Wf4Ever: Advanced Workflow Preservation Technologies for Enhanced Science

STREP FP7-ICT-2007-6 270192

Objective: ICT-2009.4.1b — “Advanced preservation scenarios”

D2.2v2 Design, implementation and deployment of workflow lifecycle management components - Phase II

Deliverable Co-ordinator: **XX**

Deliverable Co-ordinating Institution: **University of Manchester**

Other Authors: **XX**

This deliverable describes the second phase of delivery of workflow lifecycle management components. It includes a description of the Research Object Model, which facilitates interoperation between components; an initial Research Object Storage and Retrieval Service; RO Manager command line tool; and a definition of a model for workflow abstraction.

Document Identifier:	Wf4Ever/2013/D2.2v2/0.1	Date due:	July 31, 2013
Class Deliverable:	Wf4Ever FP7-ICT-2007-6 270192	Submission date:	June 1, 2013
Project start date	December 1, 2010	Version:	0.1
Project duration:	3 years	State:	Draft
		Distribution:	Public

Wf4Ever Consortium

This document is 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:

Intelligent Software Components S.A. (ISOCO) – Coordinator Edificio Testa, Avda. del Partenón 16-18, 1^o, 7^a Campo de las Naciones, 28042 Madrid Spain Contact person: Jose Manuel Gómez Pérez E-mail address: jmgomez@isoco.com	**University of Manchester (UNIMAN)** School of Computer Science Oxford Road, Manchester M13 9PL United Kingdom Contact person: Carole Goble E-mail address: carole.goble@manchester.ac.uk
Universidad Politécnica de Madrid (UPM) Departamento de Inteligencia Artificial, Facultad de Informática. 28660 Boadilla del Monte. Madrid Spain Contact person: Oscar Corcho E-mail address: ocorcho@fi.upm.es	**Instytut Chemii Bioorganicznej PAN - Poznan Supercomputing and Netowrking Center (PSNC)** Network Services Department Ul Z. Noskowskiego 12-14 61704 Poznań Poland Contact person: Raul Palma E-mail address: rpalma@man.poznan.pl
University of Oxford (OXF) Department of Zoology South Parks Road, Oxford OX1 3PS United Kingdom Contact person: Jun Zhao, David De Roure E-mail address: jun.zhao@zoo.ox.ac.uk david.deroure@oerc.ox.ac.uk	**Instituto de Astrofísica de Andalucía (IAA)** Dpto. Astronomía Extragaláctica. Glorieta de la Astronomía s/n, 18008 Granada Spain Contact person: Lourdes Verdes-Montenegro E-mail address: lourdes@iaa.es
Leiden University Medical Centre (LUMC) Department of Human Genetics Albinusdreef 2, 2333 ZA Leiden The Netherlands Contact person: Marco Roos E-mail address: M.Roos1@uva.nl	

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- iSOCO
- OXF
- PSNC
- UNIMAN
- UPM

Change Log

Version	**Date**	**Amended by**	**Changes**
0.1	01-06-2013	Khalid Belhajjame	Initial outline
0.2	09-06-2013	Khalid Belhajjame	Initial draft of Section 2 on the RO model

Executive Summary

This deliverable describes the second phase of delivery of workflow lifecycle management components. These components are focused around the Wf4Ever Research Object Model (RO Model), which provides descriptions of workflow-centric ROs – aggregations of content. This model is used to structure and describe ROs which are then stored and manipulated by the components of the Wf4Ever Toolkit.

The RO Model provides a framework for describing aggregations of content along with annotations of the aggregated resources, a vocabulary for describing workflows, and a vocabulary for describing provenance. The model underwent few changes in the last year in the light of user comments. We provide here a summary of the new version of the RO model. We also present the components developed for creating and managing Research Objects: the Research Object Storage and Retrieval API (implemented as part of the Research Object Digital Library (RODL)) and a command line tool – the RO Manager. These components and services are also discussed in D1.2v3 (Wf4Ever Sandbox – Phase II), D1.3v1 (Wf4Ever Architecture – Phase II) and D1.4v1 (Reference Wf4Ever Implementation – Phase II).

One of the main development in the last year consists in incorporating research objects within the myExperiment environment to allow scientists who already use myExperiment to create, share and reuse research objects. We discuss the efforts that went into this task, and report on an activity that we conducted to convert all existing Taverna T2 workflows into ROs.

We present advanced management functions that we developed for abstracting and indexing workflows, with the aim of supporting the discovery and reuse of workflows. We present an ontology that we developed for abstracting workflows in terms of motifs that characterize data manipulation and transformation patterns, which we term motifs. We also report on a solution that we developed for indexing workflows based on the services (processes) that they use.

This deliverable should be read in tandem with D1.3v2 (Wf4Ever Architecture – Phase II), D1.4v2 (Reference Wf4Ever Implementation – Phase II), D1.2v3 (Wf4Ever Sandbox – Phase III), D3.2v2 (Design, implementation and deployment of Workflow Evolution, Sharing and Collaboration components – Phase II) and D4.2v2 (Design, implementation and deployment of Workflow Integrity and Authenticity Maintenance components – Phase II) in order to provide a complete picture of the state of the Wf4Ever Phase II components.

Contents

1 Introduction **7**

2 The Research Object Model **7**

2.1 RO core ontology 7

2.2 RO Extension Ontologies 8

3 Research Object Storage and Retrieval **10**

4 Research Object Manager **10**

4.1 Implementation 11

4.2 Interactions 11

5 Research Object-Enabled myExperiment **14**

6 Workflow Abstraction using Motifs **15**

6.1 Representing Motifs 15

6.2 Representing Workflows and Workflow Steps 15

7 Workflow Indexation **16**

8 Conclusions **16**

Bibliography **18**

List of Tables

List of Figures

1 RO as an ORE aggregation. 7
2 The *wfdesc* ontology. 8
3 The *wfprov* ontology. 9
4 The *roevo* ontology extending PROV-O core terms. 10
5 RO Manager sequence diagram 12
6 Sample motifs in a Taverna workflow for functional genomics. The workflow transfers data files containing proteomics data to a remote server and augments several parameters for the invocation request. Then the workflow waits for job completion and inquires about the state of the submitted warping job. Once the inquiry call is returned the results are downloaded from the remote server. 16
7 Diagram showing an overview of the class taxonomy of the motif OWL ontology. 17
8 Subset of the annotations of the Taverna workflow shown in Figure 6 using the wfdesc model. . 17

1 Introduction

2 The Research Object Model

The RO model consists of a family of ontologies organized into core and extensions, which we will present in this section.

2.1 RO core ontology

The Core RO Ontology provides the minimum terms that are essential to the specification of research objects. Specifically, it caters for two essential requirements by providing a container structure that can be used by the scientists to bundle the resources and material relevant for their investigation, and by enabling annotations of such a container, its resources, as well as the relationships between resources thereby making the research object interpretable and reusable.

To cater for the specification of aggregation structures, we built the Research Object Core Ontology upon the popular ORE vocabulary. ORE defines standards for the description and exchange of aggregations of Web resources. Figure 1 illustrates the main terms that constitute the Research Object Core Ontology, which we describe in what follows.

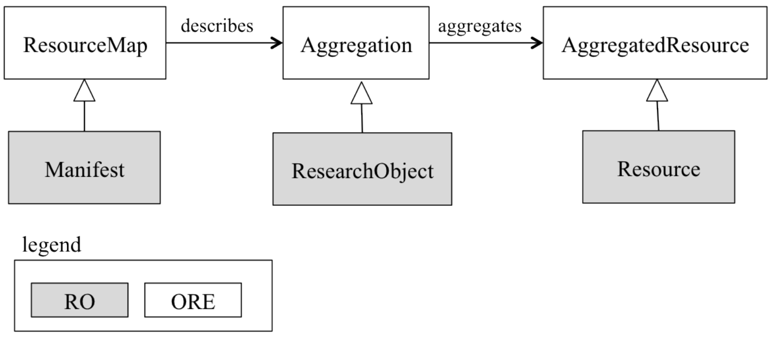


Figure 1: RO as an ORE aggregation.

- `ro:ResearchObject`[1], represents an aggregation of resources. It is a sub-class of `ore:Aggregation` and acts as an entry point to the research object.

- `ro:Resource`, represents a resource that can be aggregated within a research object and is a sub-class of `ore:AggregatedResource`. A resource can be a Dataset, Paper, Software or Annotation. Typically, a `ro:ResearchObject` aggregates multiple `ro:Resource`, and this relationship is specified using the property `ore:aggregates`.

- `ro:Manifest`, a sub-class of `ore:ResourceMap`, represents a resource that is used to describe a `ro:ResearchObject`. It plays a similar role to the manifest in a JAR or a ZIP file, and is primarily used to list the resources that are aggregated within the research object.

The second core requirement that, the Research Object Core Ontology caters for, is the descriptions of the research object and its elements. We chose the Annotation Ontology (AO) release 2.0b2 [**?**].To annotate research objects, we make use of the following three Annotation Ontology terms `ao:Annotation`[2], which represents the annotation itself; `ao:Target`, which is used to specify the `ro:Resource`(s) or `ro:ResearchObject`(s) subject to annotation; and `ao:Body`, which comprises a description of the target.

[1]The namespace of the Research Object Core Ontology ro is `http://purl.org/net/wf4ever/ro#`
[2]The namespace of ao is `http://purl.org/ao/`

In the case of research objects, we use annotations as a mean for decorating a resource (or a set of resources) with metadata information. The body is specified in the form of a set of RDF statements, which can be used to, e.g., specify the date of creation of the target or its relationship with other resources or research objects. Also, annotations can be provided for human consumption (e.g. a description of a hypothesis that is tested by a workflow-based experiment), or for machine consumption (e.g. a structured description of the provenance of results generated by a workflow run). Both kinds of annotations are accommodated using Annotation Ontology structures.

2.2 RO Extension Ontologies

We present in this section two extensions to the core Research Object ontology. The first specializes the kinds of resources that the research object can aggregate. In particular, we present extensions to specify method and experiments and the traces of their executions. The second kind of extension shows how specific metadata information, specifying the evolution of the research object over time, can be specified by specializing the Research Object core ontology.

Specifying Workflows To describe workflow research objects the workflow description vocabulary *wfdesc*[3] defines several specific resources that are involved in a workflow specification. The choice of these resources was performed by examining the commonalities between major data driven workflows, namely Taverna[4], Wings[5] and Galaxy[6], to cite a few.

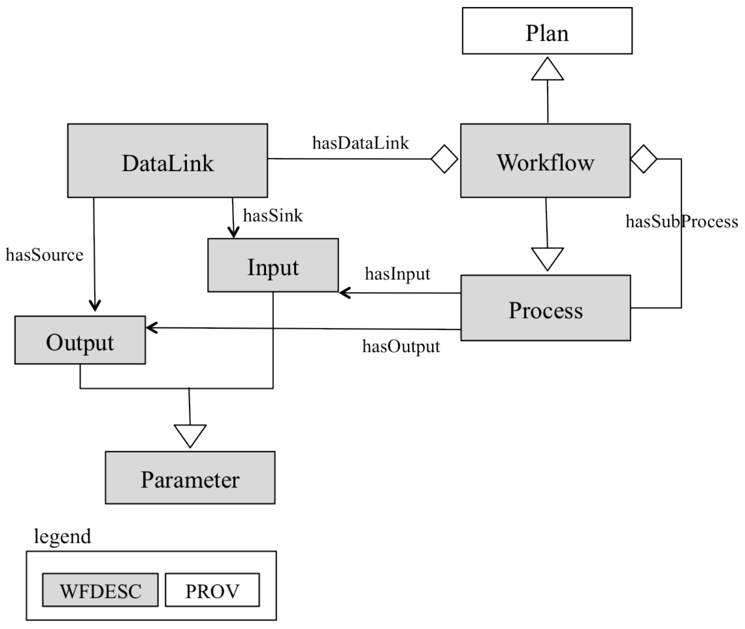


Figure 2: The *wfdesc* ontology.

Figure 2 illustrates the terms that compose the *wfdesc* ontology. Using such ontology, a workflow is described using the following three main terms:

- `wfdesc:Workflow` refers to a network in which the nodes are processes and the edges represent data links. It is defined as a subclass of the *Plan* concept from the PROV-O ontology, which represents a set of actions or steps intended by one or more agents to achieve some goals [**?**].

[3]The name space of *wfdesc* is `http://purl.org/wf4ever/wfdesc#`.
[4]http://www.taverna.org.uk
[5]http://http://wings-workflows.org
[6]http://galaxyproject.org

- `wfdesc:Process` is used to describe a class of actions that when enacted give rise to process runs. Processes specify the software component (e.g., web service) responsible for undertaking those actions.
- `wfdesc:DataLink` is used to encode the data dependencies between the processes that constitute a workflow. Specifically, a data link connects the output of a given process to the input of another process, specifying that the artifacts produced by the former are used to feed the latter.

Describing Experimental Provenance using the *wfprov* Vocabulary The *wfprov* ontology is used to describe the provenance traces obtained by enacting workflows. It is defined as an extension to the ongoing W3C PROV standard ontology - PROV-O[7].

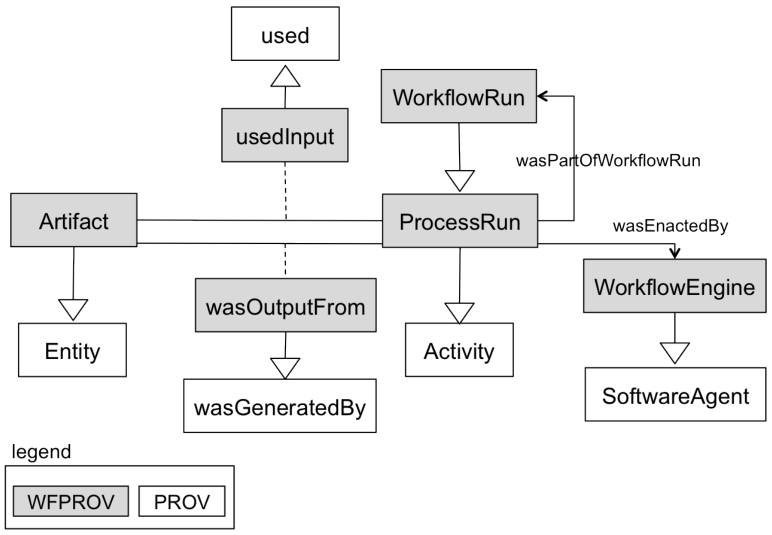


Figure 3: The *wfprov* ontology.

Figure 3 illustrates the structure of the *wfprov* ontology and its alignments with the W3C PROV-O ontology. A a workflow run (`wfprov:WorkflowRun`) represents the enactment of a given workflow. It is composed of a set of process runs (`wfprov:ProcessRun`), each representing the enactment of a process. A process run may use some artifacts (`wfprov:Artifact`) as input and generate others as output. A process run is enacted by a workflow engine (`wfprov:WorkflowEngine`), which can be seen as a PROV software agent.

By chaining the usage and generation of artifact together, the *wfprov* ontology allows scientists to trace the lineage of workflow results. For example the user can identify the input artifacts that were used to feed the wokflow run (as a whole) to obtain a given output that was generated by the workflow run.

Tracking Research Object Evolution using the *roevo* Vocabulary The *roevo* ontology is another extension to the minimal core ontology for describing an important aspect of research objects, its life cycle. To track the life cycle of a research object, we need to describe its changes at different levels of granularity, about the research object as a whole and about the individual resources. Also, we want to provide sufficient details to track the changes in order to roll back to a particular version or to quality control changes. Therefore, we need to describe when the change took place, who performed the change, and dependency relationships between the changes. Change is closely related to the provenance of a particular version of a research object or a resource. A study of the latest PROV-O ontology shows that it indeed provides all the foundational information elements for us to build the evolution ontology.

Figure 4 illustrates the core concepts of this ontology and how it extends the PROV-O:

[7]Note that the *wfprov* is reported in the W3C PROV Working Group implementation report.

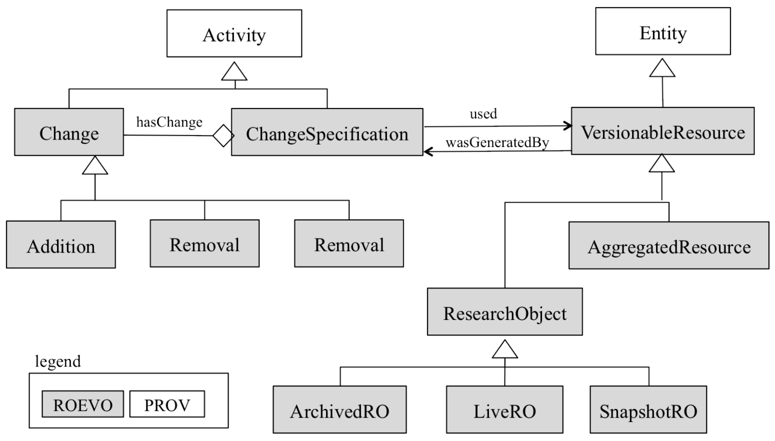


Figure 4: The *roevo* ontology extending PROV-O core terms.

- To capture different status of a research object we create three sub-classes of `ro:ResearchObject`: the `roevo:LiveRO` is a research object to capture research findings during a live investigation and it can changed, and it can either archived or snapshotted. The `roevo:ArchivedRO` can be regarded as a production research object to be preserved and archived, such as one describing findings published in an article, and it can no longer be changed; the `roevo:SnapshotRO` represents a live Research Object at a particular time.
- Both a snapshot of a live Research Object and an archived Research Object can be regarded as a versioned Research Object, i.e. a `roevo:VersionableResource`, Because it is a sub-class of `prov:Entity`, we can reuse PROV-O properties to describe the provenance or changes of this entity, such as pointing to the activity leading to any of its changes, the source research object that it was derived from, and the agent involved in its change.
- A change is a `prov:Activity`, which means that it has a start time, an end time, an input entity and a resulting entity. Also a change leading to a new Research Object can constitute a series of changes. Therefore, we have a composite `roevo:ChangeSpecification` activity, which has a number of unit `roevo:Changes`. A unit change can be adding, removing or modifying a resource or a research object. But these different changes share the same pattern of taking an input entity and producing an output entity, which can all be nicely covered by properties from PROV-O.

3 Research Object Storage and Retrieval

This section presents the components that constitute the RODL, using a UML class diagram, and show how the user can utilize RODL using a UML sequence diagram.

4 Research Object Manager

The Research Object Manager (RO Manager) is a command line tool for creating, displaying and manipulating Research Objects. The RO Manager is complementary to RODL (@@x-ref/check), in that it is primarily designed to support a user working with ROs in the host computer's local file system, with the intention being that the RODL and RO Manager can exchange ROs between them, using of the shared RO model and vocabularies. The RO Manager code base also includes the checklist evaluation functionality, described in D4.2 (@@ref), which can be invoked using a command line or REST web interface.

Experience has shown that a simple command-line tool can provide developers and users with early access to to functionality, and provide an opportunity to gather additional user feedback and requirements. RO

Manager has also been used in conjunction with built-in operating system functionality for scripting prototype tool chains for more complex operations involving Research Objects.
The RO Manager allows users and developers to:

- Create local ROs;
- Add resources to an RO;
- Add annotations to an RO;
- Read and write ROs to the RODL;
- Perform checklist evaluation of an RO;
- Obtain a raw dump of Research Object metadata.

An RO user guide is provided at http://wf4ever.github.io/ro-manager/doc/RO-manager.html[8]. An FAQ describing how to deal with various common operations using RO Manager is at http://www.wf4ever-project.org/wiki/display/docs/RO+Manager+FAQ[9].

4.1 Implementation

RO Manager is implemented in Python, and is available as an installable package through the Python Package Index (PyPI) at https://pypi.python.org/pypi/ro-manager[10].
The source code is maintained in the Github project at https://github.com/wf4ever/ro-manager[11].

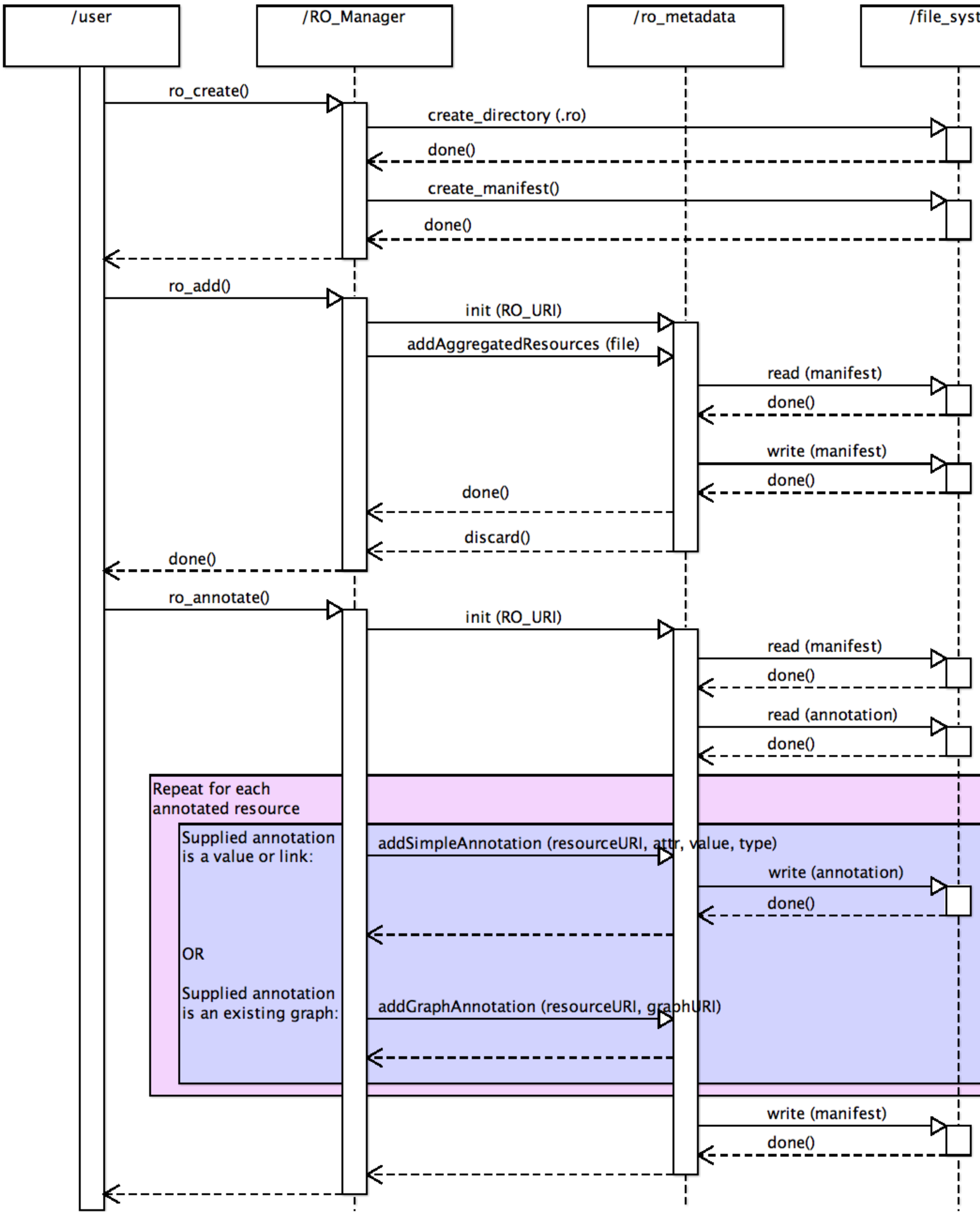
4.2 Interactions

Interactions between a user, RO Manager and the host file system substantially this pattern:
Figure 21 shows interactions for three typical RO Manager operations, `ro create`, `ro add` and `ro annotate`, which exemplify typical local RO management operations. The four interacting elements presented are the user-issued command ("/user"), the RO Manager program ("/RO_Manager"), an internal RO metadata object ("/ro_metadata") that manages the RO aggregation and annotation metadata, and the local file system ("/file_system") where ROs are persistently stored and managed.
From this, it can be seen that:

- The `ro create` command initializes an RO structure by interacting directly with the file system.
- The "ro add" command uses the RO URI to initialize an `ro\_metadata` object, and calls its `addAggregatedResources()` method to incorporate one or more files into the RO aggregation. The `ro\_metadata` object updates the RO metadata structures in the file system through a series or read and write operations.
- The "ro annotate" command similarly uses the RO URI to initialize an `ro\_metadata` object, and reads the existing annotations from disk. New annotations may be supplied as an attribute/value or attribute/link pair in which a case a new annotation graph is created in the file system. Otherwise the new annotation may already exist as a graph. In either case, the local copy of the RO manifest is updated to record the new annotation. The annotation may be applied to multiple resources in the RO. Eventually, the updated manifest is written to the file system by the `ro\_metadata` object.

8
9
10
11

Figure 5: RO Manager sequence diagram



Source code organization:
The code base that implements the RO Manager command line tool also implements the checklist evaluation service, which is covered in D4.2 (@@ref)

- `Checklists/` - contains working data files used to develop and test checklist functionality
- `Minim/` - contains the OWL ontology descriptions of the Minim models used to describe a checklist to be evaluated by the evaluation service, and the results returned. These are described further in D4.2 (@@ref)
    - `doc/` - contains files for the RO Manager usage documentation
    - `src/` - umbrella directory for the source code for RO Manager and the checklist evaluation service. Also contains the script used to create and share an installable package for the RO Manager tool.
    - `MiscLib/` - generic support functions
        - test - unit tests
    - `iaeval/` - checklist evaluation functions, used by command line tool and evaluation web service.
        - `test/` - unit tests
    - `rocommand/` - the main RO Manager command line tool, and also libraries used to access ROs by both the command line tool and the evaluation web service
        - `test/` - unit tests, including a master test suite, TestAll.py, used for checking code prior to release.
    - `roweb/` - the checklist evaluation web service, and traffic light display: this is mainly a web front end to functionality implemented by modules in `rocommand` and `roweb`.
        - `css/` - css and related files used by the web service
        - `images/` - image files used by the web service
        - `samples/` - miscellaneous sample and work-in-progress files, not part of the working software
        - `test/` - unit tests
    - `samples/` - miscellaneous sample and work-in-progress files, not part of the working software
    - `spike/` - code experiments, not part of working software
    - `uritemplate/` - working copy of URI template expansion code from http://code.google.com/p/uri-templates/ (the package has since been distributed via PyPI, so this local working copy should no longer be needed).

Key modules
Key modules that drive the execution of RO manager and the RO web services are:

- `src/ro` - a wrapper script to run the RO Manager command.
- `src/setup.py` - Python package creation and installation script.
- `src/rocommand/ro.py` - main command line interface for RO Manager, and command dispatcher. Can be used directly, or invoked via `src/ro`.
- `src/rocommand/ro_command.py` - functions to perform each of the RO Manager commands.

- `src/rocommand/ro_manifest.py` - this is the key interface between RO Manager and the RO being constructed or examined. Some access-only RO Manager commands can be performed against remote ROs (e.g. in RODL), and logic to detect and handle access to such remote ROs is included here. This module provides the main internal API between application code and the RO, and as such provides a degree of isolation.
- `src/rocommand/ro_manifest.py` - functions for accessing and manipulating an RO manifest in a local file store.
- `src/rocommand/ro_annotation.py` - functions for creating, accessing and manipulating RO annotations in a local file store.
- `src/rocommand/ROSRS_Session.py` - implements the RO API for accessing ROs in RODL (and possibly elsewhere).
- `src/roweb/rowebservices.py` - web interface for checklist evaluation and "traffic-light" display functions.
- `src/iaeval/ro_eval_minim.py` - checklist evaluation function (cf. `evaluate`)
- `src/iaeval/ro_minim.py` - Minim checklist definition access and parsing.

Dependencies
The RO Manager is very heavily dependent on RDFLib[12], which provides RDF parsing, formatting and SPARQL Query capabilities.
The RO Web service uses the Pyramid[13] web framework, and uritemplate[14] for RFC 6570[15] template expansion.

5 Research Object-Enabled myExperiment

In this section, we describe how myExperiment [] was extended in order to cater for the sharing, publication and curation of research objects. myExperiment is a virtual research environment targeted towards collaborations for sharing and publishing workflows (and experiments). myExperiment provides mechanisms for the functionalities necessary for sharing workflows within and across multiple communities. In doing so, myExperiment adopts a social web approach, which is adapted to the need of scientist. The workflows that are shared using myExperiment do not need to be specified in a particular workflow management system. For example, we find on myExpeirment workflows that have been specified using Galaxy [], Taverna [], Kepler [] and Vistrails [].
While initially targeted towards workflows, the creators of myExperiment were aware that scientists wants to share more than just workflows and experiments. Because of this, myExperiemnt was extended to supports the sharing of artifacts known as Packs. A pack can be seen as a basic aggregation of resources, which can be workflows, but also files, presentations, papers, or links to external resources. The notion of packs have been widely adopted by scientists. At the time of writing, myExperiment had 337 packs. Just like a workflow, using myExperiment a pack can be annotated and shared.
In order to support complex forms of sharing, reuse and preservation, we have worked during the last year on incorporating the notion of research objects into the development version of myExperiment [16]. In addition to the basic aggregation supported by packs, alpha myExperiment provides the mechanisms for specifying

[12]https://github.com/RDFLib
[13]http://docs.pylonsproject.org/projects/pyramid/
[14]http://code.google.com/p/uri-templates/
[15]http://tools.ietf.org/html/rfc6570
[16]http://alpha.myexperiment.org/packs/

metadata that describes the relationships between the resources within the aggregation. Moreover, the structure and the types of the resources that compose a pack are now inline with those that have been identifying thanks to the research object model. For example, a user is able to specify that a given file within a pack specifies the hypothesis, that another file specifies the workflow run obtained by enacting a given workflow, or that a given file states the conclusions drew by the scientists after analyzing the workflow run.

6 Workflow Abstraction using Motifs

Scientific workflows have been increasingly used in the last decade as an instrument for data intensive science. Workflows serve a dual function: first, as detailed documentation of the scientific method used for an experiment (i. e. the input sources and processing steps taken for the derivation of a certain data item), and second, as re-usable, executable artifacts for data-intensive analysis. Scientific workflows are composed of a variety of data manipulation activities such as Data Movement, Data Transformation, Data Analysis and Data Visualization to serve the goals of the scientific study. The composition is done through the constructs made available by the workflow system used, and is largely shaped by the function undertaken by the workflow and the environment in which the system operates.

A major difficulty in understanding workflows is their complex nature. A workflow may contain several scientifically-significant analysis steps, combined with other Data Preparation or result delivery activities, and in different implementation styles depending on the environment and context in which the workflow is executed. This difficulty in understanding stands in the way of reusing workflows.

As a first step towards addressing this issue [**?**] describes a catalogue of domain independent conceptual abstractions for workflow steps called scientific Workflow Motifs. The catalogue was built based on an empirical analysis performed over 260 workflow descriptions from Taverna [**?**], Wings [**?**], Galaxy [**?**] and Vistrails [**?**]. Motifs are provided through i) a characterization of the kinds of data-oriented activities that are carried out within workflows, which are referred to as Data-Operation motifs, and ii) a characterization of the different manners in which those activity motifs are realized/implemented within workflows, referred to as Workflow-Oriented motifs. Figure **??** shows an example of a Taverna workflow with its motifs highlighted.

This section describes the Workflow Motifs ontology[17], an OWL 2 encoding ot the aforementioned motif catalogue. The goal of this ontology is to provide the means to annotate workflows and their steps with the motifs of the vocabulary, without setting any restriction on how the workflows are defined themselves.

6.1 Representing Motifs

Figure 7 shows an overview of the class taxonomy of the ontology. The class `Motif` represents the different classes of motifs identified in the catalog. This class is categorized into two specialized sub-classes `DataOperationMotif` and `WorkflowMotif`, which are sub-classed following the taxonomy represented in [**?**].

The ontology provides three properties to link motifs to workflow specifications and their fragments. The `hasMotif` property associates workflows and their operations with their motifs. The properties `hasDataOperationMotif` and `hasWorkflowMotif` allow annotating workflows and their steps with more specificity. These properties have no domain specified, as different workflow models may use different vocabularies for describing workflows and their parts.

6.2 Representing Workflows and Workflow Steps

Workflows may be represented with different models and vocabularies like Wfdesc [**?**], OPMW [**?**], P-Plan [**?**] or D-PROV [**?**]. While providing an abstract and consistent representation of the workflow is not a pre-requisite to the usage of the Motif ontology, we consider it a best-practice to use a model that is independent

[17]http://purl.org/net/wf-motifs

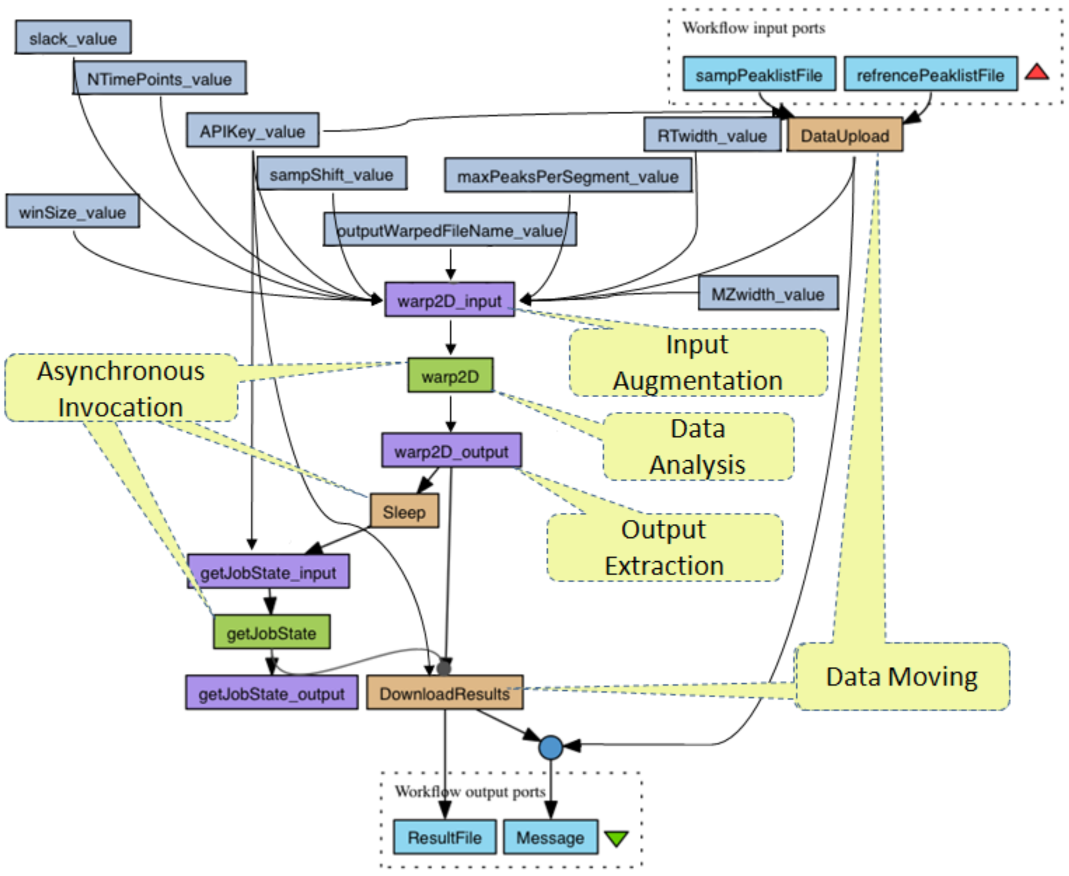


Figure 6: Sample motifs in a Taverna workflow for functional genomics. The workflow transfers data files containing proteomics data to a remote server and augments several parameters for the invocation request. Then the workflow waits for job completion and inquires about the state of the submitted warping job. Once the inquiry call is returned the results are downloaded from the remote server.

from any specific workflow language or technology. An example of annotation using the wfdesc model is given in Figure 8 by showing the annotations of part of the Taverna workflow shown in Figure 6.

The annotations encoded using the Motif Ontology could be used in a variety of applications. By providing explicit semantics on the data processing characteristic and the implementation characteristic of the operations, annotations improve understandability and interpretation. Moreover, they can be used to facilitate workflow discovery. For example, the user can issue a query to identify workflows that implement a specific flow of data manipulation and transformation (e.g., *return the workflows in which data reformatting is followed by data filtering and then data visualization*). Having information on characteristics of workflow operations allow for manipulation of workflows to generate summaries [**?**] of workflow descriptions or their execution traces.

7 Workflow Indexation

This section shows how workflows can be indexed using the trie structure. It presents the approach as well as an example workflow that is indexed.

8 Conclusions

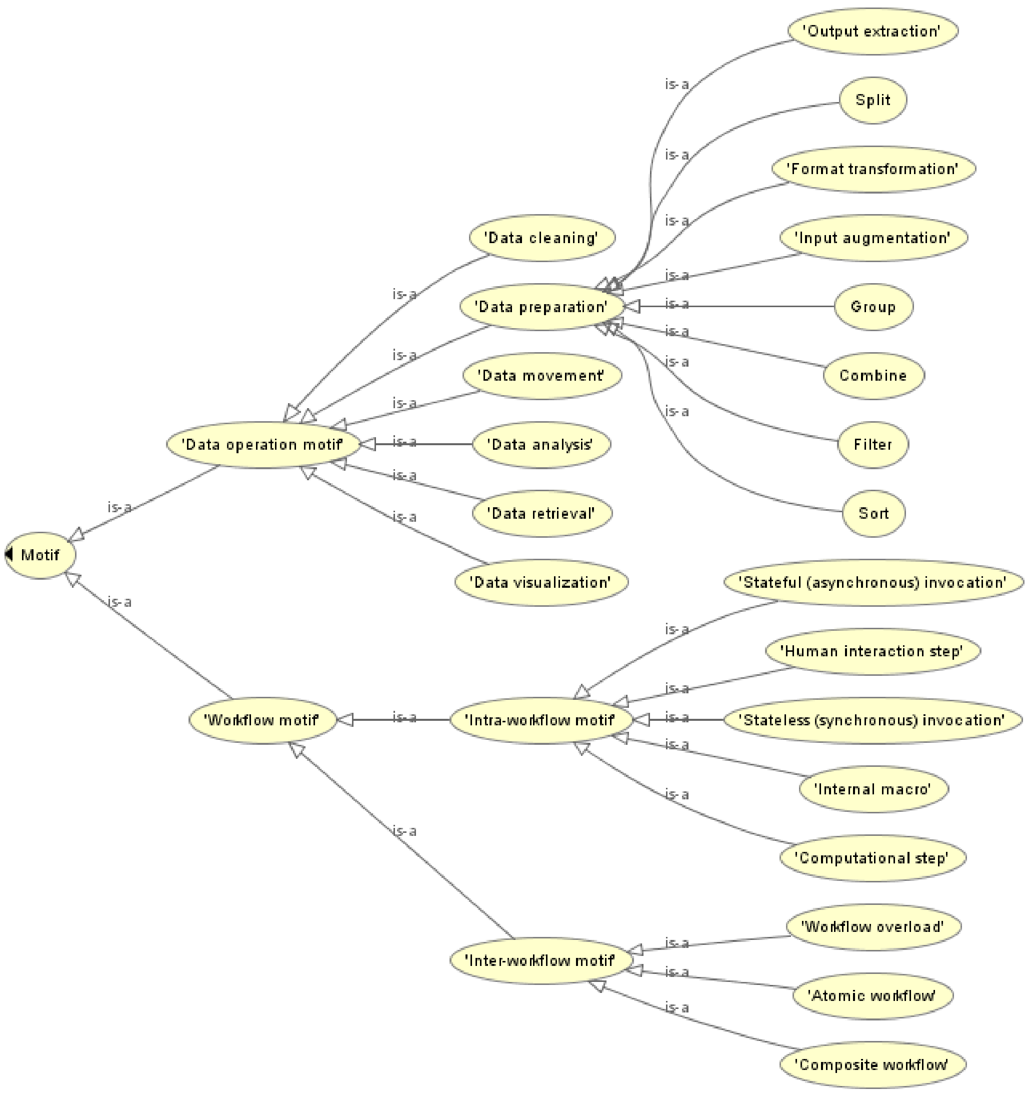


Figure 7: Diagram showing an overview of the class taxonomy of the motif OWL ontology.

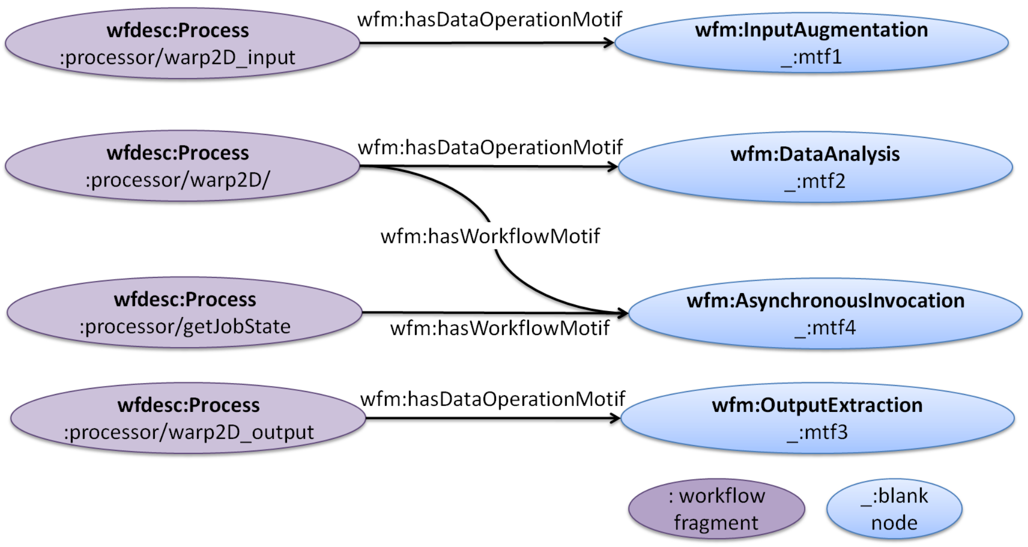

wfdesc:Process :processor/warp2D_input
wfm:hasDataOperationMotif
wfm:InputAugmentation _:mtf1
wfdesc:Process :processor/warp2D/
wfm:hasDataOperationMotif
wfm:DataAnalysis _:mtf2
wfm:hasWorkflowMotif
wfdesc:Process :processor/getJobState
wfm:hasWorkflowMotif
wfm:AsynchronousInvocation _:mtf4
wfdesc:Process :processor/warp2D_output
wfm:hasDataOperationMotif
wfm:OutputExtraction _:mtf3
: workflow fragment
_:blank node

Figure 8: Subset of the annotations of the Taverna workflow shown in Figure 6 using the wfdesc model.

References