FAIR **DIGITAL OBJECTS** FORUM

Open FDO Expert Workshop
Relation between FDOs and Operations
16/17. February 2023

Date: 16.2. & 17.2. 2023 at 15.00 CET (14.00 UCT)
Location: https://us02web.zoom.us/j/4093556900

Convener:
George Strawn, Christophe Blanchi, Larry Lannom, Ulrich Schwardmann, Peter Wittenburg

The term FAIR Digital Object (FDO) implies an "object-oriented" approach, but due to the layered structure of FDOs the relationship is not that straightforward as it may sound. The 2-days workshop will try to shed light on the possible relations between FDOs and operations. Everyone interested is welcome to participate in this open workshop, to comment, raise questions, etc.

The tentative agenda for the workshop will be as follows:

day 1:	Intentions and Challenges from the FDO perspective
	Implementing the OO Concept (OOP, Object DB, Object Stores, etc.)
	The DOIP concept of relating to operations
	Operations in the realm of I4.0 AAS
day 2:	concrete FDO implementation cases (FZJ, IU, DISSCO, DKRZ, NIST, KIT, etc.)
	summary & conclusions

For some thoughts about the relation see the appended note.

Note on the Relation between FDOs and Operations

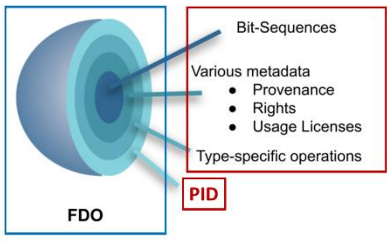
Peter, George

This note contains a few thoughts about the relation between FDOs and Operations. It is meant to stimulate discussions. We will put this into a Google doc so that others can add examples, elaborations and questions.

1. Background

We have an ongoing discussion about the relation between "operations" and FDOs and some feel that we yet did not sort this topic out to our satisfaction[1]. We have at least 5 cases where operations are addressed:

- in several papers we argue that FDOs support encapsulation and in our stylized atomic diagram we refer to operations
- in the type discussion we speak about operations that can be related to types with the relation being managed by users
- in the model used by DISSCO, values of PID kernel attributes point to operations that can be executed
- in DOIP, we speak about internal (CRUD) and external operations that can be linked with a call

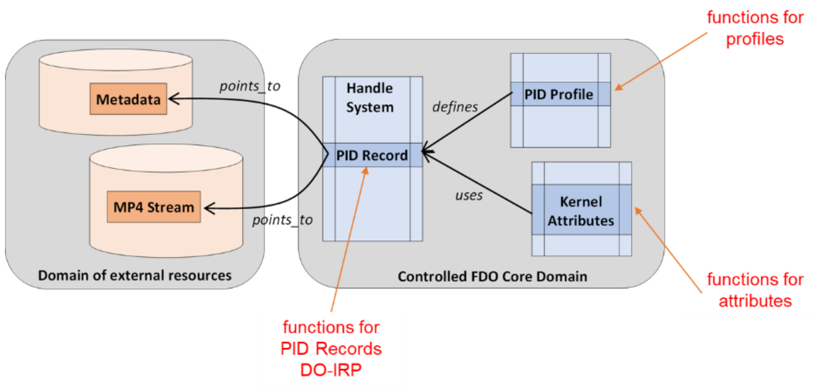


In addition, when discussing the implementing of industrial use cases, we realized that we need to define what we may call "common FDO operations".

The FDO model includes 4 layers which we need to look at when talking about the relationship:

- The PID layer
- The FDO model layer
- The layer of external resources (data, metadata, rights, etc.)
- The associated operations

The diagram indicates some details of these layers. The PID layer consists of a PID system that enables to resolve a specific PID to a set of attribute-value pairs using a PID interface protocol. The FDO model



layer basically defines profiles of the PID record, kernel attributes to be used in the profiles and FDO

[1] In this elaboration document we focus on the DO-approach, however, similar statements could be made for the Signposting approach.

types which can be used to link to operations. These two layers are defined by the FDO requirement specifications. The attribute-value pairs refer to external resources such as metadata and data which are hosted by trustworthy resource providers, i.e., they are managed by external organisations. We need to accept that there are hundreds/thousands of repositories that were developed during the last decades partly with specific applications in mind.

When we talk about the relation between FDOs and operations we need to distinguish between these layers. Questions as listed below need to be addressed in detail:

- how to restrict access to resources to certified operations if needed?
- how to issue a complex workflow process and assigning a special service?
- what kind of access protocol do we need, is DOIP sufficient and helpful?
- on purpose DOIP is simple, but where to put complexity?
- etc.

The recent discussions within the FDO Forum's working groups have shown that there are still many open questions about the exact nature of relations between FDOs and operations given the layers of the model.

2. Encapsulation

Encapsulation (the provision of an interface for a piece of software or hardware to allow or simplify access for the user) was utilized to define "Abstract Data Types" (ADTs) in the 1970s. The designer of an ADT provides a set of functions (operations) that need to be used to access and alter the state of the information (the bit string) of an ADT. The designer does not inform the outside world how the ADT is constructed. Therefore, the user can only access and manipulate the state of an ADT by using the set of provided functions.

On the other hand, we can have other types, including the MIME types (concrete data types such as csv and mp4), which permit user access to the bit string itself. That is, the user can construct operations on them that access and manipulate the bit string directly.

There are several advantages to abstract data types. First, multiple implementations of the same ADT can be constructed, either in parallel or in serial, that would be indistinguishable to users. Second, the user cannot carry out unwanted operations on the internals of an abstract ADT.

Does the FDO model now support this kind of encapsulation and does it increase security by supporting mechanisms to restrict access to data as is required for sensitive data in medicine etc.? Does the DO Interfacing Protocol (DOIP) support a unique way to apply operations to digital objects stored somewhere?

3. FDOs

3.1 PID level

This is comparatively simple and only includes the resolution of a PID into its record, which is guided by the DO-IRP protocol. DO-IRP also specifies how a PID with its PID record can be created, including the packaging of the kernel attributes and the rights.

3.2 FDO Level

The term "encapsulation" is valid for the FDO model level operations, which cover the resources which are part of the FDO Core Domain

- PID profiles (also called FDO Profiles)
- PID Kernel Attributes and value constraints

For all interactions with the PID system, as described above, we will make use of DO-IRP as a formal basis implementing encapsulation.

Dependent on the FDO Genre (characterizing the configuration type and the class of type of the included bit-sequence such as including a piece of software as bit-sequence or even the definition of a PID profile, for example) PID Profiles describe the set of attributes being used by a certain FDO provider. In the case that the FDO includes for example an mp4 stream as bit-sequence and some metadata the PID record might contain the following attribute-value pairs:

Attribute	value	mandatory
PID-Profile	<reference to the PID profile>	y
genre	"FDO-with-data"	y
type	"mp4"	y (since genre indicates data)
cksm	<some string>	n
size	<some number>	n
path-to-bit-sequence	<PID>/<URL>/<query>	y (at least one path)
path-to-bit-sequence	<PID>/<URL>/<query>	n
path-to-metadata	<PID>/<URL>/<query>	y
owner	<PID>	n (some ref such as ORCID)
other	<some>	in case of other metadata

All operations that influence in some way the FDO level details need to be encapsulated ensuring that only authorized actors can make valid changes.

Example: with the help of an operation, the bit sequence has been copied to a second location as indicated in the table. The authorities managing the second location are not allowed to change the PID record directly, i.e., they need to invoke a pre-defined method which might be called "add_or_change_location_to_PID-Record" which carries out all the required checks and actually carries out this special change of the PID record.

The following standard functions need to be provided:

- register a PID with a PID Record according to a specified profile
- change "attribute, value"
- add "attribute, value"
- delete "attribute, value"
- check_FAIR_compliance

At this FDO level there is encapsulation of the information that for example points to the bit-sequence.

3.4 Resource Level

Dealing with the resource layer is not trivial and it is not possible to speak about "encapsulation" at this layer, since the bit string may be directly accessed using functionality offered by the repository. We need to include a variety of possible resources that are associated with the FDO but are managed in different ways:

- Bit-Sequence (such as mp4 or cvs)
- Metadata (DC, Scientific, Provenance, etc.)
- Rights Record (formal)
- Usage License (formal)
- Transaction Record
- etc. (there can be many more different types of bit sequences included)

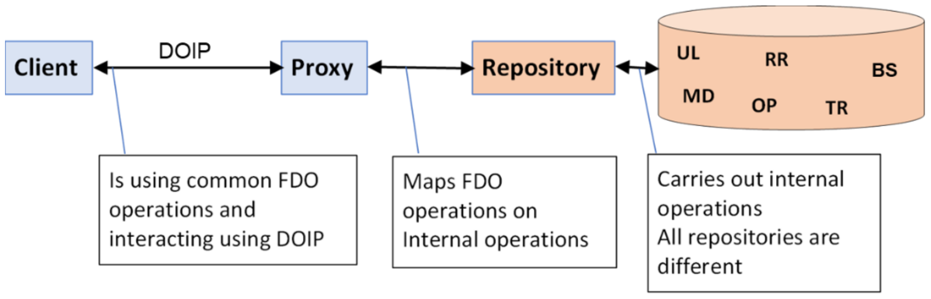
In the DISSCO example the PID Record of an object includes for example attributes which are references to operations such as

- give me access to the High-Resolution image in code format X
- give me access to the map indicating where the specimen object was found
- give me access to the gene data

In the DISSCO case the PID record does not include references to these resource types (bit sequences), but references to operations that visualize these resources for example, hence in this we can speak about encapsulation of these specific bit-sequences. We cannot make statements about how the different repositories create, manipulate or use the different bit-sequences.

3.5 DOIP

DOIP is a protocol for interacting with a Digital Object. It contains a target PID indicating the (FAIR) Digital Object being addressed and a specification of the operation ID to be performed which is being provided by some experts. The general layout is schematically indicated in the diagram. There are huge


Client
DOIP
Proxy
Repository
UL
RR
BS
MD
OP
TR
Is using common FDO operations and interacting using DOIP
Maps FDO operations on Internal operations
Carries out internal operations
All repositories are different

numbers of repositories which have implemented a variety of access operations over the years. The proxy needs to map DOIP calls to these operations so that the client can execute operations in the same way.

A typical DOIP request packet is given below. All items are uniquely specified by PIDs, i.e., it puts global unique and resolvable persistent identifiers, as for example provided by Handles, into the core. With respect to operations, it should be mentioned that DOIP comes with basic operations to create, read, manipulate and delete FDOs, however, any other operation defined by users can be linked as well.

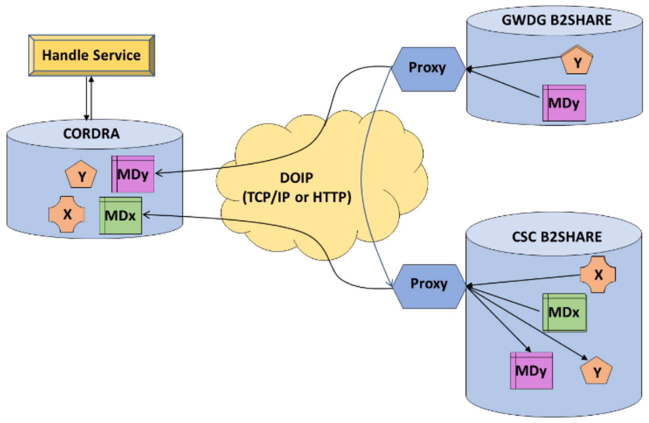
The following set attributes need to be provided in a request package:

- **requestId**: The identifier of the request provided by the client to keep track of responses.
- **clientId**: The identifier of the requesting client.
- **targetId**: The identifier of the DO on which to perform the operations
- **operationId**: the identifier of the operation to be performed on the DO.
- **attributes(optional)**: Stipulated by the Operation ID.
- **authentication(optional)**: used by clients to authenticate.
- **input(optional)**: Payload stipulated by the operation.

This implies that many of the access operations that have been developed for existing repositories would include an **operationId** pointing to an operation registered in a registry.

DOIP Example

Recently in a small project the CORDRA and B2Share repositories were connected using the DOIP protocol, i.e., a proxy had to be developed that allows to access the resources stored in B2Share to be extracted as indicated schematically in the drawing. B2Share is offering an API that allows a client to access a specific bit-sequence identified by a PID and its associated metadata.



The functions that were implemented using DOIP allowed to create copies of the bit sequence and the metadata objects, but it was not implemented that they exist on CORDRA as one FDO where the PID record includes references to both resources. Doing it this way the basic CRUD operations can be used treating each object separately. To create an FDO at the CORDRA side some more logic would be required in the proxy. Moving an FDO from CORDRA to B2Share would require calling an external operation to support the required functionality to create a B2Share object.

The question is raised where the complex operations need to be located – in the proxy or in a client?

4. Standardization of Common FDO Operations

When we were discussing the German FDO based project and its different use cases, we were confronted with the fact that all partners already have their repositories, their metadata schemes etc. and there are many differences in the way the repositories are being set up, which metadata schemes are being used, etc. Nevertheless, at an abstract level there are a couple of functions that are being used by almost all repositories. Let’s call them “common FDO operations”.

Let’s first sketch the scenario that a client is interacting with an FDO being managed in some repository by making use of a proxy which “simulates” an ideal FDO. All commands issued by the client to FDOs are mapped to local commands known by the repository as shown in the diagram. The client at first instance does not know which operations the repository is supporting, or which are available for a given type. It could request to send the list of operations that are supported (as indicated above in the skin detector example), but that does not help since all repositories and registries may use different names, APIs etc. The client could not interpret the returned set of values of the “list operations” command and would have to handover the interaction to a human user which would stop automatic behavior.

We need to define the set of basic operations one can execute on an FDO – let’s call them common operations. The person developing the proxy needs to map these common operations to the set of operations supported by the repository or registered for this type where possible. This finally would pave the way towards automation where no human interpretation of the set of returned operations is required – at least for the common cases.

In the following list we indicate a first set of such possible common operations that could be supported:

- list of operations
- create FDO
- create special FDO
- get/mod/del BS
- get/change Metadata (various?)
- get/change Access_rights

- get/change Usage_Licenses
- get/add Transaction_records
- change PID record (see chapter 2)
- ?

The implementation could be more efficient by not formulating get/change/etc. operations for any type of information associated with the FDO, but by 1) requesting the profile which delivers the kernel attributes and 2) use the kernel attribute name in a generic request.

- get PID_profile (kernel_attributes, ...)
- get info ("kernel-attribute", ...)

5. Automation with External Operations

We have seen above that we cannot encapsulate external operations on FDOs as part of the controlled FDO core domain and the number of common operations at repository level is limited. In the example in section 3 it was described that for a given task operations fulfilling certain requirements would be searched, selected, and executed. An important question is how this can be made automatically or at least how much this can be supported by automated processes.

Operations in computer science, as used inside programming languages or application systems, are defined by the type of their input parameters and of their return value(s), as well as by their code. Usually in these cases types are declared with type definitions that are given implicitly. The code is made explicit only inside the description language of the used programming language or application system. And the access to an execution process is provided by some runtime environment chosen by the user or application platform. To make these more or less implicit definitions and decisions applicable in a global framework as it is described for FDOs, all these components and interfaces need to be made explicit and accessible under well described access conditions.

The urgent need for such operations in a global framework is shown by the long-term success of the REST protocol based on HTTP. For an automation of the example in section 3 a search for operations only based on requirements on the type of their input parameters and of their return value(s) would however be needed for instance. But this is still not possible in the HTTP framework, because there is no universally available typing system for (complex) parameters as usually needed in such cases. But FDOs are intrinsically based on such a typing concept.

Whether an operation is applicable to a given FDO can be decided by syntactic constraints on the type of input requirements of the operation and the type attributes provided by the FDO. Whether the result of the operation also complies to the expected type of the wanted transformation can also be decided at a syntactical level. Therefore, a search inside operation repositories can be based on type requirements for the input and output of operations and would lead to operations that are applicable for a specified transformation.

For the execution an entry point to the operation in an application framework is needed and the execution of the function requires access for the executing instance to all needed input data. DOIP provides major components of such an application framework for the execution of operations on digital objects.

Whether a specific operation is also productive in the sense that the result is actually the wanted result, is in general a semantic question that cannot be fully answered by syntactic constraints. Even if semantic constraints can be partly covered by the use of keywords, in many cases there will be an intermediate human interaction step of selection necessary. But for the user the focused view on directly applicable operations has already a high value and will lead to a new level of productivity in data management.

6. Examples of a Complex Operation

One example will be discussed here to elaborate on operations. In the MPI lab gesture recognition in mp4 videos was carried out based on skin color detection. Such recognition is a layered and complex process which starts with detecting pixels that most probably include skin fragments based on

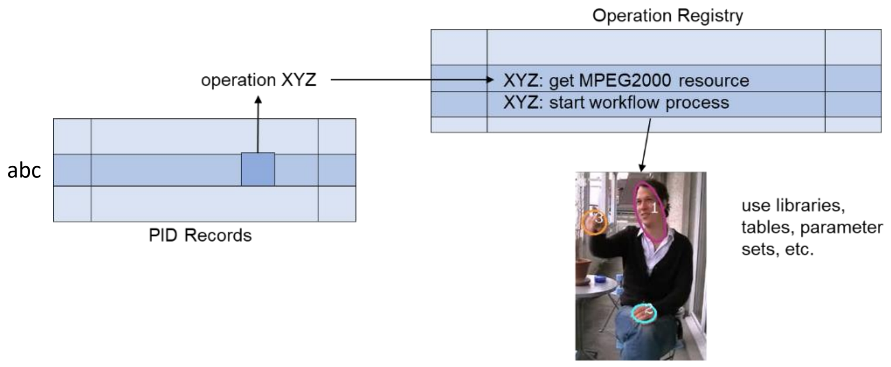
specifying a tolerance in the color information, i.e., the basic process uses mp4 libraries to operate on pixels and labels them per video frame over time. Further processing is being done on these marked areas to detect stable areas of scin color slowly moving over frames. The whole recognition package then creates overlays which one can supersede on the video such as shown in the diagram where in this frame 3 areas with skin color were detected.

A corresponding DOIP call could request to use the skin color detector being registered somewhere and having the PID "xyz" and to operate on "myvideo.mp4" with the PID "abc". Thus, we would execute an "external operation" created by some recognition experts. In addition, we do not know what kind of access to "myvideo.mp4" the corresponding repository offers for other users. Therefore, we cannot speak about "encapsulation" of the video resource.

This means that in general one could reference "external" operations with the DOIP call which are defined by someone and registered somewhere. A set of operations would be required to execute external operations like the one discussed above if "myvideo.mp4" would be selected:

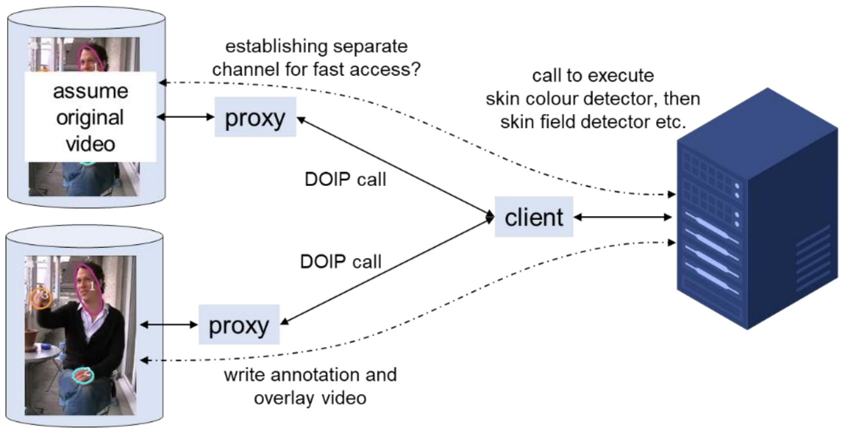
- give me a list of possible operations for the specified type (which is mp4) which would result in a search in a registry with relations between types and operations
- an operation such as motion detection based on skin colour would be selected
- carry out the operation on the target FDO

These "external" operations can be of any type to create, manipulate and analyse any of the bit-sequences associated with the FDO. In all these aspects we cannot speak about "encapsulation" since the resource provider does not define the possible operations. This is like the processing of MIME types. The following diagram indicates the use of an operation registry that knows about two operations on an MPEG2000 type of bit-sequence, for example.


Operation Registry
operation XYZ
XYZ: get MPEG2000 resource
XYZ: start workflow process
abc
PID Records
use libraries, tables, parameter sets, etc.

In the case of motion detection we would start a multistep workflow process which would use a high resolution video recording, sets of parameters being used during the process, annotations (being created during the process), creating overlays on a reduced video recording (mp4), metadata of different types and in principle also information about rights and usage options.

What would this now mean if we would use DOIP to carry out motion detection on a specific video? One possibility is sketched in the diagram below. A client would have to be developed that includes quite some logic about the process.


assume original video
establishing separate channel for fast access?
proxy
call to execute skin colour detector, then skin field detector etc.
DOIP call
client
DOIP call
proxy
write annotation and overlay video

Several questions need to be answered:

- Are there better ways to use DOIP in such a case?
- Would we be able to restrict access to the resources by using only the set of operations defined by the resource owners?
- What does DOIP help in such a complex case?

7. Summary

In this short document we are discussing the relation between FDOs and Operations looking at a few aspects and distinguishing the FDO layers. We indicate a couple of questions where clarification is required.

The major two questions are:

- Where to put complexity of operations given a simple interaction protocol?
- Can we implement strong or weak kinds of encapsulation using the FDO concept which tightly binds the rights on data?