A Quantitative Study on the Re-executability of Publicly Shared Scientific Workflows

Rudolf Mayer
SBA Research
Vienna, Austria
rmayer@sba-research.org

Andreas Rauber
Technical University of Vienna
Vienna, Austria
rauber@ifs.tuwien.ac.at

***Abstract*—Workflows have become a popular means for implementing experiments in computational sciences. They are beneficial over other forms of implementation, as they require a formalisation of the experiment process, they provide a standard set of functions to be used, and provide an abstraction of the underlying system. Thus, they facilitate understandability and repeatability of experimental research. Also, additional meta data standards such as Research Objects, which allow to add more meta-data about the research process, shall enable better reproducibility of experiments. However, as several studies have shown, merely implementing an experiment as a workflow in a workflow engine is not sufficient to achieve these goals, as still a number of challenges and pitfalls prevail.**

In this paper, we want to quantify how many workflow executions are easy to repeat. To this end, we automatically obtain and analyse a set of almost 1,500 workflows available in the myExperiment platform, focusing on the ones authored in the Taverna workflow language. We provide statistics on the types of processing steps used, and investigate what vulnerabilities in regards to re-execution are faced. We then try to automatically execute the workflows. Form these results, we conclude which are the most common causes for failures, and analyse how these can be countered, with existing or yet to be developed approaches.

I. Introduction

Reproducibility is an important topic in scientific research, as it enables the verification of the validity of the conclusions and claims drawn from scientific experiments. While many disciplines have, over sometimes long-time periods, established a set of good practices for repeating their experiments (e.g. by using experiment logbooks in disciplines such as chemistry or physics, where the researchers record their experiments), computational science often lacks behind. Many eScience investigations are hardly reproducible – sometimes not even by the original investigator. The reasons for this are manifold, ranging from the immaturity and fast pace of change in the technical solutions utilised, to an underdeveloped practice of strict requirements for experiments.

In this paper, we want to quantify technical reproducibility of scientific workflows, i.e. experiments that are encoded in a formal language defined by a workflow engine. Workflows are an interesting object of study for several reasons. On the one hand, they are gaining in popularity, especially in certain eScience disciplines. This has also lead to an increase in social platforms such as myExperiment, which allow sharing of workflows. Therefore, an increasing number of workflows is readily available to other researchers. Further, through their formalisation and programming against a well defined workflow engine, which provides the execution environment, workflows are often conceived to be a step towards reproducibility; sometimes it is even assumed that they are ready to be re-executed by other researchers without any significant additional effort, and that workflows are "self-contained", i.e. that sharing the workflow definition file itself is sufficient.

However, while workflows surely add to reproducibility, there are still difficulties with re-executing them easily. In this paper, we examine workflows implemented for the Taverna workflow engine, and published on the myExperiment platform, a platform for exchanging research objects such as research workflows. As these workflows are intentionally and explicitly published by their authors, it is assumed that they are in general of higher quality and thus more prepared for repeatability than workflows that are primarily used locally by a researcher, but never shared.

We gather a set of almost 1,500 workflows from the platform. We then analyse which types of processing steps are provided by the Taverna workflow engine, and how frequently these are used in our dataset. We further try to automatically execute the workflows using the workflow execution environment, and provide statistics on how many workflows are actually ready to be used. Expanding on previous work, we then conclude on why these workflows do not work, and discuss what additional information and data needs to be made available to make them executable.

The remainder of this paper is structured as follows. In Section II, we give an overview on related work on reproducibility, workflows, and semantic description methods for experiments. In Section III, we characterise the data set we are using for our investigation. In Section IV, we then describe which kinds of processing steps can be implemented in Taverna, and which threats of insufficient documentation and packaging can originate from. We further provide characteristics on how frequent these processing types are in our data set. Section VI then describes the findings from executing the workflows automatically. Finally, Section VIII provides conclusions and an outlook on future work.

II. Related Work

Several studies have shown the problem of repeatability and reproducibility in eScience investigations. An analysis of the reasons of lack of repeatability and reproducibility is given in [1], which defines the terms as follows: *Repetition* is the ability to re-run the exact same experiment with the same method on the same or similar system and obtain the same or very similar result. Reproducibility concerns the independent confirmation of a scientific hypothesis through reproduction by an independent researcher/lab. Reproducibility is carried out after a publication, based on the information in the paper and other information, such as data sets.

A study on the reproducibility of results presented in various computer systems research investigations is presented in [2]. The authors processed 613 papers from eight ACM conferences, out of which 515 papers used a software developed by the authors themselves. The goal of the study was to check whether this software can be used to reproduce the experiments; due to constraints, reproducibility was primarily indicated by the ability to compile and build the software. For 231 of these papers, the authors were able to obtain the source code, but for only 123 systems building the software was actually achieved. The causes for this were in many cases not having the correct environment available, e.g. wrong versions of dependencies that were not explicitly specified. We follow a similar approach in this work, but focus more on the re-execution aspects, as availability of the software is not an issue with publicly shared workflows.

[3] identifies general challenges in the use of scientific workflows, of which reproducibility of results is one prominent aspect. Specifically issues with understanding the workflow, as well as keeping workflows usable despite changes in technology are mentioned.

A study on reproducibility of workflows authored in the Taverna workflow engine is presented in [4]. The authors provide a case study of manually analysing and executing four selected Taverna workflows. The authors identify that even though workflows provide a formalisation of the data processing and orchestration of the execution, still many workflows break after a certain time, due to e.g. decaying third-party dependencies, or insufficient information, such as example input, being provided. The authors analyse common problems on their sample of selected example workflows. The authors further provide an approach to automatically check whether workflows fulfil basic requirements for being still runnable and reproducible, based on the concept of Research Objects [5], which allows for a semantic description and linking of other objects employed in the research investigation. This verification is grounded on the availability of certain types of information, such as sample input and output files, which are defined in a check-list style [6]. It constitutes a necessary condition, but not sufficient as a check for the actual reproducibility. In our work, we perform a similar investigation, but in the form of a quantitative study, rather than focusing on a few prototypical examples.

The Context Model, described in [7], [8], follows similar goal as the Research Objects, but adds a specific focus on the technical environment a workflow relies on, such as the hardware and software needed to execute a workflow. This includes the workflow engine itself, and its dependencies to other software modules. In the case of Taverna, which is implemented in the Java programming language, this would include be the Java Runtime Environment, and in turn the dependencies of that software. Also dependencies of the workflow itself, i.e. to third-party libraries or external services, can be documented and described.

Challenges in workflow reproducibility stemming from the dependency to third-party services are discussed in [9], [10]. In [11], the authors investigate the workflow decay not in a specific moment in time, but over a period of time, including the impact of workflow evolution.

III. Workflow data Set

The platform myExperiment[1] [12] is a social web platform that allows researchers to share their research objects. It is primarily used for sharing scientific workflows, implemented predominantly for the Taverna workflow engine, but also for a few other engines. The platform allows researchers to upload workflows and accompanying files to the repository, and provides a browsable and searchable interface to other researcher to search and find workflows made available by their peers. Most of the workflows are publicly available and can be downloaded by any user; only a few workflows are protected and can only be obtained with a certain authorisation privilege ("private workflows").

Utilising the myExperiment REST API[2], we downloaded the complete set of workflows available as of March 2015. The total number of 2,697 indexed objects stems from the period of time since 2007, an overview of the distribution per year is provided in Figure 1. We can see that the most active periods of workflows uploaded was in 2010 and 2011, with almost 500 objects created each year. In overall, the number of contributed workflows is relatively steady, though.

A total of 40 different workflow types can be observed, though many of these types have just a couple of instances. Most of the workflows are authored in the Taverna Workflow engine [13], in either version 1 or version 2, which differs primarily in the file format (SCUFL vs. t2flow format). Together, they account for more than 75% of the workflows. However, a number of workflows is also authored in other

[1]http://www.myexperiment.org/

[2]http://wiki.myexperiment.org/index.php/Developer:API

Figure 1. Workflows on myExperiment, grouped per creation year

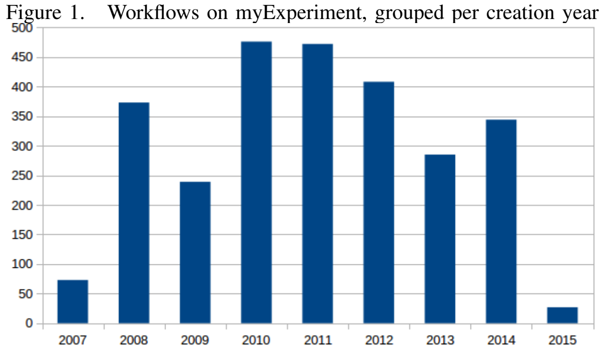

Table I
OVERVIEW ON TYPES OF WORKFLOW RESEARCH OBJECTS IN THE MYEXPERIMENT PLATFORM, MARCH 2015

Type	Count	Percentage
Others	132	4.9%
BioExtract Server	19	0.7%
GWorkflowDL	24	0.9%
LONI Pipeline	26	1%
KNIME	43	1.6%
Bioclipse Scripting Language	44	1.6%
Kepler	45	1.7%
Galaxy	54	2%
RapidMiner	271	10%
Taverna 1	565	20.9%
Taverna 2	1474	54.7%
Total	2697	

systems, notable the Rapid Miner[3] and KNIME[4] data mining applications, Galaxy[5], Kepler[6] or the LONI pipeline[7] workflow engines. A detailed overview on the exact numbers is given in Table I, including over 30 types of research objects that have between one and twelve examples each, grouped as "Others", accounting for less than 5% of all research objects.

Of the initially retrieved workflow index, 92 (3.4%) were not accessible private workflows, distributed over the different types roughly equivalent to the total numbers. Due to the large majority of workflows being authored in Taverna, we are focusing our investigation on this workflow type. This is also further motivated by the readily available, open source implementation of Taverna, which allows for automated analysis and execution of workflows. As there are differences in the semantics between the discontinued Taverna 1 and the current Taverna 2 workflow capabilities, we further focus on the latter. As 30 workflows of type Taverna 2 are marked private, this leaves us with a dataset of 1,444 workflows that are publicly available, out of which one workflow file was not readable by the Taverna workflow engine.

[3]https://rapidminer.com/
[4]https://www.knime.org/
[5]http://galaxyproject.org/
[6]https://kepler-project.org/
[7]http://pipeline.bmap.ucla.edu/

IV. TYPES OF PROCESSING STEPS AND VULNERABILITIES

Taverna, similar to other workflow engines, allows researchers to build a workflow by defining several *processors*, which represent a processing step. For each processor also the input and output ports are defined. By defining the data flow between these input and output ports, processors are then connected to each other. An example of a Taverna workflow is given in Figure 2.

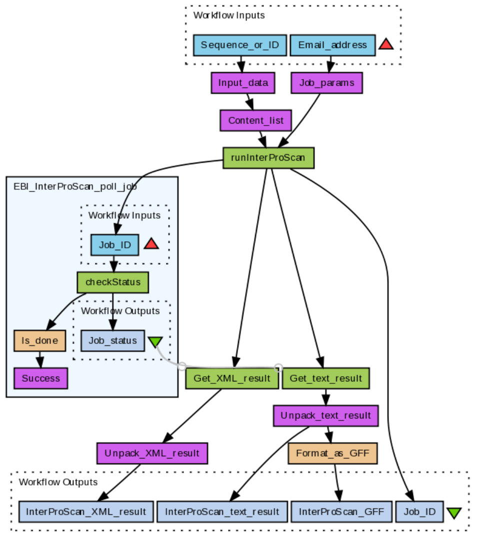

Figure 2. Example Taverna workflow

Processors in Taverna can roughly be divided in two categories. The first type of processors provide a pre-defined, configurable functionality. For example, a processor provides an implementation of a client for the call to an external Web Service; the user provides the service interface definition, and which method is going to be called, and provides the parameter values for the remote method by connecting the input ports of the web service client processor. Further, there are also processors that allow writing customised code, and thus allow implementation of almost any functionality. In the case of Taverna, this code is to be written in the *BeanShell* language, which is based on the Java programming language. Taverna uses the term "activity" for implemented functionality that is provided to a processor; we use these terms interchangeably in the rest of this paper. A more detailed overview on the Taverna workflow language is given in [14].

In the remainder of this section, we analyse potential vulnerabilities for re-execution of workflows. We thereby find some commonalities to [4], but also detail on additional vulnerabilities not identified in that work. Further, we want to provide a clear association of vulnerabilities to the specific type of Taverna processors they apply to. Each vulnerability is denoted by a named identifier, for later reference.

A. *Web Service Execution*

Taverna provides two major processors for invoking remote Web Services – WSDL and REST service processors. Both processors require configuration, such as the remote address and inputs for the parameters. Common threats with these processors not functioning as expected are:

- *WS1* The remote address is not reachable from the current location, for example, it is a service that is only reachable in the specific, not public network of the original workflow author.
- *WS2* The remote service (though it was originally in principle reachable) is not available any more.
- *WS3* The remote service requires some sort of authentication, but that authentication is unknown to the researcher, or the authentication data has changed / became invalid (e.g. a changed password, or a deleted user/agent).
- *WS4* The remote method to be invoked is not available any more, i.e. has been removed from the service specification.
- *WS5* The interface specification of the service has changed, for example, a method now requires a different number of parameters, or returns a different result type.

Besides WSDL and REST, there is a third type of processor for remote service execution provided in Taverna, the *Soaplab* processor. Soaplab[8] is a framework that allows running command line applications as a web service. Soaplab is now mostly discontinued[9]. Newly published workflows are therefore unlikely to use this functionality; however, a small number of workflows in our dataset still exhibit this functionality. The vulnerabilities are similar to the ones identified for WSDL and REST.

B. *Beanshell Processor*

A Beanshell processor allows the researcher to define custom processing logic using the expressiveness of the Java Programming Language. The functionality of libraries provided by the Java Programming Language is available as in a regular Java program. Further, custom libraries, so-called Java Archives (JARs) can be assigned to a Beanshell script, and are subsequently available for use. Common threats with this processor not functioning as expected are:

- *B1* A different version of the Java runtime is used to execute the processor, and thus some of the Java API functions are not yet or not any more available, or behave differently.
- *B2* A custom library required for a Beanshell is not available.

[8]http://soaplab.sourceforge.net/

[9]For example, the European Bioinformatics Institute (EBI), one of the principal provider of Soaplab functionality, has closed their services in February 2013 and now only offers REST and WSDL based interfaces (cf. http://www.ebi.ac.uk/soaplab)

- *B3* A custom library required for a Beanshell is available in the wrong version, which could lead to a different API, or a different behaviour of the requested functionality, such as an older version still containing a bug.
- *B4* The Beanshell code accesses some functionality outside the Java runtime environment that is not available. This can, for example, be a call to a web service using the Java API instead of the explicit Taverna processor, a call to a system process or library (e.g. via the Java Native Interface (JNI) or the *Process* class in Java). Besides executables, also the access to files, on the local file system or from remote locations, can be a source of re-executability problems.

A specific form of Beanshell processors are the so-called *local workers*. These are pre-configured beanshell scripts for performing common tasks, such as Base64 encoding or obtaining the contents from a remote URL. There are no specific additional vulnerabilities identified with these processors, and not all of the above vulnerabilities apply to them. However, while a user can utilise the local worker as defined, she can also modify them, in which case these processors become regular Beanshell scripts.

C. *RShell Processor*

The RShell processor allows the workflow creator to write scripts using the R programming language[10], which is focused on statistical computing and graphics. Taverna does not directly invoke R on the local system. Instead, it utilises the RServe[11] protocol to communicate with an R installation that runs the RServe server program. Such an RServe server can be also running locally, on the same system that executes the workflow. The user configures the (remote) address of the server, and optionally also a user name and password for authentication. Given these characteristics, the RShell processor is in principle vulnerable to similar threats as identified for the Beanshell and the Web Service processors.

- *R1* The remote address of the RServe instance is not reachable from the current location (similar to Web Service Processor vulnerability *WS1*).
- *R2* The remote address is not available anymore (similar to Web Service Processor vulnerability *WS2*).
- *R3* The authentication data for the RServe instance is unknown or invalid (similar to Web Service Processor vulnerability *WS3*).
- *R4* A different version of the R runtime than originally used is now employed to execute the processor, and thus some of the R API functions are not available, or behave differently (similar to Beanshell vulnerability *B1*).

[10]http://www.r-project.org/

[11]http://www.rforge.net/Rserve/

- *R5* A custom R package required is not available at the R instance (similar to Beanshell vulnerability *B2*).
- *R6* A custom R package required is of the wrong version, which can lead to a different API available, or a different behaviour of the requested functionality, such as an older version still containing a bug (similar to Beanshell vulnerability *B3*).
- *R7* The R code accesses some functionality outside the R runtime that is not available (e.g. via the "system" call, similar as in the Beanshell vulnerability *B4*).

D. Tool Processor

This processor allows the researcher to define an invocation of a tool, which in general is any binary that can be executed on a system. The command to invoke is defined by its full path, which is appended by the parameters (if any) that are going to be passed to the command. The parameters can be connected to the input ports of the process step.

Further, it is also possible to specify a location where the command shall be executed at. This location is by default the local machine. However, Taverna can also connect to a remote machine, primarily via the SSH (secure shell) protocol. There is no information in the processor definition or configuration that would indicate which is the expected target platform of the command to be executed (for example Windows, Mac or Linux). Neither does it include any other requirements to that platform, such as a list of required applications or packages, or requirements to the installation destination (e.g. a specific folder in the file system).

As a consequence, this processor becomes vulnerable to the following threats.

- *T1* The location is not reachable from the current location (similar to Web Service Processor vulnerability *WS1*).
- *T2* The location is not available anymore (similar to Web Service Processor vulnerability *WS2*).
- *T3* The authentication data for the (remote) location is unknown or invalid (similar to Web Service Processor vulnerability *WS3*).
- *T4* The tool to be executed is not available on the system, or cannot be identified due to a slightly different installation. This can be the case if a tool is otherwise installed in the system, but in a different folder than expected, or not available in the default search path.
- *T5* A different version of the tool than originally used is now employed to execute the workflow, and thus some of the functions are not available, or behave differently.

E. Other Vulnerabilities

One other potential Vulnerability to the successful workflow re-execution is in the *workflow input ports*. Similar to input ports for processors, these workflow input ports constitute parameters for the workflow execution, and thus control some of the characteristics of the workflow runtime behaviour. The exact values for these workflow input ports are thus to be provided before a workflow is run.

The Taverna workflow definition does allow to specify example values for the input ports, which can be seen as default values for which the workflow should perform correctly. However, not many workflows do actually provide example values. In such a situation, the user then needs to rely on other documentation, or if that is also missing, an educated guess for potential values. Taverna does not provide information on the types of input ports (for example whether there is a number or a URL as type expected). Thus, the only clue on allowed values for ports provided with the workflow definition is hidden in the naming of the workflow ports.

The lack of appropriate workflow input port values is a hard barrier preventing the execution outright.

F. Trivial Processors

There are a number of "trivial" processors available in Taverna as well. The *String Constant* activity allows to define a constant value that can be connected to another processor input. Frequently, string constants are used instead of workflow inputs to provide runtime behaviour control. Other simple processors are the WSDL and XML *splitter* and *merger* processors, which extract single values from an XML document, or combine and wrap values to an XML document. There are no specific vulnerabilities that can be identified for these processors.

V. Workflow Characteristics

In this section we provide details of a static analysis of the workflows, i.e. of analysing the workflow definition, and not considering any execution aspects yet. We perform this analysis by utilising the Taverna API and computing statistics over various properties as detailed below. Analysis of most workflows is a matter of a couple of seconds, however, many workflows can take 20 minutes or more to be properly opened by the Taverna API. This is due to Taverna performing some validity checks, e.g. with remote services, with rather long time-outs before giving up connecting to remote locations.

In our analysis, we start at the beginning of the workflow execution, the workflow input ports. There are 345 workflows, equalling to approximately 24%, which have no input ports, and 1,098 workflows that do use input ports. The average number of input ports per workflow is 1.31. Of those workflows with input ports, only 572 provide example values for all input ports, while 97 provide example values for some inputs, and 429 provide no example values at all. Thus, we can conclude that at most 917, equalling to around 63.5% of the workflows, can in principal be automatically executed without any additional information on the workflow input parameters – assuming that all example values are actually valid data.

Table II
INPUT PORT STATISTICS

Total workflows	1443	
Workflows with no input ports	345	23.91%
Workflows with input ports	1098	76.09%
Workflows with no example values	429	29.73%
Workflows with some example values	97	6.72%
Workflows with all example values	572	39.64%
Workflows that can be run	917	63.55%

In our data set, a total of 17,547 processors are used, meaning each workflow uses in average 12.16 processors. Regarding the processor types, Table III gives an overview on the cumulative counts of different types being used, as well as the relative frequencies. This table is simplified by excluding the earlier mentioned *trivial processors*, for example *string constants*, as well as *XML splitter* and *XML merger* templates. In total, these account for 25% and 15% of all the elements used in a workflow, respectively. Thus, the first percentage given in Table III represents the relative frequency of the processors among all instances, while the second percentage depicts the relative frequency only among the non-trivial processors.

Table III
TYPES OF TAVERNA PROCESSORS

Processor type	Occurrences	% of total	% of non-trivial
LocalworkerActivity	3,226	18.38%	30.42%
BeanshellActivity	2,664	15.18%	25.12%
DataflowActivity	1,401	7.98%	13.21%
WSDLActivity	707	4.03%	6.67%
ExternalToolActivity	563	3.21%	5.31%
RESTActivity	339	1.93%	3.2%
RshellActivity	337	1.92%	3.18%
XPathActivity	313	1.78%	2.95%

We can see that from the non-trivial processors, a large amount of processing is done by local workers (the pre-defined Beanshell scripts), followed by custom Beanshell scripts. WSDL, REST and other activities account for a smaller share of total processors used. This can be explained by the fact that many workflows use several other processors to prepare input data to be sent to a service, and then again subsequently to process the output from these services. The more interesting characteristic is therefore not the absolute number of processor usages, but how many workflows use a specific type of processor. We will provide detailed numbers to this below. A high number of processors are actually representing nested or sub-workflows, denoted as *DataflowActivity*. In our analysis, we resolve these nestings and handle the sub-workflows as a part of the enclosing workflow.

When a Workflow is opened in Taverna, also a basic check for the availability of each of the service processors is performed, mainly on WSDL and Soaplab processors. This is composed of a basic check on whether the given URL is reachable. If this is not the case, the activity becomes *disabled*. These account for the remaining approximately six percent of activities not listed in Table III. We can observe from Table IV that a total of 288 workflows, i.e. approximately every fifth workflow, has such a disabled activity. The majority of 87% of these disabled activities are unreachable WSDL services, the remaining 13% are Soaplab processors.

Table IV
DISABLED PROCESSORS

Workflows with disabled activities	288	19.96%
Count of disabled WSDLActivity	976	86.99%
Count of disabled SoaplabActivity	146	13.01%

Regarding remote services, we can see in Table V that web services using WSDL as an interface language are used in around 30% of all the workflows analysed. It is thus by far the more popular mechanism compared to REST, which is used in approximately 12.5% of all workflows, and Soaplab, which features in just below 3% of all workflows. Together, either one of the web services mechanism is used in 43.79%, close to almost half of all workflows. Regarding the hosts where these services run, we can observe that only 10 of them are on non-globally reachable IPs, that is either the *localhost*, or within a private IP range (such as 192.168.x.x), denoting e.g. a company or home network.

Table V
WEB SERVICE PROCESSORS

Workflows using WSDL services	411	30.04%
Workflows using REST services	172	12.57%
Workflows using Soaplab services	38	2.78%
Workflows using any web service	599	43.79%
WSDL with non-global IPs	10.0	1443

In Table VI, we give an overview on Beanshell processors. Note that this number does not include the above mentioned "local worker" activities, which are pre-defined Beanshell scripts. A total of 717 workflows, equalling almost 50% of the workflows, use a Beanshell processor to execute a script. Out of the total of 2,664 beanshell processors, 76 have dependencies to external Jar libraries defined. As a processor can define dependencies to multiple libraries, the total number of required JAR files is 90. These dependencies affect in total 33 workflows.

Analysing the libraries that are used by workflow authors, the majority of library names suggest customised implementations being used. However, there is a significant amount of dependencies to "well-known" libraries. Notable examples are database access drivers for the Java Database Connectivity (JDBC) API, e.g. for MySQL and SQLite, as well as frequently used libraries from Apache Commons[12],

[12]http://commons.apache.org/

or for the manipulation of data in the JSON format.

Table VI
BEANSHELL PROCESSORS

Beanshell processors	2664	
Workflows with beanshell processors	717	49.69%
Beanshell processors with dependencies	76	2.85%
Total Beanshell dependencies	90	3.38%
Workflows with beanshell dependencies	33	1.24%

Details on the tool processors are given in Table VII and Table VIII for local and remote instances, respectively. We can observe that the majority of processors are local tools, i.e. executed on the same machine as the workflow.

240 workflows, equalling to almost 17%, use a local tool invocation. We can observer that most local tool invocations are rather short (at most 3 lines, including comments and interpreter directives), and are thus rather invocations of one command, with potential minimal post-processing. On the other hand, there are also a number of rather lengthy shell scripts executed.

Regarding the script contents, we did a coarse analysis on major trends. We can observe that there is a high number of 55 tool invocations that use the Python language for processing, while only three make use of the Perl scripting language. Interestingly, there is also 22 scripts that call a Java application, even though that could be much easier, and more explicitly, achieved using the Taverna Beanshell processor.

Several processors also utilise rather large software applications, such as the Hadoop server for distributed computing. Sometimes the local tool processor is used to ensure that such a server application is started and running in the background, before it is utilised by other, subsequent processing steps.

Table VII
LOCAL TOOL PROCESSORS

Local tool processors	543	
Workflows with local tool processors	240	16.63%
Short tool invocations	442	81.4%
Script-style tool invocations	101	18.6%
Python	55	10.13%
Java	22	4.05%
Perl	3	0.55%
Hadoop	37	6.81%

Regarding the remote tools, twelve workflows use this type of remote invocation, in a total of 20 processor steps. The functionality of the tools varies greatly, in a similar fashion as for the local tool processors. Some tools invocations execute a number of commands in shell scripts, while others execute scripts in Python or Perl. As a processor can access multiple remote hosts for distributed computing, a total of 34 host connections are defined. A number of processors use the same hosts, thus there are 20 uniquely defined remote hosts utilised. Out of those 20 unique hosts, only 7 are actually reachable. Specifically of impact for the number of re-executable workflows is that none of the hosts that is used by multiple processors in different workflows are reachable. As a consequence, only two of the workflows have all of their remote hosts reachable and are thus technically still repeatable – if the required user name and password are known.

Table VIII
REMOTE TOOL PROCESSORS

Remote tool processors	20	
Workflows with remote tool processors	12	0.83%
Total remote hosts	34	
Unique remote hosts	20	
Reachable remote hosts	7	20.59%
Reachable unique hosts	7	35%
Workflows with reachable hosts	2	16.67%

Table IX
RSHELL PROCESSORS

Workflows with RShell processors	90	6.24%
Total remote hosts	337	
Hosts with local address	335	
Unique remote hosts	2	
Reachable hosts	0	0%
Workflows with reachable hosts	0	0%

RShell processors are used in 90 workflows, which amounts to a bit more than 6% of all workflows, as detailed in Table IX. In contrast to the remote tool invocation, the number of hosts defined for the RServe instances equals the number of RShell processors, as there is only one host definition allowed. One interesting fact is that in almost all cases, the host that a connection should be established to is the localhost, on the default port. Only one workflow makes, with two RShell processors, use of an external RServe instance, which is however also not reachable. Thus, without the local RServe instance, none of the 90 workflow using RShell is executable.

VI. WORKFLOW RE-EXECUTION

In this section we describe the results from re-executing the workflows in our collection. Table X details the data set that we are finally able to execute in our environment.

Table X
EXECUTABLE DATA SET

Initial data set	1443
Removed – missing input values	526
Removed – disabled processors	180
Removed – not executable in test environment	6
Executable workflows	731

We start by first removing all workflows that can not be run because we are missing some or all values for the input ports. As discussed in Section V, this leaves 917 workflows. From this set, we then remove all workflows that have disabled processors, and can thus not be run by Taverna. This removes another 180 workflows not already eliminated for missing example values, finally leaving a pool of 737 workflows that we are able to run.

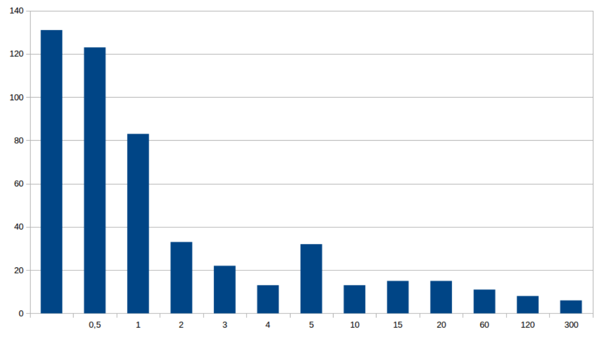

Figure 3. Histogram of execution times of the Taverna workflows, in seconds

Execution times of most workflows are rather short, in a matter of tens of seconds; however, there are several workflows that can take 30 minutes or longer to compute, on a dedicated server infrastructure where processing, RAM or network connections are not the limiting factor. An overview on the execution time is given in Figure 3. Causes for these are often looping executions of services that will eventually fail e.g. with a connection timeout, similar to the observation from performing the static analysis of the workflows as described in Section V.

Table XI
WORKFLOW EXECUTION RESULTS

Execution successful	341
Execution failed	364
REST service not reachable	4
REST service not authenticated	4
Other missing authentication (WSDL, RShell, ...)	30
Resource/File not available (local or remote)	13
Tool command not available	15

Details on the execution results are given in Table XI. From the the 737 workflows, 364 workflows produced errors during the execution, while another six workflows did not terminate execution after more than 48 hours of runtime. This means that only 341 workflows were executed without any errors, amounting to only 29.2% of the original data set of 1,443 objects.

The workflows that could no be executed failed, among others, for the following reasons. Nine workflows failed with issues with a REST service, for four of which the failure was caused by the service not being reachable. The other five workflows failed due to no proper authentication information being found in the connection details. Another 40 workflows failed for missing authentication data. This applies to workflows with processors where the authentication is explicitly configured to be required, but not obtainable from the local Taverna instance, and includes processors such as WSDL, RShell or the remote Tool Invocation.

14 workflows failed because they wanted to open a resource, for example a file, locally or on the web, that is not available anymore. Another 19 workflows failed because the local tool commands that the workflow wanted to execute was not available. Example commands are for media format conversions via “ffmpeg”, “convert”, or “dcraw”, optical character recognition (“tesseract“), or the ”hadoop” and “hive” commands.

Other workflows failed because the example input data was not pure, e.g. when there are more potential values listed, or when there is a value and a descriptive comment. Some of these issues could be fixed by human intervention, but the correct input values are not always obvious to guess from the provided data.

It has to be noted that with re-executing workflows, we just laid the foundation for assessing whether the result was actually repeatable, i.e. whether the same results were achieved. Such a verification can be then performed e.g. with the methods proposed in [15], [16], if there is sufficient information on the expected output of the workflow.

VII. RECOMMENDATIONS

After inspecting and analysing a large number of workflows and identifying patterns for failure in repetition, we can draw a couple of recommendations that could help alleviate the problem in the future. We can tackle the issues from several angles, including the workflow definition language, the completeness of specification and bundle, the publishing and the monitoring processes. While there are several proposals for best-practices for workflow authors, e.g. in [17] or by the Workflow4Ever project[13], we want to give recommendations also towards the technical infrastructure.

Capabilities of workflow engines: Regarding the implementation of logic in the workflows, we can observe that a rather large number of workflows use, in one way or another, scripting code that is run outside the workflow engine itself, e.g. by using the local tool invocation to execute a local Python interpreter with a specific script. While Taverna provides Beanshell, which basically allows for any Java ctool invocation ode to be run, the effort for authoring each script in this language might be too cumbersome. This is especially be the case when existing code is to be reused in a workflow implementation. Therefore, by extending the capabilities of Workflow

[13]http://www.wf4ever-project.org/bestpractices, archived at http://www.webcitation.org/6ZNgmtWVA

engines towards other popular languages, one source of error can be reduced. As an example, the workflow management software *Activiti*[14] allows the user to work with all languages available for the "Scripting for the Java Platform"[15], which includes JavaScript, Python, Perl, and others. If Taverna provides similar capabilities, many calls to local tool invocations could be replaced with native workflow engine scripts.

Expressiveness of workflow definition language: Regarding the verbosity of workflow definitions, the use of auxiliary descriptive languages, such as the Research Objects (RO) or the Context Model (CM) allows for resolving some of the issues identified, for example, the lack of external dependencies being defined, or the lack of matching input and output data to verify that a workflow execution actually produced the required results. However, integrating some of these aspects directly in the workflow language might be of advantage and foster use and uptake, as it requires less additional effort to the workflow authors.

Tool Invocations: Regarding the local tool invocation, it can be observed that many calls to the local system are in a platform dependent way, i.e. they are tailored to Linux or Windows, and will not work on other platforms. We propose two different solutions to this issue. On the one hand, making such indirect dependencies of a tool invocation to an execution environment explicit would be beneficial for chosing the correct platform for re-execution. A simple extension to the descriptive metadata that can be provided for the tool invocation in Taverna would be sufficient. On the other hand, workflow engines could provide a mechanism to workflow authors to provide alternative scripts for different platforms, thus making the workflow more platform independent, and thus simplifying re-execution in different environments.

External dependencies: External available definitions that help understanding the functionality, e.g. a WSDL specification for a web service, should be archived and published together with the workflow, before it becomes unavailable. Such meta-data is required to enable approaches such as automatic service substitution, as described in [10].

Quality check and monitoring: Publication platforms such as myExperiment need to further enforce certain checks when workflows are uploaded, similar to, but extending on the "Checklist Evaluation Service" provided by the Workflow4Ever project [4], [6]. Such a service could easily identify missing input parameter examples. This is partly done already e.g. in the Research Object Digital Library (ROHub)[16] developed by Workflow4Ever. However, such quality checks should also be extended to verify e.g. whether required Java libraries are provided alongside the workflow definition, among others.

Finally, monitoring for external services needs to be improved. Current approaches are often limited to WSDL definitions only, not considering the many options of how external functionality is utilised in workflows, e.g. via REST, Soaplab, remote R servers, or simply SSH invocations.

VIII. Conclusions and Future Work

In this paper, we provided an extensive study on a data set of publicly shared workflows, namely the ones authored in the Taverna 2 language and published on the myExperiment platform. We identified vulnerabilities for the different types of processing elements that are used in Taverna, followed by a static analysis of the typical use of processors and their configuration in the data set. We then attempted to re-execute each workflow in an automated fashion, with manual inspections where problems occurred. From these experiences, we provide a set of recommendations that can enhance the reproducibility of workflows. We identified several angles from which the problems can be addressed, from the workflow definition and the use of auxiliary meta-data such as Research Objects, to an improved publishing and monitoring process.

The majority of workflows are not re-executable, and often the cause is in rather trivial shortcomings. The lack of example values needed as workflow input parameters, as well as missing libraries for Java programs are frequent causes for failures in workflow execution.

Future work will focus on extending our database to also include the Taverna 1 workflow definitions, and performing a more fine-grained manual analysis of the problems encountered during execution. We also want to investigate whether there is a well supported correlation between the "age" of a workflow, and its ability to be repeated. Finally, we will focus on proposing technical drafts and implementations for the recommendations given in Section VII.

Acknowledgment

Part of this work was co-funded by the research project DEXHELPP, supported by BMVIT, BMWFW and the state of Vienna, and COMET K1, FFG - Austrian Research Promotion Agency.

References

[1] J. Vitek and T. Kalibera, "R3: Repeatability, reproducibility and rigor," *SIGPLAN Not.*, vol. 47, no. 4a, pp. 30–36, March 2012.

[14] http://activiti.org/

[15] https://www.jcp.org/en/jsr/detail?id=223

[16] http://www.rohub.org/

[2] C. Collberg, T. Proebsting, G. Moraila, A. Shankaran, Z. Shi, and A. M. Warren, "Measuring reproducibility in computer systems research," University of Arizona, Tech. Rep., 2014. [Online]. Available: http://reproducibility.cs.arizona.edu/

[3] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *IEEE Computer*, vol. 40, no. 12, pp. 24–32, Dec 2007.

[4] J. Zhao, J. M. Gómez-Pérez, K. Belhajjame, G. Klyne, E. García-Cuesta, A. Garrido, K. M. Hettne, M. Roos, D. D. Roure, and C. A. Goble, "Why workflows break - understanding and combating decay in taverna workflows," in *8th IEEE International Conference on E-Science (e-Science 2012), Chicago, IL, USA, October 8-12, 2012*. IEEE Computer Society, 2012, pp. 1–9.

[5] K. Belhajjame, O. Corcho, D. Garijo, et. al, "Workflow-centric research objects: First class citizens in scholarly discourse," in *Proceedings of Workshop on the Semantic Publishing, (SePublica 2012) 9th Extended Semantic Web Conference*, May 28 2012.

[6] J. Zhao, G. Klyne, M. Gamble, and C. A. Goble, "A checklist-based approach for quality assessment of scientific information," in *Proceedings of the 3rd International Workshop on Linked Science (LISC2013) In conjunction with the 12th International Semantic Web Conference (ISWC 2013)*, ser. CEUR Workshop Proceedings, vol. 1116. CEUR-WS.org, October 21 2013, pp. 34–45. [Online]. Available: http://ceur-ws.org/Vol-1116

[7] R. Mayer, G. Antunes, A. Caetano, M. Bakhshandeh, A. Rauber, and J. Borbinha, "Using ontologies to capture the semantics of a (business) process for digital preservation," *International Journal of Digital Libraries (IJDL)*, vol. 15, pp. 129–152, April 2015.

[8] R. Mayer, T. Miksa, and A. Rauber, "Ontologies for describing the context of scientific experiment processes," in *Proceedings of the 10th International Conference on e-Science*, Guarujá, SP, Brazil, October 20–24 2014.

[9] D. De Roure, K. Belhajjame, P. Missier, J. M. Gómez-Pérez, R. Palma, J. E. Ruiz, K. Hettne, M. Roos, G. Klyne, and C. Goble, "Towards the preservation of scientific workflows," in *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPRES 2011)*, Singapore, 2011.

[10] K. Belhajjame, C. Goble, S. Soiland-Reyes, and D. De Roure, "Fostering scientific workflow preservation through discovery of substitute services," in *E-Science (e-Science), 2011 IEEE 7th International Conference on*, Dec 2011, pp. 97–104.

[11] J. M. Gómez-Pérez, E. García-Cuesta, A. Garrido, J. E. Ruiz, J. Zhao, and G. Klyne, "When history matters - assessing reliability for the reuse of scientific workflows," in *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 8219. Springer, October 21–25 2013, pp. 81–97.

[12] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, and P. e. a. Li, "myexperiment: a repository and social network for the sharing of bioinformatics workflows," *Nucleic Acids Research*, vol. 38, no. suppl 2, pp. W677–W682, 2010.

[13] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble, "Taverna, reloaded," in *Proceedings of the 22nd international conference on Scientific and Statistical Database Management*. Berlin, Heidelberg: Springer, June 2010, pp. 471–481.

[14] W. Tan, P. Missier, R. Madduri, and I. Foster, "Building scientific workflow with taverna and BPEL: A comparative study in caGrid," in *Service-Oriented Computing – ICSOC 2008 Workshops*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5472, pp. 118–129.

[15] T. Miksa, S. Pröll, R. Mayer, S. Strodl, R. Vieira, J. Barateiro, and A. Rauber, "Framework for verification of preserved and redeployed processes," in *Proceedings of the 10th International Conference on Preservation of Digital Objects (IPRES2013)*, Lisbon, Portugal, September 2–6 2013.

[16] T. Miksa, R. Vieira, J. Barateiro, and A. Rauber, "VPlan – ontology for collection of process verification data," in *Proceedings of the 11th International Conference on Digital Preservation (iPres 2014)*, Melbourne, Australia, October 6–10 2014.

[17] K. M. Hettne, K. Wolstencroft, K. Belhajjame, C. A. Goble, E. Mina, H. Dharuri, D. D. Roure, L. Verdes-Montenegro, J. Garrido, and M. Roos, "Best practices for workflow design: How to prevent workflow decay," in *Proceedings of the 5th International Workshop on Semantic Web Applications and Tools for Life Sciences, Paris, France, November 28-30, 2012*, ser. CEUR Workshop Proceedings, vol. 952, 2012.