FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Erfassung und Visualisierung von Provenance-Information

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Fenghong Zhang, Bsc.
Matrikelnummer 01425097

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Andreas Rauber, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Wien, 1. November 2018

Fenghong Zhang	Andreas Rauber

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Capturing and Visualizing Provenance Information

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Fenghong Zhang, Bsc.
Registration Number 01425097

to the Faculty of Informatics

at the TU Wien

Advisor: Andreas Rauber, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Vienna, 1st November, 2018

Fenghong Zhang   Andreas Rauber

Erklärung zur Verfassung der Arbeit

Fenghong Zhang, Bsc.
Simmeringer Hauptstraße 170/6/12

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. November 2018

Fenghong Zhang

Danksagung

Ich möchte mich besonders bei meinem Betreuer, Ao.univ.Prof. Dr. Andreas Rauber vom Institut Software Engineering, der Technischen Universität Wien, bedanken. Die Entwicklung des ProSci-Projekts nahm viel Zeit in Anspruch und musste mehrere Entwicklungsschritte durchlaufen. Prof. Andreas Rauber hat mich bei der Entwicklung des ProSci-Projekts kontinuierlich begleitet und mich in die richtige Richtung gelenkt. Er gab mir viel Freiraum in der Entwicklung und ermutigte mich, meine eigenen Ideen durchzuführen. Bei jedem Treffen mit Ihm bekam ich neue Denkanstöße. Er empfahl mir eine Vielzahl guter APIs, um den Nutzen der Anwendung zu maximieren. Alles in allem, ohne seiner Unterstützung und Hilfe, hätte ich diese Arbeit nicht realisieren können.

Des Weiteren bin ich unglaublich Stolz auf meine Universität. Während meines Bachelor- und Masterstudiums habe ich 6 Jahre an der Technischen Universität Wien verbracht. In dieser Zeit habe ich viele wertvolle Dinge gelernt. Das Wissen und die Fähigkeiten, die ich durch das Studium erworben habe, haben mich mit der Kompetenz ausgestattet, dieses Projekt abzuschließen. Ich freue mich auf die bunte Zukunft und danke der TU Wien, dass Sie mir die grundlegenden Fähigkeiten für das spätere Leben vermittelt hat.

Zum Schluss muss ich meinen Eltern meine Anerkennung ausdrücken. Die Ermutigung und die kontinuierliche Unterstützung, die Sie mir gaben, haben mir sehr geholfen, diese Forschung durchzuführen. Ohne Ihre Unterstützung würde ich meine Ziele nicht erreichen. Großes Danke an meine Eltern.

Acknowledgements

At the very beginning, I would like to say thank you to my thesis supervisor Ao.univ.Prof. Dr. Andreas Rauber of the Information and Software Engineering Group (IFS) at Vienna University of Technology. The development of the ProSci project took quite a long time and went through several improvements. During the development of the ProSci project, Prof. Andreas Rauber guided me with continuous help and lead me in the right direction. He allowed me and gave me enough space of freedom in the development and encouraged me to insist on my ideas. Every time, after visiting him, I always came up with new clues. He suggested me a variety of good API, so that I can maximize the benefits from others' researches. All in all, without his support and help I cannot finish this work.

Secondly, I am so proud of my university. I have spent 6 years at the Vienna University of Technology for both my bachelor and master study, during the time I have learned so many valuable things. The knowledge and the capabilities which I gained from the study equipped me with the competence to be able to finish this project. I am looking forward to the colorful future and thanks to the TU Wien for giving me the fundamental abilities for the later life.

At last, I must express my appreciation to my parents. The encouragement and continuous supports they gave me helped me to be able to accomplish this research. I wouldn't achieve my goals without their support. Big Thanks to my parents.

Kurzfassung

In der modernen Wissenschaft wird eine geeignete Methodik für die Verwaltung und Kontrolle wissenschaftlicher Arbeitsabläufe, sowie deren Daten, eine immer wichtigere Rolle spielen. Die Berechnungsaufgaben umfassen das Lesen und auch die Verarbeitung der Daten aus externen Ressourcen. Die Aufgaben des gesamten Workflows müssen manchmal mehrmals abgearbeitet werden. Das Speichern und Steuern zwischen den einzelnen Workflows ist das Herzstück der Anwendung.

Wenn die Namenskonventionen nicht eingehalten werden, wird die Datei derzeit überschrieben und geht verloren. Das Ziel der Arbeit ist es, ein stabiles und funktionsfähiges Werkzeug für die Dokumentation des wissenschaftlichen Arbeitsablaufs bereitzustellen. Das Tool visualisiert diese Arbeitsabläufe mit einer hierarchischen Ansicht der historischen Zwischendateien. Darüber hinaus bietet es Zugriff auf die historisch generierten Herkunftsdaten. Der Ursprungsprozess des Werkzeugs basiert auf der Theorie der Versionskontrolle.

Abstract

In the perspective of modern science, a proper methodology for managing, monitoring and controlling scientific workflows for all kinds of collecting data from all fields and branches plays an increasingly important role. Especially for data analytical science. The computational tasks involve reading data from the external resource and computing processed data or finding evidence during the tasks processing as the final result. The tasks of the complete workflow sometimes need to run several times. Saving and controlling of the intermediation between each workflow's run is valuable and important.

Currently, if the naming conventions are not applied, the file will be overwritten and get lost. The aim of the work is to provide a stable and functionality enabled tool for scientific workflow provenance documentation. The tool visualizes the scientific workflows with the hierarchical view of the historical intermediate files. Moreover, it provides access to the historical generated provenance data. The provenance process of the tool should be realized and derived based on the version control theory.

Contents

CHAPTER 1

Introduction

Scientific workflows are clusters of computational tasks, which are connected to each other and enable different kinds of the functions, such like retrieving, processing, analyzing and abstracting from data. Those computational tasks always involve reading data from external resources, and computing the processed data or finding evidence during the tasks processing as final result. Scientific workflows can vary from simple to complex. Generally, they can be differentiated mainly in the following types: pipeline workflow, split workflow, merge workflow, diamond workflow and complex workflow.

In the perspective of modern science, a proper methodology for managing, monitoring and controlling scientific workflows for all kinds of collecting data from all fields and branches plays an increasingly important role, especially for the data analytical science. Therefore, the appearance of the scientific workflow management tool, such as Taverna, Kepler, and VisTrails, is admired. Those workflow management tools have a wide range of the abstraction levels, and if the workflows are appropriately designed, the provenance matches quite gut with the experiment semantics [PMBF17]. However, they require a high learning curve and adoption costs[STVK$^+$08].

The tasks in the complete workflows sometimes need to be run several times due to various reasons. The saving and controlling of the intermediation between each workflow's run is valuable and important. For instance, if the experiment runs several times, the cached intermediate data can help to avoid expensive operation due to the recomputing[GE11]. Currently, if the naming conventions are not applied, the file will be overwritten and get lost. Moreover, the provenance data can help scientists to discover and understand the reasons, why an experiment leads to inconclusive results [PMBF17]. Provenance captures all computational steps, used data, its output, input, intermediate data and libraries, and environment dependencies [CSF13]. The reproducing of the scientific experiment is not always possible because of a dynamic medley of hardware and software [Rot18]. The preparation of a systematically maintained archiving of the provenance data for scientific workflows is therefore critical and essential from the point of view of the scientific computational experiment.

Currently, we are missing such a tool, which can automatically catch the provenance data of the scientific experiment without any manual adaptations or learning curve required and is able to represent the provenance information which gathered during the experiment's run into a human-readable and intractable structure. We may already hear about Noworkflow[PMBF17]. This tool enables more or less requirement of our expectations. But unfortunately, it is a language related tool. It can only apply to Python[Ros95].

To overcome this limitation, the goals of this thesis is to provide a stable and functionality enabled tool for scientific workflow provenance documentation. The tool aims to supply a prototype to solve the following questions:

- How to automatically track, monitor and capture system provenance information at a wide range and system scale on the workflow steps, input, output, intermediate, library dependencies, hardware, and software environment, etc.
- How to transform the captured provenance data into provenance ontology?
- How to present the collected provenance data and the structured ontology into a human-readable format with interaction possibilities.

Furthermore, we are looking forward to identifying the outcome by the following criteria:

- How many provenance information can be efficiently and correctly collected by applying the methodologies which will be introduced in the next paragraph?
- How many information during the experiment by applying our prototype is going to be lost?
- Which provenance information is necessary for faithful reproduction of the scientific experiment?
- What kind of benefit is going to be detected through our thesis for the repeatability of the scientific experiment.

Since this paragraph, we are going to introduce the methodology approaches briefly. We will call our prototype solution "Prosci". It is an abbreviation of the sentence "Provenance of the scientific experiment". The provenance information collecting mechanism takes the benefits of the versioning and logging. As the tool intends to be deployed in the Unix system, the tracing tool targets the Strace [Wik19]. The JGit framework [DSS+14] is considered for the versioning purpose. It is applied comprehensively in our implementation to provides access to the historical generated provenance data. It is obvious that in the field of the computational experiment the environmental setting for executing the work can lead to very different results [GHJ+12]. That is why the environment for the experiment should simultaneously be taken into account. For solving this consideration the framework Oshi (https://github.com/oshi/oshi) is employed into the tool implementation. Last but not least, the representation of the transformation of the collected provenance data is based on the Prov-O Ontology[LSM+13], which was published by W3C as standard. To realize the interaction ability between the user and the transformed ontology, we are going to using Jung [Mor] to visualize the provenance ontology as a graph and using JavaFX [DHG+14] to facilitate the user interaction.

The tool initially contains three components, the application console, the file monitor and the graph visualizer. The application console should take responsibility for recording scientific computational tasks. Furthermore, the application console initializes the workspace for storing the processed results from the computational workflows and the corresponding logfiles for each separate task. In other words, it deals with the interaction with the user. The File Monitor is the monitoring tool for detecting the log file entrance under the workspace domain. The last and also the essential part of the tool is the workflow ontology visualizer. This component integrates the visualization of the workflow ontology, monitoring and controlling the computational tasks and its provenance ontology creator is used mainly for translating and converting the scientific workflows and their intermediate files into the Extensible Markup Language (XML) format and generate the visualized graph accordingly as the workflow ontology.

The thesis is organized into six chapters. In chapter 2 we are going to discuss some related terminologies and penetrate the topic of reproducibility and repeatability of the scientific experiment. At the same time, we will view some existing approaches. Chapter 3 intends to give us a brief introduction of all used APIs so that we can easily understand the description of the use cases and structure of the project's components from chapter 4. In chapter 5 we run the sample project and want to verify the functionality and the usability of the implemented tool. In the last chapter 6, we recall the research questions and reflect the limitation of Prosci and the possible future work are also mentioned.

CHAPTER 2

Theoretical Basis

In this chapter, we are going to suggest some of the essential technologies which are often used in our project, the related works which are done and published before, the techniques that exist already and the fundamental milestones on which our project is built upon. In all words, all subsections in this chapter are quite vital for you to get the idea of the crucial concepts of the ProSci project.

2.1 Reproducibility and Replication

First of all, you may ask, "what does research reproducibility mean?"[GFI16] Obviously there is no conceptual framework standard and settlement of the "research reproducibility" across the sciences. We introduce here some new lexicon for research reproducibility from the publication of Steven N et al.[GFI16]. They defined some basic terms in order to examine and enhance the reliability of research.

They define "methods reproducibility" as providing as much as possible detail about study procedures and enough data so the same procedures could theoretically or actually be exactly repeated. "Result reproducibility" points to receiving the same result from the execution of a separate study with closely matched procedures as the original experiment. Moreover, there are two important terms, that couldn't be forgotten– "robustness and generalizability". Robustness in the scientific experiment reproduction means that the variations between the experimental conclusion and the initial assumption as well as experimental procedures should be stable. Generalizability can be understood as the persistence of an effect in settings from outside of an experimental framework.

Steven N et al suggest last but not the less, "inferential reproducibility", this term. as the conducting of the qualitatively similar conclusion from an independent replication and reproducing the original one.

Why is the replication of research in computational science so important? Replication can be described as the ability to re-run the experiment under the same condition with the same methodologies and obtain the same results. "Replication is the ultimate standard by which scientific claims are judged. With replication, independent investigators address a scientific hypothesis and build up evidence for or against it."[Pen11], he says, usually the scientific experiment replication differentiates between two extreme points, full replication and no replication. Roger D. Peng drew in Fig. 2.1 a spectrum of the possibilities of the experimental research replication.

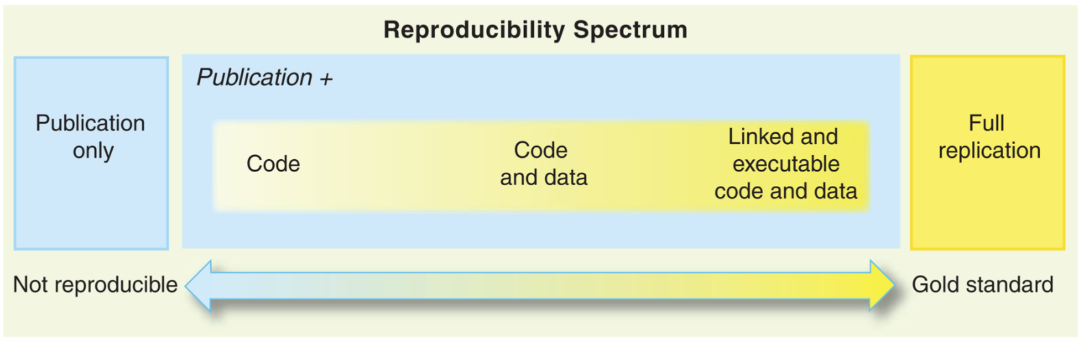


Figure 2.1: The spectrum of reproducibility [Pen11]

It is clear, that we aim to make the replication of the research settle as far as possible to the right hand of the Fig. 2.1. "Reproducibility has the potential to serve as a minimum standard for judging scientific claims when fully independent replication of a study is not possible." [Pen11]. And if there exist some universal regulations for reproducible computational research? Geir Kjetil Sandve et al. [SNTH13] proposed ten rules as below:

- For Every Result, Keep Track of How It Was Produced
- Avoid Manual Data Manipulation Steps
- Archive the Exact Versions of All External Programs Used
- Version Control All Custom Scripts

- Record All Intermediate Results, When Possible in Standardized Formats
- For Analyses That Include Randomness, Note Underlying Random Seeds
- Always Store Raw Data behind Plots
- Generate Hierarchical Analysis Output, Allowing Layers of Increasing Detail to Be Inspected
- Connect Textual Statements to Underlying Results
- Provide Public Access to Scripts, Runs, and Results

Jasonb[Bro14] extends the rules for reproducible research and targets it in the field of machine learning with some more points:

- Use a build system and have all results produced automatically by build targets.
- Automate all data selection, preprocessing and transformations.
- Use revision control and tag milestones.
- Strongly consider checking in dependencies or at least linking.
- Avoid writing code.
- Use a makeup to create reports for analysis and presentation output products.

Keeping all those criteria in mind, now we are going to start some discussion about the reproducibility of computer science research. In fact, in both wet sciences and applied computer science, the reproduction of someone's work is not as easy as we think. The problems involve not only needing expensive laboratory equipment and detailed procedures respective to the wet science. But also the unavailability of the source code, inability to build the source code in the field of the applied computer science. Furthermore, the execution environment and the incomplete codes often seem to be hurdles.

It is a fact that many researchers are making efforts to make the reproducibility in scientific experiment possible.

In the paper "Measuring Reproducibility in Computer System Research"[MSSW13], the authors collect 613 papers from eight ACM conferences and five journals. After some positive attempts to get the source code and dataset, 102 from 613 successfully ran. The authors propose a so-called "sharing specification", and they believe that if this method is generally adopted the willingness to share the code and data of the research will be positively impacted.

Vandewalle et al. [VKV09] also wrote in their paper to distinguish 6 different levels of reproducibility. The highest one is level 5 where "The results can be easily reproduced by an independent researcher with at most 15 min of user effort, requiring only standard, freely available tools", and the lowest one is 0 where "The results cannot be reproduced by an independent researcher.". Moreover, various web sites, web repositories, web portal, etc. like SHARE[GM11], CARMEN[AJF^{+}11] are also available.

Although, we can overwhelm the subjective reasons for sharing research and increase the accessibility of the source codes, dataset, etc. with the suggested methods from above. The objective factors such as different execution environment, a different version of the dependency will still impact the reproducibility of the scientific experiments.

Therefore, Koop et al. [KSM^{+}11] introduce a provenance-based system that captures workflows and Mache[BCMW11] supports the executable paper concept.

However, executable paper and the infrastructure that intends to support the life cycle of the executable paper still need a high learning curve and this requirement impacts all involved individuals, such as author, reader, reviewer of the executable paper. Furthermore, the high adoption costs which also exists in the Workflow Management Systems can't be ignored.

Jill P. Mesirov[Mes10] also proposed a Reproducible Research System(RRS) with two components, they are: Reproducible Research Environment (RRE) and Reproducible Research Publisher(RRP). RRE takes the responsibility of providing computational tools and monitoring the provenance of data, data analysis and its results, also, it packages them for the future distribution. RRP is a document preparation system, and it seems like a standard word-processing software.

RRS corresponds to the concept of the executable paper, the advantage of it is the automation of the tracing of the provenance data and the analysis of the result and packing them. RRS attempts to achieve the goal so that the reader not only look at the table or figure in the paper but also run the computational task to generate them.

However, due to the lack of the visualization of the provenance data of the background computational task, once the execution of the computational task failed, the reader can't find out the reason of the failure.

Adapting the intention of RRS we will overtake the concept of RRE and improve or change the so-called RRP into a graphic visualizer for illustrating the collected provenance information to the user directly so that they can interact with the information

Without a doubt, reproducible research plays a more and more increasingly important role in all fields of scientific experimentation. We are eager for a common methodology to realize this purpose and wondering if it can be made into a standard.

In 2008, Peng et al.[PE09] published an article on the topic "Distributed reproducible research using cached computations." In this article, they present a method which by using cached computations, stores the executions of the statistical analysis step by step in a collection of databases. Those collections of the execution steps can be published subsequently to public users. The interaction between users and researchers are facilitated.

Currently, this approach is implemented for R.

2.2 Reliability of the Replicated Research

We have already mentioned the importance of the reproducibility of scientific research, the difficulties that we face currently and the methodologies introduced by a various researchers who makes efforts to improve the standard. However, most of us don't know how to measure reliability across multiple data processing conditions. When we talk about the data processing conditions, we are pointing them to the terms, such as workstation type, operation system, CPU, RAM, etc.

As an example, we will concern ourselves the case FreeSurfer [GHJ+12], FreeSurfer is a popular software package to measure cortical thickness and volume of neuroanatomical structures. To discover the effects across different data processing conditions, we follow the research from Gronenschild et al. [GHJ+12]. In their experiment, they investigated the measurement of cortical thickness and volume with diverse altered variables, such as different FreeSurfer versions (v4.3.1, v4.5.0), different workstations (Macintosh and Hewlett-Packard), and Macintosh operating system versions (OSX 10.5 and OSX 10.6). In Fig. 2.2. we show the workstations used in the study.

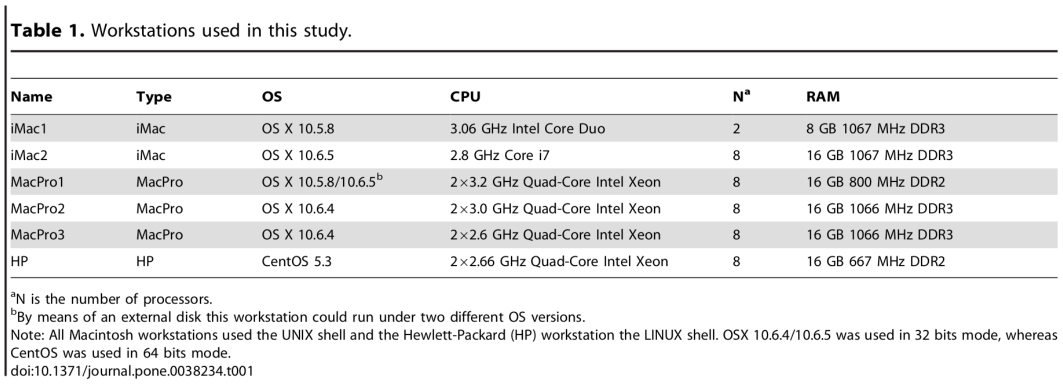


Table 1. Workstations used in this study.

Name	Type	OS	CPU	N[a]	RAM
iMac1	iMac	OS X 10.5.8	3.06 GHz Intel Core Duo	2	8 GB 1067 MHz DDR3
iMac2	iMac	OS X 10.6.5	2.8 GHz Core i7	8	16 GB 1067 MHz DDR3
MacPro1	MacPro	OS X 10.5.8/10.6.5[b]	2×3.2 GHz Quad-Core Intel Xeon	8	16 GB 800 MHz DDR2
MacPro2	MacPro	OS X 10.6.4	2×3.0 GHz Quad-Core Intel Xeon	8	16 GB 1066 MHz DDR3
MacPro3	MacPro	OS X 10.6.4	2×2.6 GHz Quad-Core Intel Xeon	8	16 GB 1066 MHz DDR3
HP	HP	CentOS 5.3	2×2.66 GHz Quad-Core Intel Xeon	8	16 GB 667 MHz DDR2

[a]N is the number of processors.
[b]By means of an external disk this workstation could run under two different OS versions.
Note: All Macintosh workstations used the UNIX shell and the Hewlett-Packard (HP) workstation the LINUX shell. OSX 10.6.4/10.6.5 was used in 32 bits mode, whereas CentOS was used in 64 bits mode.
doi:10.1371/journal.pone.0038234.t001

Figure 2.2: Workstations used in the study [GHJ+12]

In Fig. 2.3 we can see three parts, and all three parts show the detected percentage of the absolute differences between the results. Part A establishes the comparison between two workstations, Mac and HP, for three different versions of FreeSurfer. Part B demonstrates the differences between FreeSurfer v4.3.1 vs v4.5.0, v4.3.1 vs v5.0.0 and v4.5.0 vs v5.0.0. Part C describes the comparison between different OS OSX 10.5 and OSX 10.6 for three different FreeSurfer versions. Surprisedly, significant differences were detected between the FreeSurfer versions.

According to the study, the authors warn the users of the FreeSurfer to be careful before applying the upgrade in FreeSurfer versions, OS versions or switching to a new workstation during a continuing study.

When we talk about the reproducibility, we obviously must take various factors that potentially impact the experiment results into account, those factors include, not only the version of the tool but also the version of the operation system, the workstation type, etc.

Regarding the limitation and the potential weakness of the above mentioned related works. We are raising up some requirements for our work.

First of all, we want to ease the learning curve of the user and accomplish an easy-to-use conceive. Secondly, we hope the tool is not language-dependently, noWorkflow[PMBF17] can be seemed as a Counterexample. That is to say, the tool can conquer a wide range of computational tasks and should be used without special training requirement.

Furthermore, we hope the tool can benefit from tracing automation, just like RRE and it should be able to capture the environmental difference between executions, such as the different version of the dependency, used library, OS, and etc. The tool enhances the repeatability by visualizing the provenance information.

2.3 Scientific Workflow and Workflow Management System

2.3.1 Scientific Workflow

In this section, we are going to take a closer look at the scientific workflow (SWF) concept. What exactly does SWF mean? Some authors define the scientific workflow as follows:

"A scientific workflow is the description of a process for accomplishing a scientific objective, usually expressed in terms of tasks and their dependencies. Typically, scientific workflow tasks are computational steps for scientific simulations or data analysis steps. Common elements or stages in scientific workflows are acquisition, integration, reduction, visualization, and publication (e.g., in a shared database) of scientific data. The tasks of a scientific workflow are organized (at design time) and orchestrated (at runtime) according to dataflow and possibly other dependencies as specified by the workflow designer. Workflows can be designed visually, e.g., using block diagrams, or textually using a domain-specific language." [LWMB09]

Similarly, the authors of the article "The future of the scientific workflows" say that “in the context of scientific computing, a workflow is the orchestration of multiple computing tasks over the course of a science campaign." [DPA+18] They additionally divide SWFs into several sub-catalogs:

- Computational simulations
- Experimental observations
- Data analysis

- Visualization software working in concert to test a hypothesis and arrive at a conclusion

The scientific workflow is a practice to express a calculation formally, it usually involves multiple tasks, and the inter-dependencies between the tasks' parameters, inputs, and outputs. There are no limitations on the tasks, a task can be short or long, tasks couple with each other either loosely or tightly. We often run the same set of workflow set with different datasets.

The workflow series mainly consists of five elements. The first one is task specification, with or without dependencies. Sometimes the output of one task may be the input of another task. The second element is task scheduling. The tasks can run parallel or one after the other. Next, computational resources must be obtained. Metadata and the provenance data are also counted as an indispensable element. Finally, we need to handle the in- and output files. The so-called file management deals with issues, such as making input files available for execution and archive output files.

2.3.2 Scientific Workflow Management System (SWfMS)

After the clarification of the fundamental elements of the workflow, we realize that a tool that makes for planning, executing and monitoring of the complicated routine of the scientific workflow is desired.

The software products–Workflow Management System is designed to help users with creating and executing the workflows. They usually support all kinds of workflows and automate the workflow pipeline. With the help of the workflow management system, the tracking of the runtime, environment, arguments, inputs, and outputs of the execution of the workflows is automated. It ensures the availability of the data which is needed for the jobs. A prominently structured file storage is guaranteed. Fig. 2.4 illustrates the functionalities of the SWfMS. [LPVM15]

In Fig. 2.4, the scientific workflow management system is presented in layers. Basically, it is divided into five layers. The presentation layer, which in another word is the user interface, It facilitates the interaction between the user and the SWfMS. The user services layer provides the desired functions to users. The workflow execution plan (WEP) generation layer produces the WEP and the generated WEP is executed in the workflow execution plan execution layer. The necessary physical resources are accessible in the infrastructure layer. The last three layers constitute the scientific workflow execution engine.

Apparently, SWfMSs support the easy modeling of scientific experiments. Commonly, the modeling is expressed as a directed cyclic graph (DCG) or directed acyclic graph (DAG). The efficiency of SWfMS relies on several factors. Especially on the parallelism of SWFs and scheduling techniques. Under the term of the parallel execution, parallelism can be divided further into three types:

- data parallelism
- independent parallelism
- pipeline parallelism

Parallelization defines the workflow tasks which can be executed in parallelly in the WEP to accelerate the job processing time.

Under the concept of the scheduling, we understand a process of allocating concrete tasks to computing resources (i.e. computing nodes) to be executed during workflow execution.[BL13] The goal of the scheduling is to minimize the resource utilization and cost of the execution. The subcategory of the scheduling is shown as follows:

- static scheduling
- dynamic scheduling
- hybrid

The modern SWfMSs e.g. Kepler [ABJ+04b], Taverna [WMF+04], Chiron [ODS+13], etc. which are in use nowadays combine different techniques mentioned above allowing the scientists to plan, control and organize the SWF with best efforts. Fig. 2.5 shows us a comparison across different SWfMSs.

Taverna

As we talking about the SWfMSs, we can not ignore Taverna [WHF+13] , the most propagated SWfMS. Teverna concentrates on workflow executing and designing based on web services. The complex analysis pipelines of Teverna consists of distributed web services and local tools. The usage of the distributed services has the advantage that it reduces the local infrastructure and allows developing and testing the workflow more rapidly. On the contrary, using third-party web services are frequently dangerous. Because of the unavailability and the changes of the service interfaces.

By using Teverna, the user can access to several thousand different tools and resources and once the workflow constructed, they can be reused, executed and shared. Some considerable features of the Teverna are listed below:

- ability to perform implicit iteration, looping and streaming of the dataset.
- the implementation to interact with different types of services, such as WSDL Web Service, RESTful Web Services, BioMart data warehouses, etc.
- accessibility to the myExperiment repository.
- enable the execution of the workflow on remote computational infrastructure.
- enable the user to choose data and the parameters during the workflow execution.
- provenance suite records service invocation, intermediate and the final results and exports the W3C PROV model.
- the plugin architecture enable the easy code contributions and extension.

Kepler

As another example, we are going to address some features of Kepler. Kepler is one of the most popular scientific workflow management systems. It aims to provide scientists with an easy-to-use software tool for conducting analyses and run models in various software and hardware environments. "Kepler attempts to streamline the workflow creation and execution process so that scientists can design, execute, monitor, re-run, and communicate analytical procedures repeatedly with minimal effort. Kepler is unique in that it seamlessly combines high-level workflow design with execution and runtime interaction, access to local and remote data, and local and remote service invocation."[ABJ+04a]

As we mentioned in the introduction section, we notice that those SWfMSs do positively impact the reproduciability and benifit the development of the scientific experiment. But the scientists is obligated to define the experiment as workflow steps. Although the well-defined workflow can achieve a high level of the abstraction, but they still have their limitations. The user of the SWfMS must be trained before they are able to use them, thus cause the high learning expense. Moreover, SWfMSs are not using general-purpose languages and therefore lack of the flexibility[PMBF17].

In our thesis, we aim to find out a feasible solution which not only increase the useability but also enable the integration and communication between the user and the provennace information.

2.4 Provenance and Ontology

2.4.1 Provenance

Across all fields of the science domain, scientific workflows perform valuable adjustments, extraction, and processing on enormous data volume. Taking inputs and derive outputs, to discover the meaningful information behind the huge amount of data turns out to be meritorious. However, the workflow can be executed with different parameters under different conditions, the environment of the execution can vary from time to time. Therefore, a comprehensive provenance framework is vital for verifying the quantity and quality of the data and the scientific experiment in general, for reproducing and validating the scientific results and for associating the true value from the data to results.

The tracking of the provenance metadata of the SWFs is starting at the point of their creation, continues with intermediate processing, and ends with using the final results. [SNB^{+}11] Sahoo et al. illustrate the provenance life cycle with four phases in Fig. 2.6.

They distinguish between a pre- and post-publication phase of the provenance life cycle. Pre-publication is the state before the publication of the data and the results for public access and uploading it to a data repository, post-publication describes the phase, when the results are used by data mining or knowledge discovery. During the pre-phase, the provenance is collected to describe the experiment design, the platform, in which the experiment is executed and the tools which are used for analyzing and processing. Instead, the provenance in post-phase is used to conduct the analysis algorithm and interpretation of results.

The word provenance comes originally from the French language. It means "to come from". To interpret the concept of provenance we follow the instruction of the Boris Clavic et al. [GD07]. They distinguish provenance into two parts: the provenance model and the provenance management system. They also define the conceptual properties of data provenance into a hierarchy of categories in Fig. 2.7 2.8 2.9. The three major scheme are:

- Provenance model

- Query and manipulation functionality
- Storage model and recording strategy.

2.4.2 Ontology

"Before, choosing which data has to be stored, it is necessary to define how these data have to be structured so that they can be later recovered and understood " [dPHG$^+$13] Renato de Paula et al. state in their research. The development of diverse provenance models e.g., Open Provenance Model(OPM) [MCF$^+$11], PROV-DM[MM12] and Provenance Vocabulary [HZ10] satisfies the need of provenance demand on scientific workflow.

Concerning the provenance model, we need to explore a related term–ontology. "Computational ontologies are a means to formally model the structure of a system, i.e., the relevant entities and relations that emerge from its observation, and which are useful to our purposes." [GOS09]

Indeed, OPM is currently applied in a wide range, many libraries and tools are available, it facilitates the interoperability of provenance data. Since PROV-O[LSM$^+$13] is the forthcoming data model and is provided by W3C with specifications as the first official standard, for tracking the provenance metadata we introduce the PROV-O to describe the structure of our scientific workflow provenance. PROV-O using OWL2 Web Ontology Language[HKP$^+$09] and provides a set of classes, restrictions, and properties for describing and representing the provenance information which generated and produced by various systems and environmental contexts for different applications and domains. The starting points of the PROV-O Ontology are:

- prov: Entity
- prov: Activity
- prov: Agent

“Entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects", usually in the context of SWFs it means dataset, script, etc.

"Activity is something that occurs over a period of time and acts upon or with entities". Activities, therefore, equal the workflow step itself. It is the action that takes the dataset as input and runs the script with parameters and gets the result.

The agent in our curriculum is the software agent, who runs the action. It is the environment, where the workflows are executed.

Accordingly, the properties which are used to describe the relationships between starting points are derived. Fig. 2.11 illustrates some basic properties of the PROV-O. Related to that, Fig. 2.10 shows the intrinsic associations between starting points and their properties.

2.5 Version Control System

The version control system is the tool, which enables the user to recall, review or return to a specific previous version of the file system. It records the changes to a series of the files over time. Software developers use the version control system to control their programming procedures. Actually, the version control system can be applied to every kind of file.

Over the decades, the development of version control system has gone through several revolutions. Scott Chacon[Cha14] differentiates the version control systems into three types. There are local version control systems, centralized version control systems, and distributed version control system.

2.5.1 Local Version Control System

The Local VCS is built upon a database, this database which keeps all the changes to files. The famous tool which uses this technology is called rcs[rcs15].

2.5.2 Centralized Version Control Systems

When considering the integration for different developers, the next revolution of the version control system appears. The centralized VCS has a single server that contains all files and their varying versions, see Fig. **??** . Over the years it became a standard for the version control system. The implementation of this generation such as Subversion, CVS and Perforce were quite recommendable. In spite of the advantage of it. Practice showed, that this single point of failure is its downside. The entire history of the work is stored in one place, the risk of losing everything is extremely high.

2.5.3 Distributed Version Control Systems

In order to avoid the single point of failure the DVCSs step in. In this kind of version control systems such as Git, Bazaar, etc. the user checks out not only the last version of the file system but copies the entire repository, see Fig. **??**. You can collaborate your work with different people in different environments simultaneously and don't need to worry about data loss.

2.6 Summary

Reconsider the mentioned related works. We recognize that reproducibility plays a significantly important role in the computational scientific experiment. With a high reproducibility, the scientist can enhance the reliability of the research and it can seem like a minimum standard for judging the scientific claim.

Many researchers consecrate themselves to improving the reproducibility of the scientific experiment subjectively and objectively, however, we still missing a tool which highly integrated with the automation tracing process of provenance data and transformation of the provenance information into human readable ontology standard and illustrating the ontology as a user interactable representation. In the end, an infrastructure, which can be applied to diagnose and examine the possibility of the crash in the re-run procedure of the computational tasks is built.

Recall our thesis purposes, we want to implement a prototype which can compensate for the above-mentioned absence of the tool. Our Prosci project aims to extend the concept of the RRE for automatical tracking, monitoring and transforming processes. It overcomes the language boundary such as noWorkflow [PMBF17]. It eases the learning effort of the user and thus can be applied for a wide range of the domain. In the next chapter, we are going to look into the details of the used API, which are used in the implementation of Prosci.

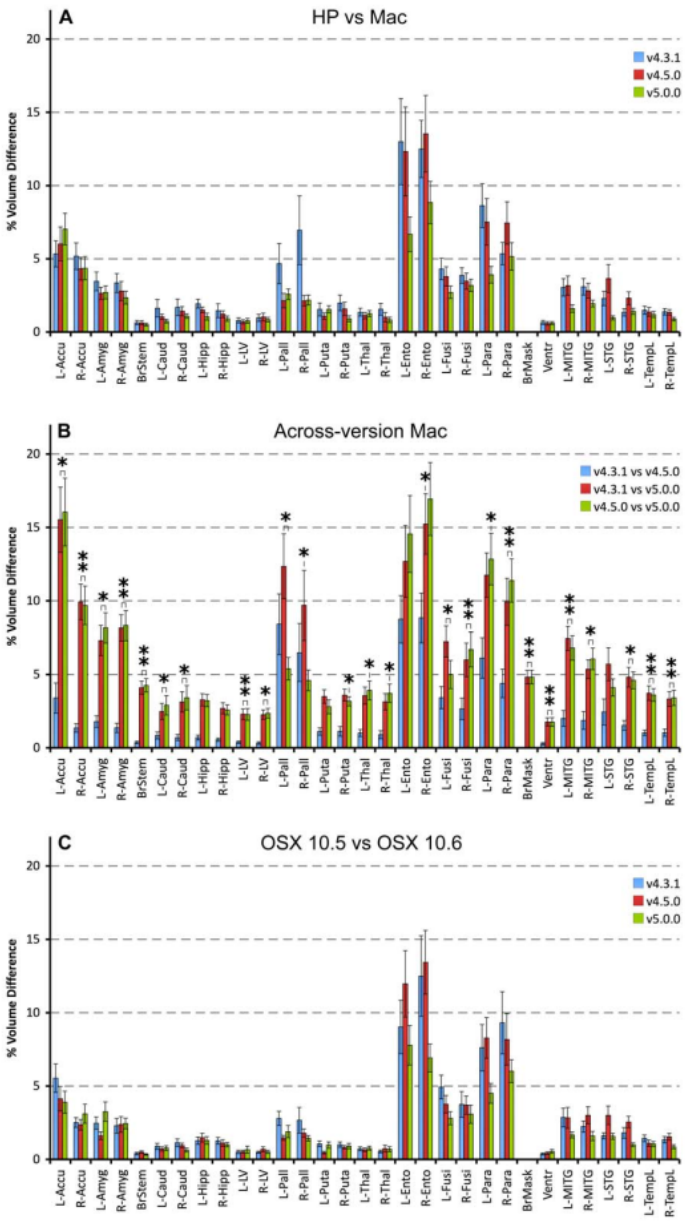


Figure 7. Effects of data processing conditions on the voxel volumes for a subsample of (sub)cortical structures. Panel A shows the detected percentage absolute differences between the results derived on a Mac and HP workstation for three different versions of FreeSurfer. Panel B depicts the differences between FreeSurfer v4.3.1 vs. v4.5.0, v4.3.1 vs. v5.0.0, and v4.5.0 vs. v5.0.0 for the Mac (OSX 10.5) (for HP these are very similar). Panel C displays the differences between OSX 10.6 and OSX 10.5. For comparison purposes the same vertical scale was used as in Figure 3 of Morey et al. [14] in which the same structures up to the left and right thalamus were considered. The significance is indicated at two levels: * : $p<0.025$ (the FDR level); ** : $p\leq0.0001$. Abbreviations: L: left; R: right; Accu: accumbens; Amyg: amygdala; BrStem: brain stem; Caud: caudate; Hipp: hippocampus; LV: lateral ventricle; Pall: pallidum; Puta: putamen; Thal: thalamus; Ento: entorhinal cortex; Fusi: fusiform; Para: parahippocampal gyrus; BrMask: brain mask; Ventr: left+right lateral and inferior lateral ventricles; MITG: medial-inferior temporal gyrus; STG: superior temporal gyrus; TempL: temporal lobe. doi:10.1371/journal.pone.0038234.g007

Figure 2.3: Effects of data processing conditions on the voxel volumes for a subsample of subcortical structures [GHJ+12]

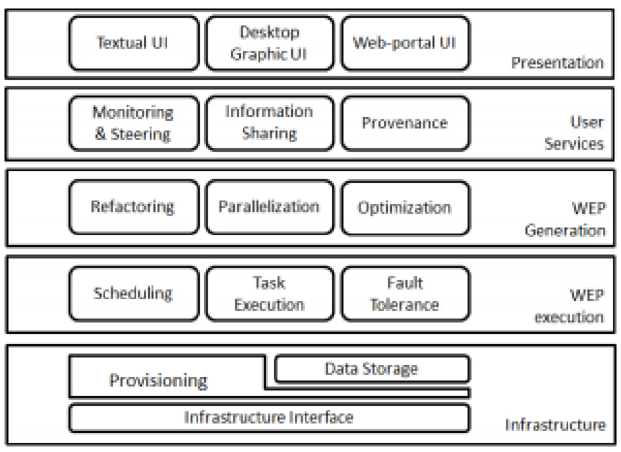


Fig. 3: **Functional architecture of a SWfMS.**

Figure 2.4: Functional architecture of a SWfMS [LPVM15]

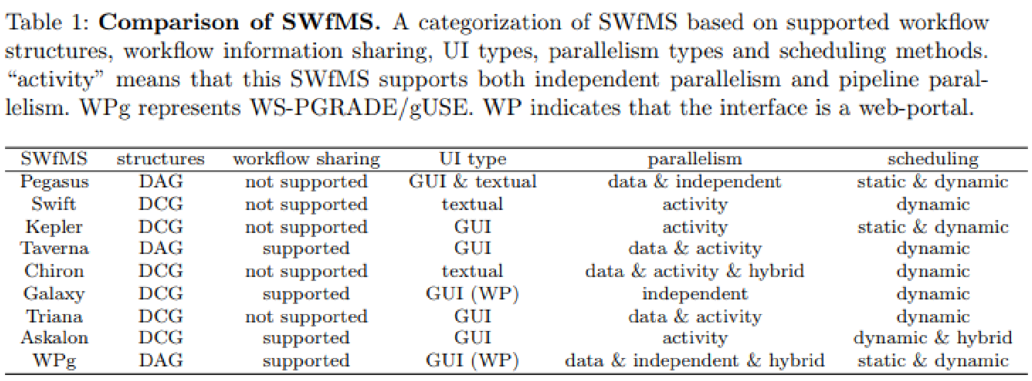


Table 1: **Comparison of SWfMS.** A categorization of SWfMS based on supported workflow structures, workflow information sharing, UI types, parallelism types and scheduling methods. “activity” means that this SWfMS supports both independent parallelism and pipeline parallelism. WPg represents WS-PGRADE/gUSE. WP indicates that the interface is a web-portal.

SWfMS	structures	workflow sharing	UI type	parallelism	scheduling
Pegasus	DAG	not supported	GUI & textual	data & independent	static & dynamic
Swift	DCG	not supported	textual	activity	dynamic
Kepler	DCG	not supported	GUI	activity	static & dynamic
Taverna	DAG	supported	GUI	data & activity	dynamic
Chiron	DCG	not supported	textual	data & activity & hybrid	dynamic
Galaxy	DCG	supported	GUI (WP)	independent	dynamic
Triana	DCG	not supported	GUI	data & activity	dynamic
Askalon	DCG	supported	GUI	activity	dynamic & hybrid
WPg	DAG	supported	GUI (WP)	data & independent & hybrid	static & dynamic

Figure 2.5: Comparison of SWfMS [LPVM15]

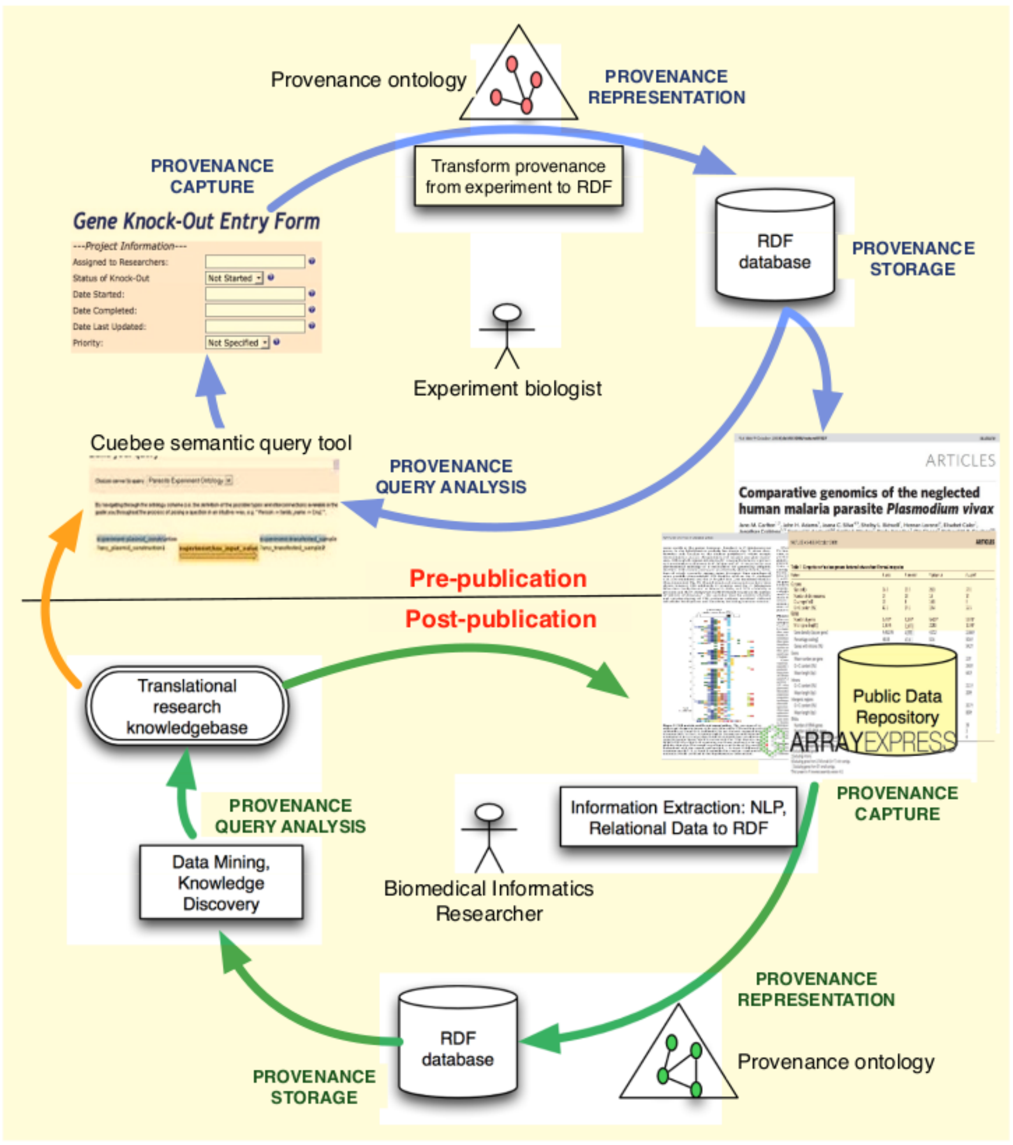

Provenance ontology
PROVENANCE REPRESENTATION
PROVENANCE CAPTURE
Transform provenance from experiment to RDF
RDF database
PROVENANCE STORAGE
Gene Knock-Out Entry Form
---Project Information---
Assigned to Researchers:
Status of Knock-Out
Not Started
Date Started:
Date Completed:
Date Last Updated:
Priority:
Not Specified
Experiment biologist
Cuebee semantic query tool
PROVENANCE QUERY ANALYSIS
ARTICLES
Comparative genomics of the neglected human malaria parasite Plasmodium vivax
Pre-publication
Post-publication
Translational research knowledgebase
Public Data Repository
ARRAYEXPRESS
PROVENANCE CAPTURE
PROVENANCE QUERY ANALYSIS
Information Extraction: NLP, Relational Data to RDF
Data Mining, Knowledge Discovery
Biomedical Informatics Researcher
PROVENANCE REPRESENTATION
RDF database
Provenance ontology
PROVENANCE STORAGE

Figure 1 Provenance lifecycle in the pre and post-publications stage of the translational research. In the pre-publication phase, the biologist uses the provenance information, generated during an experiment, for project management and for publication in literature. In the post-publication phase, provenance, of both the process of information extraction and the extracted information, is used for ranking results and refining quality of results in data mining and knowledge discovery applications.

Figure 2.6: Provenance lifecycle in the pre and post-publications stage of the translational research [SNB^{+}11]

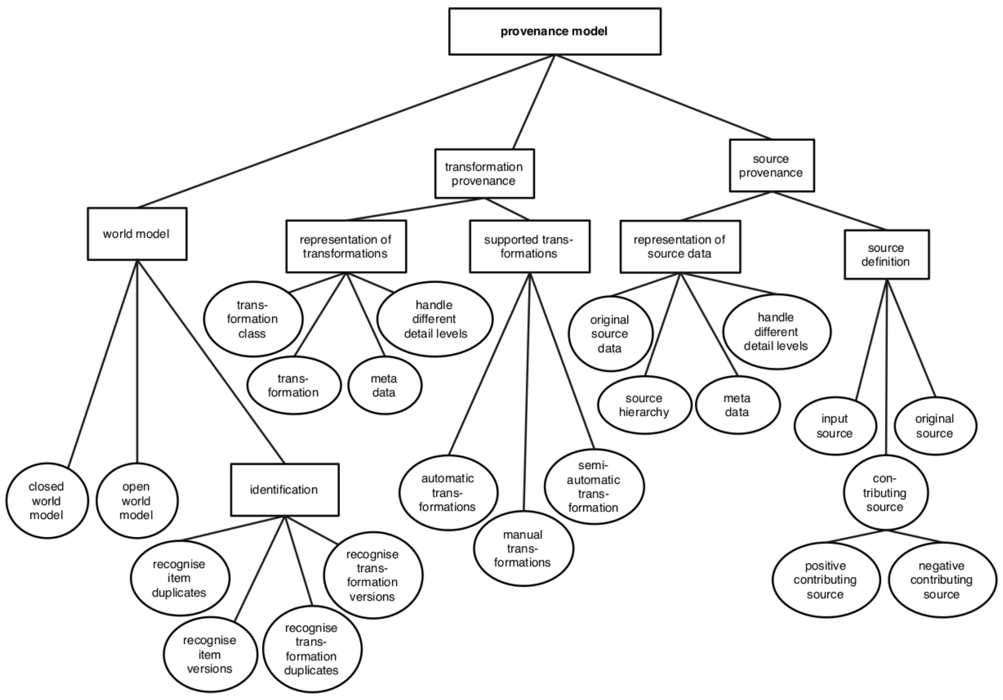

provenance model
world model
transformation provenance
source provenance
closed world model
open world model
identification
recognise item duplicates
recognise item versions
recognise trans-formation duplicates
recognise trans-formation versions
representation of transformations
trans-formation class
trans-formation
meta data
handle different detail levels
supported trans-formations
automatic trans-formations
manual trans-formations
semi-automatic trans-formation
representation of source data
original source data
source hierarchy
meta data
handle different detail levels
source definition
input source
con-tributing source
original source
positive contributing source
negative contributing source

Figure 1: Provenance model

Figure 2.7: Provenance model [GD07]

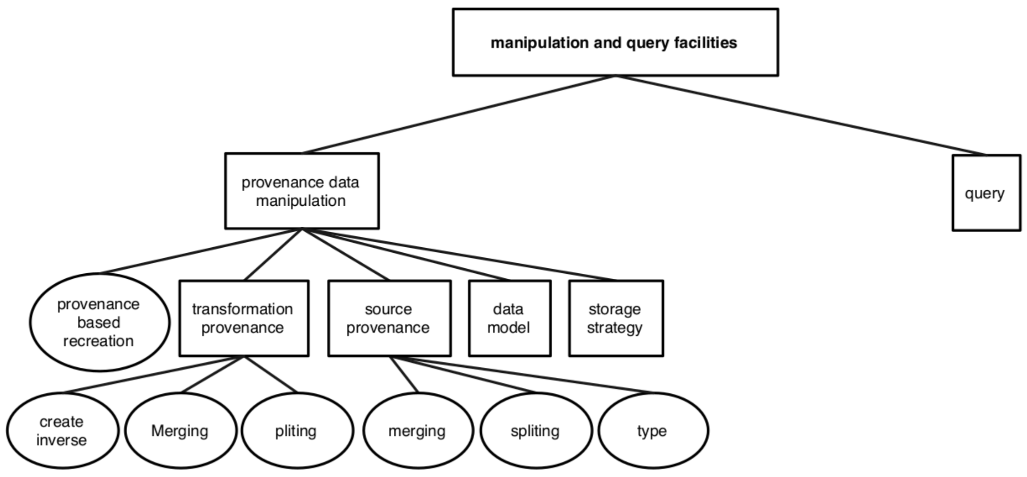

manipulation and query facilities
provenance data manipulation
query
provenance based recreation
transformation provenance
source provenance
data model
storage strategy
create inverse
Merging
pliting
merging
spliting
type

Figure 2: Query and manipulation funcionalities

Figure 2.8: Query and manipulation funcionalities [GD07]

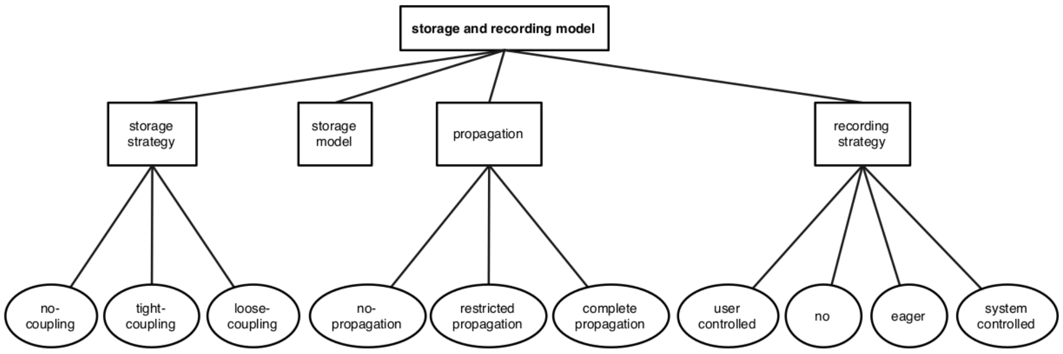


Figure 3: Storage and recording

Figure 2.9: Storage and recording [GD07]

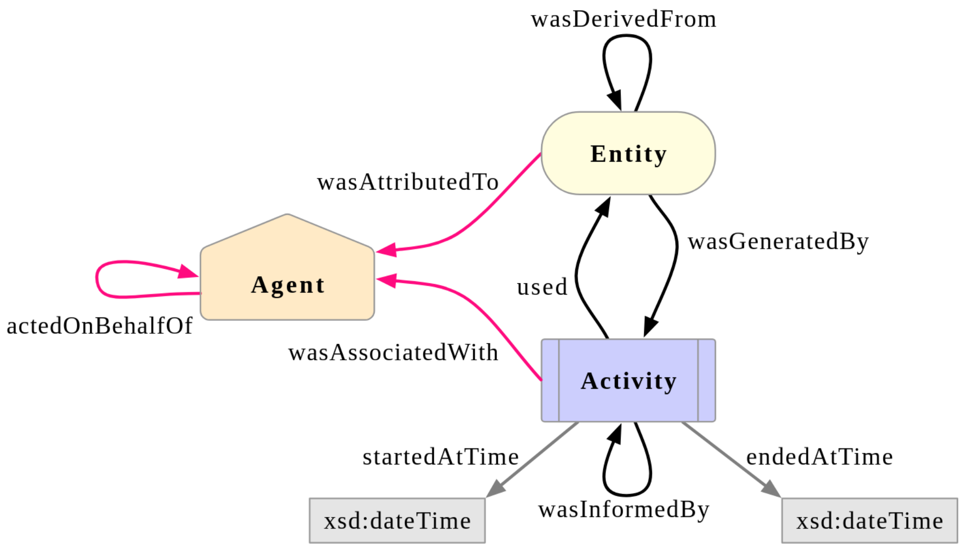


Figure 1. The three Starting Point classes and the properties that relate them. The diagrams in this document depict Entities as yellow ovals, Activities as blue rectangles, and Agents as orange pentagons. The responsibility properties are shown in pink.

Figure 2.10: The three Starting Point classes and the properties that relate them [LSM+13]

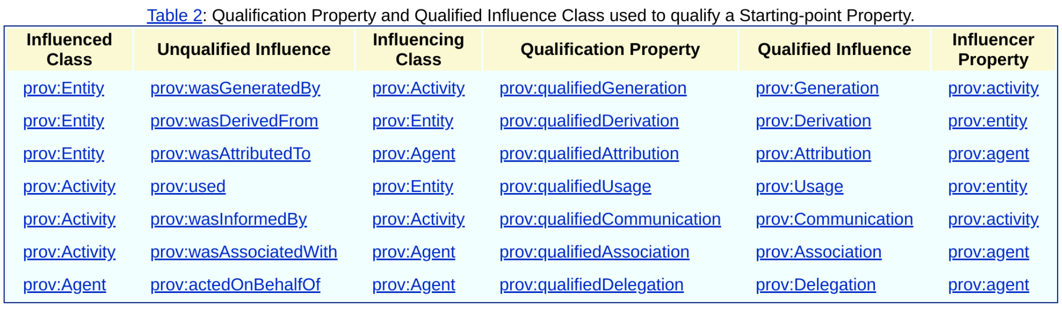


Table 2: Qualification Property and Qualified Influence Class used to qualify a Starting-point Property.

Influenced Class	Unqualified Influence	Influencing Class	Qualification Property	Qualified Influence	Influencer Property
prov:Entity	prov:wasGeneratedBy	prov:Activity	prov:qualifiedGeneration	prov:Generation	prov:activity
prov:Entity	prov:wasDerivedFrom	prov:Entity	prov:qualifiedDerivation	prov:Derivation	prov:entity
prov:Entity	prov:wasAttributedTo	prov:Agent	prov:qualifiedAttribution	prov:Attribution	prov:agent
prov:Activity	prov:used	prov:Entity	prov:qualifiedUsage	prov:Usage	prov:entity
prov:Activity	prov:wasInformedBy	prov:Activity	prov:qualifiedCommunication	prov:Communication	prov:activity
prov:Activity	prov:wasAssociatedWith	prov:Agent	prov:qualifiedAssociation	prov:Association	prov:agent
prov:Agent	prov:actedOnBehalfOf	prov:Agent	prov:qualifiedDelegation	prov:Delegation	prov:agent

Figure 2.11: Qualification Property and Qualified Influence Class used to qualify a Starting-point Property [LSM+13]

CHAPTER 3

Library Usages

After reviewing, collecting and reflecting the related works from other researchers, we gather the methodologies heuristically and intend to apply them in our ProSci Project. In this Chapter, we will introduce the external library usages which are used across the project and how ProSci benefits from their utilization.

3.1 JGit

Since you shouldn't be unfamiliar with the Distributed Version Control System Git, the name "JGit" reminds you trivially as the Java implementation of the Git VCS. JGit[DSS$^+$14] is a Java framework with very few dependencies, and hence, it is becoming the appropriate API for embedding in any Java application, which profits from version control integration.

A repository is a location, where all objects and refs are stored for managing resources. Besides, JGit has four types of objects, they are specified in the Table 3.1.

Furthermore, we have "Ref", which is an object identifier and can references to any kind of git object, such as blob, tree, commit and tag. A "RevWalk" walks a commit graph, The "RevCommit" represents a commit in Git. The "RevTag" represents a tag.

blob	store file data
tree	a directory, references other trees and blobs
commit	references to a single tree
tag	marks specific releases

Table 3.1: JGit Objects

JGit has not only low-level code but also a higher level API to work with Git repository.

- git add: add files to the index
- git commit: perform commits
- git tag: tagging options
- git log: allow the user to walk a commit graph
- ...

3.2 ProvToolbox

"ProvToolbox is a Java library to create Java representations of PROV-DM, and convert them between RDF, PROV-XML, PROV-N, and PROV-JSON" [Mor15]. The source code of the ProvToolbox can be found in the Gitlab repository:

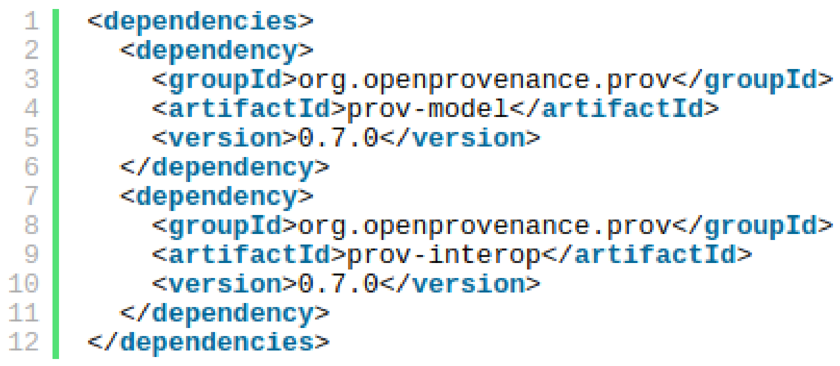
`https://github.com/lucmoreau/ProvToolbox`.

By using the maven configuration we can get the API straightforward. (See Fig. 3.1)

The last version of the ProvToolbox is 0.7.0. Since version 0.6.1 ProvToolbox is deployed on Maven central.

The Javadoc of the ProvToolbox is ready to be visited under the url:

`https://openprovenance.org/java/site/latest/apidocs/`



```xml
<dependencies>
  <dependency>
    <groupId>org.openprovenance.prov</groupId>
    <artifactId>prov-model</artifactId>
    <version>0.7.0</version>
  </dependency>
  <dependency>
    <groupId>org.openprovenance.prov</groupId>
    <artifactId>prov-interop</artifactId>
    <version>0.7.0</version>
  </dependency>
</dependencies>
```

Figure 3.1: Maven Configuration of ProvToolbox [Mor15]

3.3 JavaFX

JavaFX was originally planned to replace Swing as the standard GUI API for Java. It is essentially a software technology for developing and delivering client application for mobile devices, desktop and built on Java application. JavaFX aims to produce a modern, efficient and fully featured toolkit for the rich web application. It allows the developer with the integration of vector graphics, audio, video, and various web assets.

Over the years, JavaFX has stepped across from JavaFX 1.0 to JavaFX 11. Since JavaFX 2.0 the JavaFX is written in "native" Java code. JavaFX is now a part of the JDK/JRE for Java8. With this version of the JavaFX, the developer gains some new added features:

- 3D graphics
- sensor support
- printing and rich text
- generic dialog template
- MathML

The components of the JavaFX includes:

- The JavaFX SDK
- IDE for JavaFX

- JavaFX scene builder, this is the user interface for creating and designing the components and the designing information will later be saved in a FXML file using XML format.

You can visit the most recent project of the JavaFX there:

`https://openjfx.io/`

3.4 Strace

"Strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state."[Wik19]

Strace was first published in 1992 by Paul Kranenburg. Later Branko Lankester, Richard Sladkey, Dmitry Levin, and etc. have taken the responsibility for maintaining Strace.

The most common usage of Strace is to print out all system calls. With this functionality, the user can monitor the system and find out the outlier of their expectation. As we all know system calls are events that happen at the kernel interface, a monitoring mechanism is valuable for detecting system bugs and capturing background problem.

Strace outputs usually contain in each line a system call with name, arguments, and the return value.

An example output of strace by running command "ls" on Ubuntu System:

```
XX@XX: $ strace ls
execve("/bin/ls", ["ls"], 0x7ffea3e29cb0 /* 69 vars */) = 0
```

Table 3.2, we list the useful parameters that we are going to use in our ProSci strace tracing.

-ff	follow forks with output into separate files
-f	follow forks
-y	print paths associated with file descriptor arguments
-tt	print absolute timestamp with usecs
-s strsize	limit length of print strings to STRSIZE chars (default 32)
-o file	send trace output to FILE instead of stderr

Table 3.2: Strace Parameters

Most of the parameters in the table are trivial, one may confuse you is the "-s" parameter, this parameter is used to limit the length of the print strings. The reason why we take it into account is that we want to avoid the information loss. Normally, with 32 characters, we can't capture complete information of a Unix system call including its command name, parameters, status code and etc. Therefore, as the default value, we currently set it to 2048 in order to make sure that we won't lose any information.

3.5 Oshi

Oshi[osh] is a free Java API for collecting, retrieving and gathering information about the operating system and hardware. It is independent, the installation requires no additional libraries. Oshi is supported by almost all common operation systems, such as Windows, Linux, Mac OS X and Unix (Solaris, FreeBSD).

Currently, various projects are using Oshi for capturing hardware and software information. Some noteworthy projects are listed below:

- Apache Flink
- GoMint
- Eclipse Orbit
- Eclipse Passage
- GeoServer
- PSI Probe
- DeepLearning4J

In the list below you can see a list, which we was taken from the homepage of Oshi. In this Fig. the supported features of the Oshi are described.

- Computer System and firmware, baseboard
- Operating System and Version/Build
- Physical (core) and Logical (hyperthreaded) CPUs
- System and per-processor load and tick counters
- CPU uptime, processes, and threads
- Process uptime, CPU, memory usage
- Physical and virtual memory used/available
- Mounted filesystems (type, usable and total space)
- Disk drives (model, serial, size) and partitions
- Network interfaces (IPs, bandwidth in/out)
- Battery state (capacity, time remaining)
- Connected displays (with EDID info)
- USB Devices
- Sensors (temperature, fan speeds, voltage)

However, not all feature are implemented across all operating system types. In Windows, the load average is skipped, the sensor's indicators are read from Microsoft's Windows Management Instrumentation. For MacOS time processors spend idle will not be monitored. On the other hand, Linux, Solaris, and FreeBSD may request running as root user.

You may find the resource code of Oshi from the linke below:

`https://github.com/oshi/oshi/blob/master/UPGRADING.md`

and the API docs can be found there:

`http://oshi.github.io/oshi/apidocs/`

Currently, the latest version of Oshi is 4.x. You can configure the Oshi in the maven. Oshi 3.x is compatible with the Java 7 and Oshi 4.x requires a minimum version of Java 8.

In our case, analyzing the survey of Gronenschild et al[GHJ$^+$12]. we will take the following points into consideration:

- Computer System and firmware, baseboard
- Operating System and Version/Build

Besides, the points below are also considered:

- Processor
- Memory
- CPU

3.6 Jung

JUNG is an open-source Java library. JUNG tries to avoid the continual re-inventing from other developers who consecrate themselves to focus on working on relational data analysis by providing a common framework for graph and network analysis.

JUNG was initiated in 2003. The aim of the project was to provide "a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network."[Mor]

JUNG supports to illustrate a variety of the entities and their relations as a directed and undirected graph, a graph with different kinds of edges and also a hypergraph. Each node and edge can be annotated according to user demands.

JUNG implements a number of algorithms from graph theory, data mining and etc. The user of JUNG can use the ready-to-use layouts from JUNG and they can also create their own layout.

The last version of JUNG (v2.1.1) was released on September 7th 2016. The repository is administrated by one of the co-creators of the JUNG project, Joshua O'Madadhain. Find the source code from the URL here:

`https://github.com/jrtom/jung`

Visit the homepage of JUNG:

`https://jrtom.github.io/jung/`

The Java doc specification can be found here:

`https://jrtom.github.io/jung/javadoc/index.html`

Moreover, JUNG can also be integrated in your project by defining the following maven dependency.

```
<dependency> <groupId>net.sf.jung</groupId>
<artifactId>jung-[subpackage]</artifactId>
  <version>2.1.1</version> </dependency>
```

3.7 Summary

After the short introduction of the above-mentioned API, we construe the Prosci project into three parts and each part utilizes the corresponding APIs and consequently perform its unique function in the project.

Each part proposes to solve one of the question which arose in the introduction section. The mechanism to automatically tracking, collection, versioning and capturing the provenance data is built upon the help of the Strace, JGit, and Oshi. The possibility for automatically transforming the provenance information into ontology definition uses the ProvToolbox as a foundation. Last challenge in the project is the presentation of the ontology into a human-readable and interactable format. For this intention, Jung and JavaFX are actively involved in our Prosci project

Let's go through the implementation detail in the next chapter, to know how the functionalities of Prosci is realized by using those of the APIs.

CHAPTER 4

Project Structure

4.1 Use Case

In this section, I am going to establish the primary use cases of the Prosci tool. As shown in Fig. 4.1. Users of the Prosci, have six main functions, from which to choose. The description of each use case is shown in the table under certain subsection. The workspace plays an essential role of the Prosci tool. Every time the user starts the application, he needs to define a workspace, and this workspace needs to be an existing one. Or as an option, the user can also create a new one, after the creation the new one is automatically defined as the "current workspace." Furthermore, some helper functions, such as "print help menu", "show workspaces" and "save files" are also available, to make it convenient for the user to control the Prosci tool.

The most import two most important functionalities of Prosci, are the Xterm starter and the graphic-visualizer. The pre-condition for those two functionalities is a specified workspace domain. Xterm is a standard terminal emulator, and it has the responsibility to simulate the system terminal so that the user can run their experiment freely using commands.

What we still need to mention, is the Graphic-Visualizer. It provides the user with an overview of all of the workflows with ontology definitions. Additionally, our Graphic-Visualizer supplies some specific features. Through it, the users are capable to rerun the historical executions in the system command backlog and they are also in a position to recover the overwritten files to a specific path under the workspace directory.

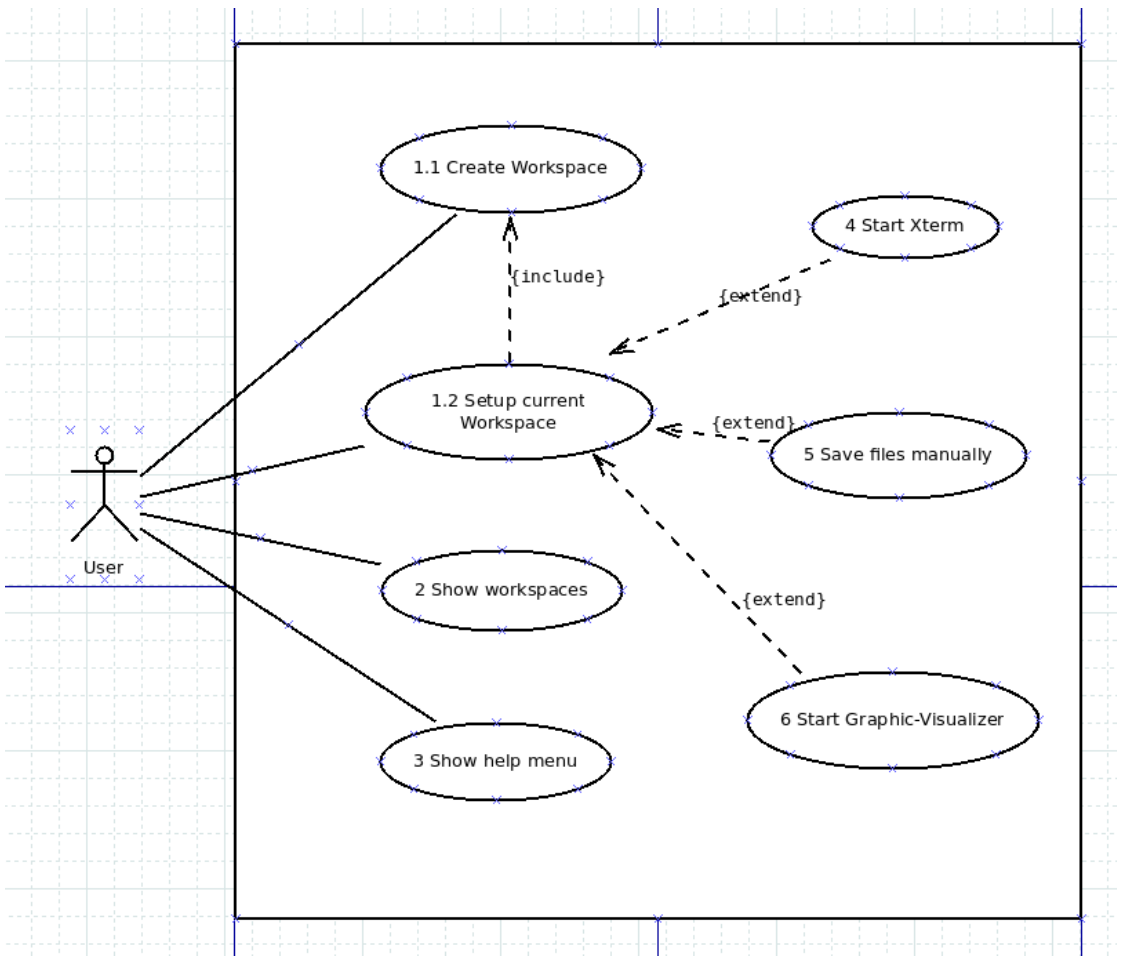


Figure 4.1: Main user cases of the Prosci

4.1.1 Create Workspace

A workspace is a dynamically changed directory, within which the source code, the generated files, the system tracing logs, etc. of the system executions are saved and organized.

As a consequence, every time the user wants to start the application, they need to define a workspace. The first option is to create a new workspace. A new workspace needs to be built with a specific creation command. In this case, the user must clarify a workspace name and define the workspace path in the system.

Moreover, if the path, does not exist or is inputted and if the workspace name is already being used, the console of the Prosci, will show up an error message with the correct command format.

Once the application is started, the user needs to define a workspace. A workspace is a working directory, where the user can save and edit the files. Only if the files are saved within the defined directory, they can be traced and reproduced. Every workspace works independently. They don't have any interactions with each other.The user can enter an existing workspace, or the user can create a new one with the following command:

workspace [workspace name] [workspace path]

All workspaces which exist in the system are stored under prosci.properties in the home directory. The user is able to switch the workspace with the following command:

workspace [workspace name]

As every works working independently, all of them have their own main directory.The users can create their workspaces according to their demand.

A workspace is a fundamental element of the project. If a workspace is created, this freshly created workspace, it will be defined automatically as the current workspace. Only if the user wants to change workspace, they can run the setup workspace command to redefine workspace.

In a word, all further executions happen under the current workspace. The details of this use case are demonstrated below in the Table 4.1

Use Case Identification	
Use Case ID:	1.1
Use Case Name:	Create workspace
End Objective:	A workspace is created with specific name and path
User/Actor	User of the Prosci application
Trigger:	The user enters the workspace creation command
Frequency of Use:	Every time the user wants to create a new workspace. The creation is mandatory if the user starts the application for first time

Preconditions		
The Prosci software package is successfully installed and the Application Console is started		
Basic Flow: The user starts the Application Console, enters the workspace creation command with correct parameters and presses the enter key.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further commands
2	The user enters the workspace creation command with correct parameters: "workspace [workspace name] [workspace path]" and presses enter.	The workspace is created with the given name and path, and the related information is saved in the Prosci system property file – prosci.properties in the home directory of the software agent.
Alternate Flow1: Print the help menu		
Alternate Flow2: Setup current workspace		
Alternate Flow3: Show worksapces		
Exception Flow: Once the user doesn't enter the command with the correct parameters or the workspace name or path is already existing, both an error message and a message which contains the correct command will be shown in the console.		
Post Conditions		
1. A workspace is created under the given path with the given name. 2. In the Prosci system file, the new workspace information is recorded. If the user creates a workspace for the first time, a prosci.properties file will initially be created under the user home directory.		
Includes or Extension Points		
1. Start xterm 2. Save files manually 3. Start graphic visualizer 5. Setup current workspace		

Table 4.1: Use Case 1.1 Create Workspace

4.1.2 Setup current workspace

Of even greater appeal, the "setup current workspace" command is an included step of the "create workspaces". The pre-condition for this use case is that the system already has some workspaces. This use case makes it possible, for the user to switch between different workspaces. As we all know that, each workspace has its independent domain and working directory, they do not conflict with each other. So the user can freely change the project without losing important data.

The command for this function has the same prefix "workspace" as the "create workspace" use case. The difference between those two use cases is that for "setup current workspace" you don't need to specify a system path for the directory creation. The workspace is recognized through the workspace name.

Only if the workspace is defined, further execution steps can be done. The Table 4.2 shows us the details of this use case.

Use Case Identification		
Use Case ID:	1.2	
Use Case Name:	Set current workspace	
End Objective:	Define the workspace domain, where the user can dynamically work on it.	
User/Actor	The user of Procsci	
Trigger:	The user enters the setup current workspace command	
Frequency of Use:	Every time the user starts the application, a current workspace needs to be defined.	
Preconditions		
1. The Prosci software package is successfully installed and the Application Console is started. 2. The workspace name given by the user must have already exist in the system.		
Basic Flow: The user starts the Application Console, enters workspace setup command with a specific workspace name, which already exits in the system.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for the further commands

2	The user enters the workspace setup command: "workspace [workspace name]" and presses enter.	The system switches to the given workspace, all further processing steps in the given workspace will be recorded.
Alternate Flow1: Print the help menu		
Alternate Flow2: Create workspace		
Alternate Flow3: Show worksapces		
Exception Flow: If the workspace setup command is not entered correctly or the given workspace name doesn't exist, an error message is printed to advise the user on how to move on.		
Post Conditions		
1. Workspace is defined as a given one, further executions under this workspace will be recorded. 2. system property – prosci.properties changes the current workspace to the given one.		
Includes or Extension Points		
1. Start xterm 2. Save files manually 3. Start graphic visualizer		

Table 4.2: Use Case 1.2 Setup Current Workspace

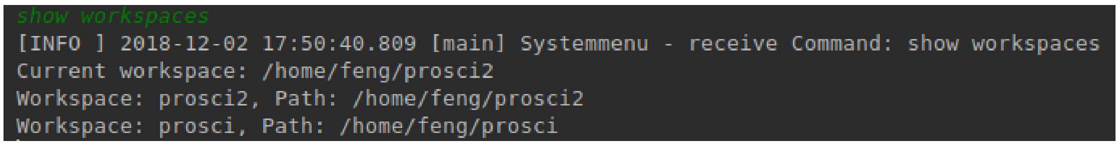
4.1.3 Show Workspaces

Equally important is the ability to show all existing workspaces in the Prosci system. The next use case is one, which makes it possible to present our Prosci user with an overview of all workspaces in the system. With the help of this function, the users can control and manage the workspaces according to their needs.

Even though the users don't define the current workspace, they are allowed to run this command. If the user has already set up the workspace during the last run, that information is also going to be printed. A possible output of this use case is illustrated in the Fig. 4.2.

As we can see from the figure, all existing workspaces are presented with their name and path.

The details about this use case are described in the Table 4.3



```
show workspaces
[INFO ] 2018-12-02 17:50:40.809 [main] Systemmenu - receive Command: show workspaces
Current workspace: /home/feng/prosci2
Workspace: prosci2, Path: /home/feng/prosci2
Workspace: prosci, Path: /home/feng/prosci
```

Figure 4.2: Main user cases of the Prosci

Use Case Identification		
Use Case ID:	2	
Use Case Name:	Show workspaces	
End Objective:	Show all existing workspaces in the system. If the current workspace is already defined, it will also be printed.	
User/Actor	Users of the Prosci application, who want to know the existing workspaces in the system	
Trigger:	The user enters the show workspaces command	
Frequency of Use:	Normally, it happens at every time the user starts the application.	
Preconditions		
The Prosci software package is successfully installed and the Application Console is started		
Basic Flow: The user starts the application console, enters the show workspaces command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the instructions
2	The user enters the show workspaces command: show workspaces and presses the enter key.	The console prints all existing workspaces in the system, if there is a current workspace defined in the prosci.properties file, it will also be printed.
Alternate Flow1: Print the help menu		
Alternate Flow2: Setup current workspace		

Alternate Flow3: Create workspace
Post Conditions
1. All workspaces, which exist in the system are shown in the console. 2. If a current workspace is defined, it is shown with the prefix: "current workspace"
Includes or Extension Points
1. Start xterm 2. Print help menu 3. Save files manually 4. Start graphic visualizer 5. Redefine workspace domain 6. Create new workspace

Table 4.3: Use Case 2 show Workspaces

Use Case Identification		
Use Case ID:	3	
Use Case Name:	Show help menu	
End Objective:	Show all possible commands and their correct format which are defined in the system.	
User/Actor	Users of the Prosci application, who want to know the correct system commands.	
Trigger:	The user enters the help command	
Frequency of Use:	Every time the user needs help getting to know the correct commands and all command options.	
Preconditions		
The Prosci software package is successfully installed and the Application Console is started		
Basic Flow: The user starts the application console, enter the help command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the commands
2	The user enters the show workspaces command: help and presses enter key.	The console prints all possible command options and their correct formats.

Alternate Flow1: Show workspaces
Alternate Flow2: Setup current workspace
Alternate Flow3: Create workspace
Post Conditions
1. All possible command options and their correct formats are shown in the console.
Includes or Extension Points
1. Start Xterm 2. Save files manually 3. Start graphic visualizer 4. Define workspace domain 5. Create new workspace 6. Show all existing workspaces.

Table 4.4: Use Case 3 Show help menu

Use Case Identification		
Use Case ID:	4	
Use Case Name:	Start Xterm	
End Objective:	Start the Xterm, the standard terminal emulator, on which all following executions in the workspace are recorded and are written into the log files for further provenance and re-production purposes.	
User/Actor	Users of the Prosci application, who want to trace their scientific workflow.	
Trigger:	The user enter the start xterm command	
Frequency of Use:	Every time the user wants to do some scientific experiment and hopes to save the workflows.	
Preconditions		
1. The Prosci software package is successfully installed and the Application Console is started. 2. The current workspace is already defined. 3. The Xterm software package is installed on your system.		
Basic Flow: The user starts the application console, enter the start xterm command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the commands

2	The user either creates a new workspace or defines an existing one as the current workspace	1. If the user creates a new workspace, after the successful creation, the current workspace is automatically set as the newly created one. 2. if the user sets the workspace as one, which already exists, the system property file – prosci.properties will change the current workspace into the defined one.
3	The user enters the start xterm command: start and presses enter key.	A new prompt console (Xterm) is started, our Prosci tool is ready for tracing everything that happens ,in the xterm terminal.
Alternate Flow:		
1. Save files manually 2. Start graphic visualizer 3. Define workspace domain 4. Create new workspace 5. Show all existing workspaces.		
Post Conditions		
The Xterm terminal is opened, on which the user can do any kind of scientific experiment. Workflows and their inputs and outputs are going to be traced and recorded, so that the provenance of the scientific workflow is enabled.		

Table 4.5: Use Case 3 Start Xterm

Use Case Identification	
Use Case ID:	5
Use Case Name:	Save files manually
End Objective:	In the current workspace, the user is able to add files manually, and the files, which add by the user manually. These files will also be traced for reproducing purposes.
User/Actor	Users of the Prosci application, who want to trace their scientific workflow.
Trigger:	The user enters the starts xterm command
Frequency of Use:	Every time the user wants to do some scientific experiment and hopes to save the workflows.

Preconditions		
1. In the Prosci application, the current workspace is defined. 2. The Xterm software package is installed in the system.		
Basic Flow: The user starts the application console, adds the files into the current workspace and runs the save file command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the commands
2	The user adds the files manually into the current workspace.	———————————
3	The user enters save file command: save and presses enter key.	The files which were previously added by the user manually are saved in the system with a separate commit ID
Alternate Flow:		
1. Save files manually 2. Start graphic visualizer 3. Define workspace domain 4. Create new workspace 5. Show all existing workspaces.		
Post Conditions		
The files which are not produced by the workflow and are manually added into the current workspace are saved in the system. A separate commit ID is generated. The files are ready to be called and reproduced through our graphic visualizer.		

Table 4.6: Use Case 5 Save files manually

Use Case Identification	
Use Case ID:	6
Use Case Name:	Start graphic visualizer

End Objective:	The graphic-visualizer tool is started. on which the user can do some further investigation. For example, they can re-run the commands from the historical workflow backlog or they can reproduce the files that were overwritten during the experiment, because the versioning of the historical files is enabled. Basically, they can view a graphic picture, on which the ontology description of the current workspace is illustrated and their detailed information is listed in tabs.	
User/Actor	Users of the Prosci application, who want to see an overview and details of the scientific workflows.	
Trigger:	The user enters the starts graphic-visualizer command	
Frequency of Use:	Every time the user wants to do get the overview of the scientific workflows.	
Preconditions		
1. In the Prosci application, the current workspace is defined.		
Basic Flow: The user starts the application console, and starts the graphic-visualizer tool by running the command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the commands
2	The user enters the start graphic-visualizer command: graphic visualizer and presses enter.	A new windows is opened, which includes an overview picture of all workflows in the system. All details are shown in tabs. The system is ready for the user commands such as rerun and reproduce.
Alternate Flow:		
1. Save files manually 2. Print help menu 3. Define workspace domain 4. Create new workspace 5. Show all existing workspaces. 6. Start Xterm		
Post Conditions		

A new window is opened, in which the user is able to view all historical commands and their input and output details. At the same time, an overview picture of the workflows is shown. In the new window, the user can also do some further investigation, such as rerun the historical command and reproduce the overwritten file.

Table 4.7: Use Case 6 Start graphic visualizer

In one word, the use cases, which are designed for operating on the workspaces in Prosci are intended to limit the monitoring range to the system. We are not going to purposelessly trace all incoming and generated files, intermediates etc. within the system. Instead we are thinking about a solution to limit the provenance tracing procedure to a specific location.

Besides the workspace, Xterm acts within the project as a bridge to connect the usual command executions and tracing and recording without any extra impact on the comfort of the usage of the tool.

The last use case "Start Graphic-Visualizer" is the starting point of user interaction and the beginning of analysis on the provenance ontology.

4.2 Description of the Components

The Prosci system consists of three components. They are the Application Console, the File Monitor, and the Graph Visualizer.

As shown in Fig. 4.3 below, we still have five external software agents. Those are Xterm, JGit, Strace, Oshi, and ProvToolbox. All the foreign software agents are highlighted in pink in the Fig. 4.3.

Xterm is a standard terminal emulator. It establishes a terminal environment, from which the users can execute commands for their scientific experiment, and it seems like the user works on a real terminal environment. At the same time, Strace will also be started to log the system's behavior. Strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and control with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. System behavior is recorded by Strace is stored under the workspace directory [workspace name]/prosci/trace/log. All log files will be engaged for further processing of the ontology structure. The Graph Visualizer will call up the generated ontology structure. Through it, the Graph Visualizer is, therefore, able to illustrate the graphic ontology structure and details of the execution.

Oshi has only one task. It records the system info comprehensively, such as model, serial number, manufacturer, version, name, etc. of the computer system. Following this, the information about the processor, memory, CPU, running processes and disks is determined. Similarly, with regard to the hardware aspect, the displays and devices are detected.

The collected information by the Oshi will then be saved in the PROV-O as an agent, the related details about the software agent will be filtered and are then going to be presented.

Simultaneously, the second primary component of the Prosci is awakened. From that moment, the File Monitor starts to control all incoming log files and save all newly added files which appear in the workspace. For this purpose, the File Monitor uses JGit to realize it. JGit is a pure Java framework of the Git version control system. JGit is more than just a Java library for working with the Git repository, and you can integrate it into an existing application or create a tool. Further information and description of JGit are given in section 4.1.1.

In other words, the Application Console is a central menu tool, and the users can select the functions they need through our central application console. It is acting as an organizer. It arranges the component-call to the corresponding target. At the same time the Application Console has the help menu, with the help of it. Using it the user can print the help menu and find out useful information from the given message to understand the options provided by the Prosci system.

As mentioned before, the File Monitor is a monitoring tool to record incoming logs and runs the Git commands. It traces the system log directory, and monitors the system working directory. Once the new registration appears in the system log directory, it will check the working directory for changes that need to be saved.

The Graph Visualizer is a visualization tool, which enables the user to build an overall view of the provenance tracks. It presents the user with a graph, which contains an ontology structure and as the sub-functions. The users can view all produced files and its related connections. With its help, the user can also reload data, which may have been overridden due to the naming convention and the user can also rerun the commands from the past command catalog. This part of these functionalities are accomplished by the help of the external toolbox "ProvToolbox". Precise information is given in section 4.3.3.

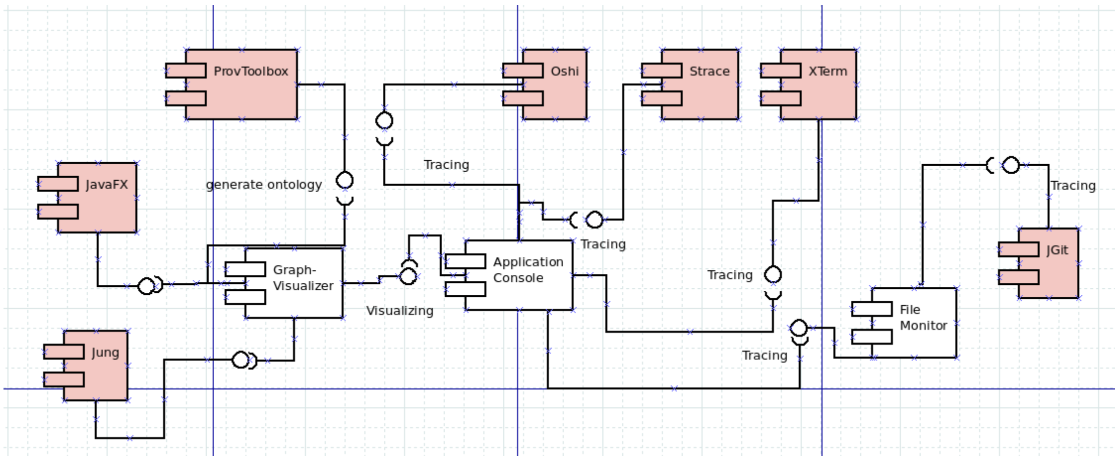


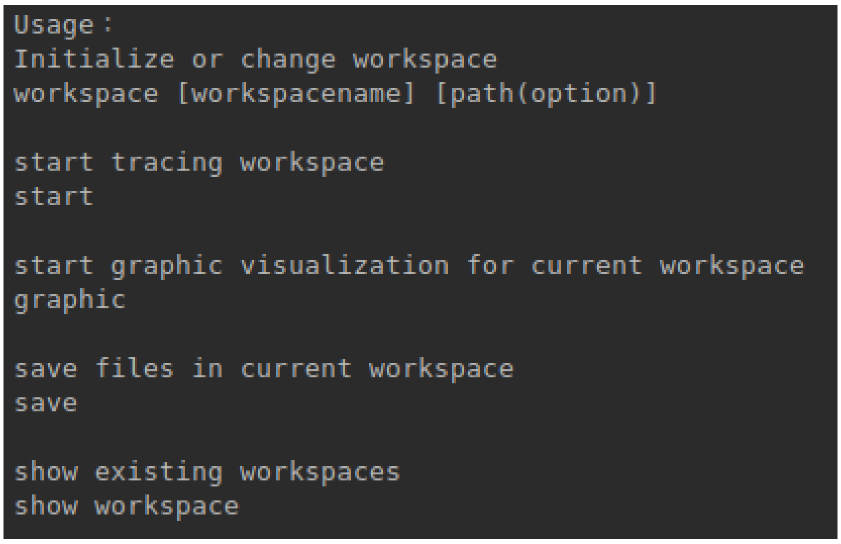
Figure 4.3: Main Components of the Prosci

4.2.1 Application console

In this section, I am going to demonstrate the usage and the functionalities of the application console. As the application console is acting as a central organizer of the Prosci System, it plays an important role. Once the system is starting, it is waiting for the first command. As described in section 4.2 our application is working based on the workspace unit. Users of Prosci must first define the workspace domain. Prosci supports some of the helper functions to make the user familiar with the main functionalities of Prosci. For example, the user can print the help menu with the command:

help

or the user can print all workspaces in the system and based on the information given by the system can choose the corresponding workspace they want to work on. The Fig. 4.4 shows us the help information.



```
Usage:
Initialize or change workspace
workspace [workspacename] [path(option)]

start tracing workspace
start

start graphic visualization for current workspace
graphic

save files in current workspace
save

show existing workspaces
show workspace
```

Figure 4.4: Help menu of the Prosci

As shown in Fig. 4.4, we know that the Application Console supplies us with five options. Now I am going to present the internal interaction between the components for each choice and the working sequences of the Application Console, so that you can get familiar with the primary function of the Prosci application.

Initialize or change workspace domain

We already know that the workspace is the core of Prosci. The first step to initiate the tracing process is to set up a workspace domain. The Fig. 4.5 demonstrates the basic workflow of Prosci.

Once the user declares that a new workspace domain should be created, he needs to enter the workspace name and its location. There also is another option: he can define the current workspace with an existing one. For this purpose, he doesn't need to enter the workspace location, he only needs to enter the workspace name of an existing one as the parameter. In both cases, the component Application Console will call its own method: "workspace". The workspace command is operating under specific criteria. We list them as the following points:

In all cases, the system property file should be available. If it doesn't exist, creating a new workspace domain is mandatory. More specifically, the user must declare the workspace with the name and the preferred location. If the given parameters are not consistent with the path parameter, an error message will be thrown.

- The user wants to create a new workspace domain:
  - Is the system property file available?
  - Did other workspaces register the workspace name?
  - Is the given workspace path valid?
- The user wants to define an existing workspace as current workspace:
  - Is the system property file available?
  - Does the given workspace name exist in the system property?

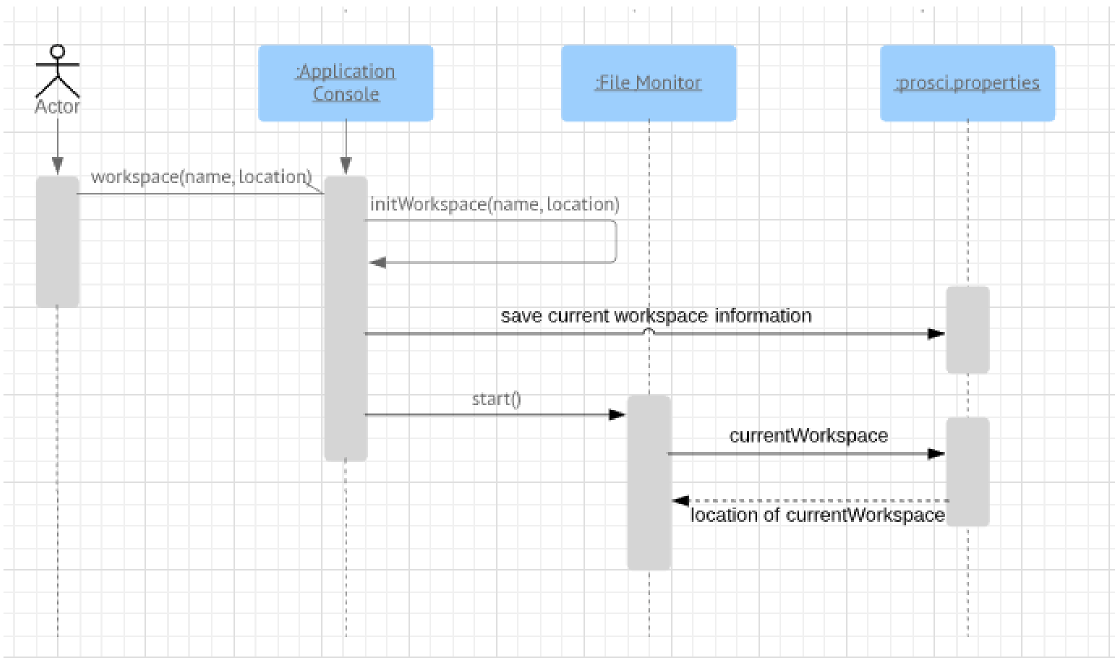

Actor
:Application Console
:File Monitor
:prosci.properties
workspace(name, location)
initWorkspace(name, location)
save current workspace information
start()
currentWorkspace
location of currentWorkspace

Figure 4.5: Initialze or change workspace domain

Fig. 4.5 is a sequence diagram of the workspace manipulation. Four components are involved in this activity. The client enters the instruction, and the Prosci system receives it, and upon receiving it, the Prosci system then calls its own class to check the preconditions of the creation of a new workspace.

If all the requirements are fulfilled, then the workspace will be created, after completion of the generation of the newly defined workspace is successfully executed, as the next rule, the calling will be forwarded to the second component of the Prosci system.

The component File Monitor later requires to check out the current workspace definition by reading in the system properties prosci.properties. The system property by default is generated under the "user.home" directory. Usually, this verification of the current workspace won't cause any error handling. Because the calling is forwarded only after the pre-condition that the workspace is successfully setup.

Fig. 4.6 shows us with a detailed overview of the workflow of the workspace manipulation.

The client enters the command, the Application Console first checks, if the path parameter is consistent. If so, the system generates a new workspace. Otherwise, an old workspace, which must be generated before will be set as the current workspace.

For creating a new workspace, we first check if the system property file is already there under the "user.home" directory, for the case. If there is no prosci.properties under "user.home", we generate a new one. After successfully finding the system properties file, we are going to verify the validity of the given path. If the directory path is already being used, an error message is thrown. If not, the new workspace with the provided path will be created, and the current workspace domain in the system property file is written as newly created one.

For switching to the existing workspace, the system, in any case, needs to check the existence of the system property file. If there is none, an error message will be shown. Oppositely, the current workspace definition in the system property file is written with the given workspace name, if it is possible to find it in the prosci.properties.

In both cases, as the final step, the system calls the File Monitor component, and it then starts monitoring any incoming changes under the current workspace domain. Detailed information about the working progress of the File Monitor is discussed in the next section.

Last but not the least: the structure of the generated workspace domain. Fig. 4.7 depicts the structure of the created workspace. The project has the name "myWorkspace". Under

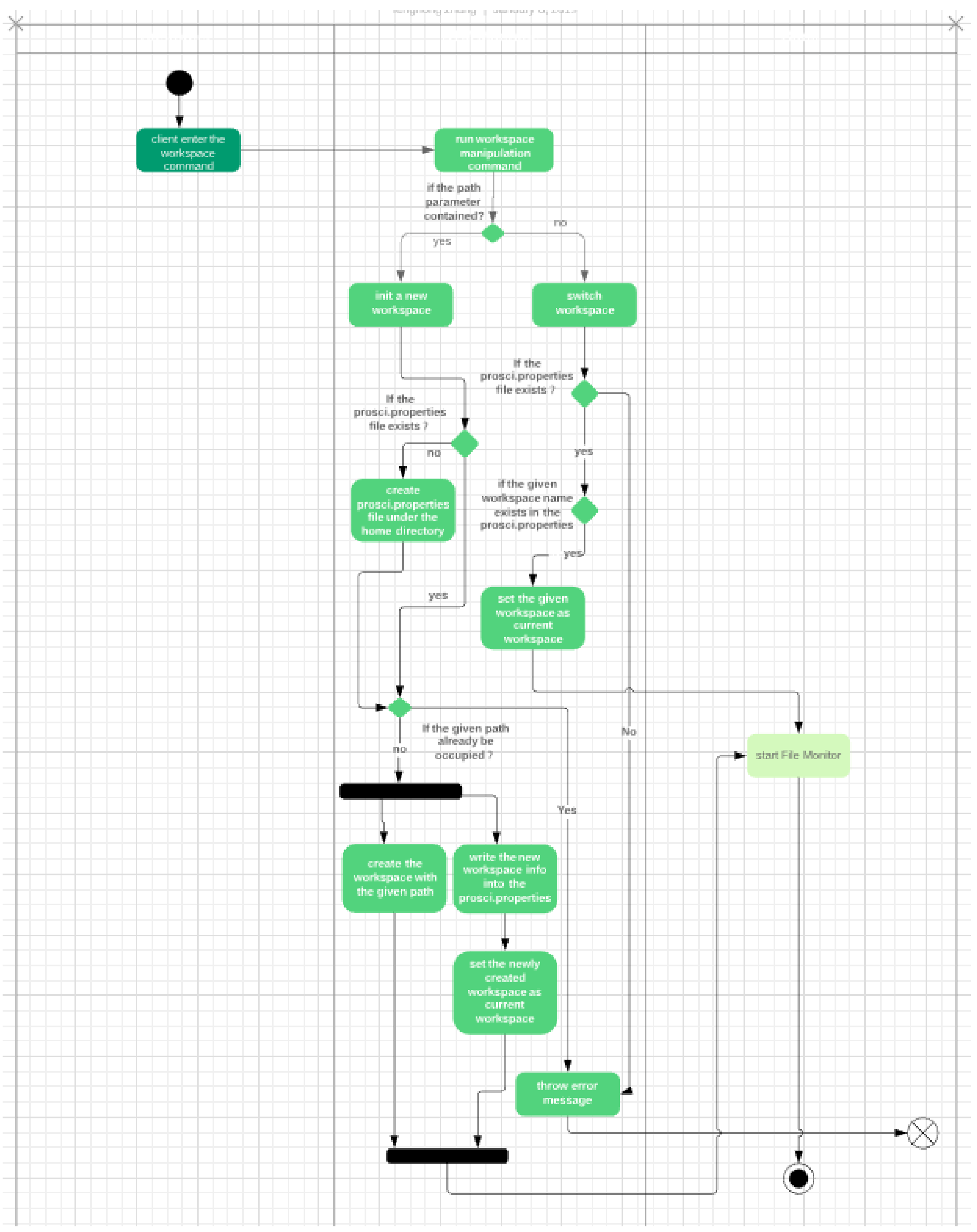

client enter the workspace command
run workspace manipulation command
if the path parameter contained?
yes
no
init a new workspace
switch workspace
If the prosci.properties file exists ?
If the prosci.properties file exists ?
no
yes
create prosci.properties file under the home directory
if the given workspace name exists in the prosci.properties
yes
yes
set the given workspace as current workspace
If the given path already be occupied ?
no
No
Yes
start File Monitor
create the workspace with the given path
write the new workspace info into the prosci.properties
set the newly created workspace as current workspace
throw error message

Figure 4.6: Activity diagram: Initialize or change workspace domain

it, two directories are created. The "input" is the actual working directory, where the files, data, scripts, graphs, etc. are stored. The user can manually add them into the directory, or they may be generated during the scientific experiment.

In the meantime, a second directory called "prosci" is created. This directory is responsible for all the systematical records and executions. Under it, another three folders are appearing. The functionalities of them are listed below.

- prov: under this folder, the compiled provenance ontology structures of the experiment from the Graph Visualizer are saved. Commonly, those are graph of the structure and an XML file. The XML file is the transformed provenance structure of the workflow of the scientific experiment. The regulation and the criteria of the transformation are discussed in section 4.3.3.
- trace: this is the location where all the tracing results are stored. The tracing results are obtained by using two external tools and one individual component "File Monitor". External tools used are Strace,and Oshi. They generate the process log files and the system info for each execution. And internal "File Monitor" takes the responsibility to write the command.txt file. Usage and formatting of this file are described in section 4.3.2.
    - log: this folder collects the generated log files of each system call by Strace.
    - systeminfo: this folder gathers all analyzed system information whenever a new tracing command is triggered.
    - command.txt: the File Monitor maintains this .txt file. The usage and formatting of this file are described in section 4.3.2.
- version: this specific folder is the location where the user gets access to various versions of the files in the "input" directory. It can be an old version which is overwritten due to the naming convention and it can also be the latest one, which currently can be found under the "input" directory.

Start tracing workspace

We now going to talk about the essential function of the Prosci program–start tracing workspace.

The sequence diagram from Fig. 4.8 gives us a rough overview of the steps for starting the tracing process. Once the application gets the instruction for starting tracing, it first

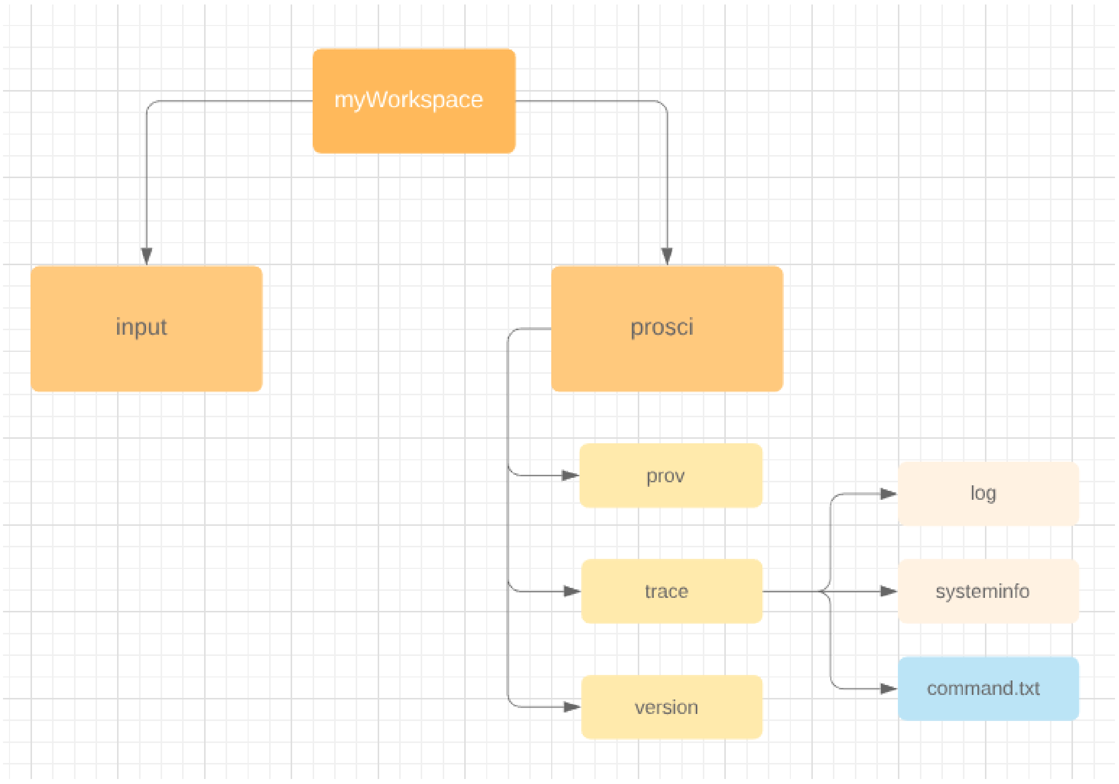


Figure 4.7: The workspace structure of a sample workspace

calls its embedded class "start". This class must finish three tasks.

To begin with, it calls Oshi, the external tool for collecting system information. The software agent information includes not only software but also hardware level information. Below is a sample system information collections of a software agent.

Below, all the information, which is collected after the analyzing from Oshi is listed.

- Computer System
  - manufacturer
  - model
  - serial number
  - firmware
    - manufacturer

        * dmi
        * version
        * release date
    - baseboard
        * manufacturer
        * model
        * version

- Processor
    - Identifier
    - ProcessorID
- Memory
    - Memory capacity
    - Swap used
- CPU
    - Uptime
    - percentage occupation
    - CPU load
    - CPU load averages
    - CPU load per processor
- Processes
    - number of processes and threads
- Operating System
    - Name
    - Version/Build
- Disks
    - size
- FileSystem
    - File Descriptors
- Network interfaces
    - Name

  - MAC Address
  - MTU:
  - IPv4
  - IPv6
  - Traffic

- Network parameters
  - Host name
  - Domain name
  - DNS servers

- Displays
  - Manuf. ID
  - ManufDate
  - Preferred Timing
  - Manufacturer Data
  - Unspecified Text (e.g. LG Display)

- USB Devices

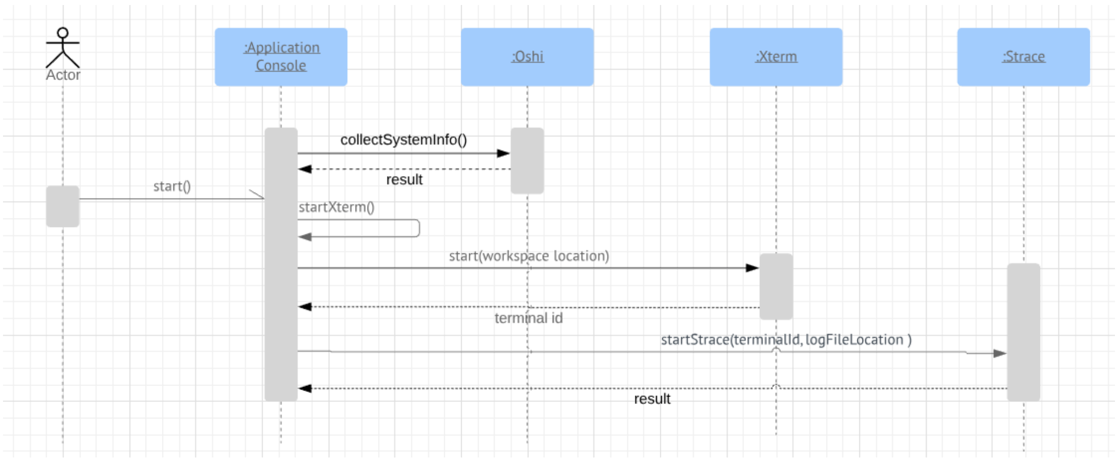


Figure 4.8: Start tracing workspace

This collected system information is then saved under the workspace directory "[workspace name]/prosci/systeminfo". Whenever the application is started, the Application Console triggers the collection of the information, but during the lifetime of the Application Console the system information will only be collected once. Even though such information is collected every time when Application Console starts, the information will only be shown to the user in the GUI if it varies from that already collected.

The next task of the program is to call the Xterm tool. Before this, Prosci calls the system property file first, for reading the current workspace domain. After the workspace domain is verified, the request to start Xterm is sent. Xterm simulates the actual terminal. Using that, the user can run any kind of execution just like they are working on a common terminal. This call to the Xterm is return with a message which contains the processID of the Xterm terminal. This processID is reserved for further usage. The exact activities are illustrated in the Fig. 4.9.

Finally, we are ready to start the tracing process using Strace by ID from before. From moment on any system activity, howsoever small runs through the Xterm terminal and makes the effects such as writing, editing, and deleting of files, which happen in the workspace "input" directory, are recorded and the log files will be saved into the folder "[workspace name]/prosci/log".

Start Graph Visualizer

The process of starting the Graph Visualizer component of Prosci is much more simpler. The sequence of the workflow that starts the component is shown in Fig. 4.10. The Application Console using Runtime API makes the system call to start the component. Once the component is started a message for confirmation is obtained.

Show workspaces

Next, we are going to talk about the "show workspaces" command. We already know that the Application Console saves every new workspace into the system property file "prosci.properties", which is located under the user.home directory. The clue for getting all the information about the workspaces is trivial. Fig. 4.11 is a sequence diagram of this command. The only thing it does is to read the "prosci.properties" and show the workspace information into the console for the user.

Save files

The last feature, which we still want to mention is the manual saving of files using Application Console. The Application Console first reads the current workspace from the system property file and then the location of the git repository is sent to the Jgit API. Jgit checks if the repository do has some new incoming files or if anything was changed. If changes are found, JGit saves them.

Actually, the saving of the new files is not obligatory. Even if the user doesn't run this command, the system is going to save the new changes in the repository if some new executions happen.

Having a look in Fig. 4.12, we again describe the process with a sequence diagram.

4.2.2 File Monitor

Since we discussed the Component "Application Console" in the previous section, you may have already heard about the second Component "File Monitor". This component is the simplest one in the whole system.

The file monitor has only one responsibility. It is to monitor the Git repositories which are created under the workspace domain. Fig. 4.7 illustrates the fundamental workspace folder structure. We already mentioned it before, when we talked about the workspace creation process. Basically, there are a Git repositories and a log directory deployed into the workspace. Those are the "input repository" and the "log directory". The "log directory" traces all incoming new log files which are generated by the Strace tool during the Xterm working period. The Git repository is much more important, it is the "input repository". It controls the workspace working directory "input". It saves all newly added files, records file changes and reports it when filed are deleted. This repository is stored under the path "[workspace name]/input". The execution of the git command for this workspace is only triggered if the program finds some new incoming log files under the "log repository". As we all know, only if the command is running in Xterm, the log files for them are going to be generated and saved, in other words, the user must run his execution inside Xterm. Otherwise, nothing is going to happen. The only action which is allowed during the provenance process of the scientific experiment is saving files manually to the workspace input folder.

Obviously, "File Monitor" plays an important role in the Prosci system. The reason why we have this extra component is that we are using it to automatically watch the file entrance of the system, especially for the log files. And the entrance of the log files triggers the versioning of the workspace files. The description of the working steps and the communications between the input repository and the log directory of the "File Monitor" is illustrated in Fig. 4.13. This time we are using the flowchart diagram for the illustrating that purpose.

As shown in this figure, the "File Monitor" still first needs to check the availability of the workspace domain. Only if the current workspace under the system property "prosci.properties" can be found, then the "File Monitor" is going to be started. After discovering the current workspace, the "File Monitor" begins to initialize the relevant Git repositories, namely "input git repository". After finishing the initialization the tracing thread is started. It runs in the background and checks if there are new log files being written within the log directroy.

If the new entrance is detected, the "File Monitor" will review the "input repository". If new modifications are appearing, the Git commit command is executed.

At the same time, the commit version ID for the previous and current commit of the input git repository are also written into the command.txt. The reason why the previous commit ID is also being marked we will be explained in the next section.

In case of there not being an entrance in the log directory, the process will just be running the checking command for the directory status infinitely, until the Prosci application is terminated.

The main functionality of the File Monitor is implemented with the Java thread. Since we all Knowing that Java is a multi-threaded programming language, the thread enables java programs to share common resources concurrently for all the different parts. The File Monitor keeps running indefinitely after the declaration of the workspace domain. It stops only when the Prosci program is terminated.

4.2.3 Graph Visualizer

This is the last section for describing the system components of ProSci, but at the same time, the most crucial part of ProSci is going to be expounded.

The Graphic Visualizer at first glance shows us a graph of workflows of the respective experiment. Through it, we are able to know the execution steps of the scientific experiment. We obtain the information about the exact commands, which are being executed and also get to know what they do. Furthermore, the status of the OS is accessible. The user can view the Operating System status by clicking on the agent button in the dashboard of the Graphic Visualizer's UI. The Agent's information is shown with i.e. its unique ID and a short description. Equally important is the visible tracing of the file version. In other words, all files, which appear in the workspace are reproducible. Even if a file is overwritten due to the naming conversion, it is still possible to recover it and load it into the workspace directory.

First of the all, we want to talk about the main functions of the Graph Visualizer.

This component basically has five functions.

1. Show the workflow of the experiment with the ontology structure in a diagram. The properties of an ontology are files, activities and agents. The properties are represented as the node of the graph, they are distinguished by color.

2. Show all activities, which happened under the workspace domain, more accurately, all executions which were running within the Xterm windows will be shown there.

3. Show all files. The files there include the resource file and the target file. For the resource file, the auto-versioning is not yet supported. The user must run the save command in the Application Console to achieve the versioning purpose.

4. Show all agents. An agent in another word is the OS, during the execution of the experiment. The agent is determined only once, at the time we start the Xterm terminal.

5. Reload the historical data. The files, which have been overwritten and don't exist anymore under the current workspace domain are reloadable. After the user opens a certain file in the Graph-Visualizer, he has the possibility to restore the file by clicking on a button called "restore". Once the user clicks on it, the file is going to be stored under `[workspace]/prosci/version`.

6. Search for the element of the ontology presentation on the graph or search for the element under separate categories.

The implementation of the user interface is based on JavaFX. The UI component is partly designed using the JavaFX Scene Builder. The style of the UI was modelled using parts of CSS and some of them are embedded into the java code.

The Graph Visualizer is written in maven project. Within the Graph Visualizer, we have one extra package "prov". This module is employed for the reasons of ontology analysis. SFor details, see the next subsection "Prov".

For this goal, we use the "prov" module to analyze the saved log files. At the same time, the command.txt file which stores all information about the commit ID also has to be considered by this module. According to the reported XML file from the ProvToolbox we extract all useful information and demonstrate them and depict it using UI component so that the user is provided with a readable view of the hidden ontology.

Prov

Prov is a helper package within the Graphvisualizer. The main involvement of Prov in the workings of Graphvisualizer is the provenance metadata extraction from the collected logs and the construction of the provenance ontology by the integration, communication, pairing and summarizing between log files and the git repository.

The Prov package has only two classes, they are: "OntologyCreator" and "VersionChecker". The workflow of Prov is illustrated in an activity diagram in Fig. 4.14. As shown in the graph, OntologyCreator calls the VersionChecker to check out the repository under the input directory and retrieve all files in the repository and sort them according to the commit ID. After the files are sorted according to their version are ready to be read. OntologyCreator fetches the records from the command.txt, connects the logs with the respective files. The command.txt serves there as a dictionary. It aims to pair the command with the correct version of the files based on the information saved in the log. In Fig. 4.15, we take a look at example to make you understand the trails of the OntologyCreator.

This is a sample record line in the command.txt:

```
2019:03:28 21:15:00|yyyyyy|xxxxxx.log|000000|
```

We separate this record into four parts. "2019:03:28 21:15:00" is the timestamp of the record. "yyyyyy" is the package name of the log file and "xxxxxx.log" is the name of the log. This log is saved under the directory "yyyyyy" and "000000" is the commit ID. Using this commit ID we can find the corresponding generated files caused by this record. Let us assume that we generated three files by running the command, A.java, b.csv, and c.tex. The OntologyCreator analyzes this record, reads the xxxxxx.log and fetches those three files from git object with the correct commit version.

After the pairing process is completed, the OntologyCreator saves the command it extracts from the log into the provenance ontology as an XML file. Those arrangements are supported by the ProvToolbox API.

As a result, the provenance ontology sketch is prepared for graphic visualization.

Graphvisualizer GUI

Our visualization GUI is built upon JavaFX. At the beginning of this section, we have already mentioned JavaFX, a short description of it can be found in section 3.3. The five main features of the Visualizer GUI are represented above. The most important external library in the GUI component is the JUNG API (see section 3.7). By using JUNG, we create an interactive provenance ontology. The user can click on each note on the ontology graph and they will be redirected to a detailed page of the selected point. Fig. 4.16 is the GUI view of the sample project. Details about it can be found in the next chapter. The next two figures show the views encountered, once the user clicks on one of the agent nodes on the graph and once the user clicks the "Files" button in the dashboard respectively.

4.3 Summary

In this chapter, we demonstrated the basic use cases and the implementation of the Prosci project in detail. We limited the provenance tracing to a certain directory which is called "workspace", improved the usability by supplying the help menu and illustrating the abstract concept of provenance ontology as an interactable Graph-Visualizer.

The construction of the project is separated into three components. Application Console aims at making the tool easy-to-use and clearly and giving the user a clear understanding of all usages of the tool. File Monitor and the Graph-Visualizer answer our main research questions. They integrate several APIs so that the automation mechanism for the provenance information collecting is enabled, and the transformation strategy from provenance data to ontology definition later facilitates the interaction of the ontology and the analyzing process the computational task. Ultimately, we hope that the tool can improve the reproducibility of scientific experiments and answer the question of what kind of provenance information is necessary for the reproduction of the scientific experiment.

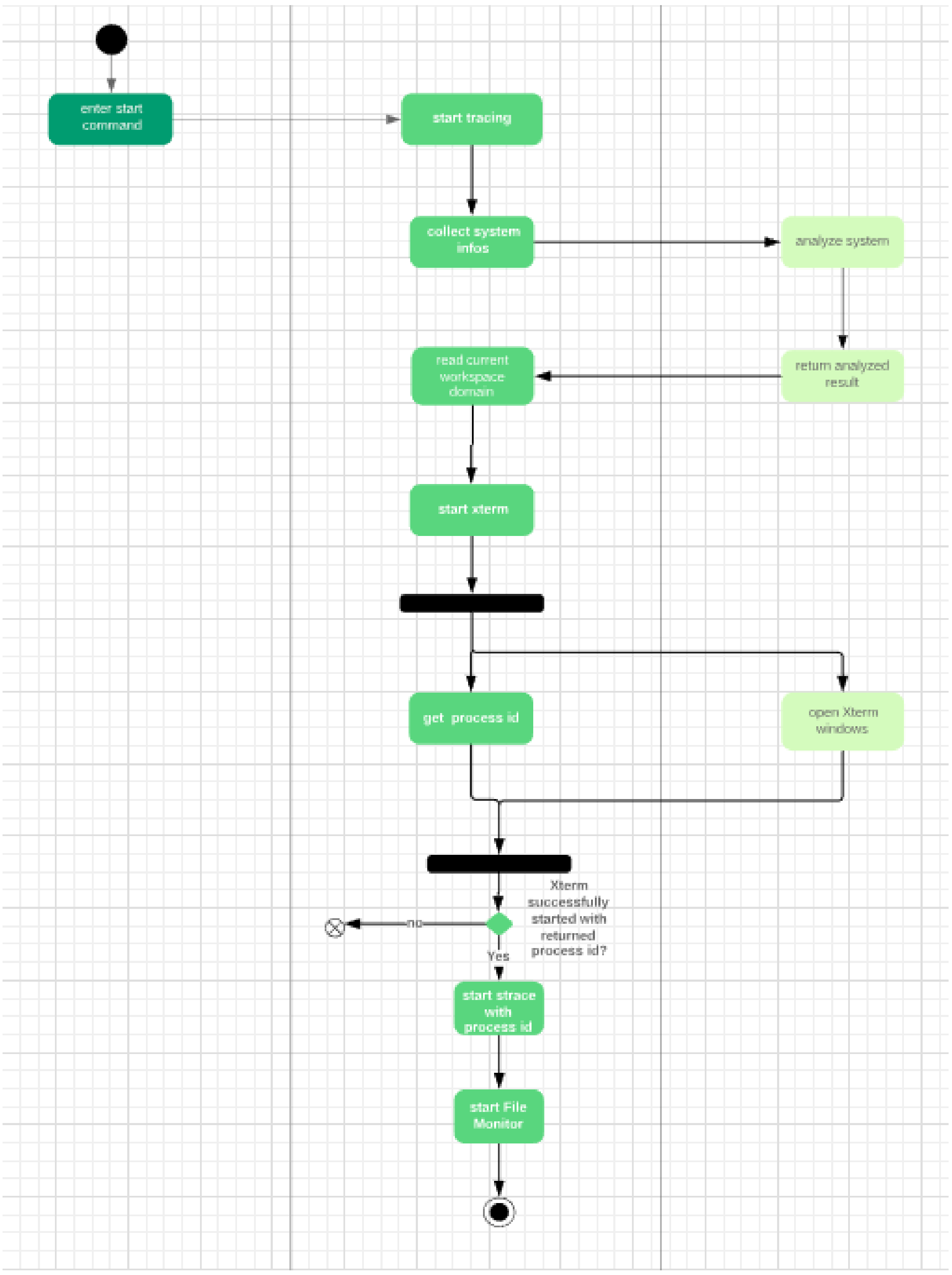


Figure 4.9: Activity Diagram: Start tracing workspace

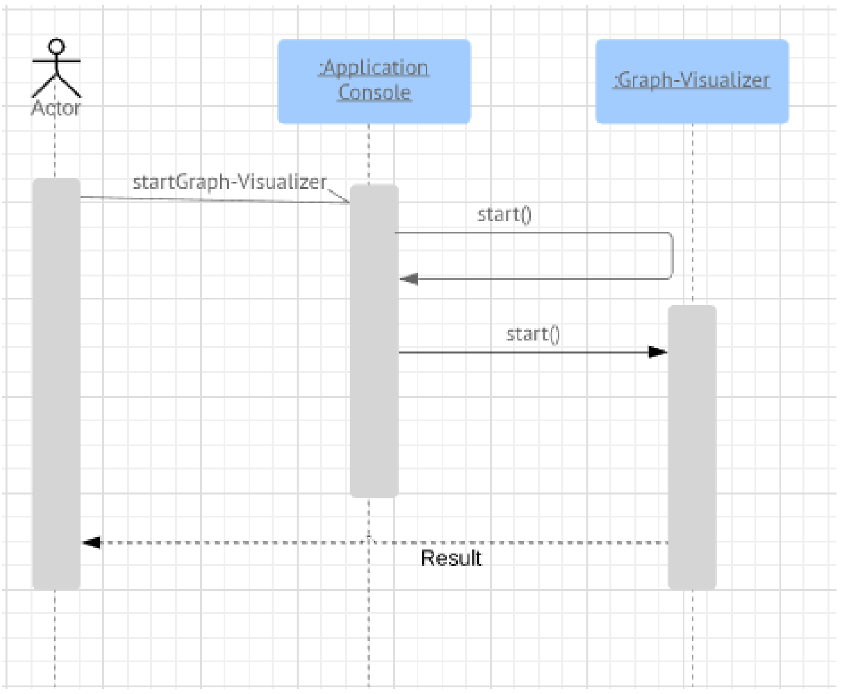


Figure 4.10: Activity Diagram: Start Graph Visualizer

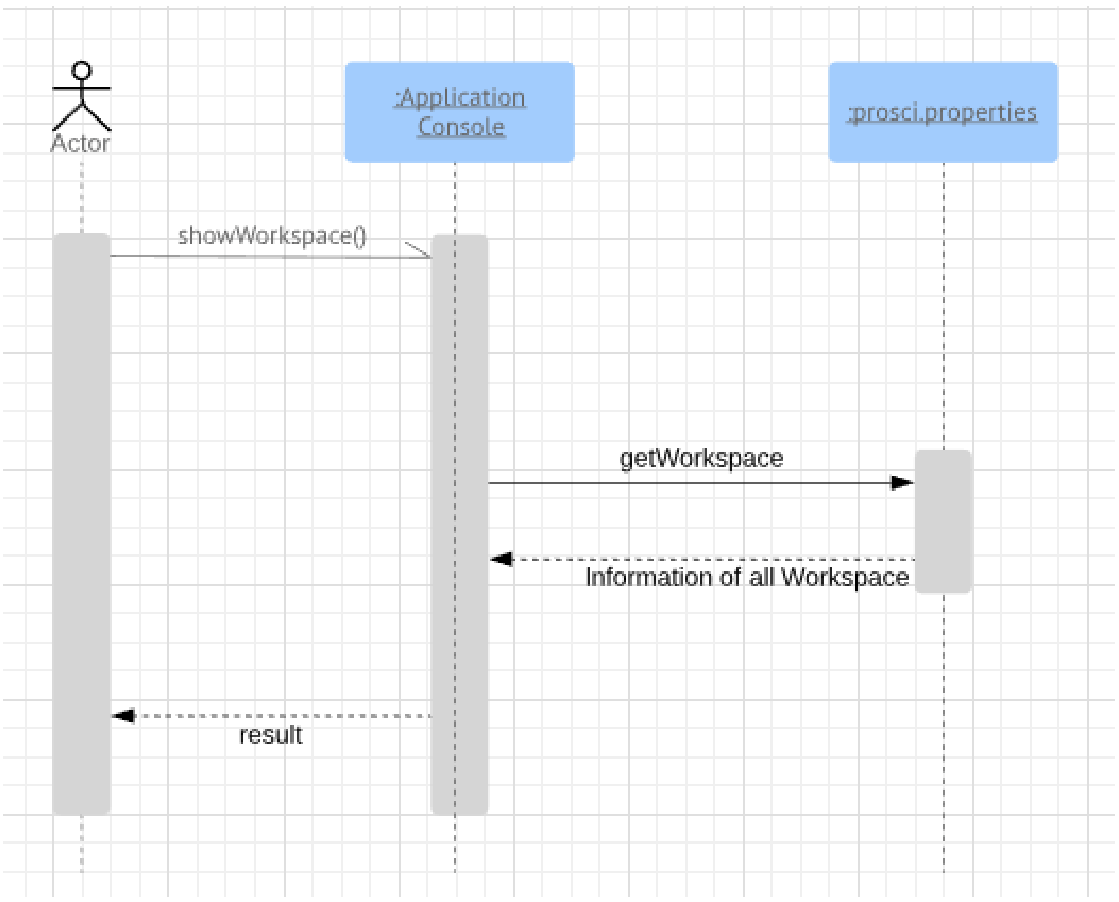


Figure 4.11: Activity Diagram: Show workspaces

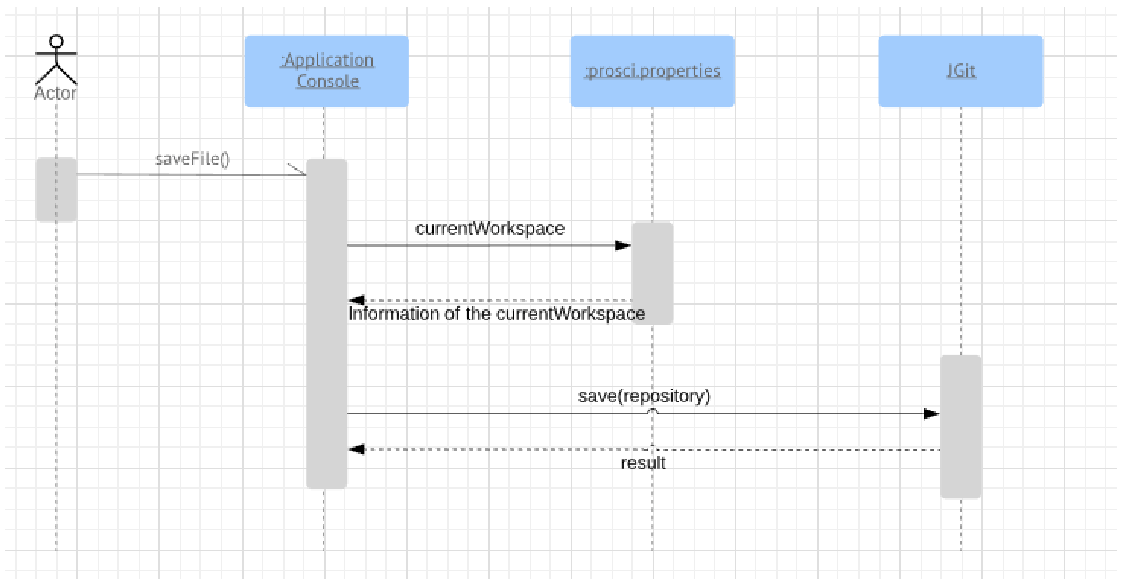


Figure 4.12: Activity Diagram: Save files

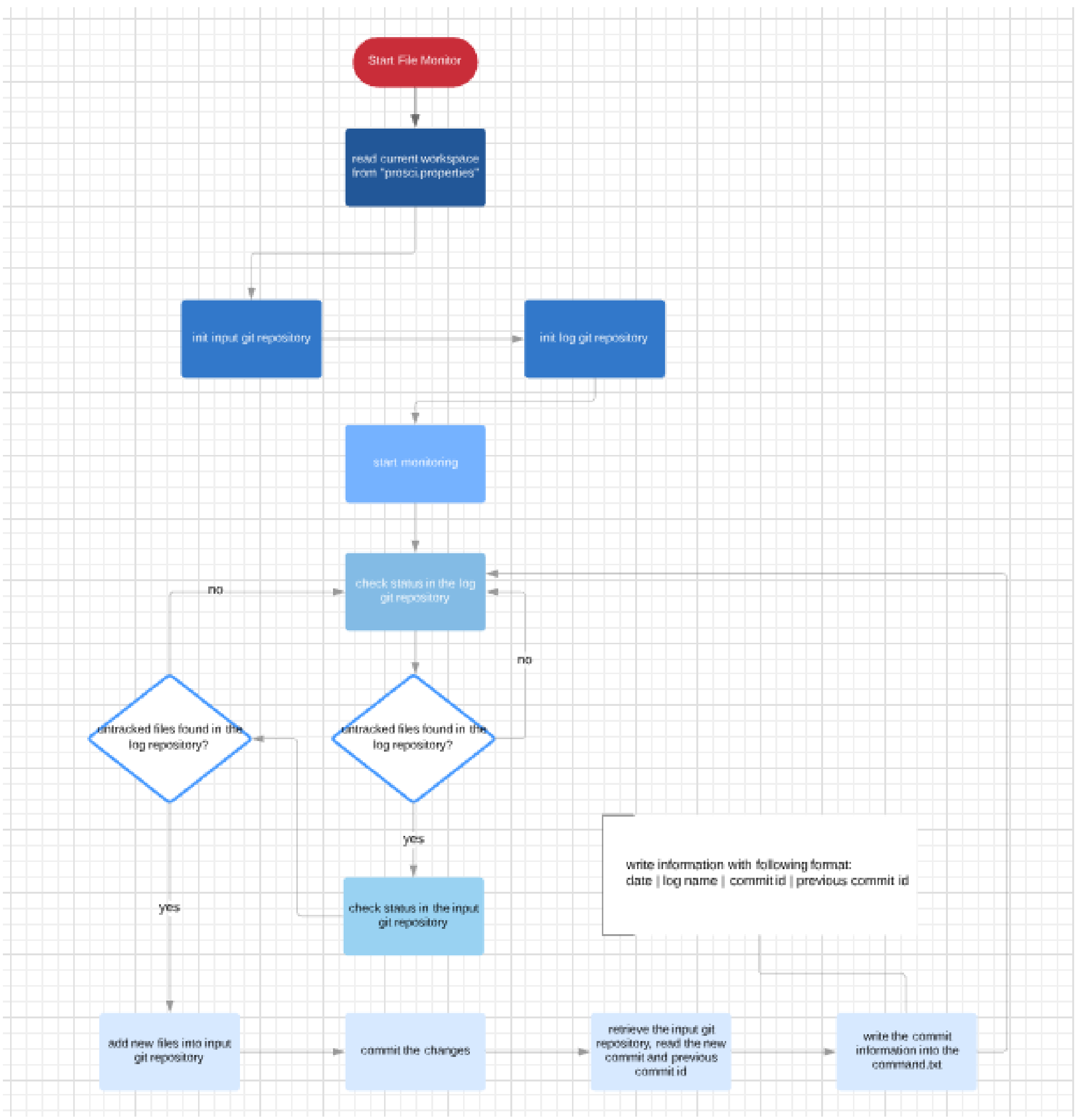
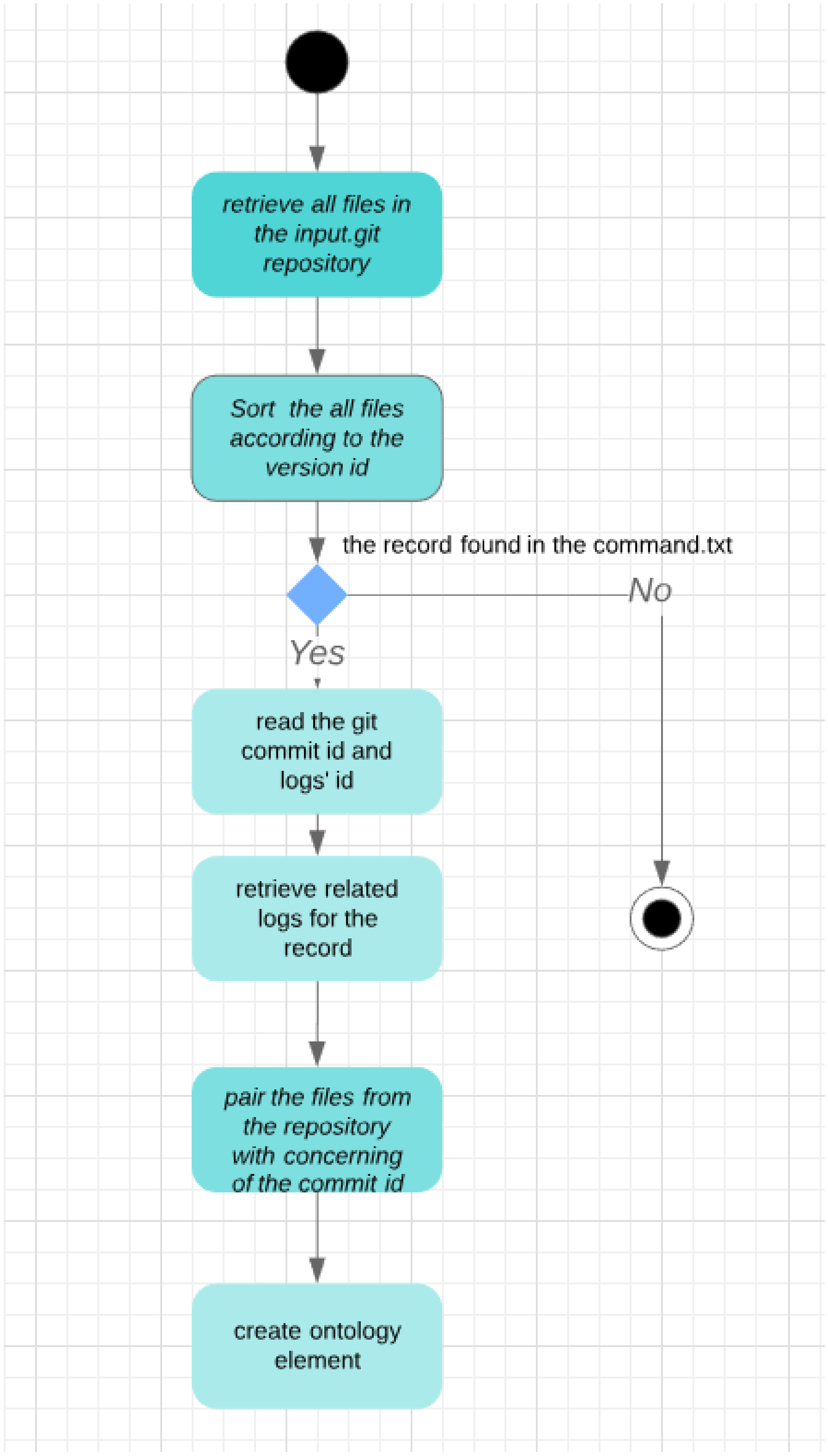

Start File Monitor
read current workspace from "prosci.properties"
init input git repository
init log git repository
start monitoring
check status in the log git repository
no
no
untracked files found in the log repository?
untracked files found in the log repository?
yes
yes
check status in the input git repository
write information with following format:
date | log name | commit id | previous commit id
add new files into input git repository
commit the changes
retrieve the input git repository, read the new commit and previous commit id
write the commit information into the command.txt

Figure 4.13: Flowchart Diagram: Workflow of the File Monitor


retrieve all files in the input.git repository
Sort the all files according to the version id
the record found in the command.txt
No
Yes
read the git commit id and logs' id
retrieve related logs for the record
pair the files from the repository with concerning of the commit id
create ontology element

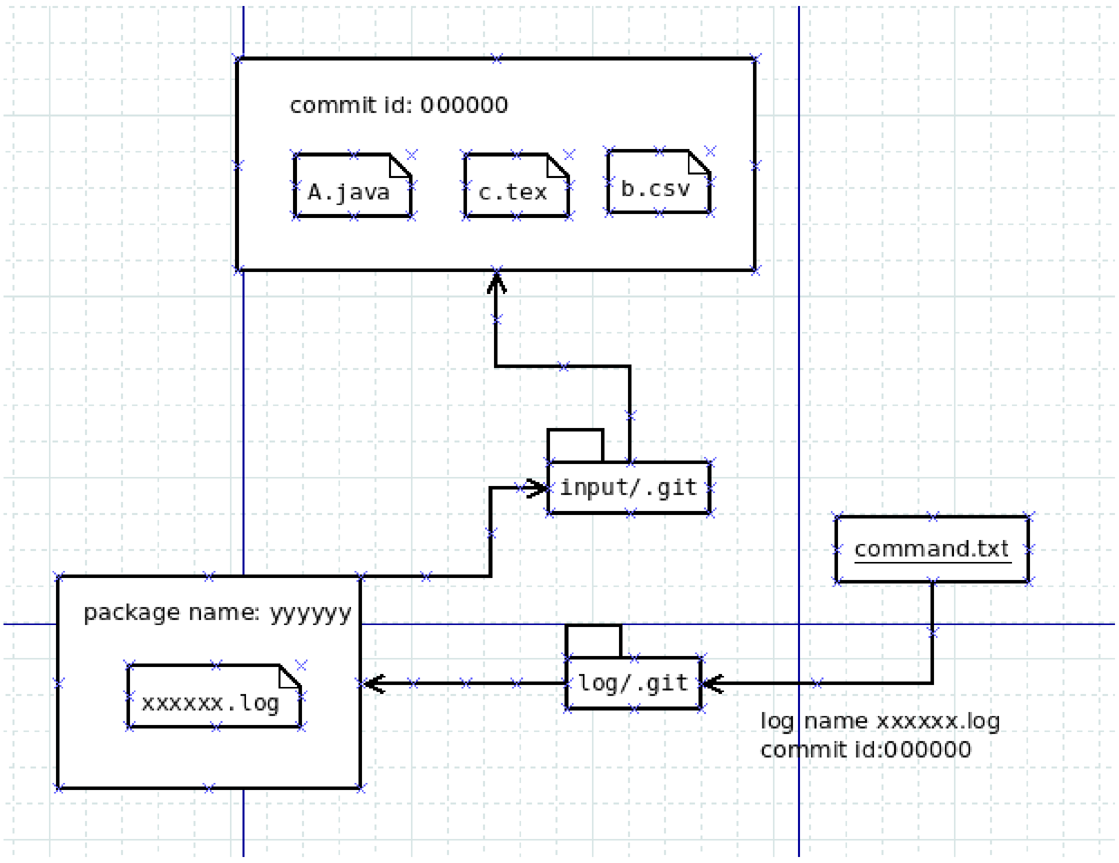


Figure 4.15: An example of the Prov working steps for a sample record `"2019:03:28 21:15:00|yyyyyy|xxxxxx.log|000000|"`

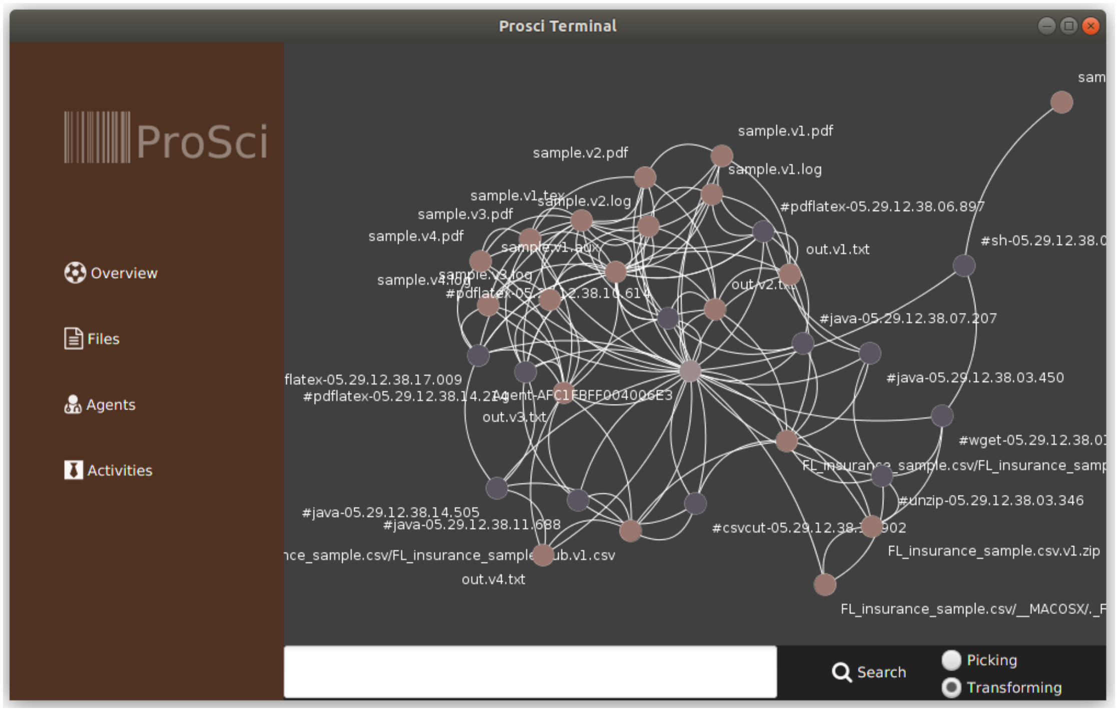


Figure 4.16: GUI view of Sample Project

CHAPTER 5

Sample Project

By now, all parts of the project have been shown to you. Last but not the least, we plan to replace the user guide with a ready to be used sample project.

In this chapter, we construct a sample scientific experiment project, to make you get familiar with the ProSci tool and teach you simulating a real solution. This sample project involves four steps. Each step is an independent workflow action. However, you still need to do some configuration and initial works to make the project run successfully on your machine. The details about the preparation are outlined in subsection 5.1. Take a look at it, follow the instructions and enjoy the final results produced by the process.

5.1 Preparation

Our sample project is included in our ProSci project package. First of all, please download the project from our Gitlab reporistory: `https://github.com/1425097/ProSci` Within the package "ProSci", you can find the source code and the packed .jar file. After the download, you need to unzip the "ProSci.zip" package and open the main directory, you will see three .jar packages, one prosci.sh script and a folder with the name "sample".

Having opened the "sample" folder you see two files. One is called sample.sh, the other sample.tex. We will use these two documents for further processing.

As a next step, we are going to look at those two files in detail.

5.2 Details about sample.sh and sample.tex

The sample.sh is a bash script. A bash script is just a plain text file, in which you can write any commands, that you normally run in the command line terminal. It is more cshowsonvenient to write the commands into a shell script so that we can use it repeatedly. Below is the descriptions of the basic workflow of the sample project. As we can see from the graph we first need to download the necessary data from the link:

```
http://spatialkeydocs.s3.amazonaws.com/FL_insurance_sample.csv.zip
```

This dataset contains 36,634 records of a sample company based in Florida from 2012 which implemented an aggressive growth plan in that same year. The data amount is adequate and it enables us to run some data analysis based on this dataset. After the dataset is downloaded successfully, we unzip it using the command line.

Next, we want to introduce you Weka[HFH+09]. It is a collection of machine learning algorithms used for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization. You need to make sure that your system has this software installed for both versions 3.8.3 and 3.9.1. If not, please recover the comment block from the sample.sh for step 3. But if you already have Weka installed, just change the path for the Weka calling command in step 4 and point it to the right directory, where you stored your Weka tool.

Step 4 intends to use ZeroR classifier from Weka 3.8.3 and Weka 3.9.1. This classifier is applied to predict the mean for a numeric class or the mode for a nominal class. In our case, we are going to have the mean as the processing result.

Next step of our program is writing the result of the Weka data processing into a .pdf file. For this purpose, we supply the sample.tex. In order to compile this file, you must have Latex in your system. Download Latex before you run the script.

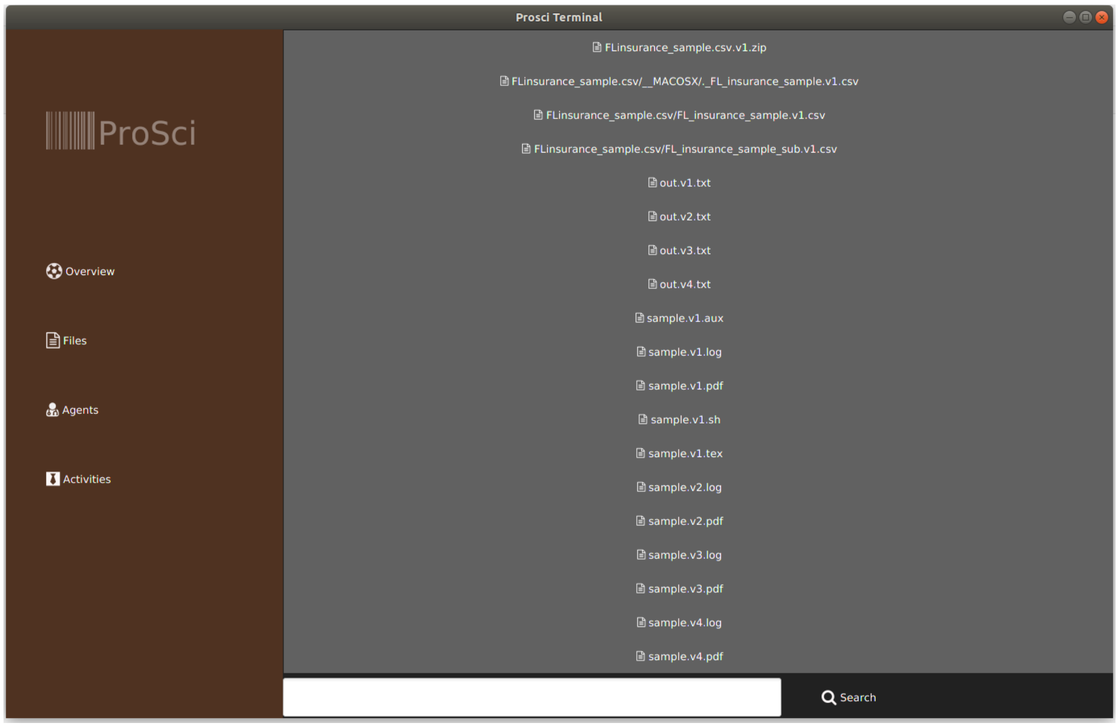
Now, we extract columns from one to six from the original dataset and use only the Weka 3.9.1 for processing and write the results into a PDF document.

5.3 Run sample

After the description above, you should be able to start ProSci and enjoy the result from our sample project. Now, when you go back to the Prosci main directory, the exact workflows are described as below:

- Start 'prosci.sh' with your terminal.
- Create a new workspace if you don't have any workspace currently in your application. Our suggestion is to create a workspace with the default name prosci, so the workspace creation command is the following: workspace prosci [path]
- Start you working section on your current workspace. with command: start
- Copy the sample.sh and sample.tex into the home directory of your [workspace path]/input folder.
- Run the sample.sh in the current working section terminal.
- As you complete running the sample.sh, you can find some new files in your workspace home folder
  - `FL_insurance_sample.csv.zip`
  - `FL_insurance_sample.csv`
  - `FL_insurance_sample_sub.csv`
  - sample.aux
  - sample.log
  - sample.pdf
  - sample.tex
  - sample.sh
  - out.txt
- Now let's use the advanced feature of the ProSci tool. Optionally, you can close the working section terminal (Xterm), but it is not necessary. Switch back to the Prosci main terminal, type the command 'graph' to initialize our graph visualizer. After some seconds, a new window is opened. From there you can get an overview of the ontology structure of the sample project.

- In total, you have four options in our graph visualizer. The overview presents you with an ontology graph. It illustrates the connectivity between every activity (command), entity (file) and agent (software agent). You can click on the graph to choose a specific element, the details of this element in the graph will be displayed as a table. As an alternative you can also search each property on the graph and on the dashboard to the left of the visualizer's window. They are sorted into three catalogs, respectively: activities, files and agents.

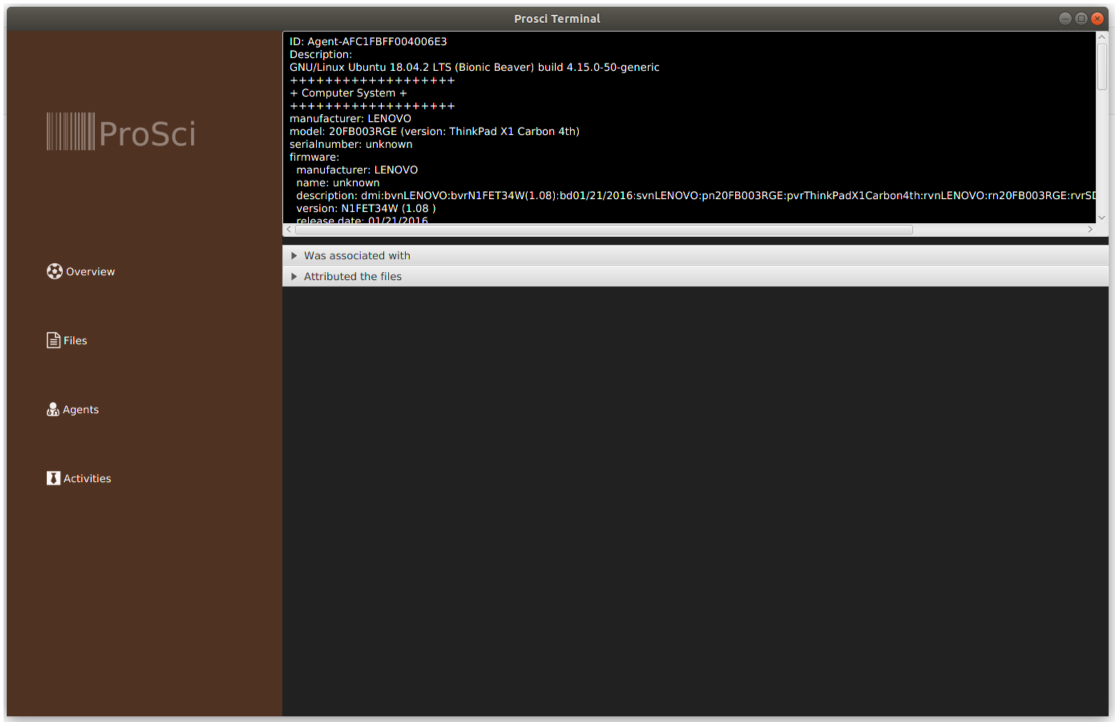


```
FLinsurance_sample.csv.v1.zip
FLinsurance_sample.csv/__MACOSX/._FL_insurance_sample.v1.csv
FLinsurance_sample.csv/FL_insurance_sample.v1.csv
FLinsurance_sample.csv/FL_insurance_sample_sub.v1.csv
out.v1.txt
out.v2.txt
out.v3.txt
out.v4.txt
sample.v1.aux
sample.v1.log
sample.v1.pdf
sample.v1.sh
sample.v1.tex
sample.v2.log
sample.v2.pdf
sample.v3.log
sample.v3.pdf
sample.v4.log
sample.v4.pdf
```

Figure 5.1: GUI view of Sample Project: click on files-button

After completing the steps. We have already generated a basic ontology structure of a simple scientific experiment, which includes downloading the dataset from external web service, processing the dataset to receive a short report about the dataset and generating a PDF document by using latex to report the experiment's result.

We verify the ability of the tool to detect the reason why the rerun of the computational tasks by using the same dataset didn't produce the same result for the PDF generation task. It is obvious, the change of the `FL_insurance_sample.csv` and the different versions of Weka cause the variation of the sample.pdf.



```
ID: Agent-AFC1FBFF004006E3
Description:
GNU/Linux Ubuntu 18.04.2 LTS (Bionic Beaver) build 4.15.0-50-generic
++++++++++++++++++
+ Computer System +
++++++++++++++++++
manufacturer: LENOVO
model: 20FB003RGE (version: ThinkPad X1 Carbon 4th)
serialnumber: unknown
firmware:
  manufacturer: LENOVO
  name: unknown
  description: dmi:bvnLENOVO:bvrN1FET34W(1.08):bd01/21/2016:svnLENOVO:pn20FB003RGE:pvrThinkPadX1Carbon4th:rvnLENOVO:rn20FB003RGE:rvrSD
  version: N1FET34W (1.08 )
  release date: 01/21/2016
```

Figure 5.2: GUI view of Sample Project: click on one of the agent

Secondly, we hope to abstract the ontology information efficiently from the ontology graph and also from the separate sub-category of the ontology properties. Fig. 5.3 shows us the search result of the ".pdf". We can see in this figure, the related file nodes are colored yellow. Fig. 5.4 shows the "Files" pane after applying the search criteria ".pdf". They are namely: `sample_v1.pdf`, `sample_v2.pdf`, `sample_v3.pdf` and `sample_v4.pdf`.

It is clear, after executing steps described above, that we still only have a small number of entities, activities, and agents.

The remaining question is to verify if the tool can deliver stable performance when processing a high volume task. To test this concentration we modify the sample.sh to run PDF generation for a hundred times and every time change one single character in "sample.tex". We successfully detected the change of the ontology graph in the Graph-Visualizer, see Fig. 5.5.

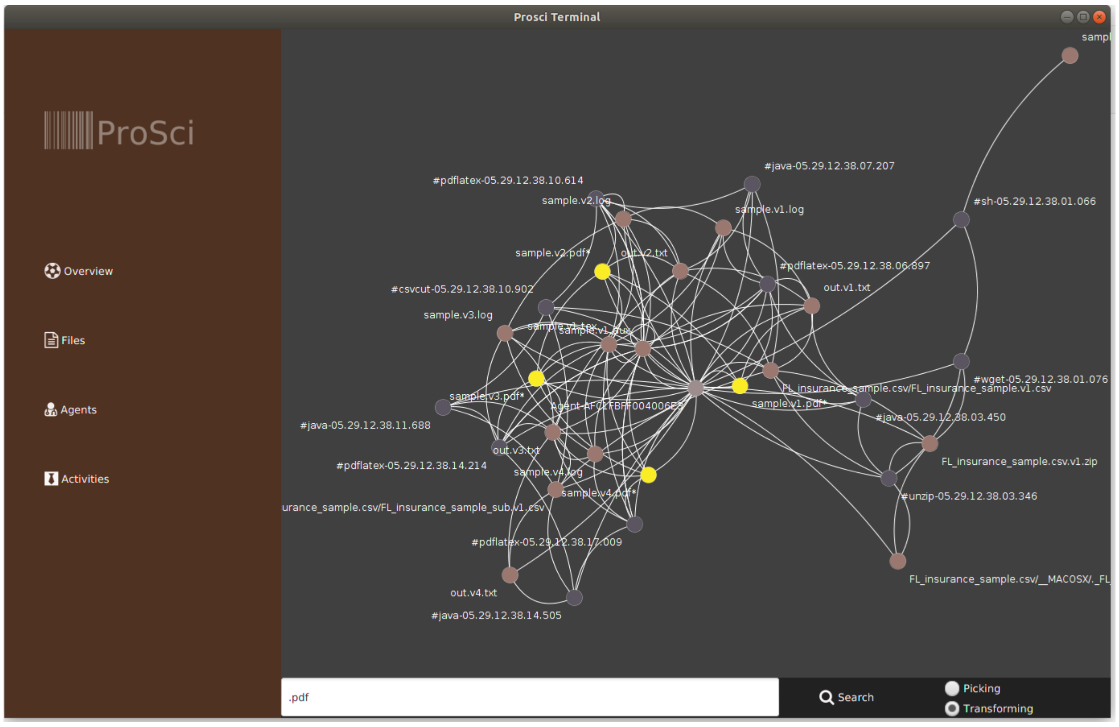


Figure 5.3: GUI view of Sample Project: search ".pdf" from the graph

5.4 Summary

During the testing phase, we noticed that the tool works slowly when it is used to process a big quantitation of the task.

However, it is undeniable, that the functionalities such as monitoring, tracking and capturing of the provenance data of a scientific experiment are working properly and that the transforming mechanism of the provenance data into an ontology graph and the user interaction ability are also available.

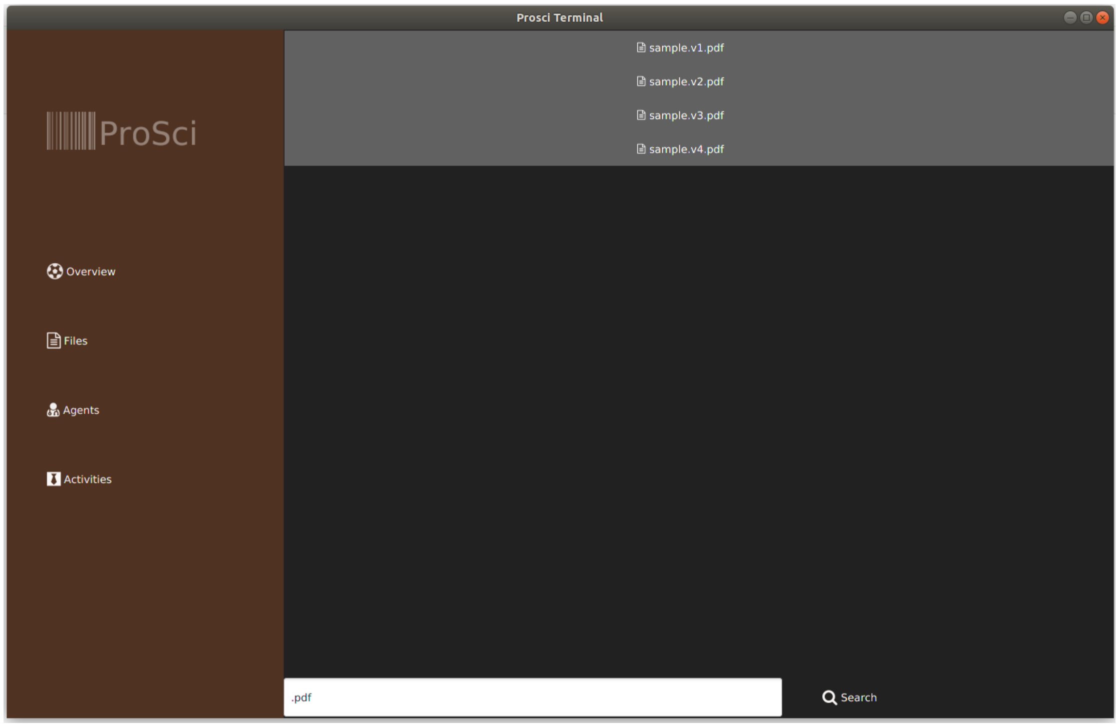


Figure 5.4: GUI view of Sample Project: search ".pdf" from the files pane

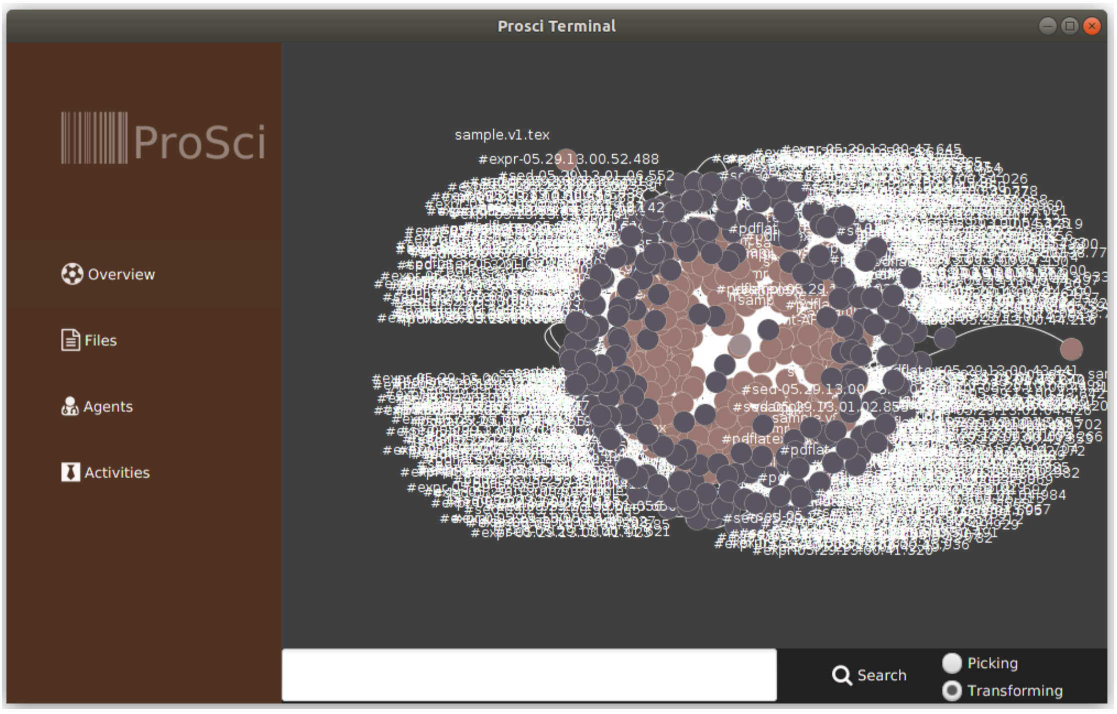


Figure 5.5: GUI view of Sample Project: PDF generation for a hundred times

CHAPTER 6

Conclusion

In our ProSci project, we have investigated a new tool, which integrates several APIs and tools. The whole project is built upon Java 8, using the Maven build tool, based on the provenance ontology theory. ProSci enables the tracking of provenance data during scientific workflow execution, visualizes the provenance metadata with a user-friendly GUI view and allows the user to reload the files, which are overwritten by the name convention.

It maximizes the value of provenance data and establishes the possibility of reproducibility and replication of the scientific experiments. As we know that, the provenance data of scientific workflows dedicates itself to verification and validation of the quality of the result of the scientific experiment.

6.1 Recall the Research Questions

Recalling the research questions proposed in the introduction section, we want to obtain provenance information without manual steps. That is to say, we want to make it fully automatic. The solution to this requirement is a combination of Strace, JGit, and Oshi. By using them we achieve automated tracking, monitoring and capturing of system provenance informationnot not only for input, output, and the intermediate workflow steps, but also the software and hardware environment and the library dependencies of the experiment.

By using ProvToolbox we are able to transform the provenance information into the W3C official standard with the programming language Java. For this purpose, we must take both log files which are collected from Strace and version control repository into account. Thus, the question about how to transform the provenance data into an ontology is solved.

A useful and meaningful interaction possibility of the provenance ontology appears to be the critical point of our thesis. Remember our goal is by using visualization of the provenance metadata to improve the reproducibility of the computational experiment. So we not only want to show the provenance ontology into a human-readable format but also want to make the representation efficiently and significantly discovery the reason if the re-run of the computational task failed. For achieving this goal, we apply JavaFX for UI implementation, Jung for graph establishing and Java DOM for searching through the provenance XML file. Therefore, we construct a meaningful Graph-Visualizer for representing the provenance information, which in return benifits the reproduction of the scientific experiment.

6.2 Limitations

Going through the whole project, we notice that we still have many features in ProSci require improvement. Some critical points are:

- Limitation on the supported system: the Project Prosci is only supported on the Linux system.
- Dependencies on external tools, such as Xterm and Strace limit the extensibility of Prosci.
- A better API for graphics visualization is desired. Currently, the API used for visualization is JUNG. Since JUNG aims only at the modeling, analysis, and visualization of data that can be represented as a graph or network, it doesn't support many advanced features for provenance ontology visualization. Some domain-specific API or plugin e.g. Ontograf provides better potentiality and illustration capabilities for the provenance ontology.
- "command.txt" serves as a dictionary for pairing files and commands. The retrieving method can be improved, so that seamless operation of the program is realized when the user attempts to work on a larger data volumes.
- The File Monitor component of the ProSci is running in the background all the time.

- No automated replication of the scientific workflow is embedded.

6.3 Future Work

Reflecting the limitations of the ProSci suggested above, it is necessary to concentrate on the improvement for both functionality and non-functionality requirements, in order to make the ProSci project more user-friendly and support it on different operating systems.

Furthermore, as a next step, we will investigate the verification of the scientific workflows and validation of the result of the scientific experiment [MRM16].

A consummate mechanism for the automation of the replication of the scientific experiment can be yielded based on a refined provenance data collecting mechanism.

List of Figures

List of Tables

Bibliography

[ABJ+04a] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424, June 2004.

[ABJ+04b] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: Towards a Grid-Enabled system for scientific workflows. *the Workflow in Grid Systems Workshop in GGF10-The Tenth Global Grid Forum, Berlin, Germany, March*, 2004.

[AJF+11] Jim Austin, Tom Jackson, Martyn Fletcher, Mark Jessop, Bojian Liang, Mike Weeks, Leslie Smith, Colin Ingram, and Paul Watson. Carmen: Code analysis, repository and modeling for e-neuroscience. proceedings of the international conference on computational science, iccs 2011. *Procedia Computer Science*, 4:768 – 777, 2011.

[BCMW11] Grant R. Brammer, Ralph W. Crosby, Suzanne J. Matthews, and Tiffani L. Williams. Paper mâché: Creating dynamic reproducible science. *Procedia Computer Science*, 4:658 – 667, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[BL13] Marc Bux and Ulf Leser. Parallelization in scientific workflow management systems. *CoRR*, abs/1303.7195, 2013.

[Bro14] Jason Brownlee. Reproducible machine learning results by default. *Machine Learning Mastery*, 2014. https://machinelearningmastery.com/reproducible-machine-learning-results-by-default/.

[Cha14] Scott Chacon. *Pro Git.* Apress, 2014.

[CSF13] Fernando Chirigati, Dennis Shasha, and Juliana Freire. Reprozip: Using provenance to support computational reproducibility. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, TaPP '13, pages 1:1–1:4, Berkeley, CA, USA, 2013. USENIX Association.

[DHG$^+$14] Carl Dea, Mark Heckler, Gerrit Grunwald, Jos Pereda, and Sean Phillips. *JavaFX 8: Introduction by Example.* Apress, Berkely, CA, USA, 2nd edition, 2014.

[DPA$^+$18] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. *Int. J. High Perform. Comput. Appl.*, 32(1):159–175, January 2018.

[dPHG$^+$13] Renato de Paula, Maristela Holanda, Luciana S. A. Gomes, Sérgio Lifschitz, and Maria Emilia Telles Walter. Provenance in bioinformatics workflows. In *BMC Bioinformatics*, volume 14 Suppl 11, 2013.

[DSS$^+$14] Moritz Post Dominik Stadler, Remy Chi Jian Suen, et al. Jgit/user guide. 11 2014.

[GD07] Boris Glavic and Klaus R. Dittrich. Data provenance: A categorization of existing approaches. In Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, and Christoph Brochhaus, editors, *BTW*, volume 103 of *LNI*, pages 227–241. GI, 2007.

[GE11] Philip J. Guo and Dawson Engler. Using automatic persistent memoization to facilitate data analysis scripting. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 287–297, New York, NY, USA, 2011. ACM.

[GFI16] Steven N. Goodman, Daniele Fanelli, and JohnP. A. Ioannidis. What does research reproducibility mean? *Science Translational Medicine*, 8:341ps12–341ps12, 2016.

[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):1–13, 06 2012.

[GM11] Pieter Van Gorp and Steffen Mazanek. Share: a web portal for creating and sharing executable research papers. *Procedia Computer Science*, 4:589 – 597, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. *What Is an Ontology?*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[HKP+09] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, World Wide Web Consortium, October 2009.

[HZ10] Olaf Hartig and Jun Zhao. Publishing and consuming provenance metadata on the web of linked data. In Deborah L. McGuinness, James R. Michaelis, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, pages 78–90, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KSM+11] David Koop, Emanuele Santos, Phillip Mates, Huy T. Vo, Philippe Bonnet, Bela Bauer, Brigitte Surer, Matthias Troyer, Dean N. Williams, Joel E. Tohline, Juliana Freire, and Cláudio T. Silva. A provenance-based infrastructure to support the life cycle of executable papers. *Procedia Computer Science*, 4:648 – 657, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[LPVM15] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *J Grid Computing*, 13(4):457–493, December 2015.

[LSM+13] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. *PROV-O: The PROV Ontology.* W3C Recommendation. World Wide Web Consortium, United States, 4 2013.

[LWMB09] Bertram Ludäscher, Mathias Weske, Timothy Mcphillips, and Shawn Bowers. Scientific workflows: Business as usual? In *Proceedings of the 7th International Conference on Business Process Management - Volume 5701*, BPM 2009, pages 31–47, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[MCF+11] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743 – 756, 2011.

[Mes10] Jill P. Mesirov. Accessible reproducible research. *Science*, 327(5964):415–416, 2010.

[MM12] L Moreau and Paolo Missier. Prov-dm: The prov data model. 01 2012.

[Mor] Luc Moreau. Jung. Technical report. https://jrtom.github.io/jung/.

[Mor15] Luc Moreau. Provtoolbox. Technical report, 2015. http://lucmoreau.github.io/ProvToolbox/.

[MRM16] Tomasz Miksa, Andreas Rauber, and Eleni Mina. Identifying impact of software dependencies on replicability of biomedical workflows. *Journal of Biomedical Informatics*, 64:232 – 254, 2016.

[MSSW13] Gina Moraila, A Shankaran, Zuoming Shi, and AM Warren. Measuring reproducibility in computer systems research. *http://reproducibility.cs.arizona.edu/*, pages 1–37, 01 2013.

[ODS$^+$13] Eduardo Ogasawara, Jonas Dias, Vítor Sousa, Fernando Chirigati, Daniel de Oliveira, Fabio Porto, Patrick Valduriez, and Marta Mattoso. Chiron: A parallel engine for algebraic scientific workflows. *Concurrency and Computation*, 25:2327–2341, 11 2013.

[osh] Oshi. Technical report. https://github.com/oshi/oshi.

[PE09] R. D. Peng and S. P. Eckel. Distributed reproducible research using cached computations. *Computing in Science Engineering*, 11(1):28–34, Jan 2009.

[Pen11] Roger D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[PMBF17] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. noworkflow: A tool for collecting, analyzing, and managing provenance from python scripts. *Proc. VLDB Endow.*, 10(12):1841–1844, August 2017.

[rcs15] Gnu rcs, 2015. https://directory.fsf.org/wiki/Rcs.

[Ros95] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[Rot18] Richard Roth. Reproduzierbarkeit via ontologischer darstellung der provenance. page 81, 1 2018.

[SNB$^+$11] Satya S Sahoo, Vinh Nguyen, Olivier Bodenreider, Priti Parikh, Todd Minning, and Amit P Sheth. A unified framework for managing provenance information in translational research. *BMC bioinformatics*, 12(1):461, 2011.

[SNTH13] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLOS Computational Biology*, 9(10):1–4, 10 2013.

[STVK$^+$08] Carlos Scheidegger, Huy T. Vo, David Koop, Juliana Freire, and Claudio Silva. Querying and re-using workflows with vstrails. pages 1251–1254, 01 2008.

[VKV09] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3):37–47, May 2009.

[WHF$^+$13] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 05 2013.

[Wik19] Wikipedia contributors. strace, 2019.

[WMF$^+$04] Anil Wipat, Darren Marvin, Justin Ferris, Kevin Glover, Mark Greenwood, Martin Senger, Matthew Addis, Matthew R. Pocock, Peter Li, Tim Carver, and Tom Oinn. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 06 2004.