Packaging research data with RO-Crate

This manuscript (permalink) was automatically generated from stain/ro-crate-paper@d3552f3 on May 3, 2021.

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-02-2018-823830); EOSC-Life (European Commission H2020-xx-824087)

- **Peter Sefton**
  0000-0002-3545-944X · ptsefton
  Faculty of Science, University Technology Sydney, Australia

- **Carole Goble**
  0000-0003-1219-2137 · carolegoble
  Department of Computer Science, The University of Manchester, UK

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

Abstract

Introduction

The move towards open science and open research practices has increased the demand for the publication of more artifacts of the research life cycle []. This is particularly apparent in domains that rely on computational experiments, for example, the publication of software, datasets and records of the dependencies that such experiments rely on [].

It is often argued that the publication of these assets and specifically software [https://doi.org/10.3233/DS-190026] and data should follow the The FAIR principles[https://doi.org/10.1038/sdata.2016.18], namely, that they are Findable, Accessible, Interoperable and Reusable. These principles are agnostic to the *implementation* strategy needed to meet them. Hence, there has been an increasing amount of work in the development of systems and specifications that aim to fulfill these goals []. Data publication with rich metadata (Zenodo) [], domain specific data deposit (PDB) [], following practices for reproducible research software [] (e.g. use of containers) are important examples.

These strategies are focused primarily on one *type* of artifact. To address this, [RO et al] introduced the notion of a *research object* - (def of RO). A research object combines the ability to document multiple of types of artifacts together, for example, a CSV files, code, examples, and figures. This provides a compelling vision as an approach for implementing FAIR. However, existing research object implementations require a large technology stack, were tailored to a particular platform and were also not easily usable for end-users.

To address this gap, a new community came together to develop **RO-Crate** - a *lightweight approach to packaging and aggregating research artifacts with their metadata*. The aim of this paper is to introduce RO-Crate and assess it as a strategy for making multiple types of research artifacts FAIR. Specifically, the contributions of this paper are as follows:

1. an introduction to RO-Crate, its purpose and context;
2. a guide to the RO-Crate community and tooling;
3. and exemplar usage of RO-Crate for different artifacts and its use as a connective tissue for such artifacts.

The rest of this paper is organized as follows. We first introduce RO-Crate, the assumptions underlying it and define it technically. We then proceed to introduce the community and tooling. We then analyze RO-Crate with respect to usage in a diverse set of domains. Finally, we present related work and conclude.

RO-Crate

As previously stated, RO-Crate provides a lightweight approach to packaging research artifacts with their metadata. What does that mean? Imagine a research paper reporting on the sequence analysis of proteins in an experiment on mice. The sequence analysis experiment code, associated sequence files and reports summarizing statistical measures would all be put in a directory. (Note, this is a brief description of actual practice. See: https://nf-co.re/chipseq/1.2.1/output https://github.com/stain/bco-ro-example-chipseq

The question then arises as to how the directory with all this material should be packaged in a manner that is accessible and usable by others. By usable we mean not just readable by humans but accessible programmatically. Fundamentally, how can one easily get this data and work with it. A defacto approach to sharing such compilations is through the use of a compressed archived (e.g. a zip file). While this solves the problem of "packaging", it does imply that the downstream user can *easily* access the data in a programmatic fashion. This leads to the need for explicit metadata about the contents of this package.

Examples of metadata description abound within the literature both in research data management (?cite) but also within library and information systems (?cite}. However, many of these approaches (discussed in the related work section) require knowledge of metadata schemas, particular annotation systems, the use of obscure or complex software stacks. Indeed, particularly, within research, these requirements have led to the under-adoption and frankly frustration with current tooling and specifications (https://cameronneylon.net/blog/as-a-researcher-im-a-bit-bloody-fed-up-with-data-management/).

RO-Crate seeks to address this complexity by:

1. being easy to understand and simple conceptually;
2. providing strong opinionated guide to current best practices;
3. adopting software stacks that are widely used on the Web.

These 3 desiderata are what is meant by "lightweight". We now show how the RO-Crate specification and ecosystem achieves these desiderata.

Conceptual Definition

A key premise of RO-Crate is the existence of a wide variety of resources on the Web that can help describe research. As such RO-Crate relies on concepts from the Web and in particular linked data.

Linked Data as a core principle

Linked Data [cite] is a core principle of RO-Crate, and IRIs[1] are therefore used to identify the RO-Crate, its constituent parts and metadata descriptions, as well as to the properties and classes used in the metadata.

Using Linked Data, where consumers can follow the links for more (ideally both human- or machine-readable) information, the RO-Crate can then be sufficiently *self-described* and related using global identifiers, without needing to recursively fully describe every referenced entity.

The base of Linked Data and shared vocabularies also means multiple RO-Crates and other Linked Data resources can be indexed, combined, queried, integrated or transformed using existing Semantic Web technologies like SPARQL and knowledge graph triple stores.

RO-Crate is a self-described container

An RO-Crate is defined as a self-described ***Root Data Entity***, which describes and contains *data entities*, which are further described using *contextual entities*. A **data entity** is primarily a *file* (bytes on disk somewhere) or a *directory* (dataset of named files); while a **contextual entity** exists outside the information system (e.g. a Person) and its bytes are primarily metadata.

The Root Data Entity_ _is a directory, the *RO-Crate Root*, identified by the presence of the *RO-Crate Metadata File* `ro-crate-metadata.json`. This is a JSON-LD file that describes the RO-Crate, its content and related metadata using Linked Data.

The minimal requirements for the root data entity metadata is *name*, *description* and *datePublished*, as well as a contextual entity identifying its *license*; but additional metadata is frequently added depending on the purpose of the particular RO-Crate.

Because the RO-Crate is just a set of related resources and datasets, it can be stored, transferred or published in multiple ways, such as BagIt [RFC8493], Oxford Common File Layout [OCFL], downloadable ZIP archives in Zenodo or dedicated online repositories, as well as published directly on the web, e.g. using GitHub Pages. Combined with Linked Data identifiers this caters for a diverse set of storage and access requirements across different scientific domains, e.g. metagenomics workflows producing ~100 GB of genome data, or cultural heritage records with access restrictions for personally identifiable data.

Data Entities are described using Contextual Entities

RO-Crate distinguishes between data and contextual entities in a similar way to HTTP terminology *information* (data) and *non-information* (contextual) resource [HttpRange-14]. While the data entities in the most common scenario are files and directories located by relative IRI references within the RO-Crate Root, they can also be web resources identified with absolute http/https IRIs.

As both types of entities are identified by IRIs, their distinction is allowed to be blurry; data entities can be located anywhere and be complex, and the contextual entities can have a Web presence beyond their description inside the RO-Crate. For instance <https://orcid.org/0000-0002-1825-0097> is primarily an identifier for a person, but is secondarily also a web page that describes that person and their academic work.

Any particular IRI might appear as a contextual entity in one RO-Crate, and a data entity in another; their distinction is that data entities can be considered to be *contained* or captured by the RO-Crate, while the contextual entities are mainly *explaining* the RO-Crate and its entities. It follows that an RO-Crate should have one or more data entities, although this is not a formal requirement.

Best Practice Guide rather than strict specification

RO-Crate builds on Linked Data standards and community-evolved vocabularies like JSON-LD and schema.org, but rather than enforce additional constraints and limits using perhaps intimidating technologies like *RDF shapes* (SHACL, ShEx), RO-Crate aims instead to build a set of best practice guides on how to apply the existing standards in a common way to describe research outputs and their provenance.

As such the RO-Crate specification [RO-Crate 1.1] can be seen as an opinionated and example-driven guide to writing schema.org metadata as JSON-LD, and leaves it open for implementers to add additional metadata using other schema.org types and properties, or even using other Linked Data vocabularies/ontologies or their own ad-hoc terms.

This does not mean that RO-Crate has not got constraints, crucially the specification is quite strict on the style of the flattened JSON-LD to facilitate lightweight developer consumption and generation of the RO-Crate Metadata file without the need for RDF libraries. In addition, work has now begun to formalize different *profiles* of RO-Crates, e.g. that submission to a workflow repository would expect an RO-Crate with at least one workflow with a declared license and workflow language. These optional

profiles are planned to build on a combination of JSON Schemas and RDF shapes, but again these will primarily provide developers with guidance and suggestions rather than strict validation.

Checking Simplicity

As previously discussed, one aim of RO-Crate is to be conceptually simple. This simplicity has been repeatable checked and confirmed through a community process. See for example discussion at (GitHub issue #71 on ad-hoc vocabularies).

To further verify this idea, we have here formalized the RO-Crate definition (see Appendix X). An important result of this exercise is that the underlying data structure of RO-Crate is a depth limited tree of 1 level. The formalization also emphasizes the *boundness* of the structure, namely, that elements are specifically identified as being either semantically *contained* by the RO-Crate (*hasPart*), or are mainly referenced (*mentions*) and typed as _external _to the Research Object (Contextual Entities).

Technical implementation of the model

The above conceptual model has been realized using JSON-LD and schema.org in a prescriptive form as discussed in the best practice approach. This technical approach again caters for simplicity.

JSON-LD [] provides a way to express Linked Data as a JSON structure, where a *context* provides mapping to RDF properties and classes. While JSON-LD can't map arbitrary JSON structure to RDF, we found it does provide a good starting point for making a JSON-based language that at the same also is interpretable as Linked Data.

In the design of RO-Crate we did however find that JSON-LD alone, like many RDF technologies, have too many degrees of freedom and hidden complexities for software developers to reliably produce and consume without specialized expertise or software libraries. A large part of the RO-Crate specification is therefore dedicated to describing JSON structures.

RO-Crate JSON-LD

RO-Crate mandates the use of flattened, compacted JSON-LD [https://www.researchobject.org/ro-crate/1.1/appendix/jsonld.html] where a single [**???**] array contains all the data and contextual entities in a flat list:

```
{ "@context": "https://w3id.org/ro/crate/1.1/context",
  "@graph": [
    { "@id": "ro-crate-metadata.json",
      "@type": "CreativeWork",
      "about": {"@id": "./"},
    },
    { "@id": "./",
      "@type": "Dataset",
      "hasPart": [
        {"@id": "data1.txt"},
        {"@id": "data2.txt"}
      ]
    },
    { "@id": "data1.txt",
      "@type": "File",
      "author": {"@id": "#alice"},
    },
    { "@id": "data2.txt",
      "@type": "File",
      "author": {"@id": "#alice"},
    },
    { "@id": "#alice",
      "@type": "Person",
      "name": "Alice",
    },
    { "@id": "http://sws.geonames.org/8152662/",
      "@type": "Place",
      "name": "Catalina Park"
    }
] }
```

Figure X: *Simplified RO-Crate JSON-LD showing the flattened compacted [**???**] array*

It can be argued that this is a more graph-like approach than the tree structure JSON would otherwise invite to, and which is normally emphasized as a feature of JSON-LD in order to "hide" its RDF nature.

We found however that the use of trees, say a *Person* entity appearing as author of a *File* which a *Dataset* nests under *hasPart*, counterintuitively lead to the need to consider the JSON-LD as an RDF Graph, as an identified Person entity then can appear at multiple and repeated points of the tree (e.g. author of multiple files), necessitating node merging or duplication.

In comparison, a single flat [**???**] array approach means that applications can process and edit each entity as pure JSON by a simple lookup based on `@id`. At the same time, lifting all entities to the same level emphasizes Research Object goals that describing the context and provenance is just as important as describing the data.

While RO-Crate mainly use schema.org, we found that JSON-based RO-Crate applications that are largely unaware of JSON-LD still may want to process the [**???**] to find Linked Data definitions of unknown properties and types. Thus RO-Crate provide its own, versioned JSON-LD context which has a similar flat list with the mapping from JSON-LD keys to their URI equivalents, e.g. author maps to http://schema.org/author. Not reusing the official schema.org context means RO-Crate is also able to map in additional vocabularies where needed, namely the *Portland Common Data Model* (PCDM) [] for repositories and Bioschemas [] for describing computational workflows.

Similarly, rather than relying on implications from "[**???**]: [**???**]" annotations in the context, RO-Crate JSON-LD distinguishes explicitly between references to other entities `{"id": "#alice"}` and string values `"Alice"` - meaning RO-Crate applications can find the corresponding entity without parsing the [**???**].

(...)

RO-Crate Community

This RO-Crate conceptual model, implementation and best practice guides are situated in a growing RO-Crate community. This community is a key aspect of the effectivenessaffectiveness of RO-Crate for making research artifacts FAIR. Fundamentally, the community provides the overall context of the implementation and model and ensures in the end interoperability.

The RO-Crate community consists of a:

1. A diverse set of people representing a variety of stakeholders
2. A set of collective norms
3. A open platform that facilitates communication (GitHub, Google Docs, monthly telcons)

People

The initial concept of RO-Crate was formed at the first Workshop on Research Objects (RO2018) at the IEEE eScience conference, which followed up considerations made at a RDA meeting on Research Data Packaging that found similar goals across multiple data packaging efforts [https://doi.org/10.5281/zenodo.3250687] of simplicity, structured metadata and the use of JSON-LD.

Discussions at RO2018 identified that while the original Wf4Ever Research Object ontologies [] in principle were sufficient for packaging research artifact with rich descriptions, in practice, due to their reliance on Semantic Web technologies and other ontologies, they were considered inaccessible for regular programmers (e.g. Web developers) and in danger of being incomprehensible for domain scientists.

DataCrate [https://doi.org/10.5281/zenodo.1445817] was presented at RO2018 as a promising lightweight alternative approach, and an agreement was made by a group of volunteers to attempt building "RO Lite" as a combination of DataCrate's implementation and Research Object's principles.

This set of volunteers were primarily digital library and semantic web experts. This group has subsequently grown to include domain scientists, developers, publishers, ++. This multiple perspective view has meant that the specification has been able to be used in a variety of domains (e.g...).

Engagement with other projects EOSC-Life, Bioschemas, etc.

A key set of stakeholders has been developers. The community has made a point of attracting developers that are able to implement the specifications but importantly takes “developer user experience” in mind. Namely, that the specifications are straightforward to implement and thus does not require expertise in technologies that are not widely deployed to implement.

This notion of catering to “developer user experience” is an example of the set of norms that have developed and now define the community.

Norms

We use the notion of norm here instead of principle, these are conventions or notions that are prevalent within the community but not formalized. Here, we distill what we as authors believe are the critical set of norms that have facilitated the development of RO-Crate and contribute to the ability for RO-Crate research packages to be FAIR. This is not to say that there are no other norms within the community or that every one in the community holds these uniformly. Instead, what we emphasize is that these norms are helpful and also shaped by community practice.

1. Simplicity
2. Developer friendliness
3. Focus on examples and best practice over rigorous specification
4. Reuse “just enough” Web standards

A core norm of RO-Crate is that of **simplicity**, which sets the scene for how we guide developers to structure metadata with RO-Crate, in that we focus mainly on documenting simple approaches to the most common use cases, for instance authors having an affiliation. This norm of simplicity also influences our take on **developer friendliness**, for instance in that we are using the Web-native JSON format and allowing only a few of JSON-LD’s flexible Linked Data features, and that the RO-Crate documentation is largely built up by **examples** showcasing **best practice**, rather than a rigorous specification that could be harder. In a sense we are allowed this by building on existing **Web standards** that themselves are defined rigorously, which we utilize ***“just enough”*** in order to benefit from the advantages of Linked Data (e.g. extensions by namespaced vocabularies) without imposing too many developer choices or uncertainties (e.g. having to choose between the many RDF syntaxes).

RO-Crate is not developed in a vacuum, and while the above norms alone could easily lead to the creation of “yet another” JSON format, we are also keeping the goal of **FAIR interoperability** of the captured metadata, and therefore follow closely FAIR best practices and current developments such as data citations, permanent identifiers, open repositories and recommendations for sharing of research outputs and software.

Open Platforms

The critical infrastructure that enables the community around the RO-Crate is the use of open development platforms. This might not seem novel but it underpins the importance of open community access to supporting FAIR. Specifically, it is difficult to build and consume FAIR research artifacts without being able to access the specifications, understand how they are developed, potential issues in implementations, and being able to discuss usage to evolve best practices.

It was seen as more important to document real-life examples and best practice guides than a rigorous specification. At the same time we agreed to be opinionated on the syntactic form to reduce the jungle of implementation choices; we wanted to keep the important aspects of Linked Data to adhere to the FAIR principles and retain the option of combining and extending the structured metadata using existing Semantic Web stack, not just build “yet another” standalone JSON format.

Further work during 2019 started adapting the DataCrate documentation through a more collaborative and exploratory *RO-Lite* phase, initially using Google Docs for review and discussion, moving to GitHub as a collaboration space for developing what is now the RO-Crate specification, maintained as Markdown in GitHub Pages and published with Zenodo.

In addition to the typical Open Source-style development with GitHub issues and pull requests, the RO-Crate Community now has two regular monthly calls, a Slack channel and mailing list for coordinating the project, and many of its participants collaborate on RO-Crate at multiple conferences and coding events such as the ELIXIR BioHackathon. The community is jointly developing the RO-Crate specification and Open Source tools, as well as providing support and considering new use cases. The RO-Crate Community is open for anyone to join, to equally participate under a code of conduct, and currently has more than 40 members.

RO-Crate Tooling

The work of the community has resulted in a number of tools for creating and using RO-Crates. Table X, shows the current set of implementations. Reviewing this list, one can see that there are tools that support commonly used programming languages including Python, Javascript, and Ruby. Additionally, these tools can integrate with commonly used_ _research environments in particular the command line (CalcyteJS). Furthermore, there are tools that cater to the end user (Describo, Workflow Hub). For example, **Describo** was developed to help researchers of the Australian Criminal Characters project (https://criminalcharacters.com/) which is annotating historical prisoner records to gain greater insight into the history of Australia.

While the development of these ~12 tools is promising, analyzing the status we see that the majority are in the beta stage. That being the stage the RO-Crate specification reached 1.0 status one year ago (Nov 2019). Now that there is a fixed point of reference, and RO-Crate 1.1 (Oct 2020) has stabilized based on feedback from application development, we expect to see an increase in the maturity of these tools.

Given the stage of the specification, the targets of these tools have been primarily developers essentially providing the core libraries for working with RO-Crates. Another target has been research data managers who need to manage and curate large amounts of data.

We argue that the adoption of simple web technologies in the RO-Crate specification has lent to the development of this wide variety of tools.

Tool Name	Targets	Language / Platform	Status	Brief Description
Describo	End users, Research Data Managers	NodeJS (Desktop)	Release Candidate	Interactive desktop application to create, update and export RO-Crates for different profiles
Describo Online	Platform developers	NodeJS (Web)	Alpha	
ro-crate-excel	Data managers	JavaScript	Beta	Command-line tool to help create RO-Crates and HTML-readable rendering
ro-crate-html-js	Developers	JavaScript	Beta	HTML rendering of RO-Crate
ro-crate-js	Research Data managers	JavaScript	Alpha	Command line tool, render HTML from RO-Crate

Tool Name	Targets	Language / Platform	Status	Brief Description
ro-crate-ruby	Developers	Ruby	Beta	
ro-crate-py	Developers	Python	Alpha	
WorkflowHub	Workflow users	Ruby	Beta	Imports and exports Workflow RO-Crates
OCFL-indexer	Repository managers	NodeJS		OCFL-based RO-Crate validation and indexing
ONI express	Repository managers	Platform		platform for publishing data and documents stored in an OCFL repository via a web interface
ocfl-tools	Developers	?		Is this just used by OCFL-indexer?
ocfl-viewer	Developers	Lua?		
RO Composer	Repository developers	Java	Alpha	REST API for gradually building ROs for given profile.
galaxy2cwl	Workflow developers	Python	Alpha	Wraps Galaxy workflow as Workflow RO-Crate

Using RO-Crate

RO-Crate is fundamentally an infrastructure to help make FAIR research artifacts. In other words, the key question can RO-Crate be used to share and (re)use research artifacts. We look at three research domains where it is being applied here: Bioinformatics, Regulatory Science and Cultural Heritages

Bioinformatics workflows

WorkflowHub.eu is a European cross-domain registry of computational workflows, supported by European Open Science Cloud projects like EOSC-Life and research infrastructures, including the pan-European bioinformatics network ELIXIR. As part of promoting workflows as reusable tools, the WorkflowHub includes documentation and high-level rendering of the workflow structure independent of its native workflow definition format - the rationale is that a domain scientist looking for a particular computational analysis can browse all relevant workflows before narrowing down their workflow engine requirements. As such the WorkflowHub is intended largely as a registry over workflows already deposited in repositories specific to particular workflow languages and domains, such as UseGalaxy.eu and Nextflow nf-core.

Being cross-domain also means the WorkflowHub has to cater for many different workflow systems. Many of these, for instance Nextflow and Snakemake, by their script-like nature, reference multiple neighbouring files typically maintained in a GitHub repository. The WorkflowHub has therefore adapted RO-Crate as the packaging mechanism typing and annotating the constituent files of a workflow, crucially marking up the workflow language, as many workflow engines use common file extensions like *.xml and* .json. RO-Crates can thus be used for interoperable deposition of workflows to the WorkflowHub, but is also used by the archive downloading workflows, embedding metadata registered with the Workflow Hub entry and translated workflow definitions such as abstract CWL and diagrams.

In WorkflowHub, RO-Crate therefore can be seen as taking on the role of an interoperability layer between registries, repositories and users. The iterative development between Workflow Hub developers and RO-Crate community heavily informed the creation of the Bioschemas profile for

Computational Workflow, which again informed the RO-Crate 1.1 specification on workflows and led to the RO-Crate Python library and WorkflowHub's Workflow RO-Crate profile which in a similar fashion to RO-Crate itself recommends which workflow resources and descriptions are required. This co-development across project boundaries exemplifies the drive for simplicity at the same time as establishing best practices.

While RO-Crates in Workflow Hub so far has focused on workflows that are ready to be run, they are now moving into the development of Workflow Run RO-Crate, for the purposes of benchmarking, testing and executing workflows.

(... more on provenance, abstract CWL, testing, execution, Galaxy converter)

Regulatory Sciences

BioCompute Objects [Alterovitz 2018] is a community-led effort to standardize submissions of computational workflows to biomedical regulatators, e.g. a genomics sequencing pipeline being part of personalized cancer treatment could be submitted to the US Food and Drugs Administration (FDA) for approval. BCOs are formalized in the standard [IEEE 2791-2020] as a combination of JSON Schemas that give the structure of JSON metadata files which describe exemplar workflow runs in detail, covering aspects such as the usability and error domain of the workflow, its runtime requirements, reference datasets used and representative output data produced.

While a BCO can give a structured view over a particular workflow, informing regulators about its workings independent of the underlying workflow definition language, it has only limited support for additional metadata. For instance, while the BCO itself can indicate authors and contributors, and in particular regulators and their review decisions, it cannot describe the provenance of individual data files or workflow definitions beyond assigning them global identifiers in the form of IRIs.

As a custom JSON format BCOs cannot be extended with Linked Data concepts, except by adding an additional top-level JSON object formalized in another JSON Schema. A BCO and workflow submitted by upload to a regulator will also frequently consist of multiple cross-related files. Crucially there is no way to identify a given *.json is a BCO file, except by reading its content checking for its indicated `spec_version`; yet many tools and workflow systems also use JSON files in their internal formats.

As such we can consider how a BCO and its referenced artifacts can be packaged and transferred following FAIR principles. BCO RO-Crate, part of the BioCompute Object user guides, defines a set of best practices for wrapping a BCO with a workflow and its exemplar outputs in a RO-Crate, which then provides typing and additional provenance metadata of the individual files, workflow definition, referenced data and the BCO metadata itself. While the BCO remains rigid, the RO-Crate is more open-ended and can therefore also describe other files in the submission not directly relatable from the BCO, such as further workflow documentation.

While there is some overlap in that RO-Crate can also describe a computational workflow, as detailed in previous section, a *separation of concerns* emerges, where the BCO is responsible for describing the inside of a workflow and its run at an abstraction level suitable for a domain scientist, the RO-Crate describes the surroundings of the workflow, classifying and relating its resources and providing provenance of their existence beyond the BCO.

A similar separation of concerns we can find within the RO-Crate itself, where the transport-level metadata such as checksum of files are delegated to BagIt manifests, a standard focusing on preservation challenges by digital libraries [RFC8493]. As such RO-Crates are not required to list all the files in its folder hierarchy, only those that are deemed to be needing a description.

Going deeper, a BCO alone is insufficient for reliable re-execution of a workflow, which would need a compatible workflow engine depending on the workflow definition language, so IEEE 2791 recommends using Common Workflow Language [10.6084/m9.figshare.3115156.v2] for interoperable pipeline execution. CWL itself relies on tool packaging in software containers using Docker or Conda. As such we can consider BCO-RO-Crate as a stack consisting of transport-level manifest of files (BagIt), provenance, typing and context of those files (RO-Crate), workflow overview and purpose (BCO), interoperable workflow definition (CWL) and tool distribution (Docker).

Separation of Concerns in BCO RO-Crate

Figure X: BioCompute Object (IEEE2791) is a JSON file that structurally explains the purpose and implementation of a computational workflow, for instance implemented in Nextflow that installs the workflow's software tools dependencies of as Docker containers or BioConda packages. An example execution of the workflow exemplifies its kind of result outputs, which may be external using GitHub LFS to support larger data. The RO-Crate gathers all these local and external resources, relating them and giving individual descriptions, for instance permanent identifiers DOIs for reused datasets accessed from Zenodo, but also adding external identifiers to attribute authors using ORCID or to identify which licenses apply to individual resources. The RO-Crate and its local files are captured in a BagIt which checksums ensures completeness, combined with Big Data Bag features to "complete" the bag with large external files such as the workflow outputs.

Digital Humanities: Cultural Heritage

PARADISEC (the Pacific And Regional Archive for Digital Sources in Endangered Cultures) maintains a repository of more than 500,000 files documenting endangered languages across more than 16.000 items, collected over many years by researchers interviewing and recording native speakers across the region. As a proposed update of the 18 year old infrastructure, the Modern PARADISEC demonstrator has been developed to help also digitally preserve these artifacts using the Oxford Common File Layout (OCFL) for file consistency and RO-Crate for structuring and capturing the metadata of each item. The existing PARADISEC data collection has been ported and captured as RO-Crates. A Web portal then exposes the repository and its entries by indexing the RO-Crate metadata files using Elasticsearch as a "NoSQL" object database, presenting a domain-specific view of the items - the RO-Crate is "hidden" and does not change the user interface.

This use case takes advantage of several RO-Crate features and principles. Firstly the transcribed metadata is now independent of the PARADISEC platform and can be archived, preserved and processed in its own right, using schema.org vocabularies augmented with PARADISEC-specific terms. The lightweight infrastructure with RO-Crate as the holder of itemized metadata in regular files (protected by OCFL) also gives flexibility for future developments and maintenance, like adding of potential Linked Data software such as a graph database queried using SPARQL triple patterns across RO-Crates, or a "last resort" fallback the generic RO-Crate HTML preview (ro-crate-html-js) which can be hosted as static files by any web server.

Related Work

With the greater digitization of research processes, there has been a significant call for the wider use of interoperable sharing of data and its associated metadata. For a comprehensive overview of this literature and recommendations in particular for data, we refer to [https://doi.org/10.1016/j.patter.2020.100136]. It highlights the wide variety of metadata and documentation that the literature prescribes for enabling the reuse of data.

Here, instead of surveying the large literature on sharing digital scholarly artifacts, we rather focus on approaches to bundling such artifacts along with their metadata. This notion has a long history [https://doi.org/10.1190/1.1822162] but recent approaches have followed three strands: 1) publishing to centralized repositories; 2) packaging approaches similar to RO-Crate; and 3) and bundling the computational workflow around a scientific experiment.

Repositories

- DataCite Metadata
- Open Science Framework

Bunding and Packaging Digital Research Artifacts

The challenge of describing computational workflows was one of the main motivations for the proposal of *Research Objects*https://doi.org/10.1016/j.future.2011.08.004 as first-class citizens for sharing and publishing, by bundling datasets, workflows, scripts, results along with traditional dissemination materials like journal articles and presentations, forming a single package. Crucially, these resources are not just gathered, but also individually typed, described and related to each-other using semantic vocabularies. As pointed out inhttps://doi.org/10.1016/j.future.2011.08.004 an open-ended *Linked Data* approach is not sufficient for scholarly communication, as a common data model is also needed in addition to common practices for managing and annotating lifecycle, ownership, versioning and attributions.

Considering the FAIR principles we can say with hindsight (the FAIR paper was published 7 years later) that the initial Research Objects approaches were strongly targeting *Interoperability*, with a particular focus on reproducibility with computational workflows and reuse of existing RDF vocabularies.

The first implementation of Research Objects in 2009 for sharing workflows in myExperimenthttps://doi.org/10.1093/nar/gkq429 was based on RDF ontologieshttp://eprints.soton.ac.uk/id/eprint/267787, building on Dublin Core, FOAF, SIOC, Creative Commons, OAI-ORE and (later) DBPedia to form myExperiment ontologies for describing social networking, attribution and credit, annotations, aggregation packs, experiments, view statistics, contributions, and workflow components.http://web.archive.org/web/20091115080336/http://rdf.myexperiment.org/ontologies

Programmatic access to Research Objects was facilitated with an RDF endpoint that exposed individual myExperiment resources, also queriable from a SPARQL endpoint, both using the myExperiment vocabularies and RDF formats RDF/XML and Turtle.

```
*   RO-Bag-It
*   BDBag
*   Ontologies ( OAI-ORE, OAC/AO/OA)
```

FAIR Digital Objects

FAIR Digital Objects (FDO) [https://doi.org/10.3390/publications8020021] have been proposed as a conceptual framework for making digital resources available in a Digital Objects (DO) architecture that encourage active use of the objects and their metadata. In particular, an FDO has five parts: (i) The FDO *content*, bit sequences stored in an accessible repository, (ii) a *Permanent Identifier* (PID) such as DOIs that identify the FDO and can resolve these parts, (iii) Associated rich *metadata*, as separate

FDOs, (iv) Type definitions, also separate FDOs, (v) Associated *operations* for the given types. A Digital Object typed as a Collection aggregates other DOs by reference.

As an "abstract protocol", Digital Objects could be implemented in multiple ways. Implementations suggested include FAIR Digital Object Framework based on HTTP and Linked Data, and while there is agreement on using permanent identifiers based on DOI, agreement on how to represent common metadata, core types and collections as FDOs have not yet been reached. We argue that RO-Crate can play an important role for FDOs:

1. By providing a predictable and extensible serialization of structured metadata
2. By formalizing how to aggregate digital objects as a collection (and adding their context)
3. The RO-Crate Metadata File forms a natural Metadata FDO
4. Based on Linked Data and schema.org vocabulary means PIDs already exist for common types and properties.

At the same time it is clear that the goal of FDO is broader than RO-Crate; namely FDOs are active objects with a distributed operations, and add further constraints such as permanent identifiers for every element. While these measures can be said to improve FAIR features of digital objects and are also useful for RO-Crate, they also severely restrict the possible FDO infrastructure that need to be implemented and maintained in order for FDOs to remain available. RO-Crate on the other hand is more flexible, it can minimally be used within any file system structure, or ideally exposed through a range of Web-based scenarios, but a *FAIR profile of RO-Crate* (e.g. enforcing PID usage) will also fit well within and help to implement a FAIR Digital Object ecosystem.

Packaging Workflows

The use of computational workflows has gained prominence, in particular life sciences, typically combining a chain of open source tools in an analytical pipeline. While the workflows initially may have been used to improve scalability, it can be argued they both assist in making computated data results FAIR, but at the same time raising additional FAIR challenges when considering the workflows as important research artifacts themselves in order to capture and explain the computational method behind an analysis[https://doi.org/10.1162/dint_a_00033].

Even when researchers follow current best practice for workflow reproducibility,https://doi.org/10.1016/j.cels.2018.03.014 https://doi.org/10.1016/j.future.2017.01.012 the communication of that outcome through traditional academic publishing routes with a textual representation adds barriers that hinder reproducibility and FAIR use of the knowledge previously captured in the workflow. Even as researchers the ambition of FAIR reproducible research, it has not yet become common practice.

As a real-life example let's look at a metagenomics article in Naturehttps://doi.org/10.1038/s41586-019-0965-1, where the authors have gone to extraordinary effort to document the individual tools that have been reused, including their citations, versions, settings, parameters and combinations. The *Methods* section is 2 pages in tight double-columns with 24 additional references, supported by data availability on FTP server (60 GB)http://ftp.ebi.ac.uk/pub/databases/metagenomics/umgs_analyses/ and the open source code in GitHub repositoryhttps://github.com/Finn-Lab/MGS-gut includes the pipeline as shell scripts and associated analysis scripts in R and Python.

This attention to reporting detail for computational workflows is unfortunately not yet the norm, and although bioinformatics journals have strong *data availability* requirements they frequently do not require authors to include or cite *software, scripts and pipelines* used for analysing and producing resultshttps://twitter.com/soilandreyes/status/1250721245622079488 - rather authors are often

penalized for doing so [cite?] as it would work against artificial limits on number of pages and references.

However detailed, for a new researcher who wants to reuse a particular computational method they may first want to assess if the described tool and workflow is Re-runnable (executable at all), Repeatable (same results for original inputs on same platform), Reproducible (same results for original inputs with different platform or newer tools) and ultimately Reusable (similar results for different input data), Repurposable (reusing parts of the method for making a new method) or Replicable (rewriting workflow following the method description).[https://doi.org/10.3389/fninf.2017.00069] [Goble: What is Reproducibility? The R* brouhaha]

Following the textual description alone, researchers would be forced to jump straight to evaluate "Replicable" by rewriting the pipeline from scratch. This can be expensive and error-prone. They may would firstly need to install all the software dependencies and reference datasets. This can be a daunting task in itself, which may have to be repeated multiple times as workflows typically are developed at small scale on their desktop computer, scaled up to a local cluster, and potentially productionized using cloud instances, each of which will have different requirements for software installations.

In recent years the situation has been greatly improved by software packaging and container technologies like Docker and Conda, which have seen increased adaptation in life scienceshttps://doi.org/10.1007/s41019-017-0050-4 with supporting collaborative efforts like BioCondahttps://doi.org/10.1038/s41592-018-0046-7, BioContainers [??] and by Linux distributions themselves (Debian Medhttps://doi.org/10.1186/1471-2105-11-S12-S5) to make more than 7000 software packages available in BioConda alonehttps://anaconda.org/bioconda/ and 9000 containers in BioContainershttps://biocontainers.pro/#/registry. Docker and Conda has gained integration in workflow systems like Snakemake, Galaxy, Nextflow, meaning a downloaded workflow definition can now be executed on a "blank" machine (except for the workflow engine) with the underlying analytical tools installed on demand.

Conclusion

Formal definition of RO-Crate

Formalizing RO-Crate in First Order Logic

Below is a brief formalization of RO-Crate as a set of relations in First Order Logic, followed by a mapping to RDF using schema.org and forward-chaining production rules for making JSON-LD.

�� *ro-crate* = { Property(p), Class(c), Literal(x), Describes(R, s) } ⁇ = ⁇ ⁇ ≡ { IRIs as defined in https://tools.ietf.org/html/rfc3987 } ℝ ≡ { real or integer numbers } ⁇ ≡ { literal strings }

Minimal RO-Crate

RO-Crate(R) ⊨ Root(R) ∧ Describes((R, R) RO-Crate(R) ⊨ hasPart(R, d) ∧ Describes((R, d) ∧ DataEntity(d) RO-Crate(R) ⊨ Describes((R, c) ∧ ContextualEntity(c)

Root(r) → Dataset(r) ∧ published(r, Date) published(e, date) → Literal(date) DataEntity(e) ≡ File(e) ⊕ Dataset(e) Entity(e) ≡ DataEntity(e) ∨ ContextualEntity(e)

Describes(R, s) ⊨ Relation(s, p, e) ⊕ Value(s, p, l) ∀x . Value(o, p, x) → Literal(x) Literal(x) ≡ x ∈ ℝ ⊕ x ∈ ��

Relation(s, p, o) ⊨ Entity(s) ∧ Property(p) ∧ Entity(o) Entity(e) → Metadata(e) Metadata(e) → Class(t) ∧ Describes(R, e)

Mapping to RDF with schema.org

Dataset(d) → type(d, http://schema.org/Dataset) File(f) → type(f, http://schema.org/MediaObject) Property(p) → type(p, http://www.w3.org/2000/01/rdf-schema#Property) Class(c) → type(c, http://www.w3.org/2000/01/rdf-schema#Class)

hasPart(e, t) → Relation(e, http://schema.org/hasPart, t) type(e, t) → Relation(e, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, t) published(e, date) → Value(e, http://schema.org/datePublished, date)

RO-Crate 1.0 Metadata File Descriptor

about(s,o) → Relation(, http://schema.org/about, o) conformsTo(s,o) → Relation(s, http://purl.org/dc/terms/conformsTo, R) CreativeWork(e) → ContextualEntity(m) ∧ type(m, http://schema.org/CreativeWork) MetadataFileDescriptor(m) → (CreativeWork(m) ∧ about(m,R) ∧ RO-Crate(R) ∧ conformsTo(m, https://w3id.org/ro/crate/1.1))

Forward-chained Production Rules for JSON-LD

Describes(R, S) ∧ Relation(S, P, O) → Describes(R, O) i ∈ ������ → i r ∈ ℝ → r s ∈ �� → “s” Relation(s,p,o) → { “[**???**]”: s, p: { “[**???**]”: o } } Value(s,p,o) → { “[**???**]”: s, p: o } RO-Crate(r) → { “[**???**]”: [Describes((r, c)] } R ≡ <./> MetadataFileDescriptor(<ro-crate-metadata.json>) .

Appendix

Formalizing RO-Crate in First Order Logic

Below is an attempt to formalize the concept of RO-Crate as a set of relations using First Order Logic:

Language

```
��<sub>ro-crate</sub> = { Property(p), Class(c), Literal(x), ℝ, ��  }


��  =  ������  \
```

������ ≡ { IRIs as defined in <https://tools.ietf.org/html/rfc3987> }
ℝ ≡ { real or integer numbers }

�� ≡ { literal strings }

Minimal RO-Crate

```
RO-Crate(R) ⊨  Root(R) ∧ Mentions(R, R) ∧ \
hasPart(R, d) ∧ Mentions(R, d) ∧ DataEntity(d) ∧ \
Mentions(R, c) ∧ ContextualEntity(c)


∀r Root(r) →  Dataset(r) ∧ name(r, n)∧ description(r, d) \
                      ∧ published(r, date) ∧ license(e, l) \
```

∀e∀n name(e, n) → Literal(n)
∀e∀d description(e, d) → Literal(d)
∀e∀date datePublished(e, date) → Literal(date)
∀e∀l license(e, l) → ContextualEntity(l)

```
DataEntity(e) ≡  File(e) ⊕ Dataset(e) \
```

Entity(e) ≡ DataEntity(e) ∨ ContextualEntity(e)
∀e Entity(e) → Class(e)

```
Mentions(R, s) ⊨  Relation(s, p, e)  ⊕  Attribute(s, p, l) \
```

Relation(s, p, o) ⊨ Entity(s) ∧ Property(p) ∧ Entity(o)
Attribute(s, p, x) ⊨ Entity(s) ∧ Property(p) ∧ Literal(x)
Literal(x) ≡ x ∈ ℝ ⊕ x ∈ ��

The domain of discourse is the set of ������ identifiers (notation <http://example.com/>), with additional descriptions using numbers ℝ (notation 13.37) and literal strings �� (notation “Hello”).

From this formalized language $\text{}_{\text{ro-crate}}$ a RO-Crate can be interpreted in any representation that can gather these descriptions, their properties, classes, and literal attributes.

An RO-Crate(R) is defined as a self-described *Root Data Entity*, which describes and contains parts (*data entities*), which are further described in *contextual entities*. These terms align with their use in the RO-Crate 1.1 terminology.

The Root(r) is a type of Dataset(r), and must have the metadata to literal attributes to provide a name, description and datePublished, as well as a contextual entity identifying its license. These predicates correspond to the RO-Crate 1.1 requirements for the root data entity.

The concept of an Entity(e) is introduced as being either a DataEntity(e), a ContextualEntity(e), or both; and must be typed with at least one Class(e).

For simplicity in this formalization (and to assist production rules below) R is a constant representing a single RO-Crate, typically written to independent RO-Crate Metadata files. R is used by Mentions(R, e) to indicate that e is an Entity described by the RO-Crate and therefore its metadata (a set of Relation and Attribute predicates) form part of the RO-Crate serialization. Relation(s, p, o) and Attribute(s, p, x)

are defined as a *subject-predicate-object* triple pattern from an Entity(s) using a Property(p) to either another Entity(o) or a Literal(x) value.

Example of formalized RO-Crate

The below is an example RO-Crate represented using the above formalization, assuming a base URI of http://example.com/ro/123/:

```
RO-Crate(&lt;[http://example.com/ro/123/](http://example.com/ro/123/)>) \
```

name(<http://example.com/ro/123/,
“Data files associated with the manuscript:Effects of facilitated family case …”)
description(<http://example.com/ro/123/,
“Palliative care planning for nursing home residents with advanced dementia …”)
datePublished(<http://example.com/ro/123/>, “2017”)
license(<http://example.com/ro/123/>, <https://creativecommons.org/licenses/by-nc-sa/3.0/au/>

```
ContextualEntity(&lt;[https://creativecommons.org/licenses/by-nc-sa/3.0/au/]
(https://creativecommons.org/licenses/by-nc-sa/3.0/au/)>) \
```

name(<https://creativecommons.org/licenses/by-nc-sa/3.0/au/,
“Attribution-NonCommercial-ShareAlike 3.0 Australia (CC BY-NC-SA 3.0 AU)”)

hasPart(<http://example.com/ro/123/>, <http://example.com/ro/123/file.txt>)
File(<http://example.com/ro/123/survey.csv>)
name(<http://example.com/ro/123/survey.csv>, “Survey of care providers”)

```
hasPart(&lt;[http://example.com/ro/123/](http://example.com/ro/123/)>, &lt;
[http://www.example.om/ro/123/folder/](http://example.com/ro/123/file.txt)>)
\
```

Dataset(<http://example.com/ro/123/interviews/>)
name(<http://example.com/ro/123/interviews/>, “Audio recordings of care provider interviews”)

In reality many additional attributes from schema.org types like http://schema.org/Dataset and http://schema.org/CreativeWork would be used to further describe the RO-Crate and its entities, but as these are optional they do not form part of this formalization.

Mapping to RDF with schema.org

A formalized RO-Crate can be mapped to different serializations. Below follows a mapping to RDF using schema.org.

```
Dataset(d) →  type(d, &lt;[http://schema.org/Dataset]
(http://schema.org/Dataset)>) \
```

File(f) → type(f, <http://schema.org/MediaObject>)
Property(p) → type(p, <http://www.w3.org/2000/01/rdf-schema#Property>)
Class(c) → type(c, <http://www.w3.org/2000/01/rdf-schema#Class>)
CreativeWork(e) → ContextualEntity(e) ∧ type(e, <http://schema.org/CreativeWork>)

```
hasPart(e, t) →  Relation(e, &lt;[http://schema.org/hasPart]
(http://schema.org/hasPart)>, t) \
```

type(e, t) → Relation(e, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, t) ∧ Class(t)
name(e, n) → Attribute(e, <http://schema.org/name>, n)
description(e, d) → Attribute(e, <http://schema.org/description>, d)
datePublished(e, date) → Attribute(e, <http://schema.org/datePublished>, date)
license(e, l) → Relation(e, <http://schema.org/license>, l) ∧ CreativeWork(l)

Note that in the JSON-LD serialization of RO-Crate the expression of Class and Property is typically indirect, as the JSON-LD `@context` maps to schema.org IRIs, which when resolved as Linked Data embeds their formal definition as RDFa.

RO-Crate 1.1 Metadata File Descriptor

An important RO-Crate principle is that of being **self-describing**. Therefore the serialization of the RO-Crate into a file should also describe itself in a Metadata File Descriptor, indicating it is about (describing) the RO-Crate root data entity, and that it conformsTo a particular version of the RO-Crate specification:

```
about(s,o) →  Relation(s, &lt;[http://schema.org/about]
(http://schema.org/about)>, o) \
```

conformsTo(s,o) → Relation(s, <http://purl.org/dc/terms/conformsTo>, R)
MetadataFileDescriptor(m) → (CreativeWork(m) ∧ about(m,R) ∧ RO-Crate(R) ∧ conformsTo(m, <https://w3id.org/ro/crate/1.1>))

Note that although the metadata file necessarily is an *information resource* written to disk or served over the network (e.g. as JSON-LD), it is not considered to be a contained *part* of the RO-Crate in the form of a *data entity*, rather it is described only as a *contextual entity*.

While in the conceptual model the *RO-Crate Metadata File* can be seen as the top-level node that describes the *RO-Crate Root*, in the formal model (and the JSON-LD format) the metadata file descriptor is an additional contextual entity and not affecting the depth-limit of the RO-Crate.

Forward-chained Production Rules for JSON-LD

Combining the above predicates and schema.org mapping with rudimentary JSON templates, these forward-chaining production rules can output JSON-LD according to the RO-Crate 1.1 specification:

```
Mentions(R, s) ∧ Relation(s, p, o) →  Mentions(R, o) \
```

i ∈ ℕ → `"i"`
r ∈ ℝ → `r`

s ∈ `"s"`

```
∀s∀p∀o Relation(s,p,o) →  `{ "@id": `s`, \
                  `p`: { "@id": `o` } \
               }`


∀s∀p∀v Attribute(s,p,v) → ` { "@id": `s`, \
                  `p`: `v` }`


∀r∀c  RO-Crate(r) → ` { "@graph": [ `Mentions(r, c)*` ] } \
```

`R ⊨ <./>
MetadataFileDescriptor(<ro-crate-metadata.json>)

This exposes the first order logic domain of discourse of IRIs, with rational numbers and strings as their corresponding JSON-LD representation. These production rules first grow the graph of R by adding a transitive rule that anything described in R which is related to o means that o is also mentioned by the RO-Crate R. For simplicity this rule is one-way; in practice the JSON-LD graph can also contain free-standing contextual entities that have outgoing relations to data- and contextual entities.

Limitations: The full list of types, relations and attribute properties from the RO-Crate specification are not included. Examples shown include *datePublished*, *CreativeWork* and *name*. Contextual entities not related from the RO-Crate (e.g. using inverse relations to a data entity) would not be covered by the single direction Mentions(R, s) production rule; see issue #122. The datePublished_(_e, date) rule do not include syntax checks for the ISO 8601 datetime format. Compared with RO-Crate examples, this generated JSON-LD does not use a `@context` as the IRIs are produced unshortened; a post-step could be JSON-LD Flattening with a versioned RO-Crate context.

References

1. IRIs, are a generalization of URIs (which include well-known http/https URLs), permitting international Unicode characters without %-encoding [RFC3987], commonly used on the browser address bar and in HTML5.↩