with the first request index and main execution protocol type.
    exe_prot_type ← Get or Infer Protocol Type (e_{f_o})
    exe_map ← {request_index 1: {exe_prot_type: {∅}}}
    for $p_e \in P_e$ **do**
        par_type, par_key, par_val, mrg_mul_entries ← $p_e[0]$, $p_e[1][0]$, $p_e[1][1]$, $p_e[1][2]$
        // Set flag to merge values of a parameter type with client input when adding to the execution map.
        if Client_Input_Provided (*par type*) **then**
            client_val ← Get_Client_Input (*par_type*)
            Set_Flag_for_Client_Input (*par_type, client_val*)
        // If the parameter value specifies a sub-execution protocol parameter array, map and transfer it recursively.
        if Is_Sub_Execution_Protocol (*par_val*) **then**

Algorithm 2. *Continued.*

```
10        sub_exe_map ← Mapping (par_val, R_fi
11        par_val ← sub_exe_map
12        exe_map.Add_to_Requests (exe_prot_type, par_type, par_key, par_val)
          // For a parameter value referencing a pit that is present in the Target fdo record, get value(s) as array.
13      else if Has_PIT_Reference (par_val, R_fi) then
14        par_val ← Get_Information_Record_Values (par_val, R_fi)
          // Check if multiple referenced parameter values exists that have to be merged using the (prefix,
             delimiter, suffix) pattern, or are mapped into separate requests.
15        if length (par_val) > 1 then
16          if mrg_mul_entries[0] is true then
17            par_val ← Merge_Record_Values (par_val, mrg_mul_entries[1])
18            exe_map.Add_to_Requests (exe_prot_type,par_type,par_key,par_val)
19          else
              // First val element is always added to existing requests.
20            exe_map.Add_to_Requests (exe_prot_type, par_type, par_key, par_val[0])
21            for element from par_val[1] do
22              exe_map.Add_New_Request_Index (exe_prot_type, par_type, par_key, element)
23          else
24            exe_map.Add_to_Requests (exe_prot_type, par_type, par_key, par_val[0])
          // Add static values directly to the execution map.
25        else
26          exe_map.Add_to_Requests (exe_prot_type, par_type, par_key, par_val)
27    return exe_map
```

3.6 Process Model

Given the Attribute Typing mechanism to associate operations, the execution protocol, the modeling of *Operation FDOs*, and the entity relationship characteristics of FDOs, we can summarize the functionalities that are available through FDO-Ops in the Process Model shown in Figure 1 using a Unified Modeling Language (UML) activity diagram. From this point of view, an FDO can be understood as an entity with a set of rules that are associated with activities to evaluate how the FDO is to be applied in the given context. Each activity in the diagram constitutes a foundational operation for FDOs and is therefore performed by a *Management Operation* by definition, whilst not all existing *Management Operations* are necessarily included in this diagram. However, two activities, i.e., *Get Entity Relationships* and *Execute Execution Protocol on Target FDO*, are not always applicable, since not all FDOs are *Operation FDOs* or are related to other FDOs. We therefore define two negation conditions to avoid any contradiction with the formalism (s. Equation (1)):

- $\mathcal{N}_1(o_m)$: *the operation requires an FDO to be related to another FDO via a PIT*
- $\mathcal{N}_2(o_m)$: *the operation requires the presence of an execution protocol*

Using the negation functions $\mathcal{N}_1$ and $\mathcal{N}_2$ enables us to define these activities as *Management Operations*, avoiding conflicts for FDOs that do not meet the criteria.

The *Management Operations* described by the activities in the diagram, in conjunction with PITs, form the backbone of an FDO type system with the following rules: the PITs are syntactically validated against the content of the FDO information record, ensuring the FDO's integrity. In principle, an FDO can work as an operation or as the target for an operation. The particular operation functionality is abstracted on purpose, providing a technology and use case agnostic way of processing the information of an FDO and its represented digital resource. The core characteristics of an FDO are integrated in the process model, i.e., the retrieval of any non-FDO Web entities, including the FDO's bit sequence, that are referenced by a PIT to be further processed by a specific client, the retrieval of associated operations as defined in our model, and the retrieval of related FDO entities through PITs. One may argue that the latter activity could also be modeled as an *Operation FDO* and executed using the *Execute Execution Protcol on Target FDO* activity. However, since FDO interconnections are very common and have to be implemented by the FDO service that can request their information records, we found this to be worth a unique activity in the FDO-Ops process model.

4. MODEL IMPLEMENTATION

In this section, we describe the implementation of the FDO-Ops model using a prototype service architecture with DOIP to enable an FDO-Ops framework and demonstrate its usability considering two real-world use cases.

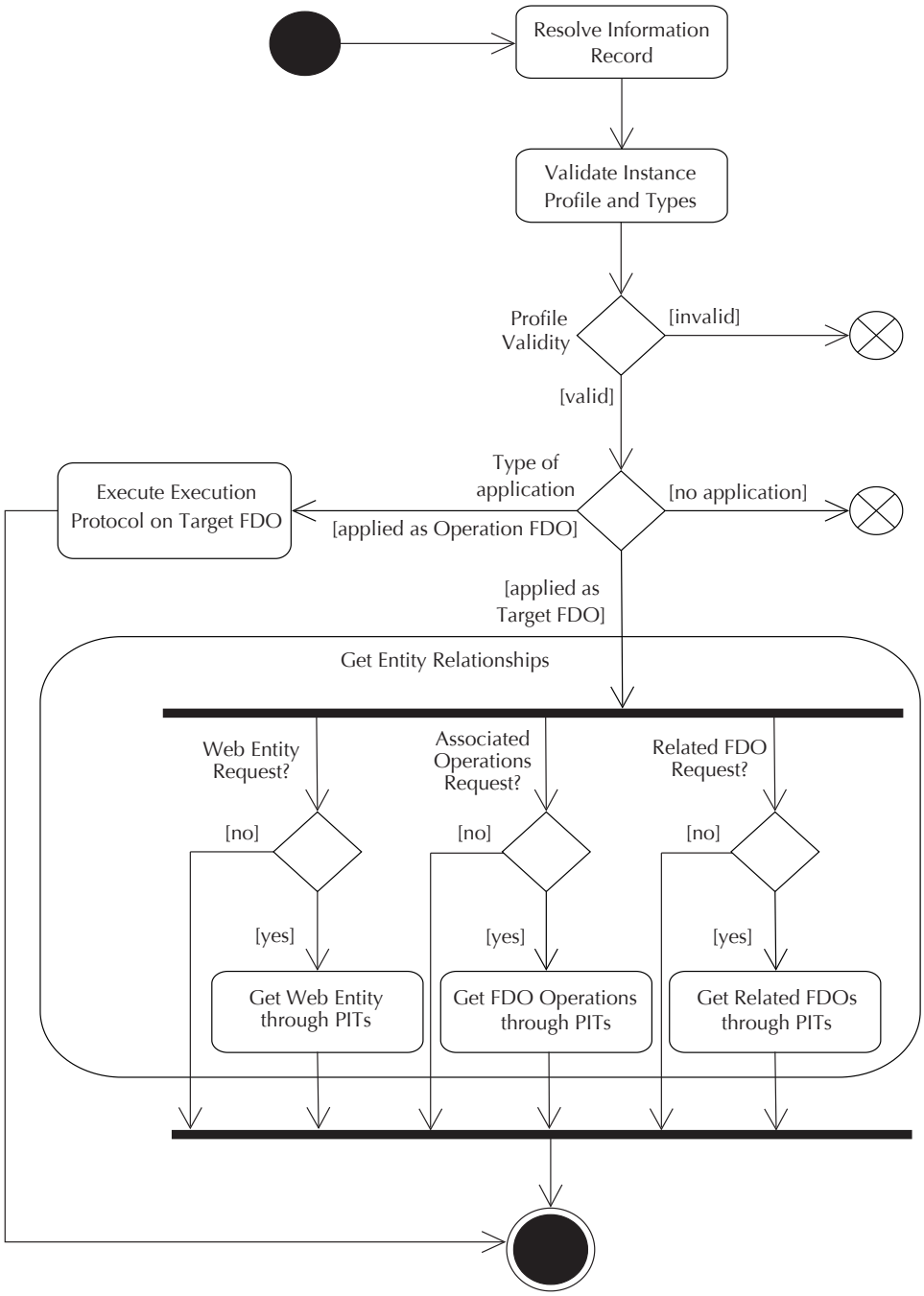

Figure 1. The FDO-Ops Process Model for a FAIR Digital Object.

4.1 Digital Object Interface Protocol

Each FDO must be initially operated by a service, be it for its instantiation, for retrieving its information record, or for applying any *Information Record*- and *Data Resource Operations*. Therefore, all operations must be requested via this service using a communication protocol which, ideally, is uniform across different FDO services, including a way to request FDO targetoperation associations as defined in our model.

DOIP fulfills these requirements; it is a communication protocol that describes how clients interact with digital objects. The request constellation is standardized using the identifiers of a target object, which may be the service itself, and the operation to be invoked on the target. Consequently, a DOIP-based FDO service is also suitable for inferring the association between FDOs and their operations, and mapping the *execution protocol* according to our model specifications. To execute one of the *Management*- or associated *Operation FDO* operations, their (P)ID must be provided using the "operationID" parameter, and the serviceID or *Target FDO* PID they are applied to must be provided using the "targetID" parameter. The payload, which may contain either the FDO's information record or a set of query parameters for the operation, is provided in JSON format using the "attributes.full=true" parameter.

DOIP can be directly implemented on the basis of TCP/IP, or as DOIP/HTTP[②], making it compatible with widely used Hypertext Transfer Protocol (HTTP) clients. In this way, all FDO operations can be requested uniformly using basic HTTP GET and POST operations. An example request for a DOIP/HTTP-based *Management Operation* (e.g. 0.DOIP/Op.GET_FDO) on an FDO (e.g. exemplaryPrefix/123) is:

```
GET /doip?operationId=0.DOIP/Op.GET_FDO
& targetId=exemplaryPrefix/123
Content-Type: application/json;charset=utf-8
```

and likewise, for any *Information Record*- or *Data Resource Operation* represented as *Operation FDO* (e.g. exemplaryPrefix/456):

```
GET /doip?operationId=exemplaryPrefix/456
& targetId=exemplaryPrefix/123
Content-Type: application/json;charset=utf-8
```

In case additional parameters must be forwarded, the HTTP POST request is used along with the payload:

```
POST /doip?operationId=exemplaryPrefix/456
& targetId=exemplaryPrefix/123&ibutes.full=true
Content-Type: application/json;charset=utf-8

{"exampleParameter": "exampleValue"}
```

② https://www.cordra.org/documentation/api/doip-api-for-http-clients.html.

4.2 Service Architecture

The service architecture we built to set up an FDO space testing environment is illustrated in Figure 2. It consists of a base infrastructure, an adapter for the Typed PID Maker (TPM) base service[3], called TPM Adapter, and an Executor, which are detailed in the following subsections. The entire implementation is available on GitHub (s. Section 7).

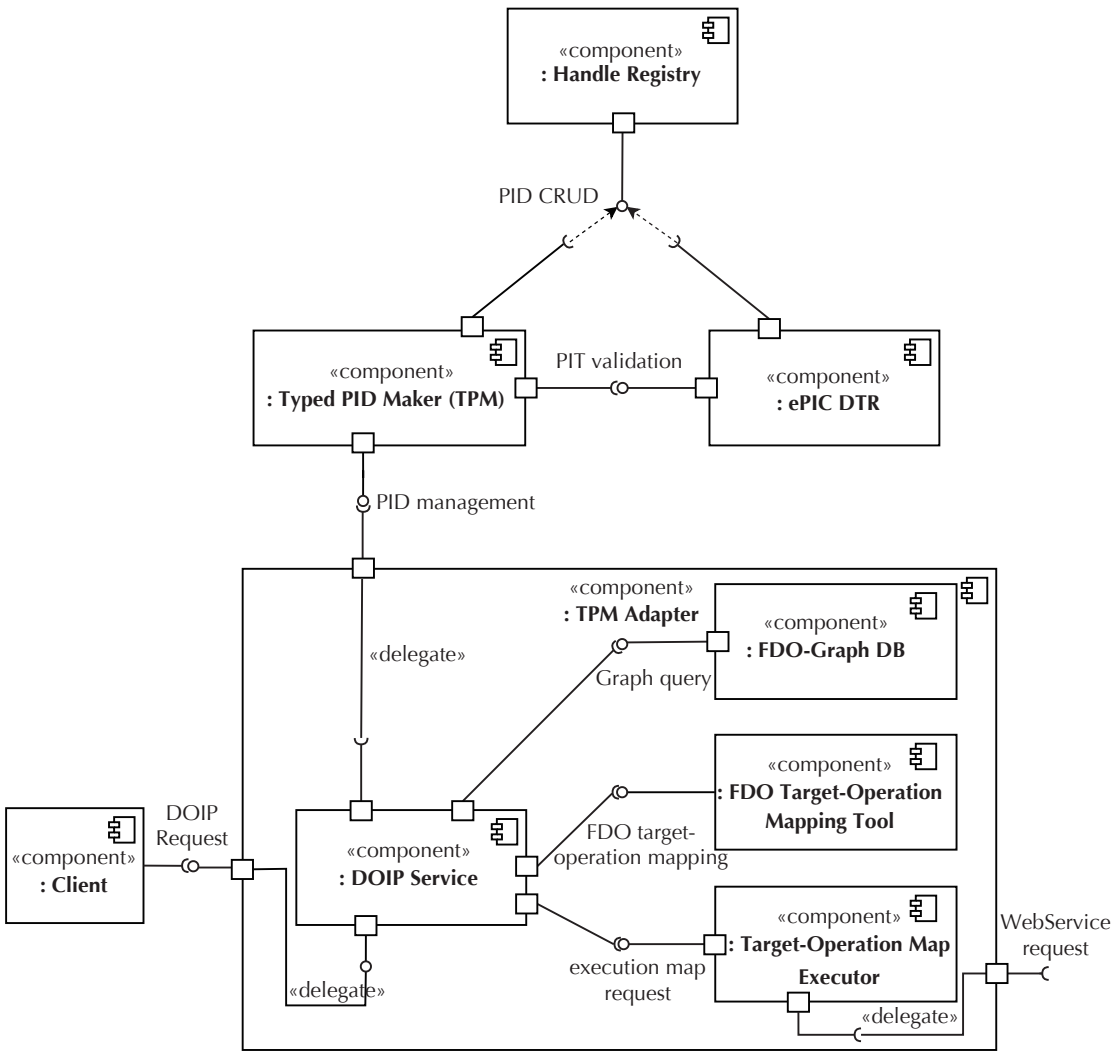

Figure 2. The FDO-Ops service architecture.

[3] https://github.com/kit-data-manager/pit-service.

4.2.1 Base Infrastructure

Our base infrastructure consists of two external services: the Handle Registry[4] to register and manage PIDs, and the ePIC DTR[5] to define and register PITs and Kernel Information Profiles (KIPs). To abstract the complexity of creating, updating, retrieving, and validating the PITs and KIPs of FDOs, a PIT service is required [49]. For this, we use an instance of the TPM service.

4.2.2 TPM Adapter

To speed up this retrieval, we integrate a Neo4j graph database[6] which ingests the labeled digraph Equation (6) representing the triples as given in Equation (7) after they have been derived from the hypergraph Equation (5) using Algorithm 1. Since Neo4j enables querying of opposite directions, storing only the hasOperation label is sufficient. Since *Get Related FDO* is implemented at the service, the graph database also stores FDO-FDO relationships constituted by PID-triples. These are directly derived from those typed attributes that reference the PID of FDOs. Again, storing one relationship of two opposite edges that have exactly opposite semantics is sufficient, e.g. hasMetadata and isMetadataFor. It is not always the case that two opposite edges exist, though.

Another sub-component, i.e., the FDO Target-Operation Mapping Tool, implements the logic of Algorithm 2 to map the given *execution protocol* of an *Operation FDO* and the information record of an associated *Target FDO* to a JSON-based *execution map* that can be further processed by the Executor. In order to make this setup work, it is crucial that the TPM Adapter is configured with the PITs expected for the *requirements* specification in each *Operation FDO* as well as the PITs for the *execution protocol types*.

4.2.3 Conflict Rules in the Neo4j Graph

To deal with duplicates and conflicting PIT-based edges in our Neo4j graph database, we define conflict rules through a combination of native Neo4j constraints and the systematic use of MERGE. For each rule, we provide a Cypher graph query language example.

- **Node Key Constraint**: Each label, (e.g. FDO), requires a unique and non-null property pid. This prevents duplicate identifiers. Example:

```
CREATE CONSTRAINT fdo_pid_key
FOR (n:FDO) REQUIRE n.pid IS NODE KEY;
```

- **Type Constraint**: The property pid must always be of type STRING. Example:

```
CREATE CONSTRAINT fdo_pid_type
FOR (n:FDO) REQUIRE n.pid IS :: STRING;
```

[4] https://www.handle.net/.
[5] https://dtr.pidconsortium.eu/.
[6] https://neo4j.com.

- **MERGE Semantics**: Relationships are created using MERGE, which guarantees idempotent creation. If the relationship already exists, no duplicate is added. Example:

```
CREATE (a:FDO {pid: 'A'})
CREATE (b:Operation_FDO {pid: 'B'});

// relationship (idempotent) with no duplicate created
MERGE (a)-[:HAS_OPERATION]->(b);
MERGE (a)-[:HAS_OPERATION]->(b);
```

We also enforce application-specific contradictions at write time using APOC (Awesome Procedures on Cypher) library triggers. The database runs Neo4j with APOC enabled and apoc.trigger.enabled=true. We register before-phase triggers that inspect the pending write set and terminate the transaction on violations via apoc.util.validate. This centralizes rule logic on the server, ensures client-agnostic enforcement, and removes duplication in the application layer. We implement the following rule, with the syntax being outlined in Appendix B

- forbid same-direction pairs: if HAS_METADATA(a, b) exists, IS_METADATA_FOR(a, b) is invalid and vice versa.

4.2.4 Executor

The Executor is a software component that must be able to handle the output of the FDO Target-Operation Mapping Tool, i.e., the ready-to-use *execution map*. To deal with the specifications in a given *execution map*, the Executor must have an implementation for each *execution protocol type*, e.g. for Web API operations, which are naturally limited due to their reuse for multiple operation implementations. On this basis, the Executor deals with the different technologies specified by an *Operation FDO*. Depending on the operation, the Executor technology module may also have to install additional software, which is detailed in the *execution map*. The command for the installation requirements is provided with a generic syntax "OS command" that the Executor replaces depending on the given OS. In addition, the Executor must be configured with the specific PITs of parameters in the *execution protocol* for the technologies it supports.

4.3 FDO-Ops Framework Sequence

The service architecture described in Section 4.2 constitutes the baseline for an FDO-Ops framework designed to support a variety of use cases. This framework illustrates how the operation categories and phases depicted in Table 1 can be applied. The sequence diagram in Figure 3 outlines the sequence of steps that are performed with our framework in order to cover these three phases. The diagram does not cover all activities given in the FDO-Ops process model in Figure 1, and does not show intermediate client actions between the steps such as inspection and selection of the individual information records of *Operation*- and *Target FDOs*. The client may optionally execute step 10 (Get Related FDOs) after step

9. Following this (or proceeding directly from step 9), the client can execute an Operation FDO in steps 13-20. The client may be an interactive computational environment such as a Jupyter Notebook, as in our demonstrative examples shown in the following section. The client is responsible for providing use case-specific input in all three phases and for evaluating the responses (such as the list of available *Information Record-* and *Data Resource Operations* for a given FDO). The framework then handles the machine-actionable processes, automating the interactions defined by the FDO-Ops process model. This automation abstracts the technical details of individual requests from the client, which are issued using the uniform DOIP/HTTP request pattern, typically via the GET method (Section 4.1). In case additional parameters are required, the client issues a POST request along with a payload. With this framework, the data and information flow according to the FDO-Ops process model is orchestrated as illustrated in Figure 4.

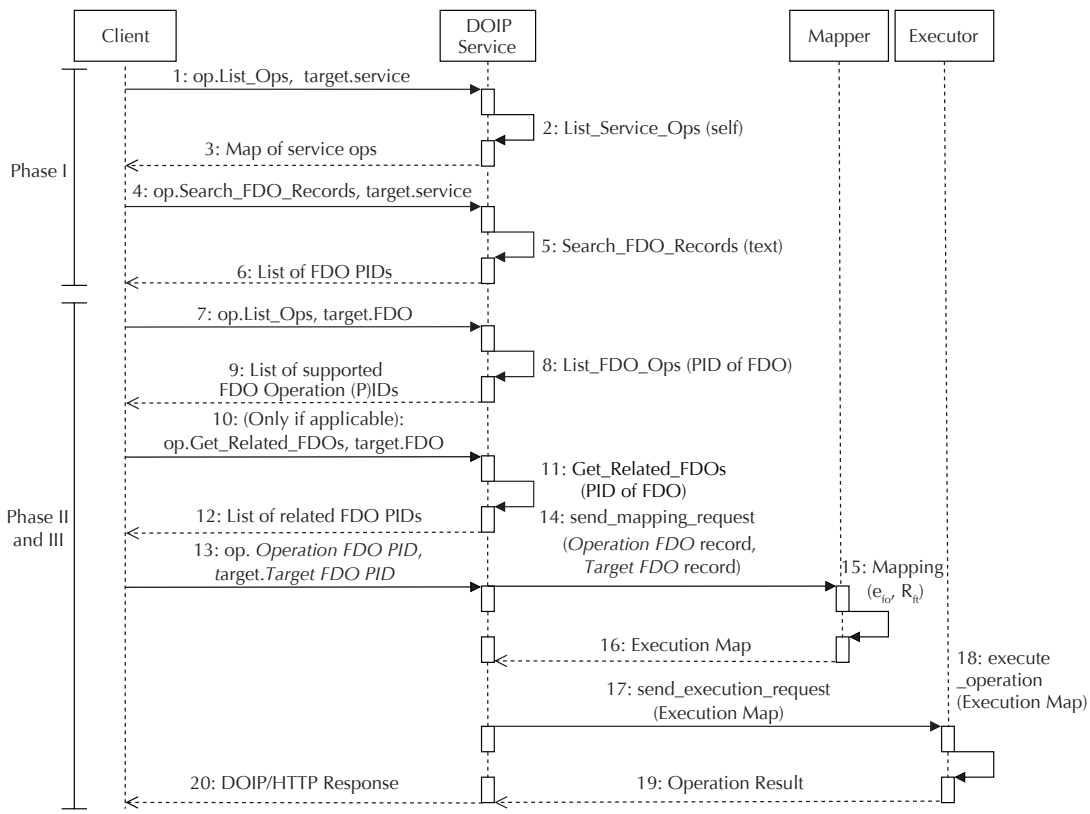

Figure 3. Sequence diagram of the FDO-Ops framework incorporating the DOIP request pattern.

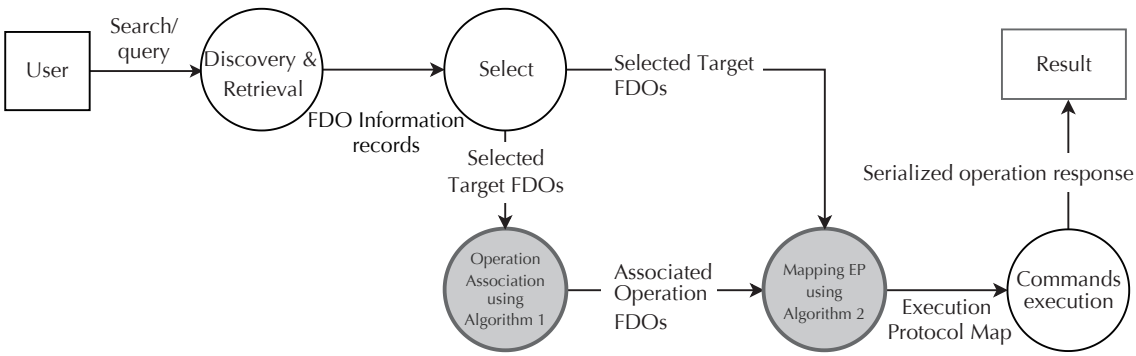

Figure 4. Data flow diagram using the FDO-Ops framework according to the process model.

4.4 Practical Example

In this section, we present a practical example that demonstrates the applicability of our model and prototype implementation, using data from two different domains and considering use cases throughout the three operation phases.

4.4.1 Profile for Operation FDOs

To represent the operations involved in our use cases as *Operation FDOs*, we define the *Operation KIP*[7]. This profile is derived from the *Helmholtz KIP* [50] that is used for all FDOs of the example datasets, and contains additional PITs for the *requirements* of the association model (Definition 1) and the *execution protocol* (Definition 3). We model two prototypes of this protocol using different implementation technologies that are executable on Unix-based systems using an Executor software. To help the client interpret the additional input and purpose of each operation, we also include PITs for the required client input, the operation name and the output type produced. All PITs that are attributes of the *Operation KIP*, namely *Required Type*, *Required Client Input*, *Web API Execution Protocol*, *Script Execution Protocol*, *Output Type*, *Operation Name*, can be further inspected via the profile that is registered in a running instance of the ePIC DTR at this time (s. PID in footnote).

4.4.2 Data and Operations Description

To demonstrate the domain-agnostic application of our model implementation, we employ two datasets, each one from a different domain, that have been earlier represented as FDOs using the FDO core data model (Section 2.2.3). To seamlessly use the content of these FDOs in our testbed environment, we replicate their information records using sandbox PIDs generated by the TPM.

The first example dataset Thermal Bridges on Building Rooftops (TBBR) [51] originates from the domain of energy research. This dataset contains multiple containerized images and corresponding metadata and

[7] https://hdl.handle.net/21.T11148/ea4e93d06a10e15d9cdf.

annotation files. The list of original PIDs for the FDOs of this dataset is available on Zenodo [52]. The FDO-FDO relationships of these data resources are derived from the PITs hasMetadata and isMetadataFor, and ingested into the Neo4j graph database as illustrated in Figure 5. Note that the original Handle PIDs are illustrated in this figure along with their file names for readability, reflecting the original PID-graph.

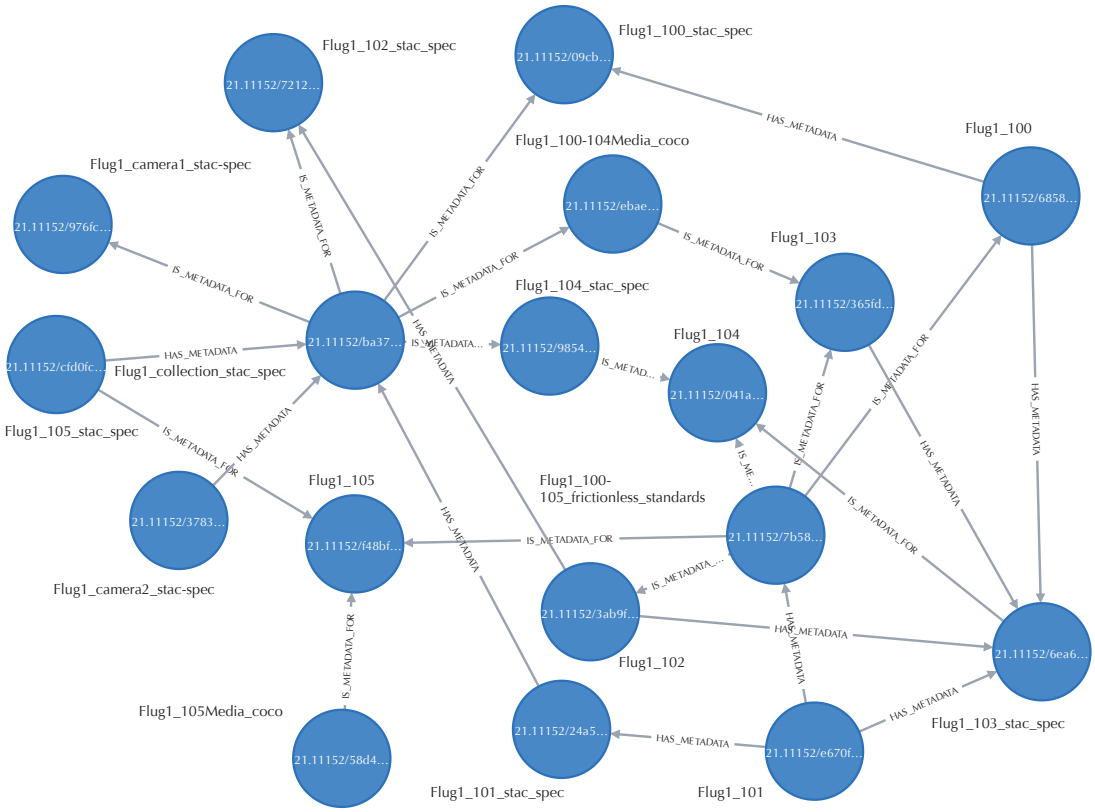

Figure 5. The Neo4j graph of the TBBR dataset representing FDO-FDO relationships.

The second example is a benchmark dataset for ontology matching [53] from the domain of digital humanities. It contains a containerized set of controlled vocabularies modeled using the Simple Knowledge Organization System (SKOS). The original PID of the FDO for this data set can be resolved at the Handle Registry®.

Next, we specify the operations, in addition to the *Management Operations* already implemented as part of the software architecture, and model them as *Operation FDOs*, using the *Operation KIP* specified in Section 4.4.1 with the appropriate *execution protocol*. We define operations for retrieving SKOS-based vocabulary terms information (*Get related terms*), converting a binary into an image data format (*Convert*

® https://hdl.handle.net/21.11152/a3f19b32-4550-40bb-9f69-b8ffd4f6d0ea?noredirect.

Numpy to PNG), and validating a specific format (*Validate SKOS*). In the following use case description, we will refer to these operations by their assigned name reported in the information record. An overview of the inferred *hasOperation* relationships between these *Operation FDOs* and their *Target FDOs*, i.e., the earlier described datasets, as ingested in the Neo4j graph database is illustrated in Figure 6.

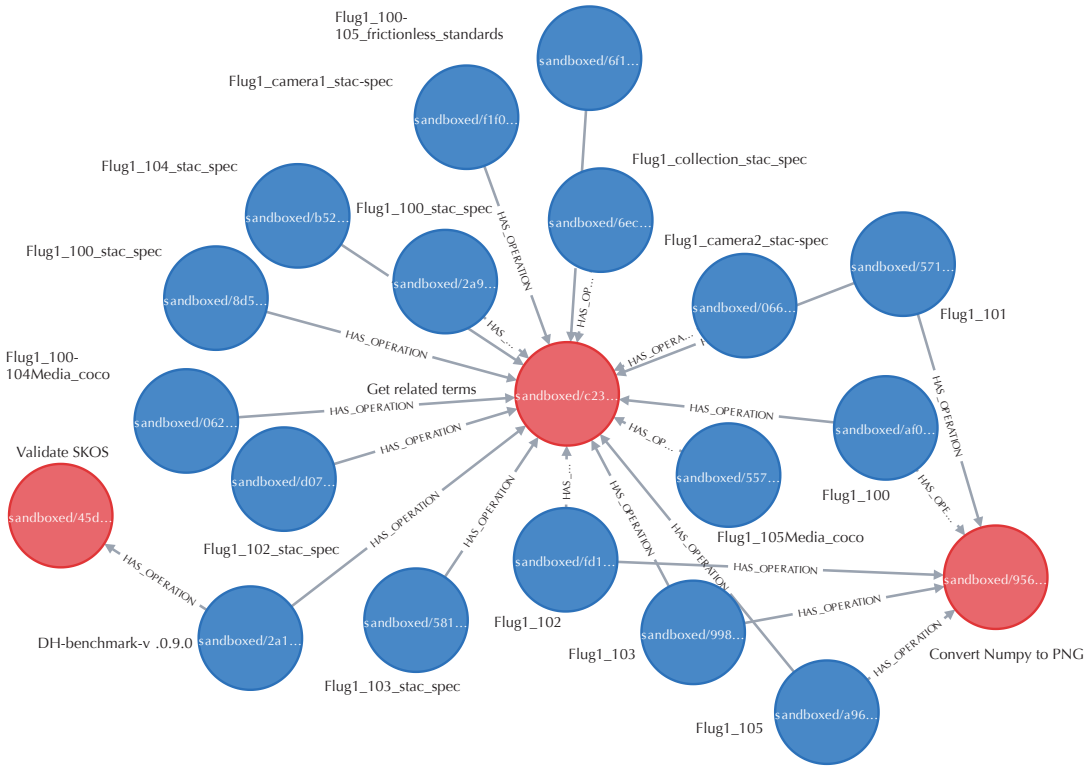

Figure 6. The Neo4j graph of the *Operation FDOs* (red) and *Target FDOs* (blue) relationships.

The JSON-information records to create the sandbox PIDs and those operations that are implemented as Python scripts can be found on GitHub (s. Section 7).

4.4.3 Use Cases

We consider different use cases to demonstrate how FDO-Ops can facilitate interoperable processes for the two example datasets using the FDO software architecture (Section 4.2) and the framework sequence (Section 4.3). The selected use cases focus on the early stages of a research activity and target users mainly in the academic context who want to discover, evaluate, and aggregate data resources for further use, e.g., Machine Learning tasks. To keep the use cases generic, consistent with what we address in Section 1, we deliberately do not further specify the potential research tasks.

Each use case is related to one of the operation phases (Table 1) and framework steps depicted in Figure 3, and begins with a brief description of the problem, followed by the identification of relevant components—specifically, the involved FDOs and operations. To demonstrate the domain-agnostic approach of FDO-Ops, we always apply them to both example datasets. In subsequent listings, we provide the technical details that are used to assess the association of the given operation according to Algorithm 1, the input by the client, if any, the execution map details (s. Algorithm 2) in case an *Operation FDO* is involved, and the operation output type. For the PITs that specify the *execution protocol parameters*, we provide the human-readable name instead of the PID for the *parameter type* which is used in the actual records, except for those PITs that are used as a parameter value. On this basis, we describe how these operations enable machine-actionable processes to facilitate the tasks described in the use cases. A Jupyter Notebook of the complete and reproducible results of these operations for each use case is available in the GitHub repository (s. Section 7).

Phase I - Initial Discovery. A common problem in research is discovering and retrieving suitable digital resources, often by searching in different locations. Typically, users already have an idea on the characteristics and technical requirements that these resources must meet, such as a specific format or license. The TPM Adapter supports full-text search of FDO information records using Elasticsearch, serving as entry point to retrieve potentially relevant FDOs via their PIDs. After exploring this capability through the List_Ops operation (Figure 3, steps 1-3) that lists all available service operations, the client acting on behalf of the user initiates an FDO query based on specific search terms, e.g., "*rdf+xml*" and "*npy-image*" for the MIME type or "*CC-BY*" for the license condition, by applying the Search_FDO_Records operation (Figure 3, steps 4-6) with the respective keywords as outlined in Listing 1. This yields the PIDs of both example datasets (i.e., those for the FDOs representing the TBBR images and the vocabulary dataset) which can then be further processed. The same PIDs are also retrieved when requesting all FDOs in the ecosystem, without using specific keywords.

Listing 1. Search FDO Records Operation Overview.

```
Implementation: Management Operation in TPM Adapter
Input provided by DOIP/HTTP Request Parameters:
    search terms "rdf+xml", "npy-image" and "CC-BY"
    in Elasticsearch syntax
Output Type: Handle-PID-list
```

As demonstrated for these examples, other FDOs from different locations that represent similar data can be retrieved likewise. At the end of this phase, the client has then yielded a selection of FDOs that are further assessed.

Phase II - Metadata Assessment. Clients typically want to investigate the relevance of the data represented by the retrieved FDOs for their specific research task prior to retrieving the actual bit sequence. Therefore, the client requests additional operations based on the association assessment using the List_ops operation on a *Target FDO* (Figure 3, steps 7-9) as outlined in Listing 2.

Listing 2. List FDO Operations Operation Overview.

```
Implementation technology: Management Operation in TPM Adapter
Input provided by DOIP/HTTP Request: PID of FDO
Output type: Handle-PID- or Service-ID-list
```

This yields a list of PIDs for *Operation FDOs* or IDs of service operations that are associated with the requested FDO. As a first action, the client may retrieve all FDOs related to those retrieved in Phase I which support the Get_Related_FDOs management operation (Figure 3, steps 10-12) based on the specific association requirements as outlined in Listing 3. In the case of the TBBR dataset, this retrieves the related annotation and experimental metadata FDOs, while the ontology matching dataset does not have a directly related FDO and is therefore not applicable to this operation.

Listing 3. Get Related FDOs Operation Overview.

```
Implementation technology: Management Operation in TPM Adapter
PITs for association requirement:
    hasMetadata (21.T11148/d0773859091aeb451528)
    OR isMetadataFor (21.T11148/4fe7cde52629b61e3b82)
Output type: Handle-PID-list
```

The client could then decide to repeat the procedure of retrieving additional operations using List_ops for those related FDOs it just retrieved, or to apply other operations to the initially retrieved FDOs. Independent from that, to get a better understanding of their semantic meaning, the client can apply the Get_Related_Terms *Operation FDO* (FDOs, steps 13-20) also retrieved in steps 7-9, and inspect the *topic* used to categorize the data. The operation is implemented as a Web API[9] and associated with the *topic* PIT (21.T11148/b415e16fbe4ca40f2270), present in all FDOs of both datasets, which takes as value a reference to a concept entry of the UNESCO Thesaurus[10]. In order to do so, the client must provide a SPARQL request as specified in the Required Client Input, indicating where the value of the *execution protocol parameter*, i.e., *httpBody* that maps to the *topic* PIT, has to be inserted, e.g.:

[9] https://vocabularies.unesco.org/sparql.
[10] https://vocabularies.unesco.org.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?prefLabel
WHERE {
        httpBody PIT skos:prefLabel ?prefLabel.
}
```

The SPARQL request is then mapped along with the value of the *topic* PIT from the *Target FDO's* information record and the *execution protocol parameters* (s. Listing 4) to an *execution map* that is processed by the Executor. Note that each *execution protocol parameter* corresponds to the structure defined in Definitions 3 and 4.

Listing 4. Get Related Terms Operation Overview.

```
Implementation technology: Web API
Required Client Input: parameter_type; httpBody PIT,
    description: Provide a SPARQL query using the skos:
    http://www.w3.org/2004/02/skos/core# prefix and the
    value of the parameter_type.
Input provided by DOIP/HTTP Request: SPARQL query
Execution protocol (Web API execution protocol parameters):
    http method: ['-G',
    '../vocabularies-unesco-org/sparql']
    http header: ['-H', ' "Accept:
    application/sparql-results+json"']
    http body: ['–data-urlencode',
    '⟨21.T11148/b415e16fbe4ca40f2270⟩',
    '[true, "[None, ' ',None]"]'⑪]
pit for association requirements:
    topic (21.T11148/b415e16fbe4ca40f2270)
Output type: application/rdf+xml, SKOS
```

The client can then inspect the related concepts of the topic, i.e., *Remote sensing* and *Artificial intelligence* for the TBBR FDOs, and *Ontology* for the vocabulary dataset FDO. Likewise, other SPARQL requests for this operation and other operations that process the contents of FDO information records can be applied. At the end of this phase, the client has evaluated the kernel information of the retrieved FDOs to assess the relevance of the underlying data resources for the given task.

⑪ merge mul entries condition (i.e., true) and the data structure for constructing mapped entries of 21.T11148/b415e16fbe-4ca40f2270 that are merged using a (prefix, delimiter, suffix) pattern (i.e., [None, ' ', None]) formatting the request as ⟨mapped PIT content 1⟩ ⟨mapped PIT content 2⟩ …

Phase III - Data Retrieval and Processing. Finally, the client retrieves and processes the represented bit sequence of those FDOs identified as being of relevance, as assessed in phase II. In phase III, the bit sequence can, in principle, be directly retrieved through the PIT it is referenced by on behalf of a *Management Operation* according to the *Get Web Entity through PITs* activity in the FDO-Ops process model (s. Figure 1). However, the operations available through the FDO-Ops framework may process the bit sequence prior to its retrieval to support the specific task the user has in mind in a machine-actionable way according to FDO-Ops. This may be, for example, the extraction and conversion of data into the required format or the validation of file formats. For demonstration, we designed two possible operations that can be applied to our two reference datasets, respectively. Those were already previously retrieved in Phase II using the List_ops management operation in steps 7-9.

An operation associated with the drone image data from the energy research example is the conversion of the *numpy* file format to the *png* image format, i.e., the Convert_Numpy_to_PNG *Operation FDO* (s. Figure 3, steps 13-20). The actual conversion function is implemented as a Python script that is published on GitHub (s. Section 7). The script accepts multiple file inputs as specified in the association requirements and returns the converted images. Likewise, an operation associated with the controlled vocabulary from the digital humanities example can be applied to validate that the corresponding files are well-formatted *SKOS RDF/XML*, i.e., by the Validate_SKOS_RDF *Operation FDO*. The validation function is also implemented as a Python script and is published in the same GitHub repository.

For both examples, the steps of retrieving input files from an associated *Target FDO*, extracting them, retrieving the Python script itself, and applying it to the files are crucial sub_operations that are modeled at the level of the *Operation FDO*, thus wrapping the actual main_operation. The sequence of these sub_operations is then handled by the Executor such that nested operations are executed first.

Listing 5 provides the overview for both operations as they have a similar structure apart from some differences between execution protocol details, their association requirements and outputs, which are indicated.

Listing 5. Convert Numpy to PNG and Validate RDF Operations Overview.

```
Implementation technology: Python Script
Execution protocols:
  main_operation (Script execution protocol parameters):
    installations (1): ['OS command', 'python3, pip3']
    installations (2 - for Convert Numpy to PNG):
    ['pip install', 'numpy, pillow']
    installations (2 - for Validate SKOS RDF):
    ['pip install', 'skosify']
    script method: ['python3', sub_operation1
    script argument: ['–files_dir', sub_operation2]

    sub_operation 1 (Web API execution protocol parameters):
      method: ['GET', '../URL_to_script_file'
      (differrent for each operation)]

    sub_operation 2 (Web API execution protocol parameters):
      method: ['GET', '21.T11148/b8457812905b83046284', 'false']
PIT for association requirements):
    digitalResourceType (21.T11148/1c699a5d1b4ad3ba4956)
    with value constellations (for Convert Numpy to PNG):
      application/x-npy
      (OR) [application/zstd AND application/x-npy]
    with value constellations (for Validate SKOS RDF):
      application/x-npy-image
      OR [application/zip AND application/x-npy-image]
    AND digitalObjectLocation
    (21.T11148/b8457812905b83046284)
Output type (for Convert Numpy to PNG): image/png
Output type (for Validate SKOS RDF): string, boolean
```

Similarly, as with these examples, other available operations can be executed on these datasets or related datasets (i.e., the metadata and annotation files). Additional suitable datasets that may originate from various data spaces can be processed the same way when they are represented as FDOs. At the end of this phase, the client has the original or properly (pre) processed version of the data resources represented by the previously analyzed FDOs. Depending on the research task, the client can then perform additional task-specific data processing that is not covered by the FDO-Ops framework.

5. EVALUATION AND DISCUSSION

5.1 FDO-Ops Model

In this section, we evaluate and discuss the implementation of FDO-Ops with respect to the model characteristics, the knowledge required by the client, the assumptions made for its applicability, and the future developments required.

5.1.1 Enabling Machine Actionability

Operation Categories. Our categorization approach for FDO operations in conjunction with the operation phases (s. Table 1) provides a baseline for clients to decide on use casespecific interactions with an FDO space. Having operation categories helps to differentiate between generic functionalities that are necessary for managing an FDO space, and advanced functionalities applied in the frame of specific tasks. These generic and advanced functionalities are, in turn, performed in three phases depending on the use case.

Attribute Typing. Our model includes three essential aspects for associating and applying operations to FDOs based on Attribute Typing: the specification of *requirements*, an *execution protocol*, and the representation of an FDO space as a graph. Implementing the validation algorithm (s. Algorithm 1) for the *requirements* including logical statements naturally aligns with the structure of an FDO's information record that is registered in the Handle Registry. Here, all key duplicates of a corresponding typed attribute $c_o \in C_o$, which constitutes a condition array of the *requirements* given by the array C_o, address a set of disjunctive statements, while each element $x_o \in c_o$ addresses a set of conjunctive statements for the given condition. This provides a mechanism for defining very coarse and generic up to highly granular and specific association requirements. The algorithm itself is relatively cost-intensive for a single FDO, i.e., the time complexity T is:

$$T = \mathcal{O}(\sum_{c_o \in C_o} |c_o|), \quad \text{or} \qquad T = \mathcal{O}(n + \sum_{c_o \in C_o} |c_o|) \text{ assuming } o \in O_m$$

and assuming R_t is a hash/set/map-based data structure. In addition, there are the costs of applying the algorithm to an entire FDO space. This is problematic for its use in large ecosystems where operations must be requested at scale. We show how this can be compensated by the use of a graph db ingesting the associations of *Target-* and *Operation FDOs* based on PID-quadruples and projection to triples that are once assessed and then stored permanently.

Operation FDOs and Execution Protocols. Representing operations as FDOs (i.e., *Operation FDOs*) as part of this design is a logical step towards a holistic view on the FDO concept to represent digital resources of all types, supporting its characteristic of being technologyagnostic. The entire FDO space can then be represented by a PID graph model whose nodes are FDO PIDs that are connected by edges derived from the key-value pairs of their PITs. The resulting triples can then be leveraged to retrieve information about the FDO's entity relationships. The *execution protocol* mapped with Algorithm 2 aligns with the *requirements* specification and provides a convenient way to specify a wrapper for an operation's

implementation based on a recursive pattern. Different technologies can be specified by different *execution protocols* that are all processable in the same way, and multiple operations can be implemented using the same technology, and can therefore be modeled using the same *execution protocol* type. In addition, multiple *Operation FDOs* could be created that wrap the same operation implementation and specify different conditions and sub-operations.

Digital resources that implement operations and are represented by FDOs are considered "active", as they expose an interface to a process that is applicable to another digital resource. These "active" resources, in turn, become operable by the same mechanism, i.e., by being associated as targets for other FDO operations. A particular advantage of this approach is the possibility to leverage already existing implementations and operation technologies available on the Web. Each technology is thereby described and made executable by wrapping it with an information record that is solely present on the uniform FDO level. The original operation does not have to be modified, as the information about the applicability and execution mechanisms of *FDO Operations* are stored in the globally available Handle Registry. From there, they can be retrieved and used with compatible software for validating the association, mapping and executing the operation, leaving relatively little expense to the client. However, not all existing digital resources that are an executable implementation for any type of data can be directly wrapped and represented as *Operation FDOs*. Many of these resources are structured and stored in a way that requires substantial overhead and may not be suitable for being modeled using our approach. Ideally, modular units like single API endpoints or specific scripts are used. The limitations for bigger applications will have to be tested in future developments.

FDO-Ops Process Model. The FDO-Ops process model defines the essential mechanisms that are available for an FDO and establishes the requirements for a client to use the FDO space. This comprises only knowledge about the uniform communication protocol and the minimal set of *Management Operations* that an FDO service must implement to make FDO-Ops' functionalities available. The goal is to keep most of the technical details abstracted from the client, such that operations or execution maps for operations from all categories across all phases can be requested in the same way, independent of their complexity. With this approach of associating and applying operations, FDOs become machine-actionable entities whose content can be processed more effectively: instead of integrating all possibly relevant metadata information and operations within one FDO, each FDO represents an individual digital resource that can interact with other resources represented as FDOs on the basis of the FDO-Ops rules set. In this way, the FDO fulfills its purpose of being an abstract and harmonized representation of a specific digital resource that encapsulates its essential information. This information is persistently and globally available and can be processed using a set of operations that are abstracted in the same way to gain and assess how the data resources can be (re)used. From this point of view, a data resource can be considered an actionable entity that performs operations to deliver a specific detail or a modified version of itself to the client upon request.

5.1.2 Implementation and Use Case Considerations

Prototype Architecture. In our prototype implementation, we use an existing infrastructure of base services that we extend with the necessary tools and components. This architecture aligns with the high-level architecture envisioned for the PID Information Types API prototype as described in [39] and constitutes the implementation of an FDO type system. For a deployed FDO space, we therefore see a federated PIT service (e.g., the TPM) that ingests, modifies, and retrieves FDOs as a crucial component. To connect to this service, and consequently to the FDO space, we propose adding a service providing functionally similar to the TPM Adapter. This service infers and stores the assessed FDO-operation associations and provides the mapped execution protocols that are processable by an Executor.

In our service architecture, we employ a Neo4j graph-database which can enforce conflict rules at write time based on the native property-level constraints on nodes and relationships, i.e., uniqueness, existence, type, and keys. With respect to scalability, Neo4j according to our graph model can efficiently handle very large FDO spaces through vertical scaling, provided sufficient memory and CPU are allocated, considering the limits of a single-server deployment: "A single instance of Neo4j can house at most 34 billion nodes, 34 billion relationships, and 68 billion properties." [54]. This enables to process large amounts of digital resources quickly based on their FDO representation. This is possible because FDO-Ops focuses on the entity-relationships and available operations. Traversing the graph and identifying these connections can be done in constant time, whilst performing an operation requires individual execution time.

Our Executor is directly integrated in the TPM Adapter service, reducing the overhead for the client to a minimum of only handling DOIP requests. In a production-ready environment, multiple Executors could be federated and resources distributed, depending on the extent of the operation demands. The set of provided Executors will thereby depend on the set of PITs that specify the different *execution protocol types* for different technologies. This will have to be governed with respect to resources and security aspects such that only verified operations are executed. We argue that finding a consensus at this level is more convenient in terms of standardization than aligning the more specific and versatile KOSs and the corresponding technologies that are used within different organizations; our model provides the baseline for this.

Our prototype implementation realizes a high level of abstraction within the operation phases by using the DOIP communication protocol pattern through DOIP/HTTP. This fully aligns with the architecture of other DOIP services, such as Cordra [55], which essentially implements "basic" operations similar to what we define as *Management Operations*. In turn, *Information Record* and *Data Resource Operations* realized by *FDO Operations* can be considered as analogous to "extended" operations in the DOIP context.

Example Use Cases. Our practical examples from different domains give an impression of the use cases FDO-Ops can address, proving the technical feasibility and domain agnosticism of the model. Whilst certain operations can be applied to all FDOs, which is typically the case for Phases I and II, others are specific to the digital resource's type an FDO specifies, mainly in Phase III. All operations are thereby associated and mapped for execution by the same algorithms we introduce in our model. Therefore, we

conclude that the applicability of FDO-Ops is not limited to a specific type of domain or data. However, the actual scope of use cases and processability options will be decided by the communities that share a common FDO space and ingest their digital resources. We want to emphasize that operations that are typically useful in multiple research tasks and in the stages of data selection, aggregation, and preparation, such as the ones we used, are first-class citizens to be modeled as *Operation FDOs*. Other operations that are very specific are less likely to be available via the FDO-Ops framework, which ultimately depends on the maturity of the FDO space. Hence, data processing by the FDO-Ops framework is at this point limited to generic tasks, but can become more versatile when FDO-Ops is adopted by different communities that provide advanced operations and define additional *execution protocols* and service components. Extended use cases could perspectively address the orchestration of multiple operations that are chained to automate complex workflows. Using AI technologies like Recommender Systems or Large Language Models could assist in finding the optimal workflow for a given use case or to enhance the capabilities of Executor tools as for example also described in [56].

5.2 Integration with KOS

The main goal of FDO-Ops in the context of KOS integration is to leverage the capabilities of digital resources and available procedures for them, whilst abstracting details from the (re)users as much as possible. Compared to the limitations of existing interoperability solutions provided by KOSs that require handling of the individual technology, FDOs modeled using FDO-Ops stand out by requiring significantly less overhead and technical knowledge from users. This is the case because the handling of technical details is abstracted by the FDO-Ops framework and a uniform communication protocol (e.g. DOIP). We argue that a machine-actionable environment according to FDO-Ops decreases the hurdles of leveraging KOSs, because they ease application of required procedures and assembling of individual data units when required. It needs to be pointed out that in contrast to this ease, setting up, maintaining and enriching an FDO space requires initial extra work which from our point of view is worthwhile in aspects of the heterogeneity of KOSs and resulting interoperability challenges.

Metadata Schemas and Service Specifications. Positive effects through the integration of FDO-Ops with KOSs such as metadata schemas take effect when the corresponding digital resource (e.g. metadata files) are represented as FDOs and declare their correspondence to their KOS in the kernel information. This may comprise minimum information for the identification of the KOS needed by procedures that know how to interpret these contents, e.g. the schema identifier and its version. Furthermore, multiple corresponding digital resources that are natively stored over distributed systems can also be interconnected through their FDO's kernel information. This way, KOS-specific barriers such as crosswalk definition and schema-fragmentation are addressed as follows: (a) PIT-typed KOS information/provenance persists context for interpretation of the metadata resource content across systems (b) Operation FDOs for related schema dereferencing (similar to dereferencing related vocabulary terms s. Listing 4) and validation (similar to retrieving the bit sequence and validating its structure e.g. s. Listing 5) can retrieve and check the authoritative KOS state at run-time; and (c) Procedures for content interpretation and manipulation such as crosswalks are represented by reusable, citable, and auditable *Operation FDOs* rather than ad-hoc scripts.

However, FDO-Ops can only leverage those functionalities that are inherent to the digital resources they represent. In case semantic artifacts such as context loss in crosswalks or ambiguity in cross-domain mappings remain, they cannot be directly addressed.

Executable specifications—such as for APIs and workflows—can either be utilized by embedding them in an *Operation FDO's execution protocol*, or by representing them purely as a *Target FDO*. In the latter case, additional operations would be required to "activate" their functionality. This decision finally depends on the feasibility of their representation as an *Operation FDO*, e.g., the availability of proper *execution protocols*, and the design decision of the creator.

Ontologies. Whilst ontologies and Knowledge Graphs focus on the semantic structure of knowledge and querying between Web entities, FDO-Ops focuses on persistency, generic relationships, and advanced processing options for these entities. Perspectively, there exist two potential synergistic effects: Ontology-based Semantic Web resources can either themselves be represented at the FDO level and profit from the features outlined above, or serve as entry point to the FDO realm, e.g., by semantically enriching and interpreting the typed metadata for traversing FDO relations in the discovery phase (Phase I). For the latter, the FDO-Ops model would have to be defined in an ontology, paving the way towards a semantic FDO Knowledge Graph that could be queried using RDF-centric requests, e.g. SPARQL.

In summary, applying FDO-Ops does not require modification of existing (meta) data management systems that use existing concepts, nor does it reinvent their functionalities. It rather enables seamless integration of established standards, protocols, and technologies—such as those from KOSs (e.g. SKOS RDF) and executable specifications (e.g. Web APIs)—into FDOs, allowing their content to be processed within the FDO-Ops framework in an interoperable way, as demonstrated by our use cases in Section 4.4.3.

5.3 Comparison to KOS-enhanced Interoperability Tools

Comparing FDO-Ops to tools enhanced by KOSs (s. Section 2.2) reveals their limitations for interoperability as a standalone in contrast to their use in conjunction with FDOs. In detail, we consider the type of operations that these tools provide, the types of KOSs involved, and the aspect of machine-actionability that is enabled in this way. The comparison results are summarized in Table 2. In essence, the main differences are that FDO-Ops types, discovers, and executes operations across data objects, irrespective of their domain, whilst the other tools either expose API affordances, script/query RDF, publish and discover metadata, or run workflows. Thus, they are much more specific and do not provide a fully type-bound, cross-service operation model that is agnostic to the technologies and processed digital resources.

Table 2. FDO-Ops compared to LDScript/ActiveRDF, Hydra/HATEOAS/RDF-OAS, FAIR Data Points + LDP, and CWL

System	Types of integrated KOSs	Type of Operations	Machine-actionability
FDO-Ops	PID-centric KOS: PIT and KIP in Handle/dtr. Integrates external KOS (e.g., SKOS) via typed links and operations.	Reusable, type-bound operation representations with association rules and an execution protocol for targeted data representations (FDOs).	Versatile actions for discovery, retrieval, and manipulation of digital resources and their metadata through different operation technologies via FDO-service execution.
LDScript/ActiveRDF	RDF-centric KOS: RDFS/ OWL ontologies and vocabularies with SHACL validation	ActiveRDF supports read/ write/find plus adapter-backed queries. LDScript provides SPARQL-native functions and class-bound procedural attachments on metadata	RDF-centric actions. ActiveRDF is confined to application code and store APIs. LDScript is actionable only inside the SPARQL engine.
Hydra/HATEOAS/ RDF-OAS	RDF and hypermedia KOS for endpoint affordances and payload/ interface schemas: Hydra vocabularies; link-relation and media-type profiles; RDF-transformed OpenAPI schemas via linked vocabularies and R2RML.	Hypermedia/API descriptions of API endpoints, CRUD-oriented	Discovery of endpoints and invocation of HTTP method plus target IRI or URI template; clients follow links or SPARQL-found endpoints.
FAIR Data Points + LDP	Metadata-centric: DCAT v2 + FDP-Ontology with LDP containment (uses FOAF, DCTERMS; optional SKOS) and SHACL for validation.	Metadata-record CRUD, access and navigation via LDP containers and DCAT/FDP-Ontology shapes.	Automated metadata traversal and CRUD via LDP for navigation and search of metadata ("follow-your-nose approach").
CWL	Schema Salad for strict schemas and JSON-LD/ RDF mapping; provenance via CWLProv.	Concrete process types: CommandLineTool, ExpressionTool, Workflow; each step is a file or data transform.	Engine-executable workflows (e.g., cwltool and other runners), often containerized, using external registries.

However, these existing tools and technologies often perform the particular tasks that are finally abstracted and are therefore a crucial component for FDO-Ops to work. We have already shown how RDF-based vocabularies in conjunction with a SPARQL-API can be used this way (s. Section 4.4.3). Other potential complementary scenarios for the considered tools are exemplified as follows:

- Represent Hydra/OAS documents as FDOs such that the specified APIs can be uniformly discovered, validated, and invoked by corresponding tools that are represented as *Operation FDOs*.
- Use of FDP + LDP to publish and index FDO information records and to link FDOs and Operation FDOs for discovery. Potential alternative to Elastic Search.
- Use of ActiveRDF to manipulate RDF graphs that describe FDOs. Alternative FDO space discovery.
- Implementation of *Information Record Operations* with LDScript inside SPARQL engines, wrapped as *Operation FDOs* for typed, cross-service invocation.
- Orchestration of *Data Resource Operations* via CWL workflows, while FDO-Ops handles typed discovery and invocation of each step.

Overall, FDO-Ops provides the missing layer for distributed data-operation associations plus uniform and abstracted client interface and execution. The other tools fulfill the specific tasks that can be leveraged by FDO-Ops, or can assist in FDO management: API description, RDF scripting, metadata discovery, and workflow runtime.

5.4 Quantification of Impact

We quantify the impact of interoperability across the three operation phases (s. Table 1) by comparing an FDO-Ops client with a client working directly against heterogeneous KOSs and system-specific interfaces. The metrics defined in the following are compared in Table 3 between the FDO-Ops and Non-FDO-Ops paths, thereby referring to the earlier example use cases. For FDO-Ops we assume an FDO space reachable via an FDO-DOIP service via DOIP or DOIP/HTTP.

Table 3. Comparison of FDO-Ops vs non-FDO-Ops for data access and processing.

Metric	FDO-Ops Path	Non-FDO-Ops Path	Examples from Use Cases
Kernel metadata querying	$\tau(a) = 1, \forall a$	$\tau(a) \gg 1$	$\tau(a) = 2$ for a =license URI: Zenodo → rightsList.rightsUri vs GitHub → license.url
Number of directly requested interfaces	$\iota = 1$	$\iota \gg 1$	$\iota = 5$
Time complexity of listing operations/ resources	$\sigma = \mathcal{O}(log(\lvert \mathcal{V}_f \rvert) +$ degree of edges for node of $v_f \in \mathcal{V}_f)$, $\rho = \mathcal{O}(log(\lvert \mathcal{V}_o \rvert) +$ degree of edges for node of $v_o \in \mathcal{V}_o)$	$\sigma = \mathcal{O}(\kappa)$, $\rho = \mathcal{O}(\lambda)$, both highly varying, depending on the given systems information about digital resources and operations are stored, and given use cases	Adding to κ, λ: drone image retrieval from Zenodo or other search index, identification of associated vocabulary term for a drone image, discovery of and navigation to the UNESCO Thesaurus SPARQL endpoint.
Data volume transferred to client	$\beta(s)$ or $\beta(\hat{s})$ via Digital Resource Operations	Often only $\beta(s)$ (full payload)	Drone image: NumPy array ~ 45MB vs PNG ~13MB

Definition 5 (Quantities used in metrics). Let $\tau(a)$ be the number of synonymous terms used for the kernel metadata attribute a across the ecosystem. Denote ι as the number of distinct interfaces across data, metadata, operation, and programming services. Let $\beta(s)$ be the number of bytes (volume) transferred for a bit sequence s, and $\beta(\hat{s})$ the number of bytes for a manipulated bit sequence $\hat{s}$, with $\beta(s) \geq \beta(\hat{s})$. Let $\mathcal{V}_f$ denote the set of nodes (vertices) in a graph database that correspond to all ingested FDOs, and $\mathcal{V}_o \subseteq \mathcal{V}_f$ denote the set of those vertices that correspond to all ingested Operation FDOs. Let κ denote the retrieval effort to list all operations associated with a given digital resource; define $\sigma = \mathcal{O}(\kappa)$ as the time complexity. Let λ denote the retrieval effort to list all digital resources associated with a given operation; define $\rho = \mathcal{O}(\lambda)$ as the time complexity.

Impacts by phase.

- **Phase I — Metadata attribute querying:** FDO-Ops enforces unambiguous typing for each kernel attribute, so a single DOIP query suffices: $\tau(a) = 1$ for each a. Non-FDO paths admit divergent terms per standard or system, so $\tau(a)$ grows with heterogeneity. Result: normalized syntactic access and discovery, unambigious typing.
- **Phase I, II & III — Interfaces:** FDO-Ops uses one uniform protocol and service API that abstract underlying APIs of the corresponding digital resources, so $\iota = 1$. Non-FDO paths add interfaces per KOS or store, so ι grows with ecosystem diversity. Result: improved technical interoperability.
- **Phase II & III — Operation and data listing:** In FDO-Ops, after graph ingestion based on triples (see Equation (7)) using Neo4j:
  $\kappa = log(|\mathcal{V}_f|) + \text{degree of edges for node of } v_f \in \mathcal{V}_f$
  and $\lambda = log(|\mathcal{V}_o|) + \text{degreeofedgesfornodeof}v_o \in \mathcal{V}_o$.

  σ then defines the complexity for the List Ops operation and ρ likewise a symmetric operation that can be defined, which traverses the same edges in the graph and is requested via the same DOIP/HTTP interface. In heterogeneous KOSs, κ, λ are affected by the number of schema inspections, terminology alignments, discovery steps (catalogs, portals) and requests to databases. Result: fixed- and depending on heterogeneity and technology of reference systems also reduced time listing of available operations, ensuring reduced discovery effort for clients.

- **Phase III — Volume transfer:** FDOs support either full transfer $\beta(s)$ or reduced transfer $\beta(\hat{s})$ via standardized *Digital Resource Operations* (e.g., image conversion), often yielding order-of-magnitude byte savings for the client. In turn, the FDO-Ops service must temporarily store and process the full number of bytes. Non-FDO-Ops connected repository data storage systems typically return only full payloads of digital resources. Result: efficiency for client that eases handling across data types.

5.5 Enhancement of Interoperability Levels

To evaluate how FDO-Ops enhance different levels of interoperability, we use the four-level interoperability model of the EOSC IF (s. Section 2.1) and describe how these levels are met by systems that comply with the FDO-Ops model. We argue that digital resources then become interoperable by these

means through their FDO representation and can be reused in an automated way. We also consider aspects introduced in the FDO core data model by [36], which are important prerequisites for the development of FDO-Ops. Note that some of the considered interoperability level enhancements are directly related to the conclusions we draw from the earlier sections, especially our quantification of impact in Section 5.4. Table 4 summarizes the relation between the levels and key components of FDO-Ops.

Table 4. Key FDO-Ops components across interoperability levels.

Interoperability Level	FDO-Ops Key Components
Technical	DOIP service (TPM Adapter), TPM service, Handle Registry, ePIC dtr, Attribute Typing across FDOs, distributed operation technologies represented by *Operation FDOs*
Syntactic	PITs, KIPs, graph-stored object relationships, DOIP/HTTP request pattern
Semantic	PIT-links to KOS implementations (e.g. SKOS/RDF vocabularies), *Operation FDOs* for dereferencing/validation
Legal	Copyright and version control metadata in Kernel Information, possibility to define access control operations
Organizational	Standardized Handle PID policy, dtr-backed types, governance through profiles (e.g. Helmholtz KIP)

Technical Interoperability: Digital resources located in different storage systems remain unmodified and are represented by uniformly structured, abstract, and machine-interpretable objects, i.e., FDOs, that are available through an FDO infrastructure. This infrastructure is the basis for an FDO framework that can be used by clients to process FDOs. The type system of FDOs represents syntactic and semantic definitions against which can be programmed, enabling specific activities that an FDO can perform. In this way, FDOs can be processed by the machine-actionable interactions between *Target FDOs*, *Operation FDOs*, FDO services and tools that utilize known *execution protocols*.

Syntactic and Semantic Interoperability: FDOs provide a minimum metadata model with core metadata elements of hierarchically structured PITs, aggregated in a KIP. This modular metadata structure allows extensions without breaking core interoperability. PITs enforce syntactic rules for metadata key-value pairs and offer semantic specification via links to other FDOs or Web entities such as definitions, ontologies, schemas, and semantic artifact repositories. Systems can communicate with each other using the uniform structure and communication protocol for FDOs. In addition, by utilizing associated operations, semantic contents of related Web entities, e.g. vocabularies, can be processed and interpreted.

Legal Interoperability: A machine-actionable license format is mandatory in each FDO complying with the RDA Kernel Information Recommendations, possibly linked to centralized knowledge sources for verification. KIP and PIT extensions support granular metadata (e.g., timestamps, license expiration) to manage copyright and version control. FDO operations enable legal frameworks and access control policies dynamically based on typed metadata.

Organizational Interoperability: Organizations share a high-level FDO service infrastructure while managing their digital resources independently. FDO services enable the provision of a clear and standardized PID policy using Handle PIDs and PITs. A consensus on the core machine-interpretable metadata that is leveraged by operations ensures interoperability within the shared FDO space. By using a controlled FDO service infrastructure, organizations can ensure that only authorized entities are permitted to publish, modify, or access digital resources.

Overall, FDO-Ops, building upon the FDO core data model which addresses legal and organizational foundations, implements machine-actionable processes that enhance technical and syntactic interoperability and provide the mechanisms to enforce organizational policies. Semantic interoperability is not inherently addressed beyond the kernel information, but can be achieved (i) when appropriate semantics are closely coupled to the syntactically interoperable PITs, and (ii) through entity-relationships when semantic resources are represented as FDOs and have appropriate operations associated for their interpretation (as described in Section 5.2). This is certainly of high relevance, since semantic interoperability is considered the ultimate characteristic of fully interoperable systems as described in [15]. The authors of this work also consider semantic interoperability as a prerequisite for machine-actionability, which is a different view from the work of [2], where machine-actionability is rather seen as a prerequisite for interoperability. One could argue that interoperability and machine-actionability are interdependent and have synergistic effects on each other. Therefore, a specification of different levels of machine-actionability in analogy to different levels of interoperability seems to be an important future work task. From our point of view, we agree that the other levels of interoperability are prerequisites for semantic interoperability, which we consider, however, not as the sole requirement for machine-actionability, at least not in the context of the FAIR Principles. Instead, it is a necessary condition to enable advanced machine-actionable and possibly intelligent decision making. For this, other interoperability levels must be met first, which we realize with FDO-Ops.

Likewise, federated interoperability (s. [57]) can be achieved through terminological interoperability (s. Section 2.1), assuming that (meta)data formats and schemas are dynamically adapted through entity mappings and schema crosswalks (s. Section 5.2). Connected to this, FAIR Services are required to achieve an IDFS that supports FDOs as representations of (meta)data statements or semantically meaningful collections of statements as crucial enablers for propositional interoperability. We believe that FDO-Ops constitutes a viable way of implementing an operations service in this context, and can be complemented by the additional terminology and schema services described. Our work thus aligns with the perspective of the authors in [15] to reuse and build on existing and ongoing work in this field to effectively implement FAIR Services.

6. CONCLUSIONS

In this paper, we introduce FDO-Ops, a comprehensive model and practical framework designed to associate and execute operations on FAIR Digital Objects, turning them into machine-actionable entities. This is achieved by providing a precise definition of operation categories and phases in conjunction

with generic mechanisms for associating operations to FDOs based on Attribute Typing, and executing operations on FDOs through dedicated execution protocols. Both mechanisms are built as part of the type system that relies on PID Information Types used in the existing FDO core data model, to which we refer.

Our prototype implementation demonstrates the applicability of FDO-Ops: representative use cases, drawn from the domains of energy research and digital humanities, confirm that operations can be uniformly associated, mapped, and executed, regardless of domain-specific data formats and technologies. The capabilities of integrated Knowledge Organization Systems are thereby leveraged and made uniformly available. This eases the early discovery phase, aggregation and quick decision making of data intensive tasks. Evaluation of FDO-Ops within the context of the EOSC IF reveals enhancement of technical and syntactic interoperability beyond the KOS-only use.

FDO-Ops thus emerges as a robust, domain- and technology-agnostic adaptable solution, capable of substantially advancing the practical implementation of FDOs to ultimately foster a more connected, interoperable, and automatically discoverable data space in the spirit of the FAIR Principles. Future research will focus on refining the FDO infrastructure service components, governance procedures, and security to advance the impact on semantic, legal, and organizational interoperability. Another aspect is to define reusable execution protocols and extend the capabilities of the FDO-Ops framework to support more complex workflows, including chained operations and dynamic interactions, possibly supported by AI.

7. CODE AND RECORDS AVAILABILITY

The entire repository with the prototype implementation, the FDO information records, and a ready-to-use Jupyter Notebook to test the FDO-Ops framework according to our described examples is available on: https://github.com/kit-data-manager/DOIP-Client-and-TPM-Adapter-Service-for-FDOs.

AUTHOR CONTRIBUTIONS

Nicolas Blumenröhr: Conceptualization, Methodology, Writing-Original draft preparation,

Project administration, Writing, Reviewing, and Editing;

Philipp Ost: Methodology, Writing, Reviewing, and Editing, Project administration;

Rossella Aversa: Project administration, Writing, Reviewing, and Editing;

Felix Kraus: Writing, Reviewing, and Editing;

Maximilian Inckmann: Writing, Reviewing, and Editing;

Achim Streit: Project administration, Writing, Reviewing, and Editing.

ACKNOWLEDGEMENTS

This work is funded by the Helmholtz Metadata Collaboration Platform (HMC), NFDI4ING (DFG – project number 442146713), and supported by the research program "Engineering Digital Futures" of the Helmholtz Association of German Research Centers.

ABBREVIATIONS

API Application Programming Interface

CRUD Create, Read, Update, Delete

DOIP Digital Object Interface Protocol

DTR Data Type Registry

EIF European Interoperability Framework

EOSC European Open Science Cloud

EOSC IF European Open Science Cloud (EOSC) Interoperability Framework

FDO FAIR Digital Object

FDOF FDO Framework

IDFS Internet of FAIR Data and Services

KIP Kernel Information Profile

KOS Knowledge Organization System

LCIM Levels of Conceptual Interoperability Model

OOP Object-Oriented Programming

PID Persistent Identifier

PIT PID Information Type

RDA Research Data Alliance

RDF Resource Description Framework

SKOS Simple Knowledge Organization System

TBBR Thermal Bridges on Building Rooftops

TPM Typed PID Maker

REFERENCES

[1] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning," *IEEE Trans. Knowl. Data Eng.*, Aug. 2019. [Online].

[2] M. D. Wilkinson, R. Verborgh, L. O. B. d. S. Santos, T. Clark, M. A. Swertz, and et al., "Interoperability and FAIRness through a novel combination of web technologies," *PeerJ Comput. Sci.*, vol. 3, p. e110, 2017. [Online].

[3] D. Haynes, "Metadata standards," in *Metadata for Information Management and Retrieval: Understanding Metadata and Its Use*, pp. 49–74, Facet, 2018. [Online].

[4] J. Lathem, K. Gomadam, and A. Sheth, "SA-REST and (S)mashups: Adding semantics to RESTful services," in *Proc. Int. Conf. Semantic Computing (ICSC)*, pp. 469–476, 2007.

[5] M. Baca, *Introduction to Metadata*. Getty Research Institute, 2016. [Online].

[6] M. D. Wilkinson, B. Vandervalk, and L. McCarthy, "The semantic automated discovery and integration (SADI) web service design-pattern, API and reference implementation," *J. Biomed. Semantics*, vol. 2, no. 1, p. 8, 2011. [Online].

[7] F. Michel, C. Faron-Zucker, O. Gargominy, and F. Gandon, "Integration of web APIs and linked data using SPARQL micro-services-application to biodiversity use cases," *Information*, vol. 9, no. 12, p. 310, 2018. [Online].

[8] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, and et al., "The FAIR guiding principles for scientific data management and stewardship," *Sci. Data*, vol. 3, no. 1, p. 160018, 2016.

[9] T. Berners-Lee and M. Fischetti, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, Sept. 1999. ISBN: 0-06-251586-1.

[10] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 concepts and abstract syntax," 2014. [Online].

[11] N. Konstantinou and D.-E. Spanos, "Introduction: Linked data and the semantic web," in *Materializing the Web of Linked Data*, pp. 1–16, Cham: Springer International Publishing, 2015.

[12] K. Smedt, D. Koureas, and P. Wittenburg, "FAIR digital objects for science: From data pieces to actionable knowledge units," *Publications*, vol. 8, p. 21, 2020.

[13] P. Wittenburg and G. Strawn, "Digital objects as drivers towards convergence in data infrastructures," 2018. [Online].

[14] S. Soiland-Reyes, C. Goble, and P. Groth, "Evaluating FAIR digital object and linked data as distributed object systems," *PeerJ Comput. Sci.*, vol. 10, p. e1781, Apr. 2024. [Online].

[15] L. Vogt, P. Strömert, N. Matentzoglu, N. Karam, M. Konrad, M. Prinz, and R. Baum, "Suggestions for extending the FAIR principles based on a linguistic perspective on semantic interoperability," *Sci. Data*, vol. 12, p. 688, Apr. 2025. [Online].

[16] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "A comprehensive survey on interoperability for IIoT: Taxonomy, standards, and future directions," *ACM Comput. Surv.*, vol. 55, no. 1, pp. 1–35, 2023. [Online].

[17] European Commission: Directorate-General for Research and Innovation, EOSC Executive Board, O. Corcho, M. Eriksson, K. Kurowski, M. Ojsteršek, C. Choirat, M. van de Sanden, and F. Coppens, *EOSC Interoperability Framework: Report from the EOSC Executive Board Working Groups FAIR and Architecture*. Publications Office of the European Union, 2021. [Online].

[18] A. Tolk and J. Muguira, "The levels of conceptual interoperability model," in *Proc. Fall Simul. Interoperability Workshop*, vol. 7, 2003.
[19] R. S. P. Maciel, P. H. D. Valle, K. S. Santos, and E. Y. Nakagawa, "Systems interoperability types: A tertiary study," *ACM Comput. Surv.*, vol. 56, pp. 254:1–254:37, June 2024. [Online].
[20] J. Riley, *Understanding Metadata*. National Information Standards Organization (NISO), 2017.
[21] L. Chan and M. Zeng, "Metadata interoperability and standardization-a study of methodology, part I," *D-Lib Mag.*, vol. 12, no. 6, 2006. [Online].
[22] K. Menzel, S. Törmä, M. Kiviniemi, K. Tsatsakis, A. Hryshchenko, and M. N. Lucky, "Linked data and ontologies for semantic interoperability," in *Innovative Tools and Methods Using BIM for an Efficient Renovation in Buildings*, pp. 17–28, Cham: Springer International Publishing, 2022. [Online].
[23] D. Martin, M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, and N. Srinivasan, "Bringing semantics to web services with OWL-S," *World Wide Web*, vol. 10, no. 3, pp. 243–277, 2007. [Online].
[24] A. Polleres and D. Fensel, "Towards intelligent web services: Web service modeling ontology (WSMO)," 2005. [Online].
[25] E. Paslaru Bontas Simperl, C. Tempich, and M. Mochol, "Cost estimation for ontology development: Applying the ONTOCOM model," in *Technologies for Business Information Systems*, pp. 327–339, Dordrecht: Springer Netherlands, 2007. [Online].
[26] O. Corby, C. Faron-Zucker, and F. Gandon, "LDScript: A linked data script language," in *The Semantic Web – ISWC 2017*, (Cham), pp. 208–224, Springer International Publishing, 2017.
[27] E. Oren, B. Heitmann, and S. Decker, "ActiveRDF: Embedding semantic web data into object-oriented languages," *J. Web Semantics*, vol. 6, pp. 191–202, Sept. 2008. [Online].
[28] W. Muhamad, Suhardi, and Y. Bandung, "Transforming OpenAPI specification 3.0 documents into RDF-based semantic web services," *J. Big Data*, vol. 9, no. 1, p. 55, 2022. [Online].
[29] B. Varanasi and M. Bartkov, "HATEOAS," in *Spring REST: Building Java Microservices and Cloud Applications*, pp. 223–236, Berkeley, CA: Apress, 2022.
[30] M. Lanthaler and C. Gütl, "Hydra: A vocabulary for hypermedia-driven web APIs," in *Proc. Linked Data on the Web Workshop (LDOW) at WWW*, 2013. [Online].
[31] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. V. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common workflow language, v1.0," July 2016. [Online].
[32] R. Kahn and R. Wilensky, "A framework for distributed digital object services," *Int. J. Digital Libraries*, vol. 6, no. 2, pp. 115–123, 2006. [Online].
[33] E. Schultes and P. Wittenburg, "FAIR Principles and Digital Objects: Accelerating Convergence on a Data Infrastructure," in *Data Analytics and Management in Data Intensive Domains* (Y. Manolopoulos and S. Stupnikov, eds.), (Cham), pp. 3–16, Springer International Publishing, 2019.
[34] G. Berg-Cross, R. Ritz, and P. Wittenburg, "Data foundation and terminology work group products," 2015. [Online].
[35] T. Weigel, B. Plale, M. Parsons, G. Zhou, Y. Luo, U. Schwardmann, R. Quick, M. Hellström, and K. Kurakawa, "RDA recommendation on PID kernel information (final)," 2019. [Online].
[36] N. Blumenröhr, P.-J. Ost, F. Kraus, and A. Streit, "FAIR digital objects for the realization of globally aligned data spaces," in *Proc. IEEE Int. Conf. Big Data (BigData)*, (Washington, DC, USA), pp. 374–383, IEEE, Dec. 2024. Conference dates: Dec. 14–18, 2024.
[37] DONA Foundation, "Digital object interface protocol specification, version 2.0," 2018. [Online].

[38] U. Schwardmann, "Automated schema extraction for PID information types," in *Proc. IEEE Int. Conf. Big Data (BigData)*, pp. 3036–3044, 2016. [Online].
[39] T. Weigel, T. DiLauro, and T. Zastrow, "PID information types WG final deliverable," Oct. 2015. [Online].
[40] W. Claus, I. Sharif, D. Broeder, I. Anders, and P. Wittenburg, "FDO machine actionability," Nov. 2022. [Online].
[41] N. Blumenröhr, J. Böhm, P. Ost, M. Kulüke, P. Wittenburg, C. Blanchi, S. Bingert, and U. Schwardmann, "A comparative analysis of modeling approaches for the association of FAIR digital objects operations," *Data Sci. J.*, vol. 24, Aug. 2025. [Online].
[42] M. Inckmann, N. Blumenröhr, and R. Aversa, "Towards Machine-actionable FAIR Digital Objects with a Typing Model that Enables Operations," in *2025 IEEE International Conference on eScience (eScience)*, pp. 111–120, Sept. 2025. ISSN: 2325–3703.
[43] L. Lannom, D. Broeder, and G. Manepalli, "RDA data type registries working group output," Apr. 2015. [Online].
[44] N. Blumenröhr and R. Aversa, "From implementation to application: FAIR digital objects for training data composition," *Res. Ideas Outcomes*, vol. 9, p. e108706, Aug. 2023. [Online].
[45] S. Islam, J. Beach, E. Ellwood, J. Fortes, L. Lannom, G. Nelson, and B. Plale, "Assessing the FAIR digital object framework for global biodiversity research," *Res. Ideas Outcomes*, vol. 9, Sept. 2023. [Online].
[46] H. tom Wörden, T. Fitschen, S. Bingert, and T. Kálmán, "Meet the FDO manager: Reference implementation of FDO operations (Create, Retrieve, Update, Delete) based on the digital object interface protocol (DOIP)," *Open Conf. Proc.*, vol. 5, 2024. [Online].
[47] L. O. Bonino da Silva Santos, T. P. Sales, C. M. Fonseca, and G. Guizzardi, "Towards a conceptual model for the FAIR digital object framework," in *Formal Ontology in Information Systems*, pp. 227–241, IOS Press, 2023.
[48] S. Soiland-Reyes, P. Sefton, S. Leo, L. J. Castro, C. Weiland, and H. Van De Sompel, "Practical webby FDOs with RO-Crate and FAIR signposting: Experiences and lessons learned," in *Open Conf. Proc.: Int. FAIR Digital Objects Implementation Summit 2024*, TIB Open Publishing, Apr. 2024. [Online].
[49] U. Schwardmann, "Digital objects-FAIR digital objects: Which services are required?," *Data Sci. J.*, vol. 19, 2020.
[50] T. Jejkal, A. Pfeil, J. Schweikert, A. Pirogov, P. Barranco, F. Krebs, C. Koch, G. Guenther, C. Curdt, and M.Weinelt, "Realizing FAIR digital objects for the german helmholtz association of research centres," *Res. Ideas Outcomes*, vol. 8, p. e94758, Oct. 2022. [Online].
[51] Z. Mayer, J. Kahn, M. Götz, Y. Hou, T. Beiersdörfer, N. Blumenröhr, R. Volk, A. Streit, and F. Schultmann, "Thermal bridges on building rooftops," *Sci. Data*, vol. 10, no. 1, p. 268, 2023.
[52] Z. Mayer, J. Kahn, Y. Hou, T. Beiersdörfer, N. Blumenröhr, M. Götz, and R. Volk, "Thermal bridges on building rooftops-hyperspectral (RGB + thermal + height) drone images of karlsruhe, germany, with thermal bridge annotations," 2022. [Dataset]. [Online].
[53] F. Kraus, N. Blumenröhr, G. Götzelmann, D. Tonne, and A. Streit, "A gold standard benchmark dataset for digital humanities," in *Proc. Int. Workshop on Ontology Matching*, vol. 3897 of *CEUR Workshop Proceedings*, (Baltimore, USA), pp. 1–17, CEUR, Nov. 2024. [Online].
[54] D. Montag, "Understanding neo4j scalability," white paper, Neo Technology, Jan. 2013. Accessed: Sep. 27, 2025. [Online].
[55] R. Tupelo-Schneck, "An introduction to cordra," *Res. Ideas Outcomes*, vol. 8, p. e95966, Oct. 2022. [Online].

[56] N. Blumenröhr and F. Kraus, "Leveraging large language models for processing and evaluating FAIR digital objects," in *Proceedings of the 2nd International Workshop on Natural Scientific Language Processing and Research Knowledge Graphs (NSLP 2025), co-located with ESWC 2025*, vol. 3977 of *CEUR Workshop Proceedings*, (Portorož, Slovenia), CEUR-WS.org, June 2025.

[57] H. Panetto and J. Cecil, "Information systems for enterprise integration, interoperability and networking: Theory and applications," *Enterp. Inf. Syst.*, vol. 7, pp. 1–6, Feb. 2013. [Online].

APPENDIX A. ALGORITHM EXAMPLES

Listing 6 provides a minimal generic example for Algorithm 1.

Listing 6. Mapping of condition object C_o to record set R_f.

```
C_o = {
   "condition1": [["recordPIT1", "value1"],
   "recordPIT2"], "condition2": [["recordPIT1", "value2"], "recordPIT2"]
}

R_f = {
   "recordPIT1": ["value1"],
   "recordPIT2": ["value3"]
}

ReturnValue = true
```

Listing 7 provides a minimal generic example for Algorithm 2.

Listing 7. Mapping of execution-protocol e_{f_o} and FDO record R_{f_t} to an *execution_map*.

```
e_fo = {
"execution_protocol_type": [
   ["protocol_parameter_PIT1", "[key1, {
      [sub_protocol_parameter_PIT1, [key2, target_record_PIT1,
         [true, prefix: '[', delimiter: ', ' , suffix: ']']
      ]]
   }]"],
   ["protocol_parameter_PIT2", "[key3, target_record_PIT2, false]"],
   ["protocol_parameter_PIT3", "[key4, value3]"]
   ]
}

R_ft = {
   "record_PIT1": ["value1", "value2"],
   "record_PIT2": ["value4", "value5"],
}

client_input = {
   "protocol_parameter_PIT3": "value3 additionalInput"
}
```

Listing 7. *Continued.*

```
execution map = {
    "request_index1": {
        "execution_protocol_type": [
            ["protocol_parameter_PIT1", "[key1: {
                request_index1: {
                    sub_execution_protocol_type: {
                        [sub_protocol_parameter_PIT1, [key2: [value1, value2]]],
                    }
                }
            }]"],
            ["protocol_parameter_PIT2", "[key3: value4]"],
            ["protocol_parameter_PIT3", "[key4: value3 additionalInput]"]
        ]
    },
    "request_index2": {
        "execution_protocol_type": [
            ["protocol_parameter_PIT1", "[key1: {
                request_index1: {
                    sub_execution_protocol_type: {
                        [sub_protocol_parameter_PIT1, [key2: [value1, value2]]],
                    }
                }
            }]"],
            ["protocol_parameter_PIT2", "[key3: value5]"],
            ["protocol_parameter_PIT3", "[key4: value3 additionalInput]"]
        ]
    }
}
```

APPENDIX B. NEO4J APOC RULES

Listing 8. APOC triggers for contradiction rules.

```
/* Prereqs: APOC installed, apoc.trigger.enabled=true */

/* Forbid both HAS_METADATA and IS_METADATA_FOR between
same pair */
CALL apoc.trigger.add('no_conflicting_metadata_edges', "
    MATCH (a:orig_FDO)-[:HAS_METADATA]-> (b:orig_FDO)
    MATCH (a)-[:IS_METADATA_FOR]->(b)
    WITH count(*) AS c
    CALL apoc.util.validate(c>0, 'HAS_METADATA and
    IS_METADATA_FOR cannot coexist', [])
    YIELD value RETURN value
", phase: 'before');
```

APPENDIX C. EXEMPLARY DOIP/HTTP REQUESTS AND RESPONSES FOR OPERATIONS

Request for List Ops on the DOIP service (Listing 9), listed as steps 1-3 in Figure 3.

Listing 9. DOIP request for LIST Ops.

```
GET .../doip?operationId=0.DOIP/Op.LIST_Ops targetId=service
Content-Type: appliaction/json;charset=utf-8

Response:

{
"available service operations": {
    "0.DOIP/Op.*": {
        "arguments": "*",
        "operationID": "Object",
        "response type": "JSON object or encoded binary data",
        "targetID": "Object"
    },
    "0.DOIP/Op.GET_FDO": {
        "arguments": "None",
        "operationID": "0.DOIP/Op.GET_FDO",
        "response type": "PID record",
        "targetID": "Object"
    },
```

Listing 9. *Continued.*

```
    "0.DOIP/Op.LIST_FDOs": {
        "arguments": "Elastic Search keywords",
        "operationID": "0.DOIP/Op.LIST_FDOs",
        "response type": "array of FDO PIDs",
        "targetID": "Service"
    },
    "0.DOIP/Op.LIST_Ops": {
        "arguments": "None",
        "operationID": "0.DOIP/Op.LIST_Ops",
        "response type": "map of service operation specifications or map
            of supported FDO Operations for the target object",
        "targetID": "Service or Object"
    },
    "0.DOIP/Op.GET_RELATED_FDOs": {
        "arguments": "None",
        "operationID": "0.DOIP/Op.GET_RELATED_FDOs",
        "response type": "array of FDO PIDs",
        "targetID": "Object"
        }
    }
}
```

Request for List FDOs, i.e., Search FDO Records operation (s. Listing 10), listed as steps 4-6 in Figure 3 and illustrated in Listing 1. Further request examples will be performed based on an example PID (i.e., sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa) representing a containerized drone image dataset from TBBR (s. Section 4.4.2).

Listing 10. DOIP request for LIST FDOs.

```
POST /doip?operationId=0.DOIP/List_FDOs
&targetId=service&attributes.full=true
Content-Type: application/json;charset=utf-8

{“digitalObjectType”: [“rdf+xml”, “npy-image”],
“licenseURL”: “CC-BY”}

Response:
{
   “available FDOs”: [
      {
        “created”: “2025-04-09T10:49:01.567743Z”,
        “modified”: “2025-04-09T10:49:01.567743Z”,
        “pid”: “sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa"
      }, …
   ]
}
```

Request for Get_FDO operation (Listing 11), intermediate step not explicitly listed in Figure 3.

Listing 11. DOIP request for GET FDO.

```
GET …/doip?operationId=0.DOIP/Op.GET_FDO&targetId=sandboxed/fa798f80-a6db-4ce5-aab2-
03188abd80aa
Content-Type: appliaction/json;charset=utf-8

Response:
{
"pid": "sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa",
"entries": {
   "21.T11148/076759916209e5d62bd5": [
      {
         "key": "21.T11148/076759916209e5d62bd5",
         "value": "21.T11148/2c3cafa4db3f3e1e51b3"
      }
   ],
   "21.T11148/1c699a5d1b4ad3ba4956": [
      {
         "key": "21.T11148/1c699a5d1b4ad3ba4956",
         "value": "application/zstd"
      },
      {
         "key": "21.T11148/1c699a5d1b4ad3ba4956",
         "value": "application/x-npy-image"
         }
      ],
   "21.T11148/2f314c8fe5fb6a0063a8": [
      {
         "key": "21.T11148/2f314c8fe5fb6a0063a8",
         "value": "https://creativecommons.org/licenses/by/4.0/"
      }
   ],
   ...
   }
}
```

Request for LIST FDO Operations operation (Listing 12) listed as steps 7-9 in Figure 3 and illustrated in Listing 2. The names are taken from the respective information record, assuming that the service is configured with the corresponding PIT.

Listing 12. DOIP request for GET FDO.

```
GET …/doip?operationId=0.DOIP/Op.LIST_Ops&targetId=sandboxed/fa798f80-a6db-4ce5-aab2-
   03188abd80aa
Content-Type: appliaction/json;charset=utf-8

Response:

{
   “available FDO Operations”: [
      {
         “name”: “GET related FDOs”,
         “(p)id”: “0.DOIP/Op.GET_Related_FDOs”
      },
      {
         “name”: “GET related terms”,
         “(p)id”: “sandboxed/547c8a14-17aa-45dc-8ac5-7642cbb5c312”
      },
      {
         “name”: “Convert Numpy to PNG”,
         “(p)id”: “sandboxed/c4cb6095-ca69-452c-86de-74bf1971e8f1”
      }, …
   ]
}
```

Request for GET Related FDOs operation Listing 13 listed as steps 10-12 in Figure 3 and illustrated in Listing 3.

Listing 13. DOIP request for GET RELATED FDOs.

```
GET …/doip?operationId=0.DOIP/Op.GET_RELATED_FDOs&targetId=sandboxed/fa798f80-a6db-
    4ce5-aab2-03188abd80aa
Content-Type: appliaction/json;charset=utf-8

Response:

{
     "related FDOs for sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa": [
     "sandboxed/21cad3fa-40eb-4b75-a087-7a463f673619", …
     ]
}
```

Request for Get Related Terms operation (*Operation FDO*) Listing 14 listed as steps 13-20 in Figure 3 and illustrated in Listing 4.

Listing 14. DOIP request for Get Related Terms.

```
POST …/doip?operationId=sandboxed/547c8a14-17aa-45dc-8ac5-7642cbb5c312&
    targetId=sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa&attributes.
    full=true
Content-Type: appliaction/json;charset=utf-8
{
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT?prefLabel
WHERE {
<21.T11148/0c055f37730ce8b376e5> skos:prefLabel ?prefLabel .
}

Response:

{ZIP file downloaded as related_concepts_for_fa798f80-a6db-4ce5-aab2-03188abd80aa.zip}

Content:
text file containing:
"Remote sensing" @en
"Télédétection" @fr
"Teledetección" @es
…
```

Request for Convert Numpy to PNG operation (*Operation FDO*) listed as steps 13-16 in Figure 3 and illustrated in Listing 5.

```
GET …/doip?operationId=sandboxed/c4cb6095-ca69-452c-86de-74bf1971e8f1&
    targetId=sandboxed/fa798f80-a6db-4ce5-aab2-03188abd80aa
Content-Type: appliaction/json;charset=utf-8

Response:

{ZIP file downloaded as convert_numpy_to_png_for_fa798f80-a6db-4ce5-aab2-03188abd80aa.
    zip}
Content:
image files:
DJI_0004_R.png
DJI_0006_R.png
…
```