Using ontologies for verification and validation of workflow-based experiments

Article *in* Journal of Web Semantics · January 2017

DOI: 10.1016/j.websem.2017.01.002

CITATIONS
19

READS
971

2 authors, including:

Tomasz Miksa
SBA Research
42 PUBLICATIONS **268** CITATIONS

SEE PROFILE

Using ontologies for verification and validation of workflow-based experiments

Tomasz Miksa

SBA Research, Wien, Austria

Andreas Rauber

Vienna University of Technology, Wien, Austria

Abstract

Scientific experiments performed in the eScience domain require special tooling, software, and workflows that allow researchers to link, transform, visualize and interpret data. Recent studies report that such experiments often cannot be replicated due to differences in the underlying infrastructure. The provenance collection mechanisms were built into workflow engines to increase research replicability. However, the traces do not contain the execution context that consists of software, hardware and external services used to produce the result which may change between executions.

The problem thus remains on how to identify such context and how to store such data. To address this challenge we propose the context model that integrates ontologies which describe workflow and its environment. It includes not only high level description of workflow steps and services but also low level technical details on infrastructure, including hardware, software, and files. In this paper we discuss which ontologies that compose the context model must be instantiated to enable verification of a workflow re-execution. We use a tool that monitors a workflow execution and automatically creates the context model. We also authored the VPlan ontology that enables modelling validation requirements. It contains a controlled vocabulary of metrics that can be used for quantification of requirements. We evaluate the proposed ontologies on five Taverna workflows that differ in the degree on which they depend on additional software and services.

The results show that the proposed ontologies are necessary and can be used for verification and validation of scientific workflows re-executions in different environments without the necessity of accessing the original environment at the same time. Thus the scientists can state whether the scientific experiment is replicable.

Keywords: verification, validation, workflow, reproducibility, context model

1. Introduction

In many natural science disciplines, complex and data driven experiments form the basis of research [1]. Scientific breakthroughs would not be possible without special tooling, software and processes that allow researchers to link, transform, visualise and interpret the data [2]. The low maturity of tools and the possible lack of scientific scrupulousness [3] led to a low reproducibility and replicability of experiments [4], [5]. Many problems can be attributed to the fact that the software is not available any more [4]. This may appear to be a pure management issue that can be overcome by imposing better policies, but recent findings show that also the context in which the software is run, that is the infrastructure and the third party dependencies, can have a crucial impact on the final results delivered by a computational experiment. In [6] the authors demonstrate that a different version of the operating system used for neuroimaging analysis in clinical research produces different visualisations. This implies that in order to replicate the same result, not only the same data must be used, but also it must be run on an equivalent software stack.

Workflow engines were proposed in order to bring some standardisation, as well as to hide complexity of the underlying infrastructure. Workflow engines like VisTrails[7], Kepler [8] or Taverna [9] have become popular in research areas like Astronomy, Bioinformatics or Clinical Research. They enable researchers to graphically represent their experiments in form of workflows that can be built using pre-defined elements. These elements range from dedicated data parsers to in-

terfaces for calling external web services. Workflows can also be shared with other researchers, so that they can replicate the original experiment or reuse it. Independent peers can verify the research by re-executing the workflow and establish trust that the experiment results are correct. This in turn accelerates research, because peers have higher confidence to reuse others workflows.

In spite of such a standardisation, a recent study [10] reports that only 30% of almost 1500 Taverna workflows published on myExperiment can be re-executed. This does not imply that the execution produces correct results, but simply that the workflow executes. So far there is no practice at all to provide data that would enable verification and validation of workflows re-executions. Research Objects [11] are aggregations of resources used in scientific investigations that enhance the description of experiments, but do not provide sufficient data, because they are focused on the scientific aspect of the experiment and not on the technical details of its implementation.

The problem thus remains of how to identify and store such data. Workflows share common infrastructure with other software running in the operating system and can delegate tasks specified in the workflow to be executed by tools installed in the environment. Such tools may require a specific configuration and presence of further tools that depend on specific software libraries or dedicated hardware. All these dependencies constitute a workflow execution context that needs to be captured and verified to state whether the workflow re-execution produced results in the right way.

To address this problem we apply the VFramework [12] which can be used to verify and validate workflow re-executions. It uses the context model [13] to document the environments in which the workflow executes and thus enables comparison of workflow executions without necessity of accessing both environments at the same time. By comparing context models of workflow executions we verify whether the workflow re-execution was obtained in a compliant way. The context model integrates ontologies that describe workflow and its environment. It includes not only high level description of workflow steps and services but also low level technical details on infrastructure, including hardware, software, and data. The context model was originally designed for preservation of business processes. In this paper we show how it can be applied in a short term settings for verification of scientific workflow executions. Hence, we discuss which ontologies that compose the context model must be instantiated to enable verification of a workflow re-execution. For that purpose we use tools that automatically source necessary information and represent it as an ontology. Our discussion is focused on Taverna workflows, but the challenges and ways of addressing them remain valid for other workflow systems as well, differing in the actual technical implementation.

We also design an extension to the context model that enables modelling validation requirements. Workflows process data in a series of steps. Sometimes only the final result of a computation is important for researchers, while in other cases the intermediate data is important as well, especially when parts of the workflow are reused in other experiments. In all cases, the researchers must be able to validate whether the partial or full re-execution was valid - produces the same result as the original execution. Hence, based on the analysis of sample workflows we define validation requirements that check the correctness of data produced at multiple stages of workflow execution. We revise the VPlan ontology [14] which extends the context model with validation requirements, metrics used to quantify them, and measurement points. Furthermore, we extended the VPlan with a comprehensive and extensible vocabulary of metrics that can be used for breaking down validation requirements.

We evaluate the proposed set of ontologies on Taverna workflows using five scientific workflows from three domains: sensor data analysis in civil engineering, music classification in information retrieval, and medical clinical research. The selected workflows require multiple local dependencies ranging from additional libraries, scripts, and specific packages to external services for completing workflow steps. The evaluation takes into account re-executions in different operating systems.

The paper is structured as follows. In Section 2 we describe related work on workflow context modelling and capturing. Section 3 provides an overview of the VFramework that we follow to collect data enabling verification and validation of workflow re-executions, and also presents the running example. Sections 4 – 7 describe ontologies constituting the context model, in particular: Section 4 describes the workflow model, Section 5 describes software and hardware dependencies, Section 6 describe the workflow instance, and Section 7 describes file formats identified for the workflow instance. In Section 8 we identify validation requirements for Taverna workflows and describe the VPlan ontology for storing these requirements. In Section 9 we present the evaluation. Section 10 presents recommendations for improving reproducibility of workflows. Conclusion are in Section 11.

2. Related work

The *wf4Ever* project [15] addressed challenges associated with the preservation of scientific experiments in data-intensive science. The aim of the project was to make the workflows preservable and reusable. The key contribution of the project is the definition of Research Objects (RO) [11] that are aggregations of resources used in scientific investigations. They contain scientific workflows, the provenance of their executions, interconnections between workflows and related resources, for example datasets, publications, and so on. Their goal is to encapsulate knowledge and provide a mechanism for sharing and discovering assets of reusable research.

In [16] a mapping between the ROs and the context model (see Section 2.1) is presented. The authors conclude that the ROs are more focused on different types of artifacts and how they are aggregated to form new units of information, while the focus of the context model is shifted to technical aspects, such as precise information on the software setup and dependencies, data formats interlinked with format registries and aspects such as licenses and other legal issues. For this reason the context model is more suitable for settings when the original environment of the workflow must be described, preserved and later re-executed in a new environment.

Provenance traces are part of the ROs and are also used by the context model. Provenance is usually considered as a strong foundation for ensuring reproducibility, since its representation encompasses the necessary details allowing checking results, whether intermediary or final [17]. The development of workflow engines, particularly when applied to enacting reproducible scientific experiments, has been a strong driver in the development of provenance models [18]. As a result many provenance ontologies were designed, for example: Janus [19], OPM [20], or PROV [18]. We use the Janus ontology in our work, because the Taverna workflow engine supports export of provenance data to this format, however, other standards also can be used.

In [21] authors show how provenance traces can be extended with information about ”scientist’s intent” to depict the goals of the scientific workflow. Thus other researchers reviewing the workflow can understand motivation for its design. In our work we use the provenance traces to source the actual data for calculation of metrics that state whether the validation requirements are fulfilled, that is, whether two executions of the workflow produced the same results.

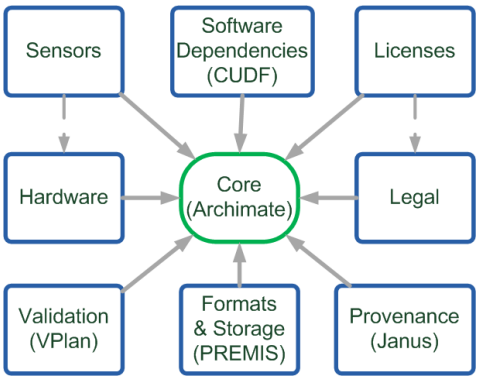


Figure 1: Overview of the context model: core ontology and extensions [22]

2.1. Software execution context

Underlying software and hardware needed to run the software can be referred to as dependencies and belong to the execution context.

We use the *context model*, described in [22], to store descriptive meta-data and documentation of the workflow and its environment. It is a meta-model designed according to the principle of modularity. The architecture is centred on a core model that is extended by other models, depending on the requirements of the application domain. The meta-model is implemented as an OWL ontology. The ontology mapping allows for a realisation of the model integration. This architecture is depicted in Figure 1.

The core model is based on the ArchiMate enterprise modelling language [23]. It includes concepts on business and technical layers ranging from services, process steps, and data exchanged between the steps, down to the technical infrastructure, including hardware, software, and files. An example extension is the ontology on software dependencies based on Common Upgradeability Description Format (CUDF) [24], which provides detailed modelling support for various relations, such as dependencies or conflicts between packages. In this paper the above described structure is referred to as context meta model, while instances of it are referred to as context model.

VisTrails is a cross-platform workflow system [25]. The authors postulate to package the whole environment into a virtual machine and suggest an external tool, which is not integrated with VisTrails, to be used for this purpose. They specifically suggest the tool CDE [26] implemented for Linux-based environments. CDE requires users to prepend CDE commands to scripts or binaries. CDE intercepts system calls and gathers all files and binaries that were used in the execution. However,

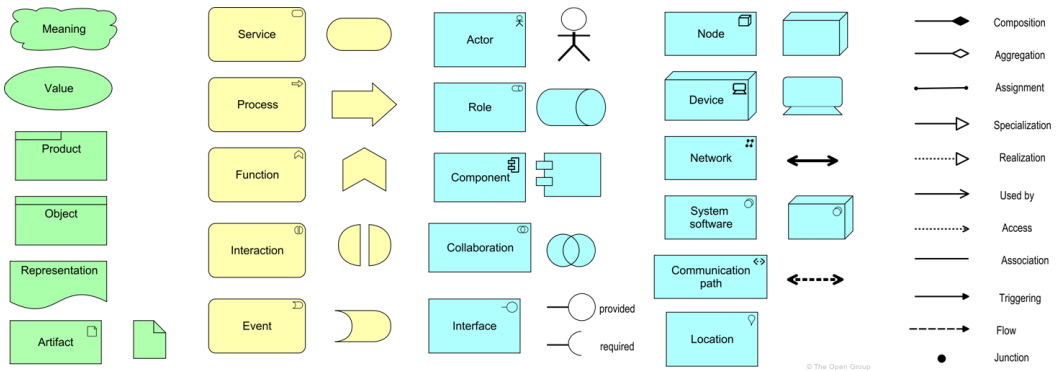


Figure 2: Overview of the ArchiMate concepts [23].

CDE is not capable of detecting external calls, for example to web services used in the workflow. Also local service applications that run in the background and are started before the workflow execution are not detected.

The process migration framework (PMF) [27] does not have these limitations and is capable of extracting a process from a Debian GNU/Linux based operating systems in which the process shares resources with other processes. The PMF saves the results into the context model [22]. PMF is based on the Linux tool strace[1], which logs all calls between a software program and the system. It records which programs (operating system processes) were called, what files were accessed or created, and which connections to other services were opened, for example connections to a local database or a remote web service. This data contains exact commands and their parameters, as well concrete addresses of services. Furthermore, it identifies which of the identified files are parts of the default operating system software sources, so-called Debian packages, and groups these together. This leads to a concise and semantically richer description of the actual dependencies of the workflow - instead of a flat list of potentially thousands of files, one obtains a much shorter list of packages. The PMF apart from creating the context model downloads packages that were identified in the system, as well as extracts files from the original system. Using this data, the context model, and Vagrant[2] it can create a new virtual machine that is configured in the same way as the original one was.

Santa-Perez et al. [28] use a similar approach to the PMF. They use ontologies and Vagrant for documenting workflow execution and its porting to virtual machines. The VFramework and the VPlan that are described in this paper can be used together with the approach proposed by Santa-Perez et al. to verify and validate the redeployments. Our approach formalizes and automates this process. It also enables identification of exact differences between environments and validates all data produced by workflow during processing - not only its outputs.

3. VFramework

In this section we describe the VFramework - a framework for verification and validation of scientific workflow re-executions. A first conceptual design of the VFramework has been outlined in [12]. Via a number of conceptual and, finally, fully implemented pilots the framework has been concretized, defining the individual steps and the data collected at each of them in detail. In this paper we focus on ontologies enabling comprehensive description of the workflow and its execution environment. We show what information needs to be captured and in what way modelled.

In the remainder of this section we provide an overview of the framework structure and explain in which settings the framework can be used. We also describe a workflow that we use as a running example on which we demonstrate how we create and integrate ontologies that we use for verification and validation of workflow re-executions.

3.1. VFramework overview

The design goals for the VFramework go beyond the mere re-execution within relatively short time frames, but include digital preservation settings in which workflows potentially need to be re-executed at much later points in time in different computational environments,

[1] http://sourceforge.net/projects/strace/
[2] https://docs.vagrantup.com/v2/boxes.html

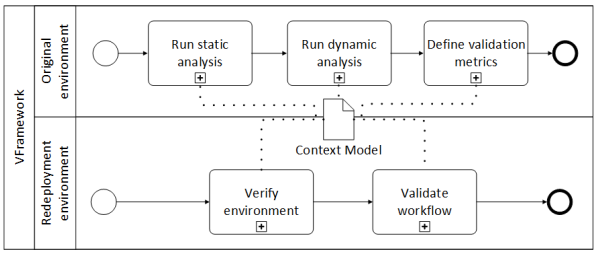


Figure 3: Framework for verification and validation of re-executed workflows.

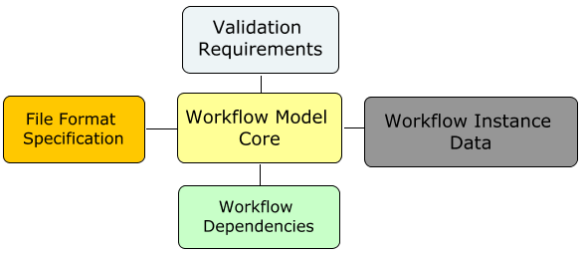


Figure 4: Overview of the context model parts instantiated during the VFramework application.

when the original workflow may not be executable any more. These long-term considerations, however, turn out to be essential even in short-term re-execution settings due to the rapid change of the underlying technology at the hardware and software level.

A workflow running in the original environment can be extracted and moved either to a new environment or preserved in a repository. Such a new environment is often referred to as the redeployment environment. A workflow can be also moved to a repository, because the project in which the workflow was engineered came to an end, and due to the research funder regulations all data including workflows must be preserved. The preserved workflows can be potentially run at a later point in time, for example, in a litigation case when the correctness of the original process has to be proven. In such cases the original system may not exist anymore. Therefore, we have to ensure that contemporary access to both the original and a new execution environment is not necessary, as this is not feasible in re-execution settings. The preferred way to compare these two executions is to collect data from the original environment and to use it as a reference in the environment in which the workflow is re-executed.

These requirements are reflected by the VFramework which is depicted in Figure 3. It consists of two sequences of actions. The first one collects information about the original execution of the workflow in the original environment. The second one uses this information to verify and validate the re-execution of the workflow in the redeployment environment. The context model (see Section 2.1) stores information collected in the first and provides this information in the second phase. The first three steps of the framework, which are performed in the original environment, are gradually adding information to the context model. Later this information is processed using tools to verify and validate the workflow re-execution.

For verification we compare the context model of the original execution with the context model of the re-execution. Thus we identify whether the executions were obtained in a compliant way, that is, whether the same resources were used during both executions. The identity of environments is not required for workflows to produce valid results, but if any discrepancies in workflow re-executions are detected, then the list of differences between environments (produced by comparing context models) allows identifying potential cause.

For validation we use the data captured during workflow execution that was collected for both executions. We compare it using appropriate metrics depending on its data format. We validate not only workflow inputs and outputs, but also intermediate data exchanged between workflow steps.

Figure 4 depicts five parts of the context model that are created during the VFramework application:

- *workflow model core* - the central element that describes the workflow model and is a basis for further extensions (see Section 4),
- *workflow dependencies* - description of software and hardware components used by the workflow (see Section 5),
- *workflow instance data* - data used, processed and produced by the workflow during its execution (see Section 6),
- *file format specification* - identified file format for the workflow instance data (see Section 7),
- *validation requirements* - requirements and metrics used to validate the re-execution (see Section 8).

We use the colour coding throughout this paper to distinguish between the parts of the context model, for example, we use the light green colour to depict elements that are workflow dependencies. All presented ontologies can be found online[3]

[3]https://opensourceprojects.eu/p/timbus/context-model/ontologies

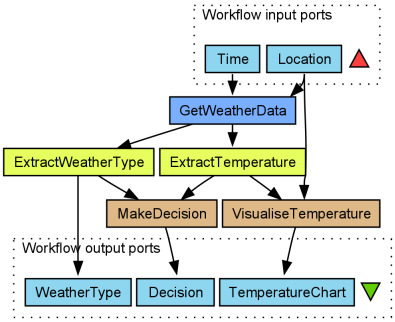


Figure 5: Weather workflow used as the running example.

3.2. Running example

In this section we present a workflow that we use as a running example to illustrate how the parts of the context model are instantiated during the VFramework application. In the remainder of this paper we refer to it as "the weather workflow".

We created a Taverna workflow presented in Figure 5. It tells whether the weather conditions for a given location are appropriate for outdoor sports and visualises the temperature in a chart.

According to Jim Gray [1] the scientific computational research includes tasks ranging from "data capture and data curation to data analysis and data visualization". The weather workflow also performs such tasks in a following way:

- Data capturing - a weather forecast is obtained from a weather web service.
- Data transformation - weather data is filtered and processed to make a decision whether the conditions are appropriate for outdoor sports.
- Data visualisation - the temperature for a given location is presented in a chart.

A detailed study of workflows published in a public research portal myExperiment is presented in [10]. The authors identified that almost 75% of scientific workflows are Taverna workflows. The latest revision of Taverna specification, that is Taverna 2, constitutes 55% of all workflows. Among the Taverna 2 workflows almost 43% use web services. Furthermore, almost 50% of Taverna 2 workflows use Beanshells that allow users to write their own code in a lightweight Java-like scripting language. For this reason:

- The weather workflow is a Taverna 2 workflow.
- It uses a REST web service to obtain weather data from a third party.
- It has two Beanshell scripts that implement data transformation and visualisation tasks.

The context model which was gradually extended during the application of the VFramework in the original environment is depicted in Figure 6. It contains all five parts of the context model (cf. Figure 4). In Sections 4 - 8 we explain how this model was created and what decisions influenced its design. The presented examples of ontologies and models can be found online[4].

4. Workflow model core

Workflows can be executed within a dedicated workflow engine, but even this does not imply that workflows are separated from the environment in which the workflow engine runs. In fact workflows can have interactions with other software that is running on the same system, for example, by connecting to a database server. Therefore, often before the workflow is started, the other services must be enabled, or they are started during workflow execution. All these additional services, on which the workflow depends, belong to the workflow's context and must be captured in the original environment.

Knowing this context we can define workflow boundaries that specify which of the identified dependencies are integral part of the workflow, that is, which of them are within the workflow boundary. Such dependencies must be verified and validated to confirm workflow replicability. The dependencies that lay outside of the workflow boundary, so called external dependencies, are also needed for the workflow to execute, however, they are not an integral part of the workflow. Their availability and conformance must be checked, but there is no need (and no means) to verify and validate them. Hence the analysis of workflow model core allows us to identify which dependencies must be present during the workflow execution and for which of them we must collect data that enables their verification and validation.

During the VFramework application we source information in different ways: we monitor workflow execution, collect provenance traces, or analyse processed data to identify file formats. We also need to know the model of the workflow to be able to assign collected information to the workflow's steps or outputs. The workflow model core allows modelling common workflow

[4]https://doi.org/10.5281/zenodo.159927

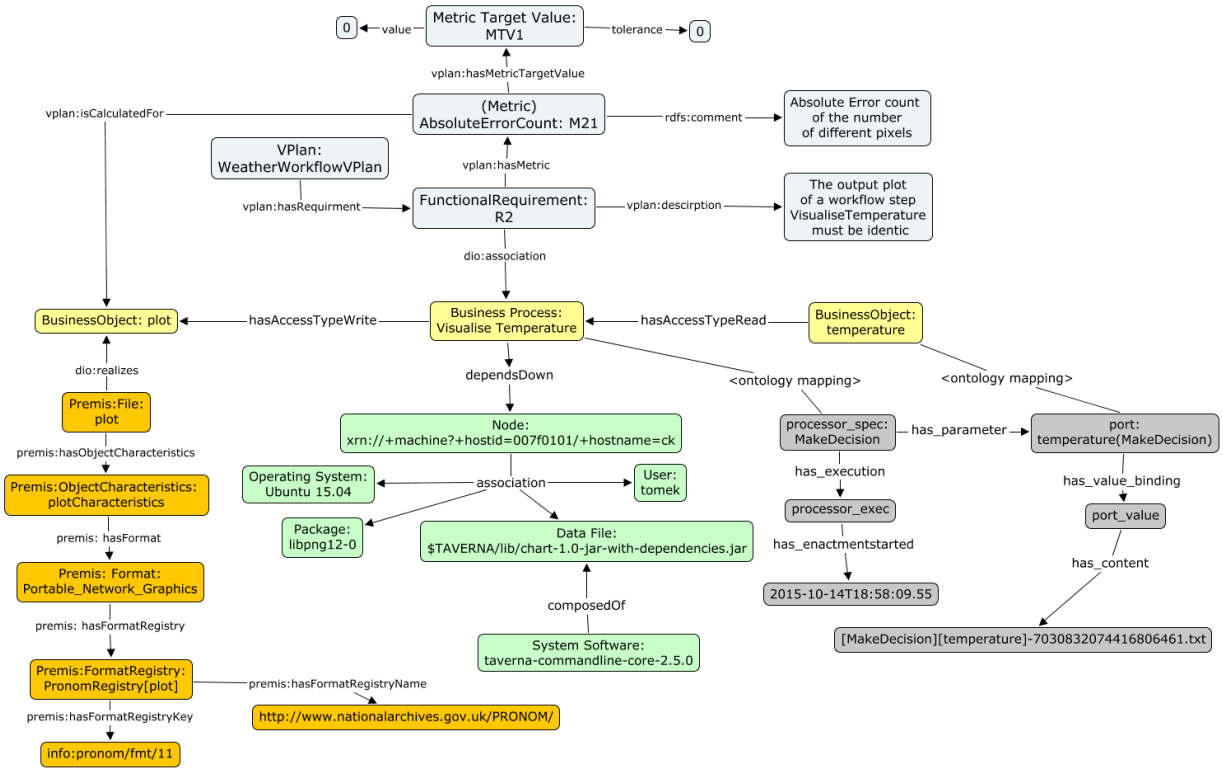


Figure 6: Overview of the context model parts instantiated during the VFramework application - a detailed view. Colour coding: yellow - workflow model core, green - workflow dependences, orange - file format specification, light blue - validation requirements, grey - workflow instance data.

concepts like steps, inputs, outputs, etc. For this reason, when transforming the original workflow representation into the workflow mode core we extract only the information that can be covered by the concepts used in the workflow model core. Hence, we do not aim for the full mapping of concepts between models, but focus only on those which are essential in further processing. Our intention is not to replace other workflow specifications.

The workflow model core[5] is the OWL representation of the ArchiMate enterprise modelling language. We use the Taverna2Archi[6] and the Archi2OWL[7] converters to automatically convert the specification of a workflow into the core ontology of the context model (workflow model core). The first tool reads Taverna workflow definition file and generates ArchiMate model, while the second one transforms the model into an OWL ontology. For other than Taverna workflows this is done manually using Archi[8] modelling tool, or directly in an ontology editor, for example, Protege[9].

The process of converting workflow specification into an OWL ontology is fully automatic and does not require users to work with the ArchiMate models. We present the ArchiMate model, which is the intermediate product of conversion, to better explain the structure and the concepts used in the workflow model core.

Figure 7 depicts an excerpt of an ArchiMate model created for the weather workflow (see Figure 2 for an overview of ArchiMate concepts). It models the *GetWeatherData* step and consists of three layers, starting from the top: business layer (yellow), application layer (light blue) and infrastructure layer (green). Each of them provides a different perspective on the workflow.

The business layer describes the workflow from a high level perspective. It specifies workflow inputs, steps and outputs, as well as describes relations between them. In the given example we can see that the *GetWeatherData* workflow step is modelled as a *Business Process*. It has two inputs which are modelled as

[5] http://bit.ly/2d9diH3
[6] http://www.ifs.tuwien.ac.at/dp/process/projects/tavernaExtractor.html
[7] http://www.ifs.tuwien.ac.at/dp/process/projects/archi2OWL.html
[8] http://www.archimatetool.com
[9] http://protege.stanford.edu

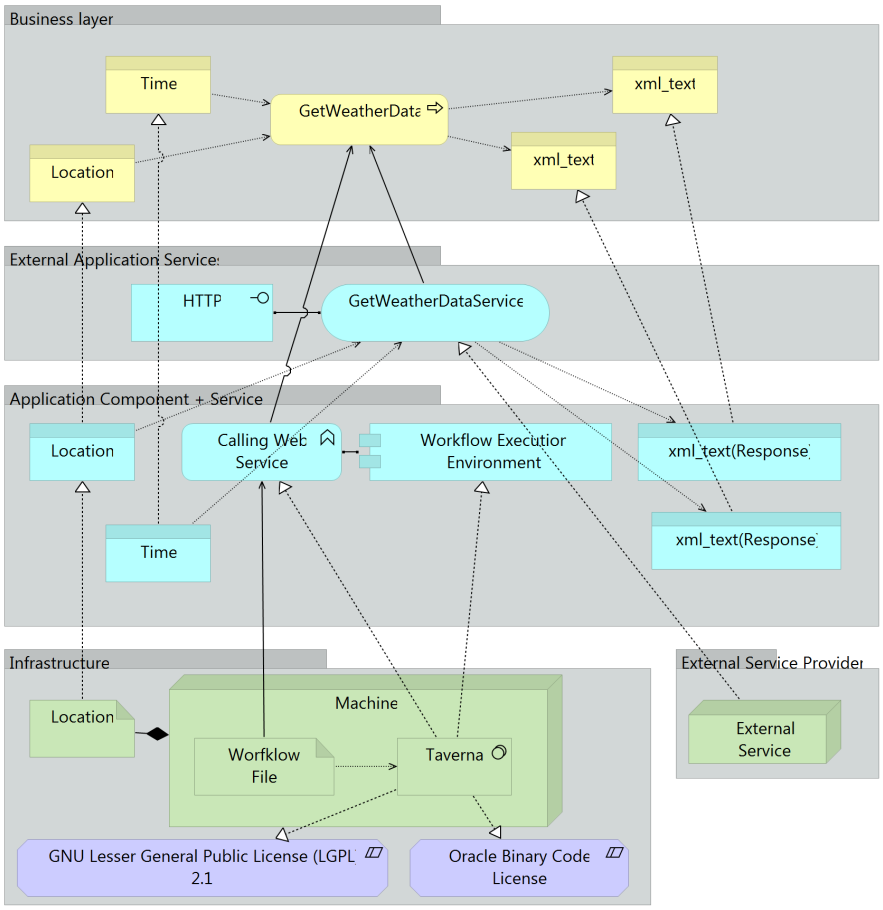


Figure 7: Excerpt of an ArchiMate model created for the weather workflow.

Business Objects and are linked with the *Business Process* using *hasAccessTypeRead* relation. The outputs are also modelled as *Business Objects* and linked with the *Business Process* using *hasAccessTypeWrite* relation.

The application layer describes functionalities and interfaces of software and hardware components that are used to run the processes described in the business layer. In the given example we can see that the *GetWeatherData* workflow step uses *GetWeatherDataService*, which is an *Application Service* that has an *HTTP Application Interface*. Furthermore, we can see that the *Workflow Execution Environment* has an *Application Function* which is *Calling Web Service*, which is *used by* the *GetWeatherData* workflow step. We can read from this that the *GetWeatherData* workflow step is performed by communicating with a REST web service. If it was a WSDL service, the model would look like slightly different, for example, the SOAP *Data Object* would be depicted.

The infrastructure layer makes it concrete which software and hardware is used to provide the functionality and services specified in the application layer. In the given example we can see that there are two *Nodes* used to perform the *GetWeatherData* business step: *Machine* and *External Service*. The first is the local machine on which the workflow is executed, while the second represents the machine on which the REST web service is deployed. The web service is beyond process boundaries, because it is hosted on a machine that is beyond our control. Furthermore, we can see that the *System Software*, which is *Taverna* is used to realize the *Workflow Execution Environment* and also to provide the functionality allowing for *Calling Web Service*. *Taverna realizes* two *Constraints* that depict licenses applying to it. We can also notice in the figure that the *Artifact Location realizes Data Object Location* which *realizes Business Object Location*. Redundant as it may seem, it is the correct way to represent data read or written by a workflow. In case of other *Business Objects* depicted in the figure, for better readability we did not depict all their

```
ASK WHERE {
?bp a dio:BusinessProcess.
?appFun dio:usedBy ?bp.

?appFun rdfs:label ?appFunLab.
FILTER (regex(str(?appFunLab),
"Calling_Web_Service"))

?appSer dio:usedBy ?bp.
?appSer dio:assignment ?appInt.
?appInt a dio:ApplicationInterface.

?appInt rdfs:label ?appIntLab.
FILTER (regex(str(?appIntLab), "HTTP"))
}
```

Listing 1: SPARQL query checking whether the workflow model specifies any REST web services.

connections to the lower layers. The infrastructure layer is further extended in the next step of the framework by linking identified software dependencies accessed during workflow execution.

The benefit of using the ontology representation of the presented model is the possibility of querying the model. Thus without manually working with the model we can:

1. Identify whether the workflow has external dependencies, like for example web services.
2. Check if any Beanshell scripts declare usage of external Java libraries.
3. List licenses applying to the workflow.
4. List workflow steps, as well as their inputs and outputs.

To query the model we use a set of ten pre-defined SPARQL queries. Listing 1 presents one of them. The query provides an answer whether the workflow model specifies any REST web services. Other queries can be found in appendix Appendix A The result of running this query is either true, when the workflow uses a REST web service or false, when it does not. This query was built taking into account the way the converter tool converts the ArchiMate models into the core ontology of the context model.

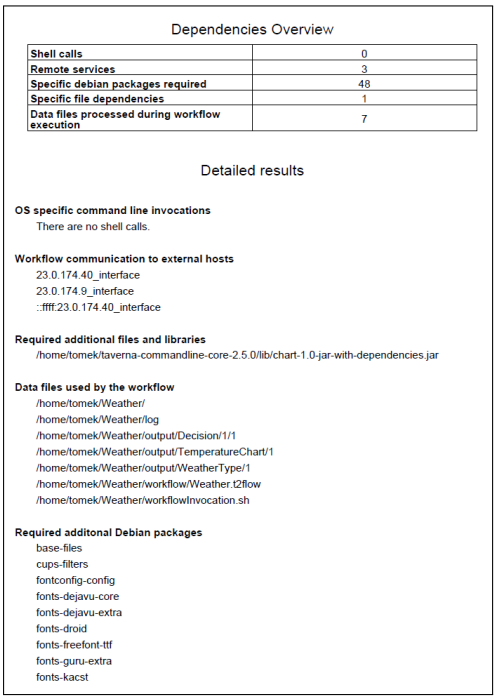
5. Workflow dependencies

Local dependencies of the workflow are all system components used during workflow execution. Examples range from software libraries imported by scripts, shell tools invoked from the workflow, to research area specific software tools. Also the specific version of the workflow engine, as well as the version of the operating system is important. In case of package based systems like Linux the exact version of packages installed must be documented. Due to the amount and the granularity of information needed, the manual description of local dependencies is limited. Modelling high level concepts like workflow engine, operating system or machine is possible, but their decomposition into fine grained information consisting of files, packages and their versions results in a rapid increase of artefacts that need to be modelled.

The external dependencies of the workflow are all components that are used to perform workflow steps but are not hosted on the same platform as the one executing the workflow. Web services are an example of such dependencies. During the V&V process it is essential to ensure repeatable conditions and therefore the dependencies must be validated. It may happen that a third party web service is not available any more, or it was upgraded to a newer version that delivers altered results due to a different computational algorithm. In such cases, the workflow executed using this service may either break or can produce altered results. A possible solution to circumvent this problem is to provide either a different service that is compliant with the original one, or to create a mock-up of the service that can provide correct responses for the requests generated using the test instances. In [29] we describe ways of creating such mock-ups.

We use the PMF (see Section 2.1) to monitor an execution of a workflow in the Taverna workflow engine. We monitor an invocation of command line version of Taverna that takes a workflow as one of its parameters. As a result we obtain an ontology describing workflow dependencies. We uses five predefined SPARQL queries to process the data from the context model and to present them in form of a dependency report. An excerpt of a dependency report for the weather workflow is presented in Figure 8. The report summarizes five main aspects of the workflow execution that must be verified:

- **Shell calls** When a workflow uses any tool that is installed in a system, it makes a shell call. The report lists all commands that were executed and provides a list of Debian packages and files that are used during the call. All such calls are beyond the control of the workflow engine and therefore pose a threat to the repeatability of workflow executions.
- **Remote services** If a workflow uses an external service, then the exact address of the service is provided. The IP address of the service is used to configure tools that intercept the traffic.
- **Specific file dependencies** These are all files that



Dependencies Overview

Shell calls	0
Remote services	3
Specific debian packages required	48
Specific file dependencies	1
Data files processed during workflow execution	7

Detailed results

OS specific command line invocations
There are no shell calls.

Workflow communication to external hosts
23.0.174.40_interface
23.0.174.9_interface
::ffff:23.0.174.40_interface

Required additional files and libraries
/home/tomek/taverna-commandline-core-2.5.0/lib/chart-1.0-jar-with-dependencies.jar

Data files used by the workflow
/home/tomek/Weather/
/home/tomek/Weather/log
/home/tomek/Weather/output/Decision/1/1
/home/tomek/Weather/output/TemperatureChart/1
/home/tomek/Weather/output/WeatherType/1
/home/tomek/Weather/workflow/Weather.t2flow
/home/tomek/Weather/workflowInvocation.sh

Required additonal Debian packages
base-files
cups-filters
fontconfig-config
fonts-dejavu-core
fonts-dejavu-extra
fonts-droid
fonts-freefont-ttf
fonts-guru-extra
fonts-kacst

Figure 8: Excerpt of dependency report for the weather workflow.

are used by a workflow but cannot be attributed to any of the installed packages. Usually, these are all sorts of configuration files and additional libraries that might have been installed manually in the system.

- **Data files processed by the workflow** These are all data files that were either input files of the workflow or were created during workflow execution, for example, temporary files, logs, or workflow outputs. Identified data files which are not part of the provenance traces are later used for validation.

- **Specific Debian packages** The list contains all Debian packages that must be installed in the system, when executing the workflow. The packages are a superset of packages identified for each shell call and are presented for a better overview of requirements.

The dependency report is easy to read by a human, but presents only a limited view on the workflow context. The report is useful when installing the workflow in a new environment to get an overview of workflow's interactions. However, for verification we use the ontology that was created by the PMF (see Figure 9). It describes *workflow dependencies* part of the context

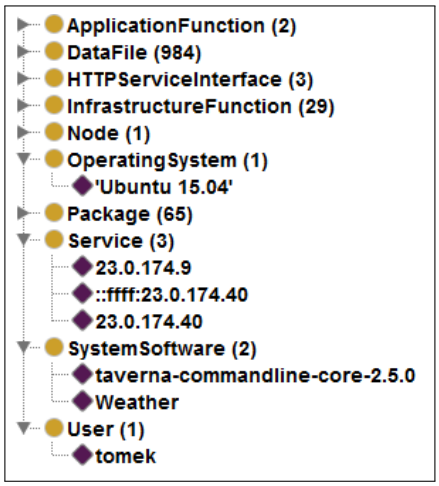


Figure 9: Workflow dependencies of the weather workflow.

model and consists of following ontologies: CUDF[10], Software DSO[11], Service DSO[12], and Security DSO[13]. An overview of ontologies is presented in Table 1. The *workflow model core* concepts are also depicted in Table 1, because we also use them in modelling. However, note that the instance of *workflow model core* created during the static analysis is not yet linked.

We integrate the *workflow model core* instance with the *workflow dependencies* instance by source level linking of persistent unique identifiers [30]. Each of the models contains a *Node* individual that represents a machine for which the analysis was performed. The tools generate always the same unique identifier for a given machine. Figure 10 depicts an excerpt of integrated context models for the weather workflow. The elements depicted in yellow were created during the static analysis of the workflow model. We can see that the *Visualise Temperature* workflow step has an input *temperature*, and an output *Temperature Chart*. The elements discovered during dynamic analysis are depicted in green. The workflow was run on a machine that has *Ubuntu 15.04* operating system, the user was *tomek*, and a file *chart1.0-jar-with-dependnecies.jar* was used in the experiment that was run using *taverna-commandline-core-2.5.0*. The *Node* element is common for both context models and thus integrates them.

[10] http://bit.ly/2driu8w
[11] http://bit.ly/2dCxXpL
[12] http://bit.ly/2d9tBDK
[13] http://bit.ly/2d9sFPQ

Table 1: Ontologies used to describe workflow dependencies.

Ontology	Description	Classes used
workflow model core	Provides high-level concepts describing technical implementation of the workflow (see Section 4).	System Software, Node, Infrastructure Function, Application Function
CUDF	Describes (Debian) Packages and dependencies between them. Based on CUDF [24].	Package
Software DSO	Domain Specific Ontology (DSO) used to distinguish static files from running software.	Data File, Operating System
Service DSO	Describes Services, their Endpoints and Protocols.	Service, HTTPServiceInterface
Security DSO	Vocabulary of terms used in computer security.	User

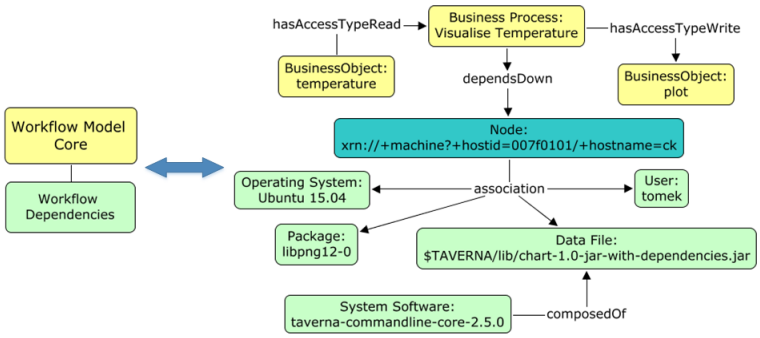


Figure 10: Excerpt of the integrated context models instantiated by static and dynamic analysis. Colour coding: yellow - workflow model core, green - workflow dependences.

6. Workflow instance data

Workflows differ in the way they produce data. This depends on their implementation. Some of the workflows process data inside the system memory and in the final step write data to the hard disk. Some other also save intermediate results to the disk and thus enable tracing workflow execution and validation of intermediate data. There are also workflows that write the result to the standard output and create no files. Various tools can be used to collect the data either from the disk, system memory or from specific system interfaces, like for example network interfaces. The problem of capturing workflow instance data was recognized by the workflow community and addressed by integration of provenance collection mechanisms that are built-in mechanisms of workflow management systems. They enable automatic capturing of data produced during execution.

The provenance traces which are saved as the result of provenance capturing contain data used during workflow execution. They became part of Research Objects [11] and are shared together with workflows to document the obtained results. Taverna allows exporting provenance using the Janus [19]. We also use this ontology to model workflow instance data.

We capture provenance by running Taverna in a mode that saves provenance data of a workflow execution. Taverna stores the input and output data of each workflow step in a local database. We use the Provenance Extractor[14] to extract this data and export it in the Janus ontology format. We further extend it by adding more information enactment timing, that is, for each step we know when its execution has started and finished.

Figure 11 depicts excerpt of provenance traces for the weather workflow. The elements depicted in green are data properties, the other elements are Janus individuals. Each individual has a class and a label, for example the *workflow_spec* is the class and the *WeatherExample* is the label. For better readability we did not depict in the figure labels which are automatically generated unique identifiers.

[14] http://www.ifs.tuwien.ac.at/dp/process/projects/provenanceExtractor.html

We can read in the figure that the *Make Decision* workflow step was modelled using the *processor_spec* class. The model depicts that this step is a *BeanshellActivity* and provides exact timestamps when its execution was started and ended. This is done by linking individuals of class *processor_spec* to the individuals of class *processor_exec* using *has_execution* object property. The workflow step *Make Decision* has also input called *temperate(MakeDecision)*. This is represented by an individual of a *port* class. The data which was captured for this workflow input is referenced by the individual of a class *port_value* which is linked to the individual of a *port* class using *has_value_bindining* object property. The actual name of the file in which the data is stored is provided in a data property of an individual of *port_value* type.

Model – Instance Integration We need to establish a link between the workflow model (*workflow model core*) and the workflow instance (*workflow instance data*), so that we know which data was collected for which workflow steps.

Ontologies can be integrated through a mapping that specifies relations between the components of two ontologies. Hence, the mapping specifies only how the information stored in two separate ontologies relates to each other, but do not merge them. In our case, the mapping describes how the workflow model relates to the workflow instance.

The ontology mapping results in a low coupling of ontology instances. This enables us to link dynamically multiple workflow instances to a single workflow model without the need of defining additional context model extensions for the management of workflow instances. Furthermore, it follows the principle of modularity that is the main design principle of the context model.

We identified a mapping of concepts between the workflow instance (Janus ontology) and the core ontology of the context model: *port* is equivalent to *Business Object*, both *workflow_spec* and *processor_spec* are equivalent to *Business Process*. The fact that both the *workflow_spec* and the *processor_spec* classes map to the *Business Process* does not result in loosing any information that is need during the VFramework application. This is because the Archimate specification assumes that the business process can contain subprocesses that are particular process steps. According to the Archimate both the process and its steps should be modelled using the *Business Process*.

The provenance traces of the workflow execution need to be integrated with the workflow model taking into account this mapping and observing the naming convention used to label the individuals of the Janus ontology.

Figure 12 depicts the workflow model integration with the provenance traces. It links the complete Janus ontology with the rest of the workflow context model. For presentation purposes we did not depict all of the Janus elements that were visible in Figure 11. In Figure 12 we can see that the workflow step *Make Decision* has the input *temperature*. The data collected for this input is stored in *[MakeDecision][temperature]-7030832074416806461.txt* file, which is also part of the provenance. For the workflow step we can read the date and time when the execution of the workflow step was started. Besides this, we can also observe in what way the names of elements describing the workflow model (business step and its input) correspond to the labels used in the provenance traces. The following pattern is used:

- The label of the *processor_spec* is identical with the name of the *Business Process*.
- The label of the *port* is a concatenation of names of the *Business Object* and the *Business Process* that are connected using either *hasAccessTypeRead* or *hasAccessTypeWrite* relation.

We use this observation to automatically generate SPARQL queries. First, we query the workflow model to get a list of *Business Processes* and *Business Objects*. Second, we construct SPARQL queries using a template that is filled with results of the previous query. Finally, we execute the constructed queries on the Janus ontology. Figure 13 depicts one of the generated queries for the weather workflow. We incorporated this mechanism into a tool used for comparison of provenance traces during validation.

7. File format specification

We also use the provenance traces to enrich the context model with information on the data format of each of the outputs of the workflow steps. As explained in Section 8, this information facilitates the validation process by allowing choosing a correct data comparison tool. We obtain this information by running the DROID[15] characterisation tool that performs automated identification of file formats. It uses information from the PRONOM [31] file format registry and it can identify over 250 file formats, with more formats being added continously. We use the PREMIS ontology [32],

[15] http://sourceforge.net/projects/droid/

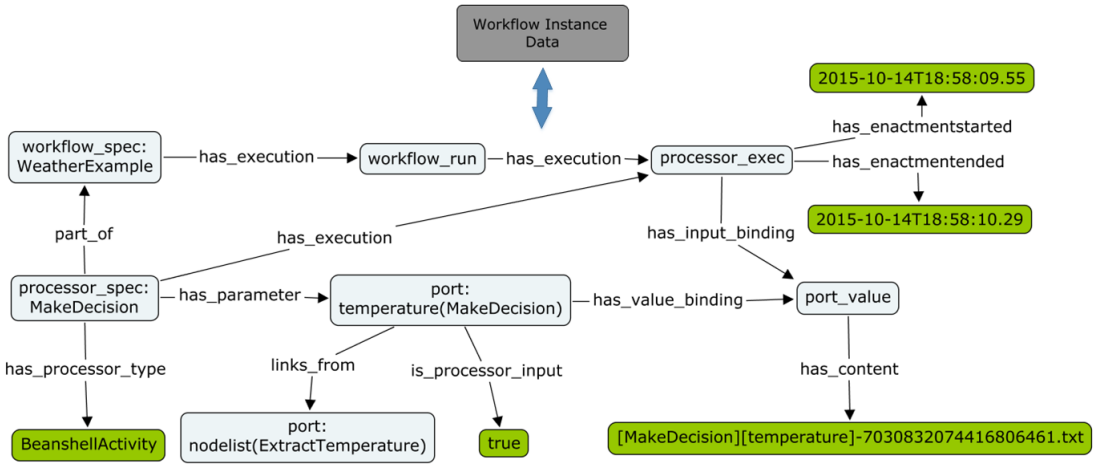


Figure 11: An excerpt of the Janus ontology depicting provenance of the weather workflow.Colour coding: grey - workflow instance data.

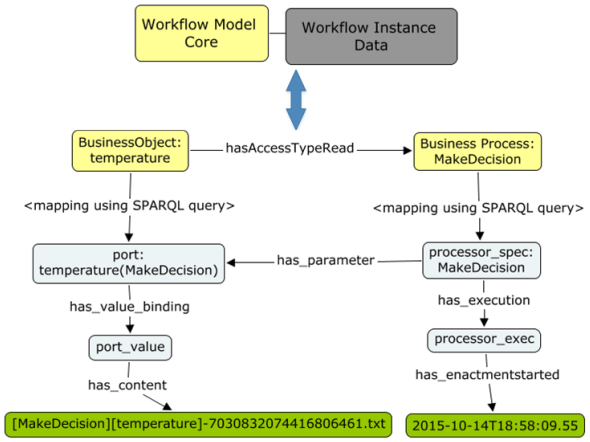


Figure 12: Integration of the provenance traces and workflow model using SPARQL queries on the example of the weather workflow. Colour coding: yellow - workflow model core, grey - workflow instance data.

which is an international metadata standard widely used in the digital curation domain, to store the format information and to extend the context model.

The integration of the PREMIS ontology and the workflow context model is depicted in Figure 14. We use the weather workflow to demonstrate in what way the information on file format is added to the output of a workflow step. The workflow step *Visualise Temperature* has the *plot* output that is modelled as a *Business Object*. The PREMIS information is linked to the *Business Object* using *realizes* relation that is always used to link objects that provide more concrete information [23]. The provenance data is a physical file for which the format analysis was performed, hence the PREMIS ontology models it as a *File* that *has Object Characteristics* for which a *Format* is one of possible characteristics. The information on file formats is stored in a central registry, therefore the address of a registry, as well as a specific key value for an identified format is provided. In the provided example, the *Portable Network Graphics (PNG) Format* has a *pronom/fmt/11* key in the *PRONOM* registry.

8. Validation requirements

In this section we discuss what influences specification of validation requirements, how they can be quantified and which data can be used for this purpose. Based on this analysis we present a thoroughly redesigned version of the VPlan ontology that was originally published in [14].

SPARQL query:

```
PREFIX janus: <http://purl.org/net/taverna/janus#>

SELECT ?procspec ?port (str(?content) as ?provenanceFile)
WHERE {
?procspec a janus:processor_spec.

?port janus:has_value_binding ?binding.
?binding janus:has_content ?content.

?procspec rdfs:label ?procspec_lab.
?port rdfs:label ?port_lab.

FILTER (regex(str(?procspec_lab), "MakeDecision"))
FILTER (regex(str(?port_lab), "temperature.MakeDecision"))
}
```

procspec	port	provenanceFile
MakeDecision	'temperature(MakeDecision)'	"[MakeDecision][temperature]-6681710419859368316.txt"

Figure 13: Example of a SPARQL query that integrates workflow model with the provenance information for the weather workflow.

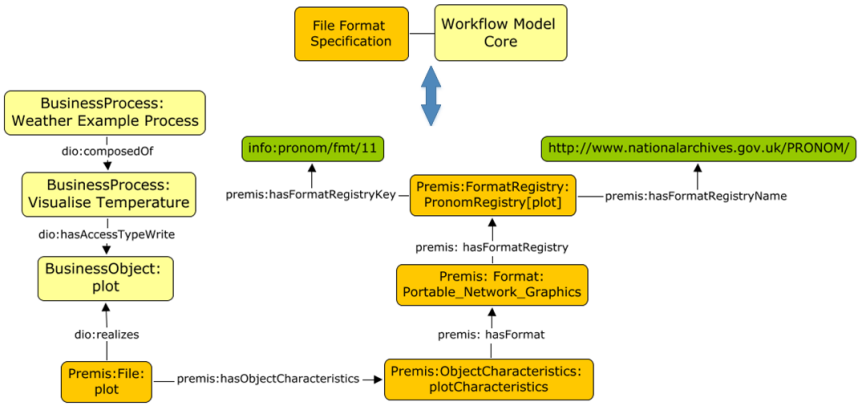


Figure 14: Extension of the context model with information on file formats using PREMIS ontology for the weather workflow. Colour coding: yellow - workflow model core, orange - file format specification.

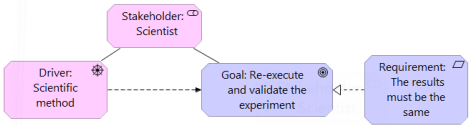


Figure 15: Top down analysis of validation requirements taking into account the scientific method.

In Section 8.1 we discuss what influences the specification of validation requirements and in what way we can express them. In Section 8.2 we provide a detailed specification of the VPlan ontology that we use for extending the *workflow mode core* with *validation requirements*.

8.1. Define validation metrics

To define a universal set of requirements that can be used for validation of workflow re-executions we performed a top-down analysis taking into account the scientific method, and a bottom-up analysis of selected workflow examples.

In the top-down analysis we took the scientific method as the starting point of our analysis. One of its main principles is that the scientific experiment must be repeatable [33]. Hence, the goal of each scientist is to build such experiments that can be re-executed and validated for providing the same results as the original experiment. We depicted this in Figure 15 using ArchiMate extension for modelling motivational concepts. The scientist is a *stakeholder* for whom the scientific method is a *driver*. The *stakeholder* has a *goal* that is to re-execute and validate the experiment. The *driver influences the goal*. Hence, the *goal is realised by a requirement* which states that ”The results must be the same”. The requirement is phrased according to the RFC 2119 [34] standard for expressing the requirements. The finest granularity of available data that can be captured in the workflow system is its single ”com-

ponent", that is, a workflow step[16]. In terms of effort it is possible to check all of them. For this reason we modify the requirement and phrase it: "The results of each workflow step must be the same".

We performed a bottom-up analysis by investigating 7 workflows from 4 different domains, namely: three workflows from the digital preservation domain[17][18][19], one workflow from the music information retrieval domain[20], one workflow from the sensor data analysis domain [40], and two workflows from the medical research domain[21]. The workflows resemble characteristics of most common workflows as identified in [10].

For each of the workflows we brainstormed to define requirements that validate whether the workflow reexecution was identical (or similar within an acceptable range) with the original one. To measure the similarity of workflow executions we defined metrics that we derived from measurements taken during both executions. We took into account the available provenance data to identify which data is available and can be used to calculate these metrics.

The analysis of defined requirements revealed that all functional requirements deal with the correctness of a single workflow step execution and the best way to validate it was to check each of its output ports. There were also requirements dealing with the correctness of workflow outputs, but these are a subset of requirements expressing that each output of a workflow step must be the same. There was only one non-functional requirement that the computation time shall be similar.

Furthermore, we noticed that the comparison of data must take into account the format of the data and therefore must be made using appropriate tools. For example if two PNG images depicting the same phenomenon are compared by computing a hash value, they can be detected as being different due to different metadata descriptions. A correct way to perform this comparison is to compare the features of the images using software for image analysis. In case of textual data formats using the right comparator also has an influence on the validation result. For example, two XML documents having different ordering of elements, but otherwise containing the same data, using a simple text comparison would detect discrepancies, while a dedicated XML comparator ignores the ordering and thus confirms the equivalence. There are also documents that contain headers or comments in which a generation time-stamp is provided. For these the comparison should focus on the actual data and ignore the time-stamps.

Based on the top-down and the bottom-up analyses of workflows, we conclude that:

- For each workflow step we can generate a functional requirement which states that the results produces by the workflow step must be the same.
- For each workflow step we can generate a non-functional requirement which states that the step execution duration shall be similar.
- Each functional requirement can be evaluated by calculating metrics for each of the outputs of the corresponding workflow step.
- The data must be compared taking into account the format of the data.

The requirements cover both functional and non-functional aspects and each of them is broken down into quantifiable metrics. The requirement is only fulfilled when all metrics associated with it are fulfilled. We use the metrics to quantify the distance between two workflow executions using measures derived from captured data for each execution. For example, in the weather workflow for the requirement "The output *plot* of a workflow step *Visualise Temperature* must be identical", we use the absolute error count of the number of different pixels for each channel to measure how far two PNG images differ.

However, providing just a metric is not sufficient, because we do not know which value of metric results in fulfilling the requirement. In the above example, we can suspect that when the absolute error count is zero, the requirement is fulfilled. Furthermore, if we allow alterations in the workflow execution, for example rounding errors when comparing two floating point numbers, then we need to specify a metric target value that fulfils the requirement. For example, we can use the Euclidean distance metric to compare two floating point numbers and specify the acceptable tolerance to 0.1. Thus we accept the Euclidean distance to be within this range. For example, for numbers 1.069 and 1.1 the Euclidean distance is 0.031 and is lower than 0.1. The metric value is within the accepted tolerance and the requirement is fulfilled.

Each metric also must be assigned to a measurement point which is a place in a workflow in which the data used for metric calculation is captured. For example, the

[16] https://taverna.incubator.apache.org/introduction/why-use-workflows

[17] http://www.myexperiment.org/workflows/2637.html

[18] http://www.myexperiment.org/workflows/3212.html

[19] http://www.myexperiment.org/workflows/3492.html

[20] http://www.myexperiment.org/workflows/3626.html

[21] http://www.myexperiment.org/workflows/3921.html

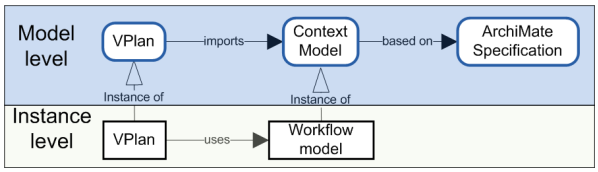


Figure 16: Relations between the models and instances of the context model and the VPlan.

data of a PNG plot looks differently in a file on a hard disk, than in the system memory when it is decoded or at the analogue output of a graphics card. For this reason, we must specify where the data used for validation is captured. The data must always be captured in the same way. For Taverna workflows we use the files that are part of the provenance traces or were identified as data files accessed by the workflow.

To recap we identified that during validation of workflow re-execution:

- For each requirement a quantifiable metric must be defined.
- Each metric must also specify its target value and allowable tolerance.
- The metrics must be computed using data captured always in the same way.
- The data comparison and metric computation process must take into account the format of data.

8.2. VPlan

In this section we present the VPlan ontology that we created for description of validation requirements. In Section 8.1 we analysed how to describe validation requirements and measure them for Taverna workflows. In this section we present how to express this information using the VPlan and in what way the validation of workflow re-executions is automated by generation of requirements and metrics using a controlled vocabulary that is a part of the VPlan model.

8.2.1. Overview

According to the IEEE 1012 standard [35] and good practices, it is essential to document the verification and validation process, so that the criteria used to perform the assessment are clear and well understood and that the decisions made can be traced at any time. For this reason, we document the requirements and data collection process using the VPlan ontology that is an ontology extending the core of the context model. Such integration facilitates the validation process, because additional information on the data captured for a specific

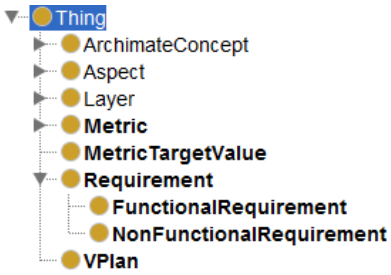


Figure 17: Overview of VPlan classes. Subclasses of the *Metric* are depicted in Figure 21.

workflow step can be obtained from the context model, for example, a file format of the workflow output can be read from the context model and used to choose a suitable way of comparing the captured data.

Figure 16 depicts the VPlan and the workflow context model in a layered view. It shows their relations at the model and instance level. We distinguish between the VPlan model and the VPlan instance:

- The VPlan model describes available classes and relations between them. It imports the context model to reuse some of its relations to link the VPlan classes, for example, the *association* relation. Thus we reduce the complexity of the model by reusing concepts with well-defined semantic meaning. This structure is reflected by the class hierarchy of the VPlan model that is depicted in Figure 17. It contains classes from the imported core ontology of the context model: *Archimate Concept*, *Aspect*, *Layer* and also classes of the VPlan model: *Metric*, *Metric Target Value*, *Requirement*, and *VPlan*. The *Requirement* class has two subclasses: *Functional Requirment* and *Non Functional Requirement*. The*Metric* class also has subclasses that constitute a controlled vocabulary. They are not depicted in this figure. We present them in Figure 21 and describe in Section 8.2.3.
- The VPlan instance describes particular validation requirements for a given workflow and links a selected subset of metrics to the requirements. The VPlan instance uses the workflow model, which we created during the VFramework application, to define measurement points and to locate data needed for computation of metrics. We describe this integration in Section 8.3. In the remainder of this section whenever we refer to the VPlan we mean the instance, if not stated otherwise.

8.2.2. Classes and their properties

In this section we describe the VPlan classes and their properties. We first present their overview by describing the purpose of each class and listing applicable properties, then we explain on the weather workflow how to use them to formulate validation requirements.

The VPlan classes and properties are:

- ***VPlan*** - root of the ontology (class)
  - *hasRequirement* - links the root to the requirement (property)
- ***Requirement (Functional Requirement / Non Functional Requirement)*** - specifies the validation requirement
  - *association* - links the requirement to the part of the workflow model for which the requirement was specified
  - *description* - stores the description of the requirement
- ***Metric*** - a distance measure to quantify the associated requirement
  - *isCalculatedFor* - specifies the measurement point by linking to the part of the workflow in which the data for calculating the metric is captured
  - *hasMetricTargetValue* - links to the metric target value
- ***Metric Target Value*** - an indicator for the associated metric stating whether its value fulfils the requirement
  - *value* - specifies the value of the metric for which the requirement is fulfilled
  - *tolerance* - specifies the acceptable range within which the metric can differ from the *value*

Figure 18 presents an excerpt of the VPlan for the weather workflow. We can see that the *VPlan* root element has two requirements: *Functional Requirement R2* and a *Non Functional Requirement R3*. Each of them has a data property *description* that express the requirement, for example, the *R2* requirement states that *The output plot of a workflow step Visualise Temperature must be identical*. The central part of the figure depicted in yellow represents the workflow model. Both requirements are linked to a workflow step *Visualise Temperature*. Hence, we know that both of them were formulated for this workflow step and if the validation requirement fails, then this means that the changes were detected in this step.

Any number of *Metrics* can be defined for a *Requirement*. In the given example we can see that for each *Requirement* one *Metric* was defined. The *Requirement R2 has metric Absolute Error Count*, and the *Requirement R3 has metric Execution Duration Ratio*. Each *Metric* has an annotation *comment* defined in the VPlan model that provides verbal explanation of the metric. For each metric we specified a measurement point using *isCalculatedFor* relation. For Taverna workflows we capture data at the outputs of workflow steps to compute metrics related to functional requirements. Therefore, we connected the *Absolute Error Count* metric to the *Business Object plot* that is the workflow step output of the *Visualise Temperature* step. The *Execution Duration Ratio* metric quantifies a non-functional requirement that deals with a computation time of a single step. Hence we need to monitor a single step, not an output. For this reason we connected the metric to the *Business Process Visualise Temperature*.

For each *Metric* we specify its target value, that is, a value for which the associated requirement is fulfilled. The *MTV1 value* is zero and its *tolerance* is also zero, hence, the metric *Absolute Error Count* must be zero, so that the requirement *R2* is fulfilled. In other words, the images produced in two executions of the workflow must be identical. In case of *MTV2*, its *value* is one and the *tolerance* is 30%. Thus the value of *Execution Duration Ratio* metric should be within the 0.7 and 1.3 range. This means that the actual computation time for a workflow step can differ up to 30% and the workflow execution is still valid. As a result, the VPlan models not only strict requirements for identity of workflow executions, but also relaxed requirements that allow for validation of similar executions.

8.2.3. Controlled vocabulary of metrics

This section presents a controlled vocabulary of metrics that we use for breaking down validation requirements. The metrics are grouped into categories that are depicted in Figure 19. In the remainder of this section we explain in what way we formulate the metrics and how we derived them for each category.

In the majority of the VFramework applications the validation metrics will be automatically generated and therefore the identity of workflow executions will be tested. For this reason we defined for each metric category a subset of metrics that are discrete metrics, for example, *Page Number Equality* for *PDF Format Metrics*.

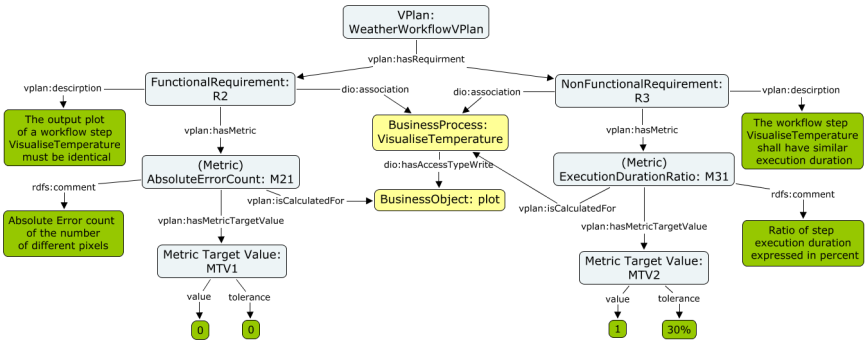


Figure 18: Excerpt of a VPlan instance depicting *Requirements*, *Metrics* and corresponding *Metric Target Values*. Measurement points are depicted using *isCalculatedFor* relation. Colour coding: yellow - workflow model core, light blue - validation requirements, dark green - data properties of the VPlan ontology.

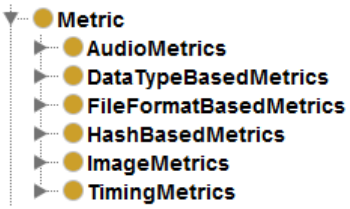


Figure 19: Categories of metrics defined in the VPlan.

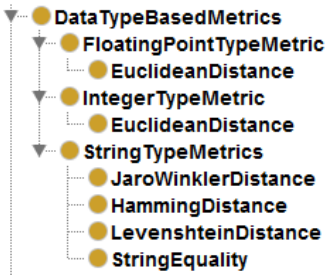


Figure 20: Categorization of data type based metrics in the VPlan.

The discrete metrics abstract the actual computation algorithm. Thus, they do not specify in what way the metric is computed, but allow all algorithms of a given class to be used, as long as they detect in a correct way the identity of objects. Let us explain it using an example. For comparison of strings we can use, for example, Jaro Winkler distance or Hamming distance. When we compare two different strings using these algorithms, both of them detect changes but quantify them differently. Each result can be transformed to provide a valid value for a discrete metric that is either zero when the strings match or one when they are different. Hence, the actual values produced by these algorithms are not important, as long as they can be made discrete. This results in a broader choice of software tools that can be used for automation of workflow re-executions validation.

We also devised non-discrete metrics that are used for validating executions similarity. They provide fine grained information on detected discrepancies. Furthermore, a tolerance that specify allowable deviations from the metric target value can be defined for them.

File format based metrics The analysis of sample workflows in Section 8.1 revealed that the metric computation process must take into account the format of captured data. We identified eight different file formats in the analysed workflows. Table 2 provides an overview of file formats, analysed software tools and metrics. Further file formats can be added to the VPlan.

Data type based metrics The workflow engine passes values between the steps outputs and inputs without documenting their data type. Hence the provenance traces do not contain information on the data type and by default all values are saved as strings in the text files. The DROID characterization tool analyses these files and detects file formats. For the files that were recognized as *Plain Text* format we perform additional analysis of their datatypes and look for integer and floating point numbers. Thus we can use a set of metrics that better fits the data type.

We performed a literature review to define a set of metrics for strings, integers and floating point numbers. In [36] authors analysed metrics that can be used for computation of distance between strings, while authors of [37] provide a dictionary of distances that can be used as metrics in mathematics to compare numeric values. Based on this review we created a categorization of metrics on account of datatype that is presented in Figure 20.

To demonstrate the impact of data type on choosing

Table 2: Overview of file format based metrics that are modelled in the VPlan.

File Format	Tool support	Metrics
HTML	*Daisy Diff*[22] is a Java library that compares HTML files. It highlights added and removed words and annotates changes to the styling.	*Web Page Appearance Equality*: The appearance of the page is the same. *Web Page Content Equality*: The content of the web page is the same.
XML	*XMLUnit*[23] provides helpers to validate the document against an XML Schema, and compare XML documents against expected outcomes[24].	*Number Of Different XML Nodes Ignore Order*: The files contain the same elements in any order. *Number Of Different XML Nodes Keep Order*: The files contain the same elements in the same order. *XML Header Equality*: The xml headers are the same.
MP3	Feature extraction tools identify audio fingerprints. *RPextract Music Feature Extractor*[25] can be used for this purpose. *Mp3agic*[26] can be used to extract MP3 file metadata, but equality of metadata is not a sufficient condition for files to be the same.	*Audio Fingerprint Equality*: Audio fingerprints match. *Audio Length Equality*: Files have identical length. *Audio Bitrate Equality*: Files have identical bitrate.
TEX	*Latexdiff*[27] is a Perl script for visual mark up and revision of significant differences between two LATEX files. Changes not directly affecting visible text, for example in formatting commands, are still marked.	*Number Of Visible Differences*: Files have no visible differences (content and outlook are identical). *String Type Metrics*: Files are identical (commands and contents are identical)
PDF	*DiffPdf*[28] produces a PDF with marked changes. PDFs can be converted to PNGs using *ImageMagick* and their visual equality can be compared. The text content of PDFs can be extracted using *Apache Tika*[29] and compared using *Diff-Patch-Match* [30].	*Number Of Pages With Different Appearance*: Number of pages that look different. *Page Number Equality*: The documents have the same number of pages. *Text Content Equality*: The documents have the same text content.
ZIP	Zip can be compressed using different compression methods. The header contains last modification date and time. Therefore we compare contents of the file by comparing its bit streams using Java.	*Number Of Different Files*: The number of different files within the archive.
PNG	*ImageMagick*[31] is a free and open-source software suite for displaying, converting, comparing, and editing raster image and vector image files. It can read and write over 200 image file formats.	*Image Fingerprint Equality*: Image fingerprints match. *Image Resolution Equality*: Image resolutions are the same. *Absolute Error Count*: Absolute Error count of the number of different pixels .

the right metric we use the weather workflow. One of the inputs of the *Make Decision* step is an integer value expressing the temperature. This input was detected as a *Plain Text* format, but it is also an integer data type. If the value of temperature in the original execution was 0 and in the re-execution was 273, then depending on a metric used we would obtain different information:

- *Hamming Distance* is three. This means that the compared strings differ in three positions. We would use this metric by default, if we did not know the data type of the value.
- *Euclidean Distance* is 273. This means that the temperature is by 273 degrees different (the distance between integers is 273).

Both of these metrics detect discrepancy and both of them can be used to validate identity of workflow re-execution, however, the second one is more informative. This is because the correct data type was used for metric computation. In the given example, the distance 273 can indicate that the temperature unit was changed from Celsius to Kelvin and in fact the temperature is the same.

Hash based metrics Workflows use external software libraries to compute data. The results are often formatted in a tool specific format that is identified by DROID as a *Plain Text* format. Based on individual format analysis we can provide a set of metrics that, for example, takes into account only the data section of the file. Thus, we can perform a more detailed validation. However, identification of new metrics for a specific format can be a time consuming processes, despite the fact that it is performed once for each format and the results are added to the VPlan model. For formats that were not analysed, there is an alternative approach that computes file hashes using algorithms like MD5 or SHA.

By comparing file hashes and testing their equality, we identify whether the files are identical. The advantage of this approach is that the hash computation is supported by processor instruction sets[32] and therefore is very quick. The downside is that we can only test whether the files are identical. This may be an issue for formats like XML that can have the same elements ordered differently. For such files the hashes are different, while in fact the files contain identical data. Moreover, comparison of hashes can also fail for other non-text based formats like for example JPEG. *The same JPEG image* after reading and writing generates different image data and thus a different hash value. This is due to the lossy compression scheme of JPEG image format. For this reason, we apply *Hash Based Metrics* only to file formats for which no *Format Based Metrics* are defined in the VPlan, or to *Plain Text* files for which no data type was identified.

Image and Audio metrics Authors of [37] provide a classification of metrics that can be used in audio and image processing. The image and audio metrics are independent of the file format. Therefore they are always a subset of metrics that can be applied for validation of audio or image file formats. For example the *Absolute Error Count* metric that defines number of differing pixels per channel can be applied to both PNG and JPEG images. For this reason the *Image Metrics* are a sub class of the *PNG Format Metric* class (see Figure 21). If a new image format is added to the dictionary, then the *Image Metrics* can become a sub class of this new format. Thus we evade multiple repetitive definitions of the same concepts.

For the same reason we use sub classing of metrics in other cases. For example, the *String Type Metrics* are a subclass of the *Data Type Based Metrics*, and the *TEX Format Metric*. This means that the metrics for string comparison can be used when a string data type was detected or when we consider the contents of Latex files to be strings.

Performance metrics We used the ISO25010 standard [38] to analyse possible metrics for non-functional requirements. We focused on the product quality model that categorizes product quality properties into eight characteristics: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability.

The performance efficiency category is the only one that fits to validation of non-functional requirements of workflow re-executions. The other categories could be applied to the assessment of workflows in general, for example, to evaluate their learnability. The performance efficiency is split into three sub-characteristics:

1. Time behaviour - *degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.*
2. Resource utilization - *degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements.*
3. Capacity - *degree to which the maximum limits of a product or system parameter meet requirements Parameters can include the number of items that can be stored, the number of concurrent users, the*

[32] https://software.intel.com/en-us/articles/intel-sha-extensions

communication bandwidth, throughput of transactions, and size of database.

For time behaviour we defined *Execution Duration Ratio* metric that is a ratio between two execution times of a component for which the metrics is measured, for example, a workflow step or a complete workflow.

We defined metrics for resource utilization using examples described in [39]: *Processor Usage Ratio*, *Memory Usage Ratio*, and *Network Usage Ratio*. Each of them specifies a usage ratio for a given resource between two executions.

We did not define metrics for capacity, because the aim of the framework is to identify possible discrepancies between executions of a workflow, rather than to test the capacity of the software and hardware used to implement the workflow.

8.3. VPlan instance generation and model integration

To automate the validation process we implemented the VPlanAssistant tool that automatically generates the VPlan instance for workflows for which identity must be confirmed. In this section we describe this process and explain in what way the VPlan integrates with the workflow model.

We use the workflow context model that contains description of workflow steps and outputs including information on file formats for each output. Furthermore, we use the captured data of the original execution to identify data types for outputs that have *Plain Text* format.

Figure 22 depicts an excerpt of the VPlan for the weather workflow. In comparison to Figure 18, Figure 22 presents additionally information on a file format detected for a workflow step output.

We repeat the following procedure for each workflow step:

1. List outputs of the workflow step.
2. For each output of the workflow step:
   (a) Generate one *Functional Requirement* and link it to the step.
   (b) Read the file format from the workflow model.
   (c) Create *Metric* based on the file format and link it to the *Requirement*.
   (d) Create *Metric Target Value* enforcing values identity.
   (e) Link the *Metric* to the workflow model.

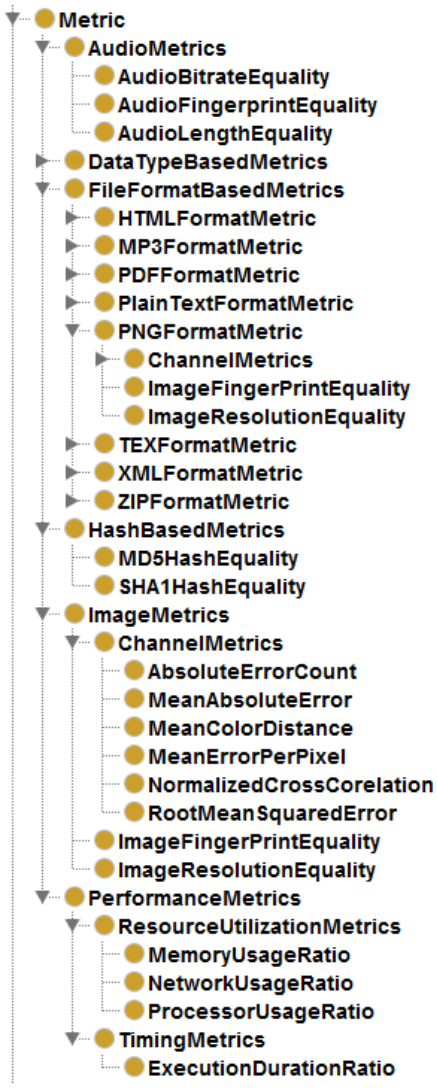


Figure 21: Detailed overview of metric categories in the VPlan.

3. Generate one *Functional Requirement* that groups together other *Functional Requirements* generated for a given step.
4. Generate one *Non Functional Requirement* and link it to the workflow model.
   (a) Create *Performance Metric* and link it to the *Non Functional Requirement*.
   (b) Create *Metric Target Value* and link it to the *Performance Metric*.

For each workflow step we listed its outputs using SPARQL queries. In the analysed example we focus on the workflow step *Visualise Temperature* that is depicted in the centre of Figure 22.

Figure 22: VPlan. Colour coding: yellow - workflow model core, orange - file format specification, light blue - validation requirements.

This step has one output for which we defined one *Functional Requirement*. For that purpose we used a template that expresses the *Functional Requirements* in the following way: ”The output <output name>of the workflow step <step name>must be identical”. We substituted the *<output name>* and *<step name>* variables with *plot* and *Visualise Temperature* names respectively. We use the SPARQL queries again to read from the workflow model information on the file format of the step output. Once we know the format, we use the controlled dictionary of metrics (see Figure 21) defined in the VPlan model to choose the right metrics.

We select and instantiate three metrics for the *plot* output of the *Visualise Temperature* step. These metrics are: *Image Resolution Equality*, *Image Fingerprint Equality* and *Absolute Error Count*. For brevity only the last one is depicted in Figure 22.

For each of these metrics we generate their *Metric Target Values* and set the *value* and *tolerance* to zero, so that only identical executions fulfil the requirement. We also add missing connections to the model that linked the *Metrics* to their *Metric Target Values*, as well as the *Metrics* to the corresponding workflow step outputs for which they are defined.

We generate an additional *Functional Requirement* that groups the *Functional Requirements* of the *Visualise Temperature* step. This requirement follows the following pattern: ”The workflow step <step name>must have identical outputs”. We group the requirements under such high level requirements for a better overview of results.

We generate the *Non Functional Requirement* using the template: ”The workflow step <step name>shall have similar execution duration” and substituted the *<step name>* variable with the name of the step, that is, *Visualise Temperature*. We create the *Execution Duration Ratio* metric that is a subclass of *Timing Metrics* defined in the VPlan. We set its *Metric Target Value* by default to one and allow for 30% deviations. We chose this value arbitrarily. We have to allow for some tolerance when quantifying non-functional requirements, because even when running the same workflow on the same machine it is almost impossible to have exactly the same execution duration.

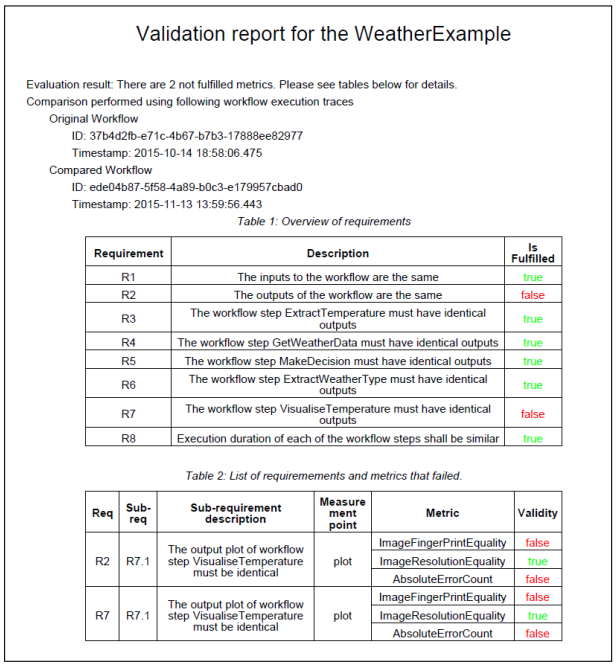
We repeat the above described procedure for all workflow steps and thus generate the VPlan instance.

8.4. Validation automation

We validate workflow re-execution by checking whether all metrics for the requirements are fulfilled. We compare the data and on that basis calculate the metrics’ values using the captured data of the re-executed and the original workflow.

We automated this process by implementing the VPlanComparator. The VPlanComparator uses the context model to source the following information from the context model:

- It reads the requirements and metrics to select the right tools for metric computation.
- It reads the measurement points to identify which data to use for metric computation.
- It reads the actual data to compute the metrics.



Validation report for the WeatherExample

Evaluation result: There are 2 not fulfilled metrics. Please see tables below for details.
Comparison performed using following workflow execution traces
Original Workflow
ID: 37b4d2fb-e71c-4b67-b7b3-17888ee82977
Timestamp: 2015-10-14 18:58:06.475
Compared Workflow
ID: ede04b87-5f58-4a89-b0c3-e179957cbad0
Timestamp: 2015-11-13 13:59:56.443

Table 1: Overview of requirements

Requirement	Description	Is Fulfilled
R1	The inputs to the workflow are the same	true
R2	The outputs of the workflow are the same	false
R3	The workflow step ExtractTemperature must have identical outputs	true
R4	The workflow step GetWeatherData must have identical outputs	true
R5	The workflow step MakeDecision must have identical outputs	true
R6	The workflow step ExtractWeatherType must have identical outputs	true
R7	The workflow step VisualiseTemperature must have identical outputs	false
R8	Execution duration of each of the workflow steps shall be similar	true

Table 2: List of requiremements and metrics that failed.

Req	Sub-req	Sub-requirement description	Measurement point	Metric	Validity
R2	R7.1	The output plot of workflow step VisualiseTemperature must be identical	plot	ImageFingerPrintEquality	false
				ImageResolutionEquality	true
				AbsoluteErrorCount	false
R7	R7.1	The output plot of workflow step VisualiseTemperature must be identical	plot	ImageFingerPrintEquality	false
				ImageResolutionEquality	true
				AbsoluteErrorCount	false

Figure 23: Excerpt of a validation report for the weather workflow.

The VPlanComparator creates reports summarizing the validation results. The overall result of validation is presented and violated requirements and metrics are provided, if detected. Furthermore, detailed descriptions of metrics, their assignment to requirements, as well data formats identified are listed.

For the weather workflow we generated eight functional requirements and six non-functional requirements that we grouped into eight high level requirements. Each high level requirement consists of requirements describing one workflow step and is expressed *"The workflow step <STEP_NAME> must have identical outputs"*. Furthermore, we automatically identified requirements that validate correctness of workflow inputs and outputs and also grouped non functional requirements related to execution duration of each step under one requirement. We performed such a grouping for a better overview of validation results.

Figure 23 presents an excerpt of a validation report for the weather workflow, we can see that:

- The workflow re-execution is not valid, because two metrics failed.
- Ids and timestamps of provenance traces for which the report was produced are provided.
- Table 1 depicts the aggregated requirements and information which of them were fulfilled. Requirements *R2* and *R7* failed, that is, the workflow output is incorrect and the deviations were detected in the *Visualise Temperature* step.
- Table 2 provides details on how the requirements were evaluated. The requirements *R2* and *R7* have the same sub-requirement and therefore failed for the same reason. The requirement *R7.1* was measured for the *plot* output of the *Visualise Temperature* step and three metrics were used to validate it. Only the *Image Resolution Equality* metric did not fail which implies that the workflow produced in fact a valid PNG image, but its content differed. This is confirmed by metrics *Image Fingerprint Equality* and *Absolute Error Count* failing.

9. Evaluation

We evaluate the proposed ontologies by applying it to use cases. We first describe how we selected the use cases, then we describe them shortly and finally discuss results from the evaluation.

9.1. Use Case Selection

Taverna workflows constitute the majority of workflows published on myExperiment [10] - a social web platform for exchange of scientific workflows. The evaluation of all workflows published on myExperiment is not feasible, because the proposed framework requires collection of data from the original environment and this can only be made by the workflows owners. Sufficient data is not available in the portal. Without it we are not able to detect why the workflow re-execution breaks (verification is not possible), or whether the run is repeatable (validation is not possible). We therefore selected workflows for which we have access to the original environment or could establish contact with the original workflow owners.

According to the study presented in [10], which analysed about 1500 workflows, almost 30% of all Taverna workflows published on myExperiment use WSDL web services to perform tasks and 15% have local tool invocations. Furthermore, almost 50% of all workflows use Beanshells that allow users to write their own code in a lightweight Java-like scripting language. Beanshells can import external Java libraries and therefore add further dependencies to the workflow. The local tool invocations may also be made by the Benshells and may require particular software tools to be installed in the environment.

For these reasons we selected five Taverna workflows that resemble these characteristics. Due to the similarities among workflow engines (support for scripting languages, provenance capturing, etc.), we chose one workflow system. Hence, the discussion is focused on Taverna workflows, but the challenges and ways of addressing them remain valid for other workflow systems as well, differing only in the actual technical implementation. We also applied the VFramework to experiments in other technical settings. These workflows did not require workflow engines because they were modelled as bash scripts or python scripts, still working successfully. An example of data captured for Aurora Image Analysis implemented in python can be found here[33]. However, these examples are beyond the scope of this paper.

9.2. Use Case Description

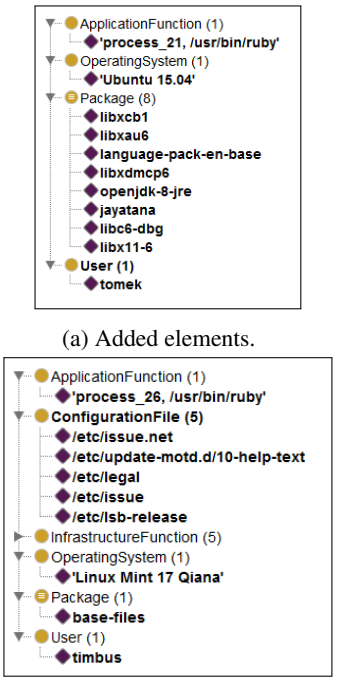
The first workflow is from the domain of music information retrieval and presents *music classification* workflow [13]. It deals with categorisation of pieces of music, for which the category (genre) is unknown, into a set of predefined categories. Workflow uses local libraries for feature extraction and web services for acquisition of music data. Data is available online[34]

The second workflow deals with *sensor data analysis* in civil engineering. It is used in dam safety analysis to estimate the physical quantities that characterize structural behaviour, for example, relative and absolute displacements, strains and stresses in concrete [40]. Workflow uses external web services and *R* for statistical computing.

The other three workflows (*annotate*, *rshell*, *ruby*) are used in the biomedical research for investigating aspects of Huntington's disease [11]. These workflows use the *R* services for performing data analysis to identify genes that overlap with a particular chromatitn state, and a series of web services that use literature information to annotate the lists of genes with most representative biological processes. They also execute *ruby* scripts to create nanopublications. Data is available online[35]

The selected workflows used six different types of workflow steps in their implementations: tool service, Beanshell, Rshell service, WSDL web service, REST web service, and XPath service. The tool services, Beanshell and Rshell scripts required additional dependencies to be present either in the local environment in which the workflows executed, or a specific configuration of services which were accessed to complete steps of the workflows.

[33] https://zenodo.org/record/57742
[34] https://doi.org/10.5281/zenodo.159934
[35] https://doi.org/10.5281/zenodo.47252



(a) Added elements.

(b) Deleted elements.

Figure 24: Comparison of context models of the original and the redeployed *ruby workflow*.

9.3. Results and Discussion

We applied the VFramework to all five workflows in the same way as to the *weather* workflow described in Sections 3 - 8. The evaluation enabled us to answer the following five questions.

Q1 How can we verify and validate workflows executed in different environments?

Q2 What is the impact of software dependencies on verification automation?

Q3 How can we identify workflow dependencies?

Q4 Which data must be published with the workflow to enable verification and validation?

Q5 How can we validate workflows with a set of automatically defined metrics?

In the reminder of the section we discuss answers to these questions.

Q1 How can we verify and validate workflows executed in different environments? In the experiments we re-executed the workflows in environments differing in the version of the operating system (Linux Ubuntu 12 and 15), its distribution (Linux Ubuntu and Mint), and architecture (Linux and Windows). When environments differed in version and distribution, we were able to use all parts of the context model in our analysis. For workflows executed in Windows and Linux the comparison was limited due to differences in the system's architecture – we had to look for equivalent software, and assess its compatibility manually. However, we could still use the *workflow instance data* and *validation requirements*, because the workflow engines provided compatible data.

Figure 24 shows the results of context model comparison for the *ruby* workflow that was run in two different Linux environments. Using the context model we identify in what they differ: different operating system is used, eight new Debian packages are used, one Debian package is not used, different users ran the workflow. The context models created for other workflows enabled similar analysis. We thus evaluated in what way the context model can be used to verify and validate not only re-executions in an exactly identical environment, but also in a similar or considerably different environment.

Q2 What is the impact of software dependencies on verification automation? The workflows analysed differ in a number of additional dependencies required to execute them. Workflows that have fewer platform specific (local) dependencies, like the *music workflow* or the *annotate workflow*, can be almost automatically verified and validated regardless of the execution platform. This is because they do not depend on specific tools that must be present in the system and require only the workflow engine and external web services for their execution. The other three use cases (the *sensor data analysis workflow*, the *rshell workflow*, and the *ruby workflow*) had more system specific dependencies and therefore more manual actions were required to re-execute, verify, and validate them. Table 3 presents tools that we used in each step of the VFramework and average time needed for completion of each step. For all cases the analysis of context models proved to be a universal method for detecting missing dependencies.

Q3 How can we identify workflow dependencies? Two workflows used the R language to execute scripts. The *sensor data analysis workflow* used the shell calls to call local instance of R, while the *rshell workflow* used an R server that was accessed through a service. For both workflows we were able to identify R dependencies correctly. For the *sensor data analysis workflow* we monitored workflow execution to identify additional libraries required by R. On that basis we configured the new environment and verified its configuration. For the *rshell workflow*, we used the static analysis of the workflow model using SPARQL queries. Using this information we configured the Rshell server running in parallel to the workflow execution.

Table 3: Tools and time needed for completion of framework steps for the music classification use case when porting workflow between Linux-based machines.

Framework step	**Tool support**	**Effort**
Run static analysis	Taverna Converter	5 min
Run dynamic analysis	PMF, Taverna Converter, Ontology Diff, Dependency Reporter	60 min
Define validation metrics	VPlanAssistant	5 min
Verify environment	PMF Ontology Diff	60 min
Validate workflow	VPlanComparator	10 min

Q4 Which data must be published with the workflow to enable verification and validation? In the evaluation we also investigated which data must be published together with the workflow to enable verification and validation. For this purpose we investigated two options: first, in which only the context model gets published, and second, in which the identified files and packages get additionally automatically sourced and published. Both of these approaches were successful. The context model is a sufficient source of information for workflow verification. The additional data eases the porting process and enables automatic configuration of a new machine (manually or using PMF and Vagrant). For workflows with few or especially no dependencies, such automation brings little benefits, but for workflows with many dependencies this can be crucial for their repeatability, especially when they rely on specific Debian packages, which are not available in public repositories. Table 4 presents statistics describing the size of the context model by specifying the number of individuals, number of classes and size of files. The first four rows provide detailed information on specific parts of the context model, which total size is depicted in row

Table 4: Number of individuals, distinct classes and file size for data collected in the music classification use case.

	Individuals	**Distinct Classes**	**File size (kB)**
Static CM	90	14	44
Dynamic CM	1056	10	1272
PREMIS	52	4	50
VPlan	65	12	69
CM total	1263	36	1435
Dynamic CM - without noise	78	5	36
Provenance ontology	237	6	254
Provenance data	N/A	N/A	53 MB
PMF data	N/A	N/A	352 MB

CM total.

Q5 How can we validate workflows with a set of automatically defined metrics? The strategy applied for generation of validation metrics that generates requirements for each workflow step and checks each value using a corresponding format comparator proved to work correctly, but has its limitations. The strategy was capable of detecting alterations on different stages of workflow processing and based on these we were able to identify steps in which workflow re-execution was altered. The application of format specific metrics enabled us to correctly compare the data collected for validation. For example, for the *ruby* workflow we compared two archives which contents were identical, while the hashes, which would be computed in a generic case, were different. This was because of a timestamp included in the header of the archive that must be ignored for proper validation.

However, in the *sensor data analysis* use case, despite the usage of the right format comparator, the validation failed because the timestamps generated for each workflow execution were embedded in the data itself, for example the first page of the generated PDF report contained its generation date. Although the usage of a timestamp to annotate data is a good way of identifying data sets, we recommend including it in file properties that can be removed during comparison of actual data. Otherwise, no generic automatic solution can be used, because there is no universal set of rules that allows separating data from the embedded metadata for a generic data type, only custom comparators can be used in such cases. A similar problem was observed by the *reproducible-builds.org*[36] project that analysed how to build sources on different machines to obtain identical Debian packages. The proposed solution is to use an environment variable which enables setting default timestamp value. Another suggested solution is to completely remove timestamps.

Last but not least, for the *music workflow*, as well as during the re-execution of the *sensor data analysis workflow* and the *ruby workflow* we noticed that for some re-executions our criteria stating that all requirements must be identical to recognize the re-execution as repeatable was too strict. In these cases, use case experts could argue that violation of some of the metrics does not have real impact on the repeatability. For this reason, we suggest that the workflow owners generating the requirements in the original environment analyse which of these requirements and metrics are crucial for workflow replicability. Adjusting metric target values, adding comments, or removing unnecessary requirements will result in relaxing the conditions.

10. Recommendations

A special attention must be paid to workflows with platform specific dependencies, because their analysis requires more effort during verification as opposed to workflows with no environment specific dependencies. We are aware that removing the dependencies is not possible, due to the distributed nature of science, as well as the fact that well-established practices and tools already exist. However, workflow owners wanting to improve repeatability of their workflow executions should consider the guidelines listed below.

- **Analyse dependencies and evade shell calls**
  Some of the tasks performed by shell calls can be completed using scripts that explicitly defined in the workflow definition file (e.g. Beanshells in Taverna). Such scripts have higher portability.

- **Publish experiment setup and context**
  Provide a list of tools which need to be present to make the workflow runnable. If you use Linux-based systems you can automatically collect them using PMF and store them in the context model. If your workflow uses additional tools that must be installed in the system, or services that are configured in a specific way describe how to configure the tools and provide specific settings you have used.

- **Publish validation data**
  Provide evidence about the original execution, to

[36] https://reproducible-builds.org/docs/timestamps/

which the re-execution can be compared to. Publish not only the provenance traces of the execution, but also other data files that were used and created by the workflow.

- **Specify validation requirements**
  Describe which workflow requirements must be met to validate the workflow re-execution. Specify which values must be identical and where they should be captured.

- **Test the replicability on your own**
  If your workflow has many complex dependencies, check whether the accompanying data you provide with the workflow is sufficient for its re-execution by doing one on a clean machine.

11. Conclusion

We presented in this paper ontologies that document workflow and its execution environment. They enable checking whether the re-executed workflow produces the same result in the same way as the original workflow did.

We based our solution on the context model that contains a comprehensive description of a workflow and integrates information from different sources describing among others the workflow model, as well as its dependencies detected during dynamic analysis of its execution. It contains also information on external services that were accessed. By comparing context models of workflow executions we verify whether the workflow re-execution was obtained in a compliant way. We based our solution on tools that substantially automate the process of environment analysis and ontology creation.

Furthermore, we analysed sample workflows, their architecture, available data, and motivations of scientists re-executing the workflows. Thus we devised a solution for generation of validation metrics and their evaluation using captured data of workflow executions, which is compared using dedicated comparators. We also formulated requirements that deal with the correctness of data produced at multiple stages of workflow execution. We described the VPlan that extends the context model with the validation requirements, metrics used to quantify them, as well as the measurement points that precisely link the requirements to the workflow model depicting where the data used for their computation is captured. The VPlan also contains a comprehensive and extensible vocabulary of metrics that are used for breaking down validation requirements. It groups metrics into categories taking into account the data format identified for the captured data, as well as its data type. Furthermore, it groups metrics into generic categories, so that these metrics can be linked to new formats added to the vocabulary.

We evaluated the proposed ontologies on five Taverna workflows from three different domains. The workflows used in the evaluation required multiple local dependencies ranging from additional Java libraries, Ruby scripts, and specific Debian packages to external services for running R scripts and web services for completing workflow steps. We re-executed the workflows in environments differing in the version of the operating system, its distribution, and architecture. Thus we showed that the proposed ontologies can be used to verify and validate not only re-executions in an exactly identical environment, but also in a similar or considerably different environments.

Future work will focus on integration of the the proposed ontologies with the Research Objects that are containers for information facilitating understanding of scientific experiments. We believe that the Research Objects should be extended with all five parts of the context model as discussed in this paper in order to complete them with precise description of the workflow environment and validation requirements. Thus the environment in which the workflow was executed is explicitly documented and similar conditions can be recreated. We are also in touch with the members of the Research Data Alliance (RDA) Interest Group on Active Data Management Plans. Together we are drafting a model for Actionable Data Management Plans that would integrate information from existing systems in a similar way to the context model.

Acknowledgment

This research was co-funded by COMET K1, FFG - Austrian Research Promotion Agency.

[1] T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, 2009.

[2] M. Van der Graaf, L. Waaijers, A Surfboard for Riding the Wave. Towards a four country action programme on research data. A Knowledge Exchange Report (2011).

[3] A. Curry, Rescue of old data offers lesson for particle physicists, Science 331 (6018) (2011) 694–695. doi:10.1126/science.331.6018.694.

[4] C. Collberg, T. Proebsting, Measuring reproducibility in computer systems research, technical report (2014).
URL `http://reproducibility.cs.arizona.edu/tr.pdf`

[5] C. G. Begley, L. M. Ellis, Drug development: Raise standards for preclinical cancer research, Nature 483 (7391) (2012) 531–533. doi:10.1038/483531a.

[6] E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, M. Marcelis, The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements, PLoS ONE 7 (6). doi:10.1371/journal.pone.0038234.

[7] C. Silva, J. Freire, S. Callahan, Provenance for visualizations: Reproducibility and beyond, IEEE Computing in Science Engineering 9 (5) (2007) 82–89. doi:10.1109/MCSE.2007.106.

[8] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice and Experience 18 (10) (2006) 1039–1065.

[9] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, C. Goble, Taverna, reloaded, in: SSDBM 2010, Heidelberg, Germany, 2010.

[10] R. Mayer, A. Rauber, A quantitative study on the reexecutability of publicly shared scientific workflows, in: Proceedings of the 11th IEEE International Conference on e-Science, Munich, Germany, 2015.

[11] K. Belhajjame, J. Zhao, D. Garijo, M. Gamble, K. Hettne, R. Palma, E. Mina, O. Corcho, J. M. Gmez-Prez, S. Bechhofer, G. Klyne, C. Goble, Using a suite of ontologies for preserving workflow-centric research objects, Web Semantics: Science, Services and Agents on the World Wide Web 32 (0).

[12] T. Miksa, S. Pröll, R. Mayer, S. Strodl, R. Vieira, J. Barateiro, A. Rauber, Framework for verification of preserved and redeployed processes, in: Proceedings of the 10th International Conference on Preservation of Digital Objects, Lisbon, 2013.

[13] R. Mayer, G. Antunes, A. Caetano, M. Bakhshandeh, A. Rauber, J. Borbinha, Using ontologies to capture the semantics of a (business) process for digital preservation, International Journal of Digital Libraries (IJDL) 15 (2015) 129–152. doi:10.1007/s00799-015-0141-7.

[14] T. Miksa, R. Vieira, J. Barateiro, A. Rauber, VPlan – ontology for collection of process verification data, in: Proceedings of the 11th International Conference on Digital Preservation, Melbourne, Australia, 2014.

[15] K. Belhajjame, O. Corcho, D. Garijo, J. Zhao, D. Newman, R. Palma, S. Bechhofer, E. Garca, J. M. Gmez-prez, G. Klyne, J. E. Ruiz, S. Soil, D. D. Roure, C. A. Goble, Workflowcentric research objects: First class citizens in scholarly discourse, in: Proceedings of Workshop on the Semantic Publishing, (SePublica 2012) 9 th Extended Semantic Web Conference, Hersonissos, Crete, Greece, 2012, pp. 1–12.

[16] R. Mayer, T. Miksa, A. Rauber, Ontologies for describing the context of scientific experiment processes, in: Proceedings of the 10th IEEE International Conference on e-Science, Guarujá, SP, Brazil, 2014.

[17] L. Moreau, Provenance-based reproducibility in the semantic web, Web Semantics: Science, Services and Agents on the World Wide Web 9 (2).

[18] L. Moreau, P. T. Groth, J. Cheney, T. Lebo, S. Miles, The rationale of prov., Web Semantics: Science, Services and Agents on the World Wide Web 35 (2015) 235–257.

[19] P. Missier, S. S. Sahoo, J. Zhao, C. A. Goble, A. P. Sheth, Janus: From Workflows to Semantic Provenance and Linked Open Data, in: Proceedings of the International Provenance and Annotation Workshop (IPAW2010), Troy, New York, USA, 2010, pp. 129–141.

[20] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. V. den Bussche, The open provenance model core specification (v1.1), Future Gener. Comput. Syst. 27 (6) (2011) 743–756. doi:10.1016/j.future.2010.07.005.

[21] E. Pignotti, P. Edwards, A. Preece, N. Gotts, G. Polhill, Enhancing workflow with a semantic description of scientific intent 9 (2) (2011) 222–244.

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

[23] T. O. Group, ArchiMate 2.0 Specification, Van Haren Publishing, 2012.

[24] R. Treinen, S. Zacchiroli, Description of the CUDF Format, Tech. rep., http://arxiv.org/abs/0811.3621 (2008).

[25] J. Freire, C. T. Silva, Making computations and publications reproducible with vistrails, IEEE Computing in Science Engineering 14 (4) (2012) 18–25.

[26] P. J. Guo, CDE: Run any Linux application on-demand without installation, in: Proceedings of the 25th International Conference on Large Installation System Administration, Berkeley, CA, USA, 2011, pp. 2–2.

[27] J. Binder, S. Strodl, A. Rauber, Process migration framework – virtualising and documenting business processes, in: Proceedings of the 18th IEEE International EDOC Conference Workshops and Demonstrations, Ulm, Germany, 2014, pp. 398–401.

[28] I. Santana-Perez, R. F. da Silva, M. Rynge, E. Deelman, M. S. Perez-Hernandez, O. Corcho, Reproducibility of execution environments in computational science using semantics and clouds, Future Generation Computer Systems accepted, funding Acknowledgements: NSF ACI SI2-SSI 1148515, NSF FutureGrid 0910812. doi:10.1016/j.future.2015.12.017.
URL `http://dx.doi.org/10.1016/j.future.2015.12.017`

[29] T. Miksa, R. Mayer, M. Unterberger, A. Rauber, Resilient web services for timeless business processes, in: Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, Hanoi, Vietnam, 2014, pp. 243–252. doi:10.1145/2684200.2684281.

[30] TIMBUS Consortium, Deliverable 6.10 Refinements to Populating and Accessing Context Model (2nd iteration), Tech. rep. (09 2014).

[31] A. Brown, Automatic format identification using PRONOM and DROID, Digital Preservation Technical Paper 1.

[32] PREMIS Editorial Committee, Premis data dictionary for preservation metadata, Tech. rep. (March 2008).

[33] H. G. Gauch, Jr., Scientific method in brief, Cambridge University Press, 2012.

[34] S. Bradner, IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels, Tech. rep. (1997).

[35] IEEE Std 1012 - 2004 IEEE Standard for Software Verification and Validation (2005).

[36] W. W. Cohen, P. Ravikumar, S. E. Fienberg, A Comparison of String Distance Metrics for Name-Matching Tasks., in: Proceedings of IJCAI-03 Workshop on Information Integration, 2003, pp. 73–78.

[37] E. Deza, M. M. Deza, Dictionary of distances, Elsevier, Amsterdam, 2006.

[38] ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, Tech. rep. (2010).

[39] C. Ryan, P. Rossi, Software, performance and resource utilisation metrics for context-aware mobile applications, in: Proceedings of 11th IEEE International Symposium on Software Metrics, 2005, pp. 10 pp.–12. doi:10.1109/METRICS.2005.44.

[40] J. Mata, Interpretation of concrete dam behaviour with artificial neural networks and multiple linear regression models, Engineering Structures 33 (3) (2011) 903–911.

Appendix A. SPARQL queries

```
#1 List workflow steps

select ?step where {
?step a dio:BusinessProcess
}

#2 List inputs of a workflow step
select * where {
?step a dio:BusinessProcess.
?step dio:hasAccessTypeWrite ?input.
?step rdfs:label ?stepLab
FILTER (regex(str(?stepLab), "<NAME>"))
}

#3 List outputs of a workflow step
select * where {
?step a dio:BusinessProcess.
?step dio:hasAccessTypeRead ?output.
?step rdfs:label ?stepLab
FILTER (regex(str(?stepLab), "<NAME>"))
}

#4 List licenses applying to the workflow

select * where {
?constraint a dio:Constraint
}

#5 List steps requiring
# additional libraries

select ?step ?library where {
?library a dio:Artifact.
?script dio:association ?library.
?script dio:realizes ?step.
}

#6 get address and port
# of the external R service
# for a given step

SELECT * WHERE {
?node a dio:Node.
?node dio:Host ?host.
?node dio:Port ?port.
?node dio:realizes ?step.
?step rdfs:label ?stepLab
FILTER (regex(str(?stepLab),
"<STEP_NAME>"))
}

#7 get contents of R scripts
# executed by external services

SELECT ?script WHERE {
?node a dio:Node.
?node dio:Script ?script.
?node dio:realizes ?step.
?step rdfs:label ?stepLab
FILTER (regex(str(?stepLab),
 "<STEP_NAME>"))
}

#8 check wehter R scripts
# load specific libraries

SELECT ?node ?host ?port ?script
WHERE {
?node a dio:Node.
?node dio:Port ?port.
?node dio:Host ?host.
?node dio:Script ?script.
FILTER (regex(str(?script),
 "library\\(.*\\)"))
}

#9 check if workflow
# uses web services

ASK WHERE {
?bp a dio:BusinessProcess.
?appFun dio:usedBy ?bp.
?appFun rdfs:label ?appFunLab.
FILTER (regex(str(?appFunLab),
"Calling_Web_Service"))
}

#10 check if workflow
# uses REST web services

ASK WHERE {
?bp a dio:BusinessProcess.
?appFun dio:usedBy ?bp.
?appFun rdfs:label ?appFunLab.
FILTER (regex(str(?appFunLab),
"Calling_Web_Service"))
?appSer dio:usedBy ?bp.
?appSer dio:assignment ?appInt.
?appInt a dio:ApplicationInterface.
?appInt rdfs:label ?appIntLab.
FILTER (regex(str(?appIntLab), "HTTP"))
}
```