



Wf4Ever: Advanced Workflow Preservation Technologies for Enhanced Science

STREP FP7-ICT-2007-6 270192

Objective ICT-2009.4.1 b) – “Advanced preservation scenarios”

D1.3v2 Wf4Ever Architecture – Phase II

Wf4Ever Lead Architect: David De Roure (OXF)

Deliverable Co-ordinating Institution: University of Oxford (OXF)

Deliverable Authors: Piotr Holubowicz (PSNC), Kevin Page (OXF), Raul Palma (PSNC)

This document

| | | | |
|----------------------|------------------------|------------------|------------|
| Document Identifier: | Wf4ever/2013/D1.3/v2.0 | Date due: | 31/03/2013 |
| Class Deliverable: | Wf4ever 270192 | Submission date: | 31/03/2013 |
| Project start date: | December 1, 2010 | Version: | V2.0 |
| Project duration: | 3 years | State: | Final |
| | | Distribution: | Public |

Wf4Ever Consortium

This document is a part of the Wf4Ever research project funded by the IST Programme of the Commission of the European Communities by the grant number FP7-ICT-2007-6 270192. The following partners are involved in the project:

| | |
|---|---|
| <p>Intelligent Software Components S.A. Edificio Testa Avda. del Partenón 16-18, 1º, 7ª Campo de las Naciones, 28042 Madrid Spain Contact person: Dr. Jose Manuel Gómez-Pérez E-mail address: jmgomez@isoco.com</p> | <p>University of Manchester Department of Computer Science, University of Manchester, Oxford Road Manchester, M13 9PL United Kingdom Contact person: Professor Carole Goble E-mail address: carole.goble@manchester.ac.uk</p> |
| <p>Universidad Politécnica de Madrid Departamento de Inteligencia Artificial Facultad de Informática, UPM 28660 Boadilla del Monte, Madrid Spain Contact person: Dr. Oscar Corcho E-mail address: ocorcho@fi.upm.es</p> | <p>University of Oxford Department of Zoology University of Oxford South Parks Road, Oxford OX1 3PS United Kingdom Contact person: Dr. Jun Zhao / Professor David De Roure E-mail address: {jun.zhao@zoo.ox.ac.uk, david.deroure@oerc.ox.ac.uk}</p> |
| <p>Poznań Supercomputing and Networking Center Network Services Department Poznań Supercomputing and Networking Center Z. Noskowskiego 12/14, 61-704 Poznan Poland Contact person: Dr. Raúl Palma de León E-mail address: rpalma@man.poznan.pl</p> | <p>Instituto de Astrófica de Andalucía Dpto. Astronomía Extragaláctica Instituto Astrofísica Andalucía Glorieta de la Astronomía s/n 18008 Granada, Spain Contact person: Dr. Lourdes Verdes-Montenegro E-mail address: lourdes@iaa.es</p> |
| <p>Leiden University Medical Centre Department of Human Genetics Leiden University Medical Centre Albinusdreef 2, 2333 ZA Leiden The Netherlands Contact person: Dr. Marco Roos E-mail address: M.Roos1@uva.nl</p> | |

Change Log

| Version | Date | Amended by | Changes |
|---------|------------|------------------|--|
| 1.0 | 30-11-2011 | Kevin Page | See D1.3v1 for earlier change log |
| 1.1 | 18-03-2013 | Raul Palma | Document outline based on v1 Addendum |
| 1.2 | 19-03-2013 | Raul Palma | Writing section compliance with standards |
| 1.3 | 21-03-2013 | Piotr Holubowicz | Input to section compliance with standards |
| 1.4 | 22-03-2013 | Raul Palma | Updating Section of Wf4Ever reference implementation |
| 1.5 | 25-03-2013 | Raul Palma | Updating Section of Example Orchestration |
| 1.6 | 25-03-2013 | Kevin Page | Merge of D1.3v1 |
| 1.7 | 25-03-2013 | Kevin Page | Major restructuring of sections |
| 1.8 | 26-03-2013 | Piotr Holubowicz | Updating API details to current state |
| 1.6 | 27-03-13 | Kevin Page | Split Section 3 |
| 1.7 | 27-03-13 | Kevin Page | 7.1 and .72 to section 2, 7.3 to section 3, preservation appendixes to section 6. |
| 1.7 | 27-03-13 | Raul Palma | Updating Section Architecture Overview |
| 1.8 | 28-03-13 | Raul Palma | Updating Section WfEver Models & Section Design methodology |
| 1.9 | 28-03-13 | Piotr Holubowicz | Updating Section Wf4Ever Services, Section Clients & Section Development methodology |
| 1.10 | 28-03-13 | Piotr Hołubowicz | Sections 2.2, 2.3, 2.4, 2.5, 6 |
| 1.11 | 28-03-13 | Kevin Page | Sections 1 & 5 |
| 1.12 | 28-03-13 | Kevin Page | Released for QA |
| 1.13 | 29-03-13 | Piotr Hołubowicz | Post-QA fixes to 2.3 and 6. |
| 1.14 | 29-03-13 | Raul Palma | Updating references |
| 1.15 | 29-03-13 | Piotr Hołubowicz | Post-QA fixes |
| 1.16 | 31-03-13 | Kevin Page | Post-QA fixes |
| 1.17 | 1-04-13 | Raul Palma | Post QA fixes |
| 2.0 | 31-03-13 | Kevin Page | Final document release |

Executive Summary

The Wf4Ever project has produced a software architecture for the design and implementation of scientific workflow preservation systems, together with a reference implementation instantiating the architecture and enabling the preservation and efficient retrieval of scientific workflows:

- The Wf4Ever Architecture provides service patterns through the prescription of a number of models and associated APIs in support of the production and preservation of scientific workflows. It adopts of Web Architecture, incorporating Linked Data and RESTful approaches.
- The Wf4Ever Toolkit is a reference implementation of services and applications that adheres to this architecture a user-centric software environment for the preservation of workflows in the bioinformatics and astronomy domains.

Building upon development of the Research Object (RO) model in the project [26], the Wf4Ever Architecture enables the capture, preservation and re-use of ROs, not simply as a semantic encoding, but as building blocks for interoperability and co-ordination in a distributed architecture of services and their APIs. The architecture combines new tools and services with the extension of an established scientific workflow sharing infrastructure (myExperiment) with RO preservation capabilities, demonstrating the future applicability and adaptability of the Architecture to other applications.

Table of contents

| | |
|--|-----------|
| Wf4Ever Consortium | 2 |
| Change Log | 3 |
| Executive Summary..... | 4 |
| Table of contents | 5 |
| List of Figures | 10 |
| 1 Introduction..... | 11 |
| 1.1 An introduction to the Wf4Ever Architecture..... | 11 |
| 1.2 Document outline | 12 |
| 1.3 How to use this document | 12 |
| 2 Wf4Ever Architectural Design Principles..... | 13 |
| 2.1 Aims | 13 |
| 2.2 Principles | 13 |
| 2.2.1 <i>Web Architecture and Web APIs</i> | 13 |
| 2.2.2 <i>REST</i> | 14 |
| 2.2.3 <i>Linked Data</i> | 15 |
| 2.2.4 <i>Summary of principles</i> | 15 |
| 2.3 Architectural structure: Models, APIs, and Services..... | 16 |
| 2.4 Development of Services and APIs | 19 |
| 3 Wf4Ever Architecture Overview | 21 |
| 3.1 Architectural Functionality..... | 21 |
| 3.2 Wf4Ever Models..... | 24 |
| 3.3 Wf4Ever APIs..... | 25 |
| 3.3.1 <i>RO API</i> | 25 |
| 3.3.2 <i>RO EVO API</i> | 26 |
| 3.3.3 <i>Checklist Evaluation API</i> | 26 |
| 3.3.4 <i>Stability Evaluation API</i> | 27 |
| 3.3.5 <i>Recommendation API</i> | 28 |
| 3.3.6 <i>Workflow Abstraction API</i> | 29 |
| 3.3.7 <i>Workflow Runner API</i> | 29 |
| 3.3.8 <i>Workflow-RO Transformation API</i> | 30 |
| 3.3.9 <i>User Management API</i> | 31 |
| 3.4 Wf4Ever Services and Clients | 31 |
| 3.4.1 <i>RO Digital Library (RODL)</i> | 32 |

| | | |
|----------|---|-----------|
| 3.4.2 | <i>myExperiment</i> | 33 |
| 3.4.3 | <i>Workflow Runner Service</i> | 33 |
| 3.4.4 | <i>Checklist Evaluation Service</i> | 33 |
| 3.4.5 | <i>Stability Evaluation Service</i> | 34 |
| 3.4.6 | <i>Recommender Service</i> | 34 |
| 3.4.7 | <i>Workflow-RO Transformation Service</i> | 35 |
| 3.4.8 | <i>Workflow Abstraction Service</i> | 35 |
| 3.4.9 | <i>RO-enabled myExperiment</i> | 36 |
| 3.4.10 | <i>RO Portal</i> | 36 |
| 3.4.11 | <i>RO Manager</i> | 37 |
| 3.4.12 | <i>Collaboration Spheres</i> | 37 |
| 4 | Example architecture orchestration | 39 |
| 5 | API Specifications | 42 |
| 5.1 | RO API..... | 42 |
| 5.1.1 | <i>API usage</i> | 42 |
| 5.1.2 | <i>HTTP methods</i> | 42 |
| 5.1.3 | <i>Link relations</i> | 42 |
| 5.1.4 | <i>Resources and formats</i> | 43 |
| 5.1.5 | <i>Sample client-server interactions</i> | 44 |
| 5.1.5.1 | Creating a new Research Object..... | 44 |
| 5.1.5.2 | Aggregating a resource | 45 |
| 5.2 | RO EVO API | 49 |
| 5.2.1 | <i>API usage</i> | 49 |
| 5.2.2 | <i>HTTP methods</i> | 49 |
| 5.2.3 | <i>Link relations</i> | 49 |
| 5.2.4 | <i>Resources and formats</i> | 50 |
| 5.2.5 | <i>Sample client-server interactions</i> | 51 |
| 5.2.5.1 | Creating an RO copy | 51 |
| 5.2.5.2 | Getting a job status..... | 51 |
| 5.3 | Checklist Evaluation API..... | 52 |
| 5.3.1 | <i>API usage</i> | 52 |
| 5.3.2 | <i>HTTP methods</i> | 53 |
| 5.3.3 | <i>Link relations</i> | 53 |
| 5.3.4 | <i>Resources and formats</i> | 53 |

| | | |
|---------|--|----|
| 5.3.5 | <i>Sample client-server interactions</i> | 53 |
| 5.3.5.1 | Retrieving the service document | 53 |
| 5.3.5.2 | Evaluating the RO..... | 54 |
| 5.4 | Stability Evaluation API..... | 54 |
| 5.4.1 | <i>API usage</i> | 55 |
| 5.4.2 | <i>HTTP methods</i> | 55 |
| 5.4.3 | <i>Link relations</i> | 55 |
| 5.4.4 | <i>Resources and formats</i> | 55 |
| 5.4.5 | <i>Sample client-server interactions</i> | 56 |
| 5.4.5.1 | Retrieving the service document | 56 |
| 5.4.5.2 | Retrieving the RO stability | 56 |
| 5.5 | Recommendation API..... | 57 |
| 5.5.1 | <i>API usage</i> | 57 |
| 5.5.2 | <i>HTTP methods</i> | 58 |
| 5.5.3 | <i>Link relations</i> | 58 |
| 5.5.4 | <i>Resources and formats</i> | 59 |
| 5.5.5 | <i>Sample client-server interactions</i> | 59 |
| 5.5.5.1 | Retrieving the service document | 59 |
| 5.5.5.2 | Getting the recommendation set | 60 |
| 5.6 | Workflow Abstraction API | 61 |
| 5.6.1 | <i>API usage</i> | 61 |
| 5.6.2 | <i>HTTP methods</i> | 61 |
| 5.6.3 | <i>Link relations</i> | 61 |
| 5.6.4 | <i>Resources and formats</i> | 62 |
| 5.6.5 | <i>Sample client-server interactions</i> | 62 |
| 5.6.5.1 | Calling the search operation | 62 |
| 5.6.5.2 | Calling the recommend operation..... | 63 |
| 5.7 | Workflow Runner API..... | 63 |
| 5.7.1 | <i>API usage</i> | 63 |
| 5.7.2 | <i>HTTP methods</i> | 64 |
| 5.7.3 | <i>Link relations</i> | 64 |
| 5.7.4 | <i>Resources and formats</i> | 64 |
| 5.7.5 | <i>Sample client-server interactions</i> | 65 |
| 5.7.5.1 | Submitting a new run to workspace..... | 65 |

| | | |
|----------|---|-----------|
| 5.7.5.2 | Retrieving the workflow status | 66 |
| 5.8 | Workflow-RO Transformation API | 67 |
| 5.8.1 | API usage | 67 |
| 5.8.2 | HTTP methods..... | 67 |
| 5.8.3 | Link relations..... | 67 |
| 5.8.4 | Resources and formats..... | 67 |
| 5.8.5 | Sample client-server interactions..... | 68 |
| 5.8.5.1 | Creating a conversion job | 68 |
| 5.8.5.2 | Checking a job status | 68 |
| 5.9 | User Management API..... | 69 |
| 5.9.1 | API usage | 69 |
| 5.9.2 | Conformance with external APIs | 69 |
| 5.9.3 | HTTP methods..... | 70 |
| 5.9.4 | Link relations..... | 70 |
| 5.9.5 | Resources and formats..... | 70 |
| 5.9.6 | Sample client-server interactions..... | 71 |
| 5.9.6.1 | Creating a user. | 71 |
| 6 | Compliance with Preservation Standards | 72 |
| 6.1 | Mapping to OAIS functional entities..... | 72 |
| 6.1.1 | OAIS Functional Entities..... | 72 |
| 6.1.2 | Information Objects in Wf4Ever..... | 74 |
| 6.1.3 | Mapping to OAIS functions | 74 |
| 6.1.3.1 | Ingest functional entity in Wf4Ever | 74 |
| 6.1.3.2 | Archival Storage functional entity in Wf4Ever | 75 |
| 6.1.3.3 | Data Management functional entity in Wf4Ever | 76 |
| 6.1.3.4 | Administration functional entity in Wf4Ever | 76 |
| 6.1.3.5 | Preservation Planning functional entity in Wf4Ever..... | 78 |
| 6.1.3.6 | Access functional entity in Wf4Ever | 78 |
| 6.2 | Mapping to UC3 Microservices..... | 79 |
| 6.2.1 | Curation Micro-Services at UC3: the Merritt approach..... | 79 |
| 6.2.2 | The Merritt approach in Wf4Ever..... | 81 |
| 7 | Summary and Reflection | 84 |
| 8 | References | 86 |

Appendix A: Wf4Ever Architecture Development Methodology88

List of Figures

| | |
|--|----|
| Figure 2.1 The generic relationship between Models, APIs, and Services | 16 |
| Figure 2.2 Overview of the models, APIs, and services | 18 |
| Figure 2.3 Summary of options for interaction between components | 19 |
| Figure 4 The Stability Evaluation API | 28 |
| Figure 5 Deployment Diagram showing some Interactions between Wf4Ever components | 40 |

1 Introduction

1.1 An introduction to the Wf4Ever Architecture

The Wf4Ever project has developed a software architecture for the design and implementation of scientific workflow preservation systems, together with a reference implementation instantiating the architecture and enabling the preservation and efficient retrieval of scientific workflows across a range of domains. The primary focus of the architecture is interoperability and compatibility between Wf4Ever software components, and in this regard the focus of interoperability is through the Research Object (RO) and associated models – this is an RO-centric Architecture.

The architecture sets out to be compliant with preservation standards (see Section 6) and best practices from Web and Linked Data approaches (Section **Error! Reference source not found.**), to combine workflow lifecycle management, social networks and digital libraries, and to unify and extend them through a Research Object (RO) centric architecture. The architecture provides new services and clients, including the RO Digital Library and RO Manager, and extends an established scientific workflow sharing infrastructure (myExperiment [3]) with RO preservation capabilities.

This document describes both the *Wf4Ever Architecture* for scientific workflow preservation, and the *Wf4Ever Toolkit* reference implementation. It serves three purposes:

1. To identify the abstract functionality of the architecture, and relate how this is exposed through definition of its Application Programming Interfaces (APIs), which in turn are based upon the models developed in the project, particularly the Research Object (RO) Model;
2. To detail the reference implementation of the Architecture – the Wf4Ever Toolkit – and how this provides an instantiation of the Wf4Ever Services and Clients;
3. To describe the general principles and practice for interoperability between services, comprising Models, APIs, and Services, and the wider compatibility with existing preservation standards and models.

Within the architecture we provide simple, lightweight, and adaptable APIs to Wf4Ever functionality for compatibility with and within the Wf4Ever Toolkit, which can be easily adopted and re-used in new tools, with new content, and in new domains.

To this end we have employed Linked Data[5] and REST[4] as our main architectural principles. We argue that these approaches, both centred on the notion of resources, are complementary: both are based on the use of URIs, Web-style linking, and HTTP as the transfer mechanism. Linked Data[2] focuses on the linking of common resources in an RDF representation, while REST uses the navigation between resources to progress application state (such that client-server interaction are stateless). In combination this leads to a design method driven by the identification of resources and their representations[6].

These Models, APIs, and Services are the glue that binds the Wf4Ever architecture, but they do not describe the *functionality* required to implement a workflow preservation system. Hence, along with this structure, our architecture defines a set of entities for grouping and categorizing the required functionality, aligning them with concepts from the OAIS reference model[1].

1.2 Document outline

Section 2 describes the aims and principles underlying the Architecture design, including adoption of REST and Linked Data principles, and how this is manifest in the Models, APIs and Services found in the Architecture.

Section 3 gives a breakdown of the functionality provided by the Wf4Ever Architecture, then lists the Models and APIs used and provided by the Architecture categorised according to function. It then details the Services and Clients that implement and use the APIs as instantiated in the Wf4Ever Toolkit – the reference implementation of the Architecture.

Section 4 provides an illustrative example of how components in the Wf4Ever Toolkit are orchestrated through the Architecture.

Section 5 provides a detailed specification of the APIs that comprise the Wf4Ever Architecture, including the HTTP methods and Link Relations provides, supported resources and formats, and sample client-server interactions at the HTTP level.

Section 6 details compliance of the Wf4Ever Architecture to existing preservation standards, including a mapping to the OAIS abstract model and an alignment with the Merritt approach.

Appendix A gives details of the development environment and process under which the Architecture and Toolkit were produced.

1.3 How to use this document

This deliverable provides a large amount of information at varying levels of detail. We suggest several “trails” through the document depending on your interest:

- If you wish to understand the varied functionalities provided by the Wf4Ever Architecture and how they interoperate: *read sections 3 and 4.*
- If you wish to write clients or services that are compatible with the Wf4Ever Architecture through use of its APIs: *read sections 3 then 5.*
- If you wish to understand the process by which the Wf4Ever Architecture was developed: *read sections 2, 3, and Appendix A.*
- If you wish to evaluate alignment between the Wf4Ever Architecture and OAIS or Merritt: *read sections 3 and 6.*

2 Wf4Ever Architectural Design Principles

The previous sections have dealt with the structure and components of the Wf4Ever Architecture and the development process through which the architecture, including its components and APIs, were realised. In this section we detail the underlying *aims* and *principles* within which the architecture was developed. As such in this section we do not go into the rationale for any specific Wf4Ever API or service (which can be found in the other sections of this document) but rather the more abstract motivations that led to the Wf4Ever Architecture taking the form and function it now has.

2.1 Aims

At the start of Architecture development within the project we identified six aims we wished the Architecture to fulfil:

1. Provide the framework within which a workflow preservation system can be constructed, and to ensure the compatibility of components within this system (i.e. interoperability within the Wf4Ever Toolkit).
2. Be adaptable to an agile development process and the co-evolution of components within the system alongside user requirements (see Section **Error! Reference source not found.**).
3. Provide enough flexibility to accommodate changes in design as fundamental research is undertaken in the development of components; but enough coherence to bring components together at an early stage for integration, testing, and user feedback.
4. Adopt, extend, or enable exchange with existing standards and best practice. For example through adoption of the Linked Data principles, and alignment with the OAIS reference model.
5. Build upon existing tools from the scientific workflow and digital library communities, enabling the integration of some within the Wf4Ever Toolkit, and the provision of simple interfaces to enable future data exchange with others.
6. Provide simple, lightweight, and adaptable interfaces to Wf4Ever functionality so that it can be easily adopted and re-used in new tools, with new content, in new domains.

2.2 Principles

In this sub-section we explore the principles we apply to the design of the Wf4Ever architecture; that is, the technical manifestation of the aims.

2.2.1 Web Architecture and Web APIs

To both provide an architecture that is flexible and decoupled to adapt to an agile and continuous development, and to provide simple, lightweight, interfaces for access to Wf4Ever functionality, we adopt a Resource Oriented approach based on Web Architecture (compared to a more traditional Service Oriented approach).

Lightweight web applications and mashups are now an established phenomenon. They are typically constructed by re-purposing as much existing functionality and data as possible, developing new code to tie

2013 © Copyright lies with the respective authors and their institutions.

current systems together in interesting and insightful ways. This re-use relies on access to data and services through Application Programming Interfaces (APIs) that are easy to understand and simple to use.

We focus on two well-tested approaches: REST and Linked Data.

2.2.2 REST

Representational state transfer (REST) is a set of design principles which have been popularly and successfully adopted in many ('RESTful') web services, and is typically framed as an alternative to 'heavyweight' web services, including as the WS-* family.

REST aims to capture the features of the Web which have enabled it to scale so successfully:

- everything is a resource which is addressable
- resources have multiple representations
- relationships between resources are expressed through hyperlinks
- all resources share a common interface with a limited set of operations
- client server communication is stateless

Although HTTP is RESTful, REST is not defined in terms of, nor limited to, the web [4], and while there have been attempts to clarify the application of REST to web services through definition of a Resource Oriented Architecture [5] the term is still often loosely, sometimes incorrectly, applied.

The key principle of REST is the use of resources for specific things that we wish to reference, and the referencing of these resources using URIs. Representations of these resources - encoded in a particular format - are then accessed through the URI, usually using HTTP. REST limits the operations exposed by a web service to a small, well-defined, standard, set - for HTTP these are GET, POST, PUT, and DELETE, which are intrinsically compatible with (and a part of) Web architecture. This contrasts with a potentially expansive set of operators (for RPC style web services) or message contracts (for SOA style web services).

This constrained set of operations leads to a design process focused on correctly identifying the resources that should be exposed for a service and their representations; while the interface to the resources is simple, the number of resources – every piece of information that could be served - is likely to be many, with a URI for each. Since an application client cannot possibly know of every URI in existence it is important that resources hyperlink to other resources so a client application can navigate around them.

A Resource Oriented Architecture also requires statelessness – that each HTTP operation is totally separate from any other. As such, any state the service has must also be exposed as a resource; an application client enters that state by accessing the URI for that resource; to enter another state a will use another URI.

Any application state a service requires to provide a representation of a resource must be completely contained within the request to the server (where the application is the client software processing and modifying the resource representations returned by the service). Transitions in application state are made by moving - "navigating" in a web sense - to alternate resources provided as URI links in the representation of a resource provided returned by the server.

2.2.3 Linked Data

Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sets, and can in turn be linked to from external data sets. [2]

The Linked Data movement has achieved considerable success constructing a semantic Web of Data. While earlier research focused on building a stack to enable reasoning and logic this more recent effort makes large scale distribution and linking a priority.

The linked data web is constructed atop a number of principles and best practice. These were first set out by Berners-Lee in his 2006 design note [6]:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things

While the use of URIs is common throughout the Semantic Web - not least as the basic element of RDF - the requirement to use HTTP URIs sets Linked Data deployment apart. It is a departure from the use of URIs purely as unique identifiers within the graph; in Linked Data they are also a means of retrieving parts of the graph relevant to that resource - the URIs can be dereferenced.

This dual use of HTTP URIs does not, however, remove the need to distinguish between the two uses: Linked Data best practice allows a web client to tell the difference between a URI representing the person (a non-information resource) and a URI providing information about Tim (an information resource); even if, in this linked data web, we dereference the former to retrieve the latter (through use of 303 redirects).

As with REST, URIs are both a retrieval mechanism and identifier; again the URI is opaque, and a the primary method for a client to access resources should be through through the links provided from one resource representation to another (not semantic encoded within the URI).

Unlike REST, the Linked Data principles favour a single underlying representation (RDF). The RDF model is itself very flexible and has been adapted to many different domains - this basic model level compatibility is a strong starting point for interoperability amongst clients and services.

2.2.4 Summary of principles

When surveying Web architecture and the means to access data through Web APIs, two prominent methodologies are noted which both adopt an approach centred around the notion of resources: REST and Linked Data.

In the sense that they are both attempting to identify and replicate the successful architectural style and scalability of the Web, REST and Linked Data have shared aspirations; their focus on the use of resource URIs, Web style linking, and implementation using HTTP provide a common technical bedrock which in, in both cases, leads to design strongly focused on the identification of resources and their representations.

While in other respects differences can be identified [7], on balance there is a strong case for a complementary approach building on both, i.e.:

1. Agile development of lightweight mashups is best supported by Resource Oriented service architectures. Complexity for mashup developers is reduced through the simplification of access methods espoused by REST. To develop a good API of this type requires careful identification and design of resources by the service provider.
2. Identification of these resources must be undertaken within the context of the domain of the data.
3. Semantic Web data structures and ontologies (RDF, RDFS, and OWL) should be used for canonical representations of resources; they share a common architectural heritage that makes them particularly suitable for use with REST. This enables development of a common domain model with self-describing link semantics beyond the relatively simple structures found in traditional REST deployments.
4. Identify resources to support not only the domain model, but also the API and its use. Provide Linked Data through content negotiation and a SPARQL endpoint, but also identify resources to enable programmatic access to services from RESTful applications where hypertext is the engine of application state.
5. RESTfully provide other representations, derived from the domain model, to enrich the service for easy application development and compatibility with existing tooling.

2.3 Architectural structure: Models, APIs, and Services

The architecture is primarily a mechanism for ensuring the compatibility of interactions between clients and services through the definition of interfaces.

In adopting a RESTful approach the definition of interfaces becomes a process of identifying the resources and representations exposed by a service (and/or required by a client). In the Wf4Ever architecture we describe this process, and the interfaces, in terms of Models, APIs, and Services.

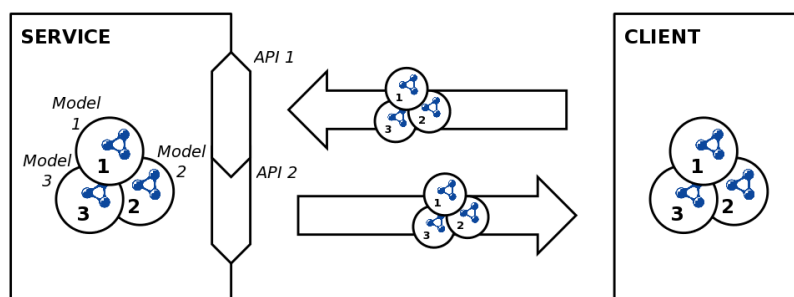


Figure 2.1 The generic relationship between Models, APIs, and Services

The relationship between Models, APIs, and Services is illustrated in Figure 5.1 and defined as follows:

- **Models** are the data structures that encode the concepts and relationships of information. In Wf4Ever there is a default assumption that our core models will be semantic structures using RDF

and ontologies, however domain information (such as dense astronomy data) may be modelled using data structures applicable to the task. Any transfer of information between client and service, and therefore the representations used for this transfer, must adhere to the model (this does not preclude multiple representations, so long as they preserve the model).

Specific models as implemented in the Wf4Ever Architecture are described in Section 3.2.

- Wf4Ever **APIs** are RESTful; they are defined sets of resources and resource patterns provided by a service that a client can reliably expect to interact with, and a defined set of representations (and therefore models) these resources can be exchanged in. An API defines a *minimal* set of resources and representations for guaranteed compatibility, others may be provided by a service.

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.

Because a RESTful client will transition states by navigation, through links, from one resource to another, an API must provide *additional* resources beyond those concepts defined in the model to simplify programmatic access to a service. It also follows that since representations conform to models, navigation between resources is also in terms of the models.

For example, an API providing workflows might be expected to use a workflow model that defines structures for individual and collections of workflows. These individual workflows would be resources exposed as part of the API. However the API might also specify a resource which is the “top 10 workflows”, which would return a collection representation according to the model, but of a resource not defined by the model.

To maintain simplicity in client implementation it is desirable to produce APIs that extend well-defined core models for programmatic access. These can then be combined and extended into more sophisticated APIs for complex services, and clients that require such functionality can implement the extended interfaces only as and if required.

In adopting the Linked Data principles a large number of resources are expected to be provided for resources with an RDF model - indeed all instances as defined by the model might reasonably be expected to have HTTP resources with RDF representations (of some valid serialisation). The API may or may not include all of these resources in its minimal set of interfaces for interoperability, and may still provide further resources to enhance programmatic access.

*Specific APIs as implemented in the Wf4Ever Architecture are introduced in Section **Error! Reference source not found.**, and specified in detail in Section **Error! Reference source not found.***

- **Services** implements functionality that is provided to multiple clients. They expose this functionality through APIs - there may be use of one or several APIs depending on the focus or plurality of the

functionality provided, which in turn will be based upon the models used when storing and operating upon information within the service.

Services are defined by the set of APIs they support and the functionality implemented internally. Functionality might be implemented internally that is not then exposed through APIs, but this is out of scope of the architecture which focuses on interoperability.

Service implementations might provide unique functionality or might be one of many implementations of a shared service pattern. In the latter case a service pattern is declared in terms of the APIs a service must implement to be compatible

Since the architecture is concerned with interoperability and compatibility between software components, it does not prescribe the internal data structures of services; however, it is likely that services within the Wf4Ever toolkit will adopt the models internally (e.g. the RO Model as used within RODL and myExperiment) and other internal services may also. However for compatibility services *may* map to the API from other internal data structures.

*Specific Services as implemented in the Wf4Ever Architecture are described in Section **Error!** Reference source not found..*

A summary of options for interaction between components described above is defined in ; a specific instantiation of this relationship for the RO model, RO API, and RODL Service is shown in Figure 5.2.

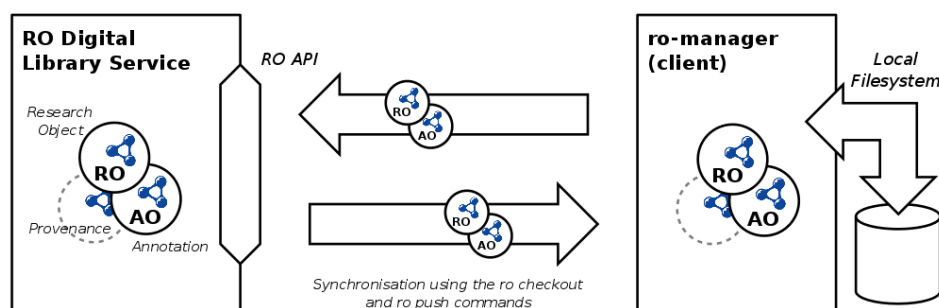


Figure 2.2 Overview of the models, APIs, and services

A note on distribution.

The definition of APIs, and indeed the architecture, is to ensure compatibility between services and clients (or other services). The adoption of RESTful Web APIs could imply distribution of such services, or at least distributed access to the services by clients over the Internet. The architecture does not mandate such distribution; while it is an aim of the architecture to support this type of interaction, it is also recognised that performance of an overall system is an important factor which may be degraded by the adoption of a synchronous fully distributed implementation (for example, previous experience of recommendation

implementation has been closely coupled to sources of social and sharing data, which may not perform adequately if distributed). For example, caching was used to improve performance of the interaction between myExperiment as a client receiving ROs through the RO API from RODL.

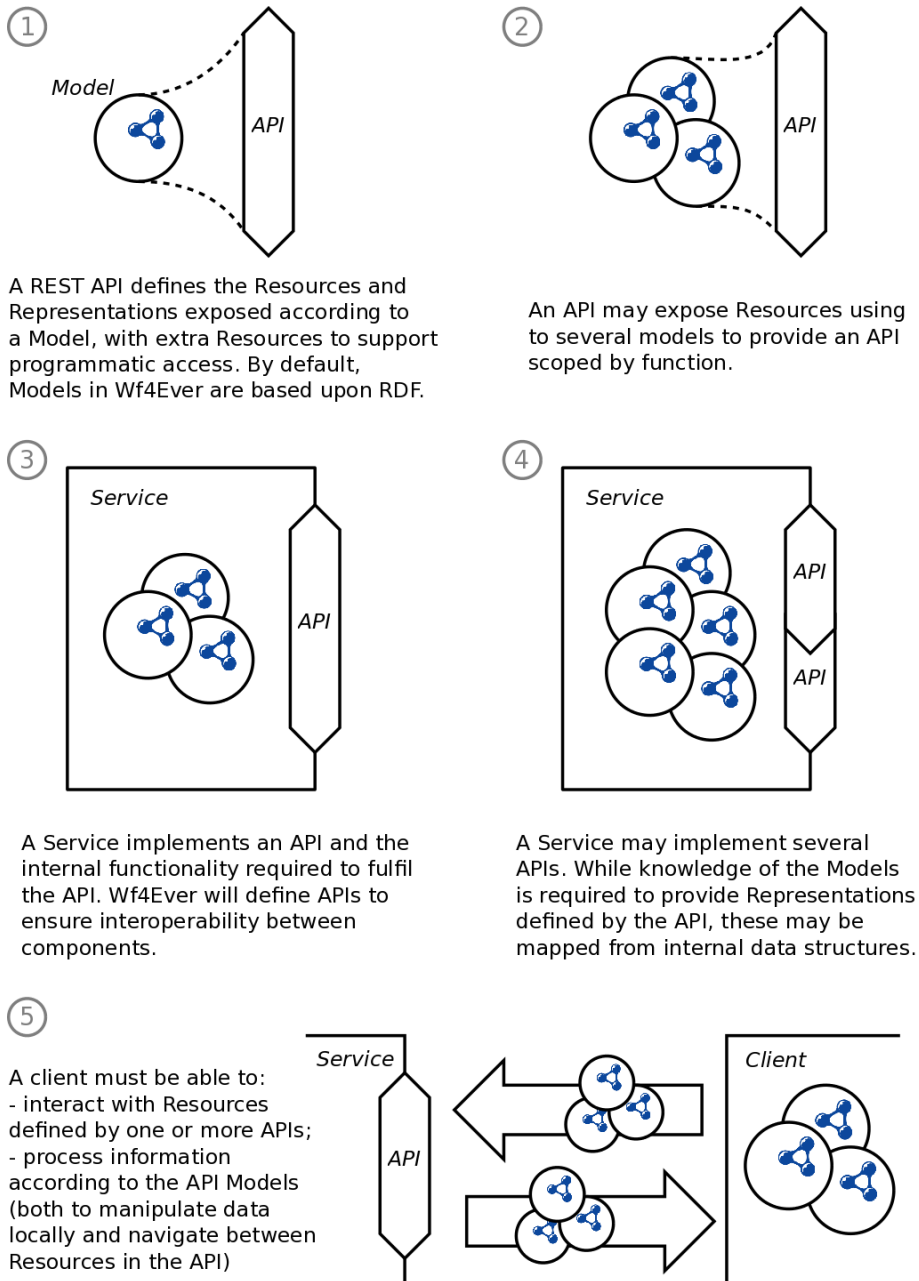


Figure 2.3 Summary of options for interaction between components

2.4 Development of Services and APIs

As described in the previous sub-section within the Wf4Ever Architecture, Services implement functionality that is provided to multiple clients, and expose this functionality through APIs. A Service can implement multiple APIs.

In Wf4Ever deliverables [D2.1, D3.1, D4.1, D5.1, D6.1] numerous requirements for functionality were identified when implementing a system for workflow preservation. These were interpreted through the principles described in Section **Error! Reference source not found.**, and through an iterative process of co-development and implementation of models and services, the APIs were stabilised thus:

- the core of each API is the resources exposed according to the models used through the API. Since several services share use of common models, these form the basis for candidates for common APIs. The core RO API, for example, is implemented in several services and mirrored in the Workflow Runner API;
- APIs build upon their models by providing extra resources and representations to support programmatic access to the functionality of a service. As services were developed, where common functionality is required (e.g. storage and retrieval of ROs) this will be refactored into common APIs that have been used by multiple services, the RO API being the prime example;
- Remaining APIs are unique to a single service implementation within the Wf4Ever Toolkit, but functionally is distinct enough to encourage use by multiple clients and potential re-implementation of the API by third-party services wishing to achieve interoperability with Wf4Ever.

3 Wf4Ever Architecture Overview

3.1 Architectural Functionality

The Wf4Ever architecture describes interfaces to functionalities required by a scientific workflow preservation infrastructure. As part of the architecture definition we have identified a number of these functionalities, grouping them into entities representing areas addressed by our preservation infrastructure, typified by the OAIS reference model. Within the abstract framework of the architecture – that is, the development of models, APIs, and services through REST and Linked Data principles as outlined in Section 4 - this work followed a continuous design cycle of co-evolution alongside the implementation of prototypes and tools, and drawing from experiences in workflow lifecycle management, social networks, semantic web and digital library communities. In identifying these functional entities, we also aim to aid designers of future systems by providing a more precise set of terms and concepts for use as reference model, and to ease alignment of our architecture with standards such as the OAIS reference model (see Section 6).

The functional entities and the broad interactions between them are illustrated in Figure 3 and described in the rest of this section.

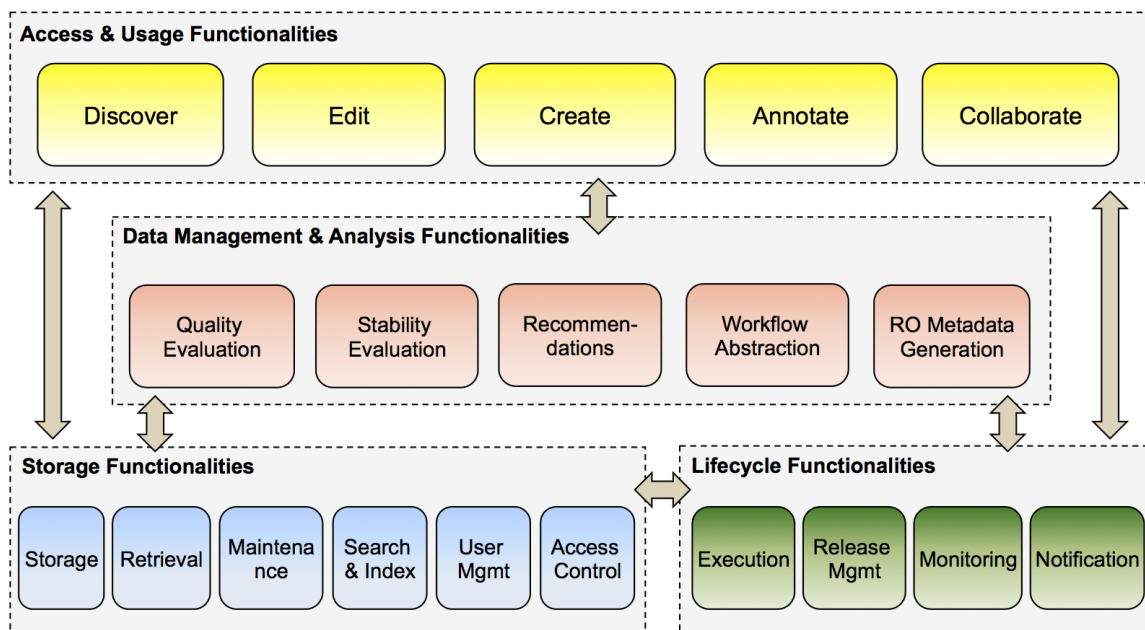


Figure 3. Wf4Ever Functionalities

Access & Usage functionalities enable consumers (users, agents, other services) to determine the existence, description, location and availability of stored Research Objects (RO) resources; to request and receive RO resources; to create, edit and annotate RO resources and to collaborate among consumers. In general, these are *user-level* functionalities, including:

- *Discover*: find, locate, request and receive RO resources, including RO aggregated resources for viewing or performing some action upon.

- *Edit*: update or delete ROs and their aggregated resources, including provenance information and other related metadata.
- *Create*: Build new ROs and aggregate resources, including provenance information and other related metadata. This functionality should also enable receiving existing aggregations of scientific data and transforming them into RO compliant resources.
- *Annotate*: manage annotations of ROs or aggregated resources. Annotations can be user or system generated structured in compliance with the RO model.
- *Collaborate*: provide mechanisms to improve the sharing and reuse of ROs and user experiences through visual metaphors that are simple and adaptable to different collaboration scenarios.

Data Management & Analysis functionalities generate, maintain and provide access to added-value data derived from, or related to, RO resources; in general these form an **extension-level** that augments the Storage and Lifecycle functionalities. In the Wf4Ever Architecture, this data includes quality-related information about the RO resources (such stability or completeness), recommendations and metadata extracted from aggregated resources. Particular functionalities identified at this stage include:

- **Stability evaluation**: validate that changes to an RO have not compromised its ability to achieve its original purpose. Specialist algorithms must evaluate each of the actions captured in the provenance trail of an RO. This evaluation can give an overview of the RO quality throughout time.
- **Quality evaluation**: evaluate different quality dimensions of an RO, such as its completeness (all information needed by the RO is present), executability (all resources and dependencies needed to run the RO are present) and repeatability (all resources and metadata to repeat an experiment are present)..
- **Recommendation**: provide recommendations of ROs, aggregated resources and relevant users, based on techniques such as keyword content-based and collaborative filtering, sorted by relevance.
- **Workflow Abstraction**: enable users to search for workflows by its functionality or other properties and recommend frequent workflow processes that appear after a predefined list of processes.
- **RO metadata generation**: extract metadata from individual resources, transform it into annotations conforming to the RO model, and generate or enrich an RO with these resources..

Storage functionalities provide the underlying mechanisms for the storage, maintenance, search and retrieval of RO resources, ensuring appropriate levels of protection, and are considered to be the **foundation-level** of the architecture. They provide basic capacities to the preservation infrastructure, concerned mainly with the static aspects of preservation, e.g., adding resources to permanent storage, managing internal data models, maintaining the RO resources associated semantic metadata, indexing, searching and protection of resources, basic versioning mechanisms, etc. Specific functionalities identified so far include:

- **Storage**: keep ROs, aggregated resources, and their associated semantic metadata in permanent storage, and provide identification of these resources for location and retrieval. As such, the store should support digital objects as well as semantic information. It also duplicates the digital objects

that are kept under custody, enabling disaster recovery mechanisms and ensuring appropriate levels of protection of the stored resources.

- Retrieval: provide the means to recover the stored resources and semantic metadata.
- Maintenance: manage internal data models, maintain the relations between ROs and their aggregated resources, and keep the associated metadata up to date. This functionality should also provide basic versioning mechanisms to keep modified copies of aggregated resources.
- Search & Index: populate and maintain indexes of the resources, and provide search mechanisms based on these indexes.
- User management: maintain and manage the identity and credentials of users. It should enable the creation, modification and retrieval of user information and manage their association with other credentials. It supports the access control mechanisms.
- Access control: define and enforce access control policies over ROs and optionally their aggregated resources, based on the user's credentials, conforming to the security measures required.

Lifecycle functionalities support the dynamic nature of RO resources. They include the execution of RO resources, such as workflows, the management of RO evolution, including versioning and curation activities, and the provision of monitoring and notification mechanisms to ensure the correct preservation of ROs. In general these are **foundation-level** functionalities, providing basic capabilities to the preservation infrastructure with respect to the dynamic aspects of preservation. Specific functionalities identified thus far include:

- Execution: run RO resources, such as workflows. This functionality is primarily concerned with live ROs, that is, ROs that are under active evolution. It also enables to verify the runnability of the RO.
- Release management: manage the evolution of ROs. It enables the creation and maintenance of RO releases, enabling their reference for sharing and citation. In creating these releases, curation information needs to be generated and maintained, usually signifying a point in the RO lifecycle that has reached some degree of maturity or once the RO lifecycle reaches its end. This functionality also manages versioning of releases, including the identification and maintenance of their dependencies, as well as keeping their change logs to support undo and redo operations (e.g. rollback to a previous version).
- Monitoring: periodically verify the ROs to identify quality degradation below some predefined threshold, and assess risks of data loss in the context of long-term preservation. It may reuse the quality evaluation and execution functions to verify if, for instance, some resources aggregated by the RO become unavailable, making the RO incomplete or impossible to run. It also verifies integrity of data.
- Notification: responsible to inform potential risks and problems that were identified during the monitoring activities, e.g., if an RO becomes incomplete or if some ROs becomes corrupt.

We define interfaces to Wf4Ever functionality in terms of models, APIs and services, which constitute the structure of our architecture. A complete explanation of the relationship between these concepts is presented

in Section 4. In the rest of this section we provide an overview of the concrete elements that realize these models, APIs and services and relate them to the categorisation of functionalities described above. For a full specification of the APIs, please refer to Section 5.

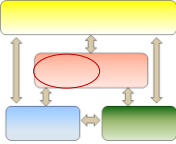
3.2 Wf4Ever Models

Wf4Ever models provide the underlying mechanisms to encode the concepts and relationships for the representation of complex workflow-centric Research Objects. These models have been selected to capture sets of concepts that can both be used in concert to implement a workflow preservation system to realize the interfaces to Wf4Ever functionalities, while also being suitable for independent use and development.

In line with Linked Data principles, these models have been implemented as lightweight and flexible ontologies using semantic web languages (RDF & OWL), which can serve as controlled vocabularies. They were developed by reusing existing standards, such as ORE [19] and AO [18], and have driven the development of other standards, such as PROV [25].

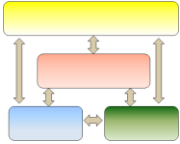
In this section we provide an overview of the concrete models and their relationship with the identified functionalities. Alignment with Wf4Ever Work Packages is also noted. Further details of requirements for the models below can be found in the Wf4Ever deliverables of the associated Work Package (D2.2v1 [26], D3.2v1 [27], D4.2v1 [28]).

| Model | Brief description | Relationship to functionalities (and associated WP) |
|--|---|--|
| Research Object (ro) | Models aggregations (ROs) and annotations. | This is the core Wf4Ever model and it is used by every functionality provided in the infrastructure. (WP2) |
| Workflow Abstract (wfdesc) | Model providing initial descriptions of workflows | Since the main goal of Wf4Ever infrastructure is the preservation of scientific workflows, this model is also utilized by every functionality. (WP2) |
| Workflow Provenance (wfprov) | A provenance convergence layer through which other models are associated. | This model is principally employed by data management & analysis functionalities and lifecycle functionalities. (WP4) |
| Research Object Evolution Model (roevo) | Models the lifecycle of Research Objects and associated changes in resources. | This model is mainly used within lifecycle functionalities. (WP3) |
| Research Object Collaboration Model (rocollab) | Models interactions and other forms of collaboration between scientists | This model is mainly used within some data management & analysis functionalities. (WP3) |

| | | | |
|--|---|--|---|
| Minimum Information Model Vocabulary (minim) | Models required information items being present in (or reported by) an RO [mim] | This model underpinns the checklist evaluation functionality of the data management & analysis layer (WP4) |  |
|--|---|--|---|

3.3 Wf4Ever APIs

3.3.1 RO API

| | |
|---|---|
| <i>Name:</i> RO API | <i>Architectural Context:</i>  |
| <i>Function:</i> Storage | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+API+6 | |

This API, formerly known as RO Storage & Retrieval API, defines how research objects can be created, accessed and maintained on the Web through a RESTful web service. In particular this API allows storing and retrieving research objects and their aggregated resources, as well as annotating them.

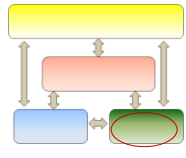
The API is heavily based on the Research Object model and, therefore, it uses the Open Annotation (AO) [18] and the Object Reuse and Exchange (ORE) [19] models. Research objects can be retrieved in a number of formats – as RDF metadata, a ZIP archive or an HTML page. Resources can be aggregated and de-aggregated by means of proxies, which simplifies handling resources that are stored outside of the service’s domain. Resources can be annotated using RDF graphs. The mechanism of using proxies and annotations as gateways for aggregated resources and annotation bodies has been based on the AtomPub specification [20].

API Operations:

- Get a list of research objects
- Create/retrieve/delete a research object
- Aggregate/de-aggregate a resource within a research object
- Update/retrieve an aggregated resource
- Annotate a resource
- Update/retrieve/delete an annotation
- Create/retrieve/delete a folder
- Add/delete a resource to/from a folder

A more detailed description of the operations and additional considerations of the API can be found in 5.1.

3.3.2 RO EVO API

| | |
|---|--|
| <i>Name:</i> RO EVO API | <i>Architectural</i> <i>Context:</i>  |
| <i>Function:</i> Lifecycle | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+evolution+API | |

The goals of this API are to enable transformation of the research objects based on their lifecycle and to facilitate retrieving the evolution history of the research objects.

The lifecycle transformation of research objects is achieved by providing a service that performs a copy of a research object and saves the copy as a live, snapshot or archived research object. The client needs to make two requests to complete the copying:

- Copy a research object. The new research object is in a transient state and is available only to the creator. It can still be modified before reaching the target evolution state.
- Finalize the state transformation. The service validates that the transient research object meets the requirements for being in the target evolution state and performs the state transition. For evolution states such as a snapshot or archived research object, that means that the research object becomes immutable.

The first and second requests can be combined into one if the client does not intend to modify the copy before finalizing the transformation.

The second API functionality is a facility for retrieving the evolution provenance. The client may send a request with a research object URI as a query parameter and the service will return an RDF graph with selected information about the research object in question, its copies and the research object that it itself was copied from. Note that typically this information can equally be achieved with a series of SPARQL queries to the RODL service hosting the research object.

The service is based on the research object evolution model.

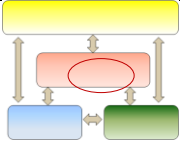
API Operations:

- Create a copy job.
- Create a finalization job.
- Check the status of a job.
- Get the evolution history of a research object.

A detailed description of the operations and additional considerations of the API can be found in 5.2.

3.3.3 Checklist Evaluation API

| | |
|---------------------------------------|----------------------|
| <i>Name:</i> Checklist Evaluation API | <i>Architectural</i> |
|---------------------------------------|----------------------|

| | |
|---|---|
| <i>Function:</i> Data Management & Analysis | <i>Context:</i>  |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+checklist+evaluation+API | |

The Checklist Evaluation API is intended to provide access to the minim-based evaluation of Research Objects, used to test for completeness, executability, repeatability and other desired features.

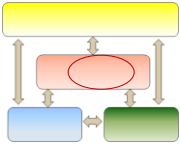
The API is intended to provide remote access to the above functionality using simple HTTP requests. Research Objects and other data are provided as web resources, and indicated in the API using their URIs.

API operations:

- Get an evaluation for an RO, a minim model and a target defined in this model.

A detailed description of the operations and additional considerations of the API can be found in 5.3.

3.3.4 Stability Evaluation API

| | |
|---|--|
| <i>Name:</i> Stability Evaluation API | <i>Architectural Context:</i>  |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Stability+Evaluation+API | |

The stability measurement is based on the evaluation of the RO along time. The idea is to capture concrete values provided by the evaluation of the RO in different moments of its evolution or, in other words, by subjecting the snapshots of a RO to the checklist evaluation (that may vary its purpose).

This API provides a global evaluation of the stability of the RO that is evaluated and a list of measurements that correspond to the snapshots of the RO. The service uses as inputs a reference to the RO that is to be evaluated, the minim model file and the target of the evaluation, similarly to what is required for the checklist evaluation service.

API operations:

- Get a stability evaluation for an RO, a minim model and a target defined in this model.

A detailed description of the operations and additional considerations of the API can be found in 5.4.

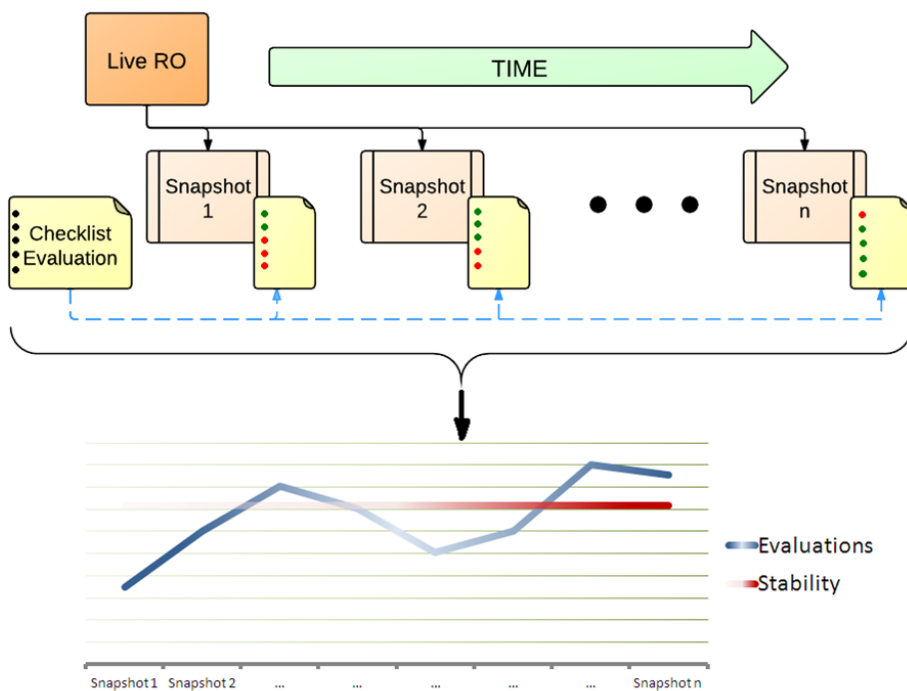
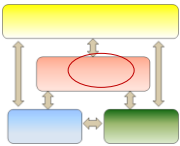


Figure 4 The Stability Evaluation API

3.3.5 Recommendation API

| | |
|---|--|
| <i>Name:</i> Recommendation API | <i>Architectural Context:</i>  |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Recommender+Service#RecommenderService-Interface | |

The Recommender Service provides a set of recommendations of scientific resources such as myExperiment files and workflows and research papers. The recipients of such recommendations must be myExperiment users, since the data used to create them is based on user's myExperiment profile and uploaded data.

API operations:

- Get a recommendation set for a user.
- Get a contextualized recommendation for a user.

A detailed description of the operations and additional considerations of the API can be found in 5.5.

3.3.6 Workflow Abstraction API

| | | |
|---|-------------------------------|--|
| <i>Name:</i> Workflow Abstraction API | <i>Architectural Context:</i> | |
| <i>Function:</i> Data Management & Analysis | | |
| <i>Level:</i> Data Management & Analysis | | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Workflow+Abstraction+API | | |

The main goal of the overall workflow abstraction work is to provide a way to scientist for searching workflows by its functionality, properties, or other conceptualization allowing their easy accessibility.

In order to make the workflow abstraction available to the users an API has been created which includes a set of services that allow searching and recommending workflow processes.

API operations:

- Find workflows that contain a provided set of workflow processes.
- Get a recommended workflow process following a provided sequence of workflow processes

A detailed description of the operations and additional considerations of the API can be found in 5.6.

3.3.7 Workflow Runner API

| | | |
|---|-------------------------------|--|
| <i>Name:</i> Workflow Runner API | <i>Architectural Context:</i> | |
| <i>Function:</i> Lifecycle | | |
| <i>Level:</i> Foundational | | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Workflow+Runner+API | | |

Research Objects, for the purpose of Wf4Ever, will generally contain workflows. In order to assess if a workflow is functional, it is generally useful to be able to (re)-execute a workflow.

Different workflow systems have different ways of running a workflow. For instance, Taverna has the Taverna Server, while Wings has a portal and a Pegasus/Condor engine in the backend. This API intends to provide a common lightweight interface within Wf4Ever for features such as "Run this workflow please" and "Show me the data from that workflow run".

At its heart, this API mirrors the RO API, but the ROs exposed by this service each represent a particular workflow run, structured to show inputs, outputs, console logs, provenance and annotations containing wfprov and wfdesc mappings. Thus it intends to be possible to use existing RO compatible tools with this service, for instance adding from the RO command line tool, browsing with the RO Portal or transforming to wfdesc using the Workflow-RO Transformation service.

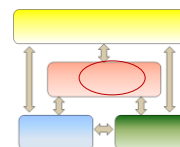
The operations specified in the API include:

- Finding default workspace to redirect to the RO workspace of the default server
- Retrieving runs in workspace
- Submitting a new run to workspace
- Retrieving a run
- Retrieving the workflow status
- Changing the workflow status to initiate running of the workflow
- Retrieving the outputs when the workflow run has finished

A detailed description of the operations and additional considerations of the API can be found in 5.7.

3.3.8 Workflow-RO Transformation API

| | |
|---|-------------------------------|
| <i>Name:</i> Workflow - RO Transformation API | <i>Architectural Context:</i> |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Wf-RO+transformation+service+API | |



The Workflow Transformation API exposes a service that transforms workflows into research objects. Workflows are often complex data structures that embed data such as their sub-resources, annotations and provenance. The service described by this API creates a research object that exposes these data according to the RO model.

The service API allows to create a transformation job which the client can subsequently monitor. The output of the transformation job contains a list of resources aggregated in the research object that have been created based on the workflow.

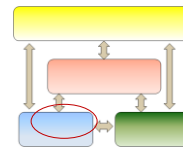
API operations:

- Create a transformation job.
- Get a status of a transformation job.
- Cancel a transformation job.

A detailed description of the operations and additional considerations of the API can be found in 5.8.

3.3.9 User Management API

| | |
|---|-------------------------------|
| <i>Name:</i> User Management API | <i>Architectural Context:</i> |
| <i>Function:</i> Storage | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/User+Management+2 | |



The user model in Wf4Ever should be as minimal as possible, so that it serves its basic purposes, can easily be mapped or reused in other contexts, and can be extended if necessary. The only absolute, project-wide requirement is that the users are identified by URIs. Whether the URIs can be dereferenced and what is returned when dereferenced depends on each application.

The following additional assumptions are made:

- The service implementing this API stores minimal information about users: their URI, a human friendly name (optional) and alternative URIs (optional). Alternative URIs may help in reusing user services in other systems (i.e. myExperiment) and may prevent users from creating duplicate accounts (i.e. two accounts with the same OpenID URI).
- There is no constraint about the user authentication method. It can be an OpenID, username/password or another mechanism. In particular, it may be possible to use more than one mechanism for one user account.

API Operations:

- Create/retrieve/update/delete a user.
- Create/retrieve/delete a client application.
- Create/delete an access token.
- Get the owner of an access token.

A detailed description of the operations and additional considerations of the API can be found in 5.9.

3.4 Wf4Ever Services and Clients

Services implement Wf4Ever functionality that is provided to multiple clients via the APIs previously described. These have been implemented as part of the Wf4Ever Toolkit, which is the reference implementation of the Wf4Ever architecture. It comprises a set of services implementing storage, lifecycle, and data management & analysis functionalities, where these functionalities are exposed via the APIs described in the previous section, alongside client applications providing access and usage functionalities. Figure 4 depicts the components within the current version of the toolkit, illustrating how it realizes Wf4Ever architecture in terms of the functionalities implemented. In the following section we describe more in detail the interrelations between these components.

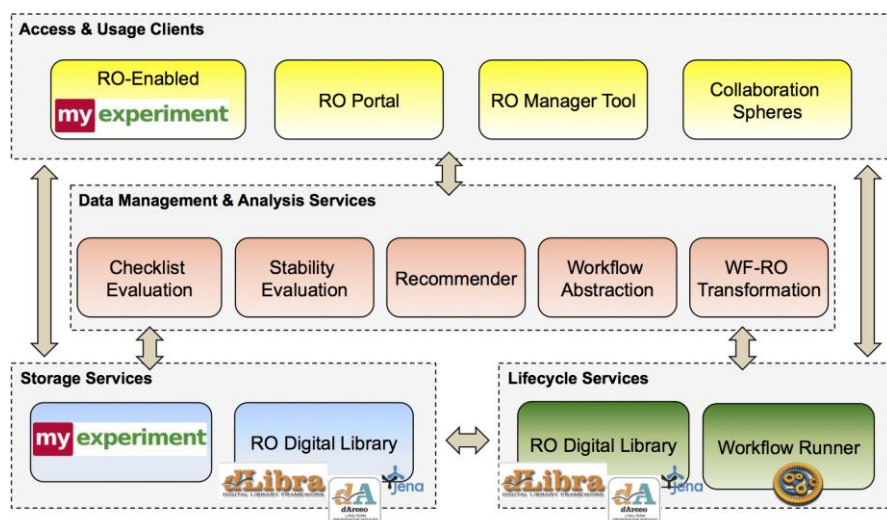


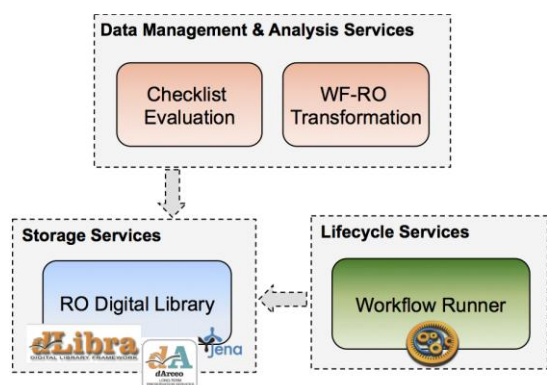
Figure 4. Wf4Ever Toolkit

3.4.1 RO Digital Library (RODL)

Software system that collects, manages and preserves aggregations of scientific workflows and related objects and annotations, packed into Research Objects.

It implements the following APIs:

- RO API (see 3.3.1)
- RO EVO API (see 3.3.2)
- User Management API¹ (see 3.3.9)
- Access Control API (work in progress, see **Error! Reference source not found.**)
- OpenSearch API [21]
- SPARQL Endpoint [22]



RODL is available at <http://sandbox.wf4ever-project.org/rodl>.

¹ RODL implements the authorization code grant with support for native applications. It uses the Bearer authorization schema.

3.4.2 myExperiment

The myExperiment social website for workflow sharing (led by OXF and UNIMAN) has been designed following the Web 2.0 principles – its form is that of a classic Web 2.0 application. While multiple instances of the myExperiment software can, and have, been deployed the publicly available primary instance of myExperiment (known internally as “the mothership”) contains the majority of the public content and social network.

Internally, the database server, search server and external workflow enactors are all separate systems to which the main application connects. This is a Ruby-on-Rails application, and the interfaces are accessed via a web server that handles load balancing over a cluster of mongrel application servers.

In the context of Wf4Ever, the myExperiment mothership provides an important repository of existing content – specifically of both workflows and packs, which have been used as benchmarks in the development of Wf4Ever models, APIs and services, and as content which has been transitioned to use the RO model and Wf4Ever services.

Workflows from myExperiment can be used as input for services such as the Workflow-RO Transformation Service and the Workflow Runner Service. Moreover, the relations between users and workflows and packs, exposed by the myExperiment SPARQL endpoint, are used by Recommender Service and the Collaboration Spheres client.

myExperiment is available at <http://www.myexperiment.org>.

3.4.3 Workflow Runner Service

Workflow execution service, key for workflow decay analysis.

It implements the Workflow Runner API (see 3.3.7).



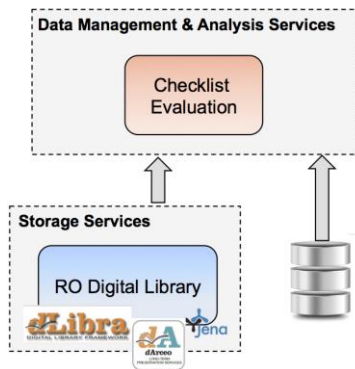
Workflow runner can execute workflows in myExperiment repository and RODL and exposes runs as ROs using the Workflow Runner API that mirrors RO API in the construction of ROs.

Workflow Runner Service is available at <http://sandbox.wf4ever-project.org/runner/>.

3.4.4 Checklist Evaluation Service

Service for testing completeness, executability, repeatability and other desired quality features of a Research Object.

It implements the Checklist Evaluation API (see 3.3.2).



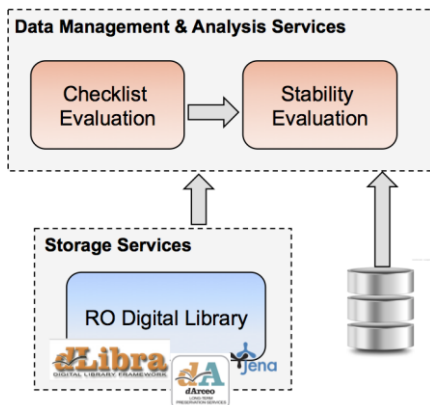
The checklist service can evaluate Research Objects located on the local file system or in a remote repository, such as RODL.

The Checklist Evaluation Service is available at <http://sandbox.wf4ever-project.org/roevaluate/>.

3.4.5 Stability Evaluation Service

Service for testing the ability of a Research Object to achieve its original purpose after being subject of changes on its resources.

It implements the Stability Evaluation API (see 3.3.4).



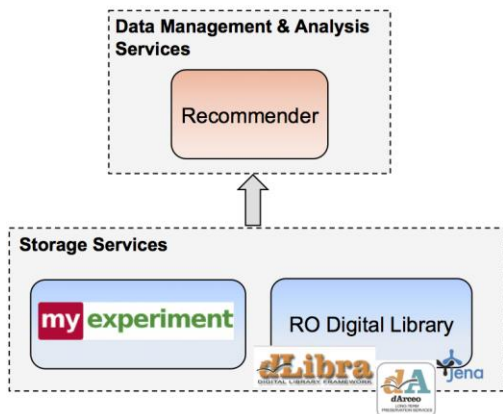
The stability service uses the checklist service to check the status of the Research Object at different moments in time, and consequently it can analyze ROs located on the local file system or in a remote repository, such as RODL.

The Stability Evaluation Service is available at <http://sandbox.wf4ever-project.org/decayMonitoring>.

3.4.6 Recommender Service

Service that provides recommendations to a particular user of relevant users, Research Objects and their aggregated resources (e.g., scientific workflows and datasets).

It implements the Recommendation API (see 3.3.5).



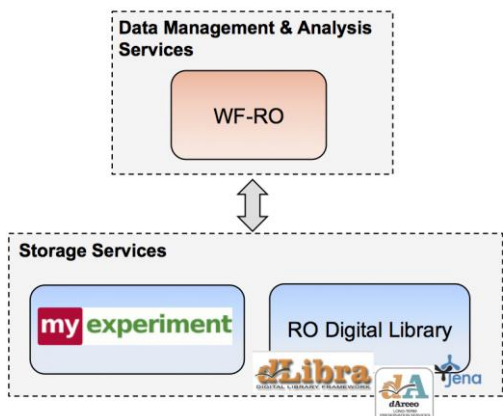
The Recommender Service can use myExperiment or RODL as the source of information to create recommendations.

The Recommender Service is available at <http://sandbox.wf4ever-project.org/epnoiServer/rest/recommender/>.

3.4.7 Workflow-RO Transformation Service

A service that converts workflows into Research Objects.

It implements the Workflow-RO Transformation API (see 3.3.8).



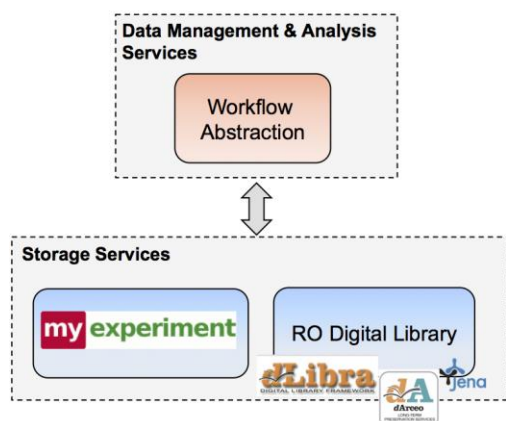
The WF-RO service takes as input a workflow (e.g., in myExperiment) and stores the resulting Research Object in RODL.

The Workflow-RO Transformation Service is available at <http://sandbox.wf4ever-project.org/wf-ro>.

3.4.8 Workflow Abstraction Service

It is a service that returns URIs of workflows that contain the sequence of executed processes that have been introduced as input parameter, or the most frequently used processes after the sequence of executed processes that has been introduced as input parameter.

It implements the Workflow Abstraction API (see 3.3.6).

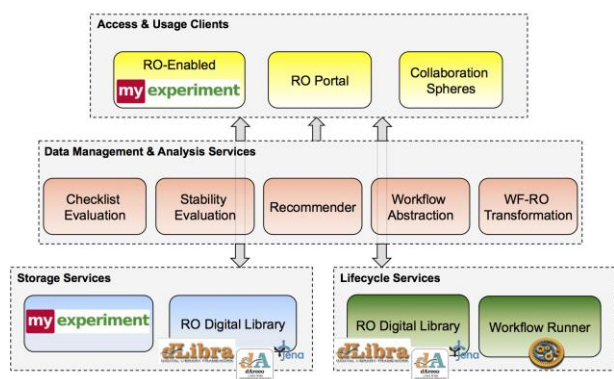


The service uses a provenance corpus (through a SPARQL endpoint) to create the indexation. This endpoint may be from RODL or myExperiment.

The Workflow Abstraction Service is available at <http://sandbox.wf4ever-project.org/wfabstraction>.

3.4.9 RO-enabled myExperiment

In addition to the myExperiment “mothership” (3.4.2, above) which has been used to bootstrap Wf4Ever content, an RO-enabled version of the myExperiment software has been developed that directly accesses Wf4Ever services to populate its data using the RO model and RO API. It is a collaborative environment where scientists can safely publish their workflows and in silico experiments packed as Research Objects, share them with groups and find those of others. It is the main Wf4Ever client application, and interacts with other Wf4Ever services.



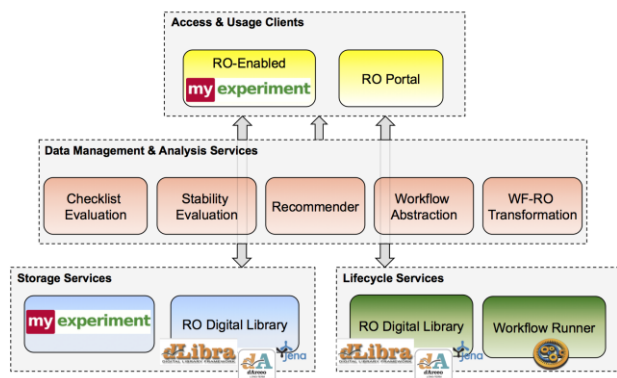
myExperiment interacts with all other Wf4Ever services (work-in-progress) and with RO Portal and the Collaboration Spheres.

The RO-enabled myExperiment is available at <http://alpha.myexperiment.org>.

3.4.10 RO Portal

It is a web client application that enables access and use of research objects and aggregated resources. It interacts with other Wf4Ever services and is an alternative client application, used mainly for prototyping.



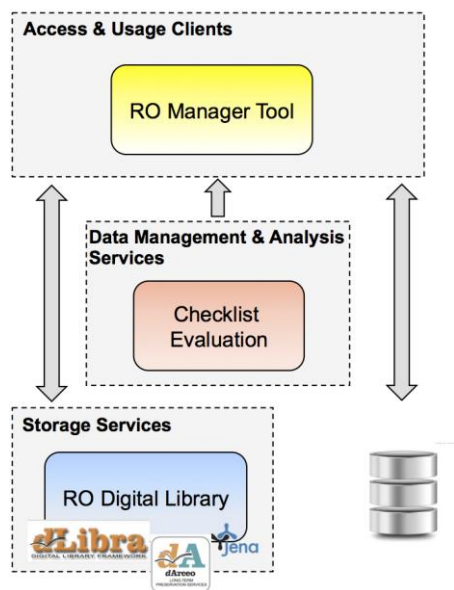


RO Portal interacts with all other Wf4Ever services (work-in-progress) and with the RO-enabled myExperiment.

The RO Portal is available at <http://sandbox.wf4ever-project.org/portal>.

3.4.11 RO Manager

A Command Line Interface (CLI) Tool that enables access and use of research objects and aggregated resources stored on the local file system, and the synchronization with remote repositories, e.g., RODL.

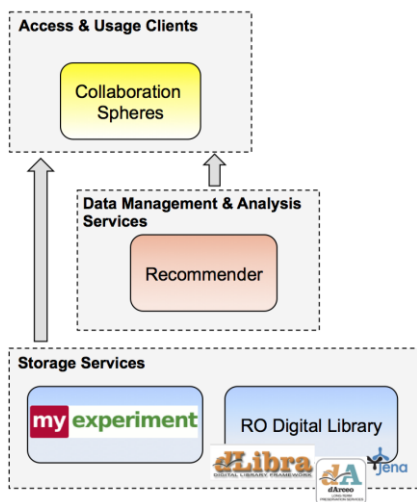


RO Manager Tool mainly uses resources on the local file system. However, it can also communicate with the RODL service (push, pull). The checklist evaluation service is currently an extension of this tool.

The RO Manager can be downloaded from <https://github.com/wf4ever/ro-manager/archive/master.zip>.

3.4.12 Collaboration Spheres

A web user interface for the visualization of correlation between similar objects (e.g., users, Research Objects) based on collaborative filtering and versatile keyword content-based recommendations.



The collaboration spheres client uses the recommender service and imports data from the storage services, i.e., myExperiment, and RODL in the future.

The Collaboration Spheres are available at <http://sandbox.wf4ever-project.org/CollaborationSpheres/circles.html>.

4 Example architecture orchestration

To illustrate how the Wf4Ever Toolkit components can be employed we describe a use case scenario, consisting of a number of smaller scenarios that although may depend on one another, they can in general be performed in any order. These scenarios are depicted in the deployment diagram in Figure 5.

- 1. Create a new Research Object from scratch.** For this task, the user either opens RO-enabled myExperiment portal (arrows colour A in Figure x) or RO portal (arrows F), creates an empty Research Object, which is stored in RO Digital Library, and then aggregates resources in the Research Object by uploading local files, which are also stored in RO Digital Library.
- 2. Create a new Research Object by reusing data from myExperiment repository.** On the one hand (arrows A+B), users can select existing resources from RO-enabled myExperiment, such as workflows or packs, and transform them into Research Objects compliant with the RO ontology, potentially using the WF-RO transformation service for the workflows. On the other hand (arrows B+F), users can select myExperiment content from RO portal (via the import wizard), and transform it into Research Objects, potentially using the WF-RO transformation service. The content itself is not modified and the available metadata, such as title and description, are copied as semantic annotations. Moreover, additional semantic annotations are created during the import including the creator of RO and the date.

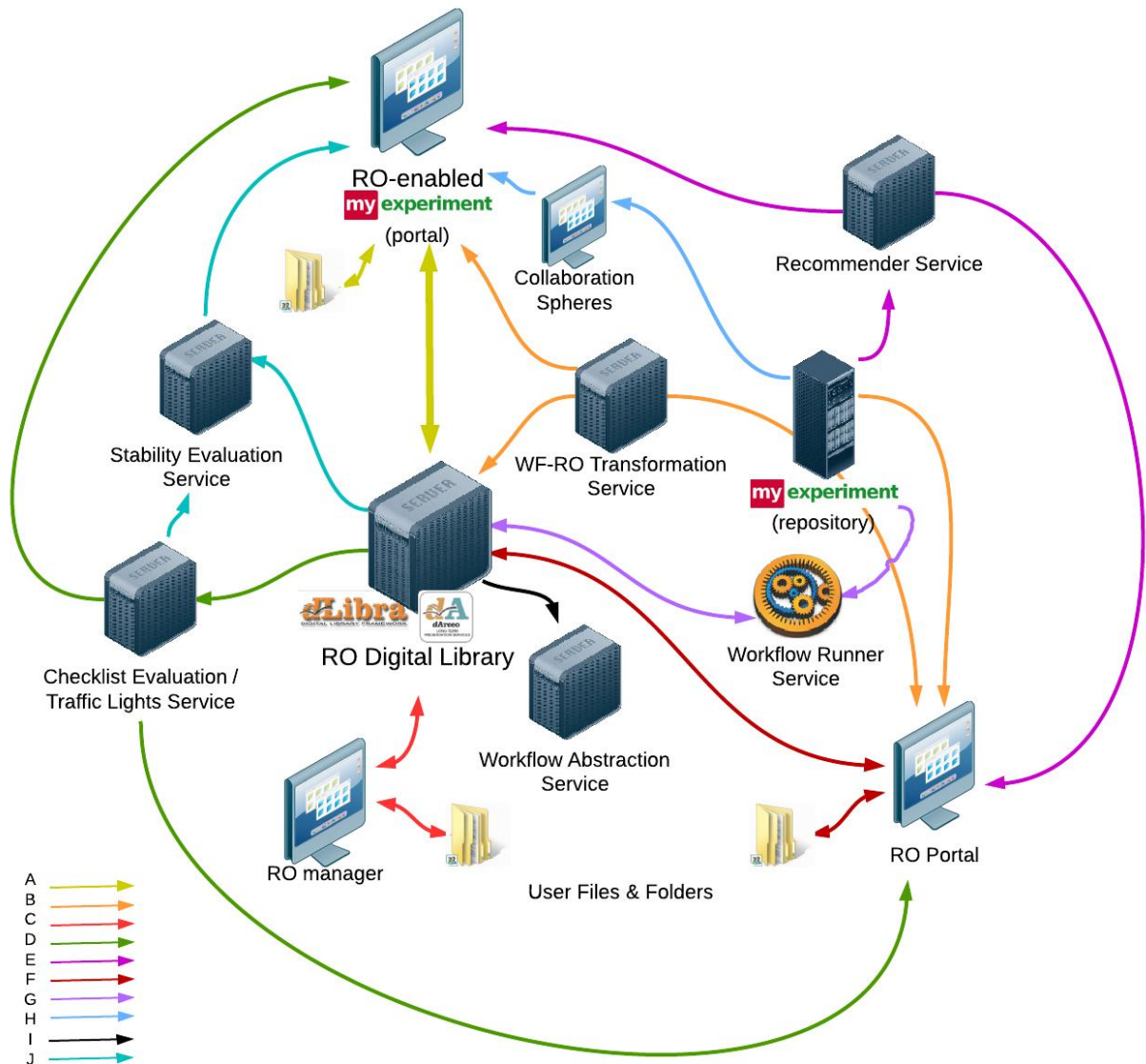


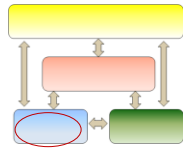
Figure 5 Deployment Diagram showing some Interactions between Wf4Ever components

3. **Discover and manage Research Objects.** RO-enabled myExperiment portal (arrows A) and RO Portal (arrows F) retrieve information from RO Digital Library to present Research Objects through a web interface. Both presents a list of Research Objects to users based on the search criteria (i.e., keywords), which can be filtered and sorted. When selecting a Research Object, its content is presented to the user graphically. Via the web interface a user may aggregate or remove resources from Research Objects and also annotate them.
4. **Create a local copy of a Research Object** (arrows C). Using the RO Manager command line tool a user may create a Research Object on a local drive, aggregate his files as resources and create annotations. The Research Object may also be synchronized with the RO Digital Library.
5. **Find recommended workflows and Research Objects** (arrows E) The user can use the RO-enabled myExperiment (or RO Portal) to call the Recommender Service and receive recommendations of myExperiment resources.

6. **Run a checklist evaluation** (arrows D). Users can run the Checklist Evaluation Service providing as input Research Objects stored in RO Digital Library. The service can be invoked from RO-enabled myExperiment and it is invoked automatically when opening a Research Object in RO Portal. Since the Checklist Evaluation Service has been developed as an extension of the RO Manager Tool, the same evaluation can be performed using the RO Manager Tool.
7. **Run a stability evaluation** (arrows J). The Stability Evaluation Service executes a series of checklist evaluations on Research Objects stored in RO Digital Library. The services can be invoked from RO-enabled myExperiment.
8. **Execute a workflow** (arrows G) Users may execute workflows from RO Digital Library or myExperiment (R) using the Workflow Runner Service. The run provenance is subsequently exposed by the service. The service will be called by the monitoring services provided by RO Digital Library as part of the preservation features.
9. **Find similar and relevant resources with a graphical interface** (arrows H) Users can visualize graphically relevant resources by using collaboration spheres, which uses in the back the recommender service. The collaboration spheres can be opened from RO-enabled myExperiment.
10. **Search workflows by their functionality, or other properties** (arrows I) Users can find workflows that contain a particular sequence of executed processes, and receive recommendation of most common used processes that follow a particular sequence. The service uses provenance traces compliant to the RO model that are available via a SPARQL endpoint, e.g., RO Digital Library.

5 API Specifications

5.1 RO API

| | |
|---|---|
| <i>Name:</i> RO API | <i>Architectural Context:</i>  |
| <i>Function:</i> Storage | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+API+6 | |

5.1.1 API usage

The clients typically start by selecting an existing Research Object or creating a new one. The Research Object redirects to the manifest, which contains links to all other resources relevant for the Research Object. By sending POST requests to the Research Object resource, the client can also create new resources.

Some requests may require authorization. Authorization and access control policy is out of scope of this document but for simplicity, the use of OAuth 2.0 with the Bearer authorization schema is assumed.

5.1.2 HTTP methods

The RO API uses four HTTP methods: GET, POST, PUT and DELETE.

GET is used to retrieve the research object and its resources. In case of the research object itself and its metadata, content negotiation is available to get a desired RDF format of the response. Also, in case of research objects, annotations, proxies and folders the client is redirected to the resource that is the representation of each resource.

POST is used to create new content: a new research object, an aggregated resource, an annotation, proxy or a new folder entry. When creating a new annotation or a proxy, it is possible to specify in the request a URI for a resource associated with it (annotation body and proxied resource, respectively), which can be added later if necessary.

PUT is used to update the content or specification of existing resources. In case the URI has been reserved by creating an annotation or a proxy, PUT can also be used for uploading the resource for the first time.

DELETE is used to delete a research object or any other resource.

5.1.3 Link relations

| Property | Description |
|--------------------------------|---|
| ore:aggregates | This relation comes from the ORE specification and is used to indicate that a resource is aggregated by the Research Object. ore:aggregates links a ro:ResearchObject with one or more ro:Resource and ro:AggregatedAnnotation. It does not link it with ore:Proxies and ao:body. |
| ro:annotatesAggregatedResource | This relation comes from the Research Object vocabulary specification |

| | |
|--------------|--|
| | and is used to indicate that a resource that is ore:aggregated by the ro:ResearchObject is annotated by another resource, called the annotation body |
| ao:body | This relation comes from the AO specification and it links a resource that is an rdfs:Graph (the annotation body) with an ro:AggregatedAnnotation. By combining the ro:annotatesAggregatedResource and ao:body, annotation bodies are linked with aggregated resources that they annotate. |
| ore:proxyFor | The ORE relation links proxies and the resources they represent. It is used for ro:Resources aggregated in ro:ResearchObjects as well as for resources aggregated in ro:Folders. |

5.1.4 Resources and formats

A research object aggregates resources and annotations. Each aggregated resource is associated with a proxy that identifies that resource in the context of that research object.

| Resource | Description |
|-----------------|--|
| Research Object | A Research Object can have 3 different formats, as defined in RO dereferencing page ² . These are: <ul style="list-style-type: none"> • a ZIP file with all internal resources and metadata. • an RDF file, i.e. its manifest. • an HTML page. |
| Resource | A resource can be any file including even a proxy or an annotation. If the resource is an RDF graph that stores the RO metadata (i.e. the manifest or an annotation body), its content can be retrieved using content negotiation. |
| Proxy | An entity of MIME media type application/vnd.wf4ever.proxy, which contains an RDF/XML expression that describes one or more ore:Proxy resource. Other information present in the RDF may be ignored. In principle, any RDF data can be used to describe a proxy, but the application/vnd.wf4ever.proxy is currently the only mechanism used for signaling that it is intended to be interpreted as a proxy description. The ore:Proxy resource type is part of the ORE aggregation structure, and as such is incorporated by reference in the structure of a Research Object. It is described by the ORE abstract model specification (http://www.openarchives.org/ore/1.0/datamodel). |
| Annotation | An entity of MIME media type application/vnd.wf4ever.annotation, which contains an RDF/XML expression that describes one or more ro:AggregatedAnnotation resource. Other information present in the RDF may be ignored. In principle, any RDF data can be used to describe a annotation, but the application/vnd.wf4ever.annotation is currently the only mechanism used for signaling that it is intended to be interpreted as an annotation description. The annotation gets stored in the manifest as ro:AggregatedAnnotation which is a specialization of the AO annotation ontology for the structure of a Research Object. ao:Annotation is described by the AO model specification |

² <http://wf4ever-project.org/wiki/display/docs/RO+dereferencing>

| | |
|--------------|---|
| | http://code.google.com/p/annotation-ontology/wiki/UnderstandingAO). |
| Folder | An entity of MIME media type application/vnd.wf4ever.folder, which contains an RDF/XML expression that describes one or more ro:Folder resources. Other information present in the RDF may be ignored. The folder description must contain an aggregation of resources. It may also contain ro:FolderEntry entities for some of the aggregated resources for the purpose of specifying the name of the resource in the folder described. |
| Folder Entry | An entity of MIME media type application/vnd.wf4ever.folderentry, which contains an RDF/XML expression that describes one or more ro:FolderEntry resources. Other information present in the RDF may be ignored. The folder entry description must contain a ore:proxyFor reference to an aggregated resources. It may contain a reference to the aggregation itself (ore:proxyIn) and the name of the resource in the folder described (ro:entryName). |

5.1.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.1.5.1 Creating a new Research Object

The Research Object id can be any string. If this id is already in use, 409 Conflict will be returned.

```
C: POST /ROs/ HTTP/1.1
C: Host: example.com
C: Content-Type: text/plain
C: Content-Length: ...
C: Accept: text/turtle
C: Slug: ro id
C: Authorization: Bearer h480djs93hd8

S: HTTP/1.1 201 Created
S: Location: http://example.com/ROs/ro%20id/
S: Content-Type: text/turtle
S: Content-Length: ...
S:
S: (the initial manifest)
```

It is possible to create an RO by uploading an existing RO in a ZIP archive. When RO service receives the zip with the RO, it should:

- Look for the .ro/manifest.rdf. If not found, 400 Bad Request is returned.
- Create the RO.
- Scan for each resource aggregated in the RO and add it to the RO.
- If there is anything else in the ZIP archive which was not aggregated, log it but don't return error.

```
C: POST /ROs/ HTTP/1.1
C: Host: example.com
C: Content-Type: application/zip
C: Content-Length: ...
C: Accept: text/turtle
C: Slug: ro id
```



```
C: Authorization: Bearer h480djs93hd8
C:
C: (the zip with manifest and resources)
```

```
S: HTTP/1.1 201 Created
S: Location: http://example.com/ROs/ro%20id/
S: Content-Type: text/turtle
S: Content-Length: ...
S:
S: (the initial manifest)
```

Or point to a ZIP archive existing on the web.

```
C: POST /ROs/ HTTP/1.1
C: Host: example.com
C: Content-Type: application/zip
C: Content-Length: ...
C: Accept: text/turtle
C: Slug: ro id
C: Link: <http://external.example.com/ro.zip>; rel="http://purl.org/wf4ever/ro/zippedRO"
C: Authorization: Bearer h480djs93hd8
```

```
S: HTTP/1.1 201 Created
S: Location: http://example.com/ROs/ro%20id/
S: Content-Type: text/turtle
S: Content-Length: ...
S:
S: (the initial manifest)
```

5.1.5.2 Aggregating a resource

Each resource aggregated in the research object has an associated proxy. Proxies are references to the resource in the context of the research object and in the case of external resources, their representations in the RO API. The structure of a proxy is described by ORE.

The API design here uses similar patterns to AtomPub, with an RO being analogous to an Atom feed, a proxy for an Atom item, and resource content for an Atom media resource. The Slug: header field is defined by AtomPub (<http://tools.ietf.org/html/rfc5023>).

Aggregating resources involves 2 steps, though in most cases one of them can be omitted:

- Creating a proxy for a resource.
- Uploading a resource.

The first step is recognized by making a request with MIME type `application/vnd.wf4ever.proxy` and using a resource URI not pointed to by an existing proxy. When a proxy is created, the proxy target is automatically aggregated in the research object.

The resource URI is indicated either using the Slug: header if a new internal resource is being aggregated (in which it is a relative URI reference), or in the request body otherwise. If the resource URI is not indicated in

the Slug: header nor in the request body, the server should generate any URI it sees fit, i.e. using a random UUID. The server will return 409 Conflict if the resulting URI has already been aggregated in this research object. Client applications should not, however, make assumptions about the URI of the resulting resource but should, instead, use the ore:proxyFor link value returned by the service.

Resources are aggregated in the following ways:

1. External resources: only create a proxy, pointing to the external resource.
2. Internal resources: only upload a resource. The service will create the proxy automatically and will return the URI of the proxy rather than the URI of the resource itself.
3. Internal resources, the long way: create a proxy and then upload the resource. This is the only method to add a proxy as an aggregated resource (a rare case).

The example below demonstrates aggregating an external resource. The URI of the external resource is given in the supplied proxy description. The content type application/vnd.wf4ever.proxy signals to the RO service that a new proxy is being created.

A application/vnd.wf4ever.proxy entity is an RDF/XML expression that describes just one ore:Proxy resource. Other information present in the RDF is ignored. Behaviour is undefined if there is not exactly one ore:Proxy resource described, and a service MAY reject such a request as ill-formed. (Future proposals may accept any format of RDF, and use a separate mechanism to signal that it is intended to be interpreted as a proxy description.)

```
C: POST /ROs/ro1/ HTTP/1.1
C: Host: example.com
C: Content-Type: application/vnd.wf4ever.proxy
C: Content-Length: ...
C: Authorization: Bearer h480djs93hd8
C:
C: <rdf:RDF
C:   xmlns:ore="http://www.openarchives.org/ore/terms/"
C:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
C:   <ore:Proxy>
C:     <ore:proxyFor rdf:resource="http://external.example.com/resource.uri" />
C:   </ore:Proxy>
C: </rdf:RDF>

S: HTTP/1.1 201 Created
S: Location: https://sandbox/ro1/ROs/ro1/.ro/proxies/9a3fdd90-becf-11e1-afa7-0800200c9a34
S: Link: <http://external.example.com/resource.uri>;
rel="http://www.openarchives.org/ore/terms/proxyFor"
S: Content-Type: application/rdf+xml
S: Content-Length: ...
S:
S: <rdf:RDF
S:   xmlns:ore="http://www.openarchives.org/ore/terms/"
S:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
S:   <rdf:Description rdf:about="https://sandbox/ro1/ROs/ro1/.ro/proxies/9a3fdd90-becf-11e1-afa7-0800200c9a34">
```

```
S: <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Proxy"/>
S: <ore:proxyIn rdf:resource="https://sandbox/rodl/ROs/ro1/" />
S: <ore:proxyFor rdf:resource="http://external.example.com/resource.uri" />
S: </rdf:Description>
S: </rdf:RDF>
```

The example below demonstrates adding a proxy for an internal resource, and then putting the new resource content at the server-indicated URI. The RO-internal name for the new resource is indicated using a Slug: header (following AtomPub). The supplied entity body is a Proxy description from which the proxyFor property is omitted.

```
C: POST /ROs/ro_id/ HTTP/1.1
C: Host: example.com
C: Content-Type: application/vnd.wf4ever.proxy
C: Slug: foo/bar.txt
C: Authorization: Bearer h480djs93hd8
C:
C: <rdf:RDF
C:   xmlns:ore="http://www.openarchives.org/ore/terms/"
C:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
C:   <ore:Proxy>
C:   </ore:Proxy>
C: </rdf:RDF>

S: HTTP/1.1 201 Created
S: Location: http://example.org/ROs/ro_id/.ro/proxies/d953d020-becc-11e1-afa7-0800200c9a66
S: Link: <http://example.com/ROs/ro_id/foo/bar.txt>;
rel="http://www.openarchives.org/ore/terms/proxyFor"
S: Content-Type: application/rdf+xml
S: Content-Length: ...
S:
S: <rdf:RDF
S:   xmlns:ore="http://www.openarchives.org/ore/terms/"
S:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
S:   <rdf:Description rdf:about="http://example.org/ROs/ro_id/.ro/proxies/d953d020-becc-11e1-afa7-
0800200c9a66">
S:     <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Proxy"/>
S:     <ore:proxyIn rdf:resource="http://example.org/ROs/ro_id/" />
S:     <ore:proxyFor rdf:resource="http://example.org/ROs/ro_id/foo/bar.txt" />
S:   </rdf:Description>
S: </rdf:RDF>

C: GET /ROs/ro_id/foo/bar.txt HTTP/1.1
C: Host: example.com

S: HTTP/1.1 404 Not Found

C: PUT /ROs/ro_id/foo/bar.txt HTTP/1.1
C: Host: example.com
C: Content-Type: text/plain
C:
```

C: This is the important note

S: HTTP/1.1 201 Created

S: Location: http://example.com/ROs/ro_id/foo/bar.txt

S: Content-Type: application/rdf+xml

S: Content-Length: ...

S:

S: <rdf:RDF

S: xmlns:ore="http://www.openarchives.org/ore/terms/"

S: xmlns:dct="http://purl.org/dc/terms/"

S: xmlns:ro="http://purl.org/wf4ever/ro#"

S: xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >

S: <rdf:Description rdf:about="http://example.com/ROs/ro_id/foo/bar.txt">

S: <dct:creator rdf:resource="http://test2.myopenid.com"/>

S: <dct:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2011-12-02T15:02:11Z</dct:created>

S: <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/AggregatedResource"/>

S: <rdf:type rdf:resource="http://purl.org/wf4ever/ro#Resource"/>

S: </rdf:Description>

S: </rdf:RDF>

The example below demonstrates the short-cut mechanism aggregating an internal resource by simply posting its content to the RO. The Slug: header is used to indicate the desired path within the RO. The server responds with the proxy description.

C: POST /ROs/ro_id/ HTTP/1.1

C: Host: example.com

C: Content-Type: text/plain

C: Content-Length: ...

C: Slug: foo/bar.txt

C: Authorization: Bearer h480djs93hd8

C:

C: Lorem ipsum

S: HTTP/1.1 201 Created

S: Location: http://example.org/ROs/ro_id/.ro/proxies/d953d020-becc-11e1-afa7-0800200c9a66

S: Link: <http://example.com/ROs/ro_id/foo/bar.txt>;

rel="http://www.openarchives.org/ore/terms/proxyFor"

S: Content-Type: application/rdf+xml

S: Content-Length: ...

S:

S: <rdf:RDF

S: xmlns:ore="http://www.openarchives.org/ore/terms/"

S: xmlns:dct="http://purl.org/dc/terms/"

S: xmlns:ro="http://purl.org/wf4ever/ro#"

S: xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >

S: <rdf:Description rdf:about="http://example.org/ROs/ro_id/.ro/proxies/d953d020-becc-11e1-afa7-0800200c9a66">

S: <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Proxy"/>

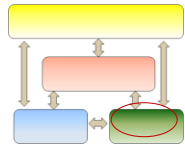
S: <ore:proxyIn rdf:resource="http://example.org/ROs/ro_id"/>

S: <ore:proxyFor rdf:resource="http://example.org/ROs/ro_id/foo/bar.txt" />


```

S: </rdf:Description>
S: <rdf:Description rdf:about="http://example.com/ROs/ro_id/foo/bar.txt">
S:   < dct:creator rdf:resource="http://test2.myopenid.com"/>
S:   < dct:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2011-12-
02T15:02:11Z</dct:created>
S:   <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/AggregatedResource"/>
S:   <rdf:type rdf:resource="http://purl.org/wf4ever/ro#Resource"/>
S: </rdf:Description>
S: </rdf:RDF>
    
```

5.2 RO EVO API

| | |
|---|--|
| <i>Name:</i> RO EVO API | <i>Architectural Context:</i>  |
| <i>Function:</i> Lifecycle | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+evolution+API | |

5.2.1 API usage

If a client wants to create a copy of an RO (i.e. a snapshot), it must perform the following operations:

- Retrieve the evolution service document and find the links to copy and finalize resources
- Create a new copy job and wait until it finishes
- Modify the RO copy in any way, i.e. delete temporary or private resources.
- Create a new finalize job and wait until it finishes

To check the evolution provenance of an RO, the steps are:

1. Perform a GET or HEAD operation to the RO to find the link to evolution information resource
2. Retrieve the evolution provenance information using the link provided

5.2.2 HTTP methods

The API uses a POST method to create a job, GET to retrieve the status of a job and DELETE to cancel a running job.

GET is also used to get the evolution history of a Research Object.

5.2.3 Link relations

| Property | Description |
|--------------------|---|
| evolution:copy | Used in the service description document. It indicates a relation between a service description and the URI of the copy resource. |
| evolution:finalize | Used in the service description document. It indicates a relation between a |

| | |
|----------------|--|
| | service description and the URI of the finalize resource. |
| evolution:info | Used in the service description document. It indicates a relation between a service description and a URI template for RO evolution history using the described service. The URI template is used to construct a service result URI by: <ol style="list-style-type: none"> 1. applying the URI template expansion procedures with caller-supplied RO URI, and 2. resolving the resulting URI-reference to an absolute URI using normal URI resolution rules (e.g. typically, using the service document URI as a base URI) |

5.2.4 Resources and formats

| Resource | Description |
|-----------------------|--|
| Service description | The evolution service description is an RDF file that contains URI templates for accessing RO evolution related services. The RDF syntax used may be content negotiated. In the absence of content negotiation, RDF/XML should be returned. |
| Copy RO | <p>The commands to copy a research object use a JSON object to store the operation parameters. JSON schema:</p> <ul style="list-style-type: none"> • copyfrom: URI of the research object that will be copied. • type: Type of the new research object. Allowed values are "live", "snapshot", "archived". • deepcopy: aggregated resources that should be copied with their content. Default is "uri-prefix": ["ro_URI"], where ro_URI is the URI of the copyfrom research object, which resolves to all resources that have the RO URI as their prefix. <p>uri-list: a list of URIs of aggregated resources uri-prefix: a list of URI prefixes of aggregated resources</p> <ul style="list-style-type: none"> • finalize: True if the service should perform a finalization of the copy operation, false otherwise. The default value is false. • target: The URI of the new research object. • status: Copy job status. Allowed values are "running", "done", "failed" or "service_error". • reason: Reason for the job status. <p>The copyfrom and type MUST be provided by the client, and deepcopy and finalize MAY be provided by the client. target, status and reason are provided by the service.</p> |
| Finalize RO | <p>The commands to copy a research object use a JSON object to store the operation parameters. JSON schema:</p> <ul style="list-style-type: none"> • target: URI of the transient research object that will be finalized. • status: Finalize job status. Allowed values are "running", "done", "failed" or "service_error". • reason: Reason for the job status. <p>The target MUST be provided by the client. status and reason are provided by the service.</p> |
| Evolution information | The evolution information is an RDF graph using the RO evolution ontology. |

5.2.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.2.5.1 Creating an RO copy

This operation creates a copy of a research object. The new research object is in a transient state unless the operation is immediately finalized. Transient state means it is only available to the creator. The type of the copy has to be defined because it influences the new research object URI form.

A "Slug" HTTP header may be used to indicate the requested RO id.

Creating and finalizing a copy may be presented to the user as one operation, only the client application will use a one step or two step process.

Copyfrom URI may be a research object in the same RO service, a research object in another RO service, or even an external non-RO content. It is more an implementation detail how the copy will be made in each of these cases.

The copy operation may take a considerable amount of time because it may involve copying resource contents. For that reason it is realized in a form of a job that can be created and monitored by the client.

Initiate a copy job.

```
C: POST http://example.com/evo/copy/ HTTP/1.1
C: Content-Type: application/json
C: Authorization: Bearer 47d5423c-b507-4e1c-7
C: Slug: copy-1
C:
C: {
C:   "copyfrom": "http://example.com/ROs/ro-id/",
C:   "type": "snapshot",
C:   "finalize": false
C: }

S: HTTP/1.1 201 Created
S: Location: http://example.com/evo/copy/12345
S:
S: {
S:   "copyfrom": "http://example.com/ROs/ro-id/",
S:   "type": "snapshot",
S:   "finalize": false,
S:   "target": "http://example.com/ROs/copy-1/",
S:   "status": "running"
S: }
```

5.2.5.2 Getting a job status.

Get a copy job status (running).

```
C: GET http://example.com/evo/copy/12345 HTTP/1.1
C: Accept: application/json
```

```
S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
S:   "copyfrom": "http://example.com/ROs/ro-id/",
S:   "type": "snapshot",
S:   "finalize": false,
S:   "target": "http://example.com/ROs/copy-1/",
S:   "status": "running"
S: }
```

Get a copy job status (finished).

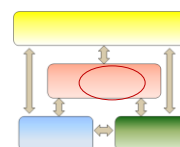
```
C: GET http://example.com/evo/copy/12345 HTTP/1.1
C: Accept: application/json
```

```
S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
S:   "copyfrom": "http://example.com/ROs/ro-id/",
S:   "type": "snapshot",
S:   "finalize": false,
S:   "target": "http://example.com/ROs/copy-1/",
S:   "status": "done"
S: }
```

The copy job may get deleted by the service after any time after completion but it is recommended to keep it for enough time for the client to check back.

5.3 Checklist Evaluation API

| | |
|---|-------------------------------|
| <i>Name:</i> Checklist Evaluation API | <i>Architectural Context:</i> |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/RO+checklist+evaluation+API | |



5.3.1 API usage

In order to perform a checklist evaluation, the following data are required:

- A publicly accessible Research Object which will be evaluated.
- A publicly accessible minimum information model resource from which the checklist definition is obtained.
- A selected checklist definition from the MINIM model, for example to test “repeatability”.
- A target resource with respect to which the evaluation is performed; the default target is the RO itself, but a component within the RO may be selected.



An example of a simple minim model can be found at <https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/simple-requirements/simple-requirements-minim.rdf>

The checklist evaluation is typically invoked in a sequence of two HTTP operations:

1. Client retrieves service document.
2. Client parses the service document, extracts the URI template for the checklist evaluation service and assembles URI for the desired evaluation result (cf. RFC6570), and issues a second HTTP GET request.

The result from the second request is the checklist evaluation result. The optional target URI parameter may be omitted if the target is the Research Object itself.

5.3.2 HTTP methods

The service description is obtained in response to an HTTP GET to a checklist evaluation service URI.

The checklist evaluation service responds to an HTTP GET with the results of a checklist evaluation, using the URI defined by expanding the template provided by the service description.

5.3.3 Link relations

| Property | Description |
|------------------------------------|---|
| evaluate:checklist | Used in the service description document. It indicates a relation between a service description and a URI template for RO evaluation results using the described service. The URI template is used to construct a service result URI by: <ol style="list-style-type: none"> 1. applying the URI template expansion procedures with caller-supplied RO URI, minim URI, purpose and target URIs, and 2. resolving the resulting URI-reference to an absolute URI using normal URI resolution rules (e.g. typically, using the service document URI as a base URI) |

5.3.4 Resources and formats

The checklist evaluation **service description** is an RDF file that contains URI templates for accessing RO related services, including checklist evaluation. The RDF syntax used may be content negotiated. In the absence of content negotiation, RDF/XML should be returned.

5.3.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.3.5.1 Retrieving the service document

Client retrieves service document:

```
C: GET /evaluate/checklist HTTP/1.1
C: Host: service.example.org
C: Accept: application/rdf+xml
```

```

S: HTTP/1.1 200 OK
S: Content-Type: application/rdf+xml
S:
S: <?xml version="1.0"?>
S: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
S:   xmlns:roe="http://purl.org/ro/service/evaluate/">
S:   <rdf:Description rdf:about="">
S:     <roe:checklist>/evaluate/checklist{?RO,minim,target,purpose}</roe:checklist>
S:   </rdf:Description>
S: </rdf:RDF>

```

Client parses the service document, extracts the URI template for the checklist evaluation service and assembles URI for the desired evaluation result (cf. RFC6570), and issues a second HTTP GET request.

5.3.5.2 Evaluating the RO

```

C: GET /evaluate/checklist?RO=http%3A%2F%2Fsandbox.example.org%2FROs%2Fmyro
    &minim=http%3A%2F%2Fanother.example.com%2Fminim%2Frepeatable.rdf
    &purpose=repeatable HTTP/1.1
C: Host: service.example.org
C: Accept: application/rdf+xml

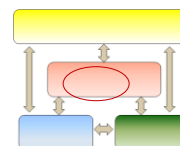
S: HTTP/1.1 200 OK
S: Content-Type: application/rdf+xml
S:
S: <?xml version="1.0"?>
S: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
S:   xmlns:...="..."
S:   >
S:   <rdf:Description rdf:about="...">
S:     (Result of checklist evaluation)
S:   </rdf:Description>
S: </rdf:RDF>

```

The result from the second request is the checklist evaluation result. The URI shown above has been split over several lines for readability - the actual HTTP request must present it without whitespace. The optional target URI parameter has been omitted in this example on the assumption that the target is the Research Object itself.

5.4 Stability Evaluation API

| | |
|---|-------------------------------|
| <i>Name:</i> Stability Evaluation API | <i>Architectural Context:</i> |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Stability+Evaluation+API | |



5.4.1 API usage

In order to perform a stability evaluation, the following data are required:

- A publicly accessible Research Object which will be evaluated.
- A publicly accessible MINIM minimum information model description containing checklist definitions.
- A selected checklist definition from the MINIM model, for example to test “repeatability”.

The stability evaluation is typically invoked in a sequence of two HTTP operations:

1. Client retrieves service document.
2. Client parses the service document, extracts the URI template for the stability evaluation service and assembles URI for the desired evaluation result (cf. RFC6570), and issues a second HTTP GET request.

The result from the second request is the stability evaluation result.

5.4.2 HTTP methods

The only HTTP method available for this API is the GET method. The stability evaluation service responds to an HTTP GET with the results of a stability evaluation, using the URI defined by expanding the template provided by the service description.

5.4.3 Link relations

| Property | Description |
|---------------------------------|---|
| <code>evaluate:stability</code> | Used in the service description document. It indicates a relation between a service description and a URI template for RO evaluation results using the described service. The URI template is used to construct a service result URI by: <ol style="list-style-type: none"> 3. applying the URI template expansion procedures with caller-supplied RO URI, minim URI, purpose and target URIs, and 4. resolving the resulting URI-reference to an absolute URI using normal URI resolution rules (e.g. typically, using the service document URI as a base URI) |

5.4.4 Resources and formats

| Resource | Description |
|-----------------------------|---|
| Service description | The stability evaluation service description is an RDF file that contains URI templates for accessing RO related services, including stability evaluation. The RDF syntax used may be content negotiated. In the absence of content negotiation, RDF/XML should be returned. |
| Stability Evaluation Result | The result of the stability evaluation contains the following fields: <ul style="list-style-type: none"> • ro: URI of the RO that has been evaluated • stabilityValue: the main result of the API, the stability value (a floating point number between 0 and 1 that reflects the standard deviation of the checklist evaluation values). |

| | |
|--|---|
| | <ul style="list-style-type: none"> • listElements: list of URIs of RO snapshots that were evaluated • uri: snapshot URI • value: result obtained by the evaluation the snapshot using the checklist evaluation service (a value between 0 and 1 that shows the percentage of requirements that have been satisfied) • date: creation date of the snapshot (DD/MM/YYYY) <p>The stability evaluation results are available in XML and JSON formats, available using content negotiation. The default format is XML.</p> |
| | |

5.4.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.4.5.1 Retrieving the service document

```
C: GET /stability/rest/getStability HTTP/1.1
C: Host: service.example.org
C: Accept: application/rdf+xml
S: HTTP/1.1 200 OK
S: Content-Type: application/rdf+xml
S:
S: <?xml version="1.0"?>
S: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
S: xmlns:roe="http://purl.org/ro/service/evaluate/stability">
S: <rdf:Description rdf:about="">
S: <roe:checklist>/stability/rest/getStability{?RO,minim,purpose}</roe:checklist>
S: </rdf:Description>
S: </rdf:RDF>
```

Client parses the service document, extracts the URI template for the checklist evaluation service and assembles URI for the desired evaluation result and issues a second HTTP GET request.

5.4.5.2 Retrieving the RO stability

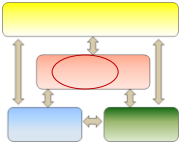
```
C: GET /stability/rest/getStability?RO=http://www.ro-examples.com/ro/fakeRO &minim=http://www.minim-examples.com/minim/Repeatability.rdf &purpose=Repeatable HTTP/1.1
C: Host: service.example.org
C: Accept: application/json
S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
S:   "ro": "http://www.ro-examples.com/ro/fakeRO",
S:   "stabilityValue": "0.8105506067403516",
S:   "listElements": [
S:     {
S:       "uri": "http://www.ro-examples.com/ro/fakeROSnapshot1",
S:       "value": "0.11102273035109067",
S:       "date": "14/06/2012"
```



```

S:   },
S:   {
S:     "uri":"http://www.ro-examples.com/ro/fakeROSsnapshot2",
S:     "value":"0.06829956253911362",
S:     "date":"17/06/2012"
S:   },
S:   {
S:     "uri":"http://www.ro-examples.com/ro/fakeROSsnapshot1",
S:     "value":"0.4898372013610631",
S:     "date":"20/06/2012"
S:   }
S: ]
S: }
    
```

5.5 Recommendation API

| | |
|---|--|
| <i>Name:</i> Recommendation API | <i>Architectural Context:</i>  |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Recommender+Service#RecommenderService-Interface | |

5.5.1 API usage

The interface is a REST API that basically can be used as follows:

```

HTTP: GET PATH/recommender/recommendations/recommendationSet/
user/{userId}/{itemType}{?max}
    
```

Where:

- <PATH> the path where the Recommender Service is deployed
- <userId> the myExperiment id of the user.
- <itemType> the class of objects that we are interested in receiving recommendations.
- <max> the maximum number of recommendations that we want to retrieve. Recommendations are ordered by their strength, so when the max parameter is specified, the more relevant recommendations up to max are retrieved.

The recommendation service is typically invoked in a sequence of four HTTP operations:

1. The client retrieves service document.
2. The client parses the service document, extracts the URI template for the recommender service and assembles URI for the desired recommendations set.
3. The client can also assemble the URI for creating the desired recommendation context for a later use of the contextualized recommender.
4. After creating the recommendation context the client can request a recommendations obtained using the provided context.

5.5.2 HTTP methods

The service description is obtained in response to an HTTP GET to a Recommender Service URI.

The Recommender Service responds to an HTTP GET with the results of a `recommendationsSet`, using the URI defined by expanding the template provided by the service description.

5.5.3 Link relations

| Property | Description |
|---|--|
| <code>recommendationsSet</code> | Used in the service description to point to the set of recommendations for the user identified as user (the integer that represents the user in <code>myExperiment</code>). Its cardinality may be restricted up to a number (max). Example value: <code>/recommender/recommendations/recommendationSet/user/{userId}{?max}</code> |
| <code>filteredRecommendationsSet</code> | Used in the service description to point to the set of recommendations for the user identified as <code>userID</code> of the item type <code>itemType</code> (i.e. workflows, files, users, packs). Its cardinality may be restricted up to a number (max). Example value: <code>/recommender/recommendations/recommendationSet/user/{user}/{type}{?max}</code> |
| <code>recommendationContext</code> | Used in the service description to point to the recommendation context must be set up in case that the user may be interested in receiving recommendations based in a group of <code>myExperiment</code> resources or keywords. The recommendation context is composed by the set of resources (0..N resources defined by the resource query parameter), the set of keywords (0..N keywords defined by the keyword query parameter), and the URI of the user that is associated with the context (user query param) Example value: <code>/recommender/contexts/recommendationContext{?user,resource, keyword}</code> |
| <code>contextualizedRecommendationsSet</code> | Used in the service description to point to the set of contextualized recommendations for the user identified as user (user query param) of items of a type (type query param), i.e. workflows, files, users, packs. Its cardinality may be restricted up to a number (max query param). Example value: <code>/recommender/recommendations/contextualizedRecommendationsSet{?user,type,max}</code> |

5.5.4 Resources and formats

| Resource | Description |
|------------------------|---|
| Recommendations Set | <p>A recommendations set represents a set of recommendations for a given user. Each recommendation is described by the following fields:</p> <ul style="list-style-type: none"> “explanation”: a user oriented description on why the recommendation is made to the user “itemType”: type of recommended item, one of the following: <code>item_type_workflow</code>, <code>item_type_file</code>, <code>item_type_pack</code>, <code>item_type_user</code>. “resource”: the URL of the recommended item “strength”: a real number that ranges from 0 to 5 with that represents the relevancy of the recommendation “title”: the title or name of the recommended item “usedTechnique”: technique used to generate this recommendation, one of the following: <code>technique_keyword_content_based</code>, <code>technique_social</code>, <code>technique_collaborative</code>, <code>technique_inferred</code>, <code>technique_group_content_based</code> |
| Recommendation Context | <p>The recommendation context resource contains the group or resources and keywords that are considered in the provisioning of contextualized recommendations for a given user. A recommendation context comprises the following fields:</p> <ul style="list-style-type: none"> “resource”: a list of resource URIs “keyword”: a list of keywords “userURI”: The user associated with the context |

5.5.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.5.5.1 Retrieving the service document

First the client retrieves service document:

```

C: GET /recommender HTTP/1.1
C: Host: service.example.org
C: Accept: application/xml

S: HTTP/1.1 200 OK
S: Content-Type: application/xml
S:
S: <recommender>
S:   <filteredRecommendationsSet>
S:     /recommender/recommendations/recommendationSet/user/{userId}/{itemType}{?max}
S:   </filteredRecommendationsSet>
S:   <recommendationsSet>
S:     /recommender/recommendations/recommendationSet/user/{userId}{?max}
S:   </recommendationsSet>
S:   <recommendationContext>
S:     /recommender/contexts/recommendationContext{?user,resource, keyword}

```

```

S: </recommendationContext>
S: <contextualizedRecommendationsSet>
S:   /recommender/recommendations/contextualizedRecommendationsSet{?user,type,max}
S: </contextualizedRecommendationsSet>
S: </recommender>

```

The client parses the service document, extracts the URI template for the recommender service and assembles URI for the desired recommendations set.

5.5.5.2 Getting the recommendation set

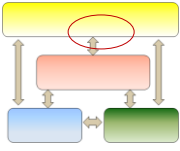
```

C: GET /recommender/recommendations/recommendationsSet/user/2 HTTP/1.1
C: Host: service.example.org
C: Accept: application/xml

S: HTTP/1.1 200 OK
S: Content-Type: application/xml
S:
S: <?xml version="1.0"?>
S: <recommendationsSet>
S:   <recommendation>
S:     <explanation>
S:       The workflow entitled Get names of proteins similar to RNA binding proteins (Simple
example SADI workflow)
S:       (URI:http://www.myexperiment.org/workflow.xml?id=2127) is recommended to you since you
used the following tags: sadl ,
S:       taverna , spreadsheet; and they partially describe its content
S:     </explanation>
S:     <itemType>
S:       item_type_workflow
S:     </itemType>
S:     <resource>
S:       http://www.myexperiment.org/workflows/2127
S:     </resource>
S:     <strength>
S:       5.0
S:     </strength>
S:     <title>
S:       Get names of proteins similar to RNA binding proteins (Simple example SADI workflow)
S:     </title>
S:     <usedTechnique>
S:       technique_keyword_content_based
S:     </usedTechnique>
S:   </recommendation>
S:
S:     ...
S: </recommendation>
S: </recommendationsSet>

```

5.6 Workflow Abstraction API

| | |
|---|--|
| <i>Name:</i> Workflow Abstraction API | <i>Architectural Context:</i>  |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Workflow+Abstraction+API | |

5.6.1 API usage

The input to the workflow abstraction services is a sequence of processes that appear in workflows. Having a sequence of names of one or more workflow processes, the client may search for other recommended processes or look for another that most typically follows in the workflow.

In both cases the workflow abstraction service is invoked in a sequence of two HTTP operations:

1. Client retrieves the service document.
2. Client parses the service document, extracts the URI template for the search or recommendation service and assembles URI for the desired result (cf. RFC6570), and issues a second HTTP GET request.

The result from the second request is the search or recommendation result.

5.6.2 HTTP methods

The only HTTP method available for this API is the GET method. The workflow abstraction service responds to an HTTP GET with the service description, which contains URI templates for the search and recommend services.

By using the URI defined by expanding these templates, the client can make GET requests to receive the search and recommendation results.

5.6.3 Link relations

| Property | Description |
|--------------------|--|
| evaluate:search | Used in the service description document. It indicates a relation between a service description and a URI template for workflow search service. The URI template is used to construct a service result URI by: <ol style="list-style-type: none"> 1. applying the URI template expansion procedures with one or more names of processes, in the order in which they appear in the workflow, and 2. resolving the resulting URI-reference to an absolute URI using normal URI resolution rules (e.g. typically, using the service document URI as a base URI) |
| evaluate:recommend | Used in the service description document. It indicates a relation between a service description and a URI template for workflow processes recommendation service. The URI template is used to construct a service result URI by: <ol style="list-style-type: none"> 1. applying the URI template expansion procedures with one or more names of |

| | |
|--|--|
| | <p>processes, in the order in which they appear in the workflow, and</p> <p>2. resolving the resulting URI-reference to an absolute URI using normal URI resolution rules (e.g. typically, using the service document URI as a base URI)</p> |
|--|--|

5.6.4 Resources and formats

| Resource | Description |
|---------------------|---|
| Service description | The service description is an RDF file that contains URI templates for accessing RO related services, including the workflow search service and the workflow process recommendation service. The RDF syntax used may be content negotiated. In the absence of content negotiation, RDF/XML should be returned. |
| Search results | <p>The result of the search service contains the following fields:</p> <ul style="list-style-type: none"> • search: the root of the response • process: the result for the query. • id: the names of the processes as they appeared in the query, separated by commas (this field is an attribute in XML and a field "@id" in JSON) • freq: the number of workflows containing the given sequence of processes (this field is an attribute in XML and a field "@freq" in JSON) • uri: a list of workflow URIs that are the result of the search <p>The search results are available in XML and JSON formats, available using content negotiation. The default format is XML.</p> |
| Recommendations | <p>The result of the recommendation service contains the following fields:</p> <ul style="list-style-type: none"> • search: the root of the response • process: the result for the query. • id: the names of the process that is recommended (this field is an attribute in XML and a field "@id" in JSON) • freq: the number of workflows containing the given sequence of processes (this field is an attribute in XML and a field "@freq" in JSON) • prob: the percentage of workflows having the recommended process appearing after the provided processes <p>The recommendation results are available in XML and JSON formats, available using content negotiation. The default format is XML.</p> |

5.6.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.6.5.1 Calling the search operation

```
C: GET /wfabstraction/rest/search?process=text1 HTTP/1.1
C: Host: service.example.org
C: Accept: application/json

S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
```

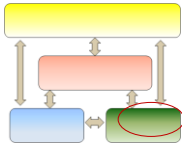


```
S:   "process": {
S:     "@freq": "30.0",
S:     "@id": "text1",
S:     "uri": [
S:       "http://ns.taverna.org.uk/2010/workflowBundle/b62064b9-4d4c-459b-bf16-
dc62f06035a3/workflow/Get_homologous_from_NCBI_homoloGene/",
S:       "http://ns.taverna.org.uk/2010/workflowBundle/5cc50154-e29b-4059-b08b-
9c35cb69c46b/workflow/Workflow32/"
S:     ]
S:   }
S: }
```

5.6.5.2 Calling the recommend operation

```
C: GET /wfabstraction/rest/recommend?process=text1 HTTP/1.1
C: Host: service.example.org
C: Accept: application/json
S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
S:   "process": {
S:     "@freq": "7.0",
S:     "@id": "Processor Split_string_into_string_list_by_regular_expression",
S:     "@prob": "0.23333333432674408"
S:   }
S: }
```

5.7 Workflow Runner API

| | |
|---|--|
| <i>Name:</i> Workflow Runner API | <i>Architectural Context:</i>  |
| <i>Function:</i> Lifecycle | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Workflow+Runner+API | |

5.7.1 API usage

Accessing the root of the service SHOULD redirect to a default server runs resource. From here the client may either:

- POST a new workflow run, providing as a minimum the workflow definition
- GET a list of existing workflow runs
- DELETE existing workflow runs

Navigating the workflow runs allows inspection of workflow status, outputs and other resources exposed by the underlying workflow server. A client may also create a new run by uploading a workflow definition, provide inputs and initiate running the workflow.

5.7.2 HTTP methods

The Workflow Runner API uses four HTTP methods: GET, POST, PUT and DELETE.

GET is used to retrieve the workflow run and its resources. In case of the workflow run itself and some other resources, content negotiation is available to get a desired RDF format of the response.

POST is used to create a new workflow run, and specifying the dependent resources at the same time. It can also be used to create a new workspace.

PUT is used to for uploading the workflow run resources, i.e. inputs, outputs, etc. It can also be used to change the status of the workflow run.

DELETE is used to delete a workflow run or its resource.

5.7.3 Link relations

| Property | Description |
|-------------------------|--|
| runner:workflow | Used in the workflow run description to link the workflow run with the main workflow to run, such as uploaded on RO creation. It is a subproperty of ore:aggregates. |
| runner:inputs | Used in the workflow run description to link the workflow run with the list of required workflow inputs, if any. It is a subproperty of ore:aggregates. |
| runner:outputs | Used in the workflow run description to link the workflow run with the list of (expected or actual) workflow outputs, if any. It is a subproperty of ore:aggregates. |
| runner:logs | Used in the workflow run description to link the workflow run with the list of logs, such as stdout, if any. It is a subproperty of ore:aggregates. |
| runner:provenance | Used in the workflow run description to link the workflow run with the list of provenance related resources, if any. It is a subproperty of ore:aggregates. |
| runner:workingDirectory | Used in the workflow run description to link the workflow run with the list of working directory and its files, if any. It is a subproperty of ore:aggregates. |

5.7.4 Resources and formats

| Resource | Description |
|--------------|---|
| Workspace | Represents a list of workflow runs, similarly to how an RO service specified a list of research objects. The only format available is text/uri-list, which returns a list of URIs that SHOULD point to research objects representing workflow runs. |
| Workflow run | A workflow run is represented as a research object and as such it shares the format of the research object as defined in the RO API. The preferred format is RDF; the support for ZIP and HTML formats is optional. The RDF format may be subject to content negotiation. |

| | |
|-------------------|---|
| Workflow | The workflow as posted by the creator. It may be a workflow description as an RDF file (format subject to content negotiation) or the actual workflow file, such as "application/vnd.taverna.t2flow+xml" in case of a Taverna 2 workflow. |
| Workflow status | A one-element list of URIs, in which the URI is one of predefined values indicating the status of the workflow run. The format is "text/uri-list". |
| Inputs | Any resource that has been submitted as an input to the workflow run. When submitting an input, it is possible to specify an external reference by using a "text/uri-list" format. |
| Outputs | Any outputs generated by the workflow run. Special formats can be used to indicate an error in generating the specific output, such as "application/vnd.wf4ever.runner.error". |
| Provenance | An ro:Folder aggregating provenance resources. |
| Working directory | An ro:Folder. |
| Logs | An ro:Folder aggregating the log files. |

5.7.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.7.5.1 Submitting a new run to workspace

Creating a new run is similar to creating a new research object, but requires the content-type text/uri-list to include the URL for the workflow definition to run.

```
C: POST http://example.com/runner/default/ HTTP/1.1
C: Content-Type: text/uri-list
C: Content-Length: ...
C: Slug: 1337
C: Authorization: Bearer h480djs93hd8C: C: http://example.net/workflow.t2flow

S: HTTP/1.1 201 Created
S: Location: http://example.com/runner/default/1337/
```

The returned location refers to a *research object* representing the run.

The client MAY provide the Slug: header to suggest a name to include in the created run, which the service MAY support. The service SHOULD ensure the returned run URI is unique, even if multiple POSTs submit the same workflow URL.

The service SHOULD attempt to retrieve the provided workflow definition before responding to the request.

The service SHOULD NOT start running the workflow immediately, but wait for the client to modify its status.

The service SHOULD fail with 502 Bad Gateway if it is unable to retrieve the submitted workflow definition due to network issues or HTTP errors (including 404), or 504 Gateway Timeout if the request for the

definition timed out. The service SHOULD include an error message in the response body to indicate the nature of this failure.

The service SHOULD fail with 501 Not Implemented if the service did successfully retrieve the workflow definition, but the underlying workflow server does not support its format. The server MAY include an error message in the response body to indicate supported workflow definition formats and/or media types.

5.7.5.2 Retrieving the workflow status

Retrieving the resource indicated with runner:status in the manifest MUST return the current status of the workflow run.

```
C: GET http://example.com/runner/default/1337/status HTTP/1.1
C: Accept: text/uri-list

S: HTTP/1.1 200 OK
S: Content-Type: text/uri-list
S:
S: http://purl.org/wf4ever/runner#Initialized
```

The returned URI list MUST include one and only one of these URIs:

| URI | Label | Description |
|---|-------------|---|
| http://purl.org/wf4ever/runner#Initialized | Initialized | The research object has been created (the RO is considered an roevo:LiveRO) |
| http://purl.org/wf4ever/runner#Ready | Ready | All required inputs and resources are provided, the workflow is ready to run (ie. the RO is an wfdesc:WorkflowInstance) |
| http://purl.org/wf4ever/runner#Queued | Queued | The workflow is in the queue, waiting to be run by the underlying workflow server |
| http://purl.org/wf4ever/runner#Running | Running | The workflow is actively running on the workflow server |
| http://purl.org/wf4ever/runner#Failed | Failed | The workflow could not run, or failed while running |
| http://purl.org/wf4ever/runner#Finished | Finished | The workflow completed running |
| http://purl.org/wf4ever/runner#Cancelled | Cancelled | The workflow run was cancelled, for instance by the client or by a server time out |
| http://purl.org/wf4ever/runner#Archived | Archived | The workflow runner service has finished post-run processing (the RO is now considered an roevo:ArchivedRO) |

The service might also include other, third-party specific URIs, like <http://api.example.com/status/StartingCloudInstance>.

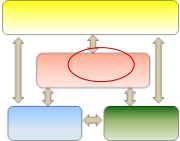
The service might not support all of the above status types, but MUST support Initialized, Running and Archived.

The service MAY do its own state transitions, like Initialized to Ready or Finished to Archived, but SHOULD NOT start the workflow as Running unless the client has requested Queued or Running. (See below).



The state Archived means that the Research Object has a complete view of the workflow run. Until the workflow run is in this state, requests for resources such as outputs, the manifest and provenance MAY give incomplete results or 404 Not Found. The workflow service SHOULD automatically transition from Finished to Archived, but SHOULD NOT do this transition from failure states such as Failed or Cancelled.

5.8 Workflow-RO Transformation API

| | |
|---|---|
| <i>Name:</i> Workflow - RO Transformation API | <i>Architectural Context:</i>  |
| <i>Function:</i> Data Management & Analysis | |
| <i>Level:</i> Data Management & Analysis | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/Wf-RO+transformation+service+API | |

5.8.1 API usage

The client starts the transformation job by sending a POST request with the following parameters:

- A URI of a t2flow workflow
- RO identifier for the new or existing RO
- RO Service URI
- OAuth access token for the RO Service

The service creates a new job resource, which can later be monitored by the client. The job resource returns a list of resources that have been created, and the status of the transformation. If necessary, the transformation job can be cancelled.

5.8.2 HTTP methods

The API uses a POST method to create a transformation job, GET to retrieve the status of a job and DELETE to cancel a running job.

5.8.3 Link relations

Creating the transformation job is done by a request to the service URI, and all other requests are done using the URI returned by the first one.

5.8.4 Resources and formats

| Resource | Description |
|--------------------|---|
| Transformation job | A job description is a JSON object with the following attributes: <ul style="list-style-type: none"> • resource: URI of the workflow that is transformed. • format: MIME type of the workflow. • ro: URI of the research object to which the service saves the resources. • status: Job status, allowed values are: "running", "done", "failed". • token: OAuth 2.0 Bearer token of the research object owner. |

- | | |
|--|---|
| | <ul style="list-style-type: none"> • added: a list of URIs of resources generated by the service. • reason: an explanation of the status, in particular in case of error. |
|--|---|

5.8.5 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.8.5.1 Creating a conversion job

The clients sends a transformation job parameters in a POST request.

```
C: POST http://example.net/translate/jobs/ HTTP/1.1
C: Content-Type: application/json
C:
C: {
C:   "resource": "http://www.example.com/workflow.t2flow"
C:   "format": "application/vnd.taverna.t2flow+xml"
C:   "ro": "http://www.example.org/rodl/ROs/someRO/"
C:   "token": "'47d5423c-b507-4e1c-7'"
C: }

S: HTTP/1.1 201 Created
S: Location: http://example.net/translate/jobs/fefe-fefefef-fefefefefefe
```

5.8.5.2 Checking a job status

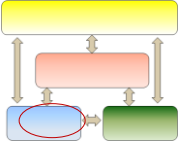
The job status may be retrieved with a GET request to the job URI.

When the job has finished, the service may provide its status for an arbitrary amount time, large enough to allow clients to check that the job has finished. Retrieving the job status after that time will result in 404 Not Found.

```
C: GET http://example.net/translate/jobs/fefe-fefefef-fefefefefefe HTTP/1.1
C: Accept: application/json

S: HTTP/1.1 200 OK
S: Content-Type: application/json
S:
S: {
S:   "resource": "http://www.example.com/workflow.t2flow"
S:   "format": "application/vnd.taverna.t2flow+xml"
S:   "ro": "http://www.example.org/rodl/ROs/someRO/"
S:   "status": "running"
S:   "added": [ "http://www.example.com/ROs/1/workflow.t2flow",
S:             "http://www.example.com/ROs/1/annotations/1.ttl" ]
S: }
```

5.9 User Management API

| | |
|---|---|
| <i>Name:</i> User Management API | <i>Architectural Context:</i>  |
| <i>Function:</i> Storage | |
| <i>Level:</i> Foundational | |
| <i>Specification URL:</i> http://wf4ever-project.org/wiki/display/docs/User+Management+2 | |

5.9.1 API usage

The users and clients are independent entities. The clients can create users, and subsequently modify them and delete them. The clients are typically created by admins, with the aim of registering an application that will be requesting an access token from the user.

Admins can request to create an access token for an existing user and client application. The access tokens cannot be modified but they can be revoked (deleted).

A separate resource is available for clients that have received an access token and want to get the user identity associated with it, i.e. the human friendly name. Sending a GET request to that resource with an access token results in a redirection to the user resource.

5.9.2 Conformance with external APIs

For managing user identification, the **Simple Cloud Identity Management API** [23] is used.

The SCIM Protocol is an application-level, REST protocol for provisioning and managing identity data on the web. The protocol supports creation, modification, retrieval, and discovery of core identity Resources; i.e., Users and Groups, as well as custom Resource extensions.

The SCIM API is used as a base and not all required resources may initially be implemented, i.e. Groups.

The following design rules are taken from SCIM:

- The Users/ resource API
- The use of JSON for resource representation
- The use of OAuth 2.0 Bearer schema for request authorization

Following the SCIM recommendation and previous experiences, the API uses **OAuth 2.0 API** [24].

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

5.9.3 HTTP methods

The API uses a GET, POST, PUT and DELETE operations to retrieve, create, update and delete resources respectively. GET is also used to verify the identity of the user associated with an access token.

5.9.4 Link relations

The Users, Clients and Access Tokens resources are available as “text/uri-list”, in which case the format is an explicit list of links.

| Property | Description |
|--------------|---|
| user:users | Used in the service description document. It indicates a relation between a service description and a URI for the Users resource. |
| user:clients | Used in the service description document. It indicates a relation between a service description and a URI for the Clients resource. |
| user:tokens | Used in the service description document. It indicates a relation between a service description and a URI for the Access Tokens resource. |
| user:whoami | Used in the service description document. It indicates a relation between a service description and a URI for the Who Am I resource that allows to identify the access token owner. |

5.9.5 Resources and formats

| Resource | Description |
|---------------------|---|
| Service description | The service description is an RDF file that contains URI templates for accessing RO related services, including the workflow search service and the workflow process recommendation service. The RDF syntax used may be content negotiated. In the absence of content negotiation, RDF/XML should be returned. |
| Users | Represents all users. The format is “text/uri-list” with a list of all users in the system. |
| User | Represents a user. The format is JSON with the following fields: <ul style="list-style-type: none"> • schemas: a list of schema URIs, based on the SCIM API • userName: user identifier, if applicable • password: user password, if applicable • openId: a list of OpenIDs associated with the user • name: various ways of presenting the user name, in particular: • formatted: the human-friendly user name |
| Clients | Represents all clients. The format is “text/uri-list” with a list of all clients in the system. |
| Client | Represents a client that may be associated with an access token. The format is JSON with the following fields: <ul style="list-style-type: none"> • name: the application human-friendly name • callbackURL: the callback URL as required by OAuth 2. |

| | |
|---------------|--|
| Access Tokens | Represents all access tokens. The format is "text/uri-list" with a list of all access token URIs in the system. |
| Access Token | Represents an access token issued for a particular user and client. The format is JSON with the following fields: <ul style="list-style-type: none"> • client: the client URI • user: the user URI |
| Who Am I | When dereferenced with an access token, this resource redirects to the User resource corresponding with the owner of the access token. |

5.9.6 Sample client-server interactions

The full specification of client-server interactions with examples can be found in the API specification. Below are selected operations, for illustration purposes.

5.9.6.1 Creating a user.

Allowed for: admins.

```
C: POST /Users HTTP/1.1
C: Host: service.example.org
C: Accept: application/json
C: Authorization: Bearer h480djs93hd8
C: Content-Length: ...
C:
C: {
  "schemas":["urn:scim:schemas:core:1.0"],
  "userName":"bjensen",
  "password":"cleartext",
  "openId":"bjensen.myopenid.com",
  "name":{
    "formatted":"Ms. Barbara J Jensen III"
  }
}

S: HTTP/1.1 201 Created
S: Content-Type: application/json
S: Location: https://sandbox/rodl/Users/2819c223-7f76-453a-919d-413861904646
S:
S: {
  "schemas":["urn:scim:schemas:core:1.0"],
  "userName":"bjensen",
  "openId":"bjensen.myopenid.com",
  "name":{
    "formatted":"Ms. Barbara J Jensen III"
  }
}
```

6 Compliance with Preservation Standards

OAIS (Open Archival Information System), ISO 14721-2012 standard, is a reference model for digital preservation developed originally by the Consultative Committee for Space Data Systems (CCSDS) [1]. OAIS describes "open archival information systems" which are concerned with preserving information for the benefit of a *designated community*. The reference model defines the basic functional components of a system dedicated to the long-term preservation of digital information, identifies and describes the entities which constitute the OAIS environment where the archive operates, details the key internal and external system interfaces which allow the interaction with these entities, and characterizes the information objects managed by the system. The reference model would also enumerate a set of minimum requirements an archival system is expected to meet. Being the preservation of workflows one of the main objectives of the project, the conformance with the widely accepted OAIS model has been considered as one key feature of Wf4ever Architecture. Hence, in line with OAIS, our architecture includes a component, namely the Research Object Digital Library, which plays the role of an open archival information system, taking advantage of dLibra and dArceo preservation functionalities. However, the OAIS specification document provides a very generic notion of conformance, which says, "a conforming OAIS Archive implementation shall support the OAIS information model (Ch.2.2)" and "shall fulfill the responsibilities listed in the document (Ch.3.1)". In fact, the meaning of 'OAIS-compliant' is necessarily vague because the reference model is a conceptual framework rather than a concrete implementation [29]. Therefore, since there is no formal OAIS certification in place yet, we followed a twofold approach for testing the compliance of our architecture against preservation standards, mapping our concepts and functionalities:

- the OAIS standard itself [1], analysing the *information model* and *functional entities* roles, services and functions defined by OAIS and checking whether they are present or can be mapped in the architecture (see Section 7.1); similar to the approach taken by the National Archives (TNA) and the UK Data Archive (UKDA) [29];
- the Merritt micro-services curation approach developed by the UC3 (University of California Curation Center) [13] [14]. UC3 microservices have been aligned to the "The DCC curation lifecycle model"[15] designed by the Digital Curation Center. They can be grouped into four categories that provide incrementally increasing levels of preservation assurance and curation value. This approach has been used within our project to analyse preservation services of our architecture realization (see Section 7.2).

6.1 Mapping to OAIS functional entities

6.1.1 OAIS Functional Entities

The OAIS functional model comprises six functional entities and related interfaces, as depicted in Figure 7.1. These main elements represent the high-level services that fulfil the OAIS role of preserving information and making it available to a designated community. Hence, an archive has to implement each of these services to be compliant with OAIS. According to OAIS, the information objects or information packages, managed by these entities comprise content information (the target information to be preserved) and preservation

description information (metadata necessary to identify and understand the environment in which the content was created). Information packages are categorized into Submission Information Package (SIP), Archival Information Packages (AIPs) and Dissemination Information Package (DIP). SIP is the package sent to an OAIS by a producer, which is transformed into one or more AIP for preservation. When requested by a consumer, OAIS provides all or part of the AIP in the form of a DIP.

A short description of the role of each of these entities is provided below.

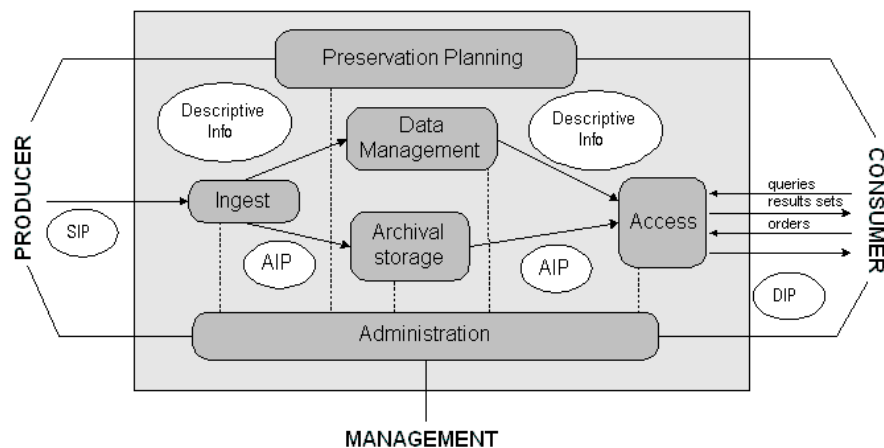


Figure 6.1. OAIS Functional Entities³

- **Ingest:** provides the services and functions to accept SIPs from producers and prepare the contents for storage and management within the archive.
- **Archival Storage:** provides the services and functions for the storage, maintenance and retrieval of AIPs.
- **Data Management:** provides the services and functions for populating, maintaining, and accessing both Descriptive Information which identifies and documents Archive holdings and administrative data used to manage the Archive
- **Administration:** provides the services and functions for the overall operation of the Archive system.
- **Preservation Planning:** provides the services and functions for monitoring the environment of the OAIS, providing recommendations and preservation plans to ensure that the information stored in the OAIS remains accessible to, and understandable by, the Designated Community over the Long Term, even if the original computing environment becomes obsolete.
- **Access:** provides the services and functions that support Consumers in determining the existence, description location and availability of information stored in the OAIS, and allowing Consumers to request and receive information products

³ Copy taken from <http://wiki.datadryad.org/wg/dryad/images/5/53/Oais-model.png>

6.1.2 Information Objects in Wf4Ever

The concept of Information Package in OAIS, which consists of the Content Information and related metadata (called Preservation Description Information), can be roughly mapped to a Research Object, which is a semantically rich aggregation of resources that bring together data, methods and people in scientific investigations. Research Objects, similar to Information Package, are conceptual containers (aggregations) of Resources (Content Information) and annotations about them (metadata), including provenance, context and reference information. In Wf4Ever, the packages submitted to the preservation infrastructure (SIP) are basically individual resources (e.g., workflows, datasets) or simple aggregation of resources, such as the Packs from myExperiment. Within our infrastructure, these resources are transformed into Research Objects (AIP). For instance, one myExperiment Pack will generate one Research Object, which includes all the resources from the pack plus metadata generated during the transformation regarding the provenance, context, semantic relations, etc. Similarly one workflow submitted to our infrastructure may generate a Research Object, which will include the original workflow, plus a workflow bundle (a new workflow format), metadata extracted from the workflow regarding its structure (e.g., inputs, outputs), and other metadata generated. In response to a request, our infrastructure can return (DIP) the whole Research Object, in different representations (e.g., as a ZIP archive, HTML page, metadata), or it can provide individual resources within the Research Object (e.g., workflows, datasets).

In the following, we discuss how the OAIS entities can be mapped to the Wf4Ever architecture.

6.1.3 Mapping to OAIS functions

6.1.3.1 Ingest functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|--------------------|---|---|
| Receive submission | Provides the appropriate storage capability or devices to receive a SIP from the <i>Producer</i> (or from <i>Administration</i>) | Clients submit Research Objects or their components via a REST API called the RO API ⁴ , implemented by RODL. |
| Quality Assurance | Validates (QA results) the successful transfer of the SIP to the temporary storage area | RODL accepts the digital objects using the REST API based over HTTP. The network protocols ensure that the messages are not broken. RODL validates that the messages conform to the API (checks the format and link relations) before creating or updating the AIP. |
| Generate AIP | Transforms one or more SIPs into one or more AIPs that conform to the Archive's data formatting and documentation standards | RODL creates a new Research Object with the provided data. In some cases external services can be used to prepare metadata according to the RO model, e.g., the Workflow-RO Transformation Service. |

⁴ <http://www.wf4ever-project.org/wiki/display/docs/RO+API+6>

| | | |
|----------------------------------|--|--|
| Generate Descriptive Information | Extracts <i>Descriptive Information</i> from the AIPs and collects descriptive information from other sources to provide to <i>Coordinate Updates</i> , and <i>Data Management</i> | RODL generates metadata about Research Objects and their resources every time the Research Object is updated. This includes the author of the changes, the date, evolution information and automatic relations between resources. Additionally, RODL extracts metadata from metadata files provided by users and stores them in the triple store and in the indexing server. |
| Coordinate Updates | Transfers the AIPs to <i>Archival Storage</i> and the <i>Descriptive Information</i> to <i>Data Management</i> | After RODL parses the requests and generates all changes that must be done to the Research Objects, respective metadata resources are updated in the triple store and all respective resources are updated in the storage backend (dLibra and dArceo). |

6.1.3.2 Archival Storage functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|--------------------------|---|---|
| Receive Data | Receives a storage request and an AIP from <i>Ingest</i> and moves the AIP to permanent storage. | The storage backend of RODL (both dLibra and dArceo) receive the resource serializations from RODL. dLibra stores them as working copies, for fast access, and dArceo stores them in a secure, replicable storage platform. |
| Manage Storage Hierarchy | Positions, via commands, the contents of the AIPs on the appropriate media based on storage management policies, operational statistics, or directions from <i>Ingest</i> via the storage request. It also conform to any special levels of service required for the AIP, or special security measures required, and ensures appropriate level of protection of AIP | RODL stores the Research Objects in dLibra (working copies) and in dArceo (secure backup copies). The transfer to dArceo is asynchronous and may not be performed immediately. |
| Replace Media | Provides the capability to reproduce the AIPs over time. | dArceo makes sure that Research Objects can be used and run in Long Time by regularly checking their status with a set of internal and external services and performing migrations when necessary. These include the Checklist Evaluation Service that checks the data description and dependencies on external resources, as well as the Workflow-RO Transformation Service that can transform and old simple t2flow format into a scufl2 format following the RO model. |
| Error Checking | Provides statistically acceptable assurance that no components of the AIP are corrupted in <i>Archival Storage</i> or | Research Objects in dArceo are stored on a secure storage platform offering replication and checksums checking. Working copies in |

| | | |
|-------------------|--|--|
| | during any internal <i>Archival Storage</i> data transfer | dLibra are periodically verified with the secure copies to ensure their fixity. |
| Disaster Recovery | Provides a mechanism for duplicating the digital contents of the Archive collection and, for example, storing the duplicate in a physically separate facility. | Research Objects in dArceo are stored on a secure storage platform offering geographical replication |
| Provide Data | Provides copies of stored AIPs to Access | Working copies are maintained in dLibra and are accessible synchronously. |

6.1.3.3 Data Management functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|--------------------------|---|---|
| Administer Database | Maintains the integrity of the data management database that provides a storage mechanism, which can be queried in some way, for storing both <i>Descriptive Information</i> (identifies and describes the Archive holdings) and system information (supports Archive operations) | The Research Object metadata are stored in the triple store, which is the primary source of information about them and can be queried using the SPARQL language. Additionally, the system metadata are stored in a MySQL database. |
| Perform Queries | Receives a query request from <i>Access</i> and executes the query to generate a query response that is transmitted to the requester | RODL implements the SPARQL endpoint, which allows clients to perform queries to the triple store containing the Research Object metadata. |
| Generate Report | Receives a report request from <i>Ingest</i> , <i>Access</i> or <i>Administration</i> and executes any queries or other processes necessary to generate the report that it supplies to the requester. | RODL implements the SPARQL endpoint, which allows clients to perform queries to the triple store and generate different types of views. Additionally, RODL REST API allows receiving summary views of Research Objects, e.g., the manifest or the manifest with all annotations, as well as reports about statuses of operations, e.g., a snapshotting operation. |
| Receive Database Updates | Adds, modifies or deletes information in the Data Management persistent storage. Mainly <i>Descriptive Information</i> for the new AIPs from <i>Ingest</i> , system updates and review updates from <i>Administration</i> . | RODL updates the triple store with metadata of new or updated Research Objects. The system information, such as users and access control tokens are stored in the MySQL database. |

6.1.3.4 Administration functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|--------------------------------|---|--|
| Negotiate Submission Agreement | Solicits desirable archival information for the OAIS and negotiates <i>Submission Agreements</i> and data submission schedule with <i>Producers</i> . | Updates to the Research Objects must be performed according to the RO API. In particular they must conform to formats and link relations specified in the API. |

| | | |
|----------------------------------|---|---|
| Manage System Configuration | Provides system engineering for the Archive system to monitor continuously the functionality of the entire Archive system and systematically control changes to the configuration. It receives migration packages (and sends requests to <i>Archival Information Update</i>) and system evolution policies. | The contents of RODL are accessible using the RO Portal web interface. Other diagnostic information is currently available to the technical team using general application monitoring tools, and changes in configuration are performed manually when necessary. |
| Archival Information Update | Provides a mechanism for updating the contents of the Archive. Receives change requests, procedures and tools from <i>Manage System Configuration</i> . | Updates to the storage contents can be performed also via the RO API. Additionally, updates to the storage backend can be performed using the dLibra administrator utilities. Updates to the triple store are currently performed using special scripts generated on demand. |
| Physical Access Control | Provides mechanisms to restrict or allow physical access (doors, locks, guards) to elements of the Archive, as determined by Archive policies | The servers hosting RODL, dLibra and dArceo are located in secure server rooms in PSNC. Servers hosting the secure, replicated storage platform (Platon U4) are located in secure server rooms in a number of research institutions in Poland. |
| Establish Standards and Policies | Establishes and maintains the Archive system standards and policies. It receives budget information and policies such as the OAIS charter, scope, resource utilization guidelines, and pricing policies from <i>Management</i> and provides periodic reports. Develops storage management, disaster recovery and security policies. | The project management board (PMB) is in charge of defining policies for migration of Research Object components, e.g., workflows, storage management and disaster recovery. Additionally, the technical team has established techniques for implementing storage management and disaster recovery using the monitoring capabilities of dArceo to verify the integrity of stored resources. |
| Audit Submission | Verifies that submissions (SIP or AIP) meet the specifications of the Submission Agreement. It verifies their understandability by the Designated Community. It verifies that the quality of the data meets the requirements of the Archive and the review committee. | In Wf4Ever the APIs are developed and approved by all members of the project, and their implementations are verified by cross-validation, which occurs when services and clients are integrated and must work together on the same resources. In case of RODL, the quality of the implementation is verified by the services that store and retrieve Research Objects in it. Additionally, the quality of data is verified with the checklist evaluation service. |
| Activate Requests | Maintains a record of event-driven requests and periodically compares it to the contents of the Archive to determine if all needed data is available | Event-driven requests are activated by the monitoring services provided by dArceo, e.g., comparing the content of RODL against the backup copy in dArceo. |
| Customer | Creates, maintains and deletes Consumer | Although Wf4Ever does not bill for accessing |

| | | |
|---------|----------|---|
| Service | accounts | the stored resources, RODL maintains accounts for the users and other services, which are the consumers of the content. |
|---------|----------|---|

6.1.3.5 Preservation Planning functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|---|---|--|
| Monitor Designated Community | Interacts with Archive Consumers and Producers to track changes in their service requirements and available product technologies. | We monitor communities of users of workflow-intensive applications. In particular, we participate and organize workshops in the communities of our case studies domains (Bioinformatics and Astronomy), and other related communities and events. In fact, as part of the description of work, Wf4Ever tasks include the creation of these communities and monitoring of community trends. |
| Monitor Technology | Tracks emerging digital technologies, information standards and computing platforms to identify technologies which could cause obsolescence in the Archive's computing environment and prevent access to some of the Archive's current holdings | As part of the description of work, Wf4Ever tasks include the tracking of scientific and technological trends in scientific workflow lifecycle management and sharing, social networks, Semantic Web and digital libraries. This task includes experimental pilots of alternative technological infrastructures and approaches. |
| Develop Preservation Strategies and Standards | Develops and recommends strategies and standards, assesses risks, to enable the Archive to make informed tradeoffs as it establishes standards, sets policies, and manages its system infrastructure. | As part of the description of work, Wf4Ever tasks include the preparation and maintenance of the quality and risk management plan. |
| Develop Packaging Designs and Migration Plans | Develops new Information Package designs and detailed migration plans and prototypes, to implement <i>Administration</i> policies and directives (e.g., migration goals). It sends migration packages to <i>Administration</i> . | We have established initial migration plans of workflows, which are realized with the WF-RO transformation service. |

6.1.3.6 Access functional entity in Wf4Ever

| Function | Brief description | Wf4Ever realization |
|------------------------------|--|--|
| Coordinate Access Activities | Provides one or more interfaces to the information holdings of the Archive. It determines if resources are available to perform a request, assures that the user is authorized to access and receive the requested items and notifies the Consumer that a request has been accepted or | The two primary access interfaces of RODL are the REST API and the SPARQL endpoint. Both use the OAuth 2 authorization framework to identify the requesting party and the access rights. |

| | | |
|------------------|--|--|
| | rejected. | |
| Generate DIP | Accepts a dissemination request, retrieves the AIP from <i>Archival Storage</i> , moves a copy of the data to a temporary storage area for further processing, and notifies the <i>Coordinate Access Activities</i> that the DIP is ready for delivery. | RODL REST API and SPARQL endpoint support content negotiation with regards to metadata. Data are transformed to a requested format on the fly. Binary data are available in the format in which they were submitted. Finally, Research Objects are also available as HTML pages, which are generated by an application called RO Portal, closely associated with RODL. |
| Deliver Response | Handles both on-line and off-line deliveries of responses (DIPs, query responses, reports and assistance) to Consumers. For on-line delivery, it accepts a response from <i>Coordinate Access Activities</i> and prepares it for on-line distribution in real time via communication links. For off-line delivery it retrieves the response from the <i>Coordinate Access Activities</i> , prepares packing lists and other shipping records, and then ships the response. | RODL REST API and SPARQL endpoint are both synchronous interfaces and the responses are generated immediately from the working copies stored in dLibra. |

6.2 Mapping to UC3 Microservices

UC3 - University of California Curation Center - (<http://www.cdlib.org/services/uc3/index.html>) has made a great and well-founded work in the definition of a micro-services approach to curation that could serve as a basis for our purposes. "*Micro-services are an approach to digital curation based on devolving curation function into a set of independent, but interoperable, services that embody curation values and strategies*". The micro-services approach provides many advantages, as each service is small and self-contained, allowing easier development, deployment, maintaining, enhancement, and if needed, replacement. UC3 microservices have been defined as the Merritt approach, and have been aligned to the "The DCC curation lifecycle model" (<http://www.dcc.ac.uk/resources/curation-lifecycle-model>) proposed by the Digital Curation Center. Moreover, the Merritt approach is OAIS compliant, which can also guide the architecture definition in accordance to OAIS.

6.2.1 Curation Micro-Services at UC3: the Merritt approach

The initial set of micro-services at UC3 (<http://www.cdlib.org/services/uc3/curation>), can be grouped into four categories that provide incrementally increasing levels of preservation assurance and curation value. These are the high level UC3 mission goals. (For more information and documentation, see the UC3 Curation wiki [11], the reports about their digital preservation system Merritt [13] and the UC3 Curation Foundations [14].)

1. *Providing security through redundancy*

- Identity service: provides a means to uniquely identify various entities of curation interest.

- Storage service: provides a means to manage digital object files.
- Fixity service: provides a means to verify the bit-level integrity of files managed by the Storage service.
- Replication service: provides a means to provide a globally-fault tolerant storage environment.

2. *Maintaining meaning through descriptive context*

- Inventory service: provides a means to associate various types of syntactic, semantic, and pragmatic descriptive information with digital objects. *Metadata*.
- Characterization service: provides one possible source for descriptive information managed by the Catalog service.

3. *Facilitating utility through service*

- Ingest service: provides a means to add new digital content into the curation environment for active management by UC3.
- Index service: builds searchable indexes of object descriptive information and content.
- Search service: supports content request and delivery via index-based search and browse of managed content and description.
- Transformation service: provides a means to transcode digital object representations (that is, sets of files) from existing forms to newly required forms.

4. Adding value through use

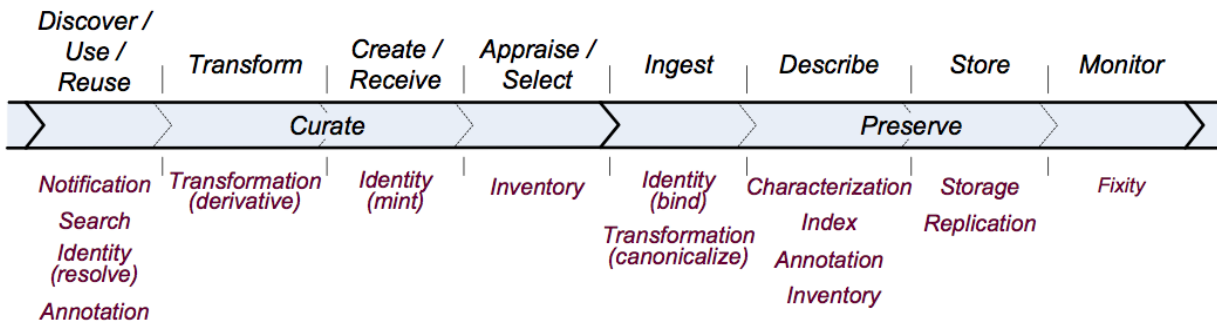
- Notification service: provides a means to notify user communities that digital content is being managed and is available for use (*also called publication in previous documents*).
- Annotation service: provides a means for user-driven enrichment of managed object description.

These services can be seen as a progression depending on the level of preservation and curation value they offer. In fact, the two first groups or levels (*providing security*>**Protection** and *maintaining meaning*>**Interpretation**) are considered "preservation" as they ensure the bit integrity of the object, the content *state* and *context*, and the two last groups (*facilitating utility*>**Application** and *adding value*>**Interoperation**) are considered "curation" services, as they focus on the content of the digital object, not only its carrier, and their goal is to maintain the *value* and usefulness / *service* of their content. The following figure A1 representing the Merritt micro-services architecture allows a better understanding of these concepts, and the relationships between the services and the levels of preservation/curation offered.

| | | | |
|-----------------|---------------------|---|---|
| Curation | <i>Value</i> | <i>Interoperation</i> <i>Annotation</i> <i>Notification</i> | <i>“Lots of uses keeps stuff valuable”</i> |
| | <i>Service</i> | <i>Application</i> <i>Transformation</i> <i>Search</i> <i>Index</i> <i>Ingest</i> | <i>“Lots of services keeps stuff useful”</i> |
| | <i>Preservation</i> | <i>Interpretation</i> <i>Characterization</i> <i>Inventory</i> | <i>“Lots of description keeps stuff meaningful”</i> |
| | <i>State</i> | <i>Protection</i> <i>Replication</i> <i>Fixity</i> <i>Storage</i> <i>Identity</i> | <i>“Lots of copies keeps stuff safe”</i> |

A 1 Merritt micro-services

Finally, this micro-services approach has been aligned to the DCC curation lifecycle model (<http://www.dcc.ac.uk/resources/curation-lifecycle-model>) – as depicted in Figure A2, easing the task of identifying the preservation and curation activities that wf4ever would take into account in its own preservation methodology and how these microservices could help to fulfill these activities.



A 2 Micro-service applicability throughout the curation lifecycle (due to Higgins)

6.2.2 The Merritt approach in Wf4Ever

The UC3 microservices may work as a general list of requirements for preserving digital objects, such as Research Objects. By identifying how each of them is implemented in Wf4Ever we can check to what extent Wf4Ever meets the general preservation requirements. Additionally, Wf4Ever should implement workflow-specific preservation requirements that are related to RO evolution model and are not included in the general microservices list below.

Providing security through redundancy

| UC3 microservice | Wf4Ever implementation |
|------------------|------------------------|
| | |

| | |
|---------------------|--|
| Identity service | URIs are used for identifying all entities. |
| Storage service | RODL is used for storing Research Objects. External resources aggregated by the Research Object are only referenced. |
| Fixity service | RODL uses checksums at file level. |
| Replication service | RODL provides replication services. Contents are replicated in dArceo, which are stored on a secure storage platform offering geographical replication |

Maintaining meaning through descriptive context

| UC3 microservice | Wf4Ever implementation |
|--------------------------|---|
| Inventory service | RO model permits annotations, which are used for arbitrary and user-generated metadata. RODL generates annotations for the research object evolution information and let the users to submit their own annotations. Additionally, RODL generates basic descriptive metadata, e.g., creator, creation date, etc. |
| Characterization service | RODL stores and publishes metadata as RO annotations, based on the RO model. |

Facilitating utility through service

| UC3 microservice | Wf4Ever implementation |
|-------------------------|--|
| Ingest service | RODL REST interface allows adding digital objects in order to create or update research objects. |
| Index service | Research objects in RODL are indexed based on their metadata, which may be generated by RODL or submitted by users. |
| Search service | It is possible to search for research objects in RODL based on the index maintained by the Index service. |
| Transformation service | Transformation services are provided via dArceo, currently for the transformation of Workflows in t2flow format into workflow bundles (the new format for Taverna workflows). Other may be provided in the |

| | |
|--|--------|
| | future |
|--|--------|

Adding value through use

| UC3 microservice | Wf4Ever implementation |
|-------------------------|--|
| Notification service | Notifications are generated by dArceo. Notifications are generated when RO quality goes below a defined threshold (determined using checklist service), when the integrity check fails or when obsolete workflow formats are detected. |
| Annotation service | RO model permits annotations that are used for system-generated and user-generated metadata. |

| Other Wf4Ever microservices | Wf4Ever implementation |
|------------------------------------|-------------------------------|
| Checklist Service | Implemented as a web service. |
| Recommender Service | Implemented as a web service |
| Workflow Abstraction Service | Implemented as a web service |
| Stability Service | Implemented as a web service |
| Workflow Runner Service | Implemented as a web service |
| WF-RO Transformation Service | Implemented as a web service |

7 Summary and Reflection

The design of the Wf4Ever Architecture and Toolkit has been a substantial task that has drawn on the full breadth of experience across the team. Having established early principles, the design was motivated not just by what we wanted to achieve but also by what we wanted to avoid – we did not wish to set the architecture in stone too early, or to design it based on an isolated exercise in requirements capture, or create artificial complexity, or to re-invent wheels.

We believe we have succeeded in defining and implementing an RO-centric architecture, and in doing so have demonstrated that the RO model (as its associated models) provide a suitable basis for interoperability within a service-based architecture for management of scientific workflows.

The instantiation of the Architecture in the Wf4Ever Toolkit provides a clear illustration of this approach to scientific preservation through this mechanism of interchange – the RO model, its extensions, the REST APIs built around it, and the services and clients that implement and use these APIs. In the “RO-enablement” of myExperiment we have demonstrated the flexibility of the Architecture in adapting, and being adapted, to existing and independently developed software and tools.

The Architecture and Toolkit were co-evolved with user requirements and as first-hand experience was gathered. Through the Architecture and Toolkit foundation services are in place and our approach has been validated. Beyond this, we believe and hope future development can and will build upon the flexibility of the RO model and the simplified client development process afforded by our APIs; to further enhance and streamline the scientist's experience through “RO-enablement” of more applications and through this integration with the Wf4Ever Architecture.

Acknowledgements

This document results from the work of the *Wf4Ever Architecture Task Force*, which draws together experience and expertise from all partners in the Wf4Ever project, further developed through *showcases* throughout the project. The task force was chaired by the triumvirate of Raul Palma (PSNC) who leads WP1 *Software Architecture and Technology for Scientific Workflow Preservation*, David De Roure (OXF) who is the Wf4Ever Lead Architect, and Stian Soiland-Reyes (UNIMAN). While this deliverable (D1.3v2) is led by the University of Oxford, it is entirely based on the work of the broader team, with significant design contributions from Raul Palma (PSNC), Piotr Holubowicz (PSNC), Kevin Page (OXF), Graham Klyne (OXF) and Stian Soiland-Reyes (UNIMAN). It is a culmination of API design and implementation in Services developed by all partners, with additional contributions specifically to the document from Esteban García Cuesta (ISOCO), Gema Bueno de la Fuente (UPM) and Rafael Gonzalez (UPM). Our expert users Kristina Hettne (LUMC), Marco Roos (LUMC) and Jose Enrique Ruiz (IAA) have provided essential input to the design, development and test of the toolkit. Jiten Bhagat (UNIMAN) made important contributions to the design process in the earlier part of the project.

8 References

- [1] Reference Model for an Open Archival Information System (OAIS), Recommended Practice, CCSDS 650.0-M-2 (Magenta Book) Issue 2, June 2012
- [2] Bizer, C., Heath, T. and Berners-Lee, T. (2009) Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5 (3). pp. 1-22.
- [3] De Roure, D., Goble, C. and Stevens, R. (2009) The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems* 25, pp. 561-567. doi:10.1016/j.future.2008.06.010
- [4] Fielding, Roy Thomas (2000), *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine
- [5] Leonard Richardson and Sam Ruby, *RESTful Web Services*, O'Reilly Media, May 2007
- [6] Berners-Lee, T. (2006), *Linked Data, Design Issues*, <http://www.w3.org/DesignIssues/LinkedData.html>
- [7] Page, K. R., De Roure, D. C. and Martinez, K. (2011) REST and Linked Data: a match made for domain driven development? In: 2nd International Workshop on RESTful Design , 28/03/2011, Hyderabad, India.
- [8] Paolo Ciccarese, Marco Ocana, Leyla J Garcia Castro, Sudeshna Das and Tim Clark (2011). An open annotation ontology for science on web 3.0, *Journal of Biomedical Semantics* 2011, 2(Suppl 2):S4 doi:10.1186/2041-1480-2-S2-S4
- [9] Carlos Ruiz, Guillermo Álvaro, and José-Manuel Gómez-Pérez. 2011. A framework and implementation for secure knowledge management in large communities. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies (i-KNOW '11)*, Stefanie Lindstaedt and Michael Granitzer (Eds.). ACM, New York, NY, USA
- [10] The ADMIRAL Project: A Data Management Infrastructure for Research Across the Life sciences, Project Web Site, <http://imageweb.zoo.ox.ac.uk/wiki/index.php/ADMIRAL>
- [11] UC3 Curation wiki. Available at: <https://confluence.ucop.edu/display/Curation/Home>.
- [12] OCLC, and CRL (2007). *Trustworthy Repositories Audit & Certification: Criteria and Checklist*. Available at http://www.crl.edu/sites/default/files/attachments/pages/trac_0.pdf.
- [13] UC Curation Center / California Digital Library (2010). *Merritt: An Emergent Approach to Digital Curation Infrastructure*. Available at <https://confluence.ucop.edu/download/attachments/13860983/Merritt-latest.pdf>.
- [14] UC Curation Center / California Digital Library (2010). *UC3 Curation Foundations*. Available at <https://confluence.ucop.edu/download/attachments/13860983/UC3-Foundations-latest.pdf>.
- [15] Digital Curation Centre (2009). *DCC Curation Lifecycle Model*. Available at <http://www.dcc.ac.uk/resources/curation-lifecycle-model>.
- [16] MIBBI: Minimum Information for Biological and Biomedical Investigations. <http://mibbi.org/>
- [17] F. Crestani. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review*, 11(6): 453-482, 1997
- [18] AO - Annotation Ontology: <http://code.google.com/p/annotation-ontology/>
- [19] Open Archives Initiative Object Reuse and Exchange: <http://www.openarchives.org/ore/>

- [20] AtomPub: <http://tools.ietf.org/html/rfc5023>
- [21] OpenSearch: <http://www.opensearch.org/Specifications/OpenSearch/1.1>
- [22] SPARQL endpoint specification: <http://www.w3.org/TR/rdf-sparql-protocol/#query-bindings-http>
- [23] Simple Cloud Identity Management: Protocol 1.0: <http://www.simplecloud.info/specs/draft-scim-api-00.html>,
- [24] OAuth 2.0: <http://oauth.net/2/>
- [25] PROV-O: The PROV Ontology: <http://www.w3.org/TR/prov-o/>
- [26] D2.2v1. Design, implementation and deployment of workflow lifecycle management components – Phase I. UNIMAN. Wf4Ever Deliverable. July 2012.
- [27] D3.2v1. Design, implementation and deployment of Workflow Evolution, Sharing and Collaboration components – Phase I. UPM. Wf4Ever Deliverable. July 2012.
- [28] D4.2v1. Design, implementation and deployment of Workflow Integrity and Authenticity Maintenance components – Phase I. ISOCO. Wf4Ever Deliverable. July 2012
- [29] H. Beedham, J. Missen, M. Palmer & R. Ruusalepp, Assessment of UKDA and TNA compliance with OAIS and METS standards, Colchester: UK Data Archive, 2005. Retrieved March 31, 2006, from: http://www.jisc.ac.uk/uploaded_documents/oaismets.pdf.
- [30] Buildhive: <https://buildhive.cloudbees.com>

Appendix A: Wf4Ever Architecture Development Methodology

In developing the Architecture and its realization through Wf4Ever Toolkit, we have embraced a co-evolutionary development model where requirements at all levels can change as implementation progresses iteratively; and we balance the demand of the realisation of user requirements through user-facing applications built upon the architecture against the division of functionality into components (models, APIs, services) for software engineering viability. In particular, the design of the architecture has been driven by technical scenarios and requirements which have been collected from user requirements, distilled, and categorized through user-stories (see Figure 1). Such requirements have led to the specification of an architecture that co-evolves with the implementation of prototypes and tools, enabling users to experience results from an early stage. These experiences are the base for new and updated user stories that feed in back into this continuous design cycle. The myExperiment platform along with dLibra framework provided an experience base and testbed – a “live laboratory” for deploying preservation features from the prototypes and tools.

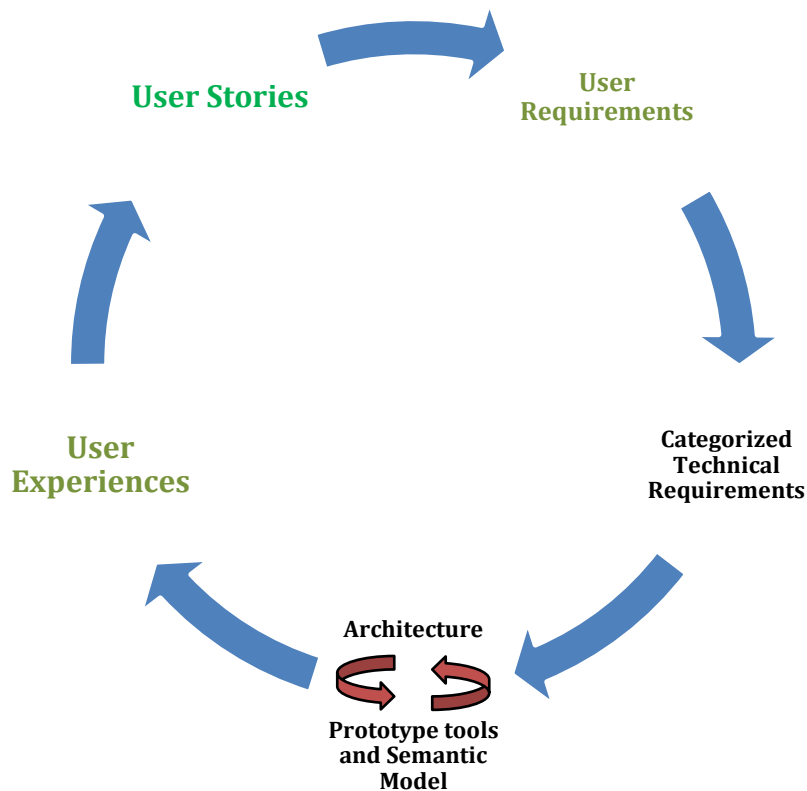


Figure 0.1 The architecture design cycle

The development process has been carried out following the plan that was specified in D1.1. We have used the SCRUM development methodology, which proposes making small iterations, based on evolving models and incoming user feedback. The development team consisted of small teams that were formed for each software component that needed to be implemented. Often, developers were also participating in other threads of work in the project, so that the software could be as closely coupled with ongoing work as possible.

Following the SCRUM methodology, the development process consisted of small phases, called *sprints*. The length of such phase depended on the size of task and number of developers involved in it. Typically, each phase consisted of the following steps:

1. Requirements specification. Depending on the type of task, requirements were specified as a wiki page cooperatively edited by interested members of the project, or as a mockup prepared by developers and reviewed by users. Whenever possible, requirements were defined as user stories.
2. Development phase. Since the teams often consisted of 1 or 2 people from the same institution, there was no need for formal meetings of the team members. For teams that were inter-institutional, development was coordinated by Skype or by emails.
3. Results deployment. A mandatory step was to publish the new code on Github. Additionally, for web applications, they were deployed on sandbox. For other types of software, installation and usage instructions were prepared.
4. Development results review. The end of each development phase was announced on the project email list so that project members, especially those representing the user communities, could review the software and send feedback. In many cases, the discussion following the end of development phase would serve as feedback for work in other work packages/task forces.

The development process was backed up by software infrastructure. In general, the following software tools have been set up for the project:

1. Git repository (source control) - <https://github.com/wf4ever>
2. FishEye (repository / code review) - <https://fisheye.man.poznan.pl/>
3. Jira (issue tracking / agile planning) - <https://jira.man.poznan.pl/jira/browse/WFE>
4. Confluence (wiki, documentation) - <http://www.wf4ever-project.org/wiki/>
5. CruiseControl (continuous integration/automatic build/test)

The Github repository has been used for storing and sharing all code that has been developed within the project. At the time of writing, it contains 34 repositories dedicated to software tools, their modules, software-related artifacts, and Wf4Ever-related data. The Git versioning control system (and thus Github as well) supports distributed development well, and promotes frequent code branching and merging. This feature has been used frequently in the project as it allowed improving the software independently by different team members focusing on different areas of improvement.

The Jira system was used extensively during the development. Even though a lot of development was done in 1 or 2 people teams in which an informal task distribution worked well, it was successfully used for coordinating work in sprints, in particular for showcases that were related to software development. It was also used for reporting errors or feature requests, in particular following service integration sprints. At the moment of writing, 977 issues have been reported in Jira.

The Confluence system has been used extensively by all project members. In particular, it has been used for:

1. Specifying requirements and writing user stories - all project members could easily contribute.
2. Describing development results and placing user manuals.
3. Discussing technical details between developers not necessarily directly involved in implementing the piece of software being discussed.
4. Brainstorming - creating pages with quick thoughts that may work as point of reference for further discussion.

The Fisheye tool has not been used, because the project members have found the Github and Confluence capabilities still sufficient. The CruiseControl continuous build system has been discarded in favour of Buildhive [30], a free continuous build system well integrated with the Github code repository.