A suite of APIs for the management of Research Objects

Raúl Palma[1], Piotr Hołubowicz[1*], Kevin Page[2], Stian Soiland-Reyes[3], Graham Klyne[2], and Cezary Mazurek[1]

[1] Poznan Supercomputing and Networking Center, Poznań, Poland,
rpalma@man.poznan.pl,piotrhol@google.com,mazurek@man.poznan.pl
[2] University of Oxford, Oxford, UK,
kevin.page@oerc.ox.ac.uk; gklyne@gmail.com
[3] University of Manchester, Manchester, UK
soiland-reyes@cs.manchester.ac.uk

Abstract. Research Objects (ROs) are semantic aggregations of related scientific resources, along with their annotations and research context. They provide the means to refer a bundle of research artefacts supporting an investigation, and the mechanisms to associate human and machine-readable metadata to these artefacts. The RO model, implemented as a suite of ontologies, provides the means for capturing and describing such objects, their provenance and lifecycle. Based on the RO model, we have specified and implemented a set of APIs for the management of ROs. These APIs have been developed following a resource-oriented approach in line with Linked Data and REST approaches. The two main APIs are the RO API, which defines the formats and links used to create and maintain ROs in a digital repository, and the RO evolution API, which defines the formats and links used to change the lifecycle stage of ROs and retrieve their evolution provenance. Services and applications can implement one or several of these APIs, like in the case of ROHub, a digital library specialised for ROs, which exposes the two main APIs among others. In this paper, we describe the design and specification of these APIs, how they tackle the lifecycle management of ROs, as well as their implementation and usage in ROHub along with the lessons learned.

1 Introduction

Research outcomes in our increasingly digital world are more than just traditional publications. They comprise all the artefacts supporting the associated investigation or study, including datasets, software code, scripts, computational methods like scientific workflows used in experimental science, and even presentation slides or logs. Thus, traditional publications are not enough to share and reuse research outcomes, especially in experimental science, and do not include all the artefacts necessary for reproducing them. Research Objects (ROs)[3] provide a container for all these related artefacts. They are aggregating objects that bundle together the resources that are essential to a computational scientific study or investigation, the people involved, and annotations needed for the understanding and interpretation of the scientific outcomes (e.g., provenance information, descriptions of the computational methods, dependency information and

[*] Present address: Google, Mountain View, CA, USA

settings about the experiment executions). The RO model[4], implemented as a suite of lightweight ontologies, provides a vocabulary and semantic relations for capturing and describing ROs, their provenance and lifecycle, facilitating the reusability, reproducibility and better understanding of scientific experiments.

Based on the RO model, we have specified a suite of APIs for the management of ROs. These APIs have been implemented in different services and applications, like the ROhub system, which exposes a subset of these APIs enabling developers to build applications that use or produce ROs. For instance, a publisher may encourage the publication of ROs through its website, and use the APIs to create and maintain them.

2 Research Object Model

The RO model[4] consists of a core ontology and several extensions. The core *ro* ontology provides the basic structure for the description of aggregated resources and annotations on those resources. The *roevo* ontology extends it to capture the RO evolution, including the different stages during their lifecycle, the corresponding versions of these objects and their aggregated resources along with their associated changes. The suite also includes two extensions for the description of workflow-centric ROs: the workflow description vocabulary (*wfdesc*) and the workflow execution provenance vocabulary (*wfprov*). The ontologies were built upon existing vocabularies: OAI ORE (Object Exchange and Reuse) [1] for specifying aggregation of resources, the Annotation Ontology (AO) [6] to support the annotations, and the PROV Ontology[2] to represent provenance information. In line with Linked Data principles, the model has been implemented using semantic web languages (RDF & OWL). Next we summarise the key terms of the core *ro* and *roevo* ontologies (depicted in Figure 1).

The *ro* core ontology defines the `ro:ResearchObject` (subclass of `ore:Aggregation`) and the `ore:AggregatedResource`(s) which may be the scientific artefacts aggregated by the RO (`ro:Resource`) or annotations about these artefacts (`ro:AggregatedAnnotation` subclass of `ao:Annotation`). `ro:Manifest` (subclass of `ore:ResourceMap`) represents the RDF resource used to describe the RO (e.g., its identity, aggregated resources, timestamp, etc.).

The *roevo* ontology defines three subclasses of `ro:ResearchObject`, representing the different states of a RO during its life time. A `roevo:LiveRO` is a RO instance used to capture research findings during the course of an investigation, and can be modified, archived or snapshotted. An `roevo:ArchivedRO` is a preserved RO instance which may be referenced with assurance that it will not subsequently be changed. A `roevo:SnapshotRO` represents a Live RO at a particular time, but is not necessarily subjected to the same level of preservation as an Archive RO. Both a snapshot and an archive are handled as versions of a RO (`roevo:VersionableResource`), with their own provenance. *roevo* can also represent changes (`roevo:Change`) between RO versions as `roevo:ChangeSpecification`, building upon the PROV vocabulary, with ROs as `prov:Entity`, and changes as `prov:Activity`.

3 The suite of APIs

APIs define the sets of resources and resource patterns provided by a service that a client can reliably expect to interact with, and the set of representations these resources

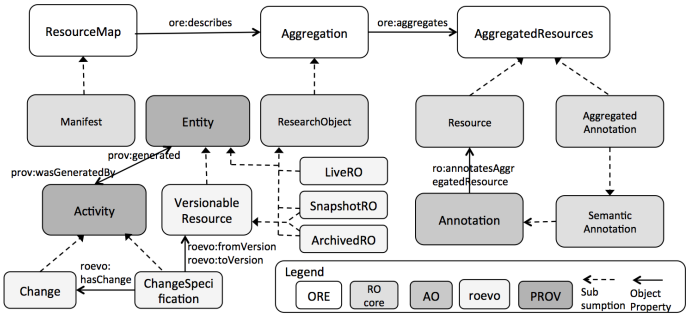

Fig. 1. Research Object ontology overview

can be exchanged as, providing a basis for interoperability between implementations. There is a strong relationship between an API and the models it uses: all resources must be exchanged in representations compatible with the models, and many (but not all) of the resources declared in an API will be instances of concepts defined in the model.

We decided to follow Linked Data [5] and REST [8] principles during the design and specification of the APIs. We argue that these approaches are complementary: they both center on the notion of resources and URIs, Web-style linking, and HTTP as the transfer mechanism. Linked Data focuses on the linking of common resources in an RDF representation, while REST uses the navigation between resources, through links, to progress application state. Therefore, the API provides separate resources for instances of the core classes from the RO model, like research objects, uploaded aggregated resources and annotations. Other classes, where instances are specific to another resource, are only represented within the resources that refer to them. The API also provides additional resources beyond the concepts defined in the underlying model in order to represent further aspects of the processing state (e.g. when assembling a new RO) or to simplify programmatic access to the service (e.g., create RO from a set of files). We also found use of existing concepts for what we initially believed would require custom API methods, e.g., DELETE an ore:Proxy to deaggregate a resource.

The suite of APIs (`http://www.wf4ever-project.org/wiki/display/docs/Wf4Ever+service+APIs`) includes several specifications for managing different aspects of ROs, including the core RO lifecycle and evolution, notifications, recommendations, quality information, workflow-centric information, etc. In the following we focus on the two basic APIs for RO management.

3.1 RO API

This API enables storing and retrieving ROs and their aggregated resources, as well as annotating them. The API is based on the *core RO model*. Using this API, ROs can be created from scratch, adding resources progressively to the RO, they can be created from a ZIP file aggregating files and folders, and they can be created from an existing RO by uploading it zipped. Similarly, ROs can be retrieved in a number of formats (using content negotiation) as RDF metadata, a ZIP archive or an HTML page. Resources can be aggregated by means of proxies (which simplifies handling resources that are stored

outside the namespace of the API service) and resources can be annotated using RDF graphs.

Summary of API operations (i) Get a list of research objects; (ii) Create/retrieve/delete a research object; (iii) Aggregate/de-aggregate a resource within a research object; (iv) Update/retrieve/delete an aggregated resource; (v) Annotate a resource; (vi) Update/retrieve/delete an annotation; (vii) Create/retrieve/delete a folder; (viii) Add/delete a resource to/from a folder.

HTTP methods GET retrieves the RO or its resources. For the RO itself and its metadata, content negotiation may be used to request a desired format of the response (e.g. ZIP of entire RO vs RDF metadata - RDF/XML, Turtle, etc.). POST create new resources: a new RO, an aggregated resource, an annotation, proxy or a new folder entry. PUT replaces the content of an existing resources, or provides content for a new resource. DELETE removes a RO or a component resource.

3.2 RO evolution API

This API enables to record the transformation of ROs based on their lifecycle, and to access the history of their evolution. The API is based on the *roevo model*. The lifecycle transformation of ROs is achieved by providing a service that performs a copy of a RO and saves the copy as a *live*, *snapshot* or *archived* research object. The client needs to make two requests to complete the operation: (i) Copy a RO - the new RO is in a transient state and is available only to the creator. It can still be modified before reaching the target evolution state; (ii) Finalise state transformation - validate that the transient RO meets the requirements for being in the target evolution state and perform the state transition. For target states like snapshot or archived, it means that the RO becomes immutable. This API also allows retrieval of the evolution provenance. The client sends a request with the URI of an RO as a query parameter and the service will return an RDF graph with selected information about the RO in question, its known copies and the RO that it was copied from.

Summary of API Operations (i) Create a copy job; (ii) Create a finalise job; (iii) Check the status of a job; (iv) Get the evolution history of a research object.

HTTP methods POST creates a job, GET retrieves the status of a job and DELETE cancel a running job. GET on the root evolution service resource provide the URIs of the resources corresponding to the above operations. Finally, a GET or HEAD to the RO is used to find the link to evolution information resource.

4 ROHub

ROHub is a digital library system for research objects that supports their storage, lifecycle management and preservation. It implements a set of APIs, being the two primary ones the RO API and the RO evolution API. Additionally, it provides the Notification API, User Management API and Access Control API (from the suite), a Solr REST API and a SPARQL endpoint. ROHub also provides a Web interface `http://www.rohub.org/portal/`, which exposes all functionalities to the users [7].

Internal architecture

Internally, ROHub[4] has a modular structure that comprises access components, longterm preservation components and the controller that manages the flow of data

[4] Source code available at: `https://github.com/wf4ever/rodl`

(see Fig. 2). ROs are stored in the access repository once created, and periodically the new and/or modified ROs are pushed to the longterm preservation repository.

The access components are the storage backend and the semantic metadata triplestore. The storage backend can be based on dLibra (`http://dlab.psnc.pl/dlibra/`) digital library system, which provides file storage and retrieval functionalities, or it can use a built-in module for storing ROs directly in the filesystem. The semantic metadata are additionally parsed and stored in a triplestore backed by Jena TDB (`http://jena.apache.org/`). The longterm preservation component is built on dArceo (`http://dlab.psnc.pl/darceo/`). It stores ROs, including resources and annotations, and monitors their quality, alerting administrators if necessary. In particular, ROHub provides fixity checking and monitors the RO quality through time against a predefined set of requirements. If a change is detected, notifications are generated as Atom feeds.

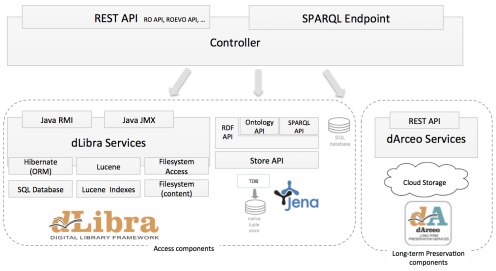

Fig. 2. ROHub internal component diagram

API implementation The starting point to the APIs in ROHub is `http://www.rohub.org/rodl/`. From there, RO API main resource is at `/rodl/ROs/`, which provides the collection of ROs, and `/rodl/ROs/ro_id` GETs a concrete RO. `/rodl/zip/create/` is the service resource where a ZIP with files and folders can be POSTed to create a new RO, while `/rodl/zip/upload/` is a service resource to which a ZIP with a complete RO can be POSTed to store this RO. The service description for the evolution API is at `/rodl/evo`. From there, the client can navigate to the service resources: (i) `/rodl/evo/copy/` - the copy resource; (ii) `/rodl/evo/finalize/` - the finalise resource; (iii) `/rodl/evo/info{?ro_id}` - the URI template of the evolution information resource.

5 Discussion

The design and specification of the APIs was conducted incrementally, co-evolving with the implementation of the services and tools, enabling users to experience results from an early stage. It was driven by technical and user requirements from use cases in experimental science (Bioinformatics, Astronomy). During development, newer versions of the API were deployed concurrently with older versions to facilitate client migration.

Creating the RO API specification took six cycles carried out over two years. Each version was implemented in ROHub, tested and validated by users and other service developers, resulting in improvements and further requirements until we reached the final version. During the specification, we had several discussions regarding the correct usage of POST and PUT operations. We agreed to use POST to create new content and PUT to update the content or resources specifications. One of the main benefits of using Linked Data and REST principles combined was that Semantic Web data structures and ontologies (RDF, OWL) are used for canonical representations of resources, which makes them particularly suitable for use with REST. This enabled the development of a common domain model with self-describing link semantics beyond the relatively simple

structures found in traditional REST deployments, and a simple mapping between the model and REST resources.

We also learned that the granularity of RO update operations was very fine, focusing on small sub-resources rather than the larger purpose of the RO. This meant that performance of some operations was lower than we would have liked. Moreover, the API ended up with close coupling between the underlying model and the operations to be performed. This was partly due to the fine granularity of operations, which somewhat goes against the goals of REST, and made evolving some aspects of the model more difficult (e.g. having annotation bodies that were or were not explicitly aggregated in the RO). Some aspects of this coupling were difficult to avoid for functions which updated the model. Yet for functions that access an RO, we have successfully achieved robust interoperability between independently developed components via the API. After the implementation phase we did realise the need for batch operations, such as being able to add multiple annotations in one call.

When implementing the RO evolution API we had to deal with time-consuming operations, and as a result the API evolved several times until we introduced the notion of asynchronous jobs. The jobs considerably simplified the API and made it more efficient and easier to use. Yet, this API can still be improved regarding the retrieval of RO evolution information, which currently is only returned as a complete RDF graph.

6 Conclusions

We have presented a suite of APIs for managing Research Objects lifecycle. The RO API and the RO evolution API enable developers to build applications/services that use or generate ROs. We built these APIs following Linked Data and REST approaches, and using a co-evolutionary approach with the implementation of services and client tools. We have shown how these API have been implemented in ROHub, a digital library for ROs, and discussed the lessons learned during this process. Finally, we identified potential improvements in these APIs, such as the introduction of batch operations.

Acknowledgements This work was partially funded by Wf4Ever EU project (FP7-270129) and its presentation was possible thanks to its partial support within FOODIE CIP project (Pilot B no.621074).

References

1. Open Archives Initiative Object Reuse and Exchange. http://www.openarchives.org/ore/1.0/toc.html
2. PROV-O: The PROV Ontology. http://www.w3.org/TR/prov-o/
3. Bechhofer, S., et al.: Why linked data is not enough for scientists. Future Generation Computer Systems (FGCS) (2011)
4. Belhajjame, K., et al.: Workflow-centric research objects: First class citizens in scholarly discourse. In: ESWC2012 Workshop on Semantic Publication (SePublica2012) (2012)
5. Bizer, C., et al.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems (IJSWIS) (3), 1–22 (Mar)
6. Ciccarese, P.: The Annotation Ontology. http://code.google.com/p/annotation-ontology
7. Palma, R., et al.: Rohubï£¡ a digital library of research objects supporting scientists towards reproducible science. In: Proceedings of ESWC2014 (sempub challenge) (MAY 2014)
8. Richardson, L., Ruby, S.: Restful web services. O'Reilly, first edn. (2007)

