FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Verification and Validation of Scientific Workflow Re-executions

PhD THESIS

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

within the

Vienna PhD School of Informatics

by

mgr inż. Tomasz Miksa
Registration Number 1128471

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.univ.Prof. Dr. Andreas Rauber

External reviewers:
Paolo Missier. Newcastle University, United Kingdom.
Josef Küng. Johannes Kepler University Linz, Austria.

Vienna, 10th March, 2016

Tomasz Miksa | Andreas Rauber

Declaration of Authorship

mgr inż. Tomasz Miksa
Wiedner Gürtel 42/2/6, 1040 Wien

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 10th March, 2016

Tomasz Miksa

Acknowledgements

This dissertation would not have been possible without the support of my colleagues, friends, and family.

Special thanks to my adviser Andreas Rauber. His input, motivation and challenging questions continuously improved my work and enabled me to become a better researcher.

A lot of work described here would have never been possible without the brilliant collaboration with my colleagues from the SBA Research who worked with me on process preservation. I would like to give special thanks to Rudi who always had time for countless discussions.

Thank you also to all my friends who supported me during this time. They have contributed more to this work than they might realize. Special thanks to Maciek without whom I would not have made the decision to pursue the academic career in Vienna. Last but not least, I would like to thank my parents for everything.

Part of this work was supported by the European Union in the 7th Framework Program through the TIMBUS project, contract ICT-269940, as well as the COMET K1, FFG – Austrian Research Promotion Agency. The discussions with partners within and outside of Timbus were a constant motivation to improve on methods and tools.

Kurzfassung

Der wissenschaftliche Fortschritt wäre nicht ohne besondere Tools, Software und Prozesse möglich, die es Forschern ermöglichen, Daten zu verknüpfen, zu transformieren, zu visualisieren und zu interpretieren. Workflowsysteme erlauben es, die Komplexität der zugrunde liegenden Infrastruktur zu verdecken, und sowohl die Wiederverwendung (*reuse*) als auch die Verifizierbarkeit der wissenschaftlichen Prozesse zu fördern. Neueste Ergebnisse zeigen jedoch, dass viele öffentlich verfügbare Workflows nicht mehr ausführbar sind. Darüber hinaus zeigte sich, dass low-level Abhängigkeiten, wie zum Beispiel das Betriebssystem oder Softwarebibliotheken, eine Auswirkung auf das Endergebnis der Berechnung haben.

Aktuelle Technologien bieten keine Garantie, dass die für die Workflowausführung notwendigen Daten verfügbar sind und der Workflow selbst ausgeführt werden kann. Obwohl aktuelle Workflowsysteme Metadaten zur Ausführung von Workflows anlegen, dokumentieren diese nicht vollständig den ganzen Prozess und die darin verarbeiteten Daten sowie Datenflüsse. Sogenannte Provenancedaten reichen nicht aus, um den Workflow vollständig zu erfassen. Beispielsweise sind oftmals keine Information bezüglich der Laufzeitumgebung vorhanden, in der ein Workflow ausgeführt wurde. Diese kann sich jedoch von System zu System unterscheiden und ist daher ein wichtiger Faktor für die Reproduzierbarkeit. Daher müssen wir nicht nur nachvollziehbar zeigen, welche Daten gesammelt werden müssen, sondern es müssen auch die Verarbeitungsschritte systematisch organisiert werden. Nur so kann ein automatischer Vergleich der Workflowausführungen ohne gleichzeitigen Zugriff auf beiden Systeme gelingen.

Verifizierung und Validierung ist heutzutage ein wichtiger Teil der Softwareentwicklung. Die entsprechenden Praktiken sind auch in der Domäne der wissenschaftlichen Workflows wichtig. Deswegen schlagen wir das VFramework vor, das die Korrektheit einer Workflowausführung verifizieren und validieren kann. Das VFramework prüft, ob die erneute Workflowausführung dasselbe Ergebnis liefert, das auch von der ursprünglichen Workflowausführung produziert wurde. Demzufolge ermöglicht das VFramework die weitere Verifizierung der Korrektheit des wissenschaftlichen Experimentes.

Workflows können die gleiche Infrastruktur mit anderen im Betriebssystem laufenden Software teilen. Workflows können auch besondere Tasks delegieren, damit sie durch die in der Laufzeitumgebung installierten Tools ausgeführt werden können. Solche Tools können auch eine spezifische Konfiguration und die Verfügbarkeit von anderen Tools benötigen, die auch von den spezifischen Softwarebibliotheken oder dedizierter Hardware

abhängig sein können. Alle diese Abhängigkeiten beeinflussen den Ausführungskontext eines Workflows. Dieser Kontext muss erfasst und verifiziert werden, um zu prüfen, ob die Workflowausführung die Ergebnisse auf korrekte Art und Weise produziert hat. Das VFramework nutzt das Kontext Model, das eine umfassende Beschreibung eines Workflows liefert. Es beschreibt nicht nur das Workflow Model, sondern es integriert auch zusätzliche Informationen aus unterschiedlichen Quellen in ein Gesamtbild. Das Kontext Model beschreibt auch die Abhängigkeiten, die während der dynamischen Analyse der Workflowausführung identifiziert wurden. Es beinhaltet auch Information über externe Services auf die während der Ausführung zugegriffen wurden. Wir vergleichen die Kontext Modelle der Workflowausführungen miteinander, um die erneute Workflowausführung entsprechend zu verifizieren.

Workflows können verteilte Systeme (*distributed systems*) nützen um entweder Daten von einem Dritten abzufragen, oder Tasks an eine spezialisierte Infrastruktur zu delegieren. Diese Services können sich im Lauf der Zeit ändern und die Workflowausführung kann sich somit ebenfalls verändern oder gar unterbrochen werden. Für Workflows, die Web Services einsetzen, haben wir das Web Service Monitoring Framework entwickelt, mit dem identifiziert werden kann, ob ein Service deterministisch ist. Er nützt vorab gesammelte Daten um eine Attrappe (*mock-up*) eines zustandslosen Services zu entwickeln, die als ein Ersatz für den ursprünglichen Service genutzt wird. Daher sind die widerholbaren Bedingungen garantieret und die Workflowausführung kann verifiziert werden.

Workflows bearbeiten Daten in einer Reihe von Schritten. Manchmal ist nur das Endergebnis der Berechnung wichtig für die Forscher. In anderen Fällen ist auch das Zwischenprodukt wichtig, insbesondere wenn Workflowkomponenten in anderen Experimenten wiederverwendet werden. In allen Fällen müssen die Forscher imstande sein, die teilweise oder komplett erneute Ausführung zu validieren. Aus diesem Grund haben wir eine Workflowprobe analysiert und die Voraussetzungen definiert, welche die Korrektheit der Daten auf allen Etappen der Workflowausführung prüfen. Wir haben auch den VPlan entwickelt. Der VPlan erwitert das Kontext Model mit Validierungsvoraussetzungen, Metriken für ihre Quantifizierung, sowohl die Messpunkte. Die Messpunkte verknüpfen die Validierungsvoraussetzungen mit dem Workflow Model und stellen dadurch dar, welche Datei für die Quantifizierung verwendet wird. Der VPlan beinhaltet ein umfassendes Vokabular von Metriken, welches die Validierungsvoraussetzungen beschreibt.

Das VFramework wurde ausführlich mit der Taverna Workflow Engine evaluiert. Wir haben fünf wissenschaftliche Workflows aus drei Domänen verwendet: Sensordatenanalyse der Bauingenieurwissenschaften, Musik Klassifizierung in Retrieval-Systemen, und die medizinisch-klinische Forschung. Die ausgewählten Workflows haben viele lokale Abhängigkeiten, von zusätzlichen Bibliotheken, Skripten und spezifischen Packages, bis zu externen Services. Die Evaluierung zieht die Ausführungen in unterschiedlichen Betriebssystemen in Betracht. Wir haben die notwendigen Schritten beschrieben, den benötigten Aufwand abgeschätzt, und dargestellt, wie das VFramework automatisiert werden kann.

Abstract

Modern scientific breakthroughs would not be possible without special tooling, software and processes that allow researchers to link, transform, visualise and interpret the data. Workflow engines were proposed in order to hide the complexity of the underlying infrastructure and foster reuse and verification of scientific processes. However, recent studies report that many publically shared workflows break and are not executable. Furthermore, the impact of low level dependencies, like operating system or software libraries, were reported to have an impact on the final result of the computation.

The existing practices for sharing workflows do not guarantee that data enabling verification and validation of their re-executions is available. Provenance traces do not contain complete data describing workflow executions and there is no information on the environment in which a workflow was executed. Hence, we need to identify how these data can be collected in a repetitive way and organised in a systematic manner to enable automatic comparison of workflow re-executions without having access to both systems at the same time.

Verification and validation is nowadays an important part of the software development lifecycle. Corresponding practices are needed in the scientific workflow domain. For this reason we propose the VFramework that can verify and validate the correctness of a workflow re-execution. The VFramework checks whether the re-executed workflow produces the same result as the original workflow. Thus it creates a starting point for a later verification of scientific experiment correctness.

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. The VFramework uses the context model that contains comprehensive description of the workflow integrating 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.

Workflows can use distributed systems to either source data provided by a third party, or to delegate computational tasks to an infrastructure offering specialized computing capabilities. These services can change and in consequence alter the workflow execution or even break it. For web service dependent workflows we developed the Web Service Monitoring Framework that detects whether a web service is deterministic. It uses the evidence collected to create mock-ups of stateless web services that are used to replace the original service and thus ensures repeatable conditions for verification of workflow re-executions.

Workflows process data in a series of steps. In some cases only the final result of a computation is important for researchers, while in other cases the intermediate data can be important as well, especially when parts of the workflow are reused in other experiments. In all cases, researchers must be able to validate whether the partial or full re-execution was valid. For this reason we analysed sample workflows and formulated requirements that check the correctness of data produced at multiple stages of workflow execution. We created 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.

We extensively evaluated the VFramework 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. We describe necessary steps, estimate the effort to complete each step and demonstrate in what way this can be automated. To complete the evaluation we simulate changes that may happen to a workflow and show how we detect them using the VFramework.

Contents

CHAPTER 1

Introduction

In many natural science disciplines, complex and data driven experiments form the basis of research [HTT09]. Scientific breakthroughs would not be possible without special tooling, software and processes that allow researchers to link, transform, visualise and interpret the data [VdGW11]. Scientific areas established their own methodologies for performing and documenting experiments. As a consequence researchers not only use a wide range of software tools, but also have to implement their own. This often forces them to acquire software engineering skills and thus they are often distracted from their core research area.

The low maturity of tools and the possible lack of scientific scrupulousness [Cur11] led to a low reproducibility and replicability of experiments [CP14], [BE12]. Many problems can be attributed to the fact that the software is not available any more [CP14]. 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 [GHJ$^+$12] 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[SFC07], Kepler [LAB$^+$06] or Taverna [MSRO$^+$10] have become popular in research areas like Astronomy, Bioinformatics or Clinical Research. They enable researchers to graphically represent their experiments in the form of workflows that can be built using pre-defined elements. These elements range from dedicated data parsers to interfaces for calling external web services. Workflow engines also allow including executable code in various programming languages and specifying system commands to be run. For example, Taverna allows Java code in form of Beanshells and uses "local tool invocation" for system commands execution. Workflows can be run on a single machine or use distributed infrastructure to complete particular steps [LYJC11]. 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 other workflows. Platforms like myExperiment[1] facilitate such exchange. Furthermore, in order to enhance the reproducibility, workflow engines are equipped with provenance collection mechanisms that gather detailed information about the workflow execution. The traces contain information specifying who ran the experiment at what point in time, what data was provided as input and what intermediate data was exchanged between the workflow steps.

In spite of such a standardisation, a recent study [MR15a] 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 [BZG$^+$15] 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 on how to identify and store such data.

Verification and validation is nowadays an inextricable part of the software development lifecycle. For instance test driven development assumes that each software component has a corresponding unit test that allows automatically testing the performance of software component at any time when a change to the system is introduced or when it is deployed on a new platform. Furthermore, a common practice is to provide logging and tracing mechanisms.

However, in the scientific workflow domain the practice that corresponds to well-established approaches from software engineering is still in its infancy. Solutions like testing and code reviews are suggested to researchers as means for improving experiments reproducibility, but no concrete solutions are in place [SLP14]. Therefore, in order to narrow this gap we describe an automated framework for verification and validation of workflow re-executions. By verification we mean checking whether the result was produced in a correct way, for example, whether the correct software libraries were used. While by validation of workflows we mean assessing whether the result produced is correct, for example, whether the output of the workflow re-execution is the same as the original one. We consider these terms in the context of workflows re-execution, when the original workflow is run to reproduce or replicate the original result, or when parts of workflow are reused in a new experiment. Our framework does not verify and validate the original scientific workflow in a sense of scientific correctness.

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

1.1 Problems and Research Questions

In this section we describe specific problems addressed in this dissertation and on that basis formulate research questions.

Verification of workflows and their execution context

Scientific workflows may require workflow management systems for their execution. These in turn share common infrastructure with other software running on 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. Thus a long chain of system processes is spawned when a workflow is executed. On top of that, workflows can use distributed systems to either source data provided by third party providers, or to delegate computational tasks to the infrastructure offering specialized computing capabilities.

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.

Validation requirements and their quantification

Scientists use dedicated platforms for sharing experiments to which they upload their workflows with the intention of enabling other researchers to re-run the workflows in order to replicate the original experiment or to re-use their workflows in new experiments built by others [DRGS09]. The workflow definition files are sometimes accompanied by additional resources like input data, software libraries that are used to run the workflow, or the provenance traces that contain data produced at each step of the workflow execution. Thus the researchers re-executing the workflows have in theory a possibility to state whether their re-execution matches the original one.

However, currently there are no means in use that enable automatic comparison of provenance traces produced by the re-executed workflow with its original traces. A simple comparison of file hashes is not possible, because the traces can differ in multiple ways. For example, they can contain execution timestamps, the same data can be organised in a different order, or can contain data formats which require special comparison methods to confirm their identity. Furthermore, the environment in which the workflow is re-executed is very likely different, because the scientists use their own infrastructure and systems in which they run other workflows. The requirements of workflow management systems do not enforce specific configuration of environments in which they run and hence, for example, Taverna can be run on different distributions of Linux, as well as on Windows using different version of Java, and so on [MSRO$^+$10]. This can lead to simple discrepancies in workflow execution like different paths for input values, because the user name was different, but can also be caused by more serious changes in the environment like a different version of a library and hence different implementation of the same algorithm, potentially delivering different results.

For this reason we need a comprehensive method that allows validating workflow re-executions using data captured in the original environment that can be shared with others and later automatically processed to provide reliable information to the researcher re-executing the workflow whether the re-execution produced the correct results. We also need to identify which data sources can be used to automatically capture data that enables comprehensive validation of workflow re-executions.

Systematic approach for scalable verification and validation of scientific workflow re-executions

There are well-established approaches for verification and validation in domains related to scientific workflows, but they cannot be applied directly in the scientific workflow domain. They are either too generic or too specific. For example, in software engineering there is the IEEE 1012 standard [IEE05] that describes procedures for system and software verification and validation, while in digital preservation there is a framework for validation of emulation results [GR12] that compares renderings produced with the same software in two different environments. The common part of all such approaches is the systematic way of evaluating the analysed system and a repeatable way of collecting evidence that supports the decision making process and enables it traceability.

Recent progress in the domain of knowledge representation has increased popularity of ontologies that enable representing knowledge about complex phenomena in a comprehensive way, including systems and processes. Thus the automatic analysis of systems requirements and reasoning about their requirements became possible and was successfully applied in the enterprise modelling domain [Ant15], as well as in the digital preservation domain for preservation of business processes [SMA$^+$13].

For this reason we investigate how the well established principles of verification and validation can be combined into a systematic approach that can be used in the scientific workflow domain. We put special attention to models enabling comprehensive description of workflow execution context. The comparison of their instances should enable verification of workflow executions without the necessity of accessing both environments at the same time. We aim from the beginning for the automation of the solution to increase the acceptance among users.

In this dissertation we address a number of research questions that are based on the above challenges, in particular:

RQ1 How can we verify whether the workflow re-executed in a way that complies with the original execution?

a) How can we identify software, data, and services used by the workflow in its execution?

b) How can we ensure determinism of workflows that use external services to complete tasks?

c) What data must be captured and how?

d) How can we compare the environments and interpret possible differences?

e) How can we verify re-executions in non-identical environments?

RQ2 How can we validate whether the workflow re-execution produced the correct results?

a) What are universally applicable validation requirements?

b) How can we measure these requirements?

c) What data must be captured and how?

d) How can we compare this data?

e) How can we validate re-executions in compliant but non-identical environments?

RQ3 How can we perform systematic and repeatable verification and validation of workflow re-executions?

a) How can we support model-based verification and validation?

b) What evidence must be collected and shipped with the original workflow that enables verification and validation of its re-execution?

c) How can we automate this verification and validation process?

In this dissertation we devise the VFramework [MPM+13] for the verification and validation of scientific workflow re-executions. The original design goals include digital preservation settings in which workflows potentially need to be re-executed at much later points in time in different computational environments, 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 hardware and software.

We instantiated the VFramework using tools that automatically capture [BSR14] a context model [MAC+15] to describe resources used during workflow execution and to identify information about dependencies of the workflow that need to be present to rerun the workflow and verify its correctness. For web service dependent workflows we developed and integrated with the VFramework the Web Service Monitoring Framework that detects whether the web service is deterministic. Based on the evidence collected, it allows for creating mock-ups of stateless web services that are used to replace the original service and thus ensure repeatable conditions for verification of workflow re-executions.

We also devised the VPlan ontology [MVBR14] that extends the context model with description of validation requirements. It links requirements to the workflow model, thus providing explicit information which data must be captured for evaluation of a given requirement. The requirements are quantified using metrics from a controlled vocabulary that is also a part of the VPlan. The actual metrics values which are used for validation are calculated by comparing the captured data using appropriate comparators depending on the data format detected.

We evaluated the VFramework on Taverna workflows. We used five scientific workflows from three domains: sensor data analysis in civil engineering, music classification in information retrieval, and medical clinical research. The selected workflows had 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. The evaluation took into account re-executions in different operating systems. We described necessary steps, estimated the required effort to complete each step and demonstrated in what way this can be automated. To complete the evaluation we simulated changes that may happen to a workflow and showed how we detect them using the VFramework.

Despite the fact that the discussion presented is focused on Taverna, the challenges and ways of addressing them remain valid for other workflow systems, differing only in the actual technical implementation.

1.2 Contributions

Work presented in this dissertation involved a network of collaborators and project partners, with my specific contributions listed in the following.

1. I conducted a thorough review and analysis of existing approaches for verification and validation in the related domains and led the definition and specification of the VFramework [MPM$^+$13].

2. I conducted an analysis of web service monitoring literature and led the specification of the Web Service Monitoring Framework. I also implemented the tools for creating mock-ups of web services [MMR15a] [MMUR14] [MMR15b].

3. I led the analysis of use cases and implementation of the VPlan as an ontology for expressing validation requirements that extends the workflow context model. I was the leading author of a paper that received the best paper award on the peer-reviewed conference iPres 2014 [MVBR14].

4. I performed a study of tools for automated comparison of data based on its format. I also conducted a literature review of distance measures used for measuring similarity of data. Based on this analysis I defined a controlled vocabulary of metrics that is part of the VPlan.

5. I implemented tools that automate the VFramework application, namely: a tool for dependency report generation, a tool for validation requirements generation,

and a tool for comparison of captured data and validation reports generation. I also devised a set of queries used for model analysis [RMMP15].

6. I led the design of evaluation scenarios and the actual evaluation of use cases.
7. Based on the initial evaluation results I proposed an extension of existing Data Management Plans with additional information enabling verification and validation of workflows [MSR14] [MR15b].
8. Based on the evaluation results I formulated guidelines for scientists sharing their workflow that enhance workflows' replicability by enabling verification and validation of their re-executions (under review).

1.3 Organization

This section outlines how this thesis is organised. It lists the different chapters and references to publications in which the main contributions of this thesis have been published in peer reviewed journals or conferences.

- Chapter 2 describes the related work. We clarify terms related to replicability and discuss definition of verification and validation [RMMP15]. We also provide an overview of reproducibility and replicability challenges and discuss which elements belong to the context of a scientific experiment. We describe in what way the context of a workflow execution can be described and which tools can be used for its automated capturing [MMR14]. Furthermore, we present digital preservation concepts that influenced the design of the proposed VFramework [MPM$^+$13], especially the framework for verification of emulation effects, and the process for preservation of business processes [MMS$^+$14]. Last but not least, we discuss ways of detecting changes in web services and provide motivation for the Web Service Monitoring Framework [MMR15a].
- Chapter 3 introduces the VFramework – a framework for verification and validation of scientific workflow re-executions [MPM$^+$13]. We discuss its general structure and explain in which settings the VFramework can be applied. We also present a running example on which we demonstrate the application of the VFramework and creation of the VPlan in Chapters 4, 5, and 6. We provide the motivation for choosing the given workflow and describe in detail each of its elements.
- Chapter 4 describes in detail the *Run static analysis* and the *Run dynamic analysis* steps of the VFramework. These steps capture the original execution of the workflow and document the environment in which it took place. Furthermore, we show how the automated context model analysis enables identification of workflow boundaries and in what way the external services being outside these boundaries can be monitored to create evidence needed for re-executing and validating workflows [MMR15b].

- Chapter 5 describes in detail the *Define validation metrics* step of the VFramework and presents the VPlan ontology that we created for describing validation requirements [MVBR14]. We introduce the VPlan model by specifying classes and relations used to connect them. We present how to model information needed for validation 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. The extensible controlled vocabulary is a part of the VPlan model and it groups metrics into categories taking into account the data format, as well as its type. The metrics were derived based on a literature review and analysis of existing tools supporting comparison of specific file formats. We also describe in what way the VPlan instance is automatically generated to validate identity of workflow re-executions.

- Chapter 6 describes the *Verify environment* and *Validate workflow* steps of the VFramework that are performed when the workflow is re-executed in a new environment. To complete them we collect information about the re-execution from the new environment and compare it to the information collected in the original environment (see Chapters 4 and 5), which was stored in the context model.

- Chapter 7 presents the evaluation of the VFramework on five Taverna workflows from three different domains, namely: music information retrieval, sensor data analysis in civil engineering, and clinical medical research. We are in direct contact with the workflow owners and have access to the original environment in which the workflows execute, hence we can perform complete analysis of their execution context. Furthermore, the selected workflows use typical workflow components that are used by the majority of Taverna workflows published on myExperiment community portal for exchange of scientific workflows. We evaluate to what extent the cross-platform verification and validation using the VFramework is possible by re-executing the use cases in Linux- and Windows- based environments [MPM$^+$13] [MVBR14]. We also consider to what extent the VFramework can be automated, which framework steps must be performed manually and what is the effort required to complete them. Furthermore, using the music classification use case we simulate potential changes that can occur during workflow re-execution and evaluate to what extent the VFramework and the proposed approach for selection of validation metrics detect such changes. We also use mock-ups of web services to ensure repeatable conditions for workflow executions and analyse changes in the execution context caused by them.

- Chapter 8 presents the conclusions of the work conducted over the course of this dissertation. We summarize our achievements, discuss limitations of the proposed approach and give an outlook on future work.

CHAPTER 2

Related work

This chapter presents the related work that influenced and motivated this dissertation, and provides an overview of the concepts and tools used to create the solutions presented. It is organised as follows.

- In Section 2.1 we provide definitions of terms related to reproducibility to establish a common language.
- In Section 2.2 we discuss different definitions of verification and validation and explain in which settings the proposed VFramework is applied.
- In Section 2.3 we explain the concept of workflows and workflow management systems. The similarity of their capabilities and popularity of the Taverna workflow engine is the reason for choosing it as an example on which we demonstrate the concepts described in this dissertation.
- In Section 2.4 we give examples of studies in which replicability fails and thus we motivate our work.
- In Section 2.5 we present digital preservation concepts that influenced the design of the proposed VFramework, which we present in Chapter 4.
- In Section 2.6 we describe how the context of a workflow execution can be described and in what way it affects verification and validation. We specifically focus on the context model that we use for describing a workflow execution context during the VFramework application.
- In Section 2.7 we discuss which tools can be used for automated capturing of the workflow execution context. We pay special attention to the tools that detect software and hardware dependencies.

- In Section 2.8 we discuss ways of detecting changes in web services and provide motivation for the Web Service Monitoring Framework developed by us and described in Section 4.2.3. The changes or unavailability of web services affects the workflow performance and for this reason workflow depending on external services need special treatment.

2.1 Reproducbility and related terms definitions

The terms *re-execute*, *rerun*, *replicate*, *repeat*, *reuse*, and *reproduce* are often used as synonyms. Based on results of a literature review described below, we summarized the differences between them in Table 2.1. We used the following distinction criteria:

- In column *Environment* we indicate whether the same or different environment is used in the experiment.
- In column *Researcher* we specify whether the experiment is performed by the same researcher.
- In column *Complete workflow* we specify whether the complete workflow (including all its steps) is used in the experiment.
- In column *V&V of workflow re-execution* we indicate whether the verification and validation of a workflow re-execution is performed. We mean a situation in which the researcher checks whether the same result was obtained using the same tools. We do not mean checking the scientific experiment correctness, that is, checking whether the whole experiment makes sense from the scientific point of view. For more details see Section 2.2.

In cases when a term does not provide distinction in a given category and both options are possible, we indicate this using *same/different* or *yes/no*. When a given category does not apply, then we use *NA*.

According to [Dru09] *replicability* assumes that the experiment is repeated using exactly the same tools and conditions, while *reproducibility* denotes a situation in which the same result is obtained in a different setup.

Replicability is sometimes contrasted with *repeatability*. In [VK11] authors define *repeatability* as a situation when the experiment is run (repeated) by the original researcher using exactly the same tools. *Replicability* can be understood that someone else repeats the same experiment in the same way.

When a workflow is *re-run* or *re-executed*, it means that a given workflow is run again [JCS$^+$15]. These two terms do not imply who and in which environment runs the workflow. *Re-running* or *re-executing* a workflow is a subset of *replicating* or *repeating* a workflow, because *re-running* or *re-executing* do not include verification and validation.

Table 2.1: Comparison of a meaning scope of terms related to reproducibility.

Term	**Environment**	**Researcher**	**Complete workflow**	**V&V of workflow re-execution**
replicate	same	different	yes	yes
repeat	same	same	yes	yes
reproduce	different	same/different	yes	NA
re-execute	same/different	same/different	yes	no
rerun	same/different	same/different	yes	no
reuse	same/different	same/different	no	(yes)

Reuse means a situation in which the workflow is used in a new experiment. Either new datasets are fed into it, or some of its parts are modified or some of its dependences are changed [DRGS09]. Usually before the workflow is *reused* the non-modified parts should be *replicated* and verified, and validated.

The VFramework can be applied in the same way to cases when the workflows is *replicated* or *repeated.* Furthermore, it can also be applied to verify and validate scenarios in which workflow parts are *reused.*

2.2 Verification and Validation

There are different perspectives on verification and validation. In this section we provide definitions of these terms with respect to project management, and systems and software engineering. We also explain in what way we apply them with respect to workflow re-executions.

The Guide To The Project Management Body Of Knowledge [PMB04] presents terminology and a set of guidelines for project management. It also defines project management related concepts and describes project life cycle. It defines verification and validation as follows:

- *Verification - The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation.*

- *Validation -The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification.*

The process of creating, sharing and replicating scientific workflows can be viewed from the software project management perspective. The needs of the researcher (stakeholder) replicating the experiment state that the re-execution (product) must provide the same results as the original execution. Furthermore the result must be obtained in a similar or identical way (requirement).

From the systems engineering point of view the workflow executing within a workflow engine is just a software executing in a computer system. Authors of [KSea11] provide definitions of verification and validation with respect to systems engineering:

- *Verification is the process of determining whether the software implements the functionality and features correctly and accurately. In other words, verification determines whether we implemented the product right.*
- *Validation, in contrast, is the process of determining whether the software satisfies the users' or customers' needs. In other words, validation determines whether we implemented the right product.*

The definition provided by the IEEE Standard for System and Software Verification and Validation [IEE05] states that verification and validation (V&V) is a continuous process of ensuring quality and satisfaction of user requirements performed in parallel to the software development process.

Similarly, for the scientific workflows V&V can be performed at multiple stages of the experiment lifecycle, ranging from experiment design and implementation to its preservation and finally reuse. To structure the discussion we distinguish the following cases:

- V&V of the implementation of a scientific workflow,
- V&V of the workflow re-execution,
- V&V of the workflow for reuse.

The verification and validation of the implementation of a scientific workflow aims to check whether the way in which the experiment is built reflects the intentions of the researchers. Such a validation is performed by the scientists themselves, when they work on the experiment. Also, they verify themselves whether the software and hardware used in the experiment performs correctly [TP04].

It is beyond the scope of our investigations to check the correctness of the scientific workflows published by researchers. It is the responsibility of scientists reviewing the experiment to perform the validation and identify flaws in the experiment design or implementation. There are frameworks that facilitate this process. In [WMF^{+}05] authors describe a framework for verification of workflow executions using provenance of experiment results and semantic descriptions of services. However, in our setting we must be able to prove whether the replicated workflow execution has the same characteristics and performs in the same way as the original workflow did. Due to this, our role is limited to the verification and validation of the *transition process* [IEE05], which corresponds to a situation in which the original workflow is extracted from its original environment and placed in a new environment. The *transition process* can alternatively be called a porting process or redeployment, especially in the digital preservation domain.

The IEEE 1012 standard [IEE05] defines also the *verification and validation of reuse software* that aims to check the correctness, completeness, accuracy and usability of reused software. This approach can also be applied to scientific workflows and is performed by scientists willing to reuse an already published process in their own experiments. In settings when the process was not designed to be reused, there is a lack of proper process documentation, source code and other artefacts fostering the reuse. Thus, techniques like black box testing or historical data analysis are performed to identify the behaviour and requirements of the process that is considered to be reused.

When applying the VFramework (see Chapter 4), data enabling verification and validation of workflows for reuse is collected. Our solution is based on the automatically created context model that is a source of information about process requirements and dependencies. Hence, the verification and validation of the *reuse process* [IEE05] is enabled by creating a comprehensive description of the workflow and its environment.

2.3 Workflow systems

So-called *in silico* experiments have emerged across many domains due to the increase in computing capabilities of research infrastructures. In silico experiments use models that reflect real world phenomena and on their basis the scientists construct new theories and make new scientific discoveries [HTT09].

For scientists like biologists or chemists, the set-up of complex technical infrastructure is a challenge that requires specific background knowledge, like for example programming skills [HWS+06]. Scientific workflows were designed to solve this problem by providing the scientists with an easy way of specifying the tasks that have to be performed in an experiment, without the necessity of knowing the technical details of its execution. Workflows allow a precise definition of the involved steps and the data flow between components [WFRG09]. Workflow management systems are software that provides an environment in which these workflows are executed, thus becoming an environment in which in silico experiments are designed and performed [GWG+07].

Different scientific workflow management systems exist that allow scientists to combine services and infrastructure for their research. Examples of well-established systems are Taverna [MSRO+10], Kepler [LAB+06] or Vistrails [SFC07]. The authors of [Tal13] describe these workflow systems and also present other workflow environments used in science and engineering on high-performance computers and distributed systems. They

Table 2.2: Features of workflow systems [MPR12].

Engine	Implement.	Script Language Support	Designer Support	Execution Engine	Provenance
Taverna	Java	Beanshell	Standalone	Integrated with designer	Database (Apache Derby)
Kepler	Java	Python	Standalone	Integrated with designer	Database (HSQLDB)
Activti	Java	JavaScript, Python, Ruby, etc.	Eclipse IDE	Web application or Java program	Database (H2 DB)

describe additionally following workflow systems: Pegasus [DShS+05], Triana [TSWR03], Askalon [FPD+05], and GWES [Hoh06].

A comparison of workflow management systems is presented in [MPR12]. The authors compared Taverna, Kepler and Activiti [Rad12]. The first two are typically used in scientific settings, while Activiti is used in business settings, because it allows modelling of workflows using the Business Process Modelling Notation (BPMN). The results of the comparison are summarized in Table 2.2. The compared systems were implemented in Java and each of them supports scripting languages. Taverna allows Beanshells that are based on Java, Kepler allows for Python scripts, while Activiti supports a wide range of different scripting languages. Each of the workflow systems records the workflow provenance in a database.

Due to the similarities among workflow engines, we chose one workflow system on which we applied the VFramework for verification and validation of workflow re-executions. For this reason the discussion presented in this dissertation 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.

In Chapter 3 we demonstrate a running example used in the thesis that is a Taverna workflow and explain the terms that we use to describe workflows.

2.4 Replicability challenges

In [CP14] 613 papers from eight different ACM Computer Science conferences were analysed. The authors were able to build only about 15% of programs described in the conference papers. The code was either not available, or was not complete. Even the original creators of the software, contacted by the authos of [CP14], were not always able to build their own software. Figure 2.1 presents the detailed results of this analysis. Out of 613 analysed papers, there were 515 papers that claimed to develop software. For 231

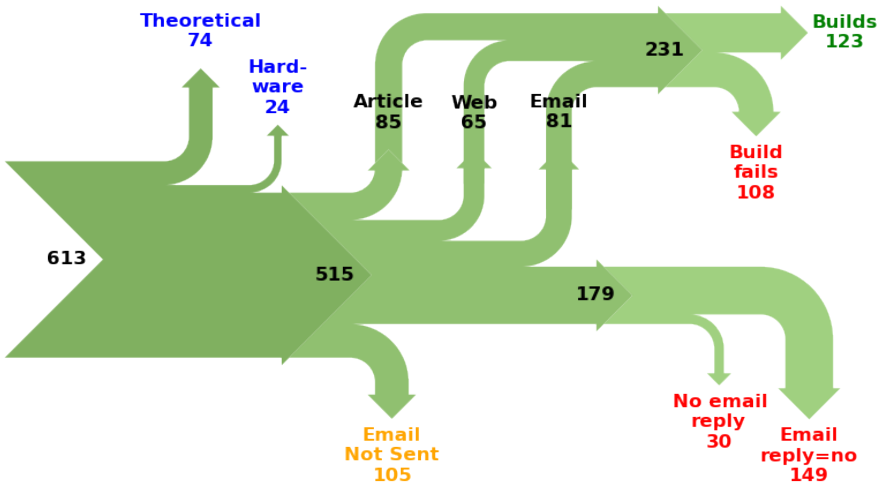


Figure 2.1: Overview of re-executability of software that was described in papers submitted to eight ACM conferences [CP14].

papers there was software available, but only in 123 cases the authors managed to build the software.

A similar study was performed in economics where the authors of [Mcc07] showed that the data and the source code submitted together with journal papers were in most cases not runnable and therefore the results were not replicable. An approach similar to validation presented in [SMS15] executes the programs uploaded to a research portal and states whether the execution finished without producing error messages.

The study presented in [GHJ+12] revealed the impact of a system software version on the final results. Analysis performed using the *FreeSurfer* software package did not break, but lead to different results depending on the operating system and libraries, and making previous analyses performed in different environments not comparable. The authors encourage to "provide not only the version of FreeSurfer that was used but also details on the OS version and workstation" as it affected the results obtained. While the example above uses a customised software, rather than workflows, similar analyses are performed via workflows, for example the analysis of kidney biopsy microscopy images using Taverna in [GDCM13].

The study presented in [YCK+10] describes the impact of Grid computing environment on the results of scientific experiments. The authors found that the experiment results can differ by 2% when the same simulation is run in different clusters within a Grid.

They used a molecular docking simulation program DOCK that is used for virtual screening of chemical compounds to test with in-vitro experiments. They conclude that the detected differences have little effect on the validity of using *virtual screening* before subsequent steps in the drug discovery process, but they also state that it is difficult to predict whether the accumulation of these discrepancies over sequentially repeated *virtual screening* experiments can significantly alter the results if *virtual screening* is used as the primary means for identifying potential drugs. Moreover, such discrepancies may be unacceptable for other applications requiring more stringent thresholds. The authors suggest using virtual machines as a layer of abstraction to mitigate the effects of platform heterogeneity. They examined the differences and variations of DOCK *virtual screening* variables across a Grid environment composed of different clusters, with and without virtualization. The uniform computer environment provided by virtual machines eliminated inconsistent results caused originally by heterogeneous clusters, however, the execution time increased.

In [GDE$^+$07] authors describe results of a workshop on examining challenges of scientific workflows. In one of the recommendations they state:

> "We need workflow representations at different levels of abstraction, so that we can represent workflows at different levels of refinement, from abstract application-level definition down to operational, system-specific description."

We identified that this recommendation is not fully implemented, especially the system specific information is still missing in workflows description. We address this recommendation using the context model (see Section 2.6) to provide a comprehensive description of workflows taking into account different perspectives and requirements. Among the challenges of scientific workflows, the authors also state that reproducibility is one of them. They stress that "reproducibility requires rich provenance information", so that researchers can verify and validate their research.

Reasons for Taverna workflows to break were described in [ZGB$^+$]. The authors show examples in which a workflow that was re-executable was in fact delivering altered results due to the changes in the third party resource. The authors claim that publishing requirements for software libraries used by the workflow and sample output data could decrease the decay of workflows.

In [MR15a] authors quantified how many workflows published in the myExperiment platform can be re-executed. They focused on Taverna 2 workflows which constitute 55% of workflows available in this portal. They analysed 1443 workflows and identified that only 731 workflows were potentially executable, because they did not miss input data values and were compatible with the test environment. They managed to execute 341 workflows only. The remaining 364 failed because:

- The external web service was not reachable or authentication details were missing (38).

- The local software tools or other resources were unavailable (28).
- “The input data was not pure”, that is, it contained more values listed which can be potentially guessed by a human (298).

2.5 Digital preservation

The mission of digital preservation is "to overcome the obsolescence threats that digital material is facing on the bit stream, the logical and the semantic level, and to provide continued, authentic long-term storage and access to digital objects in a usable form for a specific user community" [Bec10].

Several strategies are possible for digital preservation and application of any of them strongly depends on the set of information which has to be preserved. Two of them were recognized as most promising and are used most frequently, these are migration and emulation [Rot99] [Gra00].

Migration converts the original object and makes it operable in a different technical environment than originally intended [Mar96]. The SCAPE project investigated several migration tools, ranging from image converters to database migration suites [The11].

Emulation is a widely used concept in the information technology. It is used in software engineering when the development and the target systems differ, for example software developed for mobile phones is tested in an emulated environment. Emulation enables also running legacy software in production environments [Gra00] [JvdHV07] [Rot99].

2.5.1 Evaluation of emulation effects

When any of these methods is applied, effects have to be validated. A framework which was created for multi-level comparison of emulation effects was presented in [GR12]. The framework allows assessment of the degree to which the system emulation preserves original characteristics of digital objects. The framework enables preservation planners (for preservation planning see [BKG$^+$09] [BR11]) to evaluate how emulation affects the behaviour of digital objects compared to their behaviour in the original environment. The authors of [GR12] emphasise that the external dependencies of digital objects must be kept unchanged to repeat the rendering deterministically. They define levels of comparison of digital objects that impact which data and from where has to be extracted to compare two renderings (see Figure 5.2). The framework is formulated as follows:

1. Describe the original environment

   The original system’s hardware and software components have to be documented along with all their settings to allow the recreation in an emulated environment.

2. Determine external events that influence object’s behaviour

Only for objects with deterministic behaviour it is possible to ensure that differences in rendering compared to the original environment are results of the emulated environment.

3. Decide on what level to compare the digital object

   As the digital object is available in various rendered forms, it is necessary to select the one that is most suitable for the digital objects that have to be preserved and their desired form of representation.

4. Recreate the environment in emulation

   An emulator on a host system has to be configured to match the hardware and software configuration of the original environment.

5. Apply standardized input to both environments

   Depending on the digital object the most suitable way to apply automated input has to be selected. Then, the input to the original object has to be recorded and applied to the emulated environment.

6. Extract significant properties

   The significant properties of the rendered object have to be extracted from the emulation environment.

7. Compare the significant properties

   The significant properties that have been extracted automatically as well as those that were not measured automatically but manually have to be compared, evaluated, and documented.

The VFramework presented in Chapter 4 is a refinement of this framework for potentially distributed scientific workflows. The crucial difference is that the VFramework assumes that access to both environments is not possible at the same time and thus the *context model* (see Section 2.6) is used to describe the ground truth data used for verification and validation when the workflow is re-executed in a new environment. Furthermore, the VFramework focuses not only on validation of the final result of processing, but also on verifying in what way the result was produced. The VFramework is also orchestrated by several tools that automate its application, while the framework described in [GR12] is a conceptual framework that requires manual application.

2.5.2 Workflow and process preservation

Digital preservation has emerged mainly from memory institutions and the cultural heritage sector [NU03]. However, it was recognized that it affects all organizations that manage information over time, and as such it affects contemporary organizations in which information systems provide important support to the business [SMA^{+}13]. The obsolescence of not only hardware and software, but also proper context in which the data can be interpreted was identified as a threat to reproducibility of scientific investigations, as well as traceability of business decisions. For this reason projects like *TIMBUS*, or *wf4ever* were funded to devise a way of documenting workflows and processes, so that they can be re-executed in the future.

The *wf4Ever* project [BCG+12] addressed challenges associated to 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) [BZG+15] 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 [MMR14] a mapping between the ROs and the context model (see Section 2.6) 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.

In such settings the preservation framework [SMA+13] [TIM13] developed in the *TIMBUS* project can be used. It not only enables understanding of the process and evaluation of risks, but also supports its redeployment. Figure 2.2 depicts the process preservation framework. It consist of three phases:

1. Plan

   The preservation of business processes is based on a risk management approach. The risk analysis is performed on different levels within the organisation, including the detailed analysis of technical risks of current implementations. For a detailed analysis, the relevant components and dependencies are captured in the context model, which defines technical, organizational and legal aspects of a process.

2. Preserve

   In the preservation phase, the business process is captured from the original environment. The dependencies and relationships between the components of the business process need to remain intact over time. In order to verify the characteristics, behaviour and performance of the captured process in the future, validation data is captured.

3. Redeploy

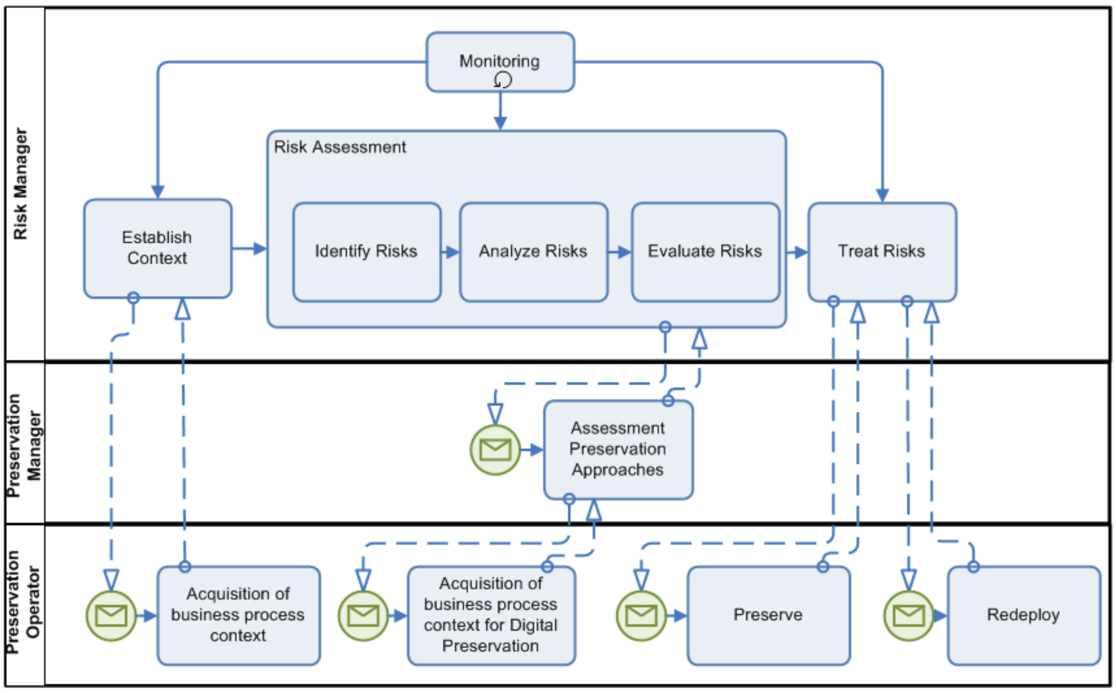


Figure 2.2: Overview of the TIMBUS preservation framework [TIM13].

The redeployment phase defines the reactivation of the preserved business process in a new environment at some time in the future. The required services need to be redeployed on a new (hardware and software) infrastructure and the process needs to be deployed. The redeployed business process is validated using measures that have been captured from the original process.

The VFramework was designed to fit into the preservation framework. Creation of the context model that is required for the preservation framework is also a part of the VFramework. Furthermore, the tools developed for analysis of workflow and context capturing enable automation of VFramework application.

Despite the fact that the preservation framework was designed to be used with business processes it can be also applied for preservation of scientific workflows [SMA+13]. As discussed in Chapter 4 the long term considerations related to preservation of workflows apply to short term settings when the workflows are shared between researchers. When the workflows are shared, they are extracted from their original environment, placed in a location that can be accessed by other researchers, and then redeployed in a new environment of a scientist re-using the workflow. This overlaps with the three phases of the preservation framework.

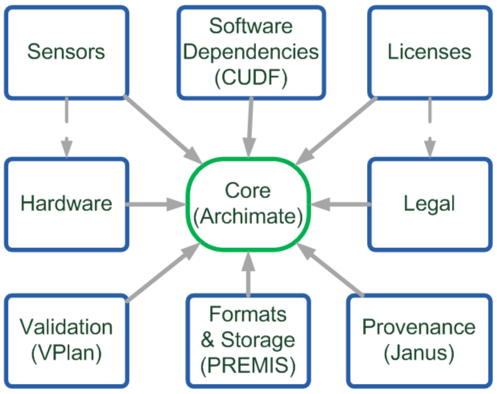


Figure 2.3: Overview of the context model: core ontology and extensions [MAC+15].

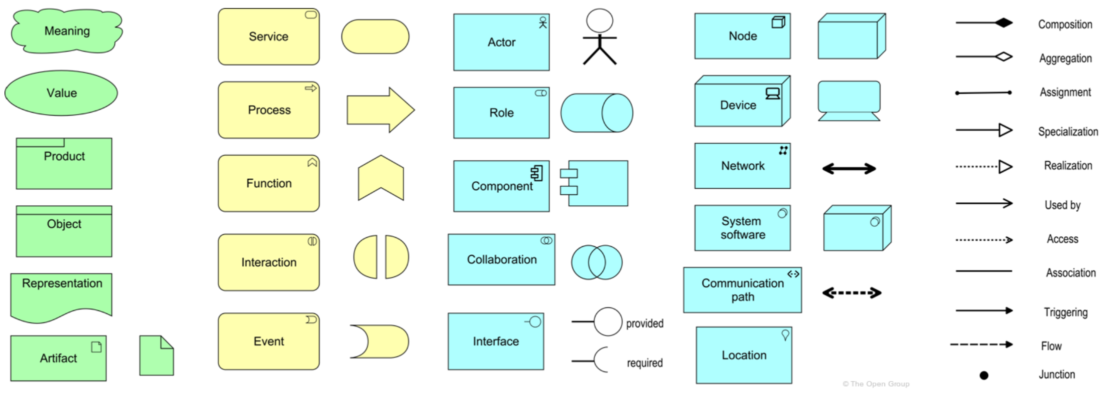


Figure 2.4: Overview of the ArchiMate concepts [Gro12].

2.6 Execution Context

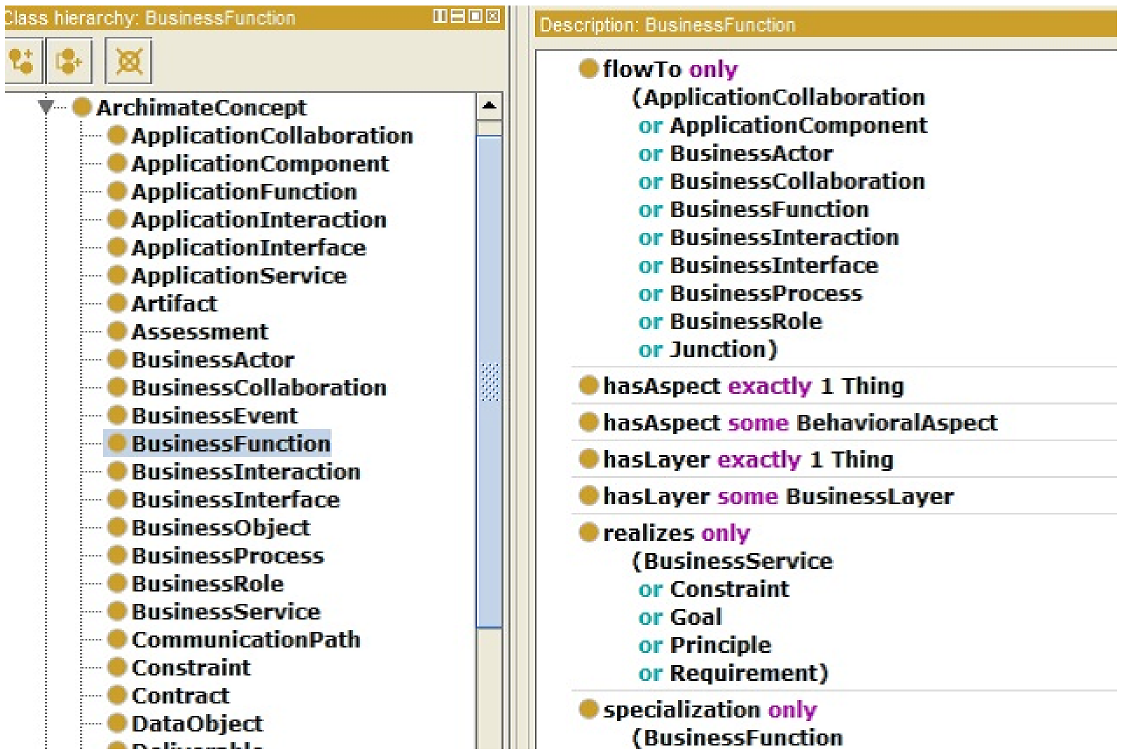
The workflow context describes the environment in which the workflow executes. The context ranges from software and hardware implementing the workflow, to laws and regulations affecting its usage [MAC+15].

Enterprise architecture deals with context modelling. Enterprise architecture can be defined as "a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure" [Lan05]. The most popular enterprise modelling languages are: ArchiMate [Gro12], The Open Group Architecture Framework (TOGAF) [The09], and the US Department of Defense Architecture Framework (DODAF) [Dep07]. Each of them models processes, their application and infrastructure components, as well as elements of the organisation affecting the process design.

The authors of [MAC+15] state that in the process preservation domain a single model cannot be used to describe the complex context of a workflow execution. For this reason

Table 2.3: Mapping of Archimate Elements to OWL Elements [MAC+15].

Archimate Element	OWL Element
Concept	Class
Relation	Object Property
Concept Instances	Individuals


lass hierarchy: BusinessFunction

- ArchimateConcept
  - ApplicationCollaboration
  - ApplicationComponent
  - ApplicationFunction
  - ApplicationInteraction
  - ApplicationInterface
  - ApplicationService
  - Artifact
  - Assessment
  - BusinessActor
  - BusinessCollaboration
  - BusinessEvent
  - BusinessFunction
  - BusinessInteraction
  - BusinessInterface
  - BusinessObject
  - BusinessProcess
  - BusinessRole
  - BusinessService
  - CommunicationPath
  - Constraint
  - Contract
  - DataObject

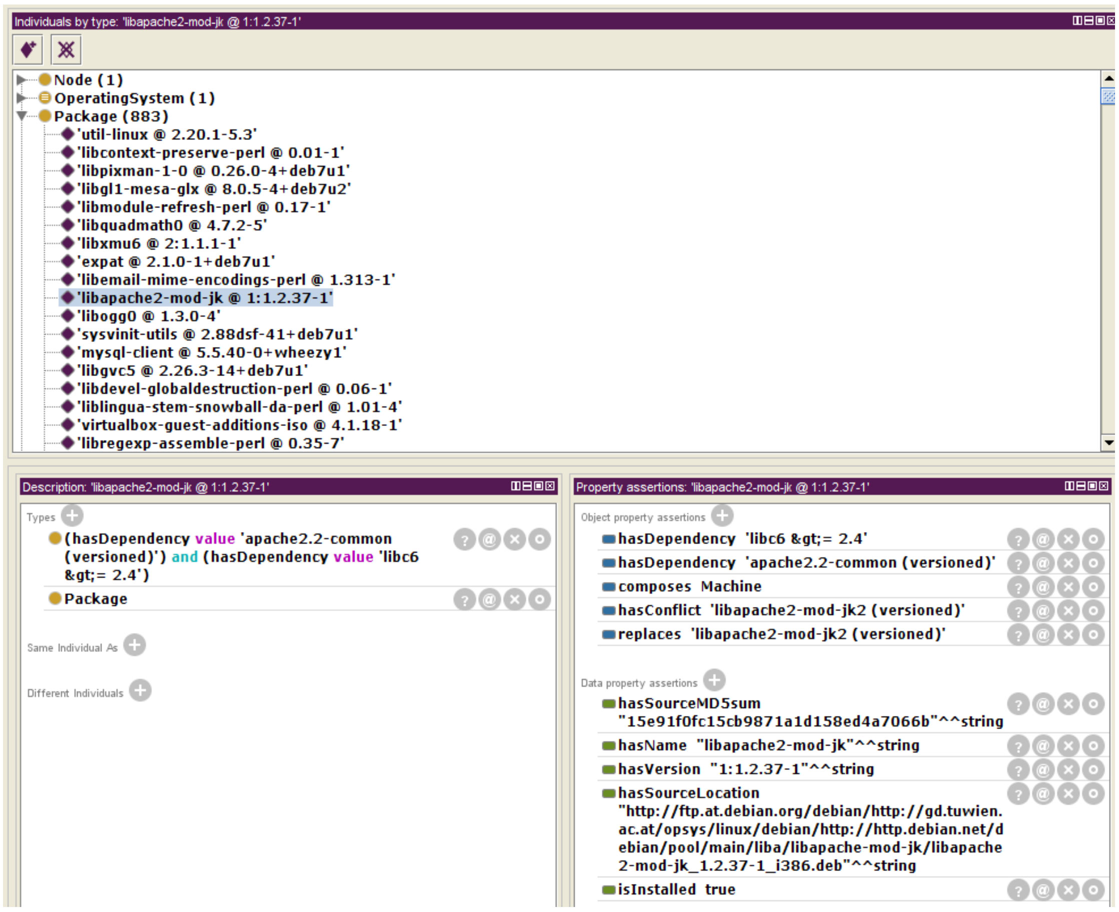
Description: BusinessFunction

```
flowTo only
   (ApplicationCollaboration
    or ApplicationComponent
    or BusinessActor
    or BusinessCollaboration
    or BusinessFunction
    or BusinessInteraction
    or BusinessInterface
    or BusinessProcess
    or BusinessRole
    or Junction)
hasAspect exactly 1 Thing
hasAspect some BehavioralAspect
hasLayer exactly 1 Thing
hasLayer some BusinessLayer
realizes only
   (BusinessService
    or Constraint
    or Goal
    or Principle
    or Requirement)
specialization only
   (BusinessFunction
```

Figure 2.5: OWL representation of the ArchiMate concepts and relations, specifically the *Business Function* Class and respective Object Properties [MAC+15].

they developed the *context model*, which was originally described in [SMA+13] and later refined in [MAC+15], 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 models, depending on the requirements of the application domain. The meta-model is implemented as an OWL ontology. Ontology mapping allows for a realisation of the model integration. This architecture is depicted in Figure 2.3. The above described structure is referred to as *context meta model*, while instances of it are referred to as *context model*.

The core model (cf. Figure 2.3) is based on the ArchiMate [Gro12] enterprise modelling language. It includes concepts on business and technical layers ranging from services, process steps, and data exchanged between the steps, down to the technical infrastructure,



Individuals by type: 'libapache2-mod-jk @ 1:1.2.37-1'

```
Node (1)
OperatingSystem (1)
Package (883)
    'util-linux @ 2.20.1-5.3'
    'libcontext-preserve-perl @ 0.01-1'
    'libpixman-1-0 @ 0.26.0-4+deb7u1'
    'libgl1-mesa-glx @ 8.0.5-4+deb7u2'
    'libmodule-refresh-perl @ 0.17-1'
    'libquadmath0 @ 4.7.2-5'
    'libxmu6 @ 2:1.1.1-1'
    'expat @ 2.1.0-1+deb7u1'
    'libemail-mime-encodings-perl @ 1.313-1'
    'libapache2-mod-jk @ 1:1.2.37-1'
    'libogg0 @ 1.3.0-4'
    'sysvinit-utils @ 2.88dsf-41+deb7u1'
    'mysql-client @ 5.5.40-0+wheezy1'
    'libgvc5 @ 2.26.3-14+deb7u1'
    'libdevel-globaldestruction-perl @ 0.06-1'
    'liblingua-stem-snowball-da-perl @ 1.01-4'
    'virtualbox-guest-additions-iso @ 4.1.18-1'
    'libregexp-assemble-perl @ 0.35-7'
```

Description: 'libapache2-mod-jk @ 1:1.2.37-1'

```
Types
    (hasDependency value 'apache2.2-common
    (versioned)') and (hasDependency value 'libc6
    &gt;= 2.4')
    Package

Same Individual As

Different Individuals
```

Property assertions: 'libapache2-mod-jk @ 1:1.2.37-1'

```
Object property assertions
    hasDependency  'libc6 &gt;= 2.4'
    hasDependency  'apache2.2-common (versioned)'
    composes  Machine
    hasConflict  'libapache2-mod-jk2 (versioned)'
    replaces  'libapache2-mod-jk2 (versioned)'

Data property assertions
    hasSourceMD5sum
    "15e91f0fc15cb9871a1d158ed4a7066b"^^string
    hasName  "libapache2-mod-jk"^^string
    hasVersion  "1:1.2.37-1"^^string
    hasSourceLocation
    "http://ftp.at.debian.org/debian/http://gd.tuwien.
    ac.at/opsys/linux/debian/http://http.debian.net/d
    ebian/pool/main/liba/libapache-mod-jk/libapache
    2-mod-jk_1.2.37-1_i386.deb"^^string
    isInstalled  true
```

Figure 2.6: An excerpt of the CUDF extension ontology created for a Debian-based system.

including hardware, software, and files. An overview of ArchiMate concepts is depicted in Figure 2.4. Sometimes such an ontology is referred to as an upper level ontology [Ant15]. The ArchiMate elements were converted into OWL Elements using a mapping presented in Table 2.3. Furthermore, restrictions were added into the object properties to facilitate reasoning and verification of model correctness. Figure 2.5 depicts an excerpt of the OWL representation of ArchiMate in the Protégé ontology editor. The *Business Function* class is highlighted in the left pane and respective properties, including restrictions are presented in the right pane.

Use cases may require specific information to be added to the context model, for this reason any ontology can be linked to the core model. The extension ontologies use a more specific language to describe particular aspects of the workflow. In Figure 2.3 we depicted eight extension ontologies that can be used to extend the model. We describe below only two extension ontologies that we later use during the VFramework application. The other ontologies, for example the VPlan ontology, we introduce later, when we actually

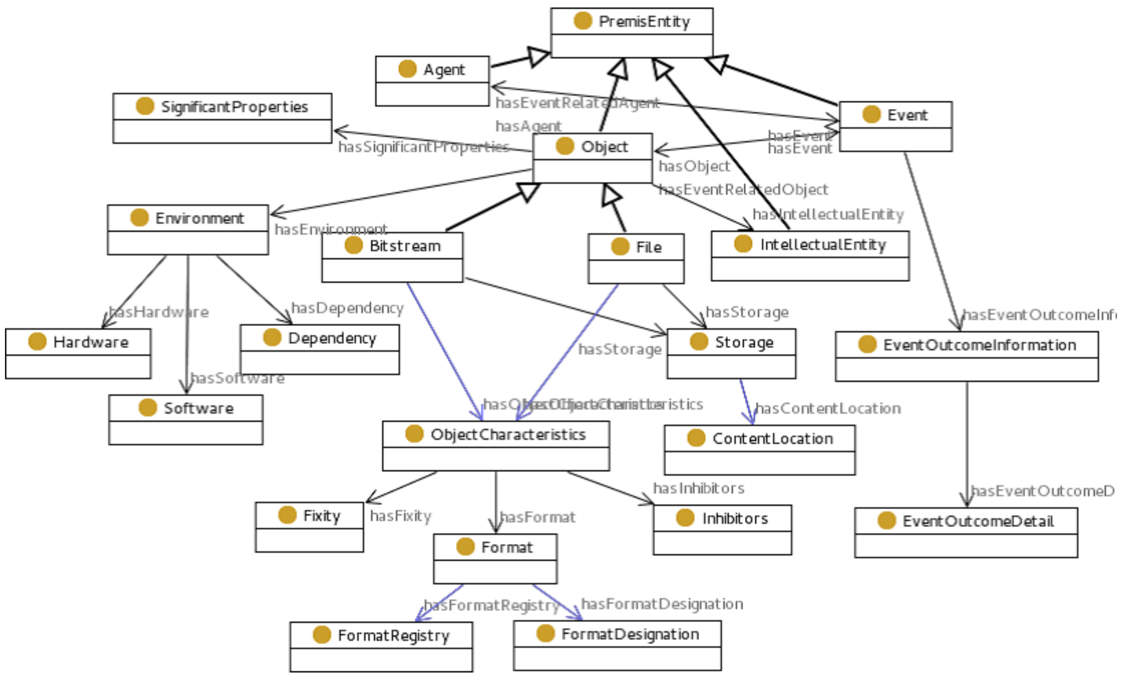


Figure 2.7: An excerpt of the PREMIS ontology [MAC+15].

need them.

The ontology on software dependencies is one of the extension ontologies. It provides detailed modelling support for various relations such as dependencies or conflicts between packages. It is based on the Common Upgradeability Description Format (CUDF) [TZ08]. Figure 2.6 depicts an instance of this ontology created for a Debian-based system viewed in Protégé. There are 883 packages installed in the analysed system. The one selected in the figure is *libapache2-mod-jk*. This package requires two others to be present in the system, namely: *libc6* in version greater than 2.4 and *apache2.2-common package*. One can also see that this package replaces *libapache2-mod-jk2* and that these two packages cannot be installed at the same time, because there is a conflict between them. Furthermore, for each package there is a source location stating from where it can be downloaded.

For the purpose of file format description, the authors of [MAC+15] use the PREMIS Data Dictionary [PRE08] that is also available in the form of an ontology. When workflows are executed, data is read, modified, and written. Information on the file format of this data helps in analysis of workflow executions, because specialised tools can be applied that take the data format into account. Thus we can better identify differences between two workflow executions that used or created different data. An excerpt of the PREMIS ontology is depicted in Figure 2.7. The PREMIS data dictionary defines five types of entities: *Intellectual*, *Object*, *Event*, *Agent*, and *Rights*. It then defines 45 concepts belonging to these types, as well as relations and data properties. In Section 4 we describe how we instantiate this ontology.

The information stored in the context model can be accessed using publically available tools for ontology manipulation like for example Protege[1]. The context model can be created using PMF that is described in Section 2.7. Furthermore it can be created using a range of extractors and converters implemented by the *TIMBUS* project that are described in [TIM14].

In this dissertation we use the context model to describe the workflow execution context, because it allows us to identify the software that was involved during workflow execution, as well as analyse the workflow structure and on that basis generate validation requirements (see Chapter 5).

2.7 Workflow execution monitoring

In this section we present tools for monitoring of workflow executions. These tools identify dependencies of a workflow that must be verified when the workflows are re-executed.

Davison [Dav12] motivates the necessity of recording which files were accessed during execution of scripts, and which source code version was used to perform the experiment with the fact that the researchers are very often not aware of the system configuration and software versions when performing their computations. For that purpose they implemented Sumatra, a python library capable of capturing the execution context of python scripts. According to the authors the context consists of:

- SVN version number of the analysed python script
- Python libraries loaded by the script
- Input files used by the script
- Output files produced by running the script
- Platform information
  - processor architecture
  - operating system type and its kernel version
  - IP address of the local system

The implementation of Sumatra is limited to the analysis of python scripts and thus is not universally applicable. The significant part of the context is in fact the provenance data that is describing data products used and created during script execution. The platform information is based on static system description and does not identify the resources

[1]http://protege.stanford.edu

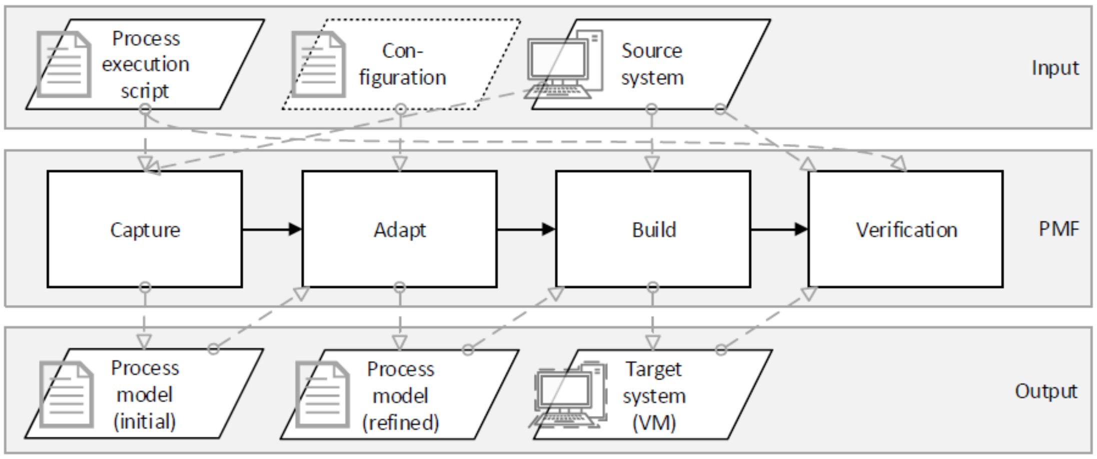


Figure 2.8: Process Migration Framework [Bin14].

actually used during script execution (apart from the Python libraries). Furthermore, in package based systems like Linux it is not sufficient to provide just a kernel version to specify a system configuration. A list of packages installed in the system that are used during workflow execution is needed to complement this information.

For verification and validation of workflow re-executions we need a tool that creates a complete list of dependencies used during workflow execution regardless of their implementation. For Linux-based systems there is the *strace* [Rob08] tool that attaches to a monitored system process and intercepts all system calls. The system calls are used to load resources, access files, create further processes, open connections to databases or external hosts, and so on. For every workflow execution a system process is created that can split into further sub processes creating a tree of processes. By monitoring this tree and its system calls we obtain a complete list of workflow dependencies. Furthermore, this method is independent of the implementation of the workflow. The downside of this approach is the amount of data produced that must be filtered and processed before it can be used for verification of workflow re-executions. There are two tools that are based on process monitoring using *strace*, namely *CDE* and *PMF*.

The CDE [Guo11] output aggregates information detected using *strace* and therefore is less detailed. Furthermore, system calls that are used for tracking resources or details of network connections are omitted. Therefore, 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. CDE gathers only files and binaries that were used in the execution. CDE also modifies analysed processes, because it requires users to prepend CDE commands to scripts or binaries.

The process migration framework (PMF) [Bin14] [BSR14] 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 [MAC+15]. PMF records which operating system processes were called, what files were accessed or created, and which connections to other services were opened. This data contains exact commands and their parameters, as well concrete addresses of services.

Furthermore, PMF identifies which of the identified files are parts of the default operating system software sources, 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.

When the PMF is run in the original environment to create the context model it downloads packages that were identified in the system, as well as extracts files from the original system that were used for processing. Using this data and Vagrant[2] it can create a new virtual machine that is configured in the same way as the original one was.

PMF is depicted in Figure 2.8 and consists of four steps:

- *Capture* – Workflow environment is analysed and described using the context model.
- *Adapt* - The model created in the previous step is adapted by, for example, replacing software. This step is optional, but adds the flexibility to handle changing requirements, like in tool versions.
- *Build* - A virtual machine that corresponds to the refined model and in which the process can be executed is created.
- *Verification* – The redeployed workflow on the target system is verified whether it shows the same behavior as on the source system.

In the VFramework we use the *Capture* step of the PMF twice: first, to create the context model of the original workflow execution, second, to create the context model of the workflow re-execution. We compare these models to verify the workflow re-execution (see Section 6.1).

The porting process is beyond the scope of our investigations and therefore the workflows can be redeployed either using PMF or manually. The VFramework takes into account both possibilities and hence the *Adapt* and *Build* steps of the PMF are optional. The *Verification* step of the PMF corresponds to the *Verify environment* and *Validate workflow* steps of the VFramework.

[2]https://docs.vagrantup.com/v2/boxes.html

2.8 Changes in Web Services

Scientific workflows may use external web services to complete tasks. For that purpose the workflows exchange data with web services that are often provided by a third party. The workflow owners have no control over such services and hence the replicability of workflow executions is at danger when the web services change their functionality or become unavailable.

In Section 4.2.3 we present an approach to verify workflows that use external web services and discuss in what way changes in web services can affect the workflow performance. We present there the Web Service Monitoring Framework that we devised for monitoring of web services. We use the data collected during framework application to create mock-ups of web services, which are used when the process is re-executed and the original web service cannot be used any more.

In this section we provide an overview of different monitoring approaches, as well as summarize possible extensions to web services that aim at enabling repeatable executions of web service dependent workflows.

2.8.1 Web service monitoring

The framework presented in [CFCB10] generates and executes automatically tests for conformance testing of a composite of web services described in BPEL. This approach was combined with passive testing, which verifies time traces with respect to a set of constraints [CCFM11]. Both solutions are limited to web services that are implemented according to the BPEL specification. Verification of behavioural conformance of services during run-time is presented in [DRK09]. Stream X-machines are applied to check the control flow of a web service and the generated responses. The traffic is intercepted from a live system and continuous monitoring for changes is performed. The Stream X-machine needs to be developed manually, and requires access to the web service implementation, which limits the application of this method. The authors also provide a classification of web services. Three major criteria are distinguished: conversational / non-conversational, private-state / shared-state, transient-state / persistent-state. In our work, we consider these criteria as sub-criteria of the stateful / stateless criterion.

The WS-TAXI framework[BBMP09] combines the coverage of web service operations with data-driven test generation. It is able to deliver a suite of test messages ready for execution, generated using a WSDL specification. WS-TAXI generates and uses purely synthetic data which may be quite different from the data exchanged in a process. It is thus more suited for web service development and testing, rather than monitoring of already deployed SOA (Service Oriented Architecture) solutions.

Monitoring whether Service Level Agreement (SLA) conditions are fulfilled by web services is a problem related to monitoring web services for changes. In [GS10] a run-time monitoring framework which allows accessing exchanged messages and comparison against designed scenarios is presented. The focus is on Quality of Service aspects, for example of a time-out mechanism detecting unavailability of the service. In [GKS11] emphasis is put on detection of violations at the functional level. SLAs are described formally using temporal logic and are used to verify the behaviour of web services at runtime, for example maximum response time.

The shortcoming of the above mentioned solutions lies in the fact that they demand specific knowledge on the kind and the nature of the web service. Also, the kind of change which will be monitored, is required to deploy the proper solution. In many scenarios, however, only the URL and interface of the service are known, but no information on whether the web service is conversational, stateful, deterministic, etc.

The Web Service Monitoring Framework (WSMF), presented in [MMR15a] and in Section 4.2.3, is thus designed to allow investigation of any kind of web service, and to facilitate reasoning about the nature of a service. If the web service is deterministic, the monitoring process can be launched and all four types of changes (see Table 4.1) can be detected. Otherwise, the monitoring framework is not able to detect any functional changes, but the other three types of changes can still be monitored.

2.8.2 Web service extensions

Apart from monitoring the technical level of services, several improvements to the specification of web services, which should lead to a higher sustainability of workflows, as well as reduction of the need for continuous monitoring, have been proposed.

The Universal Description Discovery and Integration (UDDI) is a registry which holds information on registered web services. However, the registry does not contain sufficient additional information on the service that would allow users to obtain information on the nature, behaviour, or quality of the service. Several proposals aim at enriching the purely functional description of web services (bindings, ports, etc.) with Quality of Service (QoS) aspects, for example timing aspects, availability, reputation [CP09] and pricing [LNZ04]. [W3C03] specifies requirements for QoS for web services. It lists 13 points which should be fulfilled, but none of them concerns guaranteeing continuity or non-modifiability.

Another approach is to facilitate versioning of web services. Yet in this case, approaches do not aim at specifying a way to interweave versioning into web service specification, but present workarounds to deal with the currently underspecified web service standards [KML06]. One of the exceptions is [KAC03], which provides functional requirements for a registry which notifies clients when a version of an interface changes. [KML06] is a good example of the current common view on versioning: versioning is understood as a change of interface. Changes in functionality while the interface stays the same are not considered. In [MMUR14] we thus introduced a concept of Resilient Web Services which aims at extending specification of web services, addressing the challenge of functional changes.

CHAPTER 3

VFramework

In this chapter we introduce the VFramework – a framework for verification and validation of scientific workflow re-executions. 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 define the VFramework and the VPlan in Chapters 4, 5, and 6.

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, 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.

Figure 3.1 provides a high level overview of settings in which the VFramework is applied and shows how this influenced its design. The workflow running in the original environment is 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 moved to a new environment, because a scientific experiment is replicated by other scientists who found the workflow on a workflow portal. In many cases a direct contact between the workflow owner and the workflow replicator may not always be possible. The authors of [CP14] show that reasons can be as trivial as a change of e-mail address of the workflow owner who changed his affiliation.

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 research funder regulations all data including workflows must be preserved. The preserved workflows can be potentially run at a later

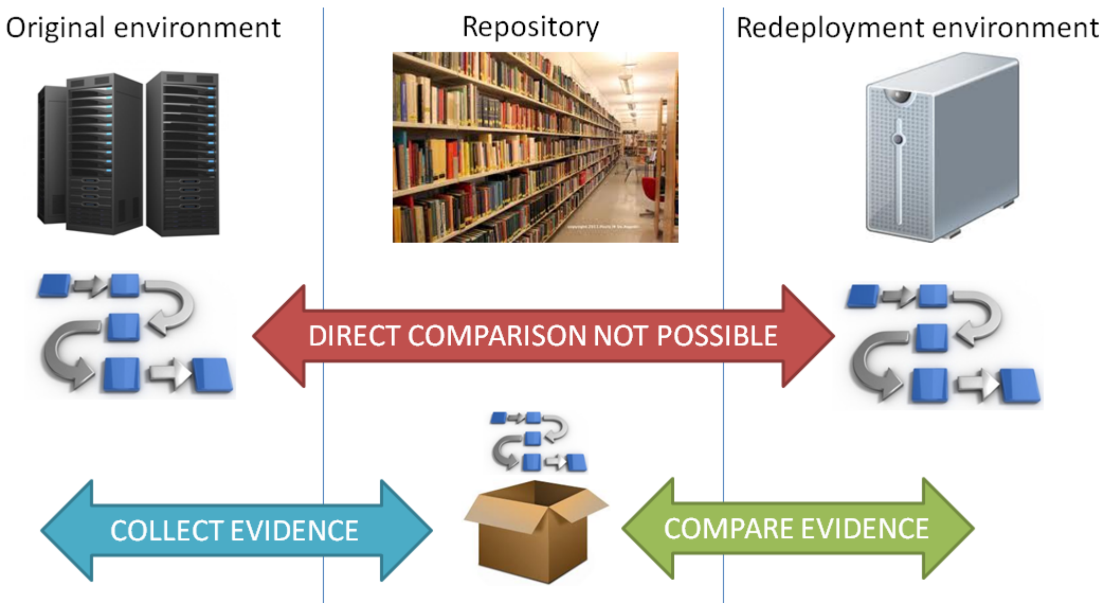


Figure 3.1: Overview of workflow verification and validation.

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 may not be feasible. 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 3.2. 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, which is described in detail in Section 2.6, 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.

Figure 3.3 depicts five parts of the context model that are created during the VFramework application. The workflow model core is the central element that describes the workflow model and is a basis for extensions providing information on validation requirements, workflow instance data, workflow dependencies, and identified file formats. We use the colour coding throughout the dissertation to distinguish between the parts of the context model, for example, we use the light green colour to depict elements that are workflow dependencies.

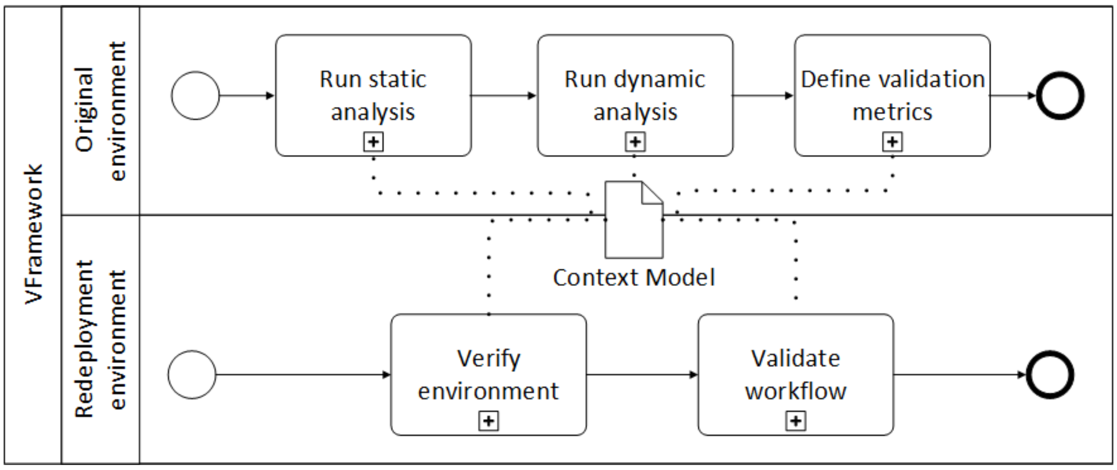


Figure 3.2: Framework for verification and validation of re-executed workflows.

In Chapters 4, 5, and 6 we describe in detail the VFramework steps and show how we apply the VFramework to Taverna workflows using the context model and tools automating its creation, and processing. The description is structured as follows:

- In Chapter 4 we describe the *Run static analysis* and the *Run dynamic analysis* steps of the VFramework. These steps capture the original execution of the workflow and document the environment in which it took place.
- In Chapter 5 we describe the *Define validation metrics* step of the VFramework. We describe a way to automatically generate validation requirements and present the VPlan ontology that we created for description of validation requirements.
- In Chapter 6 we describe the *Verify environment* and the *Validate workflow* steps of the VFramework that are completed when the workflow is re-executed in a new environment to verify and validate its re-execution.

3.2 Running example

In this section we present a workflow that we use as a running example on which we define the VFramework and the VPlan in Chapters 4, 5, and 6. In the remainder of the dissertation we refer to it as "the weather workflow". We begin with explaining our motivation for choosing the given workflow and then we describe in detail each of its elements.

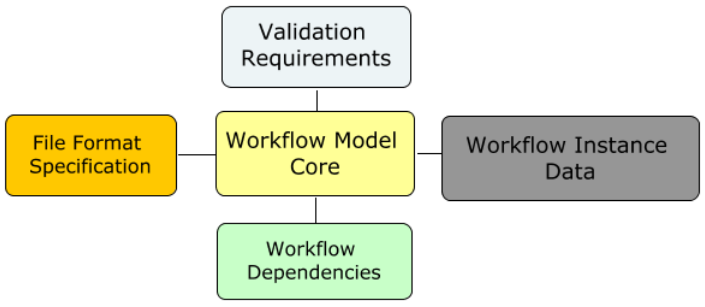


Figure 3.3: Overview of the context model parts instantiated during the VFramework application.

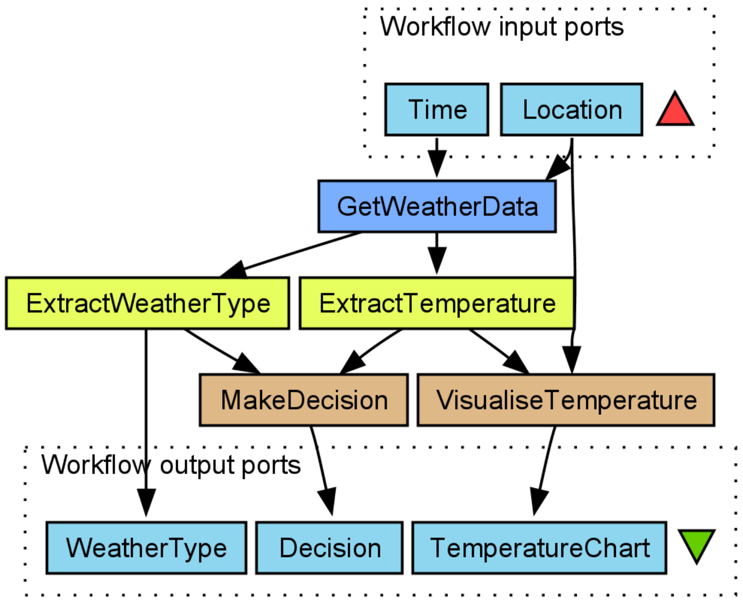


Figure 3.4: Weather workflow used as the running example.

3.2.1 Motivation

We created a Taverna workflow presented in Figure 3.4. 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 [HTT09] 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.

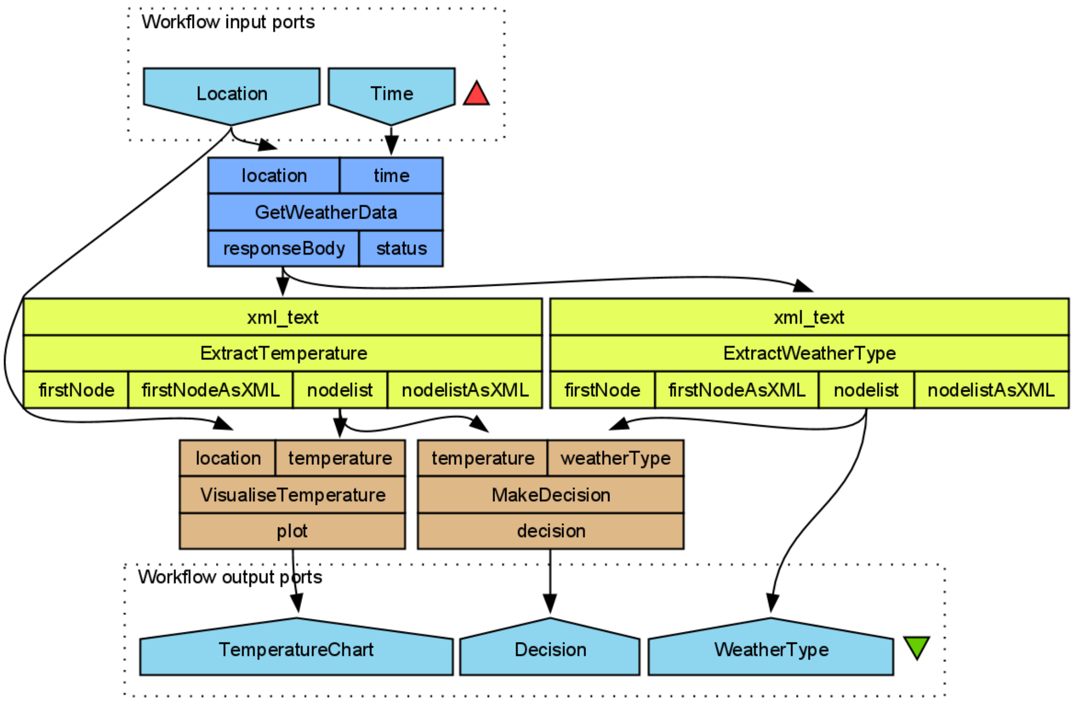


Figure 3.5: Weather workflow steps and their outputs.

- 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 [MR15a]. 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.

3.2.2 Workflow implementation

Figure 3.4 depicts inputs, steps and outputs of the weather workflow. Figure 3.5 provides more details on the same workflow and additionally depicts inputs and outputs of each workflow step. For example, the workflow step *MakeDecision* has two workflow step inputs: *temperature* and *weatherType*; and one workflow step output that is called *decision*. Both figures were generated by Taverna and depict the same workflow varying in the level of detail. In Chapter 7 we present workflows in details whenever there is a need to discuss particular workflow step outputs, otherwise we depict workflows using only workflow steps.

The workflow uses weather forecasts provided by the Met Office[1], which is the United Kingdom's national weather service. There are over 5000 available locations in the UK. A weather forecast is issued for each location in 3-hourly intervals out to five days[2]. The past data is not available. Hence, we cannot check, for example, yesterday's forecast today.

In order to run the workflow we need to set values to the workflow inputs, which are:

- *Time* – date and time of a weather forecast we want to use, for example *2015-10-12T18Z*. Which weather forecasts are available at the moment can be checked using one of the other web services (not in scope of this workflow).
- *Location* – unique identifier of a location in the UK. A list of locations and their identifiers can be found on the Met Office web sites. Examples of identifiers are: *322690* for Ramsgate, or *3772* for Heathrow.

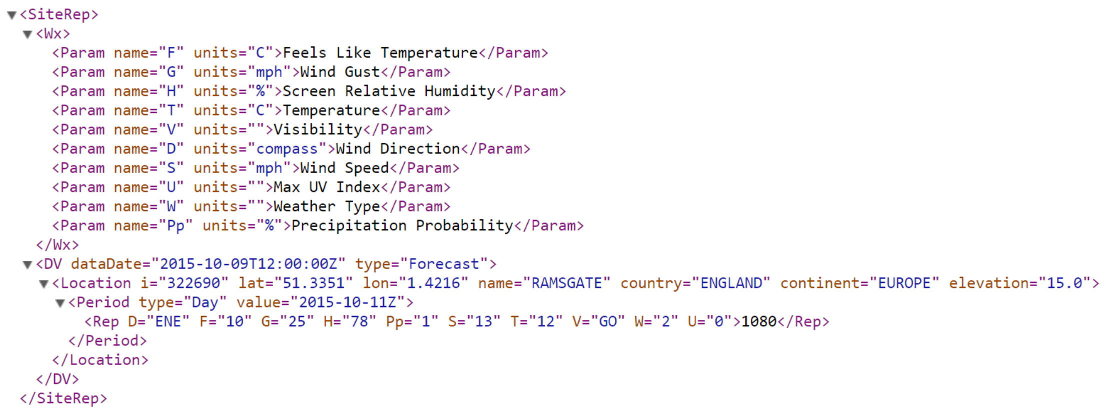
The workflow inputs are connected to the inputs of the *GetWeatherData* workflow step. This workflow step is a REST Service that is a built-in Taverna service which can be customised for accessing REST web services. The service configuration requires providing a URL[3] of the Met Office web service and annotating which parameters of the URL have to be substituted with values read from the inputs of the workflow step. The *GetWeatherData* workflow step has two outputs: *responseBody* and *status*. Both of them are fixed for this kind of services. The first one contains the response from the web service, while the other informs whether the operation was successful.

The *responseBody* output of the workflow step *GetWeatherData* is connected to the inputs of two steps, namely: *xml text* of *ExtractTemperature* and *xml text* of *ExtractWeatherType*. Both of these steps are XPath Services, which are built-in and customisable Taverna services. Each of them defines an XPath expression that is used to extract information from an XML document. The responses from the Met Office web service are XML

[1] http://www.metoffice.gov.uk

[2] http://www.metoffice.gov.uk/datapoint/product/uk-3hourly-site-specific-forecast/detailed-documentation

[3] http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/xml/{location}?res=3hourly&time={time}&key={key}



```
▼<SiteRep>
  ▼<Wx>
     <Param name="F" units="C">Feels Like Temperature</Param>
     <Param name="G" units="mph">Wind Gust</Param>
     <Param name="H" units="%">Screen Relative Humidity</Param>
     <Param name="T" units="C">Temperature</Param>
     <Param name="V" units="">Visibility</Param>
     <Param name="D" units="compass">Wind Direction</Param>
     <Param name="S" units="mph">Wind Speed</Param>
     <Param name="U" units="">Max UV Index</Param>
     <Param name="W" units="">Weather Type</Param>
     <Param name="Pp" units="%">Precipitation Probability</Param>
   </Wx>
  ▼<DV dataDate="2015-10-09T12:00:00Z" type="Forecast">
    ▼<Location i="322690" lat="51.3351" lon="1.4216" name="RAMSGATE" country="ENGLAND" continent="EUROPE" elevation="15.0">
      ▼<Period type="Day" value="2015-10-11Z">
         <Rep D="ENE" F="10" G="25" H="78" Pp="1" S="13" T="12" V="GO" W="2" U="0">1080</Rep>
       </Period>
     </Location>
   </DV>
 </SiteRep>
```

Figure 3.6: Response from the Met Office web service.

documents. An example of a response is presented in Figure 3.6. The *ExtractTemperature* workflow step uses the following XPath expression to read the temperature from a response: */SiteRep/DV/Location/Period/Rep/@T*. In the given example, the temperature's value is 12. The outputs of the XPath services are fixed and provide results of executing the XPath expression in different forms: either only the first matching node is returned or all matching nodes. The results can be formatted as plain text or as XML. Users decide which form to use by linking the corresponding output with further steps of the workflow. In the weather workflow we use the list of all matching nodes formatted as plain text.

The *nodelist* output of the *ExtractTemperature* workflow step is connected to the *temperature* inputs of two workflow steps: *VisualiseTemperature* and *MakeDecision*. Both of these workflow steps are Beanshells. Additionally, the workflow input *Location* is connected to the *location* input of the *VisualiseTemperature* workflow step, and the *MakeDecision* workflow step has the *nodelist* output of the *ExtractWeatherType* workflow step connected to its *weatherType* input.

The *VisualiseTemperature* workflow step contains a script that uses an external Java library *chart-1.0-jar-with-dependencies.jar* to create a PNG chart depicting a temperature in a given location. The temperature's value and the location are read from the workflow step inputs. The result of visualization is returned through the *plot* workflow step output. To use the external Java library we had to add the appropriate JAR file to a *lib* subdirectory of Taverna.

The *MakeDecision* workflow step contains a script that implements a very simple decision tree that uses two parameters: temperature and weather type. Both of these values are read from the inputs of the workflow step. There are two possible decisions: "Stay at home", or "Go cycling". If the temperature is above zero and the weather type is between zero and seven, the decision is to go cycling, otherwise staying at home is recommended.

The workflow has three outputs:

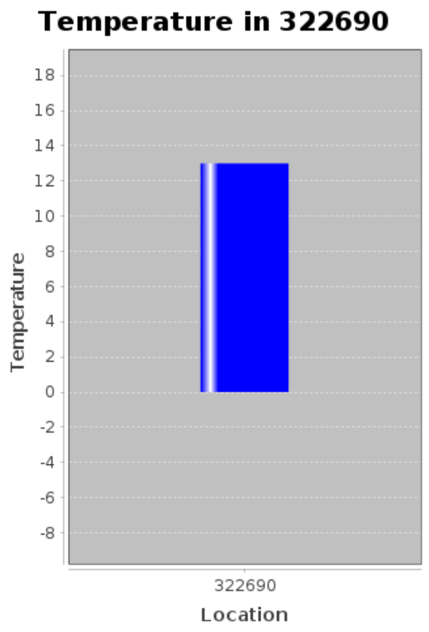


Figure 3.7: Workflow output *TemperatureChart* depicting temperature in Celsius degrees in Ramsgate.

- *TemperatureChart* - visualisation of the temperature for a specified location at a given date and time. Figure 3.7 presents an example of a temperature chart.
- *Decision* - recommendation whether to "Go cycling" or "Stay at home" made by the decision tree using temperature and weather type for a specified location at a given date and time.
- *WeatherType* - integer from 0 to 30 encoding a type of weather using the Met Office code definitions[4] for a specified location at a given date and time. For example value 1 denotes a sunny day, while 27 denotes heavy snow.

[4]http://www.metoffice.gov.uk/datapoint/support/documentation/code-definitions

CHAPTER 4

Capturing original workflow execution

In Section 2.3 we presented an overview of scientific workflows that require workflow management systems for execution. Besides these, workflows can also be implemented as scripts using various programming languages, or can be verbally defined in text documents and later executed manually. To verify and validate each workflow type, one can develop methods and tools specific to a particular workflow implementation. This requires good understanding of the workflow specifications and detailed information on the workflow execution environment. Development of methods and tools for each workflow type would result in duplication of efforts and multiplicity of similar but not compatible tools. This is due to the similarities between the workflow systems and the fact that all of the workflows are finally executed by an operating system that transforms them into system processes.

Hence, a better approach is to provide a common way for verification and validation of workflows. This can be achieved by transforming workflow specifications into a common model which analysis can be performed in the same way regardless of the original workflow specification. The process of transforming workflow specifications requires identification of mappings between the concepts used and development of tools that convert workflow specifications into the common model. In our analysis we show how Taverna workflows are transformed into the context model that is a common model that can be used to model various types of workflows.

Depending on the original workflow specification, we can obtain different levels of details. For Taverna workflows we can easily identify the workflow steps and their outputs, because they are defined in the Taverna workflow model, but in case of scripts these are not always obvious, but still can be identified. Furthermore, some workflow specifications provide explicit information on dependencies (e.g. *import modules* in Python, or explicit

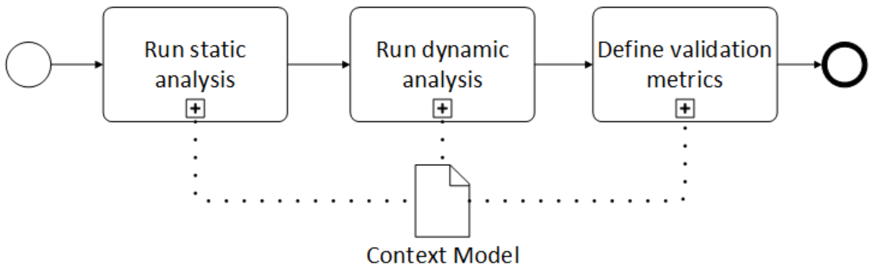


Figure 4.1: VFramework steps performed in the original environment.

documentation of Beanshell dependencies in Taverna), while for others these dependencies can only be identified during workflow execution.

It is also a complex task to identify which information on workflow dependencies must be captured. For some workflows it may be sufficient to capture only the workflow description and information on the operating system version being used, while other more complex workflows that use specific tools may require documenting specific versions of the tools together with their configurations, or even particular system libraries being loaded. These tools in turn may depend on further tools and services that must be accessible during workflow execution. Some of the services may be hosted on different machines and may belong to different owners, especially when the workflows use distributed infrastructure to complete their steps. In such cases the availability of information can be limited, but we still need to be able to verify and validate the workflow.

For this reason we need to perform the static workflow model analysis to identify the workflow structure including its steps and dependencies, as well as the dynamic analysis of workflow instance execution to identify further dependencies. When we identify what is needed for a workflow to execute and which data is created during its execution, then we know what to verify and validate.

In this chapter we describe in detail the *Run static analysis* and the *Run dynamic analysis* steps of the VFramework. These steps capture the original execution of the workflow and document the environment in which it took place. They are depicted in Figure 4.1, which depicts all the VFramework steps performed in the original environment. We use the workflow described in Chapter 3 to demonstrate how each of the framework steps is completed.

4.1 Run static analysis

The aim of the *Run static analysis* is to describe the workflow and its context and thus define boundaries of the analysis. This step is depicted in Figure 4.2 and consists of two sub-steps discussed in the consecutive section.

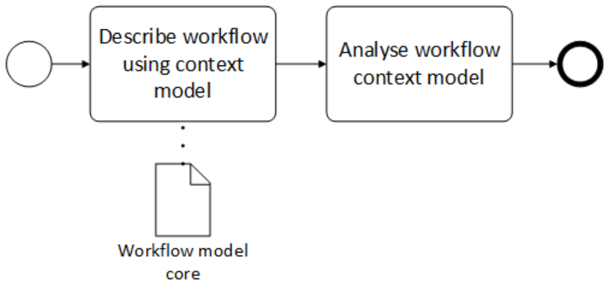

Describe workflow using context model
Analyse workflow context model
Workflow model core

Figure 4.2: The *Run static analysis* step of the VFramework in detail.

4.1.1 Describing the workflow and its context

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 (e.g. a database engine) 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 workflows 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. Furthermore, they are often provided by third parties. For this reason the analysis can be limited due to a lack of access and we can only check whether they are accessible and interact in the same way with the workflow. For example, the weather workflow uses a web service provided by a third party to obtain weather forecasts. To reproduce the result of a workflow execution, it is essential that the web service is available in an unchanged form. If we switch to any other compatible web service, the workflow performance is not affected, because the service provides only data and does not implement any processing logic. For this reason, it is an external dependency which availability and conformance must be checked, but there is no need (and no means) to verify and validate it. Hence the analysis of workflow context 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.

We use the context model to describe the workflow and its environment. We perform a static analysis using the workflow definition file. We use the Taverna2Archi[1] and the

[1] http://www.ifs.tuwien.ac.at/dp/process/projects/tavernaExtractor.html

Archi2OWL[2] converters to automatically convert the specification of a workflow into the core ontology of the context model. The first tool reads a Taverna workflow definition file and generates the core of the context model using pre-defined modelling patterns, while the second one transforms the model into an OWL ontology. For other than Taverna workflows this can be done manually using Archi[3] modelling tool, or directly in an ontology editor, for example, Protege[4].

Figure 4.3 depicts an excerpt of the ArchiMate model created for the weather workflow (for full model see Figure A.1 and for its OWL representation see Figure A.2). It models the *GetWeatherData* step. We use this example to explain how to interpret correctly the information contained in the context model. For this reason we present the intermediate stage of creating the context model, so that those who are not familiar with the Archi modelling language and its design patterns could get a better understanding of it. Normally, this process if fully automatic and does not require working with the Archi models.

Figure 4.3 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 *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

[2]http://www.ifs.tuwien.ac.at/dp/process/projects/archi2OWL.html
[3]http://www.archimatetool.com
[4]http://protege.stanford.edu

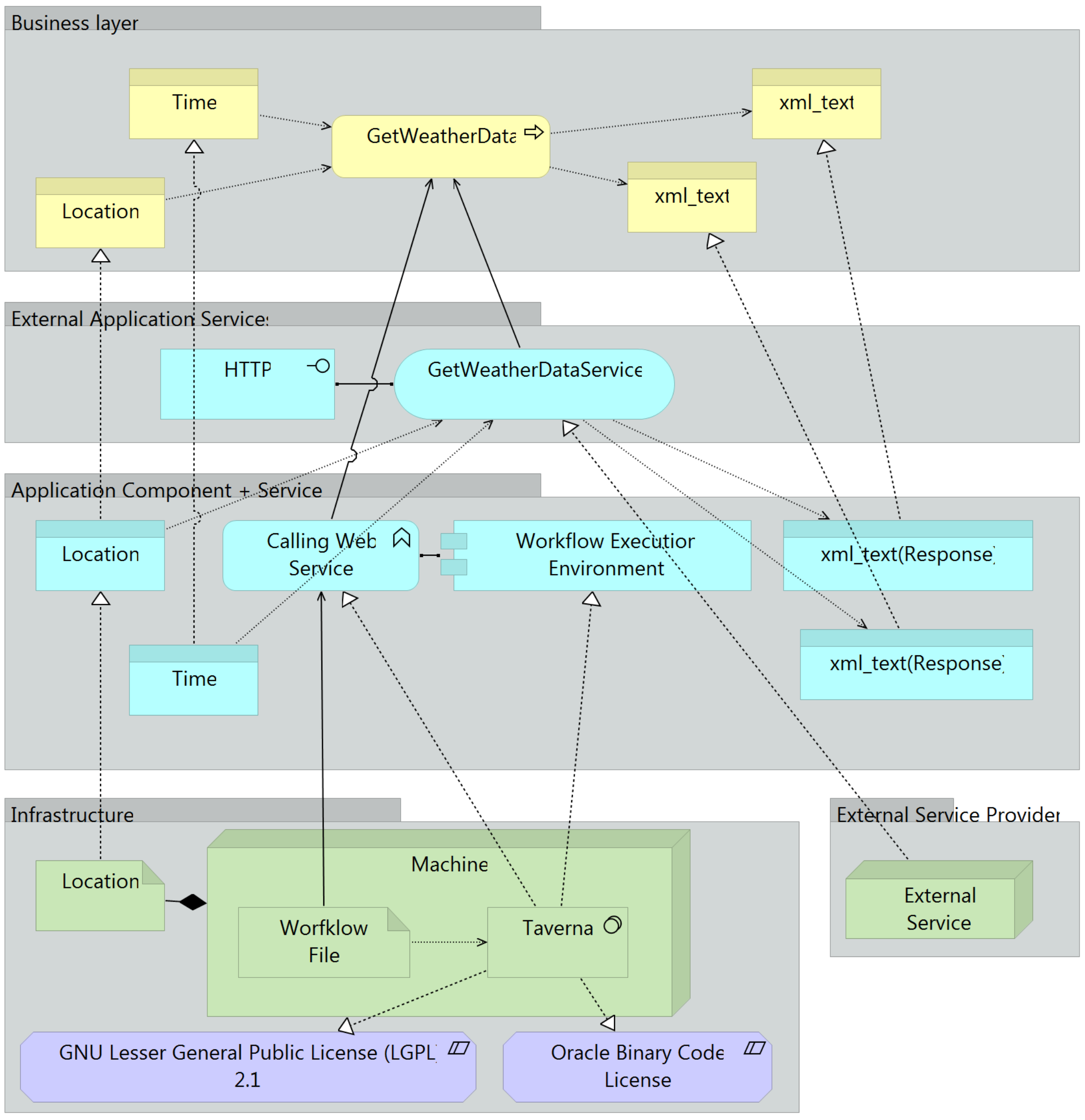


Figure 4.3: Excerpt of an ArchiMate model created for the weather workflow.

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

```
PREFIX dio: <http://timbus.teco.edu/ontologies/DIO.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

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

all their 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.

4.1.2 Workflow context analysis

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:

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

To query the model we use a set of twelve pre-defined SPARQL queries. Listing 4.1 presents one of them. The query provides an answer whether the workflow model specifies any REST web services. It looks for *Application Functions* which are called *Calling Web Service* and checks whether they are used by any *Business Process* (workflow step). Furthermore, it checks whether there is an *Application Service* that has *assignment* to an *Application Interface* that is *HTTP*, and if this *Application Service* is used by the same

Business Process. 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. The other queries enable us to:

- check if workflow uses external services,
- check if workflow uses WSDL web services,
- check if workflow uses REST web services,
- list workflow steps,
- list inputs of a workflow step,
- list outputs of a workflow step,
- list inputs of the workflow,
- list outputs of the workflow,
- list licenses applying to the workflow,
- list steps requiring additional Java libraries,
- get address and port of the external R service,
- get contents of R scripts executed by external services.

For the weather workflow model we identified that:

- One step uses a REST web service.
- One step requires a specific JAR to be placed in the workflow engine directory.

We use this information in the *Run dynamic analysis* step of the VFramework to configure the environment in which the workflow executes by placing the specific JAR file in an appropriate location. Furthermore, we know that the workflow uses a web service that is an external dependency laying beyond the workflow boundaries. For this reason, not only monitoring of the platform in which the workflow executes is necessary, but also monitoring of the network traffic to intercept the messages exchanged between the workflow and the web service (see Section 4.2.3).

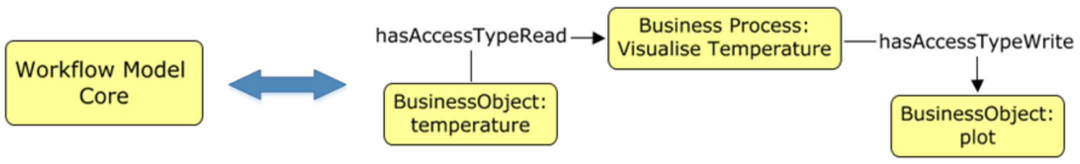


Figure 4.4: The workflow model core created for the weather workflow.

4.1.3 Step summary

We described the workflow using the context model that provides high level view on the workflow by depicting its steps, inputs and outputs, but also low level view by describing which workflow steps access external services or require additional software libraries. We also showed in what way this information is accessed using SPARQL queries and how it facilitates workflow analysis.

Based on this information we identified workflow boundaries and hence we know which of the locally completed steps will undergo dynamic analysis performed in the next framework step, and which of them are beyond our control. For these we will monitor and record the communication during workflow dynamic analysis.

Figure 4.4 depicts an excerpt of the context model created in this step for the weather workflow. It shows one workflow step *Visualise Temperature* that has one input *temperature* and one output *plot*. The model created in this step corresponds to the workflow model core (cf. Figure 3.3). We will use this example in the consecutive steps to depict how further information are integrated with the workflow model core.

4.2 Run dynamic analysis

The information collected in the first step gives overall information on the workflow execution context and is based on a static analysis of the workflow model. In this step, we complement this information by dynamic analysis of both local and external dependencies that are identifiable only during workflow execution.

We first select the test data to be used during the dynamic analysis of workflow execution and show how this kind of analysis identifies local dependencies of the workflow that lie within the workflow boundaries. Then we show how to deal with external dependencies that are services running in parallel, which only exchange data with the workflow, and are outside of the workflow boundaries. For these we describe a monitoring framework and a mock-up strategy that enables replicating the workflow and its validation. Last but not least, we present in what way automatic analysis of provenance information enriches the workflow context model. We illustrate all actions on the weather workflow that was described in Chapter 3.

Figure 4.5 depicts the *Run dynamic analysis* step of the VFramework in detail. Particular steps are described in the following sections:

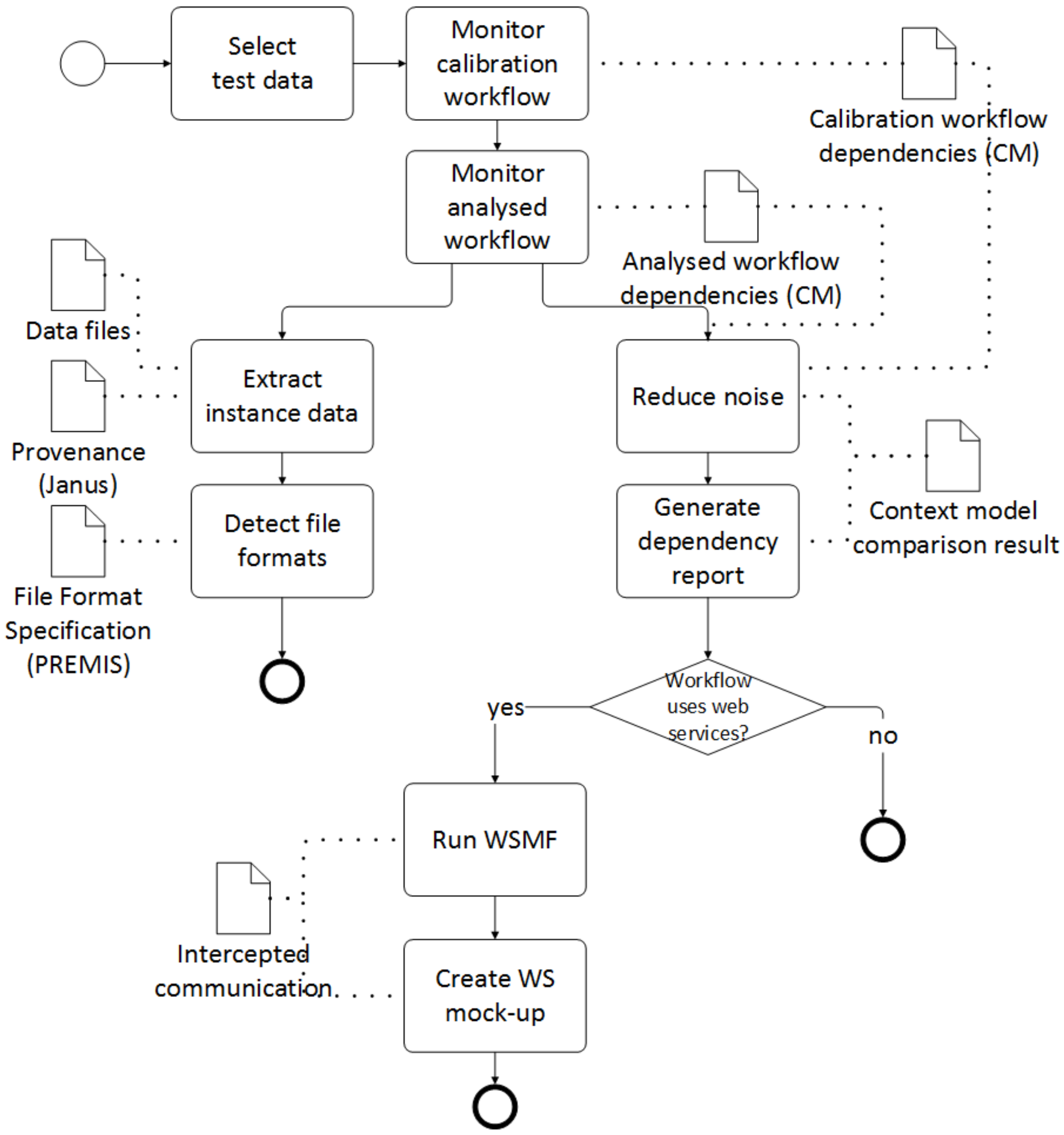

Select test data
Monitor calibration workflow
Calibration workflow dependencies (CM)
Monitor analysed workflow
Analysed workflow dependencies (CM)
Data files
Provenance (Janus)
File Format Specification (PREMIS)
Extract instance data
Detect file formats
Reduce noise
Context model comparison result
Generate dependency report
Workflow uses web services?
yes
no
Run WSMF
Intercepted communication
Create WS mock-up

Figure 4.5: The *Run dynamic analysis* step of the VFramework in detail.

- *Select test data* is described in Section 4.2.1.
- *Monitor calibration workflow*, *Monitor analysed workflow*, *Reduce noise* and *Generate dependency report* steps are described in Section 4.2.2.
- *Run WSMF* step is described in the *Web Service Monitoring Framework* sub-section of Section 4.2.3.
- *Mock web service* step is described in the *Create WS mock-up* sub-section of Section 4.2.3.
- *Extract instance data* is described in Section 4.2.4.
- *Detect file formats* is described in the *Extension of workflow model with file format information* sub-section of Section 4.2.4.

4.2.1 Test instance selection

The workflow owner must provide test instances to be used during workflow execution. Test instances are values provided to the workflow inputs. A good practice is to provide such a number of test instances so that all branches and workflow steps are executed at least once. The workflow owner must ensure that the data samples provided are in accordance with privacy and other regulation.

The weather workflow has two inputs and for the demonstration purpose we use only one set of values, thus we have one test instance. The input values are 322690 for *Location* input and 2015-10-16T18Z for *Time* input. The values of inputs will be automatically saved in a provenance trace when the workflow is executed.

4.2.2 Local dependencies

Local dependencies of the workflow are all system components used during workflow execution that are within the workflow boundary, which was defined in the previous step. 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. Furthermore, the hardware configuration of the platform also needs to be captured, especially when the workflow uses highly specific hardware to perform the computations. A good example of impact of hardware on the computational process is Graphics Processing Unit (GPU). Many different types and makes of GPUs exist, and even if the same software interface is used to perform the computation, results cannot be expected to be the same for each GPU, due to, for example, differences in precision in floating point operations. Earlier models often only supported single precision, and the accuracy of single precision models can still differ vastly in current models.

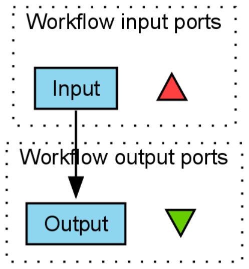


Figure 4.6: Calibration workflow for detecting workflow engine dependencies that does no processing.

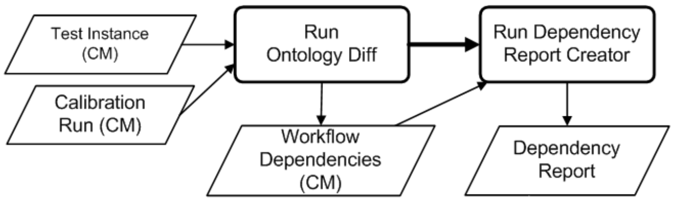


Figure 4.7: Overview of dependency reports generation.

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.

An alternative approach is the virtualisation of the complete environment in which the workflow executes. There are tools that enable migration of existing systems into virtual machines, but this approach has also drawbacks: all files, tools and services are copied including those which are not a part of the workflow. The amount of data that is captured is significantly higher than the size of the workflow, that is, the size of a virtual machine is measured in gigabytes, while the workflow and its model is measured in megabytes. Furthermore, this approach does not provide any documentation of workflow dependencies and simply hides the complexity of dependencies inside the virtual machine. This limits the analysability of workflow requirements and does not remove the dependency on external services needed to run the workflow (that were not virtualised).

Therefore, the preferred approach is to document the workflow using an automatically created workflow model that is created by monitoring workflow execution. Such a model can be used to re-create the workflow environment in a new system, because it contains technical dependencies required directly or indirectly by the workflow [Bin14]. Furthermore such a model contains explicit documentation of workflow dependencies and without re-executing the workflow we can analyse the dependencies and identify workflow requirements to configure the new environment appropriately.

We use the PMF (see Section 2.7) 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 not only a list of workflow dependencies, but also the workflow engine dependencies, for example Java libraries that are needed for Taverna to run. This is because the PMF logs every file that was accessed by the monitored process and its child processes. Figure A.3 in Appendix A presents the complete OWL ontology created by the PMF.

Due to the amount of files loaded by default by the workflow engine, it is hard to distinguish which of them are specific to a workflow and which of them are always loaded by the workflow engine. Taverna is implemented in Java and consists of multiple Java libraries that are always loaded when Taverna is started. Knowing the exact version of Taverna we can exclude all these files from the model and therefore improve its analysability. These files are still needed for the workflow to execute, but are aggregated under a higher level concept, which is Taverna and its version. Thus we can focus on the files which are necessary for running the particular workflow. If Taverna was shipped as a Debian package, then all these dependnecies would be automatically grouped under a single package by the PMF. Since Taverna is an archive that is copied into a file system we had to devise our own way of aggregating the files.

In order to filter out workflow engine specific files (so-called noise) we use a calibration workflow that allows us to discover the dependencies belonging to the workflow engine. Figure 4.6 depicts the calibration workflow. It connects only the input to the output of the workflow and makes no processing. Therefore, all dependencies identified for the calibration workflow are in fact workflow engine dependencies. We run the calibration workflow before the analysed workflow is loaded into Taverna.

The overview of noise reduction and dependency report generation is presented in Figure 4.7. Having monitored both workflow executions, we compare their context models. Thus we reduce the workflow engine noise and focus on the workflow-specific dependencies. The reduction is effective and decreases the number of identified files from one thousand to around fifty files for the weather workflow. We compare the context models using the Ontology Diff Tool[5]. We use the comparison result to generate a dependency report that provides a concise overview of workflow specific dependencies. The tool for dependency report generation uses five predefined SPARQL queries to process the data from the context model (after noise reduction).

An excerpt of a dependency report for the weather workflow is presented in Figure 4.8. 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

[5] http://www.ifs.tuwien.ac.at/dp/process/projects/diff.html

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. Sometimes more than one IP address is provided in the list for a single service. This is because its domain name is bound to more than one address. The IP address of the service is used to configure tools that intercept the traffic - for details see Section 4.2.3.

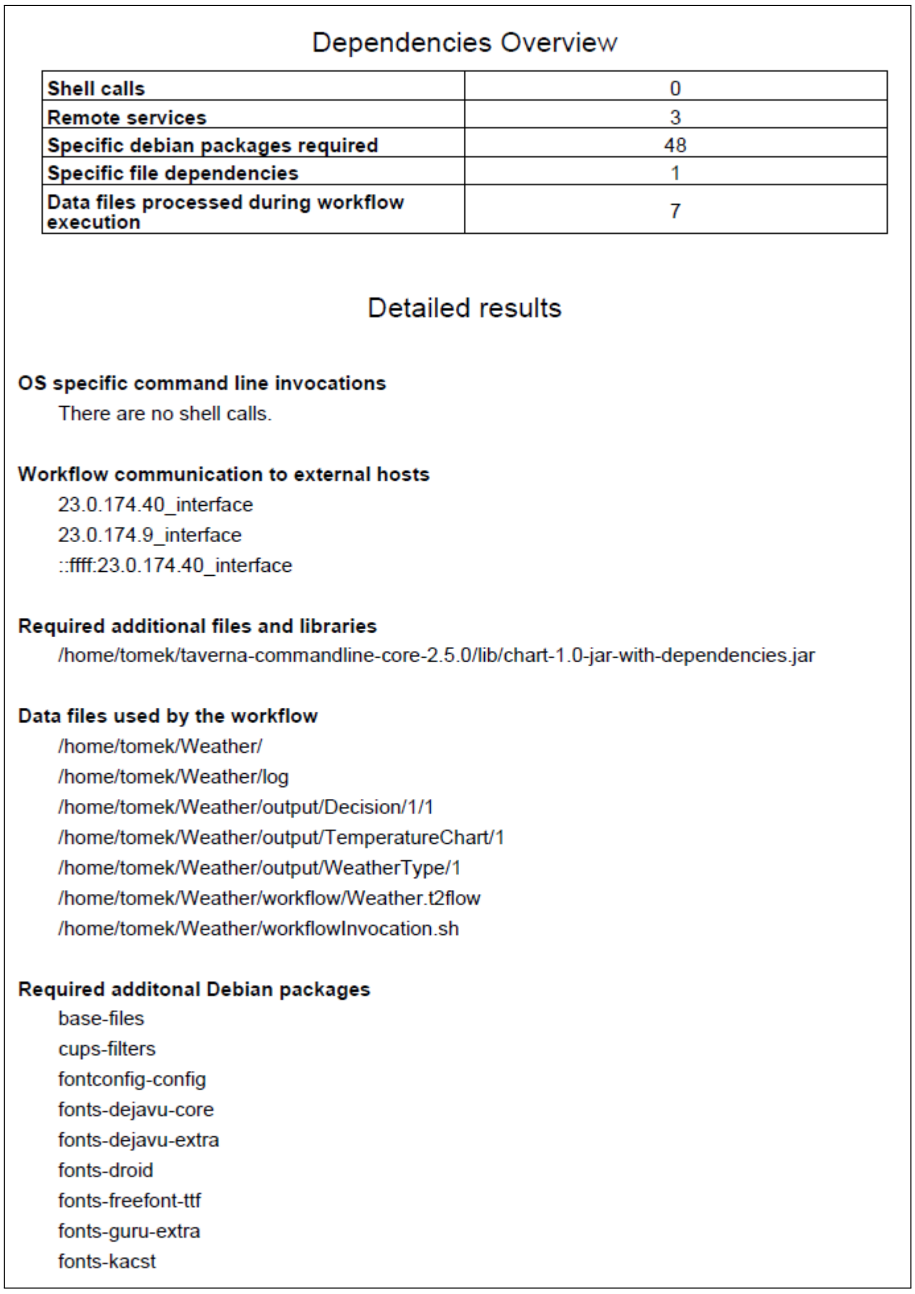
- **Specific file dependencies** These are all files that are used by a workflow but cannot be attributed to any of the installed packages. Usually, these are all sorts of configuration files and additional libraries that might have been installed manually in the system.

- **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. For details see Section 6.2.

- **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 interactions the workflow has. However, for the verification we use the complete context model which also includes the workflow engine dependencies.

We integrate the context model created in a result of dynamic analysis with the context model created by the static analysis by source level linking of persistent unique identifiers [TIM14]. 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 4.9 depicts an excerpt of integrated context models for the weather workflow (see Figure A.4 for the full model). 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.



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 4.8: Excerpt of dependency report for the weather workflow.

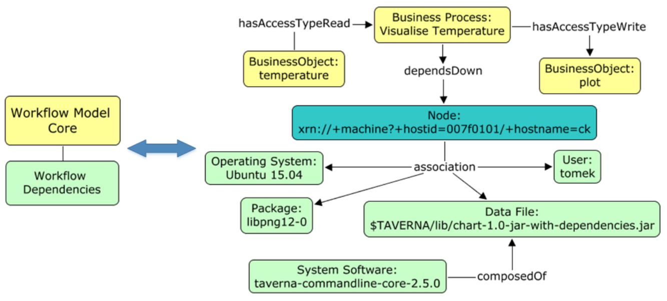

hasAccessTypeRead
Business Process: Visualise Temperature
hasAccessTypeWrite
BusinessObject: temperature
dependsDown
BusinessObject: plot
Workflow Model Core
Node: xrn://+machine?+hostid=007f0101/+hostname=ck
Workflow Dependencies
Operating System: Ubuntu 15.04
association
User: tomek
Package: libpng12-0
Data File: $TAVERNA/lib/chart-1.0-jar-with-dependencies.jar
System Software: taverna-commandline-core-2.5.0
composedOf

Figure 4.9: Excerpt of the integrated context models created by static and dynamic analysis.

4.2.3 External dependencies

The external dependencies of a workflow are all kinds of services that are used to perform workflow steps but are not under direct control of the workflow engine. These can be services that were started in parallel on a local machine or services that are hosted on a different platform than the one executing the workflow. The workflow only exchanges data with them. Web services are an example of external dependencies and according to [MR15a] they are the most common external dependency of Taverna workflows. For this reason we demonstrate how we ensure repeatable conditions during validation of workflows that use web services.

When a workflow is re-executed in a new environment, 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 may either break or can produce altered results. A summary of possible changes to a web service is presented in Table 4.1. The possible solutions to circumvent this problem are:

- To use a different service that is compliant with the original one.
- To create a mock-up of the service that can provide correct responses for the requests recorded in the original environment.

Independent of the chosen solution, we need to collect the evidence to which we can refer when either assessing compliance of a substitute service, or when creating a mock-up. In [MMR15a] we described the Web Service Monitoring Framework that describes how to collect such evidence. In [MMUR14] we described ways of creating mock-ups.

Web Service Monitoring Framework

We developed the Web Service Monitoring Framework (WSMF) for monitoring of web services which are external dependencies of workflows that undergo digital preservation actions, for example were moved to a repository. The WSMF monitors whether the

Table 4.1: Summary of changes in Web Services [MMUR14].

Change type	Description
Unavailability	Likely stops the execution of the workflow. The reasons can range from temporary technical problems, to bankruptcy of the service provider. It can be easily detected, for example by using time-outs which would alert to unavailability of the Web Service.
Interface change	Such situation may also be easily detected. It may require short pauses in the workflow execution until the changes will be adopted into the workflow. Of course, in case of significant changes in the communication interface (e.g. switch from REST to WSDL), time needed for reconnecting the Web Service into the workflow may require more effort.
Functionality change	Outputs of the Web Service change, while the interface stays the same. This threat is hard to detect, as the workflow may not break, but instead deliver outputs which are not correct. These could be, for example, changes at the semantic level, e.g. switching the unit of measurement from inches to centimetre due to a server configuration change. Other possibilities are bug fixes in the underlying algorithm (which may introduce other bugs as well), or intentional changes in the functionality, e.g. faster but less accurate computational algorithms.
Behavioural change	It does not necessarily stop the workflow from correct execution, but can change its quality of service. The behavioural changes can occur temporally and therefore are hard to notice. The examples of such cases could be different timing characteristics or delays, effects of buffering, and so on.

web services that are required for the preserved workflow to run are still available in an unchanged way. If the WSMF detects any alteration, then the preservation expert, who is responsible for the workflow preservation, is notified. The expert decides whether the detected change affects the workflow performance and acts appropriately, for example redeploys the preserved web service or finds a substitute. Thus the possibility of redeploying the workflow at any point in time is ensured.

The WSMF consists of four steps:

1. Capture
   The communication to and from the service is intercepted and stored. This is done either by capturing the network traffic between two hosts or by redirecting the communication with the web service through a special proxy server that intercepts

the data and passes on the communication to the final destination. The first solution is transparent to the workflow and does not require introducing changes to the workflow. The second solution is useful when the communication cannot be intercepted, for example when the session is encrypted. Then a proxy service can decrypt the communication, capture the requests and responses, and encrypt them back and forward to their destinations (*man-in-the-middle* attack on SSL).

2. Transform
   When the data capturing is finished, the data is transformed to a form in which requests and corresponding responses are grouped. Additional metadata and measures are added and calculated at this step, for example, a pair of request and response is enriched by number of occurrences, timestamps and calculated interval between sending the request and receiving the response.

3. Reason
   At this step the collected data is analysed, and the type of web service is determined from the data. If for the same request different responses exist, then a web service is deemed to be non-deterministic, otherwise it is deterministic, from the point of view of the observer. When it is deterministic, monitoring for changes in functionality is possible, otherwise not. Reasons for perceived non-determinism can be manifold. In many cases, it will be due to the dependence on a specific state that could, for example, be the current date and time.

4. Monitor
   Requests collected in the first step are used to query the web service. Responses collected at this step are compared to those collected in the *Capture* step. This step is replayed according to the planned schedule, for example every day or every week, etc.

A crucial requirement for using the approach described above is that the web service does not cause any changes on the world outside the system observed. In situations in which this is not the case, for example in credit card payment transaction systems, such replaying of messages for monitoring purposes cannot be employed. Thus, while not universally applicable, the approach is still useful for a majority of situations, specifically in scientific settings in which web services are deployed primarily for accessing or transforming data using computational services.

For the short term setting we use the first three steps that allow us to intercept communication with the web service and identify its type. We do not monitor the web service to detect changes, because:

- We perform verification and validation of workflow re-execution from a point of view of scientists who have no influence on changes in external web services. It is not their role to monitor the web services and they have no resources to do that.

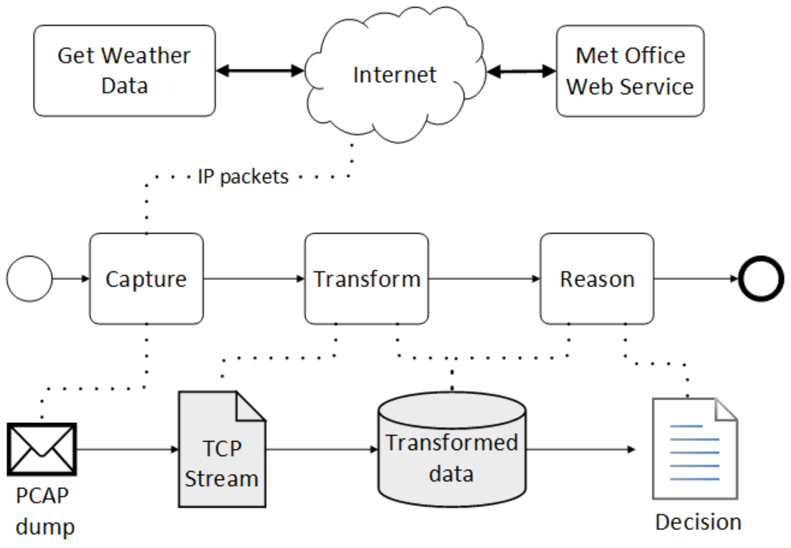


Figure 4.10: The first three steps of the WSMF applied to the weather workflow.

Hence, our task is to collect evidence from the original environment that proves how the workflow performed using the web service and to provide a possibility to replicate given workflow executions using the past data.

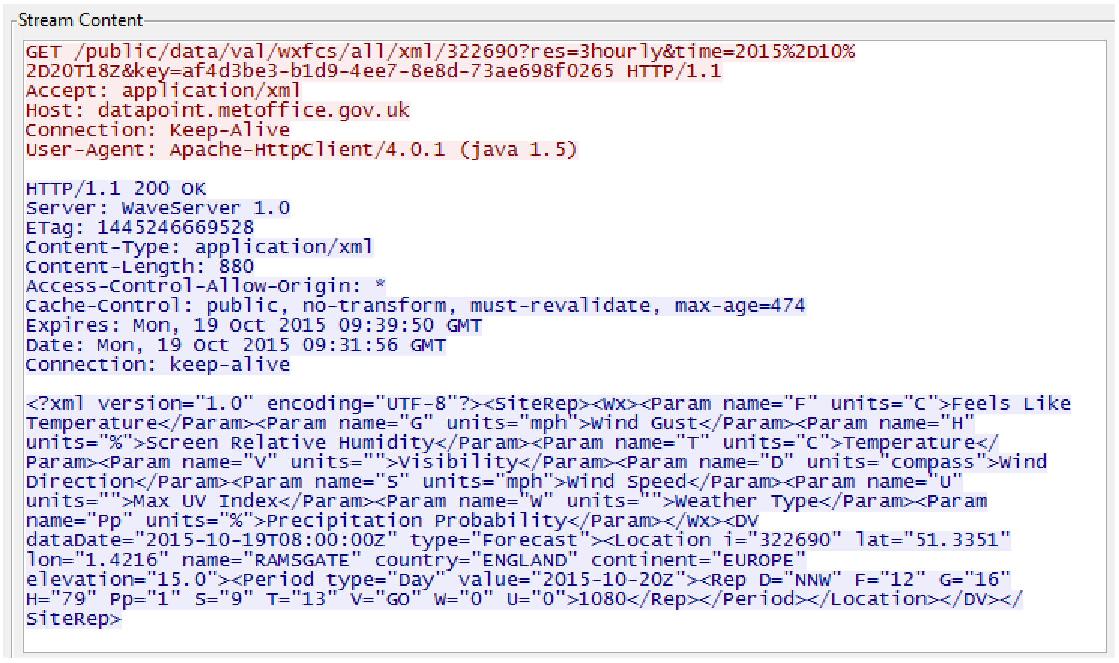
- Functionality changes in web services are detected in step *Validate workflow* of the VFramework using provenance traces.

Figure 4.10 illustrates the three steps of the WSMF applied to the weather workflow. Below we describe in detail how we complete each of them.

The dependency report (see Figure 4.8) specifies three IP addresses, but all of them point to the same service. We confirmed that by doing a reverse DNS lookup with a use of *nslookup*. To capture the network traffic during workflow execution we used Wireshark[6] which is a tool for network traffic analysis. We captured data for 24 consecutive workflow executions in the space of 1 hour each and saved them as PCAP dumps. Each workflow execution used the same input parameters. We filtered communication to the IP of the web service and recreated TCP Streams for each dump. Figure 4.11 depicts a TCP Stream recreated for the workflow execution that was also observed with PMF. It shows the requests and responses that were intercepted. We analysed the collected data, by comparing the bodies of responses (the request parameters were always the same).

We detected that the responses are different. This implies that the web service is non-deterministic and hence the results produced by the workflow depend on the date and time of running the workflow. To make the workflow executions comparable, always the same data must be provided. This can only be achieved when a web service mock-up is

[6]https://www.wireshark.org



```
GET /public/data/val/wxfcs/all/xml/322690?res=3hourly&time=2015%2D10%
2D20T18Z&key=af4d3be3-b1d9-4ee7-8e8d-73ae698f0265 HTTP/1.1
Accept: application/xml
Host: datapoint.metoffice.gov.uk
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.0.1 (java 1.5)

HTTP/1.1 200 OK
Server: WaveServer 1.0
ETag: 1445246669528
Content-Type: application/xml
Content-Length: 880
Access-Control-Allow-Origin: *
Cache-Control: public, no-transform, must-revalidate, max-age=474
Expires: Mon, 19 Oct 2015 09:39:50 GMT
Date: Mon, 19 Oct 2015 09:31:56 GMT
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?><SiteRep><Wx><Param name="F" units="C">Feels Like
Temperature</Param><Param name="G" units="mph">Wind Gust</Param><Param name="H"
units="%">Screen Relative Humidity</Param><Param name="T" units="C">Temperature</
Param><Param name="V" units="">Visibility</Param><Param name="D" units="compass">Wind
Direction</Param><Param name="S" units="mph">Wind Speed</Param><Param name="U"
units="">Max UV Index</Param><Param name="W" units="">Weather Type</Param><Param
name="Pp" units="%">Precipitation Probability</Param></Wx><DV
dataDate="2015-10-19T08:00:00Z" type="Forecast"><Location i="322690" lat="51.3351"
lon="1.4216" name="RAMSGATE" country="ENGLAND" continent="EUROPE"
elevation="15.0"><Period type="Day" value="2015-10-20Z"><Rep D="NNW" F="12" G="16"
H="79" Pp="1" S="9" T="13" V="GO" W="0" U="0">1080</Rep></Period></Location></DV></
SiteRep>
```

Figure 4.11: Intercepted TCP Stream containing a request to and a response from the web service used in the weather workflow.

used during the redeployment instead of the original web service, even if the web service is available. To confirm our verdict on non-determinism of the analysed web service, we checked the documentation available on the Met Office web sites. It states that a new weather forecast is issued every 3 hours and that the past data is not available. This explains different responses.

If the service was deterministic, that is, if it provided always the same responses, we would also store the intercepted network traffic, but would use the original web service in the redeployment environment. Only if the validation performed in the step *Validate workflow* of the VFramework would detect changes in the workflow (not necessarily due to lack of determinism, but for example due to change in the implementation), we would have to create a mock-up using the captured data.

Service Mock-up

When a web service is provided by a third party, and no access to the system hosting the service is provided, a mock-up can be created. It simulates the original web service. The most basic form of a mock-up implements a lookup table. Such a mock-up is able to send back responses to requests which were previously recorded in the original environment. As a consequence only messages which were intercepted can be replayed. The solution does not have any computing capabilities as the original system may have. However, in many cases the information collected may be sufficient to meet legal requirements or compliance regulations for documenting and proofing the correctness of workflow execution.

A mock-up can also be used as a model to validate the identity of a proposed substitute web service. The requests stored by the mock-up can be used to query the substitute, and responses received can be compared against responses of the mock-up. This approach increases the likelihood of matching a correct substitute in comparison to relying on the web service description only. A similar concept which uses provenance data to find substitutes for missing tasks in scientific workflows is described in [BGSRDR11].

We implemented the web service mock-up as an HTTP server. The server receives requests and checks whether there is a matching request in the database. If so, it sends back the matching response. The database was filled with data collected during WSMF application by splitting the recreated TCP streams of captured network traffic into requests and responses. We describe how to configure the workflow to use a mock-up in Section 6.1, in which we redeploy the workflow in a new environment.

4.2.4 Provenance 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. Others save also 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 [BZG+15] and are shared together with workflows to document the obtained results. We use the Janus [MSZ+10] provenance ontology to model workflow instance data. The data captured for any kind of workflow can be expressed using this ontology. Taverna supports exporting provenance using Janus as well.

Once the data for a workflow execution is captured we need to couple it with the workflow model to depict which data was collected for which workflow steps. This is needed because we perform validation using this data and want to know which steps are validated by comparing this data.

Last, but not least, the analysis of workflow instance data (provenance data) can be used to extend the workflow context model with further useful information. In Section 5.1 we describe in what way data must be compared during validation. The data file formats can be detected analysing the provenance traces. The data file formats have impact on the selection of an appropriate data comparison method during validation.

In the reminder of this section we describe how a workflow instance is modelled using Janus ontology. Furthermore, we show how we integrate workflow instances with the rest of the workflow model. Finally, we demonstrate in what way the workflow context model is extended with information on file formats that are automatically detected during provenance analysis.

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[7] to extract this data and export it in the Janus ontology format. We further extend it by adding more information on the input and output ports of a workflow step, for example, on enactment timing.

Figure 4.12 depicts excerpt of provenance traces for the weather workflow (Figure A.5 presents complete traces). 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.

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.

Integration of workflow instance and model

We need to establish a link between the workflow model and the workflow instance (provenance traces), 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.

Ontology mapping can also be used in ontology merging, when two ontologies are copied into a single ontology and the concepts are unified using the defined mapping. This can be useful in settings when different ontologies describing the same phenomenon are merged to create a new common vocabulary. In our setting we do not design a new

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

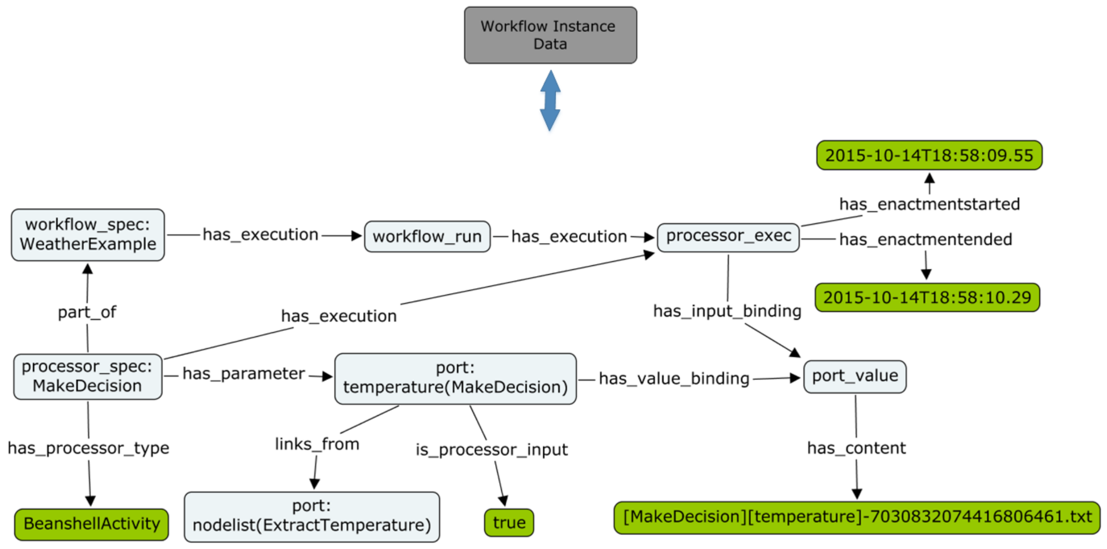


Figure 4.12: Janus ontology depicting provenance of the weather workflow for the *Make Decision* step.

ontology but reuse existing well-established ontologies for the purpose of documenting workflow execution.

The ontology integration through 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*
- *workflow_spec* is equivalent to *Business Process*
- *processor_spec* is equivalent to *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. 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 sub-processes that are particular process steps. Both the process and its steps are modelled using the *Business Process* class.

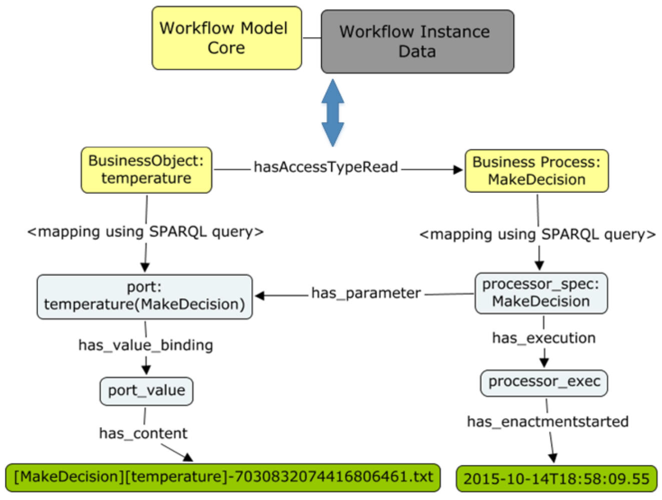
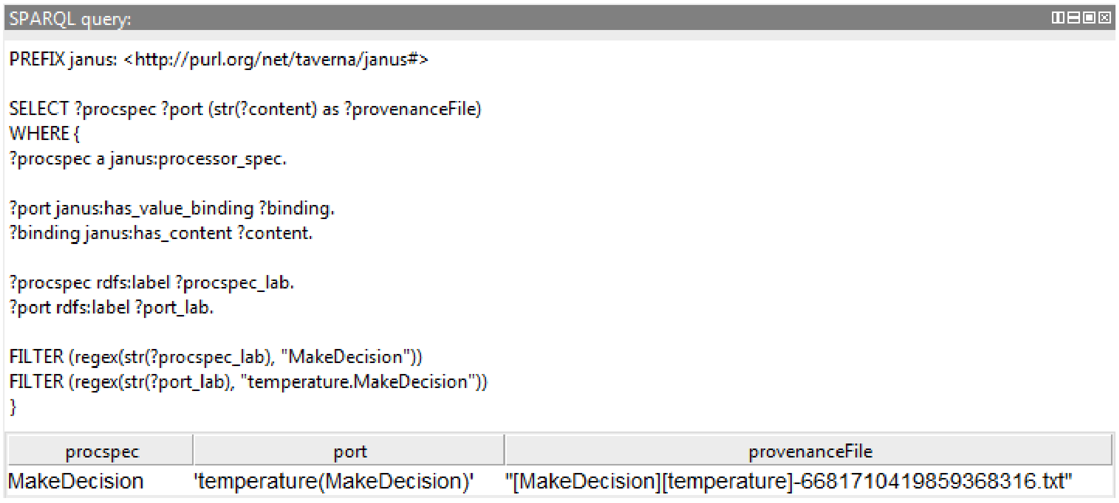

Workflow Model Core
Workflow Instance Data
BusinessObject: temperature
hasAccessTypeRead
Business Process: MakeDecision
<mapping using SPARQL query>
<mapping using SPARQL query>
port: temperature(MakeDecision)
has_parameter
processor_spec: MakeDecision
has_value_binding
has_execution
port_value
processor_exec
has_content
has_enactmentstarted
[MakeDecision][temperature]-7030832074416806461.txt
2015-10-14T18:58:09.55

Figure 4.13: Integration of the provenance traces and workflow model using SPARQL queries on the example of the weather workflow.

Figure 4.13 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 4.12. In Figure 4.13 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 4.14 depicts one of the generated queries for the weather workflow. We incorporated this mechanism into the VPlanComparator (see Section 6.2) which is a tool used for comparison of provenance traces during validation.



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

Extension of workflow model with file format information

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 the next section, this information facilitates the validation process by allowing to choose a correct data comparison tool. We obtain this information by running the DROID[8] characterisation tool that performs automated identification of file formats. It uses information from the PRONOM [Bro08] file format registry and it can identify over 250 file formats, with more formats being added continously. We use the PREMIS ontology [PRE08], 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 4.15. 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 [Gro12]. 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. For other outputs (not depicted in the Figure, for complete integration see Figure A.6) we identified following file formats:

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

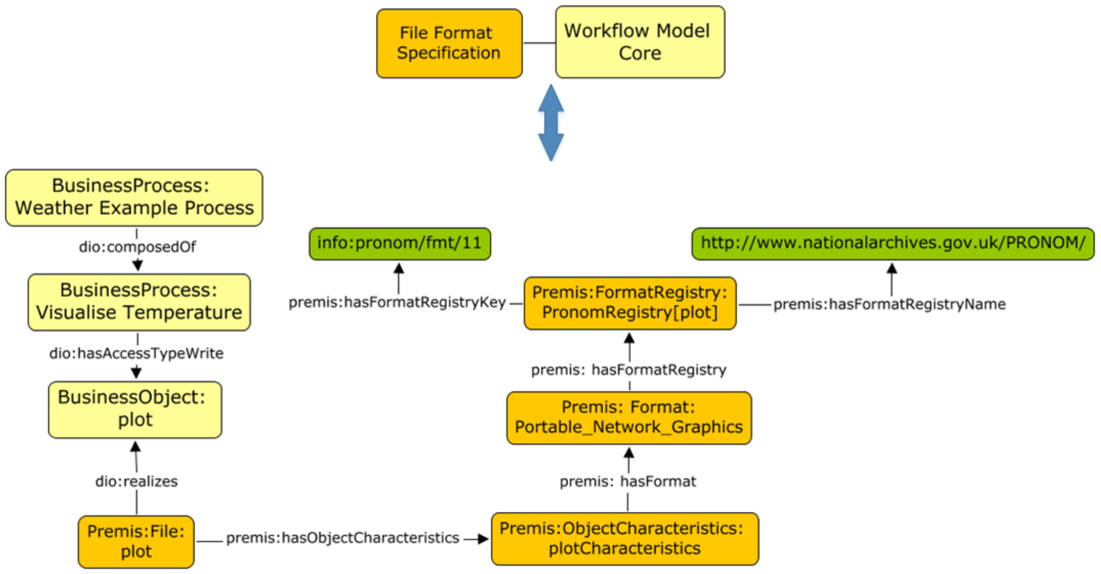

File Format Specification
Workflow Model Core
BusinessProcess: Weather Example Process
dio:composedOf
BusinessProcess: Visualise Temperature
dio:hasAccessTypeWrite
BusinessObject: plot
dio:realizes
Premis:File: plot
premis:hasObjectCharacteristics
Premis:ObjectCharacteristics: plotCharacteristics
premis: hasFormat
Premis: Format: Portable_Network_Graphics
premis: hasFormatRegistry
Premis:FormatRegistry: PronomRegistry[plot]
premis:hasFormatRegistryKey
info:pronom/fmt/11
premis:hasFormatRegistryName
http://www.nationalarchives.gov.uk/PRONOM/

Figure 4.15: Extension of the context model with information on file formats using PREMIS ontology for the weather workflow.

- XML (*pronom/fmt/101*) for *xml_text* outputs of steps *ExtractTemperature* and *ExtractWeatherType*
- PNG (*pronom/fmt/11*) for the *plot* output of the *Visualise Temperature* step
- Plain Text (*pronom/x-fmt/111*) for all other outputs

4.2.5 Step summary

We performed dynamic analysis of dependencies that are identifiable only during workflow execution. We used the Process Migration Framework (see Section 2.7) to monitor workflow execution and identify files, Debian packages and services used during computation. These local dependencies were used to extend the workflow model created in the previous step.

Furthermore, we analysed possible changes in web services that are external dependencies of workflows. We described how to monitor workflow interactions with such services and proposed creating a mock-up service that is based on an intercepted traffic, to ensure repeatable conditions during workflow re-execution. For that purpose we used Web Service Monitoring Framework.

Finally, we described how we capture and model the actual data processed by the workflow. We showed how using SPARQL queries the provenance traces are integrated with the workflow model. We also demonstrated in what way the workflow context model

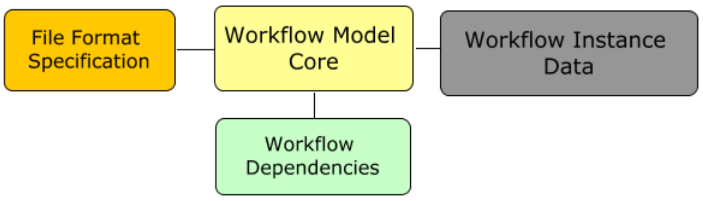


Figure 4.16: Overview of the context model parts instantiated in the result of completing the *Run dynamic analysis* step of the VFramework.

is extended with information on file formats that are automatically detected during provenance analysis.

Figure 4.16 depicts the state of the workflow context model after completing the *Run dynamic analysis* step of the VFramework. Three parts were added to the *workflow model core*, which was created in the previous step, these parts are: *workflow dependencies*, *workflow instance data*, and *file format specification* (see Figure A.7 for details). Compared to the Figure 3.3, which depicts all parts of the context model instantiated during the VFramework application, only the *validation requirements* part is missing. We add this part in the *Define validation metrics* step of the Framework that is described in Chapter 5.

The outcomes of this step are:

- Workflow context model extended with information on infrastructure components needed to execute the workflow and also with file formats of each workflow output
- Dependency report summarizing workflow dependencies in a human readable form
- Dump of intercepted communication between the workflow and external services
- Provenance traces holding the actual values of executed workflow instances
- Copy of data files accessed by the workflow during execution

4.3 Summary

In this chapter we described in detail the *Run static analysis* and the *Run dynamic analysis* steps of the VFramework which is a framework for verification and validation of workflow re-executions.

Besides a detailed description of the VFramework steps we also presented tools that can be used to foster comprehensive workflow context description, automatic data collection and comparison. We based our solution on a context model which is an extensible ontology allowing for description of workflow and its dependencies.

Furthermore we showed how the automated context model analysis enables identification of workflow boundaries and in what way the external services being outside these boundaries can be monitored to create evidence needed for re-executing and validating workflows.

In the next chapter we present the VPlan which is an ontology for description of validation requirements.

CHAPTER 5

Validation requirements

In this chapter we discuss what influences specification of validation requirements, how they can be quantified and which data can be used for this purpose.

In Section 5.1 we present the *Define validation metrics* step of the VFramework and discuss what influences the specification of validation requirements and in what way we can express them. In Section 5.2 we provide a detailed specification of the VPlan ontology that we developed for extending the workflow context model with validation requirements and metrics.

5.1 Define validation metrics

The aim of this step is to define requirements and link them to quantifiable metrics used for validation of workflow re-executions. Thus we need to identify what has to be measured and in what way compared to check the correctness of a workflow re-execution.

5.1.1 Requirements

To define a universal set of requirements that can be used for validation of workflow re-executions we performed two kinds of analysis:

- Top-down analysis taking into account the scientific method.
- 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 [Gau12]. 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

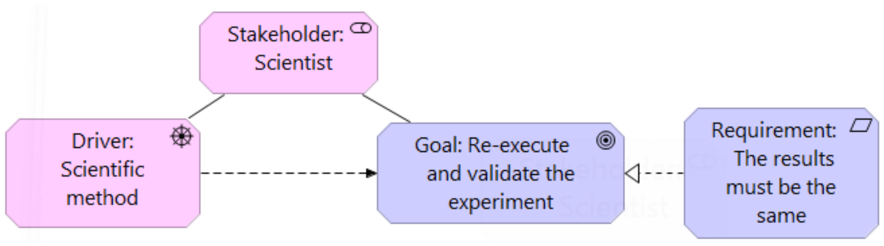

Stakeholder: Scientist
Driver: Scientific method
Goal: Re-execute and validate the experiment
Requirement: The results must be the same

Figure 5.1: Top down analysis of validation requirements taking into account the scientific method.

Figure 5.1 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 [Bra97] standard for expressing the requirements.

To find out how to apply the above requirement to workflows we analysed the documentation of Taverna, in which we read that: “Each component is only responsible for a small fragment of functionality, therefore many components need to be chained in a pipeline in order to obtain a workflow that can perform a useful task.”[1]

Hence we know that the finest granularity of available data that can be captured in the workflow system is its single “component”, that is, a workflow step. 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
- One workflow from the music information retrieval domain
- One workflow from the sensor data analysis domain
- Two workflows from the medical research domain

For each of the workflows we brainstormed to define requirements that validate whether the workflow re-execution 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.

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

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.

We also noticed that in some workflows not all of the step outputs are connected to other steps. This is because some of the default components provide the same data in different formats and the workflow owner decides which one to use. For example, the *XPathService* that is used in the weather workflow has four outputs, but only from two of them the data is used for further processing (cf. Section 3.2). Due to the fact that the data for these unused outputs is also stored in the provenance traces and taking into account that a workflow or its parts can be reused and then such outputs can be used, we believe it is safer to validate all outputs during workflow re-execution.

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 produced 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.

5.1.2 Metrics

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. In the context of this work a metric can be defined as:

- *A quantitative measure of the degree to which a system, component, or process possesses a given attribute* [IEE90].
- *A calculated or composite indicator based upon two or more measures. A quantified measure of the degree to which a system, component, or process possesses a given attribute* [Pro11].
- In mathematics, a metric or distance function is a function that defines a distance between each pair of elements of a set [DD06].

Based on these definitions 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 in how far two PNG images differ.

However, providing just a metric is not sufficient, because we do not know which value of metric corresponds to 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. In [GR12] authors introduced a notion of levels of comparison. In Figure 5.2 we can see different forms of a digital object in a system. For example, the 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.

We document the validation requirements using the VPlan ontology. Section 5.2 provides details on the VPlan structure, metrics vocabulary, as well as the integration with the workflow model. We also describe there how we generate requirements and metrics used for validation of workflow re-executions.

5.1.3 Step summary

We analysed goals and drivers of scientits for validating workflow re-executions, as well as investigated a sample of Taverna workflows. We came to the following conclusions:

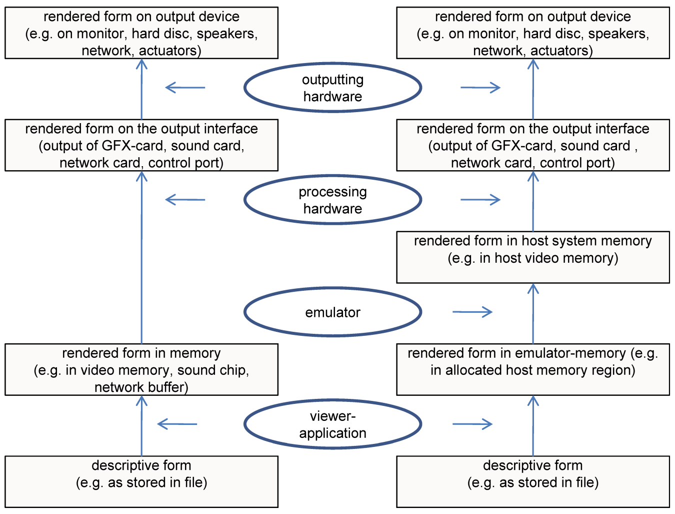

rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
outputting hardware
rendered form on the output interface (output of GFX-card, sound card, network card, control port)
rendered form on the output interface (output of GFX-card, sound card , network card, control port)
processing hardware
rendered form in host system memory (e.g. in host video memory)
emulator
rendered form in memory (e.g. in video memory, sound chip, network buffer)
rendered form in emulator-memory (e.g. in allocated host memory region)
viewer-application
descriptive form (e.g. as stored in file)
descriptive form (e.g. as stored in file)

Figure 5.2: Different forms of a digital object in a system's memory. On the left the layers in an original system are shown, on the right the layers in a system hosting a virtualized view-path are shown [GR12].

- Each workflow step must be validated and this should be done by validating outputs of each workflow step.
- 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.

Based on these conclusions we automatically generate requirements, metrics and their target values. The generated requirements validate whether the workflow re-execution is identical. The workflow owner may adapt these requirements, either by removing or adding some, or by relaxing conditions on target values, as well as by choosing different metrics. Thus also similar workflow executions can be validated.

5.2 VPlan

In this section we present the VPlan ontology that we created for describing validation requirements. In Section 5.1 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. Parts of this work were published in [MVBR14].

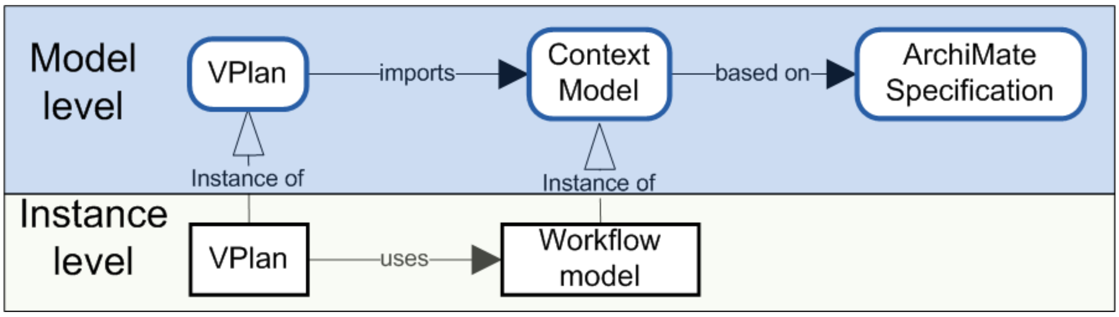


Figure 5.3: Relations between the models and instances of the context model and the VPlan.

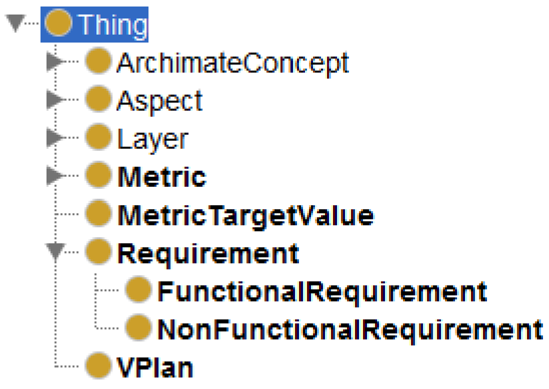


Figure 5.4: Overview of VPlan classes. Subclasses of the *Metric* are depicted in Figure 5.11.

5.2.1 Overview

According to the IEEE 1012 standard [IEE05] 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 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 5.3 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 5.4. 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 5.11, described in Section 5.2.3.

- 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 5.2.4. In the remainder of this section whenever we refer to the VPlan we mean the instance, if not stated otherwise.

5.2.2 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 5.5 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.

Figure 5.6 extends the previous figure with information on *Metrics*. 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. Figure 5.7 extends the Figure 5.6 with information on *Metric Target Value*. 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.

5.2.3 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 5.8. In the remainder of this section we explain in what way we formulate the metrics and how we derived them for each category.

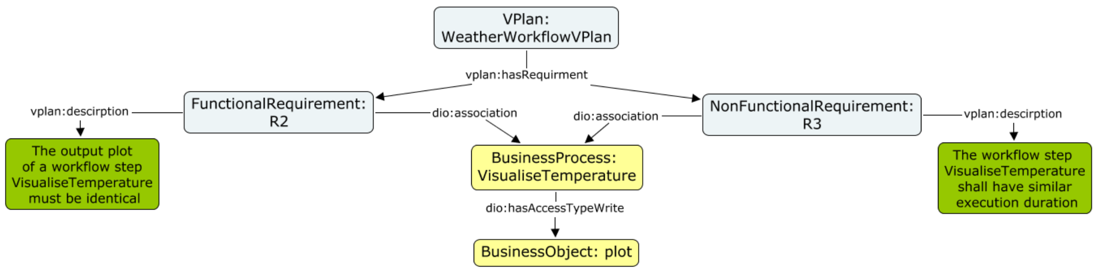

VPlan: WeatherWorkflowVPlan
vplan:hasRequirment
FunctionalRequirement: R2
NonFunctionalRequirement: R3
vplan:descirption
dio:association
dio:association
vplan:descirption
The output plot of a workflow step VisualiseTemperature must be identical
BusinessProcess: VisualiseTemperature
The workflow step VisualiseTemperature shall have similar execution duration
dio:hasAccessTypeWrite
BusinessObject: plot

Figure 5.5: VPlan instance depicting *Requirements* and their relation to the workflow model for the *Visualise Temperature* step.

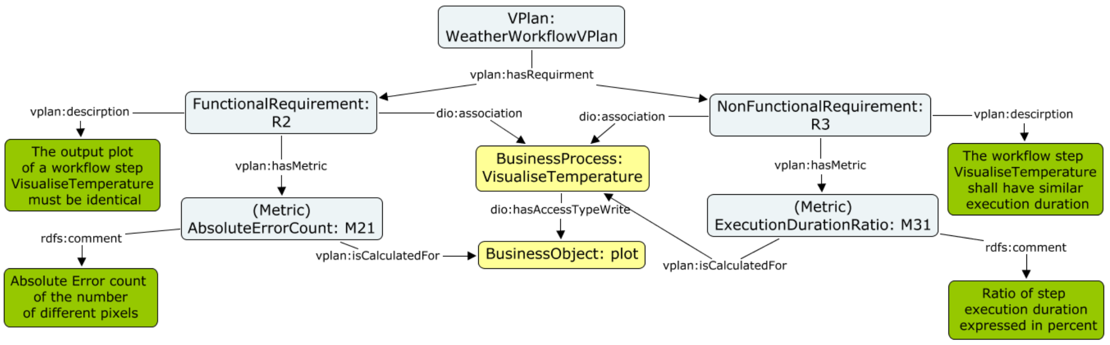

VPlan: WeatherWorkflowVPlan
vplan:hasRequirment
FunctionalRequirement: R2
NonFunctionalRequirement: R3
vplan:descirption
dio:association
dio:association
vplan:descirption
The output plot of a workflow step VisualiseTemperature must be identical
vplan:hasMetric
BusinessProcess: VisualiseTemperature
vplan:hasMetric
The workflow step VisualiseTemperature shall have similar execution duration
(Metric) AbsoluteErrorCount: M21
dio:hasAccessTypeWrite
(Metric) ExecutionDurationRatio: M31
rdfs:comment
vplan:isCalculatedFor
BusinessObject: plot
vplan:isCalculatedFor
rdfs:comment
Absolute Error count of the number of different pixels
Ratio of step execution duration expressed in percent

Figure 5.6: VPlan instance depicting *Requirements* and *Metrics* for the *Visualise Temperature* step. Measurement points are depicted using *isCalculatedFor* relation.

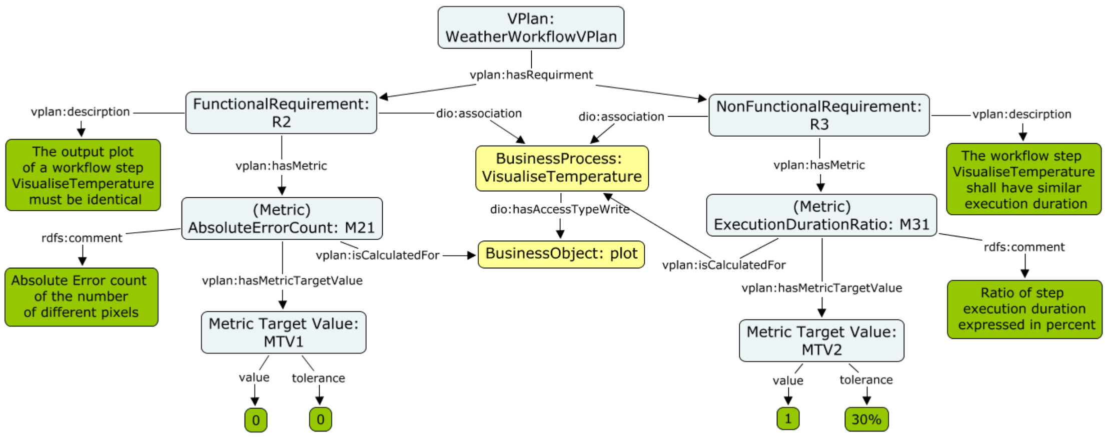

VPlan: WeatherWorkflowVPlan
vplan:hasRequirment
FunctionalRequirement: R2
NonFunctionalRequirement: R3
vplan:descirption
dio:association
dio:association
vplan:descirption
The output plot of a workflow step VisualiseTemperature must be identical
vplan:hasMetric
BusinessProcess: VisualiseTemperature
vplan:hasMetric
The workflow step VisualiseTemperature shall have similar execution duration
(Metric) AbsoluteErrorCount: M21
dio:hasAccessTypeWrite
(Metric) ExecutionDurationRatio: M31
rdfs:comment
vplan:isCalculatedFor
BusinessObject: plot
vplan:isCalculatedFor
rdfs:comment
Absolute Error count of the number of different pixels
vplan:hasMetricTargetValue
vplan:hasMetricTargetValue
Ratio of step execution duration expressed in percent
Metric Target Value: MTV1
Metric Target Value: MTV2
value
tolerance
value
tolerance
0
0
1
30%

Figure 5.7: VPlan instance depicting *Requirements*, *Metrics* and corresponding *Metric Target Values* for the *Visualise Temperature* step.

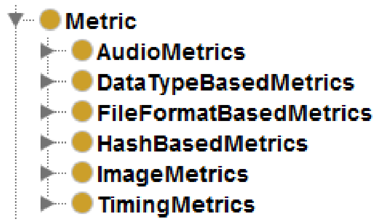


Figure 5.8: Categories of metrics defined in the VPlan.

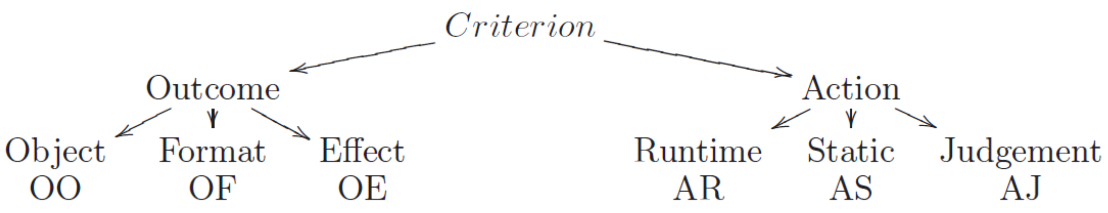


Figure 5.9: Taxonomy of criteria in digital preservation [Bec10].

In the metric specification process we aim from the very beginning on automation of the validation process and hence we focus on existing tools supporting comparison of specific file formats. We also analyse related domains to identify in what way the problem of expressing validation metrics is addressed.

A comprehensive study that presents the preservation planning process and discusses in what way the effects of applying digital preservation actions can be evaluated is presented in [Bec10]. The author created a taxonomy of criteria that are used when evaluating a preservation action. Figure 5.9 depicts this taxonomy.

Although in our setting we do not perform preservation actions, the *Outcome Object* category applies to validation of workflow re-executions, because it deals with validation of properties of digital objects. For requirements belonging to this category and stating that the results of both computations must be the same the authors suggest the following way of computing metrics:

> "For significant properties that have to be kept intact, the base measures taken on the outcome of the preservation action have to be compared to the base measures obtained from the original object. For example, the criterion Textual content unchanged is measured by analysing the original object and the outcome of the preservation action and comparing these for textual quality to get a derived measure on a Boolean scale. We thus obtain this measure by comparing the text content of the original object to the text content of the action result." [Bec10]

This aligns with our approach of breaking down requirements into quantifiable metrics that are computed using the captured data for both workflow executions. Furthermore, the author uses discrete metrics to quantify requirements expressing identity of objects. 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*.

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 5.1 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 5.1 provides an overview of file formats, analysed software tools and metrics. Further file formats can be added to the VPlan.

Table 5.1: Overview of file format based metrics that are modelled in the VPlan.

File Format	Tool support	Metrics
HTML	*Daisy Diff*[2] 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*[3] provides helpers to validate the document against an XML Schema, and compare XML documents against expected outcomes[4].	*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*[5] can be used for this purpose. *Mp3agic*[6] 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*[7] 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)

[2]https://code.google.com/p/daisydiff/
[3]http://www.xmlunit.org
[4]http://xmlunit.sourceforge.net/userguide/html/ar01s03.html
[5]http://www.ifs.tuwien.ac.at/mir/muscle/del/audio_extraction_tools.html#RPextract
[6] https://github.com/mpatric/mp3agic
[7]http://ctan.mirrorcatalogs.com/support/latexdiff/doc/latexdiff-man.pdf

PDF	*DiffPdf*[8] 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*[9] and compared using *Diff-Patch-Match* [10].	*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*[11] 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 .

[8]https://github.com/vslavik/diff-pdf
[9]https://tika.apache.org
[10]https://code.google.com/p/google-diff-match-patch/
[11]www.imagemagick.org/Usage/compare/

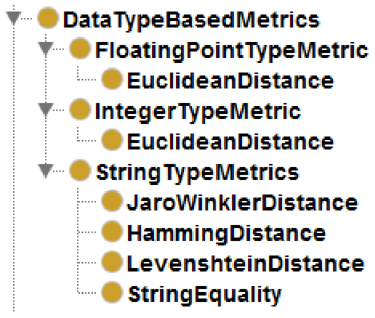


Figure 5.10: Categorization of data type based metrics in the VPlan.

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 [CRF03] authors analysed metrics that can be used for computation of distance between strings, while authors of [DD06] 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 5.10.

To demonstrate the impact of data type on choosing 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. The results are often formatted in a tool specific format that is identified by DROID as a *Plain Text* format. For example, the Music Classification workflow that we describe in Section 7.1.1 uses the Weka machine learning software[12]. The results are formatted using ARFF format that "is an ASCII text file that describes a list of instances sharing a set of attributes (. . .) ARFF files have two distinct sections. The first section is the Header information, which is followed by the Data information." [13].

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[14] 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 identic 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 [DD06] 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 5.11). If a new

[12]http://www.cs.waikato.ac.nz/ml/weka/
[13]http://www.cs.waikato.ac.nz/ml/weka/arff.html
[14]https://software.intel.com/en-us/articles/intel-sha-extensions

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 [ISO10] 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.*

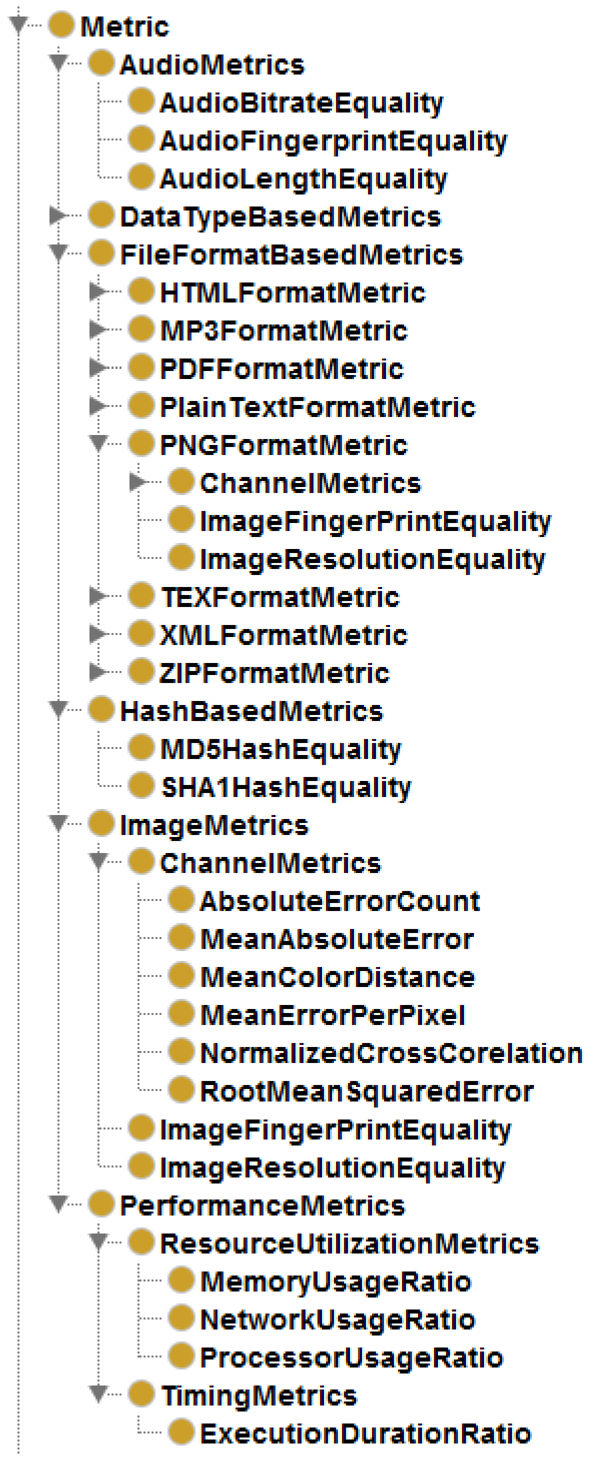
2. Resource utilization - *degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements.*

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 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 [RR05]: *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.



- Metric
  - AudioMetrics
    - AudioBitrateEquality
    - AudioFingerprintEquality
    - AudioLengthEquality
  - DataTypeBasedMetrics
  - FileFormatBasedMetrics
    - HTMLFormatMetric
    - MP3FormatMetric
    - PDFFormatMetric
    - PlainTextFormatMetric
    - PNGFormatMetric
      - ChannelMetrics
      - ImageFingerPrintEquality
      - ImageResolutionEquality
    - TEXFormatMetric
    - XMLFormatMetric
    - ZIPFormatMetric
  - HashBasedMetrics
    - MD5HashEquality
    - SHA1HashEquality
  - ImageMetrics
    - ChannelMetrics
      - AbsoluteErrorCount
      - MeanAbsoluteError
      - MeanColorDistance
      - MeanErrorPerPixel
      - NormalizedCrossCorelation
      - RootMeanSquaredError
    - ImageFingerPrintEquality
    - ImageResolutionEquality
  - PerformanceMetrics
    - ResourceUtilizationMetrics
      - MemoryUsageRatio
      - NetworkUsageRatio
      - ProcessorUsageRatio
    - TimingMetrics
      - ExecutionDurationRatio

Figure 5.11: Detailed overview of metric categories in the VPlan.

5.2.4 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 should be confirmed. In this section we describe this process and explain in what way the VPlan integrates with the workflow model.

The VPlan is generated in the *Define validation metrics* step of the VFramework (see Section 5.1). 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 type for outputs that have *Plain Text* format.

Figure 5.12 depicts an excerpt of the VPlan for the weather workflow. In comparison to Figure 5.7, Figure 5.12 presents additional information on a file format detected for a workflow step output. In the remainder of this section we use this figure to illustrate how we generated the VPlan for the weather workflow. We begin with presenting a list of steps performed to generate the VPlan and then we discuss each of them.

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.
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 list its outputs using SPARQL queries that find *Business Processes* that are connected using *hasAccessTypeWrite* relation with *Business Objects*. In the analysed example we focus on the workflow step *Visualise Temperature* that is depicted in the centre of Figure 5.12.

This step has one output for which we automatically define one *Functional Requirement*. For that purpose we use a template that expresses the *Functional Requirements* in the

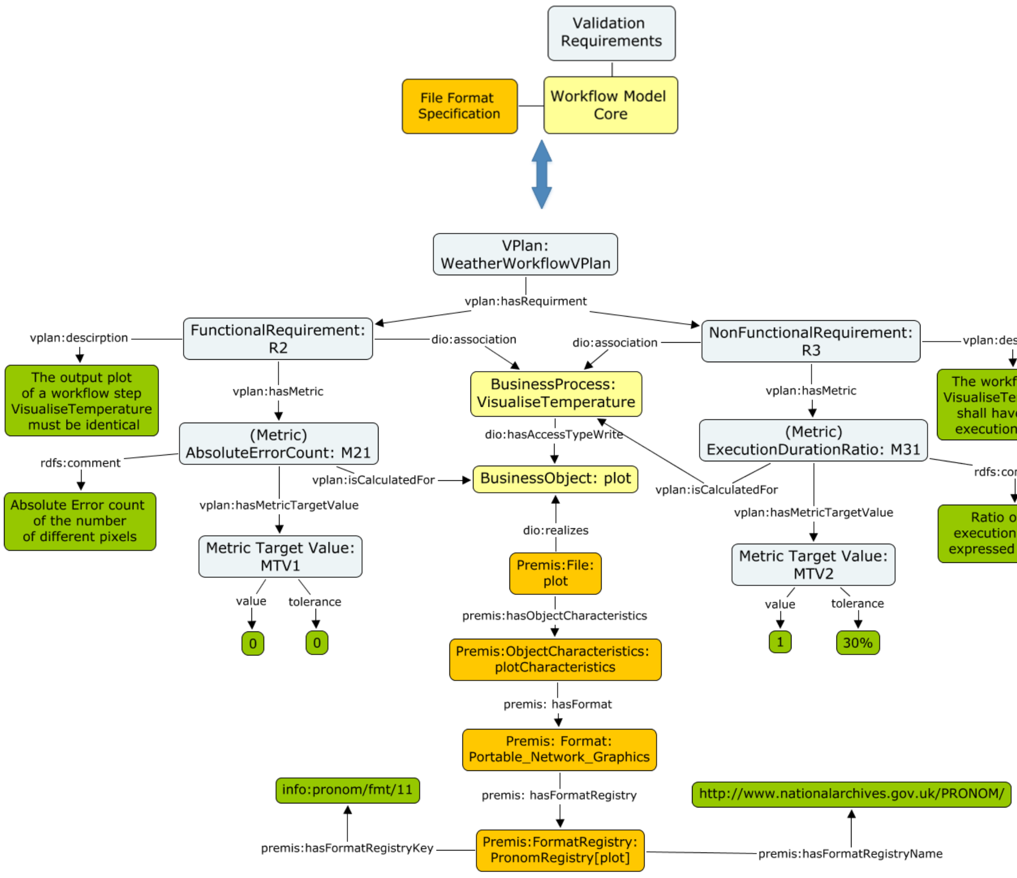

Validation Requirements
File Format Specification
Workflow Model Core
VPlan: WeatherWorkflowVPlan
vplan:hasRequirment
vplan:descirption
FunctionalRequirement: R2
dio:association
dio:association
NonFunctionalRequirement: R3
vplan:descirption
The output plot of a workflow step VisualiseTemperature must be identical
vplan:hasMetric
BusinessProcess: VisualiseTemperature
vplan:hasMetric
The workflow step VisualiseTemperature shall have similar execution duration
rdfs:comment
(Metric) AbsoluteErrorCount: M21
dio:hasAccessTypeWrite
(Metric) ExecutionDurationRatio: M31
rdfs:comment
Absolute Error count of the number of different pixels
vplan:isCalculatedFor
BusinessObject: plot
vplan:isCalculatedFor
vplan:hasMetricTargetValue
vplan:hasMetricTargetValue
Ratio of step execution duration expressed in percent
dio:realizes
Metric Target Value: MTV1
Premis:File: plot
Metric Target Value: MTV2
value
tolerance
value
tolerance
premis:hasObjectCharacteristics
0
0
1
30%
Premis:ObjectCharacteristics: plotCharacteristics
premis: hasFormat
Premis: Format: Portable_Network_Graphics
info:pronom/fmt/11
premis: hasFormatRegistry
http://www.nationalarchives.gov.uk/PRONOM/
premis:hasFormatRegistryKey
Premis:FormatRegistry: PronomRegistry[plot]
premis:hasFormatRegistryName

Figure 5.12: VPlan

following way: "The output <output name> of the workflow step <step name> must be identical". We substitute the *<output name>* and *<step name>* variables with *plot* and *Visualise Temperature* names respectively.

Each requirement has a label that consists of fixed letter ‘R’ and an integer number that is incremented each time before a new requirement is created. In the given example, the label of the *Functional Requirement* is *R2*. We use the labels for our convenience only and they have no impact on the ontology integration. For other VPlan classes we create labels by analogy.

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 5.11) defined in the VPlan model to choose the right metrics.

- If the identified file format is *Plain Text*:
  - Analyse provenance data to identify the data type.

We read data from the provenance traces of the original execution and try parsing it into Integer, Double or String type in Java. For example, the *temperature* input of the *Visualise Temperature* step contains value 12 that can be parsed to all of these types. However, for the numeric formats the tool discards String as a viable option and selects Integer, because it is a subset of Double.

  - Select *Data Type Based Metric* using the identified data type. If the data type was not identified, then use the *Hash Based Metric*.

    Continuing the example from the point above we select the *Euclidean Distance* for the *Integer Type Metric*.

- If the identified file format is one of those for which metrics were defined in the VPlan and is different from *Plain Text*:

  - Find *Metric* category within the *File Format Based Metrics* that corresponds to the identified format.

    For the identified PNG format we found the *PNG Format Metric* category.

  - Select discrete metrics that are direct subclasses of the selected category and which have no further subclasses. All discrete metrics in the VPlan have the suffix "Equality".

    The *PNG Format Metric* contains *Image Fingerprint Equality* and *Image Resolution Equality* metrics that are its direct subclasses and have no further sub classes.

  - For subclasses that have further subclasses select all their discrete metrics, or choose one metric at random if no discrete metrics are present.

    The *Channel Metric* is a subclass of the *PNG Format Metric*, but contains six further subclasses. In the design of the VPlan we assumed that metrics within the same category can be used interchangeably. For this reason we select at random one of the *Channel Metrics*, because there are no discrete metrics. We select the *Absolute Error Count*.

- Else:

  - Select one of the *Hash Based Metrics*.

To recap 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 5.12 (for full model see Figure A.8).

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 added 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 substitute 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 arbitrary. 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.

The requirements are generated in the same way for the data files identified by the PMF, because all these data files are assigned to a *Business Process* representing the workflow, that is a parent node for all the workflow steps, which are also modelled as *Business Processes*.

5.2.5 Summary

In this section we presented the VPlan that is an ontology for description of validation requirements. We described its model by specifying classes and relations used to connect them. The VPlan model reuses relations from the context model to reduce the semantic complexity by reusing well-defined concepts. The VPlan contains also a comprehensive vocabulary of metrics that are used for breaking down validation requirements. These metrics are later referenced by the VPlan instances to quantify requirements for the analysed workflows.

The controlled vocabulary of metrics groups the metrics into categories taking into account the data format identified for the captured data, as well as data type. Furthermore, it groups metrics into generic categories like Audio or Music, so that these metrics can be linked to new formats added to the vocabulary. Thus we evade double definitions of the same concepts and ensure coherence of the vocabulary. The metrics were derived based on a literature review and an analysis of sample workflows. In the metric specification process we aimed from the very beginning on the automation of the validation process and hence we focused on existing tools supporting comparison of specific file formats that are later used to compute these metrics.

We also demonstrated in what way the VPlan instance is automatically generated to validate workflows identity. For each workflow step we generated requirements,

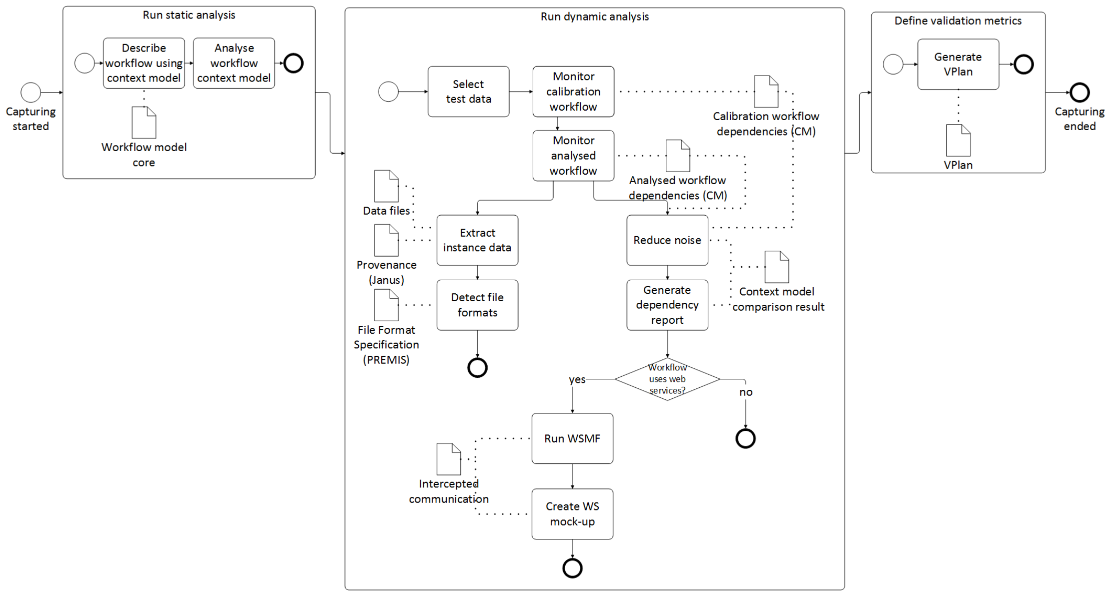

Run static analysis
Describe workflow using context model
Analyse workflow context model
Workflow model core
Capturing started
Run dynamic analysis
Select test data
Monitor calibration workflow
Calibration workflow dependencies (CM)
Monitor analysed workflow
Analysed workflow dependencies (CM)
Data files
Provenance (Janus)
File Format Specification (PREMIS)
Extract instance data
Detect file formats
Reduce noise
Context model comparison result
Generate dependency report
Workflow uses web services?
yes
no
Run WSMF
Intercepted communication
Create WS mock-up
Define validation metrics
Generate VPlan
VPlan
Capturing ended

Figure 5.13: The VFramework steps performed in the original environment - a detailed view.

metrics used to quantify these requirements and metric target values to specify when the requirement is fulfilled. The generated VPlan instance integrates with the workflow model not only by linking its requirements to particular workflow elements for which the requirements were specified, but also by depicting in which parts of the workflow the data for metrics computation must be captured.

5.3 Summary

In this chapter we analysed goals and drivers of scientits for validating workflow re-executions, as well as investigated a sample of Taverna workflows. Based on this analysis we designed the VPlan that is an ontology for description of validation requirements. We also demonstrated in what way the VPlan instance is automatically generated to validate workflows identity.

Figure 5.13 depicts a detailed view on the VFramework steps performed in the original environment. The first two steps which are *Run static analysis* and *Run dynamic analysis* were described in Chapter 4, while the last step which is *Define validation metrics* was described in this chapter. The context model which was gradually extended during the application of the VFramework in the original environment is depicted in Figure 5.14. It contains all five parts of the workflow context model (cf. Figure 3.3). The information contained in the workflow context model is later used for reference when the workflow re-execution is verified and validated. This is described in Chapter 6.

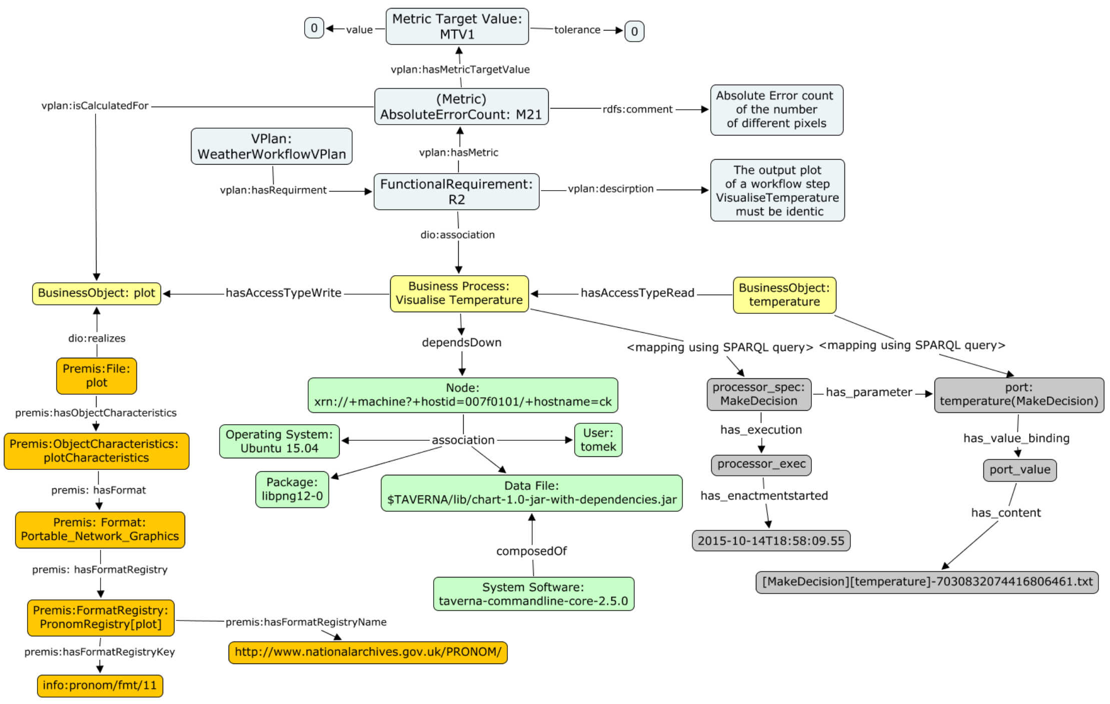

0
value
Metric Target Value: MTV1
tolerance
0
vplan:hasMetricTargetValue
vplan:isCalculatedFor
(Metric) AbsoluteErrorCount: M21
rdfs:comment
Absolute Error count of the number of different pixels
VPlan: WeatherWorkflowVPlan
vplan:hasMetric
vplan:hasRequirment
FunctionalRequirement: R2
vplan:descirption
The output plot of a workflow step VisualiseTemperature must be identic
dio:association
BusinessObject: plot
hasAccessTypeWrite
Business Process: Visualise Temperature
hasAccessTypeRead
BusinessObject: temperature
dio:realizes
dependsDown
<mapping using SPARQL query>
<mapping using SPARQL query>
Premis:File: plot
Node: xrn://+machine?+hostid=007f0101/+hostname=ck
processor_spec: MakeDecision
has_parameter
port: temperature(MakeDecision)
premis:hasObjectCharacteristics
Operating System: Ubuntu 15.04
association
User: tomek
has_execution
has_value_binding
Premis:ObjectCharacteristics: plotCharacteristics
processor_exec
port_value
Package: libpng12-0
Data File: $TAVERNA/lib/chart-1.0-jar-with-dependencies.jar
has_enactmentstarted
premis: hasFormat
has_content
Premis: Format: Portable_Network_Graphics
2015-10-14T18:58:09.55
composedOf
premis: hasFormatRegistry
System Software: taverna-commandline-core-2.5.0
[MakeDecision][temperature]-7030832074416806461.txt
Premis:FormatRegistry: PronomRegistry[plot]
premis:hasFormatRegistryName
premis:hasFormatRegistryKey
http://www.nationalarchives.gov.uk/PRONOM/
info:pronom/fmt/11

Figure 5.14: Overview of the context model parts instantiated during the VFramework application for the *Visualise Temperature* step of the weather workflow.

CHAPTER 6

Verification and validation of workflow re-execution

In this chapter we describe the VFramework steps that are performed when the workflow is re-executed in a new environment. These steps are depicted in Figure 6.1 and are: *Verify environment* and *Validate workflow*. To complete them we collect information about the re-execution from the new environment and compare it to the information collected in the original environment (see Chapters 4 and 5), which was stored in the context model.

6.1 Verify environment

This is the first step performed in the environment in which the workflow is re-executed. The verification process is iterative and coupled with the validation process that is described in Section 6.2.

Figure 6.2 presents an overview of the verification and validation processes. We start by copying the workflow to a new environment in which the experiment is replicated. We run the workflow and monitor its execution using the PMF to create the context model of the re-executed workflow. In case of Taverna workflows when the workflow model is defined in the workflow file, it is safe to assume that the workflow model has not changed, that is, has the same steps and outputs. For this reason, we can skip the static analysis of the workflow and perform only the dynamic analysis. In cases when the VFramework is applied to verify and validate workflows that were re-engineered, then the static analysis should be perform as well to identify differences between the workflow models. Having captured the context of the workflow re-execution, we compare it to the original workflow model to verify whether the same dependencies were used to compute the result. If the workflow is executed in an environment configured in the exactly same

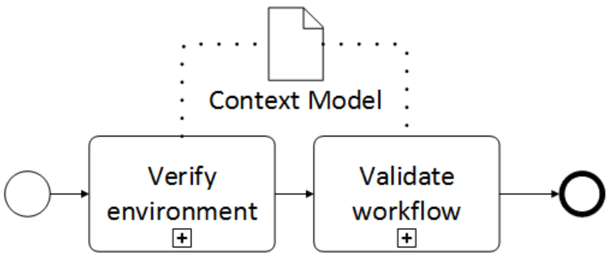


Figure 6.1: Overview of the VFramework steps performed in the redeployment environment.

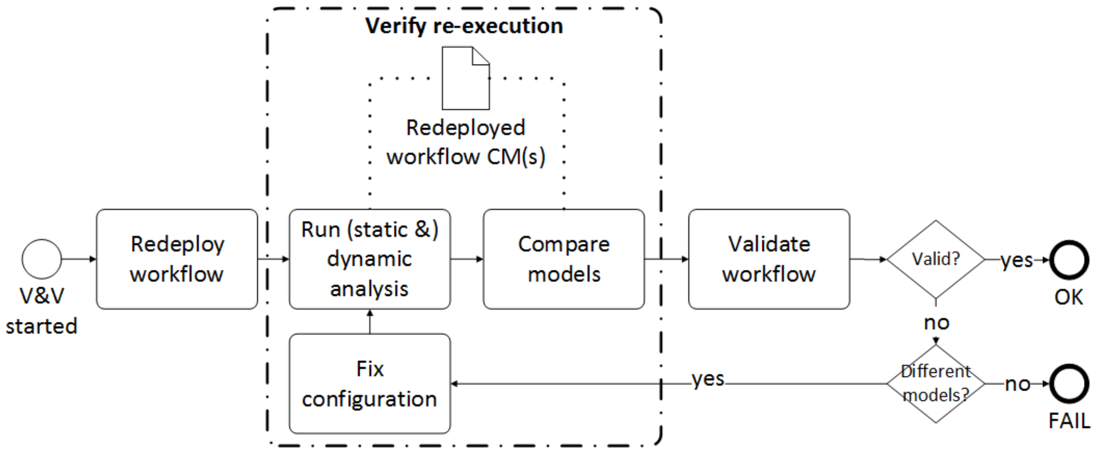


Figure 6.2: Verification of workflow re-execution as an iterative process.

way, then there are no differences between the context models. However, researchers can also replicate the workflow executions in already existing environments in which they used to run other workflows. Then there can be differences between the models, but they can have no impact on the result of the computation. To find out whether the differences between environments have impact on the workflow re-execution, we validate the re-execution in the *Validate workflow* step of the VFramework. When the validation is positive, that is the workflow re-execution matches the original execution, then such deviations are acceptable. Otherwise, the workflow environment has to be reconfigured so that the differences are eliminated. This process is iterative and repeated until the re-execution is valid or no differences between configurations of both environments exist. If the environment configurations are identical and the validation still fails, then this implies that the workflow re-execution is not replicable and may indicate that the original data was fabricated.

In the remainder of this section we describe how we perform each of the steps depicted in Figure 6.2 using the weather workflow as the example.

6.1.1 Redeploy workflow

The installation of the workflow in a new environment can be done in two ways: manually or semi-automatically using PMF. Regardless of the installation approach the rest of the verification process is performed in the same way.

The semi-automatic installation with PMF uses information stored in the context model to configure a new environment from scratch. This approach sometimes requires manual assistance, because not all of the workflow dependencies can be automatically sourced, for example packages that are not available in public repositories. PMF lists these artefacts.

The manual installation is supported with information provided in the dependency report created in the *Run dynamic analysis* step. This way of installation may be preferred by users who do not want to set up a new virtual machine, but want to install the workflow in an existing system. Another reason could be that the workflow owner was only willing to publish the context model, but not the automatically collected binaries from their system, because they had concerns that too much sensitive information would be revealed.

To manually set up the workflow environment we first check in the dependency report created for the original execution whether the workflow has any local tools invocations and whether the tools that are needed to complete them are installed in the system. On a system that uses packages we run the package manager to install all of the previously identified packages. Then we analyse which additional files and libraries were used by the workflow and copy them into appropriate locations. Some of them are likely Java libraries used by the Beanshell scripts that need to be copied into an appropriate folder of the workflow engine. However, there may be also other files which may be parts of other software running in the background. In such cases we need to perform manual investigation to identify what is the file used for and how it should be installed. Such investigation can also be needed during the semi-automatic installation made using the PMF, because such background processes are beyond workflow monitoring boundaries.

We have to make sure that external dependencies of the workflow are reachable. By capturing the provenance traces and recording traffic to the external services, we created the evidence that allows us to validate whether the external services work in the same way. If the service changed its functionality, then this will be detected during validation phase. Therefore, here we only need to set up mock-ups if we identified in the original environment that the original services cannot be used, because of lack of determinism (see Section 4.2.3).

We redeploy the weather workflow on a machine that is used by our colleague to run his own workflows. This environment should be in theory compatible with the original environment, because it is possible to run other Taverna workflows in it. Using the dependency report (see Figure 4.8), we make sure that the dependencies identified in the original environment are present. According to the report, we have to:

- Check whether the identified Debian packages are installed in the system. We do it using a package manager and install the missing packages if necessary.

- Copy identified files and libraries into corresponding folders. The weather workflow uses one Java library *chart-1.0-jar-with-dependencies.jar* that is a dependency for a Beanshell.
- Configure external dependencies. In the *Run dynamic analysis* step we identified that the workflow calls a web service that is non-deterministic and decided to create a mock-up. We run the mock-up now and spoof the identified IP addresses by editing the *hosts* file on the machine in which the workflow is executed. Thus all requests from the workflow engine are forwarded to the mock-up and not to the original service that would provide different responses than in the original execution. We do not introduce any direct changes into the workflow file.

6.1.2 Verify re-execution

We monitor every test instance for which the provenance traces were collected in the original environment. Hence, we repeat the procedure described below for each of the test instances. For the weather workflow we use one test instance. We read the values of the workflow input values from the provenance traces of the original execution using a SPARQL query.

During the verification step we make sure that all dependencies that are needed to re-execute the workflow are in the system and that they are in fact used by the workflow. For that purpose we employ the same approach as in the original environment, that is, we run the PMF to monitor the workflow execution. Thus we create a context model of the re-executed workflow that we compare with the context model of the original workflow. We compare them using Ontology Diff Tool (see Section 4.2).

Figure 6.3 depicts the results of comparison for the weather workflow. Figure 6.3a presents components that were not present in the original system but are used during workflow re-execution. We can see that:

- Two data files are accessed. The first one is a font property file used by Java, while the second is a Java library additionally loaded by the workflow engine.
- The workflow accessed *datapoint.metoffice.gov.uk* host and three other IP addresses. These are the default addresses for the localhost on which the mock-up for the *datapoint.metoffice.gov.uk* was hosted. Thus we see that the mock up was properly configured.
- The operating system is *Linux Mint 17 Qiana* and is different than in the original execution.
- A different user was running the workflow.

Figure 6.3b presents components that were used during the original computation, but are not used during the re-execution. We can see that:

- The workflow does not use two data files that are dependencies of the fonts installed in the system.
- The workflow does not access the original web service, thus we see that the mock-up is used.
- The original system was *Ubuntu 15.04* and is not used.
- There are 48 Debian packages that are not used.

The workflow context models differ also in *Infrastructure Functions* that represent the system processes created in the operating system during workflow execution. Linux-based systems use the *fork*[1] system call for that purpose and the order of process creation is non-deterministic. This means that two workflows that produce identical results using exactly the same resources can process the same data in a different order. The forking is controlled by the operating system and is beyond the scope of our analysis, because it is guranteed by the operating system, that it has no effect on the computation result. For this reason we exclude the *Infrastructure Functions* from the comparison. Please note that we still analyse resources accessed by the system processes, but do not assign these resources to particular processes.

At this stage we conclude that the way in which the workflow result is obtained is different than in the original execution, however, the differences do not necessarily affect the workflow results. To check that, we perform validation as will be described in more details in Section 6.2.

The validation failed for this re-execution. The requirement *"The output plot of the Visualise Temperature step must be identical"* failed (see Figure 6.8). This means that the rendered figures were different. Figure 6.5 depicts these figures. They contain the same information and values, but look differently.

We analysed the verification results again and looked for possible reasons that lead to altered results. We went through the lists of added and deleted elements and checked which role they play in the execution process.

The most visible discrepancy between the systems was the fact that the re-executed workflow did not use 48 Debian packages. Using a package manager we confirmed that all of them are installed in the system. Hence, the problem was not that the workflow engine is missing a dependency, but it simply did not even try loading it.

Then we checked the *Data File* that contains properties of Java fonts, but it turned out that the file contains list of fonts installed in the system and that all unused packages are also in this list.

[1] http://linux.die.net/man/2/fork

The last difference was the additional Java library loaded by the workflow engine that is only used in a different experiment. It was loaded, because it was previously added to the default folder from which Taverna loads additional libraries. There was no need to use this library when re-executing the weather workflow. Therefore, we deleted this library and ran the workflow again, also monitoring its execution with PMF.

Figure 6.4 depicts the results of the context model comparison for the re-execution of the weather workflow after deleting the library that was a dependency of a different workflow. The list of differences was significantly shorter. We can see that:

- Taverna does not load any additional Java libraries.
- Seven packages providing fonts are loaded.
- Twelve packages, also related to fonts, are not loaded.

There were still differences between the environments, but we ran the validation again and all requirements were fulfilled. Hence, the differences between the systems are acceptable and we conclude that it was possible to replicate the workflow execution.

The differences between these two environments stem from the fact that the operating systems are different. They are different flavours of Debian-based Linux systems and thus consist of slightly different set of packages. The systems differ especially in the graphical user interface and this is very likely the reason for the font packages to be different. However, as the validation shows, this has no impact on the workflow computation results. If it had, then we would have to change the redeployment platform to *Ubuntu 15.04* to further reduce differences between the systems.

Out of curiosity we investigated in what way the additional library impacted the workflow engine, so that different results were produced and also different Debian packages were used.

Taverna is a Java program that consists of multiple Java libraries that are loaded by the Java class loader. The *lib* folder in which the users place additional libraries is one of the default locations in which the class loader looks for libraries to load. All of these libraries are loaded when Taverna is started.

It turned out that the *somtoolbox_full.jar* and the *chart-1.0-jar-with-dependencies*, which is a dependency of the weather workflow, use the same *JFreeChart* library to draw plots. The classes of this library were included in each jar file. The class loader found the *somtoolbox_full.jar* first and loaded all classes found in it. When it was loading classes from the *chart-1.0-jar-with-dependencies*, then it skipped the *JFreeChart* classes because they were already loaded, even though the versions of the *JFreeChart* library were different. The *somtoolbox_full.jar* used a very old version, while the *chart-1.0-jar-with-dependencies* used the newest. The class interfaces remained the same, but their implementation changed. This led to different dependencies being loaded and different visualisations being produced. For this reason we recommend redeploying workflows into clean Taverna installations.

6.1.3 Step summary

We used the evidence created it the *Run dynamic analysis* step of the VFramework to redeploy the workflow in a new environment. We used a mock-up of a web service in order to ensure determinism of the workflow execution.

We monitored the execution of the workflow and created its context model that we compared with the context model of the original workflow execution. Thus we verified whether the same dependencies are used to compute the workflow results.

We used the weather workflow to illustrate actions performed during completion of this step. We showed that additional dependencies of other workflows that are present in the redeployment environment can alter the way in which the workflow executes. We also demonstrated in what way the information provided by the context model used together with the validation requirements helps in identification of such alterations. Finally, we showed that it is sufficient for a workflow environment to be similar to the original environment to produce the same results.

6.2 Validate workflow

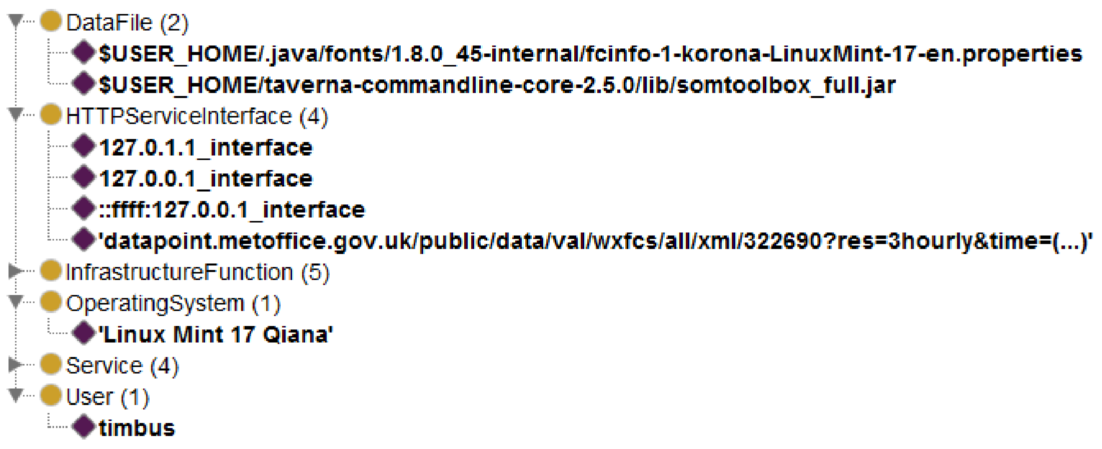
Validation checks whether the result of workflow re-execution is correct. In Chapter 5 we identified how to express validation requirements and in what way to measure them. In this step we need to check whether they are fulfilled.

We validate workflow re-execution by checking whether all metrics for the requirements defined in the step *Define validation metrics* 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 the *Validate workflow* step by implementing the VPlanComparator. Figure 6.6 presents how we use this tool. The VPlanComparator uses the context model to source the following information from the context model:

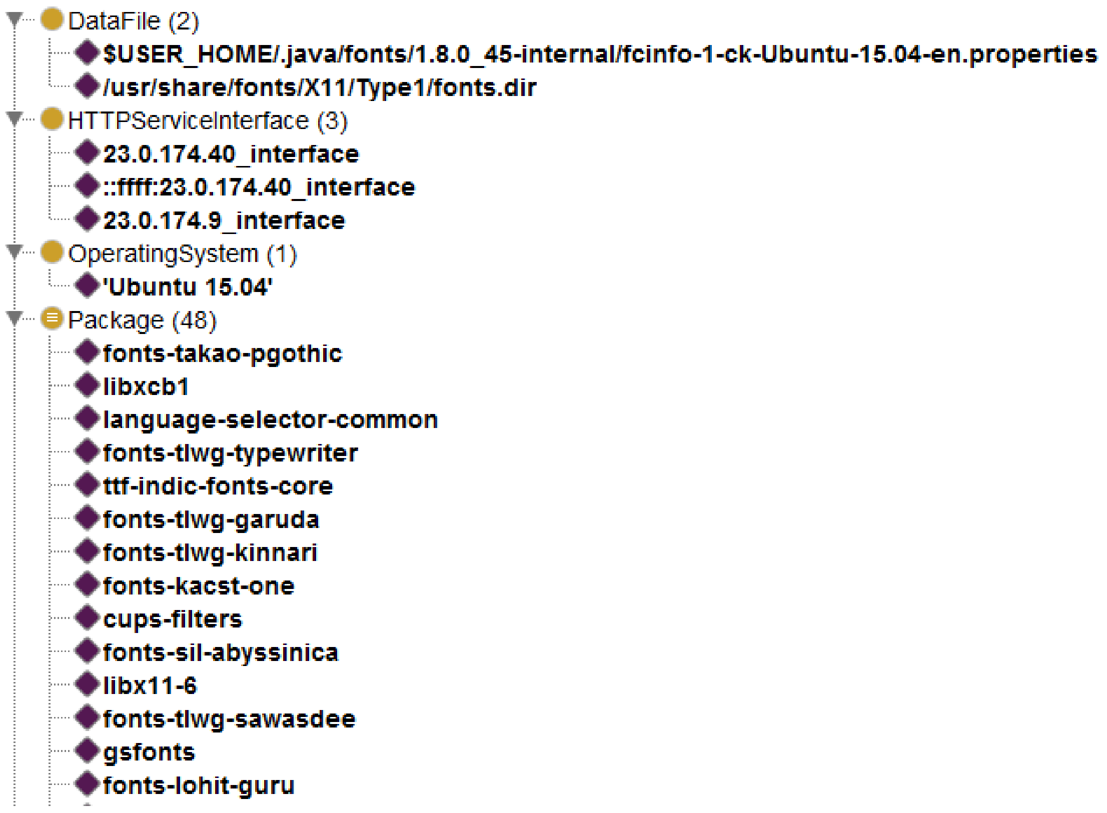
- It reads the requirements and metrics to select the right tools for metric computation.
- It reads the measurement points to identify which data to use for metric computation.
- It reads the actual data to compute the metrics.

The context model contains links to the actual data files that were captured in the original environment. The captured data consists of provenance traces and data files identified by the PMF in the *Run dynamic analysis* step of the VFramework (see *Data files used by the*



- DataFile (2)
  - $USER_HOME/.java/fonts/1.8.0_45-internal/fcinfo-1-korona-LinuxMint-17-en.properties
  - $USER_HOME/taverna-commandline-core-2.5.0/lib/somtoolbox_full.jar
- HTTPServiceInterface (4)
  - 127.0.1.1_interface
  - 127.0.0.1_interface
  - ::ffff:127.0.0.1_interface
  - 'datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/xml/322690?res=3hourly&time=(...)'
- InfrastructureFunction (5)
- OperatingSystem (1)
  - 'Linux Mint 17 Qiana'
- Service (4)
- User (1)
  - timbus

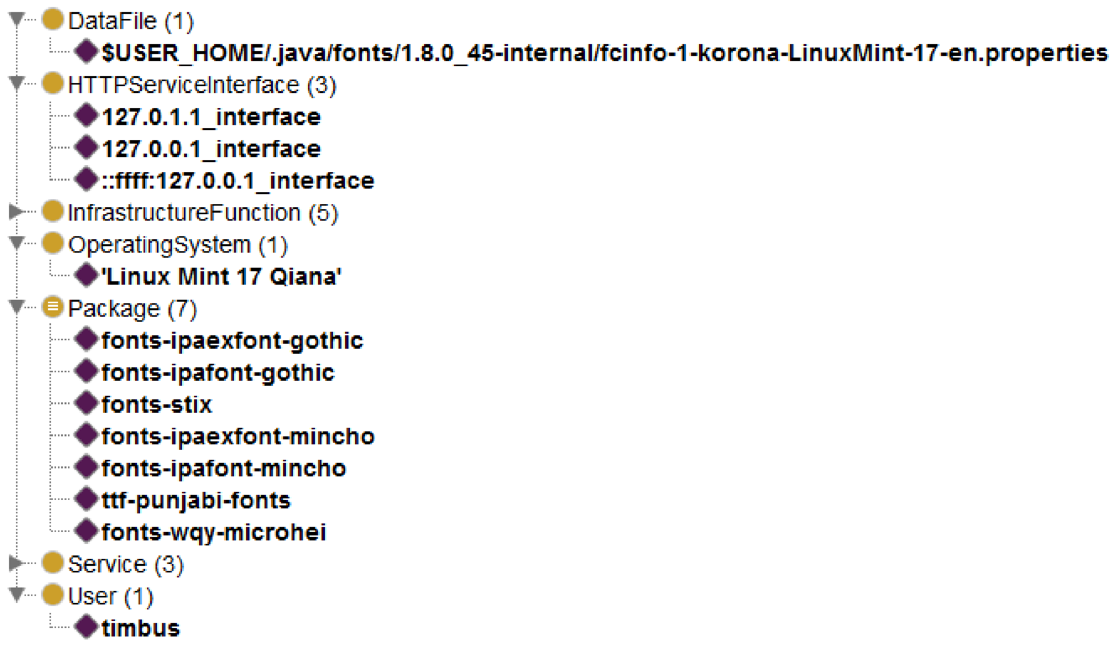
(a) Added components in the new environment that were not used in the original execution.



- DataFile (2)
  - $USER_HOME/.java/fonts/1.8.0_45-internal/fcinfo-1-ck-Ubuntu-15.04-en.properties
  - /usr/share/fonts/X11/Type1/fonts.dir
- HTTPServiceInterface (3)
  - 23.0.174.40_interface
  - ::ffff:23.0.174.40_interface
  - 23.0.174.9_interface
- OperatingSystem (1)
  - 'Ubuntu 15.04'
- Package (48)
  - fonts-takao-pgothic
  - libxcb1
  - language-selector-common
  - fonts-tlwg-typewriter
  - ttf-indic-fonts-core
  - fonts-tlwg-garuda
  - fonts-tlwg-kinnari
  - fonts-kacst-one
  - cups-filters
  - fonts-sil-abyssinica
  - libx11-6
  - fonts-tlwg-sawasdee
  - gsfonts
  - fonts-lohit-guru

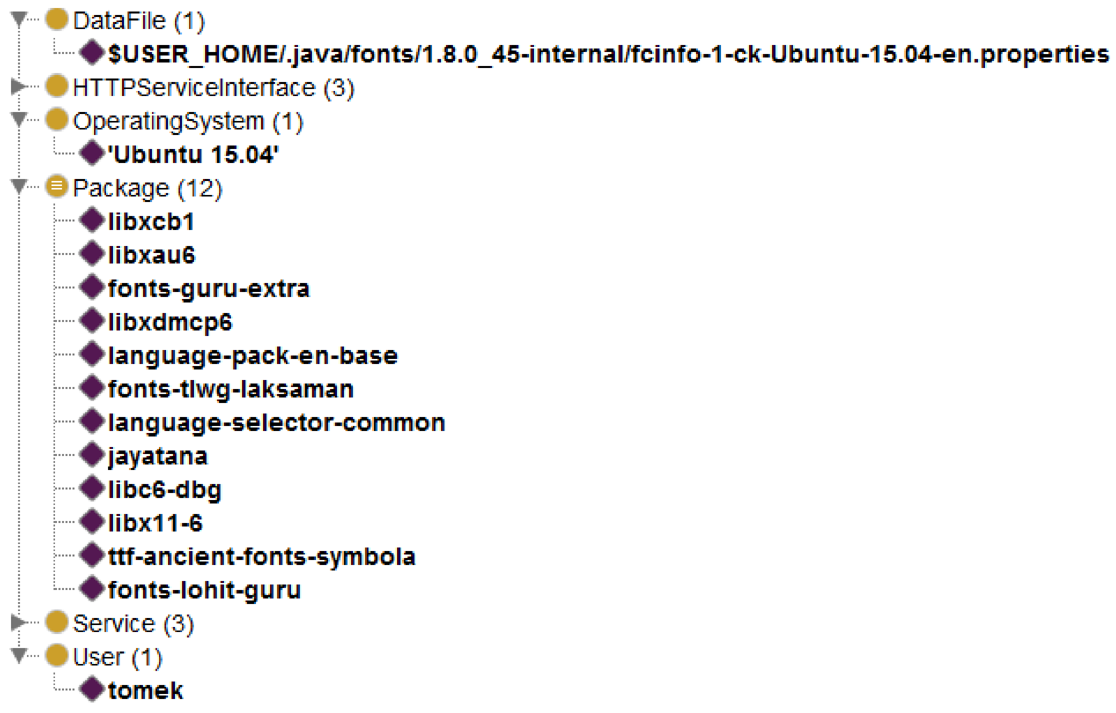
(b) Deleted components that are not used in the new environment but were used in the original execution.

Figure 6.3: Comparison of software dependencies for the original and the re-executed workflow.



- DataFile (1)
  - $USER_HOME/.java/fonts/1.8.0_45-internal/fcinfo-1-korona-LinuxMint-17-en.properties
- HTTPServiceInterface (3)
  - 127.0.1.1_interface
  - 127.0.0.1_interface
  - ::ffff:127.0.0.1_interface
- InfrastructureFunction (5)
- OperatingSystem (1)
  - 'Linux Mint 17 Qiana'
- Package (7)
  - fonts-ipaexfont-gothic
  - fonts-ipafont-gothic
  - fonts-stix
  - fonts-ipaexfont-mincho
  - fonts-ipafont-mincho
  - ttf-punjabi-fonts
  - fonts-wqy-microhei
- Service (3)
- User (1)
  - timbus

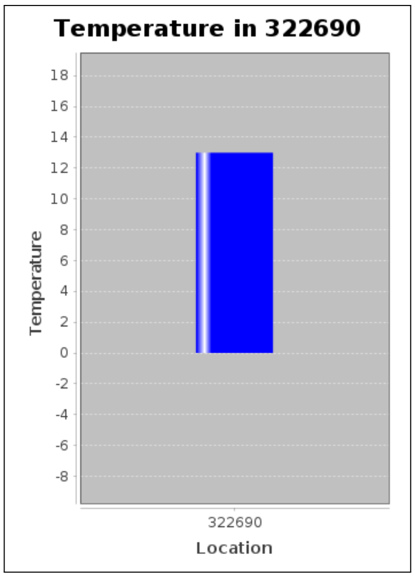
(a) Added components in the new environment that were not used in the original execution.



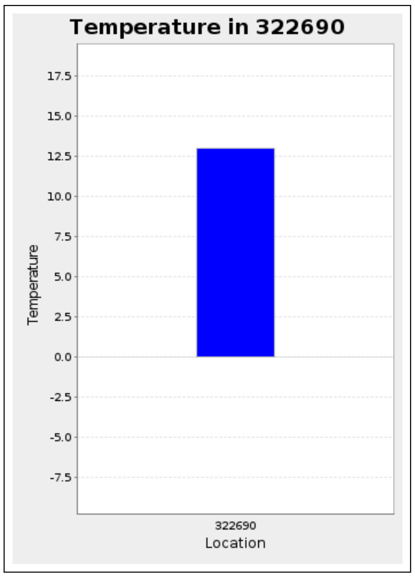
- DataFile (1)
  - $USER_HOME/.java/fonts/1.8.0_45-internal/fcinfo-1-ck-Ubuntu-15.04-en.properties
- HTTPServiceInterface (3)
- OperatingSystem (1)
  - 'Ubuntu 15.04'
- Package (12)
  - libxcb1
  - libxau6
  - fonts-guru-extra
  - libxdmcp6
  - language-pack-en-base
  - fonts-tlwg-laksaman
  - language-selector-common
  - jayatana
  - libc6-dbg
  - libx11-6
  - ttf-ancient-fonts-symbola
  - fonts-lohit-guru
- Service (3)
- User (1)
  - tomek

(b) Deleted components that are not used in the new environment but were used in the original execution.

Figure 6.4: Comparison of software dependencies for the original and the re-executed workflow after deleting the library that was a dependency of a different workflow.



(a) Visualisation produced by the original workflow.



(b) Visualisation produced by the re-executed workflow.

Figure 6.5: The outputs of the *Temperature Chart* of the weather workflow for two re-executions. Figures differ in the Y-axis labels and colour scheme. The values depicted are the same.

workflow section of the dependency report in Figure 4.8). The context model contains links to both sources of data. Both of them are processed by the VPlanCompartor taking into account following differences:

- The provenance traces associate data to a particular workflow step output, while the workflow data files identified by the PMF are linked to the element representing the workflow that aggregates all workflow steps. Hence, if any discrepancies are detected for the data which is part of the provenance traces, we can precisely identify which workflow step is affected. However, in case of the workflow data files identified by the PMF, we do not know which particular step is affected by the the detected discrepancy. This is because of the limitations of monitoring the whole workflow engine and not the individual workflow steps.

- The provenance traces always contain data describing workflow execution, while the workflow data files identified by the PMF can be empty when the workflow does not access or create files. Furthermore, to optimise the process and validate the same data once only, we can exclude duplicates from the comparison by specifying appropriate parameters to the VPlanComparator. For example, the */output/TemperatureChart/1* file depicted in Figure 4.8 is identical with the *plot* output of the *Visualise Temperature* workflow step that is already included in the provenance traces. For this reason we can exclude the file from the comparison. As

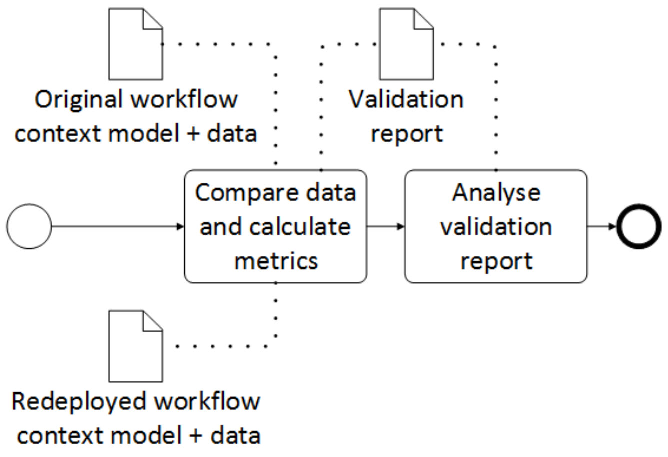

Original workflow context model + data
Validation report
Compare data and calculate metrics
Analyse validation report
Redeployed workflow context model + data

Figure 6.6: Overview of the validation process.

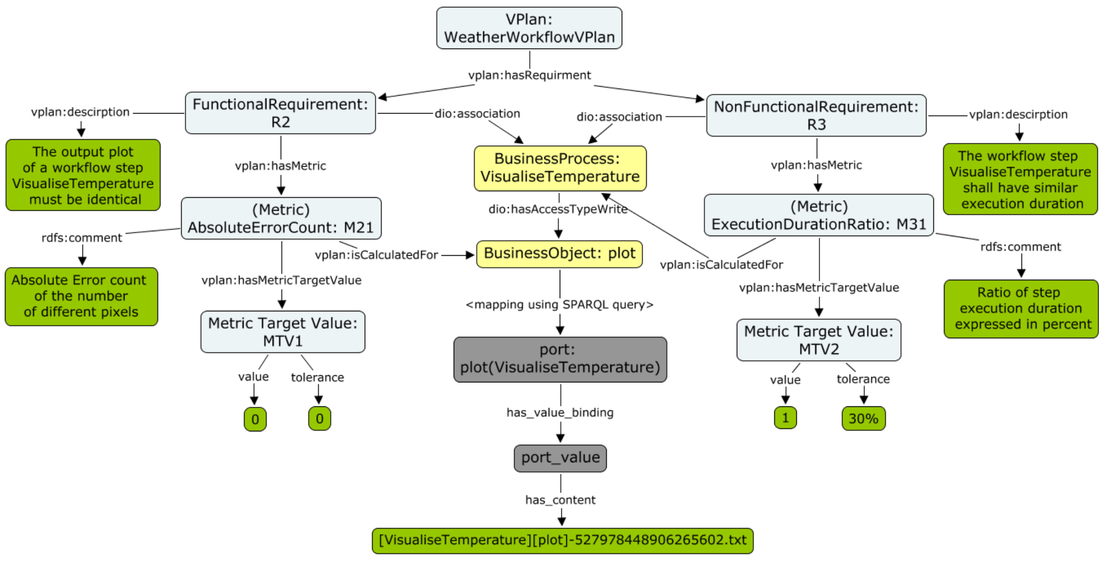

VPlan: WeatherWorkflowVPlan
vplan:hasRequirment
FunctionalRequirement: R2
NonFunctionalRequirement: R3
vplan:descirption
vplan:descirption
dio:association
dio:association
The output plot of a workflow step VisualiseTemperature must be identical
BusinessProcess: VisualiseTemperature
The workflow step VisualiseTemperature shall have similar execution duration
vplan:hasMetric
vplan:hasMetric
(Metric) AbsoluteErrorCount: M21
dio:hasAccessTypeWrite
(Metric) ExecutionDurationRatio: M31
rdfs:comment
vplan:isCalculatedFor
BusinessObject: plot
vplan:isCalculatedFor
rdfs:comment
Absolute Error count of the number of different pixels
vplan:hasMetricTargetValue
vplan:hasMetricTargetValue
Ratio of step execution duration expressed in percent
<mapping using SPARQL query>
Metric Target Value: MTV1
Metric Target Value: MTV2
port: plot(VisualiseTemperature)
value
tolerance
value
tolerance
0
0
1
30%
has_value_binding
port_value
has_content
[VisualiseTemperature][plot]-5279784489062656O2.txt

Figure 6.7: Context model depicting elements from the VPlan, workflow model and provenance traces for the *Visualise Temperature* step.

a result, for some workflows this results in only provenance traces being used. Hence, in the remainder of the dissertation we use the provenance traces to illustrate the process of data comparison and requirements evaluation. Whenever the workflow data files captured by the PMF are additionally used in the validation, we inform the reader about this.

Figure 6.7 depicts integration of the VPlan with the workflow model and the provenance traces. It combines Figure 4.13, which describes integration of provenance traces (workflow instance) with the workflow model, and Figure 5.7, which depicts integration of the VPlan and the workflow model. We described in Section 5.1 how the requirements and metrics are defined, while in Section 5.2.4 we described in what way they are automatically generated.

We can see in Figure 6.7 that the *Functional Requirement R2* is quantified using *AbsoluteErrorCount* metric that is calculated for the *plot* output of the *Visualise Temperature* workflow step. The data captured for this step is stored in file *[VisualiseTemperature][plot]-5279784489062656602.txt*. The VPlanComparator loads the VPlan and the workflow model and switches between the provenance traces to read data for both executions. This is possible because of the integration between the model and the instance data using the SPARQL queries as described in Section 4.2.4.

We explained in Section 5.1.1 and Section 5.2.3 the impact of the data formats on the provenance data comparison process. The VPlanComparator uses a suitable comparator depending on the type of metric defined in the VPlan. We implemented eight comparators that allow computation of metrics for the following file formats: HTML, MP3, PDF, PNG, TEX, XML, and ZIP. If there is no suitable comparator for the identified format, then MD5 hashes are computed. The architecture of the VPlanComparator allows adding further comparators via a plugin mechanism.

The VPlanComparator takes also into account that the sequence in which particular workflow steps are completed may sometimes differ and this has impact on the structure of provenance traces. This is because workflow engines are often multi-thread programs. For example, the Taverna workflow engine has a queue to which threads are submitted for execution [MSRO$^+$10]. If a list of values is passed to the workflow step for processing, then for each element a new thread is created and submitted to the queue. The processing of a next step begins when all of the threads belonging to a given step have finished their execution. Such a design makes it possible to process several steps at the same time. If many steps process lists of elements, there is a higher likelihood that the threads are ordered differently in the queue. This has no impact on the final result of the workflow run and the validity of results, but it results in different ordering of data captured for each step. Hence, the comparison process using provenance traces is not only comparison for equal text contents.

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

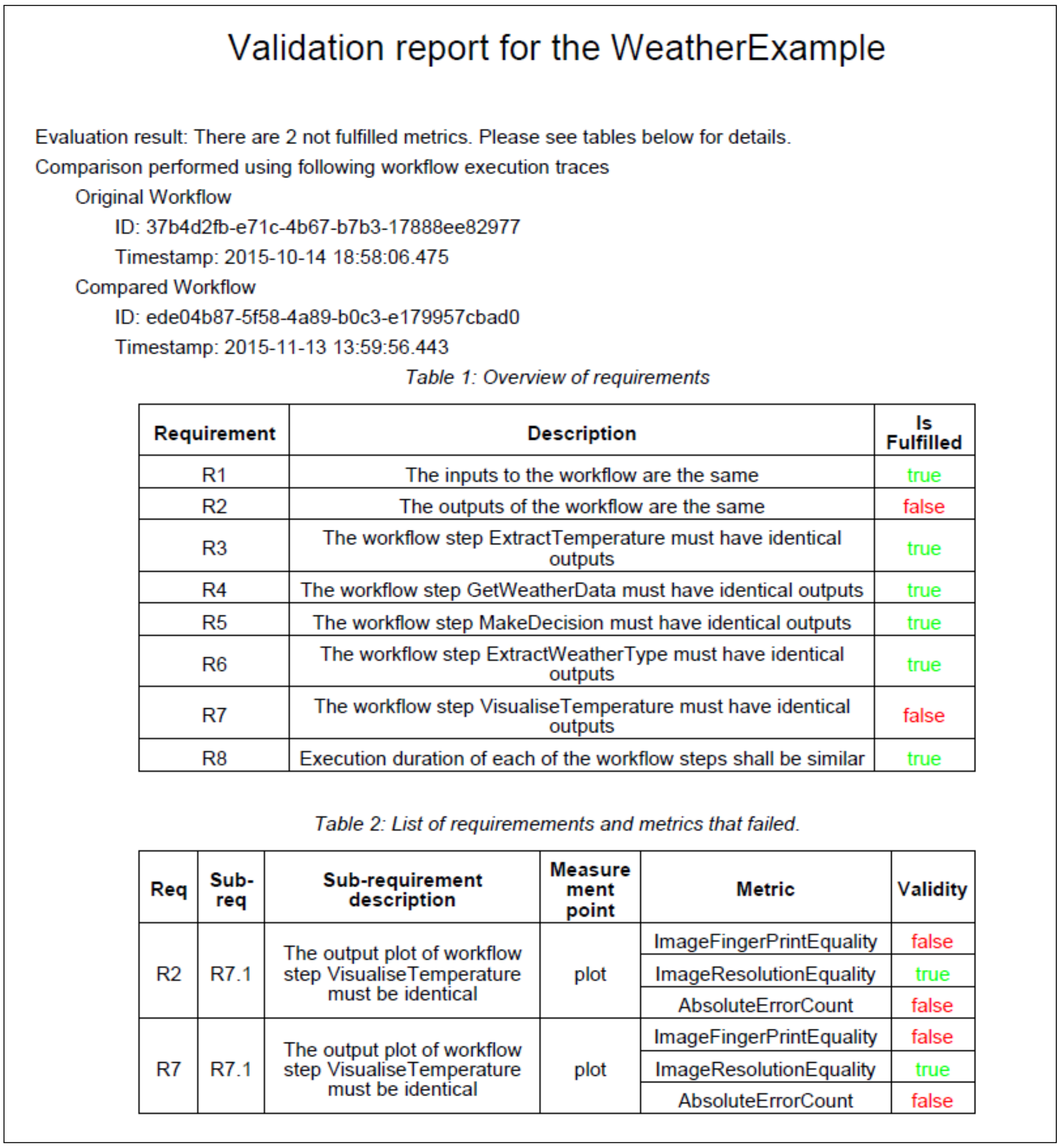
- The workflow re-execution is not valid, because two metrics failed.
- Ids and timestamps of provenance traces for which the report was produced are provided.
- Table 1 depicts the aggregated requirements and information which of them were fulfilled. Requirements *R2* and *R7* failed, that is, the workflow 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.

After tracing the problems of different libraries loaded identified in the verification step, and fixing this issue as described in Section 6.1.2, the validation confirmed successful re-execution.

6.2.1 Step summary

The *Validate workflow* step is the last step of the VFramework, which produces the result stating whether the re-execution successfully replicated the results stored in an initial set of executions.

For that purpose we used the integrated context model consisting of the workflow model, workflow instance, and the validation requirements to automatically validate workflow re-execution. We implemented a tool that based on a metric type defined in the VPlan calculates metrics' values using data captured for the original and the redeployed workflow. The tool creates a summary of requirements and metrics that were fulfilled and those which failed. Based on this report we make decision on workflow validity.



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 requirements and metrics that failed.

Req	Sub-req	Sub-requirement description	Measure ment 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 6.8: Excerpt of a validation report for the weather workflow.

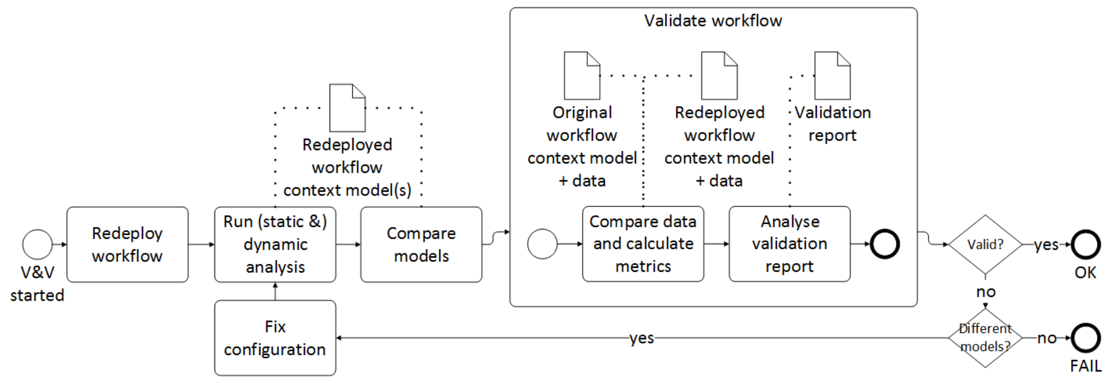

Validate workflow
Original workflow context model + data
Redeployed workflow context model + data
Validation report
Redeployed workflow context model(s)
V&V started
Redeploy workflow
Run (static &) dynamic analysis
Compare models
Compare data and calculate metrics
Analyse validation report
Valid?
yes
OK
no
Different models?
no
FAIL
yes
Fix configuration

Figure 6.9: The VFramework steps performed in the redeployment environment - a detailed view.

6.3 Summary

In this chapter we described in detail the *Verify environment* and the *Validate workflow* steps of the VFramework that are completed in the redeployment environment to verify and validate workflow re-executions. Figure 6.9 presents a detailed view on these steps.

We used the evidence created it the *Run dynamic analysis* step of the VFramework to redeploy the workflow in a new environment. We verified whether the same dependencies are used to compute the workflow results. We showed that additional dependencies of other workflows that are present in the redeployment environment can alter the way in which the workflow executes. We also demonstrated in what way the information provided by the context model used together with the validation requirements helps in identification of such alterations. We implemented a tool that based on a metric type defined in the VPlan calculates metrics' values using data captured for the original and the redeployed workflow. The tool creates a summary of requirements and metrics that were fulfilled and those which failed. Based on this report we make decision on workflow validity. Finally, we showed that it is sufficient for a workflow environment to be similar to the original environment to produce the same results.

These were the last steps of the VFramework. In the next chapter we present its evaluation on use cases.

CHAPTER 7

Evaluation

In this chapter we evalute the VFramework on a range of use cases. Taverna workflows constitute the majority of workflows published on myExperiment [MR15a] - 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 a 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 [MR15a] 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 this reason we selected five Taverna workflows that resemble these characteristics. The first workflow is from the domain of music information retrieval and presents music classification. The second workflow deals with sensor data analysis in civil engineering. The other three workflows are used in the clinical medical research for investigating aspects of Huntington's disease.

The Taverna workflow engine is available for both Linux- and Windows- based systems and hence it should be possible to obtain repeatable workflow executions in both systems. We evaluate to what extent the cross-platform verification, and validation using the VFramework is possible by re-executing two of the use cases in Linux- and Windows-based environments. We also consider to what extent the VFramework can be automated, which framework steps must be performed manually and what is the effort required to complete them. Furthermore, using the music classification use case we simulate potential changes that may occur during workflow re-execution and evaluate to what extent the VFramework and the proposed approach for selection of validation metrics detect such changes. The data used in the evaluation is available online[1].

7.1 Music classification

In this section we present the music classification use case and describe actions performed in the original environment, which are common for both re-executions. Then we discuss how the framework steps are performed in Linux and Windows environments when the workflow is re-executed. Finally, we present the simulation of changes.

7.1.1 Use case description

The first use case is a typical process in the domain of music information retrieval and machine learning, specifically the task of musical genre classification. This task deals with the categorisation of pieces of music, for which the category (genre) is unknown, into one of a set of predefined categories. Genre classification is employed in music retrieval systems. It also functions as an important mean to evaluate new techniques in the research community.

A detailed description of workflow[2] steps can be found in [MAC+15]. A schematic overview of the experiment is presented in Figure 7.1 depicting the Taverna workflow. The feature extraction analyses the spectrum of the music, and extracts information that should be representative of a piece of music, such as rhythm, instrumentation, melody, and more low-level features, into a numeric representation. Besides the music processing and machine learning aspects, the acquisition of music data and ground truth (gold standard) are also part of the workflow. Both are fetched from remote sources, e.g. a content provider such as the Free Music Archive[3], and websites such as Musicbrainz.org.

7.1.2 Original environment

In the *Run static analysis* step we generate the core of the context model using Taverna2Archi and Archi2Owl converters. This is a fully automatic process. The analysis reveals that the workflow uses one WSDL web service that we need to monitor during the dynamic analysis.

In the *Run dynamic analysis* step, we execute the workflow and capture its execution using the PMF. We use the example workflow input values as the test instance in our experiment. These values were specified by the workflow owner in the workflow definition file. We decided to enable automatic redeployment into a clean Linux machine and for

[1]https://zenodo.org/record/47252
[2]http://www.myexperiment.org/workflows/3626.html
[3]http://freemusicarchive.org/

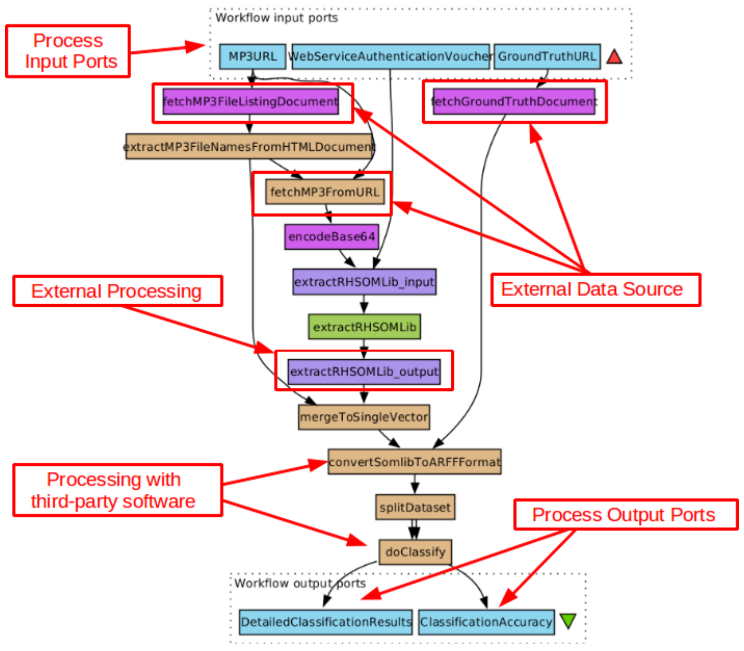


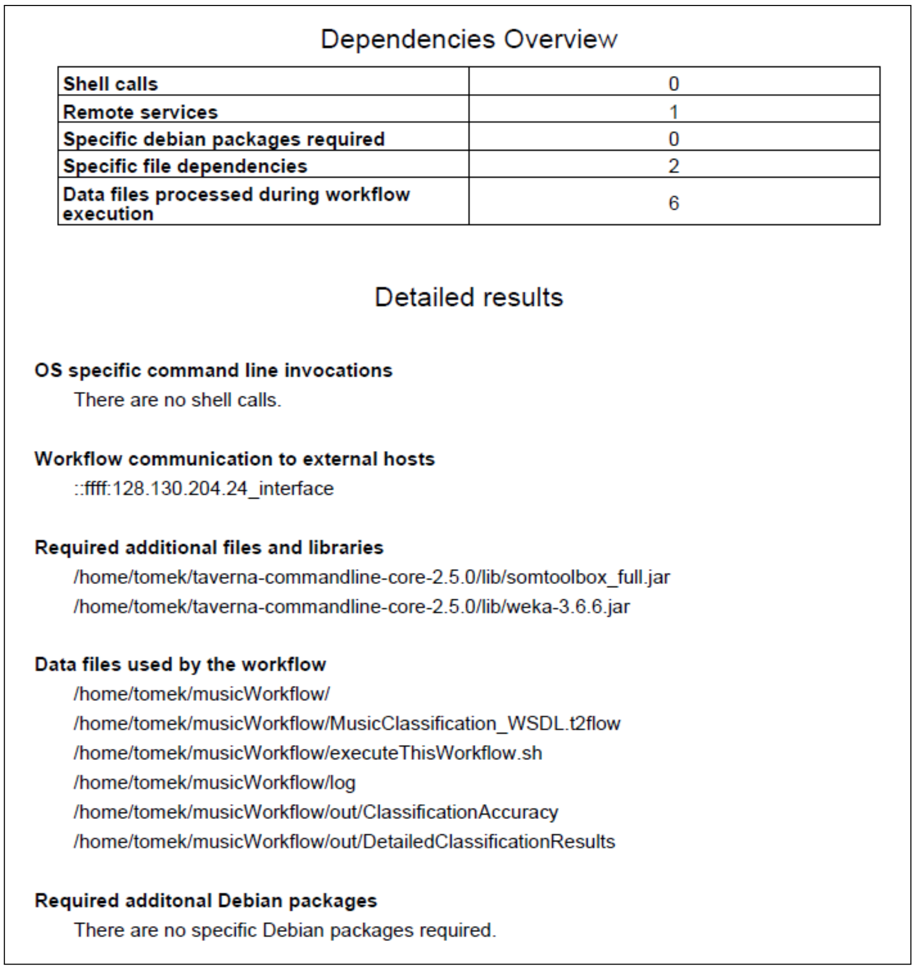
Figure 7.1: Musical genre classification, including fetching of data, modelled in the Taverna workflow engine [MAC+15].

this reason we run the PMF in a mode that not only creates the context model, but also downloads the packages and extracts the files needed to run the workflow from the original system. The extracted data contains 20 Debian packages (123 MB) and the Taverna home directory including all Java libraries (229 MB).

We also run and monitor with the PMF the calibration workflow (see Figure 4.6). Then we compare both context models to reduce the workflow engine noise and generate the dependency report that lists workflow dependencies. The dependency report is presented in Figure 7.2, while an overview of the context model elements before the noise reduction is depicted in Figure 7.3. The dependency report states that:

- The workflow does not have any shell calls.
- The workflow uses one external service.
- Two non-standard Java libraries are need for the workflow to execute.
- The workflow does not access files that are not included in the provenance traces.
- The workflow does not require specific Debian packages to be installed in the system.

We use the Web Service Monitoring Framework (see Section 4.2.3) to monitor the identified web service to check whether it is deterministic and whether we need to use a



Dependencies Overview

Shell calls	0
Remote services	1
Specific debian packages required	0
Specific file dependencies	2
Data files processed during workflow execution	6

Detailed results

OS specific command line invocations
There are no shell calls.

Workflow communication to external hosts
::ffff:128.130.204.24_interface

Required additional files and libraries
/home/tomek/taverna-commandline-core-2.5.0/lib/somtoolbox_full.jar
/home/tomek/taverna-commandline-core-2.5.0/lib/weka-3.6.6.jar

Data files used by the workflow
/home/tomek/musicWorkflow/
/home/tomek/musicWorkflow/MusicClassification_WSDL.t2flow
/home/tomek/musicWorkflow/executeThisWorkflow.sh
/home/tomek/musicWorkflow/log
/home/tomek/musicWorkflow/out/ClassificationAccuracy
/home/tomek/musicWorkflow/out/DetailedClassificationResults

Required additonal Debian packages
There are no specific Debian packages required.

Figure 7.2: Dependency report for the music classification use case.

mock-up when re-executing the workflow. We use the intercepted data of the workflow execution and monitor the service for 24 hours. The responses are always identical, hence the service is deterministic and can be used during the redeployment.

We also save the provenance traces of workflow execution and identify file formats for each step. We integrate this information with the context model.

To recap, we collected the following data during the *Run dynamic analysis* step:

- context model including provenance and file format information
- extracted data (packages and libraries)
- dependency report

In the *Define validation metrics* step, we used the VPlanAssistant tool, which implements the procedure described in Section 5.2.4, to automatically generate following requirements and sub-requirements.

R1 The workflow step *extract MP3 File Names From HTML Document* must have identical outputs.
- **R1.1** The output *mp3Names* of the workflow step *extract MP3 File Names From HTML Document* must be identical.

R2 The workflow step *merge To Single Vector* must have identical outputs
- **R2.1** The output *somLibVector* of the workflow step *merge To Single Vector* must be identical.

R3 The workflow step *fetc hMP3 File Listing Document* must have identical outputs
- **R3.1** The output contents of the workflow step *fetch MP3 File Listing Document* must be identical.

R4 The workflow step *extract RHSOMLib* must have identical outputs.
- **R4.1** The output parameters of the workflow step *extract RHSOMLib* must be identical.

R5 The workflow step *convert Somlib To ARFF Format* must have identical outputs.
- **R5.1** The output *arff* of the workflow step *convert Somlib To ARFF Format* must be identical.

R6 The workflow step *fetch MP3 From URL* must have identical outputs.
- **R6.1** The output *mp3ByteArray* of the workflow step *fetch MP3 From URL* must be identical.

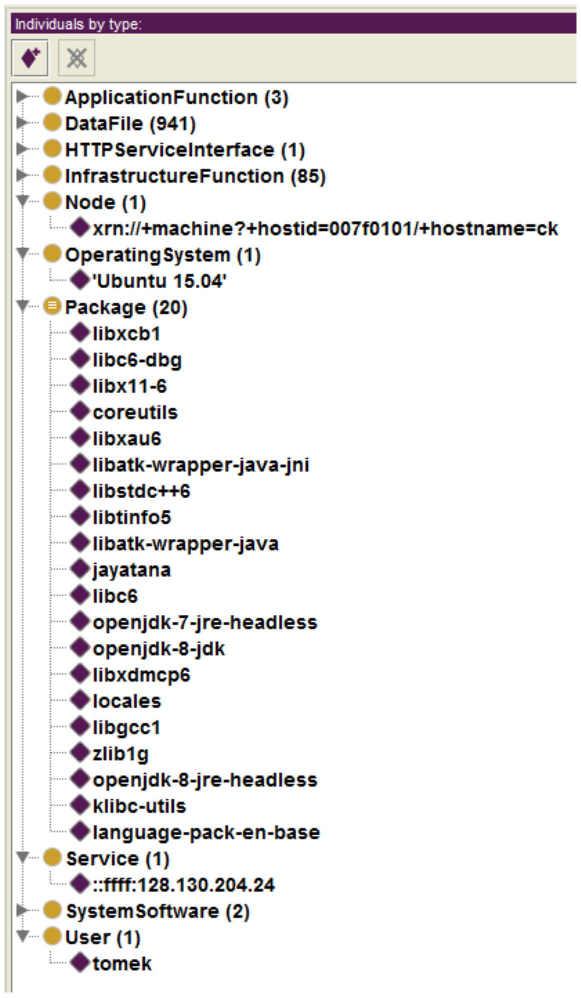
R7 The workflow step *encode Base64* must have identical outputs.
- **R7.1** The output *base64* of the workflow step *encode Base64* must be identical.

R8 The workflow step *extract RHSOMLib _output* must have identical outputs.
- **R8.1** The output *fexResult* of the workflow step *extract RHSOMLib _output* must be identical.

R9 The workflow step *do Classify* must have identical outputs.
- **R9.1** The output *detailedResults* of the workflow step *do Classify* must be identical.
- **R9.2** The output accuracy of the workflow step *do Classify* must be identical.

R10 The workflow step *extract RHSOMLib _input* must have identical outputs.
- **R10.1** The output *output* of the workflow step *extract RHSOMLib _input* must be identical.

R11 The workflow step *fetch Ground Truth Document* must have identical outputs.
- **R11.1** The output contents of the workflow step *fetch Ground Truth Document* must be identical.



```
Individuals by type:
▶ ApplicationFunction (3)
▶ DataFile (941)
▶ HTTPServiceInterface (1)
▶ InfrastructureFunction (85)
▼ Node (1)
    xrn://+machine?+hostid=007f0101/+hostname=ck
▼ OperatingSystem (1)
    'Ubuntu 15.04'
▼ Package (20)
    libxcb1
    libc6-dbg
    libx11-6
    coreutils
    libxau6
    libatk-wrapper-java-jni
    libstdc++6
    libtinfo5
    libatk-wrapper-java
    jayatana
    libc6
    openjdk-7-jre-headless
    openjdk-8-jdk
    libxdmcp6
    locales
    libgcc1
    zlib1g
    openjdk-8-jre-headless
    klibc-utils
    language-pack-en-base
▼ Service (1)
    ::ffff:128.130.204.24
▶ SystemSoftware (2)
▼ User (1)
    tomek
```

Figure 7.3: Context model elements detected by the PMF for the music classification use case.

R12 The inputs of the workflow are the same.

- **R12.1** The input *MP3URL* is identical.
- **R12.2** The input *Ground Truth URL* is identical.
- **R12.3** The *Web Service Authentication Voucher* is identical.

R13 The outputs of the workflow are the same.

- **R9.1** The output *detailedResults* of the workflow step *do Classify* must be identical.
- **R9.2** The output accuracy of the workflow step *do Classify* must be identical.

R14 Execution duration of each of the workflow steps shall be similar.

Requirements R1 to R11 deal with the correctness of results produced by each workflow step. Requirements R12 and R13 focus specifically on the correctness of workflow inputs and outputs, while the requirement R14 is a non functional requirement that validates workflow execution duration.

We use these requirements to get a high level overview of validation results. If any discrepancies are detected, then we focus on the sub-requirements that were defined for each requirement. The sub-requirements validate each output of a workflow step, for example, the requirement R1 has a sub-requirement R1.1, and the requirement R9 has two sub-requirements R9.1 and R9.2.

Table 7.1 shows how each of the sub-requirements is measured, that is, which metrics are used and for which values of these metrics the requirements are fulfilled. Due to the fact that the requirements were automatically generated and that we validate workflow identity, the target values and tolerance for all functional requirements measured by discrete metrics were set to zero. The non-functional requirement R14 says that the execution duration is similar as long as the difference is not higher than 30%. The requirements are fulfilled when all of their sub-requirements are fulfilled. For these reason the Table 7.1 depicts only sub-requirements.

7.1.3 Re-execution in Linux

We set up the workflow in a new environment by creating a new Linux virtual machine using PMF and the collected data. We select the same version of Ubuntu as the one that was run in the original environment. We obtain this information from the context model. PMF installs all packages and copies all files to the newly created virtual machine. It also copies the Taverna workflow engine, as well as the workflow file itself. Due to the lack of any specific dependencies, there are no manual adjustments required.

Then we re-execute the workflow on the newly created virtual machine with the PMF capturing the execution. We compare the context models of the re-executed workflow and the original workflow. The comparison detects no differences (apart from the number of system processes spawned by the workflow engine, but we do not consider them in the verification - see Section 6.1.2). Thus we verify that the new environment is compatible with the original one.

Using the VPlanComparator we compared the provenance traces of the re-executed workflow against the traces from the original environment. All 14 requirements are met. The workflow outputs, as well as intermediate steps including the communication to an external web service performed in the same way as the original execution.

The results of applying the VFramework confirm that the workflow re-execution is replicable. The total effort required to perform the steps of the framework is summarized in Table 7.3.

Table 7.1: Mapping of requirements to metrics and their target values for the music classification use case.

Requirement	**Metric**	**Format**	**Target Value**	**Tolerance**
R1.1	String Equality	Plain Text	0	0
R2.1	String Equality	Plain Text	0	0
R3.1	Web Page Content Equality Web Page Appearance Equality	HTML	0	0
R4.1	String Equality	Plain Text	0	0
R5.1	String Equality	Plain Text	0	0
R6.1	Audio Bitrate Equality Audio Fingerprint Equality Audio Length Equality	MP3	0	0
R7.1	String Equality	Plain Text	0	0
R8.1	String Equality	Plain Text	0	0
R9.1	String Equality	Plain Text	0	0
R9.2	Euclidean Distance	Plain Text (Double)	0	0
R10.1	String Equality	Plain Text	0	0
R11.1	Web Page Content Equality Web Page Appearance Equality	HTML	0	0
R12.1	String Equality	Plain Text	0	0
R12.2	String Equality	Plain Text	0	0
R12.3	String Equality	Plain Text	0	0
R14	Execution Duration Ratio	N/A	1	30%

7.1.4 Re-execution in Windows

The installation of the workflow on a Windows machine is a manual process. We re-execute the workflow on a Windows 7 machine on which the same version of Taverna as in the original system is installed. According to the dependency report, the workflow has only two Java libraries that need to be copied into the workflow engine directory. There are no local tools called and therefore no equivalent Windows tools are required. Having copied the libraries and having checked the availability of the external service, we execute the workflow to confirm it produces no errors.

The tools that we use for the provenance traces capturing and their comparison are implemented in Java and run on both platforms. Therefore, the provenance capturing,

Table 7.2: Number of individuals, distinct classes and file size for data collected in the music classification use case.

	Indiviudals	**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

Table 7.3: Tools and time needed for completion of framework steps for the music classification use case.

	Linux re-execution		**Windows re-execution**	
VFramework step	**Tool support**	**Effort**	**Tool support**	**Effort**
Run static analysis	Taverna Converter	5 min	Taverna Converter	5 min
Run dynamic analysis	PMF, Taverna Converter Ontology Diff Dependency Reporter	60 min	PMF, Taverna Converter Ontology Diff Dependency Reporter	60 min
Define validation metrics	VPlanAssistant	5 min	VPlanAssistant	5 min
Verify environment	PMF + Vagrant Ontology Diff	90 min	manually copy workflow and libraries	10 min
Validate workflow	VPlanComparator	5 min	VPlanComparator	5 min

as well as the validation process are performed exactly in the same way and require same effort as it was in the case of re-execution on the Linux platform.

The validation report states that the workflow executions are equivalent and therefore the workflow re-execution on a Windows machine is possible. Such an analysis was mainly possible because of two reasons: the workflow had few dependencies, and the approach proposed by us to use the provenance traces for validation is independent of the platform in which the Taverna workflow engine is installed. The total effort required to perform the steps of the framework is summarized in Table 7.3.

7.1.5 Simulation of changes

We simulated changes that may occur during workflow re-execution and evaluated to what extent the VFramework and the proposed approach for generation of requirements and validation metrics detects such changes. The test cases are based on our findings concerning repeatability of workflows and also interviews with the use case owner concerning typical alterations and mistakes in workflows engineered in the music classification domain. Table 7.4 summarises a list of potential changes to the workflow. For each of these, we identify the component or workflow step that is affected, provide a short description and discuss the expected impact of the change.

A great number of changes affect the steps that communicate with external services, as these are usually not under the direct control of the workflow owner, and may change their behaviour without any prior notice to the service consumers. In the analysed workflow, all data is loaded from external services. Some of these changes may lead to different outcomes of the workflow, e.g. when the music data is offered in a different quality, or in a different length, such as 30 second preview snippets only. This influences the values of the extracted features, and can lead to a different end result. Another major factor could be if the genre ground truth assignment, provided by a community portal, changed for some of the songs. Other changes might not have such a severe impact – e.g. the order of elements in the ground truth might change, but this is not of relevance to the implementation of the workflow, as the resulting file is aligned by the order of the music data. This, in turn, means that changes in that order can have an impact on classification algorithms that perform iterative learning and thus depend on that very same order. Besides the changes in services, we also simulate changes to the environment in which the workflow executes, by changing the operating system, Java libraries and Taverna version.

To be able to simulate all these changes, we provide a setup of the workflow in which we have full control also over the external services, and thus can modify their setup to reflect the changes in the scenarios identified above. Each of these scenarios is subsequently compared to the baseline of the original workflow execution. Table 7.5 aggregates the results of validation for each scenario using the generated requirements that we presented in Section 7.1.2. In this table *ok* indicates that the requirement was fulfilled, while *fail* indicates the opposite.

Table 7.4: Overview on possible changes in the workflow execution

ID	Component	Description	Expected Impact
C1	Ground truth	Order of elements is different	Changes in provenance of the download step, no impact on the ARFF file and classification.
C2	Music Data	Different quality of recording (e.g. lower bit-rate)	Changes in extracted features, very likely changes in classification.
C3	Music Data	Different length of music files (e.g. 30 second snippets)	Changes in extracted features, likely changes in classification.
C4	Ground truth	Genres named differently (e.g. "Hip-Hop" instead of "Hip Hop")	Syntactical changes in the ARFF file & detailed classification report, no changes on classification accuracy.
C5	Ground truth	Extra content in the genre assignment	Changes in provenance of the download step, no impact on the ARFF file and classification.
C6	Ground truth	Different ground truth assignment	Changes in provenance of the download step, the ARFF file and likely in the classification.
C7	Ground truth	Different provider, but same content	Change in input port, but no change afterwards
C8	Music Data	Different provider, but same content	Change in input port, but no change afterwards
C9	Format conversion	Different somtoolbox library	Newer version imposed ordering of elements during format conversion (arff to weka). Different split of data.
C10	Classification	Different WEKA library	Likely same results
C11	WF engine	Different Taverna version	Different implementation, but no impact on processing
C12	Operating System	Different operating system	Different environment, but no impact on processing
C13	Web Service	Interface change from WSDL to REST	Change in WS invocation, no change in results

Table 7.5: Results from the workflow validation

ID	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
R1.1	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
R2.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
R3.1	ok	fail	ok	ok	fail	fail	fail	ok	ok	ok	ok	ok	ok	ok
R4.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	fail
R5.1	ok	ok	fail	fail	fail	ok	ok	ok	ok	fail	ok	ok	ok	ok
R6.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
R7.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
R8.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	fail
R9.1	ok	ok	fail	fail	fail	ok	ok	ok	ok	fail	ok	ok	ok	ok
R9.2	ok	ok	fail	fail	ok	ok	ok	ok	ok	fail	ok	ok	ok	ok
R10.1	ok	ok	fail	fail	ok	ok	ok	ok	ok	ok	ok	ok	ok	fail
R11.1	ok	ok	fail	fail	ok	ok	ok	ok	fail	ok	ok	ok	ok	ok
R12.1	ok	ok	fail	fail	ok	ok	ok	ok	fail	ok	ok	ok	ok	ok
R12.2	ok	fail	ok	fail	fail	ok	fail	fail	ok	ok	ok	ok	ok	ok
R12.3	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
R14	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok

Runs C1, C5, C6, C7, and C8 were exposed to a change in the ground truth, and to a change in the location of the data provided at the input. These changes had no impact on the workflow computation and its final result (R9.1 and R9.2). The violated requirements indicate that different data or data sources were used.

Changes to the input data, as with a different length of the sample, and a different quality of the audio compression, as in runs C2 and C3, respectively, are indicated by all requirements from R4 to R12 failing. The origin of the wrong behaviour is correctly identified by the requirement R6.1, which is evaluated by metrics that check the length of the audio samples.

In run C4, the name of one of the ground truth genres has changed, from *Hip Hop* to *Hip-hop*. As a consequence, one of the workflow output ports, the one reporting only the aggregated performance statistic of how many music pieces were categorised accurately (R9.2), is unchanged, while the ground truth comparison metrics, feature vectors (R5.1 & R3.1) split training and test data, as well as the detailed classification report (R9.1) are all different. In this case the results are in fact semantically correct, but not identical. For the workflow owners, the result would be valid, even though some metrics failed. This is because in our validation approach we test the identity of the workflow re-execution. One may also argue that such a deviation in the way the result was obtained is already reproduction or reuse of the original workflow.

In run C9 we changed the *somtoolbox* library to one of its older versions that we obtained from the use case owner. The metrics R5.1, R9.1, and R9.2 indicated a change in the experiment result. After examination of provenance traces, we discovered that the data was differently split into training and test sets. The reason for that was a change in a format conversion library. The previous version did not sort the returned result. Thus the different order of elements returned by the older version of library changed the result of the experiment. To alleviate such a change in the workflow computation, the most promising strategy would be to ensure a certain order after every step where data is obtained from sources that are not directly under control by the workflow owner.

In run C10 we ran the workflow using different versions of the *weka* library, which is a Beanshell dependency. The original workflow uses version 3.6.6. We tested the workflow using versions 3.0.6, 3.3.1, 3.4.19 and 3.5.8. In the first three cases, the workflow execution failed. The examination of libraries revealed that the method called by a Beanshell was either not available in the expected class due to a different organisation of packages within the library, or exhibited a different method signature. Only the version 3.5.8 turned out to be compatible with 3.6.6. The result presented in Table 7.5 depicts the re-execution using the version 3.5.8.

In run C11 we executed the workflow using Taverna 2.4. The workflow specification is not compatible with Taverna 1 and therefore this version was omitted. We also tried running the workflow using Taverna 2.3. This required also downgrading Java environment to version 6. For both Taverna 2.3 and 2.4 the change did not have an effect on the workflow execution.

In run C12 we tested the impact of the operating system using Windows 7, Debian 7.4 and Ubuntu 15.04. All the requirements were fulfilled in all cases and thus the operating system had no impact on the workflow computation.

In run C13 we simulated the changes in a web service by changing its interface from WSDL to REST. This enforced changes to the workflow design in which three steps had to be adapted. These changes concerned the way in which the data submitted to the web service is formatted, how the web service is called and how the results are split for further processing. All other steps of the workflow remained unaltered. Such adaptations may be necessary when the original web service is not available, or when the workflow is reused with a different web service. Requirements R4.1, R8.1 and R10.1 failed because of different formatting of data sent and received from the web service. The result of the experiment was identical.

The results of these simulations show that when all of the requirements and metrics are fulfilled, the workflow re-execution is replicable. There are also cases when few metrics are not fulfilled but there is no impact on the replicability of workflow re-execution, i.e. those dealing with the location of data or ordering of the ground truth file. Changes to the key elements of the environment like the operating system or Taverna version did not have impact on the computation, while different versions of loaded Java libraries made the workflow not executable, or delivered changed results. The scientists validating the re-execution have to decide themselves whether a violation of a given requirement is crucial for workflow replicability.

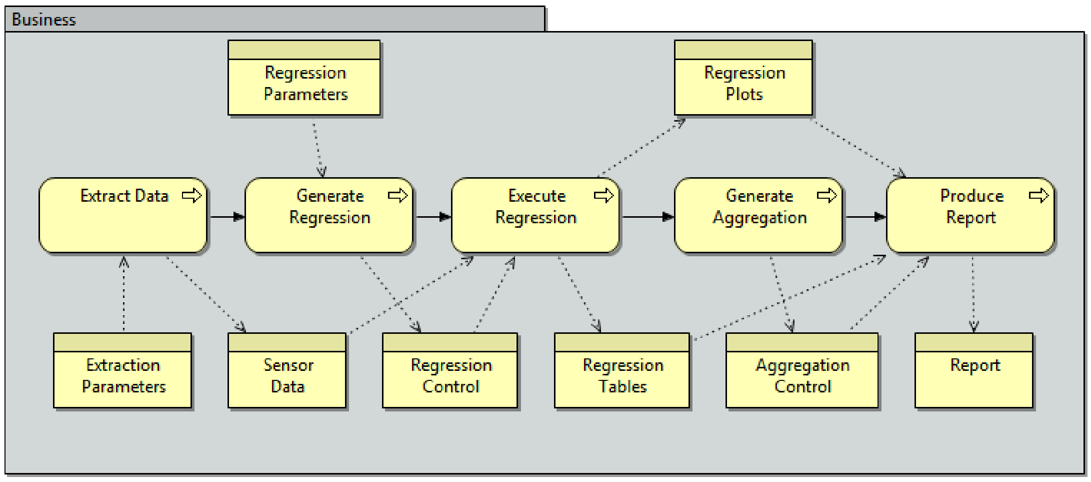


Figure 7.4: Multiple linear regression process in dam safety.

7.2 Sensor data analysis

In this section we present results for the sensor data analysis workflow that we re-execute in Linux and Windows environments. We start by providing a use case description and then we briefly describe actions performed in the original environment, which are common for both analysed re-executions. We discuss how the VFramework steps are performed in Linux, but contrary to the other use case we perform a manual installation of the workflow in an already existing system, without creating a new virtual machine with the PMF. We also describe challenges in re-executing the workflow in Windows and finally describe which assumptions must be taken to verify and validate such re-executions.

7.2.1 Use case description

This use case comes from the domain of civil engineering. The safety control of large dams is based on the monitoring of important physical quantities that characterize the structural behaviour, for example, relative and absolute displacements, strains and stresses in the concrete, discharges through the foundations, and so on. The analysis of data captured by the monitoring systems (sensor networks strategically located at dams) and their comparison with statistical, physical and mathematical models is critical for the safety control assessment [Mat11].

Figure 7.4 details a multiple linear regression process used in dam safety analysis to estimate the physical quantities based on the effects of hydrostatic pressure, temperature and time. For demonstration purposes, this process was isolated from the information system that consists of multiple hardware and software components that enable collection, processing, transformation, aggregation, and visualisation of data.

The process depicted in Figure 7.4 was modelled as a workflow consiting of four steps.

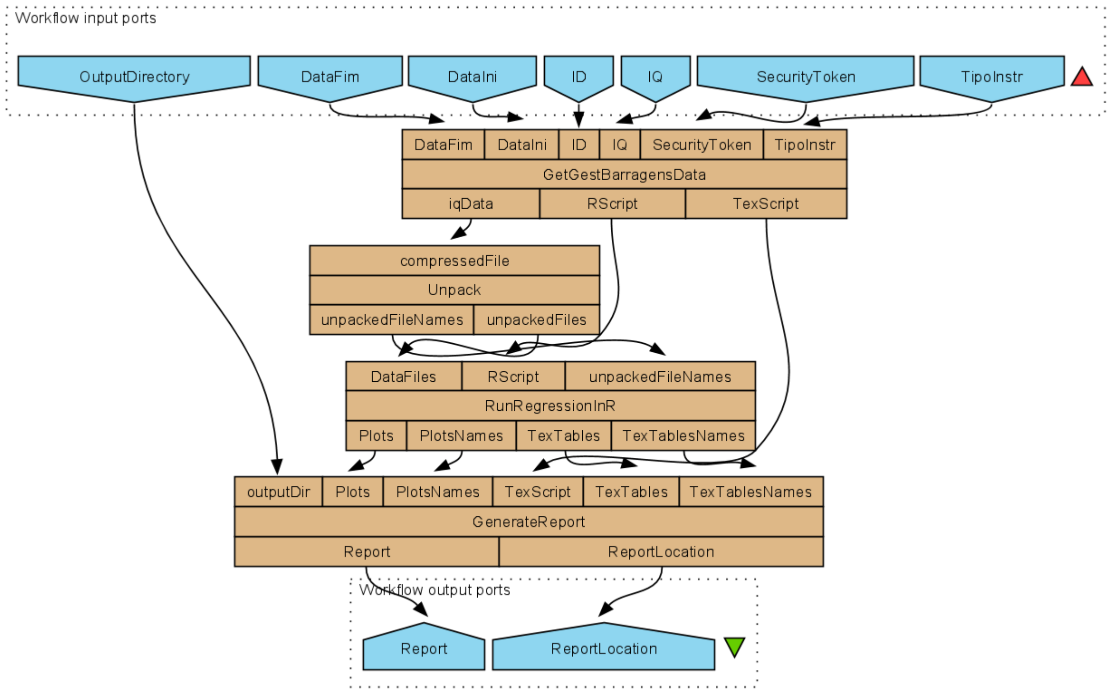


Figure 7.5: Multiple linear regression process modelled as a Taverna workflow.

The workflow is depicted in Figure 7.5. Each of the steps was implemented as a Beanshell. In the first step, the sensor data is fetched as a compressed archive from the database of the system. This is done by a client software implemented in Java that connects to a WSDL web service. The software client has a built-in authorisation token which is required to access the web service. In the second step the files are extracted and converted to an appropriate encoding. In the third step the scripts that were created previously by scientists and placed in the workflow directory are executed. They use the downloaded data and require the statistical package R. A set of PNG plots and the corresponding TEX descriptions are the outputs of the third step. In the last step the TEX scripts are compiled into a PDF report. A LaTeX environment is used for that purpose.

For accessing web services Taverna has a special pre-defined step that requires only the address of the WSDL file. However, the use case owners use their own client application instead. When analysing other workflows we identified many examples when such workarounds were introduced by their owners, even though such a functionality is provided out of the box. This shows that workflow engines are sometimes not used in a proper way and this makes it an even more challenging use case for our evaluation.

7.2.2 Original environment

We complete the first two steps of the VFramework in the same way as in the previous use case. The effort required is similar, because these two steps are well automated.

The first difference between the use cases becomes visible at the end of the *Run dynamic analysis* step, when we analyse the dependency report that is presented in Figure 7.6. The dependency report shows that part of the experiment's tasks is completed outside of the workflow engine. The reports states that:

- The workflow makes four shell calls. It uses the system shell *bash* to call *R* and *pdflatex*.
- Each of the identified shell calls requires specific Debian packages.
- The workflow uses one external service.
- There are six files that are used during processing, but cannot be attributed to any of the Debian packages.
- The workflow creates TEX and PNG files in a *temp* directory.

We use the WSMF to monitor the identified web service to check whether it is deterministic and whether we need to use a mock-up when re-executing the workflow. We use the intercepted data of the workflow execution and monitor the service for 24 hours. The responses are always identical, hence the service is deterministic and can be used during the redeployment.

We do not extract the actual files containing environment configuration data, because we use the PMF for workflow execution monitoring, but not for automatic workflow re-deployment. As sometimes scientists are reluctant to share any files that were automatically copied from their machines, we want to check the feasibility of sharing only the context model that provides the environment description, which contains among others the names and versions of Debian packages or names and paths of identified files, but not the packages and files themselves. We only copy manually the files that the workflow creates in a *temp* directory, because they are used by the workflow during its execution and must be validated.

Table 7.7 presents statistics on the size of the context model by specifying a number of individuals, number of distinct classes and size of files. It also contains information on the size of provenance traces that consist of two parts: an ontology describing the execution that has links to specific data files storing actual data captured, and this actual data.

We generate validation requirements and metrics in the same way as in the previous use case using the VPlanAssistant. Table 7.6 presents the mapping of requirements and metrics to their target values. We generated the following requirements:

R1 The workflow step *Run Regression In R* must have identical outputs.

R1.1 The output *TexTablesNames* of the workflow step *Run Regression In R* must be identical.

- **R1.2** The output *TexTables* of the workflow step *Run Regression In R* must be identical.
- **R1.3** The output *PlotsNames* of the workflow step *Run Regression In R* must be identical.
- **R1.4** The output *Plots* of the workflow step *Run Regression In R* must be identical.

R2 The workflow step *Get Gest Barragens Data* must have identical outputs.
- **R2.1** The output *iqData* of the workflow step *Get Gest Barragens Data* must be identical.
- **R2.2** The output *RScript* of the workflow step *Get Gest Barragens Data* must be identical.
- **R2.3** The output *TexScript* of the workflow step *Get Gest Barragens Data* must be identical.

R3 The workflow step *Unpack* must have identical outputs.
- **R3.1** The output *unpackedFileNames* of the workflow step Unpack must be identical.
- **R3.2** The output *unpackedFiles* of the workflow step Unpack must be identical.

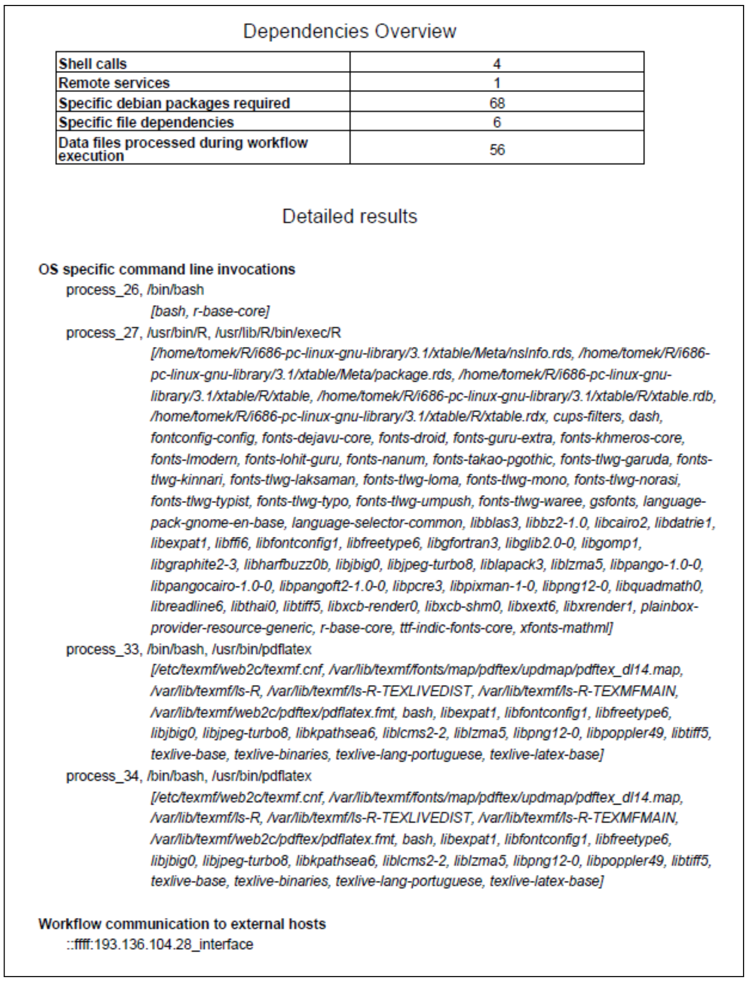
R4 The workflow step *Generate Report* must have identical outputs.
- **R4.1** The output *ReportLocation* of the workflow step *Generate Report* must be identical.
- **R4.2** The output *Report* of the workflow step *Generate Report* must be identical.

R5 The inputs of the workflow are the same.
- **R5.1** The input *TipoInstr* is identical.
- **R5.2** The input *ID* is identical.
- **R5.3** The input *IQ* is identical.
- **R5.4** The input *DataIni* is identical.
- **R5.5** The input *DataFim* is identical.
- **R5.6** The input *SecurityToken* is identical.
- **R5.7** The input *OutputDirectory* is identical.

R6 The outputs of the workflow are the same.
- **R4.1** The output *ReportLocation* of the workflow step *Generate Report* must be identical.
- **R4.2** The output *Report* of the workflow step *Generate Report* must be identical.

R7 Execution duration of each of the workflow steps shall be similar.

R8 The data files read and produced by the workflow must be identical.



Dependencies Overview

Shell calls	4
Remote services	1
Specific debian packages required	68
Specific file dependencies	6
Data files processed during workflow execution	56

Detailed results

OS specific command line invocations

process_26, /bin/bash

[bash, r-base-core]

process_27, /usr/bin/R, /usr/lib/R/bin/exec/R

[/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/Meta/nsInfo.rds, /home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/Meta/package.rds, /home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable, /home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable.rdb, /home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable.rdx, cups-filters, dash, fontconfig-config, fonts-dejavu-core, fonts-droid, fonts-guru-extra, fonts-khmeros-core, fonts-lmodern, fonts-lohit-guru, fonts-nanum, fonts-takao-pgothic, fonts-tlwg-garuda, fonts-tlwg-kinnari, fonts-tlwg-laksaman, fonts-tlwg-loma, fonts-tlwg-mono, fonts-tlwg-norasi, fonts-tlwg-typist, fonts-tlwg-typo, fonts-tlwg-umpush, fonts-tlwg-waree, gsfonts, language-pack-gnome-en-base, language-selector-common, libblas3, libbz2-1.0, libcairo2, libdatrie1, libexpat1, libffi6, libfontconfig1, libfreetype6, libgfortran3, libglib2.0-0, libgomp1, libgraphite2-3, libharfbuzz0b, libjbig0, libjpeg-turbo8, liblapack3, liblzma5, libpango-1.0-0, libpangocairo-1.0-0, libpangoft2-1.0-0, libpcre3, libpixman-1-0, libpng12-0, libquadmath0, libreadline6, libthai0, libtiff5, libxcb-render0, libxcb-shm0, libxext6, libxrender1, plainbox-provider-resource-generic, r-base-core, ttf-indic-fonts-core, xfonts-mathml]

process_33, /bin/bash, /usr/bin/pdflatex

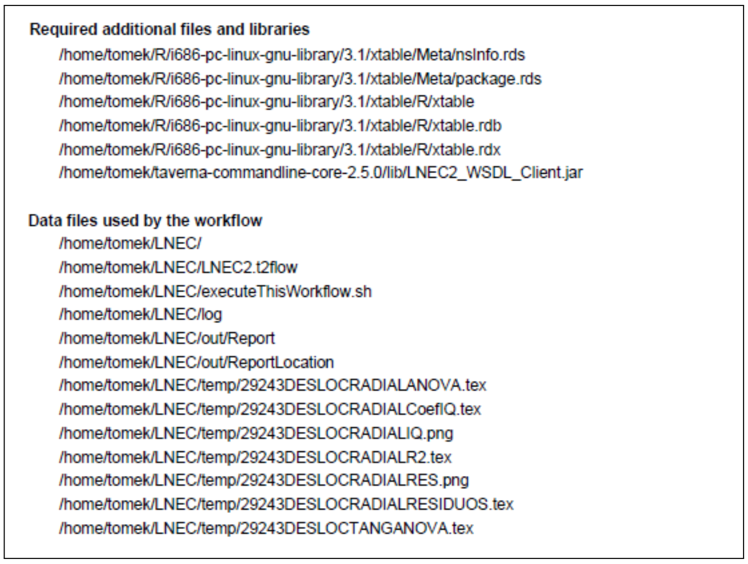
[/etc/texmf/web2c/texmf.cnf, /var/lib/texmf/fonts/map/pdftex/updmap/pdftex_dl14.map, /var/lib/texmf/ls-R, /var/lib/texmf/ls-R-TEXLIVEDIST, /var/lib/texmf/ls-R-TEXMFMAIN, /var/lib/texmf/web2c/pdftex/pdflatex.fmt, bash, libexpat1, libfontconfig1, libfreetype6, libjbig0, libjpeg-turbo8, libkpathsea6, liblcms2-2, liblzma5, libpng12-0, libpoppler49, libtiff5, texlive-base, texlive-binaries, texlive-lang-portuguese, texlive-latex-base]

process_34, /bin/bash, /usr/bin/pdflatex

[/etc/texmf/web2c/texmf.cnf, /var/lib/texmf/fonts/map/pdftex/updmap/pdftex_dl14.map, /var/lib/texmf/ls-R, /var/lib/texmf/ls-R-TEXLIVEDIST, /var/lib/texmf/ls-R-TEXMFMAIN, /var/lib/texmf/web2c/pdftex/pdflatex.fmt, bash, libexpat1, libfontconfig1, libfreetype6, libjbig0, libjpeg-turbo8, libkpathsea6, liblcms2-2, liblzma5, libpng12-0, libpoppler49, libtiff5, texlive-base, texlive-binaries, texlive-lang-portuguese, texlive-latex-base]

Workflow communication to external hosts

::ffff:193.136.104.28_interface



Required additional files and libraries

/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/Meta/nsInfo.rds
/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/Meta/package.rds
/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable
/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable.rdb
/home/tomek/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable.rdx
/home/tomek/taverna-commandline-core-2.5.0/lib/LNEC2_WSDL_Client.jar

Data files used by the workflow

/home/tomek/LNEC/
/home/tomek/LNEC/LNEC2.t2flow
/home/tomek/LNEC/executeThisWorkflow.sh
/home/tomek/LNEC/log
/home/tomek/LNEC/out/Report
/home/tomek/LNEC/out/ReportLocation
/home/tomek/LNEC/temp/29243DESLOCRADIALANOVA.tex
/home/tomek/LNEC/temp/29243DESLOCRADIALCoefIQ.tex
/home/tomek/LNEC/temp/29243DESLOCRADIALIQ.png
/home/tomek/LNEC/temp/29243DESLOCRADIALR2.tex
/home/tomek/LNEC/temp/29243DESLOCRADIALRES.png
/home/tomek/LNEC/temp/29243DESLOCRADIALRESIDUOS.tex
/home/tomek/LNEC/temp/29243DESLOCTANGANOVA.tex

Figure 7.6: An excerpt of the dependency report for the sensor data analysis use case.

Table 7.6: Mapping of requirements and metrics to their target values for the sensor data analysis use case.

Requirement	**Metric**	**Format**	**Target Value**	**Tolerance**
R1.1	String Equality	Plain Text	0	0
R1.2	String Equality Number Of Visible Differences	TEX	0	0
R1.3	String Equality	Plain Text	0	0
R1.4	Image Fingerprint Equality Image Resolution Equality Absolute Error Count	PNG	0	0
R2.1	Number Of Different Files	ZIP	0	0
R2.2	String Equality	Plain Text	0	0
R2.3	String Equality Number Of Visible Differences	TEX	0	0
R3.1	String Equality	Plain Text	0	0
R3.2	String Equality	Plain Text	0	0
R4.1	String Equality	Plain Text	0	0
R4.2	Page Number Equality Text Content Equality Number of Pages With Different Appearance	PDF	0	0
R5.1 (...) R5.7	String Equality	Plain Text	0	0
R7	Execution Duration Ratio	N/A	1	30%

Table 7.7: Number of individuals, distinct classes and file size for data collected in a sensor data analysis use case.

	Indiviudals	**Distinct Classes**	**File size (kB)**
Static CM	66	12	42
Dynamic CM	1177	10	1404
PREMIS	68	4	41
VPlan	66	14	76
CM total	1377	35	1563
Dynamic CM - without noise	199	7	138
Provenance ontology	74	6	104
Provenance data	N/A	N/A	3.17 MB

7.2.3 Re-execution in Linux

We copy the workflow file to an existing Ubuntu 15.04 machine that had Taverna 2.5 installed and was used for running other experiments. We install all the Debian packages as listed by the dependency report using a package manager. Some of the packages were already present in the system, for example *texlive-base*, and their installation was skipped, while other like for example *textlive-lang-portuguese* has to be added. The installation process takes a couple of minutes and depends primarily on the internet connection speed.

Then we focus on the shell calls made by the workflow and investigate whether the programs called are installed in the system. Both *R* and *pdflatex* are present in the system. In the list of additional files and libraries we find *LNEC2_WSDL_Client.jar* which has to be copied into the Taverna library folder. This file is provided together with the workflow. The list also contains files located in the following directory: *$userhome/R/i686-pc-linux-gnu-library/3.1/xtable/R/xtable.* To identify how to install them in the system we make a simple web search, which reveals that they are part of a special extension of *R* tool suite in version 3.1. *R* was already installed in the system. The *xtable* extension has to be installed in a different way using *R* commands.

Having installed the dependencies we run the PMF to monitor the workflow execution. We compare the context model of the re-executed workflow with the original one. There are no discrepancies between them, that is, the re-executed workflow makes the same shell calls, accesses the same web service, and during its execution the files having the same names are created in a local folder. We validate the actual contents of these files later.

We compare the provenance traces of both executions using the VPlanComparator. We also use the TEX and PNG files detected by the PMF, which were copied from the original environment, to compare them against the files created in the re-execution. In total, there are five different file formats identified in the workflow and for each an appropriate comparator is used.

The requirements R3, R5, and R7 that deal with the correctness of input data and workflow duration are fulfilled. This means that the same data is used in the workflow execution and that the workflow execution duration is similar.

Requirements R1, R2, R4, R6, and R8 failed. We investigate reasons for each of them failing by looking into their sub-requirements and checking logs created by the VPlanComparator that computed the metrics for each sub-requirement.

The requirements R1 and R8 failed because of the requirement R1.2 being not fulfilled. The requirement R1.2 checks whether the TEX tables generated in the *Run Regression In R* step are correct. There were two metrics used for evaluation of this requirement: *String Equality* that failed because the contents of the files were different, and *Number Of Visible Differences* that was fulfilled. This means that the data was correct, but likely the structure of the TEX file was different. The TEX comparator uses *latexdiff* that provides a detailed comparison summary. We checked the summary report and it turned out that the TEX files contain automatically added comments in their beginning which contain a creation date that is set for each re-execution. Apart from this discrepancy the remainder of the files are identical.

The requirements R2 and R8 failed because of the requirement R2.3 that validates another TEX script used in the workflow. The reason for failing is identical as in the requirement R1.2, that is, the files had a creation date in their header and the rest of the content was the same.

The requirements R4 and R6 failed because of the requirement R4.2 that validates the PDF report produced as the output of running the workflow. All four metrics used to evaluate the requirement R4.2 failed, that is the report had different contents, different number of pages and thus different appearance. We checked the log files of the comparator used for PDF comparison, as well as the report itself, and we recognized that:

- The first page contains a date when the report was generated and this date varies for each re-execution. For this reason the *Text Content Equality* metric failed.
- The report varies in a number of pages and has different alignment of plots and tables, but the data depicted is identical.

The fact that data presented in the report was identical is confirmed by the requirement R1.4 being fulfilled and R1.2 that failed but only due to header differences and not the content differences. This implied that the PNG plots were identical and the data used to produce the PDF report was the same. The PDF report contained the same information (apart from the timestamp in the first page) and was only differently formatted.

According to the automatic analysis performed using the VFramework, this workflow re-execution was not replicable. However, one can claim that there are no semantic differences in the final version of the report and the intermediate data used to produce the final report was confirmed to be correct. The evidence provided by the VFramework allows making both claims.

To fix the problem of the final report, we had to contact the use case owner for help. It turned out that a default latex style file *article.cls* was modified in the original environment. Once we copied the file into our system, the workflow delivered reports differing only in the timestamp in the first page. The PMF detected the style file but it also identified it as being part of a default Latex package and therefore it was not included in the model. The PMF does not check whether the files that belong to a package were modified. It is also a bad practice to replace default Latex files to create considerably new versions, but as this example shows, such situations must also be taken into account. This also shows that the verification should always be accompanied by validation to state whether a workflow re-execution is replicable. Furthermore, it might be recommended to verify all files using hash signatures.

7.2.4 Re-execution in Windows

The installation of the workflow on a Windows machine is a manual process. The context model and the dependency report created for the original environment are used to gain insight into the original configuration of the workflow environment.

We port the workflow to a Windows 7 machine on which Taverna in the same version is installed. We also copy the Java library used by the workflow. We analyse the local tool invocations and look for equivalent software tools for Windows platform. We find R in the same Windows version and install it in the system. In case of Latex environment we have to switch from *texlive* to *MiKTeX*. We know from the dependency report that the basic Latex installation needs additional packages, but the Windows version installs them on the go, whenever the package is needed. We only have to copy the *article.cls* style sheet that we identified during the preceding re-execution in Linux as crucial. We already know from the Linux re-execution that we have to install additional *xtable* package manually.

We run the workflow and it produces errors, the analysis of the error log reveals that the syntax of commands used to make shell calls is incompatible with Windows. We have to rewrite the commands to make the workflow executable. This is another example of how local dependencies limit workflow portability.

We save the provenance traces which we use for comparison in the last step of the VFramework. The validation report states that the same requirements as for the redeployment on Linux are violated. This was again due to the timestamp included in the TEX files and in the first page of PDF report. Apart from this, the workflow executions matches and therefore the workflow re-execution on a Windows machine can be regarded as replicable.

In contrast to the other use case, which had significantly less local dependencies, the installation process was more complex and required more effort. It also involved introducing changes into the workflow file itself. The effort required to perform the steps of the framework is summarized in Table 7.8.

Table 7.8: Tools and time needed for completion of framework steps for the sensor data analysis use case.

	Linux re-execution		**Windows re-execution**	
VFramework step	**Tool support**	**Effort**	**Tool support**	**Effort**
Run static analysis	Taverna Converter	5 min	Taverna Converter	5 min
Run dynamic analysis	PMF, Taverna Converter Ontology Diff Dependency Reporter	60 min	PMF, Taverna Converter Ontology Diff Dependency Reporter	60 min
Define validation metrics	VPlanAssistant	5 min	VPlanAssistant	5 min
Verify environment	PMF Ontology Diff	75 min	manual actions	45 min
Validate workflow	VPlanComparator	10 min	VPlanComparator	10 min

7.3 Clinical medical research

The third use case consists of three workflows and comes from the clinical medical research domain. These workflows are used in experiments investigating aspects of Huntington's disease and were implemented by a team of scientists from the Leiden University Medical Centre (HG-LUMC) and motivated as follows:

"Huntington's disease (HD) is the most commonly inherited neurodegenerative disorder in Europe, that affects 1 out of 10 000 people. Although the genetic mutation that causes HD was identified 20 years ago, the mechanisms leading to the HD are still poorly understood.

Transcriptional deregulation is a prominent feature of HD with gene expression changes taking place even before first symptoms arise. Epigenetic alterations can be responsible for such transcriptional abnormalities. Linking changes in gene expression to epigenetic information might shed light on the disease aetiology.

The team from HG-LUMC analysed HD gene expression data from three different brain regions. They integrated it with publicly available epigenetic data to test for overlaps between differentially expressed genes in HD and these epigenetic datasets."[BZG+15]

The team uses Taverna workflows for their experiments. Many of the workflows are published in the *myExperiment*[4] platform. Together with the use case owners we apply the VFramework to the three workflows described below.

In Section 7.3.1 we present redeployment of the workflow in an equivalent Linux environment and show how the calibration workflow helps in filtering the specific workflow dependencies. The annotate workflow uses WSDL web services to perform tasks. We compare two redeployments of this workflow, first using the original web service, second using a mock-up of the web service. We evaluate whether the mock-up has impact on the processing of the workflow and its results.

In Section 7.3.2 we present how to verify and validate a workflow that uses an external service which is an R server running in parallel to the workflow engine. We show how the static analysis of the context model helps in identifying service dependencies. Furthermore, we check to what extent we can verify and validate tasks that are completed outside of the workflow engine, namely in the R server. We compare the implementation of the workflow to the sensor data analysis use case (see Section 7.2) that also executes R scripts but through shell calls.

In Section 7.3.3 we apply the VFramework to a workflow that uses Ruby scripts to complete tasks. We investigate whether we can detect Ruby dependencies and thus verify the workflow execution. We also show that the provenance data can be incomplete when a workflow completes tasks using shell calls. In such cases we use files that were detected by the PMF to validate workflow re-execution.

7.3.1 Annotate genes workflow

We first present the use case description and then describe how we completed the steps of the VFramework in the original and the redeployment environment.

Use case description

Figure 7.7 depicts the *annotate workflow*[5] that is used to annotate gene lists. The workflow uses a local knowledge base to enrich the knowledge about a set of input genes. Therefore, this workflow takes as an input a list of comma separated *entrez gene identifiers*, the database name that will be used to map the gene ids to the local concept profile identifiers, a cut-off parameter for the number of annotations to be obtained, and an identifier for the predefined concept set id that is used in the local database. The workflow uses *anni* web services. This workflow was used in the [BZG+15] to evaluate Research Objects.

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

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

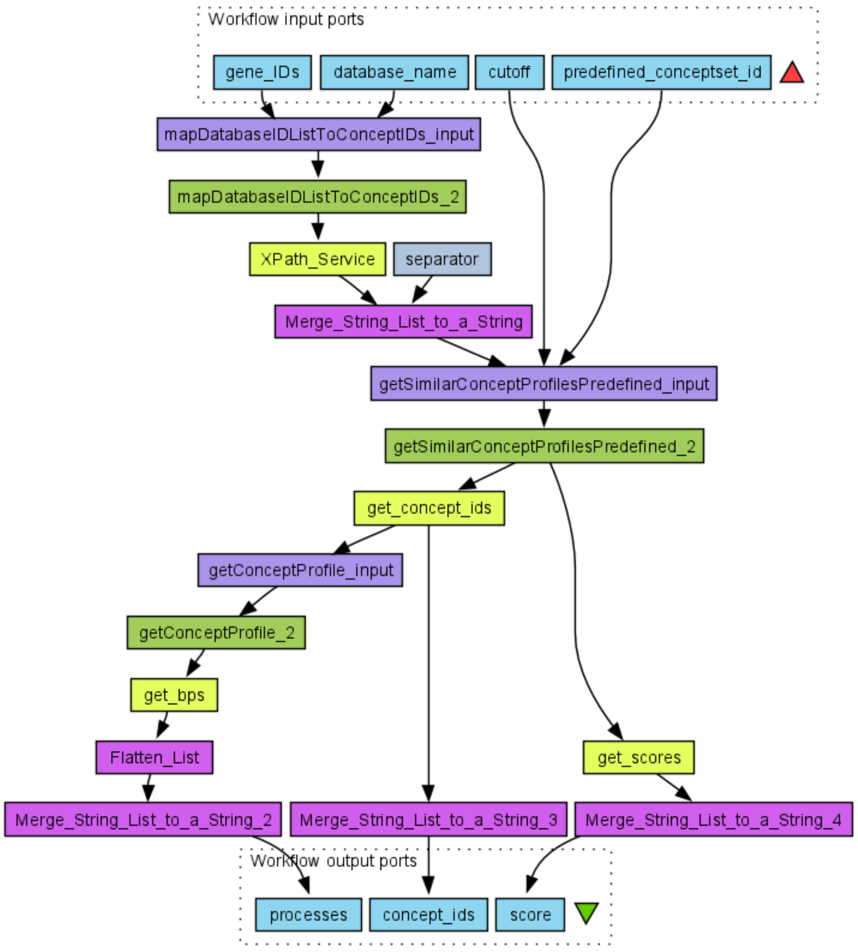

Workflow input ports
gene_IDs
database_name
cutoff
predefined_conceptset_id
mapDatabaseIDListToConceptIDs_input
mapDatabaseIDListToConceptIDs_2
XPath_Service
separator
Merge_String_List_to_a_String
getSimilarConceptProfilesPredefined_input
getSimilarConceptProfilesPredefined_2
get_concept_ids
getConceptProfile_input
getConceptProfile_2
get_bps
Flatten_List
get_scores
Merge_String_List_to_a_String_2
Merge_String_List_to_a_String_3
Merge_String_List_to_a_String_4
Workflow output ports
processes
concept_ids
score

Figure 7.7: The *annotate workflow*.

Original environment

In the *Run static analysis* step of the VFramework we identify that:

- The workflow uses one WSDL web service hosted on a remote server.
- The workflow uses Beanshell scripts that do not require any additional libraries to be shipped with the workflow.
- The workflow uses XPath services.

Based on these results, in the *Run dynamic analysis* step of the VFramework we capture the communication between the workflow and the web service. The workflow owner confirmed that the web service is deterministic and hence there is no need for its continuous monitoring. We transform the captured data by grouping it into request and response pairs. We use this data to create a mock-up of the service when the workflow is re-executed.

We also monitor the workflow execution using the PMF. Figure 7.8a depicts the workflow context model created by the PMF. We can read from it:

- The unique identifier of the platform for which the capturing took place (*Node*).
- The name of the user who ran the workflow (*User*).
- The version of Taverna that was used for running the workflow (*System Software*).
- The IP Address, as well as the domain name of the WSDL web service (*Service, SOAP Service Interface, HTTP Service Interface*).
- The version of the operating system (*Operating System*).
- The workflow does not access files that are not included in the provenance traces (*Data File*).

There are also almost a thousand of *Data Files* that include specific Taverna dependencies and temporary files. To filter out only the workflow specific dependencies and thus increase the readability of the dependency report we use the calibration workflow that does no processing. We also monitor its execution with the PMF and then compare the context models of both workflows. The results of comparison are depicted in Figure 7.8b and Figure 7.8c.

Figure 7.8b depicts the elements that were not used by the calibration workflow and thus are specific to the analysed workflow. We can see that the number of *Data Files* was reduced from 964 to 12. The list contains nine files that are workflow data files such as input and output data. The other three files belong to the *java-7-oracle* installation and were not assigned to a corresponding *oracle-java-7-instaler* package. These files must be saved and moved with the workflow to the redeployment environment. We can also see that as a consequence of noise reduction the amount of *Packages* was reduced from ten to one *Package*. This means that the other nine packages were common for both workflow executions and that Taverna uses them by default. The *libnss-mdns* package is the only specific package required by the analysed workflow.

Figure 7.8c depicts elements that were used by the calibration workflow, but are not used by the analysed workflow. We can see that they are only data files of the calibration workflow and the workflow itself. This confirms that we did not remove other key dependencies of the workflow, like Debian packages or services.

In the *Define validation metrics* step of the VFramework we generate validation requirements that are presented below. The workflow steps exchange only *Plain Text* data hence for all of the sub-requirements we use the *String Equality* metric. Only the data collected for evaluation of requirements R12.1, R16.1, and R16.2 consists of floating point numbers and in such case we use the *Euclidean Distance* metric instead.

R1 The workflow step *getSimilarConceptProfilesPredefined_input* must have identical outputs.
- **R1.1** The output *output* of the workflow step *getSimilarConceptProfilesPredefined_input* must be identical.

R2 The workflow step *Merge_String_List_to_a_String_3* must have identical outputs
- **R2.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_3* must be identical

R3 The workflow step *getConceptProfile_2* must have identical outputs
- **R3.1** The output *parameters* of the workflow step *getConceptProfile_2* must be identical

R4 The workflow step *getConceptProfile_input* must have identical outputs
- **R4.1** The output *output* of the workflow step *getConceptProfile_input* must be identical

R5 The workflow step *Merge_String_List_to_a_String_2* must have identical outputs
- **R5.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_2* must be identical

R6 The workflow step *getSimilarConceptProfilesPredefined_2* must have identical outputs
- **R6.1** The output *parameters* of the workflow step getSimilarConceptProfilesPredefined_2 must be identical

R7 The workflow step *Merge_String_List_to_a_String* must have identical outputs
- **R7.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String* must be identical

R8 The workflow step get_bps must have identical outputs
- **R8.1** The output *nodelist* of the workflow step *get_bps* must be identical

R9 The workflow step *XPath_Service* must have identical outputs
- **R9.1** The output *nodelist* of the workflow step *XPath_Service* must be identical

R10 The workflow step *mapDatabaseIDListToConceptIDs_2* must have identical outputs
- **R10.1** The output *parameters* of the workflow step *mapDatabaseIDListToConceptIDs_2* must be identical

R11 The workflow step *get_scores* must have identical outputs
- **R11.1** The output *nodelist* of the workflow step *get_scores* must be identical

R12 The workflow step *mapDatabaseIDListToConceptIDs_input* must have identical outputs

- **R12.1** The output *output* of the workflow step *mapDatabaseIDListToConceptIDs_input* must be identical

R13 The workflow step *Flatten_List* must have identical outputs

- **R13.1** The output *outputlist* of the workflow step *Flatten_List* must be identical

R14 The workflow step *get_concept_ids* must have identical outputs

- **R14.1** The output *nodelist* of the workflow step *get_concept_ids* must be identical

R15 The workflow step *Merge_String_List_to_a_String_4* must have identical outputs

- **R15.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_4* must be identical

R16 The inputs of the workflow are the same.

- **R16.1** The input *cutoff* is identical.
- **R16.2** The input *predefined_coneptset_id* is identical.
- **R16.3** The input *gene_IDs* is identical.
- **R16.4** The input *database_name* is identical.

R17 The outputs of the workflow are the same.

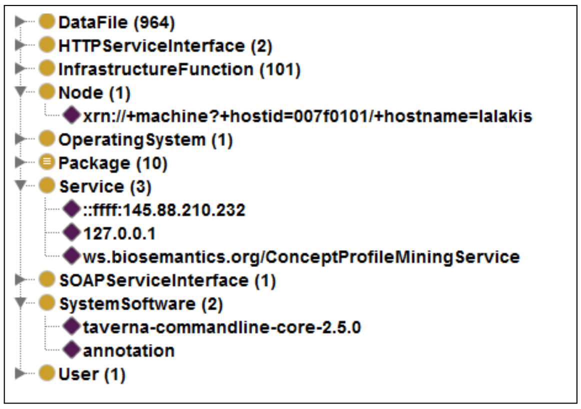
- **R2.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_3* must be identical
- **R5.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_2* must be identical
- **R15.1** The output *concatenated* of the workflow step *Merge_String_List_to_a_String_4* must be identical

R18 Execution duration of each of the workflow steps shall be similar.

Redeployment environment

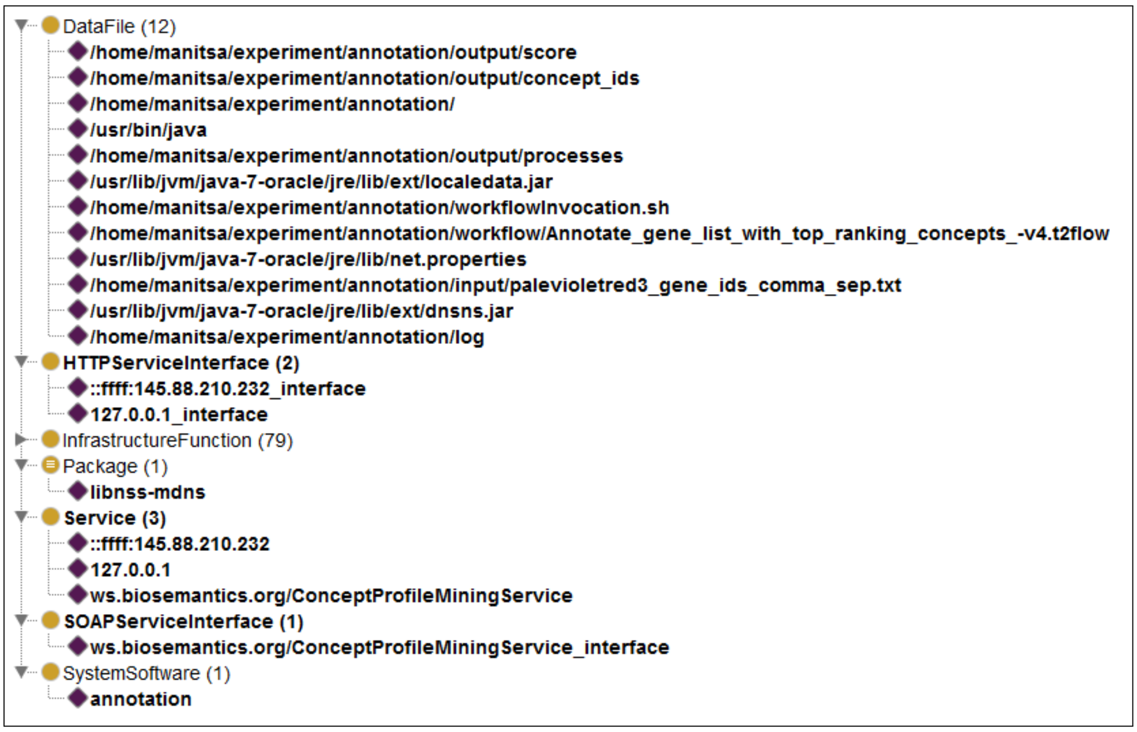
We redeploy the workflow into an existing Linux environment that is used to run other workflows. The original platform was *Ubuntu 12.04.5 LTS*, while the redeployment platform is *Linux Mint 17 Qiana*.

We re-execute and monitor with the PMF the workflow that uses the original WSDL web service. Then we compare its context model with the context model of the original execution and discover that, apart from the different operating system, also the installed Java version differs, that is, the original system used *oracle java 7*, while the redeployment system uses *open jdk 8*. Despite these discrepancies, the validation performed in the *Validate workflow* step of the VFramework confirms that the workflow re-execution is valid, that is, it produces the same results as the original workflow.



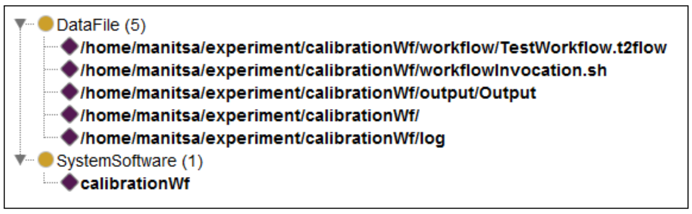
```
DataFile (964)
HTTPServiceInterface (2)
InfrastructureFunction (101)
Node (1)
    xrn://+machine?+hostid=007f0101/+hostname=lalakis
OperatingSystem (1)
Package (10)
Service (3)
    ::ffff:145.88.210.232
    127.0.0.1
    ws.biosemantics.org/ConceptProfileMiningService
SOAPServiceInterface (1)
SystemSoftware (2)
    taverna-commandline-core-2.5.0
    annotation
User (1)
```

(a) Original context model.



```
DataFile (12)
    /home/manitsa/experiment/annotation/output/score
    /home/manitsa/experiment/annotation/output/concept_ids
    /home/manitsa/experiment/annotation/
    /usr/bin/java
    /home/manitsa/experiment/annotation/output/processes
    /usr/lib/jvm/java-7-oracle/jre/lib/ext/localedata.jar
    /home/manitsa/experiment/annotation/workflowInvocation.sh
    /home/manitsa/experiment/annotation/workflow/Annotate_gene_list_with_top_ranking_concepts_-v4.t2flow
    /usr/lib/jvm/java-7-oracle/jre/lib/net.properties
    /home/manitsa/experiment/annotation/input/palevioletred3_gene_ids_comma_sep.txt
    /usr/lib/jvm/java-7-oracle/jre/lib/ext/dnsns.jar
    /home/manitsa/experiment/annotation/log
HTTPServiceInterface (2)
    ::ffff:145.88.210.232_interface
    127.0.0.1_interface
InfrastructureFunction (79)
Package (1)
    libnss-mdns
Service (3)
    ::ffff:145.88.210.232
    127.0.0.1
    ws.biosemantics.org/ConceptProfileMiningService
SOAPServiceInterface (1)
    ws.biosemantics.org/ConceptProfileMiningService_interface
SystemSoftware (1)
    annotation
```

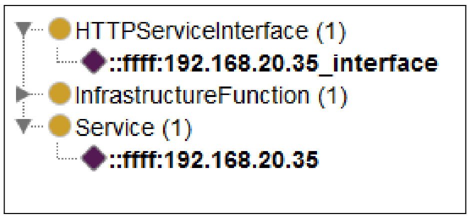
(b) The *annotate workflow* dependencies.



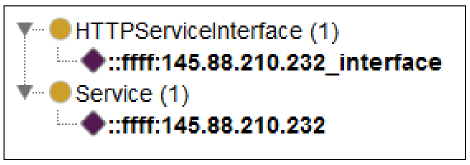
```
DataFile (5)
    /home/manitsa/experiment/calibrationWf/workflow/TestWorkflow.t2flow
    /home/manitsa/experiment/calibrationWf/workflowInvocation.sh
    /home/manitsa/experiment/calibrationWf/output/Output
    /home/manitsa/experiment/calibrationWf/
    /home/manitsa/experiment/calibrationWf/log
SystemSoftware (1)
    calibrationWf
```

(c) The calibration workflow dependencies, not used in the *annotate workflow*.

Figure 7.8: Noise reduction process for the *annotate workflow*.


HTTPServiceInterface (1)
::ffff:192.168.20.35_interface
InfrastructureFunction (1)
Service (1)
::ffff:192.168.20.35

(a) Added elements.


HTTPServiceInterface (1)
::ffff:145.88.210.232_interface
Service (1)
::ffff:145.88.210.232

(b) Deleted elements.

Figure 7.9: Comparison of context models of the re-executed *annotate workflow* that used original WSDL web service and its mock-up.

Knowing that the platform differences do not affect workflow repeatability, we test the impact of the web service mock-up on the workflow re-execution. For this reason we run a mock-up application on a machine that is within the same sub-network. The application uses the intercepted original requests and responses. To make the workflow connect to the mock-up, we edit the DNS configuration in the redeployment machine. Thus, we redirect the traffic to and from the *ws.biosemantics.org* host to the machine on which the mock-up is deployed. No changes in the workflow file itself are necessary.

Then we re-execute the workflow and monitor it with the PMF. The mock-up log shows that the workflow made six GET requests asking for the WSDL scheme of the web service. It also made six POST requests containing SOAP requests for data. There were no requests from the workflow for which the mock-up did not have a response.

We validate the workflow re-execution that used the mock-up and all requirements are fulfilled. Thus the mock-up is a valid strategy to be used for workflows that use web services to ensure repeatable conditions.

Last, but not least, we compare the context models of both re-executions to identify impact of the mock-up on the changes in the workflow context model. Figure 7.9 presents results of this comparison. We can see that only the IP address of the host providing the external service has changed. There were no other differences (We do not take into account the *Infrastructure Functions*, because they correspond to the operating system processes which order is highly non-deterministic - see Section 6.1.2).

7.3.2 Rshell workflow

We first present the use case description and then describe how we completed the steps of the VFramework in the original and the redeployment environment.

Use case description

Figure 7.10 presents the *rshell workflow*. It finds the overlap between two datasets that contain genomic information (e.g. gene id, chromosome name, gene start, gene end), and computes basic statistics. The workflow returns rows of the first file that overlap with

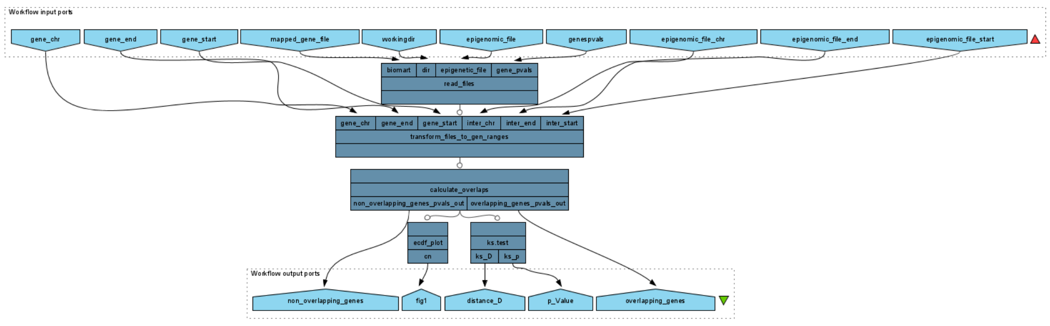


Figure 7.10: The *rshell workflow*.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dio: <http://timbus.teco.edu/ontologies/DIO.owl#>

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\\(.*\\)"))
}
```

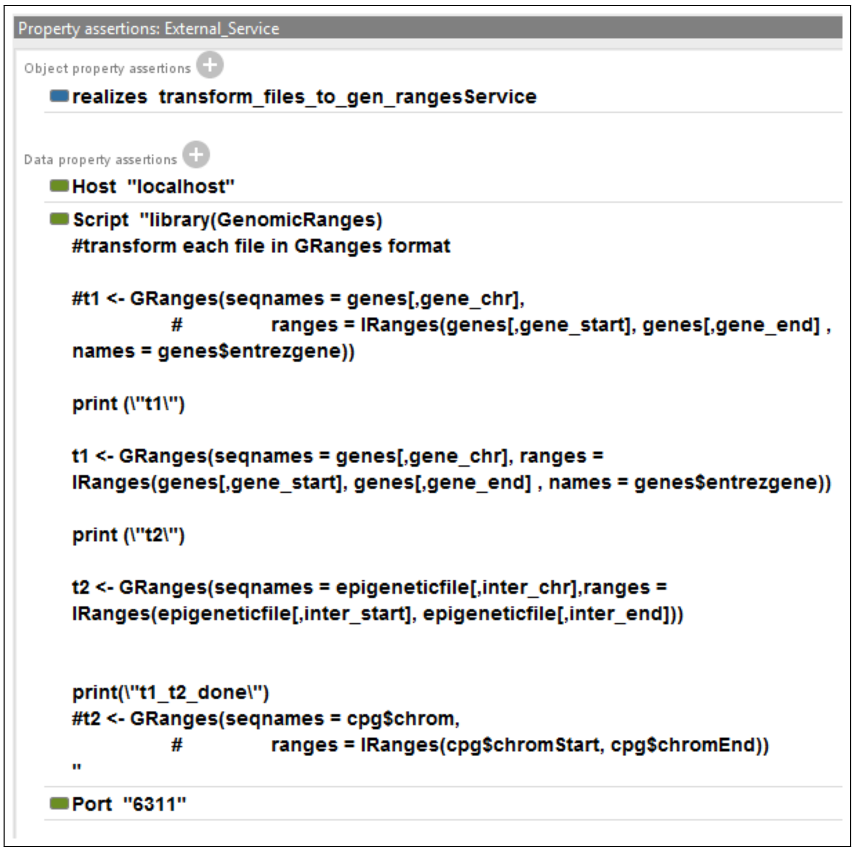
Listing 7.1: SPARQL query checking whether the R scripts load specific libraries.

the second file. A Kolmogorov-Smirnov test is applied between the list of genes that overlap with the epigenetic file, and the one that does not, to test for differences between the p-value distributions of genes that overlap with the ones that are not overlapping.

Original environment

Similar to the other use cases, in the *Run static analysis* step of the VFramework we convert the workflow definition file into the core of the context model and use SPARQL queries to analyse the workflow. We identify that the workflow consists of five steps that are *Rshell services*. These services communicate with an external host to execute R scripts.

Figure 7.11 depicts an excerpt of the context model describing the external service realizing the *transform_file_to_gen_range_Service* workflow step. We can see that the service is run on a localhost which means that the R server is started on the same machine in which the workflow executes. The workflow engine and the service communicate on port 6311. The context model also contains the R script that is executed at the given step in the R server.


Property assertions: External_Service
Object property assertions
Data property assertions

realizes transform_files_to_gen_rangesService

Host "localhost"

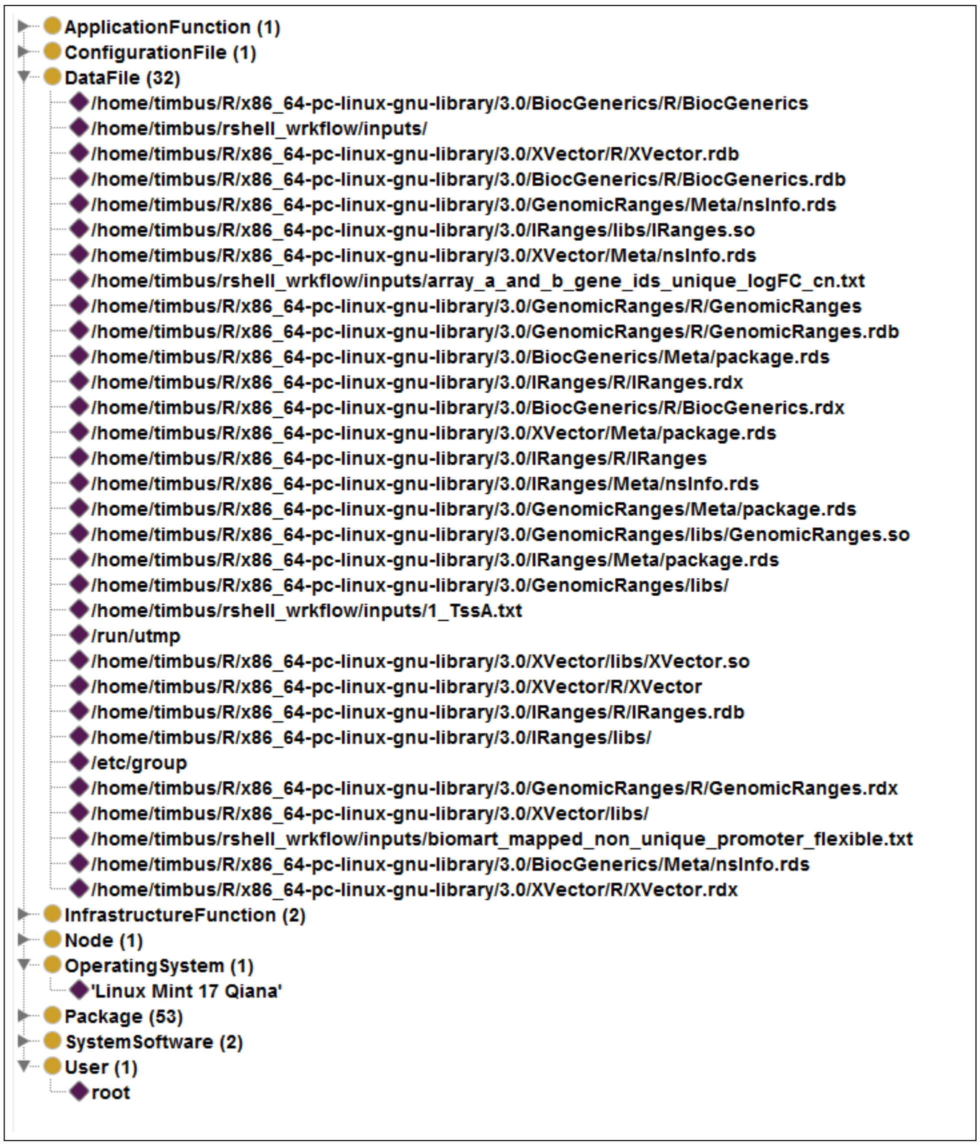
```
Script "library(GenomicRanges)
#transform each file in GRanges format

#t1 <- GRanges(seqnames = genes[,gene_chr],
        #           ranges = IRanges(genes[,gene_start], genes[,gene_end] ,
names = genes$entrezgene))

print (\"t1\")

t1 <- GRanges(seqnames = genes[,gene_chr], ranges =
IRanges(genes[,gene_start], genes[,gene_end] , names = genes$entrezgene))

print (\"t2\")

t2 <- GRanges(seqnames = epigeneticfile[,inter_chr],ranges =
IRanges(epigeneticfile[,inter_start], epigeneticfile[,inter_end]))


print(\"t1_t2_done\")
#t2 <- GRanges(seqnames = cpg$chrom,
        #           ranges = IRanges(cpg$chromStart, cpg$chromEnd))
"
```

Port "6311"

Figure 7.11: Analysis of the static part of the context model.

The R scripts can require specific libraries to be installed in the R environment. In the analysed case, the script requires the *Genomic Ranges* library. Therefore, to re-execute the workflow, it is not sufficient to start the R server in parallel to the workflow engine, but also to load appropriate libraries. Using the SPARQL queries we can identify in the context model which steps require additional libraries. Listing 7.1 presents an example of a query that lists external services, their addresses, and ports that have scripts which require additional libraries.

We use this information to configure the redeployment environment in a proper way before the workflow is re-executed. We also consider creating a mock-up and for that purpose we capture and analyse the communication between the workflow engine and the R server. This is a *telnet* communication in which the scripts contained in the workflow are executed on the server. The server responds with standard messages confirming execution of consecutive scripts, but does not send back any data. Only the final result of running the sequence of scripts, that is the PNG image, is sent back by the server. Hence, we conclude that the service is stateful (depends on the sequence of previously executed commands) and requires more complex implementation than a lookup table used for the



```
ApplicationFunction (1)
ConfigurationFile (1)
DataFile (32)
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/BiocGenerics/R/BiocGenerics
    /home/timbus/rshell_wrkflow/inputs/
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/R/XVector.rdb
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/BiocGenerics/R/BiocGenerics.rdb
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/Meta/nsInfo.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/libs/IRanges.so
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/Meta/nsInfo.rds
    /home/timbus/rshell_wrkflow/inputs/array_a_and_b_gene_ids_unique_logFC_cn.txt
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/R/GenomicRanges
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/R/GenomicRanges.rdb
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/BiocGenerics/Meta/package.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/R/IRanges.rdx
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/BiocGenerics/R/BiocGenerics.rdx
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/Meta/package.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/R/IRanges
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/Meta/nsInfo.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/Meta/package.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/libs/GenomicRanges.so
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/Meta/package.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/libs/
    /home/timbus/rshell_wrkflow/inputs/1_TssA.txt
    /run/utmp
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/libs/XVector.so
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/R/XVector
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/R/IRanges.rdb
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/IRanges/libs/
    /etc/group
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/GenomicRanges/R/GenomicRanges.rdx
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/libs/
    /home/timbus/rshell_wrkflow/inputs/biomart_mapped_non_unique_promoter_flexible.txt
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/BiocGenerics/Meta/nsInfo.rds
    /home/timbus/R/x86_64-pc-linux-gnu-library/3.0/XVector/R/XVector.rdx
InfrastructureFunction (2)
Node (1)
OperatingSystem (1)
    'Linux Mint 17 Qiana'
Package (53)
SystemSoftware (2)
User (1)
    root
```

Figure 7.12: Dependencies of the external R server.

stateless web services (the response does not depend on the history of previous requests), like used in the other use cases. For this reason, the effort required to configure the R server that includes already identified libraries is lower than implementation of a mock-up of a stateful service.

In the *Run dynamic analysis* step of the VFramework we run the PMF and create a dependency report that is depicted in Figure 7.14. We can see that the workflow does neither shell calls nor does it require any specific Debian packages. The report shows that the workflow communicates with the 127.0.0.1 address which is the localhost. The report does not depict any dependencies of the R server, and especially none of the additional libraries needed by the R server, because the R server and its libraries are beyond the workflow's boundary.

However, it is also possible to detect dependencies of the server by monitoring the R server with the PMF. Figure 7.12 shows the dependencies of the R server that were detected by the PMF which attached to system process of the R server and monitored it for the period of time when the workflow was executing and exchanging data with the server. In the figure we can see that *Genomic Ranges* library, as identified by the static analysis of the workflow model, was loaded by the server. We can see that also other libraries like *IRanges* or *XVector* are used during the computation. PMF also detected 53 Debian packages and identified local paths of the files from which the scripts loaded data that was sent to the server. This shows that in settings when we have access to the environment of the external service and can monitor service execution, we are able to provide a modular documentation of both local and external dependencies.

The *sensor data analysis workflow*, described in Section 7.2, also uses R scripts to perform tasks. Contrary to the *rshell workflow*, it uses shell calls to execute R scripts. Thus, the execution of the script is a local dependency of the workflow that is captured by the PMF when monitoring a workflow execution. In the sensor data analysis use case we identify that the R script requires the *xtables* library to be loaded. We do this by analysing the workflow context model created during the dynamic analysis. This shows that the static analysis of the workflow as well as the dynamic analysis of workflow execution complement each other and allow for detection of R script dependencies regardless of the way the R scripts are run.

In the *Define validation metrics* step of the VFramework we generate the following validation requirements:

R1 The workflow step *calculate_overlaps* must have identical outputs.

R1.1 The output *non_overlapping_genes_pvals_out* of the workflow step *calculate_overlaps* must be identical.

R1.2 The *output overlapping_genes_pvals_out* of the workflow step *calculate_overlaps* must be identical.

R2 The workflow step *ks.test* must have identical outputs.

R2.1 The output *ks_p* of the workflow step *ks.test* must be identical.

R2.2 The output *ks_D* of the workflow step *ks.test* must be identical.

R3 (The workflow step*read_files* must have identical outputs.)

This workflow step does not have outputs. This step cannot be validated.

R4 (The workflow step *transform_files_to_gen_ranges* must have identical outputs.)

This workflow step does not have outputs. This step cannot be validated.

R5 The workflow step *ecdf_plot* must have identical outputs.
- **R5.1** The output *cn* of the workflow step *ecdf_plot* must be identical.

R6 The inputs of the workflow are the same.
- **R6.1** The input *gene_end* is identical.
- **R6.2** The input *gene_start* is identical.
- **R6.3** The input *mapped_gene_file* is identical.
- **R6.4** The input *workingdir* is identical.
- **R6.5** The input *epigenomic_file* is identical.
- **R6.6** The input *genespvals* is identical.
- **R6.7** The input *epigenominc_file_chr* is identical.
- **R6.8** The input *epigenomic_file_end* is identical.
- **R6.9** The input *epigenomic_file_start* is identical.

R7 The outputs of the workflow are the same.
- **R2.1** The output *ks_p* of the workflow step *ks.test* must be identical.
- **R2.2** The output *ks_D* of the workflow step *ks.test* must be identical.
- **R5.1** The output *cn* of the workflow step *ecdf_plot* must be identical.

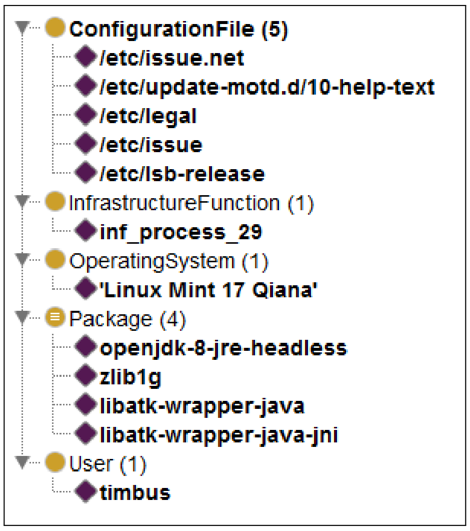
R8 Execution duration of each of the workflow steps shall be similar.

We depicted the requirements R3 and R4 in this list, but they did not get generated. This is because the workflow steps *read_files* and *transform_file_to_gen_ranges* do not have outputs. Hence, we are not able to validate them in the same way as we did it for the other workflows. If we look again at the script presented in Figure 7.11 we can observe the reason for that. The R scripts use their own variables to which they load data. These variables are persisted on the server and are available during the whole workflow execution. In this way the workflow steps exchange the data outside of the workflow engine! For this reason we are not able to validate these workflow steps automatically.

Table 7.9 depicts metrics that we generated for the identified requirements. Each sub-requirement of requirements R1, R2, and R6 is validated using *String Equality* metric. For the requirement R5.1 we use the *PNG Format Based Metrics*. The non-functional requirement R14 is validated in the same way as in the other use cases.

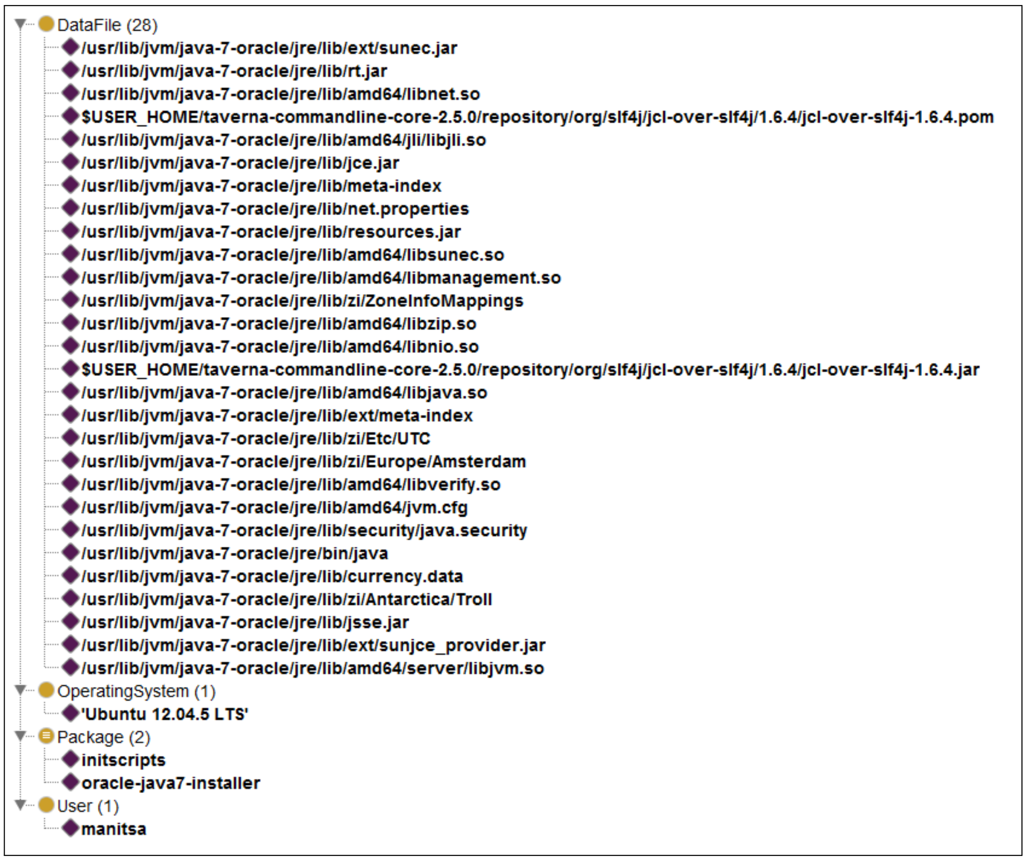
Redeployment environment

The original and the redeployment platforms are the same for this workflow as for the *annotate workflow* described in Section 7.3.1. In the redeployment platform we had to additionally install the R server and the *Genomic Ranges* library. We started the R server on port 6311 as identified in the *Run static analysis* step of the VFramework and



```
ConfigurationFile (5)
    /etc/issue.net
    /etc/update-motd.d/10-help-text
    /etc/legal
    /etc/issue
    /etc/lsb-release
InfrastructureFunction (1)
    inf_process_29
OperatingSystem (1)
    'Linux Mint 17 Qiana'
Package (4)
    openjdk-8-jre-headless
    zlib1g
    libatk-wrapper-java
    libatk-wrapper-java-jni
User (1)
    timbus
```

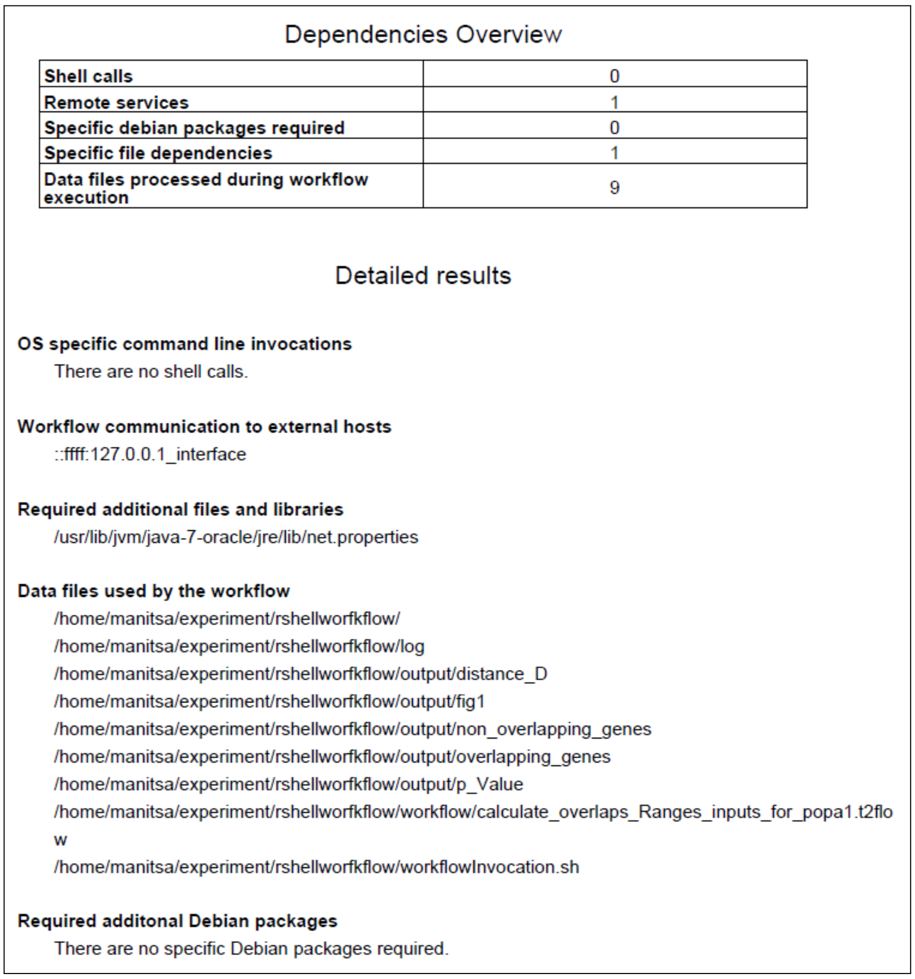
(a) Added elements.



```
DataFile (28)
    /usr/lib/jvm/java-7-oracle/jre/lib/ext/sunec.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/rt.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libnet.so
    $USER_HOME/taverna-commandline-core-2.5.0/repository/org/slf4j/jcl-over-slf4j/1.6.4/jcl-over-slf4j-1.6.4.pom
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/jli/libjli.so
    /usr/lib/jvm/java-7-oracle/jre/lib/jce.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/meta-index
    /usr/lib/jvm/java-7-oracle/jre/lib/net.properties
    /usr/lib/jvm/java-7-oracle/jre/lib/resources.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libsunec.so
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libmanagement.so
    /usr/lib/jvm/java-7-oracle/jre/lib/zi/ZoneInfoMappings
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libzip.so
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libnio.so
    $USER_HOME/taverna-commandline-core-2.5.0/repository/org/slf4j/jcl-over-slf4j/1.6.4/jcl-over-slf4j-1.6.4.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libjava.so
    /usr/lib/jvm/java-7-oracle/jre/lib/ext/meta-index
    /usr/lib/jvm/java-7-oracle/jre/lib/zi/Etc/UTC
    /usr/lib/jvm/java-7-oracle/jre/lib/zi/Europe/Amsterdam
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libverify.so
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/jvm.cfg
    /usr/lib/jvm/java-7-oracle/jre/lib/security/java.security
    /usr/lib/jvm/java-7-oracle/jre/bin/java
    /usr/lib/jvm/java-7-oracle/jre/lib/currency.data
    /usr/lib/jvm/java-7-oracle/jre/lib/zi/Antarctica/Troll
    /usr/lib/jvm/java-7-oracle/jre/lib/jsse.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/ext/sunjce_provider.jar
    /usr/lib/jvm/java-7-oracle/jre/lib/amd64/server/libjvm.so
OperatingSystem (1)
    'Ubuntu 12.04.5 LTS'
Package (2)
    initscripts
    oracle-java7-installer
User (1)
    manitsa
```

(b) Deleted elements.

Figure 7.13: Comparison of context models for the re-executed and the original *rshell workflow*.


Dependencies Overview

Shell calls	0
Remote services	1
Specific debian packages required	0
Specific file dependencies	1
Data files processed during workflow execution	9

Detailed results

OS specific command line invocations
There are no shell calls.

Workflow communication to external hosts
::ffff:127.0.0.1_interface

Required additional files and libraries
/usr/lib/jvm/java-7-oracle/jre/lib/net.properties

Data files used by the workflow
/home/manitsa/experiment/rshellworfkflow/
/home/manitsa/experiment/rshellworfkflow/log
/home/manitsa/experiment/rshellworfkflow/output/distance_D
/home/manitsa/experiment/rshellworfkflow/output/fig1
/home/manitsa/experiment/rshellworfkflow/output/non_overlapping_genes
/home/manitsa/experiment/rshellworfkflow/output/overlapping_genes
/home/manitsa/experiment/rshellworfkflow/output/p_Value
/home/manitsa/experiment/rshellworfkflow/workflow/calculate_overlaps_Ranges_inputs_for_popa1.t2flo
w
/home/manitsa/experiment/rshellworfkflow/workflowInvocation.sh

Required additonal Debian packages
There are no specific Debian packages required.

Figure 7.14: Dependency report for the *rshell workflow*.

Table 7.9: Mapping of requirements and metrics to their target values for the *rshell workflow*.

Requirement	**Metric**	**Format**	**Target Value**	**Tolerance**
R1.*	String Equality	Plain Text	0	0
R2.*	String Equality	Plain Text	0	0
R5.1	Image Fingerprint Equality Image Resolution Equality Absolute Error Count	PNG	0	0
R6.*	String Equality	Plain Text	0	0
R14	Execution Duration Ratio	N/A	1	30%

then executed the workflow. We monitored its execution and compared it to the original context model.

Figure 7.13 depicts the discrepancies between the context models of the original and the re-executed workflow. Figure 7.13a shows elements that were not present in the original execution, but are used during the execution. Figure 7.13b presents the opposite, that is the elements that were used in the original execution, but are not used any more. Based on this comparison we can see that:

1. Different users ran the workflow.
2. Different operating systems were used.
3. The original execution used *java oracle 7* while the re-execution used *openjdk 8*.
4. The original execution used *initscripts* package which is not used in the re-execution.
5. The re-execution uses two *libatk* packages and one *zlib1g* package.
6. The original execution loaded *jcl-overl-slf4j-1.6.4.jar* which is not loaded in the re-execution.
7. The workflow does not access files that are not included in the provenance traces.

The differences 4 and 5 can be attributed to the fact that the operating systems use different graphical user interfaces and the fact that different versions of Java from different providers was used. The difference 6 could have an impact on the workflow processing result like it was the case for the weather workflow (see Section 6.1). This library might have been placed in the Taverna directory by the researcher to run other workflows.

We validate the workflow and all requirements are fulfilled. Hence the above discrepancies did not have an impact on the workflow re-execution.

7.3.3 Ruby workflow

We first present the use case description and then describe how we completed the steps of the VFramework in the original and the redeployment environment.

Use case description

Figure 7.15 presents the *ruby workflow*. The workflow uses Ruby[6] scripts to create *nanopublications*[7] for a gene list that is associated to Huntington's Disease. The nanopublications allow disseminating individual data as independent publications and can be uniquely identified and attributed to its author. They can be serialized to RDF.

[6]https://www.ruby-lang.org
[7]http://nanopub.org

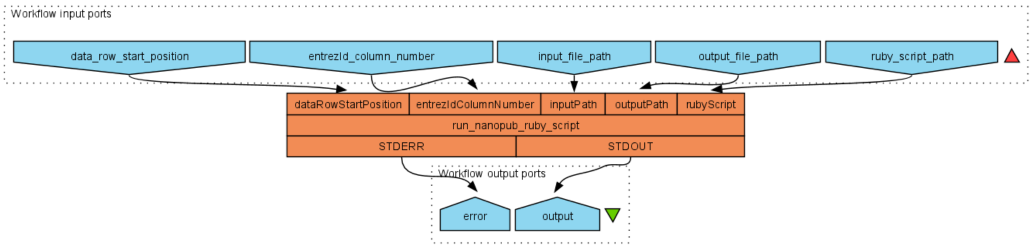


Figure 7.15: The *ruby workflow*.

Original environment

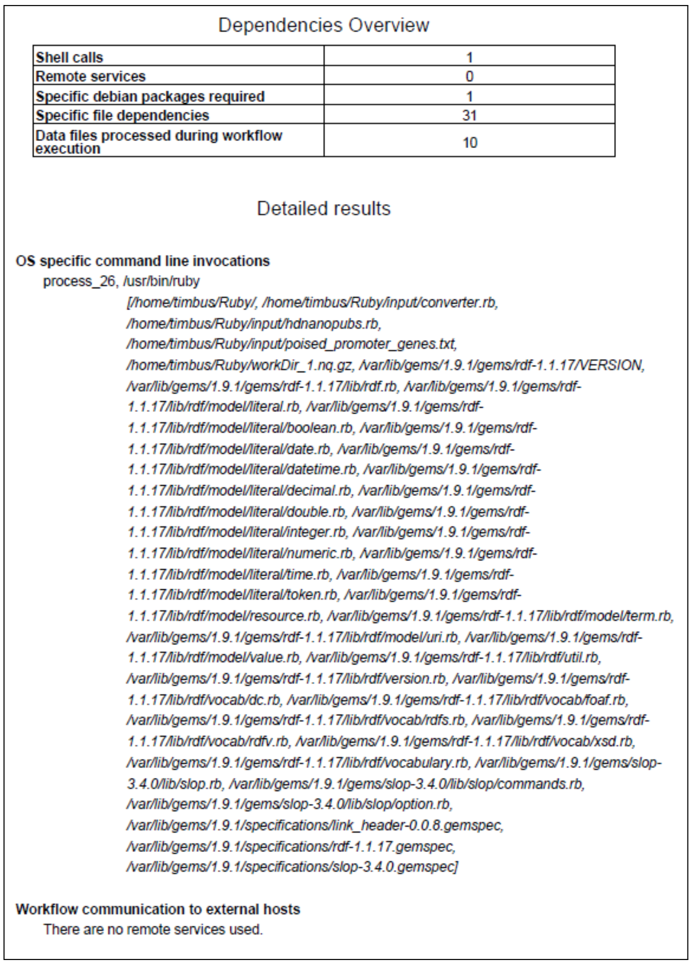
We run the static analysis of the workflow and identify one step that is a *Tool Invocation Service*. This step contains a command executing a Ruby script that is provided at the input of the workflow.

In the *Run dynamic analysis* step of the VFramework we monitor the workflow execution and create a dependency report that is depicted in Figure 7.16. The report shows that:

- The workflow makes one shell call which uses */usr/bin/ruby*.
- This shell call requires additional files that are additional Ruby libraries, so-called gems, which are *rdf-1.1.17* and *slop-3.4.0*.
- The gems were installed for the version 1.9.1 of Ruby.
- The workflow has no external communications.
- The workflow requires one Debian package.
- The workflow creates *workdDir_1.nq.gz* file that is not part of the provenance traces.

The workflow owners explained that the *workdDir_1.nq.gz* file is the actual output of the workflow execution. It was produced during workflow execution by running the Ruby script. The file is not included in the Taverna provenance traces, because only the values exchanged between the workflow steps through their outputs and inputs are captured. Hence to validate workflows that perform tasks using shell calls we have two options:

1. Run the PMF in a mode that creates the context model only and manually select additional files that were detected by it.
2. Run the PMF in a mode that creates the context model and sources all identified files automatically.



Dependencies Overview

Shell calls	1
Remote services	0
Specific debian packages required	1
Specific file dependencies	31
Data files processed during workflow execution	10

Detailed results

OS specific command line invocations

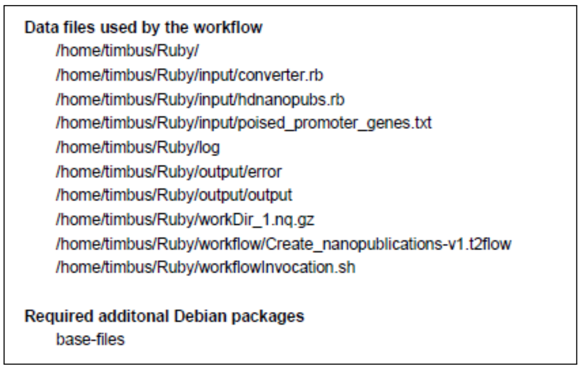
process_26, /usr/bin/ruby

*[/home/timbus/Ruby/, /home/timbus/Ruby/input/converter.rb,
/home/timbus/Ruby/input/hdnanopubs.rb,
/home/timbus/Ruby/input/poised_promoter_genes.txt,
/home/timbus/Ruby/workDir_1.nq.gz, /var/lib/gems/1.9.1/gems/rdf-1.1.17/VERSION,
/var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/boolean.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/date.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/datetime.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/decimal.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/double.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/integer.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/numeric.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/time.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/literal/token.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/resource.rb, /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/term.rb,
/var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/uri.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/model/value.rb, /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/util.rb,
/var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/version.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/vocab/dc.rb, /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/foaf.rb,
/var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/rdfs.rb, /var/lib/gems/1.9.1/gems/rdf-
1.1.17/lib/rdf/vocab/rdfv.rb, /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/xsd.rb,
/var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocabulary.rb, /var/lib/gems/1.9.1/gems/slop-
3.4.0/lib/slop.rb, /var/lib/gems/1.9.1/gems/slop-3.4.0/lib/slop/commands.rb,
/var/lib/gems/1.9.1/gems/slop-3.4.0/lib/slop/option.rb,
/var/lib/gems/1.9.1/specifications/link_header-0.0.8.gemspec,
/var/lib/gems/1.9.1/specifications/rdf-1.1.17.gemspec,
/var/lib/gems/1.9.1/specifications/slop-3.4.0.gemspec]*

Workflow communication to external hosts

There are no remote services used.

(a) Page 1/2



Data files used by the workflow

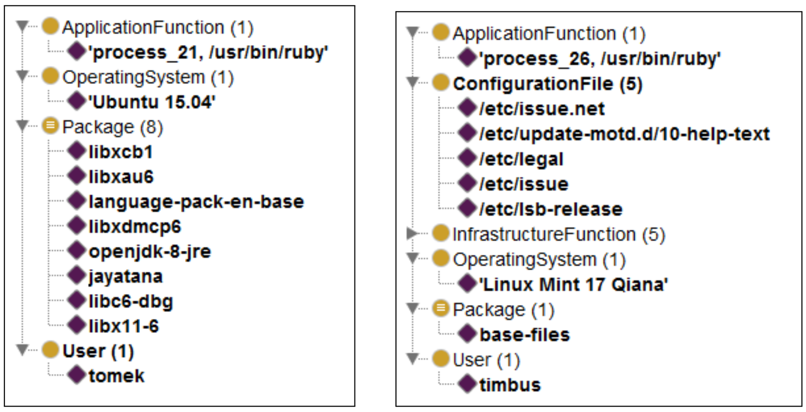
/home/timbus/Ruby/
/home/timbus/Ruby/input/converter.rb
/home/timbus/Ruby/input/hdnanopubs.rb
/home/timbus/Ruby/input/poised_promoter_genes.txt
/home/timbus/Ruby/log
/home/timbus/Ruby/output/error
/home/timbus/Ruby/output/output
/home/timbus/Ruby/workDir_1.nq.gz
/home/timbus/Ruby/workflow/Create_nanopublications-v1.t2flow
/home/timbus/Ruby/workflowInvocation.sh

Required additonal Debian packages

base-files

(b) Page 2/2

Figure 7.16: Dependency report for the *ruby workflow*.



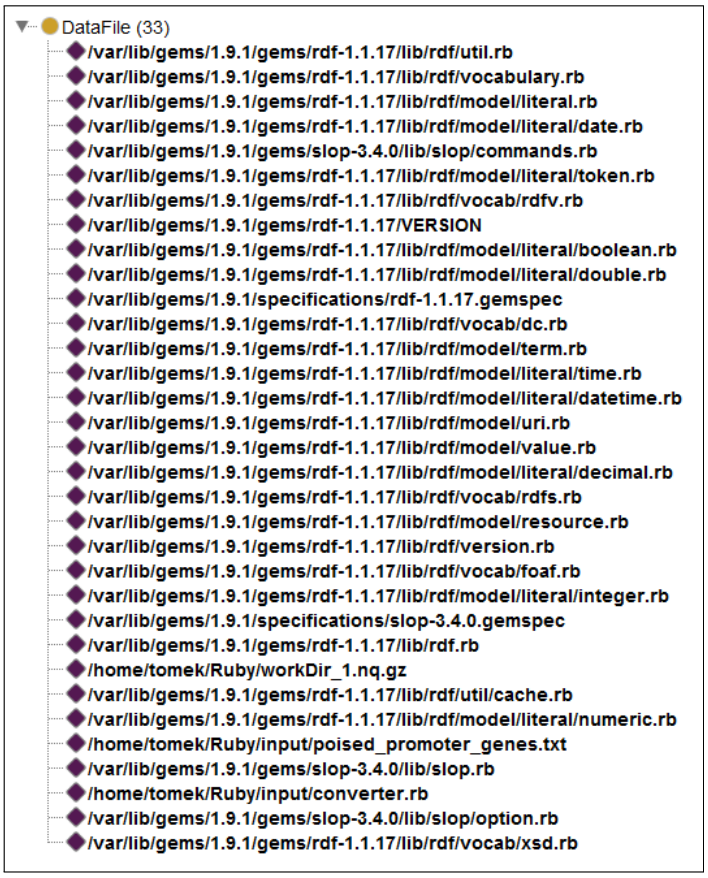
- ApplicationFunction (1)
  - 'process_21, /usr/bin/ruby'
- OperatingSystem (1)
  - 'Ubuntu 15.04'
- Package (8)
  - libxcb1
  - libxau6
  - language-pack-en-base
  - libxdmcp6
  - openjdk-8-jre
  - jayatana
  - libc6-dbg
  - libx11-6
- User (1)
  - tomek

(a) Added elements.

- ApplicationFunction (1)
  - 'process_26, /usr/bin/ruby'
- ConfigurationFile (5)
  - /etc/issue.net
  - /etc/update-motd.d/10-help-text
  - /etc/legal
  - /etc/issue
  - /etc/lsb-release
- InfrastructureFunction (5)
- OperatingSystem (1)
  - 'Linux Mint 17 Qiana'
- Package (1)
  - base-files
- User (1)
  - timbus

(b) Deleted elements.

Figure 7.17: Comparison of context models of the original and the redeployed *ruby workflow*.



- DataFile (33)
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/util.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocabulary.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/date.rb
  - /var/lib/gems/1.9.1/gems/slop-3.4.0/lib/slop/commands.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/token.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/rdfv.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/VERSION
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/boolean.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/double.rb
  - /var/lib/gems/1.9.1/specifications/rdf-1.1.17.gemspec
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/dc.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/term.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/time.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/datetime.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/uri.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/value.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/decimal.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/rdfs.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/resource.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/version.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/foaf.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/integer.rb
  - /var/lib/gems/1.9.1/specifications/slop-3.4.0.gemspec
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf.rb
  - /home/tomek/Ruby/workDir_1.nq.gz
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/util/cache.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/model/literal/numeric.rb
  - /home/tomek/Ruby/input/poised_promoter_genes.txt
  - /var/lib/gems/1.9.1/gems/slop-3.4.0/lib/slop.rb
  - /home/tomek/Ruby/input/converter.rb
  - /var/lib/gems/1.9.1/gems/slop-3.4.0/lib/slop/option.rb
  - /var/lib/gems/1.9.1/gems/rdf-1.1.17/lib/rdf/vocab/xsd.rb

Figure 7.18: Comparison of context models of the re-execution missing dependencies and the correct re-execution.

In this experiment we choose the first option, that is, we ask the workflow owners to share the folder in which the workflow was executed that also includes the detected file. This coresponds to a practice used for creating Research Objects [BZG$^+$15] that are not only used for sharing workflows but also input, output, and other data used in experiments.

Redeployment environment

We redeploy the workflow in a Linux environment that again differs in the version of the system. We move the workflow from the *Linux Mint 17 Qiana* to *Linux Ubuntu 15.04*. We install Ruby in version 1.9.1 and the required gems as described by the dependency report. Then we re-execute the workflow and monitor its execution with the PMF. We compare the context models of the re-executed and the original workflow. The results of the comparison are depicted in Figure 7.17.

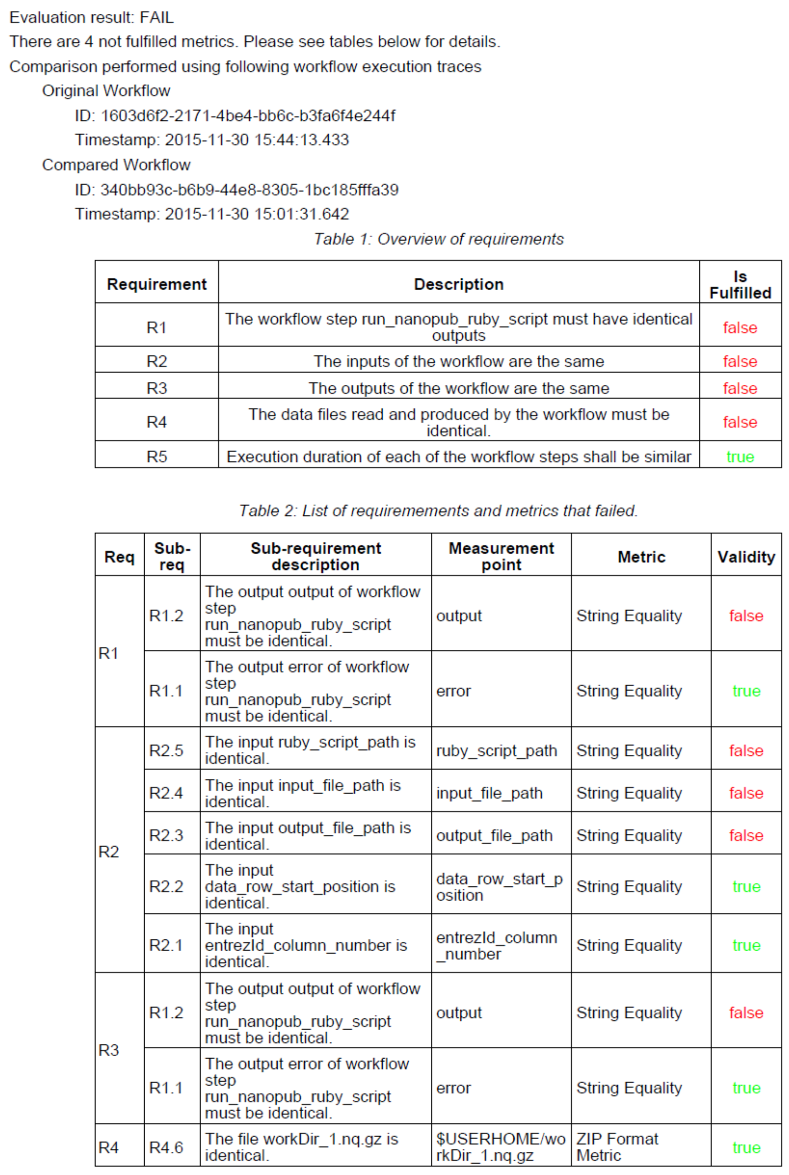
We can see that the redeployment environment differs from the original:

- Different operating system is used.
- Eight new Debian packages are used.
- One Debian package is not used.
- Different users ran the workflow.

These changes, similarly to the other use cases, can be attributed to the differences between the operating systems and exact versions of Java. To confirm these observations we validate the workflow in the next step.

Figure 7.19 presents the validation report created in the *Validate workflow* step of the VFramework. Only the execution duration requirement was fulfilled, while four other requirements failed:

- Requirements R1 and R3 failed because the output *output* was not identical. This was because the output *output* contains the standard output of the Ruby script invocation, which in turn contains timestamp and execution duration. Hence, these values are always different for every re-execution.
- Requirement R2 failed because the inputs specifying paths of the input parameters were not identical. The paths were different because different users were running the workflow in their home directories.
- Requirement R4 failed for the same reason as R1 and R3, because the output *output* was different. However, this requirement was measured differently than other two requirements, because not the provenance traces were used, but the actual file produced by the workflow that was detected by the PMF.



Evaluation result: FAIL

There are 4 not fulfilled metrics. Please see tables below for details.

Comparison performed using following workflow execution traces

- Original Workflow
  - ID: 1603d6f2-2171-4be4-bb6c-b3fa6f4e244f
  - Timestamp: 2015-11-30 15:44:13.433
- Compared Workflow
  - ID: 340bb93c-b6b9-44e8-8305-1bc185fffa39
  - Timestamp: 2015-11-30 15:01:31.642

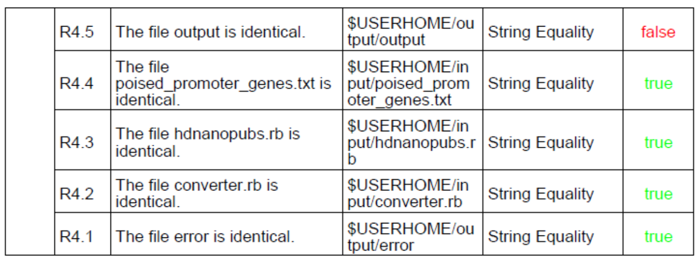
Table 1: Overview of requirements

Requirement	Description	Is Fulfilled
R1	The workflow step run_nanopub_ruby_script must have identical outputs	false
R2	The inputs of the workflow are the same	false
R3	The outputs of the workflow are the same	false
R4	The data files read and produced by the workflow must be identical.	false
R5	Execution duration of each of the workflow steps shall be similar	true

Table 2: List of requirements and metrics that failed.

Req	Sub-req	Sub-requirement description	Measurement point	Metric	Validity
R1	R1.2	The output output of workflow step run_nanopub_ruby_script must be identical.	output	String Equality	false
	R1.1	The output error of workflow step run_nanopub_ruby_script must be identical.	error	String Equality	true
R2	R2.5	The input ruby_script_path is identical.	ruby_script_path	String Equality	false
	R2.4	The input input_file_path is identical.	input_file_path	String Equality	false
	R2.3	The input output_file_path is identical.	output_file_path	String Equality	false
	R2.2	The input data_row_start_position is identical.	data_row_start_position	String Equality	true
	R2.1	The input entrezId_column_number is identical.	entrezId_column_number	String Equality	true
R3	R1.2	The output output of workflow step run_nanopub_ruby_script must be identical.	output	String Equality	false
	R1.1	The output error of workflow step run_nanopub_ruby_script must be identical.	error	String Equality	true
R4	R4.6	The file workDir_1.nq.gz is identical.	$USERHOME/workDir_1.nq.gz	ZIP Format Metric	true

(a) Page 1/2



	R4.5	The file output is identical.	$USERHOME/output/output	String Equality	false
	R4.4	The file poised_promoter_genes.txt is identical.	$USERHOME/input/poised_promoter_genes.txt	String Equality	true
	R4.3	The file hdnanopubs.rb is identical.	$USERHOME/input/hdnanopubs.rb	String Equality	true
	R4.2	The file converter.rb is identical.	$USERHOME/input/converter.rb	String Equality	true
	R4.1	The file error is identical.	$USERHOME/output/error	String Equality	true

(b) Page 2/2

Figure 7.19: Validation report for the *ruby workflow*.

None of the requirements is violated due to the discrepancies between the environments. The requirements R2.3, R2.4, and R2.5 that failed have actually no impact on the validity of the workflow re-execution, because they require identity of paths of the input files. The requirements R4.2, R4.3, and R4.4 are more important, because they validate the actual data used to run the workflow. They are fulfilled, which means that the data provided to the workflow is identical, but simply located in a different directory. Similarly, the real output of the workflow is the *workdDir_1.nq.gz* file, the contents of which are identical.

The data files that were detected by the PMF and grouped under requirement R4 were validated also using the VPlanComparator. Hence we ran the file format characterization tool to identify their format and on that basis choose suitable metrics. In the given example, the *workDir_1.nq.gz* is a compressed library for which comparison of hashes shows discrepancies. This is due to a different header of the file. The actual data inside the library is the same and for this reason we used the *ZIP Format Metric* that aggregates results of metrics used to validate files contained within the library.

The requirements validating the data used by the workflow, as well as the data created by the workflow are fulfilled, hence we state that the workflow re-execution is replicable.

As we already observed in the evaluation, workflows that make shell calls depend on other software libraries and packages installed in the environment. Any of their dependencies missing can stop the workflow from executing and that is why the *Verify environment* step of the VFramework is dedicated to verification of workflow dependencies. We make one more experiment in which we evaluate how the VFramework detects workflow re-executions that fail to re-execute in a new environment and how it helps in identifying the cause of failure. For this reason, we modify the environment used in the previous redeployment by removing *gem* packages required by the *ruby workflow* to execute. We monitor the re-execution of the workflow with the PMF and then compare its context model with the context model of the previous valid re-execution.

The re-executed workflow produces errors which we observe in its error output. The error message informs that some libraries required by the workflow are missing and this information already explains the reason for the workflow execution to fail. However, the implementations of other workflows using shell calls can be different, for example, the standard error output of the script does not necessarily have to be linked to the workflow output, or the message may not be informative to the user, because of the wrong exception handling mechanism implemented by an external library used. For this reason the analysis of context models is a universal method for detecting missing dependencies.

Figure 7.18 depicts the elements that are missing from the environment and which were present in the original re-execution. We can see that the *slop* and *rdf* gems were uninstalled. Furthermore, we can see that the *workDir_1.nq.gz* file was not created and the *converter.rb* script was not even loaded. Hence we know that the workflow missed gems and the error occurred in the other script loaded by the workflow (the workflow shell call uses two Ruby scripts). The comparison of context models correctly identified the reason for the workflow execution to break.

7.4 Summary

In this chapter we evaluated the proposed VFramework on five Taverna workflows from three different domains, namely: music classification, sensor data analysis, and clinical medical research.

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.

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). We thus evaluated in what way the VFramework can be applied to verify and validate not only re-executions in an exactly identical environment, but also in a similar or considerably different environment.

The workflows analysed differ in a number of additional dependencies required to execute them. We showed that 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.

Although the changes, or especially unavailability, of the external web services are a serious threat to the replicability of experiments, we used a universal method of creating mock-ups for web services that can be used to replace the original web-services. Thus we reduced workflow dependability on external services and enabled replication of experiments that use the original data. Furthermore, we demonstrated that the proposed implementation does not require re-engineering of workflows and do not alter the workflow execution environment.

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.

In the sensor data analysis use case the contact with the workflow owner was necessary, to resolve one of the problems (modified Latex style file). This is often not possible in a real-world setting. Furthermore, the syntax of the shell calls made by this workflow was incompatible with a new environment (Linux to Windows redeployment) and changes in the workflow file were necessary.

The two other workflows used the R language to execute scripts. The *sensor data analysis workflow* used shell calls to call local instance of R, while the *rshell workflow* used an R server that was accessed through a service. For both cases 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. We were not able to verify the server configuration using the dynamic workflow monitoring, because the server was not within the workflow boundaries. However, we were able to monitor in parallel the server to identify its dependencies that are loaded when workflow communicates with it.

Hence, special attention is needed for workflows with platform specific dependencies, because their analysis is more complex and requires more effort during verification as opposed to workflows with no dependencies, for which the framework can be almost fully automated.

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 describing the environment gets published together with the provenance traces and data files accessed by the workflow, and second, in which the identified files and packages are additionally automatically sourced. 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. For workflows with few or 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, that are not available in public repositories.

Furthermore, in case of workflows heavily depending on shell calls to complete their steps, we recommend that the workflow owners make a test redeployment of their workflow into a clean machine before they publish their workflow. Thus they ensure that the information contained in the context model is sufficient for the redeployment. The current implementation of tools allows detecting all dependencies, but cannot identify ways in which these have to be installed. For this reason, explicit information from the workflow owner would make the porting process easier.

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, in the Ruby use case 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*[8] project that analyses how to build sources on different machines to obtain identical Debian packages. The proposed solution is to use an environment variable which enables setting a default timestamp value. Another suggested solution is to completely remove timestamps.

The evaluation also confirmed that validation of workflow executions using only the provenance traces is not sufficient. This is because workflows using shell calls can read and write files without workflow engine mediation. Thus, these files are not included in the provenance traces. Such files are detected during workflow monitoring using the PMF. We demonstrated in the *ruby workflow*, that the actual inputs and outputs of the workflow contain only paths to the data provided to the workflow and information whether the workflow execution was correct, but do not contain the data. Validation of such data will fail for all re-executions, because the paths in which the data is located can be different, and the standard output also contains execution timestamps. For this reason, we focused on requirements that validated the files detected by the PMF as those which were accessed by the workflow during execution. Thus we validated the actual inputs and outputs of the workflow that have scientific value to the researcher.

Furthermore, not all of the Rshell services have outputs that can be validated. For such steps we cannot generate requirements. Such steps exchange data through global variables created in a remote service. These variables are persisted on the server and are available during the workflow execution. In this way the workflow steps exchange the data outside of the workflow engine. Thus verification and validation of such workflows resembles black box testing techniques, in which only the inputs and outputs of the workflow can be validated, but not the intermediate steps.

Last, but not least, during the simulation of changes 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.

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

CHAPTER 8

Conclusions and Outlook

Researchers make scientific breakthroughs by processing, linking and exchanging data. They use special software tools and processing workflows that allow them to link, transform, visualise and interpret the data. This methodology is also referred as the fourth paradigm of science, namely e-Science.

Workflow engines were proposed to hide the complexity of the underlying infrastructure and to improve verification and reuse of scientific experiments by sharing workflows. They enable researchers to graphically represent their experiments in form of workflows that can be built using pre-defined elements and do not require special software engineering skills.

In spite of such a standardisation there are still workflows that cannot be re-executed. The reasons for workflows to break range from lack of input data and additional software tools, to the unavailability of external web services. Even if the workflows can be re-executed there is no well-established approach that enables trustworthy and evidence based verification and validation of their re-executions which would in turn enable researchers re-running the workflows to state that the experiment is replicable.

For this reason in Chapters 4, 5, and 6 we presented the VFramework that checks whether the re-executed workflow produces the same result in the same way as the original workflow did. It consists of five steps that are performed in both the original and the redeployment environment. In a systematic way it collects evidence that creates confidence that the workflow re-execution is replicable.

The VFramework uses the context model that contains a comprehensive description of a workflow which 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 also showed how the automated context model analysis enables identification of workflow boundaries and in what way the external services being outside these boundaries can be monitored to create evidence needed for re-executing and validating workflows.

In Chapter 5 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. Furthermore, 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 contains also 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. Thus we evade double definitions of the same concepts and ensure coherence of the vocabulary.

In Chapter 6 we described the *Verify environment* and *Validate workflow* steps of the VFramework that are performed when the workflow is re-executed in a new environment. To complete them we monitored the re-execution in the new environment and compared it to the information collected in the original environment.

In Chapter 7 we evaluated the proposed VFramework on five Taverna workflows from three different domains, namely: music classification, sensor data analysis, and clinical medical research. Based on this evaluation we identified limitations of the proposed solution.

8.1 Research questions revisited

In Chapter 1 we defined a number of research questions to be answered in this dissertation. We revisit these questions and show how they were answered.

RQ1 How can we verify whether the workflow re-executed in a way that complies with the original execution?

The workflow executions are operating system processes that use system functions to access resources or to execute code. The traces of operating system processes provide a complete description of workflow dependencies. For this reason, the models of workflow executions that are build upon these traces enable verification of workflow re-executions. In the related work, described in Chapter 2, we described a tool that we use for capturing of workflow dependencies that are used by the VFramework, described in Chapters 3, 4, and 6, to redeploy the workflows in the new environment and to verify their re-executions.

To ensure repeatable conditions any changes in the services on which the workflows depend must be avoided. These changes can stem from the fact that the services can be non-deterministic by their nature, or simply because their provider changed their implementation or functionality. For web service dependent workflows we presented the Web Service Monitoring Framework in Chapter 4 that detects whether the web service is deterministic and based on the evidence collected, it allows for creating mock-ups of stateless web services that are used to replace the original service and thus ensure repeatable conditions for verification of workflow re-executions.

We also analysed in the evaluation, presented in Chapter 7, how to verify re-executions in environments that by assumption differ in their configuration and architecture. We showed that the verification is strongly coupled with the validation, and these two processes must be performed together. Thus we can verify whether the differences between systems detected by the verification have real impact on the outcome of workflows computation by validating them. The evaluation showed that the full identity of environments is not required for workflows to produce the right results in the right way.

RQ2 How can we validate whether the workflow re-execution produced the correct results?

In Chapter 5 we analysed sample workflows, their architecture, available data and also considered motivations of scientists re-executing the workflows. On that basis we devised a way of generating validation requirements for workflows. The requirements deal with the correctness of data produced at multiple stages of workflow execution. They are quantified using metrics from a controlled vocabulary, which is a part of an ontology for documenting validation requirements, and are automatically selected based on the identified format of data that was captured. Thus we ensure that data is compared in a right way and the whole process scales up.

Furthermore, in Chapter 7 we confirmed by applying our solution to the evaluated set of workflows that it is not sufficient to validate workflows using only provenance traces, because these may be incomplete, especially when the workflows complete tasks outside of the workflow engine, for example by calling local tools installed in a system that process data. We also observed that there are workflows in which deviations between executions are acceptable from the scientific point of view, especially when environments in which the workflows are executed differ in their configuration and these differences do not affect the core idea of the experiment. In such cases the scientists must take individual decisions based on an indicative reports depicting a comprehensive set of requirements and metrics used in the validation.

RQ3 How can we perform systematic and repeatable verification and validation of workflow re-executions?

We described the VFramework in Chapters 3, 4, and 6 that consists of five steps that are performed in both the original and the redeployment environment. In a systematic way it creates evidence describing workflow execution and its environment that is used for verification and validation in the redeployment platform without the necessity of accessing simultaneously the original environment. For this reason the VFramework not only can be applied in a short term settings but also in typical digital preservation settings.

The VFramework uses the context model that contains comprehensive description of the workflow which integrates information from different sources describing among others the static workflow model, as well as its dependencies detected during dynamic analysis of its execution. It contains also information on external services that were accessed. For this reason it is used to describe environments in which the workflow executes. By comparing context models of workflow executions we verify whether the workflow re-execution was obtained in a compliant way.

The automatically collected data that was collected during workflow execution in both environments is used for validation of automatically generated requirements that were quantified using metrics from a controlled vocabulary that we described in Chapter 5. Thus by using ontology models, tools for workflow context capturing, data collection, and validation criteria generation, we devised a systematic and repeatable framework for verification and validation of workflow re-executions.

8.2 Limitations

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 VFramework can be applied to verify and validate not only re-executions in an exactly identical environment, but also in a similar or considerably different environments. Based on this analysis we detected the following limitations of the devised approach.

- **Limited analysability of environments differing in architecture**

  We showed that workflows which had no specific (local) dependencies can be almost automatically verified and validated regardless of the execution platform. For such workflows it would be possible to implement the VFramework as two Taverna workflows, each run in a different environment. However, workflows that make shell calls to complete tasks using software installed in the environment without mediation of the workflow engine cannot be verified and validated automatically when moved across different environments, for example, from Linux to Windows. The verification is not possible because there are no tools that capture workflow execution on windows and produce compatible output. Moreover, the architectural differences between the operating systems make the identification of corresponding software a challenging task. The re-execution may also involve re-engineering of workflows due to the incompatibility of syntax of the shell calls.

The validation of such workflows is also limited, because the provenance traces do not contain data that was accessed and created by the workflow steps making shell calls. The reason again is the fact that such calls bypass the workflow engine. Similar limitations apply to Beanshell scripts that load additional Java libraries.

However these limitations do not apply to redeployments between different distributions of the same operating system family, e.g. Linux, because the tools can monitor the workflow execution and collect all sources of information needed to establish a complete view of the workflow execution.

- **Limited verification of processes executing in parallel**

  The verification is also limited to dependencies that are within the workflow boundary. This is because we monitor workflow execution and detect its dependencies using the tools for tracing operating system processes. Thus only the monitored process and its child processes are analysed and other processes which were started in parallel to the monitored execution are not included in these traces. For this reason we are not able to verify services that run in parallel in the same machine and are accessed by the workflow during its execution. We only validate whether the data exchanged between the workflow and the service running in parallel is correct. However, we shown that parallel monitoring of such services to identify their dependencies is also possible.

- **Monitoring and mocking-up limited to stateless web services**

  For web services to which the workflow connects we used the Web Service Monitoring Framework that captures data exchanged between the workflow and the web service during the original execution and later using this data enables creating mock-ups for web services that can be used to replace the original web-services. We demonstrated that the proposed implementation does not require re-engineering of workflows and does not alter the workflow execution environment and is a valid way of ensuring replicable execution conditions when stateless web services are used.

  However, a crucial requirement is that the web service does not cause any changes on the world outside the system observed. Hence it can be only applied to stateless web services in which actions performed by the service do not depend on the previous series of requests and which do not change the state of the system behind the web service.

- **Limited automation of validation for data with embedded metadata**

  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. We were capable of detecting alterations on different stages of workflow processing and based on these we identified steps in which the workflow re-execution was altered. The application of format specific metrics enabled us to correctly compare the data collected for validation.

  However, in cases when the metadata of the execution is embedded in the data itself this approach failed, for example when the generated PDF report contained its generation date in the first page. In such cases 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.

- **Limited validation of externally exchanged data**

  For workflows using R scripts that execute them on a server running in parallel, we used the static analysis of the workflow model using SPARQL queries. Thus we detected which libraries must be loaded by the server additionally. Using this information we were able to configure the Rshell server running in parallel to the workflow execution.

  However, not all of the Rshell services have outputs that can be validated, because some steps exchange data through global variables created in an external service. These variables are persisted on the server and are available during the workflow execution. In this way the workflow steps exchange the data outside of the workflow engine. Thus verification and validation of such workflows resemble black box testing technique, in which only the inputs and outputs of the workflow can be validated, but not the intermediate steps.

- **Strict automatically generated validation requirements**

  We also observed that criteria stating that all requirements must be identical to recognize the re-execution as repeatable was too strict. Such strict requirements result in false negatives, but not in false positives, that is, in some case use case owners argued that violation of some of the requirements does not have real impact on the replicability of the scientific experiment. 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.

8.3 Recommendations

The general conclusion from the evaluation is that special attention must be paid to workflows with platform specific dependencies, because their analysis is more complex and requires more effort during verification as opposed to workflows with no dependencies, for which the framework can be almost fully automated. We are aware that removing the dependencies is not possible, due to the distributed nature of modern 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. The VFramework and tools described in this dissertation can be used to implement them.

- **Analyse dependencies and evade shell calls**
  Some of the tasks performed by shell calls can be completed using Beanshells that are better portable and explicitly listed in the workflow definition file.
- **Write code that runs on all platforms**
  If a shell call cannot be evaded, then provide its alternative invocation to be picked automatically depending on the execution environment.
- **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 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 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. You can choose from the list of automatically generated requirements.
- **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.

8.4 Future work

Based on the conclusions and recommendations we describe future work that is needed in order to increase replicability of scientific experiments that use workflows.

- **Improvement of workflow provenance traces**
  Currently the monitoring with the PMF is made outside of the workflow engine and only detection of workflow specific dependencies is possible. If the monitoring was integrated into the workflow engine, then more fine grained information on particular dependencies of a single step could be provided and therefore more detailed analysis could be performed. This, in turn, would facilitate verification of reused workflow steps.

  Furthermore, this would solve the problem of provenance traces missing the data for steps that were completed using shell calls. If the monitoring was performed per each thread executing workflow step, then the data would be detected and the provenance traces would be complete.

  The information should include name, path and a hash value of an identified file. Such an extension should have no significant impact on the size of the provenance traces as the size of the context model is much smaller compared to the size of provenance data (cf. Table 7.2 and Table 7.7).

- **Extension of Research Objects with the context model**
  Research Objects are containers for information facilitating understanding of scientific experiments. They contain items like workflow files, data used to run the workflow and provenance traces, as well as scientific publications and slides.

  They miss formally defined descriptions of the environment in which the workflow was executed. We showed in the course of this dissertation that this information is crucial for verification of workflow re-execution. For this reason, we believe that the Research Objects should be extended with the context model that is created applying the first three steps of the VFramework on the original execution. Thus the environment in which the workflow was executed is explicitly documented and similar conditions can be recreated. Furthermore, the validation requirements that were selected by the workflow owner from a set of automatically generated requirements enable validation of workflows re-execution and thus build the trust between the parties.

- **Integration of mock-up capabilities for web services**
  The capturing of data exchanged between a workflow and a web service can be integrated into the workflow engine, because the workflow engines are already a proxy that has access to this data. In the re-execution another built-in mechanism could automatically create a mock-up of external web services using data of the original. The user should be able to choose in the workflow engine whether to run the workflow using the service specified in the workflow file or to use the captured data to mock-up the service. Such automatically collected data could be part of the provenance traces, or an independent entity added to the Research Objects.

- **Resilient web services**
  Web services perform computations outside of a workflow engine and thus limit the

information on how the result was obtained. The implementation of a web service can change and deliver different results than in the original workflow execution. Mechanism similar to web service versioning, but including not only changes to the web service interface, but also to its implementation [MMUR14], is needed. Thus we can verify the web service and check whether it conforms to the version specified in the workflow.

- **Embedded metadata detection**
When comparing collected data we found that some of it contains embedded metadata within the data, for example, the PDF report contained its generation timestamp in the first page. We could not simply extract the text contents from the document, remove all timestamps, and compare the contents, because some of them can be the actual data. For this reason, research on ways of excluding such embedded metadata from comparisons is needed. In some cases changes to the tools and libraries processing data may suffice, while in other cases development of heuristic algorithms may be needed. Such algorithms could check whether the timestamp appears only in a footer of a PDF document, or is within a commented section of an R script, and so on.

- **Further reduction of workflow dependencies**
Although the workflow management systems provide an abstraction layer for workflows that minimise their dependencies, there are still many interactions with the underlying infrastructure that must be captured. Further research should identify how to limit the amount of dependencies that needs to be checked for workflow replicability without limiting the workflow processing capabilities. The research could include investigation of using virtual containers for encapsulating environment configurations that are needed for running workflow steps which have these dependencies. In case of Debian-based systems we showed that an aggregation of multiple files constituting a package improves the analysability of workflow dependencies. Similarly, the dependencies needed for completion of a single workflow step can be aggregated within a container. Containers in contrast with virtualisation do not contain the complete operating system and contain only workflow specific dependencies. Thus the workflow would depend on a specific container version that could be downloaded from a central repository. Currently Taverna allows using beanshells that depend on specific Java libraries that are explicitly listed in the workflow definition file. Similarly, there could be a new kind of workflow step that would specify the container version needed to run the given workflow step. When comparing two workflow executions it would be sufficient to check whether both of them use the same container version in a given workflow step. However, the containers should only be considered as aggregation of dependencies that ease the porting and verification process. They still should be accompanied by a context model describing the dependencies which they aggregate so that the full analysis of workflow dependencies is possible.

APPENDIX A

Context model of the weather workflow

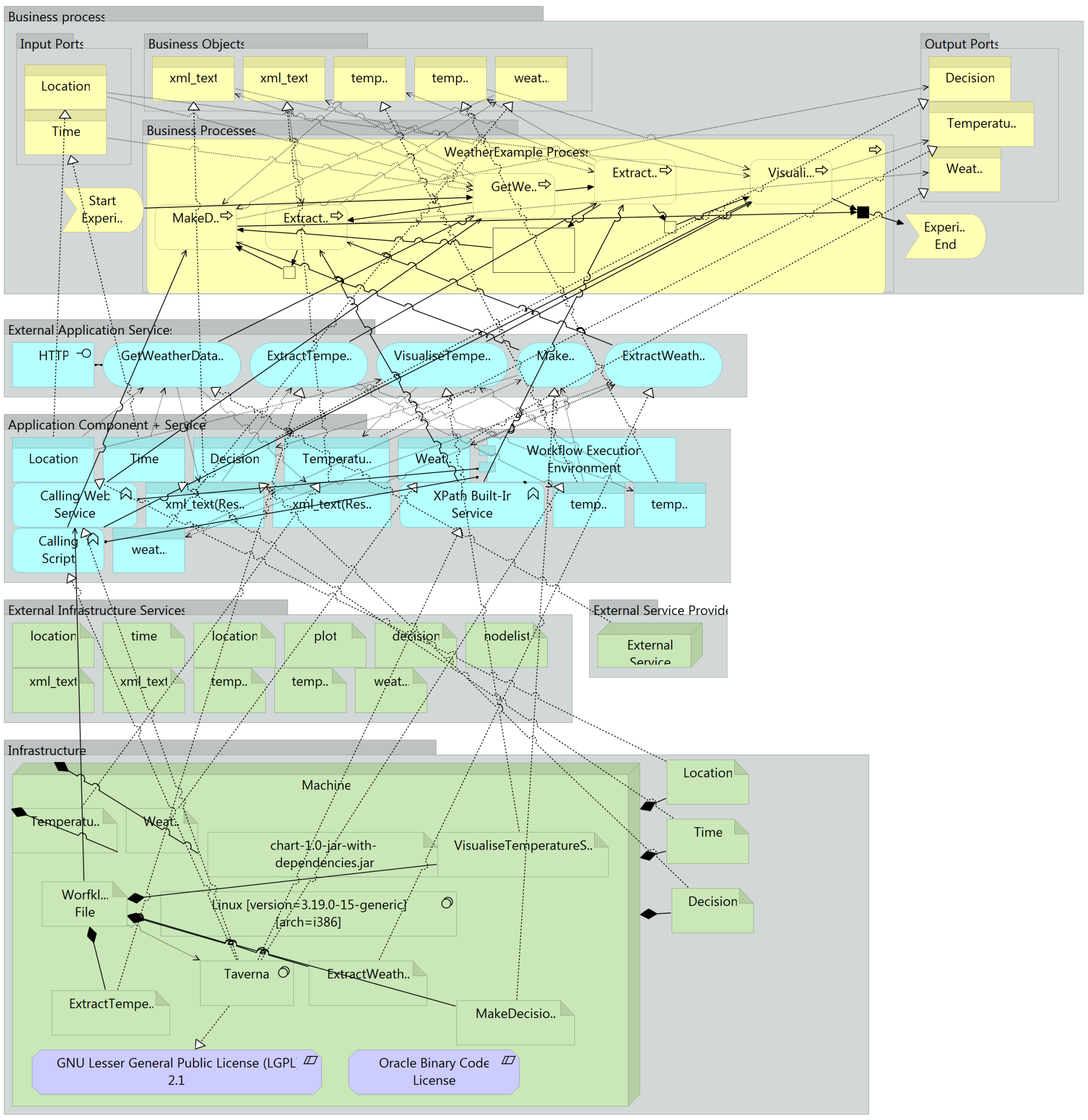


Figure A.1: Complete automatically generated ArchiMate model of the weather workflow.

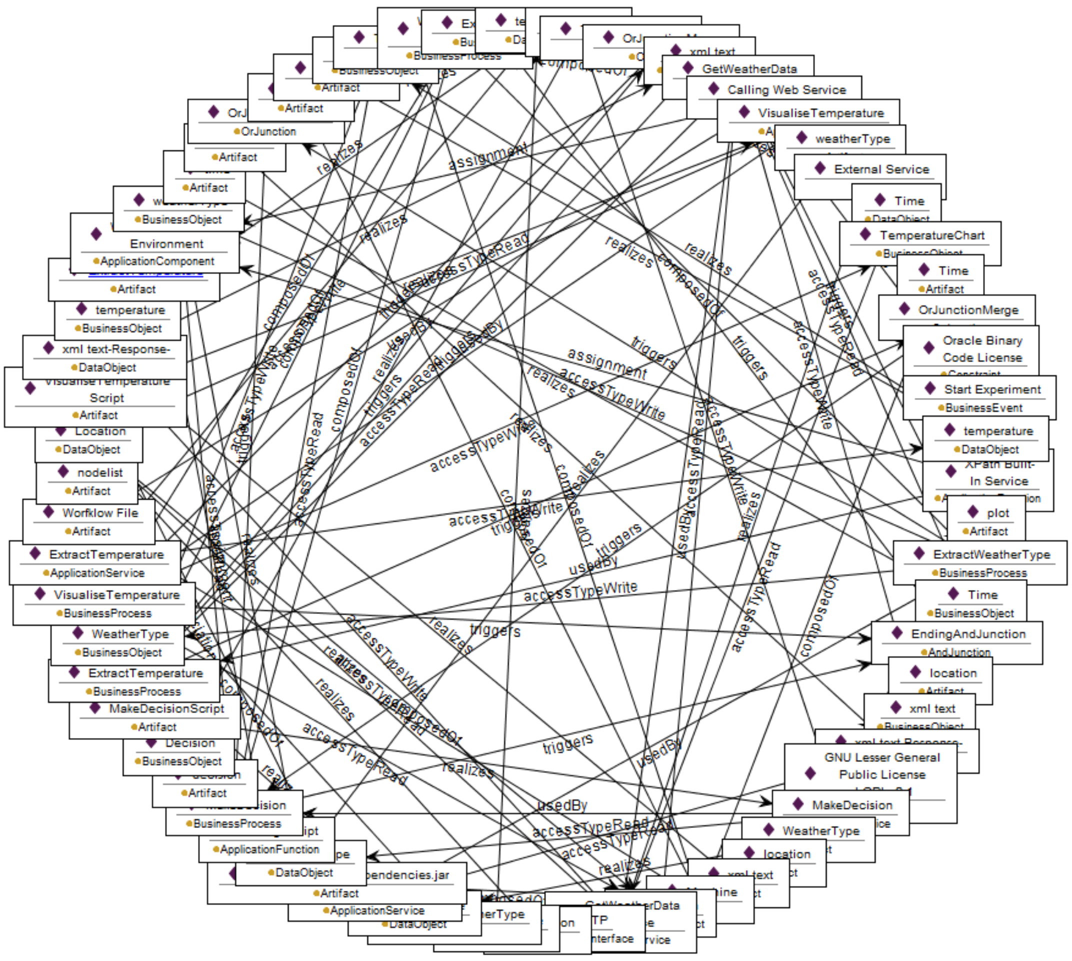


Figure A.2: *Workflow Model Core* of the weather workflow.

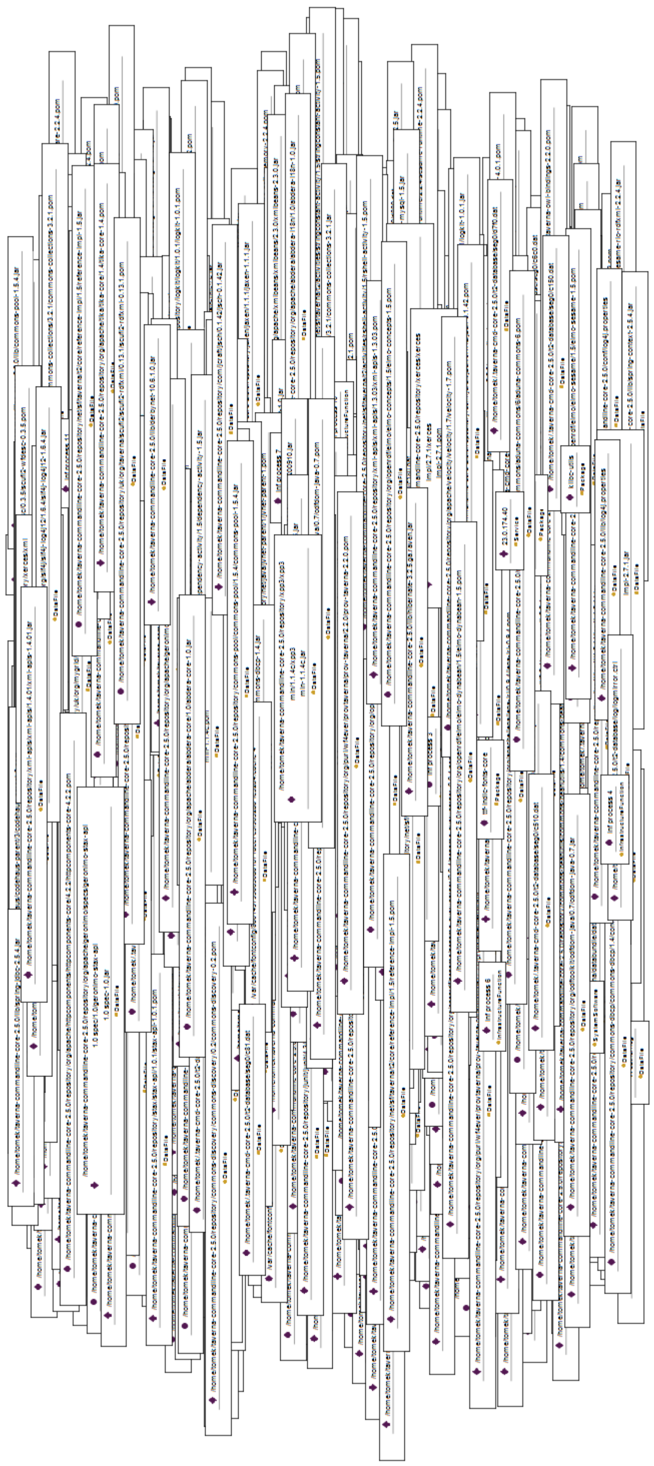
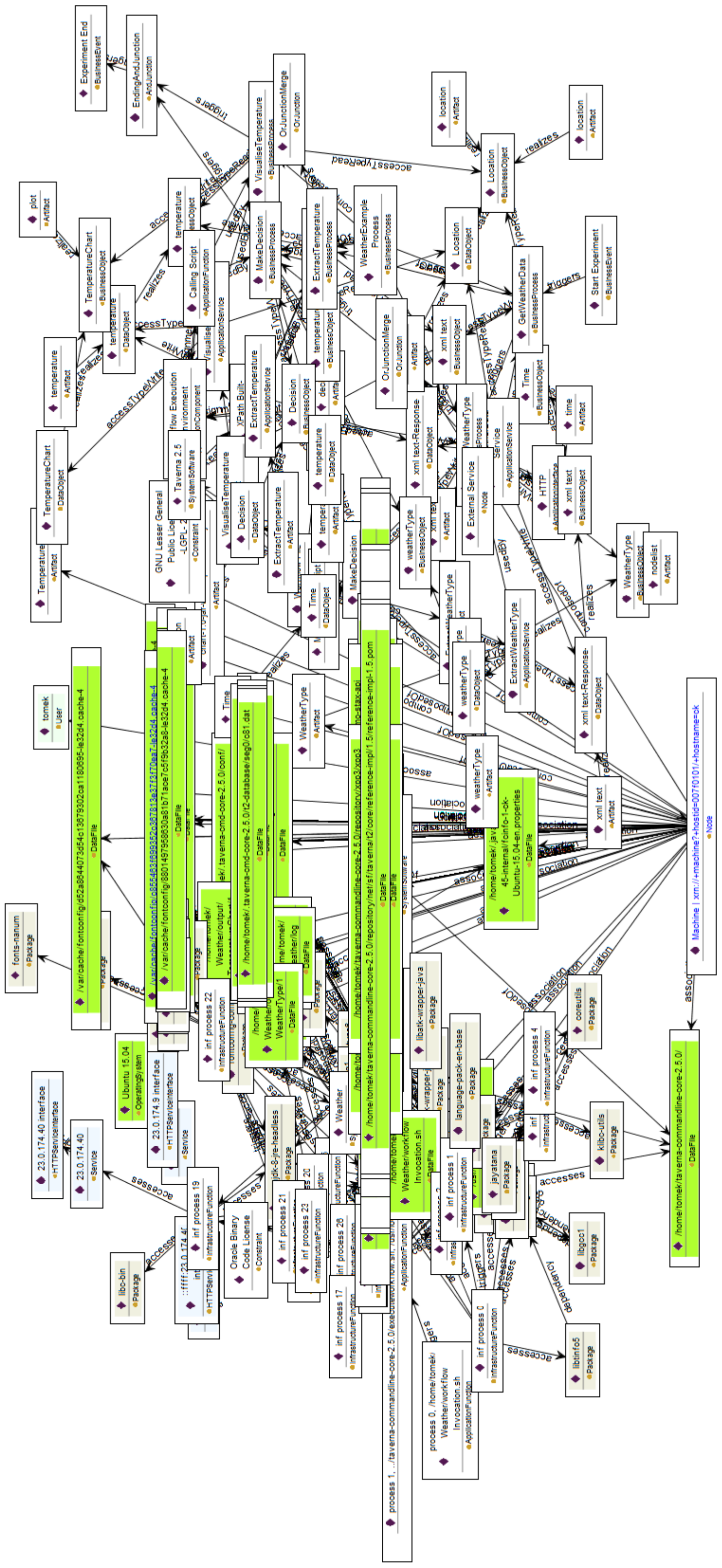


Figure A.3: *Workflow Dependencies* of the weather workflow.



169

Figure A.4: *Workflow Model Core* and *Workflow Dependencies* of the weather workflow.

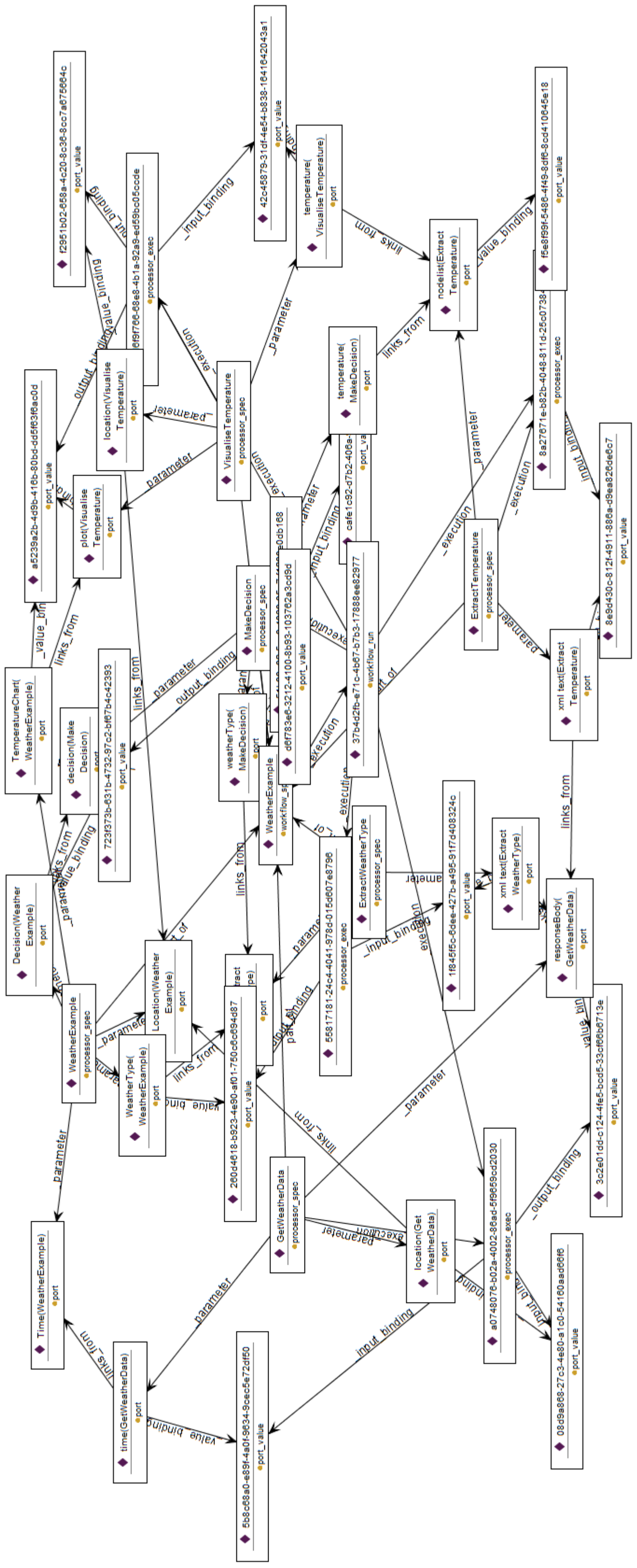


Figure A.5: *Workflow Instance Data* of the weather workflow.

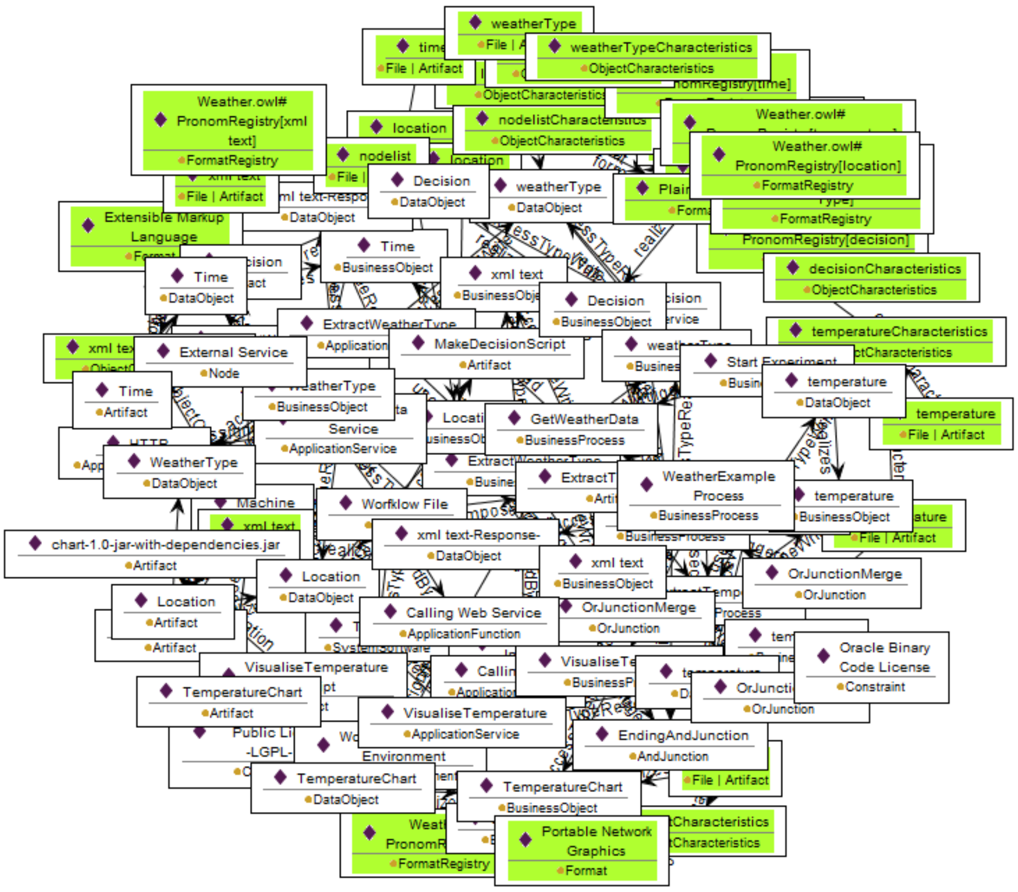


Figure A.6: *Workflow Model Core* and *File Format Specification* of the weather workflow.

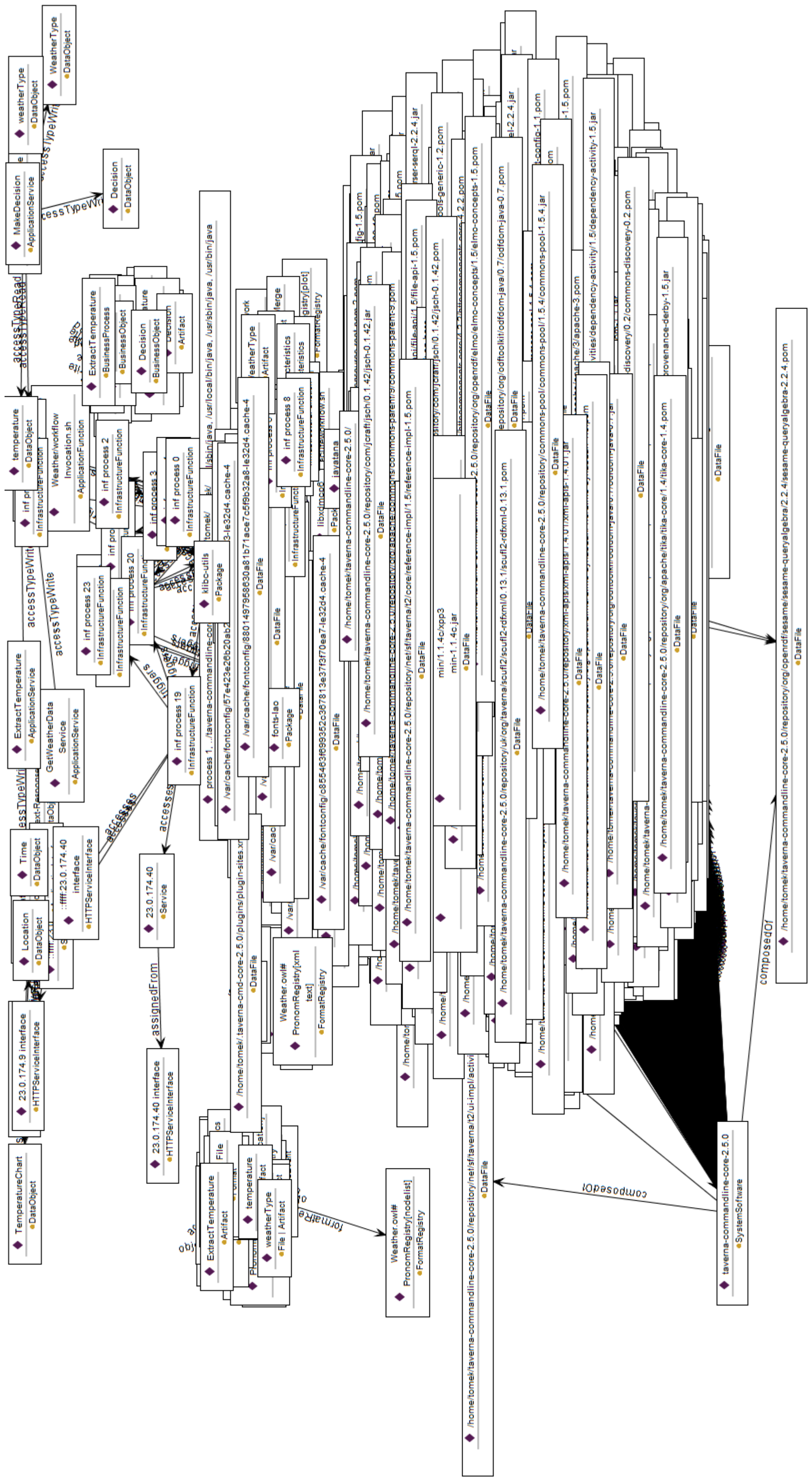


Figure A.7: *Workflow Model Core*, *Workflow Dependencies*, and *File Format Specification* of the weather workflow.

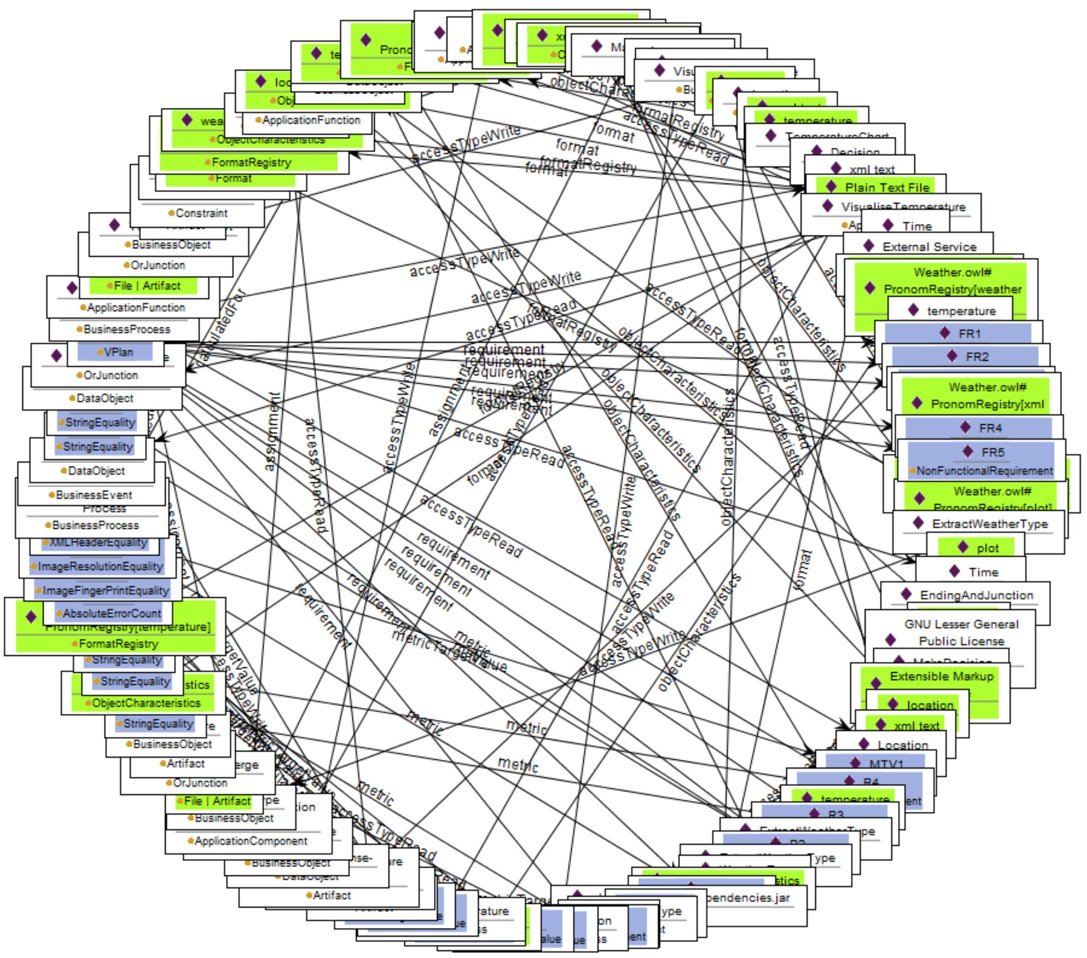


Figure A.8: *Workflow Model Core*, *File Format Specification*, and *Validation Requirements* of the weather workflow.

List of Figures

List of Tables

Bibliography

[Ant15] Goncalo Antunes. *Enterprise Architecture Model Analysis: An Application of Ontologies to the Enterprise Architecture Domain.* PhD thesis, Instituto Superior Técnico, Portugal, 2015.

[BBMP09] Cesare Bartolini, Antonia Bertolino, Eda Marchetti, and Andrea Polini. WS-TAXI: A WSDL-based Testing Tool for Web Services. In *Proceedings of International Conference on Software Testing Verification and Validation*, pages 326–335, April 2009.

[BCG$^+$12] Khalid Belhajjame, Oscar Corcho, Daniel Garijo, Jun Zhao, David Newman, Raúl Palma, Sean Bechhofer, Esteban García, José Manuel Gómez-pérez, Graham Klyne, José Enrique Ruiz, Stian Soil, David De Roure, and Carole 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*, pages 1–12, Hersonissos, Crete, Greece, 2012.

[BE12] C. Glenn Begley and Lee M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, March 2012.

[Bec10] Christoph Becker. *Trustworthy Preservation Planning.* PhD thesis, Vienna University of Technology, Austria, 2010.

[BGSRDR11] Khalid Belhajjame, Carole Goble, Stian Soiland-Reyes, and David De Roure. Fostering Scientific Workflow Preservation through Discovery of Substitute Services. In *Proceedings of 7th IEEE International Conference on E-Science*, pages 97–104, December 2011.

[Bin14] Johannes Binder. Migration of processes from shared to dedicated systems. Master's thesis, Vienna University of Technology, Wien, Austria, 2014.

[BKG$^+$09] Christoph Becker, Hannes Kulovits, Mark Guttenbrunner, Stephan Strodl, Andreas Rauber, and Hans Hofman. Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans. *International Journal on Digital Libraries (IJDL)*, December 2009. http://dx.doi.org/10.1007/s00799-009-0057-1.

[BR11] Christoph Becker and Andreas Rauber. Preservation decisions: Terms and Conditions Apply. Challenges, Misperceptions and Lessons Learned in Preservation Planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, June 2011.

[Bra97] S. Bradner. IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. Technical report, 1997.

[Bro08] Adrian Brown. Automatic format identification using PRONOM and DROID. *Digital Preservation Technical Paper 1*, 2008.

[BSR14] Johannes Binder, Stephan Strodl, and Andreas Rauber. Process migration framework – virtualising and documenting business processes. In *Proceedings of the 18th IEEE International EDOC Conference Workshops and Demonstrations*, pages 398–401, Ulm, Germany, September 2014.

[BZG$^+$15] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, Oscar Corcho, José Manuel Gómez-Pérez, Sean Bechhofer, Graham Klyne, and Carole 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), 2015.

[CCFM11] Tien-Dung Cao, Richard Castanet, Patrick Felix, and Gerardo Morales. Testing of Web Services: Tools and Experiments. In *Proceedings of IEEE Asia-Pacific Services Computing Conference*, pages 78–85, December 2011.

[CFCB10] Tien-Dung Cao, Patrick Felix, Richard Castanet, and Ismail Berrada. Online testing framework for web services. In *Proceedings of Third International Conference on Software Testing, Verification and Validation*, pages 363–372, April 2010.

[CP09] Marco Comuzzi and Barbara Pernici. A framework for QoS-based Web service contracting. *ACM Transactions on the Web*, 3(3):10:1–10:52, July 2009.

[CP14] Christian Collberg and Todd Proebsting. Measuring reproducibility in computer systems research, technical report, 2014.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, August 2003.

[Cur11] Andrew Curry. Rescue of old data offers lesson for particle physicists. *Science*, 331(6018):694–695, 2011.

[Dav12] A.P. Davison. Automated capture of experiment context for easier reproducibility in computational research. *IEEE Computing in Science Engineering*, 14(4):48–56, July 2012.

[DD06] Elena Deza and Michel Marie Deza. *Dictionary of distances.* Elsevier, Amsterdam, 2006.

[Dep07] Department of Defense. DoD Architecture Framework, Version 1.5, 04 2007.

[DRGS09] David De Roure, Carole Goble, and Robert Stevens. The design and realisation of the virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561–567, 2009.

[DRK09] Dimitris Dranidis, Ervin Ramollari, and Dimitros Kourtesis. Run-time Verification of Behavioural Conformance for Conversational Web Services. In *Proceedings of 7th IEEE European Conference on Web Services*, pages 139–147, 2009.

[Dru09] C. Drummond. Replicability is not reproducibility: Nor is it good science. In *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*, 2009.

[DShS$^+$05] Ewa Deelman, Gurmeet Singh, Mei hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[FPD$^+$05] T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wieczorek. Askalon: a grid application development and computing environment. In *Proceedings of 6th IEEE/ACM International Workshop on Grid Computing*, November 2005.

[Gau12] Hugh G. Gauch, Jr. *Scientific method in brief.* Cambridge University Press, 2012.

[GDCM13] T. Goudas, C. Doukas, A. Chatziioannou, and I. Maglogiannis. A collaborative biomedical image-mining framework: Application on the image analysis of microscopic kidney biopsies. *IEEE Journal of Biomedical and Health Informatics*, 17(1):82–91, January 2013.

[GDE$^+$07] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, December 2007.

[GHJ$^+$12] Ed H. B. M. Gronenschild, Petra Habets, Heidi I. L. Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS ONE*, 7(6), June 2012.

[GKS11] Nihita Goel, N.V. Narendra Kumar, and R.K. Shyamasundar. SLA Monitor: A System for Dynamic Monitoring of Adaptive Web Services. In *Proceedings of 9th IEEE European Conference on Web Services*, pages 109–116, September 2011.

[GR12] Mark Guttenbrunner and Andreas Rauber. A measurement framework for evaluating emulators for digital preservation. *ACM Transactions on Information Systems*, 30(2):14:1–14:28, May 2012.

[Gra00] Stewart Granger. Emulation as a digital preservation strategy. *D-Lib Magazine*, 6(10), 2000.

[Gro12] The Open Group. *ArchiMate 2.0 Specification.* Van Haren Publishing, 2012.

[GS10] Nihita Goel and R.K. Shyamasundar. Automatic Monitoring of SLAs of Web Services. In *Proceedings of IEEE Asia-Pacific Services Computing Conference*, pages 99–106, December 2010.

[Guo11] Philip 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.

[GWG$^+$07] Carole Goble, Katy Wolstencroft, Antoon Goderis, Duncan Hull, Jun Zhao, Pinar Alper, Phillip Lord, Chris Wroe, Khalid Belhajjame, Daniele Turi, Robert Stevens, Tom Oinn, and David De Roure. Knowledge discovery for biology with taverna. In *Semantic Web*, pages 355–395. Springer US, 2007.

[Hoh06] Andreas Hoheisel. User tools and languages for graph-based grid workflows. *Concurrency and Computation: Practice and Experience*, 18(10):1101–1113, 2006.

[HTT09] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research, 2009.

[HWS$^+$06] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:W729–W732, July 2006.

[IEE90] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, December 1990.

[IEE05] IEEE Std 1012 - 2004 IEEE Standard for Software Verification and Validation, 2005.

[ISO10] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.

[JCS+15] Fan Jiang, Claris Castillo, Charles Schmitt, Anirban Mandal, Paul Ruth, and Ilya Baldin. Enabling workflow repeatability with virtualization support. In *Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*, pages 8:1–8:10, New York, NY, USA, 2015. ACM.

[JvdHV07] Bram Lohman Jeffrey van der Hoeven and Remco Verdegem. Emulation for digital preservation in practice: The results. *International Journal of Digital Curation*, 2(2):123–132, 2007.

[KAC03] Bahman Kalali, Paulo Alencar, and Don Cowan. A service-oriented monitoring registry. In *Proceedings of Conference of the Centre for Advanced Studies on Collaborative research*, pages 107–121. IBM Press, 2003.

[KML06] Piotr Kaminski, Hausi Müller, and Marin Litoiu. A design for adaptive web service evolution. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 86–92, New York, NY, USA, 2006. ACM.

[KSea11] Alexander Kossiakoff, William N. Sweet, and Samuel J. Seymour ... [et al.]. *Systems engineering : principles and practice.* Wiley series in systems engineering and management. Hoboken, N.J. Wiley-Interscience, 2011.

[LAB+06] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[Lan05] M. Lankhorst. *Enterprise architecture at work: modelling, communication, and analysis.* Springer, 2005.

[LNZ04] Yutu Liu, Anne H. Ngu, and Liang Zeng Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of 13th International World Wide Web Conference*, pages 66–73, New York, NY, USA, 2004. ACM.

[LYJC11] Xiao Liu, Yun Yang, Yuanchun Jiang, and Jinjun Chen. Preventing temporal violations in scientific workflows: Where and how. *IEEE Transactions on Software Engineering*, 37(6):805–825, November 2011.

[MAC$^+$15] Rudolf Mayer, Gonçalo Antunes, Artur Caetano, Marzieh Bakhshandeh, Andreas Rauber, and José Borbinha. Using ontologies to capture the semantics of a (business) process for digital preservation. *International Journal of Digital Libraries*, 15:129–152, April 2015.

[Mar96] D. Marcum. The preservation of digital information. *The Journal of Academic Librarianship*, 22(6):451–454, 1996.

[Mat11] Juan Mata. Interpretation of concrete dam behaviour with artificial neural networks and multiple linear regression models. *Engineering Structures*, 33(3):903–911, 2011.

[Mcc07] B. D. Mccullough. Got Replicability? The Journal of Money, Credit, and Banking Archive. *Econ Journal Watch*, 4(3):326–337, September 2007.

[MMR14] Rudolf Mayer, Tomasz Miksa, and Andreas Rauber. Ontologies for describing the context of scientific experiment processes. In *Proceedings of the 10th IEEE International Conference on e-Science*, Guarujá, SP, Brazil, October 20–24 2014.

[MMR15a] Tomasz Miksa, Rudolf Mayer, and Andreas Rauber. Ensuring sustainability of web services dependent processes. *International Journal of Computational Science and Engineering (IJCSE)*, 10(1/2):70–81, 2015.

[MMR15b] Tomasz Miksa, Rudolf Mayer, and Andreas Rauber. Raising resilience of web service dependent repository systems. *International Journal of Web Information Systems*, 11(3), 2015.

[MMS$^+$14] Tomasz Miksa, Rudolf Mayer, Stephan Strodl, Andreas Rauber, Ricardo Vieira, and Goncalo Antunes. Risk driven selection of preservation activities for increasing sustainability of open source systems and workflows. In *Proceedings of the 11th International Conference on Digital Preservation*, Melbourne, Australia, October 6–10 2014.

[MMUR14] Tomasz Miksa, Rudolf Mayer, Marco Unterberger, and Andreas Rauber. Resilient web services for timeless business processes. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, pages 243–252, Hanoi, Vietnam, December 4–6 2014.

[MPM$^+$13] Tomasz Miksa, Stefan Pröll, Rudolf Mayer, Stephan Strodl, Ricardo Vieira, José Barateiro, and Andreas Rauber. Framework for verification of preserved and redeployed processes. In *Proceedings of the 10th International Conference on Preservation of Digital Objects*, Lisbon, Portugal, September 2–6 2013.

[MPR12] Rudolf Mayer, Stefan Pröll, and Andreas Rauber. On the applicability of workflow management systems for the preservation of business processes. In *Proceedings of the 9th International Conference on Digital Preservation*, pages 58–65, Toronto, Canada, October 1-5 2012.

[MR15a] Rudolf Mayer and Andreas Rauber. A quantitative study on the re-executability of publicly shared scientific workflows. In *Proceedings of the 11th IEEE International Conference on e-Science*, Munich, Germany, August 31 – September 04 2015.

[MR15b] Tomasz Miksa and Andreas Rauber. Beyond Data: Process Sharing and Reuse. *ERCIM News*, 100:18–19, January 2015.

[MSR14] Tomasz Miksa, Stephan Strodl, and Andreas Rauber. Process management plans. *International Journal of Digital Curation*, 9(1):83–97, 2014.

[MSRO^{+}10] Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Aleksandra Nenadic, Ian Dunlop, Alan Williams, Thomas Oinn, and Carole Goble. Taverna, reloaded. In *SSDBM 2010*, Heidelberg, Germany, June 2010.

[MSZ^{+}10] Paolo Missier, Satya Sanket Sahoo, Jun Zhao, Carole A. Goble, and Amit P. Sheth. Janus: From Workflows to Semantic Provenance and Linked Open Data. In *Proceedings of the International Provenance and Annotation Workshop (IPAW2010)*, pages 129–141, Troy, New York, USA, June 15–16 2010.

[MVBR14] Tomasz Miksa, Ricardo Vieira, José Barateiro, and Andreas Rauber. VPlan – ontology for collection of process verification data. In *Proceedings of the 11th International Conference on Digital Preservation*, Melbourne, Australia, October 6–10 2014.

[NU03] National Library of Australia and Unesco. *Guidelines for the preservation of digital heritage / Prepared by the National Library of Australia.* National Library of Australia ; Information Society Division, United Nations Educational, Scientific and Cultural Organization, 2003.

[PMB04] *A Guide To The Project Management Body Of Knowledge (PMBOK Guides).* Project Management Institute, 2004.

[PRE08] PREMIS Editorial Committee. Premis data dictionary for preservation metadata. Technical report, March 2008.

[Pro11] K. Scott Proctor. *Optimizing and Assessing Information Technology: Improving Business Project Execution.* Wiley, Oct 2011.

[Rad12] Tijs Rademakers. *Activiti in Action : Executable business processes in BPMN 2.0.* Manning Publications, Shelter Island, NY, first edition, 2012.

[RMMP15] Andreas Rauber, Tomasz Miksa, Rudolf Mayer, and Stefan Proell. Repeatability and Re-Usability in Scientific Processes: Process Context, Data Identification and Verification. In *Proceedings of the 17th International Conference on Data Analytics and Management in Data Intensive Domains*, Obninsk, Russia, October 2015.

[Rob08] Arnold Robbins. *Unix in a nutshell.* O'Reilly, 2008.

[Rot99] Jeff Rothenberg. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation.* Council on Library & Information Resources, 1999.

[RR05] C. Ryan and P. Rossi. Software, performance and resource utilisation metrics for context-aware mobile applications. In *Proceedings of 11th IEEE International Symposium on Software Metrics*, pages 10 – 12, September 2005.

[SFC07] C.T. Silva, J. Freire, and S.P. Callahan. Provenance for visualizations: Reproducibility and beyond. *IEEE Computing in Science Engineering*, 9(5):82–89, October 2007.

[SLP14] Victoria Stodden, Friedrich Leisch, and Roger D. Peng, editors. *Implementing Reproducible Research.* The R Series. Chapman and Hall/CRC, April 2014.

[SMA$^+$13] Stephan Strodl, Rudolf Mayer, Goncalo Antunes, Daniel Draws, and Andreas Rauber. Digital preservation of a process and its application to e-science experiments. In *Proceedings of the 10th International Conference on Preservation of Digital Objects*, pages 117–125, Lisbon, Portugal, September 2013.

[SMS15] V. Stodden, S. Miguez, and J. Seiler. Researchcompendia.org: Cyberinfrastructure for reproducibility and collaboration in computational science. *IEEE Computing in Science Engineering*, 17(1):12–19, January 2015.

[Tal13] Domenico Talia. Workflow Systems for Science: Concepts and Tools. *ISRN Software Engineering*, 2013:1–15, 2013.

[The09] The Open Group. TOGAF 9 - The Open Group Architecture Framework Version 9, 2009.

[The11] The SCAPE Project. Identification and selection of large-scale migration tools and services (D10.1). Technical report, 06 2011. Accessed: 20/11/2015.

[TIM13] TIMBUS Consortium. Deliverable 4.6 : Use Case Specific DP and Holistic Escrow. Technical report, 03 2013.

[TIM14] TIMBUS Consortium. Deliverable 6.10 Refinements to Populating and Accessing Context Model (2nd iteration). Technical report, 09 2014.

[TP04] T. Trucano and D. Post. Guest editors' introduction: Verification and validation in computational science and engineering. *IEEE Computing in Science Engineering*, 6(5):8–9, September 2004.

[TSWR03] Ian Taylor, Matthew Shields, Ian Wang, and Omer Rana. Triana applications within grid computing and peer to peer environments. *Journal of Grid Computing*, 1:2003, 2003.

[TZ08] Ralf Treinen and Stefano Zacchiroli. Description of the CUDF Format. Technical report, 2008. http://arxiv.org/abs/0811.3621.

[VdGW11] M. Van der Graaf and L. Waaijers. A Surfboard for Riding the Wave. Towards a four country action programme on research data. A Knowledge Exchange Report, 2011.

[VK11] Jan Vitek and Tomas Kalibera. Repeatability, reproducibility, and rigor in systems research. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, pages 33–38, New York, NY, USA, 2011. ACM.

[W3C03] W3C Working Group. QoS for Web Services: Requirements and Possible Approaches. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, 2003. Accessed: 30/06/2014.

[WFRG09] Katy Wolstencroft, Paul Fisher, David De Roure, and Carole Goble. Scientific workflows. *OpenStax CNX*, 2009.

[WMF$^+$05] SylviaC. Wong, Simon Miles, Weijian Fang, Paul Groth, and Luc Moreau. Provenance-based validation of e-science experiments. In Yolanda Gil, Enrico Motta, V.Richard Benjamins, and MarkA. Musen, editors, *The Semantic Web – ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 801–815. Springer Berlin Heidelberg, 2005.

[YCK$^+$10] Wen-wai Yim, Shu Chien, Yasuyuki Kusumoto, Susumu Date, and Jason H. Haga. Grid heterogeneity in in-silico experiments: An exploration of drug screening using DOCK on cloud environments. In Tony Solomonides, Ignacio Blanquer, Vincent Breton, Tristan Glatard, and Yannick Legré, editors, *Healthgrid Applications and Core Technologies - Proceedings of HealthGrid 2010, University Paris XI, Orsay, Paris, France, June 28-30, 2010*, volume 159 of *Studies in Health Technology and Informatics*, pages 181–190. IOS Press, 2010.

[ZGB$^+$] Jun Zhao, José Manuél Gómez-Pérez, Khalid Belhajjame, Graham Klyne, Esteban García-Cuesta, Aleix Garrido, Kristina M. Hettne, Marco Roos, David De Roure, and Carole A. Goble. Why workflows break - understanding and combating decay in taverna workflows. In *Proceedings of 8th IEEE International Conference on E-Science, Chicago, IL, USA, October 8-12, 2012.*