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 der externen Ressource. 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 der hierarchischen Ansicht der historischen Zwischendateien. Darüber hinaus bietet es Zugriff zu den 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

The scientific workflows are a cluster of the computational tasks, which are connected to each other and enable the different kinds of the function, such like retrieving, processing, analyzing and abstracting from data. Those computational tasks always involve the reading data from external resource and computing the processed data or finding evidence during the tasks processing as final result. The scientific work flows 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 the collecting data from all fields and branches plays increasingly important role. especially for the data analytical science. Therefore, the appearance of the scientific work flow management tool, such like Taverna, Kepler and VisTrails, is admired. However, the tasks in the complete work flows sometimes need to be run several times due to various reasons.

Without doubt, the saving and controlling of the intermediation between each workflow's run are valuable and important. Currently, if the naming conventions are not applied, the file will be overwritten and get lost. The preparation of a systematically maintained archiving of the produced intermediate files for scientific work flows is critical and essential from the point of view of the scientific computational experiment.

The aim of the work is to provide a stable and a functionality enabled tool for scientific workflow provenance documentation. The tool visualizes the scientific work flows with the hierarchical view of the historical intermediate files. The implementation of the tool is based on the PROV Ontology definition, which was published by W3C as standard. The tool initially contains tree components, the application console, the file monitor and the graph visualizer. The application console should take the response for recording the 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 most essential part of the tool is the work flow ontology visualizer. This component integrates the visualization of the work flow ontology, monitoring and controlling the computational tasks and its provenance ontology creator is used mainly for translating and converting the scientific work flows and their intermediate files into the Extensible Markup Language (XML) format and generate the visualized graph accordingly as the workflow ontology.

Moreover it provides the access to the historical generated provenance data. The provenance process of the tool should be realized and derived based on the version control theory. In order to comprehensive and sophisticated apply to the version control concept, the Jgit framework is considered. Secondly, it is obviously that in the field of the computational experiment the environmental setting for executing the work can lead to very different results. 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 the least, the tracing of the work tasks. As the tool intends to deploy in the Unix system , the tracing tool targets to the Strace.

CHAPTER 2

Theoretical Basis

In this chapter, we are going to suggest some of the essential technologies which 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. 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 in reality be exactly repeated. "Result reproducibility" points to the receiving of 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 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 and 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 the research in computational science so important? "Reproducibility has the potential to serve as a minimum standard for judging scientific claims when full independent replication of a study is not possible." [Pen11] , 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.

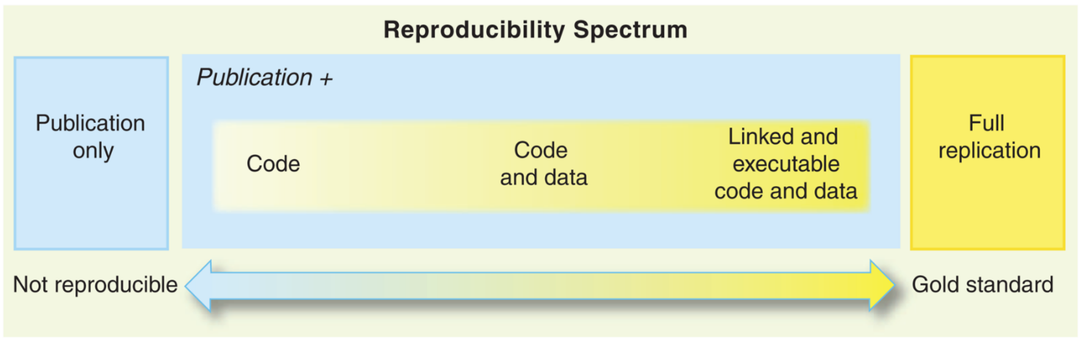


Figure 2.1: The spectrum of reproducibility [Pen11]

It is clear, that we aim to make the replication of the research as far as possible to settle in the right hand of the Fig. 2.1 as possible. 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 Datail to Be In spected
- 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 filed of the 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 so easy as we think about. The problems involved not only the 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 completion of the codes often occur to be hurdles.

The facts is 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 proposed 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 at. [VKV09] also wrote in their paper and distinguish the 6 different levels of reproducibility. The level starts with 5 "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 ends up with 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$^+$11]. Besides, Koop et al. [KSM$^+$11] introduce a provenance-based system that captures workflows and Mache[BCMW11] supports the executable paper concept.

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, analysis and their results, also, it packages them for the future distribution. RRP is a document preparation system, and it seems like a standard word-processing software.

Without a doubt, reproducible research plays a more and more important role in all field of the scientific experiment. 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 about the topic "Distributed reproducible research using cached computations." In this article, they present a method, by using cached computations, the executions of the statistical analysis are stored 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 researcher who makes efforts to improve the standard. However, most of us don't know the measurement 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 concern 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 Machteld Marcelis et al. []. In their experiment, they investigated the measurement of cortical thickness and volume with diverse variables conditions, such as different FreeSurfer version (v4.3.1, v4.5.0), different workstation (Macintosh and Hewlett-Packard), and Macintosh operating system version (OSX 10.5 and OSX 10.6). In Fig. 2.2. we show the workstations used in the study.

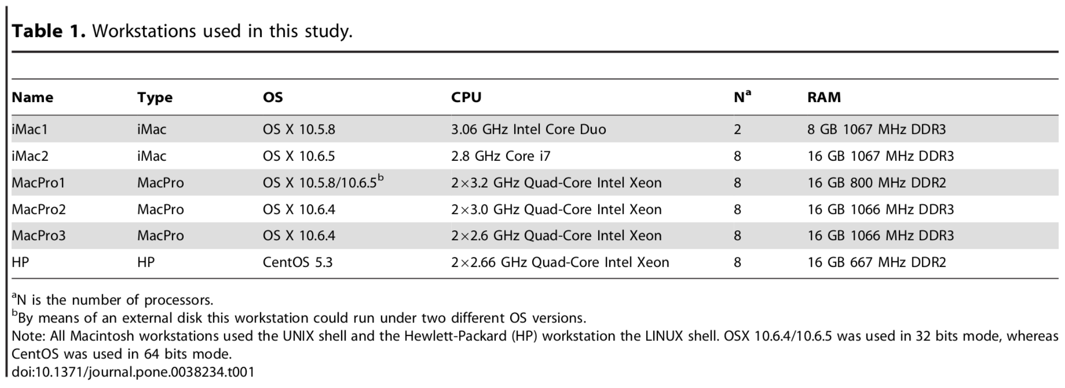


Table 1. Workstations used in this study.

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]

Surprisedly, significant differences were detected between the FreeSurfer version. The results are listed in details in Fig. 2.3.

According to the study, the authors warn the user of the FreeSurfer to be careful before applying the upgrade in the FreeSurfer version, OS version or switch 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. We had better keep those variations the same in an ongoing project.

2.3 Scientific Workflow and Workflow Management System

2.3.1 Scientific Workflow

In this section, we are going to dig out the conceptual of the scientific workflows. What exactly SWFs mean? Some authors define the scientific workflow in the following sentences.

"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]

At the meantime, in the article "the future of the scientific workflows" the authors say, “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 additional define the 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

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

The workflow series mainly consists of five elements. Task specification, with or without dependencies. Sometimes, the output of one task may be the input of another task. The second element is the 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 input files and the output files. The so-called file management deals with the issues, such as making the input files available for execution and archive the output files.

2.3.2 Scientific Workflow Management System

After the clarification of the fundamental elements of the workflow, we realize that the tool that makes the 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 the Fig. 2.4, the scientific workflow management system is presented as layers. Basically, it is divided into five layers. The presentation layer, which in other word is the user interface, it facilitates the interaction between the user and the SWfMS. The user services layer provide 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(WEP) execution layer. The necessary physical resource is 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 and scheduling techniques. Under the term of the parallel execution, the parallelism can be divided further into three types:

- data parallelism
- independent parallelism
- pipeline parallelism

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

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

- static scheduling
- dynamic scheduling
- hybrid

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

Teverna

As we talking about the SWfMSs, we can not ignore the Teverna [WHF$^+$13] , the most propagated SWfMSs. Teverna concentrates on the workflow executing and designing based on the 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 such as for reduction of the local infrastructure and maintenance costs and development and testing the workflow repidly. 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 the scientists with an easy-to-use software tool for conducting analyses and run models in various software and hardware environment. 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]

2.4 Provenance and Ontology

2.4.1 Provenance

Across all fields of the science domain, the scientific workflows which perform the valuable adjustment, extraction, and processing on enormous data volume. Taking inputs and derive meaning outputs, discover the meaningful information behind the huge amount of the 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 quantitative and qualitative of the data and the scientific experiment, 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 beginning from the point of their creation to intermediate processing, and end up with their end use of the final results. [SNB+11] Sahoo et al. illustrate the provenance life cycle with four phases in Fig. 2.6.

They distinct the provenance life cycle between pre- and post-publication. The pre-publication is the state before the publication of the data and the results for the public access and to the data repository, post-publication describe 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 French. It means "to come from". To interpret the concept of the provenance we follow the instruction of the Boris Clavic et al. [GD07]. They distinguish the 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 the 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]

For tracking the provenance metadata we introduce the PROV-O[LSM+13] 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.

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 start points are derived. Fig. 2.11 illustrates some basic properties of the PROV-O. At the meantime, The 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 the specific previous version of the file system. It records the changes to a series of the file 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 files.

Over the decades, the development of the version control system had gone through several revolutions. Scott Chacon[Cha14] differentiates the version control system into three types. They 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 keeps all the changes to files, see Fig. 2.12. The famous tool which uses this technology is called rcs.

2.5.2 Centralized Version Control Systems

When considering the integration between 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. 2.13 . 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, The practice prone that single point of failure to be its downside. The entire history of the works 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 system 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. 2.14. You can collaborate your work with different people in different environments simultaneously and don't need to worry about the data loss.

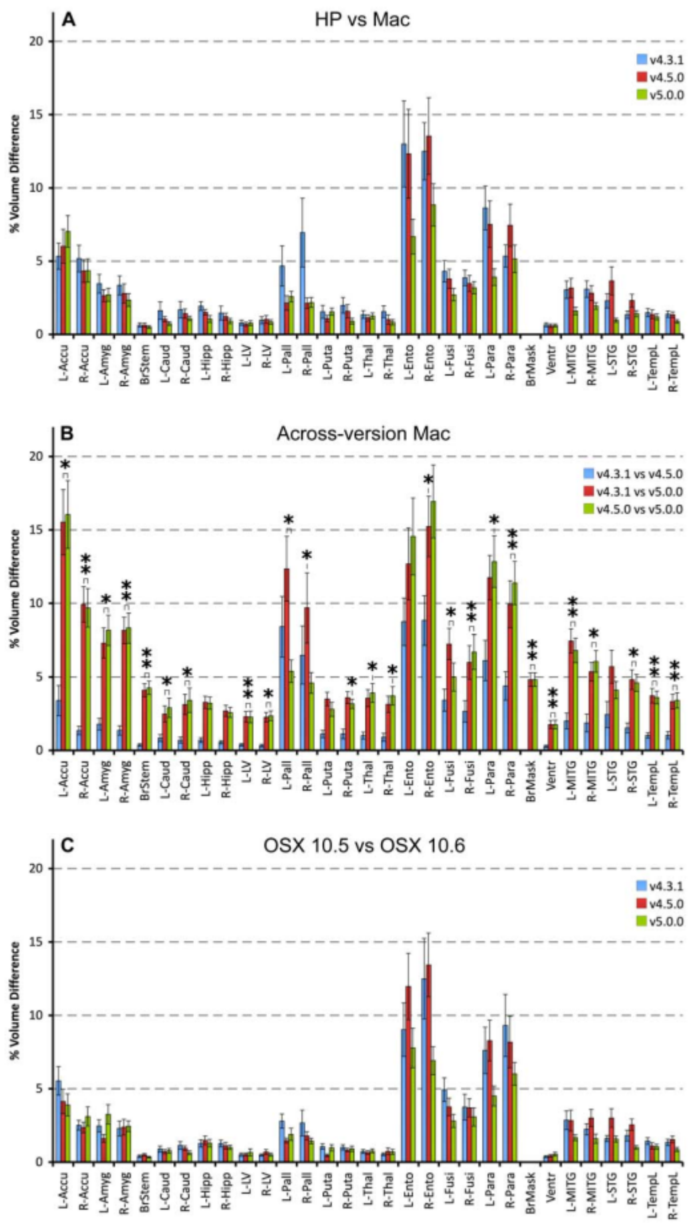


Figure 7. Effects of data processing conditions on the voxel volumes for a subsample of (sub)cortical structures. Panel A shows the detected percentage absolute differences between the results derived on a Mac and HP workstation for three different versions of FreeSurfer. Panel B depicts the differences 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); ** : $p\leq 0.0001$. 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]

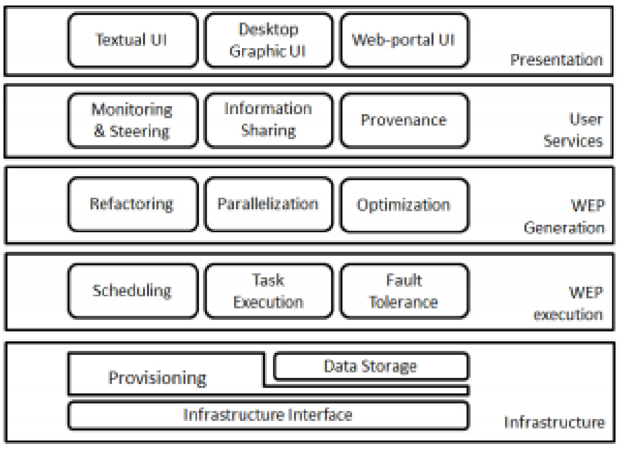


Fig. 3: **Functional architecture of a SWfMS.**

Figure 2.4: Functional architecture of a SWfMS [LPVM15]

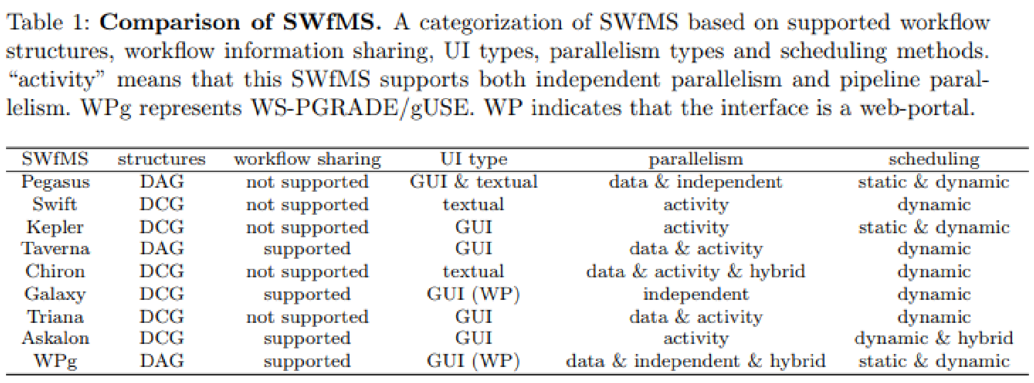


Table 1: **Comparison of SWfMS.** A categorization of SWfMS based on supported workflow structures, workflow information sharing, UI types, parallelism types and scheduling methods. “activity” means that this SWfMS supports both independent parallelism and pipeline parallelism. WPg represents WS-PGRADE/gUSE.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

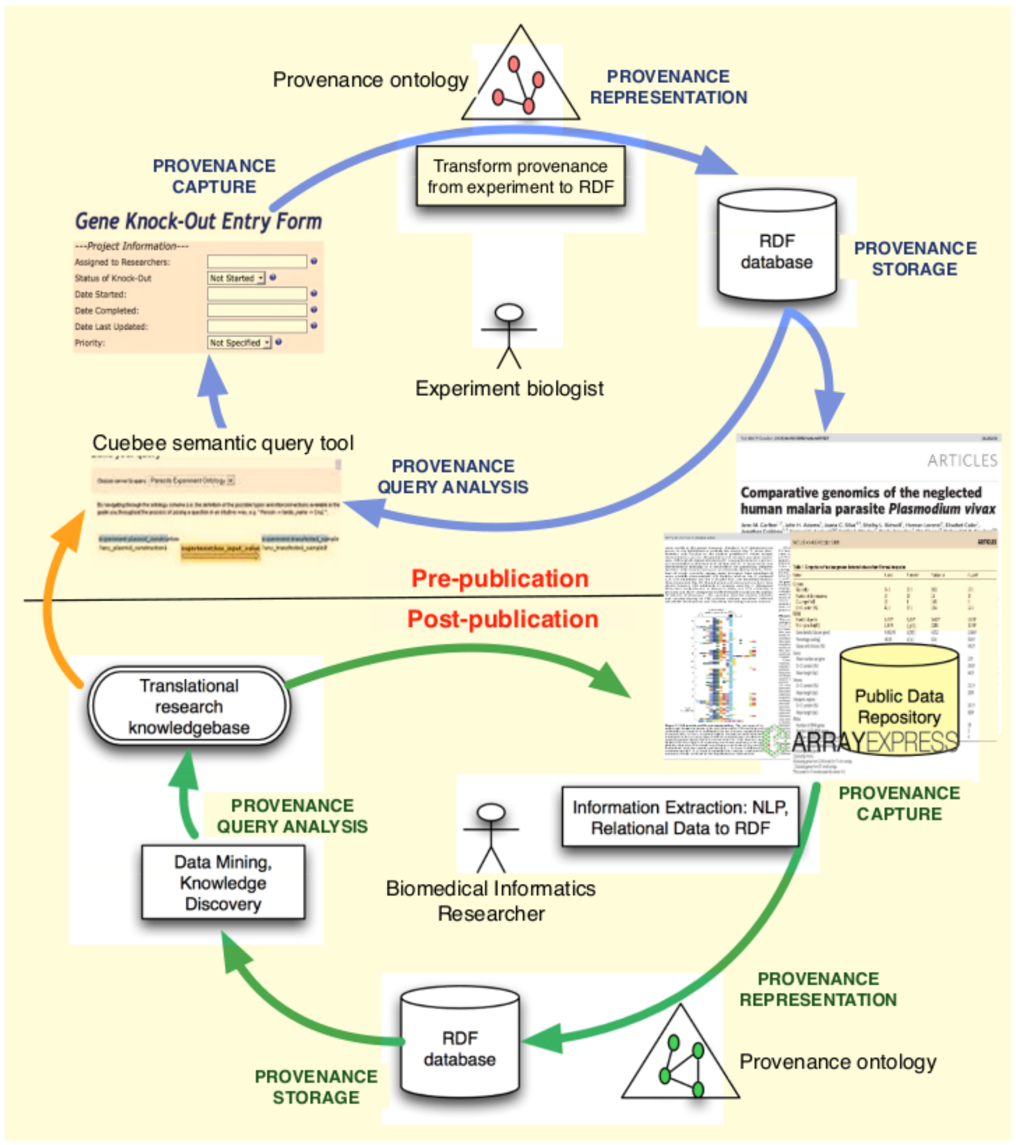

Provenance ontology
PROVENANCE REPRESENTATION
PROVENANCE CAPTURE
Transform provenance from experiment to RDF
RDF database
PROVENANCE STORAGE
Gene Knock-Out Entry Form
---Project Information---
Assigned to Researchers:
Status of Knock-Out
Not Started
Date Started:
Date Completed:
Date Last Updated:
Priority:
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]

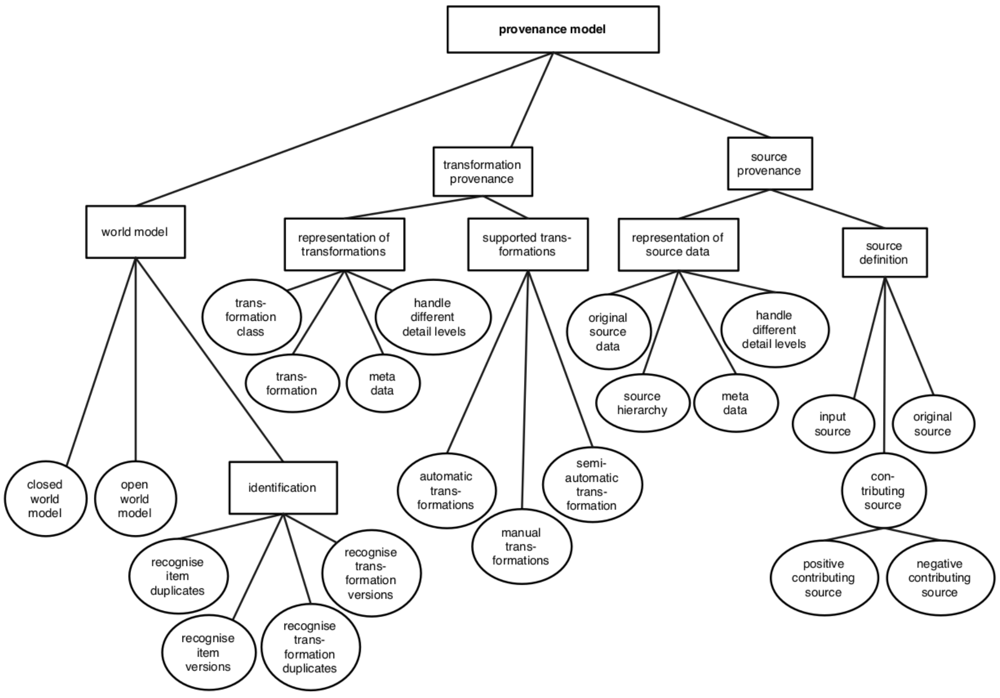

provenance model
world model
transformation provenance
source provenance
closed world model
open world model
identification
recognise item duplicates
recognise item versions
recognise trans-formation duplicates
recognise trans-formation versions
representation of transformations
trans-formation class
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]

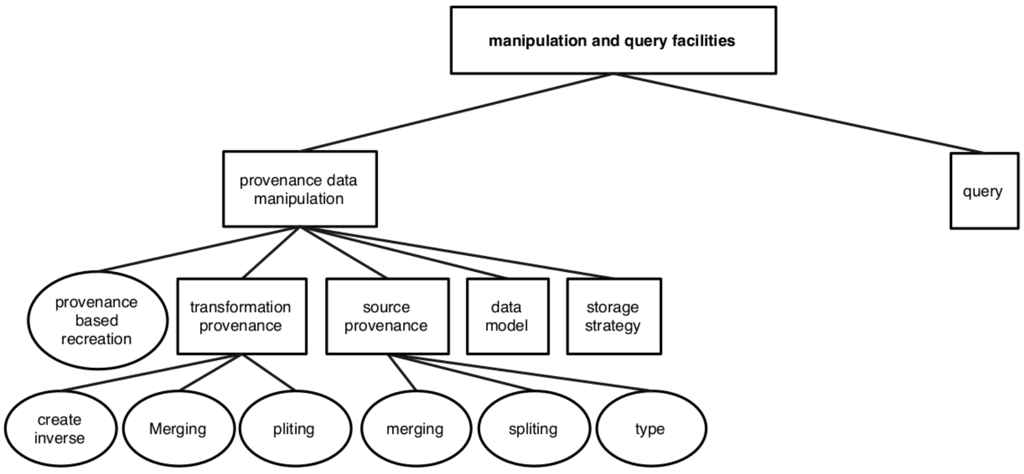

manipulation and query facilities
provenance data manipulation
query
provenance based recreation
transformation provenance
source provenance
data model
storage strategy
create inverse
Merging
pliting
merging
spliting
type

Figure 2: Query and manipulation funcionalities

Figure 2.8: Query and manipulation funcionalities [GD07]

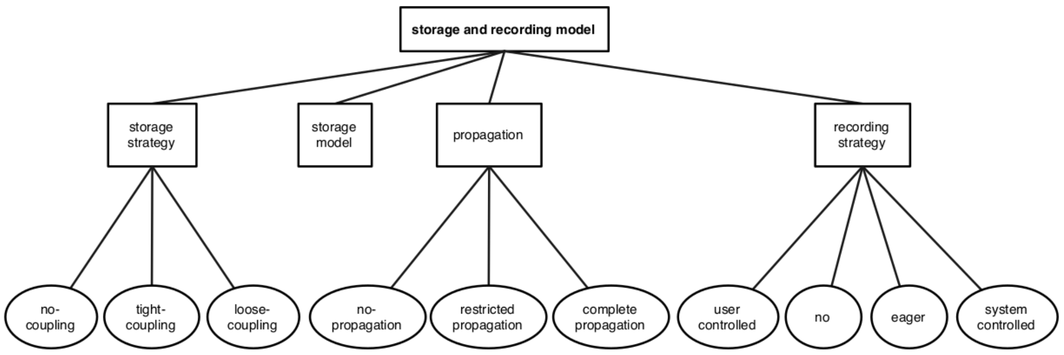


Figure 3: Storage and recording

Figure 2.9: Storage and recording [GD07]

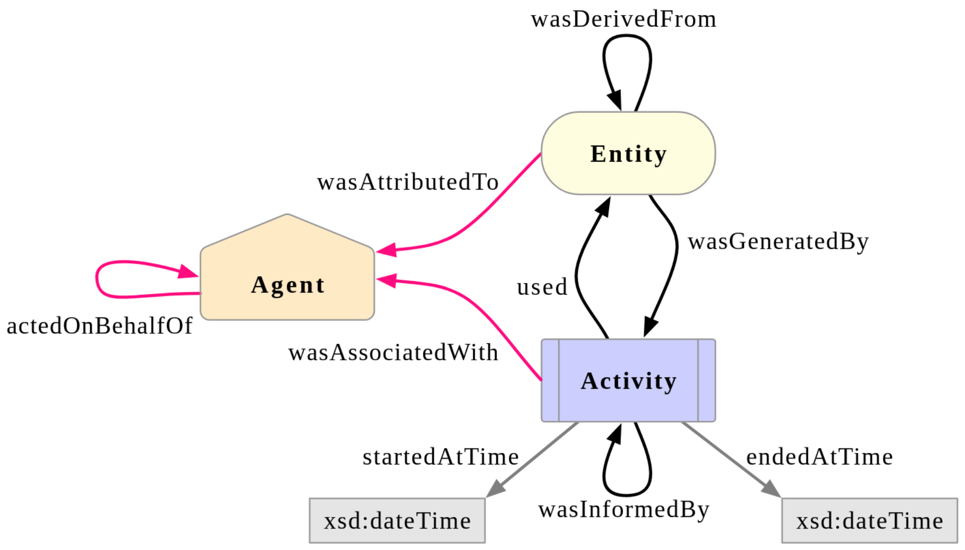


Figure 1. The three Starting Point classes and the properties that relate them.
The diagrams in this document depict Entities as yellow ovals,
Activities as blue rectangles, and Agents as orange pentagons.
The responsibility properties are shown in pink.

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

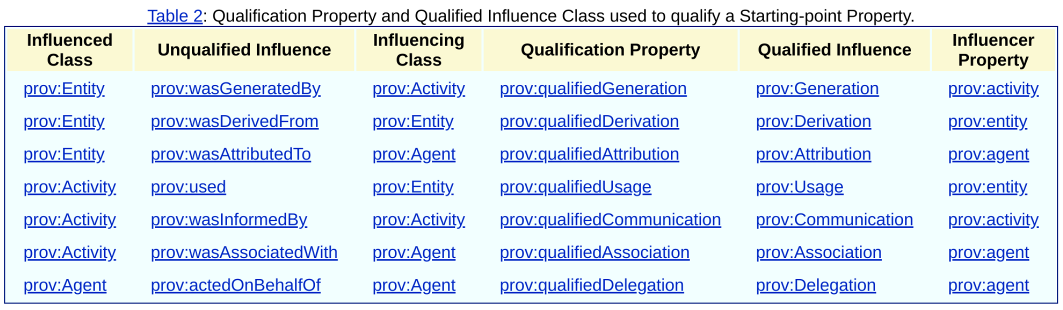


Table 2: Qualification Property and Qualified Influence Class used to qualify a Starting-point Property.

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]

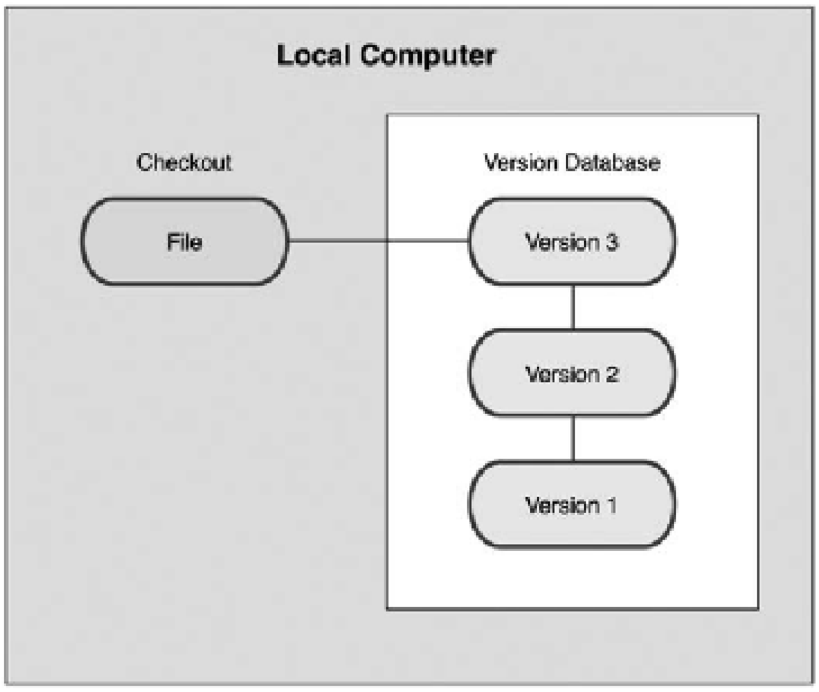


Figure 1-1. *Local version control diagram*

Figure 2.12: Local version control diagram [Cha14]

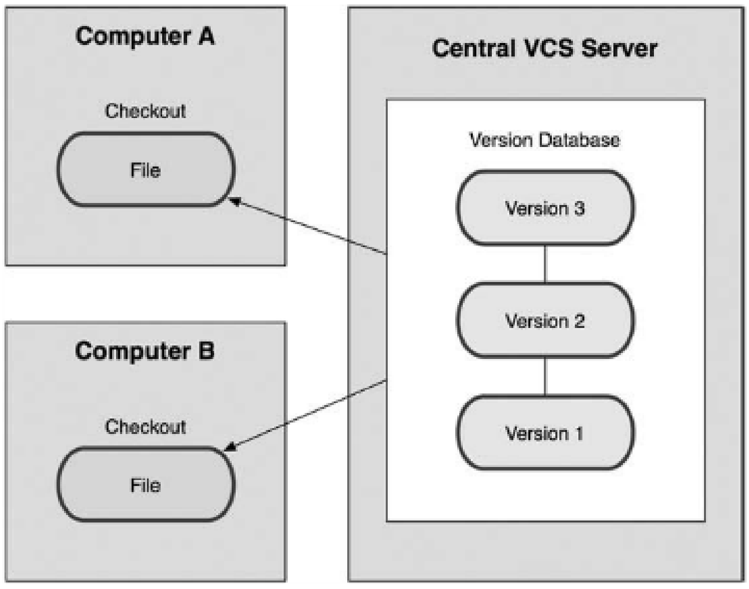


Figure 1-2. *Centralized version control diagram*

Figure 2.13: Centralized version control diagram [Cha14]

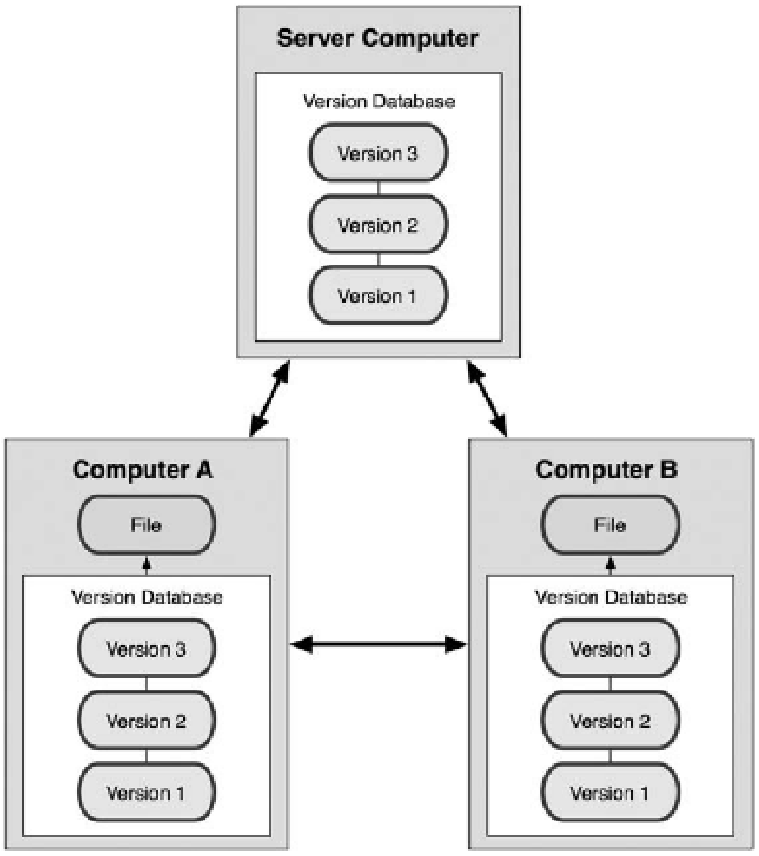


Figure 1-3. *Distributed version control diagram*

Figure 2.14: Distributed version control diagram [Cha14]

CHAPTER 3

Library Usages

After reviewing, collecting and reflecting all 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 the ProSci is benefited from the utilization of them.

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 has the demand on the version control integration.

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

Furthermore, we have "Ref", it is an object identifier and can references to any kind of the 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.

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

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

- 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 Fig3.1)

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 the 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 the strace.

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

The strace outputs contain in each line usually a system call which name, arguments, and the return value.

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

```
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

3.5 Oshi

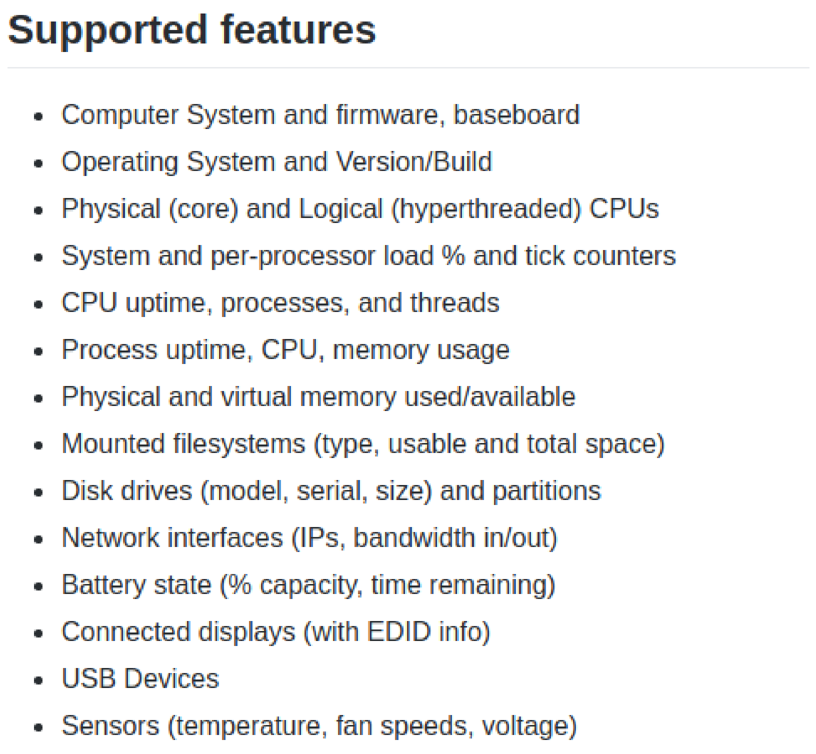
Oshi[osh] is a free Java API for collecting, retrieving and gathering the information about the operating system and hardware. It is independently, the installation requires no additional libraries. Oshi is supported almost in all common operation system, such as Windows, Linux, Mac OS X and Unix (Solaris,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

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

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

In the Fig. 3.2 we stick a screenshot, which we grabbed out from the homepage of the Oshi. In this Fig. the supported features of the Oshi are described.

However, not all feature are implemented across all operating system type. 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.



Supported features

- 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)

Figure 3.2: Supported features of Oshi

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 last 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.

3.6 Xterm

Xterm[Wik18] is a terminal emulator program for the X Window System, Xterm provides compatible terminals for the program which can't directly use the window system.

The current maintainer starts working on xterm since 1996. The xterm is much older. Xterm can runs on a wide range of the platforms, such as Linux, SunOS, Solaris and etc.

The customize xterm is also available by configuring the following assets:

- change the font size
- print the screen
- set up function keys
- set the title
- make the cursor blink

The details about how to make those configurations can be found here:

`https://invisible-island.net/xterm/xterm.faq.html`

3.7 Jung

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

JUNG is 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."[Mor03]

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

JUNG implements a number of algorithms of the 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 release of JUNG version 2.1.1 is on 7 September 2016. The repository is administrated by one of the co-creator of the JUNG project Joshua O'Madadhain. Find the source code from the URL:

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

Visit the homepage of JUNG:

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

The Java doc specification is there:

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

Moreover, the JUNG can also integranotenotete with your project by defining the following maven dependency.

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

CHAPTER 4

Project Structure

4.1 Use Case

Within this section, I am going to establish the primary use cases of the Prosci tool. As shown in Figure 4.1. The user of the Prosci, has six main functions to choose. The description of each use case is shown in the table under certain subsection. The workspace is playing an essential role of the Prosci tool, every time one the user starts the application, he needs to define a workspace, the workspace which is defined need 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 functionalities of the 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 about, is the Graphic-Visualizer. It provides the user an Overview of the entire workflows with the ontology definitions. Additionally, our Graphic-Visualizer supplies some specific features. Through it, the user is 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.

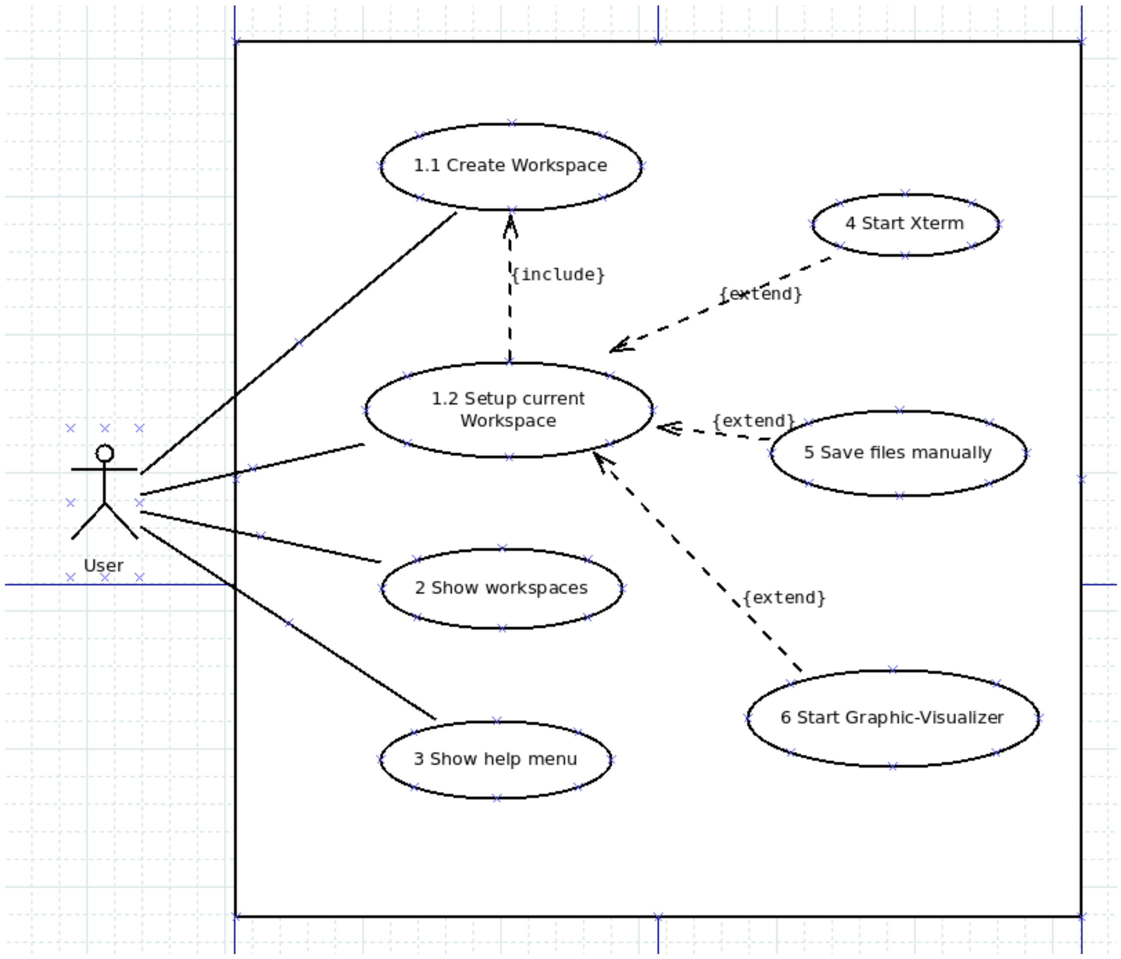


Figure 4.1: Main user cases of the Prosci

4.1.1 Create Workspace

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

As a consequence, every time once the user wants to start the application, they need to explain 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, not exit or path is given wrongly and if the workspace name is already being used. The console of the Prosci, will show up the error message with the correct command format to instruct the user for the next step.

Once the application be started, the user need 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 unter 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 exit in the system are stored under the prosci.properties in the home directory. The user is able to switch the workspace with the following command:

```
workspace [workspace name]
```

As every workspace is working independently. They have their own main directory.The user can create thier workspace according to thier demand.

A workspace is a fundamental milestone. If a workspace is created, this freshly created workspace will be defined automatically as the current workspace. Only if the user wants to change workspace, they can run 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 enter the workspace creation command
Frequency of Use:	Every time when the user wants to create a new workspace and 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 Application Console, enter the workspace creation command with correct parameters and press enter key.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for the further command
2	The user enter the workspace creation command with correct parameters: "workspace [workspace name] [workspace path]" and press enter.	The workspace is created with given name and path, and the related information is saved under 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. An error message will be shown and a message which contains the correct command will also be print into the console.		
Post Conditions		
1. A workspace is created under the given path with given name. 2. In the Prosci system file, the new workspace information is recorded. If the user creates the 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 don't trouble each other. So that the user can freely change the project and without losing the important data.

The command for this function has the same prefix "workspace" as for the "create workspace" use case. The difference between those two use case 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, the further execution steps can be done. The Table 4.2 shows us with 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 working on it.	
User/Actor	The user of the Procsci	
Trigger:	The user enter the setup current workspace command	
Frequency of Use:	Every time when the user start the application, a current workspace need to be defined.	
Preconditions		
1. The Prosci software package is successfully installed and the Application Console is started. 2. The workspace name which given by the user must have existed in the system.		
Basic Flow: The user starts Application Console, enter workspace setup command with specific workspace name, which already exits in our system.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for the further command

2	The user enter the workspace setup command: "workspace [workspace name]" and press enter.	The system switchs into 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 not entered correctly or the given workspace name doesn't exit. The error message is printed to advise the user for the further reaction.		
Post Conditions		
1. Workspace is defined as given one, further executions under this workspace will be recorded. 2. system property – prosci.properties change 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

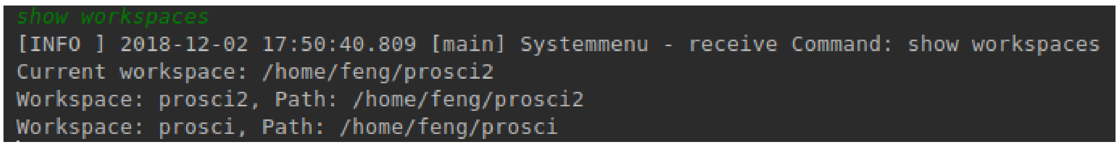
4.1.3 Show Workspaces

Equally important is the ability to show all existing workspaces in the Prosci system. Next use case is one, which makes it possible to present our Prosci user with an overview of all workspaces in our system. With the help of this function, the user 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 during the last run, if the user has already set up the working workspace, this information is also going to be printed. A possible output of this use case is illustrated in the Figure 4.2.

As we can see from the figure, the existing workspaces are presented with their name and the 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 current workspace is already defined, it will also be printed.	
User/Actor	User of the Prosci application, who want to know the existing workspaces in the system	
Trigger:	The user enter the show workspaces command	
Frequency of Use:	Normally, it happens at every first time, when the user start the application.	
Preconditions		
The Prosci software package is successfully installed and the Application Console is started		
Basic Flow: The user starts the application console, enter the show workspaces command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the command
2	The user enter the show workspaces command: show workspaces and press enter key.	The console prints all existing workspaces in the system, if there also 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 into the console. 2. If a current workspace is defined, it is shown with 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 defined in the system.	
User/Actor	The user of the Prosci application, who want to know the correct system commands.	
Trigger:	The user enter the help command	
Frequency of Use:	Every time when the user needs the help for 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 command
2	The user enter the show workspaces command: help and press 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 are shown into the console and their correct formats.
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	The user of the Prosci application, who want to trace his scientific workflow.	
Trigger:	The user enter the start xterm command	
Frequency of Use:	Every time, when the user wants to do some scientific experiment and hope 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 in the 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 command

2	The user either create a new workspace or define a 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 fresh created one. 2. if the user set the workspace as one, which already exits, the system property file – prosci.properties will change the current workspace into the defined one.
3	The user enter the start xterm command: start and press 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 the scientific experiment, the 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:	Under the given current workspace, the user is able to add files manually, and the files, which add by the user manually, will also be traced for the reproducing purpose.
User/Actor	The user of the Prosci application, who want to trace his scientific workflow.
Trigger:	The user enter the start xterm command
Frequency of Use:	Every time, when the user wants to do some scientific experiment and hope 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 current workspace and run the save file command.		
Step	User Actions	System Actions
1	Start Application Console	The Prosci system is started and waiting for further the command
2	The user adds the files manually into the current workspace.	————————————
3	The user enter save file command: save and press enter key.	The files which are added before manually by the user are saved into 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 current workspace are saved into 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 historical workflow backlog or they can reproduce the file that is overwritten during the experiment. Basically, they can view the graphic picture, on which the ontology description of the current workspace is illustrated and their detailed information are listed as tabs.	
User/Actor	The user of the Prosci application, who want to see the overview and details of the scientific workflows.	
Trigger:	The user enter the start graphic-visualizer command	
Frequency of Use:	Every time, when 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, start 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 command
2	The user enter start graphic-visualizer command: graphic visualizer and press enter key.	A new windows is open, which includes a overview picture of the all workflow in the system. All details are shown as tabs. System is ready for the user command 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. On which the user is able to view all historical commands and their input and output details. In the meantime, 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

4.2 Description Of The Components

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

As shown in the figure 4.3 below, we still have five external software agents, they are Xterm, JGit, Strace, Oshi, and ProvToolbox. All the foreign software agents are marked with the pink color in the Figure 4.3.

Xterm is a standard terminal emulator, it establishes a terminal environment, from which the user can execute the commands for the scientific experiment, and it seems like the user works on the real terminal environment. At the same time, the Strace will also be started to log the system behaviors. 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. The system behaviors which are recorded by the Strace into the log files are stored under the workspace directory [workspace name]/prosci/trace/log. All log files will be engaged for the further processing of the ontology structure. The Graph Visualizer will call the generated ontology structure. Through it, the Graph Visualizer is, therefore, able to illustrate us the graphic ontology structure and the detail of the execution.

The 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, disks, file system, network interfaces, network parameters, and power sources are recognized. Similarly, in the hardware aspect, the displays, sensors, and devices are detected.

Simultaneously, the second primary component of the Prosci is awaked. Since that moment, File Monitor starts to control all incoming log files and save all newly added files which appear in the workspace. For this purpose, File Monitor uses the 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 Git repository, and you can integrate it into an existing application or create a tool. The further information and description of the JGit are talking in section 4.1.1.

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

As mentioned before, File Monitor is a monitoring tool to record the incoming logs and runs the git commands. It traces the system log directory, and monitor the system working directory, and once the new registration happens to appear in the system log directory, it will check the working directory for saving.

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 user can view all produced files and its related connections. With the help of it, the user can also reload the data, which may have been overridden due to the name convention and the user can also rerun the command from the past command catalog. This part of the functionalities is accomplished with the help of the external toolbox "ProvToolbox". The precise information is given in section 4.3.3.

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. The user of the Prosci must first define the workspace domain. The Prosci supports some of the helper functions to make the user get familiar with the main

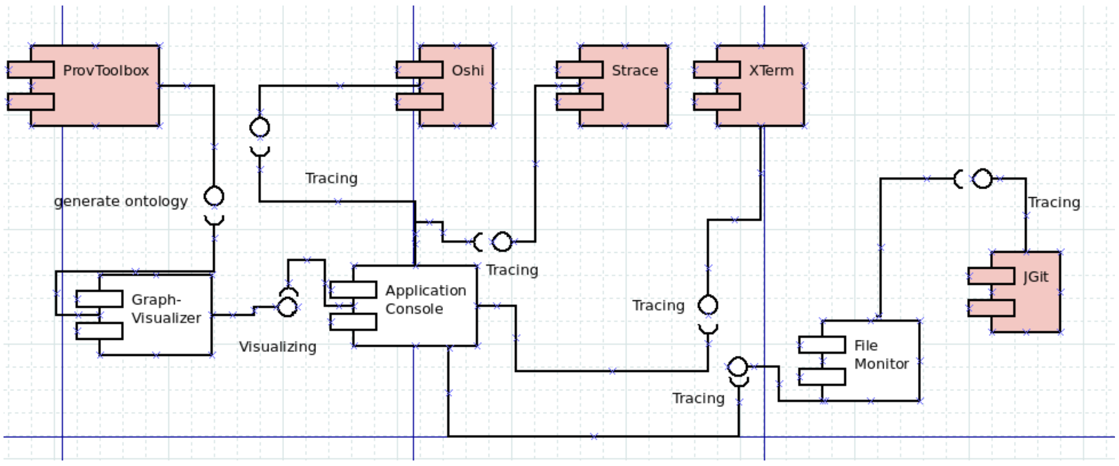
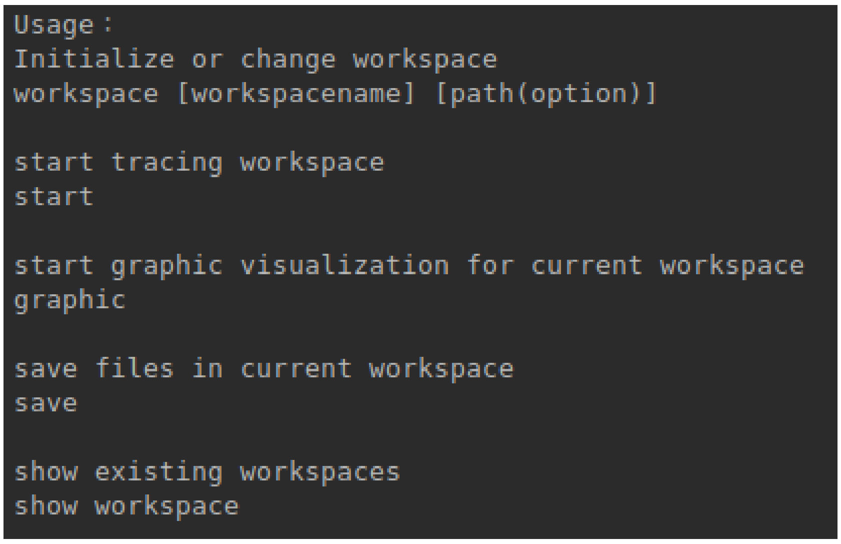


Figure 4.3: Main Components of the Prosci

functionalities of the 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 to choose the corresponding workspace, which they want to work on. The figure 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 the figure 4.4, we know that the Application Console supplies us with five options. Next step I am going to present the internal interaction between the components for each choice and the working sequences of the element: Application Console. So that you can get familiar with the primary constructor of the Prosci application.

Initialize or change workspace domain

Without the doubt, we know that the workspace is the milestone of the Prosci. The first step to initiate the tracing process is to set up the workspace domain. The figure 4.5 demonstrates the basic workflow of the Prosci.

Once the user declares that there a new workspace domain should be created, he needs to enter the workspace name and its location. Or he has a second choice. 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" is working 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 especially, the user must declare the workspace with the name and the supposed location of it. If the given parameters do not consist path parameter, the error message will be thrown.

- If the user wants to create a new workspace domain.
  - If the system property file available?
  - If other workspaces have registered the workspace name?
  - If given workspace path valid?
- If the user wants to define an existing workspace as current workspace.
  - If the system property file available?
  - If given workspace name exists in the system property?

The Figure 4.5 is a sequence diagram of the workspace manipulation. Four participants are involved in this activity. The client enters the instruction, and the Prosci system receives it, it then calls its own class to check the pre-conditions of the creation of a new workspace.

If all the requirements fulfilled, then the workspace is allowed to be formed, after

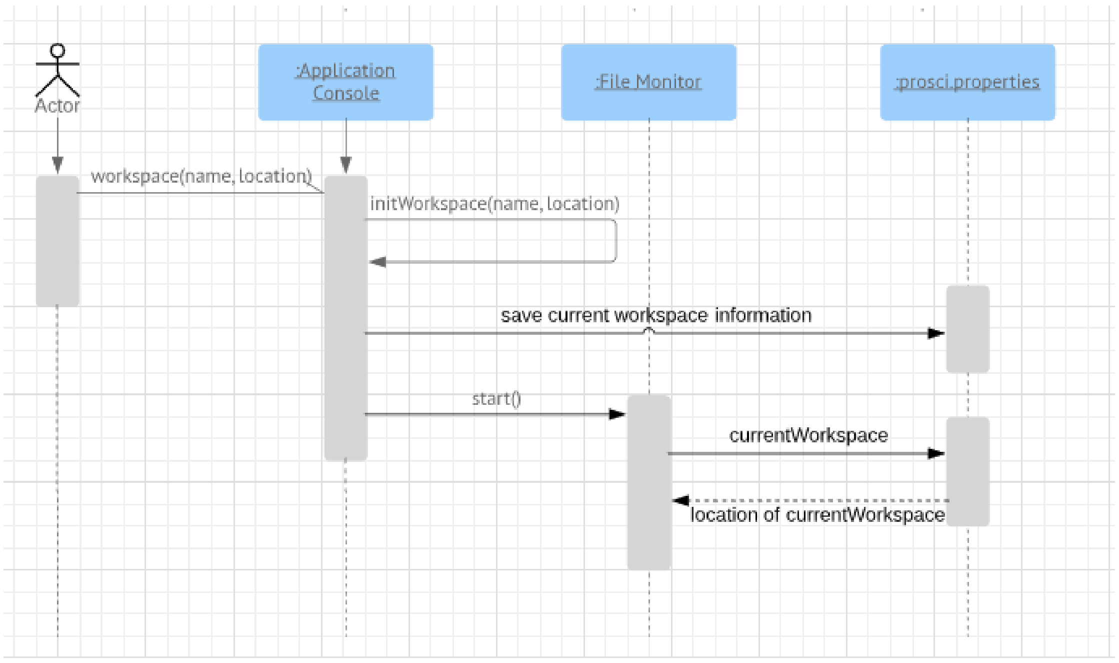

Actor
:Application Console
:File Monitor
:prosci.properties
workspace(name, location)
initWorkspace(name, location)
save current workspace information
start()
currentWorkspace
location of currentWorkspace

Figure 4.5: Initialze or change workspace domain

completion of generation of the newly defined workspace successfully executed, as the next rule, the calling to the second component of the Prosci system will be forwarded.

The component File Monitor later requires to check out the current workspace definition by reading the system property prosci.properties in. The system property as 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.

Figure 4.6 demonstrates us with a detailed overview of the workflow of the workspace manipulation.

The client enters the command, the Application Console first checks the command, if the path parameter consists in it. If yes, the system runs the generation of a new workspace. Otherwise, an old workspace, which must be generated before will be set as the current workspace.

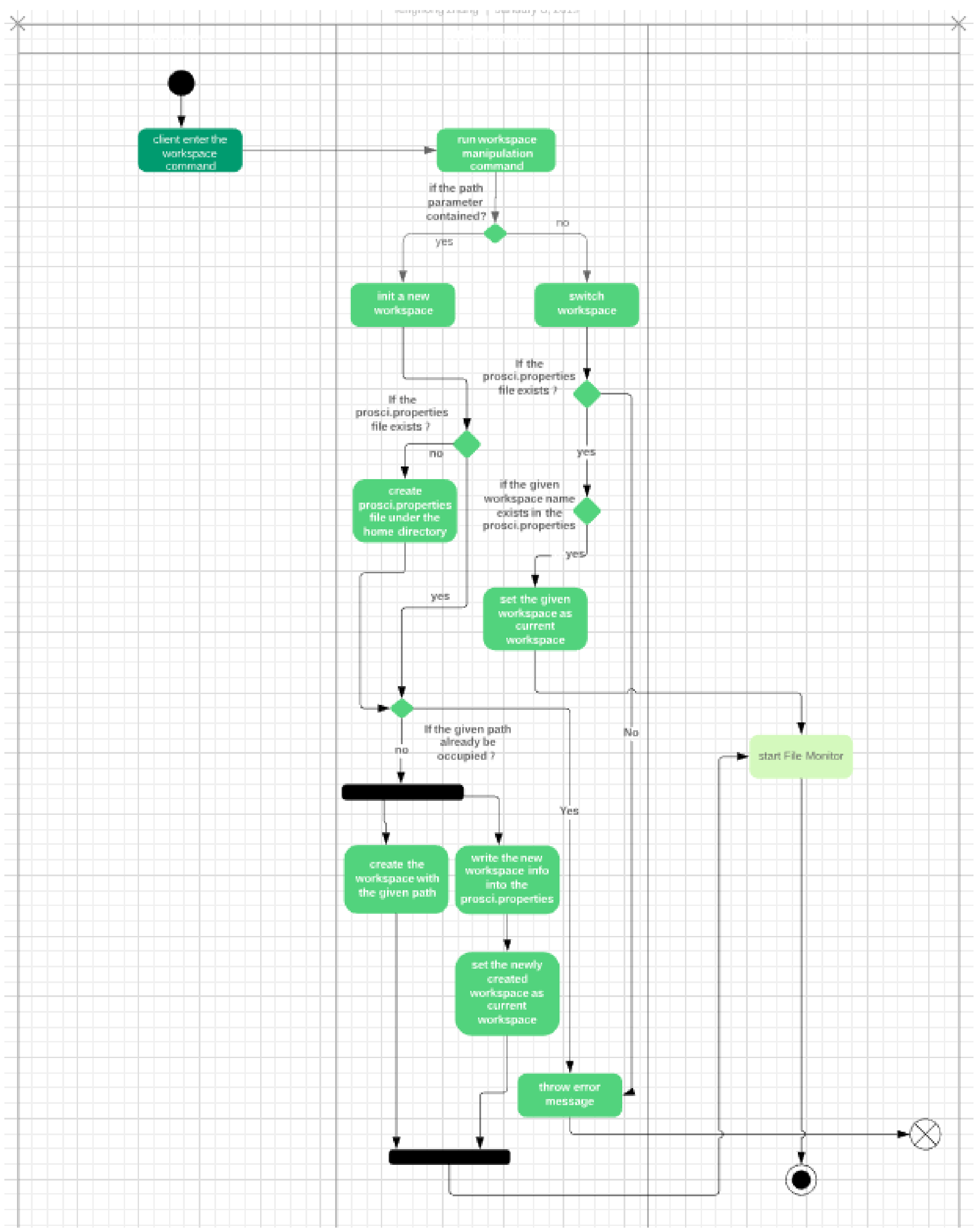

client enter the workspace command
run workspace manipulation command
if the path parameter contained?
yes
no
init a new workspace
switch workspace
If the prosci.properties file exists ?
If the prosci.properties file exists ?
no
yes
create prosci.properties file under the home directory
if the given workspace name exists in the prosci.properties
yes
set the given workspace as current workspace
yes
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

For making a new workspace, we first check if the system property file is already there under the "user.home" directory, for the case, there is no prosci.properties under "user.home", we generate a new one. Soon after the success in finding system property, we are going to verify the validity of the given path of the client. If the directory path is already being used, the error message is thrown. Contrarily, 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 into the existing workspace, the system, in any case, need to check the existence of the system property file. The difference is that, by lacking it, the error is presented. 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 to monitoring any incoming changes under the current workspace domain. The detailed information about the working progress of the File Monitor is discussed in the next section.

Last but not the less is the structure of the generated workspace domain. Figure 4.7 is the structure of the created workspace. The project has the name "myWorkspace". Beneath it, two directories are created. The "input" is the actual working directory, where the files, data, scripts, graphs, etc. are stored in. The user can manually add them into the directory, or they may be generated during the scientific experiment.

In the meantime, the 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 complied provenance ontology structures of the experiment from the Graph Visualizer are saved. They are commonly a 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: is the place where all the tracing results are kept. The tracing results are obtained by using two external tools and one individual component "File Monitor". External tools are Strace, Oshi, and 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. The 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 the access to the various version of the files in the "input" directory. They can be the old version which is overwritten due to the name convention and also can be the latest one, which currently can be found under the "input" directory.

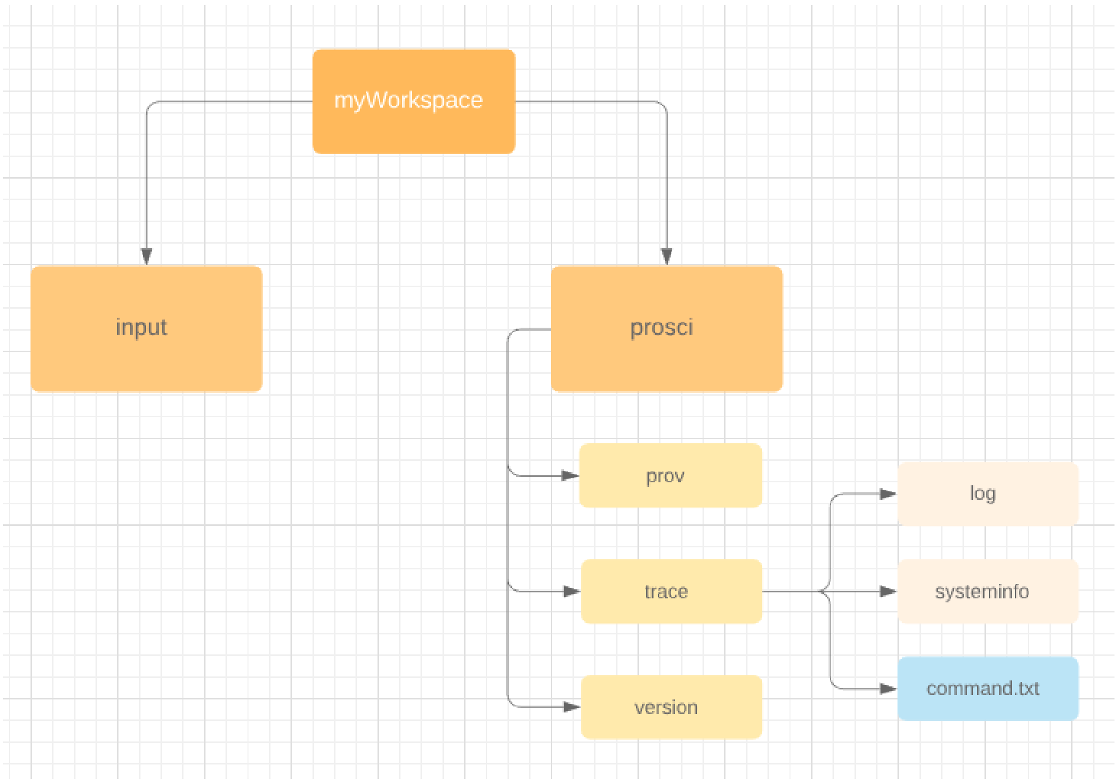


Figure 4.7: The workspace structure of a sample workspace

Start tracing workspace

Of even greater appeal, we now going to talk about the essential function of the Prosci program–start tracing workspace.

Along with the sequence diagram from the figure 4.8 we get a rough overview of the steps for starting the tracing process. Once the application gets the instruction for starting tracing. It first calls its embedded class "start". This class must finish three tasks.

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

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

- 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 CPU load per processor

- Processes
  - number of the processes and threads

- Sensors
  - CPU Temperature
  - Fan Speeds
  - CPU Voltage

- Power Sources
  - Power(charging/ not charging)
  - percentage of the power

- 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

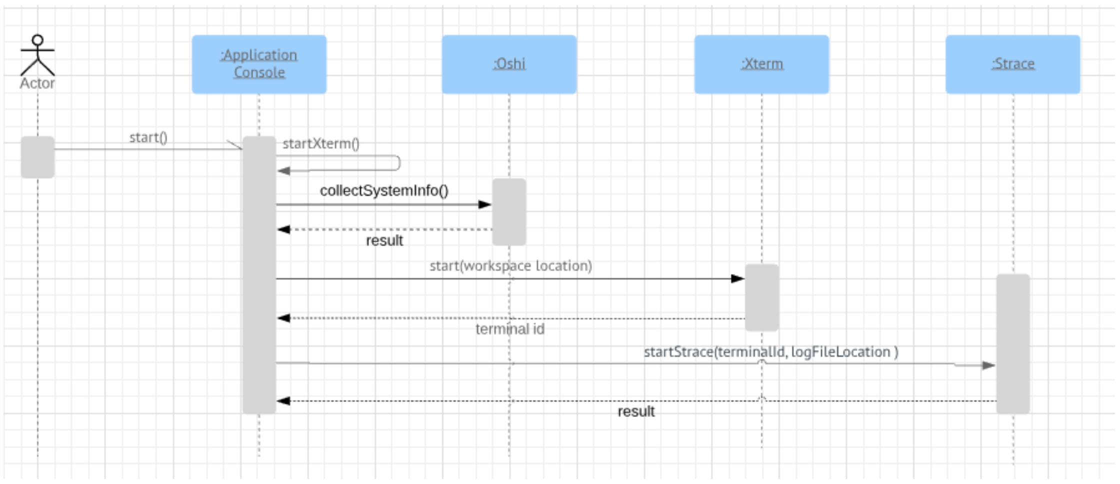


Figure 4.8: Start tracing workspace

This collected system information is saved then under the workspace directory "[workspace name]/prosci/systeminfo". Whenever the application is started, and the user enters the command "start", the Application Console then triggered to collect the information, but during the whole lifetime of the Xterm, those kinds of information will only be collected once.

Next, the task of the program is to call the Xterm tool. Before this action, Prosci calls the system property file first, for reading the current workspace domain. After the workspace domain is verified. The request to start the Xterm is sent. Xterm simulate the real terminal. Inside it, the user can run any kinds of execution just like they are working on a common terminal. This call to the Xterm is return with a message which contains the process id of the Xterm terminal. This returned process id is reserved for further usage. The exact activities are illustrated in the figure 4.9.

Finally, we are ready to start the tracing process using the Strace by giving the tagged process id from previous. Since this moment any tiny activity from the system which runs through the Xterm terminal and makes the effects such as writing, editing, and deleting of the 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 to start Graph Visualizer component of the Prosci is much more simple. The sequence of the workflow to start the component is shown in the figure 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 the new workspace into the system property file "prosci.properties", which located under the user.home directory. The clue for getting all the information about the workspaces is trivial. The figure 4.11 is a sequence diagram of this command. The only thing to do is 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 about is the manual saving of the files using Application Console. The Application Console first read 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 have some new incoming files or some changes do really happen. If the changes are found. The Jgit saves the changes.

Actually, the saving of the new file 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 there are some new execution happen. But the risk of this default saving is that we may lose some of the tracings if after the adding no execution runs.

Having a look into the figure 4.12, we describe the process again with the sequence diagram.

4.2.2 File Monitor

Since we have discussed the Component "Application Console" in the section before, you may already hear about the second Component "File Monitor" several times. This part of the component is the simplest one the whole system.

The file monitor has only one responsibility, it is to monitor the git repositories which are created under the workspace domain. The figure 4.7 illustrates us the fundamental workspace folder structure, we have already mentioned it before for the workspace creation process. Basically, there two git repositories are deployed into the workspace. They are "log repository" and "input repository". The "log repository" traces all incoming new log files which are generated by the Strace tool during the Xterm working period. The .git data can be found under the directory "[workspace name]/prosci/log/.git". The second git repository is much more important, it is "input repository", it controls the workspace working directory "input", it saves all newly added files and records the file changes, also the deleting of the files are reported. This repository is stored under the path "[workspace name]/input". The execution of the git command for this workspace is only triggered when we find some new incoming log files under the "log repository". As we all know, only if the command is running inside the Xterm, the log files for them are going to be generated and saved, in other words, the user must run his execution inside the Xterm. Otherwise, nothing is going to happen. The only one action which is allowed during the provenance process for the scientific experiment is the manual saving of the file into 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 two git repository of the "File Monitor" is illustrated in figure 4.13. This time we are using the flowchart diagram for the illustrating purpose.

As shown in the figure, the "File Monitor" still first need 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 the discovering of the current workspace, the "File Monitor" begins to initialize the two relevant git repository, namely "input git repository" and the "log git repository". After finishing the initialization the tracing thread is starting. It runs in the background and keeps to check if there are some new log files is written within the log git repository. In order to check the "log git repository" the following git command is executed:

logRepository.untracked

If the new entrance is detected. The "File Monitor" will review the "input git repository" with the same git execution command. If new modifies are appearing, the git executions are listed as below:

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

In case of the no entrance in the log repository, the process just running the checking git command for the repository status infinitely, till the end of the total Prosci application.

The implementation of the main functionality of the File Monitor is implemented with the Java thread. Since we all know Java is a multi-threaded programming language. The thread enables java program to share the common resource concurrently for the different parts. The File Monitor is keeking running after the declaration of the workspace domain all the time. It stops, only when the Prosci program is stopped.

4.2.3 Graph Visualizer

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

The Graphic Visualizer at first glance, it illustrates us with a graph of workflows of the certain experiment. Through it, we are able to know the execution steps of the scientific experiment, we obtain the information about the exact command, which are run and the effects of them. Furthermore, the status of the OS is catchable. Equally important is the readable tracing of the file version. In other words, all files, which appear in the workspace is reproducible, even if the file is overwritten due to the naming conversion, is still possible to be reloaded into the workspace directory

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

Principal, this component has five functions.

1. Show workflow of the experiment with the ontology structure in the diagram. The properties of an ontology are files, activities and agents. The properties are represented as the node of the graph, they differentiate from each other through the color.

2. Show all activities, which happened under the workspace domain, more accurately, all executions which were running inside the Xterm windows are called activities and 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 of it is not yet supported. The user must run the system command save in the Application Console to achieve the versioning purpose.
4. Show all agents. Agent in another word is the OS status, during the execution of the scientific experiment. The agent is be determined only once, at the time we start the Xterm terminal.
5. Reload the historical data. The files, which are overwritten and don't exist anymore under the current workspace domain are reloadable.

The implementation of the user interface is based on the JavaFX. The UI component is partly designed under the JavaFX Scene Builder. The style of the UI is configured with parts of the CSS and some are embedded into the java codes.

The Graph Visualizer is written in maven project. Inside the Graph Visualizer, we have one extra package "prov". This module is employed for the reasoning of the ontology analysis.

For this goal, we invite the "prov" module to analyze the saved log files. At the meantime, the command.txt file which stores all information about the commit id is also considered. According to the reported XML file from the ProvToolbox we extract the useful information and demonstrate them into our UI component so that the user can receive a readable view of the hiding ontology.

Prov

Prov is a helper package inside the Graphvisualizer. The main involvement of Prov in the 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 the 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 regardless of the version are ready to be read. OntologyCreator fetches the records in the command.txt with the logs and the 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 an 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 package name of the log file, "xxxxxx.log" is the name of the log. This log is saved under the directory "yyyyyy" and "000000" is the commit id, with this commit id we can find the corresponding generated files caused by this record. Assuming that we generated three files by running the command, they are A.java, b.csv, and c.tex. The OntologyCreator analyzes this record, reads the xxxxxx.log get those three files from git object from correct commit version.

After the successful pairing, the OntologyCreator save the command which it extracts from the log as activity and the file as the entity in the provenance ontology with the XML file format, those arrangements are supported by the ProvToolbox API.

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

Graphvisualizer GUI

Our visualization GUI is built upon the JavaFX. At the beginning of this section, we have already mentioned the JavaFX, a short description of JavaFX can be found in section 3.3. The five main features of the Visualizer GUI is 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 interactable provenance ontology. The user can click on each note in the ontology graph and they will be redirected to the detailed page of the selected point. Fig. 4.16 is the GUI view of the sample project, the detail about our sample project can be discovered in the next chapter. The next two figures show the views, once the user clicks on one of the agent node in the graph and once the user clicks the "Files" button in the dashboard respectively.

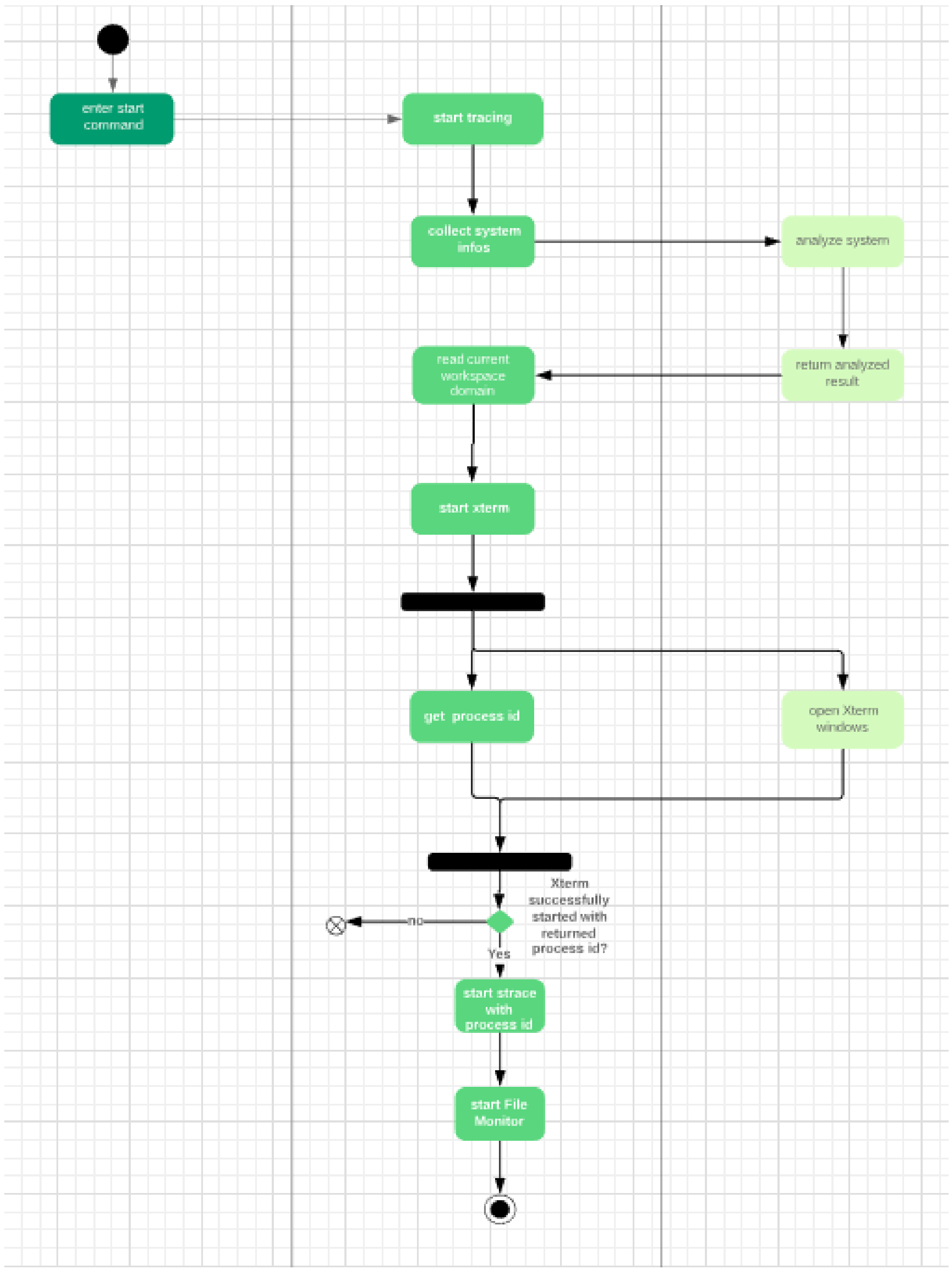


Figure 4.9: Activity Diagram: Start tracing workspace

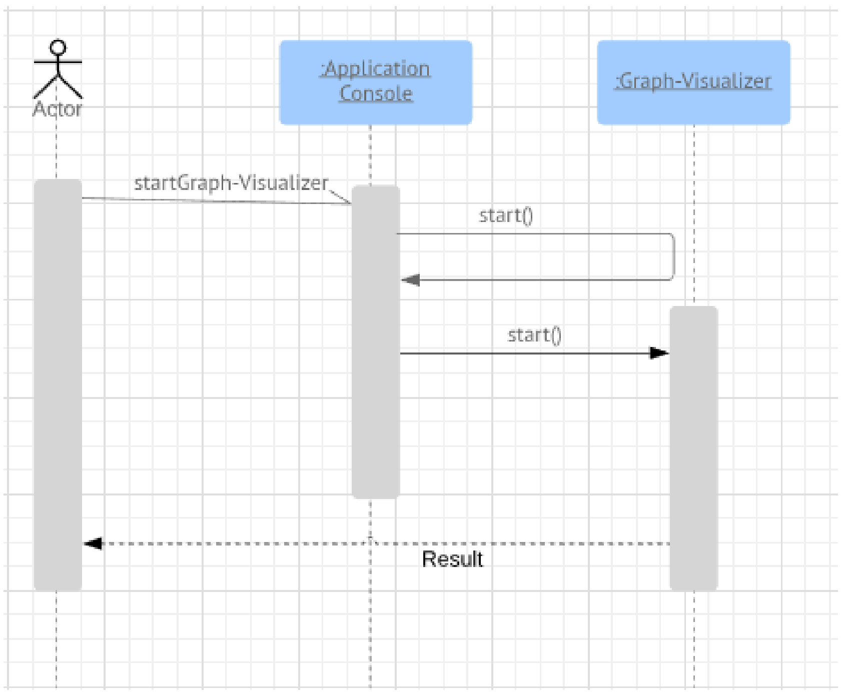


Figure 4.10: Activity Diagram: Start Graph Visualizer

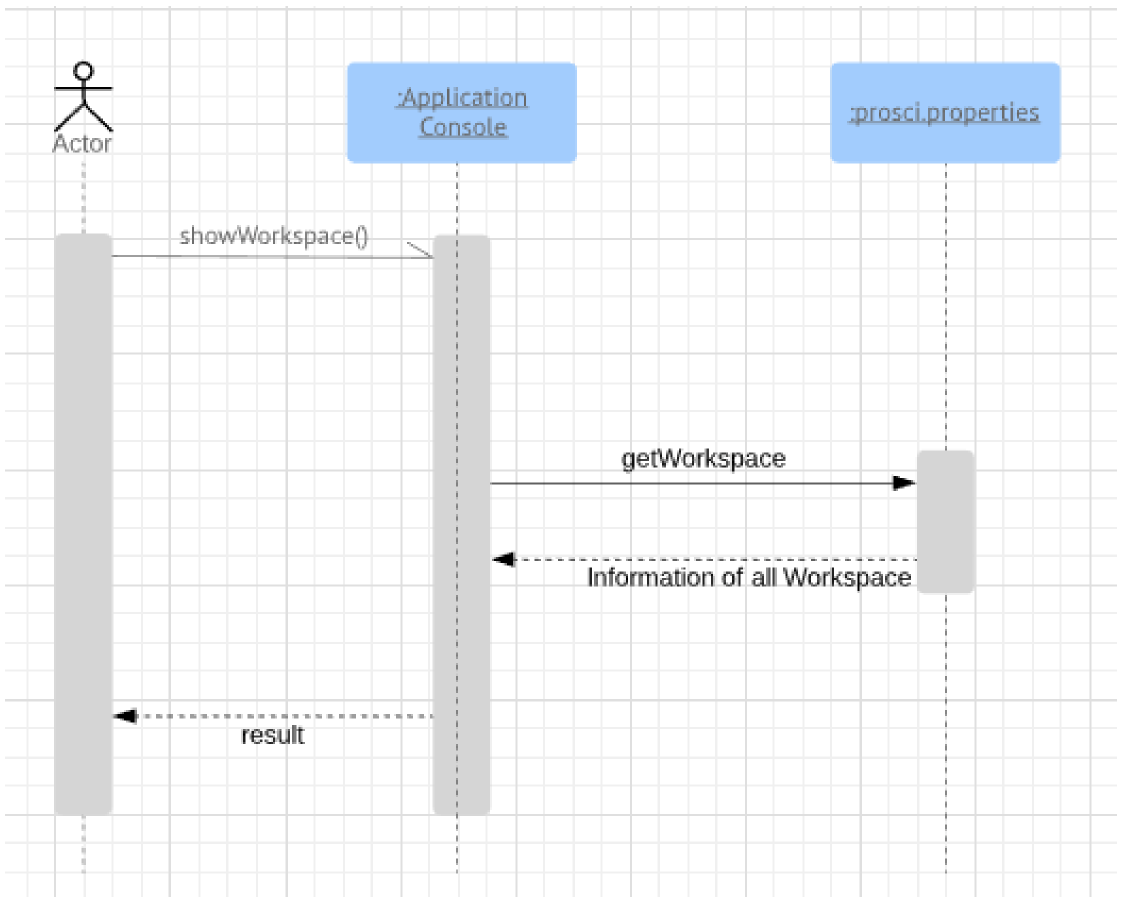


Figure 4.11: Activity Diagram: Show workspaces

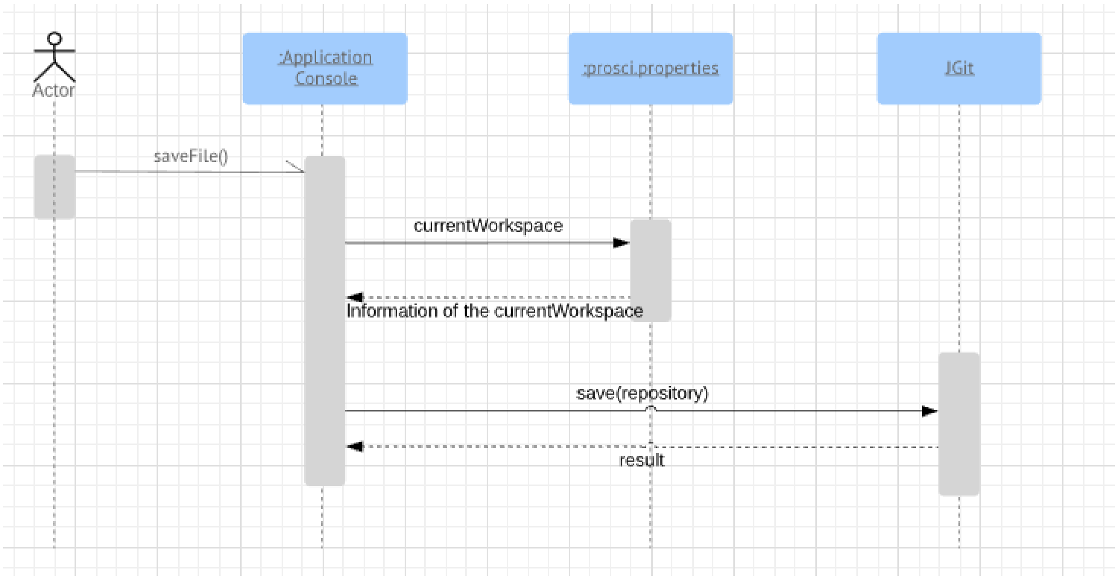


Figure 4.12: Activity Diagram: Save files

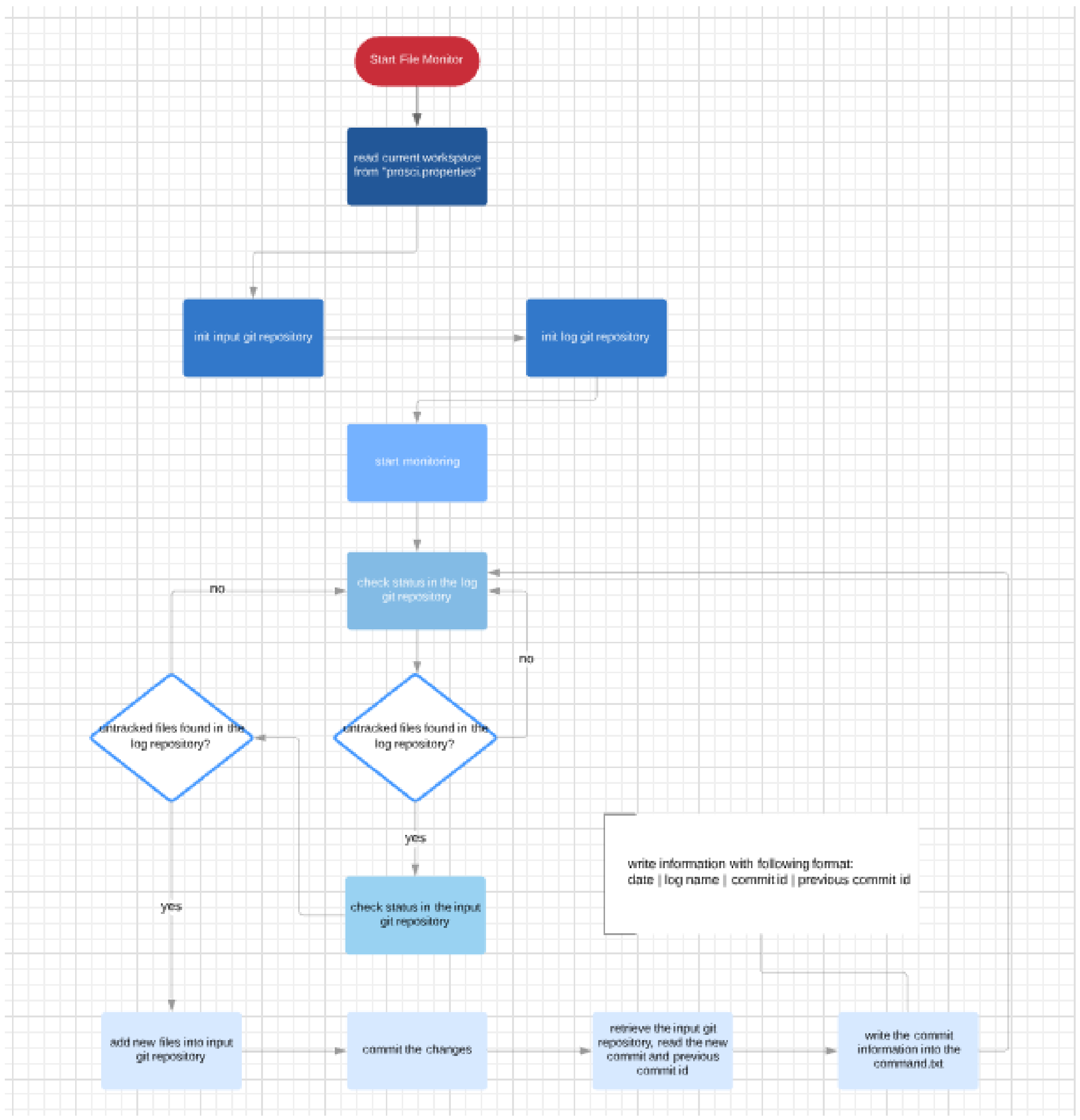

Start File Monitor
read current workspace from "prosci.properties"
init input git repository
init log git repository
start monitoring
check status in the log git repository
no
untracked files found in the log repository?
untracked files found in the log repository?
no
yes
yes
check status in the input 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

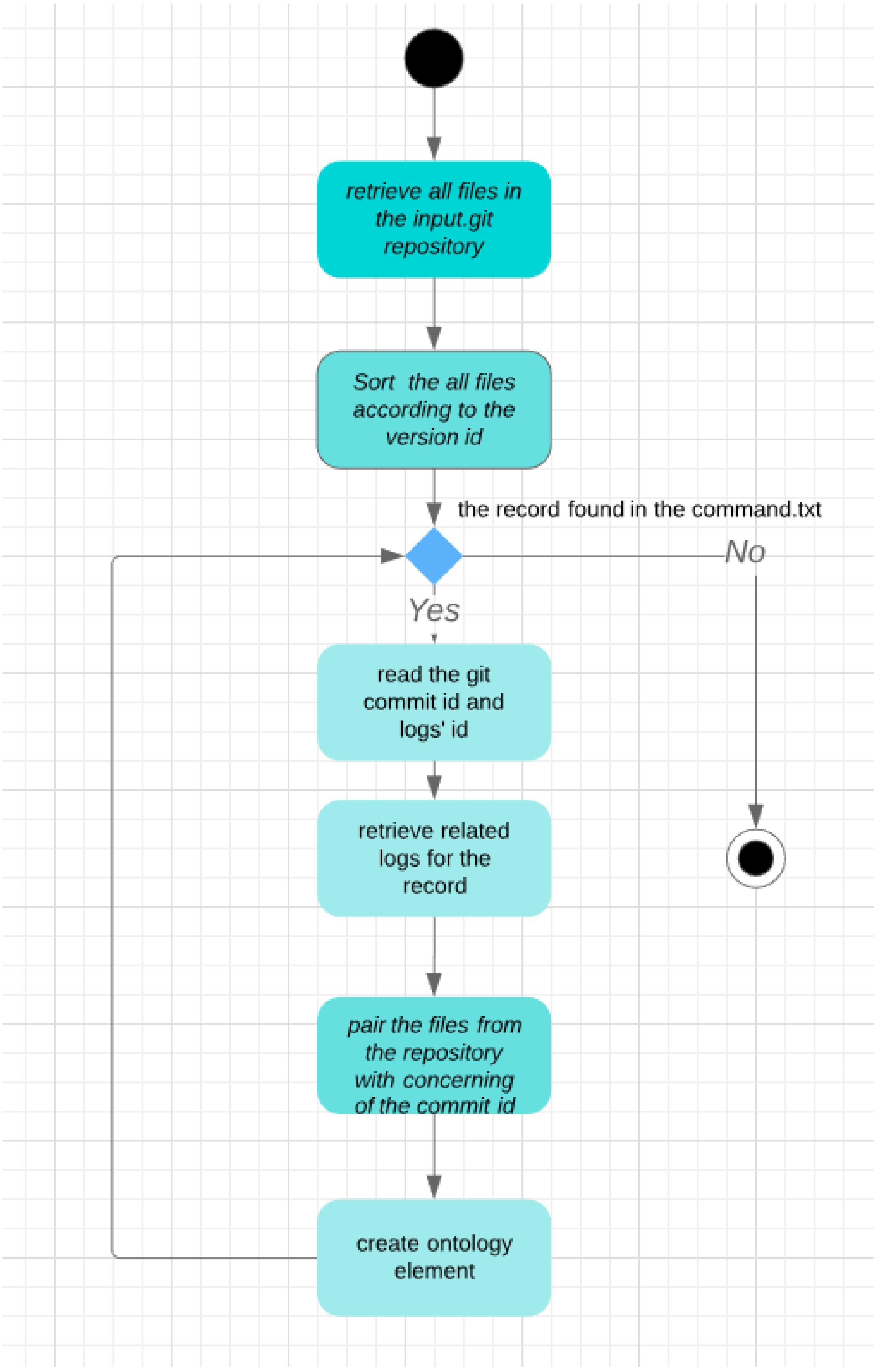

retrieve all files in the input.git repository
Sort the all files according to the version id
the record found in the command.txt
No
Yes
read the git commit id and logs' id
retrieve related logs for the record
pair the files from the repository with concerning of the commit id
create ontology element

Figure 4.14: Workflow of helper package: Prov

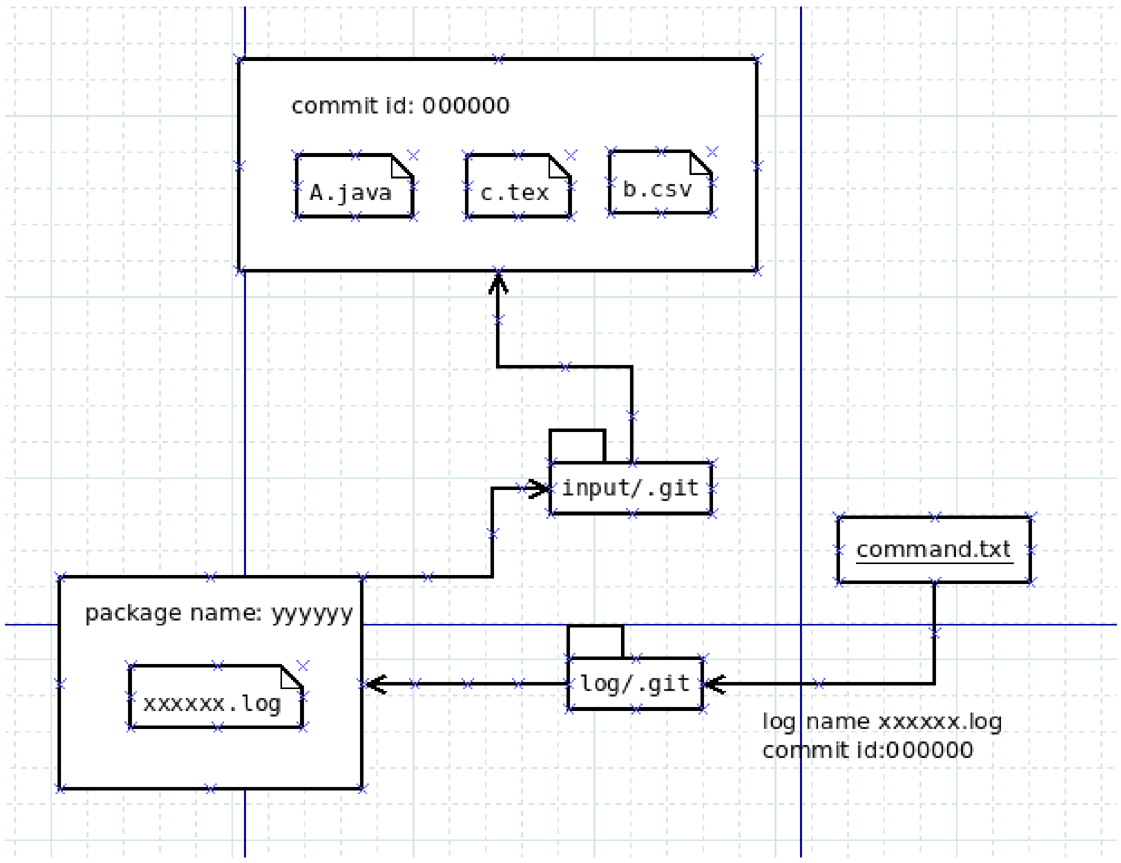


Figure 4.15: An example of the Prov work steps

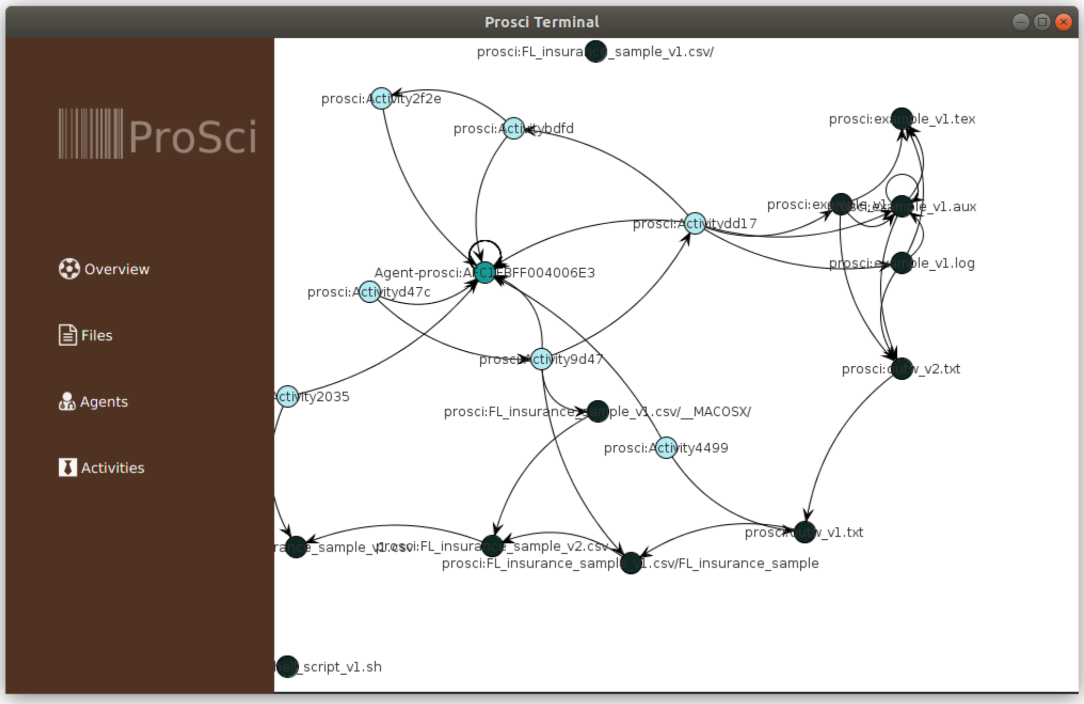


Figure 4.16: GUI view of Sample Project

CHAPTER 5

Sample Project

Until now, all parts of the project was already be shown to you. Last but not the less, 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 with a real solution. This sample project contains 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 in your machine. The details about the preparation are outlined in subsection 5.1. Take a look on it, follow the instructions and enjoy the final results which 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` In the package "ProSci", you can find both source code and the packed .jar file. After the download, you need to unzip the "ProSci.zip" package, open the main directory, you will see three .jar packages, one prosci.sh script and a folder with name "sample".

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

As the next step, we are going to look inside those two files in details.

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 and it is more convenient to write the commands into a shell script so that we can use it repeatedly. The Fig below shows us the basic workflow of the sample project. As we can see from the graph. We first download the necessary data from the link:

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

This dataset contains 36,634 records in Florida for 2012 from a sample company that implemented an aggressive growth plan in 2012. 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 the Weka[HFH$^+$09]. Weka is a sufficient tool for Data mining. Weka is a collection of machine learning algorithms 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. If not, please recovery 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.

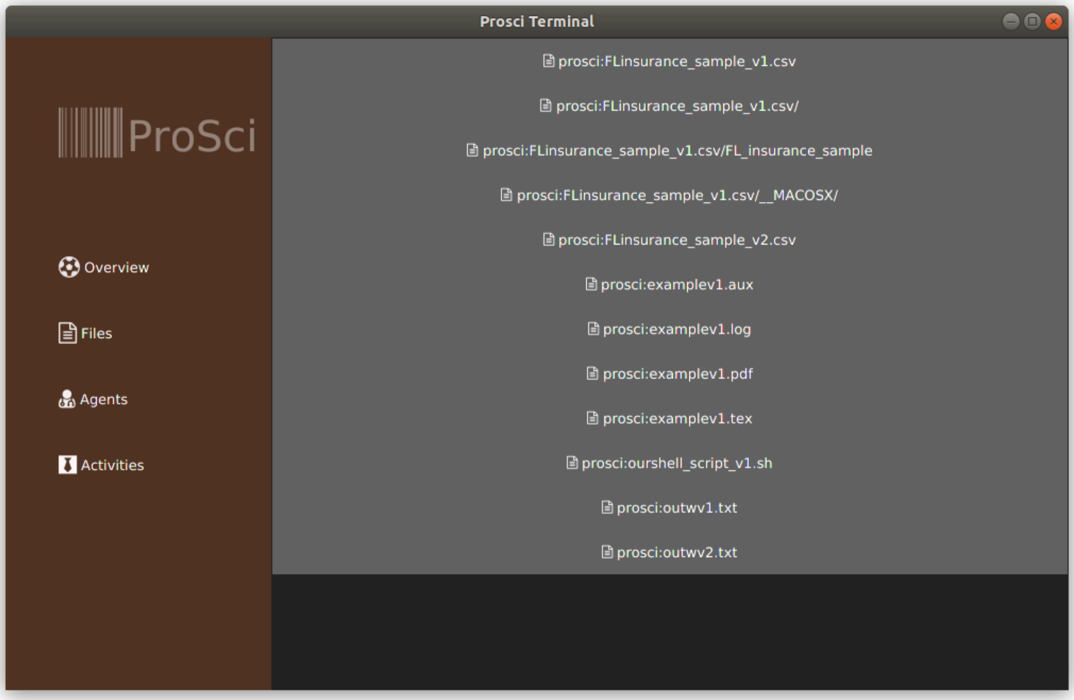
Step 4 intends to using ZeroR classifier. This classifier is used to predict the mean for a numeric class or the mode for a nominal class. In our case, we are going to have the mean as the processing result.

Last step from 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 make this file compiled, you must have Latex in your system. As an alternative step, you can comment out the step 5a. Step 5a is using for download Latex.

5.3 Run sample

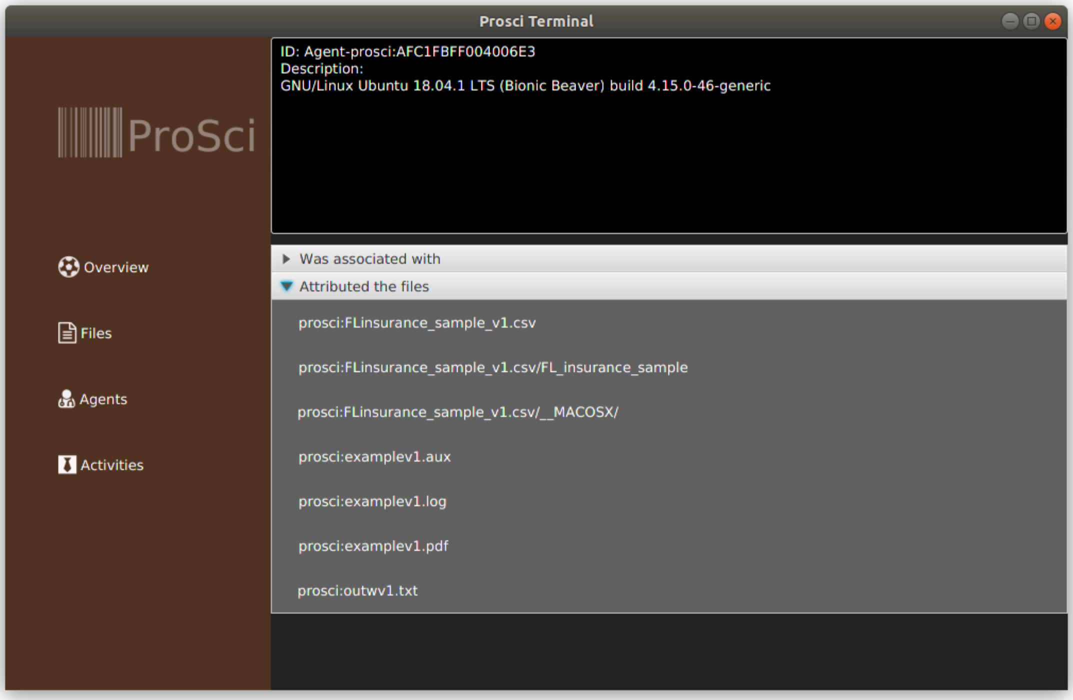
After the description above, you should able to start the ProSci and enjoy the running result from our sample project. Now, go back to the Prosci main directory, the exact workflows are described as below:

- start the ’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 default name prosci. So the workspace creation command is like 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 to run the sample.sh. You can find some new files in your workspace home folder, they are:
  - `FL_insurance_sample.csv.zip`
  - `FL_insurance_sample.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 ’graphic’ 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.
- Totally, you have four options in our graph visualizer. The overview shows you with a ontology graph, it illustrates the connectivity between every activity(command), entity (file) and agent(software agent). You can click on the graph to choose the element, the details of each element in the graph will be displayed as a table. As an alternative you can also search each property on the dashboard on the left of the visualizer’s window. They are sorted into three catalogs, respectively, activities, files and agents.



prosci:FLinsurance_sample_v1.csv
prosci:FLinsurance_sample_v1.csv/
prosci:FLinsurance_sample_v1.csv/FL_insurance_sample
prosci:FLinsurance_sample_v1.csv/__MACOSX/
prosci:FLinsurance_sample_v2.csv
prosci:examplev1.aux
prosci:examplev1.log
prosci:examplev1.pdf
prosci:examplev1.tex
prosci:ourshell_script_v1.sh
prosci:outwv1.txt
prosci:outwv2.txt

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


Prosci Terminal

ID: Agent-prosci:AFC1FBFF004006E3
Description:
GNU/Linux Ubuntu 18.04.1 LTS (Bionic Beaver) build 4.15.0-46-generic

▶ Was associated with
▼ Attributed the files

prosci:FLinsurance_sample_v1.csv
prosci:FLinsurance_sample_v1.csv/FL_insurance_sample
prosci:FLinsurance_sample_v1.csv/__MACOSX/
prosci:examplev1.aux
prosci:examplev1.log
prosci:examplev1.pdf
prosci:outwv1.txt

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

CHAPTER 6

Conclusion

In our ProSci project, we have investigated a new tool, which integrates several different and remarkable 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 the provenance data during the scientific workflow execution, visualizes the provenance metadata with the user-friendly GUI view and allows the user to reload the file, which is overwritten by the name convention.

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

6.1 Limitations

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

- Limitation on the supported system: the Project Prosci is only supported in the Linux system.
- Dependencies on the external tool, such as Xterm and Strace limit the extensibility of the Prosci.
- Better API for graphics visualization is desired. Currently, the API used for visualization is JUNG. Since JUNG aims only to the modeling, analysis, and

visualization of data that can be represented as a graph or network, it doesn't support many advanced features for the provenance ontology visualization. Some domain-specific API or plugin e.g. Ontograf provides better potentiality and capability for illustrating the provenance ontology.

- "command.txt" serves as a dictionary for pairing files and commands. The retrieving method can be improved. So that the seamless program is realized when the user attempts to work on a huge amount to data volume.
- File Monitor component of the ProSci is keeping running in the background all the time.
- No automatical replication of the scientific workflow is embedded.

6.2 Future Work

Reflecting the limitations of the ProSci suggested above, our future work will concentrate on the improvement for both functionality and non-functionality requirements, in order to make the ProSci project more user-friendly and support in more operating systems.

Furthermore, as the next step, we will investigate the topic about 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 the excellent 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] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludscher, and Stephen A. Mock. Kepler: Towards a grid-enabled system for scientific workflows. 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. *Procedia Computer Science*, 4:768 – 777, 2011. Proceedings of the International Conference on Computational Science, ICCS 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.

[Bro14] Jason Brownlee. Reproducible machine learning results by default. 2014.

[Cha14] Scott Chacon. *Pro Git.* Apress, 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*, 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 *BTW*, 2007.

[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 Comput.*, 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.

[Mor03] Luc Moreau. Jung. Technical report, 2003. 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, 10 2016.

[MSSW13] Gina Moraila, A Shankaran, Zuoming Shi, and AM Warren. Measuring reproducibility in computer systems research. 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.

[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. 12(1):461, 2011. Exported from https://app.dimensions.ai on 2019/04/07.

[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.

[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.

[Wik18] Wikipedia contributors. Xterm, 2018.

[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.