University of Manchester
School of Computer Science
Third Year Project Report May 2017

Reproducible Research using Research Objects

Author: Mark Robinson
Degree: BSc Computer Science

Supervisor: Prof Carole Goble

Licensing

Abstract

Reproducible Research
using Research Objects

Author: Mark Robinson

Scientific workflows play a crucial role in conducting large scale scientific experiments and enabling them to be easily reproducible. However, it is vital that these are specified well with useful metadata for consumption by applications in order to make sharing them simple and convenient.
This project focuses on two up-and-coming standards - the Common Workflow Language and Research Objects - which complement each other to achieve this goal and thus aid in reproducible and transparent research.
In order to unite these standards, this report concerns the planning, development and evaluation process of a web application, 'CWL Viewer', to allow the sharing of workflows written in the Common Workflow Language. This is accomplished by providing visualisation and summary of important details as well as a downloadable Research Object Bundle designed to provide metadata relevant to workflow management and other applications.
CWL Viewer has been developed using an open source agile approach with continuous integration/deployment and is now the de facto standard for visualising workflows written in the Common Workflow Language, being used to present and document them in many organisations across a variety of fields.

Supervisor: Prof Carole Goble

Acknowledgements

I would like to thank my supervisor, Professor Carole Goble, for all of her exceptional support and feedback throughout the project.

I am also very grateful to Stian Soiland-Reyes, Technical Software Architect within the eScience Lab, for his valuable assistance during the course of my project with development and deployment.

Finally, I would like to thank Michael R. Crusoe, Peter Amstutz and everyone else within the Common Workflow Language community who provided help with my understanding of the specification and valuable feedback on the evolving state of the application.

Contents

Figures

Listings

Chapter 1

Context

This chapter introduces the concepts and some of the key technologies used in the project, how they are relevant to the idea of encouraging reproducible research and what tbe work throughout the project aims to achieve. Existing work being done within the space is also highlighted.

1.1 Key Concepts

1.1.1 Scientific Workflows

It is common in the scientific domain to require a series of computational/data management steps which can involve large scale and repetitive processes (especially in data-intensive fields such as bioinformatics, medical imaging or astronomy) [2].
This means that it would be labour intensive to run each of these steps by hand and manage conversions of input/output types between them. In addition to this issue, sharing the methodology of the experiment and allowing it to be run by others would be very difficult which is a major barrier to reproducible research [16].
Workflows have come about from a need to tackle excessive complexity in scientific analyses. They allow complex pipelines of applications to be specified in a high-level declarative way, allowing them to be easily executed [2]. A very simple example of a workflow can be seen in figure 1.1.

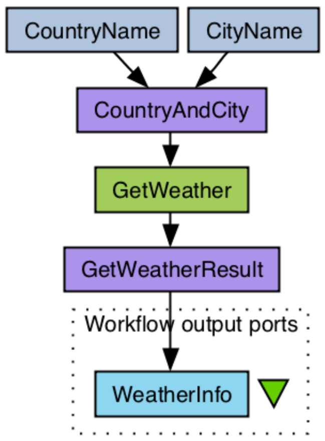


Figure 1.1: Workflow to retrieve a weather forecast for a specified city [2]

Sections or particular steps within the workflows can also be swapped out with others [19] to experiment with and compare different techniques within an overall pipeline.
However, in real world processes the workflows will be magnitudes higher in complexity such as the examples seen in appendix A, providing major challenges to execution.
Workflow management systems exist to make setting up, running and monitoring the execution of these workflows simpler. These systems provide helpful features such as graphical editors, viewing of intermediate results and setting up the environment necessary to run tools [21].
Because of the popularity of workflows, many competing standards have emerged targeting different fields. There are currently over 100 different computational data analysis workflow systems [18] which express their workflows/tooling in different formats.
As a result of this lack of interoperability, the concept of having a workflow which will ‘just run’ is undermined [25]. Users may need to download multiple different workflow systems in order to run all the workflows which they encounter. Tooling developed may also need to be specified in several languages to be used with different systems, wasting resources.
In order to run workflows, three major components are required: a language to describe them; packaging to allow tools to run in a compatible environment and tools built to aid understanding and enable execution.

1.1.2 Common Workflow Language

The need for a universal language for expressing workflows utilising open standards is clear, hence the Common Workflow Language (CWL) project has emerged [10]. This format allows for the writing of workflows and wrapping of command-line tools as a set of structured text files (YAML). Various extensions to the format are also allowed to implement extra functionality while adhering to the standard [10].
The language expresses workflows using a few main elements. These are the overall inputs and outputs as well as steps between them. In addition to this, requirements and hints provide a way of expressing the environment required to execute it, accomplishing part of the packaging goal required to execute the workflows.
An example of this is supporting the use of Docker to create a consistent environment for computation. This is a container platform used to make programs run on any machine by packaging libraries and configuration required to make the software work as needed. These containers do not contain the entire operating system like virtual machines, making for more lightweight packages which are still OS and system independent [30].
An example of a CWL workflow, `grep_and_count.cwl` from the community repository [8] can be seen in listing 1.1. This searches for a term in a collection of files using the unix command `grep` followed by finding a word count using `wc`. Steps run command line tools or subworkflows and have their own inputs and outputs to express the whole workflow.

```
class: Workflow
cwlVersion: v1.0

requirements:
 - class: ScatterFeatureRequirement

inputs:
  pattern: string
  infiles: File[]
```

```
outputs:
  outfile:
    type: File
    outputSource: wc/outfile

steps:
  grep:
    run: grep.cwl
    in:
      pattern: pattern
      infile: infiles
    scatter: infile
    out: [outfile]

  wc:
    run: wc.cwl
    in:
      infiles: grep/outfile
    out: [outfile]
```

Listing 1.1: Example Common Workflow Language Workflow Description

One of the command-line tool descriptions within this workflow (for the unix `grep` command, usage `grep 'searchterm' filename`) can be seen in listing 1.2. This defines the inputs 'pattern' being provided as the first command line argument and 'infile' as the second.

```
#!/usr/bin/env cwl-runner
class: CommandLineTool
cwlVersion: v1.0

inputs:
  pattern:
    type: string
    inputBinding: {position: 0}
  infile:
    type: File
    inputBinding: {position: 1}

outputs:
  outfile:
    type: stdout

baseCommand: grep
```

Listing 1.2: Example Common Workflow Language Command-Line Tool Description

CWL has seen increasing levels of adoption within industry, being mentioned in several Bioinformatics Open Source Conference talks [26], having a high profile in the National Cancer Institute 'Cancer Cloud' pilot [28] and being featured in the Netherlands 'National Plan Open Science' [40]. It has also been adopted by BioExcel and ELIXIR [41].

1.1.3 Research Objects

Research Objects (RO) are an emerging approach to publishing scholarly information on the web and support reuse and reproducibility [11]. They consist of a container of files with a manifest to provide meaningful information about what the those files are, what they mean, how they relate and provide provenance and versioning information [11].
One format for this container is a Research Object Bundle, which is defined by the specification as "a file format for storage and distribution of Research Objects as a ZIP archive" [36]. An example directory structure of a Research Object Bundle can be seen in listing 1.3 and a matching very simple example of the manifest with basic metadata in listing 1.4.
Research Objects use linked data in the manifest so that metadata can be expressed in a semantically rich way by consuming applications [11]. Linked Data is about using the web to connect related data using URIs, being able to understand it and allowing the data to be semantically queried to make it more useful [13]. In this case the manifest is written in JSON-LD [39].
Various helpful ontologies (formal definitions of types, properties and relationships of entities in a particular domain) already commonly exist for fields which use scientific workflows, such as the EDAM ontology for Bioinformatics [27]. This means vocabulary terms from the domain can be used in the manifest if required.

```
bundle.zip
├── mimetype
├── .ro
│   └── manifest.json
└── files
    ├── example_file.txt
    └── example_subfolder
        └── subdirectory_file.txt
```

Listing 1.3: Example Research Object Bundle Structure

```
{
  "@context" : [ "https://w3id.org/bundle/context" ],
  "id" : "/",
  "manifest" : [ "manifest.json" ],
  "createdOn" : "2017-03-24T11:20:24.123Z",
  "createdBy" : {
    "uri" : "http://www.example.com",
    "name" : "Example Tool"
  },
  "authoredBy" : [ {
    "uri" : "https://github.com/MarkRobbo",
    "name" : "Mark Robinson"
  }, {
    "uri" : "https://github.com/stain",
    "name" : "Stian Soiland-Reyes"
  } ],
  "aggregates" : [ {
    "uri" : "/files/example_file.txt",
    "mediatype" : "text/plain",
    "createdOn" : "2017-03-24T11:20:25.379Z",
    "authoredBy" : [ {
      "uri" : "https://github.com/MarkRobbo",
      "name" : "Mark Robinson"
    } ],
    "bundledAs" : {
      "uri" : "urn:uuid:50c0773b-b244-4594-8504-5d46f6fcc474",
      "folder" : "/files/"
    }
  }, {
    "uri" : "/files/example_subfolder/subdirectory_file.txt",
    "mediatype" : "text/plain",
    "createdOn" : "2017-03-24T11:20:25.379Z",
    "authoredBy" : [ {
      "uri" : "https://github.com/stain",
      "name" : "Stian Soiland-Reyes"
    } ],
    "bundledAs" : {
      "uri" : "urn:uuid:50c0773b-b244-4594-8504-5d46f6fcc475",
      "folder" : "/files/example_subfolder/"
    }
  } ]
}
```

Listing 1.4: Example Research Object Bundle Manifest

Research Objects are very general in their bare implementation. This can be specialised by using the idea of a Research Object ‘Profile’ which defines the expectation and purpose of the object in terms of what files and metadata should be expected and what assumptions can safely be made about the contents [24] [37]. This is helpful to allow the consumption of the container by an application which may have requirements for the data it is loading.
One such profile could be a specialised workflow object which would contain the Common Workflow Language files making up the overall workflow with extra metadata specific to the concept. This could contain information such as an ‘entry point’ to the main workflow within the collection of files and its corresponding input(s) [12].
This would add information on description, provenance, versioning etc. in the form of a standard manifest which is something CWL is not designed to capture within a workflow description [10]. Together the two are a natural combination to provide both the language and packaging elements of execution and produce a self-contained packaged workflow which can easily be run and reasoned about when published online.

1.2 Expected Outcome

It was expected over the course of the project to, within a web application, produce a richly featured Common Workflow Language visualisation suite, ‘CWL Viewer’. This would graphically present and list the details of CWL workflows with their inputs, outputs and steps.
In addition to this it was important to utilise Research Objects Bundles to package the files and include attribution, versioning and dependency metadata in the manifest to allow it to be easily

shared and consumed by supporting applications.
This tool was required to operate over any CWL workflow in a Github repository and provide the latest versions of the workflows within it, with a cache for meeting performance requirements.
Other planned features of the application included a gallery of previously submitted workflows; workflow visualisation download in a range of image formats and support for nested workflows/private Github repositories.
This allows for the application to have a set of features which are relevant across the entire lifecycle of interactions with the Common Workflow Language from creation to sharing and subsequent use.
These features and challenges the various features pose are detailed below within the main objectives.

1.2.1 Objectives

Visualisations of CWL Workflows

Workflow systems are moving increasingly towards user friendly GUIs and interfaces which allow comprehension of any complex pipelines involved in a much easier fashion than the user attempting to map them out in their mind from the source code.
Common Workflow Language files are currently almost exclusively being written by hand and due to their syntax and expression in multi-file collections are not conducive to easy browsing, exchange and understanding.
Visualisation can be used to facilitate this, allowing the structure of a workflow to be easily viewed at a glance for users to quickly understand its purpose and how it works.
Being able to view a simple graph also assists with the sharing and explaining of workflows to other domain scientists who may not know the CWL syntax but still require an understanding of how they work in order to use them.
The images can also be downloaded and included in reports, presentations and external documentation to illustrate and explain their function.
Despite this, visualising workflows also poses significant challenges in implementation:

- Parsing - Before visualisation can be attempted the workflows need to be parsed from their representation in CWL over multiple files to a single data structure for processing. This is a challenge with CWL as it is designed to be easily written by hand and there is a significant amount of ambiguity in the expression of many fields to allow for this, making consuming them more difficult.

- Versioning - A challenge for providing up-to-date visualisations from workflows parsed from a Github repository is keeping these continuously updated when the contents change without fetching the contents every time (which would be a huge performance penalty).

- Nesting - Workflows can be nested within each other to provide layers of abstraction. This can quickly become very complex with large pipelines which poses challenges for non-functional performance requirements. It also poses issues for user comprehension where workflows would be too much to comprehend when viewed all at once - so there must be a trade-off between information provided and complexity.

- Interactivity - Providing this in the browser provides some challenges in being able to parse the graph from the client and compute useful information such as connectivity of nodes in an efficient manner in order to add features such as highlighting.

Downloads of CWL Workflows as Research Object Bundles

A download link for the CWL workflow in the form of a Research Object Bundle (discussed above in section 1.1.3) allows for useful metadata to be provided to users to facilitate the easy sharing of a self contained object representing the entire workflow.
However, there are also difficulties associated with the construction of this bundle. The first of which is identifying which metadata would be useful to include in the manifest and which of these will be added.
The second and most significant challenge is the complexity of calculating or obtaining this information from just the CWL files and Github metadata - as naturally the values which are interesting to include are not already easily available from the files themselves.

Gallery of CWL Workflows for Discovery

The sharing of workflows is only beneficial if people can find those which are relevant to them so this became an important consideration. Although CWL workflows could be shared today by linking to source code, there is a use case for being able to search for and locate existing work to increase reuse and avoid repeating it [20].
As a learning resource, it is also helpful for users of CWL to have a bank of workflows to look at and see different techniques being demonstrated. This is already done to some extent in the form of a community repository but contributions there must be donated under an Apache open source license [8]. This is not an appropriate method in all cases as many people may not be able to distribute or develop their workflow in this fashion.

1.2.2 Requirements

These requirements were gathered and refined over the course of the project by communicating with members of the CWL community rather than upfront, due to the possibilities of requirements change from the changing environment or needs of the user base. More details of the agile methodology can be viewed in section 2.2.
The MoSCoW method [15] has been used to prioritise requirements below in order to evaluate their importance to the project. This involves using the following priorities:

1. **M**ust have - Critical to the application in order to make it a success
2. **S**hould have - Important but not absolutely necessary to the functionality of the application
3. **C**ould have - Desirable to implement but not necessary to the functionality
4. **W**on‘t have - Not appropriate to be developed within the time span or are the least critical pieces of functionality

For “Won‘t have” requirements, see the ‘Future Enhancements’ section in the conclusion (4.2).

Functional Requirements

Type	Requirement	Priority
Input	Retrieval of workflow files from a Github repository	M
	Robust parsing of Common Workflow Language V1.0 workflow descriptions	M
	Robust parsing of Common Workflow Language draft-3 workflow descriptions	S
	Storage of workflow representations	M
Visualisation	Generate a directed acyclic graph of a workflow representation	M
	Include default values differentiated from regular steps	S
	Include nested workflows differentiated from regular steps	S
	Provide the DOT graph description language source code for the visualisation	S
	Download links for the visualisation image in various formats	M
	Panning and zooming of the visualisation on the workflow page	S
	Full screen enlargement of the visualisation on the workflow page	C
	Highlighting of nodes within the graph visualisation	C
	Selection of parent and child nodes of a selection within the visualisation	C
Download	Provide a download link to a Research Object Bundle containing the workflow files	M
	Add extra relevant metadata for attribution and provenance to the Research Object manifest	S
Reporting	Provide a unique ID to each workflow to be able to navigate to it in a URL	M
	Details of inputs, outputs, steps and Github origin repository on the workflow page	M
	Details from external files run by steps	C
Gallery	Gallery of added workflows displaying visualisation thumbnails and basic information	S
	Paginate the gallery to allow easier browsing of many workflows	C

Non-Functional Requirements

Requirement	Priority
Display, operate correctly and be usable across a variety of devices (including touch screens) and screen sizes	M
Take no longer than 15 seconds to retrieve a workflow and display a visualisation page for it in normal operation	M
Maintain a similar look and feel as the official Common Workflow Language website at `http://commonwl.org`	M

1.3 Relevant Previous Work

Both Research Objects and CWL are relatively new and thus there are not a huge number of examples of them being used. In particular the massively beneficial interaction between the two is something for which tooling does not currently exist although the concept has been explored in preliminary work by the Wf4Ever project [17].
Within CWL the amount of tooling is growing in response to interest from the scientific community, but is still sparse for some popular languages.
A few examples of relevant tooling, visualisation and sharing within the community are detailed below. However, these projects differ significantly in their requirements and overall aims to CWL Viewer. This can be seen in table 1.1.

1.3.1 Dockstore

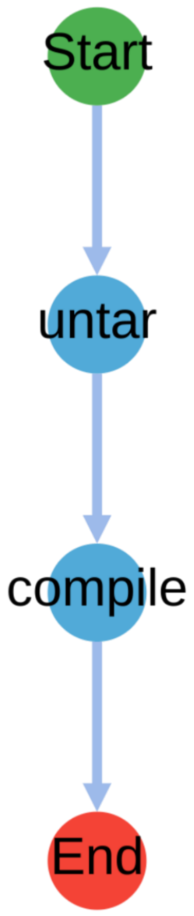
Dockstore was developed by the Cancer Genome Collaboratory and is an "open platform for sharing Docker-based tools described with the Common Workflow Language" [45] which is a specialist application for sharing of CWL workflows and tools.
However, the Dockstore visualisation (figure 1.2) does not currently take a dominant role in the application and instead gives just the most basic of details about a CWL workflow. For the purposes of viewing more information Dockstore provides the full source of tooling and the workflow. This duplicates data already available in the source repository and makes it more difficult for users to parse relevant details quickly.
The directed acyclic graph lacks inputs and outputs for the workflow. It also requires mousing over to view details of the file which is run in that step and the type of the node (tool, subworkflow etc). This encourages 'minesweeper' type behaviour of mousing over every node in order to comprehend the whole workflow. This is illustrated by figure 1.2b.

1.3.2 cwltool

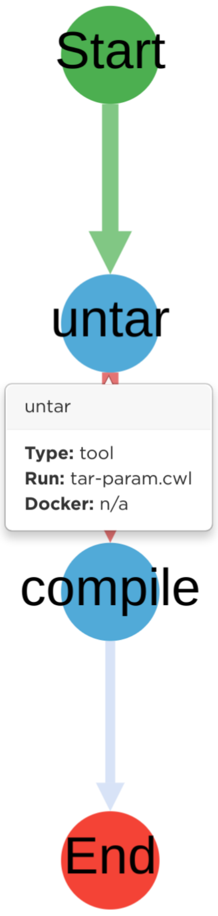
Cwltool is the reference implementation of the Common Workflow Language, providing validation and other tools to work with CWL files [10].
As part of its feature set, there is a command line option `--print-dot` which produces source code for a graph using Graphviz DOT (a popular graph description language) [22]. This can then be fed into Graphviz software to produce an image representing the workflow as a graph. An example of this output is in figure 1.3.
Despite being a feature in the reference implementation, the graph requires an external tool to render it and lacks a listing of the inputs and outputs along with how they are connected as well as default values being injected into the tools. These are aspects which are hugely important for users to understand the aim of the overall workflow.
To illustrate this, figure 1.3 can be compared with the output of the finished application with the same workflow in appendix B.2.1.

1.3.3 myExperiment

This is a website designed for researchers to share Research Objects such as Scientific Workflows [20]. It is the largest public repository of these workflows on the internet and aims to allow people to download a whole body of work with a single click [1].



(a) Default state



untar

Type: tool
Run: tar-param.cwl
Docker: n/a

(b) Mousing over the untar step

Figure 1.2: Example Dockstore graph

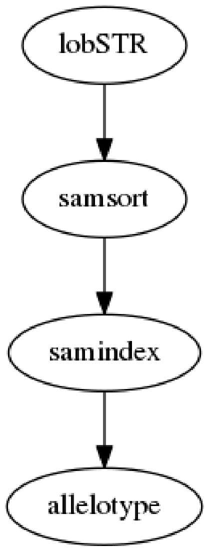
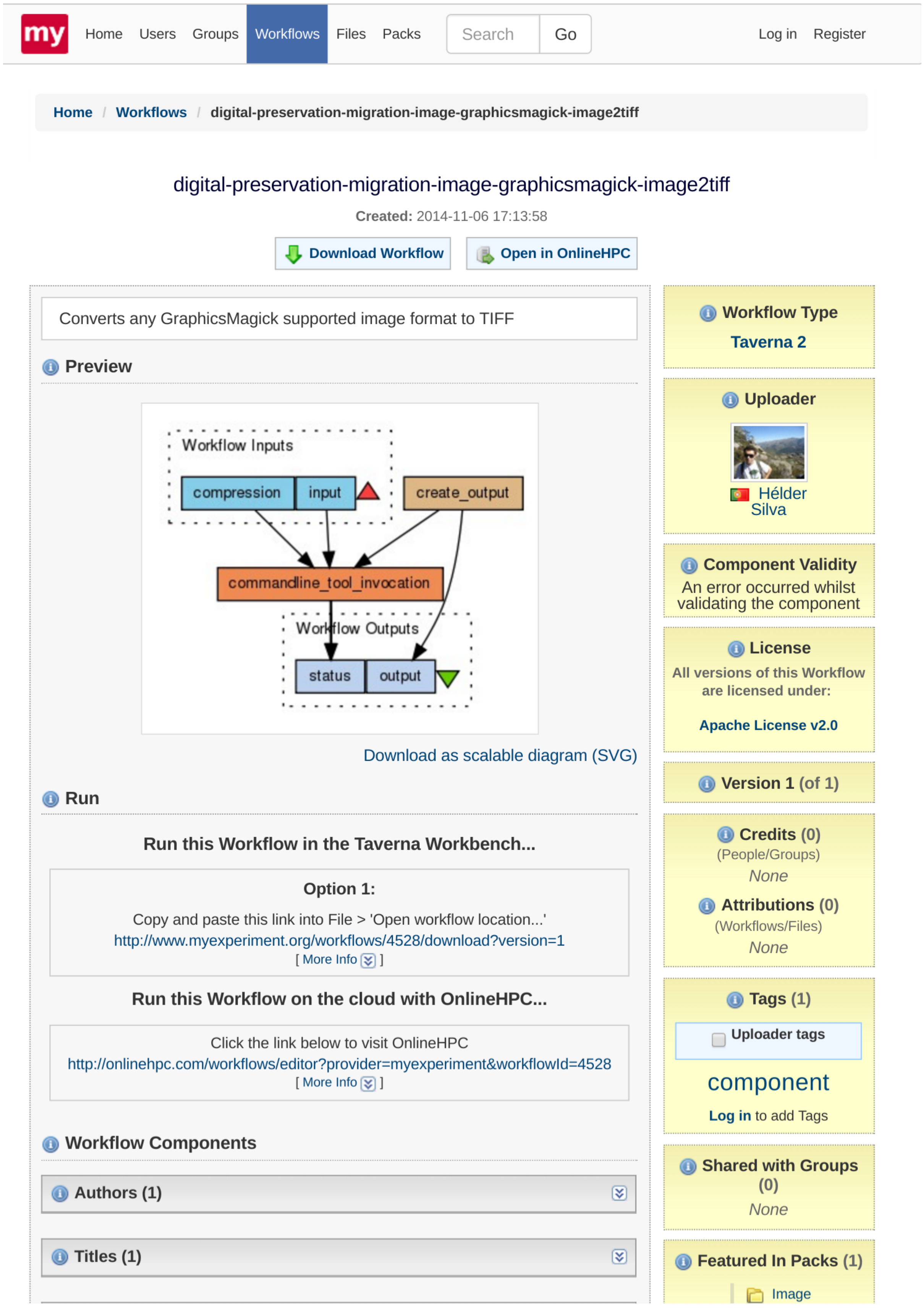


Figure 1.3: Example cwltool graph

The workflow download pages on myExperiment (figure 1.4) provide an overview, visualisation, download link and details of how to run it.
Currently CWL workflows are difficult to handle for myExperiment. This is due to it assuming a workflow is a single file and can provide its own visualisation image. CWL workflows are packaged as a collection of YAML files as explained in section 1.1.2 so do not fit with this model.

1.3.4 Taverna RO Bundle API

An API was developed to support Research Object Bundles (see section 1.1.3 above) as part of work for Apache Taverna, a workflow management system developed here at the University of Manchester [42], to support data bundles packaged within Research Objects.
This provides functionality to create and modify them as well as providing support for the basic metadata within the manifest outlined by the specification. Despite this, the functionality is quite poorly documented as it is designed for internal use within Taverna - with unit tests providing partial coverage of the intended usage.
Some elements of the specification (support for the RDF predicates `retrievedOn`, `retrievedFrom` and `retrievedBy` in the manifest) were also missing from the library at the beginning of the project and were contributed by myself to the Apache Taverna project in order to achieve my aims [34].



Home / Workflows / digital-preservation-migration-image-graphicsmagick-image2tiff

digital-preservation-migration-image-graphicsmagick-image2tiff

Created: 2014-11-06 17:13:58

Converts any GraphicsMagick supported image format to TIFF

Preview

Download as scalable diagram (SVG)

Run

Run this Workflow in the Taverna Workbench...

Option 1:

Copy and paste this link into File > 'Open workflow location...'
http://www.myexperiment.org/workflows/4528/download?version=1
[More Info]

Run this Workflow on the cloud with OnlineHPC...

Click the link below to visit OnlineHPC
http://onlinehpc.com/workflows/editor?provider=myexperiment&workflowId=4528
[More Info]

Workflow Components

Authors (1)

Titles (1)

Workflow Type
Taverna 2

Uploader
Hélder Silva

Component Validity
An error occurred whilst validating the component

License
All versions of this Workflow are licensed under:
Apache License v2.0

Version 1 (of 1)

Credits (0)
(People/Groups)
None

Attributions (0)
(Workflows/Files)
None

Tags (1)
Uploader tags
component
Log in to add Tags

Shared with Groups (0)
None

Featured In Packs (1)
Image

Figure 1.4: Example myExperiment Workflow Overview Page

Type	Requirement	Dockstore	cwltool	myEx	API
Input	Retrieval of workflow files from a Github repository	Yes	Yes	No	No
	Robust parsing of Common Workflow Language V1.0 workflow descriptions	Yes	Yes	No	No
	Robust parsing of Common Workflow Language draft-3 workflow descriptions	Yes	Yes	No	No
	Storage of workflow representations	Yes	No	No	No
Visualisation	Generate a directed acyclic graph of a workflow representation	Yes	Partial	No	No
	Include default values differentiated from regular steps	No	No	No	No
	Include nested workflows differentiated from regular steps	Partial	Yes	No	No
	Provide the DOT graph description language source code for the visualisation	No	Yes	No	No
	Download links for the visualisation image in various formats	Yes	No	No	No
	Panning and zooming of the visualisation on the workflow page	Yes	No	No	No
	Full screen enlargement of the visualisation on the workflow page	Yes	No	No	No
	Highlighting of nodes within the graph visualisation	Partial	No	No	No
	Selection of parent and child nodes of a selection within the visualisation	No	No	No	No
Download	Provide a download link to a Research Object Bundle	No	No	Yes	No
	Add extra relevant metadata for attribution and provenance to the Research Object manifest	No	No	Yes	Yes
Reporting	Provide a unique ID to each workflow to be able to navigate to it in a URL	Yes	No	Yes	No
	Details of inputs, outputs, steps and Github origin repository on the workflow page	Partial	No	Partial	No
	Details from external files run by steps	Yes	No	Yes	No
Gallery	Gallery of added workflows displaying visualisation thumbnails and basic information	Partial	No	Yes	No
	Paginate the gallery to allow easier browsing of many workflows	Yes	No	Yes	No

Table 1.1: Comparison of Functional Requirements with Capabilities of Some Existing Tools

Chapter 2

Design and Development

In this chapter the design and development process are discussed. An overview of the architecture of the application and some interesting implementation details for complex aspects are also given.

2.1 Licensing

During the course of development it was important to follow the principles of free and open source software which are demonstrated in the technologies being utilised.
For this reason, the software portion of this project is licensed under the Apache License Version 2.0 (a free software license which grants extensive rights to modify and redistribute it) and is hosted on Github under the official Common-Workflow-Language organisation at `https://github.com/common-workflow-language/cwlviewer`.
All dependencies of the application are also free and open source, having similar and compatible licenses.

2.2 Methodology

An agile methodology was embraced when working on the project code, first creating a minimum viable product of a basic visualisation along the lines of what cwltool can already produce with the added value of being on the web and the link being shareable with others.
Shortly afterwards, continuous integration was set up with Travis-CI, a tool to automatically build and test projects hosted on Github to prevent regression before merging changes. This can be seen at `https://travis-ci.org/common-workflow-language/cwlviewer`.
Finally a Docker image for the project was produced at `https://hub.docker.com/r/stain/cwlviewer` with its dependencies and continuous deployment set up to push the master branch of the project to the production server at `https://view.commonwl.org`, a subdomain of the official CWL domain name.
These two steps combined not only allowed quick and efficient changes to the code base, but also enabled the building up of a user base and method to collect continued feedback as functionality continued to be added and bugs were fixed in an iterative manner. This element of feedback is critical to an Agile process to ensure that the features which deliver the most value are being produced by the development process.

Both the Common Workflow Language Gitter chat room at `https://gitter.im/common-workflow-language/common-workflow-language` and issues added to the public Github repository linked in section 2.1 were primary methods to see which features the community wanted most so these could be prioritised.

2.3 Structure

2.3.1 Packaging

There are a total number of 24 classes which make up the application itself, packaged by feature to ensure that it is easy to find the class which corresponds to a particular change regardless of the layer of the application it belongs to.
These features are:

- Workflow - Handling of overall workflow related functionality
- Graphviz - Generation of graph files and writing of DOT source code
- ResearchObject - Generation of RO Bundles
- CWL - Classes concerning Common Workflow Language representation and parsing
- Docker - Docker related functionality
- Github - Wrappers and data types for the Github API

How these packages interact to complete the application and satisfy the requirements can be seen in figure 2.1.

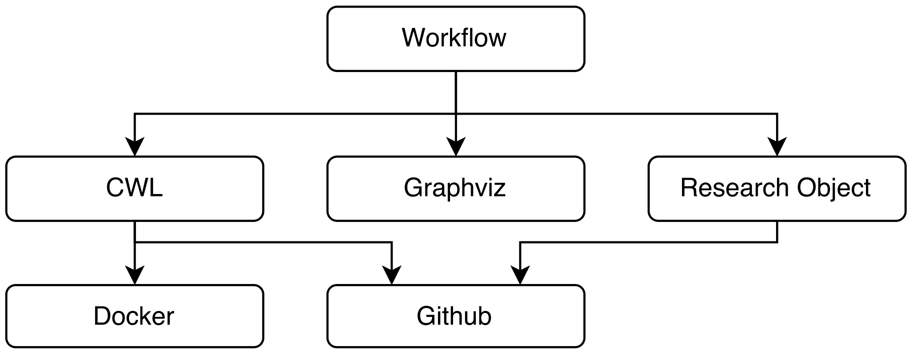


Figure 2.1: Interaction Between Packages of the Application

2.3.2 MVC Pattern

CWL Viewer uses a Model-View-Controller pattern to separate concerns of the application into:

- **Model** - This is the main component of the pattern and expresses the application's main behaviour in the problem domain. In this case it would be the combination of classes which describe a collection of files and create the workflow and DOT source code for the visualisation.

- **View** - This is any representation of the output of the application. For this application views are usually in the form of HTML templates with model information injected into them, or in the form of a file to download such as the workflow image or Research Object Bundle.

- **Controller** - This is the component which can update or retrieve details from the model. In CWL Viewer the controllers are explicitly named and defined 'Controller' classes which map particular HTTP request patterns to methods.

The overall purpose of this is to reduce model-view dependency by inserting a controller in the middle. This means the model, and possibly the view, is reusable without modification and allows for neater and clearer code in each [29].

2.4 Functionality

2.4.1 Main Application Flow

The basic application flow is pictured in figure 2.2 and described below.

1. A link to a Github file or directory is entered into the form on the main page of the application (figure 2.3).

2. If the link was a directory, a list of workflows within that directory with label and description pulled from the files is displayed. This allows the user to select the specific workflow they are interested in (figure 2.4).

3. This workflow is parsed (if it had not been already and is in the cache) and displayed on the page with a visualisation as well as variety of information about it (figure 2.5).

4. A Research Object Bundle is constructed in the background and added when complete (see section 2.5.3). Attribution for the tool as well as metadata such as author information, mime types and CWL versioning are aggregated from the Github API and included in the manifest as seen in appendix B.2.2.

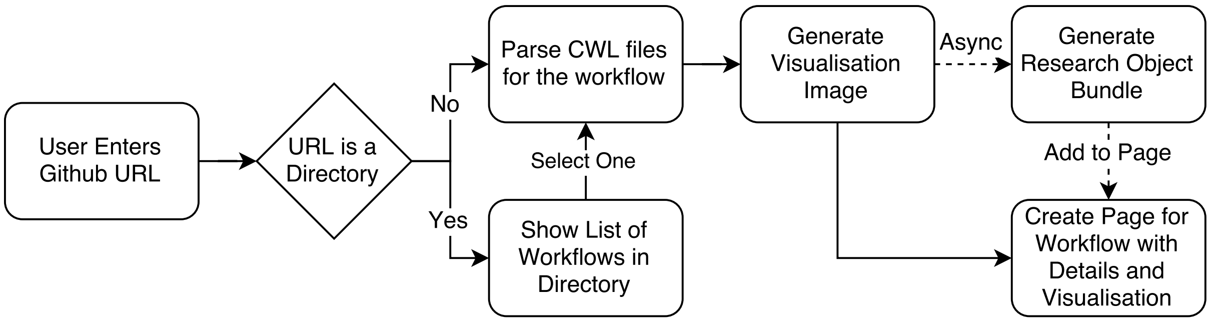
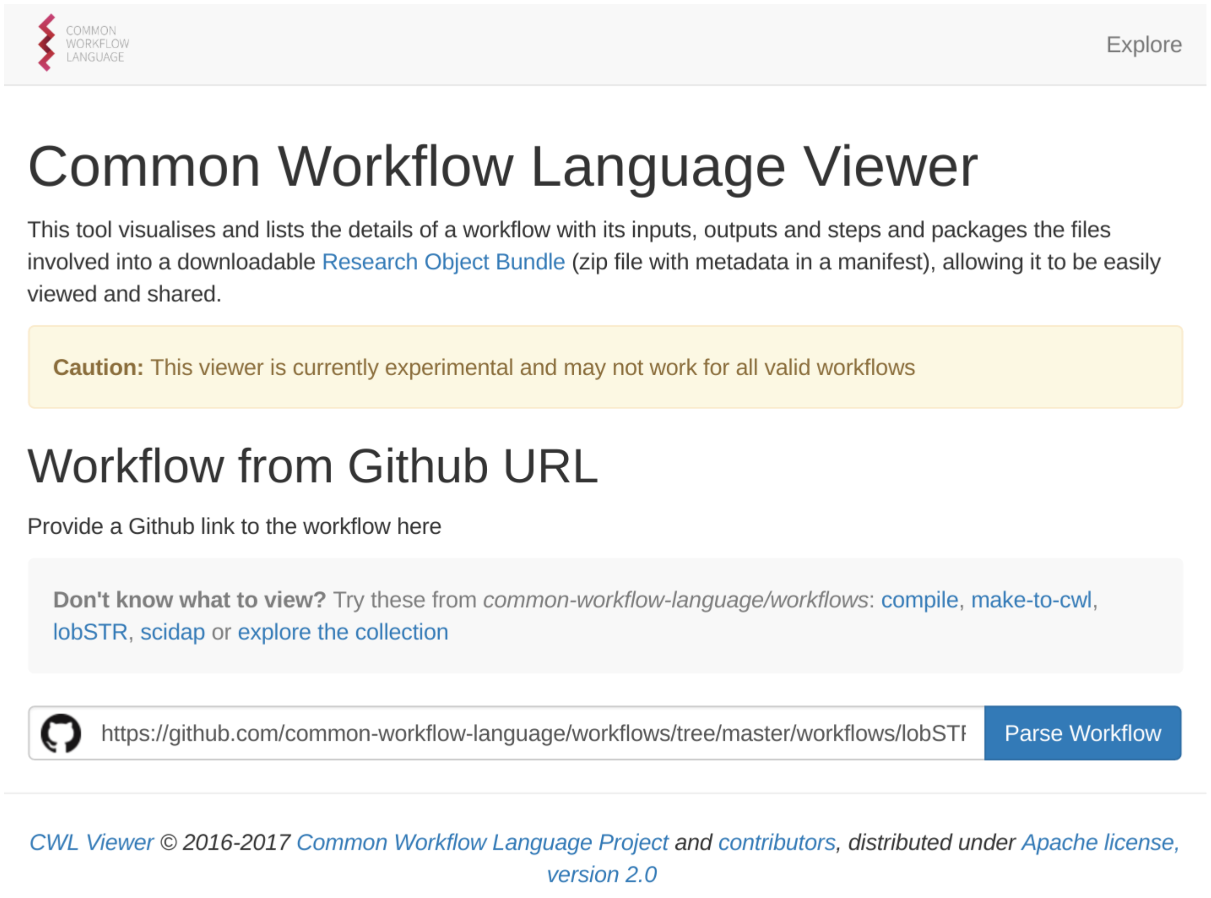

User Enters Github URL
URL is a Directory
No
Yes
Parse CWL files for the workflow
Select One
Show List of Workflows in Directory
Generate Visualisation Image
Async
Generate Research Object Bundle
Add to Page
Create Page for Workflow with Details and Visualisation

Figure 2.2: Basic Application Flow



Common Workflow Language Viewer

This tool visualises and lists the details of a workflow with its inputs, outputs and steps and packages the files involved into a downloadable Research Object Bundle (zip file with metadata in a manifest), allowing it to be easily viewed and shared.

Caution: This viewer is currently experimental and may not work for all valid workflows

Workflow from Github URL

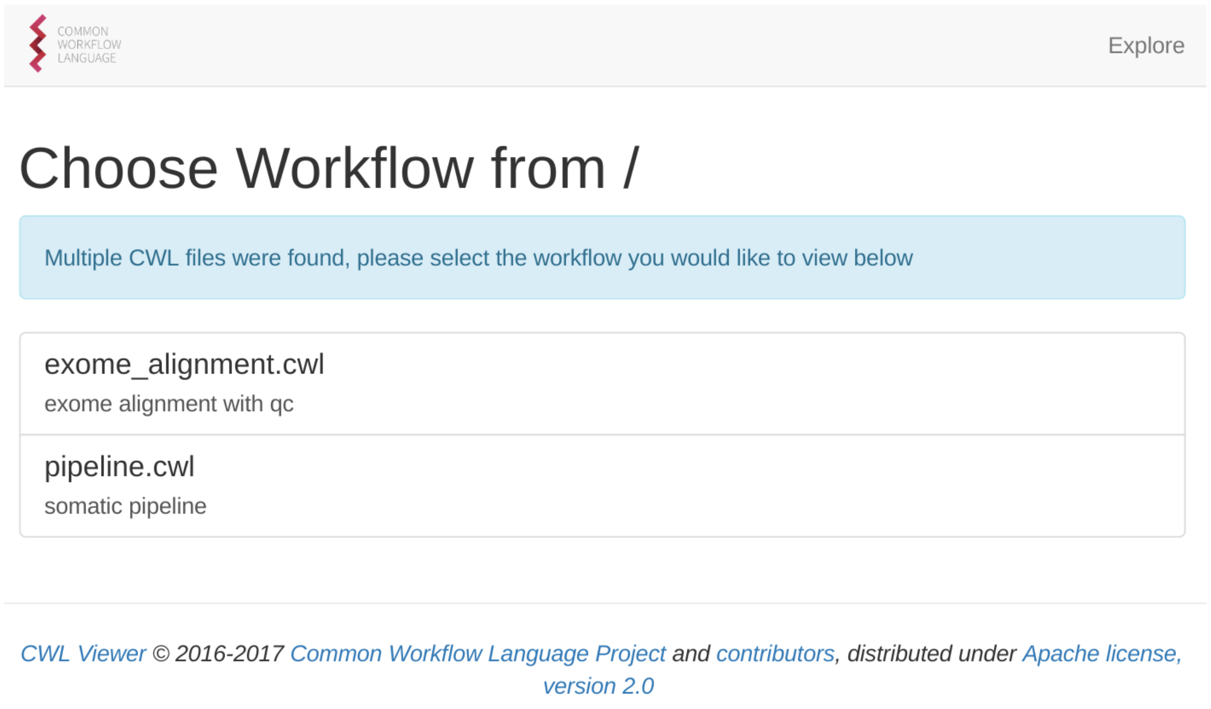
Provide a Github link to the workflow here

Don't know what to view? Try these from *common-workflow-language/workflows*: compile, make-to-cwl, lobSTR, scidap or explore the collection

https://github.com/common-workflow-language/workflows/tree/master/workflows/lobSTI

CWL Viewer © 2016-2017 Common Workflow Language Project and contributors, distributed under Apache license, version 2.0

Figure 2.3: Entering a URL into the main page



Choose Workflow from /

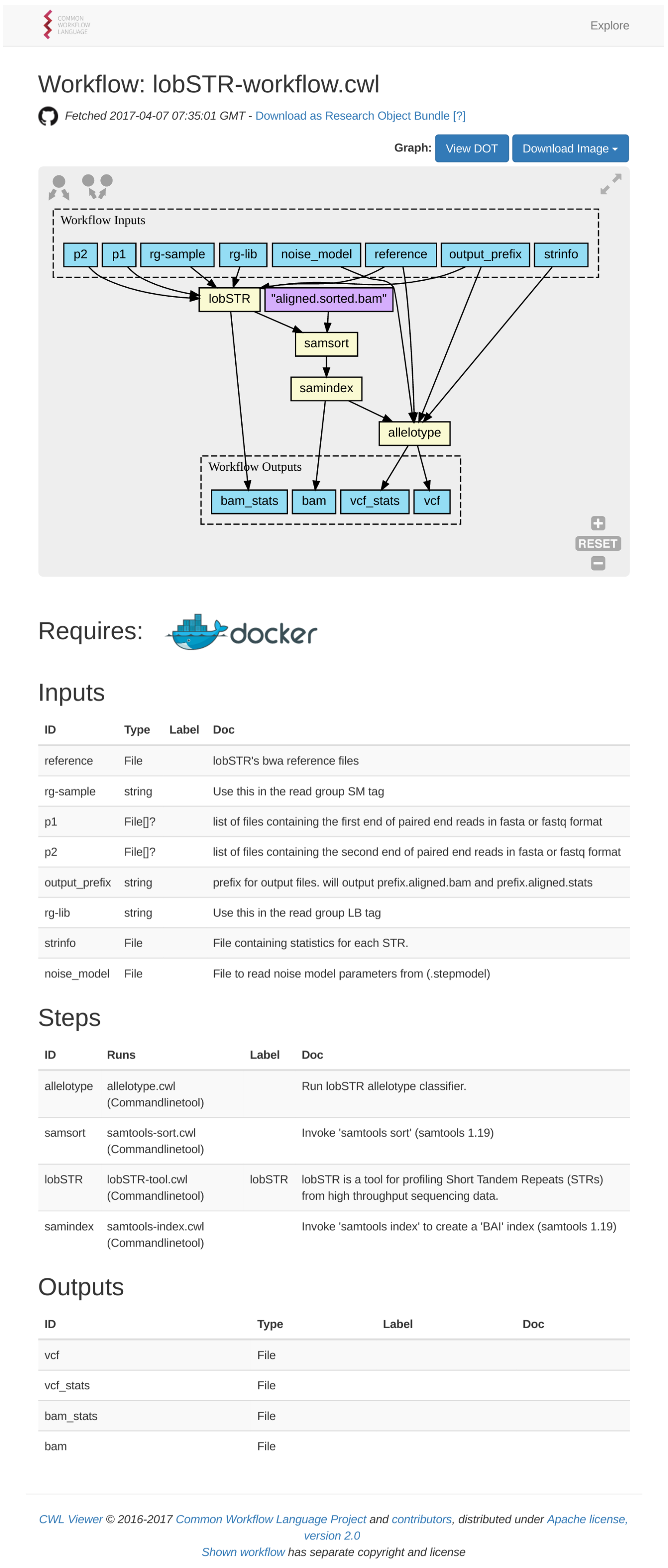
Multiple CWL files were found, please select the workflow you would like to view below

exome_alignment.cwl
exome alignment with qc

pipeline.cwl
somatic pipeline

CWL Viewer © 2016-2017 Common Workflow Language Project and contributors, distributed under Apache license, version 2.0

Figure 2.4: Selecting a workflow to view within a directory



Workflow: lobSTR-workflow.cwl

Fetched 2017-04-07 07:35:01 GMT - Download as Research Object Bundle [?]

Requires:

Inputs

ID	Type	Label	Doc
reference	File		lobSTR's bwa reference files
rg-sample	string		Use this in the read group SM tag
p1	File[]?		list of files containing the first end of paired end reads in fasta or fastq format
p2	File[]?		list of files containing the second end of paired end reads in fasta or fastq format
output_prefix	string		prefix for output files. will output prefix.aligned.bam and prefix.aligned.stats
rg-lib	string		Use this in the read group LB tag
strinfo	File		File containing statistics for each STR.
noise_model	File		File to read noise model parameters from (.stepmodel)

Steps

ID	Runs	Label	Doc
allelotype	allelotype.cwl (Commandlinetool)		Run lobSTR allelotype classifier.
samsort	samtools-sort.cwl (Commandlinetool)		Invoke 'samtools sort' (samtools 1.19)
lobSTR	lobSTR-tool.cwl (Commandlinetool)	lobSTR	lobSTR is a tool for profiling Short Tandem Repeats (STRs) from high throughput sequencing data.
samindex	samtools-index.cwl (Commandlinetool)		Invoke 'samtools index' to create a 'BAI' index (samtools 1.19)

Outputs

ID	Type	Label	Doc
vcf	File		
vcf_stats	File		
bam_stats	File		
bam	File		

CWL Viewer © 2016-2017 Common Workflow Language Project and contributors, distributed under Apache license, version 2.0
Shown workflow has separate copyright and license

Figure 2.5: Viewing a workflow

2.4.2 Gallery

Separate to this main flow above, the application is capable of listing workflows which allows for the gallery based requirements to be met. This can be seen in figure 2.6.
The list provides origin information, top level documentation and thumbnails of the graphs which is aimed to provide easy recognition of what a workflow is designed to do.
It is also paginated as per the requirements to deal with large numbers of workflows added which may be too many to load on a single page.

2.5 Challenges

In this section the areas of the requirements (1.2.2) which posed challenges or required significant design decisions during the implementation process will be elaborated upon.

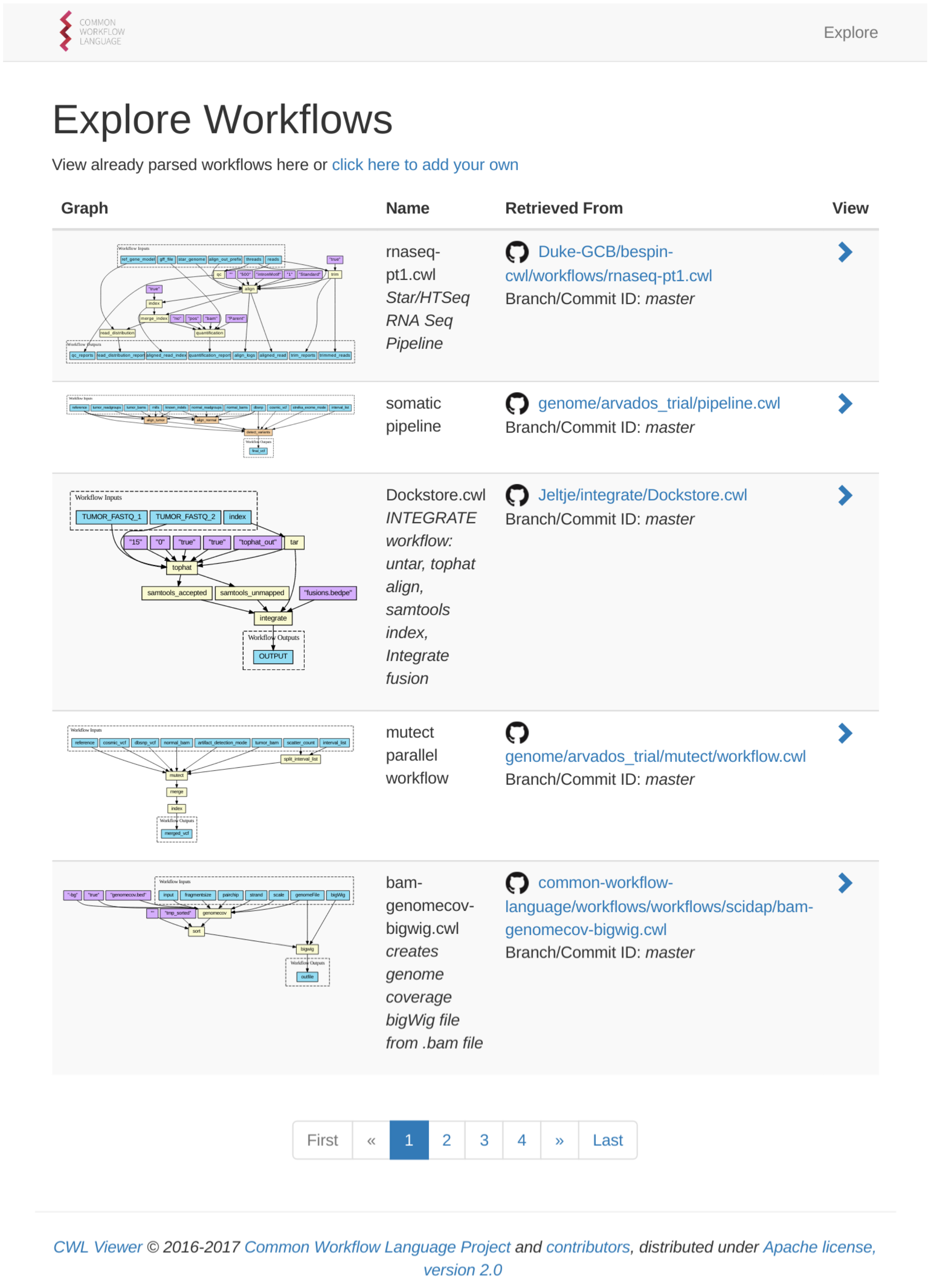
2.5.1 Input

CWL Parsing

As mentioned previously in section 1.1.2, CWL workflows exist in multi-file collections which are commonly written by hand.
With this in mind, the specification is built around making expression of the workflows in a text format simple and provides various alternative shorthands in order to make writing them easier.
However, this ambiguity complicates matters for parsing done by external tools such as CWL Viewer. This meant writing a robust parser which could handle both V1.0 and draft-3 versioned CWL in order to meet the requirements for this section was a challenge.
Handling multiple files with external tools and particularly nested workflows being involved was another added complexity to parsing.

Caching Strategy

Finding an effective and efficient caching strategy for the workflows being parsed was a critical part of developing the application. This was because the Github API has a rate limit when properly authenticated of 5000 requests per hour [3].
Repeatedly regenerating workflows when they had not changed would mean drastically increasing the number of API calls which were used and could quickly cause the website to hit that limit.
However, never checking for updates with Github would mean that changes to workflows would not be reflected in the website. It was important to support this ‘archival’ of a particular state of a workflow without forcing this on the user.
One solution to this, and the one utilised in the application, is that if the workflow was fetched from Github by a reference to a branch which could change (e.g. ‘master’), to have a time span for which it is cached. After this expires, the latest commit ID (an SHA1 hash) of the file is checked and if changes are found the workflow model is recreated.
If a workflow is fetched originally by a commit ID, the state of it is ‘frozen in time’ under this scheme as the ID does not change.



Explore Workflows

View already parsed workflows here or click here to add your own

Graph	Name	Retrieved From	View
	rnaseq-pt1.cwl *Star/HTSeq RNA Seq Pipeline*	Duke-GCB/bespin-cwl/workflows/rnaseq-pt1.cwl Branch/Commit ID: *master*	›
	somatic pipeline	genome/arvados_trial/pipeline.cwl Branch/Commit ID: *master*	›
	Dockstore.cwl *INTEGRATE workflow: untar, tophat align, samtools index, Integrate fusion*	Jeltje/integrate/Dockstore.cwl Branch/Commit ID: *master*	›
	mutect parallel workflow	genome/arvados_trial/mutect/workflow.cwl Branch/Commit ID: *master*	›
	bam-genomecov-bigwig.cwl *creates genome coverage bigWig file from .bam file*	common-workflow-language/workflows/workflows/scidap/bam-genomecov-bigwig.cwl Branch/Commit ID: *master*	›

CWL Viewer © 2016-2017 Common Workflow Language Project and contributors, distributed under Apache license, version 2.0

Figure 2.6: Viewing the paginated list of parsed workflows

2.5.2 Visualisation

Styling

The styling of the visualisation was a major consideration for the application. Emulating the style of an existing workflow management system, Apache Taverna, was considered to be the best solution.
This was due to their tried and tested styling in the area and to capitalise on any user familiarity with the system being transferable to CWL Viewer.

Interactivity

Scalable Vector Graphic (SVG) images are XML formatted images which consist of a series of scalable shapes allowing them to be increased in size without quality loss [32]. The difference between this and traditional formats is illustrated by figure 2.7

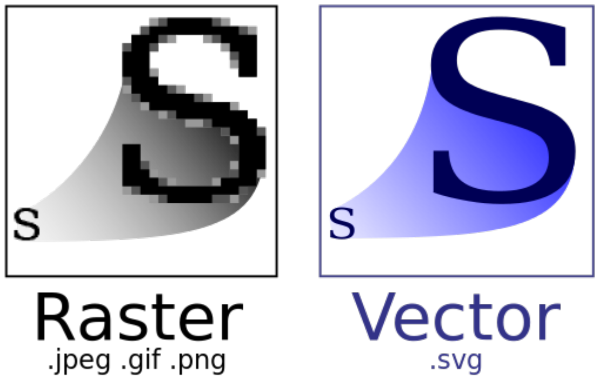


Figure 2.7: Difference between bitmap and vector images

This file format is perfect for enabling panning and zooming and is supported by Graphviz [22]. However, browsers handle the embedding of SVG files in traditional `img` or `object` tags by treating them as separate documents, which began to be an issue with the introduction of the highlighting feature described below.
This feature came out of a need to easily reference the visualisation from the table with more details below or vice versa. In addition to this, finding out the origin or resulting values from a step could be implemented using 'select parents' or 'select children' functionality after this was added. This was an important priority and ended up being included in the final application as seen in figures 2.8 and 2.9.
However it was very difficult to support by using the existing method of using an `object` tag due to the inability for jQuery [33], the main front end library used in this project, to access the SVG DOM (Document Object Model - a programming interface for HTML and XML documents [44]) [43] in order to change colouring etc.
To add this functionality a plugin had to be introduced, 'jQuery SVG', which allows interaction with the SVG DOM. Further to this the image must be inserted into the page inline by loading it from the external URL [43].
Graphviz also did not add helpful IDs to steps allowing the addition of colouring in a convenient manner, which meant it was necessary to scrape the image for information from other tags in order to understand which boxes represent which steps and how they are connected.

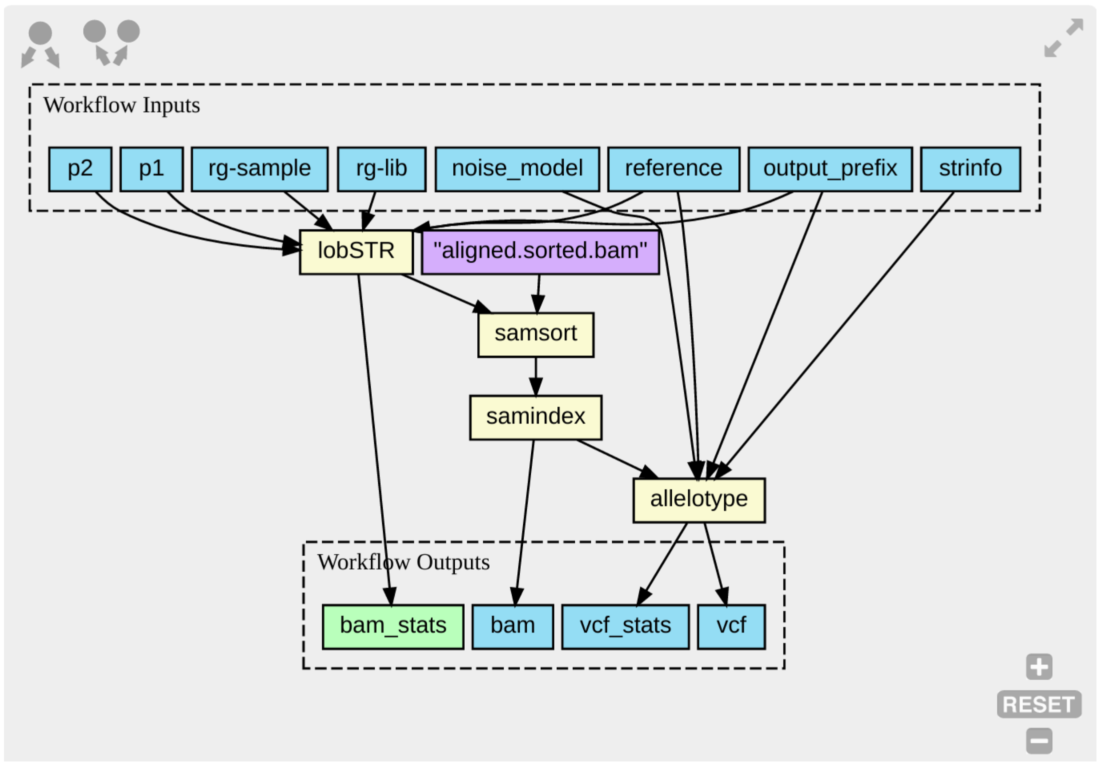


Figure 2.8: Selecting an output on the graph

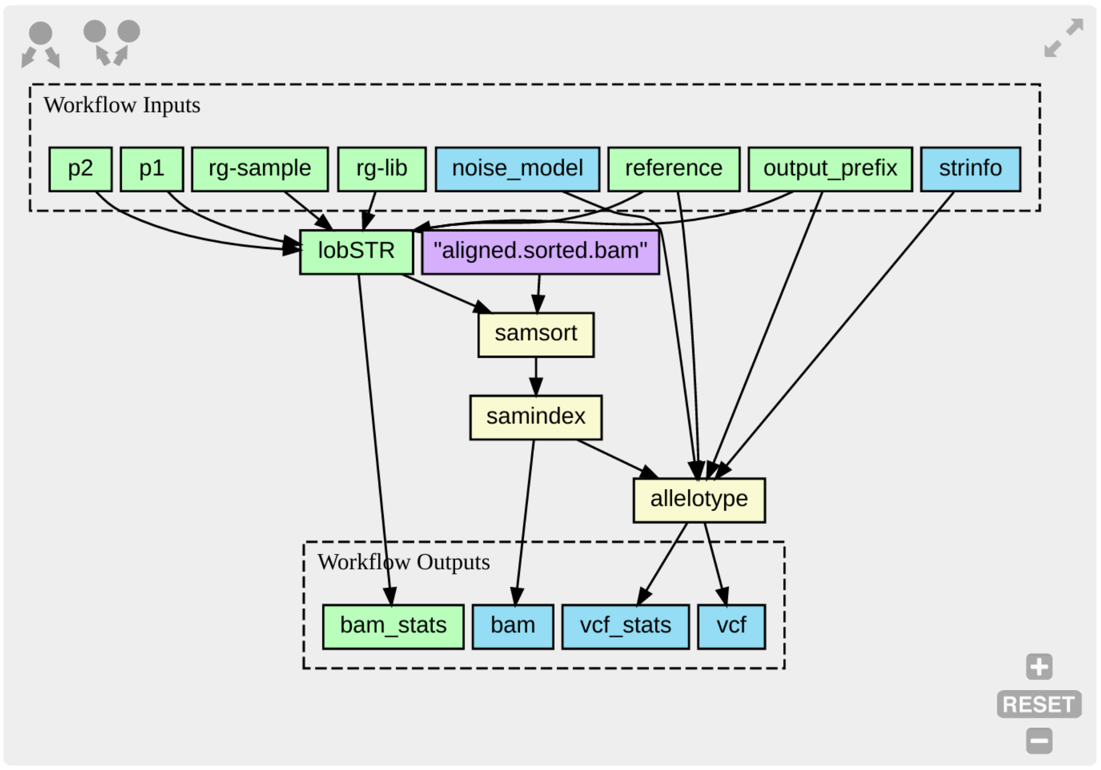


Figure 2.9: Selecting parents of a selection (figure 2.8)

This was particularly relevant when designing the selection algorithms for parents and children mentioned above where efficiency was important due to their recursive nature. For this dynamic programming techniques were used to speed up runtime for large workflows, storing the computed structure above in a helpful graph object before computation rather than attempting to scrape the DOM for each step in a recursive fashion.

Nested Workflows

Nested workflows allow for the abstraction of large processes by capturing them in a whole workflow, as a step within a much larger one.
This is a feature which is commonly utilised in the Common Workflow Language due to the aforementioned complex pipelines involved in many of the fields where people use it [8].
For this reason it was important to include this in some way in the visualisation and information listed on a workflow page. A few techniques were considered to convey this information:

1. **Including subworkflows in the graph as a cluster** - Similarly to inputs and outputs, the steps within these workflows could be represented in the same way and included within a cluster.

2. **Colouring subworkflow steps** - This indicates to the user that a particular step is a separate subworkflow.

3. **Linking subworkflows** - As the subworkflow is, in itself, a valid workflow by definition, it can be parsed separately like any other and so can be linked to within the website under the URL scheme discussed in section 2.5.4.

Method 1, including the subworkflows in the visualisation, seems at first to be a good solution. However this causes issues with complexity when subworkflows are very large or there are multiple layers of them. The resulting graph image would become complex to interpret by defeating the point of the abstraction involved in the technique in the first place.
It also adds additional complexity to the task of generating unique IDs required by the Graphviz DOT format for each step.
The user interface of Apache Taverna faced this same challenge and ended up having ‘show/hide nested workflows’ functionality which could be a possible solution. Instead, the second and third techniques above were combined and steps are coloured in the visualisation (figure 2.10) while also linking to the subworkflows in the table below on the workflow page (figure 2.11). Labels and descriptions from the nested file are also pulled in and displayed which allows for reuse of documentation (also demonstrated by figure 2.11).

2.5.3 Download

Asynchronous Research Object Generation

Page loads when processing large numbers of files associated with real world workflows can be large and focus turned to where this could be reduced in order to meet the ‘must have’ non-functional performance requirement.
This would only be relevant when the workflow is first processed and added to the database or when it needed to be updated as per the caching strategy in section 2.5.1 but the high loading times (up to 25 seconds) were still a point of concern.

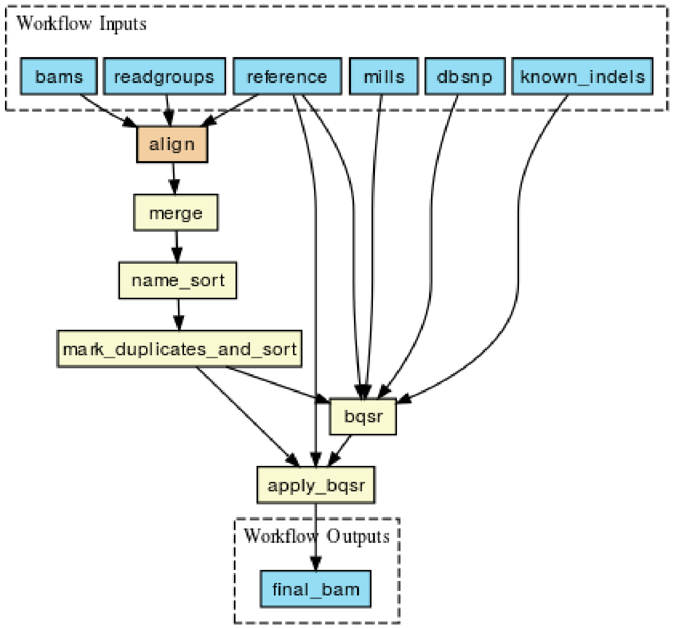
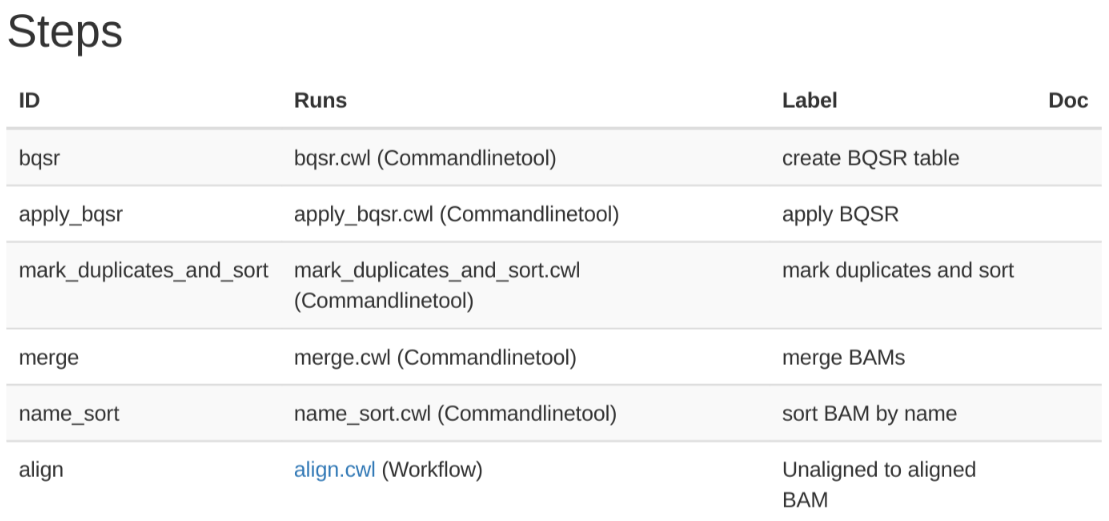


Figure 2.10: A visualisation of a workflow with nested subworkflow ('align', seen in orange)



Steps

ID	Runs	Label	Doc
bqsr	bqsr.cwl (Commandlinetool)	create BQSR table	
apply_bqsr	apply_bqsr.cwl (Commandlinetool)	apply BQSR	
mark_duplicates_and_sort	mark_duplicates_and_sort.cwl (Commandlinetool)	mark duplicates and sort	
merge	merge.cwl (Commandlinetool)	merge BAMs	
name_sort	name_sort.cwl (Commandlinetool)	sort BAM by name	
align	align.cwl (Workflow)	Unaligned to aligned BAM	

Figure 2.11: Linked subworkflow in the 'Steps' table ('align')

One of the main contributors to the time taken to process workflows was the production of the Research Object Bundle (see 1.1.3), as files need to be gathered and information from them processed in order to populate the manifest with relevant metadata.
This is something which is not necessarily immediately relevant to the user as they are expected to want to inspect the visualisation and details of a workflow before wanting to download it. It may not even be relevant at all if the user ends up not wanting to download the bundle.
For this reason and a lack of dependency on the rest of the application, the Research Object Bundle generation was made asynchronous and a repeated call added from the view to check when the bundle is generated and add it to the page (with a loading image being shown in the meantime). This reduced page load times to under half their previous values, resulting in the current performance metrics in section 3.5.

2.5.4 Reporting

Workflow ID Scheme for URLs

The workflow ID scheme to use in URLs being easily editable was a 'must have' requirement for the application. There were a few possible approaches to this.
The first of these is the simplest: using the MongoDB 'ObjectID' from the database (a 12-byte hex value as seen in listing 2.1).

```
/workflows/507f1f77bcf86cd799439011
```

Listing 2.1: Database Identifier URL Scheme

However, this scheme does not convey any information to the user about where the workflow was fetched from or details about it just from the URL. It also does not remain the same if for any reason the database is cleared and the same workflow is added.
Another option was to have the URL contain the full Github URL as illustrated in listing 2.2 and 2.3. This gives information as to the user and repository it is from as well as branch and path within that repository.

```
/workflows/github.com/{owner}/{repository name}/tree/{branch
    ref}/{path/workflow.cwl}
```

Listing 2.2: Full Github URL Scheme

```
/workflows/github.com/NCI-GDC/gdc-dnaseq-cwl/tree/master/
    workflows/dnaseq/integrity.cwl
```

Listing 2.3: Real World Example of Full Github URL Scheme

This allows the URL to be modified by the user directly to browse to a new workflow. Users can also easily paste in a Github URL they have navigated to in order to view it.
For these reasons this is the design implemented in the final version of the application as it has clear advantages over the database identifier approach.

2.6 Technologies Used

2.6.1 Languages and Storage

Java

Java was chosen as the implementation language due to the maturity of the platform, meaning there are many available frameworks and libraries available to utilise in order to achieve the goals of the project (with over 6 million in Maven repositories alone [31]), a few of which are used and listed below.
In particular, the availability of the Apache Taverna RO Bundle API for Java described in section 1.3.4 was one of the most important factors for the decision, as it is one of the few current libraries for generating Research Objects [4].

Thymeleaf

Thymeleaf is a templating engine utilized for the view of the web application. The advantage of using it over the many alternatives such as JSP is that the completed templates are valid HTML which can be viewed in a regular web browser [23].
This makes it very easy to make stylistic changes to them without needing to debug it using the full application. The templates can also be used as static prototypes before the development of the model or controller are even started.

2.6.2 MongoDB

Storage of the workflows is one of the major requirements of the system, therefore some form of database is necessary for this task.
MongoDB is an open source document 'NoSQL' database which uses JSON-like documents with a flexible schema [14]. This makes it well suited to this application of storing workflows which also have a very flexible schema with lots of optional fields.
Another large advantage of this kind of database is that documents can be sharded (separated into more easily managed parts) based on their key and so the database itself can easily be 'scaled out' by adding more server resources [14].

2.6.3 External Libraries and Frameworks

Spring Boot

The Spring Framework is an application framework for Java which contains a feature rich MVC framework providing very powerful interfaces such as 'Controller' and 'ViewResolver' with various implementations of each already provided. The methods within these are called by the framework itself when requests come in - a design principle called 'inversion of control' [7].
Spring Boot is a variation of the framework to make stand-alone, production grade Spring applications easy to set up by taking an opinionated view on configuration. It also includes an embedded Apache Tomcat web server to easily deploy the finished application [5].

Spring Data

Spring Data is a module for the Spring Framework which handles object-relational mapping - the conversion of data from a domain object to a suitable format to be stored in the database and vice versa. It also provides powerful abstractions for the database in the form of 'repositories' and queries derived from method names, meaning that no database specific code must be written [6].
This means the database layer is not tied to the rest of the application and could be swapped out with another if MongoDB is no longer suitable due to changing requirements.

SnakeYAML

As per the specification of the Common Workflow Language [10], workflow and tooling files are written in YAML, meaning the use of a parser for these files is essential.
SnakeYAML is a popular open source parser with a high level Java API which makes it well suited to this application [38].

GitHub Java API (org.eclipse.egit.github.core)

Various features of the Github API were required to gather workflows from Github, get user information and find details of commits to files. This also required support for authentication due to limitations on numbers of API calls required and enabling support for private repositories [3].
Due to this high level of usage for the API and large amount of work involved in implementing this functionality otherwise, a third party library was used and a wrapper utilised in order to convert values to those which suit the application.

Taverna RO Bundle API

As described in section 1.3.4.

Chapter 3

Evaluation

Throughout the process of development it was essential to continuously test the behaviour of the application so that issues could be found and rectified as early as possible. The methods used to do this are described in this chapter.

3.1 Unit Testing

Unit tests were an important part of assessing code quality throughout the process as loosely coupled and well designed code is naturally easy to test. They also increase confidence when making changes as a good suite of tests can be relied upon to prevent regression.
JUnit was used to create tests for each component in the system and Mockito (a library designed for producing mock objects for testing) combined with dependency injection were utilized in order to isolate the particular class which was under test. This was done to easily narrow down bugs to one area of the system for debugging.
There are currently 46 automated unit tests with a line coverage of 87%. Most of the uncovered lines of code are either getters and setters with no logic contained within them (which are unnecessary to test), are rare exceptions or are part of the Github service class which is just designed to be a wrapper for the third party API described in section 2.6.3.
The tests are run automatically when building using Travis-CI. This service checks any pull requests against the Github repository for failures and prevent merging before these tests pass to aid in continuous integration.

3.2 Integration Testing

As well as the automated tests described above, manual integration tests were used to test the full capability of the application to parse CWL files correctly and display accurate information from the workflows there. This is important in order to expose bugs in the interaction between different modules of code which unit tests would not necessarily discover.
It would also not be realistic to test visualisations automatically as they are graphical images which only have a purpose to convey information about the workflows to the user.
A mixture of 22 workflows were selected from the official Common Workflow Language conformance tests and the community repository [8] to get a representative sample with both real-world non-trivial examples and an extensive collection written to specifically test the feature set of tooling.

3.3 Cross-Browser Testing

Due to this being a web project, testing needed to be done in all major browsers to ensure that the application behaved as expected despite differences in rendering and support for technologies. This was done in the latest versions of Google Chrome, Firefox, Safari and Internet Explorer.
In addition to this, due to responsive design techniques involved in the project and support for scaling on all devices, testing was also done using Google Chrome for Android using a Nexus 6P device to ensure proper operation and display on mobile devices as well as Safari Mobile using an Apple iPad for tablet support. This ensured that the non-functional requirement of displaying, operating correctly and being usable across a variety of devices and screen sizes was met.
One important bug which was discovered during this testing was the lack of multi-touch support when scaling the visualisation, which was fixed by adding a custom event handler to implement zooming on touch devices.

3.4 Usability Testing

CWL Viewer was heavily used by the community with many workflows from organizations including the [US] National Cancer Institute, Netherlands eScience Center, Duke Center for Genomic and Computational Biology, McDonnell Genome Institute and KnowEnG being added. This resulted in a very large suite of real world workflows which were handled by the application and this was vital in testing the adherence to the Common Workflow Language specification as well as the handling of various edge cases which are utilised by people which were not necessarily thought of when the specification was designed.
In one case, the flaw was not in the application but in the workflow itself, where duplicate IDs were used. This previously worked in the reference implementation of the workflow runner 'cwltool' as they happen to be handled in different contexts[1], but is disallowed by the specification [10].
Feedback from users was also vitally important to evaluate the effectiveness of the user experience of the application as this would be very difficult to assess in an objective manner otherwise. This is especially important for this application due to the primary focal point being the workflow visualisation.
This method of formative evaluation through continual feedback meant that changes could be regularly made based on it such as the addition of example workflows to visualise on the main page and a dropdown for different image formats.
Over 50 different versions of the application were pushed to production over the course of development as part of this iterative agile approach.

3.5 Performance Testing

One of the non-functional requirements (1.2.2) was that workflow pages should load within 15 seconds in normal operation.

[1]On 2017-03-27, validation was added to cwltool to check for duplicate IDs [9]

This was evaluated by using a selection of 10 real-world workflows which were sufficiently complex to require significant work to parse, extract information and generate the visualisation. The results of this performance test with each workflow can be seen in table 3.1. The average page load time is 7.3 seconds with none being above 15 seconds in the sample, which successfully meets the requirement.

Workflow	Page Load Time (s)
`https://github.com/common-workflow-language/workflows/blob/master/workflows/lobSTR/lobSTR-workflow.cwl`	4.75
`https://github.com/common-workflow-language/workflows/blob/master/workflows/scidap/bam-genomecov-bigwig-rna-dutp.cwl`	3.77
`https://github.com/bxlab/vision-workflows/blob/master/chipseq_tf_align.cwl`	7.33
`https://github.com/ProteinsWebTeam/ebi-metagenomics-cwl/tree/ef3c7b2/tools/tRNA_selection.cwl`	10.1
`https://github.com/NCI-GDC/gdc-dnaseq-cwl/blob/master/workflows/dnaseq/integrity.cwl`	9.83
`https://github.com/nlesc-sherlock/deeplearning/blob/master/CWLworkflow/pipeline.cwl`	10.7
`https://github.com/pitagora-galaxy/cwl/blob/master/workflows/bowtie/BowtieWorkflow-se.cwl`	5.05
`https://github.com/NCBI-Hackathons/Virus_Detection_SRA/blob/master/cwl/tools/sidearm.cwl`	6.46
`https://github.com/Jeltje/integrate/blob/master/Dockstore.cwl`	6.00
`https://github.com/Duke-GCB/bespin-cwl/blob/master/workflows/rnaseq-pt1.cwl`	8.78

Table 3.1: Performance Testing Results

Chapter 4

Reflection and Consideration

This chapter reflects on the achievements of my project, what I learned during the process and future enhancements which could be made to the application to improve it.

4.1 Achievements

All the functional and non-functional requirements set out in section 1.2.2 have been met including those with 'should have' and 'could have' priorities, evaluated thoroughly by multiple forms of testing in Chapter 3.
This has resulted in a product which has features spanning the entire lifecycle of CWL workflow descriptions: allowing for easy understanding and recognition of workflows through visualisation and enabling sharing through providing the Research Object Bundle and gallery.
The application is deployed and actively used across a variety of scientific fields and by users located across the world. It is now the de facto standard for visualising Common Workflow Language workflows and is being regularly used to illustrate the contents of them in presentations and documentation. One such presentation can be seen in figure 4.1 at the Netherlands eScience Center.
Tools like CWL Viewer are a major driver for uptake and adoption of the Common Workflow Language standard, as specifications without tooling are useless for real world applications.
The application also accomplished the goal of encouraging documentation from these new users within their workflows by displaying these details on the page and highlighting the lack of label and description fields if they do not exist (seen in figure 4.2). The success of this aim is also reflected in feedback such as in figure 4.3.

4.2 Future Enhancements

With a real-world project like this there is always room for improvement and the addition of features which would be useful (or improvement and refinement of existing features).
A few of those considered to be the highest priority for the future are listed below.

4.2.1 Alternative Methods for Importing

Currently the functionality relies upon workflows being uploaded to Github. This was chosen as it is widely used by the CWL community and provides a feature rich API for avoiding

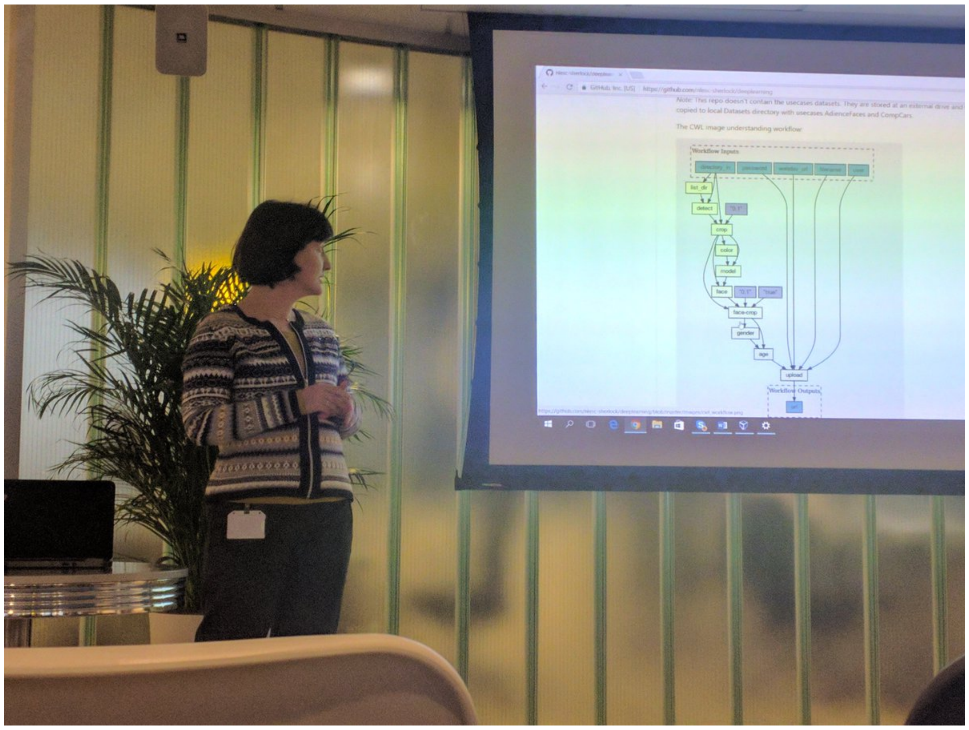
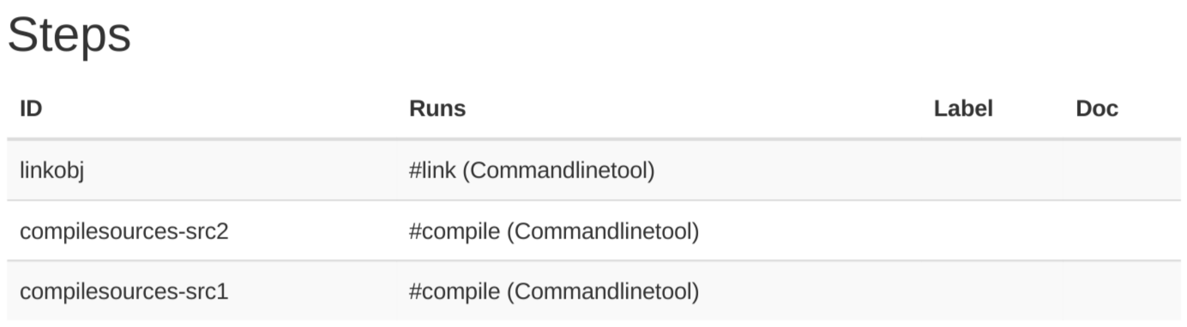


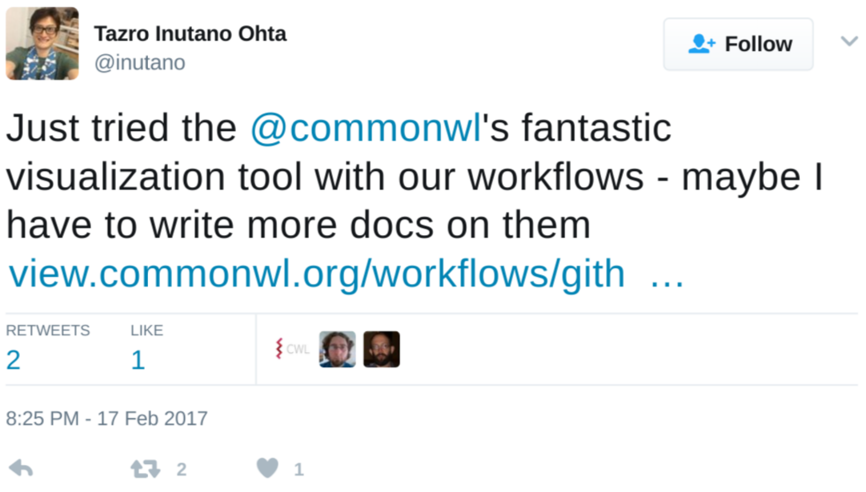
Figure 4.1: Workflow for deep learning image classification being presented using a CWL Viewer visualisation



Steps

ID	Runs	Label	Doc
linkobj	#link (Commandlinetool)		
compilesources-src2	#compile (Commandlinetool)		
compilesources-src1	#compile (Commandlinetool)		

Figure 4.2: Table of undocumented steps


Tazro Inutano Ohta
@inutano

Just tried the @commonwl's fantastic visualization tool with our workflows - maybe I have to write more docs on them view.commonwl.org/workflows/gith …

RETWEETS 2 LIKE 1

8:25 PM - 17 Feb 2017

Figure 4.3: Tweet with a testimonial

downloading the entirety of a git repository and collecting metadata for Research Objects [3]. However, many workflows are hosted in alternative locations such as on Gitlab, Bitbucket or not currently in a public location at all in which case an Research Object Bundle or general archive could be uploaded.
A generic interface for fetching workflows could be added in future in order to support these methods.

4.2.2 Command Line Tool Visualisations

So far the application has focused on workflow visualisation and not the underlying tooling behind some of the steps. However in some situations it could be useful to view details of that tooling. it is possible to display information from them in a graphical format similarly to the techniques already used for the workflows.

4.2.3 Support for Linked Data Ontologies

Linked data ontologies can be used within the Common Workflow Language, most commonly to specify formats for file types [10]. This would be valuable information to display in CWL Viewer when viewing a workflow.

4.2.4 Schema.org Support

Schema.org is a collaborative effort by some of the largest search engines to create a common set of schemas to standardise Linked Data on the web [35].
This allows them to extract semantic data from pages which can be displayed in results and is something which could be utilised by the application to provide standardised data about workflows.

4.2.5 Improved Research Object Generation

Currently, the Research Objects are constructed by including the entire contents of the containing folder. This can cause problems if tooling is external to the current folder in a repository or there are unrelated items in the same folder.
There is a difficulty associated with fixing this without imposing limitations on the structure of workflows when they are written in terms of locations of files.
Files referenced within a workflow could be detected in a ’spidering‘ approach. However, the directory structure within a Research Object Bundle would be difficult to create without arbitrarily creating folder structures and changing the content of workflow files in order to keep links intact. At this point the workflow files are no longer the same and properties in the manifest related to provenance are made semantically incorrect in many cases.
Another issue with this approach is files which relate to the workflow but are not denoted as such in any meaningful way which can be detected by the application. This includes any documentation, example output files and most of the time workflow input files (though it is possible to define default inputs using the ‘cwl:tool’ field [10]). If these are not included, the goal of the Research Object, which is to encapsulate everything which is related to the workflow and necessary to run it, is not accomplished.
Preferably a semi-automatic method of doing this could be implemented, perhaps with the help of a more rigid framework for the expression of the Research Object in the form of a profile mentioned in section 1.1.3.

4.2.6 myExperiment Integration

Due to the most popular workflow sharing platform in the space being myExperiment [1], but the limitation being the lack of container format and visualisation (as discussed in section 1.3.3), my application is a natural fit for integration with the site in order for CWL workflows to be easily shared there.
This would allow Common Workflow Language workflows to gain exposure with a much wider community and further encourage adoption of the standard.

4.3 Learnings

I have learnt a huge amount over the process of developing CWL Viewer and have enjoyed exploring the new technologies as well as exploiting them to most effectively implement the features of the application.
Prior to this project I had no experience with using Java for a web project or production application. I also had not used the Model-View-Controller pattern (section 2.3.2) in any projects despite it being very widely used and being familiar with the concepts involved.
There was also the normal learning curve associated with learning the functionality of the libraries and frameworks used, which in the case of very popular software such as the Spring Framework is highly transferable to other projects.
From an operations point of view I have also learnt a great deal, such as how to set up a dockerfile and docker compose to deploy an application with its dependencies and how to set up and use a continuous integration service to run tests for an application (and integrate this with Github).

However, the learning from my project was not only technical but also involved some understanding of fields which commonly use scientific workflows, in particular bioinformatics. This was important to understand the use cases where workflows are used and be able to provide a product which would be useful in those applications.
In addition to all of the above, working in a research software engineering environment with the eScience lab has taught me a great deal about the world of academia and the processes which are involved.

4.4 Conclusion

I have learnt a great deal over the course of my third year project and feel that I have accomplished a large amount during the process making a contribution towards assisting research efforts.
The development of the project was challenging due to the complex and somewhat subjective nature of the task in regards to the visualisations and tackling of new technologies with limited existing tooling. However, these difficulties have been overcome in order to develop an application which meets the original requirements set out at the start of the project and has users across the world utilising the application for reproducible research.

Bibliography

[1] About myExperiment. `https://www.myexperiment.org/about`. [Online; accessed 30-March-2017].

[2] Apache Taverna - why use workflows? `https://taverna.incubator.apache.org/introduction/why-use-workflows`. [Online; accessed 24-March-2017].

[3] GitHub API v3 — GitHub Developer Guide. `https://developer.github.com/v3/`. [Online; accessed 6-April-2017].

[4] Research Objects: Specifications and tooling. `http://www.researchobject.org/specifications/`. [Online; accessed 30-March-2017].

[5] Spring Boot. `https://projects.spring.io/spring-boot/`. [Online; accessed 6-April-2017].

[6] Spring Data. `https://projects.spring.io/spring-data/`. [Online; accessed 6-April-2017].

[7] Spring Framework. `https://projects.spring.io/spring-framework/`. [Online; accessed 6-April-2017].

[8] Common workflow language community workflow repository. `https://github.com/common-workflow-language/workflows`, 2015. [Online; accessed 30-March-2017].

[9] Peter Amstutz. Github issue: Validation checks for duplicate ids. `https://github.com/common-workflow-language/schema_salad/issues/56`. [Online; accessed 3-April-2017].

[10] Peter Amstutz, Michael R. Crusoe, Neboja Tijani, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Herv Mnager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common Workflow Language, v1.0. 7 2016. URL: `https://figshare.com/articles/Common_Workflow_Language_draft_3/3115156`, `doi:10.6084/m9.figshare.3115156.v2`.

[11] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, et al. Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2):599–611, 2013. `doi:10.1016/j.future.2011.08.004`.

[12] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, Oscar Corcho, José Manuel Gómez-Pérez, Sean Bechhofer, et al.

Using a suite of ontologies for preserving workflow-centric research objects. *Web Semantics: Science, Services and Agents on the World Wide Web*, 32:16–42, 2015. doi:10.1016/j.websem.2015.01.003.

[13] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, 2009. doi:10.4018/jswis.2009081901.

[14] Kristina Chodorow. *MongoDB: the definitive guide.* " O'Reilly Media, Inc.", 2013.

[15] Dai Clegg and Richard Barker. *Case method fast-track: a RAD approach.* Addison-Wesley Longman Publishing Co., Inc., 1994.

[16] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, et al. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 2017.

[17] Oscar Corcho, Daniel Garijo Verdejo, K Belhajjame, Jun Zhao, Paolo Missier, David Newman, Raul Palma, Sean Bechhofer, Esteban García Cuesta, Jose Manuel Gomez-Perez, et al. Workflow-centric research objects: First class citizens in scholarly discourse. 2012.

[18] Michael R. Crusoe, John Pellman, Peter Amstutz, Niels Drost, R. Burke Squires, Carlos Roman, Utkarsh Sengar, and Geoff Gentry. Existing workflow systems. https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems. [Online; accessed 24-March-2017].

[19] Bamdad Dashtban. *Scientific Workflow Patterns.* PhD thesis, Masters thesis, University of Manchester, 2012.

[20] David De, Roure Carole, and Goble Robert Stevens. The design and realisation of the myExperiment virtual research environment for social sharing of workflows. 2008. doi:10.1016/j.future.2008.06.010.

[21] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009. doi:doi.org/10.1016/j.future.2008.06.012.

[22] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphvizopen source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001. doi:10.1007/3-540-45848-4_57.

[23] Daniel Fernández, Emanuel Rabina, Joris Kuipers, Michal Kreuzman, sorayasl, Sunil Kumar A, Arne-Christian Blystad, Voicu Pop, Junilu Lacar, Sean Hinkley, Danny Trunk, Christopher Kluwe, Niels, Rustam Miftakhutdinov, Jose Samper, good92, James Thomson, and rynkowsw. Thymeleaf. http://www.thymeleaf.org/. [Online; accessed 6-April-2017].

[24] Matthew Gamble, Carole Goble, Graham Klyne, and Jun Zhao. Mim: A minimum information model vocabulary and framework for scientific linked data. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE, 2012. doi:10.1109/eScience.2012.6404489.

[25] Daniel Garijo, Yolanda Gil, and Oscar Corcho. Towards workflow ecosystems through semantic and standard representations. In *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science*, pages 94–104. IEEE Press, 2014. doi:10.1109/WORKS.2014.13.

[26] Nomi L Harris, Peter JA Cock, Brad Chapman, Christopher J Fields, Karsten Hokamp, Hilmar Lapp, Monica Muñoz-Torres, and Heather Wiencko. The 2016 Bioinformatics Open Source Conference (BOSC). *F1000Research*, 5, 2016. doi:10.12688/f1000research.9663.1.

[27] Jon Ison, Matúš Kalaš, Inge Jonassen, Dan Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer, and Peter Rice. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10):1325–1332, 2013. doi:10.1093/bioinformatics/btt113.

[28] Tony Kerlavage. Computational and data challenges in cancer research. https://sc15compgenome.hpc.mssm.edu/Kerlavagesc15compgenome.pdf#page=24, November 2015. [Online; accessed 24-March-2017].

[29] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[30] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[31] Fernando Rodriguez Olivera. Maven repository: Repositories. https://mvnrepository.com/repos. [Online; accessed 6-April-2017].

[32] Antoine Quint. Scalable vector graphics. *IEEE MultiMedia*, 10(3):99–102, 2003. doi:10.1109/MMUL.2003.1218261.

[33] John Resig. jQuery: The Write Less, Do More, JavaScript Library. https://jquery.com/, 2006. [Online; accessed 6-April-2017].

[34] Mark Robinson. Github pull request: Add retrievedFrom, retrievedOn and retrievedBy. https://github.com/apache/incubator-taverna-language/pull/36. [Online; accessed 30-March-2017].

[35] Jason Ronallo. HTML5 Microdata and Schema. org. *Code4Lib Journal*, 16, 2012. URL: http://journal.code4lib.org/articles/6400.

[36] Stian Soiland-Reyes, Matthew Gamble, and Robert Haines. Research Object Bundle 1.0. *Specification, researchobject.org*, 2014. URL: https://w3id.org/bundle, doi:10.5281/zenodo.12586.

[37] Stian Soiland-Reyes and Graham Klyne. Minim model for defining checklists. https://github.com/wf4ever/ro-manager/blob/master/Minim/Minim-description.md. [Online; accessed 30-March-2017].

[38] Andrey Somov. asomov / snakeyaml - Bitbucket. https://bitbucket.org/asomov/snakeyaml. [Online; accessed 6-April-2017].

[39] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD 1.0. *W3C Recommendation*, 16, 2014.

[40] WJSM van Wezenbeek, HJJ Touwen, AMC Versteeg, and AJM van Wesenbeeck. Nationaal plan open science. 2017. doi:10.4233/uuid:9e9fa82e-06c1-4d0d-9e20-5620259a6c65.

[41] Mirren White. Bioexcel Announces ELIXIR Partnership. http://bioexcel.eu/bioexcel-announces-elixir-partnership/, March 2017. [Online; accessed 26-April-2017].

[42] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, et al. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, page gkt328, 2013. doi:10.1093/nar/gkt328.

[43] Keith Wood. jQuery SVG. http://keith-wood.name/svg.html. [Online; accessed 6-April-2017].

[44] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. Document object model (DOM) level 1 specification. *W3C recommendation*, 1, 1998. URL: https://www.w3.org/TR/REC-DOM-Level-1/.

[45] Denis Yuen, Andrew Duncan, Victor Liu, Brian O'Connor, Janice Patricia, Oicr-vchung, Peter Amstutz, and The Gitter Badger. ga4gh/dockstore: 1.0, September 2016. doi:10.5281/zenodo.154185.

Appendix A

Examples of Workflow Complexity

A.1 EBI Sequencing Workflow

This visualisation is a European Bioinformatics Institute Paired-End DNA Sequencing workflow generated using CWL Viewer. Note the orange nested workflow which hides further complexity.

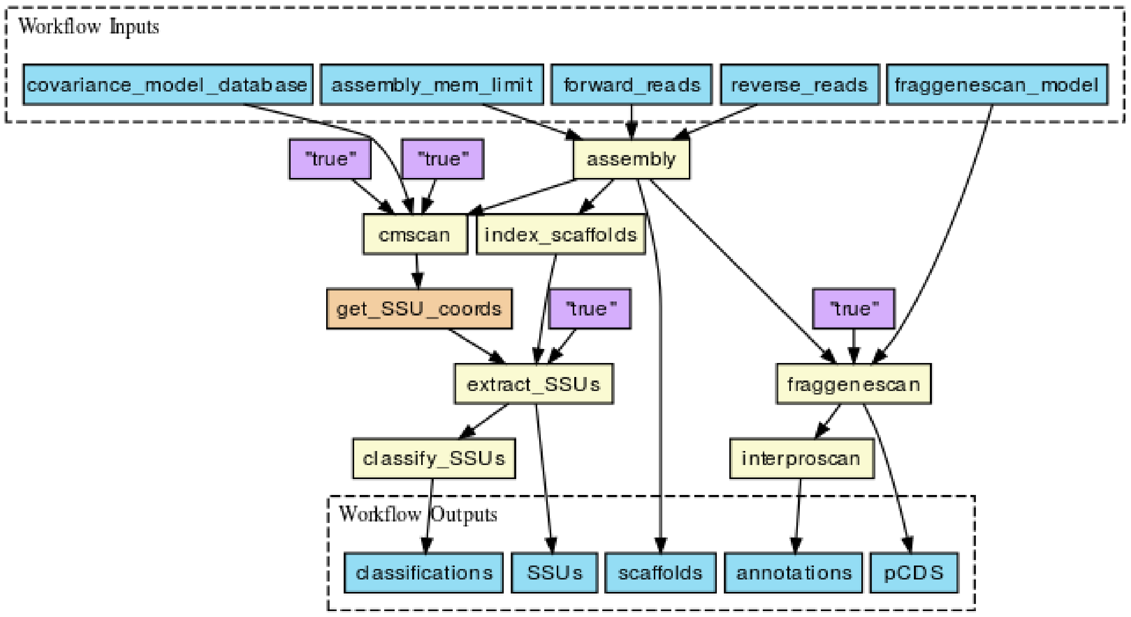


Figure A.1: EBI EMG Assembly Workflow

A.2 NCI-GDC Sequencing Error Detection Workflow

This visualisation is generated from a National Cancer Institute's Genomic Data Commons Base Quality Score Recalibration workflow - a data pre-processing step which detects systematic errors made by a sequencer.

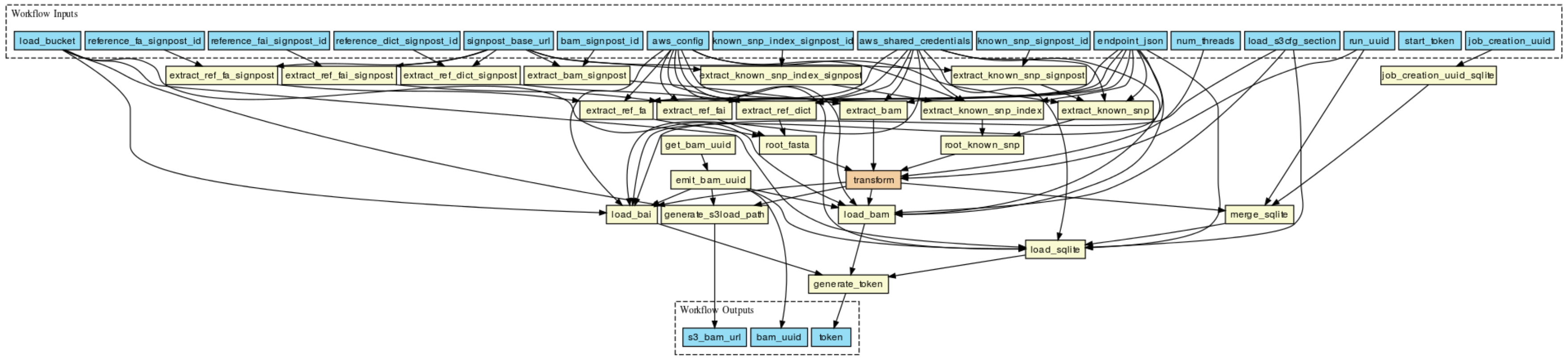


Figure A.2: NCI-GDC BQSR Workflow

Appendix B

Example Inputs and Outputs

B.1 Inputs

B.1.1 Github URL

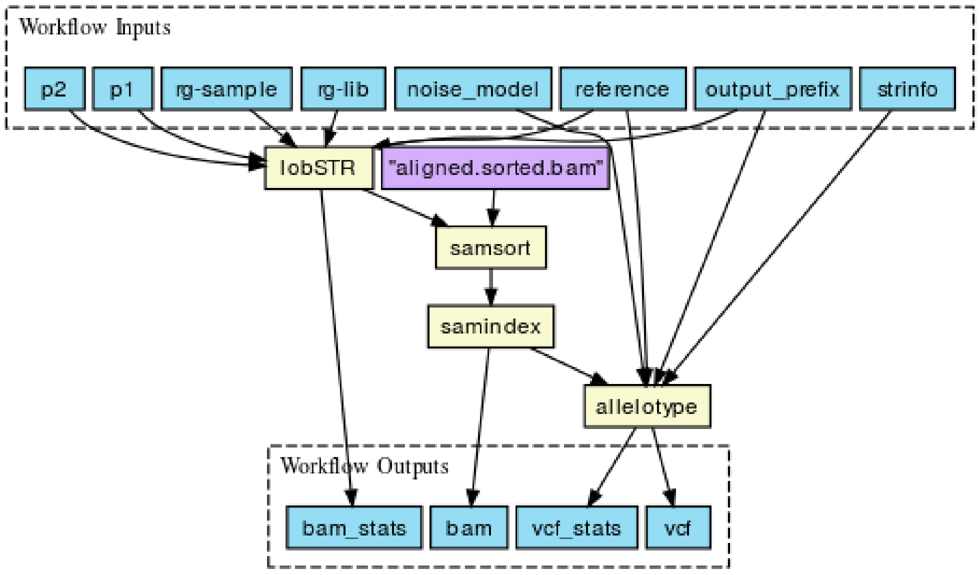
A Github URL to a workflow is entered into the main page to add it, in this case:
`https://github.com/common-workflow-language/workflows/tree/master/workflows/lobSTR/lobSTR-workflow.cwl`
This URL contains a workflow involving 4 steps and a single default value.

B.2 Outputs

B.2.1 Visualisation

The following is an example of a visualised workflow from the URL in section B.1.1 above.

Image



Graphviz DOT Source

The source of the above image is listed below

```
digraph workflow {
  graph [
    bgcolor = "#eeeeee"
    color = "black"
    fontsize = "10"
    labeljust = "left"
    clusterrank = "local"
    ranksep = "0.22"
    nodesep = "0.05"
  ]
  node [
    fontname = "Helvetica"
    fontsize = "10"
    fontcolor = "black"
    shape = "record"
    height = "0"
    width = "0"
    color = "black"
    fillcolor = "lightgoldenrodyellow"
    style = "filled"
  ];
  edge [
    fontname="Helvetica"
    fontsize="8"
    fontcolor="black"
    color="black"
    arrowsize="0.7"
  ];
  subgraph cluster_inputs {
    rank = "same";
    style = "dashed";
    label = "Workflow Inputs";
    "reference" [fillcolor="#94DDF4"];
    "rg-sample" [fillcolor="#94DDF4"];
    "p1" [fillcolor="#94DDF4"];
    "p2" [fillcolor="#94DDF4"];
    "output_prefix" [fillcolor="#94DDF4"];
    "rg-lib" [fillcolor="#94DDF4"];
    "strinfo" [fillcolor="#94DDF4"];
    "noise_model" [fillcolor="#94DDF4"];
  }
  subgraph cluster_outputs {
    rank = "same";
    style = "dashed";
    label = "Workflow Outputs";
    "vcf" [fillcolor="#94DDF4"];
    "vcf_stats" [fillcolor="#94DDF4"];
    "bam_stats" [fillcolor="#94DDF4"];
    "bam" [fillcolor="#94DDF4"];
  }
  "allelotype";
  "samsort";
  "lobSTR";
  "samindex";
  "allelotype" -> "vcf";
  "allelotype" -> "vcf_stats";
  "lobSTR" -> "bam_stats";
  "samindex" -> "bam";
  "reference" -> "allelotype";
  "output_prefix" -> "allelotype";
  "strinfo" -> "allelotype";
  "samindex" -> "allelotype";
  "noise_model" -> "allelotype";
  "default1" [label="\"aligned.sorted.bam\"", fillcolor="#
    D5AEFC"]
  "default1" -> "samsort";
  "lobSTR" -> "samsort";
  "reference" -> "lobSTR";
  "rg-sample" -> "lobSTR";
  "p1" -> "lobSTR";
  "p2" -> "lobSTR";
  "output_prefix" -> "lobSTR";
  "rg-lib" -> "lobSTR";
  "samsort" -> "samindex";
}
```

B.2.2 Research Object Bundle

Directory Structure

```
bundle.zip
├── mimetype
├── .ro
│   └── manifest.json
└── workflow
    ├── allelotype.cwl
    ├── lobSTR-arvados-demo.json
    ├── lobSTR-demo.json
    ├── lobSTR-tool.cwl
    ├── lobSTR-workflow.cwl
    ├── README
    ├── samtools-index.cwl
    ├── samtools-sort.cwl
    ├── tmp_1.fq
    ├── tmp_2.fq
    └── models
        ├── illumina_v3.pcrfree.stepmodel
        └── illumina_v3.pcrfree.stuttermodel
```

Partial Contents of .ro/manifest.json

```
{
  "@context" : [ "https://w3id.org/bundle/context" ],
  "id" : "/",
  "manifest" : [ "manifest.json" ],
  "createdOn" : "2017-03-24T11:20:24.123Z",
  "createdBy" : {
    "uri" : "https://view.commonwl.org",
    "name" : "Common Workflow Language Viewer"
  },
  "authoredBy" : [ {
    "uri" : "https://github.com/mr-c",
    "name" : "Michael R. Crusoe"
  }, {
    "uri" : "https://github.com/portah",
    "name" : "Andrey Kartashov"
  }, {
    "uri" : "https://github.com/tetron",
    "name" : "Peter Amstutz"
  } ],
  "aggregates" : [ {
    "uri" : "/workflow/README",
    "mediatype" : "application/octet-stream",
    "createdOn" : "2017-03-24T11:20:25.379Z",
  "authoredBy" : [ {
    "uri" : "https://github.com/mr-c",
    "name" : "Michael R. Crusoe"
  }, {
    "uri" : "https://github.com/tetron",
    "name" : "Peter Amstutz"
  } ],
  "retrievedFrom" : "https://raw.githubusercontent.com/
     common-workflow-language/workflows/920
     c6be45f08e979e715a0018f22c532b024074f/workflows/lobSTR/
     README",
  "retrievedBy" : {
    "uri" : "https://view.commonwl.org",
    "name" : "Common Workflow Language Viewer"
  },
  "bundledAs" : {
    "uri" : "urn:uuid:50c0773b-b244-4594-8504-5d46f6fcc474",
    "folder" : "/workflow/"
  }
}
...
{
  "uri" : "/workflow/samtools-index.cwl",
  "mediatype" : "text/x-yaml",
  "createdOn" : "2017-03-24T11:20:31.997Z",
  "authoredBy" : [ {
    "uri" : "https://github.com/mr-c",
    "name" : "Michael R. Crusoe"
  }, {
    "uri" : "https://github.com/portah",
    "name" : "Andrey Kartashov"
  }, {
    "uri" : "https://github.com/tetron",
    "name" : "Peter Amstutz"
  } ],
  "retrievedFrom" : "https://raw.githubusercontent.com/
     common-workflow-language/workflows/87195224
     c932d63bfa1ec928f25ecdfafb1337d7/workflows/lobSTR/
     samtools-index.cwl",
  "retrievedBy" : {
    "uri" : "https://view.commonwl.org",
    "name" : "Common Workflow Language Viewer"
  },
  "conformsTo" : "https://w3id.org/cwl/draft-3",
  "bundledAs" : {
    "uri" : "urn:uuid:5af12e79-f1ed-435e-b370-ee540855b088",
    "folder" : "/workflow/"
```

```
        }
    }
    . . .  ]
}
```