Evaluating FAIR Digital Object as a distributed object system

This manuscript (permalink) was automatically generated from stain/2022-fdo-paper@b046b0b on May 9, 2022.

Authors

- **Stian Soiland-Reyes**
  0000-0001-9842-9718 · stain · soilandreyes
  Department of Computer Science, The University of Manchester, UK; Informatics Institute, Faculty of Science, University of Amsterdam, NL · Funded by BioExcel-2 (European Commission H2020-INFRAEDI-2018-1 823830); BY-COVID (European Commission HORIZON-INFRA-2021-EMERGENCY-01 101046203)

- **Carole Goble**
  0000-0003-1219-2137 · carolegoble · CaroleAnneGoble
  Department of Computer Science, The University of Manchester, UK · Funded by BioExcel-2 (European Commission H2020-INFRAEDI-2018-1 823830); EOSC-Life (European Commission H2020-INFRAEOSC-2018-2 824087); BY-COVID (European Commission HORIZON-INFRA-2021-EMERGENCY-01 101046203)

- **Paul Groth**
  0000-0003-0183-6910 · pgroth · pgroth
  Informatics Institute, Faculty of Science, University of Amsterdam, NL

Abstract

Interoperability Framework for Fast Data

Considering FDO/Web as interoperability framework for Fast Data

Table 1: Considering FDO and Web according to the levels of interoperability [1]:

Quality	FDO w/ DOIP	Web w/ Linked Data
Symbiotic: *to what extent multiple applications can agree to interact/align/collaborate/cooperate*	Purpose of FDO is to enable federated machine actionable digital objects for scholarly purposes, in practice this also requires agreement of or compatibility between FDO types. FDO encourages research communities to develop common type registries to be shared across instances. In current DOIP practice, each service have their own types, attributes and operations. The wider symbiosis is consistent use of PIDs.	Web is loosely coupled and encourages collaboration and linking by URL. In practice, REST APIs end up being mandated centrally by dominant (often commercial) providers, which clients are required to use as-is with special code per service. Use of Linked Data enables common tooling and semantic mapping across differences.
Pragmatic: *using interaction contracts so processes can be choreographed in workflows*	FDO types and operations enable detailed choreography (see CWFP). `0.TYPE/DOIPOperation` has lightweight definition of operation, e.g. `0.DOIP/Request` / `0.DOIP/Response` may give FDO Type (or any other kind of “specifics”). Semantics/purpose of operations not formalized (similar operations can be grouped with `0.DOIP/OperationReference`).	“Follow your nose” crawler navigation, which may lead to frequent dead ends. Operational composition, typically within a single API provider, by OpenAPI 3 [2], schema.org Actions [[3]), WSDL/SOAP [4]
Semantic: *ensuring consistent understanding of messages, interoperability of rules, knowledge and ontologies*	FDO semantic enable navigation and typing. Every FDO have a type. Types maintained in FDO Type registries, which may add additional semantics, e.g. the ePIC PID-InfoType for Model. No single type semantic, Type FDOs can link to existing vocabularies & ontologies. JSON-LD used within some FDO objects (e.g. DISSCO Digital Specimen, NIST Material Science schema) [5]	Lightweight HTTP semantics for authenticity/navigation. Semantic Type not commonly expressed on PID/header level, may be declared within Linked Data metadata. Semantic of type implied by Linked Data formats (e.g. OWL2, RDFS), although choice of type system may not be explicit.
Syntactic: *serializing messages for digital exchange, structure representation*	DOIP serialize FDOs as JSON, metadata commonly use JSON, typed with JSON Schema. Multiple byte stream attachments of any media type.	Textual HTTP headers (including any signposting), single byte stream of any media type, e.g. Linked Data formats (JSON-LD, Turtle, RDF/XML) or embedded in document (HTML with RDFa, JSON-LD or Microdata). XML previously main syntax used by APIs, JSON now dominant.
Connective: *transferring messages to another application, e.g. wrapping in other protocols*	DOIP [6] is transport-independent, commonly TLS TCP/IP port 9000), DOIP over HTTP	HTTP/1.1 (TCP/IP port 80), HTTP/1.1+TLS (TCP/IP 443), HTTP/2 (as HTTP/1* but binary), HTTP/3 (like HTTP/2+TLS but UDP)

Quality	FDO w/ DOIP	Web w/ Linked Data
Environmental: *how applications are deployed and affected by its environment, portability*	Main DOIP implementation is Cordra, which can be single-instance or distributed. Cordra storage backends include file system, S3, MongoDB (itself scalable). Unique DOIP protocol can be hard to add to existing Web application frameworks, although proxy services have been developed (e.g. B2SHARE adapter).	HTTP services widely deployed in a myriad of ways, ranging from single instance servers, horizontally & vertically scaled application servers, to (for static content) multi-cloud Content-Delivery Networks (CDN). Current scalable cloud technologies for Web hosting may not support HTTP features previously seen as important for Semantic Web, e.g. content negotiation and semantic HTTP status codes.

Mapping of Metamodel concepts:

Table 2: Mapping the Metamodel concepts from the Interoperability Framework for Fast Data [1] to equivalent concepts for FDO and Web:

Metamodel concept	FDO/DOIP concept	Web/LD concept
Resource	FDO/DO	Resource
Service	DOIP service	Server/endpoint
Transaction	(not supported)	Conditional requests, `409 Conflict`
Process	Extended operations	Primarily stateless, `100 Continue`, `202 Accepted`
Operation	DOIP Operation	Method
Request	Request	Request
Response	Response	Response
Message	Segment	Message, Representation
Channel	TCP/IP, TLS	TCP/IP, TLS, UDP
Protocol	DOIP 2.0, ++	HTTP/1.1, HTTP/2, HTTP/3
Link	PID/Handle	URL

A comparison framework for middleware infrastructures

Comparing FDO and Web as middleware infrastructures

Table 3: Comparing FAIR Digital Object (with the DOIP 2.0 protocol [6]) and Web technologies (using Linked Data) as middleware infrastructures [7]

Quality	FDO w/ DOIP	Web w/ Linked Data
Openness: *framework enable extension of applications*	FDOs can be cross-linked using PIDs, pointing to multiple FDO endpoints. Custom DOIP operations can be exposed, although it is unclear if these can be outside the FDO server. PID minting requires Handle.net prefix subscription, or use of services like Datacite, B2Handle.	The Web is inheritedly open and made by cross-linked URLs. Participation requires DNS domain purchase (many free alternatives also exists). PID minting can be free using CURL/ARK services, or can use DOI/Handle with HTTP redirects.

Quality	FDO w/ DOIP	Web w/ Linked Data
Scalability: *application should be effective at many different scales*	No defined methods for caching or mirroring, although this could be handled by backend, depending on exposed FDO operations (e.g. Cordra can scale to multiple backend nodes)	Cache control headers reduce repeated transfer and assist explicit and transparent proxies for speed-up. HTTP `GET` can be scaled to world-population-wide with Content-Delivery Networks (CDNs), while write-access scalability is typically manage by backend.
Performance: *efficient and predictable execution*	DOIP has been shown moderately scalable to 100 millions of objects, create operation at 900 requests/second [8]. DOIP protocol is serial and reusable, but multiple connections can be made. Setup is typically through TCP and TLS which adds latency.	HTTP traffic is about 10% of global Internet traffic, excluding video and social networks [{ 9}]. HTTP 1 connections are serial and reusable, and concurrent connections is common. HTTP/2 adds multiplexed streams [10] but still has TCP+TLS startup costs. For reduced latency [11], HTTP/3 [{ 12}] use QUIC [13]) rather than TCP, already adapted heavily (30% of EMEA traffic) of which Instagram & Facebook video is the majority of traffic.
Distribution transparency: *application perceived as a consistent whole rather than independent elements.*	Each FDO is accessed separately along with its components (typically from the same endpoint). FDOs should provide the mandatory kernel metadata fields. FDOs of the same declared type typically share additional attributes (although that schema may not be declared). DOIP does not enforce metadata typing constraints, this need to be established as FDO conventions.	Each URL accessed separately. Common HTTP headers provide basic metadata, although it is often not reliable. A multitude of schemas and serializations for metadata exists, conventions might be implied by a declared profile or certain media types. Metadata is not always machine findable, may need pre-agreed API URI Templates [14], content-negotiation [15] or FAIR Signposting [16].
Access transparency: *local/remote elements accessed similarly*	FDOs always accessed through PID indirection, but this means difficult to make private test setup.	Global HTTP protocol frequently used locally and behind firewalls, but at risk of non-global URIs (e.g. `http://localhost/object/1`) and SSL issues (e.g. self-signed certificates, local CAs)
Location transparency: *elements accessed without knowledge of physical location*	FDOs always accessed through PIDs. Multiple locations possible in Handle system, can expose geo-info.	PIDs and URL redirects. DNS aliases and IP routing can hide location. Geo-localized servers common for large cloud deployments.
Concurrency transparency: *concurrent processing without interference*	No explicit concurrency measures. FDO kernel metadata can include checksum and date.	HTTP operations are classified as being stateless/idempotent or not (e.g. `PUT` changes state, but can be repeated on failure), although these constraints are occassionally violated by Web applications. Cache control, `ETag` (~ checksum) and modification date in HTTP headers allows detection of concurrent changes on a single resource.
Failure transparency: *service provisioning resiliant to failures*	DOIP status codes, e.g. `0.DOIP/Status.104`, additional codes can be added as custom attributes	HTTP status codes e.g. `404 Not Found`, structured error documents in Open API (??)

Quality	**FDO w/ DOIP**	**Web w/ Linked Data**
Migration transparency: *allow relocating elements without interferring application*	Update of PID record URLs, indirection through `0.TYPE/DOIPServiceInfo` (not always used consistently). No redirection from DOIP service.	HTTP `30x` status codes provide temporary or permanent redirections, commonly used for CURLs but also by endpoints.
Persistence transparency: *conceal deactivation/reactivation of elements from their users*	FDO requires use of PIDs for object persistence, including a thumbstone response for deleted objects. There is no guarantee that an FDO is immutable or will even stay the same type (note: CORDRA extends DOIP with version tracking).	URLs are not required to persist, although encouraged [17]. Persistence requires convention to use PIDs/CURLs and HTTP `410 Gone`. An URL may change its content, change in type may sometimes force new URLs if exposing extensions like `.json`. Memento [18] expose versioned snapshots. WebDAV `VERSION-CONTROL` method [19] (used by SVN).
Transaction transparency: *coordinate execution of atomic/isolated transactions*	No transaction capabilities declared by FDO or DOIP. Internal synchronization possible in backend for Extended operations.	Limited transaction capabilities (e.g. `If-Unmodified-Since`) on same resource. WebDAV locking mechanisms [20] with `LOCK` and `UNLOCK` methods.
Modularity: *application as collection of connected/distributed elements*	FDOs are inheritedly modular using global PID spaces and their cross-references. In practice, FDOs of a given type are exposed through a single server shared within a particular community/institution.	The Web is inheritently modular in that distributed objects are cross-referenced within a global URI space. In practice, an API's set of resources will be exposed through a single HTTP service, but modularity enables fine-grained scalability in backend.
Encapsulation: *separate interface from implementation. Specify interface as contract, multiple implementations possible*	Indirection by PID gives separation. FDO principles are protocol independent, although it may be unclear which protocol to use for which FDO (although `0.DOIP/Transport` can be specified after already contacting DOIP). Cordra supports native DOIP, DOIP over HTTP and Cordra REST API)	HTTP/1.1 semantics can seemlessly upgrade to HTTP/2 and HTTP/3. `http` vs `https` URIs exposes encryption detail [1]. Implementation details may leak into URIs (e.g. `search.aspx`), countered by deliberate design of URI patterns [22] and PIDs via Persistent URLs (PURL).
Inheritance: *Deriving specialized interface from another type*	DOIP types nested with parents, implying shared FDO structures (unclear if operations are inherited). FDO establishes need for multiple Data Type Registries (e.g. managed by a community for a particular domain). Semantics of type system currently undefined for FDO and DOIP, syntactic types can also piggyback of FDO type's schema (e.g. CORDRA `$ref` use of JSON Schema references [23])	Syntactically Media Type with multiple suffixes [24] (mainly used with `+json`), declaration of subtypes as profiles (RFC6906) [25]. In metadata, semantic type systems (RDFS [26], OWL2 [27], SKOS [28]). OpenAPI 3 [2] inheritance and Polymorphism. XML `xsd:schemaLocation` & `xsd:type` [29], JSON `$schema` [23]), JSON-LD `@context` [30]. Large number of domain-specific and general ontologies define semantic types, but finding and selecting remains a challenge.
Signal interfaces: *asynchronous handling of messages*	DOIP 2.0 is synchronous, in FDO async operations undefined. Could be handled as custom jobs/futures FDOs	HTTP/2 multiplexed streams [10], Web Sockets [31], Linked Data Notifications [32], AtomPub [33], SWORD [34], Micropub, more typically ad-hoc jobs/futures REST resources

Quality	FDO w/ DOIP	Web w/ Linked Data
Operation interfaces: *defining operations possible on an instance, interface of request/response messages*	CRUD predefined in DOIP, custom operations through `0.DOIP/Op.ListOperations` (can be FDOs of type `0.TYPE/DOIPOperation`, more typically local identifiers like `"getProvenance"`)	CRUD predefined in HTTP methods [35], (extended by registration), URI Templates [14], OpenAPI operations [2], HATEOAS incl. schema.org Actions [[3]), JSON HAL [36] & Link headers (RFC8288) [37]
Stream interfaces: *operations that can handle continuous information streams*	Undefined in FDO. DOIP can support multiple byte stream elements (need custom FDO type to determine stream semantics)	HTTP 1.1 [38] chunked transfer, HLS (RFC8216) [39], MPEG-DASH (ISO/IEC 23009-1:2019) [40]

Assessing DOIP against FDO

Assessing FDO against FAIR

References

1. **An Interoperability Framework and Distributed Platform for Fast Data Applications**
José Carlos Martins Delgado
Data Science and Big Data Computing (2016) https://doi.org/gp3rds
DOI: 10.1007/978-3-319-31861-5_1

2. **OpenAPI Specification v3.1.0 | Introduction, Definitions, & More**
https://spec.openapis.org/oas/v3.1.0.html

3. **Schema.org Actions - schema.org** https://schema.org/docs/actions.html

4. **Web Services Description Language (WSDL) Version 2.0 Part 0: Primer**
http://www.w3.org/TR/wsdl20-primer/

5. **FAIR digital object demonstrators 2021**
Peter Wittenburg, Ivonne Anders, Christophe Blanchi, Merret Buurman, Carole Goble, Jonas Grieb, Alex Hardisty, Sharif Islam, Thomas Jejkal, Tibor Kálmán, … Philipp Wieder
Zenodo (2022) https://zenodo.org/record/5872645
DOI: 10.5281/zenodo.5872645

6. **Digital object interface protocol specification, version 2.0**
DONA Foundation
DONA foundation (2018-11-12) https://hdl.handle.net/0.DOIP/DOIPV2.0

7. **A Comparison Framework for Middleware Infrastructures.**
Apostolos Zarras
The Journal of Object Technology (2004) https://doi.org/cj5q8r
DOI: 10.5381/jot.2004.3.5.a2

8. https://www.rd-alliance.org/sites/default/files/Cordra.2022.pdf

9. **Global Internet Phenomena Report 2022**
Sandvine
https://www.sandvine.com/global-internet-phenomena-report-2022

10. **Hypertext Transfer Protocol Version 2 (HTTP/2)**
M Belshe, R Peon
RFC Editor (2015-05) https://doi.org/gp32q9
DOI: 10.17487/rfc7540

11. https://blog.cloudflare.com/http-3-vs-http-2/

12. **draft-ietf-quic-http-34** https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34

13. **QUIC: A UDP-Based Multiplexed and Secure Transport**
J Iyengar, M Thomson (editors)
RFC Editor (2021-05) https://doi.org/gkctrr
DOI: 10.17487/rfc9000

14. **URI Template**
J Gregorio, R Fielding, M Hadley, M Nottingham, D Orchard
RFC Editor (2012-03) https://doi.org/gp33dw
DOI: 10.17487/rfc6570

15. **Content negotiation - HTTP | MDN** https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation

16. **FAIR Signposting Profile - Signposting the Scholarly Web** https://signposting.org/FAIR/

17. **Hypertext Style: Cool URIs don't change.** https://www.w3.org/Provider/Style/URI

18. **HTTP Framework for Time-Based Access to Resource States -- Memento**
H Van de Sompel, M Nelson, R Sanderson
RFC Editor (2013-12) https://doi.org/ggqvps
DOI: 10.17487/rfc7089

19. **Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)**
G Clemm, J Amsden, T Ellison, C Kaler, J Whitehead
RFC Editor (2002-03) https://doi.org/gp37bd
DOI: 10.17487/rfc3253

20. **HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)**
L Dusseault (editor)
RFC Editor (2007-06) https://doi.org/gp37bf
DOI: 10.17487/rfc4918

21. **Upgrading to TLS Within HTTP/1.1**
R Khare, S Lawrence
RFC Editor (2000-05) https://doi.org/gp33dv
DOI: 10.17487/rfc2817

22. **Hypertext Style: Cool URIs don't change.** https://www.w3.org/Provider/Style/URI.html

23. **draft-bhutton-json-schema-00** https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-00

24. **draft-ietf-mediaman-suffixes-00 - Media Types with Multiple Suffixes**
https://datatracker.ietf.org/doc/draft-ietf-mediaman-suffixes/00/

25. **The 'profile' Link Relation Type**
E Wilde
RFC Editor (2013-03) https://doi.org/gp32q7
DOI: 10.17487/rfc6906

26. **RDF Schema 1.1** http://www.w3.org/TR/rdf-schema/

27. **OWL 2 Web Ontology Language Document Overview (Second Edition)**
http://www.w3.org/TR/owl2-overview/

28. **SKOS Simple Knowledge Organization System Reference** http://www.w3.org/TR/skos-reference/

29. **W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures**
http://www.w3.org/TR/xmlschema11-1/

30. **JSON-LD 1.1** http://www.w3.org/TR/json-ld/

31. **WebSockets Standard** https://websockets.spec.whatwg.org/

32. **Linked Data Notifications** https://www.w3.org/TR/ldn/

33. **The Atom Publishing Protocol**
J Gregorio, B de hOra (editors)
RFC Editor (2007-10) https://doi.org/gp4p2c
DOI: 10.17487/rfc5023

34. **SWORD 3.0 Specification** https://swordapp.github.io/swordv3/swordv3.html

35. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**
R Fielding, J Reschke (editors)
RFC Editor (2014-06) https://doi.org/gh4jxc
DOI: 10.17487/rfc7231

36. **draft-kelly-json-hal-08** https://datatracker.ietf.org/doc/html/draft-kelly-json-hal-08

37. **Web Linking**
M Nottingham
RFC Editor (2017-10) https://doi.org/gf8jcd
DOI: 10.17487/rfc8288

38. **Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**
R Fielding, J Reschke (editors)
RFC Editor (2014-06) https://doi.org/gp32q8
DOI: 10.17487/rfc7230

39. **HTTP Live Streaming**
W May
RFC Editor (2017-08) https://doi.org/gp32rc
DOI: 10.17487/rfc8216

40. **ISO/IEC 23009-1:2019**
14:00-17:00
ISO
https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/93/79329.html

---

1. The `http` protocol (port 80) can in theory also upgrade [21] to TLS encryption, as commonly used by Internet Printing Protocol for `ipp` URIs, but on the Web, best practice is explicit `https` (port 443) URLs to ensure following links stay secure.↩